Polska Akademia Nauk
Instytut Badań Systemowych

mgr inż. Sławomir Dadas

Iteracyjne metody wykrywania hierarchicznych struktur jednostek nazewniczych

Rozprawa doktorska

Promotor:
prof. dr hab. inż. Witold Pedrycz

Promotor pomocniczy:
dr inż. Jarosław Protasiewicz

Warszawa, 2021

Spis treści

Stosowana notacja matematyczna

Podstawowe symbole

x	skalar
$\boldsymbol{x} = [x_1, x_2, ..., x_n]$	wektor n-elementowy
x_i	element wektora $\boldsymbol{x}$ na pozycji i
$\boldsymbol{X}$	macierz
$X_{i,j}$	element macierzy $\boldsymbol{X}$ na pozycji (i, j)
$A = \{a_1, a_2, ..., a_n\}$	zbiór n-elementowy
$\|A\|$	moc zbioru (liczba elementów w zbiorze)
$\boldsymbol{x} \cdot \boldsymbol{y}$	iloczyn skalarny wektorów $\boldsymbol{x}$ i $\boldsymbol{y}$
$\boldsymbol{x} \odot \boldsymbol{y}$	operator konkatenacji wektorów $\boldsymbol{x}$ i $\boldsymbol{y}$

Funkcje

$f : x \to y$	funkcja, która elementowi x przyporządkowuje element y
$\exp(x)$	funkcja eksponencjalna
$\ln(x)$	logarytm naturalny
$\mathbb{1}_{\{a=b\}}$	funkcja, która przyjmuje wartość 1 jeżeli warunek $a = b$ jest spełniony, 0 w przeciwnym przypadku
$\arg\max_{x \in X} f(x)$	element x ze zbioru X, który maksymalizuje wartość funkcji $f(x)$

W pracy opisane zostały modele uczenia maszynowego wykorzystywane do przetwarzania uporządkowanych sekwencji obserwacji. Wektory reprezentujące tego typu dane sekwencyjne wyróżniono za pomocą następującego zapisu: $\vec{\boldsymbol{x}}$. Odrębna notacja ma na celu uwypuklenie faktu, że kolejność elementów w tych przypadkach jest z góry ustalona i ma znaczenie z punktu widzenia wyniku predykcji. Odróżnia to modele sekwencyjne od tradycyjnych metod uczenia maszynowego, w których próbkę danych możemy opisać pewnym wektorem cech $\boldsymbol{x}$, natomiast kolejność elementów takiego wektora ustalana jest arbitralnie i nie ma wpływu na działanie modelu.

Przedmowa

Praktyczne zastosowania uczenia maszynowego często wiążą się z przetwarzaniem danych sekwencyjnych. W problemach sekwencyjnych mamy do czynienia z listą obserwacji o ustalonej kolejności, na podstawie której model ma za zadanie dokonać predykcji. Powszechnie spotykanymi zadaniami z zakresu przetwarzania sekwencji są między innymi przewidywanie kolejnego elementu na podstawie elementów go poprzedzających, wykrywanie charakterystycznych wzorców w danych (np. w zadaniach związanych z detekcją anomalii), klasyfikacja całej sekwencji lub jej części, czy generowanie zupełnie nowej sekwencji z oryginalnych danych. Przykładami danych, które można przedstawić w postaci uporządkowanego ciągu obserwacji, są szeregi czasowe jako sekwencje danych numerycznych, filmy jako sekwencje obrazów czy dokumenty tekstowe jako sekwencje pojedynczych znaków, grup znaków lub słów.

Niniejsza praca skupia się na wykrywaniu jednostek nazewniczych (ang. *named entity recognition, NER*), czyli zadaniu należącym do dziedziny przetwarzania języka naturalnego, którego głównym założeniem jest identyfikacja i klasyfikacja fraz w tekście odwołujących się różnych kategorii obiektów lub pojęć. Definicja jednostki nazewniczej, jak również lista dopuszczalnych kategorii, jest uzależniona od danych, którymi dysponujemy oraz celu, których chcemy osiągnąć. Przykładowo, dla artykułów prasowych typowymi klasami mogą być imiona i nazwiska osób, nazwy organizacji, nazwy określające obiekty geograficzne i administracyjne, wyrażenia związane z datą i czasem. Natomiast w przypadku artykułów biomedycznych lista klas może uwzględniać na przykład nazwy substancji i związków chemicznych, nazwy wirusów, typy komórek i tkanek, symbole i nazwy genów. Na podstawie powyższych przykładów można zauważyć, że klasy jednostek nazewniczych często związane są z nazwami własnymi, bowiem podobnie jak one służą identyfikacji konkretnych obiektów. Nie jest to jednak reguła - w niektórych instancjach problemu wykrywania jednostek nazewniczych mogą występować klasy uwzględniające również nazwy pospolite, odnoszące się do szerszego zakresu obiektów.

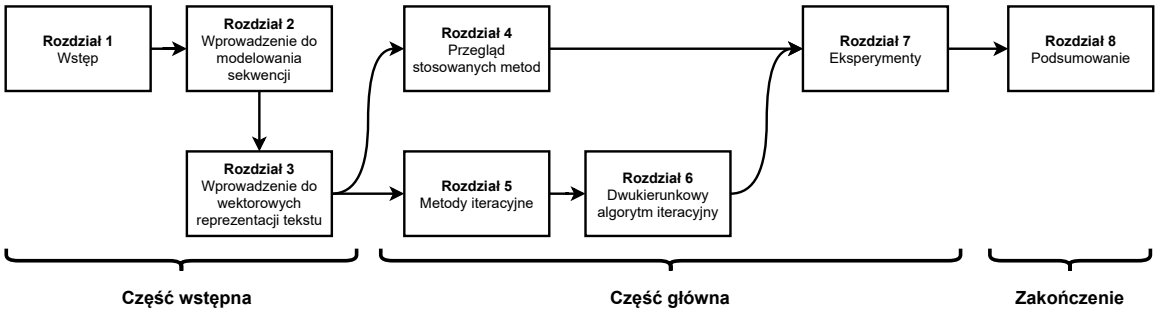
Najczęściej zadanie wykrywania jednostek nazewniczych jest sprowadzane do postaci problemu tagowania sekwencji (ang. *sequence tagging*). Jest to rodzaj uczenia pod nadzorem, w którym model przypisuje klasę każdemu elementowi sekwencji niezależnie - w przeciwieństwie do standardowej klasyfikacji, w której klasyfikowany jest tekst jako całość. Tak zdefiniowany problem ma również zastosowanie przy identyfikacji fraz. W uproszczeniu, kilka następujących po sobie elementów przynależących do tej samej klasy może zostać wyodrębnionych jako samodzielna jednostka nazewnicza. Przyjęcie powyższych założeń pozwala na budowanie modeli, które w sposób sekwencyjny identyfikują frazy w tekście. Do niedawna zainteresowanie naukowców

zajmujących się przetwarzaniem języka naturalnego skupiało się na prostych przykładach wykrywania jednostek nazewniczych. W szczególności przyjmowano założenie, że jednostki nie mogą się na siebie nakładać. Tymczasem niektóre praktyczne zastosowania wymagają wykrywania również jednostek zagnieżdżonych, będących częścią jednostek nadrzędnych. W takim przypadku efektem działania modelu powinna być wielowarstwowa, hierarchiczna struktura jednostek nazewniczych. Liczba warstw w strukturze jest uzależniona od liczby klas i szczegółowości postawionego problemu. W niektórych zastosowaniach praktycznych wymagana jest identyfikacja wielu kategorii jednostek o drobnej granularności, co w znaczący sposób zwiększa prawdopodobieństwo nakładania się jednostek.

Temat hierarchicznej identyfikacji jednostek nazewniczych zyskał na popularności w ostatnich dziesięciu latach. Zaproponowane w literaturze ujęcia początkowo opierały się na wykorzystaniu modeli statystycznych takich jak conditional random fields (CRF). Próbowano też sprowadzać ten problem do zadania generowania hipergrafu reprezentującego zagnieżdżoną strukturę encji. Wraz z rozwojem metod uczenia głębokiego (ang. *deep learning*) zaczęły pojawiać się dedykowane architektury neuronowe, które obecnie stały się dominującą grupą metod stosowanych dla tego typu problemów. Pomimo dynamicznego rozwoju, wiele z zaproponowanych rozwiązań charakteryzuje się wysoką złożonością obliczeniową, uzależnioną od takich parametrów zbioru danych jak liczba zdefiniowanych klas, głębokość zagnieżdżenia czy maksymalna długość jednostki nazewniczej. Ponadto niektóre z modeli narzucają sztywne ograniczenia na powyższe parametry i wymagają ich ustalenia przed rozpoczęciem procesu uczenia modelu. Współczesne rozwiązania bazujące na sieciach neuronowych często są złożonymi systemami, wymagającymi wytrenowania kilku wyspecjalizowanych modeli.

Przedmiotem niniejszej pracy jest zaproponowanie nowego iteracyjnego ujęcia problemu wykrywania hierarchicznych struktur jednostek nazewniczych. W ujęciu tym zakładamy, że problem predykcji struktury hierarchicznej można rozwiązać w sposób iteracyjny, zaczynając od pustego zbioru predykcji i rozszerzając go z każdą kolejną iteracją do momentu, gdy nie zostaną wykryte żadne nowe jednostki nazewnicze. Zaletą zaproponowanej metody jest możliwość łatwego dostosowania istniejących modeli stosowanych w problemach niezagnieżdżonych do działania w sposób iteracyjny, a tym samym rozwiązywania za ich pomocą bardziej złożonych problemów hierarchicznej predykcji. Ponadto przedstawiona metoda nie narzuca żadnych ograniczeń na głębokość zagnieżdżeń czy maksymalną długość jednostki nazewniczej - model iteracyjny automatycznie dostosowuje się do charakterystyki zbioru treningowego w procesie uczenia.

Przedstawione w pracy ujęcie iteracyjne stanowi oryginalny wkład autora w problem wykrywania zagnieżdżonych struktur jednostek nazewniczych. W ramach badań opisanych w tej pracy opracowane zostały również nowe metody wykrywania jednostek będące praktyczną implementacją tego ujęcia. Podstawowym z zaproponowanych rozwiązań jest neuronowy model iteracyjny, którego główna idea polega na przekazywaniu zakodowanego wektora stanu pomiędzy poszczególnymi iteracjami, co pozwala na wykorzystanie informacji o relacjach hierarchicznych pomiędzy jednostkami przy dokonywaniu predykcji. W dalszej części pracy przedstawiono bardziej zaawansowane metody iteracyjne oparte na wykorzystaniu dwóch modeli trenowanych w przeciwnych kierunkach. W ramach predykcji dwukierunkowej rozpatrywane są różne sposoby łączenia wyników modeli (tzw. funkcje selekcji) oraz propagacji wektorów stanu pomiędzy



Rysunek 1: Struktura pracy.

modelami.

Niniejsza praca podzielona została na osiem rozdziałów. Struktura pracy została przedstawiona w postaci schematu graficznego na Rysunku 1.

Pierwszy rozdział stanowi wprowadzenie do zagadnienia wykrywania jednostek nazewniczych. Pokazane zostaną przykładowe praktyczne zastosowania modeli identyfikacji jednostek. Przedstawiona zostanie również formalna definicja problemu wykrywania jednostek nazewniczych. Zdefiniowane zostaną cele oraz hipotezy badawcze będące przedmiotem niniejszej rozprawy.

Kolejne dwa rozdziały zawierają niezbędne podstawy teoretyczne wprowadzające czytelnika w problem wykrywania jednostek nazewniczych. Część wprowadzająca została podzielona na dwie grupy zagadnień. W rozdziale drugim opisane zostaną metody wykorzystywane przy modelowaniu danych sekwencyjnych, w tym modele graficzne (ang. *graphical models*) oraz architektury sieci neuronowych dedykowanych do przetwarzania sekwencji. Rozdział trzeci natomiast dotyczy tematyki związanej z reprezentowaniem tekstu w modelach uczenia maszynowego. Omówione zostaną między innymi wektorowe reprezentacje tekstu, autoregresyjne i maskowane modele języka oraz metody podziału tekstu na sekwencje.

Rozdział czwarty przestawia aktualny stan wiedzy w kontekście problemu hierarchicznego wykrywania jednostek nazewniczych. Uwzględnia przegląd i krytyczną analizę stosowanych do tej pory metod. Wprowadzona zostanie taksonomia ujęć, wyróżniająca trzy główne grupy: metody statystyczne, metody oparte na hipergrafach oraz metody oparte na sieciach neuronowych.

W rozdziale piątym przedstawiono nowe iteracyjne ujęcie problemu hierarchicznego wykrywania jednostek nazewniczych będące zasadniczym przedmiotem tej rozprawy. Zaproponowana zostanie konkretna neuronowa architektura iteracyjna jako praktyczna implementacja tego ujęcia.

W rozdziale szóstym wprowadzono dwukierunkowy algorytm iteracyjny i omówiono poszczególne jego elementy, w tym funkcje selekcji oraz sposób integracji modeli iteracyjnych w ramach algorytmu.

Rozdział siódmy zawiera omówienie i krytyczną analizę wyników doświadczeń przeprowadzonych przy użyciu zaproponowanej metody. Wyniki zostaną porównane z wynikami innych metod omówionych w rozdziale czwartym. W rozdziale zidentyfikowane zostaną również kluczowe cechy algorytmu wpływające na jakość predykcji (*ablation study*). Porównane zostaną warianty zaproponowanego rozwiązania różniące się funk-

cją selekcji oraz wybranymi hiperparametrami.

Rozdział ósmy stanowi podsumowanie zasadniczej części rozprawy. Zawiera dyskusję na temat osiągniętych rezultatów, wkładu pracy w rozwój dziedziny oraz praktycznych możliwości zastosowania przedstawionych w pracy rozwiązań.

Rozdział 1

Wstęp

Zagadnienie wykrywania jednostek nazewniczych w kontekście automatycznych metod przetwarzania języka naturalnego pojawiło się kręgu zainteresowania naukowców w latach 90-tych ubiegłego wieku [108, 38] i od początku związane było z praktycznym zastosowaniem metod ekstrakcji wiedzy. Zauważono, że identyfikacja nazw określających firmy, osoby, lokalizacje, wyrażenia odnoszące się do daty lub czasu, może stanowić istotny etap w procesie rozumienia tekstu [95]. Pozwala ona bowiem na wzbogacenie nieustrukturalizowanych dokumentów o dodatkową ustrukturalizowaną wiedzę, która może być wykorzystana między innymi do efektywniejszego ich przeszukiwania, znajdowania relacji pomiędzy encjami, klasyfikacji dokumentów lub samych encji. Niniejsza praca używa słowa *encja* w odniesieniu do rzeczywistego obiektu lub pojęcia, natomiast *jednostka nazewnicza* w odniesieniu do nazwy - potencjalnie jednej z wielu możliwych - odwołującej się do tego obiektu.

Proces wykrywania jednostek nazewniczych jest często jednym ze wstępnych etapów w złożonych systemach przetwarzania tekstu, a jego wyniki są wykorzystywane przez kolejne modele realizujące bardziej wymagające zadania. W związku z tym identyfikacja jednostek nazewniczych znajduje liczne zastosowania, do których należą:

1. **Wspomaganie systemów wyszukiwania i rekomendacji treści** [101, 40] - Rosnąca liczba dokumentów w formie cyfrowej stawia wyzwania przed instytucjami, które zajmują się tworzeniem i przechowywaniem tego typu treści. Dotyczy to przede wszystkim portali informacyjnych publikujących artykuły prasowe ale także właścicieli innych baz takich jak platformy blogowe czy bazy publikacji naukowych. Identyfikacja encji w tekstach umożliwia późniejsze wykorzystanie ich w celach indeksowania dokumentów. Dzięki temu możliwe jest automatyczne nadawanie tagów artykułom, grupowanie ich według encji czy porównywanie ze sobą dokumentów na ich podstawie. W oparciu o wyodrębnioną wiedzę, platformy mogą budować skuteczniejsze algorytmy wyszukiwania lub rekomendowania treści użytkownikom.

2. **Systemy odpowiadające na pytania** [92, 111] (ang. *question answering*) - Identyfikacja encji jest również kluczowym elementem w systemach dialogowych, których celem jest odpowiadanie na pytania użytkowników. Systemy takie często korzystają z zewnętrznych baz wiedzy stanowiących źródło faktów, na podstawie których konstruowana jest odpowiedź. Jeżeli pytanie dotyczy konkretnej encji, system musi wykryć jednostki nazewnicze wskazujące na nią w treści pytania, wyszukać odwołania do niej

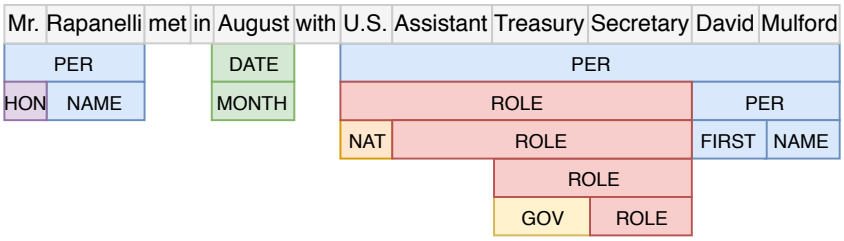
w bazie wiedzy, przetworzyć znalezione w bazie treści i zwrócić odpowiedź użytkownikowi. Tego typu systemy dialogowe coraz częściej wykorzystywane są również komercyjnie jako automatyczne wsparcie klienta. W takim przypadku, gdy klient zgłasza problem dotyczący produktu lub usługi, system może udzielić automatycznej odpowiedzi po znalezieniu jej w wewnętrznej bazie wiedzy lub przekierować użytkownika do odpowiedniego konsultanta, jeżeli automatyczna odpowiedź nie może zostać wygenerowana.

3. **Analiza sentymentu w badaniach rynku** [142, 68, 146, 42] - Media społecznościowe stały się dla firm ważnym źródłem opinii o ich działalności i produktach. W odpowiedzi na to zapotrzebowanie pojawiły się serwisy wykorzystujące metody przetwarzania języka naturalnego w celu automatycznego gromadzenia oraz analizy treści pojawiających się w Internecie. Usługi te pozwalają między innymi na badanie nacechowania emocjonalnego użytkowników w stosunku do firmy oraz zmiany sentymentu klientów w czasie. Niezbędna do tego celu jest identyfikacja komentarzy dotyczących konkretnych firm oraz ich produktów, co może być osiągnięte dzięki metodom wykrywania jednostek nazewniczych.

4. **Automatyczna konstrukcja baz wiedzy** [28, 27] (ang. *knowledge base population*) - Zadanie to polega na uzupełnianiu bazy wiedzy na podstawie korpusu dokumentów tekstowych. Jeżeli informacje znajdujące się w dokumencie dotyczą encji już występującej w bazie, nazwa tej encji powinna zostać wykryta i automatycznie powiązana z istniejącą encją. W ramach procesu mogą być generowane również nowe pozycje, w przypadku gdy w dokumentach wykryte zostaną wystąpienia encji, która do tej pory nie była uwzględniona w bazie.

5. **Anonimizacja danych** [34, 61, 45] - Identyfikacja danych wrażliwych w dokumentach tekstowych oraz ich automatyczna anonimizacja jest kolejnym zadaniem, w którym wykorzystywane są metody wykrywania jednostek nazewniczych. Typowymi przykładami danych podlegających ochronie są dane osobowe, adresy oraz dane kontaktowe takie jak numer telefonu czy adres e-mail. Część fraz odpowiadających tym kategoriom danych można zidentyfikować metodami heurystycznymi, przy wykorzystaniu zestawu reguł lub wyrażeń regularnych. Niektóre dane wrażliwe wymagają jednak bardziej zaawansowanych rozwiązań uwzględniających kontekst wystąpienia jednostki nazewniczej. W takich przypadkach modele uczenia maszynowego uzyskują wyższą jakość identyfikacji od systemów regułowych.

Powyższe przykłady pokazują, że proces wykrywania jednostek nazewniczych na ogół nie występuje samodzielnie, natomiast stanowi niezbędny element systemów uczenia maszynowego, w których predykcja odbywa się na podstawie określonych kategorii fraz. Prawidłowa identyfikacja jednostek nazewniczych często wymaga nie tylko zlokalizowania danego słowa czy frazy, ale też rozumienia kontekstu, w którym te słowa wystąpiły. Przykładowo, słowo *Polska* w zależności od kontekstu może należeć do różnych kategorii jednostek. Może wystąpić jako nazwa kraju, nazwa ulicy, nazwisko, nazwa czasopisma lub utworu, czy część nazwy organizacji (np. *Polska Akcja Humanitarna*). W związku z tym tradycyjne metody wykrywania jednostek, oparte na manualnie stworzonych regułach lub wstępnie przygotowanych słownikach, mogą być



Rysunek 1.1: Przykład zdania z oznaczoną zagnieżdżoną strukturą jednostek nazewniczych. Zdanie pochodzi ze zbioru NNE [112].

używane tylko w najprostszych przypadkach. Osiągnięcie wyższej poprawności identyfikacji i klasyfikacji jednostek wymaga skorzystania z metod opartych na uczeniu maszynowym.

Pomimo dynamicznego rozwoju metod uczenia maszynowego, problem wykrywania jednostek nazewniczych wciąż jest daleki od pełnego rozwiązania. W związku z tym większość badań dotyczących tego zagadnienia nadal skupia się na jego uproszczonej definicji, w której zakłada się, że pojedyncze słowo może wchodzić w skład co najwyżej jednej jednostki nazewniczej. Natomiast w rzeczywistych zastosowaniach pojawiają się bardziej skomplikowane przypadki, wymagające opracowania dedykowanych rozwiązań. Znane z literatury problemy zwiększające trudność zdania wykrywania jednostek to między innymi:

• **Zagnieżdżone jednostki nazewnicze** [21] (*nested named entity recognition*) - W tej wersji problemu jednostki nazewnicze mogą się na siebie nakładać i tworzyć wielowarstwowe struktury. Na Rysunku 1.1 zaprezentowano przykładową graficzną reprezentację zdania z oznaczoną zagnieżdżoną strukturą jednostek. Nakładając na siebie wykryte frazy wraz z przypisaną im klasą, rozpoczynając od najdłuższych fraz a kończąc na najkrótszych, możemy utworzyć strukturę złożoną z warstw, w której każda warstwa zawiera odrębny zbiór jednostek.

• **Brak ciągłości w jednostkach nazewniczych** [93, 21, 22] (*discontinuous entities*) - Słowa wchodzące w skład jednostki nazewniczej nie muszą ze sobą sąsiadować, mogą być oddzielone innymi słowami nie będącymi częścią jednostki.

• **Duża liczba klas** [72, 77, 82, 73] (*fine-grained named entity recognition*) - W tego typu problemach liczba możliwych klas jednostek może wynosić od kilkuset do kilku tysięcy.

• **Wykrywanie encji w dynamicznie zmieniających się danych** [23] (*emerging entities*) - Przykłady dotyczą przede wszystkim zbiorów danych pochodzących z mediów społecznościowych, gdzie cały czas pojawiają się nowe tematy dyskusji i nowe jednostki nazewnicze różniące się od tych, na których model był trenowany. W tego typu problemach wymagana jest większa generalizacja, nawet kosztem jakości modelu dla znanych przykładów.

Niniejsza praca skupia się na zagadnieniu wykrywania zagnieżdżonych jednostek nazewniczych. Spośród wymienionych powyżej wariantów problemu, obecność jednostek

zagnieżdżonych jest jednym z najczęściej występujących w praktycznych zastosowaniach. Problem nakładających się jednostek pojawiał się już kilkanaście lat temu w przypadku takich zbiorów danych jak GENIA [96], ACE2004 [25] i ACE2005. Początkowo ignorowano go lub próbowano rozwiązywać prostymi metodami opartymi na ręcznie zaprogramowanych regułach lub heurystykach.

Znaczny wzrost zainteresowania tym problemem mogliśmy zaobserwować dopiero w ostatnim dziesięcioleciu. Ostatnie lata przyniosły wyraźny postęp w zakresie predykcji zagnieżdżonych struktur jednostek, który był możliwy przede wszystkim dzięki rozwojowi metod przetwarzania języka naturalnego w kontekście sieci neuronowych, w szczególności współczesnych metod wektorowej reprezentacji tekstu oraz neuronowych modeli języka trenowanych na dużych korpusach tekstowych. Wraz ze wzrostem jakości predykcji modeli rośnie jednak również ich złożoność, zarówno w znaczeniu złożoności obliczeniowej jak i w znaczeniu stopnia skomplikowania metod. Zwiększa się liczba hiperparametrów, które sterują procesem trenowania i predykcji modeli. Zwiększa się również liczba komponentów, z których złożone są proponowane w literaturze rozwiązania. Powyższe fakty sprawiają, że coraz trudniejsza staje się inżynieria wsteczna mająca na celu zrozumienie działania modelu oraz identyfikację elementów mających wpływ na jakość predykcji.

W pracy przedstawione zostanie nowe iteracyjne ujęcie problemu wykrywania hierarchicznych struktur jednostek nazewniczych. Ujęcie to zakłada, że wielowarstwową strukturę jednostek nazewniczych możemy skonstruować w sposób iteracyjny, dzieląc ją na pojedyncze warstwy i dokonując predykcji warstwa po warstwie. Jednocześnie suma wyników dotychczasowych iteracji przechowywana jest w postaci zbioru jednostek nazewniczych, którego zawartość jest kodowana i przekazywana jako dane wejściowe do każdej iteracji modelu. Działanie modelu jest przerywane w momencie, gdy w wyniku danej iteracji zbiór nie zostanie rozszerzony o żadne nowe jednostki nazewnicze. Na potrzeby badań empirycznych zaproponowana zostanie konkretna architektura neuronowa dostosowana do uruchamiania w sposób iteracyjny. Założenia ujęcia iteracyjnego są jednak na tyle ogólne, że możliwe byłoby zastosowanie również innych modeli znanych z literatury. Uproszczony problem identyfikacji jednostek nazewniczych bez zagnieżdżeń, na którym wciąż skupia się większość badań, w ujęciu iteracyjnym nie jest traktowany jako odrębne zagadnienie, ale jako przypadek szczególny problemu z pojedynczą warstwą. Modele działające na płaskiej strukturze jednostek w większości mogłyby być dostosowane do działania w sposób iteracyjny za pomocą metody, która zostanie zaproponowana w dalszej części pracy.

Ponadto przedstawiony zostanie również dwukierunkowy algorytm iteracyjny, łączący predykcje dwóch niezależnie trenowanych modeli: modelu *inside-out* oraz *outside-in*. Zasada działania algorytmu jest następująca. Każdy z modeli generuje odrębny zbiór jednostek nazewniczych. Zbiory te następnie są łączone przy użyciu funkcji selekcji, której zadaniem jest wybór finalnego zbioru predykcji na podstawie predykcji każdego z modeli. W zależności od rodzaju wybranej funkcji selekcji, możemy uzyskać różną charakterystykę wyników. W pracy zaproponowanych i przetestowanych zostanie sześć różnych funkcji selekcji: inside-out, outside-in, suma zbiorów, część wspólna zbiorów, selekcja oparta na pewności oraz selekcja oparta na klasyfikatorze.

1.1. Cele pracy

Głównymi celami niniejszej pracy są:

• Zaproponowanie nowego iteracyjnego ujęcia wykrywania hierarchicznych struktur jednostek nazewniczych. Wykazanie, że problem predykcji wielowarstwowej struktury jednostek można rozwiązać za pomocą pojedynczej sieci neuronowej wywoływanej w sposób iteracyjny, jeżeli wyniki poprzednich iteracji zakodujemy w danych wejściowych modelu.

• Przedstawienie przykładowej architektury sieci neuronowej pozwalającej na generowanie wyników w sposób iteracyjny. Sieć zostanie zbudowana w oparciu o współczesne wektorowe reprezentacje tekstu pochodzące z neuronowych modeli języka, warstwy rekurencyjne typu LSTM (*long short-term memory*) oraz warstwę predykcyjną CRF (*conditional random fields*).

• Zaproponowanie dwukierunkowego algorytmu iteracyjnego, łączącego wyniki modeli iteracyjnych trenowanych w przeciwnych kierunkach: *inside-out* oraz *outside-in*.

• Wykazanie, że iteracyjne metody charakteryzują się poprawnością wyników porównywalną lub wyższą od stosowanych do tej pory rozwiązań dla szerokiego zakresu zadań związanych z wykrywaniem zagnieżdżonych struktur jednostek nazewniczych. Metody zaproponowane w tej pracy zostaną przetestowane na powszechnie używanych zbiorach danych dla tego problemu, różniących się od siebie m.in. językiem, domeną czy poziomem złożoności zagnieżdżonych struktur.

1.2. Oryginalne aspekty pracy

Badania przedstawione w pracy zawierają oryginalny wkład autora w problem badawczy wykrywania zagnieżdżonych struktur jednostek nazewniczych. Podstawowymi aspektami oryginalności pracy są:

• Nowe iteracyjne ujęcie problemu identyfikacji hierarchicznych struktur jednostek nazewniczych, nie stosowane do tej pory w innych metodach rozwiązujących ten problem.

• Zaproponowanie algorytmu dwukierunkowego w ramach ujęcia iteracyjnego, wykorzystującego dwa modele iteracyjne oraz funkcję selekcji. Zdefiniowanie wariantów algorytmu różniących się funkcją selekcji.

• Zaproponowanie architektury sieci neuronowej pozwalającej na iteracyjne generowanie wyników. Opracowanie metody przekazywania stanu pomiędzy poszczególnymi iteracjami modelu. Zaproponowanie procedury trenowania sieci neuronowych stosujących ujęcie iteracyjne.

1.3. Wykrywanie jednostek nazewniczych jako problem tagowania sekwencji

Niniejsza sekcja zawiera wprowadzenie do zagadnienia identyfikacji jednostek nazewniczych metodą tagowania sekwencji. Opisana metodologia dotyczy tagowania sekwencji w problemach płaskiej, jednowarstwowej predykcji. Uogólnienie tej metody do postaci iteracyjnej, pozwalającej na predykcję dowolnej liczby warstw, zostanie przedstawione w Rozdziale 5.

Tagowanie sekwencji jest najczęściej stosowaną metodą wykrywania jednostek nazewniczych. Ze względu na ograniczenia obliczeniowe, na ogół modele nie przetwarzają całych dokumentów tekstowych a mniejsze segmenty tekstu. W typowym modelu tagowania sekwencji są to zdania. Niech dany będzie zbiór X składający się z pewnej liczby sekwencji o zmiennej długości, z których każda reprezentuje pojedynczy segment tekstu. Segment o długości n zdefiniujmy jako sekwencję elementów

$$\vec{\boldsymbol{x}} = [x_1, x_2, ..., x_{n-1}, x_n] \tag{1.1}$$

gdzie elementy $x_1, ..., x_n \in V$, natomiast V jest zdefiniowanym słownikiem symboli dopuszczalnych dla danego problemu, reprezentujących słowa, cząstki słów lub pojedyncze znaki. Słownik V może również zawierać symbol specjalny *unk*, którym zastępuje się wszystkie nieznane elementy, nie mające swojej reprezentacji w słowniku. Ponadto niech dana będzie sekwencja tagów

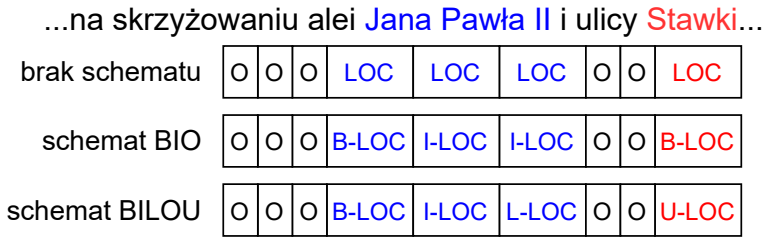
$$\vec{\boldsymbol{y}} = [y_1, y_2, ..., y_{n-1}, y_n] \tag{1.2}$$

gdzie tagi $y_1, ..., y_n \in C$, natomiast C jest zbiorem dopuszczalnych klas. W ogólnym problemie tagowania sekwencji, naszym zadaniem jest znalezienie modelu f przypisującego każdemu elementowi x_i sekwencji $\vec{\boldsymbol{x}}$ odpowiadający mu element sekwencji $\vec{\boldsymbol{y}}$.

$$f : \vec{\boldsymbol{x}} \rightarrow \vec{\boldsymbol{y}} \tag{1.3}$$

Takie podstawowe sformułowanie problemu jest wystarczające dla niektórych zastosowań. Tagowanie części mowy (ang. *part-of-speech tagging*) jest prostym przykładem tagowania sekwencji, w którym każdemu słowu przypisujemy odpowiadającą mu część mowy [33, s. 79]. Ten przypadek nie narzuca na wektor $\vec{\boldsymbol{y}}$ ograniczeń dotyczących jego struktury, zatem każda możliwa sekwencja wynikowa $\vec{\boldsymbol{y}}$ jest dozwolona.

Rozpatrzmy prosty przypadek tagowania części mowy na przykładzie zdania "I like Japanese food." Przyjmując kryterium podziału sekwencji na słowa, otrzymujemy sekwencję składającą się z pięciu elementów $\vec{\boldsymbol{x}} = [I, \textit{like}, \textit{Japanese}, \textit{food}, .]$, czterech słów oraz kropki traktowanej jako niezależny element. W rzeczywistych problemach tagowania części mowy używane są rozbudowane zbiory dopuszczalnych klas C liczące nawet kilkadziesiąt tagów, odpowiadających wszystkim częściom mowy występującym w danym języku wraz z ich wariantami. Dla uproszczenia przykładu przyjmijmy, że zbiór tagów C składa się z następujących pozycji: P - przyimek (ang. *preposition*), N - rzeczownik (ang. *noun*), V - czasownik (ang. *verb*), A - przymiotnik (ang. *adjective*), I - interpunkcja (ang. *punctuation*). Zatem dla powyższego zdania poprawnym wynikiem



Rysunek 1.2: Różnice w kodowaniu jednostek nazewniczych bez stosowania schematu tagowania oraz przy użyciu schematów BIO (beginning, inside, outside) i BILOU (beginning, inside, last, outside, unit). Na powyższym przykładzie klasa LOC oznacza jednostkę wskazującą na lokalizację.

działania modelu będzie następująca sekwencja tagów $\vec{\boldsymbol{y}} = [P, V, A, N, I]$.

Wykrywanie jednostek nazewniczych jest przypadkiem bardziej złożonym od tagowania części mowy. Mamy tutaj bowiem do czynienia z problemem klasyfikacji nie pojedynczych elementów a ciągów elementów. Tak postawiony problem wymaga zdefiniowania sposobu tagowania poszczególnych elementów sekwencji w taki sposób, aby dało się w jednoznaczny sposób zinterpretować wektor $\vec{\boldsymbol{y}}$ jako zbiór sklasyfikowanych fraz. Najprostszym sposobem byłoby założenie, że każdy podciąg sąsiadujących ze sobą elementów oznaczonych tą samą klasą traktujemy jako samodzielną jednostkę nazewniczą. Ponadto do zbioru klas C dodajemy specjalną klasę O, oznaczającą, że dany element nie przynależy do żadnej jednostki nazewniczej. W praktyce jednak taki sposób tagowania jest rzadko stosowany. Jego podstawowym mankamentem jest brak możliwości odróżnienia kilku odrębnych jednostek nazewniczych tej samej klasy znajdujących się obok siebie. Wyższą poprawność predykcji można osiągnąć stosując jeden ze znanych *schematów tagowania* (ang. *tagging scheme, chunk representation*) takich jak *BIO* czy *BILOU* [107]. W schemacie *BIO* każdą klasę encji dzielimy na dwa podtypy: Beginning (tag służący do oznaczania pierwszego elementu jednostki nazewniczej) oraz Inside (tag, którym oznaczane są kolejne elementy jednostki). Poza tym występuje standardowa klasa Outside (O) do oznaczania pozostałych słów w zdaniu. W schemacie *BILOU* zestaw ten jest rozszerzony o podtypy Last (ostatni element jednostki nazewniczej) oraz Unit (którym oznaczane są wszystkie jednostki składające się z pojedynczego elementu). Rysunek 1.2 pokazuje różnice pomiędzy poszczególnymi schematami tagowania.

Zastosowanie jednego ze schematów tagowania pozwala w łatwy sposób przetworzyć wektor tagów $\vec{\boldsymbol{y}}$ do postaci zbioru jednostek nazewniczych z odpowiadającymi im klasami. Może się jednak zdarzyć, że w wynikiem działania modelu będzie wektor niezgodny z definicją zastosowanego schematu - na przykład taki, w którym tag typu Inside wystąpi bezpośrednio po tagu Outside. Takie przypadki są mogą być korygowane regułami heurystycznymi, natomiast częstotliwość ich występowania jest uzależniona od architektury modelu. Metody, które wykorzystują modele graficzne takie jak CRF w warstwie predykcyjnej, pozwalają na znaczne zredukowanie liczby tego typu błędów [109].

1.4. Podsumowanie

W rozdziale przedstawiono problem wykrywania jednostek nazewniczych oraz jego bardziej skomplikowane warianty, w tym wykrywanie hierarchicznych struktur jednostek nazewniczych będące tematem niniejszej rozprawy. Wskazano również cele oraz oryginalne aspekty pracy. Identyfikacja jednostek nazewniczych jest istotnym zagadnieniem w kontekście przetwarzania języka naturalnego, mającym liczne praktyczne zastosowania, których przykłady zostały opisane w rozdziale. Wykrywanie jednostek nazewniczych zajduje zastosowanie przede wszystkim w systemach związanych z ekstrakcją wiedzy z tekstu i jest jednym ze wstępnych etapów przetwarzania danych w tego typu procesach. Wykryte jednostki nazewnicze wraz z kontekstem ich wystąpienia są używane jako wejście do innych modeli predykcyjnych, realizujących konkretne potrzeby biznesowe. Praktyczne znaczenie metod identyfikacji jednostek nazewniczych stanowi motywację do podjęcia tego tematu w konkteście badawczym. Modele wykrywające jednostki nazewnicze najczęściej opierają się na tagowaniu sekwencji, czyli ujęciu polegającym na transformacji tekstu do postaci sekwencji elementów i przypisaniu klasy do każdego elementu.

Rozdział 2

Wprowadzenie do modelowania sekwencji

Rozpatrywany w tej pracy problem wykrywania hierarchicznych struktur jednostek nazewniczych jest skomplikowanym zagadnieniem, a proponowane rozwiązania najczęściej łączą w sobie kilka metod uczenia maszynowego oraz techniki znane z przetwarzania języka naturalnego. Właściwe zrozumienie tych rozwiązań wymaga zatem odpowiedniego wprowadzenia i zapoznania się z klasycznymi modelami, które są najczęściej wykorzystywane jako elementy składowe bardziej złożonych systemów wykrywania jednostek. Niniejszy rozdział stanowi przegląd oraz analizę podstawowych metod stosowanych obecnie w problemach modelowania danych sekwencyjnych, skupiając się przede wszystkim na zastosowaniach związanych z przetwarzaniem tekstu. W pierwszej części opisane zostaną modele graficzne. Następnie przedstawione zostaną najpopularniejsze architektury sieci neuronowych przystosowane do przetwarzania sekwencji: sieci rekurencyjne ze szczególnym naciskiem na architekturę LSTM, sieci typu Transformer oraz sieci splotowe. W ostatniej części rozdziału pokazano sposób na połączenie modeli graficznych z architekturami neuronowymi.

2.1. Modele graficzne

Klasycznymi modelami stosowanymi w zadaniach modelowania sekwencji są probabilistyczne modele graficzne (ang. *probabilistic graphical model, PGM*). Są to modele statystyczne, w których zmienne losowe oraz zależności pomiędzy nimi mogą być przedstawione w postaci graficznej, na przykład za pomocą grafu skierowanego, nieskierowanego lub grafu czynników (ang. *factor graph*) [79]. Historia modeli graficznych sięga początku XX wieku, natomiast obecnie wciąż znajdują one wiele zastosowań. Rozwój sieci neuronowych w ostatnich latach nie spowodował mniejszego zainteresowania tego typu metodami, w badaniach z zakresu uczenia maszynowego coraz powszechniejsze stało się integrowanie metod graficznych z sieciami neuronowymi [51, 114]. W ogólnej postaci modele graficzne są używane do modelowania szerokiego zakresu zjawisk, w których występują skomplikowane zależności probabilistyczne pomiędzy zmiennymi. Uporządkowana sekwencja obserwacji jest jednak przypadkiem szczególnym, dla którego zastosowanie mają jedynie określone rodzaje modeli graficznych. Niniejsza sekcja

ograniczy się do omówienia metod stosowanych w problemach modelowania sekwencji.

Prostym przykładem sekwencyjnego modelu graficznego jest ukryty model Markowa (ang. *hidden Markov model, HMM*) [62, s. 5]. W modelu tym zakładamy istnienie sekwencji obserwacji $\vec{\boldsymbol{x}}$ oraz sekwencji stanów ukrytych $\vec{\boldsymbol{y}}$. Przyjmujemy również założenie, iż mając dany stan y_i na pozycji i-tej, stan przyszły y_{i+1} zależy wyłącznie od bieżącego stanu oraz od zaobserwowanej zmiennej x_i dla tej pozycji sekwencji. Prawdopodobieństwo łączne $p(\vec{\boldsymbol{y}}, \vec{\boldsymbol{x}})$ dane jest wzorem:

$$p(\vec{\boldsymbol{y}}, \vec{\boldsymbol{x}}) = \prod_{i=1}^{n} p(y_i|y_{i-1})p(x_i|y_i) \tag{2.1}$$

gdzie $\vec{\boldsymbol{x}}$ i $\vec{\boldsymbol{y}}$ są sekwencjami obserwacji oraz stanów modelu a n jest długością sekwencji. Dla uproszczenia wzoru przyjęliśmy istnienie dodatkowego elementu inicjalnego sekwencji y_0, tak aby rozkład prawdopodobieństwa dla pierwszego elementu $p(y_1|y_0) = p(y_1)$ [62, s. 5]. Prawdopodobieństwo sekwencji w modelu HMM liczone jest jako iloczyn dwóch składowych dla każdej pozycji sekwencji. Pierwsza składowa $p(y_i|y_{i-1})$, zwana też prawdopodobieństwem przejścia (ang. *transition probability*), informuje o tym jak prawdopodobne jest przejście ze stanu y_{i-1} do stanu y_i. Druga składowa $p(x_i|y_i)$, zwana prawdopodobieństwem emisji (ang. *emission probability*), odwierciedla prawdopodobieństwo zaobserwowania cechy x_i w stanie y_i. Obserwacje x_i mogą reprezentować bezpośrednio elementy sekwencji, na przykład słowa w przypadku danych tekstowych, lub inne cechy wyliczone na podstawie tych elementów.

Wartości prawdopodobieństw $p(x_i|y_i)$ oraz $p(y_i|y_{i-1})$ można oszacować na podstawie zbioru danych, zliczając częstotliwość występowania cech oraz stanów. Pozwala to na bezpośrednie zastosowanie wzoru 2.1 do obliczenia prawdopodobieństwa sekwencji. Natomiast z punktu widzenia problemu tagowania sekwencji, zadaniem modelu jest znalezienie "najlepszej" sekwencji stanów $\vec{\boldsymbol{y}}_*$, to znaczy takiej, która maksymalizuje prawdopodobieństwo $p(\vec{\boldsymbol{y}}, \vec{\boldsymbol{x}})$, mając daną sekwencję obserwacji $\vec{\boldsymbol{x}}$:

$$\arg\max_{\vec{\boldsymbol{y}}_* \in Y} p(\vec{\boldsymbol{y}}_*, \vec{\boldsymbol{x}}) \tag{2.2}$$

gdzie Y jest zbiorem wszystkich możliwych sekwencji stanów. Jeżeli $C = \{c_1, c_2, ..., c_k\}$ jest zbiorem dopuszczalnych stanów a n rozmiarem sekwencji, to łatwo można obliczyć, że liczba wszystkich możliwych sekwencji wynosi $|C|^n$. Podejście naiwne, polegające na sprawdzeniu wszystkich sekwencji, jest zatem w większości przypadków zbyt nieefektywne aby możliwe było jego zastosowanie w praktyce. Do wyznaczania $\vec{\boldsymbol{y}}_*$ powszechnie używany jest algorytm dekodowania Viterbiego [131], wykorzystujący programowanie dynamiczne. Algorytm ten polega na konstrukcji dwóch tablic o wymiarach $n \times |C|$, które oznaczymy jako $\boldsymbol{T^{(1)}}$ oraz $\boldsymbol{T^{(2)}}$. Dla indeksów i, j, element $T^{(1)}_{i,j}$ przechowuje maksymalne prawdopodobieństwo podsekwencji złożonej z pierwszych i stanów, której stan na pozycji i równy jest c_j. Element $T^{(2)}_{i,j}$ zawiera natomiast symbol poprzedniego stanu najlepszej znalezionej do tej pory sekwencji. Po wypełnieniu obu tablic, z $\boldsymbol{T^{(1)}}$ jesteśmy w stanie odczytać maksymalne prawdopodobieństwo całej sekwencji, natomiast z $\boldsymbol{T^{(2)}}$ odtworzyć optymalną sekwencję stanów $\vec{\boldsymbol{y}}_*$, której to prawdopodobieństwo odpowiada. Obliczenie każdej pozycji w tablicy wymaga wykonania |C| operacji, zatem złożoność

obliczeniowa całego algorytmu dekodowania wynosi $O(n \cdot |C|^2)$.

Inną powszechnie stosowaną metodą modelowania sekwencji są warunkowe pola losowe (ang. *conditional random fields, CRF*). Potocznie nazwy CRF używa się w odniesieniu do oryginalnej metody zaproponowanej przez Lafferty et al. [65] na potrzeby tagowania sekwencji obserwacji, w późniejszych latach wprowadzone zostały również inne warianty modelu operujące na bardziej złożonych strukturach [123, 10, 69]. Podstawowa metoda, zwana też linear-chain CRF, ma ogólną postać:

$$p(\vec{\boldsymbol{y}}|\vec{\boldsymbol{x}}) = \frac{1}{Z(\vec{\boldsymbol{x}})} \prod_{i=1}^{n} \Psi_i(\vec{\boldsymbol{y}}, \vec{\boldsymbol{x}}) \tag{2.3}$$

gdzie, podobnie jak w przypadku HMM, $\vec{\boldsymbol{x}}$ jest sekwencją obserwacji, $\vec{\boldsymbol{y}}$ jest sekwencją stanów, n jest rozmiarem sekwencji, $\Psi_i(\vec{\boldsymbol{y}}, \vec{\boldsymbol{x}})$ jest funkcją zwaną czynnikiem (ang. *factor*) natomiast $Z(\vec{\boldsymbol{x}})$ jest funkcją normalizującą. W ogólnym przypadku, czynniki $\Psi_i(\vec{\boldsymbol{y}}, \vec{\boldsymbol{x}})$ są dowolnymi funkcjami, które na podstawie podzbioru elementów sekwencji $\vec{\boldsymbol{y}}$ i $\vec{\boldsymbol{x}}$ określają ich poziom zgodności, przyjmując wartości nieujemne. Natomiast definicja czynników dla sekwencyjnej wersji CRF jest następująca:

$$\Psi_i(\vec{\boldsymbol{y}}, \vec{\boldsymbol{x}}) = \exp\left(\sum_{j=1}^{m} \lambda_j f_j(y_i, y_{i-1}, \vec{\boldsymbol{x}}, i)\right) \tag{2.4}$$

gdzie $f_j(y_i, y_{i-1}, \vec{\boldsymbol{x}}, i)$ są funkcjami opartymi na lokalnych cechach sekwencji dla pozycji i, natomiast wartości λ_j są parametrami modelu. W zależności od problemu możliwe jest zastosowanie różnych funkcji f_j ekstrahujących cechy z sekwencji. W przypadku danych tekstowych mogą być to cechy oparte na słowie na pozycji i, słowach leżących po prawo lub lewo od bieżącego elementu, własnościach słów takich jak wielkość liter, części mowy, pozycja w zdaniu itp. Poniżej przedstawiono prosty przykład definicji modelu, uwzględniający bieżący element sekwencji x_i oraz przejścia pomiędzy stanami y_{i-1} i y_i:

$$\Psi_i(\vec{\boldsymbol{y}}, \vec{\boldsymbol{x}}) = \exp\left(\sum_{j,k \in C} T_{j,k} \mathbb{1}_{\{y_i=j\}} \mathbb{1}_{\{y_{i-1}=k\}} + \sum_{c \in C} \sum_{w \in V} \lambda_{c,w} \mathbb{1}_{\{y_i=c\}} \mathbb{1}_{\{x_i=w\}}\right) \tag{2.5}$$

gdzie $T_{j,k}$ oraz $\lambda_{c,w}$ są parametrami modelu, C jest zbiorem dopuszczalnych stanów, V jest zbiorem symboli (słów), natomiast $\mathbb{1}_{\{a=b\}}$ jest funkcją przyjmującą wartość 1 jeżeli warunek $a = b$ jest spełniony, 0 w przeciwnym przypadku. W powyższym przykładzie $\boldsymbol{T}$ jest macierzą wag przejść pomiędzy stanami, zatem $T_{j,k}$ jest wagą przejścia sekwencji ze stanu $y_{i-1} = c_k$ do stanu $y_i = c_j$.

Należy zwrócić uwagę, że w modelu CRF czynniki nie reprezentują bezpośrednio prawdopodobieństw, ale dowolne wartości nieujemne. Dlatego też obliczenie prawdopodobieństwa warunkowego $p(\vec{\boldsymbol{y}}|\vec{\boldsymbol{x}})$ wymaga podzielenia wartości przypisanej danej sekwencji przez wartość funkcji normalizującej $Z(\vec{\boldsymbol{x}})$, reprezentującej sumę wartości

przypisanym wszystkich możliwych sekwencji stanów $\vec{\boldsymbol{y}}$:

$$Z(\vec{\boldsymbol{x}}) = \sum_{\vec{\boldsymbol{y}} \in Y} \prod_{i=1}^{n} \Psi_i(\vec{\boldsymbol{y}}, \vec{\boldsymbol{x}}) \tag{2.6}$$

Do obliczenia wartości $Z(\vec{\boldsymbol{x}})$ wykorzystuje się algorytm Forward-Backward [103] działający, podobnie jak algorytm Viterbiego, w oparciu o programowanie dynamiczne. W algorytmie Forward-Backward budowane są dwie tablice $\boldsymbol{\alpha}$ i $\boldsymbol{\beta}$ o wymiarach $n \times |C|$. Dla indeksów i, j, element $\alpha_{i,j}$ przechowuje sumę wartości wszystkich podsekwencji złożonych z pierwszych i stanów, których stan na pozycji i jest równy c_j. Tablica $\boldsymbol{\beta}$ również zlicza sumy wartości podsekwencji, ale w przeciwnym kierunku, rozpoczynając od ostatniej pozycji w sekwencji. Suma elementów dla wiersza $\alpha_{n,*}$ lub $\beta_{n,*}$ jest wartością funkcji normalizującej $Z(\vec{\boldsymbol{x}})$. Tablice $\boldsymbol{\alpha}$ i $\boldsymbol{\beta}$ mogą być również wykorzystywane do wyznaczania prawdopodobieństwa $p(y_i|\vec{\boldsymbol{x}})$, czyli prawdopodobieństwa wystąpienia stanu na pozycji i dla sekwencji obserwacji $\vec{\boldsymbol{x}}$.

Do znalezienia optymalnego wektora $\vec{\boldsymbol{y}}_*$, maksymalizującego prawdopodobieństwo $p(\vec{\boldsymbol{y}}_*|\vec{\boldsymbol{x}})$:

$$\arg\max_{\vec{\boldsymbol{y}}_* \in Y} p(\vec{\boldsymbol{y}}_*|\vec{\boldsymbol{x}}) \tag{2.7}$$

można użyć, tak jak w przypadku HMM, algorytmu dekodowania Viterbiego. Jeżeli nie zależy nam na wartości prawdopodobieństwa $p(\vec{\boldsymbol{y}}_*|\vec{\boldsymbol{x}})$, samo wyznaczenie wektora $\vec{\boldsymbol{y}}_*$ podczas dekodowania nie wymaga obliczania wartości $Z(\vec{\boldsymbol{x}})$, możemy posługiwać się nieznormalizowanymi wartościami przypisanymi do poszczególnych sekwencji.

Trenowanie modelu linear-chain CRF polega na minimalizacji funkcji kosztu względem parametrów modelu λ_j:

$$J = -\sum_{k=1}^{N} \sum_{i=1}^{n} \sum_{j=1}^{m} \lambda_j f_j(y_i^{(k)}, y_{i-1}^{(k)}, \boldsymbol{x}^{\vec{(k)}}, i) + \sum_{k=1}^{N} \ln Z(\boldsymbol{x}^{\vec{(k)}}) \tag{2.8}$$

gdzie N jest rozmiarem zbioru danych, n jest rozmiarem pojedynczej sekwencji, m jest liczą funkcji cech, natomiast $\boldsymbol{x}^{\vec{(k)}}$ i $\boldsymbol{y}^{\vec{(k)}}$ są sekwencjami obserwacji i stanów dla k-tej próbki treningowej.

2.2. Rekurencyjne sieci neuronowe

Sieci neuronowe są, poza modelami graficznymi, koleją grupą metod powszechnie stosowaną do przetwarzania sekwencji obserwacji, szczególnie w przypadkach gdy dane wejściowe są skomplikowanymi obiektami reprezentowanymi przez macierze o wysokiej liczbie wymiarów, dla których manualna ekstrakcja cech ma ograniczone zastosowanie. Klasyczne jednokierunkowe sieci neuronowe nie są jednak przystosowane do przetwarzania uporządkowanych sekwencji. Możliwe jest co prawda zdefiniowanie wielu wejść odpowiadających poszczególnym elementom sekwencji, natomiast dla tego typu architektur kolejność wejść nie ma znaczenia, sieć nie jest więc w stanie modelować zależności temporalnych pomiędzy elementami. Powstało kilka grup architektur neuronowych, które próbują zaadresować ten problem. Niniejsza sekcja skupia się na rekurencyjnych

sieciach neuronowych, rodzinie modeli stosowanych od kilkudziesięciu lat w problemach przetwarzania sekwencji.

Rekurencyjne sieci neuronowe zostały spopularyzowane w latach 80-tych [113, 49], natomiast początkowo ich zastosowanie było ograniczone ze względu na problemy ze stabilnością procesu trenowania oraz trudności w modelowaniu długich sekwencji. Duże znaczenie dla rozwoju sieci rekurencyjnych miało wprowadzenie architektury LSTM (long short-term memory) [48], która rozwiązała część z tych kwestii. Większość stosowanych współcześnie sieci rekurencyjnych opiera się na architekturze LSTM lub jej uproszczonej wersji - GRU (gated recurrent unit) [13].

Podstawową różnicą pomiędzy klasycznymi sieciami neuronowymi a sieciami rekurencyjnymi jest obecność pewnego rodzaju pamięci, w której zapisywany jest aktualny stan sieci podczas przetwarzania kolejnych elementów sekwencji. Na wysokim poziomie ogólności, dla elementu sekwencji na pozycji t sieć rekurencyjna oblicza nowy stan pamięci $\boldsymbol{h_t}$ na podstawie tego elementu oraz stanu pamięci w poprzednim kroku, zaś odpowiedź sieci może być wygenerowana na podstawie aktualnego stanu pamięci:

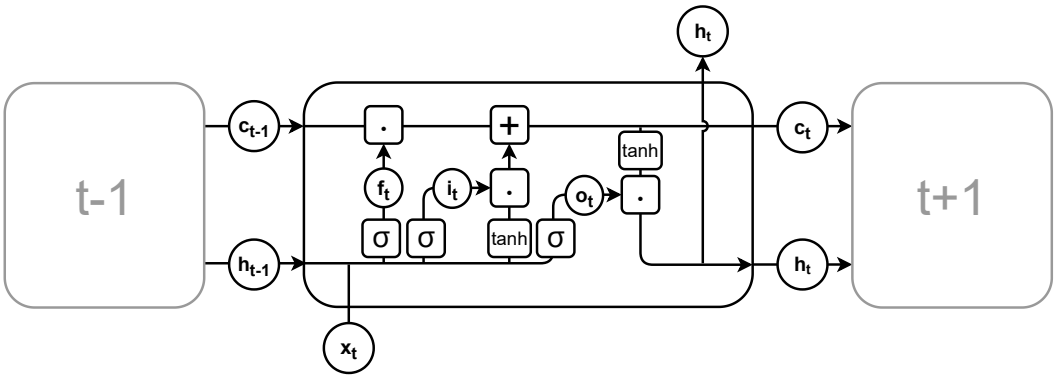
$$\boldsymbol{h_t} = g(\boldsymbol{x_t}, \boldsymbol{h_{t-1}}), \quad \boldsymbol{y_t} = f(\boldsymbol{h_t}) \tag{2.9}$$

gdzie $\boldsymbol{h_{t-1}}$ i $\boldsymbol{h_t}$ są stanami pamięci sieci w poprzednim oraz aktualnym kroku, $\boldsymbol{x_t}$ i $\boldsymbol{y_t}$ są odpowiednio wejściem i wyjściem sieci w aktualnym kroku, g jest funkcją obliczającą stan pamięci dla warstwy rekurencyjnej, a f jest funkcją obliczającą wyjście sieci.

Architektury rekurencyjne różnią się reprezentacją pamięci oraz sposobem jej aktualizacji podczas przetwarzania sekwencji. W sieciach LSTM przepływ informacji pochodzących z wektora wejściowego oraz z pamięci sieci jest kontrolowany przez specjalne bramki (*gates*), reprezentowane przez wektory wartości z zakresu 0-1. W warstwie LSTM występują trzy takie bramki: *input gate*, *forget gate* oraz *output gate*. Wewnętrzny stan sieci, zwany też stanem komórki (*cell state*), obliczany jest jako suma poprzedniego stanu oraz wektora wyliczonego na podstawie aktualnego elementu sekwencji $\boldsymbol{x_t}$. Zakres danych przepływających do nowego stanu komórki zależy od wektora *forget gate*, który ogranicza przepływ poprzedniego stanu, oraz *input gate*, mającego za zadanie ograniczyć przepływ nowego stanu. Dzięki temu sieć LSTM może precyzyjnie sterować jaka część pamięci powinna zostać "zapomniana", a jaka "zapamiętana" w komórce. Cały proces przetwarzania elementu sekwencji w kroku t możemy zapisać za pomocą następujących wzorów [48]:

$$\begin{aligned}
\boldsymbol{i_t} &= \sigma(\boldsymbol{W_{ii}}\boldsymbol{x_t} + \boldsymbol{W_{hi}}\boldsymbol{h_{t-1}} + \boldsymbol{b_i}) \\
\boldsymbol{f_t} &= \sigma(\boldsymbol{W_{if}}\boldsymbol{x_t} + \boldsymbol{W_{hf}}\boldsymbol{h_{t-1}} + \boldsymbol{b_f}) \\
\boldsymbol{o_t} &= \sigma(\boldsymbol{W_{io}}\boldsymbol{x_t} + \boldsymbol{W_{ho}}\boldsymbol{h_{t-1}} + \boldsymbol{b_o}) \\
\boldsymbol{c_t} &= \boldsymbol{f_t} \cdot \boldsymbol{c_{t-1}} + \boldsymbol{i_t} \cdot \tanh(\boldsymbol{W_{ig}}\boldsymbol{x_t} + \boldsymbol{W_{hg}}\boldsymbol{h_{t-1}} + \boldsymbol{b_g}) \\
\boldsymbol{h_t} &= \boldsymbol{o_t} \cdot \tanh(\boldsymbol{c_t})
\end{aligned} \tag{2.10}$$

gdzie $\cdot$ jest iloczynem skalarnym wektorów, σ oznacza funkcję sigmoid, $\boldsymbol{i_t}$, $\boldsymbol{f_t}$, i $\boldsymbol{o_t}$ są odpowiednio wektorami bramek *input gate*, *forget gate* oraz *output gate*. Każda bramka jest parametryzowana przez indywidualne macierze wag (symbole $\boldsymbol{W}$) oraz wektor bias (symbole $\boldsymbol{b}$). $\boldsymbol{c_t}$ jest wyliczonym stanem komórki, a $\boldsymbol{h_t}$ stanem wyjściowym



Rysunek 2.1: Schemat przetwarzania elementu sekwencji x_t w komórce LSTM. Symbole w kółkach oznaczają wektory, natomiast zaokrąglone prostokąty operacje na wektorach. Symbole $\cdot$, $+$, σ, tanh oznaczają odpowiednio iloczyn skalarny, sumę, funkcję sigmoid oraz tangens hiperboliczny. Funkcje *tanh* oraz σ są obliczane niezależnie dla każdej współrzędnej wektora.

warstwy LSTM w kroku t. Podczas wyliczania wewnętrznego oraz wyjściowego stanu warstwy używana jest funkcja *tanh* we celu normalizacji wartości wektorów do zakresu od -1 do 1. Argumentami funkcji *tanh* oraz σ w powyższych formułach są wektory. Zastosowanie tych funkcji na wektorze oznacza obliczenie wartości funkcji dla każdej współrzędnej wektora niezależnie. Schemat działania warstwy LSTM przedstawiono również w formie graficznej na Rysunku 2.1.

Prostym oraz często stosowanym w praktyce rozszerzeniem architektur rekurencyjnych są dwukierunkowe sieci rekurencyjne [116, 37]. Idea sieci dwukierunkowych polega na wykorzystaniu dwóch osobnych warstw rekurencyjnych, z których jedna przetwarza sekwencję od pierwszego do ostatniego elementu, a druga od ostatniego do pierwszego. Dla każdej pozycji t, sieć zamiast pojedynczego wektora $\boldsymbol{h_t}$ zwraca zatem dwa wektory wyjściowe: $\boldsymbol{fh_t}$ oraz $\boldsymbol{bh_t}$. Ostateczny wektor wyjściowy jest w tym przypadku budowany poprzez konkatenację obu wektorów kierunkowych:

$$\boldsymbol{h_t} = \boldsymbol{fh_t} \odot \boldsymbol{bh_t} \tag{2.11}$$

gdzie $\odot$ jest operacją konkatenacji wektorów, $\boldsymbol{fh_t}$ i $\boldsymbol{bh_t}$ są wektorami warstw przetwarzających sekwencję w przód oraz w tył, natomiast $\boldsymbol{h_t}$ jest wynikowym wektorem sieci dwukierunkowej. Połączenie różnych reprezentacji tej samej sekwencji na ogół pozwala na budowanie modeli o wyższej skuteczności predykcji niż przypadku zastosowania jednokierunkowej warstwy rekurencyjnej.

2.3. Sieci Transformer

W obszarze sieci neuronowych, sieci rekurencyjne przez długi czas były standardowym rozwiązaniem stosowanym w problemach modelowania sekwencji. W ostatnich latach coraz większą popularność zdobywają jednak architektury opierające się na innych podejściach niż rekurencyjne. Zdecydowanie najpopularniejszym spośród tych rozwiązań jest architektura Transformera, pierwszy raz zaprezentowana przez Vaswani et al. [130]

w kontekście problemu tłumaczenia maszynowego. Pierwsza wersja Transformera była architekturą typu enkoder-dekoder, w której część sieci odpowiadała za budowanie ukrytej reprezentacji sekwencji wejściowej, natomiast druga część za wygenerowanie sekwencji wyjściowej na podstawie tej reprezentacji. Na dużym poziomie ogólności architektura Transformera składa się z pewnej liczby powtarzających się struktur obliczeniowych, nazywanymi odpowiednio *blokami enkodera* oraz *blokami dekodera*. Pełna architektura stosowana jest głównie w problemach, w których zarówno wejściem jak i wyjściem sieci jest tekst, na przykład w tłumaczeniu maszynowym, sumaryzacji, parafrazowaniu, normalizacji tekstu, automatycznej korekcie gramatycznej. Na potrzeby problemów predykcji takich jak klasyfikacja tekstu czy tagowanie sekwencji, nie jest konieczne korzystanie z części dekodującej sieci[1], dlatego też stosuje się prostsze rozwiązania wykorzystujące jedynie część enkodującą. Podobnie jak w przypadku warstw rekurencyjnych, bloki enkodera służą do obliczenia ukrytej reprezentacji wektorowej poszczególnych elementów sekwencji, która może być następnie wykorzystana w warstwie predykcyjnej do wygenerowania odpowiedzi sieci.

Niniejsza sekcja skupia się na omówieniu działania enkodera w sieciach typu Transformer. Obecnie istnieją różne warianty tej architektury, będziemy się zatem opierać na oryginalnym modelu BERT [24], który jako pierwszy zaprezentował możliwości Transformera w zadaniach klasyfikacji i tagowania sekwencji. Dla danej sekwencji elementów $\vec{\boldsymbol{x}}$, sieć buduje inicjalną reprezentację wektorową tych elementów. Następnie wektory są przetwarzane przez pewną liczbę bloków enkodera, uzależnioną od rozmiaru sieci - dla małych sieci jest to kilka bloków, natomiast największe stosowane modele wykorzystują ich kilkadziesiąt. Struktura wszystkich bloków jest identyczna, natomiast każdy z nich jest parametryzowany przez indywidualny zestaw wag. Pojedynczy blok przyjmuje na wejściu sekwencję wektorów i generuje na wyjściu nową sekwencję o identycznej liczbie elementów i takiej samej wymiarowości wektora. Pozwala to na bezpośrednie użycie sekwencji wyjściowej jednego bloku jako wejścia do kolejnego bloku. Wyjście ostatniego z bloków jest finalną reprezentacją sekwencji.

Wstępna reprezentacja sekwencji Zakładając, że istnieje zdefiniowany słownik dopuszczalnych symboli V, sieć przechowuje reprezentację wektorową każdego z symboli w macierzy $\boldsymbol{W_e}$ o wymiarach $|V| \times d$, gdzie d jest przyjętym rozmiarem wektora. Przed rozpoczęciem uczenia modelu, macierz ta jest inicjalizowana losowo, a podczas uczenia trenowana tak jak pozostałe wagi modelu. Każdy wiersz macierzy stanowi inicjalną reprezentację tokena, niezależną od kontekstu jego wystąpienia. Nie jest to jednak jedyna składowa wstępnej reprezentacji elementu sekwencji. Reprezentacja wektorowa elementu jest wyliczana jako suma trzech wektorów:

$$\begin{gathered} \boldsymbol{w_{e,i}} = f_e(\boldsymbol{W_e}, i), \quad \boldsymbol{w_{s,i}} = f_s(\boldsymbol{W_s}, i), \quad \boldsymbol{w_{p,i}} = f_p(i) \\ \boldsymbol{x_i} = \boldsymbol{w_{e,i}} + \boldsymbol{w_{s,i}} + \boldsymbol{w_{p,i}} \end{gathered} \tag{2.12}$$

[1]W celach budowy wektorowych reprezentacji sekwencji stosowane są głównie uproszczone wersje Transformera wykorzystujące tylko bloki enkodera, istnieją jednak również rozwiązania wykorzystujące pełną architekturę enkoder-dekoder [74, 106], których omówienie wykracza poza zakres tej pracy.

gdzie $\boldsymbol{x_i} \in \mathbb{R}^d$ jest wynikowym wektorem dla elementu na pozycji i, $\boldsymbol{w_{e,i}}, \boldsymbol{w_{s,i}}, \boldsymbol{w_{p,i}} \in \mathbb{R}^d$ są odpowiednio reprezentacjami tokena, segmentu oraz pozycji, f_e jest funkcją zwracającą reprezentację wektorową tokena znajdującego się na pozycji i z macierzy $\boldsymbol{W_e}$, f_s jest funkcją zwracającą reprezentację wektorową segmentu sekwencji dla pozycji i z macierzy $\boldsymbol{W_s}$, natomiast f_p jest funkcją obliczającą wektor pozycyjny dla pozycji i. Znaczenie macierzy $\boldsymbol{W_s}$ jest podobne jak $\boldsymbol{W_e}$, przy czym służy ona do przechowywania reprezentacji segmentów a nie tokenów. W oryginalnej implementacji reprezentacja ta była używana do odróżniania od siebie kilku sekwencji, jeżeli zostały połączone w jedną sekwencję i przekazane jako wejście modelu, na przykład w zadaniu klasyfikacji par zdań. W większości popularnych wariantów Transformera reprezentacja segmentów nie jest już używana. Ostatnią składową jest reprezentacja pozycyjna $\boldsymbol{w_{p,i}}$, służąca do zakodowania informacji o pozycji elementu w sekwencji. W przeciwieństwie do sieci rekurencyjnych, w omawianej architekturze elementy nie są przetwarzane w kolejności ich występowania, konieczne jest więc bezpośrednie umieszczenie takiej informacji w wektorze. W oryginalnym wariancie modelu wektor pozycyjny obliczany jest poprzez naprzemienne stosowanie funkcji *sinus* i *cosinus* odpowiednio dla parzystych i nieparzystych wymiarów wektora, zgodnie z następującym schematem:

$$\boldsymbol{w_{p,i}} = [sin(p(i,1)), cos(p(i,1)), sin(p(i,2)), cos(p(i,2)), ..., cos(p(i,d/2))] \tag{2.13}$$

gdzie i jest pozycją tokena, d jest rozmiarem wektora, a $p(i,k)$ jest funkcją zdefiniowaną następująco [130]:

$$p(i,k) = \frac{i}{10000^{2k/d}} \tag{2.14}$$

W niektórych implementacjach zamiast wyliczanych wektorów pozycyjnych stosowana jest dodatkowa macierz parametrów $\boldsymbol{W_p}$, pozwalająca na wytrenowanie reprezentacji wektorowej dla każdej pozycji, analogicznie jak ma to miejsce w przypadku tokenów i segmentów.

Obliczone w powyższy sposób reprezentacje wektorowe wszystkich elementów sekwencji są przekazywane jako wejście do pierwszego bloku enkodera.

Blok enkodera Pojęcie bloku rozumiemy jako zbiór operacji, które przekształcają wejściową sekwencję składającą się z reprezentacji wektorowych tokenów $[\boldsymbol{x_1}, \boldsymbol{x_2}, ..., \boldsymbol{x_n}]$ na sekwencję wyjściową zawierającą przekształcone wektory $[\boldsymbol{x'_1}, \boldsymbol{x'_2}, ..., \boldsymbol{x'_n}]$. W przeciwieństwie do sieci rekurencyjnych, w których ukryta reprezentacja elementu na pozycji i jest wyliczana tylko na podstawie elementów go poprzedzających, w architekturze Transformera każdy stan ukryty enkodera jest obliczany na podstawie wszystkich elementów sekwencji przy pomocy mechanizmu uwagi (ang. *attention*). W ogólnej postaci mechanizm uwagi można przedstawić jako ważoną sumę:

$$\boldsymbol{o_i} = \sum_{j=1}^{n} a_{i,j} f(\boldsymbol{x_j}) \tag{2.15}$$

gdzie $\boldsymbol{o_i}$ jest reprezentacją wynikową dla pozycji i, $a_{i,j}$ jest wagą oznaczającą miarę "zgodności" pomiędzy wektorami na pozycjach i i j, natomiast $f(\boldsymbol{x_j})$ jest pewną funkcją przekształcającą wektor wejściowy na pozycji j.

W przypadku architektury Transformera, stosowany jest wariant mechanizmu uwagi zwany *self-attention*. W wariancie tym dla każdej pozycji i posługujemy się trzema typami wektorów, będącymi prostymi przekształceniami liniowymi wektora wejściowego. Z wektora $\boldsymbol{x_i}$ generowane są następujące reprezentacje pośrednie: $\boldsymbol{q_i}$ zwana *zapytaniem* (ang. *query*), $\boldsymbol{k_i}$ zwana *kluczem* (ang. *key*) oraz $\boldsymbol{v_i}$ zwana *wartością* (ang. *value*). Dla pary tokenów znajdujących się na pozycjach i i j, miara "zgodności" $a_{i,j}$ wyliczana jest na podstawie *zapytania* $\boldsymbol{q_i}$ oraz *klucza* $\boldsymbol{k_j}$ jako wynik funkcji softmax ze znormalizowanego iloczynu skalarnego tych wektorów, natomiast w sumowaniu wykorzystywany jest wektor *wartości* $\boldsymbol{v_j}$. Celem funkcji softmax jest sprowadzenie wspomnianych iloczynów skalarnych wektorów do zakresu [0,1]. Mając dany k elementowy wektor $\boldsymbol{x}$, funkcja softmax jest stosowana dla każdego elementu tego wektora w następujący sposób:

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_k}} \tag{2.16}$$

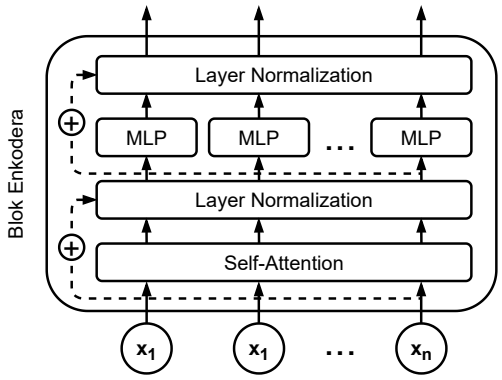
Ostatecznie, pełny proces obliczania wektora wynikowego w mechanizmie *self-attention* dla pozycji i można zapisać za pomocą następujących formuł:

$$\begin{aligned} \boldsymbol{q_i} = \boldsymbol{W_q}\boldsymbol{x_i}, \quad \boldsymbol{k_i} = \boldsymbol{W_k}\boldsymbol{x_i}, \quad \boldsymbol{v_i} = \boldsymbol{W_v}\boldsymbol{x_i} \\ a_{i,j} = \text{softmax}\left(\frac{q_i \cdot k_i}{\sqrt{d}}\right) = \frac{e^{\frac{1}{\sqrt{d}}(q_i \cdot k_i)}}{\sum_{j=1}^{n} e^{\frac{1}{\sqrt{d}}(q_i \cdot k_j)}} \\ \boldsymbol{o_i} = \sum_{j=1}^{n} a_{i,j}\boldsymbol{v_j} \end{aligned} \tag{2.17}$$

gdzie $\cdot$ jest iloczynem skalarnym wektorów, $\boldsymbol{o_i}$ jest wektorem wynikowym, $\boldsymbol{W_q}, \boldsymbol{W_k}, \boldsymbol{W_v}$ są macierzami wag przekształcającymi wektor wejściowy do postaci *zapytań* $\boldsymbol{q_i}$, *kluczy* $\boldsymbol{k_i}$ oraz *wartości* $\boldsymbol{v_i}$, d jest rozmiarem wektorów, a pierwiastek $\sqrt{d}$ jest używany jako stała normalizująca.

Dodatkowym rozszerzeniem mechanizmu *self-attention* stosowanym w enkoderze jest tzw. *multi-head self-attention*. Polega ono na podzieleniu wektora wejściowego $\boldsymbol{x_i}$ na m równych części, a następnie zastosowania *self-attention* dla każdej z tych części osobno. Dla d wymiarowego wektora, każda część będzie miała zatem wymiar d/m. Technika ta jest równoznaczna z zastosowaniem m niezależnych, osobno parametryzowanych warstw *self-attention*. Uzasadnieniem dla takiego rozwiązania jest możliwość wyodrębnienia różnych cech podczas konstruowania wynikowej reprezentacji wektorowej. Po przeliczeniu wektorów wynikowych, są one ponownie konkatenowane, tworząc jeden wektor wynikowy o wymiarze d.

Przeliczenie nowych reprezentacji wektorowych przy użyciu mechanizmu *self-attention* jest pierwszym etapem przetwarzania sekwencji w bloku enkodera. Schemat całego bloku przedstawiono na Rysunku 2.2. Składa się on z dwóch transformacji oraz następującej po każdej z nich warstwy normalizującej z dodatkowym połączeniem rezydualnym [46]. Normalizacja wektorów w tej warstwie polega na dodaniu do siebie wynikowych reprezentacji wektorowych oraz reprezentacji wejściowych, a następnie zastosowaniu metody opisanej w Ba et al. [4]. Normalizacja ta ma na celu stabilizację uczenia sieci, zmniejszenie ryzyka wystąpienia problemów takich jak eksplodujący lub



Rysunek 2.2: Schemat bloku enkodera w architekturze Transformera. Linie przerywane oznaczają połączenia rezydualne pomiędzy warstwami. Mechanizm uwagi oraz normalizacja są operacjami wykonywanymi na pełnej sekwencji wektorów. Transformacja poprzez sieć perceptronową jest wykonywana na każdym wektorze niezależnie.

zanikający gradient [33, s. 60], pełni również rolę regularyzacyjną i może potencjalnie przyspieszyć proces uczenia poprzez zmniejszenie liczby iteracji niezbędnych do osiągnięcia optimum. Drugą z transformacji występujących w bloku enkodera jest prosta sieć perceptronowa z jedną warstwą ukrytą, parametryzowana przez macierze wag $\boldsymbol{W_f^{(1)}}$ i $\boldsymbol{W_f^{(2)}}$. Pomnożenie wektora przez pierwszą macierz zwiększa jego wymiar, następnie jest on transformowany przez funkcję aktywacji ReLU (ang. *rectifier linear unit*) [33, s. 45], a kolejna macierz wag sprowadza wektor do oryginalnego wymiaru d. Operacje te możemy zapisać wzorem:

$$\boldsymbol{o_i} = \boldsymbol{W_f^{(2)}}\,\mathrm{ReLU}(\boldsymbol{W_f^{(1)}}\boldsymbol{x_i} + \boldsymbol{b_f^{(1)}}) + \boldsymbol{b_f^{(2)}} \tag{2.18}$$

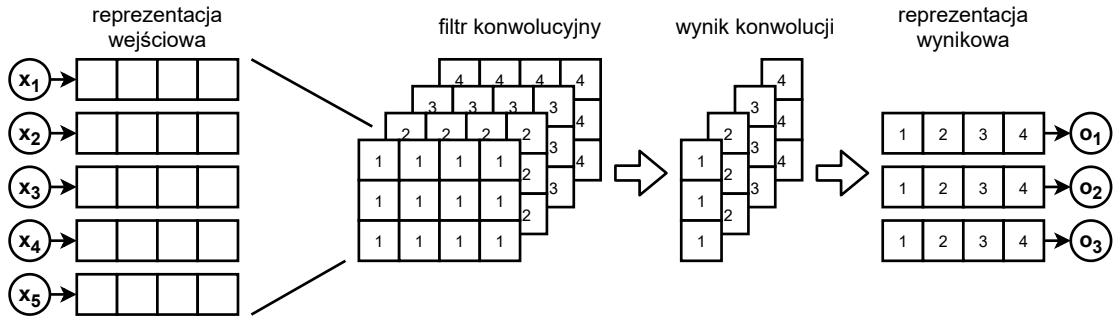
gdzie $\boldsymbol{x_i}$ jest wektorem wejściowym, $\boldsymbol{o_i}$ jest wektorem wynikowym, $\boldsymbol{W_f^{(1)}}$, $\boldsymbol{W_f^{(2)}}$, $\boldsymbol{b_f^{(1)}}$, $\boldsymbol{b_f^{(2)}}$ są macierzami wag i wektorami bias sieci perceptronowej, natomiast ReLU jest funkcją nieliniową o następującej definicji:

$$\mathrm{ReLU}(x) = max(0, x) \tag{2.19}$$

Po powyżej opisanym przekształceniu następuje kolejna warstwa normalizująca, wynik której jest ostateczną reprezentacją wyjściową dla pojedynczego bloku enkodera.

2.4. Splotowe sieci neuronowe

Splotowe sieci neuronowe (ang. *convolutional neural networks, CNN*) są jedną z najbardziej znanych architektur neuronowych. Współcześnie występują w wielu wariantach będących kombinacją kilku podstawowych struktur takich jak filtry splotowe lub operacje grupowania (ang. *pooling*). Popularność tego typu modeli jest związana przede wszystkim z zastosowaniami w obszarze przetwarzania obrazów. Pierwsze sukcesy w rozpoznawaniu obrazów przy ich użyciu osiągnięto na przełomie lat 80-tych i 90-tych [70, 71]. W ostatniej dekadzie również mogliśmy zaobserwować duży wzrost zaintereso-



Rysunek 2.3: Schemat działania operacji splotu na przykładzie danych sekwencyjnych $\vec{\boldsymbol{x}} = [\boldsymbol{x_1}, ..., \boldsymbol{x_5}]$ oraz czterech filtrów splotowych o wymiarach 3×4.

wania sieciami splotowymi, na co wpływ miało przede wszystkich upowszechnienie wydajnych procesorów graficznych, pozwalających na trenowanie modeli z większą liczbą parametrów i na większych zbiorach danych. Współczesne modele oparte na głębokich, wielowarstwowych architekturach udowodniły efektywność podejścia splotowego w rozpoznawaniu tysięcy kategorii obiektów na skomplikowanych obrazach [63, 120, 124].

Podstawą działania sieci splotowych jest operacja splotu (konwolucji). W kontekście przetwarzania obrazu, polega ona na nałożeniu filtra splotowego w określonym punkcie tego obrazu, dając w wyniku sumę wartości z najbliższego sąsiedztwa tego punktu ważoną przez wartości nałożonego filtra. Dla przypadku dwóch wymiarów, mając daną macierz wejściową $\boldsymbol{X}$ oraz filtr splotowy $\boldsymbol{W}$ o wymiarach $n \times m$, operacja splotu w punkcie i, j dana jest wzorem:

$$s(i,j) = \sum_{k=1}^{n} \sum_{l=1}^{m} X_{i+k-1,j+l-1} W_{k,l} \tag{2.20}$$

gdzie $X_{i,j}$ jest wartością w macierzy $\boldsymbol{X}$ w punkcie i, j a $W_{k,l}$ jest wartością filtra splotowego na pozycji k, l. Warstwa splotowa działa na zasadzie okna przesuwnego - filtr splotowy jest aplikowany dla każdej dopuszczalnej pozycji macierzy, tworząc w wyniku kolejną macierz zwaną mapą cech (ang. *feature map*). W klasycznych sieciach splotowych macierz ta może być bezpośrednio użyta jako wejście dla kolejnej warstwy splotowej, może też być wykonana na niej operacja grupowania w celu dodatkowej redukcji wymiarów. Powszechne jest też stosowanie wielu filtrów splotowych w pojedynczej warstwie, parametryzowanych osobnymi macierzami wag. W założeniu każdy z takich filtrów może w procesie uczenia sieci stać się wyspecjalizowanym ekstraktorem cech, dostosowanym do wykrywania określonego rodzaju struktur lokalnych występujących w danych.

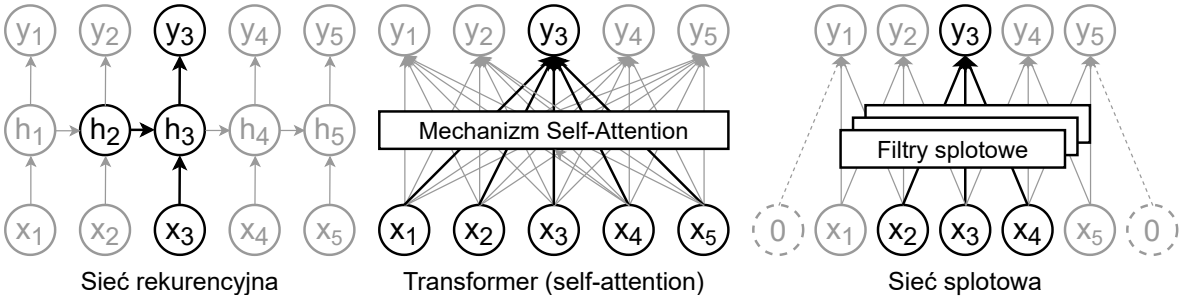
Zastosowanie sieci splotowych w przetwarzaniu języka naturalnego wydaje się mniej intuicyjne niż w przypadku rozpoznawania obrazów, ich głównym przeznaczeniem jest bowiem przetwarzania danych o strukturze przestrzennej, w których występują silne zależności lokalne pomiędzy zmiennymi. W kontekście języka wciąż popularniejszymi architekturami pozostają sieci rekurencyjne oraz Transformery, natomiast rozwiązania oparte o sieci splotowe również znalazły zastosowanie. Modelowanie sekwencji wektorów w sieciach splotowych na ogół polega na sprowadzeniu jej do postaci macierzy

o wymiarach $n \times d$, gdzie n jest rozmiarem sekwencji a d wymiarem pojedynczego wektora reprezentującego jej element. Standardowy filtr splotowy jest konstruowany w taki sposób aby przetwarzał reprezentacje wektorowe sąsiadujących ze sobą elementów. Przyjmując rozmiar kontekstu jako k, rozmiar pojedynczego filtra będzie wynosił $k \times d$. Na Rysunku 2.3 przedstawiono przykład konstrukcji wynikowych reprezentacji dla pięcioelementowej sekwencji przy pomocy splotu, z zastosowaniem czterech filtrów oraz kontekstem $k = 3$.

W powyższym przykładzie zastosowanie splotu spowodowało zmniejszenie liczby wektorów z pięciu do trzech. Filtr splotowy nie mógł być zastosowany na pozycjach brzegowych, ponieważ wykraczałby on poza rozmiar macierzy wejściowej. Ten typ splotu nazywany jest splotem wąskim (ang. *narrow convolution*). Z uwagi efekt redukcji wymiarów, wielowarstwowe sieci ze splotem wąskim mają zastosowanie głównie w problemach modelowania sekwencji, w których pożądane jest wygenerowanie zwartej reprezentacji będącej podsumowaniem całej sekwencji. Z tego też względu taki typ splotu stosowany był przede wszystkim w klasyfikacji tekstu [59, 55, 26, 145]. Rozwiązaniem pozwalającym na zachowanie tej samej liczby elementów w sekwencji wyjściowej jest rozszerzenie macierzy o pewną liczbę zerowych wektorów, dzięki którym możliwe staje się zastosowanie filtra splotowego również na pozycjach brzegowych macierzy. Jest to tak zwany splot szeroki (ang. *wide convolution*), mający zastosowanie w zadaniach wymagających osobnych reprezentacji dla każdego elementu sekwencji takich jak tagowanie sekwencji [138, 147].

2.5. Porównanie architektur neuronowych

W poprzednich sekcjach opisane zostały architektury sieci neuronowych służące do modelowania danych sekwencyjnych. Każda z tych architektur proponuje inny sposób przetwarzania sekwencji wektorów. Każde z ujęć modelowania sekwencji posiada też charakterystyczne dla niego ograniczenia i problemy. W niniejszej sekcji podsumowane zostaną różnice pomiędzy modelami neuronowymi, wskazane zostaną również ich główne wady i zalety. Na Rysunku 2.4 pokazano uproszczony schemat konstruowania wynikowej reprezentacji na podstawie wejściowej sekwencji wektorów w przypadku warstwy rekurencyjnej, warstwy *self-attention* występującej w sieciach Transformer oraz warstwy splotowej. Warto zwrócić uwagę na oznaczone ciemniejszym kolorem zależności pomiędzy wejściem a wyjściem warstwy. Warstwa rekurencyjna przetwarza dane pozycja po pozycji, zapisując aktualny stan sekwencji w pamięci komórki, dlatego też liczba bezpośrednich zależności w jej przypadku jest najmniejsza - stan wynikowy sieci zależy wyłącznie od poprzedniego stanu pamięci oraz wektora wejściowego dla tej samej pozycji. Przeciwieństwem tego podejścia jest moduł *self-attention*, w którym wyjście jest konstruowane w oparciu o wszystkie reprezentacje wejściowe. Warstwa jest implementacją relacji "każdy z każdym", pozwalając na bezpośrednie modelowanie zależności pomiędzy parami wektorów na dowolnych pozycjach. W warstwie splotowej reprezentacja wektorowa dla danej pozycji jest zależna od najbliższego kontekstu - wektora znajdującego się na tej pozycji oraz ustalonej liczby wektorów z nim sąsiadujących. W przeciwieństwie do dwóch pozostałych ujęć, pojedyncza warstwa splotowa nie jest w stanie modelować zależności pomiędzy odległymi elementami sekwencji jeżeli nie na-



Rysunek 2.4: Porównanie neuronowych architektur wykorzystywanych w problemach modelowania sekwencji: sieci rekurencyjnej, mechanizmu *self-attention* oraz sieci splotowej. Elementy $\boldsymbol{x_1}, ..., \boldsymbol{x_5}$ są wektorami wejściowymi, elementy $\boldsymbol{y_1}, ..., \boldsymbol{y_5}$ wektorami wynikowymi sieci. Ciemniejszym kolorem oznaczono wektory biorące udział w obliczeniu reprezentacji wektora $\boldsymbol{y_3}$.

leżą one do tego samego kontekstu. Modelowanie dalszych relacji wymaga rozszerzania kontekstu lub nakładania na siebie kilku warstw splotowych.

Poniżej krótko opisano zalety i wady poszczególnych architektur neuronowych:

• **Sieci LSTM** działają na zasadzie iteracyjnej aktualizacji zawartości pamięci, powtarzając te same operacje dla każdego elementu sekwencji. Struktura sieci nie narzuca zatem żadnych ograniczeń na rozmiar danych. Dodatkowo złożoność obliczeniowa warstwy rośnie liniowo wraz ze wzrostem liczby elementów, pozwalają więc na przetwarzania w rozsądnym czasie nawet bardzo długich sekwencji. Z drugiej strony sieci rekurencyjne nie są w stanie w pełni wykorzystać potencjału współczesnych procesorów graficznych, będących obecnie standardową platformą sprzętową do trenowania i wdrażania sztucznych sieci neuronowych. Specyfika tych jednostek obliczeniowych umożliwia uruchamianie wielu obliczeń współbieżnie, tymczasem sieci rekurencyjne nie pozwalają na zrównoleglenie przetwarzania sekwencji. Obliczenie reprezentacji wektorowej dla elementu na danej pozycji wymaga wcześniejszego przeliczenia wszystkich poprzedzających pozycji. W kontekście optymalizacji sprzętowej wypadają więc gorzej od dwóch pozostałych architektur, które mają potencjał do większego zrównoleglania obliczeń. Wadą tej architektury jest też rozmiar pamięci w warstwie rekurencyjnej. Podczas przetwarzania sekwencji cały dotychczasowy jej stan jest trzymany w pojedynczej komórce pamięci. Rozmiar tej pamięci jest ograniczony i z każdą pozycją komórka jest nadpisywana nowymi danymi, w związku z tym modelowanie zależności pomiędzy odległymi elementami sekwencji może być utrudnione. Natomiast liczba parametrów w wastwie LSTM rośnie kwadratowo od rozmiaru komórki, nie można więc łatwy sposób jej rozszerzać bez istotnego zwiększenia złożoności pamięciowej i obliczeniowej całego modelu.

• **Sieci Transformer** opierają się na mechanizmie *self-attention*, mającym kluczowy wpływ na złożoność obliczeniową tej architektury. Operacja ta wymaga porównania ze sobą każdej pary elementów w sekwencji, jej złożoność jest więc kwadratowa od liczby elementów. Jest to główna wada Transformerów, która w istotny sposób ogranicza możliwość przetwarzania długich sekwencji. Aktywnym obszarem badań są obecnie próby zastąpienia operacji *self-attention* przez jej aproksymację lub inne metody o niższej

Rodzaj warstwy	**Złożoność obliczeniowa**	**Operacje sekwencyjne**	**Maksymalna długość ścieżki**
LSTM	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Self-attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Splotowa	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k n)$

Tabela 2.1: Porównanie różnych aspektów złożoności warstw LSTM, *self-attention* i splotowej. (Źródło: Vaswani et al. [130])

złożoności obliczeniowej [60, 56, 5, 14]. Używane w praktyce modele charakteryzują się też większą liczbą parametrów niż modele w NLP oparte na sieciach rekurencyjnych lub splotowych. Z punktu widzenia przetwarzania współbieżnego, Transformer pozwala na niezależne wyliczenie reprezentacji każdego elementu sekwencji, ma więc duży potencjał do zrównoleglania obliczeń.

• **Sieci splotowe** mogą być dobrym wyborem z punktu widzenia efektywności obliczeniowej. Podobnie jak w przypadku sieci rekurencyjnych, ich złożoność jest liniowa od liczby elementów sekwencji. Dodatkowo oferują duże możliwości zrównoleglania obliczeń - każdy filtr splotowy może być aplikowany niezależnie, reprezentacja wektorowa na każdej pozycji zależy tylko od reprezentacji wejściowych w lokalnym sąsiedztwie elementu. Sieci splotowe mogą przetwarzać sekwencje o dowolnej długości. Podstawową ich wadą jest natomiast możliwość modelowania zależności pomiędzy elementami jedynie w obrębie kontekstu. Jak zostało wspomniane wcześniej, pojedyncza warstwa splotowa jest na ogół niewystarczająca do wygenerowania dobrych jakościowo reprezentacji elementów sekwencji, stosowane są sieci składające się z większej liczby warstw. Ponadto badania empiryczne pokazały, że rozwiązania oparte wyłącznie o warstwy splotowe osiągają na ogół gorsze wyniki od rozwiązań wykorzystujących modele rekurencyjne lub Transformery w zadaniach związanych z przetwarzaniem języka naturalnego.

Tabela 2.1 zawiera podsumowanie aspektów związanych ze złożonością obliczeniową poszczególnych architektur. Symbol n oznacza liczbę elementów w sekwencji, d wymiar reprezentacji wektorowej elementu, k rozmiar kontekstu w filtrze splotowym. W kolumnie "Złożoność obliczeniowa" przedstawiono koszt przetworzenia sekwencji o długości n. Kolumna "Operacje sekwencyjne" informuje o tym, jaka jest liczba operacji, które muszą być w danej architekturze wykonane w uporządkowanej kolejności. Wartość ta określa potencjał architektury do zrównoleglania obliczeń. Kolumna "Operacje sekwencyjne" mówi o tym, ile operacji musi zostać wykonanych do zamodelowania relacji pomiędzy elementami oddalonymi o n pozycji. Należy pamiętać, że w przypadku sieci LSTM i Transformer wystarczy do tego jedna warstwa lub pojedynczy blok enkodera. W przypadku sieci splotowej zakładamy istnienie wystarczającej liczby warstw.

2.6. Integracja modeli graficznych z sieciami neuronowymi

Modele graficzne oraz sieci neuronowe są dwiema grupami metod powszechnie stosowanymi w problemach modelowania sekwencji. Metody te mogą być stosowane niezależnie, popularną techniką stało się jednak używanie CRF jako warstwy predykcyjnej w sekwencyjnych sieciach neuronowych [50, 109, 143]. Standardowo warstwa predykcyjna w neuronowych modelach tagowania lub klasyfikacji sekwencji składa się z $|C|$ wyjść, wartość k-tego wyjścia jest obliczana za pomocą funkcji softmax według następującej formuły:

$$P(y = c_k|\boldsymbol{h_i}) = \frac{\boldsymbol{e^{h_i^T w_k}}}{\sum_{j=1}^{|C|} \boldsymbol{e^{h_i^T w_j}}} \tag{2.21}$$

gdzie $\boldsymbol{h_i}$ jest wektorem warstwy ukrytej sieci na pozycji i, c_k jest k-tą klasą, a $\boldsymbol{w_j}$ i $\boldsymbol{w_k}$ są wektorami wag dla j-tego i k-tego wyjścia.

W przypadku warstwy predykcyjnej CRF, softmax jest zastępowany transformacją liniową wektora $\boldsymbol{h_t}$ do wektora $\boldsymbol{o_t}$ przy użyciu macierzy wag $\boldsymbol{W_o}$:

$$\boldsymbol{o_i} = \boldsymbol{W_o h_i} \tag{2.22}$$

Wynikiem jest wektor $\boldsymbol{o_t}$ o liczbie wymiarów równej $|C|$, w którym k-ty element oznacza wartość przypisaną klasie c_k. Wektora tego można użyć bezpośrednio do obliczenia czynników modelu CRF, zastępując funkcję przypisującą wartość bieżącemu elementowi sekwencji funkcją wykorzystującą wektor wynikowy sieci neuronowej na pozycji i. Modyfikując formułę 2.5, otrzymujemy:

$$\Psi_i(\vec{\boldsymbol{y}}, \vec{\boldsymbol{x}}) = \exp\left(\sum_{j,k \in C} T_{j,k} \mathbb{1}_{\{y_i = j\}} \mathbb{1}_{\{y_{i-1} = k\}} + \sum_{p=1}^{|C|} o_{i,p} \mathbb{1}_{\{y_i = c_p\}}\right) \tag{2.23}$$

gdzie $\boldsymbol{o_i}$ jest wektorem wyjściowym sieci neuronowej dla i-tego elementu sekwencji, natomiast $o_{i,p}$ jest wartością p-tego wymiaru tego wektora. Sieć neuronowa oraz model CRF mogą być następnie trenowane łącznie przy użyciu optymalizatorów opartych o metody gradientowe.

2.7. Podsumowanie

W problemach związanych z przetwarzaniem danych sekwencyjnych stosowane są różne grupy rozwiązań. W pierwszej części rozdziału opisane zostały klasyczne sekwencyjne modele graficzne: ukryte modele Markowa (ang. *hidden Markov model, HMM*) oraz warunkowe pola losowe (ang. *conditional random fields, CRF*). Tego typu modele przez wiele lat były dominującym podejściem do przetwarzania sekwencji. Dopiero rozwój metod uczenia głębokiego w ostatnim dziesięcioleciu, możliwy dzięki łatwej dostępności procesorów graficznych (GPU), doprowadził do upowszechnienia metod przetwarzania sekwencji opartych na sieciach neuronowych. W drugiej części rozdziału opisano trzy najpopularniejsze architektury neuronowe stosowane dla tego typu proble-

mów: rekurencyjne sieci neuronowe, sieci Transformer oraz splotowe sieci neuronowe. Przedstawiono również wady i zalety tych architektur. Niniejsza praca skupia się na przetwarzaniu danych tekstowych. Dla sekwencji zbudowanych w oparciu o tekst stosowane są przede wszystkim sieci LSTM jako typ rekurencyjnych sieci neuronowych oraz różne warianty architektury Transformer, która pojawiła się zaledwie kilka lat temu ale w szybkim tempie zdobyła popularność w kontekście przetwarzania języka naturalnego. Rzadziej stosowanym w NLP typem sieci są sieci splotowe, których zastosowania dotyczą przede wszystkim przetwarzania obrazów. Pomimo popularyzacji sieci neuronowych, nie wyparły one zupełnie modeli graficznych. Współcześnie często używa się modeli hybrydowych, łączących architekturę neuronową z warstwą predykcyjną opartą o CRF.

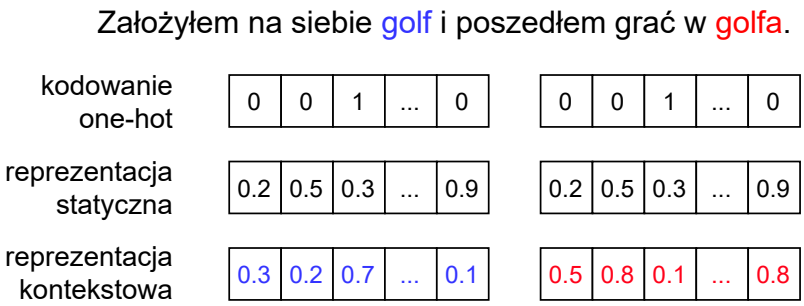
Rozdział 3

Wprowadzenie do wektorowych reprezentacji tekstu

W poprzednim rozdziale opisane zostały metody przetwarzania danych sekwencyjnych mające zastosowanie do szerokiego zakresu problemów, w których wejściem modelu może być uporządkowana lista elementów. Tego typu modele oczekują jednak danych w postaci numerycznej. W przypadku danych, które w oryginalnej formie nie są liczbami lub wektorami, niezbędne jest uprzednie przekształcenie ich do takiej postaci. Niniejsza sekcja skupia się na sposobach reprezentacji tekstu w metodach uczenia maszynowego, a w szczególności na reprezentacjach współcześnie wykorzystywanych w sieciach neuronowych. Opisane w tym podrozdziale metody stanowią element składowy modeli będących przedmiotem tej pracy jak również innych modeli wykrywania jednostek nazewniczych z literatury.

Charakterystyczną cechą przedstawionych technik jest wykorzystywanie wstępnie zdefiniowanego słownika V. Słownik ten jest zbiorem wszystkich tokenów, dla których istnieje w modelu reprezentacja wektorowa. Tokeny natomiast są elementami składowymi tekstu i w zależności od zastosowanej reprezentacji mogą być to słowa, części słów lub pojedyncze znaki. W typowym procesie wstępnego przetwarzania danych tekstowych dokonujemy segmentacji tekstu, czyli jego podziału na mniejsze jednostki. Najczęściej stosowana jest bezpośrednia segmentacja tekstu do postaci sekwencji tokenów lub segmentacja dwustopniowa, w której tekst dzielony jest najpierw na zdania a następnie na tokeny. W językach fleksyjnych, takich jak język polski, procesowi temu może towarzyszyć również ujednolicanie form słów za pomocą narzędzi do stemmingu (wyodrębnienia ze słowa nieodmiennego *rdzenia*) lub lematyzacji (sprowadzenia słowa do jego formy podstawowej) [83, s. 32]. Na podstawie otrzymanej listy tokenów, jesteśmy w stanie wygenerować listę wektorów będących numeryczną reprezentacją tekstu. Warto przy tym wspomnieć, iż w opisywanych metodach reprezentacja wejściowa, ograniczona słownikiem V, nie musi dokładnie odpowiadać wynikowej reprezentacji wektorowej. Przykładowo, istnieją modele reprezentacji tekstu operujące na sekwencjach pojedynczych znaków, które agregują grupy znaków w słowa, zatem wynikiem działania takich modeli jest lista wektorów słów a nie znaków.

Najprostszym sposobem reprezentacji tokena w postaci wektora liczb jest metoda *one-hot*. W tym podejściu, każdemu unikalnemu tokenowi przyporządkowujemy wektor o wymiarze równym rozmiarowi słownika $|V|$ a jego reprezentacja numeryczna



Rysunek 3.1: Różnice pomiędzy rodzajami reprezentacji słów na przykładzie dwóch znaczeń słowa *golf*. W przypadku kodowania one-hot oraz modeli statycznych, istnieje tylko jedna reprezentacja wektorowa danego słowa. W przypadku reprezentacji kontekstowych, model może wygenerować różne wektory w zależności od kontekstu, w którym słowo wystąpiło.

składa się z wartości 0 we wszystkich wymiarach poza wymiarem i, dla którego jest równa 1, gdzie i jest pozycją tego tokena w słowniku. Taki sposób kodowania tokenów jest obecnie rzadko stosowany ze względu na bardzo wysoką wymiarowość wektorów. Dla dużych słowników, pokrywających pełen zasób słów w wybranym języku, rozmiar słownika może wynosić nawet kilkaset tysięcy pozycji, a uwzględnienie obcojęzycznych wtrąceń i odmian w językach fleksyjnych może rozszerzyć go do kilku milionów. Dlatego też wraz z rozwojem sieci neuronowych zaczęto poszukiwać metod pozwalających na reprezentowanie tokenów w postaci bardziej kompaktowych wektorów o niższej liczbie wymiarów. Tego typu reprezentacje można podzielić na dwie główne grupy: statyczne i kontekstowe. Reprezentacje statyczne przypisują każdemu tokenowi stały wektor, niezależny od kontekstu, w którym token występuje. Można je zatem zapisać w postaci macierzy o wymiarach $|V| \times d$, gdzie d jest ustaloną wymiarowością wektora. Reprezentacje kontekstowe natomiast uwzględniają otoczenie tokena, generując dla każdego jego wystąpienia indywidualną reprezentację wektorową. W związku z tym stosowanie ich wiąże się z użyciem dodatkowego modelu, który dla każdego kontekstu przelicza na nowo wartości wektorów. Rysunek 3.1 przedstawia przykład zastosowania różnych reprezentacji dla tego samego fragmentu tekstu.

3.1. Reprezentacje statyczne

W zadaniach związanych z przetwarzaniem języka naturalnego wykorzystywanych jest wiele rodzajów wektorowych reprezentacji tekstu, między innymi wektory rzadkie bazujące bezpośrednio na współwystąpieniach słów (np. TF-IDF), modele tematyczne (ang. *topic models*), reprezentacje konstruowane przy użyciu autoencoderów. Natomiast w kontekście metod uczenia głębokiego, w szczególności w zadaniach takich jak tagowanie, wymagających osobnej reprezentacji wektorowej dla poszczególnych elementów sekwencji, najczęściej używane są gęste reprezentacje słów o relatywnie niskiej liczbie wymiarów. Reprezentacje te, w języku angielskim określane mianem *word embeddings*, konstruuje się w sposób nienadzorowany (ang. *unsupervised*) lub samonadzorowany (ang. *self-supervised*) poprzez wstępne trenowanie na dużych korpusach tekstowych.

Nazywane są również semantycznymi modelami dystrybucyjnymi, bowiem proces ich treningu opiera się na hipotezie dystrybucyjnej zakładającej, że znaczenie słów jest determinowane przez kontekst, w jakim te słowa występują [44]. Co za tym idzie, większość tego typu modeli jako podstawową jednostkę składową tekstu uznaje całe słowa a nie ich części. W efekcie przestrzeń wektorów jest pewnym przybliżeniem relacji semantycznych pomiędzy słowami występującymi w języku, a wykonywanie operacji arytmetycznych na wektorach pozwala wyciągać wnioski na przykład dotyczące podobieństwa słów.

Przykłady tego typu semantycznych reprezentacji słów pojawiały się od początku XXI wieku [6, 16, 17] natomiast spopularyzowane zostały przez model **Word2Vec** zaproponowany przez Mikolov et al. [90], pozwalający na efektywne trenowanie reprezentacji wektorowych na korpusach tekstowych o dużej skali, przekraczających kilka gigabajtów tekstu. Model Word2Vec jest prostą siecią neuronową z warstwą wejściową reprezentującą słowa ze słownika V zakodowane jako wektor one-hot, jedną warstwą ukrytą oraz warstwą wyjściową[1]. Podstawową ideą tej reprezentacji jest wykorzystanie korpusu tekstowego do wygenerowania danych treningowych na podstawie kontekstu występowania słowa. Każda próbka ucząca jest tworzona ze słowa $\boldsymbol{w}$ oraz słów leżących w jego najbliższym sąsiedztwie zwanych kontekstem C. Przykładowo, dla kontekstu o rozmiarze 4 dołączane są do niego dwa słowa znajdujące się po lewo oraz po prawo od słowa centralnego. W publikacji zaproponowano dwa warianty modelu Word2Vec. W wariancie *Skip-Gram* proces uczenia modelu polega na predykcji słów z kontekstu C na podstawie słowa $\boldsymbol{w}$. W wariancie *CBOW* (continuous bag-of-words) model uczy się przewidywać słowo $\boldsymbol{w}$ na podstawie słów z kontekstu C. Bez względu na wybór wariantu, po zakończeniu uczenia wagi warstwy ukrytej modelu reprezentują wytrenowane wektory słów, których można używać jak statycznej macierzy. W celu przyspieszenia procesu trenowania i ograniczenia liczby modyfikacji wag, Word2Vec najczęściej jest trenowany metodą *negative sampling* [91]. W tej metodzie optymalizowana funkcja dla każdej próbki treningowej zależy tylko od pewnego wybranego podzbioru słów a nie od wszystkich słów znajdujących się w słowniku. Dla każdego słowa, będącego oczekiwanym wynikiem predykcji modelu, losowanych jest k "negatywnych" słów, nie pochodzących z kontekstu tego słowa. Funkcja wymusza by wektory dla "pozytywnych" przypadków były do siebie zbliżone a dla "negatywnych" - różne. Przykładowo, dla wariantu *Skip-gram* minimalizujemy funkcję:

$$J = -\Big[\ln P(+|\boldsymbol{c_{pos}}, \boldsymbol{w}) + \sum_{i=1}^{k} \ln P(-|\boldsymbol{c_{neg,i}}, \boldsymbol{w}) \Big] \tag{3.1}$$

gdzie $\boldsymbol{w}$ jest wektorem reprezentującym słowo wejściowe, $\boldsymbol{c_{pos}}$ jest wektorem słowa pochodzącego z kontekstu $\boldsymbol{w}$ natomiast $\boldsymbol{c_{neg,i}}$ jest i-tym wektorem "negatywnym", nie pochodzącym z kontekstu $\boldsymbol{w}$. $P(+|\boldsymbol{c_{pos}}, \boldsymbol{w})$ oznacza prawdopodobieństwo, że $\boldsymbol{c_{pos}}$ należy do kontekstu $\boldsymbol{w}$. $P(-|\boldsymbol{c_{neg,i}}, \boldsymbol{w})$ oznacza prawdopodobieństwo, że $\boldsymbol{c_{neg,i}}$ nie należy do kontekstu $\boldsymbol{w}$. Wartości te są wyliczane jako funkcja sigmoid z iloczynu skalarnego

[1] W zależności od wybranego wariantu modelu liczba warstw wejściowych lub wyjściowych może być zwielokrotniona. W modelu *Skip-Gram* występuje jedna warstwa wejściowa i C wyjściowych. W modelu *CBOW* występuje C warstw wejściowych i jedna wyjściowa.

wektorów w następujący sposób:

$$P(+|\boldsymbol{c}, \boldsymbol{w}) = \sigma(\boldsymbol{c} \cdot \boldsymbol{w}) = \frac{1}{1 + e^{-\boldsymbol{c} \cdot \boldsymbol{w}}}, \quad P(-|\boldsymbol{c}, \boldsymbol{w}) = \sigma(-\boldsymbol{c} \cdot \boldsymbol{w}) = \frac{1}{1 + e^{\boldsymbol{c} \cdot \boldsymbol{w}}} \tag{3.2}$$

Innym popularnym przykładem dystrybucyjnej reprezentacji słów jest **FastText** [8, 53], będący rozszerzeniem modelu Word2Vec. Podstawową zmianą w stosunku do oryginalnego modelu jest wzbogacenie wektorów słów o wektory n-gramów znakowych. W modelu FastText, dane słowo nie jest już reprezentowane przez pojedynczy wektor ale przez sumę wektorów, w skład której wchodzi wektor reprezentujący całe słowo oraz wektory wszystkich n-gramów będących częścią tego słowa. Przykładowo, dla słowa "test" model z n-gramami o długości 3 zbuduje reprezentację sumując następujące wektory składowe: ["test", "<te", "tes", "est", "st>"], gdzie "<" i ">" oznaczają odpowiednio specjalne znaczniki początku i końca słowa. Wadą modelu FastText jest jego wyższa złożoność, natomiast główną zaletą możliwość generowania wektorów dla nowych słów, na których model nie był trenowany. Jeżeli dane słowo nie występuje ze słownika V, jego wektor jest wyliczany jako suma samych wektorów n-gramowych.

Alternatywną w stosunku do modelu Word2Vec metodą trenowania wektorów słów jest **GloVe** zaproponowany przez Pennington et al. [98]. W tej metodzie na podstawie korpusu tekstowego budujemy bezpośrednio macierz $\boldsymbol{X}$, w której element $X_{i,j}$ odpowiada częstości współwystępowania słów o indeksach i i j w słowniku V. Częstotliwość ta jest dodatkowo ważona odległością słów - wystąpienia par słów leżących bliżej siebie mają wyższą wagę od par oddzielonych innymi słowami. Ponadto autorzy metody przyjmują założenie, że docelowe podobieństwo pomiędzy wytrenowanymi wektorami, mierzone jako iloczyn skalarny dwóch wektorów, powinno wynosić $\ln(X_{i,j})$. Podczas trenowania modelu każde słowo jest reprezentowane przez dwa wektory: $\boldsymbol{w}$ - podstawowy wektor słowa (wektor wynikowy modelu), $\tilde{\boldsymbol{w}}$ - wektor kontekstowy słowa, oraz dwa parametry bias: b - podstawowy bias, $\tilde{b}$ - bias kontekstowy. Wektory są trenowane metodą najmniejszych kwadratów z następującą funkcją kosztu:

$$J = \sum_{i,j=1}^{|V|} f(X_{i,j})(\boldsymbol{w_i^T}\tilde{\boldsymbol{w}}_{\boldsymbol{j}} + b_i + \tilde{b_j} - \ln(X_{i,j}))^2 \tag{3.3}$$

gdzie $\boldsymbol{w_i}$ oraz b_i oznaczają wektor oraz bias i-tego słowa, $\tilde{\boldsymbol{w}}_{\boldsymbol{j}}$ oraz $\tilde{b_j}$ oznaczają wektor oraz bias kontekstowy j-tego słowa, $X_{i,j}$ jest miarą częstotliwości współwystępowania słów, natomiast $f(X_{i,j})$ jest dodatkową funkcją ważącą. Funkcja ważąca zaproponowana przez autorów została skonstruowana w taki sposób, aby zwiększać wagę dla par słów rzadko występujących a obniżać dla par występujących często. Zapobiega to zdominowaniu procesu uczenia przez pary o wysokich częstotliwościach współwystępowania.

3.2. Reprezentacje kontekstowe

Jednym z dominujących kierunków rozwoju metod przetwarzania języka naturalnego w ostatnich kilku latach były kontekstowe modele reprezentacji tekstu. Zastosowanie

opisanych w poprzedniej sekcji reprezentacji statycznych pozwoliło poprawić jakość predykcji dla wielu problemów w tej dziedzinie, natomiast zidentyfikowano też podstawowe wady tego typu reprezentacji wektorowych. Najważniejszą z nich jest brak możliwości modelowania wieloznaczności słów. W reprezentacjach statycznych każdy element słownika V jest mapowany na dokładnie jeden wektor. W przypadku słów, które w różnych kontekstach mogą przyjmować różne znaczenie, wektor ten reprezentuje uśrednioną pozycję tych znaczeń w przestrzeni semantycznej. Podejmowano próby budowania reprezentacji znaczeniowych, w których pozycje w słowniku V odpowiadały odrębnym znaczeniom słów a nie samym słowom, a proces mapowania słowa na wektor poprzedzany był wstępnym procesem ujednoznaczniania słów (ang. *word sense disambiguation*) [126]. Natomiast wyraźny skok jakościowy w praktycznych zastosowaniach przyniosły dopiero modele, które budują reprezentację tokena w sposób dynamiczny, za każdym razem wyliczając ją na podstawie kontekstu, w którym dany token występuje.

Zadaniem, które w naturalny sposób nadaje się do trenowania kontekstowych reprezentacji tekstu, jest modelowanie języka. Model języka to model statystyczny, którego celem jest estymowanie prawdopodobieństwa sekwencji słów w danym języku [33, s. 105]. Najbardziej rozpowszechnionym rodzajem modeli języka są modele autoregresyjne. W autoregresyjnym ujęciu, model jest trenowany na zadaniu predykcji elementu sekwencji na pozycji $t+1$ mając dane poprzedzające elementy sekwencji od 1 do t. Prawdopodobieństwo pełnej sekwencji elementów $\vec{\boldsymbol{x}}$ jest zatem dane wzorem:

$$P(\vec{\boldsymbol{x}}) = P(x_1) \prod_{t=1}^{n-1} P(x_{t+1} | \boldsymbol{x}_{\boldsymbol{1:t}}^{\rightarrow}) \tag{3.4}$$

gdzie n jest długością sekwencji a $\boldsymbol{x}_{\boldsymbol{1:t}}^{\rightarrow}$ jest podciągiem elementów sekwencji od 1 do t. Zgodnie z tymi założeniami, funkcja kosztu używana podczas trenowania modeli języka ma ogólną postać:

$$J = -\ln P(x_1) - \sum_{t=1}^{n-1} \ln P(x_{t+1} | \boldsymbol{x}_{\boldsymbol{1:t}}^{\rightarrow}) \tag{3.5}$$

W neuronowych modelach języka prawdopodobieństwo elementu sekwencji dla pozycji t jest najczęściej wyliczane przy pomocy funkcji softmax [36]. Mając dany słownik V składający się z $|V|$ tokenów, warstwa softmax zwraca rozkład prawdopodobieństwa po wszystkich słowach w słowniku. Warstwa będzie zatem składała się z $|V|$ wyjść, a wartość funkcji dla k-tego wyjścia dana jest wzorem:

$$P(y = k | \boldsymbol{h_t}) = \frac{e^{\boldsymbol{h_t^T w_k}}}{\sum_{j=1}^{|V|} e^{\boldsymbol{h_t^T w_j}}} \tag{3.6}$$

gdzie $\boldsymbol{h_t}$ jest wektorem warstwy ukrytej sieci na pozycji t, a $\boldsymbol{w_j}$ i $\boldsymbol{w_k}$ są wektorami wag dla j-tego i k-tego wyjścia.

Architektury autoregresyjnych neuronowych modeli języka najczęściej konstruowane są w oparciu o warstwy rekurencyjne, w szczególności warstwy typu LSTM, ze względu na ich sekwencyjny proces przetwarzania danych. Budowanie kontekstowych reprezentacji tekstu z tego typu modeli najczęściej polega na wstępnym wytrenowaniu sieci neuronowej na dużym korpusie tekstowym na zadaniu modelowania języka, a następnie

wyrzuceniu warsty softmax i wykorzystaniu bezpośrednio warstw ukrytych sieci, jednej lub kilku, do reprezentowania wektorów dla poszczególnych elementów sekwencji. Wektory te mogą posłużyć jako wejście do innej sieci trenowanej na zadaniu docelowym. Możliwe jest też bezpośrednie dołączenie pretrenowanej sieci do architektury docelowej i dalsze jej trenowanie na zadaniu innym niż modelowanie języka. W przeciwieństwie do reprezentacji statycznych, które w większości modelują całe słowa, w neuronowych modelach języka spotyka się wiele możliwych metod segmentacji tekstu, od słów, poprzez cząstki słów, do pojedynczych znaków.

Popularnym przykładem kontekstowej reprezentacji tekstu opartej o model autoregresyjny jest **ELMo** (embeddings from language models) [99, 100]. Jest to model języka trenowany na pełnych słowach, pozwalający również na generowanie wektorów dla nowych słów, nie pochodzących ze słownika. Pierwsze warstwy modelu odpowiadają za generowanie reprezentacji słów na podstawie sekwencji znaków wchodzących w skład słowa. Na tym etapie każde słowo jest przetwarzane osobno, jest to więc rodzaj reprezentacji bezkontekstowej, która po zakończonych trenowaniu może być zapisana w postaci statycznej macierzy. Następne dwie warstwy rekurencyjne LSTM enkodują informację o kontekście słowa. W procesie uczenia sieci trenowane są dwa bloki warstw LSTM, jeden dokonujący predykcji sekwencji od lewej do prawej, drugi w przeciwnym kierunku. Po zakończonym treningu dwukierunkowe reprezentacje ukryte są łączone i wykorzystywane jako finalna reprezentacja słowa. Autorzy sugerują aby używać średniej ważonej z wektorów warstw ukrytych: statycznej reprezentacji znakowej oraz dwóch warstw LSTM.

Innym modelem wykorzystującym sieci LSTM jest **Flair** [1, 2]. Jest to w pełni znakowy model języka, czyli uczony na zadaniu predykcji następnego znaku w tekście. Autorzy wykorzystali prostą architekturę składającą się z dwóch warstw LSTM i warstwy wyjściowej softmax, w której zarówno wejście jak i wyjście sieci reprezentuje sekwencję znaków. Sposób wyliczania reprezentacji wektorowej zmienia się jednak po zakończeniu trenowania modelu. Autorzy proponują aby model znakowy wykorzystywać do generowania reprezentacji pełnych słów w taki sposób, że wektor słowa jest wyodrębniany z warstwy ukrytej LSTM w pozycji następującej bezpośrednio po słowie. Inaczej mówiąc, w modelu Flair reprezentacja słowa odpowiada reprezentacji przerw pomiędzy znakami takich jak spacje lub inne białe znaki. Podobnie jak w przypadku modelu ELMo, w tej metodzie również trenowane są dwa modele przetwarzające sekwencję w obu kierunkach, a ostateczna reprezentacja słowa jest konkatenacją wektorów pochodzących z obu modeli.

Obecnie aktywnym obszarem badań w zakresie kontekstowych reprezentacji tekstu są metody oparte na architekturze Transformera. W przeciwieństwie do sieci rekurencyjnych, tego typu architektura w swojej oryginalnej formie nie może być wykorzystana do trenowania modeli autoregresyjnych, bowiem dane nie są w niej przetwarzane w sposób sekwencyjny - w każdym bloku enkodera wektor wyjściowy na danej pozycji jest wyliczany na podstawie wszystkich pozostałych pozycji w sekwencji. Model ma więc dostęp do całej sekwencji na każdym etapie przetwarzania. Istnieje oczywiście możliwość zmodyfikowania architektury poprzez wyłączenie części wag sieci w taki sposób, aby działała tak jak model sekwencyjny. Popularnymi przykładami takich sieci są modele z rodziny GPT (generative pre-trained transformer) [104, 105, 11], znalazły one jednak głównie zastosowanie w zadaniach związanych z generowaniem tekstu. Repre-

zentacje ogólnego przeznaczenia pochodzące z Transformera trenuje się zazwyczaj na innych zadaniach, bardziej dostosowanych do specyfiki tej architektury.

Najbardziej znaną metodą trenowania Transformera, która doczekała się licznych wariantów i modyfikacji, jest maskowane modelowanie języka (ang. *masked language modeling, MLM*) zaproponowane w modelu **BERT** (Bidirectional Encoder Representations from Transformers) przez Devlin et al. [24]. W tej metodzie model również ma za zadanie dokonać predykcji tokenów, ale nie w sposób sekwencyjny. Zamiast tego, dla każdej próbki losowo wybierany jest podzbiór pozycji w sekwencji, które model musi wypełnić. Pozycje te są maskowane, czyli zastępowane przez specjalny token *[MASK]*. W oryginalnej implementacji proces trenowania przebiega następująco. Wybieranych jest 15% pozycji w sekwencji do predykcji. Spośród tych pozycji, 80% z nich jest zamieniane na token *[MASK]*, 10% na losowy token ze słownika, a 10% pozostaje bez zmian. Według argumentacji autorów, użycie innych tokenów poza tokenem *[MASK]* jest konieczne dla wytrenowania dobrej jakości reprezentacji wektorowych, w przeciwnym wypadku model uczyłby się wyłącznie reprezentacji na zamaskowanych pozycjach, ignorując pozostałe pozycje w tekście. Po wytrenowaniu modelu, wektory wyjściowe bloków enkodera mogą być używane jako reprezentacja tekstu w zadaniach docelowych.

Trenowanie reprezentacji wektorowych z Transformerów jest obecnie aktywnym obszarem badań. W ostatnich latach powstały inne niż maskowane modelowanie języka metody uczenia. W oryginalnym modelu BERT [24] używano problemu predykcji następnego zdania (ang. *next sentence prediction*), w którym podawano modelowi na wejściu dwie sekwencje, a zadaniem modelu było określenie czy sekwencje wystąpiły obok siebie w oryginalnym korpusie. W obecnych implementacjach metoda ta na ogół nie jest już używana, udowodniono bowiem, że nie poprawia jakości wyuczonych reprezentacji [78]. Zamiast pojedynczych tokenów próbowano też maskować grupy tokenów reprezentujące całe słowa [18, 88] lub dłuższe fragmenty tekstu [52]. Używana była klasyfikacja kolejności zdań [67] i predykcja tokenów w permutowanych sekwencjach [140]. Proponowane były też metody bazujące na architekturach enkoder-dekoder [74, 106] oraz enkoder-dyskryminator [15].

3.3. Metody tokenizacji tekstu

W poprzednich sekcjach opisane zostały metody pozwalające na przekształcenie tekstu podzielonego na sekwencję tokenów do postaci sekwencji wektorów numerycznych. Niektóre z metod reprezentacji wektorowej wymuszają określony sposób podziału tekstu, inne są bardziej elastyczne i mogą być stosowane przy różnych technikach podziału. Niniejsza sekcja skupia się na samym procesie tokenizacji, czyli zamianie tekstu na sekwencję tokenów. Metody tokenizacji tekstu możemy podzielić na trzy grupy: podział na słowa, części słów lub znaki. Poniżej scharakteryzowano każdą z tych metod, natomiast w Tabeli 3.1 pokazano podstawowe różnice pomiędzy nimi.

Podział na słowa W tej metodzie tokenizacji każdy element sekwencji reprezentuje pojedyncze słowo. Prostą tokenizację na słowa można przeprowadzić w trywialny sposób, dzieląc tekst na podstawie ustalonego zestawu separatorów - białych znaków oraz znaków interpunkcyjnych. W praktyce języki składają się jednak również z trudniej-

Rodzaj podziału	Rozmiar słownika	Rozmiar sekwencji	Problem OOV	Proces tokenizacji
słowa	kilkaset tysięcy	mały	częsty	uzależniony od języka
części słów†	kilkadziesiąt tysięcy	średni	rzadki	uniwersalny
znaki†	kilkadziesiąt do kilkuset	duży	bardzo rzadki	uniwersalny

Tabela 3.1: Podsumowanie różnic pomiędzy metodami tokenizacji tekstu. † Wykorzystanie segmentacji na znaki lub części słów w językach z rozbudowanym alfabetem lub stosujących zapis bez odstępów pomiędzy słowami, takich jak chiński czy tajski, może wiązać się z dodatkowymi problemami. Informacje zawarte w powyższej tabeli nie odnoszą się do takich języków, wymagających dedykowanych rozwiązań.

szych przypadków, których nie da się podzielić w tak prosty sposób. W związku z tym do uzyskania wysokiej poprawności tokenizacji wymagane jest użycie indywidualnej metody podziału, uwzględniającej reguły składniowe danego języka. Podział na słowa wiążę się zatem z korzystaniem z dedykowanych dla danego języka narzędzi do tokenizacji. Z punktu widzenia reprezentacji wektorowych podział na słowa wydaje się najbardziej intuicyjny - słowo, w przeciwieństwie do mniejszych jednostek tekstu, posiada znaczenie semantyczne. Dlatego też w statycznych reprezentacjach wektorowych stosowana jest prawie wyłącznie taka metoda podziału.

Problemami, które pojawiają się w przypadku reprezentacji pełnych słów, jest rozmiar słownika oraz występowanie słów spoza słownika (ang. *out-of-vocabulary, OOV*). Pokrycie większości słów występujących w języku wiążę się z korzystaniem ze słownika zawierającego od kilkuset do nawet kilku milionów pozycji. W językach z bogatą fleksją można zredukować rozmiar słownika poprzez wcześniejsze sprowadzenie słów do ich form podstawowych, co zwiększa jednak złożoność procesu przetwarzania. Problem OOV dotyczy sposobu reprezentacji słów nie znajdujących się w słowniku. Najprostszą metodą jest użycie dla takiego słowa specjalnego statycznego wektora reprezentującego nieznany token. Współczesne modele reprezentacji słów stosują też bardziej wyrafinowane metody, pozwalające na oszacowanie wektora dla nowego słowa, na przykład poprzez agregację n-gramów (FastText [8]) lub znaków (ELMo [100]).

Podział na części słów Metoda polegająca na podziale tekstu na arbitralnie części, które mogą reprezentować znaki lub grupy znaków. Ten rodzaj podziału jest coraz częściej stosowany w modelach przetwarzania języka naturalnego jako rozwiązanie kompromisowe, łączące zalety reprezentacji słownej oraz znakowej. Rosnąca popularność takiej tokenizacji jest związana z pojawieniem się modeli statystycznych, które potrafią w automatyczny sposób zbudować słownik oraz metodę podziału tekstu dla danego języka na podstawie korpusu tekstu w tym języku. Przykładem takiego modelu jest BPE [117], który buduje słownik tokenów na podstawie częstotliwości ich występowania, rozpoczynając od zbioru pojedynczych znaków, następnie łącząc je i dodając do słownika najczęściej występujące cząstki. Inny popularny model, Unigram LM [64], konstruuje

podział na tokeny poprzez optymalizację prawdopodobieństwa sekwencji cząstek liczonego jako iloczyn prawdopodobieństwa poszczególnych cząstek w korpusie. Modele te pozwalają na zbudowanie słownika o z góry ustalonym rozmiarze. Najczęściej stosowane są słowniki liczące kilkadziesiąt tysięcy pozycji, czyli o rząd wielkości mniejsze niż w przypadku podziału na słowa. Znanym modelem stosującym ten podział tekstu jest oryginalny model BERT [24].

Podział na znaki Podział na znaki jest najprostszym koncepcyjnie sposobem podziału tekstu. W językach bazujących na alfabecie łacińskim nie wymaga też stosowania żadnych dodatkowych narzędzi do tokenizacji. Rozmiar słownika na ogół jest niewielki i liczy od kilkudziesięciu do kilkuset pozycji, uwzględniając litery, liczby, białe znaki, znaki interpunkcyjne i ewentualnie część znaków specjalnych. Modele trenowane przy użyciu tej reprezentacji dobrze radzą sobie z fleksją języka, lepiej od pozostałych reprezentacji obsługują też drobne błędy gramatyczne, literówki czy słowa rzadko występujące. Głównym problemem z podziałem na znaki jest jednak liczba elementów w sekwencji. Sekwencje w tej metodzie tokenizacji są istotnie dłuższe, co w przypadku sieci rekurencyjnych może powodować problemy z modelowaniem relacji pomiędzy odległymi częściami zdania, natomiast w przypadku architektury Transformera zwiększa złożoność obliczeniową, która jest kwadratowa od liczby elementów wejściowych. Dlatego też w praktyce jest to najrzadziej spotykany sposób podziału tekstu. Czasami spotykane są reprezentacje hybrydowe, łączące podział na znaki z innych sposobem reprezentowania tekstu. Przykładem takiego modelu jest Flair [1], który stosuje reprezentację znakową na wstępnym etapie przetwarzania tekstu, a następne agreguje grupy znaków do postaci wektorów słów.

3.4. Podsumowanie

Istotnym problemem w kontekście przetwarzania języka naturalnego jest przekształcenie tekstu do postaci numerycznej, która może być łatwo konsumowana przez sieci neuronowe oraz inne modele uczenia maszynowego, z jednoczesnym zachowaniem informacji semantycznej o słowach i relacjach pomiędzy nimi. W rozdziale opisane zostały dwie grupy metod reprezentowania w postaci wektorowej tokenów, czyli części składowych tekstu. Reprezentacje te są modelami uczenia maszynowego, trenowanymi w sposób nienadzorowany lub samonadzorowany na dużych korpusach tekstu w danym języku. W reprezentacjach statycznych każdemu tokenowi przypisany jest stały wektor, niezależny od kontekstu wystąpienia tego tokena. Reprezentacje kontekstowe dla każdego elementu w sekwencji generują jego indywidualny wektor, uwzględniający zarówno znaczenie semantyczne tego elementu jak i kontekst jego wystąpienia. Współcześnie reprezentacje kontekstowe wzbudzają większe zainteresowanie badaczy zajmujących się NLP i są częściej wykorzystywane w praktycznych zastosowaniach. Nie wyparły jednak one zupełnie reprezentacji statycznych, które nadal znajdują zastosowanie w niektórych problemach. W ostatniej części rozdziału poruszone zostało zagadnienie podziału tekstu na tokeny. Przedstawiono trzy postawowe spodoby dzielenia tekstu: podział na słowa, cząstki słów lub znaki. Wybór sposobu tokenizacji tekstu jest w dużej mierze uzależniony od stosowanej architektury oraz problemu, który próbujemy rozwiązać.

Każda z metod stanowi bowiem pewien kompromis pomiędzy oczekiwaną długością sekwencji, rozmiarem słownika oraz częstotliwością występowania przypadków OOV (out-of-vocabulary).

Rozdział 4

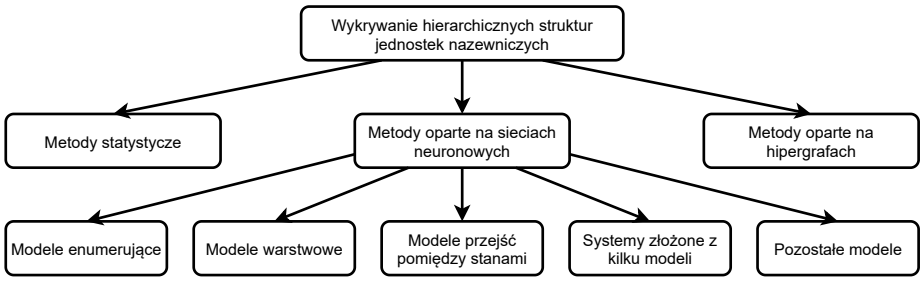
Przegląd stosowanych metod

Opisane w poprzednich rozdziałach metody modelowania danych sekwencyjnych oraz reprezentacji tekstu w postaci wektorów są wykorzystywane do budowania rozwiązań pozwalających na dokonywanie predykcji na podstawie tekstu, między innymi w problemach klasyfikacji, regresji czy tagowania sekwencji. Wynikiem działania modeli w tego typu problemach są pojedyncze klasy, wartości numeryczne lub płaskie sekwencje klas. Bezpośrednie użycie opisanych metod nie pozwala jednak na generowanie bardziej skomplikowanych wyników predykcji, na przykład zbiorów wartości, struktur drzewiastych i grafowych, czy więcej niż jednej sekwencji klas jednocześnie. Problemy, w których oczekiwanym wynikiem modelu są złożone struktury, wymagają dedykowanych rozwiązań. Przedstawione wcześniej proste modele sekwencyjne również znajdują w nich zastosowanie, najczęściej bowiem opracowywanie rozwiązań o większym stopniu złożoności polega na dostosowaniu znanych już metod do nowego problemu.

Wykrywanie hierarchicznych struktur jednostek nazewniczych, będące tematem tej rozprawy, jest jednym z takich problemów wymagających indywidualnego podejścia. Niniejszy rozdział ma na celu dokonanie przeglądu stosowanych do tej pory metod identyfikacji zagnieżdżonych jednostek nazewniczych oraz ich krytyczną analizę. Jest to pierwszy tak obszerny przegląd rozwiązań stosowanych dla tego problemu. Oryginalnym aspektem tego rozdziału jest również wprowadzenie taksonomii metod opisanych w literaturze. Taksonomia pozwala na przedstawienie aktualnego stanu badań na tym zagadnieniem w usystematyzowany sposób oraz na odróżnienie publikacji przedstawiających oryginalne ujęcie problemu od publikacji inspirowanych wcześniejszymi rozwiązaniami.

Temat automatycznego wykrywania jednostek nazewniczych jest obecny w badaniach z zakresu przetwarzania języka naturalnego już od kilkudziesięciu lat, natomiast jego bardziej złożone warianty, w tym wykrywanie struktur hierarchicznych, pojawiły się dopiero po 2000 roku. Jednym z pierwszych zbiorów danych, który zawierał zagnieżdżone jednostki nazewnicze oraz zwrócił uwagę na potrzebę ich wykrywania, był korpus abstraktów pochodzących z publikacji naukowych o tematyce medycznej i biologicznej GENIA [97]. Krótko po jego udostępnieniu pojawiły się również zbiory z oznaczonymi zagnieżdżonymi strukturami encji zawierające teksty z innych domen niż biomedyczna [25]. Stanowiło to impuls do pojawienia się pierwszych metod uwzględniających hierarchiczną strukturę jednostek nazewniczych.

Sama zagnieżdżona struktura jednostek może być reprezentowana w różny sposób



Rysunek 4.1: Proponowana taksonomia metod wykrywania hierarchicznych struktur jednostek nazewniczych.

- w postaci zbioru sklasyfikowanych fraz, w postaci drzewa, w postaci nakładających się na siebie warstw sekwencji tagów, jako ciąg zdefiniowanych operacji na tekście. W związku z tym zaproponowane w literaturze ujęcia znacznie różnią się między sobą. W celu uporządkowania aktualnego stanu wiedzy, przedstawione w niniejszym rozdziale metody podzielono na trzy grupy. W pierwszej kolejności przedstawione zostaną ujęcia statystyczne wykrywania hierarchicznych struktur jednostek nazewniczych. W tej grupie znalazły się wszystkie metody oparte na klasycznych modelach uczenia maszynowego, modele graficzne takie jak HMM i CRF lub inne metody statystyczne nie wykorzystujące sieci neuronowych. Tego typu podejścia były dominujące przez ponad dekadę, od wczesnych lat dwutysięcznych do okolic 2015 roku. W drugiej części tego rozdziału przedstawione zostaną metody, w których struktura jednostek nazewniczych jest modelowana w postaci hipergrafu. W tej grupie znalazło się jedynie kilka publikacji, natomiast prezentowane przez nie ujęcie problemu jest na tyle charakterystyczne i odrębne od pozostałych rozwiązań, że uzasadnione jest wydzielenie ich jako osobnej grupy. Popularność metod opartych na hipergrafach przypada na lata 2015-2018. Ostatnia dekada przyniosła znaczący postęp w zakresie sieci neuronowych i miało to również wpływ na metody wykrywania zagnieżdżonych jednostek nazewniczych. Trzecia grupa opisuje metody neuronowe, które zdobyły popularność po 2015 i obecnie stanowią główny kierunek badań w kontekście tego zadania. Systemy oparte na sieciach neuronowych charakteryzują się też dużą różnorodnością. W ciągu ostatnich lat zaproponowano wiele metod neuronowych prezentujących oryginalne ujęcia tego problemu. Ostatnia część tego rozdziału zawiera zatem dodatkowy podział na podgrupy podobnych do siebie ujęć neuronowych jak również opisy architektur neuronowych nie pasujących do żadnej z podgrup. Zaproponowana w tej pracy taksonomia metod identyfikacji hierarchicznych struktur jednostek nazewniczych przedstawiona została w postaci graficznej na Rysunku 4.1.

Coraz częściej zdarza się, że proponowane w literaturze rozwiązania łączą różne ujęcia wykrywania jednostek nazewniczych w jeden system. Wiele sieci neuronowych wykorzystuje modele graficzne, niektóre z metod opartych na hipergrafach generują wektorowe reprezentacje tekstu przy użyciu sieci neuronowych. Powyższy podział nie jest więc ścisły a część z przedstawionych w rozdziale metod można byłoby sklasyfikować do więcej niż jednej grupy. Zazwyczaj główna idea stojąca za danym rozwiązaniem w jednoznaczny sposób pozwala przypisać je do konkretnej grupy. Należy jednak mieć na uwadze, że ta przynależność nie wyklucza zastosowania w tym rozwiązaniu również

elementów charakterystycznych dla innego rodzaju metod.

4.1. Metody statystyczne

Jak zostało wspomniane wcześniej, pierwsze metody pozwalające na wykrywanie zagnieżdżonych jednostek nazewniczych skupiały się na zbiorze biomedycznym GENIA [97], którego użyto w zadaniu *bio-entity recognition task* realizowanym w ramach konferencji JNLPBA[1] [58]. Z punktu widzenia struktury hierarchicznej, jest to dość prosty zbiór, w którym większość zdań posiada jedną lub dwie warstwy jednostek. Prawie wszystkie zgłoszone rozwiązania zignorowały problem zagnieżdżeń, skupiając się na wykrywaniu tylko zewnętrznej warstwy. Pojawiło się tylko kilka systemów obsługujących struktury hierarchiczne. Sposób ich działania polegał na połączeniu klasycznych modeli graficznych z ręcznie zaprogramowanymi regułami heurystycznymi. W kilku metodach [118, 149, 141, 41] użyto modelu HMM bazującego na cechach wyodrębnionych z tekstu do wykrywania pierwszej warstwy jednostek. Jednostki zagnieżdżone były natomiast identyfikowane na kolejnym etapie, na podstawie manualnie skonstruowanego zestawu reguł składniowych przypominających gramatykę bezkontekstową (ang. *context-free grammar*). Podobne podejście zostało zastosowane również przez Tsai et al. [127], gdzie predykcja pierwszej warstwy encji opierała się na modelu CRF trenowanym na zestawie cech wyodrębnionych bezpośrednio z tekstu oraz ze słowników pojęć biomedycznych, a wynik tej predykcji był transformowany przez dwa dodatkowe moduły: zestaw reguł składniowych wykrywających jednostki zagnieżdżone oraz moduł korygujący wyniki predykcji na podstawie częstotliwości wystąpień słów. We wszystkich wymienionych powyżej rozwiązaniach problem zagnieżdżonych encji był obsługiwany przez dodatkowy proces nie wykorzystujący metod uczenia maszynowego. Zarówno ten proces jak i zestaw atrybutów użytych podczas uczenia modeli były ściśle dostosowane do jednego konkretnego zbioru danych. W związku z tym powyższe metody nie mają zastosowania do innych zbiorów niż GENIA. Bardziej uniwersalna wydaje się metoda zaproponowana przez Gu [39], w której dwa modele SVM (support-vector machine) były trenowane do wykrywania najbardziej zewnętrznej oraz wewnętrznej warstwy encji.

Ważną z punktu widzenia problemu identyfikacji zagnieżdżonych jednostek nazewniczych publikacją jest Alex et al. [3]. Autorzy przedstawili w niej trzy techniki rozwiązywania tego problemu, które stanowiły inspirację dla przyszłych metod i do dziś są używane w niektórych badaniach jako rozwiązania bazowe (ang. *baselines*), stanowiące podstawę do porównań z wynikami bardziej zaawansowanych modeli. Dwie spośród opisanych technik proponują podział struktury jednostek na warstwy i trenowanie osobnych modeli do wykrywania każdej z warstw. Podstawowym modelem wykorzystywanym przez autorów jest model CRF, który jest trenowany na zestawie atrybutów wyodrębnionych z tekstu.

Pierwszą zaproponowaną metodą jest *layering*. Polega ona na podziale hierarchicznej struktury jednostek nazewniczych na warstwy w jednym z dwóch wariantów: *outside-in* lub *inside-out*. W *outside-in* jednostki są układane w warstwy rozpoczy-

[1] Joint Workshop on Natural Language Processing in Biomedicine and its Application: `https://dl.acm.org/conference/jnlpba`

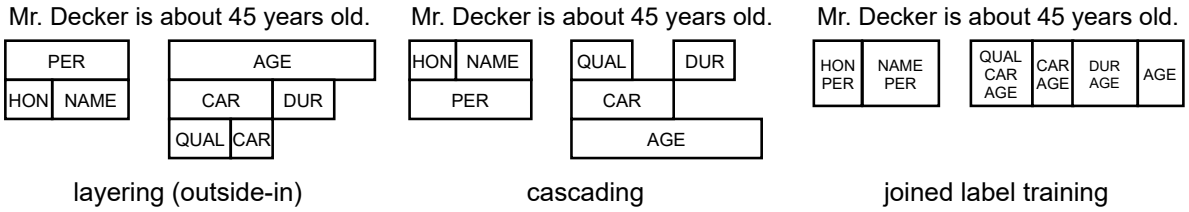
nając od bardziej ogólnych, odpowiadających dłuższym frazom w tekście, a kończąc na bardziej szczegółowych. W *inside-out* kolejność jest odwrotna - jednostki bardziej szczegółowe trafiają do pierwszych warstw, a ogólne do ostatnich. Działanie obu wariantów jest podobne. Kolejnym krokiem jest wytrenowanie osobnych modeli CRF, których zadaniem jest predykcja jednostek tylko dla jednej warstwy. Wejściem modelu odpowiadającego pierwszej warstwie jest zestaw atrybutów tekstowych, modele dla pozostałych warstw dodatkowo otrzymują na wejściu wyniki predykcji modeli z poprzedniej warstwy, dzięki czemu możliwe jest przekazywanie informacji pomiędzy modelami w procesie predykcji. Liczba warstw, a tym samym liczba modeli koniecznych do nauczenia, jest równa maksymalnej głębokości zagnieżdżenia jednostek nazewniczych w zbiorze danych.

Innym sposobem podziału struktury na warstwy jest *cascading*. W tej technice podział jest dokonywany nie w oparciu o długość jednostki ale w oparciu o jej przynależność do określonej klasy lub zestawu klas. Pojedyncza warstwa *cascadingu* zawiera tylko jednostki należące do ustalonych typów. Podobnie jak w przypadku techniki opisanej powyżej, tutaj również należy ustalić kolejność warstw i wytrenować osobne modele CRF dla każdej z warstw, przekazując następnemu modelowi wyniki predykcji poprzedniego modelu. Potencjalnym problemem z tym podejściem jest brak jednoznacznego algorytmu podziału oraz szeregowania warstw. W przypadku *layeringu* mamy do wyboru dwa kierunki reprezentacji warstw, z których każdy w prosty sposób determinuje przynależność danej jednostki do warstwy. W *cascadingu* liczba możliwych sposobów podzielenia zbioru klas na podzbiory odpowiadające warstwom, a następnie uszeregowania tych warstw w kolejności, jest bardzo duża. Różne sposoby podziału mogą mieć znaczny wpływ na jakość predykcji modelu, konieczne może być więc przetestowanie wielu możliwości, co jest kosztowne obliczeniowo. Dodatkowym ograniczeniem tej reprezentacji jest brak możliwości reprezentowania kilku zagnieżdżonych jednostek nazewniczych jeżeli są one tego samego typu.

Ostatnim z zaproponowanych rozwiązań jest *joined label training*. Metoda ta polega na sprowadzeniu hierarchicznej struktury jednostek do postaci płaskiej poprzez rozszerzenie zbioru klas. Z nakładających się typów jednostek tworzona jest nowa klasa będąca reprezentacją wszystkich tych typów. W tym podejściu trenowany jest tylko jeden model CRF. Technika ta może być jednak stosowana tylko do prostych problemów z niewielką głębokością zagnieżdżeń - w przypadku skomplikowanej struktury hierarchicznej liczba możliwych kombinacji typów może okazać się zbyt duża. Poza tym istnieje ryzyko, że niektóre klasy nie wystąpią w ogóle w zbiorze treningowym lub wystąpią tylko kilka razy, co w istotny sposób ogranicza możliwość nauczenia modelu wykrywania wszystkich połączonych kombinacji klas.

Na Rysunku 4.2 pokazano zdanie z trzema warstwami jednostek nazewniczych oraz jego przykładową reprezentację w każdej z metod opisanych w Alex et al. [3].

Inne ujęcie problemu zostało zaproponowane w Finkel and Manning [29]. Autorzy traktują wykrywanie zagnieżdżonej struktury jednostek nazewniczych jako problem parsowania, przedstawiając zdanie w postaci drzewa, w którym poszczególne wierzchołki reprezentują jednostki nazewnicze odpowiadające frazom z tego zdania. Reprezentacja taka jest bardzo podobna do rozkładu zdania wykorzystywanego w metodach analizy składniowej (ang. *constituency parsing*). W związku z tym autorzy dostosowują używany już wcześniej w analizie składniowej parser [30] bazujący na modelu CRF. W



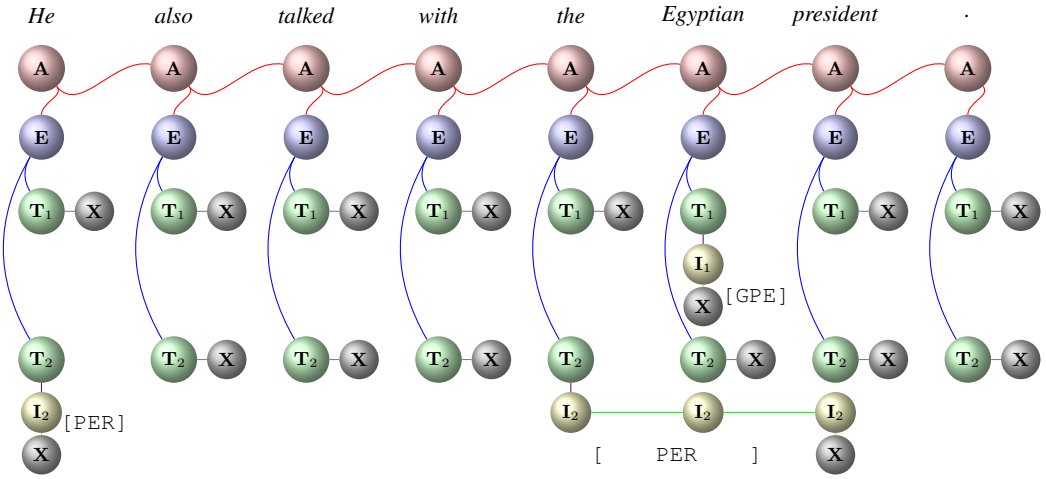
Rysunek 4.2: Przykład reprezentacji hierarchicznej struktury jednostek nazewniczych za pomocą trzech metod opisanych w Alex et al. [3]: *layering* w wariancie *outside-in*, *cascading* z podziałem klas na trzy podzbiory reprezentujące kolejne warstwy ($\{HON, NAME, QUAL, DUR\}, \{PER, CAR\}, \{AGE\}$), *joined label training*.

przeciwieństwie do sekwencyjnego modelu *linear-chain CRF*, w którym czynniki Ψ są zdefiniowane jako funkcje działające na podzbiorze elementów sekwencji, w opisywanym parserze czynniki operują na cechach wyodrębnionych z podstruktur drzewa utworzonego na podstawie jednostek nazewniczych. Główną wadą zaproponowanego algorytmu, na co uwagę zwracają sami autorzy, jest jego wysoka złożoność obliczeniowa wynosząca $O(n^3)$, gdzie n jest liczbą elementów sekwencji. Wykrywanie jednostek nazewniczych jako parsowanie zdania było też badane przez Zhang et al. [144]. Autorzy skupili się na języku chińskim, przekształcając sekwencje z przypisanymi im jednostkami do postaci rozkładu zdania, a następnie przetestowali kilka dostępnych parserów do analizy składniowej.

Obecnie metody oparte o modele graficzne lub inne klasyczne modele uczenia maszynowego są rzadko wykorzystywane samodzielnie w zadaniach wykrywania hierarchicznych struktur jednostek nazewniczych, najczęściej stanowią one komponent bardziej złożonych systemów neuronowych. Współczesne systemy neuronowe osiągają lepsze wyniki predykcji dla publicznie dostępnych zbiorów danych. Jest to spowodowane między innymi zastosowaniem wytrenowanych w sposób samonadzorowany reprezentacji tekstu, które dla tego problemu okazały się bardziej efektywne i uniwersalne od klasycznego podejścia do konstruowania cech dla modeli uczenia maszynowego, polegającego na manualnym projektowaniu zestawu atrybutów w oparciu o tekst, często przy wykorzystaniu dodatkowej wiedzy domenowej ze słowników czy baz danych.

4.2. Metody oparte na hipergrafach

W 2015 roku, w publikacji Lu and Roth [80] zaproponowane zostało nowe ujęcie problemu wykrywania hierarchicznych struktur jednostek nazewniczych, w którym struktura jednostek reprezentowana jest w postaci hipergrafu. Rozwiązanie to wzbudziło zainteresowanie, w wyniku którego przez kolejne lata powstało kilka modyfikacji oryginalnego modelu, różniących się przede wszystkim sposobem generowania wyniku lub użytymi metodami reprezentacji cech. Główną ideą tego ujęcia jest zdefiniowanie pewnej formalnej struktury, którą można przedstawić graficznie w postaci hipergrafu. W tak zdefiniowanej strukturze istnieją różne typu wierzchołków odpowiadające określonym atrybutom jednostek nazewniczych takim jak np. pozycje brzegowe wyznaczające granice jednostki, kategoria jednostki czy słowa należące do danej jednostki. Ponadto



Rysunek 4.3: Przykładowy hipergraf opisujący zagnieżdżoną strukturę jednostek nazewniczych dla zdania "He also talked with the Egyptian president." oraz zbioru dwóch możliwych klas jednostek $\{GPE, PER\}$. (Źródło: Lu and Roth [80])

struktura narzuca również określoną relację pomiędzy wierzchołkami - każdy wierzchołek może mieć co najwyżej jeden wierzchołek nadrzędny (*rodzica*) oraz listę wierzchołków podrzędnych (*dzieci*), z którymi połączony jest hiperkrawędzią. W stosowanych w tego typu metodach hipergrafach relacja ta na ogół służy uszczegółowieniu pewnych atrybutów jednostki nazewniczej. Przykładowo, może istnieć pewien wierzchołek E_k o następującej definicji: *jeżeli wierzchołek istnieje, to istnieje również co najmniej jedna jednostka nazewnicza rozpoczynająca się od k-tego słowa w zdaniu.* Potencjalnymi dziećmi tego wierzchołka mogą być wierzchołki $\{T_{k,c_1}, T_{k,c_2}, ..., T_{k,c_m}\}$, gdzie c_i jest klasą ze zbioru wszystkich możliwych kategorii encji C, a definicja wierzchołka T_{k,c_i} jest następująca: *jeżeli wierzchołek istnieje, to istnieje również co najmniej jedna jednostka nazewnicza należąca do klasy c_i, która rozpoczyna się od k-tego słowa w zdaniu.* Ponieważ rodzic określa bardziej ogólne cechy jednostki nazewniczej, to w poprawnie skonstruowanym hipergrafie opisującym zagnieżdżoną strukturę jednostek żadne z dzieci nie może istnieć jeżeli nie istnieje rodzic.

W opisywanym ujęciu dla każdej sekwencji słów można skonstruować wiele różnych hipergrafów opisujących różne struktury jednostek nazewniczych. Natomiast każdy poprawny - w rozumieniu tego ujęcia - hipergraf w jednoznaczny sposób definiuje określony zbiór jednostek wraz z ich kategoriami. Celem modelu jest zatem znalezienie optymalnego hipergrafu opisującego najbardziej prawdopodobny zbiór jednostek nazewniczych dla każdej sekwencji. Autorzy do rozwiązania tego problemu proponują model oparty na CRF, jednak w odróżnieniu od *linear-chain CRF* ich model buduje nie liniową sekwencję tagów ale strukturę grafową ograniczoną poprzez zdefiniowane reguły konstrukcji wierzchołków oraz możliwych relacji pomiędzy nimi.

Przykład takiej struktury pochodzący z publikacji Lu and Roth [80] pokazano na Rysunku 4.3. W oryginalnym modelu zdefiniowanych zostało pięć typów wierzchołków. Wartość każdego wierzchołka wyliczana była w oparciu o zaproponowane przez autorów funkcje cech. Funkcje te korzystały zarówno z cech pochodzących bezpośrednio z tekstu

takich jak słowa, n-gramy czy części mowy występujące w lokalnym kontekście pozycji sekwencji przypisanej do danego wierzchołka, jak i z cech określonych na podstawie sąsiedztwa z innymi wierzchołkami w hipergrafie. Podobny model oraz zestaw cech został zastosowany również przez Muis and Lu [94]. Nowe aspekty zaproponowanej przez nich metody polegały przede wszystkim na modelowaniu pewnych właściwości jednostek nazewniczych nie tylko na poziomie poszczególnych słów, ale także przerw pomiędzy słowami, które w strukturze grafowej mogły być reprezentowane poprzez hiperkrawędzie. W opisanym przez nich hipergrafie występowały więc nie tylko różne typy wierzchołków ale też połączeń pomiędzy poszczególnymi wierzchołkami. W Wang and Lu [132] przedstawiono kolejny model inspirowany ujęciem hipegrafowym. Autorzy w dużej mierze opierają się na oryginalnym modelu Lu and Roth [80], łącząc go z sieciami neuronowymi. W opisanej przez nich metodzie manualnie definiowane cechy zostały zastąpione przez reprezentacje wektorowe konstruowane w oparciu o dwukierunkowe warstwy LSTM działające na trzech poziomach granularności: na poziomie znaków, słów oraz fraz.

Inny rodzaj reprezentacji opartej na hipergrafie został zaproponowany przez Katiyar and Cardie [57]. Porównując do metod opisanych powyżej, jest to prostsza struktura bazująca na schemacie BILOU wykorzystywanym w klasycznym tagowaniu sekwencji. W tej reprezentacji każdy wierzchołek odpowiada określonemu tagowi ze schematu BILOU przypisanemu do słowa na danej pozycji w sekwencji. W przypadku słów, które wchodzą w skład zagnieżdżonej struktury jednostek, występuje kilka wierzchołków odpowiadających różnym tagom dla jednej pozycji. Pojedynczy wierzchołek dla pozycji k może się więc "rozgałęzić" na kilka wierzchołków dla pozycji $k+1$, a następnie ponownie połączyć w jeden wierzchołek dla dalszych pozycji. Autorzy proponują model neuronowy w architekturze enkoder-dekoder, w którym część enkodera składa się pewnej liczby dwukierunkowych warstw LSTM, natomiast dekoder składa się z pojedynczej jednokierunkowej warstwy LSTM połączonej z warstwą predykcyjną zwracającą dla każdego słowa jedną lub kilka klas, na podstawie których generowane są reprezentacje wektorowe wierzchołków.

Metody oparte na hipergrafach oferują uniwersalną reprezentację hierarchicznych struktur jednostek nazewniczych, która nie narzuca ograniczeń na głębokość zagnieżdżenia, długość jednostki czy inne cechy specyficzne dla poszczególnych zbiorów danych. Niestety, złożoność obliczeniowa tego typu rozwiązań jest uzależniona od poziomu skomplikowania samej struktury jednostek i rośnie wraz ze wzrostem liczby klas oraz długości sekwencji. Prowadzi to też do zwiększania się liczby wierzchołków w hipergrafie, a w tego typu modelach dla każdego wierzchołka niezbędne jest wyliczenie jego indywidualnego zestawu cech. Dla złożonych problemów identyfikacji jednostek liczba wszystkich elementów, dla których konieczne jest wyliczenie reprezentacji, może być więc znacząco wyższa niż w metodach operujących na strukturach sekwencyjnych, w których łączna liczba takich reprezentacji jest uzależniona przede wszystkim od długości sekwencji.

4.3. Metody oparte na sieciach neuronowych

Sieci neuronowe są wykorzystywane w problemach wykrywania hierarchicznych struktur jednostek nazewniczych od niedawna, jednak wzrastająca popularność tego zagadnienia sprawiła, że w ciągu ostatnich lat powstało wiele rozwiązań tego typu. Większość z tych systemów oparta jest na pewnych wspólnych elementach takich jak wykorzystywanie pretrenowanych wektorowych reprezentacji tekstu. Jednocześnie występuje też wiele różnic utrudniających jednoznaczną kategoryzację metod neuronowych. W niniejszej sekcji wyodrębnione zostaną pewne charakterystyczne grupy ujęć powtarzających się w literaturze, opisane zostaną również modele, których nie da się przypisać do żadnej z tych grup.

Modele enumerujące Prostym pomysłem na identyfikację nakładających się jednostek nazewniczych w zdaniu jest zbadanie wszystkich możliwych podciągów słów wchodzących w skład zdania i rozpatrywanie każdego takiego podciągu niezależnie, to znaczy określenie czy podciąg reprezentuje jednostkę nazewniczą oraz przypisanie go do odpowiedniej kategorii jednostek. Jak łatwo zauważyć, liczba takich podciągów rośnie bardzo szybko wraz ze wzrostem długości sekwencji i dla n-elementowej sekwencji wynosi $(n^2 + n)/2$. W celu ograniczenia złożoności obliczeniowej, możemy ustalić pewną maksymalną długość pociągu k i rozpatrywać tylko frazy zawierające od 1 do k słów. Tego typu podejście do wykrywania jednostek nazewniczych zostało zaproponowane przez Xu et al. [139]. Autorzy zaimplementowali model neuronowy, który dla każdej rozpatrywanej frazy konstruuje trzy reprezentacje wektorowe - reprezentację samej frazy oraz reprezentacje wszystkich słów znajdujących się na prawo i na lewo od tej frazy. Wektory mają stałą liczbę wymiarów, niezależną od liczby słów w poszczególnych fragmentach zdania. W oparciu o nie budowana jest prosta kilkuwarstwowa sieć perceptronowa, która klasyfikuje każdy podciąg do odpowiedniej kategorii jednostek nazewniczych lub do kategorii oznaczającej, że dana fraza nie jest jednostką nazewniczą. Podobny model analizujący podsekwencje opisano również w Sohrab and Miwa [121]. Podstawową różnicą jest sposób reprezentowania fraz - reprezentacja wektorowa w tej architekturze jest konstruowana w oparciu o dwukierunkowe sieci LSTM, prawy i lewy kontekst zdania jest więc już uwzględniony w stanie ukrytym sieci. Główną wadą tego typu rozwiązań jest ich koszt obliczeniowy, nawet przy ograniczeniu maksymalnej długości frazy. Ponadto wprowadzanie tego typu ograniczeń do modelu powoduje, że może nie być on w stanie wykryć długich jednostek nazewniczych.

Modele warstwowe Działanie tej grupy modeli opiera się na wielowarstwowych sieciach neuronowych, w których każda warstwa lub grupa warstw odpowiada za predykcję podzbioru jednostek nazewniczych. W zakresie podziału hierarchicznej struktury na warstwy metody są podobne do metod iteracyjnych będących tematem niniejszej pracy. Główna różnica jest taka, że w przypadku modeli warstwowych architektura sieci jest dostosowana do identyfikacji wszystkich jednostek w pojedynczym przebiegu sieci. Wiąże się to z powielaniem w architekturze pewnych bloków obliczeniowych mających za zadanie wykrywać pojedynczą warstwę jednostek nazewniczych. Przykładem takiego podejścia jest model zaproponowany przez Ju et al. [54]. Jest to sieć

neuronowa składająca się z pewnej liczby warstw LSTM, które wykrywają jednostki nazewnicze uszeregowane w strukturze *inside-out*. Każda warstwa LSTM jest połączona z indywidualną warstwą predykcyjną CRF, generującą jednostki dla danego poziomu zagnieżdżenia. Dodatkowo każda wykryta jednostka nazewnicza, reprezentowana przez pewną liczbę wektorów odpowiadających elementom sekwencji, jest łączona w pojedynczy wektor poprzez wyliczenia średniej z wektorów składowych. W efekcie liczba elementów sekwencji w wyższych warstwach jest redukowana, ponieważ pojedyncze wektory mogą odpowiadać nie tylko słowom, ale też frazom złożonym z kilku słów. Podobna idea została też wykorzystana przez Fisher and Vlachos [31]. Autorzy proponują architekturę, w której poszczególne bloki składające się z par warstw zwanych *structure layer* i *update layer* identyfikują jednostki nazewnicze dla różnych poziomów zagnieżdżenia, a pojedyncza warstwa predykcyjna dokonuje ich klasyfikacji. Modele warstwowe można traktować jako rozwinięcie podejścia *layering* z Alex et al. [3]. Natomiast o ile w oryginalnej metodzie trenowana była pewna liczba osobnych modeli odpowiadających poszczególnym warstwom jednostek, o tyle opisane powyżej metody łączą je w pojedynczy złożony model neuronowy. Potencjalnym problemem w tego typu modelach jest konieczność dostosowywania ich do charakterystyki zbioru danych. Zbiory o dużej głębokości zagnieżdżeń będą potrzebowały sieci z większą liczbą warstw i parametrów, wymagających dłuższego trenowania.

Modele przejść pomiędzy stanami Ta grupa metod inspirowana jest parserami typu przesunięcie-redukcja (ang. *shift-reduce*), wykorzystywanymi między innymi w problemach analizy składniowej oraz zależnościowej zdań. Główna idea modeli opiera się na przechowywaniu stanu modelu w strukturach danych zwanych stosem oraz buforem. Bufor zawiera reprezentację sekwencji, stos natomiast jest wynikową reprezentacją hierarchicznej struktury jednostek nazewniczych. Przed rozpoczęciem predykcji bufor zawiera całą sekwencję elementów reprezentujących zdanie. Predykcja składa się z sekwencji kroków, a w każdym kroku model wykonuje jedną dopuszczalnych operacji w oparciu o aktualny stan stosu oraz bufora. Operacje parsera polegają na modyfikacji bufora lub stosu. Przykładowa operacja może polegać na pobraniu elementu z bufora i odłożeniu go na stos lub oznaczeniu dwóch sąsiadujących ze sobą elementów na stosie jako nowej jednostki nazewniczej przynależącej do konkretnej klasy. Po zakończeniu działania modelu na danej sekwencji, stos zawiera reprezentację zidentyfikowanych jednostek zapisaną w postaci struktury drzewiastej. Wang et al. [133] oraz Marinho et al. [87] zaproponowali modele neuronowe oparte na tym ujęciu problemu identyfikacji jednostek. Reprezentacja bufora oraz stosu w obu modelach jest podobna, autorzy wykorzystują sieci LSTM oraz wytrenowane statyczne reprezentacje wektorowe słów do zbudowania stanu modelu. Główna różnica polega na sposobie zapisu zagnieżdżonej struktury encji w postaci drzew. W publikacjach zaproponowano różne wynikowe reprezentacje, a co za tym idzie, różne zestawy operacji dopuszczalnych dla parsera. W tej grupie metod złożoność obliczeniowa również jest problemem. Co prawda liczba poszczególnych kroków predykcji nie musi być duża - jest ona uzależniona od długości sekwencji oraz złożoności samej struktury jednostek nazewniczych. Natomiast koszt pojedynczego kroku jest wysoki, wymaga bowiem przeliczenia nowych reprezentacji bufora oraz stosu.

Systemy złożone z kilku modeli Jednym z klasycznych podejść do złożonych problemów jest ich dekompozycja na prostsze podproblemy i rozwiązanie każdego z nich niezależnie. Tego typu metodyka stosowana jest również w przypadku wykrywania hierarchicznych struktur jednostek nazewniczych. Problem identyfikacji struktury jednostek może być rozbity na dwa lub więcej podzadań, a zamiast jednej sieci neuronowej trenowanych jest kilka sieci wyspecjalizowanych do rozwiązywania każdego z tych podzadań. Lin et al. [76] jest przykładem systemu złożonego z dwóch modeli, w którym pierwszy (*anchor detector*) ma za zadanie zidentyfikować "główne" (ang. *head word, anchor word*) słowo każdej jednostki nazewniczej oraz przypisać proponowaną klasę dla tego słowa, natomiast drugi (*region recognizer*) dla każdego zidentyfikowanego w poprzednim kroku słowa określa jego kontekst reprezentujący całą jednostkę nazewniczą. Dwa modele są również wykorzystywane w Xia et al. [137]. W tym podejściu pierwszy model odpowiada za identyfikację w tekście fraz, które są jednostkami nazewniczymi. Reprezentacja każdej z tych fraz jest następnie przekazywana jako wejście do kolejnego modelu, który klasyfikuje jednostki od odpowiadających im kategorii. Jeszcze inny podział problemu identyfikacji jednostek został zaproponowany w Chen et al. [12]. W tym przypadku pierwszy model identyfikuje oraz klasyfikuje słowa graniczne (ang. *boundary words*) jednostek, czyli pierwsze oraz ostatnie słowo każdej jednostki. Z pozycji granicznych generowana jest następnie lista fraz będących kandydatami na jednostki nazewnicze. Kolejny model dokonuje klasyfikacji tych fraz, określając czy reprezentują one prawidłowe jednostki. Potencjalnym problemem opisanych powyżej systemów jest ich stopień skomplikowania. Metody oparte na kilku modelach zawierają więcej hiperparametrów, dostosowywanie ich do nowych domen lub języków może być zatem trudniejsze niż w przypadku pojedynczych modeli. Trudniejsze jest też zrozumienie działania takiego systemu, co może być istotne z punktu widzenia wyjaśnialności predykcji systemu oraz identyfikacji źródeł potencjalnych błędów predykcji.

Poza opisanymi grupami modeli neuronowych istnieją również pojedyncze rozwiązania nie pasujące do żadnej z grup. Przykładowo, Zheng et al. [148] wykorzystują uczenie modelu na kilku zadaniach (ang. *multi-task learning*). Zaproponowany przez nich model składa się z dwukierunkowej warstwy LSTM oraz dwóch warstw predykcyjnych, z których jedna odpowiada za wykrywanie pozycji brzegowych jednostek nazewniczych, natomiast druga klasyfikuje jednostki wyodrębnione na podstawie tych pozycji. Stosowany jest więc tutaj podział na podproblemy charakterystyczny dla systemów złożonych z kilku modeli, natomiast cały proces predykcji jest obsługiwany przez pojedynczy model.

Trudnym do sklasyfikowania rozwiązaniem z uwagi na złożoność architektury jest model opisany w Sun et al. [122]. Autorzy proponują sieć neuronową złożoną dwóch jednokierunkowych modułów o architekturze Transformera [130] odpowiedzialnych za konstruowanie reprezentacji wektorowych dla elementów sekwencji. Wektory te następnie są przekształcane przez warstwę splotową, po której następuje zaproponowana specjalnie dla tego problemu operacja grupowania (ang. *pooling*), generująca wektory o stałej liczbie wymiarów reprezentujące frazy z oryginalnej sekwencji. Wektory fraz są ostatecznie przekazywane do warstwy predykcyjnej, która określa czy dana fraza jest jednostką nazewniczą oraz przypisuje jej klasę.

Oryginalne podejście do wykrywania zagnieżdżonych struktur jednostek nazewniczych zostało zaproponowane przez Shibuya and Hovy [119]. Autorzy wykorzystują pretrenowany model BERT [24] z warstwą wyjściową CRF - architekturę, która jest obecnie powszechnie stosowana w typowych zadaniach płaskiego tagowania sekwencji. W celu obsłużenia kilku warstw jednostek nazewniczych nie proponują modyfikacji w samej architekturze sieci, ale w algorytmie dekodowania Viterbiego, służącym do znajdowania optymalnej sekwencji tagów. Zagnieżdżone jednostki są identyfikowane poprzez wielokrotne wykonanie dekodowania. Pierwsza warstwa jednostek jest identyfikowana w sposób identyczny jak w problemach płaskiej predykcji. Następnie dla każdej wykrytej w ten sposób jednostki algorytm dekodowania jest ponawiany w celu znalezienie drugiej w kolejności sekwencji tagów, czyli sekwencji o drugiej najwyższej wartości po sekwencji optymalnej. Proces ten jest powtarzany do osiągnięcia określonego poziomu głębokości zagnieżdżenia.

4.4. Podsumowanie

W rozdziale przeprowadzono analizę istniejących metod wykrywania hierarchicznych struktur jednostek nazewniczych. Wprowadzona została nowa taksonomia stosowanych ujęć, z podziałem na trzy główne grupy: metody statystyczne, metody oparte na hipergrafach oraz metody oparte na sieciach neuronowych. Spośród wymienionych grup, najliczniej reprezentowane są metody neuronowe. Współcześnie w badaniach dotyczących identyfikacji jednostek nazewniczych sieci neuronowe są dominującym podejściem. Problem identyfikacji zagnieżdżonych struktur jednostek cieszy się obecnie znacznie większym zainteresowaniem niż na początku XXI wieku, kiedy pojawiały się pierwsze publikacje na ten temat, co ma przełożenie również na liczbę i różnorodność architektur neuronowych, które zostały zaprezentowane w ciągu ostatnich kilku lat.

Rozdział 5

Metody iteracyjne

W poprzednich rozdziałach opisane zostały metody i narzędzia stosowane do modelowania danych sekwencyjnych w obszarze przetwarzania języka naturalnego. Przeprowadzony został również przegląd różnych ujęć problemu wykrywania hierarchicznych struktur jednostek nazewniczych oraz modeli rozwiązujących ten problem. Niniejszy rozdział rozpoczyna omówienie ujęcia iteracyjnego, będącego zasadniczym tematem tej rozprawy oraz oryginalnym wkładem autora w problem rozpoznawania jednostek nazewniczych. W pierwszej części rozdziału przedstawione zostanie wprowadzenie do iteracyjnego tagowania sekwencji, które stanowi rozszerzenie klasycznego tagowania sekwencji do problemów identyfikacji struktur hierarchicznych. W dalszej części rozdziału omówione zostaną praktyczne aspekty predykcji iteracyjnej. Zaproponowana zostanie konkretna architektura neuronowa wykorzystująca ujęcie iteracyjne, wprowadzone zostanie pojęcie wektora stanu oraz opisany zostanie sposób jego kodowania i przekazywania w procesie predykcji iteracyjnej. Ponadto opisany zostanie sposób konstrukcji zbioru danych oraz trenowania tego typu modeli.

5.1. Tagowanie sekwencji w ujęciu iteracyjnym

W podrozdziale 1.3 przedstawiona została formalna definicja problemu wykrywania jednostek nazewniczych w metodzie tagowania sekwencji. Oryginalne sformułowanie tego problemu zakłada, że celem tagowania sekwencji jest znalezienie funkcji f przekształcającej sekwencję tokenów $\vec{\boldsymbol{x}}$ reprezentujących segment tekstu na sekwencję tagów $\vec{\boldsymbol{y}}$, w której każdy tag przypisany jest do tokena znajdującego się na tej samej pozycji. Przy założeniu, że element sekwencji wynikowej może zawierać tylko jeden tag, w zadaniach wymagających predykcji zagnieżdżonych struktur jednostek nazewniczych nie jest możliwe zapisanie pełnej struktury w postaci pojedynczej sekwencji tagów $\vec{\boldsymbol{y}}$. Wynik predykcji modelu jest zapisywany w postaci bardziej złożonych struktur, różniących się z zależności od zastosowanego podejścia.

Ujęcie iteracyjne jest rozszerzeniem klasycznej definicji problemu tagowania sekwencji. Zakładamy w nim, że hierarchiczną strukturę jednostek nazewniczych można podzielić na warstwy w taki sposób, aby każdą warstwę dało się zapisać postaci płaskiej sekwencji tagów, analogicznie jak w klasycznym problemie tagowania sekwencji. Iteracyjne tagowanie sekwencji polega na wykonaniu k etapów predykcji, gdzie k jest liczbą

warstw (głębokością struktury). Każdy etap predykcji jest podobny do predykcji płaskiej. Zmiana w stosunku do klasycznej metody tagowania polega na wprowadzeniu stanu modelu, oznaczonego jako S_i, w i-tym kroku predykcji. W ujęciu iteracyjnym naszym zadaniem jest zatem znalezienie funkcji:

$$f : (\vec{\boldsymbol{x}}, S_{i-1}) \rightarrow \vec{\boldsymbol{y}}^{(\boldsymbol{i})} \tag{5.1}$$

gdzie $\vec{\boldsymbol{x}}$ jest wektorem tokenów, S_{i-1} jest stanem modelu w poprzednim kroku a $\vec{\boldsymbol{y}}^{(\boldsymbol{i})}$ jest i-tą sekwencją tagów, reprezentującą pewien podzbiór jednostek nazewniczych.

Stan modelu S_i jest zbiorem wszystkich jednostek nazewniczych, które zostały zostały identyfikowane w iteracjach od 1 do i:

$$S_i = \{m_1, m_2, ..., m_l\} \tag{5.2}$$

gdzie m_j jest j-tą zidentyfikowaną jednostką nazewniczą a l jest całkowitą liczbą wykrytych jednostek. Pojedynczą jednostkę nazewniczą przypisaną do podciągu sekwencji $\vec{\boldsymbol{x}}$, obejmującą elementy od pozycji j do pozycji $j+k-1$, będziemy reprezentować jako parę:

$$m = ([x_j, x_{j+1}, ..., x_{j+k-1}], c_m), \tag{5.3}$$

której pierwszy element jest podsekwencją tokenów o długości k, natomiast c_m jest klasą encji przypisaną do tej jednostki nazewniczej. Zgodnie z tą definicją, jednostkę nazewniczą będziemy uznawać za duplikat innej jednostki wtedy i tylko wtedy, gdy obie jednostki będą przypisane do tego samego podciągu elementów sekwencji oraz będą należeć do tej samej klasy. Dopuszczalne jest istnienie w zbiorze kilku jednostek dla tego samego podciągu, ale przypisanych do różnych klas.

Zdefiniujmy również funkcję rozszerzającą zbiór S_{i-1} na podstawie sekwencji tagów $\vec{\boldsymbol{y}}^{(\boldsymbol{i})}$:

$$r : (\vec{\boldsymbol{y}}^{(\boldsymbol{i})}, S_{i-1}) \rightarrow S_i, \tag{5.4}$$

której celem aktualizacja stanu modelu przed kolejną iteracją. Funkcja przekształca sekwencję wynikową $\vec{\boldsymbol{y}}^{(\boldsymbol{i})}$ na jednostki nazewnicze reprezentowane przez tą sekwencję, a następnie dodaje do zbioru nowe jednostki, które nie zostały jeszcze wykryte w poprzednich iteracjach.

Korzystając z powyższych definicji, procedurę predykcji hierarchicznej struktury jednostek nazewniczych w ujęciu iteracyjnym można zapisać za pomocą następujących kroków:

1. Inicjalizujemy stan modelu przed pierwszą iteracją jako pusty zbiór jednostek nazewniczych $S_0 = \varnothing$.

2. Dla kolejnych iteracji i, zaczynając od $i = 1$, wykonujemy krok predykcji, po którym aktualizujemy stan modelu o nowe jednostki nazewnicze:

$$\vec{\boldsymbol{y}}^{(\boldsymbol{i})} = f(\vec{\boldsymbol{x}}, S_{i-1}), \quad S_i = r(\vec{\boldsymbol{y}}^{(\boldsymbol{i})}, S_{i-1}) \tag{5.5}$$

Predykcja iteracyjna jest przerywana w momencie, gdy w danym kroku nie zostały wykryte żadne nowe jednostki nazewnicze, czyli stan modelu S_{i-1} w poprzednim kroku jest równy stanowi modelu w aktualnym kroku S_i: $|S_{i-1}| = |S_i|$.

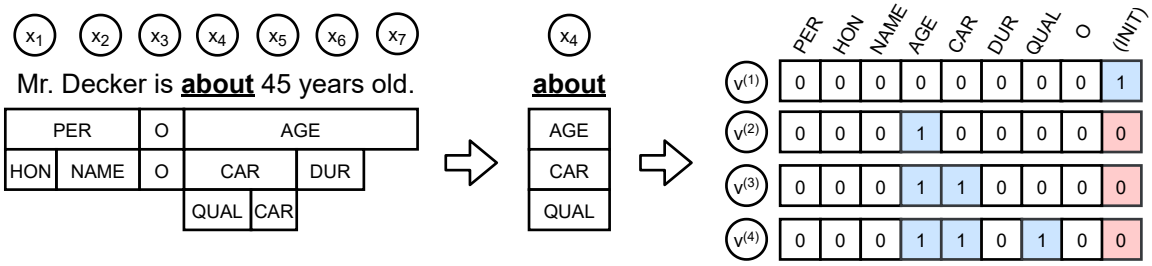
3. Wynikiem działania modelu iteracyjnego jest ostatni zapisany stan modelu.

Zwróćmy uwagę na fakt, że w ujęciu iteracyjnym klasyczny płaski problem tagowania sekwencji jest przypadkiem szczególnym, w którym ograniczamy się do pojedynczej iteracji. Stan modelu jest pustym zbiorem, więc predykcja w pierwszym kroku odbywa się wyłącznie w oparciu o sekwencję tokenów $\vec{\boldsymbol{x}}$. Ze względu na podobieństwo do klasycznego ujęcia tagowania sekwencji, istniejące modele rozwiązujące ten problem mogą być w prosty sposób dostosowane do działania iteracyjnego. W następnej części tego rozdziału przedstawiony zostanie konkretny przykład neuronowej architektury iteracyjnej. Zaproponowana w tej pracy architektura jest jedną z możliwych propozycji. Wykorzystując analogiczne techniki, możliwe byłoby wprowadzenie modyfikacji również w innych modelach, które oryginalnie używane są w problemach płaskiego tagowania sekwencji.

Można również zauważyć, że opisana powyżej procedura predykcji iteracyjnej wskazuje na pewne podobieństwa do metody *layering* opisanej w Alex et al. [3]. Zaproponowana tam technika również opiera się na podziale hierarchicznej struktury jednostek nazewniczych na warstwy, a predykcja każdej z tych warstw odbywa się w osobnym kroku. Występują jednak istotne elementy odróżniające *layering* od metod iteracyjnych będących przedmiotem tej rozprawy. Przede wszystkimi, w Alex et al. [3] każdy etap predykcji jest dokonywany przez osobny model, zatem liczba modeli wymaganych do wytrenowania jest równa maksymalnej liczbie obsługiwanych warstw występującej w zbiorze danych lub założonej jako hiperparametr przed rozpoczęciem trenowania. W przypadku metod iteracyjnych wszystkie kroki predykcji są wykonywane przez ten sam model, co pozwala nie tylko na uproszczenie architektury rozwiązania i zmniejszenie czasu trenowania całego systemu, ale też umożliwia wykonywanie predykcji w sposób dynamiczny, bez konieczności ustalania stałej liczby iteracji. W ujęciu iteracyjnym proces predykcji jest przerywany w oparciu o aktualny i poprzedni stan zbioru wykrytych jednostek nazewniczych, model więc sam dostosowuje się do poziomu złożoności struktury występującej dla danej sekwencji tokenów i decyduje o liczbie wymaganych iteracji. Kolejną istotną różnicą jest reprezentacja stanu przekazywanego pomiędzy poszczególnymi iteracjami. W *layeringu* lista atrybutów wejściowych każdego modelu uwzględnia tylko wyniki predykcji modelu dla poprzedniej warstwy. W naszej metodzie zakładamy, że w każdej iteracji model otrzymuje wyniki predykcji wszystkich poprzedzających iteracji zapisane w zbiorze S_i, czyli reprezentację wszystkich dotychczas zidentyfikowanych jednostek nazewniczych.

5.2. Neuronowy model iteracyjny

Niniejsza sekcja opisuje zaproponowaną w tej pracy neuronową architekturę iteracyjną. Architektura składa się ze standardowych komponentów używanych w modelach sekwencyjnych. Pierwszym etapem przetwarzania sekwencji tokenów w neuronowym modelu iteracyjnym jest ich transformacja do postaci wektorowej. Dla każdego elementu sekwencji wejściowej, jego reprezentacja wektorowa tworzona jest w oparciu o moduł reprezentacji tekstu oraz zakodowany stan modelu w iteracji i. Przez moduł reprezentacji tekstu rozumiemy jeden model lub zestaw modeli służących do reprezentowania tokenów w postaci wektorowej, które zostały opisane szczegółowo w rozdziale 3. Ze



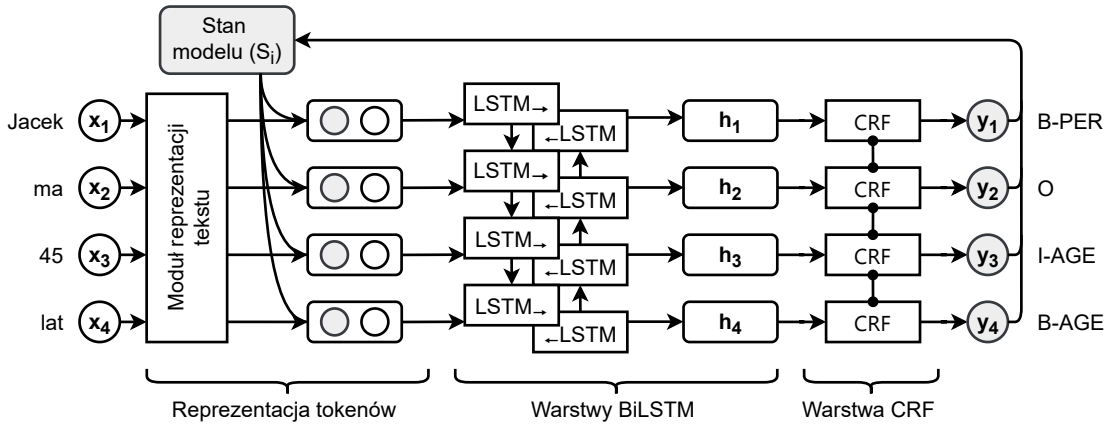
Rysunek 5.1: Przykład kodowania wektorów stanu w kolejnych iteracjach modelu dla elementu x_4 sekwencji, której przypisano trzypoziomową strukturę jednostek nazewniczych.

względu na fakt, iż modele te są na ogół wstępnie trenowane na korpusach tekstowych pochodzących z konkretnego języka i konkretnej domeny, ich wybór jest uzależniony od zbioru danych, na którym docelowo trenowany będzie model iteracyjny. Opisane w dalszej części tej pracy eksperymenty dotyczą zbiorów dla różnych języków i domen, w związku z użyte zostały różne metody reprezentacji wektorowej. Rozdział 7 zawiera opis przeprowadzonych eksperymentów oraz dyskusję na temat wybranych sposobów reprezentowania tokenów. W ogólnym ujęciu, ostateczna reprezentacja wektorowa każdego elementu sekwencji jest konkatenacją wszystkich wektorów składowych:

$$\boldsymbol{x_j^{(i)}} = \boldsymbol{v_j^{(i)}} \odot \boldsymbol{w_{j,1}} \odot ... \odot \boldsymbol{w_{j,m}} \tag{5.6}$$

gdzie $\odot$ jest operatorem konkatenacji wektorów, $\boldsymbol{x_j^{(i)}}$ reprezentacją elementu sekwencji na pozycji j podczas iteracji i, $\boldsymbol{v_j^{(i)}}$ jest zakodowanym stanem modelu dla tej pozycji oraz iteracji, a wektory od $\boldsymbol{w_{j,1}}$ do $\boldsymbol{w_{j,m}}$ są reprezentacjami tekstu dla pozycji j. Reprezentacje tekstowe pozostają takie same dla wszystkich iteracji modelu, zmianie ulega jedynie zakodowany wektor stanu.

Wektor stanu, konstruowany na podstawie zbioru S_i, jest odzwierciedleniem klas encji, które zostały przypisane do elementu sekwencji w poprzednich iteracjach. Jest to wektor binarny, którego poszczególne wymiary odpowiadają klasom ze zbioru wszystkich dopuszczalnych klas C. Jeżeli klasa c_k została zidentyfikowana dla danego elementu w dotychczasowych iteracjach, wymiar k tego wektora będzie równy 1, w przeciwnym wypadku 0. Ostatnim elementem wektora jest dodatkowa flaga *(INIT)*, która jest równa 1 tylko w pierwszej iteracji modelu, natomiast dla pozostałych iteracji jest ustawiana na 0. Celem tej flagi jest wyróżnienie pierwszej iteracji, w której nie zostały wykryte jeszcze żadne jednostki nazewnicze. Dzięki temu model może nauczyć się ignorować wektor stanu podczas inicjalnej predykcji i opierać się wyłącznie na reprezentacji tekstu, natomiast uwzględniać go w kolejnych iteracjach, w których wektor zawiera już wyniki poprzednich predykcji. Na Rysunku 5.1 przedstawiono przykład kodowania wektora stanu w czterech pierwszych iteracjach modelu dla problemu wykrywania jednostek nazewniczych z ośmioma klasami: *{PER, HON, NAME, AGE, CAR, DUR, QUAL, O}*. Element x_4 należy do trzech jednostek nazewniczych, które zostały przypisane do klas *AGE*, *CAR* oraz *QUAL*. W pierwszej iteracji wektor stanu będzie składał się z samych zer oraz wartości 1 na pozycji *(INIT)*. W kolejnych iteracjach mo-



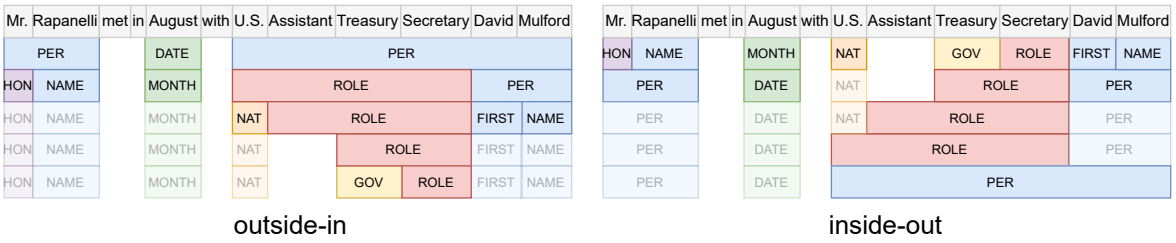
Rysunek 5.2: Schemat architektury neuronowego modelu iteracyjnego.

del dokonuje predykcji nowych jednostek nazewniczych, aktywując tym samym kolejne pozycje w wektorze stanu odpowiadające klasom wykrytym dla danego elementu sekwencji. Dla uproszczenia powyższego przykładu przyjęliśmy, że każda kategoria encji jest reprezentowana przez tylko jedną klasę. W praktyce model wykorzystuje schemat kodowania jednostek nazewniczych *BIO*, w którym kategoria jest reprezentowana przez tagi początku oraz środka jednostki. W związku z tym rzeczywisty wymiar wektora stanu ponad dwukrotnie większy niż liczba wszystkich kategorii jednostek.

Schemat całego modelu neuronowego przedstawiono na Rysunku 5.2. Utworzona dla każdego elementu sekwencji reprezentacja wektorowa jest transformowana przez blok składający się z dwukierunkowych warstw LSTM. Wynikiem tych warstw są reprezentacje ukryte tokenów, na podstawie których dokonywana jest predykcja modelu. Jako warstwy predykcyjnej użyto *linear-chain CRF*, co pozwala na uwzględnienie strukturalnych zależności pomiędzy poszczególnymi elementami sekwencji wyjściowej modelu. Wyjściem modelu jest sekwencja tagów, z której po dokonanej predykcji ekstrahowane są wykryte jednostki nazewnicze, a następnie na ich podstawie aktualizowany jest stan modelu. Jeżeli stan nie uległ zmianie, to znaczy zbiór wykrytych do tej pory jednostek nie powiększył się o nowe elementy, model kończy działanie i zwraca aktualny zbiór jednostek. Jeżeli stan zmienił się, wykonywana jest kolejna iteracja.

5.3. Trenowanie modeli iteracyjnych

Modele iteracyjne mogą być trenowane poprzez minimalizację funkcji kosztu za pomocą powszechnie stosowanych w sieciach neuronowych metod spadku gradientu (ang. *gradient descent*). Proces trenowania jest podobny do płaskich modeli tagowania sekwencji wykorzystujących CRF w warstwie predykcyjnej [50, 66]. Potencjalnym problemem w trenowaniu modeli iteracyjnych jest konieczność przekazywania stanu pomiędzy iteracjami, co mogłoby skomplikować oraz wydłużyć proces uczenia. Dlatego też w zaproponowanej w tej pracy metodzie stosujemy uproszczone podejście, którego założeniem jest traktowanie każdej iteracji modelu jako osobnej próbki uczącej. Dzięki takiemu podejściu możliwe jest trenowanie modeli iteracyjnych bez konieczności wykonywania rzeczywistych iteracji na etapie uczenia oraz bez konieczności dokonywania dużych



Rysunek 5.3: Przykład podziału struktury jednostek nazewniczych na warstwy w ujęciu *outside-in* oraz *inside-out*. W warstwach, w których dla danego elementu sekwencji nie zdefiniowano żadnej jednostki nazewiczej, model jest trenowany do powtarzania swojej ostatniej predykcji. Powtórne predykcje oznaczono jaśniejszym kolorem.

zmian w samym procesie optymalizacji aby dostosować go do ujęcia iteracyjnego. Z pojedynczej sekwencji wraz z jej wielowarstwową strukturą jednostek nazewniczych o głębokości zagnieżdżenia wynoszącej d, generowanych jest d osobnych próbek treningowych. Każda próbka reprezentuje oczekiwany wynik predykcji modelu dla tej sekwencji w pojedynczej iteracji. Poprzedni stan modelu S_i dla danej próbki jest tworzony sztucznie na etapie przygotowywania zbioru danych na podstawie rzeczywistych tagów przypisanych elementom sekwencji. Podczas treningu model otrzymuje zatem na wejściu stan iteracji, który byłby wygenerowany przez bezbłędny model iteracyjny. Taka technika uczenia nazywana jest *teacher forcing* [136] i była wcześniej stosowana w innych rodzajach modeli, których zasadą działania jest budowanie struktury wynikowej na podstawie wielokrotnych predykcji. Znalazła ona zastosowanie między innymi w zadaniach generowania obrazu [128, 115] oraz dźwięku [129].

Istnieją dwa warianty modeli iteracyjnych. Podobnie jak w przypadku metody *layering* z Alex et al. [3], możemy dokonać podziału struktury jednostek nazewniczych na warstwy jednym z dwóch kierunków: od bardziej ogólnych do bardziej szczegółowych (*outside-in*) lub od bardziej szczegółowych do bardziej ogólnych (*inside-out*). Wiąże się to z różnymi sposobami przygotowania zbioru, na którym model jest trenowany. Sama idea dzielenia struktury na warstwy jest dosyć intuicyjna, poszczególne implementacje mogą się jednak różnić szczegółami mającymi istotny wpływ na jakość wytrenowanego modelu. Podział ten na potrzeby modeli iteracyjnych różni się w dwóch aspektach od metody zaproponowanej w Alex et al. [3]. Wizualizację zaproponowanej w tej pracy techniki podziału pokazano na Rysunku 5.3. Widzimy na nim przykład zdania z hierarchiczną strukturą jednostek nazewniczych uszeregowaną w kierunkach *outside-in* oraz *inside-out*.

Pierwszym problemem, na który warto zwrócić uwagę, są występujące ”przerwy” w niektórych iteracjach. Przerwy takie mogą wystąpić, gdy dla danej warstwy oraz danego elementu sekwencji nie ma już więcej zdefiniowanych jednostek nazewniczych lub gdy nie można dodać jednostki nazewniczej do warstwy ponieważ jej pozycja jest już zajęta przez inną jednostkę. W klasycznym modelu *layering* oczekiwanym wynikiem predykcji dla takiej iteracji jest brak klasy encji, reprezentowany przez tag O (*outside*). W przypadku modelu iteracyjnego oczekiwane jest powtórzenie predykcji ostatnio zidentyfikowanej klasy. Wstępne prace nad ujęciem iteracyjnym wykazały, że modele osiągają znacząco lepsze wyniki, gdy są trenowane do powtarzania ostatniej

predykcji w przypadku, gdy w danej warstwie nie jest zdefiniowana żadna nowa jednostka nazewnicza. Powtórna predykcja nie ma wpływu na działanie samego modelu, bowiem ponowne wykrycie już wcześniej zidentyfikowanej encji jest równoznaczne z nie wykryciem encji - w obu przypadkach stan modelu na koniec iteracji nie ulega zmianie.

Drugim aspektem wyróżniającym stosowany w modelach iteracyjnych podział na warstwy jest rozwiązywanie niejednoznaczności w szeregowaniu jednostek nazewniczych. Zazwyczaj ustalenie kolejności jednostek w warstwach jest proste: dłuższa jednostka jest uznawana za bardziej ogólną a krótsza za bardziej szczegółową. Problem pojawia się w przypadku gdy mamy do czynienia z dwoma klasami jednostek o takiej samej długości. W omawianym przykładzie są to klasy *DATE* i *MONTH* przypisane do słowa "August". Algorytm podziału nie jest w stanie automatycznie określić kolejności pomiędzy tymi jednostkami na podstawie pojedynczej sekwencji, ale może to zrobić w oparciu o statystyki wyliczone z całego zbioru danych. Dlatego też przed wygenerowaniem próbek do zbioru treningowego zliczana jest częstotliwość wystąpień wszystkich jednoznacznych relacji hierarchicznych pomiędzy klasami jednostek. W momencie wystąpienia jednostek o równej długości, algorytm sprawdza, która z klas występowała częściej w zbiorze jako klasa nadrzędna i na tej podstawie szereguje jednostki w warstwach. W publikacji Alex et al. [3] kwestia ujednoznaczniania kolejności jednostek nie została poruszona.

5.4. Podsumowanie

W rozdziale przedstawiono oryginalne iteracyjne ujęcie problemu wykrywania hierarchicznych struktur jednostek nazewniczych. Modele iteracyjne można traktować jako rozszerzenie klasycznych modeli tagowania sekwencji, ich działanie polega bowiem na wykonaniu kilku niezależnych kroków predykcji odpowiadających poszczególnym warstwom zagnieżdżonej struktury jednostek. W porównaniu do standardowych modeli tagujących, w ujęciu iteracyjnym wprowadzamy stan modelu, który przekazywany jest pomiędzy kolejnymi iteracjami. Dzięki zakodowanemu stanomi model może nauczyć się wykrywania innego podzbioru jednostek w każdej iteracji. W przeciwieństwie do większości rozwiązań znanych z literatury, ogólne ujęcie iteracyjne nie narzuca żadnych szczególnych ograniczeń na architekturę rozwiązania - jedynym wymogiem jest uwzględnienie stanu jako wejścia modelu. W dalszej części rozdziału przedstawiony został neuronowy model iteracyjny, będący przykładową implementacją powyższego ujęcia oraz stanowiący podstawę do eksperymentów przeprowadzonych w ramach niniejszej pracy. Opisano szczegółowo sposób obsługi stanu w modelu oraz metody trenowania modeli iteracyjnych.

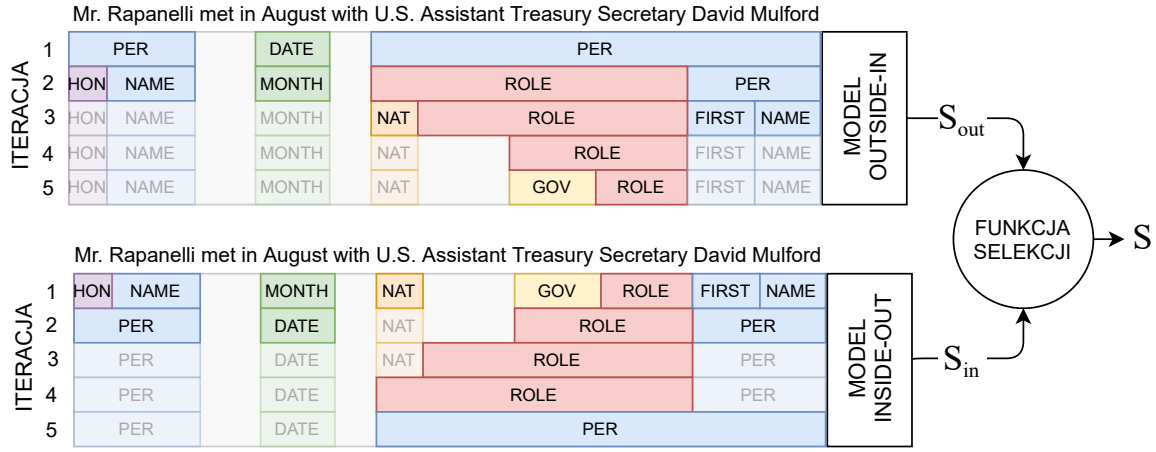
Rozdział 6

Dwukierunkowy algorytm iteracyjny

Poprzedni rozdział stanowił wprowadzenie do iteracyjnych metod wykrywania hierarchicznych struktur jednostek nazewniczych. Opisany w nim neuronowy model iteracyjny może być wykorzystywany do rozwiązywania problemu identyfikacji zagnieżdżonych jednostek. Niniejszy rozdział porusza bardziej zaawansowane zagadnienia dotyczące ujęcia iteracyjnego. Jego głównym celem jest zaproponowanie nowej metody identyfikacji jednostek nazewniczych polegającej na połączeniu wyników dwóch modeli iteracyjnych przy pomocy dodatkowej funkcji selekcji. Przedstawiony w tym rozdziale dwukierunkowy algorytm iteracyjny stanowi alternatywę dla prostej predykcji iteracyjnej opartej na pojedynczym modelu. Na początku opisany zostanie sposób działania algorytmu dwukierunkowego, następnie opisane zostaną przykłady funkcji selekcji, które mogą być stosowane w kontekście tego algorytmu. W dalszej części rozdziału zawarto dyskusję na temat alternatywnych metod przekazywania stanu modelu w predykcji dwukierunkowej oraz porównanie złożoności obliczeniowej metod iteracyjnych z innymi metodami wykrywania jednostek znanymi z literatury.

6.1. Predykcja dwukierunkowa

Jak pamiętamy z poprzedniego rozdziału, pojedynczy model iteracyjny buduje wynikowy zbiór jednostek nazewniczych w sposób przyrostowy, rozpoczynając od pustego zbioru i rozszerzając go w kolejnych iteracjach o dodatkowo zidentyfikowane jednostki. W przypadku struktury jednostek składającej się z d warstw, wynikiem działania modelu w kolejnych iteracjach będą stany modelu oznaczone $\{S_1, S_2, ..., S_d\}$, przy czym ostatni stan S_d zawiera wszystkie wykryte przez model jednostki i jest traktowany końcowy wynik predykcji. Model taki może być wykorzystywany samodzielnie w problemach wykrywania hierarchicznych struktur jednostek, natomiast interesującym pomysłem wydaje się wykorzystanie dwóch różnych sposobów predykcji iteracyjnej. Modele iteracyjne mogą być trenowane w kierunku *inside-out* lub *outside-in*, a w każdym z tych kierunków modelowi prezentowany jest inny układ danych. W efekcie modele trenowane w przeciwnych kierunkach mogą nauczyć się wykrywania różnych wzorców w danych, a co za tym idzie, mogą się wzajemnie uzupełniać. Zakładamy, że pewne struktury jednostek nazewniczych łatwiej jest wykrywać rozpoczynając od jednostek bardziej ogólnych, inne zaś rozpoczynając od bardziej szczegółowych. Połączenie wy-



Rysunek 6.1: Schemat wykrywania hierarchicznej struktury jednostek nazewniczych w dwukierunkowym algorytmie iteracyjnym. Dwa modele iteracyjne wykrywają jednostki identyfikują jednostki warstwa po warstwie w kierunkach *inside-out* oraz *outside-in*. Wynikiem działania modeli są zbiory jednostek S_{in} i S_{out}, które są następnie łączone w zbiór wynikowy S przy wykorzystaniu dodatkowego komponentu zwanego *funkcją selekcji*.

ników modeli *inside-out* oraz *outside-in* może zatem przyczynić się do poprawy jakości predykcji systemu.

Powyższe założenia są podstawą działania dwukierunkowego algorytmu iteracyjnego. Zasada działania algorytmu jest prosta. Trenujemy dwa niezależne modele iteracyjne do wykrywania struktury jednostek nazewniczych w przeciwnych kierunkach. Podczas predykcji, każdy z modeli na podstawie wejściowej sekwencji słów generuje odrębne zbiory zidentyfikowanych jednostek, które następnie są łączone w jeden zbiór będący wynikiem działania algorytmu. Predykcja w systemie dwukierunkowym składa się więc z dwóch opisanych wcześniej modeli iteracyjnych oraz nowego komponentu, którego zadaniem jest połączenie wyników tych modeli. W dalszej części rozdziału komponent ten będziemy nazywać *funkcją selekcji*. Jeżeli wynik modelu *inside-out* oznaczymy jako S_{in} a wynik modelu *outside-in* jako S_{out}, to funkcję selekcji s definiujemy następująco:

$$s : (S_{in}, S_{out}) \rightarrow S \tag{6.1}$$

gdzie S finalnym zbiorem jednostek nazewniczych w algorytmie dwukierunkowym. Zbiór S zawiera pewien podzbiór jednostek nazewniczych z zbiorów S_{in} i S_{out}, nie zawiera natomiast nowych jednostek nazewniczych, które nie zostały wcześniej zidentyfikowane. Celem funkcji selekcji jest zatem wybranie, które spośród wykrytych przez modele jednostek powinny być uwzględnione w ostatecznym wyniku predykcji. Funkcja selekcji może mieć dowolną postać. Może działać w oparciu o manualnie zaimplementowany proces lub zestaw reguł, może być również dodatkowym modelem uczenia maszynowego wymagającym wytrenowania na zbiorze danych. W kolejnej sekcji zaprezentowane zostaną przykłady funkcji selekcji, które zaimplementowano oraz przetestowano w ramach niniejszej pracy.

Przykład wykrywania zagnieżdżonej struktury jednostek nazewniczych za pomocą

dwukierunkowego algorytmu iteracyjnego przedstawiono na Rysunku 6.1.

6.2. Funkcje selekcji

Funkcja selekcji pełni w dwukierunkowym algorytmie iteracyjnym rolę modułu decyzyjnego, który ma za zadanie zaakceptować lub odrzucić jednostki nazewnicze zidentyfikowane przez modele iteracyjne. Funkcja selekcji nie operuje zatem bezpośrednio na wejściowej sekwencji tokenów, ale na wynikach predykcji wykonanej przez modele. W uproszczeniu funkcję tą można traktować jak klasyfikator binarny, który dla każdej jednostki niezależnie podejmuje decyzję o uwzględnieniu jej w ostatecznym zbiorze wyników. Decyzja ta jest podejmowana w oparciu o zestaw atrybutów dotyczących jednostki, wyodrębnionych z danych oraz predykcji modeli iteracyjnych. Zwróćmy uwagę na fakt, że ta sama jednostka, czyli jednostka zdefiniowana dla tej samej sekwencji tokenów i posiadająca taką samą klasę, może być zidentyfikowana przez dwa modele iteracyjne lub tylko przez jeden z nich. Jeżeli jednostka nazewnicza została wykryta przez dwa modele, mamy do czynienia z kompletnym zestawem atrybutów. Jeżeli tylko jeden model wykrył daną jednostkę nazewniczą, zestaw atrybutów jest niekompletny - możemy określić jedynie cechy jednostki w kontekście wyników tego modelu. W takich przypadkach atrybuty dla drugiego modelu przyjmują wartość 0 (dla cech o wartościach liczbowych) lub *NULL* (specjalna kategoria dla cech o wartościach nominalnych).

Jedną z cech wykorzystywanych przez zaproponowane w tej pracy funkcje selekcji jest *wskaźnik pewności* predykcji modelu. Jest to wartość liczbowa z zakresu od 0 do 1, określająca jak wysokie prawdopodobieństwo model przypisał do sekwencji tagów reprezentowanych przez jednostkę nazewniczą w porównaniu do alternatywnych sekwencji tagów możliwych dla tej samej frazy. Wartość ta może być wyliczona bezpośrednio z tabeli $\boldsymbol{T^{(1)}}$, konstruowanej w algorytmie Viterbiego [131], którego opis znajduje się w rozdziale 2. Przypomnijmy, że element $T^{(1)}_{i,j}$ w tej tablicy reprezentuje wartość najlepszej znalezionej przez model sekwencji tagów, której na pozycji i przypisano klasę c_j. Na początek zdefiniujmy wskaźnik pewności dla pojedynczego elementu sekwencji x_i i klasy c_j jako wartość $T^{(1)}_{i,j}$ podzieloną przez sumę wszystkich wartości i-tym wierszu tej tablicy:

$$\mathrm{cnf}(x_i, c_j) = \frac{T^{(1)}_{i,j}}{\sum_{c_k \in C} T^{(1)}_{i,k}} \tag{6.2}$$

gdzie C jest zbiorem dopuszczalnych klas jednostek. Mając daną funkcję $\mathrm{cnf}(x_i, c_j)$ dla pojedynczego elementu x_i, możemy określić również wskaźnik pewności dla jednostki nazewniczej m o klasie c_m i długości k, przypisanej do podsekwencji elementów $[x_i, x_{i+1}, ..., x_{i+k-1}]$. Wskaźnik ten będzie równy średniej ważonej z wartości $\mathrm{cnf}(x_i, c_m)$ wszystkich elementów wchodzących w skład tej jednostki:

$$\mathrm{cnf}(m) = \mathrm{cnf}([x_i, x_{i+1}, ..., x_{i+k-1}], c_m) = \frac{\sum_{j=0}^{k-1} \mathrm{cnf}(x_{i+j}, c_m)}{k} \tag{6.3}$$

Warto zwrócić uwagę, że w kontekście predykcji iteracyjnej ta sama jednostka może być zidentyfikowana przez model kilkukrotnie w różnych iteracjach, za każdym razem

z inną wartością cnf(m). W szczególności model uczony do powtarzania poprzedniej predykcji może zwracać tą samą jednostkę z coraz wyższym wskaźnikiem pewności w kolejnych iteracjach. Dlatego też przez funkcje selekcji brana pod uwagę jest jedynie wartość tego wskaźnika dla pierwszej iteracji, w której dana jednostka została wykryta.

W ramach niniejszej pracy zdefiniowano oraz przetestowano sześć funkcji selekcji. Znalazły się wśród nich zarówno funkcje oparte na prostych regułach, które można stosować jako rozwiązania bazowe (ang. *baselines*), jak również bardziej zaawansowane funkcje parametryczne. Możliwe jest oczywiście zdefiniowanie wielu innych rodzajów funkcji selekcji, w szczególności dostosowanych do charakterystyki konkretnego zbioru danych czy zastosowania. Funkcje zaproponowane w tej pracy są dosyć uniwersalne, nadają się więc do stosowania w większości problemów wykrywania hierarchicznych struktur jednostek nazewniczych. W pracy dokonano ewaluacji następujących funkcji selekcji:

• **Tylko inside-out** oraz **tylko outside-in** - Jest to najprostszy rodzaj selekcji regułowej wybierający tylko jednostki nazewnicze pochodzące z modelu *inside-out* lub z modelu *outside-in*. Zastosowanie tego rodzaju selekcji jest równoważne z używaniem samodzielnego neuronowego modelu iteracyjnego. Funkcje te zostały uwzględnione z uwagi na możliwość porównania jakości pojedynczych modeli do funkcji łączących wyniki obu modeli.

• **Suma modeli** - Ta funkcja selekcji wybiera wszystkie jednostki nazewnicze znajdujące się w zbiorach S_{in} oraz S_{out}. Odpowiada ona zatem operacji sumy zbiorów.

• **Część wspólna modeli** - W tej funkcji selekcji jednostka nazewnicza jest uwzględniona w zbiorze wynikowym S wtedy i tylko wtedy, gdy znajduje się ona jednocześnie w zbiorach S_{in} oraz S_{out}. Odpowiada ona zatem operacji przecięcia (części wspólnej) zbiorów.

• **Selekcja oparta na pewności** - Jest to rodzaj selekcji, który bazuje na zdefiniowanym wcześniej wskaźniku pewności predykcji jednostki nazewniczej cnf(m). W tej funkcji selekcji jednostka jest uwzględniana w zbiorze wynikowym S jeżeli zajdzie jeden z trzech warunków: 1) jednostka została zidentyfikowana przez oba modele, 2) jednostka została zidentyfikowana tylko przez model *outside-in* ale ze wskaźnikiem pewności wyższym niż h_{out}, 3) jednostka została zidentyfikowana tylko przez model *inside-out* ale ze wskaźnikiem pewności wyższym niż h_{in}. Minimalne progi pewności h_{in} i h_{out} są hiperparametrami tej funkcji. Po wytrenowaniu modeli na treningowej części zbioru danych, cześć walidacyjna zbioru jest używana do ustalenia optymalnych wartości hiperparametrów. Wartości progów są ustalane automatycznie na poziomie, który maksymalizuje wartość metryki F1 dla zbioru walidacyjnego.

• **Selekcja oparta na klasyfikatorze** - Ta selekcja wykorzystuje dodatkowy model regresji logistycznej, trenowany na zestawie atrybutów jednostek nazewniczych wyodrębnionych z metadanych dotyczących predykcji modeli iteracyjnych oraz treningowej części zbioru danych. Kryterium decydujące o uwzględnieniu jednostki w zbiorze wynikowym S oparte jest na wartości funkcji sigmoid $\sigma(\boldsymbol{x})$, będącej wyjściem wytrenowanego modelu. Akceptowane są tylko jednostki, dla których wartość $\sigma(\boldsymbol{x})$ jest większa

Opis atrybutów	Liczba	Rodzaj
Wartości wskaźnika cnf(m) dla jednostki nazewniczej w zbiorach S_{out} oraz S_{in}. 0 w przypadku, gdy model nie wykrył danej jednostki.	2	liczbowe
Wartości wskaźnika cnf(m) dla bezpośredniego rodzica (jednostki nadrzędnej) w stosunku do danej jednostki w zbiorach S_{out} oraz S_{in}. 0 w przypadku, gdy model nie wykrył danej jednostki.	2	liczbowe
Maksimum oraz minimum z wartości wskaźników cnf(m) dla jednostki nazewniczej w zbiorach S_{out} oraz S_{in}.	2	liczbowe
Klasa jednostki nazewniczej.	1	nominalny
Klasa bezpośredniego rodzica (jednostki nadrzędnej) danej jednostki nazewniczej w zbiorach S_{out} oraz S_{in}.	2	nominalne

Tabela 6.1: Opis atrybutów jednostkek nazewniczych wykorzystywanych w metodzie selekcji opartej na klasyfikatorze.

od ustalonego progu h. Minimalny próg h jest hiperparametrem tej metody selekcji i, podobnie jak w przypadku poprzedniej metody, jest dobierany automatycznie przy wykorzystaniu walidacyjnej części zbioru danych, maksymalizując wartość F1 dla tego podzbioru danych.

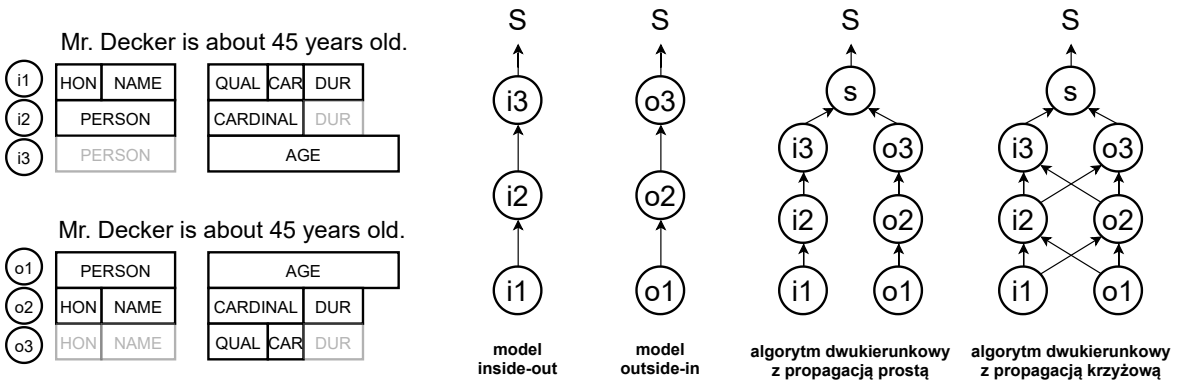
Listę atrybutów wykorzystywanych przez selekcję opartą na regresji logistycznej przedstawiono w Tabeli 6.1. Wyodrębnione cechy jednostki nazewniczej mają postać liczbową lub nominalną. Wartości liczbowe mogą być bezpośrednio użyte jako elementy wektora wejściowego modelu. Każdy atrybut nominalny jest natomiast przekształcany na $k + 1$ osobnych atrybutów liczbowych zakodowanych przy pomocy metody *one-hot*, gdzie k jest liczbą dopuszczalnych kategorii dla tego atrybutu a dodatkowy wymiar służy do reprezentowania specjalnej kategorii *NULL* (brak wartości). Wszystkie wartości są następnie łączone w wektor x, który stanowi wejście dla modelu regresji logistycznej:

$$\sigma(\boldsymbol{x}) = \frac{1}{1 + e^{-(\boldsymbol{w}\boldsymbol{x}+b)}} \tag{6.4}$$

gdzie $\boldsymbol{w}$ jest wektorem wag modelu a b jest wartością bias. Model regresji logistycznej użyty w tej funkcji selekcji uczony jest metodą stochastycznego spadku wzdłuż gradientu (ang. *stochastic gradient descent*) z dodatkową regularyzacją L2.

6.3. Propagacja w modelach dwukierunkowych

Jednym z kluczowych mechanizmów w iteracyjnym ujęciu problemu wykrywania hierarchicznych struktur jednostek nazewniczych jest przekazywanie stanu modelu pomiędzy poszczególnymi iteracjami. Dzięki informacji na temat wykrytych wcześniej jednostek, model ma możliwość dokonywania różnych predykcji w kolejnych iteracjach i rozszerzania zbioru zidentyfikowanych fraz. W przypadku pojedynczego modelu predykcja jest wykonywana na podstawie liniowo uporządkowanej sekwencji operacji, w związku



Rysunek 6.2: Propagacja stanu w modelach dwukierunkowych.

z czym istnieje jeden logiczny sposób przekazywania stanu - stan modelu jest propagowany z poprzedniej do następnej iteracji. Wykorzystanie dwóch modeli w dwukierunkowym algorytmie iteracyjnym daje możliwość przetestowania alternatywnych sposobów propagacji stanu.

W klasycznej wersji algorytmu dwukierunkowego traktujemy modele *outside-in* i *inside-out* niezależnie. Modele są trenowane osobno metodą *teacher forcing* opisaną w poprzednim rozdziale. Każdy z modeli dokonuje również samodzielnej predykcji, otrzymując na wejściu zakodowany stan pochodzący z poprzedniej iteracji tego samego modelu. Pierwsza interakcja pomiędzy modelami zachodzi dopiero na końcowym etapie działania algorytmu, kiedy zbiory wynikowe S_{in} i S_{out} są łączone za pomocą funkcji selekcji. Mając do dyspozycji więcej niż jeden model, możemy jednak rozważać również metody propagacji stanu, które zakładają współdzielenie stanu pomiędzy modelami. Na Rysunku 6.2 pokazano przykład zdania z trójwarstwową strukturą jednostek nazewniczych oraz cztery różne metody propagacji stanu. Jest to liniowa propagacja typu *inside-out* i *outside-in* charakterystyczna dla samodzielnych modeli iteracyjnych, algorytm dwukierunkowy z propagacją prostą działający według schematu opisanego powyżej, oraz algorytm dwukierunkowy z propagacją krzyżową jako przykład metody współdzielenia stanu pomiędzy modelami. W propagacji krzyżowej każdy model otrzymuje na wejściu dwa wektory stanu - jeden pochodzący z poprzedniej iteracji tego samego modelu oraz kolejny pochodzący z iteracji modelu działającego w przeciwnym kierunku. Ta technika przekazywania stanu wymaga interakcji pomiędzy modelami już na wstępnym etapie generowania predykcji.

Propagacja krzyżowa wymusza wzajemną zależność pomiędzy modelami. Wytrenowane w ten sposób modele nie mogą być używane samodzielnie z pominięciem algorytmu dwukierunkowego, każdy z nich do prawidłowego działania wymaga bowiem wyników drugiego z modeli. Proces uczenia przy zastosowaniu propagacji krzyżowej jest podobny do trenowania niezależnych modeli. W tym przypadku również zastosowanie ma metoda *teacher forcing*, tym razem jednak w danych treningowych musi być przechowywana struktura jednostek w obu kierunkach aby możliwe było wygenerowanie sztucznego stanu modelu dla danej próbki symulującego działanie zarówno modelu *inside-out* jak i *outside-in*. Wadą modeli działających w sposób krzyżowy jest bardziej skomplikowany mechanizm generowania predykcji. Proces budowy wynikowych

zbiorów jednostek nazewniczych S_{in} i S_{out} musi uwzględniać synchronizację pomiędzy modelami. W schemacie krzyżowym wykonywana jest pojedyncza iteracja obu modeli, następnie generowane są stany odpowiadające tej iteracji, które są przekazywane jako wejście kolejnej iteracji. Iteracje obu modeli muszą być wykonywane jedna po drugiej w tym samym czasie. Zarówno w przypadku propagacji prostej jak i krzyżowej możliwe jest współbieżne uruchamianie obu modeli, jednak konieczność synchronizowania stanu po każdej iteracji sprawia, że algorytm dwukierunkowy z propagacją krzyżową może działać nieco wolniej.

W części eksperymentalnej pracy znajduje się porównanie wyników dwukierunkowego algorytmu iteracyjnego przy zastosowaniu propagacji prostej oraz krzyżowej.

6.4. Złożoność obliczeniowa

Poprzedni oraz aktualny rozdział stanowił opis iteracyjnego ujęcia problemu identyfikacji zagnieżdżonych struktur jednostek nazewniczych. Przedstawiony został sposób działania pojedynczego neuronowego modelu iteracyjnego oraz dwukierunkowego algorytmu iteracyjnego. Po zapoznaniu się metodami iteracyjnymi możemy przejść do rozważań na temat złożoności obliczeniowej tych metod w kontekście innych technik wykrywania struktur jednostek znanych z literatury. Niestety dyskusja na temat złożoności jest utrudniona ze względu na dwa problemy. Po pierwsze, autorzy metod rozwiązujących problem identyfikacji jednostek stosują daleko idące uproszczenia w dyskusji na temat złożoności, w szczególności skupiając się przede wszystkim na koszcie obliczeniowym części predykcyjnej modeli a pomijając koszt generowania reprezentacji cech. W przypadku metod opartych na sieciach neuronowych przygotowanie wektorowych reprezentacji elementów sekwencji może znacznie przewyższać późniejszy koszt identyfikacji encji na ich podstawie. Dlatego też w niniejszych rozważaniach złożoność obliczeniowa została rozbita na dwa komponenty - złożoność generowania reprezentacji oraz złożoność predykcji. Złożoność predykcji jest łatwa do pozyskania, najczęściej jest ona bowiem bezpośrednio raportowana przez autorów publikacji. Złożoność generowania reprezentacji natomiast została oszacowana na potrzeby niniejszej pracy na podstawie opisów architektury rozwiązań proponowanych w literaturze. Pozwala ona spojrzeć z innej perspektywy na stosowane rozwiązania i zrozumieć, że w niektórych przypadkach niższa złożoność predykcji została osiągnięta kosztem wyższej złożoności generowania reprezentacji. Drugim z problemów w dyskusji na temat złożoności jest koszt operacji podstawowej. Rząd złożoności rozpatrujemy w kategoriach liczby takich operacji w zależności od rozmiaru wejścia oraz innych parametrów danych wejściowych. Natomiast ze względu na dużą różnorodność stosowanych metod, operacje wykonywane przez poszczególne modele mogą również znacznie różnić się w zakresie rzeczywistego czasu ich wykonania. Czas ten może być uzależniony od specyfiki modelu, ale też od konkretnej implementacji czy platformy sprzętowej pod którą implementacja została przygotowana (np. CPU lub GPU). Warto zatem przypomnieć, że rząd złożoności nie odpowiada szacunkowi rzeczywistego czasu zegarowego wykonania danego algorytmu, mówi jedynie o tym w jaki sposób czas ten rośnie w zależności od parametrów danych wejściowych.

Zestawienie pesymistycznej złożoności obliczeniowej dla wybranych metod wykry-

Model	**Generowanie reprezentacji**	**Predykcja**
Płaskie tagowanie sekwencji		
BiLSTM-CRF [50]	$O(n)$	$O(n \cdot c^2)$
Metody statystyczne		
Alex et al. [3] (layering)	$O(max(d) \cdot n)$	$O(max(d) \cdot n \cdot c^2)$
Alex et al. [3] (cascading)	$O(g(c) \cdot n)$	$O(g(c) \cdot n \cdot c^2)$
Alex et al. [3] (joined label training)	$O(n)$	$O(n \cdot comb(c)^2)$
Finkel and Manning [29]	$O(n^3)$	$O(n^3)$
Metody oparte na hipergrafach		
Lu and Roth [80]	$O(c \cdot n)$	$O(c \cdot n)$
Muis and Lu [94]	$O(c \cdot n)$	$O(c \cdot n)$
Wang and Lu [132]	$O(c \cdot l \cdot n)$	$O(c \cdot l \cdot n)$
Metody neuronowe		
Xu et al. [139]	$O(l \cdot n)$	$O(c \cdot l \cdot n)$
Sohrab and Miwa [121]	$O(l \cdot n)$	$O(c \cdot l \cdot n)$
Ju et al. [54]	$O(max(d) \cdot n)$	$O(max(d) \cdot n \cdot c^2)$
Wang et al. [133]	$O(n^2)$	$O(c \cdot n)$
Lin et al. [76]	$O(c \cdot n + l \cdot k)$	$O(c \cdot n + l \cdot k)$
Xia et al. [137]	$O(l \cdot n + k \cdot n)$	$O(c \cdot k)$
Shibuya and Hovy [119]	$O(n)$	$O(c \cdot n^2)$
Pojedynczy model iteracyjny	$O(d \cdot n)$	$O(d \cdot n \cdot c^2)$
Dwukierunkowy algorytm iteracyjny	$O(d \cdot n + k)$	$O(d \cdot n \cdot c^2 + k)$

Tabela 6.2: Porównanie złożoności obliczeniowej wybranych metod wykrywania jednostek nazewniczych. Znaczenie poszczególnych symboli: n - liczba elementów w sekwencji, c - liczba możliwych klas jednostek, $g(c)$ - liczba zdefiniowanych podgrup klas jednostek, $comb(c)$ - całkowita liczba wszystkich kombinacji klas jednostek przypisanych do pojedynczego słowa w zbiorze danych, d - głębokość zagnieżdżenia jednostek w danej sekwencji, $max(d)$ - maksymalna głębokość zagnieżdżenia w zbiorze danych, l - maksymalna długość jednostki nazewniczej w zbiorze danych, k - liczba zidentyfikowanych jednostek nazewniczych.

wania jednostek nazewniczych przedstawiono w Tabeli 6.2. Jest to koszt przetworzenia pojedynczej sekwencji składającej się z n elementów, zawierającej strukturę jednostek nazewniczych należących do c możliwych klas, o głębokości struktury wynoszącej d. W porównaniu zostały uwzględnione wszystkie główne grupy ujęć opisane w Rozdziale 4. Dodano również przykład architektury stosowanej w problemach płaskiego (jednowarstwowego) wykrywania jednostek nazewniczych - model BiLSTM-CRF [50], jeden z najpopularniejszych obecnie modeli wykorzystywanych w tego typu zadaniach. Wiele spośród rozpatrywanych metod korzysta w warstwy predykcyjnej opartej na modelu linear-chain CRF. W standardowym wariancie tego modelu używany jest algorytm Viterbiego oparty na programowaniu dynamicznym, służący do znajdowania optymalnej sekwencji tagów, którego złożoność obliczeniowa wynosi $O(n \cdot c^2)$. Złożoność ta wynika z konstrukcji tablicy $\boldsymbol{T}^{(1)}$ przechowującej wyniki cząstkowe algorytmu. Sama tablica ma wymiar $n \times c$, a koszt wyliczenia każdego z jej elementów wynosi c. Stąd też można zauważyć często powtarzający się element $n \cdot c^2$ w złożoności predykcyjnej porównywanych metod. Koszt obliczenia wyniku warstwy CRF rośnie co prawda kwadratowo od liczby możliwych klas jednostek nazewniczych, ale należy mieć na uwadze fakt, że konstrukcja tablicy $\boldsymbol{T}^{(1)}$ składa się z operacji na wartościach skalarnych, jej rzeczywisty czas jest zatem akceptowalny nawet dla dużych wartości c. Złożoność kwadratowa jest bardziej problematyczna, gdy występuje na etapie generowania reprezentacji tekstu, gdzie wykonywane są głównie operacje wektorowe.

BiLSTM-CRF dla płaskiego tagowania sekwencji jest najprostszym spośród rozpatrywanych modeli. Metoda ta generuje n wektorowych reprezentacji elementów, a następnie dokonuje na ich podstawie predykcji przy pomocy warstwy CRF. Określenie złożoności neuronowego modelu iteracyjnego jest równie intuicyjne. Predykcja w modelu iteracyjnym jest równoważna d niezależnym krokom predykcji płaskiej. Mamy więc $O(n \cdot d)$ reprezentacji elementów sekwencji, po n reprezentacji na każdą iterację, oraz łącznie $O(d \cdot n \cdot c^2)$ operacji w warstwie CRF. Wynikiem działania modelu jest zbiór zawierający k jednostek nazewniczych. W dwukierunkowym algorytmie iteracyjnym funkcja selekcji dla każdej wykrytej jednostki podejmuje decyzję o uwzględnieniu jej w ostatecznych wynikach algorytmu. Mamy zatem dodatkowe k reprezentacji jednostek oraz k kroków predykcyjnych, co daje rząd złożoności $O(d \cdot n + k)$ dla generowania reprezentacji oraz $O(d \cdot n \cdot c^2 + k)$ dla predykcji.

Podobną złożoność do modelu iteracyjnego ma również metoda *layering* znana z Alex et al. [3] oraz warstwowy model neuronowy z Ju et al. [54]. Główną różnicą pomiędzy modelem iteracyjnym a tymi metodami jest jednak fakt, że liczba kroków predykcyjnych nie jest uzależniona od aktualnej głębokości struktury jednostek dla danego zdania, ale od maksymalnej głębokości struktury dla całego zbioru danych $max(d)$, modele te nie przewidują bowiem żadnego warunku wcześniejszego zatrzymania (ang. *early stopping*). Pozostałe techniki zaproponowane w Alex et al. [3] mają trudniejszą do oszacowania złożoność, w dużym stopniu uzależnioną od specyfiki zbioru danych. W metodzie *cascading* na złożoność wpływa liczba grup, na które podzielimy zbiór wszystkich klas jednostek. W *joined label training* na złożoność modelu CRF wpływa nie tyle liczba klas c, co liczba wszystkich kombinacji klas przypisanych do poszczególnych słów w zbiorze danych.

W metodach opartych na hipergrafach [80, 94, 132] oraz enumerujących modelach neuronowych [139, 121] oryginalna sekwencja słów jest przekształcana do innego ro-

dzaju struktury danych - hipergrafu lub zbioru fraz - a następnie na tej strukturze wykonywana jest predykcja. W związku z tym koszt zarówno generowania reprezentacji jak i predykcji jest ściśle uzależniony od rozmiaru powstałej struktury. Systemy złożone z dwóch modeli [76, 137] identyfikują jednostki nazewnicze etapami. Pierwszy z modeli generuje uproszczone lub cząstkowe wyniki, a kolejny na ich podstawie konstruuje ostateczny zbiór jednostek nazewniczych. W tego typu systemach każdy z modeli działa niezależnie, ich złożoność jest zatem sumą złożoności poszczególnych modeli.

6.5. Podsumowanie

W rozdziale omówione zostały zaawansowane aspekty metod iteracyjnych w problemach wykrywania hierarchicznych struktur jednostek nazewniczych. Głównym tematem rozdziału był dwukierunkowy algorytm iteracyjny, którego działanie polega na połączeniu wyników dwóch modeli iteracyjnych trenowanych w przeciwnych kierunkach przy pomocy dodatkowej funkcji selekcji. Przedstawiono sześć przykładowych funkcji selekcji: tylko inside-out, tylko outside-in, suma modeli, część wspólna modeli, selekcja oparta na pewności oraz selekcja oparta na klasyfikatorze. Motywacją do wprowadzenia predykcji dwukierunkowej było założenie, że poszczególne modele składowe mogą nauczyć się różnych wzorców w danych, co podniesie jakość predykcji całego systemu. Teza ta zostanie zweryfikowana w rozdziale eksperymentalnym, gdzie przedstawione zostaną wyniki pojedynczych modeli iteracyjnych oraz algorytmu dwukierunkowego z różnymi funkcjami selekcji. Stosowanie dwóch modeli umożliwia również alternatywną metodę propagacji stanu pomiędzy iteracjami, w której modele mogą współdzielić stan. Ten wariant algorytmu dwukierunkowego, zaproponowany w dalszej części rozdziału, nazwano algorytmem z propagacją krzyżową. Ostatnia część rozdziału poświęcona została porównaniu złożoności obliczeniowej metod iteracyjnych oraz pozostałych rozwiazań znanych z literatury. W celu dokładniejszego zobrazowania różnic, wprowadzono podział na dwa rodzaje złożoności obliczeniowej metod wykrywania jednostek nazewniczych: złożoność predykcji oraz złożoność generowania reprezentacji, która jest często pomijana w dyskusji na temat tego typu modeli.

Rozdział 7

Eksperymenty

Jednym z celów niniejszej rozprawy jest wykazanie, że zaproponowane iteracyjne ujęcie problemu wykrywania hierarchicznych struktur jednostek nazewniczych jest konkurencyjne w stosunku do innych rozwiązań znanych z literatury. Innymi słowy, poprawność predykcji, którą możemy uzyskać dzięki modelom iteracyjnym, jest porównywalna lub wyższa od stosowanych do tej pory metod. Metody te zostały omówione szczegółowo w rozdziale czwartym, natomiast rozdział piąty i szósty stanowiły wprowadzanie do ujęcia iteracyjnego, będącego zasadniczym tematem tej pracy. W tym rozdziale przedstawione zostaną wyniki eksperymentów przeprowadzonych przy pomocy modeli iteracyjnych. W celu uzyskania pełniejszego, wiarygodnego obrazu wyników modeli iteracyjnych, eksperymenty te wykonano na czterech zbiorach danych o różnej charakterystyce - różniących się między innymi językiem, domeną zawartych w nich tekstów, liczbą klas czy poziomem skomplikowania struktur jednostek nazewniczych. Dla każdego zbioru zebrane zostały również wyniki zaraportowane przez autorów innych metod. W indywidualnych sekcjach dotyczących poszczególnych zbiorów danych umieszczono dyskusję, w której zawarta jest analiza rezultatów działania modeli iteracyjnych dla konkretnego zbioru oraz porównanie z pozostałymi rozwiązaniami.

Niniejszy rozdział rozpoczyna się od informacji wstępnych na temat przeprowadzonych badań. Opisana zostanie metodyka oraz główne założenia przyjęte podczas eksperymentów. Następnie przedstawione zostaną metryki stosowane przy porównywaniu modeli oraz procedura doboru hiperparametrów dla bazowego modelu iteracyjnego. Główna część rozdziału zawiera omówienie wyników modeli iteracyjnych oraz dwukierunkowego algorytmu iteracyjnego dla różnych funkcji selekcji na każdym z użytych podczas badań zbiorów danych. W kolejnej części omówione zostaną dodatkowe eksperymenty wykonane przy wykorzystaniu dwukierunkowego algorytmu iteracyjnego z propagacją krzyżową. Ostatnią część rozdziału poświęcono na podsumowanie wszystkich eksperymentów oraz ogólne wnioski dotyczące praktycznych zastosowań modeli iteracyjnych.

7.1. Metodyka eksperymentów

Wykrywanie jednostek nazewniczych w klasycznym wariancie, w którym mamy do czynienia z płaską, jednowarstwową strukturą jednostek, jest jednym z najpopularniejszych

problemów w dziedzinie przetwarzania języka naturalnego. Popularność ta wiąże się też z dużą liczbą publicznie dostępnych zbiorów danych. Sytuacja wygląda inaczej w przypadku wykrywania zagnieżdżonych struktur jednostek nazewniczych. Ten wariant problemu zyskał na popularności dopiero w ostatnich kilku latach i na chwilę obecną istnieje jedynie kilka standardowych zbiorów, na których autorzy testują wyniki opracowanych przez siebie modeli. Na potrzeby eksperymentów opisanych w tym rozdziale, wykorzystane zostały następujące zbiory:

• **GENIA** [96] - Angielskojęzyczny zbiór bazujący na abstraktach pochodzących z publikacji o tematyce biomedycznej. Jego charakterystyczne cechy to wąska, specjalistyczna domena tekstów, stosunkowo długie jednostki nazewnicze (np. odpowiadające nazwom związków chemicznych), ale jednocześnie dosyć płytka struktura zagnieżdżeń. W większości przypadków zagnieżdżone struktury jednostek mają tylko dwie warstwy głębokości. Jest to też najstarszy i najbardziej znany sposód zbiorów stosowanych dla tego problemu, w związku z tym pełni rolę standardowego benchmarku, na którym testowanych jest większość modeli publikowanych w literaturze.

• **NNE** [112] - Angielskojęzyczny zbiór bazujący na popularnym korpusie tekstowym Penn Treebank [86]. Domena tekstów w większości składa się z fragmentów artykułów prasowych oraz dokumentów urzędowych. Jest to największy oraz najbardziej wymagający, z punktu widzenia problemu identyfikacji hierarchicznych struktur jednostek nazewniczych, spośród publicznie dostępnych zbiorów. Zawiera skomplikowane, wielowarstwowe struktury jednostek należących do ponad stu klas.

• **PolEval** [134] - Polskojęzyczny zbiór składający się z części bazującej na korpusie NKJP [102] oraz części testowej przygotowanej na potrzeby konkursu PolEval. Zawiera teksty pochodzące z różnych źródeł i domen. Pod względem poziomu skomplikowania struktur jednostek stanowi wyzwanie porównywalne do zbioru NNE. Liczba klas jednostek jest mniejsza, natomiast często tworzą one głębokie struktury zagnieżdżeń.

• **GermEval** [7] - Zbiór niemieckojęzyczny, w którego skład wchodzą fragmenty tekstów pochodzących z niemieckiej Wikipedii oraz internetowych portali informacyjnych. Pod względem struktury jednostek nazewniczych jest to najprostszy z opisanych zbiorów. Maksymalna liczba warstw jednostek wynosi dwie, natomiast większość zdań posiada prostą jednowarstwową strukturę.

Każdy z wymienionych zbiorów jest udostępniony publicznie oraz był wcześniej używany do testowania jakości predykcji metod wykrywania hierarchicznych struktur jednostek nazewniczych. Poza powyższymi pozycjami, jedynym znanym zbiorem pominiętym w eksperymentach jest ACE [25], który jest zbiorem komercyjnym, wymagającym płatnej licencji na użytkowanie.

Zgodnie z założeniami eksperymentów, każdy zbiór został podzielony na trzy części: treningową, walidacyjną oraz testową. W niektórych przypadkach taki podział został już wcześniej ustalony przez autorów danego zbioru. W tej sytuacji w eksperymentach zachowano oryginalny podział. W przypadku zbiorów, które nie zostały podzielone wcześniej, dokonano podziału wzorując się na najczęściej stosowanej w literaturze procedurze, tak aby zachować maksymalną zgodność z wynikami prezentowanymi w

innych pracach. Modele iteracyjne są uczone przy użyciu treningowej części zbioru. Rolą części walidacyjnej jest dobór wartości hiperparametrów modelu. Ta część zbioru jest stosowana również do ustalenia optymalnych progów dla funkcji selekcji opartych na poziomie pewności oraz na klasyfikatorze. Wyniki na części walidacyjnej są także używane jako kryterium oceny modelu po każdej epoce w procesie uczenia. Część testowa zbioru jest natomiast używana wyłącznie do wygenerowania ostatecznych wyników modeli, które zostały przedstawione w tym rozdziale. Informacje szczegółowe na temat wykorzystania każdego ze zbiorów zostały umieszczone w odpowiadających im podrozdziałach. Zawarto w nich także opis wykorzystanych w danym eksperymencie wektorowych reprezentacji tekstu. Jak można zauważyć, wymienione powyżej zbiory różnią się w wielu aspektach, przede wszystkim językiem oraz źródłem pochodzenia tekstów. Nie jest więc możliwe znalezienie jednego uniwersalnego modelu reprezentacji tekstu, który sprawdziłby się na każdym z nich. Dla każdego zbioru danych wybrano zatem indywidualne reprezentacje wektorowe. W niniejszej pracy skorzystano jedynie z dostępnych publicznie modeli reprezentacji tekstu, w żadnym z eksperymentów nie trenowano dodatkowo tego typu modeli.

W przypadku zbiorów GENIA oraz NNE przeprowadzono dodatkowo tzw. *ablation study*, czyli badanie polegające na ograniczaniu pewnych parametrów modelu lub usuwaniu jego części. Ewaluacja na ograniczonych wersjach modelu pomaga w zrozumieniu, które elementy zaproponowanego rozwiązania mają największy wpływ na jakość predykcji. Jest to szczególnie ważne w skomplikowanych sieciach neuronowych, w których nie jesteśmy w stanie badać indywidualnie poszczególnych komponentów systemu a jedynie model jako całość.

Wszystkie eksperymenty opisane w głównej części tego rozdziału zostały powtórzone pięciokrotnie. Oznacza to, że na każdym zbiorze wytrenowano pięć par składających się z modelu *inside-out* oraz *outside-in*. Podane wyniki modeli iteracyjnych oraz dwukierunkowego algorytmu iteracyjnego są średnią z wartości metryk wyliczonych na zbiorze testowym dla tych pięciu instancji. Podawanie uśrednionych wyników modeli ma na celu zmniejszenie wpływu czynników losowych na proces trenowania. Czynnikami tymi jest między innymi losowa inicjalizacja wag modelu przed rozpoczęciem uczenia oraz losowa kolejność próbek uczących w każdej epoce treningu. W efekcie mogą wystąpić niewielkie różnice w ostatecznych wynikach modelu na zbiorze testowym. Pojedynczy wynik mógłby zatem być zbyt optymistyczny lub zbyt pesymistyczny i nie stanowiłby podstawy do wyciągania wniosków na temat rzeczywistej efektywności ujęcia iteracyjnego. Niestety, w przypadku innych metod opisanych w literaturze takie podejście do raportowania wyników modelu jest rzadkością. Autorzy na ogół podają pojedynczy zestaw metryk bez informowania o tym ile razy dany eksperyment został powtórzony i jakie było odchylenie standardowe wyników. W związku z tym utrudnione jest porównywanie wyników modeli iteracyjnych z innymi rozwiązaniami w kontekście ich istotności statystycznej.

Dla wszystkich eksperymentów opisanych w tym rozdziale zastosowano taki sam proces trenowania modeli iteracyjnych. Każdy z modeli był uczony metodą spadku wzdłuż gradientu z mini-partiami (ang. *mini-batch gradient descent*) oraz rozmiarem partii danych wynoszącym 32 rekordy. Zastosowano również adaptacyjny współczynnik uczenia, którego wartość zmieniała się w oparciu o wyniki modelu na zbiorze walidacyjnym. W momencie rozpoczęcia uczenia przyjęto inicjalną wartość współczynnika

uczenia wynoszącą 0,1. Wartość ta była zmniejszana o połowę po każdych trzech epokach treningu, po których nie zanotowano żadnej poprawy wyników dla zbioru walidacyjnego. Proces był przerywany w momencie osiągnięcia współczynnika uczenia wynoszącego 0,0001 lub po osiągnięciu 150 epok. Po każdej epoce zapisywany był aktualny stan wag modelu. Po zakończeniu treningu do ewaluacji wybierany był stan z najlepszej epoki, czyli stan z najniższą wartością funkcji kosztu na zbiorze walidacyjnym.

7.2. Metryki jakości

W klasycznych problemach klasyfikacji lub regresji zadaniem modelu uczenia maszynowego jest przydzielenie do każdego rekordu odpowiedzi w postaci klasy lub wartości numerycznej. Ocena jakości tego typu systemów opiera się zatem na porównaniu wyników predykcji modelu dla każdej próbki z wartościami referencyjnymi. Sytuacja wygląda inaczej w przypadku zadań polegających na identyfikowania pewnych wzorców w danych, do których należy również wykrywanie jednostek nazewniczych. W tym przypadku dla pojedynczego rekordu może istnieć zero, jedna lub kilka fraz odpowiadających jednostkom nazewniczym. Podstawą dla pomiaru jakości modelu nie są więc całe próbki danych ale ich części. Bardziej konkretnie, podczas oceny modelu każdą jednostkę nazewniczą zidentyfikowaną przez model lub występującą w danych referencyjnych możemy traktować niezależne zdarzenie. Rozpatrujemy jednostki nazewnicze osobno, zliczając, które z jednostek wykrytych przez model pokrywają się z jednostkami referencyjnymi. Dokonując takiego porównania, możemy wyróżnić następujące sytuacje:

• **wynik prawdziwie pozytywny (ang. *true positive, TP*)** - Jednostka nazewnicza wykryta przez model występuje w danych referencyjnych. Klasa jednostki oraz podciąg elementów sekwencji odpowiadający jednostce są takie same.

• **wynik fałszywie pozytywny (ang. *false positive, FP*)** - Jednostka nazewnicza została wykryta przez model, ale w danych referencyjnych nie występuje jednostka tej samej klasy oraz przypisana do tych samych elementów w sekwencji.

• **wynik fałszywie negatywny (ang. *false negative, FN*)** - Jednostka nazewnicza występuje w danych referencyjnych, ale model nie wykrył jednostki tej samej klasy oraz przypisanej do tych samych elementów sekwencji.

W niektórych zadaniach z zakresu uczenia maszynowego wyróżnia się także wyniki prawdziwie negatywne (ang. *true negative, TN*), jednak w kontekście wykrywania jednostek nazewniczych nie ma jednoznacznej definicji dla takich przypadków i nie są one uwzględniane w miarach jakości. Na podstawie zagregowanej liczby opisanych powyżej zdarzeń, możemy zdefiniować miarę precyzji (ang. *precision*) oraz czułości (ang. *recall*) modelu [75]:

$$P = \frac{TP}{TP + FP}, \quad R = \frac{TP}{TP + FN} \tag{7.1}$$

gdzie TP, FP, FN są odpowiednio liczbami wystąpień wyników prawdziwie pozytywnych, fałszywie pozytywnych oraz fałszywie negatywnych. Precyzja mówi o tym, jaka część jednostek wykrytych przez model była prawidłowa. Czułość informuje natomiast o tym, jaka część spośród jednostek znajdujących się w danych referencyjnych została prawidłowo zidentyfikowana przez model. Z punktu widzenia rozpatrywanego problemu obie wartości przekazują istotną informację na temat jakości modelu. Dlatego też najczęściej wykorzystywaną w praktyce metryką oceny modeli wykrywania jednostek nazewniczych jest miara F1-score, która łączy obie te cechy. Wartość tej miary jest równa średniej harmonicznej z wartości precyzji oraz czułości [75]:

$$\mathrm{F1} = 2 \cdot \frac{P \cdot R}{P + R} \tag{7.2}$$

W przypadku problemów wieloklasowych w literaturze spotykane są dwie metody wyliczania miary F1-score różniące się sposobem agregacji wyników pochodzących z poszczególnych klas. Mikro uśrednianie (ang. *micro-averaging*) polega na zliczaniu przypadków TP, FP i FN dla wszystkich jednostek nazewniczych łącznie, bez względu na klasę, do której są przypisane. W efekcie wyliczana jest pojedyncza miara F1-score uwzględniająca wszystkie instancje jednostek. W tym sposobie uśredniania klasy jednostek częściej występujące w zbiorze danych będą miały większy wpływ na wartość metryki. Jeżeli chcemy aby wpływ klas na wartość F1-score był równy, nie zaś proporcjonalny do liczności występowania danej klasy, możemy zastosować uśrednianie typu makro (ang. *macro-averaging*). Sposób ten polega na pogrupowaniu wszystkich przypadków według klas jednostek i wyliczeniu osobnych wartości F1-score dla każdej z klas. Ostateczna wartość F1-score jest obliczana jako średnia arytmetyczna z wartości składowych.

Dla zbiorów danych uwzględnionych w tym rozdziale standardem w literaturze jest stosowanie mikro uśrednionej metryki F1-score, dlatego też porównanie wyników modeli iteracyjnych z innymi metodami wykrywania hierarchicznych struktur jednostek nazewniczych będzie się opierać na tej mierze. Jedynym odstępstwem od tej reguły jest zbiór PolEval. Zbiór ten został wykorzystany w konkursie wykrywania jednostek nazewniczych dla języka polskiego, w którym rozwiązania były oceniane przy wykorzystaniu specjalnej metryki bazującej na zmodyfikowanej wersji F1-score. Podczas oceny wykorzystano dwa warianty tej miary: *exact* oraz *overlap*. Wariant *exact* jest identyczny z opisanym powyżej sposobem wyliczania mikro uśrednionego F1-score. W wariancie *overlap* poluzowane zostały kryteria pozwalające sklasyfikować przypadek jako prawdziwie pozytywny (TP). W klasycznym podejściu za wynik prawdziwie pozytywny uznajemy tylko sytuację, kiedy obie jednostki nazewnicze, pochodzące z wyników predykcji oraz ze zbioru referencyjnego, mają tę samą klasę oraz są przypisane do tych samych elementów sekwencji. W wersji *overlap* jako wyniki prawdziwie pozytywne uznawane są nawet wyniki pokrywające się częściowo, czyli takie, w których tylko część z elementów sekwencji wchodzących w skład jednostek jest wspólna - pod warunkiem, że obie jednostki należą do tej samej klasy. Wartość *overlap* nigdy zatem nie będzie niższa od wartości *exact*. Kryterium oceny w konkursie była średnia ważona obu miar:

$$\mathrm{F1}' = 0.2 \cdot \mathrm{F1}_E + 0.8 \cdot \mathrm{F1}_O \tag{7.3}$$

gdzie F1′ jest miarą wynikową a $F1_E$ i $F1_O$ są wariantami *exact* i *overlap* metryki F1-score.

7.3. Dobór wartości hiperparametrów

W rozdziale 5 przedstawiono ogólną postać neuronowej architektury iteracyjnej. W niniejszej sekcji omówiona zostanie procedura doboru hiperparametów dla tej architektury. Liczba parametrów (wag) sieci neuronowych rozpatrywanych w tej pracy waha się od kilkudziesięciu do kilkuset milionów. Wiąże się to z wysokim kosztem wytrenowania pojedynczego modelu. Dlatego też we wstępnych eksperymentach mających na celu dobór optymalnej architektury wprowadzona założenia ograniczające liczbę niezbędnych do wytrenowania modeli. Przede wszystkim rozpatrywane są trzy rodzaje hiperparametrów mające wpływ na jakość predykcji: rodzaj zastosowanej kontekstowej reprezentacji tekstu, rozmiar komórki pamięci w pojedynczej warstwie LSTM oraz liczba warstw BiLSTM w sieci neuronowej. Te same wartości hiperparametrów zostały użyte dla wszystkich zbiorów danych, na których przeprowadzono testy - nie stosowano indywidualnych hiperparametrów sieci dla każdego ze zbiorów. Jest prawdopodobne, że wybór indywidualnych wartości hiperparametrów pozwoliłby uzyskać lepsze wyniki na poszczególnych zbiorach danych, natomiast dzięki wspólnej dla wszystkich eksperymentów architekturze łatwiejsze jest wyciąganie wniosków na temat efektywności ujęcia iteracyjnego bez dodatkowych czynników wpływających na wyniki modelu.

Selekcję hiperparametrów wykonano na zbiorze NNE [112]. Jest to największy spośród dostępnych zbiorów danych, ponadto zawiera najbardziej złożone struktury jednostek nazewniczych należących do dużej liczby kategorii, stanowiące wyzwanie dla metod wykrywania jednostek. Korpus użyty w tym zbiorze składa się z tekstów pisanych językiem potocznym, niespecjalistycznym. Zbiór można więc uznać za reprezentatywny przykład zadania identyfikacji hierarchicznych struktur jednostek nazewniczych. Podczas doboru hiperparametrów wykorzystana została tylko część treningowa oraz walidacyjna zbioru. W każdym eksperymencie trenowano osobno model inside-out oraz outside-in przy użyciu części treningowej a ewaluacji dokonywano na części walidacyjnej. Każdy rozpatrywany przypadek na etapie selekcji hiperparametrów został powtórzony trzykrotnie a zaprezentowane wyniki są średnią z trzech wartości. Przetestowane zostały trzy możliwe warstwy reprezentacji tekstu, cztery rozmiary pamięci komórki LSTM oraz cztery liczby warstw BiLSTM. Daje to 11 przypadków, dla których trzykrotnie wytrenowano po dwa modele. Łącznie w ramach selekcji hiperparametrów wytrenowano 66 sieci neuronowych.

Przed rozpoczęciem selekcji przyjęte zostały inicjalne wartości hiperparametrów. W każdym eksperymencie zmieniany był tylko jeden hiperparametr, natomiast dla pozostałych przyjmowano wartości inicjalne. Wstępna konfiguracja hiperparametrów została zaczerpnięta z artykułu Akbik et al. [1]. Autorzy zaproponowali architekturę neuronową opartą na sieciach LSTM, która okazała się wyjątkowo efektywna dla wielu zadań płaskiego wykrywania jednostek nazewniczych, osiągając najlepsze wyniki na popularnych zbiorach danych takich jak CoNLL-2003 [125], OntoNotes 5.0 [135], WNUT 2017 [23], oraz na niemieckiej i niderlandzkiej wersji zbioru CoNLL. Ze względu na podobieństwa pomiędzy płaskim tagowaniem sekwencji a metodami iteracyjnymi, dla

neuronowego modelu iteracyjnego przyjęto takie same parametry, tam gdzie było to możliwe. Inicjalna konfiguracja składała się z dwóch warstw BiLSTM, z rozmiarem pamięci wynoszącym 256 w każdej warstwie. Użyto także metody dropout dla elementów sekwencji z prawdopodobieństwem 0,05 oraz variational dropout [32] z prawdopodobieństwem 0,5 dla każdej warstwy BiLSTM.

Wartość	**Model outside-in**	**Model inside-out**
Rodzaj wektorowej reprezentacji tekstu		
Flair	**93.68**	**93.79**
BERT	92.22 (-1.46)	92.64 (-1.15)
ELMo	93.48 (-0.20)	93.48 (-0.31)
Rozmiar pamięci w warstwie LSTM		
128	92.95 (-0.83)	93.11 (-0.74)
256	**93.78**	**93.85**
384	93.63 (-0.15)	93.58 (-0.27)
512	93.53 (-0.25)	93.72 (-0.13)
Liczba warstw BiLSTM		
1	93.47 (-0.31)	93.66 (-0.07)
2	**93.78**	**93.73**
3	93.37 (-0.41)	93.66 (-0.07)
4	93.34 (-0.44)	93.49 (-0.24)

Tabela 7.1: Wyniki neuronowych modeli iteracyjnych na walidacyjnej części zbioru NNE dla różnych wartości hiperparametrów. Rozpatrywano trzy kryteria: rodzaj wektorowej reprezentacji tekstu, rozmiar pamięci w warstwie LSTM oraz liczbę warstw BiLSTM.

Wyniki procesu selekcji hiperparametrów przedstawiono w Tabeli 7.1. Optymalne wartości F1-score zostały pogrubione, dla wartości nieoptymalnych dodatkowo w nawiasie podano całkowitą różnicę w wartości tej miary w porównaniu z wartością optymalną.

W przypadku kontekstowych reprezentacji tekstu rozpatrywane były następujące modele: oryginalny duży model ELMo dla języka angielskiego [100], oryginalny duży model BERT uwzględniający wielkość znaków (*cased*) [24] oraz model Flair [1] trenowany na angielskim korpusie wiadomości prasowych. Dla neuronowego modelu iteracyjnego, najbardziej efektywna okazała się ostatnia z reprezentacji. Zaskoczeniem okazał się wyraźnie słabszy wynik reprezentacji wygenerowanych przez model BERT, które zazwyczaj osiągają konkurencyjne wyniki dla wielu zadań w zakresie przetwarzania języka naturalnego. Możliwe, że zaproponowana w tej pracy architektura iteracyjna bazująca na sieciach rekurencyjnych nie jest w pełni efektywna w połączeniu z modelem opartym o architekturę Transformera. Prawdopodobnie inne architektury sieci, dostosowane do współpracy z Transformerem, są w stanie wykorzystać potencjał tej reprezentacji tekstu. Przykładem takiej metody opartej w całości na architekturze Transformera jest model zaproponowany przez Shibuya and Hovy [119]. Przewaga reprezentacji Flair nad pozostałymi metodami może też wynikać ze specyfiki zadania identyfikacji jednostek nazewniczych, w którym duże znaczenie mają cechy oparte na

znakach. Wielkie litery lub znaki takie jak cudzysłów mogą być istotnymi atrybutami przy oznaczaniu danej frazy jako jednostki nazewniczej. Istotność cech opartych na znakach była też podkreślana przez wcześniejsze publikacje na temat tego problemu [66, 81, 109]. Flair, w przeciwieństwie do pozostałych metod, buduje reprezentację tekstu wyłącznie w oparciu o znaki.

W przypadku rozmiaru pamięci w warstwie LSTM widzimy wyraźny spadek jakości predykcji dla wartości 128, co może sugerować zbyt małą liczbę parametrów modelu. Model uzyskuje najlepsze wyniki dla wartości inicjalnej, natomiast różnice nie są aż tak znaczące pomiędzy pozostałymi wartościami. Podobnie jest w przypadku liczby warstw BiLSTM, aczkolwiek dla tego hiperparametru najniższa wartość nie powoduje tak dużego spadku jakości jak w przypadku rozmiaru pamięci. Warto zauważyć, że model outside-in wydaje się być bardziej wrażliwy na liczbę warstw niż model inside-out, dla którego różnice pomiędzy poszczególnymi wartościami są niewielkie.

Wybrane hiperparametry zostały użyte we wszystkich kolejnych eksperymentach opisanych w tym rozdziale. Neuronowe modele iteracyjne składały się z dwóch warstw BiLSTM z rozmiarem pamięci wynoszącym 256 w każdej warstwie. Tam gdzie było to możliwe, skorzystano z reprezentacji kontekstowych opartych o model Flair. Jedynie w przypadku polskojęzycznego zbioru PolEval użyty został inny rodzaj reprezentacji.

7.4. GENIA

GENIA [97] jest najstarszym i najbardziej znanym zbiorem danych dla problemu wykrywania zagnieżdżonych struktur jednostek nazewniczych, którego pojawienie się w znacznej mierze przyczyniło się do spopularyzowania tego zagadnienia wśród naukowców zajmujących się NLP. Identyfikacja jednostek nazewniczych z wykorzystaniem zbioru GENIA była równiez przedmiotem konkursu *bio-entity recognition task* organizowanego w ramach konferencji JNLPBA w 2004 roku. Zbiór zawiera około 2 tys. abstraktów pochodzących z publikacji znajdujących się w bazie MEDLINE/PubMed, gromadzącej artykuły o tematyce związanej z biologią i medycyną. W tekstach manualnie oznaczono jednostki nazewnicze należące do specjalistycznych kategorii biomedycznych takich jak terminy odnoszące się do substancji chemicznych, tkanek, komórek czy wirusów. W pierwszej zewnętrznej warstwie oznaczono 51 546 jednostek nazewniczych, z których 4 895 (9,5%) zawiera jednostki zagnieżdżone. Maksymalna głębokość zagnieżdżenia wynosi cztery, natomiast dominujące są proste struktury zagnieżdżeń z dwoma poziomami jednostek. Na potrzeby zadania wykrywania jednostek abstrakty są dzielone na pojedyncze zdania, tak by każde zdanie było odrębną próbką.

W oryginalnej wersji zbioru wyróżniono 36 kategorii jednostek nazewniczych, jednak współcześnie metody wykrywania jednostek są ewaluowane na uproszczonej wersji zbioru. W pierwszych latach po udostępnieniu zbioru GENIA w badaniach stosowane były różne procedury wstępnego przetwarzania danych, co utrudniało porównania wyników poszczególnych modeli. Obecnie w większości publikacji stosowana jest ta sama standardowa procedura, która pierwszy raz została opisana przez Finkel and Manning [29]. Aby możliwe było uczciwe porównanie wyników metod iteracyjnych z innymi rozwiązaniami z literatury, w tym eksperymencie dokonano wstępnego przetworzenia danych w dokładnie taki sam sposób. W pierwszej kolejności zbiór został podzielony

na część treningową, walidacyjną oraz testową. Pierwsze 90% próbek, liczone według kolejności ich występowania w oryginalnym zbiorze, zostało wydzielonych na potrzeby uczenia i walidacji, natomiast ostatnie 10% posłużyło jako część testowa. Większą część zbioru ponownie podzielono w proporcjach 90% do 10%, wydzielając osobno część treningową i walidacyjną. Ponadto zmniejszona została liczba klas jednostek nazewniczych. Wszystkie podkategorie należące do grupy *DNA* zostały połączone w pojedynczą kategorię *DNA*, w podobny sposób zgrupowane zostały również podkategorie należące do grup *RNA* oraz *protein*. Kategorie *cell line* oraz *cell type* nie uległy zmianie, natomiast wszystkie pozostałe klasy zostały usunięte. W efekcie w zmodyfikowanej wersji zbioru pozostało pięć klas jednostek nazewniczych.

Metoda	**P**	**R**	**F1**
Metody nie wykorzystujące sieci neuronowych			
Alex et al. [3]	74.5	66.0	70.0
Finkel and Manning [29]	75.4	65.9	70.3
Lu and Roth [80]	74.2	66.7	70.3
Muis and Lu [94]	75.4	66.8	70.8
Wang and Lu [132]	76.2	67.5	71.6
Metody wykorzystujące sieci neuronowe			
Xu et al. [139]	71.2	64.3	67.6
Katiyar and Cardie [57]	79.8	68.2	73.6
Ju et al. [54]	78.5	71.3	74.7
Wang et al. [133]	78.0	70.2	73.9
Wang and Lu [132]	77.0	73.3	75.1
Sohrab and Miwa [121]	**93.2**	64.0	77.1
Marinho et al. [87]	74.0	72.0	73.0
Lin et al. [76]	75.8	73.9	74.8
Zheng et al. [148]	75.9	73.6	74.7
Sun et al. [122]	77.4	74.9	76.2
Shibuya and Hovy [119]	78.7	75.7	77.2
Metody iteracyjne			
Tylko outside-in	75.5	79.0	77.2 ($\pm$0.18)
Tylko inside-out	78.0	75.8	76.9 ($\pm$0.20)
Część wspólna modeli	80.4	73.9	77.0 ($\pm$0.22)
Suma modeli	73.6	**80.9**	77.1 ($\pm$0.17)
Selekcja oparta na pewności	78.9	75.3	77.0 ($\pm$0.12)
Selekcja oparta na klasyfikatorze	77.8	76.9	**77.3** ($\pm$0.18)

Tabela 7.2: Porównanie wyników na zbiorze danych GENIA.

W ramach tego eksperymentu wykorzystane zostały wektorowe reprezentacje tekstu trenowane na korpusach biomedycznych. Jako warstwy reprezentacji tekstu w neuronowym modelu iteracyjnym użyto publicznie dostępnych 400 wymiarowych statycznych wektorów Word2Vec trenowanych na abstraktach z bazy MEDLINE/PubMed [89] oraz reprezentacji kontekstowych w postaci biomedycznej wersji modelu Flair [1] trenowanej na tym samym źródle danych.

Tabela 7.2 zawiera porównanie dwukierunkowego algorytmu iteracyjnego wykorzystującego różne funkcje selekcji z innymi metodami identyfikacji jednostek nazewniczych. W tabeli podano precyzję, czułość oraz wartość miary F1-score dla każdego z rozwiązań. W przypadku metod iteracyjnych podane zostały wartości uśrednione wraz z odchyleniem standardowym wyników. Spośród przetestowanych funkcji selekcji, najlepszy wynik uzyskała selekcja oparta na klasyfikatorze, będąc przy tym najbardziej zbalansowaną w kontekście stosunku wartości precyzji i czułości. Wyniki sumy oraz części wspólnej modeli są zgodne z oczekiwaniami. Część wspólna jest rodzajem selekcji nastawionym na maksymalizację wartości precyzji, wybiera bowiem tylko te jednostki, co do których oba modele są zgodne. Suma modeli natomiast maksymalizuje czułość wyników, nie pomija bowiem żadnej z jednostek wykrytych przez którykolwiek z modeli. Z punktu widzenia praktycznych zastosowań daje to pewną swobodę wyboru. W zadaniach, w których bardziej zależy nam na wyższej wartości precyzji lub czułości niż na maksymalizacji F1, można rozważyć użycie powyższych metod selekcji. Wszystkie funkcje selekcji uzyskały zbliżone wyniki, w rozpatrywanych przypadkach jedynie samodzielny model inside-out uzyskał średnią wartość F1-score niższą niż 77.0. Brak większych różnic pomiędzy rodzajami selekcji można tłumaczyć faktem, że zbiór GENIA składa się głównie z płaskich jednostek nazewniczych, a występujące w nim struktury hierarchiczne jednostek są zbyt proste, aby miały znaczący wpływ na jakość poszczególnych metod. Zaproponowany w tej pracy model wypada korzystnie w porównaniu z innymi rozwiązaniami, porównywalne wyniki uzyskują jedynie metody zaproponowane przez Shibuya and Hovy [119] oraz Sohrab and Miwa [121].

Metoda	**P**	**R**	**F1**
Pełny algorytm dwukierunkowy	77.8	76.9	77.3
Tylko outside-in	75.5	79.0	77.2 (-0.1)
Tylko inside-out	78.0	75.8	76.9 (-0.4)
Bez iteracji	77.1	76.9	77.0 (-0.3)
Bez kontekstowych reprezentacji tekstu	77.4	75.3	76.3 (-1.0)
Bez biomedycznych reprezentacji tekstu	76.4	74.8	75.6 (-1.7)

Tabela 7.3: *Ablation study* dla zbioru danych GENIA.

W celu dokładniejszego zbadania wpływu poszczególnych cech modeli iteracyjnych oraz algorytmu dwukierunkowego na jakość predykcji dla zbioru GENIA, wykonano dodatkowe eksperymenty w postaci *ablation study*. Eksperymenty polegały na ograniczeniu możliwości modeli poprzez usuwanie określonych cech lub funkcji. Wyniki tego badania pokazane zostały w Tabeli 7.3. Dla każdej pozycji pokazano wartości miar jakości uzyskiwanych przez daną metodę oraz całkowitą różnicę w wartości F1-score w porównaniu z najlepszym rozwiązaniem. Pierwsza pozycja w tabeli odpowiada pełnemu dwukierunkowemu algorytmowi iteracyjnemu z selekcją opartą na klasyfikatorze. Druga i trzecia pozycja to samodzielne modele iteracyjne typu outside-in oraz inside-out. Wyniki dla tych pozycji zostały skopiowane z poprzednich eksperymentów.

W czwartym podejściu zbadano efekt uruchomienia algorytmu dwukierunkowego bez iteracji. W tym przypadku każdy z modeli uruchomiono tylko dla jednej warstwy jednostek nazewniczych, analogicznie jak w płaskich modelach tagowania sekwencji -

najbardziej zewnętrznej warstwy dla modelu outside-in oraz najbardziej zagnieżdżonej dla modelu inside-out. Wyniki pojedynczej iteracji połączono w standardowy sposób przy użyciu funkcji selekcji opartej na klasyfikatorze. Można zauważyć spadek jakości predykcji, natomiast w przypadku zbioru GENIA nie jest on znaczny. Piąta pozycja w tabeli zawiera wyniki algorytmu iteracyjnego opartego na prostszych modelach neuronowych, pozbawionych kontekstowych reprezentacji tekstu Flair [1]. Modele w tej wersji korzystały jedynie z wymienionej wcześniej statycznej reprezentacji słów opartej na Word2Vec oraz dodatkowego enkodera znakowego, naśladując klasyczny model tagowania sekwencji BiLSTM-CRF [66]. Ostatni przypadek rozpatrywany w *ablation study* polegał na zastąpieniu wektorowych reprezentacji tekstu trenowanych na korpusach biomedycznych przez reprezentacje ogólnego przeznaczenia, trenowane na języku potocznym. W tej wersji biomedyczna wersja modelu Flair została zastąpiona wersją uczoną na korpusie wiadomości prasowych. Biomedyczny Word2Vec został natomiast zastąpiony 300 wymiarowym modelem GloVe [98]. Rozwiązania ze modyfikowaną warstwą reprezentacji tekstu uzyskują wyraźnie gorsze wyniki od oryginalnego modelu dwukierunkowego. Największy spadek F1-score możemy zaobserwować szczególnie w ostatnim przypadku, w którym wykorzystano reprezentacje pochodzące spoza domeny problemu. Sposób, w jaki budowane są reprezentacje wektorowe dla elementów sekwencji ma zatem w przypadku zbioru GENIA o wiele większe znaczenie niż podejście do wykrywania zagnieżdżonych jednostek nazewniczych. Metody wykorzystujące specjalistyczne, biomedyczne reprezentacje uzyskują znaczną przewagę nad metodami nie korzystającymi z tego typu rozwiązań. Tylko niektóre z modeli wymienionych w Tabeli 7.2 korzystają z biomedycznych reprezentacji tekstu. Poza neuronowym modelem iteracyjnym są to również modele: Sohrab and Miwa [121], Zheng et al. [148], Sun et al. [122] oraz Shibuya and Hovy [119]. Wykluczając te rozwiązania, najlepszym spośród pozostałych modeli jest Wang and Lu [132], uzyskujący F1-score na poziomie 75.1. Jak można odczytać z Tabeli 7.3, dla dwukierunkowego modelu iteracyjnego bez reprezentacji biomedycznych wartość tej metryki wynosi 75.6, jest ona zatem wciąż wyższa od porównywalnych rozwiązań.

7.5. NNE

Nested Names Entities (NNE) jest dużym angielskojęzycznym zbiorem danych udostępnionym wraz z publikacją Ringland et al. [112], który powstał w celu ewaluacji metod wykrywania hierarchicznych struktur jednostek nazewniczych. Jego cechą charakterystyczną są złożone struktury jednostek o wielu poziomach zagnieżdżenia. Do konstrukcji zbioru wykorzystano część Wall Street Journal korpusu Penn Treebank [86], w związku z tym dominującym rodzajem tekstów w zbiorze są typowe artykuły informacyjne. W zewnętrznej warstwie oznaczonych zostało 118 525 jednostek nazewniczych, natomiast łączna liczba wszystkich otagowanych jednostek razem z jednostkami zagnieżdżonymi wynosi 279 795. Ponad 60% zewnętrznych jednostek posiada jednostki zagnieżdżone, a maksymalna liczba warstw w strukturach zagnieżdżonych to sześć. W zbiorze wykorzystano również szczegółową taksonomię jednostek, w której skład wchodzi 112 kategorii encji. Najczęściej spotykane w zbiorze jednostki nazewnicze przypisano do kategorii typowych dla artykułów prasowych. Są to między innymi

nazwy odnoszące się do ludzi, organizacji, obiektów geograficznych, daty i czasu, czy różne rodzaje wartości numerycznych (np. kwoty w walucie). Same kategorie są zatem łatwiejsze do zidentyfikowania niż w przypadku bardziej specjalistycznych zbiorów takich jak GENIA, natomiast wyzwanie stanowi ich duża liczba oraz poziom skomplikowania struktur jednostek występujących w danych referencyjnych. Dane nie wymagają szczególnej procedury wstępnego przetwarzania. Udostępniony zbiór jest już podzielony na próbki w postaci pojedynczych zdań oraz wyodrębniono w nim standardową część treningową, walidacyjną i testową. W niniejszej pracy skorzystano więc z podziału zaproponowanego przez autorów.

W tym eksperymencie wykorzystano wektorowe reprezentacje tekstu uczone na angielskojęzycznych korpusach składających się głównie z tekstów w języku potocznym. Jako reprezentacji statycznej użyto 300 wymiarowego modelu GloVe [98] trenowanego na angielskiej Wikipedii oraz korpusie Gigaword. Wykorzystano także kontekstowe reprezentacje pochodzące z modelu Flair [1] trenowanego na korpusie wiadomości prasowych.

Metoda	**P**	**R**	**F1**
Metody wykorzystujące sieci neuronowe			
Baseline (zewnętrzne) [112] †	89.9	38.0	53.5
Baseline (wewnętrzne) [112] †	93.8	62.0	74.7
Baseline (kombinacja) [112] †	92.2	85.8	88.9
Wang et al. [133] †	77.4	70.1	73.6
Wang and Lu [132] †	91.8	91.0	91.4
Zheng et al. [148] ‡	89.3	88.5	88.9
Ju et al. [54] ‡	92.3	90.0	91.1
Shibuya and Hovy [119] ‡	93.3	93.7	93.5
Metody iteracyjne			
Tylko outside-in	93.7	93.0	93.3 (±0.12)
Tylko inside-out	93.4	93.8	93.6 (±0.07)
Część wspólna modeli	**95.0**	91.5	93.2 (±0.16)
Suma modeli	92.1	**95.2**	93.6 (±0.11)
Selekcja oparta na pewności	93.9	93.9	93.9 (±0.13)
Selekcja oparta na klasyfikatorze	94.2	93.9	**94.1** (±0.08)

Tabela 7.4: Porównanie wyników na zbiorze danych NNE. Rozwiązania *baseline* zawarte w tabeli odnoszą się do modeli neuronowych bazujących na architekturze BiLSTM-CRF [66]: pojedynczego modelu tagowania sekwencji wykrywającego tylko zewnętrzną warstwę jednostek nazewniczych, takiego samego modelu wykrywającego warstwę wewnętrzną oraz połączenia obu modeli. Symbolem † oznaczono metody przetestowane przez Ringland et al. [112], natomiast symbolem ‡ metody przetestowane w ramach tej pracy.

Wyniki metod iteracyjnych w porównaniu z innymi rozwiązaniami przedstawiono w Tabeli 7.4. Z uwagi na fakt, że zbiór NNE został udostępniony niedawno, lista przetestowanych na nim metod jest krótsza niż w przypadku zbioru GENIA. Część z zaprezentowanych w tabeli wyników zaczerpnięto z publikacji Ringland et al. [112]. Au-

torzy przetestowali trzy proste metody bazowe (ang. *baselines*) oparte na standardowej architekturze neuronowej BiLSTM-CRF [66]. Pierwsza z metod polegała na zastosowaniu płaskiego tagowania sekwencji dla najbardziej zewnętrznej warstwy jednostek, w drugiej zastosowano to samo podejście, ale dla najbardziej wewnętrznej warstwy jednostek. Trzecia z metod bazowych stanowiła połączenie dwóch poprzednich - zbiór jednostek wynikowych utworzono z sumy wyników predykcji pierwszego i drugiego modelu. Poza prostymi modelami, autorzy dokonali też ewaluacji innnych rozwiązań: neuronowego modelu przejść pomiędzy stanami Wang et al. [133] oraz modelu opartego na reprezentacji hipergrafowej Wang and Lu [132]. Lista przetestowanych metod została ponadto poszerzona o trzy kolejne modele neuronowe: model warstwowy Ju et al. [54], model Zheng et al. [148] wykorzystujący multi-task learning oraz model Shibuya and Hovy [119] oparty na zmodyfikowanym algorytmie Viterbiego. Ewaluację tych trzech modeli wykonano w ramach eksperymentów opisanych w tej pracy, na podstawie kodów źródłowych udostępnionych publicznie przez ich autorów.

Podobnie jak w przypadku zbioru GENIA, najwyższą uśrednioną wartość F1-score udało się uzyskać przy pomocy funkcji selekcji opartej na klasyfikatorze. Jest to zarazem jedyna spośród przetestowanych metod, dla której wartość tej miary przekroczyła 94. Za zbiorze NNE widać też wyraźniejszą przewagę tego rodzaju selekcji nad stosowaniem indywidualnych neuronowych modeli iteracyjnych, które uzyskują istotnie gorsze wyniki. Porównując ujęcie iteracyjne do innych rozwiązań, metoda Shibuya and Hovy [119] uzyskała wynik o 0,6 punktu gorszy od najlepszej funkcji selekcji. Pozostałe rozwiązania okazały się natomiast zdecydowanie gorsze, uzyskując o co najmniej kilka punktów niższe wyniki.

Metoda	**P**	**R**	**F1**
Pełny algorytm dwukierunkowy	94.2	93.9	94.1
Tylko outside-in	93.7	93.0	93.3 (-0.8)
Tylko inside-out	93.4	93.8	93.6 (-0.5)
Bez iteracji	94.3	86.4	90.2 (-3.9)
Bez kontekstowych reprezentacji tekstu	93.7	92.9	93.3 (-0.8)

Tabela 7.5: *Ablation study* dla zbioru danych NNE.

Na tym zbiorze również wykonane zostało *ablation study*. Wyniki tego badania przedstawione zostały w Tabeli 7.5. Rozpatrywane były przypadki analogiczne do zastosowanych na zbiorze GENIA, poza ostatnim dotyczącym biomedycznych reprezentacji tekstu, który nie miał zastosowania dla zbioru NNE. Pierwsze trzy pozycje w tabeli dotyczą wyników dwukierunkowego algorytmu iteracyjnego z funkcją selekcji opartą na klasyfikatorze oraz dwóch samodzielnych neuronowych modeli iteracyjnych. Przypadek czwarty to algorytm bez iteracji, łączący wyniki predykcji pojedynczej warstwy *inside-out* oraz *outside-in*. Ostatnia pozycja to dwukierunkowy algorytm iteracyjny łączący uproszczone modele iteracyjne inspirowane architekturą BiLSTM-CRF [66], pozbawione kontekstowych reprezentacji tekstu. Możemy zaobserwować, że wyniki modyfikacji modeli iteracyjnych dla zbioru NNE w istotny sposób różnią się od tych uzyskanych na zbiorze GENIA. Największy spadek jakości predykcji jest zauważalny dla wariantu bez iteracji, który uzyskuje F1-score o prawie cztery punkty niższy

od pełnej wersji algorytmu. Wynik ten nie jest zaskakujący, bowiem struktury jednostek nazewniczych w tym zbiorze są głębsze i bardziej złożone, a wiele jednostek nazewniczych znajduje się w wewnętrznych warstwach, które w wersji bez iteracji są pomijane. Inne modyfikacje algorytmu mają mniejszy wpływ na wynik. Nawet model bez reprezentacji kontekstowych lub pojedyncze modele iteracyjne uzyskują wyniki porównywalne z modelem Shibuya and Hovy [119] i wyższe niż pozostałe rozwiązania z literatury.

7.6. PolEval

PolEval jest cyklicznie organizowanym wydarzeniem inspirowanym konferencją SemEval[1], podczas którego naukowcy mogą rywalizować w konkursach związanych z przetwarzaniem języka polskiego. W każdej edycji ogłaszanych jest kilka problemów, dla których organizatorzy udostępniają zbiory danych oraz przeprowadzają ocenę rozwiązań w oparciu o ustaloną przez siebie procedurę ewaluacji. Zadanie związane z identyfikacją hierarchicznych struktur jednostek nazewniczych było częścią edycji PolEval trwającej od czerwca do sierpnia 2018 roku. Zbiór danych użyty w konkursie składał się z dwóch części: treningowej opartej na ręcznie anotowanym podkorpusie milionowym Narodowego Korpusu Języka Polskiego (NKJP) [102] oraz testowej przygotowanej specjalnie na potrzeby tego konkursu. Dla obu części zastosowano ten sam zestaw 14 kategorii jednostek nazewniczych, z czego 6 stanowi kategorie główne a 8 kategorie podrzędne. Kategorie te odnoszą się do prawdziwych i fikcyjnych osób, organizacji, obiektów geograficznych i geopolitycznych, struktur zbudowanych przez człowieka, fraz związanych z datą i czasem. Jednostki nazewnicze należące do głównych kategorii mogą występować zarówno w warstwie zewnętrznej jak i w postaci jednostek zagnieżdżonych, natomiast jednostki z kategoriami podrzędnymi tylko w warstwach zagnieżdżonych, jako uszczegółowienie jednostki nadrzędnej. Zbiór danych składa się łącznie z 121 707 zdań, w których oznaczono 127 937 jednostek nazewniczych. Około 42% jednostek zewnętrznych posiada jednostki zagnieżdżone, a maksymalna głębokość zagnieżdżenia wynosi 6. W korpusie znajdują się też przypadki długich jednostek nazewniczych - najdłuższe frazy liczą do 16 słów. Pod względem poziomu złożoności struktur hierarchicznych zbiór ten jest zatem zbliżony do zbioru NNE opisywanego w poprzednim podrozdziale. Składa się z tekstów o różnym pochodzeniu, między innymi literatury (współczesnej oraz historycznej), reportaży, artykułów prasowych, tekstów naukowych, dokumentów prawniczych, dyskusji internetowych czy transkrypcji mowy.

Ze względu na fakt, że w zbiorze PolEval nie została wydzielona standardowa część walidacyjna, na potrzeby opisanych w tej sekcji eksperymentów do walidacji wybrano losowo 10% zdań z części treningowej zbioru. Część testową pozostawiono bez zmian. Wykorzystane zostały również polskojęzyczne wektorowe reprezentacje tekstu. Do tego celu użyto polskich modeli udostępnionych w ramach publikacji Dadas [19], konkretnie 100 wymiarowego modelu Word2Vec oraz kontekstowych reprezentacji opartych na architekturze ELMo.

W tabeli 7.6 pokazane zostały wyniki modeli iteracyjnych w porównaniu do trzech najlepszych rozwiązań z konkursu PolEval [9, 85, 84] oraz dwóch wariantów modelu za-

[1] `https://en.wikipedia.org/wiki/SemEval`

Metoda	**$F1_E$**	**$F1_O$**	**F1'**
Metody nie wykorzystujące sieci neuronowych			
Marcińczuk et al. [84]	77.8	81.8	81.0
Metody wykorzystujące sieci neuronowe			
Marcińczuk et al. [85]	82.2	85.9	85.1
Borchmann et al. [9]	82.6	87.7	86.6
Dadas [19] (Word2Vec)	80.2	85.7	84.6
Dadas [19] (ELMo)	84.5	88.8	87.9
Metody iteracyjne			
Tylko outside-in	87.5	89.4	89.0 ($\pm$0.09)
Tylko inside-out	87.1	89.1	88.7 ($\pm$0.17)
Część wspólna modeli	87.7	89.6	89.2 ($\pm$0.25)
Suma modeli	86.7	88.5	88.1 ($\pm$0.13)
Selekcja oparta na pewności	**87.8**	**89.7**	**89.3** ($\pm$0.13)
Selekcja oparta na klasyfikatorze	**87.8**	**89.7**	**89.3** ($\pm$0.18)

Tabela 7.6: Porównanie wyników na zbiorze danych PolEval. $F1_E$ oznacza wartość F1-score dla dokładnej zgodności jednostek nazewniczych, $F1_O$ dla zgodności częściowej, a F1′ średnią ważoną z tych dwóch wartości ($0.2 \cdot F1_E + 0.8 \cdot F1_O$).

proponowanego w Dadas [19], jednego wykorzystującego tylko statyczną reprezentację tekstu oraz drugiego opartego na reprezentacji kontekstowej ELMo. Jak zostało wspomniane wcześniej w sekcji dotyczącej metryk jakości, oceny poszczególnych rozwiązań w konkursie dokonywano nie w oparciu o klasyczną miarę F1-score, ale w oparciu o jej zmodyfikowaną wersję (F1′) będącą średnią ważoną z F1-score liczonego według dokładnej zgodności jednostek nazewniczych (*exact*, $F1_E$) oraz zgodności częściowej (*overlap*, $F1_O$). Dlatego też, aby móc porównać metody iteracyjne z pozostałymi modelami, w opisanych w tej sekcji eksperymentach użyte zostały te same metryki. Pojedyncze neuronowe modele iteracyjne oraz wszystkie warianty dwukierunkowego algorytmu iteracyjnego uzyskały wynik istotnie wyższy niż dotychczas stosowane metody. W rozbiciu na metryki składowe, można zauważyć kilkupunktową przewagę modeli iteracyjnych dla wartości *exact*, natomiast dla wartości *overlap* wynosi ona około jednego punktu w porównaniu z najlepszym modelem z literatury. Warto również zwrócić uwagę, że w ramach funkcji selekcji najgorzej spisała się suma modeli, uzyskując wartość F1′ o ponad jeden punkt niższą niż pozostałe funkcje. Selekcje oparte na współczynniku pewności oraz na klasyfikatorze uzyskały najwyższe uśrednione wyniki predykcji, natomiast niewiele gorsza okazała się część wspólna modeli. W przypadku tego zbioru danych szczególnie ważny jest odpowiedni balans pomiędzy wartością dokładności a czułości. Maksymalizacja czułości powoduje zbyt duży spadek dokładności, co można zauważyć po wyraźnie gorszym wyniku sumy modeli.

7.7. GermEval

GermEval jest konferencją analogiczną do SemEval, skupiającą się na problemach związanych z przetwarzaniem języka niemieckiego. W 2014 roku w ramach tego wydarze-

nia organizowany był konkurs dotyczących wykrywania jednostek nazewniczych. Na potrzeby konkursu stworzono zbiór danych składający się z 31 297 zdań, w których oznaczono jednostki nazewnicze należące do 12 kategorii [7]. Z punktu widzenia identyfikacji hierarchicznych struktur jednostek jest to dosyć prosty zbiór, nie stanowiący dużego wyzwania dla tego typu metod. Maksymalna liczba warstw jednostek została ograniczona do dwóch. Ponadto, spośród 37 832 zewnętrznych jednostek tylko około 8% (3 173) posiada jednostki zagnieżdżone. Maksymalna długość pojedynczej jednostki w zbiorze wynosi 20 słów. Zbiór został zbudowany w oparciu o teksty pochodzące z niemieckojęzycznej Wikipedii oraz niemieckich portali informacyjnych. Zdefiniowane kategorie są typowe dla problemów identyfikacji jednostek i uwzględniają między innymi nazwy odnoszące się do lokalizacji, organizacji, ludzi, oraz dodatkową kategorię innych nazw własnych, nie pasujących do żadnej z podstawowych klas.

W udostępnionej wersji zbioru został już uwzględniony podział na część treningową, walidacyjną i testową. W opisanych eksperymentach zastosowano się do tego podziału. Ponadto wykorzystane zostały niemieckojęzyczne modele reprezentacji tekstu. Jako reprezentacji statycznej skorzystano z 300 wymiarowych niemieckich wektorów FastText [35], natomiast niemiecka wersja modelu Flair [1] została użyta jako reprezentacja kontekstowa.

Metoda	**P**	**R**	**F1**
Uczestnicy GermEval 2014			
ExB Group (pierwsze miejsce) [43]	78.1	74.8	76.4
UKP (drugie miejsce) [110]	79.5	71.1	75.1
MoSTNER (trzecie miejsce)	79.2	65.3	71.6
Pozostałe metody			
Sohrab and Miwa [121]	75.0	60.8	67.2
Ju et al. [54]	72.9	61.5	66.7
Zheng et al. [148]	74.5	69.1	71.7
Metody iteracyjne			
Tylko outside-in	83.0	82.9	83.0 (±0.15)
Tylko inside-out	82.3	83.2	82.8 (±0.10)
Część wspólna modeli	**86.6**	80.6	**83.5** (±0.09)
Suma modeli	79.3	**85.4**	82.2 (±0.16)
Selekcja oparta na pewności	86.2	80.9	**83.5** (±0.13)
Selekcja oparta na klasyfikatorze	86.4	80.6	83.4 (±0.17)

Tabela 7.7: Porównanie wyników na zbiorze danych GermEval.

Wyniki ewaluacji metod iteracyjnych na zbiorze GermEval pokazano w tabeli 7.7. W zestawieniu uwzględniono również trzy najwyżej ocenione rozwiązania z konkursu GermEval 2014 oraz trzy inne metody z literatury, których autorzy wykonali ewaluację na tym zbiorze [121, 54, 148]. Modele biorące udział w konkursie oceniane były w oparciu o standardową miarę F1-score. W tabeli podano precyzję, czułość oraz F1-score każdego z rozwiązań a dla metod iteracyjnych dodatkowo odchylenie standardowe uśrednionych wartości F1-score. Najlepszy z wariantów dwukierunkowego algorytmu iteracyjnego osiągnął wartość tej miary na poziomie 83.5, ponad siedem punktów wię-

cej od zwycięzców konkursu oraz o kilkanaście punktów więcej od metod opisanych w literaturze. Zbliżone wyniki uzyskały funkcje selekcji oparte na części wspólnej modeli, na pewności oraz na klasyfikatorze. Po raz kolejny niższą jakość predykcji można zaobserwować dla funkcji selekcji opartej na sumie modeli, która uzyskuje F1-score gorszy niż samodzielne modele iteracyjne. Natomiast z uwagi na fakt, iż ten zbiór charakteryzuje się najprostszymi strukturami jednostek nazewniczych spośród wszystkich zbiorów rozpatrywanych w tej pracy, trudno jest wyciągać na jego podstawie jednoznaczne wnioski na temat wpływu poszczególnych funkcji selekcji. Dużą przewagę modeli iteracyjnych nad pozostałymi metodami w tym przypadku można tłumaczyć wykorzystaniem bardziej efektywnej reprezentacji tekstu, bowiem modele oparte o reprezentację Flair [1] uzyskiwały wysoką jakość predykcji również dla innych zadań NLP w języku niemieckim.

7.8. Algorytm z propagacją krzyżową

Poprzednie sekcje opisują wyniki eksperymentów wykorzystujących klasyczny wariant dwukierunkowego algorytmu iteracyjnego, w którym każdy z modeli iteracyjnych dokonuje niezależnych predykcji. W rozdziale 6 wskazano alternatywne podejście do predykcji dwukierunkowej. Jego charakterystyczną cechą jest zmiana sposobu propagacji stanu pomiędzy poszczególnymi iteracjami modeli - w tym podejściu stan przekazywany jest również krzyżowo. Oznacza to, że model iteracyjny otrzymuje na wejściu nie tylko zakodowany wektor stanu pochodzących z wyniku poprzedniej iteracji tego samego modelu, ale także wektor stanu drugiego z modeli. Ten sposób wymiany informacji pomiędzy modelami może potencjalnie przyczynić się do zwiększenia poprawności wyników algorytmu dwukierunkowego, narzuca jednak pewne ograniczenia. Modele uczone w trybie propagacji krzyżowej stają się od siebie zależne i nie mogą być stosowane samodzielnie jak ma to miejsce w przypadku klasycznego algorytmu dwukierunkowego. Celem niniejszej sekcji jest odpowiedź na pytanie czy propagacja krzyżowa przynosi rzeczywiste korzyści w kontekście poprawności wykrywania jednostek nazewniczych i czy korzyści te są na tyle istotne, że warto jest ją stosować pomimo wskazanych wcześniej wad.

W kontekście mechanizmu propagacji w dwukierunkowym algorytmie iteracyjnym możemy również rozpatrzyć możliwość wprowadzenia dodatkowej operacji dropout na zakodowanym wektorze stanu modelu. Dropout jest prostą techniką zaproponowaną przez Hinton et al. [47] mającą na celu zwiększenie generalizacji oraz zapobieganie przeuczeniu (ang. *overfitting*) sieci neuronowych. Głównym założeniem dropoutu jest tymczasowe wyłączanie (usuwanie) w sposób losowy neuronów z warstw sieci lub niektórych wartości z danych wejściowych podczas trenowania. Dropout jest popularną operacją stosowaną współcześnie w wielu architekturach neuronowych. Jak zostało wspomniane w sekcji dotyczącej hiperparametrów, model iteracyjny w domyślnej konfiguracji również wykorzystuje dropout na wektorowych reprezentacjach tekstu oraz w warstwach BiLSTM. Propagacja krzyżowa wprowadza do każdego z modeli dodatkową zależność od wektora stanu drugiego modelu. Uzasadnionym pomysłem wydaje się zatem wprowadzenie dodatkowej operacji regularyzującej aby zapobiec zbudowaniu zbyt silnych zależności pomiędzy modelami. Dlatego też w opisanych poniżej eks-

perymentach uwzględniono nowy hiperparametr zwany *dropoutem stanu*. Podobnie jak w przypadku innych rodzajów dropoutu, wartość tego hiperparametru wyznacza prawdopodobieństwo usunięcia elementu z wektora. Mówiąc konkretniej, przy aktywnej operacji dropout każdy niezerowy bit w wektorze stanu jest ustawiany na zero z prawdopodobieństwem równym wartości hiperparametru. Wyjątkiem jest flaga *(INIT)*, której wartość w wektorze stanu nie ulega zmianie. W przypadku propagacji krzyżowej ten rodzaj dropoutu jest stosowany na wektorach stanu pochodzących z obu modeli.

Metoda	**GENIA**	**NNE**	**PolEval** †	**GermEval**
Dropout stanu = 0.0				
Tylko outside-in	77.2 (±0.18)	93.3 (±0.12)	89.0 (±0.09)	83.0 (±0.15)
Tylko inside-out	76.9 (±0.20)	93.6 (±0.07)	88.7 (±0.17)	82.8 (±0.10)
Propagacja prosta	**77.3** (±0.18)	**94.1** (±0.08)	89.3 (±0.18)	83.5 (±0.17)
Propagacja krzyżowa	76.9 (±0.13)	**94.1** (±0.07)	89.4 (±0.12)	83.5 (±0.13)
Dropout stanu = 0.5				
Tylko outside-in	76.9 (±0.15)	93.2 (±0.13)	89.0 (±0.09)	83.0 (±0.21)
Tylko inside-out	76.3 (±0.21)	93.5 (±0.06)	88.8 (±0.09)	82.7 (±0.19)
Propagacja prosta	76.8 (±0.17)	94.0 (±0.05)	89.2 (±0.02)	**83.6** (±0.19)
Propagacja krzyżowa	77.2 (±0.13)	**94.1** (±0.04)	**89.5** (±0.11)	83.5 (±0.12)

Tabela 7.8: Porównanie wyników F1-score propagacji prostej i krzyżowej na czterech zbiorach danych. † W przypadku zbioru PolEval podano wartości zmodyfikowanej metryki F1-score użytej w konkursie PolEval. Sposób jej obliczania został opisany w sekcji dotyczącej metryk.

W celu zbadania wpływu propagacji krzyżowej na wyniki działania algorytmu iteracyjnego, przeprowadzono dodatkowe eksperymenty na każdym ze zbiorów danych opisanych w poprzednich podrozdziałach. Stosowana wcześniej wersja algorytmu dwukierunkowego opierała się na propagacji prostej bez operacji dropoutu stanu. W ramach dodatkowych eksperymentów przetestowano trzy inne warianty algorytmu: propagacja prosta z dropoutem, propagacja krzyżowa z dropoutem oraz propagacja krzyżowa bez dropoutu. Podobnie jak we wcześniejszych eksperymentach, każdy przypadek został powtórzony pięciokrotnie a przedstawione wartości metryk są wynikami uśrednionymi. W ramach tego porównania wykorzystanych zostało łącznie 160 sieci neuronowych (dwa modele iteracyjne w każdym indywidualnym przypadku, cztery warianty algorytmu dwukierunkowego, cztery zbiory danych, pięć powtórzeń).

Podsumowanie uzyskanych wyników przedstawiono w tabeli 7.8. Tabela zawiera uśrednione wartości oraz odchylenia standardowe F1-score dla samodzielnych modeli *inside-out* oraz *outside-in*, algorytmu dwukierunkowego z propagacją prostą oraz algorytmu dwukierunkowego z propagacją krzyżową. W przypadku wariantów dwukierunkowych podane wyniki odpowiadają funkcji selekcji, która uzyskała najwyższą wartość F1-score dla danej pozycji, w większości sytuacji była to funkcja oparta na klasyfikatorze. W górnej części tabeli znajdują się modele nie wykorzystujące dropoutu stanu, w dolnej części natomiast modele, dla których wartość dropoutu stanu została ustalona na 0.5. W pierwszej kolejności warto zwrócić uwagę na efekt tego hiperparametru na działanie poszczególnych wariantów iteracyjnych. Wprowadzenie tego hiperparametru

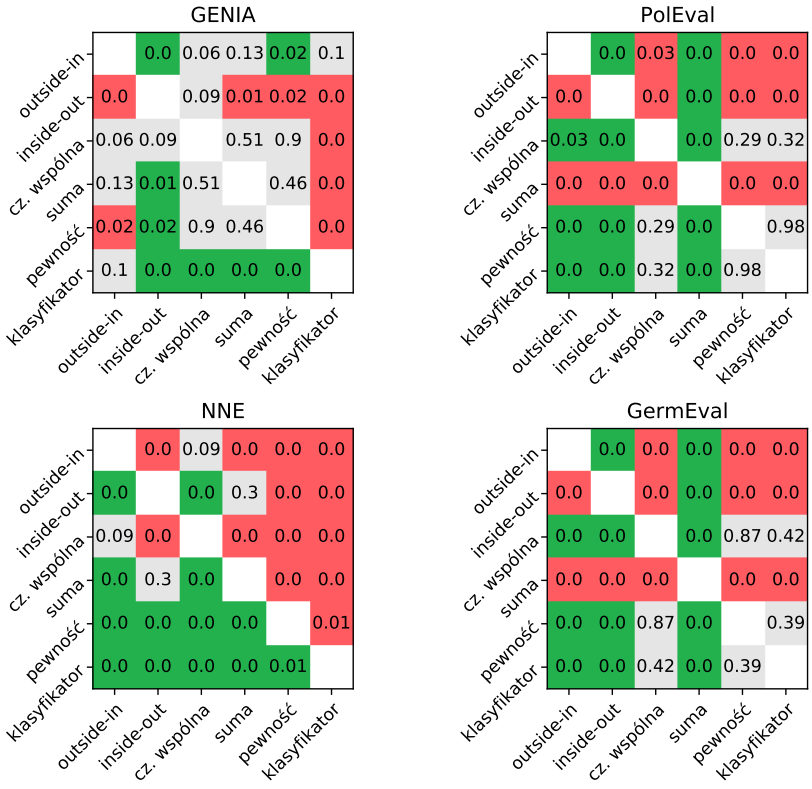
miało negatywny wpływ na samodzielne modele oraz algorytm z propagacją prostą, natomiast na ogół poprawiało wyniki algorytmu z propagacją krzyżową. Efekt ten jest szczególnie widoczny na zbiorze GENIA, gdzie różnice pomiędzy parami przypadków różniących się dropoutem sięgają 0.6 punktu. Dla pozostałych zbiorów wpływ tego hiperparametru jest mniej zauważalny. Dropout stanu ma zatem sens wyłącznie w przypadku propagacji krzyżowej, dla której wyniki na każdym ze zbiorów danych są co najmniej takie same lub wyższe po zastosowaniu tego rodzaju regularyzacji.

Porównując najlepsze wyniki uzyskane przez algorytm z propagacją prostą i krzyżową, nie widać istotnych różnic, które wskazywałyby na przewagę którejś z metod propagacji stanu. Wykonano testy t-Studenta dla prób niezależnych z poziomem istotności α równym 0.05, porównujące wyniki eksperymentów dla propagacji prostej bez dropoutu oraz propagacji krzyżowej z dropoutem, osobno na każdym ze zbiorów danych. Dla zbiorów NNE oraz GermEval różnice okazały się nieistotne statystycznie. Istotne statystycznie różnice uzyskazno w przypadku zbioru GENIA, gdzie lepiej sprawdziło się podejście z propagacją prostą, oraz w przypadku zbioru PolEval, gdzie wyższy wynik uzyskała propagacja krzyżowa. Biorąc pod uwagę wyższy poziom złożoności propagacji krzyżowej oraz opisane wcześniej trudności w używaniu tego wariantu algorytmu dwukierunkowego, można przyjąć, że nie ma uzasadnionych podstaw do jego stosowania. Algorytm oparty o propagację prostą gwarantuje podobną jakość predykcji, będąc jednocześnie łatwiejszym w implementacji, oferując lepszą współbieżność oraz niezależność modeli iteracyjnych.

7.9. Dyskusja

W rozdziale omówione zostały eksperymenty, których celem były zbadanie efektywności ujęcia iteracyjnego oraz neuronowego modelu iteracyjnego w praktycznych zadaniach wykrywania hierarchicznych struktur jednostek nazewniczych. Do badań wybrano cztery dostępne publicznie zbiory danych, które są powszechnie stosowane do ewaluacji metod rozwiązujących ten problem. Zbiory te posiadają różną charakterystykę, pokrywając szeroki zakres wariantów problemu identyfikacji jednostek. Umożliwiło to przetestowanie metod iteracyjnych na tekstach różniących się między innymi językiem (polski, angielski, niemiecki), stylem (od tekstów specjalistycznych po potoczne) czy źródłem pochodzenia (publikacje naukowe, artykuły prasowe, literatura, treści pochodzące z Internetu). Zbiory różniły się także pod względem struktur jednostek nazewniczych, od prostych przypadków obejmujących kilka klas i dwie warstwy zagnieżdżenia do ponad stu klas i złożonych, wielowarstwowych struktur jednostek.

Eksperymenty te wykazały, że zaproponowane w tej pracy metody iteracyjne mogą z powodzeniem konkurować z innymi rozwiązaniami znanymi z literatury. Na wszystkich zbiorach danych przetestowane zostały zarówno pojedyncze modele iteracyjne uczone do wykrywania jednostek w kierunku *outside-in* oraz *inside-out*, jak również dwukierunkowy algorytm iteracyjny wraz z sześcioma funkcjami selekcji. W przypadku każdego z rozpatrywanych zadań algorytm dwukierunkowy charakteryzował się poprawnością wyników porównywalną lub wyższą od stosowanych wcześniej rozwiązań. Zastosowanie predykcji dwukierunkowej poprzez połączenie wyników dwóch modeli iteracyjnych okazało się być skutecznym sposobem na zwiększenie jakości predykcji. Dzięki podejściu



Rysunek 7.1: Porównanie uśrednionych wyników F1-score uzyskanych przez poszczególne funkcje dla dwukierunkowego algorytmu iteracyjnego. Kolor komórek oraz liczby podane w komórkach reprezentują wynik testu t-Studenta dla prób niezależnych odpowiadający porównaniu pary funkcji selekcji.

dwukierunkowemu udało się uzyskać wyższe wyniki F1-score niż w przypadku samodzielnych modeli iteracyjnych. W niektórych sytuacjach użycie niezależnych modeli iteracyjnych może być jednak wystarczające. Jakość ich predykcji nie ustępuje bowiem innym metodom opisanym w literaturze, a zysk wynikający z predykcji dwukierunkowej często oscylował w granicach 1%.

W osobnym eksperymencie dokonano również porównania klasycznej wersji algorytmu dwukierunkowego z zaproponowanym w 6 rozdziale algorytmem z krzyżową propagacją stanu. W wyniku tego porównania można wnioskować, że oba warianty algorytmu uzyskują zbliżone wyniki, zatem stosowanie bardziej skomplikowanej propagacji krzyżowej nie przynosi istotnych korzyści.

Innym aspektem, na który warto zwrócić uwagę w kontekście dyskusji o ujęciu iteracyjnym, są różnice pomiędzy zastosowanymi w algorytmie dwukierunkowym funkcjami selekcji. W celu dokładniejszego porównania wpływu funkcji selekcji na jakość wykrywania jednostek, przeprowadzono dodatkową serię testów t-Studenta dla prób niezależnych, porównując parami średnie wyniki F1-score uzyskane przez poszczególne funkcje na każdym ze zbiorów danych. Dla testów przyjęto poziom istotności α równy 0.05. Wyniki testów przedstawione zostały w formie graficznej na Rysunku 7.1. Składa się on z czterech tabel odpowiadających zbiorom danych, na których przeprowadzano eksperymenty. Wiersze oraz kolumny każdej tabeli odpowiadają natomiast wynikom

poszczególnych funkcji selekcji dla konkretnego zadania. Każda komórka reprezentuje wynik testu pomiędzy parą funkcji, natomiast liczba w tej komórce odpowiada wartości p-value dla tego konkretnego testu zaokrąglonej do dwóch miejsc po przecinku. Czytając tabelę wierszami, możemy w łatwy sposób porównać jakość predykcji danej funkcji selekcji do wszystkich pozostałych funkcji. Jeżeli wartość metryki dla funkcji w danym wierszu jest wyższa i wynik jest istotny statystycznie, komórka została zaznaczona kolorem zielonym. Jeżeli wartość metryki jest niższa i wynik jest istotny statystycznie, komórkę zaznaczono kolorem czerwonym. Jeżeli nie można odrzucić hipotezy zerowej dla testu, kolor komórki jest szary.

Na podstawie powyższych wyników można wyciągnąć wniosek, że funkcja selekcji oparta na klasyfikatorze charakteryzuje się zadowalającą jakością predykcji na każdym ze zbiorów danych i może być stosowana jako domyślny wybór dla algorytmu dwukierunkowego. W 15 spośród 20 wykonanych porównań uzyskała wynik istotnie wyższy od pozostałych metod selekcji. Była ponadto jedyną funkcją, która na każdym zbiorze danych odznaczała się wyższą jakością predykcji od metod znanych z literatury. Funkcja ta wyróżnia się szczególnie na zbiorze GENIA, na którym różnice pomiędzy ujęciem iteracyjnym a pozostałymi metodami były najmniejsze. Uwzględniając pozostałe zbiory danych, selekcja oparta na pewności również uzyskała bardzo dobre wyniki, na zbiorze GENIA ustępuje jednak ona selekcji opartej na klasyfikatorze. Niewielkie różnice pomiędzy wynikami metod iteracyjnych a pozostałymi rozwiązaniami na zbiorze GENIA można tłumaczyć niskim poziomem złożoności struktur jednostek nazewniczych. W przypadku zbiorów NNE lub PolEval, gdzie zarówno liczba klas jednostek jak i głębokość zagnieżdżonych struktur jest znacząco wyższa, bardziej widoczna jest również przewaga ujęcia iteracyjnego. W przypadku tych dwóch zbiorów co najmniej cztery funkcje selekcji uzyskały wyniki wyższe od rozwiązań z literatury. Na podstawie tych wyników oraz przeprowadzonych *ablation studies* można stwierdzić, że metody iteracyjne są szczególnie efektywne w przypadku problemów o skomplikowanych i głębokich strukturach jednostek nazewniczych. Przypuszczalnie, przewaga ta może wynikać z bezpośredniego przekazywania do modelu zakodowanej informacji o jednostkach wykrytych w poprzednich iteracjach (warstwach). Inne metody wykrywania zagnieżdżonych struktur na ogół nie modelują relacji pomiędzy jednostkami nadrzędnymi a podrzędnymi lub modelują ją w sposób niebezpośredni - oczekując, że model sam nauczy się uwzględniać tego typu informacje w procesie treningu.

Zaletą iteracyjnego algorytmu dwukierunkowego, na którą warto zwrócić uwagę, jest możliwość doboru odpowiedniej funkcji selekcji do konkretnego zastosowania lub zaimplementowania nowej funkcji o pożądanych cechach. Daje to dodatkową elastyczność pozwalającą na sterowanie zachowaniem modelu. Jest to cecha, którą nie może się pochwalić większość znanych metod wykrywania hierarchicznych struktur jednostek nazewniczych. Zaproponowane w niniejszej pracy funkcje selekcji różnią się między innymi precyzją i czułością wyników. Dla metod takich jak selekcja oparta na klasyfikatorze czy pewności wartości tych metryk są zbalansowane. Z kolei część wspólna lub suma modeli maksymalizują jedną z wartości. W przypadku niektórych praktycznych zastosowań użyteczne może okazać się wybranie funkcji o tego typu charakterystyce. Jeżeli koszt popełnienia błędu jest wysoki, może zależeć nam na maksymalizacji precyzji. Natomiast jeżeli chcemy wykryć jak najwięcej jednostek, nawet kosztem większej liczby błędów, skupiamy się na jak najwyższej czułości modelu. Oczywiście możliwe jest też

definiowanie bardziej skomplikowanych funkcji selekcji, przykładowo stosujących różne kryteria decyzyjne dla różnych klas jednostek, co pozwala na jeszcze dokładniejsze dostosowanie algorytmu do zadania.

Rozdział 8

Podsumowanie

Niniejsza praca skupiała się na problemie wykrywania hierarchicznych struktur jednostek nazewniczych, będącego bardziej wymagającym wariantem powszechnie znanego w dziedzinie przetwarzania języka naturalnego zadania identyfikacji jednostek nazewniczych (ang. *named entity recognition, NER*). Celem podjętych prac badawczych było wykazanie, że możliwa jest predykcja tego typu struktur hierarchicznych w sposób iteracyjny, rozpoczynając od pustego zbioru i rozszerzając go z każdą iteracją do momentu otrzymania kompletnego zbioru jednostek. Iteracyjne ujęcie tego problemu zaproponowane w niniejszej rozprawie pozwala na opracowywanie nowego rodzaju metod iteracyjnych, które do tej pory nie były używane w zadaniach predykcji hierarchicznych struktur jednostek.

Cel ten został osiągnięty poprzez zaproponowanie konkretnej architektury neuronowej będącej implementacją ujęcia iteracyjnego oraz przeprowadzenie eksperymentalnej weryfikacji jej działania na czterech publicznie dostępnych zbiorach danych stosowanych dla tego problemu. Sieć neuronowa zbudowana została w oparciu o współczesne wektorowe reprezentacje tekstu, warstwy rekurencyjne typu LSTM (long short-term memory) oraz warstwę predykcyjną CRF (conditional random fields). Funkcjonowanie sieci w sposób iteracyjny zapewniono dzięki dodatkowemu wektorowi stanu, przekazywanemu jako wejście podczas każdej z iteracji. W wektorze stanu zakodowana została informacja o jednostkach nazewniczych wykrytych przez model w poprzedzających iteracjach, na podstawie której sieć była w stanie nauczyć się identyfikacji różnych typów jednostek w zależności od aktualnego stanu modelu.

Poza neuronowym modelem iteracyjnym, w ramach pracy przedstawiono również koncepcję bardziej zaawansowanych systemów predykcyjnych do wykrywania hierarchicznych struktur jednostek. Zaproponowany został dwukierunkowy algorytm iteracyjny, którego głównym założeniem było łączenie wyników modeli iteracyjnych wykrywających struktury w przeciwnych kierunkach: *inside-out* oraz *outside-in*. Generowanie finalnego zbioru wyników odbywało się za pomocą specjalnej funkcji selekcji. W pracy opisanych zostało sześć funkcji selekcji, które przetestowano na wspomnianych powyżej zbiorach danych. Ponadto skupiono się na kwestii propagacji stanu w systemach dwukierunkowych. Omówione zostały dwa warianty algorytmu dwukierunkowego. W wariancie podstawowym każdy z modeli iteracyjnych działał niezależnie, przekazując na wejściu wektor stanu pochodzący z tego samego modelu. W wariancie z propagacją krzyżową modele iteracyjne współdzieliły stan, wektor stanu był przekazy-

wany nie tylko pomiędzy iteracjami tego samego modelu, ale również pomiędzy dwoma modelami.

W rozdziale 7 zawarto szczegółowy opis przeprowadzonych eksperymentów obejmujących zarówno samodzielne neuronowe modele iteracyjne jak i dwukierunkowy algorytm iteracyjny. Przeprowadzono również eksperymenty typu *ablation study* mające na celu dokładniejsze zbadanie wpływu poszczególnych cech zaproponowanych w pracy metod na jakość predykcji. Wyniki osiągnięte przez metody iteracyjne zostały porównane z kilkudziesięcioma innymi rozwiązaniami problemu wykrywania hierarchicznych struktur jednostek nazewniczych opisanymi w literaturze. Przeprowadzone badania wskazują, że ujęcie iteracyjne jest w stanie konkurować z najlepszymi spośród stosowanych do tej pory metod, osiągając poprawność predykcji na poziomie porównywalnym lub wyższym od nich.

Podsumowując, rozprawa stanowi istotny wkład w rozwój metod wykrywania hierarchicznych struktur jednostek nazewniczych. Zaproponowane w pracy ujęcie iteracyjne przedstawia alternatywne w stosunku do znanych wcześniej metod spojrzenie na ten problem. Praktyczne eksperymenty wykazały, że modele iteracyjne charakteryzują się wysoką jakością predykcji pozwalającą na ich praktyczne zastosowanie w rzeczywistych zadaniach identyfikacji jednostek. Można zatem stwierdzić, że wszystkie założone cele rozprawy wymienione w rozdziale 1 zostały zrealizowane.

8.1. Główne osiągnięcia pracy

Zasadniczym tematem pracy były iteracyjne metody wykrywania hierarchicznych struktur jednostek nazewniczych. Część rezultatów osiągniętych w ramach przeprowadzonych badań wykracza jednak poza same metody iteracyjne i dotyczy szerszego spojrzenia na rozpatrywany problem badawczy. Możemy wyróżnić następujące elementy będące oryginalnym wkładem autora w badania nad identyfikacją jednostek nazewniczych:

• Przedstawienie nowego iteracyjnego ujęcia problemu wykrywania hierarchicznych struktur jednostek nazewniczych. Zaproponowanie metod pozwalających na zastosowanie tego ujęcia w praktyce: neuronowego modelu iteracyjnego oraz dwukierunkowego algorytmu iteracyjnego. Wprowadzenie wariantów algorytmu dwukierunkowego różniących się funkcją selekcji oraz metodą propagacji stanu pomiędzy modelami.

• Przeprowadzenie obszernego przeglądu metod wykrywania hierarchicznych struktur jednostek nazewniczych. Zgodnie z wiedzą autora, do tej pory nie powstała żadna publikacja przeglądowa skupiająca się na tym zagadnieniu. Artykuły opisujące nowe metody zawierają elementy przeglądu istniejących ujęć, najczęściej są one jednak wybiórcze i ograniczone. W ramach niniejszej pracy udało się usystematyzować wiedzę na temat znanych rozwiązań dla tego problemu. Wprowadzona została także taksonomia proponująca podział metod na trzy główne grupy: metody statystyczne, metody oparte na hipergrafach oraz metody oparte na sieciach neuronowych.

• Przeprowadzenie analizy złożoności obliczeniowej metod wykrywania hierarchicznych struktur jednostek nazewniczych. Wyróżnienie dwóch etapów przetwarzania danych

w tego typu rozwiązaniach: etapu generowania reprezentacji oraz etapu predykcji. Złożoność obliczeniowa jest aspektem często pomijanym przez autorów publikacji. W ramach pracy udało się określić złożoność rozwiązań na podstawie opisów rozwiązań zawartych w artykułach, również w przypadkach, w których autorzy nie podali jej bezpośrednio.

• Eksperymentalne porównanie metod iteracyjnych z innymi rozwiązaniami z literatury. Doświadczenia wykonane zostały na czterech zbiorach danych o różnej charakterystyce, różniących się między innymi językiem, źródłem pochodzenia tekstów czy poziomem złożoności zagnieżdżonych struktur. Na podstawie osiągniętych wyników udało się wykazać, że metody iteracyjne są konkurencyjne w stosunku do stosowanych wcześniej rozwiązań. Pokazano również, że zaproponowana architektura iteracyjna jest w niewielkim stopniu wrażliwa na zmieniającą się charakterystykę danych. Ten sam model iteracyjny wykorzystujący taki sam zestaw hiperparametrów uzyskał zadowalające wyniki na każdym z rozpatrywanych zbiorów.

• Zbadanie wariantów dwukierunkowego algorytmu iteracyjnego różniących się funkcją selekcji. Najwyższą jakością predykcji spośród metod selekcji charakteryzowała się metoda oparta na klasyfikatorze, uzyskując wysokie wyniki na każdym ze zbiorów, na których wykonano ewaluację. Inne metody selekcji wykazywały równie wysoką skuteczność na wybranych zbiorach, ale okazały się nie tak stabilne jak selekcja oparta na klasyfikatorze.

• Porównanie wariantów dwukierunkowego algorytmu iteracyjnego różniących się metodą propagacji stanu. Wyniki doświadczeń wskazują, że algorytm wykorzystujący krzyżową propagację stanu nie uzyskuje istotnie wyższych wyników od algorytmu z propagacją prostą. W przypadku propagacji krzyżowej warto jest stosować dodatkową operację dropout na wektorze stanu, w ten sposób można uzyskać jakość predykcji porównywalną z wariantem prostym. W algorytmie z propagacją prostą zastosowanie dropoutu stanu nie poprawia jakości predykcji.

Część badań wykonanych przez autora w ramach zagadnienia identyfikacji jednostek nazewniczych, których omówienie znalazło się w niniejszej rozprawie, została wcześniej opublikowana w postaci artykułu w recenzowanym czasopiśmie naukowym [20] oraz recenzowanych materiałach konferencyjnych [19].

8.2. Charakterystyka metod iteracyjnych

Na podstawie przeprowadzonych badań możliwe jest scharakteryzowanie iteracyjnych metod identyfikacji struktur jednostek nazewniczych poprzez wskazanie cech, które wyróżniają je na tle innych ujęć tego problemu. Warto zwrócić uwagę na silne strony zaprezentowanych w niniejszej rozprawie metod:

• Ujęcie iteracyjne oferuje intuicyjny, łatwy do zrozumienia mechanizm identyfikacji jednostek nazewniczych. Odróżnia to przedstawione w pracy metody od większości współczesnych rozwiązań opartych na sieciach neuronowych, które podążają w kierunku coraz bardziej złożonych systemów. Zwiększa się zarówno liczba parametrów

(wag) sieci jak i hiperparametrów, które sterują procesem trenowania oraz predykcji. W tego typu rozwiązaniach trudniejsze staje się identyfikowanie głównych elementów wpływających na jakość predykcji, trudniej jest także wyjaśnić przyczynę potencjalnych błędnych odpowiedzi systemu. W modelach iteracyjnych natomiast predykcja przebiega w sposób krokowy, jesteśmy w stanie prześledzić proces generowania ostatecznego wyniku warstwa po warstwie. W zakresie wyjaśnialności modelu otrzymujemy zatem dodatkowe możliwości, których nie oferuje większość metod wykorzystujących sieci neuronowe.

• Ujęcie iteracyjne jest naturalnym rozszerzeniem klasycznej metody wykrywania jednostek nazewniczych za pomocą tagowania sekwencji. Oznacza to, że wiele spośród znanych modeli stosowanych w problemach płaskiej identyfikacji jednostek można dostosować do działania w sposób iteracyjny.

• Jak pokazały wyniki przeprowadzonych eksperymentów, metody iteracyjne charakteryzują się wysoką poprawnością predykcji, porównywalną lub wyższą od wcześniej stosowanych rozwiązań. Wysoką efektywność modeli iteracyjnych można było zaobserwować zarówno na zbiorach danych z prostymi strukturami jednostek nazewniczych jak i na zbiorach ze skomplikowanymi strukturami o dużej głębokości zagnieżdżeń. Istotą ujęcia iteracyjnego jest jednak propagacja informacji na temat wykonanych predykcji pomiędzy poszczególnymi iteracjami modelu. Z założenia modele tego typu powinny zatem sprawdzać się szczególnie dobrze w przypadkach, w których mamy do czynienia z głębokimi strukturami jednostek nazewniczych należących do wielu klas.

• Zastosowanie dwukierunkowego algorytmu iteracyjnego jest łatwym sposobem na poprawienie jakości predykcji. Ponadto algorytm dwukierunkowy wprowadza dodatkowy element w postaci funkcji selekcji, która umożliwia sterowanie działaniem algorytmu oraz ma wpływ na charakterystykę wyników generowanych przez system. Dobór odpowiedniej funkcji selekcji do problemu pozwala między innymi na balansowanie metrykami precyzji i czułości. Dzięki zdefiniowaniu własnej funkcji selekcji można dostosować algorytm do konkretnego zadania. Wymiana funkcji selekcji na inną jest prosta, nie wymaga modyfikacji samych modeli iteracyjnych ani nawet ich ponownego trenowania.

• W algorytmie dwukierunkowym z propagacją prostą modele iteracyjne działają w sposób niezależny. Daje to możliwość przyspieszenia generowania wyników poprzez uruchamianie ich w sposób współbieżny. Możliwe jest wykorzystanie dwóch jednostek obliczeniowych (np. kart graficznych) lub nawet dwóch różnych maszyn do obsługi każdego z modeli, a następnie połączenie ich dopiero na etapie selekcji.

• Opisany w pracy dwukierunkowy algorytm iteracyjny jest systemem predykcyjnym składającym się z trzech elementów: modelu *inside-out*, modelu *outside-in* oraz funkcji selekcji. Nie istnieją silne zależności pomiędzy tymi modułami, w każdej chwili możliwa jest zatem wymiana jednego z nich bez konieczności wprowadzania zmian w pozostałych. Ułatwia to testowanie różnych wariantów systemu oraz jego utrzymanie w środowisku produkcyjnym.

Poza wspomnianymi powyżej zaletami metod iteracyjnych, istnieją również pewne mankamenty tego podejścia. W ramach pracy udało się zidentyfikować następujące

słabsze strony proponowanego rozwiązania:

• Czas potrzebny do wygenerowania predykcji w modelach iteracyjnych nie wyróżnia się na tle innych metod rozwiązujących ten problem. Istnieją grupy metod charakteryzujące się mniejszą efektywnością pod tym względem, na przykład modele oparte na hipergrafach, modele enumerujące lub modele przejść pomiędzy stanami. Niektóre ze współczesnych metod oferują jednak szybsze czasy wykonania, przykładowo Shibuya and Hovy [119] zaproponowali model wykrywania zagnieżdżonych struktur jednostek nazewniczych, który wymaga pojedynczego przejścia przez sieć neuronową dla każdego rekordu. W modelu iteracyjnym liczba takich przejść jest równa liczbie iteracji, która odpowiada głębokości struktury jednostek nazewniczych w danym rekordzie. W kontekście wszystkich metod są to wciąż przyzwoite wyniki, natomiast odbiegają one od najszybszych znanych obecnie modeli.

• Również czas potrzebny do wytrenowania modelu może być dłuższy niż w przypadku innych metod opartych na sieciach neuronowych. Zaproponowana w pracy metoda treningu polega na rozbiciu pojedynczych próbek treningowych na grupy próbek odpowiadające poszczególnym iteracjom, lub warstwom jednostek nazewniczych. Prowadzi to do kilkukrotnego powiększenia rozmiaru zbioru treningowego, co wydłuża czas pojedynczej epoki uczenia sieci. W praktyce czas wymagany do wytrenowania modelu iteracyjnego na dużym zbiorze danych takim jak NNE sięgał kilkudziesięciu godzin na sprzęcie, na którym przeprowadzano eksperymenty.

• Stosowana w ujęciu iteracyjnym metoda przekazywania stanu pomiędzy poszczególnymi etapami predykcji niesie za sobą ryzyko propagacji błędów. Błąd popełniony na wstępnym etapie predykcji zostanie zapisany w wektorze stanu i będzie przekazywany do modelu w kolejnych iteracjach, co może prowadzić do zidentyfikowania kolejnych błędnych jednostek. Będzie to działać na niekorzyść metod iteracyjnych na przykład w przypadku małych zbiorów danych, na których model w procesie trenowania nie osiągnie dostatecznie wysokiej jakości predykcji.

8.3. Kierunki rozwoju

Podstawowe założenia zaproponowanego w niniejszej pracy iteracyjnego ujęcia problemu wykrywania hierarchicznych struktur jednostek nazewniczych mają ogólną postać, która jest możliwa do zaadaptowania w różnego typu architekturach neuronowych. Iteracyjny model neuronowy opisany w rozdziale 5 jest konkretnym przykładem zastosowania tego ujęcia w praktyce. Naturalnym kierunkiem dalszych badań w tym zakresie wydaje się zatem dostosowanie innego typu modeli do działania w sposób iteracyjny. W ostatnich latach możemy zaobserwować szczególnie dynamiczny rozwój neuronowych metod przetwarzania języka naturalnego. Wiąże się to przede wszystkim z upowszechnieniem architektury Transformer, czego efektem są liczne publikacje związane z jej zastosowaniami w problemach z zakresu NLP oraz artykuły proponujące usprawnienia i modyfikacje oryginalnego modelu Transformera. Możemy także zauważyć zainteresowanie innymi rodzajami sieci neuronowych używanymi w przetwarzaniu

języka. Wciąż dużą popularnością cieszą się rozwiązania oparte na sieciach rekurencyjnych, swoje zastosowania znalazły także grafowe sieci neuronowe [150]. W kontekście tych badań, interesującym eksperymentem byłoby przetestowanie ujęcia iteracyjnego na architekturze opartej na mechanizmie *self-attention* wzorując się na sieciach Transformer. Wiedza na temat tego rodzaju modeli jest coraz większa. Od czasu opublikowania oryginalnego modelu BERT [24] dokonał się istotny postęp w zrozumieniu architektury Transformer, udało się wyeliminować pewne mankamenty wczesnych modeli, wskazano także bardziej efektywne sposoby na ich pretrenowanie i strojenie pod kątem wykorzystania w praktycznych zastosowaniach.

Równolegle z rozwojem architektur sieci neuronowych dokonuje się także postęp w zakresie ich skalowania. Można to zauważyć w szczególności w przypadku neuronowych modeli języka. Organizacje dysponujące dużymi centrami danych wykorzystują moc obliczeniową od uczenia modeli o coraz większej liczbie parametrów oraz na większych korpusach tekstowych. Wytrenowane w ten sposób modele języka mogą być następnie użyte jako kontekstowe reprezentacje tekstu, wpływając na jakość modeli rozwiązujących konkretne problemy w oparciu o dane tekstowe. Ma to również bezpośrednie przełożenie na metody identyfikacji hierarchicznych struktur jednostek nazewniczych. Ze względu na fakt, iż badania prowadzone w ramach niniejszej rozprawy były rozciągnięte w czasie, samo zastosowanie nowszych reprezentacji wektorowych pochodzących z większych modeli języka z dużym prawdopodobieństwem mogłoby w chwili obecnej poprawić wyniki osiągane przez neuronowy model iteracyjny. Kwestia stosowania najbardziej efektywnych dostępnych reprezentacji tekstu jest zatem kolejnym obszarem, w którym istnieje pole do usprawnienia działania modelu.

Poza kwestiami związanymi z rozwojem metod przetwarzania tekstu i wykorzystywaniem nowych możliwości w modelach iteracyjnych, równie ciekawym zagadnieniem jest dokładniejsza eksploracja procesu predykcji w metodach iteracyjnych. Jak zostało wspomniane wcześniej, jedną z zalet ujęcia iteracyjnego jest możliwość badania zachowania modelu poprzez śledzenie identyfikowanych jednostek warstwa po warstwie. Zbudowanie automatycznych narzędzi, które byłyby w stanie na podstawie tego procesu wyliczać statystyki oraz wykrywać powtarzające się wzorce pozwoliłoby na lepsze zrozumienie działania modeli iteracyjnych. W tego typu modelach duże znaczenie mają relacje hierarchiczne pomiędzy jednostkami, informacje o jednostkach nadrzędnych są bowiem przekazywane w postaci stanu modelu i stanowią podstawę do identyfikacji nowych jednostek w kolejnych iteracjach. W przypadku konkretnego zadania wykrywania jednostek interesujące wydaje się pytanie jakie pary klas encji tworzące relacje nadrzędna-podrzędna są dla modelu najbardziej problematyczne, czyli powodują powstanie największej liczby błędów. Pozwoliłoby to na zidentyfikowanie najczęściej występujących przypadków propagacji błędów pomiędzy iteracjami i ułatwiłoby odnalezienie ich źródła.

Na koniec warto zwrócić uwagę na fakt, że w ramach zagadnienia wykrywania jednostek nazewniczych wciąż mamy do czynienia z dosyć prostymi, z ludzkiego punktu widzenia, problemami, często ograniczonymi do określonej dziedziny i polegającymi na identyfikacji niewielkiej liczby klas. Tymczasem wykorzystywane w rzeczywistości taksonomie do kategoryzacji pojęć, na przykład struktura kategorii zdefiniowana w Wikipedii, obejmują setki tysięcy pozycji. Wraz z rozwojem metod przetwarzania języka naturalnego możemy się zatem spodziewać bardziej wymagających zadań, obej-

mujących teksty z wielu domen i tysiące kategorii jednostek nazewniczych, których celem będzie budowanie modeli ogólnego przeznaczenia umożliwiających generowanie szczegółowych struktur nazw występujących w dowolnego typu dokumentach. Takie problemy stanowić będą znacznie większe wyzwanie dla metod identyfikacji jednostek. Przeprowadzone w pracy badania wskazują, że metody iteracyjne dobrze radzą sobie z najbardziej złożonymi spośród obecnie dostępnych zbiorów danych. Przypuszczalnie mogą się skalować również do większych zbiorów, z głębszymi i bardziej szczegółowymi strukturami jednostek nazewniczych.

Bibliografia

[1] Alan Akbik, Duncan Blythe, and Roland Vollgraf. Contextual string embeddings for sequence labeling. In *COLING 2018: 27th International Conference on Computational Linguistics*, pages 1638–1649, 2018.

[2] Alan Akbik, Tanja Bergmann, and Roland Vollgraf. Pooled contextualized embeddings for named entity recognition. In *NAACL-HLT 2019: Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 724–728. Association for Computational Linguistics, 2019.

[3] Beatrice Alex, Barry Haddow, and Claire Grover. Recognising nested named entities in biomedical text. In *Biological, translational, and clinical language processing*, pages 65–72. Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, 2007.

[4] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[5] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[6] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *The journal of machine learning research*, 3:1137–1155, 2003.

[7] Darina Benikova, Chris Biemann, and Marc Reznicek. NoSta-d named entity annotation for German: Guidelines and dataset. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 2524–2531, Reykjavik, Iceland, May 2014. European Language Resources Association (ELRA).

[8] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[9] Łukasz Borchmann, Andrzej Gretkowski, and Filip Graliński. Approaching nested named entity recognition with parallel LSTM-CRFs. In *Proceedings of AI and NLP Workshop Day 2018*, 2018.

[10] Joseph K Bradley and Carlos Guestrin. Learning tree conditional random fields. In *ICML*, 2010.

[11] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[12] Yanping Chen, Yuefei Wu, Yongbin Qin, Ying Hu, Zeyu Wang, Ruizhang Huang, Xinyu Cheng, and Ping Chen. Recognizing nested named entity based on the neural network boundary assembling model. *IEEE Intelligent Systems*, pages 74–81, 2020.

[13] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *EMNLP*, 2014.

[14] Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J Colwell, and Adrian Weller. Rethinking attention with performers. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=Ua6zuk0WRH`.

[15] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. Electra: Pre-training text encoders as discriminators rather than generators. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=r1xMH1BtvB`.

[16] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167, 2008.

[17] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(ARTICLE):2493–2537, 2011.

[18] Yiming Cui, Wanxiang Che, Ting Liu, Bing Qin, Ziqing Yang, Shijin Wang, and Guoping Hu. Pre-training with whole word masking for chinese BERT. *arXiv preprint arXiv:1906.08101*, 2019.

[19] Sławomir Dadas. Combining neural and knowledge-based approaches to named entity recognition in Polish. In *International Conference on Artificial Intelligence and Soft Computing*, pages 39–50. Springer, 2019.

[20] Sławomir Dadas and Jarosław Protasiewicz. A bidirectional iterative algorithm for nested named entity recognition. *IEEE Access*, 8:135091–135102, 2020. doi: 10.1109/ACCESS.2020.3011598.

[21] Xiang Dai. Recognizing complex entity mentions: A review and future directions. In *Proceedings of ACL 2018, Student Research Workshop*, pages 37–44,

Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-3006. URL https://www.aclweb.org/anthology/P18-3006.

[22] Xiang Dai, Sarvnaz Karimi, Ben Hachey, and Cecile Paris. An effective transition-based model for discontinuous NER. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5860–5870, 2020.

[23] Leon Derczynski, Eric Nichols, Marieke van Erp, and Nut Limsopatham. Results of the WNUT2017 shared task on novel and emerging entity recognition. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pages 140–147, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-4418. URL https://www.aclweb.org/anthology/W17-4418.

[24] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://www.aclweb.org/anthology/N19-1423.

[25] George R. Doddington, Alexis Mitchell, Mark A. Przybocki, Lance A. Ramshaw, Stephanie M. Strassel, and Ralph M. Weischedel. The automatic content extraction (ACE) program tasks, data, and evaluation. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC-2004)*. Language Resources and Evaluation, European Language Resources Association (ELRA), 2004.

[26] Cicero Dos Santos and Maira Gatti. Deep convolutional neural networks for sentiment analysis of short texts. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 69–78, 2014.

[27] Mark Dredze, Paul McNamee, Delip Rao, Adam Gerber, and Tim Finin. Entity disambiguation for knowledge base population. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 277–285, 2010.

[28] Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S Weld, and Alexander Yates. Unsupervised named-entity extraction from the web: An experimental study. *Artificial intelligence*, 165(1):91–134, 2005.

[29] Jenny Rose Finkel and Christopher D. Manning. Nested named entity recognition. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 141–150. Empirical Methods in Natural Language Processing, Association for Computational Linguistics, 2009.

[30] Jenny Rose Finkel, Alex Kleeman, and Christopher D. Manning. Efficient, feature-based, conditional random field parsing. *Proceedings of ACL-08: HLT*, pages 959–967, 2008.

[31] Joseph Fisher and Andreas Vlachos. Merge and label: A novel neural network architecture for nested NER. *arXiv preprint arXiv:1907.00464*, 2019.

[32] Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in neural information processing systems*, pages 1019–1027, 2016.

[33] Yoav Goldberg. Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies*, 10(1):1–309, 2017. doi: 10.2200/S00762ED1V01Y201703HLT037. URL `https://doi.org/10.2200/S00762ED1V01Y201703HLT037`.

[34] Filip Graliński, Krzysztof Jassem, Michał Marcińczuk, and Paweł Wawrzyniak. Named entity recognition in machine anonymization. *Recent Advances in Intelligent Information Systems*, pages 247–260, 2009.

[35] Edouard Grave, Piotr Bojanowski, Prakhar Gupta, Armand Joulin, and Tomas Mikolov. Learning word vectors for 157 languages. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 2018. European Language Resources Association (ELRA). URL `https://www.aclweb.org/anthology/L18-1550`.

[36] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[37] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural networks*, 18 (5-6):602–610, 2005.

[38] Ralph Grishman and Beth Sundheim. Message understanding conference-6: A brief history. In *Proceedings of the 16th Conference on Computational Linguistics - Volume 1*, COLING '96, page 466–471, USA, 1996. Association for Computational Linguistics. doi: 10.3115/992628.992709. URL `https://doi.org/10.3115/992628.992709`.

[39] Baohua Gu. Recognizing nested named entities in GENIA corpus. In *Proceedings of the HLT-NAACL BioNLP Workshop on Linking Natural Language and Biology*, pages 112–113. North American Chapter of the Association for Computational Linguistics, Association for Computational Linguistics, 2006.

[40] Jiafeng Guo, Gu Xu, Xueqi Cheng, and Hang Li. Named entity recognition in query. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 267–274, 2009.

[41] Zhou GuoDong and Su Jian. Exploring deep knowledge resources in biomedical name recognition. In *JNLPBA '04 Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications*, pages 96–99. JNLPBA '04 Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications, Association for Computational Linguistics, 2004.

[42] Neha Gupta and Rashmi Agrawal. Chapter 1 - application and techniques of opinion mining. In Siddhartha Bhattacharyya, Václav Snášel, Deepak Gupta, and Ashish Khanna, editors, *Hybrid Computational Intelligence*, Hybrid Computational Intelligence for Pattern Analysis and Understanding, pages 1 – 23. Academic Press, 2020. ISBN 978-0-12-818699-2. doi: https://doi.org/10.1016/B978-0-12-818699-2.00001-9. URL `http://www.sciencedirect.com/science/article/pii/B9780128186992000019`.

[43] Christian Hänig, Stefan Thomas, and Stefan Bordag. Modular classifier ensemble architecture for named entity recognition on low resource systems. In *GermEval 2014*, 2014.

[44] Zellig S Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

[45] Fadi Hassan, Josep Domingo-Ferrer, and Jordi Soria-Comas. Anonymization of unstructured data via named-entity recognition. In *International conference on modeling decisions for artificial intelligence*, pages 296–305. Springer, 2018.

[46] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[47] Geoffrey E. Hinton, Nitish Srivastava, A. Krizhevsky, Ilya Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *ArXiv*, abs/1207.0580, 2012.

[48] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, pages 1735–1780, 1997.

[49] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8): 2554–2558, 1982.

[50] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional LSTM-CRF models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.

[51] Matthew James Johnson, David Duvenaud, Alexander B Wiltschko, Sandeep R Datta, and Ryan P Adams. Composing graphical models with neural networks for structured representations and fast inference. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 2954–2962, 2016.

[52] Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. SpanBERT: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics*, 8:64–77, 2020. doi: 10.1162/tacl_a_00300. URL `https://www.aclweb.org/anthology/2020.tacl-1.5`.

[53] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 427–431, Valencia, Spain, April 2017. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/E17-2068`.

[54] Meizhi Ju, Makoto Miwa, and Sophia Ananiadou. A neural layered model for nested named entity recognition. In *NAACL HLT 2018: 16th Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1446–1459. North American Chapter of the Association for Computational Linguistics, 2018.

[55] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 655–665, Baltimore, Maryland, June 2014. Association for Computational Linguistics. doi: 10.3115/v1/P14-1062. URL `https://www.aclweb.org/anthology/P14-1062`.

[56] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, pages 5156–5165. PMLR, 2020.

[57] Arzoo Katiyar and Claire Cardie. Nested named entity recognition revisited. In *NAACL HLT 2018: 16th Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 861–871. North American Chapter of the Association for Computational Linguistics, 2018.

[58] Jin-Dong Kim, Tomoko Ohta, Yoshimasa Tsuruoka, Yuka Tateisi, and Nigel Collier. Introduction to the bio-entity recognition task at JNLPBA. *JNLPBA '04 Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications*, pages 70–75, 2004.

[59] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1181. URL `https://www.aclweb.org/anthology/D14-1181`.

[60] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rkgNKkHtvB.

[61] Bennett Kleinberg, Maximilian Mozes, Yaloe van der Toolen, et al. Netanos-named entity-based text anonymization for open science. 2017.

[62] Roman Klinger and Katrin Tomanek. *Classical probabilistic models and conditional random fields*. Citeseer, 2007.

[63] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[64] Taku Kudo. Subword regularization: Improving neural network translation models with multiple subword candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 66–75, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1007. URL https://www.aclweb.org/anthology/P18-1007.

[65] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: probabilistic models for segmenting and labeling sequence data. In *Proc. of the 18th Intl. Conf. on Machine Learning (ICML-2001)*, 2001.

[66] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 260–270, San Diego, California, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/N16-1030. URL https://www.aclweb.org/anthology/N16-1030.

[67] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1eA7AEtvS.

[68] Joseph Lark, Emmanuel Morin, and Sebastián Peña Saldarriaga. A comparative study of target-based and entity-based opinion extraction. In *International Conference on Computational Linguistics and Intelligent Text Processing*, pages 211–223. Springer, 2017.

[69] Thomas Lavergne and François Yvon. Learning the structure of variable-order CRFs: a finite-state perspective. In *Proceedings of the 2017 Conference on*

Empirical Methods in Natural Language Processing, pages 433–439, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1044. URL https://www.aclweb.org/anthology/D17-1044.

[70] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[71] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[72] Changki Lee, Yi-Gyu Hwang, Hyo-Jung Oh, Soojong Lim, Jeong Heo, Chung-Hee Lee, Hyeon-Jin Kim, Ji-Hyun Wang, and Myung-Gil Jang. Fine-grained named entity recognition using conditional random fields for question answering. In *Asia information retrieval symposium*, pages 581–587. Springer, 2006.

[73] Elena Leitner, Georg Rehm, and Julian Moreno-Schneider. Fine-grained named entity recognition in legal documents. In *International Conference on Semantic Systems*, pages 272–287. Springer, 2019.

[74] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.703. URL https://www.aclweb.org/anthology/2020.acl-main.703.

[75] Jing Li, Aixin Sun, Jianglei Han, and Chenliang Li. A survey on deep learning for named entity recognition. *arXiv preprint arXiv:1812.09449*, 2018.

[76] Hongyu Lin, Yaojie Lu, Xianpei Han, and Le Sun. Sequence-to-nuggets: Nested entity mention detection via anchor-region networks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5182–5192, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1511. URL https://www.aclweb.org/anthology/P19-1511.

[77] Xiao Ling and Daniel Weld. Fine-grained entity recognition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, 2012.

[78] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[79] H-A Loeliger. An introduction to factor graphs. *IEEE Signal Processing Magazine*, 21(1):28–41, 2004.

[80] Wei Lu and Dan Roth. Joint mention extraction and classification with mention hypergraphs. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 857–867. Empirical Methods in Natural Language Processing, Association for Computational Linguistics (ACL), 2015.

[81] Xuezhe Ma and Eduard H. Hovy. End-to-end sequence labeling via bi-directional LSTM-cnns-CRF. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1064–1074, 2016.

[82] Khai Mai, Thai-Hoang Pham, Minh Trung Nguyen, Tuan Duc Nguyen, Danushka Bollegala, Ryohei Sasano, and Satoshi Sekine. An empirical study on fine-grained named entity recognition. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 711–722, Santa Fe, New Mexico, USA, August 2018. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/C18-1060`.

[83] Christopher D Manning, Prabhakar Raghavan, and H. Schutze. Introduction to information retrieval, 2008.

[84] Michał Marcińczuk, Jan Kocoń, and Maciej Janicki. Liner2–a customizable framework for proper names recognition for Polish. In *Intelligent tools for building a scientific information platform*, pages 231–253. Springer, 2013.

[85] Michał Marcińczuk, Jan Kocoń, and Michał Gawor. Recognition of named entities for Polish-comparison of deep learning and conditional random fields approaches. In Maciej Ogrodniczuk and Łukasz Kobyliński, editors, *Proceedings of the PolEval 2018 Workshop*, pages 77–92. Institute of Computer Science, Polish Academy of Science, 2018.

[86] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993. URL `https://www.aclweb.org/anthology/J93-2004`.

[87] Zita Marinho, Afonso Mendes, Sebastião Miranda, and David Nogueira. Hierarchical nested named entity recognition. In *Proceedings of the 2nd Clinical Natural Language Processing Workshop*, pages 28–34, Minneapolis, Minnesota, USA, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-1904.

[88] Louis Martin, Benjamin Muller, Pedro Javier Ortiz Suárez, Yoann Dupont, Laurent Romary, Éric de la Clergerie, Djamé Seddah, and Benoît Sagot. CamemBERT: a tasty French language model. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7203–7219, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.645. URL `https://www.aclweb.org/anthology/2020.acl-main.645`.

[89] Ryan McDonald, George Brokos, and Ion Androutsopoulos. Deep relevance ranking using enhanced document-query interactions. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1849–1860, 2018.

[90] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[91] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26, pages 3111–3119. Curran Associates, Inc., 2013. URL `https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf`.

[92] Diego Mollá, Menno van Zaanen, and Daniel Smith. Named entity recognition for question answering. In *Proceedings of the Australasian Language Technology Workshop 2006*, pages 51–58, Sydney, Australia, November 2006. URL `https://www.aclweb.org/anthology/U06-1009`.

[93] Aldrian Obaja Muis and Wei Lu. Learning to recognize discontiguous entities. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 75–84, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1008. URL `https://www.aclweb.org/anthology/D16-1008`.

[94] Aldrian Obaja Muis and Wei Lu. Labeling gaps between words: Recognizing overlapping mentions with mention separators. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2608–2618. Empirical Methods in Natural Language Processing, Association for Computational Linguistics, 2017.

[95] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26, 2007.

[96] Tomoko Ohta, Yuka Tateisi, and Jin-Dong Kim. The GENIA corpus: An annotated research abstract corpus in molecular biology domain. In *Proceedings of the Second International Conference on Human Language Technology Research*, HLT '02, pages 82–86, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc. URL `http://dl.acm.org/citation.cfm?id=1289189.1289260`.

[97] Tomoko Ohta, Yuka Tateisi, and Jin-Dong Kim. The GENIA corpus: an annotated research abstract corpus in molecular biology domain. In *the second international conference*, pages 82–82. the second international conference, 2002.

[98] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543. Association for Computational Linguistics, 2014.

[99] Matthew Peters, Waleed Ammar, Chandra Bhagavatula, and Russell Power. Semi-supervised sequence tagging with bidirectional language models. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1756–1765, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1161. URL https://www.aclweb.org/anthology/P17-1161.

[100] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of NAACL-HLT*, pages 2227–2237, 2018.

[101] Desislava Petkova and W Bruce Croft. Proximity-based document representation for named entity retrieval. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 731–740, 2007.

[102] Adam Przepiórkowski, Mirosław Banko, Rafał L Górski, and Barbara Lewandowska-Tomaszczyk. *National Corpus of Polish*. Polish Scientific Publishers PWN, Warsaw, 2012.

[103] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[104] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *OpenAI*, 2018.

[105] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI*, 2019.

[106] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

[107] Lev Ratinov and Dan Roth. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL-2009)*, pages 147–155, Boulder, Colorado, June 2009. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W09-1119.

[108] Lisa F. Rau and Paul S. Jacobs. Creating segmented databases from free text for text retrieval. In *Proceedings of the 14th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '91, page 337–346, New York, NY, USA, 1991. Association for Computing Machinery. ISBN 0897914481. doi: 10.1145/122860.122894. URL https://doi.org/10.1145/122860.122894.

[109] Nils Reimers and Iryna Gurevych. Optimal hyperparameters for deep LSTM-networks for sequence labeling tasks. *arXiv preprint arXiv:1707.06799*, 2017.

[110] Nils Reimers, Judith Eckle-Kohler, Carsten Schnober, Jungi Kim, and Iryna Gurevych. Germeval-2014: Nested named entity recognition with neural networks. In *GermEval 2014*, 2014. ISBN 978-3-934105-47-8. URL `http://nbn-resolving.de/urn:nbn:de:gbv:hil2-opus-3023`.

[111] S Reshmi and Kannan Balakrishnan. Enhancing inquisitiveness of chatbots through ner integration. In *2018 International Conference on Data Science and Engineering (ICDSE)*, pages 1–5. IEEE, 2018.

[112] Nicky Ringland, Xiang Dai, Sarvnaz Karimi, Ben Hachey, Cecile Paris, and James R. Curran. NNE: A dataset for nested named entity recognition in english newswire. In *ACL 2019 : The 57th Annual Meeting of the Association for Computational Linguistics*, pages 5176–5181. Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, 2019.

[113] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[114] Alexander M Rush. Torch-struct: Deep structured prediction library. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 335–342, 2020.

[115] Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P. Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *ICLR (Poster)*, 2017.

[116] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

[117] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1162. URL `https://www.aclweb.org/anthology/P16-1162`.

[118] Dan Shen, Jie Zhang, Guodong Zhou, Jian Su, and Chew-Lim Tan. Effective adaptation of hidden markov model-based named entity recognizer for biomedical domain. In *Proceedings of the ACL 2003 Workshop on Natural Language Processing in Biomedicine*, pages 49–56. Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, 2003.

[119] Takashi Shibuya and Eduard Hovy. Nested named entity recognition via second-best sequence learning and decoding. *arXiv preprint arXiv:1909.02250*, 2019.

[120] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1409.1556`.

[121] Mohammad Golam Sohrab and Makoto Miwa. Deep exhaustive model for nested named entity recognition. In *EMNLP 2018: 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2843–2849. Empirical Methods in Natural Language Processing, 2018.

[122] Lin Sun, Fule Ji, Kai Zhang, and Chi Wang. Multilayer ToI detection approach for nested ner. *IEEE Access*, 7:186600–186608, 2019.

[123] Charles Sutton, Andrew McCallum, and Khashayar Rohanimanesh. Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data. *Journal of Machine Learning Research*, 8(3), 2007.

[124] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[125] Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147, 2003. URL `https://www.aclweb.org/anthology/W03-0419`.

[126] Andrew Trask, Phil Michalak, and John Liu. Sense2Vec - a fast and accurate method for word sense disambiguation in neural word embeddings. *arXiv preprint arXiv:1511.06388*, 2015.

[127] Tzong-han Tsai, Wen-Chi Chou, Shih-Hung Wu, Ting-Yi Sung, Jieh Hsiang, and Wen-Lian Hsu. Integrating linguistic knowledge into a conditional random fieldframework to identify biomedical named entities. *Expert Systems With Applications*, pages 117–128, 2006.

[128] Aaron van den Oord, Nal Kalchbrenner, Lasse Espeholt, koray kavukcuoglu, Oriol Vinyals, and Alex Graves. Conditional image generation with pixelcnn decoders. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4790–4798. Curran Associates, Inc., 2016. URL `http://papers.nips.cc/paper/6527-conditional-image-generation-with-pixelcnn-decoders.pdf`.

[129] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alexander Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. In *Arxiv*, 2016. URL `https://arxiv.org/abs/1609.03499`.

[130] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017. URL `http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf`.

[131] Andrew Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*, 13(2): 260–269, 1967.

[132] Bailin Wang and Wei Lu. Neural segmental hypergraphs for overlapping mention recognition. *Empirical Methods in Natural Language Processing*, pages 204–214, 2018.

[133] Bailin Wang, Wei Lu, Yu Wang, and Hongxia Jin. A neural transition-based model for nested mention recognition. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1011–1017, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1124. URL `https://www.aclweb.org/anthology/D18-1124`.

[134] Aleksander Wawer and Estera Małek. Results of the poleval 2018 shared task 2: Named entity recognition. In *Proceedings of the PolEval 2018 Workshop*, Warsaw, Poland, 2018. Institute of Computer Science, Polish Academy of Sciences. ISBN 978-83-63159-27-6. URL `http://poleval.pl/files/poleval2018.pdf`.

[135] Ralph Weischedel, Martha Palmer, Mitchell Marcus, Eduard Hovy, Sameer Pradhan, Lance Ramshaw, Nianwen Xue, Ann Taylor, Jeff Kaufman, Michelle Franchini, et al. Ontonotes release 5.0. *Linguistic Data Consortium, Philadelphia, PA*, 23, 2013.

[136] Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, pages 270–280, 1989.

[137] Congying Xia, Chenwei Zhang, Tao Yang, Yaliang Li, Nan Du, Xian Wu, Wei Fan, Fenglong Ma, and Philip Yu. Multi-grained named entity recognition. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1430–1440, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1138. URL `https://www.aclweb.org/anthology/P19-1138`.

[138] Hu Xu, Bing Liu, Lei Shu, and Philip S. Yu. Double embeddings and CNN-based sequence labeling for aspect extraction. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 592–598, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-2094. URL `https://www.aclweb.org/anthology/P18-2094`.

[139] Mingbin Xu, Hui Jiang, and Sedtawut Watcharawittayakul. A local detection approach for named entity recognition and mention detection. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1237–1247. Meeting of the Association for Computational Linguistics, 2017.

[140] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining

for language understanding. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, pages 5753–5763. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/dc6a7e655d7e5840e66733e9ee67cc69-Paper.pdf.

[141] Jie Zhang, Dan Shen, Guodong Zhou, Jian Su, and Chew-Lim Tan. Enhancing HMM-based biomedical named entity recognition by studying special phenomena. *Journal of Biomedical Informatics*, pages 411–422, 2004.

[142] Lei Zhang and Bing Liu. Aspect and entity extraction for opinion mining. In *Data mining and knowledge discovery for big data*, pages 1–40. Springer, 2014.

[143] Linhao Zhang and Houfeng Wang. Using bidirectional Transformer-CRF for spoken language understanding. In *CCF International Conference on Natural Language Processing and Chinese Computing*, pages 130–141. Springer, 2019.

[144] Xiantao Zhang, Dongchen Li, and Xihong Wu. Parsing named entity as syntactic structure. *INTERSPEECH*, pages 278–282, 2014.

[145] Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 253–263, Taipei, Taiwan, November 2017. Asian Federation of Natural Language Processing. URL https://www.aclweb.org/anthology/I17-1026.

[146] Lingyun Zhao, Lin Li, and Xinhao Zheng. A BERT based sentiment analysis and key entity detection approach for online financial texts. *arXiv preprint arXiv:2001.05326*, 2020.

[147] Lujun Zhao, Xipeng Qiu, Qi Zhang, and Xuanjing Huang. Sequence labeling with deep gated dual path cnn. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 27(12):2326–2335, 2019.

[148] Changmeng Zheng, Yi Cai, Jingyun Xu, Ho-fung Leung, and Guandong Xu. A boundary-aware neural model for nested named entity recognition. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 357–366, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1034. URL https://www.aclweb.org/anthology/D19-1034.

[149] Guodong Zhou, Jie Zhang, Jian Su, Dan Shen, and Chewlim Tan. Recognizing names in biomedical texts: a machine learning approach. *Bioinformatics*, pages 1178–1190, 2004.

[150] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.