**Uniwersytet Warszawski**
Wydział Psychologii

**Szymon Rutkowski**
Nr albumu: 332006

# Modele automatycznego poprawiania błędów w języku polskim

**Praca magisterska**
**na kierunku KOGNITYWISTYKA**
**w ramach Międzyobszarowych Indywidualnych Studiów Humanistycznych i Społecznych**

Praca wykonana pod kierunkiem
**dr. Jakuba Pawlewicza**
Wydział Matematyki, Informatyki i Mechaniki
**prof. dr hab. Joanny Rączaszek-Leonardi**
Katedra Psychologii Poznawczej

Październik 2018

## Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data Podpis kierującego pracą

## Oświadczenie autora (autorów) pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data Podpis autora pracy

## Streszczenie

Praca podejmuje zadanie uporządkowania problemu automatycznego poprawiania błędów w tekstach polskich, skupiając się na błędach nieprowadzących do powstania innego poprawnego słowa (niewyrazotwórczych). Przedstawione zostają podstawowe techniki używane w podobnych zastosowaniach w języku angielskim oraz dotychczasowa literatura dotycząca poprawy pisowni polszczyzny. Testom empirycznym poddano metody oparte na mierzeniu odległości edycyjnej oraz odległości wektorów znaczeniowych (ang. word embeddings), a także rekurencyjne sieci neuronowe przetwarzające wyrazy. W projektowaniu tych ostatnich użyto komórek LSTM i reprezentacji ELMo. Sieci neuronowe osiągnęły lepsze wyniki niż inne metody, zwłaszcza w wypadku odmian odczytujących słowa jednocześnie od początku i od końca (dwukierunkowo). Użycie zanurzeń ELMo pozwoliło na poprawienie wyników jeszcze bardziej.



## Słowa kluczowe

poprawianie pisowni, poprawność językowa, odległość edycyjna, word embeddings, sieci neuronowe, rekurencyjne sieci neuronowe, LSTM


## Dziedzina pracy (kod wg programu Socrates-Erasmus)

14.4 psychologia

## Tytuł pracy w języku angielskim

Models of Automatic Spelling Correction for Polish

# Spis treści

# Rozdział 1

# Wprowadzenie

Jeżeli język jest systemem znaków, które są określane przez ich wzajemne relacje, oraz dozwolonych połączeń tych znaków (Saussure 1991), to sensowność danej wypowiedzi zależy od jej zgodności z regułami języka. Mimo to wypowiedzi zaszumione w jakiś sposób, czy to przez błędy powstałe przy tworzeniu, czy też przy transmisji, często dają się nadal zrozumieć.

Konwencja językowa dyktuje zazwyczaj jedną uznaną formę zapisu każdego wyrazu. Teksty zgodne z regułami pisowni są łatwiejsze do odczytania zarówno dla człowieka, jak i dla sztucznych systemów przetwarzających język. Ponieważ jednak istnieją sytuacje, w których otrzymujemy jedynie wypowiedź zapisaną w sposób niepoprawny, można co najwyżej podjąć próbę wykorzystania redundancji mowy do wywnioskowania poprawnej formy i właściwego znaczenia tekstu.

## 1.1. Rodzaje błędów i zadań dla komputera

Karen Kukich (Kukich 1992) wydziela trzy główne problemy, jakich dotyczyły badania nad poprawianiem błędów językowych przez komputer. Są to:

1. Wykrywanie błędnych słów nienależących do słownika,
2. Poprawianie błędnych słów w izolacji,
3. Poprawianie błędnych słów w kontekście.

Metody wykrywania słów nienależących do słownika w większej mierze opracowano w latach 70. i 80. XX wieku. W tym okresie praca dotyczyła dopasowywania wzorców i porównywania napisów, żeby w efektywny sposób ocenić, czy dany wyraz należy do zbioru znanych słów. Zadanie znalezienia odpowiednich poprawek do tekstu pozostawiano użytkownikowi. Postęp informatyki, zarówno na polu oprogramowania, jak i sprzętu, stosunkowo uprościł to zadanie, które nigdy nie miało ściśle językowego charakteru.

Pozostałe dwa problemy, dotyczące wyboru poprawek do tekstu (jego korekcji), pozostają nadal bardzo aktualne dla dziedziny przetwarzania języka. W ich wypadku celem programu jest nie tylko wykrycie miejsc, gdzie tekst zawiera błędy, ale również wybór albo sugestia najbardziej prawdopodobnej poprawki.

Podstawowa korekcja błędów w izolacji obejmuje selekcję słów błędnych (nienależących do słownika), wyłonienie kandydatów na poprawki oraz ułożenie listy kandydatów w ranking

najlepszych korekt. Oczywiście bez kontekstu nie da się wykryć błędów, które doprowadziły do powstania innych poprawnych słów (tak jak *lak*, gdzie powinno być *łąk*) albo konsekwentnie dokonywać odpowiedniej selekcji, jeżeli błędna postać wyrazu zawiera niejednoznaczną informację (czy użytkownik, który napisał *wies*, miał na myśli *wieś* czy formę czasownika *wiesz*?). Stąd już na początku lat 90. Kukich zauważała, że efektywność ludzi w zadaniu izolowanej korekty słów jest podobna do skuteczności najlepszych algorytmów. Oznacza to udzielanie trafnej odpowiedzi dla około 75% słów w wypadku języka angielskiego.

Jednakże zadanie izolowanego, bezkontekstowego poprawiania tekstu ma pewną przewagę teoretyczną pomimo swoich ograniczeń: pozwala na zbadanie efektywności algorytmów we względnie uproszczonych warunkach. Można założyć, że najlepszy algorytm do korekty bezkontekstowej będzie dobrym kandydatem do poszerzenia o element kontekstowy.

Ponieważ specyfika korekty języka polskiego nie została dotąd dogłębnie zbadana, w niniejszej pracy skupiono się na poprawianiu izolowanych słów polszczyzny, w nadziei, że będzie to mogło posłużyć za punkt wyjścia do ambitniejszych poszukiwań.

## 1.2. Poprawianie błędnych słów polszczyzny w izolacji

Tak jak już wspomniano, na zadanie bezkontekstowego poprawiania słów składają się trzy podproblemy:

1. Wykrycie niepoprawnych słów (co zakłada dokonaną wcześniej segmentację tekstu na jednostki wyrazowe).
2. Wytworzenie słów-kandydatów, które mogłyby zastąpić błędny wyraz jako potencjalnie poprawne alternatywy.
3. Wyłonienie jednego słowa-kandydata (lub rankingu kandydatów), które z największym prawdopodobieństwem nada całemu dokumentowi oczekiwaną postać.

Aby stwierdzić, czy dany wyraz jest poprawny, wykorzystuje się zazwyczaj słownik. Możliwe jest także wykorzystanie algorytmu, takiego jak model n-gramowy (Hanson, Riseman et al. 1976) czy sieć neuronowa (Dronen 2016), przystosowanego do rozróżniania słów rzeczywistych od błędnych po wytrenowanych właściwościach. Wszystkie testowane przez nas rozwiązania korzystają jednak z metody słownikowej za pośrednictwem anotacji błędów w korpusie PlEWi (Grundkiewicz 2013). Dane słownikowe do niej zaczerpnięto z polskiej wersji programu Hunspell, chociaż wykorzystano także informacje o modyfikacjach poszczególnych słów wydobyte z historii edycji Wikipedii.

Jeżeli chodzi o wyłanianie i selekcję kandydatów, każda metoda stosuje własne algorytmy i oba te kroki są często ściśle ze sobą powiązane. Możliwe jest też wytwarzanie jednego, najlepszego kandydata od razu. Dzieje się tak zazwyczaj w wypadku metod działających nie tyle na całych wyrazach, ile na znakach.

Rozwiązywanie tych właśnie dwóch problemów – wytwarzania i oceny jakości poprawek – stało się główną kwestią podejmowaną w tej pracy.

## 1.3. Przyjęta terminologia

Sformułowania: *poprawianie błędów*, *korekta* czy *korekcja pisowni* oraz tym podobne stosowane są przez nas zamiennie, aby pomniejszyć monotonię językową pracy. Zamiennie są stosowane także terminy *wyraz*, *słowo* i *segment* (ang. *token*), chociaż w praktyce językoznawstwa i komputerowego przetwarzania języka często się je rozdziela. Mamy na myśli tutaj zawsze konkretną formę morfosyntaktyczną obecną w tekście, przy czym wersje poprawne i błędne tej formy uznajemy za w zasadzie ekwiwalentne w sensie przynależności do tego samego wyrazu. Dla formy błędnej słowa przyjmujemy określenie *błąd*, każda zaś potencjalna forma, którą program może zastąpić błąd, to dla nas *poprawka*. Zazwyczaj tylko jedna poprawka jest najlepsza i naprawdę odpowiednia.

## 1.4. Plan pracy

W rozdziale drugim niniejszej pracy opisujemy stan badań nad technikami poprawiania błędów niewyrazotwórczych (czyli nie przyjmujących postaci innego poprawnego wyrazu) w izolacji. Przyglądamy się najpierw głównym podejściom do tego problemu dla języka angielskiego, aby przejść do przeglądu literatury na temat poprawiania pisowni polszczyzny.

W rozdziale trzecim opisujemy modele poddane przez nas empirycznym testom. Zaczynamy od prostych modeli referencyjnych, których zadaniem jest ustalenie punktu odniesienia dla bardziej wyrafinowanych technik. Posłużą nam do tego dwa modele: jeden oparty na wynajdywaniu poprawek na podstawie odległości edycyjnej oraz drugi, działający przez podmianę diakrytyków – czyli zastępowanie liter specyficznych dla alfabetu polskiego przez ich odpowiedniki z podstawowego alfabetu łacińskiego oraz odwrotnie.

Jednym z przykładów metod bardziej złożonych będzie model posługujący się odległością wektorów semantycznych. Oprócz tego wypróbowanych zostanie kilka rodzajów rekurencyjnych sieci neuronowych typu LSTM, w tym jedna wykorzystująca opublikowaną niedawno technikę zanurzeń znaczeniowych ELMo.

W rozdziale czwartym zawarty jest opis techniczny eksperymentów oraz omówienie ich wyników. W rozdziale piątym dokonujemy podsumowania prac oraz wskazujemy na bardziej ogólne wnioski dla perspektyw dalszych badań nad automatycznym poprawianiem błędów w polskich tekstach.

# Rozdział 2

# Stan badań nad poprawianiem błędów w wyrazach

Korekta tekstu należy do najstarszych problemów w przetwarzaniu języka naturalnego, a także w historii oprogramowania komputerów w ogóle. Z tego względu istnieje ogromna literatura przedmiotu, chociaż przede wszystkim dla języka angielskiego. W tej pracy dokonujemy przeglądu jedynie najważniejszych nurtów i najnowszych prac, tak aby dać generalny obraz rozwoju dziedziny oraz miejsca, jakie zajmują w niej rozwiązania dla polszczyzny.

## 2.1. Podziały błędów na rodzaje

Błędy językowe mogą mieć różne przyczyny, które wpływają z kolei na charakter poszczególnych usterek w tekście. Najbardziej podstawowa klasyfikacja rozdziela błędy popełnione przez maszynę od tych popełnionych przez człowieka. Przykładem tych pierwszych są teksty zczytane z plików graficznych za pomocą oprogramowania OCR (ang. *optical character recognition*), a także teksty produkowane przez systemy rozpoznawania mowy. Korekcja błędów popełnianych przez programy rozpoznające tekst w obrazach i dźwiękach stanowi kierunek badań w dużej mierze oddzielny od korekty tekstów tworzonych bezpośrednio przez człowieka.

Także wśród błędów popełnianych przez ludzi wyróżnia się błędy typograficzne, kognitywne i fonetyczne (Kukich 1992). Te pierwsze wynikają z omsknięcia się i, jak pisze badaczka, „chwilowego niepowodzenia koordynacji ruchowej piszącego'‘. Błędy „kognitywne" (często po prostu ortograficzne, takie jak zamiany w ang. *receive* na *recieve* czy *conspiracy* na *conspiricy*) wynikają z niedostatecznej wiedzy użytkownika języka. Błąd fonetyczny to specjalna klasa błędu kognitywnego, gdzie wyraz zostaje zapisany fonetycznie trafnie, ale w sposób niepoprawny ortograficznie.

Z punktu widzenia czysto technicznego można wyróżnić błędy wstawienia, pominięcia i zamiany. Nazwy tych typów odnoszą się do operacji, która została dokonana w miejscu błędu.

Stosunkowo najdonioślejszym rozróżnieniem wśród błędów popełnianych przez piszących są błędy **wyrazotwórcze** i **niewyrazotwórcze** (ang. *word* i *non-word errors*). Błędy niewyrazotwórcze powodują po prostu powstanie napisu nieobecnego jako pozycja w słowniku. Błędy wyrazotwórcze prowadzą zaś do powstania innego, poprawnego wyrazu, przez co oprogramowanie komputerowe może mieć duży problem nawet z zauważeniem, że coś jest nie tak.

Przykładem błędu wyrazotwórczego może być błędny napis *wiec* (odpowiadający rzeczownikowi w języku polskim), kiedy piszący miał na myśli *więc*. Błędem niewyrazotwórczym byłby zaś napis *węic*, nieodpowiadający żadnemu istniejącemu słowu.

Możliwe są bardziej szczegółowe klasyfikacje błędów. Często powstają one z językoznawczego punktu widzenia, chociaż wydobywane prawidłowości i zjawiska mogą się potencjalnie przysłużyć rozwijaniu rozwiązań komputerowych. Praca Jana Bušty i współpracowników (Bušta, Hlavácková-Schindler et al. 2009) proponuje następujący podział dla tekstów czeskich:

1. Błędy zapisu: obejmujące błędy ortograficzne, pomylone końcówki wyrazów, niepoprawne użycie wielkich i małych liter itd.
2. Błędy interpunkcyjne.
3. Błędy leksykalno-semantyczne: niepoprawne użycie słów i wyrażeń wielosłownych, pomijanie całych słów.
4. Błędy stylistyczne: tutaj autorzy umieszczają używanie nieadekwatnych rejestrów języka (potoczny, archaiczny), złą kolejność wyrazów, zbyt długie zdania i problemy stylu specyficzne dla języka czeskiego.
5. Błędy typograficzne: przede wszystkim stosowanie niewłaściwych znaków (chodzi o rodzaje cudzysłowów czy myślników), a także niezadowalające formatowanie strony i użycie fontów.

Z kolei praca Ogrodniczuka i Kopcia (Ogrodniczuk i Kopeć 2017) wydziela następujące zjawiska pozasłownikowe – niekoniecznie błędy – w korpusie polskich tweetów:

1. Pomijanie polskich znaków diakrytycznych (znanych popularnie jako „polskie znaki").
2. Wprowadzanie nowych skrótów (*dzienn.* – dziennikarz) i skrótowców (*PDT* – premier Donald Tusk), a także często niepoprawny zapis istniejących.
3. Problemy z użyciem wielkich i małych liter oraz literówki.
4. Emotikony (które wymagają szczególnego traktowania ze strony systemów przetwarzania języka).
5. Wprowadzanie słów i fraz z języków obcych.
6. Inne: zwłaszcza mniej czy bardziej spontaniczne neologizmy oraz elementy slangu.

Duże konsekwencje ma zwłaszcza zauważenie pierwszej z tych kwestii – dużego udziału błędów powstających przez zamianę polskich diakrytyków na ich litery podstawowe albo, rzadziej, odwrotnie. Problemy mogą wynikać z niedbałości piszących, z trudności technicznych, lub z błędów typograficznych w ujęciu Kukich – czyli „omsknięć" przy wprowadzaniu znaków polskiego alfabetu.

## 2.2. Problem poprawiania błędów w literaturze światowej

W tym rozdziale przedstawimy krótką historię kluczowych podejść do poprawiania izolowanego błędów niewyrazotwórczych, przede wszystkim dla angielszczyzny. Chociaż wiele systemów, zwłaszcza stosowanych w praktyce oprogramowania użytkowego, łączy ze sobą różne rodzaje modeli, dzielimy je tutaj na cztery ogólne typy: oparte na odległości (zazwyczaj edycyjnej) między formami wyrazowymi, oparte na modelu probabilistycznym błędów lub języka, działające przez stosowanie systemu reguł oraz oparte na sieci neuronowej przetwarzającej teksty.

### 2.2.1. Odległość edycyjna

Od początku komputerowego przetwarzania tekstu próbowano podjąć problem literówek, opierając się na założeniu, że odpowiednia poprawka będzie podobna do błędnego wyrazu. Pionierskie były tutaj prace (Damerau 1964) i (Levenshtein 1966).

Odległość edycyjna dla dwóch ciągów znaków równa jest liczbie transformacji koniecznych, aby jeden ciąg stał się identyczny z drugim. Odległość Lewensztejna dopuszcza jako transformacje dodanie, usunięcie albo zamianę znaku. Odległość Damerau-Lewensztejna traktuje jako jedną transformację także zamianę dwóch sąsiednich znaków miejscami. Na przykład słowa *filtr* i *flirt* dzieli odległość Damerau-Lewensztejna równa dwa, ale odległość Lewensztejna równa cztery (trzeba dwukrotnie oddzielnie usunąć jedną literę z pary `il` oraz `tr` i później wstawić ją ponownie w innym miejscu).

Wspomniane miary odległości stworzone zostały od razu z myślą o ich zastosowaniu do poprawiania błędów. Poprawna forma wyrazu jest bowiem często do niego bliższa niż wszelkie inne słowa. Intuicyjnie wydaje się to obserwacja słuszna, zwłaszcza gdy chodzi o techniczne pomyłki przy pisaniu. Jeżeli jednak użytkownik napisze zupełnie inne słowo, niż chciał, albo ma błędne wyobrażenie o jego ortografii (co jest dotkliwe zwłaszcza w językach o niekonsekwentnym zapisie, takich jak angielski), wskazania odległości edycyjnej oparte na znakach mogą być mylące.

Pewną pomoc przynoszą tutaj algorytmy mające na celu nadanie podobnie wymawianym słowom podobnej reprezentacji, niezależnie od zawiłości ortografii. Dopiero na takiej umownej reprezentacji obliczana jest miara odległości edycyjnej. Tego typu algorytmy powstały bardzo wcześnie – na przykład popularny do dzisiaj SOUNDEX opatentowano w 1918 roku – i służyły do wyszukiwania nazwisk, których niekonsekwentny zapis wprowadzał komplikacje w wielu dziedzinach.

Najbardziej udoskonalonym systemem tego rodzaju jest, o ile nam wiadomo, Double Metaphone (Philips 2000). Na przykład napisy *photo* i *foto* (drugi z nich jest w angielskim niepoprawny) otrzymają w Double Metaphone taką samą postać `FT`, pomimo różnicy ortograficznej. Przy porównywaniu dwóch wyrazów można stosować właśnie taką postać konwencjonalną, dochodząc do wniosku, że *photo* jest dobrą poprawką dla *foto* (odległość zero).

Kolejnym rozszerzeniem idei szeregowania kandydatów według ich odległości od błędnego słowa jest włączenie do obliczeń odległości zanurzeń słownych (ang. *word embeddings*) (Nagata, Takamura et al. 2017). Autorzy wykorzystują przyporządkowanie do każdego segmentu w słowniku wektora liczb rzeczywistych, wyuczonego do modelowania znaczenia; istnieją ogólnodostępne zasoby gotowych wektorów wytrenowanych na dużych korpusach. Słowa występujące w podobnych sąsiedztwach (i przez to często także podobne semantycznie) otrzymują podobne wektory, bliskie sobie w wielowymiarowej przestrzeni.

Opisywana koncepcja bierze się z obserwacji, że jeżeli przestrzeń zanurzeń słownych wyuczona jest na korpusie zawierającym także segmenty błędne, to otrzymają one reprezentacje podobne do swoich poprawnych odpowiedników. Stąd odległość wektorów semantycznych została dodana do średniej ważonej, biorącej pod uwagę również odległość edycyjną nieprzetworzonych słów oraz ich postaci według wspomnianego algorytmu Double Metaphone. W eksperymentach tak poszerzona metoda osiągnęła lepsze wyniki niż inne przyjęte podejścia.

W praktycznie działających programach stosuje się metody optymalizujące wydobywanie pozycji ze słownika, unikając porównywania każdego błędu od początku, znak po znaku,

do wszystkich znanych wyrazów. Stosowane w tym celu są zwłaszcza automaty skończone (Hassan, Noeman et al. 2008; Pirinen i Lindén 2014). Umożliwia to działanie oprogramowania na tyle szybkie, by można było sprawdzać pisownię na bieżąco bez sprawiania niewygody użytkownikowi.

### 2.2.2. N-gramy i modele probabilistyczne

Ponieważ bardzo proste założenia stojące za odległością edycyjną okazują się niewystarczające do uchwycenia zjawisk istotnych dla problemu, wypracowano szereg podejść wydobywających bardziej złożoną wiedzę z korpusów tekstów. Eleganckim teoretycznie ujęciem takiej wiedzy jest probabilistyczny model języka, w którym zależności występowania w tekście jednych elementów od wystąpienia innych ujmowane są jako rozkłady prawdopodobieństwa.

W zadaniu poprawiania pisowni tworzony jest model wybierający najbardziej prawdopodobne poprawki dla błędów. Jego parametry ustala się na podstawie korpusu równoległego dokumentów (przyporządkowującego poprawne teksty ich błędnym odpowiednikom).

Jednym z podstawowych rodzajów modeli języka jest model n-gramowy, rozpatrujący tekst jako zbiór ciągów o długości $n$, zawierających następujące po sobie symbole. Przez „symbol" może być w tym wypadku rozumiany tak znak, jak i całe słowo. W pierwszym z tych podejść zdanie *Ala ma kota i jaszczurkę* będzie reprezentowane przez ciągi takie jak `Ala`, `la_`, `a_m`, `_ma` i tak dalej (jeżeli mowa o trigramach – n-gramach o $n = 3$), a w drugim przez `Ala ma kota`, `ma kota i`, `kota i jaszczurkę`.

Początkowo n-gramowe reprezentacje słów były wykorzystywane do ich wektoryzacji, co z kolei umożliwiało stosowanie matematycznych miar podobieństwa wektorów binarnych właściwości (Angell, Freund et al. 1983). Binarne właściwości segmentów to tutaj obecność w nich poszczególnych możliwych uni-, bi- czy trigramów (n-gramów o $n$ równym kolejno jeden, dwa i trzy).

Oczekujemy wówczas, że wektory błędów będą podobne w takiej n-gramowej przestrzeni do swojej poprawnej wersji. W tym sensie koncepcja przypomina odległość edycyjną, z tym że jest odporniejsza na lokalne przemieszania znaków. Ideę n-gramów da się w praktyce także stosować prościej (bez budowania wektorów), na przykład poprzez liczenie wspólnych bigramów (par) znaków, tak jak to robi narzędzie MultiSpell (Ahmed, Luca et al. 2009).

Często używa się informacji o n-gramach do zbudowania modelu probabilistycznego popełniania błędów, takiego jak ukryte modele Markowa (model zaszumionego kanału). Model tego rodzaju traktuje tekst jako ciąg stanów, odpowiadających kolejnym symbolom. Symbole definiuje się tu analogicznie do symboli w n-gramach, jako słowa bądź znaki. „Ukryty" charakter konstrukcji przejawia się w tym, że prawdziwy stan nie jest obserwowalny w całości i widoczne są tylko pewne jego objawy. W wypadku zadania poprawiania pisowni ukrytym stanem rzeczywistym będą znaki poprawnego wyrazu, a stanem widocznym (objawem) – znaki błędu.

Szacując prawdopodobieństwa przejścia od stanu do stanu (nastąpienia danego znaku w poprawnej wersji po innym) oraz emisji stanu widocznego przez ukryty (popełnienia danego błędu w danym stanie), można wyliczyć w ukrytych modelach Markowa najbardziej prawdopodobne prawdziwe słowo kryjące się za danym błędem. Poprawkę tę wnioskuje się z najbardziej prawdopodobnej ścieżki, jaką by można przejść po stanach modelu emitując obserwowany błąd.

Przyjmuje się upraszczające założenie, że prawdopodobieństwo wystąpienia znaku na każdej kolejnej pozycji nie zależy od całego wcześniejszego tekstu, lecz jedynie od kilku poprzednich stanów modelu – dokładnie $n$, gdzie $n$ oznacza założoną liczbę symboli w n-gramach. Prawdopodobieństwa przejść między stanami ukrytymi oraz emisji stanów pozornych szacuje się na podstawie korpusu.

Na przykład w modelu probabilistycznym w pracy (Bassil i Alwani 2012) wykorzystano udostępnione przez Google informacje o n-gramach występujących w Internecie, stosując zarówno n-gramy złożone ze słów, jak i ze znaków. Błędy są wykrywane na podstawie n-gramów wyrazowych (co pozwala też uchwycić elementy kontekstu), ale dobór poprawek dokonuje się już na podstawie modeli zbudowanych za pomocą n-gramów znakowych.

Tworzone były także rozwiązania probabilistyczne, biorące pod uwagę najbliższy i dalszy kontekst słowa bez trzymania się modelu n-gramowego (Golding 1995). Metoda Goldinga wykorzystywała obecność konkretnych słów albo ciągów wyrazów jako przesłanki dla klasyfikatora bayesowskiego, wybierającego odpowiednią poprawkę ze zbioru możliwości. Zarówno prawdopodobieństwa błędów, jak i poprawek są szacowane na podstawie korpusu.

### 2.2.3. Metody regułowe

Do problemu poprawiania pisowni można podejść w sposób niestatystyczny, poprzez zaimplementowanie reguł opartych na wiedzy lingwistycznej – zazwyczaj konstruowanych ręcznie.

Warto co prawda zwrócić uwagę, że reguły mogą być także automatycznie tworzone z pomocą korpusu (Mangu i Brill 1997). Dzieje się to w sposób przypominający w swoim działaniu podejście probabilistyczne, chociaż nie operujący na prawdopodobieństwach wprost. W tym wypadku zachowanie modułu poprawy pisowni może zależeć od słów napotykanych w najbliższym kontekście. Przykładem takiej reguły byłoby poprawianie słowa *principle* (zasada) na *principal* (dyrektor), jeżeli w pobliżu znajdzie się wyraz *school*. Jak widać, systemy regułowe często starają się wykorzystać kontekst do podjęcia problemu błędów wyrazotwórczych.

Ręcznie wytwarzane reguły powstały między innymi dla programu *hunspell* (wykorzystywanego przez wiele aplikacji open source) i jego poprzedników, sięgających *ispell* oraz *spell* z lat siedemdziesiątych. W tym wypadku reguły służą pomocniczo do analizy morfologicznej wyrazów i opierają się głównie na rozpoznawaniu końcówek i przedrostków – to znaczy, w terminologii lingwistycznej, afiksów. Zasoby przygotowane na potrzeby otwartego oprogramowania do korekty błędów zostały następnie wykorzystane przez uczonych budujących systemy z myślą o samej analizie morfologicznej (Trón, Kornai et al. 2005; Ahn 2017).

Istotnym współczesnym przykładem zastosowania podejścia regułowego jest LanguageTool (Miłkowski 2010). Reguły są formułowane ręcznie i w większości przechowywane w plikach XML. Dotyczą one zarówno konkretnych segmentów, jak i wyrazów należących do którejś rozpoznanej części mowy (ang. *parts of speech* – POS). W wypadku języka polskiego, jak zauważa Miłkowski, ekspresywność przyjętego formatu XML okazała się niewystarczająca dla złożoności systemu języka. Okazało się konieczne wprowadzenie m.in. operatorów logicznych i bardziej rozbudowanej obsługi wyjątków oraz fleksji. LanguageTool zawiera również zestawy reguł dla języków takich jak angielski i niemiecki. Stosunkowo niedawno zaprezentowano hybrydowy system regułowy dla indonezyjskiego, włączający elementy wnioskowania statystycznego w postaci modeli Markowa (Fahda i Purwarianti 2017).

### 2.2.4. Sieci neuronowe

Rozwój wykorzystywania sieci sztucznych neuronowych do poprawy pisowni szedł w parze z postępem na polu dostępnej mocy obliczeniowej i algorytmów uczących. Co prawda wczesne przykłady zastosowania sieci neuronowych pojawiły się niedługo po wprowadzeniu algorytmu propagacji wstecznej (Kukich 1988), ale najbardziej aktywne badania rozpoczęły się w epoce uczenia głębokiego (ang. *deep learning*). Ponieważ sieć neuronowa ma teoretyczną zdolność przybliżania dowolnych funkcji, powinna potrafić nauczyć się poprawiania błędów z odpowiednio dużego korpusu.

Sieci neuronowe można stosować albo jako elementy większego systemu, albo jako samodzielne rozwiązania kompleksowej poprawy pisowni. Pierwsze z tych podejść zapewnia nieco lepszą interpretowalność działania systemu, to drugie zaś pozwala procesowi uczenia dowolnie wynajdywać najbardziej skuteczne architektury. Dronen (Dronen 2016) trenował oddzielnie sieci konwolucyjne dla zadania rozpoznawania błędów, wydobywania korekt ze słownika oraz szeregowania ich według trafności. Dokonano eksperymentów zarówno dla sytuacji bezkontekstowej, jak i ze znanym kontekstem na błędach wyrazotwórczych.

Częściej spotykane jest jednak podejście, w którym do sieci podawane jest całe słowo albo tekst, które może ona poprawiać wedle uznania. Wykorzystuje się tutaj podobne architektury jak dla innych zadań uczenia maszynowego, zwłaszcza dotyczących przetwarzania języka – przykładem może być sieć rekurencyjna typu RNN (ang. *recurrent neural network*), symulująca ludzką umiejętność odczytywania wyrazów z wymieszanymi literami (Sakaguchi, Duh et al. 2017). W dość częstym użyciu znajdują się mechanizmy *seq2seq*, które zakodowują tekst do pewnej wewnętrznej reprezentacji, by później odkodować go do poprawnej formy (Schmaltz, Kim et al. 2017).

Modele neuronowe wyuczane są też obecnie do znacznie ambitniejszego zadania, niż zwyczajne poprawianie literówek w angielskim: korygowane mają być wszelkie niedoskonałości gramatyki w tekście, tak jak we wspólnym zadaniu CoNLL 2014 (Ng, Wu et al. 2014). Badacze zastosowali rozwiązania zaczerpnięte z tłumaczenia maszynowego pomiędzy różnymi językami (Yuan i Felice 2013; Chollampatt, Taghipour et al. 2016; Junczys-Dowmunt, Grundkiewicz et al. 2018). Na krótko przed ukończeniem tej pracy raportowano osiągnięcie skuteczności analogicznej do ludzkiej przy poprawianiu błędów gramatycznych (Ge, Wei et al. 2018). Postęp okazał się tu możliwy dzięki automatycznemu powiększaniu korpusu treningowego o zdania częściowo poprawione oraz zdania sztucznie zepsute przez system (przez odwrócenie kierunku działania istniejącego modelu – z poprawiania na uszkadzanie). Z tego względu nauka sieci w kolejnych iteracjach może brać pod uwagę wiele podobnych do siebie zdań, zawierających jednak nieco odmienne zjawiska językowe.

## 2.3. Poprawianie błędów w języku polskim

Problem poprawiania błędów w języku polskim z konieczności pojawił się wcześnie, ale wykazuje, w porównaniu z innymi językami, mniejszą ciągłość literatury.

Wiadomo, że wczesne oprogramowanie, jeszcze na sprzęt krajowej produkcji taki jak komputery K-202 i ODRA 1305, próbowało znaleźć w słowniku wyraz o odległości edycyjnej równej jeden i zastąpić nim błędne słowo na wejściu (Subieta 1976, 1985). Na odległości edycyjnej opierały się długo inne systemy, takie jak te obecne w edytorach tekstu. Pojawiły się postulaty

korekty wag, żeby uwzględnić częstość błędów polegających na unikaniu polskich liter, oraz projekty kompresji słownika dzięki wiedzy o końcówkach morfologicznych (Dorosz 2008). Na polu poprawiania błędów z uwzględnieniem kontekstu zaproponowano Grafy Przyzwyczajeń Lingwistycznych, modelujące język w sposób nieco podobny do n-gramów, jednak oszczędzające pamięć i dokonujące pewnej ekstrapolacji informacji z korpusu (Gadamer i Horzyk 2009).

Na polu praktycznych zastosowań opublikowano opis systemu przystosowanego specjalnie do poprawiania raportów z badań mammograficznych (Mykowiecka i Marciniak 2007). Fundamentem tego rozwiązania jest ponownie odległość Lewensztejna (zob. 2.2.1), lecz posiada ono także komponenty odpowiedzialne za słowa sklejone, błędy interpunkcyjne i przewidywanie korekty na podstawie kontekstu (reprezentowanego za pomocą bigramów). Wyrazy różniące się brakiem i obecnością znaków diakrytycznych uznawane są za bliższe edycyjnie niż wyrazy różniące się w inny sposób.

Miłkowski w swoim sprawozdaniu z prac nad rozwojem LanguageTool odnosi się również do problemów polszczyzny (Miłkowski 2010). Tradycyjne moduły poprawiania pisowni często potrafią wytwarzać więcej szkód niż pożytku, na przykład forsując (wbrew zasadom ortografii) rozdzielną pisownię *nie* z rzeczownikami, kiedy wersja bez *nie* należy do słownika. Piętą achillesową wielu rozwiązań są także błędy prowadzące do powstawania bardzo rzadko używanych wyrazów: na przykład istnieje teoretycznie forma *musze* jako celownik słowa *mucha*, ale zwykle taki segment powstaje przez błędny zapis słowa *muszę*. Zbiór reguł LanguageTool jest w stanie elastycznie radzić sobie z tego rodzaju problemami. Dzięki wbudowanej analizie morfologicznej i swojemu wzbogaconemu językowi reguł jest on w stanie na przykład pilnować związków zgody w zdaniu.

LanguageTool znalazł się wśród rozwiązań wypróbowanych stosunkowo niedawno przy poprawianiu pisowni polskich tweetów (Ogrodniczuk i Kopeć 2017). Autorzy zauważają, że w internetowej praktyce niestandardowego języka i odnoszenia się wciąż do bieżących zdarzeń (przez co słowniki stają się przestarzałe) wiedza językowa zawarta w oprogramowaniu korygującym okazuje się nieadekwatna. Przykładami są wprowadzone poprawki frazy *Baracka Obamy* na *Baranka Obawy* oraz wyrazu *smartfonów* na *smart fonów*.

Jak się okazało, na zebranym korpusie tweetów nawet prosta metoda, ograniczająca się do wyszukiwania poprawek w słowniku przez dodawanie elementów diakrytycznych (kropek, kresek) do liter, okazała się skuteczniejsza niż LanguageTool. Pokazuje to, że dla każdego specyficznego podzbioru języka najprawdopodobniej optymalne są przystosowane do niego rozwiązania.

# Rozdział 3

# Modele sprawdzane doświadczalnie

Za cel w tej pracy postawiliśmy sobie przede wszystkim sprawdzenie metod inspirowanych trendami badawczymi na polu przetwarzania języka naturalnego, które pojawiły się względnie niedawno. Dlatego głównym przedmiotem tekstów są modele wykorzystujące zanurzenia wektorowe wyrazów oraz rekurencyjne sieci neuronowe. Jednakże, aby móc ocenić rzeczywistą wartość tych podejść, do tego samego zadania zastosowaliśmy także dwie proste metody znane od dawna w literaturze przedmiotu.

## 3.1. Referencyjne modele poprawiania błędów

Jako punkty referencyjne dla skuteczności innych modeli posłużyło nam obliczanie zwykłej odległości edycyjnej, znane od lat 60. XX wieku, oraz podmianę diakrytyków, której koncepcja przewijała się w wielu pracach o korekcie polszczyzny, ale została wyraźnie sformułowana w pracy o korekcie języka tweetów (Ogrodniczuk i Kopeć 2017).

### 3.1.1. Odległość edycyjna

Model korekty błędów na podstawie odległości edycyjnej korzysta ze Słownika Gramatycznego Języka Polskiego (Saloni, Woliński et al. 2018) do tworzenia listy kandydatów, którzy są najbliżsi danej formie błędnej spośród wyrazów istniejących w słowniku. Następnie jako poprawka wybierana jest ta pozycja ze słownika, która znajduje się w najmniejszej odległości od błędu. W algorytmie ustalania odległości jest wykorzystywana wersja Lewensztejna, która traktuje wstawienie, zamianę lub usunięcie znaku jako odsunięcie się na odległość równą jeden. Zamiana sąsiednich znaków miejscami, w przeciwieństwie do odległości Damerau-Lewensztejna, powoduje odsunięcie się na odległość równą dwa.

### 3.1.2. Podmiana diakrytyków

Prace zajmujące się korektą polszczyzny w praktyce zwracają uwagę na fakt, że za bardzo duży odsetek błędów odpowiada pomijanie znaków diakrytycznych – diakrytyków (Mykowiecka i Marciniak 2007; Ogrodniczuk i Kopeć 2017). Znaki diakrytyczne powstają z podstawowych

liter alfabetu łacińskiego przez dodanie dodatkowych elementów graficznych, takich jak kreski czy ogonki.

Zaniedbywanie ich stosowania wynika w pewnej mierze ze stosowania w Polsce międzynarodowego układu klawiatury (tzw. klawiatury programisty), w którym znaki specyficzne dla polskiego alfabetu nie mają przypisanych własnych klawiszy. Ich wprowadzenie wymaga użycia przycisku `AltGr`, którego wciskanie może być zaniedbywane przez użytkowników, albo nawet może im się ono nie udawać. Historycznie niektóre komputery nie umożliwiały pisania w alfabecie polskim w ogóle. Pochodzi stąd dziedzictwo częstej nieobecności polskich znaków w komputerowych tekstach. Użytkownicy języka zwykle są w stanie odczytać słowa zapisane w ten sposób, nie są to jednak formy poprawne.

Ogrodniczuk i Kopeć (2017) użyli mechanizmu poprawiającego błędy jedynie przez wyszukiwanie poprawnych wyrazów dających się uzyskać z formy wyjściowej przez dodawanie i usuwanie elementów diakrytycznych do liter. Co ciekawe, taka metoda okazała się bardziej skuteczna nawet niż kilka możliwych ustawień regułowego narzędzia LanguageTool. Sugeruje to, że takie proste podejście może stanowić dobre porównanie dla złożonych rozwiązań.

Użyliśmy metody rozpatrującej wszystkie wyrazy dające się uzyskać z błędu poprzez zamianę `a` na `ą`, `e` na `ę` i tak dalej dla wszystkich polskich znaków diakrytycznych, dopuszczając również transformacje odwrotne. Uznaliśmy, że znaki `z`, `ż` i `ź` są wszystkie podmienialne na siebie nawzajem. Utworzona w taki sposób lista kandydatów jest zawężana do wyrazów znanych w Słowniku Gramatycznym Języka Polskiego. Spośród nich wyraz o najmniejszej odległości edycyjnej od formy wyjściowej wybierany jest jako właściwa poprawka i decyzja modelu.

Trzeba zaznaczyć, że w praktyce metoda ta powoduje znaczne obciążenie obliczeniowe, przynajmniej w naiwnej implementacji – bez usprawnień takich, jak automaty skończone. Znajdujący się w korpusie PlEWi błędny segment *Modlin-Zegrze-Pultusk-Różan-Ostrołęka-Łomża-Osowiec.* (poprawna forma zawiera *Pułtusk* przez `ł`) umożliwia dojście do ponad $2^{29} = 536.870.912$ stanów wedle tej metody, nie licząc nawet większej liczby możliwości produkowanej przez znaki `z` i `ż`. Stąd w testach wprowadziliśmy ograniczenie: model podejmuje próbę znalezienia poprawki jedynie dla wyrazów o długości nie większej niż 17 znaków. (Takie samo ograniczenie, z innych przyczyn technicznych, przyjęliśmy w modelu sieci rekurencyjnej).

## 3.2. Odległość wektorów semantycznych

Szacowanie najlepszych poprawek słów na podstawie samej odległości edycyjnej ma dobrze znane słabe punkty. Na przykład wyraz *marzę* zapisany omyłkowo jako *mazrę* jest równie bliski słowu *mażę* według algorytmu Lewensztejna (odległość równa 2). Dla *mażę* w pierwszym kroku odejmujemy `r`, a w drugim, zamiast dodać je ponownie w dobrym miejscu, zamieniamy `z` na `ż`. Nawet systemy konwersji tekstu do zapisu motywowanego fonologicznie, na którym następnie mierzy się odległość edycyjną, mają problem z obsługą wymieszanych dwuznaków. Dwuznaki są zaś dosyć częste w języku polskim.

Sposób obejścia tego problemu proponowany przez badaczy zajmujących się błędami uczniów języka angielskiego (Nagata, Takamura et al. 2017) upodabnia nieco zachowanie maszyny do metod wykorzystywanych przez człowieka, ponieważ wykorzystuje semantyczne dopasowywanie poprawki do kontekstu. Co prawda w zadaniu poprawiania bezkontekstowego sąsiednie słowa nie są bezpośrednio dostępne, ale można je wykorzystać na etapie uczenia, stwierdzając, w jakich typach kontekstów pewne błędy są najbardziej prawdopodobne.

Cytowana praca wykorzystała zatem sumę odległości edycyjnej i odległości wektorów znaczeniowych do poprawiania wypracowań pisanych przez osoby uczące się języka angielskiego. Sposób ten sprawdził się najlepiej w porównaniu z „czystą" odległością edycyjną oraz modelami opartymi na n-gramach wyrazowych i znakowych. Podbudowa teoretyczna dla wykorzystanej przez nas wersji tego rozwiązania przedstawiona jest w tym rozdziale.

### 3.2.1. Semantyka dystrybucyjna

Koncepcja wykorzystania odległości wektorów znaczeniowych stanowi przykład kreatywnego wykorzystania osiągnięć semantyki dystrybucyjnej. Jest to gałąź językoznawstwa, a w ostatnich dekadach również lingwistyki obliczeniowej skupiająca się na wydobywaniu z dużych korpusów tekstów informacji semantycznej w formie liczbowej. Hasło przyświecające temu kierunkowi badań sformułował w 1957 roku John R. Firth: *You shall know a word by the company it keeps* (*Poznacie słowo po jego towarzystwie*). Kryje się tutaj obserwacja, że słowa powiązane ze sobą znaczeniowo często pojawiają się w wypowiedziach w pobliżu podobnych wyrazów: innymi słowy, mają podobne sąsiedztwo.

Metoda, jaką wykorzystaliśmy, wykorzystuje fakt, że słowa o niepoprawnej pisowni mają to samo znaczenie i, co za tym idzie, podobne sąsiedztwo, co ich poprawne odpowiedniki. Duże korpusy, które zwykle nie są dodatkowo poprawiane, podobnie jak zwykłe dokumenty zawierają błędy. Owe niepoprawne formy otrzymują swoje reprezentacje znaczeniowe razem ze wszystkimi wyrazami. Nawet niewielka liczba wystąpień błędnych postaci słów pozwala upodobnić je do poprawnych postaci, choć oczywiście te drugie uzyskują więcej rzeczywistej informacji. Można więc założyć, że słowa o najbardziej podobnym znaczeniu do błędnego wyrazu będą dobrymi kandydatami na poprawki.

W pracach z zakresu semantyki dystrybucyjnej reprezentuje się zazwyczaj słowa jako wektory liczb rzeczywistych – mówi się o nich jako o zanurzeniach wyrazowych (ang. *word embeddings*). Takie przechowywanie informacji semantycznej umożliwia względnie wygodne wyuczanie jej i wykorzystywanie do zadań takich jak mierzenie podobieństwa znaczeniowego wyrazów.

Zanim popularne stały się sieci neuronowe, wydobywano takie informacje z macierzy współwystępowań wyrazów w dokumentach. Te następnie przetwarzano za pomocą metod redukcji informacji (takich jak SVD – z ang. *Singular Value Decomposition*) do postaci o mniejszej liczbie wymiarów, łatwiejszej do udźwignięcia obliczeniowo. Jedno z najważniejszych podejść tego typu to analiza semantyki ukrytej (ang. *Latent Semantic Analysis*) (Deerwester, Dumais et al. 1990). Obecnie powszechnie stosowane do uzyskiwania reprezentacji są sieci neuronowe, które w ogóle nie wymagają tworzenia macierzy współwystępowań wyrazów dla całego korpusu. Pozwala to na istotne uproszczenie aspektu obliczeniowego i wykorzystanie większych zbiorów danych tekstowych.

Postęp w wyuczaniu reprezentacji znaczeń zaczął umożliwiać używanie jej w sposób bardziej wyrafinowany niż samo sprawdzanie podobieństwa. Zaprezentowano próby używania pewnego rodzaju arytmetyki, w której na przykład suma wektorów reprezentujących wyrazy *German* i *airlines* jest bliska reprezentacji *Lufthansa* (Mikolov, Sutskever et al. 2013). Pokazuje to, że wektorowe zanurzenia wyrazowe mogą potencjalnie odzwierciedlać różne niuanse znaczeniowe wyrazów, pozwalając nawet na wyodrębnianie poszczególnych podczynników. Owa potencjalna kompozycyjność semantyki dystrybucyjnej nie jest jednak szczególnie istotna dla niniejszej pracy.

### 3.2.2. Word2vec

Wektory wykorzystane przez nas zostały uzyskane przez zespół z Instytutu Podstaw Informatyki PAN (Mykowiecka, Marciniak et al. 2017) za pomocą oprogramowania GenSim (Řehůřek i Sojka 2010). Korzysta ono z algorytmu word2vec (Mikolov, Sutskever et al. 2013). Wersja użyta dla tej pracy została obliczona przy ustawieniach GenSim do użycia próbkowania negatywnego (ang. *negative sampling*) i skipgramów; znaczenie owych terminów objaśniamy poniżej.

Algorytm word2vec produkuje wektory jako skutek uboczny treningu płytkiej sieci neuronowej do modelowania języka. Wczesne prace nad zanurzeniami wektorowymi wyrazów formułowały modelowanie języka po prostu jako przewidywanie kolejnych słów w ciągu (Bengio, Ducharme et al. 2003). Jednym z osiągnięć zespołu Mikolova (Mikolov, Sutskever et al. 2013) było uproszczenie treningu poprzez ograniczenie przestrzeni możliwych odpowiedzi. Reprezentacje nazwane przez nich *continuous bags of words* i skipgramami dążą do uproszczenia kontekstu rozpatrywanego przez sieć neuronową. Zamiast reprezentować całą sekwencję sąsiednich słów, pobierana jest albo średnia ich wektorów (w wypadku *continuous bags of words*) albo tylko jedno losowe słowo z sąsiedztwa (tworzące „skipgram" razem ze słowem wyjściowym). Taką uproszczoną informację podaje się na wejście sieci.

Kolejne triki techniczne w wyuczaniu reprezentacji semantycznych dotyczą formatu słowa zwracanego na wyjściu modelu neuronowego. Początkowo (Bengio, Ducharme et al. 2003) stosowana była funkcja softmax, która wytwarza rozkład prawdopodobieństwa po wszystkich możliwych wyrazach. Pierwszym usprawnieniem był softmax hierarchiczny (Mnih i Hinton 2009), gdzie słowa uporządkowane są w drzewo binarne, a wektor na wyjściu sieci reprezentuje ścieżkę prowadzącą do odpowiedniej pozycji w słowniku. Wreszcie, próbkowanie negatywne rezygnuje w ogóle z przewidywania konkretnych słów na rzecz klasyfikacji binarnej: czy dane dwa słowa prawdopodobnie wystąpiłyby w swoim sąsiedztwie, czy nie? Okazuje się, że takie bardzo uproszczone zadanie wciąż wymaga podobnie dużej wiedzy o języku i, co za tym idzie, wymusza na sieci neuronowej wytworzenie odpowiedniej reprezentacji wektorowej wyrazów.

Metoda word2vec nie stanowi, ściśle rzecz biorąc, przykładu uczenia głębokiego, ponieważ model jest względnie prosty i zawiera tylko jedną warstwę ukrytą. Dzięki temu złożoność obliczeniowa staje się również relatywnie niewielka i można się skupić na trenowaniu sieci na jeszcze większych korpusach.

### 3.2.3. Odległość zanurzeń wektorowych jako wskaźnik podobieństwa wyrazów

Tradycyjnie stosowaną w semantyce dystrybucyjnej miarą podobieństwa wektorów jest podobieństwo kosinusowe (i wywiedziona z niego odległość kosinusowa). Owa miara bierze pod uwagę różnicę w kierunku wektorów, abstrahując od konkretnych długości w poszczególnych wymiarach. Jest to przewaga tej metody ponad bardziej intuicyjnymi miarami, takimi jak odległość euklidesowa w przestrzeni wielowymiarowej – będąca po prostu uogólnieniem odległości między punktami na płaszczyźnie. Odległość kosinusowa nie daje się „zwieść" niewielkim czy dużym wartościom komórek wektorów, zwracając uwagę głównie na fakt ich różnienia się lub nie.

Podobieństwo kosinusowe wektorów $u$ i $v$ podaje się jako kosinus kąta $\theta$ pomiędzy tymi wektorami (w przestrzeni o liczbie wymiarów odpowiadającej długości wektorów):

$$\cos(\theta) = \frac{u \cdot v}{\|u\| \cdot \|v\|} \tag{3.1}$$

W powyższym równaniu $a \cdot b$ oznacza iloczyn skalarny wektorów:

$$a \cdot b = a_1 b_1 + a_2 b_2 + ... + a_n b_n, \tag{3.2}$$

co umożliwia nam także zdefiniowanie $\|a\|$, czyli normy euklidesowej $a$, jako:

$$\|a\| = \sqrt{a \cdot a} \tag{3.3}$$

Miara podobieństwa podana w równaniu (3.1) przyjmuje większe wartości dla podobnych wektorów (odpowiadających na przykład wyrazom o pokrewnej semantyce), a mniejsze dla wektorów mniej podobnych. Co za tym idzie, właściwą odległość kosinusową rozumie się jako podobieństwo kosinusowe wektorów odjęte od jedności:

$$\text{dist}(u, v) = 1 - \frac{u \cdot v}{\|u\| \cdot \|v\|} \tag{3.4}$$

Jest to miara, którą stosujemy do ustalania stopnia różnienia się kandydatów od wyjściowej błędnej formy.

### 3.2.4. Wykorzystanie odległości kosinusowej do poprawiania tekstów

Idąc za przykładem przywoływanej pracy (Nagata, Takamura et al. 2017), wykorzystaliśmy odległość wektorów znaczeniowych, łącznie z odległością edycyjną, do podejmowania decyzji w korekcie pisowni. Właściwą odległość pomiędzy wyrazem błędnym a kandydatem na poprawkę definiujemy zatem jako sumę odległości edycyjnej pomiędzy napisami oraz odległości kosinusowej między ich reprezentacjami wektorowymi.

Dość istotnym teoretycznym problemem jest kwestia skalowania obu rodzajów odległości – potencjalnie miara o większej wartości bezwzględnej wartości dominowałaby nad drugą. Jednak w praktyce, jak stwierdziliśmy, odległość kosinusowa pomiędzy zanurzeniami wektorowymi mieści się w zakresie $[0, 1]$, a wartość odległości edycyjnej zwracana przez bibliotekę PyLucene – wykorzystywaną w eksperymentach – również jest już przeskalowana do tego przedziału. Sprawia to, że skala danych o obu rodzajach odległości jest taka sama i zachowują się, jakby otrzymywały równe wagi. Kiedy brakowało wektora znaczeniowego dla jednego ze słów (czy to błędu, czy poprawki), przyjmowano zastępczą odległość znaczeniową równą jeden.

Trzeba podkreślić, że słowa brane pod uwagę przy obliczaniu odległości wektorowej należą do dziesięciu najbliższych błędnej formie według samej miary odległości edycyjnej. Ograniczenie to umotywowane jest zarówno koncepcyjnie (przy korekcie bezkontekstowej nie ma sensu brać pod uwagę słów skrajnie różnych graficznie od formy wyjściowej), jak i technicznie – ponieważ dostępne są zaimplementowane rozwiązania techniczne sprawnego wydobywania najbliższych sąsiadów dla odległości edycyjnej, ale nie (o ile nam wiadomo) dla odległości wektorów.

Słowo o najniższej sumie odległości edycyjnej i wektorowej zwracane jest jako poprawka wybrana przez model.

## 3.3. Rekurencyjne sieci neuronowe

W ciągu ostatniej dekady sieci neuronowe stały się standardowym podejściem do wielu problemów, w których stosuje się metody uczenia maszynowego. Na polu przetwarzania języka sieci rekurencyjne odniosły sukcesy, między innymi, w tagowaniu morfosyntaktycznym (Plank, Søgaard et al. 2016), streszczaniu tekstów (Nallapati, Xiang et al. 2016), a także w poprawianiu błędów gramatycznych (o czym wspomniano w Rozdziale 2). Stąd uznaliśmy, że bez wypróbowania głębokich sieci neuronowych nie osiągnęlibyśmy pełnego obrazu możliwości, jeżeli chodzi o poprawianie błędów w polszczyźnie.

### 3.3.1. Long Short-Term Memory

Sieci neuronowe w zadaniach obejmujących wytworzenie większej liczby podobnych elementów, takich jak słowa czy znaki, narażone są na problem braku generalizacji. Jeżeli, na przykład, każdy kolejny znak słowa przyjmowanego przez sieć jest reprezentowany przez zestaw swoich własnych komórek o oddzielnych połączeniach do kolejnych warstw, algorytm musi nauczyć się tej samej wiedzy dla pierwszego, drugiego (itd.) znaku.

Zjawisko to można złagodzić przez użycie sieci konwolucyjnej, przykładającej zestaw takich samych filtrów do wielu podzbiorów wejścia sieci. Jest to technika szczególnie często stosowana przy przetwarzaniu obrazów, ale znajduje również zastosowanie w pracy z językiem naturalnym.

Możemy jednak chcieć nie tylko używać tego samego zestawu wag do każdego kolejnego elementu wejścia (takiego jak znak), ale i zapamiętywać stan pomiędzy kolejnymi wejściami. Takim wymaganiom odpowiada rekurencyjna sieć neuronowa operująca na szeregu słów albo znaków po kolei, czyli o architekturze znanej po angielsku po prostu jako *recurrent neural network* (RNN). Czytając słowo *kot*, algorytm może przy czytaniu litery `t` zapamiętywać szczegóły z czasu, kiedy odczytywał znaki `k` i `o`. Sieć typu RNN ma w każdym momencie nie tylko swój normalny stan aktywacji (analogicznie jak każdy rodzaj warstwy neuronowej), ale także stan ukryty, gdzie przechowywana jest jej pamięć. O rekurencji można mówić, ponieważ komórka RNN przetwarza również własny stan ukryty i podaje go do ponownego przetworzenia przez siebie w kolejnej jednostce czasu działania sieci.

W komórce *Long Short-Term Memory* (Hochreiter i Schmidhuber 1997) zachowanie warstwy rekurencyjnej kontrolują bramki: zapominająca, zapamiętująca i przepuszczająca. Bramka ($g$) jest złożeniem funkcji liniowej i sigmoidalnej, działającym na wektorze utworzonym przez sklejenie wektora na wejściu ($X$) i ukrytego wektora reprezentującego pamięć ($H$):

$$g = \sigma(W_g(X, H) + b_g) \tag{3.5}$$

Sklejanie wektorów reprezentujemy oznaczeniem $(A, B)$: jest to operacja, na której skutek otrzymujemy kopię wektora $A$ z kopią zawartości $B$ dodaną na końcu. Zamiast łączenia wektorów w ten sposób mogą być też zastosowane oddzielne wagi dla istniejącego stanu ukrytego i nowego wejścia. Funkcja sigmoidalna służy przełamaniu liniowej zależności wartości bramki od jej wejścia i umiejscawia odpowiedź w przedziale $[0, 1]$.

Wartość zwracana przez bramkę (należąca, jak wspomnieliśmy, do przedziału $[0, 1]$ w każdym punkcie) jest następnie używana do obliczenia wartości komórki LSTM w momencie $t$ (oznaczanej jako $C_t$) oraz jej stanu ukrytego ($H_t$):

$$C_t = g_f \cdot C_{t-1} + g_p \cdot \text{tgh}(W_C(X, H) + b_C) \tag{3.6}$$

$$H_t = g_r \cdot \text{tgh}(C_t) \tag{3.7}$$

W powyższych równaniach $g_f$ oznacza bramkę zapominającą, $g_p$ bramkę przepuszczającą, a $g_r$ bramkę zapamiętującą; $\text{tgh}(x)$ to standardowe oznaczenie tangensa hiperbolicznego.

W większości zastosowań pożyteczne jest przetwarzanie sekwencji wejściowej jednocześnie przez dwie sieci, jedną czytającą od początku i jedną od końca. Ich odpowiedź jest następnie uśredniana dla każdego kolejnego znaku wyjścia. Powstaje w ten sposób dwukierunkowa sieć rekurencyjna (Schuster i Paliwal 1997).

### 3.3.2. ELMo

Zazwyczaj stany ukryte komórek LSTM są inicjalizowane losowym szumem albo wyuczane w procesie treningu. W doświadczeniach opisywanych w tej pracy podjęliśmy próbę wykorzystania istniejącego modelu języka dla ułatwienia działania systemowi poprawy pisowni. Idea ta bierze się z faktu, że ludzie przy odczytywaniu tekstów z błędami wykorzystują swoją wiedzę o języku w ogóle. Byłoby więc pożyteczne udostępnić podobny rodzaj informacji modelowi komputerowemu.

Do inicjalizacji jednej z rekurencyjnych sieci neuronowych wykorzystaliśmy zatem zanurzenia ELMo wytrenowane i udostępnione dla wielu języków, w tym polskiego (Che, Liu et al. 2018).

Koncepcja zanurzeń ELMo, które pojawiły się niedawno i poprawiły najlepsze wyniki w wielu zadaniach przetwarzania języka (Peters, Neumann et al. 2018), jest w ogólnych zarysach podobna do istniejących typów zanurzeń. Podobnie jak zwyczajne zanurzenia wektorowe (Bengio, Ducharme et al. 2003), ELMo powstają jako skutek uboczny wyuczania sieci do zadania przewidywania kolejnych segmentów wypowiedzi (modelowania języka).

Różnica polega jednak na tym, że ELMo powstają z całego zdania (potrafią zatem uwzględniać kontekst) i zawierają informacje z wielu warstw sieci przetwarzającej język, nie tylko z jednej. Z tego powodu w zanurzeniach powinny być ukryte oddzielnie informacje o fonologicznym przetwarzaniu znaków, o morfologii i wreszcie o całych wyrazach. Jako reprezentacja traktowany jest bowiem cały stan odpowiednich warstw sieci, pobierany po przetworzeniu każdego słowa: każde z nich otrzymuje więc swoje własne kontekstowe zanurzenie.

Zanurzenia ELMo powstają z wielowarstwowej, dwukierunkowej sieci LSTM. W naszych eksperymentach początkowy stan ukryty sieci poprawiającej błędy ($H_0$) powstaje przez nieliniowe przekształcenie wektora powstałego jako przekształcenie liniowe (suma ważona) warstw zanurzenia ELMo:

$$H_0 = \text{ReLU}(W \cdot O_{ELMo} + b) \tag{3.8}$$

$O_{ELMo}$ powstaje, jako się rzekło, jako suma ważona warstw sieci ELMo – wagi sumy są wyuczane algorytmicznie dla najskuteczniejszej obsługi danego zadania (w naszym wypadku przewidywania poprawek pisowni):

$$O_{ELMo} = \sum_{j}^{L} s_j h_j, \tag{3.9}$$

gdzie $L$ oznacza liczbę warstw (idąc za cytowaną pracą – trzy), $s$ wagi, a $h$ warstwy. Waga dla warstwy jest wielkością skalarną, mnożoną przez każdą komórkę $h$. Pominęliśmy oddzielny parametr skali $\gamma$ obecny w artykule wprowadzającym ELMo, ponieważ nasza implementacja i tak nie wymusza na $s$ żadnej szczególnej skali, w szczególności zaś sumowania się do jeden.

ReLU oznacza zaś rektyfikowaną funkcję liniową, aplikowalną oddzielnie do każdego elementu wektora bądź macierzy:

$$\mathrm{ReLU}(x) = \max(0, x) \tag{3.10}$$

Sieć działała następnie analogicznie, jak postępowałaby zwyczajna sieć typu RNN.

### 3.3.3. Architektury testowanych sieci

Wszystkie testowane sieci dzieliły tę samą podstawową architekturę. Sieci nazwane na potrzeby tej pracy LSTM-1 i LSTM-2 różnią się jedynie tym, że pierwsza z nich odczytuje błędne słowo od początku do końca, a druga przetwarza je jednocześnie od początku i od końca (i uśrednia rozkłady prawdopodobieństwa przewidujące wynik).

Obie z nich wykorzystują wyuczane zanurzenia znaków w wektorach 50-wymiarowych. Oznacza to, że każdy kolejny znak wejścia (błędnego wyrazu) jest podawany do sieci jako wektor odpowiedni dla tego znaku. Przetwarzają go dwie kolejne warstwy komórek LSTM o 512 wymiarach stanu ukrytego. Odpowiedź sieci uzyskiwana jest przez przekształcenie liniowe odpowiedzi ostatniej warstwy LSTM i uzyskanie logarytmu funkcji softmax:

$$O_i = \log \frac{e^{C_i}}{\sum_j e^{C_i j}} \tag{3.11}$$

Funkcja softmax służy do normalizacji przewidywań sieci i wytworzenia rozkładu prawdopodobieństwa po wszystkich znakach, które sieć może wygenerować jako następne. Logarytm stosujemy ze względu na lepsze własności numeryczne, które ułatwiają wykonanie algorytmu przez komputer.

Po podaniu każdego kolejnego znaku formy błędnej sieć podaje zatem swoje przewidywanie, w postaci rozkładu prawdopodobieństwa, znaku występującego na tej pozycji w formie poprawnej. Te przewidywania są składane w cały wyraz, który uznajemy za odpowiedź sieci. W wypadku sieci dwukierunkowej pozwalamy sieci ukończyć pracę w obie strony, przez co uzyskujemy dla każdej pozycji dwa rozkłady: ten „widziany" czytając od początku i ten od końca. Owe rozkłady są uśredniane, żeby ustalić ostateczną odpowiedź.

Podczas treningu uzyskany rozkład prawdopodobieństwa jest za każdym porównywany z rozkładem uznanym przez nas za prawdziwy, gdzie poprawnemu znakowi na danej pozycji przypisywana jest jedność, a pozostałym możliwościom zero. Funkcja porównująca to entropia krzyżowa (ang. *cross-entropy*) dla rozkładów prawdopodobieństwa $P$ i $Q$:

$$H(P,Q) = -\sum_{x} P(x) \log Q(x) \tag{3.12}$$

Do wstecznej propagacji błędu i modyfikacji parametrów sieci w celu minimalizacji wartości entropii krzyżowej wykorzystywany jest algorytm Adam (Kingma i Ba 2014).

Ostatnia testowana sieć, LSTM-ELMo, różni się od LSTM-1 i LSTM-2 wstępnym odczytaniem całego wyrazu przez sieć ELMo. Jej końcowy stan służy do inicjalizacji pamięci komórek LSTM, tak jak to zostało opisane w poprzednim rozdziale. Również zanurzenia znaków stosowane w głównej sieci pochodzą w modelu LSTM-ELMo z udostępnionej sieci ELMo (Che, Liu et al. 2018) i nie są specjalnie trenowane.

Liczba znaków generowanych przez wszystkie sieci musi być stała, ponieważ nie jest ona powiązana z długością błędnej formy (odpowiednia poprawka może być od niej dłuższa lub krótsza). Sieć ma możliwość wygenerowania w zasadzie dowolnej długości znaków, po których następuje seria „pustych” znaczników wyrównujących ciąg do długości maksymalnej. Tę, ponownie jak w wypadku metody podmiany diakrytyków, ustaliliśmy na 17.

# Rozdział 4

# Eksperymenty

Metody poprawy błędów opisane w Rozdziale 3 poddane zostały jednolitej procedurze testowej. Każda z nich otrzymała zadanie poprawienia tego samego zbioru błędów niewyrazotwórczych. Podejścia zostały ocenione miarą trafności (ang. *accuracy*), czyli odsetka błędów spośród całego zbioru, dla których dany algorytm zaproponował poprawkę zgodną ze wzorcem w korpusie.

## 4.1. Korpus PlEWi

Jako korpus do testów posłużył korpus PlEWi, opublikowany przez Grundkiewicza (Grundkiewicz 2013). Powstał on z kopii polskiej Wikipedii z 2012 roku, gdzie błędy były automatycznie oznaczone przez analizę historii edycji. Większość krótkich fragmentów zmienionych przez kolejnych edytorów Wikipedii stanowi przykłady błędów, które zostały poprawione w kolejnej wersji danego artykułu. Za pomocą różnych heurystyk oddzielono od błędów poszerzenia treści oraz tak zwane wojny edycyjne, a także podzielono same błędy na kategorie.

Z korpusu wydobyliśmy 550 755 błędów (par: forma błędna, forma poprawna) oznaczonych jako `:nonword`. Są to błędy niewyrazotwórcze należące do zakresu zainteresowań tej pracy (twórca korpusu wykorzystał słownik programu Hunspell do wykrycia tych błędów). Przykłady należą do 298 715 unikalnych typów (par), co oznacza, że przeciętny błąd powtarza się w korpusie 1,88 raza. Spośród przykładów 5% posłużyło do celów budowy oprogramowania (ang. *development set*) oraz kontroli poprawności implementacji, 25% do testów, a 70% do treningu modeli.

## 4.2. Realizacja techniczna

Obliczeń odległości edycyjnej dokonała biblioteka PyLucene (Apache Software Foundation 2018), stanowiąca interfejs dla Lucene, biblioteki indeksującej teksty w maszynie wirtualnej Javy. Do eksperymentów z odległością wektorów znaczeniowych posłużyły gotowe wektory wyuczone przez naukowców z Instytutu Podstaw Informatyki PAN (Mykowiecka, Marciniak et al. 2017). Do zaprogramowania sieci neuronowych wykorzystaliśmy pakiet Pytorch (Paszke, Gross et al. 2017), przy czym do obliczania samych zanurzeń ELMo użyliśmy kodu dostarczonego przez badaczy udostępniających odpowiednią sieć (Che, Liu et al. 2018). Pełen kod

| Metoda | Trafność | Zdziwienie | Strata (trening) | Strata (test) |
|---|---|---|---|---|
| Odległość edycyjna | 0,3453 | - | - | - |
| Podmiana diakrytyków | 0,2279 | - | - | - |
| Odległość wektorowa | 0,3945 | - | - | - |
| Sieć LSTM-1 | 0,4183 | **907** | 0,3 | 0,41 |
| Sieć LSTM-2 | 0,6634 | 11182 | 0,1 | **0,37** |
| Sieć LSTM-ELMo | **0,6818** | 706166 | **0,07** | 0,38 |

Tabela 4.1: Wyniki testu dla wszystkich zastosowanych metod. Zastosowane miary objaśnione są w tekście; strata oznacza wartość entropii krzyżowej.

do dokonania eksperymentów jest opublikowany w Internecie.[1]

Eksperymentów dokonaliśmy na maszynie wyposażonej w sześciordzeniowy procesor Intel Xeon E5-1650 12x3,5 GHz, 32 gigabajtów RAM oraz kartę graficzną NVIDIA GeForce GTX 1060 z 6 GB pamięci wewnętrznej. W większości metod sam test na całym fragmencie testowym PlEWi trwał do kilkudziesięciu minut. Wyuczanie sieci LSTM-1 potrwało nieco ponad godzinę, LSTM-2 dwie godziny, LSTM-ELMo zaś około 6 godzin. Wszystkie te modele trenowaliśmy przy pomocy karty grafiki.

## 4.3. Wyniki

Trafność (ang. *accuracy*) dla wszystkich metod została pokazana w Tabeli 4.1. Dla sieci neuronowych podaliśmy także finalne wartości funkcji straty i obliczyliśmy zdziwienie (ang. *perplexity*) korpusem tekstowym. Miara zdziwienia jest tym większa, im niższe prawdopodobieństwo przypisuje model znakom prawdziwego wyrazu. Dokładny wzór, według którego obliczamy tę miarę, to

$$\text{perplexity}(P, x) = 2^{-\frac{1}{N}\sum_{i \leqslant N} \log P(x_i)}, \tag{4.1}$$

gdzie $P$ jest rozkładem prawdopodobieństwa wytwarzanym przez model, a $x$ ciągiem znaków poprawnych słów oraz pustych znaczników wyrównujących słowa do długości maksymalnej. Podajemy średnie zdziwienie dla pojedynczego znaku wyjścia.

Patrząc na trafność, na naszym korpusie podmiana diakrytyków osiągnęła względnie niewielki wynik, czego powodem może być fakt, że autorzy Wikipedii najpewniej zazwyczaj pisali swoje teksty nieco staranniej niż normalnie, przez co pomijanie polskich znaków diakrytycznych z niedbalstwa powinno być rzadsze.

Odległość wektorowa oraz jednokierunkowa sieć LSTM osiągnęły podobny stopień przewagi nad zwykłą odległością edycyjną. Ta druga wypadła jednak nieco lepiej. Wydaje się to pokazywać, że informacje o szczegółach formy wyrazów, wykorzystywane przez sieć rekurencyjną, są bardziej użyteczne, niż rozpoznawanie częstych form błędnych powiązanych z konkretnymi wyrazami (tak jak to robi model wykorzystujący odległość wektorową).

Poważna, ponad pięćdziesięcioprocentowa, przewaga sieci dwukierunkowych jeszcze to podkreśla. Mają one większy wgląd w formy wyrazów, mając perspektywę zarówno od początku,

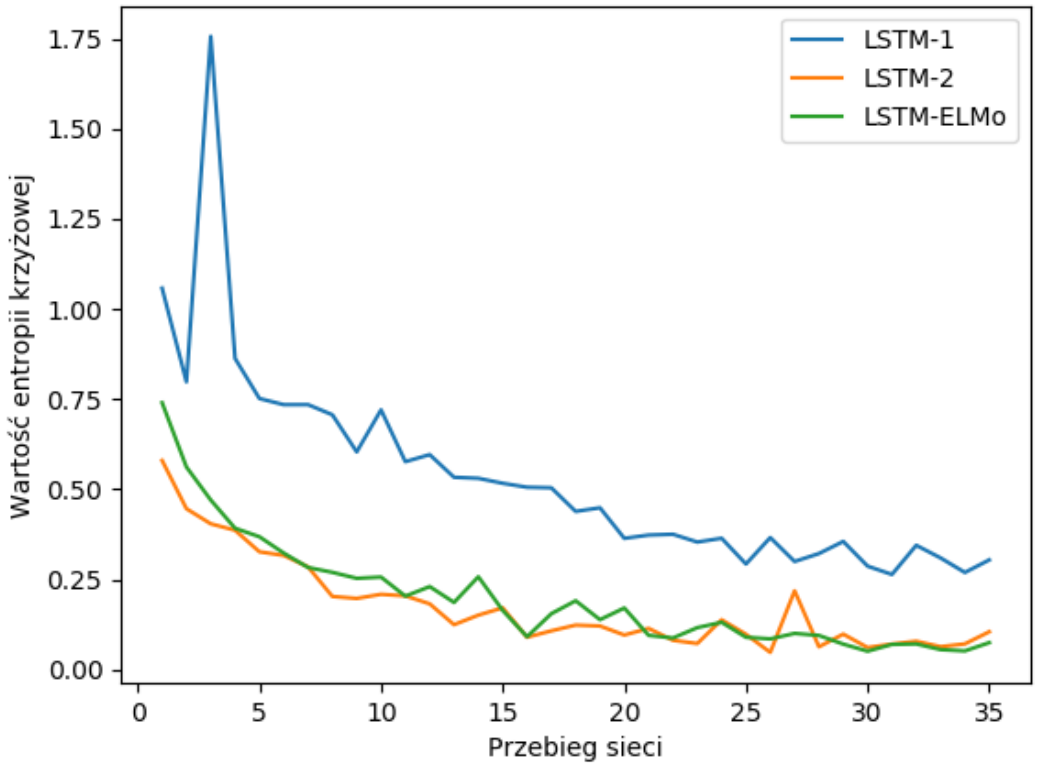
[1] `https://github.com/szmer/cogsci_master`

| I warstwa | II warstwa | III warstwa |
|---|---|---|
| 0,036849 | 0,08134 | 0,039395 |

Tabela 4.2: Wagi przypisane w procesie uczenia poszczególnym warstwom wydobytym z sieci zanurzającej ELMo (nie sumują się do jedności). Warstwy numerowane są od najbliższej do najdalszej wejściu sieci.

jak i od końca. Sieć wsparta wiedzą zawartą w zanurzeniach ELMo zdołała osiągnąć jeszcze niecałe dwa punkty procentowe przewagi nad siecią inicjalizowaną losowym szumem. Jest to ilościowy postęp porównywalny z poprawą wyników osiągniętą przez autorów metody przez dodanie owych zanurzeń do modelów rozwiązujących inne zadania (Peters, Neumann et al. 2018).

Pewien wgląd w sposób wykorzystania zanurzeń przez sieć dają wyuczone wagi przypisane poszczególnym warstwom ELMo (Tabela 4.2). Choć sieci neuronowe znane są z trudności intepretacji ich „wnętrzności", istnieją badania sugerujące istnienie naturalnej struktury przetwarzania języka: modele ciążyłyby ku przetwarzaniu znaków, następnie ich zbitek, w końcu całych wyrazów (por. przegląd w Peters, Neumann et al. (2018)). Nasze wyniki zdają się wskazywać, że najwięcej istotnej informacji dla przewidywania błędów nie leży ani na najniższym poziomie znaków, ani na wysokim poziomie słów i semantyki, lecz najpewniej na poziomie morfemów czy przynajmniej specyficznych zbitek liter. Byłaby to wiadomość istotna nie tylko dla neuronowych modeli korekty pisowni.




Rysunek 4.1: Historia entropii krzyżowej w czasie treningu sieci.

Można też dostrzec tendencję wzrostu miary zdziwienia w coraz bardziej złożonych modelach. Tłumaczymy to bardziej równomiernym („łagodnym") rozkładem prawdopodobieństwa, jaki wytwarza nauka tych algorytmów. Co za tym idzie, mogą one częściej wybierać właściwą odpowiedź, ale jednocześnie przypisywać jej względnie mniejszą przewagę w prawdopodobieństwie w stosunku do innych możliwości, niż się to dzieje w prostszym modelu.

Na komentarz zasługują również tendencje dotyczące miary entropii krzyżowej widoczne w Tabeli 4.1 oraz na Rysunku 4.1. Wartość funkcji straty w wypadku sieci LSTM-ELMo rozwijała się podobnie jak w wypadku LSTM-2, osiągając na końcu niższy poziom na korpusie treningowym i wyższy na korpusie testowym. Takie zjawiska każą sądzić, że dochodziło w jakiejś mierze do przeuczenia (ang. *overfitting*) algorytmu, przynajmniej w tym sensie, że uciekał się do używania zanurzeń ELMo do uczenia się słów na pamięć, zamiast korzystać z głębszych prawidłowości. Nie jest wykluczone, że w jakiejś mierze jest to rozwiązanie usprawiedliwione dla słów posiadających specyficzne dla siebie błędy. Możliwe też, że jest to problem, któremu mógłby zaradzić jeszcze większy korpus, pozwalający pokazać sieci więcej zanurzeń znaczeniowych i umożliwić odnalezienie drogi do ich bardziej inteligentnego wykorzystania.

# Rozdział 5

# Podsumowanie

W niniejszej pracy został przetestowany szereg podejść do zadania izolowanego poprawiania błędów niewyrazotwórczych w języku polskim. Pomimo ograniczonego zakresu problemu leży on w centrum szerszej kwestii optymalnej korekcji pisowni dla polszczyzny. Jest ona potrzebna w wielu zastosowaniach, takich jak przygotowywanie tekstów, gromadzenie baz danych, odczytywanie dokumentów za pomocą technologii OCR czy rozpoznawanie mowy.

W szerszym sensie bez odpowiedniej normalizacji pisowni większość systemów przetwarzania języka naturalnego nie może działać poprawnie. W każdej z tych sytuacji poprawa wyrazów nieodnalezionych w słowniku może być rozwiązaniem najprostszym czy przynajmniej awaryjnym, kiedy bardziej skomplikowany model (na przykład rozpoznający kontekst) nie może sobie poradzić.

Na potrzeby pracy dokonaliśmy przeglądu literatury przedmiotu i selekcji najbardziej obiecujących podejść do postawionego problemu. Wytworzyliśmy sformułowania kilku różnych algorytmów do poprawiania tekstów w języku polskim – dbając, by nasze rozwiązania czyniły użytek z najnowszych osiągnięć w dziedzinie komputerowego przetwarzania języka naturalnego.

Algorytmy zostały zaimplementowane razem z zestawem powtarzalnych testów, które mogły posłużyć do oceny relatywnej skuteczności poszczególnych koncepcji. Efekty owych eksperymentów przedstawił rozdział 4.

Z dokonanych doświadczeń należy wyciągnąć wniosek, że większość rozwiązań przynosi niewielkie, chociaż zauważalne postępy w porównaniu z odległością edycyjną. Może to zachęcać do stosowania w praktyce systemów typu *ensemble* (zespołowych), gdzie uzgadniane są propozycje wielu oddzielnych modeli. Należy jednak zwrócić uwagę, że każde z podejść niesie ze sobą własne obciążenie obliczeniowe. Rozwiązania wykorzystujące w jakiejś formie odległość edycyjną czy inne przeszukiwanie przestrzeni odmian wyjściowego wyrazu (jak np. podmiana diakrytyków) wymagają zastosowania odpowiednich algorytmów w implementacji, czy też użycia bibliotek zoptymalizowanych do działania na dużą skalę, takich jak Lucene. Sieci neuronowe wymagają przede wszystkim przechowywania wag w pamięci, pomijając nawet obliczenia konieczne przy ich każdym ich przebiegu.

Bardzo wyraźna poprawa wyników wniesiona przed dwukierunkowe sieci LSTM pozwala liczyć na dalszy postęp dzięki wprowadzeniu najnowszych osiągnięć uczenia głębokiego (ang. *deep learning*) do zadania korekty polszczyzny. Zapewne dalsze podniesienie trafności mogłoby być osiągnięte dzięki dodaniu mechanizmu uwagi (*attention*), czy też sztucznego wytwarzania

dodatkowych przykładów błędów w układzie typu GAN (*Generative Adversarial Network*). Jest to jeden z obiecujących kierunków dalszych badań na tym polu.

Pożyteczne mogłoby być wydanie korpusu z bezkontekstową anotacją błędów dokonaną przez ludzi, tak by można było porównywać wyniki maszynowe z ich trafnością. Dochodzi do tego postulat wytworzenia większych korpusów w ogóle, a także (z drugiej strony) opracowywania rozwiązań pozwalających na wykorzystanie do nauki zasobów o uboższej anotacji. Przykładem takiego korpusu byłby WikEd, stanowiący ideową kontynuację PlEwi dla wielu języków, pozbawioną jednak m.in. szczegółowej informacji o rodzaju błędów (Grundkiewicz i Junczys-Dowmunt 2014).

Oczywiste wydaje się też rozszerzenie zakresu samego zadania – wykorzystując podstawowe wyniki uzyskane w tej pracy oraz szereg raportów z zastosowań praktycznych (Mykowiecka i Marciniak 2007; Ogrodniczuk i Kopeć 2017) – tak, by obejmowało ono wszelkie błędy w pisowni, poprawiane z wykorzystaniem kontekstu, czy nawet, idąc za literaturą światową, również błędy gramatyczne i przykłady nieporadności językowej.

# Bibliografia


Ahmed, Farag i Luca, Ernesto William De [et al.] (2009): Revised n-gram based automatic spelling correction tool to improve retrieval effectiveness. – *Polibits*, 39–48.

Ahn, Natalie (2017): Rule-Based Spanish Morphological Analyzer Built From Spell Checking Lexicon. – *CoRR* abs/1707.07331.

Angell, Richard C. i Freund, George E. [et al.] (1983): Automatic spelling correction using a trigram similarity measure. – *Information Processing & Management* 19, 255–261.

Apache Software Foundation (2018): PyLucene. Dostępne pod adresem: https://lucene.apache.org/pylucene/.

Bassil, Y. i Alwani, M. (2012): Context-sensitive Spelling Correction Using Google Web 1T 5-Gram Information. – *ArXiv e-prints*.

Bengio, Yoshua i Ducharme, Réjean [et al.] (2003): A Neural Probabilistic Language Model. – *JOURNAL OF MACHINE LEARNING RESEARCH* 3, 1137–1155.

Bušta, Jan i Hlavácková-Schindler, Dana [et al.] (2009): Classification of Errors in Text. – (w:) *RASLAN*.

Che, Wanxiang i Liu, Yijia [et al.] (2018): Towards Better UD Parsing: Deep Contextualized Word Embeddings, Ensemble, and Treebank Concatenation. – *CoRR* abs/1807.03121.

Chollampatt, Shamil i Taghipour, Kaveh [et al.] (2016): Neural Network Translation Models for Grammatical Error Correction. – *CoRR* abs/1606.00189.

Damerau, Fred J. (1964): A Technique for Computer Detection and Correction of Spelling Errors. – *Commun. ACM* 7, 171–176.

Deerwester, Scott i Dumais, Susan T. [et al.] (1990): Indexing by latent semantic analysis. – *Journal of the American Society for Information Science* 41, 391–407.

Dorosz, Krzysztof (2008): Automatyczne sprawdzanie poprawności pisowni w języku polskim oparte na odległości Levenshteina. – *Automatyka/Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie* 12, 29–40.

Dronen, Nicholas A. (2016): Correcting Writing Errors with Convolutional Neural Networks. – 200 University of Colorado at Boulder.

Fahda, A. i Purwarianti, A. (2017): A statistical and rule-based spelling and grammar checker for Indonesian text. – (w:) *2017 International Conference on Data and Software Engineering (ICoDSE)*. 1–6.

Gadamer, Marcin i Horzyk, Adrian (2009): Automatyczna kontekstowa korekta tekstów z wykorzystaniem grafu LHG. – *Computer Science* Vol. 10, 37–55.

Ge, Tao i Wei, Furu [et al.] (2018): Reaching Human-level Performance in Automatic Grammatical Error Correction: An Empirical Study. – *CoRR* abs/1807.01270.

Golding, Andrew R. (1995): A Bayesian hybrid method for context-sensitive spelling correction. – (w:) *In Proceedings of the Third Workshop on Very Large Corpora*. 39–53.

Grundkiewicz, Roman (2013): Automatic Extraction of Polish Language Errors from Text Edition History. – (w:) Ivan Habernal i Václav Matoušek (red.): *Text, Speech, and Dialogue*. – Berlin, Heidelberg: Springer Berlin Heidelberg. 129–136.

Grundkiewicz, Roman i Junczys-Dowmunt, Marcin (2014): The WikEd Error Corpus: A Corpus of Corrective Wikipedia Edits and Its Application to Grammatical Error Correction. – (w:) *PolTAL*.

Hanson, A.R. i Riseman, E.M. [et al.] (1976): Context in word recognition. – *Pattern Recognition* 8, 35–45.

Hassan, Ahmed i Noeman, Sara [et al.] (2008): Language Independent Text Correction using Finite State Automata. – (w:) *Proceedings of the Third International Joint Conference on Natural Language Processing: Volume-II*. Dostępne pod adresem: http://www.aclweb.org/anthology/I08-2131.

Hochreiter, Sepp i Schmidhuber, Jürgen (1997): Long Short-Term Memory. – *Neural Comput.* 9, 1735–1780.

Junczys-Dowmunt, Marcin i Grundkiewicz, Roman [et al.] (2018): Approaching Neural Grammatical Error Correction as a Low-Resource Machine Translation Task. – *CoRR* abs/1804.05940.

Kingma, Diederik P. i Ba, Jimmy (2014): Adam: A Method for Stochastic Optimization. – *CoRR* abs/1412.6980.

Kukich, Karen (1988): Variations on a back-propagation name recognition net. – (w:) *Proceedings of the Advanced Technology Conference*. 722–735. (= 2).

Kukich, Karen (1992): Techniques for automatically correcting words in text. – *Acm Computing Surveys (CSUR)* 24, 377–439.

Levenshtein, VI (1966): Binary Codes Capable of Correcting Deletions, Insertions and Reversals. – *Soviet Physics Doklady* 10, 707.

Mangu, Lidia i Brill, Eric (1997): Automatic Rule Acquisition for Spelling Correction. – (w:) *Proceedings of the Fourteenth International Conference on Machine Learning*. – San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. 187–194. (= ICML '97). Dostępne pod adresem: http://dl.acm.org/citation.cfm?id=645526.657126.

Mikolov, Tomas i Sutskever, Ilya [et al.] (2013): Distributed Representations of Words and Phrases and their Compositionality. – (w:) C. J. C. Burges, L. Bottou et al. (red.): *Advances in Neural Information Processing Systems 26*. – Curran Associates, Inc., 3111–3119.

Miłkowski, Marcin (2010): Developing an open-source, rule-based proofreading tool. – *Software: Practice and Experience* 40, 543–566.

Mnih, Andriy i Hinton, Geoffrey E (2009): A Scalable Hierarchical Distributed Language Model. – (w:) D. Koller, D. Schuurmans et al. (red.): *Advances in Neural Information Processing Systems 21.* – Curran Associates, Inc., 1081–1088.

Mykowiecka, Agnieszka i Marciniak, Małgorzata (2007): Domain–Driven Automatic Spelling Correction for Mammography Reports. 521–530.

Mykowiecka, Agnieszka i Marciniak, Małgorzata [et al.] (2017): Testing Word Embeddings for Polish. – *Cognitive Studies / Études Cognitives* 17, 1–19.

Nagata, Ryo i Takamura, Hiroya [et al.] (2017): Adaptive Spelling Error Correction Models for Learner English. – *Procedia Computer Science* 112, 474–483.

Nallapati, Ramesh i Xiang, Bing [et al.] (2016): Sequence-to-Sequence RNNs for Text Summarization. – *CoRR* abs/1602.06023.

Ng, Hwee Tou i Wu, Siew Mei [et al.] (2014): The CoNLL-2014 Shared Task on Grammatical Error Correction. – (w:) *CoNLL Shared Task*.

Ogrodniczuk, Maciej i Kopeć, Mateusz (2017): Lexical Correction of Polish Twitter Political Data. – (w:) *Proceedings of the Joint SIGHUM Workshop on Computational Linguistics for Cultural Heritage, Social Sciences, Humanities and Literature.* – Vancouver, Canada: Association for Computational Linguistics. 115–125. Dostępne pod adresem: http://www.aclweb.org/anthology/W17-2215.

Paszke, Adam i Gross, Sam [et al.] (2017): Automatic differentiation in PyTorch.

Peters, Matthew E. i Neumann, Mark [et al.] (2018): Deep contextualized word representations. – *CoRR* abs/1802.05365.

Philips, Lawrence (2000): The Double Metaphone Search Algorithm. – *C/C++ Users J.* 18, 38–43.

Pirinen, Tommi A. i Lindén, Krister (2014): State-of-the-Art in Weighted Finite-State Spell-Checking. – (w:) *Proceedings of the 15th International Conference on Computational Linguistics and Intelligent Text Processing - Volume 8404.* – Berlin, Heidelberg: Springer-Verlag. 519–532. (= CICLing 2014). Dostępne pod adresem: https://doi.org/10.1007/978-3-642-54903-8_43.

Plank, Barbara i Søgaard, Anders [et al.] (2016): Multilingual Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Models and Auxiliary Loss. – *CoRR* abs/1604.05529.

Řehůřek, Radim i Sojka, Petr (2010): Software Framework for Topic Modelling with Large Corpora. – (w:) *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks.* – Valletta, Malta: ELRA. 45–50.

Sakaguchi, Keisuke i Duh, Kevin [et al.] (2017): Robsut Wrod Reocginiton via Semi-Character Recurrent Neural Network. – (w:) *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.* – AAAI Press. 3281–3287. Dostępne pod adresem: http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14332.

Saloni, Zygmunt i Woliński, Marcin [et al.] (2018): Słownik gramatyczny języka polskiego — wersja online. Dostępne pod adresem: http://sgjp.pl.

Saussure, Ferdinand de (1991): *Kurs językoznawstwa ogólnego.* – Warszawa: Państwowe Wydawnictwa Naukowe.

Schmaltz, Allen i Kim, Yoon [et al.] (2017): Adapting Sequence Models for Sentence Correction. – (w:) *EMNLP*.

Schuster, M. i Paliwal, K.K. (1997): Bidirectional Recurrent Neural Networks. – *Trans. Sig. Proc.* 45, 2673–2681.

Subieta, Kazimierz (1976): Korekcja pojedynczych błędów w wyrazach na podstawie słownika wzorców. – *Informatyka* 11, 15–18.

Subieta, Kazimierz (1985): A simple method of data correction. – *The Computer Journal* 28, 372–374.

Trón, Viktor i Kornai, András [et al.] (2005): Hunmorph: Open Source Word Analysis. – (w:) *Proceedings of the Workshop on Software*. – Stroudsburg, PA, USA: Association for Computational Linguistics. 77–85. (= oprogramowanie '05). Dostępne pod adresem: http://dl.acm.org/citation.cfm?id=1626315.1626321.

Yuan, Zheng i Felice, Mariano (2013): Constrained Grammatical Error Correction using Statistical Machine Translation. – (w:) *CoNLL Shared Task*.