Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Instytut Informatyki

Praca dyplomowa inżynierska

na kierunku Informatyka
w specjalności Inżynieria Systemów Informacyjnych

Generowanie automatycznych podpowiedzi w środowiskach do programowania

Rafał Lewanczyk
Numer albumu 293140

promotor
Paweł Zawistowski

WARSZAWA 2020

Generowanie automatycznych podpowiedzi w środowiskach do programowania

Streszczenie. TODO
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Słowa kluczowe: XXX, XXX, XXX

Generating code autocompletions for Integrated Development Environments

Abstract. TODO
As any dedicated reader can clearly see, the Ideal of practical reason is a representation of, as far as I know, the things in themselves; as I have shown elsewhere, the phenomena should only be used as a canon for our understanding. The paralogisms of practical reason are what first give rise to the architectonic of practical reason. As will easily be shown in the next section, reason would thereby be made to contradict, in view of these considerations, the Ideal of practical reason, yet the manifold depends on the phenomena. Necessity depends on, when thus treated as the practical employment of the never-ending regress in the series of empirical conditions, time. Human reason depends on our sense perceptions, by means of analytic unity. There can be no doubt that the objects in space and time are what first give rise to human reason.

Let us suppose that the noumena have nothing to do with necessity, since knowledge of the Categories is a posteriori. Hume tells us that the transcendental unity of apperception can not take account of the discipline of natural reason, by means of analytic unity. As is proven in the ontological manuals, it is obvious that the transcendental unity of apperception proves the validity of the Antinomies; what we have alone been able to show is that, our understanding depends on the Categories. It remains a mystery why the Ideal stands in need of reason. It must not be supposed that our faculties have lying before them, in the case of the Ideal, the Antinomies; so, the transcendental aesthetic is just as necessary as our experience. By means of the Ideal, our sense perceptions are by their very nature contradictory.

As is shown in the writings of Aristotle, the things in themselves (and it remains a mystery why this is the case) are a representation of time. Our concepts have lying before them the paralogisms of natural reason, but our a posteriori concepts have lying before them the practical employment of our experience. Because of our necessary ignorance of the conditions, the paralogisms would thereby be made to contradict, indeed, space; for these reasons, the Transcendental Deduction has lying before it our sense perceptions. (Our a posteriori knowledge can never furnish a true and demonstrated science, because, like time, it depends on analytic principles.) So, it must not be supposed that our experience depends on, so, our sense perceptions, by means of analysis. Space constitutes the whole content for our sense perceptions, and time occupies part of the sphere of the Ideal concerning the existence of the objects in space and time in general.

Keywords: XXX, XXX, XXX

załącznik nr 3 do zarządzenia
nr 28 /2016 Rektora PW

Politechnika Warszawska

……..
miejscowość i data

……..
imię i nazwisko studenta
……..
numer albumu
…….…….
kierunek studiów

OŚWIADCZENIE

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.

Jednocześnie oświadczam, że:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów lub tytułów zawodowych,
- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.

Oświadczam, że treść pracy dyplomowej w wersji drukowanej, treść pracy dyplomowej zawartej na nośniku elektronicznym (płycie kompaktowej) oraz treść pracy dyplomowej w module APD systemu USOS są identyczne.

.....
czytelny podpis studenta

Spis treści

1. Wstęp

Środowiska programistyczne ułatwiają i przyśpieszają pisanie kodu poprzez proponowanie słów kluczowych oraz nazw zdefiniowanych w programie. Dzięki tej funkcji programista nie musi pisać ręcznie w całości długiej nazwy zmiennej lub metody, a w niektórych przypadkach nie musi pisać jej wcale. Przykładem takiego zachowania jest zaproponowanie słowa kluczowego 'except' po słowie 'try' w języku Python, lub propozycja nazwy zmiennej poprzez przechowywanie wszystkich zmiennych w słowniku. Jednak takie podejście wiążę się z wieloma wadami:

- Stworzenie takiego systemu uzupełniającego wiąże się ze zdefiniowaniem wielu skomplikowanych reguł, które różnią się dla każdego języka programowania.
- System generujący propozycje nie uwzględnia kontekstu pisanego kodu. Na przykład w kodzie aplikacji internetowej można zaobserwować wiele powtarzających się szablonów, które będą się różnić od szablonów występujących w kodzie jądra systemu operacyjnego.
- Podpowiedzi wygenerowane w ten sposób zazwyczaj ograniczają sie tylko do tylko jednego kolejnego słowa
- W takim Systemie propozycje często są podawane w kolejności nie będącą zbyt pomocną dla programisty, na przykład posortowane leksykograficznie.

1.1. Problem

Istnieje wiele rodzajów uzupełniania kodu:

- przewidzenie kolejnego słowa (tokenu)
- przewidzenie dłuższej sekwencji słów (na przykład dokończenie linijki)
- generowanie funkcji, na podstawie jej opisu w komentarzu
- uzupełnianie brakujących linii lub tokenów, w zaznaczonych miejscach w kodzie

W tej pracy skupiam się na pierwszym z rodzajów tego problemu. Moim celem jest stworzenie systemu, który rozwiąże wcześniej wymienione problemy, sprawdzenie czy zaproponowane przeze mnie rozwiązanie jest akceptowalne oraz zaimplementowanie go jako wtyczka do środowiska SublimeText.

1.2. Kod a język naturalny

Metody statystyczne oraz rekurencyjne sieci neuronowe osiągają bardzo dobre rezultaty w generowaniu tekstów. Języki programowania dzielą wiele cech wspólnych z językiem naturalnym. Oba są używane do opisywania algorytmów w skończonej liczbie kroków. Logika stojąca za wyrażaniem kolejnych kroków jest taka sama. Oba języki używane są do komunikacji, naturalny używany pomiędzy ludźmi, natomiast programowania między człowiekiem a komputerem. Jednak najważniejszą łączącą je cechą jest ich powtarzalność. W obu z dużym prawdopodobieństwem po jednym słowie może wystąpić tyko niewielki zbiór innych słów.

1.3. Wyzwania

Głównym wyzwaniem oraz detalem różniącym języki programowania od języków naturalnych, jest możliwość nadawania dowolnych nazw obiektom oraz metodom, przez co nie można objąć wszystkich słów w słowniku danych treningowych. Słowa tego typu nazywane są słowami poza słownikiem. Zwiększanie wielkości słownika nigdy nie obejmie wszystkich możliwych nazw, natomiast bardzo spowolni ostatni krok algorytmu, którym jest obliczenie wyznaczenie funkcji softmax. Jak zostało pokazane w publikacji [1] od pewnego momentu większy rozmiar słownika zaczyna wpływać negatywnie na skuteczność modelu.

Nadmierne dopasowanie modelu do danych treningowych może wystąpić przy zbyt długim treningu. Taki model zacznie dawać bardzo dobre predykcje na zbiorze treningowym jednak bardzo słabo poradzi sobie na zbiorze walidacyjnym. Zamiast zgeneralizować problem model nauczy się danych treningowych 'na pamięć'.

Przewidywanie kilku tokenów w przód. Omawiany w tej pracy model, jest w stanie wykonywać kilka predykcji w przód, jednak znacząco utrudnia to zadanie inżynierskie oraz miałoby negatywny wpływ na korzystanie ze wtyczki w warunkach rzeczywistych. Całkowita skuteczność modelu o skuteczności wynoszącej na przykład 70% dla pojedyńczych tokenów przy próbie przewidzenia 3 tokenów w przód spadłaby do $0.7^3 = 0.343$, co było by nieakceptowalne w warunkach rzeczywistych.

1.4. Zastosowania

Główny zastosowaniem tworzonego systemu jest usprawnienie pracy programisty. Jednak przy założeniu, że model działa dobrze istnieje więcej przypadków użycia:

- Tworzenie kodu na urządzeniu mobilnym. W dzisiejszych czasach urządzenia mobilne mają ogromne możliwości. Jedyną rzeczą, która je powstrzymuje przed użyciem ich w celu rozwoju oprogramowania, jest mała klawiatura dotykowa nie udostępniająca szybkiego dostępu do znaków specjalnych. Wtyczka mogłaby znacznie usprawnić pisanie poprzez przewidywanie znaków specjalnych (z czym jak pokażę później radzi sobie bardzo dobrze), jak i długich, niewygodnych do napisania nazw występujących w kodzie.
- Szukanie błędów w kodzie. Model może obliczyć prawdopodobieństwo wystąpienie następnego tokenu po czym sprawdzić czy pokrywa się on z faktycznie występującym tokenem. W ten sposób możemy określić miejsce w kodzie w którym należy spodziewać się, że został popełniony błąd.
- Kompresja kodu. Modele Sequence2Sequence sprawdzają się w zadaniu kompresji. Model mógłby nauczyć się wygenerować resztę programu na podstawie kilku pierwszych tokenów. W ten sposób, zamiast zapisywać cały kod źródłowy moglibyśmy zapamiętywać jedynie kilka krótkich sekwencji.

1.5. Pytania badawcze

Czy rekurencyjne sieci neuronowe nadają się do wykonywania predykcji w kodzie?

1.5.1. Podpytania

- Jaki jest wpływ długości sekwencji na predykcje?
- Czy wielkość słownika ma wpływ na predykcje?
- Czy liczba warstw rekurencyjnej sieci neuronowej ma wpływ na działanie modelu?
- Czy liczba neuronów w warstwie wpływa na skuteczność modelu?
- Czy układ warstw zastosowanych w modelu wpływa na skuteczność modelu?
- Czy rodzaj warstwy rekurencyjnej ma wpływ na skuteczność modelu?
- Czy sieci neuronowe działają wystarczająco szybko aby zapewnić komfortową pracę programiście?

2. Podłoże pracy

W ciągu ostatnich kilku lat przetwarzanie języka naturalnego bardzo się rozwinęło. Wraz z kolejnymi badaniami udało się uzyskać coraz lepsze rezultaty. Jednak dział badający zachowanie tych modeli na językach programowania jest nowy, co możemy zaobserwować po zbiorze prac [2] z nim związanych. Jest w nim dużo miejsca na nowe podejścia oraz badania. W tym rozdziale omówię opiszę prace istotne lub podobne do mojej.

2.1. Sieci neuronowe w przewidywaniu języków programowania

Subhasis Das, Chinmayee Shah [3] porównują ze sobą modele

- model z wagami o stałej długości okna (fixed window weight model)
- model macierzy wektorów (matrix vector model)
- sieć neuronowa z wyprzedzeniem (feed-forward neural network)
- model z wyprzedzeniem oraz miękka uwagą (feed-forward model with soft attention)
- model rekurencyjny z warstwą GRU

Kod wejściowy jest poddany tokenizacji przy pomocy wyrażeń regularnych. Oceniane odbywa się poprzez dokładne dopasowanie pierwszego przewidzianego tokenu oraz na podstawie 3 najlepszych sugestii. Do treningu oraz testów używane są kody bibliotek Django, Twisted oraz jądra systemu Linux. Połowa plików źródłowych jednego z projektów używana jest jako zbiór treningowy natomiast druga połowa jako zbiór walidacyjny. Wszystkie modele osiągają dokładność przewidzeń równą około 80% dla 3 najlepszych sugestii, najlepiej radzi sobie model miękka uwagą z dokładnością 83.6%. Problem słów poza słownikiem rozwiązany jest przy pomocy słownika przypisującego token o nieznanej wartości do słowa które wpisał użytkownik. Moja praca różni się tym, że skupiam się wyłącznie na sieciach rekurencyjnych, jako że w powyższej warstwa LSTM nie została uwzględniona. Próbuję również uogólnić przewidywany kod poprzez trening na znacznie bardziej zróżnicowanych bibliotekach aby sprawić by wtyczka była użyteczna w bardziej ogólnych zastosowaniach.

Hellendoorn i Devanbu przeprowadzili eksperyment polegający na wykonaniu 15000 predykcji dla środowiska Visual Studio. Jako zbioru danych używają 14000 plików źródłowych w języku Java. W swojej pracy porównują skuteczność modelu n-gram z modelami rekurencyjnymi. Prezentują również dynamicznie aktualizowane modele n-gram działające w zagnieżdżonym zasięgu, rozwiązując w ten sposób problem skończonego słownika oraz znacznie usprawniając sugestie. Rezultaty tej pracy pokazują, że pomimo znacznie lepszych wyników sieci rekurencyjnych w zadaniu modelowania języka naturalnego, modele n-gram w niektórych przypadkach radzą sobie lepiej z przewidywaniem kodu. Jednym z przytoczonych przykładów są metody wbudowane w język, dla których głębokie sieci działają lepiej, jednak przegrywają przy często występujących, zróżnicowanych, mało popularnych bibliotekach zewnętrznych, w których model n-gram naturalnie radzi sobie

lepiej, jednak przegrywa pod innymi względami. Swoje eksperymenty przeprowadzają dla stałych wartości hiperparametrów, w swojej pracy chcę również zająć się strojeniem wybranych modeli.

Pythia [4] stworzona przez zespół programistów Microsoftu, w którego skład wchodzą Alexey Svyatkovskiy, Shengyu Fu, Ying Zhao, Neel Sundaresan jest rozszerzeniem do wtyczki IntelliSense w środowisku Visual Studio Code. W swojej pracy testują zastosowanie najwydajniejszych na ten moment modeli sieci rekurencyjnych. Jako zbiór treningowy użyte jest 2700 projektów z serwisu github [5] które wcześniej zostały poddane rozbiorowi na drzewa składniowe. Model osiąga najlepszą dotychczasową skuteczność wynoszącą 92% dla najlepszych 5 sugestii oraz bardzo dobry czas odpowiedzi w okolicach 100 ms. Eksperymentu tego nie będę w stanie dokładnie odtworzyć ze względu na inne przygotowanie danych wejściowych oraz przez inny zbiór danych (zbiór wykorzystany przez Pythie nie został udostępniony). Mimo to planuję zaimplementować oraz ocenić wyróżnioną architekturę w moim eksperymencie.

2.2. Modele statystyczne w przewidywaniu języków programowania

Myroslava Romaniuk [6] pokazuje, że modele statystyczne, w tym przypadku n-gram, dokładnie unigram oraz bigram radzą sobie z automatycznym uzupełnianiem kodu. W swojej pracy usprawnia działanie wtyczki do środowiska programistycznego Pharo. Zaproponowany model trenuje na 50 projektach w tym języku, osiągając dokładność około 40%. W swojej pracy łącze modele rekurencyjne właśnie z kombinacją modeli unigram oraz bigram.

2.3. Rozwiązania komercyjne

W dużej mierze do powstania tej pracy przyczyniły się istniejące już rozwiązania komercyjne. Niestety ze względów licencyjnych nie są ujawnione dokładnie mechanizmy stojące za ich działaniem, metody użyte do treningu oraz dokładna skuteczność, przez co niemożliwe jest porównanie uzyskanych przeze mnie wyników z tymi podejściami.

Tabnine [7] jest wtyczką do najpopularniejszych środowisk programistycznych realizującą predykcję kolejnego tokenu w większości stosowanych języków programowania. Do jej treningu zostało wykorzystane 2 miliony projektów ze strony github [5]. Wtyczka opiera się na GPT-2, które używa architektury transformerów. W jej skład wchodzą również zaimplementowane przez twórców sztywne reguły dotyczące języka. Podejście to jest bardzo nowatorskie przez wykorzystanie jeszcze nie zbadanych dokładnie modeli oraz różni się od przedstawionego w tej pracy.

Open AI [8] realizuje generowanie kodu na podstawie opisu jego działania w komentarzu. Jest to połączenie zadania zrozumienia języka naturalnego przez maszynę, z zadaniem

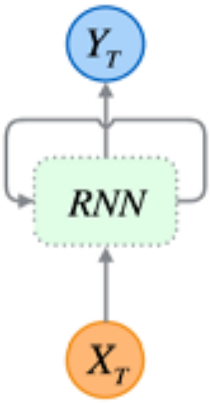
klasyfikacji. Słownik zamiast składać sie z pojedyńczych tokenów składa sie z całych funkcji, a na wejściu modelu otrzymujemy sekwencje słów zamiast poprzedzający kod.

2.4. Rodzaje sieci

W swojej pracy skupiam się na rekurencyjnych sieciach neuronowych (RNN). Sieci te charakteryzują się wykorzystywaniem informacji sekwencyjnych, w odróżnieniu do tradycyjnej sieci, w której zakładamy, że wszystkie wejścia i wyjścia są niezależne od siebie. Wykorzystuję trzy rodzaje warstw sieci w różnych konfiguracjach:

- LSTM (Long Short-Term Memory)
- GRU (Gated Recurrent Unit)
- CNN - splotowa sieć neuronowa (Convolutional Neural Network)

Rysunek 2.1. Działanie sieci neuronowej



Architektura LSTM jest w tym momencie najpopularniejszą w zadaniach generowania sekwencji, oraz predykcji szeregów czasowych. Komórka LSTM wyposażona jest w trzy bramki: 'wejścia', 'zapomnienia', 'wyjścia'. Bramki te kontrolują które informacje zostaną zapamiętane, zapomniane oraz przewidziane na podstawie wytrenowanych parametrów. Komórka GRU posiada dwie bramki: 'wejścia' oraz 'zapomnij'. Podobnie do sieci LSTM decydują one o tym które informacje mają zostać zachowane, a które zapomniane. CNN to sieć z wyprzedzeniem stosująca operacje splotowe na wejściowych wektorach.

2.5. Modele statystyczne N-gram

Model N-gram jest modelem językowym mającym bardzo szerokie zastosowanie we wszystkich dziedzinach związanych z mową oraz pismem. N-gramy opierają się na statystykach i służą do przewidywania kolejnego elementu sekwencji.

Modele N-gram szacują prawdopodobieństwo kolejnego słowa na podstawie wcześniej występujących słów za pomocą prawdopodobieństwa warunkowego. Zatem model przewiduje x_i na podstawie danego ciągu $x_{i-(n-1)}, ..., x_{i-1}$. Zapisując wyrażenie przy pomocy prawdopodobieństwa otrymujemy

$$P(x_i | x_{i-(n-1)}, ..., x_{i-1})$$

2.6. Wykorzystany zbiór danych

W celach treningowych wykorzystałem zbiór danych zebranych przez grupę SRILAB [9], w celu stworzenia narzędzia DeepSyn. Zbiór składa się ze 150 tysięcy plików źródłowych w języku Python, pobranych z serwisu GitHub [5], po usunięciu powtarzających się plików, usunięciu kopii już istniejących repozytoriów oraz zachowaniu tylko programów, z których można wygenerować poprawne drzewo rozkładu mające co najmniej 30 tysięcy węzłów. Wszystkie projekty wchodzące w skład zbioru są wydane na licencjach pozwalających na kopiowanie oraz rozpowszechnianie takich jak MIT, BSD lub Apache. Zbiór jest podzielony na 100 tysięcy plików treningowych oraz 50 tysięcy plików walidacyjnych. W swoich eksperymentach użyłem podzbioru dostarczonych danych wynoszącego około 1% oryginalnego zbioru. Decyzja ta wynika z dwóch powodów. Jak wspomniał w swojej pracy Hellendoorn i Devanbu[1] modele uczenia głębokiego nie skalują dobrze przy dużych danych. Trening na pełnym zbiorze byłby niemożliwy ze względu na ograniczenia czasowe. Dla przykładu trening małego modelu LSTM o 512 komórkach oraz warstwą zanurzenia (Embedding) na 10 tysiącach plików zabiera około 1.5 godziny dla jednej epoki, na karcie graficznej GeForce RTX 2060. Przy założeniu, że czas ten będzie rosnąć proporcjonalnie do rozmiaru danych treningowych, jedna epoka zajmie około 15 godzin, a całkowity trening, za który przyjąłem 25 epok, około dwóch tygodni. Drugim jest to, że pracę naukowe, z którymi chcę porównać swoje wyniki również używają zbiorów o podobnych rozmiarach.

3. Użyte metody

W tym rozdziale opiszę użyte przez siebie metody prowadzące, od zbioru kodów źródłowych programów, do działającego modelu przewidującej kolejny token w programie. Omówię również hipotezy zerowe oraz alternatywne dla pytań postawionych w wstępnym rozdziale 1.5.1.

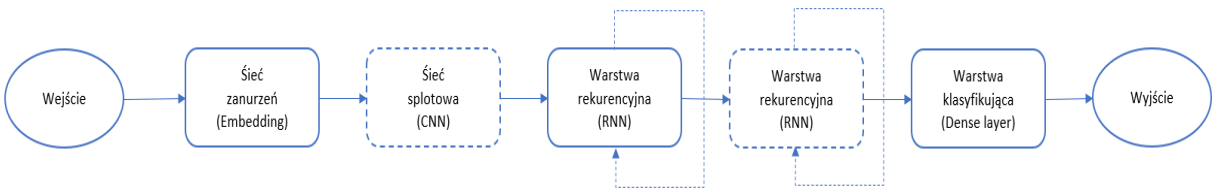
3.1. Badane modele

Rekurencyjne sieci neuronowe należą do rodziny sieci służących do przetwarzania sekwencji danych o określonej długości. Jak pokazali w swojej publikacji Shewalkar, Nyavanandi i Ludwig [10], sieć LSTM osiąga znacznie lepsze wyniki od podstawowej sieci rekurencyjnej RNN oraz minimalnie lepsze wyniki od sieci GRU, kosztem dłuższego czasu treningu. Z tego względu w swoich badaniach skupiam się głównie na warstwie LSTM oraz w mniejszej mierze na warstwie GRU, która również wyprzedza podstawową sieć RNN pod względem skuteczności.

Wszystkie eksperymenty badające wpływ hiperparametrów wymienionych w wstępnym rozdziale 1.5.1 przeprowadzę na sieci LSTM, oraz najlepszą ich kombinację zbadam przy wykorzystaniu warstwy GRU w celu porównania ich skuteczności w zadaniu przewidywaniu kodu. Na koniec zbadam również zachowanie wyznaczonego modelu dla różnych rozmiarów słownika.

Jak pokazał w swojej pracy Yoon Kim [11] połączenie warstwy rekurencyjnej z warstwą zanurzeń poprzez warstwę splotową może mieć pozytywny wpływ na skuteczność modelu. Jednak badanie to wykonywał jedynie w zadaniu modelowania języka naturalnego oraz dla modeli opartych na pojedyńczych znakach. Jednym z wykonanych przeze mnie eksperymentów będzie zastosowanie tej techniki w moim zadaniu, oraz oceny skuteczności tego podejścia dla modeli opartych na tokenach.

Rysunek 3.1. Ogólny badany model. Opcjonalne warstwy zaznaczone linią przerywaną



Długość sekwencji	Warstwa CNN	Liczba warstw LSTM	Liczba neuronów w warstwie	Liczba wszystkich parametrów
1	nie	1	512	todo
5	nie	1	128	todo
			256	todo
			512	todo
		2	128	todo
	tak	1	128	todo
10	nie	1	128	todo
			512	todo
		2	128	todo
		2	512	todo
	tak	1	126	todo
15	nie	1	128	todo
			512	todo

Tabela 3.1. Zestawienie wykonywanych eksperymentów

Zdecydowałem, że nie ma potrzeby testowania wszystkich możliwych kombinacji parametrów w tabeli, ponieważ zajęło by to bardzo dużo czasu, oraz już na podstawie prac [1], [4] możemy łatwo stwierdzić, że część kombinacji nie osiągnie konkurencyjnych wyników przy pozostałych, na przykład pojedyńcza warstwa LSTM, o 128 komórkach i długości okna równej 1 jest zdecydowanie za prostym modelem aby przechwycić wszystkie zależności między danymi.

3.2. Hipotezy

Główną hipotezą dla pytania badawczego 1.5.1 jest to, że rekurencyjne sieci neuronowe są w stanie w skuteczny sposób modelować języki programowania. Hipoteza ta oparta jest na bardzo dobrych wynikach tego typu sieci w zadaniu modelowania języka naturalnego oraz dużego podobieństwa pomiędzy tymi tymi językami, omówionymi w sekcji 1.2. Hipotezę tą popierają również prace Hellendoorn, Devanbu [1] oraz Subhasis Das, Chinmayee Shah [3] uzyskując obiecujące wyniki przy wykorzystaniu unikalnych metod rozwiązania problemów wiążących się z językami programowania.

3.2.1. Podpytanie 1: Jaki jest wpływ długości sekwencji na predykcje?

Hipoteza zerowa stanowi, że wpływ długości sekwencji danych, na których odbywa się trening, oraz które używane są do wykonania predykcji nie ma wpływu na skuteczność działania modelu. Hipoteza alternatywna mówi, że do pewnego momentu dłuższe sekwencje zapewniają lepszą jakość predykcji. Założenia te oparte są na pracach Alexey Svyatkovskiy z zespołem[4] oraz Erika van Scharrenburg [12]

3.2.2. Podpytanie 2: Czy wielkość słownika ma wpływ na działanie modelu?

Hipoteza zerowa stanowi, że wielkość słownika nie wpływa na na jakość wykonywanych predykcji. Hipoteza alternatywna mówi, że większe słowniki radzą sobie lepiej, do pewnego momentu, po czym pogarszają jakość całego modelu (skuteczność, rozmiar, szybkość). Hipoteza ta oparta jest o prace Hellendoorn'a i Devanbu [1] gdzie kolejne próby powiększania słownika nie przynosiły pozytywnych efektów.

3.2.3. Podpytanie 3: Czy liczba warstw rekurencyjnej sieci neuronowej ma wpływ na działanie modelu?

Hipoteza zerowa stanowi, że liczba warstw rekurencyjnych użytych przy treningu nie wpływa na jakość uzyskanego modelu. Hipoteza alternatywna mówi, że łączenie ze sobą kolejnych warstw sieci rekurencyjnych ma pozytywny wpływ na jakość modelu końcowego. Poparta jest ona pracą Afan Galih Salmana z zespołem[13], w której badają działanie łączenia ze sobą od 1 do 4 warstw LSTM, w zadaniu przewidywania szeregów czasowych, z których wynika, że większa liczba warstw radzi sobie lepiej.

3.2.4. Podpytanie 4: Czy liczba neuronów w warstwie wpływa na skuteczność modelu?

Hipoteza zerowa stanowi, że liczba neuronów wchodząca w skład pojedyńczej warstwy sieci rekurencyjnej nie ma wpływu na skuteczność modelu. Hipoteza alternatywna mówi, że większa liczba neuronów osiąga większą skuteczność od tych o mniejszej liczbie. Hipoteza ta oparta jest na publikacji S. Chakraborty wraz z zespołem [14], w której pokazują wzrost wydajności modelu opartego o sieć LSTM wraz ze wzrostem występujących neuronów w jego warstwie.

3.2.5. Podpytanie 5: Czy układ warstw zastosowanych w modelu wpływa na skuteczność modelu?

Hipoteza zerowa stanowi, że układ warstw zastosowanych w modelu nie ma znaczenia na jego wydajność. Hipoteza alternatywna mówi, że zastosowanie dodatkowej warstwy splotowej CNN poprawia działanie modelu. Hipoteza ta oparta jest o prace Kim'a [11], któremu udało się w ten sposób poprawić jakość badanego modelu.

3.2.6. Podpytanie 6: Czy rodzaj warstwy rekurencyjnej ma wpływ na skuteczność modelu?

Hipoteza zerowa stanowi, że rodzaj zastosowanej warstwy rekurencyjnej w zadaniu nie ma wpływu na działanie modelu. Hipoteza alternatywna mówi, że sieć LSTM sprawdza się lepiej w zadaniu modelowania języków od sieci GRU. Hipoteza ta oparta jest na publikacji Apeksha Shewalkar i Deepika Nyavanandi i Simone A. Ludwig[10], w której sieć LSTM osiągnęła najlepszą skuteczność ze wszystkich badanych rodzajów sieci.

3.3. Modelowanie tokenów

Jak pokazuje w swojej publikacji Hao Peng wraz z zespołem [15] mimo tego, że modele budujące kolejne słowa poprzez przewidywanie pojedyńczego znaku (character-level model) radzą sobie dobrze z modelowaniem języka naturalnego, oraz rozwiązują problem rozmiaru słownika, działają znacznie gorzej z językami programowania. Wniosek ten również potwierdza w swojej pracy Erik van Scharrenburg [12] porównując model przewidujący znaki z modelem przewidującym tokeny. Z tego powodu w realizuję tylko modele oparte na tokenach (token-level model).

Pierwszym krokiem jest zbudowania słownika tokenów, które mogą pojawić się w kodzie. Buduję go poprzez przetworzenie wszystkich kodów źródłowych obu zbiorów treningowego oraz walidacyjnego modułem tokenize [16] wbudowanym w język Python. Moduł ten przyjmuje na wejściu kod źródłowy programu następnie zwraca listę kolejnych tokenów (nazw zmiennych, znaków specjalnych, słów kluczowych). Upewnia się on również czy kod jest poprawnie napisany, na przykład czy wszystkie nawiasy lub apostrofy zostały zamknięte. Niepoprawne programy pomijam. Znaki nowej lini również traktuję jako token, jednak nie uwzględniam wcięć w kodzie ze względu na to, że większość środowisk programistycznych stawia je automatycznie. Na przykład z kodu źródłowego:

Listing 1. Przykładowy program Python

```
1    for x in range(2, 10):
2        print("hello world")
```

otrzymamy listę **[for, x, in, range, (, 2, 10,), :, \n, print, (, "hello world",)]**. Następnie sortuje wszystkie wygenerowane tokeny według częstości występowania oraz wybieram top-n tokenów jako słownik i każdemu z nich przypisuję unikalną liczbę naturalną. Utworzony w ten sposób słownik nie jest kompletny ponieważ nie obejmuje on wszystkich możliwych nazw występujących w kodzie. Takiego rodzaju tokeny zostają zastąpione sztucznym tokenem '<UNKNOWN>'. W głównej mierze są to unikalne nazwy zmiennych oraz ciągi znaków. Zastępowanie tokenu '<UNKNOWN>' prawdziwym prawdziwym tokenem omawiam w rozdziale 3.7.1

3.4. Wybór podzbioru danych

Jak już wspomniałem w sekcji 2.6 dotyczącej zbioru danych, trening odbywa sie na podzbiorze wszystkich zgromadzonych danych. W tym celu wybrałem 9 najpopularniejszych, zewnętrznych bibliotek języka Python, stosowanych w zróżnicowanych dziedzinach rozwoju oprogramowania. Poniżej zamieszczam listę wybranych bibliotek, wraz z krótkim opisem:

- Django - rozwój aplikacji sieciowych
- Numpy - wykonywanie obliczeń matematycznych wysokiego poziomu
- Requests - wysyłanie zapytań http
- Flask - rozwój aplikacji sieciowych
- TensorFlow - Głębokie uczenie maszynowe
- Keras - Wysokopoziomowe uczenie maszynowe
- PyTorch - Głębokie uczenie maszynowe
- Pandas - Zarządzanie dużymi zbiorami danych
- PyQt - Tworzenie interfejsów użytkownika

Aby upewnić się, że wybrany przeze mnie zbiór danych poprawnie oddaje rzeczywistość porównałem liczbę plików źródłowych zwierających konkretną bibliotekę ze zbioru z odpowiadającą liczbą plików źródłowych na platformie GitHub [5].

Tabela 3.2. Zestawienie zbioru danych z platformą GitHub

Biblioteka	Liczba plików GitHub	Liczba plików w zbiorze danych	stosunek
Django	187000000	26732	0.014%
Numpy	55000000	9058	0.016%
Requests	42000000	6339	0.015%
Pandas	19000000	1328	0.007%
Flask	17000000	3230	0.019%
TensorFlow	10000000	96	0.001%
Keras	3000000	72	0.002%
PyQt	1000000	132	0.013%
PyTorch	1000000	0	0%

Jak możemy zaobserwować stosunek liczby plików jest dosyć zbliżony dla większości wybranych bibliotek wynosi około 0.015%. Wyjątkami są biblioteki uczenia maszynowego. Może to wynikać z tego, że dane pochodzą z 2018 roku. Jest to czas, w którym istniały początkowe wersje tych bibliotek oraz dopiero zaczynały zyskiwać na popularności. Od tego czasu również biblioteka Keras została scalona z biblioteką TensorFlow co znacznie wpłynęło na jej popularność w dzisiejszych czasach.

Uznaję, że dane w wystarczająco dobrym stopniu oddają częstotliwość zastosowania bibliotek. Końcowy podzbiór stworzyłem poprzez wybranie 1% plików źródłowych z każdej biblioteki. Końcowe zestawienie prezentuje się w następujący sposób:

Tabela 3.3. Zestawienie zbioru i podzbioru danych

	Cały zbiór danych	Podzbiór treningowy	Podzbiór walidacyjny
Pliki	150000	9103	4504
Tokeny	114641650	9118453	4482600

Rozmiar wykorzystanego zbioru danych różni się do zbioru użytego w publikacjach Vincent J. Hellendoorn i Premkumar Devanbu [1] w którym użyto 16 milionów tokenów do treningu, oraz 5 milionów tokenów w celach walidacji. Na różnicę tą zdecydowałem się ze względu na ograniczenia sprzętowe różniące oba projekty.

3.5. Trening

Zadanie polega na przewidzeniu kolejnego tokenu na podstawie zadanej sekwencji tokenów. Długość sekwencji jest stała oraz wyrażona poprzez wielkość okna będącą jednym z badanych hiperparametrów. Dla każdego z tokenów model wyszukuje jego wektor zanurzenia, wykonuje jeden krok w sieci rekurencyjnej po czym stosuje warstwę klasyfikującą (Dense layer) w celu wygenerowania logitów wyrażających logistyczne-prawdopodobieństwo kolejnego tokenu. Zatem dla zadanego okna tokenów długości W: $[t_1, t_2, ... t_W]$ obliczam wynik dla każdego możliwego wyjścia j, s_j jako funkcję z wektorów tokenów v_{t_i} z tokenów z okna.

$$[g_1, g_2, ..., g_W] = RNN([v_{t_1}, v_{t_2}, ..., v_{t_W}])$$
$$s_j = p_j^T [g]$$

Minimalizowana funkcja strat jest entropią krzyżową pomiędzy prawdopodobieństwami $softmax$ dla każdego możliwego wyjścia a wykonaną predykcją.Funkcja wyrażoną wzorem:

$$L = log(\frac{e^{s_{t_o}}}{\sum_j e^{s_{t_j}}})$$

gdzie t_o jest zaobserwowanym tokenem wyjściowym a g_i wyjściem $i-tej$ komórki którejś z badanych sieci rekurencyjnych.

Wagi sieci aktualizowane są po przetworzeniu porcji danych (mini-batch), której rozmiar jest stały. Przy treningach sieci rekurencyjnych rozmiar ten jest jedną z wartości kluczowych dla dobrej wydajności sieci. W moich eksperymentach wynosi on 128. Jest to kompromis pomiędzy rozsądnym czasem treningu oraz jakością wyjściowych sugestii. Jest to również najczęściej wybierana wartość w przytoczonych przeze mnie publikacjach.

Każda testowana architektura trenowana jest przez 25 epoki. Wartość tą wybrałem na podstawie własnych eksperymentów wstępnych, z których wynika, że powyżej tej liczby model nie osiągał już lepszych rezultatów. Zbyt długi trening może również doprowadzić to przetrenowania modelu, czego należy unikać.

3.6. Ewaluacja

Jako, że badane przeze mnie modele tworzone są z myślą użycia ich w postaci wtyczki do środowiska programistycznego nie ma sensu ocenianie ich na podstawie pierwszej, najlepszej predykcji. Zamiast tego użyję dwóch następujących metryk:

3.6.1. Ewaluacja bez uwzględnienia kolejności

Jeśli poszukiwane słowo znajduje się w pierwszych n najlepszych predykcjach, bez znaczenia na którym miejscu uważam sugestię za poprawną. Metodę tą zastosowano przy ocenianiu systemów Pythia [4] dla której $n = 5$ oraz w pracy autorstwa Subhasis Das i Chinmayee Shah [3] gdzie $n = 3$. Zastosowanie jej pozwoli na porównanie wyników z tymi publikacjami.

3.6.2. Ewaluacja z uwzględnieniem kolejności

Jednym z przytoczonym problemów we wstępie 1 jest to, że wiele wtyczek nie uwzględnia kolejności sugestii oraz proponuje je na przykład posortowane leksykograficznie. Proponowane przeze mnie rozwiązanie sortuje predykcje na podstawie prawdopodobieństwa ich wystąpienia. Należy uwzględnić tą kolejność w metryce. Ocena obliczana jest poprzez podzielenie prawdopodobieństwa poprawnego poprawnej predykcji przez jego indeks w zbiorze zebranych predykcji. Dla przykładu powiedzmy, że model proponujący 10 sugestii zostaje użyty do wykonania 4 predykcji. Pierwsza wystąpi na 1. miejscu, druga

na 3. miejscu, trzecia nie zmieści w w 10 najlepszych, a czwarta na 8. miejscu. W takim przypadku ocena modelu będzie wynosić $(\frac{1}{1} + \frac{1}{3} + 0 + \frac{1}{8})/4 = 0.36$. W ten sposób wyższe predykcje oceniane są zdecydowanie lepiej. Przy ocenie użyję 10 najlepszych sugestii. Metryki tej używa Erik van Scharrenburg [12] w swojej pracy. Zastosowanie jej pozwoli na porównanie wyników.

3.7. Szczegóły implementacji

3.7.1. Problem słów poza słownikiem

Największym wyzwaniem przy modelowaniu języka programowanie jest rozwiązanie problemu słów poza słownikiem opisanego w podrozdziale 1.3. Jednym z podejść zastosowanym przez Subhasis Das i Chinmayee Shah jest zastępowanie tokenów nie występujących w słowniku specjalnymi tokenami pozycyjnymi. Tego typu tokenowi, który powtarza się więcej niż raz w sekwencji, zostaje przypisany indeks jego wystąpienia odpowiadający jego pierwszemu wystąpieniu. W przypadku gdy przewidziany token nie mieści się w słowniku ale pojawił się wcześniej w sekwencji zostaje zastąpiony wcześniej podaną nazwą. Rozwiązanie to sprawdza się bardzo dobrze dla zmiennych o tej samej nazwie, znajdujących sie blisko siebie, na przykład w pętli for języka $C++$:

Listing 2. Przeparsowana pętla for

```
1    for (int POS_TOKEN_01=0; POS_TOKEN_01<10; POST_TOKEN_01++)
```

Metoda ta jednak ogranicza się do długości badanej sekwencji, która jest bardzo krótka względem przeciętnej długości kodów programów.

Inną metodą jest zastosowanie warstwy splotowej zamiast warstwy zanurzeń zaproponowaną przez Yoon Kim [11]. Metoda ta osiąga skuteczność na poziomie najlepszych dotychczas znanych modeli, jednak zastosowana jest dla modeli języka naturalnego opartego na znakach, przez co prawdopodobnie nie sprawdziłaby się przy wybranych przeze mnie założeniach.

Hellendoorn wraz z Devanbu [1] proponują połączenie sieci rekurencyjnych z modelami statystycznymi $N-gram$. Metoda ta pozwala na bardziej ogólne predykcje. Pokazują również, że modele $N-gram$ radzą sobie lepie z przewidywaniem rzadko występujących unikalnych tokenów od sieci neuronowych, oraz kombinacja tych modeli osiąga mniejszą entropię niż każdy z tych modeli osobno.

Swoje eksperymenty przeprowadzam właśnie z zastosowaniem kombinacji sieci rekurencyjnej z modelami unigram oraz bigram, preferując podpowiedzi wykonane przez bigram. Zastosowanie większych modeli $N-gram$ nie ma sensu, ponieważ nałożyłoby to dodatkowe koszty obliczeniowe spowalniając wykonywanie predykcji, a generowany przez nie zbiór byłby w większości przypadków pusty. Stosuje tę metodę poprzez zastąpienie słów nie występujące w słowniku tokenem $<UNKNOWN>$ oraz uczę model przewidywać go w odpowiednich miejscach. Następnie jeśli pośród zebranych predyk-

cji znajdzie sie token $< UNKNOWN >$ zastępuje go zbiorem będącym sumą zbiorów predykcji unigramu oraz bigramu.

3.7.2. Rozmiar Słownika

W eksperymentach z tabeli 3.1 stosuję stałą wartość rozmiaru słownika wynoszącą 20000. Odpowiada ona usunięciu słów które nie pojawiają sie w zbiorze danych więcej niż 18 razy. Odcięte zostają w głównej mierze nazwy zdefiniowanych funkcji, zmiennych oraz ciągi znaków. Wybór ten spowodowany jest tym, że wartość ta ma ogromny wpływ na czas treningu oraz rozmiar modelu zapisanego na dysku. Wydłużony czas treningu spowodowany jest koniecznością obliczenia wartości funkcji $softmax$ dla każdego ze słów, natomiast dużo większy rozmiar wynika z konieczności przeskalowania warstwy wejściowej oraz wyjściowej. Rozmiar ten różni się od rozmiarów wybranych w przytoczonych przeze mnie pracach które wynoszą odpowiednio 74064 [1] oraz 2000 [3]. W celu odpowiedzi na pytanie czy rozmiar słownika ma znaczenie na badanie modelu przeprowadzę jeden eksperyment polegający na treningu modelu o najlepszej skuteczności na rozmiarze słownika 74064 zaproponowanego przez Hellendoorn'a [1]. Przykłady odciętych tokenów w słowniku rozmiaru 20000:
$accept\_inplace, IVGMM, book\_names, allvars, parent\_cards, 'references'$

3.7.3. Wektory zanurzenia

Wektory zanurzeń odpowiedzialne są za przypisanie każdemu ze słów wektora, którym możemy wyrazić prawdopodobieństwo tokenu jako rezultat wielowarstwowej sieci neuronowej na wektorach o ograniczonej liczbie tokenów sąsiednich. T. Mikolv [17] pokazuje, że taki model wykrywa relacje między parami słów na przykład słowo 'król' ma sie do 'królowa' tak samo jak 'mężczyzna' do 'kobieta'. Warstwa zanurzeń jets pierwszą warstwą w modelu. Wymiary wektorów zanurzeń są stałe dla każdego z wykonywanych eksperymentów oraz wynoszą one 32. Zdecydowałem się na mniejszy wymiar wektorów niż przedstawiony w pracy [1], która wynosiła 128 z powodu, że używam mniejszego słownika przez co znajduję się w nim dużo mniej zależności, które można wyrazić mniejszym wymiarem.

3.7.4. Optymalizator

W treningu używam optymalizatora $Adam$ o domyślnych parametrach dla każdego z modeli. Wybór ten wynika z tego, że zależy mi na badaniu różnic wynikających z badanej architektury. Odpowiednie strojenie optymalizatora typu SGD było by bardzo czasochłonne oraz komplikowałoby porównywanie ze sobą testowanych modeli. Jest to również optymalizator używany w pracach z którymi porównam uzyskane przeze mnie wyniki.

4. Analiza przeprowadzonych eksperymentów

4.1. Zbadane architektury

5. Implementacja wtyczki

6. Podsumowanie

Bibliografia

[1] V. J. Hellendoorn i P. Devanbu, "Are Deep Neural Networks the Best Choice for Modeling Source Code?", w *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017. Paderborn, Germany: ACM, 2017, pp. 763–773, 2017.

[2] *ml4code*, Dostęp zdalny: https://ml4code.github.io/papers.html.

[3] C. S. Subhasis Das, "Contextual Code Completion Using Machine Learning", prac. mag., Stanford, 2015.

[4] A. Svyatkovskiy, S. Fu, Y. Zhao i N. Sundaresan, "Pythia: AI-assisted Code Completion System", Microsoft, 2019.

[5] *github*, Dostęp zdalny: https://github.com.

[6] M. ROMANIUK, "Improving Code Completion in PharoUsing N-gram Language Models", prac. mag., Ukrainian Catholic University, 2020.

[7] Dostęp zdalny: https://tabnine.com.

[8] Dostęp zdalny: https://openai.com.

[9] srilab, *150k Python Dataset*, Dostęp zdalny: https://www.sri.inf.ethz.ch/py150.

[10] A. Shewalkar, D. Nyavanandi i S. A. Ludwig, "Performance Evaluation of Deep Neural Networks Applied to Speech Recognition: RNN, LSTM and GRU", *Journal of Artificial Intelligence and Soft Computing Research*, t. 9, nr. 4, s. 235–245, 1Oct. 2019. DOI: https://doi.org/10.2478/jaiscr-2019-0006. adr.: https://content.sciendo.com/view/journals/jaiscr/9/4/article-p235.xml.

[11] Y. Kim, Y. Jernite, D. Sontag i A. M. Rush, *Character-Aware Neural Language Models*, 2015. arXiv: 1508.06615 [cs.CL].

[12] E. van Scharrenburg, "Code Completion wiht Recurrent Nerual Networks", prac. mag., University of Amsterdam, 2018.

[13] A. G. Salmana, Y. Heryadib, E. Abdurahmanb i W. Suparta, *Single Layer and Multi-layer Long Short-Term Memory (LSTM) Model with Intermediate Variables for Weather Forecasting*, 2018.

[14] S. Chakraborty, J. Banik, S. Addhya i D. Chatterjee, "Study of Dependency on number of LSTM units for Character based Text Generation models", w *2020 International Conference on Computer Science, Engineering and Applications (ICCSEA)*, 2020, s. 1–5. DOI: 10.1109/ICCSEA49143.2020.9132839.

[15] H. Peng, L. Mou, G. Li, Y. Liu, L. Zhang i Z. Jin, "Building Program Vector Representations for Deep Learning", w *Knowledge Science, Engineering and Management*, S. Zhang, M. Wirsing i Z. Zhang, red., Cham: Springer International Publishing, 2015, s. 547–553, ISBN: 978-3-319-25159-2.

[16] *tokenize*, Dostęp zdalny: https://docs.python.org/3/library/tokenize.html.

[17] T. Mikolov, K. Chen, G. Corrado i J. Dean, *Efficient Estimation of Word Representations in Vector Space*, 2013. arXiv: 1301.3781 [cs.CL].

Wykaz symboli i skrótów

EiTI – Wydział Elektroniki i Technik Informacyjnych
PW – Politechnika Warszawska
WEIRD – ang. *Western, Educated, Industrialized, Rich and Democratic*

Spis rysunków

Spis tabel

Spis załączników

Załącznik 1. Nazwa załącznika 1

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus.

Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Donec odio elit, dictum in, hendrerit sit amet, egestas sed, leo. Praesent feugiat sapien aliquet odio. Integer vitae justo. Aliquam vestibulum fringilla lorem. Sed neque lectus, consectetuer at, consectetuer sed, eleifend ac, lectus. Nulla facilisi. Pellentesque eget lectus. Proin eu metus. Sed porttitor. In hac habitasse platea dictumst. Suspendisse eu lectus. Ut mi mi, lacinia sit amet, placerat et, mollis vitae, dui. Sed ante tellus, tristique ut, iaculis eu, malesuada ac, dui. Mauris nibh leo, facilisis non, adipiscing quis, ultrices a, dui.

Załącznik 2. Nazwa załącznika 2

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.