Rok akademicki 2015/2016

Politechnika Warszawska
Wydział Elektroniki i Technik Informacyjnych
Instytut Informatyki

PRACA DYPLOMOWA MAGISTERSKA

Barbara Rychalska

Sieci neuronowe w rozpoznawaniu podobieństwa semantycznego

Opiekun pracy
dr inż. Piotr Andruszkiewicz

Ocena: .....
.....

Podpis Przewodniczącego
Komisji Egzaminu Dyplomowego

Kierunek: Informatyka

Specjalność: Inżynieria Systemów Informatycznych

Data rozpoczęcia studiów: 2014.10.01

.....
Podpis studenta

EGZAMIN DYPLOMOWY

Złożył egzamin dyplomowy w dniu .....20__ r

z wynikiem .....

Ogólny wynik studiów: .....

Dodatkowe wnioski i uwagi Komisji: .....

.....

.....

STRESZCZENIE

Tematem niniejszej pracy jest zagadnienie rozpoznawania podobieństwa semantycznego tekstów w języku angielskim za pomocą sieci neuronowych. Zaproponowano trzy nowe architektury głębokiej sieci neuronowej opartej na modelu Vanilla-RNN, które służą do rozpoznawania podobieństwa pełnych zdań oraz pojedynczych fraz. Architektury te różnią się liczbą warstw ukrytych oraz wzorem połączeń sieci (wersja liniowa i drzewiasta). Skuteczność poszczególnych wersji algorytmów została sprawdzona na zbiorze ocenionych przez człowieka par zdań pochodzących z dwóch zadań konkursu SemEval.

Słowa kluczowe: podobieństwo semantyczne, rozpoznawanie parafraz, przetwarzanie języka naturalnego, sieci neuronowe

---

NEURAL NETWORKS IN SEMANTIC SIMILARITY RECOGNITION

The subject of this thesis is the problem of semantic similarity recognition in the English language with the use of deep neural networks. The thesis presents three new deep net architectures based on the model of Vanilla-RNN net, which are used to determine semantic similarity of full sentences and separate phrases. These architectures vary by the number of hidden layers and the pattern of connections (linear structure versus tree-like structure). The quality of results produced by each algorithm was verified on a dataset composed of human-annotated pairs of sentences stemming from two tasks from the SemEval contest.

Keywords: semantic similarity, paraphrase recognition, natural language processing, neural networks

Spis treści

1 Wstęp

Zadanie rozpoznawania podobieństwa semantycznego wypowiedzi, w literaturze często nazywane rozpoznawaniem parafraz, jest jednym z głównych problemów związanych z przetwarzaniem i rozumieniem języka naturalnego (NLP, ang. *Natural Language Processing*). Parafrazami nazywamy pary słów, wyrażeń lub zdań o tej samej treści, mogących różnić się budową leksykalną oraz składniową [1]. Rozpoznawanie treści synonimicznych nie sprawia trudności ludziom dzięki posiadanej przez nich intuicji, wynikającej z głębokiej znajomości języka. Tworzone obecnie algorytmy NLP dążą do uzyskania podobnego stopnia rozumienia wypowiedzi. Można wyróżnić 2 grupy stosowanych tu rozwiązań:

1. metody regułowe
2. metody statystyczne

Systemy regułowe często osiągają dużą dokładność oraz szybkość działania, są jednak kosztowne (ze względu na ogromną pracę wykonywaną przez zespoły ekspertów-lingwistów, którzy opracowują słowniki i bazy wiedzy dla poszczególnych języków oraz ręcznie piszą obszerne zestawy reguł) i są często ograniczone do jednej dziedziny. Systemy statystyczne natomiast pozwalają większość pracy oddelegować bezpośrednio do maszyny, dzięki czemu praca ekspertów jest ograniczana do niezbędnego minimum. Celem jest zbudowanie jak najlepszego modelu statystycznego, pozwalającego maszynie samodzielnie uczyć się oraz wnioskować na podstawie uprzednio przygotowanych danych treningowych. Jednym z najbardziej popularnych modeli tego typu są głębokie sieci neuronowe. Systemy zbudowane z ich pomocą osiągają obecnie sukcesy nie tylko w NLP, ale również w innych dziedzinach, takich jak rozpoznawanie obrazów, rozpoznawanie i generacja dźwięku czy gry dwuosobowe.

Celem niniejszej pracy jest prezentacja różnych architektur opartych na neuronowej sieci rekurencyjnej Vanilla-RNN dla problemu rozpoznawania parafraz w języku angielskim. Stworzone i przetestowane na użytek pracy nowe struktury wykorzystują szybkość trenowania sieci Vanilla, połączoną z elastycznością jej architektury. Przedstawione sieci do trenowania wykorzystują niewielkie zbiory danych (rzędu kilkuset przykładów), w odróżnieniu od innych opisywanych architektur (ilości danych rzędu kilkuset tysięcy przykładów). Przedmiotem testów jest wpływ poszczególnych modyfikacji architektur na jakość wyników. Trenowanie i testy przeprowadzane są na zbiorach

danych pochodzących z dwóch zadań konkursu SemEval [2]. Prezentowany system w jednym z zadań SemEval (Interpretable STS) osiąga wyniki porównywalne do ręcznie tworzonego rozwiązania regułowego, które wygrało konkurs w 2015 roku.

2 Wprowadzenie do przetwarzania języka naturalnego

Przetwarzanie języka naturalnego (NLP) to dziedzina sztucznej inteligencji skupiająca się na rozumieniu oraz generowaniu ludzkiego języka przez maszyny. Główne zadania NLP to między innymi:

- tłumaczenie maszynowe
- oznaczanie części mowy
- parsowanie
- odpowiadanie na pytania
- analiza wydźwięku

Z przetwarzaniem języka naturalnego wiążą się liczne i trudne do rozwiązania problemy. Po pierwsze, strukturę języka ciężko jest zapisać w postaci symbolicznej. Świadczyć o tym może mnogość istniejących gramatyk (takich jak gramatyki zależnościowe czy składnikowe), z której każda opisuje wybrany aspekt języka. Nie istnieje uniwersalna gramatyka, opisująca pełną charakterystykę budowy języka. Autorzy algorytmów NLP muszą brać to pod uwagę, wybierając do przetwarzania tekstów określony typ gramatyki.

Innym problemem związanym z gramatykami jest fakt, że w niektórych przypadkach dokonanie poprawnego rozkładu gramatycznego wymaga nie tylko znajomości reguł wybranej gramatyki, ale również wiedzy o otaczającym świecie. Na przykład, ustalenie do którego elementu zdania odnosi się przymiotnik w następujących zdaniach jest dla człowieka banalne, a dla komputera - przeciwnie:

- Daliśmy małpom banany, bo były już niejadalne.
- Daliśmy małpom banany, bo były głodne.

Zachodzące w języku abstrakcyjne zjawiska takie jak niejednoznaczność słów czy wyrażenia o znaczeniu przenośnym są przez ludzi przyswajane podczas ich interakcji z otaczającym światem. Stopień zrozumienia może wynikać

z czynników takich jak znajomość dzieł czy zwyczajów określonej kultury, aktualnej sytuacji politycznej lub praw rządzących światem fizycznym, których uczymy się empirycznie. Teksty w języku naturalnym często opisują środowisko życia ludzi, niejednokrotnie więc mamy do czynienia z pojęciami, które należy rozumieć w zupełnie różny sposób w zależności od kontekstu. Na przykład ustalenie czy słowo *wojna* w danym zdaniu jest pojęciem pozytywnym czy negatywnym, lub czy 50 kilogramów to duża masa w kontekście zdania na temat wagi dorosłego człowieka. Tego typu informacje nie są dostępne dla maszyn, ponieważ są one trenowane w izolowanym środowisku, skupionym na wykonaniu pojedynczego zadania. By zrozumieć całość kontekstów, być może konieczne byłoby wystawienie maszyny na identyczne bodźce, z którymi spotykają się ludzie, oraz zapewnienie, że będą one postrzegane w ten sam sposób.

W związku z licznymi problemami, do analizy języka wprowadzono poziomy abstrakcji. Opisują one kolejne kroki, jakie można podjąć przy badaniu języka, od przetwarzania prostych zależności do najbardziej złożonych zjawisk. Można wyróżnić następujące fazy analizy [3], z których korzystają algorytmy przetwarzania języka naturalnego:

1. Analiza morfologiczna - badanie sposobu tworzenia słów i odmiany części mowy. Podstawową jednostką analizy jest morfem – najmniejsza i niepodzielna część wyrazu. Morfemami są końcówki fleksyjne i elementy semantyczne słów. Na przykład, słowo *wyczytała* może zostać podzielone na następujące morfemy: przedrostek *wy*, rdzeń *czyt* i przyrostek *ała* . System komputerowy jest w stanie rozdzielić słowo na morfemy według reguł danego języka i na tej podstawie pozyskać dodatkowe informacje o znaczeniu zdania. Ilość informacji pozyskana w ten sposób zależy od typu języka. Na przykład, język polski jako język fleksyjny dysponuje morfemami czasownikowymi identyfikującymi takie informacje jak płeć i liczba wykonawców czynności. Języki analityczne takie jak angielski nie zawierają podobnych struktur. Nie jest możliwe więc automatyczne ustalenie wielu cech podmiotu czynności.

2. Analiza składniowa - badanie budowy wypowiedzi. Na tym poziomie analizowana jest struktura całego zdania. W ramach badania struktury przede wszystkim określane są funkcje poszczególnych wyrazów, pełnione w zdaniu (takie jak podmiot czy dopełnienie) oraz zależności między słowami w zdaniu. Kolejność wystąpienia słów bardzo często

wpływa na strukturę składniową oraz znaczenie wypowiedzi, szczególnie w języku polskim. Na przykład, dwa zdania:

- Klient zapłacił za nie mało.
- Klient nie zapłacił za mało

zawierają ten sam zbiór słów, a różnią się jedynie ich kolejnością, a co za tym idzie - składnią. W związku z tym znaczą zupełnie co innego.

3. Analiza semantyczna - ustalanie znaczenia zdania lub poszczególnych jego wyrazów. Sens słowa określany jest w stosunku do reszty zdania, tak by ujednolicić znaczenie słów potencjalnie wieloznacznych. Np. znaczenie słowa *mysz* jako gryzonia lub elementu sprzętu komputerowego w danym zdaniu może być ustalone tylko na podstawie znaczenia innych słów w tym zdaniu.

4. Analiza pragmatyczna - ustalanie znaczenia zdania w odniesieniu do kontekstu wypowiedzi, a więc innych zdań otaczających. Na tym poziomie rozwiązuje się najtrudniejsze problemy, takie jak rozwikłanie znaczenia wspomnianych już wyrażeń metaforycznych, a także wykrycie ironii lub komizmu wypowiedzi. Poprawne zinterpretowanie pragmatyki może być niełatwe nawet dla ludzi, gdyż zależy od czynników takich jak środowisko w jakim funkcjonują czy posiadana przez nich wiedza. Na przykład, znajdujące się w tekście odniesienie do zachowania postaci pochodzącej z literatury danego kręgu kulturowego może prowadzić do uzyskania przez autora efektu komicznego, który nie zostanie zrozumiany przez przedstawiciela innej kultury. By zapewnić konieczny zakres wiedzy, systemy komputerowe mogą stosować źródła zewnętrzne, takie jak zestawy reguł lub bazy wiedzy.

3 Wprowadzenie do problematyki rozpoznawania parafraz

Parafrazami nazywamy pary słów, wyrażeń lub zdań o tej samej treści, mogących różnić się budową leksykalną oraz składniową. Rozpoznawanie bliskości semantycznej zdań nie sprawia trudności ludziom dzięki posiadanej przez nich intuicji i wiedzy, wynikającej z głębokiej znajomości języka. Celem algorytmów NLP rozpoznających parafrazy jest uzyskanie podobnego stopnia zrozumienia języka, tak by mogły one oceniać bliskość semantyczną tekstów dowolnego rodzaju.

Zadanie rozpoznawania parafraz jest jednym z głównych problemów przetwarzania języka naturalnego. Może znaleźć zastosowanie szczególnie w systemach dialogowych, których celem jest odpowiadanie na pytania użytkownika (np. interfejsy encyklopedyczne) lub wykonywanie jego poleceń (np. inteligentne domy). Systemy takie korzystają z dodatkowych źródeł wiedzy, takich jak bazy wiedzy, które zawierają odpowiedzi na potencjalne pytania użytkownika. Zadanie uzyskania odpowiedzi polega na odpowiednim dopasowaniu pytania do przechowywanej odpowiedzi lub możliwej do wykonania czynności [4]. Jeden z problemów które trzeba tu rozwiązać to fakt, że to samo pytanie może być zadane na wiele sposobów (różnice leksykalne, składniowe). System rozpoznawania parafraz może wspierać poszukiwanie odpowiedzi, dokonując mapowania synonimicznych pytań do jednego wzorca, dla którego znana jest odpowiedź lub możliwe jest wykonanie pożądanej czynności. Na przykład, kolejne wypowiedzi użytkownika są synonimiczne w kontekście systemu odpowiadającego na pytania dotyczące wiedzy ogólnej:

- Podaj imię i nazwisko autora Sonaty Księżycowej.
- Który ze słynnych muzyków skomponował Sonatę Księżycową?

System może przyporządkować te pytania do tego samego wzorca, dla którego znana jest odpowiedź: Ludwig van Beethoven.

Zadanie rozpoznawania parafraz jest ciekawym eksperymentem nawet bez wykorzystania go w konkretnym zastosowaniu. Pozwala ocenić stopień intuicji językowej wykazywanej przez dany algorytm w stosunku to wzorca, jakim jest człowiek. Ocena parafraz zawiera wszystkie najtrudniejsze wyzwania przetwarzania języka naturalnego, takie jak analiza kontekstu i wymagana

wiedza o świecie. Dzięki temu możemy ocenić, jak dużo jeszcze brakuje maszynom do właściwego stopnia zrozumienia rzeczywistości.

3.1 Skale parafrazowania

Zasadniczym problemem, który należy rozwiązać podczas przygotowywania danych trenujących i testowych dla algorytmów rozpoznawania parafraz jest określenie skali parafrazowania. Skala ta określa zestaw dopuszczalnych etykiet, które system ma przypisywać wejściowym parom zdań. Może to być określony zakres liczb całkowitych (skala binarna 0-1 lub szersza), zakres ciągły (dopuszczający wartości ułamkowe), lub też zakres opisowy. Zakres opisowy pojawił się dotychczas tylko w pojedynczym zadaniu pilotażowym konkursu SemEval oraz dotyczy porównywania fraz (zamiast całych zdań). Został on skrótowo opisany w Rozdziale 3.2.2 niniejszej pracy. Poniżej natomiast przedstawiam 2 najpopularniejsze skale stosowane dla oceny pełnych zdań, powszechnie pojawiające się w publikacjach.

3.1.1 Skala SemEval

Skala parafrazowania, do której odwołuje się większość publikacji naukowych opiera się na ustaleniach zadania Semantic Textual Similarity (STS) z konkursu SemEval [2]. Jest to jeden z najbardziej prestiżowych konkursów rozwiązań NLP, w ramach którego każdego roku publikowana jest seria zadań dotyczących różnych zagadnień przetwarzania języka naturalnego, takich jak detekcja emocji, systemy dialogowe czy rozumienie języka figuratywnego.

Według ustaleń SemEval, podobieństwo semantyczne definiowane jest na 6 poziomach, gdzie 0 oznacza parę zdań o zupełnie różnym znaczeniu, zaś 5 – parę zdań synonimicznych (o identycznym znaczeniu). Punktacja 3-4 zarezerwowana jest dla par w których jedno ze zdań zawiera dodatkowe informacje, których brak w drugim, przy czym niższa ocena zawsze oznacza mniejsze pokrewieństwo semantyczne pomiędzy zdaniami. Ocenę 1-2 otrzymują pary, w których występują poważne rozbieżności (np. zmieniony jest podmiot – wykonawca czynności). W Tablicy 1 prezentuję przykładowe pary zdań wraz z ocenami.

W praktyce oceny par zdań pojawiające się w zbiorach trenujących, testowych oraz walidacyjnych SemEval często są liczbami z częścią ułamkową.

5.0	1) The man cut down a tree with an axe. 2) A man chops down a tree with an axe.
4.0	1) Some men are fighting. 2) Two men are fighting.
3.0	1) People are playing cricket. 2) Men are playing cricket.
1.5	1) The man is playing the piano. 2) The man is playing the guitar.
0.0	1) A woman is slicing big pepper. 2) A dog is moving its mouth.

Tablica 1: Przykłady ocen w skali SemEval.

Wynika to z braku jednomyślności pomiędzy lingwistami oceniającymi pary. Każdą parę zdań oceniało kilka osób, tak by możliwe było wyeliminowanie pomyłek, które w sposób naturalny czasem pojawiają się podczas pracy ekspertów. Niestety, zasady przyznawania ocen nie zostały dokładnie zdefiniowane w udostępnionych materiałach konkursu. Nie można wprawdzie wykluczyć możliwości, że pracującym nad zbiorami SemEval ekspertom udostępniono jakiś zestaw reguł, według których mieli pracować, jednak szczegółowa analiza wybranych par zdań wskazuje na to, że nie stosowano się do stałych kryteriów oceniania (np. identyczne pary zdań pojawiają się wielokrotnie i za każdym razem mają inną ocenę). Prawdopodobnie więc eksperci pracujący nad punktowaniem zdań musieli kierować się w dużym stopniu subiektywnym wyczuciem. Powoduje to dodatkowe problemy dla systemów maszynowych, ponieważ w takich warunkach podstawą do podjęcia przez nie decyzji mogą stać się niepoprawne wnioski sformułowane na podstawie szumu informacyjnego. Wprowadza to do działania systemów element przypadkowości i może powodować trudne do wytłumaczenia błędy. Pomimo to, dane i sposób anotacji SemEval są obecnie najczęściej używane, korzysta z nich również niniejsza praca.

Nie ma pewności czy tego typu błędy da się wyeliminować ze zbiorów danych. Ustalenie ścisłego zbioru reguł dla tak skomplikowanego zadania jest trudne ze względu na problemy z opisem struktury języka i skłonność do występowania wyjątków od każdej zasady go dotyczącej. Wydaje się, że nawet przy najlepszym zestawie reguł po rozszerzeniu zbioru o przykłady pochodzące z nowego źródła pojawią się takie przykłady, których nie będą

pokrywały żadne reguły, lub w których dojdzie do konfliktu reguł. Ocena tych przykładów będzie więc zależała od indywidualnego wyczucia osoby oceniającej. Tak więc nie można jednoznacznie stwierdzić, że oparcie na ludzkiej intuicji w ocenach danych SemEval jest błędem.

3.1.2 Skala MS Paraphrase

Konkurencyjną skalą parafrazowania jest skala binarna (0-1). Ta skala została przyjęta w zbiorze Microsoft Research Paraphrase Corpus [5], również popularnym wśród autorów systemów rozpoznawania parafraz. W skali binarnej nie określa się stopnia podobieństwa semantycznego. Wszystkie zdania otrzymujące według SemEval oceny poniżej 4 lub 5 powinny w tym systemie odtrzymać ocenę 0. Ocena 1 jest zarezerwowana dla doskonałych oraz bardzo bliskich semantycznie parafraz.

3.2 Zbiory danych SemEval

W ramach konkursu SemEval udostępniono dane pochodzące z wielu źródeł, tak by zapewnić różnorodność stylów, słownictwa i zawartości semantycznej. Zbiory te rozdzielone są na 2 duże zadania: ogólne podobieństwo znaczeniowe (Zadanie 1: Semantic Textual Similarity) oraz szczegółowe podobieństwo fraz (Zadanie 2: Interpretable Semantic Textual Similarity). Z roku na rok publikowane są nowe zbiory. Każdy z nich zawiera inne wyzwania dla rywalizujących systemów, takie jak specjalistyczne słownictwo (np. naukowe, codzienne lub charakterystyczne dla mediów społecznościowych) i specyficzne konstrukcje gramatyczne (uproszczone lub wręcz zaburzone gramatycznie wypowiedzi z Twittera kontra złożone zdania z publikacji o polityce). Z powodu dużych różnic w charakterystyce tekstów autorzy rozwiązań maszynowych często budują kilka modeli dla swoich algorytmów, każdy przeznaczony dla innego typu tekstu ([6], [7], [8]).

3.2.1 Zadanie 1: Semantic Textual Similarity

W ramach Zadania 1 dostępne jest wiele zestawów danych. Poniżej zamieszczony został opis tych zbiorów, które zostały zastosowane do trenowania i testowania systemu prezentowanego w tej pracy. Są to następujące 2 zbiory:

- *MSRvid* – opisy obrazków z serwisu Flickr, przedstawiających ludzi i zwierzęta. Opisy te są krótkie i nieskomplikowane gramatycznie. Zbiór zawiera 750 par zdań treningowych i 751 par zdań testowych.
- *MSRpar* – zdania zaczerpnięte z gazet i publikacji. W porównaniu do zbioru *MSRvid* są to zadania zdecydowanie bardziej skomplikowane, o tematyce obejmującej ekonomię, politykę i problemy świata współczesnego. Zbiór zawiera 750 par zdań treningowych i 750 par zdań testowych.

Poza tym dostępne są zbiory złożone z pytań i odpowiedzi pobranych z forów internetowych (zbiory *answers*), zbiory zebrane z systemów antyplagiatowych (zbiór *plagiarism*) oraz wielu innych ciekawych źródeł.

3.2.2 Zadanie 2: Interpretable Semantic Textual Similarity

Zadanie to polega na zbudowaniu dopasowań pomiędzy częściami zdania, które decydują o tym, że dana para zdań jest uznana na parafrazę lub nie. Zdania wejściowe podzielone są na frazy (ang. *chunks*), zadaniem algorytmu jest ocena podobieństwa tych fraz. Podział na frazy może być dokonywany przez algorytm uczestniczący w konkursie lub można korzystać z gotowych podziałów fraz. Podobieństwo należy ocenić w skali 0-5 oraz dodatkowo w skali identyfikatorów:

- EQUI – frazy są semantycznymi ekwiwalentami:
  - *in Olympics*
  - *at Olympics*
- OPPO – fraza 1 jest odwrotnością frazy 2 w danym kontekście lub też bez wzięcia pod uwagę kontekstu:
  - *lower*
  - *higher*
- SPE1 i SPE2 - jedna z fraz zawiera więcej szczegółów:
  - *Palestinian prisoners*

  - *104 Palestinian prisoners*

- SIM - pomiędzy frazami istnieje pokrewieństwo semantyczne lecz nie jest spełniona żadna z zależności EQUI, OPPO, SPE1 i SPE2:
  - *Israeli envoys*
  - *Israeli envoy*
- REL - pomiędzy frazami istnieje mniejsze pokrewieństwo niż SIM i nie jest spełniona żadna z powyższych zależności:
  - *against protesters*
  - *about protests*
- ALIC - specjalny identyfikator dla fraz które nie mogą być dopasowane ze względu na ograniczenie dopasowania fraz 1:1.
- NOALI - fraza nie ma swojego odpowiednika wśród fraz drugiego zdania.

W niniejszej pracy do testów używany jest zbiór *images* - jego struktura jest podobna do zbioru *MSRvid* z poprzedniego zadania. Zadanie podziału na frazy nie jest częścią tematyki rozważanych algorytmów, więc używane są podziały zdań udostępnione przez SemEval.

4 Przegląd istniejących systemów rozpoznawania parafraz

Pod względem sposobu działania, systemy rozpoznawania parafraz (jak i inne systemy NLP) można przyporządkować do jednej z dwóch podstawowych grup: regułowej (ang. *rule-based systems*) oraz statystycznej [9]. Systemy regułowe często osiągają dużą dokładność, są jednak kosztowne – ze względu na ogromną pracę wykonywaną na ich potrzeby przez zespoły ekspertów-lingwistów – i z reguły ograniczone do jednej dziedziny. Najczęściej opierają się na dodatkowych źródłach wiedzy takich jak specjalistyczne słowniki, które są przeszukiwanie za każdym razem gdy system musi podjąć decyzję.

Systemy statystyczne natomiast pozwalają większość pracy oddelegować bezpośrednio do maszyny, dzięki czemu praca ekspertów jest ograniczana do niezbędnego minimum. Celem systemów tej klasy jest samodzielne zbudowanie przez algorytm jak najlepszego modelu danej dziedziny, pozwalającego maszynie wnioskować. Uczenie i testowanie algorytmów statystycznych odbywa się z użyciem uprzednio przygotowanych zestawów danych treningowych i testowych. Na tych samych danych można trenować różne algorytmy, co pozwala na łatwe porównanie jakości ich działania. Jednym z najbardziej teraz popularnych modeli tego typu są głębokie sieci neuronowe.

Istnieją także systemy hybrydowe, łączące zalety i wady obu podejść. Podobnie, systemy regułowe mogą zawierać prosty klasyfikator statystyczny, zaś systemy statystyczne często pracują na wstępnie przetworzonych danych. Wstępne przetwarzanie może opierać się na prostych zasobach reguł lub zasobach lingwistycznych, np. zbiorach tzw. *stopwords*, czyli słów o małej wartości semantycznej (takich jak zaimki osobowe czy czasowniki posiłkowe), które są usuwane z tekstów na pierwszym etapie przetwarzania. Mimo tych cech wspólnych, w większości istniejących systemów można zauważyć wyraźną przewagę jednego paradygmatu: albo części statystycznej, albo regułowej.

4.1 Systemy regułowe

Przed przedstawieniem metod opartych na głębokich sieciach neuronowych opisany zostanie przykładowy system regułowy opisany w [10] - system UMBC EBIQUITY, który osiągnął najlepszy wynik w konkursie SemEval2014. Na

tym przykładzie będzie można następnie pokazać zalety systemów statystycznych, a w szczególności sieci neuronowych.

Metody regułowe podobne do systemu UMBC EBIQUITY dominują w rozpoznawaniu parafraz. Ich cechą wyróżniającą jest silne oparcie na zewnętrznych zasobach przygotowanych przez ekspertów, takich jak zbiory synonimów (zasób WordNet [3]), słowniki, ręcznie budowane bazy wiedzy czy gotowe zbiory reguł pisane przez ludzi. Rozwiązania takie są niezwykle kosztowne oraz mało dynamiczne – słowniki muszą być regularnie aktualizowane, by dotrzymać tempa zmieniającemu się słownictwu. Co więcej, dobrej jakości materiały słownikowe istnieją tylko dla niektórych (najpopopularniejszych) języków. Element uczenia maszynowego najczęściej jest w systemach regułowych zredukowany do minimum – system wykonuje proste przetwarzanie wstępne zdań (najczęściej jest to stemowanie i tokenizacja przeprowadzane z użyciem zewnętrznych narzędzi), po czym główna część pracy to przeszukiwanie słowników celem sprawdzenia, czy spełniony jest określony przez ekspertów zbiór reguł. Dodatkowo, w systemach tych często występują miary podobieństwa parametryzowanie współczynnikami, które muszą być wyznaczane w drodze eksperymentu. Same wzory miar podobieństwa też są kolejno sprawdzane i wybierana jest miara, która najbardziej pasuje do danego zadania. Konieczność przeszukiwania zasobów dla każdej pary zdań wejściowych może powodować wolniejsze działanie tych systemów.

System UMBC EBIQUITY dla każdego słowa z danego zdania próbuje odnaleźć jak najbardziej znaczeniowo spokrewnione słowo w zdaniu drugim. Schemat ten pasuje do większości tego typu rozwiązań. Różnice występują w zastosowanych miarach podobieństwa oraz metodach wyznaczania kar i nagród. Podstawowy schemat działania można przedstawić w następujących krokach:

1. Pobierz słowo S ze zdania 1.
2. Oblicz odległość wektorów słów w zdaniu 2 względem wektora słowa S używając danych kryteriów podobieństwa.
3. Pobierz informacje na temat bliskości semantycznej (zaczerpnięte z zewnętrznych słowników), zwiększ miarę podobieństwa dla odnalezionych sąsiadów semantycznych.

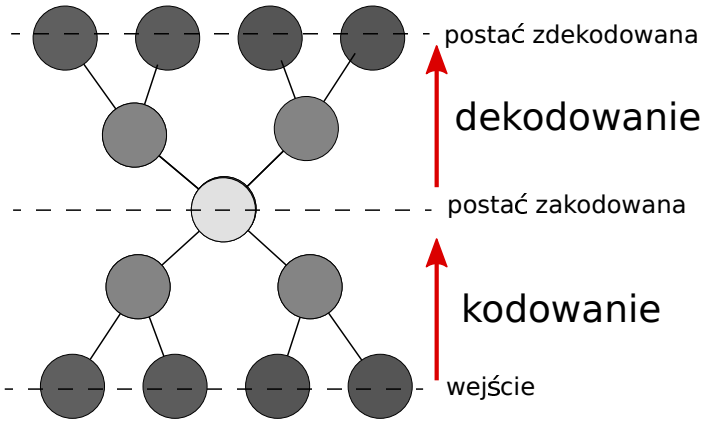
4. Wyszukaj „fałszywe synonimy” lub inne pary, które należy karać (również w zewnętrznych słownikach), dodaj modyfikator kary dla odnalezionych par.

5. Spośród wszystkich par wybierz najlepsze dopasowanie (największą miarę podobieństwa).

Systemy luźno oparte na tym schemacie zwyciężały w ostatnich edycjach SemEval, począwszy od roku 2013 [11] [12]. Często osiągają one wyniki lepsze niż systemy statystyczne, ponieważ w sposób bezpośredni opierają się na ludzkiej wiedzy.

4.2 Pierwsza sieć neuronowa do rozpoznawania parafraz – autoenkoder z dynamicznym odbijaniem

Pierwsza sieć, która wyznaczyła wynik rekordowy w identyfikacji parafraz (choć nie w konkursie SemEval, a na zbiorach Microsoft Research Paraphrase Corpus) została opisana w [13]. Sieć ta ma architekturę autoenkodera (ang. *autoencoder*), przedstawioną na Rys. 1. Oznacza to, że sieć składa się z 2 części: kodującej i dekodującej. Na wyjściu części dekodującej sieć próbuje odtworzyć sygnał podany na wejście części kodującej.

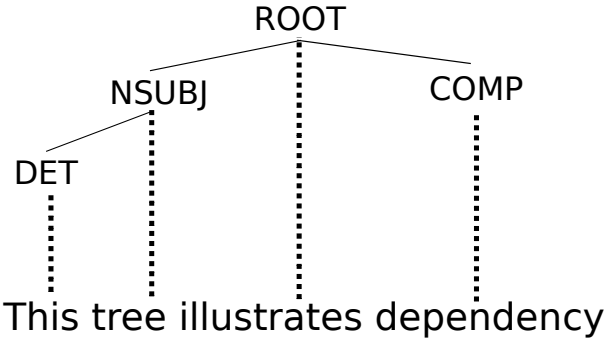


Rysunek 1: Architektura autoenkodera

Ten typ sieci może być trenowany za pomocą danych nieustrukturyzowanych i nie zaanotowanych (nie przypisanych do konkretnych klas), np.

tekstów pobranych wprost z prasy czy internetu, co pozwala na stosowanie go dla dowolnego języka oraz dziedziny tekstu. Nie są potrzebne żadne dodatkowe zasoby lingwistyczne.

Część kodująca autoenkodera ma postać drzewa zależnościowego zdania [14]. Przykład drzewa rozbioru zdania zgodnego z gramatyką zależnościową pokazany został na Rys. 2. Ważną cechą tych drzew jest fakt, że dzieckiem każdego węzła jest dokładnie jeden węzeł terminalny - liść (w którym znajduje się słowo zdania) oraz - w ogólnym przypadku - dowolna liczbę węzłów nieterminalnych. Węzły nieterminalne reprezentują kolejne frazy. Autoenkoder wykorzystuje zbinaryzowaną formę drzew zależnościowych. W tych uproszczonych drzewach występuje dodatkowe ograniczenie: każdy węzeł może posiadać tylko jedno dziecko będące węzłem nieterminalnym. Drzewo przedstawione na Rys. 2 nie spełnia tego warunku, ponieważ węzeł ROOT posiada dwoje dzieci, które są węzłami nieterminalnymi: frazy COMP i NSUBJ. Narzędzia programistyczne takie jak parser zależnościowy z biblioteki Stanford Core NLP pozwalają na tworzenie wybranego rodzaju drzew: zbinaryzowanych lub niezbinaryzowanych, zależnie od opcji podanej przez użytkownika.



Rysunek 2: Drzewo rozkładu zależnościowego przykładowego zdania: *This tree illustrates dependency.*

W liściach drzewa autoenkodera znajdują się wektory reprezentacji (udostępnione w [15]) dla kolejnych słów pojedynczego zdania. Każda para liści jest łączona w wektor o długości 2 razy większej niż pojedynczy liść oraz kodowana (kompresowana) do rozmiaru pojedynczego liścia i umieszczana w węźle rodzica. Pierwszy etap działania enkodera kończy się, gdy całe zdanie jest już zakodowane w najwyższym węźle. Następnie rozpoczyna się procedura dekodowania (odbijania), podczas której powstaje lustrzane odbicie

drzewa kodowania. Procedura ta pozwala zweryfikować, jak użyteczne informacje zostały zachowane w węzłach drzewa podczas kompresji. Trenowanie sieci redukuje błąd dekodowania (jest to odległość euklidesowa pomiędzy wektorami reprezentacji liści drzewa oryginalnego i drzewa odbitego) , co po pewnym czasie trenowania pozwala na uzyskanie optymalnego algorytmu kompresji w węzłach – po dekompresji otrzymujemy wartości nieznacznie różniące się od wartości wejściowych. Ostatecznie otrzymujemy algorytm kompresji o niewielkiej stracie informacji. Kompresja wszystkich węzłów do ustalonego rozmiaru jest konieczna, by można było porównać ze sobą 2 zdania o różnych długościach.

Porównanie 2 najwyższych węzłów pary drzew (np. na podstawie wyliczenia odległości euklidesowej pomiędzy skompresowanymi wektorami) dla pary zdań pozwala ustalić różnicę semantyczną tych zdań. Jak wykazały eksperymenty, zakodowane reprezentacje par parafraz są istotnie położone blisko siebie na płaszczyźnie. Odległość ta zwiększa się wraz z rosnącym oddaleniem semantycznym zdań. Zamiast prostego porównania wektorów można też stosować metodę porównania polegającą na złożeniu wszystkich węzłów pary drzew w macierz odległości i przeprowadzenie tzw. poolingu – redukcji rozmiaru macierzy za pomocą przesuwającego się okna. Metoda ta nieznacznie poprawia wyniki.

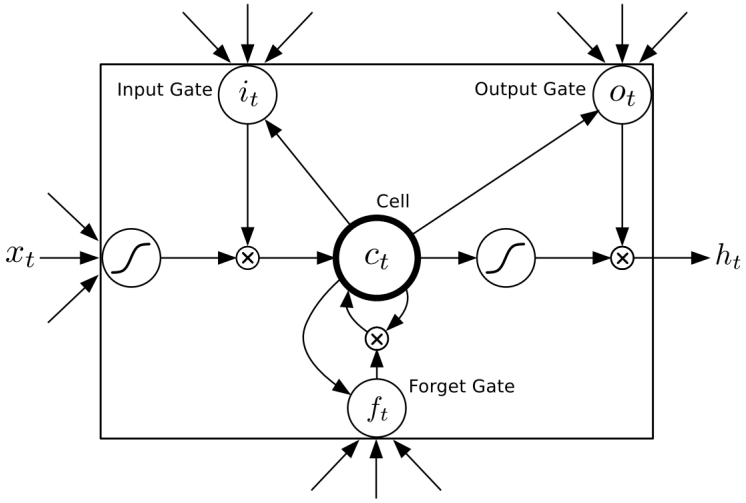
Autoenkoder z dynamicznym odbijaniem udowodnił przydatność sieci neuronowych do rozpoznawania głębokiego znaczenia języka. Wyniki przez niego osiągane przewyższyły wszystkie istniejące w czasie publikacji rozwiązania tradycyjne na zbiorze Microsoft Research Paraphrase Corpus.

4.3 Nowoczesne sieci rekurencyjne

Obecnie uważa się, że jeszcze lepsze działanie można uzyskać wprowadzając do sieci skomplikowane zależności rekurencyjne. Dodatkowe pętle wprowadzają możliwość wzmacniania sygnału podczas propagowania informacji poprzez sieć, co pozwala uniknąć problemu zanikającego gradientu (ang. *vanishing gradient problem*), który polega na stopniowym zmniejszaniu tempa uczenia w dalekich węzłach (warstwach) sieci. Problem ten powstaje, ponieważ podczas uczenia na każdym poziomie sieci następuje mnożenie przez siebie wartości ułamkowych mniejszych od 1. Wielokrotne mnożenie tak małych wartości powoduje stopniowe zanikanie sygnału. W zaawansowanych

sieciach rekurencyjnych nie obserwuje się zanikającego gradientu, co prowadzi do założenia, że trenowanie wszystkich węzłów przebiega w mniej więcej jednakowym tempie. Sieci te są znacznie bardziej skomplikowane koncepcyjnie niż tradycyjne architektury.

Publikacja [16] przedstawia zastosowanie sieci typu LSTM (ang. *Long Short-Term Memory*), po raz pierwszy opisanych w [17], do problemu rozpoznawania parafraz. Sieć LSTM składa się z bramek (Rys. 3), pozwalających na szczegółowe manipulacje przekazywaną informacją – selektywne zapamiętywanie, propagację do kolejnych węzłów oraz modyfikacje – daje to możliwości nieporównywalnie większe, niż zwykła komórka enkodera, dokonująca prostego mnożenia macierzy i pojedynczego nałożenia tzw. nieliniowości (np. funkcji tangensa hiperbolicznego lub funkcji logistycznej).



Rysunek 3: Bramka LSTM. Źródło: [17]

W pojedynczej bramce LSTM mamy do czynienia z wielokrotnym stosowaniem nieliniowości i skomplikowanym, wielostopniowym nakładaniem filtrów na sygnał.

Proponowana w publikacji architektura łączy bramki LSTM w strukturę drzewiastą, zgodną z drzewem parsowania zależnościowego zdania. Użycie poprawnego gramatycznie drzewa podyktowane jest intuicją, zgodnie z którą lepiej jest operować na reprezentacjach sensownych fraz (np. fraz rzeczownikowych lub czasownikowych) niż przypadkowych kombinacji słów. Parsowanie zależnościowe wykonywane jest przez zewnętrzne biblioteki. Podobnie

jak w przypadku autoenkodera, podobieństwo semantyczne ustalane jest na podstawie obliczonej odległości (liczonej wybraną miarą) dwóch najwyższych węzłów porównywanych zdań. Opisany system osiągnął najlepszy wynik dla zadania rozpoznawania parafraz na zbiorach SICK [18] – 0.89 korelacji Pearsona. W ostatnich latach pojawiło się wiele innych publikacji wykorzystujących sieci rekurencyjne, takie jak LSTM i GRU (Gated Recurrent Neural Network, [19]) np. [20] i [21]. Wydaje się, że jest to obecnie wiodący kierunek badań w rozpoznawaniu parafraz.

5 Proponowane rozwiązanie – sieć Vanilla-RNN dla rozpoznawania parafraz

Przedstawione dotychczas systemy sieci rekurencyjnych opierają się na złożonych architekturach. Niniejsza praca koncentruje się natomiast na zbadaniu przydatności prostszego modelu sieci rekurencyjnej – sieci typu Vanilla. Jest to podstawowy, popularny typ sieci rekurencyjnej. Sieci Vanilla były już z sukcesem stosowane w zadaniach powiązanych z przetwarzaniem sekwencji znaków, słów i dźwięków, takich jak generacja tekstu [22] czy generacja pisma odręcznego [23].

Właśnie to udowodnione powodzenie w przetwarzaniu sekwencji było jednym z głównych czynników, który zadecydował o wybraniu sieci Vanilla jako podstawy modeli zaprezentowanych w niniejszej pracy. Rozumienie tekstu może być bowiem uznane za formę przetwarzania sekwencji słów. Ponadto, typ Vanilla jest prostszy koncepcyjnie niż inne typy sieci rekurencyjnych (np. wspomniane wyżej LSTM i GRU) oraz znacznie szybszy w trenowaniu, ze względu na nieskomplikowaną architekturę bramki.

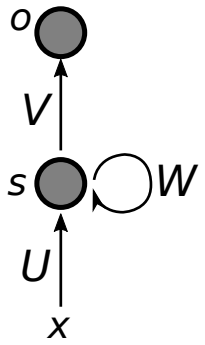
W ramach pracy prezentuję 3 nowe architektury oparte na sieci Vanilla oraz sprawdzam ich działanie w porównaniu do architektury podstawowej. Celem modyfikacji było polepszenie jakości rozpoznawania parafraz z zachowaniem dużej szybkości trenowania sieci typu Vanilla.

Szybkość i prostota sieci Vanilla mają potencjalnie znaczenie dla wszystkich systemów, których będą częścią, a szczególnie systemów dialogowych, w których istotna jest szybkość reakcji oraz stabilność systemu. Aby zapewnić niezawodne działanie, ważne jest zarówno szybkie trenowanie (niepożądane długie przerwy celem uaktualnienia modelu) jak i szybkie podejmowanie decyzji w reakcji na działanie użytkownika. Szybkość działania jest tym ważniejsza, że system rozpoznawania parafraz może być tylko jednym z wielu stopni przetwarzania tekstu w systemie dialogowym, więc dochodzić będzie do kumulacji opóźnień pochodzących ze wszystkich elementów.

Dodatkową zaletą prezentowanych architektur (niezwiązaną już z faktem, że są one oparte na sieci Vanilla) jest możliwość trenowania ich na ograniczonej ilości danych. W porównaniu do opisywanego w Rozdziale 4.2 autoenkodera, który do wytrenowania wymagał kilkuset tysięcy zdań, prezentowane w tej pracy sieci są trenowane wyłącznie na zbiorach SemEval (w sumie kilka tysięcy zdań).

5.1 Podstawowa architektura sieci Vanilla-RNN

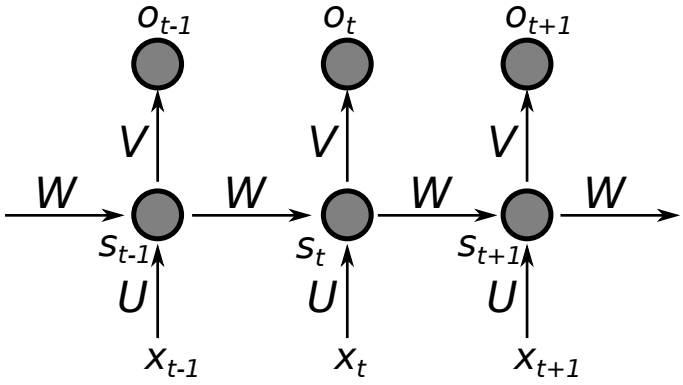
Schemat przedstawiony na Rys. 4 pokazuje podstawową postać sieci Vanilla z zaznaczoną zależnością rekurencyjną (strzałka powracająca do stanu s). Aktualny stan s ustalany jest na podstawie stanu poprzedniego oraz sygnału podawanego na wejście – x. Wyjście o zależy tylko od aktualnego stanu i pozwala na generowanie wartości wyjściowej. Wyliczenie stanu wyjścia o nie jest konieczne do działania sieci, ma znaczenie tylko dla zadań o charakterze generacyjnym.



Rysunek 4: Sieć Vanilla-RNN

Sieć Vanilla można również pokazać w postaci rozwiniętej w czasie (Rys. 5). Ta postać pokazuje sekwencję czasową kolejnych stanów. Zgodnie z oznaczeniami na rysunku, w każdym kroku czasowym używane są te same zestawy parametrów (U, V oraz W), w odróżnieniu od nie-rekurencyjnych głębokich sieci neuronowych, gdzie na różnych warstwach mogą być używane różne zestawy parametrów.

Sieć Vanilla-RNN służącą do modelowania sensu zdań można przedstawić jako sieć, której stan w danej chwili czasowej t jest zależny od dotychczas wczytanych słów (z chwil *t-1*, *t-2*, itd.). Każde kolejne słowo zdania pełni rolę sygnału wejściowego x. Sygnał x może być w takim przypadku pojedynczą liczbą lub całym wektorem liczb, np. wektorem reprezentacji znaczeniowej słowa z [15] lub [24]. Tak więc bieżący stan sieci jest uzyskiwany poprzez mnożenie wektora poprzedniego stanu pamięci oraz kolejnego słowa przez odpowiednio dobrane macierze/wektory parametrów U oraz W. Wczytywanie serii słów w kolejności, w której pojawiają się one w zdaniu oraz utrzymywanie stanu pamięci o poprzednich słowach przypomina sposób czytania tekstu



Rysunek 5: Sieć Vanilla-RNN rozwinięta w czasie

przez człowieka. Można mieć więc nadzieję, że model zdania, który wraz z kolejnymi słowami pojawia się w sieci w postaci stanu s będzie przypominał model tekstu, który formuje się w ludzkim mózgu podczas czytania.

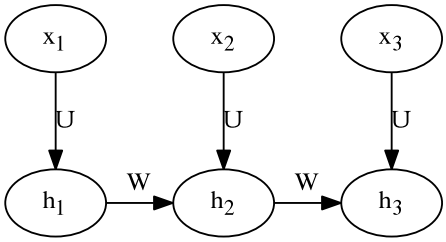
Elastyczna postać sieci Vanilla-RNN pozwala na jej liczne modyfikacje, np. wprowadzanie dodatkowych warstw pośrednich, dodawanie kolejnych sygnałów wejściowych oraz wyjściowych. Zmiany takie powodują niewielkie zwiększanie liczby koniecznych parametrów, ponieważ w każdym nowo wprowadzanym poziomie jest używany 1 dodatkowy parametr, ten sam dla każdego kroku czasowego.

5.2 Architektury Vanilla-RNN dla rozpoznawania parafraz

Niniejsza praca wprowadza i bada trzy nowe architektury oparte na Vanilla-RNN, z użyciem wersji podstawowej jako punktu odniesienia dla osiąganych wyników. Wersja podstawowa sieci Vanilla jest w dalszej części tekstu określona jako VRNN1-LIN, zaś nowe architektury mają następujące identyfikatory: VRNN1-TREE, VRNN2-LIN i VRNN2-TREE. Poniżej zamieszczony został opis wszystkich czterech architektur, które uczestniczą w testach.

1. (VRNN1-LIN) Vanilla-RNN w wersji podstawowej (Rys. 6)

   Procedura wyliczania pojedynczego stanu ukrytego (w tym przykładzie h_n) przebiega zgodnie z formułą:



Rysunek 6: Vanilla-RNN z w wersji podstawowej

$$h_n = \tanh(U \cdot x_n + W \cdot h_{n-1}) \tag{1}$$

Szczegółowy opis sekwencji operacji:

(a) Na wejściu pobierany jest wektor reprezentacji słowa x_n. Wektory wszystkich słów x_n mają jednakową długość - 50 lub 100 elementów.

(b) Wektor wejściowy x_n mnożony jest poprzez macierz U odpowiednich rozmiarów, zależnie od długości wektorów reprezentacji (50x50 lub 100x100).

(c) Podobnie wektor stanu poprzedniego h_{n-1} mnożony jest przez macierz W odpowiedniej wielkości równej wymiarom U.

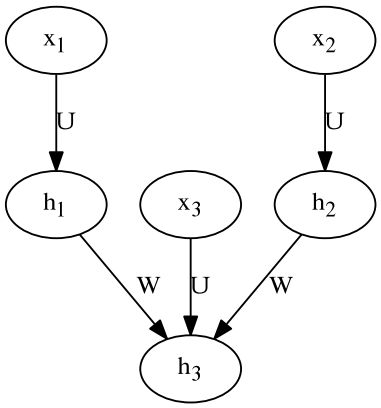
(d) Wektory otrzymane w punktach (b) i (c) są dodawane.

(e) Wszystkie elementy otrzymanego wektora są poddawane funkcji tanh. Wektor wynikowy to wektor h_n o długości równej wszystkim swoim wektorom wejściowym (x_n i h_{n-1}). Ostatni wektor h sekwencji czasowej zawiera reprezentację znaczenia całego zdania.

2. (VRNN1-TREE) Drzewiasta sieć Vanilla-RNN (Rys. 7)

Struktura sieci jest oparta na drzewie parsowania zależnościowego, podobnie jak w [13] i [16]. Sekwencja kolejności wczytywanych słów jest zaburzona poprzez wprowadzenie wczytywania nowych słów zgodnie ze strukturą drzewa, jednak można mieć nadzieję, że sensowna struktura gramatyczna drzewa przyniesie dodatkowy zysk. Co ciekawe, po rysunkowym rozwinięciu tego drzewa w czasie okazało się, że architektura ta przypomina architekturę przedstawioną w [13], bez części dekodującej.

Zaleta VRNN1-TREE w stosunku do [13] polega na braku wymogu binaryzacji drzewa.



Rysunek 7: Drzewiasta sieć Vanilla-RNN

Procedura wyliczania pojedynczego stanu ukrytego h_n o pojedynczym słowie w węźle oraz m poprzedzających stanów ukrytych:

$$h_n = \tanh(U \cdot x_n + \sum_{i=1}^{m} W \cdot h_{n-1,m}) \tag{2}$$

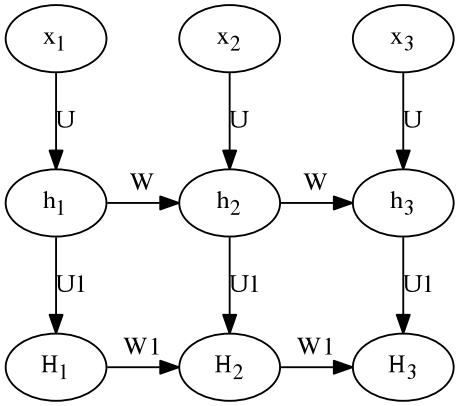
3. (VRNN2-LIN) Vanilla-RNN z dodatkową warstwą ukrytą (Rys. 8).

   Ta sieć składa się z klasycznego łańcucha z dodaną sekwencją stanów ukrytych. Wektor dodatkowego stanu ukrytego H obliczany jest poprzez nałożenie nieliniowości (funkcji tangensa hiperbolicznego) na konkatenację stanu h z wektorem dodatkowego stanu ukrytego dla poprzedniego momentu w czasie. Wielokrotne nakładanie nieliniowości pozwala na modelowanie bardziej skomplikowanych funkcji. W ten sposób ucząca się sieć może potencjalnie zakodować bardziej skomplikowane zależności, które nie zostały umieszczone w warstwie stanów h. Kolejność czasowa wczytywania słów nie jest w żaden sposób zaburzona.

   Procedura wyliczania pojedynczego stanu ukrytego h_n oraz wyższego stanu ukrytego H_n:

$$h_n = \tanh(U \cdot x_n + W \cdot h_{n-1}) \tag{3}$$

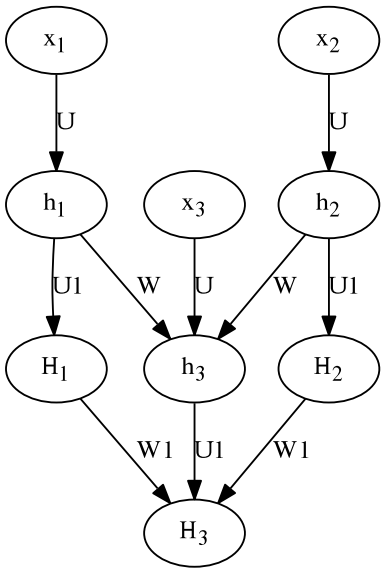
$$H_n = \tanh(U1 \cdot h_n + W1 \cdot H_{n-1}) \tag{4}$$



Rysunek 8: Vanilla-RNN z dodatkową warstwą ukrytą

4. (VRNN2-TREE) Drzewiasta sieć Vanilla-RNN z dodatkową warstwą ukrytą (Rys. 9)

   Połączenie koncepcji sieci z punktu 2 i 3.



Rysunek 9: Drzewiasta sieć Vanilla-RNN z dodatkową warstwą ukrytą

Procedura wyliczania pojedynczego stanu ukrytego h_n oraz wyższego stanu ukrytego H_n o pojedynczym słowie w węźle oraz m poprzedzają-

cych stanów ukrytych:

$$h_n = \tanh(U \cdot x_n + \sum_{i=1}^{m} W \cdot h_{n-1,m}) \tag{5}$$

$$H_n = \tanh(U1 \cdot h_n + \sum_{i=1}^{m} W1 \cdot H_{n-1,m}) \tag{6}$$

5.3 Pełna architektura sieci

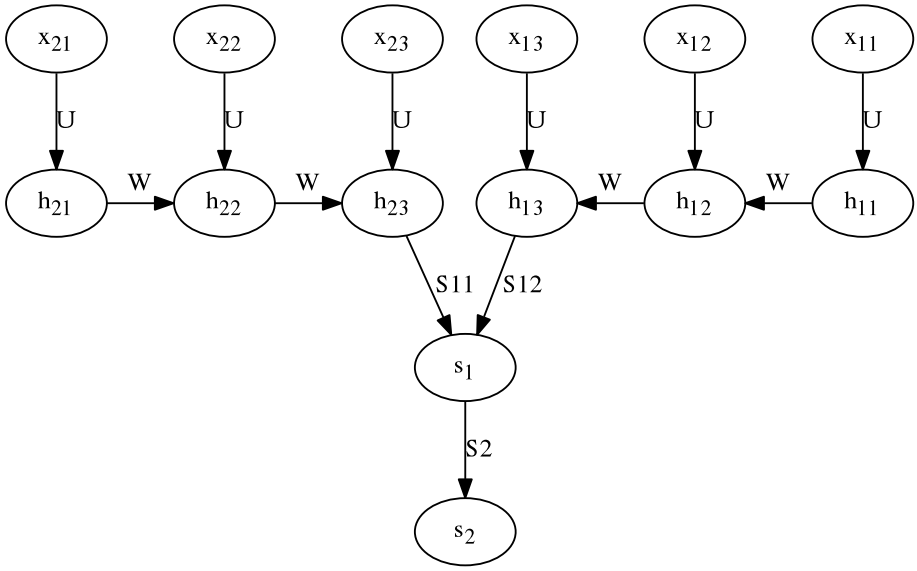
Aby sklasyfikować parę zdań, sieć potrzebuje warstwy prezentującej finalny wynik, na przykład warstwy typu *softmax* [25], przyjętej w niniejszym rozwiązaniu. Warstwa *softmax* składa się z 6 wyjściowych neuronów, które oznaczają dany wynik klasyfikacji na możliwej skali 0-5 (lub 8 neuronów na skali zadania *Interpretable STS*, jednak w dalszych rozważaniach dla uproszczenia przyjmuję liczbę 6). W każdym z neuronów wyjściowych warstwy *softmax* zwracane jest prawdopodobnieństwo wystąpienia danego wyniku. W najprostszym podejściu do problemu wynikiem ostatecznym klasyfikacji jest numer neuronu z największym prawdopodobieństwem, choć można brać także pod uwagę inne neurony o niezerowym prawdopodobieństwie (patrz Rozdział 5.5).

Pełna architektura sieci łączy wyjścia podsieci dla 2 zdań z finalną warstwą *softmax* (Rys. 10). Dla sieci z jedną warstwą ukrytą (VRNN1 - tylko wektory h) węzłem reprezentującym całe zdanie jest ostatni (wg kolejności czasowej lub najwyższy z węzłów drzewa zależnościowego) wektor h. Dla sieci z 2 warstawmi ukrytymi (VRNN2 - wektory h i H) wektorem reprezentującym zdanie jest ostatni wektor H (Rys. 11).

Węzły s_1 i s_2 na Rys. 10 i Rys. 11 służą uzyskaniu finalnego wyniku klasyfikacji. Węzeł s_2 reprezentuje warstwę *softmax*, natomiast węzeł s_1 służy do kompresji wektorów wyjściowych z podsieci obu zdań do rozmiarów warstwy *softmax*. Ponieważ warstwa *softmax* jest wektorem o długości 6, zaś wektory wyjściowe z podsieci reprezentacji 2 zdań mają długość 50 lub 100 elementów, wymiary macierzy parametrów *S11* oraz *S12* muszą wynosić odpowiednio 50x50 lub 100x100, zaś macierzy *S2* - 6x50 lub 6x100, aby w węźle s_2 otrzymać wektor o wymaganej długości 6 elementów zgodnie z następującymi równaniami (oznaczenia zgodne z przykładowymi stanami s_1 i s_2 z Rys. 10)

$$s_1 = \tanh(S11 \cdot h_{23} + S12 \cdot h_{13}) \tag{7}$$

$$s_2 = softmax(S2 \cdot s_1) \tag{8}$$



Rysunek 10: Pełna sieć VRNN1-LIN do klasyfikacji parafraz

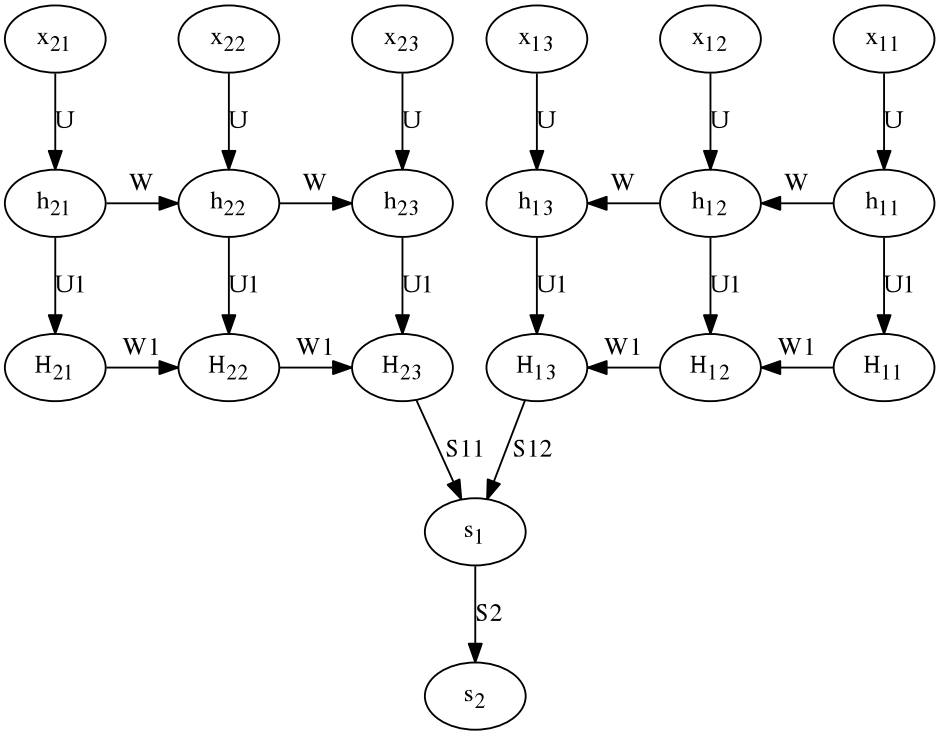
5.4 Trenowanie

Trenowanie prowadzone było z użyciem klasycznego algorytmu propagacji wstecznej. Dla nowych architektur sieci konieczne było wyprowadzenie wzorów aktualizacji gradientów dla tego algorytmu. Wzory uzyskałam zgodnie z regułą łańcuchową pochodnych funkcji złożonych (ang. *chain rule*).

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx} \tag{9}$$

Wzory aktualizacji gradientów wyprowadziłam dla macierzy parametrów: *W*, *U*, *W1*, *U1*. Wyprowadzenie sprowadzało się do obliczenia pochodnych, określających wpływ wartości w danej macierzy na wartość funkcji kosztu (a więc wielkość błędu popełnianego przez sieć).

$$\frac{dJ}{dW} = \frac{dJ}{dh} \cdot \frac{dh}{dW} \tag{10}$$



Rysunek 11: Pełna sieć VRNN2-LIN do klasyfikacji parafraz

5.4.1 Funkcja kosztu

Jako funkcję kosztu wykorzystałam funkcję *log-loss*, która dla dwóch dyskretnych rozkładów prawdopodobieństwa p i q przyjmuje postać:

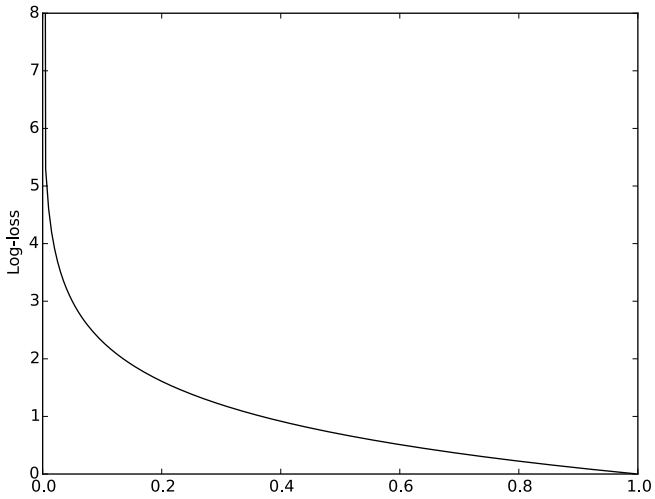
$$J(p, q) = -\sum_{x} p(x) \log q(x) \tag{11}$$

Jako rozkład q przyjmowany był wynik działania klasyfikatora, natomiast jako rozkład p - poprawny wynik docelowy. Funkcja *log-loss* jest odpowiednią funkcją kosztu dla opisywanego problemu, ponieważ w przypadku warstwy *softmax* mamy do czynienia z pełnym rozkładem prawdopodobieństw uzyskania poszczególnych klas, a nie tylko zwróceniem identyfikatora najbardziej prawdopodobnej klasy. Funkcja *log-loss* korzysta z pojęcia entropii i silnie karze przypadki, gdy klasyfikator z dużą pewnością dokonuje błędnej oceny. Przypadki wyboru poprawnej klasy z mniejszą pewnością otrzymują jedynie nieznaczną karę. Na Rys. 12 przedstawiony został wykres funkcji dla przypadku binarnej klasyfikacji. Na osi poziomej widoczna jest pewność klasyfikatora (wartości poniżej 0,5 to wybór klasy zero, zaś powyżej - wybór klasy 1), na osi pionowej - wartość kary. Poprawnym identyfikatorem jest klasa 1. Uwidoczniony został szybki wzrost kary dla błędu popełnianego w prawdopodobieństwach poprawnego wyniku poniżej 0.25.

5.4.2 Gradienty propagacji wstecznej

W równaniach (7) - (13) zaprezentowałam podstawowe wzory używane w procedurze propagacji wstecznej. Przyjęto oznaczenie rodzica węzła jako przedrostek *parent*. Rodzicem określany jest węzeł wyższy (słowo późniejsze w sekwencji czasowej lub umieszczone wyżej w drzewie) z którego propagowany jest błąd do węzła obecnie rozważanego. Dla węzłów posiadających więcej niż 1 rodzica (np. węzły h w architekturze VRNN2) konieczne było sumowanie błędów pochodzących od wszystkich rodziców. Przyjęto oznaczenia *delta* (standardowo dla wyrażenia δ) i *grad* (dla wyrażenia gradientu aktualizacji danej macierzy parametrów). Nie wyszczególniłam wzorów propagacji dla warstw s_1 i s_2, ponieważ są to wzory standardowej sieci. Wzory, których brakuje w opisie VRNN2 są identyczne jak w VRNN1.

1. VRNN1



Rysunek 12: Wykres funkcji *log-loss*

$$delta_h = \tanh'(h \cdot (W^T \cdot parentDelta_h)) \tag{12}$$

$$grad_W = parentDelta_h \cdot h^T \tag{13}$$

$$grad_U = parentDelta_h \cdot x^T \tag{14}$$

2. VRNN2

$$delta_H = \tanh'(H \cdot (W1^T \cdot parentDelta_H)) \tag{15}$$

$$delta_h = \tanh'(h * (W^T \cdot parentDelta_h + U1^T \cdot delta_H)) \tag{16}$$

$$grad_{W1} = parentDelta_H \cdot H^T \tag{17}$$

$$grad_{U1} = delta_H \cdot h^T \tag{18}$$

Do obliczania gradientu w trenowanych zmiennych używany jest algorytm L-BFGS (ang. *Limited-memory BFGS*) [26]. Algorytm ten automatycznie ustala długość kroku gradientu, eliminując konieczność ręcznego ustalania współczynników tempa uczenia.

5.4.3 Wektory reprezentacji słów

Jako wektory reprezentacji słów używane są wektory udostępnione przez [15], o długości 50 i 100 elementów. Rozważałam użycie wektorów udostępnionych przez [24], ponieważ zgodnie z publikacją wektory te powodują podniesienie wyników niektórych algorytmów. Niestety, ich zasadniczą wadą jest brak wektora słowa nieznanego (w [15] jest to wektor *UNKNOWN*). Wektor ten można obliczyć za pomocą różnych metod aproksymacji na podstawie wektorów pozostałych słów (np. uśrednienie wszystkich wektorów lub uśrednienie określonej liczby wektorów najrzadszych słów). Jednak są to pomysły nie gwarantujące otrzymania sensownych reprezentacji. Według komentarzy autorów [24], wytrenowany na danych wektor *UNKNOWN* ma się pojawić w przyszłości, jednak w czasie pisania tej pracy nie był on jeszcze dostępny.

5.4.4 Regularyzacja

Regularyzacja sieci neuronowej służy wprowadzeniu kar dla dużych wartości parametrów w macierzach wag. Odpowiednio dobrana regularyzacja promuje macierze o wyrównanej skali wartości parametrów. Intuicyjnie oznacza to, że sieć jest motywowana do trenowania z równą intensywnością wszystkich wag macierzy, zamiast wykorzystywać tylko określony ich podzbiór.

Jako współczynnik regularyzacji wykorzystałam w tej pracy często używaną metodę *L2*. Według tej metody, do każdego elementu macierzy wag trenowanej w sieci (*W*, *U*, *W1*, *U1*) dodawany jest współczynnik kary, liczony oddzielnie dla każdego elementu macierzy w, gdzie współczynnik λ wyznacza siłę regularyzacji:

$$k = \frac{1}{2}\lambda w^2 \tag{19}$$

Ten wzór regularyzacji pozwala na uzyskanie prostego wzoru aktualizacji danej macierzy W po wyciągnięciu pochodnej:

$$W_{reg} = W - \lambda \cdot W \tag{20}$$

Zgodnie z tym wzorem, w regularyzacji *L2* mamy do czynienia z liniowym zmniejszaniem wag w stronę wartości 0.

5.5 Rozwiązanie problemu ułamkowych anotacji

Często spotykane liczby ułamkowe w anotacjach zbiorów SemEval wymuszają podjęcie decyzji co do ich reprezentacji w warstwie docelowej *softmax*. W niniejszej pracy proponuję 2 podejścia:

1. Zaokrąglanie części ułamkowej

   W przypadku napotkania liczby ułamkowej jest ona zaokrąglana do najbliższej liczby całkowitej. Np. 4.75 jest zaokrąglane do 5 i rozkład liczb w warstwie *softmax* wygląda następująco: (0 0 0 0 0 1), czyli wynik 5 jest przyjmowany z prawdopodobieństwem równym 1.0.

   Uzasadnieniem takiego działania może być fakt, że oceniający pary zdań lingwiści mogli popełniać błędy, stąd oceny np. 4.75, gdzie większość oceniających wystawiła ocenę 5 a jedna w wyniku błędu - ocenę 4. Zaokrąglanie powoduje, że mniejszość głosów jest traktowana jako pomyłka i w całości odrzucana.

2. Wyważenie głosów

   W przypadku napotkania liczby ułamkowej następuje próba odtworzenia warunków głosowania. Na przykład ocena 4.75 oznacza, że w głosowaniu wzięły udział 4 osoby i uzyskano 3 głosy na ocenę "5"i 1 głos na ocenę „4" - czyli warstwa docelowa *softmax* przyjmuje postać (0 0 0 0 0.25 0.75), a więc wynik 5 jest przyjmowany z prawdopodobieństwem 0.75 a wynik 4 - z prawdopodobieństwem 0.25. Ta wersja zamiast odrzucać, bierze pod uwagę również mniejszości głosów.

Oba podejścia są prezentowane w sekcji wyników celem porównania ich efektywności.

5.6 Szczegóły implementacji

Kod rozwiązania został napisany w języku Java. Język ten został wybrałam, ponieważ pozwala na bezpośrednie użycie struktur drzewa parsowania zależnościowego z parsera Stanforda oraz innych użytecznych metod z biblioteki

Stanford Core NLP [27]. Pewną niedogodnością języka Java dla implementacji sieci neuronowej jest brak wbudowanego wsparcia dla operacji macierzowych, co jednak ułatwiła biblioteka Jblas [28]. Poniżej szczegółowy opis wykorzystanych funkcjonalności bibliotek:

- Stanford Core NLP

  Wykorzystany został parser zależnościowy Stanforda oraz struktury danych ułatwiające operacje na drzewach parsowania, takie jak graf zależności (obiekty klasy *SemanticGraph*) oraz reprezentacja drzewa (obiekty klasy *Tree*). Pośrednio używane są różne inne elementy łańcucha przetwarzania Core NLP (tokenizator, lematyzator itd.), ponieważ wymaga tego proces parsowania. Wykorzystałam również gotową implementację algorytmu L-BFGS z klasy *QNMinimizer*.

- Jblas

  Biblioteka zawiera struktury danych oraz metody ułatwiające operacje na macierzach. Wykorzystałam klasę *DoubleMatrix* do tworzenia obiektów macierzy oraz przeprowadzenia niezbędnych operacji takich jak transpozycja, mnożenie czy dodawanie macierzy.

Kod sieci VRNN został napisany ręcznie, bez użycia gotowych bibliotek do implementacji sieci neuronowych. Jest to łatwiejsze rozwiązanie dla implementacji własnej architektury sieci.

Jedynym źródłem danych wejściowych są gotowe wektory reprezentacji słów, udostępnione przez [15], oraz zbiory danych testowych i trenujących. Na wyjściu programu zwracane są wyniki jakości działania poszczególnych architektur, wyrażone wynikiem korelacji Pearsona.

Działanie programu rozpoczyna się od wczytania zbioru trenującego i testowego w postaci par zdań z ocenami oraz ich gotowych drzew zależnościowych, które są przechowywane w plikach. Dostępna jest także opcja stworzenia i zapisania nowych drzew. Następnie budowane są obiekty reprezentujące poszczególne architektury VRNN i rozpoczyna się trenowanie sieci. Po zakończeniu ustalonej liczby iteracji, gotowe modele wykorzystywane są do przeprowadzenia testów na zbiorze danych testowych.

Do sprawdzenia poprawności implementacji wykorzystałam metodę sprawdzania gradientu (ang. *gradient checking*) [25]. Metoda ta porównuje wyliczony w algorytmie propagacji wstecznej gradient z idealnym gradientem

otrzymanym metodą numeryczną. Gradient numeryczny otrzymywany jest poprzez obliczenie różnicy pomiędzy wartością funkcji kosztu dla wybranej macierzy W, której elementy zostały zwiększone oraz zmniejszone o wybraną, bardzo małą wartość ϵ.

$$grad \approx \frac{J(W + \epsilon) - J(W - \epsilon)}{2 \cdot \epsilon} \tag{21}$$

Różnica pomiędzy gradientem aproksymowanym a faktycznym powinna być jak najmniejsza, w praktyce utrzymanie różnicy na poziomie mniejszym niż 10^{-10} gwarantuje poprawność implementacji. Sprawdzenie należy przeprowadzić dla wszystkich macierzy trenowanych w danej sieci (w niniejszej pracy są to macierze: W, U, $W1$, $U1$).

6 Wyniki testów i analiza

Testy przeprowadziłam dla następujących zmiennych:

1. liczba iteracji algorytmu L-BFGS
2. długość wektorów reprezentacji
3. rodzaj postępowania z ułamkowymi anotacjami

Jako miara wydajności algorytmu wykorzystany został współczynnik korelacji Pearsona pomiędzy poprawnymi wynikami klasyfikacji a tymi zwróconymi przez algorytm. Współczynnik ten został wybrany ponieważ pozwala na wystąpienie wartości niecałkowitych w wartościach docelowych. Standardowe miary takie jak dokładność czy czułość wymagałyby zaokrąglania wyników, co jest formą utraty informacji. Ponadto korelacja Pearsona jest przyjęta jako miara jakości w konkursie SemEval, co pozwala na porównanie proponowanych algorytmów z innymi publikacjami.

Choć autorzy publikacji wykorzystujących zbiory SemEval często ręcznie dobierają zbiory do trenowania modeli (np. modele trenowane są na wszystkich zbiorach z poprzednich lat, lub wybranej części najbardziej podobnej gramatycznie i/lub leksykalnie do danego zbioru testowego), jak pokazano w [6], [7], [8], to w niniejszej pracy dla każdego zbioru testującego stosowałam tylko odpowiadający zbiór treningowy (czyli dla zbioru testującego *MSRpar* model był trenowany tylko na zbiorze trenującym *MSRpar* itd.), tak by ograniczyć do minimum wpływ ludzkiego poziomu wiedzy lingwistycznej.

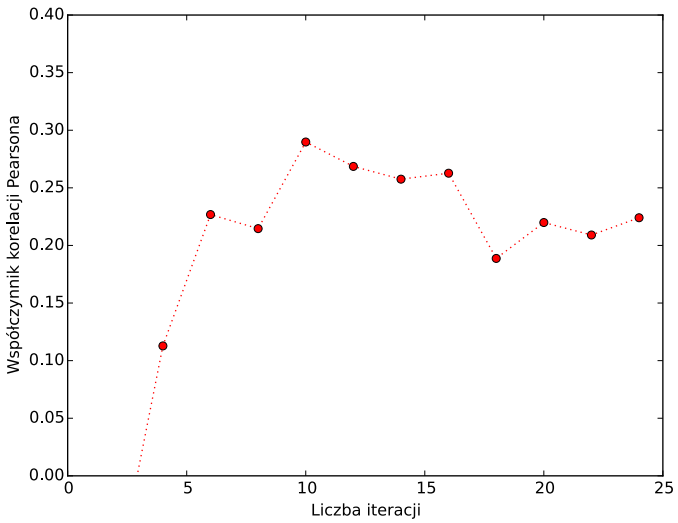
6.1 Liczba iteracji

Testy były przeprowadzane na określonym zakresie wartości iteracji (2-25) lub (2-45) algorytmu L-BFGS. Dla konkretnej liczby iteracji przeprowadziłam 3 testy, których wyniki korelacji Pearsona były następnie uśredniane. Wszystkie wyniki (poza tymi zawartymi w Rozdziale 6.3) uzyskałam korzystając z algorytmu wyważania głosów (patrz Rozdział 5.5) dla rozwiązania problemu ułamkowych anotacji (w drodze pierwszych testów zauważyłam lepsze wyniki tego algorytmu, co potwierdziły kolejne, obszerne testy).

1. Zbiór Zadanie1:MSRvid

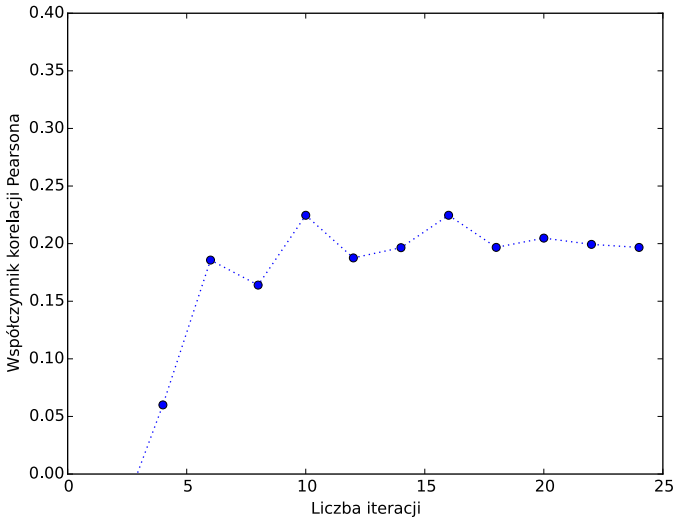
   Najlepszym z algorytmów na tym zbiorze okazał się VRNN1-LIN. Wszystkie algorytmy uzyskiwały maksymalne wyniki w okolicy 10-12 iteracji (punkt przegięcia wykresów), po czym następował spadek dokładności, prawdopodobnie spowodowany przetrenowaniem. Mimo otrzymania przez VRNN1-LIN najlepszego wyniku maksymalnego, był on też najbardziej wrażliwy na przetrenowanie. Różnica pomiędzy wartością maksymalną a wartościami uzyskiwanymi dla ponad 20 iteracji jest tu największa spośród wszystkich 4 algorytmów. Otrzymane wartości maksymalne korelacji Pearsona:

   - VRNN1-LIN: 0,29 (Rys 13)
   - VRNN1-TREE: 0,23 (Rys 14)
   - VRNN2-LIN: 0,25 (Rys 15)
   - VRNN2-TREE: 0,24 (Rys 16)

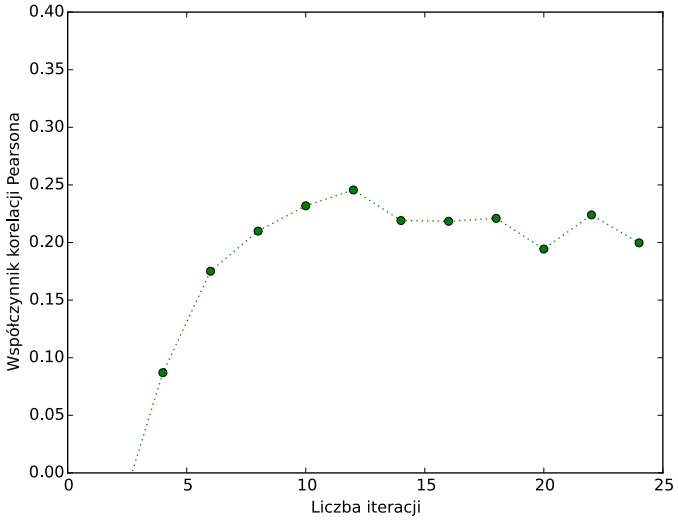


Rysunek 13: VRNN1-LIN na zbiorze MSRvid - test liczby iteracji

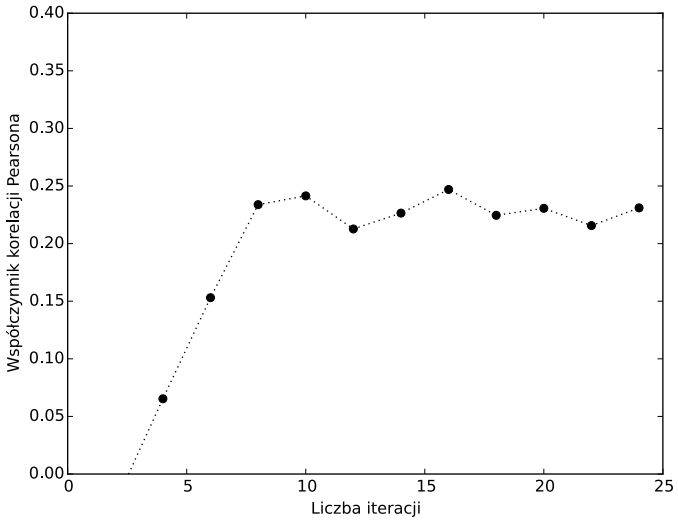
2. Zbiór Zadanie1:MSRpar



Rysunek 14: VRNN1-TREE na zbiorze MSRvid - test liczby iteracji



Rysunek 15: VRNN2-LIN na zbiorze MSRvid - test liczby iteracji

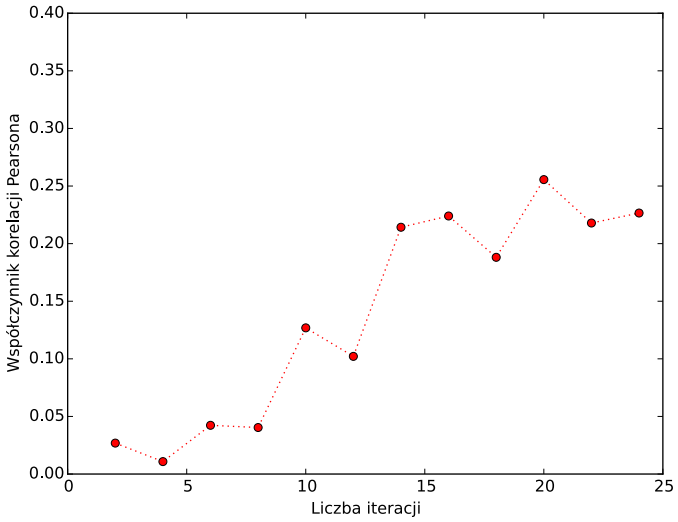


Rysunek 16: VRNN2-TREE na zbiorze MSRvid - test liczby iteracji

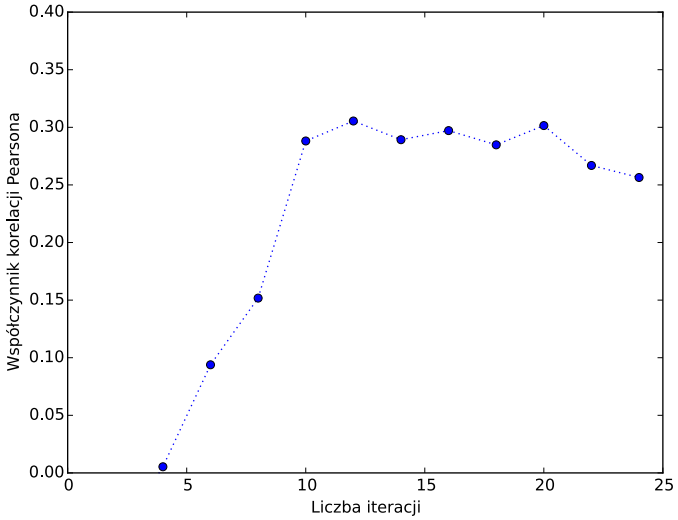
Zbiór ten zawiera znacznie trudniejsze gramatycznie zdania niż zbiór MSRvid, w związku z tym można zaobserwować przesunięcie maksimum jakości wyników w stronę większej liczby iteracji oraz najlepsze wyniki pojawiające się w najbardziej skomplikowanej architekturze: VRNN2-TREE. Mimo stopnia skomplikowania gramatyki i słownictwa zdań algorytmy poradziły sobie lepiej niż z poprzednim zbiorem. Może to być związane z pomyłkami w odległościach wektorów pokrewnych słów (opisane w Rozdziale 6.4) które mają większy wpływ w przypadku analizy krótkich zdań (mamy wtedy do czynienia z niewielką liczbą wektorów i pomyłka w porównaniu którejś z par ma większe znaczenie).

- VRNN1-LIN: 0,26 (Rys 17)
- VRNN1-TREE: 0,30 (Rys 18)
- VRNN2-LIN: 0,28 (Rys 19)
- VRNN2-TREE: 0,33 (Rys 20)

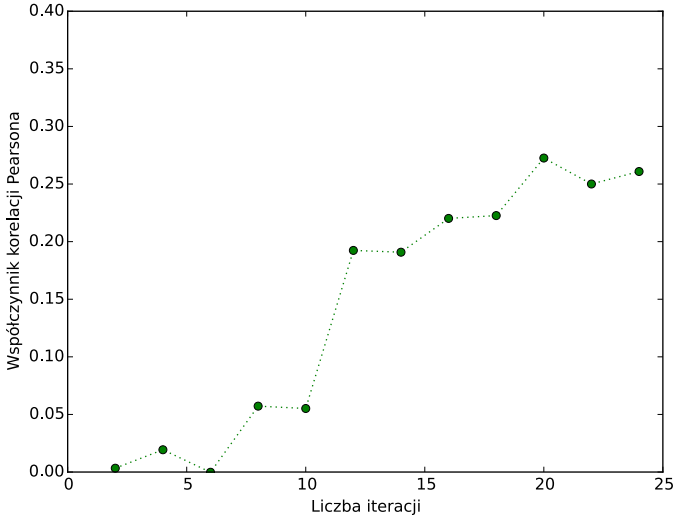
3. Zbiór Zadanie2:images



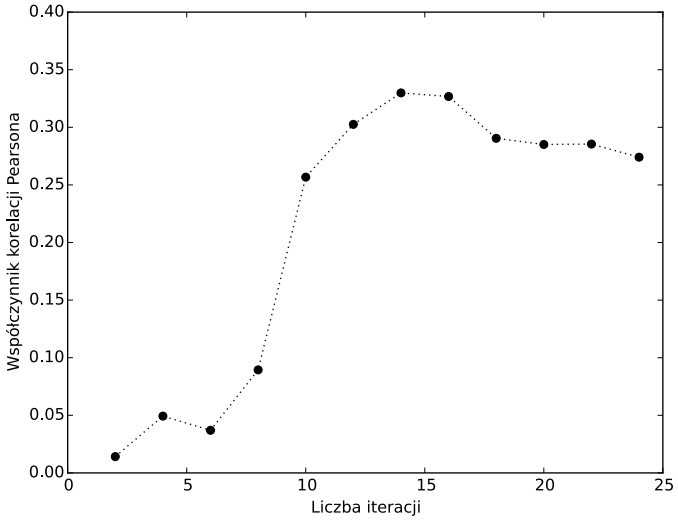
Rysunek 17: VRNN1-LIN na zbiorze MSRpar - test liczby iteracji



Rysunek 18: VRNN1-TREE na zbiorze MSRpar - test liczby iteracji



Rysunek 19: VRNN1-TREE na zbiorze MSRpar - test liczby iteracji



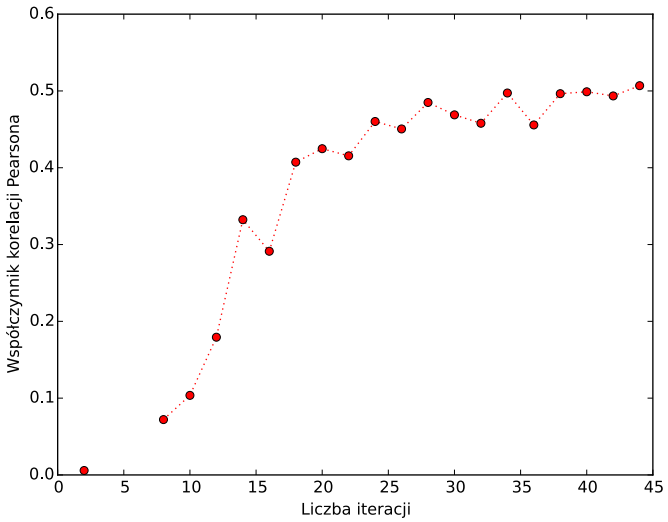
Rysunek 20: VRNN2-TREE na zbiorze MSRpar - test liczby iteracji

Zbiór Zadania 2 (Interpretable STS) zawiera inne wyzwania niż zbiory Zadania 1. Frazy są krótsze, a więc potencjalnie łatwiejsze do oceny, jednak mamy do czynienia z bogatszym systemem wartości docelowych (wartości OPPO, EQUI, SPE1, SPE2, SIM, REL, ALIC, NOALI zamiast skali 0-5). Wartości te w bardziej szczegółowy sposób opisują podobieństwo niż prosta skala liczbowa. Pomimo to, wyniki na tym zbiorze są znacząco lepsze niż na zbiorach Zadania 1.

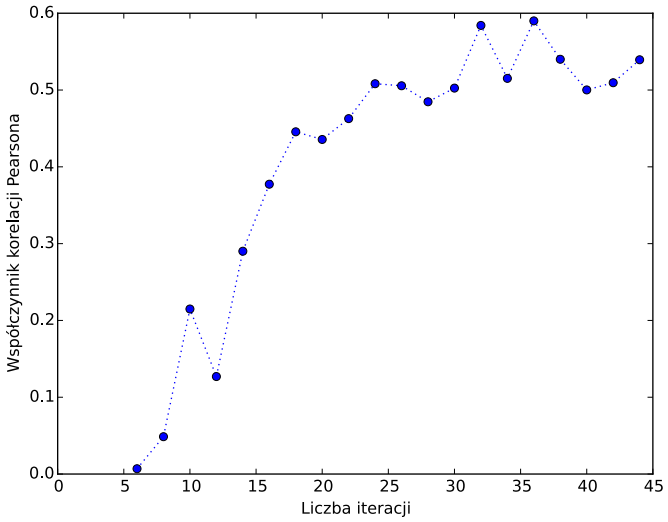
Na zbiorach Zadania 2 nie zaobserwowałam efektów przetrenowania widocznych w wynikach Zadania 1. Wręcz przeciwnie, mimo zwiększenia maksymalnego zakresu iteracji z 25 do 45, algorytmy wciąż poprawiały swoje wyniki, lub przynajmniej utrzymywały je na mniej więcej stałym poziomie, aż do osiągnięcia maksymalnej ilości iteracji.

Otrzymane wartości maksymalne korelacji Pearsona:

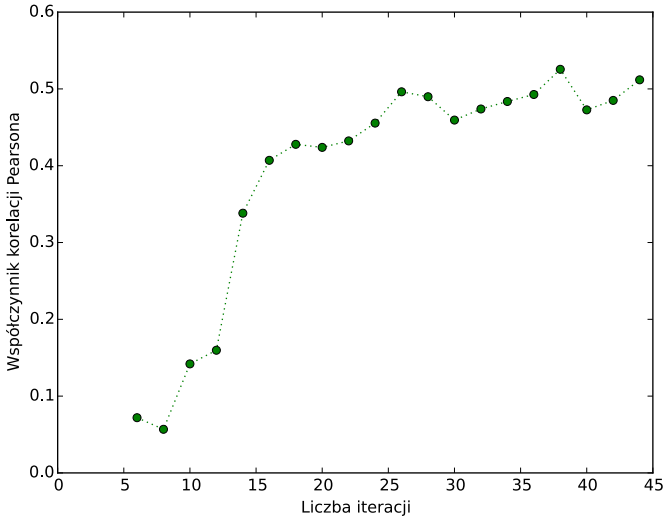
- VRNN1-LIN: 0,50 (Rys 21)
- VRNN1-TREE: 0,59 (Rys 22)
- VRNN2-LIN: 0,53 (Rys 23)
- VRNN2-TREE: 0,55 (Rys 24)



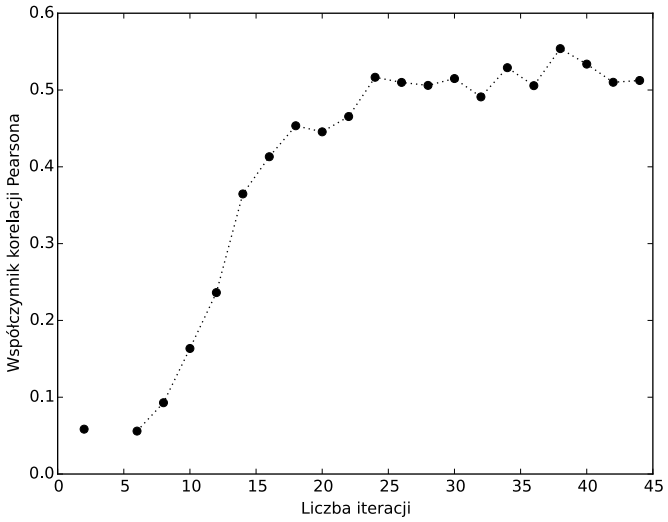
Rysunek 21: VRNN1-LIN na zbiorze Images - test liczby iteracji



Rysunek 22: VRNN1-TREE na zbiorze Images - test liczby iteracji



Rysunek 23: VRNN2-LIN na zbiorze Images - test liczby iteracji



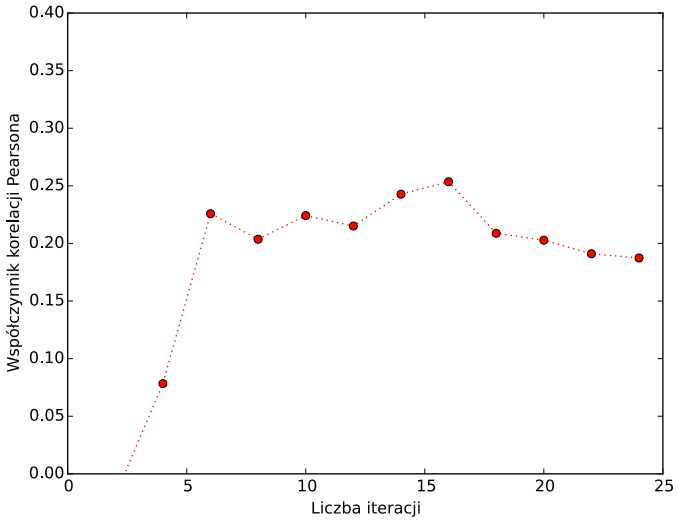
Rysunek 24: VRNN2-TREE na zbiorze Images - test liczby iteracji

6.2 Długość wektorów reprezentacji

Wyniki dla długości wektorów równej 50 z poprzedniego podrozdziału (na zbiorze MSRvid) zostały porównane z wynikami na tym samym zbiorze z użyciem wektorów o długości 100. Mimo, jak mogłoby się wydawać, zawarcia w dłuższych wektorach większej ilości użytecznych informacji wyniki nie uległy znaczącemu polepszeniu, a w przypadku sieci VRNN1-LIN - wręcz lekkiemu pogorszeniu. Dla większej długości wektorów najlepszą z architektur okazała się być VRNN2-Tree. Uwidoczniło się przesunięcie punktu przegięcia wykresów w okolice 15-16 iteracji. Jest to zgodne z intuicją - dłuższe wektory inicjalizowane losowymi wartościami wewnątrz sieci potrzebowały więcej iteracji, by osiągnąć maksymalną wydajność. Podobnie jak w przypadku wektorów o długości 50, dla 100 architektury używające drzewa zależnościowego nie wykazywały znaczącej wrażliwości na przetrenowanie. Wykresy tych architektur są wyrównane od osiągnięcia punktu najlepszego wytrenowania do końca badanej skali iteracji. Wykresy architektur liniowych dla dużych wartości iteracji stopniowo opadają.

- VRNN1-LIN: 0,25 (Rys 25)
- VRNN1-TREE: 0,23 (Rys 26)

- VRNN2-LIN: 0,24 (Rys 27)
- VRNN2-TREE: 0,26 (Rys 28)

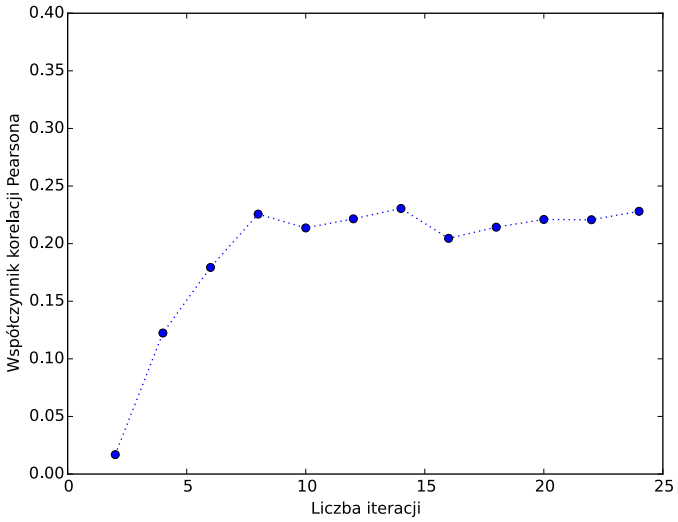


Rysunek 25: VRNN1-LIN - test długości wektorów reprezentacji

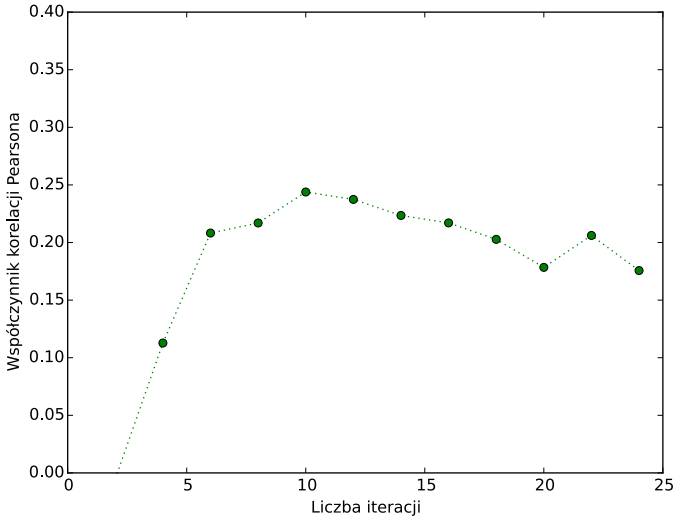
6.3 Ułamkowe anotacje

Poniżej zamieściłam wyniki dla zbioru MSRvid z użyciem algorytmu zaokrąglania. W porównaniu do wyników zbioru MSRvid z Rozdziału 6.1, otrzymanych z użyciem algorytmu wyważania, wyniki poniższe są nieco gorsze lub w najlepszym wypadku równe. Z tego powodu we wszystkich innych testach był używany algorytm wyważania, który najwyraźniej lepiej dopasowuje się do problemu ułamkowych anotacji.

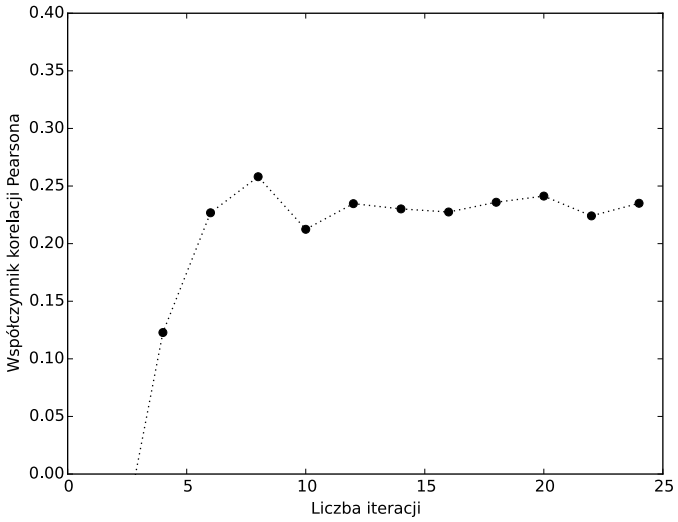
- VRNN1-LIN: 0,28 (Rys 29)
- VRNN1-TREE: 0,22 (Rys 30)
- VRNN2-LIN: 0,25 (Rys 31)
- VRNN2-TREE: 0,23 (Rys 32)



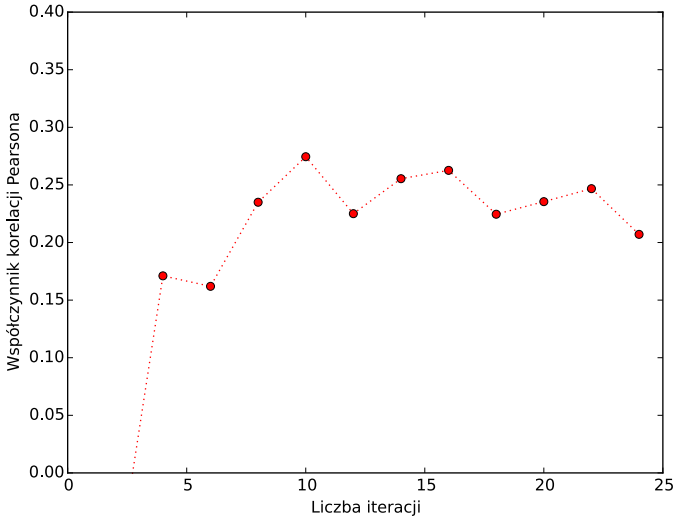
Rysunek 26: VRNN1-TREE - test długości wektorów reprezentacji



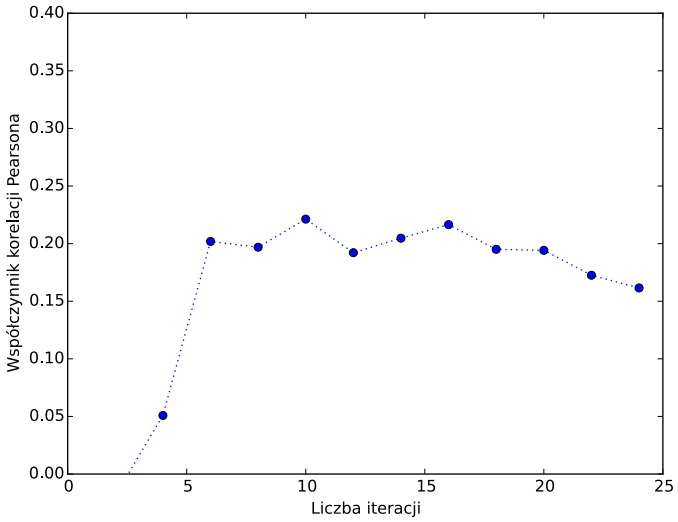
Rysunek 27: VRNN2-LIN - test długości wektorów reprezentacji



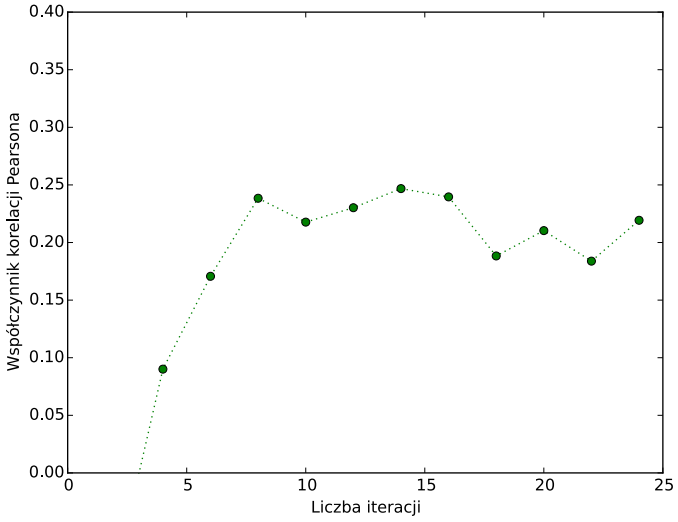
Rysunek 28: VRNN2-TREE - test długości wektorów reprezentacji



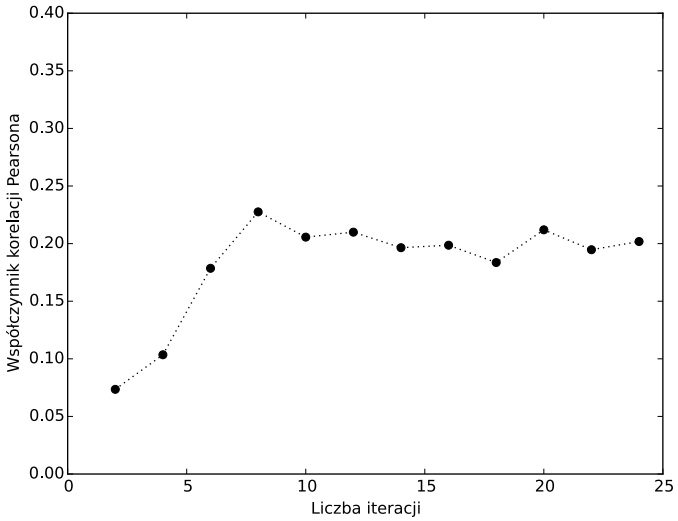
Rysunek 29: VRNN1-LIN - test ułamkowych reprezentacji



Rysunek 30: VRNN1-TREE - test ułamkowych reprezentacji



Rysunek 31: VRNN2-LIN - test ułamkowych reprezentacji



Rysunek 32: VRNN2-TREE - test ułamkowych reprezentacji

6.4 Problemy wektorów reprezentacji

Podczas testów zaobserwowałam pojedyncze przypadki dużych pomyłek algorytmu w ocenianiu zdań o niemal identycznej treści i strukturze. Pomyłki te częściowo wynikają z faktu, że wektory reprezentacji niektórych podobnych znaczeniowo słów są rozłożone dość daleko od siebie w przestrzeni. Trudno wytłumaczyć, dlaczego tak się dzieje. Najwyraźniej mimo podobnego znaczenia słowa te pojawiają się w bardzo różnych kontekstach.

Podobnie, wektory słów nie będących synonimami, a należące do np. tej samej klasy pojęć (np. nazwy dni tygodnia, gatunków zwierząt, krajów) są niejednokrotnie zlokalizowane zbyt blisko siebie, a przecież w wykorzystywanych zbiorach danych różnica między wykonawcą czynności którym jest *cat* lub *dog* jest różnicą decydującą o wyniku 0 lub 5. W rozpoznawaniu parafraz nie jest istotne, że oba pojęcia dotyczą zwierząt. Wektory reprezentacji słów nie dokonują rozróżnienia tych klas podobieństwa, pokazują jedynie relację częstego wystąpienia pary słów w podobnym kontekście. Dlatego nie są one idealną reprezentacją dla celów rozpoznawania parafraz.

Podobne obserwacje poczyniono w [29], gdzie na zamieszczonych wizualizacjach można zobaczyć ułożenie pokrewnych fraz w przestrzeni. Niestety,

Zadanie	MSRvid	MSRpar	images	Średnia
VRNN1-LIN	**0,29**	0,26	0,50	0,35
VRNN1-TREE	0,23	0,30	**0,59**	0,36
VRNN2-LIN	0,25	0,28	0,53	0,35
VRNN2-TREE	0,24	**0,33**	0,55	**0,37**

Tablica 2: Podsumowanie wyników testów.

potwierdza to obserwację, że pokrewieństwo fraz jest stwierdzane w pojedynczej płaszczyźnie na wszystkich możliwych poziomach na raz (nie tylko na poziomie semantycznym). Tak więc wyrażenia czasowe *a few seconds* i *a few minutes* znajdują się bardzo blisko siebie, mimo tego, że w zdaniu nie można podmienić jednego z nich na drugie bez zasadniczej zmiany znaczenia.

6.5 Podsumowanie testów

Wyniki testów podsumowałam w Tablicy 2. Wybrane zostały najlepsze wyniki. Dodana została także kolumna zawierająca średnią arytmetyczną dla wszystkich wyników danej architektury.

Prezentowane algorytmy sprawdzają się szczególnie dobrze w zadaniu dopasowywania fraz (Zadanie 2). Wyniki przez nie osiągane można porównywać z wynikami systemu regułowego NeRoSim, który wygrał zadanie SemEval Interpretable STS w roku 2015. Opis systemu znajduje się w [30], gdzie przedstawiono wyniki działania systemu dla 3 podzadań: odnalezienia pasujących par fraz (zadanie A), przypisanie etykiet (EQUI, NOALI i pozostałe) do par fraz (zadanie T) i przypisanie wyników w skali 0-5 (zadanie S). Aby porównać wynik systemu do jakości działania sieci testowanych w tej pracy należy wrócić uwagę na wynik zadania T, gdzie system NeRoSim osiągnął maksymalny wynik 0.614 korelacji Pearsona. Trzeba przy tym pamiętać, że wynik ten osiągnięto dzięki szczegółowej inżynierii cech budowanych przez ludzi. Architektury sieci VRNN1-TREE i VRNN2-TREE osiągnęły wyniki dochodzące do 0.6 korelacji Pearsona, będąc systemami w pełni automatycznymi. Pozostałe systemy konkursowe SemEval na zbiorze Images osiągnęły wyniki od ok. 0.22 do 0.609 korelacji Pearsona [11].

Na podstawie dobrych wyników uzyskanych dla Zadania 2, które są porównywalne z najlepszymi systemami występującymi w SemEval 2015, można

rozważyć zbudowanie konkursowych systemów opartych na prezentowanych sieciach. Systemy te, wzbogacone o dodatkowe reguły czy cechy, mogłyby z powodzeniem wystąpić w konkursie.

Gorsze wyniki osiągane były na zbiorach Zadania 1. Najlepszy wynik uzyskany dla MSRpar przez badane sieci to 0.33, podczas gdy systemy konkursowe osiągały wynik maksymalny ok. 0.7 korelacji Pearsona [31]. Dla zbioru MSRvid różnica była jeszcze większa - systemy konkursowe osiągały wyniki przekraczające 0.8 korelacji Pearsona. Wskazuje to na fakt, że dla dłuższych tekstów takich jak pełne zdania ważnym czynnikiem może być bardziej złożona struktura bramki sieci rekurencyjnej. Kierunkiem rozwoju mogłoby tu być zastąpienie bramki Vanilla w proponowanych architekturach np. poprzez bramkę GRU i przeprowadzenie testów.

Wyniki testów wskazują na to, że przy trenowaniu proponowanych architektur warto zaplanować liczbę iteracji algorytmu L-BFGS powyżej 10, tak by uzyskać optymalny stopień wytrenowania. Część architektur okazała się odporna na przetrenowanie, tak więc nie należy się zbytnio obawiać ustalenia zbyt dużej ilości iteracji. Nie zaobserwowałam zysków z zastosowania dłuższych wektorów reprezentacji - długość 50 elementów w zupełności wystarczy do uzyskania satysfakcjonujących wyników. Problem ułamkowych anotacji najlepiej rozwiązać poprzez zastosowanie wyważania zamiast zaokrąglania wyników. Zaobserwowałam poprawę wyników w sieciach wzbogaconych o dodatkowe warstwy ukryte oraz architekturę drzewa zależnościowego, co wskazuje na to, że jest to dobry kierunek badań.

7 Możliwe ulepszenia

Na podstawie obserwacji wydajności proponowanych algorytmów wyróżniłam możliwe dalsze kierunki rozwoju sieci Vanilla dla rozpoznawania parafraz:

1. Stworzenie sieci Vanilla o jeszcze większej liczbie warstw ukrytych oraz architekturze opartej na drzewie rozkładu zdania innej gramatyki (np. składnikowej). Liczba warstw ukrytych może być parametrem algorytmu, który zostanie poddany testom.

2. Stworzenie systemu statystyczno-regułowego. Teksty podawane do sieci Vanilla mogą być wstępnie przetwarzane zgodnie z ustalonymi regułami (np. może być to usuwanie tzw. *stopwords* lub zastępowanie wykrytych synonimów zamiennikami), aby nieco zredukować ilość szumu informacyjnego i ułatwić sieci podjęcie decyzji.

3. Zmiana bramki Vanilla na bramkę GRU lub LSTM. Zamiana bramki na bardziej skomplikowaną zmniejszy szybkość działania, ale może też poprawić jakość, szczególnie na złożonych tekstach.

4. Wykorzystanie wprowadzonych architektur w systemie typu *ensemble* [32]. Jedna z proponowanych sieci (lub nawet wszystkie architektury na raz) mogą być włączone do zespołu głosujących klasyfikatorów. Fakt, że poszczególne klasyfikatory działają w inny sposób pozwala całemu systemowi *ensemble* reagować na różne cechy problemu, co często pozwala na uzyskanie dobrych wyników przez tego typu systemy.

8 Podsumowanie

Głębokie sieci neuronowe są podstawą wielu powstających obecnie systemów rozpoznawania parafraz. Systemy te są niezależne od specjalistycznych zasobów lingwistycznych i można je trenować na nieprzetworzonych przez ekspertów tekstach.

Zadanie rozpoznawania parafraz jest dla maszyn szczególnie trudne, ponieważ wymaga połączenia znajomości znaczenia słów (semantyki), zasad gramatycznych (składni) oraz zjawisk językowych takich jak przenośnie lub wyrażenia wieloznaczne. Ludzki mózg, wytrenowany do komunikacji za pomocą języka naturalnego rozumie te koncepcje instynktownie. Na podstawie obecnego stanu wiedzy na temat mózgu, wydaje się, że własności budowy sieci neuronowych zbliżają je do funkcjonowania tego organu. Być może więc obecnie tworzone sieci neuronowe stanowią krok w stronę osiągnięcia przez maszyny głębokiej znajomości języka naturalnego.

Niniejsza praca pokazuje, że możliwe jest stworzenie prostych koncepcyjnie modeli sieci neuronowych dających dobre wyniki rozpoznawania parafraz. Prezentowane nowe architektury oparte o sieci Vanilla-RNN używają niewielkich ilości danych treningowych i nie wymagają wstępnego przetworzenia tekstu w sposób wymagający wiedzy o języku, którą posiada człowiek. Wyniki osiągane przez sieci w zadaniu SemEval Interpretable STS są porównywalne do najnowszych systemów regułowych, tworzonych przez ekspertów lingwistycznych. Przeprowadzone testy pokazują czynniki wpływające na efektywność działania każdego z modeli.

Na podstawie uzyskanych w tej pracy wyników oraz mnogości publikacji dotyczących tematu rozpoznawania parafraz przez sieci neuronowe można mieć nadzieję, że trwające obecnie badania nad nowymi architekturami głębokich sieci to krok w odpowiednim kierunku, który doprowadzi do uzyskania nowych rekordów skuteczności.

Literatura

[1] *Oxford Advanced Learner's Dictionary of Current English*. Oxford University Press, 2010.

[2] Steven Bethard, Daniel Cer, Marine Carpuat, David Jurgens, Preslav Nakov, Torsten Zesch. SemEval-2016: Semantic Evaluation Exercises. http://alt.qcri.org/semeval2016/.

[3] Raymond Hickey. Levels of language. https://www.uni-due.de/ELE/, 2016.

[4] J. Chu-Carroll, J. Fan, B. K. Boguraev, D. Carmel, D. Sheinwald, C. Welty. Finding Needles in the Haystack: Search and Candidate Generation. *IBM J. Res. Dev.*, 56(3):300–311, Maj 2012.

[5] William B. Dolan, Chris Brockett. Automatically Constructing a Corpus of Sentential Paraphrases. *Third International Workshop on Paraphrasing (IWP2005)*. Asia Federation of Natural Language Processing, 2005.

[6] Nataliia Plotnikova, Gabriella Lapesa, Thomas Proisl, Stefan Evert. SemantiKLUE: Semantic Textual Similarity with Maximum Weight Matching. *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, strony 111–116, Denver, Colorado, 2015. Association for Computational Linguistics.

[7] Jiang Zhao, Man Lan, Jun Feng Tian. ECNU: Using Traditional Similarity Measurements and Word Embedding for Semantic Textual Similarity Estimation. *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, strony 117–122, Denver, Colorado, 2015. Association for Computational Linguistics.

[8] Hamed Hassanzadeh, Tudor Groza, Anthony Nguyen, Jane Hunter. UQeResearch: Semantic Textual Similarity Quantification. *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, strony 123–127, Denver, Colorado. Association for Computational Linguistics.

[9] Marta Costa-Jussà, Jose Mariño Acebal Mireia Farrús, Jose Fonollosa. Study and comparison of rule-based and statistical Catalan-Spanish machine translation systems. 2011.

[10] Lushan Han, Abhay L. Kashyap, Tim Finin, James Mayfield, Johnathan Weese. UMBC EBIQUITY CORE: Semantic Textual Similarity Systems. *Proceedings of the Second Joint Conference on Lexical and Computational Semantics*. Association for Computational Linguistics, 2013.

[11] Eneko Agirre, Carmen Banea, Claire Cardie, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, Weiwei Guo, Inigo Lopez-Gazpio, Montse Maritxalar, Rada Mihalcea, German Rigau, Larraitz Uria, Janyce Wiebe. SemEval-2015 Task 2: Semantic Textual Similarity, English, Spanish and Pilot on Interpretability. *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, strony 252–263, Denver, Colorado, 2015. Association for Computational Linguistics.

[12] Md Arafat Sultan, Steven Bethard, Tamara Sumner. DLS@CU: Sentence Similarity from Word Alignment and Semantic Vector Composition. *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, strony 148–153, Denver, Colorado, 2015. Association for Computational Linguistics.

[13] Richard Socher, Eric H. Huang, Jeffrey Pennington, Andrew Y. Ng, Christopher D. Manning. Dynamic Pooling and Unfolding Recursive Autoencoders for Paraphrase Detection. *Advances in Neural Information Processing Systems 24*. 2011.

[14] Joakim Nivre. Dependency grammar and dependency parsing. Raport instytutowy, Växjö University, 2005.

[15] Joseph Turian, Lev Ratinov, Yoshua Bengio. Word Representations: A Simple and General Method for Semi-supervised Learning. *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, strony 384–394, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[16] Kai Sheng Tai, Richard Socher, Christopher D. Manning. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. *CoRR*, abs/1503.00075, 2015.

[17] Sepp Hochreiter, Jürgen Schmidhuber. Long Short-Term Memory. *Neural Comput.*, 9(8):1735–1780, 1997.

[18] Marco Marelli, Stefano Menini, Marco Baroni, Luisa Bentivogli, Raffaella bernardi, Roberto Zamparelli. A SICK cure for the evaluation of compositional distributional semantic models. Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Hrafn Loftsson, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, Stelios Piperidis, redaktorzy, *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, strony 216–223, Reykjavik, Iceland, May 2014. European Language Resources Association (ELRA). ACL Anthology Identifier: L14-1314.

[19] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *CoRR*, abs/1412.3555, 2014.

[20] Xinchi Chen, Xipeng Qiu, Chenxi Zhu, Shiyu Wu, Xuanjing Huang. Sentence Modeling with Gated Recursive Neural Network. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, strony 793–798, Lizbona, Portugalia, 2015. Association for Computational Linguistics.

[21] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, Yoshua Bengio. Gated Feedback Recurrent Neural Networks. *CoRR*, abs/1502.02367, 2015.

[22] Andrej Karpathy, Fei-Fei Li. Deep Visual-Semantic Alignments for Generating Image Descriptions. *CoRR*, abs/1412.2306, 2014.

[23] Alex Graves. Generating Sequences With Recurrent Neural Networks. *CoRR*, abs/1308.0850, 2013.

[24] Jeffrey Pennington, Richard Socher, Christopher D. Manning. GloVe: Global Vectors for Word Representation. *Empirical Methods in Natural Language Processing (EMNLP)*, strony 1532–1543, 2014.

[25] Andrew Ng, Jiquan Ngiam, Chuan Yu Foo, Yifan Mai, Caroline Suen. Unsupervised feature learning and deep learning tutorial. http://ufldl.stanford.edu.

[26] Dong C. Liu, Jorge Nocedal. On the Limited Memory BFGS Method for Large Scale Optimization. *MATHEMATICAL PROGRAMMING*, 45:503–528, 1989.

[27] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, David McClosky. The Stanford CoreNLP natural language processing toolkit. *Association for Computational Linguistics (ACL) System Demonstrations*, strony 55–60, 2014.

[28] Mikio L. Braun, Johannes Schaback, Matthias L. Jugel, Nicolas Oury. jblas - Linear Algebra for Java. http://jblas.org/, 2009–2016.

[29] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[30] Rajendra Banjade, Nobal Bikram Niraula, Nabin Maharjan, Vasile Rus, Dan Stefanescu, Mihai Lintean, Dipesh Gautam. NeRoSim: A System for Measuring and Interpreting Semantic Textual Similarity. *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, strony 164–171, Denver, Colorado, 2015. Association for Computational Linguistics.

[31] Eneko Agirre, Mona Diab, Daniel Cer, Aitor Gonzalez-Agirre. SemEval-2012 Task 6: A Pilot on Semantic Textual Similarity. *Proceedings of the First Joint Conference on Lexical and Computational Semantics - Volume 1: Proceedings of the Main Conference and the Shared Task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation*, SemEval '12, strony 385–393, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[32] Thomas G. Dietterich. Ensemble Methods in Machine Learning. *Proceedings of the First International Workshop on Multiple Classifier Systems*, MCS '00, strony 1–15, London, UK, UK, 2000. Springer-Verlag.