Bot odpowiadający na pytania użytkownika.

Kobiela Rafał i Krubiński Mateusz

Warsaw University of Technology

Streszczenie Próba implementacji systemu, który dla pytania i zestawu proponowanych odpowiedzi wybiera poprawną. Projekt wykonany w ramach przedmiotu 'Eksploracja danych tekstowych z uczeniem głębokim' w semestrze letnim 2017/2018.

Keywords: NLP · QA bot · Anwser selection.

1 Zarys problemu / wstęp, opis dziedziny

Zagadnienie wyboru odpowiedzi jest ważnym problemem w całym dziale botów konwersacyjnych. Formalnie zadaniem jest znalezienie odpowiedzi na pytanie q wśród zbioru zdań c1, c2, ..., cn. W standardowej architekturze pipelinu [1], wybór odpowiedzi jest aplikowany na wynik modułu który wykonuje text retrival. Wybrane odpowiedzi mogą następnie być bezpośrednio przekazane użytkownikowi albo przekazane jako wejście do kolejnych modułów które zidentyfikują kolejne odpowiedzi.

Naszym zadaniem w tym projekcie będzie jedynie wybranie odpowiedniej odpowiedzi spośród zbioru zdań.Pomimo tego, że problem wyboru odpowiedzi jest formalnie kwalifikowany jako problem klasyfikacyjny, to w kandydaci na odpowiedź są rangowani prawdopodobieństwami zawierania odpowiedzi na pytanie. Na podstawie tego można stwierdzić, ze jest to tak na prawdę problem rankingowy.

Postaramy się zaimplementować nowatorskie podejście rankingowe "pairwise". Mając dane pytanie, naszym podejściem będzie branie par odpowiedzi jako wejście do sieci neuronowej i uczenie się która z nich jest bardziej prawdopodobną odpowiedzią. Będziemy używać modelu nazwanego "Noise Contrastive Estimation" by nauczyć się reprezentacji trójek (pytanie, dobra odpowiedź, zła odpowiedź), które będą wejściem. Następnie użyjemy "triplet loss function" aby nauczyć się nielinowych korelacji z naszych trójek.

2 Cel pracy projektowej

Celem naszej pracy jest zaimplementowanie rozwiązania przedstawionego w pracy [4]. Jako input do sieci autorzy wykorzystują trójkę pytania/dobra odpowiedź/zła odpowiedź. Sieć, której uzyliśmy miała trzy zasadnicze części - odpowiedzialną za reprezentację każdego ze zdań, odpowiedzialną za ich wzajemne

porównywanie. oraz trzecią, agregującą wyniki w odpowiedniej funkcji straty. Implementując dwie pierwsze części opieralismy się na pracy [2].

Używaliśmy języka Python, oraz przede wszystkim bibliotek keras i tensorflow. Korzystaliśmy z rad zamieszczonych w książce [5].

3 Przegląd rozwiązań SOTA (state-of-the-art)

Praca [4] jest pierwszą pracą z dziedziny wyboru odpowiedzi (anwser selection), która zamiast podejścia 'pointwise' (input to para pytanie/odpowiedź) implementuje nowatorskie podejście 'pairwaise' (input to trójkę pytania/dobra odpowiedź/zła odpowiedź). Jej dokładną architekturę omówimy później. Za tym podejściem stoi intuicja oparta o zdolności sieci do rozróżniania dobrych odpowiedzi od złych, a co za tym idzie, przyznawania większego wyniku odpowiedziom dobrym.

Wcześniejsze rozwiązania opierały się intuicji, że odpowiedź poprawna powinna być w jakimś sensie bardziej podobna do pytania niż odpowiedzi negatywne. H. He, K. Gimpel i J. Lin w "Multi-perspective sentence similarity modeling with convolutional neural networks" oraz H. He and J. Lin w "Pairwise word interaction modeling with deep neural networks for semantic similarity measurement" wykorzystują konwolucyjne sieci neuronowe, które poprzez używania kilku rodzajów filtrów próbują uzyskać reprezentację zdań jako wektrorów, a nastepnie używają warstwy gęstej do ich porównania. Inne pomysły, to chociażby użycie połączonego mechanizmu uwagi oraz sieci konwolucyjnych, które stosują autorzy w "Attention-based multi-perspective convolutional neural networks for textual similarity measurement". Jeszcze inne współczesne podejścia próbują używać jak największej ilości dodatkowych źródeł informacji, żeby jeszcze lepiej umieć reprezentować zdania (zobacz "Learning to rank short text pairs with convolutional deep neural networks").

4 Opis danych

Do trenowania modeli ożyliśmy zbioru danych WikiQA. Jest to zescrapowana sekcja "Czy wiesz" z angielskiej wikipedii. Zbiór zawiera pytania z tej sekcji oraz wszystkie zdania które znalazły się w artykule do którego prowadził link. Mamy tutaj 3047 pytań i około 20 000 odpowiedzi na pytania. Odpowiedzi oczywiście zawierają takie zdania w których znajdziemy odpowiedź na pytanie, jak i takie w których nie znajdziemy odpowiedzi na pytanie.

Przykładowo dla zdania "How African Americans were immigrated to the US" otrzymujemy następujący zbiór zdań:

Sentence	Label
African immigration to the United States refers to immigrants to the United States who are or were nationals of Africa .	0
The term African in the scope of this article refers to geographical or national origins rather than racial affiliation.	0
From the Immigration and Nationality Act of 1965 to 2007, an estimated total of 0.8 to 0.9 million Africans immigrated to the United State	0
African immigrants in the United States come from almost all regions in Africa and do not constitute a homogeneous group.	0
They include people from different national, linguistic, ethnic, racial, cultural and social backgrounds.	0
As such, African immigrants are to be distinguished from African American people, the latter of whom are descendants of mostly West a	1

Widać, że jedno ze zdań jest oznaczone flagą 1 i zawiera odpowiedź na postawione pytanie.

Do zanurzenia użyliśmy nauczonej sieci Word2vec, każde słowo reprezentując jako wektor w 300 wymiarowej przestrzeni. Zdania obcinaliśmy do 20 słów, a dla zdań krótszych dopełnialiśmy zerami. Ponieważ zdania posiadały różne ilości negatywnych odpowiedzi dokonaliśmy transformacji zbioru. Dla każdego pytania poza dobrą odpowiedzią losowaliśmy 6 negatywnych odpowiedzi na to pytanie, oraz jedną dodatkową odpowiedź z całego zbioru negatywnych odpowiedzi (założyliśmy, że w odpowiedziach na inne pytania nie ma odpowiedzi na rozważane pytanie). Jeśli pytanie nie posiadało 6 negatywnych odpowiedzi, braliśmy ile mogliśmy, dopełniając odpowiedziami z innych pytań. Dostaliśmy ramkę danych następującej postaci:

	Question	Pos_answer	Neg_answer
9599	when is it memorial day	Memorial Day is a United States federal holida...	Memorial Day is a day of remembering the men a...
9602	when is it memorial day	Memorial Day is a United States federal holida...	It typically marks the start of the summer vac...
9601	when is it memorial day	Memorial Day is a United States federal holida...	By the 20th century Memorial Day had been exte...
9611	when is it memorial day	Memorial Day is a United States federal holida...	It is believed that this practice began before...
9612	when is it memorial day	Memorial Day is a United States federal holida...	Memorial Day is not to be confused with Vetera...
9602	when is it memorial day	Memorial Day is a United States federal holida...	It typically marks the start of the summer vac...
4929	when is it memorial day	Memorial Day is a United States federal holida...	Artist's impression of the moon during the Lat...

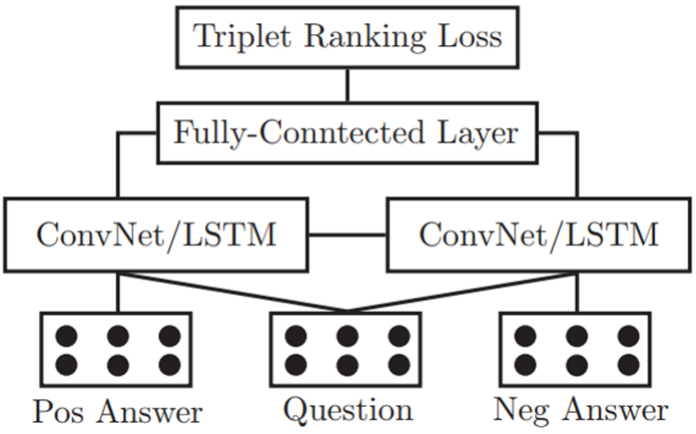
Niektóre pytania musieliśmy odrzucić, i ostatecznie nasz zbiór miał 8484 elementów, po 7 trójek dla każdego z 1212 pytań.

5 Sposób rozwiązania, metody

Opiszmy teraz dokładnie architekturę naszego modelu.

5.1 Architektura modelu

Na obrazku widoczna jest ogólna architektura naszego modelu rankingowego. Widzimy dwa główne komponenty.



Bazowy komponent składa się z dwóch sieci neuronowych typu pointwise. Każda z nich bierze jako wejście parę (pytanie, odpowiedź) i produkuje score podobieństwa który reprezentuje odległość semantyczną w parze. Nasz pairwise model powinien dawać większy score dla par pytanie dobra odpowiedź i mniejszy dla par pytanie, zła odpowiedź. Na samej górze modelu znajduje się "triplet loss function" o której więcej poniżej.

Nasz model ma strukturę "Siamese" [3]. Oznacza to, że składa się z dwóch równoległych modeli pointwise, każdym przetwarzającym parę pytanie odpowiedź. Parametry obu modeli są dzielone podczas uczenia.

Jako części pointwise zaimplementowaliśmy rozwiązanie z [2] oraz dwie uproszczone wersje tego rozwiązania.

5.2 Triplet Ranking Loss Function

Używany przez nas komponent reprezentujący i porównujący zdania bierze jako wejście pary (pytanie, odpowiedź) i daje na wyjściu score który jest wartością funkcji f(). Score ten mówi nam jak semantycznie podobne są nasze dwa zdania wejściowe. Naszym celem jest nauczenie się reprezentacji funkcji f() tak aby dla danego pytania q przypisywała pozytywnym parom (q, p^+) większy score niż parom negatywnym (q, p^-):

$$f(q, p^+) > f(q, p^-), \forall q, p^+, p^-$$

gdzie q, p^+, p^- oznaczają odpowiednio pytanie, dobrą odpowiedź na pytanie, złą odpowiedź na pytanie. Następnie używamy 'trilpet ranking function' jako funkcji straty która minimalizuje odległość pomiędzy q i dobrą odpowiedzią p^+ oraz maksymalizuje odległość pomiędzy q i negatywną odpowiedzią p^-. W funkcji następuje sumowanie po wszystkich pozytywnych parach (q, p^+) a następnie sumowanie po wszystkich negatywnych odpowiedziach dla pytania q:

$$min_W \sum_{(q,p^+)} \sum_{p^- \in N} max(0, 1 - (f(q, p^+) - f(q, p^-))) + \lambda ||W||^2$$

gdzie λ jest parametrem regularyzacyjnym (tak jak autorzy w pracy używaliśmy $\lambda = 10^{-4}$), a W jest macierzą parametrów sieci neuronowej.

Ponieważ nie jest to standardowa funkcja straty, zaimplementowaliśmy ją sami. To co ciekawe, to że wartości funkcji nie zależą w żaden sposób od prawdziwych etykiet obiektów. Musieliśmy je jednak jakoś zasymilować, ponieważ wymaga tego biblioteka keras z której korzystaliśmy.

```
def triplet_loss(y_true,y_pred):


    positive_pred = y_pred[:,1]
    negative_pred = y_pred[:,0]

    sum_loss = 1 - positive_pred + negative_pred

    loss = K.maximum(sum_loss,0.0)

    return K.sum(loss)
```

5.3 Warstwa reprezentująca i porównująca zdania

Sposób 1 W pierwszej fazie projektów zaimplementowaliśmy własną, dość prostą strukturę reprezentującą zdanie. Składała się ona z dwóch warstw konwolucyjnych, z max i avg poolingiem.

```
left_input = Input(shape=(20,300), name = 'n')
mid_input = Input(shape=(20,300), name = 'q')
right_input = Input(shape=(20,300), name = 'p')

conv1 = layers.Conv1D(128,5, activation='tanh',kernel_regularizer=regularizers.l2(1e-4),input_shape =
(None,20,300))
max_pool1 = layers.MaxPooling1D(3)

conv2 = layers.Conv1D(256, 3, activation='tanh',kernel_regularizer=regularizers.l2(1e-4))
max_pool2 = layers.MaxPooling1D(2)
avg_pool2 = layers.AveragePooling1D(2)

flatten = layers.Flatten()
```

Następnie każdy z trzech inputów przechodził przez te warstwy, poniżej przykład dla lewego, czyli negatywnej odpowiedzi.

```
left_output = conv1(left_input)
left_output = max_pool1(left_output)

left_output = conv2(left_output)
left_output = layers.concatenate([max_pool2(left_output),avg_pool2(left_output)], axis = -1)

left_output = flatten(left_output)
```

W kolejnym kroku łączymy wyniki dla pytania i negatywnej odpowiedzi oraz pytania i pozytywnej odpowiedzi. Konkatenujemy je, dołączamy dwie warstwy gęste, i zwracamy parę - score dla negatywnej oraz pozytywnej odpowiedzi.

```
merged_L = layers.concatenate([left_output, mid_output], axis=-1)
merged_R = layers.concatenate([right_output, mid_output], axis=-1)


dens1 = layers.Dense(256,activation='tanh',kernel_regularizer=regularizers.l2(1e-4))

droprout1 = layers.Dropout(0.05)

merged_L = droprout1(dens1(merged_L))
merged_R = droprout1(dens1(merged_R))



dens2 = layers.Dense(256, activation='tanh',kernel_regularizer=regularizers.l2(1e-4))
dens_pred = layers.Dense(1, activation='softmax',kernel_regularizer=regularizers.l2(1e-4))

pred_L = dens_pred(dens2(merged_L))
pred_R = dens_pred(dens2(merged_R))

pred = layers.concatenate([pred_L,pred_R], axis = -1)

model = Model(inputs = [left_input, mid_input, right_input], outputs = pred)
```

Sposób 2 Drugi sposób opierał się na zastosowaniu warstw LSTM. Pierwsza i druga cześć sieci są analogiczne, trzecia jest dokładnie taka sama.

```
left_input = Input(shape=(20,300), name = 'n')
mid_input = Input(shape=(20,300), name = 'q')
right_input = Input(shape=(20,300), name = 'p')

lstm1 = layers.LSTM(50, kernel_regularizer = regularizers.l2(1e-4), input_shape = (None,20,300))

max_pool1 = layers.MaxPooling1D(2)

lstm2 = layers.LSTM(50, kernel_regularizer = regularizers.l2(1e-4), input_shape = (None,20,300))

max_pool2 = layers.MaxPooling1D(2)
avg_pool2 = layers.AveragePooling1D(2)

flatten = layers.Flatten()
```

```
left_output = lstm1(left_input)
left_output = max_pool1(left_output)

left_output = lstm2(left_output)
left_output = layers.concatenate([max_pool2(left_output),avg_pool2(left_output)], axis = -1)

left_output = flatten(left_output)
```

Sposób 3 Nasze finalne podejście to dokładna implementacja modelu z [2].

Kod w pythonie jest mało czytelny (dołączony z raportem), ograniczymy się więc do opisu.

Na początku, dla każdego z 300 wymiarów zanurzenia, dla dwóch typów n-gramu - $n = 2, n = 3$ oraz dla 2 typów globalpoolingu - $minimax$ tworzę oddzielną 1-wymiarową sieć konwolucyjną. Daje mi to pewną macierz cech dla każdego

ze zdań (macierz, ponieważ ilość kolumn zależy od ilości filtrów użytych w sieci). Następnie, dla par pytanie/dobra odpowiedź oraz pytanie/zła odpowiedź tworzę wektor, w którym składuję informacje o odległościach pomiędzy odpowiadającymi kolumnami z macierzy. Odległość liczę na 3 sposoby, i pamiętam wszystkie wyniki - liczę odległość L_1, L_2 oraz cosinusow-ą.

Podobnie postępuje dla całych zanurzeń - stosuję na całym zdaniu 3 typy okien $n = 2, n = 3$ oraz $n = \infty$ oraz 3 typy globalpoolingów $max, miniavg$. Analogicznie, daje mi to pewną macierz (wiersze przechowują zebrane przez pooling informacje, a kolumny to kolejne filtry sieci konwolucyjnej). Do wektora z informacjami o porównaniach dołączam analogicznie odległości (3 typy, jak wyżej) dla odpowiadających kolumn - jest to algorytm nr 2 z pracy, tzw. Vertical comparison.

W kolejnym kroku, porównuje odpowiadające wiersze w macierzach, licząc teraz tylko odległość L_2 i cosinusow-ą 9algorytm nr 1, tzw. Horizontal comparison).

Konkatenuję następnie otrzymane wyniki, dostając wektory $merged_L$ i $merged_R$, identycznie jak w prostych sieciach. Na koniec, przepuszczam je przez warstwy gęste, dokładnie tak jak wczesniej.

Jak widzimy, powyższe 3 sposoby na inaczej reprezentują zdania i badają związek pomiędzy reprezentacjami, ale wszystkie finalnie przechodza przez warstwę gęsta i produkują score, będący funkcją $f()$ ze wcześniejszych rozważań na temat triplet loss function.

6 Wyniki i ich analiza, wnioski

Niestety, zarówno pierwszy i drugi sposób okazał się nieskuteczny, W czasie uczenia, funkcja straty przyjmowała ciągle średnią wartość 1. Odkryliśmy, że parom dobrym i złym przyporządkowuje dokładnie ten sam score, czyli 1. Można na to patrzeć, jak na wpadanie w lokalne minimum, ale uważamy, że obwiniać należy raczej ubogą strukturę modelu. Warstwy reprezentujące są bardzo ubogie, i nie były w stanie rozróżnić dobrych odpowiedzi od złych.

Sposób trzeci, który jak wierzymy jest dokładną implementacją struktury opisanej w [2], wywołał niestety inne problemy. Gdy próbowaliśmy używać modelu na domowych komputerach, bardzo szybko kończył się RAM, nawet nie w trakcie uczenia, a w trakcie tworzenie warstw. Wykorzystaliśmy sposobność, i uruchomiliśmy nasz kod na serwerze obliczeniowym. Jednak również tutaj, model zajął całe dostępne 80GB RAM-u i musieliśmy przerwać. Do tego momentu, nie otrzymaliśmy żadnego komunikatu o błędzie.

7 Podsumowanie i pomysły na dalsze prace

Celem pracy było zaimplementowanie algorytmu znajdującego odpowiedź na dane pytanie w danym zbiorze zdań. Wzorowaliśmy się na pracy naukowej, która robiła to z bardzo dobrymi wynikami. Jej głównym nowatorskim elementem była

funkcja straty "Triplet ranking function", którą zaimplementowaliśmy. Zaimplementowaliśmy trzy rozwiązania. Dwa z nich były prostsze a jedno było wiernym odzwierciedleniem tego z pracy. Niestety podczas testów nasze lekkie modele, prawdopodobnie, wpadały w lokalne minimum i wszystkim wynikom przypisywały score równy 1. Naszym zdaniem wynikało to ze zbyt ubogiej struktury modelu. Wierne odwzorowanie modelu z pracy , niestety również się nie powiodło, ale z innego względu. Po prostu zabrakło nam pamięci RAM do zaalokowania obiektu modelu w pamięci. Był on bardzo złożony i składał się z wielu warstw. Podejrzewamy, że ta sieć zadziałałaby bardzo dobrze, ponieważ w pracy, na której się wzorowaliśmy, wypadała zdecydowanie najlepiej.

Pierwszym pomysłem na dalsze prace jest próba uruchomienia modelu na klastrze obliczeniowym z dostępną większą pamięcią RAM. Innym sposobem na podejście do tego zagadnienia jest zastosowanie bardziej klasycznej funkcji straty. Następnie można by też zaimplementować bardziej klasyczne podejście "pointwise", w którym model będzie przyjmował, nie trójkę, ale dwójkę pytanie-zdanie, oraz flagę, która oznacza czy dane zdanie odpowiada na pytanie. Takie podejście nie wymaga aż tak skomplikowanej sieci neuronowej i myślimy że jest do zrealizowania na zwykłych domowych komputerach.

Uważamy, że cel projektu, pomimo braku pozytywnych wyników, został osiągnięty. Temat został zgłębiony, a sieć zaimplementowana. Jedyną przeszkodą była tutaj złożoność modelu.

8 Zapisane modele i zbiory treningowe

W folderze Final, w katalogu projektowym, można znaleźć wszystkie zbiory danych oraz kody użyte do budowy sieci. Plik *output.csv* zawiera przetworzony zbiór danych WikiQA. Pliki *Negative.npy*, *Positive.npy* i *Question.npy* zawierają odpowiednio zdania negatywne, pozytywne i pytania przetworzone na wektory Word2Vec. Można je otworzyć za pomocą pakietu *numpy* poleceniem $numpy.load(plik)$. Następnie pliki $modelConv.h5$ oraz $modelLSTM.h5$ zawierają modele sieci. By je otworzyć musimy zaimportować moduł $load\_model$ z $keras.models$ i użyć funkcji $load\_model(plik)$. Ponadto w notebooku $Word2vec+models.ipynb$ znajdują się wszystkie kody które napisaliśmy.

Literatura

1. S. Tellex, B. Katz, J. Lin, G. Marton, and A. Fernandes. *Quantitative evaluation of passage retrieval algorithms for question answering.* SIGIR, 2003.
2. Hua He, Kevin Gimpel, and Jimmy Lin *Multi-Perspective Sentence Similarity Modeling with Convolutional Neural Networks.* EMNLP 2015.
3. J. Bromley, J. W. Bentz, L. Bottou, I. Guyon, Y. LeCun, C. Moore, E. Sackinger, and R. Shah. *Signature verication using a siamese time delay neural network.* IJPRAI, 1993.
4. Jinfeng Rao1, Hua He1, and Jimmy Lin. *Noise-Contrastive Estimation for Answer Selection with Deep Neural Networks.* CIKM, 2016.
5. Francois Chollet *Deep Learning with Python* MANNING PUBN, 2017.