Evaluation of LSTM-cell based Recurrent Neural Network on Next Word Prediction

CS523 Summer II

Prof.Peter Chin

Chaobang Huang

Yixiu Zhu

Contents

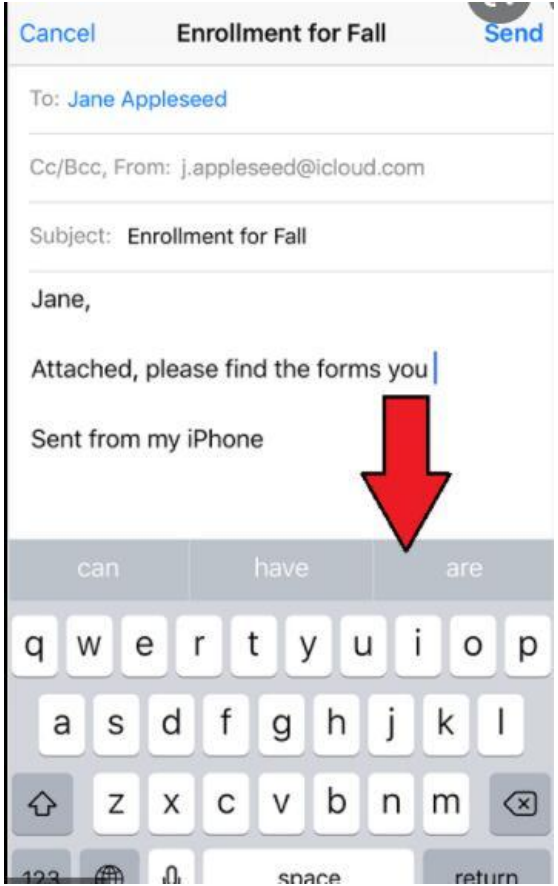
- Introduction
- Background & Initial Ideas
    - Why LSTM
- Dataset & Pre-processing
- Implementation
    - Feature Engineering
    - How to design loss and metrics
- Training Practices
    - Architecture Design
    - Regularization and hyperparameters
    - Training results
- Testing
- Future Work

Introduction

In this project, we firstly compared long short-term memory (LSTM) unit and the gradient recurrent unit (GRU).

We then focus on feature engineering and architecture design. After training, we will let the model make prediction based on a set of 5-word sentences from the text.

We would also conduct a closer analysis of architecture's performance and efficiency in this data set.



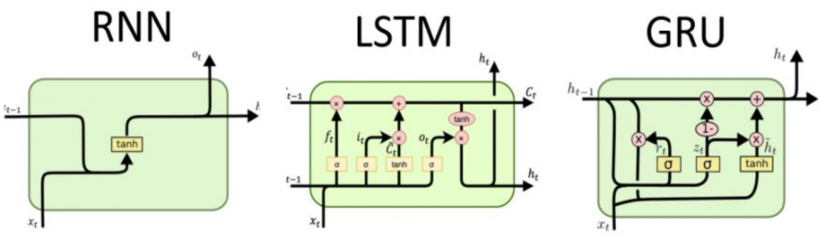
Background & Initial Ideas

RNN(Vanilla) :

Purpose: send all the old state information at all time steps (long-term dependencies)

Issue: vanishing & exploding gradient (large errors & unstability)

LSTM	GRU
More accurate in longer sequence training	Faster in training
More gates to control, though it can take more time to train	Efficient due to low memory usage and simpler structure



Reference: http://dprogrammer.org/rnn-lstm-gru

BOSTON UNIVERSITY

Dataset

Data Source:

- In our next word prediction model, we train our LSTM with a fiction(long string):
  - We selected the book “The Adventures of Sherlock Holmes”
    - relative small dataset (around 100000 words)

text1.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

Project Gutenberg's The Adventures of Sherlock Holmes, by Arthur Conan Doyle

This eBook is for the use of anyone anywhere at no cost and with
almost no restrictions whatsoever. You may copy it, give it away or
re-use it under the terms of the Project Gutenberg License included
with this eBook or online at www.gutenberg.net

BOSTON UNIVERSITY

Data Pre-processing

```
>>> text[0:100]
result: '\nproject gutenbergs the adventures of sherlock holmes by arthur
conan doyle\n\nthis ebook is for the u'
```

Step 1: read the entire string

```
>>> words[0:10]
result:
['project',
 'gutenbergs',
 'the',
 'adventures',
 'of',
 'sherlock',
 'holmes',
 'by',
 'arthur',
 'conan']
```

Step 2: get rid of all punctuation marks and split the entire string into list of words

```
>>> unique_words[0:10]
result:
array(['1', '10', '100', '1000', '10_s_', '10th', '1100', '1115', '1130',
       '117'], dtype='<U26')
```

Step 3: use np.unique to get unique occurrence of words and sort them

```
>>> unique_words=unique_words[eliminate(unique_words):]

>>> unique_words[:10]
result:
array(['a', 'abandoned', 'abandons', 'abbots', 'aberdeen', 'abhorrent',
       'abide', 'abjure', 'able', 'abnormal'], dtype='<U26')
```

Step 4: remove training-irrelevant “noises”

Step 5: Split the data into features and labels.

“ To Sherlock Holmes she is always the woman ...”

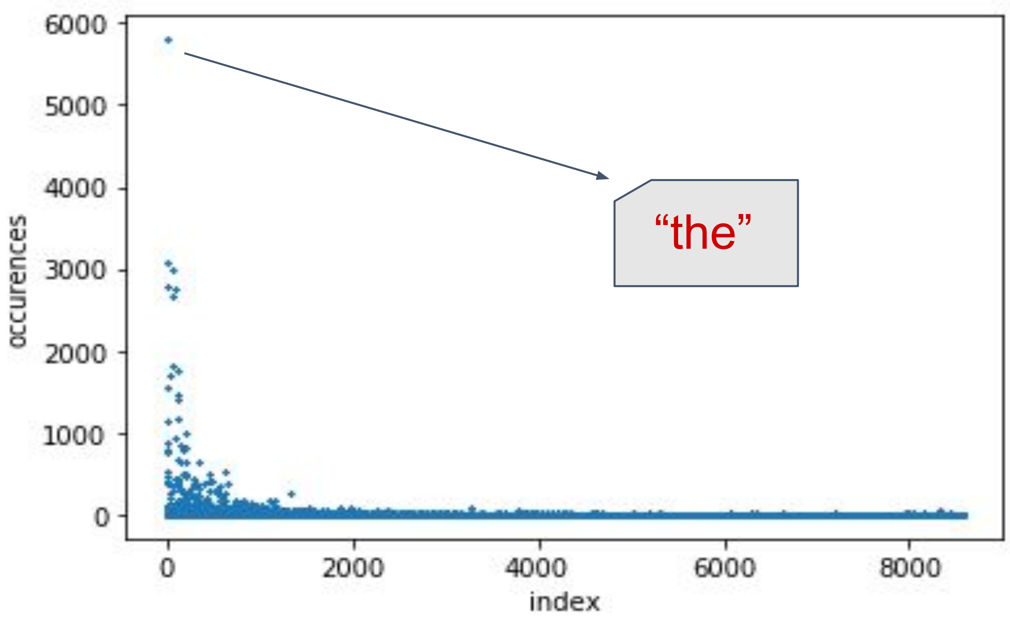
```
>>> X.shape
result: (107542, 5, 50)

>>> Y.shape
result: (107542, 50)
```

Distribution of unique words

```
>>> len( unique_words)
result: 8576
```

Figure 9: “unique_words” obtained from previous data processing

Figure 10: Distribution of “unique_words”



BOSTON UNIVERSITY

Implementation

- Feature Engineering:
  - Using the domain knowledge of the data to create features that can be used in training a machine learning algorithm.
- How to design features, labels, loss, and metrics

Feature Engineering

First Attempt: **One-hot encoding**

- Pros: Easy to implement
- Cons:
  - Waste tremendous amount of RAM

    ```
    MemoryError: Unable to allocate 7.01 GiB for an array with shape
    (107542, 8744) and data type float64
    ```

  - Two similar words are linearly independent (same loss)
  - Unrealistic to do the classification for over 8000 classes

Feature Engineering

New Approach: **GloVe Vector**

- **Pros**
  1. Similar words are closer to each other (euclidean distance)
  2. No longer need to occupy significant amount of RAM for whatever number of unique words in the text
  3. Turn classification question into kind of regression problem
- **Cons**
  1. These vectors are pre-trained, which may not be best representation for our corpus and application

Demo of GloVe Vector

glove.6B.50d.txt - Notepad

File Edit Format View Help

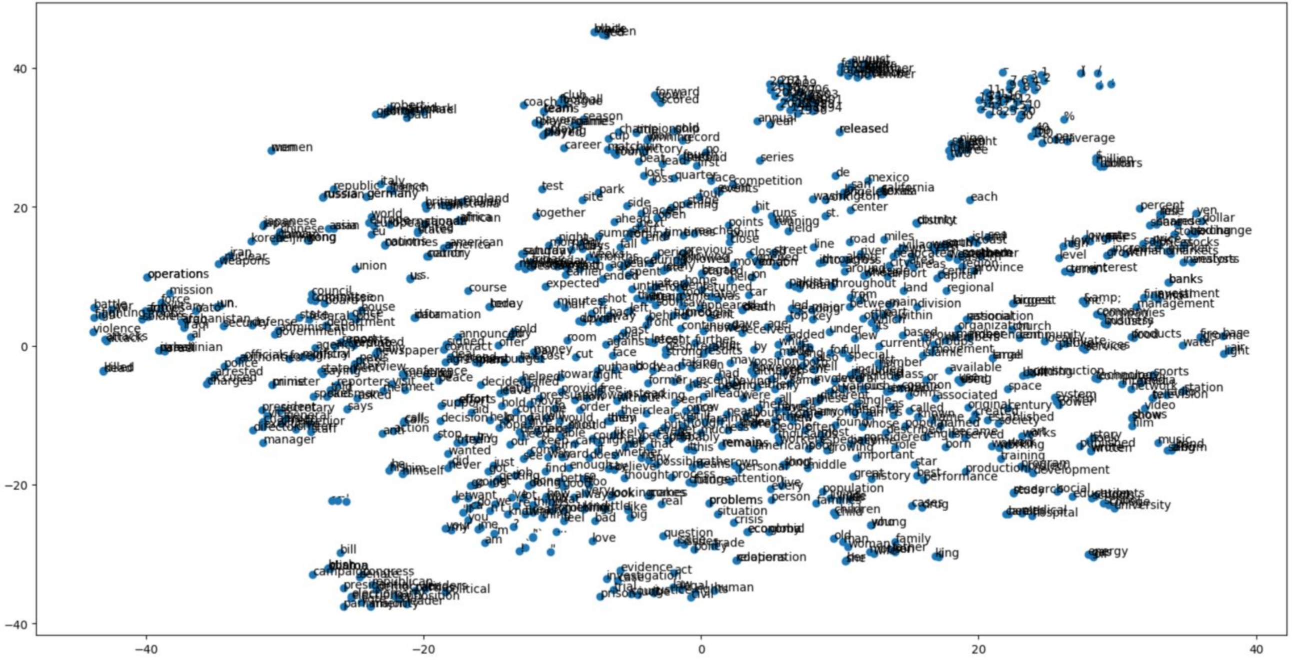
```
the 0.418 0.24968 -0.41242 0.1217 0.34527 -0.044457 -0.49688 -0.17862 -0.00066023 -0.6566 0.27843 -0.14767 -0.55677 0.14658 -0.0095095 0.011658 0.10204 -0.12792 -0.8443 -0.12181 -0.016801 -0.33279 -0.1552 -0.23131 -0.19181 -1.8823 -0.767
, 0.013441 0.23682 -0.16899 0.40951 0.63812 0.47709 -0.42852 -0.55641 -0.364 -0.23938 0.13001 -0.063734 -0.39575 -0.48162 0.23291 0.090201 -0.13324 0.078639 -0.41634 -0.15428 0.10068 0.48891 0.31226 -0.1252 -0.037512 -1.5179 0.12612 -0.0
. 0.15164 0.30177 -0.16763 0.17684 0.31719 0.33973 -0.43478 -0.31086 -0.44999 -0.29486 0.16608 0.11963 -0.41328 -0.42353 0.59868 0.28825 -0.11547 -0.041848 -0.67989 -0.25063 0.18472 0.086876 0.46582 0.015035 0.043474 -1.4671 -0.30384 -0.
of 0.70853 0.57088 -0.4716 0.18048 0.54449 0.72603 0.18157 -0.52393 0.10381 -0.17566 0.078852 -0.36216 -0.11829 -0.83336 0.11917 -0.16605 0.061555 -0.012719 -0.56623 0.013616 0.22851 -0.14396 -0.067549 -0.38157 -0.23698 -1.7037 -0.86692
to 0.68047 -0.039263 0.30186 -0.17792 0.42962 0.032246 -0.41376 0.13228 -0.29847 -0.085253 0.17118 0.22419 -0.10046 -0.43653 0.33418 0.67846 0.057204 -0.34448 -0.42785 -0.43275 0.55963 0.10032 0.18677 -0.26854 0.037334 -2.0932 0.22171 -0
and 0.26818 0.14346 -0.27877 0.016257 0.11384 0.69923 -0.51332 -0.47368 -0.33075 -0.13834 0.2702 0.30938 -0.45012 -0.4127 -0.09932 0.038085 0.029749 0.10076 -0.25058 -0.51818 0.34558 0.44922 0.48791 -0.080866 -0.10121 -1.3777 -0.10866 -0
in 0.33042 0.24995 -0.60874 0.10923 0.036372 0.151 -0.55083 -0.074239 -0.092307 -0.32821 0.09598 -0.82269 -0.36717 -0.67009 0.42909 0.016496 -0.23573 0.12864 -1.0953 0.43334 0.57067 -0.1036 0.20422 0.078308 -0.42795 -1.7984 -0.27865 0.11
a 0.21705 0.46515 -0.46757 0.10082 1.0135 0.74845 -0.53104 -0.26256 0.16812 0.13182 -0.24909 -0.44185 -0.21739 0.51004 0.13448 -0.43141 -0.03123 0.20674 -0.78138 -0.20148 -0.097401 0.16088 -0.61836 -0.18504 -0.12461 -2.2526 -0.22321 0.50
" 0.25769 0.45629 -0.76974 -0.37679 0.59272 -0.063527 0.20545 -0.57385 -0.29009 -0.13662 0.32728 1.4719 -0.73681 -0.12036 0.71354 -0.46098 0.65248 0.48887 -0.51558 0.039951 -0.34307 -0.014087 0.86488 0.3546 0.7999 -1.4995 -1.8153 0.41128
's 0.23727 0.40478 -0.20547 0.58805 0.65533 0.32867 -0.81964 -0.23236 0.27428 0.24265 0.054992 0.16296 -1.2555 -0.086437 0.44536 0.096561 -0.16519 0.058378 -0.38598 0.086977 0.0033869 0.55095 -0.77697 -0.62096 0.092948 -2.5685 -0.67739 0
for 0.15272 0.36181 -0.22168 0.066051 0.13029 0.37075 -0.75874 -0.44722 0.22563 0.10208 0.054225 0.13494 -0.43052 -0.2134 0.56139 -0.21445 0.077974 0.10137 -0.51306 -0.40295 0.40639 0.23309 0.20696 -0.12668 -0.50634 -1.7131 0.077183 -0.3
- -0.16768 1.2151 0.49515 0.26836 -0.4585 -0.23311 -0.52822 -1.3557 0.16098 0.37691 -0.92702 -0.43904 -1.0634 1.028 0.0053943 0.04153 -0.018638 -0.55451 0.026166 0.28066 -0.66245 0.23435 0.2451 0.025668 -1.0869 -2.844 -0.51272 0.27286 0.
that 0.88387 -0.14199 0.13566 0.098682 0.51218 0.49138 -0.47155 -0.30742 0.01963 0.12686 0.073524 0.35836 -0.60874 -0.18676 0.78935 0.54534 0.1106 -0.2923 0.059041 -0.69551 -0.18804 0.19455 0.32269 -0.49981 0.306 -2.3902 -0.60749 0.37107
on 0.30045 0.25006 -0.16692 0.1923 0.026921 -0.079486 -0.91383 -0.1974 -0.053413 -0.40846 -0.26844 -0.28212 -0.5 0.1221 0.3903 0.17797 -0.4429 -0.40478 -0.9505 -0.16897 0.77793 0.33525 0.3346 -0.1754 -0.12017 -1.7861 0.29241 0.55933 0.02
is 0.6185 0.64254 -0.46552 0.3757 0.74838 0.53739 0.0022239 -0.60577 0.26408 0.11703 0.43722 0.20092 -0.057859 -0.34589 0.21664 0.58573 0.53919 0.6949 -0.15618 0.05583 -0.60515 -0.28997 -0.025594 0.55593 0.25356 -1.9612 -0.51381 0.69096
was 0.086888 -0.19416 -0.24267 -0.33391 0.56731 0.39783 -0.97809 0.03159 -0.61469 -0.31406 0.56145 0.12886 -0.84193 -0.46992 0.47097 0.023012 -0.59609 0.22291 -1.1614 0.3865 0.067412 0.44883 0.17394 -0.53574 0.17909 -2.1647 -0.12827 0.29
said 0.38973 -0.2121 0.51837 0.80136 1.0336 -0.27784 -0.84525 -0.25333 0.12586 -0.90342 0.24975 0.22022 -1.2053 -0.53771 1.0446 0.62778 0.39704 -0.15812 0.38102 -0.54674 -0.44009 1.0976 0.013069 -0.89971 0.41226 -2.2309 0.28997 0.32175 -
with 0.25616 0.43694 -0.11889 0.20345 0.41959 0.85863 -0.60344 -0.31835 -0.6718 0.003984 -0.075159 0.11043 -0.73534 0.27436 0.054015 -0.23828 -0.13767 0.011573 -0.46623 -0.55233 0.083317 0.55938 0.51903 -0.27065 -0.28211 -1.3918 0.17498
he -0.20092 -0.060271 -0.61766 -0.8444 0.5781 0.14671 -0.86098 0.6705 -0.86556 -0.18234 0.15856 0.45814 -1.0163 -0.35874 0.73869 -0.24048 -0.33893 0.25742 -0.78192 0.083528 0.1775 0.91773 0.64531 -0.19896 0.37416 -2.7525 -0.091586 0.0403
as 0.20782 0.12713 -0.30188 -0.23125 0.30175 0.33194 -0.52776 -0.44042 -0.48348 0.03502 0.34782 0.54574 -0.2066 -0.083713 0.2462 0.15931 -0.0031349 0.32443 -0.4527 -0.22178 0.022652 -0.041714 0.31815 0.088633 -0.03801 -1.8212 -0.50917 -0
it 0.61183 -0.22072 -0.10898 -0.052967 0.50804 0.34684 -0.33558 -0.19152 -0.035865 0.1051 0.07935 0.2449 -0.4373 -0.33344 0.57479 0.69052 0.29713 0.090669 -0.54992 -0.46176 0.10113 -0.02024 0.28479 0.043512 0.45735 -2.0466 -0.58084 0.617
by 0.35215 -0.35603 0.25708 -0.10611 -0.20718 0.63596 -1.0129 -0.45964 -0.48749 -0.080555 0.43769 0.46046 -0.80943 -0.23336 0.46623 -0.10866 -0.1221 -0.63544 -0.73486 -0.24848 0.4317 0.092264 0.52033 -0.46784 0.016798 -1.5124 -0.19986 -0
at 0.27724 0.88469 -0.26247 0.084104 0.40813 -1.1697 -0.68522 0.1427 -0.57345 -0.58575 -0.50834 -0.86411 -0.52596 -0.56379 0.32862 0.43393 -0.21248 0.49365 -1.8137 -0.035741 1.3227 0.80865 0.012217 -0.087017 -0.16813 -1.5935 0.47034 0.26
( -0.24978 1.0476 0.21602 0.23278 0.12371 0.2761 0.51184 -1.36 -0.6902 -0.66679 0.49105 0.51671 -0.027218 -0.52056 0.49539 -0.097307 0.12779 0.44388 -1.2612 0.66209 -0.55461 -0.43498 0.81247 0.40855 -0.094327 -0.622 0.36498 -1.0038 -0.77
) -0.28314 1.0028 0.14746 0.22262 0.0070985 0.23108 0.57082 -1.2767 -0.72415 -0.7527 0.52624 0.39498 0.0018922 -0.39396 0.44859 -0.019057 0.068143 0.45082 -1.2849 0.68088 -0.48318 -0.45829 0.85504 0.47712 -0.16152 -0.74784 0.40742 -0.973
from 0.41037 0.11342 0.051524 -0.53833 -0.12913 0.22247 -0.9494 -0.18963 -0.36623 -0.067011 0.19356 -0.33044 0.11615 -0.58585 0.36106 0.12555 -0.3581 -0.023201 -1.2319 0.23383 0.71256 0.14824 0.50874 -0.12313 -0.20353 -1.82 0.22291 0.020
his -0.033537 0.47537 -0.68746 -0.72661 0.84028 0.64304 -0.75975 0.63242 -0.54176 0.11632 -0.20254 0.63321 -1.2677 -0.17674 0.35284 -0.55096 -0.65025 -0.3405 -0.31658 -0.077908 -0.11085 0.97299 -0.016844 -0.73752 0.47852 -2.7069 -0.42417
```

Demo of GloVe Vector

```
>>> glovelModel["person"]
result:
array([ 0.61734  ,  0.40035  ,  0.067786 , -0.34263  ,  2.0647   ,
        0.60844  ,  0.32558  ,  0.3869   ,  0.36906  ,  0.16553  ,
        0.0065053, -0.075674 ,  0.57099  ,  0.17314  ,  1.0142   ,
       -0.49581  , -0.38152  ,  0.49255  , -0.16737  , -0.33948  ,
       -0.44405  ,  0.77543  ,  0.20935  ,  0.6007   ,  0.86649  ,
       -1.8923   , -0.37901  , -0.28044  ,  0.64214  , -0.23549  ,
        2.9358   , -0.086004 , -0.14327  , -0.50161  ,  0.25291  ,
       -0.065446 ,  0.60768  ,  0.13984  ,  0.018135 , -0.34877  ,
        0.039985 ,  0.07943  ,  0.39318  ,  1.0562   , -0.23624  ,
       -0.4194   , -0.35332  , -0.15234  ,  0.62158  ,  0.79257  ])

>>> find_closest_embeddings(glovelModel["person"])[0:5]
result: ['person', 'someone', 'actually', 'every', 'knowing']
```

How do we design loss function and metrics?

Loss: Euclidean Distance



Reference:https://medium.com/analytics-vidhya/basics-of-using-pre-trained-glove-vectors-in-python-d38905f356db

BOSTON UNIVERSITY

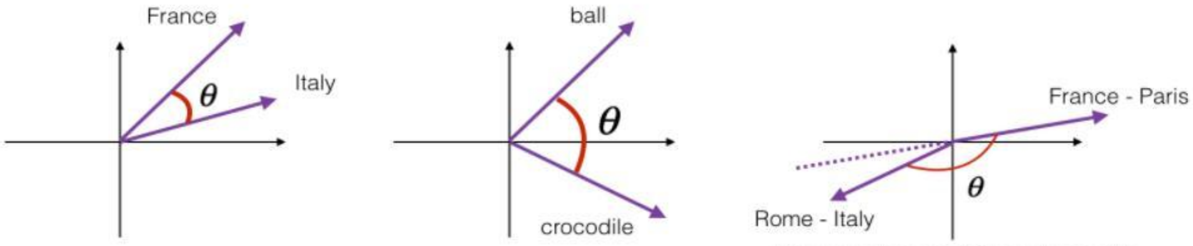
Metrics: Cosine similarity

1 - Cosine similarity

To measure how similar two words are, we need a way to measure the degree of similarity between two embedding vectors for the two words. Given two vectors u and v, cosine similarity is defined as follows:

$$\text{CosineSimilarity(u, v)} = \frac{u.\,v}{||u||_2||v||_2} = cos(\theta) \tag{1}$$

where $u.\,v$ is the dot product (or inner product) of two vectors, $||u||_2$ is the norm (or length) of the vector u, and θ is the angle between u and v. This similarity depends on the angle between u and v. If u and v are very similar, their cosine similarity will be close to 1; if they are dissimilar, the cosine similarity will take a smaller value.



France and Italy are quite similar
θ is close to 0°
$\cos(\theta) \approx 1$

ball and crocodile are not similar
θ is close to 90°
$\cos(\theta) \approx 0$

the two vectors are similar but opposite
the first one encodes (city - country)
while the second one encodes (country - city)
θ is close to 180°
$\cos(\theta) \approx -1$

Figure 1: The cosine of the angle between two vectors is a measure of how similar they are

Reference:https://datascience-enthusiast.com/DL/Operations_on_word_vectors.html

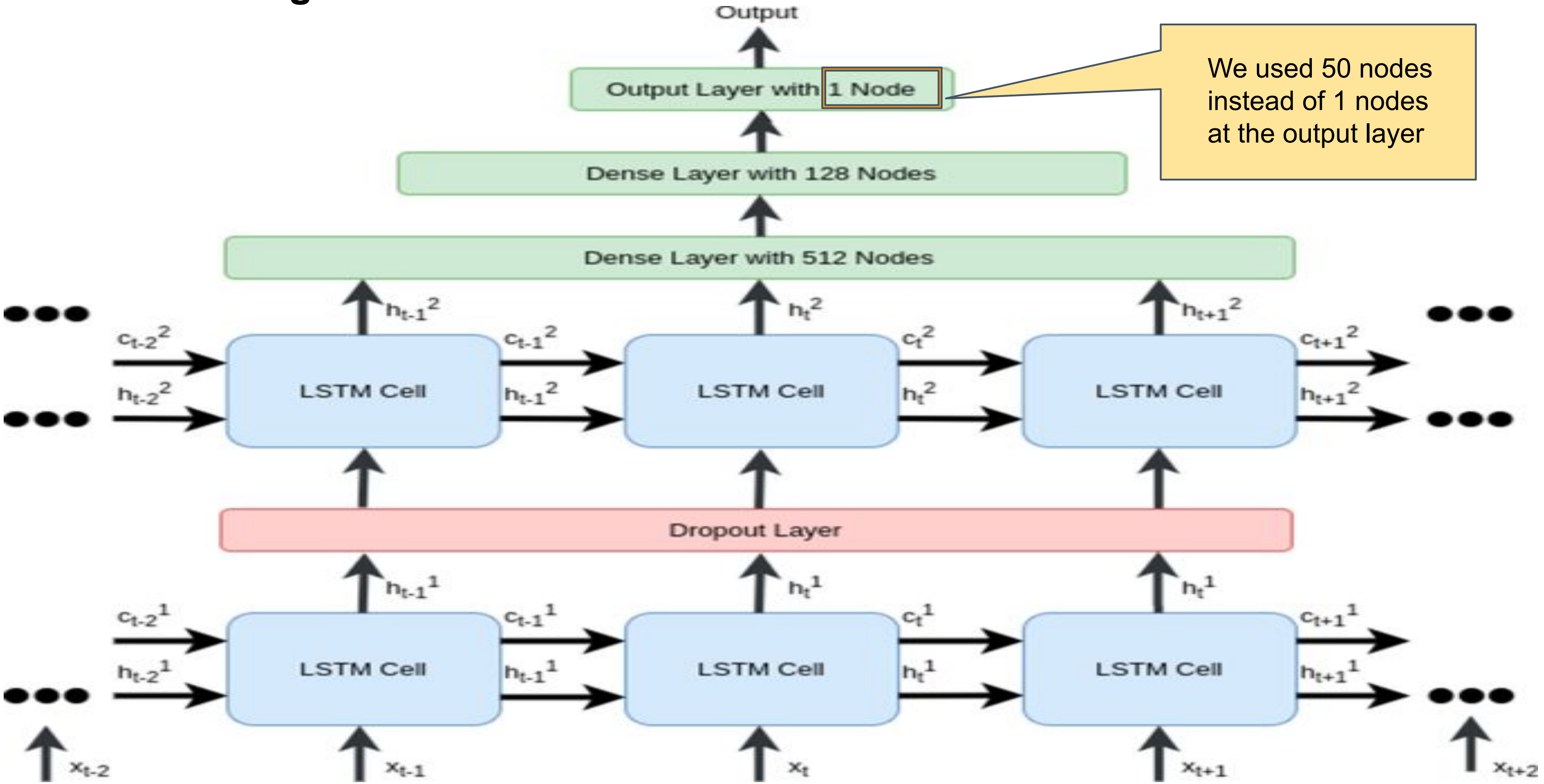
```python
"""demonstration of coscine similarity"""

a=glovelModel["person"]
temp=find_closest_embeddings(a)[0:5]
print("top 5 five words:", temp)
b=glovelModel[temp[0]]
c=glovelModel[temp[1]]
d=glovelModel[temp[2]]
print("")
print("similarity between a and b:",a@b/(np.linalg.norm(a)*np.linalg.norm(b)))
print("distance between a and b:",euclidean_distance_loss(a,b).numpy())
print("")
print("similarity between a and c:",a@c/(np.linalg.norm(a)*np.linalg.norm(c)))
print("distance between a and c:",euclidean_distance_loss(a,c).numpy())
print("")
print("similarity between a and d:",a@d/(np.linalg.norm(a)*np.linalg.norm(d)))
print("distance between a and d:",euclidean_distance_loss(a,d).numpy())
```

```
similarity between a and b: 1.0
distance between a and b: 0.0

similarity between a and c: 0.8526201331553841
distance between a and c: 2.843090933653908

similarity between a and d: 0.8010346183707522
distance between a and d: 3.0690672667375813

>>>
```

Training Practices

- Architecture Design
- Regularization and optimization techniques
- Training results

Architecture Design



We used 50 nodes instead of 1 nodes at the output layer

Reference:https://bmcmedinformdecismak.biomedcentral.com/articles/10.1186/s12911-021-01462-5

List of techniques used

Regularization

1. Dropout
2. Early Stopping

Optimization

1. Rmsprop
2. Momentum
3. Learning rate exponentially decay

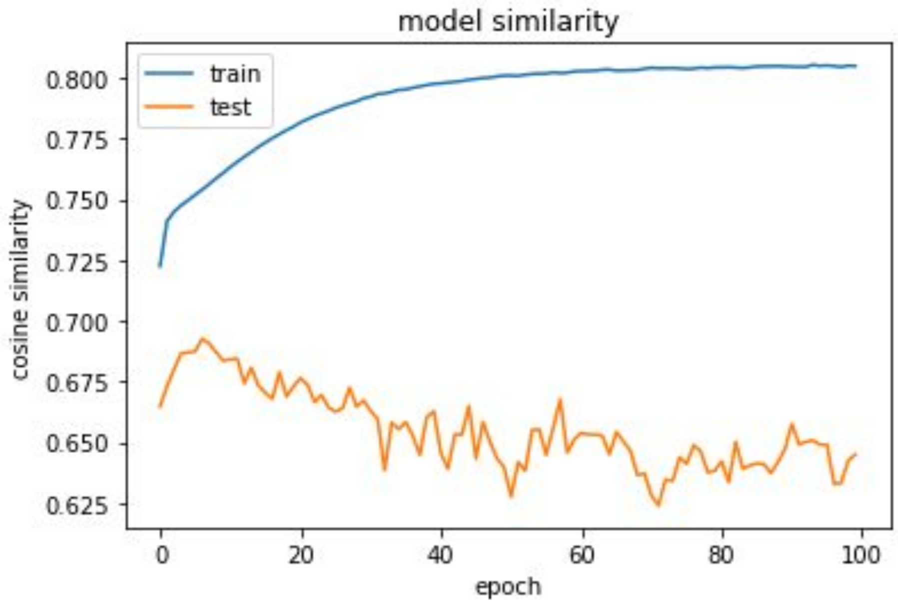
Results

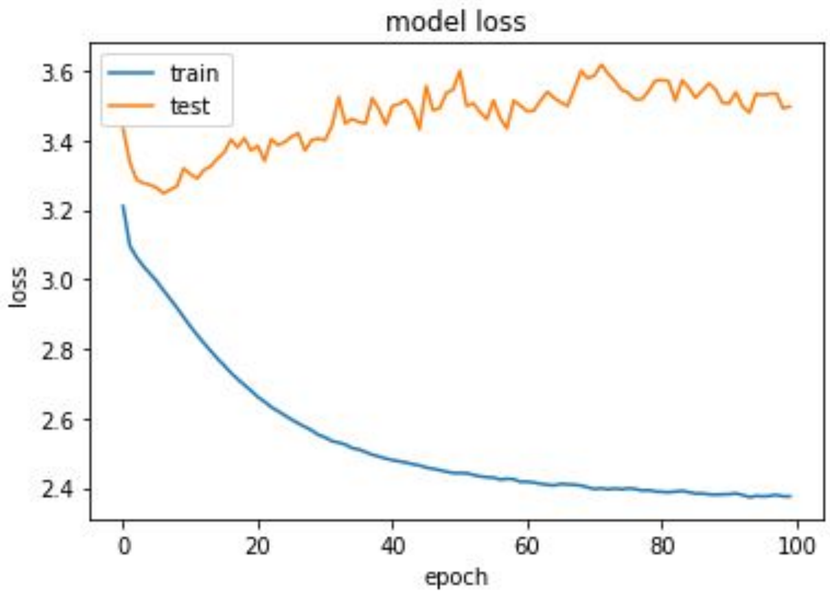
Training result for features constructed by **One-Hot** encoding

Bad 

```
Epoch 9/50
1006/1006 [==============================] - 65s 64ms/step - loss: 1.5202 - categorical_accuracy: 0.1103  val_loss: 11.0
379 - val_categorical_accuracy: 0.0562
Epoch 10/50
1006/1006 [==============================] - 65s 65ms/step - loss: 1.2803 - categorical_accuracy: 0.1232  val_loss: 11.4
638 - val_categorical_accuracy: 0.0543
Epoch 11/50
1006/1006 [==============================] - 65s 65ms/step - loss: 1.1266 - categorical_accuracy: 0.1340  val_loss: 11.5
618 - val_categorical_accuracy: 0.0504
Epoch 12/50
1006/1006 [==============================] - 65s 64ms/step - loss: 1.0187 - categorical_accuracy: 0.1428  val_loss: 11.9
797 - val_categorical_accuracy: 0.0504
Epoch 13/50
1006/1006 [==============================] - 64s 64ms/step - loss: 0.9279 - categorical_accuracy: 0.1510  val_loss: 12.2
054 - val_categorical_accuracy: 0.0581
Epoch 14/50
1006/1006 [==============================] - 65s 65ms/step - loss: 0.8576 - categorical_accuracy: 0.1591  val_loss: 12.4
238 - val_categorical_accuracy: 0.0543
Epoch 15/50
1006/1006 [==============================] - 66s 65ms/step - loss: 0.7961 - categorical_accuracy: 0.1660  val_loss: 12.2
790 - val_categorical_accuracy: 0.0504
Epoch 16/50
1006/1006 [==============================] - 65s 64ms/step - loss: 0.7501 - categorical_accuracy: 0.1728  val_loss: 12.6
762 - val_categorical_accuracy: 0.0543
Epoch 17/50
1006/1006 [==============================] - 65s 65ms/step - loss: 0.7050 - categorical_accuracy: 0.1776  val_loss: 12.8
732 - val_categorical_accuracy: 0.0514
Epoch 18/50
1006/1006 [==============================] - 65s 65ms/step - loss: 0.6758 - categorical_accuracy: 0.1822  val_loss: 12.8
780 - val_categorical_accuracy: 0.0552
Epoch 19/50
1006/1006 [==============================] - 65s 65ms/step - loss: 0.6637 - categorical_accuracy: 0.1875  val_loss: 13.0
222 - val_categorical_accuracy: 0.0552
Epoch 20/50
1006/1006 [==============================] - 65s 64ms/step - loss: 0.6412 - categorical_accuracy: 0.1919  val_loss: 13.0
035 - val_categorical_accuracy: 0.0523
Epoch 21/50
1006/1006 [==============================] - 65s 65ms/step - loss: 0.6141 - categorical_accuracy: 0.1964  val_loss: 13.1
320 - val_categorical_accuracy: 0.0514
Epoch 22/50
1006/1006 [==============================] - 65s 65ms/step - loss: 0.5978 - categorical_accuracy: 0.2011  val_loss: 13.3
441 - val_categorical_accuracy: 0.0514
Epoch 23/50
1006/1006 [==============================] - 65s 65ms/step - loss: 0.5781 - categorical_accuracy: 0.2044  val_loss: 13.4
474 - val_categorical_accuracy: 0.0533
Epoch 24/50
1006/1006 [==============================] - 64s 64ms/step - loss: 0.5643 - categorical_accuracy: 0.2081  val_loss: 13.4
518 - val_categorical_accuracy: 0.0523
Epoch 25/50
```

Results of GloVe vector

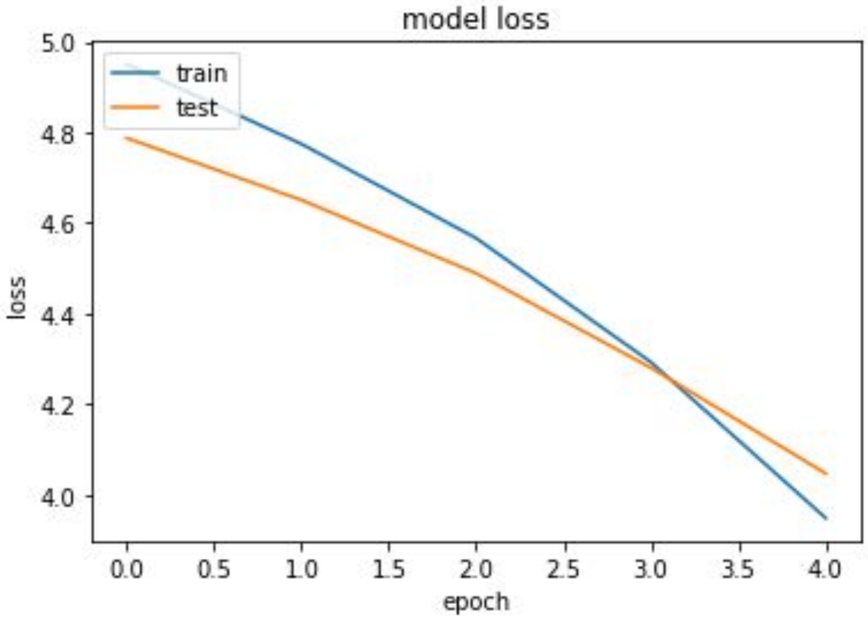


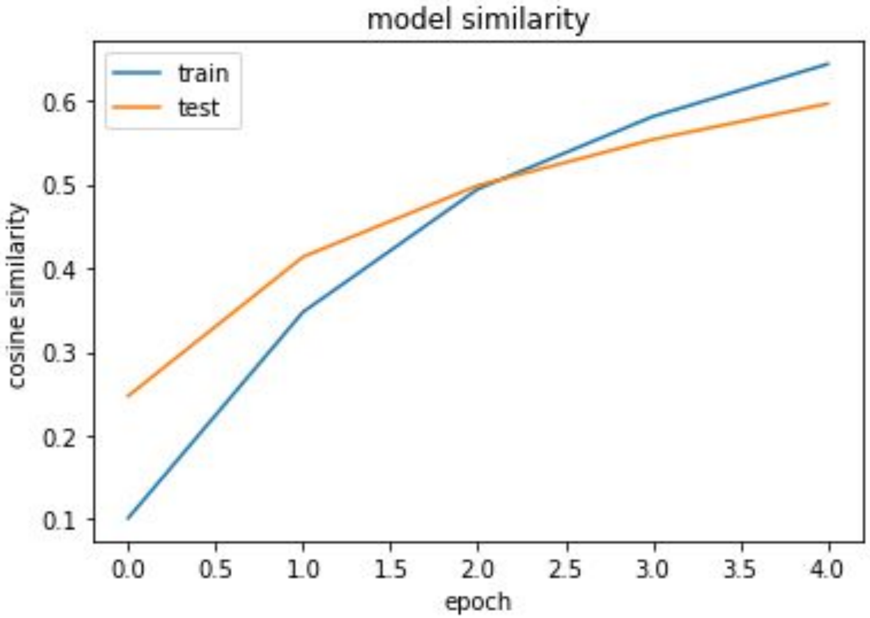


Console 5/A Console 6/A

```
Epoch 1/100
107434/107434 [==============================] - 15s 143us/sample - loss:
3.2095 - cosine_similarity: 0.7225 - val_loss: 3.4328 -
val_cosine_similarity: 0.6647
Epoch 2/100
107434/107434 [==============================] - 9s 86us/sample - loss:
3.0954 - cosine_similarity: 0.7411 - val_loss: 3.3356 -
val_cosine_similarity: 0.6731
Epoch 3/100
107434/107434 [==============================] - 10s 97us/sample - loss:
3.0627 - cosine_similarity: 0.7450 - val_loss: 3.2863 -
val_cosine_similarity: 0.6801
Epoch 4/100
107434/107434 [==============================] - 11s 104us/sample - loss:
3.0376 - cosine_similarity: 0.7476 - val_loss: 3.2764 -
val_cosine_similarity: 0.6866
Epoch 5/100
107434/107434 [==============================] - 10s 90us/sample - loss:
3.0156 - cosine_similarity: 0.7497 - val_loss: 3.2708 -
val_cosine_similarity: 0.6870
Epoch 6/100
107434/107434 [==============================] - 10s 96us/sample - loss:
2.9934 - cosine_similarity: 0.7519 - val_loss: 3.2628 -
val_cosine_similarity: 0.6874
Epoch 7/100
107434/107434 [==============================] - 10s 97us/sample - loss:
2.9677 - cosine_similarity: 0.7541 - val_loss: 3.2469 -
val_cosine_similarity: 0.6926
Epoch 8/100
107434/107434 [==============================] - 10s 97us/sample - loss:
2.9426 - cosine_similarity: 0.7563 - val_loss: 3.2570 -
val_cosine_similarity: 0.6907
Epoch 9/100
107434/107434 [==============================] - 11s 107us/sample - loss:
2.9170 - cosine_similarity: 0.7587 - val_loss: 3.2678 -
val_cosine_similarity: 0.6871
Epoch 10/100
107434/107434 [==============================] - 11s 104us/sample - loss:
```

if lr=0.00001


model loss
train
test
loss
5.0
4.8
4.6
4.4
4.2
4.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
epoch


model similarity
train
test
cosine similarity
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
epoch

Testing

Prediction for 9 randomly chosen samples

Note: the context of this application is for words we have typed before, so input that has not been seen before is not very important

```
>>> for i in range(9):
...:     x=X[1000+i] #(5,50)
...:     x=np.reshape(x,(1,5,50))
...:     y_hat=model.predict(x)
...:     print("predictions are: ", find_closest_embeddings(y_hat)[0:5])
...:     print("true label is: ", find_closest_embeddings(Y[1000+i])[0])
...:     print("")
predictions are:  ['turning', 'presumably', 'unfortunately', 'sight',
'keeps']
true label is:  mess

predictions are:  ['.', 'but', 'as', 'same', 'though']
true label is:  but

predictions are:  ['.', 'as', 'same', 'one', 'well']
true label is:  as

predictions are:  ['i', 'me', "'d", 'never', 'know']
true label is:  i

predictions are:  ['have', 'those', 'some', 'are', 'already']
true label is:  have

predictions are:  ['but', 'though', '.', 'once', 'still']
true label is:  changed

predictions are:  ['my', 'me', 'i', "'d", 'you']
true label is:  my

predictions are:  ['hand', 'touch', 'eyes', 'little', 'hands']
true label is:  clothes

predictions are:  ['i', 'me', "'d", "n't", 'know']
true label is:  i
```

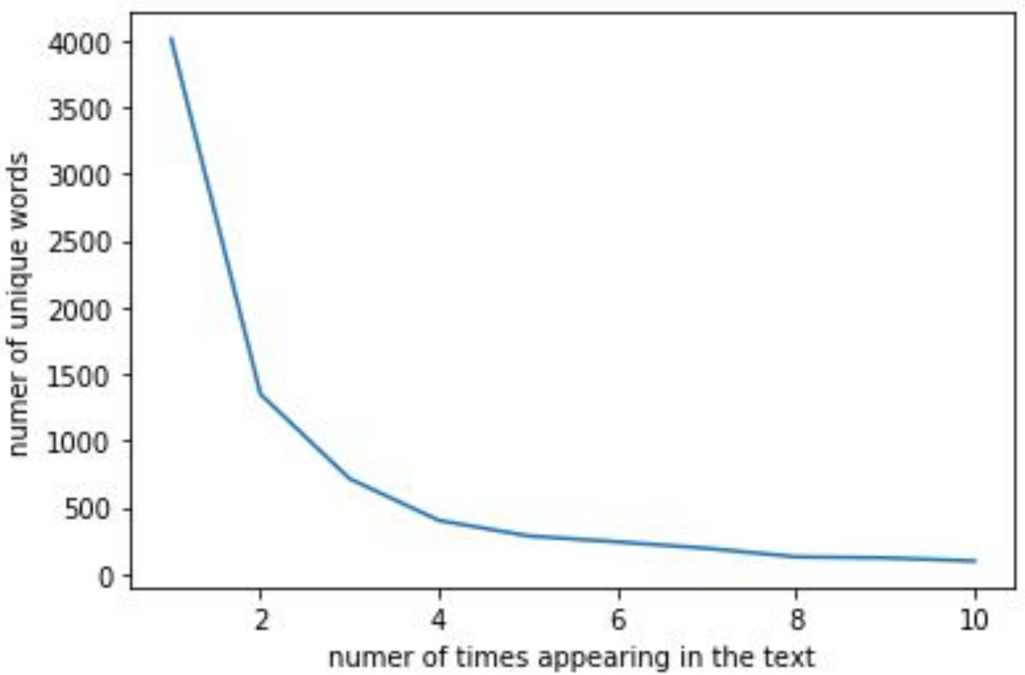

BOSTON UNIVERSITY

Future Work

- Improvements:
  - Data augmentation, larger data size, weighted loss for solving “imbalanced classes” issue
  - Learn feature vectors on our own (through embedding layer)
  - Punctuations

- Possible Applications:
  - Next word suggestion (as shown before)
  - Many to many translation (requires decoder network)
  - Auto Correction (character-level based)

Note: We only have 8000+ unique words in total



Thanks for listening!

Q&A section