Keyboard Autocomplete & Autocorrect

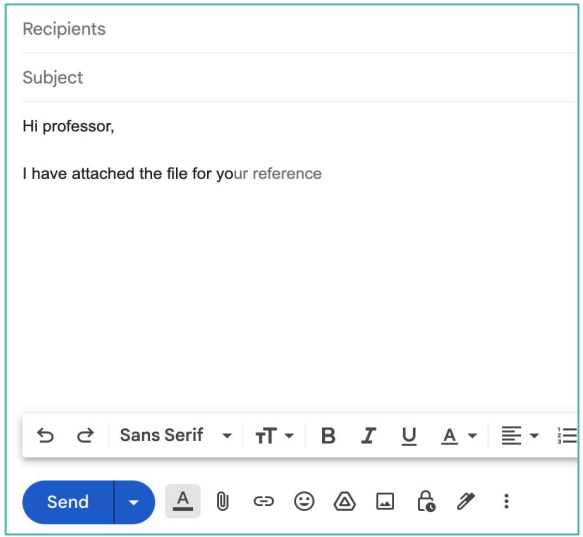
Shide Pu, Shiwei Yu, Xiaomeng Chen,
Hongyu Zhao

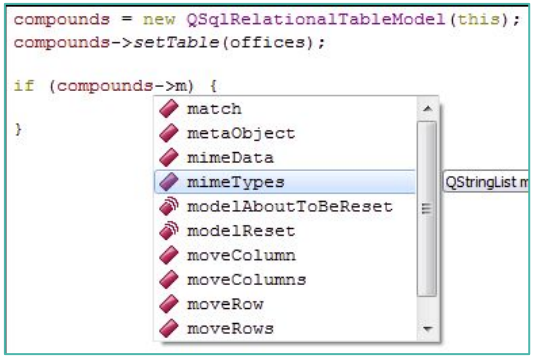
Agenda

- Introduction & purpose
- Dataset
- Autocomplete
  - Doc2vec and Trie
  - Embedding match
  - LSTM
  - NNLM & n-gram
  - Text RNN with attention
- Autocorrect
  - LSTM
  - Bayesian with BOW
- Conclusion

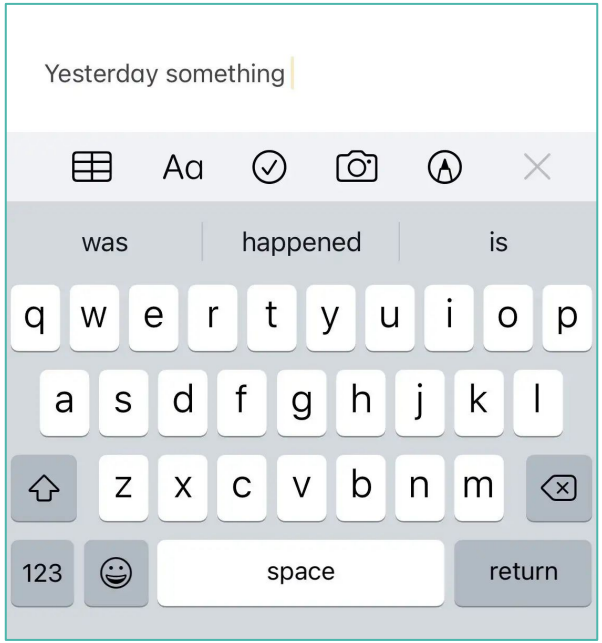
Usage by Software

- In web browsers
- In E-mail programs
- In search engines
- In code editors





Introduction and Purpose

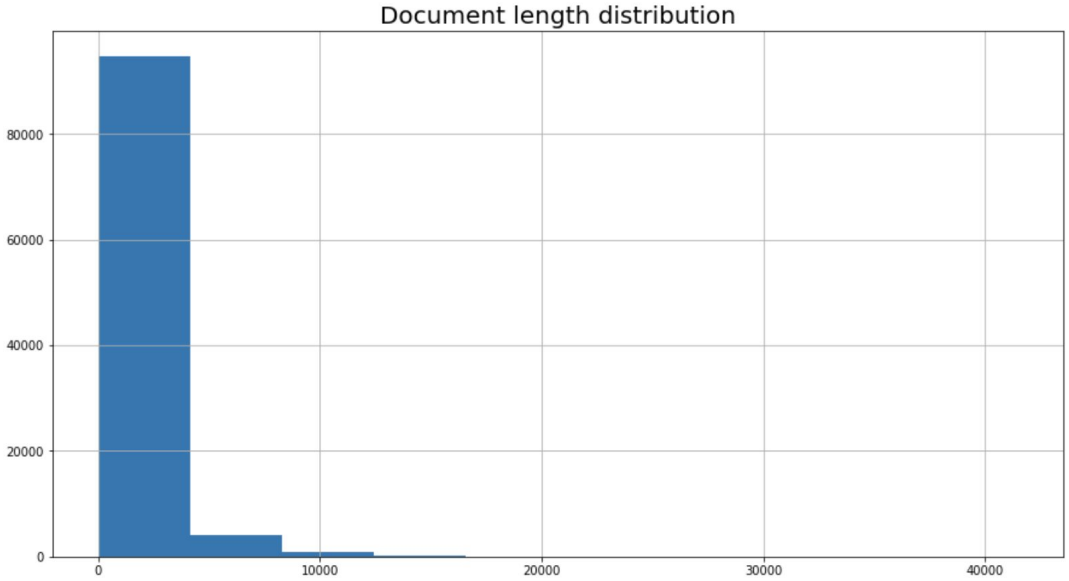


Autocomplete is a feature in which an application **predicts the rest of a word** a user is typing. In graphical user interfaces, users can typically press the tab key to accept a suggestion or the down arrow key to accept one of several.

Autocorrection is an automatic data validation function commonly found in word processors and text editing interfaces. Its principal purpose is as part of the spell checker to **correct common spelling** or **typing errors**, saving time for the user.

Purpose: To understand how it works, we will learn Natural Language Processing in this project and then we will use Python to build the autocomplete & autocorrection feature.

Dataset

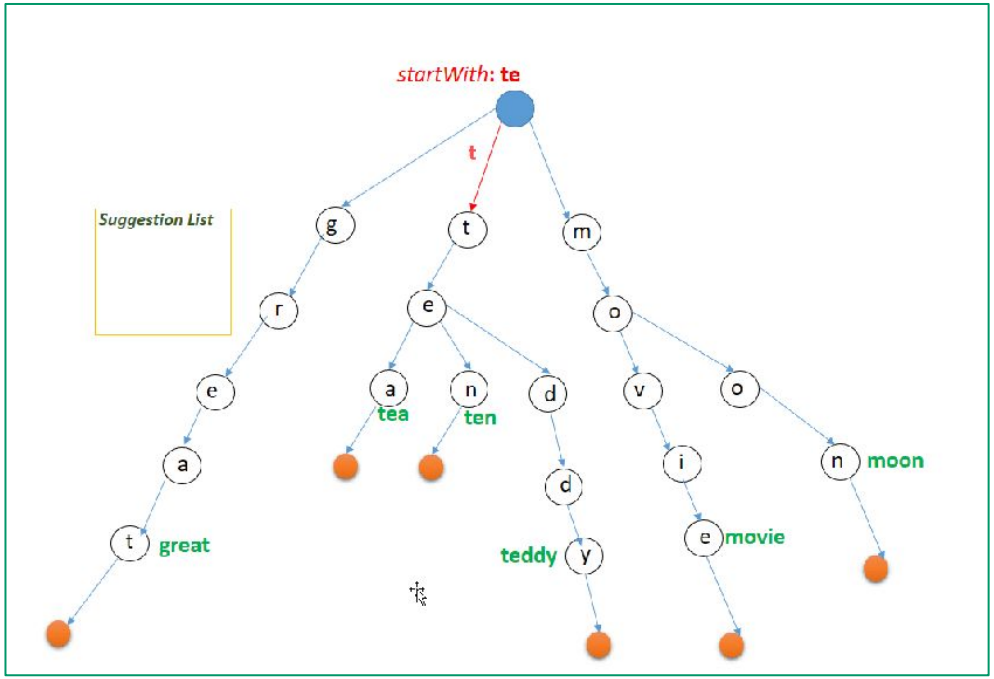


This project will use **Wikipedia data sets** from Hugging Face which contain cleaned articles of all languages and are built from the Wikipedia dump with one split per language. Each example contains the content of one full Wikipedia article with cleaning to strip markdown and unwanted sections. The pre-processed English subset is around 20275.52 MB. We randomly select **100,000 documents**, around **720mb**, to train the model.

Doc2vec & Trie

Trie, aka digital tree or prefix tree

- k-ary search tree
- Search all strings with the input prefix
- Links between nodes are defined by individual characters
- Traversed depth-first search, O(n)
- For example: if the Trie store {“tea”, “ten”, “teddy”, “movie”} and the user types in “te” then he must be shown {“tea”, “ten”, “teddy”}.



Doc2vec & Trie

Task: Find appropriate documents to generate a set of words to build a trie

Assume that similar documents are likely to contain similar word patterns.

- Find similar training documents to the input document

Doc2vec

- Vector size = 32
- Store word embedding in memory. Limit the size of the model

Doc2vec & Trie - Result

```
autocomplete("Anarchism is a political philosophy and mov", 200)
```

```
movement
movements
moving
```

```
autocomplete("April is the fourth month of the year in the Julian and Gregorian cal", 200)
```

```
cal
caleb
california
caliph
caliphate
call
called
calleja
calling
caldera
calderón
calhoun
```

```
autocomplete("Natural language processing is a subfield of linguistics, compu")
```

```
compute
computer
computers
computerworld
computerized
computed
computes
computation
computational
computationally
computations
computability
computable
computing
compuflo
```

Doc2vec & Trie - Summary

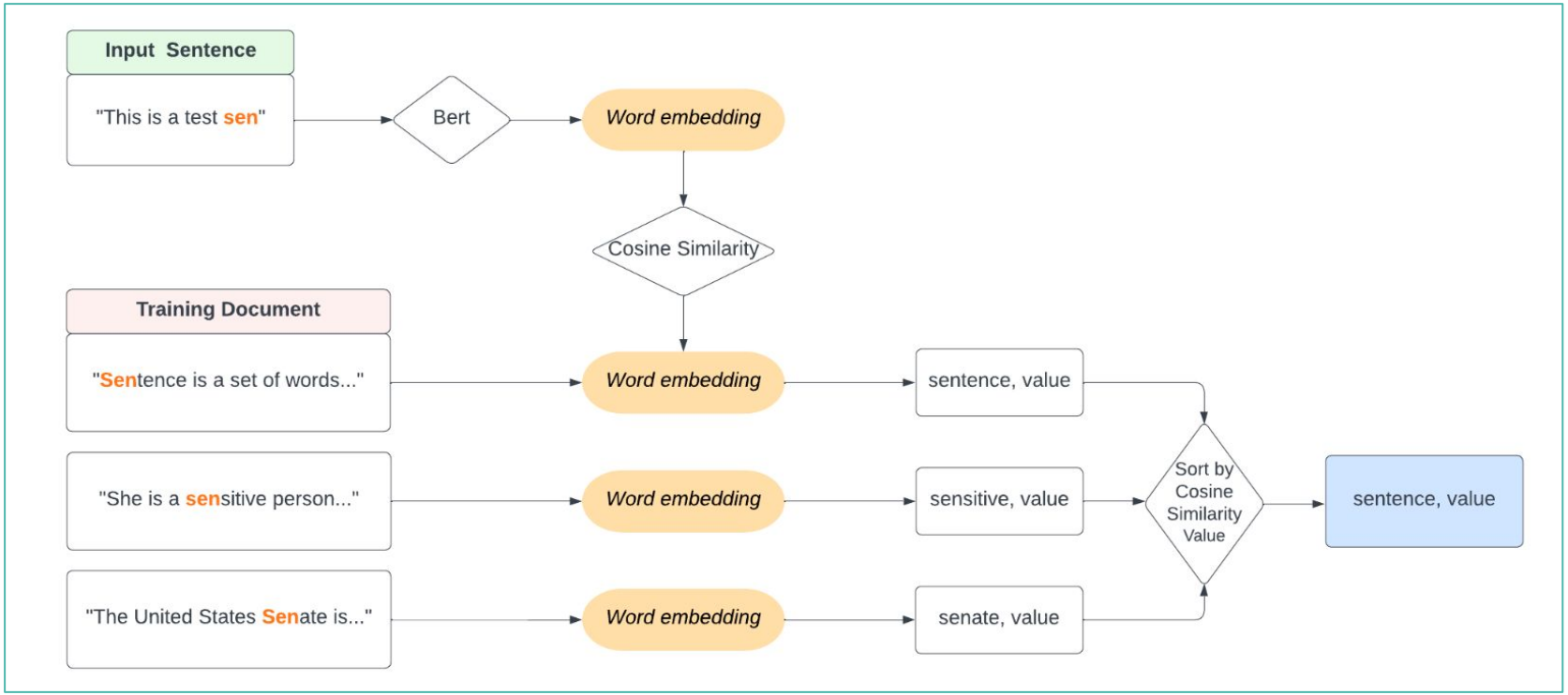
- Advantage:
  - Prediction is very Fast
  - Easy to implement
- Disadvantage:
  - **Simple listing of all possible words without ranking**
  - Depends on the performance of Doc2vec
  - May not find result
  - Model size is large
    - 32 vector size → 170 MB
    - 128 vector size → 800 MB

Embedding Match

- Split 20000 documents into sentences
  - About 1.1 million sentences
  - 140 MB

	sentence
0	adrienne mayor is a historian of ancient scien...
1	mayor specializes in ancient history and the s...
2	her work in pre scientific fossil discoveries ...
3	mayor book, greek fire, poison arrows, the sco...
4	lifefrom number to number, she worked as a cop...
5	since number, mayor has been a research schola...
6	mayor has published books and articles on the ...
7	her books the first fossil hunters and fossil ...
8	feder book frauds, myths, and mysteries scienc...
9	her books have been translated into french, ge...

Embedding Match - Process



Embedding Match - Result

```
text = "That which does not kill us make us str"
bert_autocomplete.autocomplete(text)
```

```
100%|██████████| 200/200 [00:35<00:00,  5.63it/s]
[('strangely', 0.5754252),
 ('strong', 0.5282924),
 ('street', 0.5196317),
 ('strategy', 0.4686629),
 ('structure', 0.44376093)]
```

```
text = "Natural language processing is a subfield of linguistics, computer sci"
bert_autocomplete.autocomplete(text)
```

```
100%|██████████| 200/200 [00:31<00:00,  6.41it/s]
[('science', 0.6582532),
 ('sciences', 0.6273753),
 ('scientist', 0.53728306),
 ('scissors', 0.5202313),
 ('scientists', 0.51241314)]
```

Very slow!!!

About 40 seconds to search 200 documents

Embedding Match - Result

```
bert_autocomplete.autocomplete("t")
```

```
100%|██████████| 200/200 [00:31<00:00,  6.40it/s]
[('technology', 0.59731483),
 ('thought', 0.5147018),
 ('tactics', 0.51239157),
 ('terminology', 0.45969442),
 ('together', 0.4554178)]
```

```
bert_autocomplete.autocomplete("p")
```

```
100%|██████████| 200/200 [00:21<00:00,  9.39it/s]
[('point', 0.5408966),
 ('points', 0.5329482),
 ('p', 0.51109064),
 ('perform', 0.50498146),
 ('party', 0.47670937)]
```

Able to complete single character

Embedding Match - Edge Case

- If the word is not contained in the tokenizer, we cannot get word cosine similarity
  - “vrindavana”, “subjunctive”
- Collect words from training documents as result

```
# correct word: vrindavana
text = "kameshvara temple, in kamyavan, one of the twelve forests of vrind"
bert_autocomplete.autocomplete(text, similarity=False)
```

```
100%|██████████| 10/10 [00:02<00:00, 3.56it/s]
['vrindavanam', 'vrindavan', 'vrindabanin', 'vrindaban', 'vrindavana']
```

```
# correct word: subjunctive
text = "some important ones are declarative, affirmative, negative, emphatic, conditional, imperative, interrogative and subju"
bert_autocomplete.autocomplete(text, similarity=False)
```

```
100%|██████████| 88/88 [00:11<00:00, 7.51it/s]
['subjugated', 'subjugation', 'subjunctive', 'subjugating', 'subjugate']
```

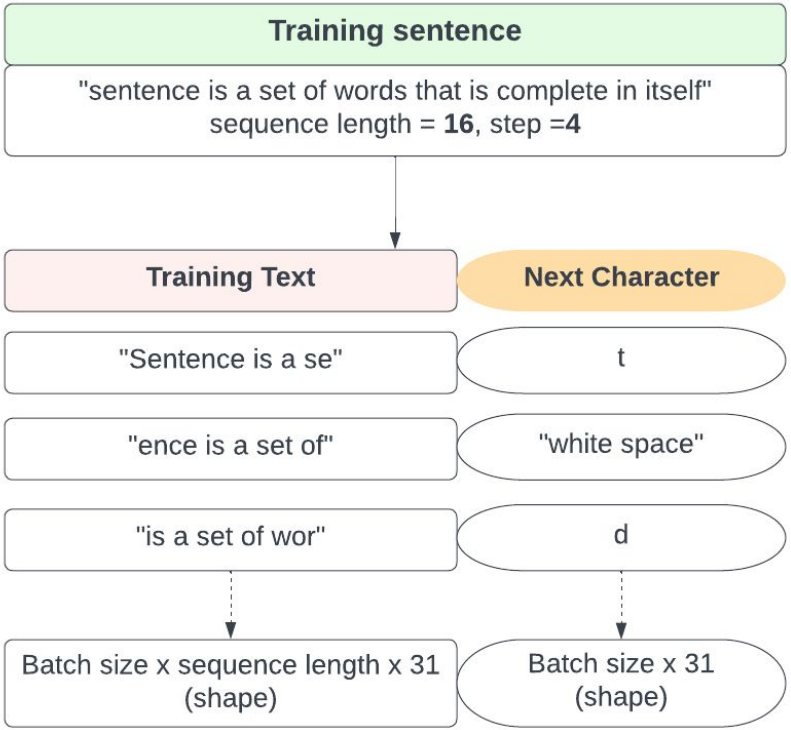
Embedding Match - Summary

- Advantage:
  - Prediction results are good
  - Complete input sentences with any lengths
- Disadvantage:
  - May not have result
  - **Slow, a single prediction takes about 40 seconds in Colab**
  - To speed up prediction, cache word embeddings into memory
  - Require a large amount of memory
    - 1,000,000 × 15 × 768 × 4 bytes = 43 GB

LSTM

- Receive incomplete words as input and predict the next character
- Bag of characters : [' ', '!', ',', '.', '?', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j','k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
  - 26 English characters, 4 punctuation marks, 1 **white space**
  - Dimension of character is **31**
- White space is very important:
  - Indicates end of a word
  - Model continually makes predictions until hits white space

LSTM - Generate Training Data

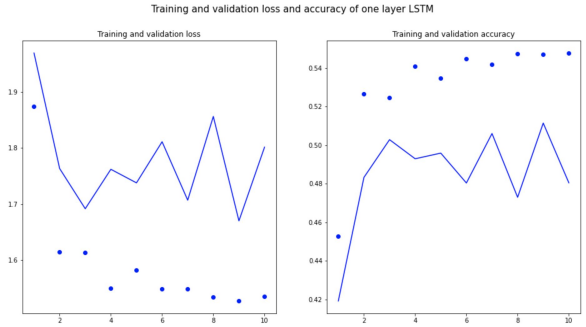


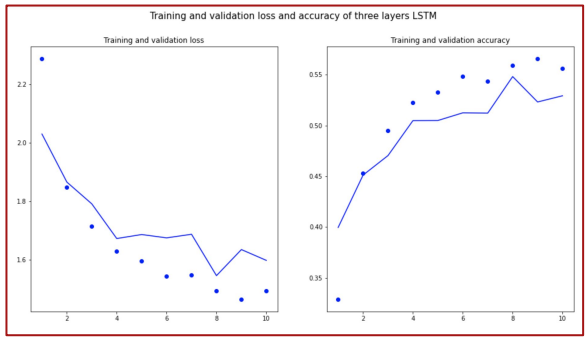
- Training documents: **fixed length sequences**
  - sequence_length: length of training sequence
  - step: number of characters jump between two sequences
  - sequence_length=32, step=4
- Prediction documents: **next characters**

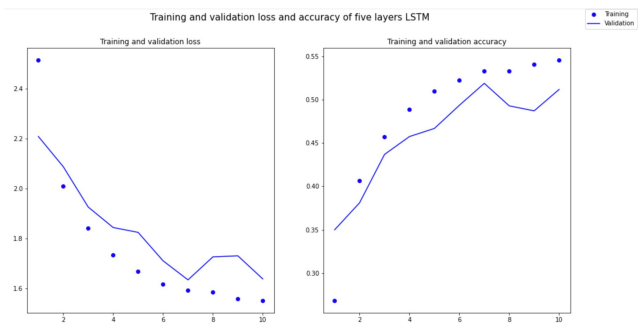
LSTM - Model Selection

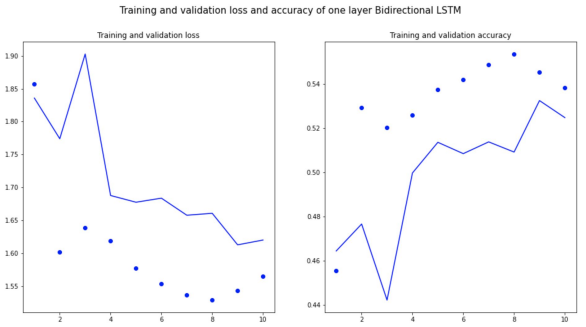
LSTM Structure	Training Loss, Accuracy	Validation Loss, Accuracy
1 layer LSTM	1.5361 , 0.5478	1.8017 , 0.4805
3 layer LSTM	**1.4943 , 0.5563**	**1.5986 , 0.5293**
5 layer LSTM	1.5511 , 0.5456	1.6383 , 0.5117
1 layer Bidirectional LSTM	1.5648 , 0.5383	1.6199 , 0.5248
2 layer Bidirectional LSTM	1.5076 , 0.5550	1.6408 , 0.5246
1 layer of LSTM and 1 layer Bidirectional LSTM	1.4854 , 0.5606	1.6576 , 0.5084

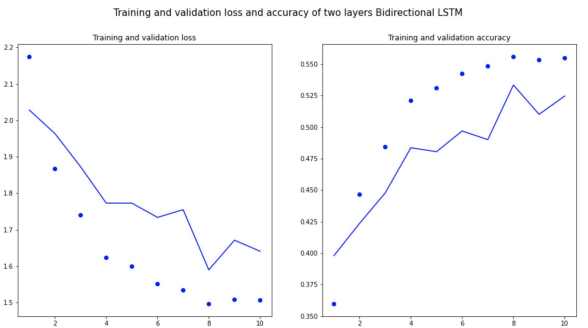
LSTM - Model Selection

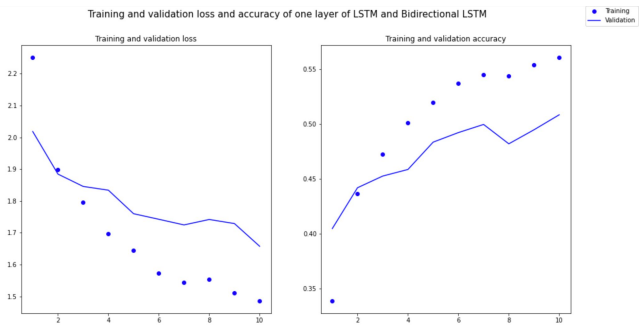












LSTM - Result

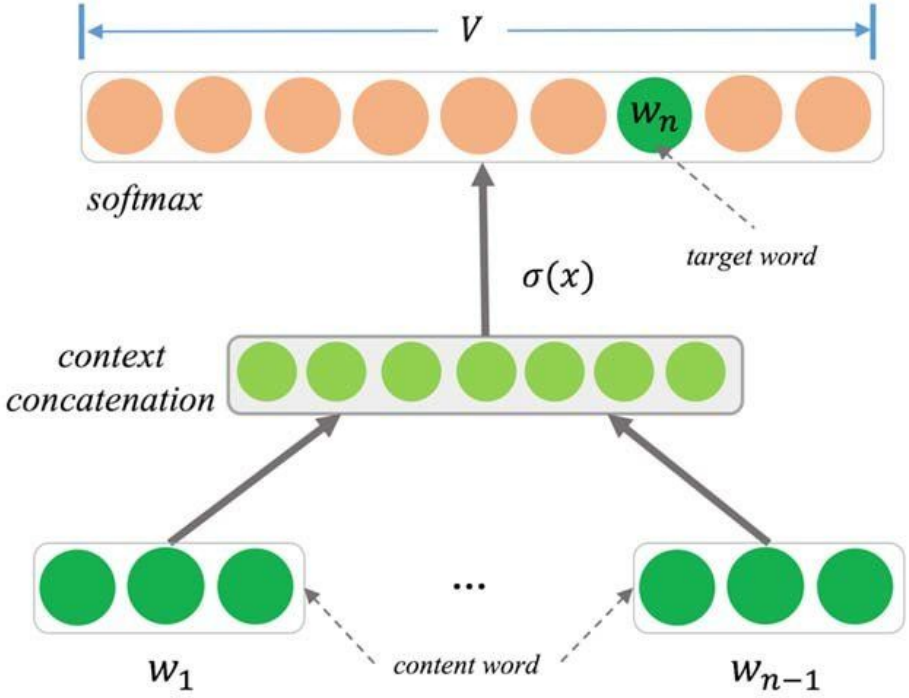
```
sentence_example = ["This is an incomplete example se",
                    "Sensitivity refers to the probab",
                    "Natural language processing is a subfield of linguistics, computer sci",
                    "Mayor specializes in ancient history",
                    "Since number, mayor has been a research",
                    "Sentence is a set of words that is comp",
                    "Machine learning is a field of inquiry devoted to und"]
for sentence in sentence_example:
        word_completion.autocomplete(sentence, 5)
```

```
This is an incomplete example se
['rved ', 'cret ', 'en ', 'ason ', 't ']
Sensitivity refers to the probab
['ly ', ' and ', 'ily ', 'e ', 'ory ']
Natural language processing is a subfield of linguistics, computer sci
['ence ', 'le ', 'te ', 'nce ', 're ']
Mayor specializes in ancient history
[' of ', ', ', '. ', 'ing ', 's ']
Since number, mayor has been a research
[' and ', 'er ', ', ', 'ing ', '. ']
Sentence is a set of words that is comp
['any ', 'leted ', 'eted ', 'osed ', 'rised ']
Machine learning is a field of inquiry devoted to und
['er ', 'ive ', 'ar ', 'ourgh ', 'ust ']
```

3-layer LSTM - Summary

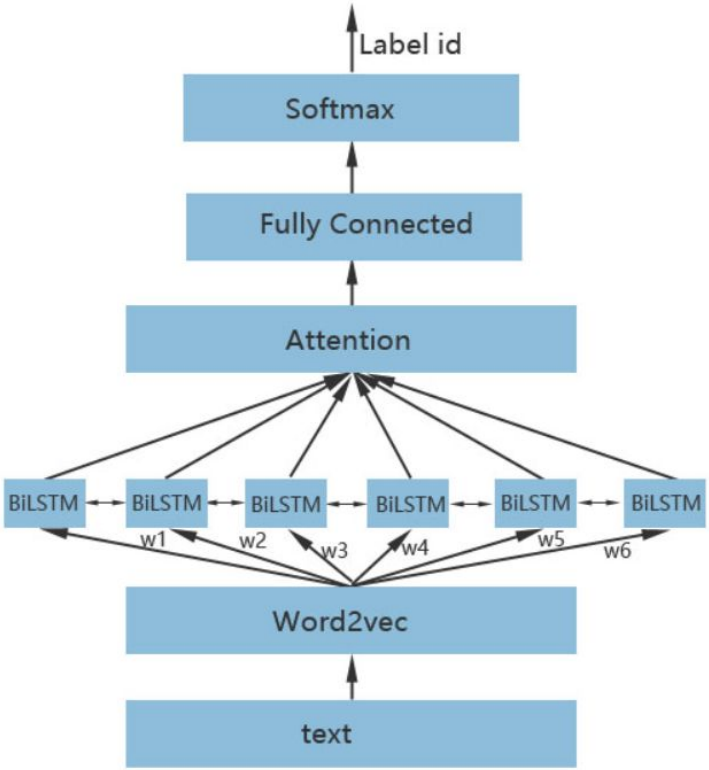
- Advantage:
  - Training is efficient
    - Five layer LSTM takes 10 minutes
  - Definitely generate prediction results
  - Effectively complete words based on prefix
  - Some ability to predict next words
- Disadvantage:
  - **If the input sentence is less than the sequence length, predictions are bad**
  - The input sentence can only contains characters in the set

NNLM & ngram



- In the two models of Neural Network Language Model (NNLM) and n-gram, I perform word embedding layer followed by fully connected layer and activation function.
    - Due to its fully connected characteristics, the parameters of the general fully connected layer are also the most.
- Then, we take the classification with maximum probability for all softmax function.

Text RNN with attention



- We connect the TextRNN model to a layer of RNN after the embedding layer, and then classify it.
- It is worth noting that we also introduced the concept of attention, which is to add a layer of attention to the TextRNN model.
- Attention is similar to human observation of other things. When people observe something, they generally do not treat it as a whole, but tend to selectively obtain important parts according to their own needs.

Delicious food and the service was great

Results of 4 models in characters

```
main_alpha('NNLM_alpha')
main_alpha('ngram_alpha')
main_alpha('TextRnn_alpha')
main_alpha('TextRnnAttention_alpha')
test_loss_models_alpha
test_ppl_models_alpha
```

```
[train epoch 0] loss: 2.931 ppl: 18.737 : 100%| 45100/45100 [02:35<00:00, 290.29it/s]
[valid epoch 0] loss: 2.561 ppl: 12.944 : 100%| 33418/33418 [00:11<00:00, 2861.03it/s]
[train epoch 1] loss: 2.503 ppl: 12.217 : 100%| 45100/45100 [02:34<00:00, 291.54it/s]
[valid epoch 1] loss: 2.434 ppl: 11.400 : 100%| 33418/33418 [00:12<00:00, 2740.05it/s]
[train epoch 2] loss: 2.421 ppl: 11.259 : 100%| 45100/45100 [02:32<00:00, 295.65it/s]
[valid epoch 2] loss: 2.406 ppl: 11.087 : 100%| 33418/33418 [00:11<00:00, 2859.71it/s]
[valid epoch 2] loss: 2.416 ppl: 11.204 : 100%| 28570/28570 [00:10<00:00, 2807.30it/s]
[train epoch 0] loss: 2.623 ppl: 13.770 : 100%| 45100/45100 [04:00<00:00, 187.55it/s]
[valid epoch 0] loss: 2.642 ppl: 14.038 : 100%| 33418/33418 [00:10<00:00, 3146.62it/s]
[train epoch 1] loss: 2.670 ppl: 14.434 : 100%| 45100/45100 [04:55<00:00, 152.74it/s]
[valid epoch 1] loss: 2.812 ppl: 16.635 : 100%| 33418/33418 [00:10<00:00, 3112.12it/s]
[train epoch 2] loss: 2.745 ppl: 15.564 : 100%| 45100/45100 [05:21<00:00, 140.31it/s]
[valid epoch 2] loss: 2.769 ppl: 15.944 : 100%| 33418/33418 [00:10<00:00, 3124.31it/s]
[valid epoch 2] loss: 2.766 ppl: 15.892 : 100%| 28570/28570 [00:09<00:00, 3014.08it/s]
[train epoch 0] loss: 2.704 ppl: 14.937 : 100%| 45100/45100 [02:21<00:00, 317.82it/s]
[valid epoch 0] loss: 2.597 ppl: 13.424 : 100%| 33418/33418 [00:18<00:00, 1842.24it/s]
[train epoch 1] loss: 2.605 ppl: 13.529 : 100%| 45100/45100 [02:44<00:00, 273.73it/s]
[valid epoch 1] loss: 2.601 ppl: 13.478 : 100%| 33418/33418 [00:18<00:00, 1778.12it/s]
[train epoch 2] loss: 2.612 ppl: 13.625 : 100%| 45100/45100 [02:54<00:00, 257.85it/s]
[valid epoch 2] loss: 2.610 ppl: 13.600 : 100%| 33418/33418 [00:19<00:00, 1731.96it/s]
[valid epoch 2] loss: 2.610 ppl: 13.596 : 100%| 28570/28570 [00:15<00:00, 1861.42it/s]
[train epoch 0] loss: 2.681 ppl: 14.604 : 100%| 45100/45100 [02:37<00:00, 286.15it/s]
[valid epoch 0] loss: 2.563 ppl: 12.971 : 100%| 33418/33418 [00:26<00:00, 1277.21it/s]
[train epoch 1] loss: 2.571 ppl: 13.074 : 100%| 45100/45100 [02:55<00:00, 256.35it/s]
[valid epoch 1] loss: 2.567 ppl: 13.031 : 100%| 33418/33418 [00:26<00:00, 1265.73it/s]
[train epoch 2] loss: 2.564 ppl: 12.986 : 100%| 45100/45100 [03:20<00:00, 225.37it/s]
[valid epoch 2] loss: 2.556 ppl: 12.888 : 100%| 33418/33418 [00:22<00:00, 1491.54it/s]
[valid epoch 2] loss: 2.559 ppl: 12.923 : 100%| 28570/28570 [00:16<00:00, 1690.81it/s]
```

```
{'NNLM_alpha': 11.20437723120408,
 'ngram_alpha': 15.891757135551002,
 'TextRnn_alpha': 13.595520055159893,
 'TextRnnAttention_alpha': 12.92293164156136}
```

```
sentence = 'w h a t e'
test_model = './TextRnnAttention_alpha/weights-{}.ckpt'.format(epochs)
next_alpha_predict(test_model,sentence)
rank_alpha_predict(test_model,sentence)
```

```
Enter letters: w h a t e
Word completion:  w h a t e d #
Next letter candidate: ['d', 'r', '#', 's', 'n']
```

```
sentence = 's t u d e n'
test_model = './TextRnnAttention_alpha/weights-{}.ckpt'.format(epochs)
next_alpha_predict(test_model,sentence)
rank_alpha_predict(test_model,sentence)
```

```
Enter letters: s t u d e n
Word completion:  s t u d e n #
Next letter candidate: ['#', 'a', 'e', 's', 't']
```

```
sentence = 'w a c t h'
test_model = './TextRnnAttention_alpha/weights-{}.ckpt'.format(epochs)
next_alpha_predict(test_model,sentence)
rank_alpha_predict(test_model,sentence)
```

```
Enter letters: w a c t h
Word completion:  w a c t h e d #
Next letter candidate: ['e', 'i', 'a', 'n', 'o']
```

- Since 26 characters are compared to words, the amount of calculation will be much less.
- Considering the CPU of the device, we only selected a small part of the data for training. Characters have higher accuracy than words.

Results of 4 models in words

```
[train epoch 0] loss: 8.759 ppl: 6365.140 : 100%|| 38039/38039 [1:39:06<00:00,  6.40it/s]
[valid epoch 0] loss: 9.120 ppl: 9137.179 : 100%|| 31780/31780 [01:34<00:00, 335.03it/s]
[valid epoch 0] loss: 9.125 ppl: 9182.384 : 100%|| 25560/25560 [01:13<00:00, 346.89it/s]
[train epoch 0] loss: 9.855 ppl: 19053.962 : 100%|| 38039/38039 [35:47<00:00, 17.71it/s]
[valid epoch 0] loss: 11.557 ppl: 104501.943 : 100%|| 31780/31780 [00:44<00:00, 719.28it/s]
[valid epoch 0] loss: 11.672 ppl: 117185.032 : 100%|| 25560/25560 [00:32<00:00, 789.80it/s]
[train epoch 0] loss: 9.493 ppl: 13264.919 : 100%|| 38039/38039 [27:15<00:00, 23.26it/s]
[valid epoch 0] loss: 11.495 ppl: 98206.987 : 100%|| 31780/31780 [00:51<00:00, 620.84it/s]
[valid epoch 0] loss: 11.611 ppl: 110286.279 : 100%|| 25560/25560 [00:40<00:00, 626.76it/s]
[train epoch 0] loss: 9.468 ppl: 12941.417 : 100%|| 38039/38039 [24:20<00:00, 26.05it/s]
[valid epoch 0] loss: 11.537 ppl: 102399.150 : 100%|| 31780/31780 [00:37<00:00, 838.64it/s]
[valid epoch 0] loss: 11.639 ppl: 113426.148 : 100%|| 25560/25560 [00:29<00:00, 870.27it/s]
```

```
#  was contacted by Maxim and was given a photo shoot that appeared in the men's magazine.
sentence = 'was contacted by Maxim and was given a photo shoot that appeared'
test_model = './ngram/weights-{}.ckpt'.format(epochs)
test_next_word(test_model,sentence)
rank_words_predict(test_model,sentence)
```

```
Enter a sentence: was contacted by Maxim and was given a photo shoot that appeared
Predict the word ten times ahead: was contacted by Maxim and was given a photo shoot that appeared the the the the the the t
he the the the
Next word candidate: ['the', 'a', 'in', 'of', 'to']
```

```
sentence = 'Nice to meet you'
test_model = './NNLM/weights-{}.ckpt'.format(epochs)
test_next_word(test_model,sentence)
rank_words_predict(test_model,sentence)
```

```
Enter a sentence: Nice to meet you
Predict the word ten times ahead: Nice to meet you to He was the by saying, on the John and
Next word candidate: ['to', 'with', 'after', 'which', 'the']
```

```
sentence = 'Happy Birthday to'
test_model = './NNLM/weights-{}.ckpt'.format(epochs)
test_next_word(test_model,sentence)
rank_words_predict(test_model,sentence)
```

```
Enter a sentence: Happy Birthday to
Predict the word ten times ahead: Happy Birthday to be able to the enough of the latter I and
Next word candidate: ['be', 'Creek', 'work', 'Notes', 'she']
```

- This can be seen from the perplexity. The lower the perplexity index, the more smooth and continuous the sentence.
- The words part is very poor due to the small amount of training data.

NNLM & ngram & TextRNN with Attention - Summary

NNLM & ngram

- Advantage:
  - It contains all the information that the first N-1 words can provide, which have strong constraints on the current word.
- Disadvantage:
  - It requires a considerable amount of training text to determine the parameters of the model.
  - When N is large, the parameter space of the model is too large.

TextRNN with Attention

- Advantage:
  - Visually give the contribution of each word to the result
- Disadvantage:
  - cannot be directly observed, especially when analyzing badcase

Auto-correction

- Package: Autocorrect, TextBlob, Pycorrector

```
d=TextBlob("Mind your own bussiness and do the neccessary")
d.correct()
```

```
TextBlob("Mind your own business and do the necessary")
```

```
[ ]  ### Spelling correction
     from autocorrect import Speller

     spell = Speller()
     spell("I'm not sleapy and tehre is no place I'm giong to.")
```

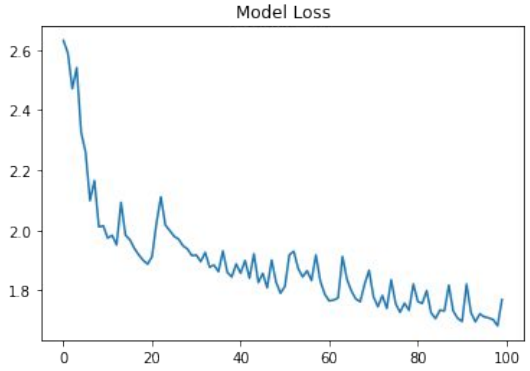
```
"I'm not sleepy and there is no place I'm going to."
```

```
[ ]  import pycorrector

     sent = "what happending? how to speling it, can you gorrect it?"
     corrected_text, details = pycorrector.en_correct(sent)

     print(sent, '=>', corrected_text)
     print(details)
```

```
2022-11-26 21:56:28.715 | DEBUG    | pycorrector.en_spell:_init:39 - load en spell data: C:\jupyter notebook\lib\si
what happending? how to speling it, can you gorrect it? => what happening? how to spelling it, can you correct it?
[('happending', 'happening', 5, 15), ('speling', 'spelling', 24, 31), ('gorrect', 'correct', 44, 51)]
```

- Build a spell checker with an LSTM model
  - generate wrong words - randomly delete/add/replace

```
Model: "model"
_________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #     Connected to
=================================================================================================
 input_1 (InputLayer)        [(None, None, 37)]        0           []

 input_2 (InputLayer)        [(None, None, 39)]        0           []

 lstm (LSTM)                 [(None, 256),             301056      ['input_1[0][0]']
                              (None, 256),
                              (None, 256)]

 lstm_1 (LSTM)               [(None, None, 256),       303104      ['input_2[0][0]',
                              (None, 256),                         'lstm[0][1]',
                              (None, 256)]                         'lstm[0][2]']

 dense (Dense)               (None, None, 39)          10023       ['lstm_1[0][0]']

=================================================================================================
Total params: 614,183
Trainable params: 614,183
Non-trainable params: 0
_________________________________________________________________________________________________
```



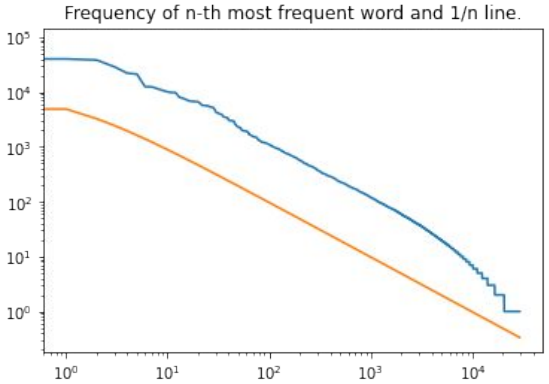
Auto-correction LSTM Result

```
[ ]  for seq_index in range(1):
         input_seq = encoder_input_data[seq_index: seq_index + 1]
         decoded_sentence = decode_sequence(input_seq)
         print('-')
         print('Wrong sentence:', input_texts[seq_index])
         print('Corrected sentence:', decoded_sentence)
         print('Ground Truth:',target_texts[seq_index])
```

```
-
Wrong sentence: contemlpborary
Corrected sentence: contelpariman

Ground Truth:   contemporary
```

Auto-correction Bayesian

- Bayesian with BOW
  - simplified version - avoid orders but keep frequencies
  - **Theorem** from Dr. George Zipf: *in the case of large amounts of data, the probability of word occurrence is approximately the inverse of the word frequency ranking*
- Edit Distance (Levenshtein distance)
  - defined as the use of several insertions, deletion, swap, replace operation changes from one word to another




```
[ ]  splits('speech')
     print (edits0('speech'))
     print (edits1('speech'))
     print (len(edits2('speech')))

     {'speech'}
     {'spench', 'peech', 'speevh', 'spedech', 'speegh', 'spoech', 'sweech', 'speecbh', 'speecb', 'sheech',
     51013
```

Auto-correction Bayesian Result

```
[ ]  correct_text('spech is noot goad')

     'speech is not good'

[ ]  correct_text('havee yoo dane thaat projecct')

     'have you dane that project'

[ ]  correct_text('Profesor is a nyce persan')

     'Professor is a nice person'
```

Conclusions

Auto-complete	Doc2vec & Trie	• Easy to implement • Simply list without ranking • Huge size of model
	Embedding match	• Good results with any length • Slow operation • Require huge memory
	LSTM	• 3 layer LSTM performs best • Efficient results • Prediction depends on input length
	NNLM & N-gram	• It contains all the information that the first N-1 word can provide, and these words have strong constraints on the occurrence of the current word. • It requires a considerable amount of training text to determine the parameters of the model. • When N is large, the parameter space of the model is too large.
	Text RNN with attention	• Not intuitive enough • interpretability is not good, especially when analyzing badcase • intuitively give the contribution of each word to the result

Conclusions

Auto-correct	Mature English text auto-correction package: Autocorrect, TextBlob, Pycorrector	• relatively mature • easy to use
	LSTM-based text auto-correction	• needs to make negative samples • brings high computational overhead • the generalization ability is not strong • very dependent on rich and large corpus
	Auto-correction based on bag of words + Bayes theorem + edit distance	• simple principle of the unsupervised model • small computational overhead • the larger the corpus, the better the effect • stronger generalization ability

Thanks for watching

Reference

- https://aclanthology.org/P16-2034.pdf
- https://www.researchgate.net/publication/320653244_Enhanced_word_embedding_with_multiple_prototypes#pf2
- https://en.wikipedia.org/wiki/Trie