Recurrent Neural Network

Teach the machine writing news headlines

Content

Overview 2

Dataset 2

RNN Construction 2

- Data Loading & Sampling 2
- Tokenization 3
- Modeling 5
- Training 5
- Writing 6

RNN Experiment 7

- Model 1: Base Model 7
- Model 2: A Model With Halved Number Of The Hidden Units 11
- Model 3: A Model With Doubled Number Of The Hidden Units 13
- Model 4: A Model With Halved Length Of The Sequence 15
- Model 5: A Model With Doubled Length Of The Sequence 17
- Model 6: A Model With One More Dense Layer 19
- Code1: Training File 21
- Code1: Writing File 25

Most Funny Title 27

Overall Conclusion 27

- Number of hidden units 27
- Length of sequence 27
- Dense Layer 28

Overview

This report is about the recurrent neural networks(RNN) I designed to learn how to write news headlines based on a dataset on Kaggle.com. To improve the performance of my RNN model, I tried various parameters including the number hidden units in Long Short-term Memory(LSTM) memory layer, number of hidden Dense layers, and sequence length that feeds into the network. At the end, I compared the performances of them and reported the results.

Dataset

The dataset I used is original sourced from the reputable Australian news source ABC (Australian Broadcasting Corp), and is available to be downloaded on Kaggle.com (Kulkarni, 2017). This dataset contains 1,103,663 news headlines published over a period of 15 years from Feb 19, 2003 to Dec 31, 2017.

The dataset has two columns as follows, “date of publishing” in yyyyMMdd, and “headline_text” as string. In the RNN model, I only used the second column to train the data.

publish_date	headline_text
20030219	aba decides against community broadcasting licence
20030219	act fire witnesses must be aware of defamation
20030219	a g calls for infrastructure protection summit
20030219	air nz staff in aust strike for pay rise
20030219	air nz strike to affect australian travellers
20030219	ambitious olsson wins triple jump
20030219	antic delighted with record breaking barca

RNN Construction

This section shows how my RNN model is built based on this particular task, including how I load and preprocess data, build and train model, and let model generate titles based on the trained model.

Data Loading & Sampling

This section shows how I load data from original data file “abcnew-date-text.csv” with “utf8” encoding, clean the data, and do random sampling to get a smaller training data set.

Since the titles are in the second column of the dataset while the first column is useless, I only extract the second column by a ”for” loop with function “readline()”. Next, I got rid of the header line by a simple index slice. The codes are as follows.

```
"""
Load Data Section
"""
#take only the second column of the dataset and get rid of the first header line
filename = 'abcnews-date-text.csv'
file = open(filename, 'r', encoding="utf8")
text_read=[]
for line in file.readlines():
    col1,col2 = line.split(",")
    content = col2.rstrip()
    text_read.append(content)
file.close()

#Randomly select a certain number of records to train the model
sample_size =15000
text_read=text_read[1:len(text_read)]
text_read=random.sample(text_read, sample_size)
```

Because the original dataset is too large to run on my computer, I randomly sampled a 15000 lines as training data set.

The result of this session is a list of titles as follows.

```
In [12]: text_read
Out[12]:
['bombs explode in thailand after travel warnings',
 'tunnel skater fined released from custody',
 'territory overnight visits down',
 'nrn breaking down the door',
 'shares rise on stimulus hope',
 'brazil corruption scandal threatens to overshadow rio olympics',
 'abc weather',
 'new joint strike fighter maintenance facilities open at william',
 'bombers might challenge hocking penalty',
 'military pounds taliban in afghanistan five rebels',
 'nursing home re accredited after sexual assault',
```

Tokenization

This section shows how I tokenized the words in the sampled titles and divided the data into predictors and response.

```
"""
Pretreat Data Section
"""
# integer encode sequences of words
tokenizer = Tokenizer()
tokenizer.fit_on_texts(text_read)
sequences = tokenizer.texts_to_sequences(text_read)
#pad/truncate every data point to be in same length
sequences = pad_sequences(sequences, maxlen=length,padding='pre',truncating='post')
vocab_size = len(tokenizer.word_index) + 1
sequences = array(sequences)
#use the last word in each record as reponse and use the previous words as the predictors
X, Y = sequences[:,:-1], sequences[:,-1]
#change Y into dummy variables
Y = to_categorical(Y, num_classes=vocab_size)
#record the input length of data
seq_length = X.shape[1]
```

First, I used the tokenizer above to encode the sequences of words, transforming the title list into an integer list. Here, each unique word in the title is represented by a unique integer. The result of the step is as follows.

```
[[63, 64, 3, 65, 6, 66, 67],
 [68, 69, 70, 71, 23, 72],
 [73, 74, 75, 10],
 [76, 77, 10, 24, 11],
 [78, 12, 7, 79, 25],
 [80, 26, 81, 82, 1, 83, 84, 85],
 [86, 87],
 [88, 27, 89, 90, 91, 92, 93, 28, 94],
 [95, 96, 97, 98, 99],
 [100, 101, 102, 3, 103, 104, 105],
 [106, 29, 107, 108, 6, 109, 110],
 [111, 112, 6, 30, 113],
```

Since the data point are in different lengths, I used function “pad_sequences” to pad or truncate them in same lengths. Especially, since I want to use the last word as response, I pad “0” in front of the integer list if it is shorten than the length I specified. The result of this step is as follows.

```
In [17]: sequences
Out[17]:
array([[ 63,  64,   3,  65],
       [ 68,  69,  70,  71],
       [ 73,  74,  75,  10],
       [ 76,  77,  10,  24],
       [ 78,  12,   7,  79],
       [ 80,  26,  81,  82],
       [  0,   0,  86,  87],
       [ 88,  27,  89,  90],
       [ 95,  96,  97,  98],
       [100, 101, 102,   3],
       [106,  29, 107, 108],
       [111, 112,   6,  30],
       [  0,  13, 114, 115],
```

Then I divide the data into predictors X and responses Y. Y is the last item in each record while X are the remaining items in each record. In other words, the model is designed to predict the next word based on the first few words in a title. Since this is a multi-categorical prediction task, Y need to be transformed into dummy variables by function “to_categorical”.

Modeling

This section shows my general idea of building the RNN model.

The layers of the model are still in a sequential structure. Therefore, the first step is to create a model by “Sequential()”. Next there should be a embedding layer to further transform the input structure, followed by some Long Short-Term Memory(LSTM) layers to learn the titles, each with a certain number of hidden units. Finally, Dense layers are used to transform the data into a list. Finally, there is an Dense layer producing the output in the same structure as Y. The number of output dimensions should be the same as the vocabulary size because the model is predicting different words as multiple categorical responses. The activation function is “softmax” because this is a multi-categorical problem. Correspondingly, the loss function of compiler is “categorical_crossentropy”, and the metrics is “accuracy”.

```
model = Sequential()
model.add(Embedding(vocab_size, 50, input_length=seq_length))
model.add(LSTM(hidden_units, return_sequences=True))
model.add(LSTM(hidden_units))
model.add(Dense(hidden_units, activation='relu'))
model.add(Dense(vocab_size, activation='softmax'))
#print(model.summary())
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Training

This section shows how I fit the data.

Since training takes a long time, I set five check points and saved five corresponding models as follows. Especially, the “period” parameter of the “ModelCheckpoint” function determines when to save the check points.

```
checkpoint = ModelCheckpoint(filepath, monitor='loss', verbose=1, save_best_only=True, mode='min',period=epoch_num/5)
callbacks_list = [checkpoint]
history=model.fit(X, Y, epochs=epoch_num, batch_size=32, callbacks=callbacks_list,shuffle=True,verbose=1)
```

In addition to the check points, I also saved the entire model and the corresponding tokenizers as follows.

```
# save the model to file
model.save(model_name)
# save the tokenizer
dump(tokenizer, open(token_name, 'wb'))
```

Writing

This section shows how I load the model I saved above and generate news titles by the model.

At first, I loaded the model and tokenizer I trained in the previous sections.

Then I wrote I loop to write news titles. For each loop, a news title is generated. Therefore, the loop will run 15 times to generate 15 titles. The length of each is randomly chosen from 4 words to 8 words.

To generate a certain title, I firstly randomly select a record from X as the seed text and use it to predict what is the next word based on the trained model. Next, I put the predicted item right after the input text to form a news list of words. Each time, I only use the last few items (according to the length of my x) in the news list to predict the next word. Then I repeat the steps above until the news title is generated.

Finally, I just need to print out every title the model generated.

```
"""
Pretreat Data Section
"""
# Reload the tokenizer and preprocess the data
tokenizer = load(open(token_name, 'rb'))
sequences = tokenizer.texts_to_sequences(text_read)
sequences = pad_sequences(sequences, maxlen=length,padding='pre',truncating='post')
sequences = array(sequences)
X = sequences[:,:-1]

# Reload the trained model
model = load_model(model_name)

# for 15 times, randomly choose a seed_text from X, and use it to predict the next
for i in range(15):
    seed_text = X[randint(0,X.shape[0])]
    seed_text_1=list(seed_text.reshape(length-1,1))
    tokenizer.sequences_to_texts(seed_text_1)
    seed_text_2=np.reshape(seed_text,(1,length-1))
    num_words_to_generate = random.randint(4,8)
    result=[]
    for i in range(num_words_to_generate):
        yhat = model.predict_classes(seed_text_2, verbose=0)
        yhat1=list(yhat)
        result.append(tokenizer.sequences_to_texts([yhat1]))
        seed3=list(seed_text_2[0])
        seed3.append(yhat1[0])
        del seed3[0]
        seed_text_2= np.reshape(seed3,(1,length-1))
        text_n=""
    for i in result:
        text_n = text_n+ str(i[0])+" "
    print(text_n)
```

RNN Experiment

Since everything else is covered in previous RNN Construction section, this section only talks about the special settings including epoch number, sequence length, number of hidden units in LSTM layers, sample size and layers.

This section will cover 6 models I build, including a base model, a model with halved number of the hidden units, a model with doubled number of the hidden units, a model with halved length of the sequence, a model with doubled length of the sequence, and a model with one more dense layer.

Model 1: Base Model

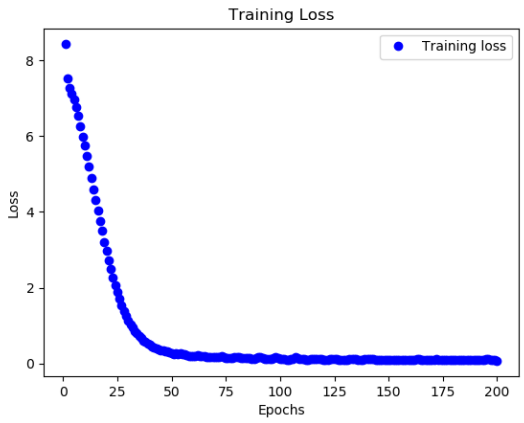
In the base model, the special settings are as follows.

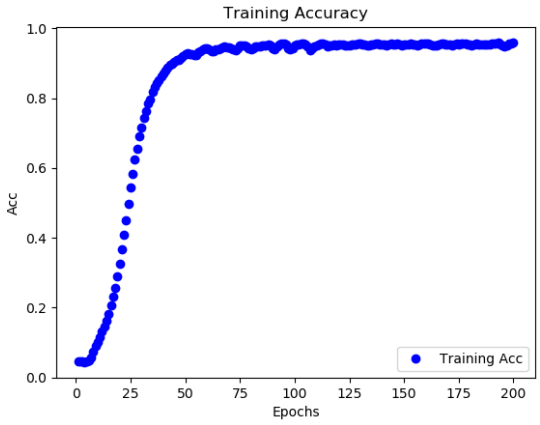
1. Number of epoch: 200
2. Sample size: 15000
3. Sequence length: 4 (that is, the total length including the last word is 4 while the input sequence length is 3)
4. Layers: 1 Embedding layer, 2 LSTM layers, 2 Dense layers
5. Number of hidden units in LSTM layers: 100

Model Summary

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_44 (Embedding)     (None, 3, 50)             785250
_________________________________________________________________
lstm_84 (LSTM)               (None, 3, 100)            60400
_________________________________________________________________
lstm_85 (LSTM)               (None, 100)               80400
_________________________________________________________________
dense_109 (Dense)            (None, 100)               10100
_________________________________________________________________
dense_110 (Dense)            (None, 15705)             1586205
=================================================================
Total params: 2,522,355
Trainable params: 2,522,355
Non-trainable params: 0
```

Plotting





Overall Writing Results:

- winners hit report reveals mislead zones debate
- give 15 first for kalgoorlie
- looking group the migration and september
- more swim federal to monitor
- pay fuel clubs in planning fuel
- term there blamed with cancelling augustas
- ashes interest power ashes charges at way for
- on murray says bill
- fishing promise development go

- as victorian out may plant missing gets launched
- party contain turn frustration
- cash unearthed to cummings many more low
- over qld rural by indictment
- as world's may shape faces to boom to
- cars from richest any to

Breaking point 1: epoch 40

The writing results of weights-improvement-double-unit-40-0.4989-bigger-dropout.hdf5

- threatening with ran stupid account
- crash upgrade killed to keep nicholls action to
- up missing increased by
- charged with pingelly profile appointed bashing
- leader roddick charged with vacant profile
- view paramount melbourne raid closes
- environmental lining cull receives coast reported at a
- urged to nominate info takes top return up
- plans in bushland timeline win unnecessary
- accident eoi water cane favour galilee farm
- to sinks behind highway
- to most urges future
- of weekend leaving strongwoman sales single remember
- security attacks analysis mascarenhas
- oranges endemic other argues claims get

Breaking point 2: epoch 80

The writing results of weights-improvement-double-unit-80-0.1812-bigger-dropout.hdf5

- claims broke settle origin
- national safety to apologise for semi doo narrowly
- police day more concerns death on review of
- charged over break support over heroin calls
- author in appeal for
- unemployment wide view refugee
- found out out safety anti three stabs cricket
- confident of host patrols inquiry qld goulburn
- festival backs murgon glen deal september face
- reply quits for centrelink
- crops fence in lasseter entitlements takeover telstra oakeshott
- for irrigators paint 3 council may
- to monitor tv iron mooted

- hospital against house concert interstate
- simpson murder battle outback for

Breaking point 3: epoch 120

The writing results of weights-improvement-double-unit-120-0.1096-bigger-dropout.hdf5

- media for white lauder
- town backs smashing any officer final re
- ralph industry to banking
- waters sales surge to boost
- native fun new lower new stawell
- refugee erupts khadija underground reasons 17b 300k
- cook top in south philippines bushfire home
- racial jobs from traffic drug sites targeted arrest
- unclear buhrer prefer believed 65 close
- child message continue as pope firing and explosives
- fair to get to extends chief
- rowland pm leaders prices to save more protests
- on one on action
- link accept of conflict that
- malaysia echuca p of telstra down back on

Breaking point 3: epoch 160

The writing results of weights-improvement-double-unit-160-0.0952-bigger-dropout.hdf5

- welcome oranges desal in south australia
- over mental calls for protest
- pressure over jumped babies says irwin opener1 western
- horses their 29 rally to make more
- baghdad village into death roads australia 15 expansion
- groovin militant concerned claims
- fire protection delaying gully
- blame for more involvement charged over detention
- weather pass israel records
- warrnambool road called as jaguar
- australians all calls to slump charged
- closer for to appeal faces
- long gets station as shark echuca may pay
- closure out on nsw the
- riverina native more measures

Breaking point 3: epoch 200

The writing results of weights-improvement-double-unit-200-0.0846-bigger-dropout.hdf5

- pakistan court to chest creates faces president drug
- australia to break upgrade outback sparks
- govt water killed to see goalposts
- get acl with mastermind kills
- land upgrade in schoolies court
- look for ran 10
- driving dragons that darwin broadcasts
- after nt spotlight plains charging rule investigated
- suspect refused bail over armed
- thai australia men to ease double
- win legislation underage work in
- missing man charged with
- lose to success for help
- city copper marsh escalates spirited explosives
- focus at crash could continue to queensland

Comment

The graph shows that the training loss decreases sharply before the epoch number reaches 35 but decreases slowly afterwards. When it reaches 75, the loss decreases very slowly. It shows that the model only takes 75 epochs to learn almost every information. This is the baseline of learning speed for other models.

Interestingly, the writing results of the five checking points are similar. This fact is reasonable because after the model finishes learning 40 epochs, it has already learned most of the information. Therefore, there will not be a huge differences among those checking points.

When we look at its loss, we can see the difference better. The loss for epoch 40 (checking point model) is 0.4989; the loss for epoch 80 (checking point model) is 0.1812; the loss for epoch 120 (checking point model) is 0.1096; the loss for epoch 160 (checking point model) is 0.0952; the loss for epoch 200 (checking point model) is 0.0846. Actually, the loss decreases stably when the epoch number increases. This fact shows that the more the number of epochs, the lower the loss. However, reading writing result is not helpful to determine the results here.

Model 2: A Model With Halved Number Of The Hidden Units

Settings

In the Model 2, the special settings are as follows.

1. Number of epoch: 200
2. Sample size: 15000
3. Sequence length: 4 (that is, the total length including the last word is 4 while the input sequence length is 3)
4. Layers: 1 Embedding layer, 2 LSTM layers, 2 Dense layers
5. Number of hidden units in LSTM layers: 50 (the only difference from base model)

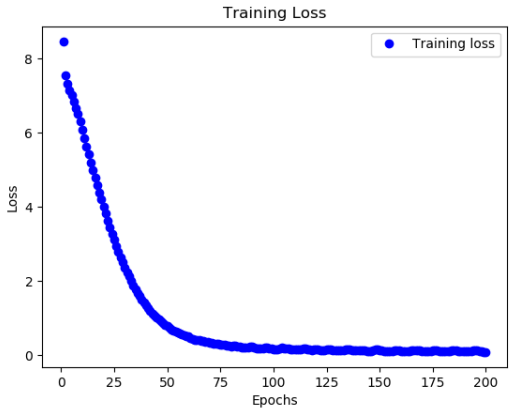
Model Summary

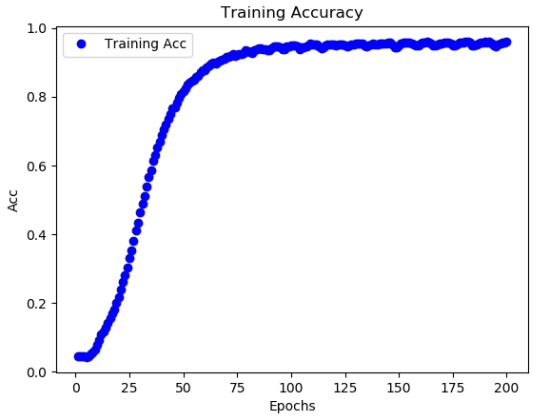
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_45 (Embedding)     (None, 3, 50)             784800
_________________________________________________________________
lstm_86 (LSTM)               (None, 3, 50)             20200
_________________________________________________________________
lstm_87 (LSTM)               (None, 50)                20200
_________________________________________________________________
dense_111 (Dense)            (None, 50)                2550
_________________________________________________________________
dense_112 (Dense)            (None, 15696)             800496
=================================================================
Total params: 1,628,246
Trainable params: 1,628,246
Non-trainable params: 0
```

Writing Result

- stockland surgeons housing has has market
- blow keep bickering mill india on anti brisbane
- news with exposed closely worries officers birchall
- miss new istanbul keen to help help solve
- court as pyne housing that investigation surgery
- to end close in north mcgrath
- high debt cue behind after dam program
- but nrls simple problems markets aide
- drug with naval nsw sam
- water education changing habib coal trial combat war
- prosecuted border divided plant
- coves sams approach to
- plans to release murder to take police
- with little government down centres tax negotiations
- jancourt trail on hit at sydney turned

Plotting





Comment

The graph shows that the training loss decreases sharply before the epoch number reaches 60 but decreases slowly afterwards. When it reaches 100, the loss decreases very slowly. It shows that the model only takes 100 epochs to learn almost every information. These numbers are clearly larger than the base model, showing that this model learns more slowly than the base model.

The writing results are similar with the first model. When we look at its loss, we can see the difference better. The loss for epoch 40 (checking point model) is 1.3431, which is greatly higher than base model, 0.4989; the loss for epoch 80 (checking point model) is 0.2461, also higher than base model, 0.1812; the loss for epoch 120 (checking point model) is 0.1517, higher than base model, 0.1096; the loss for epoch 160 (checking point model) is 0.1195, also higher than base model, 0.0952; the loss for epoch 200 (checking point model) is 0.0945, which is slightly higher than based model 0.0846.

This shows that for each epoch, the training result of this model is worse than base model. However, at the end, the resulting loss is very close the base model, showing that the model just learn more slowly than base model.

The reason is that this model does not have enough hidden units for LSTM training. In conclusion, the performance of 100 hidden units in LSTM is obviously better than 50 hidden units.

Model 3: A Model With Doubled Number Of The Hidden Units

Settings

In the Model 3, the special settings are as follows.

1. Number of epoch: 200
2. Sample size: 15000
3. Sequence length: 4 (that is, the total length including the last word is 4 while the input sequence length is 3)
4. Layers: 1 Embedding layer, 2 LSTM layers, 2 Dense layers
5. Number of hidden units in LSTM layers: 200 (the only difference from base model)

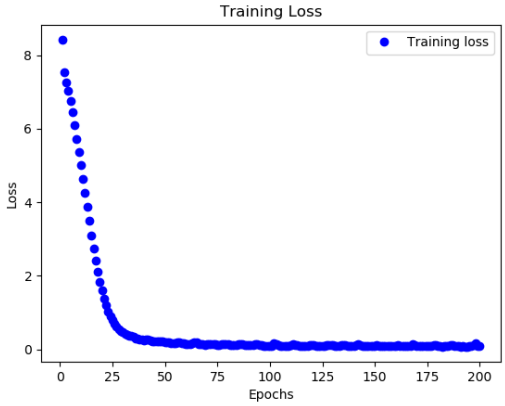
Model Summary

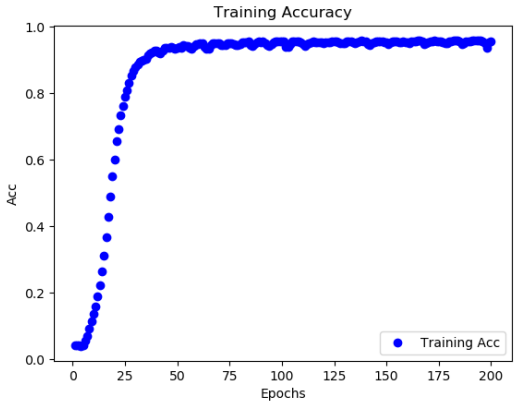
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_46 (Embedding)     (None, 3, 50)             785000
_________________________________________________________________
lstm_88 (LSTM)               (None, 3, 200)            200800
_________________________________________________________________
lstm_89 (LSTM)               (None, 200)               320800
_________________________________________________________________
dense_113 (Dense)            (None, 200)               40200
_________________________________________________________________
dense_114 (Dense)            (None, 15700)             3155700
=================================================================
Total params: 4,502,500
Trainable params: 4,502,500
Non-trainable params: 0
```

Writing Result

- suffered nears scully members on silence
- day day merger drops change profile after
- for tourism competition as
- of ex hostage sydney be roll stars
- dope mount government marked funding robbed
- warriors as pregnant demand
- inquiry into market farm
- leaks low two rebuild
- crash ambulance change cuts leaves allow
- deadline deadline in michael bias
- australia study make dream hockeyroos for govt for
- case attack after warns about government survivor
- homes conservationist proteas african confidence in face court
- weekend helps protesters bathurst flat in
- david turnbull orange gets finds cloud must port

Plotting





Comment

The graph shows that the training loss decreases sharply before the epoch number reaches 40 but decreases slowly afterwards. When it reaches 60, the loss decreases very slowly. It shows that the model only takes 60 epochs to learn almost information. These numbers are close to base model, showing that this model learns in a similar speed as the base model.

The writing results are similar with the first model. When we look at its loss, we can see the difference better. The loss for epoch 40 (checking point model) is 0.2570, which is greatly lower than base model, 0.4989; the loss for epoch 80 (checking point model) is 0.1351, also lower than base model, 0.1812; the loss for epoch 120 (checking point model) is 0.1138, interestingly higher than base model, 0.1096; the loss for epoch 160 (checking point model) is 0.0991, also higher than base model, 0.0952; the loss for epoch 200 (checking point model) is 0.0836, which is slightly lower than based model 0.0846.

Interestingly, in the first 80 epoch, the performance of this model is apparently better than the base model. However, after reaching 120 epochs, they are very close. At the end, the resulting loss is very close the base model. This fact shows that this model learns faster than base model in the first 100 epochs. However, after that, the performance is very close to the base model.

Possibly, the reason is that more hidden units for LSTM training is helpful to learn information in a shorter time. However, it does not improve the final result.

Model 4: A Model With Halved Length Of The Sequence

Settings

In the Model 3, the special settings are as follows.

1. Number of epoch: 200
2. Sample size: 15000
3. Sequence length: 2 (the only change from base model)
   (that is, the total length including the last word is 2 while the input sequence length is 1)
4. Layers: 1 Embedding layer, 2 LSTM layers, 2 Dense layers
5. Number of hidden units in LSTM layers: 100

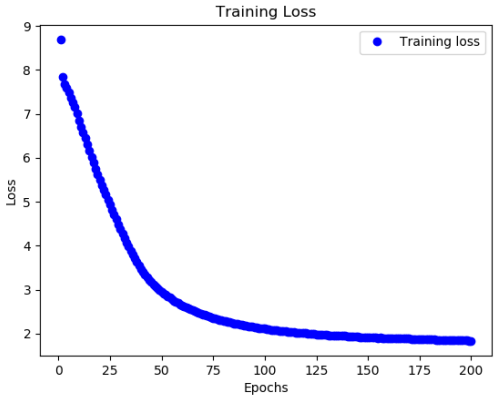
Model Summary

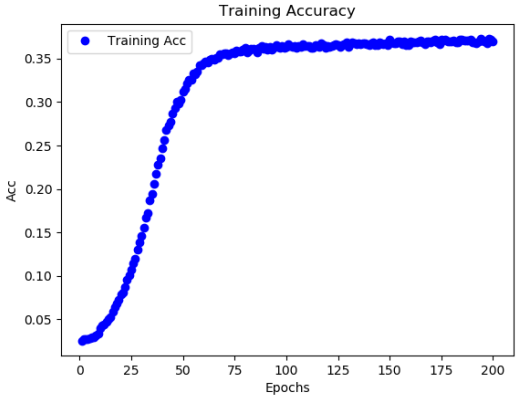
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_47 (Embedding)     (None, 1, 50)             768250
_________________________________________________________________
lstm_90 (LSTM)               (None, 1, 100)            60400
_________________________________________________________________
lstm_91 (LSTM)               (None, 100)               80400
_________________________________________________________________
dense_115 (Dense)            (None, 100)               10100
_________________________________________________________________
dense_116 (Dense)            (None, 15365)             1551865
=================================================================
Total params: 2,471,015
Trainable params: 2,471,015
Non-trainable params: 0
```

Writing Results

- country hour concerns raised beds concerns raised beds
- president trump holds concerns raised beds
- minister to deal owners horrified minister to deal
- coast conference puts minister to deal owners horrified
- down concerns raised beds concerns raised beds
- nsw country hour concerns raised
- find owners horrified minister
- direct deal owners horrified
- country hour concerns raised beds concerns raised beds
- says minister to deal owners horrified
- students to deal owners horrified minister
- to deal owners horrified minister to deal owners
- media call for sale of deal owners
- owners horrified minister to deal owners horrified minister
- hunter water plan for sale

Plotting





Comment

The graph shows that the training loss decreases sharply before the epoch number reaches 55, but decreases slowly afterwards. When it reaches 175, the loss decreases very slowly. It shows that the model only takes 175 epochs to learn almost every information. These numbers are clearly larger than the base model, showing that this model learns much more slowly than the base model.

For the first time, we can see an obvious difference in writing results from the base model. This model frequently generated repeated pattern such as “concerns raised beds concerns raised beds”. This shows that generating the next word only by the previous word is not feasible.

Then we can take a look at the loss. The loss for epoch 40 (checking point model) is 3.4737, which is lower than base model, 0.4989; the loss for epoch 80 (checking point model) is 2.2959, becoming greater than base model, 0.1812; the loss for epoch 120 (checking point model) is 2.0074, higher than base model, 0.1096; the loss for epoch 160 (checking point model) is 1.8973, also higher than base model, 0.0952; the loss for epoch 200 (checking point model) is 1.8404, which is much higher than based model 0.0846.

Except the starting point, the model learn much worse than the base model. Again, it shows that generating the next word only by the previous word is not feasible.

Model 5: A Model With Doubled Length Of The Sequence

Settings

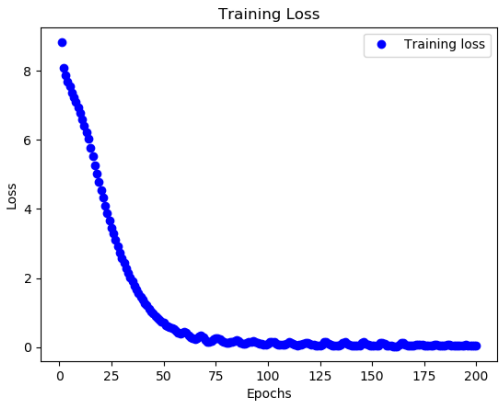
In the Model 5, the special settings are as follows.

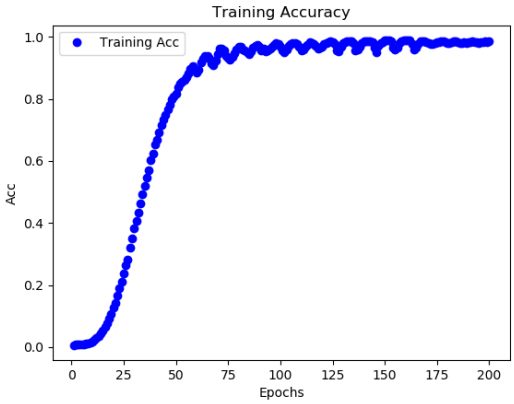
1. Number of epoch: 200
2. Sample size: 15000
3. Sequence length: 8 (the only change from base model)
   (that is, the total length including the last word is 8 while the input sequence length is 7)
4. Layers: 1 Embedding layer, 2 LSTM layers, 2 Dense layers
5. Number of hidden units in LSTM layers: 100

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_48 (Embedding)     (None, 7, 50)             781000
_________________________________________________________________
lstm_92 (LSTM)               (None, 7, 100)            60400
_________________________________________________________________
lstm_93 (LSTM)               (None, 100)               80400
_________________________________________________________________
dense_117 (Dense)            (None, 100)               10100
_________________________________________________________________
dense_118 (Dense)            (None, 15620)             1577620
=================================================================
Total params: 2,509,520
Trainable params: 2,509,520
Non-trainable params: 0
```

- suspicious falls stagnation dentists allowance fall time off
- concerns russia days shows pornography task laden
- plans funds decision funds
- pic hodgson electricity water down tomorrow system police
- japan sa sa hearing workers
- donations marriage region allocations
- car in wa schools shooting claims official
- native slug bullying spy quit redevelopment
- allowance house off on coast claims
- 9 station disease push course push time
- announced 1911 virus projects
- tigers zealand dayers more
- works freeze rural breaches pledge head
- wet twice crisis melbourne sa bank robbery
- ripper views use scottsdale use

Plotting





Comment

The graph shows that the training loss decreases sharply before the epoch number reaches 55, but decreases slowly afterwards. When it reaches 175, the loss decreases very slowly. It shows that the model only takes 175 epochs to learn almost information. These numbers are clearly larger than the base model, showing that this model learns much more slowly than the base model. This is due to the doubled input information.

The writing results are similar with the first model, maybe a little bit more readable than base model. When we look at its loss, we can see the difference better. The loss for epoch 40 (checking point model) is 1.3716, which is greatly lower than base model, 0.4989; the loss for epoch 80 (checking point model) is 0.13711, also lower than base model, 0.1812; the loss for epoch 120 (checking point model) is 0.1024, interestingly higher than base model, 0.1096; the loss for epoch 160 (checking point model) is 0.0326, lower than base model, 0.0952; the loss for epoch 200 (checking point model) is 0.0301, which is much lower than based model 0.0846.

Except the starting point, the model learns much better than the base model, showing that using all previous words to predict next word is much more accurate.

Model 6: A Model With One More Dense Layer.

Settings

In the Model 3, the special settings are as follows.

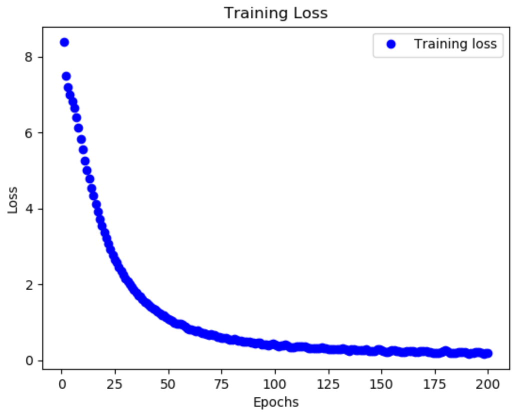
1. Number of epoch: 200
2. Sample size: 15000
3. Sequence length: 4 (that is, the total length including the last word is 4 while the input sequence length is 3)
4. Layers: 1 Embedding layer, 2 LSTM layers, 3 Dense layers(the only change from base model)
5. Number of hidden units in LSTM layers: 100

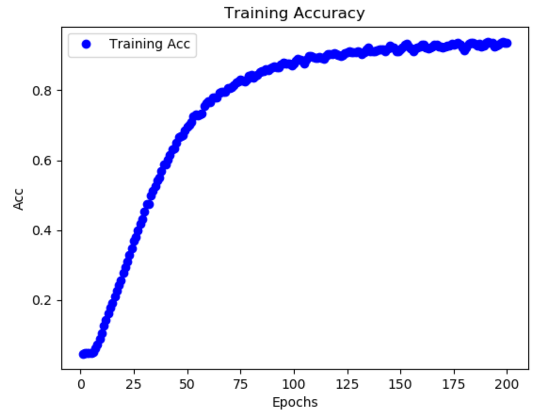
Model Summary

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_49 (Embedding)     (None, 3, 50)             785950
_________________________________________________________________
lstm_94 (LSTM)               (None, 3, 100)            60400
_________________________________________________________________
lstm_95 (LSTM)               (None, 100)               80400
_________________________________________________________________
dense_119 (Dense)            (None, 100)               10100
_________________________________________________________________
dense_120 (Dense)            (None, 100)               10100
_________________________________________________________________
dense_121 (Dense)            (None, 15719)             1587619
=================================================================
Total params: 2,534,569
Trainable params: 2,534,569
Non-trainable params: 0
```

Writing Results

- howard new school part over hit financial
- days turned people out in record security
- end over charlestown three not
- in class sons title another snapshot snapshot
- abuse property killed dead 24
- in being tunnel all losing spies south
- slay cause finding bull conversion caught world
- honduras gympie experts ceasefire increase
- out to fast western 21
- celebrations revenue lead growers single families
- cannabis shooting fight over lack
- sold at outback die candidates dutch for cut
- hayden kills job volunteer to
- closure camp as european education istanbul a university
- bail and gambling test at 20 speed escape





Comment

The graph shows that the training loss decreases sharply before the epoch number reaches 40, but decreases slowly afterwards. When it reaches 150, the loss decreases very slowly. It shows that the model only takes 150 epochs to learn almost every information. These numbers are slightly larger than the base model, showing that this model learns more slowly than the base model. This is due to the more complex model with an additional Dense layer.

The writing results are similar with the first model. When we look at its loss, we can see the difference better. The loss for epoch 40 (checking point model) is 1.5109, which is greatly larger than base model, 0.4989; the loss for epoch 80 (checking point model) is 0.5434, also larger than base model, 0.1812; the loss for epoch 120 (checking point model) is 0.3146, higher than base model, 0.1096; the loss for epoch 160 (checking point model) is 0.2087, higher than base model, 0.0952; the loss for epoch 200 (checking point model) is 0.1853, which is much lower than based model 0.0846.

For each epoch number, the model learns much worse than the base model with lower speed. It shows that adding another dense layer is not a good choice.

Code1: Training File

I wrote a loop to train each model mentioned above.

```
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Import Libraries Section
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

from numpy import array
from pickle import dump
from keras.preprocessing.text import Tokenizer
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Embedding
from random import randint
```

```
from pickle import load
from keras.models import load_model
from keras.preprocessing.sequence import pad_sequences
import random
from keras.callbacks import ModelCheckpoint
import matplotlib.pyplot as plt

"""
Parameters Section
"""
epoch_num = 200
sample_size = 15000

for i in [0, 1, 2, 3, 4, 5]:
    if i == 0:
        hidden_units = 100
        length = 4
        model_name = 'base.h5'
        token_name = 'base.pkl'
        filepath = "0/weights-improvement-double-unit-{epoch:02d}-{loss:.4f}-bigger-
dropout.hdf5"

    if i == 1:
        hidden_units = 50
        length = 4
        model_name = 'hidden_half.h5'
        token_name = 'hidden_half.pkl'
        filepath = "1/weights-improvement-double-unit-{epoch:02d}-{loss:.4f}-bigger-
dropout.hdf5"

    if i == 2:
        hidden_units = 200
        length = 4
        model_name = 'hidden_double.h5'
        token_name = 'hidden_double.pkl'
        filepath = "2/weights-improvement-double-unit-{epoch:02d}-{loss:.4f}-bigger-
dropout.hdf5"

    if i == 3:
        hidden_units = 100
        length = 2
        model_name = 'length_half.h5'
        token_name = 'length_half.pkl'
        filepath = "3/weights-improvement-double-unit-{epoch:02d}-{loss:.4f}-bigger-
dropout.hdf5"
```

```
    if i == 4:
        hidden_units = 100
        length = 8
        model_name = 'length_double.h5'
        token_name = 'length_double.pkl'
        filepath = "4/weights-improvement-double-unit-{epoch:02d}-{loss:.4f}-bigger-
dropout.hdf5"

    if i == 5:
        hidden_units = 100
        length = 4
        model_name = 'dense.h5'
        token_name = 'dense.pkl'
        filepath = "5/weights-improvement-double-unit-{epoch:02d}-{loss:.4f}-bigger-
dropout.hdf5"

    filename = 'abcnews-date-text.csv'
    file = open(filename, 'r', encoding="utf8")
    text_read = []
    for line in file.readlines():
        col1, col2 = line.split(",")
        content = col2.rstrip()
        text_read.append(content)
    file.close()

    text_read = text_read[1:len(text_read)]
    text_read = random.sample(text_read, sample_size)

    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(text_read)
    sequences = tokenizer.texts_to_sequences(text_read)
    sequences = pad_sequences(sequences, maxlen=length, padding='pre',
truncating='post')
    vocab_size = len(tokenizer.word_index) + 1
    sequences = array(sequences)
    X, Y = sequences[:, :-1], sequences[:, -1]
    Y = to_categorical(Y, num_classes=vocab_size)
    seq_length = X.shape[1]

    if i != 5:
        model = Sequential()
        model.add(Embedding(vocab_size, 50, input_length=seq_length))
        model.add(LSTM(hidden_units, return_sequences=True))
        model.add(LSTM(hidden_units))
```

```
        model.add(Dense(hidden_units, activation='relu'))
        model.add(Dense(vocab_size, activation='softmax'))
        # print(model.summary())
        model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

    else:
        model = Sequential()
        model.add(Embedding(vocab_size, 50, input_length=seq_length))
        model.add(LSTM(hidden_units, return_sequences=True))
        model.add(LSTM(hidden_units))
        model.add(Dense(hidden_units, activation='relu'))
        model.add(Dense(hidden_units, activation='relu'))
        model.add(Dense(vocab_size, activation='softmax'))
        # print(model.summary())
        model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

    checkpoint = ModelCheckpoint(filepath, monitor='loss', verbose=1,
save_best_only=True, mode='min', period=40)
    callbacks_list = [checkpoint]
    if i == 0:
        history0 = model.fit(X, Y, epochs=epoch_num, batch_size=32,
callbacks=callbacks_list, shuffle=True, verbose=1)
    if i == 1:
        history1 = model.fit(X, Y, epochs=epoch_num, batch_size=32,
callbacks=callbacks_list, shuffle=True, verbose=1)
    if i == 2:
        history2 = model.fit(X, Y, epochs=epoch_num, batch_size=32,
callbacks=callbacks_list, shuffle=True, verbose=1)
    if i == 3:
        history3 = model.fit(X, Y, epochs=epoch_num, batch_size=32,
callbacks=callbacks_list, shuffle=True, verbose=1)
    if i == 4:
        history4 = model.fit(X, Y, epochs=epoch_num, batch_size=32,
callbacks=callbacks_list, shuffle=True, verbose=1)
    if i == 5:
        history5 = model.fit(X, Y, epochs=epoch_num, batch_size=32,
callbacks=callbacks_list, shuffle=True, verbose=1)

    # save the model to file
    model.save(model_name)
    # save the tokenizer
    dump(tokenizer, open(token_name, 'wb'))
```

```
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Show output Section
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
plt.figure()
loss = history5.history['loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.title('Training Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# plot the training accuracy and testing accuracy against number of epochs
plt.figure()
acc = history5.history['acc']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training Acc')
plt.title('Training Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Acc')
plt.legend()
plt.show()
```

Code1: Writing File

```
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Import Libraries Section
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
from numpy import array
import numpy as np
from pickle import dump
from keras.preprocessing.text import Tokenizer
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Embedding
from random import randint
from pickle import load
from keras.models import load_model
from keras.preprocessing.sequence import pad_sequences
import random
from keras.callbacks import ModelCheckpoint
import matplotlib.pyplot as plt
```

```
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Parameters Section
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
length = 4 #the only thing to change
model_name='5/dense.h5'
token_name='5/dense.pkl'
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Load Data Section
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
#take only the second column of the dataset and get rid of the first header line
filename = 'abcnews-date-text.csv'
file = open(filename, 'r', encoding="utf8")
text_read=[]
for line in file.readlines():
    col1,col2 = line.split(",")
    content = col2.rstrip()
    text_read.append(content)
file.close()

#Randomly select a certain number of records to train the model
text_read=text_read[1:len(text_read)]

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Pretreat Data Section
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
# Reload the tokenizer and preprocess the data
tokenizer = load(open(token_name, 'rb'))
sequences = tokenizer.texts_to_sequences(text_read)
sequences = pad_sequences(sequences, maxlen=length,padding='pre',truncating='post')
sequences = array(sequences)
X = sequences[:,:-1]

# Reload the trained model
model = load_model(model_name)

# for 15 times, randomly choose a seed_text from X, and use it to predict the next
for i in range(15):
    seed_text = X[randint(0,X.shape[0])]
    seed_text_1=list(seed_text.reshape(length-1,1))
    tokenizer.sequences_to_texts(seed_text_1)
    seed_text_2=np.reshape(seed_text,(1,length-1))
    num_words_to_generate = random.randint(4,8)
    result=[]
    for i in range(num_words_to_generate):
```

```
        yhat = model.predict_classes(seed_text_2, verbose=0)
        yhat1=list(yhat)
        result.append(tokenizer.sequences_to_texts([yhat1]))
        seed3=list(seed_text_2[0])
        seed3.append(yhat1[0])
        del seed3[0]
        seed_text_2= np.reshape(seed3,(1,length-1))
        text_n=""
    for i in result:
        text_n = text_n+ str(i[0])+" "
    print(text_n)
```

Most Funny Title

"Car in WA Schools Shooting Claims Official", which is from Model 5.

Overall Conclusion

Number of hidden units

The model with halved number of hidden units learns more slowly than the base model and the performance is always worse than the base model when the epoch number increases. The ending result is very close to but slightly worse than the base model. In addition, we can see no clear differences between their writing results. The reason is that this model does not have enough hidden units for LSTM training. In conclusion, the performance of 100 hidden units in LSTM is obviously better than 50 hidden units.

The model with doubled number of hidden units learns in a similar speed as the base model and the performance is very close to the base model (just slightly lower than it). In addition, we can see no clear differences between their writing results.

In conclusion, the number of hidden units in LSTM layer does not affect the result a lot. However, 200-unit-model is slightly better than 100-unit model, and 100-unit model is slightly better than 50-unit mode. Maybe we can conclude that more hidden units for LSTM training is helpful to learn more information in a shorter time.

Length of sequence

We can see an obvious difference in writing results that the model with halved length of sequence is much worse than the base model, and the model with doubled length of sequence is slightly better than the base model.

Moreover, the performance of the model with halved length of sequence is much worse than the base model while the performance of the model with doubled length of sequence is much better than the base model.

In conclusion, the length of sequence is an important factor that influences the training performance. Using more previous words to predict next word is more accurate. However, it takes more time to run.

Dense Layer

The model with another dense layer learns more slowly than the base model and performs much worse than the base model. Therefore, in this case, no additional dense layer is needed.