Keras Tutorial - Spoken Language Understanding

05 January 2017

In previous tutorial, we have had a introduction to convolutional neural networks(CNNs) and keras deep learning framework. We have used them to solve a computer vision(CV) problem: traffic sign recognition. Today, we will solve a natural language processing (NLP) problem with keras.

Problem and the Dataset

Problem we are going to tackle is Natural Language Understanding. It aims to extract meanining of speech utterances. However, this is still an unsolved problem. Therefore, we break this problem into a solvable practical problem of understanding the speaker in a limited context. In particular, we want to identify the intent of a speaker asking for information about flights.

Dataset we are going to use is Airline Travel Information System (ATIS). This dataset was collected by DARPA in the early 90s. ATIS consists of spoken queries on flight related information. An example utterance is *I want to go from Boston to Atlanta on Monday*. Understanding this is then reduced to identifying arguments like *Destination* and

Departure Day. This task is called slot-filling.

Here is an example sentence and its labels [1] from the dataset:

[1] You will observe that labels are encoded in Inside Outside Beginning (IOB) representation.

Words	Show	flights	from	Boston	to	New	York	today
Labels	O	O	O	B-dept	O	B-arr	I-arr	B-date

The ATIS official split contains 4,978/893 sentences for a total of 56,590/9,198 words (average sentence length is 15) in the train/test set. The number of classes (different slots) is 128 including the O label (NULL). Unseen words in the test set are encoded by `<UNK>` token and each digit is replaced with string `DIGIT`, i.e `20` is converted to `DIGITDIGIT`.

Our approach to the problem is to use

- Word embeddings
- Recurrent Neural Networks

I'll talk about these briefly in the following sections.

Word Embeddings

Word embeddings maps words to a vector in a high-dimensional space. If learnt the right way, these word embeddings can learn semantic and syntactic information of the words i.e, similar words are close to each other in this space and dissimilar words far apart.

These can be learnt either using large amount of text like Wikipedia or specifically for a given problem. We will take the second approach for this problem.

As an illustation, I have shown here the nearest neighbors in the word embedding space for some of the words . This embedding space was learnt by the model we define later in the post.

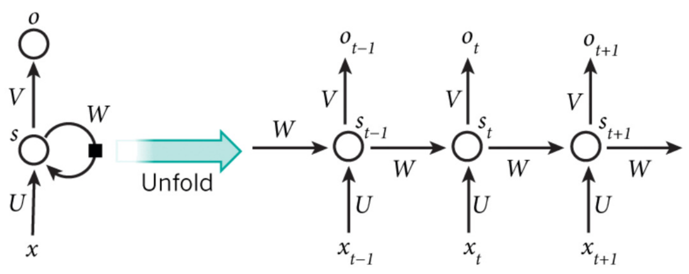
sunday	delta	california	boston	august	time	car
wednesday	continental	colorado	nashville	september	schedule	rental
saturday	united	florida	toronto	july	times	limousine
friday	american	ohio	chicago	june	schedules	rentals
monday	eastern	georgia	phoenix	december	dinnertime	cars
tuesday	northwest	pennsylvania	cleveland	november	ord	taxi
thursday	us	north	atlanta	april	f28	train
wednesdays	nationair	tennessee	milwaukee	october	limo	limo
saturdays	lufthansa	minnesota	columbus	january	departure	ap
sundays	midwest	michigan	minneapolis	may	sfo	later

Recurrent Neural Networks

Convolutional layers can be a great way to pool local information, but they do not really capture the sequentiality of the data. Recurrent Neural Networks (RNNs) help us tackle sequential information like natural language.

If we are going to predict properties of the current word, we better remember the words before it too. An RNN has such an internal state/memory which stores the summary of the sequence it has seen so far. This allows us to use RNNs to solve complicated word tagging problems like part of speech (POS) tagging or slot filling as in our case.

Following diagram illustrates the internals of RNN:



Source: Nature

Let's briefly go through the diagram:

- $x_1, x_2, \ldots, x_{t-1}, x_t, x_{t+1} \ldots$ is input to the RNN.
- s_t is the hidden state of the RNN at the step t . This is computed based on the state at the step $t-1$ as $s_t = f(Ux_t + Ws_{t-1})$. Here f is a nonlinearity like tanh or ReLU.
- o_t is the output at the step t. Computed as $o_t = f(Vs_t)$
- U, V, W are the learnable parameters of RNN.

For our problem, we will pass word embeddings sequence as the input to the RNN.

Putting it all together

Now that we've setup the problem and have a understanding of the basic blocks, let's code it up.

Since we are using IOB representation for labels, it's not trivial to calculate the scores of our model. We therefore use the conlleval perl script to compute the F1 Scores. I've adapted the code from here for the data preprocessing and score calculation. Complete code is available at GitHub

```
$ git clone https://github.com/chsasank/
$ cd ATIS.keras
```

I recommend using jupyter notebook to run and experiment with the snippets from the tutorial.

```
$ jupyter notebook
```

Loading Data

Let's load the data using `data.load.atisfull()`. It will download the data first time it is run. Words and labels are encoded as indexes to a vocabulary. This vocabulary is stored in `w2idx` and `labels2idx`.

```
import numpy as np
import data.load

train_set, valid_set, dicts = data.load.
w2idx, labels2idx = dicts['words2idx'], 

train_x, _, train_label = train_set
val_x, _, val_label = valid_set

# Create index to word/label dicts
idx2w  = {w2idx[k]:k for k in w2idx}
```

```
idx2la = {labels2idx[k]:k for k in label

# For conlleval script
words_train = [ list(map(lambda x: idx2w
labels_train = [ list(map(lambda x: idx2
words_val = [ list(map(lambda x: idx2w[x
labels_val = [ list(map(lambda x: idx2la

n_classes = len(idx2la)
n_vocab = len(idx2w)
```

Let's print an example sentence and label.

```
print("Example sentence : {}".format(wor
print("Encoded form: {}".format(train_x[
print()
print("It's label : {}".format(labels_tr
print("Encoded form: {}".format(train_la
```

Output:

```
Example sentence : ['i', 'want', 'to', '
Encoded form: [232 542 502 196 208  77

It's label : ['O', 'O', 'O', 'O', 'O', '
Encoded form: [126 126 126 126 126  48 1
```

Keras model

Next we define the keras model. Keras has inbuilt `Embedding` layer for word embeddings. It expects integer indices. `SimpleRNN` is the recurrent neural network layer described above. We will have to use `TimeDistributed` to pass the output of RNN o_t at each time step t to a fully connected layer. Otherwise, output at the final time step will be passed on to the next layer.

```
from keras.models import Sequential
from keras.layers.embeddings import Embe
from keras.layers.recurrent import Simpl
from keras.layers.core import Dense, Dro
from keras.layers.wrappers import TimeDi
from keras.layers import Convolution1D

model = Sequential()
model.add(Embedding(n_vocab,100))
model.add(Dropout(0.25))
model.add(SimpleRNN(100,return_sequences
model.add(TimeDistributed(Dense(n_classe
model.compile('rmsprop', 'categorical_cr
```

Training

Now, let's start training our model. We will pass each sentence as a batch to the model. We cannot use `model.fit()` as it expects all the sentences to be of same size. We will therefore use `model.train_on_batch()`.

Training is very fast as the dataset is relatively small. Each epoch takes 20 seconds on my Macbook Air.

```
import progressbar
n_epochs = 30

for i in range(n_epochs):
    print("Training epoch {}".format(i))

    bar = progressbar.ProgressBar(max_va
    for n_batch, sent in bar(enumerate(t
        label = train_label[n_batch]
        # Make labels one hot
        label = np.eye(n_classes)[label]
        # View each sentence as a batch
        sent = sent[np.newaxis,:]

        if sent.shape[1] > 1: #ignore 1
            model.train_on_batch(sent, l
```

Evaluation

To measure the accuracy of the model, we use `model.predict_on_batch()` and `metrics.accuracy.conlleval()`.

```
from metrics.accuracy import conlleval

labels_pred_val = []

bar = progressbar.ProgressBar(max_value=
for n_batch, sent in bar(enumerate(val_x
    label = val_label[n_batch]
    label = np.eye(n_classes)[label][np.
    sent = sent[np.newaxis,:]

    pred = model.predict_on_batch(sent)
    pred = np.argmax(pred,-1)[0]
    labels_pred_val.append(pred)

labels_pred_val = [ list(map(lambda x: i
                                for 
con_dict = conlleval(labels_pred_val, la
                            words_val, '

print('Precision = {}, Recall = {}, F1 =
            con_dict['r'], con_dict['p']
```

With this model, I get **92.36** F1 Score.

```
Precision = 92.07, Recall = 92.66, F1 = 
```

Note that for the sake of brevity, I've not showed logging part of the code. Loggging losses and accuracies is an important part of coding up an model. An improved model (described in the next section) with logging is at `main.py`. You can run it as :

```
$ python main.py
```

Improvements

One drawback with our current model is that there is no lookahead. i.e, output o_t depends only on the current and previous words but not on the words next to it. One can imagine that clues about the properties of the current word is also held by next word.

Lookahead can easily be implemented by having a convolutional layer before RNN and after word embeddings:

```
model = Sequential()
model.add(Embedding(n_vocab,100))
model.add(Convolution1D(128, 5, border_m
model.add(Dropout(0.25))
model.add(GRU(100,return_sequences=True)
model.add(TimeDistributed(Dense(n_classe
model.compile('rmsprop', 'categorical_cr
```

With this improved model, I get **94.90** F1 Score.

Conclusion

In this tutorial, we have learnt about word embeddings and RNNs. We have applied these to a NLP problem: ATIS. We also have made an improvement to our model.

To improve the model further, we could try using word embeddings learnt on a large corpus like Wikipedia. Also, there are variants of RNNs like LSTM or GRU which can be experimented with.

References

1. Grégoire Mesnil, Xiaodong He, Li Deng and Yoshua Bengio. Investigation of Recurrent-Neural-Network Architectures and Learning Methods for Spoken Language Understanding. Interspeech, 2013. pdf

2. Recurrent Neural Networks with Word Embeddings, theano tutorial