PharmaSUG 2021 : Paper HT-210

Hands-on Training for Machine Learning Programming - Natural Language Processing

Kevin Lee, Genpact

ABSTRACT

One of the most popular Machine Learning implementation is Natural Language Processing (NLP). NLP is a Machine Learning application or service which are able to understand human language. Some practical implementations are speech recognition, machine translation and chatbot. Siri, Alexa and Google Home are popular applications whose technologies are based on NLP.

Hands-on Training of NLP Machine Learning Programming is intended for statistical programmers and biostatisticians who want to learn how to conduct simple NLP Machine Learning projects. Hands-on NLP training will use the most popular Machine Learning program - Python. The training will also use the most popular Machine Learning platform, Jupyter Notebook/Lab. During hands-on training, programmers will use actual Python codes in Jupyter notebook to run simple NLP Machine Learning projects. In the training, programmers will also get introduced popular NLP Machine Learning packages such as keras, pytorch, nltk, BERT, spacy and others.

Natural Language Processing (NLP) using RNN

Introduction of NLP – An area of artificial intelligence on an interaction between computer and human natural language. NLP can program computers to process and analyze natural language data.

- Input data – Language
- Output data - Language
**Popular NLP Implementation**

Text notation. # of inputs = # of outputs

![Diagram of a recurrent neural network](image)

Sentimental Analysis (PV signal) : \( x = \text{text}, y = 0/1 \) or \( 1 \) to \( 5 \)

![Diagram of a recurrent neural network](image)

Music generation \ Picture Description: \( x = \text{vector}, y = \text{text} \)

![Diagram of a recurrent neural network](image)

Machine translation : \( x = \text{text in English}, y = \text{text in French} \)

![Diagram of a recurrent neural network](image)

**NLP Machine Learning Model - Recurrent Neural Network**

Introduction – recurrent neural network model to use sequential information.

Why RNN?

- In traditional DNN, all inputs and outputs are independent of each other. But, in some case, they could be dependent.
- RNN is useful when inputs are dependent.
- Some problems such as text analysis and translation, we need to understand which words come before.
- RNN has a memory which captures previous information about what has been calculated so far.
Basic RNN Structure and Algorithms

\[ a^{t} = g(Wa* a^{t-1} + Wax * X^{t} + ba) \] when \( g \) is an activation function (e.g., tanh, ReLU)

\[ Y^{t} = g(Wya * a^{t} + by) \] when \( g \) is an activation function (e.g., sigmoid, softmax)

Note that \( a^{t} \) includes the information from previous \( X \).

RNN unit - LSTM (Long Short-Term Memory Unit)

- It is composed of 4 gates – input, forget, gate and output.
- LSTM remembers values over arbitrary time intervals and the 3 gates regulate the flow of information into and out of LSTM unit.
- LSTMs were developed to deal with the vanishing gradient problems.
- Relative insensitivity to gap length is an advantage of LSTM over RNNs.
Simple RNN architecture using NLP

- Input data – “I am smiling”, “I laugh now”, “I am crying”, “I feel good”, “I am not sure now”
- Embedding – to convert words to vector number
- LSTM – to learn language
- Softmax – to provide probability of output
- Output data - “very unhappy”, “unhappy”, “happy”, “very happy”

Natural Language Processing (NLP) procedures

1. Import data and preparation
2. Tokenizing – representing each word to numeric integer number: “the” to 50
3. Padding – fixing all the records to the same length
4. Embedding – representing word(numeric number) to vectors of numbers
   5o to [0.418, 0.24968, -0.41242, 0.1217, 0.34527, -0.044457, -0.49688, -0.17862, -0.00066023,...]
5. Training with RNN models

1. Import Data and Preparation
   - Import document to working area
- Convert document to sentences
- Split sentence to words
- Filter out punctuation
- Convert words to lower case

```python
In [3]:
## import Libraries
import numpy as np
import pandas as pd

## algorithm splitting train and test data
from sklearn.model_selection import train_test_split

## metric algorithm
from sklearn import metrics

In [60]:
inputs = ['Well Done', 'Good Work', 'Great effort', 'Nice Work', 'Excellent',
'Awesome', 'It is magnificent', 'How good are you', 'Outstanding', 'That is outstanding',
'Weak', 'Poor Effort', 'not good', 'poor work', 'Could have done better',
'It is low', 'That is not good', 'really bad', 'I am sad', 'That is cheap shot']

labels = [1,1,1,1,1,
1,1,1,1,1,
0,0,0,0,0,
0,0,0,0,0]
```

### 2. Tokenization
- Tokens - words
- Tokenizing - Representing word (numeric number) to integer number
- Word 'the' to 1
- Words are represented by dense vectors where a vector represents the projection of the word into a continuous vector space

```python
In [61]:
## Python Programming for importing word preparation
from keras.preprocessing.text import text_to_word_sequence

line =[
for text in inputs:
    words = text_to_word_sequence(text)
    line.append(words)
print(line)
```

```
[['well', 'done'], ['good', 'work'], ['great', 'effort'], ['nice', 'work'], ['excellent'], ['awesome'], ['it', 'is', 'magnificent'], ['how', 'good', 'are', 'you'], ['outstanding'], ['that', 'is', 'outstanding'], ['weak'], ['poor', 'effort'], ['not good', 'poor work', 'could have done better'],
['it is low', 'that is not good', 'really bad', 'i am sad', 'that is cheap shot']
```

### 3. Padding
- Padding - Preparing tokenized input data into the same length so that input data can go into Embedding layer
- It will make all the records to fixed, consistent length.

```python
In [62]:
## Python Programming for importing word preparation
from keras.preprocessing.text import Tokenizer

tokenizer = Tokenizer()
tokenizer.fit_on_texts(line)
tokenized_words = tokenizer.texts_to_sequences(line)
print(tokenized_words)
```

```
[[11, 5], [2, 3], [12, 6], [13, 3], [14], [15], [7, 1, 16], [17, 2, 18, 19], [8], [4, 1, 8], [20], [9, 6], [10, 2],
[9, 3], [21, 22, 5, 23], [7, 1, 24], [4, 1, 10, 2], [25, 26], [27, 28, 29], [4, 1, 30, 31]]
```
In [63]: from keras.preprocessing.sequence import pad_sequences
   : max_length = 5
   : padded_lines = pad_sequences(tokenized_words, maxlen=max_length, padding='post')

   : print(padded_lines)


```
[[11  5  0  0  0]
 [ 2  3  0  0  0]
 [12  6  0  0  0]
 [13  3  0  0  0]
 [14  0  0  0  0]
 [15  0  0  0  0]
 [ 7  1 16  0  0]
 [17  2 18 19  0]
 [ 8  0  0  0  0]
 [ 4  1  8  0  0]
 [20  0  0  0  0]
 [ 9  6  0  0  0]
 [10  2  0  0  0]
 [ 9  3  0  0  0]
[21 22  5 23  0]
 [ 7  1 24  0  0]
 [ 4  1 10  2  0]
[25 26  0  0  0]
[27 28 29  0  0]
 [ 4  1 30 31  0]]
```

4. Embedding

- Representing word(numeric number) to vectors of numbers
  - Word “the” to [0.418, 0.24968, -0.41242, 0.1217, 0.34527, -0.044457, -0.49688, -0.17862, -0.00066023, ....]
- Words are represented by dense vectors where a vector represents the projection of the word into a continuous vector space
- Embedding layer has embedding matrix which will be trained as a part of model.
- The shape of embedding matrix is (vocabulary size, # of vector)
- Embedding layer types
  - Keras Embedding layer – Random initiation on embedding matrix
  - Pre-trained embedding methods
    - Word2Vec
    - GloVe

In [64]: from gensim.models import Word2Vec
In [65]:
```python
# create NLP model using gensim for embedding
model_NLP = Word2Vec(line, size=100, min_count=1)
print("Gensim model vocab : ", model_NLP.wv.vocab)
print("Input X data ", line)
print("Embedding - Good : ", model_NLP["good"]).shape
```
```
Gensim model vocab : ['well'] gensim.models.keyedvectors.Vocab object at 0x000001C0A4E970F0, 'done' gensim.models.keyedvectors.Vocab object at 0x000001C0A4E97128, 'good' gensim.models.keyedvectors.Vocab object at 0x000001C0A4E9715B, 'work' gensim.models.keyedvectors.Vocab object at 0x000001C0A4E9718B, 'great' gensim.models.keyedvectors.Vocab object at 0x000001C0A4E9720F, 'nice' gensim.models.keyedvectors.Vocab object at 0x000001C0A4E97245, 'excellent' gensim.models.keyedvectors.Vocab object at 0x000001C0A4E97274, 'awesome' gensim.models.keyedvectors.Vocab object at 0x000001C0A4E972A3, 'it' gensim.models.keyedvectors.Vocab object at 0x000001C0A4E972C2, 'is' gensim.models.keyedvectors.Vocab object at 0x000001C0A4E972F1, 'magnificent' gensim.models.keyedvectors.Vocab object at 0x000001C0A4E9732B, 'how' gensim.models.keyedvectors.Vocab object at 0x000001C0A4E9735A, 'are' gensim.models.keyedvectors.Vocab object at 0x000001C0A4E97389, 'you' gensim.models.keyedvectors.Vocab object at 0x000001C0A4E973B8, 'outstanding' gensim.models.keyedvectors.Vocab object at 0x000001C0A4E97437, 'that' gensim.models.keyedvectors.Vocab object at 0x000001C0A4E97476, 'weak' gensim.models.keyedvectors.Vocab object at 0x000001C0A4E974A5, 'poor' gensim.models.keyedvectors.Vocab object at 0x000001C0A4E974D4, 'not' gensim.models.keyedvectors.Vocab object at 0x000001C0A4E9750C, 'could' gensim.models.keyedvectors.Vocab object at 0x000001C0A4E9753B, 'have' gensim.models.keyedvectors.Vocab object at 0x000001C0A4E9756A, 'better' gensim.models.keyedvectors.Vocab object at 0x000001C0A4E97599, 'lo w' gensim.models.keyedvectors.Vocab object at 0x000001C0A4E975C8, 'am' gensim.models.keyedvectors.Vocab object at 0x000001C0A4E97632, 'better' gensim.models.keyedvectors.Vocab object at 0x000001C0A4E97661, 'bad' gensim.models.keyedvectors.Vocab object at 0x000001C0A4E9769A, 'am' gensim.models.keyedvectors.Vocab object at 0x000001C0A4E97729, 'true' gensim.models.keyedvectors.Vocab object at 0x000001C0A4E97758, 'shot' gensim.models.keyedvectors.Vocab object at 0x000001C0A4E97787, 'Input X data : [\'well\', \'done\', \'good\', \'work\', \'great\', \'effort\', \'nice\', \'work\', \'could\', \'have\', \'done\', \'better\', \'it\', \'is\', \'low\', \'that\', \'is\', \'not\', \'good\', \'really\', \'bad\', \'i\', \'am\', \'sad\', \'that\', \'is\', \'cheap\', \'sho t\'] Embedding - Good :
```
```
C:\Users\kevin\Anaconda3\envs\nlp\lib\site-packages\ipykernel_launcher.py:5: DeprecationWarning: Call to deprecated `__getitem__` (Method will be removed in 4.0.8, use self.wv.__getitem__() instead).
  ***
C:\Users\kevin\Anaconda3\envs\nlp\lib\site-packages\ipykernel_launcher.py:6: DeprecationWarning: Call to deprecated `__getitem__` (Method will be removed in 4.0.8, use self.wv.__getitem__() instead).
```
```
In [66]:
```python
# Create word Vectors
word_vectors = model_NLP.wv
```
5. Train with RNN model

```python
## import RNN model
from keras.preprocessing.text import Tokenizer
from keras.models import Sequential
from keras.layers import Dense, LSTM, Embedding

tokenizer = Tokenizer().fit_on_texts(X_train_rnn) X_input = tokenizer.texts_to_sequence(X_train_rnn)
```

```python
model_rnn = Sequential()
model_rnn.add(Embedding(input_dim=40, output_dim=100, input_length=max_length))
model_rnn.add(LSTM(10))
model_rnn.add(Dense(10, activation='relu'))
model_rnn.add(Dense(1, activation='sigmoid'))

model_rnn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model_rnn.summary()
```

```
<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Param #</th>
</tr>
</thead>
<tbody>
<tr>
<td>embedding_6 (Embedding)</td>
<td>(None, 5, 100)</td>
<td>4000</td>
</tr>
<tr>
<td>lstm_6 (LSTM)</td>
<td>(None, 10)</td>
<td>4440</td>
</tr>
<tr>
<td>dense_11 (Dense)</td>
<td>(None, 10)</td>
<td>110</td>
</tr>
<tr>
<td>dense_12 (Dense)</td>
<td>(None, 1)</td>
<td>11</td>
</tr>
</tbody>
</table>

Total params: 8,561
Trainable params: 8,561
Non-trainable params: 0
```

```python
X_train_rnn = padded_lines
y_train_rnn = labels
```
In [71]: ## Train RNN model with
model_rnn.fit(X_train_rnn, y_train_rnn, epochs=10)

## Predict with the trained RNN model
rnn_preds = model_rnn.predict(X_train_rnn)

## Evaluate RNN model
model_rnn.evaluate(X_train_rnn, y_train_rnn)

Epoch 1/10
20/20 [==================================] - 2s 99ms/step - loss: 0.6933 - acc: 0.4500
Epoch 2/10
20/20 [==================================] - 0s 299us/step - loss: 0.6921 - acc: 0.7500
Epoch 3/10
20/20 [==================================] - 0s 349us/step - loss: 0.6918 - acc: 0.6500
Epoch 4/10
20/20 [==================================] - 0s 349us/step - loss: 0.6908 - acc: 0.6500
Epoch 5/10
20/20 [==================================] - 0s 299us/step - loss: 0.6903 - acc: 0.6500
Epoch 6/10
20/20 [==================================] - 0s 349us/step - loss: 0.6897 - acc: 0.6500
Epoch 7/10
20/20 [==================================] - 0s 349us/step - loss: 0.6891 - acc: 0.8500
Epoch 8/10
20/20 [==================================] - 0s 399us/step - loss: 0.6883 - acc: 0.8500
Epoch 9/10
20/20 [==================================] - 0s 299us/step - loss: 0.6875 - acc: 0.8500
Epoch 10/10
20/20 [==================================] - 0s 299us/step - loss: 0.6868 - acc: 0.8500

Let's test NLP model

In [77]: test_data = 'you are bad'

## Data Preparation
test_data2 = []
words2 = text_to_word_sequence(test_data) # Vocabulization
test_data2.append(words2)
test_data3 = tokenizer.texts_to_sequences(test_data2) # Tokenization
test_data4 = pad_sequences(test_data3, maxlen=max_length, padding='post') # padded Line

## Prediction
rnn_preds1 = model_rnn.predict(test_data4)
rnn_preds2 = model_rnn.predict_classes(test_data4)
rnn_preds3 = np.where(rnn_preds2 == 1, 'Positive', 'Negative')

print(rnn_preds1)
print(rnn_preds3)

[[0.48418682]]
[[ 'Negative']]

NLP Implementation

- Speech Recognition
- Music Generation
- Sentiment Classification
- Machine Translation
- Video Activity Recognition
- Texts Classification and Extraction
- DNA Sequence Analysis
- Chatbot
CONCLUSION

Natural Language Processing (NLP) is one of the most popular Machine Learning implementation. NLP is used to analyze the human language, allowing the machine to understand what human communicates. Its market is expected to grow from USD 11.6 billion in 2020 to USD 35.1 billion in 2024 at the rate of 20.3%.

Hands-on training has walked thru the basic NLP processes from Embedding, Tokenization and RNN model training. The user could build a lot more complicated NLP models to analyze complex language projects.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Please contact at:
Kevin Lee
AVP of Machine Learning and AI Consultant
Kevin.lee@genpact.com (mailto:Kevin.lee@genpact.com)
Tweet: @HelloKevinLee
LinkedIn: www.linkedin.com/in/HelloKevinLee/ (http://www.linkedin.com/in/HelloKevinLee/)