PharmaSUG 2021 : Paper HT-210

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

Kevin Lee, Genpact

ABSTRACT

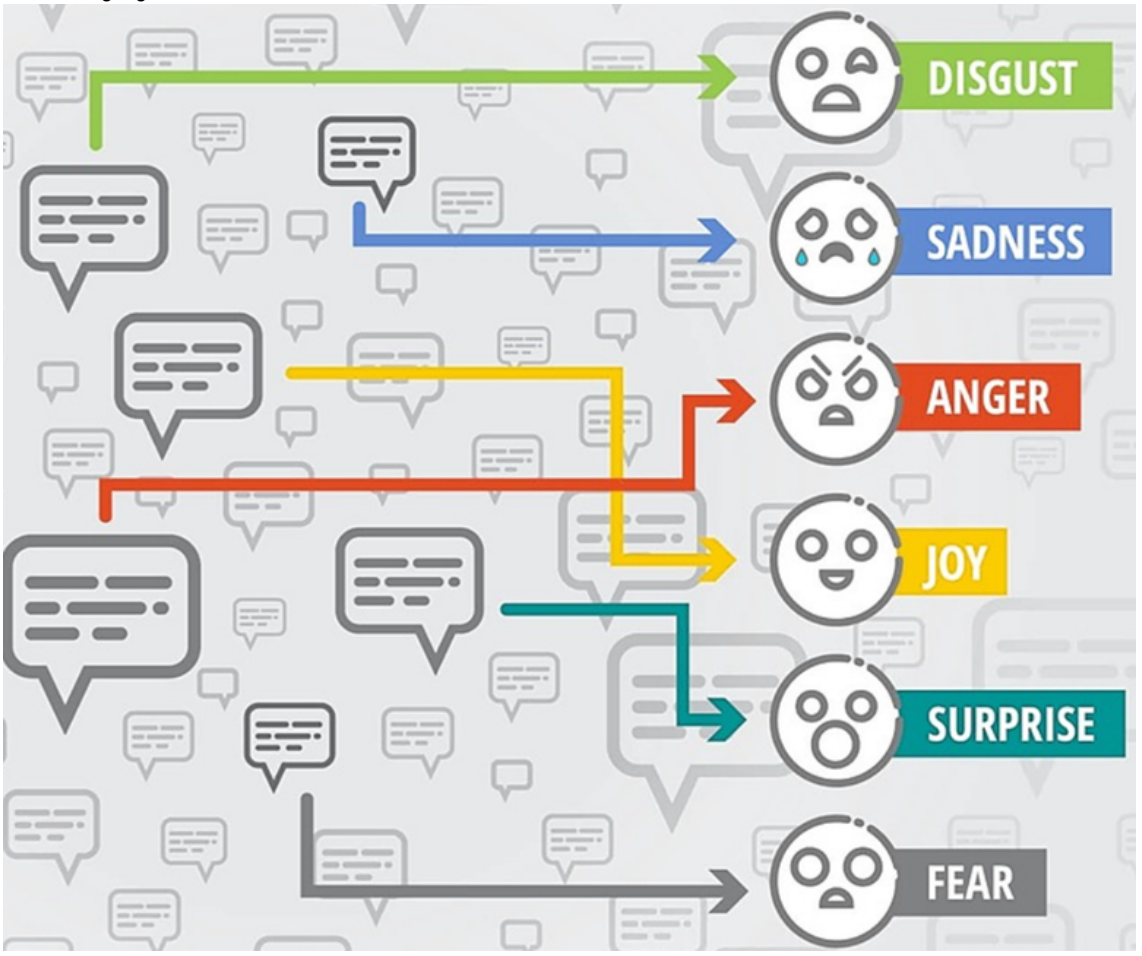
One of the most popular Machine Learning implementation is Natural Language Processing (NLP). NLP is a Machine Learning application or service which are able to understand human language. Some practical implementations are speech recognition, machine translation and chatbot. Sri, Alexa and Google Home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

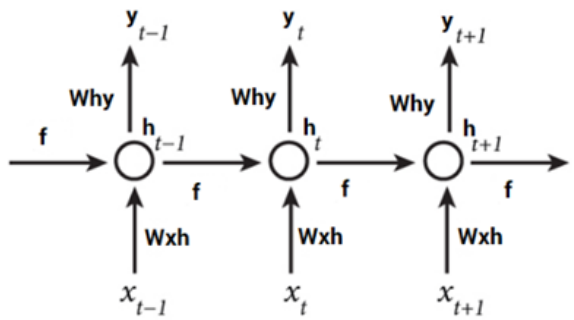
Introduction of NLP – An area of artificial intelligence on an interaction between computer and human natural language. NLP can program computers to process and analyze natural language data.

- Input data – Language
- Output data - Language

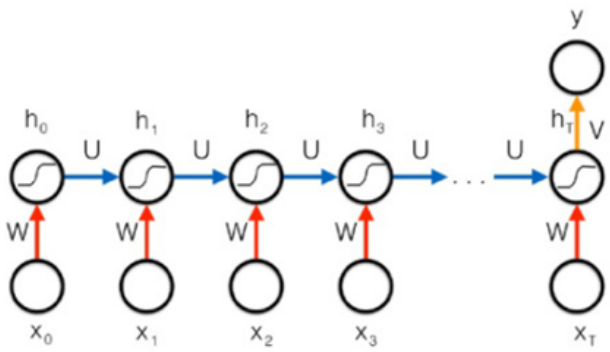


Popular NLP Implementation

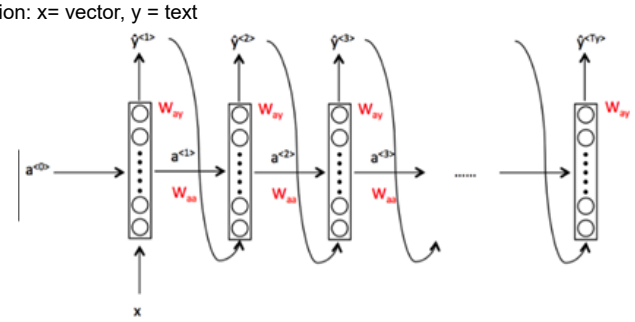
Text notation. # of inputs = # of outputs



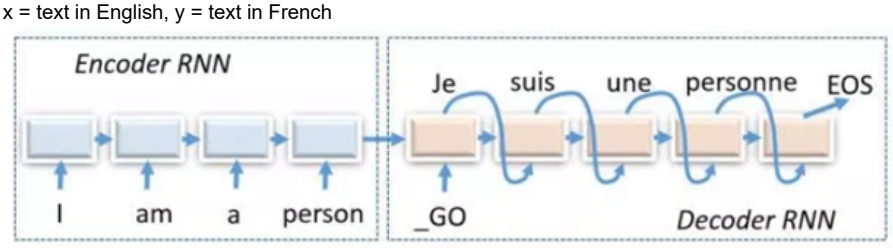
Sentimental Analysis (PV signal) : x = text, y = 0/1 or 1 to 5



Music generation \ Picture Description: x= vector, y = text



Machine translation : x = text in English, y = text in French

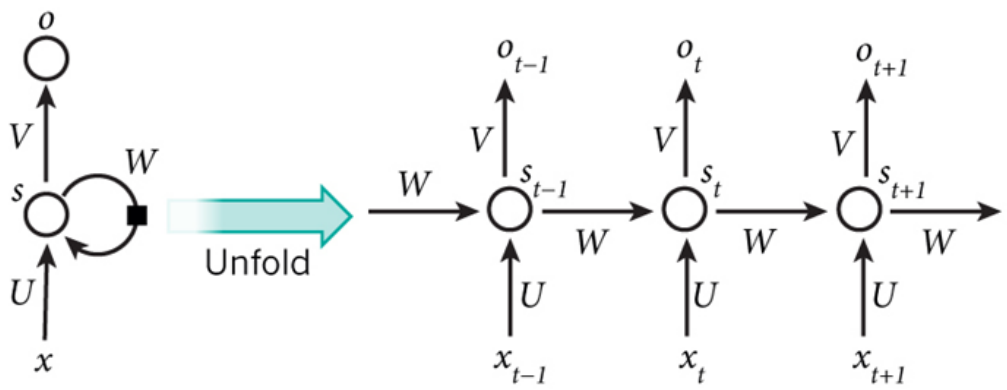


NLP Machine Learning Model - Recurrent Neural Network

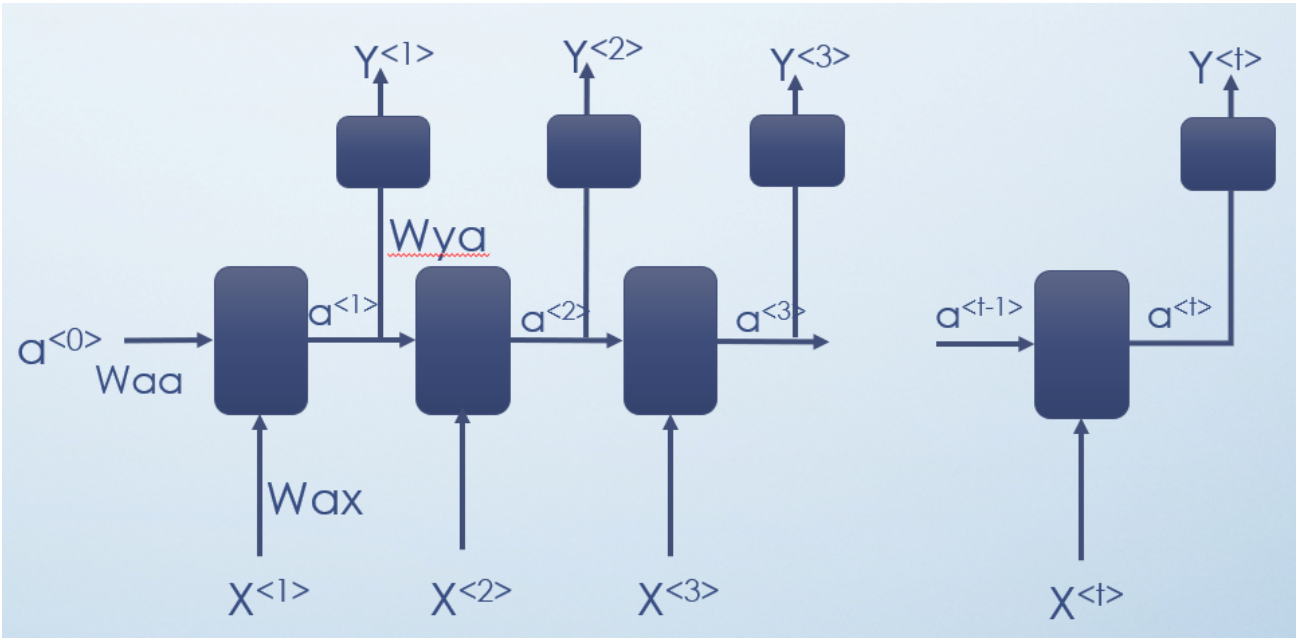
Introduction – recurrent neural network model to use sequential information.

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

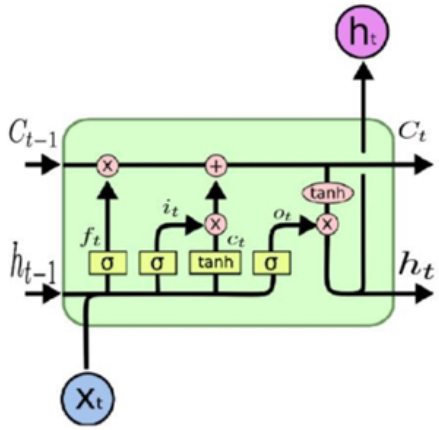


$a^{<t>}$ =g(Waa*$a^{<t-1>}$ + Wax*$X^{<t>}$ +ba) when g is an activation function (e.g., tahn, ReLU)
$Y^{<t>}$ =g(Wya*$a^{<t>}$ + by) when g is an activation function (e.g., sigmoid, softmax)

Note that $a^{<t>}$ includes the information from previous X.

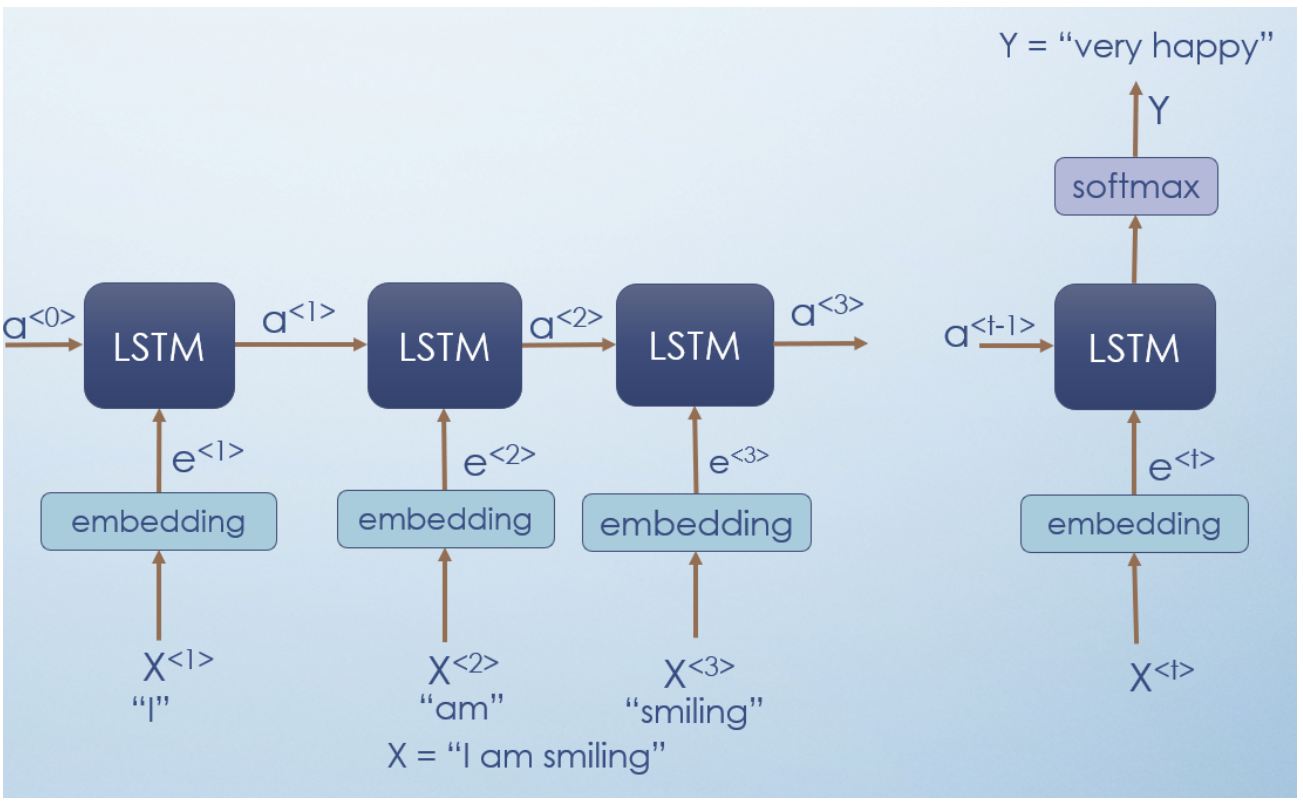
RNN unit - LSTM (Long Short-Term Memory Unit)

- 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

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

## algorithm spliting 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]
```

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', 'i
s', 'magnificent'], ['how', 'good', 'are', 'you'], ['outstanding'], ['that', 'is', 'outstanding'], ['weak'], ['poo
r', 'effort'], ['not', 'good'], ['poor', 'work'], ['could', 'have', 'done', 'better'], ['it', 'is', 'low'], ['tha
t', 'is', 'not', 'good'], ['really', 'bad'], ['i', 'am', 'sad'], ['that', 'is', 'cheap', 'shot']]
```

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

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]]
```

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.

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]:
```
## import gensim model
from gensim.models import Word2Vec
```

In [65]:
```
## creat NLP model using gensim for embeding
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   : \n", model_NLP['good'])
print(model_NLP['good'].shape)
```

```
Gensim model vocab :  {'well': <gensim.models.keyedvectors.Vocab object at 0x000001C0A4E970F0>, 'done': <gensim.mod
els.keyedvectors.Vocab object at 0x000001C0A4E97128>, 'good': <gensim.models.keyedvectors.Vocab object at 0x000001C
0A4E97198>, 'work': <gensim.models.keyedvectors.Vocab object at 0x000001C0A4E971D0>, 'great': <gensim.models.keyedv
ectors.Vocab object at 0x000001C0A4E97208>, 'effort': <gensim.models.keyedvectors.Vocab object at 0x000001C0A4E9724
0>, 'nice': <gensim.models.keyedvectors.Vocab object at 0x000001C0A4E97278>, 'excellent': <gensim.models.keyedvecto
rs.Vocab object at 0x000001C0A4E972B0>, 'awesome': <gensim.models.keyedvectors.Vocab object at 0x000001C0A4E972E8>,
'it': <gensim.models.keyedvectors.Vocab object at 0x000001C0A4E97320>, 'is': <gensim.models.keyedvectors.Vocab obje
ct at 0x000001C0A4E97358>, 'magnificent': <gensim.models.keyedvectors.Vocab object at 0x000001C0A4E97390>, 'how': <
gensim.models.keyedvectors.Vocab object at 0x000001C0A4E973C8>, 'are': <gensim.models.keyedvectors.Vocab object at
0x000001C0A4E97400>, 'you': <gensim.models.keyedvectors.Vocab object at 0x000001C0A4E97438>, 'outstanding': <gensi
m.models.keyedvectors.Vocab object at 0x000001C0A4E97470>, 'that': <gensim.models.keyedvectors.Vocab object at 0x00
0001C0A4E974A8>, 'weak': <gensim.models.keyedvectors.Vocab object at 0x000001C0A4E974E0>, 'poor': <gensim.models.ke
yedvectors.Vocab object at 0x000001C0A4E97518>, 'not': <gensim.models.keyedvectors.Vocab object at 0x000001C0A4E975
50>, 'could': <gensim.models.keyedvectors.Vocab object at 0x000001C0A4E97588>, 'have': <gensim.models.keyedvectors.
Vocab object at 0x000001C0A4E975C0>, 'better': <gensim.models.keyedvectors.Vocab object at 0x000001C0A4E975F8>, 'lo
w': <gensim.models.keyedvectors.Vocab object at 0x000001C0A4E97630>, 'really': <gensim.models.keyedvectors.Vocab ob
ject at 0x000001C0A4E97668>, 'bad': <gensim.models.keyedvectors.Vocab object at 0x000001C0A4E976A0>, 'i': <gensim.m
odels.keyedvectors.Vocab object at 0x000001C0A4E976D8>, 'am': <gensim.models.keyedvectors.Vocab object at 0x000001C
0A4E97710>, 'sad': <gensim.models.keyedvectors.Vocab object at 0x000001C0A4E97748>, 'cheap': <gensim.models.keyedve
ctors.Vocab object at 0x000001C0A4E97780>, 'shot': <gensim.models.keyedvectors.Vocab object at 0x000001C0A4E977B8>}
Input X data       :  [['well', 'done'], ['good', 'work'], ['great', 'effort'], ['nice', 'work'], ['excellent'],
['awesome'], ['it', 'is', 'magnificent'], ['how', 'good', 'are', 'you'], ['outstanding'], ['that', 'is', 'outstandi
ng'], ['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', 'sho
t']]
Embedding - Good   :
 [-1.4533636e-03  4.6793139e-03 -2.0805956e-03 -2.7451206e-03
 -6.3593144e-04 -4.2631300e-03 -3.7492267e-05 -3.7312353e-04
 -1.1376112e-03  3.8287267e-03  1.8349462e-03  2.0946632e-03
  4.0369807e-03  7.1678520e-04 -1.7933737e-03  1.3851834e-03
  1.9001529e-03 -1.8953750e-03  1.2831722e-03  4.8779659e-03
 -2.6340478e-03  1.9446217e-03 -1.8672617e-03  2.8829912e-03
  1.2137515e-03 -2.1444922e-03  4.9312273e-04  2.8437264e-03
 -1.5472607e-03  3.0090131e-03 -1.8892944e-03 -1.9783955e-03
 -1.1676401e-03  4.8910012e-03 -2.2670643e-03 -4.2750966e-03
  4.3589966e-03 -9.1649877e-04 -3.7843182e-03 -1.0158399e-03
 -2.5928186e-03 -1.0462092e-03 -4.2352448e-03 -3.0230389e-03
  1.2504178e-04  4.3295622e-03 -1.8677751e-03 -1.4039027e-03
  1.3582467e-03 -5.2804593e-04  1.0884306e-03  3.7401523e-03
 -1.3738587e-03 -9.7763073e-04  4.0383716e-03 -3.1996120e-03
  3.0188081e-03  4.9238914e-04 -2.0367047e-03  8.7173725e-04
  3.6459105e-04 -3.9577824e-03  2.7407527e-03 -4.6094321e-03
 -2.5742382e-03  1.6579209e-03  1.2938243e-03 -2.5387951e-03
  4.6142661e-03 -1.2891849e-03  1.7220690e-03 -4.9882825e-03
 -2.8447541e-03  3.7592719e-03  4.8412890e-03  2.0945731e-03
 -2.6060701e-03 -4.8655886e-03 -2.3329542e-03  5.6088017e-04
  1.2234852e-04 -2.4019456e-03  1.9249524e-03  2.4553921e-04
  2.9099218e-03  2.9856393e-03  5.0158519e-04  2.1499090e-03
  1.3449148e-03  2.0016772e-03 -1.1095370e-03  3.0042202e-04
  7.8377553e-04 -1.9172652e-03  4.5066546e-03  3.0026180e-05
  4.0907981e-03  1.7273215e-03  3.6051169e-03  1.1216741e-03]
(100,)


C:\Users\kevin\Anaconda3\envs\nlp\lib\site-packages\ipykernel_launcher.py:5: DeprecationWarning: Call to deprecated
`__getitem__` (Method will be removed in 4.0.0, 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 4.0.0, use self.wv.__getitem__() instead).
```

In [66]:
```
## Create word Vectors
word_vectors = model_NLP.wv
```

In [67]:
```
word_vectors.similar_by_word(word='good', topn=5)
```

Out[67]:
```
[('awesome', 0.1937175989151001),
 ('cheap', 0.18643496930599213),
 ('am', 0.11635179072618484),
 ('outstanding', 0.1018410474061966),
 ('shot', 0.09762841463088989)]
```

5. Train with RNN model

In [68]:
```
## import RNN model
from keras.preprocessing.text import Tokenizer
from keras.models import Sequential
from keras.layers import Dense, LSTM, Embedding
```

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

In [69]:
```
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()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_6 (Embedding)      (None, 5, 100)            4000
_________________________________________________________________
lstm_6 (LSTM)                (None, 10)                4440
_________________________________________________________________
dense_11 (Dense)             (None, 10)                110
_________________________________________________________________
dense_12 (Dense)             (None, 1)                 11
=================================================================
Total params: 8,561
Trainable params: 8,561
Non-trainable params: 0
_________________________________________________________________
```

In [70]:
```
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: 0.7500
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
```

Let's test NLP model

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

## Data Prepareation
test_data2 =[ ]
words2 = text_to_word_sequence(test_data) ## Vocabulization
test_data2.append(words)
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)
```

```
[[0.48418602]]
[['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 35.1billion 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/)