# Natural Language Processing using Deep Learning

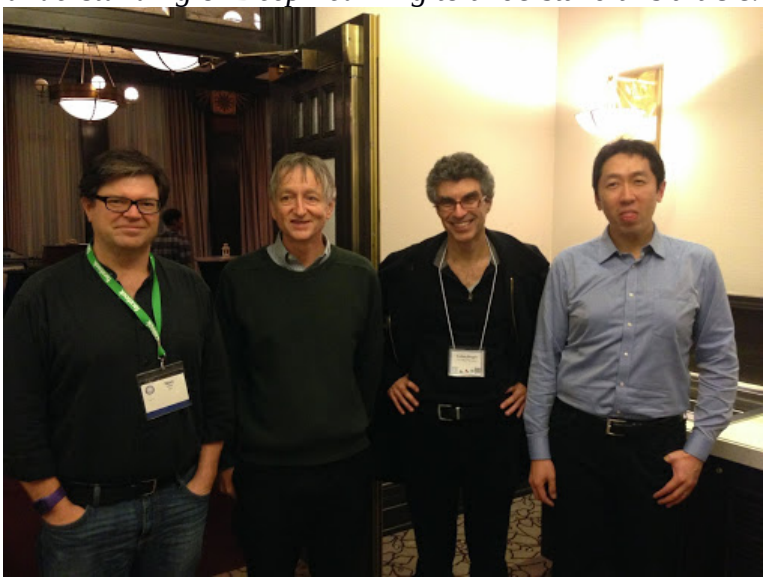
## INTRODUCTION

The goal of natural language processing is to understand human language. Internet is full of text data which has not been understood well and can convey a lot of information that can be used to perform multiple task such as text summarisation, machine translation, information extraction etc. It is unsurprising that a lot nlp startups are trying to take advantage of this opportunity.

Basic Machine Learning algorithms like logistic regression, decision trees etc require hand designed features that can only be done for simple applications. Human Language is very complex and it is impossible for us to hand label all the features. So we need to automate feature representation and also do it in a way that is computationally efficient. This can be done by Deep Learning.

Deep Learning has become very popular recently and has been used in various applications such as Self Driving Cars, Alpha Go, Machine Translation, Image Recognition etc. The recent rise of Deep Learning . The recent advance in Deep Learning can be attributed to the huge amount of *data, GPU* that can perform complex computations quickly and the *powerful algorithms* recently discovered.

This article contains some *advanced* Deep Learning concepts. You *need* to have some *basic understanding* of *Deep Learning* to understand this article.



*https://www.kdnuggets.com/2015/03/talking-machine-deep-learning-gurus-p1.html*

*From left to right* Yan LeCun, Geoffrey Hinton, Yoshua Bengio and Andrew Ng are the pioneers of Deep Learning and first to establish Deep Learning research labs.

## REPRESENTATION FOR NLP

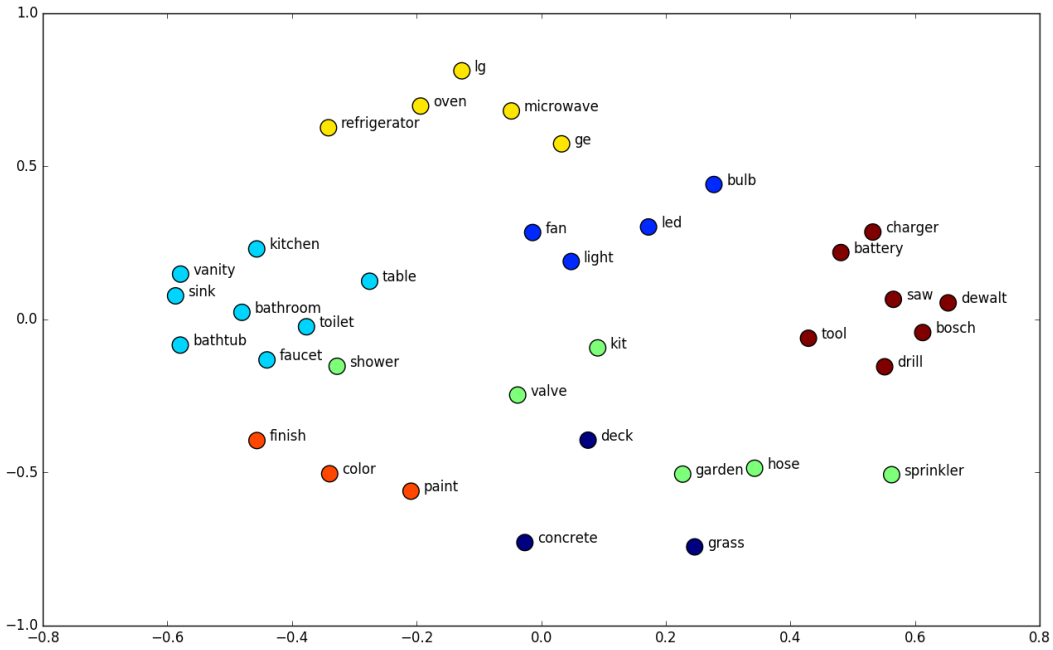
The basic representation of NLP is very important, because it helps us understand the low level details of the language.

BAG-OF-WORDS model

Bag of words algorithm counts the number of times a word appears in a document. If a word appears a lot of times in a document, there is a high probability that the article is about that. For eg :- if the word smartphone occurs a lot in the article, that means that the article is about the word. But the problem with bag of words is that it pays no attention to the order in which the words appear, and so it does not help us understand the semantic structure of the language.

WORD EMBEDDINGS

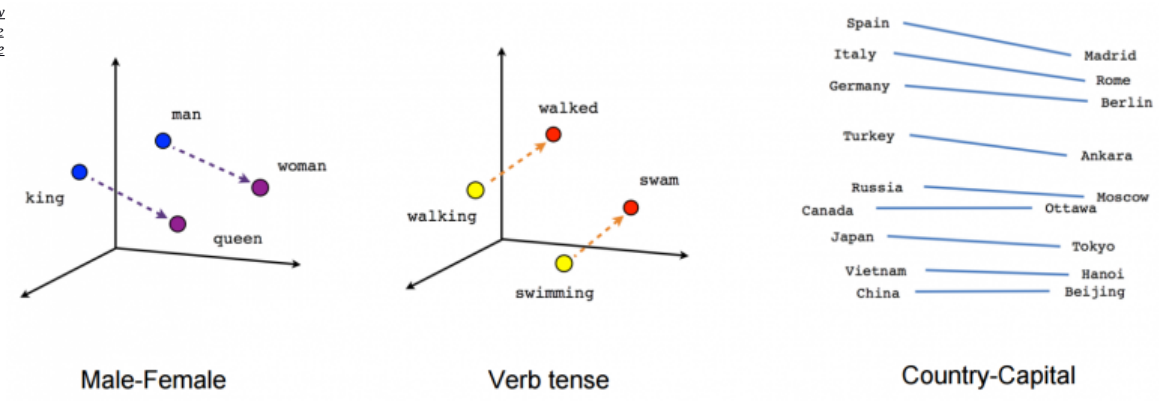
Word Embeddings represent words in the form of multi dimensional vectors. Words that have similar meanings have similar representations.




https://www.shanelynn.ie/get-busy-with-word-embeddings-introduction/

*This image shows a high dimensional view of word embeddings. Here you will find that the words that have similar meanings are placed closer.*

https://www.shanelynn.ie/get-busy-with-word-



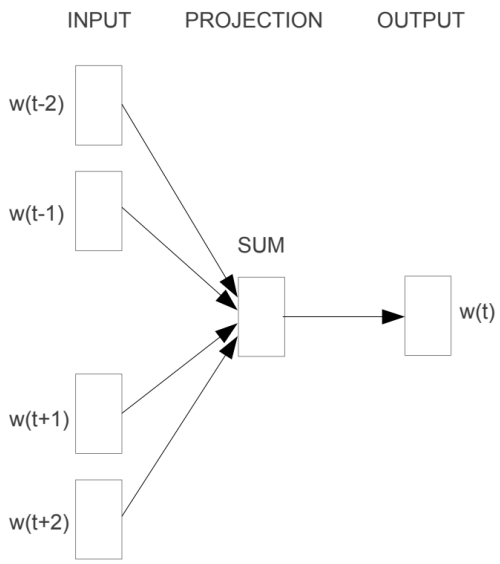

embeddings-introduction/

*In this image you will find that the word embedding understand semantic meaning. The words that have similar relation are placed at the same distance from each other.*

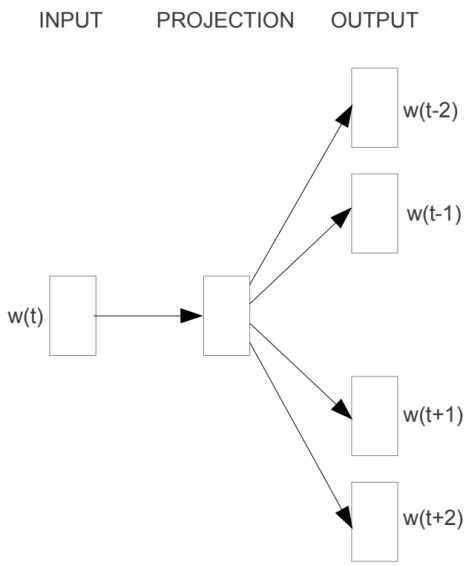
## Word2Vec Model

*I*t was the first word embedding model introduced by *Mikolov et al* in 2013. It is of two types CBOW(Continuous Bag of Words) and Skip-Gram.

In CBOW, we predict the current word based on its context. Each word has a context window n, and based on the words present in the context window, we predict the current word.
In Skip Gram we predict the surrounding word using the given word.



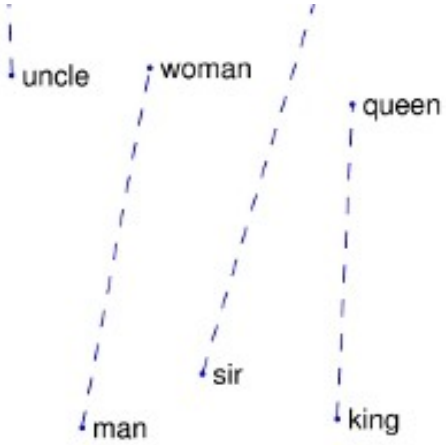

CBOW




Skip-gram

GloVe Embeddings

GloVe Embedding was introduced by Chris Manning and Richard Socher of Stanford Research Lab in 2014.




This is the high level representation of GloVe vectors. But it looks similar to Word2Vec. SO what's the difference ?

GloVe Vector gives more weightage to words that occur less frequently(example smartphone) and less weightage to words that occur frequently(example an,the). This is done because words that occur less frequently convey more information. GloVe Embeddings are shown to give better performance than Word2Vec.

$$J= \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \widetilde{w}_j + b_i + \widetilde{b}_j - \log\left(X_{ij}\right)\right)^2$$

This formula represents GloVe Vector.

$\sum_{i,j=1}^{V}$ ❑ i is the current word and j is the context word and V is the total numbers of words present.

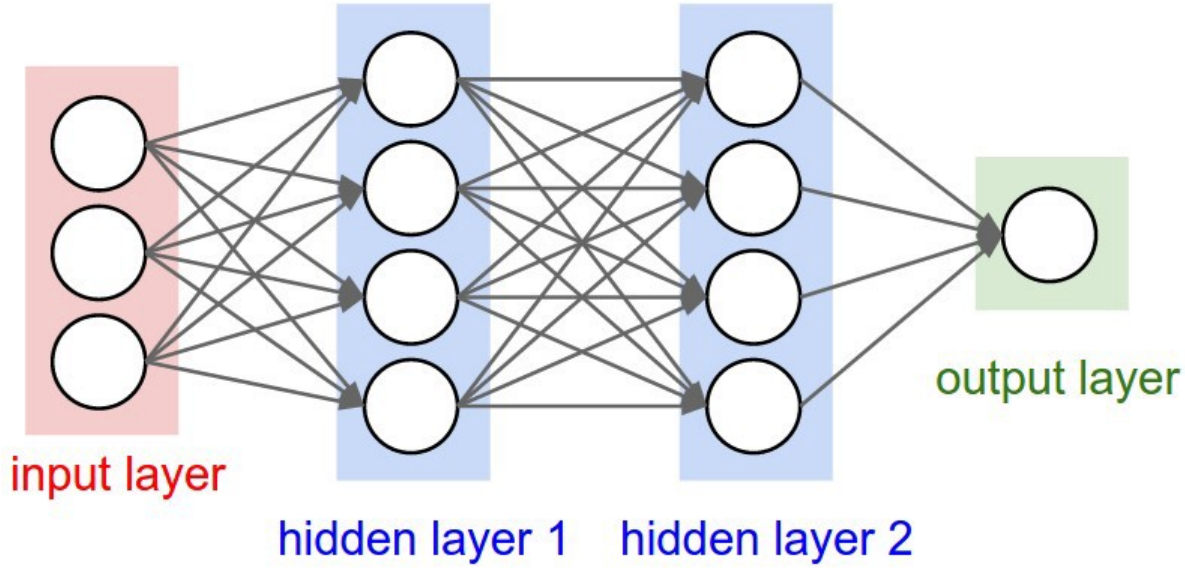
$f(X_{ij})$ represents the importance of the word. Words that are less frequent are given more importance.

$w_i^T \widetilde{w}_i$ represents the vector multiplication of current word and context word and the output tells us the degree of correlation between the two words. If the result of the word is high, it means that the words are highly correlated and vice versa.

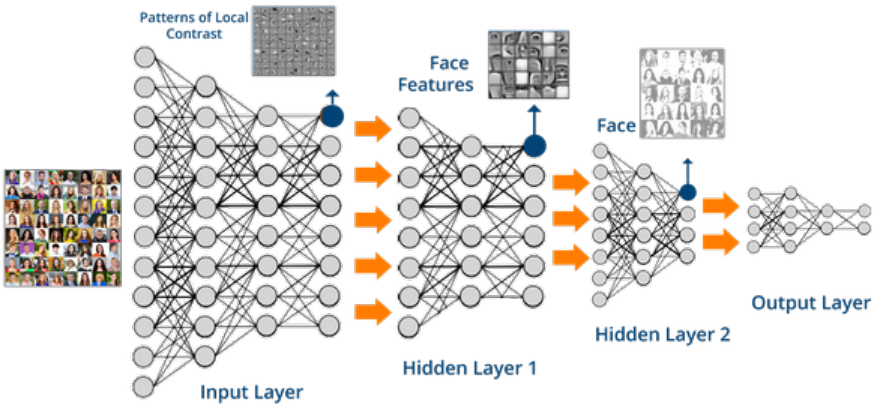
$b_i$ and $b_j$ are the bias for current word and context word respectively.
$X_{ij}$ tells us the frequency of occurrence of words i and j together.

# Deep Learning

Neural Networks are algorithms inspired by the human brain. Deep Learning algorithms are large neural networks.



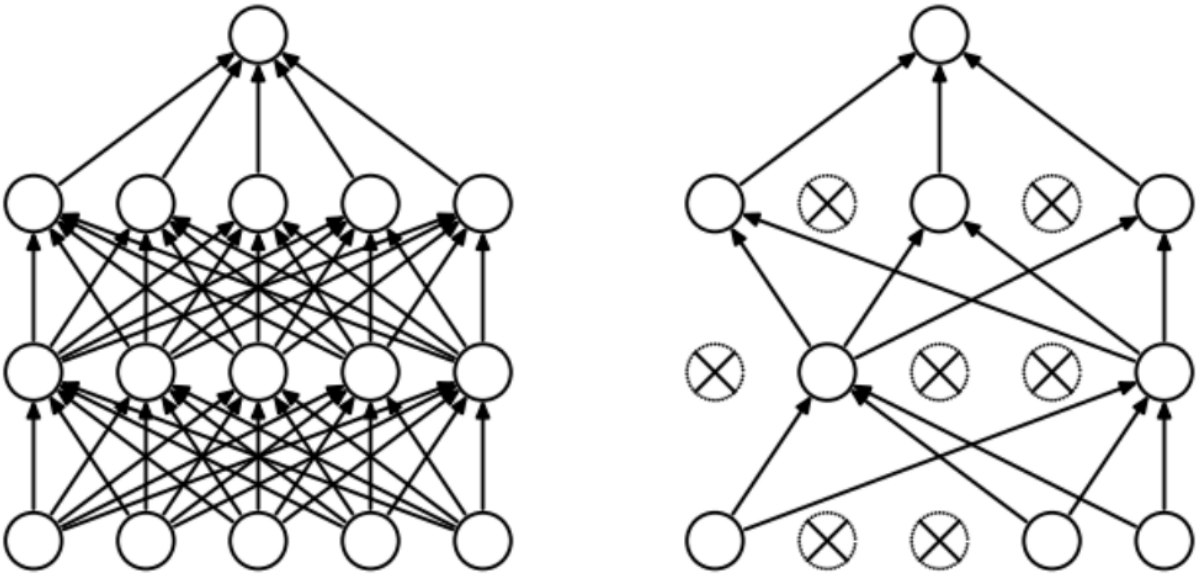

This image shows the *basic feedforward neural network.*These layers are called as dense layer because every neuron in the present layer is connected to neuron in the next layer. Input layer takes in the input and performs some computation on it and sends the output to the first hidden layer which does the same, and hidden layer performs the computation and sends the output to the output layer which tells us the label of the data.
The neural network has multiple hidden layers that allows it to learn complex features.




In this image you can see that the initial layers learn simple features, the later layers builds up on them to learn more complex features. This hierarchical feature learning has an additional advantage that it is computationally efficient. The layers have to build on the features learnt by the previous layer and does not learn the features from scratch, which saves a lot of computation.

The basic feedforward neural networks cannot be used to understand natural language. To understand natural language data we have to understand large sequences of data, which is not its feature.

DROPOUT

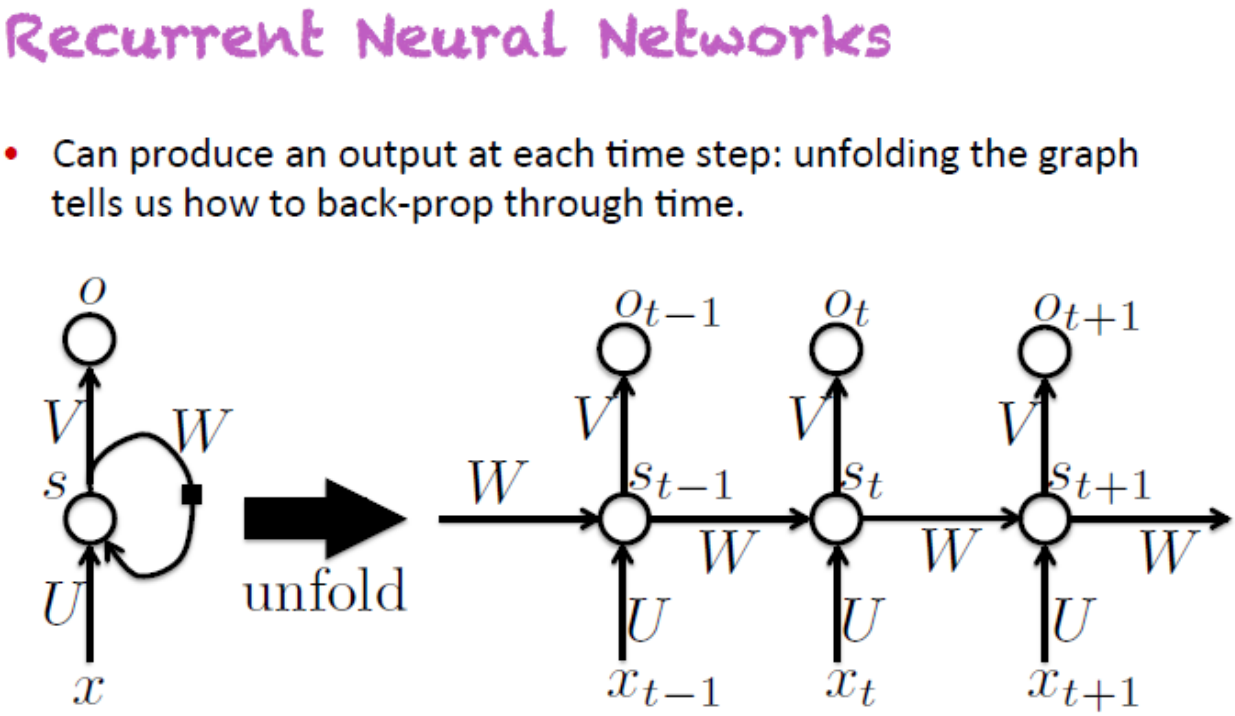


(a) Standard Neural Net (b) After applying dropout.

Dropout is a technique applied to

RECURRENT NEURAL NETWORK

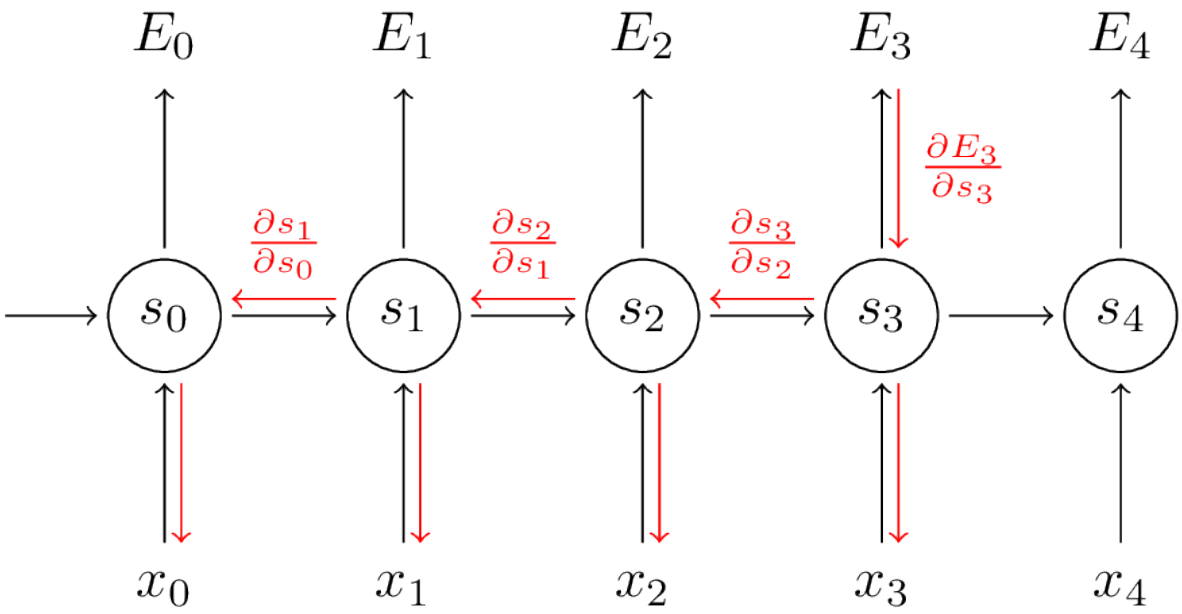
Recurrent Neural Networks are different from feedforward neural network because they have a time dimension. They are used for text classification, machine translation, image captioning etc.




In this image we have to predict a word $x_t$. We have the word $x_{t-1}$(previous word), and we multiply it by a weight W and we get output $o_{t-1}$. The output $o_{t-1}$ is the word we are looking for, $x_t$. Now we have to find the next word $x_{t+1}$. We feed $x_t(o_{t-1})$ which is the output of the previous step as the input to the neuron, we multiply it with the *same* word that was used in the previous step, and we get the next word $o_t(x_{t+1})$. Note that the weight w is same.

This means that the neural network can be summarised as that shown the left side of the image. The input keeps changing but the weight stays the same.

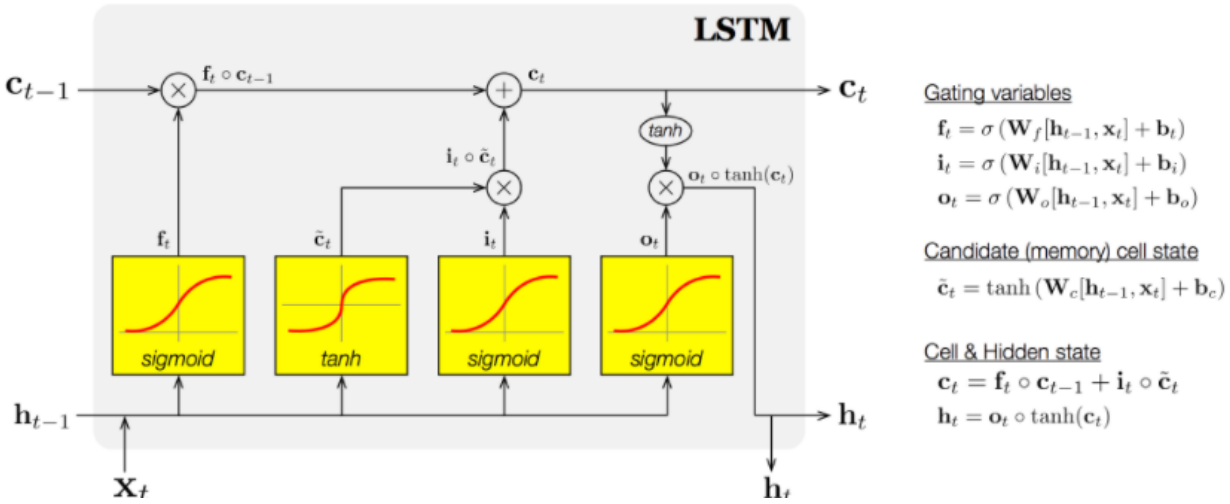
The problem with recurrent neural network is that it suffers from *gradient vanishing problem.* I will explain it in short. When you train a neural network you *change the weights of the neuron slightly in order to reduce the loss function(reduce error).* In order to reduce the error we have differentiation the loss function with weight. When there are large sequences of data which is very common in natural language data, the output of the differentiation is close to zero and the training of the neural network stops.




This image shows how gradients are calculated.

## LSTM(Long Short Term Memory)

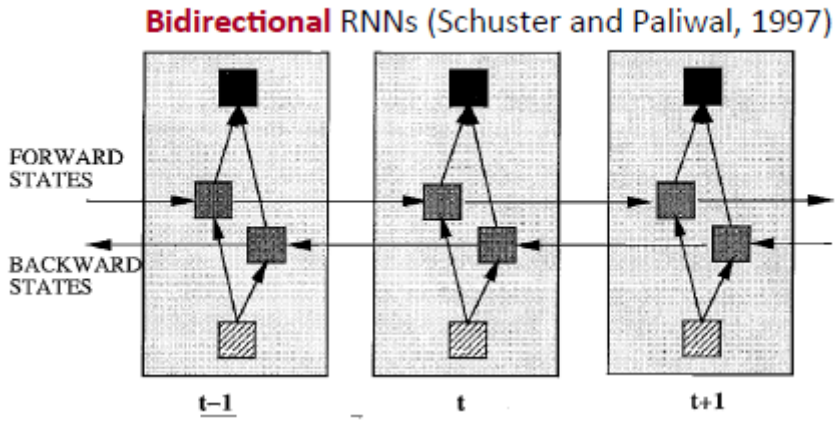
To deal with gradient vanishing problem LSTM is used.




This represents LSTM. It looks complex, so lets break it down.

$f_t$ represents forget gate. This is the most important part of LSTM. This helps us forget information that is not useful. This helps us deal with large sequence, as the unimportant parts of the sequence is removed, the neural network has to learn less and this helps it deal with gradient vanishing problem.
$i_t$ is the input gate. If the input word contains important information, it will remember the information forget it.
$o_t$ represents the output gate.
Both these gates have sigmoid function that introduces non-linearity.
$\widetilde{C}_t$ represents the actual value that has to be remembered. It has tanh function instead of sigmoid function because tanh function has range between -1 and +1 and it helps us remember the actual value.
Cell and hidden state combine Gating variable and Memory cell state to give the output.

Bidirectional LSTM




In unidirectional LSTM the direction of input is in only one direction i.e from the first word to the last word. In Bidirectional LSTM has a neuron that has direction of input from first word to the last word and the other neuron has input from the last word to the first word. So when we train the neural network, the word has the context in both directions.

**SAMPLE CODE**

This example shows the neural network trained on Kaggle Competition Toxic Comments Classification challenge. The Conversation AI team, a research initiative founded by Jigsaw and Google (both a part of Alphabet) are working on tools to help improve online conversation. One area of focus is the study of negative online behaviors, like toxic comments (i.e. comments that are rude, disrespectful or otherwise likely to make someone leave a discussion)
Look at this link for more details. https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge
You are provided with a large number of Wikipedia comments which have been labeled by human raters for toxic behavior. The types of toxicity are:

toxic severe_toxic
obscene
threat
insult
identity_hate
You must create a model which predicts a probability of each type of toxicity for each comment.
You can download train.csv.zip here https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/data. The dataset will then be split into training and validation sets.

We will use keras, a deep learning library built on top of tensorflow to build the model. You can download the code from here https://github.com/balajibalasubramanian/Kaggle-Toxic-Comments-Challenge

```
#import libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Embedding
from keras.layers import
Bidirectional,LSTM,RepeatVector,TimeDistributed,Activation
from keras.optimizers import Adam
from keras.layers import BatchNormalization, Flatten, Conv1D,
MaxPooling1D,GlobalMaxPool1D,CuDNNLSTM,CuDNNGRU

from keras.layers import Dropout,SpatialDropout1D
from keras.callbacks import EarlyStopping, ModelCheckpoint,
ReduceLROnPlateau

# read in the data

df_train = pd.read_csv('data/train.csv')
df_test = pd.read_csv('data/test.csv')

print(df_train.shape)
print(df_test.shape)

# combine the train and test sets for encoding and padding. Combining train and test set will help take advantage of any
# leakage in test dataset. A common trick in Kaggle Competitions.

train_len = len(df_train)
df_combined =  pd.concat(objs=[df_train, df_test], axis=0).reset_index(drop=True)

print(df_combined.shape)


docs_combined = df_combined['comment_text'].astype(str)

# initialize the tokenizer. Given a character sequence and a defined document unit, tokenization is the task of chopping it up into pieces, called tokens , perhaps at the same time throwing away certain characters, such as punctuation.
```

```
t = Tokenizer(num_words=200000)
t.fit_on_texts(docs_combined)
vocab_size = len(t.word_index) + 1

# integer encode the text data
encoded_docs = t.texts_to_sequences(docs_combined)

# pad the vectors to create uniform length
padded_docs_combined = pad_sequences(encoded_docs, maxlen=150, padding='post')

# separate the train and test sets

df_train_padded = padded_docs_combined[:train_len]
df_test_padded = padded_docs_combined[train_len:]

print(df_train_padded.shape)
print(df_test_padded.shape)

# load the glove840B embedding into memory after downloading and unzippping it. You can
download glove dataset from here https://nlp.stanford.edu/projects/glove/

embeddings_index = dict()
f = open('/home/nbuser/glove/ glove.840B.300d.txt', encoding="utf8")

for line in f:
   # Note: use split(' ') instead of split() if you get an error. First index is the token(word) name, and
the rest of the token are the embedding values.
        values = line.split(' ')
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs
f.close()
print('Loaded %s word vectors.' % len(embeddings_index))

# create a weight matrix
embedding_matrix = np.zeros((vocab_size, 300))
for word, i in t.word_index.items():
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
                embedding_matrix[i] = embedding_vector

#Define X and y
X = df_train_padded
X_test = df_test_padded
list_classes = ["toxic", "severe_toxic", "obscene", "threat", "insult", "identity_hate"]
y = df_train[list_classes].values

# Splitting dataset into training data and test data.
X_train, X_eval, y_train ,y_eval = train_test_split(X, y,test_size=0.05,shuffle=True)

#MAIN Create LSTM model
```

```
#sequential is a keras format and it states that the layers will be in sequential form.
#model.add is used for adding layers.  Embedding is the embedding layer which contains the
#pretrained glove weights(already trained in other dataset like wikipedia/twitter. We will use the
#features learned by the embedding trained on these dataset for our problem. Many of the semantic
#characteristics are same.
#Dropout(0.2) means that 20% of the neurons are randomly dropped.
# CuDNNLSTM represents LSTM layer accelerated by nvidia CUDA cores. You can replace it by
#LSTM if you don't have a gpu
# maxpooling replaces 2x2 matrix by 1 value, which is the maximum value. This removes the
#unwanted information learned by  LSTM.
# activation in Dense layer represents the activation function used.

model=Sequential()
model.add(Embedding(vocab_size, 300, weights=[embedding_matrix],
                    input_length=150, trainable=False))
model.add(Dropout(0.2))
model.add((Bidirectional(CuDNNLSTM(50,return_sequences=True))))
model.add(GlobalMaxPool1D())
model.add(Dense(70, activation="relu"))
model.add(Dropout(0.3))
model.add(Dense(6, activation="sigmoid"))


# This provides a summary of the neural network
model.summary()

# compile the model
# adam is the optimization function used to train the neural network.
# EarlyStopping stops the training of the neural network when the validation loss is not reducing for
#a few iteration and it is clear that the model is overfitting.
# model.fit trains the neural network and calculates the validation accuracy on the validation set.

Adam_opt = Adam(lr=0.0003, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.000015)
model.compile(optimizer=Adam_opt, loss='binary_crossentropy', metrics=['acc'])

early_stopping = EarlyStopping(monitor='val_loss', patience=4, mode='min',min_delta=0.0005)
save_best = ModelCheckpoint('/home/nbuser/toxiclstmspatial.hdf', save_best_only=True,
                            monitor='val_acc', mode='max')

history = model.fit(X_train, y_train, validation_data=(X_eval, y_eval),
                    epochs=40, verbose=1,callbacks=[early_stopping,save_best],batch_size=128)
```

# CONCLUSION

We  have  Seen a basic introduction of  Deep Learning for NLP. There are lot of different models that we have not covered in this article like GRU,attention models, recursive neural network,image captioning etc. This could be something we can cover in further articles.

We have  not done a survey of the startups in NLP space that are using this technology. This could also be the subject of a future article.

# REFERENCES


https://deeplearning4j.org/bagofwords-tf-idf
http://ruder.io/word-embeddings-1/index.html#word2vec
https://machinelearningmastery.com/what-are-word-embeddings/
http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/
https://www.youtube.com/watch?v=9zhrxE5PQgY
http://cs224d.stanford.edu/syllabus.html
http://videolectures.net/deeplearning2016_bengio_neural_networks/?q=recurrent%20neural
https://nlp.stanford.edu/IR-book/html/htmledition/tokenization-1.html
https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/


# ABOUT THE AUTHOR


My name is Balaji Balasubramanian. I am a Third Year Engineering student in Vidyalankar Institute of Technology, Mumbai,India. I am passionate about Machine Learning and Deep Learning and I have been working in this field for 3 years. My email address is balajib26@gmail.com . Contact me if you have any queries about this article or you want to collaborate with me on some project.