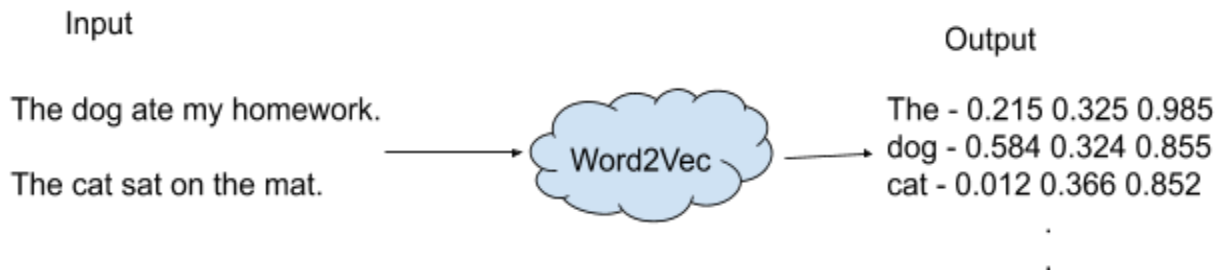
Introduction to Word2Vec

In the previous article, we have discussed what word embeddings are and how to train them using neural networks. This blog gives an introduction to one of the popular word embeddings model Word2Vec created by Google in 2013. It is a combination of deep learning models to compute continuous dense vector representations of words as shown in the diagram below:



When given a text corpus, these unsupervised Word2Vec models first creates a vocabulary of possible words and generates dense word embeddings for each word in the vocabulary which represents the corresponding vectors. This results in very low dimensions of the embedding layer compared to high dimensional sparse Bag of Words models.

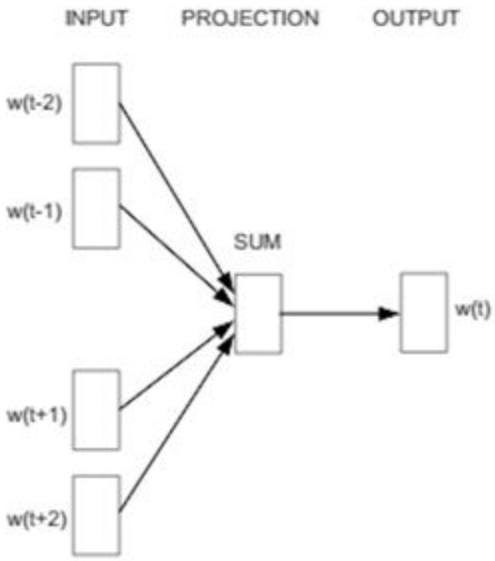
There are two different versions of Word2Vec for creating word embedding representations. These include,

- Continuous Bag of Words Model
- Skip-gram Model

Continuous Bag of Words (CBOW) Model:

Given a corpus the CBOW model loops on all the sentences selecting contexts(surrounding words) to predict current target word. To lower the number of words in the context, a parameter called **window size** is used. Now, consider a sample sentence "An idle brain is the devil's workshop" taking window size of 2, pairs of (context words, target words) can be like **([idle, is], brain), ([An, brain], idle), ([the,**

workshop], devil's) and so on. Inshort the CBOW model predicts target word based on context window words.

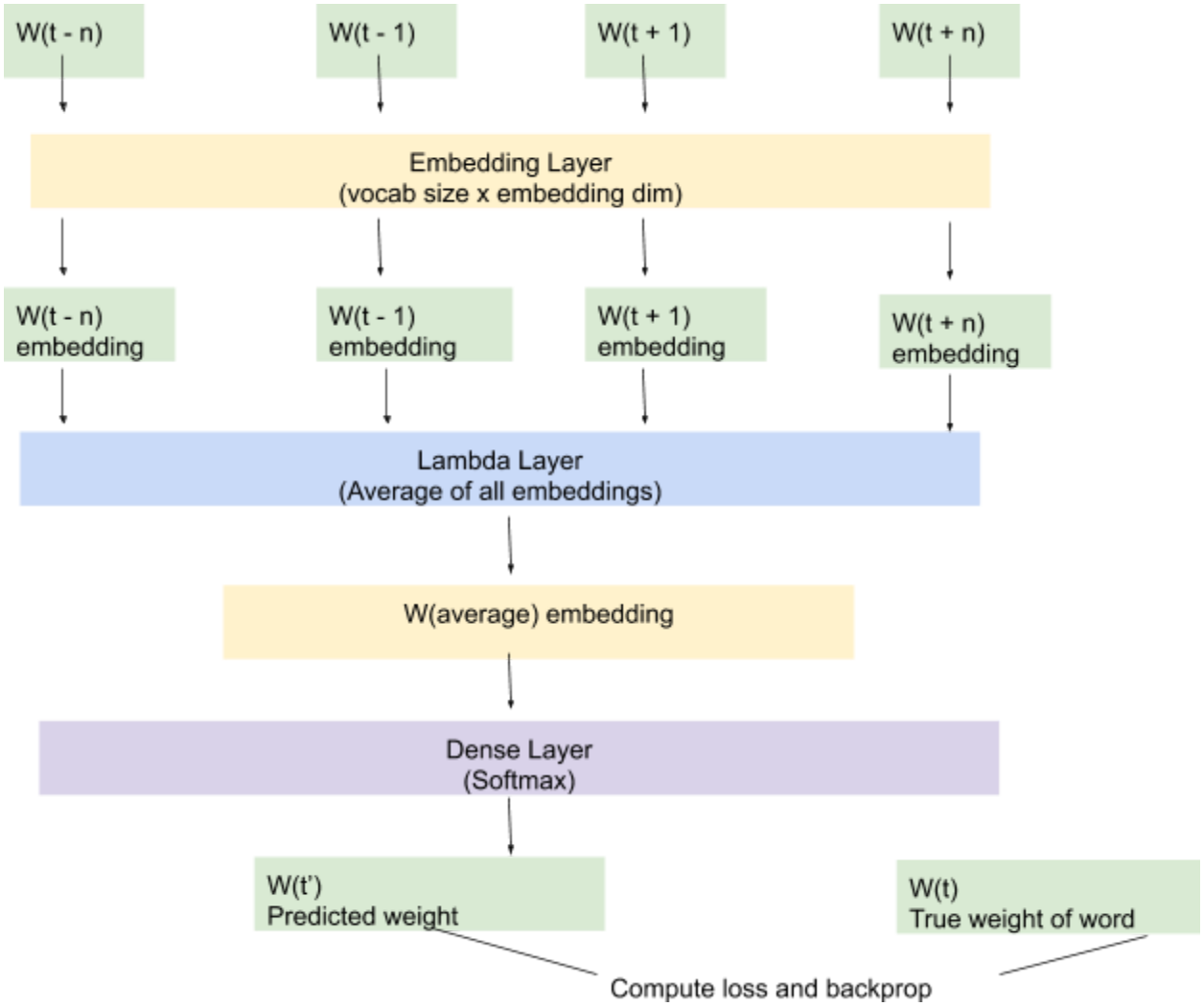


The CBOW model architecture

To summarize the above model architecture, first we take the surrounding context words of specific window length. We get input context words of dimensions **(2 x window size)**, we will pass them to an embedding layer of dimension **(vocab size x embedding dim)** which gives us a dense embedding vector for each word **(1 x embedding dim)**. Next, we average out all these embeddings using a lambda layer to get an average dense embedding **(1 x embedding dim)**.

Finally the dense softmax layer is used to predict the target word. It is compared with the true target word and loss is computed and the parameters or weights of embedding layers are updated using the regular back propagation. This process is repeated for all (context words, target word) for multiple epochs to obtain a trained word embeddings.

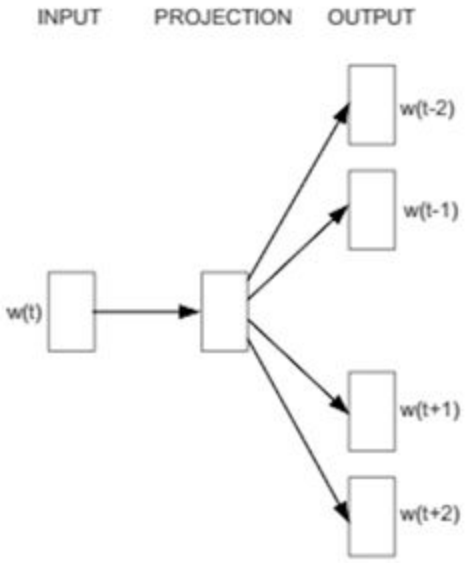
The figure below depicts the same. Next, we discuss the skip-gram model.



Visual outline of CBOW model

Skip - gram Model:

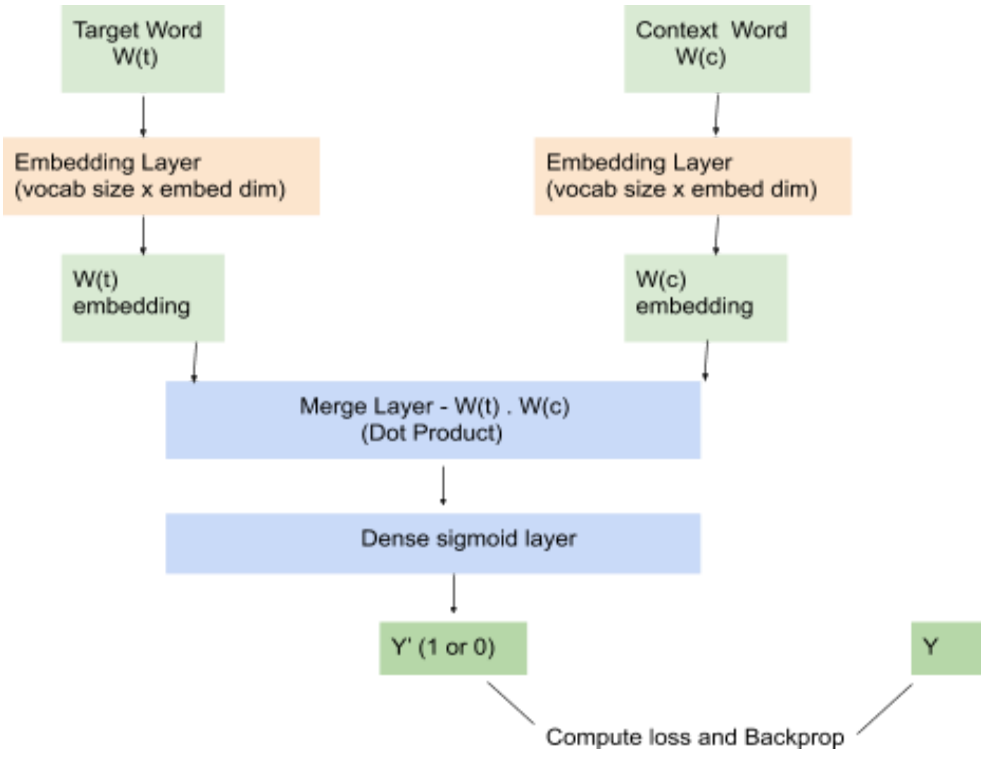
The skip-gram model is exactly the reverse of the Continuous Bag of Words (CBOW) model. Here, we predict the context words when given a target word. Consider the following **(context window, target words)** in the sentence "Film industry looks good from the distance". **([looks, from], good)**, **([Film, looks], industry)....etc.** Therefore, the task is to predict the context window **[looks, from]** given the target word **'good'**. The skip-gram model produces more accurate results for large datasets. The model architecture is shown below.



Skip-gram model architecture:

When given an unsupervised corpus of text data, the skip-gram model converts it into a supervised task of predicting (context words) based on target word. To explain model architecture further, it takes inputs of both context window and target word and passes to the embedding layer **(vocab_size, embedding dim)**.

Next, these two independent layers are **merged together** with a dot product layer. The computed value is passed to a Dense **sigmoid** layer to get a binary output **1 or 0**. The predicted output is compared with a true output "1" if the given input pairs form (context words, target word) else 0. The figure below explains the same.



Visual outline of skip-gram model

Enough of theory, let's look at implementation of Wrd2Vec model trained on google news with 300 embed dim.

First download and load the model, then we can find most similar words, find similar pair words and finally plot these word embeddings.

```
#Import and load word2vec-300 model
import numpy as np
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import gensim.downloader as api

word2vec_model = api.load('word2vec-google-news-300')


word2vec_model["computer"].shape #Getting 300 dim vectors for the word


Output: (300,)
```

Finding similar words:

It computes closet vectors to the word "boy".

```
word2vec_model.most_similar("boy")
Output:
[('girl', 0.8543272018432617),
 ('teenager', 0.7606689929962158),
 ('toddler', 0.7043969631195068),
 ('teenage_girl', 0.6851483583450317),
 ('man', 0.6824870109558105),
 ('teen_ager', 0.6499968767166138),
 ('son', 0.6337764263153076),
 ('kid', 0.63228440284729),
 ('youngster', 0.6183817386627197),
 ('stepfather', 0.5989422798156738)]
```

Finding similar Pair words:

To get what, if **girl** corresponds to **boy**, then **queen** corresponds to ____. It performs the following calculation on vectors and returns the closest vector.

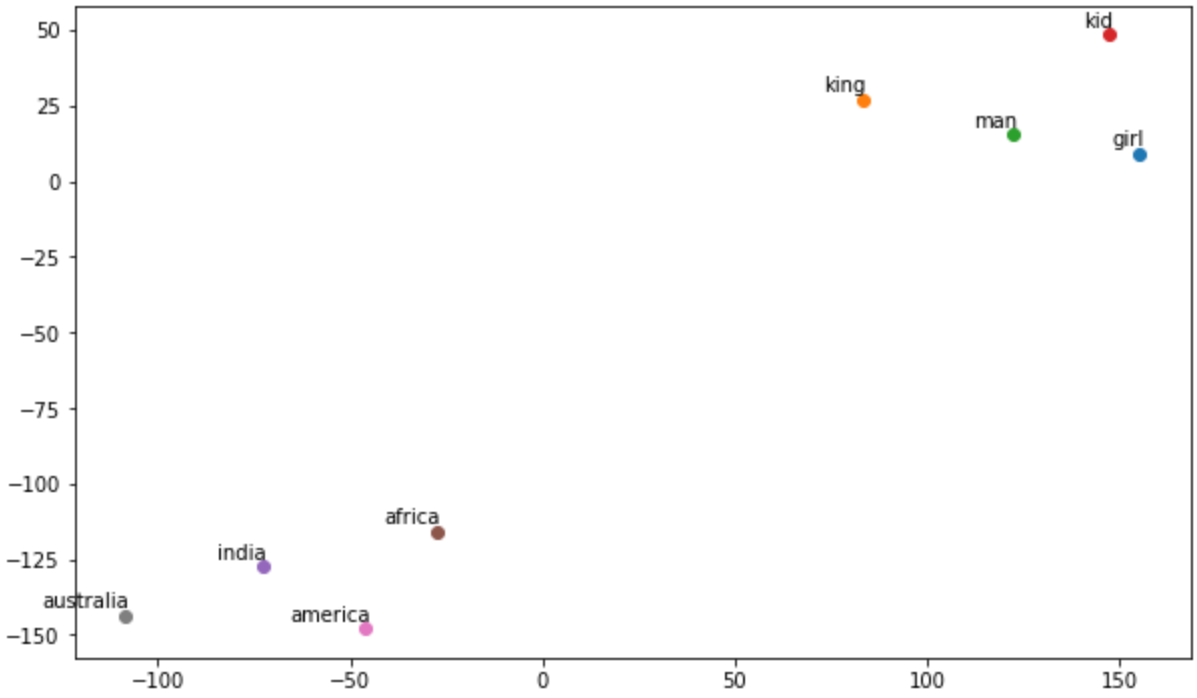
Target_word = boy + queen - girl

```
word2vec_model.most_similar(positive=['boy','queen'], negative = ['girl'],
topn=1)

Output:
[('king', 0.7298422455787659)]
```

Plotting 300 dim word embedding in a 2D chart:

Here, using sklearn's manifold model we translate 300 dimension word embedding vectors to 2 dimensions using Principal component analysis and plot these words on a scatter plot.

```
vocab = ['girl','king','man','kid', 'india','africa','america','australia']

labels = []
wordvectors = []

#Get word vectors for words in vocab
for word in vocab:
  labels.append(word)
  wordvectors.append(word2vec_model[word])

#Converts 300 dim vectors to 2 dim using PCA
tsne_model = TSNE(perplexity=3, n_components=2, init='pca')
coordinates = tsne_model.fit_transform(wordvectors)

x = []
y = []
for value in coordinates:
  x.append(value[0])
  y.append(value[1])

#Plot
plt.figure(figsize=(10,6))
for i in range(len(x)):
  plt.scatter(x[i],y[i])
  plt.annotate(labels[i], xy=(x[i], y[i]), xytext=(2, 2),
               textcoords='offset points', ha='right', va='bottom')

plt.show()
```



Conclusion:

We have learned two Word2Vec models where word embeddings are trained from a corpus of text. Try to implement these models from scratch, few reference articles are provided to help you with the task.

Next,we shall see another popular approach of training word embeddings called GloVe also we'll look into the usage of transfer learning for NLP tasks.

References:

- https://www.kdnuggets.com/2018/04/implementing-deep-learning-methods-feature-engineering-text-data-cbow.html
- https://www.kdnuggets.com/2018/04/implementing-deep-learning-methods-feature-engineering-text-data-skip-gram.html