Deep Learning for Sentiment Analysis using Stock Market News

Jacob Weiss

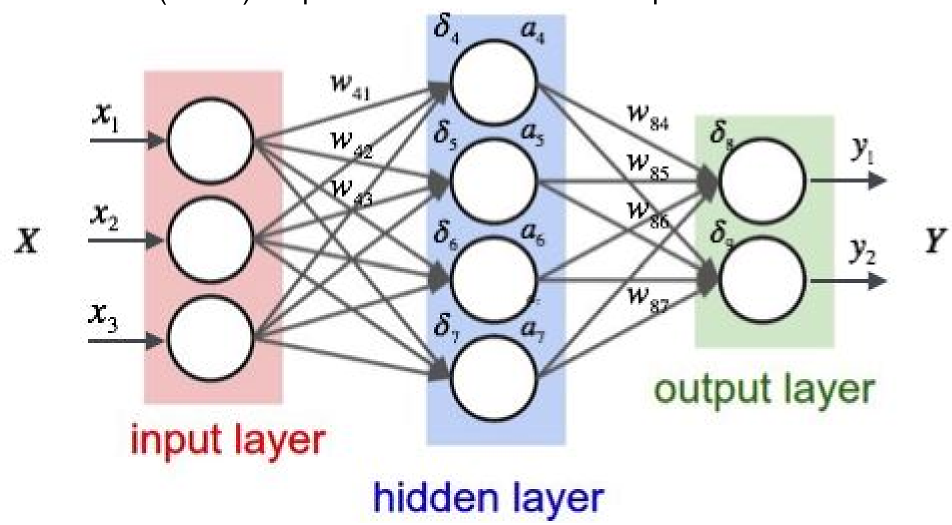
Johns Hopkins University

Introduction

With the shear amount of financial information its quite impossible for investors to stay up to speed with the market. Thanks to the cutting edge techniques in mathematics and computer science we can use machines to help us consume this information. This paper will cover one of those techniques, deep learning. Specifically, the research is focused on classifying sentiment from news articles using a 3-layer densely connected neural network.

Basics of a Neural Network

There are a multitude of neural networks in the world today. This paper will soley focus on the feed forward neural network (FFNN). Depicted below shows the components of a FFNN:

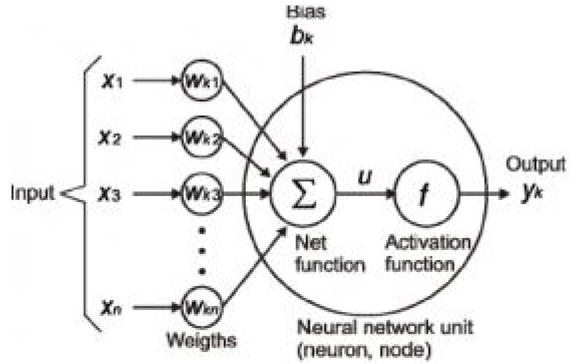


The input layer and weights can be represented as a vector:

$$x = \begin{bmatrix} x_1 & \dots & x_n \end{bmatrix}^T$$

$$w = \begin{bmatrix} w_1 & \dots & w_n \end{bmatrix}$$

Before the input layer gets passed to the hidden layer we need to assign it a weight. Think of the hidden layer as a vector of nodes, where each node is the weighted sum. Each weighted sum (or node) will be passed into an *activation* function. Activation functions are a way to represent the mapping as a non-linear (or linear) representation. Some common activation functions are sigmoid, tanh, or ReLU. Not to over complicate the hidden layer but we can also assign a bias variable to each node as well. To summarize, the hidden layer can be represented as:



$$H = f(x^T w) + b$$
$$b = \begin{bmatrix} b_1 & \dots & b_n \end{bmatrix}$$

We repeat the process from the hidden layer to the output layer. The output layer can be represented as 1 node which is referred to as a regressor, or multiple nodes which is referred to as a classifier

Deep Learning

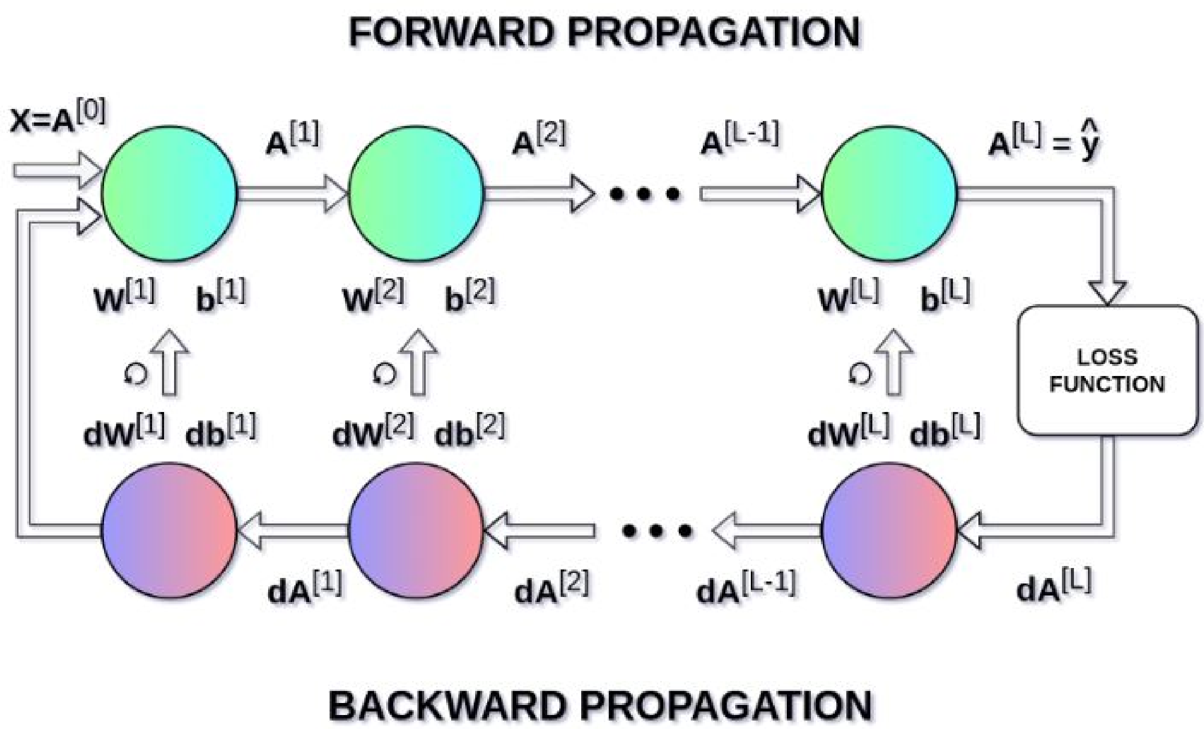
The true power of a neural network is how we train the model to learn the right parameters to predict the best output layer. To measure the learning process we use a loss function. The loss function is designed to show how far the model is from our ideal solution.

$$J(W, b) = 1/m \sum_{i=1}^{m} L(\hat{y_i}, y_i)$$

Our goal is to decrease the loss function (close the gap between actual and predicited solutions) and increase accuracy. To do this we use a technique called gradient descent. This method allows us to calculate the values of the loss function partial derivatives with respect to each the parameters in the model. To calculate the derivatives we will use a tool called Backpropagation. The parameters of the neural network are optimized using the following formula:

$$H = H - \alpha \partial H$$
$$b = b - \alpha \partial b$$

Alpha represents the learning rate with respect to our hidden layer ***H*** and bias ***b***. Choosing the best learning rate can be meticulous. If we set it too low our model will learn slowly verse setting it too high the model will not be able to hit the minimum. The derivatives are calculate using the chain rule with respect to ***H*** and **b**. Here is a clear representation of how the propagation will work:



Stock Market News Data

The news collected for this model was derived from Stock News Api (stocknewsapi.com). They collect news stories from The Street, CNBC, Zacks, Benzinga, and much more. The stocks used in this analysis were randomly selected from the Nasdaq stock exchange.

In [85]:
```
import pandas as pd
os.chdir("E:\datasets\Ticker_History")
news = pd.read_excel("newsData.xlsx")
news.head()
```

Out[85]:

	date	text	sentiment	ticker
0	Thu, 11 Apr 2019 09:30:09 -0400	CACI vs. PDFS: Which Stock Is the Better Value...	Neutral	AAME
1	Tue, 16 Apr 2019 10:40:31 -0400	Shares of Adobe are racing back to all-time hi...	Positive	AAOI
2	Sat, 13 Apr 2019 09:30:13 -0400	Adobe (ADBE) reported earnings 30 days ago. Wh...	Positive	AAOI
3	Mon, 08 Apr 2019 16:51:27 -0400	Adobe boasts a strong fundamental business mod...	Positive	AAOI
4	Wed, 03 Apr 2019 08:22:58 -0400	But can the two underdogs catch up to the mark...	Positive	AAOI

The date is the date of the news announcement. The text column is the text from the news article. The sentiment column is the sentiment labelled by Stock News API. As a reminder the type of deep learning we are doing is called supervised learning. This means that for every input we have a labelled output. The model will learn how to classify the sentiment using the pre-labeled dataset.

In [86]:
```
news.describe()
```
Out[86]:

	date	text	sentiment	ticker
count	3071	3071	3071	3071
unique	2121	2535	3	501
top	Wed, 08 Aug 2018 20:00:00 -0400	The "Halftime Report" traders give their top s...	Positive	ANDA
freq	17	38	1506	58

Data Preparation

The packages used in the model are labelled below. The deep learning package is Keras.

In [87]:
```
import numpy as np #data prep
import re #data prep
import collections #data prep
import matplotlib.pyplot as plt #plotting our results

from sklearn.model_selection import train_test_split # split the dataset into
import nltk #this packages will be used to remove stop words from the dataset

from keras.preprocessing.text import Tokenizer #Tokenize our dataset
from keras.utils.np_utils import to_categorical #used for one hot encoding
from sklearn.preprocessing import LabelEncoder #used for one hot encoding
from keras import models #deep learning model
from keras import layers #layers for the model
from keras import regularizers
```

Here are descriptions of the functions used:

- one_hot_seq, Converting integers into one-hot encoded features
- remove_stopwords, In natural language processing we want to remove stop words from our analysis. Stop words are redudant words that provide no value in prediction.
- deep_model, the deep learning model that will fit the training data and validated using the validation dataset
- eval_metric, evaluate model performace
- compare_loss_with_baseline, compare new models with our baseline model
- test_model, testing the models for accuracy

In [56]:

```
def one_hot_seq(seqs, nb_features = NB_WORDS):
    ohs = np.zeros((len(seqs), nb_features))
    for i, s in enumerate(seqs):
        ohs[i, s] = 1.
    return ohs

nltk.download('stopwords')
def remove_stopwords(input_text):
        stopwords_list = stopwords.words('english')
        whitelist = ["n't", "not", "no"]
        words = input_text.split()
        clean_words = [word for word in words if (word not in stopwords_list
        return " ".join(clean_words)

def deep_model(model):
    model.compile(optimizer='rmsprop'
                  , loss='categorical_crossentropy'
                  , metrics=['accuracy'])
    history = model.fit(X_train_rest
                       , y_train_rest
                       , epochs=NB_START_EPOCHS
                       , batch_size=BATCH_SIZE
                       , validation_data=(X_valid, y_valid)
                       , verbose=0)

    return history

def eval_metric(history, metric_name):
    metric = history.history[metric_name]
    val_metric = history.history['val_' + metric_name]
    e = range(1, NB_START_EPOCHS + 1)
    plt.plot(e, metric, 'bo', label='Train ' + metric_name)
    plt.plot(e, val_metric, 'b', label='Validation ' + metric_name)
    plt.legend()
    plt.show()

def compare_loss_with_baseline(h, model_name):
    loss_base_model = base_history.history['val_loss']
    loss_model = h.history['val_loss']
    e = range(1, NB_START_EPOCHS + 1)
    plt.plot(e, loss_base_model, 'bo', label='Validation Loss Baseline Model'
    plt.plot(e, loss_model, 'b', label='Validation Loss ' + model_name)
    plt.legend()
    plt.show()

def test_model(model, epoch_stop):
    model.fit(X_train_oh
              , y_train_oh
              , epochs=epoch_stop
              , batch_size=BATCH_SIZE
              , verbose=0)
    results = model.evaluate(X_test_oh, y_test_oh)

    return results
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\jawei\AppData\Roaming\nltk_data...
```

```
[nltk_data]   Package stopwords is already up-to-date!
```

The hyperparameters were selected at random. In a full deep learning analysis we would use an optimizer to find the best hyperparameters.

In [88]:
```
NB_WORDS = 10000  #Number of words in dictionary
VAL_SIZE = 1000  #Validation size
NB_START_EPOCHS = 20  #Number of epochs to train the model
BATCH_SIZE = 512  #Step size for gradient descent
```

To evaluate the model performance we will need to measure the model on a seperate test data set. Using Sci-kit learn we can split the data set into training and test set. The test set will be 30% of the overall dataframe. As such, we can estimate how well the model generalizes.

In [89]:
```
X_train, X_test, y_train, y_test = train_test_split(df.text, df.sentiment, te
print('Train size:', X_train.shape[0])
print('Test size:', X_test.shape[0])
```

```
Train size: 2149
Test size: 922
```

Next we will need to convert the text into numbers. Tokenizer is the standard method for converting text to numbers. We are keeping the most frequent words in the training set.

In [91]:
```
tk = Tokenizer(num_words=NB_WORDS,
               filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n',
               lower=True,
               split=" ")
tk.fit_on_texts(X_train)
```

Now that the dictionary is built we will convert the text into a list of indexes from the dictionary.

In [94]:
```
X_train_seq = tk.texts_to_sequences(X_train)
X_test_seq = tk.texts_to_sequences(X_test)
print('"{}" is converted into {}'.format(X_train[0], X_train_seq[0]))
```

```
"CACI vs. PDFS: Which Stock Is Better Value Option?" is converted into [25
7, 1019, 3095, 1228, 3096, 1229, 1509, 3097, 613, 614, 1020, 507, 1229, 13
1, 3098, 50]
```

Typically we would have used Word Embeddings to represent the list. Word embeddings are a class of techniques where individual words are represented by a vector. Each word is mapped to one specific vector and the vector values are learned by the neural network. In this analysis, I kept to the basics of one-hot encoding. The issue with one-hot encoding is that we end up with sparse vectors of high dimensionality. Despite the performance issues, one-hot encoding exposes us to semantic issues with words. For example stock and equity would be classified as two different objects when in reality they are the same

In [96]:
```
X_train_oh = one_hot_seq(X_train_seq)
X_test_oh = one_hot_seq(X_test_seq)
le = LabelEncoder()
y_train_le = le.fit_transform(y_train)
y_test_le = le.transform(y_test)
y_train_oh = to_categorical(y_train_le)
y_test_oh = to_categorical(y_test_le)
X_train_rest, X_valid, y_train_rest, y_valid = train_test_split(X_train_oh, y
```

Model Development

Here is the baseline model. We have 64 nodes (randomly choosen) and 3 dense layers. RelU is the activation function used which is standard for most neural networks. The last activation is softmax which squishes our 2nd hidden layer into the final output.

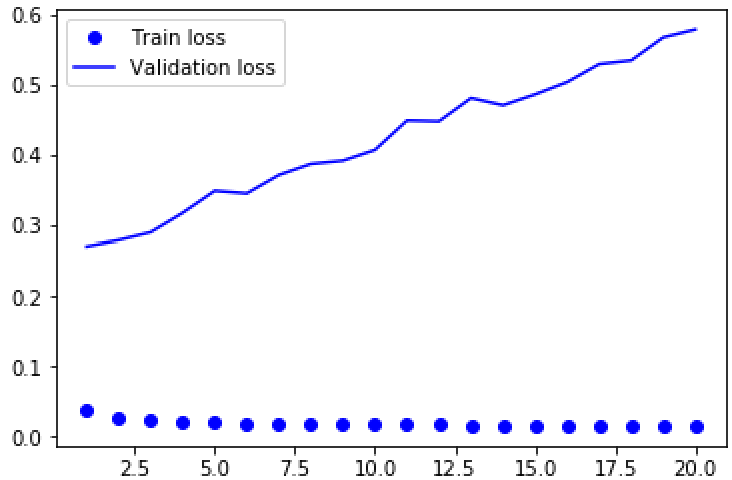
In [51]:
```
base_model = models.Sequential()
base_model.add(layers.Dense(64, activation='relu', input_shape=(NB_WORDS,)))
base_model.add(layers.Dense(64, activation='relu'))
base_model.add(layers.Dense(3, activation='softmax'))
base_model.summary()
```

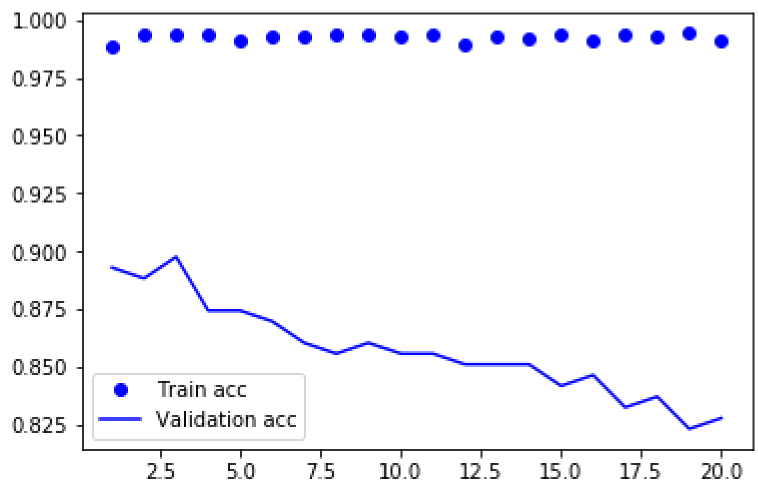
```
WARNING:tensorflow:From C:\Users\jawei\Anaconda3\envs\tensorflow\lib\site-p
ackages\tensorflow\python\framework\op_def_library.py:263: colocate_with (f
rom tensorflow.python.framework.ops) is deprecated and will be removed in a
future version.
Instructions for updating:
Colocations handled automatically by placer.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 64)                640064
_________________________________________________________________
dense_2 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_3 (Dense)              (None, 3)                 195
=================================================================
Total params: 644,419
Trainable params: 644,419
Non-trainable params: 0
_________________________________________________________________
```

In [98]:
```
base_history = deep_model(base_model)
```

The validation loss starts to increase as from epoch 5. The training loss continues to lower, which is normal because the model is fitting the training data. Similiar to the validation loss, the validation accuracy starts to peak around epoch 5.

In [99]:
```
eval_metric(base_history, 'loss')
eval_metric(base_history, 'acc')
```





To address overfitting we will try a few options:

- reduce the size of the network by trimming layers and nodes
- Add a cost to the loss function for large weights (regularization)
- Add dropout layers, which randomly kicks out certain features.

In [59]:

```
reduced_model = models.Sequential()
reduced_model.add(layers.Dense(32, activation='relu', input_shape=(NB_WORDS,)
reduced_model.add(layers.Dense(3, activation='softmax'))
reduced_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_4 (Dense)              (None, 32)                320032    
_________________________________________________________________
dense_5 (Dense)              (None, 3)                 99        
=================================================================
Total params: 320,131
Trainable params: 320,131
Non-trainable params: 0
_________________________________________________________________
```

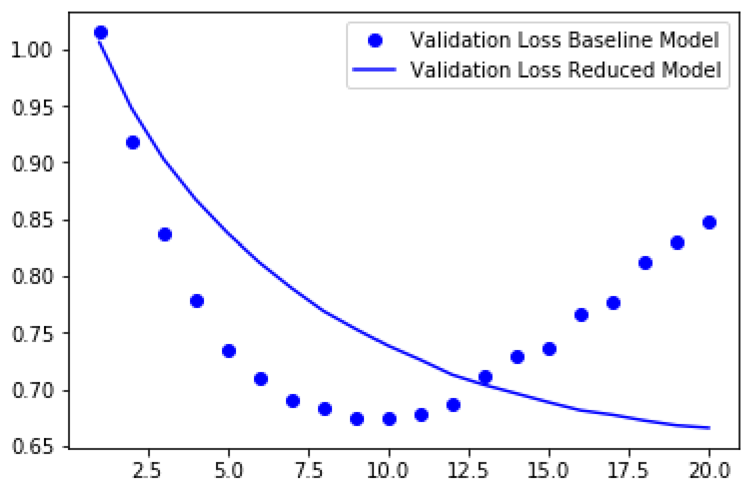
In [60]:

```
reduced_history = deep_model(reduced_model)
```

It takes much longer before the reduced model starts overfitting (around 12.5 epochs).

In [61]:

```
compare_loss_with_baseline(reduced_history, 'Reduced Model')
```



In [62]:

```
reg_model = models.Sequential()
reg_model.add(layers.Dense(64, kernel_regularizer=regularizers.l2(0.001), act
reg_model.add(layers.Dense(64, kernel_regularizer=regularizers.l2(0.001), act
reg_model.add(layers.Dense(3, activation='softmax'))
reg_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 64)                640064
_________________________________________________________________
dense_7 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_8 (Dense)              (None, 3)                 195
=================================================================
Total params: 644,419
Trainable params: 644,419
Non-trainable params: 0
_________________________________________________________________
```

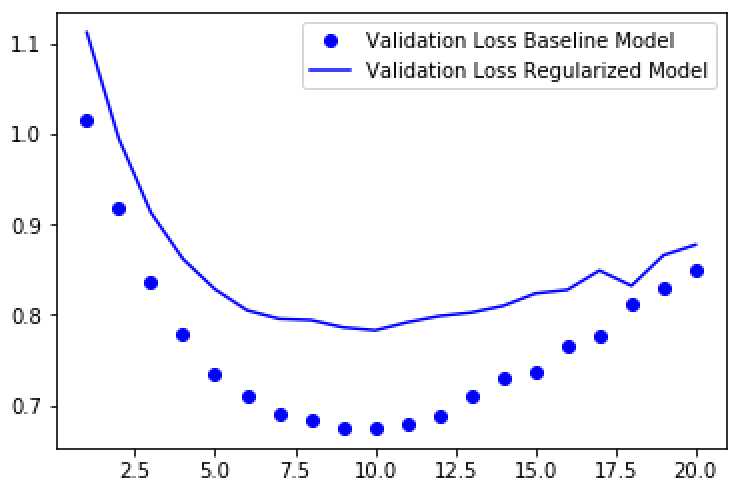
In [63]:

```
reg_history = deep_model(reg_model)
```

For the regularized model we notice that it never overfits the data which is very interesting.

In [64]:

```
compare_loss_with_baseline(reg_history, 'Regularized Model')
```



In [65]:

```
drop_model = models.Sequential()
drop_model.add(layers.Dense(64, activation='relu', input_shape=(NB_WORDS,)))
drop_model.add(layers.Dropout(0.5))
drop_model.add(layers.Dense(64, activation='relu'))
drop_model.add(layers.Dropout(0.5))
drop_model.add(layers.Dense(3, activation='softmax'))
drop_model.summary()
```

```
WARNING:tensorflow:From C:\Users\jawei\Anaconda3\envs\tensorflow\lib\site-p
ackages\keras\backend\tensorflow_backend.py:3445: calling dropout (from ten
sorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed
in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 -
keep_prob`.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_9 (Dense)              (None, 64)                640064
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_10 (Dense)             (None, 64)                4160
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_11 (Dense)             (None, 3)                 195
=================================================================
Total params: 644,419
Trainable params: 644,419
Non-trainable params: 0
_________________________________________________________________
```

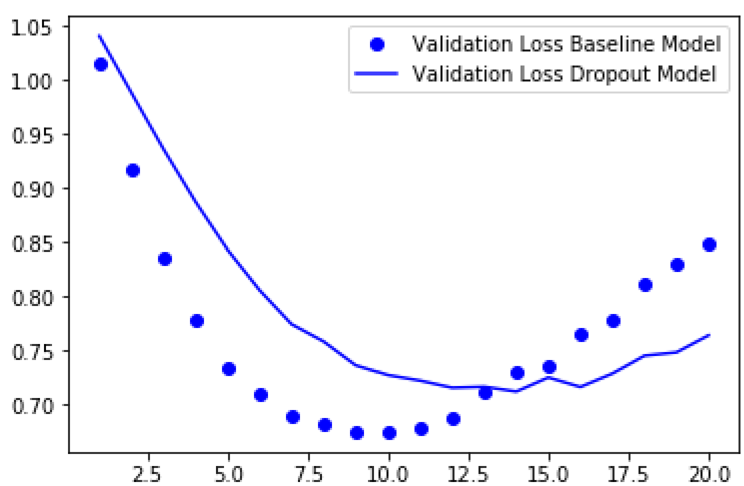
In [66]:

```
drop_history = deep_model(drop_model)
```

The dropout model starts to overfit around 14 epochs.

In [67]:

```
compare_loss_with_baseline(drop_history, 'Dropout Model')
```



Results

By a slim margin, the reduced model performs the best when classifying stock market news (75.6%).

In [68]:
```
base_results = test_model(base_model, 4)
print('/n')
print('Test accuracy of baseline model: {0:.2f}%'.format(base_results[1]*100)
```

```
922/922 [==============================] - 0s 389us/step
/n
Test accuracy of baseline model: 74.95%
```

In [69]:
```
reduced_results = test_model(reduced_model, 10)
print('/n')
print('Test accuracy of reduced model: {0:.2f}%'.format(reduced_results[1]*1(
```

```
922/922 [==============================] - 0s 251us/step
/n
Test accuracy of reduced model: 75.60%
```

In [70]:
```
reg_results = test_model(reg_model, 5)
print('/n')
print('Test accuracy of regularized model: {0:.2f}%'.format(reg_results[1]*1(
```

```
922/922 [==============================] - 0s 359us/step
/n
Test accuracy of regularized model: 73.86%
```

In [71]:
```
drop_results = test_model(drop_model, 6)
print('/n')
print('Test accuracy of dropout model: {0:.2f}%'.format(drop_results[1]*100))
```

```
922/922 [==============================] - 0s 410us/step
/n
Test accuracy of dropout model: 75.27%
```

Conclusion

Deep learning can be used to classify news articles to help investors quickly descipher the sentiment. The mathematics of deep learning allows us to simply vectorize data to perform powerful computation. We also use matrix theory to format words into a vector. Using deep learning we found that our drop out model is the most accurate when classifing market news.

Resources

- https://www.kaggle.com/bertcarremans/deep-learning-for-sentiment-analysis (https://www.kaggle.com/bertcarremans/deep-learning-for-sentiment-analysis)
- https://towardsdatascience.com/https-medium-com-piotr-skalski92-deep-dive-into-deep-networks-math-17660bc376ba (https://towardsdatascience.com/https-medium-com-piotr-skalski92-deep-dive-into-deep-networks-math-17660bc376ba)

- https://medium.com/deep-math-machine-learning-ai/chapter-7-artificial-neural-networks-with-math-bb711169481b (https://medium.com/deep-math-machine-learning-ai/chapter-7-artificial-neural-networks-with-math-bb711169481b)
- https://stocknewsapi.com (https://stocknewsapi.com)