Deep Dive in Recurrent Neural Networks for Binary Classification

By: Joe Klein

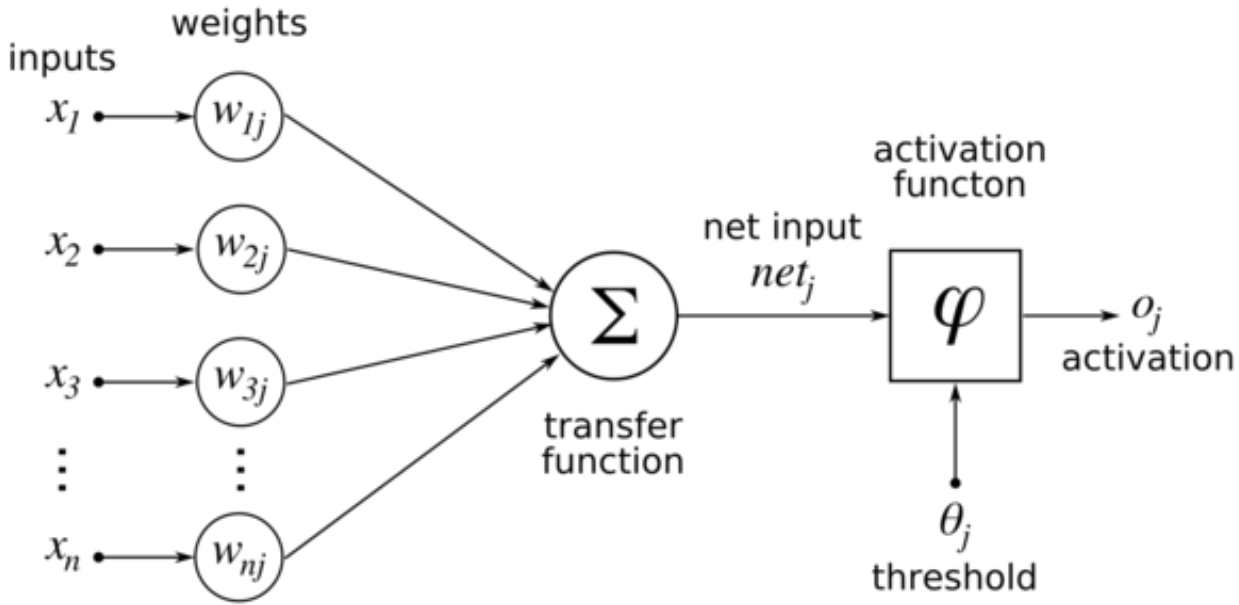
GOAL

- Use Neural Networks to predict the a binary classification
  - Primary dataset: Credit Risk Classification

Neural Networks Intro

Binary Classification

Simple Neural Network



Simple Neural Network in Keras

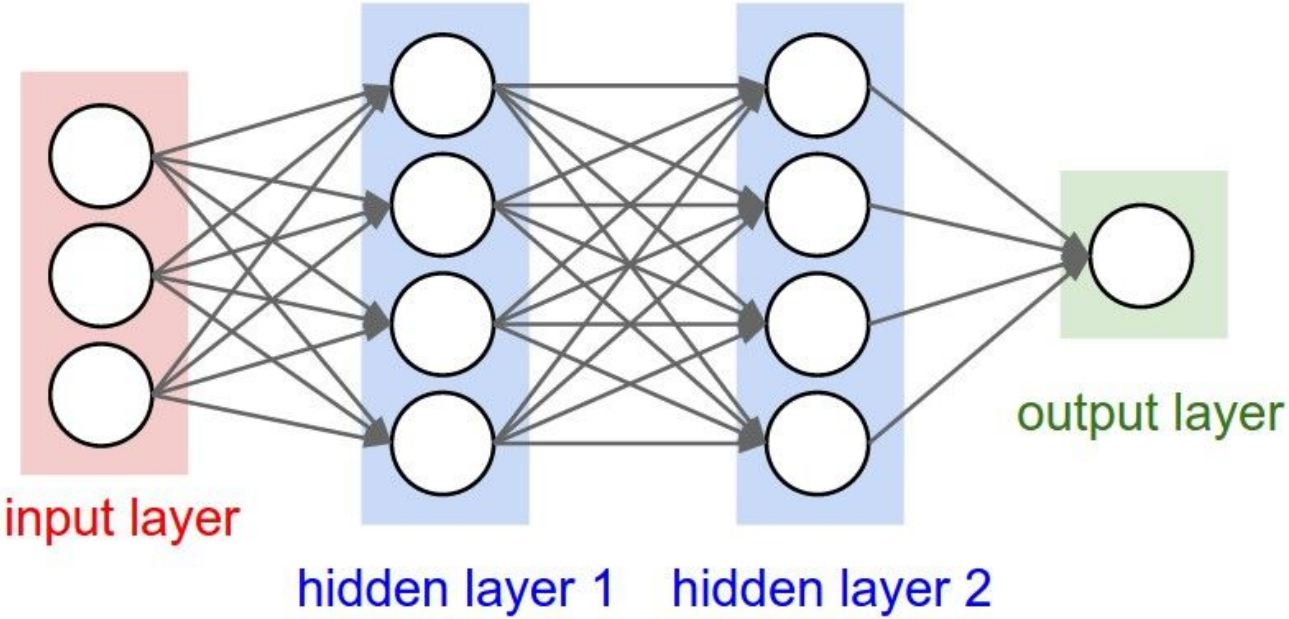
Model:

```
model = Sequential()
model.add(Dense(2, input_dim=input_dim))
model.add(Dense(y_nn_test.shape[1], kernel_initializer='normal', activation='softmax'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Results:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_111 (Dense)            (None, 300)               6300
_________________________________________________________________
dense_112 (Dense)            (None, 300)               90300
_________________________________________________________________
dense_113 (Dense)            (None, 2)                 602
=================================================================
Total params: 97,202
Trainable params: 97,202
Non-trainable params: 0
_________________________________________________________________
None
ROC:  0.719142047754
Continued Avg:  0.723060127492
('Average ROC:', 0.72306012749152138)
```

Multi-Layer Neural Network



Multi-Layer Neural Network in Keras

Model:

```
model = Sequential()
model.add(Dense(300, input_dim=input_dim, kernel_initializer='normal', activation='relu'))

# Build the second layer of your neural network
model.add(Dense(300, kernel_initializer='normal', activation='relu'))
model.add(Dense(y_nn_test.shape[1], kernel_initializer='normal', activation='softmax'))

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

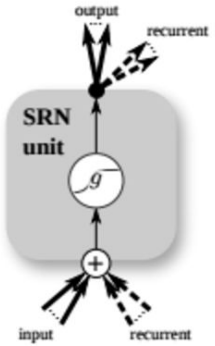
Results:

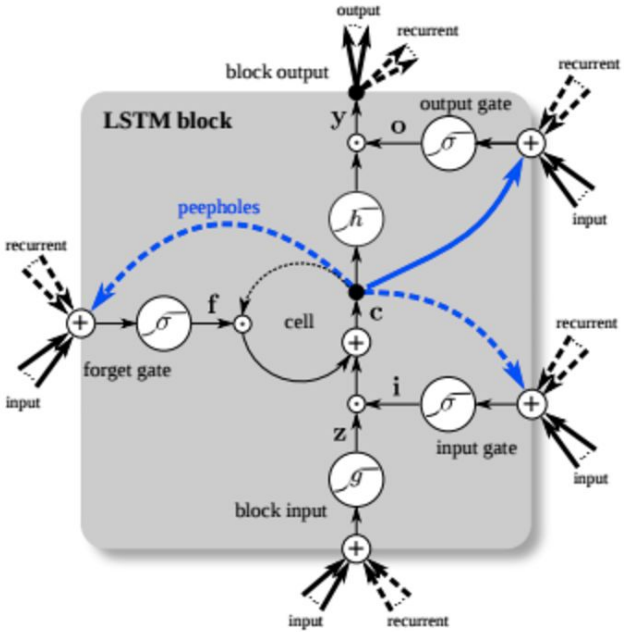
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_111 (Dense)            (None, 300)               6300
_________________________________________________________________
dense_112 (Dense)            (None, 300)               90300
_________________________________________________________________
dense_113 (Dense)            (None, 2)                 602
=================================================================
Total params: 97,202
Trainable params: 97,202
Non-trainable params: 0
_________________________________________________________________
None
ROC:  0.719142047754
Continued Avg:  0.723060127492
('Average ROC:', 0.72306012749152138)
```

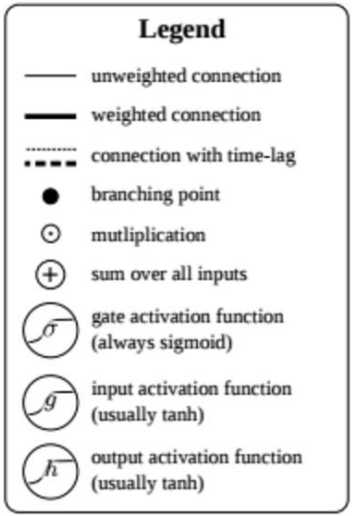
Recurrent Neural Network using LSTM

- In a traditional neural network we assume that all inputs (and outputs) are independent of each other.
- RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations.
- RNNs is that they have a “memory” which captures information about what has been calculated so far.

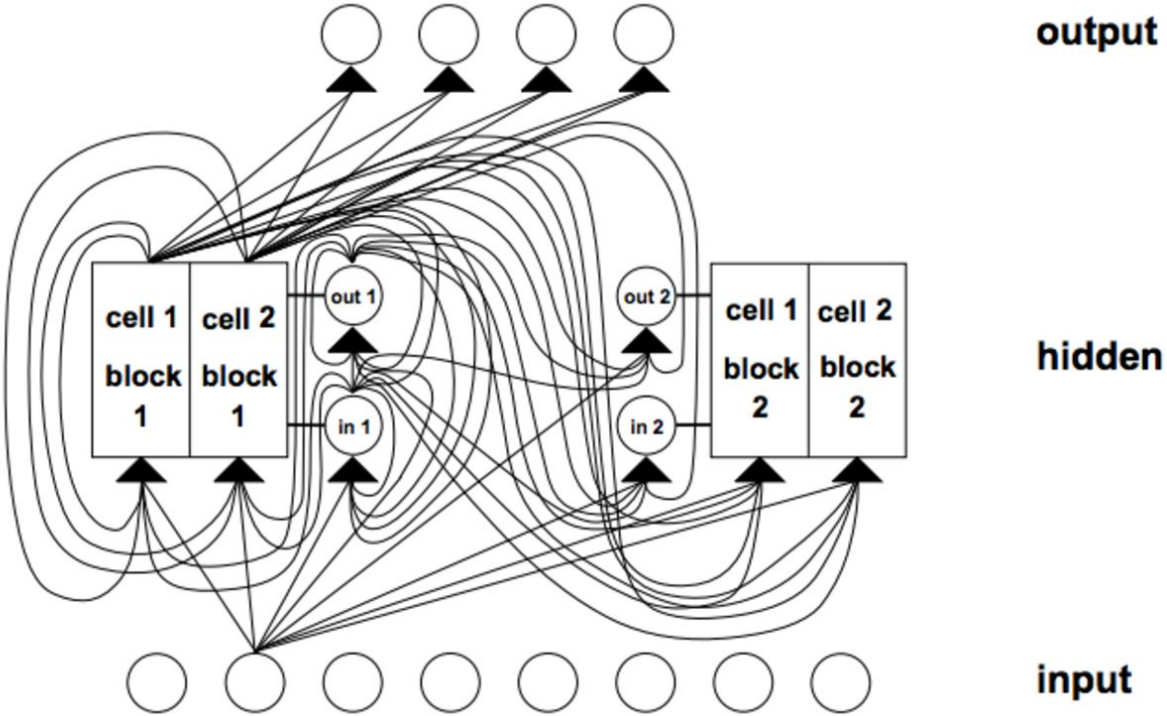
Long Short Term Memory Perceptron



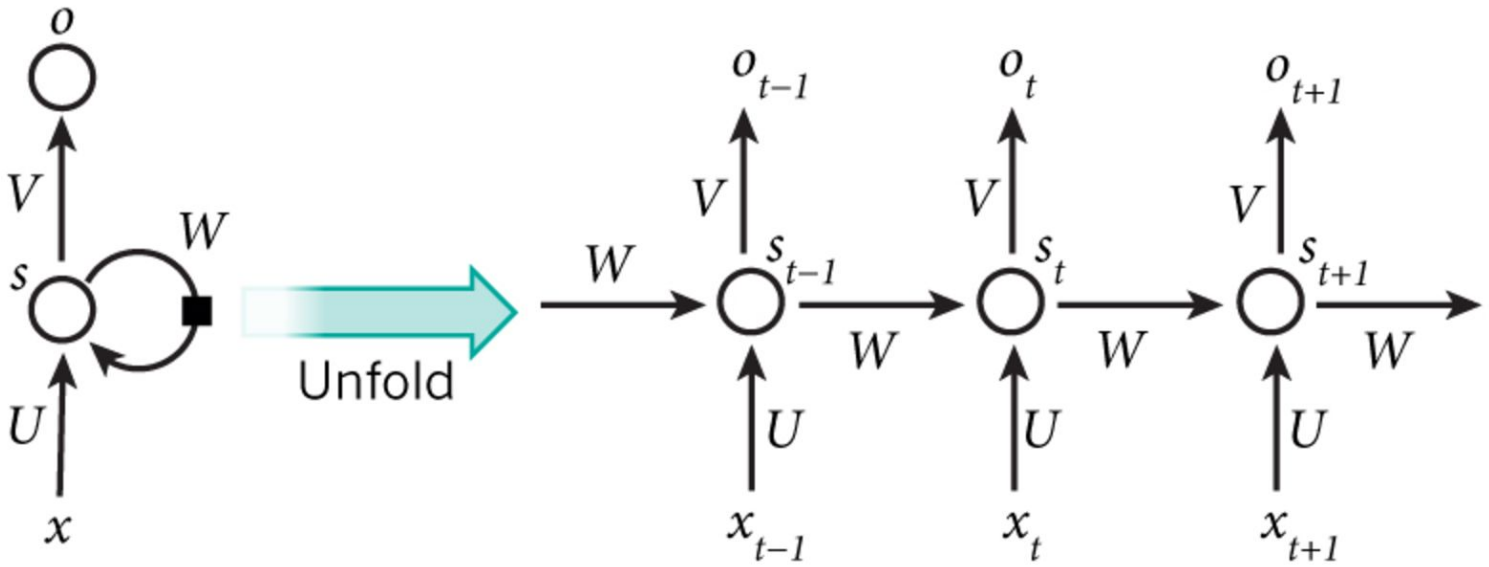




Another LTSM Interpretation



Unfolding Recurrent Neural Networks



Recurrent Neural Network in Keras

Model:

```
model = Sequential()
model.add(LSTM(layer_size1, return_sequences=True, input_shape=(timesteps, data_dim)))
model.add(LeakyReLU(alpha=alpha1))
model.add(LSTM(layer_size2, return_sequences=True))  # returns a sequence of vectors of
model.add(LeakyReLU(alpha=alpha2))
model.add(LSTM(layer_size3))
model.add(LeakyReLU(alpha=alpha1))
model.add(Dense(2, activation='softmax'))

start = time.time()
model.compile(loss='binary_crossentropy', optimizer='nadam', metrics=['accuracy'])
```

Results:

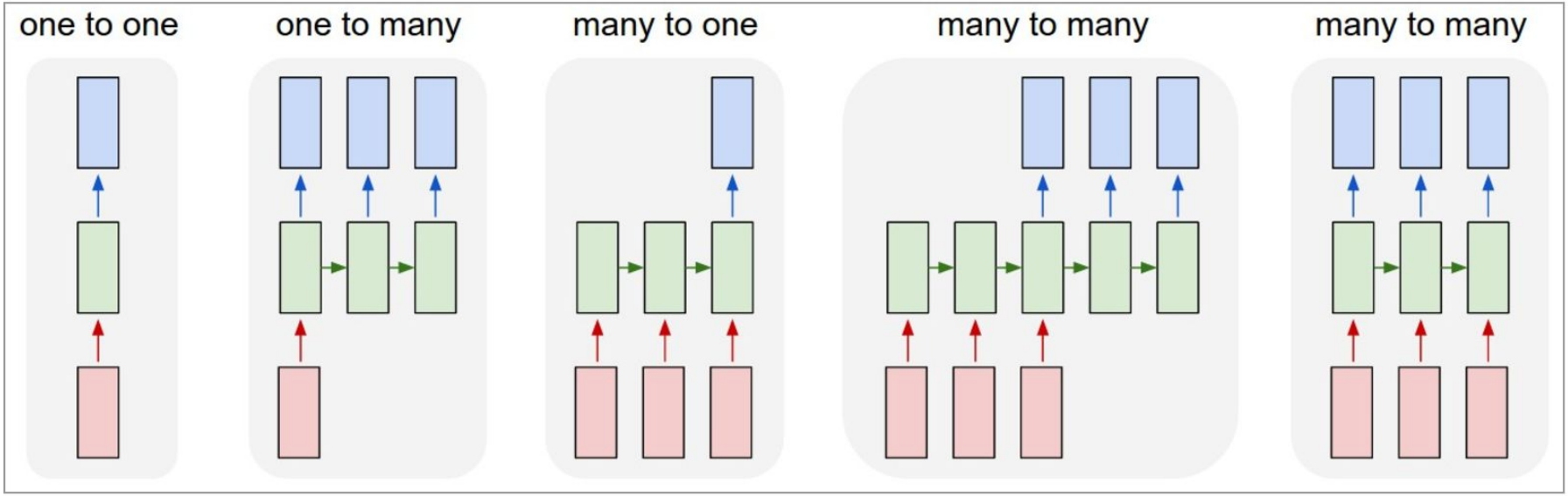
```
Epoch 50/50
500/500 [==============================] - 3s - loss: 0.4711 - acc: 0.7860
0.5668 - val_acc: 0.7240 - val_categorical_accuracy: 0.7240
('Training time : ', 204.2492311000824)
Evaluating model...
480/500 [===========================>..] - ETA: 0s
ROC:  0.705151182286
('Continued Avg: ', 0.66115661240834733)
('Evaluation time : ', 2.7722508907318115)
('Average ROC:', 0.66115661240834733)
480/500 [===========================>..] - ETA: 0s
```

“Why did the recurrent neural network do worse?

They are designed to hold onto more information and learn more.

Grid Searching a Neural Network

Different ways to build a Neural Network



Tuning Parameters of a Neural Network

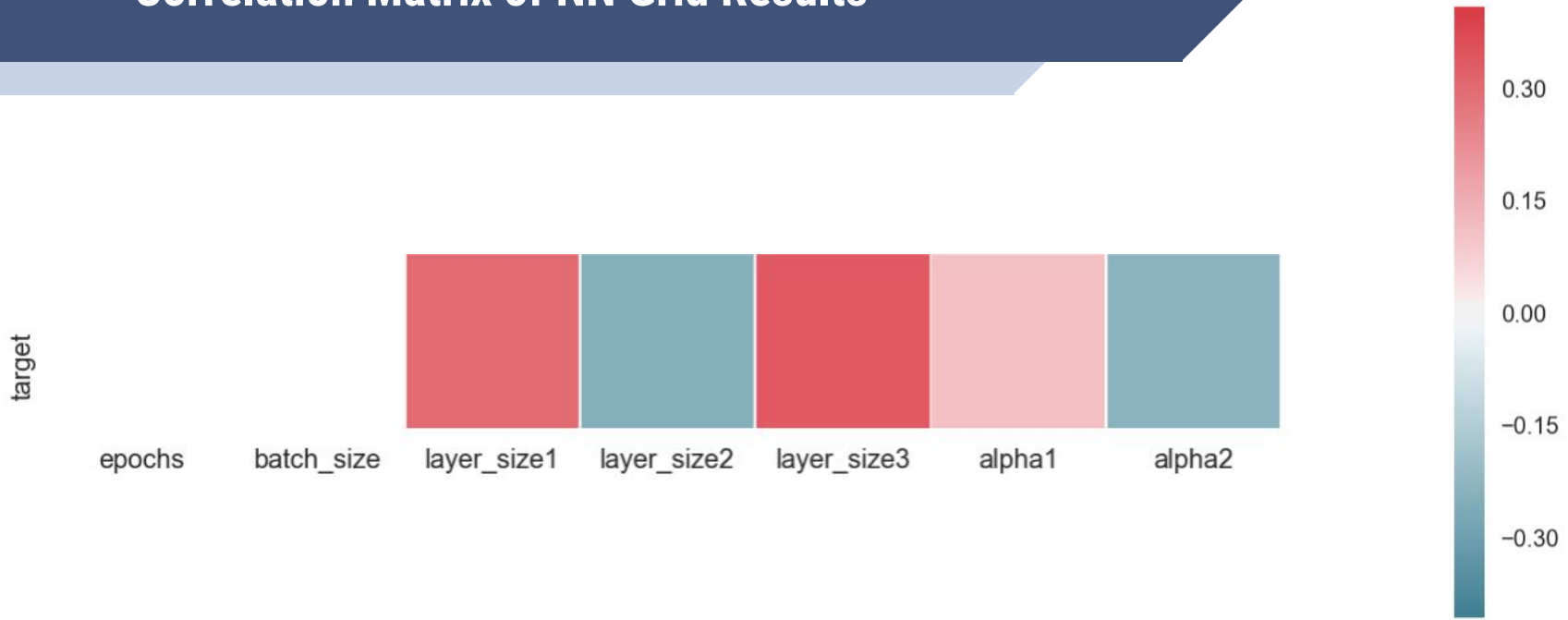
```
epoch = np.array([100])
batch_size = np.array([50, 5])
layer_size1 = np.array([300, 500, 1000])
layer_size2 = np.array([12, 32])
layer_size3 = np.array([6, 12, 32])
alpha1 = np.array([1e-5, 1])
alpha2 = np.array([1e-5, 1])

param_grids = [epoch, batch_size, layer_size1, layer_size2, layer_size3, alpha1, alpha2]
```

Total NN's: 144

Very Computationally Expensive

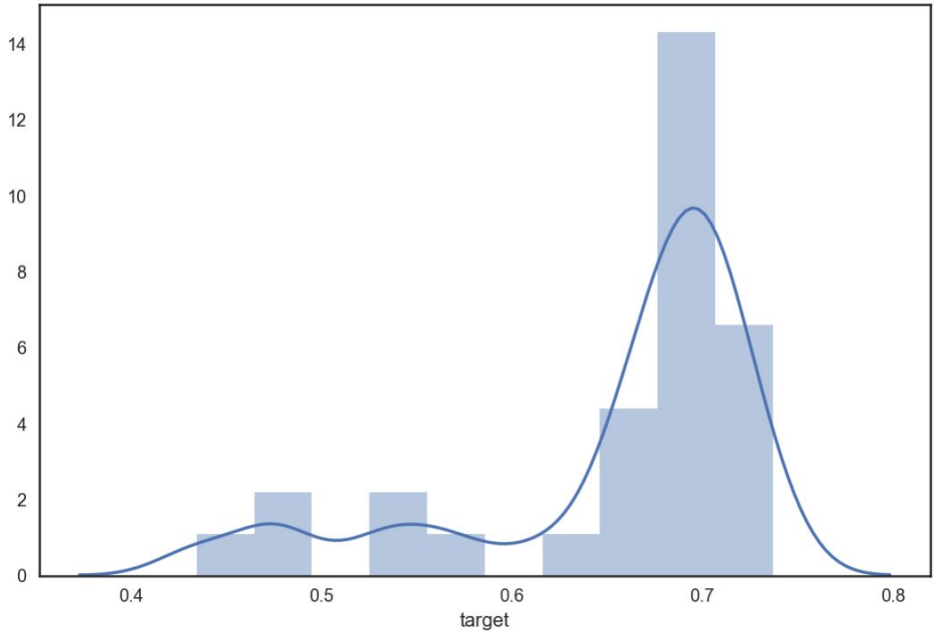
Computational Activity in Google Cloud DB

Correlation Matrix of NN Grid Results



Recurrent Neural Network Distribution Plot

Distribution Plot of ROC Scores



New Datasets: Sklearn Cancer

Simple:

```
Total params: 68
Trainable params: 68
Non-trainable params: 0
_________________________________________________________________
None
ROC:  0.960612454688
Continued Avg:  0.976112378323
('Average ROC:', 0.97611237832261066)
```

Normal:

```
Total params: 100,202
Trainable params: 100,202
Non-trainable params: 0
_________________________________________________________________
None
ROC:  0.98965265379
Continued Avg:  0.988114874063
('Average ROC:', 0.98811487406284204)
```

Recurrent:

```
Evaluating model...
224/284 [======================>.......] - ETA: 0s
ROC:  0.96542769031
('Continued Avg: ', 0.97795046108127837)
('Evaluation time : ', 3.6583268642425537)
('Average ROC:', 0.97795046108127837)
224/284 [======================>.......] - ETA: 0s
```

New Datasets: Kaggle West Nile Virus

Simple:

```
Total params: 358
Trainable params: 358
Non-trainable params: 0
_________________________________________________________________
None
ROC:  0.762607732358
Continued Avg:  0.757393288081
('Average ROC:', 0.75739328808087036)
```

Normal:

```
Total params: 143,702
Trainable params: 143,702
Non-trainable params: 0
_________________________________________________________________
None
ROC:  0.783863132824
Continued Avg:  0.770528541639
('Average ROC:', 0.77052854163869045)
```

Recurrent:

```
Evaluating model...
4237/4237 [==============================] - 17s

ROC:  0.630828071342
('Continued Avg: ', 0.61308660009998373)
('Evaluation time : ', 19.13680386543274)
('Average ROC:', 0.61308660009998373)
4237/4237 [==============================] - 17s
```

LSTM Limitations and Possible Explanations

- Due to complexity and number of parameters fine-precision may be required in order to get accurate results.
- Keras did not have a metric to test learn on a roc score, which was the metric I wanted to test my accuracy with.
- Keras wrapper of tensor flow modules may not work as simply as they may appear.