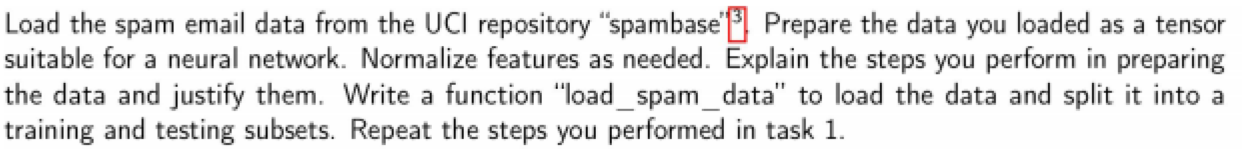
Problem Statement:



Load the spam email data from the UCI repository “spambase”[3]. Prepare the data you loaded as a tensor suitable for a neural network. Normalize features as needed. Explain the steps you perform in preparing the data and justify them. Write a function “load_spam_data” to load the data and split it into a training and testing subsets. Repeat the steps you performed in task 1.

We are trying to improve the performance of our model which use the Spamsbase data set.

Proposed Solution:

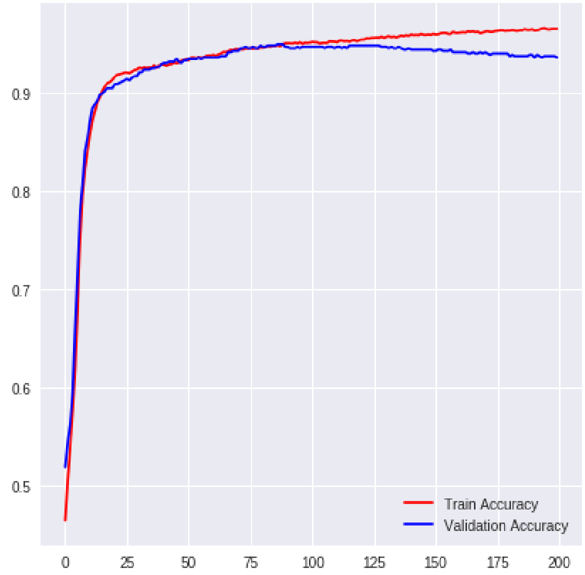
Loading Data

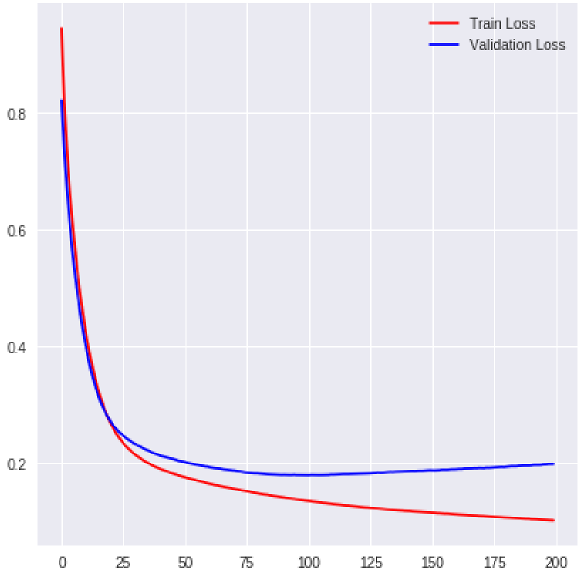
1. Dataset URL: https://archive.ics.uci.edu/ml/machine-learning-databases/spambase/spambase.data
2. Dataset was split into Train and Test sets using SciKitLearn’strain_test_splitmethod, with an 80-20 split.
3. Train Data was further split into an 80-20 split of Train and Validation sets.
4. Final Dataset shapes: x_train: (2944, 57) y_train: (2944,) x_val: (736, 57) y_val: (736,) x_test: (921, 57) y_test: (921,)

Implementation details:

Normalizing Features & Base Model

1. X_train was normalized to have a Mean of 0 and StandarDeviation of 1.
2. Validation and Test predictors were normalized around the mean and standard deviation of the Train predictors.
3. A Base model of 1 layer with 16 Neurons was fit to the dataset yielding the following results:

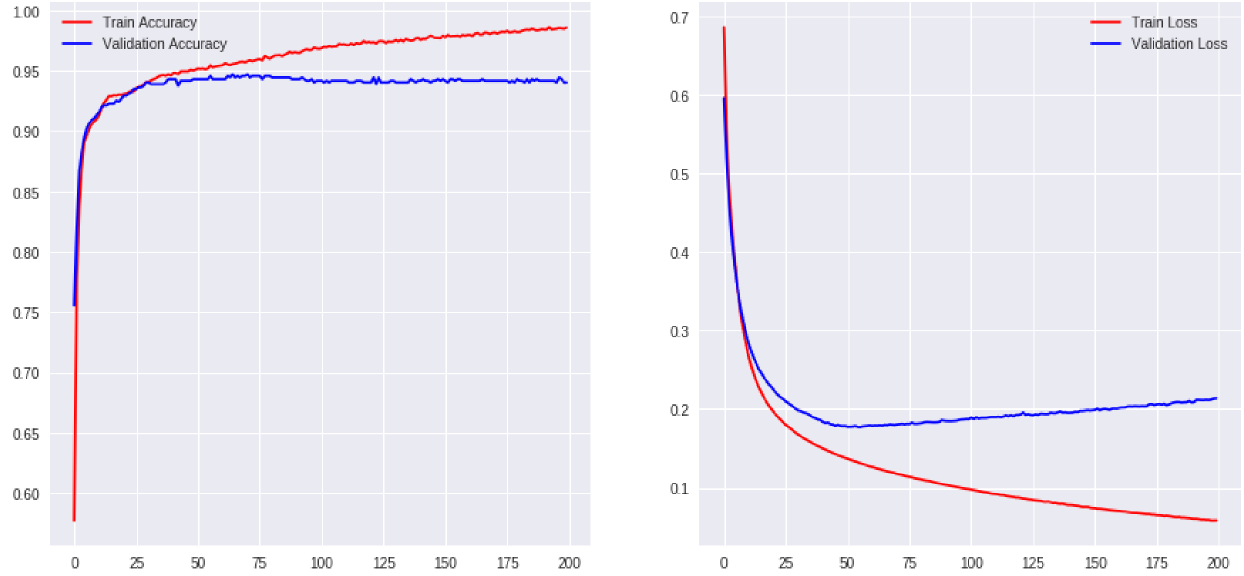




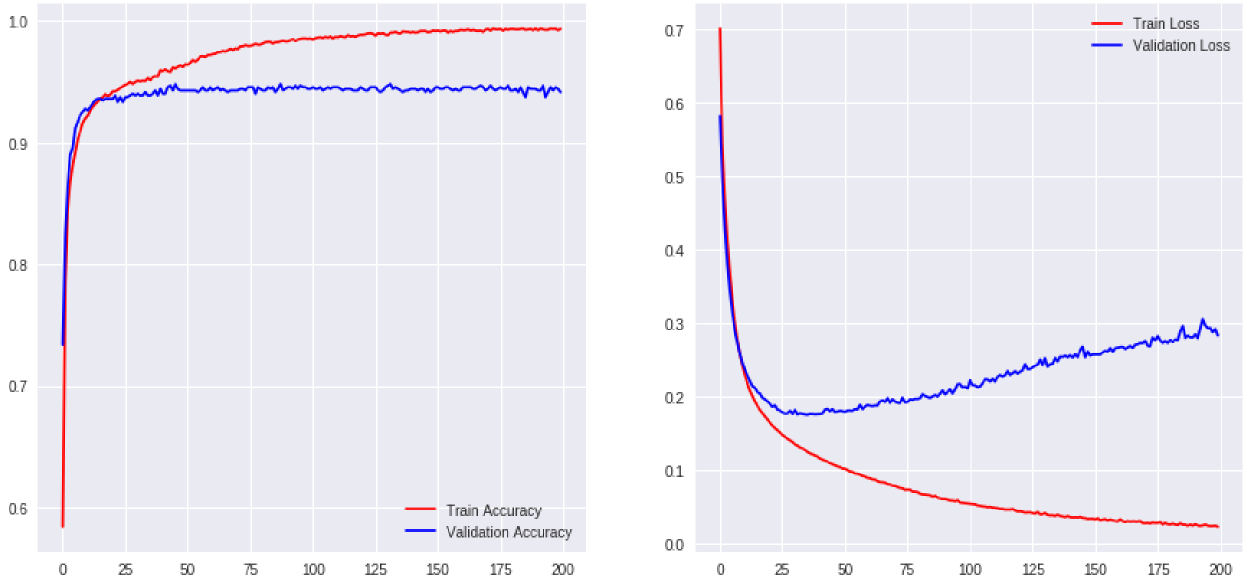
Tuning the Layers

- Two variations of Architectures were implemented and their performance tested:

1 Layer–64 Neurons



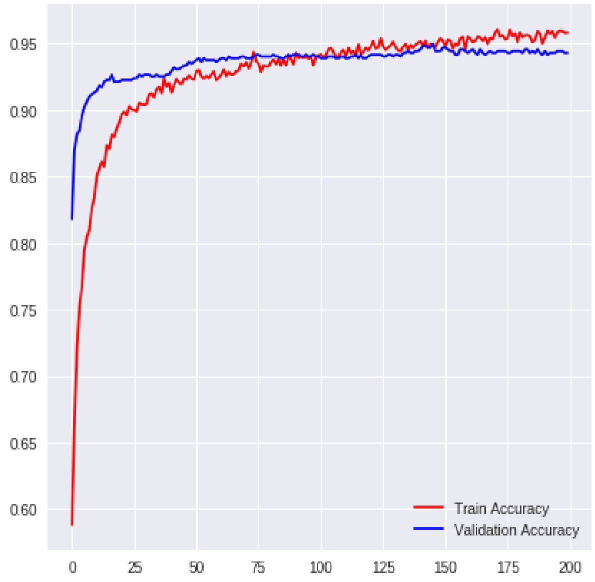
2 Layers –[64, 16] Neurons

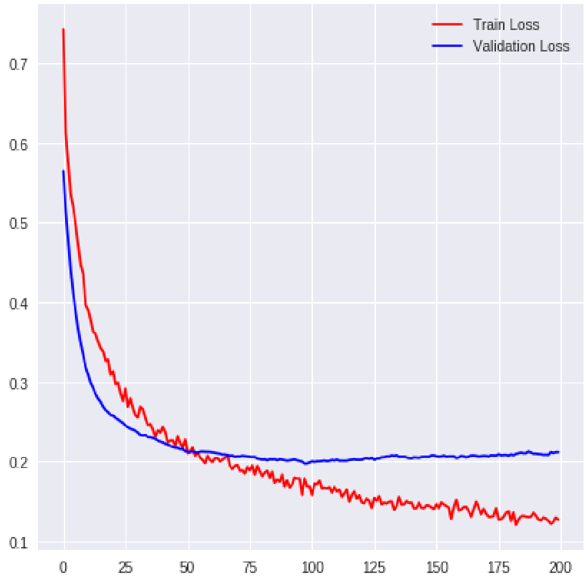


We can see validation loss is going to overfitting, on the other side we are also getting higher accuracy in 2 layer model [64, 16]. Will use it for future.

Added Dropouts:

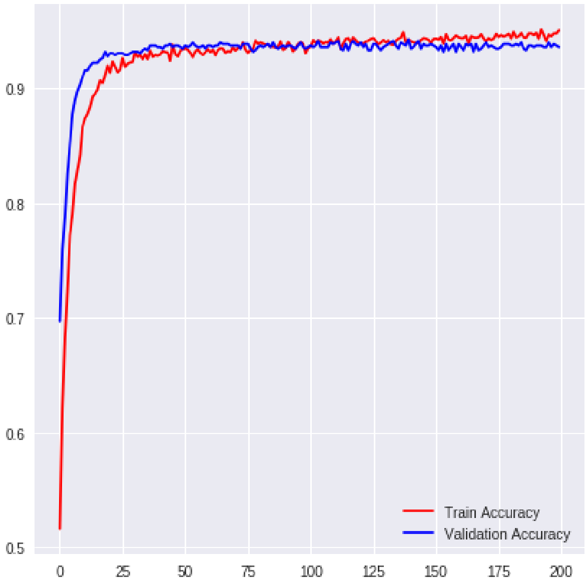
After adding dropouts of 0.2 and 0.5, I can made a conclusion that dropout value 0.5 is more suitable and its reducing the overfitting.

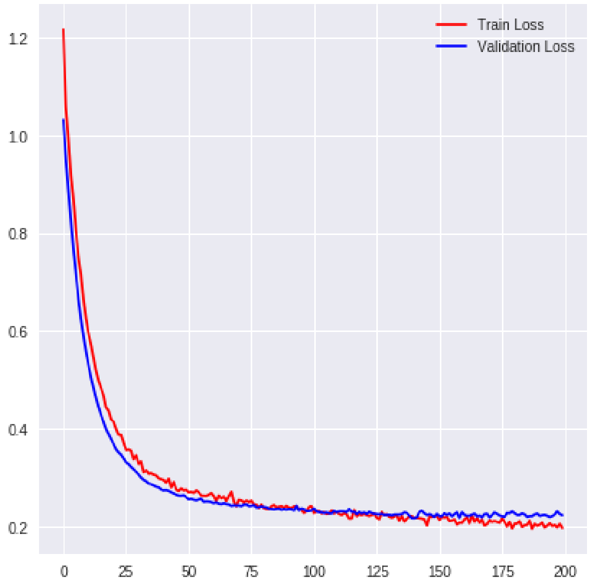




Added Regularization:

I did regularized (L1 and L2) my model on different Regularization rates and found the best combination with L2 on Rate value 0.005 and dropout of 0.5 has the best performance.

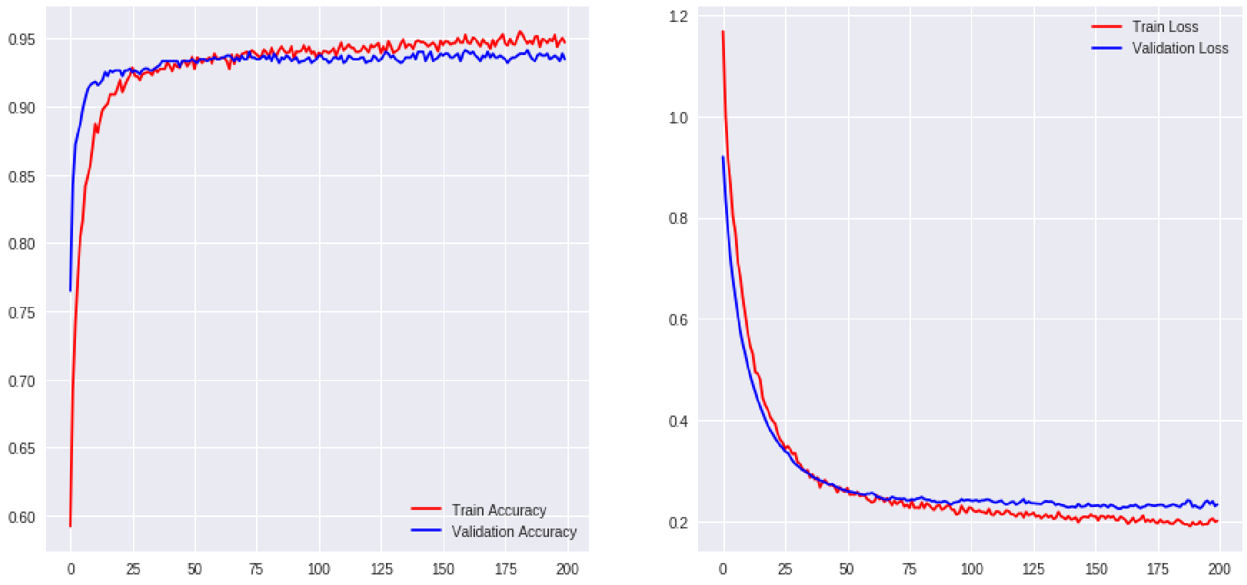




Tuning Learning Rate

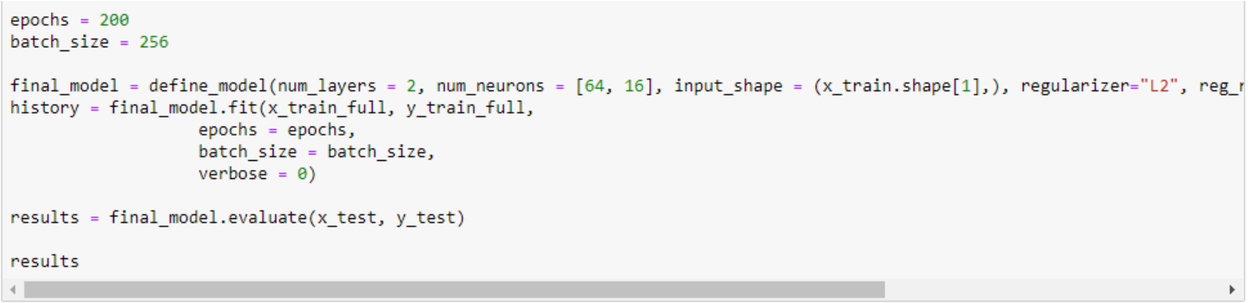
An RMSprop Optimizer is being used for this problem. The following Learning Rates were implemented, with a Learning Rate of 0.001 being most performant. a. 0.01 b. 0.05 c. 0.001 d. 0.005

Performance on a Learning Rate of 0.001



Results and Discussion:

#Model Details
#Number of Layers: 2
#Number of Neurons: [64, 16]
#Learning Rate: 0.001
#Dropout: 0.5
#Regularization: L2
#Regularization Rate: 0.005



```
epochs = 200
batch_size = 256

final_model = define_model(num_layers = 2, num_neurons = [64, 16], input_shape = (x_train.shape[1],), regularizer="L2", reg_
history = final_model.fit(x_train_full, y_train_full,
                          epochs = epochs,
                          batch_size = batch_size,
                          verbose = 0)

results = final_model.evaluate(x_test, y_test)

results
```

print(results)
[0.21839932882177454, 0.9326818585395813]
Accuracy: 93.26%