# Classification of Sonar Signals Using Neural Network

**Dataset:**

| | |
|---|---|
| #Samples | 208 |
| #Features per sample | 60 |
| Feature value range | (0, 1) |
| Class: "Metal" #samples | 111 |
| Class: "Rock" #samples | 97 |

**Methods**:

**Dataset**: Since the dataset size (number of samples) is small, I have used 5-fold testing with each fold is created while maintaining the ratio of each class samples same as in full dataset. For data normalization, I have subtracted 0.5 out of every feature, making the feature values ∈ (-0.5, +0.5). I have not used a separate validation set for fine-tuning the parameters.

**Network**: I have neural networks (Fully Connected layers only) while experimenting with adjusting hyper-parameters. The hyper-parameters description with best test-set performance given in table. Throughout the scope of this work, I have used a batch size of 32.

| | |
|---|---|
| #Hidden Layers | 3 |
| #Units per Layer | 64 |
| Non-Linearity | ReLU |
| Optimizer | RMSprop |
| Loss Function | Binary Cross-Entropy |

**Setup**: I have used Keras with TensorFlow backend to run my experiments with Intel i5-6300HQ 2.3 GHz Quad-Core machine with 8GB RAM and NVIDIA GeForce GTX 960M 4GB GDDR5 GPU.

**Results**: Accuracy and loss results are obtained by averaging performance on all folds. Since no validation set is used, the best results (in the form of confusion matrix) are obtained on test-set for the network that did well on test-set (not a good practice in Machine Learning world). The accuracy is around 83%.

| Confusion Matrix | Predicted 0 | Predicted 1 |
|---|---|---|
| **True 0** | TN = 19.8 | FP = 2.4 |
| **True 1** | FN = 4.6 | TP = 14.8 |

Class 0: Metal
Class 1: Rock

Below are the experimentations that I conducted to analyze the effect the specific hyper-parameter adjustments. whenever needed, the plots are smoothed by a box-filter of window size 11 to suppress the fluctuating values in representation.

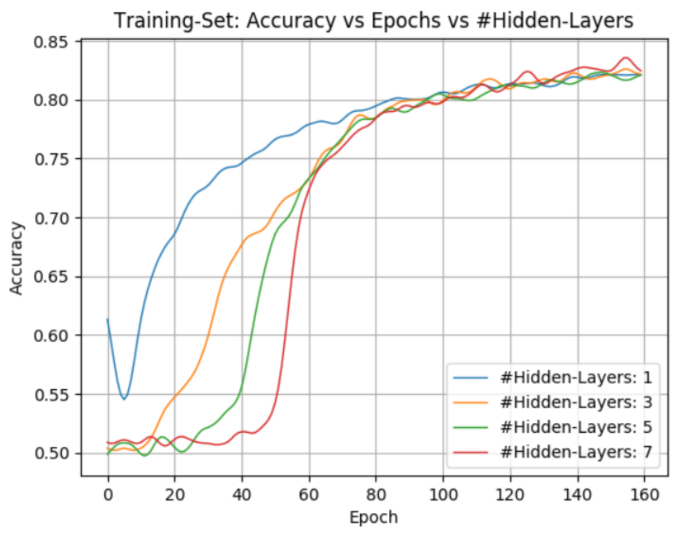
**Effect of number of layers**:

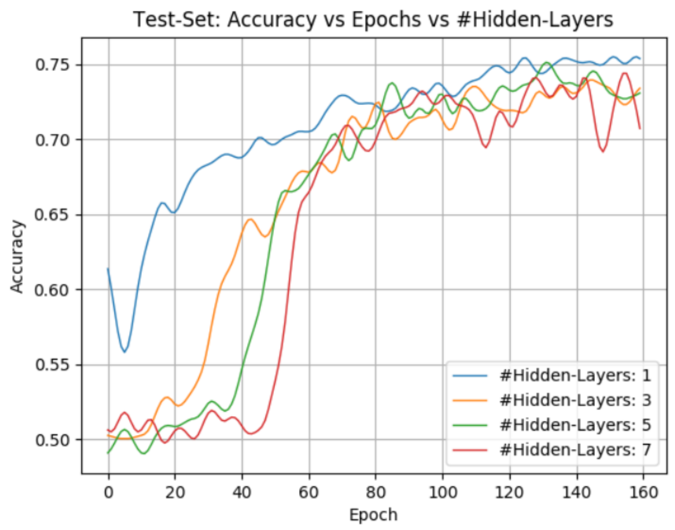
The network parameters used to conduct this experiments:

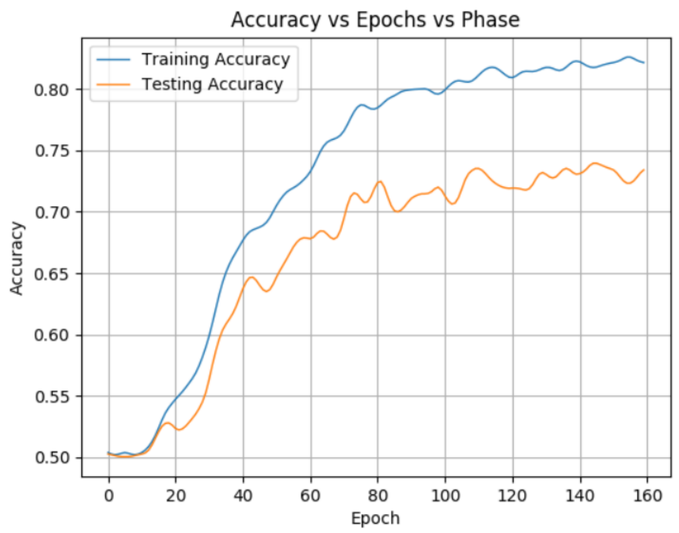
| | |
|---|---|
| #Hidden Layers | --variable-- |
| #Units per Layer | 128 |
| Non-Linearity | Sigmoid |
| Optimizer | RMSprop |
| Loss Function | Binary Cross-Entropy |

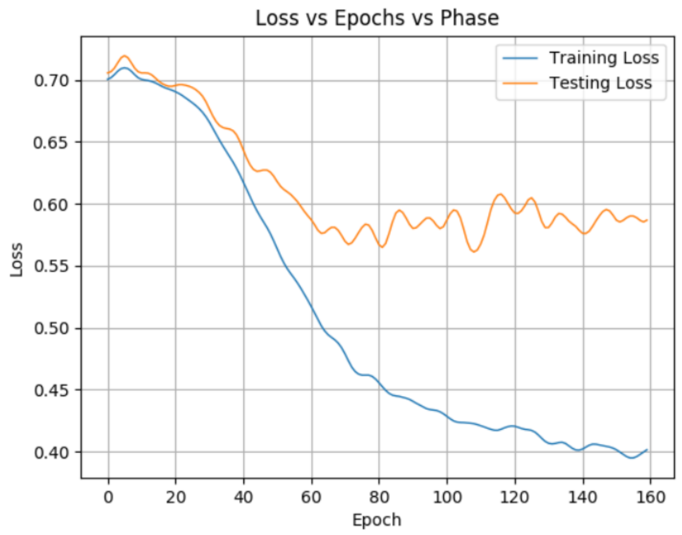
**First plot**: As expected, the final training accuracy improves with increasing layers. However, the difference in accuracy is not significant for 1 to 7 hidden layers. Another observation from plots is that, the time taken for the network performance to saturate is higher with higher number of hidden layers. This is as expected since large number of parameters would need more gradient updates (or epochs) to converge. The best training accuracy was around 0.83.

**Second plot**: The test set performance stays lower than training set with best accuracy around 0.75. Here, we can see the overfitting caused by higher number of layers. We found the best test set performance with just a single hidden layer, and the accuracy starts to come down beyond 5 hidden layers.









**Third plot**: I have fixed the number of layers to 3 to get a comparison between training and testing accuracy. While training accuracy stays higher than test set (as expected), but the test set accuracy saturates at around 70 epochs. The network starts overfitting after that, however we didn't see that causing a significant negative impact on test-set performance.

**Fourth plot**: Just as third plot, number of layers stays 3. The training loss continues to decline with epochs up to 150, however testing loss saturates at around 70 epochs.

## <u>Effect of number of Units</u>:

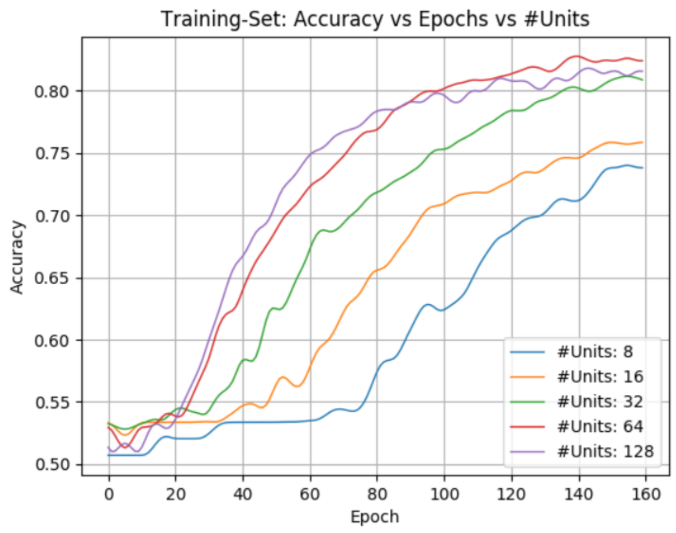
The network parameters used to conduct this experiments:

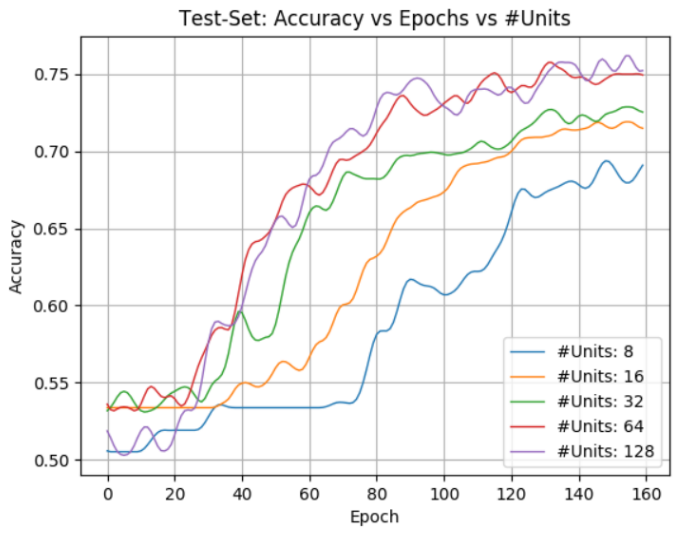
| | |
|---|---|
| #Hidden Layers | 3 |
| #Units per Layer | --variable-- |
| Non-Linearity | Sigmoid |
| Optimizer | RMSprop |
| Loss Function | Binary Cross-Entropy |

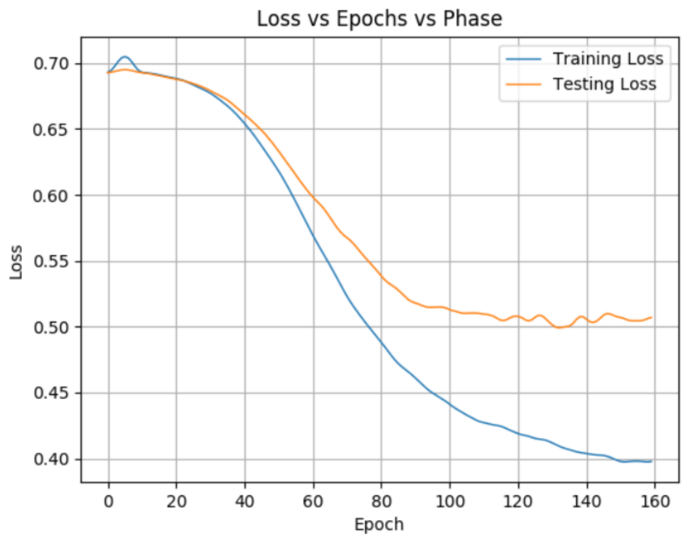
**First plot**: As expected, the final training accuracy improves with increasing number of units. The accuracy at the end of 150 epochs vary from 0.73 to 0.83 for units 8 to 128. Just as in layer experiments, the time taken for the network to converge is higher with higher number of units. The reason being same, that is, large number of parameters would need more gradient updates (or epochs) to converge.

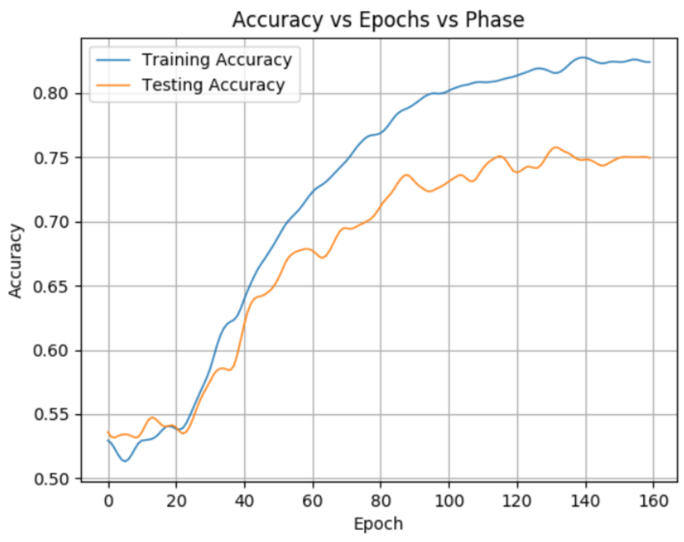
The training accuracy for 128 units is lower than that in 64 units. I don't understand if this is expected or just a glitch.

**Second plot**: The test set performance stays lower than training set with best accuracy around 0.75. Unlike layer experiments, we didn't see evidence of overfitting with increased number of units till 150 epochs. This implies that layers increase model capacity much more than units.









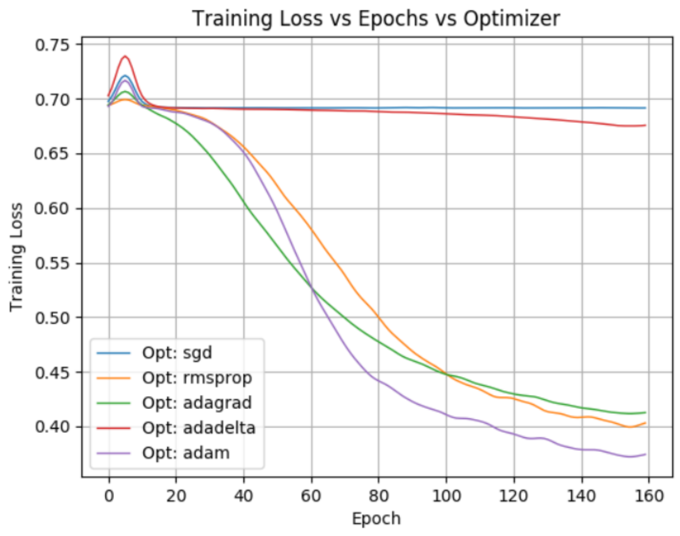
**Third plot**: Fixing the number of units to 64 to see the learning curve, the training loss continues to decrease with epochs (as expected). The test-set loss saturates at around 120 epochs.

**Fourth plot**: Just as third plot, number of units stays 64. The training loss continues to decline with epochs up to 150, however testing loss saturates at around 120 epochs.

**Effect of Optimizers**:

To see the effect of optimizers on the learning curve I have used a few state-of-the-art optimizers.

| | |
|---|---|
| #Hidden Layers | 3 |
| #Units per Layer | 64 |
| Non-Linearity | Sigmoid |
| Optimizer | RMSprop |
| Loss Function | Binary Cross-Entropy |



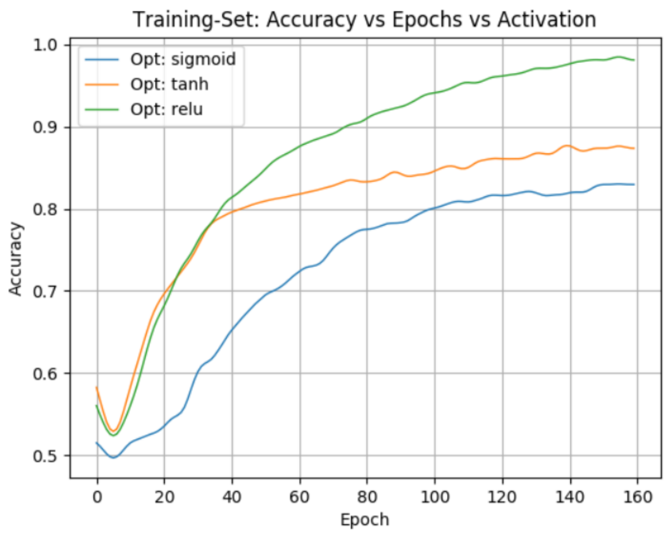
In my experiments, I saw Adam to perform best and SGD didn’t yield any learning at all.

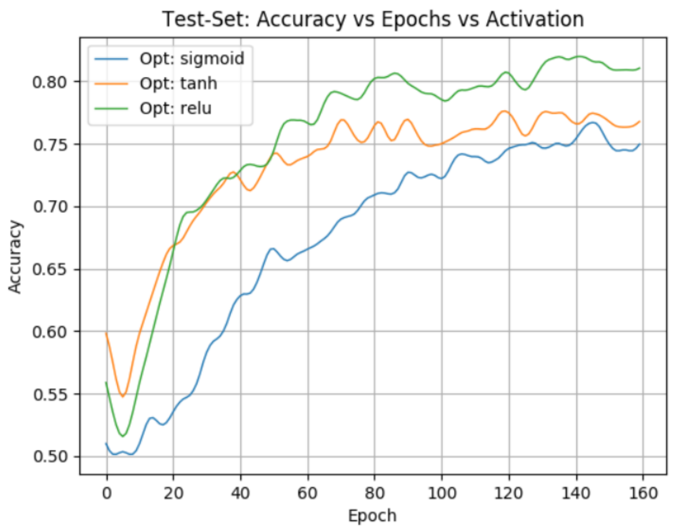
Note: This comparison is not fair, since I didn’t fine-tune the optimizer parameters e.g. learning rate, decay etc. and directly used the default values set by Keras due to limited time availability. With optimal learning parameters, I expect SGD too to converge.

**Effect of Activation Functions**:

To see the effect of optimizers on the learning curve I have used a few state-of-the-art optimizers: Sigmoid, tanh and ReLU. The configuration and results are summarized below.

| | |
|---|---|
| #Hidden Layers | 3 |
| #Units per Layer | 64 |
| Non-Linearity | --variable-- |
| Optimizer | RMSprop |
| Loss Function | Binary Cross-Entropy |





In my experiments, I found that ReLU performed overall best, with Sigmoid being the worst. Since the network is only three hidden layers deep, I did not expect the fading-gradient to be an issue here. However, as can be seen from the epochs 1 to 100, Sigmoid activation learns slower than other (because of smaller gradient magnitude). And even after convergence, the Sigmoid activation does restrict the learning capacity of the network.