**1. LSTM Model v1:**

Below we have defined our first LSTM model. Our model consist of 2 layers(LSTM & Dense). LSTM layers consist of 32 units. Here units generally refers to that many nodes in that layers. Dropout are introduced to prevent model from overfitting. Dropout of 0.2 ignores 20% of outputs and set them to 0 before giving it as input to next layer. Dense layer consist of 1 unit with activation function as sigmoid which will transfer LSTM’s output in range (0,1.0) which is probability of that sample. Probability >0.5 will predict output as 1 and <=0.5 will predict output as 0.

LSTM layer’s units are used to store stateful information in it. We have kept 32 units which will help it store information. Due to 32 units, it requires 10,368 parameters which are calculated as follows. ((input\_size + units + 1) \* input\_size) \* 4 = ((48 + 32 + 1 )\*32) \* 4 = 10,386. Here parameters refers to that many weights will be trained for layer to store information. We have found out that trying more that 50 units overfits data and hence we have tried keep it below 50 units.

Dense layer is simple linear layer which has 33 parameter and single output. Sigmoid activation function transfers output of LSTM from shape of (batch\_size,33) to (batch\_size,1) which is 1 prediction for all samples of batch. Here Dense layer has 33 parameters because it takes as input 32 units of LSTM layer and 1 bias parameter.

Total Parameters are LSTM + Dense = 10,386 + 33 = 10,401

All parameters are trainable. We don’t have single parameter which does not need training. We need to train all.

Below we have printed summary of our model which shows layers and their shapes along with parameters that particular layer will train as well as whole model.

Build model...

Model: "sequential\_8"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type)                 Output Shape              Param #

===============================================================

lstm\_7 (LSTM)                (None, 32)                10368

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

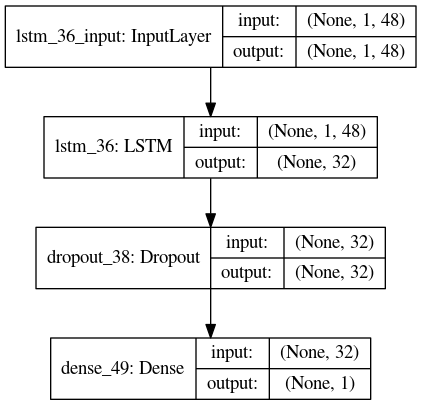
dense\_11 (Dense)             (None, 1)                 33

=================================================================

Total params: 10,401

Trainable params: 10,401

Non-trainable params: 0



**We have tried list of hyper parameters to check performance of model with them:**

We have tried below mentioned list of hyper parameters values on2 train/test splits. One split consist of 80% train – 20% test and another consist of 70% train and 30% test. We looped through all hyper parameter setting and tried all combinations of parameters. We also have plotted model performance as confusion matrix and loss/accuracy graphs over number of epochs. We are also printing model loss/accuracy on train and test sets for all hyperparameter settings. We later print it as single dataframe with all outputs.

**1. Activation Functions**

**Relu** - max(0,x)

**Tanh** - Hyperbolic Tangent of x = (e^2x - 1)/ (e^2x + 1)

**2. Units**

We have tried 2 values of number of units to use for LSTM model 32, 50. We have tried values of units and feel that more than 50 units will take more time for model to train and also it'll overfit to data giving more accuracy on train set and not fitting well with test data.

**3. Dropout**

We are trying out various values of dropout like 205%, 30% and 50%. Dropout zeroes that many output from total output of layer to prevent model from overfitting.

**4. Optimizer**

**Adam** - Adam is an adaptive learning rate optimization algorithm.The algorithms leverages the power of adaptive learning rates methods to find individual learning rates for each parameter.

**Rmsprop** - Its same as SGD with momentum. The RMSprop optimizer restricts the oscillations in the vertical direction. Therefore, we can increase our learning rate and our algorithm could take larger steps in the horizontal direction converging faster.

**5. Regularization**

**l1 Regularization** - It adds “absolute value of magnitude” of coefficient as penalty term to the loss function.It helps preventing model from overfitting.

**l2 Regularization** - It adds “squared magnitude” of coefficient as penalty term to the loss function. It prevents model from overfitting.

**6. Weights Initializations**

**Glorot Uniform** - It finds a good variance for the distribution from which the initial parameters are drawn. This variance is adapted to the activation function used and is derived without explicitly considering the type of the distribution.

**Random Uniform** - It randomly initializes weights.

**Sample Outputs:**

**Output of Default Model:**

Train on 7200 samples, validate on 800 samples

Epoch 1/10

7200/7200 [==============================] - 1s 186us/step - loss: 0.6200 - accuracy: 0.7169 - val\_loss: 0.3409 - val\_accuracy: 0.8775

Epoch 2/10

7200/7200 [==============================] - 1s 85us/step - loss: 0.3320 - accuracy: 0.8608 - val\_loss: 0.2484 - val\_accuracy: 0.9150

Epoch 3/10

7200/7200 [==============================] - 1s 83us/step - loss: 0.2671 - accuracy: 0.8869 - val\_loss: 0.2000 - val\_accuracy: 0.9212

Epoch 4/10

7200/7200 [==============================] - 1s 85us/step - loss: 0.2263 - accuracy: 0.9100 - val\_loss: 0.1833 - val\_accuracy: 0.9388

Epoch 5/10

7200/7200 [==============================] - 1s 82us/step - loss: 0.1997 - accuracy: 0.9207 - val\_loss: 0.1721 - val\_accuracy: 0.9362

Epoch 6/10

7200/7200 [==============================] - 1s 83us/step - loss: 0.1856 - accuracy: 0.9276 - val\_loss: 0.1568 - val\_accuracy: 0.9400

Epoch 7/10

7200/7200 [==============================] - 1s 85us/step - loss: 0.1817 - accuracy: 0.9290 - val\_loss: 0.1537 - val\_accuracy: 0.9450

Epoch 8/10

7200/7200 [==============================] - 1s 83us/step - loss: 0.1730 - accuracy: 0.9349 - val\_loss: 0.1482 - val\_accuracy: 0.9500

Epoch 9/10

7200/7200 [==============================] - 1s 84us/step - loss: 0.1660 - accuracy: 0.9354 - val\_loss: 0.1521 - val\_accuracy: 0.9488

Epoch 10/10

7200/7200 [==============================] - 1s 83us/step - loss: 0.1623 - accuracy: 0.9364 - val\_loss: 0.1479 - val\_accuracy: 0.9513

2000/2000 [==============================] - 0s 19us/step

8000/8000 [==============================] - 0s 17us/step

Test Loss: 0.151

Train Loss: 0.128

Test accuracy: 94.65

Train accuracy: 95.51

**Hyperparameter output:**

Activation : relu, Units : 32, Dropout : 0.25, Optimizer : adam, Regularization : l1, Weights Inits : glorot\_uniform

2000/2000 [==============================] - 0s 35us/step

8000/8000 [==============================] - 0s 30us/step

Test Loss: 0.423

Train Loss: 0.419

Test accuracy: 90.55

Train accuracy: 91.47

Activation : relu, Units : 32, Dropout : 0.25, Optimizer : adam, Regularization : l1, Weights Inits : random\_uniform

2000/2000 [==============================] - 0s 30us/step

8000/8000 [==============================] - 0s 27us/step

Test Loss: 0.383

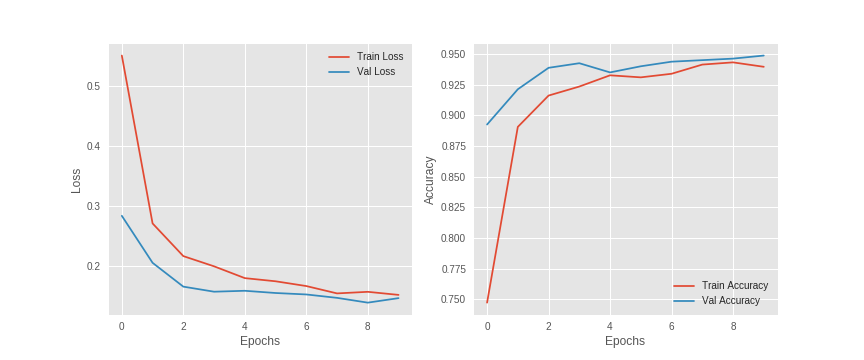
Train Loss: 0.378

Test accuracy: 90.80

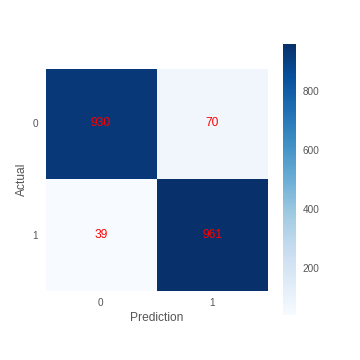
Train accuracy: 91.75

Here we are printing all parameter settings and their values which were tried on model as well as test loss, train loss, test accuracy and train accuracy. We first print what parameter and what values were tried. Then we train model using that parameter. Once done we evaluate model performance on train and test data both. We print loss and accuracy on both train and test sets.

**Plotting Training Progress:**

We have also plotted model performance by plotting decrease in loss on train and validation sets as well as accuracy increase on train and validation during training. Here x axis refers to epochs and y axis refers to loss in first figure and accuracy in second figure. We can see that as number of epochs increase loss is decreasing and accuracy is increasing. Train and Validation loss/accuracy are represented by different lines with legends to identify which one is which.

**Plotting Confusion Matrix:**



We can see from confusion matrix on left that how many of actual positive predictions were actually predicted positive and same for negative as well. Here y axis refers to original labels and y axis refers to predictions.

We can see 930 original label of 0 were predicted as 0 which is called **True Negative**. 961 of original label 1 were predicted as 1 which is called **True Positive**. 70 original label which were 0 were predicted as 1 which is called **False Positive**. 39 original label 1 were predicted as 0 which is called **False Negatives**.

**2. LSTM Model v2:**

Below we have defined our second LSTM model. Our model consist of 4 layers(3 LSTM & 1 Dense). LSTM layers consist of 32 units. Here units generally refers to that many nodes in that layers. Dropout are introduced to prevent model from overfitting. Dropout of 0.2 ignores 20% of outputs and set them to 0 before giving it as input to next layer. Dense layer consist of 1 unit with activation function as sigmoid which will transfer LSTM’s output in range (0,1.0) which is probability of that sample. Probability >0.5 will predict output as 1 and <=0.5 will predict output as 0.

LSTM layer’s units are used to store stateful information in it. We have kept 32 units which will help it store information. Due to 32 units, it requires 10,368 parameters which are calculated as follows. ((input\_size + units + 1) \* input\_size) \* 4 = ((48 + 32 + 1 )\*32) \* 4 = 10,386. Here parameters refers to that many weights will be trained for layer to store information. We have found out that trying more that 50 units overfits data and hence we have tried keep it below 50 units.

Dense layer is simple linear layer which has 33 parameter and single output. Sigmoid activation function transfers output of LSTM from shape of (batch\_size,33) to (batch\_size,1) which is 1 prediction for all samples of batch. Here Dense layer has 33 parameters because it takes as input 32 units of LSTM layer and 1 bias parameter.

All parameters are trainable. We don’t have single parameter which does not need training. We need to train all.

We selected this model as our another LSTM model to check whether adding more LSTM layers adds performance improvement to our existing LSTM model of 1 LSTM layers. To check performance against 1 layer LSTM, we added 2 more LSTM layers to see whether there is any performance improvements.

Below we have printed summary of our model which shows layers and their shapes along with parameters that particular layer will train as well as whole model.

Build model...

Model: "sequential\_28"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

lstm\_30 (LSTM) (None, 1, 32) 10368

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

lstm\_31 (LSTM) (None, 1, 32) 8320

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

lstm\_32 (LSTM) (None, 32) 8320

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

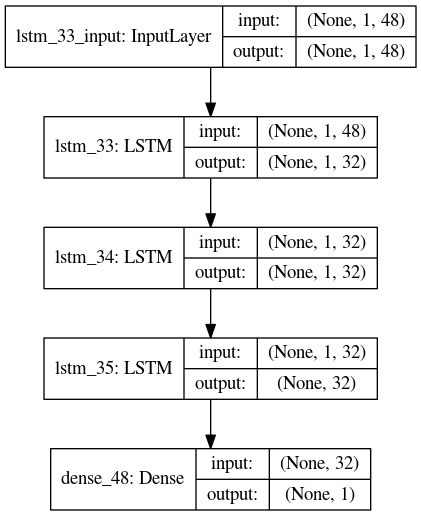
dense\_43 (Dense) (None, 1) 33

=================================================================

Total params: 27,041

Trainable params: 27,041

Non-trainable params: 0



**We have tried list of hyper parameters to check performance of model with them. All parameters have same values as that of tried with version 1 model defined above.**

**Sample Outputs:**

**Output of Default Model:**

Train on 7200 samples, validate on 800 samples

Epoch 1/10

7200/7200 [==============================] - 2s 336us/step - loss: 0.5763 - accuracy: 0.7089 - val\_loss: 0.3207 - val\_accuracy: 0.8938

Epoch 2/10

7200/7200 [==============================] - 1s 138us/step - loss: 0.3423 - accuracy: 0.8537 - val\_loss: 0.2152 - val\_accuracy: 0.9237

Epoch 3/10

7200/7200 [==============================] - 1s 136us/step - loss: 0.3146 - accuracy: 0.8683 - val\_loss: 0.2195 - val\_accuracy: 0.9187

Epoch 4/10

7200/7200 [==============================] - 1s 138us/step - loss: 0.2971 - accuracy: 0.8768 - val\_loss: 0.2029 - val\_accuracy: 0.9300

Epoch 5/10

7200/7200 [==============================] - 1s 132us/step - loss: 0.2762 - accuracy: 0.8811 - val\_loss: 0.1969 - val\_accuracy: 0.9225

Epoch 6/10

7200/7200 [==============================] - 1s 134us/step - loss: 0.2718 - accuracy: 0.8824 - val\_loss: 0.1966 - val\_accuracy: 0.9287

Epoch 7/10

7200/7200 [==============================] - 1s 137us/step - loss: 0.2699 - accuracy: 0.8842 - val\_loss: 0.1963 - val\_accuracy: 0.9237

Epoch 8/10

7200/7200 [==============================] - 1s 134us/step - loss: 0.2652 - accuracy: 0.8864 - val\_loss: 0.2364 - val\_accuracy: 0.9125

Epoch 9/10

7200/7200 [==============================] - 1s 132us/step - loss: 0.2660 - accuracy: 0.8879 - val\_loss: 0.1941 - val\_accuracy: 0.9237

Epoch 10/10

7200/7200 [==============================] - 1s 134us/step - loss: 0.2747 - accuracy: 0.8861 - val\_loss: 0.1775 - val\_accuracy: 0.9325

2000/2000 [==============================] - 0s 25us/step

8000/8000 [==============================] - 0s 24us/step

Test Loss: 0.186

Train Loss: 0.167

Test accuracy: 93.00

Train accuracy: 93.86

**Hyperparameter output:**

Activation : relu, Units : 32, Dropout : 0.50, Optimizer : rmsprop, Regularization : l2, Weights Inits : glorot\_uniform

Test Loss: 0.398

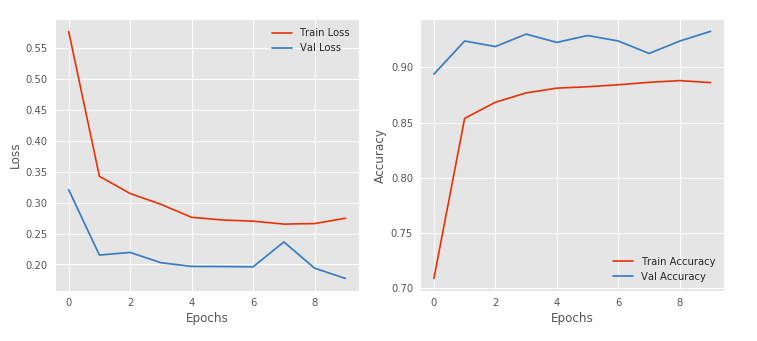
Train Loss: 0.386

Test accuracy: 87.50

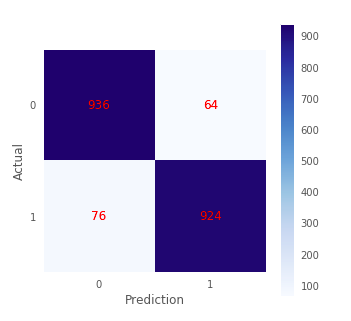
Train accuracy: 88.48

Here we are printing all parameter settings and their values which were tried on model as well as test loss, train loss, test accuracy and train accuracy. We first print what parameter and what values were tried. Then we train model using that parameter. Once done we evaluate model performance on train and test data both. We print loss and accuracy on both train and test sets.

**Plotting Training Progress:**

****

**Plotting Confusion Matrix:**

****

**3. Dense Model**

Below we have defined out deep learning model which consists of only dense layers (linear layers). We have defined 4 dense layers with units 256,128,64 and 1. Here units generally refers to that many nodes in that layers. Input to first layer will be array of shape (batch\_size, 48) and it’ll go through all layers. Each layer has activation function as relu (max(val, 0)) which removes negative outputs after each layer performs its operation. Output layer is Dense layer with sigmoid function which will transfer output between (0,1.0). Each layers has parameters which are calculated as follows.

We have selected model with 4 layers because we have found out after trying models with 2-3 layers that it gives better accuracy than 2-3 layer models also fits our data as well. We have selected units of model as power of 2 and have tried various combination in same way before reaching this final model.

Dense Layer 1 = (input\_size \* units) + biases = (48 \* 256) + 256 = 12,544

Dense Layer 2 = (256 \* 128) + 128 = 32,896 ( Please make a note that here input is 256 from Dense Layer 1)

Dense Layer 3 = (128 \* 64) + 64 = 8,256

Dense Layer 4 = (64 \* 1) + 1 = 65

Build model...

Model: "sequential\_9"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type)                 Output Shape              Param #

=================================================================

dense\_12 (Dense)             (None, 256)               12544

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_13 (Dense)             (None, 128)               32896

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_14 (Dense)             (None, 64)                8256

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

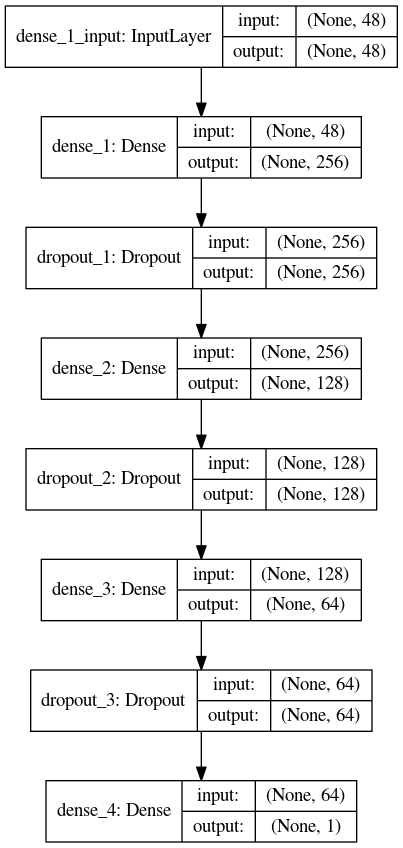
dense\_15 (Dense)             (None, 1)                 65

=================================================================

Total params: 53,761

Trainable params: 53,761

Non-trainable params: 0

****

**We have tried list of hyper parameters to check performance of model with them. All parameters have same values as that of tried with version 1 model defined above.**

**Sample Outputs:**

**Output of Default Model:**

Train on 6300 samples, validate on 700 samples

Epoch 1/10

6300/6300 [==============================] - 15s 2ms/step - loss: 0.7529 - accuracy: 0.6094 - val\_loss: 0.4845 - val\_accuracy: 0.8229

Epoch 2/10

6300/6300 [==============================] - 3s 469us/step - loss: 0.4109 - accuracy: 0.8214 - val\_loss: 0.2588 - val\_accuracy: 0.8871

Epoch 3/10

6300/6300 [==============================] - 3s 469us/step - loss: 0.2688 - accuracy: 0.8965 - val\_loss: 0.2208 - val\_accuracy: 0.9114

Epoch 4/10

6300/6300 [==============================] - 3s 473us/step - loss: 0.2149 - accuracy: 0.9197 - val\_loss: 0.1754 - val\_accuracy: 0.9329

Epoch 5/10

6300/6300 [==============================] - 3s 476us/step - loss: 0.1920 - accuracy: 0.9289 - val\_loss: 0.1741 - val\_accuracy: 0.9357

Epoch 6/10

6300/6300 [==============================] - 3s 478us/step - loss: 0.1897 - accuracy: 0.9276 - val\_loss: 0.1604 - val\_accuracy: 0.9443

Epoch 7/10

6300/6300 [==============================] - 3s 472us/step - loss: 0.1628 - accuracy: 0.9397 - val\_loss: 0.1549 - val\_accuracy: 0.9429

Epoch 8/10

6300/6300 [==============================] - 3s 469us/step - loss: 0.1603 - accuracy: 0.9397 - val\_loss: 0.1481 - val\_accuracy: 0.9443

Epoch 9/10

6300/6300 [==============================] - 3s 504us/step - loss: 0.1561 - accuracy: 0.9400 - val\_loss: 0.1519 - val\_accuracy: 0.9443

Epoch 10/10

6300/6300 [==============================] - 3s 484us/step - loss: 0.1429 - accuracy: 0.9454 - val\_loss: 0.1661 - val\_accuracy: 0.9400

3000/3000 [==============================] - 1s 179us/step

7000/7000 [==============================] - 1s 184us/step

Test Loss: 0.159

Train Loss: 0.140

Test accuracy: 94.93

Train accuracy: 94.93

**Hyperparameter output:**

Activation : relu, Dropout : 0.25, Optimizer : rmsprop, Regularization : l2, Weights Inits : glorot\_uniform

Test Loss: 0.444

Train Loss: 0.424

Test accuracy: 84.63

Train accuracy: 86.23

Activation : relu, Dropout : 0.25, Optimizer : rmsprop, Regularization : l2, Weights Inits : random\_uniform

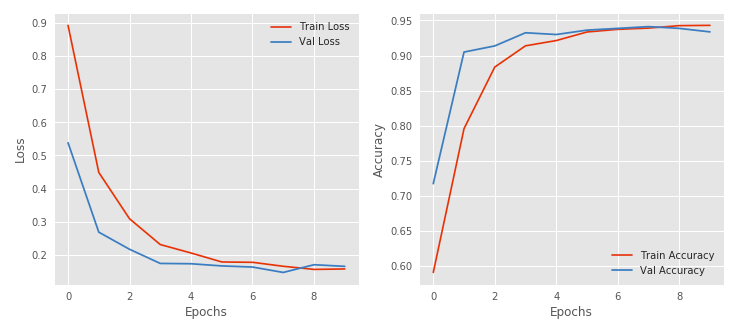
Test Loss: 0.347

Train Loss: 0.333

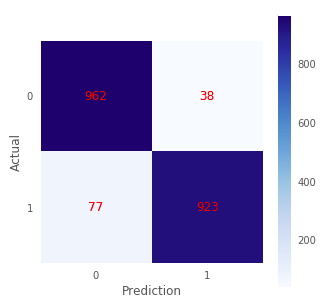
Test accuracy: 89.63

Train accuracy: 89.89

**Plotting Training Progress:**

****

**Plotting Confusion Matrix:**

****