# Assignment12

June 1, 2019

```
In [1]: # if you keras is not using tensorflow as backend set "KERAS_BACKEND=tensorflow" use t
        from keras.utils import np_utils
        from keras.datasets import mnist
        import seaborn as sns
        from keras.initializers import RandomNormal
```

```
Using TensorFlow backend.
```

```
In [0]: %matplotlib notebook
        import matplotlib.pyplot as plt
        import numpy as np
        import time
        # https://gist.github.com/greydanus/f6eee59eaf1d90fcb3b534a25362cea4
        # https://stackoverflow.com/a/14434334
        # this function is used to update the plots for each epoch and error
        def plt_dynamic(x, vy, ty, ax, colors=['b']):
            ax.plot(x, vy, 'b', label="Validation Loss")
            ax.plot(x, ty, 'r', label="Train Loss")
            plt.legend()
            plt.grid()
            fig.canvas.draw()
```

```
In [0]: # the data, shuffled and split between train and test sets
        (X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
In [4]: print("Number of training examples :", X_train.shape[0], "and each image is of shape (
        print("Number of testing examples :", X_test.shape[0], "and each image is of shape (%d
```

```
Number of training examples : 60000 and each image is of shape (28, 28)
Number of testing examples : 10000 and each image is of shape (28, 28)
```

```
In [0]: # if you observe the input shape its 2 dimensional vector
        # for each image we have a (28*28) vector
        # we will convert the (28*28) vector into single dimensional vector of 1 * 784

        X_train = X_train.reshape(X_train.shape[0], X_train.shape[1]*X_train.shape[2])
        X_test = X_test.reshape(X_test.shape[0], X_test.shape[1]*X_test.shape[2])
```

```
In [6]: X_train.shape

Out[6]: (60000, 784)

In [7]: # after converting the input images from 3d to 2d vectors

        print("Number of training examples :", X_train.shape[0], "and each image is of shape (
        print("Number of training examples :", X_test.shape[0], "and each image is of shape (%

Number of training examples : 60000 and each image is of shape (784)
Number of training examples : 10000 and each image is of shape (784)


In [8]: # An example data point
        print(X_train[0])

[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   3  18  18  18 126 136 175  26 166 255
 247 127   0   0   0   0   0   0   0   0   0   0   0   0  30  36  94 154
 170 253 253 253 253 253 225 172 253 242 195  64   0   0   0   0   0   0
   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251  93  82
  82  56  39   0   0   0   0   0   0   0   0   0   0   0   0  18 219 253
 253 253 253 253 198 182 247 241   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0  14   1 154 253  90   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0  11 190 253  70   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  35 241
 225 160 108   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0  81 240 253 253 119  25   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  45 186 253 253 150  27   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252 253 187
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0 249 253 249  64   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253
 253 207   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0  39 148 229 253 253 253 250 182   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253
 253 201  78   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

```
   0   0  23  66 213 253 253 253 253 198  81   2   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0  18 171 219 253 253 253 253 195
  80   9   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  55 172 226 253 253 253 253 244 133  11   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0 136 253 253 253 212 135 132  16
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
```

```
In [0]: # if we observe the above matrix each cell is having a value between 0-255
        # before we move to apply machine learning algorithms lets try to normalize the data
        # X => (X - Xmin)/(Xmax-Xmin) = X/255

        X_train = X_train/255
        X_test = X_test/255
```

```
In [10]: # example data point after normlizing
         print(X_train[0])
```

```
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
```

```
0.         0.         0.01176471 0.07058824 0.07058824 0.07058824
0.49411765 0.53333333 0.68627451 0.10196078 0.65098039 1.
0.96862745 0.49803922 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.11764706 0.14117647 0.36862745 0.60392157
0.66666667 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
0.88235294 0.6745098  0.99215686 0.94901961 0.76470588 0.25098039
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.19215686
0.93333333 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
0.99215686 0.99215686 0.99215686 0.98431373 0.36470588 0.32156863
0.32156863 0.21960784 0.15294118 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.07058824 0.85882353 0.99215686
0.99215686 0.99215686 0.99215686 0.99215686 0.77647059 0.71372549
0.96862745 0.94509804 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.31372549 0.61176471 0.41960784 0.99215686
0.99215686 0.80392157 0.04313725 0.         0.16862745 0.60392157
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.05490196 0.00392157 0.60392157 0.99215686 0.35294118
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.54509804 0.99215686 0.74509804 0.00784314 0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.04313725
0.74509804 0.99215686 0.2745098  0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.1372549  0.94509804
0.88235294 0.62745098 0.42352941 0.00392157 0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.31764706 0.94117647 0.99215686
0.99215686 0.46666667 0.09803922 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.17647059 0.72941176 0.99215686 0.99215686
```

```
0.58823529 0.10588235 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.0627451  0.36470588 0.98823529 0.99215686 0.73333333
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.97647059 0.99215686 0.97647059 0.25098039 0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.18039216 0.50980392 0.71764706 0.99215686
0.99215686 0.81176471 0.00784314 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.15294118 0.58039216
0.89803922 0.99215686 0.99215686 0.99215686 0.98039216 0.71372549
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.09411765 0.44705882 0.86666667 0.99215686 0.99215686 0.99215686
0.99215686 0.78823529 0.30588235 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.09019608 0.25882353 0.83529412 0.99215686
0.99215686 0.99215686 0.99215686 0.77647059 0.31764706 0.00784314
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.07058824 0.67058824
0.85882353 0.99215686 0.99215686 0.99215686 0.99215686 0.76470588
0.31372549 0.03529412 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.21568627 0.6745098  0.88627451 0.99215686 0.99215686 0.99215686
0.99215686 0.95686275 0.52156863 0.04313725 0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.53333333 0.99215686
0.99215686 0.99215686 0.83137255 0.52941176 0.51764706 0.0627451
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
```

```
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.        ]
```

```
In [11]: # here we are having a class number for each image
         print("Class label of first image :", y_train[0])

         # lets convert this into a 10 dimensional vector
         # ex: consider an image is 5 convert it into 5 => [0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
         # this conversion needed for MLPs

         Y_train = np_utils.to_categorical(y_train, 10)
         Y_test = np_utils.to_categorical(y_test, 10)

         print("After converting the output into a vector : ",Y_train[0])
```

```
Class label of first image : 5
After converting the output into a vector :  [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

```
In [0]: from keras.models import Sequential
        from keras.layers import Dense, Activation
        # some model parameters

        output_dim = 10
        input_dim = X_train.shape[1]

        batch_size = 128
```

# 1 Defining a function for model with dropout as well as batch normalisation

```
In [0]: # https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormalization-

        from keras.layers import Dropout
        from keras.layers.normalization import BatchNormalization
        from keras import initializers as K

        def nn(n_layers,d):
          model = Sequential()
```

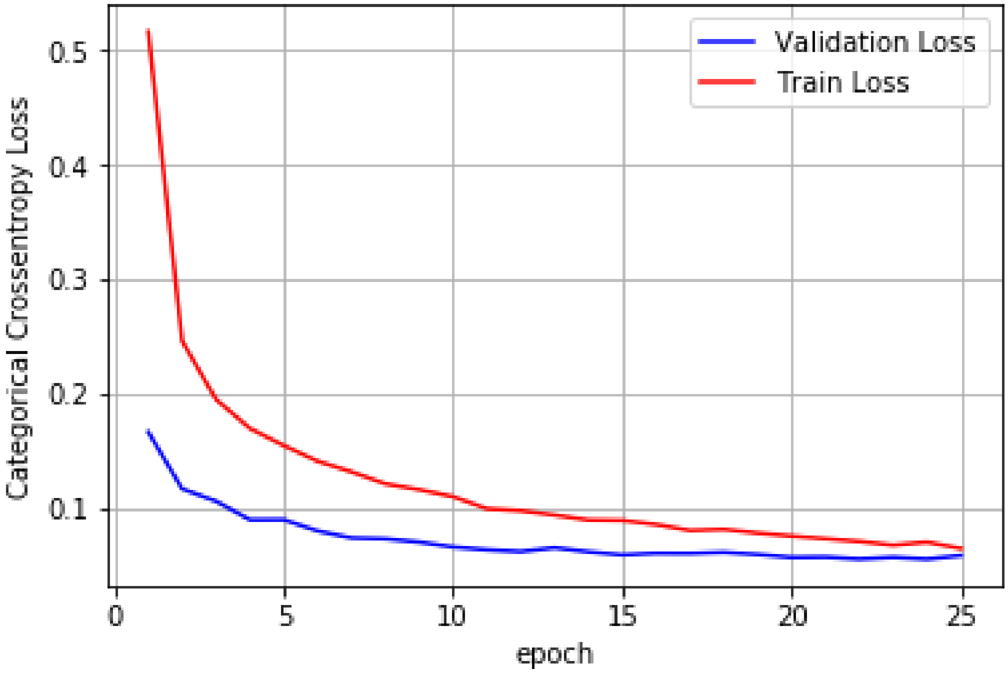
```
model.add(Dense(n_layers[0], activation='relu', input_shape=(input_dim,), kernel_ini
model.add(BatchNormalization())
model.add(Dropout(d[0]))

for i in range(len(n_layers)-1):
  model.add(Dense(n_layers[i+1], activation='relu',kernel_initializer=K.he_normal(se
  model.add(BatchNormalization())
  model.add(Dropout(d[i+1]))

model.add(Dense(output_dim, activation='softmax'))


model.summary()


model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'

history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbos


%matplotlib inline
score = model.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

 #dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,

# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

### 1.1 Architecture with 2 hidden layers.

```
In [78]: n_layers = [468,92]
         d = [0.6,0.5]
         nb_epoch = 25
         nn(n_layers,d)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_128 (Dense)            (None, 468)               367380
_________________________________________________________________
batch_normalization_85 (Batc (None, 468)               1872
_________________________________________________________________
dropout_65 (Dropout)         (None, 468)               0
_________________________________________________________________
dense_129 (Dense)            (None, 92)                43148
_________________________________________________________________
batch_normalization_86 (Batc (None, 92)                368
_________________________________________________________________
dropout_66 (Dropout)         (None, 92)                0
_________________________________________________________________
dense_130 (Dense)            (None, 10)                930
=================================================================
Total params: 413,698
Trainable params: 412,578
Non-trainable params: 1,120
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/25
60000/60000 [==============================] - 13s 210us/step - loss: 0.5165 - acc: 0.8445 - va
Epoch 2/25
60000/60000 [==============================] - 4s 64us/step - loss: 0.2460 - acc: 0.9273 - val_
Epoch 3/25
60000/60000 [==============================] - 4s 65us/step - loss: 0.1948 - acc: 0.9419 - val_
Epoch 4/25
60000/60000 [==============================] - 4s 65us/step - loss: 0.1696 - acc: 0.9491 - val_
Epoch 5/25
60000/60000 [==============================] - 4s 65us/step - loss: 0.1546 - acc: 0.9530 - val_
Epoch 6/25
60000/60000 [==============================] - 4s 66us/step - loss: 0.1410 - acc: 0.9573 - val_
Epoch 7/25
60000/60000 [==============================] - 4s 65us/step - loss: 0.1316 - acc: 0.9617 - val_
Epoch 8/25
60000/60000 [==============================] - 4s 65us/step - loss: 0.1211 - acc: 0.9641 - val_
Epoch 9/25
60000/60000 [==============================] - 4s 66us/step - loss: 0.1160 - acc: 0.9640 - val_
Epoch 10/25
60000/60000 [==============================] - 4s 65us/step - loss: 0.1099 - acc: 0.9664 - val_
```

```
Epoch 11/25
60000/60000 [==============================] - 4s 65us/step - loss: 0.0994 - acc: 0.9702 - val_
Epoch 12/25
60000/60000 [==============================] - 4s 65us/step - loss: 0.0975 - acc: 0.9701 - val_
Epoch 13/25
60000/60000 [==============================] - 4s 66us/step - loss: 0.0937 - acc: 0.9710 - val_
Epoch 14/25
60000/60000 [==============================] - 4s 65us/step - loss: 0.0894 - acc: 0.9720 - val_
Epoch 15/25
60000/60000 [==============================] - 4s 64us/step - loss: 0.0891 - acc: 0.9731 - val_
Epoch 16/25
60000/60000 [==============================] - 4s 64us/step - loss: 0.0855 - acc: 0.9730 - val_
Epoch 17/25
60000/60000 [==============================] - 4s 64us/step - loss: 0.0803 - acc: 0.9752 - val_
Epoch 18/25
60000/60000 [==============================] - 4s 64us/step - loss: 0.0812 - acc: 0.9755 - val_
Epoch 19/25
60000/60000 [==============================] - 4s 64us/step - loss: 0.0780 - acc: 0.9751 - val_
Epoch 20/25
60000/60000 [==============================] - 4s 64us/step - loss: 0.0754 - acc: 0.9772 - val_
Epoch 21/25
60000/60000 [==============================] - 4s 64us/step - loss: 0.0730 - acc: 0.9778 - val_
Epoch 22/25
60000/60000 [==============================] - 4s 64us/step - loss: 0.0707 - acc: 0.9781 - val_
Epoch 23/25
60000/60000 [==============================] - 4s 64us/step - loss: 0.0674 - acc: 0.9791 - val_
Epoch 24/25
60000/60000 [==============================] - 4s 64us/step - loss: 0.0701 - acc: 0.9785 - val_
Epoch 25/25
60000/60000 [==============================] - 4s 67us/step - loss: 0.0645 - acc: 0.9796 - val_
Test score: 0.058353246974284415
Test accuracy: 0.9828
```



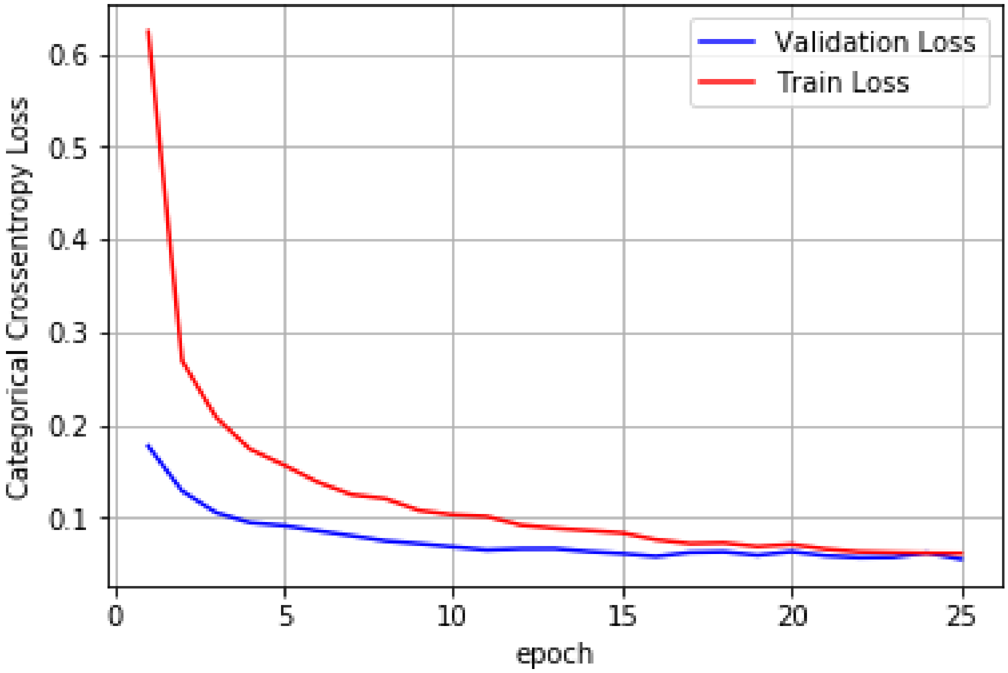
### 1.2 Architecture with 3 hidden layers.

```
In [79]: n_layers = [512,256,128]
         d = [0.5,0.6,0.5]
         nb_epoch = 25
         nn(n_layers,d)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_131 (Dense)            (None, 512)               401920
_________________________________________________________________
batch_normalization_87 (Batc (None, 512)               2048
_________________________________________________________________
dropout_67 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_132 (Dense)            (None, 256)               131328
_________________________________________________________________
batch_normalization_88 (Batc (None, 256)               1024
_________________________________________________________________
dropout_68 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_133 (Dense)            (None, 128)               32896
_________________________________________________________________
batch_normalization_89 (Batc (None, 128)               512
```

```
-----------------------------------------------------------------
dropout_69 (Dropout)         (None, 128)               0
-----------------------------------------------------------------
dense_134 (Dense)            (None, 10)                1290
=================================================================
Total params: 571,018
Trainable params: 569,226
Non-trainable params: 1,792
-----------------------------------------------------------------
Train on 60000 samples, validate on 10000 samples
Epoch 1/25
60000/60000 [==============================] - 14s 233us/step - loss: 0.6239 - acc: 0.8107 - va
Epoch 2/25
60000/60000 [==============================] - 5s 78us/step - loss: 0.2689 - acc: 0.9211 - val_
Epoch 3/25
60000/60000 [==============================] - 5s 77us/step - loss: 0.2078 - acc: 0.9396 - val_
Epoch 4/25
60000/60000 [==============================] - 5s 76us/step - loss: 0.1737 - acc: 0.9488 - val_
Epoch 5/25
60000/60000 [==============================] - 5s 77us/step - loss: 0.1565 - acc: 0.9537 - val_
Epoch 6/25
60000/60000 [==============================] - 5s 77us/step - loss: 0.1381 - acc: 0.9598 - val_
Epoch 7/25
60000/60000 [==============================] - 5s 79us/step - loss: 0.1246 - acc: 0.9625 - val_
Epoch 8/25
60000/60000 [==============================] - 5s 78us/step - loss: 0.1201 - acc: 0.9645 - val_
Epoch 9/25
60000/60000 [==============================] - 5s 78us/step - loss: 0.1074 - acc: 0.9685 - val_
Epoch 10/25
60000/60000 [==============================] - 5s 77us/step - loss: 0.1027 - acc: 0.9692 - val_
Epoch 11/25
60000/60000 [==============================] - 5s 79us/step - loss: 0.1010 - acc: 0.9700 - val_
Epoch 12/25
60000/60000 [==============================] - 5s 78us/step - loss: 0.0916 - acc: 0.9727 - val_
Epoch 13/25
60000/60000 [==============================] - 5s 80us/step - loss: 0.0882 - acc: 0.9732 - val_
Epoch 14/25
60000/60000 [==============================] - 5s 79us/step - loss: 0.0857 - acc: 0.9744 - val_
Epoch 15/25
60000/60000 [==============================] - 5s 82us/step - loss: 0.0832 - acc: 0.9750 - val_
Epoch 16/25
60000/60000 [==============================] - 5s 80us/step - loss: 0.0758 - acc: 0.9770 - val_
Epoch 17/25
60000/60000 [==============================] - 5s 80us/step - loss: 0.0717 - acc: 0.9786 - val_
Epoch 18/25
60000/60000 [==============================] - 5s 82us/step - loss: 0.0724 - acc: 0.9783 - val_
Epoch 19/25
60000/60000 [==============================] - 5s 81us/step - loss: 0.0684 - acc: 0.9798 - val_
```

```
Epoch 20/25
60000/60000 [==============================] - 5s 81us/step - loss: 0.0708 - acc: 0.9790 - val
Epoch 21/25
60000/60000 [==============================] - 5s 79us/step - loss: 0.0657 - acc: 0.9802 - val
Epoch 22/25
60000/60000 [==============================] - 5s 80us/step - loss: 0.0627 - acc: 0.9810 - val
Epoch 23/25
60000/60000 [==============================] - 5s 79us/step - loss: 0.0620 - acc: 0.9813 - val
Epoch 24/25
60000/60000 [==============================] - 5s 79us/step - loss: 0.0614 - acc: 0.9813 - val
Epoch 25/25
60000/60000 [==============================] - 5s 80us/step - loss: 0.0612 - acc: 0.9815 - val
Test score: 0.0548867928153486
Test accuracy: 0.985
```



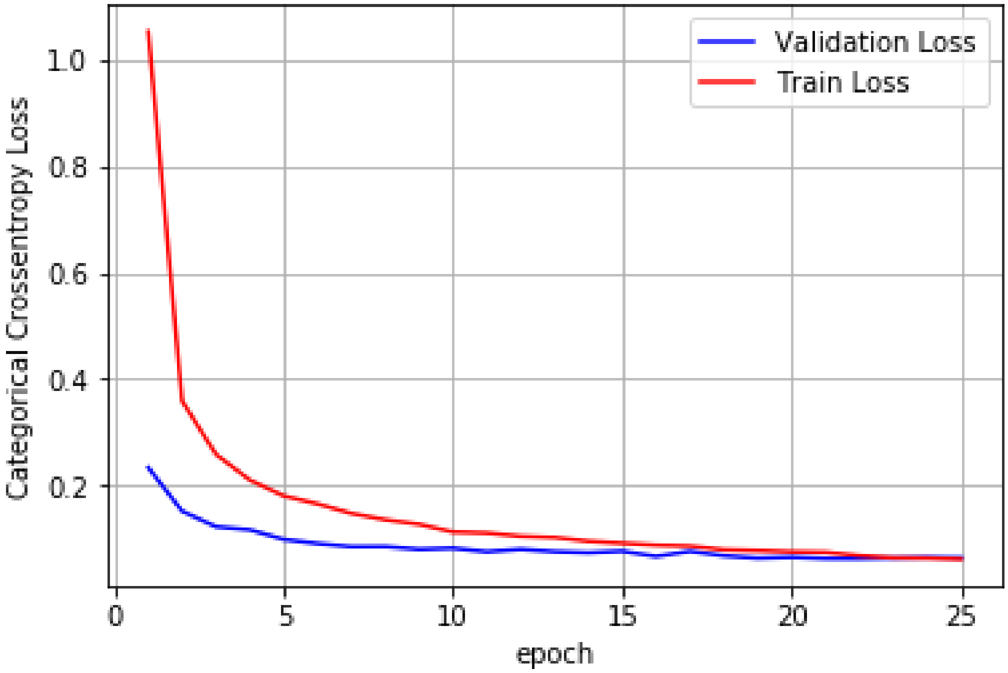
### 1.3 Architecture with 4 hidden layers( for experimentation)

```
In [56]: #Using 4 hidden layers for experimentation
         n_layers = [600,460,290,112]
         d = [0.4,0.7,0.5,0.7]
         nb_epoch = 25
         nn(n_layers,d)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

```
=================================================================
dense_56 (Dense)             (None, 600)               471000
_________________________________________________________________
batch_normalization_44 (Batc (None, 600)               2400
_________________________________________________________________
dropout_44 (Dropout)         (None, 600)               0
_________________________________________________________________
dense_57 (Dense)             (None, 460)               276460
_________________________________________________________________
batch_normalization_45 (Batc (None, 460)               1840
_________________________________________________________________
dropout_45 (Dropout)         (None, 460)               0
_________________________________________________________________
dense_58 (Dense)             (None, 290)               133690
_________________________________________________________________
batch_normalization_46 (Batc (None, 290)               1160
_________________________________________________________________
dropout_46 (Dropout)         (None, 290)               0
_________________________________________________________________
dense_59 (Dense)             (None, 112)               32592
_________________________________________________________________
batch_normalization_47 (Batc (None, 112)               448
_________________________________________________________________
dropout_47 (Dropout)         (None, 112)               0
_________________________________________________________________
dense_60 (Dense)             (None, 10)                1130
=================================================================
Total params: 920,720
Trainable params: 917,796
Non-trainable params: 2,924
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/25
60000/60000 [==============================] - 10s 175us/step - loss: 1.0535 - acc: 0.6908 - va
Epoch 2/25
60000/60000 [==============================] - 5s 91us/step - loss: 0.3588 - acc: 0.8999 - val_
Epoch 3/25
60000/60000 [==============================] - 5s 89us/step - loss: 0.2583 - acc: 0.9291 - val_
Epoch 4/25
60000/60000 [==============================] - 5s 90us/step - loss: 0.2100 - acc: 0.9431 - val_
Epoch 5/25
60000/60000 [==============================] - 5s 90us/step - loss: 0.1803 - acc: 0.9518 - val_
Epoch 6/25
60000/60000 [==============================] - 5s 91us/step - loss: 0.1652 - acc: 0.9559 - val_
Epoch 7/25
60000/60000 [==============================] - 5s 89us/step - loss: 0.1473 - acc: 0.9612 - val_
Epoch 8/25
60000/60000 [==============================] - 5s 90us/step - loss: 0.1353 - acc: 0.9637 - val_
```

```
Epoch 9/25
60000/60000 [==============================] - 5s 89us/step - loss: 0.1268 - acc: 0.9655 - val
Epoch 10/25
60000/60000 [==============================] - 5s 90us/step - loss: 0.1123 - acc: 0.9691 - val
Epoch 11/25
60000/60000 [==============================] - 5s 90us/step - loss: 0.1104 - acc: 0.9708 - val
Epoch 12/25
60000/60000 [==============================] - 5s 90us/step - loss: 0.1042 - acc: 0.9724 - val
Epoch 13/25
60000/60000 [==============================] - 5s 90us/step - loss: 0.1020 - acc: 0.9728 - val
Epoch 14/25
60000/60000 [==============================] - 5s 90us/step - loss: 0.0949 - acc: 0.9748 - val
Epoch 15/25
60000/60000 [==============================] - 5s 91us/step - loss: 0.0911 - acc: 0.9756 - val
Epoch 16/25
60000/60000 [==============================] - 6s 92us/step - loss: 0.0876 - acc: 0.9763 - val
Epoch 17/25
60000/60000 [==============================] - 5s 90us/step - loss: 0.0857 - acc: 0.9771 - val
Epoch 18/25
60000/60000 [==============================] - 5s 89us/step - loss: 0.0789 - acc: 0.9784 - val
Epoch 19/25
60000/60000 [==============================] - 5s 89us/step - loss: 0.0773 - acc: 0.9789 - val
Epoch 20/25
60000/60000 [==============================] - 5s 89us/step - loss: 0.0749 - acc: 0.9799 - val
Epoch 21/25
60000/60000 [==============================] - 5s 89us/step - loss: 0.0741 - acc: 0.9804 - val
Epoch 22/25
60000/60000 [==============================] - 5s 89us/step - loss: 0.0678 - acc: 0.9815 - val
Epoch 23/25
60000/60000 [==============================] - 5s 89us/step - loss: 0.0638 - acc: 0.9824 - val
Epoch 24/25
60000/60000 [==============================] - 5s 91us/step - loss: 0.0631 - acc: 0.9827 - val
Epoch 25/25
60000/60000 [==============================] - 6s 93us/step - loss: 0.0612 - acc: 0.9831 - val
Test score: 0.06384382131070597
Test accuracy: 0.9847
```



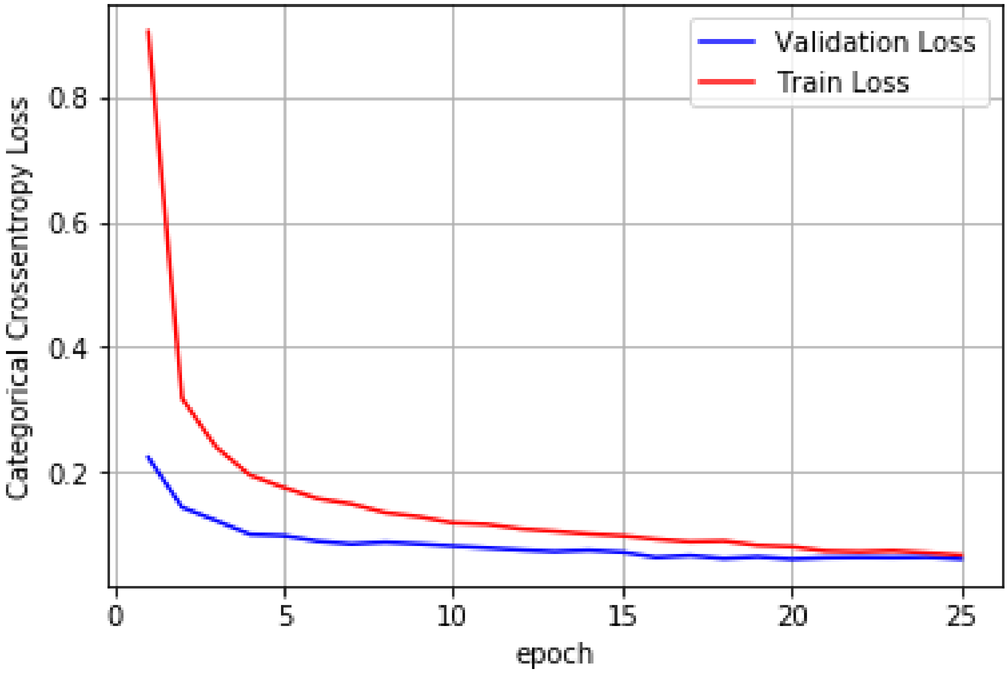
## 1.4 Architecture with 5 hidden layers.

```
In [59]: n_layers = [650,520,410,290,125]
         d = [0.5,0.6,0.5,0.5,0.5]
         nb_epoch = 25
         nn(n_layers,d)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_72 (Dense)             (None, 650)               510250
_________________________________________________________________
batch_normalization_58 (Batc (None, 650)               2600
_________________________________________________________________
dropout_57 (Dropout)         (None, 650)               0
_________________________________________________________________
dense_73 (Dense)             (None, 520)               338520
_________________________________________________________________
batch_normalization_59 (Batc (None, 520)               2080
_________________________________________________________________
dropout_58 (Dropout)         (None, 520)               0
_________________________________________________________________
dense_74 (Dense)             (None, 410)               213610
_________________________________________________________________
batch_normalization_60 (Batc (None, 410)               1640
```

```
_________________________________________________________________
dropout_59 (Dropout)         (None, 410)               0
_________________________________________________________________
dense_75 (Dense)             (None, 290)               119190
_________________________________________________________________
batch_normalization_61 (Batc (None, 290)               1160
_________________________________________________________________
dropout_60 (Dropout)         (None, 290)               0
_________________________________________________________________
dense_76 (Dense)             (None, 125)               36375
_________________________________________________________________
batch_normalization_62 (Batc (None, 125)               500
_________________________________________________________________
dropout_61 (Dropout)         (None, 125)               0
_________________________________________________________________
dense_77 (Dense)             (None, 10)                1260
=================================================================
Total params: 1,227,185
Trainable params: 1,223,195
Non-trainable params: 3,990
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/25
60000/60000 [==============================] - 13s 215us/step - loss: 0.9056 - acc: 0.7211 - va
Epoch 2/25
60000/60000 [==============================] - 6s 107us/step - loss: 0.3179 - acc: 0.9073 - val
Epoch 3/25
60000/60000 [==============================] - 6s 105us/step - loss: 0.2403 - acc: 0.9319 - val
Epoch 4/25
60000/60000 [==============================] - 6s 107us/step - loss: 0.1951 - acc: 0.9447 - val
Epoch 5/25
60000/60000 [==============================] - 7s 110us/step - loss: 0.1753 - acc: 0.9509 - val
Epoch 6/25
60000/60000 [==============================] - 6s 107us/step - loss: 0.1575 - acc: 0.9554 - val
Epoch 7/25
60000/60000 [==============================] - 6s 106us/step - loss: 0.1490 - acc: 0.9583 - val
Epoch 8/25
60000/60000 [==============================] - 6s 105us/step - loss: 0.1349 - acc: 0.9621 - val
Epoch 9/25
60000/60000 [==============================] - 6s 106us/step - loss: 0.1284 - acc: 0.9640 - val
Epoch 10/25
60000/60000 [==============================] - 6s 105us/step - loss: 0.1186 - acc: 0.9668 - val
Epoch 11/25
60000/60000 [==============================] - 6s 107us/step - loss: 0.1163 - acc: 0.9672 - val
Epoch 12/25
60000/60000 [==============================] - 6s 106us/step - loss: 0.1090 - acc: 0.9697 - val
Epoch 13/25
60000/60000 [==============================] - 6s 105us/step - loss: 0.1050 - acc: 0.9705 - val
```

```
Epoch 14/25
60000/60000 [==============================] - 6s 106us/step - loss: 0.1008 - acc: 0.9720 - val
Epoch 15/25
60000/60000 [==============================] - 6s 107us/step - loss: 0.0978 - acc: 0.9729 - val
Epoch 16/25
60000/60000 [==============================] - 7s 116us/step - loss: 0.0923 - acc: 0.9743 - val
Epoch 17/25
60000/60000 [==============================] - 7s 111us/step - loss: 0.0884 - acc: 0.9753 - val
Epoch 18/25
60000/60000 [==============================] - 7s 110us/step - loss: 0.0896 - acc: 0.9744 - val
Epoch 19/25
60000/60000 [==============================] - 6s 108us/step - loss: 0.0825 - acc: 0.9768 - val
Epoch 20/25
60000/60000 [==============================] - 6s 108us/step - loss: 0.0803 - acc: 0.9782 - val
Epoch 21/25
60000/60000 [==============================] - 6s 107us/step - loss: 0.0735 - acc: 0.9790 - val
Epoch 22/25
60000/60000 [==============================] - 6s 107us/step - loss: 0.0722 - acc: 0.9789 - val
Epoch 23/25
60000/60000 [==============================] - 6s 108us/step - loss: 0.0738 - acc: 0.9789 - val
Epoch 24/25
60000/60000 [==============================] - 6s 108us/step - loss: 0.0706 - acc: 0.9795 - val
Epoch 25/25
60000/60000 [==============================] - 6s 107us/step - loss: 0.0669 - acc: 0.9807 - val
Test score: 0.06073720659725368
Test accuracy: 0.9846
```



## 2 Defining a function for model with only batch normalisation and no dropout.

```
In [0]: # https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormalization-


        def nn1(n_layers,bn):
          model = Sequential()
          model.add(Dense(n_layers[0], activation='relu', input_shape=(input_dim,), kernel_ini
          if(bn==1):
            model.add(BatchNormalization())
            for i in range(len(n_layers)-1):
              model.add(Dense(n_layers[i+1], activation='relu',kernel_initializer=K.he_normal(
              model.add(BatchNormalization())
          else:
            for i in range(len(n_layers)-1):
              model.add(Dense(n_layers[i+1], activation='relu',kernel_initializer=K.he_normal(


          model.add(Dense(output_dim, activation='softmax'))


          model.summary()


          model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'

          history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbos


          %matplotlib inline
          score = model.evaluate(X_test, Y_test, verbose=0)
          print('Test score:', score[0])
          print('Test accuracy:', score[1])

          fig,ax = plt.subplots(1,1)
          ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

          # list of epoch numbers
          x = list(range(1,nb_epoch+1))

           #dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
          # history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,
```

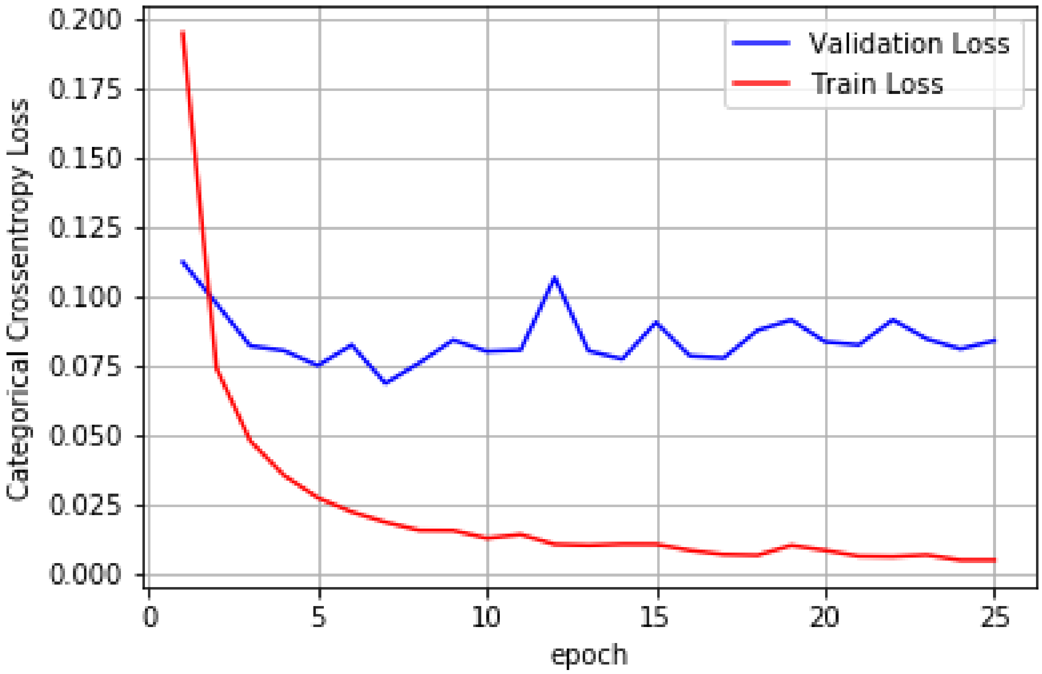
```
# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

### 2.0.1 Architecture with 2 hidden layers.

```
In [70]: n_layers = [468,92]
         bn = 1
         nb_epoch = 25
         nn1(n_layers,bn)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_102 (Dense)            (None, 468)               367380
_________________________________________________________________
batch_normalization_75 (Batc (None, 468)               1872
_________________________________________________________________
dense_103 (Dense)            (None, 92)                43148
_________________________________________________________________
batch_normalization_76 (Batc (None, 92)                368
_________________________________________________________________
dense_104 (Dense)            (None, 10)                930
=================================================================
Total params: 413,698
Trainable params: 412,578
Non-trainable params: 1,120
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/25
60000/60000 [==============================] - 11s 179us/step - loss: 0.1946 - acc: 0.9427 - va
Epoch 2/25
60000/60000 [==============================] - 4s 62us/step - loss: 0.0741 - acc: 0.9776 - val_
Epoch 3/25
60000/60000 [==============================] - 4s 63us/step - loss: 0.0479 - acc: 0.9853 - val_
Epoch 4/25
60000/60000 [==============================] - 4s 63us/step - loss: 0.0355 - acc: 0.9889 - val_
Epoch 5/25
60000/60000 [==============================] - 4s 63us/step - loss: 0.0275 - acc: 0.9914 - val_
Epoch 6/25
60000/60000 [==============================] - 4s 64us/step - loss: 0.0224 - acc: 0.9926 - val_
```

```
Epoch 7/25
60000/60000 [==============================] - 4s 64us/step - loss: 0.0187 - acc: 0.9942 - val_
Epoch 8/25
60000/60000 [==============================] - 4s 64us/step - loss: 0.0158 - acc: 0.9947 - val_
Epoch 9/25
60000/60000 [==============================] - 4s 66us/step - loss: 0.0157 - acc: 0.9948 - val_
Epoch 10/25
60000/60000 [==============================] - 4s 63us/step - loss: 0.0129 - acc: 0.9959 - val_
Epoch 11/25
60000/60000 [==============================] - 4s 62us/step - loss: 0.0142 - acc: 0.9951 - val_
Epoch 12/25
60000/60000 [==============================] - 4s 64us/step - loss: 0.0107 - acc: 0.9965 - val_
Epoch 13/25
60000/60000 [==============================] - 4s 63us/step - loss: 0.0104 - acc: 0.9966 - val_
Epoch 14/25
60000/60000 [==============================] - 4s 62us/step - loss: 0.0108 - acc: 0.9963 - val_
Epoch 15/25
60000/60000 [==============================] - 4s 63us/step - loss: 0.0107 - acc: 0.9963 - val_
Epoch 16/25
60000/60000 [==============================] - 4s 61us/step - loss: 0.0085 - acc: 0.9971 - val_
Epoch 17/25
60000/60000 [==============================] - 4s 63us/step - loss: 0.0071 - acc: 0.9977 - val_
Epoch 18/25
60000/60000 [==============================] - 4s 62us/step - loss: 0.0068 - acc: 0.9978 - val_
Epoch 19/25
60000/60000 [==============================] - 4s 62us/step - loss: 0.0103 - acc: 0.9966 - val_
Epoch 20/25
60000/60000 [==============================] - 4s 62us/step - loss: 0.0085 - acc: 0.9971 - val_
Epoch 21/25
60000/60000 [==============================] - 4s 62us/step - loss: 0.0064 - acc: 0.9978 - val_
Epoch 22/25
60000/60000 [==============================] - 4s 62us/step - loss: 0.0063 - acc: 0.9981 - val_
Epoch 23/25
60000/60000 [==============================] - 4s 63us/step - loss: 0.0069 - acc: 0.9978 - val_
Epoch 24/25
60000/60000 [==============================] - 4s 62us/step - loss: 0.0050 - acc: 0.9983 - val_
Epoch 25/25
60000/60000 [==============================] - 4s 62us/step - loss: 0.0050 - acc: 0.9984 - val_
Test score: 0.08396656601358045
Test accuracy: 0.9808
```



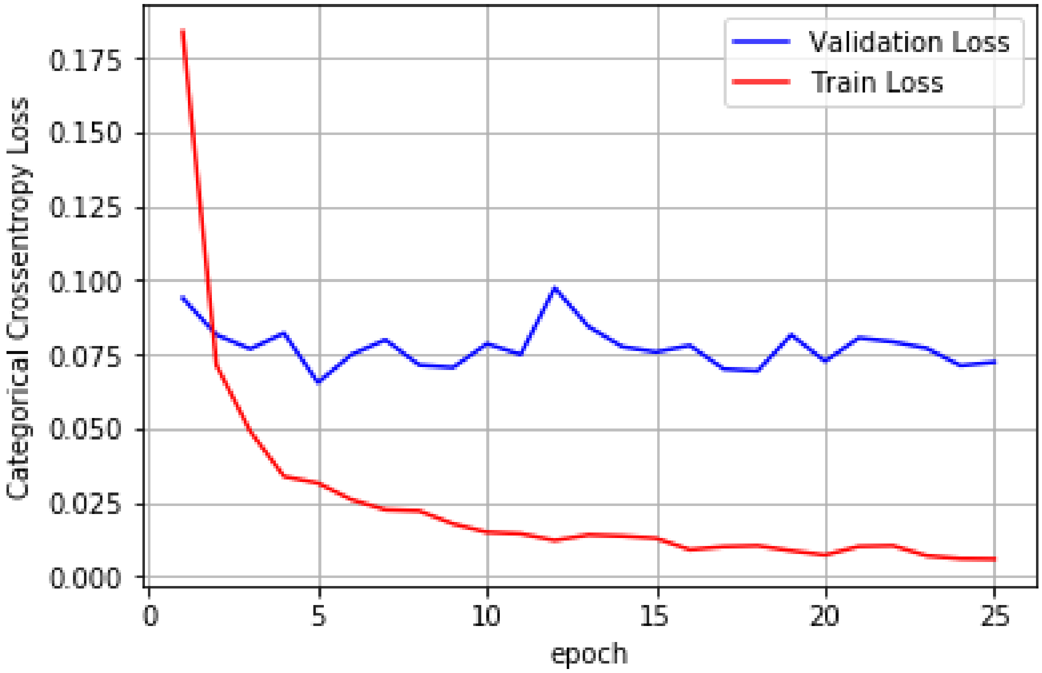
## 2.1 Architecture with 3 hidden layers.

```
In [71]: n_layers = [512,256,128]
         bn=1
         nb_epoch = 25
         nn1(n_layers,bn)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_105 (Dense)            (None, 512)               401920    
_________________________________________________________________
batch_normalization_77 (Batc (None, 512)               2048      
_________________________________________________________________
dense_106 (Dense)            (None, 256)               131328    
_________________________________________________________________
batch_normalization_78 (Batc (None, 256)               1024      
_________________________________________________________________
dense_107 (Dense)            (None, 128)               32896     
_________________________________________________________________
batch_normalization_79 (Batc (None, 128)               512       
_________________________________________________________________
dense_108 (Dense)            (None, 10)                1290      
=================================================================
Total params: 571,018
```

```
Trainable params: 569,226
Non-trainable params: 1,792
-----------------------------------------------------------------
Train on 60000 samples, validate on 10000 samples
Epoch 1/25
60000/60000 [==============================] - 12s 206us/step - loss: 0.1840 - acc: 0.9448 - va
Epoch 2/25
60000/60000 [==============================] - 5s 75us/step - loss: 0.0713 - acc: 0.9782 - val_
Epoch 3/25
60000/60000 [==============================] - 5s 76us/step - loss: 0.0490 - acc: 0.9843 - val_
Epoch 4/25
60000/60000 [==============================] - 5s 78us/step - loss: 0.0337 - acc: 0.9889 - val_
Epoch 5/25
60000/60000 [==============================] - 5s 78us/step - loss: 0.0315 - acc: 0.9896 - val_
Epoch 6/25
60000/60000 [==============================] - 5s 75us/step - loss: 0.0259 - acc: 0.9912 - val_
Epoch 7/25
60000/60000 [==============================] - 4s 74us/step - loss: 0.0225 - acc: 0.9926 - val_
Epoch 8/25
60000/60000 [==============================] - 4s 74us/step - loss: 0.0221 - acc: 0.9928 - val_
Epoch 9/25
60000/60000 [==============================] - 4s 74us/step - loss: 0.0177 - acc: 0.9940 - val_
Epoch 10/25
60000/60000 [==============================] - 4s 74us/step - loss: 0.0149 - acc: 0.9950 - val_
Epoch 11/25
60000/60000 [==============================] - 5s 75us/step - loss: 0.0144 - acc: 0.9955 - val_
Epoch 12/25
60000/60000 [==============================] - 4s 74us/step - loss: 0.0121 - acc: 0.9963 - val_
Epoch 13/25
60000/60000 [==============================] - 4s 75us/step - loss: 0.0140 - acc: 0.9953 - val_
Epoch 14/25
60000/60000 [==============================] - 4s 74us/step - loss: 0.0136 - acc: 0.9956 - val_
Epoch 15/25
60000/60000 [==============================] - 5s 76us/step - loss: 0.0128 - acc: 0.9957 - val_
Epoch 16/25
60000/60000 [==============================] - 5s 79us/step - loss: 0.0090 - acc: 0.9970 - val_
Epoch 17/25
60000/60000 [==============================] - 4s 74us/step - loss: 0.0100 - acc: 0.9965 - val_
Epoch 18/25
60000/60000 [==============================] - 5s 76us/step - loss: 0.0103 - acc: 0.9969 - val_
Epoch 19/25
60000/60000 [==============================] - 5s 76us/step - loss: 0.0086 - acc: 0.9973 - val_
Epoch 20/25
60000/60000 [==============================] - 5s 77us/step - loss: 0.0072 - acc: 0.9977 - val_
Epoch 21/25
60000/60000 [==============================] - 4s 75us/step - loss: 0.0101 - acc: 0.9962 - val_
Epoch 22/25
60000/60000 [==============================] - 4s 74us/step - loss: 0.0103 - acc: 0.9967 - val_
```

```
Epoch 23/25
60000/60000 [==============================] - 4s 75us/step - loss: 0.0069 - acc: 0.9977 - val_
Epoch 24/25
60000/60000 [==============================] - 4s 75us/step - loss: 0.0060 - acc: 0.9981 - val_
Epoch 25/25
60000/60000 [==============================] - 4s 75us/step - loss: 0.0058 - acc: 0.9981 - val_
Test score: 0.07232702655499207
Test accuracy: 0.9833
```



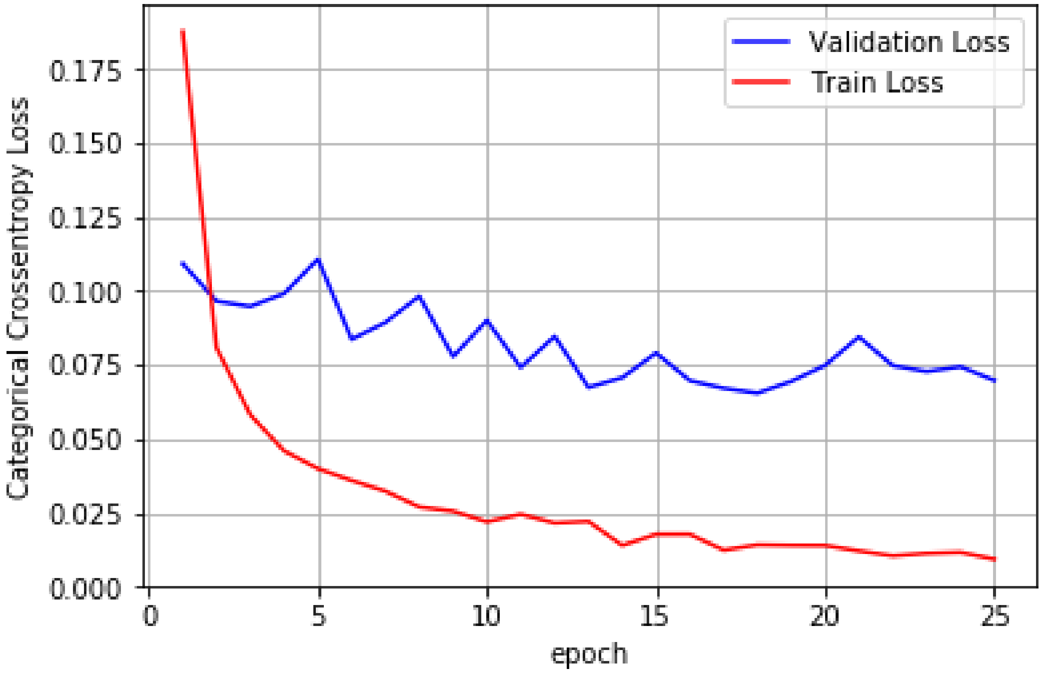
## 2.2 Architecture with 5 hidden layers.

```
In [72]: n_layers = [650,520,410,290,125]
         bn=1
         nb_epoch = 25
         nn1(n_layers,bn)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_109 (Dense)            (None, 650)               510250    
_________________________________________________________________
batch_normalization_80 (Batc (None, 650)               2600      
_________________________________________________________________
dense_110 (Dense)            (None, 520)               338520    
```

```
_________________________________________________________________
batch_normalization_81 (Batc (None, 520)              2080
_________________________________________________________________
dense_111 (Dense)            (None, 410)              213610
_________________________________________________________________
batch_normalization_82 (Batc (None, 410)              1640
_________________________________________________________________
dense_112 (Dense)            (None, 290)              119190
_________________________________________________________________
batch_normalization_83 (Batc (None, 290)              1160
_________________________________________________________________
dense_113 (Dense)            (None, 125)              36375
_________________________________________________________________
batch_normalization_84 (Batc (None, 125)              500
_________________________________________________________________
dense_114 (Dense)            (None, 10)               1260
=================================================================
Total params: 1,227,185
Trainable params: 1,223,195
Non-trainable params: 3,990
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/25
60000/60000 [==============================] - 15s 242us/step - loss: 0.1877 - acc: 0.9439 - va
Epoch 2/25
60000/60000 [==============================] - 6s 102us/step - loss: 0.0808 - acc: 0.9746 - val
Epoch 3/25
60000/60000 [==============================] - 6s 100us/step - loss: 0.0581 - acc: 0.9813 - val
Epoch 4/25
60000/60000 [==============================] - 6s 103us/step - loss: 0.0460 - acc: 0.9852 - val
Epoch 5/25
60000/60000 [==============================] - 6s 103us/step - loss: 0.0400 - acc: 0.9869 - val
Epoch 6/25
60000/60000 [==============================] - 6s 104us/step - loss: 0.0360 - acc: 0.9882 - val
Epoch 7/25
60000/60000 [==============================] - 6s 104us/step - loss: 0.0323 - acc: 0.9894 - val
Epoch 8/25
60000/60000 [==============================] - 6s 104us/step - loss: 0.0270 - acc: 0.9909 - val
Epoch 9/25
60000/60000 [==============================] - 6s 102us/step - loss: 0.0256 - acc: 0.9915 - val
Epoch 10/25
60000/60000 [==============================] - 6s 102us/step - loss: 0.0220 - acc: 0.9927 - val
Epoch 11/25
60000/60000 [==============================] - 6s 103us/step - loss: 0.0245 - acc: 0.9918 - val
Epoch 12/25
60000/60000 [==============================] - 6s 102us/step - loss: 0.0215 - acc: 0.9928 - val
Epoch 13/25
60000/60000 [==============================] - 6s 102us/step - loss: 0.0221 - acc: 0.9928 - val
```

```
Epoch 14/25
60000/60000 [==============================] - 6s 101us/step - loss: 0.0140 - acc: 0.9954 - val
Epoch 15/25
60000/60000 [==============================] - 6s 101us/step - loss: 0.0177 - acc: 0.9939 - val
Epoch 16/25
60000/60000 [==============================] - 6s 101us/step - loss: 0.0178 - acc: 0.9941 - val
Epoch 17/25
60000/60000 [==============================] - 6s 102us/step - loss: 0.0124 - acc: 0.9960 - val
Epoch 18/25
60000/60000 [==============================] - 6s 102us/step - loss: 0.0142 - acc: 0.9955 - val
Epoch 19/25
60000/60000 [==============================] - 6s 102us/step - loss: 0.0140 - acc: 0.9953 - val
Epoch 20/25
60000/60000 [==============================] - 6s 103us/step - loss: 0.0139 - acc: 0.9953 - val
Epoch 21/25
60000/60000 [==============================] - 6s 104us/step - loss: 0.0121 - acc: 0.9957 - val
Epoch 22/25
60000/60000 [==============================] - 6s 100us/step - loss: 0.0104 - acc: 0.9966 - val
Epoch 23/25
60000/60000 [==============================] - 6s 101us/step - loss: 0.0113 - acc: 0.9963 - val
Epoch 24/25
60000/60000 [==============================] - 6s 99us/step - loss: 0.0116 - acc: 0.9962 - val
Epoch 25/25
60000/60000 [==============================] - 6s 101us/step - loss: 0.0093 - acc: 0.9972 - val
Test score: 0.06976913923782413
Test accuracy: 0.984
```



## 3 Defining a function for model with no dropout and no batch normalisation.

```
In [0]: # https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormalization-


        def nn2(n_layers):
          model = Sequential()
          model.add(Dense(n_layers[0], activation='relu', input_shape=(input_dim,), kernel_ini
          for i in range(len(n_layers)-1):
              model.add(Dense(n_layers[i+1], activation='relu',kernel_initializer=K.he_normal(


          model.add(Dense(output_dim, activation='softmax'))


          model.summary()


          model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'

          history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbos


          %matplotlib inline
          score = model.evaluate(X_test, Y_test, verbose=0)
          print('Test score:', score[0])
          print('Test accuracy:', score[1])

          fig,ax = plt.subplots(1,1)
          ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

          # list of epoch numbers
          x = list(range(1,nb_epoch+1))

           #dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
          # history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,

          # we will get val_loss and val_acc only when you pass the paramter validation_data
          # val_loss : validation loss
          # val_acc : validation accuracy

          # loss : training loss
          # acc : train accuracy
```

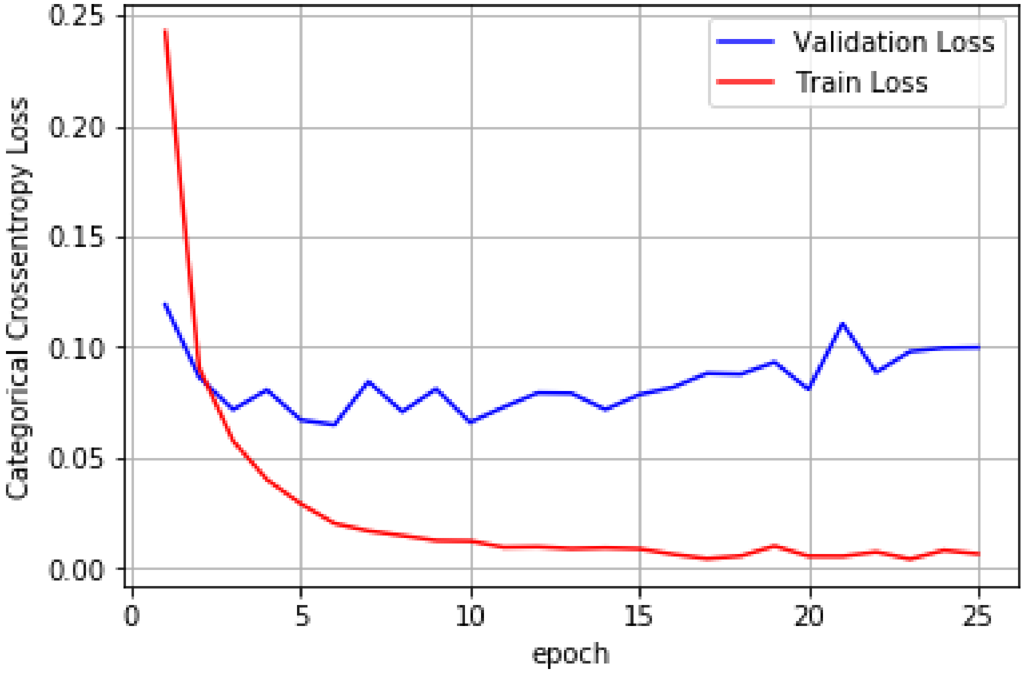
```
    # for each key in histrory.histrory we will have a list of length equal to number of
    vy = history.history['val_loss']
    ty = history.history['loss']
    plt_dynamic(x, vy, ty, ax)
```

### 3.0.1 Architecture with 2 hidden layers.

```
In [75]: n_layers = [468,92]
         nb_epoch = 25
         nn2(n_layers)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_115 (Dense)            (None, 468)               367380
_________________________________________________________________
dense_116 (Dense)            (None, 92)                43148
_________________________________________________________________
dense_117 (Dense)            (None, 10)                930
=================================================================
Total params: 411,458
Trainable params: 411,458
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/25
60000/60000 [==============================] - 10s 170us/step - loss: 0.2425 - acc: 0.9295 - va
Epoch 2/25
60000/60000 [==============================] - 3s 42us/step - loss: 0.0907 - acc: 0.9724 - val_
Epoch 3/25
60000/60000 [==============================] - 3s 42us/step - loss: 0.0576 - acc: 0.9822 - val_
Epoch 4/25
60000/60000 [==============================] - 3s 42us/step - loss: 0.0402 - acc: 0.9878 - val_
Epoch 5/25
60000/60000 [==============================] - 3s 42us/step - loss: 0.0292 - acc: 0.9905 - val_
Epoch 6/25
60000/60000 [==============================] - 3s 43us/step - loss: 0.0203 - acc: 0.9936 - val_
Epoch 7/25
60000/60000 [==============================] - 3s 42us/step - loss: 0.0169 - acc: 0.9944 - val_
Epoch 8/25
60000/60000 [==============================] - 3s 42us/step - loss: 0.0148 - acc: 0.9951 - val_
Epoch 9/25
60000/60000 [==============================] - 3s 42us/step - loss: 0.0126 - acc: 0.9960 - val_
Epoch 10/25
60000/60000 [==============================] - 3s 42us/step - loss: 0.0124 - acc: 0.9958 - val_
Epoch 11/25
60000/60000 [==============================] - 3s 44us/step - loss: 0.0095 - acc: 0.9969 - val_
Epoch 12/25
```

```
60000/60000 [==============================] - 3s 44us/step - loss: 0.0097 - acc: 0.9968 - val_
Epoch 13/25
60000/60000 [==============================] - 3s 44us/step - loss: 0.0089 - acc: 0.9971 - val_
Epoch 14/25
60000/60000 [==============================] - 3s 44us/step - loss: 0.0093 - acc: 0.9969 - val_
Epoch 15/25
60000/60000 [==============================] - 3s 42us/step - loss: 0.0089 - acc: 0.9971 - val_
Epoch 16/25
60000/60000 [==============================] - 3s 43us/step - loss: 0.0064 - acc: 0.9980 - val_
Epoch 17/25
60000/60000 [==============================] - 3s 42us/step - loss: 0.0044 - acc: 0.9986 - val_
Epoch 18/25
60000/60000 [==============================] - 2s 41us/step - loss: 0.0056 - acc: 0.9981 - val_
Epoch 19/25
60000/60000 [==============================] - 3s 42us/step - loss: 0.0101 - acc: 0.9967 - val_
Epoch 20/25
60000/60000 [==============================] - 3s 42us/step - loss: 0.0055 - acc: 0.9979 - val_
Epoch 21/25
60000/60000 [==============================] - 3s 42us/step - loss: 0.0055 - acc: 0.9981 - val_
Epoch 22/25
60000/60000 [==============================] - 3s 42us/step - loss: 0.0074 - acc: 0.9978 - val_
Epoch 23/25
60000/60000 [==============================] - 2s 42us/step - loss: 0.0042 - acc: 0.9986 - val_
Epoch 24/25
60000/60000 [==============================] - 3s 42us/step - loss: 0.0082 - acc: 0.9974 - val_
Epoch 25/25
60000/60000 [==============================] - 3s 42us/step - loss: 0.0065 - acc: 0.9978 - val_
Test score: 0.09983180044603286
Test accuracy: 0.9807
```



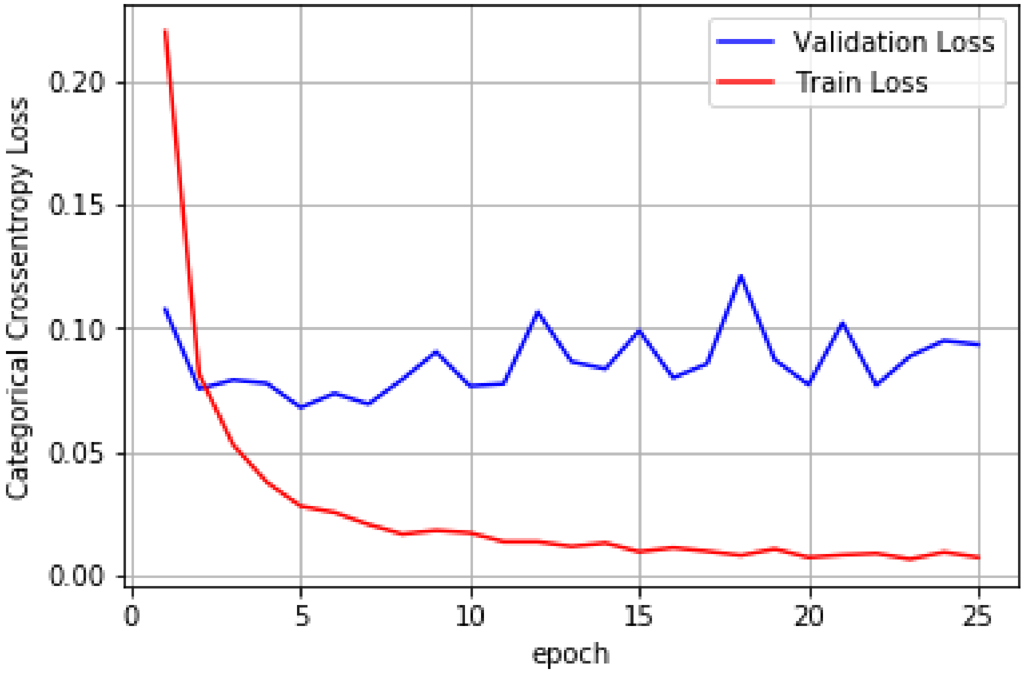
### 3.1 Architecture with 3 hidden layers

```
In [76]: n_layers = [512,256,128]
         nb_epoch = 25
         nn2(n_layers)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_118 (Dense)            (None, 512)               401920
_________________________________________________________________
dense_119 (Dense)            (None, 256)               131328
_________________________________________________________________
dense_120 (Dense)            (None, 128)               32896
_________________________________________________________________
dense_121 (Dense)            (None, 10)                1290
=================================================================
Total params: 567,434
Trainable params: 567,434
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/25
60000/60000 [==============================] - 11s 180us/step - loss: 0.2199 - acc: 0.9339 - va
Epoch 2/25
```

```
60000/60000 [==============================] - 3s 46us/step - loss: 0.0813 - acc: 0.9754 - val_
Epoch 3/25
60000/60000 [==============================] - 3s 45us/step - loss: 0.0528 - acc: 0.9829 - val_
Epoch 4/25
60000/60000 [==============================] - 3s 44us/step - loss: 0.0376 - acc: 0.9877 - val_
Epoch 5/25
60000/60000 [==============================] - 3s 45us/step - loss: 0.0279 - acc: 0.9910 - val_
Epoch 6/25
60000/60000 [==============================] - 3s 45us/step - loss: 0.0254 - acc: 0.9915 - val_
Epoch 7/25
60000/60000 [==============================] - 3s 45us/step - loss: 0.0205 - acc: 0.9932 - val_
Epoch 8/25
60000/60000 [==============================] - 3s 45us/step - loss: 0.0166 - acc: 0.9945 - val_
Epoch 9/25
60000/60000 [==============================] - 3s 45us/step - loss: 0.0181 - acc: 0.9941 - val_
Epoch 10/25
60000/60000 [==============================] - 3s 45us/step - loss: 0.0171 - acc: 0.9942 - val_
Epoch 11/25
60000/60000 [==============================] - 3s 46us/step - loss: 0.0135 - acc: 0.9956 - val_
Epoch 12/25
60000/60000 [==============================] - 3s 46us/step - loss: 0.0135 - acc: 0.9957 - val_
Epoch 13/25
60000/60000 [==============================] - 3s 47us/step - loss: 0.0116 - acc: 0.9965 - val_
Epoch 14/25
60000/60000 [==============================] - 3s 47us/step - loss: 0.0130 - acc: 0.9957 - val_
Epoch 15/25
60000/60000 [==============================] - 3s 46us/step - loss: 0.0095 - acc: 0.9970 - val_
Epoch 16/25
60000/60000 [==============================] - 3s 45us/step - loss: 0.0111 - acc: 0.9965 - val_
Epoch 17/25
60000/60000 [==============================] - 3s 45us/step - loss: 0.0097 - acc: 0.9970 - val_
Epoch 18/25
60000/60000 [==============================] - 3s 45us/step - loss: 0.0080 - acc: 0.9974 - val_
Epoch 19/25
60000/60000 [==============================] - 3s 45us/step - loss: 0.0106 - acc: 0.9968 - val_
Epoch 20/25
60000/60000 [==============================] - 3s 45us/step - loss: 0.0072 - acc: 0.9980 - val_
Epoch 21/25
60000/60000 [==============================] - 3s 45us/step - loss: 0.0082 - acc: 0.9976 - val_
Epoch 22/25
60000/60000 [==============================] - 3s 45us/step - loss: 0.0088 - acc: 0.9974 - val_
Epoch 23/25
60000/60000 [==============================] - 3s 45us/step - loss: 0.0065 - acc: 0.9981 - val_
Epoch 24/25
60000/60000 [==============================] - 3s 45us/step - loss: 0.0093 - acc: 0.9973 - val_
Epoch 25/25
60000/60000 [==============================] - 3s 45us/step - loss: 0.0072 - acc: 0.9979 - val_
Test score: 0.0932893890325599
```

```
Test accuracy: 0.9814
```



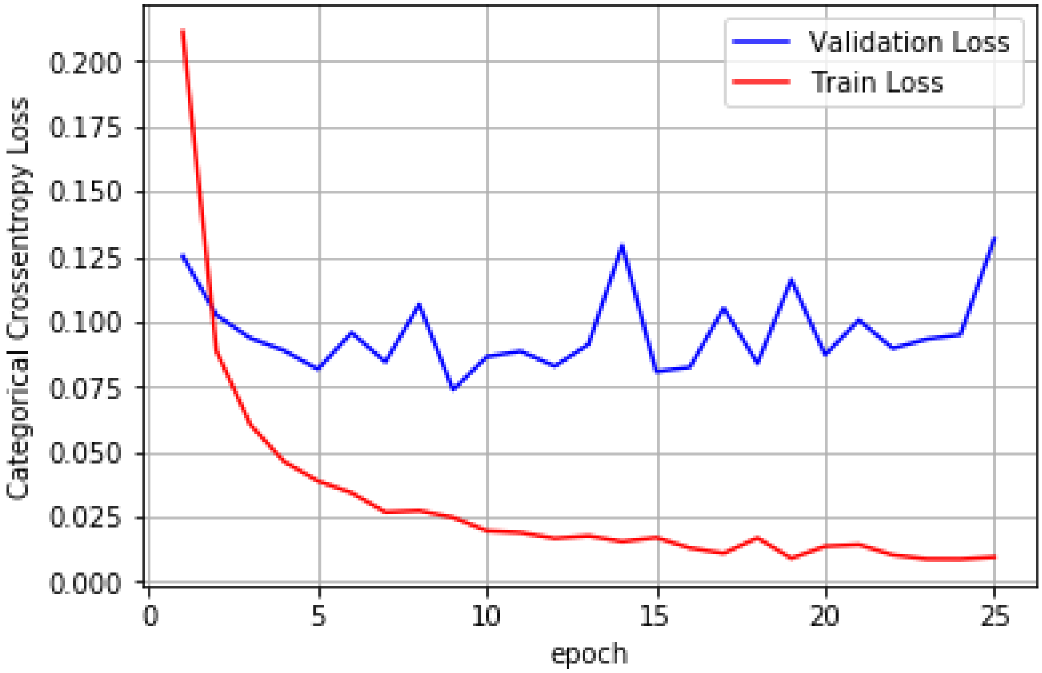
### 3.2 Architecture with 5 hidden layers.

```
In [77]: n_layers = [650,520,410,290,125]
         nb_epoch = 25
         nn2(n_layers)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_122 (Dense)            (None, 650)               510250
_________________________________________________________________
dense_123 (Dense)            (None, 520)               338520
_________________________________________________________________
dense_124 (Dense)            (None, 410)               213610
_________________________________________________________________
dense_125 (Dense)            (None, 290)               119190
_________________________________________________________________
dense_126 (Dense)            (None, 125)               36375
_________________________________________________________________
dense_127 (Dense)            (None, 10)                1260
=================================================================
Total params: 1,219,205
```

```
Trainable params: 1,219,205
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/25
60000/60000 [==============================] - 11s 190us/step - loss: 0.2114 - acc: 0.9360 - va
Epoch 2/25
60000/60000 [==============================] - 3s 53us/step - loss: 0.0887 - acc: 0.9726 - val_
Epoch 3/25
60000/60000 [==============================] - 3s 53us/step - loss: 0.0605 - acc: 0.9813 - val_
Epoch 4/25
60000/60000 [==============================] - 3s 52us/step - loss: 0.0463 - acc: 0.9858 - val_
Epoch 5/25
60000/60000 [==============================] - 3s 52us/step - loss: 0.0388 - acc: 0.9880 - val_
Epoch 6/25
60000/60000 [==============================] - 3s 53us/step - loss: 0.0342 - acc: 0.9895 - val_
Epoch 7/25
60000/60000 [==============================] - 3s 52us/step - loss: 0.0268 - acc: 0.9914 - val_
Epoch 8/25
60000/60000 [==============================] - 3s 53us/step - loss: 0.0274 - acc: 0.9916 - val_
Epoch 9/25
60000/60000 [==============================] - 3s 52us/step - loss: 0.0248 - acc: 0.9923 - val_
Epoch 10/25
60000/60000 [==============================] - 3s 52us/step - loss: 0.0196 - acc: 0.9942 - val_
Epoch 11/25
60000/60000 [==============================] - 3s 52us/step - loss: 0.0189 - acc: 0.9944 - val_
Epoch 12/25
60000/60000 [==============================] - 3s 54us/step - loss: 0.0167 - acc: 0.9952 - val_
Epoch 13/25
60000/60000 [==============================] - 3s 53us/step - loss: 0.0177 - acc: 0.9949 - val_
Epoch 14/25
60000/60000 [==============================] - 3s 53us/step - loss: 0.0155 - acc: 0.9952 - val_
Epoch 15/25
60000/60000 [==============================] - 3s 54us/step - loss: 0.0170 - acc: 0.9951 - val_
Epoch 16/25
60000/60000 [==============================] - 3s 52us/step - loss: 0.0130 - acc: 0.9966 - val_
Epoch 17/25
60000/60000 [==============================] - 3s 53us/step - loss: 0.0110 - acc: 0.9970 - val_
Epoch 18/25
60000/60000 [==============================] - 3s 53us/step - loss: 0.0169 - acc: 0.9951 - val_
Epoch 19/25
60000/60000 [==============================] - 3s 52us/step - loss: 0.0090 - acc: 0.9973 - val_
Epoch 20/25
60000/60000 [==============================] - 3s 52us/step - loss: 0.0137 - acc: 0.9965 - val_
Epoch 21/25
60000/60000 [==============================] - 3s 52us/step - loss: 0.0143 - acc: 0.9960 - val_
Epoch 22/25
60000/60000 [==============================] - 3s 52us/step - loss: 0.0103 - acc: 0.9974 - val_
```

```
Epoch 23/25
60000/60000 [==============================] - 3s 52us/step - loss: 0.0088 - acc: 0.9975 - val_
Epoch 24/25
60000/60000 [==============================] - 3s 52us/step - loss: 0.0088 - acc: 0.9976 - val_
Epoch 25/25
60000/60000 [==============================] - 3s 53us/step - loss: 0.0095 - acc: 0.9972 - val_
Test score: 0.13170449254391386
Test accuracy: 0.9788
```



## 4 Defining a function for model with dropout and no batch normalisation.

```
In [0]: # https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormalization-


        def nn3(n_layers,d):
          model = Sequential()

          model.add(Dense(n_layers[0], activation='relu', input_shape=(input_dim,), kernel_ini
          model.add(Dropout(d[0]))

          for i in range(len(n_layers)-1):
            model.add(Dense(n_layers[i+1], activation='relu',kernel_initializer=K.he_normal(se
```

```
      model.add(Dropout(d[i+1]))

    model.add(Dense(output_dim, activation='softmax'))


    model.summary()


    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy']

    history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbos


    %matplotlib inline
    score = model.evaluate(X_test, Y_test, verbose=0)
    print('Test score:', score[0])
    print('Test accuracy:', score[1])

    fig,ax = plt.subplots(1,1)
    ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

    # list of epoch numbers
    x = list(range(1,nb_epoch+1))

     #dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
    # history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,

    # we will get val_loss and val_acc only when you pass the paramter validation_data
    # val_loss : validation loss
    # val_acc : validation accuracy

    # loss : training loss
    # acc : train accuracy
    # for each key in histrory.histrory we will have a list of length equal to number of
    vy = history.history['val_loss']
    ty = history.history['loss']
    plt_dynamic(x, vy, ty, ax)
```

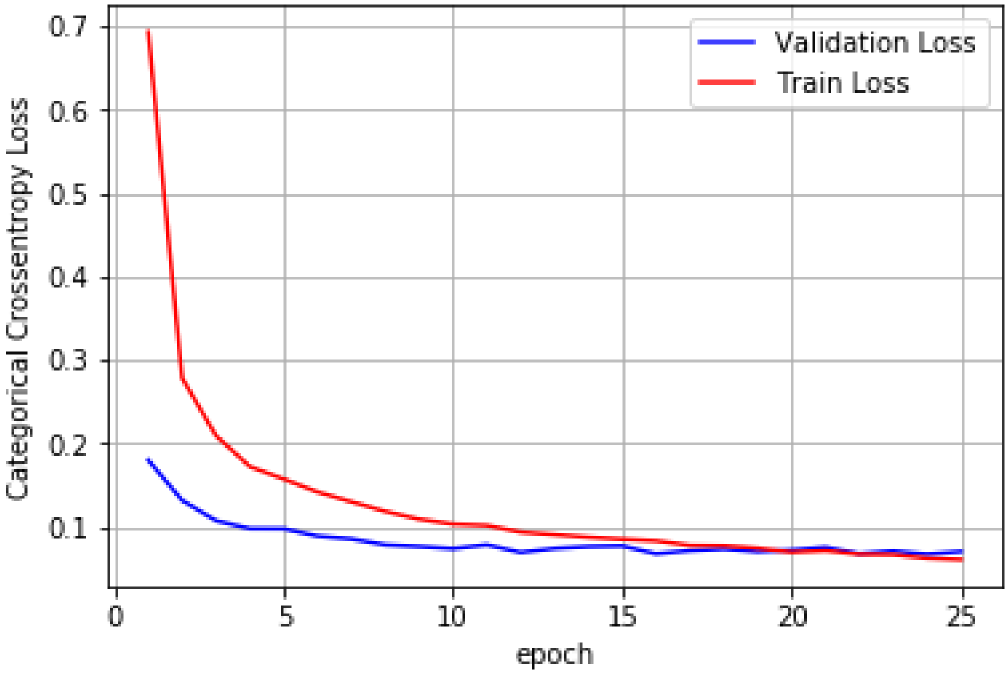
## 5 Architecture with 3 hidden layers.

```
In [81]: n_layers = [512,256,128]
         d = [0.5,0.6,0.5]
         nb_epoch = 25
         nn3(n_layers,d)
```

```
-----------------------------------------------------------------
Layer (type)                 Output Shape              Param #
```

```
=================================================================
dense_135 (Dense)            (None, 512)               401920
_________________________________________________________________
dropout_70 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_136 (Dense)            (None, 256)               131328
_________________________________________________________________
dropout_71 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_137 (Dense)            (None, 128)               32896
_________________________________________________________________
dropout_72 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_138 (Dense)            (None, 10)                1290
=================================================================
Total params: 567,434
Trainable params: 567,434
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/25
60000/60000 [==============================] - 13s 211us/step - loss: 0.6933 - acc: 0.7798 - va
Epoch 2/25
60000/60000 [==============================] - 3s 55us/step - loss: 0.2783 - acc: 0.9225 - val_
Epoch 3/25
60000/60000 [==============================] - 3s 53us/step - loss: 0.2091 - acc: 0.9422 - val_
Epoch 4/25
60000/60000 [==============================] - 3s 51us/step - loss: 0.1721 - acc: 0.9521 - val_
Epoch 5/25
60000/60000 [==============================] - 3s 51us/step - loss: 0.1572 - acc: 0.9570 - val_
Epoch 6/25
60000/60000 [==============================] - 3s 51us/step - loss: 0.1416 - acc: 0.9613 - val_
Epoch 7/25
60000/60000 [==============================] - 3s 52us/step - loss: 0.1299 - acc: 0.9645 - val_
Epoch 8/25
60000/60000 [==============================] - 3s 52us/step - loss: 0.1183 - acc: 0.9675 - val_
Epoch 9/25
60000/60000 [==============================] - 3s 52us/step - loss: 0.1090 - acc: 0.9695 - val_
Epoch 10/25
60000/60000 [==============================] - 3s 51us/step - loss: 0.1031 - acc: 0.9710 - val_
Epoch 11/25
60000/60000 [==============================] - 3s 51us/step - loss: 0.1015 - acc: 0.9721 - val_
Epoch 12/25
60000/60000 [==============================] - 3s 52us/step - loss: 0.0935 - acc: 0.9738 - val_
Epoch 13/25
60000/60000 [==============================] - 3s 52us/step - loss: 0.0904 - acc: 0.9743 - val_
Epoch 14/25
60000/60000 [==============================] - 3s 52us/step - loss: 0.0874 - acc: 0.9759 - val_
```

```
Epoch 15/25
60000/60000 [==============================] - 3s 51us/step - loss: 0.0850 - acc: 0.9758 - val_
Epoch 16/25
60000/60000 [==============================] - 3s 52us/step - loss: 0.0830 - acc: 0.9765 - val_
Epoch 17/25
60000/60000 [==============================] - 3s 51us/step - loss: 0.0775 - acc: 0.9776 - val_
Epoch 18/25
60000/60000 [==============================] - 3s 51us/step - loss: 0.0771 - acc: 0.9788 - val_
Epoch 19/25
60000/60000 [==============================] - 3s 51us/step - loss: 0.0742 - acc: 0.9796 - val_
Epoch 20/25
60000/60000 [==============================] - 3s 50us/step - loss: 0.0693 - acc: 0.9801 - val_
Epoch 21/25
60000/60000 [==============================] - 3s 50us/step - loss: 0.0712 - acc: 0.9803 - val_
Epoch 22/25
60000/60000 [==============================] - 3s 51us/step - loss: 0.0669 - acc: 0.9804 - val_
Epoch 23/25
60000/60000 [==============================] - 3s 51us/step - loss: 0.0665 - acc: 0.9805 - val_
Epoch 24/25
60000/60000 [==============================] - 3s 51us/step - loss: 0.0622 - acc: 0.9822 - val_
Epoch 25/25
60000/60000 [==============================] - 3s 51us/step - loss: 0.0605 - acc: 0.9829 - val_
Test score: 0.07005962141250102
Test accuracy: 0.9826
```

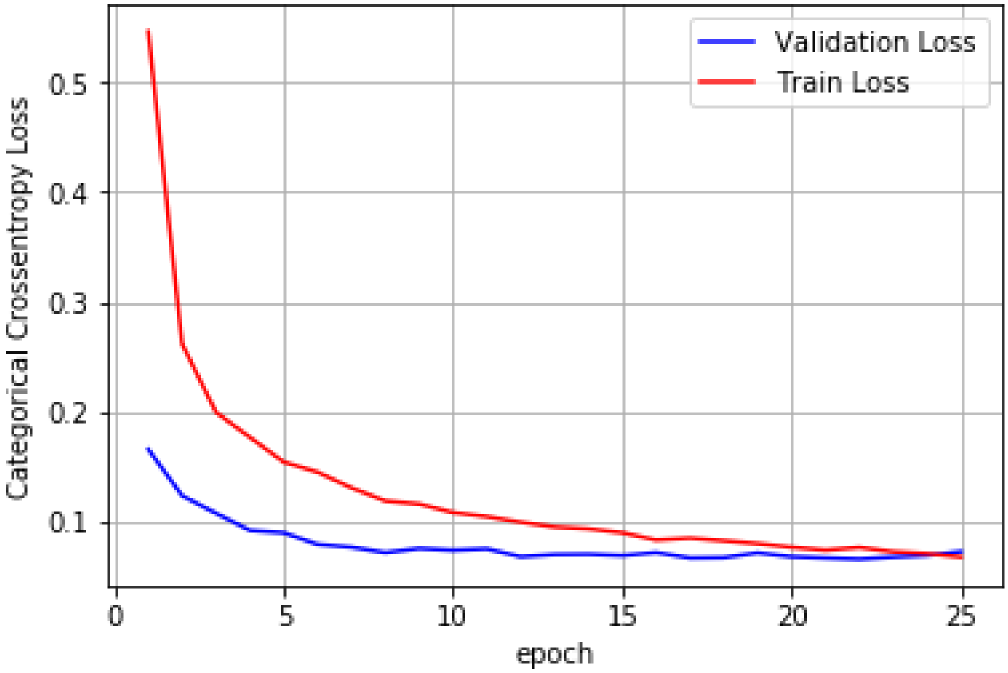


## 6 Architecture with 2 hidden layers.

```
In [82]: n_layers = [468,92]
         d = [0.6,0.5]
         nb_epoch = 25
         nn3(n_layers,d)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_139 (Dense)            (None, 468)               367380
_________________________________________________________________
dropout_73 (Dropout)         (None, 468)               0
_________________________________________________________________
dense_140 (Dense)            (None, 92)                43148
_________________________________________________________________
dropout_74 (Dropout)         (None, 92)                0
_________________________________________________________________
dense_141 (Dense)            (None, 10)                930
=================================================================
Total params: 411,458
Trainable params: 411,458
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/25
60000/60000 [==============================] - 12s 204us/step - loss: 0.5460 - acc: 0.8301 - va
Epoch 2/25
60000/60000 [==============================] - 3s 48us/step - loss: 0.2620 - acc: 0.9256 - val_
Epoch 3/25
60000/60000 [==============================] - 3s 48us/step - loss: 0.1996 - acc: 0.9425 - val_
Epoch 4/25
60000/60000 [==============================] - 3s 47us/step - loss: 0.1770 - acc: 0.9495 - val_
Epoch 5/25
60000/60000 [==============================] - 3s 47us/step - loss: 0.1545 - acc: 0.9549 - val_
Epoch 6/25
60000/60000 [==============================] - 3s 48us/step - loss: 0.1453 - acc: 0.9576 - val_
Epoch 7/25
60000/60000 [==============================] - 3s 48us/step - loss: 0.1310 - acc: 0.9618 - val_
Epoch 8/25
60000/60000 [==============================] - 3s 48us/step - loss: 0.1191 - acc: 0.9645 - val_
Epoch 9/25
60000/60000 [==============================] - 3s 47us/step - loss: 0.1163 - acc: 0.9656 - val_
Epoch 10/25
60000/60000 [==============================] - 3s 47us/step - loss: 0.1085 - acc: 0.9682 - val_
Epoch 11/25
60000/60000 [==============================] - 3s 47us/step - loss: 0.1048 - acc: 0.9691 - val_
Epoch 12/25
```

```
60000/60000 [==============================] - 3s 47us/step - loss: 0.0997 - acc: 0.9707 - val_
Epoch 13/25
60000/60000 [==============================] - 3s 48us/step - loss: 0.0954 - acc: 0.9715 - val_
Epoch 14/25
60000/60000 [==============================] - 3s 47us/step - loss: 0.0937 - acc: 0.9730 - val_
Epoch 15/25
60000/60000 [==============================] - 3s 47us/step - loss: 0.0901 - acc: 0.9732 - val_
Epoch 16/25
60000/60000 [==============================] - 3s 47us/step - loss: 0.0835 - acc: 0.9750 - val_
Epoch 17/25
60000/60000 [==============================] - 3s 48us/step - loss: 0.0852 - acc: 0.9744 - val_
Epoch 18/25
60000/60000 [==============================] - 3s 47us/step - loss: 0.0828 - acc: 0.9748 - val_
Epoch 19/25
60000/60000 [==============================] - 3s 47us/step - loss: 0.0802 - acc: 0.9758 - val_
Epoch 20/25
60000/60000 [==============================] - 3s 47us/step - loss: 0.0769 - acc: 0.9768 - val_
Epoch 21/25
60000/60000 [==============================] - 3s 48us/step - loss: 0.0743 - acc: 0.9778 - val_
Epoch 22/25
60000/60000 [==============================] - 3s 48us/step - loss: 0.0767 - acc: 0.9764 - val_
Epoch 23/25
60000/60000 [==============================] - 3s 48us/step - loss: 0.0728 - acc: 0.9777 - val_
Epoch 24/25
60000/60000 [==============================] - 3s 47us/step - loss: 0.0712 - acc: 0.9783 - val_
Epoch 25/25
60000/60000 [==============================] - 3s 47us/step - loss: 0.0681 - acc: 0.9795 - val_
Test score: 0.07291174245599469
Test accuracy: 0.9823
```



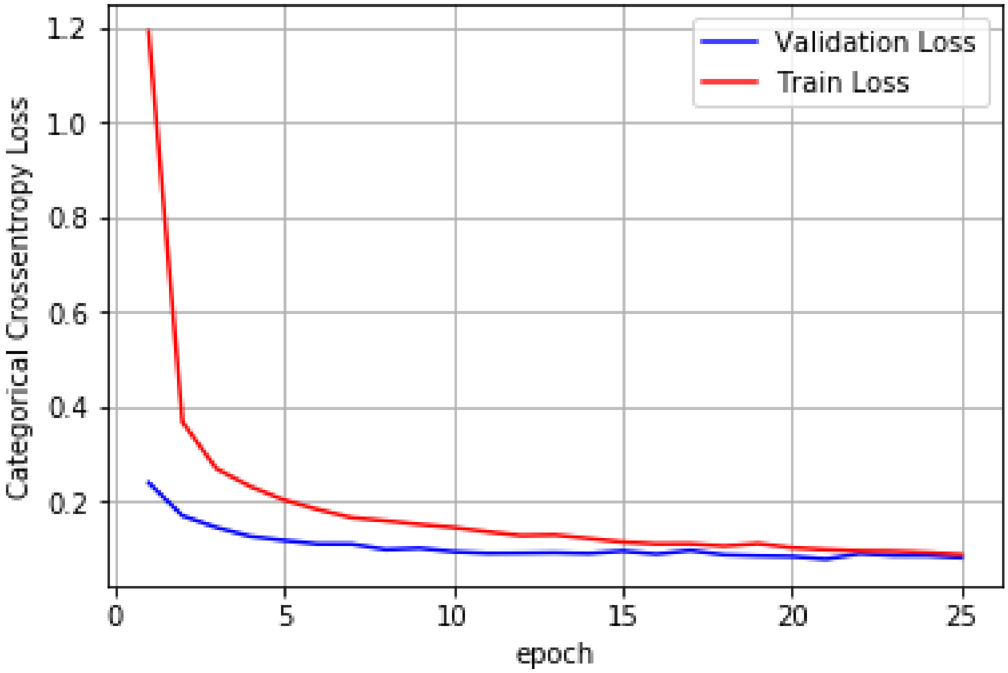
## 7 Architecture with 5 hidden layers

```
In [83]: n_layers = [650,520,410,290,125]
         d = [0.5,0.6,0.5,0.5,0.5]
         nb_epoch = 25
         nn3(n_layers,d)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_142 (Dense)            (None, 650)               510250
_________________________________________________________________
dropout_75 (Dropout)         (None, 650)               0
_________________________________________________________________
dense_143 (Dense)            (None, 520)               338520
_________________________________________________________________
dropout_76 (Dropout)         (None, 520)               0
_________________________________________________________________
dense_144 (Dense)            (None, 410)               213610
_________________________________________________________________
dropout_77 (Dropout)         (None, 410)               0
_________________________________________________________________
dense_145 (Dense)            (None, 290)               119190
_________________________________________________________________
```

```
dropout_78 (Dropout)         (None, 290)               0

-----------------------------------------------------------------
dense_146 (Dense)            (None, 125)               36375

-----------------------------------------------------------------
dropout_79 (Dropout)         (None, 125)               0

-----------------------------------------------------------------
dense_147 (Dense)            (None, 10)                1260
=================================================================
Total params: 1,219,205
Trainable params: 1,219,205
Non-trainable params: 0

-----------------------------------------------------------------
Train on 60000 samples, validate on 10000 samples
Epoch 1/25
60000/60000 [==============================] - 13s 217us/step - loss: 1.1927 - acc: 0.5892 - v
Epoch 2/25
60000/60000 [==============================] - 4s 61us/step - loss: 0.3680 - acc: 0.9039 - val_
Epoch 3/25
60000/60000 [==============================] - 4s 60us/step - loss: 0.2690 - acc: 0.9318 - val_
Epoch 4/25
60000/60000 [==============================] - 4s 60us/step - loss: 0.2314 - acc: 0.9419 - val_
Epoch 5/25
60000/60000 [==============================] - 4s 59us/step - loss: 0.2028 - acc: 0.9509 - val_
Epoch 6/25
60000/60000 [==============================] - 4s 59us/step - loss: 0.1827 - acc: 0.9546 - val_
Epoch 7/25
60000/60000 [==============================] - 4s 61us/step - loss: 0.1654 - acc: 0.9595 - val_
Epoch 8/25
60000/60000 [==============================] - 4s 61us/step - loss: 0.1588 - acc: 0.9606 - val_
Epoch 9/25
60000/60000 [==============================] - 4s 62us/step - loss: 0.1512 - acc: 0.9623 - val_
Epoch 10/25
60000/60000 [==============================] - 4s 61us/step - loss: 0.1447 - acc: 0.9647 - val_
Epoch 11/25
60000/60000 [==============================] - 4s 60us/step - loss: 0.1356 - acc: 0.9666 - val_
Epoch 12/25
60000/60000 [==============================] - 4s 59us/step - loss: 0.1276 - acc: 0.9689 - val_
Epoch 13/25
60000/60000 [==============================] - 4s 60us/step - loss: 0.1289 - acc: 0.9686 - val_
Epoch 14/25
60000/60000 [==============================] - 4s 60us/step - loss: 0.1214 - acc: 0.9703 - val_
Epoch 15/25
60000/60000 [==============================] - 4s 60us/step - loss: 0.1139 - acc: 0.9728 - val_
Epoch 16/25
60000/60000 [==============================] - 4s 59us/step - loss: 0.1100 - acc: 0.9727 - val_
Epoch 17/25
60000/60000 [==============================] - 4s 60us/step - loss: 0.1101 - acc: 0.9729 - val_
Epoch 18/25
```

```
60000/60000 [==============================] - 4s 59us/step - loss: 0.1053 - acc: 0.9743 - val
Epoch 19/25
60000/60000 [==============================] - 4s 60us/step - loss: 0.1106 - acc: 0.9731 - val
Epoch 20/25
60000/60000 [==============================] - 4s 60us/step - loss: 0.1019 - acc: 0.9760 - val
Epoch 21/25
60000/60000 [==============================] - 4s 60us/step - loss: 0.0978 - acc: 0.9759 - val
Epoch 22/25
60000/60000 [==============================] - 4s 60us/step - loss: 0.0950 - acc: 0.9769 - val
Epoch 23/25
60000/60000 [==============================] - 4s 60us/step - loss: 0.0943 - acc: 0.9773 - val
Epoch 24/25
60000/60000 [==============================] - 4s 60us/step - loss: 0.0916 - acc: 0.9777 - val
Epoch 25/25
60000/60000 [==============================] - 4s 59us/step - loss: 0.0877 - acc: 0.9781 - val
Test score: 0.08099026859523369
Test accuracy: 0.9819
```



## 8 Experimenting with conditional batch normalisation and dropout with 3 hidden layers.

```
In [0]: # https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormalization-
        def expt():
```

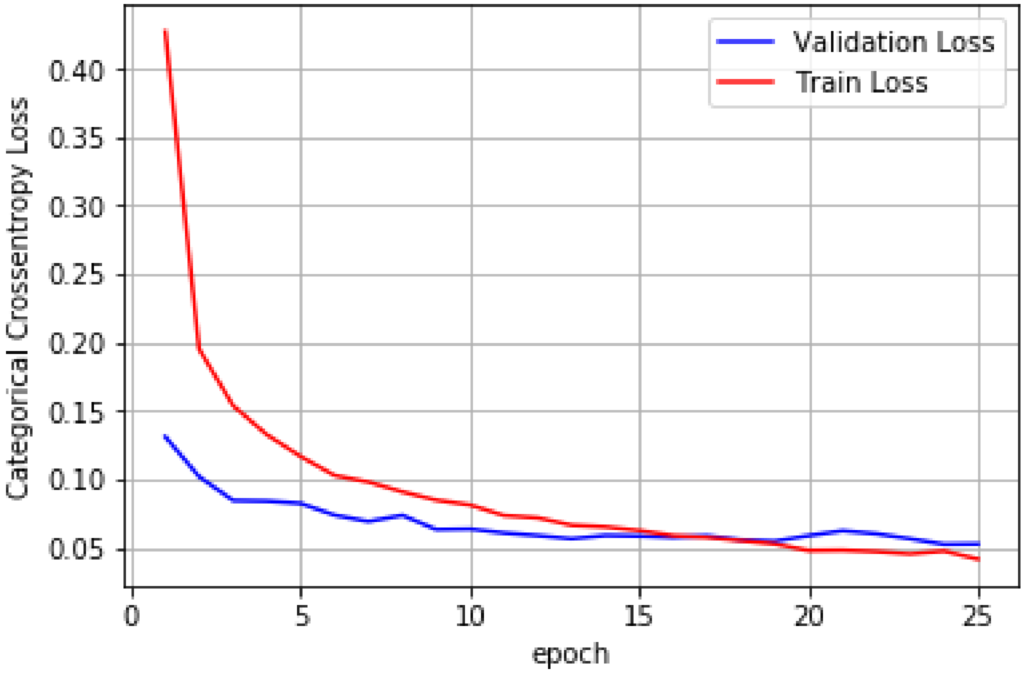
```
    model = Sequential()
    model.add(Dense(584, activation='relu', input_shape=(input_dim,), kernel_initializer
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(320, activation='relu',kernel_initializer=K.he_normal(seed=None)) )
    model.add(BatchNormalization())
    model.add(Dense(125, activation='relu',kernel_initializer=K.he_normal(seed=None)) )
    model.add(Dropout(0.5))


    model.add(Dense(output_dim, activation='softmax'))


    model.summary()


    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'

    history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbos


    %matplotlib inline
    score = model.evaluate(X_test, Y_test, verbose=0)
    print('Test score:', score[0])
    print('Test accuracy:', score[1])

    fig,ax = plt.subplots(1,1)
    ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

    # list of epoch numbers
    x = list(range(1,nb_epoch+1))

     #dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
    # history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,

    # we will get val_loss and val_acc only when you pass the paramter validation_data
    # val_loss : validation loss
    # val_acc : validation accuracy

    # loss : training loss
    # acc : train accuracy
    # for each key in histrory.histrory we will have a list of length equal to number of
    vy = history.history['val_loss']
    ty = history.history['loss']
    plt_dynamic(x, vy, ty, ax)
```

```
In [87]: expt()
```

```
-----------------------------------------------------------------
```

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_148 (Dense)            (None, 584)               458440
_________________________________________________________________
batch_normalization_90 (Batc (None, 584)               2336
_________________________________________________________________
dropout_80 (Dropout)         (None, 584)               0
_________________________________________________________________
dense_149 (Dense)            (None, 320)               187200
_________________________________________________________________
batch_normalization_91 (Batc (None, 320)               1280
_________________________________________________________________
dense_150 (Dense)            (None, 125)               40125
_________________________________________________________________
dropout_81 (Dropout)         (None, 125)               0
_________________________________________________________________
dense_151 (Dense)            (None, 10)                1260
=================================================================
Total params: 690,641
Trainable params: 688,833
Non-trainable params: 1,808
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/25
60000/60000 [==============================] - 14s 240us/step - loss: 0.4274 - acc: 0.8725 - va
Epoch 2/25
60000/60000 [==============================] - 4s 73us/step - loss: 0.1952 - acc: 0.9413 - val_
Epoch 3/25
60000/60000 [==============================] - 4s 73us/step - loss: 0.1537 - acc: 0.9543 - val_
Epoch 4/25
60000/60000 [==============================] - 4s 75us/step - loss: 0.1326 - acc: 0.9605 - val_
Epoch 5/25
60000/60000 [==============================] - 4s 74us/step - loss: 0.1166 - acc: 0.9649 - val_
Epoch 6/25
60000/60000 [==============================] - 5s 75us/step - loss: 0.1027 - acc: 0.9687 - val_
Epoch 7/25
60000/60000 [==============================] - 5s 76us/step - loss: 0.0979 - acc: 0.9701 - val_
Epoch 8/25
60000/60000 [==============================] - 5s 76us/step - loss: 0.0907 - acc: 0.9718 - val_
Epoch 9/25
60000/60000 [==============================] - 4s 72us/step - loss: 0.0847 - acc: 0.9740 - val_
Epoch 10/25
60000/60000 [==============================] - 4s 73us/step - loss: 0.0812 - acc: 0.9745 - val_
Epoch 11/25
60000/60000 [==============================] - 4s 75us/step - loss: 0.0734 - acc: 0.9773 - val_
Epoch 12/25
60000/60000 [==============================] - 4s 71us/step - loss: 0.0717 - acc: 0.9780 - val_
Epoch 13/25
```

```
60000/60000 [==============================] - 4s 72us/step - loss: 0.0663 - acc: 0.9793 - val
Epoch 14/25
60000/60000 [==============================] - 4s 72us/step - loss: 0.0651 - acc: 0.9792 - val
Epoch 15/25
60000/60000 [==============================] - 4s 72us/step - loss: 0.0624 - acc: 0.9805 - val
Epoch 16/25
60000/60000 [==============================] - 4s 73us/step - loss: 0.0586 - acc: 0.9817 - val
Epoch 17/25
60000/60000 [==============================] - 4s 72us/step - loss: 0.0573 - acc: 0.9821 - val
Epoch 18/25
60000/60000 [==============================] - 4s 72us/step - loss: 0.0550 - acc: 0.9823 - val
Epoch 19/25
60000/60000 [==============================] - 4s 71us/step - loss: 0.0526 - acc: 0.9840 - val
Epoch 20/25
60000/60000 [==============================] - 4s 71us/step - loss: 0.0478 - acc: 0.9846 - val
Epoch 21/25
60000/60000 [==============================] - 4s 70us/step - loss: 0.0479 - acc: 0.9845 - val
Epoch 22/25
60000/60000 [==============================] - 4s 71us/step - loss: 0.0469 - acc: 0.9852 - val
Epoch 23/25
60000/60000 [==============================] - 4s 73us/step - loss: 0.0453 - acc: 0.9855 - val
Epoch 24/25
60000/60000 [==============================] - 4s 72us/step - loss: 0.0473 - acc: 0.9850 - val
Epoch 25/25
60000/60000 [==============================] - 4s 71us/step - loss: 0.0415 - acc: 0.9864 - val
Test score: 0.05252337242660105
Test accuracy: 0.9855
```



# 9 Conclusions

```
In [90]: from prettytable import PrettyTable
         print("===================================================================")
         print(" FOR DROPOUT+BATCH NORMALISATION")
         x = PrettyTable()
         x.field_names = ["No. of hidden layers", "Train Loss", " Val Loss", "Val AUC"]
         x.add_row([2, 0.0645, 0.0584, 0.9828])
         x.add_row([3, 0.0612, 0.0549, 0.9850])
         x.add_row([4, 0.0612, 0.0638, 0.9847])
         x.add_row([5, 0.0669, 0.0607, 0.9846])
         print(x)
         print("====================================================================")
         print("====================================================================")
         print(" FOR ONLY BATCH NORMALISATION and NO DROPOUT")
         x = PrettyTable()
         x.field_names = ["No. of hidden layers", "Train Loss", " Val Loss", "Val AUC"]
         x.add_row([2, 0.0050, 0.0840, 0.9808])
         x.add_row([3, 0.0058, 0.0723, 0.9833])
         x.add_row([5, 0.0093, 0.0698, 0.9840])
         print(x)

         print("====================================================================")
         print("====================================================================")
```

```
print(" FOR NO DROPOUT AND NO BATCH NORMALISATION")
x = PrettyTable()
x.field_names = ["No. of hidden layers", "Train Loss", " Val Loss", "Val AUC"]
x.add_row([2, 0.0065, 0.0998, 0.9807])
x.add_row([3, 0.0072, 0.0933, 0.9814])
x.add_row([5, 0.0095, 0.1317, 0.9788])
print(x)

print("====================================================================")
print("====================================================================")
print(" FOR DROPOUT ONLY AND NO BATCH NORMALISATION")
x = PrettyTable()
x.field_names = ["No. of hidden layers", "Train Loss", " Val Loss", "Val AUC"]
x.add_row([2, 0.0681, 0.0729, 0.9823])
x.add_row([3, 0.0605, 0.0701, 0.9826])
x.add_row([5, 0.0877, 0.0810, 0.9819])
print(x)

print("====================================================================")
print("====================================================================")
print("EXPERIMENTAL _ DROPOUT-NORMALISATION-DROPOUT")
x = PrettyTable()
x.field_names = ["No. of hidden layers", "Train Loss", " Val Loss", "Val AUC"]
x.add_row([3, 0.0550, 0.0556, 0.9845])
print(x)
```

```
===================================================================
 FOR DROPOUT+BATCH NORMALISATION
+----------------------+------------+-----------+---------+
| No. of hidden layers | Train Loss |  Val Loss | Val AUC |
+----------------------+------------+-----------+---------+
|          2           |   0.0645   |   0.0584  |  0.9828 |
|          3           |   0.0612   |   0.0549  |  0.985  |
|          4           |   0.0612   |   0.0638  |  0.9847 |
|          5           |   0.0669   |   0.0607  |  0.9846 |
+----------------------+------------+-----------+---------+
====================================================================
====================================================================
 FOR ONLY BATCH NORMALISATION and NO DROPOUT
+----------------------+------------+-----------+---------+
| No. of hidden layers | Train Loss |  Val Loss | Val AUC |
+----------------------+------------+-----------+---------+
|          2           |   0.005    |   0.084   |  0.9808 |
|          3           |   0.0058   |   0.0723  |  0.9833 |
|          5           |   0.0093   |   0.0698  |  0.984  |
+----------------------+------------+-----------+---------+
====================================================================
====================================================================
```

```
 FOR NO DROPOUT AND NO BATCH NORMALISATION
+----------------------+------------+-----------+---------+
| No. of hidden layers | Train Loss |  Val Loss | Val AUC |
+----------------------+------------+-----------+---------+
|          2           |   0.0065   |   0.0998  |  0.9807 |
|          3           |   0.0072   |   0.0933  |  0.9814 |
|          5           |   0.0095   |   0.1317  |  0.9788 |
+----------------------+------------+-----------+---------+
=====================================================================
=====================================================================
 FOR DROPOUT ONLY AND NO BATCH NORMALISATION
+----------------------+------------+-----------+---------+
| No. of hidden layers | Train Loss |  Val Loss | Val AUC |
+----------------------+------------+-----------+---------+
|          2           |   0.0681   |   0.0729  |  0.9823 |
|          3           |   0.0605   |   0.0701  |  0.9826 |
|          5           |   0.0877   |   0.081   |  0.9819 |
+----------------------+------------+-----------+---------+
=====================================================================
=====================================================================
EXPERIMENTAL _ DROPOUT-NORMALISATION-DROPOUT
+----------------------+------------+-----------+---------+
| No. of hidden layers | Train Loss |  Val Loss | Val AUC |
+----------------------+------------+-----------+---------+
|          3           |   0.055    |   0.0556  |  0.9845 |
+----------------------+------------+-----------+---------+
```

### 9.0.1 Conclusions

1) Observing all the pretty tables, we conclude that ( Dropout+Batch Normalisation) > only Dropout > only batch normalisation > no dropout or normalisation

2) As can be observed form the graphs , overfitting occurs, wherever there is no dropout or batch normalisation.

3) The best model was found from the experimental model.(Considering upto epoch 18.) It was done as : For this , drop out was applied on first hidden layer.Then, batch normalisation on second hidden layer. Then again, dropout on third hidden layer. This way,instead of applying both of them on all layers, time and cost of computing was saved. The result was also good. The diff between train loss and val loss was 0.0006 at epoch 18(Least of all models).Other models took till epoch near to 25 to converge.

4) We can infer that dropout and batch normalisation helps a lot in deep learning models especially in avoiding overfitting.