TRYING DIFFERENT MLP ARCHITECTURE ON MNIST DATASET USING KERAS

In [56]:
```
from keras.utils import np_utils
from keras.datasets import mnist
import keras
import seaborn as sns
from keras.initializers import RandomNormal
```

Function which plots between loss and each epoch for Both test and Train

In [57]:
```
import matplotlib.pyplot as plt
import numpy as np
import time
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()
```

Splitting Train and Test

In [58]:
```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

Before Normalization

```
In [59]: X_train = X_train.reshape(X_train.shape[0], X_train.shape[1]*X_train.sh
ape[2])
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1]*X_test.shape[2
])
X_train[0]
```

```
Out[59]: array([  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   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], dtype=uint8)
```

After Normalization

In [60]:
```
X_train = X_train/255
X_test = X_test/255
X_train[0]
```

Out[60]:
```
array([0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
```

```
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 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.        ])
```

Reshaping the ytain and y test

In [61]:
```
print("Class label of first image :", y_train[0])
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.]
```

Softmax Classifier

In [62]:
```
from keras.models import Sequential
from keras.layers import Dense, Activation
```

Initializing

In [63]:
```
# some model parameters

out_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
no_epoch = 20
```

MLP+ReLu Activation+Adam optimizer

In [64]:
```
model_relu=Sequential()
model_relu.add(Dense(500, activation='relu', input_shape=(input_dim,),
kernel_initializer=keras.initializers.he_normal(seed=None)))
model_relu.add(Dense(100, activation='relu', kernel_initializer=keras.i
nitializers.he_normal(seed=None)) )
```

```
model_relu.add(Dense(out_dim, activation='softmax'))
model_relu.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_20 (Dense)             (None, 500)               392500
_________________________________________________________________
dense_21 (Dense)             (None, 100)               50100
_________________________________________________________________
dense_22 (Dense)             (None, 10)                1010
=================================================================
Total params: 443,610
Trainable params: 443,610
Non-trainable params: 0
_________________________________________________________________
```

In [65]:
```
model_relu.compile(optimizer='adam', loss='categorical_crossentropy', m
etrics=['accuracy'])

history = model_relu.fit(X_train, Y_train, batch_size=batch_size, epoch
s=no_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.
2415 - acc: 0.9288 - val_loss: 0.1206 - val_acc: 0.9632
Epoch 2/20
60000/60000 [==============================] - 6s 106us/step - loss: 0.
0906 - acc: 0.9727 - val_loss: 0.0873 - val_acc: 0.9739
Epoch 3/20
60000/60000 [==============================] - 7s 109us/step - loss: 0.
0586 - acc: 0.9816 - val_loss: 0.0805 - val_acc: 0.9755
Epoch 4/20
60000/60000 [==============================] - 6s 108us/step - loss: 0.
0382 - acc: 0.9880 - val_loss: 0.0709 - val_acc: 0.9791
Epoch 5/20
60000/60000 [==============================] - 6s 108us/step - loss: 0.
0287 - acc: 0.9908 - val_loss: 0.0739 - val_acc: 0.9767
Epoch 6/20
```

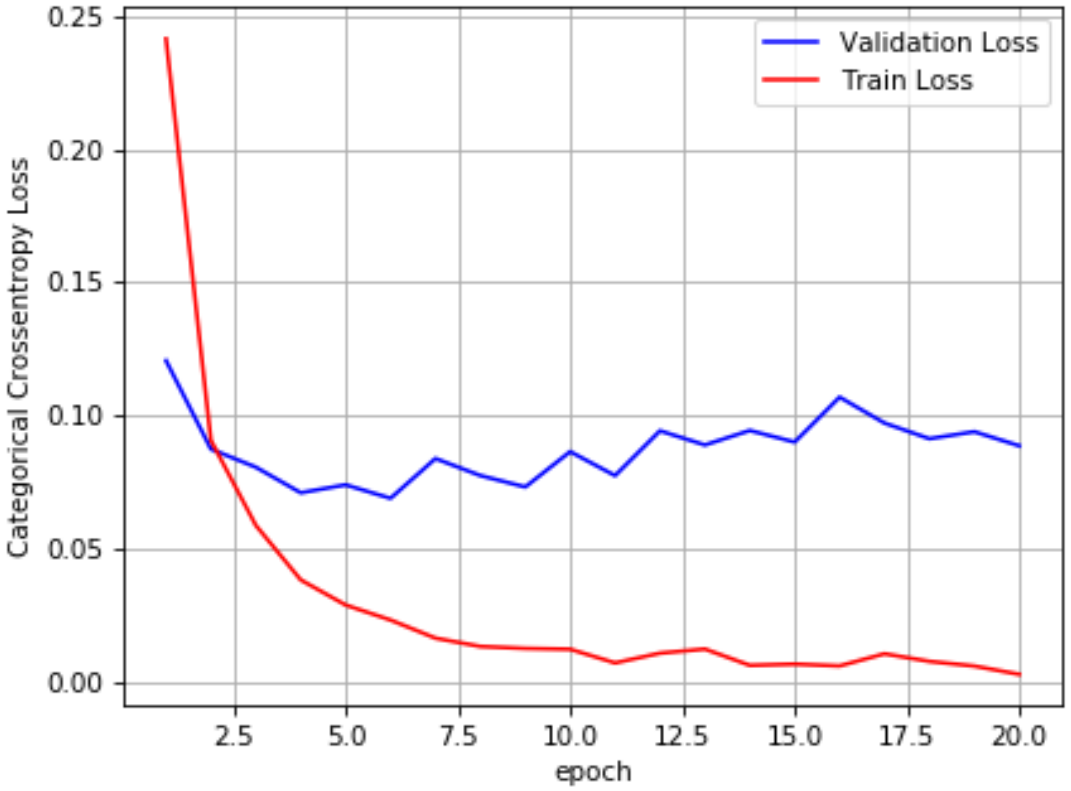
```
60000/60000 [==============================] - 7s 109us/step - loss: 0.
0231 - acc: 0.9929 - val_loss: 0.0688 - val_acc: 0.9797
Epoch 7/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.
0163 - acc: 0.9947 - val_loss: 0.0838 - val_acc: 0.9771
Epoch 8/20
60000/60000 [==============================] - 6s 108us/step - loss: 0.
0131 - acc: 0.9956 - val_loss: 0.0774 - val_acc: 0.9803
Epoch 9/20
60000/60000 [==============================] - 6s 102us/step - loss: 0.
0124 - acc: 0.9958 - val_loss: 0.0731 - val_acc: 0.9799
Epoch 10/20
60000/60000 [==============================] - 6s 99us/step - loss: 0.0
121 - acc: 0.9962 - val_loss: 0.0863 - val_acc: 0.9783
Epoch 11/20
60000/60000 [==============================] - 6s 99us/step - loss: 0.0
070 - acc: 0.9978 - val_loss: 0.0773 - val_acc: 0.9824
Epoch 12/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.
0106 - acc: 0.9964 - val_loss: 0.0942 - val_acc: 0.9778
Epoch 13/20
60000/60000 [==============================] - 7s 110us/step - loss: 0.
0122 - acc: 0.9959 - val_loss: 0.0888 - val_acc: 0.9803
Epoch 14/20
60000/60000 [==============================] - 6s 99us/step - loss: 0.0
060 - acc: 0.9979 - val_loss: 0.0943 - val_acc: 0.9804
Epoch 15/20
60000/60000 [==============================] - 6s 99us/step - loss: 0.0
065 - acc: 0.9980 - val_loss: 0.0899 - val_acc: 0.9798
Epoch 16/20
60000/60000 [==============================] - 6s 99us/step - loss: 0.0
058 - acc: 0.9980 - val_loss: 0.1068 - val_acc: 0.9774
Epoch 17/20
60000/60000 [==============================] - 6s 100us/step - loss: 0.
0104 - acc: 0.9967 - val_loss: 0.0971 - val_acc: 0.9800
Epoch 18/20
60000/60000 [==============================] - 6s 98us/step - loss: 0.0
075 - acc: 0.9974 - val_loss: 0.0912 - val_acc: 0.9799
Epoch 19/20
```

```
60000/60000 [==============================] - 6s 98us/step - loss: 0.0
058 - acc: 0.9980 - val_loss: 0.0938 - val_acc: 0.9823
Epoch 20/20
60000/60000 [==============================] - 6s 98us/step - loss: 0.0
027 - acc: 0.9992 - val_loss: 0.0885 - val_acc: 0.9821
```

In [66]:

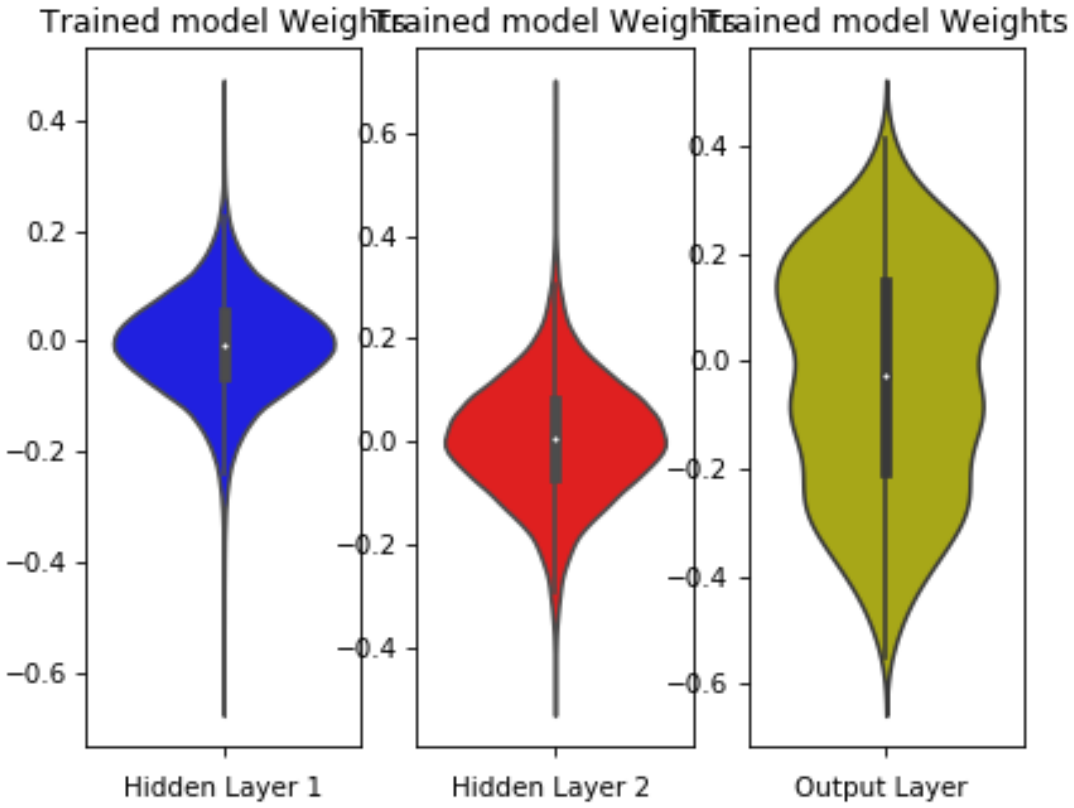
```
score = model.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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


x = list(range(1,nb_epoch+1))


vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.3360099180340767
Test accuracy: 0.9073
```



In [67]:
```
w_after = model_relu.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
```

```
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



MLP+ReLu Activation+Adam optimizer+BN+Dropout

In [68]:
```
from keras.layers import Dropout
from keras.layers import BatchNormalization
model_relu=Sequential()
model_relu.add(Dense(500, activation='relu', input_shape=(input_dim,),
kernel_initializer=keras.initializers.he_normal(seed=None)))
model_relu.add(BatchNormalization())
```

```
model_relu.add(Dropout(0.5))
model_relu.add(Dense(100, activation='relu', kernel_initializer=keras.i
nitializers.he_normal(seed=None)) )
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(out_dim, activation='softmax'))
model_relu.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_23 (Dense)             (None, 500)               392500
_________________________________________________________________
batch_normalization_11 (Batc (None, 500)               2000
_________________________________________________________________
dropout_11 (Dropout)         (None, 500)               0
_________________________________________________________________
dense_24 (Dense)             (None, 100)               50100
_________________________________________________________________
batch_normalization_12 (Batc (None, 100)               400
_________________________________________________________________
dropout_12 (Dropout)         (None, 100)               0
_________________________________________________________________
dense_25 (Dense)             (None, 10)                1010
=================================================================
Total params: 446,010
Trainable params: 444,810
Non-trainable params: 1,200
_________________________________________________________________
```

In [69]:
```
model_relu.compile(optimizer='adam', loss='categorical_crossentropy', m
etrics=['accuracy'])

history = model_relu.fit(X_train, Y_train, batch_size=batch_size, epoch
s=no_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 11s 185us/step - loss:
0.4455 - acc: 0.8657 - val_loss: 0.1490 - val_acc: 0.9551
```

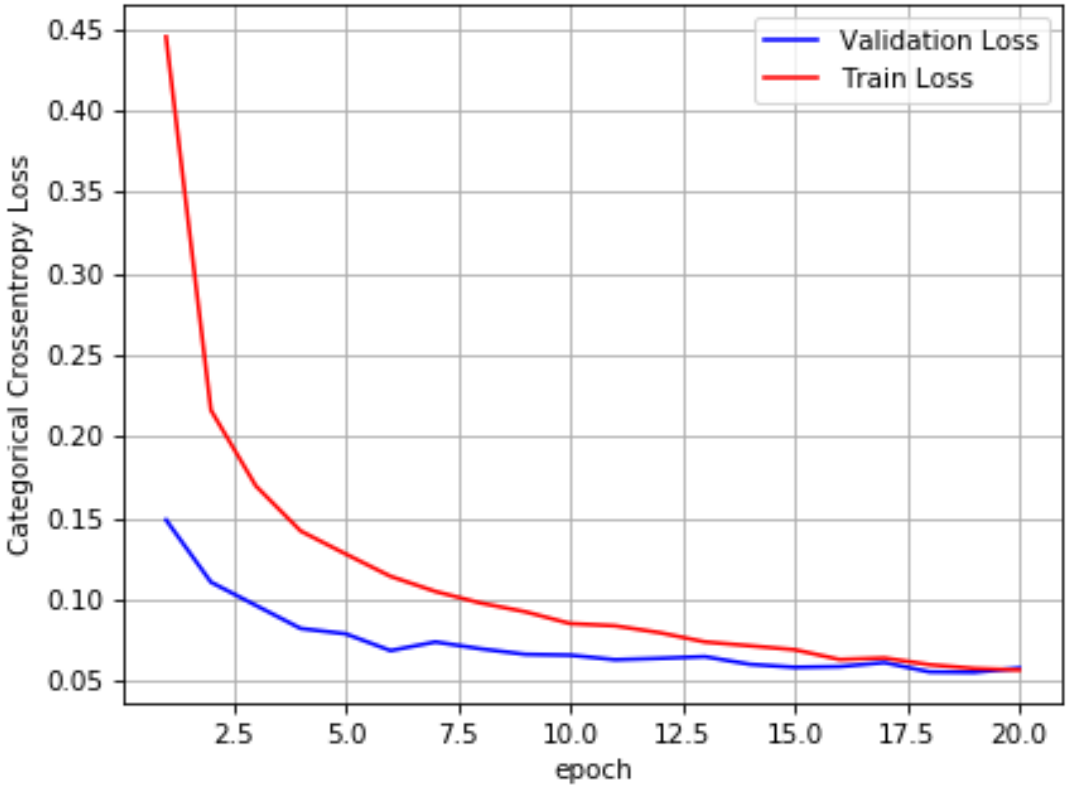
```
Epoch 2/20
60000/60000 [==============================] - 9s 151us/step - loss: 0.
2163 - acc: 0.9351 - val_loss: 0.1108 - val_acc: 0.9671
Epoch 3/20
60000/60000 [==============================] - 10s 159us/step - loss:
0.1698 - acc: 0.9489 - val_loss: 0.0965 - val_acc: 0.9702
Epoch 4/20
60000/60000 [==============================] - 9s 144us/step - loss: 0.
1421 - acc: 0.9571 - val_loss: 0.0823 - val_acc: 0.9733
Epoch 5/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.
1280 - acc: 0.9618 - val_loss: 0.0790 - val_acc: 0.9757
Epoch 6/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.
1143 - acc: 0.9648 - val_loss: 0.0687 - val_acc: 0.9789
Epoch 7/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.
1050 - acc: 0.9685 - val_loss: 0.0741 - val_acc: 0.9779
Epoch 8/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.
0980 - acc: 0.9696 - val_loss: 0.0699 - val_acc: 0.9782
Epoch 9/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.
0926 - acc: 0.9713 - val_loss: 0.0665 - val_acc: 0.9806
Epoch 10/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.
0853 - acc: 0.9741 - val_loss: 0.0659 - val_acc: 0.9806
Epoch 11/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.
0840 - acc: 0.9740 - val_loss: 0.0631 - val_acc: 0.9809
Epoch 12/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.
0797 - acc: 0.9754 - val_loss: 0.0640 - val_acc: 0.9812
Epoch 13/20
60000/60000 [==============================] - 8s 140us/step - loss: 0.
0741 - acc: 0.9773 - val_loss: 0.0650 - val_acc: 0.9796
Epoch 14/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.
0717 - acc: 0.9779 - val_loss: 0.0603 - val_acc: 0.9823
```

```
Epoch 15/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.
0693 - acc: 0.9784 - val_loss: 0.0585 - val_acc: 0.9816
Epoch 16/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.
0633 - acc: 0.9798 - val_loss: 0.0589 - val_acc: 0.9826
Epoch 17/20
60000/60000 [==============================] - 10s 161us/step - loss:
0.0642 - acc: 0.9800 - val_loss: 0.0614 - val_acc: 0.9809
Epoch 18/20
60000/60000 [==============================] - 9s 144us/step - loss: 0.
0601 - acc: 0.9810 - val_loss: 0.0557 - val_acc: 0.9833
Epoch 19/20
60000/60000 [==============================] - 9s 153us/step - loss: 0.
0578 - acc: 0.9820 - val_loss: 0.0554 - val_acc: 0.9831
Epoch 20/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.
0568 - acc: 0.9818 - val_loss: 0.0580 - val_acc: 0.9826
```

In [70]:

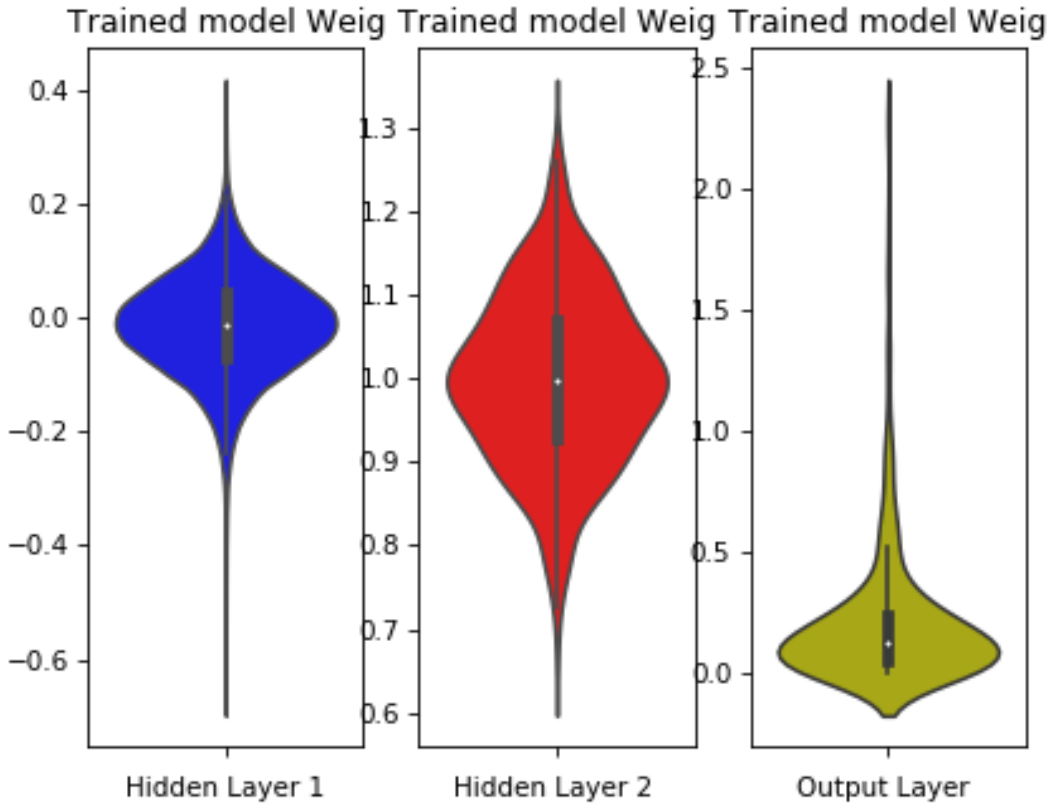
```
score = model_relu.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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


vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.05802350947747473
Test accuracy: 0.9826
```



In [71]:
```
w_after = model_relu.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weig")
```

```
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weig")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 3, 3)
plt.title("Trained model Weig")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



MLP+Relu Function+Adam+ with 3 hidden layers

Here ,I am Changing the variables and initializing again

In [72]:
```
out_dim=10
input_dim=X_train.shape[1]
Batch_size=150
no_epoch=10
```

In [73]:
```
model_relu=Sequential()
model_relu.add(Dense(600, activation='relu', input_shape=(input_dim,),
kernel_initializer=keras.initializers.he_normal(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(200, activation='relu', kernel_initializer=keras.i
nitializers.he_normal(seed=None)) )
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(70,activation='relu',kernel_initializer=keras.init
ializers.he_normal(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(out_dim, activation='softmax'))
model_relu.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_26 (Dense)             (None, 600)               471000
_________________________________________________________________
batch_normalization_13 (Batc (None, 600)               2400
_________________________________________________________________
dropout_13 (Dropout)         (None, 600)               0
_________________________________________________________________
dense_27 (Dense)             (None, 200)               120200
_________________________________________________________________
batch_normalization_14 (Batc (None, 200)               800
_________________________________________________________________
dropout_14 (Dropout)         (None, 200)               0
_________________________________________________________________
dense_28 (Dense)             (None, 70)                14070
_________________________________________________________________
batch_normalization_15 (Batc (None, 70)                280
_________________________________________________________________
dropout_15 (Dropout)         (None, 70)                0
_________________________________________________________________
dense_29 (Dense)             (None, 10)                710
=================================================================
```

```
Total params: 609,460
Trainable params: 607,720
Non-trainable params: 1,740
_________________________________________________________________
```

In [74]:
```
model_relu.compile(optimizer='adam', loss='categorical_crossentropy', m
etrics=['accuracy'])

history = model_relu.fit(X_train, Y_train, batch_size=Batch_size, epoch
s=no_epoch, verbose=1, validation_data=(X_test, Y_test))
```

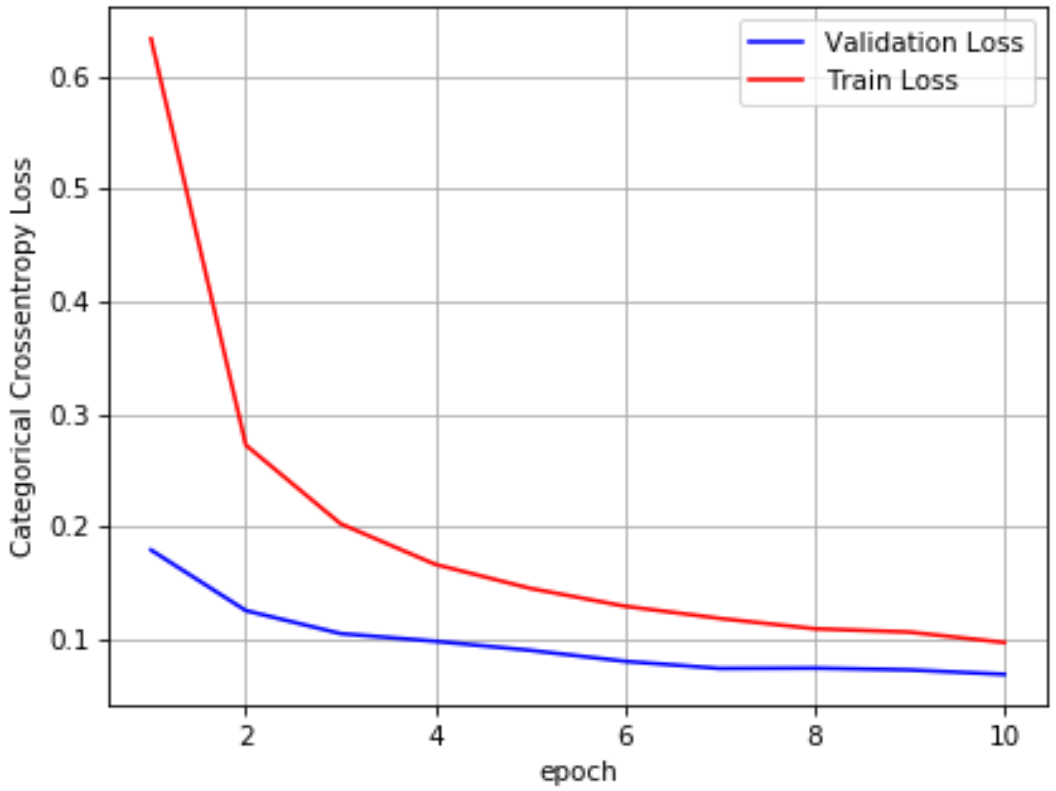
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 12s 208us/step - loss:
0.6336 - acc: 0.8094 - val_loss: 0.1789 - val_acc: 0.9456
Epoch 2/10
60000/60000 [==============================] - 11s 176us/step - loss:
0.2722 - acc: 0.9222 - val_loss: 0.1250 - val_acc: 0.9628
Epoch 3/10
60000/60000 [==============================] - 10s 174us/step - loss:
0.2023 - acc: 0.9423 - val_loss: 0.1046 - val_acc: 0.9686
Epoch 4/10
60000/60000 [==============================] - 10s 171us/step - loss:
0.1661 - acc: 0.9527 - val_loss: 0.0977 - val_acc: 0.9700
Epoch 5/10
60000/60000 [==============================] - 10s 172us/step - loss:
0.1448 - acc: 0.9586 - val_loss: 0.0897 - val_acc: 0.9741
Epoch 6/10
60000/60000 [==============================] - 10s 173us/step - loss:
0.1289 - acc: 0.9626 - val_loss: 0.0799 - val_acc: 0.9768
Epoch 7/10
60000/60000 [==============================] - 10s 173us/step - loss:
0.1180 - acc: 0.9664 - val_loss: 0.0735 - val_acc: 0.9784
Epoch 8/10
60000/60000 [==============================] - 10s 170us/step - loss:
0.1089 - acc: 0.9685 - val_loss: 0.0739 - val_acc: 0.9803
Epoch 9/10
60000/60000 [==============================] - 10s 173us/step - loss:
0.1059 - acc: 0.9694 - val_loss: 0.0723 - val_acc: 0.9783
```

```
Epoch 10/10
60000/60000 [==============================] - 10s 171us/step - loss:
0.0967 - acc: 0.9718 - val_loss: 0.0683 - val_acc: 0.9804
```

In [75]:

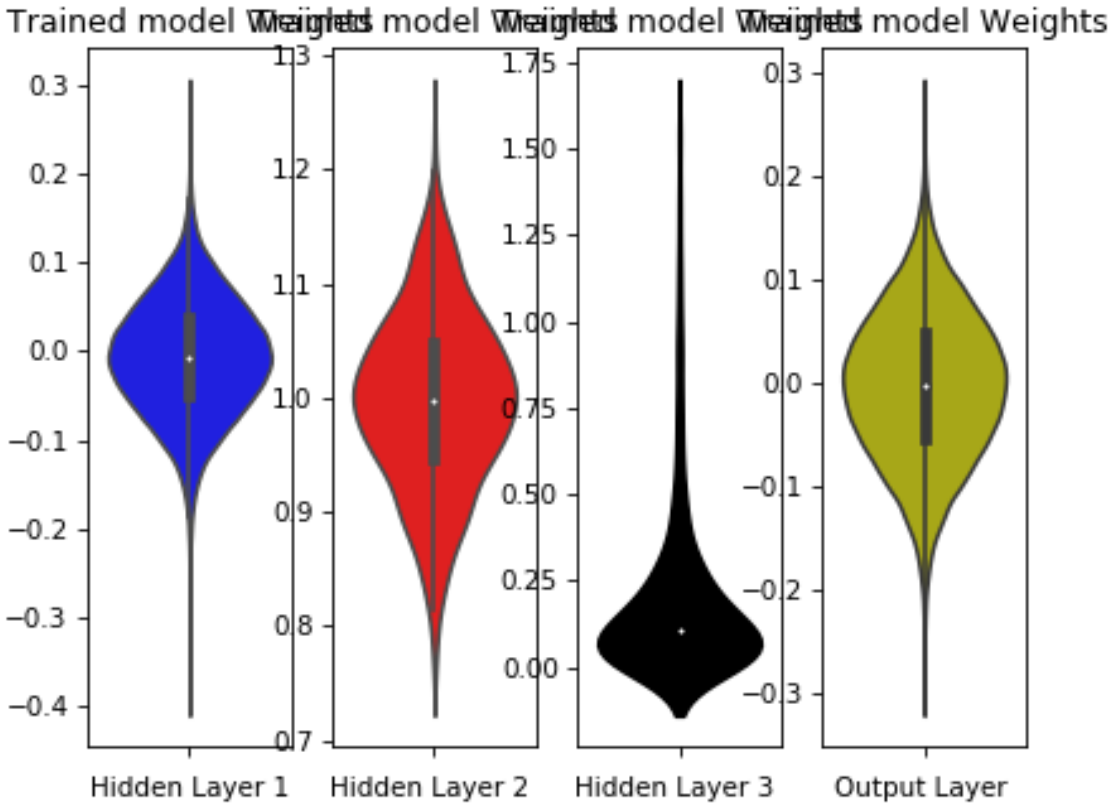
```
score = model_relu.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.06832263506012969
Test accuracy: 0.9804
```



In [76]:
```
w_after = model_relu.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w=w_after[4].flatten().reshape(-1,1)
out_w = w_after[6].flatten().reshape(-1,1)


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 4, 1)
```

```
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 4, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 4, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='black')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(1, 4, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



MLP+Relu Function+Adam+ with 5 hidden layers

In [77]:
```
model_relu=Sequential()
model_relu.add(Dense(700, activation='relu', input_shape=(input_dim,),
kernel_initializer=keras.initializers.he_normal(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(500, activation='relu', kernel_initializer=keras.i
```

```
nitializers.he_normal(seed=None)) )
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(250,activation='relu',kernel_initializer=keras.ini
tializers.he_normal(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(150,activation='relu',kernel_initializer=keras.ini
tializers.he_normal(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(70,activation='relu',kernel_initializer=keras.init
ializers.he_normal(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(out_dim, activation='softmax'))
model_relu.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_30 (Dense)             (None, 700)               549500
_________________________________________________________________
batch_normalization_16 (Batc (None, 700)               2800
_________________________________________________________________
dropout_16 (Dropout)         (None, 700)               0
_________________________________________________________________
dense_31 (Dense)             (None, 500)               350500
_________________________________________________________________
batch_normalization_17 (Batc (None, 500)               2000
_________________________________________________________________
dropout_17 (Dropout)         (None, 500)               0
_________________________________________________________________
dense_32 (Dense)             (None, 250)               125250
_________________________________________________________________
batch_normalization_18 (Batc (None, 250)               1000
_________________________________________________________________
dropout_18 (Dropout)         (None, 250)               0
_________________________________________________________________
dense_33 (Dense)             (None, 150)               37650
```

```
dense_33 (Dense)             (None, 150)               37650
_________________________________________________________________
batch_normalization_19 (Batc (None, 150)               600
_________________________________________________________________
dropout_19 (Dropout)         (None, 150)               0
_________________________________________________________________
dense_34 (Dense)             (None, 70)                10570
_________________________________________________________________
batch_normalization_20 (Batc (None, 70)                280
_________________________________________________________________
dropout_20 (Dropout)         (None, 70)                0
_________________________________________________________________
dense_35 (Dense)             (None, 10)                710
=================================================================
Total params: 1,080,860
Trainable params: 1,077,520
Non-trainable params: 3,340
_________________________________________________________________
```

In [78]:
```
model_relu.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_relu.fit(X_train, Y_train, batch_size=Batch_size, epochs=no_epoch, verbose=1, validation_data=(X_test, Y_test))
```

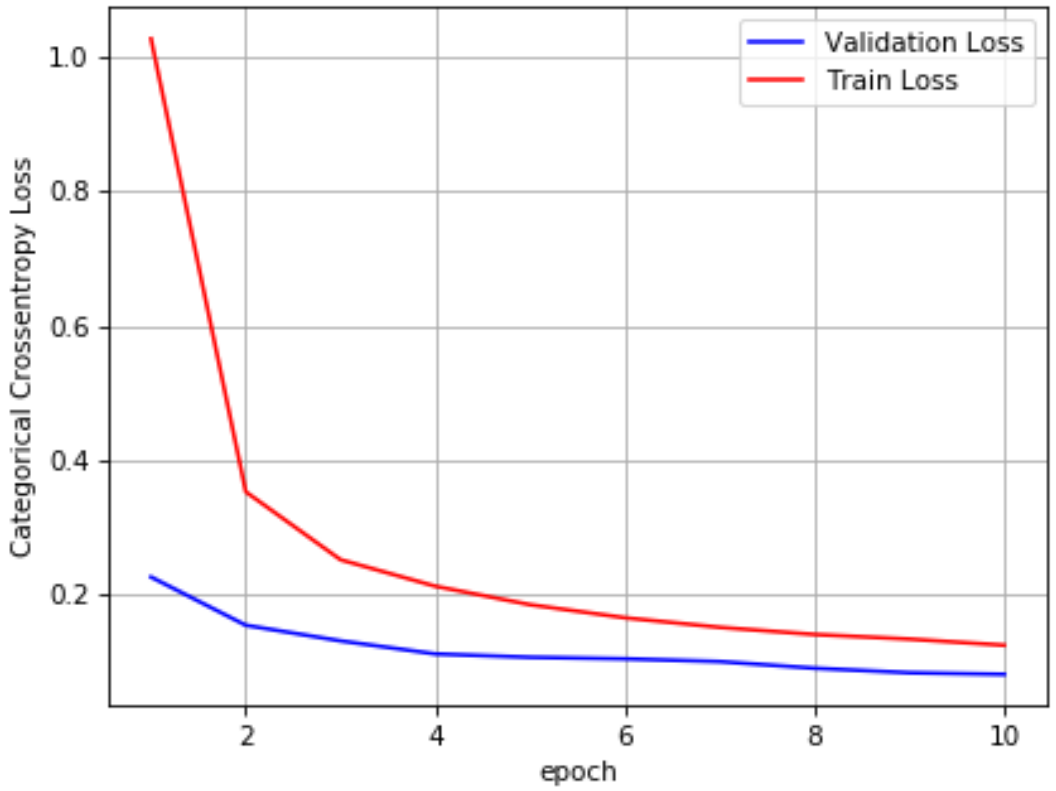
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 22s 365us/step - loss:
1.0271 - acc: 0.6803 - val_loss: 0.2260 - val_acc: 0.9348
Epoch 2/10
60000/60000 [==============================] - 20s 340us/step - loss:
0.3527 - acc: 0.9032 - val_loss: 0.1541 - val_acc: 0.9582
Epoch 3/10
60000/60000 [==============================] - 18s 299us/step - loss:
0.2516 - acc: 0.9312 - val_loss: 0.1309 - val_acc: 0.9655
Epoch 4/10
60000/60000 [==============================] - 18s 304us/step - loss:
0.2122 - acc: 0.9433 - val_loss: 0.1114 - val_acc: 0.9705
Epoch 5/10
60000/60000 [==============================] - 18s 303us/step - loss:
```

```
0.1847 - acc: 0.9512 - val_loss: 0.1065 - val_acc: 0.9735
Epoch 6/10
60000/60000 [==============================] - 18s 303us/step - loss:
0.1653 - acc: 0.9561 - val_loss: 0.1042 - val_acc: 0.9728
Epoch 7/10
60000/60000 [==============================] - 18s 301us/step - loss:
0.1510 - acc: 0.9601 - val_loss: 0.1001 - val_acc: 0.9749
Epoch 8/10
60000/60000 [==============================] - 18s 302us/step - loss:
0.1404 - acc: 0.9623 - val_loss: 0.0902 - val_acc: 0.9767
Epoch 9/10
60000/60000 [==============================] - 18s 304us/step - loss:
0.1336 - acc: 0.9648 - val_loss: 0.0835 - val_acc: 0.9784
Epoch 10/10
60000/60000 [==============================] - 18s 299us/step - loss:
0.1244 - acc: 0.9667 - val_loss: 0.0811 - val_acc: 0.9797
```

In [79]:

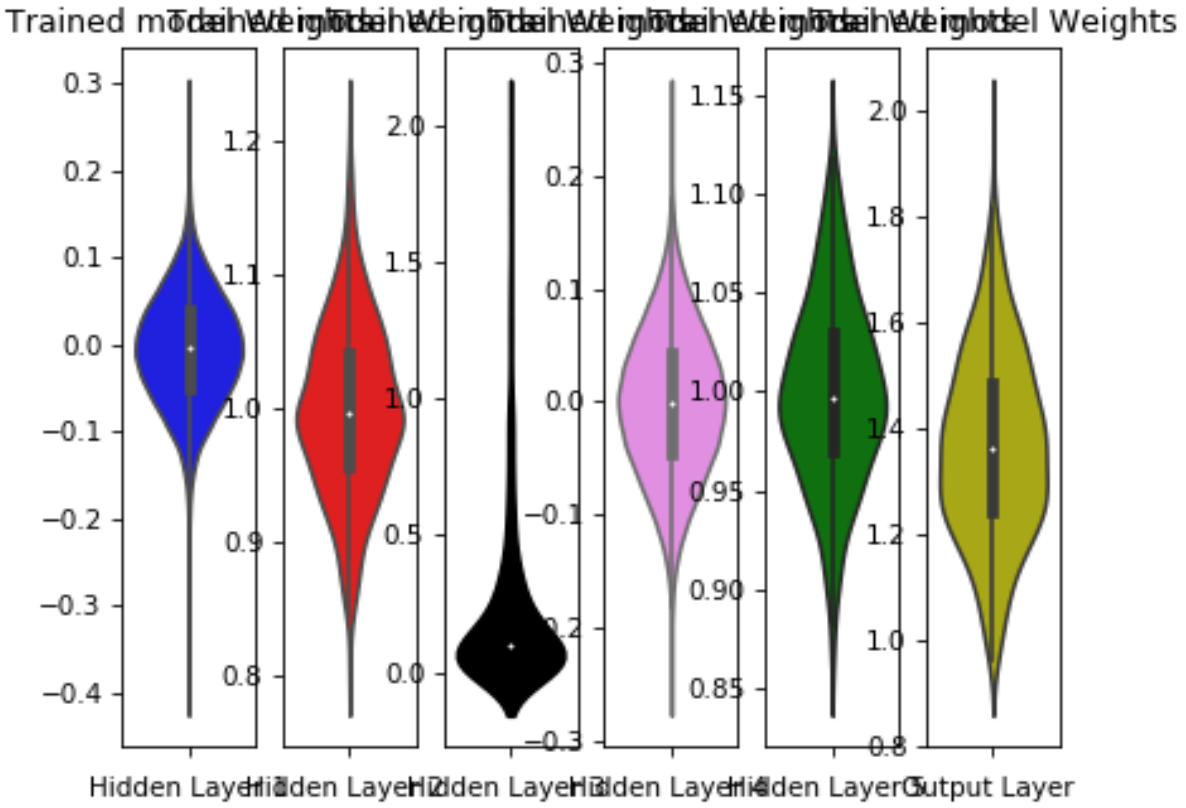
```
score = model_relu.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.08111194996708072
Test accuracy: 0.9797
```



In [80]:
```
w_after = model_relu.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w=w_after[4].flatten().reshape(-1,1)
h4_w=w_after[6].flatten().reshape(-1,1)
h5_w=w_after[8].flatten().reshape(-1,1)
out_w = w_after[10].flatten().reshape(-1,1)


fig = plt.figure()
```

```
plt.title("Weight matrices after model trained")
plt.subplot(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 6, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='black')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='violet')
plt.xlabel('Hidden Layer 4 ')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='green')
plt.xlabel('Hidden Layer 5 ')

plt.subplot(1, 6, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



In [81]:
```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Hidden layers","Test score","Test accuracy","no_of_epoch"]

x.add_row(["2","5.8","98.04","20"])
x.add_row(["3","6.8","98.28","10"])
x.add_row(["5","8.1","97.97","10"])
```

```
print(x)
```

```
+---------------+------------+---------------+-------------+
| Hidden layers | Test score | Test accuracy | no_of_epoch |
+---------------+------------+---------------+-------------+
|       2       |    5.8     |     98.04     |      20     |
|       3       |    6.8     |     98.28     |      10     |
|       5       |    8.1     |     97.97     |      10     |
+---------------+------------+---------------+-------------+
```