1.Trird different MLP architectures on MNIST dataset

Model1.-Hidden layers-2

```
                         * MLP+Relu+Adam
                         * MLP + Batch-Norm on hidden Layers + Adam
Optimizer
                          *MLP + Dropout + AdamOptimizer
```

Model2.-Hidden layers-3

```
                       * MLP+Relu+Adam
                     * MLP + Batch-Norm on hidden Layers + AdamOpti
mizer
                      *MLP + Dropout + AdamOptimizer
```

Model3.-Hidden layers-5

```
                       * MLP+Relu+Adam
                     * MLP + Batch-Norm on hidden Layers + AdamOpti
mizer
                      *MLP + Dropout + AdamOptimizer
```

Model-1

In [0]:
```
from keras.utils import np_utils
from keras.datasets import mnist
```

```
import seaborn as sns
from keras.initializers import RandomNormal
```

In [0]:
```
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 [26]:
```
print("Number of training examples :", X_train.shape[0], "and each imag
e is of shape (%d, %d)"%(X_train.shape[1], X_train.shape[2]))
print("Number of training examples :", X_test.shape[0], "and each image
 is of shape (%d, %d)"%(X_test.shape[1], X_test.shape[2]))
```

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

In [0]:
```
# if you observe the input shape its 3 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.sh
ape[2])
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1]*X_test.shape[2
])
```

In [28]:
```
# after converting the input images from 3d to 2d vectors

print("Number of training examples :", X_train.shape[0], "and each imag
e is of shape (%d)"%(X_train.shape[1]))
print("Number of training examples :", X_test.shape[0], "and each image
 is of shape (%d)"%(X_test.shape[1]))
```

```
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 [29]:
```
# 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  25
5
  247  127    0    0    0    0    0    0    0    0    0    0    0    0   30   36   94  15
4
  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   8
2
   82   56   39    0    0    0    0    0    0    0    0    0    0    0    0   18  219  25
3
  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 15
4
   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 24
1
 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 18
7
   0   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 25
3
 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 25
3
 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 19
 5
   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   1
 6
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 0
    0   0   0   0   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 norma
lize the data
# X => (X - Xmin)/(Xmax-Xmin) = X/255

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

In [31]:

```
# 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 [32]:

```
# 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]:

```
# Softmax

from keras.models import Sequential
from keras.layers import Dense, Activation
```

In [0]:

```
# some model parameters

output_dim = 10
input_dim = X_train.shape[1]
```

```
batch_size = 128
nb_epoch = 20
```

MLP+Relu+Adam

In [35]:
```
model_relu = Sequential()
model_relu.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.074, seed=None)))
model_relu.add(Dense(328, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.196, seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_4 (Dense)              (None, 512)               401920    
_________________________________________________________________
dense_5 (Dense)              (None, 328)               168264    
_________________________________________________________________
dense_6 (Dense)              (None, 10)                3290      
=================================================================
Total params: 573,474
Trainable params: 573,474
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 15s 257us/step - loss: 0.2402 - acc: 0.9284 - val_loss: 0.1193 - val_acc: 0.9615
```

```
Epoch 2/20
60000/60000 [==============================] - 15s 250us/step - loss:
0.0812 - acc: 0.9753 - val_loss: 0.0839 - val_acc: 0.9747
Epoch 3/20
60000/60000 [==============================] - 15s 254us/step - loss:
0.0502 - acc: 0.9842 - val_loss: 0.0811 - val_acc: 0.9748
Epoch 4/20
60000/60000 [==============================] - 15s 247us/step - loss:
0.0346 - acc: 0.9892 - val_loss: 0.0850 - val_acc: 0.9766
Epoch 5/20
60000/60000 [==============================] - 15s 245us/step - loss:
0.0248 - acc: 0.9919 - val_loss: 0.0797 - val_acc: 0.9769
Epoch 6/20
60000/60000 [==============================] - 15s 247us/step - loss:
0.0200 - acc: 0.9931 - val_loss: 0.0847 - val_acc: 0.9746
Epoch 7/20
60000/60000 [==============================] - 15s 248us/step - loss:
0.0181 - acc: 0.9940 - val_loss: 0.0835 - val_acc: 0.9799
Epoch 8/20
60000/60000 [==============================] - 15s 248us/step - loss:
0.0149 - acc: 0.9950 - val_loss: 0.1133 - val_acc: 0.9742
Epoch 9/20
60000/60000 [==============================] - 15s 253us/step - loss:
0.0168 - acc: 0.9944 - val_loss: 0.0996 - val_acc: 0.9752
Epoch 10/20
60000/60000 [==============================] - 15s 248us/step - loss:
0.0128 - acc: 0.9956 - val_loss: 0.0858 - val_acc: 0.9798
Epoch 11/20
60000/60000 [==============================] - 15s 249us/step - loss:
0.0124 - acc: 0.9957 - val_loss: 0.1051 - val_acc: 0.9774
Epoch 12/20
60000/60000 [==============================] - 15s 249us/step - loss:
0.0113 - acc: 0.9966 - val_loss: 0.0935 - val_acc: 0.9794
Epoch 13/20
60000/60000 [==============================] - 15s 247us/step - loss:
0.0126 - acc: 0.9954 - val_loss: 0.1069 - val_acc: 0.9779
Epoch 14/20
60000/60000 [==============================] - 15s 252us/step - loss:
0.0107 - acc: 0.9964 - val_loss: 0.0956 - val_acc: 0.9801
```

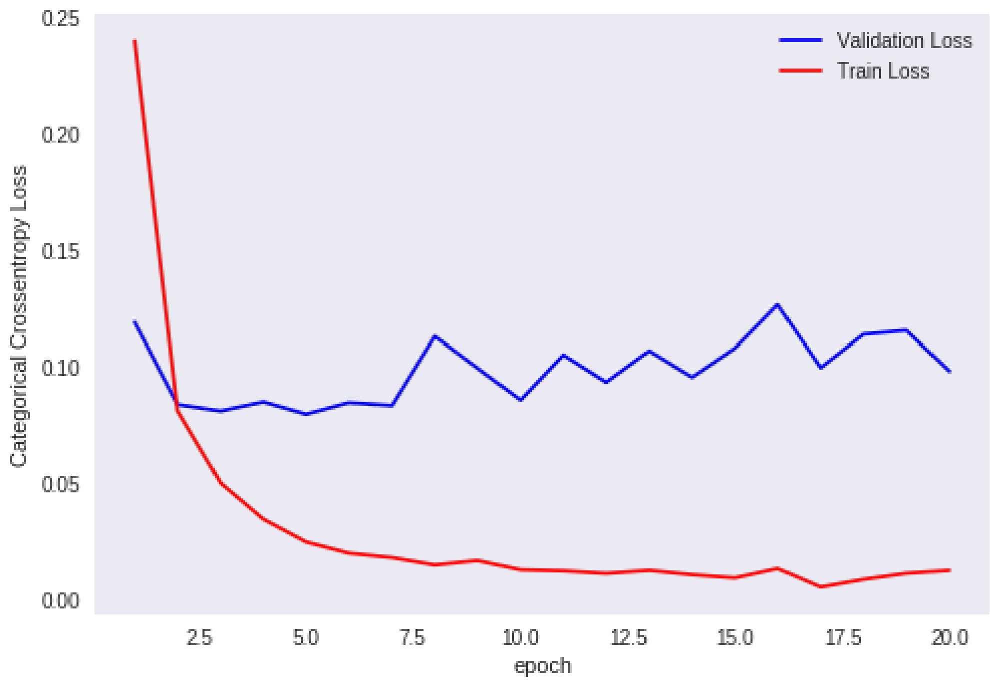
```
Epoch 15/20
60000/60000 [==============================] - 15s 249us/step - loss: 
0.0093 - acc: 0.9970 - val_loss: 0.1080 - val_acc: 0.9779
Epoch 16/20
60000/60000 [==============================] - 15s 256us/step - loss: 
0.0133 - acc: 0.9957 - val_loss: 0.1268 - val_acc: 0.9759
Epoch 17/20
60000/60000 [==============================] - 15s 251us/step - loss: 
0.0054 - acc: 0.9983 - val_loss: 0.0997 - val_acc: 0.9809
Epoch 18/20
60000/60000 [==============================] - 15s 249us/step - loss: 
0.0087 - acc: 0.9974 - val_loss: 0.1141 - val_acc: 0.9793
Epoch 19/20
60000/60000 [==============================] - 15s 252us/step - loss: 
0.0113 - acc: 0.9965 - val_loss: 0.1158 - val_acc: 0.9769
Epoch 20/20
60000/60000 [==============================] - 15s 246us/step - loss: 
0.0125 - acc: 0.9961 - val_loss: 0.0982 - val_acc: 0.9806
```

In [36]:

```
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,nb_epoch+1))

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epo
chs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

# we will get val_loss and val_acc only when you pass the paramter vali
dation_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 epochs


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

```
Test score: 0.09823002113027446
Test accuracy: 0.9806
```



In [37]:
```
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)
```

```python
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()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: Futu
reWarning: remove_na is deprecated and is a private function. Do not us
e.
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: Futu
reWarning: remove_na is deprecated and is a private function. Do not us
e.
  violin_data = remove_na(group_data)
```

MLP + Batch-Norm on hidden Layers + AdamOptimizer

In [38]:

```
# Multilayer perceptron

# https://intoli.com/blog/neural-network-initialization/
# If we sample weights from a normal distribution N(0,σ) we satisfy thi
s condition with σ=√(2/(ni+ni+1).
# h1 =>  σ=√(2/(ni+ni+1) = 0.039  => N(0,σ) = N(0,0.039)
# h2 =>  σ=√(2/(ni+ni+1) = 0.055  => N(0,σ) = N(0,0.055)
# h1 =>  σ=√(2/(ni+ni+1) = 0.120  => N(0,σ) = N(0,0.120)

from keras.layers.normalization import BatchNormalization

model_batch = Sequential()
```

```python
model_batch.add(Dense(512, activation='relu', input_shape=(input_dim,),
 kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_batch.add(BatchNormalization())

model_batch.add(Dense(328, activation='relu', kernel_initializer=Random
Normal(mean=0.0, stddev=0.055, seed=None)) )
model_batch.add(BatchNormalization())

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


model_batch.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_7 (Dense)              (None, 512)               401920    
_________________________________________________________________
batch_normalization_1 (Batch (None, 512)               2048      
_________________________________________________________________
dense_8 (Dense)              (None, 328)               168264    
_________________________________________________________________
batch_normalization_2 (Batch (None, 328)               1312      
_________________________________________________________________
dense_9 (Dense)              (None, 10)                3290      
=================================================================
Total params: 576,834
Trainable params: 575,154
Non-trainable params: 1,680
_________________________________________________________________
```

In [39]:

```python
model_batch.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

history = model_batch.fit(X_train, Y_train, batch_size=batch_size, epoc
hs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 19s 318us/step - loss:
```

```
0.1729 - acc: 0.9471 - val_loss: 0.1041 - val_acc: 0.9653
Epoch 2/20
60000/60000 [==============================] - 17s 288us/step - loss:
0.0704 - acc: 0.9781 - val_loss: 0.0858 - val_acc: 0.9717
Epoch 3/20
60000/60000 [==============================] - 17s 291us/step - loss:
0.0458 - acc: 0.9849 - val_loss: 0.0808 - val_acc: 0.9743
Epoch 4/20
60000/60000 [==============================] - 17s 285us/step - loss:
0.0345 - acc: 0.9888 - val_loss: 0.0789 - val_acc: 0.9786
Epoch 5/20
60000/60000 [==============================] - 17s 290us/step - loss:
0.0288 - acc: 0.9903 - val_loss: 0.0848 - val_acc: 0.9748
Epoch 6/20
60000/60000 [==============================] - 18s 293us/step - loss:
0.0244 - acc: 0.9917 - val_loss: 0.0865 - val_acc: 0.9747
Epoch 7/20
60000/60000 [==============================] - 17s 289us/step - loss:
0.0180 - acc: 0.9941 - val_loss: 0.0724 - val_acc: 0.9795
Epoch 8/20
60000/60000 [==============================] - 17s 288us/step - loss:
0.0163 - acc: 0.9945 - val_loss: 0.0768 - val_acc: 0.9778
Epoch 9/20
60000/60000 [==============================] - 17s 286us/step - loss:
0.0176 - acc: 0.9941 - val_loss: 0.0811 - val_acc: 0.9786
Epoch 10/20
60000/60000 [==============================] - 17s 290us/step - loss:
0.0177 - acc: 0.9938 - val_loss: 0.0786 - val_acc: 0.9802
Epoch 11/20
60000/60000 [==============================] - 18s 292us/step - loss:
0.0136 - acc: 0.9955 - val_loss: 0.0878 - val_acc: 0.9791
Epoch 12/20
60000/60000 [==============================] - 17s 283us/step - loss:
0.0129 - acc: 0.9955 - val_loss: 0.0845 - val_acc: 0.9785
Epoch 13/20
60000/60000 [==============================] - 17s 283us/step - loss:
0.0118 - acc: 0.9963 - val_loss: 0.0741 - val_acc: 0.9816
Epoch 14/20
60000/60000 [==============================] - 17s 291us/step - loss:
```

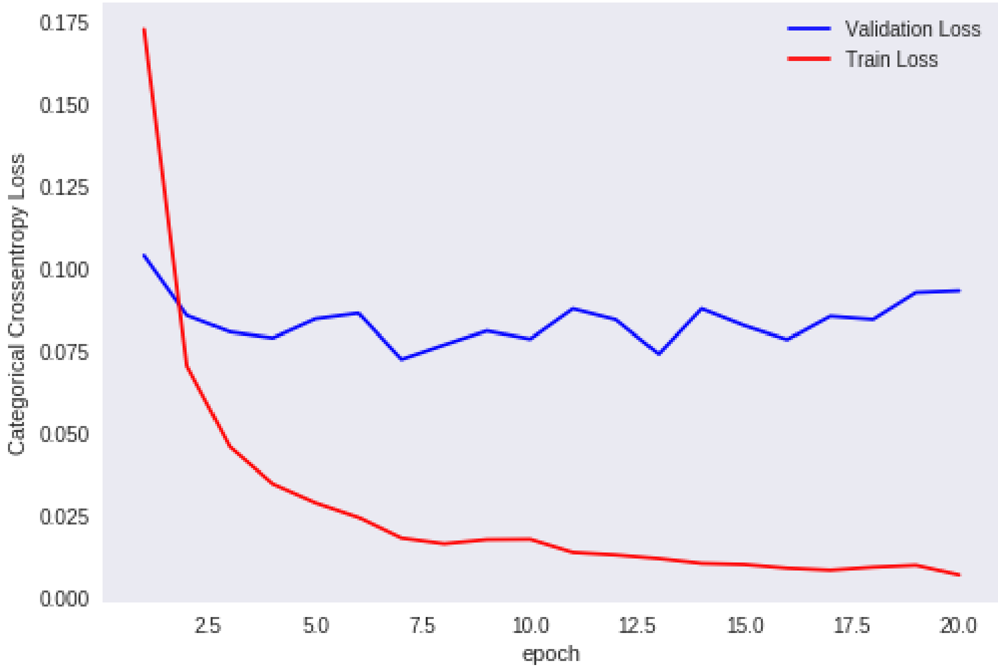
```
0.0103 - acc: 0.9965 - val_loss: 0.0878 - val_acc: 0.9800
Epoch 15/20
60000/60000 [==============================] - 17s 290us/step - loss:
0.0100 - acc: 0.9967 - val_loss: 0.0827 - val_acc: 0.9799
Epoch 16/20
60000/60000 [==============================] - 18s 293us/step - loss:
0.0088 - acc: 0.9970 - val_loss: 0.0784 - val_acc: 0.9818
Epoch 17/20
60000/60000 [==============================] - 17s 289us/step - loss:
0.0082 - acc: 0.9971 - val_loss: 0.0855 - val_acc: 0.9810
Epoch 18/20
60000/60000 [==============================] - 18s 297us/step - loss:
0.0091 - acc: 0.9970 - val_loss: 0.0845 - val_acc: 0.9807
Epoch 19/20
60000/60000 [==============================] - 17s 289us/step - loss:
0.0097 - acc: 0.9966 - val_loss: 0.0927 - val_acc: 0.9796
Epoch 20/20
60000/60000 [==============================] - 17s 290us/step - loss:
0.0068 - acc: 0.9977 - val_loss: 0.0933 - val_acc: 0.9789
```

In [40]:

```
score = model_batch.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))

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epo
chs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

# we will get val_loss and val_acc only when you pass the paramter vali
dation_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 epochs

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

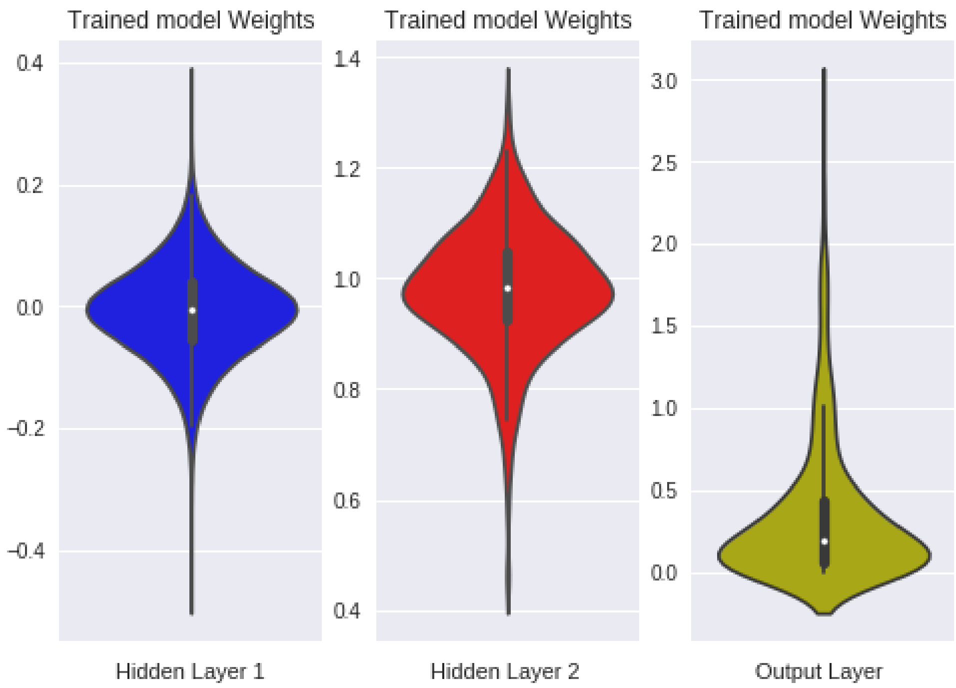
```
Test score: 0.09328906010493092
Test accuracy: 0.9789
```



In [41]:
```
w_after = model_batch.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)
```

```python
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()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: Futu
reWarning: remove_na is deprecated and is a private function. Do not us
e.
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: Futu
reWarning: remove_na is deprecated and is a private function. Do not us
e.
  violin_data = remove_na(group_data)
```



In [0]:
```
# MLP + Dropout + AdamOptimizer
```

MLP + Dropout + AdamOptimizer

In [43]:
```
# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batc
hnormalization-function-in-keras

from keras.layers import Dropout

model_drop = Sequential()

model_drop.add(Dense(512, activation='relu', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))
```

```
model_drop.add(Dense(328, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.055, seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

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


model_drop.summary()
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:3445: calling dropout (from tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_10 (Dense)             (None, 512)               401920    
_________________________________________________________________
batch_normalization_3 (Batch (None, 512)               2048      
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_11 (Dense)             (None, 328)               168264    
_________________________________________________________________
batch_normalization_4 (Batch (None, 328)               1312      
_________________________________________________________________
dropout_2 (Dropout)          (None, 328)               0         
_________________________________________________________________
dense_12 (Dense)             (None, 10)                3290      
=================================================================
Total params: 576,834
Trainable params: 575,154
Non-trainable params: 1,680
_________________________________________________________________
```

In [44]:
```
model_drop.compile(optimizer='adam', loss='categorical_crossentropy', m
etrics=['accuracy'])

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 21s 352us/step - loss:
0.3750 - acc: 0.8879 - val_loss: 0.1417 - val_acc: 0.9566
Epoch 2/20
60000/60000 [==============================] - 19s 310us/step - loss:
0.1835 - acc: 0.9439 - val_loss: 0.0933 - val_acc: 0.9707
Epoch 3/20
60000/60000 [==============================] - 19s 310us/step - loss:
0.1466 - acc: 0.9540 - val_loss: 0.0812 - val_acc: 0.9745
Epoch 4/20
60000/60000 [==============================] - 18s 305us/step - loss:
0.1268 - acc: 0.9608 - val_loss: 0.0771 - val_acc: 0.9764
Epoch 5/20
60000/60000 [==============================] - 18s 307us/step - loss:
0.1106 - acc: 0.9657 - val_loss: 0.0709 - val_acc: 0.9782
Epoch 6/20
60000/60000 [==============================] - 19s 313us/step - loss:
0.0990 - acc: 0.9682 - val_loss: 0.0745 - val_acc: 0.9774
Epoch 7/20
60000/60000 [==============================] - 18s 308us/step - loss:
0.0928 - acc: 0.9710 - val_loss: 0.0687 - val_acc: 0.9788
Epoch 8/20
60000/60000 [==============================] - 19s 313us/step - loss:
0.0862 - acc: 0.9722 - val_loss: 0.0633 - val_acc: 0.9806
Epoch 9/20
60000/60000 [==============================] - 18s 308us/step - loss:
0.0804 - acc: 0.9744 - val_loss: 0.0614 - val_acc: 0.9813
Epoch 10/20
60000/60000 [==============================] - 19s 312us/step - loss:
0.0767 - acc: 0.9751 - val_loss: 0.0593 - val_acc: 0.9812
Epoch 11/20
60000/60000 [==============================] - 19s 313us/step - loss:
0.0752 - acc: 0.9756 - val_loss: 0.0563 - val_acc: 0.9829
```

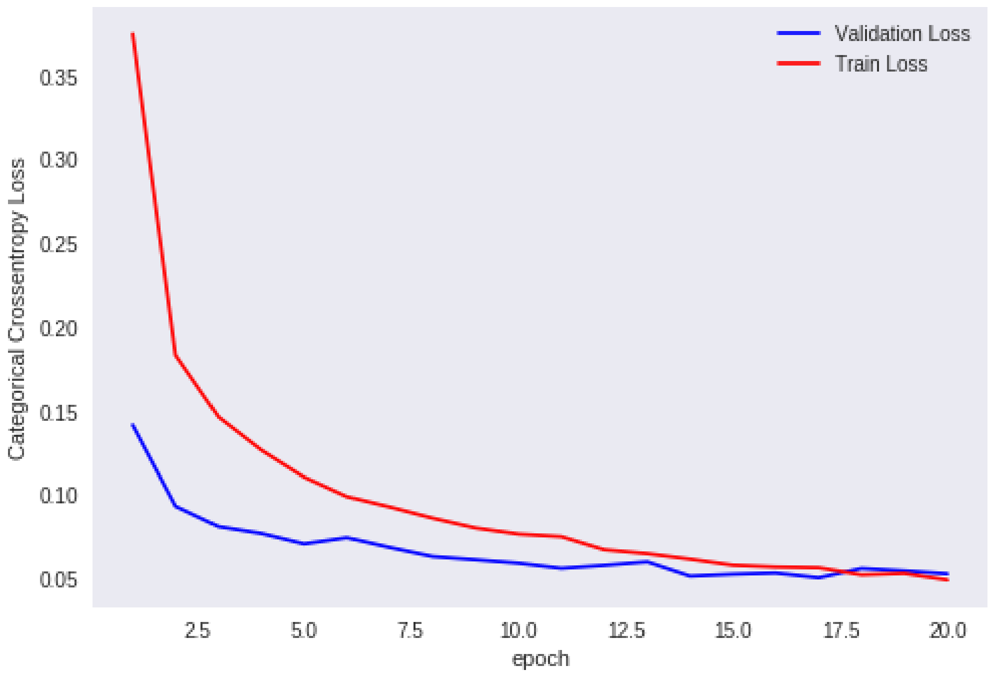
```
Epoch 12/20
60000/60000 [==============================] - 19s 314us/step - loss:
0.0674 - acc: 0.9778 - val_loss: 0.0580 - val_acc: 0.9819
Epoch 13/20
60000/60000 [==============================] - 19s 321us/step - loss:
0.0650 - acc: 0.9796 - val_loss: 0.0601 - val_acc: 0.9823
Epoch 14/20
60000/60000 [==============================] - 19s 312us/step - loss:
0.0617 - acc: 0.9795 - val_loss: 0.0516 - val_acc: 0.9839
Epoch 15/20
60000/60000 [==============================] - 19s 310us/step - loss:
0.0581 - acc: 0.9811 - val_loss: 0.0527 - val_acc: 0.9838
Epoch 16/20
60000/60000 [==============================] - 18s 306us/step - loss:
0.0571 - acc: 0.9818 - val_loss: 0.0535 - val_acc: 0.9842
Epoch 17/20
60000/60000 [==============================] - 18s 306us/step - loss:
0.0566 - acc: 0.9818 - val_loss: 0.0506 - val_acc: 0.9849
Epoch 18/20
60000/60000 [==============================] - 19s 309us/step - loss:
0.0524 - acc: 0.9832 - val_loss: 0.0562 - val_acc: 0.9846
Epoch 19/20
60000/60000 [==============================] - 19s 310us/step - loss:
0.0532 - acc: 0.9830 - val_loss: 0.0548 - val_acc: 0.9839
Epoch 20/20
60000/60000 [==============================] - 19s 312us/step - loss:
0.0493 - acc: 0.9839 - val_loss: 0.0530 - val_acc: 0.9852
```

In [45]:

```
score = model_drop.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))

# print(history.history.keys())
```

```
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epo
chs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

# we will get val_loss and val_acc only when you pass the paramter vali
dation_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 epochs

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

```
Test score: 0.052987941403864536
Test accuracy: 0.9852
```



In [46]:
```
w_after = model_drop.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()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: Futu
reWarning: remove_na is deprecated and is a private function. Do not us
e.
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: Futu
reWarning: remove_na is deprecated and is a private function. Do not us
e.
  violin_data = remove_na(group_data)
```

In [0]:
```
# Part-2, 3 hidden layers
```

Model-2 (3-hidden layer)

In [0]:
```
model_relu = Sequential()
model_relu.add(Dense(364, activation='relu', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.074, seed=None)))
model_relu.add(Dense(52, activation='relu', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.196, seed=None)) )
model_relu.add(Dense(32, activation='relu', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.25, seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_10 (Dense)             (None, 364)               285740    
_________________________________________________________________
dense_11 (Dense)             (None, 52)                18980     
_________________________________________________________________
dense_12 (Dense)             (None, 32)                1696      
_________________________________________________________________
dense_13 (Dense)             (None, 10)                330       
=================================================================
Total params: 306,746
Trainable params: 306,746
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 92us/step - loss: 0.3
205 - acc: 0.9021 - val_loss: 0.1441 - val_acc: 0.9566
```

```
Epoch 2/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.1
100 - acc: 0.9678 - val_loss: 0.1014 - val_acc: 0.9689
Epoch 3/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0
729 - acc: 0.9778 - val_loss: 0.0852 - val_acc: 0.9733
Epoch 4/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0
540 - acc: 0.9836 - val_loss: 0.0883 - val_acc: 0.9740
Epoch 5/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0
389 - acc: 0.9877 - val_loss: 0.0869 - val_acc: 0.9747
Epoch 6/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0
331 - acc: 0.9891 - val_loss: 0.0861 - val_acc: 0.9749
Epoch 7/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0
234 - acc: 0.9932 - val_loss: 0.0889 - val_acc: 0.9779
Epoch 8/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0
255 - acc: 0.9921 - val_loss: 0.0939 - val_acc: 0.9758
Epoch 9/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0
200 - acc: 0.9935 - val_loss: 0.0993 - val_acc: 0.9745
Epoch 10/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0
147 - acc: 0.9951 - val_loss: 0.0782 - val_acc: 0.9792
Epoch 11/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0
181 - acc: 0.9939 - val_loss: 0.1120 - val_acc: 0.9721
Epoch 12/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0
132 - acc: 0.9958 - val_loss: 0.1055 - val_acc: 0.9753
Epoch 13/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0
096 - acc: 0.9968 - val_loss: 0.1003 - val_acc: 0.9767
Epoch 14/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0
150 - acc: 0.9949 - val_loss: 0.1093 - val_acc: 0.9763
```

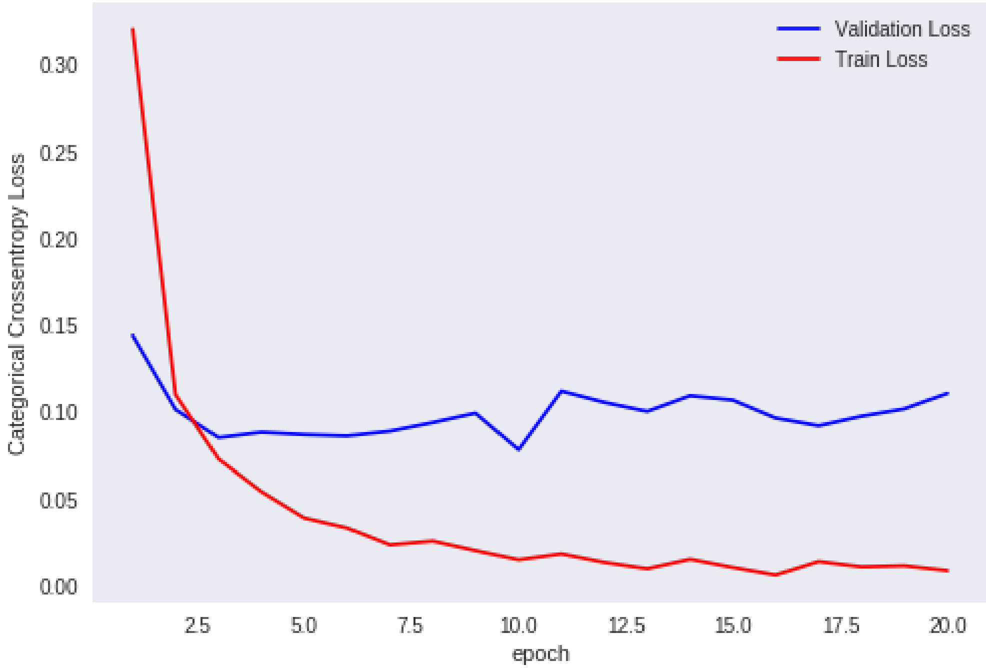
```
Epoch 15/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0
103 - acc: 0.9967 - val_loss: 0.1068 - val_acc: 0.9768
Epoch 16/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0
059 - acc: 0.9981 - val_loss: 0.0965 - val_acc: 0.9803
Epoch 17/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0
137 - acc: 0.9958 - val_loss: 0.0920 - val_acc: 0.9787
Epoch 18/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0
106 - acc: 0.9964 - val_loss: 0.0975 - val_acc: 0.9804
Epoch 19/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0
112 - acc: 0.9965 - val_loss: 0.1017 - val_acc: 0.9781
Epoch 20/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0
085 - acc: 0.9972 - val_loss: 0.1107 - val_acc: 0.9785
```

In [0]:

```
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,nb_epoch+1))

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epo
chs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

# we will get val_loss and val_acc only when you pass the paramter vali
dation_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 epochs


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

```
Test score: 0.1106776758507105
Test accuracy: 0.9785
```



In [0]:
```
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='r')
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()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: Futu
reWarning: remove_na is deprecated and is a private function. Do not us
e.
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: Futu
reWarning: remove_na is deprecated and is a private function. Do not us
e.
  violin_data = remove_na(group_data)
```

In [0]:
```
#MLP + Batch-Norm on hidden Layers + AdamOptimizer
```

MLP + Batch-Norm on hidden Layers + AdamOptimizer

In [0]:
```
from keras.layers.normalization import BatchNormalization

model_batch = Sequential()

model_batch.add(Dense(364, activation='relu', input_shape=(input_dim,),
 kernel_initializer=RandomNormal(mean=0.0, stddev=0.0741, seed=None)))
model_batch.add(BatchNormalization())

model_batch.add(Dense(64, activation='relu', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.196, seed=None)) )
```

```
model_batch.add(BatchNormalization())

model_batch.add(Dense(32, activation='relu', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.25, seed=None)) )
model_batch.add(BatchNormalization())

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


model_batch.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_14 (Dense)             (None, 364)               285740
_________________________________________________________________
batch_normalization_5 (Batch (None, 364)               1456
_________________________________________________________________
dense_15 (Dense)             (None, 64)                23360
_________________________________________________________________
batch_normalization_6 (Batch (None, 64)                256
_________________________________________________________________
dense_16 (Dense)             (None, 32)                2080
_________________________________________________________________
batch_normalization_7 (Batch (None, 32)                128
_________________________________________________________________
dense_17 (Dense)             (None, 10)                330
=================================================================
Total params: 313,350
Trainable params: 312,430
Non-trainable params: 920
_________________________________________________________________
```

In [0]:

```
model_batch.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

history = model_batch.fit(X_train, Y_train, batch_size=batch_size, epoc
hs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
```

```
Epoch 1/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.
2903 - acc: 0.9195 - val_loss: 0.1229 - val_acc: 0.9631
Epoch 2/20
60000/60000 [==============================] - 6s 102us/step - loss: 0.
0970 - acc: 0.9719 - val_loss: 0.0946 - val_acc: 0.9692
Epoch 3/20
60000/60000 [==============================] - 6s 102us/step - loss: 0.
0614 - acc: 0.9814 - val_loss: 0.0874 - val_acc: 0.9721
Epoch 4/20
60000/60000 [==============================] - 6s 103us/step - loss: 0.
0438 - acc: 0.9870 - val_loss: 0.0809 - val_acc: 0.9753
Epoch 5/20
60000/60000 [==============================] - 6s 102us/step - loss: 0.
0334 - acc: 0.9895 - val_loss: 0.0796 - val_acc: 0.9752
Epoch 6/20
60000/60000 [==============================] - 6s 103us/step - loss: 0.
0252 - acc: 0.9922 - val_loss: 0.0915 - val_acc: 0.9731
Epoch 7/20
60000/60000 [==============================] - 6s 102us/step - loss: 0.
0243 - acc: 0.9923 - val_loss: 0.0801 - val_acc: 0.9780
Epoch 8/20
60000/60000 [==============================] - 6s 103us/step - loss: 0.
0183 - acc: 0.9942 - val_loss: 0.0872 - val_acc: 0.9742
Epoch 9/20
60000/60000 [==============================] - 6s 103us/step - loss: 0.
0176 - acc: 0.9945 - val_loss: 0.0913 - val_acc: 0.9754
Epoch 10/20
60000/60000 [==============================] - 6s 103us/step - loss: 0.
0165 - acc: 0.9946 - val_loss: 0.0815 - val_acc: 0.9778
Epoch 11/20
60000/60000 [==============================] - 6s 102us/step - loss: 0.
0144 - acc: 0.9954 - val_loss: 0.0767 - val_acc: 0.9798
Epoch 12/20
60000/60000 [==============================] - 6s 101us/step - loss: 0.
0130 - acc: 0.9956 - val_loss: 0.0828 - val_acc: 0.9787
Epoch 13/20
60000/60000 [==============================] - 6s 102us/step - loss: 0.
0138 - acc: 0.9955 - val_loss: 0.0961 - val_acc: 0.9756
Epoch 14/20
```

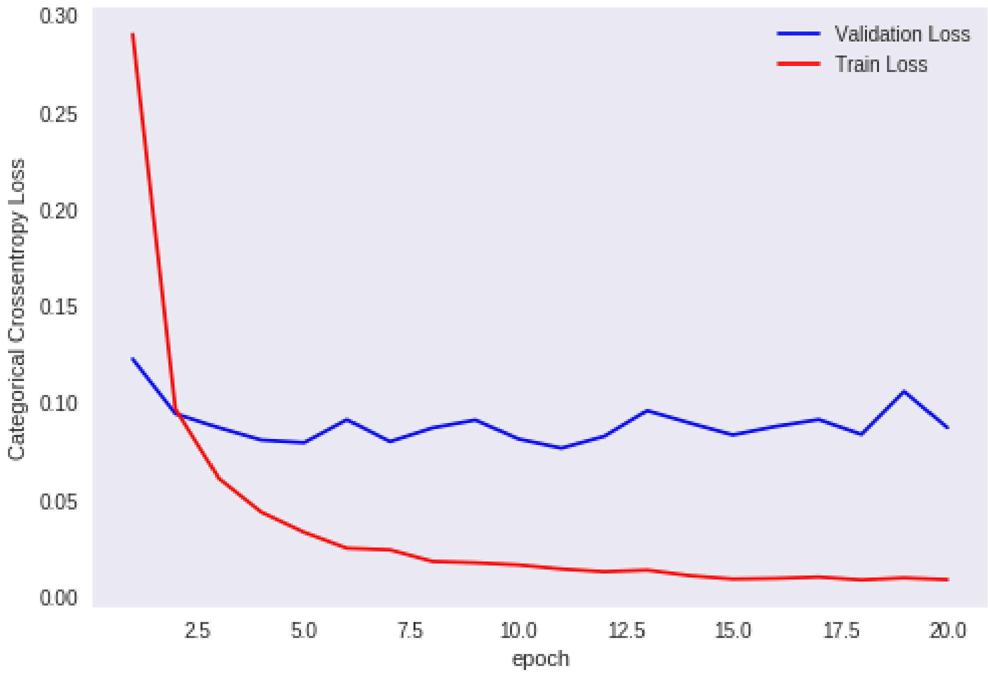
```
60000/60000 [==============================] - 6s 102us/step - loss: 0.
0109 - acc: 0.9964 - val_loss: 0.0897 - val_acc: 0.9762
Epoch 15/20
60000/60000 [==============================] - 6s 102us/step - loss: 0.
0092 - acc: 0.9970 - val_loss: 0.0836 - val_acc: 0.9794
Epoch 16/20
60000/60000 [==============================] - 6s 106us/step - loss: 0.
0095 - acc: 0.9968 - val_loss: 0.0880 - val_acc: 0.9778
Epoch 17/20
60000/60000 [==============================] - 6s 102us/step - loss: 0.
0102 - acc: 0.9969 - val_loss: 0.0916 - val_acc: 0.9774
Epoch 18/20
60000/60000 [==============================] - 6s 101us/step - loss: 0.
0087 - acc: 0.9970 - val_loss: 0.0838 - val_acc: 0.9788
Epoch 19/20
60000/60000 [==============================] - 6s 101us/step - loss: 0.
0098 - acc: 0.9967 - val_loss: 0.1061 - val_acc: 0.9767
Epoch 20/20
60000/60000 [==============================] - 6s 102us/step - loss: 0.
0089 - acc: 0.9970 - val_loss: 0.0874 - val_acc: 0.9799
```

In [0]:

```
score = model_batch.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))

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epo
chs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

# we will get val_loss and val_acc only when you pass the paramter vali
dation_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 epochs

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

```
Test score: 0.08735586702148576
Test accuracy: 0.9799
```



In [0]:
```
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='r')
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()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: Futu
reWarning: remove_na is deprecated and is a private function. Do not us
e.
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: Futu
reWarning: remove_na is deprecated and is a private function. Do not us
e.
  violin_data = remove_na(group_data)
```

In [0]:
```
#MLP + Dropout + AdamOptimizer
```

MLP + Dropout + AdamOptimizer

In [0]:
```
# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batc
hnormalization-function-in-keras

from keras.layers import Dropout

model_drop = Sequential()

model_drop.add(Dense(364, activation='relu', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.0741, seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))
```

```
model_drop.add(Dense(64, activation='relu', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.196, seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(32, activation='relu', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.25, seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

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


model_drop.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_18 (Dense)             (None, 364)               285740    
_________________________________________________________________
batch_normalization_8 (Batch (None, 364)               1456      
_________________________________________________________________
dropout_3 (Dropout)          (None, 364)               0         
_________________________________________________________________
dense_19 (Dense)             (None, 64)                23360     
_________________________________________________________________
batch_normalization_9 (Batch (None, 64)                256       
_________________________________________________________________
dropout_4 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_20 (Dense)             (None, 32)                2080      
_________________________________________________________________
batch_normalization_10 (Batc (None, 32)                128       
_________________________________________________________________
dropout_5 (Dropout)          (None, 32)                0         
_________________________________________________________________
dense_21 (Dense)             (None, 10)                330       
=================================================================
Total params: 313,350
```

```
Trainable params: 312,430
Non-trainable params: 920
_________________________________________________________________
```

In [0]:
```
model_drop.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 8s 138us/step - loss: 1.1575 - acc: 0.6323 - val_loss: 0.2918 - val_acc: 0.9161
Epoch 2/20
60000/60000 [==============================] - 7s 110us/step - loss: 0.5283 - acc: 0.8461 - val_loss: 0.1892 - val_acc: 0.9437
Epoch 3/20
60000/60000 [==============================] - 7s 111us/step - loss: 0.3873 - acc: 0.8930 - val_loss: 0.1542 - val_acc: 0.9548
Epoch 4/20
60000/60000 [==============================] - 7s 111us/step - loss: 0.3210 - acc: 0.9134 - val_loss: 0.1362 - val_acc: 0.9617
Epoch 5/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.2724 - acc: 0.9278 - val_loss: 0.1206 - val_acc: 0.9668
Epoch 6/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.2472 - acc: 0.9356 - val_loss: 0.1091 - val_acc: 0.9697
Epoch 7/20
60000/60000 [==============================] - 7s 111us/step - loss: 0.2224 - acc: 0.9416 - val_loss: 0.1037 - val_acc: 0.9702
Epoch 8/20
60000/60000 [==============================] - 7s 110us/step - loss: 0.2107 - acc: 0.9449 - val_loss: 0.1026 - val_acc: 0.9717
Epoch 9/20
60000/60000 [==============================] - 7s 111us/step - loss: 0.1958 - acc: 0.9492 - val_loss: 0.0967 - val_acc: 0.9740
Epoch 10/20
```

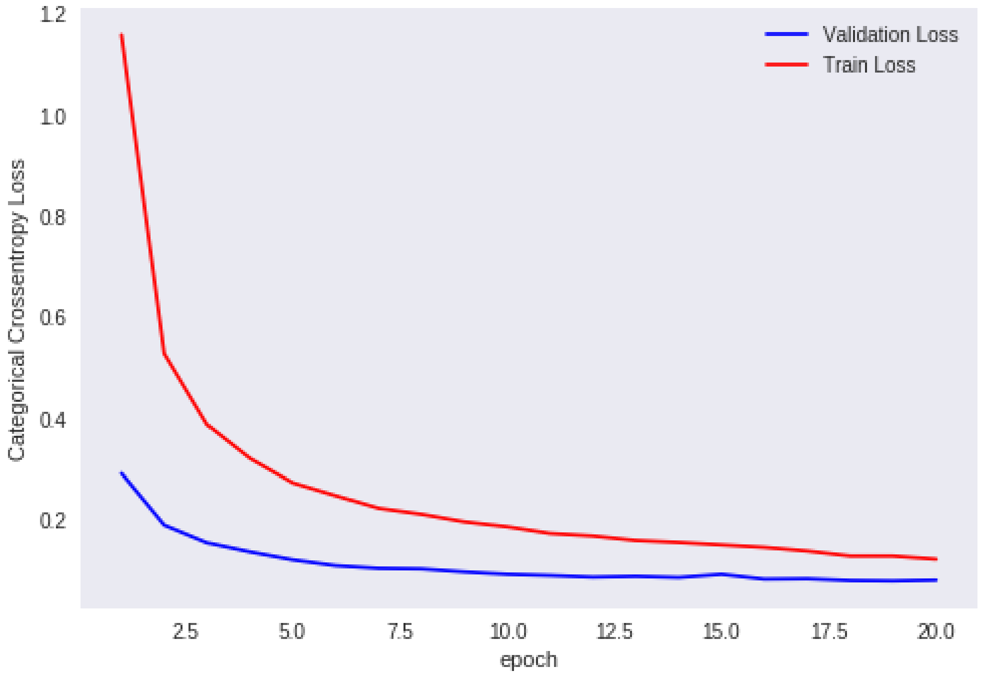
```
60000/60000 [==============================] - 7s 110us/step - loss: 0.
1859 - acc: 0.9525 - val_loss: 0.0920 - val_acc: 0.9740
Epoch 11/20
60000/60000 [==============================] - 7s 110us/step - loss: 0.
1728 - acc: 0.9548 - val_loss: 0.0896 - val_acc: 0.9754
Epoch 12/20
60000/60000 [==============================] - 7s 110us/step - loss: 0.
1677 - acc: 0.9570 - val_loss: 0.0863 - val_acc: 0.9765
Epoch 13/20
60000/60000 [==============================] - 7s 112us/step - loss: 0.
1589 - acc: 0.9589 - val_loss: 0.0879 - val_acc: 0.9757
Epoch 14/20
60000/60000 [==============================] - 7s 110us/step - loss: 0.
1549 - acc: 0.9610 - val_loss: 0.0856 - val_acc: 0.9771
Epoch 15/20
60000/60000 [==============================] - 7s 110us/step - loss: 0.
1501 - acc: 0.9614 - val_loss: 0.0920 - val_acc: 0.9761
Epoch 16/20
60000/60000 [==============================] - 7s 110us/step - loss: 0.
1452 - acc: 0.9632 - val_loss: 0.0823 - val_acc: 0.9795
Epoch 17/20
60000/60000 [==============================] - 7s 111us/step - loss: 0.
1379 - acc: 0.9642 - val_loss: 0.0830 - val_acc: 0.9786
Epoch 18/20
60000/60000 [==============================] - 7s 111us/step - loss: 0.
1282 - acc: 0.9666 - val_loss: 0.0796 - val_acc: 0.9784
Epoch 19/20
60000/60000 [==============================] - 7s 111us/step - loss: 0.
1281 - acc: 0.9672 - val_loss: 0.0788 - val_acc: 0.9801
Epoch 20/20
60000/60000 [==============================] - 7s 111us/step - loss: 0.
1218 - acc: 0.9686 - val_loss: 0.0804 - val_acc: 0.9799
```

In [0]:

```
score = model_drop.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))

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

# 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 epochs

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

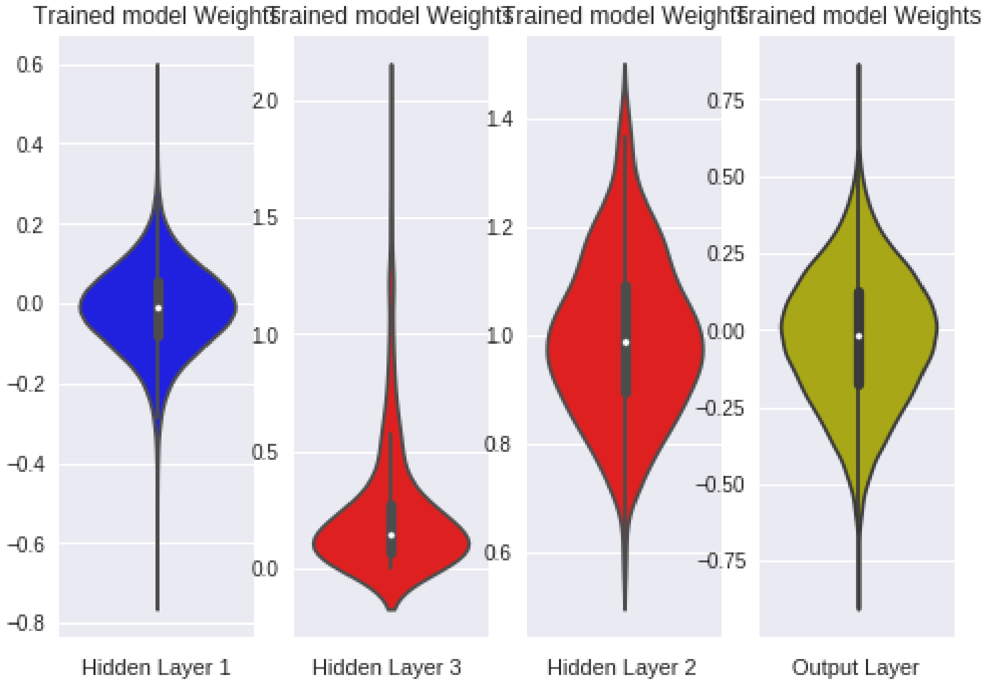
```
Test score: 0.08036984513603966
Test accuracy: 0.9799
```



In [0]:
```
w_after = model_drop.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=h3_w, color='r')
```

```
plt.xlabel('Hidden Layer 3 ')

plt.subplot(1, 4, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

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

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: Futu
reWarning: remove_na is deprecated and is a private function. Do not us
e.
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: Futu
reWarning: remove_na is deprecated and is a private function. Do not us
e.
  violin_data = remove_na(group_data)
```



Model-3-[5 hidden layer]

In [0]:
```
model_relu = Sequential()
model_relu.add(Dense(512, activation='relu', input_shape=(input_dim,), 
kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_relu.add(Dense(340, activation='relu', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.196, seed=None)) )
model_relu.add(Dense(180, activation='relu', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.105, seed=None)) )
model_relu.add(Dense(90, activation='relu', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.149, seed=None)) )
model_relu.add(Dense(30, activation='relu', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.25, seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())
```

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

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_22 (Dense)             (None, 512)               401920
_________________________________________________________________
dense_23 (Dense)             (None, 340)               174420
_________________________________________________________________
dense_24 (Dense)             (None, 180)               61380
_________________________________________________________________
dense_25 (Dense)             (None, 90)                16290
_________________________________________________________________
dense_26 (Dense)             (None, 30)                2730
_________________________________________________________________
dense_27 (Dense)             (None, 10)                310
=================================================================
Total params: 657,050
Trainable params: 657,050
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 11s 182us/step - loss:
0.3214 - acc: 0.9074 - val_loss: 0.1507 - val_acc: 0.9536
Epoch 2/20
60000/60000 [==============================] - 10s 161us/step - loss:
0.1097 - acc: 0.9669 - val_loss: 0.1314 - val_acc: 0.9603
Epoch 3/20
60000/60000 [==============================] - 10s 161us/step - loss:
0.0761 - acc: 0.9763 - val_loss: 0.1099 - val_acc: 0.9683
Epoch 4/20
60000/60000 [==============================] - 10s 163us/step - loss:
0.0536 - acc: 0.9828 - val_loss: 0.0967 - val_acc: 0.9743
```

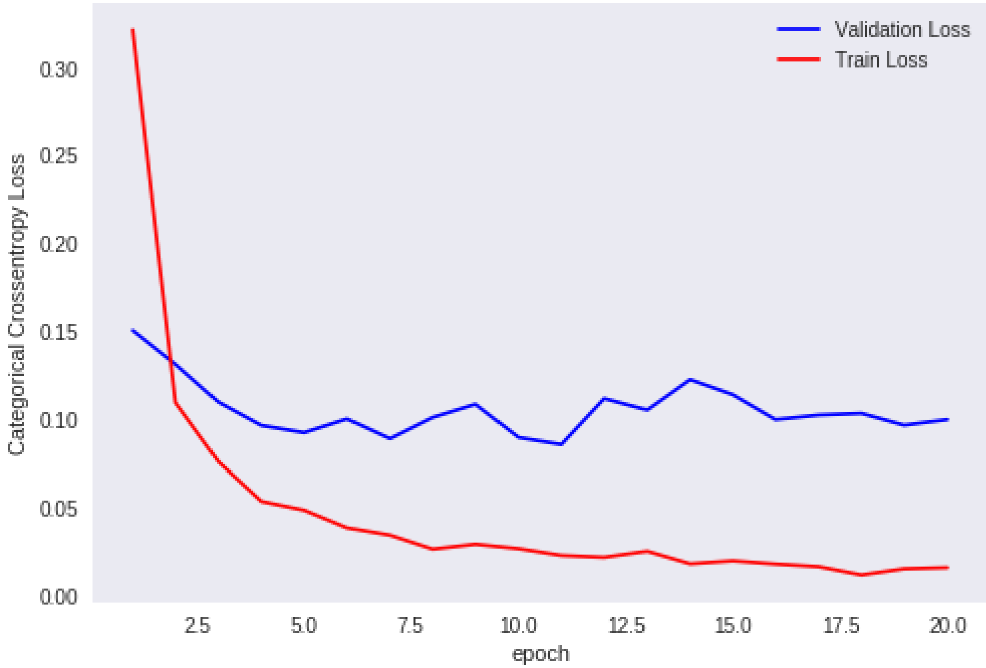
```
0.0336 - acc: 0.9828 - val_loss: 0.0967 - val_acc: 0.9743
Epoch 5/20
60000/60000 [==============================] - 10s 162us/step - loss:
0.0486 - acc: 0.9842 - val_loss: 0.0926 - val_acc: 0.9741
Epoch 6/20
60000/60000 [==============================] - 10s 162us/step - loss:
0.0386 - acc: 0.9872 - val_loss: 0.1003 - val_acc: 0.9739
Epoch 7/20
60000/60000 [==============================] - 10s 162us/step - loss:
0.0344 - acc: 0.9889 - val_loss: 0.0892 - val_acc: 0.9765
Epoch 8/20
60000/60000 [==============================] - 10s 163us/step - loss:
0.0265 - acc: 0.9913 - val_loss: 0.1012 - val_acc: 0.9760
Epoch 9/20
60000/60000 [==============================] - 10s 162us/step - loss:
0.0293 - acc: 0.9910 - val_loss: 0.1087 - val_acc: 0.9743
Epoch 10/20
60000/60000 [==============================] - 10s 162us/step - loss:
0.0267 - acc: 0.9916 - val_loss: 0.0898 - val_acc: 0.9783
Epoch 11/20
60000/60000 [==============================] - 10s 161us/step - loss:
0.0229 - acc: 0.9931 - val_loss: 0.0859 - val_acc: 0.9799
Epoch 12/20
60000/60000 [==============================] - 10s 161us/step - loss:
0.0218 - acc: 0.9928 - val_loss: 0.1119 - val_acc: 0.9740
Epoch 13/20
60000/60000 [==============================] - 10s 160us/step - loss:
0.0252 - acc: 0.9925 - val_loss: 0.1054 - val_acc: 0.9766
Epoch 14/20
60000/60000 [==============================] - 10s 163us/step - loss:
0.0181 - acc: 0.9945 - val_loss: 0.1227 - val_acc: 0.9731
Epoch 15/20
60000/60000 [==============================] - 10s 160us/step - loss:
0.0198 - acc: 0.9938 - val_loss: 0.1142 - val_acc: 0.9740
Epoch 16/20
60000/60000 [==============================] - 10s 161us/step - loss:
0.0180 - acc: 0.9944 - val_loss: 0.1000 - val_acc: 0.9782
Epoch 17/20
60000/60000 [==============================] - 10s 161us/step - loss:

0.0165 - acc: 0.9951 - val_loss: 0.1026 - val_acc: 0.9781
```

```
0.0165 - acc: 0.9951 - val_loss: 0.1026 - val_acc: 0.9781
Epoch 18/20
60000/60000 [==============================] - 10s 161us/step - loss:
0.0118 - acc: 0.9962 - val_loss: 0.1035 - val_acc: 0.9792
Epoch 19/20
60000/60000 [==============================] - 10s 161us/step - loss:
0.0153 - acc: 0.9954 - val_loss: 0.0968 - val_acc: 0.9783
Epoch 20/20
60000/60000 [==============================] - 10s 162us/step - loss:
0.0159 - acc: 0.9951 - val_loss: 0.1000 - val_acc: 0.9797
```

In [0]:

```
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,nb_epoch+1))

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epo
chs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

# we will get val_loss and val_acc only when you pass the paramter vali
dation_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 epochs


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

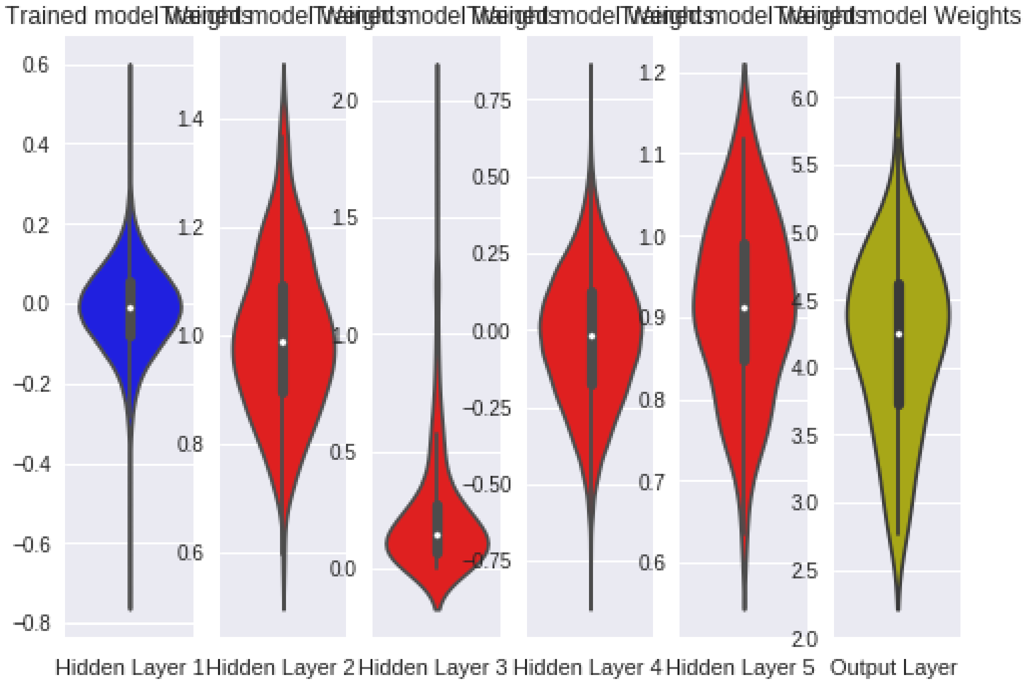
```
Test score: 0.09998379269517791
Test accuracy: 0.9797
```



In [0]:
```
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='r')
plt.xlabel('Hidden Layer 3 ')

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

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='r')
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()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: Futu
reWarning: remove_na is deprecated and is a private function. Do not us
e.
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: Futu
reWarning: remove_na is deprecated and is a private function. Do not us
e.
  violin_data = remove_na(group_data)
```




```
## MLP + Batch-Norm on hidden Layers + AdamOptimizer
```

MLP + Batch-Norm on hidden Layers + AdamOptimizer

```
from keras.layers.normalization import BatchNormalization

model_batch = Sequential()

model_batch.add(Dense(512, activation='relu', input_shape=(input_dim,),
 kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_batch.add(BatchNormalization())

model_batch.add(Dense(340, activation='relu', kernel_initializer=Random
Normal(mean=0.0, stddev=0.196, seed=None)) )
```

```
model_batch.add(BatchNormalization())

model_batch.add(Dense(180, activation='relu', kernel_initializer=Random
Normal(mean=0.0, stddev=0.105, seed=None)) )
model_batch.add(BatchNormalization())

model_batch.add(Dense(90, activation='relu', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.149, seed=None)) )
model_batch.add(BatchNormalization())

model_batch.add(Dense(30, activation='relu', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.25, seed=None)) )
model_batch.add(BatchNormalization())

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


model_batch.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_28 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_11 (Batc (None, 512)               2048
_________________________________________________________________
dense_29 (Dense)             (None, 340)               174420
_________________________________________________________________
batch_normalization_12 (Batc (None, 340)               1360
_________________________________________________________________
dense_30 (Dense)             (None, 180)               61380
_________________________________________________________________
batch_normalization_13 (Batc (None, 180)               720
_________________________________________________________________
dense_31 (Dense)             (None, 90)                16290
_________________________________________________________________
batch_normalization_14 (Batc (None, 90)                360
_________________________________________________________________
dense_32 (Dense)             (None, 30)                2730
_________________________________________________________________
```

```
batch_normalization_15 (Batc (None, 30)                120
_________________________________________________________________
dense_33 (Dense)             (None, 10)                310
=================================================================
Total params: 661,658
Trainable params: 659,354
Non-trainable params: 2,304
_________________________________________________________________
```

In [0]:
```
model_batch.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_batch.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 15s 250us/step - loss: 0.2588 - acc: 0.9311 - val_loss: 0.1188 - val_acc: 0.9641
Epoch 2/20
60000/60000 [==============================] - 12s 196us/step - loss: 0.0883 - acc: 0.9740 - val_loss: 0.0900 - val_acc: 0.9735
Epoch 3/20
60000/60000 [==============================] - 12s 196us/step - loss: 0.0620 - acc: 0.9811 - val_loss: 0.0772 - val_acc: 0.9762
Epoch 4/20
60000/60000 [==============================] - 12s 198us/step - loss: 0.0455 - acc: 0.9858 - val_loss: 0.0910 - val_acc: 0.9736
Epoch 5/20
60000/60000 [==============================] - 12s 196us/step - loss: 0.0359 - acc: 0.9888 - val_loss: 0.0869 - val_acc: 0.9751
Epoch 6/20
60000/60000 [==============================] - 12s 197us/step - loss: 0.0344 - acc: 0.9887 - val_loss: 0.0882 - val_acc: 0.9738
Epoch 7/20
60000/60000 [==============================] - 12s 198us/step - loss: 0.0254 - acc: 0.9920 - val_loss: 0.0749 - val_acc: 0.9770
Epoch 8/20
60000/60000 [==============================] - 12s 199us/step - loss:
```

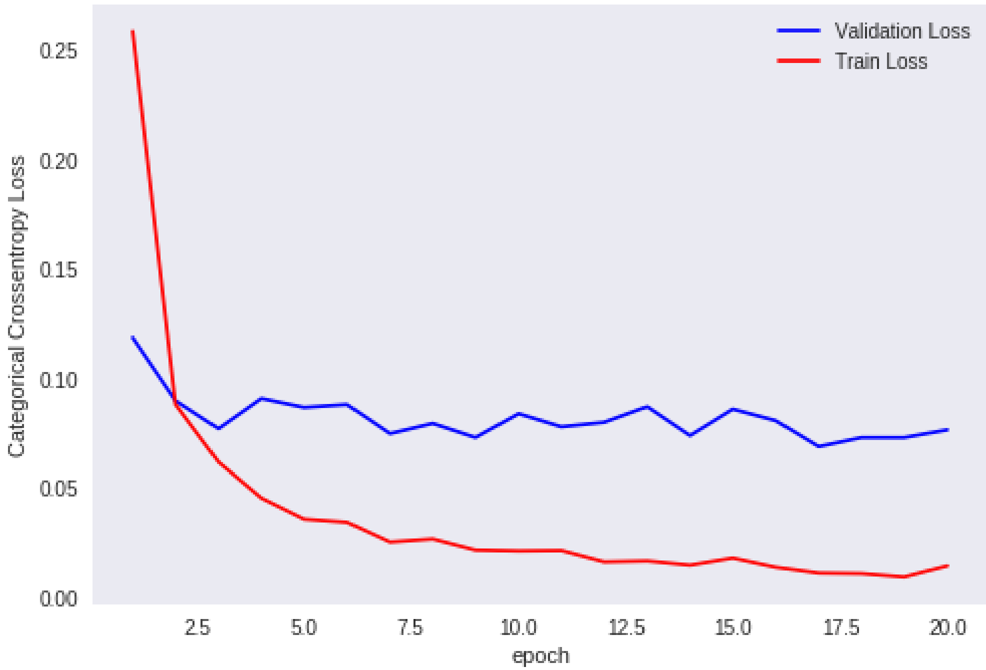
```
0.0268 - acc: 0.9910 - val_loss: 0.0796 - val_acc: 0.9778
Epoch 9/20
60000/60000 [==============================] - 12s 199us/step - loss:
0.0217 - acc: 0.9929 - val_loss: 0.0732 - val_acc: 0.9785
Epoch 10/20
60000/60000 [==============================] - 12s 199us/step - loss:
0.0213 - acc: 0.9930 - val_loss: 0.0840 - val_acc: 0.9776
Epoch 11/20
60000/60000 [==============================] - 12s 199us/step - loss:
0.0215 - acc: 0.9929 - val_loss: 0.0781 - val_acc: 0.9767
Epoch 12/20
60000/60000 [==============================] - 12s 199us/step - loss:
0.0163 - acc: 0.9947 - val_loss: 0.0801 - val_acc: 0.9794
Epoch 13/20
60000/60000 [==============================] - 12s 197us/step - loss:
0.0167 - acc: 0.9946 - val_loss: 0.0872 - val_acc: 0.9794
Epoch 14/20
60000/60000 [==============================] - 12s 199us/step - loss:
0.0148 - acc: 0.9951 - val_loss: 0.0740 - val_acc: 0.9811
Epoch 15/20
60000/60000 [==============================] - 12s 199us/step - loss:
0.0180 - acc: 0.9936 - val_loss: 0.0861 - val_acc: 0.9791
Epoch 16/20
60000/60000 [==============================] - 12s 198us/step - loss:
0.0139 - acc: 0.9953 - val_loss: 0.0809 - val_acc: 0.9808
Epoch 17/20
60000/60000 [==============================] - 12s 199us/step - loss:
0.0112 - acc: 0.9963 - val_loss: 0.0690 - val_acc: 0.9822
Epoch 18/20
60000/60000 [==============================] - 12s 198us/step - loss:
0.0110 - acc: 0.9964 - val_loss: 0.0731 - val_acc: 0.9832
Epoch 19/20
60000/60000 [==============================] - 12s 197us/step - loss:
0.0094 - acc: 0.9966 - val_loss: 0.0731 - val_acc: 0.9820
Epoch 20/20
60000/60000 [==============================] - 12s 201us/step - loss:
0.0145 - acc: 0.9953 - val_loss: 0.0767 - val_acc: 0.9815
```

In [0]:

```
score = model_batch.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))

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epo
chs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

# we will get val_loss and val_acc only when you pass the paramter vali
dation_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 epochs

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

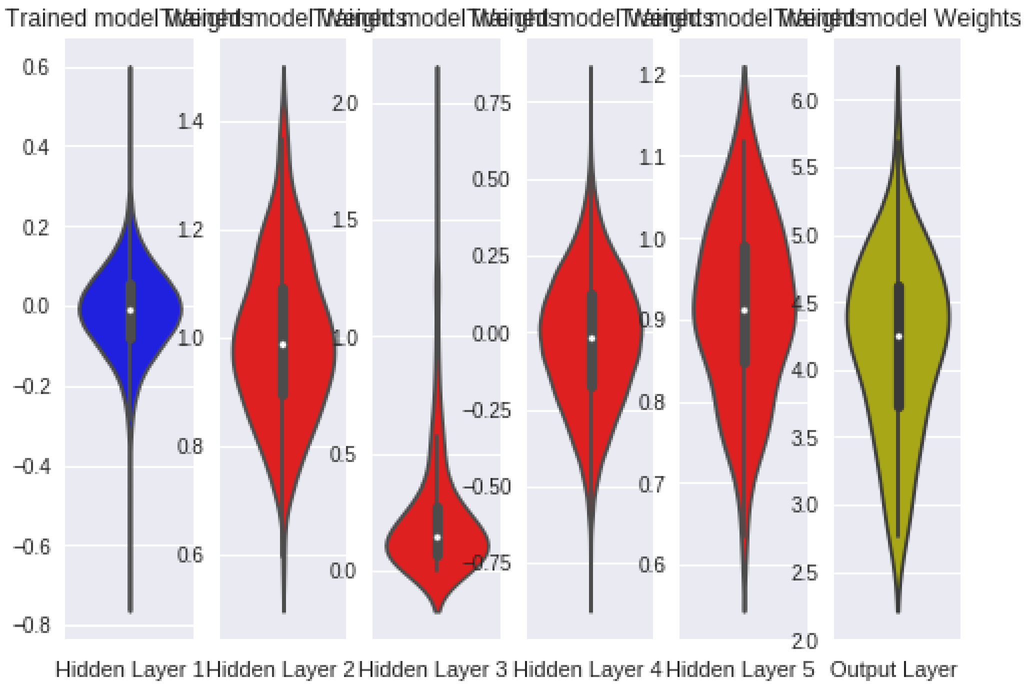
```
Test score: 0.0766602623711835
Test accuracy: 0.9815
```



In [0]:
```
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='r')
plt.xlabel('Hidden Layer 3 ')

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

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='r')
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()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: Futu
reWarning: remove_na is deprecated and is a private function. Do not us
e.
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: Futu
reWarning: remove_na is deprecated and is a private function. Do not us
e.
  violin_data = remove_na(group_data)
```



In [0]:
```
# MLP + Dropout + AdamOptimizer
```

MLP + Dropout + AdamOptimizer

In [0]:
```
# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batc
hnormalization-function-in-keras

from keras.layers import Dropout

#model_drop = Sequential()
model_drop = Sequential()

model_drop.add(Dense(512, activation='relu', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_drop.add(BatchNormalization())
```

```
model_drop.add(Dropout(0.5))

model_drop.add(Dense(340, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.196, seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(180, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.105, seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))


model_drop.add(Dense(90, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.149, seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(30, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.25, seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

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


model_drop.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_34 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_16 (Batc (None, 512)               2048
_________________________________________________________________
dropout_6 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_35 (Dense)             (None, 340)               174420
_________________________________________________________________
batch_normalization_17 (Batc (None, 340)               1360
```

```
dropout_7 (Dropout)          (None, 340)               0
_________________________________________________________________
dense_36 (Dense)             (None, 180)               61380
_________________________________________________________________
batch_normalization_18 (Batc (None, 180)               720
_________________________________________________________________
dropout_8 (Dropout)          (None, 180)               0
_________________________________________________________________
dense_37 (Dense)             (None, 90)                16290
_________________________________________________________________
batch_normalization_19 (Batc (None, 90)                360
_________________________________________________________________
dropout_9 (Dropout)          (None, 90)                0
_________________________________________________________________
dense_38 (Dense)             (None, 30)                2730
_________________________________________________________________
batch_normalization_20 (Batc (None, 30)                120
_________________________________________________________________
dropout_10 (Dropout)         (None, 30)                0
_________________________________________________________________
dense_39 (Dense)             (None, 10)                310
=================================================================
Total params: 661,658
Trainable params: 659,354
Non-trainable params: 2,304
_________________________________________________________________
```

In [0]:

```
model_drop.compile(optimizer='adam', loss='categorical_crossentropy', m
etrics=['accuracy'])

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 16s 271us/step - loss:
1.5646 - acc: 0.4964 - val_loss: 0.3700 - val_acc: 0.9005
Epoch 2/20
```

```
60000/60000 [==============================] - 13s 221us/step - loss:
0.6445 - acc: 0.8138 - val_loss: 0.2128 - val_acc: 0.9383
Epoch 3/20
60000/60000 [==============================] - 13s 222us/step - loss:
0.4319 - acc: 0.8859 - val_loss: 0.1704 - val_acc: 0.9535
Epoch 4/20
60000/60000 [==============================] - 13s 221us/step - loss:
0.3444 - acc: 0.9134 - val_loss: 0.1503 - val_acc: 0.9593
Epoch 5/20
60000/60000 [==============================] - 14s 225us/step - loss:
0.3034 - acc: 0.9257 - val_loss: 0.1386 - val_acc: 0.9644
Epoch 6/20
60000/60000 [==============================] - 14s 228us/step - loss:
0.2726 - acc: 0.9341 - val_loss: 0.1242 - val_acc: 0.9695
Epoch 7/20
60000/60000 [==============================] - 13s 221us/step - loss:
0.2446 - acc: 0.9419 - val_loss: 0.1180 - val_acc: 0.9714
Epoch 8/20
60000/60000 [==============================] - 13s 221us/step - loss:
0.2305 - acc: 0.9454 - val_loss: 0.1138 - val_acc: 0.9703
Epoch 9/20
60000/60000 [==============================] - 13s 221us/step - loss:
0.2122 - acc: 0.9500 - val_loss: 0.1072 - val_acc: 0.9737
Epoch 10/20
60000/60000 [==============================] - 13s 222us/step - loss:
0.2020 - acc: 0.9527 - val_loss: 0.1049 - val_acc: 0.9745
Epoch 11/20
60000/60000 [==============================] - 13s 221us/step - loss:
0.1870 - acc: 0.9559 - val_loss: 0.1048 - val_acc: 0.9751
Epoch 12/20
60000/60000 [==============================] - 13s 221us/step - loss:
0.1769 - acc: 0.9583 - val_loss: 0.0972 - val_acc: 0.9779
Epoch 13/20
60000/60000 [==============================] - 13s 221us/step - loss:
0.1714 - acc: 0.9594 - val_loss: 0.0997 - val_acc: 0.9761
Epoch 14/20
60000/60000 [==============================] - 13s 220us/step - loss:
0.1638 - acc: 0.9619 - val_loss: 0.0948 - val_acc: 0.9777
Epoch 15/20
```

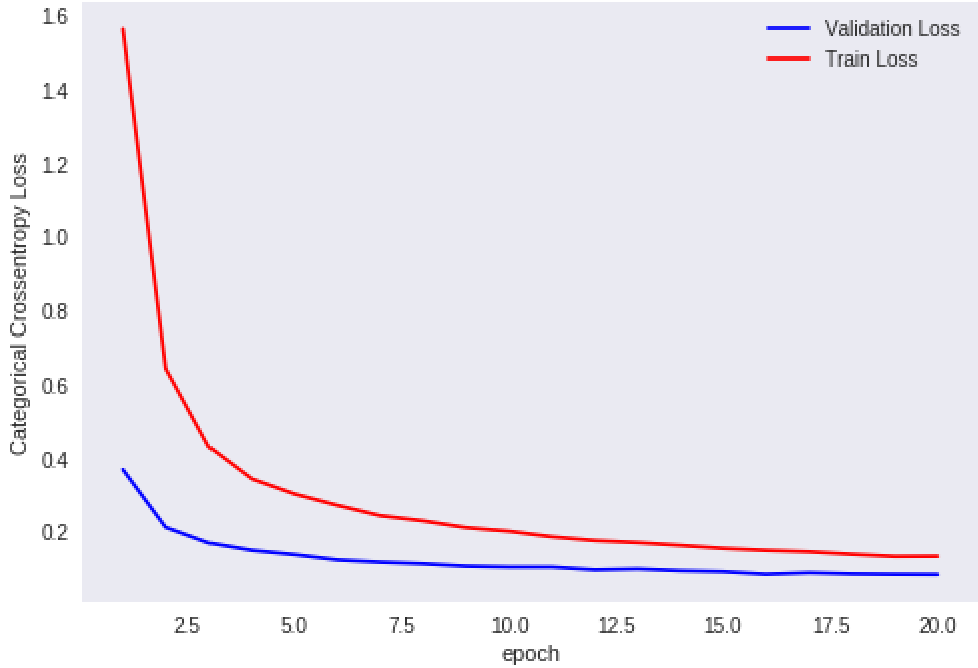
```
60000/60000 [==============================] - 13s 220us/step - loss:
0.1557 - acc: 0.9643 - val_loss: 0.0923 - val_acc: 0.9778
Epoch 16/20
60000/60000 [==============================] - 13s 219us/step - loss:
0.1503 - acc: 0.9655 - val_loss: 0.0852 - val_acc: 0.9805
Epoch 17/20
60000/60000 [==============================] - 13s 221us/step - loss:
0.1464 - acc: 0.9656 - val_loss: 0.0892 - val_acc: 0.9799
Epoch 18/20
60000/60000 [==============================] - 13s 219us/step - loss:
0.1396 - acc: 0.9677 - val_loss: 0.0862 - val_acc: 0.9809
Epoch 19/20
60000/60000 [==============================] - 13s 221us/step - loss:
0.1337 - acc: 0.9696 - val_loss: 0.0851 - val_acc: 0.9797
Epoch 20/20
60000/60000 [==============================] - 13s 221us/step - loss:
0.1342 - acc: 0.9688 - val_loss: 0.0846 - val_acc: 0.9807
```

In [0]:

```
score = model_drop.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))

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epo
chs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

# we will get val_loss and val_acc only when you pass the paramter vali
dation_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 epochs

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

```
Test score: 0.08459711499696132
Test accuracy: 0.9807
```



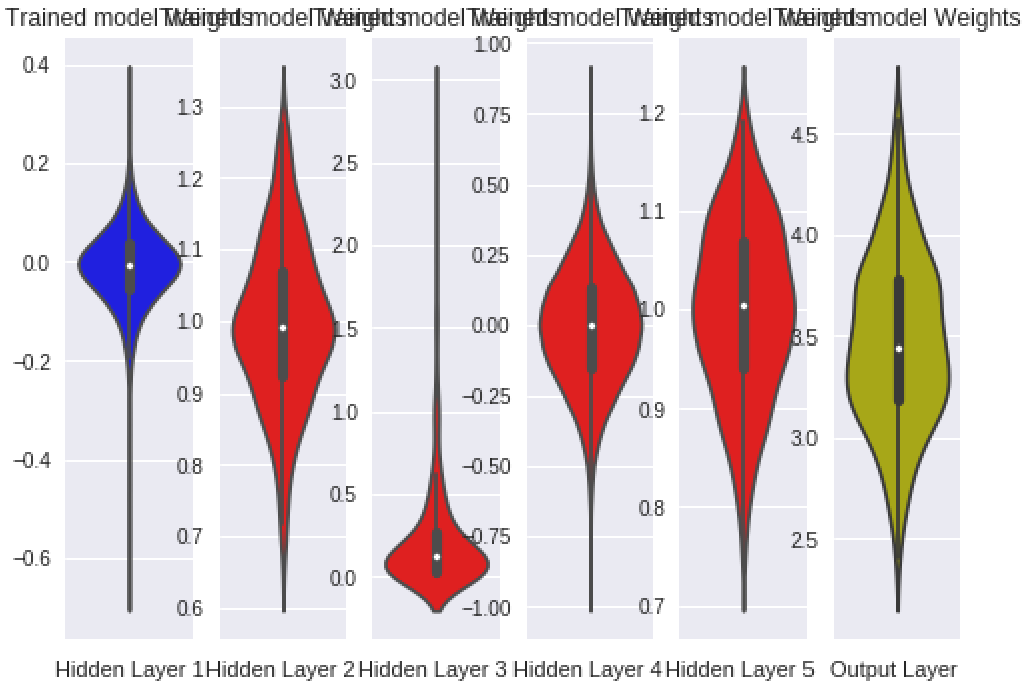
In [0]:

```
w_after = model_drop.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='r')
plt.xlabel('Hidden Layer 3 ')


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

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='r')
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()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: Futu
reWarning: remove_na is deprecated and is a private function. Do not us
e.
```

```
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: Futu
reWarning: remove_na is deprecated and is a private function. Do not us
e.
  violin_data = remove_na(group_data)
```



Conclusion-

Model1.-Hidden layers-2

```
                     * MLP+Relu+Adam-
         -loss -0.0982,Test Accuracy-98.06%
                     * MLP + Batch-Norm on hidden Layers + AdamOpti
   mizer -loss -0.0933,Test Accuracy-97.89%
                      *MLP + Dropout + AdamOptimizer
```

```
        -loss -0.0530,Test Accuracy-98.52%
```

Model2.-Hidden layers-3

```
                   * MLP+Relu+Adam
    -loss-0.1107,Test Accuracy-97.85%
                 * MLP + Batch-Norm on hidden Layers + AdamOptimize
  r-loss-0.0874,Test Accuracy-97.99%
                  *MLP + Dropout + AdamOptimizer
    -loss-0.0804,Test Accuracy-97.99%
```

Model3.-Hidden layers-5

```
                   * MLP+Relu+Adam
    -loss=0.1000,Test Accuracy-97.97%
                 * MLP + Batch-Norm on hidden Layers + AdamOptimize
  r-loss=0.0767,Test Accuracy-98.15%
                  *MLP + Dropout + AdamOptimizer
    -loss=0.0846,Test Accuracy-98.07%
```

1.Used Relu-activation function- It speeds up the convergence and . -It avoids vanishing gradient problem.

2.We are adding dropout to avoid the modeloverfitting.

3.Batch Normalization- To avoid internal Co-Varience Shift