MLP on MNIST dataset using Keras

In [1]:
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
from keras.utils import np_utils
from keras.datasets import mnist
import seaborn as sns
import keras
```

```
Using TensorFlow backend.
```

In [2]:
```
%matplotlib notebook
import matplotlib.pyplot as plt
import numpy as np
import time
# https://gist.github.com/greydanus/f6eee59eaf1d90fcb3b534a25362cea4
# https://stackoverflow.com/a/14434334
# this function is used to update the plots for each epoch and error
def plt_dynamic(x, vy, ty, ax, colors=['b']):
    ax.plot(x, vy, 'b', label="Validation Loss")
    ax.plot(x, ty, 'r', label="Train Loss")
    plt.legend()
    plt.grid()
    fig.canvas.draw()
```

In [3]:
```
(x_train,y_train),(x_test,y_test)= mnist.load_data()
```

In [4]:
```
x_train.shape
```

Out[4]:
```
(60000, 28, 28)
```

In [5]:
```
print('Number of trianing examples:',x_train.shape[0],'and Images of di
mension:',x_train.shape[1:])
```

```
print('Number of test examples:',x_test.shape[0],'and Images of dimensi
on:',x_test.shape[1:])
```

```
Number of trianing examples: 60000 and Images of dimension: (28, 28)
Number of test examples: 10000 and Images of dimension: (28, 28)
```

Converting data to desired form

In [6]:
```
#images should be flattened to 1-dim, from 28X28 to 784
x_train= x_train.reshape(x_train.shape[0],x_train.shape[1]*x_train.shap
e[2])
x_test= x_test.reshape(x_test.shape[0],x_test.shape[1]*x_test.shape[2])
print('Number of trianing examples:',x_train.shape[0],'and Images of di
mension:',x_train.shape[1:])
print('Number of test examples:',x_test.shape[0],'and Images of dimensi
on:',x_test.shape[1:])
```

```
Number of trianing examples: 60000 and Images of dimension: (784,)
Number of test examples: 10000 and Images of dimension: (784,)
```

In [7]:
```
x_train[1]
```

Out[7]:
```
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,  51, 159, 253,
       159,  50,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  48, 238,
       252, 252, 252, 237,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  54,
       227, 253, 252, 239, 233, 252,  57,   6,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  10,
```

```
  60, 224, 252, 253, 252, 202,  84, 252, 253, 122,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0, 163, 252, 252, 252, 253, 252, 252,  96, 189, 253, 167,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,  51, 238, 253, 253, 190, 114, 253, 228,  47,  79, 255,
 168,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,  48, 238, 252, 252, 179,  12,  75, 121,  21,   0,
   0, 253, 243,  50,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,  38, 165, 253, 233, 208,  84,   0,   0,   0,
   0,   0,   0, 253, 252, 165,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   7, 178, 252, 240,  71,  19,  28,   0,
   0,   0,   0,   0,   0, 253, 252, 195,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,  57, 252, 252,  63,   0,   0,
   0,   0,   0,   0,   0,   0,   0, 253, 252, 195,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0, 198, 253, 190,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0, 255, 253, 196,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  76, 246, 252,
 112,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 253, 252,
 148,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  85,
 252, 230,  25,   0,   0,   0,   0,   0,   0,   0,   0,   7, 135,
 253, 186,  12,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,  85, 252, 223,   0,   0,   0,   0,   0,   0,   0,   0,   7,
 131, 252, 225,  71,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,  85, 252, 145,   0,   0,   0,   0,   0,   0,   0,
  48, 165, 252, 173,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,  86, 253, 225,   0,   0,   0,   0,   0,
   0, 114, 238, 253, 162,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,  85, 252, 249, 146,  48,  29,
  85, 178, 225, 253, 223, 167,  56,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,  85, 252, 252, 252,
 229, 215, 252, 252, 252, 196, 130,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  28, 199,
 252, 252, 253, 252, 252, 233, 145,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,  25, 128, 252, 253, 252, 141,  37,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
```

```
         0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
         0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
         0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
         0,    0,    0,    0,    0,    0,    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)
```

In [8]:

```python
#normalizing the data
x_train=x_train/255
x_test=x_test/255
x_train[1]
```

Out[8]:

```
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.        , 0.2       , 0.62352941, 0.99215686,
       0.62352941, 0.19607843, 0.        , 0.        , 0.        ,
```

```
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.18823529,
0.93333333, 0.98823529, 0.98823529, 0.98823529, 0.92941176,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.21176471, 0.89019608, 0.99215686, 0.98823529,
0.9372549 , 0.91372549, 0.98823529, 0.22352941, 0.02352941,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.03921569, 0.23529412, 0.87843137,
0.98823529, 0.99215686, 0.98823529, 0.79215686, 0.32941176,
0.98823529, 0.99215686, 0.47843137, 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.63921569, 0.98823529, 0.98823529, 0.98823529, 0.99215686,
0.98823529, 0.98823529, 0.37647059, 0.74117647, 0.99215686,
0.65490196, 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.2       , 0.93333333, 0.99215686,
0.99215686, 0.74509804, 0.44705882, 0.99215686, 0.89411765,
0.18431373, 0.30980392, 1.        , 0.65882353, 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.18823529,
0.93333333, 0.98823529, 0.98823529, 0.70196078, 0.04705882,
0.29411765, 0.4745098 , 0.08235294, 0.        , 0.        ,
0.99215686, 0.95294118, 0.19607843, 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.14901961, 0.64705882, 0.99215686, 0.91372549,
0.81568627, 0.32941176, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.99215686, 0.98823529,
```

```
0.64705882, 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.02745098, 0.69803922,
0.98823529, 0.94117647, 0.27843137, 0.0745098 , 0.10980392,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.99215686, 0.98823529, 0.76470588, 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.22352941, 0.98823529, 0.98823529, 0.24705882,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.99215686,
0.98823529, 0.76470588, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.77647059,
0.99215686, 0.74509804, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 1.        , 0.99215686, 0.76862745,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.29803922, 0.96470588, 0.98823529, 0.43921569,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.99215686, 0.98823529, 0.58039216, 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.33333333,
0.98823529, 0.90196078, 0.09803922, 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.02745098, 0.52941176, 0.99215686, 0.72941176,
0.04705882, 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.33333333, 0.98823529, 0.8745098 ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.02745098, 0.51372549,
0.98823529, 0.88235294, 0.27843137, 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.33333333, 0.98823529, 0.56862745, 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.18823529, 0.64705882, 0.98823529, 0.67843137, 0.        ,
```

```
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.3372549 , 0.99215686,
0.88235294, 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.44705882, 0.93333333, 0.99215686,
0.63529412, 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.33333333, 0.98823529, 0.97647059, 0.57254902,
0.18823529, 0.11372549, 0.33333333, 0.69803922, 0.88235294,
0.99215686, 0.8745098 , 0.65490196, 0.21960784, 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.33333333,
0.98823529, 0.98823529, 0.98823529, 0.89803922, 0.84313725,
0.98823529, 0.98823529, 0.98823529, 0.76862745, 0.50980392,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.10980392, 0.78039216, 0.98823529,
0.98823529, 0.99215686, 0.98823529, 0.98823529, 0.91372549,
0.56862745, 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.09803922, 0.50196078, 0.98823529, 0.99215686,
0.98823529, 0.55294118, 0.14509804, 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
```

```
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       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 [9]:
```
#output is number with 10 types 0-9
print('image classified as ',y_train[1])
#we need to one-Hot encode the output as it is needed for MLPs
y_train= np_utils.to_categorical(y_train,10)
y_test= np_utils.to_categorical(y_test,10)
print('after conversion image classified as ',y_train[1])
```

```
image classified as  0
after conversion image classified as  [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

MODEL1- with 2-Hidden layers input(784)--hidden(400)--hidden(100)--Output(10)

In [10]:
```
from keras.models import Sequential
from keras.layers import Activation,Dense,Dropout
from keras import initializers
```

Simple 2 hidden layer design model

In [11]:
```
#model declaration and initialization
model_1A= Sequential()
```

```
model_1A.add(Dense(400, activation='relu', \
        input_shape= (784,), kernel_initializer=keras.initializers.he_n
ormal(seed=None)))
model_1A.add(Dense(100, activation='relu', \
        kernel_initializer= keras.initializers.he_normal(seed=None)))
model_1A.add(Dense(10, activation='softmax'))

print(model_1A.summary())
```

```
WARNING:tensorflow:From C:\Anaconda3\lib\site-packages\tensorflow\pytho
n\framework\op_def_library.py:263: colocate_with (from tensorflow.pytho
n.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 400)               314000    
_________________________________________________________________
dense_2 (Dense)              (None, 100)               40100     
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1010      
=================================================================
Total params: 355,110
Trainable params: 355,110
Non-trainable params: 0
_________________________________________________________________
None
```

In [12]:

```
#run
model_1A.compile(optimizer='adam', loss='categorical_crossentropy', met
rics=['accuracy'])
history_1A = model_1A.fit(x_train,y_train,batch_size=200,epochs=20,verb
ose=1,validation_data=(x_test,y_test))
```

```
WARNING:tensorflow:From C:\Anaconda3\lib\site-packages\tensorflow\pytho
n\ops\math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops)
is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 109us/step - loss: 0.
2968 - acc: 0.9153 - val_loss: 0.1412 - val_acc: 0.9565 loss: 0.
Epoch 2/20
60000/60000 [==============================] - 6s 99us/step - loss: 0.1
062 - acc: 0.9685 - val_loss: 0.1024 - val_acc: 0.9688
Epoch 3/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0
690 - acc: 0.9792 - val_loss: 0.0801 - val_acc: 0.9758
Epoch 4/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0
503 - acc: 0.9849 - val_loss: 0.0682 - val_acc: 0.9778
Epoch 5/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.0
346 - acc: 0.9899 - val_loss: 0.0710 - val_acc: 0.9784
Epoch 6/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.0
261 - acc: 0.9924 - val_loss: 0.0762 - val_acc: 0.9775
Epoch 7/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0
204 - acc: 0.9939 - val_loss: 0.0718 - val_acc: 0.9798
Epoch 8/20
60000/60000 [==============================] - 5s 92us/step - loss: 0.0
163 - acc: 0.9949 - val_loss: 0.0745 - val_acc: 0.9780
Epoch 9/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0
121 - acc: 0.9967 - val_loss: 0.0711 - val_acc: 0.9815
Epoch 10/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0
087 - acc: 0.9977 - val_loss: 0.0716 - val_acc: 0.9805
Epoch 11/20
60000/60000 [==============================] - 5s 92us/step - loss: 0.0
076 - acc: 0.9979 - val_loss: 0.0698 - val_acc: 0.9822
Epoch 12/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0
118 - acc: 0.9962 - val_loss: 0.0782 - val_acc: 0.9801
Epoch 13/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0
```

```
122 - acc: 0.9959 - val_loss: 0.0803 - val_acc: 0.9799
Epoch 14/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.0
054 - acc: 0.9985 - val_loss: 0.0806 - val_acc: 0.9810
Epoch 15/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0
029 - acc: 0.9993 - val_loss: 0.0813 - val_acc: 0.9811
Epoch 16/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.0
051 - acc: 0.9985 - val_loss: 0.0993 - val_acc: 0.9781
Epoch 17/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0
115 - acc: 0.9961 - val_loss: 0.0922 - val_acc: 0.9790
Epoch 18/20
60000/60000 [==============================] - 5s 92us/step - loss: 0.0
085 - acc: 0.9969 - val_loss: 0.1020 - val_acc: 0.9773
Epoch 19/20
60000/60000 [==============================] - 5s 92us/step - loss: 0.0
079 - acc: 0.9976 - val_loss: 0.0901 - val_acc: 0.9816
Epoch 20/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0
059 - acc: 0.9980 - val_loss: 0.0968 - val_acc: 0.9784
```

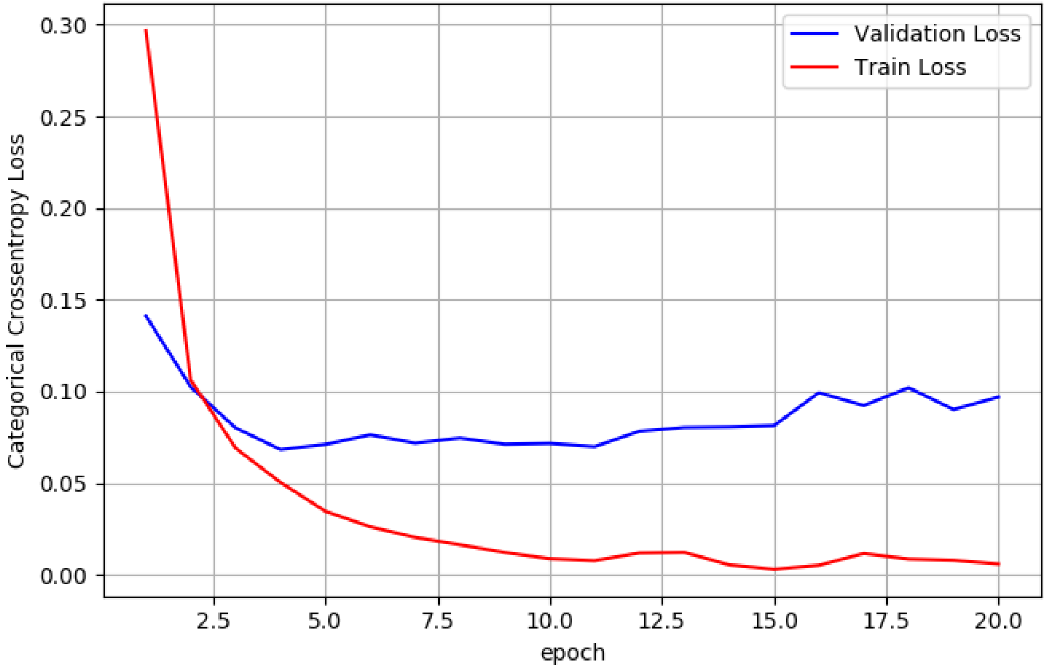
In [13]:

```
score= model_1A.evaluate(x_test, y_test, verbose=0)
print('Test score: ',score[0])
print('Test accuracy: ',score[1])
```

```
Test score:  0.09683440083766617
Test accuracy:  0.9784
```

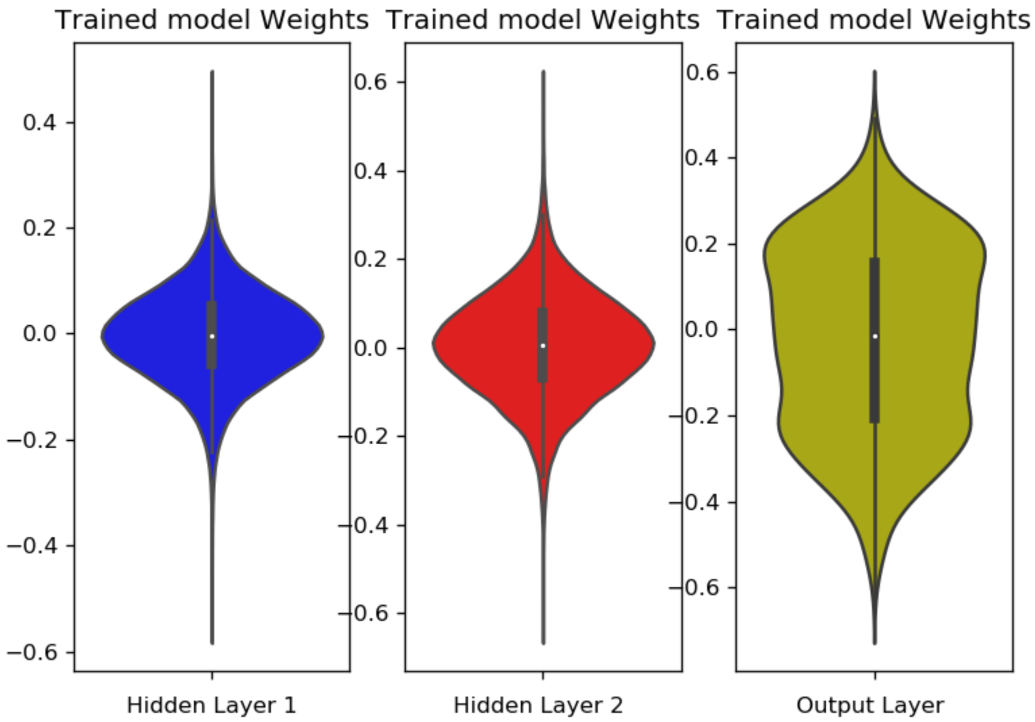
In [52]:

```
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')
x = list(range(1,21))
vy = history_1A.history['val_loss']
ty = history_1A.history['loss']
plt_dynamic(x, vy, ty, ax)
```



In [17]:
```
w_after = model_1A.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()
```



2 hidden layer with batch normalization

In [18]:
```
#model declaration and initialization
from keras.layers.normalization import BatchNormalization
model_1B= Sequential()
model_1B.add(Dense(400, activation='relu', \
        input_shape= (784,), kernel_initializer=keras.initializers.he_n
ormal(seed=None)))
model_1B.add(BatchNormalization())

model_1B.add(Dense(100, activation='relu', \
        kernel_initializer= keras.initializers.he_normal(seed=None)))
model_1B.add(BatchNormalization())

model_1B.add(Dense(10, activation='softmax', \
                   kernel_initializer=keras.initializers.he_normal(seed
=None)))

print(model_1B.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 400)               314000
_________________________________________________________________
batch_normalization_1 (Batch (None, 400)               1600
_________________________________________________________________
dense_5 (Dense)              (None, 100)               40100
_________________________________________________________________
batch_normalization_2 (Batch (None, 100)               400
_________________________________________________________________
dense_6 (Dense)              (None, 10)                1010
=================================================================
Total params: 357,110
Trainable params: 356,110
Non-trainable params: 1,000
_________________________________________________________________
None
```

In [19]:
```
#run
model_1B.compile(optimizer='adam', loss='categorical_crossentropy', met
```

```
rics=['accuracy'])
history_1B = model_1B.fit(x_train,y_train,batch_size=200,epochs=20,verb
ose=1,validation_data=(x_test,y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 125us/step - loss: 0.
2218 - acc: 0.9355 - val_loss: 0.1130 - val_acc: 0.9648
Epoch 2/20
60000/60000 [==============================] - 7s 114us/step - loss: 0.
0775 - acc: 0.9774 - val_loss: 0.0877 - val_acc: 0.9736
Epoch 3/20
60000/60000 [==============================] - 7s 111us/step - loss: 0.
0456 - acc: 0.9863 - val_loss: 0.0816 - val_acc: 0.9751
Epoch 4/20
60000/60000 [==============================] - 7s 109us/step - loss: 0.
0302 - acc: 0.9907 - val_loss: 0.0735 - val_acc: 0.9772
Epoch 5/20
60000/60000 [==============================] - 7s 109us/step - loss: 0.
0225 - acc: 0.9934 - val_loss: 0.0686 - val_acc: 0.9790
Epoch 6/20
60000/60000 [==============================] - 7s 108us/step - loss: 0.
0169 - acc: 0.9954 - val_loss: 0.0675 - val_acc: 0.9795
Epoch 7/20
60000/60000 [==============================] - 7s 117us/step - loss: 0.
0145 - acc: 0.9954 - val_loss: 0.0775 - val_acc: 0.9784
Epoch 8/20
60000/60000 [==============================] - 7s 109us/step - loss: 0.
0158 - acc: 0.9948 - val_loss: 0.0847 - val_acc: 0.9762
Epoch 9/20
60000/60000 [==============================] - 7s 109us/step - loss: 0.
0135 - acc: 0.9958 - val_loss: 0.0712 - val_acc: 0.9804
Epoch 10/20
60000/60000 [==============================] - 7s 109us/step - loss: 0.
0107 - acc: 0.9968 - val_loss: 0.0754 - val_acc: 0.9793
Epoch 11/20
60000/60000 [==============================] - 7s 109us/step - loss: 0.
0125 - acc: 0.9961 - val_loss: 0.0734 - val_acc: 0.9806
Epoch 12/20
60000/60000 [==============================] - 7s 109us/step - loss: 0.
```

```
0086 - acc: 0.9972 - val_loss: 0.0722 - val_acc: 0.9808
Epoch 13/20
60000/60000 [==============================] - 6s 108us/step - loss: 0.
0074 - acc: 0.9978 - val_loss: 0.0765 - val_acc: 0.9802
Epoch 14/20
60000/60000 [==============================] - 6s 108us/step - loss: 0.
0086 - acc: 0.9974 - val_loss: 0.0927 - val_acc: 0.9777
Epoch 15/20
60000/60000 [==============================] - 7s 110us/step - loss: 0.
0077 - acc: 0.9976 - val_loss: 0.0813 - val_acc: 0.9802
Epoch 16/20
60000/60000 [==============================] - 7s 109us/step - loss: 0.
0073 - acc: 0.9979 - val_loss: 0.0809 - val_acc: 0.9806
Epoch 17/20
60000/60000 [==============================] - 7s 110us/step - loss: 0.
0070 - acc: 0.9976 - val_loss: 0.1018 - val_acc: 0.9772
Epoch 18/20
60000/60000 [==============================] - 7s 109us/step - loss: 0.
0061 - acc: 0.9979 - val_loss: 0.1088 - val_acc: 0.9759
Epoch 19/20
60000/60000 [==============================] - 6s 108us/step - loss: 0.
0080 - acc: 0.9974 - val_loss: 0.0860 - val_acc: 0.9782
Epoch 20/20
60000/60000 [==============================] - 7s 109us/step - loss: 0.
0065 - acc: 0.9979 - val_loss: 0.0780 - val_acc: 0.9806
```

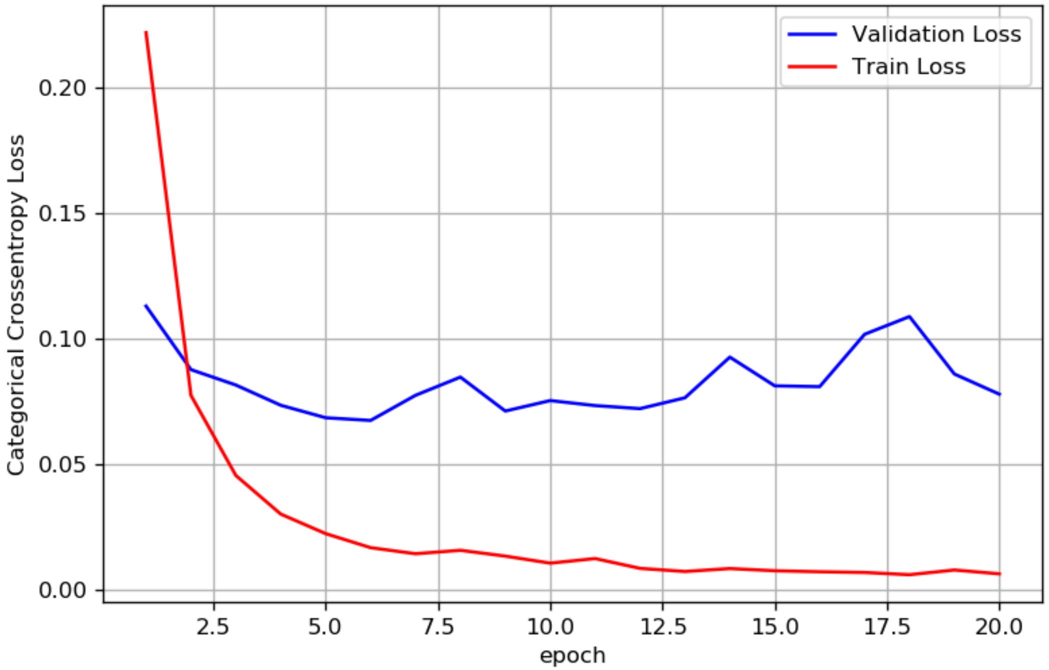
In [20]:

```
score= model_1B.evaluate(x_test, y_test, verbose=0)
print('Test score: ',score[0])
print('Test accuracy: ',score[1])
```

```
Test score:  0.07795610921084517
Test accuracy:  0.9806
```

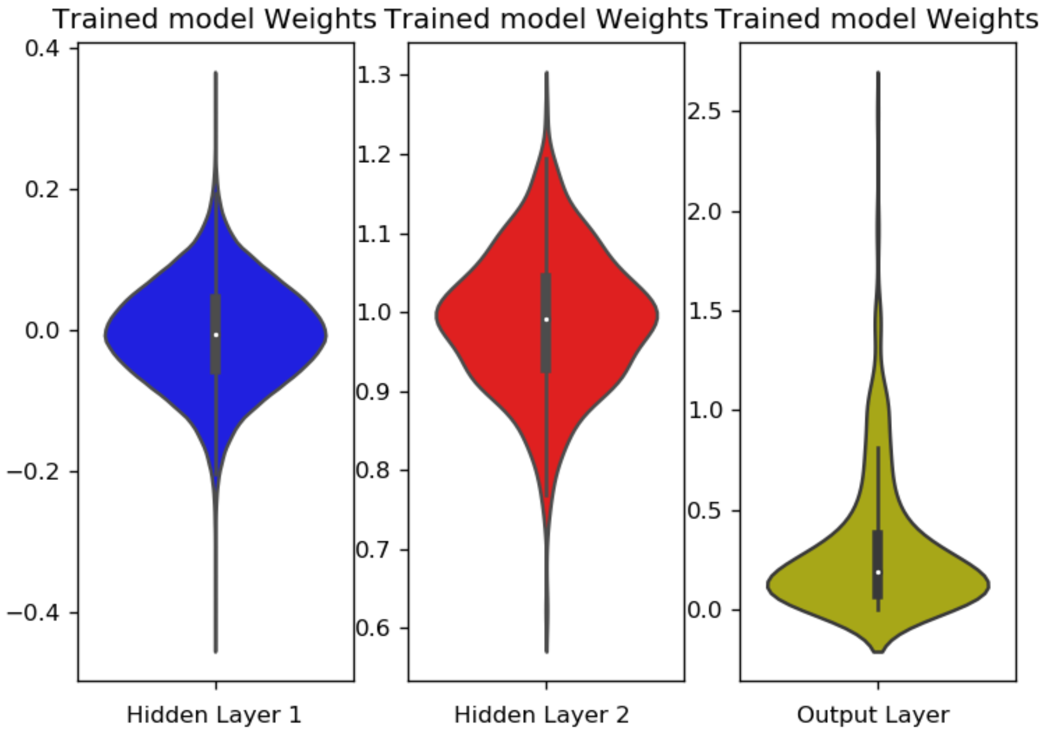
In [21]:

```
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')
x = list(range(1,21))
vy = history_1B.history['val_loss']
ty = history_1B.history['loss']
plt_dynamic(x, vy, ty, ax)
```



In [22]:
```
w_after = model_1B.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()
```



2 hidden layer with dropouts

In [23]:

```
#model declaration and initialization
model_1C= Sequential()

model_1C.add(Dense(400, activation='relu', \
        input_shape= (784,), kernel_initializer=keras.initializers.he_n
ormal(seed=None)))
model_1C.add(Dropout(0.25))

model_1C.add(Dense(100, activation='relu', \
        kernel_initializer= keras.initializers.he_normal(seed=None)))
model_1C.add(Dropout(0.25))

model_1C.add(Dense(10, activation='softmax'))

print(model_1C.summary())
```

```
WARNING:tensorflow:From C:\Anaconda3\lib\site-packages\keras\backend\te
nsorflow_backend.py:3445: calling dropout (from tensorflow.python.ops.n
n_ops) with keep_prob is deprecated and will be removed in a future ver
sion.
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_7 (Dense)              (None, 400)               314000
_________________________________________________________________
dropout_1 (Dropout)          (None, 400)               0
_________________________________________________________________
dense_8 (Dense)              (None, 100)               40100
_________________________________________________________________
dropout_2 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_9 (Dense)              (None, 10)                1010
=================================================================
Total params: 355,110
Trainable params: 355,110
Non-trainable params: 0
```

None

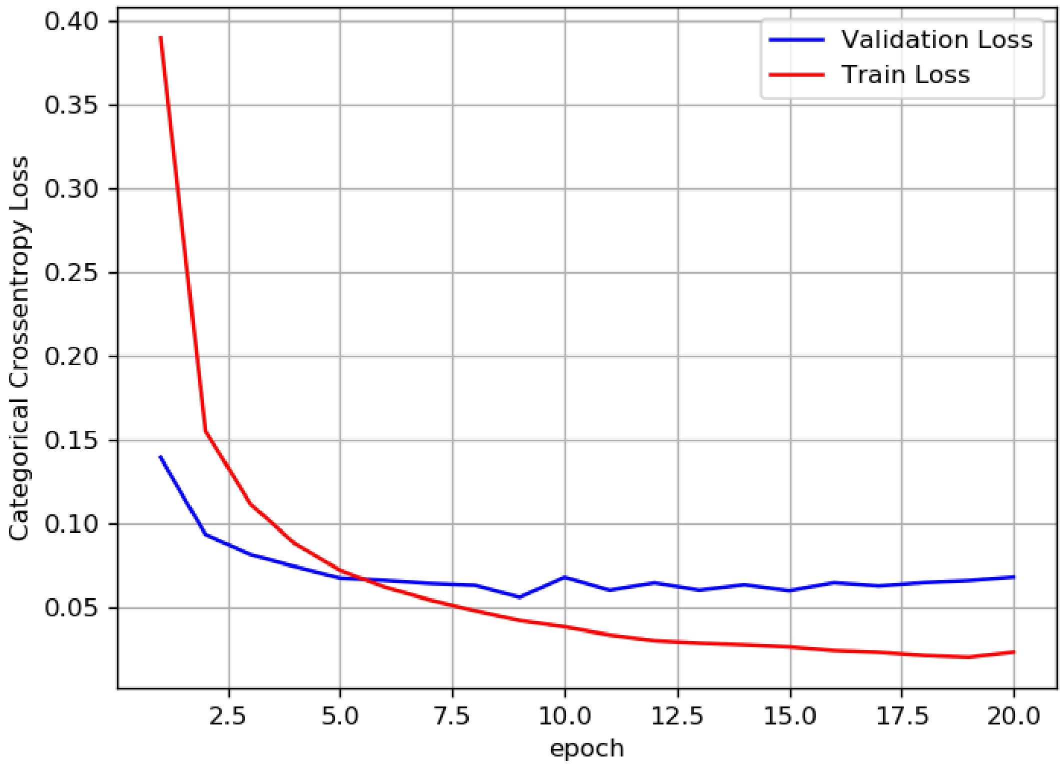
In [24]:

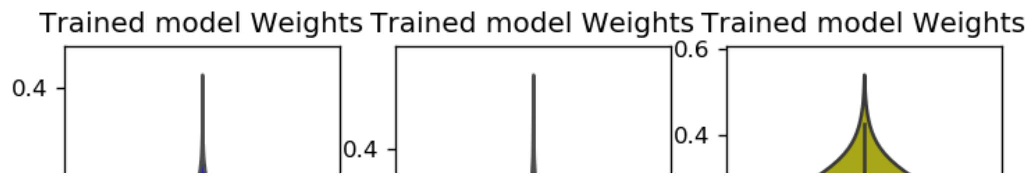
```
#run
model_1C.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history_1C = model_1C.fit(x_train,y_train,batch_size=200,epochs=20,verbose=1,validation_data=(x_test,y_test))
print('-------------------------------------------------------------')

score= model_1C.evaluate(x_test, y_test, verbose=0)
print('Test score: ',score[0])
print('Test accuracy: ',score[1])
print('-------------------------------------------------------------')

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')
x = list(range(1,21))
vy = history_1C.history['val_loss']
ty = history_1C.history['loss']
plt_dynamic(x, vy, ty, ax)
print('-------------------------------------------------------------')

w_after = model_1C.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 ')
```

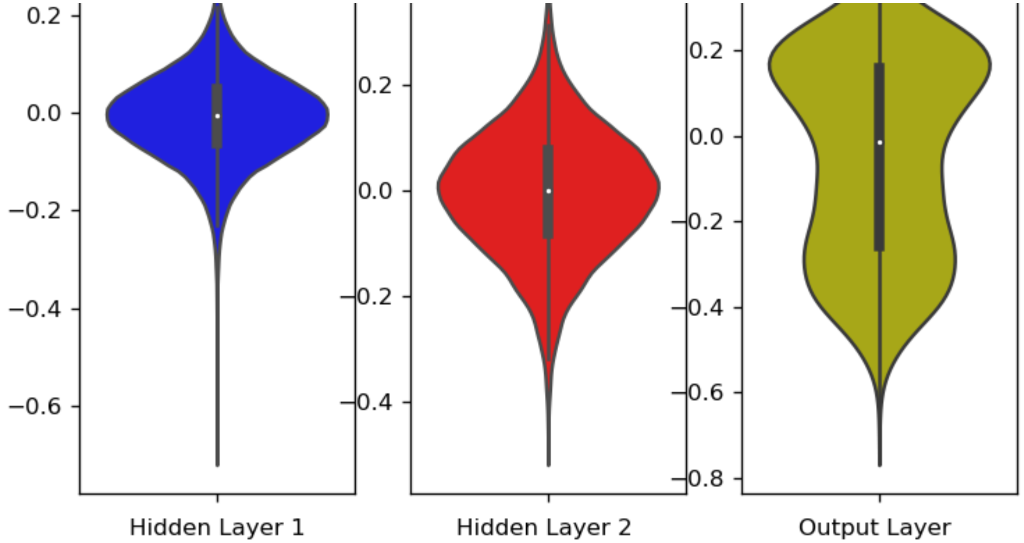
```python
plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 115us/step - loss: 0.
3900 - acc: 0.8815 - val_loss: 0.1394 - val_acc: 0.9575
Epoch 2/20
60000/60000 [==============================] - 6s 103us/step - loss: 0.
1554 - acc: 0.9531 - val_loss: 0.0936 - val_acc: 0.9718
Epoch 3/20
60000/60000 [==============================] - 6s 103us/step - loss: 0.
1119 - acc: 0.9662 - val_loss: 0.0815 - val_acc: 0.9733
Epoch 4/20
60000/60000 [==============================] - 6s 102us/step - loss: 0.
0879 - acc: 0.9723 - val_loss: 0.0743 - val_acc: 0.9765
Epoch 5/20
60000/60000 [==============================] - 6s 101us/step - loss: 0.
0721 - acc: 0.9768 - val_loss: 0.0675 - val_acc: 0.9794
Epoch 6/20
60000/60000 [==============================] - 6s 102us/step - loss: 0.
0620 - acc: 0.9804 - val_loss: 0.0661 - val_acc: 0.9793
Epoch 7/20
60000/60000 [==============================] - 6s 101us/step - loss: 0.
0544 - acc: 0.9831 - val_loss: 0.0643 - val_acc: 0.9809
Epoch 8/20
60000/60000 [==============================] - 6s 102us/step - loss: 0.
0480 - acc: 0.9850 - val_loss: 0.0632 - val_acc: 0.9812
Epoch 9/20
60000/60000 [==============================] - 6s 102us/step - loss: 0.
0422 - acc: 0.9869 - val_loss: 0.0560 - val_acc: 0.9832
Epoch 10/20
60000/60000 [==============================] - 6s 102us/step - loss: 0.
0384 - acc: 0.9875 - val_loss: 0.0680 - val_acc: 0.9812
Epoch 11/20
60000/60000 [==============================] - 6s 102us/step - loss: 0.
```

```
60000/60000 [==============================] - 6s 102us/step - loss: 0.
0335 - acc: 0.9892 - val_loss: 0.0602 - val_acc: 0.9831
Epoch 12/20
60000/60000 [==============================] - 6s 102us/step - loss: 0.
0301 - acc: 0.9901 - val_loss: 0.0646 - val_acc: 0.9823
Epoch 13/20
60000/60000 [==============================] - 6s 101us/step - loss: 0.
0287 - acc: 0.9908 - val_loss: 0.0602 - val_acc: 0.9836
Epoch 14/20
60000/60000 [==============================] - 6s 102us/step - loss: 0.
0277 - acc: 0.9908 - val_loss: 0.0634 - val_acc: 0.9833
Epoch 15/20
60000/60000 [==============================] - 6s 102us/step - loss: 0.
0264 - acc: 0.9915 - val_loss: 0.0599 - val_acc: 0.9839
Epoch 16/20
60000/60000 [==============================] - 6s 102us/step - loss: 0.
0242 - acc: 0.9920 - val_loss: 0.0647 - val_acc: 0.9827
Epoch 17/20
60000/60000 [==============================] - 6s 103us/step - loss: 0.
0232 - acc: 0.9920 - val_loss: 0.0628 - val_acc: 0.9832
Epoch 18/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.
0213 - acc: 0.9931 - val_loss: 0.0648 - val_acc: 0.9838
Epoch 19/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.
0202 - acc: 0.9931 - val_loss: 0.0660 - val_acc: 0.9828
Epoch 20/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.
0233 - acc: 0.9919 - val_loss: 0.0681 - val_acc: 0.9838
------------------------------------------------------------
Test score:  0.0681003057749891
Test accuracy:  0.9838
------------------------------------------------------------
```


Validation Loss
Train Loss
Categorical Crossentropy Loss
epoch
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Trained model Weights
Trained model Weights
Trained model Weights
0.4
0.4
0.6
0.4



2 hidden layer with dropouts and batch normalization

In [25]:
```
#model declaration and initialization
model_1D= Sequential()
model_1D.add(Dense(400, activation='relu', \
        input_shape= (784,), kernel_initializer=keras.initializers.he_n
ormal(seed=None)))
model_1D.add(BatchNormalization())
model_1D.add(Dropout(0.25))

model_1D.add(Dense(100, activation='relu', \
        kernel_initializer= keras.initializers.he_normal(seed=None)))
model_1D.add(BatchNormalization())
model_1D.add(Dropout(0.25))

model_1D.add(Dense(10, activation='softmax'))

print(model_1D.summary())
```

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_10 (Dense)             (None, 400)               314000
_________________________________________________________________
batch_normalization_3 (Batch (None, 400)               1600
_________________________________________________________________
dropout_3 (Dropout)          (None, 400)               0
_________________________________________________________________
dense_11 (Dense)             (None, 100)               40100
_________________________________________________________________
batch_normalization_4 (Batch (None, 100)               400
_________________________________________________________________
dropout_4 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_12 (Dense)             (None, 10)                1010
=================================================================
Total params: 357,110
Trainable params: 356,110
Non-trainable params: 1,000
_________________________________________________________________
None
```

In [26]:

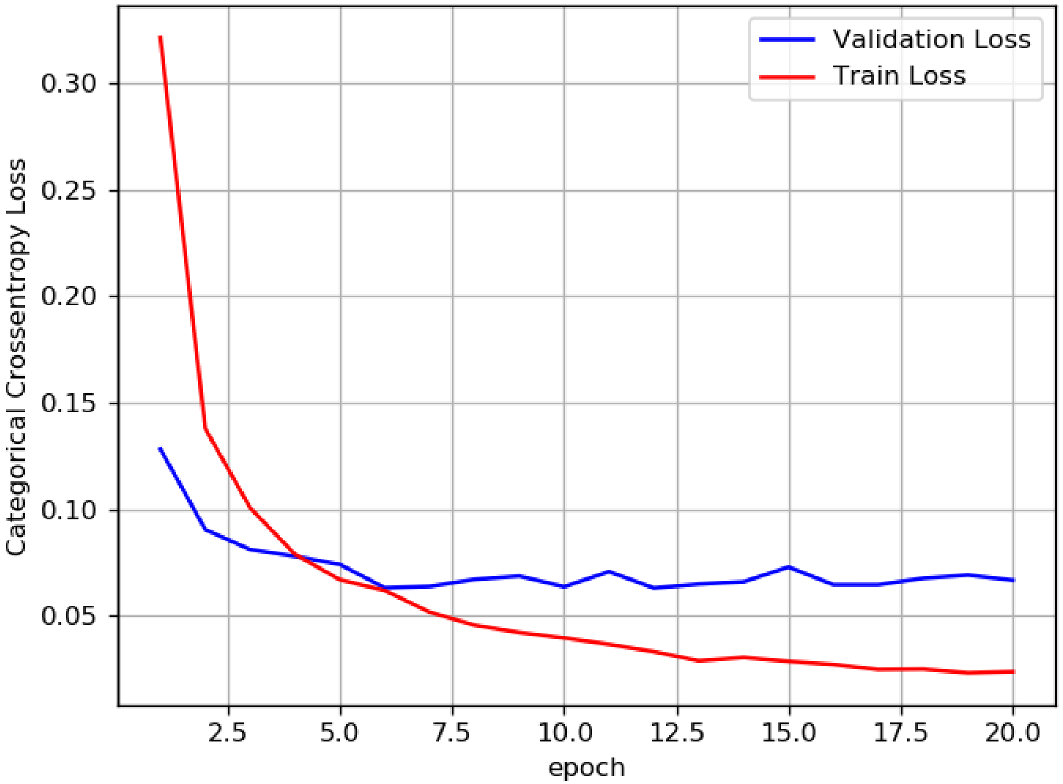
```
#run
model_1D.compile(optimizer='adam', loss='categorical_crossentropy', met
rics=['accuracy'])
history_1D = model_1D.fit(x_train,y_train,batch_size=200,epochs=20,verb
ose=1,validation_data=(x_test,y_test))
print('-------------------------------------------------------------')

score= model_1D.evaluate(x_test, y_test, verbose=0)
print('Test score: ',score[0])
print('Test accuracy: ',score[1])
print('-------------------------------------------------------------')

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')
x = list(range(1,21))
vy = history_1D.history['val_loss']
ty = history_1D.history['loss']
```

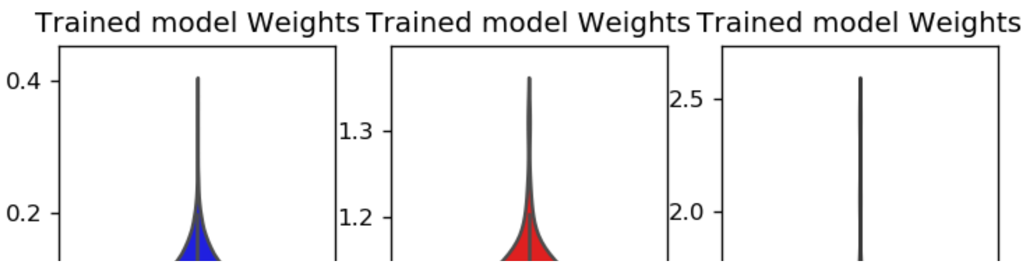
```
plt_dynamic(x, vy, ty, ax)
print('------------------------------------------------------------')

w_after = model_1D.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()
```

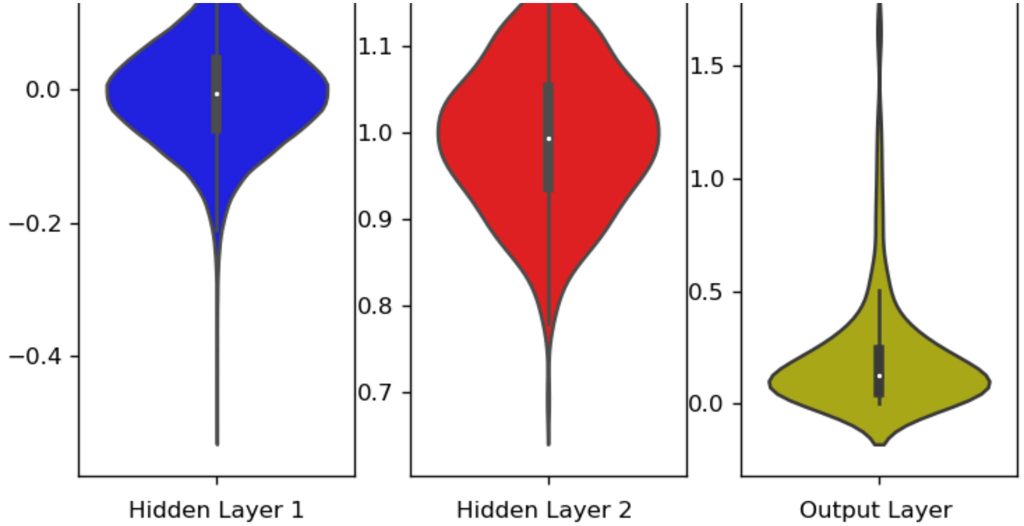
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.
3211 - acc: 0.9030 - val_loss: 0.1282 - val_acc: 0.9615
Epoch 2/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.
1379 - acc: 0.9589 - val_loss: 0.0905 - val_acc: 0.9708
Epoch 3/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.
1007 - acc: 0.9687 - val_loss: 0.0811 - val_acc: 0.9756
Epoch 4/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.
```

```
0788 - acc: 0.9754 - val_loss: 0.0779 - val_acc: 0.9765
Epoch 5/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.
0669 - acc: 0.9786 - val_loss: 0.0741 - val_acc: 0.9763- loss: 0.0665 -
acc
Epoch 6/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.
0619 - acc: 0.9799 - val_loss: 0.0631 - val_acc: 0.9791
Epoch 7/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.
0518 - acc: 0.9831 - val_loss: 0.0638 - val_acc: 0.9797
Epoch 8/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.
0456 - acc: 0.9855 - val_loss: 0.0670 - val_acc: 0.9787
Epoch 9/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.
0422 - acc: 0.9857 - val_loss: 0.0686 - val_acc: 0.9796
Epoch 10/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.
0396 - acc: 0.9869 - val_loss: 0.0637 - val_acc: 0.9805
Epoch 11/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.
0366 - acc: 0.9878 - val_loss: 0.0707 - val_acc: 0.9791
Epoch 12/20
60000/60000 [==============================] - 7s 117us/step - loss: 0.
0332 - acc: 0.9888 - val_loss: 0.0630 - val_acc: 0.9814
Epoch 13/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.
0289 - acc: 0.9903 - val_loss: 0.0649 - val_acc: 0.9798
Epoch 14/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.
0305 - acc: 0.9900 - val_loss: 0.0660 - val_acc: 0.9814
Epoch 15/20
60000/60000 [==============================] - 7s 117us/step - loss: 0.
0286 - acc: 0.9904 - val_loss: 0.0729 - val_acc: 0.9798
Epoch 16/20
60000/60000 [==============================] - 7s 117us/step - loss: 0.
0272 - acc: 0.9910 - val_loss: 0.0646 - val_acc: 0.9820
Epoch 17/20
```

```
60000/60000 [==============================] - 7s 117us/step - loss: 0.
0248 - acc: 0.9919 - val_loss: 0.0646 - val_acc: 0.9815
Epoch 18/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.
0250 - acc: 0.9915 - val_loss: 0.0676 - val_acc: 0.9822
Epoch 19/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.
0232 - acc: 0.9922 - val_loss: 0.0691 - val_acc: 0.9809
Epoch 20/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.
0238 - acc: 0.9921 - val_loss: 0.0667 - val_acc: 0.9822
------------------------------------------------------------
Test score:  0.06667604332640185
Test accuracy:  0.9822
------------------------------------------------------------
```


Validation Loss
Train Loss
Categorical Crossentropy Loss
0.30
0.25
0.20
0.15
0.10
0.05
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
epoch


Trained model Weights
Trained model Weights
Trained model Weights
0.4
0.2
1.3
1.2
2.5
2.0



MODEL2- with 3-Hidden layers

3 hidden layer simple

In [27]:
```
#model declaration and initialization
model_2A= Sequential()

model_2A.add(Dense(550, activation='relu', \
        input_shape= (784,), kernel_initializer=keras.initializers.he_n
ormal(seed=None)))

model_2A.add(Dense(300, activation='relu', \
        input_shape= (784,), kernel_initializer=keras.initializers.he_n
ormal(seed=None)))
```

```
model_2A.add(Dense(80, activation='relu', \
        kernel_initializer= keras.initializers.he_normal(seed=None)))

model_2A.add(Dense(10, activation='softmax'))

print(model_2A.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_13 (Dense)             (None, 550)               431750    
_________________________________________________________________
dense_14 (Dense)             (None, 300)               165300    
_________________________________________________________________
dense_15 (Dense)             (None, 80)                24080     
_________________________________________________________________
dense_16 (Dense)             (None, 10)                810       
=================================================================
Total params: 621,940
Trainable params: 621,940
Non-trainable params: 0
_________________________________________________________________
None
```

In [28]:

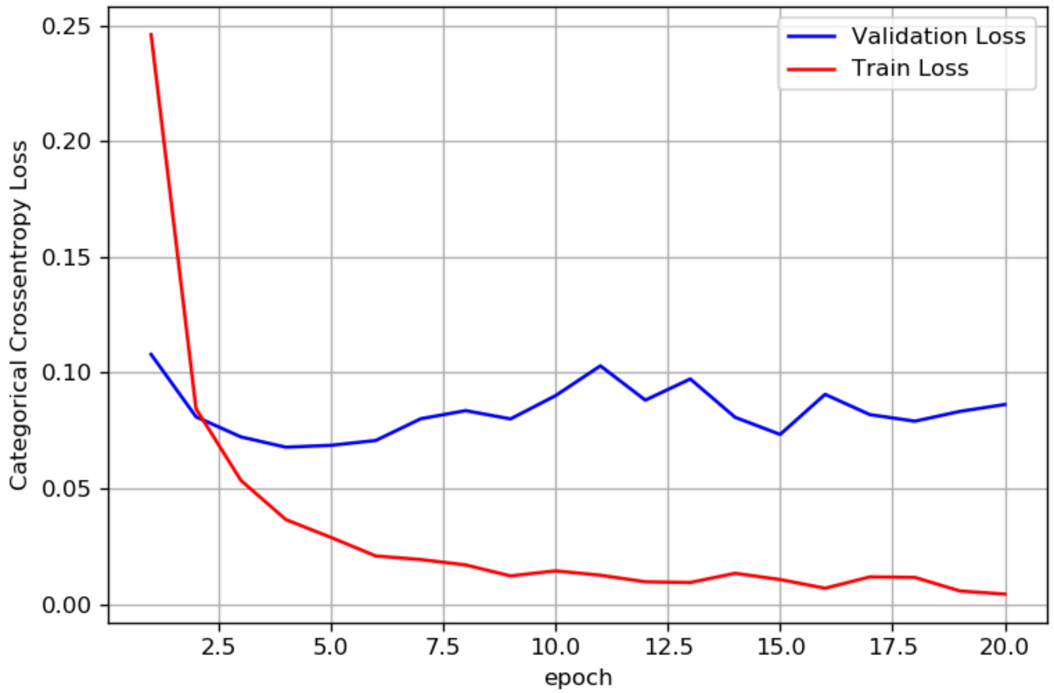
```
#run
model_2A.compile(optimizer='adam', loss='categorical_crossentropy', met
rics=['accuracy'])
history_2A = model_2A.fit(x_train,y_train,batch_size=200,epochs=20,verb
ose=1,validation_data=(x_test,y_test))
print('-------------------------------------------------------------')

score= model_2A.evaluate(x_test, y_test, verbose=0)
print('Test score: ',score[0])
print('Test accuracy: ',score[1])
print('-------------------------------------------------------------')

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')
x = list(range(1,21))
```

```
vy = history_2A.history['val_loss']
ty = history_2A.history['loss']
plt_dynamic(x, vy, ty, ax)
print('--------------------------------------------------------------')

w_after = model_2A.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()
```

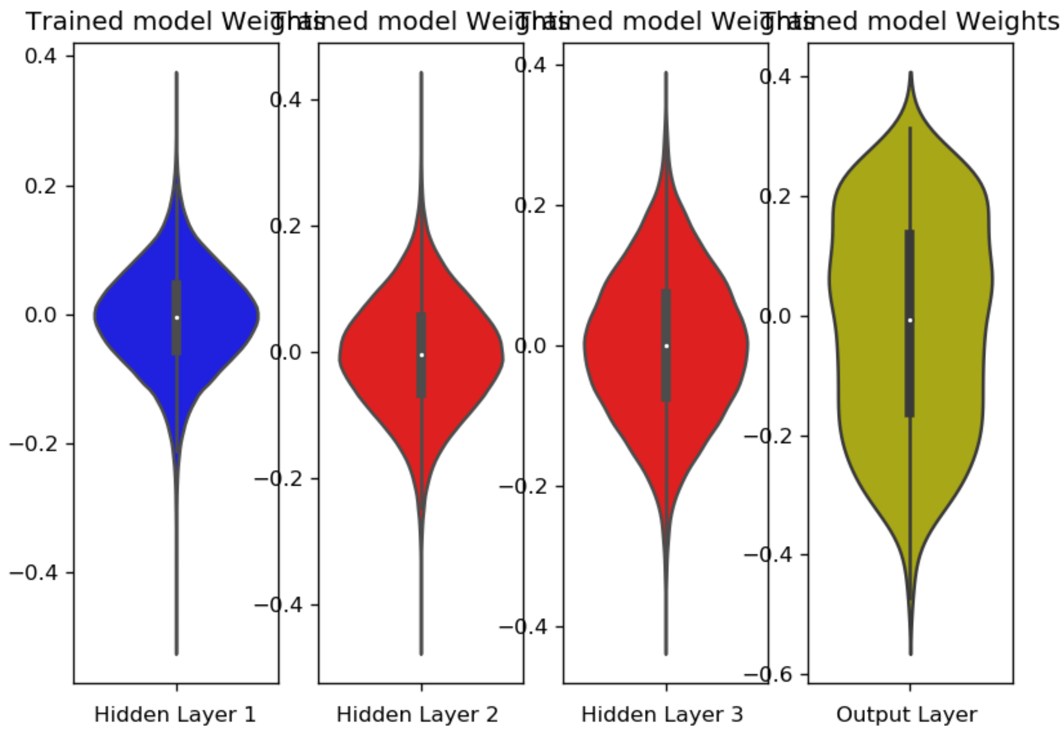
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 10s 169us/step - loss:
0.2461 - acc: 0.9276 - val_loss: 0.1079 - val_acc: 0.9672
Epoch 2/20
```

```
60000/60000 [==============================] - 9s 157us/step - loss: 0.
0844 - acc: 0.9741 - val_loss: 0.0810 - val_acc: 0.9731
Epoch 3/20
60000/60000 [==============================] - 9s 157us/step - loss: 0.
0535 - acc: 0.9831 - val_loss: 0.0723 - val_acc: 0.9764
Epoch 4/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.
0366 - acc: 0.9885 - val_loss: 0.0678 - val_acc: 0.9783
Epoch 5/20
60000/60000 [==============================] - 9s 154us/step - loss: 0.
0289 - acc: 0.9906 - val_loss: 0.0686 - val_acc: 0.9791
Epoch 6/20
60000/60000 [==============================] - 9s 154us/step - loss: 0.
0209 - acc: 0.9936 - val_loss: 0.0707 - val_acc: 0.9802
Epoch 7/20
60000/60000 [==============================] - 9s 154us/step - loss: 0.
0193 - acc: 0.9937 - val_loss: 0.0801 - val_acc: 0.9790
Epoch 8/20
60000/60000 [==============================] - 9s 154us/step - loss: 0.
0170 - acc: 0.9941 - val_loss: 0.0836 - val_acc: 0.9774
Epoch 9/20
60000/60000 [==============================] - 9s 153us/step - loss: 0.
0122 - acc: 0.9962 - val_loss: 0.0801 - val_acc: 0.9802
Epoch 10/20
60000/60000 [==============================] - 9s 153us/step - loss: 0.
0144 - acc: 0.9954 - val_loss: 0.0901 - val_acc: 0.9778
Epoch 11/20
60000/60000 [==============================] - 9s 152us/step - loss: 0.
0125 - acc: 0.9960 - val_loss: 0.1030 - val_acc: 0.9770
Epoch 12/20
60000/60000 [==============================] - 9s 153us/step - loss: 0.
0097 - acc: 0.9970 - val_loss: 0.0882 - val_acc: 0.9809
Epoch 13/20
60000/60000 [==============================] - 9s 153us/step - loss: 0.
0094 - acc: 0.9971 - val_loss: 0.0973 - val_acc: 0.9786
Epoch 14/20
60000/60000 [==============================] - 9s 153us/step - loss: 0.
0134 - acc: 0.9956 - val_loss: 0.0807 - val_acc: 0.9832

Epoch 15/20
```

```
60000/60000 [==============================] - 9s 152us/step - loss: 0.
0107 - acc: 0.9967 - val_loss: 0.0733 - val_acc: 0.9833
Epoch 16/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.
0069 - acc: 0.9979 - val_loss: 0.0907 - val_acc: 0.9824
Epoch 17/20
60000/60000 [==============================] - 9s 153us/step - loss: 0.
0118 - acc: 0.9962 - val_loss: 0.0819 - val_acc: 0.9820
Epoch 18/20
60000/60000 [==============================] - 9s 154us/step - loss: 0.
0116 - acc: 0.9964 - val_loss: 0.0791 - val_acc: 0.9828
Epoch 19/20
60000/60000 [==============================] - 9s 153us/step - loss: 0.
0057 - acc: 0.9981 - val_loss: 0.0833 - val_acc: 0.9829
Epoch 20/20
60000/60000 [==============================] - 9s 152us/step - loss: 0.
0044 - acc: 0.9988 - val_loss: 0.0863 - val_acc: 0.9818
------------------------------------------------------------
Test score:  0.08627982623030493
Test accuracy:  0.9818
------------------------------------------------------------
```


Validation Loss
Train Loss
0.25
0.20
0.15
0.10
0.05
0.00
Categorical Crossentropy Loss
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
epoch

- -



3 hidden layer with batch normalization

In [29]:

```
#model declaration and initialization
model_2B= Sequential()

model_2B.add(Dense(550, activation='relu', \
        input_shape= (784,), kernel_initializer=keras.initializers.he_n
ormal(seed=None)))
model_2B.add(BatchNormalization())

model_2B.add(Dense(300, activation='relu', \
        input_shape= (784,), kernel_initializer=keras.initializers.he_n
ormal(seed=None)))
model_2B.add(BatchNormalization())

model_2B.add(Dense(80, activation='relu', \
        kernel_initializer= keras.initializers.he_normal(seed=None)))
model_2B.add(BatchNormalization())

model_2B.add(Dense(10, activation='softmax'))

print(model_2B.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_17 (Dense)             (None, 550)               431750
_________________________________________________________________
batch_normalization_5 (Batch (None, 550)               2200
_________________________________________________________________
dense_18 (Dense)             (None, 300)               165300
_________________________________________________________________
batch_normalization_6 (Batch (None, 300)               1200
_________________________________________________________________
dense_19 (Dense)             (None, 80)                24080
_________________________________________________________________
batch_normalization_7 (Batch (None, 80)                320
_________________________________________________________________
dense_20 (Dense)             (None, 10)                810
=================================================================
Total params: 625,660
Trainable params: 623,800
```

```
Non-trainable params: 1,860
_________________________________________________________________
None
```

In [30]:

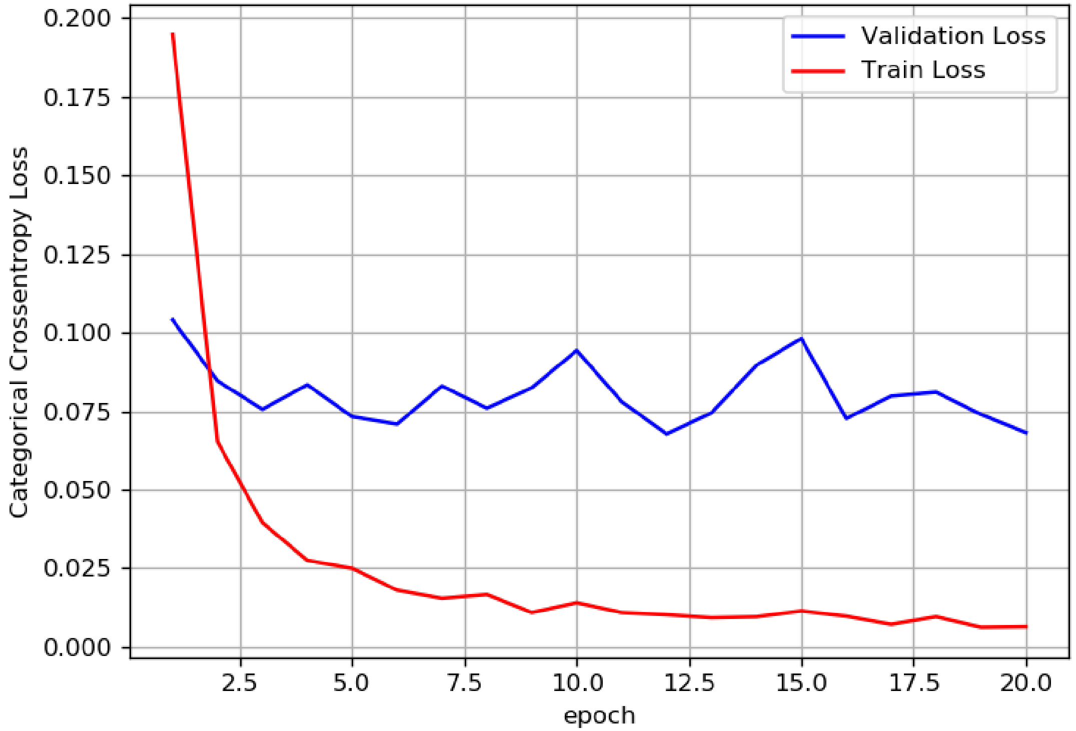
```
#run
model_2B.compile(optimizer='adam', loss='categorical_crossentropy', met
rics=['accuracy'])
history_2B = model_2B.fit(x_train,y_train,batch_size=200,epochs=20,verb
ose=1,validation_data=(x_test,y_test))
print('-------------------------------------------------------------')

score= model_2B.evaluate(x_test, y_test, verbose=0)
print('Test score: ',score[0])
print('Test accuracy: ',score[1])
print('-------------------------------------------------------------')

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')
x = list(range(1,21))
vy = history_2B.history['val_loss']
ty = history_2B.history['loss']
plt_dynamic(x, vy, ty, ax)
print('-------------------------------------------------------------')

w_after = model_2B.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()
```

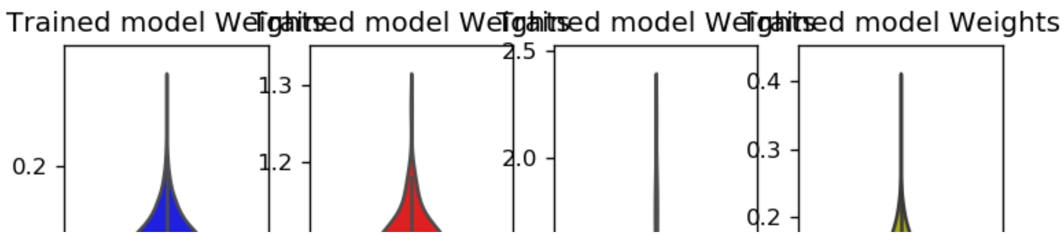
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 12s 208us/step - loss:
0.1949 - acc: 0.9416 - val_loss: 0.1041 - val_acc: 0.9672
Epoch 2/20
60000/60000 [==============================] - 11s 184us/step - loss:
0.0653 - acc: 0.9807 - val_loss: 0.0846 - val_acc: 0.9741
Epoch 3/20
60000/60000 [==============================] - 11s 184us/step - loss:
0.0395 - acc: 0.9878 - val_loss: 0.0756 - val_acc: 0.9765
Epoch 4/20
60000/60000 [==============================] - 11s 182us/step - loss:
0.0275 - acc: 0.9912 - val_loss: 0.0834 - val_acc: 0.9742
Epoch 5/20
60000/60000 [==============================] - 11s 182us/step - loss:
0.0250 - acc: 0.9920 - val_loss: 0.0733 - val_acc: 0.9774
Epoch 6/20
60000/60000 [==============================] - 11s 181us/step - loss:
0.0180 - acc: 0.9942 - val_loss: 0.0709 - val_acc: 0.9789
Epoch 7/20
60000/60000 [==============================] - 11s 183us/step - loss:
0.0154 - acc: 0.9951 - val_loss: 0.0830 - val_acc: 0.9767
Epoch 8/20

60000/60000 [==============================] - 11s 180us/step - loss:
0.0166 - acc: 0.9947 - val_loss: 0.0760 - val_acc: 0.9778
```

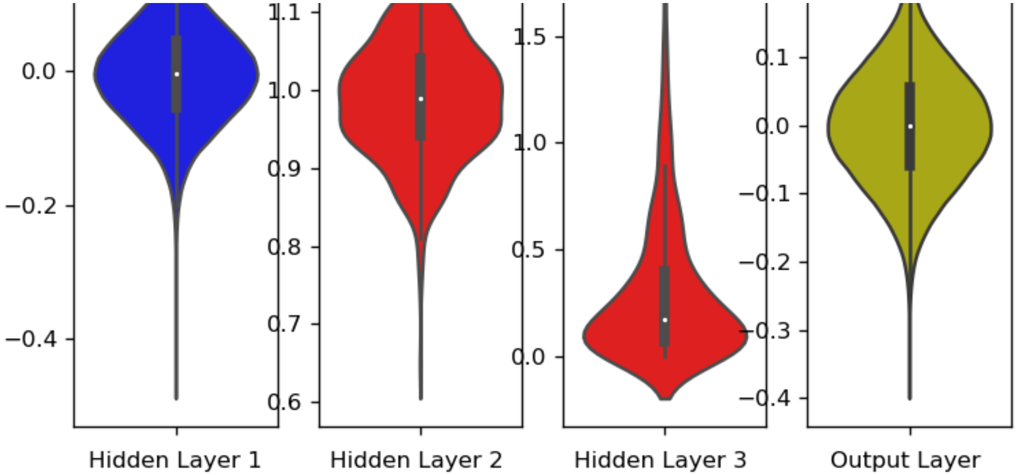
```
Epoch 9/20
60000/60000 [==============================] - 11s 181us/step - loss:
0.0109 - acc: 0.9965 - val_loss: 0.0824 - val_acc: 0.9771
Epoch 10/20
60000/60000 [==============================] - 11s 181us/step - loss:
0.0140 - acc: 0.9951 - val_loss: 0.0943 - val_acc: 0.9746
Epoch 11/20
60000/60000 [==============================] - 11s 181us/step - loss:
0.0109 - acc: 0.9963 - val_loss: 0.0780 - val_acc: 0.9786
Epoch 12/20
60000/60000 [==============================] - 11s 182us/step - loss:
0.0102 - acc: 0.9967 - val_loss: 0.0677 - val_acc: 0.9814
Epoch 13/20
60000/60000 [==============================] - 11s 181us/step - loss:
0.0093 - acc: 0.9970 - val_loss: 0.0745 - val_acc: 0.9814
Epoch 14/20
60000/60000 [==============================] - 11s 184us/step - loss:
0.0096 - acc: 0.9968 - val_loss: 0.0896 - val_acc: 0.9772
Epoch 15/20
60000/60000 [==============================] - 11s 185us/step - loss:
0.0114 - acc: 0.9963 - val_loss: 0.0981 - val_acc: 0.9749
Epoch 16/20
60000/60000 [==============================] - 11s 185us/step - loss:
0.0098 - acc: 0.9966 - val_loss: 0.0727 - val_acc: 0.9811
Epoch 17/20
60000/60000 [==============================] - 11s 187us/step - loss:
0.0072 - acc: 0.9977 - val_loss: 0.0798 - val_acc: 0.9815
Epoch 18/20
60000/60000 [==============================] - 11s 184us/step - loss:
0.0096 - acc: 0.9967 - val_loss: 0.0811 - val_acc: 0.9792
Epoch 19/20
60000/60000 [==============================] - 11s 184us/step - loss:
0.0063 - acc: 0.9979 - val_loss: 0.0740 - val_acc: 0.9828
Epoch 20/20
60000/60000 [==============================] - 11s 186us/step - loss:
0.0064 - acc: 0.9981 - val_loss: 0.0682 - val_acc: 0.9824
------------------------------------------------------------

Test score:  0.06820325384677489
Test accuracy:  0.9824
```

```
Test accuracy: 0.9824
--------------------------------------------------------------------------------------------------------
```




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





3 hidden layer with dropouts

In [31]:

```
#model declaration and initialization
model_2C= Sequential()

model_2C.add(Dense(550, activation='relu', \
        input_shape= (784,), kernel_initializer=keras.initializers.he_n
ormal(seed=None)))
model_2C.add(Dropout(0.25))

model_2C.add(Dense(300, activation='relu', \
        input_shape= (784,), kernel_initializer=keras.initializers.he_n
ormal(seed=None)))
model_2C.add(Dropout(0.25))

model_2C.add(Dense(80, activation='relu', \
        kernel_initializer= keras.initializers.he_normal(seed=None)))
model_2C.add(Dropout(0.25))

model_2C.add(Dense(10, activation='softmax'))

print(model_2A.summary())
```

```
Layer (type)                 Output Shape              Param #   
=================================================================
dense_13 (Dense)             (None, 550)               431750    
_________________________________________________________________
dense_14 (Dense)             (None, 300)               165300    
_________________________________________________________________
dense_15 (Dense)             (None, 80)                24080     
_________________________________________________________________
dense_16 (Dense)             (None, 10)                810       
=================================================================
Total params: 621,940
Trainable params: 621,940
Non-trainable params: 0
_________________________________________________________________
None
```

In [32]:

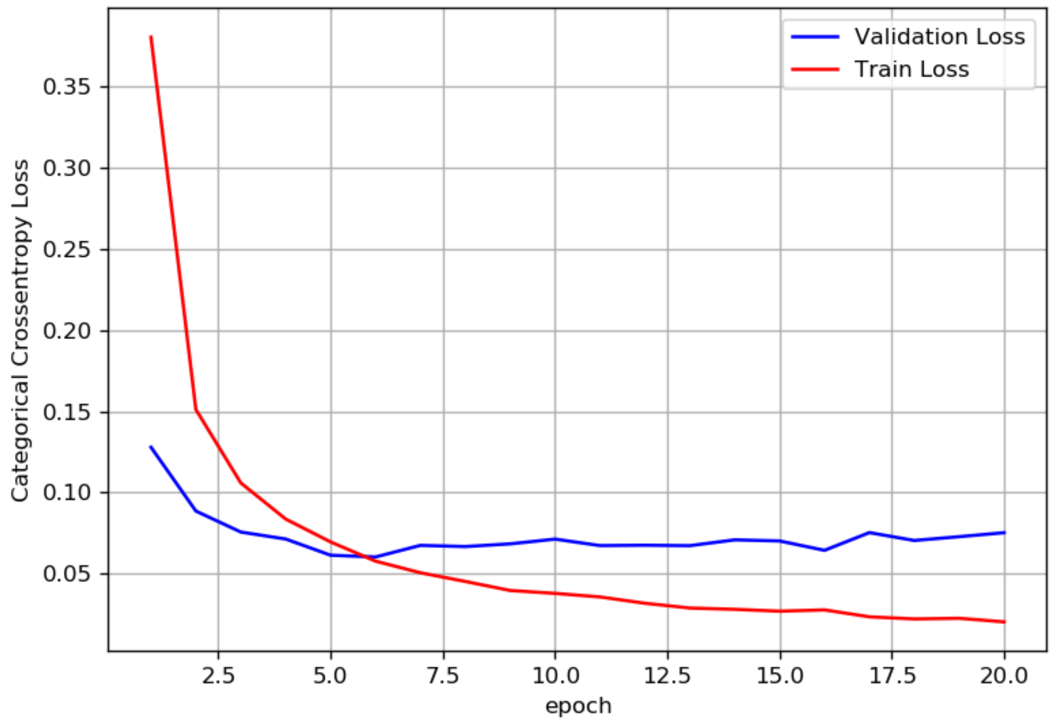
```
#run
model_2C.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history_2C = model_2C.fit(x_train,y_train,batch_size=200,epochs=20,verbose=1,validation_data=(x_test,y_test))
print('--------------------------------------------------------------')

score= model_2C.evaluate(x_test, y_test, verbose=0)
print('Test score: ',score[0])
print('Test accuracy: ',score[1])
print('--------------------------------------------------------------')

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')
x = list(range(1,21))
vy = history_2C.history['val_loss']
ty = history_2C.history['loss']
plt_dynamic(x, vy, ty, ax)
print('--------------------------------------------------------------')

w_after = model_2C.get_weights()
```

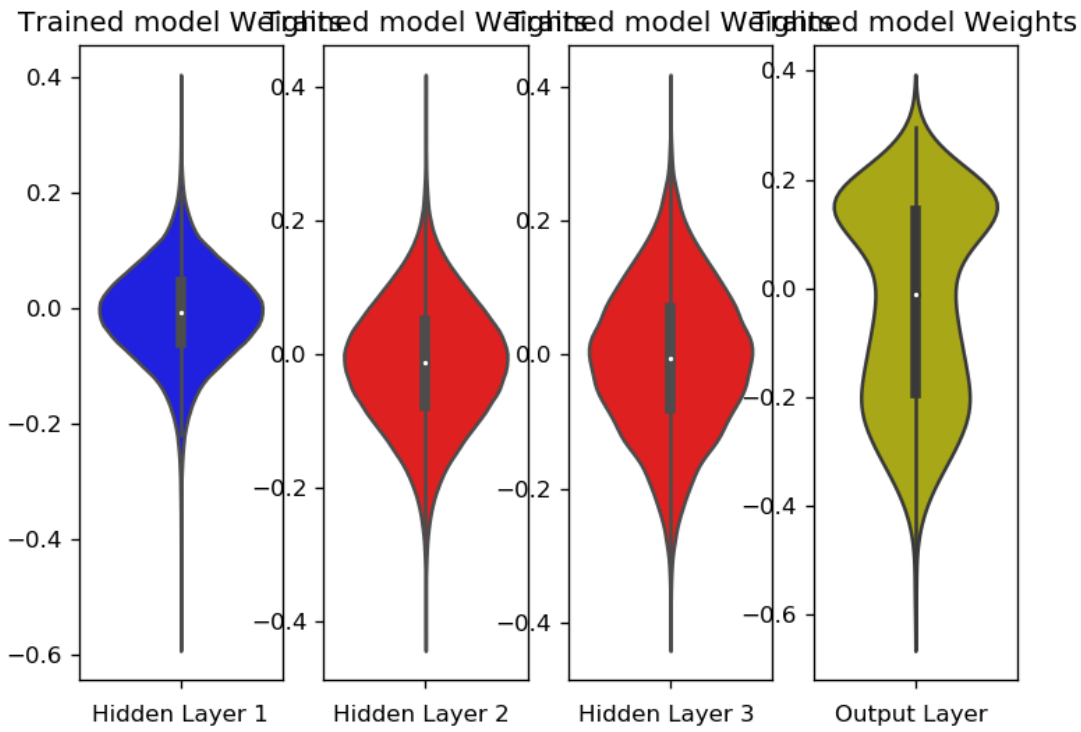
```
h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
out_w = w_after[6].flatten().reshape(-1,1)

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()
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 12s 194us/step - loss:
0.3805 - acc: 0.8839 - val_loss: 0.1279 - val_acc: 0.9601
Epoch 2/20
60000/60000 [==============================] - 10s 175us/step - loss:
0.1511 - acc: 0.9550 - val_loss: 0.0886 - val_acc: 0.9725
Epoch 3/20
60000/60000 [==============================] - 11s 175us/step - loss:
0.1060 - acc: 0.9681 - val_loss: 0.0757 - val_acc: 0.9763
Epoch 4/20
```

```
60000/60000 [==============================] - 11s 176us/step - loss:
0.0837 - acc: 0.9751 - val_loss: 0.0714 - val_acc: 0.9793
Epoch 5/20
60000/60000 [==============================] - 11s 177us/step - loss:
0.0696 - acc: 0.9795 - val_loss: 0.0614 - val_acc: 0.9822
Epoch 6/20
60000/60000 [==============================] - 11s 176us/step - loss:
0.0578 - acc: 0.9821 - val_loss: 0.0603 - val_acc: 0.9833
Epoch 7/20
60000/60000 [==============================] - 11s 178us/step - loss:
0.0506 - acc: 0.9842 - val_loss: 0.0675 - val_acc: 0.9818
Epoch 8/20
60000/60000 [==============================] - 11s 178us/step - loss:
0.0452 - acc: 0.9858 - val_loss: 0.0668 - val_acc: 0.9824
Epoch 9/20
60000/60000 [==============================] - 10s 175us/step - loss:
0.0397 - acc: 0.9877 - val_loss: 0.0684 - val_acc: 0.9800
Epoch 10/20
60000/60000 [==============================] - 10s 174us/step - loss:
0.0379 - acc: 0.9887 - val_loss: 0.0713 - val_acc: 0.9817
Epoch 11/20
60000/60000 [==============================] - 11s 177us/step - loss:
0.0357 - acc: 0.9886 - val_loss: 0.0673 - val_acc: 0.9815
Epoch 12/20
60000/60000 [==============================] - 11s 178us/step - loss:
0.0318 - acc: 0.9897 - val_loss: 0.0676 - val_acc: 0.9822
Epoch 13/20
60000/60000 [==============================] - 11s 177us/step - loss:
0.0289 - acc: 0.9911 - val_loss: 0.0673 - val_acc: 0.9834
Epoch 14/20
60000/60000 [==============================] - 11s 176us/step - loss:
0.0281 - acc: 0.9912 - val_loss: 0.0709 - val_acc: 0.9820
Epoch 15/20
60000/60000 [==============================] - 11s 178us/step - loss:
0.0269 - acc: 0.9916 - val_loss: 0.0702 - val_acc: 0.9816
Epoch 16/20
60000/60000 [==============================] - 11s 178us/step - loss:

0.0277 - acc: 0.9914 - val_loss: 0.0644 - val_acc: 0.9830
Epoch 17/20
```

```
60000/60000 [==============================] - 11s 179us/step - loss:
0.0235 - acc: 0.9923 - val_loss: 0.0753 - val_acc: 0.9836
Epoch 18/20
60000/60000 [==============================] - 11s 179us/step - loss:
0.0221 - acc: 0.9929 - val_loss: 0.0705 - val_acc: 0.9850
Epoch 19/20
60000/60000 [==============================] - 11s 179us/step - loss:
0.0225 - acc: 0.9930 - val_loss: 0.0728 - val_acc: 0.9830
Epoch 20/20
60000/60000 [==============================] - 11s 177us/step - loss:
0.0203 - acc: 0.9939 - val_loss: 0.0753 - val_acc: 0.9836
------------------------------------------------------------
Test score:  0.07527206434430508
Test accuracy:  0.9836
------------------------------------------------------------
```


Validation Loss
Train Loss
Categorical Crossentropy Loss
0.35
0.30
0.25
0.20
0.15
0.10
0.05
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
epoch



3 hidden layer with dropouts and batch normalization

In [33]:
```
#model declaration and initialization
model_2D= Sequential()
```

```
model_2D.add(Dense(550, activation='relu', \
        input_shape= (784,), kernel_initializer=keras.initializers.he_n
ormal(seed=None)))
model_2D.add(BatchNormalization())
model_2D.add(Dropout(0.25))

model_2D.add(Dense(300, activation='relu', \
        input_shape= (784,), kernel_initializer=keras.initializers.he_n
ormal(seed=None)))
model_2D.add(BatchNormalization())
model_2D.add(Dropout(0.25))

model_2D.add(Dense(80, activation='relu', \
        kernel_initializer= keras.initializers.he_normal(seed=None)))
model_2D.add(BatchNormalization())
model_2D.add(Dropout(0.25))

model_2D.add(Dense(10, activation='softmax'))

print(model_2D.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_25 (Dense)             (None, 550)               431750    
_________________________________________________________________
batch_normalization_8 (Batch (None, 550)               2200      
_________________________________________________________________
dropout_8 (Dropout)          (None, 550)               0         
_________________________________________________________________
dense_26 (Dense)             (None, 300)               165300    
_________________________________________________________________
batch_normalization_9 (Batch (None, 300)               1200      
_________________________________________________________________
dropout_9 (Dropout)          (None, 300)               0         
_________________________________________________________________
dense_27 (Dense)             (None, 80)                24080     
_________________________________________________________________
batch_normalization_10 (Batc (None, 80)                320       
```

```
batch_normalization_10 (Batc (None, 80)                320
_________________________________________________________________
dropout_10 (Dropout)         (None, 80)                0
_________________________________________________________________
dense_28 (Dense)             (None, 10)                810
=================================================================
Total params: 625,660
Trainable params: 623,800
Non-trainable params: 1,860
_________________________________________________________________
None
```

In [34]:

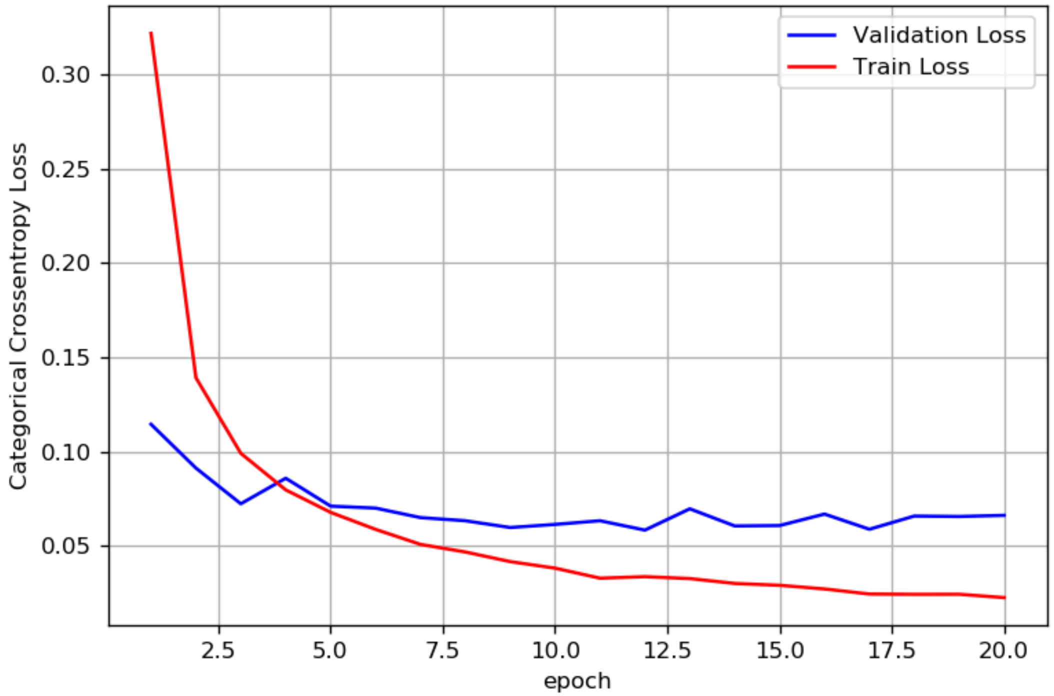
```
#run
model_2D.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history_2D = model_2D.fit(x_train,y_train,batch_size=200,epochs=20,verbose=1,validation_data=(x_test,y_test))
print('-------------------------------------------------------------')

score= model_2D.evaluate(x_test, y_test, verbose=0)
print('Test score: ',score[0])
print('Test accuracy: ',score[1])
print('-------------------------------------------------------------')

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')
x = list(range(1,21))
vy = history_2D.history['val_loss']
ty = history_2D.history['loss']
plt_dynamic(x, vy, ty, ax)
print('-------------------------------------------------------------')

w_after = model_2D.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()
```

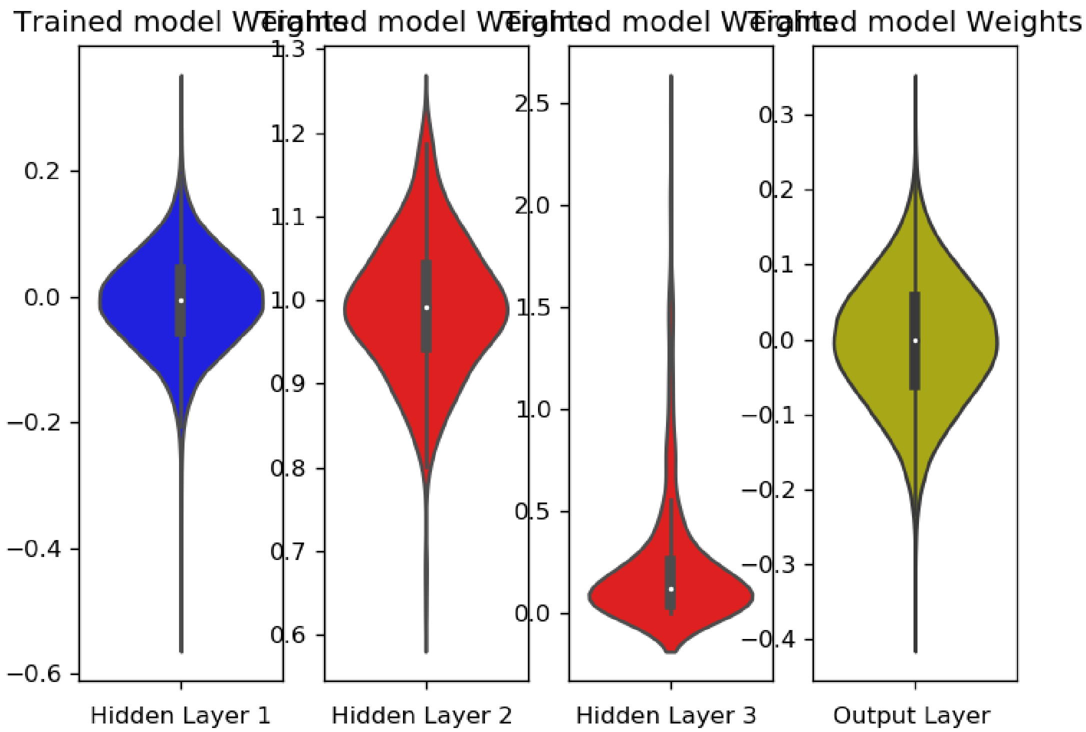
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 14s 235us/step - loss:
0.3217 - acc: 0.9036 - val_loss: 0.1146 - val_acc: 0.9640
Epoch 2/20
60000/60000 [==============================] - 12s 205us/step - loss:
0.1392 - acc: 0.9585 - val_loss: 0.0913 - val_acc: 0.9714
Epoch 3/20
60000/60000 [==============================] - 12s 205us/step - loss:
0.0991 - acc: 0.9693 - val_loss: 0.0724 - val_acc: 0.9767
Epoch 4/20
60000/60000 [==============================] - 12s 203us/step - loss:
0.0798 - acc: 0.9749 - val_loss: 0.0860 - val_acc: 0.9733
Epoch 5/20
60000/60000 [==============================] - 12s 204us/step - loss:
0.0679 - acc: 0.9787 - val_loss: 0.0712 - val_acc: 0.9789
Epoch 6/20
```

```
60000/60000 [==============================] - 12s 206us/step - loss:
0.0589 - acc: 0.9811 - val_loss: 0.0701 - val_acc: 0.9783
Epoch 7/20
60000/60000 [==============================] - 12s 204us/step - loss:
0.0509 - acc: 0.9834 - val_loss: 0.0651 - val_acc: 0.9818
Epoch 8/20
60000/60000 [==============================] - 12s 204us/step - loss:
0.0469 - acc: 0.9849 - val_loss: 0.0634 - val_acc: 0.9803
Epoch 9/20
60000/60000 [==============================] - 12s 205us/step - loss:
0.0418 - acc: 0.9865 - val_loss: 0.0598 - val_acc: 0.9822
Epoch 10/20
60000/60000 [==============================] - 12s 203us/step - loss:
0.0383 - acc: 0.9878 - val_loss: 0.0615 - val_acc: 0.9828
Epoch 11/20
60000/60000 [==============================] - 12s 203us/step - loss:
0.0330 - acc: 0.9891 - val_loss: 0.0634 - val_acc: 0.9817
Epoch 12/20
60000/60000 [==============================] - 12s 203us/step - loss:
0.0338 - acc: 0.9891 - val_loss: 0.0585 - val_acc: 0.9841
Epoch 13/20
60000/60000 [==============================] - 12s 203us/step - loss:
0.0328 - acc: 0.9886 - val_loss: 0.0697 - val_acc: 0.9806
Epoch 14/20
60000/60000 [==============================] - 12s 204us/step - loss:
0.0302 - acc: 0.9900 - val_loss: 0.0607 - val_acc: 0.9835
Epoch 15/20
60000/60000 [==============================] - 12s 203us/step - loss:
0.0292 - acc: 0.9904 - val_loss: 0.0609 - val_acc: 0.9830
Epoch 16/20
60000/60000 [==============================] - 12s 200us/step - loss:
0.0273 - acc: 0.9907 - val_loss: 0.0670 - val_acc: 0.9817
Epoch 17/20
60000/60000 [==============================] - 12s 203us/step - loss:
0.0246 - acc: 0.9921 - val_loss: 0.0589 - val_acc: 0.9846
Epoch 18/20
60000/60000 [==============================] - 12s 204us/step - loss:
0.0244 - acc: 0.9923 - val_loss: 0.0659 - val_acc: 0.9820
Epoch 19/20
```

```
60000/60000 [==============================] - 12s 203us/step - loss:
0.0245 - acc: 0.9923 - val_loss: 0.0657 - val_acc: 0.9823
Epoch 20/20
60000/60000 [==============================] - 12s 203us/step - loss:
0.0227 - acc: 0.9929 - val_loss: 0.0663 - val_acc: 0.9838
------------------------------------------------------------
Test score:  0.06628003185314228
Test accuracy:  0.9838
------------------------------------------------------------
```




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



MODEL3- with 5-Hidden layers

5 hidden layer simple

In [44]:
```
#model declaration and initialization
model_3A= Sequential()

model_3A.add(Dense(630, activation='relu', \
```

```
        input_shape= (784,), kernel_initializer=keras.initializers.he_n
ormal(seed=None)))

model_3A.add(Dense(480, activation='relu', \
        kernel_initializer= keras.initializers.he_normal(seed=None)))

model_3A.add(Dense(330, activation='relu', \
        input_shape= (784,), kernel_initializer=keras.initializers.he_n
ormal(seed=None)))

model_3A.add(Dense(180, activation='relu', \
        kernel_initializer= keras.initializers.he_normal(seed=None)))

model_3A.add(Dense(80, activation='relu', \
        kernel_initializer= keras.initializers.he_normal(seed=None)))

model_3A.add(Dense(10, activation='softmax'))

print(model_3A.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_53 (Dense)             (None, 630)               494550    
_________________________________________________________________
dense_54 (Dense)             (None, 480)               302880    
_________________________________________________________________
dense_55 (Dense)             (None, 330)               158730    
_________________________________________________________________
dense_56 (Dense)             (None, 180)               59580     
_________________________________________________________________
dense_57 (Dense)             (None, 80)                14480     
_________________________________________________________________
dense_58 (Dense)             (None, 10)                810       
=================================================================
Total params: 1,031,030
Trainable params: 1,031,030
Non-trainable params: 0
_________________________________________________________________
None
```

In [45]:

```
#run
model_3A.compile(optimizer='adam', loss='categorical_crossentropy', met
rics=['accuracy'])
history_3A = model_3A.fit(x_train,y_train,batch_size=200,epochs=20,verb
ose=1,validation_data=(x_test,y_test))
print('-------------------------------------------------------------')

score= model_3A.evaluate(x_test, y_test, verbose=0)
print('Test score: ',score[0])
print('Test accuracy: ',score[1])
print('-------------------------------------------------------------')

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')
x = list(range(1,21))
vy = history_3A.history['val_loss']
ty = history_3A.history['loss']
plt_dynamic(x, vy, ty, ax)
print('-------------------------------------------------------------')

w_after = model_3A.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)
```

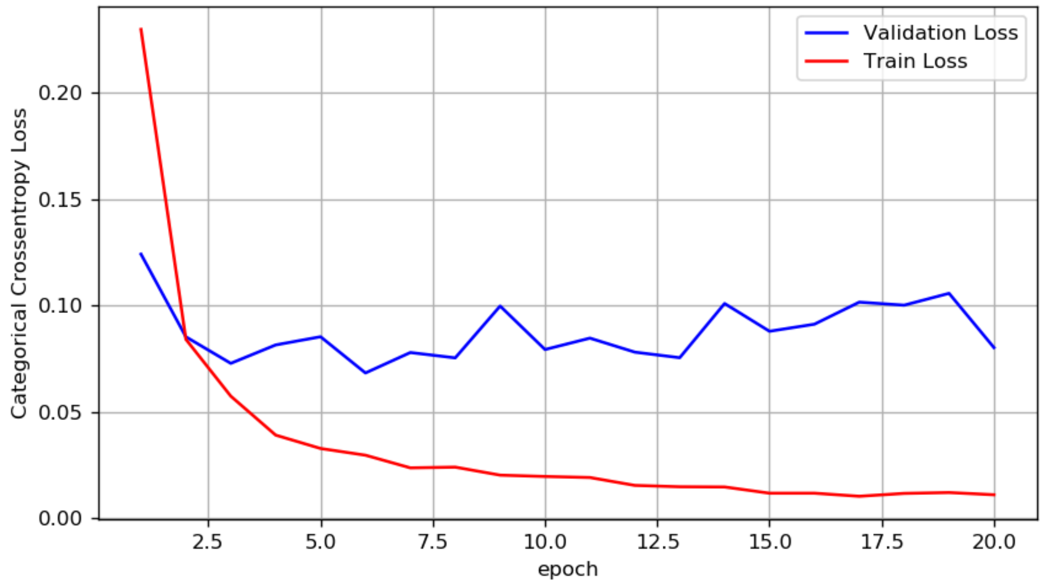
```
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

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()
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 18s 306us/step - loss:
0.2300 - acc: 0.9310 - val_loss: 0.1241 - val_acc: 0.9609
Epoch 2/20
60000/60000 [==============================] - 16s 271us/step - loss:
0.0840 - acc: 0.9739 - val_loss: 0.0852 - val_acc: 0.9724
Epoch 3/20
60000/60000 [==============================] - 16s 270us/step - loss:
0.0573 - acc: 0.9820 - val_loss: 0.0727 - val_acc: 0.9769
Epoch 4/20
60000/60000 [==============================] - 16s 269us/step - loss:
0.0389 - acc: 0.9876 - val_loss: 0.0814 - val_acc: 0.9749
Epoch 5/20
60000/60000 [==============================] - 16s 270us/step - loss:
```

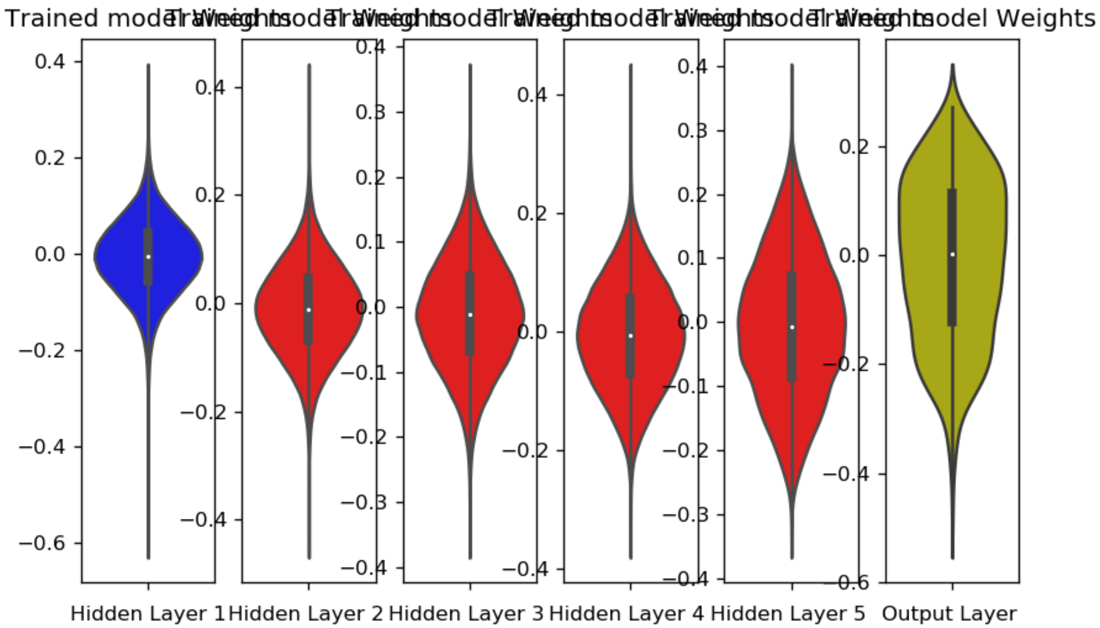
```
0.0327 - acc: 0.9891 - val_loss: 0.0853 - val_acc: 0.9746
Epoch 6/20
60000/60000 [==============================] - 16s 271us/step - loss:
0.0295 - acc: 0.9902 - val_loss: 0.0682 - val_acc: 0.9821
Epoch 7/20
60000/60000 [==============================] - 16s 271us/step - loss:
0.0235 - acc: 0.9926 - val_loss: 0.0778 - val_acc: 0.9798
Epoch 8/20
60000/60000 [==============================] - 16s 271us/step - loss:
0.0239 - acc: 0.9920 - val_loss: 0.0753 - val_acc: 0.9810
Epoch 9/20
60000/60000 [==============================] - 16s 273us/step - loss:
0.0201 - acc: 0.9938 - val_loss: 0.0997 - val_acc: 0.9755
Epoch 10/20
60000/60000 [==============================] - 16s 271us/step - loss:
0.0195 - acc: 0.9941 - val_loss: 0.0792 - val_acc: 0.9802196 - acc: 0.
Epoch 11/20
60000/60000 [==============================] - 16s 272us/step - loss:
0.0190 - acc: 0.9941 - val_loss: 0.0846 - val_acc: 0.9773
Epoch 12/20
60000/60000 [==============================] - 16s 270us/step - loss:
0.0153 - acc: 0.9952 - val_loss: 0.0780 - val_acc: 0.9802
Epoch 13/20
60000/60000 [==============================] - 16s 270us/step - loss:
0.0146 - acc: 0.9956 - val_loss: 0.0754 - val_acc: 0.9830
Epoch 14/20
60000/60000 [==============================] - 16s 271us/step - loss:
0.0145 - acc: 0.9956 - val_loss: 0.1009 - val_acc: 0.9779
Epoch 15/20
60000/60000 [==============================] - 16s 272us/step - loss:
0.0116 - acc: 0.9963 - val_loss: 0.0878 - val_acc: 0.9817
Epoch 16/20
60000/60000 [==============================] - 17s 275us/step - loss:
0.0116 - acc: 0.9962 - val_loss: 0.0912 - val_acc: 0.9807
Epoch 17/20
60000/60000 [==============================] - 16s 273us/step - loss:
0.0101 - acc: 0.9973 - val_loss: 0.1016 - val_acc: 0.9794
Epoch 18/20
60000/60000 [==============================] - 16s 270us/step - loss:
```

```
0.0115 - acc: 0.9967 - val_loss: 0.1001 - val_acc: 0.9783
Epoch 19/20
60000/60000 [==============================] - 16s 272us/step - loss:
0.0119 - acc: 0.9967 - val_loss: 0.1057 - val_acc: 0.9774
Epoch 20/20
60000/60000 [==============================] - 16s 273us/step - loss:
0.0109 - acc: 0.9970 - val_loss: 0.0802 - val_acc: 0.9835
------------------------------------------------------------
Test score:  0.08015283856078673
Test accuracy:  0.9835
------------------------------------------------------------
```

```
C:\Anaconda3\lib\site-packages\matplotlib\pyplot.py:537: RuntimeWarnin
g: More than 20 figures have been opened. Figures created through the p
yplot interface (`matplotlib.pyplot.figure`) are retained until explici
tly closed and may consume too much memory. (To control this warning, s
ee the rcParam `figure.max_open_warning`).
  max_open_warning, RuntimeWarning)
```




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



5 hidden layer with batch normalization

In [46]:
```
#model declaration and initialization
model_3B= Sequential()

model_3B.add(Dense(630, activation='relu', \
        input_shape= (784,), kernel_initializer=keras.initializers.he_n
ormal(seed=None)))
model_3B.add(BatchNormalization())

model_3B.add(Dense(480, activation='relu', \
        input_shape= (784,), kernel_initializer=keras.initializers.he_n
ormal(seed=None)))
model_3B.add(BatchNormalization())
```

```python
model_3B.add(Dense(330, activation='relu', \
        kernel_initializer= keras.initializers.he_normal(seed=None)))
model_3B.add(BatchNormalization())

model_3B.add(Dense(180, activation='relu', \
        input_shape= (784,), kernel_initializer=keras.initializers.he_n
ormal(seed=None)))
model_3B.add(BatchNormalization())

model_3B.add(Dense(80, activation='relu', \
        kernel_initializer= keras.initializers.he_normal(seed=None)))
model_3B.add(BatchNormalization())

model_3B.add(Dense(10, activation='softmax'))

print(model_3B.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_59 (Dense)             (None, 630)               494550
_________________________________________________________________
batch_normalization_21 (Batc (None, 630)               2520
_________________________________________________________________
dense_60 (Dense)             (None, 480)               302880
_________________________________________________________________
batch_normalization_22 (Batc (None, 480)               1920
_________________________________________________________________
dense_61 (Dense)             (None, 330)               158730
_________________________________________________________________
batch_normalization_23 (Batc (None, 330)               1320
_________________________________________________________________
dense_62 (Dense)             (None, 180)               59580
_________________________________________________________________
batch_normalization_24 (Batc (None, 180)               720
_________________________________________________________________
dense_63 (Dense)             (None, 80)                14480
_________________________________________________________________
batch_normalization_25 (Batc (None, 80)                320
```

```
dense_64 (Dense)             (None, 10)                810
=================================================================
Total params: 1,037,830
Trainable params: 1,034,430
Non-trainable params: 3,400
_________________________________________________________________
None
```

In [47]:
```
#run
model_3B.compile(optimizer='adam', loss='categorical_crossentropy', met
rics=['accuracy'])
history_3B = model_3B.fit(x_train,y_train,batch_size=200,epochs=20,verb
ose=1,validation_data=(x_test,y_test))
print('---------------------------------------------------------------')

score= model_3B.evaluate(x_test, y_test, verbose=0)
print('Test score: ',score[0])
print('Test accuracy: ',score[1])
print('---------------------------------------------------------------')

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')
x = list(range(1,21))
vy = history_3B.history['val_loss']
ty = history_3B.history['loss']
plt_dynamic(x, vy, ty, ax)
print('---------------------------------------------------------------')

w_after = model_3B.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)
```

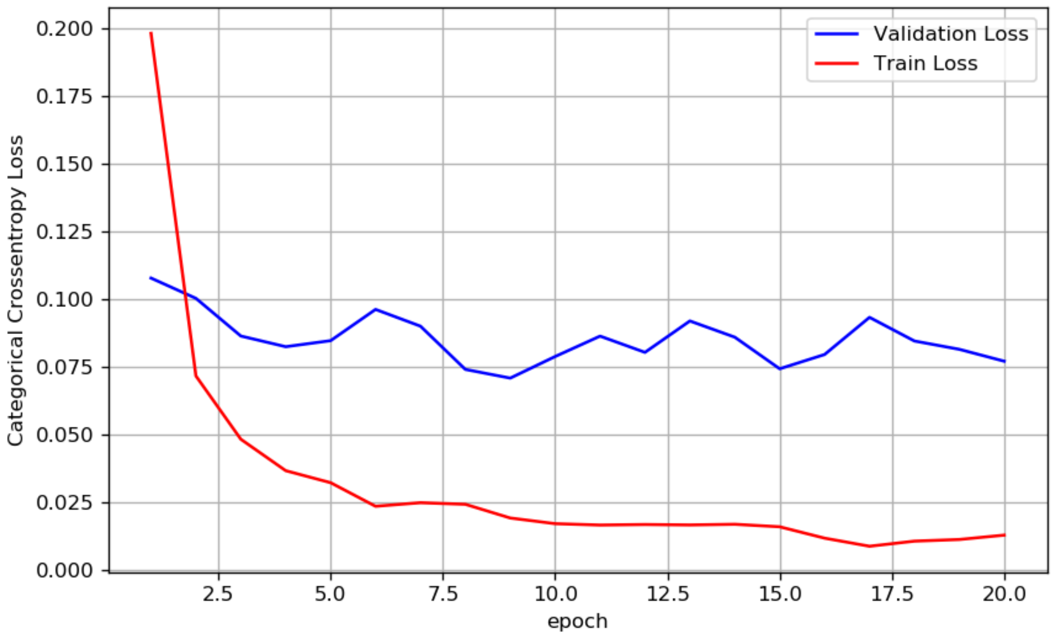
```
fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

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()
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 24s 395us/step - loss:
0.1981 - acc: 0.9403 - val_loss: 0.1077 - val_acc: 0.9668
Epoch 2/20
60000/60000 [==============================] - 21s 349us/step - loss:
0.0715 - acc: 0.9775 - val_loss: 0.1002 - val_acc: 0.9698
```

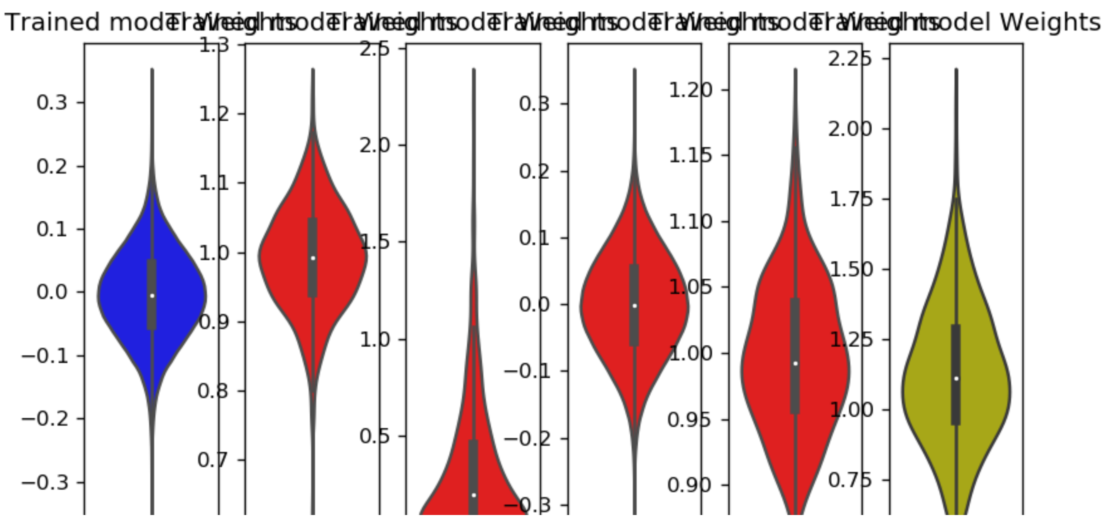
```
Epoch 3/20
60000/60000 [==============================] - 21s 344us/step - loss:
0.0482 - acc: 0.9843 - val_loss: 0.0863 - val_acc: 0.974977 - acc: 0.
Epoch 4/20
60000/60000 [==============================] - 21s 345us/step - loss:
0.0365 - acc: 0.9878 - val_loss: 0.0823 - val_acc: 0.9756
Epoch 5/20
60000/60000 [==============================] - 21s 344us/step - loss:
0.0321 - acc: 0.9897 - val_loss: 0.0846 - val_acc: 0.9760
Epoch 6/20
60000/60000 [==============================] - 21s 344us/step - loss:
0.0234 - acc: 0.9923 - val_loss: 0.0961 - val_acc: 0.9731
Epoch 7/20
60000/60000 [==============================] - 21s 343us/step - loss:
0.0247 - acc: 0.9915 - val_loss: 0.0899 - val_acc: 0.9760
Epoch 8/20
60000/60000 [==============================] - 21s 344us/step - loss:
0.0241 - acc: 0.9922 - val_loss: 0.0740 - val_acc: 0.9786
Epoch 9/20
60000/60000 [==============================] - 21s 344us/step - loss:
0.0191 - acc: 0.9937 - val_loss: 0.0707 - val_acc: 0.9801
Epoch 10/20
60000/60000 [==============================] - 20s 342us/step - loss:
0.0170 - acc: 0.9946 - val_loss: 0.0787 - val_acc: 0.9790
Epoch 11/20
60000/60000 [==============================] - 21s 343us/step - loss:
0.0165 - acc: 0.9946 - val_loss: 0.0863 - val_acc: 0.9774
Epoch 12/20
60000/60000 [==============================] - 21s 343us/step - loss:
0.0167 - acc: 0.9946 - val_loss: 0.0803 - val_acc: 0.9793
Epoch 13/20
60000/60000 [==============================] - 21s 343us/step - loss:
0.0165 - acc: 0.9949 - val_loss: 0.0918 - val_acc: 0.9774
Epoch 14/20
60000/60000 [==============================] - 21s 345us/step - loss:
0.0167 - acc: 0.9943 - val_loss: 0.0859 - val_acc: 0.9789
Epoch 15/20
60000/60000 [==============================] - 21s 344us/step - loss:

0.0158 - acc: 0.9947 - val_loss: 0.0742 - val_acc: 0.9795
```

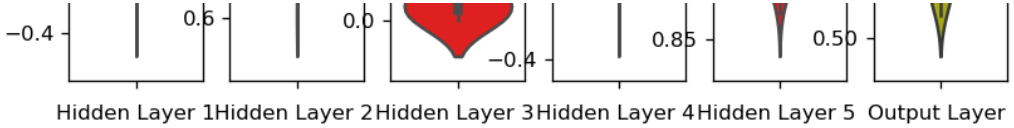
```
Epoch 16/20
60000/60000 [==============================] - 21s 343us/step - loss: 0.0116 - acc: 0.9964 - val_loss: 0.0795 - val_acc: 0.9801
Epoch 17/20
60000/60000 [==============================] - 21s 344us/step - loss: 0.0086 - acc: 0.9973 - val_loss: 0.0932 - val_acc: 0.9783
Epoch 18/20
60000/60000 [==============================] - 21s 343us/step - loss: 0.0105 - acc: 0.9968 - val_loss: 0.0845 - val_acc: 0.9798
Epoch 19/20
60000/60000 [==============================] - 21s 344us/step - loss: 0.0111 - acc: 0.9967 - val_loss: 0.0814 - val_acc: 0.9801
Epoch 20/20
60000/60000 [==============================] - 21s 345us/step - loss: 0.0127 - acc: 0.9960 - val_loss: 0.0770 - val_acc: 0.9802
------------------------------------------------------------
Test score:  0.07698586521920515
Test accuracy:  0.9802
------------------------------------------------------------
```

```
C:\Anaconda3\lib\site-packages\matplotlib\pyplot.py:537: RuntimeWarning: More than 20 figures have been opened. Figures created through the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume too much memory. (To control this warning, see the rcParam `figure.max_open_warning`).
  max_open_warning, RuntimeWarning)
```


Validation Loss
Train Loss
Categorical Crossentropy Loss
epoch
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

- -


Trained model Weights
Trained model Weights
Trained model Weights
Trained model Weights
Trained model Weights
Trained model Weights



5 hidden layer with dropouts

In [48]:
```
#model declaration and initialization
model_3C= Sequential()

model_3C.add(Dense(630, activation='relu', \
        input_shape= (784,), kernel_initializer=keras.initializers.he_n
ormal(seed=None)))
model_3C.add(Dropout(0.25))

model_3C.add(Dense(480, activation='relu', \
        input_shape= (784,), kernel_initializer=keras.initializers.he_n
ormal(seed=None)))
model_3C.add(Dropout(0.25))

model_3C.add(Dense(330, activation='relu', \
        input_shape= (784,), kernel_initializer=keras.initializers.he_n
ormal(seed=None)))
model_3C.add(Dropout(0.25))

model_3C.add(Dense(180, activation='relu', \
        input_shape= (784,), kernel_initializer=keras.initializers.he_n
ormal(seed=None)))
model_3C.add(Dropout(0.25))

model_3C.add(Dense(80, activation='relu', \
        kernel_initializer= keras.initializers.he_normal(seed=None)))
model_3C.add(Dropout(0.25))

model_3C.add(Dense(10, activation='softmax'))

print(model_3C.summary())
```

```
Layer (type)                 Output Shape              Param #   
=================================================================
dense_65 (Dense)             (None, 630)               494550    
_________________________________________________________________
dropout_21 (Dropout)         (None, 630)               0         
_________________________________________________________________
dense_66 (Dense)             (None, 480)               302880    
_________________________________________________________________
dropout_22 (Dropout)         (None, 480)               0         
_________________________________________________________________
dense_67 (Dense)             (None, 330)               158730    
_________________________________________________________________
dropout_23 (Dropout)         (None, 330)               0         
_________________________________________________________________
dense_68 (Dense)             (None, 180)               59580     
_________________________________________________________________
dropout_24 (Dropout)         (None, 180)               0         
_________________________________________________________________
dense_69 (Dense)             (None, 80)                14480     
_________________________________________________________________
dropout_25 (Dropout)         (None, 80)                0         
_________________________________________________________________
dense_70 (Dense)             (None, 10)                810       
=================================================================
Total params: 1,031,030
Trainable params: 1,031,030
Non-trainable params: 0
_________________________________________________________________
None
```

In [49]:

```
#run
model_3C.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history_3C = model_3C.fit(x_train,y_train,batch_size=200,epochs=20,verbose=1,validation_data=(x_test,y_test))
print('---------------------------------------------------------------')

score= model_3C.evaluate(x_test, y_test, verbose=0)
```

```
print('Test score: ',score[0])
print('Test accuracy: ',score[1])
print('-------------------------------------------------------------')

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')
x = list(range(1,21))
vy = history_3C.history['val_loss']
ty = history_3C.history['loss']
plt_dynamic(x, vy, ty, ax)
print('-------------------------------------------------------------')

w_after = model_3C.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 ')
```

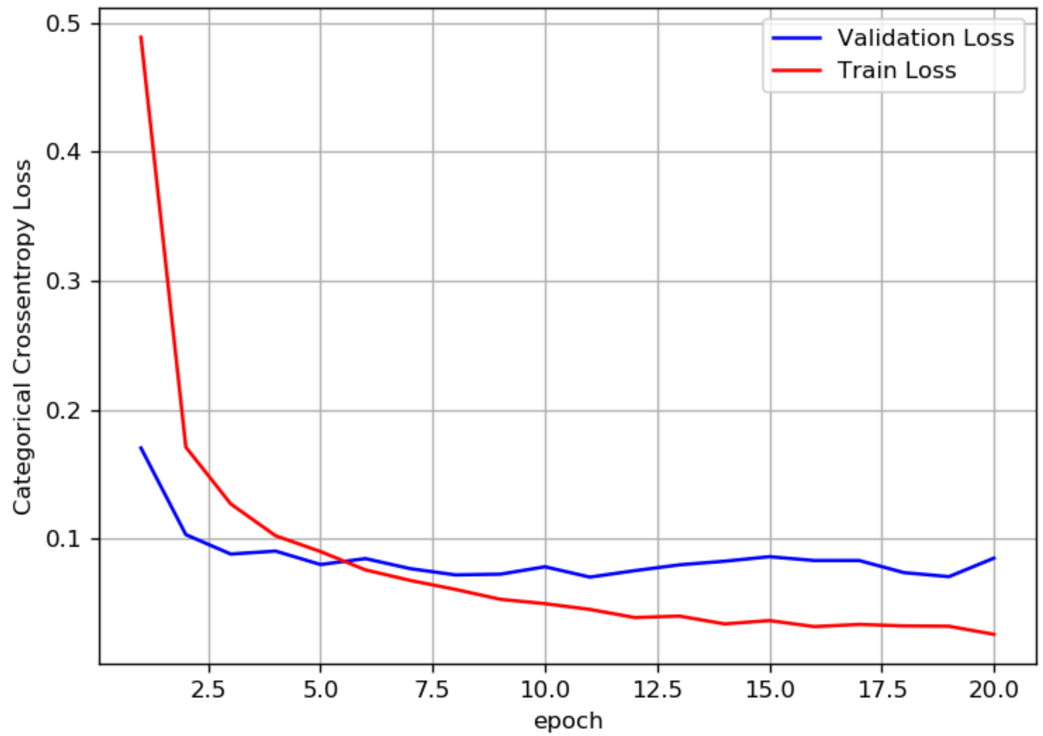
```
plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 4 ')

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()
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 21s 355us/step - loss:
0.4887 - acc: 0.8458 - val_loss: 0.1705 - val_acc: 0.9495
Epoch 2/20
60000/60000 [==============================] - 19s 311us/step - loss:
0.1712 - acc: 0.9533 - val_loss: 0.1034 - val_acc: 0.9708
Epoch 3/20
60000/60000 [==============================] - 19s 312us/step - loss:
0.1272 - acc: 0.9654 - val_loss: 0.0883 - val_acc: 0.9760
Epoch 4/20
60000/60000 [==============================] - 19s 312us/step - loss:
0.1025 - acc: 0.9719 - val_loss: 0.0906 - val_acc: 0.9762
Epoch 5/20
60000/60000 [==============================] - 19s 314us/step - loss:
0.0903 - acc: 0.9757 - val_loss: 0.0802 - val_acc: 0.9784
Epoch 6/20
60000/60000 [==============================] - 18s 308us/step - loss:
0.0761 - acc: 0.9793 - val_loss: 0.0848 - val_acc: 0.9777
Epoch 7/20
60000/60000 [==============================] - 19s 309us/step - loss:
0.0679 - acc: 0.9806 - val_loss: 0.0770 - val_acc: 0.9808
Epoch 8/20
60000/60000 [==============================] - 19s 311us/step - loss:
```

```
0.0610 - acc: 0.9834 - val_loss: 0.0722 - val_acc: 0.9803
Epoch 9/20
60000/60000 [==============================] - 19s 311us/step - loss:
0.0533 - acc: 0.9851 - val_loss: 0.0727 - val_acc: 0.9819
Epoch 10/20
60000/60000 [==============================] - 19s 312us/step - loss:
0.0499 - acc: 0.9859 - val_loss: 0.0785 - val_acc: 0.9814
Epoch 11/20
60000/60000 [==============================] - 19s 312us/step - loss:
0.0455 - acc: 0.9873 - val_loss: 0.0705 - val_acc: 0.9833
Epoch 12/20
60000/60000 [==============================] - 19s 313us/step - loss:
0.0392 - acc: 0.9885 - val_loss: 0.0755 - val_acc: 0.9824
Epoch 13/20
60000/60000 [==============================] - 19s 311us/step - loss:
0.0403 - acc: 0.9889 - val_loss: 0.0800 - val_acc: 0.9833
Epoch 14/20
60000/60000 [==============================] - 19s 312us/step - loss:
0.0342 - acc: 0.9900 - val_loss: 0.0828 - val_acc: 0.9807
Epoch 15/20
60000/60000 [==============================] - 19s 312us/step - loss:
0.0368 - acc: 0.9899 - val_loss: 0.0863 - val_acc: 0.9826
Epoch 16/20
60000/60000 [==============================] - 19s 310us/step - loss:
0.0321 - acc: 0.9904 - val_loss: 0.0833 - val_acc: 0.9797
Epoch 17/20
60000/60000 [==============================] - 19s 310us/step - loss:
0.0338 - acc: 0.9906 - val_loss: 0.0833 - val_acc: 0.9826
Epoch 18/20
60000/60000 [==============================] - 19s 313us/step - loss:
0.0327 - acc: 0.9907 - val_loss: 0.0740 - val_acc: 0.9827
Epoch 19/20
60000/60000 [==============================] - 19s 310us/step - loss:
0.0324 - acc: 0.9908 - val_loss: 0.0708 - val_acc: 0.9855
Epoch 20/20
60000/60000 [==============================] - 19s 310us/step - loss:
0.0262 - acc: 0.9925 - val_loss: 0.0852 - val_acc: 0.9829
------------------------------------------------------------
Test score:  0.08516765279469246
```

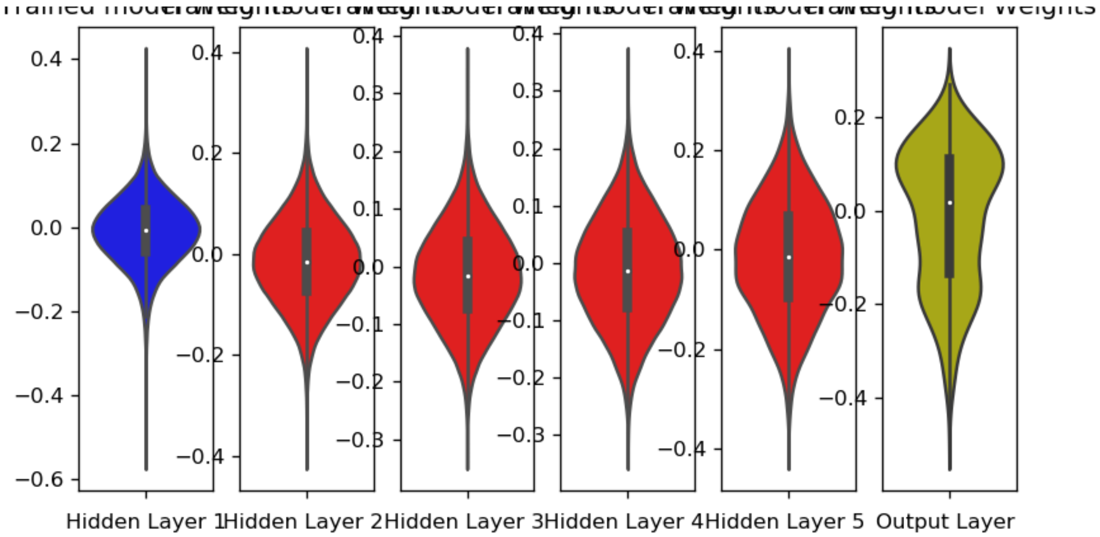
```
Test accuracy:  0.9829
------------------------------------------------------------
```

```
C:\Anaconda3\lib\site-packages\matplotlib\pyplot.py:537: RuntimeWarnin
g: More than 20 figures have been opened. Figures created through the p
yplot interface (`matplotlib.pyplot.figure`) are retained until explici
tly closed and may consume too much memory. (To control this warning, s
ee the rcParam `figure.max_open_warning`).
  max_open_warning, RuntimeWarning)
```




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

Trained model Weights



5 hidden layer with dropouts and batch normalization

In [50]:
```
#model declaration and initialization
model_3D= Sequential()

model_3D.add(Dense(630, activation='relu', \
        input_shape= (784,), kernel_initializer=keras.initializers.he_n
ormal(seed=None)))
model_3D.add(BatchNormalization())
model_3D.add(Dropout(0.25))

model_3D.add(Dense(480, activation='relu', \
        input_shape= (784,), kernel_initializer=keras.initializers.he_n
ormal(seed=None)))
model_3D.add(BatchNormalization())
model_3D.add(Dropout(0.25))

model_3D.add(Dense(330, activation='relu', \
        input_shape= (784,), kernel_initializer=keras.initializers.he_n
ormal(seed=None)))
```

```
model_3D.add(BatchNormalization())
model_3D.add(Dropout(0.25))

model_3D.add(Dense(180, activation='relu', \
        input_shape= (784,), kernel_initializer=keras.initializers.he_n
ormal(seed=None)))
model_3D.add(BatchNormalization())
model_3D.add(Dropout(0.25))

model_3D.add(Dense(80, activation='relu', \
        kernel_initializer= keras.initializers.he_normal(seed=None)))
model_3D.add(BatchNormalization())
model_3D.add(Dropout(0.25))

model_3D.add(Dense(10, activation='softmax'))

print(model_3D.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_71 (Dense)             (None, 630)               494550    
_________________________________________________________________
batch_normalization_26 (Batc (None, 630)               2520      
_________________________________________________________________
dropout_26 (Dropout)         (None, 630)               0         
_________________________________________________________________
dense_72 (Dense)             (None, 480)               302880    
_________________________________________________________________
batch_normalization_27 (Batc (None, 480)               1920      
_________________________________________________________________
dropout_27 (Dropout)         (None, 480)               0         
_________________________________________________________________
dense_73 (Dense)             (None, 330)               158730    
_________________________________________________________________
batch_normalization_28 (Batc (None, 330)               1320      
_________________________________________________________________
dropout_28 (Dropout)         (None, 330)               0         
_________________________________________________________________
dense_74 (Dense)             (None, 180)               59580     
```

```
_________________________________________________________________
batch_normalization_29 (Batc (None, 180)               720       
_________________________________________________________________
dropout_29 (Dropout)         (None, 180)               0         
_________________________________________________________________
dense_75 (Dense)             (None, 80)                14480     
_________________________________________________________________
batch_normalization_30 (Batc (None, 80)                320       
_________________________________________________________________
dropout_30 (Dropout)         (None, 80)                0         
_________________________________________________________________
dense_76 (Dense)             (None, 10)                810       
=================================================================
Total params: 1,037,830
Trainable params: 1,034,430
Non-trainable params: 3,400
_________________________________________________________________
None
```

In [51]:

```
#run
model_3D.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history_3D = model_3D.fit(x_train,y_train,batch_size=200,epochs=20,verbose=1,validation_data=(x_test,y_test))
print('-------------------------------------------------------------')

score= model_3D.evaluate(x_test, y_test, verbose=0)
print('Test score: ',score[0])
print('Test accuracy: ',score[1])
print('-------------------------------------------------------------')

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')
x = list(range(1,21))
vy = history_3D.history['val_loss']
ty = history_3D.history['loss']
plt_dynamic(x, vy, ty, ax)
print('-------------------------------------------------------------')
```

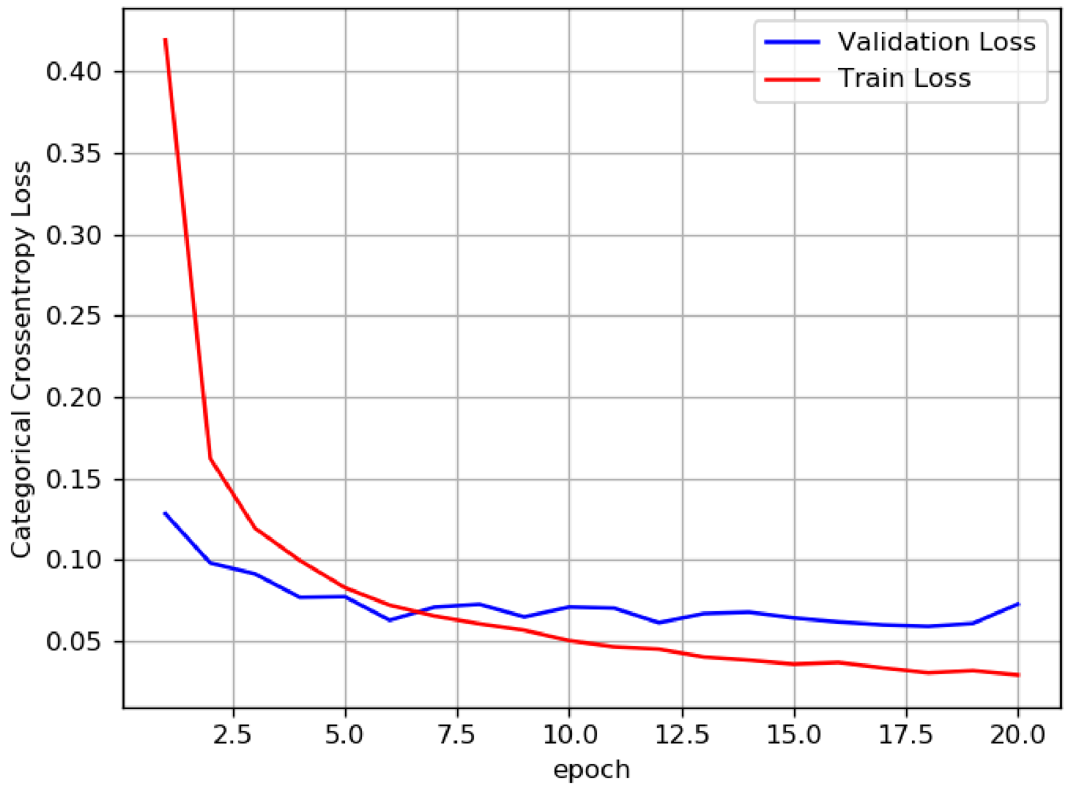
```
w_after = model_3D.get_weights()

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)
```

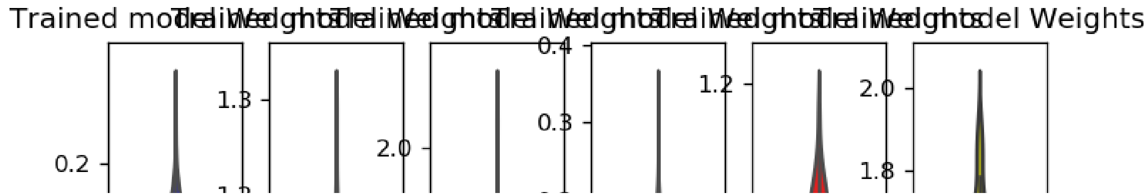
```
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```

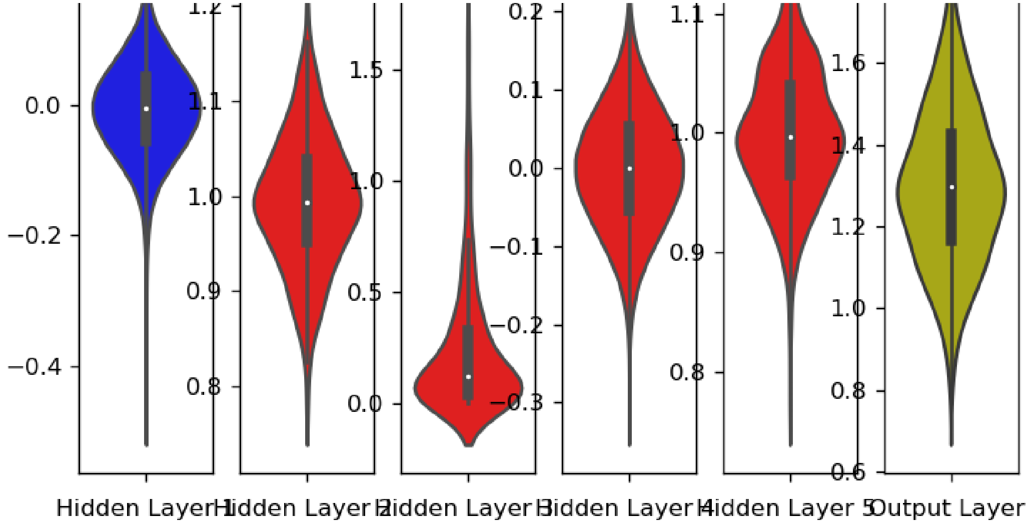
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 27s 451us/step - loss:
0.4195 - acc: 0.8724 - val_loss: 0.1282 - val_acc: 0.9620
Epoch 2/20
60000/60000 [==============================] - 23s 389us/step - loss:
0.1621 - acc: 0.9516 - val_loss: 0.0979 - val_acc: 0.9711
Epoch 3/20
60000/60000 [==============================] - 23s 390us/step - loss:
0.1193 - acc: 0.9635 - val_loss: 0.0911 - val_acc: 0.9730
Epoch 4/20
60000/60000 [==============================] - 23s 389us/step - loss:
0.0994 - acc: 0.9699 - val_loss: 0.0767 - val_acc: 0.9781
Epoch 5/20
60000/60000 [==============================] - 23s 388us/step - loss:
0.0828 - acc: 0.9750 - val_loss: 0.0772 - val_acc: 0.9776
Epoch 6/20
60000/60000 [==============================] - 23s 390us/step - loss:
0.0719 - acc: 0.9782 - val_loss: 0.0627 - val_acc: 0.9813
Epoch 7/20
60000/60000 [==============================] - 23s 388us/step - loss:
0.0653 - acc: 0.9803 - val_loss: 0.0707 - val_acc: 0.9799
Epoch 8/20
60000/60000 [==============================] - 23s 390us/step - loss:
0.0605 - acc: 0.9810 - val_loss: 0.0725 - val_acc: 0.9794
Epoch 9/20
60000/60000 [==============================] - 23s 390us/step - loss:
0.0565 - acc: 0.9828 - val_loss: 0.0647 - val_acc: 0.9812
Epoch 10/20
60000/60000 [==============================] - 23s 392us/step - loss:
0.0501 - acc: 0.9842 - val_loss: 0.0708 - val_acc: 0.9796
Epoch 11/20
60000/60000 [==============================] - 23s 390us/step - loss:
0.0463 - acc: 0.9861 - val_loss: 0.0701 - val_acc: 0.9811
Epoch 12/20
```

```
60000/60000 [==============================] - 23s 390us/step - loss:
0.0449 - acc: 0.9863 - val_loss: 0.0612 - val_acc: 0.9830
Epoch 13/20
60000/60000 [==============================] - 23s 391us/step - loss:
0.0401 - acc: 0.9879 - val_loss: 0.0667 - val_acc: 0.9818
Epoch 14/20
60000/60000 [==============================] - 23s 388us/step - loss:
0.0381 - acc: 0.9885 - val_loss: 0.0676 - val_acc: 0.9807
Epoch 15/20
60000/60000 [==============================] - 23s 389us/step - loss:
0.0357 - acc: 0.9890 - val_loss: 0.0641 - val_acc: 0.9832
Epoch 16/20
60000/60000 [==============================] - 24s 392us/step - loss:
0.0367 - acc: 0.9885 - val_loss: 0.0616 - val_acc: 0.9828
Epoch 17/20
60000/60000 [==============================] - 24s 394us/step - loss:
0.0333 - acc: 0.9897 - val_loss: 0.0599 - val_acc: 0.9839
Epoch 18/20
60000/60000 [==============================] - 24s 394us/step - loss:
0.0303 - acc: 0.9911 - val_loss: 0.0589 - val_acc: 0.9840
Epoch 19/20
60000/60000 [==============================] - 23s 390us/step - loss:
0.0317 - acc: 0.9902 - val_loss: 0.0607 - val_acc: 0.9827
Epoch 20/20
60000/60000 [==============================] - 24s 392us/step - loss:
0.0290 - acc: 0.9907 - val_loss: 0.0726 - val_acc: 0.9829
------------------------------------------------------------
Test score:  0.07257580204863334
Test accuracy:  0.9829
------------------------------------------------------------
```

```
C:\Anaconda3\lib\site-packages\matplotlib\pyplot.py:537: RuntimeWarnin
g: More than 20 figures have been opened. Figures created through the p
yplot interface (`matplotlib.pyplot.figure`) are retained until explici
tly closed and may consume too much memory. (To control this warning, s
ee the rcParam `figure.max_open_warning`).
  max_open_warning, RuntimeWarning)
```


Validation Loss
Train Loss
Categorical Crossentropy Loss
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
epoch


Trained model Weights
0.2
1.3
2.0
0.4
0.3
1.2
2.0
1.8



Summary

Model	Hidden Layers	simple	with batchNorm	with Dropout	with batchNorm and Dropout
1	2	0.9784	0.9806	0.9838	0.9822
2	3	0.9818	0.9824	0.9836	0.9838
3	5	0.9835	0.9802	0.9829	0.9829

- Model with 3 hidden layered structure performed well in our case taking batchnormalization and dropout with accuracy of 98.38
- Dropout and BatchNormalization had good impact on accuracy and time to train the model
- So performance was incresed by adding batchNormalization and Dropout layer