Apply different Architectures on MNIST dataset using Keras

In [19]:

```
import warnings
warnings.filterwarnings("ignore")
# if you keras is not using tensorflow as backend set "KERAS_BACKEND=tensorflow" use th
is command
from keras.utils import np_utils
from keras.datasets import mnist
import seaborn as sns
from keras.initializers import RandomNormal
#from keras.utils.visualize_util import to_graph
from keras.models import Sequential
#to_graph(Sequential())
```

READING DATA

In [4]:

```
%matplotlib notebook
%matplotlib inline
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 [5]:

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

```
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 164s 14us/step
```

In [6]:

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

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

In [7]:

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

In [8]:

```
# after converting the input images from 3d to 2d vectors
print("Number of training examples :", X_train.shape[0], "and each image is of shape(%d
)"%(X_train.shape[1]))
print("Number of training examples :", X_test.shape[0], "and each image is of shape (%d
)"%(X_test.shape[1]))
```

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

In [9]:

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

In [10]:

```
# here we are having a class number for each image
print("Class label of first image :", y_train[0])
# lets convert this into a 10 dimensional vector
# ex: consider an image is 5 convert it into 5 => [0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
# this conversion needed for MLPs
Y_train = np_utils.to_categorical(y_train, 10)
Y_test = np_utils.to_categorical(y_test, 10)

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

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

In [11]:

```
# some model parameters
output_dim = 10
input_dim = X_train.shape[1]

batch_size = 112
nb_epoch = 20
print(input_dim)
```

```
784
```

Model 1 -- with 2 Hidden layers

1. MLP + ReLU + adam

In [22]:

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

In [23]:

```
# Multilayer perceptron
# https://arxiv.org/pdf/1707.09725.pdf#page=95
# for relu layers
# If we sample weights from a normal distribution N(0,σ) we satisfy this condition with σ=√(2/(ni).
# h1 => σ=√(2/(fan_in) = 0.062 => N(0,σ) = N(0,0.062)
# h2 => σ=√(2/(fan_in) = 0.125 => N(0,σ) = N(0,0.125)
# out => σ=√(2/(fan_in+1) = 0.120 => N(0,σ) = N(0,0.120)

model_relu = Sequential()
model_relu.add(Dense(610, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_relu.add(Dense(325, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))
model_relu.summary()
```

```
WARNING:tensorflow:From C:\anaconda\lib\site-packages\tensorflow\python\framework\op_def_library.py:263: colocate_with (from tensorflow.python.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, 610)               478850
_________________________________________________________________
dense_2 (Dense)              (None, 325)               198575
_________________________________________________________________
dense_3 (Dense)              (None, 10)                3260
=================================================================
Total params: 680,685
Trainable params: 680,685
Non-trainable params: 0
_________________________________________________________________
```

In [24]:

```
model_relu.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accurac
y'])
history = model_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verb
ose=1, validation_data=(X_test, Y_test))
```

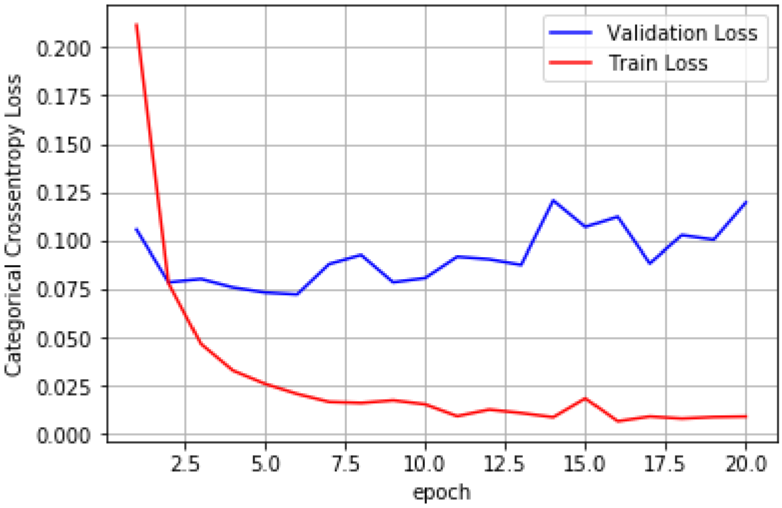
```
WARNING:tensorflow:From C:\anaconda\lib\site-packages\tensorflow\python\op
s\math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is depr
ecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 19s 322us/step - loss: 0.21
13 - acc: 0.9366 - val_loss: 0.1056 - val_acc: 0.9668
Epoch 2/20
60000/60000 [==============================] - 17s 275us/step - loss: 0.07
81 - acc: 0.9759 - val_loss: 0.0785 - val_acc: 0.9743
Epoch 3/20
60000/60000 [==============================] - 17s 276us/step - loss: 0.04
68 - acc: 0.9848 - val_loss: 0.0803 - val_acc: 0.9747
Epoch 4/20
60000/60000 [==============================] - 16s 266us/step - loss: 0.03
30 - acc: 0.9896 - val_loss: 0.0759 - val_acc: 0.9763
Epoch 5/20
60000/60000 [==============================] - 16s 265us/step - loss: 0.02
61 - acc: 0.9914 - val_loss: 0.0733 - val_acc: 0.9788
Epoch 6/20
60000/60000 [==============================] - 16s 274us/step - loss: 0.02
09 - acc: 0.9931 - val_loss: 0.0723 - val_acc: 0.9796
Epoch 7/20
60000/60000 [==============================] - 17s 275us/step - loss: 0.01
68 - acc: 0.9947 - val_loss: 0.0880 - val_acc: 0.9760
Epoch 8/20
60000/60000 [==============================] - 13s 215us/step - loss: 0.01
62 - acc: 0.9950 - val_loss: 0.0927 - val_acc: 0.9786
Epoch 9/20
60000/60000 [==============================] - 13s 218us/step - loss: 0.01
76 - acc: 0.9943 - val_loss: 0.0786 - val_acc: 0.9801
Epoch 10/20
60000/60000 [==============================] - 13s 214us/step - loss: 0.01
55 - acc: 0.9950 - val_loss: 0.0808 - val_acc: 0.9819
Epoch 11/20
60000/60000 [==============================] - 13s 212us/step - loss: 0.00
94 - acc: 0.9969 - val_loss: 0.0918 - val_acc: 0.9777
Epoch 12/20
60000/60000 [==============================] - 13s 215us/step - loss: 0.01
28 - acc: 0.9955 - val_loss: 0.0905 - val_acc: 0.9800
Epoch 13/20
60000/60000 [==============================] - 13s 213us/step - loss: 0.01
10 - acc: 0.9965 - val_loss: 0.0876 - val_acc: 0.9811
Epoch 14/20
60000/60000 [==============================] - 13s 210us/step - loss: 0.00
89 - acc: 0.9971 - val_loss: 0.1207 - val_acc: 0.9748
Epoch 15/20
60000/60000 [==============================] - 13s 212us/step - loss: 0.01
86 - acc: 0.9940 - val_loss: 0.1070 - val_acc: 0.9767
Epoch 16/20
60000/60000 [==============================] - 13s 212us/step - loss: 0.00
68 - acc: 0.9979 - val_loss: 0.1123 - val_acc: 0.9796
Epoch 17/20
60000/60000 [==============================] - 13s 215us/step - loss: 0.00
93 - acc: 0.9971 - val_loss: 0.0883 - val_acc: 0.9813
Epoch 18/20
60000/60000 [==============================] - 13s 213us/step - loss: 0.00
82 - acc: 0.9974 - val_loss: 0.1028 - val_acc: 0.9804
Epoch 19/20
```

```
60000/60000 [==============================] - 13s 209us/step - loss: 0.00
90 - acc: 0.9973 - val_loss: 0.1004 - val_acc: 0.9804
Epoch 20/20
60000/60000 [==============================] - 13s 209us/step - loss: 0.00
92 - acc: 0.9971 - val_loss: 0.1196 - val_acc: 0.9782
```

In [25]:

```
#Evaluate your model with accuracy and plot of (NUmber of epoches VS train_and_val_los
s)
#Train accuracy
score = model_relu.evaluate(X_train, Y_train, verbose=0)
print('Train score:', score[0])
print('Train accuracy:', score[1]*100)
print('\n************************ *********************\n')
#test accuracy
score = model_relu.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1]*100)

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')
# list of epoch numbers
x = list(range(1,nb_epoch+1))
# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ve
rbose=1, validation_data=(X_test, Y_test))
# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy
# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of ep
ochs
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Train score: 0.01175171572082836
Train accuracy: 99.61166666666666

*********************** ********************

Test score: 0.11959466045504177
Test accuracy: 97.82
```

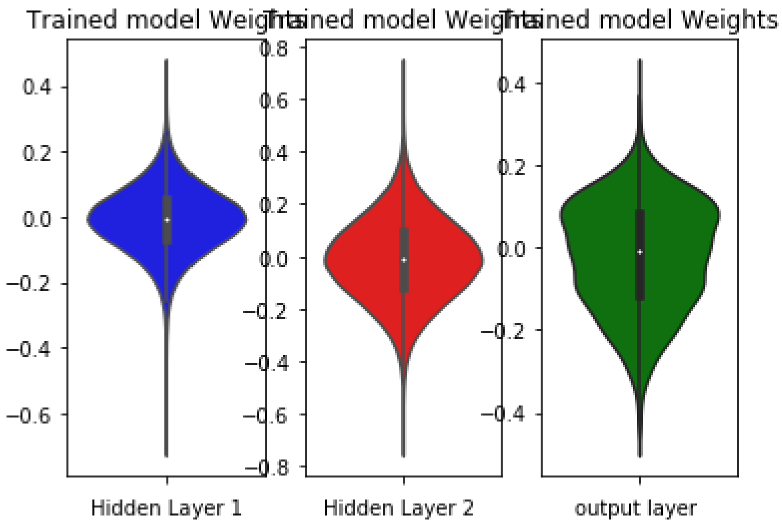


In [26]:

```
# Weights after trainning
# 1 2 3
# input->h1->h2->output
w_after = model_relu.get_weights()
# if 2 hidden layer then
# w_after[0]is the inpupt layer weights w_after[1]is the input layer bias weights input to hidden1
# w_after[2]is the hidde layer weights w_after[3]is the hidde layer bias weights hidden 1to hidden2
# w_after[4]is the hidde layer weights w_after[5]is the hidde layer bias weights hidden 2 to output
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='g')
plt.xlabel('output layer ')
```

Out[26]:

```
Text(0.5, 0, 'output layer ')
```



2. MLP + ReLU + adam + batch__normalization

In [27]:

```
from keras.layers.normalization import BatchNormalization
model_batch = Sequential()
model_batch.add(Dense(610, activation='relu', input_shape=(input_dim,), kernel_initiali
zer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_batch.add(BatchNormalization())
model_batch.add(Dense(325, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
stddev=0.55, seed=None)) )
model_batch.add(BatchNormalization())
model_batch.add(Dense(output_dim, activation='softmax'))
model_batch.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_4 (Dense)              (None, 610)               478850    
_________________________________________________________________
batch_normalization_1 (Batch (None, 610)               2440      
_________________________________________________________________
dense_5 (Dense)              (None, 325)               198575    
_________________________________________________________________
batch_normalization_2 (Batch (None, 325)               1300      
_________________________________________________________________
dense_6 (Dense)              (None, 10)                3260      
=================================================================
Total params: 684,425
Trainable params: 682,555
Non-trainable params: 1,870
_________________________________________________________________
```

In [28]:

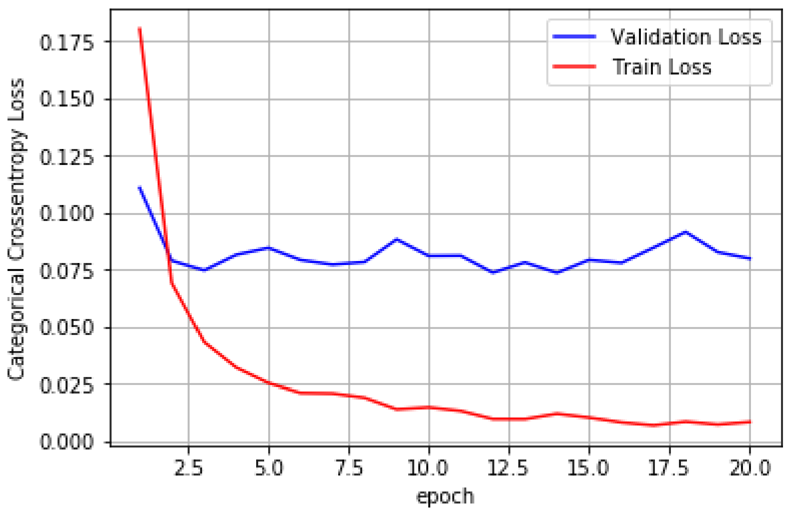
```
model_batch.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura
cy'])
history = model_batch.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ver
bose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 19s 314us/step - loss: 0.18
04 - acc: 0.9459 - val_loss: 0.1109 - val_acc: 0.9638
Epoch 2/20
60000/60000 [==============================] - 15s 256us/step - loss: 0.06
91 - acc: 0.9785 - val_loss: 0.0789 - val_acc: 0.9747
Epoch 3/20
60000/60000 [==============================] - 15s 258us/step - loss: 0.04
32 - acc: 0.9864 - val_loss: 0.0747 - val_acc: 0.9762
Epoch 4/20
60000/60000 [==============================] - 17s 285us/step - loss: 0.03
20 - acc: 0.9902 - val_loss: 0.0815 - val_acc: 0.9748
Epoch 5/20
60000/60000 [==============================] - 15s 249us/step - loss: 0.02
53 - acc: 0.9920 - val_loss: 0.0845 - val_acc: 0.9747
Epoch 6/20
60000/60000 [==============================] - 15s 249us/step - loss: 0.02
10 - acc: 0.9934 - val_loss: 0.0792 - val_acc: 0.9781
Epoch 7/20
60000/60000 [==============================] - 15s 254us/step - loss: 0.02
08 - acc: 0.9934 - val_loss: 0.0772 - val_acc: 0.9770
Epoch 8/20
60000/60000 [==============================] - 15s 251us/step - loss: 0.01
89 - acc: 0.9940 - val_loss: 0.0783 - val_acc: 0.9781
Epoch 9/20
60000/60000 [==============================] - 15s 247us/step - loss: 0.01
38 - acc: 0.9950 - val_loss: 0.0883 - val_acc: 0.9753
Epoch 10/20
60000/60000 [==============================] - 15s 248us/step - loss: 0.01
47 - acc: 0.9951 - val_loss: 0.0810 - val_acc: 0.9782
Epoch 11/20
60000/60000 [==============================] - 15s 250us/step - loss: 0.01
31 - acc: 0.9956 - val_loss: 0.0811 - val_acc: 0.9794
Epoch 12/20
60000/60000 [==============================] - 17s 275us/step - loss: 0.00
96 - acc: 0.9968 - val_loss: 0.0736 - val_acc: 0.9798
Epoch 13/20
60000/60000 [==============================] - 15s 250us/step - loss: 0.00
96 - acc: 0.9970 - val_loss: 0.0781 - val_acc: 0.9798
Epoch 14/20
60000/60000 [==============================] - 15s 250us/step - loss: 0.01
19 - acc: 0.9960 - val_loss: 0.0736 - val_acc: 0.9812
Epoch 15/20
60000/60000 [==============================] - 15s 255us/step - loss: 0.01
02 - acc: 0.9965 - val_loss: 0.0792 - val_acc: 0.9797
Epoch 16/20
60000/60000 [==============================] - 15s 249us/step - loss: 0.00
81 - acc: 0.9974 - val_loss: 0.0779 - val_acc: 0.9791
Epoch 17/20
60000/60000 [==============================] - 15s 250us/step - loss: 0.00
68 - acc: 0.9979 - val_loss: 0.0846 - val_acc: 0.9803
Epoch 18/20
60000/60000 [==============================] - 15s 246us/step - loss: 0.00
84 - acc: 0.9972 - val_loss: 0.0915 - val_acc: 0.9782
Epoch 19/20
60000/60000 [==============================] - 15s 249us/step - loss: 0.00
72 - acc: 0.9977 - val_loss: 0.0827 - val_acc: 0.9816
Epoch 20/20
60000/60000 [==============================] - 15s 246us/step - loss: 0.00
83 - acc: 0.9971 - val_loss: 0.0799 - val_acc: 0.9820
```

In [30]:

```
#Evaluate your model with accuracy and plot of (NUmber of epoches VS train_and_val_los
s)
#Train accuracy
score = model_batch.evaluate(X_train, Y_train, verbose=0)
print('Train score:', score[0])
print('Train accuracy:', score[1]*100)
print('\n*********************** ********************\n')
#test accuracy
score = model_batch.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1]*100)
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')
# list of epoch numbers
x = list(range(1,nb_epoch+1))
# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ve
rbose=1, validation_data=(X_test, Y_test))
# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy
# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of ep
ochs
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Train score: 0.0036005233980133805
Train accuracy: 99.88333333333334

*********************** ********************

Test score: 0.07985942602099326
Test accuracy: 98.2
```

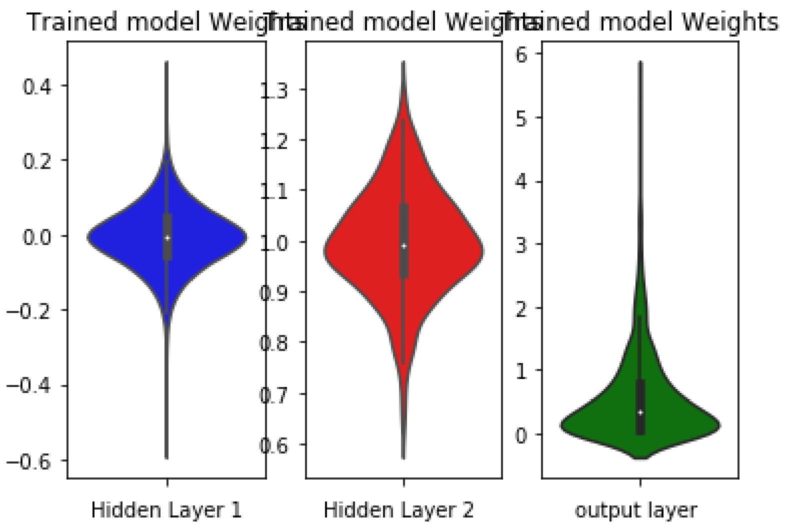


In [31]:

```
# Weights after trainning
# 1 2 3
# input->h1->h2->output
w_after = model_batch.get_weights()
# if 2 hidden layer then
# w_after[0]is the inpupt layer weights w_after[1]is the input layer bias weights input to hidden1
# w_after[2]is the hidde layer weights w_after[3]is the hidde layer bias weights hidden 1 to hidden2
# w_after[4]is the hidde layer weights w_after[5]is the hidde layer bias weights hidden 2 to output
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='g')
plt.xlabel('output layer ')
```

Out[31]:

```
Text(0.5, 0, 'output layer ')
```



3. MLP + ReLU + adam + dropout

In [32]:

```
# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormalization-function-inkeras
from keras.layers import Dropout
model_drop = Sequential()
model_drop.add(Dense(610, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
#model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))
model_drop.add(Dense(325, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.55,seed=None)) )
#model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))
model_drop.add(Dense(output_dim, activation='softmax'))
model_drop.summary()
```

```
WARNING:tensorflow:From C:\anaconda\lib\site-packages\keras\backend\tensor
flow_backend.py:3445: calling dropout (from tensorflow.python.ops.nn_ops)
with keep_prob is deprecated and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1
- keep_prob`.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (None, 610)               478850
_________________________________________________________________
dropout_1 (Dropout)          (None, 610)               0
_________________________________________________________________
dense_8 (Dense)              (None, 325)               198575
_________________________________________________________________
dropout_2 (Dropout)          (None, 325)               0
_________________________________________________________________
dense_9 (Dense)              (None, 10)                3260
=================================================================
Total params: 680,685
Trainable params: 680,685
Non-trainable params: 0
_________________________________________________________________
```

In [33]:

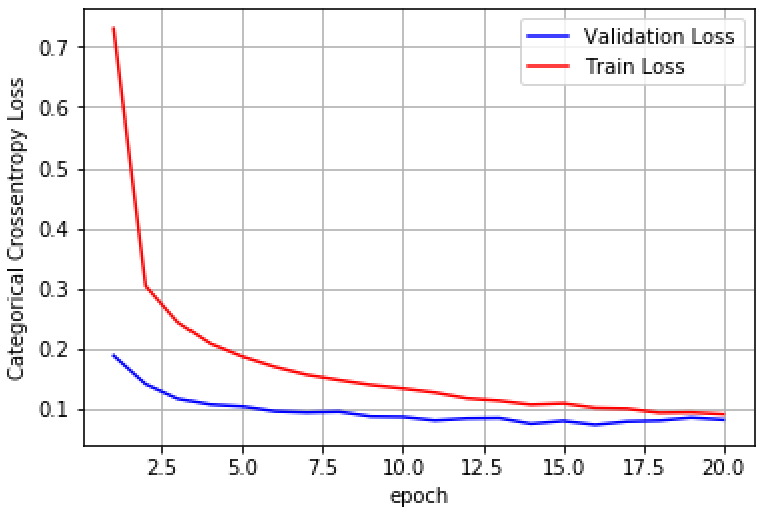
```
model_drop.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accurac
y'])
history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=20, verbose=1,
validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 17s 285us/step - loss: 0.73
05 - acc: 0.8101 - val_loss: 0.1879 - val_acc: 0.9447
Epoch 2/20
60000/60000 [==============================] - 14s 232us/step - loss: 0.30
38 - acc: 0.9110 - val_loss: 0.1403 - val_acc: 0.9594
Epoch 3/20
60000/60000 [==============================] - 14s 231us/step - loss: 0.24
30 - acc: 0.9283 - val_loss: 0.1160 - val_acc: 0.9659
Epoch 4/20
60000/60000 [==============================] - 14s 231us/step - loss: 0.20
77 - acc: 0.9398 - val_loss: 0.1064 - val_acc: 0.9686
Epoch 5/20
60000/60000 [==============================] - 14s 233us/step - loss: 0.18
61 - acc: 0.9460 - val_loss: 0.1031 - val_acc: 0.9724- lo
Epoch 6/20
60000/60000 [==============================] - 14s 233us/step - loss: 0.16
91 - acc: 0.9505 - val_loss: 0.0952 - val_acc: 0.9739
Epoch 7/20
60000/60000 [==============================] - 15s 250us/step - loss: 0.15
59 - acc: 0.9536 - val_loss: 0.0932 - val_acc: 0.9730
Epoch 8/20
60000/60000 [==============================] - 14s 241us/step - loss: 0.14
71 - acc: 0.9572 - val_loss: 0.0946 - val_acc: 0.9733
Epoch 9/20
60000/60000 [==============================] - 14s 234us/step - loss: 0.13
90 - acc: 0.9604 - val_loss: 0.0866 - val_acc: 0.9744
Epoch 10/20
60000/60000 [==============================] - 14s 235us/step - loss: 0.13
31 - acc: 0.9607 - val_loss: 0.0858 - val_acc: 0.9766
Epoch 11/20
60000/60000 [==============================] - 14s 235us/step - loss: 0.12
64 - acc: 0.9628 - val_loss: 0.0799 - val_acc: 0.9775
Epoch 12/20
60000/60000 [==============================] - 14s 235us/step - loss: 0.11
67 - acc: 0.9654 - val_loss: 0.0833 - val_acc: 0.9773
Epoch 13/20
60000/60000 [==============================] - 14s 239us/step - loss: 0.11
27 - acc: 0.9673 - val_loss: 0.0839 - val_acc: 0.9748
Epoch 14/20
60000/60000 [==============================] - 15s 245us/step - loss: 0.10
62 - acc: 0.9680 - val_loss: 0.0743 - val_acc: 0.9793
Epoch 15/20
60000/60000 [==============================] - 14s 237us/step - loss: 0.10
84 - acc: 0.9691 - val_loss: 0.0792 - val_acc: 0.9769
Epoch 16/20
60000/60000 [==============================] - 14s 234us/step - loss: 0.10
08 - acc: 0.9706 - val_loss: 0.0726 - val_acc: 0.9796
Epoch 17/20
60000/60000 [==============================] - 14s 234us/step - loss: 0.09
95 - acc: 0.9709 - val_loss: 0.0782 - val_acc: 0.9787
Epoch 18/20
60000/60000 [==============================] - 15s 248us/step - loss: 0.09
28 - acc: 0.9732 - val_loss: 0.0796 - val_acc: 0.9791
Epoch 19/20
60000/60000 [==============================] - 15s 245us/step - loss: 0.09
34 - acc: 0.9725 - val_loss: 0.0847 - val_acc: 0.9771
Epoch 20/20
60000/60000 [==============================] - 16s 260us/step - loss: 0.09
03 - acc: 0.9736 - val_loss: 0.0811 - val_acc: 0.9777
```

In [34]:

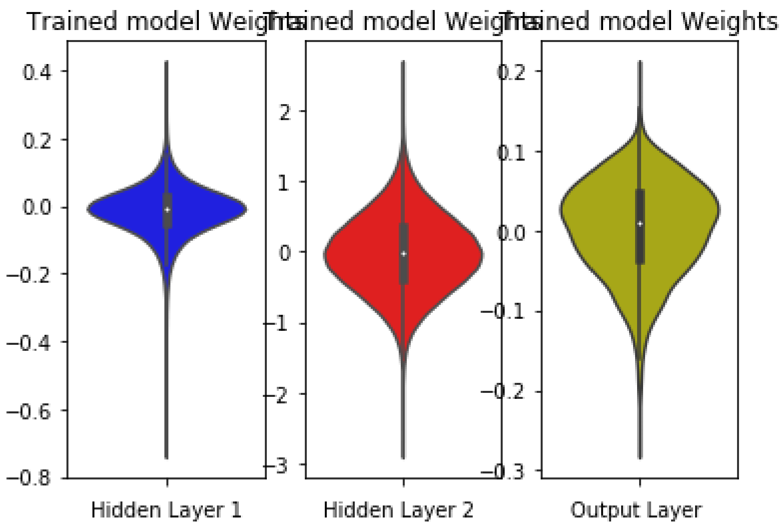
```
#Evaluate your model with accuracy and plot of (NUmber of epoches VS train_and_val_los
s)
#Train accuracy
score = model_drop.evaluate(X_train, Y_train, verbose=0)
print('Train score:', score[0])
print('Train accuracy:', score[1]*100)
print('\n*********************** ********************\n')
#test accuracy
score = model_drop.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1]*100)
fig,ax = plt.subplots(1,1)

ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')
# list of epoch numbers
x = list(range(1,nb_epoch+1))
# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ve
rbose=1, validation_data=(X_test, Y_test))
# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy
# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of ep
ochs
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Train score: 0.02664580340325483
Train accuracy: 99.25833333333334

*********************** ********************

Test score: 0.08111033427524962
Test accuracy: 97.77
```



In [35]:

```
w_after = model_drop.get_weights()
h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)
fig = plt.figure()
plt.title("Weight matrices after model trained")

plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

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

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



4. MLP + ReLU + adam + dropout+ batch_normalization

In [36]:

```
# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormalization-function-inkeras
from keras.layers import Dropout
model_drop = Sequential()
model_drop.add(Dense(610, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))
model_drop.add(Dense(325, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.55,seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))
model_drop.add(Dense(output_dim, activation='softmax'))
model_drop.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_10 (Dense)             (None, 610)               478850    
_________________________________________________________________
batch_normalization_3 (Batch (None, 610)               2440      
_________________________________________________________________
dropout_3 (Dropout)          (None, 610)               0         
_________________________________________________________________
dense_11 (Dense)             (None, 325)               198575    
_________________________________________________________________
batch_normalization_4 (Batch (None, 325)               1300      
_________________________________________________________________
dropout_4 (Dropout)          (None, 325)               0         
_________________________________________________________________
dense_12 (Dense)             (None, 10)                3260      
=================================================================
Total params: 684,425
Trainable params: 682,555
Non-trainable params: 1,870
_________________________________________________________________
```

In [37]:

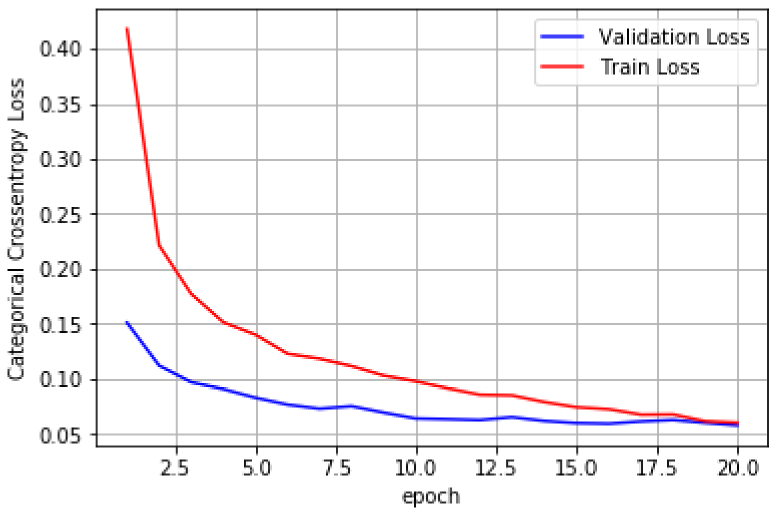
```
model_drop.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accurac
y'])
history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=20, verbose=1,
validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 25s 414us/step - loss: 0.41
80 - acc: 0.8724 - val_loss: 0.1510 - val_acc: 0.9508
Epoch 2/20
60000/60000 [==============================] - 20s 334us/step - loss: 0.22
14 - acc: 0.9328 - val_loss: 0.1119 - val_acc: 0.9643
Epoch 3/20
60000/60000 [==============================] - 20s 334us/step - loss: 0.17
71 - acc: 0.9452 - val_loss: 0.0968 - val_acc: 0.9698
Epoch 4/20
60000/60000 [==============================] - 18s 307us/step - loss: 0.15
13 - acc: 0.9536 - val_loss: 0.0906 - val_acc: 0.9714
Epoch 5/20
60000/60000 [==============================] - 19s 319us/step - loss: 0.14
00 - acc: 0.9577 - val_loss: 0.0829 - val_acc: 0.9740
Epoch 6/20
60000/60000 [==============================] - 18s 294us/step - loss: 0.12
27 - acc: 0.9621 - val_loss: 0.0767 - val_acc: 0.9758
Epoch 7/20
60000/60000 [==============================] - 18s 305us/step - loss: 0.11
81 - acc: 0.9627 - val_loss: 0.0730 - val_acc: 0.9778
Epoch 8/20
60000/60000 [==============================] - 19s 310us/step - loss: 0.11
14 - acc: 0.9651 - val_loss: 0.0753 - val_acc: 0.9756
Epoch 9/20
60000/60000 [==============================] - 19s 310us/step - loss: 0.10
29 - acc: 0.9670 - val_loss: 0.0694 - val_acc: 0.9774
Epoch 10/20
60000/60000 [==============================] - 19s 314us/step - loss: 0.09
77 - acc: 0.9693 - val_loss: 0.0640 - val_acc: 0.9786
Epoch 11/20
60000/60000 [==============================] - 18s 306us/step - loss: 0.09
13 - acc: 0.9720 - val_loss: 0.0634 - val_acc: 0.9809
Epoch 12/20
60000/60000 [==============================] - 19s 310us/step - loss: 0.08
56 - acc: 0.9727 - val_loss: 0.0626 - val_acc: 0.9806
Epoch 13/20
60000/60000 [==============================] - 18s 303us/step - loss: 0.08
51 - acc: 0.9731 - val_loss: 0.0653 - val_acc: 0.9797
Epoch 14/20
60000/60000 [==============================] - 18s 303us/step - loss: 0.07
91 - acc: 0.9746 - val_loss: 0.0618 - val_acc: 0.9810
Epoch 15/20
60000/60000 [==============================] - 18s 306us/step - loss: 0.07
44 - acc: 0.9763 - val_loss: 0.0599 - val_acc: 0.9819
Epoch 16/20
60000/60000 [==============================] - 18s 302us/step - loss: 0.07
25 - acc: 0.9768 - val_loss: 0.0594 - val_acc: 0.9816
Epoch 17/20
60000/60000 [==============================] - 18s 304us/step - loss: 0.06
75 - acc: 0.9783 - val_loss: 0.0614 - val_acc: 0.9815
Epoch 18/20
60000/60000 [==============================] - 18s 301us/step - loss: 0.06
77 - acc: 0.9783 - val_loss: 0.0627 - val_acc: 0.9804
Epoch 19/20
60000/60000 [==============================] - 18s 300us/step - loss: 0.06
15 - acc: 0.9802 - val_loss: 0.0601 - val_acc: 0.9810
Epoch 20/20
60000/60000 [==============================] - 19s 310us/step - loss: 0.06
02 - acc: 0.9807 - val_loss: 0.0578 - val_acc: 0.9819
```

In [38]:

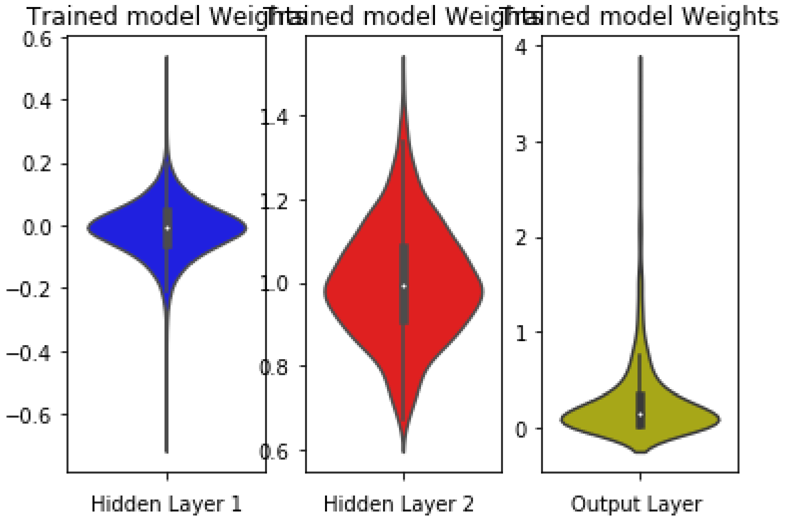
```
#Evaluate your model with accuracy and plot of (NUmber of epoches VS train_and_val_los
s)
#Train accuracy
score = model_drop.evaluate(X_train, Y_train, verbose=0)
print('Train score:', score[0])
print('Train accuracy:', score[1]*100)
print('\n*********************** ********************\n')
#test accuracy
score = model_drop.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1]*100)
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')
# list of epoch numbers
x = list(range(1,nb_epoch+1))
# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ve
rbose=1, validation_data=(X_test, Y_test))
# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy
# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of ep
ochs
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Train score: 0.014805679358745692
Train accuracy: 99.53666666666666

*********************** ********************

Test score: 0.05782464738052222
Test accuracy: 98.19
```



In [39]:

```
w_after = model_drop.get_weights()
h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)
fig = plt.figure()
plt.title("Weight matrices after model trained")

plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

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

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



Model 2 -- with 3 Hidden layers

1. MLP + ReLU + adam

In [40]:

```
# Multilayer perceptron
# https://arxiv.org/pdf/1707.09725.pdf#page=95
# for relu layers
# If we sample weights from a normal distribution N(0,σ) we satisfy this condition with σ=√(2/(ni).
# h1 => σ=√(2/(fan_in) = 0.062 => N(0,σ) = N(0,0.062)
# h2 => σ=√(2/(fan_in) = 0.125 => N(0,σ) = N(0,0.125)
# out => σ=√(2/(fan_in+1) = 0.120 => N(0,σ) = N(0,0.120)
model_relu = Sequential()
model_relu.add(Dense(610, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_relu.add(Dense(420, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_relu.add(Dense(210, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))
model_relu.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_13 (Dense)             (None, 610)               478850    
_________________________________________________________________
dense_14 (Dense)             (None, 420)               256620    
_________________________________________________________________
dense_15 (Dense)             (None, 210)               88410     
_________________________________________________________________
dense_16 (Dense)             (None, 10)                2110      
=================================================================
Total params: 825,990
Trainable params: 825,990
Non-trainable params: 0
_________________________________________________________________
```

In [41]:

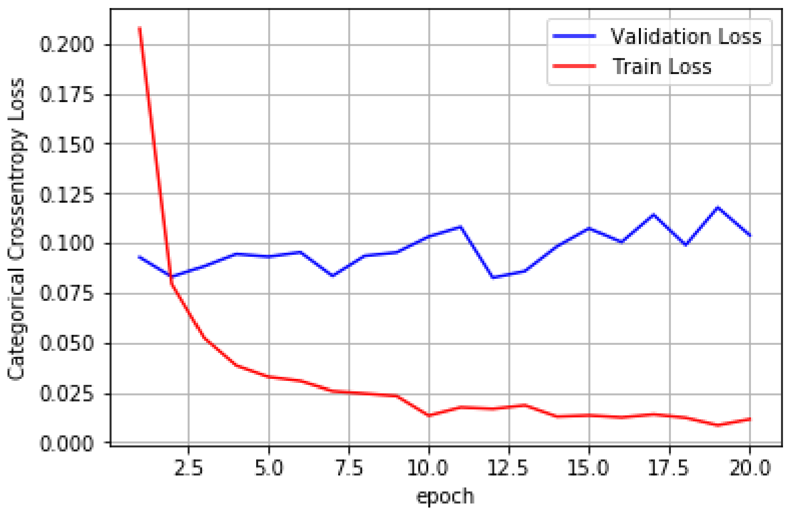
```
model_relu.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accurac
y'])
history = model_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verb
ose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 20s 330us/step - loss: 0.20
76 - acc: 0.9368 - val_loss: 0.0928 - val_acc: 0.9695
Epoch 2/20
60000/60000 [==============================] - 14s 240us/step - loss: 0.07
94 - acc: 0.9751 - val_loss: 0.0830 - val_acc: 0.9755
Epoch 3/20
60000/60000 [==============================] - 14s 240us/step - loss: 0.05
22 - acc: 0.9831 - val_loss: 0.0883 - val_acc: 0.9752
Epoch 4/20
60000/60000 [==============================] - 14s 239us/step - loss: 0.03
83 - acc: 0.9879 - val_loss: 0.0944 - val_acc: 0.9699
Epoch 5/20
60000/60000 [==============================] - 15s 244us/step - loss: 0.03
26 - acc: 0.9887 - val_loss: 0.0931 - val_acc: 0.9754
Epoch 6/20
60000/60000 [==============================] - 14s 240us/step - loss: 0.03
06 - acc: 0.9905 - val_loss: 0.0953 - val_acc: 0.9740
Epoch 7/20
60000/60000 [==============================] - 14s 236us/step - loss: 0.02
56 - acc: 0.9915 - val_loss: 0.0834 - val_acc: 0.9781
Epoch 8/20
60000/60000 [==============================] - 14s 234us/step - loss: 0.02
44 - acc: 0.9920 - val_loss: 0.0935 - val_acc: 0.9783
Epoch 9/20
60000/60000 [==============================] - 14s 235us/step - loss: 0.02
32 - acc: 0.9923 - val_loss: 0.0952 - val_acc: 0.9763
Epoch 10/20
60000/60000 [==============================] - 14s 237us/step - loss: 0.01
34 - acc: 0.9960 - val_loss: 0.1032 - val_acc: 0.9778
Epoch 11/20
60000/60000 [==============================] - 19s 324us/step - loss: 0.01
75 - acc: 0.9942 - val_loss: 0.1080 - val_acc: 0.9745
Epoch 12/20
60000/60000 [==============================] - 16s 271us/step - loss: 0.01
68 - acc: 0.9948 - val_loss: 0.0825 - val_acc: 0.9803
Epoch 13/20
60000/60000 [==============================] - 14s 238us/step - loss: 0.01
86 - acc: 0.9942 - val_loss: 0.0858 - val_acc: 0.9798
Epoch 14/20
60000/60000 [==============================] - 14s 237us/step - loss: 0.01
29 - acc: 0.9960 - val_loss: 0.0982 - val_acc: 0.9794
Epoch 15/20
60000/60000 [==============================] - 14s 240us/step - loss: 0.01
35 - acc: 0.9960 - val_loss: 0.1073 - val_acc: 0.9751
Epoch 16/20
60000/60000 [==============================] - 14s 238us/step - loss: 0.01
25 - acc: 0.9961 - val_loss: 0.1004 - val_acc: 0.9778
Epoch 17/20
60000/60000 [==============================] - 14s 236us/step - loss: 0.01
40 - acc: 0.9958 - val_loss: 0.1142 - val_acc: 0.9771
Epoch 18/20
60000/60000 [==============================] - 14s 237us/step - loss: 0.01
23 - acc: 0.9963 - val_loss: 0.0989 - val_acc: 0.9785
Epoch 19/20
60000/60000 [==============================] - 15s 245us/step - loss: 0.00
85 - acc: 0.9979 - val_loss: 0.1178 - val_acc: 0.9774
Epoch 20/20
60000/60000 [==============================] - 14s 237us/step - loss: 0.01
15 - acc: 0.9965 - val_loss: 0.1039 - val_acc: 0.9799
```

In [42]:

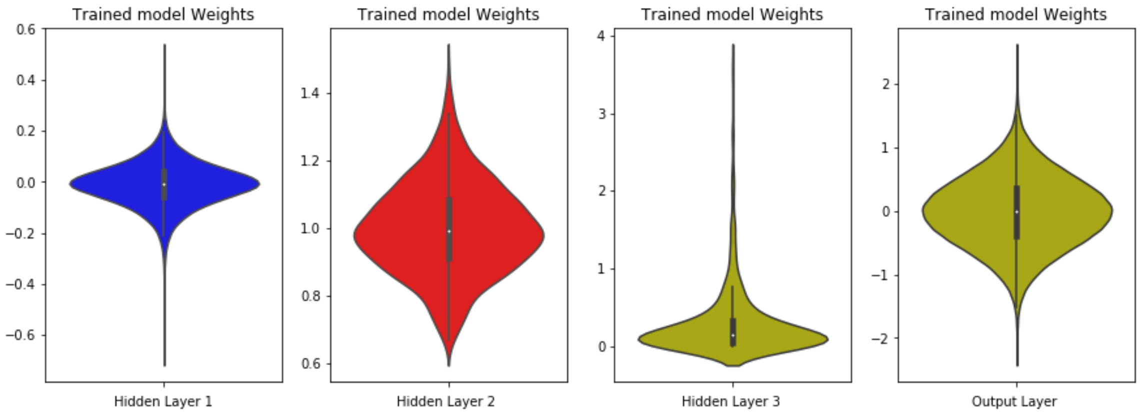
```
#Evualate your model with accuracy and plot of (NUmber of epoches VS train_and_val_los
s)
#Train accuracy
score = model_relu.evaluate(X_train, Y_train, verbose=0)
print('Train score:', score[0])
print('Train accuracy:', score[1]*100)
print('\n*********************** ********************\n')
#test accuracy
score = model_relu.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1]*100)
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')
# list of epoch numbers
x = list(range(1,nb_epoch+1))
# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ve
rbose=1, validation_data=(X_test, Y_test))
# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy
# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of ep
ochs
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Train score: 0.011968852381023158
Train accuracy: 99.63833333333334

*********************** ********************

Test score: 0.1038783551045065
Test accuracy: 97.99
```



In [43]:

```
w_after = model_drop.get_weights()
h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
out_w = w_after[6].flatten().reshape(-1,1)
fig = plt.figure(figsize=(15,5))
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='y')
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()
```



2. MLP + ReLU + adam +batch_normalization

In [44]:

```
from keras.layers.normalization import BatchNormalization
model_batch = Sequential()
model_batch.add(Dense(610, activation='relu', input_shape=(input_dim,), kernel_initiali
zer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_batch.add(BatchNormalization())
model_batch.add(Dense(420, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
stddev=0.55, seed=None)) )
model_batch.add(BatchNormalization())
model_batch.add(Dense(210, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
stddev=0.55, seed=None)) )
model_batch.add(BatchNormalization())
model_batch.add(Dense(output_dim, activation='softmax'))
model_batch.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_17 (Dense)             (None, 610)               478850    
_________________________________________________________________
batch_normalization_5 (Batch (None, 610)               2440      
_________________________________________________________________
dense_18 (Dense)             (None, 420)               256620    
_________________________________________________________________
batch_normalization_6 (Batch (None, 420)               1680      
_________________________________________________________________
dense_19 (Dense)             (None, 210)               88410     
_________________________________________________________________
batch_normalization_7 (Batch (None, 210)               840       
_________________________________________________________________
dense_20 (Dense)             (None, 10)                2110      
=================================================================
Total params: 830,950
Trainable params: 828,470
Non-trainable params: 2,480
_________________________________________________________________
```

In [46]:

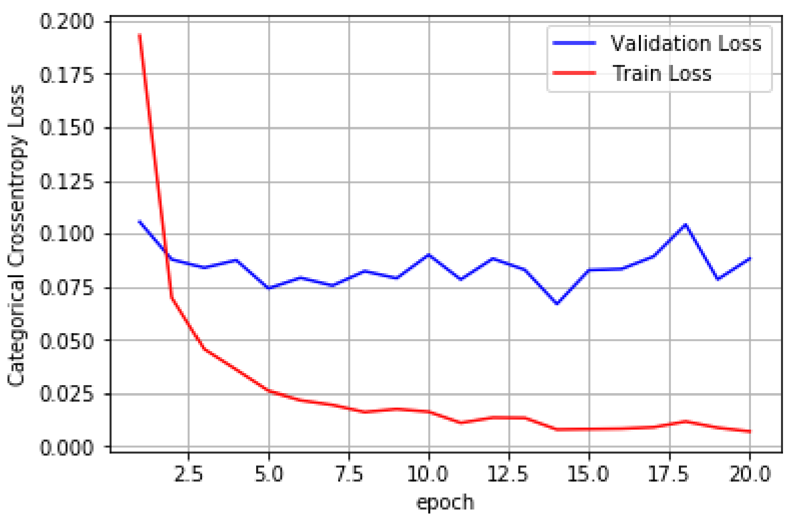
```
model_batch.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura
cy'])
history = model_batch.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ver
bose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 29s 490us/step - loss: 0.19
31 - acc: 0.9414 - val_loss: 0.1055 - val_acc: 0.9669
Epoch 2/20
60000/60000 [==============================] - 22s 373us/step - loss: 0.06
97 - acc: 0.9786 - val_loss: 0.0878 - val_acc: 0.9723
Epoch 3/20
60000/60000 [==============================] - 20s 337us/step - loss: 0.04
55 - acc: 0.9855 - val_loss: 0.0838 - val_acc: 0.9722
Epoch 4/20
60000/60000 [==============================] - 21s 342us/step - loss: 0.03
58 - acc: 0.9889 - val_loss: 0.0874 - val_acc: 0.9747
Epoch 5/20
60000/60000 [==============================] - 22s 370us/step - loss: 0.02
58 - acc: 0.9919 - val_loss: 0.0741 - val_acc: 0.9776
Epoch 6/20
60000/60000 [==============================] - 24s 394us/step - loss: 0.02
16 - acc: 0.9927 - val_loss: 0.0791 - val_acc: 0.9775
Epoch 7/20
60000/60000 [==============================] - 21s 351us/step - loss: 0.01
94 - acc: 0.9939 - val_loss: 0.0754 - val_acc: 0.9791
Epoch 8/20
60000/60000 [==============================] - 21s 347us/step - loss: 0.01
60 - acc: 0.9945 - val_loss: 0.0822 - val_acc: 0.9786
Epoch 9/20
60000/60000 [==============================] - 21s 358us/step - loss: 0.01
74 - acc: 0.9941 - val_loss: 0.0789 - val_acc: 0.9793
Epoch 10/20
60000/60000 [==============================] - 20s 334us/step - loss: 0.01
62 - acc: 0.9946 - val_loss: 0.0899 - val_acc: 0.9770
Epoch 11/20
60000/60000 [==============================] - 20s 329us/step - loss: 0.01
09 - acc: 0.9965 - val_loss: 0.0783 - val_acc: 0.9797
Epoch 12/20
60000/60000 [==============================] - 22s 367us/step - loss: 0.01
34 - acc: 0.9955 - val_loss: 0.0881 - val_acc: 0.9781
Epoch 13/20
60000/60000 [==============================] - 23s 378us/step - loss: 0.01
33 - acc: 0.9955 - val_loss: 0.0828 - val_acc: 0.9794
Epoch 14/20
60000/60000 [==============================] - 23s 390us/step - loss: 0.00
78 - acc: 0.9975 - val_loss: 0.0667 - val_acc: 0.9827
Epoch 15/20
60000/60000 [==============================] - 21s 349us/step - loss: 0.00
80 - acc: 0.9972 - val_loss: 0.0827 - val_acc: 0.9800
Epoch 16/20
60000/60000 [==============================] - 19s 323us/step - loss: 0.00
82 - acc: 0.9974 - val_loss: 0.0832 - val_acc: 0.9793
Epoch 17/20
60000/60000 [==============================] - 15s 243us/step - loss: 0.00
89 - acc: 0.9971 - val_loss: 0.0892 - val_acc: 0.9799
Epoch 18/20
60000/60000 [==============================] - 14s 228us/step - loss: 0.01
16 - acc: 0.9961 - val_loss: 0.1042 - val_acc: 0.9738
Epoch 19/20
60000/60000 [==============================] - 15s 258us/step - loss: 0.00
86 - acc: 0.9970 - val_loss: 0.0784 - val_acc: 0.9809
Epoch 20/20
60000/60000 [==============================] - 16s 263us/step - loss: 0.00
69 - acc: 0.9976 - val_loss: 0.0881 - val_acc: 0.9803
```

In [48]:

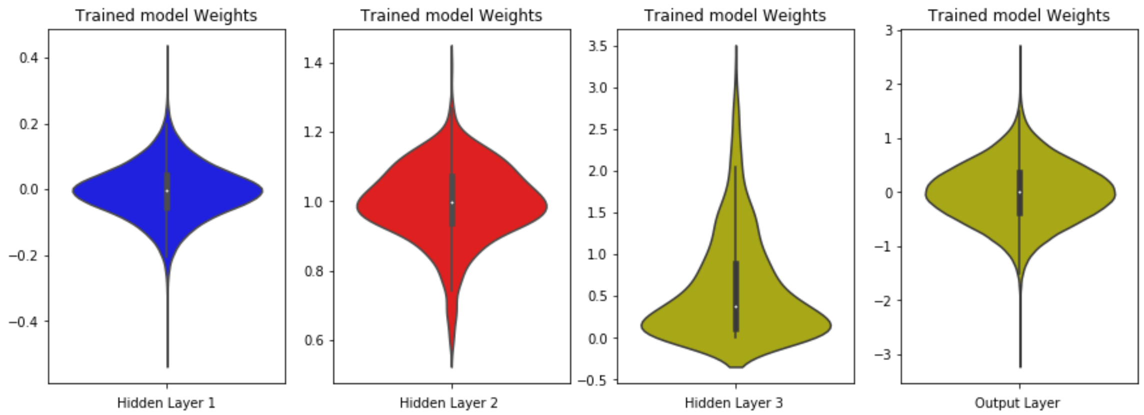
```
#Evaluate your model with accuracy and plot of (NUmber of epoches VS train_and_val_los
s)
#Train accuracy
score = model_batch.evaluate(X_train, Y_train, verbose=0)
print('Train score:', score[0])
print('Train accuracy:', score[1]*100)
print('\n*********************** ********************\n')
#test accuracy
score = model_batch.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1]*100)
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')
# list of epoch numbers
x = list(range(1,nb_epoch+1))
# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ve
rbose=1, validation_data=(X_test, Y_test))
# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy
# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of ep
ochs
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Train score: 0.00419184478967436
Train accuracy: 99.87833333333333

*********************** ********************

Test score: 0.08813276136659533
Test accuracy: 98.03
```



In [49]:

```
w_after = model_batch.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(figsize=(15,5))
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='y')
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()
```



3. MLP + ReLU + adam +dropout

In [50]:

```
# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormalization-f
unction-inkeras
from keras.layers import Dropout
model_drop = Sequential()
model_drop.add(Dense(610, activation='relu', input_shape=(input_dim,), kernel_initializ
er=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
#model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))
model_drop.add(Dense(420, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
stddev=0.55,seed=None)) )
#model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))
model_drop.add(Dense(210, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
stddev=0.55,seed=None)) )
#model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))
model_drop.add(Dense(output_dim, activation='softmax'))
model_drop.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_21 (Dense)             (None, 610)               478850
_________________________________________________________________
dropout_5 (Dropout)          (None, 610)               0
_________________________________________________________________
dense_22 (Dense)             (None, 420)               256620
_________________________________________________________________
dropout_6 (Dropout)          (None, 420)               0
_________________________________________________________________
dense_23 (Dense)             (None, 210)               88410
_________________________________________________________________
dropout_7 (Dropout)          (None, 210)               0
_________________________________________________________________
dense_24 (Dense)             (None, 10)                2110
=================================================================
Total params: 825,990
Trainable params: 825,990
Non-trainable params: 0
_________________________________________________________________
```

In [51]:

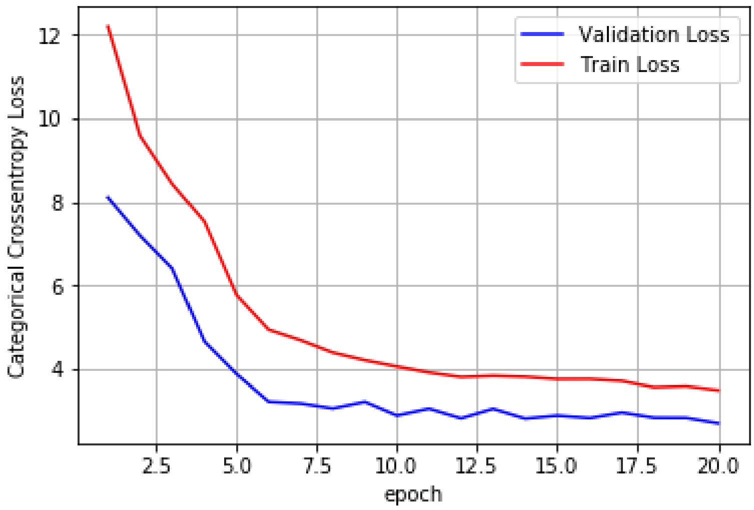
```
model_drop.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accurac
y'])
history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verb
ose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 13s 222us/step - loss: 12.1
946 - acc: 0.2356 - val_loss: 8.1071 - val_acc: 0.4922
Epoch 2/20
60000/60000 [==============================] - 12s 197us/step - loss: 9.58
85 - acc: 0.3991 - val_loss: 7.1958 - val_acc: 0.5484
Epoch 3/20
60000/60000 [==============================] - 12s 200us/step - loss: 8.43
35 - acc: 0.4718 - val_loss: 6.4075 - val_acc: 0.5999
Epoch 4/20
60000/60000 [==============================] - 12s 196us/step - loss: 7.53
02 - acc: 0.5274 - val_loss: 4.6527 - val_acc: 0.7075
Epoch 5/20
60000/60000 [==============================] - 11s 185us/step - loss: 5.76
84 - acc: 0.6372 - val_loss: 3.8773 - val_acc: 0.7567
Epoch 6/20
60000/60000 [==============================] - 10s 175us/step - loss: 4.93
82 - acc: 0.6893 - val_loss: 3.2168 - val_acc: 0.7982
Epoch 7/20
60000/60000 [==============================] - 10s 174us/step - loss: 4.68
53 - acc: 0.7055 - val_loss: 3.1759 - val_acc: 0.8007
Epoch 8/20
60000/60000 [==============================] - 10s 174us/step - loss: 4.38
97 - acc: 0.7244 - val_loss: 3.0566 - val_acc: 0.8088
Epoch 9/20
60000/60000 [==============================] - 10s 174us/step - loss: 4.20
75 - acc: 0.7361 - val_loss: 3.2145 - val_acc: 0.7993
Epoch 10/20
60000/60000 [==============================] - 10s 175us/step - loss: 4.05
65 - acc: 0.7455 - val_loss: 2.8893 - val_acc: 0.8197
Epoch 11/20
60000/60000 [==============================] - 11s 184us/step - loss: 3.91
38 - acc: 0.7547 - val_loss: 3.0524 - val_acc: 0.8097
Epoch 12/20
60000/60000 [==============================] - 10s 174us/step - loss: 3.80
66 - acc: 0.7616 - val_loss: 2.8225 - val_acc: 0.8236
Epoch 13/20
60000/60000 [==============================] - 10s 174us/step - loss: 3.83
58 - acc: 0.7599 - val_loss: 3.0483 - val_acc: 0.8094
Epoch 14/20
60000/60000 [==============================] - 10s 174us/step - loss: 3.81
29 - acc: 0.7613 - val_loss: 2.8157 - val_acc: 0.8247
Epoch 15/20
60000/60000 [==============================] - 10s 174us/step - loss: 3.75
80 - acc: 0.7652 - val_loss: 2.8874 - val_acc: 0.8200
Epoch 16/20
60000/60000 [==============================] - 10s 174us/step - loss: 3.75
95 - acc: 0.7652 - val_loss: 2.8289 - val_acc: 0.8235
Epoch 17/20
60000/60000 [==============================] - 10s 174us/step - loss: 3.71
41 - acc: 0.7681 - val_loss: 2.9597 - val_acc: 0.8155
Epoch 18/20
60000/60000 [==============================] - 11s 177us/step - loss: 3.56
06 - acc: 0.7777 - val_loss: 2.8367 - val_acc: 0.8231
Epoch 19/20
60000/60000 [==============================] - 10s 174us/step - loss: 3.58
46 - acc: 0.7763 - val_loss: 2.8327 - val_acc: 0.8235
Epoch 20/20
60000/60000 [==============================] - 10s 173us/step - loss: 3.48
81 - acc: 0.7821 - val_loss: 2.7042 - val_acc: 0.8314
```

In [52]:

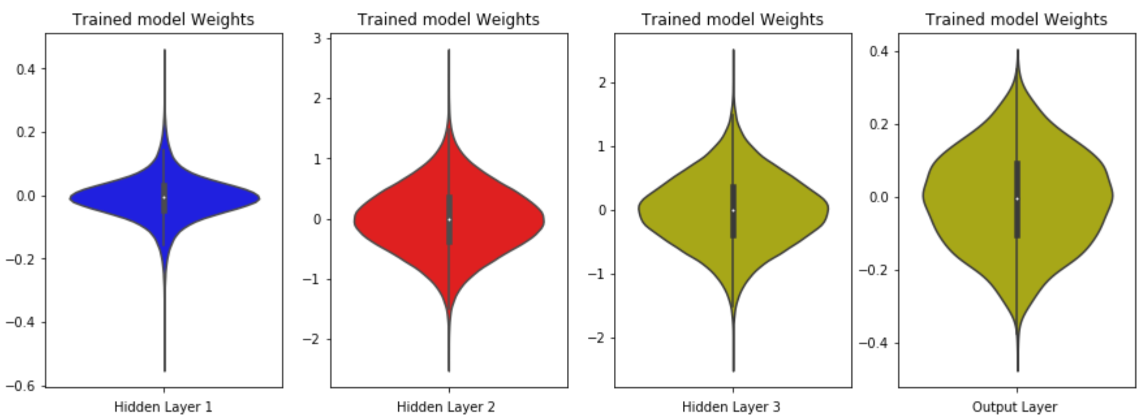
```
#Evaluate your model with accuracy and plot of (NUmber of epoches VS train_and_val_los
s)
#Train accuracy
score = model_drop.evaluate(X_train, Y_train, verbose=0)
print('Train score:', score[0])
print('Train accuracy:', score[1]*100)
print('\n*********************** ********************\n')
#test accuracy
score = model_drop.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1]*100)
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')
# list of epoch numbers
x = list(range(1,nb_epoch+1))
# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ve
rbose=1, validation_data=(X_test, Y_test))
# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy
# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of ep
ochs
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Train score: 2.7385955852190653
Train accuracy: 82.94666666666667

*********************** ********************

Test score: 2.7042304149627685
Test accuracy: 83.14
```



In [53]:

```
w_after = model_drop.get_weights()
h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
out_w = w_after[6].flatten().reshape(-1,1)
fig = plt.figure(figsize=(15,5))
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='y')
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()
```



4. MLP + ReLU + adam +dropout+batch_normalization

In [54]:

```
# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormalization-f
unction-inkeras
from keras.layers import Dropout
model_drop = Sequential()
model_drop.add(Dense(610, activation='relu', input_shape=(input_dim,), kernel_initializ
er=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))
model_drop.add(Dense(420, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
stddev=0.55,seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))
model_drop.add(Dense(210, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
stddev=0.55,seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))
model_drop.add(Dense(output_dim, activation='softmax'))
model_drop.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_25 (Dense)             (None, 610)               478850
_________________________________________________________________
batch_normalization_8 (Batch (None, 610)               2440
_________________________________________________________________
dropout_8 (Dropout)          (None, 610)               0
_________________________________________________________________
dense_26 (Dense)             (None, 420)               256620
_________________________________________________________________
batch_normalization_9 (Batch (None, 420)               1680
_________________________________________________________________
dropout_9 (Dropout)          (None, 420)               0
_________________________________________________________________
dense_27 (Dense)             (None, 210)               88410
_________________________________________________________________
batch_normalization_10 (Batc (None, 210)               840
_________________________________________________________________
dropout_10 (Dropout)         (None, 210)               0
_________________________________________________________________
dense_28 (Dense)             (None, 10)                2110
=================================================================
Total params: 830,950
Trainable params: 828,470
Non-trainable params: 2,480
_________________________________________________________________
```

In [55]:

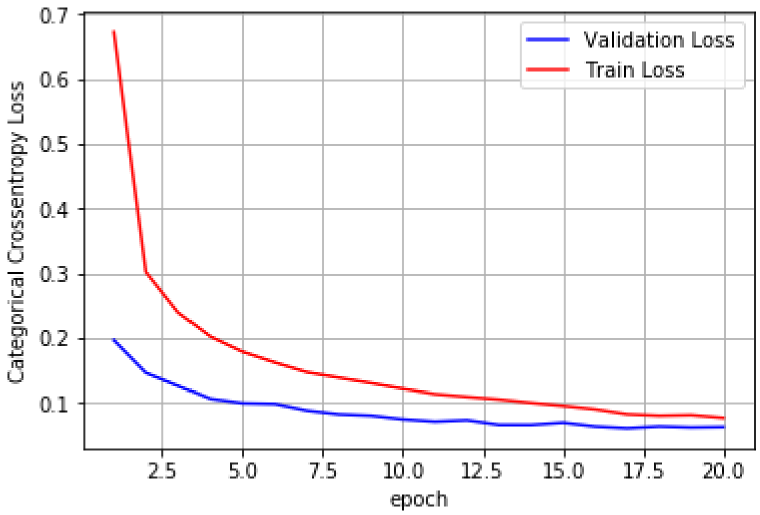
```
model_drop.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accurac
y'])
history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verb
ose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 17s 279us/step - loss: 0.67
29 - acc: 0.7925 - val_loss: 0.1967 - val_acc: 0.9368
Epoch 2/20
60000/60000 [==============================] - 14s 236us/step - loss: 0.30
22 - acc: 0.9089 - val_loss: 0.1463 - val_acc: 0.9517
Epoch 3/20
60000/60000 [==============================] - 13s 219us/step - loss: 0.23
94 - acc: 0.9295 - val_loss: 0.1262 - val_acc: 0.9610
Epoch 4/20
60000/60000 [==============================] - 13s 225us/step - loss: 0.20
20 - acc: 0.9397 - val_loss: 0.1061 - val_acc: 0.9669
Epoch 5/20
60000/60000 [==============================] - 13s 212us/step - loss: 0.17
87 - acc: 0.9465 - val_loss: 0.0994 - val_acc: 0.9705
Epoch 6/20
60000/60000 [==============================] - 14s 234us/step - loss: 0.16
23 - acc: 0.9520 - val_loss: 0.0983 - val_acc: 0.9719
Epoch 7/20
60000/60000 [==============================] - 14s 226us/step - loss: 0.14
73 - acc: 0.9554 - val_loss: 0.0881 - val_acc: 0.9736
Epoch 8/20
60000/60000 [==============================] - 15s 251us/step - loss: 0.13
89 - acc: 0.9581 - val_loss: 0.0823 - val_acc: 0.9742
Epoch 9/20
60000/60000 [==============================] - 19s 312us/step - loss: 0.13
06 - acc: 0.9610 - val_loss: 0.0802 - val_acc: 0.9756
Epoch 10/20
60000/60000 [==============================] - 17s 288us/step - loss: 0.12
18 - acc: 0.9628 - val_loss: 0.0745 - val_acc: 0.9762
Epoch 11/20
60000/60000 [==============================] - 19s 320us/step - loss: 0.11
33 - acc: 0.9665 - val_loss: 0.0711 - val_acc: 0.9792
Epoch 12/20
60000/60000 [==============================] - 19s 314us/step - loss: 0.10
91 - acc: 0.9669 - val_loss: 0.0732 - val_acc: 0.9781
Epoch 13/20
60000/60000 [==============================] - 18s 295us/step - loss: 0.10
52 - acc: 0.9679 - val_loss: 0.0660 - val_acc: 0.9807
Epoch 14/20
60000/60000 [==============================] - 17s 282us/step - loss: 0.10
00 - acc: 0.9695 - val_loss: 0.0660 - val_acc: 0.9793
Epoch 15/20
60000/60000 [==============================] - 16s 266us/step - loss: 0.09
54 - acc: 0.9712 - val_loss: 0.0694 - val_acc: 0.9802
Epoch 16/20
60000/60000 [==============================] - 16s 272us/step - loss: 0.09
01 - acc: 0.9729 - val_loss: 0.0635 - val_acc: 0.9825
Epoch 17/20
60000/60000 [==============================] - 17s 280us/step - loss: 0.08
25 - acc: 0.9746 - val_loss: 0.0609 - val_acc: 0.9832
Epoch 18/20
60000/60000 [==============================] - 19s 310us/step - loss: 0.08
02 - acc: 0.9756 - val_loss: 0.0636 - val_acc: 0.9819
Epoch 19/20
60000/60000 [==============================] - 20s 336us/step - loss: 0.08
11 - acc: 0.9752 - val_loss: 0.0622 - val_acc: 0.9820oss: 0.0814 -
Epoch 20/20
60000/60000 [==============================] - 17s 283us/step - loss: 0.07
69 - acc: 0.9763 - val_loss: 0.0628 - val_acc: 0.9821
```

In [56]:

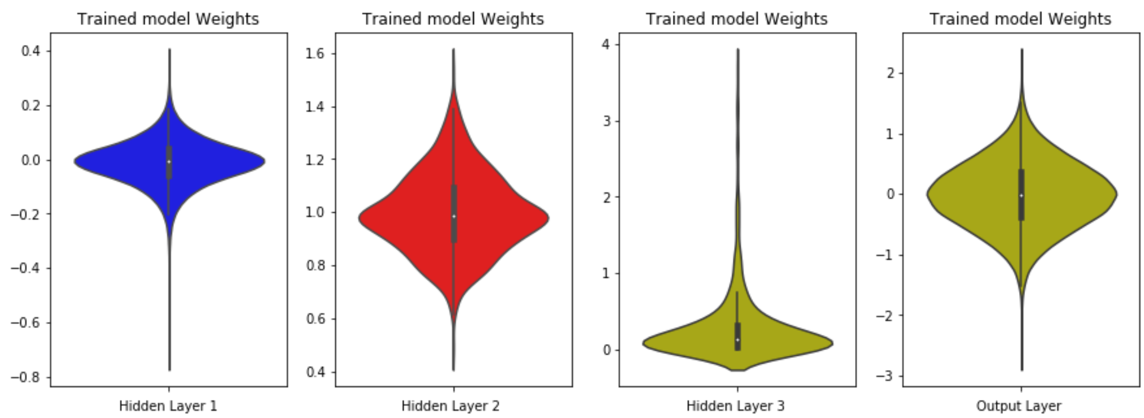
```
#Evaluate your model with accuracy and plot of (NUmber of epoches VS train_and_val_los
s)
#Train accuracy
score = model_drop.evaluate(X_train, Y_train, verbose=0)
print('Train score:', score[0])
print('Train accuracy:', score[1]*100)
print('\n*********************** ********************\n')
#test accuracy
score = model_drop.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1]*100)
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')
# list of epoch numbers
x = list(range(1,nb_epoch+1))
# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ve
rbose=1, validation_data=(X_test, Y_test))
# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy
# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of ep
ochs
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Train score: 0.021243010153949343
Train accuracy: 99.33166666666666

*********************** ********************

Test score: 0.0627561581715534
Test accuracy: 98.21
```



In [57]:

```
w_after = model_drop.get_weights()
h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
out_w = w_after[6].flatten().reshape(-1,1)
fig = plt.figure(figsize=(15,5))
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='y')
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()
```



Model 3 -- with 5 Hidden layers

1. MLP + ReLU + adam

In [58]:

```
# Multilayer perceptron
# https://arxiv.org/pdf/1707.09725.pdf#page=95
# for relu layers
# If we sample weights from a normal distribution N(0,σ) we satisfy this condition with
σ=√(2/(ni).
# h1 => σ=√(2/(fan_in) = 0.062 => N(0,σ) = N(0,0.062)
# h2 => σ=√(2/(fan_in) = 0.125 => N(0,σ) = N(0,0.125)
# out => σ=√(2/(fan_in+1) = 0.120 => N(0,σ) = N(0,0.120)
model_relu = Sequential()
model_relu.add(Dense(690, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_relu.add(Dense(530, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_relu.add(Dense(412, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_relu.add(Dense(231, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_relu.add(Dense(112, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))
model_relu.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_29 (Dense)             (None, 690)               541650    
_________________________________________________________________
dense_30 (Dense)             (None, 530)               366230    
_________________________________________________________________
dense_31 (Dense)             (None, 412)               218772    
_________________________________________________________________
dense_32 (Dense)             (None, 231)               95403     
_________________________________________________________________
dense_33 (Dense)             (None, 112)               25984     
_________________________________________________________________
dense_34 (Dense)             (None, 10)                1130      
=================================================================
Total params: 1,249,169
Trainable params: 1,249,169
Non-trainable params: 0
_________________________________________________________________
```

In [59]:

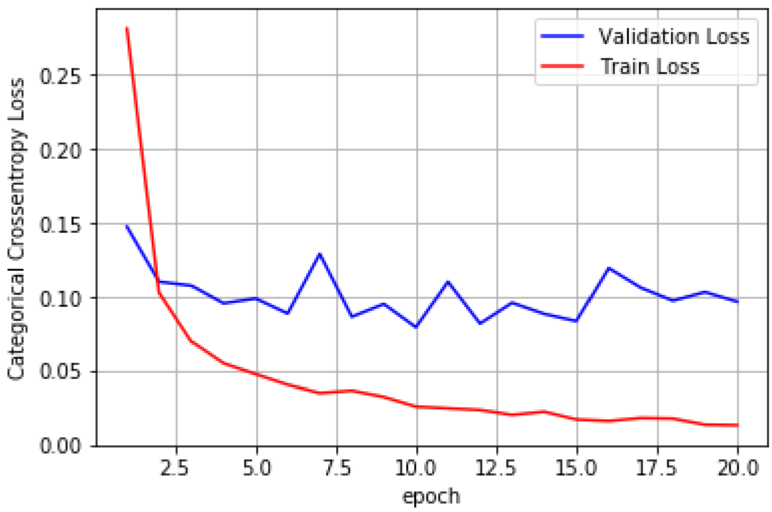
```
model_relu.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accurac
y'])
history = model_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verb
ose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 17s 290us/step - loss: 0.28
13 - acc: 0.9226 - val_loss: 0.1477 - val_acc: 0.9539
Epoch 2/20
60000/60000 [==============================] - 15s 249us/step - loss: 0.10
31 - acc: 0.9676 - val_loss: 0.1105 - val_acc: 0.9676
Epoch 3/20
60000/60000 [==============================] - 17s 281us/step - loss: 0.07
02 - acc: 0.9777 - val_loss: 0.1078 - val_acc: 0.9687
Epoch 4/20
60000/60000 [==============================] - 17s 283us/step - loss: 0.05
54 - acc: 0.9823 - val_loss: 0.0959 - val_acc: 0.9752
Epoch 5/20
60000/60000 [==============================] - 16s 259us/step - loss: 0.04
81 - acc: 0.9847 - val_loss: 0.0991 - val_acc: 0.9724
Epoch 6/20
60000/60000 [==============================] - 17s 277us/step - loss: 0.04
09 - acc: 0.9870 - val_loss: 0.0890 - val_acc: 0.9738
Epoch 7/20
60000/60000 [==============================] - 15s 258us/step - loss: 0.03
54 - acc: 0.9890 - val_loss: 0.1293 - val_acc: 0.9684
Epoch 8/20
60000/60000 [==============================] - 15s 256us/step - loss: 0.03
70 - acc: 0.9884 - val_loss: 0.0868 - val_acc: 0.9764
Epoch 9/20
60000/60000 [==============================] - 17s 289us/step - loss: 0.03
28 - acc: 0.9897 - val_loss: 0.0954 - val_acc: 0.9750
Epoch 10/20
60000/60000 [==============================] - 16s 265us/step - loss: 0.02
63 - acc: 0.9915 - val_loss: 0.0797 - val_acc: 0.9799
Epoch 11/20
60000/60000 [==============================] - 17s 289us/step - loss: 0.02
52 - acc: 0.9925 - val_loss: 0.1105 - val_acc: 0.9750
Epoch 12/20
60000/60000 [==============================] - 16s 267us/step - loss: 0.02
41 - acc: 0.9925 - val_loss: 0.0822 - val_acc: 0.9801
Epoch 13/20
60000/60000 [==============================] - 18s 295us/step - loss: 0.02
07 - acc: 0.9937 - val_loss: 0.0962 - val_acc: 0.9777
Epoch 14/20
60000/60000 [==============================] - 18s 306us/step - loss: 0.02
29 - acc: 0.9931 - val_loss: 0.0887 - val_acc: 0.9770
Epoch 15/20
60000/60000 [==============================] - 19s 322us/step - loss: 0.01
77 - acc: 0.9942 - val_loss: 0.0838 - val_acc: 0.9815
Epoch 16/20
60000/60000 [==============================] - 17s 275us/step - loss: 0.01
66 - acc: 0.9952 - val_loss: 0.1196 - val_acc: 0.9735
Epoch 17/20
60000/60000 [==============================] - 17s 278us/step - loss: 0.01
86 - acc: 0.9946 - val_loss: 0.1064 - val_acc: 0.9767
Epoch 18/20
60000/60000 [==============================] - 18s 295us/step - loss: 0.01
82 - acc: 0.9946 - val_loss: 0.0977 - val_acc: 0.9782
Epoch 19/20
60000/60000 [==============================] - 19s 317us/step - loss: 0.01
41 - acc: 0.9960 - val_loss: 0.1035 - val_acc: 0.9790
Epoch 20/20
60000/60000 [==============================] - 18s 296us/step - loss: 0.01
38 - acc: 0.9963 - val_loss: 0.0970 - val_acc: 0.9802
```

In [60]:

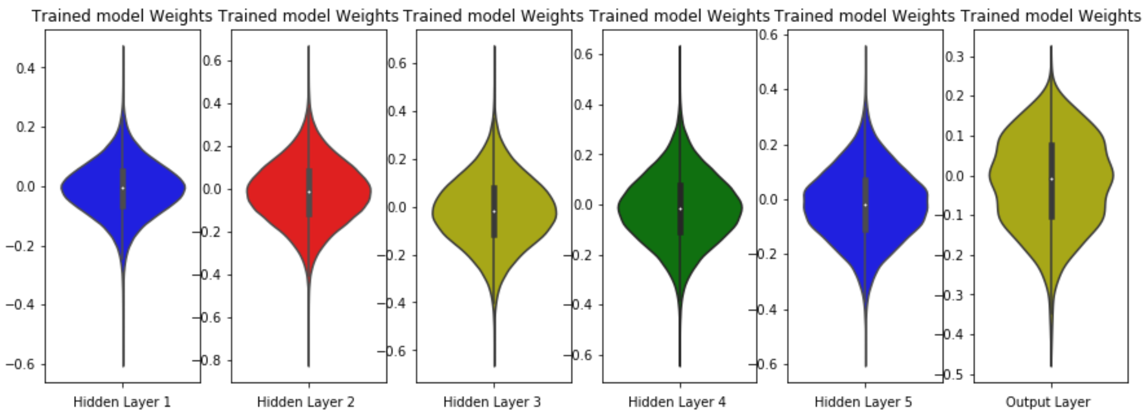
```
#Evaluate your model with accuracy and plot of (NUmber of epoches VS train_and_val_los
s)
#Train accuracy
score = model_relu.evaluate(X_train, Y_train, verbose=0)
print('Train score:', score[0])
print('Train accuracy:', score[1]*100)
print('\n*********************** ********************\n')
#test accuracy
score = model_relu.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1]*100)
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')
# list of epoch numbers
x = list(range(1,nb_epoch+1))
# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ve
rbose=1, validation_data=(X_test, Y_test))
# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy
# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of ep
ochs
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Train score: 0.007615429904182383
Train accuracy: 99.78333333333333

*********************** ********************

Test score: 0.09702783975718078
Test accuracy: 98.02
```



In [61]:

```
w_after = model_relu.get_weights()
h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
h4_w = w_after[6].flatten().reshape(-1,1)
h5_w = w_after[8].flatten().reshape(-1,1)
out_w = w_after[10].flatten().reshape(-1,1)
fig = plt.figure(figsize=(15,5))
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='y')
plt.xlabel('Hidden Layer 3 ')

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

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



2. MLP + ReLU + adam +batch_normalization

In [62]:

```
# Multilayer perceptron
# https://arxiv.org/pdf/1707.09725.pdf#page=95
# for relu layers
# If we sample weights from a normal distribution N(0,σ) we satisfy this condition with σ=√(2/(ni).
# h1 => σ=√(2/(fan_in) = 0.062 => N(0,σ) = N(0,0.062)
# h2 => σ=√(2/(fan_in) = 0.125 => N(0,σ) = N(0,0.125)
# out => σ=√(2/(fan_in+1) = 0.120 => N(0,σ) = N(0,0.120)
model_relu = Sequential()
model_relu.add(Dense(690, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dense(530, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_relu.add(BatchNormalization())
model_relu.add(Dense(412, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_relu.add(BatchNormalization())
model_relu.add(Dense(231, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_relu.add(BatchNormalization())
model_relu.add(Dense(112, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_relu.add(BatchNormalization())
model_relu.add(Dense(output_dim, activation='softmax'))
model_relu.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_35 (Dense)             (None, 690)               541650    
_________________________________________________________________
batch_normalization_11 (Batc (None, 690)               2760      
_________________________________________________________________
dense_36 (Dense)             (None, 530)               366230    
_________________________________________________________________
batch_normalization_12 (Batc (None, 530)               2120      
_________________________________________________________________
dense_37 (Dense)             (None, 412)               218772    
_________________________________________________________________
batch_normalization_13 (Batc (None, 412)               1648      
_________________________________________________________________
dense_38 (Dense)             (None, 231)               95403     
_________________________________________________________________
batch_normalization_14 (Batc (None, 231)               924       
_________________________________________________________________
dense_39 (Dense)             (None, 112)               25984     
_________________________________________________________________
batch_normalization_15 (Batc (None, 112)               448       
_________________________________________________________________
dense_40 (Dense)             (None, 10)                1130      
=================================================================
Total params: 1,257,069
Trainable params: 1,253,119
Non-trainable params: 3,950
_________________________________________________________________
```

In [63]:

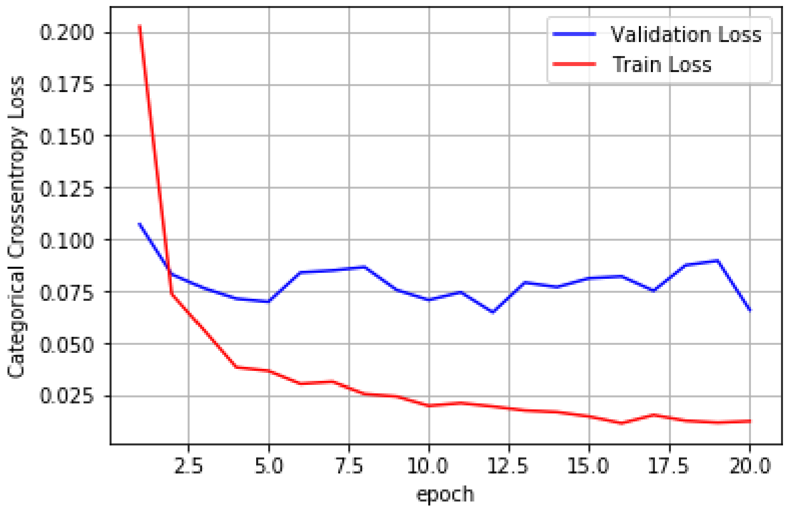
```
model_relu.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accurac
y'])
history = model_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verb
ose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 23s 383us/step - loss: 0.20
23 - acc: 0.9385 - val_loss: 0.1073 - val_acc: 0.9675
Epoch 2/20
60000/60000 [==============================] - 18s 305us/step - loss: 0.07
37 - acc: 0.9770 - val_loss: 0.0832 - val_acc: 0.9740
Epoch 3/20
60000/60000 [==============================] - 20s 327us/step - loss: 0.05
62 - acc: 0.9816 - val_loss: 0.0765 - val_acc: 0.9750
Epoch 4/20
60000/60000 [==============================] - 21s 344us/step - loss: 0.03
84 - acc: 0.9880 - val_loss: 0.0715 - val_acc: 0.9772
Epoch 5/20
60000/60000 [==============================] - 20s 333us/step - loss: 0.03
67 - acc: 0.9874 - val_loss: 0.0700 - val_acc: 0.9789
Epoch 6/20
60000/60000 [==============================] - 18s 297us/step - loss: 0.03
05 - acc: 0.9901 - val_loss: 0.0841 - val_acc: 0.9759
Epoch 7/20
60000/60000 [==============================] - 19s 313us/step - loss: 0.03
15 - acc: 0.9895 - val_loss: 0.0850 - val_acc: 0.9780
Epoch 8/20
60000/60000 [==============================] - 19s 318us/step - loss: 0.02
58 - acc: 0.9914 - val_loss: 0.0867 - val_acc: 0.9753
Epoch 9/20
60000/60000 [==============================] - 19s 321us/step - loss: 0.02
46 - acc: 0.9919 - val_loss: 0.0757 - val_acc: 0.9785
Epoch 10/20
60000/60000 [==============================] - 20s 327us/step - loss: 0.02
01 - acc: 0.9935 - val_loss: 0.0708 - val_acc: 0.9793
Epoch 11/20
60000/60000 [==============================] - 20s 337us/step - loss: 0.02
14 - acc: 0.9926 - val_loss: 0.0746 - val_acc: 0.9795
Epoch 12/20
60000/60000 [==============================] - 21s 350us/step - loss: 0.01
98 - acc: 0.9934 - val_loss: 0.0649 - val_acc: 0.9814
Epoch 13/20
60000/60000 [==============================] - 22s 367us/step - loss: 0.01
79 - acc: 0.9941 - val_loss: 0.0792 - val_acc: 0.9779
Epoch 14/20
60000/60000 [==============================] - 21s 347us/step - loss: 0.01
71 - acc: 0.9943 - val_loss: 0.0770 - val_acc: 0.9804
Epoch 15/20
60000/60000 [==============================] - 19s 323us/step - loss: 0.01
50 - acc: 0.9950 - val_loss: 0.0813 - val_acc: 0.9801
Epoch 16/20
60000/60000 [==============================] - 16s 268us/step - loss: 0.01
17 - acc: 0.9965 - val_loss: 0.0822 - val_acc: 0.9792
Epoch 17/20
60000/60000 [==============================] - 16s 267us/step - loss: 0.01
57 - acc: 0.9947 - val_loss: 0.0753 - val_acc: 0.9804
Epoch 18/20
60000/60000 [==============================] - 16s 265us/step - loss: 0.01
29 - acc: 0.9956 - val_loss: 0.0876 - val_acc: 0.9790
Epoch 19/20
60000/60000 [==============================] - 16s 263us/step - loss: 0.01
20 - acc: 0.9960 - val_loss: 0.0897 - val_acc: 0.9780
Epoch 20/20
60000/60000 [==============================] - 16s 263us/step - loss: 0.01
27 - acc: 0.9959 - val_loss: 0.0660 - val_acc: 0.9834
```

In [64]:

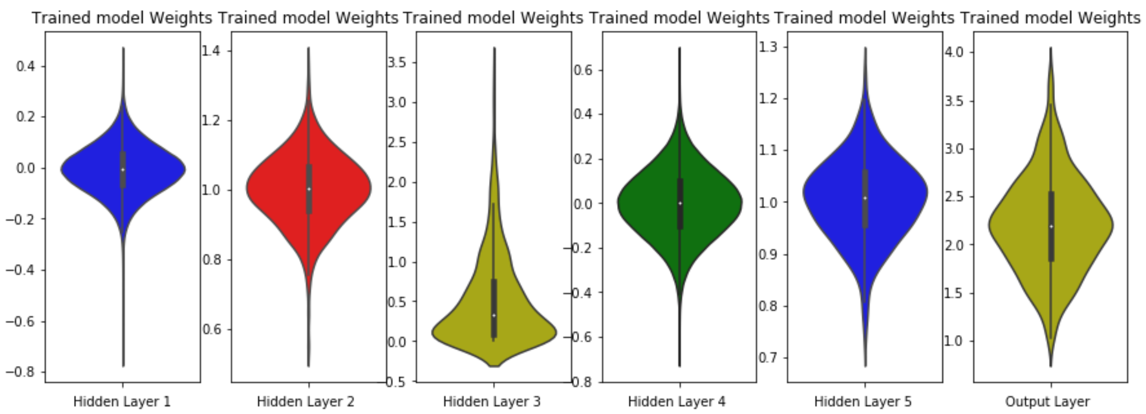
```
#Evaluate your model with accuracy and plot of (NUmber of epoches VS train_and_val_los
s)
#Train accuracy
score = model_relu.evaluate(X_train, Y_train, verbose=0)
print('Train score:', score[0])
print('Train accuracy:', score[1]*100)
print('\n*********************** ********************\n')
#test accuracy
score = model_relu.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1]*100)
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')
# list of epoch numbers
x = list(range(1,nb_epoch+1))
# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ve
rbose=1, validation_data=(X_test, Y_test))
# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy
# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of ep
ochs
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Train score: 0.006093753856421487
Train accuracy: 99.79833333333333

*********************** ********************

Test score: 0.06601150656397804
Test accuracy: 98.34
```



In [65]:

```
w_after = model_relu.get_weights()
h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
h4_w = w_after[6].flatten().reshape(-1,1)
h5_w = w_after[8].flatten().reshape(-1,1)
out_w = w_after[10].flatten().reshape(-1,1)
fig = plt.figure(figsize=(15,5))
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='y')
plt.xlabel('Hidden Layer 3 ')

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

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



3. MLP + ReLU + adam + dropout

In [66]:

```
# Multilayer perceptron
# https://arxiv.org/pdf/1707.09725.pdf#page=95
# for relu layers
# If we sample weights from a normal distribution N(0,σ) we satisfy this condition with
σ=√(2/(ni).
# h1 => σ=√(2/(fan_in) = 0.062 => N(0,σ) = N(0,0.062)
# h2 => σ=√(2/(fan_in) = 0.125 => N(0,σ) = N(0,0.125)
# out => σ=√(2/(fan_in+1) = 0.120 => N(0,σ) = N(0,0.120)
model_relu = Sequential()
model_relu.add(Dense(690, activation='relu', input_shape=(input_dim,), kernel_initializ
er=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
#model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(530, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
stddev=0.125, seed=None)) )
#model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(412, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
stddev=0.125, seed=None)) )
#model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(231, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
stddev=0.125, seed=None)) )
#model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(112, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
stddev=0.125, seed=None)) )
#model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(output_dim, activation='softmax'))
model_relu.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_41 (Dense)             (None, 690)               541650    
_________________________________________________________________
dropout_11 (Dropout)         (None, 690)               0         
_________________________________________________________________
dense_42 (Dense)             (None, 530)               366230    
_________________________________________________________________
dropout_12 (Dropout)         (None, 530)               0         
_________________________________________________________________
dense_43 (Dense)             (None, 412)               218772    
_________________________________________________________________
dropout_13 (Dropout)         (None, 412)               0         
_________________________________________________________________
dense_44 (Dense)             (None, 231)               95403     
_________________________________________________________________
dropout_14 (Dropout)         (None, 231)               0         
_________________________________________________________________
dense_45 (Dense)             (None, 112)               25984     
_________________________________________________________________
dropout_15 (Dropout)         (None, 112)               0         
_________________________________________________________________
dense_46 (Dense)             (None, 10)                1130      
=================================================================
Total params: 1,249,169
Trainable params: 1,249,169
Non-trainable params: 0
_________________________________________________________________
```

In [67]:

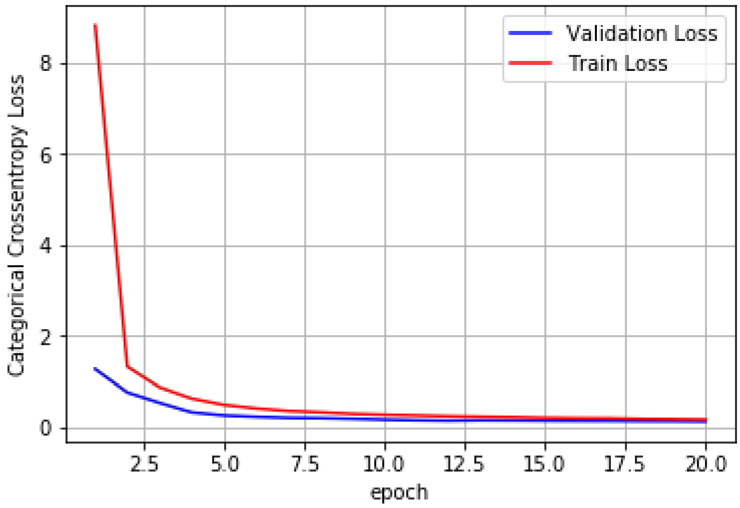
```
model_relu.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accurac
y'])
history = model_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verb
ose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 17s 282us/step - loss: 8.82
55 - acc: 0.2601 - val_loss: 1.2662 - val_acc: 0.6625
Epoch 2/20
60000/60000 [==============================] - 14s 240us/step - loss: 1.32
11 - acc: 0.5528 - val_loss: 0.7544 - val_acc: 0.7617
Epoch 3/20
60000/60000 [==============================] - 14s 241us/step - loss: 0.86
94 - acc: 0.7162 - val_loss: 0.5276 - val_acc: 0.8670
Epoch 4/20
60000/60000 [==============================] - 14s 240us/step - loss: 0.61
93 - acc: 0.8110 - val_loss: 0.3173 - val_acc: 0.9238
Epoch 5/20
60000/60000 [==============================] - 15s 243us/step - loss: 0.48
21 - acc: 0.8656 - val_loss: 0.2496 - val_acc: 0.9372
Epoch 6/20
60000/60000 [==============================] - 14s 241us/step - loss: 0.40
34 - acc: 0.8948 - val_loss: 0.2200 - val_acc: 0.9475
Epoch 7/20
60000/60000 [==============================] - 14s 240us/step - loss: 0.34
84 - acc: 0.9117 - val_loss: 0.1971 - val_acc: 0.9506
Epoch 8/20
60000/60000 [==============================] - 14s 240us/step - loss: 0.31
98 - acc: 0.9193 - val_loss: 0.1902 - val_acc: 0.9541
Epoch 9/20
60000/60000 [==============================] - 15s 243us/step - loss: 0.28
56 - acc: 0.9282 - val_loss: 0.1759 - val_acc: 0.9565
Epoch 10/20
60000/60000 [==============================] - 14s 241us/step - loss: 0.26
54 - acc: 0.9344 - val_loss: 0.1572 - val_acc: 0.9599
Epoch 11/20
60000/60000 [==============================] - 14s 240us/step - loss: 0.24
93 - acc: 0.9382 - val_loss: 0.1440 - val_acc: 0.9655
Epoch 12/20
60000/60000 [==============================] - 14s 241us/step - loss: 0.23
19 - acc: 0.9433 - val_loss: 0.1354 - val_acc: 0.9663
Epoch 13/20
60000/60000 [==============================] - 14s 241us/step - loss: 0.22
19 - acc: 0.9452 - val_loss: 0.1430 - val_acc: 0.9661
Epoch 14/20
60000/60000 [==============================] - 14s 242us/step - loss: 0.21
14 - acc: 0.9490 - val_loss: 0.1399 - val_acc: 0.9655
Epoch 15/20
60000/60000 [==============================] - 14s 241us/step - loss: 0.19
60 - acc: 0.9513 - val_loss: 0.1356 - val_acc: 0.9666
Epoch 16/20
60000/60000 [==============================] - 14s 241us/step - loss: 0.18
86 - acc: 0.9537 - val_loss: 0.1317 - val_acc: 0.9679
Epoch 17/20
60000/60000 [==============================] - 14s 241us/step - loss: 0.18
67 - acc: 0.9545 - val_loss: 0.1302 - val_acc: 0.9678
Epoch 18/20
60000/60000 [==============================] - 14s 241us/step - loss: 0.17
29 - acc: 0.9578 - val_loss: 0.1254 - val_acc: 0.9690
Epoch 19/20
60000/60000 [==============================] - 14s 240us/step - loss: 0.16
58 - acc: 0.9609 - val_loss: 0.1231 - val_acc: 0.9711
Epoch 20/20
60000/60000 [==============================] - 15s 244us/step - loss: 0.15
90 - acc: 0.9606 - val_loss: 0.1188 - val_acc: 0.9736
```

In [68]:

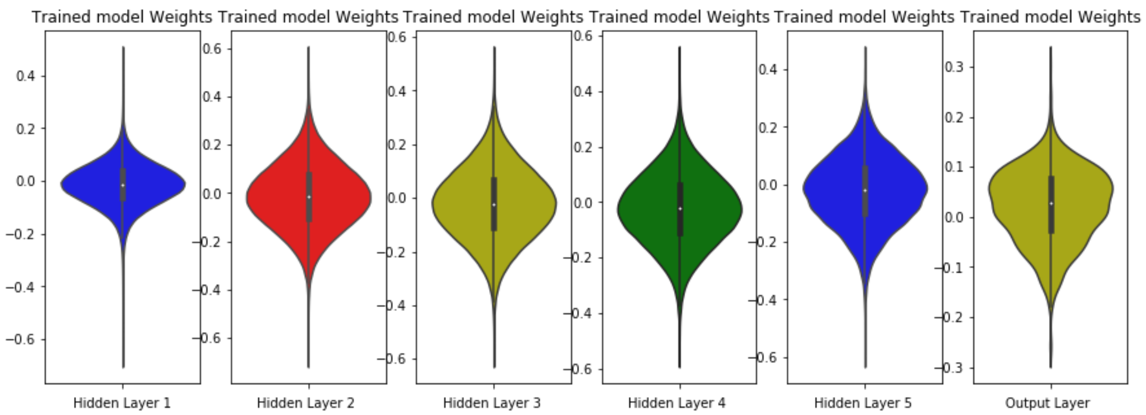
```
#Evualate your model with accuracy and plot of (NUmber of epoches VS train_and_val_los
s)
#Train accuracy
score = model_relu.evaluate(X_train, Y_train, verbose=0)
print('Train score:', score[0])
print('Train accuracy:', score[1]*100)
print('\n*********************** ********************\n')
#test accuracy
score = model_relu.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1]*100)
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')
# list of epoch numbers
x = list(range(1,nb_epoch+1))
# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ve
rbose=1, validation_data=(X_test, Y_test))
# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy
# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of ep
ochs
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Train score: 0.06144777707796311
Train accuracy: 98.44166666666668

*********************** ********************

Test score: 0.11881388411391526
Test accuracy: 97.36
```



In [70]:

```
w_after = model_relu.get_weights()
h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
h4_w = w_after[6].flatten().reshape(-1,1)
h5_w = w_after[8].flatten().reshape(-1,1)
out_w = w_after[10].flatten().reshape(-1,1)
fig = plt.figure(figsize=(15,5))
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='y')
plt.xlabel('Hidden Layer 3 ')

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

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



4. MLP + ReLU + adam + dropout + batch_normalization

In [71]:

```
# Multilayer perceptron
# https://arxiv.org/pdf/1707.09725.pdf#page=95
# for relu layers
# If we sample weights from a normal distribution N(0,σ) we satisfy this condition with
σ=√(2/(ni).
# h1 => σ=√(2/(fan_in) = 0.062 => N(0,σ) = N(0,0.062)
# h2 => σ=√(2/(fan_in) = 0.125 => N(0,σ) = N(0,0.125)
# out => σ=√(2/(fan_in+1) = 0.120 => N(0,σ) = N(0,0.120)
model_relu = Sequential()
model_relu.add(Dense(690, activation='relu', input_shape=(input_dim,), kernel_initializ
er=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(530, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
stddev=0.125, seed=None)) )
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(412, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
stddev=0.125, seed=None)) )
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(231, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
stddev=0.125, seed=None)) )
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(112, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
stddev=0.125, seed=None)) )
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(output_dim, activation='softmax'))
model_relu.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_47 (Dense)             (None, 690)               541650    
_________________________________________________________________
batch_normalization_16 (Batc (None, 690)               2760      
_________________________________________________________________
dropout_16 (Dropout)         (None, 690)               0         
_________________________________________________________________
dense_48 (Dense)             (None, 530)               366230    
_________________________________________________________________
batch_normalization_17 (Batc (None, 530)               2120      
_________________________________________________________________
dropout_17 (Dropout)         (None, 530)               0         
_________________________________________________________________
dense_49 (Dense)             (None, 412)               218772    
_________________________________________________________________
batch_normalization_18 (Batc (None, 412)               1648      
_________________________________________________________________
dropout_18 (Dropout)         (None, 412)               0         
_________________________________________________________________
dense_50 (Dense)             (None, 231)               95403     
_________________________________________________________________
batch_normalization_19 (Batc (None, 231)               924       
_________________________________________________________________
dropout_19 (Dropout)         (None, 231)               0         
_________________________________________________________________
dense_51 (Dense)             (None, 112)               25984     
_________________________________________________________________
batch_normalization_20 (Batc (None, 112)               448       
_________________________________________________________________
dropout_20 (Dropout)         (None, 112)               0         
_________________________________________________________________
dense_52 (Dense)             (None, 10)                1130      
=================================================================
Total params: 1,257,069
Trainable params: 1,253,119
Non-trainable params: 3,950
_________________________________________________________________
```

In [72]:

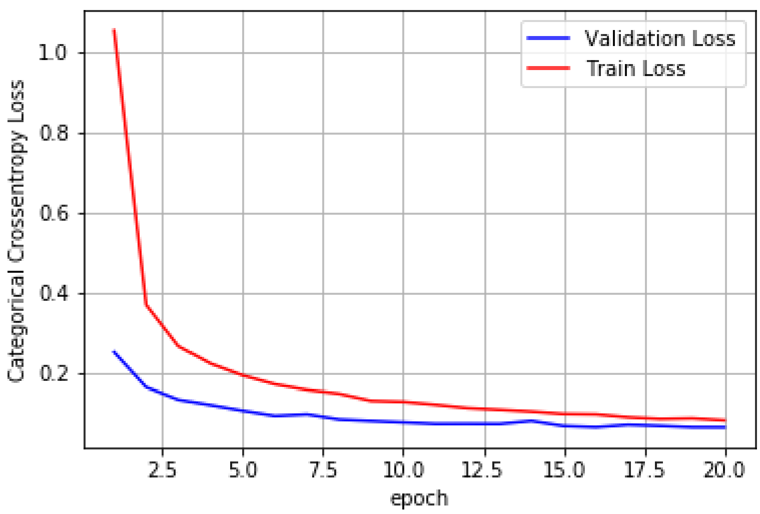
```
model_relu.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accurac
y'])
history = model_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verb
ose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 26s 428us/step - loss: 1.05
27 - acc: 0.6697 - val_loss: 0.2500 - val_acc: 0.9261
Epoch 2/20
60000/60000 [==============================] - 21s 357us/step - loss: 0.36
79 - acc: 0.8910 - val_loss: 0.1626 - val_acc: 0.9519
Epoch 3/20
60000/60000 [==============================] - 24s 406us/step - loss: 0.26
47 - acc: 0.9257 - val_loss: 0.1312 - val_acc: 0.9607
Epoch 4/20
60000/60000 [==============================] - 24s 404us/step - loss: 0.22
13 - acc: 0.9374 - val_loss: 0.1179 - val_acc: 0.9667
Epoch 5/20
60000/60000 [==============================] - 23s 385us/step - loss: 0.19
19 - acc: 0.9465 - val_loss: 0.1039 - val_acc: 0.9714
Epoch 6/20
60000/60000 [==============================] - 24s 407us/step - loss: 0.17
00 - acc: 0.9528 - val_loss: 0.0911 - val_acc: 0.9737
Epoch 7/20
60000/60000 [==============================] - 22s 375us/step - loss: 0.15
57 - acc: 0.9561 - val_loss: 0.0951 - val_acc: 0.9737
Epoch 8/20
60000/60000 [==============================] - 21s 346us/step - loss: 0.14
61 - acc: 0.9586 - val_loss: 0.0824 - val_acc: 0.9764
Epoch 9/20
60000/60000 [==============================] - 21s 353us/step - loss: 0.12
85 - acc: 0.9640 - val_loss: 0.0786 - val_acc: 0.9789
Epoch 10/20
60000/60000 [==============================] - 21s 356us/step - loss: 0.12
62 - acc: 0.9649 - val_loss: 0.0752 - val_acc: 0.9795
Epoch 11/20
60000/60000 [==============================] - 23s 391us/step - loss: 0.11
90 - acc: 0.9670 - val_loss: 0.0717 - val_acc: 0.9807
Epoch 12/20
60000/60000 [==============================] - 25s 414us/step - loss: 0.11
07 - acc: 0.9692 - val_loss: 0.0718 - val_acc: 0.9806
Epoch 13/20
60000/60000 [==============================] - 20s 333us/step - loss: 0.10
66 - acc: 0.9704 - val_loss: 0.0714 - val_acc: 0.9812
Epoch 14/20
60000/60000 [==============================] - 22s 365us/step - loss: 0.10
21 - acc: 0.9717 - val_loss: 0.0789 - val_acc: 0.9791
Epoch 15/20
60000/60000 [==============================] - 20s 341us/step - loss: 0.09
61 - acc: 0.9729 - val_loss: 0.0664 - val_acc: 0.9830
Epoch 16/20
60000/60000 [==============================] - 25s 412us/step - loss: 0.09
52 - acc: 0.9736 - val_loss: 0.0634 - val_acc: 0.9838
Epoch 17/20
60000/60000 [==============================] - 24s 398us/step - loss: 0.08
79 - acc: 0.9755 - val_loss: 0.0694 - val_acc: 0.9835
Epoch 18/20
60000/60000 [==============================] - 20s 335us/step - loss: 0.08
39 - acc: 0.9762 - val_loss: 0.0666 - val_acc: 0.9824
Epoch 19/20
60000/60000 [==============================] - 20s 337us/step - loss: 0.08
53 - acc: 0.9760 - val_loss: 0.0633 - val_acc: 0.9829
Epoch 20/20
60000/60000 [==============================] - 20s 341us/step - loss: 0.08
04 - acc: 0.9769 - val_loss: 0.0630 - val_acc: 0.9830
```

In [73]:

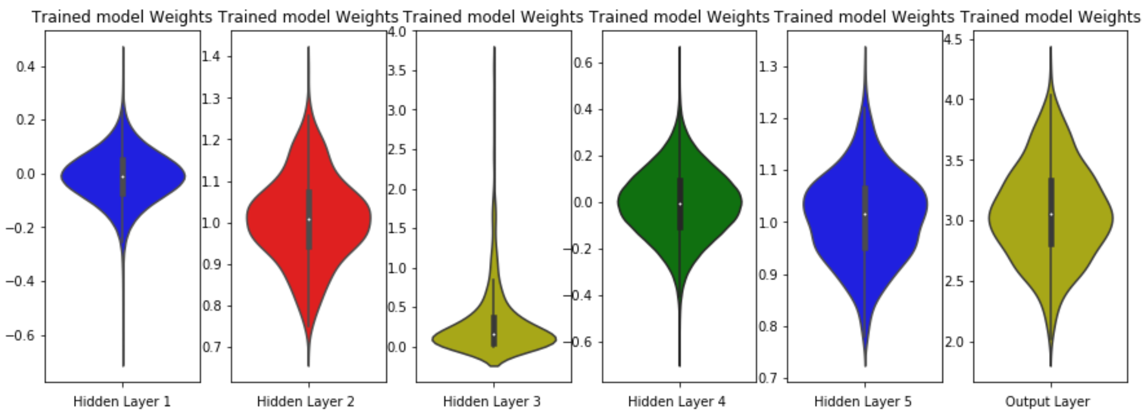
```
#Evualate your model with accuracy and plot of (NUmber of epoches VS train_and_val_los
s)
#Train accuracy
score = model_relu.evaluate(X_train, Y_train, verbose=0)
print('Train score:', score[0])
print('Train accuracy:', score[1]*100)
print('\n*********************** ********************\n')
#test accuracy
score = model_relu.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1]*100)
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')
# list of epoch numbers
x = list(range(1,nb_epoch+1))
# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ve
rbose=1, validation_data=(X_test, Y_test))
# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy
# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of ep
ochs
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Train score: 0.019241652972002823
Train accuracy: 99.47833333333334

*********************** ********************

Test score: 0.06302054594261572
Test accuracy: 98.3
```



In [74]:

```
w_after = model_relu.get_weights()
h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
h4_w = w_after[6].flatten().reshape(-1,1)
h5_w = w_after[8].flatten().reshape(-1,1)
out_w = w_after[10].flatten().reshape(-1,1)
fig = plt.figure(figsize=(15,5))
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='y')
plt.xlabel('Hidden Layer 3 ')

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

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



CONCLUSION:

In [76]:

```
from prettytable import PrettyTable
tb = PrettyTable()
tb.field_names= ("Hidden Layers", "Model", "Accuracy")
tb.add_row(["2", "MLP + ADAM + RELU",97.82])
tb.add_row(["2", "MLP + ADAM + RELU + batch_normalization",98.2])
tb.add_row(["2", "MLP + ADAM + RELU + dropout",97.77])
tb.add_row(["2", "MLP + ADAM + RELU + dropout+ batch_normalization",98.19])
tb.add_row([" ", " "," "])
tb.add_row([" ", " "," "])
tb.field_names= ("Hidden Layers", "Model", "Accuracy")
tb.add_row(["3", "MLP + ADAM + RELU",97.99])
tb.add_row(["3", "MLP + ADAM + RELU + batch_normalization",98.03])
tb.add_row(["3", "MLP + ADAM + RELU + dropout",83.14])
tb.add_row(["3", "MLP + ADAM + RELU + dropout+ batch_normalization",98.21])
tb.add_row([" ", " "," "])
tb.add_row([" ", " "," "])
tb.field_names= ("Hidden Layers", "Model", "Accuracy")
tb.add_row(["5", "MLP + ADAM + RELU",98.02])
tb.add_row(["5", "MLP + ADAM + RELU + batch_normalization",98.34])
tb.add_row(["5", "MLP + ADAM + RELU + dropout",97.36])
tb.add_row(["5", "MLP + ADAM + RELU + dropout+ batch_normalization",98.3])
print(tb.get_string(titles = "MLP Models - Observations"))
```

Hidden Layers	Model	Accuracy
2	MLP + ADAM + RELU	97.82
2	MLP + ADAM + RELU + batch_normalization	98.2
2	MLP + ADAM + RELU + dropout	97.77
2	MLP + ADAM + RELU + dropout+ batch_normalization	98.19
3	MLP + ADAM + RELU	97.99
3	MLP + ADAM + RELU + batch_normalization	98.03
3	MLP + ADAM + RELU + dropout	83.14
3	MLP + ADAM + RELU + dropout+ batch_normalization	98.21
5	MLP + ADAM + RELU	98.02
5	MLP + ADAM + RELU + batch_normalization	98.34
5	MLP + ADAM + RELU + dropout	97.36
5	MLP + ADAM + RELU + dropout+ batch_normalization	98.3