Keras -- MLPs on MNIST

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

```
Using TensorFlow backend.
```

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

In [3]:
```
# the data, shuffled and split between train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

In [4]:
```
print("Number of training examples :", X_train.shape[0], "and each image is of
shape (%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 [5]:
```
# if you observe the input shape its 2 dimensional vector
# for each image we have a (28*28) vector
# we will convert the (28*28) vector into single dimensional vector of 1 * 784

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

In [6]:
```
# 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 [7]:
```
# An example data point
print(X_train[0])
```

```
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   3  18  18  18 126 136 175  26 166 255
 247 127   0   0   0   0   0   0   0   0   0   0   0   0  30  36  94 154
 170 253 253 253 253 253 225 172 253 242 195  64   0   0   0   0   0   0
   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251  93  82
  82  56  39   0   0   0   0   0   0   0   0   0   0   0   0  18 219 253
 253 253 253 253 198 182 247 241   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0  14   1 154 253  90   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0  11 190 253  70   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  35 241
 225 160 108   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0  81 240 253 253 119  25   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  45 186 253 253 150  27   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252 253 187
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0 249 253 249  64   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253
 253 207   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0  39 148 229 253 253 253 250 182   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253
 253 201  78   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  23  66 213 253 253 253 253 198  81   2   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0  18 171 219 253 253 253 253 195
  80   9   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  55 172 226 253 253 253 253 244 133  11   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0 136 253 253 253 212 135 132  16
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
```

In [8]:
```
# 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 th
e data
# X => (X - Xmin)/(Xmax-Xmin) = X/255

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

In [9]:
```
# example data point after normlizing
print(X_train[0])
```

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

```
 0.         0.         0.         0.         0.         0.04313725
 0.74509804 0.99215686 0.2745098  0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.1372549  0.94509804
 0.88235294 0.62745098 0.42352941 0.00392157 0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.31764706 0.94117647 0.99215686
 0.99215686 0.46666667 0.09803922 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.17647059 0.72941176 0.99215686 0.99215686
 0.58823529 0.10588235 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.0627451  0.36470588 0.98823529 0.99215686 0.73333333
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.97647059 0.99215686 0.97647059 0.25098039 0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.18039216 0.50980392 0.71764706 0.99215686
 0.99215686 0.81176471 0.00784314 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.15294118 0.58039216
 0.89803922 0.99215686 0.99215686 0.99215686 0.98039216 0.71372549
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.09411765 0.44705882 0.86666667 0.99215686 0.99215686 0.99215686
 0.99215686 0.78823529 0.30588235 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.09019608 0.25882353 0.83529412 0.99215686
 0.99215686 0.99215686 0.99215686 0.77647059 0.31764706 0.00784314
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.07058824 0.67058824
 0.85882353 0.99215686 0.99215686 0.99215686 0.99215686 0.76470588
 0.31372549 0.03529412 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.21568627 0.6745098  0.88627451 0.99215686 0.99215686 0.99215686
 0.99215686 0.95686275 0.52156863 0.04313725 0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.53333333 0.99215686
 0.99215686 0.99215686 0.83137255 0.52941176 0.51764706 0.0627451
```

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

In [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.]
```

Softmax classifier

In [11]:
```
# https://keras.io/getting-started/sequential-model-guide/

# The Sequential model is a linear stack of layers.
# you can create a Sequential model by passing a list of layer instances to th
e constructor:

# model = Sequential([
#     Dense(32, input_shape=(784,)),
#     Activation('relu'),
#     Dense(10),
#     Activation('softmax'),
# ])

# You can also simply add layers via the .add() method:

# model = Sequential()
# model.add(Dense(32, input_dim=784))
# model.add(Activation('relu'))

###

# https://keras.io/layers/core/

# keras.layers.Dense(units, activation=None, use_bias=True, kernel_initializer
='glorot_uniform',
# bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=None, ac
tivity_regularizer=None,
# kernel_constraint=None, bias_constraint=None)

# Dense implements the operation: output = activation(dot(input, kernel) + bia
s) where
# activation is the element-wise activation function passed as the activation
 argument,
# kernel is a weights matrix created by the layer, and
# bias is a bias vector created by the layer (only applicable if use_bias is T
rue).

# output = activation(dot(input, kernel) + bias)  => y = activation(WT. X + b)

####

# https://keras.io/activations/

# Activations can either be used through an Activation layer, or through the a
ctivation argument supported by all forward layers:

# from keras.layers import Activation, Dense

# model.add(Dense(64))
# model.add(Activation('tanh'))

# This is equivalent to:
# model.add(Dense(64, activation='tanh'))

# there are many activation functions ar available ex: tanh, relu, softmax
```

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

In [12]:
```
# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 20
```

In [13]:
```
# start building a model
model = Sequential()

# The model needs to know what input shape it should expect.
# For this reason, the first layer in a Sequential model
# (and only the first, because following layers can do automatic shape inferen
ce)
# needs to receive information about its input shape.
# you can use input_shape and input_dim to pass the shape of input

# output_dim represent the number of nodes need in that layer
# here we have 10 nodes

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

```
WARNING:tensorflow:From C:\Users\LENOVO\Anaconda3\lib\site-packages\keras\bac
kend\tensorflow_backend.py:66: The name tf.get_default_graph is deprecated. P
lease use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From C:\Users\LENOVO\Anaconda3\lib\site-packages\keras\bac
kend\tensorflow_backend.py:541: The name tf.placeholder is deprecated. Please
use tf.compat.v1.placeholder instead.

WARNING:tensorflow:From C:\Users\LENOVO\Anaconda3\lib\site-packages\keras\bac
kend\tensorflow_backend.py:4432: The name tf.random_uniform is deprecated. Pl
ease use tf.random.uniform instead.
```

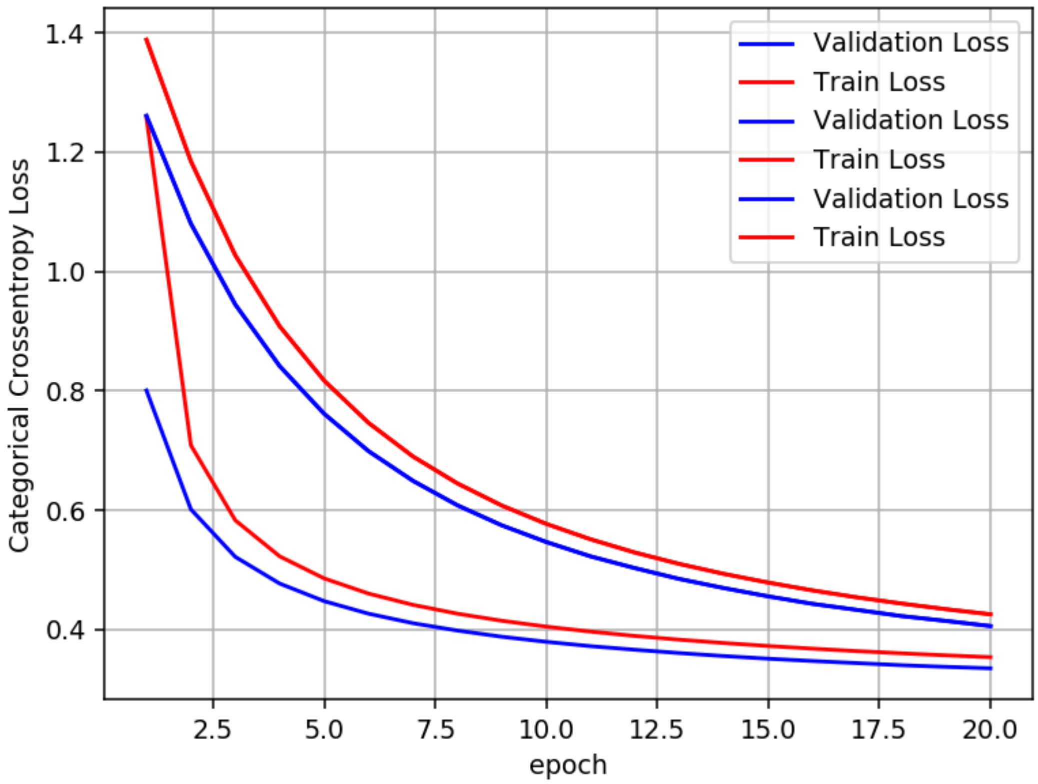
In [15]:
```
# Before training a model, you need to configure the learning process, which i
s done via the compile method

# It receives three arguments:
# An optimizer. This could be the string identifier of an existing optimizer ,
https://keras.io/optimizers/
# A loss function. This is the objective that the model will try to minimize.,
https://keras.io/losses/
# A list of metrics. For any classification problem you will want to set this
 to metrics=['accuracy'].  https://keras.io/metrics/


# Note: when using the categorical_crossentropy loss, your targets should be i
n categorical format
# (e.g. if you have 10 classes, the target for each sample should be a 10-dime
nsional vector that is all-zeros except
# for a 1 at the index corresponding to the class of the sample).

# that is why we converted out labels into vectors

model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accu
racy'])

# Keras models are trained on Numpy arrays of input data and labels.
# For training a model, you will typically use the  fit function

# fit(self, x=None, y=None, batch_size=None, epochs=1, verbose=1, callbacks=No
ne, validation_split=0.0,
# validation_data=None, shuffle=True, class_weight=None, sample_weight=None, i
nitial_epoch=0, steps_per_epoch=None,
# validation_steps=None)

# fit() function Trains the model for a fixed number of epochs (iterations on
 a dataset).

# it returns A History object. Its History.history attribute is a record of tr
aining loss values and
# metrics values at successive epochs, as well as validation loss values and v
alidation metrics values (if applicable).

# https://github.com/openai/baselines/issues/20

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 3s 46us/step - loss: 1.2577 -
acc: 0.7112 - val_loss: 0.8008 - val_acc: 0.8329
Epoch 2/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.7090 -
acc: 0.8427 - val_loss: 0.6017 - val_acc: 0.8620
Epoch 3/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.5833 -
acc: 0.8603 - val_loss: 0.5221 - val_acc: 0.8739
Epoch 4/20
60000/60000 [==============================] - 2s 30us/step - loss: 0.5229 -
acc: 0.8693 - val_loss: 0.4773 - val_acc: 0.8812
Epoch 5/20
60000/60000 [==============================] - 2s 37us/step - loss: 0.4860 -
acc: 0.8763 - val_loss: 0.4479 - val_acc: 0.8853
Epoch 6/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.4606 -
acc: 0.8808 - val_loss: 0.4269 - val_acc: 0.8904
Epoch 7/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.4417 -
acc: 0.8843 - val_loss: 0.4110 - val_acc: 0.8930
Epoch 8/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.4270 -
acc: 0.8866 - val_loss: 0.3986 - val_acc: 0.8957
Epoch 9/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.4151 -
acc: 0.8890 - val_loss: 0.3883 - val_acc: 0.8976
Epoch 10/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.4053 -
acc: 0.8912 - val_loss: 0.3798 - val_acc: 0.8986
Epoch 11/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.3969 -
acc: 0.8930 - val_loss: 0.3725 - val_acc: 0.9001
Epoch 12/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.3896 -
acc: 0.8947 - val_loss: 0.3665 - val_acc: 0.9014
Epoch 13/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.3834 -
acc: 0.8960 - val_loss: 0.3609 - val_acc: 0.9023
Epoch 14/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.3778 -
acc: 0.8975 - val_loss: 0.3559 - val_acc: 0.9033
Epoch 15/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.3728 -
acc: 0.8987 - val_loss: 0.3515 - val_acc: 0.9041
Epoch 16/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.3683 -
acc: 0.8996 - val_loss: 0.3477 - val_acc: 0.9050
Epoch 17/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.3643 -
acc: 0.9004 - val_loss: 0.3441 - val_acc: 0.9045
Epoch 18/20
60000/60000 [==============================] - 2s 30us/step - loss: 0.3606 -
acc: 0.9015 - val_loss: 0.3406 - val_acc: 0.9062
Epoch 19/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.3572 -
```

```
acc: 0.9018 - val_loss: 0.3378 - val_acc: 0.9064
Epoch 20/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.3541 -
acc: 0.9026 - val_loss: 0.3354 - val_acc: 0.9069
```

In [16]:
```
score = model.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

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

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to num
ber of epochs

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

```
Test score: 0.33542446033954620
Test accuracy: 0.9069
```



MLP + Sigmoid activation + SGDOptimizer

In [65]:
```
from tensorflow.keras.callbacks import TensorBoard
import time

NAME = 'cc-{}'.format(int(time.time()))
tensorboardd = TensorBoard(log_dir='logss\{}'.format(NAME))
```

In [66]:
```
# Multilayer perceptron

model_sigmoid = Sequential()
model_sigmoid.add(Dense(512, activation='sigmoid', input_shape=(input_dim,)))
model_sigmoid.add(Dense(128, activation='sigmoid'))
model_sigmoid.add(Dense(output_dim, activation='softmax'))

model_sigmoid.summary()
```

```
Model: "sequential_27"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_74 (Dense)             (None, 512)               401920    
_________________________________________________________________
dense_75 (Dense)             (None, 128)               65664     
_________________________________________________________________
dense_76 (Dense)             (None, 10)                1290      
=================================================================
Total params: 468,874
Trainable params: 468,874
Non-trainable params: 0
_________________________________________________________________
```

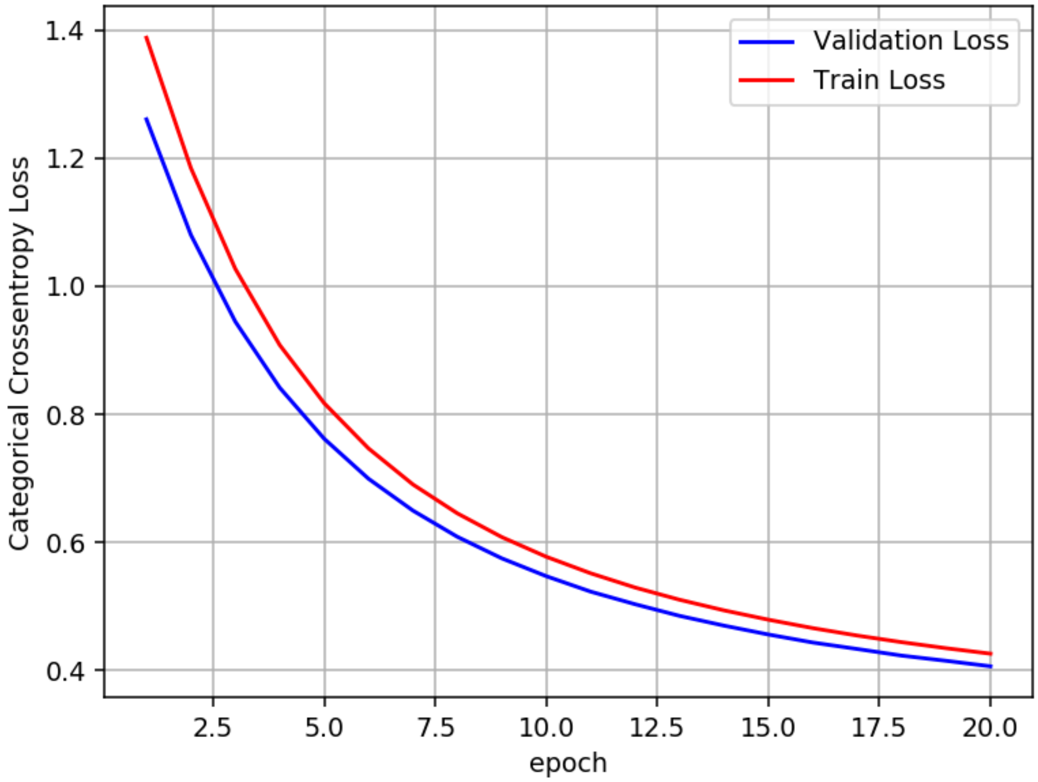
In [67]:
```
model_sigmoid.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_sigmoid.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test),callbacks=[tensorboardd])
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 88us/step - loss: 2.2681 -
acc: 0.2242 - val_loss: 2.2209 - val_acc: 0.4627
Epoch 2/20
60000/60000 [==============================] - 3s 52us/step - loss: 2.1769 -
acc: 0.4827 - val_loss: 2.1215 - val_acc: 0.4781
Epoch 3/20
60000/60000 [==============================] - 3s 52us/step - loss: 2.0601 -
acc: 0.5842 - val_loss: 1.9797 - val_acc: 0.6648
Epoch 4/20
60000/60000 [==============================] - 3s 55us/step - loss: 1.8937 -
acc: 0.6380 - val_loss: 1.7819 - val_acc: 0.6533
Epoch 5/20
60000/60000 [==============================] - 3s 50us/step - loss: 1.6773 -
acc: 0.6711 - val_loss: 1.5480 - val_acc: 0.7032
Epoch 6/20
60000/60000 [==============================] - 3s 48us/step - loss: 1.4451 -
acc: 0.7103 - val_loss: 1.3199 - val_acc: 0.7293
Epoch 7/20
60000/60000 [==============================] - 4s 60us/step - loss: 1.2382 -
acc: 0.7431 - val_loss: 1.1337 - val_acc: 0.7753
Epoch 8/20
60000/60000 [==============================] - 3s 57us/step - loss: 1.0737 -
acc: 0.7725 - val_loss: 0.9899 - val_acc: 0.7895
Epoch 9/20
60000/60000 [==============================] - 3s 56us/step - loss: 0.9471 -
acc: 0.7944 - val_loss: 0.8788 - val_acc: 0.8038
Epoch 10/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.8494 -
acc: 0.8116 - val_loss: 0.7934 - val_acc: 0.8197
Epoch 11/20
60000/60000 [==============================] - 4s 59us/step - loss: 0.7727 -
acc: 0.8237 - val_loss: 0.7241 - val_acc: 0.8332
Epoch 12/20
60000/60000 [==============================] - 3s 56us/step - loss: 0.7112 -
acc: 0.8347 - val_loss: 0.6692 - val_acc: 0.8440
Epoch 13/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.6613 -
acc: 0.8435 - val_loss: 0.6243 - val_acc: 0.8511
Epoch 14/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.6203 -
acc: 0.8504 - val_loss: 0.5875 - val_acc: 0.8590
Epoch 15/20
60000/60000 [==============================] - 4s 59us/step - loss: 0.5863 -
acc: 0.8559 - val_loss: 0.5562 - val_acc: 0.8630
Epoch 16/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.5578 -
acc: 0.8606 - val_loss: 0.5294 - val_acc: 0.8684
Epoch 17/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.5336 -
acc: 0.8650 - val_loss: 0.5081 - val_acc: 0.8699
Epoch 18/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.5129 -
acc: 0.8686 - val_loss: 0.4886 - val_acc: 0.8739
Epoch 19/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.4949 -
```

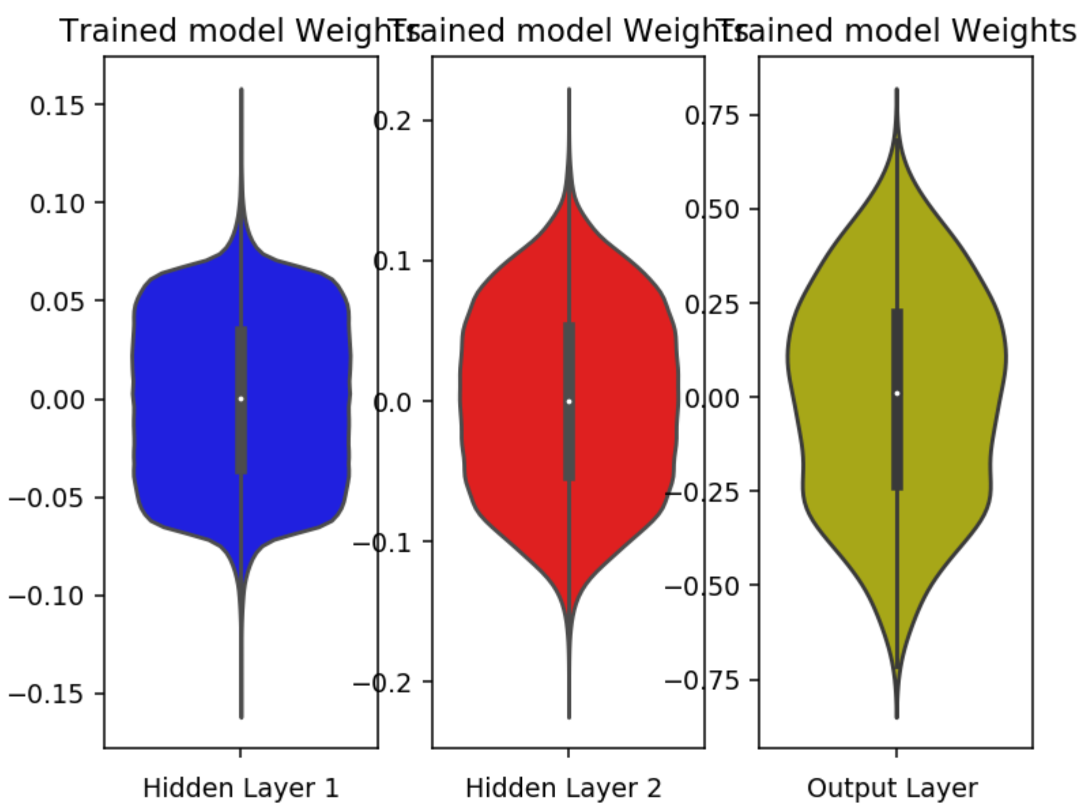
```
acc: 0.8715 - val_loss: 0.4716 - val_acc: 0.8782
Epoch 20/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.4793 -
acc: 0.8750 - val_loss: 0.4565 - val_acc: 0.8811
```

In [23]:
```
score = model_sigmoid.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

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

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to num
ber of epochs

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

```
Test score: 0.40619140058923721
Test accuracy: 0.8895
```



In [30]:
```
w_after = model_sigmoid.get_weights()

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


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

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

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



MLP + Sigmoid activation + ADAM

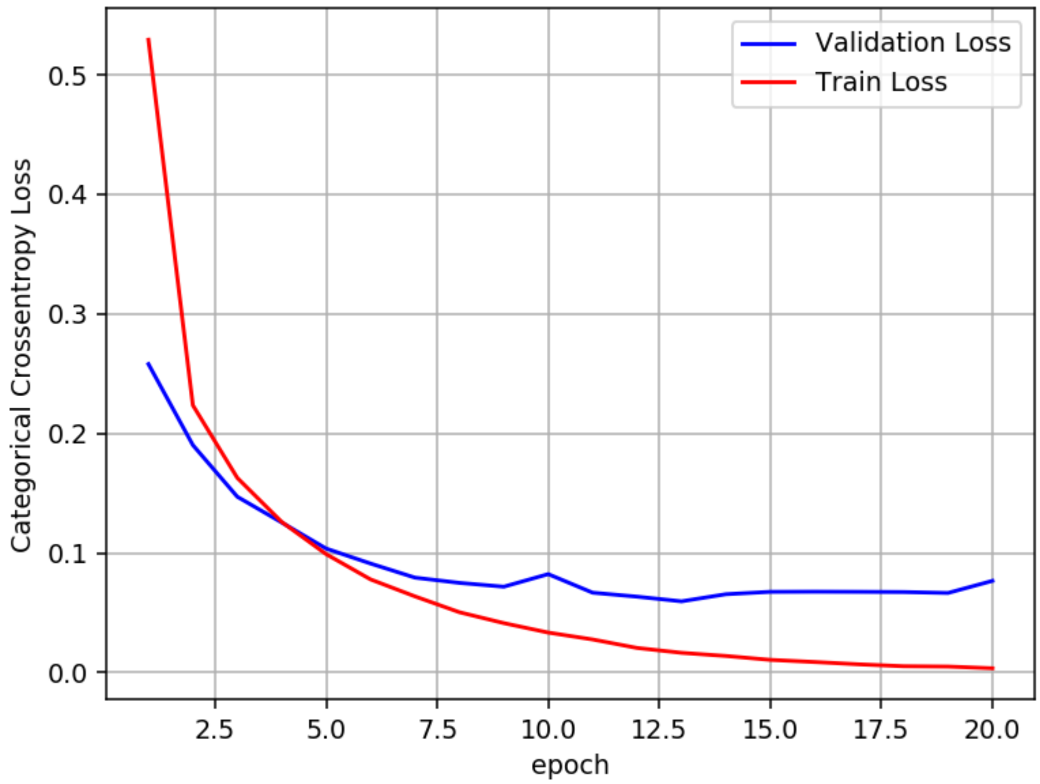
In [34]:
```
model_sigmoid = Sequential()
model_sigmoid.add(Dense(512, activation='sigmoid', input_shape=(input_dim,)))
model_sigmoid.add(Dense(128, activation='sigmoid'))
model_sigmoid.add(Dense(output_dim, activation='softmax'))

model_sigmoid.summary()

model_sigmoid.compile(optimizer='adam', loss='categorical_crossentropy', metri
cs=['accuracy'])

history = model_sigmoid.fit(X_train, Y_train, batch_size=batch_size, epochs=nb
_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_11 (Dense)             (None, 512)               401920    
_________________________________________________________________
dense_12 (Dense)             (None, 128)               65664     
_________________________________________________________________
dense_13 (Dense)             (None, 10)                1290      
=================================================================
Total params: 468,874
Trainable params: 468,874
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 4s 61us/step - loss: 0.5294 -
acc: 0.8621 - val_loss: 0.2580 - val_acc: 0.9256
Epoch 2/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.2236 -
acc: 0.9338 - val_loss: 0.1903 - val_acc: 0.9429
Epoch 3/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.1629 -
acc: 0.9515 - val_loss: 0.1471 - val_acc: 0.9563
Epoch 4/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.1260 -
acc: 0.9621 - val_loss: 0.1255 - val_acc: 0.9612
Epoch 5/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0990 -
acc: 0.9716 - val_loss: 0.1037 - val_acc: 0.9679
Epoch 6/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0779 -
acc: 0.9768 - val_loss: 0.0912 - val_acc: 0.9722
Epoch 7/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0637 -
acc: 0.9810 - val_loss: 0.0794 - val_acc: 0.9758
Epoch 8/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0504 -
acc: 0.9851 - val_loss: 0.0750 - val_acc: 0.9769
Epoch 9/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0413 -
acc: 0.9878 - val_loss: 0.0718 - val_acc: 0.9781
Epoch 10/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0334 -
acc: 0.9907 - val_loss: 0.0823 - val_acc: 0.9730
Epoch 11/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0277 -
acc: 0.9922 - val_loss: 0.0668 - val_acc: 0.9788
Epoch 12/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0206 -
acc: 0.9946 - val_loss: 0.0635 - val_acc: 0.9805
Epoch 13/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0165 -
acc: 0.9961 - val_loss: 0.0596 - val_acc: 0.9825
Epoch 14/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0139 -
acc: 0.9966 - val_loss: 0.0655 - val_acc: 0.9821
```

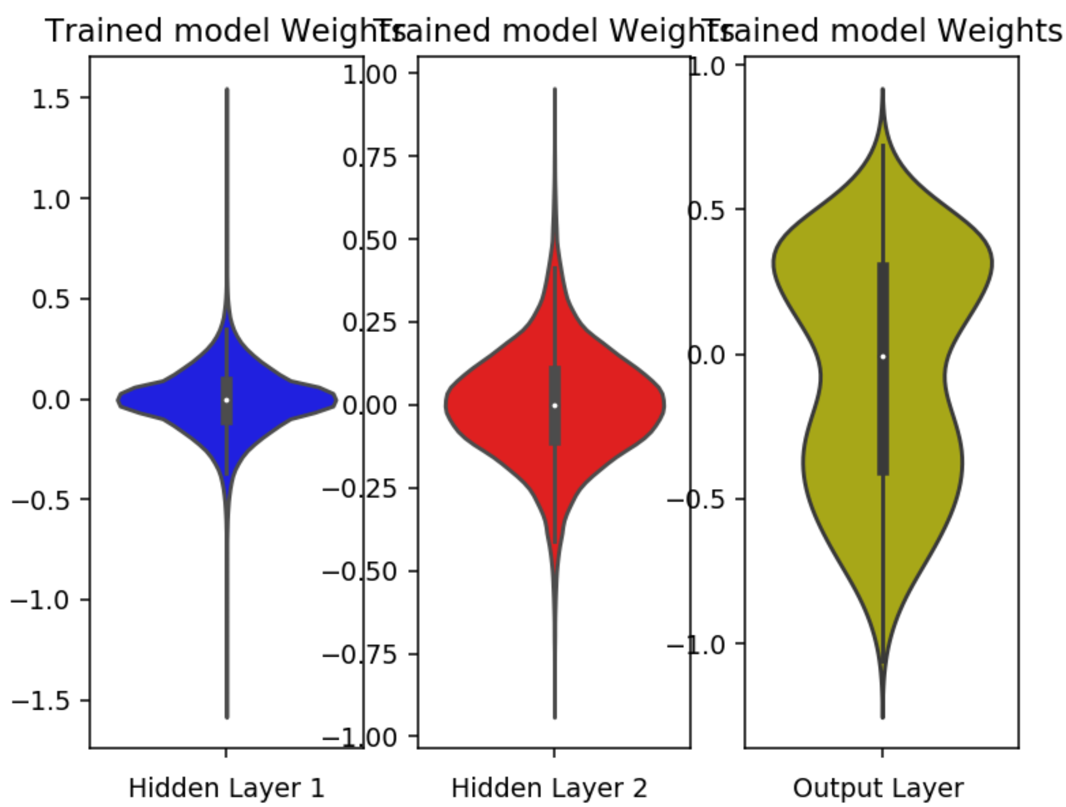
```
Epoch 15/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0106 -
acc: 0.9975 - val_loss: 0.0675 - val_acc: 0.9810
Epoch 16/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0088 -
acc: 0.9980 - val_loss: 0.0676 - val_acc: 0.9809
Epoch 17/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0068 -
acc: 0.9986 - val_loss: 0.0675 - val_acc: 0.9810
Epoch 18/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0053 -
acc: 0.9989 - val_loss: 0.0673 - val_acc: 0.9828
Epoch 19/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0050 -
acc: 0.9990 - val_loss: 0.0666 - val_acc: 0.9824
Epoch 20/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0035 -
acc: 0.9993 - val_loss: 0.0767 - val_acc: 0.9802
```

In [35]:
```
score = model_sigmoid.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

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

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to num
ber of epochs

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

```
Test score: 0.07668884656769806
Test accuracy: 0.9802
```



In [36]:
```
w_after = model_sigmoid.get_weights()

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


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

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

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



MLP + ReLU +SGD

In [37]:
```
# 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 condi
tion 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(512, activation='relu', input_shape=(input_dim,), kernel_
initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_relu.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(m
ean=0.0, stddev=0.125, seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))

model_relu.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_14 (Dense)             (None, 512)               401920    
_________________________________________________________________
dense_15 (Dense)             (None, 128)               65664     
_________________________________________________________________
dense_16 (Dense)             (None, 10)                1290      
=================================================================
Total params: 468,874
Trainable params: 468,874
Non-trainable params: 0
_________________________________________________________________
```

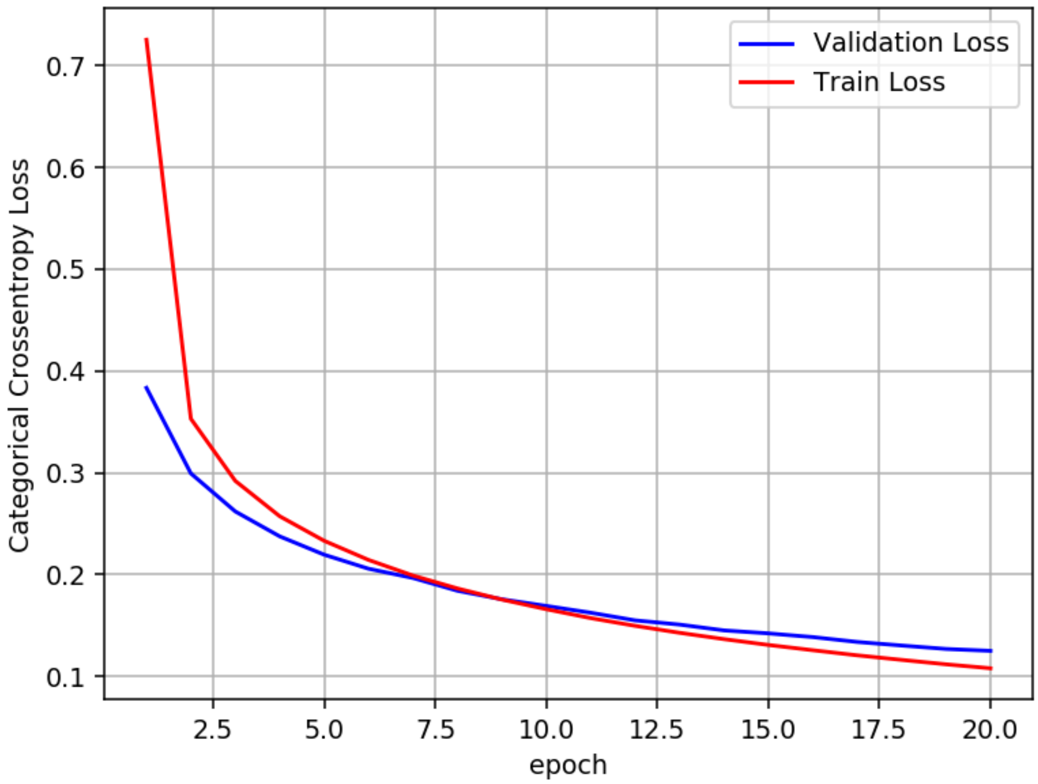
In [38]:
```
model_relu.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=[
'accuracy'])

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.7253 -
acc: 0.7946 - val_loss: 0.3834 - val_acc: 0.8899
Epoch 2/20
60000/60000 [==============================] - 2s 42us/step - loss: 0.3531 -
acc: 0.8999 - val_loss: 0.2994 - val_acc: 0.9154
Epoch 3/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.2921 -
acc: 0.9172 - val_loss: 0.2621 - val_acc: 0.9251
Epoch 4/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.2574 -
acc: 0.9267 - val_loss: 0.2376 - val_acc: 0.9320
Epoch 5/20
60000/60000 [==============================] - 3s 46us/step - loss: 0.2333 -
acc: 0.9332 - val_loss: 0.2196 - val_acc: 0.9378
Epoch 6/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.2145 -
acc: 0.9390 - val_loss: 0.2058 - val_acc: 0.9402
Epoch 7/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.1991 -
acc: 0.9428 - val_loss: 0.1967 - val_acc: 0.9454
Epoch 8/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.1865 -
acc: 0.9469 - val_loss: 0.1842 - val_acc: 0.9476
Epoch 9/20
60000/60000 [==============================] - 2s 42us/step - loss: 0.1756 -
acc: 0.9501 - val_loss: 0.1758 - val_acc: 0.9499
Epoch 10/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.1661 -
acc: 0.9528 - val_loss: 0.1693 - val_acc: 0.9515
Epoch 11/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.1573 -
acc: 0.9555 - val_loss: 0.1626 - val_acc: 0.9534
Epoch 12/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.1498 -
acc: 0.9585 - val_loss: 0.1551 - val_acc: 0.9552
Epoch 13/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.1429 -
acc: 0.9604 - val_loss: 0.1510 - val_acc: 0.9564
Epoch 14/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.1367 -
acc: 0.9622 - val_loss: 0.1453 - val_acc: 0.9575
Epoch 15/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.1310 -
acc: 0.9638 - val_loss: 0.1423 - val_acc: 0.9587
Epoch 16/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.1258 -
acc: 0.9652 - val_loss: 0.1387 - val_acc: 0.9593
Epoch 17/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.1208 -
acc: 0.9667 - val_loss: 0.1339 - val_acc: 0.9605
Epoch 18/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.1163 -
acc: 0.9676 - val_loss: 0.1304 - val_acc: 0.9630
Epoch 19/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.1120 -
```

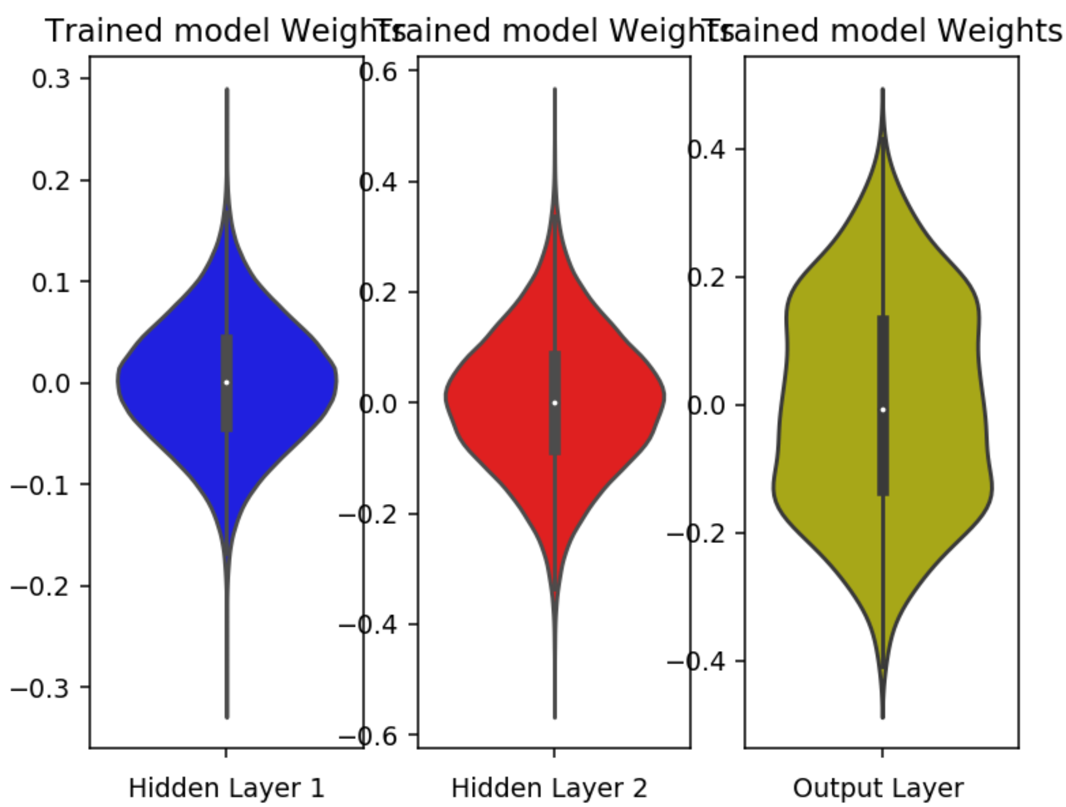
```
acc: 0.9693 - val_loss: 0.1270 - val_acc: 0.9628
Epoch 20/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.1081 -
acc: 0.9701 - val_loss: 0.1252 - val_acc: 0.9640
```

In [39]:
```
score = model_relu.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

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

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to num
ber of epochs

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

```
Test score: 0.12520071088634432
Test accuracy: 0.964
```



In [40]:
```
w_after = model_relu.get_weights()

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


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

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

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



MLP + ReLU + ADAM

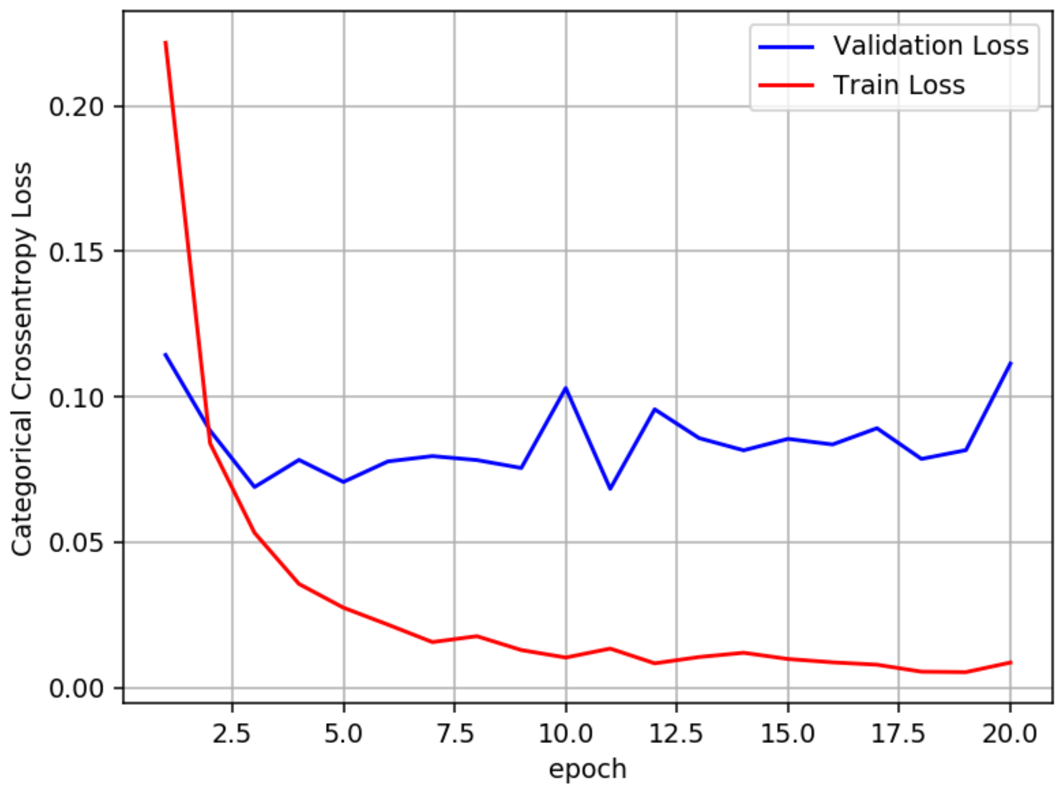
In [41]:
```
model_relu = Sequential()
model_relu.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_
initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_relu.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(m
ean=0.0, stddev=0.125, seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

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

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_17 (Dense)             (None, 512)               401920    
_________________________________________________________________
dense_18 (Dense)             (None, 128)               65664     
_________________________________________________________________
dense_19 (Dense)             (None, 10)                1290      
=================================================================
Total params: 468,874
Trainable params: 468,874
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.2218 - acc: 0.9331 - val_loss: 0.1144 - val_acc: 0.9651
Epoch 2/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0840 - acc: 0.9747 - val_loss: 0.0884 - val_acc: 0.9718
Epoch 3/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0532 - acc: 0.9840 - val_loss: 0.0689 - val_acc: 0.9794
Epoch 4/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.0356 - acc: 0.9887 - val_loss: 0.0783 - val_acc: 0.9771
Epoch 5/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.0274 - acc: 0.9913 - val_loss: 0.0707 - val_acc: 0.9791
Epoch 6/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0216 - acc: 0.9935 - val_loss: 0.0777 - val_acc: 0.9782
Epoch 7/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.0156 - acc: 0.9952 - val_loss: 0.0796 - val_acc: 0.9752
Epoch 8/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.0176 - acc: 0.9939 - val_loss: 0.0782 - val_acc: 0.9790
Epoch 9/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.0128 - acc: 0.9959 - val_loss: 0.0755 - val_acc: 0.9794
Epoch 10/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.0103 - acc: 0.9964 - val_loss: 0.1030 - val_acc: 0.9755
Epoch 11/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.0134 - acc: 0.9954 - val_loss: 0.0683 - val_acc: 0.9838
Epoch 12/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.0083 - acc: 0.9974 - val_loss: 0.0957 - val_acc: 0.9777
Epoch 13/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.0104 - acc: 0.9964 - val_loss: 0.0858 - val_acc: 0.9791
Epoch 14/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.0119 -
```

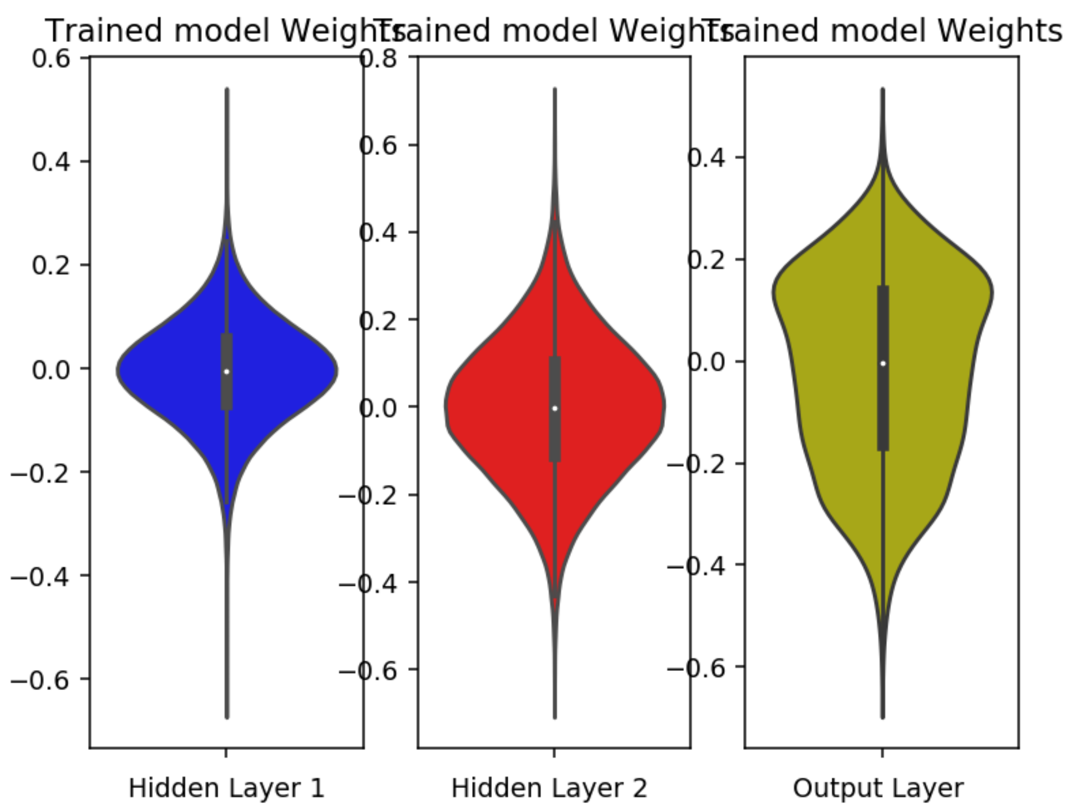
```
acc: 0.9963 - val_loss: 0.0816 - val_acc: 0.9799
Epoch 15/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.0098 -
acc: 0.9968 - val_loss: 0.0855 - val_acc: 0.9807
Epoch 16/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.0086 -
acc: 0.9972 - val_loss: 0.0836 - val_acc: 0.9819
Epoch 17/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.0078 -
acc: 0.9975 - val_loss: 0.0892 - val_acc: 0.9819
Epoch 18/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.0054 -
acc: 0.9984 - val_loss: 0.0786 - val_acc: 0.9831
Epoch 19/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.0052 -
acc: 0.9984 - val_loss: 0.0816 - val_acc: 0.9828
Epoch 20/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.0086 -
acc: 0.9972 - val_loss: 0.1114 - val_acc: 0.9768
```

In [42]:
```
score = model_relu.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

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

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to num
ber of epochs


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

```
Test score: 0.11140352230805939
Test accuracy: 0.9768
```



In [43]:
```
w_after = model_relu.get_weights()

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


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

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

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



MLP + Batch-Norm on hidden Layers + AdamOptimizer </2>

In [15]:
```
# Multilayer perceptron

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

from keras.layers.normalization import BatchNormalization

model_batch = Sequential()

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

model_batch.add(Dense(128, activation='sigmoid', kernel_initializer=RandomNorm
al(mean=0.0, stddev=0.55, seed=None)) )
model_batch.add(BatchNormalization())

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


model_batch.summary()
```

```
WARNING:tensorflow:From C:\Users\LENOVO\Anaconda3\lib\site-packages\keras\bac
kend\tensorflow_backend.py:148: The name tf.placeholder_with_default is depre
cated. Please use tf.compat.v1.placeholder_with_default instead.

WARNING:tensorflow:From C:\Users\LENOVO\Anaconda3\lib\site-packages\keras\bac
kend\tensorflow_backend.py:4432: The name tf.random_uniform is deprecated. Pl
ease use tf.random.uniform instead.

Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_2 (Dense)              (None, 512)               401920    
_________________________________________________________________
batch_normalization_1 (Batch (None, 512)               2048      
_________________________________________________________________
dense_3 (Dense)              (None, 128)               65664     
_________________________________________________________________
batch_normalization_2 (Batch (None, 128)               512       
_________________________________________________________________
dense_4 (Dense)              (None, 10)                1290      
=================================================================
Total params: 471,434
Trainable params: 470,154
Non-trainable params: 1,280
_________________________________________________________________
```

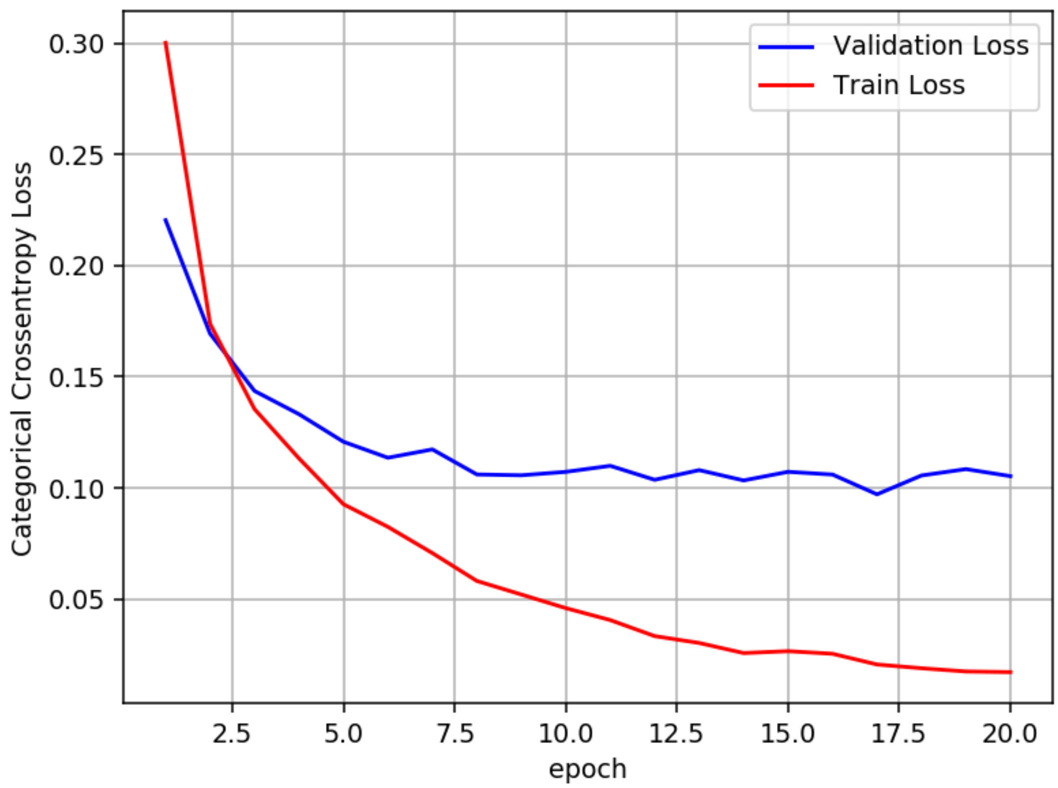
In [45]:
```
model_batch.compile(optimizer='adam', loss='categorical_crossentropy', metrics
=['accuracy'])

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 110us/step - loss: 0.3003 -
acc: 0.9115 - val_loss: 0.2204 - val_acc: 0.9340
Epoch 2/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.1738 -
acc: 0.9486 - val_loss: 0.1694 - val_acc: 0.9498
Epoch 3/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.1354 -
acc: 0.9598 - val_loss: 0.1436 - val_acc: 0.9553
Epoch 4/20
60000/60000 [==============================] - 5s 87us/step - loss: 0.1132 -
acc: 0.9667 - val_loss: 0.1332 - val_acc: 0.9594
Epoch 5/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.0926 -
acc: 0.9716 - val_loss: 0.1207 - val_acc: 0.9634
Epoch 6/20
60000/60000 [==============================] - 5s 80us/step - loss: 0.0824 -
acc: 0.9748 - val_loss: 0.1135 - val_acc: 0.9646
Epoch 7/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.0706 -
acc: 0.9781 - val_loss: 0.1173 - val_acc: 0.9641
Epoch 8/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.0581 -
acc: 0.9819 - val_loss: 0.1060 - val_acc: 0.9674
Epoch 9/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.0520 -
acc: 0.9839 - val_loss: 0.1057 - val_acc: 0.9685
Epoch 10/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.0459 -
acc: 0.9857 - val_loss: 0.1072 - val_acc: 0.9688
Epoch 11/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.0405 -
acc: 0.9870 - val_loss: 0.1099 - val_acc: 0.9688
Epoch 12/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.0333 -
acc: 0.9896 - val_loss: 0.1036 - val_acc: 0.9705
Epoch 13/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.0302 -
acc: 0.9908 - val_loss: 0.1080 - val_acc: 0.9698
Epoch 14/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.0256 -
acc: 0.9919 - val_loss: 0.1033 - val_acc: 0.9712
Epoch 15/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.0265 -
acc: 0.9913 - val_loss: 0.1072 - val_acc: 0.9708
Epoch 16/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.0253 -
acc: 0.9919 - val_loss: 0.1060 - val_acc: 0.9721
Epoch 17/20
60000/60000 [==============================] - 5s 80us/step - loss: 0.0205 -
acc: 0.9935 - val_loss: 0.0971 - val_acc: 0.9732
Epoch 18/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.0188 -
acc: 0.9937 - val_loss: 0.1056 - val_acc: 0.9730
Epoch 19/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.0174 -
```

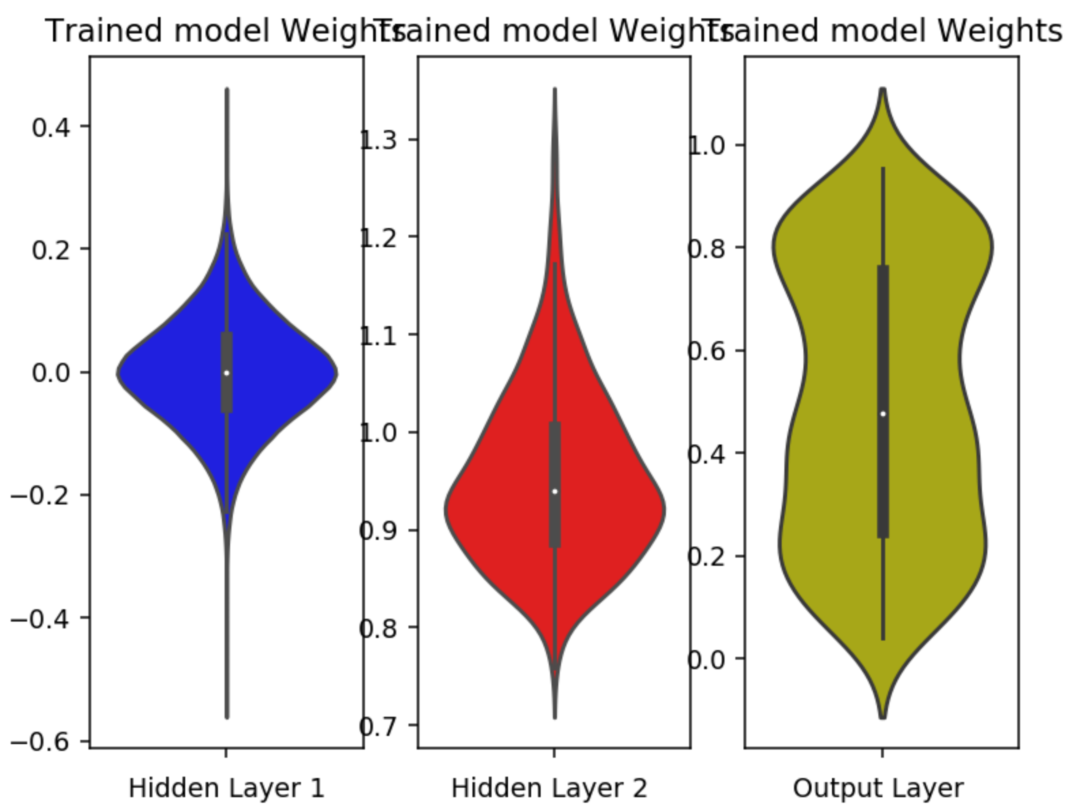
```
acc: 0.9942 - val_loss: 0.1084 - val_acc: 0.9710
Epoch 20/20
60000/60000 [==============================] - 5s 79us/step - loss: 0.0171 -
acc: 0.9944 - val_loss: 0.1053 - val_acc: 0.9735
```

In [46]:
```
score = model_batch.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

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

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to num
ber of epochs

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

```
Test score: 0.10525882752165198
Test accuracy: 0.9735
```



In [47]:
```
w_after = model_batch.get_weights()

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


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()
```



5. MLP + Dropout + AdamOptimizer

In [16]:
```
# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormal
ization-function-in-keras

from keras.layers import Dropout

model_drop = Sequential()

model_drop.add(Dense(512, activation='sigmoid', input_shape=(input_dim,), kern
el_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(128, activation='sigmoid', kernel_initializer=RandomNorma
l(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:\Users\LENOVO\Anaconda3\lib\site-packages\keras\bac
kend\tensorflow_backend.py:3733: 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 - k
eep_prob`.
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_5 (Dense)              (None, 512)               401920    
_________________________________________________________________
batch_normalization_3 (Batch (None, 512)               2048      
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_6 (Dense)              (None, 128)               65664     
_________________________________________________________________
batch_normalization_4 (Batch (None, 128)               512       
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_7 (Dense)              (None, 10)                1290      
=================================================================
Total params: 471,434
Trainable params: 470,154
Non-trainable params: 1,280
_________________________________________________________________
```

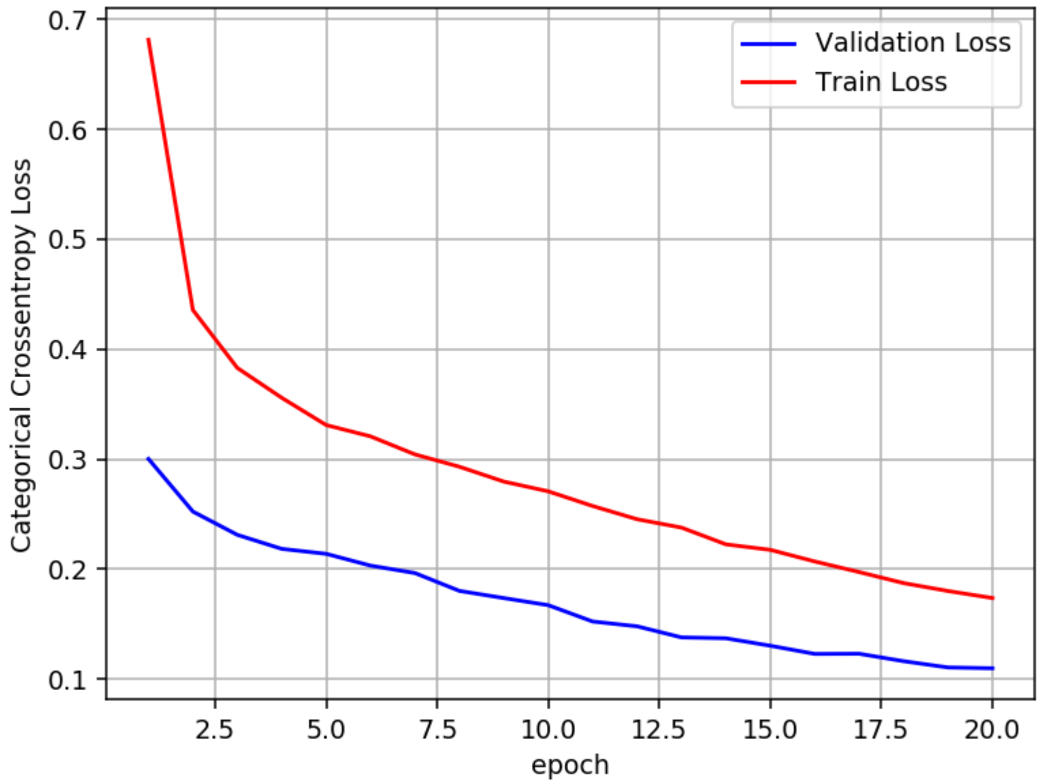
In [18]:
```
model_drop.compile(optimizer='adam', loss='categorical_crossentropy', metrics=
['accuracy'])

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 117us/step - loss: 0.6815 -
acc: 0.7898 - val_loss: 0.3003 - val_acc: 0.9099
Epoch 2/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.4357 -
acc: 0.8681 - val_loss: 0.2524 - val_acc: 0.9262
Epoch 3/20
60000/60000 [==============================] - 5s 81us/step - loss: 0.3831 -
acc: 0.8834 - val_loss: 0.2312 - val_acc: 0.9326
Epoch 4/20
60000/60000 [==============================] - 5s 81us/step - loss: 0.3559 -
acc: 0.8928 - val_loss: 0.2185 - val_acc: 0.9344
Epoch 5/20
60000/60000 [==============================] - 5s 81us/step - loss: 0.3311 -
acc: 0.8997 - val_loss: 0.2140 - val_acc: 0.9391
Epoch 6/20
60000/60000 [==============================] - 5s 81us/step - loss: 0.3208 -
acc: 0.9033 - val_loss: 0.2032 - val_acc: 0.9386
Epoch 7/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.3044 -
acc: 0.9074 - val_loss: 0.1965 - val_acc: 0.9421
Epoch 8/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.2932 -
acc: 0.9112 - val_loss: 0.1803 - val_acc: 0.9458
Epoch 9/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.2797 -
acc: 0.9159 - val_loss: 0.1737 - val_acc: 0.9491
Epoch 10/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.2708 -
acc: 0.9182 - val_loss: 0.1673 - val_acc: 0.9490
Epoch 11/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.2575 -
acc: 0.9221 - val_loss: 0.1524 - val_acc: 0.9542
Epoch 12/20
60000/60000 [==============================] - 5s 80us/step - loss: 0.2454 -
acc: 0.9260 - val_loss: 0.1480 - val_acc: 0.9545
Epoch 13/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.2378 -
acc: 0.9276 - val_loss: 0.1380 - val_acc: 0.9593
Epoch 14/20
60000/60000 [==============================] - 5s 79us/step - loss: 0.2226 -
acc: 0.9331 - val_loss: 0.1372 - val_acc: 0.9591
Epoch 15/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.2176 -
acc: 0.9342 - val_loss: 0.1304 - val_acc: 0.9617
Epoch 16/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.2070 -
acc: 0.9369 - val_loss: 0.1230 - val_acc: 0.9621
Epoch 17/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.1974 -
acc: 0.9403 - val_loss: 0.1231 - val_acc: 0.9635
Epoch 18/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.1873 -
acc: 0.9432 - val_loss: 0.1163 - val_acc: 0.9641
Epoch 19/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.1802 -
```

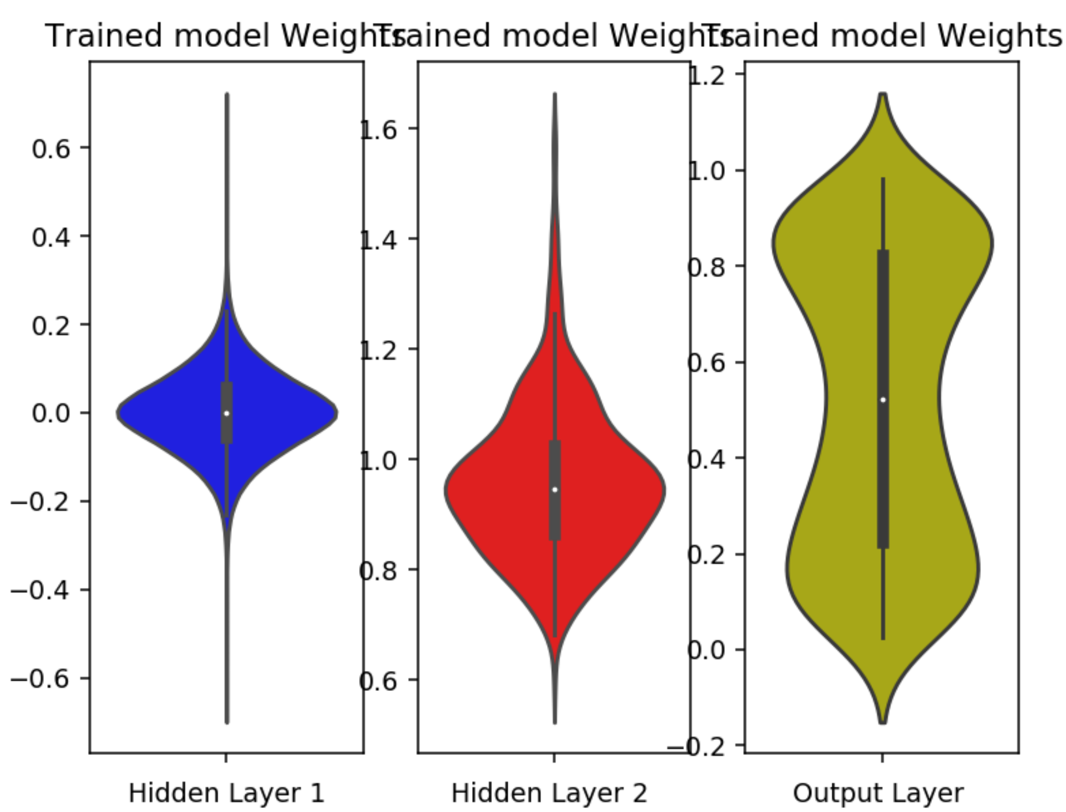
```
acc: 0.9458 - val_loss: 0.1106 - val_acc: 0.9671
Epoch 20/20
60000/60000 [==============================] - 5s 87us/step - loss: 0.1738 -
acc: 0.9475 - val_loss: 0.1099 - val_acc: 0.9667
```

In [19]:
```
score = model_drop.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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


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

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

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

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to num
ber of epochs

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

```
Test score: 0.1098791686380282
Test accuracy: 0.9667
```



In [20]:
```
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()
```



Hyper-parameter tuning of Keras models using Sklearn

In [21]:
```
from keras.optimizers import Adam,RMSprop,SGD
def best_hyperparameters(activ):

    model = Sequential()
    model.add(Dense(512, activation=activ, input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
    model.add(Dense(128, activation=activ, kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
    model.add(Dense(output_dim, activation='softmax'))


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

    return model
```

In [23]:
```
# https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/
import time
start_time = time.time()

activ = ['sigmoid','relu']

from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV

model = KerasClassifier(build_fn=best_hyperparameters, epochs=nb_epoch, batch_size=batch_size, verbose=0)
param_grid = dict(activ=activ)

# if you are using CPU
# grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1)
# if you are using GPU dont use the n_jobs parameter

grid = GridSearchCV(estimator=model, param_grid=param_grid)
grid_result = grid.fit(X_train, Y_train)
print("Execution time: " + str((time.time() - start_time)) + ' ms')
```

```
Execution time: 322.01695251464844 ms
```

In [24]:
```
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.975867 using {'activ': 'sigmoid'}
0.975867 (0.001274) with: {'activ': 'sigmoid'}
0.973517 (0.004000) with: {'activ': 'relu'}
```

Assignment :

We'll fix Adam optimizer and Relu activation units for all the architechtures

ARCHITECTURE 1(624,430) : MLP + Batch-Norm and Dropout(0.5) on hidden Layers

In [16]:
```
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout

model_arch1 = Sequential()

model_arch1.add(Dense(624, activation='relu', input_shape=(input_dim,), kernel
_initializer=RandomNormal(mean=0.0, stddev=0.056, seed=None)))
model_arch1.add(BatchNormalization())
model_arch1.add(Dropout(0.5))

model_arch1.add(Dense(430, activation='relu', kernel_initializer=RandomNormal(
mean=0.0, stddev=0.068, seed=None)) )
model_arch1.add(BatchNormalization())
model_arch1.add(Dropout(0.5))

model_arch1.add(Dense(output_dim, activation='softmax'))


model_arch1.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 624)               489840    
_________________________________________________________________
batch_normalization_1 (Batch (None, 624)               2496      
_________________________________________________________________
dropout_1 (Dropout)          (None, 624)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 430)               268750    
_________________________________________________________________
batch_normalization_2 (Batch (None, 430)               1720      
_________________________________________________________________
dropout_2 (Dropout)          (None, 430)               0         
_________________________________________________________________
dense_3 (Dense)              (None, 10)                4310      
=================================================================
Total params: 767,116
Trainable params: 765,008
Non-trainable params: 2,108
_________________________________________________________________
```

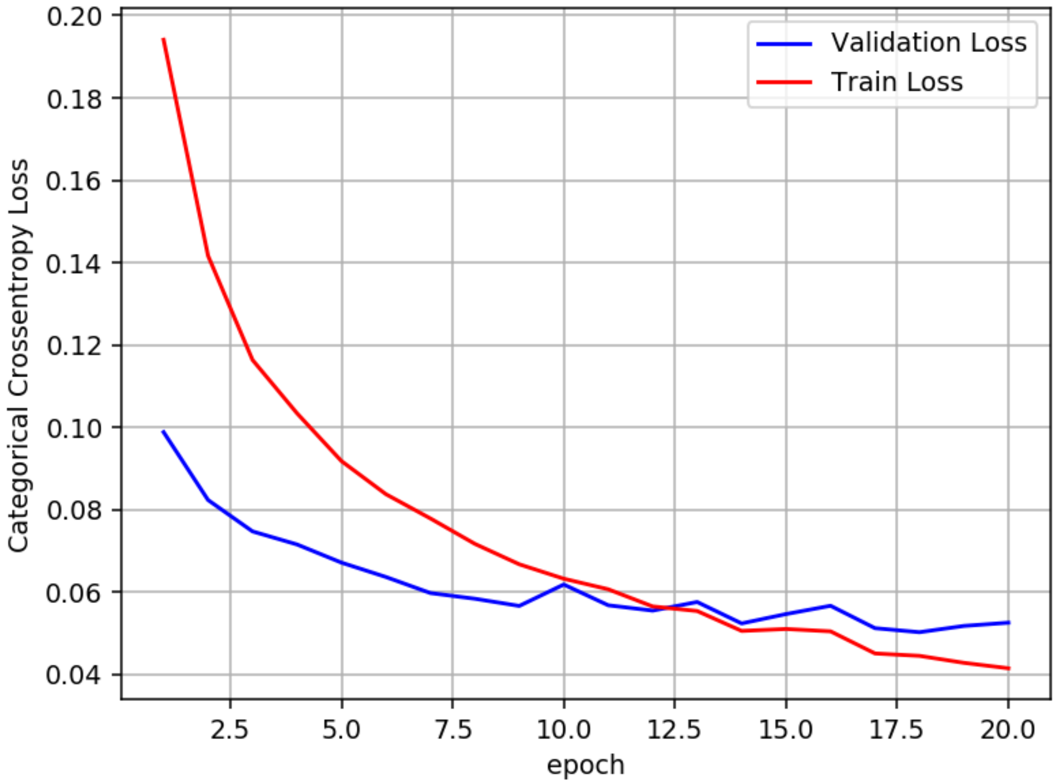
In [18]:
```
model_arch1.compile(optimizer='adam', loss='categorical_crossentropy', metrics
=['accuracy'])

history = model_arch1.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_e
poch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 109us/step - loss: 0.1941 -
acc: 0.9409 - val_loss: 0.0989 - val_acc: 0.9698
Epoch 2/20
60000/60000 [==============================] - 6s 99us/step - loss: 0.1417 -
acc: 0.9565 - val_loss: 0.0824 - val_acc: 0.9745
Epoch 3/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.1164 -
acc: 0.9639 - val_loss: 0.0748 - val_acc: 0.9766
Epoch 4/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.1035 -
acc: 0.9667 - val_loss: 0.0716 - val_acc: 0.9778
Epoch 5/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0918 -
acc: 0.9710 - val_loss: 0.0672 - val_acc: 0.9789
Epoch 6/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0838 -
acc: 0.9734 - val_loss: 0.0637 - val_acc: 0.9799
Epoch 7/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0779 -
acc: 0.9754 - val_loss: 0.0598 - val_acc: 0.9831
Epoch 8/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0718 -
acc: 0.9772 - val_loss: 0.0584 - val_acc: 0.9814
Epoch 9/20
60000/60000 [==============================] - 5s 92us/step - loss: 0.0668 -
acc: 0.9785 - val_loss: 0.0567 - val_acc: 0.9838
Epoch 10/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0633 -
acc: 0.9791 - val_loss: 0.0619 - val_acc: 0.9831
Epoch 11/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0607 -
acc: 0.9800 - val_loss: 0.0568 - val_acc: 0.9833
Epoch 12/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0566 -
acc: 0.9820 - val_loss: 0.0555 - val_acc: 0.9832
Epoch 13/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0555 -
acc: 0.9826 - val_loss: 0.0576 - val_acc: 0.9829
Epoch 14/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.0506 -
acc: 0.9830 - val_loss: 0.0524 - val_acc: 0.9852
Epoch 15/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.0511 -
acc: 0.9835 - val_loss: 0.0547 - val_acc: 0.9837
Epoch 16/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.0505 -
acc: 0.9832 - val_loss: 0.0567 - val_acc: 0.9839
Epoch 17/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0452 -
acc: 0.9847 - val_loss: 0.0513 - val_acc: 0.9855
Epoch 18/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.0446 -
acc: 0.9854 - val_loss: 0.0503 - val_acc: 0.9852
Epoch 19/20
60000/60000 [==============================] - 6s 101us/step - loss: 0.0429 -
```

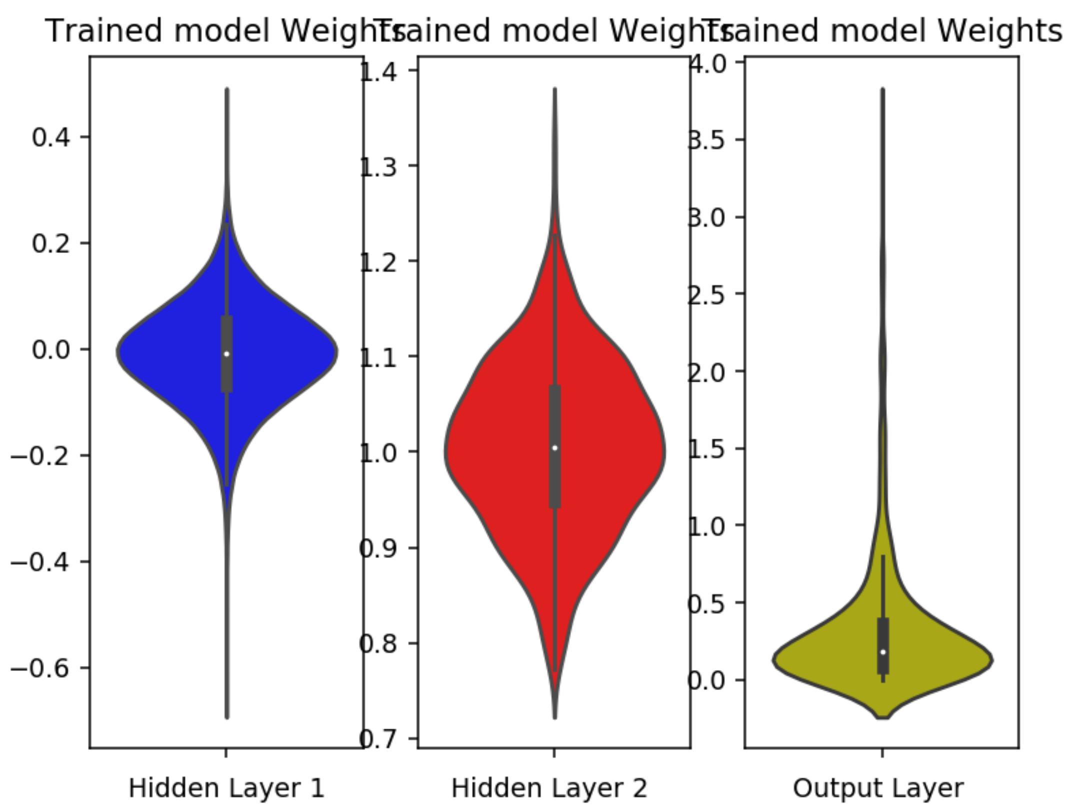
```
acc: 0.9865 - val_loss: 0.0518 - val_acc: 0.9841
Epoch 20/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0415 -
acc: 0.9869 - val_loss: 0.0526 - val_acc: 0.9851
```

In [19]:
```
score = model_arch1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

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

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to num
ber of epochs

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

```
Test score: 0.052614248323663196
Test accuracy: 0.9851
```



In [20]:
```
w_after = model_arch1.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()
```



In [ ]:

ARCHITECTURE 2(512,364,58) : MLP + Batch-Norm and Dropout(0.5) on hidden Layers

In [27]:
```
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout

model_arch2 = Sequential()

model_arch2.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel
_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_arch2.add(BatchNormalization())
model_arch2.add(Dropout(0.5))

model_arch2.add(Dense(364, activation='relu', kernel_initializer=RandomNormal(
mean=0.0, stddev=0.074, seed=None)))
model_arch2.add(BatchNormalization())
model_arch2.add(Dropout(0.5))

model_arch2.add(Dense(58, activation='relu', kernel_initializer=RandomNormal(m
ean=0.0, stddev=0.185, seed=None)) )
model_arch2.add(BatchNormalization())
model_arch2.add(Dropout(0.5))

model_arch2.add(Dense(output_dim, activation='softmax'))


model_arch2.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_8 (Dense)              (None, 512)               401920    
_________________________________________________________________
batch_normalization_6 (Batch (None, 512)               2048      
_________________________________________________________________
dropout_6 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_9 (Dense)              (None, 364)               186732    
_________________________________________________________________
batch_normalization_7 (Batch (None, 364)               1456      
_________________________________________________________________
dropout_7 (Dropout)          (None, 364)               0         
_________________________________________________________________
dense_10 (Dense)             (None, 58)                21170     
_________________________________________________________________
batch_normalization_8 (Batch (None, 58)                232       
_________________________________________________________________
dropout_8 (Dropout)          (None, 58)                0         
_________________________________________________________________
dense_11 (Dense)             (None, 10)                590       
=================================================================
Total params: 614,148
Trainable params: 612,280
Non-trainable params: 1,868
_________________________________________________________________
```

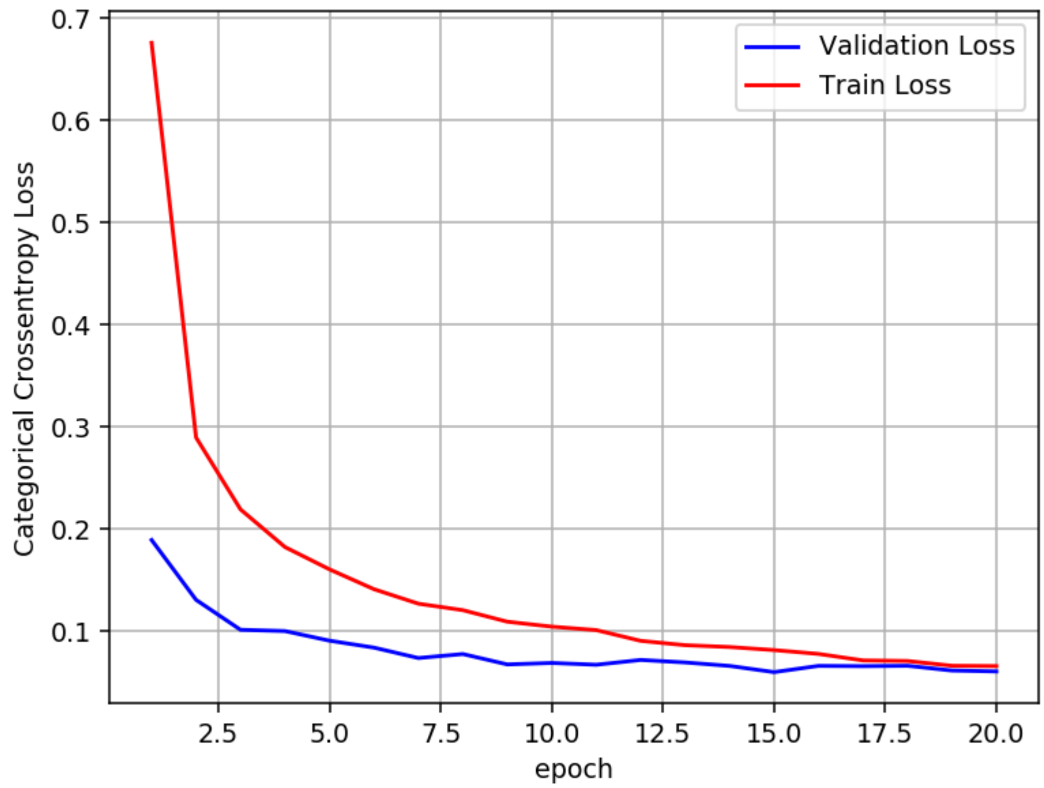
In [28]:
```
model_arch2.compile(optimizer='adam', loss='categorical_crossentropy', metrics
=['accuracy'])

history = model_arch2.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_e
poch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 8s 141us/step - loss: 0.6757 -
acc: 0.7967 - val_loss: 0.1893 - val_acc: 0.9429
Epoch 2/20
60000/60000 [==============================] - 6s 106us/step - loss: 0.2895 -
acc: 0.9178 - val_loss: 0.1307 - val_acc: 0.9589
Epoch 3/20
60000/60000 [==============================] - 7s 112us/step - loss: 0.2192 -
acc: 0.9370 - val_loss: 0.1014 - val_acc: 0.9693
Epoch 4/20
60000/60000 [==============================] - 7s 109us/step - loss: 0.1824 -
acc: 0.9481 - val_loss: 0.1002 - val_acc: 0.9706
Epoch 5/20
60000/60000 [==============================] - 7s 109us/step - loss: 0.1606 -
acc: 0.9540 - val_loss: 0.0908 - val_acc: 0.9731
Epoch 6/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.1412 -
acc: 0.9589 - val_loss: 0.0840 - val_acc: 0.9762
Epoch 7/20
60000/60000 [==============================] - 6s 106us/step - loss: 0.1270 -
acc: 0.9629 - val_loss: 0.0738 - val_acc: 0.9785
Epoch 8/20
60000/60000 [==============================] - 6s 106us/step - loss: 0.1206 -
acc: 0.9662 - val_loss: 0.0777 - val_acc: 0.9784
Epoch 9/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.1093 -
acc: 0.9690 - val_loss: 0.0675 - val_acc: 0.9800
Epoch 10/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.1045 -
acc: 0.9697 - val_loss: 0.0690 - val_acc: 0.9794
Epoch 11/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.1011 -
acc: 0.9708 - val_loss: 0.0672 - val_acc: 0.9791
Epoch 12/20
60000/60000 [==============================] - 6s 106us/step - loss: 0.0906 -
acc: 0.9734 - val_loss: 0.0719 - val_acc: 0.9788
Epoch 13/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.0864 -
acc: 0.9745 - val_loss: 0.0695 - val_acc: 0.9791
Epoch 14/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.0846 -
acc: 0.9750 - val_loss: 0.0661 - val_acc: 0.9810
Epoch 15/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.0816 -
acc: 0.9768 - val_loss: 0.0600 - val_acc: 0.9826
Epoch 16/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.0778 -
acc: 0.9772 - val_loss: 0.0661 - val_acc: 0.9798
Epoch 17/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.0714 -
acc: 0.9789 - val_loss: 0.0659 - val_acc: 0.9824
Epoch 18/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.0709 -
acc: 0.9791 - val_loss: 0.0662 - val_acc: 0.9824
Epoch 19/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.0662 -
```

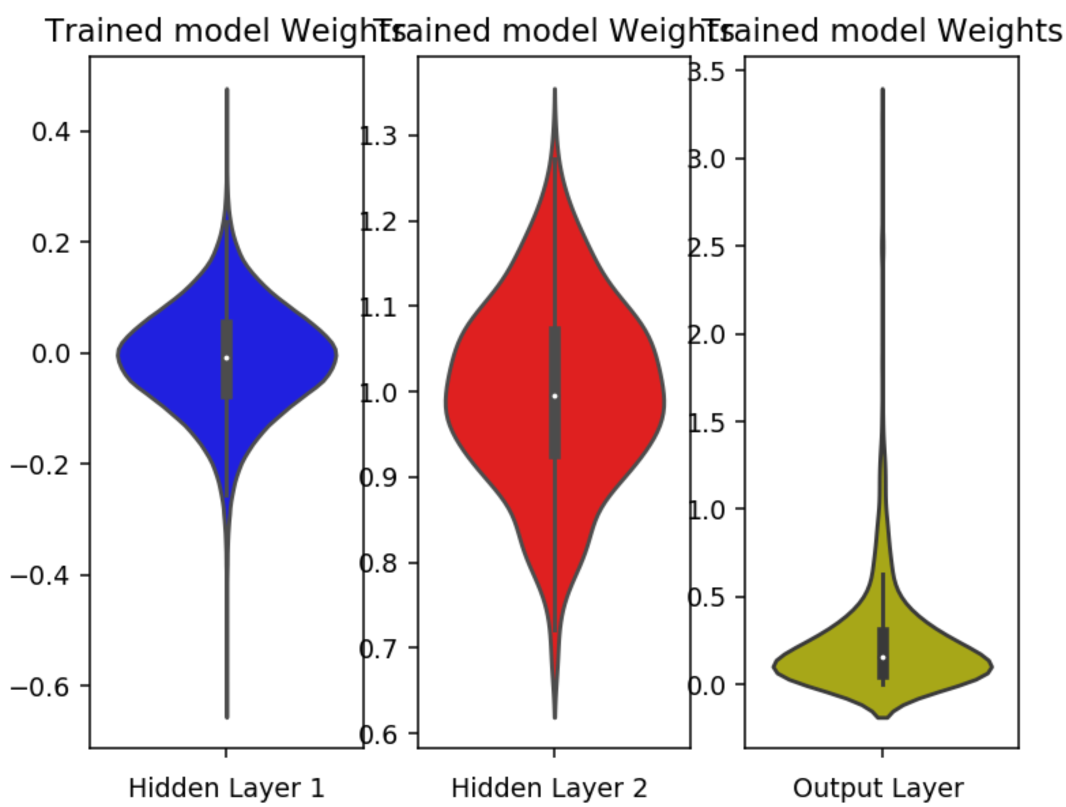
```
acc: 0.9809 - val_loss: 0.0616 - val_acc: 0.9827
Epoch 20/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.0659 -
acc: 0.9807 - val_loss: 0.0607 - val_acc: 0.9819
```

In [29]:
```
score = model_arch2.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

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

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to num
ber of epochs

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

```
Test score: 0.060684567966146276
Test accuracy: 0.9819
```



In [30]:
```
w_after = model_arch2.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()
```



In [ ]:

ARCHITECTURE 3(584,452,312,256,128) : MLP + Batch-Norm and Dropout(0.5) on hidden Layers

In [31]:
```
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout

model_arch3 = Sequential()

model_arch3.add(Dense(584, activation='relu', input_shape=(input_dim,), kernel
_initializer=RandomNormal(mean=0.0, stddev=0.058, seed=None)))
model_arch3.add(BatchNormalization())
model_arch3.add(Dropout(0.5))

model_arch3.add(Dense(452, activation='relu', kernel_initializer=RandomNormal(
mean=0.0, stddev=0.066, seed=None)))
model_arch3.add(BatchNormalization())
model_arch3.add(Dropout(0.5))

model_arch3.add(Dense(312, activation='relu', kernel_initializer=RandomNormal(
mean=0.0, stddev=0.080, seed=None)))
model_arch3.add(BatchNormalization())
model_arch3.add(Dropout(0.5))

model_arch3.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(
mean=0.0, stddev=0.088, seed=None)))
model_arch3.add(BatchNormalization())
model_arch3.add(Dropout(0.5))

model_arch3.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(
mean=0.0, stddev=0.125, seed=None)) )
model_arch3.add(BatchNormalization())
model_arch3.add(Dropout(0.5))

model_arch3.add(Dense(output_dim, activation='softmax'))


model_arch3.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_12 (Dense)             (None, 584)               458440    
_________________________________________________________________
batch_normalization_9 (Batch (None, 584)               2336      
_________________________________________________________________
dropout_9 (Dropout)          (None, 584)               0         
_________________________________________________________________
dense_13 (Dense)             (None, 452)               264420    
_________________________________________________________________
batch_normalization_10 (Batc (None, 452)               1808      
_________________________________________________________________
dropout_10 (Dropout)         (None, 452)               0         
_________________________________________________________________
dense_14 (Dense)             (None, 312)               141336    
_________________________________________________________________
batch_normalization_11 (Batc (None, 312)               1248      
_________________________________________________________________
dropout_11 (Dropout)         (None, 312)               0         
_________________________________________________________________
dense_15 (Dense)             (None, 256)               80128     
_________________________________________________________________
batch_normalization_12 (Batc (None, 256)               1024      
_________________________________________________________________
dropout_12 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_16 (Dense)             (None, 128)               32896     
_________________________________________________________________
batch_normalization_13 (Batc (None, 128)               512       
_________________________________________________________________
dropout_13 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_17 (Dense)             (None, 10)                1290      
=================================================================
Total params: 985,438
Trainable params: 981,974
Non-trainable params: 3,464
_________________________________________________________________
```

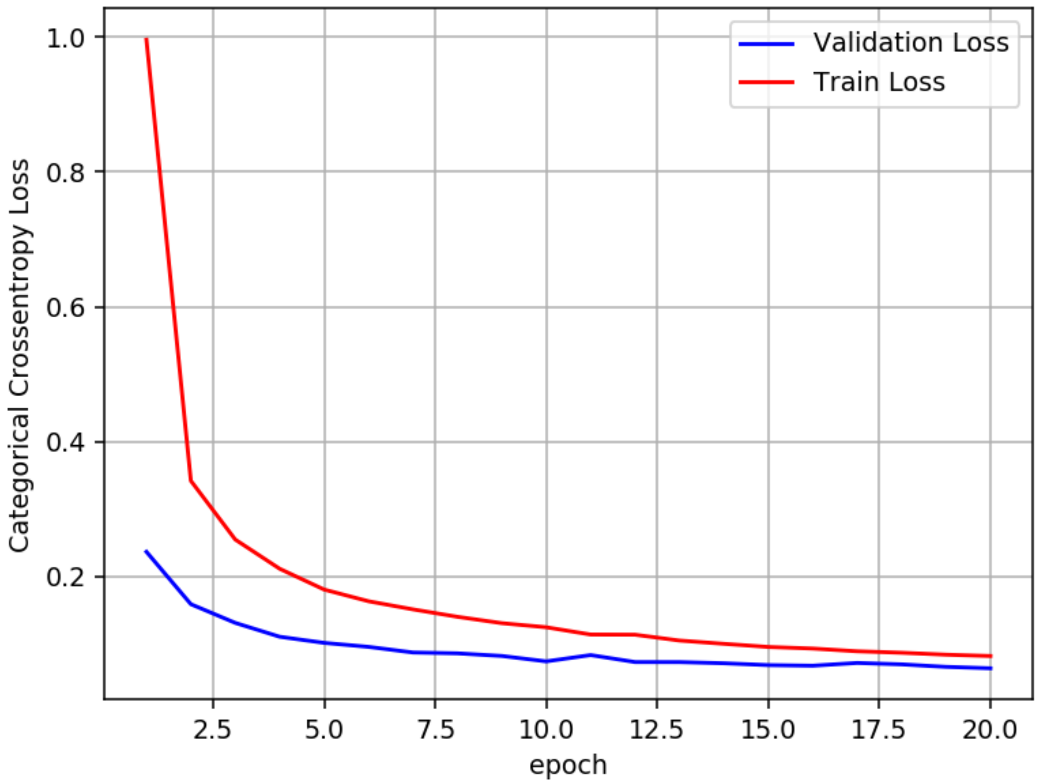
In [32]:
```
model_arch3.compile(optimizer='adam', loss='categorical_crossentropy', metrics
=['accuracy'])

history = model_arch3.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_e
poch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 11s 190us/step - loss: 0.9959
- acc: 0.6929 - val_loss: 0.2367 - val_acc: 0.9306
Epoch 2/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.3420 -
acc: 0.9013 - val_loss: 0.1591 - val_acc: 0.9543
Epoch 3/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.2548 -
acc: 0.9276 - val_loss: 0.1313 - val_acc: 0.9630
Epoch 4/20
60000/60000 [==============================] - 9s 151us/step - loss: 0.2115 -
acc: 0.9397 - val_loss: 0.1108 - val_acc: 0.9698
Epoch 5/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.1807 -
acc: 0.9502 - val_loss: 0.1017 - val_acc: 0.9701
Epoch 6/20
60000/60000 [==============================] - 9s 142us/step - loss: 0.1634 -
acc: 0.9547 - val_loss: 0.0958 - val_acc: 0.9742
Epoch 7/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.1514 -
acc: 0.9578 - val_loss: 0.0874 - val_acc: 0.9768
Epoch 8/20
60000/60000 [==============================] - 9s 155us/step - loss: 0.1404 -
acc: 0.9611 - val_loss: 0.0862 - val_acc: 0.9752
Epoch 9/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.1309 -
acc: 0.9630 - val_loss: 0.0823 - val_acc: 0.9763
Epoch 10/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.1248 -
acc: 0.9654 - val_loss: 0.0743 - val_acc: 0.9783
Epoch 11/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.1141 -
acc: 0.9685 - val_loss: 0.0836 - val_acc: 0.9763
Epoch 12/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.1138 -
acc: 0.9686 - val_loss: 0.0733 - val_acc: 0.9797
Epoch 13/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.1052 -
acc: 0.9707 - val_loss: 0.0732 - val_acc: 0.9803
Epoch 14/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.1003 -
acc: 0.9715 - val_loss: 0.0715 - val_acc: 0.9814
Epoch 15/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.0957 -
acc: 0.9733 - val_loss: 0.0687 - val_acc: 0.9816
Epoch 16/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0933 -
acc: 0.9736 - val_loss: 0.0679 - val_acc: 0.9825
Epoch 17/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.0893 -
acc: 0.9746 - val_loss: 0.0719 - val_acc: 0.9817
Epoch 18/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0871 -
acc: 0.9756 - val_loss: 0.0699 - val_acc: 0.9831
Epoch 19/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0842 -
```

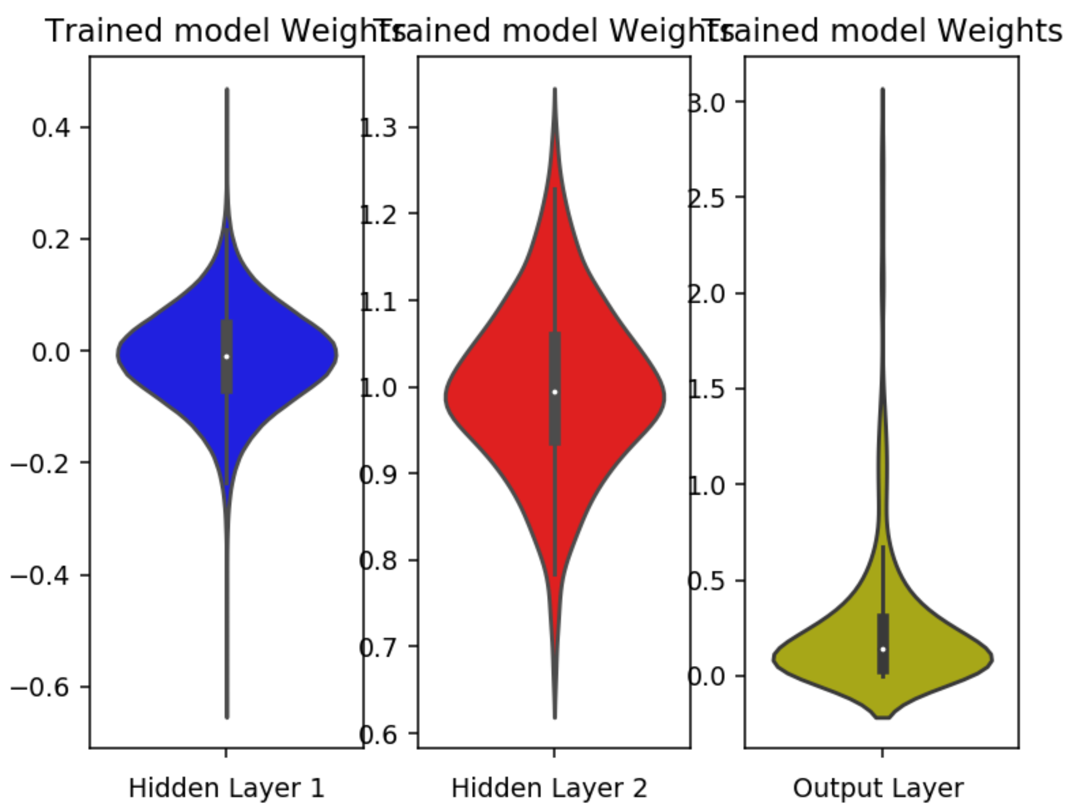
```
acc: 0.9768 - val_loss: 0.0661 - val_acc: 0.9826
Epoch 20/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.0822 -
acc: 0.9775 - val_loss: 0.0640 - val_acc: 0.9828
```

In [33]:
```
score = model_arch3.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

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

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to num
ber of epochs

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

```
Test score: 0.064016801341902446
Test accuracy: 0.9828
```



In [34]:
```
w_after = model_arch3.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()
```



Now lets play around with Dropout rates, Batch nomalization and without batch normalization

ARCHITECTURE 4(682,452,312,256,128,64) : MLP + Batch-Norm and Dropout(0.6,0.3,0.3,0.5,0.3,0.2), Relu on hidden Layers

In [13]:
```
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout

model_arch4 = Sequential()

model_arch4.add(Dense(682, activation='relu', input_shape=(input_dim,), kernel
_initializer=RandomNormal(mean=0.0, stddev=0.054, seed=None)))
model_arch4.add(BatchNormalization())
model_arch4.add(Dropout(0.6))

model_arch4.add(Dense(452, activation='relu', kernel_initializer=RandomNormal(
mean=0.0, stddev=0.066, seed=None)))
model_arch4.add(BatchNormalization())
model_arch4.add(Dropout(0.3))

model_arch4.add(Dense(312, activation='relu', kernel_initializer=RandomNormal(
mean=0.0, stddev=0.080, seed=None)))
model_arch4.add(BatchNormalization())
model_arch4.add(Dropout(0.3))

model_arch4.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(
mean=0.0, stddev=0.088, seed=None)))
model_arch4.add(BatchNormalization())
model_arch4.add(Dropout(0.5))

model_arch4.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(
mean=0.0, stddev=0.125, seed=None)) )
model_arch4.add(BatchNormalization())
model_arch4.add(Dropout(0.3))

model_arch4.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(m
ean=0.0, stddev=0.176, seed=None)) )
model_arch4.add(BatchNormalization())
model_arch4.add(Dropout(0.2))

model_arch4.add(Dense(output_dim, activation='softmax'))


model_arch4.summary()
```

```
WARNING:tensorflow:From C:\Users\LENOVO\Anaconda3\lib\site-packages\keras\bac
kend\tensorflow_backend.py:66: The name tf.get_default_graph is deprecated. P
lease use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From C:\Users\LENOVO\Anaconda3\lib\site-packages\keras\bac
kend\tensorflow_backend.py:541: The name tf.placeholder is deprecated. Please
use tf.compat.v1.placeholder instead.

WARNING:tensorflow:From C:\Users\LENOVO\Anaconda3\lib\site-packages\keras\bac
kend\tensorflow_backend.py:4409: The name tf.random_normal is deprecated. Ple
ase use tf.random.normal instead.

WARNING:tensorflow:From C:\Users\LENOVO\Anaconda3\lib\site-packages\keras\bac
kend\tensorflow_backend.py:148: The name tf.placeholder_with_default is depre
cated. Please use tf.compat.v1.placeholder_with_default instead.

WARNING:tensorflow:From C:\Users\LENOVO\Anaconda3\lib\site-packages\keras\bac
kend\tensorflow_backend.py:3733: 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 - k
eep_prob`.
WARNING:tensorflow:Large dropout rate: 0.6 (>0.5). In TensorFlow 2.x, dropout
() uses dropout rate instead of keep_prob. Please ensure that this is intende
d.
WARNING:tensorflow:From C:\Users\LENOVO\Anaconda3\lib\site-packages\keras\bac
kend\tensorflow_backend.py:4432: The name tf.random_uniform is deprecated. Pl
ease use tf.random.uniform instead.

Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 682)               535370    
_________________________________________________________________
batch_normalization_1 (Batch (None, 682)               2728      
_________________________________________________________________
dropout_1 (Dropout)          (None, 682)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 452)               308716    
_________________________________________________________________
batch_normalization_2 (Batch (None, 452)               1808      
_________________________________________________________________
dropout_2 (Dropout)          (None, 452)               0         
_________________________________________________________________
dense_3 (Dense)              (None, 312)               141336    
_________________________________________________________________
batch_normalization_3 (Batch (None, 312)               1248      
_________________________________________________________________
dropout_3 (Dropout)          (None, 312)               0         
_________________________________________________________________
dense_4 (Dense)              (None, 256)               80128     
_________________________________________________________________
batch_normalization_4 (Batch (None, 256)               1024      
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0         
_________________________________________________________________
```

```
dense_5 (Dense)              (None, 128)               32896     
_________________________________________________________________
batch_normalization_5 (Batch (None, 128)               512       
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_6 (Dense)              (None, 64)                8256      
_________________________________________________________________
batch_normalization_6 (Batch (None, 64)                256       
_________________________________________________________________
dropout_6 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_7 (Dense)              (None, 10)                650       
=================================================================
Total params: 1,114,928
Trainable params: 1,111,140
Non-trainable params: 3,788
_________________________________________________________________
```

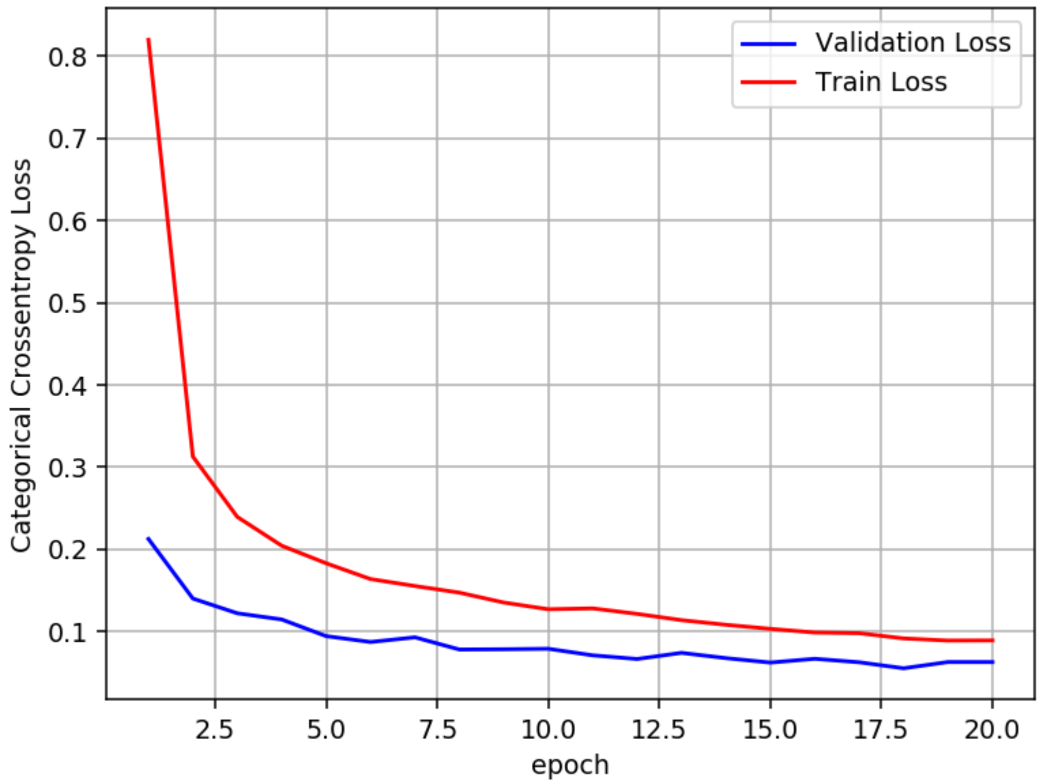
In [15]:
```
model_arch4.compile(optimizer='adam', loss='categorical_crossentropy', metrics
=['accuracy'])

history = model_arch4.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_e
poch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 12s 198us/step - loss: 0.8202
- acc: 0.7423 - val_loss: 0.2126 - val_acc: 0.9380
Epoch 2/20
60000/60000 [==============================] - 9s 154us/step - loss: 0.3129 -
acc: 0.9115 - val_loss: 0.1401 - val_acc: 0.9603
Epoch 3/20
60000/60000 [==============================] - 9s 155us/step - loss: 0.2395 -
acc: 0.9329 - val_loss: 0.1222 - val_acc: 0.9646
Epoch 4/20
60000/60000 [==============================] - 10s 160us/step - loss: 0.2044
- acc: 0.9426 - val_loss: 0.1146 - val_acc: 0.9691
Epoch 5/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.1830 -
acc: 0.9472 - val_loss: 0.0944 - val_acc: 0.9736
Epoch 6/20
60000/60000 [==============================] - 10s 167us/step - loss: 0.1638
- acc: 0.9534 - val_loss: 0.0872 - val_acc: 0.9766
Epoch 7/20
60000/60000 [==============================] - 31s 524us/step - loss: 0.1553
- acc: 0.9554 - val_loss: 0.0929 - val_acc: 0.9750
Epoch 8/20
60000/60000 [==============================] - 31s 518us/step - loss: 0.1473
- acc: 0.9585 - val_loss: 0.0780 - val_acc: 0.9787
Epoch 9/20
60000/60000 [==============================] - 30s 506us/step - loss: 0.1352
- acc: 0.9616 - val_loss: 0.0783 - val_acc: 0.9793
Epoch 10/20
60000/60000 [==============================] - 31s 516us/step - loss: 0.1270
- acc: 0.9634 - val_loss: 0.0790 - val_acc: 0.9774
Epoch 11/20
60000/60000 [==============================] - 31s 509us/step - loss: 0.1281
- acc: 0.9639 - val_loss: 0.0710 - val_acc: 0.9807
Epoch 12/20
60000/60000 [==============================] - 31s 510us/step - loss: 0.1214
- acc: 0.9654 - val_loss: 0.0665 - val_acc: 0.9801
Epoch 13/20
60000/60000 [==============================] - 31s 517us/step - loss: 0.1138
- acc: 0.9680 - val_loss: 0.0740 - val_acc: 0.9787
Epoch 14/20
60000/60000 [==============================] - 30s 508us/step - loss: 0.1081
- acc: 0.9695 - val_loss: 0.0676 - val_acc: 0.9814
Epoch 15/20
60000/60000 [==============================] - 31s 521us/step - loss: 0.1032
- acc: 0.9710 - val_loss: 0.0622 - val_acc: 0.9830
Epoch 16/20
60000/60000 [==============================] - 31s 516us/step - loss: 0.0987
- acc: 0.9722 - val_loss: 0.0668 - val_acc: 0.9831
Epoch 17/20
60000/60000 [==============================] - 31s 509us/step - loss: 0.0979
- acc: 0.9713 - val_loss: 0.0625 - val_acc: 0.9819
Epoch 18/20
60000/60000 [==============================] - 30s 505us/step - loss: 0.0915
- acc: 0.9741 - val_loss: 0.0552 - val_acc: 0.9853
Epoch 19/20
60000/60000 [==============================] - 31s 517us/step - loss: 0.0890
```

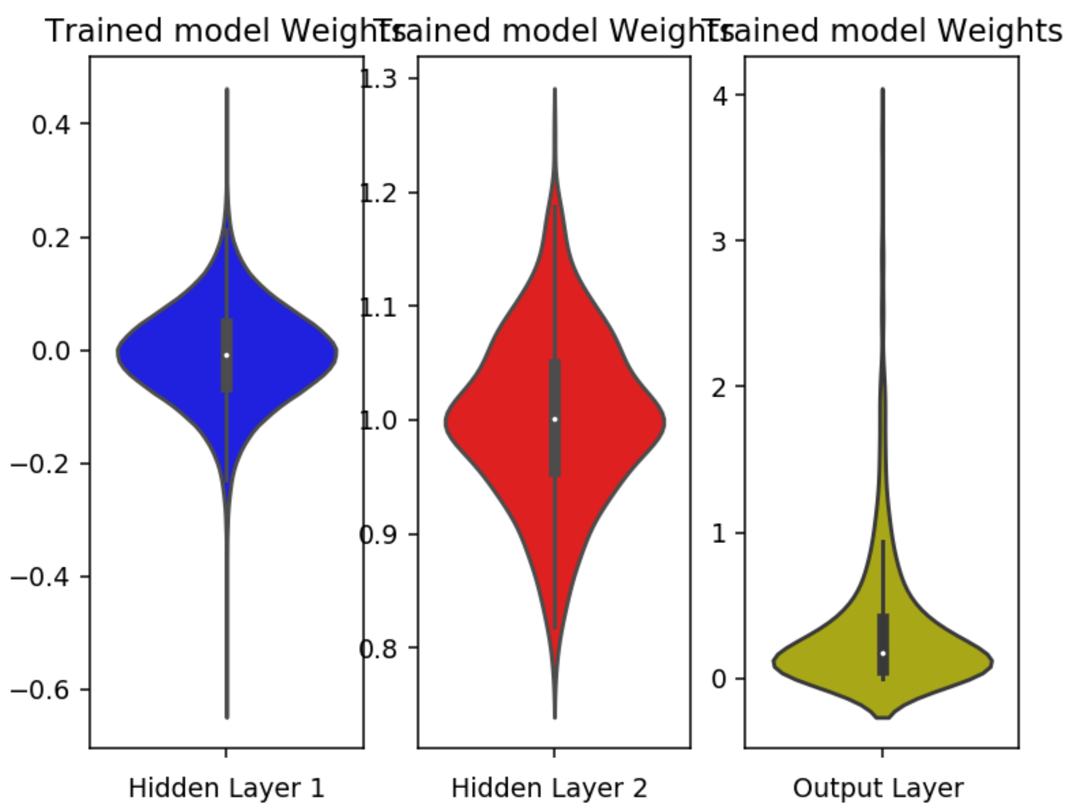
```
- acc: 0.9745 - val_loss: 0.0628 - val_acc: 0.9839
Epoch 20/20
60000/60000 [==============================] - 31s 516us/step - loss: 0.0891
- acc: 0.9750 - val_loss: 0.0628 - val_acc: 0.9833
```

In [16]:
```
score = model_arch4.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

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

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to num
ber of epochs

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

```
Test score: 0.06279641111334786
Test accuracy: 0.9833
```



In [17]:
```
w_after = model_arch4.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()
```



Lets increase the number of epochs, that might increase the accuracy

ARCHITECTURE 5(624,430) : MLP + Batch-Norm and Dropout(0.6,0.5) on hidden Layers, 50 epochs

In [18]:
```
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout

model_arch5 = Sequential()

model_arch5.add(Dense(624, activation='relu', input_shape=(input_dim,), kernel
_initializer=RandomNormal(mean=0.0, stddev=0.056, seed=None)))
model_arch5.add(BatchNormalization())
model_arch5.add(Dropout(0.6))

model_arch5.add(Dense(430, activation='relu', kernel_initializer=RandomNormal(
mean=0.0, stddev=0.068, seed=None)) )
model_arch5.add(BatchNormalization())
model_arch5.add(Dropout(0.5))

model_arch5.add(Dense(output_dim, activation='softmax'))


model_arch5.summary()
```

```
WARNING:tensorflow:Large dropout rate: 0.6 (>0.5). In TensorFlow 2.x, dropout
() uses dropout rate instead of keep_prob. Please ensure that this is intende
d.
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_8 (Dense)              (None, 624)               489840    
_________________________________________________________________
batch_normalization_7 (Batch (None, 624)               2496      
_________________________________________________________________
dropout_7 (Dropout)          (None, 624)               0         
_________________________________________________________________
dense_9 (Dense)              (None, 430)               268750    
_________________________________________________________________
batch_normalization_8 (Batch (None, 430)               1720      
_________________________________________________________________
dropout_8 (Dropout)          (None, 430)               0         
_________________________________________________________________
dense_10 (Dense)             (None, 10)                4310      
=================================================================
Total params: 767,116
Trainable params: 765,008
Non-trainable params: 2,108
_________________________________________________________________
```

In [19]:
```
model_arch5.compile(optimizer='adam', loss='categorical_crossentropy', metrics
=['accuracy'])

history = model_arch5.fit(X_train, Y_train, batch_size=batch_size, epochs=50,
verbose=1, validation_data=(X_test, Y_test))
```

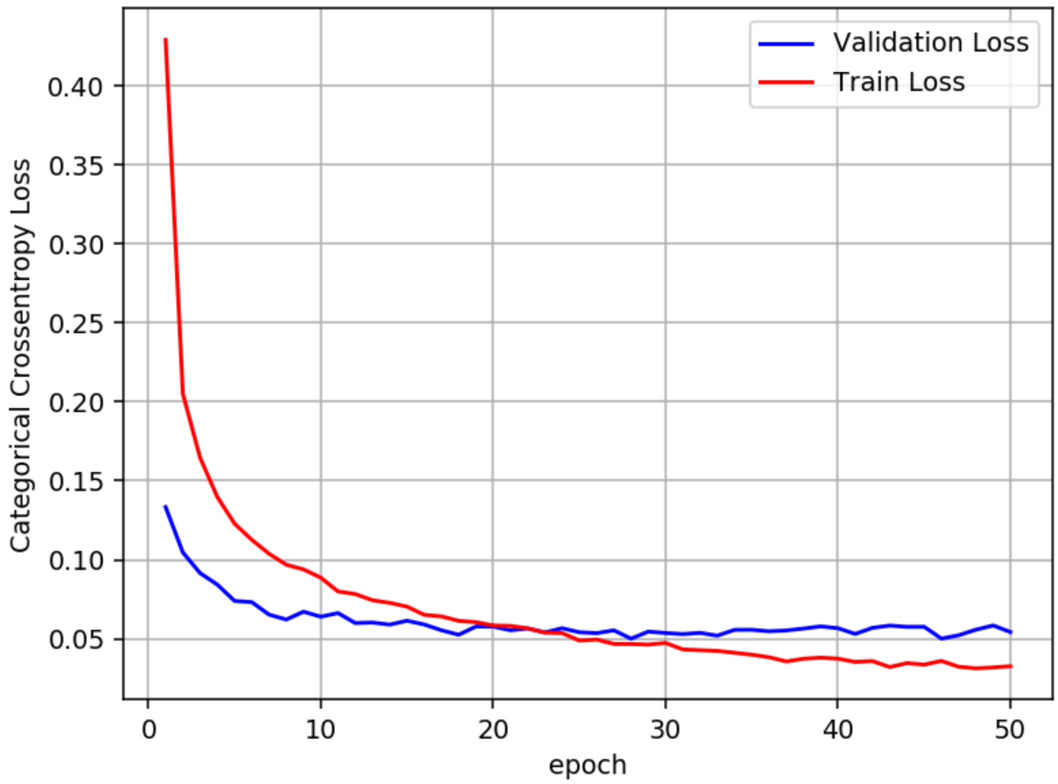
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/50
60000/60000 [==============================] - 20s 338us/step - loss: 0.4291
- acc: 0.8716 - val_loss: 0.1334 - val_acc: 0.9579
Epoch 2/50
60000/60000 [==============================] - 16s 266us/step - loss: 0.2051
- acc: 0.9376 - val_loss: 0.1046 - val_acc: 0.9676
Epoch 3/50
60000/60000 [==============================] - 17s 281us/step - loss: 0.1646
- acc: 0.9494 - val_loss: 0.0916 - val_acc: 0.9725
Epoch 4/50
60000/60000 [==============================] - 16s 261us/step - loss: 0.1397
- acc: 0.9570 - val_loss: 0.0843 - val_acc: 0.9740
Epoch 5/50
60000/60000 [==============================] - 15s 258us/step - loss: 0.1229
- acc: 0.9614 - val_loss: 0.0739 - val_acc: 0.9778
Epoch 6/50
60000/60000 [==============================] - 16s 261us/step - loss: 0.1126
- acc: 0.9645 - val_loss: 0.0732 - val_acc: 0.9769
Epoch 7/50
60000/60000 [==============================] - 15s 256us/step - loss: 0.1038
- acc: 0.9674 - val_loss: 0.0653 - val_acc: 0.9793
Epoch 8/50
60000/60000 [==============================] - 17s 286us/step - loss: 0.0969
- acc: 0.9695 - val_loss: 0.0622 - val_acc: 0.9798
Epoch 9/50
60000/60000 [==============================] - 16s 264us/step - loss: 0.0939
- acc: 0.9702 - val_loss: 0.0671 - val_acc: 0.9796
Epoch 10/50
60000/60000 [==============================] - 16s 259us/step - loss: 0.0887
- acc: 0.9716 - val_loss: 0.0640 - val_acc: 0.9818
Epoch 11/50
60000/60000 [==============================] - 16s 264us/step - loss: 0.0800
- acc: 0.9738 - val_loss: 0.0663 - val_acc: 0.9808
Epoch 12/50
60000/60000 [==============================] - 16s 261us/step - loss: 0.0783
- acc: 0.9752 - val_loss: 0.0600 - val_acc: 0.9834
Epoch 13/50
60000/60000 [==============================] - 16s 260us/step - loss: 0.0744
- acc: 0.9759 - val_loss: 0.0603 - val_acc: 0.9816
Epoch 14/50
60000/60000 [==============================] - 16s 265us/step - loss: 0.0727
- acc: 0.9771 - val_loss: 0.0590 - val_acc: 0.9806
Epoch 15/50
60000/60000 [==============================] - 16s 265us/step - loss: 0.0704
- acc: 0.9769 - val_loss: 0.0614 - val_acc: 0.9812
Epoch 16/50
60000/60000 [==============================] - 16s 261us/step - loss: 0.0652
- acc: 0.9779 - val_loss: 0.0590 - val_acc: 0.9834
Epoch 17/50
60000/60000 [==============================] - 16s 260us/step - loss: 0.0642
- acc: 0.9792 - val_loss: 0.0554 - val_acc: 0.9845
Epoch 18/50
60000/60000 [==============================] - 16s 262us/step - loss: 0.0614
- acc: 0.9803 - val_loss: 0.0525 - val_acc: 0.9846
Epoch 19/50
60000/60000 [==============================] - 16s 263us/step - loss: 0.0605
```

```
- acc: 0.9800 - val_loss: 0.0578 - val_acc: 0.9829
Epoch 20/50
60000/60000 [==============================] - 16s 265us/step - loss: 0.0584
- acc: 0.9802 - val_loss: 0.0577 - val_acc: 0.9828
Epoch 21/50
60000/60000 [==============================] - 16s 265us/step - loss: 0.0581
- acc: 0.9812 - val_loss: 0.0554 - val_acc: 0.9839
Epoch 22/50
60000/60000 [==============================] - 16s 272us/step - loss: 0.0566
- acc: 0.9813 - val_loss: 0.0565 - val_acc: 0.9841
Epoch 23/50
60000/60000 [==============================] - 16s 265us/step - loss: 0.0539
- acc: 0.9823 - val_loss: 0.0540 - val_acc: 0.9848
Epoch 24/50
60000/60000 [==============================] - 15s 252us/step - loss: 0.0536
- acc: 0.9832 - val_loss: 0.0567 - val_acc: 0.9839
Epoch 25/50
60000/60000 [==============================] - 15s 254us/step - loss: 0.0489
- acc: 0.9839 - val_loss: 0.0541 - val_acc: 0.9844
Epoch 26/50
60000/60000 [==============================] - 16s 266us/step - loss: 0.0495
- acc: 0.9839 - val_loss: 0.0536 - val_acc: 0.9851
Epoch 27/50
60000/60000 [==============================] - 16s 271us/step - loss: 0.0467
- acc: 0.9850 - val_loss: 0.0553 - val_acc: 0.9857
Epoch 28/50
60000/60000 [==============================] - 9s 155us/step - loss: 0.0467 -
acc: 0.9848 - val_loss: 0.0501 - val_acc: 0.9860
Epoch 29/50
60000/60000 [==============================] - 5s 86us/step - loss: 0.0463 -
acc: 0.9849 - val_loss: 0.0546 - val_acc: 0.9844
Epoch 30/50
60000/60000 [==============================] - 5s 86us/step - loss: 0.0474 -
acc: 0.9848 - val_loss: 0.0536 - val_acc: 0.9845
Epoch 31/50
60000/60000 [==============================] - 6s 93us/step - loss: 0.0433 -
acc: 0.9854 - val_loss: 0.0529 - val_acc: 0.9854
Epoch 32/50
60000/60000 [==============================] - 6s 98us/step - loss: 0.0428 -
acc: 0.9853 - val_loss: 0.0537 - val_acc: 0.9850
Epoch 33/50
60000/60000 [==============================] - 5s 87us/step - loss: 0.0423 -
acc: 0.9862 - val_loss: 0.0520 - val_acc: 0.9853
Epoch 34/50
60000/60000 [==============================] - 5s 86us/step - loss: 0.0412 -
acc: 0.9862 - val_loss: 0.0556 - val_acc: 0.9855
Epoch 35/50
60000/60000 [==============================] - 5s 85us/step - loss: 0.0399 -
acc: 0.9870 - val_loss: 0.0557 - val_acc: 0.9846
Epoch 36/50
60000/60000 [==============================] - 5s 84us/step - loss: 0.0383 -
acc: 0.9873 - val_loss: 0.0547 - val_acc: 0.9853
Epoch 37/50
60000/60000 [==============================] - 5s 82us/step - loss: 0.0357 -
acc: 0.9878 - val_loss: 0.0553 - val_acc: 0.9851
Epoch 38/50
60000/60000 [==============================] - 5s 89us/step - loss: 0.0374 -
```

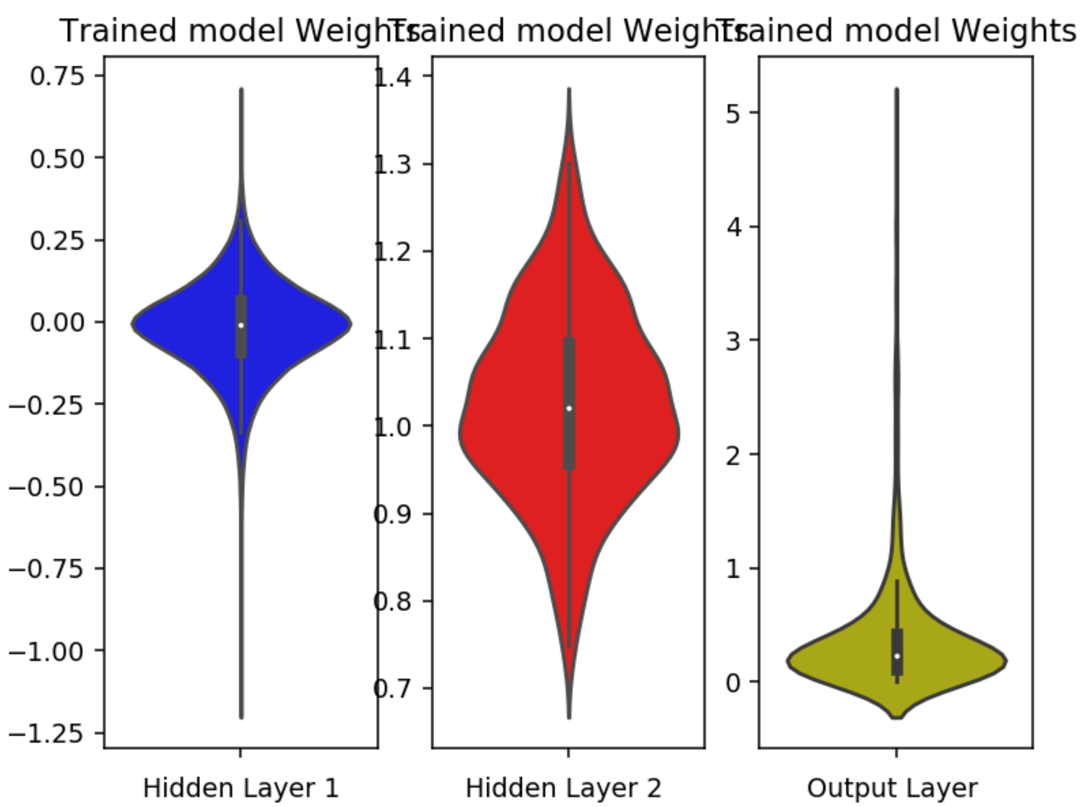
```
acc: 0.9870 - val_loss: 0.0565 - val_acc: 0.9848
Epoch 39/50
60000/60000 [==============================] - 6s 95us/step - loss: 0.0381 -
acc: 0.9872 - val_loss: 0.0578 - val_acc: 0.9854
Epoch 40/50
60000/60000 [==============================] - 6s 94us/step - loss: 0.0374 -
acc: 0.9878 - val_loss: 0.0567 - val_acc: 0.9852
Epoch 41/50
60000/60000 [==============================] - 5s 85us/step - loss: 0.0353 -
acc: 0.9882 - val_loss: 0.0531 - val_acc: 0.9852
Epoch 42/50
60000/60000 [==============================] - 5s 85us/step - loss: 0.0359 -
acc: 0.9877 - val_loss: 0.0569 - val_acc: 0.9843
Epoch 43/50
60000/60000 [==============================] - 5s 86us/step - loss: 0.0321 -
acc: 0.9890 - val_loss: 0.0584 - val_acc: 0.9844
Epoch 44/50
60000/60000 [==============================] - 5s 91us/step - loss: 0.0346 -
acc: 0.9883 - val_loss: 0.0575 - val_acc: 0.9853
Epoch 45/50
60000/60000 [==============================] - 5s 86us/step - loss: 0.0336 -
acc: 0.9889 - val_loss: 0.0576 - val_acc: 0.9856
Epoch 46/50
60000/60000 [==============================] - 5s 85us/step - loss: 0.0360 -
acc: 0.9884 - val_loss: 0.0501 - val_acc: 0.9866
Epoch 47/50
60000/60000 [==============================] - 5s 89us/step - loss: 0.0323 -
acc: 0.9891 - val_loss: 0.0522 - val_acc: 0.9864
Epoch 48/50
60000/60000 [==============================] - 5s 85us/step - loss: 0.0313 -
acc: 0.9896 - val_loss: 0.0557 - val_acc: 0.9850
Epoch 49/50
60000/60000 [==============================] - 5s 85us/step - loss: 0.0318 -
acc: 0.9891 - val_loss: 0.0584 - val_acc: 0.9853
Epoch 50/50
60000/60000 [==============================] - 5s 89us/step - loss: 0.0326 -
acc: 0.9892 - val_loss: 0.0543 - val_acc: 0.9853
```

In [20]:
```
score = model_arch5.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

# list of epoch numbers
x = list(range(1,50+1))

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

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

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to num
ber of epochs

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

```
Test score: 0.054263918649523483
Test accuracy: 0.9853
```



In [21]:
```
w_after = model_arch5.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()
```



ARCHITECTURE 6(512,430,320) : MLP + without Batch-Norm and Dropout(0.7,0.5,0.2) on hidden Layers, 50 epochs and sigmoid on hidden layers

In [22]:
```
# from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout

model_arch6 = Sequential()

model_arch6.add(Dense(512, activation='sigmoid', input_shape=(input_dim,)))
# model_arch5.add(BatchNormalization())
model_arch6.add(Dropout(0.7))

model_arch6.add(Dense(430, activation='sigmoid') )
# model_arch5.add(BatchNormalization())
model_arch6.add(Dropout(0.5))

model_arch6.add(Dense(320, activation='sigmoid') )
# model_arch5.add(BatchNormalization())
model_arch6.add(Dropout(0.2))

model_arch6.add(Dense(output_dim, activation='softmax'))


model_arch6.summary()
```

```
WARNING:tensorflow:Large dropout rate: 0.7 (>0.5). In TensorFlow 2.x, dropout
() uses dropout rate instead of keep_prob. Please ensure that this is intende
d.
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_11 (Dense)             (None, 512)               401920    
_________________________________________________________________
dropout_9 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_12 (Dense)             (None, 430)               220590    
_________________________________________________________________
dropout_10 (Dropout)         (None, 430)               0         
_________________________________________________________________
dense_13 (Dense)             (None, 320)               137920    
_________________________________________________________________
dropout_11 (Dropout)         (None, 320)               0         
_________________________________________________________________
dense_14 (Dense)             (None, 10)                3210      
=================================================================
Total params: 763,640
Trainable params: 763,640
Non-trainable params: 0
_________________________________________________________________
```

In [23]:
```
model_arch6.compile(optimizer='adam', loss='categorical_crossentropy', metrics
=['accuracy'])

history = model_arch6.fit(X_train, Y_train, batch_size=batch_size, epochs=50,
verbose=1, validation_data=(X_test, Y_test))
```

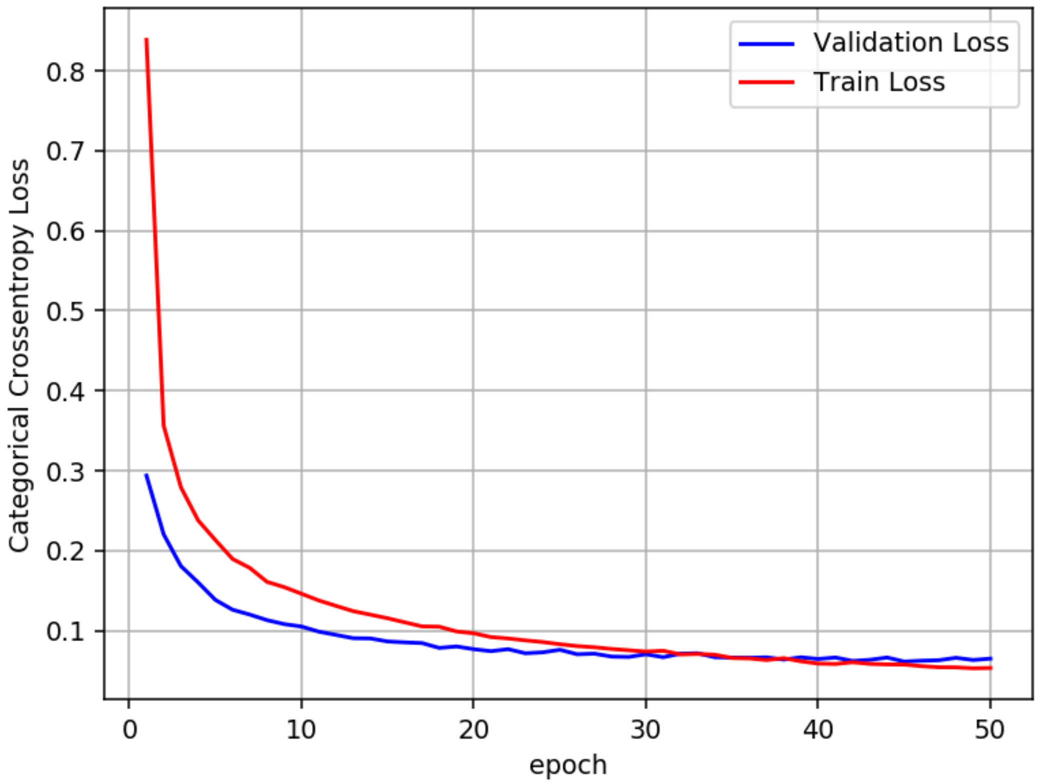
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/50
60000/60000 [==============================] - 5s 80us/step - loss: 0.8386 -
acc: 0.7214 - val_loss: 0.2939 - val_acc: 0.9114
Epoch 2/50
60000/60000 [==============================] - 4s 59us/step - loss: 0.3561 -
acc: 0.8913 - val_loss: 0.2205 - val_acc: 0.9302
Epoch 3/50
60000/60000 [==============================] - 4s 59us/step - loss: 0.2793 -
acc: 0.9145 - val_loss: 0.1808 - val_acc: 0.9436
Epoch 4/50
60000/60000 [==============================] - 4s 59us/step - loss: 0.2379 -
acc: 0.9277 - val_loss: 0.1604 - val_acc: 0.9515
Epoch 5/50
60000/60000 [==============================] - 4s 59us/step - loss: 0.2134 -
acc: 0.9345 - val_loss: 0.1386 - val_acc: 0.9569
Epoch 6/50
60000/60000 [==============================] - 4s 59us/step - loss: 0.1897 -
acc: 0.9415 - val_loss: 0.1263 - val_acc: 0.9616
Epoch 7/50
60000/60000 [==============================] - 4s 60us/step - loss: 0.1785 -
acc: 0.9453 - val_loss: 0.1202 - val_acc: 0.9644
Epoch 8/50
60000/60000 [==============================] - 4s 59us/step - loss: 0.1611 -
acc: 0.9506 - val_loss: 0.1133 - val_acc: 0.9652
Epoch 9/50
60000/60000 [==============================] - 4s 60us/step - loss: 0.1545 -
acc: 0.9530 - val_loss: 0.1083 - val_acc: 0.9685
Epoch 10/50
60000/60000 [==============================] - 4s 60us/step - loss: 0.1463 -
acc: 0.9548 - val_loss: 0.1053 - val_acc: 0.9682
Epoch 11/50
60000/60000 [==============================] - 4s 60us/step - loss: 0.1379 -
acc: 0.9574 - val_loss: 0.0989 - val_acc: 0.9706
Epoch 12/50
60000/60000 [==============================] - 4s 60us/step - loss: 0.1311 -
acc: 0.9594 - val_loss: 0.0948 - val_acc: 0.9712
Epoch 13/50
60000/60000 [==============================] - 4s 60us/step - loss: 0.1245 -
acc: 0.9617 - val_loss: 0.0907 - val_acc: 0.9731
Epoch 14/50
60000/60000 [==============================] - 4s 60us/step - loss: 0.1201 -
acc: 0.9627 - val_loss: 0.0904 - val_acc: 0.9734
Epoch 15/50
60000/60000 [==============================] - 4s 60us/step - loss: 0.1156 -
acc: 0.9635 - val_loss: 0.0866 - val_acc: 0.9737
Epoch 16/50
60000/60000 [==============================] - 4s 60us/step - loss: 0.1105 -
acc: 0.9650 - val_loss: 0.0855 - val_acc: 0.9748
Epoch 17/50
60000/60000 [==============================] - 4s 61us/step - loss: 0.1055 -
acc: 0.9669 - val_loss: 0.0846 - val_acc: 0.9741
Epoch 18/50
60000/60000 [==============================] - 4s 60us/step - loss: 0.1051 -
acc: 0.9678 - val_loss: 0.0785 - val_acc: 0.9767
Epoch 19/50
60000/60000 [==============================] - 4s 63us/step - loss: 0.0991 -
```

```
acc: 0.9689 - val_loss: 0.0804 - val_acc: 0.9766
Epoch 20/50
60000/60000 [==============================] - 4s 60us/step - loss: 0.0968 -
acc: 0.9700 - val_loss: 0.0770 - val_acc: 0.9765
Epoch 21/50
60000/60000 [==============================] - 4s 61us/step - loss: 0.0921 -
acc: 0.9713 - val_loss: 0.0745 - val_acc: 0.9767
Epoch 22/50
60000/60000 [==============================] - 4s 60us/step - loss: 0.0904 -
acc: 0.9712 - val_loss: 0.0770 - val_acc: 0.9768
Epoch 23/50
60000/60000 [==============================] - 4s 60us/step - loss: 0.0880 -
acc: 0.9721 - val_loss: 0.0718 - val_acc: 0.9793
Epoch 24/50
60000/60000 [==============================] - 4s 60us/step - loss: 0.0859 -
acc: 0.9731 - val_loss: 0.0730 - val_acc: 0.9788
Epoch 25/50
60000/60000 [==============================] - 4s 60us/step - loss: 0.0832 -
acc: 0.9745 - val_loss: 0.0762 - val_acc: 0.9783
Epoch 26/50
60000/60000 [==============================] - 4s 60us/step - loss: 0.0808 -
acc: 0.9751 - val_loss: 0.0706 - val_acc: 0.9788
Epoch 27/50
60000/60000 [==============================] - 4s 64us/step - loss: 0.0794 -
acc: 0.9755 - val_loss: 0.0714 - val_acc: 0.9788
Epoch 28/50
60000/60000 [==============================] - 4s 61us/step - loss: 0.0773 -
acc: 0.9753 - val_loss: 0.0677 - val_acc: 0.9802
Epoch 29/50
60000/60000 [==============================] - 4s 60us/step - loss: 0.0756 -
acc: 0.9765 - val_loss: 0.0672 - val_acc: 0.9798
Epoch 30/50
60000/60000 [==============================] - 4s 61us/step - loss: 0.0740 -
acc: 0.9766 - val_loss: 0.0707 - val_acc: 0.9783
Epoch 31/50
60000/60000 [==============================] - 4s 60us/step - loss: 0.0750 -
acc: 0.9760 - val_loss: 0.0669 - val_acc: 0.9799
Epoch 32/50
60000/60000 [==============================] - 4s 60us/step - loss: 0.0704 -
acc: 0.9776 - val_loss: 0.0713 - val_acc: 0.9789
Epoch 33/50
60000/60000 [==============================] - 4s 61us/step - loss: 0.0710 -
acc: 0.9775 - val_loss: 0.0719 - val_acc: 0.9801
Epoch 34/50
60000/60000 [==============================] - 4s 63us/step - loss: 0.0700 -
acc: 0.9780 - val_loss: 0.0669 - val_acc: 0.9807
Epoch 35/50
60000/60000 [==============================] - 4s 62us/step - loss: 0.0659 -
acc: 0.9787 - val_loss: 0.0665 - val_acc: 0.9800
Epoch 36/50
60000/60000 [==============================] - 4s 62us/step - loss: 0.0654 -
acc: 0.9785 - val_loss: 0.0662 - val_acc: 0.9799
Epoch 37/50
60000/60000 [==============================] - 4s 62us/step - loss: 0.0634 -
acc: 0.9794 - val_loss: 0.0668 - val_acc: 0.9808
Epoch 38/50
60000/60000 [==============================] - 4s 62us/step - loss: 0.0657 -
```

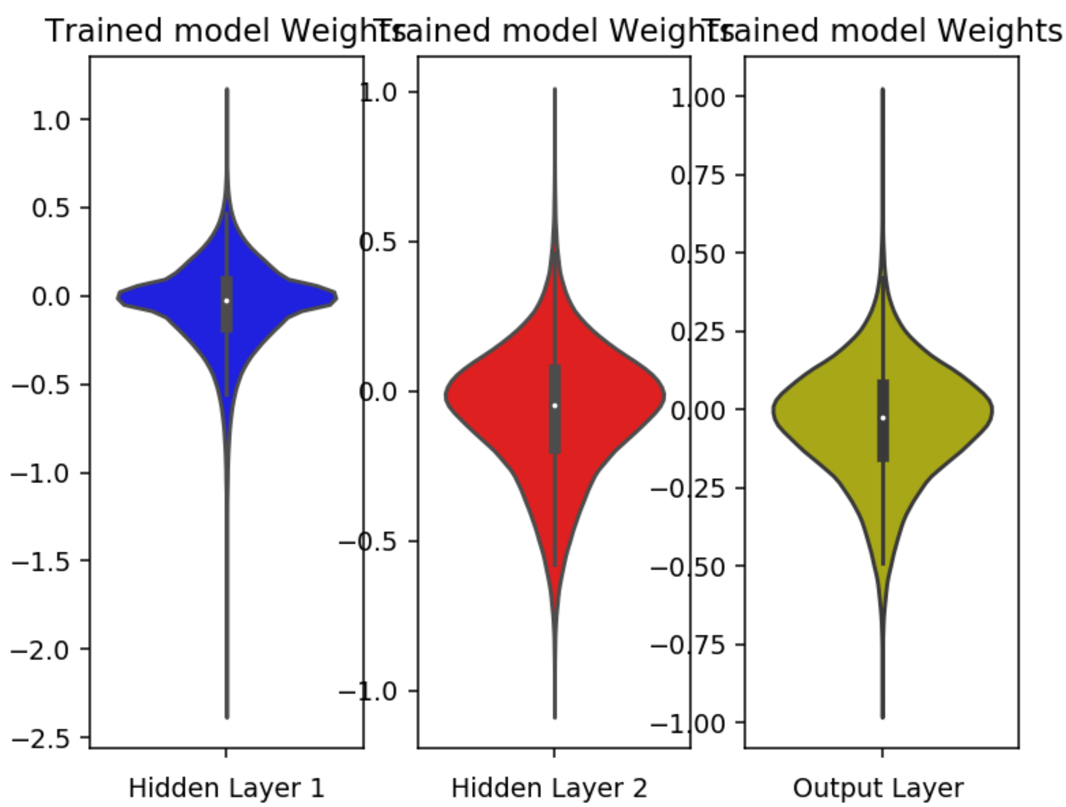
```
acc: 0.9789 - val_loss: 0.0639 - val_acc: 0.9810
Epoch 39/50
60000/60000 [==============================] - 4s 62us/step - loss: 0.0618 -
acc: 0.9799 - val_loss: 0.0668 - val_acc: 0.9813
Epoch 40/50
60000/60000 [==============================] - 4s 62us/step - loss: 0.0590 -
acc: 0.9811 - val_loss: 0.0647 - val_acc: 0.9823
Epoch 41/50
60000/60000 [==============================] - 4s 62us/step - loss: 0.0585 -
acc: 0.9805 - val_loss: 0.0666 - val_acc: 0.9813
Epoch 42/50
60000/60000 [==============================] - 4s 63us/step - loss: 0.0606 -
acc: 0.9804 - val_loss: 0.0622 - val_acc: 0.9821
Epoch 43/50
60000/60000 [==============================] - 4s 62us/step - loss: 0.0586 -
acc: 0.9817 - val_loss: 0.0636 - val_acc: 0.9817
Epoch 44/50
60000/60000 [==============================] - 4s 62us/step - loss: 0.0580 -
acc: 0.9817 - val_loss: 0.0666 - val_acc: 0.9815
Epoch 45/50
60000/60000 [==============================] - 4s 62us/step - loss: 0.0577 -
acc: 0.9819 - val_loss: 0.0615 - val_acc: 0.9826
Epoch 46/50
60000/60000 [==============================] - 4s 62us/step - loss: 0.0558 -
acc: 0.9821 - val_loss: 0.0626 - val_acc: 0.9821
Epoch 47/50
60000/60000 [==============================] - 4s 63us/step - loss: 0.0544 -
acc: 0.9826 - val_loss: 0.0631 - val_acc: 0.9819
Epoch 48/50
60000/60000 [==============================] - 4s 62us/step - loss: 0.0542 -
acc: 0.9826 - val_loss: 0.0662 - val_acc: 0.9813
Epoch 49/50
60000/60000 [==============================] - 4s 62us/step - loss: 0.0531 -
acc: 0.9827 - val_loss: 0.0634 - val_acc: 0.9821
Epoch 50/50
60000/60000 [==============================] - 4s 62us/step - loss: 0.0535 -
acc: 0.9824 - val_loss: 0.0651 - val_acc: 0.9823
```

In [24]:
```
score = model_arch6.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

# list of epoch numbers
x = list(range(1,50+1))

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

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

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to num
ber of epochs

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

```
Test score: 0.06514910679099849
Test accuracy: 0.9823
```



In [25]:
```
w_after = model_arch6.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()
```



ARCHITECTURE 7(624,430) : MLP + without Batch-Norm and Dropout(0.6,0.3) on hidden Layers, 100 epochs with Relu in hidden layers

In [26]:

```
# from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout

model_arch7 = Sequential()

model_arch7.add(Dense(624, activation='relu', input_shape=(input_dim,), kernel
_initializer=RandomNormal(mean=0.0, stddev=0.056, seed=None)))
# model_arch7.add(BatchNormalization())
model_arch7.add(Dropout(0.6))

model_arch7.add(Dense(430, activation='relu', kernel_initializer=RandomNormal(
mean=0.0, stddev=0.068, seed=None)) )
# model_arch7.add(BatchNormalization())
model_arch7.add(Dropout(0.3))

model_arch7.add(Dense(output_dim, activation='softmax'))


model_arch7.summary()
```

```
WARNING:tensorflow:Large dropout rate: 0.6 (>0.5). In TensorFlow 2.x, dropout
() uses dropout rate instead of keep_prob. Please ensure that this is intende
d.
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_15 (Dense)             (None, 624)               489840    
_________________________________________________________________
dropout_12 (Dropout)         (None, 624)               0         
_________________________________________________________________
dense_16 (Dense)             (None, 430)               268750    
_________________________________________________________________
dropout_13 (Dropout)         (None, 430)               0         
_________________________________________________________________
dense_17 (Dense)             (None, 10)                4310      
=================================================================
Total params: 762,900
Trainable params: 762,900
Non-trainable params: 0
_________________________________________________________________
```

In [27]:
```
model_arch7.compile(optimizer='adam', loss='categorical_crossentropy', metrics
=['accuracy'])

history = model_arch7.fit(X_train, Y_train, batch_size=batch_size, epochs=100,
verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/100
60000/60000 [==============================] - 5s 80us/step - loss: 0.3829 -
acc: 0.8818 - val_loss: 0.1313 - val_acc: 0.9583
Epoch 2/100
60000/60000 [==============================] - 4s 59us/step - loss: 0.1808 -
acc: 0.9444 - val_loss: 0.0982 - val_acc: 0.9680
Epoch 3/100
60000/60000 [==============================] - 4s 59us/step - loss: 0.1444 -
acc: 0.9565 - val_loss: 0.0859 - val_acc: 0.9736
Epoch 4/100
60000/60000 [==============================] - 4s 60us/step - loss: 0.1241 -
acc: 0.9610 - val_loss: 0.0720 - val_acc: 0.9782
Epoch 5/100
60000/60000 [==============================] - 4s 61us/step - loss: 0.1091 -
acc: 0.9658 - val_loss: 0.0658 - val_acc: 0.9794
Epoch 6/100
60000/60000 [==============================] - 4s 60us/step - loss: 0.0998 -
acc: 0.9685 - val_loss: 0.0711 - val_acc: 0.9783
Epoch 7/100
60000/60000 [==============================] - 4s 60us/step - loss: 0.0926 -
acc: 0.9713 - val_loss: 0.0667 - val_acc: 0.9802
Epoch 8/100
60000/60000 [==============================] - 4s 59us/step - loss: 0.0833 -
acc: 0.9737 - val_loss: 0.0702 - val_acc: 0.9783
Epoch 9/100
60000/60000 [==============================] - 4s 59us/step - loss: 0.0791 -
acc: 0.9752 - val_loss: 0.0636 - val_acc: 0.9815
Epoch 10/100
60000/60000 [==============================] - 4s 60us/step - loss: 0.0778 -
acc: 0.9753 - val_loss: 0.0626 - val_acc: 0.9810
Epoch 11/100
60000/60000 [==============================] - 4s 61us/step - loss: 0.0733 -
acc: 0.9759 - val_loss: 0.0630 - val_acc: 0.9805
Epoch 12/100
60000/60000 [==============================] - 4s 60us/step - loss: 0.0682 -
acc: 0.9779 - val_loss: 0.0622 - val_acc: 0.9822
Epoch 13/100
60000/60000 [==============================] - 4s 63us/step - loss: 0.0645 -
acc: 0.9795 - val_loss: 0.0598 - val_acc: 0.9816
Epoch 14/100
60000/60000 [==============================] - 4s 59us/step - loss: 0.0645 -
acc: 0.9799 - val_loss: 0.0563 - val_acc: 0.9839
Epoch 15/100
60000/60000 [==============================] - 4s 61us/step - loss: 0.0609 -
acc: 0.9801 - val_loss: 0.0570 - val_acc: 0.9826
Epoch 16/100
60000/60000 [==============================] - 4s 59us/step - loss: 0.0601 -
acc: 0.9814 - val_loss: 0.0579 - val_acc: 0.9831
Epoch 17/100
60000/60000 [==============================] - 4s 59us/step - loss: 0.0586 -
acc: 0.9817 - val_loss: 0.0574 - val_acc: 0.9830
Epoch 18/100
60000/60000 [==============================] - 4s 60us/step - loss: 0.0539 -
acc: 0.9826 - val_loss: 0.0531 - val_acc: 0.9852
Epoch 19/100
60000/60000 [==============================] - 4s 60us/step - loss: 0.0539 -
```

```
acc: 0.9828 - val_loss: 0.0594 - val_acc: 0.9829
Epoch 20/100
60000/60000 [==============================] - 4s 60us/step - loss: 0.0527 -
acc: 0.9830 - val_loss: 0.0568 - val_acc: 0.9836
Epoch 21/100
60000/60000 [==============================] - 4s 59us/step - loss: 0.0518 -
acc: 0.9832 - val_loss: 0.0581 - val_acc: 0.9836
Epoch 22/100
60000/60000 [==============================] - 4s 64us/step - loss: 0.0504 -
acc: 0.9841 - val_loss: 0.0562 - val_acc: 0.9834
Epoch 23/100
60000/60000 [==============================] - 4s 70us/step - loss: 0.0501 -
acc: 0.9835 - val_loss: 0.0522 - val_acc: 0.9856
Epoch 24/100
60000/60000 [==============================] - 3s 58us/step - loss: 0.0495 -
acc: 0.9845 - val_loss: 0.0541 - val_acc: 0.9840
Epoch 25/100
60000/60000 [==============================] - 4s 73us/step - loss: 0.0487 -
acc: 0.9848 - val_loss: 0.0492 - val_acc: 0.9855
Epoch 26/100
60000/60000 [==============================] - 4s 73us/step - loss: 0.0472 -
acc: 0.9847 - val_loss: 0.0550 - val_acc: 0.9848
Epoch 27/100
60000/60000 [==============================] - 4s 72us/step - loss: 0.0479 -
acc: 0.9845 - val_loss: 0.0576 - val_acc: 0.9840
Epoch 28/100
60000/60000 [==============================] - 4s 71us/step - loss: 0.0478 -
acc: 0.9849 - val_loss: 0.0552 - val_acc: 0.9856
Epoch 29/100
60000/60000 [==============================] - 4s 68us/step - loss: 0.0445 -
acc: 0.9854 - val_loss: 0.0552 - val_acc: 0.9847
Epoch 30/100
60000/60000 [==============================] - 4s 69us/step - loss: 0.0424 -
acc: 0.9864 - val_loss: 0.0603 - val_acc: 0.9842
Epoch 31/100
60000/60000 [==============================] - 4s 59us/step - loss: 0.0388 -
acc: 0.9872 - val_loss: 0.0556 - val_acc: 0.9841
Epoch 32/100
60000/60000 [==============================] - 4s 60us/step - loss: 0.0431 -
acc: 0.9860 - val_loss: 0.0535 - val_acc: 0.9855
Epoch 33/100
60000/60000 [==============================] - 4s 60us/step - loss: 0.0406 -
acc: 0.9870 - val_loss: 0.0598 - val_acc: 0.9845
Epoch 34/100
60000/60000 [==============================] - 4s 60us/step - loss: 0.0402 -
acc: 0.9872 - val_loss: 0.0607 - val_acc: 0.9841
Epoch 35/100
60000/60000 [==============================] - 4s 60us/step - loss: 0.0399 -
acc: 0.9872 - val_loss: 0.0538 - val_acc: 0.9854
Epoch 36/100
60000/60000 [==============================] - 4s 59us/step - loss: 0.0395 -
acc: 0.9878 - val_loss: 0.0514 - val_acc: 0.9852
Epoch 37/100
60000/60000 [==============================] - 4s 60us/step - loss: 0.0396 -
acc: 0.9874 - val_loss: 0.0549 - val_acc: 0.9860
Epoch 38/100
60000/60000 [==============================] - 4s 60us/step - loss: 0.0397 -
```

```
acc: 0.9875 - val_loss: 0.0573 - val_acc: 0.9853
Epoch 39/100
60000/60000 [==============================] - 4s 60us/step - loss: 0.0364 -
acc: 0.9890 - val_loss: 0.0627 - val_acc: 0.9843
Epoch 40/100
60000/60000 [==============================] - 4s 60us/step - loss: 0.0380 -
acc: 0.9880 - val_loss: 0.0534 - val_acc: 0.9859
Epoch 41/100
60000/60000 [==============================] - 4s 60us/step - loss: 0.0393 -
acc: 0.9873 - val_loss: 0.0520 - val_acc: 0.9846
Epoch 42/100
60000/60000 [==============================] - 4s 61us/step - loss: 0.0341 -
acc: 0.9890 - val_loss: 0.0553 - val_acc: 0.9854
Epoch 43/100
60000/60000 [==============================] - 4s 62us/step - loss: 0.0377 -
acc: 0.9881 - val_loss: 0.0583 - val_acc: 0.9843
Epoch 44/100
60000/60000 [==============================] - 4s 62us/step - loss: 0.0362 -
acc: 0.9885 - val_loss: 0.0563 - val_acc: 0.9853
Epoch 45/100
60000/60000 [==============================] - 4s 64us/step - loss: 0.0333 -
acc: 0.9896 - val_loss: 0.0586 - val_acc: 0.9851
Epoch 46/100
60000/60000 [==============================] - 4s 62us/step - loss: 0.0355 -
acc: 0.9892 - val_loss: 0.0599 - val_acc: 0.9858
Epoch 47/100
60000/60000 [==============================] - 4s 62us/step - loss: 0.0351 -
acc: 0.9890 - val_loss: 0.0579 - val_acc: 0.9844
Epoch 48/100
60000/60000 [==============================] - 4s 62us/step - loss: 0.0355 -
acc: 0.9894 - val_loss: 0.0581 - val_acc: 0.9853
Epoch 49/100
60000/60000 [==============================] - 4s 62us/step - loss: 0.0363 -
acc: 0.9885 - val_loss: 0.0595 - val_acc: 0.9838
Epoch 50/100
60000/60000 [==============================] - 4s 68us/step - loss: 0.0356 -
acc: 0.9889 - val_loss: 0.0649 - val_acc: 0.9839
Epoch 51/100
60000/60000 [==============================] - 4s 66us/step - loss: 0.0352 -
acc: 0.9888 - val_loss: 0.0658 - val_acc: 0.9838
Epoch 52/100
60000/60000 [==============================] - 4s 71us/step - loss: 0.0342 -
acc: 0.9893 - val_loss: 0.0573 - val_acc: 0.9848
Epoch 53/100
60000/60000 [==============================] - 4s 67us/step - loss: 0.0345 -
acc: 0.9891 - val_loss: 0.0624 - val_acc: 0.9845
Epoch 54/100
60000/60000 [==============================] - 4s 65us/step - loss: 0.0330 -
acc: 0.9898 - val_loss: 0.0608 - val_acc: 0.9852
Epoch 55/100
60000/60000 [==============================] - 4s 63us/step - loss: 0.0317 -
acc: 0.9902 - val_loss: 0.0600 - val_acc: 0.9846
Epoch 56/100
60000/60000 [==============================] - 4s 65us/step - loss: 0.0305 -
acc: 0.9906 - val_loss: 0.0591 - val_acc: 0.9856
Epoch 57/100
60000/60000 [==============================] - 4s 63us/step - loss: 0.0354 -
```

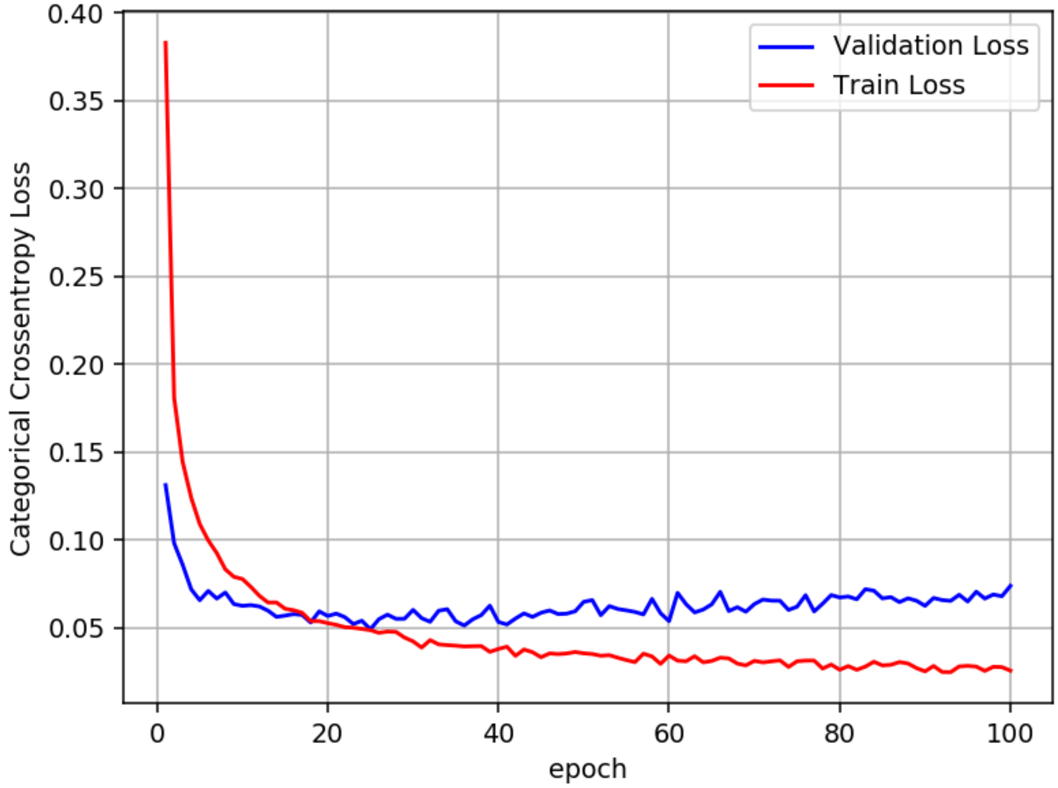
```
acc: 0.9890 - val_loss: 0.0577 - val_acc: 0.9846
Epoch 58/100
60000/60000 [==============================] - 4s 62us/step - loss: 0.0338 -
acc: 0.9898 - val_loss: 0.0665 - val_acc: 0.9834
Epoch 59/100
60000/60000 [==============================] - 4s 61us/step - loss: 0.0296 -
acc: 0.9906 - val_loss: 0.0584 - val_acc: 0.9844
Epoch 60/100
60000/60000 [==============================] - 4s 63us/step - loss: 0.0343 -
acc: 0.9892 - val_loss: 0.0539 - val_acc: 0.9856
Epoch 61/100
60000/60000 [==============================] - 4s 61us/step - loss: 0.0314 -
acc: 0.9906 - val_loss: 0.0699 - val_acc: 0.9834
Epoch 62/100
60000/60000 [==============================] - 4s 60us/step - loss: 0.0311 -
acc: 0.9905 - val_loss: 0.0635 - val_acc: 0.9847
Epoch 63/100
60000/60000 [==============================] - 4s 60us/step - loss: 0.0339 -
acc: 0.9899 - val_loss: 0.0588 - val_acc: 0.9852
Epoch 64/100
60000/60000 [==============================] - 4s 63us/step - loss: 0.0304 -
acc: 0.9908 - val_loss: 0.0605 - val_acc: 0.9853
Epoch 65/100
60000/60000 [==============================] - 4s 62us/step - loss: 0.0312 -
acc: 0.9905 - val_loss: 0.0634 - val_acc: 0.9846
Epoch 66/100
60000/60000 [==============================] - 4s 61us/step - loss: 0.0331 -
acc: 0.9904 - val_loss: 0.0705 - val_acc: 0.9842
Epoch 67/100
60000/60000 [==============================] - 4s 61us/step - loss: 0.0327 -
acc: 0.9903 - val_loss: 0.0596 - val_acc: 0.9850
Epoch 68/100
60000/60000 [==============================] - 4s 61us/step - loss: 0.0297 -
acc: 0.9908 - val_loss: 0.0618 - val_acc: 0.9854
Epoch 69/100
60000/60000 [==============================] - 4s 61us/step - loss: 0.0287 -
acc: 0.9919 - val_loss: 0.0592 - val_acc: 0.9860
Epoch 70/100
60000/60000 [==============================] - 4s 61us/step - loss: 0.0313 -
acc: 0.9907 - val_loss: 0.0636 - val_acc: 0.9844
Epoch 71/100
60000/60000 [==============================] - 4s 61us/step - loss: 0.0304 -
acc: 0.9910 - val_loss: 0.0661 - val_acc: 0.9851
Epoch 72/100
60000/60000 [==============================] - 4s 61us/step - loss: 0.0310 -
acc: 0.9911 - val_loss: 0.0655 - val_acc: 0.9855
Epoch 73/100
60000/60000 [==============================] - 4s 61us/step - loss: 0.0315 -
acc: 0.9912 - val_loss: 0.0655 - val_acc: 0.9851
Epoch 74/100
60000/60000 [==============================] - 4s 62us/step - loss: 0.0278 -
acc: 0.9919 - val_loss: 0.0603 - val_acc: 0.9864
Epoch 75/100
60000/60000 [==============================] - 4s 61us/step - loss: 0.0310 -
acc: 0.9912 - val_loss: 0.0621 - val_acc: 0.9847
Epoch 76/100
60000/60000 [==============================] - 4s 62us/step - loss: 0.0314 -
```

```
acc: 0.9913 - val_loss: 0.0686 - val_acc: 0.9850
Epoch 77/100
60000/60000 [==============================] - 4s 62us/step - loss: 0.0314 -
acc: 0.9907 - val_loss: 0.0594 - val_acc: 0.9860
Epoch 78/100
60000/60000 [==============================] - 4s 62us/step - loss: 0.0269 -
acc: 0.9921 - val_loss: 0.0638 - val_acc: 0.9874
Epoch 79/100
60000/60000 [==============================] - 4s 63us/step - loss: 0.0292 -
acc: 0.9916 - val_loss: 0.0687 - val_acc: 0.9847
Epoch 80/100
60000/60000 [==============================] - 4s 61us/step - loss: 0.0262 -
acc: 0.9920 - val_loss: 0.0673 - val_acc: 0.9855
Epoch 81/100
60000/60000 [==============================] - 4s 61us/step - loss: 0.0283 -
acc: 0.9914 - val_loss: 0.0679 - val_acc: 0.9853
Epoch 82/100
60000/60000 [==============================] - 4s 61us/step - loss: 0.0262 -
acc: 0.9922 - val_loss: 0.0663 - val_acc: 0.9840
Epoch 83/100
60000/60000 [==============================] - 4s 61us/step - loss: 0.0280 -
acc: 0.9917 - val_loss: 0.0720 - val_acc: 0.9848
Epoch 84/100
60000/60000 [==============================] - 4s 62us/step - loss: 0.0308 -
acc: 0.9911 - val_loss: 0.0712 - val_acc: 0.9853
Epoch 85/100
60000/60000 [==============================] - 4s 61us/step - loss: 0.0287 -
acc: 0.9916 - val_loss: 0.0668 - val_acc: 0.9850
Epoch 86/100
60000/60000 [==============================] - 4s 64us/step - loss: 0.0290 -
acc: 0.9916 - val_loss: 0.0675 - val_acc: 0.9852
Epoch 87/100
60000/60000 [==============================] - 4s 62us/step - loss: 0.0306 -
acc: 0.9914 - val_loss: 0.0647 - val_acc: 0.9862
Epoch 88/100
60000/60000 [==============================] - 4s 61us/step - loss: 0.0298 -
acc: 0.9917 - val_loss: 0.0669 - val_acc: 0.9856
Epoch 89/100
60000/60000 [==============================] - 4s 62us/step - loss: 0.0272 -
acc: 0.9925 - val_loss: 0.0655 - val_acc: 0.9849
Epoch 90/100
60000/60000 [==============================] - 4s 61us/step - loss: 0.0253 -
acc: 0.9930 - val_loss: 0.0626 - val_acc: 0.9861
Epoch 91/100
60000/60000 [==============================] - 4s 61us/step - loss: 0.0284 -
acc: 0.9919 - val_loss: 0.0670 - val_acc: 0.9858
Epoch 92/100
60000/60000 [==============================] - 4s 62us/step - loss: 0.0249 -
acc: 0.9928 - val_loss: 0.0659 - val_acc: 0.9852
Epoch 93/100
60000/60000 [==============================] - 4s 62us/step - loss: 0.0249 -
acc: 0.9926 - val_loss: 0.0654 - val_acc: 0.9866
Epoch 94/100
60000/60000 [==============================] - 4s 62us/step - loss: 0.0281 -
acc: 0.9919 - val_loss: 0.0689 - val_acc: 0.9845
Epoch 95/100
60000/60000 [==============================] - 4s 65us/step - loss: 0.0284 -
```

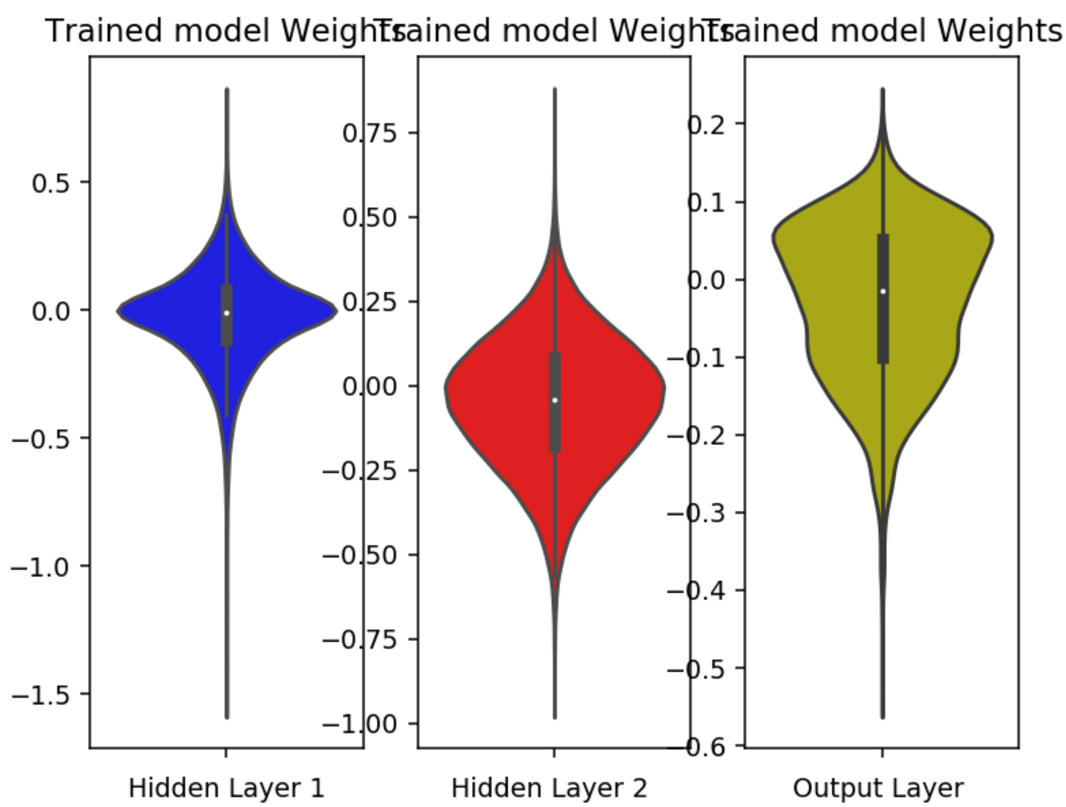
```
acc: 0.9920 - val_loss: 0.0650 - val_acc: 0.9860
Epoch 96/100
60000/60000 [==============================] - 4s 60us/step - loss: 0.0280 -
acc: 0.9918 - val_loss: 0.0706 - val_acc: 0.9858
Epoch 97/100
60000/60000 [==============================] - 4s 61us/step - loss: 0.0256 -
acc: 0.9927 - val_loss: 0.0667 - val_acc: 0.9852
Epoch 98/100
60000/60000 [==============================] - 4s 66us/step - loss: 0.0279 -
acc: 0.9918 - val_loss: 0.0690 - val_acc: 0.9860
Epoch 99/100
60000/60000 [==============================] - 4s 61us/step - loss: 0.0277 -
acc: 0.9923 - val_loss: 0.0680 - val_acc: 0.9847
Epoch 100/100
60000/60000 [==============================] - 4s 61us/step - loss: 0.0257 -
acc: 0.9924 - val_loss: 0.0739 - val_acc: 0.9838
```

In [28]:
```
score = model_arch7.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

# list of epoch numbers
x = list(range(1,100+1))

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

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

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to num
ber of epochs

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

```
Test score: 0.07392368958264603
Test accuracy: 0.9838
```



In [29]:
```
w_after = model_arch7.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()
```



In [ ]:

Results(Pretty Table)

In [32]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Architecture No.","Layers","Dropout rate", "Test loss", "Tes
t Accuracy","Epochs","Batch Norm or not"]
x.add_row(["1","(624,430)", "(0.5,0.5)" ,"0.052", "0.985", "20","YES"])
x.add_row(["2","(512,364,58)", "(0.5,0.5,0.5)" ,"0.060", "0.981", "20","YES"])
x.add_row(["3","(584,452,312,256,128)", "(0.5,0.5,0.5,0.5,0.5)" ,"0.064", "0.9
82", "20","YES"])
x.add_row(["4","(682,452,312,256,128,64)", "(0.6,0.3,0.3,0.5,0.3,0.2)" ,"0.06
2", "0.983", "20","YES"])
x.add_row(["5","(624,430)", "(0.6,0.5)" ,"0.054", "0.985", "50","YES"])
x.add_row(["6","(512,430,320)", "(0.7,0.5,0.2)" ,"0.065", "0.982", "50","NO"])
x.add_row(["7","(624,430)", "(0.6,0.3)" ,"0.073", "0.983", "100","NO"])
print(x)
```

Architecture No.	Layers	Dropout rate	Test loss	Test Accuracy	Epochs	Batch Norm or not
1	(624,430)	(0.5,0.5)	0.052	0.985	20	YES
2	(512,364,58)	(0.5,0.5,0.5)	0.060	0.981	20	YES
3	(584,452,312,256,128)	(0.5,0.5,0.5,0.5,0.5)	0.064	0.982	20	YES
4	(682,452,312,256,128,64)	(0.6,0.3,0.3,0.5,0.3,0.2)	0.062	0.983	20	YES
5	(624,430)	(0.6,0.5)	0.054	0.985	50	YES
6	(512,430,320)	(0.7,0.5,0.2)	0.065	0.982	50	NO
7	(624,430)	(0.6,0.3)	0.073	0.983	100	NO

Conclusion :

1. As you can see from the above table , i ran the first 4 models for 20 epochs, got the best test score for the 1st model.
2. But for 5th and 6th architecture i ran for 50 epochs to see if there might be an improvement, but the highest i could get is 98.5%
3. I even didn't used Batch normalization for the 6th model but got good score for it
4. Now for the 7th architecture i used 100 epochs with no Batch normalization , but didn't improved much
5. So, for MNIST dataset, i think it is better to have 2-3 hidden layers instead of complex network
6. I have made the above(5th) conclution because of the 1st model

In [ ]: