Keras: MLP architectures on MNIST dataset

In [0]:
```
# If your keras is not using tensorflow as backend set "KERAS_BACKEND=t
ensorflow" use this command
from keras.utils import np_utils
from keras.datasets import mnist
import seaborn as sns
from keras.initializers import RandomNormal
```

In [0]:
```
%matplotlib notebook
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import time
# https://gist.github.com/greydanus/f6eee59eaf1d90fcb3b534a25362cea4
# https://stackoverflow.com/a/14434334
# this function is used to update the plots for each epoch and error
def plt_dynamic(x, vy, ty, ax, color=['b']):
  ax.plot(x, vy, 'b', label="Validation Loss")
  ax.plot(x, ty, 'r', label="Train Loss")
  plt.legend()
  plt.grid()
  print('\n')
  plt.title("Epoch Vs Categorical Cross Entropy Loss")
  fig.canvas.draw()
```

In [0]:
```
# The data shuffled and split between train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

In [56]:
```
print("Number of training examples :", X_train.shape[0], "and each imag
e is of shape (%d, %d)" %(X_train.shape[1], X_train.shape[2]))
print("Number of test 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 test examples : 10000 and each image is of shape (28, 28)
```

In [0]:

```
# Ff 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.sh
ape[2])
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1]*X_test.shape[2
])
```

In [58]:

```
# after converting the input images from 3d to 2d vectors

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

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

In [59]:

```
# An example data point
print(X_train[0])
```

```
[   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
```

```
0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   3  18  18  18 126 136 175  26 166 25
5
 247 127   0   0   0   0   0   0   0   0   0   0   0   0  30  36  94 15
4
 170 253 253 253 253 253 225 172 253 242 195  64   0   0   0   0   0
0
   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251  93  8
2
  82  56  39   0   0   0   0   0   0   0   0   0   0   0   0  18 219 25
3
 253 253 253 253 198 182 247 241   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 15
4
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0  14   1 154 253  90   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0   0   0   0 139 253 190   2
0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0  11 190 253  70   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  35 24
1
 225 160 108   1   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0  81 240 253 253 119  25   0   0
0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0  45 186 253 253 150  27   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252 253 18
7
```

```
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0 249 253 249  64   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 25
3
 253 207   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0  39 148 229 253 253 253 250 182   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 25
3
 253 201  78   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0  23  66 213 253 253 253 253 198  81   2   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0  18 171 219 253 253 253 253 19
5
  80   9   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
  55 172 226 253 253 253 253 244 133  11   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0 136 253 253 253 212 135 132   1
6
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0]
```

In [0]:

```
# Data Normalization
# if we observe the above matrix each cell is having a value between 0-
255
```

```
# before we move to apply machine learning algorithms lets try to norma
lize the data
# X => (X - Xmin)/(Xmax-Xmin) = X/255

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

In [61]:
```
# example data point after normlization
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 [62]:

```
# 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.]
```

1 (a) MLP + ReLU + ADAM with 2 hidden layers

In [0]:
```
# https://keras.io/activations/
# Activations can either be used through an Activation layer, or throug
h the activation argument supported by all forward layers:
from keras.models import Sequential
from keras.layers import Dense, Activation
```

In [0]:
```
# Some model parameters
output_dim = 10
input_dim = X_train.shape[1]
batch_size = 128
nb_epoch = 20
```

In [65]:
```
model_relu = Sequential()

# for relu layers
# If we sample weights from a normal distribution N(0,σ) we satisfy thi
s condition with σ=√(2/(ni).
# h1 =>  σ=√(2/(fan_in) = 0.073  => N(0,σ) = N(0,0.073)
# h2 =>  σ=√(2/(fan_in) = 0.133  => N(0,σ) = N(0,0.133)

model_relu.add(Dense(368, activation='relu', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.073, seed=None)))
model_relu.add(Dense(112, activation='relu',kernel_initializer=RandomNo
rmal(mean =0.0, stddev=0.133, seed=None)))
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())
model_relu.compile(optimizer='adam', loss='categorical_crossentropy', m
etrics=['accuracy'])
history = model_relu.fit(X_train, Y_train, batch_size = batch_size, epo
chs = nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_53 (Dense)             (None, 368)               288880
_________________________________________________________________
dense_54 (Dense)             (None, 112)               41328
_________________________________________________________________
dense_55 (Dense)             (None, 10)                1130
=================================================================
Total params: 331,338
Trainable params: 331,338
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.2
490 - acc: 0.9274 - val_loss: 0.1333 - val_acc: 0.9576
Epoch 2/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.0
956 - acc: 0.9712 - val_loss: 0.0874 - val_acc: 0.9723
Epoch 3/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.0
618 - acc: 0.9815 - val_loss: 0.0769 - val_acc: 0.9763
Epoch 4/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.0
417 - acc: 0.9872 - val_loss: 0.0746 - val_acc: 0.9783
Epoch 5/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.0
308 - acc: 0.9904 - val_loss: 0.0828 - val_acc: 0.9756
Epoch 6/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.0
227 - acc: 0.9933 - val_loss: 0.0849 - val_acc: 0.9762
Epoch 7/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.0
189 - acc: 0.9941 - val_loss: 0.0768 - val_acc: 0.9777
Epoch 8/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.0
186 - acc: 0.9935 - val_loss: 0.0769 - val_acc: 0.9800
```

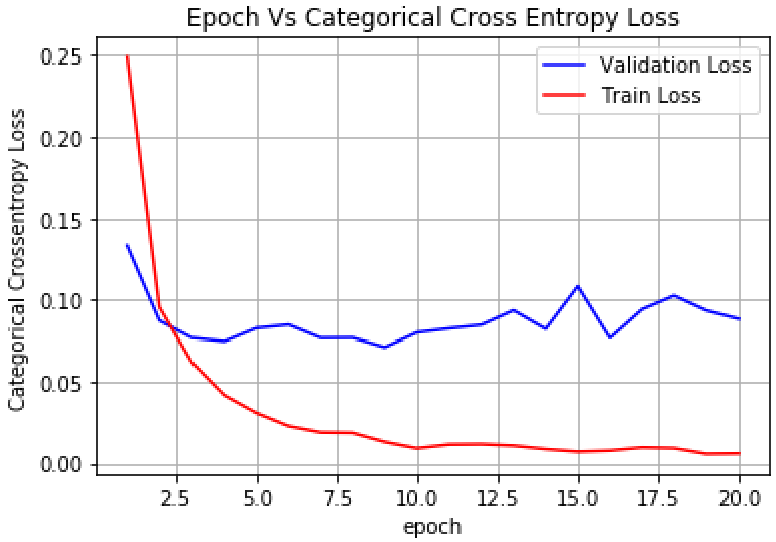
```
Epoch 9/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.0
130 - acc: 0.9960 - val_loss: 0.0709 - val_acc: 0.9818
Epoch 10/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.0
091 - acc: 0.9970 - val_loss: 0.0801 - val_acc: 0.9788
Epoch 11/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.0
115 - acc: 0.9962 - val_loss: 0.0826 - val_acc: 0.9786
Epoch 12/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.0
116 - acc: 0.9958 - val_loss: 0.0847 - val_acc: 0.9786
Epoch 13/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.0
107 - acc: 0.9963 - val_loss: 0.0936 - val_acc: 0.9795
Epoch 14/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.0
086 - acc: 0.9973 - val_loss: 0.0822 - val_acc: 0.9815
Epoch 15/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.0
070 - acc: 0.9978 - val_loss: 0.1083 - val_acc: 0.9773
Epoch 16/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.0
077 - acc: 0.9977 - val_loss: 0.0766 - val_acc: 0.9831
Epoch 17/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.0
096 - acc: 0.9967 - val_loss: 0.0942 - val_acc: 0.9796
Epoch 18/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.0
092 - acc: 0.9970 - val_loss: 0.1026 - val_acc: 0.9811
Epoch 19/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.0
057 - acc: 0.9982 - val_loss: 0.0935 - val_acc: 0.9818
Epoch 20/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.0
059 - acc: 0.9981 - val_loss: 0.0883 - val_acc: 0.9815
```

In [66]:

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

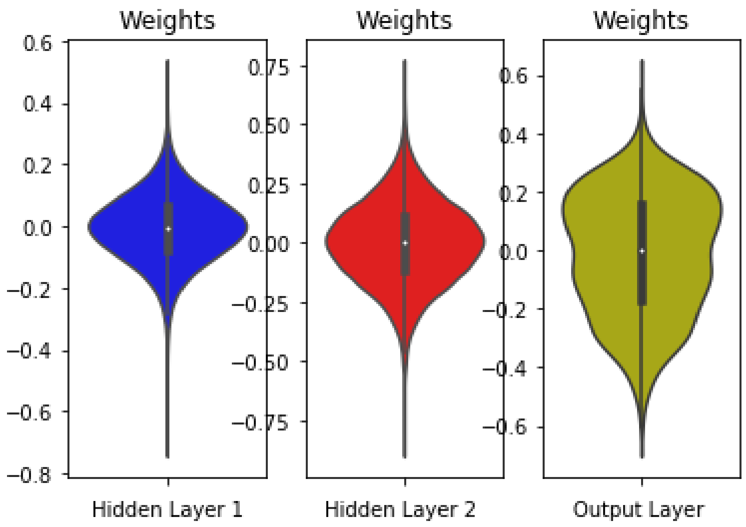
```
print ('Test accuacy:', score[1])

accuracy_1a= 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))
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.08830814581377532
Test accuacy: 0.9815
```



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

h1_w = w_after[0].flatten().reshape(-1,1)
```

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


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

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

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



1 (b) MLP + ReLU + ADAM + Batch Normalization with 2 hidden layers

In [68]:

```
from keras.layers.normalization import BatchNormalization

model_relu = Sequential()

# for relu layers
# If we sample weights from a normal distribution N(0,σ) we satisfy thi
s condition with σ=√(2/(ni).
# h1 =>  σ=√(2/(fan_in) = 0.073  => N(0,σ) = N(0,0.073)
# h2 =>  σ=√(2/(fan_in) = 0.133  => N(0,σ) = N(0,0.133)

model_relu.add(Dense(368, activation='relu', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.073, seed=None)))
model_relu.add(BatchNormalization())

model_relu.add(Dense(112, activation='relu',kernel_initializer=RandomNo
rmal(mean =0.0, stddev=0.133, seed=None)))
model_relu.add(BatchNormalization())

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

print(model_relu.summary())

model_relu.compile(optimizer='adam', loss='categorical_crossentropy', m
etrics=['accuracy'])
history = model_relu.fit(X_train, Y_train, batch_size = batch_size, epo
chs = nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_56 (Dense)             (None, 368)               288880    
_________________________________________________________________
batch_normalization_30 (Batc (None, 368)               1472      
_________________________________________________________________
dense_57 (Dense)             (None, 112)               41328     
_________________________________________________________________
```

```
batch_normalization_31 (Batc (None, 112)              448
_________________________________________________________________
dense_58 (Dense)             (None, 10)               1130
=================================================================
Total params: 333,258
Trainable params: 332,298
Non-trainable params: 960
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.2
313 - acc: 0.9323 - val_loss: 0.1113 - val_acc: 0.9662
Epoch 2/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0
824 - acc: 0.9761 - val_loss: 0.0892 - val_acc: 0.9719
Epoch 3/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.0
517 - acc: 0.9841 - val_loss: 0.0762 - val_acc: 0.9753
Epoch 4/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.0
344 - acc: 0.9895 - val_loss: 0.0785 - val_acc: 0.9750
Epoch 5/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.0
245 - acc: 0.9926 - val_loss: 0.0719 - val_acc: 0.9776
Epoch 6/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.0
207 - acc: 0.9935 - val_loss: 0.0723 - val_acc: 0.9775
Epoch 7/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0
175 - acc: 0.9946 - val_loss: 0.0793 - val_acc: 0.9768
Epoch 8/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.0
162 - acc: 0.9950 - val_loss: 0.0791 - val_acc: 0.9777
Epoch 9/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0
122 - acc: 0.9964 - val_loss: 0.0794 - val_acc: 0.9780
Epoch 10/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0
```

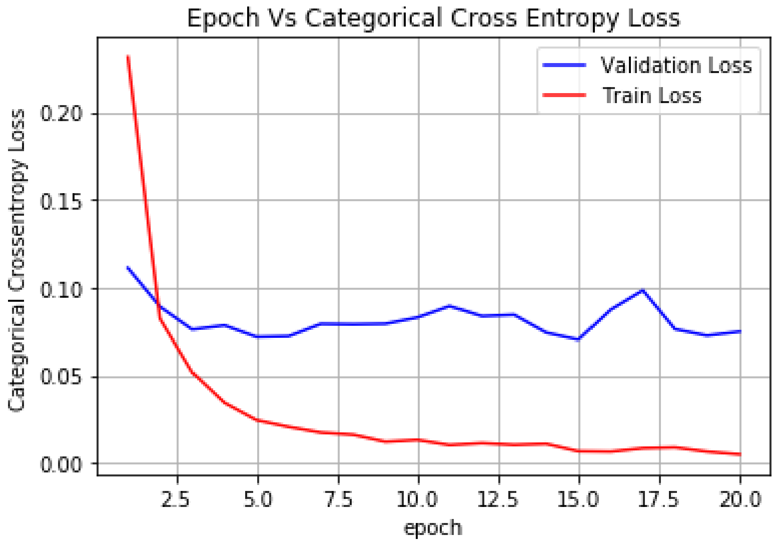
```
132 - acc: 0.9957 - val_loss: 0.0830 - val_acc: 0.9764
Epoch 11/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0
103 - acc: 0.9969 - val_loss: 0.0895 - val_acc: 0.9766
Epoch 12/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0
114 - acc: 0.9961 - val_loss: 0.0838 - val_acc: 0.9775
Epoch 13/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0
104 - acc: 0.9967 - val_loss: 0.0846 - val_acc: 0.9769
Epoch 14/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0
110 - acc: 0.9962 - val_loss: 0.0744 - val_acc: 0.9811
Epoch 15/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0
068 - acc: 0.9979 - val_loss: 0.0704 - val_acc: 0.9811
Epoch 16/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0
066 - acc: 0.9979 - val_loss: 0.0875 - val_acc: 0.9782
Epoch 17/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0
085 - acc: 0.9972 - val_loss: 0.0985 - val_acc: 0.9759
Epoch 18/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0
089 - acc: 0.9969 - val_loss: 0.0764 - val_acc: 0.9807
Epoch 19/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0
066 - acc: 0.9977 - val_loss: 0.0727 - val_acc: 0.9823
Epoch 20/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0
050 - acc: 0.9986 - val_loss: 0.0749 - val_acc: 0.9834
```

In [69]:

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

accuracy_1b= score[1]

fig, ax = plt.subplots(1,1)
```

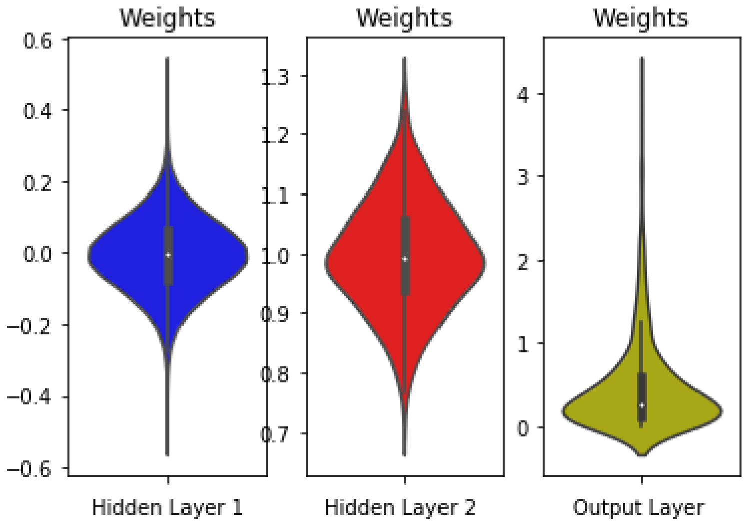
```
ax.set_xlabel('epoch')
ax.set_ylabel('Categorical Crossentropy Loss')

#List of epoch numbers
x = list(range(1, nb_epoch+1))
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.07494691710416282
Test accuacy: 0.9834
```



In [70]:
```
w_after = model_relu.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
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("Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

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

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



1 (c) MLP + ReLU + ADAM + Batch Normalization + Dropout(0.4) with 2 hidden layers

In [71]:
```
from keras.layers import Dropout
```

```
model_relu = Sequential()

# for relu layers
# If we sample weights from a normal distribution N(0,σ) we satisfy thi
s condition with σ=√(2/(ni).
# h1 =>  σ=√(2/(fan_in) = 0.073  => N(0,σ) = N(0,0.073)
# h2 =>  σ=√(2/(fan_in) = 0.133  => N(0,σ) = N(0,0.133)

model_relu.add(Dense(368, activation='relu', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.073, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.4))

model_relu.add(Dense(112, activation='relu',kernel_initializer=RandomNo
rmal(mean =0.0, stddev=0.133, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.4))

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

print(model_relu.summary())

model_relu.compile(optimizer='adam', loss='categorical_crossentropy', m
etrics=['accuracy'])
history = model_relu.fit(X_train, Y_train, batch_size = batch_size, epo
chs = nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_59 (Dense)             (None, 368)               288880    
_________________________________________________________________
batch_normalization_32 (Batc (None, 368)               1472      
_________________________________________________________________
dropout_20 (Dropout)         (None, 368)               0         
_________________________________________________________________
dense_60 (Dense)             (None, 112)               41328     
_________________________________________________________________
batch_normalization_33 (Batc (None, 112)               448       
```

```
_________________________________________________________________
dropout_21 (Dropout)         (None, 112)               0
_________________________________________________________________
dense_61 (Dense)             (None, 10)                1130
=================================================================
Total params: 333,258
Trainable params: 332,298
Non-trainable params: 960
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.4
490 - acc: 0.8626 - val_loss: 0.1546 - val_acc: 0.9530
Epoch 2/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.2
083 - acc: 0.9376 - val_loss: 0.1117 - val_acc: 0.9642
Epoch 3/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.1
589 - acc: 0.9523 - val_loss: 0.0880 - val_acc: 0.9723
Epoch 4/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.1
358 - acc: 0.9577 - val_loss: 0.0810 - val_acc: 0.9744
Epoch 5/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.1
159 - acc: 0.9641 - val_loss: 0.0771 - val_acc: 0.9752
Epoch 6/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.1
046 - acc: 0.9676 - val_loss: 0.0730 - val_acc: 0.9771
Epoch 7/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0
933 - acc: 0.9715 - val_loss: 0.0680 - val_acc: 0.9796
Epoch 8/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0
842 - acc: 0.9734 - val_loss: 0.0681 - val_acc: 0.9789
Epoch 9/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0
811 - acc: 0.9747 - val_loss: 0.0616 - val_acc: 0.9814
Epoch 10/20
```

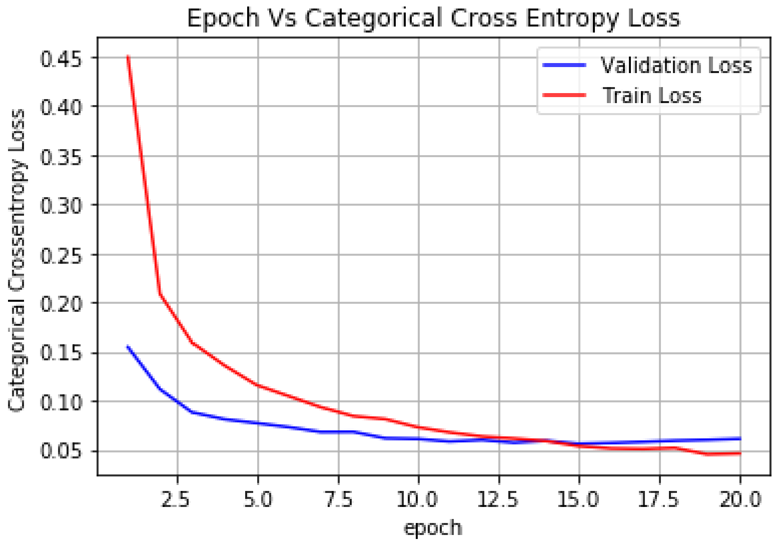
```
60000/60000 [==============================] - 3s 53us/step - loss: 0.0
729 - acc: 0.9765 - val_loss: 0.0610 - val_acc: 0.9815
Epoch 11/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0
676 - acc: 0.9780 - val_loss: 0.0583 - val_acc: 0.9824
Epoch 12/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0
634 - acc: 0.9802 - val_loss: 0.0599 - val_acc: 0.9818
Epoch 13/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0
613 - acc: 0.9799 - val_loss: 0.0571 - val_acc: 0.9825
Epoch 14/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0
587 - acc: 0.9816 - val_loss: 0.0591 - val_acc: 0.9817
Epoch 15/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0
536 - acc: 0.9825 - val_loss: 0.0560 - val_acc: 0.9825
Epoch 16/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0
512 - acc: 0.9835 - val_loss: 0.0568 - val_acc: 0.9829
Epoch 17/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0
507 - acc: 0.9835 - val_loss: 0.0579 - val_acc: 0.9823
Epoch 18/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0
517 - acc: 0.9832 - val_loss: 0.0592 - val_acc: 0.9826
Epoch 19/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0
453 - acc: 0.9852 - val_loss: 0.0599 - val_acc: 0.9832
Epoch 20/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0
460 - acc: 0.9853 - val_loss: 0.0610 - val_acc: 0.9827
```

In [72]:

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

accuracy_1c= score[1]
```

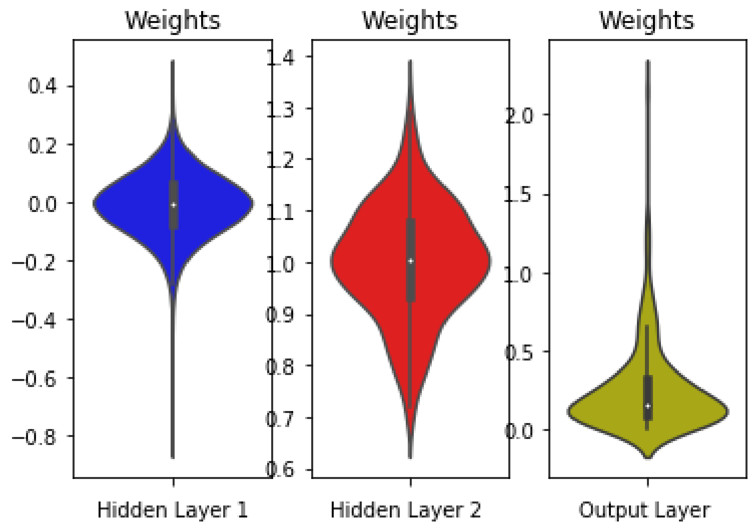
```
fig, ax = plt.subplots(1,1)
ax.set_xlabel('epoch')
ax.set_ylabel('Categorical Crossentropy Loss')

#List of epoch numbers
x = list(range(1, nb_epoch+1))
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.061045588629494885
Test accuacy: 0.9827
```



In [73]:
```
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("Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

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

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



1 (d) MLP + ReLU + ADAM + Batch Normalization + Dropout(0.5) with 2 hidden layers

In [74]:
```
from keras.layers import Dropout

model_relu = Sequential()

# for relu layers
# If we sample weights from a normal distribution N(0,σ) we satisfy thi
s condition with σ=√(2/(ni).
# h1 =>  σ=√(2/(fan_in) = 0.073  => N(0,σ) = N(0,0.073)
# h2 =>  σ=√(2/(fan_in) = 0.133  => N(0,σ) = N(0,0.133)

model_relu.add(Dense(368, activation='relu', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.073, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))

model_relu.add(Dense(112, activation='relu',kernel_initializer=RandomNo
rmal(mean =0.0, stddev=0.133, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))

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

print(model_relu.summary())

model_relu.compile(optimizer='adam', loss='categorical_crossentropy', m
etrics=['accuracy'])
history = model_relu.fit(X_train, Y_train, batch_size = batch_size, epo
chs = nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_62 (Dense)             (None, 368)               288880    
_________________________________________________________________
batch_normalization_34 (Batc (None, 368)               1472      
_________________________________________________________________
dropout_22 (Dropout)         (None, 368)               0         
_________________________________________________________________
dense_63 (Dense)             (None, 112)               41328     
_________________________________________________________________
```

```
batch_normalization_35 (Batc (None, 112)              448
_________________________________________________________________
dropout_23 (Dropout)         (None, 112)              0
_________________________________________________________________
dense_64 (Dense)             (None, 10)               1130
=================================================================
Total params: 333,258
Trainable params: 332,298
Non-trainable params: 960
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 6s 101us/step - loss: 0.
5493 - acc: 0.8347 - val_loss: 0.1784 - val_acc: 0.9437
Epoch 2/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.2
607 - acc: 0.9229 - val_loss: 0.1306 - val_acc: 0.9593
Epoch 3/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.2
022 - acc: 0.9396 - val_loss: 0.1035 - val_acc: 0.9676
Epoch 4/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.1
690 - acc: 0.9494 - val_loss: 0.0926 - val_acc: 0.9707
Epoch 5/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.1
471 - acc: 0.9562 - val_loss: 0.0864 - val_acc: 0.9722
Epoch 6/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.1
338 - acc: 0.9598 - val_loss: 0.0793 - val_acc: 0.9753
Epoch 7/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.1
219 - acc: 0.9634 - val_loss: 0.0767 - val_acc: 0.9763
Epoch 8/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.1
128 - acc: 0.9658 - val_loss: 0.0690 - val_acc: 0.9785
Epoch 9/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.1
105 - acc: 0.9663 - val_loss: 0.0746 - val_acc: 0.9779
```

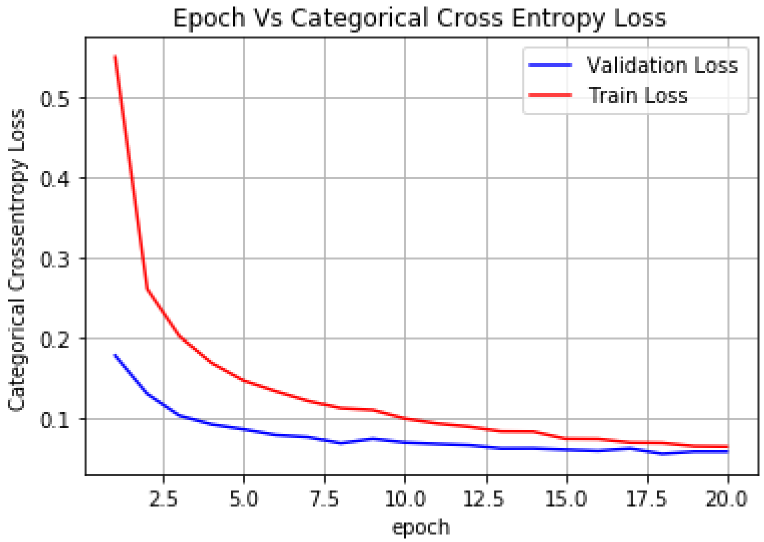
```
Epoch 10/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0
998 - acc: 0.9686 - val_loss: 0.0699 - val_acc: 0.9782
Epoch 11/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0
937 - acc: 0.9707 - val_loss: 0.0681 - val_acc: 0.9794
Epoch 12/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0
897 - acc: 0.9726 - val_loss: 0.0666 - val_acc: 0.9795
Epoch 13/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0
838 - acc: 0.9749 - val_loss: 0.0625 - val_acc: 0.9806
Epoch 14/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0
836 - acc: 0.9744 - val_loss: 0.0626 - val_acc: 0.9830
Epoch 15/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0
746 - acc: 0.9762 - val_loss: 0.0609 - val_acc: 0.9809
Epoch 16/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0
743 - acc: 0.9766 - val_loss: 0.0595 - val_acc: 0.9818
Epoch 17/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0
698 - acc: 0.9780 - val_loss: 0.0625 - val_acc: 0.9822
Epoch 18/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0
693 - acc: 0.9789 - val_loss: 0.0558 - val_acc: 0.9844
Epoch 19/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.0
653 - acc: 0.9790 - val_loss: 0.0586 - val_acc: 0.9834
Epoch 20/20
60000/60000 [==============================] - 3s 56us/step - loss: 0.0
648 - acc: 0.9797 - val_loss: 0.0586 - val_acc: 0.9834
```

In [75]:

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

accuracy_1d= score[1]
```

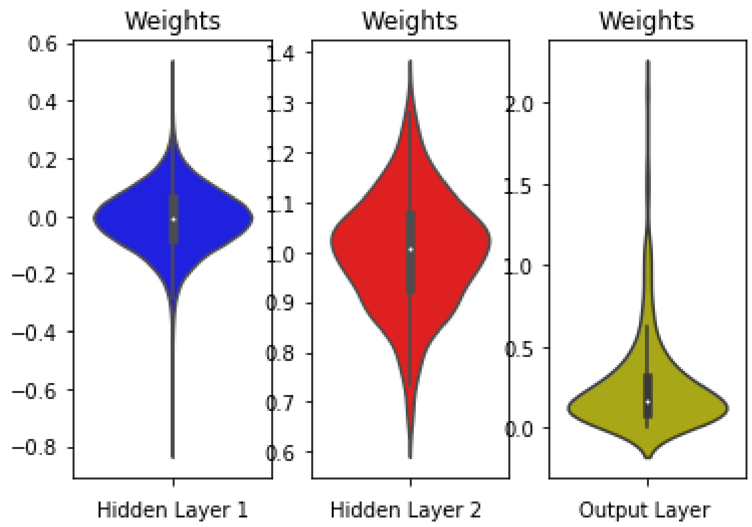
```
fig, ax = plt.subplots(1,1)
ax.set_xlabel('epoch')
ax.set_ylabel('Categorical Crossentropy Loss')

#List of epoch numbers
x = list(range(1, nb_epoch+1))
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.0585743734298856
Test accuacy: 0.9834
```



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

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
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("Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

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

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



1 (e) MLP + ReLU + ADAM + Batch Normalization + Dropout(0.6) with 2 hidden layers

In [77]:

```
from keras.layers import Dropout

model_relu = Sequential()

# for relu layers
# If we sample weights from a normal distribution N(0,σ) we satisfy thi
s condition with σ=√(2/(ni).
# h1 =>  σ=√(2/(fan_in) = 0.073  => N(0,σ) = N(0,0.073)
# h2 =>  σ=√(2/(fan_in) = 0.133  => N(0,σ) = N(0,0.133)

model_relu.add(Dense(368, activation='relu', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.073, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.6))

model_relu.add(Dense(112, activation='relu',kernel_initializer=RandomNo
rmal(mean =0.0, stddev=0.133, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.6))

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

print(model_relu.summary())

model_relu.compile(optimizer='adam', loss='categorical_crossentropy', m
etrics=['accuracy'])
history = model_relu.fit(X_train, Y_train, batch_size = batch_size, epo
chs = nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_65 (Dense)             (None, 368)               288880    
_________________________________________________________________
batch_normalization_36 (Batc (None, 368)               1472      
_________________________________________________________________
dropout_24 (Dropout)         (None, 368)               0         
_________________________________________________________________
dense_66 (Dense)             (None, 112)               41328     
```

```
_________________________________________________________________
batch_normalization_37 (Batc (None, 112)               448
_________________________________________________________________
dropout_25 (Dropout)         (None, 112)               0
_________________________________________________________________
dense_67 (Dense)             (None, 10)                1130
=================================================================
Total params: 333,258
Trainable params: 332,298
Non-trainable params: 960
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 6s 106us/step - loss: 0.
7216 - acc: 0.7808 - val_loss: 0.2077 - val_acc: 0.9364
Epoch 2/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.3
329 - acc: 0.9003 - val_loss: 0.1592 - val_acc: 0.9515
Epoch 3/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.2
606 - acc: 0.9235 - val_loss: 0.1266 - val_acc: 0.9605
Epoch 4/20
60000/60000 [==============================] - 3s 56us/step - loss: 0.2
161 - acc: 0.9358 - val_loss: 0.1145 - val_acc: 0.9644
Epoch 5/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.1
976 - acc: 0.9424 - val_loss: 0.1005 - val_acc: 0.9698
Epoch 6/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.1
759 - acc: 0.9487 - val_loss: 0.0933 - val_acc: 0.9716
Epoch 7/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.1
645 - acc: 0.9523 - val_loss: 0.0892 - val_acc: 0.9733
Epoch 8/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.1
506 - acc: 0.9564 - val_loss: 0.0877 - val_acc: 0.9742
Epoch 9/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.1
```

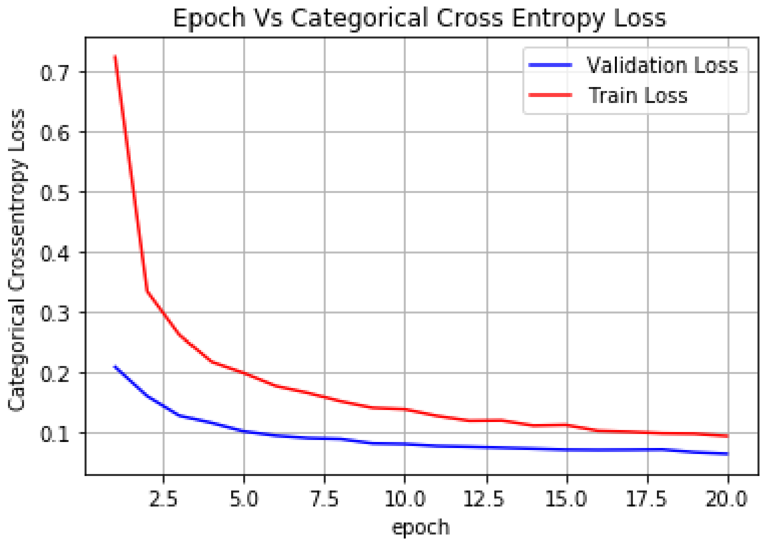
```
396 - acc: 0.9588 - val_loss: 0.0804 - val_acc: 0.9762
Epoch 10/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.1
373 - acc: 0.9600 - val_loss: 0.0793 - val_acc: 0.9756
Epoch 11/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.1
262 - acc: 0.9622 - val_loss: 0.0760 - val_acc: 0.9776
Epoch 12/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.1
181 - acc: 0.9655 - val_loss: 0.0748 - val_acc: 0.9782
Epoch 13/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.1
187 - acc: 0.9652 - val_loss: 0.0731 - val_acc: 0.9783
Epoch 14/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.1
100 - acc: 0.9676 - val_loss: 0.0716 - val_acc: 0.9784
Epoch 15/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.1
112 - acc: 0.9665 - val_loss: 0.0698 - val_acc: 0.9786
Epoch 16/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.1
014 - acc: 0.9699 - val_loss: 0.0694 - val_acc: 0.9795
Epoch 17/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.0
997 - acc: 0.9696 - val_loss: 0.0697 - val_acc: 0.9793
Epoch 18/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0
972 - acc: 0.9703 - val_loss: 0.0702 - val_acc: 0.9791
Epoch 19/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0
965 - acc: 0.9713 - val_loss: 0.0656 - val_acc: 0.9802
Epoch 20/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0
927 - acc: 0.9719 - val_loss: 0.0630 - val_acc: 0.9811
```

In [78]:

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

```
accuracy_1e= 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))
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

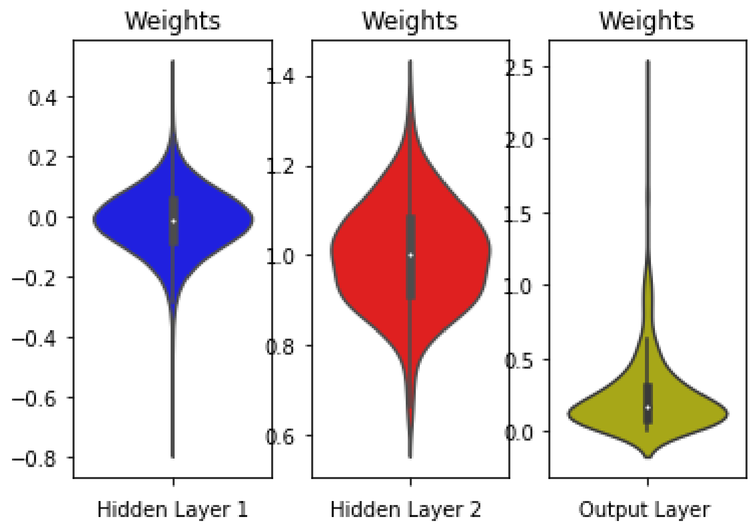
```
Test score: 0.0630017356009921
Test accuacy: 0.9811
```



In [79]:

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

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

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



2 (a) MLP + ReLU + ADAM with 3 hidden layers

In [80]:

```
model_relu = Sequential()

# for relu layers
# If we sample weights from a normal distribution N(0,σ) we satisfy thi
s condition with σ=√(2/(ni).
# h1 =>  σ=√(2/(fan_in) = 0.062  => N(0,σ) = N(0,0.073)
# h2 =>  σ=√(2/(fan_in) = 0.125  => N(0,σ) = N(0,0.133)

model_relu.add(Dense(368, activation='relu', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.073, seed=None)))
model_relu.add(Dense(112, activation='relu',kernel_initializer=RandomNo
rmal(mean =0.0, stddev=0.133, seed=None)))
model_relu.add(Dense(54, activation='relu',kernel_initializer=RandomNor
mal(mean =0.0, stddev=0.192, seed=None)))
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())
print('\n')
model_relu.compile(optimizer='adam', loss='categorical_crossentropy', m
etrics=['accuracy'])
history = model_relu.fit(X_train, Y_train, batch_size = batch_size, epo
chs = nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_68 (Dense)             (None, 368)               288880
_________________________________________________________________
dense_69 (Dense)             (None, 112)               41328
_________________________________________________________________
dense_70 (Dense)             (None, 54)                6102
_________________________________________________________________
dense_71 (Dense)             (None, 10)                550
=================================================================
Total params: 336,860
Trainable params: 336,860
Non-trainable params: 0
_________________________________________________________________
None
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.2
749 - acc: 0.9166 - val_loss: 0.1294 - val_acc: 0.9611
Epoch 2/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0
950 - acc: 0.9714 - val_loss: 0.0870 - val_acc: 0.9730
Epoch 3/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0
609 - acc: 0.9812 - val_loss: 0.0900 - val_acc: 0.9723
Epoch 4/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0
438 - acc: 0.9860 - val_loss: 0.0870 - val_acc: 0.9714
Epoch 5/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.0
333 - acc: 0.9888 - val_loss: 0.0900 - val_acc: 0.9724
Epoch 6/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0
259 - acc: 0.9920 - val_loss: 0.0782 - val_acc: 0.9774
Epoch 7/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0
225 - acc: 0.9927 - val_loss: 0.0987 - val_acc: 0.9742
Epoch 8/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.0
204 - acc: 0.9933 - val_loss: 0.0864 - val_acc: 0.9771
Epoch 9/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0
185 - acc: 0.9937 - val_loss: 0.0984 - val_acc: 0.9744
Epoch 10/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0
146 - acc: 0.9952 - val_loss: 0.0883 - val_acc: 0.9782
Epoch 11/20
60000/60000 [==============================] - 2s 37us/step - loss: 0.0
140 - acc: 0.9955 - val_loss: 0.0915 - val_acc: 0.9778
Epoch 12/20
60000/60000 [==============================] - 2s 37us/step - loss: 0.0
154 - acc: 0.9948 - val_loss: 0.0852 - val_acc: 0.9805
```

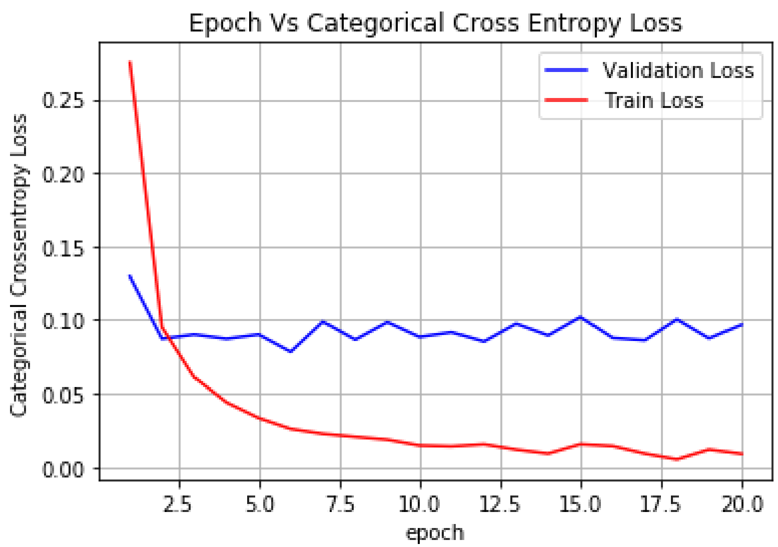
```
Epoch 13/20
60000/60000 [==============================] - 2s 37us/step - loss: 0.0
117 - acc: 0.9960 - val_loss: 0.0974 - val_acc: 0.9799
Epoch 14/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0
090 - acc: 0.9969 - val_loss: 0.0894 - val_acc: 0.9804
Epoch 15/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0
154 - acc: 0.9953 - val_loss: 0.1019 - val_acc: 0.9776
Epoch 16/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0
142 - acc: 0.9955 - val_loss: 0.0876 - val_acc: 0.9811
Epoch 17/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.0
090 - acc: 0.9972 - val_loss: 0.0860 - val_acc: 0.9807
Epoch 18/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.0
051 - acc: 0.9984 - val_loss: 0.1003 - val_acc: 0.9794
Epoch 19/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0
118 - acc: 0.9962 - val_loss: 0.0873 - val_acc: 0.9801
Epoch 20/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0
089 - acc: 0.9971 - val_loss: 0.0966 - val_acc: 0.9791
```

In [81]:

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

accuracy_2a= 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))
vy = history.history['val_loss']
```

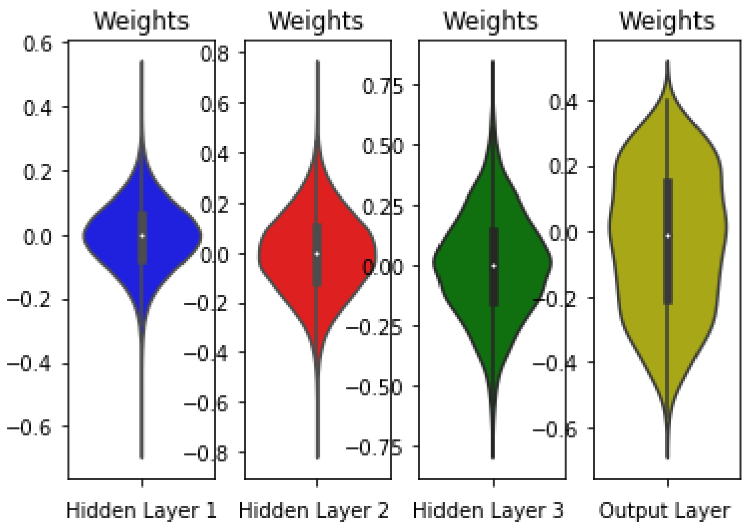
```
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.09661503153744422
Test accuacy: 0.9791
```



In [82]:
```
w_after = model_relu.get_weights()

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


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 4, 1)
plt.title("Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')
```

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

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

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



2 (b) MLP + ReLU + ADAM + Batch Normalization with 3 hidden layers

In [83]:
```
from keras.layers.normalization import BatchNormalization
```

```
model_relu = Sequential()

# for relu layers
# If we sample weights from a normal distribution N(0,σ) we satisfy thi
s condition with σ=√(2/(ni).
# h1 =>  σ=√(2/(fan_in) = 0.073  => N(0,σ) = N(0,0.073)
# h2 =>  σ=√(2/(fan_in) = 0.133  => N(0,σ) = N(0,0.133)

model_relu.add(Dense(368, activation='relu', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.073, seed=None)))
model_relu.add(BatchNormalization())

model_relu.add(Dense(112, activation='relu',kernel_initializer=RandomNo
rmal(mean =0.0, stddev=0.133, seed=None)))
model_relu.add(BatchNormalization())

model_relu.add(Dense(54, activation='relu',kernel_initializer=RandomNor
mal(mean =0.0, stddev=0.192, seed=None)))
model_relu.add(BatchNormalization())

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

print(model_relu.summary())
print('\n')

model_relu.compile(optimizer='adam', loss='categorical_crossentropy', m
etrics=['accuracy'])
history = model_relu.fit(X_train, Y_train, batch_size = batch_size, epo
chs = nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_72 (Dense)             (None, 368)               288880
_________________________________________________________________
batch_normalization_38 (Batc (None, 368)               1472
_________________________________________________________________
dense_73 (Dense)             (None, 112)               41328
_________________________________________________________________
```

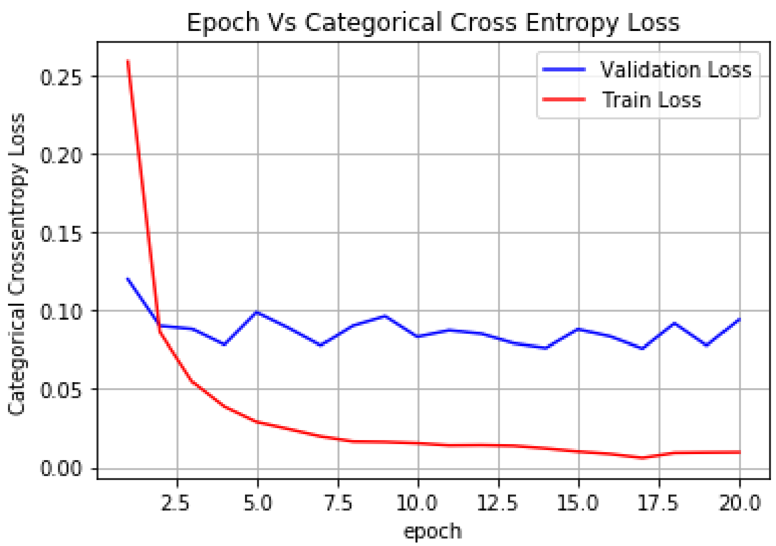
```
batch_normalization_39 (Batc (None, 112)              448
_________________________________________________________________
dense_74 (Dense)             (None, 54)               6102
_________________________________________________________________
batch_normalization_40 (Batc (None, 54)               216
_________________________________________________________________
dense_75 (Dense)             (None, 10)               550
=================================================================
Total params: 338,996
Trainable params: 337,928
Non-trainable params: 1,068
_________________________________________________________________
None


Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.
2583 - acc: 0.9254 - val_loss: 0.1199 - val_acc: 0.9635
Epoch 2/20
60000/60000 [==============================] - 4s 62us/step - loss: 0.0
862 - acc: 0.9742 - val_loss: 0.0901 - val_acc: 0.9711
Epoch 3/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.0
546 - acc: 0.9839 - val_loss: 0.0881 - val_acc: 0.9717
Epoch 4/20
60000/60000 [==============================] - 4s 62us/step - loss: 0.0
387 - acc: 0.9878 - val_loss: 0.0782 - val_acc: 0.9754
Epoch 5/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.0
290 - acc: 0.9911 - val_loss: 0.0988 - val_acc: 0.9701
Epoch 6/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.0
244 - acc: 0.9921 - val_loss: 0.0887 - val_acc: 0.9748
Epoch 7/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.0
197 - acc: 0.9936 - val_loss: 0.0777 - val_acc: 0.9782
Epoch 8/20
60000/60000 [==============================] - 4s 62us/step - loss: 0.0
```

```
164 - acc: 0.9949 - val_loss: 0.0902 - val_acc: 0.9753
Epoch 9/20
60000/60000 [==============================] - 4s 62us/step - loss: 0.0
161 - acc: 0.9944 - val_loss: 0.0962 - val_acc: 0.9740
Epoch 10/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.0
153 - acc: 0.9950 - val_loss: 0.0831 - val_acc: 0.9787
Epoch 11/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0
140 - acc: 0.9954 - val_loss: 0.0872 - val_acc: 0.9767
Epoch 12/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.0
141 - acc: 0.9950 - val_loss: 0.0850 - val_acc: 0.9774
Epoch 13/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.0
137 - acc: 0.9953 - val_loss: 0.0790 - val_acc: 0.9803
Epoch 14/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.0
120 - acc: 0.9959 - val_loss: 0.0760 - val_acc: 0.9802
Epoch 15/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.0
100 - acc: 0.9967 - val_loss: 0.0879 - val_acc: 0.9787
Epoch 16/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.0
084 - acc: 0.9973 - val_loss: 0.0833 - val_acc: 0.9798
Epoch 17/20
60000/60000 [==============================] - 4s 62us/step - loss: 0.0
060 - acc: 0.9982 - val_loss: 0.0757 - val_acc: 0.9805
Epoch 18/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0
092 - acc: 0.9969 - val_loss: 0.0917 - val_acc: 0.9777
Epoch 19/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.0
094 - acc: 0.9965 - val_loss: 0.0776 - val_acc: 0.9810
Epoch 20/20
60000/60000 [==============================] - 4s 62us/step - loss: 0.0
096 - acc: 0.9966 - val_loss: 0.0940 - val_acc: 0.9790
```

In [84]:
```
score = model_relu.evaluate(X_test, Y_test, verbose=0)
```

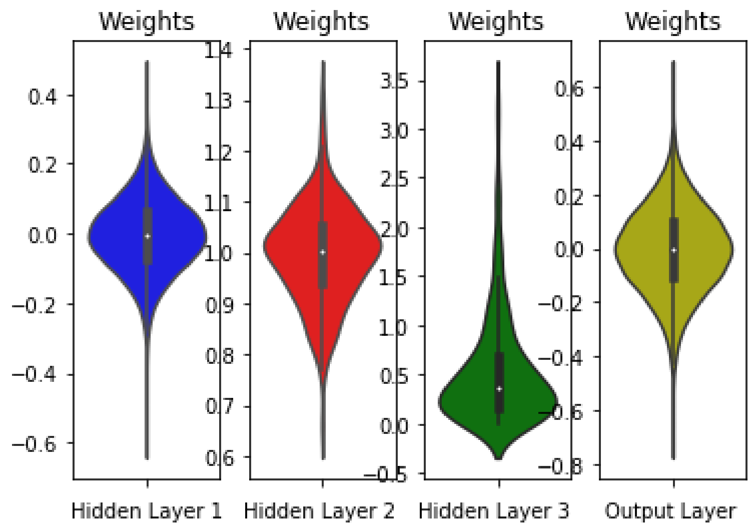
```
print ('Test score:', score[0])
print ('Test accuacy:', score[1])

accuracy_2b= 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))
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.09398488705815872
Test accuacy: 0.979
```



In [85]:
```
w_after = model_relu.get_weights()
```

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


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 4, 1)
plt.title("Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

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

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

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



2 (c) MLP + ReLU + ADAM + Batch Normalization + Dropout (0.4) with 3 hidden layers

In [86]:

```
from keras.layers.normalization import BatchNormalization

model_relu = Sequential()

# for relu layers
# If we sample weights from a normal distribution N(0,σ) we satisfy thi
s condition with σ=√(2/(ni).
# h1 =>  σ=√(2/(fan_in) = 0.073  => N(0,σ) = N(0,0.073)
# h2 =>  σ=√(2/(fan_in) = 0.133  => N(0,σ) = N(0,0.133)

model_relu.add(Dense(368, activation='relu', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.073, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.4))

model_relu.add(Dense(112, activation='relu',kernel_initializer=RandomNo
rmal(mean =0.0, stddev=0.133, seed=None)))
```

```
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.4))

model_relu.add(Dense(54, activation='relu',kernel_initializer=RandomNormal(mean =0.0, stddev=0.192, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.4))

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

print(model_relu.summary())
print('\n')

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_76 (Dense)             (None, 368)               288880
_________________________________________________________________
batch_normalization_41 (Batc (None, 368)               1472
_________________________________________________________________
dropout_26 (Dropout)         (None, 368)               0
_________________________________________________________________
dense_77 (Dense)             (None, 112)               41328
_________________________________________________________________
batch_normalization_42 (Batc (None, 112)               448
_________________________________________________________________
dropout_27 (Dropout)         (None, 112)               0
_________________________________________________________________
dense_78 (Dense)             (None, 54)                6102
_________________________________________________________________
batch_normalization_43 (Batc (None, 54)                216
_________________________________________________________________
dropout_28 (Dropout)         (None, 54)                0
_________________________________________________________________
dense_79 (Dense)             (None, 10)                550
```

```
=================================================================
Total params: 338,996
Trainable params: 337,928
Non-trainable params: 1,068
_________________________________________________________________
None


Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.
6901 - acc: 0.7904 - val_loss: 0.1926 - val_acc: 0.9411
Epoch 2/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.2
870 - acc: 0.9175 - val_loss: 0.1392 - val_acc: 0.9573
Epoch 3/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.2
203 - acc: 0.9373 - val_loss: 0.1100 - val_acc: 0.9671
Epoch 4/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.1
741 - acc: 0.9496 - val_loss: 0.0996 - val_acc: 0.9696
Epoch 5/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.1
518 - acc: 0.9565 - val_loss: 0.0983 - val_acc: 0.9714
Epoch 6/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.1
383 - acc: 0.9602 - val_loss: 0.0861 - val_acc: 0.9763
Epoch 7/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.1
245 - acc: 0.9644 - val_loss: 0.0811 - val_acc: 0.9761
Epoch 8/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.1
138 - acc: 0.9667 - val_loss: 0.0797 - val_acc: 0.9776
Epoch 9/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.1
086 - acc: 0.9688 - val_loss: 0.0762 - val_acc: 0.9780
Epoch 10/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.0
984 - acc: 0.9719 - val_loss: 0.0736 - val_acc: 0.9769
```

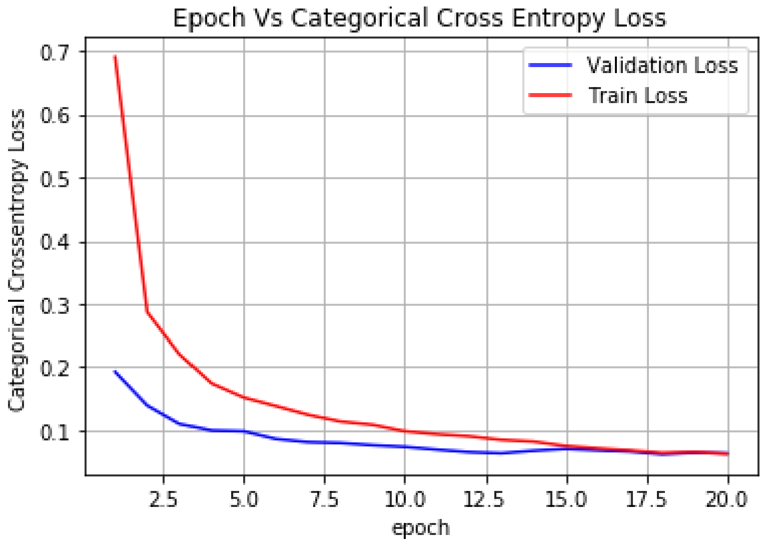
```
Epoch 11/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.0
937 - acc: 0.9729 - val_loss: 0.0692 - val_acc: 0.9786
Epoch 12/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0
904 - acc: 0.9738 - val_loss: 0.0652 - val_acc: 0.9806
Epoch 13/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0
847 - acc: 0.9752 - val_loss: 0.0634 - val_acc: 0.9814
Epoch 14/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0
820 - acc: 0.9760 - val_loss: 0.0675 - val_acc: 0.9805
Epoch 15/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0
751 - acc: 0.9776 - val_loss: 0.0703 - val_acc: 0.9789
Epoch 16/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0
712 - acc: 0.9786 - val_loss: 0.0681 - val_acc: 0.9811
Epoch 17/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.0
679 - acc: 0.9801 - val_loss: 0.0661 - val_acc: 0.9809
Epoch 18/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0
640 - acc: 0.9810 - val_loss: 0.0618 - val_acc: 0.9825
Epoch 19/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0
653 - acc: 0.9805 - val_loss: 0.0644 - val_acc: 0.9816
Epoch 20/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0
624 - acc: 0.9816 - val_loss: 0.0634 - val_acc: 0.9821
```

In [87]:

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

accuracy_2c= score[1]

fig, ax = plt.subplots(1,1)
```

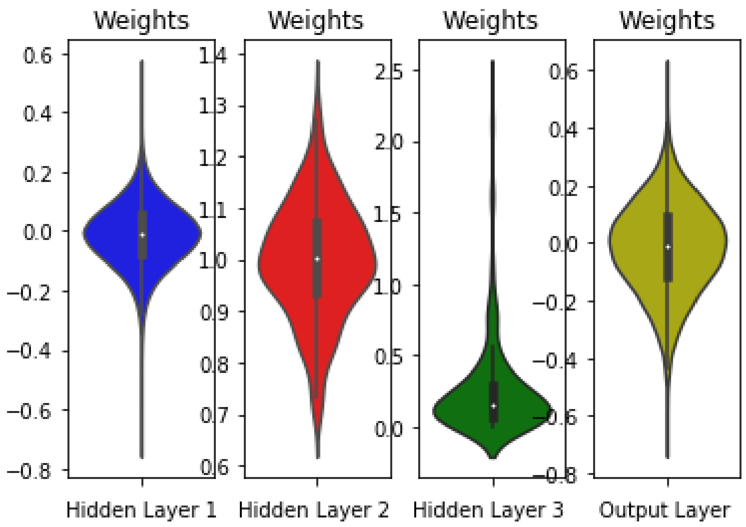
```
ax.set_xlabel('epoch')
ax.set_ylabel('Categorical Crossentropy Loss')

#List of epoch numbers
x = list(range(1, nb_epoch+1))
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.06336671170063783
Test accuacy: 0.9821
```



In [88]:
```
w_after = model_relu.get_weights()

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

```
fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 4, 1)
plt.title("Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

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

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

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



2 (d) MLP + ReLU + ADAM + Batch Normalization + Dropout (0.5) with 3 hidden layers

In [89]:
```
from keras.layers.normalization import BatchNormalization

model_relu = Sequential()

# for relu layers
# If we sample weights from a normal distribution N(0,σ) we satisfy thi
s condition with σ=√(2/(ni).
# h1 =>  σ=√(2/(fan_in) = 0.073  => N(0,σ) = N(0,0.073)
# h2 =>  σ=√(2/(fan_in) = 0.133  => N(0,σ) = N(0,0.133)

model_relu.add(Dense(368, activation='relu', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.073, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))

model_relu.add(Dense(112, activation='relu',kernel_initializer=RandomNo
rmal(mean =0.0, stddev=0.133, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))

model_relu.add(Dense(54, activation='relu',kernel_initializer=RandomNor
mal(mean =0.0, stddev=0.192, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))

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

print(model_relu.summary())
print('\n')

model_relu.compile(optimizer='adam', loss='categorical_crossentropy', m
etrics=['accuracy'])
history = model_relu.fit(X_train, Y_train, batch_size = batch_size, epo
chs = nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_80 (Dense)             (None, 368)               288880
_________________________________________________________________
batch_normalization_44 (Batc (None, 368)               1472
_________________________________________________________________
dropout_29 (Dropout)         (None, 368)               0
_________________________________________________________________
dense_81 (Dense)             (None, 112)               41328
_________________________________________________________________
batch_normalization_45 (Batc (None, 112)               448
_________________________________________________________________
dropout_30 (Dropout)         (None, 112)               0
_________________________________________________________________
dense_82 (Dense)             (None, 54)                6102
_________________________________________________________________
batch_normalization_46 (Batc (None, 54)                216
_________________________________________________________________
dropout_31 (Dropout)         (None, 54)                0
_________________________________________________________________
dense_83 (Dense)             (None, 10)                550
=================================================================
Total params: 338,996
Trainable params: 337,928
Non-trainable params: 1,068
_________________________________________________________________
None


Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 8s 141us/step - loss: 0.
9417 - acc: 0.7092 - val_loss: 0.2406 - val_acc: 0.9293
Epoch 2/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.3
996 - acc: 0.8859 - val_loss: 0.1663 - val_acc: 0.9507
Epoch 3/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.2
950 - acc: 0.9180 - val_loss: 0.1343 - val_acc: 0.9609
```

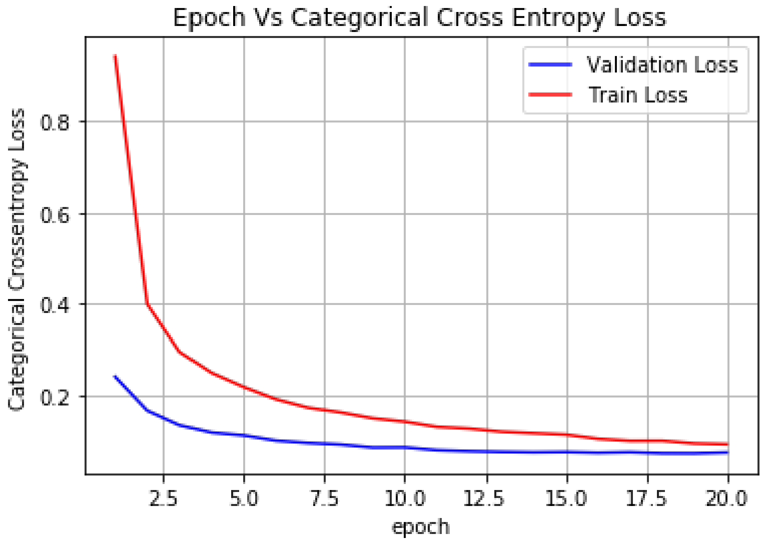
```
Epoch 4/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.2
490 - acc: 0.9312 - val_loss: 0.1180 - val_acc: 0.9656
Epoch 5/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.2
180 - acc: 0.9401 - val_loss: 0.1118 - val_acc: 0.9679
Epoch 6/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.1
913 - acc: 0.9472 - val_loss: 0.1003 - val_acc: 0.9702
Epoch 7/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.1
728 - acc: 0.9525 - val_loss: 0.0952 - val_acc: 0.9712
Epoch 8/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.1
624 - acc: 0.9561 - val_loss: 0.0917 - val_acc: 0.9756
Epoch 9/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.1
495 - acc: 0.9598 - val_loss: 0.0851 - val_acc: 0.9744
Epoch 10/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.1
421 - acc: 0.9620 - val_loss: 0.0855 - val_acc: 0.9751
Epoch 11/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.1
307 - acc: 0.9646 - val_loss: 0.0793 - val_acc: 0.9775
Epoch 12/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.1
269 - acc: 0.9655 - val_loss: 0.0770 - val_acc: 0.9783
Epoch 13/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.1
200 - acc: 0.9677 - val_loss: 0.0755 - val_acc: 0.9794
Epoch 14/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.1
167 - acc: 0.9679 - val_loss: 0.0746 - val_acc: 0.9791
Epoch 15/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.1
137 - acc: 0.9693 - val_loss: 0.0752 - val_acc: 0.9787
Epoch 16/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.1
043 - acc: 0.9703 - val_loss: 0.0736 - val_acc: 0.9810
```

```
Epoch 17/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.0
999 - acc: 0.9720 - val_loss: 0.0748 - val_acc: 0.9790
Epoch 18/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.1
000 - acc: 0.9715 - val_loss: 0.0725 - val_acc: 0.9812
Epoch 19/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.0
942 - acc: 0.9738 - val_loss: 0.0723 - val_acc: 0.9803
Epoch 20/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.0
924 - acc: 0.9747 - val_loss: 0.0742 - val_acc: 0.9804
```

In [90]:

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

accuracy_2d= 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))
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

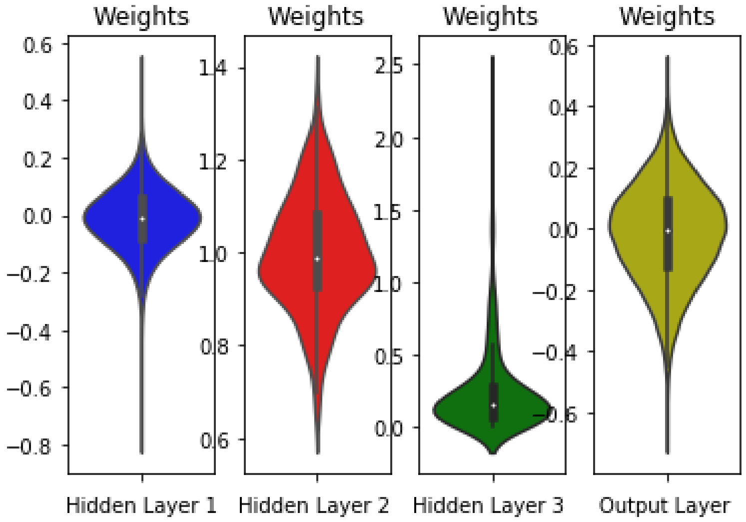
```
Test score: 0.07416196312536485
Test accuacy: 0.9804
```



In [91]:

```
w_after = model_relu.get_weights()

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


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 4, 1)
plt.title("Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

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

plt.subplot(1, 4, 3)
plt.title("Weights")
```

```
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3 ')

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



2 (e) MLP + ReLU + ADAM + Batch Normalization + Dropout (0.6) with 3 hidden layers

In [92]:
```
from keras.layers.normalization import BatchNormalization

model_relu = Sequential()

# for relu layers
# If we sample weights from a normal distribution N(0,σ) we satisfy thi
s condition with σ=√(2/(ni).
# h1 =>  σ=√(2/(fan_in) = 0.073  => N(0,σ) = N(0,0.073)
# h2 =>  σ=√(2/(fan_in) = 0.133  => N(0,σ) = N(0,0.133)
```

```
model_relu.add(Dense(368, activation='relu', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.073, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.6))

model_relu.add(Dense(112, activation='relu',kernel_initializer=RandomNo
rmal(mean =0.0, stddev=0.133, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.6))

model_relu.add(Dense(54, activation='relu',kernel_initializer=RandomNor
mal(mean =0.0, stddev=0.192, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.6))

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

print(model_relu.summary())
print('\n')

model_relu.compile(optimizer='adam', loss='categorical_crossentropy', m
etrics=['accuracy'])
history = model_relu.fit(X_train, Y_train, batch_size = batch_size, epo
chs = nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

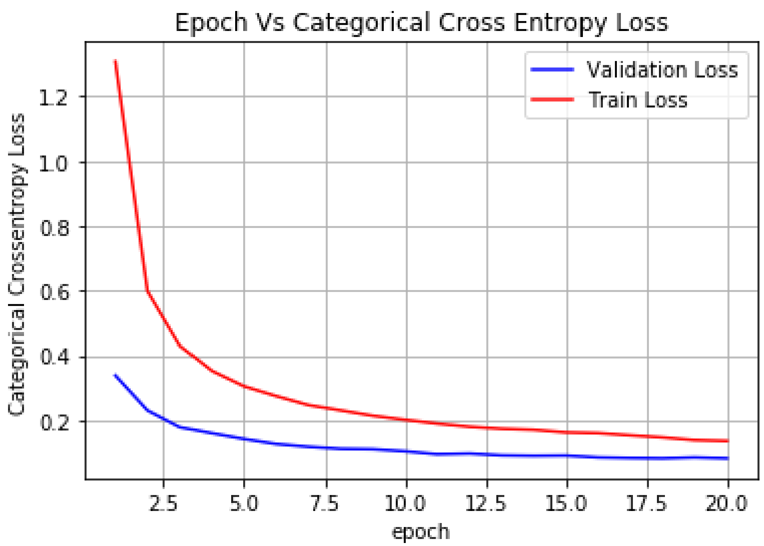
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_84 (Dense)             (None, 368)               288880    
_________________________________________________________________
batch_normalization_47 (Batc (None, 368)               1472      
_________________________________________________________________
dropout_32 (Dropout)         (None, 368)               0         
_________________________________________________________________
dense_85 (Dense)             (None, 112)               41328     
_________________________________________________________________
batch_normalization_48 (Batc (None, 112)               448       
_________________________________________________________________
dropout_33 (Dropout)         (None, 112)               0         
```

```
_________________________________________________________________
dense_86 (Dense)             (None, 54)                6102      
_________________________________________________________________
batch_normalization_49 (Batc (None, 54)                216       
_________________________________________________________________
dropout_34 (Dropout)         (None, 54)                0         
_________________________________________________________________
dense_87 (Dense)             (None, 10)                550       
=================================================================
Total params: 338,996
Trainable params: 337,928
Non-trainable params: 1,068
_________________________________________________________________
None


Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 8s 141us/step - loss: 1.
3064 - acc: 0.5828 - val_loss: 0.3388 - val_acc: 0.9123
Epoch 2/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.5
988 - acc: 0.8253 - val_loss: 0.2308 - val_acc: 0.9316
Epoch 3/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.4
305 - acc: 0.8799 - val_loss: 0.1790 - val_acc: 0.9484
Epoch 4/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.3
526 - acc: 0.9046 - val_loss: 0.1603 - val_acc: 0.9532
Epoch 5/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.3
052 - acc: 0.9188 - val_loss: 0.1424 - val_acc: 0.9589
Epoch 6/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.2
752 - acc: 0.9273 - val_loss: 0.1270 - val_acc: 0.9635
Epoch 7/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.2
475 - acc: 0.9349 - val_loss: 0.1184 - val_acc: 0.9662
Epoch 8/20
```

```
60000/60000 [==============================] - 4s 67us/step - loss: 0.2
309 - acc: 0.9393 - val_loss: 0.1120 - val_acc: 0.9685
Epoch 9/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.2
143 - acc: 0.9444 - val_loss: 0.1107 - val_acc: 0.9694
Epoch 10/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.2
018 - acc: 0.9479 - val_loss: 0.1044 - val_acc: 0.9712
Epoch 11/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.1
901 - acc: 0.9498 - val_loss: 0.0953 - val_acc: 0.9733
Epoch 12/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.1
802 - acc: 0.9529 - val_loss: 0.0974 - val_acc: 0.9733
Epoch 13/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.1
741 - acc: 0.9550 - val_loss: 0.0917 - val_acc: 0.9748
Epoch 14/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.1
709 - acc: 0.9560 - val_loss: 0.0904 - val_acc: 0.9747
Epoch 15/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.1
628 - acc: 0.9581 - val_loss: 0.0911 - val_acc: 0.9747
Epoch 16/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.1
605 - acc: 0.9586 - val_loss: 0.0853 - val_acc: 0.9771
Epoch 17/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.1
537 - acc: 0.9596 - val_loss: 0.0836 - val_acc: 0.9777
Epoch 18/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.1
474 - acc: 0.9607 - val_loss: 0.0825 - val_acc: 0.9782
Epoch 19/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.1
389 - acc: 0.9633 - val_loss: 0.0853 - val_acc: 0.9789
Epoch 20/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.1
364 - acc: 0.9646 - val_loss: 0.0825 - val_acc: 0.9786
```

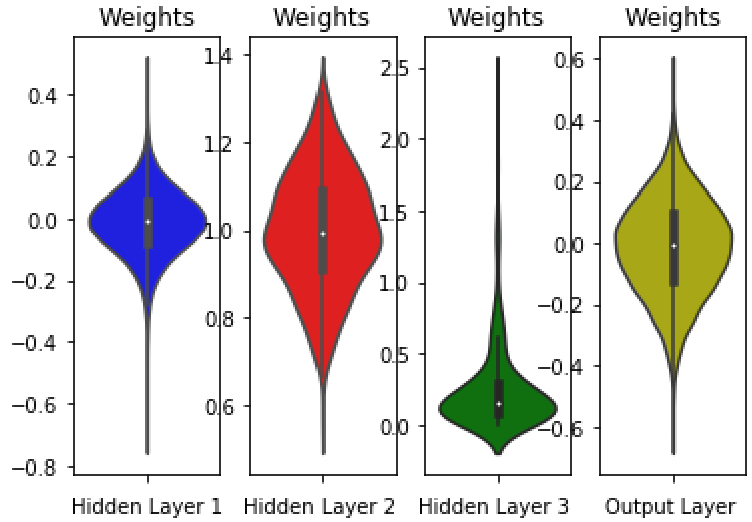
In [93]:
```
score = model_relu.evaluate(X_test, Y_test, verbose=0)
print ('Test score:', score[0])
print ('Test accuacy:', score[1])

accuracy_2e= 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))
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.08248090120132547
Test accuacy: 0.9786
```



In [94]:
```
w_after = model_relu.get_weights()
```

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


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 4, 1)
plt.title("Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

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

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

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



3 (a) MLP + ReLU + ADAM with 5 hidden layers

In [95]:
```
model_relu = Sequential()

# for relu layers
# If we sample weights from a normal distribution N(0,σ) we satisfy thi
s condition with σ=√(2/(ni).
# h1 =>  σ=√(2/(fan_in) = 0.062  => N(0,σ) = N(0,0.073)
# h2 =>  σ=√(2/(fan_in) = 0.125  => N(0,σ) = N(0,0.133)

model_relu.add(Dense(368, activation='relu', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.073, seed=None)))
model_relu.add(Dense(112, activation='relu',kernel_initializer=RandomNo
rmal(mean =0.0, stddev=0.133, seed=None)))
model_relu.add(Dense(54, activation='relu',kernel_initializer=RandomNor
mal(mean =0.0, stddev=0.192, seed=None)))
model_relu.add(Dense(28, activation='relu',kernel_initializer=RandomNor
mal(mean =0.0, stddev=0.267, seed=None)))
model_relu.add(Dense(16, activation='relu',kernel_initializer=RandomNor
mal(mean =0.0, stddev=0.353, seed=None)))
```

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

print(model_relu.summary())
print('\n')
model_relu.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = model_relu.fit(X_train, Y_train, batch_size = batch_size, epochs = nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_88 (Dense)             (None, 368)               288880    
_________________________________________________________________
dense_89 (Dense)             (None, 112)               41328     
_________________________________________________________________
dense_90 (Dense)             (None, 54)                6102      
_________________________________________________________________
dense_91 (Dense)             (None, 28)                1540      
_________________________________________________________________
dense_92 (Dense)             (None, 16)                464       
_________________________________________________________________
dense_93 (Dense)             (None, 10)                170       
=================================================================
Total params: 338,484
Trainable params: 338,484
Non-trainable params: 0
_________________________________________________________________
None


Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 110us/step - loss: 0.3656 - acc: 0.8884 - val_loss: 0.1626 - val_acc: 0.9530
Epoch 2/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.1205 - acc: 0.9650 - val_loss: 0.1107 - val_acc: 0.9671
Epoch 3/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0
```

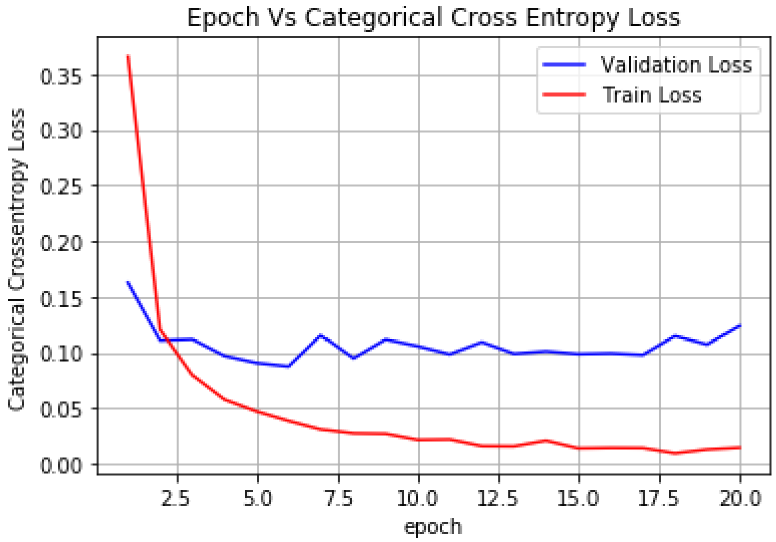
```
797 - acc: 0.9756 - val_loss: 0.1117 - val_acc: 0.9679
Epoch 4/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0
577 - acc: 0.9824 - val_loss: 0.0969 - val_acc: 0.9733
Epoch 5/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0
470 - acc: 0.9854 - val_loss: 0.0905 - val_acc: 0.9742
Epoch 6/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0
384 - acc: 0.9878 - val_loss: 0.0874 - val_acc: 0.9756
Epoch 7/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0
307 - acc: 0.9901 - val_loss: 0.1153 - val_acc: 0.9696
Epoch 8/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0
272 - acc: 0.9909 - val_loss: 0.0947 - val_acc: 0.9748
Epoch 9/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0
268 - acc: 0.9912 - val_loss: 0.1115 - val_acc: 0.9725
Epoch 10/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0
213 - acc: 0.9929 - val_loss: 0.1055 - val_acc: 0.9763
Epoch 11/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0
216 - acc: 0.9929 - val_loss: 0.0983 - val_acc: 0.9764
Epoch 12/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0
157 - acc: 0.9948 - val_loss: 0.1091 - val_acc: 0.9753
Epoch 13/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0
155 - acc: 0.9947 - val_loss: 0.0988 - val_acc: 0.9766
Epoch 14/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0
206 - acc: 0.9936 - val_loss: 0.1010 - val_acc: 0.9774
Epoch 15/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0
137 - acc: 0.9956 - val_loss: 0.0986 - val_acc: 0.9787
Epoch 16/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0
```

```
140 - acc: 0.9951 - val_loss: 0.0991 - val_acc: 0.9799
Epoch 17/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0
139 - acc: 0.9956 - val_loss: 0.0978 - val_acc: 0.9792
Epoch 18/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0
092 - acc: 0.9970 - val_loss: 0.1148 - val_acc: 0.9773
Epoch 19/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0
126 - acc: 0.9960 - val_loss: 0.1070 - val_acc: 0.9778
Epoch 20/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0
142 - acc: 0.9957 - val_loss: 0.1238 - val_acc: 0.9763
```

In [96]:

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

accuracy_3a= 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))
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

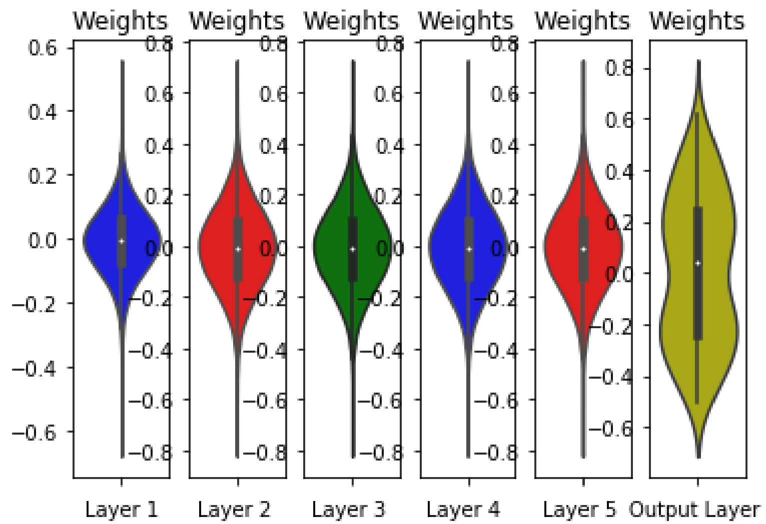
```
Test score: 0.12384451112879369
Test accuacy: 0.9763
```



In [97]:

```
w_after = model_relu.get_weights()

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


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

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

```
plt.subplot(1, 6, 3)
plt.title("Weights")
ax = sns.violinplot(y=h2_w, color='g')
plt.xlabel('Layer 3 ')

plt.subplot(1, 6, 4)
plt.title("Weights")
ax = sns.violinplot(y=h2_w, color='b')
plt.xlabel('Layer 4 ')

plt.subplot(1, 6, 5)
plt.title("Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Layer 5 ')

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



3 (b) MLP + ReLU + ADAM + Batch Normalization with 5 hidden layers

In [98]:

```
model_relu = Sequential()

# for relu layers
# If we sample weights from a normal distribution N(0,σ) we satisfy thi
s condition with σ=√(2/(ni).
# h1 =>  σ=√(2/(fan_in) = 0.062  => N(0,σ) = N(0,0.073)
# h2 =>  σ=√(2/(fan_in) = 0.125  => N(0,σ) = N(0,0.133)

model_relu.add(Dense(368, activation='relu', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.073, seed=None)))
model_relu.add(BatchNormalization())

model_relu.add(Dense(112, activation='relu',kernel_initializer=RandomNo
rmal(mean =0.0, stddev=0.133, seed=None)))
model_relu.add(BatchNormalization())

model_relu.add(Dense(54, activation='relu',kernel_initializer=RandomNor
mal(mean =0.0, stddev=0.192, seed=None)))
model_relu.add(BatchNormalization())

model_relu.add(Dense(28, activation='relu',kernel_initializer=RandomNor
mal(mean =0.0, stddev=0.267, seed=None)))
model_relu.add(BatchNormalization())

model_relu.add(Dense(16, activation='relu',kernel_initializer=RandomNor
mal(mean =0.0, stddev=0.353, seed=None)))
model_relu.add(BatchNormalization())

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

print(model_relu.summary())
print('\n')
model_relu.compile(optimizer='adam', loss='categorical_crossentropy', m
etrics=['accuracy'])
```

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_94 (Dense)             (None, 368)               288880    
_________________________________________________________________
batch_normalization_50 (Batc (None, 368)               1472      
_________________________________________________________________
dense_95 (Dense)             (None, 112)               41328     
_________________________________________________________________
batch_normalization_51 (Batc (None, 112)               448       
_________________________________________________________________
dense_96 (Dense)             (None, 54)                6102      
_________________________________________________________________
batch_normalization_52 (Batc (None, 54)                216       
_________________________________________________________________
dense_97 (Dense)             (None, 28)                1540      
_________________________________________________________________
batch_normalization_53 (Batc (None, 28)                112       
_________________________________________________________________
dense_98 (Dense)             (None, 16)                464       
_________________________________________________________________
batch_normalization_54 (Batc (None, 16)                64        
_________________________________________________________________
dense_99 (Dense)             (None, 10)                170       
=================================================================
Total params: 340,796
Trainable params: 339,640
Non-trainable params: 1,156
_________________________________________________________________
None


Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 10s 173us/step - loss:
0.4396 - acc: 0.8921 - val_loss: 0.1558 - val_acc: 0.9576
Epoch 2/20
```

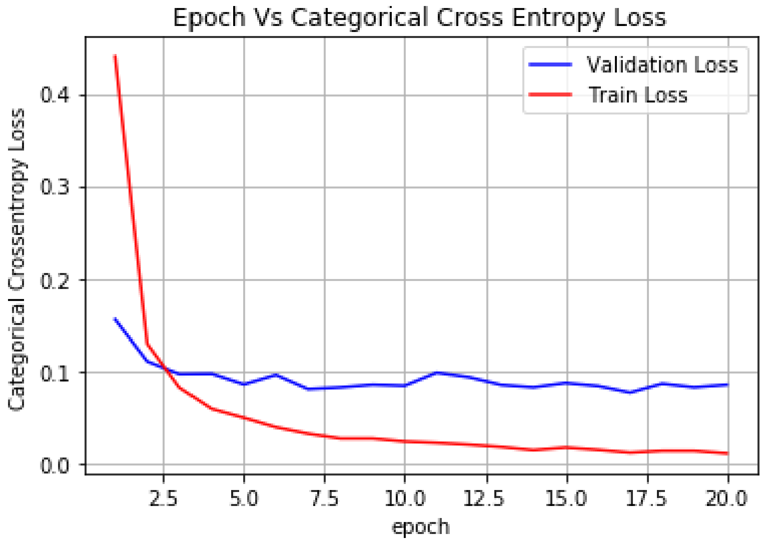
```
60000/60000 [==============================] - 5s 89us/step - loss: 0.1
295 - acc: 0.9650 - val_loss: 0.1105 - val_acc: 0.9679
Epoch 3/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0
822 - acc: 0.9767 - val_loss: 0.0968 - val_acc: 0.9718
Epoch 4/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0
592 - acc: 0.9830 - val_loss: 0.0971 - val_acc: 0.9707
Epoch 5/20
60000/60000 [==============================] - 5s 87us/step - loss: 0.0
496 - acc: 0.9848 - val_loss: 0.0857 - val_acc: 0.9755
Epoch 6/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0
394 - acc: 0.9879 - val_loss: 0.0959 - val_acc: 0.9732
Epoch 7/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0
324 - acc: 0.9905 - val_loss: 0.0806 - val_acc: 0.9759
Epoch 8/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0
271 - acc: 0.9916 - val_loss: 0.0825 - val_acc: 0.9756
Epoch 9/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.0
271 - acc: 0.9912 - val_loss: 0.0854 - val_acc: 0.9780
Epoch 10/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0
239 - acc: 0.9919 - val_loss: 0.0843 - val_acc: 0.9777
Epoch 11/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0
225 - acc: 0.9926 - val_loss: 0.0984 - val_acc: 0.9734
Epoch 12/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0
205 - acc: 0.9934 - val_loss: 0.0934 - val_acc: 0.9764
Epoch 13/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0
179 - acc: 0.9940 - val_loss: 0.0851 - val_acc: 0.9782
Epoch 14/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.0
145 - acc: 0.9954 - val_loss: 0.0825 - val_acc: 0.9781
Epoch 15/20
```

```
60000/60000 [==============================] - 5s 91us/step - loss: 0.0
173 - acc: 0.9944 - val_loss: 0.0872 - val_acc: 0.9781
Epoch 16/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.0
148 - acc: 0.9952 - val_loss: 0.0839 - val_acc: 0.9780
Epoch 17/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0
118 - acc: 0.9961 - val_loss: 0.0768 - val_acc: 0.9799
Epoch 18/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0
138 - acc: 0.9953 - val_loss: 0.0865 - val_acc: 0.9789
Epoch 19/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0
137 - acc: 0.9956 - val_loss: 0.0826 - val_acc: 0.9790
Epoch 20/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.0
110 - acc: 0.9966 - val_loss: 0.0853 - val_acc: 0.9794
```

In [99]:

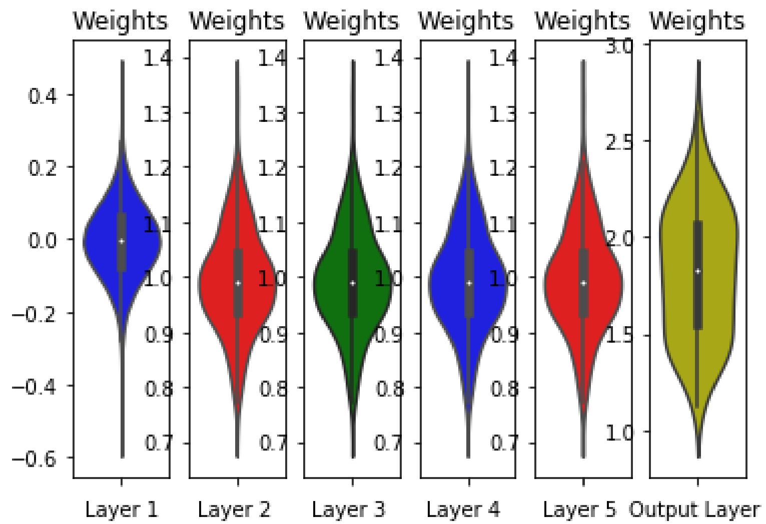
```
score = model_relu.evaluate(X_test, Y_test, verbose=0)
print ('Test score:', score[0])
print ('Test accuacy:', score[1])

accuracy_3b= 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))
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.08531874578401911
Test accuacy: 0.9794
```



In [100]:
```
w_after = model_relu.get_weights()

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


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

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

```
plt.subplot(1, 6, 3)
plt.title("Weights")
ax = sns.violinplot(y=h2_w, color='g')
plt.xlabel('Layer 3 ')

plt.subplot(1, 6, 4)
plt.title("Weights")
ax = sns.violinplot(y=h2_w, color='b')
plt.xlabel('Layer 4 ')

plt.subplot(1, 6, 5)
plt.title("Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Layer 5 ')

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



3 (c) MLP + ReLU + ADAM + Batch Normalization + Dropout(0.4) with 5 hidden layers

In [101]:

```
model_relu = Sequential()

# for relu layers
# If we sample weights from a normal distribution N(0,σ) we satisfy thi
s condition with σ=√(2/(ni).
# h1 =>  σ=√(2/(fan_in) = 0.062  => N(0,σ) = N(0,0.073)
# h2 =>  σ=√(2/(fan_in) = 0.125  => N(0,σ) = N(0,0.133)

model_relu.add(Dense(368, activation='relu', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.073, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.4))

model_relu.add(Dense(112, activation='relu',kernel_initializer=RandomNo
rmal(mean =0.0, stddev=0.133, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.4))

model_relu.add(Dense(54, activation='relu',kernel_initializer=RandomNor
mal(mean =0.0, stddev=0.192, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.4))

model_relu.add(Dense(28, activation='relu',kernel_initializer=RandomNor
mal(mean =0.0, stddev=0.267, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.4))

model_relu.add(Dense(16, activation='relu',kernel_initializer=RandomNor
mal(mean =0.0, stddev=0.353, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.4))

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

```
print(model_relu.summary())
print('\n')
model_relu.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = model_relu.fit(X_train, Y_train, batch_size = batch_size, epochs = nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_100 (Dense)            (None, 368)               288880    
_________________________________________________________________
batch_normalization_55 (Batc (None, 368)               1472      
_________________________________________________________________
dropout_35 (Dropout)         (None, 368)               0         
_________________________________________________________________
dense_101 (Dense)            (None, 112)               41328     
_________________________________________________________________
batch_normalization_56 (Batc (None, 112)               448       
_________________________________________________________________
dropout_36 (Dropout)         (None, 112)               0         
_________________________________________________________________
dense_102 (Dense)            (None, 54)                6102      
_________________________________________________________________
batch_normalization_57 (Batc (None, 54)                216       
_________________________________________________________________
dropout_37 (Dropout)         (None, 54)                0         
_________________________________________________________________
dense_103 (Dense)            (None, 28)                1540      
_________________________________________________________________
batch_normalization_58 (Batc (None, 28)                112       
_________________________________________________________________
dropout_38 (Dropout)         (None, 28)                0         
_________________________________________________________________
dense_104 (Dense)            (None, 16)                464       
_________________________________________________________________
batch_normalization_59 (Batc (None, 16)                64        
_________________________________________________________________
dropout_39 (Dropout)         (None, 16)                0         
_________________________________________________________________
```

```
dense_105 (Dense)            (None, 10)                170
=================================================================
Total params: 340,796
Trainable params: 339,640
Non-trainable params: 1,156
_________________________________________________________________
None


Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 11s 185us/step - loss:
1.7393 - acc: 0.4098 - val_loss: 0.6295 - val_acc: 0.8676
Epoch 2/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.9
038 - acc: 0.7119 - val_loss: 0.2672 - val_acc: 0.9344
Epoch 3/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.5
987 - acc: 0.8227 - val_loss: 0.1998 - val_acc: 0.9473
Epoch 4/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.4
686 - acc: 0.8728 - val_loss: 0.1606 - val_acc: 0.9600
Epoch 5/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.4
086 - acc: 0.8923 - val_loss: 0.1498 - val_acc: 0.9634
Epoch 6/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.3
552 - acc: 0.9100 - val_loss: 0.1305 - val_acc: 0.9682
Epoch 7/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.3
229 - acc: 0.9191 - val_loss: 0.1266 - val_acc: 0.9688
Epoch 8/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.2
989 - acc: 0.9263 - val_loss: 0.1205 - val_acc: 0.9713
Epoch 9/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.2
848 - acc: 0.9300 - val_loss: 0.1136 - val_acc: 0.9721
Epoch 10/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.2
```

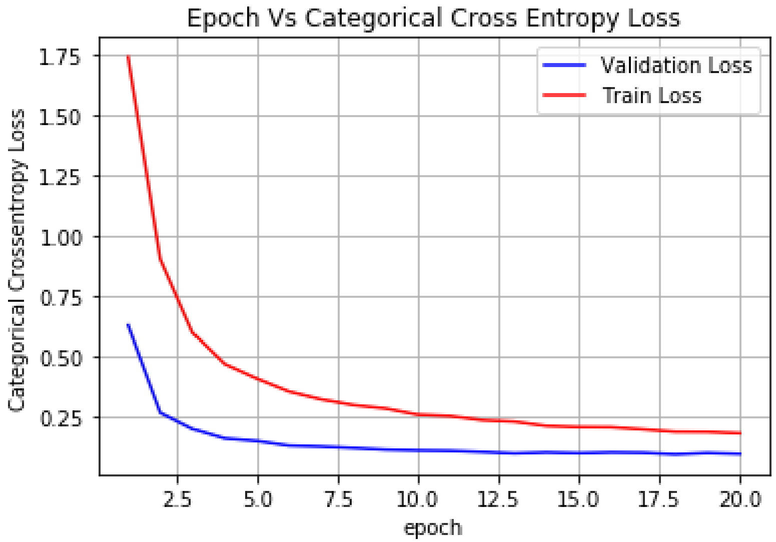
```
591 - acc: 0.9367 - val_loss: 0.1105 - val_acc: 0.9748
Epoch 11/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.2
535 - acc: 0.9405 - val_loss: 0.1090 - val_acc: 0.9757
Epoch 12/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.2
362 - acc: 0.9430 - val_loss: 0.1039 - val_acc: 0.9757
Epoch 13/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.2
301 - acc: 0.9460 - val_loss: 0.0982 - val_acc: 0.9779
Epoch 14/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.2
122 - acc: 0.9485 - val_loss: 0.1020 - val_acc: 0.9780
Epoch 15/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.2
077 - acc: 0.9505 - val_loss: 0.0991 - val_acc: 0.9790
Epoch 16/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.2
069 - acc: 0.9517 - val_loss: 0.1021 - val_acc: 0.9781
Epoch 17/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.1
980 - acc: 0.9531 - val_loss: 0.1009 - val_acc: 0.9800
Epoch 18/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.1
881 - acc: 0.9557 - val_loss: 0.0942 - val_acc: 0.9792
Epoch 19/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.1
875 - acc: 0.9569 - val_loss: 0.0998 - val_acc: 0.9791
Epoch 20/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.1
820 - acc: 0.9572 - val_loss: 0.0960 - val_acc: 0.9797
```

In [102]:

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

accuracy_3c= 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))
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

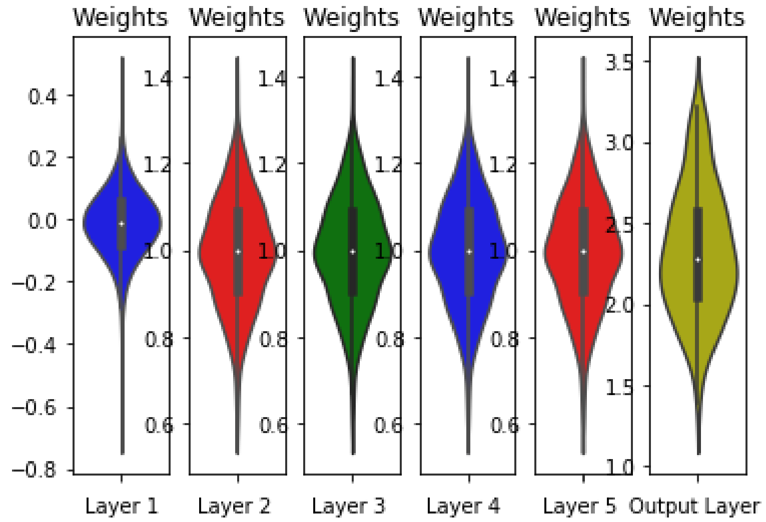
```
Test score: 0.09595172328427434
Test accuacy: 0.9797
```



In [103]:

```
w_after = model_relu.get_weights()

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

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

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

plt.subplot(1, 6, 3)
plt.title("Weights")
ax = sns.violinplot(y=h2_w, color='g')
plt.xlabel('Layer 3 ')

plt.subplot(1, 6, 4)
plt.title("Weights")
ax = sns.violinplot(y=h2_w, color='b')
plt.xlabel('Layer 4 ')

plt.subplot(1, 6, 5)
plt.title("Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Layer 5 ')

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



3 (d) MLP + ReLU + ADAM + Batch Normalization + Dropout(0.5) with 5 hidden layers

In [104]:

```
model_relu = Sequential()

# for relu layers
# If we sample weights from a normal distribution N(0,σ) we satisfy thi
s condition with σ=√(2/(ni).
# h1 =>  σ=√(2/(fan_in) = 0.062  => N(0,σ) = N(0,0.073)
# h2 =>  σ=√(2/(fan_in) = 0.125  => N(0,σ) = N(0,0.133)

model_relu.add(Dense(368, activation='relu', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.073, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))

model_relu.add(Dense(112, activation='relu',kernel_initializer=RandomNo
rmal(mean =0.0, stddev=0.133, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
```

```
model_relu.add(Dense(54, activation='relu',kernel_initializer=RandomNor
mal(mean =0.0, stddev=0.192, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))

model_relu.add(Dense(28, activation='relu',kernel_initializer=RandomNor
mal(mean =0.0, stddev=0.267, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))

model_relu.add(Dense(16, activation='relu',kernel_initializer=RandomNor
mal(mean =0.0, stddev=0.353, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))

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

print(model_relu.summary())
print('\n')
model_relu.compile(optimizer='adam', loss='categorical_crossentropy', m
etrics=['accuracy'])
history = model_relu.fit(X_train, Y_train, batch_size = batch_size, epo
chs = nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_106 (Dense)            (None, 368)               288880    
_________________________________________________________________
batch_normalization_60 (Batc (None, 368)               1472      
_________________________________________________________________
dropout_40 (Dropout)         (None, 368)               0         
_________________________________________________________________
dense_107 (Dense)            (None, 112)               41328     
_________________________________________________________________
batch_normalization_61 (Batc (None, 112)               448       
_________________________________________________________________
dropout_41 (Dropout)         (None, 112)               0         
_________________________________________________________________
```

```
dense_108 (Dense)            (None, 54)                6102
_________________________________________________________________
batch_normalization_62 (Batc (None, 54)                216
_________________________________________________________________
dropout_42 (Dropout)         (None, 54)                0
_________________________________________________________________
dense_109 (Dense)            (None, 28)                1540
_________________________________________________________________
batch_normalization_63 (Batc (None, 28)                112
_________________________________________________________________
dropout_43 (Dropout)         (None, 28)                0
_________________________________________________________________
dense_110 (Dense)            (None, 16)                464
_________________________________________________________________
batch_normalization_64 (Batc (None, 16)                64
_________________________________________________________________
dropout_44 (Dropout)         (None, 16)                0
_________________________________________________________________
dense_111 (Dense)            (None, 10)                170
=================================================================
Total params: 340,796
Trainable params: 339,640
Non-trainable params: 1,156
_________________________________________________________________
None


Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 11s 191us/step - loss:
2.1301 - acc: 0.2682 - val_loss: 1.0570 - val_acc: 0.6764
Epoch 2/20
60000/60000 [==============================] - 6s 93us/step - loss: 1.3
601 - acc: 0.5146 - val_loss: 0.6491 - val_acc: 0.8335
Epoch 3/20
60000/60000 [==============================] - 6s 97us/step - loss: 1.0
379 - acc: 0.6367 - val_loss: 0.4739 - val_acc: 0.9047
Epoch 4/20
60000/60000 [==============================] - 6s 100us/step - loss: 0.
```

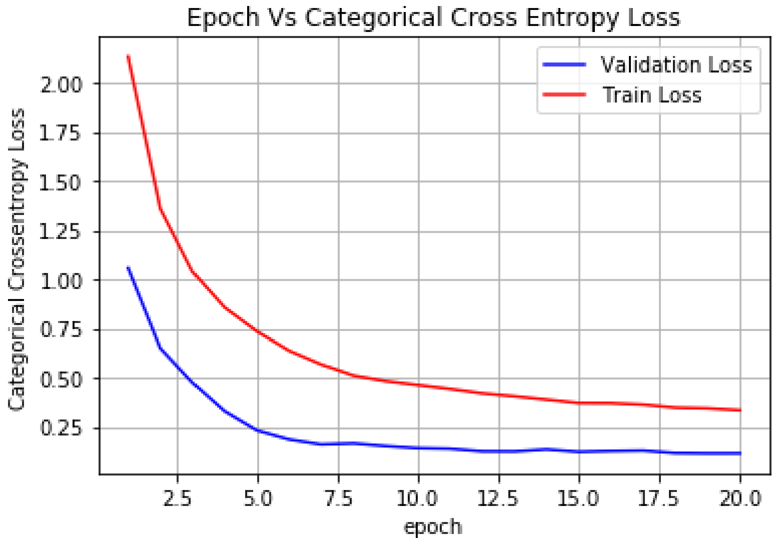
```
8556 - acc: 0.7115 - val_loss: 0.3282 - val_acc: 0.9279
Epoch 5/20
60000/60000 [==============================] - 6s 98us/step - loss: 0.7
351 - acc: 0.7641 - val_loss: 0.2309 - val_acc: 0.9501
Epoch 6/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.6
363 - acc: 0.8065 - val_loss: 0.1850 - val_acc: 0.9536
Epoch 7/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.5
664 - acc: 0.8297 - val_loss: 0.1596 - val_acc: 0.9617
Epoch 8/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.5
100 - acc: 0.8508 - val_loss: 0.1647 - val_acc: 0.9613
Epoch 9/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.4
821 - acc: 0.8604 - val_loss: 0.1511 - val_acc: 0.9650
Epoch 10/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.4
623 - acc: 0.8682 - val_loss: 0.1405 - val_acc: 0.9682
Epoch 11/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.4
424 - acc: 0.8740 - val_loss: 0.1375 - val_acc: 0.9687
Epoch 12/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.4
197 - acc: 0.8821 - val_loss: 0.1242 - val_acc: 0.9713
Epoch 13/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.4
048 - acc: 0.8871 - val_loss: 0.1237 - val_acc: 0.9718
Epoch 14/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.3
879 - acc: 0.8919 - val_loss: 0.1340 - val_acc: 0.9702
Epoch 15/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.3
706 - acc: 0.8962 - val_loss: 0.1218 - val_acc: 0.9739
Epoch 16/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.3
696 - acc: 0.8964 - val_loss: 0.1258 - val_acc: 0.9737
Epoch 17/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.3
```

```
624 - acc: 0.8993 - val_loss: 0.1280 - val_acc: 0.9734
Epoch 18/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.3
470 - acc: 0.9017 - val_loss: 0.1154 - val_acc: 0.9758
Epoch 19/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.3
434 - acc: 0.9042 - val_loss: 0.1138 - val_acc: 0.9767
Epoch 20/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.3
339 - acc: 0.9066 - val_loss: 0.1138 - val_acc: 0.9774
```

In [105]:

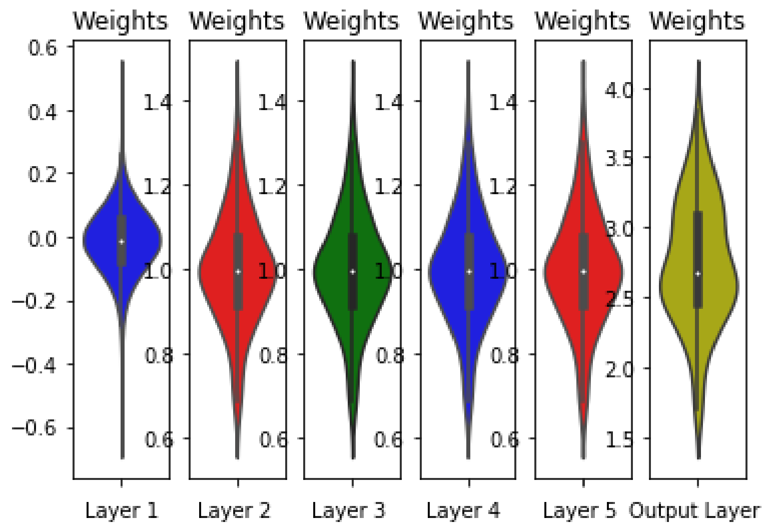
```
score = model_relu.evaluate(X_test, Y_test, verbose=0)
print ('Test score:', score[0])
print ('Test accuacy:', score[1])

accuracy_3d= 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))
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.11384864466506987
Test accuacy: 0.9774
```



In [106]:
```
w_after = model_relu.get_weights()

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


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

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

```
plt.subplot(1, 6, 3)
plt.title("Weights")
ax = sns.violinplot(y=h2_w, color='g')
plt.xlabel('Layer 3 ')

plt.subplot(1, 6, 4)
plt.title("Weights")
ax = sns.violinplot(y=h2_w, color='b')
plt.xlabel('Layer 4 ')

plt.subplot(1, 6, 5)
plt.title("Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Layer 5 ')

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



3 (e) MLP + ReLU + ADAM + Batch Normalization + Dropout(0.6) with 5 hidden layers

In [107]:

```
model_relu = Sequential()

# for relu layers
# If we sample weights from a normal distribution N(0,σ) we satisfy thi
s condition with σ=√(2/(ni).
# h1 =>  σ=√(2/(fan_in) = 0.062  => N(0,σ) = N(0,0.073)
# h2 =>  σ=√(2/(fan_in) = 0.125  => N(0,σ) = N(0,0.133)

model_relu.add(Dense(368, activation='relu', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.073, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.6))

model_relu.add(Dense(112, activation='relu',kernel_initializer=RandomNo
rmal(mean =0.0, stddev=0.133, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.6))

model_relu.add(Dense(54, activation='relu',kernel_initializer=RandomNor
mal(mean =0.0, stddev=0.192, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.6))

model_relu.add(Dense(28, activation='relu',kernel_initializer=RandomNor
mal(mean =0.0, stddev=0.267, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.6))

model_relu.add(Dense(16, activation='relu',kernel_initializer=RandomNor
mal(mean =0.0, stddev=0.353, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.6))

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

```
print(model_relu.summary())
print('\n')
model_relu.compile(optimizer='adam', loss='categorical_crossentropy', m
etrics=['accuracy'])
history = model_relu.fit(X_train, Y_train, batch_size = batch_size, epo
chs = nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_112 (Dense)            (None, 368)               288880    
_________________________________________________________________
batch_normalization_65 (Batc (None, 368)               1472      
_________________________________________________________________
dropout_45 (Dropout)         (None, 368)               0         
_________________________________________________________________
dense_113 (Dense)            (None, 112)               41328     
_________________________________________________________________
batch_normalization_66 (Batc (None, 112)               448       
_________________________________________________________________
dropout_46 (Dropout)         (None, 112)               0         
_________________________________________________________________
dense_114 (Dense)            (None, 54)                6102      
_________________________________________________________________
batch_normalization_67 (Batc (None, 54)                216       
_________________________________________________________________
dropout_47 (Dropout)         (None, 54)                0         
_________________________________________________________________
dense_115 (Dense)            (None, 28)                1540      
_________________________________________________________________
batch_normalization_68 (Batc (None, 28)                112       
_________________________________________________________________
dropout_48 (Dropout)         (None, 28)                0         
_________________________________________________________________
dense_116 (Dense)            (None, 16)                464       
_________________________________________________________________
batch_normalization_69 (Batc (None, 16)                64        
_________________________________________________________________
dropout_49 (Dropout)         (None, 16)                0         
_________________________________________________________________
```

```
dense_117 (Dense)            (None, 10)                170
=================================================================
Total params: 340,796
Trainable params: 339,640
Non-trainable params: 1,156
_________________________________________________________________
None


Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 12s 204us/step - loss: 2.4759 - acc: 0.1602 - val_loss: 1.7847 - val_acc: 0.4354
Epoch 2/20
60000/60000 [==============================] - 6s 95us/step - loss: 1.9158 - acc: 0.2868 - val_loss: 1.4256 - val_acc: 0.6718
Epoch 3/20
60000/60000 [==============================] - 6s 95us/step - loss: 1.6487 - acc: 0.3883 - val_loss: 1.0411 - val_acc: 0.7306
Epoch 4/20
60000/60000 [==============================] - 6s 97us/step - loss: 1.4051 - acc: 0.4742 - val_loss: 0.7940 - val_acc: 0.7484
Epoch 5/20
60000/60000 [==============================] - 6s 97us/step - loss: 1.2284 - acc: 0.5322 - val_loss: 0.6732 - val_acc: 0.8191
Epoch 6/20
60000/60000 [==============================] - 6s 94us/step - loss: 1.1199 - acc: 0.5722 - val_loss: 0.6110 - val_acc: 0.8291
Epoch 7/20
60000/60000 [==============================] - 6s 94us/step - loss: 1.0372 - acc: 0.6057 - val_loss: 0.5386 - val_acc: 0.8503
Epoch 8/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.9872 - acc: 0.6276 - val_loss: 0.5028 - val_acc: 0.8442
Epoch 9/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.9467 - acc: 0.6433 - val_loss: 0.4719 - val_acc: 0.8572
Epoch 10/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.9
```

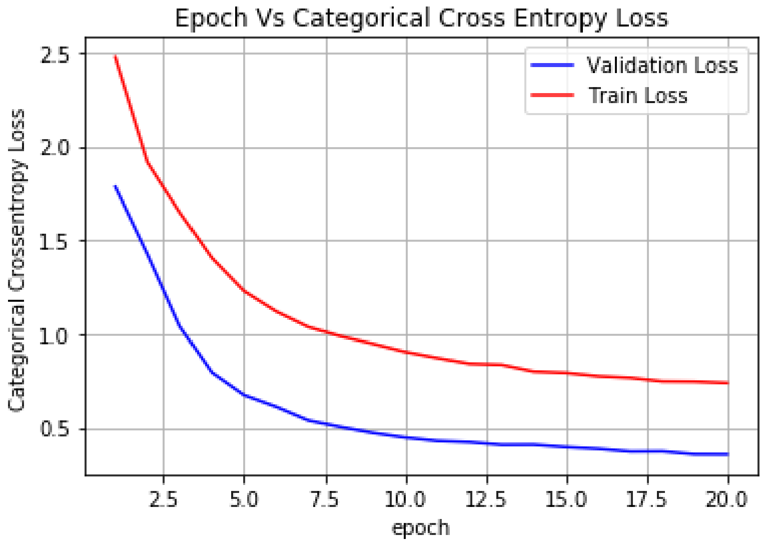
```
038 - acc: 0.6594 - val_loss: 0.4472 - val_acc: 0.8301
Epoch 11/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.8
713 - acc: 0.6673 - val_loss: 0.4292 - val_acc: 0.8188
Epoch 12/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.8
404 - acc: 0.6760 - val_loss: 0.4225 - val_acc: 0.8353
Epoch 13/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.8
349 - acc: 0.6808 - val_loss: 0.4083 - val_acc: 0.8536
Epoch 14/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.7
990 - acc: 0.6894 - val_loss: 0.4089 - val_acc: 0.8408
Epoch 15/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.7
921 - acc: 0.6915 - val_loss: 0.3960 - val_acc: 0.8587
Epoch 16/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.7
742 - acc: 0.7013 - val_loss: 0.3867 - val_acc: 0.9119
Epoch 17/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.7
657 - acc: 0.7036 - val_loss: 0.3724 - val_acc: 0.8687
Epoch 18/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.7
468 - acc: 0.7080 - val_loss: 0.3733 - val_acc: 0.9071
Epoch 19/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.7
454 - acc: 0.7083 - val_loss: 0.3576 - val_acc: 0.8720
Epoch 20/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.7
385 - acc: 0.7146 - val_loss: 0.3569 - val_acc: 0.8666
```

In [108]:

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

accuracy_3e= score[1]

fig, ax = plt.subplots(1,1)
```

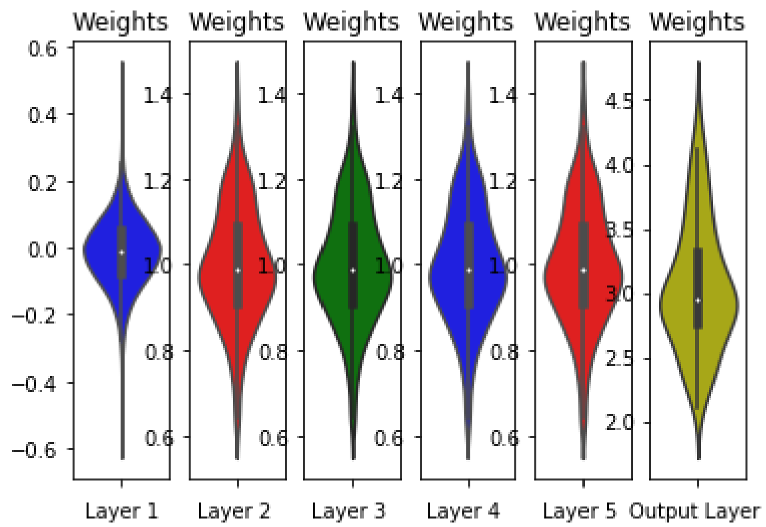
```
ax.set_xlabel('epoch')
ax.set_ylabel('Categorical Crossentropy Loss')

#List of epoch numbers
x = list(range(1, nb_epoch+1))
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.35688953695297243
Test accuacy: 0.8666
```



In [109]:
```
w_after = model_relu.get_weights()

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

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

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

plt.subplot(1, 6, 3)
plt.title("Weights")
ax = sns.violinplot(y=h2_w, color='g')
plt.xlabel('Layer 3 ')

plt.subplot(1, 6, 4)
plt.title("Weights")
ax = sns.violinplot(y=h2_w, color='b')
plt.xlabel('Layer 4 ')

plt.subplot(1, 6, 5)
plt.title("Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Layer 5 ')

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



4 a) Procedure Followed:

STEP 1: Load the MNIST dataset, shuffle the data and split it between train and test.

STEP 2: Normalize the input data and one-hot encode class label data before feeding into the model. This conversion is needed for Multi layer perceptron.

STEP 3: Add the hidden layers and provide the required input parameters like number of neurons for each layer, activation unit, input shape and weight initilization with kernel_initializer.

STEP 4: Compile the model and fit it on training data.

STEP 5: Evaluate the model on test data and plot the error plots for each epoch.

STEP 6: Get the weights from trained model and plot them to see their distribution.

STEP 7: Repeat the steps from 3 to 6 for hidden layers 2, 3, 5 and try the cases with batch normalization and different dropout for each of the architectures.

STEP 8: Summarize the test accuaracy for each model.

4 b) Models performance comparision with different architectures

In [110]:

```
# comparing Models performance using Prettytable library
from prettytable import PrettyTable
x = PrettyTable()

# Name of the models
models =['MLP + ReLU + ADAM with 2 hidden layers                      ',
         'MLP + ReLU + ADAM + Batch Normalization with 2 hidden layers'
,
         'MLP + ReLU + ADAM + Batch Normalization + Dropout(0.4) with 2
 hidden layers',
         'MLP + ReLU + ADAM + Batch Normalization + Dropout(0.5) with 2
 hidden layers',
         'MLP + ReLU + ADAM + Batch Normalization + Dropout(0.6) with 2
 hidden layers',
        'MLP + ReLU + ADAM with 3 hidden layers                      ',
         'MLP + ReLU + ADAM + Batch Normalization with 3 hidden layers'
,
         'MLP + ReLU + ADAM + Batch Normalization + Dropout(0.4) with 3
 hidden layers',
         'MLP + ReLU + ADAM + Batch Normalization + Dropout(0.5) with 3
 hidden layers',
         'MLP + ReLU + ADAM + Batch Normalization + Dropout(0.6) with 3
 hidden layers',
        'MLP + ReLU + ADAM with 5 hidden layers                      ',
         'MLP + ReLU + ADAM + Batch Normalization with 5 hidden layers'
,
         'MLP + ReLU + ADAM + Batch Normalization + Dropout(0.4) with 5
 hidden layers',
         'MLP + ReLU + ADAM + Batch Normalization + Dropout(0.5) with 5
 hidden layers',
         'MLP + ReLU + ADAM + Batch Normalization + Dropout(0.6) with 5
 hidden layers'
        ]
```

```
# Test Accuracy
test_accuracy = [accuracy_1a,accuracy_1b,accuracy_1c,accuracy_1d,accura
cy_1e,accuracy_2a,accuracy_2b,accuracy_2c,accuracy_2d,accuracy_2e,accur
acy_3a,accuracy_3b,accuracy_3c,accuracy_3d,accuracy_3e]
sno = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]

# Adding columns
x.add_column("S.NO.",sno)
x.add_column("MLP MODEL",models)
x.add_column("Test Accuracy" ,test_accuracy )

# Printing the Table
print(x)
```

```
+-------+-----------------------------------------------------------------
---------------+---------------+
| S.NO. |                                  MLP MODEL
               | Test Accuracy |
+-------+-----------------------------------------------------------------
---------------+---------------+
|   1   |         MLP + ReLU + ADAM with 2 hidden layers
               |     0.9815    |
|   2   |         MLP + ReLU + ADAM + Batch Normalization with 2 hidden
layers         |     0.9834    |
|   3   | MLP + ReLU + ADAM + Batch Normalization + Dropout(0.4) with 2
hidden layers |     0.9827    |
|   4   | MLP + ReLU + ADAM + Batch Normalization + Dropout(0.5) with 2
hidden layers |     0.9834    |
|   5   | MLP + ReLU + ADAM + Batch Normalization + Dropout(0.6) with 2
hidden layers |     0.9811    |
|   6   |         MLP + ReLU + ADAM with 3 hidden layers
               |     0.9791    |
|   7   |         MLP + ReLU + ADAM + Batch Normalization with 3 hidden
layers         |     0.979     |
|   8   | MLP + ReLU + ADAM + Batch Normalization + Dropout(0.4) with 3
hidden layers |     0.9821    |
|   9   | MLP + ReLU + ADAM + Batch Normalization + Dropout(0.5) with 3
hidden layers |     0.9804    |
|   10  | MLP + ReLU + ADAM + Batch Normalization + Dropout(0.6) with 3
hidden layers |     0.9786    |
```

```
|   11  |         MLP + ReLU + ADAM with 5 hidden layers
               |     0.9763    |
|   12  |         MLP + ReLU + ADAM + Batch Normalization with 5 hidden
layers        |     0.9794    |
|   13  | MLP + ReLU + ADAM + Batch Normalization + Dropout(0.4) with 5
hidden layers |     0.9797    |
|   14  | MLP + ReLU + ADAM + Batch Normalization + Dropout(0.5) with 5
hidden layers |     0.9774    |
|   15  | MLP + ReLU + ADAM + Batch Normalization + Dropout(0.6) with 5
hidden layers |     0.8666    |
+-------+-------------------------------------------------------------
---------------+--------------+
```

Observation:

1) It is observed that accuracy decreased slightly when dropout is more than 0.5.

2) Accuracy increased when batch normalization and dropout is added to the models.