Keras -- MLPs on MNIST

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

```
Using TensorFlow backend.
```

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

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

In [4]:
```
print("Number of training examples :", X_train.shape[0], "and each imag
e is of shape (%d, %d)"%(X_train.shape[1], X_train.shape[2]))
print("Number of training examples :", X_test.shape[0], "and each image
 is of shape (%d, %d)"%(X_test.shape[1], X_test.shape[2]))
```

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

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

In [0]:
```python
# if you observe the input shape its 2 dimensional vector
# for each image we have a (28*28) vector
# we will convert the (28*28) vector into single dimensional vector of
 1 * 784

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

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

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

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

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

In [0]:

```
# if we observe the above matrix each cell is having a value between 0-
255
# before we move to apply machine learning algorithms lets try to norma
lize the data
```

```python
# X => (X - Xmin)/(Xmax-Xmin) = X/255

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

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

```
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.01176471 0.07058824 0.07058824 0.07058824
 0.49411765 0.53333333 0.68627451 0.10196078 0.65098039 1.
 0.96862745 0.49803922 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.11764706 0.14117647 0.36862745 0.60392157
 0.66666667 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
```

```
 0.88235294 0.6745098  0.99215686 0.94901961 0.76470588 0.25098039
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.19215686
 0.93333333 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
 0.99215686 0.99215686 0.99215686 0.98431373 0.36470588 0.32156863
 0.32156863 0.21960784 0.15294118 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.07058824 0.85882353 0.99215686
 0.99215686 0.99215686 0.99215686 0.99215686 0.77647059 0.71372549
 0.96862745 0.94509804 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.31372549 0.61176471 0.41960784 0.99215686
 0.99215686 0.80392157 0.04313725 0.         0.16862745 0.60392157
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.05490196 0.00392157 0.60392157 0.99215686 0.35294118
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.54509804 0.99215686 0.74509804 0.00784314 0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.04313725
 0.74509804 0.99215686 0.2745098  0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.1372549  0.94509804
 0.88235294 0.62745098 0.42352941 0.00392157 0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.31764706 0.94117647 0.99215686
 0.99215686 0.46666667 0.09803922 0.         0.         0.
 0.         0.         0.         0.         0.         0.
```

```
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.17647059  0.72941176  0.99215686  0.99215686
 0.58823529  0.10588235  0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.0627451   0.36470588  0.98823529  0.99215686  0.73333333
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.97647059  0.99215686  0.97647059  0.25098039  0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.18039216  0.50980392  0.71764706  0.99215686
 0.99215686  0.81176471  0.00784314  0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.15294118  0.58039216
 0.89803922  0.99215686  0.99215686  0.99215686  0.98039216  0.71372549
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.09411765  0.44705882  0.86666667  0.99215686  0.99215686  0.99215686
 0.99215686  0.78823529  0.30588235  0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.09019608  0.25882353  0.83529412  0.99215686
 0.99215686  0.99215686  0.99215686  0.77647059  0.31764706  0.00784314
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.07058824  0.67058824
 0.85882353  0.99215686  0.99215686  0.99215686  0.99215686  0.76470588
 0.31372549  0.03529412  0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.21568627  0.6745098   0.88627451  0.99215686  0.99215686  0.99215686
```

```
 0.99215686 0.95686275 0.52156863 0.04313725 0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.53333333 0.99215686
 0.99215686 0.99215686 0.83137255 0.52941176 0.51764706 0.0627451
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
```

In [10]:

```
# here we are having a class number for each image
print("Class label of first image :", y_train[0])

# lets convert this into a 10 dimensional vector
# ex: consider an image is 5 convert it into 5 => [0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
# this conversion needed for MLPs

Y_train = np_utils.to_categorical(y_train, 10)
Y_test = np_utils.to_categorical(y_test, 10)

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

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

Softmax classifier

In [0]:
```
# https://keras.io/getting-started/sequential-model-guide/

# The Sequential model is a linear stack of layers.
# you can create a Sequential model by passing a list of layer instance
s to the constructor:

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

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

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

###

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

# keras.layers.Dense(units, activation=None, use_bias=True, kernel_init
ializer='glorot_uniform',
# bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=N
one, activity_regularizer=None,
# kernel_constraint=None, bias_constraint=None)

# Dense implements the operation: output = activation(dot(input, kerne
l) + bias) where
# activation is the element-wise activation function passed as the acti
vation argument,
# kernel is a weights matrix created by the layer, and
# bias is a bias vector created by the layer (only applicable if use_bi
```

```
as is True).

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

####

# https://keras.io/activations/

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

# from keras.layers import Activation, Dense

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

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

# there are many activation functions ar available ex: tanh, relu, soft
max


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
```

1.USING TWO HIDDEN LAYERS

a.MLP + ReLU + ADAM

In [62]:
```
model_relu = Sequential()
model_relu.add(Dense(396, activation='relu', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.071, seed=None)))
model_relu.add(Dense(198, activation='relu', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.1, seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_44 (Dense)             (None, 396)               310860
_________________________________________________________________
dense_45 (Dense)             (None, 198)               78606
_________________________________________________________________
dense_46 (Dense)             (None, 10)                1990
=================================================================
Total params: 391,456
Trainable params: 391,456
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.2
327 - acc: 0.9309 - val_loss: 0.1019 - val_acc: 0.9675
Epoch 2/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0
```

```
865 - acc: 0.9739 - val_loss: 0.0915 - val_acc: 0.9716
Epoch 3/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0
574 - acc: 0.9826 - val_loss: 0.0709 - val_acc: 0.9771
Epoch 4/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0
386 - acc: 0.9879 - val_loss: 0.0716 - val_acc: 0.9777
Epoch 5/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0
262 - acc: 0.9921 - val_loss: 0.0691 - val_acc: 0.9794
Epoch 6/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0
207 - acc: 0.9935 - val_loss: 0.0786 - val_acc: 0.9773
Epoch 7/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.0
182 - acc: 0.9943 - val_loss: 0.0695 - val_acc: 0.9799
Epoch 8/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.0
144 - acc: 0.9952 - val_loss: 0.0734 - val_acc: 0.9797
Epoch 9/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.0
130 - acc: 0.9958 - val_loss: 0.0860 - val_acc: 0.9774
Epoch 10/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.0
121 - acc: 0.9960 - val_loss: 0.0870 - val_acc: 0.9782
Epoch 11/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.0
098 - acc: 0.9969 - val_loss: 0.0862 - val_acc: 0.9800
Epoch 12/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.0
081 - acc: 0.9974 - val_loss: 0.0941 - val_acc: 0.9780
Epoch 13/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.0
111 - acc: 0.9964 - val_loss: 0.0881 - val_acc: 0.9794
Epoch 14/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0
108 - acc: 0.9963 - val_loss: 0.1193 - val_acc: 0.9754
Epoch 15/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0
```

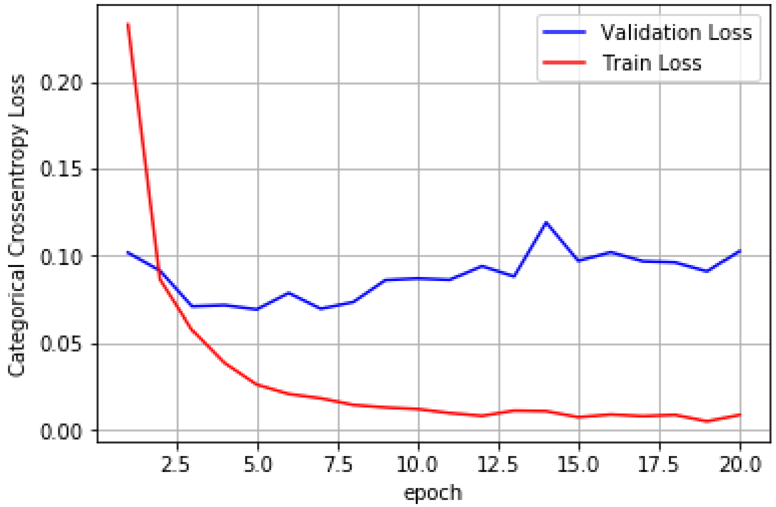
```
074 - acc: 0.9977 - val_loss: 0.0971 - val_acc: 0.9798
Epoch 16/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0
089 - acc: 0.9968 - val_loss: 0.1021 - val_acc: 0.9800
Epoch 17/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0
079 - acc: 0.9972 - val_loss: 0.0969 - val_acc: 0.9804
Epoch 18/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0
087 - acc: 0.9970 - val_loss: 0.0962 - val_acc: 0.9804
Epoch 19/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0
050 - acc: 0.9982 - val_loss: 0.0911 - val_acc: 0.9809
Epoch 20/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0
086 - acc: 0.9970 - val_loss: 0.1027 - val_acc: 0.9801
```

In [63]:

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

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

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

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

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

# loss : training loss
# acc : train accuracy
```

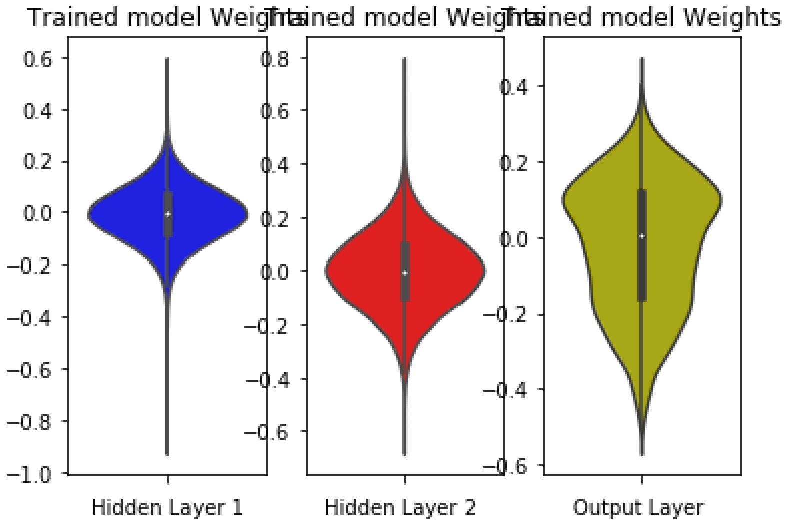
```
# for each key in histrory.histrory we will have a list of length equal
 to number of epochs


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

```
Test score: 0.10270734503133722
Test accuracy: 0.9801
```



In [64]:
```
w_after = model_relu.get_weights()

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


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

```
plt.xlabel('Hidden Layer 1')

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

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



(b)MLP + Batch-Norm on hidden Layers + AdamOptimizer +RELU</2>

In [65]:
```
# Multilayer perceptron

# https://intoli.com/blog/neural-network-initialization/
# If we sample weights from a normal distribution N(0,σ) we satisfy thi
```

```
s condition with σ=√(2/(ni+ni+1).
# h1 =>  σ=√(2/(ni+ni+1) = 0.041  => N(0,σ) = N(0,0.041)
# h2 =>  σ=√(2/(ni+ni+1) = 0.058  => N(0,σ) = N(0,0.058)
# out =>  σ=√(2/(ni+ni+1) = 0.098  => N(0,σ) = N(0,0.98)

from keras.layers.normalization import BatchNormalization

model_batch = Sequential()

model_batch.add(Dense(396, activation='relu', input_shape=(input_dim,),
 kernel_initializer=RandomNormal(mean=0.0, stddev=0.041, seed=None)))
model_batch.add(BatchNormalization())

model_batch.add(Dense(198, activation='relu', kernel_initializer=Random
Normal(mean=0.0, stddev=0.58, seed=None)) )
model_batch.add(BatchNormalization())

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


model_batch.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_47 (Dense)             (None, 396)               310860
_________________________________________________________________
batch_normalization_9 (Batch (None, 396)               1584
_________________________________________________________________
dense_48 (Dense)             (None, 198)               78606
_________________________________________________________________
batch_normalization_10 (Batc (None, 198)               792
_________________________________________________________________
dense_49 (Dense)             (None, 10)                1990
=================================================================
Total params: 393,832
Trainable params: 392,644
Non-trainable params: 1,188
_________________________________________________________________
```

In [66]:
```
model_batch.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.1
936 - acc: 0.9422 - val_loss: 0.1086 - val_acc: 0.9683
Epoch 2/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0
754 - acc: 0.9774 - val_loss: 0.0879 - val_acc: 0.9732
Epoch 3/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.0
486 - acc: 0.9851 - val_loss: 0.0773 - val_acc: 0.9780
Epoch 4/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.0
344 - acc: 0.9895 - val_loss: 0.0825 - val_acc: 0.9768
Epoch 5/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.0
253 - acc: 0.9922 - val_loss: 0.0682 - val_acc: 0.9795
Epoch 6/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.0
198 - acc: 0.9940 - val_loss: 0.0832 - val_acc: 0.9756
Epoch 7/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.0
150 - acc: 0.9957 - val_loss: 0.0825 - val_acc: 0.9768
Epoch 8/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.0
151 - acc: 0.9951 - val_loss: 0.0977 - val_acc: 0.9721
Epoch 9/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.0
162 - acc: 0.9946 - val_loss: 0.0842 - val_acc: 0.9761
Epoch 10/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.0
119 - acc: 0.9963 - val_loss: 0.0856 - val_acc: 0.9785
Epoch 11/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.0
102 - acc: 0.9969 - val_loss: 0.0772 - val_acc: 0.9805
```

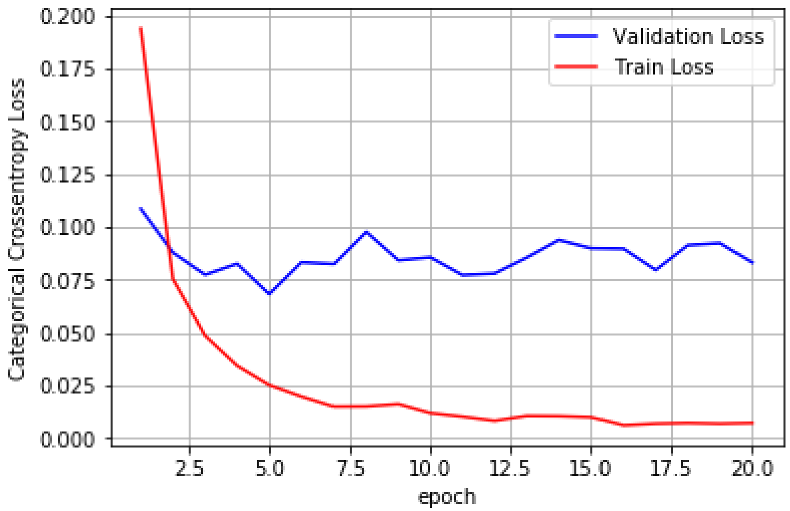
```
Epoch 12/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.0
083 - acc: 0.9974 - val_loss: 0.0779 - val_acc: 0.9793
Epoch 13/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.0
106 - acc: 0.9964 - val_loss: 0.0855 - val_acc: 0.9767
Epoch 14/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.0
105 - acc: 0.9965 - val_loss: 0.0938 - val_acc: 0.9762
Epoch 15/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.0
100 - acc: 0.9967 - val_loss: 0.0899 - val_acc: 0.9781
Epoch 16/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.0
061 - acc: 0.9981 - val_loss: 0.0896 - val_acc: 0.9792
Epoch 17/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.0
069 - acc: 0.9976 - val_loss: 0.0796 - val_acc: 0.9812
Epoch 18/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.0
072 - acc: 0.9978 - val_loss: 0.0913 - val_acc: 0.9780
Epoch 19/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.0
069 - acc: 0.9976 - val_loss: 0.0923 - val_acc: 0.9803
Epoch 20/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.0
072 - acc: 0.9977 - val_loss: 0.0831 - val_acc: 0.9795
```

In [67]:

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

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

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

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

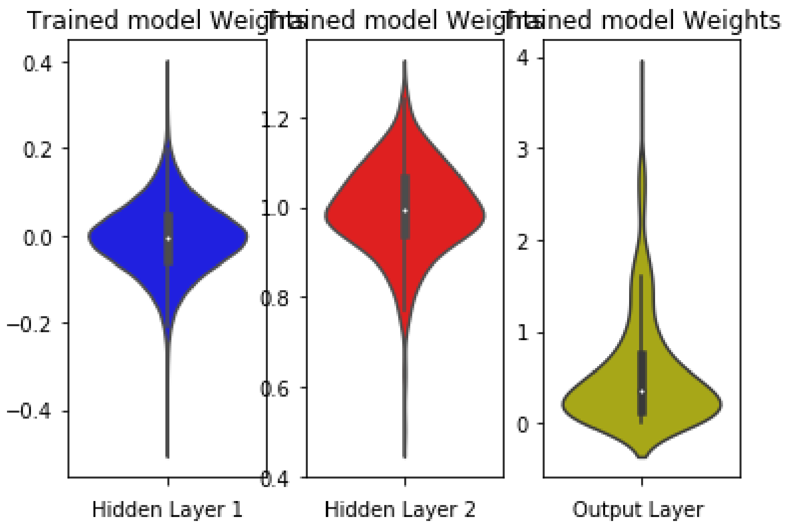
```
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epo
chs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

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

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

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

```
Test score: 0.08313359864633312
Test accuracy: 0.9795
```



In [68]:
```
w_after = model_batch.get_weights()
```

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


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

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

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



(c) MLP + Dropout + AdamOptimizer +Relu+BatchNormalization

In [69]:

```
# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormalization-function-in-keras

from keras.layers import Dropout

model_drop = Sequential()

model_drop.add(Dense(396, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.041, seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(198, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.58, seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

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


model_drop.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_50 (Dense)             (None, 396)               310860
_________________________________________________________________
batch_normalization_11 (Batc (None, 396)               1584
_________________________________________________________________
dropout_11 (Dropout)         (None, 396)               0
_________________________________________________________________
dense_51 (Dense)             (None, 198)               78606
_________________________________________________________________
batch_normalization_12 (Batc (None, 198)               792
_________________________________________________________________
dropout_12 (Dropout)         (None, 198)               0
```

```
_________________________________________________________________
dense_52 (Dense)             (None, 10)                1990
=================================================================
Total params: 393,832
Trainable params: 392,644
Non-trainable params: 1,188
_________________________________________________________________
```

In [70]:
```
model_drop.compile(optimizer='adam', loss='categorical_crossentropy', m
etrics=['accuracy'])

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.5
034 - acc: 0.8470 - val_loss: 0.1727 - val_acc: 0.9459
Epoch 2/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.2
593 - acc: 0.9214 - val_loss: 0.1276 - val_acc: 0.9596
Epoch 3/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.2
024 - acc: 0.9383 - val_loss: 0.1144 - val_acc: 0.9643
Epoch 4/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.1
763 - acc: 0.9468 - val_loss: 0.1017 - val_acc: 0.9673
Epoch 5/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.1
585 - acc: 0.9516 - val_loss: 0.0922 - val_acc: 0.9711
Epoch 6/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.1
445 - acc: 0.9550 - val_loss: 0.0894 - val_acc: 0.9727
Epoch 7/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.1
318 - acc: 0.9594 - val_loss: 0.0817 - val_acc: 0.9752
Epoch 8/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.1
251 - acc: 0.9614 - val_loss: 0.0756 - val_acc: 0.9761
```

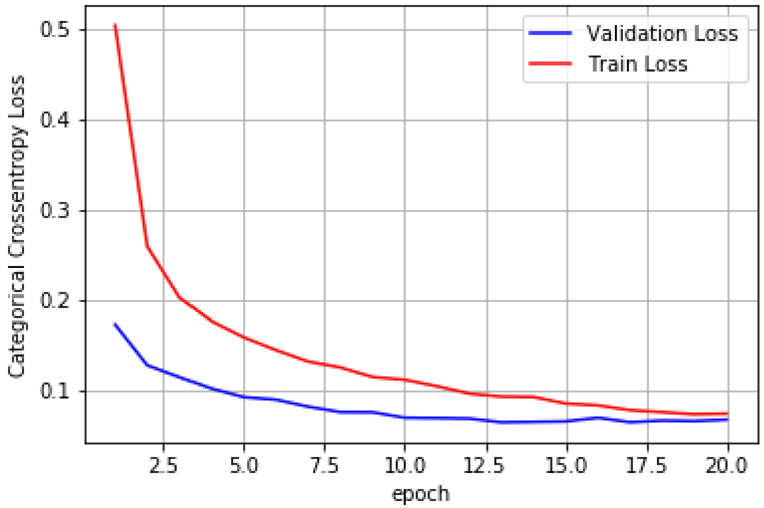
```
Epoch 9/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.1
146 - acc: 0.9651 - val_loss: 0.0754 - val_acc: 0.9774
Epoch 10/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.1
115 - acc: 0.9658 - val_loss: 0.0694 - val_acc: 0.9788
Epoch 11/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.1
044 - acc: 0.9675 - val_loss: 0.0690 - val_acc: 0.9788
Epoch 12/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0
961 - acc: 0.9699 - val_loss: 0.0685 - val_acc: 0.9793
Epoch 13/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0
929 - acc: 0.9713 - val_loss: 0.0643 - val_acc: 0.9790
Epoch 14/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0
924 - acc: 0.9715 - val_loss: 0.0648 - val_acc: 0.9796
Epoch 15/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0
852 - acc: 0.9737 - val_loss: 0.0653 - val_acc: 0.9798
Epoch 16/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0
831 - acc: 0.9737 - val_loss: 0.0694 - val_acc: 0.9782
Epoch 17/20
60000/60000 [==============================] - 3s 58us/step - loss: 0.0
778 - acc: 0.9756 - val_loss: 0.0643 - val_acc: 0.9805
Epoch 18/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.0
756 - acc: 0.9761 - val_loss: 0.0662 - val_acc: 0.9809
Epoch 19/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.0
732 - acc: 0.9764 - val_loss: 0.0657 - val_acc: 0.9815
Epoch 20/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0
739 - acc: 0.9766 - val_loss: 0.0672 - val_acc: 0.9810
```

In [71]:

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

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

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

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

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

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

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

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

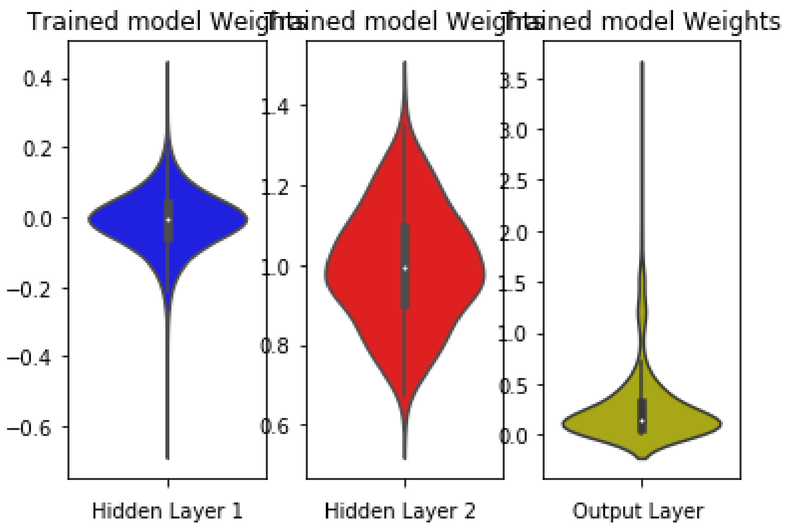
```
Test score: 0.06723630262503284
Test accuracy: 0.981
```



In [72]:

```
w_after = model_drop.get_weights()

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


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

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

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

```
plt.xlabel('Output Layer ')
plt.show()
```



(d)MLP + Dropout + AdamOptimizer +Relu

In [73]:
```
# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormalization-function-in-keras

from keras.layers import Dropout

model_drop = Sequential()

model_drop.add(Dense(396, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.071, seed=None)))
model_drop.add(Dropout(0.5))

model_drop.add(Dense(198, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.1, seed=None)) )
model_drop.add(Dropout(0.5))

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

```
model_drop.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_53 (Dense)             (None, 396)               310860    
_________________________________________________________________
dropout_13 (Dropout)         (None, 396)               0         
_________________________________________________________________
dense_54 (Dense)             (None, 198)               78606     
_________________________________________________________________
dropout_14 (Dropout)         (None, 198)               0         
_________________________________________________________________
dense_55 (Dense)             (None, 10)                1990      
=================================================================
Total params: 391,456
Trainable params: 391,456
Non-trainable params: 0
_________________________________________________________________
```

In [74]:
```
model_drop.compile(optimizer='adam', loss='categorical_crossentropy', m
etrics=['accuracy'])

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 4s 59us/step - loss: 0.5
392 - acc: 0.8326 - val_loss: 0.1643 - val_acc: 0.9499
Epoch 2/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.2
342 - acc: 0.9297 - val_loss: 0.1232 - val_acc: 0.9626
Epoch 3/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.1
814 - acc: 0.9465 - val_loss: 0.1045 - val_acc: 0.9662
Epoch 4/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.1
```

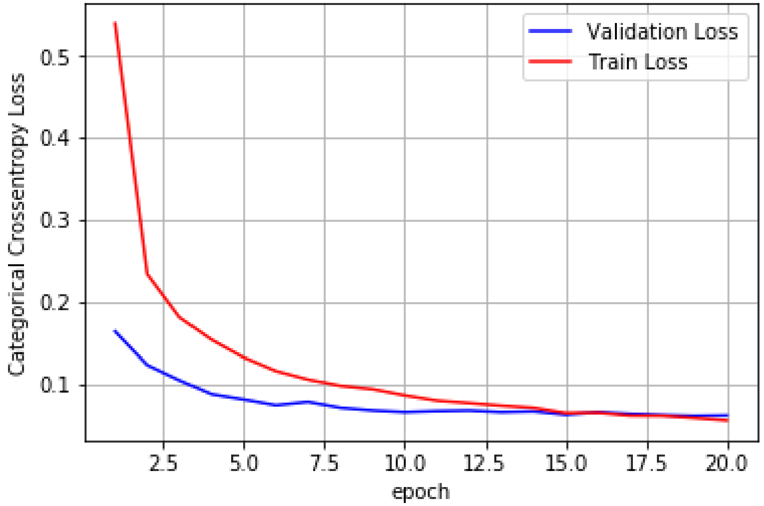
```
540 - acc: 0.9553 - val_loss: 0.0877 - val_acc: 0.9716
Epoch 5/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.1
323 - acc: 0.9598 - val_loss: 0.0812 - val_acc: 0.9745
Epoch 6/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.1
158 - acc: 0.9652 - val_loss: 0.0742 - val_acc: 0.9749
Epoch 7/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.1
055 - acc: 0.9674 - val_loss: 0.0783 - val_acc: 0.9762
Epoch 8/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.0
979 - acc: 0.9706 - val_loss: 0.0711 - val_acc: 0.9789
Epoch 9/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.0
938 - acc: 0.9718 - val_loss: 0.0678 - val_acc: 0.9792
Epoch 10/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.0
863 - acc: 0.9734 - val_loss: 0.0658 - val_acc: 0.9798
Epoch 11/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.0
799 - acc: 0.9755 - val_loss: 0.0671 - val_acc: 0.9811
Epoch 12/20
60000/60000 [==============================] - 2s 37us/step - loss: 0.0
768 - acc: 0.9760 - val_loss: 0.0677 - val_acc: 0.9818
Epoch 13/20
60000/60000 [==============================] - 2s 38us/step - loss: 0.0
737 - acc: 0.9774 - val_loss: 0.0658 - val_acc: 0.9814
Epoch 14/20
60000/60000 [==============================] - 2s 38us/step - loss: 0.0
711 - acc: 0.9783 - val_loss: 0.0667 - val_acc: 0.9813
Epoch 15/20
60000/60000 [==============================] - 2s 38us/step - loss: 0.0
647 - acc: 0.9798 - val_loss: 0.0627 - val_acc: 0.9833
Epoch 16/20
60000/60000 [==============================] - 2s 37us/step - loss: 0.0
650 - acc: 0.9802 - val_loss: 0.0656 - val_acc: 0.9818
Epoch 17/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.0
```

```
615 - acc: 0.9807 - val_loss: 0.0637 - val_acc: 0.9827
Epoch 18/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.0
613 - acc: 0.9811 - val_loss: 0.0621 - val_acc: 0.9826
Epoch 19/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.0
587 - acc: 0.9817 - val_loss: 0.0612 - val_acc: 0.9824
Epoch 20/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.0
557 - acc: 0.9820 - val_loss: 0.0617 - val_acc: 0.9822
```

In [75]:

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

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

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

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

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

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

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

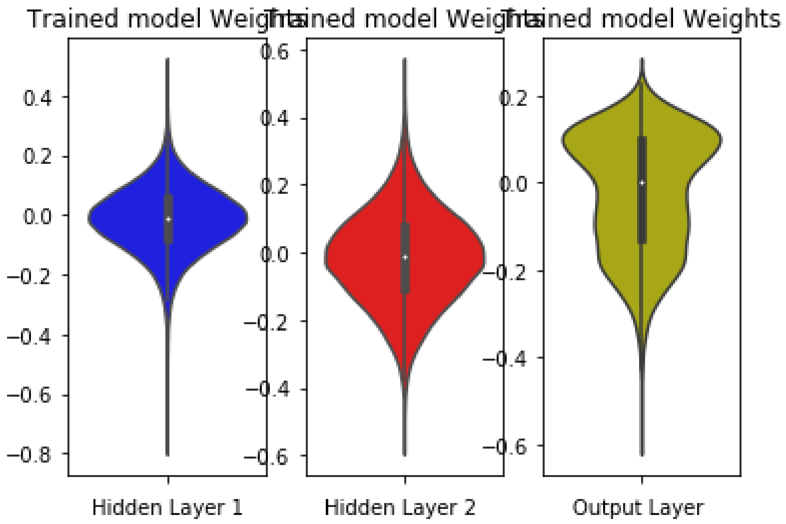
```
Test score: 0.06173285202749103
Test accuracy: 0.9822
```



In [76]:

```
w_after = model_drop.get_weights()

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


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

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

plt.subplot(1, 3, 3)
```

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



2.Using hidden three layers

(a) MLP + ReLU + ADAM

In [99]:
```
model_relu = Sequential()
model_relu.add(Dense(392, activation='relu', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.071, seed=None)))
model_relu.add(Dense(196, activation='relu', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.1, seed=None)) )
model_relu.add(Dense(98, activation='relu', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.14, seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())
```

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

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_88 (Dense)             (None, 392)               307720
_________________________________________________________________
dense_89 (Dense)             (None, 196)               77028
_________________________________________________________________
dense_90 (Dense)             (None, 98)                19306
_________________________________________________________________
dense_91 (Dense)             (None, 10)                990
=================================================================
Total params: 405,044
Trainable params: 405,044
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.2
327 - acc: 0.9298 - val_loss: 0.1094 - val_acc: 0.9663
Epoch 2/20
60000/60000 [==============================] - 2s 40us/step - loss: 0.0
863 - acc: 0.9734 - val_loss: 0.0958 - val_acc: 0.9712
Epoch 3/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0
572 - acc: 0.9818 - val_loss: 0.0762 - val_acc: 0.9771
Epoch 4/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0
389 - acc: 0.9874 - val_loss: 0.0765 - val_acc: 0.9775
Epoch 5/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0
280 - acc: 0.9909 - val_loss: 0.0714 - val_acc: 0.9789
Epoch 6/20
```

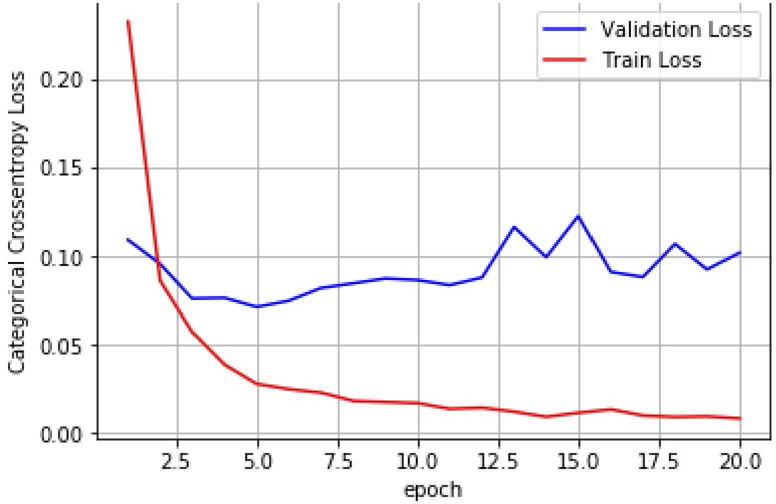
```
60000/60000 [==============================] - 2s 36us/step - loss: 0.0
250 - acc: 0.9918 - val_loss: 0.0748 - val_acc: 0.9788
Epoch 7/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0
231 - acc: 0.9922 - val_loss: 0.0820 - val_acc: 0.9775
Epoch 8/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0
183 - acc: 0.9941 - val_loss: 0.0848 - val_acc: 0.9784
Epoch 9/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0
177 - acc: 0.9942 - val_loss: 0.0875 - val_acc: 0.9801
Epoch 10/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0
171 - acc: 0.9946 - val_loss: 0.0866 - val_acc: 0.9775
Epoch 11/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0
139 - acc: 0.9952 - val_loss: 0.0837 - val_acc: 0.9803
Epoch 12/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0
145 - acc: 0.9956 - val_loss: 0.0880 - val_acc: 0.9797
Epoch 13/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0
122 - acc: 0.9959 - val_loss: 0.1167 - val_acc: 0.9753
Epoch 14/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0
094 - acc: 0.9969 - val_loss: 0.0995 - val_acc: 0.9785
Epoch 15/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0
116 - acc: 0.9963 - val_loss: 0.1228 - val_acc: 0.9756
Epoch 16/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0
135 - acc: 0.9958 - val_loss: 0.0912 - val_acc: 0.9795
Epoch 17/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0
101 - acc: 0.9967 - val_loss: 0.0883 - val_acc: 0.9801
Epoch 18/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0
093 - acc: 0.9970 - val_loss: 0.1071 - val_acc: 0.9766
Epoch 19/20
```

```
60000/60000 [==============================] - 2s 36us/step - loss: 0.0
096 - acc: 0.9968 - val_loss: 0.0926 - val_acc: 0.9809
Epoch 20/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0
084 - acc: 0.9972 - val_loss: 0.1020 - val_acc: 0.9810
```

In [100]:

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

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

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

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

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

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


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

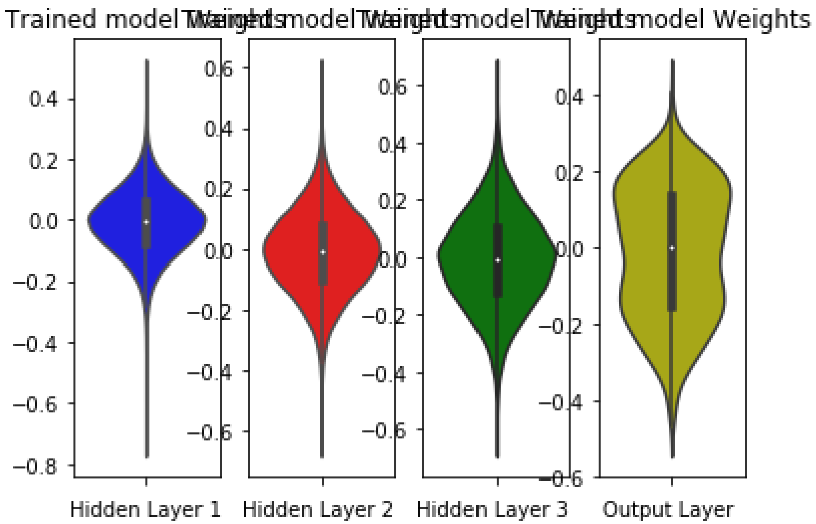
```
Test score: 0.10204988605169434
Test accuracy: 0.981
```



In [107]:

```
w_after = model_relu.get_weights()

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


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

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

plt.subplot(1, 4, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
```

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

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



(b) MLP + Batch-Norm on hidden Layers + AdamOptimizer +RELU

In [144]:

```
# Multilayer perceptron

# https://intoli.com/blog/neural-network-initialization/
# If we sample weights from a normal distribution N(0,σ) we satisfy thi
s condition with σ=√(2/(ni+ni+1).
# h1 =>  σ=√(2/(ni+ni+1) = 0.041  => N(0,σ) = N(0,0.041)
# h2 =>  σ=√(2/(ni+ni+1) = 0.058  => N(0,σ) = N(0,0.058)
# out =>  σ=√(2/(ni+ni+1) = 0.098  => N(0,σ) = N(0,0.98)

from keras.layers.normalization import BatchNormalization
```

```
model_batch = Sequential()

model_batch.add(Dense(396, activation='relu', input_shape=(input_dim,),
 kernel_initializer=RandomNormal(mean=0.0, stddev=0.041, seed=None)))
model_batch.add(BatchNormalization())

model_batch.add(Dense(198, activation='relu', kernel_initializer=Random
Normal(mean=0.0, stddev=0.58, seed=None)) )
model_batch.add(BatchNormalization())

model_batch.add(Dense(98, activation='relu', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.08, seed=None)) )
model_batch.add(BatchNormalization())


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


model_batch.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_124 (Dense)            (None, 396)               310860    
_________________________________________________________________
batch_normalization_34 (Batc (None, 396)               1584      
_________________________________________________________________
dense_125 (Dense)            (None, 198)               78606     
_________________________________________________________________
batch_normalization_35 (Batc (None, 198)               792       
_________________________________________________________________
dense_126 (Dense)            (None, 98)                19502     
_________________________________________________________________
batch_normalization_36 (Batc (None, 98)                392       
_________________________________________________________________
dense_127 (Dense)            (None, 10)                990       
=================================================================
Total params: 412,726
Trainable params: 411,342
```

```
Non-trainable params: 1,384
_________________________________________________________________
```

In [145]:
```
model_batch.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.
2091 - acc: 0.9375 - val_loss: 0.1060 - val_acc: 0.9699
Epoch 2/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0
775 - acc: 0.9766 - val_loss: 0.0932 - val_acc: 0.9693
Epoch 3/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0
520 - acc: 0.9835 - val_loss: 0.0760 - val_acc: 0.9758
Epoch 4/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.0
370 - acc: 0.9884 - val_loss: 0.0805 - val_acc: 0.9743
Epoch 5/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0
315 - acc: 0.9896 - val_loss: 0.0844 - val_acc: 0.9744
Epoch 6/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0
259 - acc: 0.9915 - val_loss: 0.0762 - val_acc: 0.9775
Epoch 7/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0
220 - acc: 0.9926 - val_loss: 0.0658 - val_acc: 0.9796
Epoch 8/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0
174 - acc: 0.9941 - val_loss: 0.0707 - val_acc: 0.9784
Epoch 9/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0
155 - acc: 0.9951 - val_loss: 0.0702 - val_acc: 0.9791
Epoch 10/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0
```

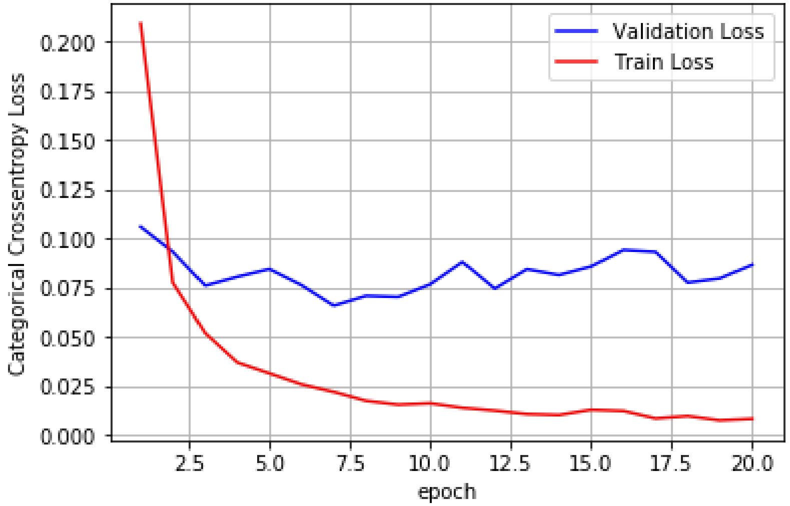
```
162 - acc: 0.9948 - val_loss: 0.0767 - val_acc: 0.9786
Epoch 11/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0
139 - acc: 0.9952 - val_loss: 0.0880 - val_acc: 0.9756
Epoch 12/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0
125 - acc: 0.9956 - val_loss: 0.0745 - val_acc: 0.9803
Epoch 13/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0
107 - acc: 0.9967 - val_loss: 0.0843 - val_acc: 0.9785
Epoch 14/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0
102 - acc: 0.9968 - val_loss: 0.0814 - val_acc: 0.9782
Epoch 15/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0
129 - acc: 0.9956 - val_loss: 0.0857 - val_acc: 0.9772
Epoch 16/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0
123 - acc: 0.9960 - val_loss: 0.0942 - val_acc: 0.9764
Epoch 17/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0
085 - acc: 0.9972 - val_loss: 0.0931 - val_acc: 0.9768
Epoch 18/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0
097 - acc: 0.9967 - val_loss: 0.0775 - val_acc: 0.9814
Epoch 19/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0
075 - acc: 0.9977 - val_loss: 0.0797 - val_acc: 0.9811
Epoch 20/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0
083 - acc: 0.9972 - val_loss: 0.0866 - val_acc: 0.9790
```

In [146]:

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

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

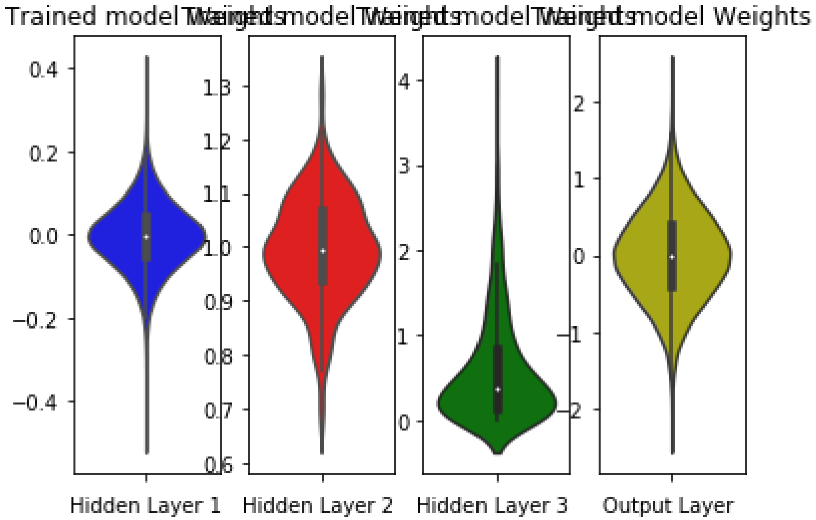
```
# list of epoch numbers
x = list(range(1,nb_epoch+1))

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

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

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

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

```
Test score: 0.08658756976571458
Test accuracy: 0.979
```



In [147]:
```
w_after = model_batch.get_weights()

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


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

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

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

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

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



(c)MLP + Dropout + AdamOptimizer +Relu +batch

In [148]:
```
model_drop = Sequential()

model_drop.add(Dense(396, activation='relu', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.041, seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(198, activation='relu', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.58, seed=None)) )
model_drop.add(BatchNormalization())
```

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

model_drop.add(Dense(198, activation='relu', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.08, seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

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


model_drop.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_128 (Dense)            (None, 396)               310860    
_________________________________________________________________
batch_normalization_37 (Batc (None, 396)               1584      
_________________________________________________________________
dropout_29 (Dropout)         (None, 396)               0         
_________________________________________________________________
dense_129 (Dense)            (None, 198)               78606     
_________________________________________________________________
batch_normalization_38 (Batc (None, 198)               792       
_________________________________________________________________
dropout_30 (Dropout)         (None, 198)               0         
_________________________________________________________________
dense_130 (Dense)            (None, 198)               39402     
_________________________________________________________________
batch_normalization_39 (Batc (None, 198)               792       
_________________________________________________________________
dropout_31 (Dropout)         (None, 198)               0         
_________________________________________________________________
dense_131 (Dense)            (None, 10)                1990      
=================================================================
Total params: 434,026
Trainable params: 432,442
Non-trainable params: 1,584
_________________________________________________________________
```

In [149]:

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

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 9s 154us/step - loss: 0.
6811 - acc: 0.7915 - val_loss: 0.1994 - val_acc: 0.9372
Epoch 2/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.3
135 - acc: 0.9063 - val_loss: 0.1488 - val_acc: 0.9541
Epoch 3/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.2
522 - acc: 0.9262 - val_loss: 0.1256 - val_acc: 0.9613
Epoch 4/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.2
187 - acc: 0.9357 - val_loss: 0.1158 - val_acc: 0.9633
Epoch 5/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.1
870 - acc: 0.9444 - val_loss: 0.1006 - val_acc: 0.9702
Epoch 6/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.1
704 - acc: 0.9490 - val_loss: 0.0914 - val_acc: 0.9721
Epoch 7/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.1
600 - acc: 0.9520 - val_loss: 0.0905 - val_acc: 0.9715
Epoch 8/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.1
457 - acc: 0.9568 - val_loss: 0.0898 - val_acc: 0.9740
Epoch 9/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.1
405 - acc: 0.9584 - val_loss: 0.0845 - val_acc: 0.9748
Epoch 10/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.1
341 - acc: 0.9605 - val_loss: 0.0814 - val_acc: 0.9757
Epoch 11/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.1
254 - acc: 0.9626 - val_loss: 0.0740 - val_acc: 0.9782
```

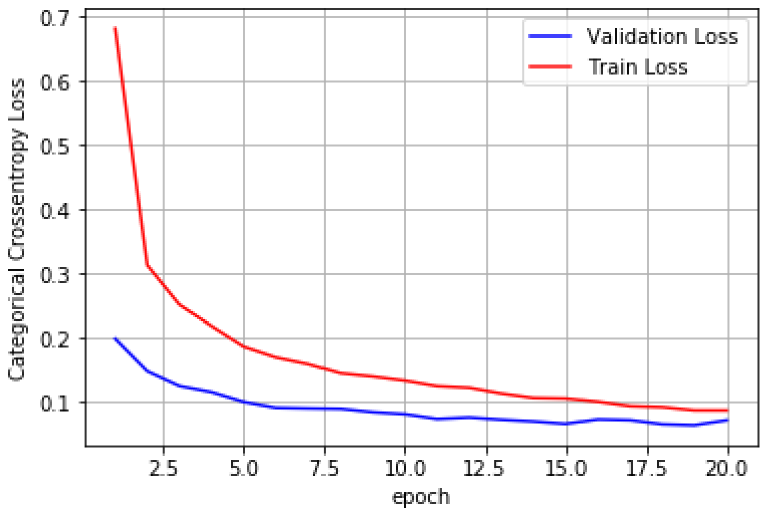
```
Epoch 12/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.1
228 - acc: 0.9636 - val_loss: 0.0763 - val_acc: 0.9777
Epoch 13/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.1
138 - acc: 0.9651 - val_loss: 0.0729 - val_acc: 0.9785
Epoch 14/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.1
068 - acc: 0.9680 - val_loss: 0.0701 - val_acc: 0.9804
Epoch 15/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.1
060 - acc: 0.9682 - val_loss: 0.0664 - val_acc: 0.9812
Epoch 16/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.1
009 - acc: 0.9700 - val_loss: 0.0733 - val_acc: 0.9796
Epoch 17/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0
940 - acc: 0.9703 - val_loss: 0.0721 - val_acc: 0.9798
Epoch 18/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.0
925 - acc: 0.9719 - val_loss: 0.0657 - val_acc: 0.9808
Epoch 19/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0
875 - acc: 0.9735 - val_loss: 0.0643 - val_acc: 0.9813
Epoch 20/20
60000/60000 [==============================] - 5s 76us/step - loss: 0.0
874 - acc: 0.9731 - val_loss: 0.0720 - val_acc: 0.9800
```

In [150]:

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

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

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

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

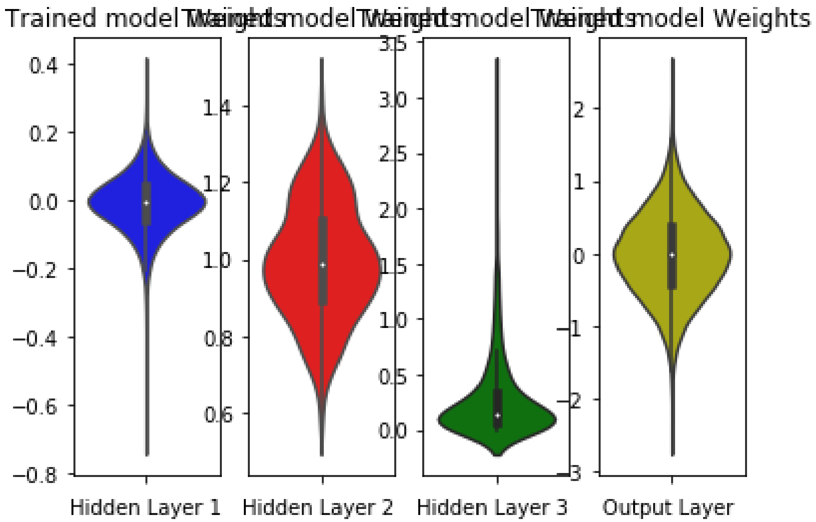
```
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epo
chs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

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

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

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

```
Test score: 0.07200911457028705
Test accuracy: 0.98
```



In [151]:
```
w_after = model_drop.get_weights()
```

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


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

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

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

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



(d) MLP + Dropout + AdamOptimizer +Relu

In [114]:
```
# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batc
hnormalization-function-in-keras

from keras.layers import Dropout

model_drop = Sequential()

model_drop.add(Dense(396, activation='relu', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.071, seed=None)))
model_drop.add(Dropout(0.5))

model_drop.add(Dense(198, activation='relu', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.1, seed=None)) )
model_drop.add(Dropout(0.5))

model_drop.add(Dense(98, activation='relu', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.14, seed=None)) )
model_drop.add(Dropout(0.5))
```

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

model_drop.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_96 (Dense)             (None, 396)               310860    
_________________________________________________________________
dropout_21 (Dropout)         (None, 396)               0         
_________________________________________________________________
dense_97 (Dense)             (None, 198)               78606     
_________________________________________________________________
dropout_22 (Dropout)         (None, 198)               0         
_________________________________________________________________
dense_98 (Dense)             (None, 98)                19502     
_________________________________________________________________
dropout_23 (Dropout)         (None, 98)                0         
_________________________________________________________________
dense_99 (Dense)             (None, 10)                990       
=================================================================
Total params: 409,958
Trainable params: 409,958
Non-trainable params: 0
_________________________________________________________________
```

In [115]:

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

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.9453 - acc: 0.6996 - val_loss: 0.2284 - val_acc: 0.9359
Epoch 2/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.3
```

```
629 - acc: 0.8999 - val_loss: 0.1711 - val_acc: 0.9529
Epoch 3/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.2
708 - acc: 0.9277 - val_loss: 0.1486 - val_acc: 0.9585
Epoch 4/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.2
256 - acc: 0.9383 - val_loss: 0.1244 - val_acc: 0.9637
Epoch 5/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.1
934 - acc: 0.9485 - val_loss: 0.1097 - val_acc: 0.9702
Epoch 6/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.1
753 - acc: 0.9518 - val_loss: 0.1028 - val_acc: 0.9732
Epoch 7/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.1
570 - acc: 0.9579 - val_loss: 0.0989 - val_acc: 0.9739
Epoch 8/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.1
467 - acc: 0.9608 - val_loss: 0.0924 - val_acc: 0.9750
Epoch 9/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.1
378 - acc: 0.9621 - val_loss: 0.0874 - val_acc: 0.9752
Epoch 10/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.1
238 - acc: 0.9666 - val_loss: 0.0862 - val_acc: 0.9773
Epoch 11/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.1
163 - acc: 0.9675 - val_loss: 0.0867 - val_acc: 0.9755
Epoch 12/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.1
129 - acc: 0.9689 - val_loss: 0.0824 - val_acc: 0.9792
Epoch 13/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.1
062 - acc: 0.9712 - val_loss: 0.0858 - val_acc: 0.9789
Epoch 14/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.1
023 - acc: 0.9722 - val_loss: 0.0827 - val_acc: 0.9775
Epoch 15/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.0
```

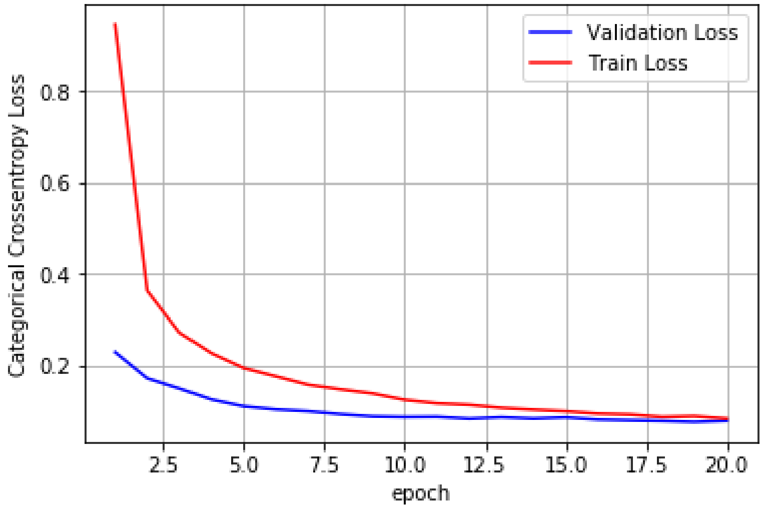
```
988 - acc: 0.9732 - val_loss: 0.0854 - val_acc: 0.9784
Epoch 16/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.0
936 - acc: 0.9735 - val_loss: 0.0804 - val_acc: 0.9808
Epoch 17/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.0
918 - acc: 0.9744 - val_loss: 0.0789 - val_acc: 0.9785
Epoch 18/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.0
863 - acc: 0.9751 - val_loss: 0.0776 - val_acc: 0.9794
Epoch 19/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.0
880 - acc: 0.9755 - val_loss: 0.0754 - val_acc: 0.9805
Epoch 20/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.0
827 - acc: 0.9774 - val_loss: 0.0780 - val_acc: 0.9806
```

In [116]:

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

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

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

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

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

# loss : training loss
# acc : train accuracy
```

```
# for each key in histrory.histrory we will have a list of length equal
 to number of epochs

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

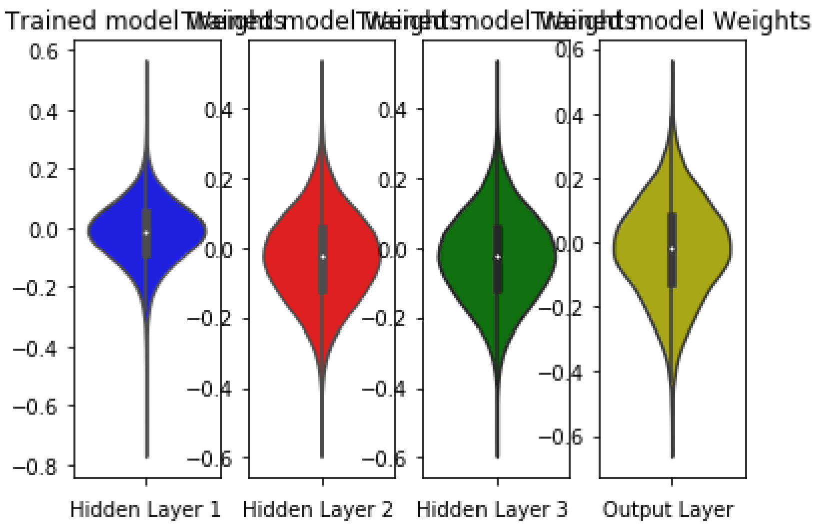
```
Test score: 0.0779710623358118
Test accuracy: 0.9806
```



In [118]:

```
w_after = model_drop.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[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, 4, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
```

```
plt.xlabel('Hidden Layer 1')

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


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

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



3.using five layers

(a)MLP + ReLU + ADAM

In [119]:
```
model_relu = Sequential()
model_relu.add(Dense(392, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.071, seed=None)))
model_relu.add(Dense(196, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.1, seed=None)) )
model_relu.add(Dense(98, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.14, seed=None)) )
model_relu.add(Dense(49, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.2, seed=None)) )
model_relu.add(Dense(25, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.28, seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_100 (Dense)            (None, 392)               307720    
_________________________________________________________________
dense_101 (Dense)            (None, 196)               77028     
_________________________________________________________________
dense_102 (Dense)            (None, 98)                19306     
_________________________________________________________________
dense_103 (Dense)            (None, 49)                4851      
_________________________________________________________________
dense_104 (Dense)            (None, 25)                1250      
_________________________________________________________________
dense_105 (Dense)            (None, 10)                260       
=================================================================
Total params: 410,415
```

```
Trainable params: 410,415
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.3
188 - acc: 0.9015 - val_loss: 0.1328 - val_acc: 0.9596
Epoch 2/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.1
040 - acc: 0.9690 - val_loss: 0.1062 - val_acc: 0.9692
Epoch 3/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.0
664 - acc: 0.9793 - val_loss: 0.0807 - val_acc: 0.9741
Epoch 4/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.0
523 - acc: 0.9839 - val_loss: 0.0929 - val_acc: 0.9722
Epoch 5/20
60000/60000 [==============================] - 2s 40us/step - loss: 0.0
410 - acc: 0.9868 - val_loss: 0.0820 - val_acc: 0.9759
Epoch 6/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.0
312 - acc: 0.9901 - val_loss: 0.0800 - val_acc: 0.9764
Epoch 7/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.0
303 - acc: 0.9903 - val_loss: 0.0887 - val_acc: 0.9743
Epoch 8/20
60000/60000 [==============================] - 2s 40us/step - loss: 0.0
273 - acc: 0.9907 - val_loss: 0.1054 - val_acc: 0.9743
Epoch 9/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.0
218 - acc: 0.9929 - val_loss: 0.0935 - val_acc: 0.9781
Epoch 10/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.0
259 - acc: 0.9915 - val_loss: 0.0956 - val_acc: 0.9767
Epoch 11/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.0
208 - acc: 0.9932 - val_loss: 0.0886 - val_acc: 0.9782
Epoch 12/20
```

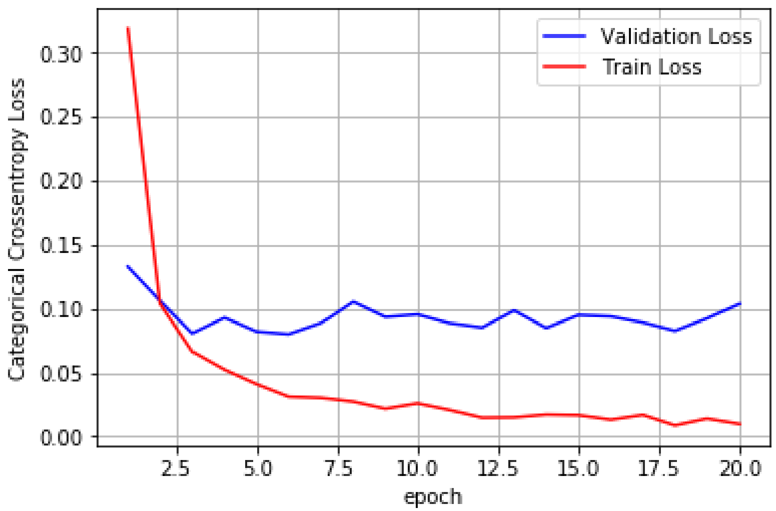
```
60000/60000 [==============================] - 2s 40us/step - loss: 0.0
147 - acc: 0.9954 - val_loss: 0.0851 - val_acc: 0.9807
Epoch 13/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.0
150 - acc: 0.9953 - val_loss: 0.0986 - val_acc: 0.9753
Epoch 14/20
60000/60000 [==============================] - 2s 40us/step - loss: 0.0
172 - acc: 0.9944 - val_loss: 0.0848 - val_acc: 0.9810
Epoch 15/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.0
167 - acc: 0.9950 - val_loss: 0.0950 - val_acc: 0.9782
Epoch 16/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.0
132 - acc: 0.9956 - val_loss: 0.0940 - val_acc: 0.9808
Epoch 17/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.0
169 - acc: 0.9945 - val_loss: 0.0892 - val_acc: 0.9796
Epoch 18/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.0
087 - acc: 0.9972 - val_loss: 0.0826 - val_acc: 0.9799
Epoch 19/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.0
140 - acc: 0.9958 - val_loss: 0.0925 - val_acc: 0.9818
Epoch 20/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.0
098 - acc: 0.9968 - val_loss: 0.1038 - val_acc: 0.9776
```

In [120]:

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

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

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

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```

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

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

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


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

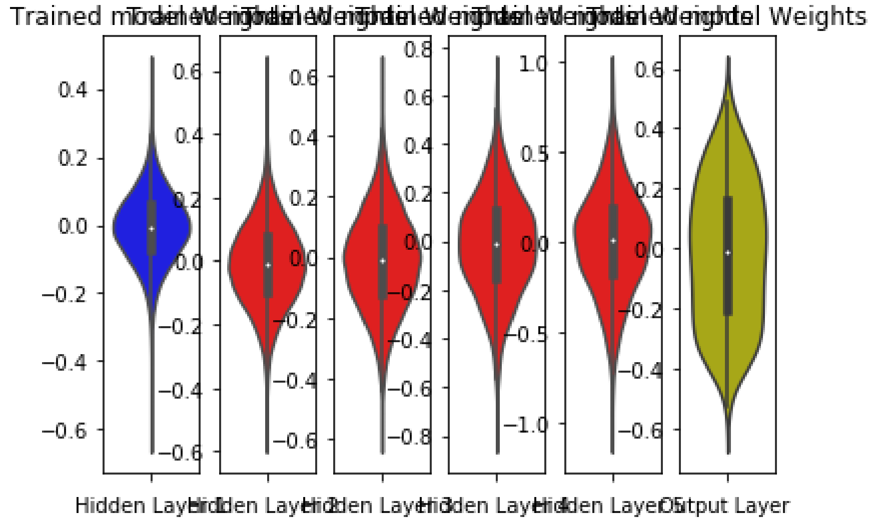
```
Test score: 0.10375401149833269
Test accuracy: 0.9776
```



In [127]:
```
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("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

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

plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='r')
plt.xlabel('Hidden Layer 3 ')

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

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

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

```
plt.xlabel('Output Layer ')
plt.show()
```



(b)MLP + Batch-Norm on hidden Layers + AdamOptimizer +RELU

In [136]:
```
# Multilayer perceptron

from keras.layers.normalization import BatchNormalization

model_batch = Sequential()

model_batch.add(Dense(392, activation='relu', input_shape=(input_dim,),
 kernel_initializer=RandomNormal(mean=0.0, stddev=0.041, seed=None)))
model_batch.add(BatchNormalization())

model_batch.add(Dense(196, activation='relu', kernel_initializer=Random
Normal(mean=0.0, stddev=0.58, seed=None)) )
model_batch.add(BatchNormalization())

model_batch.add(Dense(98, activation='relu', kernel_initializer=RandomN
```

```
ormal(mean=0.0, stddev=0.08, seed=None)) )
model_batch.add(BatchNormalization())

model_batch.add(Dense(49, activation='relu', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.11, seed=None)) )
model_batch.add(BatchNormalization())

model_batch.add(Dense(25, activation='relu', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.16, seed=None)) )
model_batch.add(BatchNormalization())


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


model_batch.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_112 (Dense)            (None, 392)               307720    
_________________________________________________________________
batch_normalization_29 (Batc (None, 392)               1568      
_________________________________________________________________
dense_113 (Dense)            (None, 196)               77028     
_________________________________________________________________
batch_normalization_30 (Batc (None, 196)               784       
_________________________________________________________________
dense_114 (Dense)            (None, 98)                19306     
_________________________________________________________________
batch_normalization_31 (Batc (None, 98)                392       
_________________________________________________________________
dense_115 (Dense)            (None, 49)                4851      
_________________________________________________________________
batch_normalization_32 (Batc (None, 49)                196       
_________________________________________________________________
dense_116 (Dense)            (None, 25)                1250      
_________________________________________________________________
batch_normalization_33 (Batc (None, 25)                100       
_________________________________________________________________
```

```
dense_117 (Dense)            (None, 10)                260
=================================================================
Total params: 413,455
Trainable params: 411,935
Non-trainable params: 1,520
_________________________________________________________________
```

In [137]:
```
model_batch.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 9s 152us/step - loss: 0.
2931 - acc: 0.9211 - val_loss: 0.1364 - val_acc: 0.9580
Epoch 2/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.1
012 - acc: 0.9707 - val_loss: 0.1037 - val_acc: 0.9693
Epoch 3/20
60000/60000 [==============================] - 6s 101us/step - loss: 0.
0665 - acc: 0.9798 - val_loss: 0.0901 - val_acc: 0.9724
Epoch 4/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.0
539 - acc: 0.9829 - val_loss: 0.0897 - val_acc: 0.9719
Epoch 5/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0
414 - acc: 0.9867 - val_loss: 0.0849 - val_acc: 0.9738
Epoch 6/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0
372 - acc: 0.9885 - val_loss: 0.0930 - val_acc: 0.9743
Epoch 7/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0
307 - acc: 0.9904 - val_loss: 0.0927 - val_acc: 0.9748
Epoch 8/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0
281 - acc: 0.9910 - val_loss: 0.1120 - val_acc: 0.9705
Epoch 9/20
```

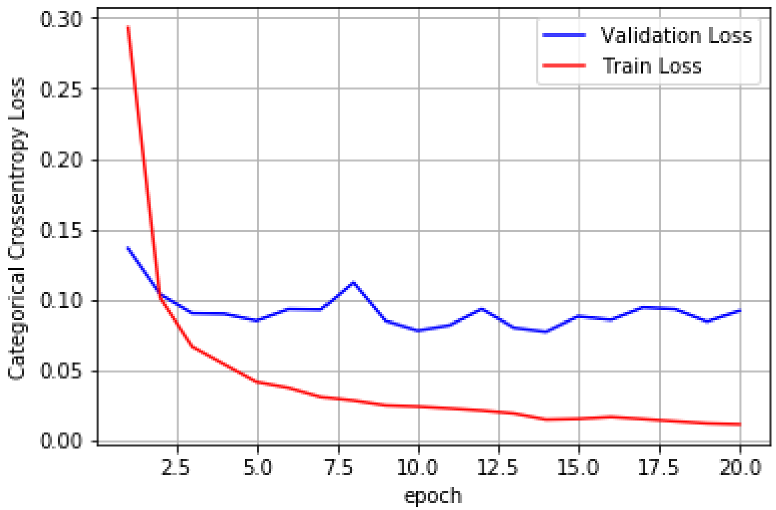
```
60000/60000 [==============================] - 5s 89us/step - loss: 0.0
247 - acc: 0.9921 - val_loss: 0.0848 - val_acc: 0.9759
Epoch 10/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0
239 - acc: 0.9921 - val_loss: 0.0779 - val_acc: 0.9785
Epoch 11/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0
225 - acc: 0.9926 - val_loss: 0.0817 - val_acc: 0.9779
Epoch 12/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0
210 - acc: 0.9933 - val_loss: 0.0932 - val_acc: 0.9746
Epoch 13/20
60000/60000 [==============================] - 6s 101us/step - loss: 0.
0189 - acc: 0.9939 - val_loss: 0.0800 - val_acc: 0.9801
Epoch 14/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0
146 - acc: 0.9952 - val_loss: 0.0771 - val_acc: 0.9811
Epoch 15/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0
151 - acc: 0.9952 - val_loss: 0.0880 - val_acc: 0.9786
Epoch 16/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0
164 - acc: 0.9945 - val_loss: 0.0855 - val_acc: 0.9774
Epoch 17/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0
149 - acc: 0.9950 - val_loss: 0.0943 - val_acc: 0.9775
Epoch 18/20
60000/60000 [==============================] - 6s 101us/step - loss: 0.
0133 - acc: 0.9956 - val_loss: 0.0930 - val_acc: 0.9783
Epoch 19/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0
118 - acc: 0.9963 - val_loss: 0.0843 - val_acc: 0.9799
Epoch 20/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0
112 - acc: 0.9961 - val_loss: 0.0919 - val_acc: 0.9799
```

In [138]:

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

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

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

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

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

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

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

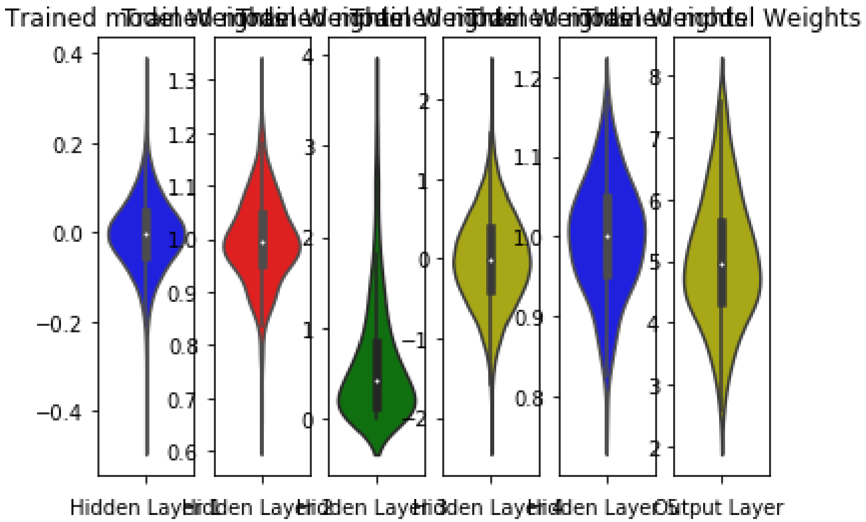
```
Test score: 0.0918853635030333
Test accuracy: 0.9799
```



In [139]:

```
w_after = model_batch.get_weights()

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


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

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

```
plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3 ')

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

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

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



(c)MLP + Dropout + AdamOptimizer +Relu +batch

In [152]:
```
model_drop = Sequential()

model_drop.add(Dense(396, activation='relu', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.041, seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(198, activation='relu', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.58, seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(198, activation='relu', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.08, seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(198, activation='relu', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.11, seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(198, activation='relu', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.16, seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

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


model_drop.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_132 (Dense)            (None, 396)               310860
_________________________________________________________________
```

```
batch_normalization_40 (Batc (None, 396)              1584
_________________________________________________________________
dropout_32 (Dropout)         (None, 396)              0
_________________________________________________________________
dense_133 (Dense)            (None, 198)              78606
_________________________________________________________________
batch_normalization_41 (Batc (None, 198)              792
_________________________________________________________________
dropout_33 (Dropout)         (None, 198)              0
_________________________________________________________________
dense_134 (Dense)            (None, 198)              39402
_________________________________________________________________
batch_normalization_42 (Batc (None, 198)              792
_________________________________________________________________
dropout_34 (Dropout)         (None, 198)              0
_________________________________________________________________
dense_135 (Dense)            (None, 198)              39402
_________________________________________________________________
batch_normalization_43 (Batc (None, 198)              792
_________________________________________________________________
dropout_35 (Dropout)         (None, 198)              0
_________________________________________________________________
dense_136 (Dense)            (None, 198)              39402
_________________________________________________________________
batch_normalization_44 (Batc (None, 198)              792
_________________________________________________________________
dropout_36 (Dropout)         (None, 198)              0
_________________________________________________________________
dense_137 (Dense)            (None, 10)               1990
=================================================================
Total params: 514,414
Trainable params: 512,038
Non-trainable params: 2,376
_________________________________________________________________
```

In [153]:

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

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 12s 196us/step - loss:
1.2813 - acc: 0.5969 - val_loss: 0.3154 - val_acc: 0.9050
Epoch 2/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.4
875 - acc: 0.8544 - val_loss: 0.1973 - val_acc: 0.9420
Epoch 3/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.3
567 - acc: 0.8957 - val_loss: 0.1590 - val_acc: 0.9525
Epoch 4/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.2
937 - acc: 0.9160 - val_loss: 0.1432 - val_acc: 0.9587
Epoch 5/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.2
639 - acc: 0.9248 - val_loss: 0.1436 - val_acc: 0.9602
Epoch 6/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.2
352 - acc: 0.9354 - val_loss: 0.1306 - val_acc: 0.9639
Epoch 7/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.2
151 - acc: 0.9393 - val_loss: 0.1199 - val_acc: 0.9666
Epoch 8/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.2
029 - acc: 0.9437 - val_loss: 0.1166 - val_acc: 0.9684
Epoch 9/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.1
902 - acc: 0.9475 - val_loss: 0.1049 - val_acc: 0.9722
Epoch 10/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.1
789 - acc: 0.9497 - val_loss: 0.1086 - val_acc: 0.9717
Epoch 11/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.1
688 - acc: 0.9523 - val_loss: 0.1016 - val_acc: 0.9719
Epoch 12/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.1
579 - acc: 0.9564 - val_loss: 0.0948 - val_acc: 0.9741
```

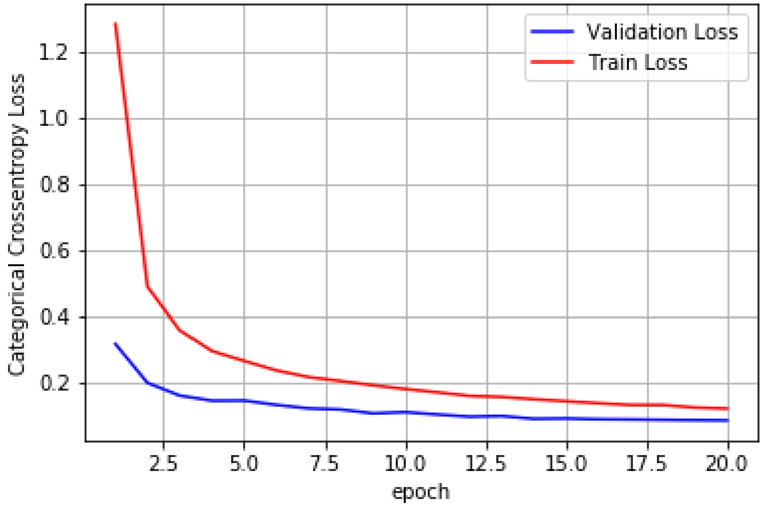
```
379 - acc: 0.9564 - val_loss: 0.0948 - val_acc: 0.9741
Epoch 13/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.1
548 - acc: 0.9578 - val_loss: 0.0969 - val_acc: 0.9736
Epoch 14/20
60000/60000 [==============================] - 6s 104us/step - loss: 0.
1479 - acc: 0.9587 - val_loss: 0.0883 - val_acc: 0.9763
Epoch 15/20
60000/60000 [==============================] - 7s 109us/step - loss: 0.
1417 - acc: 0.9610 - val_loss: 0.0894 - val_acc: 0.9769
Epoch 16/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.1
356 - acc: 0.9621 - val_loss: 0.0871 - val_acc: 0.9771
Epoch 17/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.1
304 - acc: 0.9640 - val_loss: 0.0863 - val_acc: 0.9775
Epoch 18/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.1
303 - acc: 0.9640 - val_loss: 0.0852 - val_acc: 0.9769
Epoch 19/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.1
226 - acc: 0.9661 - val_loss: 0.0843 - val_acc: 0.9775
Epoch 20/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.1
194 - acc: 0.9671 - val_loss: 0.0834 - val_acc: 0.9772
```

In [154]:

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

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

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

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epo
```

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

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

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

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

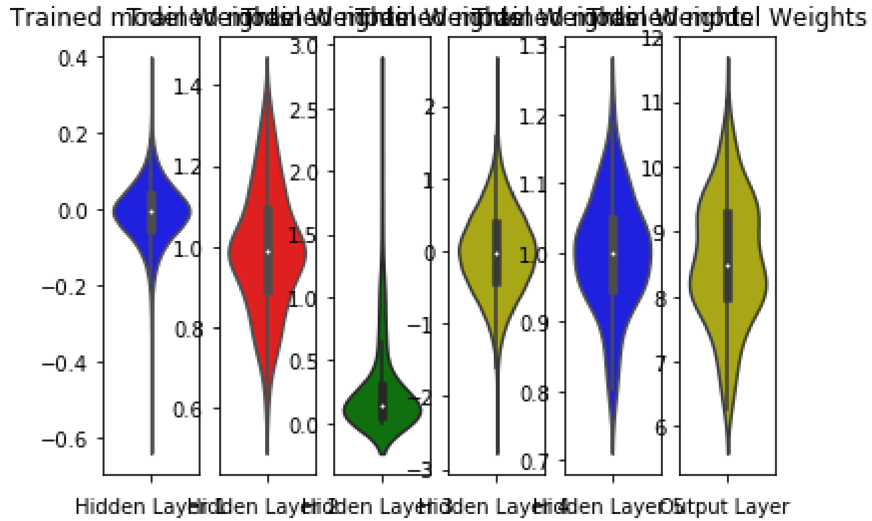
```
Test score: 0.083420745628234
Test accuracy: 0.9772
```



In [155]:

```
w_after = model_drop.get_weights()

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

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


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

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

plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3 ')

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

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

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



(d) MLP + Dropout + AdamOptimizer +Relu

In [140]:
```
# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batc
hnormalization-function-in-keras

from keras.layers import Dropout

model_drop = Sequential()

model_drop.add(Dense(396, activation='relu', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.071, seed=None)))
model_drop.add(Dropout(0.5))

model_drop.add(Dense(198, activation='relu', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.1, seed=None)) )
model_drop.add(Dropout(0.5))

model_drop.add(Dense(98, activation='relu', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.14, seed=None)) )
model_drop.add(Dropout(0.5))
```

```
model_drop.add(Dense(98, activation='relu', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.2, seed=None)) )
model_drop.add(Dropout(0.5))

model_drop.add(Dense(98, activation='relu', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.28, seed=None)) )
model_drop.add(Dropout(0.5))

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


model_drop.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_118 (Dense)            (None, 396)               310860    
_________________________________________________________________
dropout_24 (Dropout)         (None, 396)               0         
_________________________________________________________________
dense_119 (Dense)            (None, 198)               78606     
_________________________________________________________________
dropout_25 (Dropout)         (None, 198)               0         
_________________________________________________________________
dense_120 (Dense)            (None, 98)                19502     
_________________________________________________________________
dropout_26 (Dropout)         (None, 98)                0         
_________________________________________________________________
dense_121 (Dense)            (None, 98)                9702      
_________________________________________________________________
dropout_27 (Dropout)         (None, 98)                0         
_________________________________________________________________
dense_122 (Dense)            (None, 98)                9702      
_________________________________________________________________
dropout_28 (Dropout)         (None, 98)                0         
_________________________________________________________________
dense_123 (Dense)            (None, 10)                990       
=================================================================
Total params: 429,362

Trainable params: 429,362
```

```
Trainable params: 429,362
Non-trainable params: 0
_________________________________________________________________
```

In [141]:
```
model_drop.compile(optimizer='adam', loss='categorical_crossentropy', m
etrics=['accuracy'])

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 109us/step - loss: 3.
0582 - acc: 0.1556 - val_loss: 1.9679 - val_acc: 0.2860
Epoch 2/20
60000/60000 [==============================] - 3s 47us/step - loss: 1.7
921 - acc: 0.3399 - val_loss: 1.4082 - val_acc: 0.4728
Epoch 3/20
60000/60000 [==============================] - 3s 47us/step - loss: 1.4
161 - acc: 0.4615 - val_loss: 1.1216 - val_acc: 0.5522
Epoch 4/20
60000/60000 [==============================] - 3s 49us/step - loss: 1.2
070 - acc: 0.5386 - val_loss: 0.9771 - val_acc: 0.6124
Epoch 5/20
60000/60000 [==============================] - 3s 53us/step - loss: 1.0
516 - acc: 0.6083 - val_loss: 0.7809 - val_acc: 0.7109
Epoch 6/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.9
086 - acc: 0.6694 - val_loss: 0.6112 - val_acc: 0.7783
Epoch 7/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.7
851 - acc: 0.7270 - val_loss: 0.5624 - val_acc: 0.7598
Epoch 8/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.6
841 - acc: 0.7842 - val_loss: 0.5044 - val_acc: 0.8145
Epoch 9/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.5
944 - acc: 0.8349 - val_loss: 0.3985 - val_acc: 0.8392
Epoch 10/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.5
```

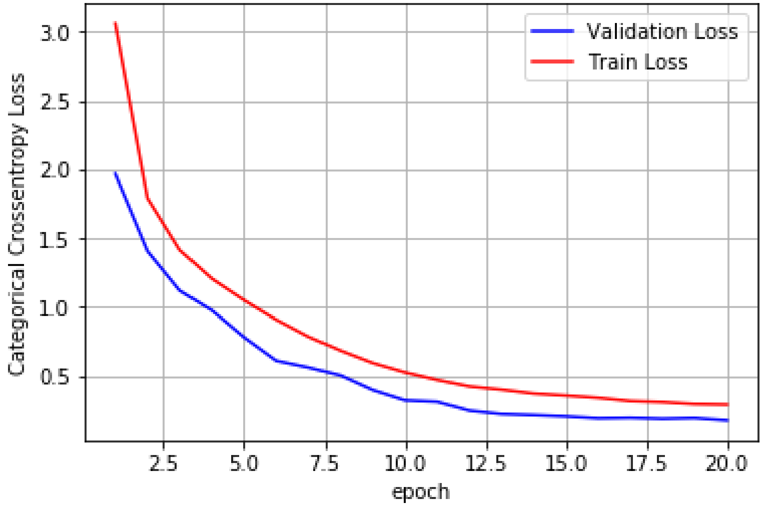
```
267 - acc: 0.8578 - val_loss: 0.3241 - val_acc: 0.8919
Epoch 11/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.4
723 - acc: 0.8759 - val_loss: 0.3136 - val_acc: 0.8852
Epoch 12/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.4
247 - acc: 0.8891 - val_loss: 0.2491 - val_acc: 0.9310
Epoch 13/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.4
012 - acc: 0.8972 - val_loss: 0.2242 - val_acc: 0.9437
Epoch 14/20
60000/60000 [==============================] - 3s 46us/step - loss: 0.3
728 - acc: 0.9068 - val_loss: 0.2165 - val_acc: 0.9465
Epoch 15/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.3
586 - acc: 0.9112 - val_loss: 0.2071 - val_acc: 0.9493
Epoch 16/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.3
419 - acc: 0.9165 - val_loss: 0.1928 - val_acc: 0.9538
Epoch 17/20
60000/60000 [==============================] - 3s 46us/step - loss: 0.3
187 - acc: 0.9223 - val_loss: 0.1964 - val_acc: 0.9526
Epoch 18/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.3
109 - acc: 0.9246 - val_loss: 0.1897 - val_acc: 0.9560
Epoch 19/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.2
971 - acc: 0.9295 - val_loss: 0.1947 - val_acc: 0.9548
Epoch 20/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.2
931 - acc: 0.9297 - val_loss: 0.1779 - val_acc: 0.9567
```

In [142]:

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

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

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

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

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

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

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

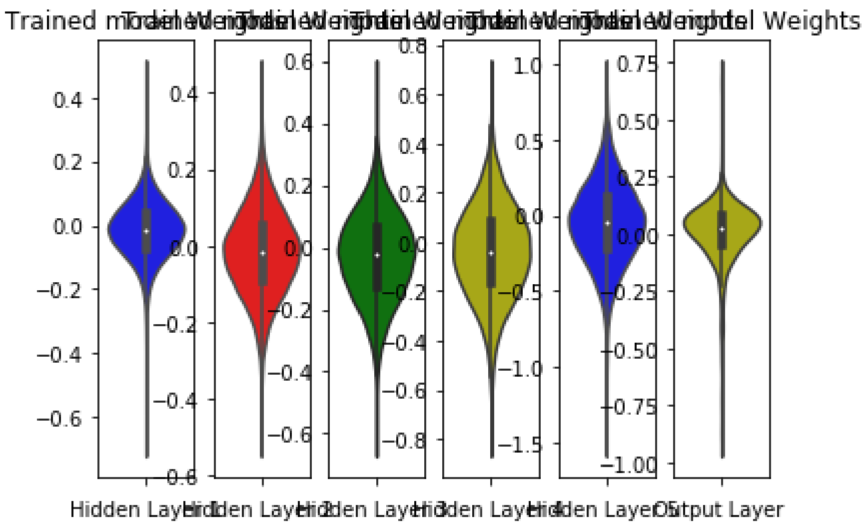
```
Test score: 0.17786330469548703
Test accuracy: 0.9567
```



In [143]:

```
w_after = model_drop.get_weights()

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


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

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

```
plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3 ')

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

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

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



conclusion

- STEP1:collected mnist data
- STEP2:converted into single dimension i.e column
- STEP3:using 2/3/5 hidden layers and activation function as RELU and optimzation as ADAM and BATCHING and DROPOUT are done for the hidden layers

In [158]:

```
from prettytable import PrettyTable

print("******************************using 2 hidden layers***********************************")
names = ['mlp-relu-adam','mlp-batch-adam-relu','mlp-drop-adam-relu-batch','mlp-drop-adam-relu']

acc = [98.01,97.75,98.10,98.22]

numbering = [1,2,3,4]

# Initializing prettytable
table = PrettyTable()

# Adding columns
table.add_column("S.NO.",numbering)
table.add_column("MODEL",names)
table.add_column("acc ",acc)

# Printing the Table
print(table)


print("******************************using 3 hidden layers***********************************")
names = ['mlp-relu-adam','mlp-batch-adam-relu','mlp-drop-adam-relu-batch','mlp-drop-adam-relu']

acc = [98.10,97.90,98.00,95.06]
```

```
numbering = [1,2,3,4]

# Initializing prettytable
table = PrettyTable()

# Adding columns
table.add_column("S.NO.",numbering)
table.add_column("MODEL",names)
table.add_column("acc ",acc)

# Printing the Table
print(table)


print("*****************************using 5 hidden layers*************
**********************")
names = ['mlp-relu-adam','mlp-batch-adam-relu','mlp-drop-adam-relu-batc
h','mlp-drop-adam-relu']

acc = [97.76,97.99,98.72,95.67]

numbering = [1,2,3,4]

# Initializing prettytable
table = PrettyTable()

# Adding columns
table.add_column("S.NO.",numbering)
table.add_column("MODEL",names)
table.add_column("acc ",acc)

# Printing the Table
print(table)
```

```
*****************************using 2 hidden layers*******************
***************
+-------+-------------------------+-------+
| S.NO. |          MODEL          |  acc  |
+-------+-------------------------+-------+
```

```
|   1   |      mlp-relu-adam       | 98.01 |
|   2   |   mlp-batch-adam-relu    | 97.75 |
|   3   | mlp-drop-adam-relu-batch |  98.1 |
|   4   |    mlp-drop-adam-relu    | 98.22 |
+-------+--------------------------+-------+
******************************using 3 hidden layers********************
***************
+-------+--------------------------+-------+
| S.NO. |          MODEL           |  acc  |
+-------+--------------------------+-------+
|   1   |      mlp-relu-adam       |  98.1 |
|   2   |   mlp-batch-adam-relu    |  97.9 |
|   3   | mlp-drop-adam-relu-batch |  98.0 |
|   4   |    mlp-drop-adam-relu    | 95.06 |
+-------+--------------------------+-------+
******************************using 5 hidden layers********************
***************
+-------+--------------------------+-------+
| S.NO. |          MODEL           |  acc  |
+-------+--------------------------+-------+
|   1   |      mlp-relu-adam       | 97.76 |
|   2   |   mlp-batch-adam-relu    | 97.99 |
|   3   | mlp-drop-adam-relu-batch | 98.72 |
|   4   |    mlp-drop-adam-relu    | 95.67 |
+-------+--------------------------+-------+
```