Keras MLPs on MNIST

In [1]:
```
import tensorflow as tf
from tensorflow.keras import utils
from tensorflow.keras.datasets import mnist
import seaborn as sns
from tensorflow.keras.initializers import RandomNormal
from tensorflow.python.keras import Input, Model
from tensorflow.keras.layers import Dense, Activation
from tensorflow.python.keras.layers import Dense, BatchNormalization
from tensorflow.keras.models import Sequential
%matplotlib notebook
import matplotlib.pyplot as plt
import numpy as np
import time
```

In [2]:
```
# 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()
```

1.Load data

In [3]:
```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

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

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

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

In [5]:

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

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

In [7]:

```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  26 166 255
  247 127   0   0   0   0   0   0   0   0   0   0   0   0  30  36  94 154
  170 253 253 253 253 253 225 172 253 242 195  64   0   0   0   0   0   0
    0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251  93  82
   82  56  39   0   0   0   0   0   0   0   0   0   0   0   0  18 219 253
  253 253 253 253 198 182 247 241   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0  14   1 154 253  90   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0  11 190 253  70   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  35 241
  225 160 108   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0  81 240 253 253 119  25   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

```
   0   0  45 186 253 253 150  27   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252 253 187
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0 249 253 249  64   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253
 253 207   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0  39 148 229 253 253 253 250 182   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253
 253 201  78   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  23  66 213 253 253 253 253 198  81   2   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0  18 171 219 253 253 253 253 195
  80   9   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  55 172 226 253 253 253 253 244 133  11   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0 136 253 253 253 212 135 132  16
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
```

In [8]:

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

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

In [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.y{0,1,2.....9}
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 = utils.to_categorical(y_train, 10)
Y_test = 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.]
```

2.simple Input output model with no hidden layer

2.1 intialize parametrs

In [11]:
```
# some model parameters

output_dim = 10
input_dim = X_train.shape[1] # 784

batch_size = 128
nb_epoch = 20
```

2.2 intialize model

In [12]:
```
# start building a model
#input(728)-->outpu(10) as 10 classes are there

model = Sequential()

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

2.3 Configure model and fit on train data

In [13]:
```
# Before training a model, you need to configure the learning process, which is done via the compile method
#fit() function Trains the model for a fixed number of epochs (iterations on a dataset).

model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(X_train, Y_train, steps_per_epoch=500, epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_tes
```

```
Epoch 1/20
500/500 [==============================] - 2s 3ms/step - loss: 1.2684 - accuracy: 0.6990 - val_loss: 0.7962 - val_acc
uracy: 0.8329
Epoch 2/20
500/500 [==============================] - 1s 2ms/step - loss: 0.7006 - accuracy: 0.8413 - val_loss: 0.5967 - val_acc
uracy: 0.8615
Epoch 3/20
500/500 [==============================] - 1s 2ms/step - loss: 0.5761 - accuracy: 0.8595 - val_loss: 0.5176 - val_acc
uracy: 0.8760
Epoch 4/20
500/500 [==============================] - 1s 3ms/step - loss: 0.5165 - accuracy: 0.8697 - val_loss: 0.4731 - val_acc
```

```
uracy: 0.8829
Epoch 5/20
500/500 [==============================] - 1s 3ms/step - loss: 0.4803 - accuracy: 0.8763 - val_loss: 0.4443 - val_acc
uracy: 0.8878
Epoch 6/20
500/500 [==============================] - 1s 2ms/step - loss: 0.4553 - accuracy: 0.8806 - val_loss: 0.4235 - val_acc
uracy: 0.8908
Epoch 7/20
500/500 [==============================] - 1s 2ms/step - loss: 0.4368 - accuracy: 0.8838 - val_loss: 0.4080 - val_acc
uracy: 0.8940
Epoch 8/20
500/500 [==============================] - 1s 2ms/step - loss: 0.4223 - accuracy: 0.8870 - val_loss: 0.3955 - val_acc
uracy: 0.8964
Epoch 9/20
500/500 [==============================] - 1s 2ms/step - loss: 0.4107 - accuracy: 0.8896 - val_loss: 0.3853 - val_acc
uracy: 0.8980
Epoch 10/20
500/500 [==============================] - 1s 2ms/step - loss: 0.4010 - accuracy: 0.8920 - val_loss: 0.3768 - val_acc
uracy: 0.8995
Epoch 11/20
500/500 [==============================] - 1s 3ms/step - loss: 0.3928 - accuracy: 0.8939 - val_loss: 0.3695 - val_acc
uracy: 0.9005
Epoch 12/20
500/500 [==============================] - 1s 2ms/step - loss: 0.3857 - accuracy: 0.8956 - val_loss: 0.3637 - val_acc
uracy: 0.9018
Epoch 13/20
500/500 [==============================] - 1s 3ms/step - loss: 0.3795 - accuracy: 0.8967 - val_loss: 0.3582 - val_acc
uracy: 0.9028
Epoch 14/20
500/500 [==============================] - 1s 2ms/step - loss: 0.3741 - accuracy: 0.8980 - val_loss: 0.3533 - val_acc
uracy: 0.9042
Epoch 15/20
500/500 [==============================] - 1s 2ms/step - loss: 0.3691 - accuracy: 0.8986 - val_loss: 0.3494 - val_acc
uracy: 0.9040
Epoch 16/20
500/500 [==============================] - 1s 2ms/step - loss: 0.3648 - accuracy: 0.8996 - val_loss: 0.3451 - val_acc
uracy: 0.9062
Epoch 17/20
500/500 [==============================] - 1s 2ms/step - loss: 0.3608 - accuracy: 0.9006 - val_loss: 0.3416 - val_acc
uracy: 0.9064
Epoch 18/20
500/500 [==============================] - 1s 2ms/step - loss: 0.3572 - accuracy: 0.9013 - val_loss: 0.3387 - val_acc
uracy: 0.9070
Epoch 19/20
500/500 [==============================] - 1s 2ms/step - loss: 0.3538 - accuracy: 0.9021 - val_loss: 0.3357 - val_acc
```

```
uracy: 0.9077
Epoch 20/20
500/500 [==============================] - 1s 2ms/step - loss: 0.3507 - accuracy: 0.9024 - val_loss: 0.3330 - val_acc
uracy: 0.9081
```

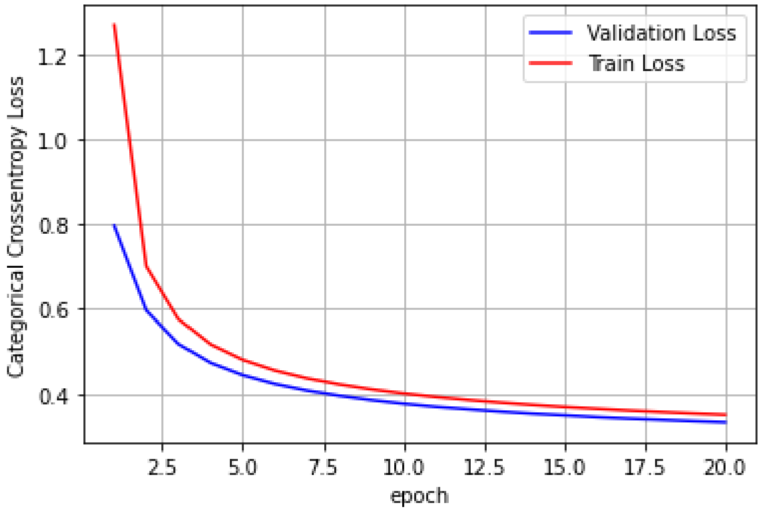
In [13]:

```

```

In [14]:

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

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

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

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

# 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: 0.3329598903656006
Test accuracy: 0.9081000089645386
```



Assignment start Below

1.Two hidden layer Architecture

1.1 Two hidden layer with Relu + Adam

In [28]:
```
model1_relu = Sequential()
model1_relu.add(Dense(420, activation='relu', input_shape=(input_dim,)))
model1_relu.add(Dense(100, activation='relu'))
model1_relu.add(Dense(output_dim, activation='softmax'))
model1_relu.build()
model1_relu.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_3 (Dense)              (None, 420)               329700    
_________________________________________________________________
dense_4 (Dense)              (None, 100)               42100     
_________________________________________________________________
```

```
dense_5 (Dense)              (None, 10)               1010
=================================================================
Total params: 372,810
Trainable params: 372,810
Non-trainable params: 0
_________________________________________________________________
```

In [18]:
```
model1_relu.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model1_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_tes
```

```
Epoch 1/20
469/469 [==============================] - 5s 9ms/step - loss: 0.2457 - accuracy: 0.9300 - val_loss: 0.1191 - val_acc
uracy: 0.9641
Epoch 2/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0928 - accuracy: 0.9722 - val_loss: 0.0884 - val_acc
uracy: 0.9710
Epoch 3/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0586 - accuracy: 0.9820 - val_loss: 0.0790 - val_acc
uracy: 0.9762
Epoch 4/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0411 - accuracy: 0.9870 - val_loss: 0.0800 - val_acc
uracy: 0.9756
Epoch 5/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0319 - accuracy: 0.9899 - val_loss: 0.0694 - val_acc
uracy: 0.9803
Epoch 6/20
469/469 [==============================] - 4s 8ms/step - loss: 0.0223 - accuracy: 0.9929 - val_loss: 0.0705 - val_acc
uracy: 0.9802
Epoch 7/20
469/469 [==============================] - 4s 8ms/step - loss: 0.0195 - accuracy: 0.9938 - val_loss: 0.0730 - val_acc
uracy: 0.9793
Epoch 8/20
469/469 [==============================] - 4s 8ms/step - loss: 0.0140 - accuracy: 0.9955 - val_loss: 0.0782 - val_acc
uracy: 0.9786
Epoch 9/20
469/469 [==============================] - 4s 8ms/step - loss: 0.0138 - accuracy: 0.9955 - val_loss: 0.0963 - val_acc
uracy: 0.9772
Epoch 10/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0120 - accuracy: 0.9960 - val_loss: 0.0760 - val_acc
uracy: 0.9790
Epoch 11/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0108 - accuracy: 0.9967 - val_loss: 0.0870 - val_acc
uracy: 0.9791
Epoch 12/20
```

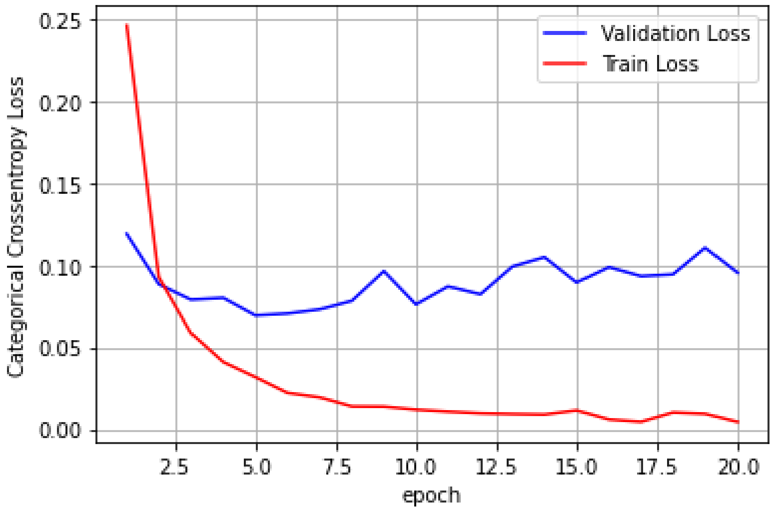
```
469/469 [==============================] - 4s 9ms/step - loss: 0.0098 - accuracy: 0.9967 - val_loss: 0.0822 - val_acc
uracy: 0.9801
Epoch 13/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0093 - accuracy: 0.9969 - val_loss: 0.0991 - val_acc
uracy: 0.9788
Epoch 14/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0090 - accuracy: 0.9968 - val_loss: 0.1048 - val_acc
uracy: 0.9792
Epoch 15/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0115 - accuracy: 0.9957 - val_loss: 0.0894 - val_acc
uracy: 0.9804
Epoch 16/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0060 - accuracy: 0.9979 - val_loss: 0.0987 - val_acc
uracy: 0.9800
Epoch 17/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0046 - accuracy: 0.9985 - val_loss: 0.0932 - val_acc
uracy: 0.9811
Epoch 18/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0103 - accuracy: 0.9967 - val_loss: 0.0944 - val_acc
uracy: 0.9812
Epoch 19/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0094 - accuracy: 0.9967 - val_loss: 0.1106 - val_acc
uracy: 0.9792
Epoch 20/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0045 - accuracy: 0.9987 - val_loss: 0.0953 - val_acc
uracy: 0.9829
```

In [19]:

```
score = model1_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')


x = list(range(1,nb_epoch+1))



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

```
Test score: 0.09534730017185211
```

```
Test accuracy: 0.9829000234603882
```



In [21]:

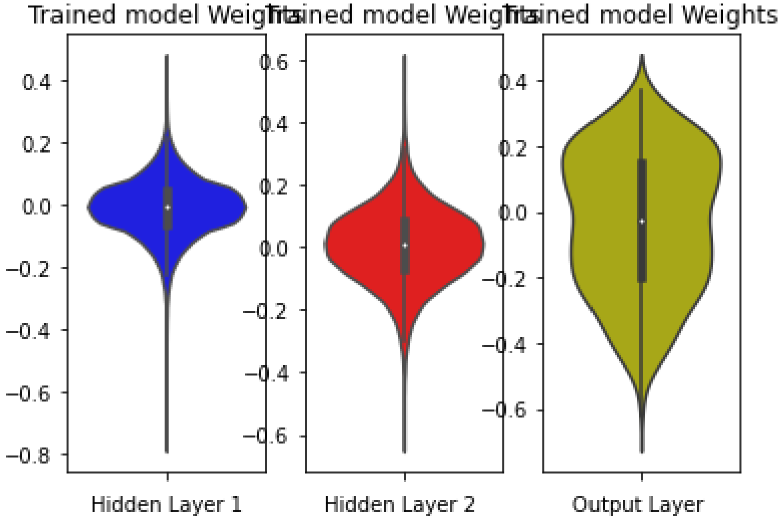
```
for layers in model1_relu.layers:
    print("layer name",layers.name,",input shape",layers.input_shape,",output shape",layers.output_shape)
w_after = model1_relu.get_weights()
print(len(w_after))

for i in range(len(w_after)):
    print(w_after[i].shape)
```

```
layer name module_wrapper_7 ,input shape (None, 784) ,output shape (None, 420)
layer name module_wrapper_8 ,input shape (None, 420) ,output shape (None, 100)
layer name module_wrapper_9 ,input shape (None, 100) ,output shape (None, 10)
6
(784, 420)
(420,)
(420, 100)
(100,)
(100, 10)
(10,)
```

In [22]:

```
w_after = model1_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()
```



batch normalization

In [27]:
```
model1_batch = Sequential()
```

```
model1_batch.add(Dense(420, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.069,
model1_batch.add(BatchNormalization())

model1_batch.add(Dense(100, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.141, seed=None)) )
model1_batch.add(BatchNormalization())

model1_batch.add(Dense(output_dim, activation='softmax'))

model.build()
model1_batch.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense (Dense)                (None, 420)               329700    
_________________________________________________________________
module_wrapper_10 (ModuleWra (None, 420)               1680      
_________________________________________________________________
dense_1 (Dense)              (None, 100)               42100     
_________________________________________________________________
module_wrapper_11 (ModuleWra (None, 100)               400       
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1010      
=================================================================
Total params: 374,890
Trainable params: 373,850
Non-trainable params: 1,040
_________________________________________________________________
```

In [29]:

```
model1_batch.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model1_batch.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_te
```

```
Epoch 1/20
469/469 [==============================] - 6s 11ms/step - loss: 0.3112 - accuracy: 0.9096 - val_loss: 0.1775 - val_ac
curacy: 0.9477
Epoch 2/20
469/469 [==============================] - 5s 10ms/step - loss: 0.1577 - accuracy: 0.9543 - val_loss: 0.1359 - val_ac
curacy: 0.9596
Epoch 3/20
469/469 [==============================] - 5s 10ms/step - loss: 0.1183 - accuracy: 0.9650 - val_loss: 0.1070 - val_ac
curacy: 0.9687
Epoch 4/20
```

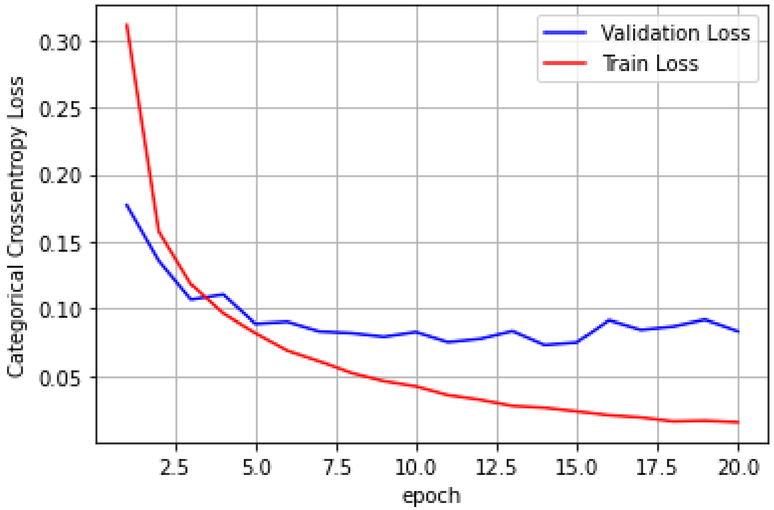
```
469/469 [==============================] - 5s 11ms/step - loss: 0.0967 - accuracy: 0.9714 - val_loss: 0.1107 - val_ac
curacy: 0.9670
Epoch 5/20
469/469 [==============================] - 5s 11ms/step - loss: 0.0821 - accuracy: 0.9747 - val_loss: 0.0890 - val_ac
curacy: 0.9724
Epoch 6/20
469/469 [==============================] - 5s 11ms/step - loss: 0.0691 - accuracy: 0.9788 - val_loss: 0.0905 - val_ac
curacy: 0.9737
Epoch 7/20
469/469 [==============================] - 5s 11ms/step - loss: 0.0611 - accuracy: 0.9812 - val_loss: 0.0833 - val_ac
curacy: 0.9747
Epoch 8/20
469/469 [==============================] - 5s 11ms/step - loss: 0.0523 - accuracy: 0.9840 - val_loss: 0.0821 - val_ac
curacy: 0.9757
Epoch 9/20
469/469 [==============================] - 5s 11ms/step - loss: 0.0463 - accuracy: 0.9854 - val_loss: 0.0794 - val_ac
curacy: 0.9765
Epoch 10/20
469/469 [==============================] - 5s 11ms/step - loss: 0.0424 - accuracy: 0.9867 - val_loss: 0.0830 - val_ac
curacy: 0.9759
Epoch 11/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0358 - accuracy: 0.9885 - val_loss: 0.0754 - val_ac
curacy: 0.9777
Epoch 12/20
469/469 [==============================] - 5s 11ms/step - loss: 0.0323 - accuracy: 0.9895 - val_loss: 0.0779 - val_ac
curacy: 0.9791
Epoch 13/20
469/469 [==============================] - 5s 11ms/step - loss: 0.0278 - accuracy: 0.9913 - val_loss: 0.0837 - val_ac
curacy: 0.9768
Epoch 14/20
469/469 [==============================] - 5s 11ms/step - loss: 0.0264 - accuracy: 0.9914 - val_loss: 0.0733 - val_ac
curacy: 0.9781
Epoch 15/20
469/469 [==============================] - 5s 10ms/step - loss: 0.0237 - accuracy: 0.9926 - val_loss: 0.0751 - val_ac
curacy: 0.9781
Epoch 16/20
469/469 [==============================] - 5s 11ms/step - loss: 0.0208 - accuracy: 0.9935 - val_loss: 0.0916 - val_ac
curacy: 0.9760
Epoch 17/20
469/469 [==============================] - 5s 12ms/step - loss: 0.0192 - accuracy: 0.9938 - val_loss: 0.0845 - val_ac
curacy: 0.9782
Epoch 18/20
469/469 [==============================] - 7s 14ms/step - loss: 0.0162 - accuracy: 0.9951 - val_loss: 0.0870 - val_ac
curacy: 0.9767
Epoch 19/20
```

```
469/469 [==============================] - 5s 11ms/step - loss: 0.0167 - accuracy: 0.9948 - val_loss: 0.0921 - val_ac
curacy: 0.9762
Epoch 20/20
469/469 [==============================] - 5s 11ms/step - loss: 0.0156 - accuracy: 0.9949 - val_loss: 0.0835 - val_ac
curacy: 0.9791
```

In [30]:

```
score = model1_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_te

# 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: 0.08354396373033524
Test accuracy: 0.9790999889373779
```



In [31]:

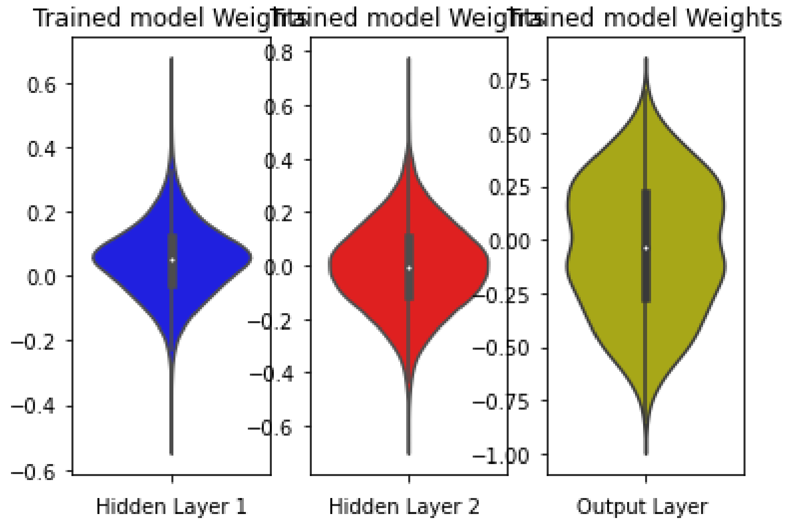
```
for layers in model1_batch.layers:
    print("layer name",layers.name,",input shape",layers.input_shape,",output shape",layers.output_shape)
w_after = model1_batch.get_weights()
print(len(w_after))

for i in range(len(w_after)):
    print(w_after[i].shape)
```

```
layer name dense ,input shape (None, 784) ,output shape (None, 420)
layer name module_wrapper_10 ,input shape (None, 420) ,output shape (None, 420)
layer name dense_1 ,input shape (None, 420) ,output shape (None, 100)
layer name module_wrapper_11 ,input shape (None, 100) ,output shape (None, 100)
layer name dense_2 ,input shape (None, 100) ,output shape (None, 10)
14
(784, 420)
(420,)
(420,)
(420,)
(420,)
(420,)
(420, 100)
(100,)
(100,)
(100,)
(100,)
(100,)
```

```
(100, 10)
(10,)
```

In [32]:

```
w_after = model1_batch.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[6].flatten().reshape(-1,1)
out_w = w_after[12].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()
```



dropout

In [ ]:
```
# Using relu
#If we sample weights from a normal distribution N(0,σ) we satisfy this condition with σ=√(2/(ni).
#h1 => σ=√(2/(fan_in) = 0.062 => N(0,σ) = N(0,0.062)
#h2 => σ=√(2/(fan_in) = 0.125 => N(0,σ) = N(0,0.125)
#out => σ=√(2/(fan_in+1) = 0.120 => N(0,σ) = N(0,0.120)
```

In [33]:
```
from tensorflow.keras.layers import Dropout

model1_drop = Sequential()

model1_drop.add(Dense(420, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, std
model1_drop.add(BatchNormalization())
model1_drop.add(Dropout(0.5))

model1_drop.add(Dense(100, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.141, seed=None)) )
model1_drop.add(BatchNormalization())
model1_drop.add(Dropout(0.5))

model1_drop.add(Dense(output_dim, activation='softmax'))
```

```
model1_drop.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_6 (Dense)              (None, 420)               329700    
_________________________________________________________________
module_wrapper_12 (ModuleWra (None, 420)               1680      
_________________________________________________________________
dropout (Dropout)            (None, 420)               0         
_________________________________________________________________
dense_7 (Dense)              (None, 100)               42100     
_________________________________________________________________
module_wrapper_13 (ModuleWra (None, 100)               400       
_________________________________________________________________
dropout_1 (Dropout)          (None, 100)               0         
_________________________________________________________________
dense_8 (Dense)              (None, 10)                1010      
=================================================================
Total params: 374,890
Trainable params: 373,850
Non-trainable params: 1,040
_________________________________________________________________
```

In [34]:

```
model1_drop.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model1_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_tes
```

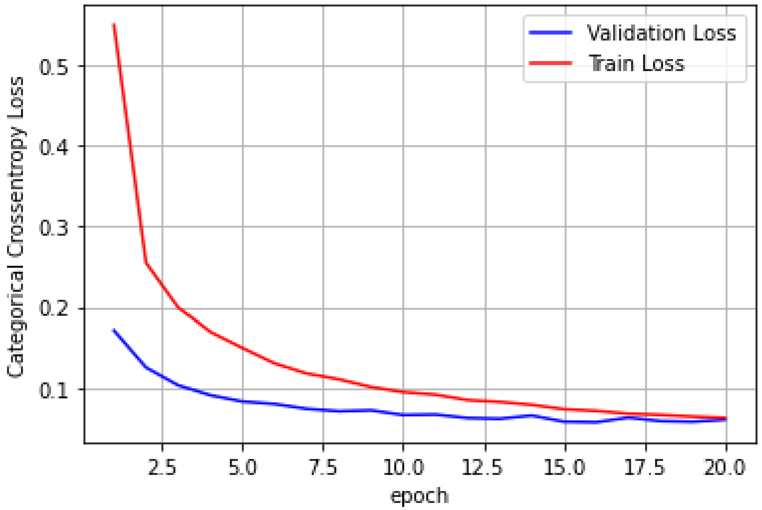
```
Epoch 1/20
469/469 [==============================] - 6s 12ms/step - loss: 0.5487 - accuracy: 0.8320 - val_loss: 0.1711 - val_ac
curacy: 0.9469
Epoch 2/20
469/469 [==============================] - 5s 11ms/step - loss: 0.2547 - accuracy: 0.9239 - val_loss: 0.1253 - val_ac
curacy: 0.9602
Epoch 3/20
469/469 [==============================] - 5s 12ms/step - loss: 0.1995 - accuracy: 0.9407 - val_loss: 0.1032 - val_ac
curacy: 0.9686
Epoch 4/20
469/469 [==============================] - 5s 11ms/step - loss: 0.1692 - accuracy: 0.9500 - val_loss: 0.0908 - val_ac
curacy: 0.9722
Epoch 5/20
469/469 [==============================] - 5s 12ms/step - loss: 0.1493 - accuracy: 0.9552 - val_loss: 0.0830 - val_ac
curacy: 0.9740
```

```
Epoch 6/20
469/469 [==============================] - 6s 12ms/step - loss: 0.1304 - accuracy: 0.9610 - val_loss: 0.0800 - val_ac
curacy: 0.9738
Epoch 7/20
469/469 [==============================] - 6s 12ms/step - loss: 0.1179 - accuracy: 0.9642 - val_loss: 0.0740 - val_ac
curacy: 0.9759
Epoch 8/20
469/469 [==============================] - 5s 11ms/step - loss: 0.1106 - accuracy: 0.9657 - val_loss: 0.0709 - val_ac
curacy: 0.9776
Epoch 9/20
469/469 [==============================] - 5s 11ms/step - loss: 0.1010 - accuracy: 0.9687 - val_loss: 0.0721 - val_ac
curacy: 0.9789
Epoch 10/20
469/469 [==============================] - 5s 11ms/step - loss: 0.0947 - accuracy: 0.9720 - val_loss: 0.0663 - val_ac
curacy: 0.9801
Epoch 11/20
469/469 [==============================] - 5s 11ms/step - loss: 0.0916 - accuracy: 0.9718 - val_loss: 0.0669 - val_ac
curacy: 0.9796
Epoch 12/20
469/469 [==============================] - 5s 11ms/step - loss: 0.0848 - accuracy: 0.9735 - val_loss: 0.0625 - val_ac
curacy: 0.9805
Epoch 13/20
469/469 [==============================] - 5s 10ms/step - loss: 0.0825 - accuracy: 0.9746 - val_loss: 0.0616 - val_ac
curacy: 0.9813
Epoch 14/20
469/469 [==============================] - 5s 11ms/step - loss: 0.0789 - accuracy: 0.9746 - val_loss: 0.0657 - val_ac
curacy: 0.9798
Epoch 15/20
469/469 [==============================] - 5s 11ms/step - loss: 0.0735 - accuracy: 0.9769 - val_loss: 0.0580 - val_ac
curacy: 0.9820
Epoch 16/20
469/469 [==============================] - 5s 11ms/step - loss: 0.0714 - accuracy: 0.9780 - val_loss: 0.0573 - val_ac
curacy: 0.9819
Epoch 17/20
469/469 [==============================] - 5s 11ms/step - loss: 0.0679 - accuracy: 0.9784 - val_loss: 0.0631 - val_ac
curacy: 0.9804
Epoch 18/20
469/469 [==============================] - 5s 11ms/step - loss: 0.0665 - accuracy: 0.9786 - val_loss: 0.0587 - val_ac
curacy: 0.9829
Epoch 19/20
469/469 [==============================] - 5s 11ms/step - loss: 0.0643 - accuracy: 0.9798 - val_loss: 0.0579 - val_ac
curacy: 0.9822
Epoch 20/20
469/469 [==============================] - 5s 11ms/step - loss: 0.0628 - accuracy: 0.9805 - val_loss: 0.0604 - val_ac
curacy: 0.9819
```

In [35]:

```
score = model1_drop.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

# 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: 0.060421794652938 84
Test accuracy: 0.9818999767303467
```



In [36]:

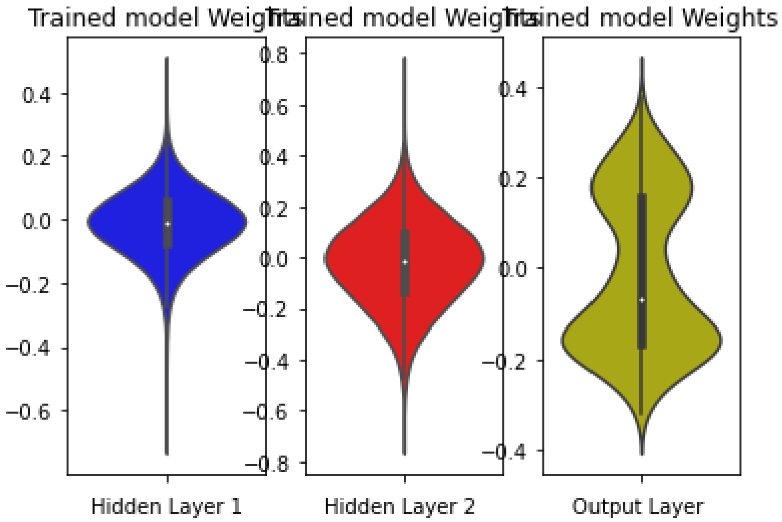
```
for layers in model1_drop.layers:
    print("layer name",layers.name,",input shape",layers.input_shape,",output shape",layers.output_shape)
w_after = model1_drop.get_weights()
print(len(w_after))

for i in range(len(w_after)):
    print(w_after[i].shape)
```

```
layer name dense_6 ,input shape (None, 784) ,output shape (None, 420)
layer name module_wrapper_12 ,input shape (None, 420) ,output shape (None, 420)
layer name dropout ,input shape (None, 420) ,output shape (None, 420)
layer name dense_7 ,input shape (None, 420) ,output shape (None, 100)
layer name module_wrapper_13 ,input shape (None, 100) ,output shape (None, 100)
layer name dropout_1 ,input shape (None, 100) ,output shape (None, 100)
layer name dense_8 ,input shape (None, 100) ,output shape (None, 10)
14
(784, 420)
(420,)
(420,)
(420,)
(420,)
(420,)
(420, 100)
(100,)
(100,)
(100,)
```

```
(100,)
(100,)
(100, 10)
(10,)
```

In [37]:

```
w_after = model1_drop.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[6].flatten().reshape(-1,1)
out_w = w_after[12].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.Three hidden layer Architecture

2.1 Three hidden layer with Relu + Adam

In [38]:
```
batch_size = 150
nb_epoch = 30
```

In [39]:
```
model2_relu = Sequential()
model2_relu.add(Dense(520, activation='relu', input_shape=(input_dim,)))
model2_relu.add(Dense(320, activation='relu'))
model2_relu.add(Dense(120, activation='relu'))
model2_relu.add(Dense(output_dim, activation='softmax'))
model2_relu.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_9 (Dense)              (None, 520)               408200
_________________________________________________________________
dense_10 (Dense)             (None, 320)               166720
_________________________________________________________________
```

```
dense_11 (Dense)             (None, 120)               38520
_________________________________________________________________
dense_12 (Dense)             (None, 10)                1210
=================================================================
Total params: 614,650
Trainable params: 614,650
Non-trainable params: 0
_________________________________________________________________
```

In [40]:
```
model2_relu.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model2_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_tes
```

```
Epoch 1/30
400/400 [==============================] - 7s 16ms/step - loss: 0.2379 - accuracy: 0.9280 - val_loss: 0.1198 - val_ac
curacy: 0.9630
Epoch 2/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0861 - accuracy: 0.9727 - val_loss: 0.0812 - val_ac
curacy: 0.9731
Epoch 3/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0577 - accuracy: 0.9821 - val_loss: 0.0752 - val_ac
curacy: 0.9773
Epoch 4/30
400/400 [==============================] - 6s 16ms/step - loss: 0.0407 - accuracy: 0.9867 - val_loss: 0.0783 - val_ac
curacy: 0.9769
Epoch 5/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0296 - accuracy: 0.9907 - val_loss: 0.0691 - val_ac
curacy: 0.9797
Epoch 6/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0261 - accuracy: 0.9915 - val_loss: 0.0780 - val_ac
curacy: 0.9784
Epoch 7/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0210 - accuracy: 0.9931 - val_loss: 0.0629 - val_ac
curacy: 0.9817
Epoch 8/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0188 - accuracy: 0.9936 - val_loss: 0.0651 - val_ac
curacy: 0.9820
Epoch 9/30
400/400 [==============================] - 6s 16ms/step - loss: 0.0174 - accuracy: 0.9947 - val_loss: 0.0786 - val_ac
curacy: 0.9803
Epoch 10/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0162 - accuracy: 0.9947 - val_loss: 0.0729 - val_ac
curacy: 0.9809
Epoch 11/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0153 - accuracy: 0.9945 - val_loss: 0.0823 - val_ac
```

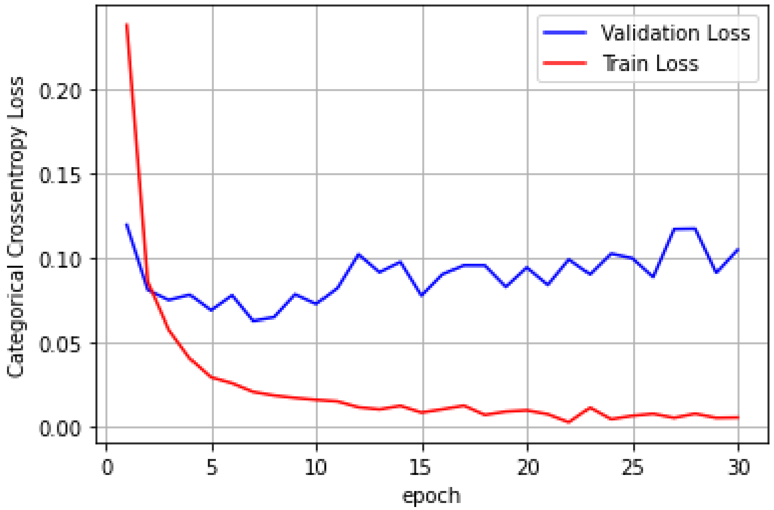
```
curacy: 0.9805
Epoch 12/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0118 - accuracy: 0.9962 - val_loss: 0.1023 - val_ac
curacy: 0.9775
Epoch 13/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0106 - accuracy: 0.9966 - val_loss: 0.0917 - val_ac
curacy: 0.9794
Epoch 14/30
400/400 [==============================] - 6s 16ms/step - loss: 0.0127 - accuracy: 0.9962 - val_loss: 0.0978 - val_ac
curacy: 0.9784
Epoch 15/30
400/400 [==============================] - 6s 16ms/step - loss: 0.0087 - accuracy: 0.9973 - val_loss: 0.0778 - val_ac
curacy: 0.9822
Epoch 16/30
400/400 [==============================] - 6s 16ms/step - loss: 0.0106 - accuracy: 0.9966 - val_loss: 0.0905 - val_ac
curacy: 0.9806
Epoch 17/30
400/400 [==============================] - 6s 16ms/step - loss: 0.0128 - accuracy: 0.9959 - val_loss: 0.0958 - val_ac
curacy: 0.9818
Epoch 18/30
400/400 [==============================] - 6s 16ms/step - loss: 0.0073 - accuracy: 0.9978 - val_loss: 0.0957 - val_ac
curacy: 0.9821
Epoch 19/30
400/400 [==============================] - 6s 16ms/step - loss: 0.0093 - accuracy: 0.9970 - val_loss: 0.0830 - val_ac
curacy: 0.9842
Epoch 20/30
400/400 [==============================] - 6s 16ms/step - loss: 0.0100 - accuracy: 0.9967 - val_loss: 0.0945 - val_ac
curacy: 0.9824
Epoch 21/30
400/400 [==============================] - 6s 16ms/step - loss: 0.0077 - accuracy: 0.9978 - val_loss: 0.0842 - val_ac
curacy: 0.9842
Epoch 22/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0029 - accuracy: 0.9991 - val_loss: 0.0993 - val_ac
curacy: 0.9823
Epoch 23/30
400/400 [==============================] - 6s 16ms/step - loss: 0.0116 - accuracy: 0.9965 - val_loss: 0.0904 - val_ac
curacy: 0.9818
Epoch 24/30
400/400 [==============================] - 6s 16ms/step - loss: 0.0048 - accuracy: 0.9986 - val_loss: 0.1027 - val_ac
curacy: 0.9815
Epoch 25/30
400/400 [==============================] - 6s 16ms/step - loss: 0.0068 - accuracy: 0.9981 - val_loss: 0.1000 - val_ac
curacy: 0.9822
Epoch 26/30
400/400 [==============================] - 6s 16ms/step - loss: 0.0080 - accuracy: 0.9974 - val_loss: 0.0888 - val_ac
```

```
curacy: 0.9848
Epoch 27/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0056 - accuracy: 0.9983 - val_loss: 0.1173 - val_ac
curacy: 0.9789
Epoch 28/30
400/400 [==============================] - 6s 16ms/step - loss: 0.0080 - accuracy: 0.9974 - val_loss: 0.1175 - val_ac
curacy: 0.9791
Epoch 29/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0055 - accuracy: 0.9982 - val_loss: 0.0915 - val_ac
curacy: 0.9839
Epoch 30/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0057 - accuracy: 0.9985 - val_loss: 0.1050 - val_ac
curacy: 0.9813
```

In [41]:

```
score = model2_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')


x = list(range(1,nb_epoch+1))



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

```
Test score: 0.1049598753452301
Test accuracy: 0.9812999963760376
```

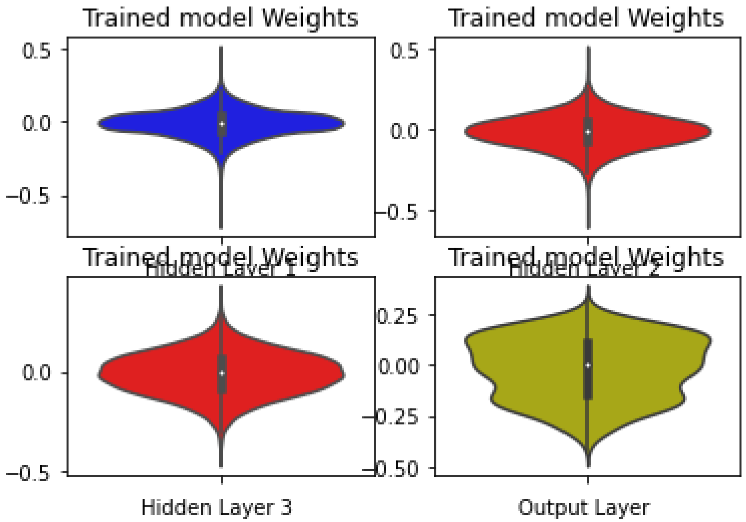


In [42]:
```
for layers in model2_relu.layers:
    print("layer name",layers.name,",input shape",layers.input_shape,",output shape",layers.output_shape,",bias shape
w_after = model2_relu.get_weights()
print(len(w_after))

for i in range(len(w_after)):
    print(w_after[i].shape)
```

```
layer name dense_9 ,input shape (None, 784) ,output shape (None, 520) ,bias shape (520,)
layer name dense_10 ,input shape (None, 520) ,output shape (None, 320) ,bias shape (320,)
layer name dense_11 ,input shape (None, 320) ,output shape (None, 120) ,bias shape (120,)
layer name dense_12 ,input shape (None, 120) ,output shape (None, 10) ,bias shape (10,)
8
(784, 520)
(520,)
(520, 320)
(320,)
(320, 120)
(120,)
(120, 10)
(10,)
```

In [43]:
```
w_after = model2_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(2, 2, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

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

plt.subplot(2, 2, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='r')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(2, 2, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



2.2 batch normalization

In [46]:
```
model2_batch = Sequential()

model2_batch.add(Dense(500, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, st
model2_batch.add(BatchNormalization())

model2_batch.add(Dense(200, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.1, seed=None)) )
model2_batch.add(BatchNormalization())

model2_batch.add(Dense(50, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.2, seed=None)) )
model2_batch.add(BatchNormalization())

model2_batch.add(Dense(output_dim, activation='softmax'))


model2_batch.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_14 (Dense)             (None, 500)               392500
_________________________________________________________________
```

```
module_wrapper_14 (ModuleWra (None, 500)              2000
_________________________________________________________________
dense_15 (Dense)             (None, 200)              100200
_________________________________________________________________
module_wrapper_15 (ModuleWra (None, 200)              800
_________________________________________________________________
dense_16 (Dense)             (None, 50)               10050
_________________________________________________________________
module_wrapper_16 (ModuleWra (None, 50)               200
_________________________________________________________________
dense_17 (Dense)             (None, 10)               510
=================================================================
Total params: 506,260
Trainable params: 504,760
Non-trainable params: 1,500
_________________________________________________________________
```

In [47]:

```
model2_batch.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model2_batch.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_te
```

```
Epoch 1/30
400/400 [==============================] - 7s 16ms/step - loss: 0.2290 - accuracy: 0.9350 - val_loss: 0.1178 - val_ac
curacy: 0.9637
Epoch 2/30
400/400 [==============================] - 6s 16ms/step - loss: 0.0743 - accuracy: 0.9780 - val_loss: 0.0802 - val_ac
curacy: 0.9758
Epoch 3/30
400/400 [==============================] - 6s 16ms/step - loss: 0.0442 - accuracy: 0.9870 - val_loss: 0.0752 - val_ac
curacy: 0.9767
Epoch 4/30
400/400 [==============================] - 6s 16ms/step - loss: 0.0301 - accuracy: 0.9908 - val_loss: 0.0854 - val_ac
curacy: 0.9740
Epoch 5/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0268 - accuracy: 0.9916 - val_loss: 0.0915 - val_ac
curacy: 0.9738
Epoch 6/30
400/400 [==============================] - 6s 16ms/step - loss: 0.0217 - accuracy: 0.9936 - val_loss: 0.0976 - val_ac
curacy: 0.9717
Epoch 7/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0207 - accuracy: 0.9935 - val_loss: 0.0760 - val_ac
curacy: 0.9771
Epoch 8/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0162 - accuracy: 0.9947 - val_loss: 0.0770 - val_ac
curacy: 0.9779
```

```
Epoch 9/30
400/400 [==============================] - 6s 16ms/step - loss: 0.0147 - accuracy: 0.9952 - val_loss: 0.0820 - val_ac
curacy: 0.9779
Epoch 10/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0146 - accuracy: 0.9951 - val_loss: 0.0770 - val_ac
curacy: 0.9778
Epoch 11/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0105 - accuracy: 0.9967 - val_loss: 0.0809 - val_ac
curacy: 0.9790
Epoch 12/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0105 - accuracy: 0.9964 - val_loss: 0.0783 - val_ac
curacy: 0.9811
Epoch 13/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0093 - accuracy: 0.9970 - val_loss: 0.0772 - val_ac
curacy: 0.9784
Epoch 14/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0114 - accuracy: 0.9958 - val_loss: 0.0724 - val_ac
curacy: 0.9799
Epoch 15/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0090 - accuracy: 0.9969 - val_loss: 0.0661 - val_ac
curacy: 0.9815
Epoch 16/30
400/400 [==============================] - 6s 14ms/step - loss: 0.0083 - accuracy: 0.9973 - val_loss: 0.0846 - val_ac
curacy: 0.9785
Epoch 17/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0110 - accuracy: 0.9966 - val_loss: 0.0948 - val_ac
curacy: 0.9774
Epoch 18/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0085 - accuracy: 0.9974 - val_loss: 0.0715 - val_ac
curacy: 0.9820
Epoch 19/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0069 - accuracy: 0.9978 - val_loss: 0.0677 - val_ac
curacy: 0.9832
Epoch 20/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0060 - accuracy: 0.9980 - val_loss: 0.0833 - val_ac
curacy: 0.9813
Epoch 21/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0074 - accuracy: 0.9976 - val_loss: 0.0826 - val_ac
curacy: 0.9807
Epoch 22/30
400/400 [==============================] - 6s 14ms/step - loss: 0.0090 - accuracy: 0.9967 - val_loss: 0.0745 - val_ac
curacy: 0.9828
Epoch 23/30
400/400 [==============================] - 6s 14ms/step - loss: 0.0057 - accuracy: 0.9980 - val_loss: 0.0768 - val_ac
curacy: 0.9827
```

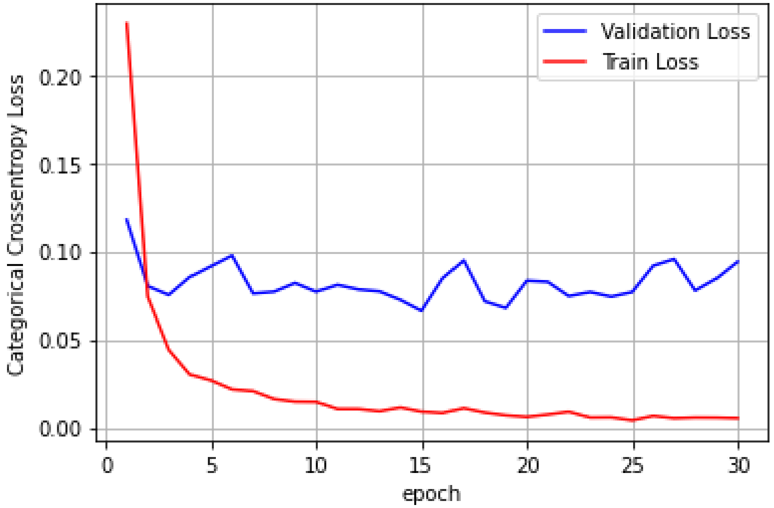
```
Epoch 24/30
400/400 [==============================] - 6s 14ms/step - loss: 0.0058 - accuracy: 0.9982 - val_loss: 0.0742 - val_ac
curacy: 0.9823
Epoch 25/30
400/400 [==============================] - 6s 14ms/step - loss: 0.0041 - accuracy: 0.9985 - val_loss: 0.0768 - val_ac
curacy: 0.9811
Epoch 26/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0066 - accuracy: 0.9978 - val_loss: 0.0917 - val_ac
curacy: 0.9797
Epoch 27/30
400/400 [==============================] - 6s 14ms/step - loss: 0.0053 - accuracy: 0.9983 - val_loss: 0.0955 - val_ac
curacy: 0.9804
Epoch 28/30
400/400 [==============================] - 6s 14ms/step - loss: 0.0056 - accuracy: 0.9983 - val_loss: 0.0777 - val_ac
curacy: 0.9820
Epoch 29/30
400/400 [==============================] - 6s 14ms/step - loss: 0.0056 - accuracy: 0.9980 - val_loss: 0.0844 - val_ac
curacy: 0.9806
Epoch 30/30
400/400 [==============================] - 6s 14ms/step - loss: 0.0053 - accuracy: 0.9981 - val_loss: 0.0940 - val_ac
curacy: 0.9804
```

In [48]:

```
score = model2_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_te

# 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: 0.09398580342531204
Test accuracy: 0.980400025844574
```



In [50]:

```
for layers in model2_batch.layers:
    print("layer name",layers.name,",input shape",layers.input_shape,",output shape",layers.output_shape)
w_after = model2_batch.get_weights()
print(len(w_after))

for i in range(len(w_after)):
    print(w_after[i].shape)
```

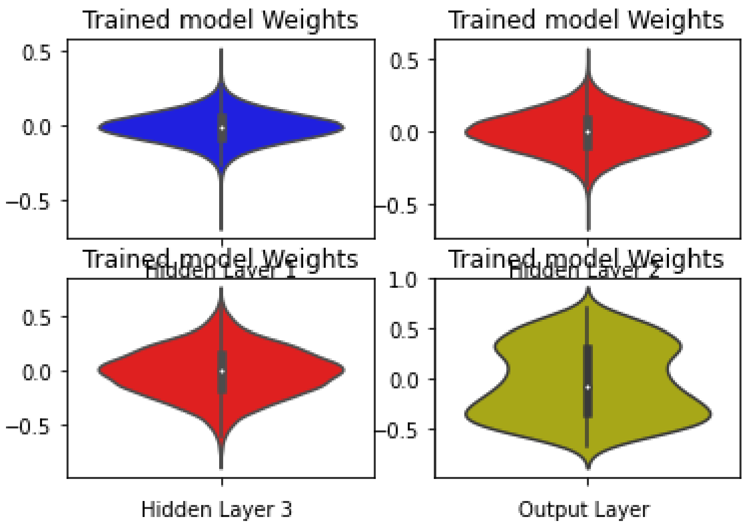
```
layer name dense_14 ,input shape (None, 784) ,output shape (None, 500)
layer name module_wrapper_14 ,input shape (None, 500) ,output shape (None, 500)
layer name dense_15 ,input shape (None, 500) ,output shape (None, 200)
layer name module_wrapper_15 ,input shape (None, 200) ,output shape (None, 200)
layer name dense_16 ,input shape (None, 200) ,output shape (None, 50)
layer name module_wrapper_16 ,input shape (None, 50) ,output shape (None, 50)
layer name dense_17 ,input shape (None, 50) ,output shape (None, 10)
20
(784, 500)
(500,)
(500,)
```

```
(500,)
(500,)
(500,)
(500, 200)
(200,)
(200,)
(200,)
(200,)
(200,)
(200, 50)
(50,)
(50,)
(50,)
(50,)
(50,)
(50, 10)
(10,)
```

In [51]:

```
w_after = model2_batch.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[6].flatten().reshape(-1,1)
h3_w = w_after[12].flatten().reshape(-1,1)
out_w = w_after[18].flatten().reshape(-1,1)


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(2, 2, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

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

plt.subplot(2, 2, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='r')
plt.xlabel('Hidden Layer 3 ')
```

```
plt.subplot(2, 2, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



2.3 dropout

In [52]:

```
from tensorflow.keras.layers import Dropout

model2_drop = Sequential()

model2_drop.add(Dense(500, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, std
model2_drop.add(BatchNormalization())
model2_drop.add(Dropout(0.5))

model2_drop.add(Dense(200, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.1, seed=None)) )
model2_drop.add(BatchNormalization())
model2_drop.add(Dropout(0.5))

model2_drop.add(Dense(50, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.2, seed=None)) )
model2_drop.add(BatchNormalization())
model2_drop.add(Dropout(0.5))
```

```
model2_drop.add(Dense(output_dim, activation='softmax'))


model2_drop.summary()
```

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_18 (Dense)             (None, 500)               392500
_________________________________________________________________
module_wrapper_17 (ModuleWra (None, 500)               2000
_________________________________________________________________
dropout_2 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_19 (Dense)             (None, 200)               100200
_________________________________________________________________
module_wrapper_18 (ModuleWra (None, 200)               800
_________________________________________________________________
dropout_3 (Dropout)          (None, 200)               0
_________________________________________________________________
dense_20 (Dense)             (None, 50)                10050
_________________________________________________________________
module_wrapper_19 (ModuleWra (None, 50)                200
_________________________________________________________________
dropout_4 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_21 (Dense)             (None, 10)                510
=================================================================
Total params: 506,260
Trainable params: 504,760
Non-trainable params: 1,500
_________________________________________________________________
```

In [53]:

```
model2_drop.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model2_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_tes
```

```
Epoch 1/30
400/400 [==============================] - 7s 16ms/step - loss: 0.8117 - accuracy: 0.7549 - val_loss: 0.2113 - val_ac
curacy: 0.9345
Epoch 2/30
400/400 [==============================] - 6s 15ms/step - loss: 0.3406 - accuracy: 0.9036 - val_loss: 0.1478 - val_ac
curacy: 0.9549
Epoch 3/30
```

```
400/400 [==============================] - 6s 15ms/step - loss: 0.2537 - accuracy: 0.9296 - val_loss: 0.1221 - val_ac
curacy: 0.9623
Epoch 4/30
400/400 [==============================] - 6s 15ms/step - loss: 0.2066 - accuracy: 0.9436 - val_loss: 0.1039 - val_ac
curacy: 0.9688
Epoch 5/30
400/400 [==============================] - 6s 15ms/step - loss: 0.1823 - accuracy: 0.9496 - val_loss: 0.0958 - val_ac
curacy: 0.9707
Epoch 6/30
400/400 [==============================] - 6s 15ms/step - loss: 0.1599 - accuracy: 0.9558 - val_loss: 0.0931 - val_ac
curacy: 0.9731
Epoch 7/30
400/400 [==============================] - 6s 15ms/step - loss: 0.1440 - accuracy: 0.9599 - val_loss: 0.0854 - val_ac
curacy: 0.9751
Epoch 8/30
400/400 [==============================] - 6s 16ms/step - loss: 0.1342 - accuracy: 0.9620 - val_loss: 0.0757 - val_ac
curacy: 0.9792
Epoch 9/30
400/400 [==============================] - 6s 16ms/step - loss: 0.1291 - accuracy: 0.9647 - val_loss: 0.0756 - val_ac
curacy: 0.9785
Epoch 10/30
400/400 [==============================] - 6s 15ms/step - loss: 0.1156 - accuracy: 0.9675 - val_loss: 0.0690 - val_ac
curacy: 0.9807
Epoch 11/30
400/400 [==============================] - 6s 15ms/step - loss: 0.1064 - accuracy: 0.9708 - val_loss: 0.0730 - val_ac
curacy: 0.9795
Epoch 12/30
400/400 [==============================] - 6s 15ms/step - loss: 0.1043 - accuracy: 0.9705 - val_loss: 0.0678 - val_ac
curacy: 0.9812
Epoch 13/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0960 - accuracy: 0.9727 - val_loss: 0.0732 - val_ac
curacy: 0.9805
Epoch 14/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0954 - accuracy: 0.9736 - val_loss: 0.0691 - val_ac
curacy: 0.9813
Epoch 15/30
400/400 [==============================] - 6s 16ms/step - loss: 0.0902 - accuracy: 0.9746 - val_loss: 0.0648 - val_ac
curacy: 0.9811
Epoch 16/30
400/400 [==============================] - 6s 16ms/step - loss: 0.0853 - accuracy: 0.9757 - val_loss: 0.0654 - val_ac
curacy: 0.9830
Epoch 17/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0835 - accuracy: 0.9768 - val_loss: 0.0697 - val_ac
curacy: 0.9808
Epoch 18/30
```

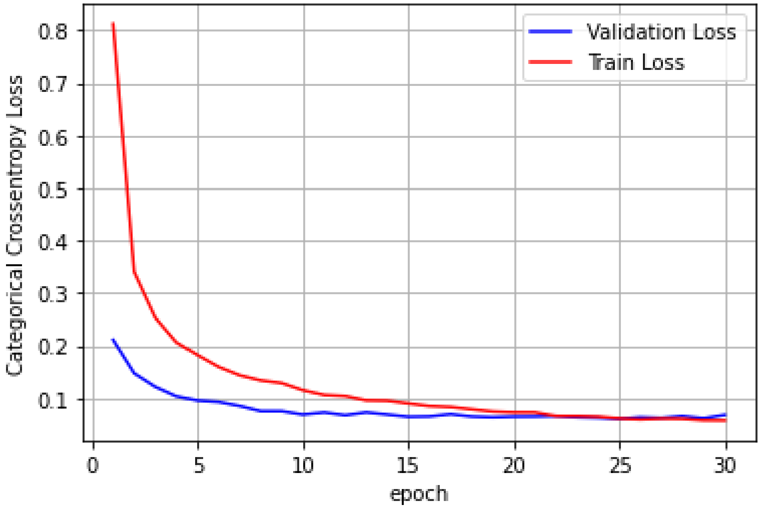
```
400/400 [==============================] - 6s 16ms/step - loss: 0.0791 - accuracy: 0.9771 - val_loss: 0.0652 - val_ac
curacy: 0.9821
Epoch 19/30
400/400 [==============================] - 6s 16ms/step - loss: 0.0749 - accuracy: 0.9785 - val_loss: 0.0639 - val_ac
curacy: 0.9836
Epoch 20/30
400/400 [==============================] - 6s 16ms/step - loss: 0.0733 - accuracy: 0.9791 - val_loss: 0.0650 - val_ac
curacy: 0.9831
Epoch 21/30
400/400 [==============================] - 6s 16ms/step - loss: 0.0732 - accuracy: 0.9790 - val_loss: 0.0653 - val_ac
curacy: 0.9832
Epoch 22/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0663 - accuracy: 0.9810 - val_loss: 0.0662 - val_ac
curacy: 0.9830
Epoch 23/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0659 - accuracy: 0.9808 - val_loss: 0.0638 - val_ac
curacy: 0.9841
Epoch 24/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0649 - accuracy: 0.9812 - val_loss: 0.0628 - val_ac
curacy: 0.9830
Epoch 25/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0616 - accuracy: 0.9825 - val_loss: 0.0609 - val_ac
curacy: 0.9838
Epoch 26/30
400/400 [==============================] - 6s 16ms/step - loss: 0.0597 - accuracy: 0.9822 - val_loss: 0.0638 - val_ac
curacy: 0.9844
Epoch 27/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0614 - accuracy: 0.9826 - val_loss: 0.0625 - val_ac
curacy: 0.9843
Epoch 28/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0609 - accuracy: 0.9829 - val_loss: 0.0660 - val_ac
curacy: 0.9825
Epoch 29/30
400/400 [==============================] - 6s 14ms/step - loss: 0.0581 - accuracy: 0.9829 - val_loss: 0.0619 - val_ac
curacy: 0.9845
Epoch 30/30
400/400 [==============================] - 6s 15ms/step - loss: 0.0578 - accuracy: 0.9837 - val_loss: 0.0683 - val_ac
curacy: 0.9823
```

In [54]:

```
score = model2_drop.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

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

# 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: 0.06829633563756943
Test accuracy: 0.9822999835014343
```



In [56]:
```
for layers in model2_drop.layers:
    print("layer name",layers.name,",input shape",layers.input_shape,",output shape",layers.output_shape)
```

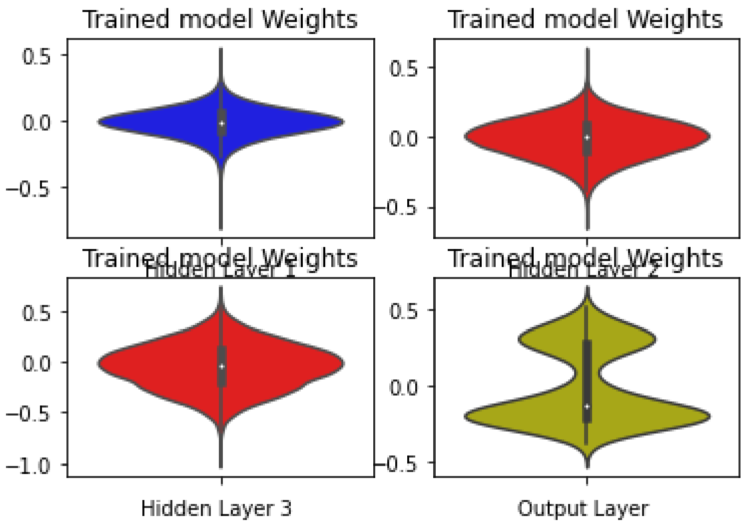
```
w_after = model2_drop.get_weights()
print(len(w_after))

for i in range(len(w_after)):
    print(w_after[i].shape)
```

```
layer name dense_18 ,input shape (None, 784) ,output shape (None, 500)
layer name module_wrapper_17 ,input shape (None, 500) ,output shape (None, 500)
layer name dropout_2 ,input shape (None, 500) ,output shape (None, 500)
layer name dense_19 ,input shape (None, 500) ,output shape (None, 200)
layer name module_wrapper_18 ,input shape (None, 200) ,output shape (None, 200)
layer name dropout_3 ,input shape (None, 200) ,output shape (None, 200)
layer name dense_20 ,input shape (None, 200) ,output shape (None, 50)
layer name module_wrapper_19 ,input shape (None, 50) ,output shape (None, 50)
layer name dropout_4 ,input shape (None, 50) ,output shape (None, 50)
layer name dense_21 ,input shape (None, 50) ,output shape (None, 10)
20
(784, 500)
(500,)
(500,)
(500,)
(500,)
(500,)
(500, 200)
(200,)
(200,)
(200,)
(200,)
(200,)
(200, 50)
(50,)
(50,)
(50,)
(50,)
(50,)
(50, 10)
(10,)
```

In [57]:

```
w_after =  model2_drop.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[6].flatten().reshape(-1,1)
h3_w = w_after[12].flatten().reshape(-1,1)
out_w = w_after[18].flatten().reshape(-1,1)
```

```
fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(2, 2, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

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

plt.subplot(2, 2, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='r')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(2, 2, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



3.Five hidden layer Architecture

3.1 Five hidden layer with Relu + Adam

In [58]:
```
batch_size = 100
nb_epoch = 20
```

In [59]:
```
model3_relu = Sequential()
model3_relu.add(Dense(600, activation='relu', input_shape=(input_dim,)))
model3_relu.add(Dense(300, activation='relu'))
model3_relu.add(Dense(150, activation='relu'))
model3_relu.add(Dense(75, activation='relu'))
model3_relu.add(Dense(30, activation='relu'))
model3_relu.add(Dense(output_dim, activation='softmax'))
model3_relu.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_22 (Dense)             (None, 600)               471000    
_________________________________________________________________
dense_23 (Dense)             (None, 300)               180300    
_________________________________________________________________
dense_24 (Dense)             (None, 150)               45150     
_________________________________________________________________
dense_25 (Dense)             (None, 75)                11325     
_________________________________________________________________
dense_26 (Dense)             (None, 30)                2280      
_________________________________________________________________
dense_27 (Dense)             (None, 10)                310       
=================================================================
Total params: 710,365
Trainable params: 710,365
Non-trainable params: 0
_________________________________________________________________
```

In [60]:
```
model3_relu.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model3_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_tes
```

```
Epoch 1/20
```

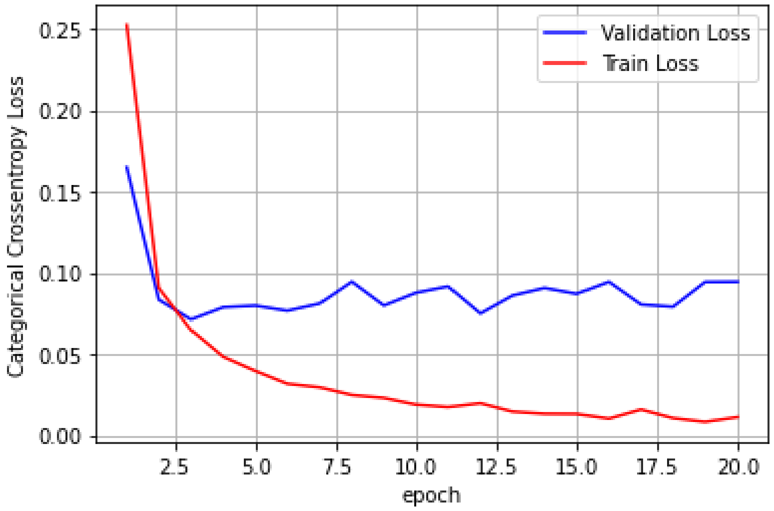
```
600/600 [==============================] - 9s 14ms/step - loss: 0.2523 - accuracy: 0.9239 - val_loss: 0.1648 - val_ac
curacy: 0.9506
Epoch 2/20
600/600 [==============================] - 8s 13ms/step - loss: 0.0906 - accuracy: 0.9718 - val_loss: 0.0835 - val_ac
curacy: 0.9744
Epoch 3/20
600/600 [==============================] - 7s 12ms/step - loss: 0.0649 - accuracy: 0.9796 - val_loss: 0.0714 - val_ac
curacy: 0.9774
Epoch 4/20
600/600 [==============================] - 7s 12ms/step - loss: 0.0484 - accuracy: 0.9846 - val_loss: 0.0788 - val_ac
curacy: 0.9772
Epoch 5/20
600/600 [==============================] - 7s 12ms/step - loss: 0.0397 - accuracy: 0.9876 - val_loss: 0.0798 - val_ac
curacy: 0.9791
Epoch 6/20
600/600 [==============================] - 8s 13ms/step - loss: 0.0318 - accuracy: 0.9901 - val_loss: 0.0766 - val_ac
curacy: 0.9798
Epoch 7/20
600/600 [==============================] - 8s 13ms/step - loss: 0.0296 - accuracy: 0.9904 - val_loss: 0.0810 - val_ac
curacy: 0.9787
Epoch 8/20
600/600 [==============================] - 7s 12ms/step - loss: 0.0249 - accuracy: 0.9920 - val_loss: 0.0944 - val_ac
curacy: 0.9769
Epoch 9/20
600/600 [==============================] - 8s 13ms/step - loss: 0.0230 - accuracy: 0.9927 - val_loss: 0.0798 - val_ac
curacy: 0.9803
Epoch 10/20
600/600 [==============================] - 8s 13ms/step - loss: 0.0189 - accuracy: 0.9943 - val_loss: 0.0876 - val_ac
curacy: 0.9792
Epoch 11/20
600/600 [==============================] - 8s 13ms/step - loss: 0.0174 - accuracy: 0.9950 - val_loss: 0.0915 - val_ac
curacy: 0.9771
Epoch 12/20
600/600 [==============================] - 7s 12ms/step - loss: 0.0198 - accuracy: 0.9941 - val_loss: 0.0749 - val_ac
curacy: 0.9816
Epoch 13/20
600/600 [==============================] - 8s 13ms/step - loss: 0.0146 - accuracy: 0.9959 - val_loss: 0.0860 - val_ac
curacy: 0.9826
Epoch 14/20
600/600 [==============================] - 7s 12ms/step - loss: 0.0133 - accuracy: 0.9958 - val_loss: 0.0906 - val_ac
curacy: 0.9797
Epoch 15/20
600/600 [==============================] - 8s 13ms/step - loss: 0.0132 - accuracy: 0.9963 - val_loss: 0.0870 - val_ac
curacy: 0.9803
Epoch 16/20
```

```
600/600 [==============================] - 8s 13ms/step - loss: 0.0103 - accuracy: 0.9968 - val_loss: 0.0944 - val_ac
curacy: 0.9822
Epoch 17/20
600/600 [==============================] - 7s 12ms/step - loss: 0.0159 - accuracy: 0.9952 - val_loss: 0.0804 - val_ac
curacy: 0.9835
Epoch 18/20
600/600 [==============================] - 8s 13ms/step - loss: 0.0106 - accuracy: 0.9969 - val_loss: 0.0790 - val_ac
curacy: 0.9827
Epoch 19/20
600/600 [==============================] - 8s 13ms/step - loss: 0.0083 - accuracy: 0.9976 - val_loss: 0.0943 - val_ac
curacy: 0.9816
Epoch 20/20
600/600 [==============================] - 8s 13ms/step - loss: 0.0112 - accuracy: 0.9966 - val_loss: 0.0944 - val_ac
curacy: 0.9844
```

In [61]:

```
score = model3_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')


x = list(range(1,nb_epoch+1))



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

```
Test score: 0.09438823908567429
Test accuracy: 0.9843999743461609
```



In [62]:
```
for layers in model3_relu.layers:
    print("layer name",layers.name,",input shape",layers.input_shape,",output shape",layers.output_shape,",bias shape
w_after = model3_relu.get_weights()
print(len(w_after))

for i in range(len(w_after)):
    print(w_after[i].shape)
```

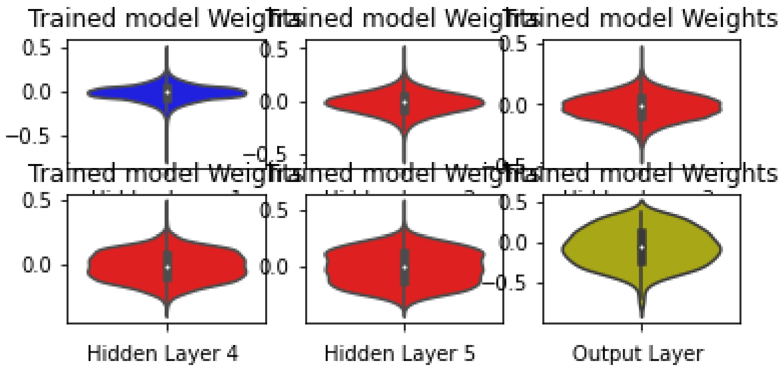
```
layer name dense_22 ,input shape (None, 784) ,output shape (None, 600) ,bias shape (600,)
layer name dense_23 ,input shape (None, 600) ,output shape (None, 300) ,bias shape (300,)
layer name dense_24 ,input shape (None, 300) ,output shape (None, 150) ,bias shape (150,)
layer name dense_25 ,input shape (None, 150) ,output shape (None, 75) ,bias shape (75,)
layer name dense_26 ,input shape (None, 75) ,output shape (None, 30) ,bias shape (30,)
layer name dense_27 ,input shape (None, 30) ,output shape (None, 10) ,bias shape (10,)
12
(784, 600)
(600,)
(600, 300)
(300,)
(300, 150)
(150,)
(150, 75)
(75,)
(75, 30)
(30,)
```

```
(30, 10)
(10,)
```

In [63]:

```
w_after = model3_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(3, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

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

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

plt.subplot(3, 3, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 4 ')

plt.subplot(3, 3, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='r')
plt.xlabel('Hidden Layer 5 ')

plt.subplot(3, 3, 6)
plt.title("Trained model Weights")
```

```
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



3.2 batch Normalization

In [66]:
```
model3_batch = Sequential()

model3_batch.add(Dense(600, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, st
model3_batch.add(BatchNormalization())

model3_batch.add(Dense(300, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.08, seed=None)) )
model3_batch.add(BatchNormalization())

model3_batch.add(Dense(150, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.115, seed=None)) )
model3_batch.add(BatchNormalization())

model3_batch.add(Dense(75, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.163, seed=None)) )
model3_batch.add(BatchNormalization())

model3_batch.add(Dense(30, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.258, seed=None)) )
model3_batch.add(BatchNormalization())

model3_batch.add(Dense(output_dim, activation='softmax'))


model3_batch.summary()
```

```
Model: "sequential_13"
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_29 (Dense)             (None, 600)               471000    
_________________________________________________________________
module_wrapper_20 (ModuleWra (None, 600)               2400      
_________________________________________________________________
dense_30 (Dense)             (None, 300)               180300    
_________________________________________________________________
module_wrapper_21 (ModuleWra (None, 300)               1200      
_________________________________________________________________
dense_31 (Dense)             (None, 150)               45150     
_________________________________________________________________
module_wrapper_22 (ModuleWra (None, 150)               600       
_________________________________________________________________
dense_32 (Dense)             (None, 75)                11325     
_________________________________________________________________
module_wrapper_23 (ModuleWra (None, 75)                300       
_________________________________________________________________
dense_33 (Dense)             (None, 30)                2280      
_________________________________________________________________
module_wrapper_24 (ModuleWra (None, 30)                120       
_________________________________________________________________
dense_34 (Dense)             (None, 10)                310       
=================================================================
Total params: 714,985
Trainable params: 712,675
Non-trainable params: 2,310
_________________________________________________________________
```

In [67]:

```
model3_batch.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model3_batch.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_te
```

```
Epoch 1/20
600/600 [==============================] - 10s 15ms/step - loss: 0.2466 - accuracy: 0.9336 - val_loss: 0.1189 - val_a
ccuracy: 0.9641
Epoch 2/20
600/600 [==============================] - 9s 15ms/step - loss: 0.0928 - accuracy: 0.9719 - val_loss: 0.0979 - val_ac
curacy: 0.9704
Epoch 3/20
600/600 [==============================] - 9s 16ms/step - loss: 0.0671 - accuracy: 0.9790 - val_loss: 0.0857 - val_ac
curacy: 0.9733
Epoch 4/20
600/600 [==============================] - 9s 15ms/step - loss: 0.0506 - accuracy: 0.9832 - val_loss: 0.0709 - val_ac
```

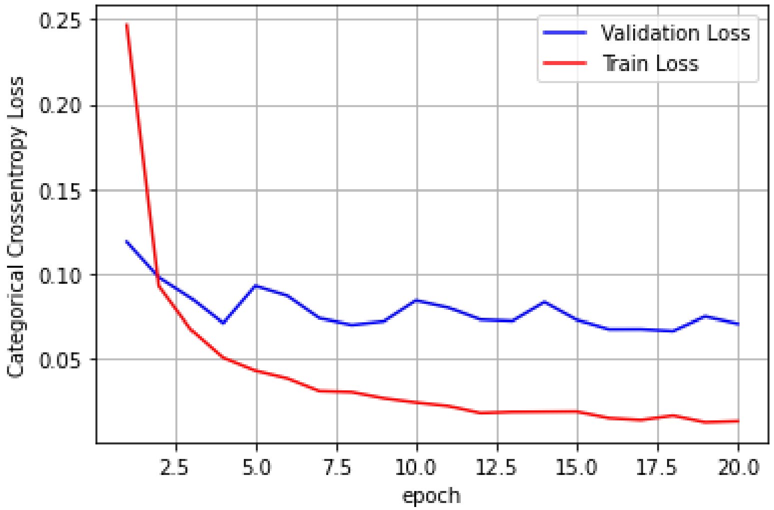
```
curacy: 0.9798
Epoch 5/20
600/600 [==============================] - 9s 16ms/step - loss: 0.0430 - accuracy: 0.9863 - val_loss: 0.0929 - val_ac
curacy: 0.9727
Epoch 6/20
600/600 [==============================] - 9s 16ms/step - loss: 0.0384 - accuracy: 0.9880 - val_loss: 0.0871 - val_ac
curacy: 0.9751
Epoch 7/20
600/600 [==============================] - 9s 15ms/step - loss: 0.0309 - accuracy: 0.9905 - val_loss: 0.0739 - val_ac
curacy: 0.9783
Epoch 8/20
600/600 [==============================] - 9s 15ms/step - loss: 0.0303 - accuracy: 0.9897 - val_loss: 0.0699 - val_ac
curacy: 0.9805
Epoch 9/20
600/600 [==============================] - 9s 15ms/step - loss: 0.0266 - accuracy: 0.9914 - val_loss: 0.0720 - val_ac
curacy: 0.9793
Epoch 10/20
600/600 [==============================] - 10s 16ms/step - loss: 0.0241 - accuracy: 0.9921 - val_loss: 0.0843 - val_a
ccuracy: 0.9776
Epoch 11/20
600/600 [==============================] - 9s 15ms/step - loss: 0.0221 - accuracy: 0.9931 - val_loss: 0.0801 - val_ac
curacy: 0.9788
Epoch 12/20
600/600 [==============================] - 9s 15ms/step - loss: 0.0179 - accuracy: 0.9944 - val_loss: 0.0731 - val_ac
curacy: 0.9799
Epoch 13/20
600/600 [==============================] - 9s 15ms/step - loss: 0.0186 - accuracy: 0.9942 - val_loss: 0.0722 - val_ac
curacy: 0.9804
Epoch 14/20
600/600 [==============================] - 9s 15ms/step - loss: 0.0187 - accuracy: 0.9938 - val_loss: 0.0834 - val_ac
curacy: 0.9792
Epoch 15/20
600/600 [==============================] - 9s 15ms/step - loss: 0.0188 - accuracy: 0.9940 - val_loss: 0.0729 - val_ac
curacy: 0.9807
Epoch 16/20
600/600 [==============================] - 9s 15ms/step - loss: 0.0149 - accuracy: 0.9952 - val_loss: 0.0674 - val_ac
curacy: 0.9844
Epoch 17/20
600/600 [==============================] - 9s 16ms/step - loss: 0.0138 - accuracy: 0.9954 - val_loss: 0.0674 - val_ac
curacy: 0.9814
Epoch 18/20
600/600 [==============================] - 9s 16ms/step - loss: 0.0164 - accuracy: 0.9946 - val_loss: 0.0663 - val_ac
curacy: 0.9815
Epoch 19/20
600/600 [==============================] - 9s 16ms/step - loss: 0.0125 - accuracy: 0.9959 - val_loss: 0.0749 - val_ac
```

```
curacy: 0.9816
Epoch 20/20
600/600 [==============================] - 9s 16ms/step - loss: 0.0131 - accuracy: 0.9959 - val_loss: 0.0706 - val_ac
curacy: 0.9830
```

In [68]:

```
score = model3_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_te

# 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: 0.07062286138534546
Test accuracy: 0.9829999804496765
```



In [70]:
```
for layers in model3_batch.layers:
    print("layer name",layers.name,",input shape",layers.input_shape,",output shape",layers.output_shape)
w_after = model3_batch.get_weights()
print(len(w_after))

for i in range(len(w_after)):
    print(w_after[i].shape)
```

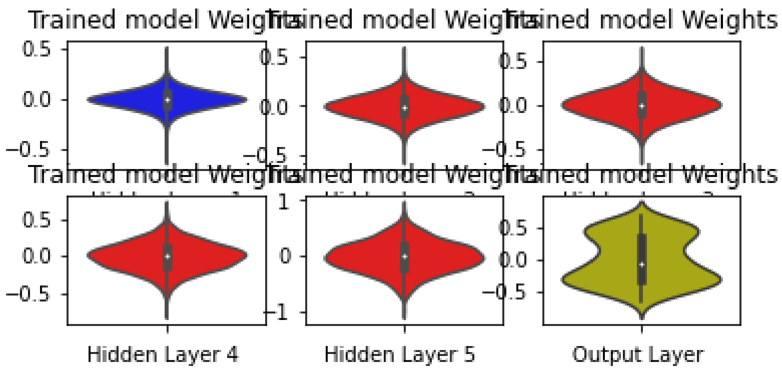
```
layer name dense_29 ,input shape (None, 784) ,output shape (None, 600)
layer name module_wrapper_20 ,input shape (None, 600) ,output shape (None, 600)
layer name dense_30 ,input shape (None, 600) ,output shape (None, 300)
layer name module_wrapper_21 ,input shape (None, 300) ,output shape (None, 300)
layer name dense_31 ,input shape (None, 300) ,output shape (None, 150)
layer name module_wrapper_22 ,input shape (None, 150) ,output shape (None, 150)
layer name dense_32 ,input shape (None, 150) ,output shape (None, 75)
layer name module_wrapper_23 ,input shape (None, 75) ,output shape (None, 75)
layer name dense_33 ,input shape (None, 75) ,output shape (None, 30)
layer name module_wrapper_24 ,input shape (None, 30) ,output shape (None, 30)
layer name dense_34 ,input shape (None, 30) ,output shape (None, 10)
32
(784, 600)
(600,)
(600,)
(600,)
(600,)
(600,)
```

```
(600, 300)
(300,)
(300,)
(300,)
(300,)
(300,)
(300, 150)
(150,)
(150,)
(150,)
(150,)
(150,)
(150, 75)
(75,)
(75,)
(75,)
(75,)
(75,)
(75, 30)
(30,)
(30,)
(30,)
(30,)
(30,)
(30, 10)
(10,)
```

In [72]:

```
w_after = model3_batch.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[6].flatten().reshape(-1,1)
h3_w = w_after[12].flatten().reshape(-1,1)
h4_w = w_after[18].flatten().reshape(-1,1)
h5_w = w_after[24].flatten().reshape(-1,1)
out_w = w_after[30].flatten().reshape(-1,1)


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

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

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

plt.subplot(3, 3,4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 4 ')

plt.subplot(3, 3, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='r')
plt.xlabel('Hidden Layer 5 ')

plt.subplot(3, 3, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



3.3 drop out

```
from tensorflow.keras.layers import Dropout
```

In [73]:

```
model3_drop = Sequential()

model3_drop.add(Dense(600, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, std
model3_drop.add(BatchNormalization())
model3_drop.add(Dropout(0.5))

model3_drop.add(Dense(300, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.08, seed=None)) )
model3_drop.add(BatchNormalization())
model3_drop.add(Dropout(0.5))

model3_drop.add(Dense(150, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.115, seed=None)) )
model3_drop.add(BatchNormalization())
model3_drop.add(Dropout(0.5))

model3_drop.add(Dense(75, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.163, seed=None)) )
model3_drop.add(BatchNormalization())
model3_drop.add(Dropout(0.5))

model3_drop.add(Dense(30, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.258, seed=None)) )
model3_drop.add(BatchNormalization())
model3_drop.add(Dropout(0.5))

model3_drop.add(Dense(output_dim, activation='softmax'))


model3_drop.summary()
```

```
Model: "sequential_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_35 (Dense)             (None, 600)               471000
_________________________________________________________________
module_wrapper_25 (ModuleWra (None, 600)               2400
_________________________________________________________________
dropout_5 (Dropout)          (None, 600)               0
_________________________________________________________________
dense_36 (Dense)             (None, 300)               180300
_________________________________________________________________
module_wrapper_26 (ModuleWra (None, 300)               1200
_________________________________________________________________
dropout_6 (Dropout)          (None, 300)               0
```

```
dense_37 (Dense)             (None, 150)               45150
_________________________________________________________________
module_wrapper_27 (ModuleWra (None, 150)               600
_________________________________________________________________
dropout_7 (Dropout)          (None, 150)               0
_________________________________________________________________
dense_38 (Dense)             (None, 75)                11325
_________________________________________________________________
module_wrapper_28 (ModuleWra (None, 75)                300
_________________________________________________________________
dropout_8 (Dropout)          (None, 75)                0
_________________________________________________________________
dense_39 (Dense)             (None, 30)                2280
_________________________________________________________________
module_wrapper_29 (ModuleWra (None, 30)                120
_________________________________________________________________
dropout_9 (Dropout)          (None, 30)                0
_________________________________________________________________
dense_40 (Dense)             (None, 10)                310
=================================================================
Total params: 714,985
Trainable params: 712,675
Non-trainable params: 2,310
_________________________________________________________________
```

In [74]:

```
model3_drop.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model3_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_tes
```

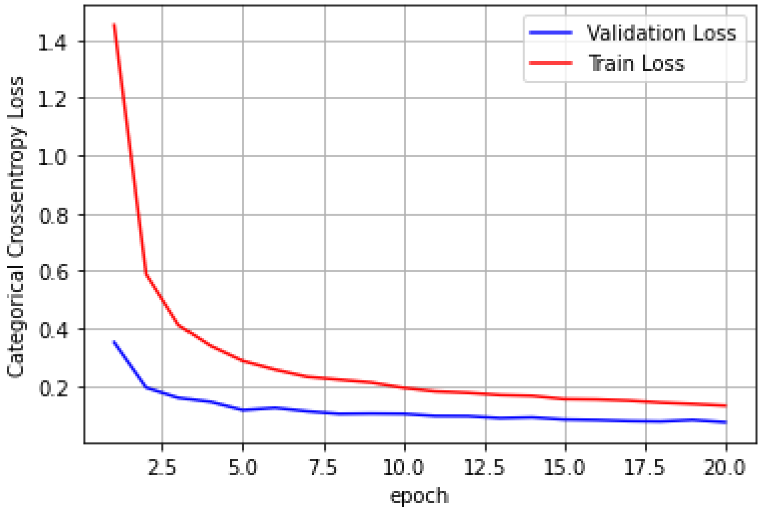
```
Epoch 1/20
600/600 [==============================] - 12s 17ms/step - loss: 1.4529 - accuracy: 0.5181 - val_loss: 0.3517 - val_a
ccuracy: 0.9005
Epoch 2/20
600/600 [==============================] - 10s 17ms/step - loss: 0.5887 - accuracy: 0.8265 - val_loss: 0.1942 - val_a
ccuracy: 0.9475
Epoch 3/20
600/600 [==============================] - 10s 17ms/step - loss: 0.4119 - accuracy: 0.8928 - val_loss: 0.1587 - val_a
ccuracy: 0.9571
Epoch 4/20
600/600 [==============================] - 10s 17ms/step - loss: 0.3392 - accuracy: 0.9167 - val_loss: 0.1446 - val_a
ccuracy: 0.9627
Epoch 5/20
600/600 [==============================] - 10s 17ms/step - loss: 0.2871 - accuracy: 0.9305 - val_loss: 0.1158 - val_a
ccuracy: 0.9700
```

```
Epoch 6/20
600/600 [==============================] - 10s 17ms/step - loss: 0.2568 - accuracy: 0.9398 - val_loss: 0.1238 - val_a
ccuracy: 0.9696
Epoch 7/20
600/600 [==============================] - 10s 17ms/step - loss: 0.2320 - accuracy: 0.9455 - val_loss: 0.1117 - val_a
ccuracy: 0.9732
Epoch 8/20
600/600 [==============================] - 10s 17ms/step - loss: 0.2219 - accuracy: 0.9477 - val_loss: 0.1028 - val_a
ccuracy: 0.9759
Epoch 9/20
600/600 [==============================] - 10s 17ms/step - loss: 0.2121 - accuracy: 0.9515 - val_loss: 0.1039 - val_a
ccuracy: 0.9760
Epoch 10/20
600/600 [==============================] - 10s 17ms/step - loss: 0.1937 - accuracy: 0.9563 - val_loss: 0.1031 - val_a
ccuracy: 0.9771
Epoch 11/20
600/600 [==============================] - 10s 16ms/step - loss: 0.1811 - accuracy: 0.9588 - val_loss: 0.0954 - val_a
ccuracy: 0.9775
Epoch 12/20
600/600 [==============================] - 10s 17ms/step - loss: 0.1762 - accuracy: 0.9614 - val_loss: 0.0949 - val_a
ccuracy: 0.9767
Epoch 13/20
600/600 [==============================] - 10s 17ms/step - loss: 0.1684 - accuracy: 0.9622 - val_loss: 0.0879 - val_a
ccuracy: 0.9805
Epoch 14/20
600/600 [==============================] - 10s 17ms/step - loss: 0.1656 - accuracy: 0.9623 - val_loss: 0.0905 - val_a
ccuracy: 0.9789
Epoch 15/20
600/600 [==============================] - 10s 17ms/step - loss: 0.1544 - accuracy: 0.9659 - val_loss: 0.0831 - val_a
ccuracy: 0.9806
Epoch 16/20
600/600 [==============================] - 10s 17ms/step - loss: 0.1530 - accuracy: 0.9656 - val_loss: 0.0813 - val_a
ccuracy: 0.9812
Epoch 17/20
600/600 [==============================] - 10s 17ms/step - loss: 0.1492 - accuracy: 0.9669 - val_loss: 0.0779 - val_a
ccuracy: 0.9829
Epoch 18/20
600/600 [==============================] - 10s 17ms/step - loss: 0.1421 - accuracy: 0.9681 - val_loss: 0.0764 - val_a
ccuracy: 0.9818
Epoch 19/20
600/600 [==============================] - 10s 17ms/step - loss: 0.1374 - accuracy: 0.9689 - val_loss: 0.0816 - val_a
ccuracy: 0.9821
Epoch 20/20
600/600 [==============================] - 10s 17ms/step - loss: 0.1314 - accuracy: 0.9700 - val_loss: 0.0738 - val_a
ccuracy: 0.9837
```

In [75]:

```
score = model3_drop.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

# 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: 0.07375040650367737
Test accuracy: 0.9836999773979187
```



In [77]:

```
for layers in model3_drop.layers:
    print("layer name",layers.name,",input shape",layers.input_shape,",output shape",layers.output_shape)
w_after = model3_drop.get_weights()
print(len(w_after))

for i in range(len(w_after)):
    print(w_after[i].shape)
```

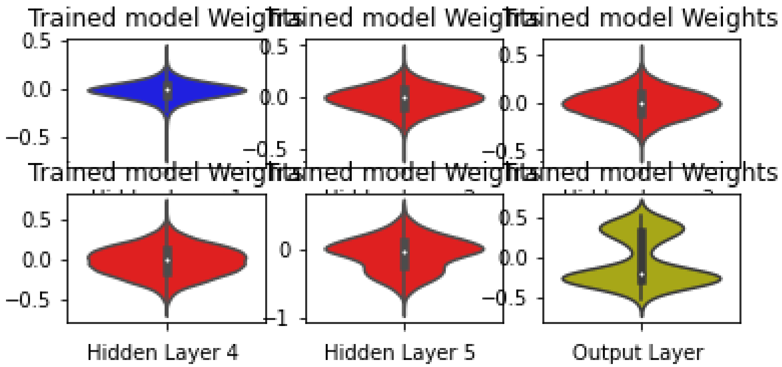
```
layer name dense_35 ,input shape (None, 784) ,output shape (None, 600)
layer name module_wrapper_25 ,input shape (None, 600) ,output shape (None, 600)
layer name dropout_5 ,input shape (None, 600) ,output shape (None, 600)
layer name dense_36 ,input shape (None, 600) ,output shape (None, 300)
layer name module_wrapper_26 ,input shape (None, 300) ,output shape (None, 300)
layer name dropout_6 ,input shape (None, 300) ,output shape (None, 300)
layer name dense_37 ,input shape (None, 300) ,output shape (None, 150)
layer name module_wrapper_27 ,input shape (None, 150) ,output shape (None, 150)
layer name dropout_7 ,input shape (None, 150) ,output shape (None, 150)
layer name dense_38 ,input shape (None, 150) ,output shape (None, 75)
layer name module_wrapper_28 ,input shape (None, 75) ,output shape (None, 75)
layer name dropout_8 ,input shape (None, 75) ,output shape (None, 75)
layer name dense_39 ,input shape (None, 75) ,output shape (None, 30)
layer name module_wrapper_29 ,input shape (None, 30) ,output shape (None, 30)
layer name dropout_9 ,input shape (None, 30) ,output shape (None, 30)
layer name dense_40 ,input shape (None, 30) ,output shape (None, 10)
32
(784, 600)
```

```
(600,)
(600,)
(600,)
(600,)
(600,)
(600, 300)
(300,)
(300,)
(300,)
(300,)
(300,)
(300, 150)
(150,)
(150,)
(150,)
(150,)
(150,)
(150, 75)
(75,)
(75,)
(75,)
(75,)
(75,)
(75, 30)
(30,)
(30,)
(30,)
(30,)
(30,)
(30, 10)
(10,)
```

In [78]:

```
w_after = model3_drop.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[6].flatten().reshape(-1,1)
h3_w = w_after[12].flatten().reshape(-1,1)
h4_w = w_after[18].flatten().reshape(-1,1)
h5_w = w_after[24].flatten().reshape(-1,1)
out_w = w_after[30].flatten().reshape(-1,1)


fig = plt.figure()
plt.title("Weight matrices after model trained")
```

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

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

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

plt.subplot(3, 3,4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 4 ')

plt.subplot(3, 3, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='r')
plt.xlabel('Hidden Layer 5 ')

plt.subplot(3, 3, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



4.observation table

In [81]:

```
from prettytable import PrettyTable
x = PrettyTable()

x = PrettyTable()
x.field_names = ["layers", "Model", "epoch", "batch size","test score","accuracy"]
x.add_row(["two layers", "Relu+adam optimizer", "20" ,"128","0.09","98"])
x.add_row(["two layers", "Relu+Batch Nor.+adam optimizer", "20" ,"128","0.08","97"])
x.add_row(["two layers", "Relu+Batch Nor+dropout+adam optimizer", "20" ,"128","0.06","98"])

x.add_row(["three layers", "Relu+adam optimizer", "30" ,"150","0.10","98"])
x.add_row(["three layers", "Relu+Batch Nor.+adam optimizer", "30" ,"150","0.09","98"])
x.add_row(["three layers", "Relu+Batch Nor+dropout+adam optimizer", "30" ,"150","0.06","98"])

x.add_row(["five layers", "Relu+adam optimizer", "20" ,"100","0.09","98"])
x.add_row(["five layers", "Relu+Batch Nor.+adam optimizer", "20" ,"100","0.07","98"])
x.add_row(["five layers", "Relu+Batch Nor+dropout+adam optimizer", "20" ,"100","0.07","98"])


print(x)
```

```
+--------------+---------------------------------------+-------+------------+------------+----------+
|    layers    |                 Model                 | epoch | batch size | test score | accuracy |
+--------------+---------------------------------------+-------+------------+------------+----------+
|  two layers  |          Relu+adam optimizer          |   20  |    128     |    0.09    |    98    |
|  two layers  |     Relu+Batch Nor.+adam optimizer    |   20  |    128     |    0.08    |    97    |
|  two layers  | Relu+Batch Nor+dropout+adam optimizer |   20  |    128     |    0.06    |    98    |
```

three layers	Relu+adam optimizer	30	150	0.10	98
three layers	Relu+Batch Nor.+adam optimizer	30	150	0.09	98
three layers	Relu+Batch Nor+dropout+adam optimizer	30	150	0.06	98
five layers	Relu+adam optimizer	20	100	0.09	98
five layers	Relu+Batch Nor.+adam optimizer	20	100	0.07	98
five layers	Relu+Batch Nor+dropout+adam optimizer	20	100	0.07	98

5.Conclusion

1.As the number of layers increse We face overfit proble

2.Model give better performance for batch narmalization and drop out