Keras -- MLPs on MNIST

In [80]:

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

In [81]:

```
from keras.initializers import glorot_normal
from keras.initializers import he_normal
```

In [82]:

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

In [83]:

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

In [84]:

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

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

In [85]:

```
# 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 [86]:

```
# 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 test 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 test examples : 10000 and each image is of shape (784)
```

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

In [87]:

```
# 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 [88]:

```
# 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 [89]:

```
# 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 [90]:

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

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

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

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

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

Softmax classifier

In [91]:

```
# https://keras.io/getting-started/sequential-model-guide/

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

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

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

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

###

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

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

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

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

####

# https://keras.io/activations/

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

# from keras.layers import Activation, Dense

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

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

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


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

In [92]:

```
# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 20
```

In [93]:

```
# start building a model
model = Sequential()

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

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

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

In [94]:

```
# Before training a model, you need to configure the learning process, which is done via the compi
le method

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


# Note: when using the categorical_crossentropy loss, your targets should be in categorical format

# (e.g. if you have 10 classes, the target for each sample should be a 10-dimensional vector that
is all-zeros except
# for a 1 at the index corresponding to the class of the sample).

# that is why we converted out labels into vectors

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

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

# fit(self, x=None, y=None, batch_size=None, epochs=1, verbose=1, callbacks=None,
validation_split=0.0,
# validation_data=None, shuffle=True, class_weight=None, sample_weight=None, initial_epoch=0, step
s_per_epoch=None,
# validation_steps=None)

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

# it returns A History object. Its History.history attribute is a record of training loss values a
nd
# metrics values at successive epochs, as well as validation loss values and validation metrics va
lues (if applicable).

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

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

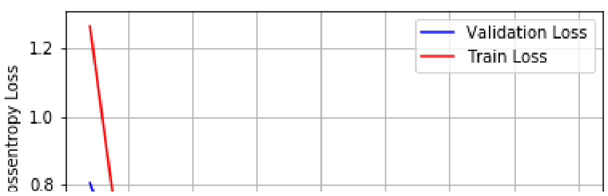
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 111us/step - loss: 1.2659 - acc: 0.7094 -
val_loss: 0.8067 - val_acc: 0.8280
Epoch 2/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.7120 - acc: 0.8417 -
val_loss: 0.6061 - val_acc: 0.8616
Epoch 3/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.5854 - acc: 0.8597 -
val_loss: 0.5252 - val_acc: 0.8740
Epoch 4/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.5244 - acc: 0.8694 -
val_loss: 0.4799 - val_acc: 0.8815
Epoch 5/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.4871 - acc: 0.8753 -
val_loss: 0.4507 - val_acc: 0.8866
Epoch 6/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.4614 - acc: 0.8796 -
val_loss: 0.4292 - val_acc: 0.8889
Epoch 7/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.4423 - acc: 0.8833 -
val_loss: 0.4131 - val_acc: 0.8920
Epoch 8/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.4274 - acc: 0.8864 -
val_loss: 0.4002 - val_acc: 0.8949
Epoch 9/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.4154 - acc: 0.8886 -
val_loss: 0.3901 - val_acc: 0.8962
```

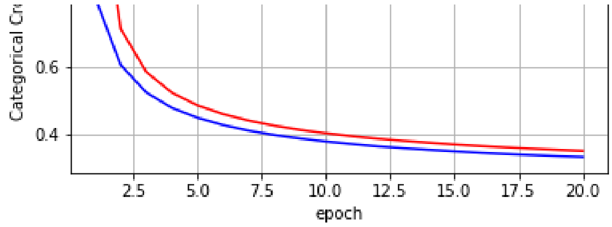
```
Epoch 10/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.4054 - acc: 0.8904 -
val_loss: 0.3811 - val_acc: 0.8976
Epoch 11/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.3969 - acc: 0.8922 -
val_loss: 0.3739 - val_acc: 0.9004
Epoch 12/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.3896 - acc: 0.8939 -
val_loss: 0.3671 - val_acc: 0.9017
Epoch 13/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.3833 - acc: 0.8952 -
val_loss: 0.3617 - val_acc: 0.9027
Epoch 14/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.3777 - acc: 0.8965 -
val_loss: 0.3568 - val_acc: 0.9033
Epoch 15/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.3726 - acc: 0.8975 -
val_loss: 0.3524 - val_acc: 0.9046
Epoch 16/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.3681 - acc: 0.8986 -
val_loss: 0.3481 - val_acc: 0.9047
Epoch 17/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.3640 - acc: 0.8994 -
val_loss: 0.3447 - val_acc: 0.9052
Epoch 18/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.3603 - acc: 0.9003 -
val_loss: 0.3414 - val_acc: 0.9069
Epoch 19/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.3568 - acc: 0.9013 -
val_loss: 0.3383 - val_acc: 0.9076
Epoch 20/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.3537 - acc: 0.9020 -
val_loss: 0.3357 - val_acc: 0.9086
```

In [95]:

```python
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, va
lidation_data=(X_test, Y_test))

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

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

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

```
Test score: 0.3357250750780106
Test accuracy: 0.9086
```





MLP + Sigmoid activation + SGDOptimizer

In [96]:

```
# Multilayer perceptron

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

model_sigmoid.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_103 (Dense)            (None, 512)               401920    
_________________________________________________________________
dense_104 (Dense)            (None, 128)               65664     
_________________________________________________________________
dense_105 (Dense)            (None, 10)                1290      
=================================================================
Total params: 468,874
Trainable params: 468,874
Non-trainable params: 0
_________________________________________________________________
```

In [97]:

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

history = model_sigmoid.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, va
lidation_data=(X_test, Y_test))
```

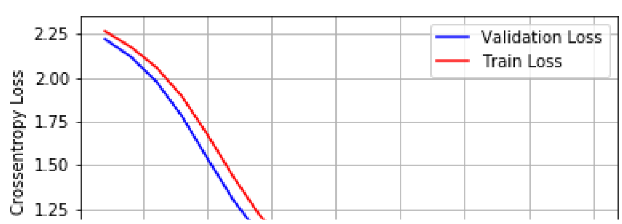
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 116us/step - loss: 2.2665 - acc: 0.2311 -
val_loss: 2.2210 - val_acc: 0.4829
Epoch 2/20
60000/60000 [==============================] - 3s 46us/step - loss: 2.1767 - acc: 0.4668 -
val_loss: 2.1211 - val_acc: 0.5636
Epoch 3/20
60000/60000 [==============================] - 3s 47us/step - loss: 2.0609 - acc: 0.5972 -
val_loss: 1.9797 - val_acc: 0.6697
Epoch 4/20
60000/60000 [==============================] - 3s 46us/step - loss: 1.8947 - acc: 0.6608 -
val_loss: 1.7812 - val_acc: 0.6553
Epoch 5/20
60000/60000 [==============================] - 3s 47us/step - loss: 1.6747 - acc: 0.6968 -
val_loss: 1.5382 - val_acc: 0.7561
Epoch 6/20
60000/60000 [==============================] - 3s 47us/step - loss: 1.4344 - acc: 0.7361 -
val_loss: 1.3019 - val_acc: 0.7611
Epoch 7/20
60000/60000 [==============================] - 3s 46us/step - loss: 1.2204 - acc: 0.7636 -
val_loss: 1.1097 - val_acc: 0.7879
Epoch 8/20
60000/60000 [==============================] - 3s 46us/step - loss: 1.0534 - acc: 0.7845 -
val_loss: 0.9659 - val_acc: 0.7938
Epoch 9/20
60000/60000 [==============================] - 3s 46us/step - loss: 0.9287 - acc: 0.7973 -
val_loss: 0.8579 - val_acc: 0.8082
Epoch 10/20
```

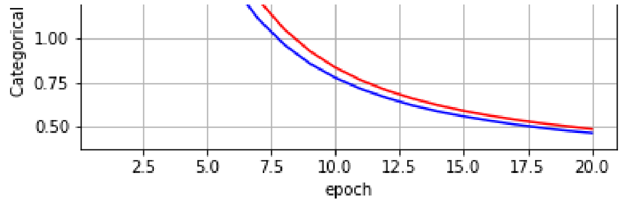
```
60000/60000 [==============================] - 3s 45us/step - loss: 0.8347 - acc: 0.8097 -
val_loss: 0.7763 - val_acc: 0.8234
Epoch 11/20
60000/60000 [==============================] - 3s 46us/step - loss: 0.7623 - acc: 0.8208 -
val_loss: 0.7125 - val_acc: 0.8313
Epoch 12/20
60000/60000 [==============================] - 3s 46us/step - loss: 0.7049 - acc: 0.8292 -
val_loss: 0.6622 - val_acc: 0.8421
Epoch 13/20
60000/60000 [==============================] - 3s 45us/step - loss: 0.6585 - acc: 0.8369 -
val_loss: 0.6196 - val_acc: 0.8463
Epoch 14/20
60000/60000 [==============================] - 3s 46us/step - loss: 0.6206 - acc: 0.8437 -
val_loss: 0.5854 - val_acc: 0.8509
Epoch 15/20
60000/60000 [==============================] - 3s 46us/step - loss: 0.5888 - acc: 0.8493 -
val_loss: 0.5572 - val_acc: 0.8578
Epoch 16/20
60000/60000 [==============================] - 3s 45us/step - loss: 0.5619 - acc: 0.8549 -
val_loss: 0.5329 - val_acc: 0.8620
Epoch 17/20
60000/60000 [==============================] - 3s 45us/step - loss: 0.5388 - acc: 0.8593 -
val_loss: 0.5110 - val_acc: 0.8666
Epoch 18/20
60000/60000 [==============================] - 3s 46us/step - loss: 0.5187 - acc: 0.8634 -
val_loss: 0.4927 - val_acc: 0.8702
Epoch 19/20
60000/60000 [==============================] - 3s 46us/step - loss: 0.5012 - acc: 0.8671 -
val_loss: 0.4761 - val_acc: 0.8727
Epoch 20/20
60000/60000 [==============================] - 3s 46us/step - loss: 0.4856 - acc: 0.8700 -
val_loss: 0.4625 - val_acc: 0.8764
```

In [98]:

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

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

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

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

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

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

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

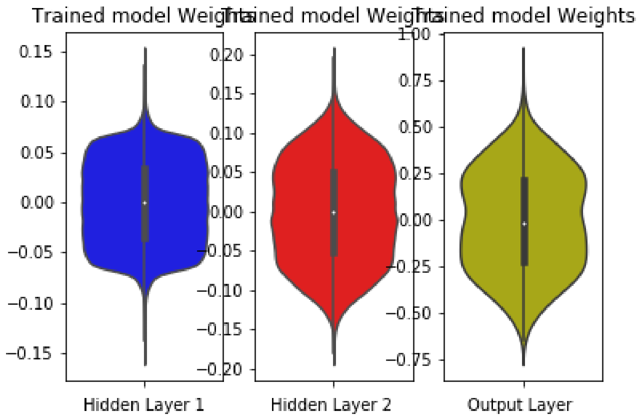
```
Test score: 0.4624546233654022
Test accuracy: 0.8764
```





In [99]:

```
w_after = model_sigmoid.get_weights()

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


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

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

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



MLP + Sigmoid activation + ADAM

In [100]:

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

model_sigmoid.summary()

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

history = model_sigmoid.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, va
lidation_data=(X_test, Y_test))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

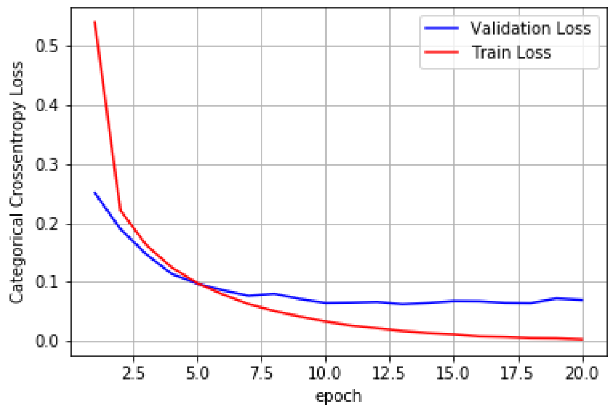
```
=================================================================
dense_106 (Dense)            (None, 512)               401920
_________________________________________________________________
dense_107 (Dense)            (None, 128)               65664
_________________________________________________________________
dense_108 (Dense)            (None, 10)                1290
=================================================================
Total params: 468,874
Trainable params: 468,874
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 8s 125us/step - loss: 0.5394 - acc: 0.8597 -
val_loss: 0.2513 - val_acc: 0.9265
Epoch 2/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.2212 - acc: 0.9357 -
val_loss: 0.1897 - val_acc: 0.9416
Epoch 3/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.1631 - acc: 0.9518 -
val_loss: 0.1475 - val_acc: 0.9547
Epoch 4/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.1251 - acc: 0.9627 -
val_loss: 0.1142 - val_acc: 0.9649
Epoch 5/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0985 - acc: 0.9711 -
val_loss: 0.0977 - val_acc: 0.9703
Epoch 6/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0793 - acc: 0.9762 -
val_loss: 0.0865 - val_acc: 0.9729
Epoch 7/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0631 - acc: 0.9812 -
val_loss: 0.0771 - val_acc: 0.9760
Epoch 8/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0514 - acc: 0.9846 -
val_loss: 0.0805 - val_acc: 0.9754
Epoch 9/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0421 - acc: 0.9879 -
val_loss: 0.0718 - val_acc: 0.9781
Epoch 10/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0337 - acc: 0.9906 -
val_loss: 0.0650 - val_acc: 0.9789
Epoch 11/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0268 - acc: 0.9929 -
val_loss: 0.0655 - val_acc: 0.9803
Epoch 12/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0223 - acc: 0.9940 -
val_loss: 0.0668 - val_acc: 0.9797
Epoch 13/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0175 - acc: 0.9956 -
val_loss: 0.0628 - val_acc: 0.9813
Epoch 14/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0141 - acc: 0.9966 -
val_loss: 0.0650 - val_acc: 0.9818
Epoch 15/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.0119 - acc: 0.9971 -
val_loss: 0.0682 - val_acc: 0.9802
Epoch 16/20
60000/60000 [==============================] - 3s 57us/step - loss: 0.0084 - acc: 0.9983 -
val_loss: 0.0679 - val_acc: 0.9812
Epoch 17/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0075 - acc: 0.9984 -
val_loss: 0.0651 - val_acc: 0.9819
Epoch 18/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0056 - acc: 0.9990 -
val_loss: 0.0647 - val_acc: 0.9823
Epoch 19/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0053 - acc: 0.9988 -
val_loss: 0.0727 - val_acc: 0.9809
Epoch 20/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0034 - acc: 0.9994 -
val_loss: 0.0700 - val_acc: 0.9826
```

In [101]:

```
score = model_sigmoid.evaluate(X_test, Y_test, verbose=0)
```

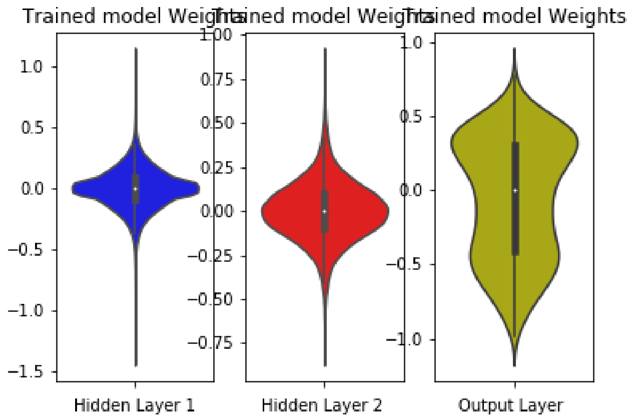
```
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

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

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

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

```
Test score: 0.0699703871981983
Test accuracy: 0.9826
```



In [102]:

```
w_after = model_sigmoid.get_weights()

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


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

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

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



MLP + ReLU +SGD

In [103]:

```
# Multilayer perceptron

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

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

model_relu.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_109 (Dense)            (None, 512)               401920
_________________________________________________________________
dense_110 (Dense)            (None, 128)               65664
_________________________________________________________________
dense_111 (Dense)            (None, 10)                1290
=================================================================
Total params: 468,874
Trainable params: 468,874
Non-trainable params: 0
_________________________________________________________________
```

In [104]:

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

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.7727 - acc: 0.7796 -
val_loss: 0.3984 - val_acc: 0.8890
Epoch 2/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.3619 - acc: 0.8984 -
val_loss: 0.3029 - val_acc: 0.9131
Epoch 3/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.2946 - acc: 0.9168 -
val_loss: 0.2617 - val_acc: 0.9263
```

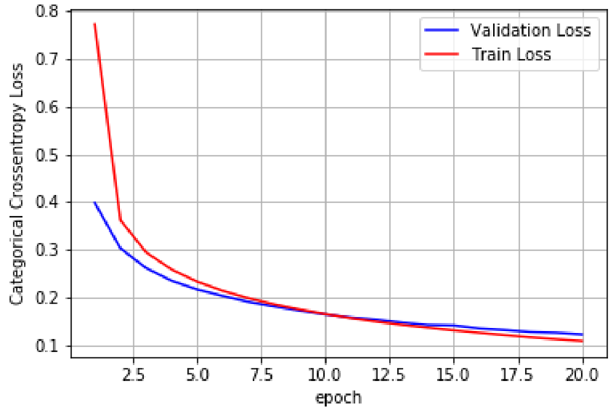
```
Epoch 4/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.2585 - acc: 0.9263 -
val_loss: 0.2349 - val_acc: 0.9343
Epoch 5/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.2330 - acc: 0.9342 -
val_loss: 0.2166 - val_acc: 0.9387
Epoch 6/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.2137 - acc: 0.9399 -
val_loss: 0.2030 - val_acc: 0.9409
Epoch 7/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.1983 - acc: 0.9437 -
val_loss: 0.1905 - val_acc: 0.9453
Epoch 8/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.1855 - acc: 0.9478 -
val_loss: 0.1810 - val_acc: 0.9477
Epoch 9/20
60000/60000 [==============================] - 3s 46us/step - loss: 0.1747 - acc: 0.9501 -
val_loss: 0.1721 - val_acc: 0.9485
Epoch 10/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.1651 - acc: 0.9534 -
val_loss: 0.1646 - val_acc: 0.9510
Epoch 11/20
60000/60000 [==============================] - 3s 45us/step - loss: 0.1567 - acc: 0.9556 -
val_loss: 0.1579 - val_acc: 0.9527
Epoch 12/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.1494 - acc: 0.9573 -
val_loss: 0.1536 - val_acc: 0.9545
Epoch 13/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.1424 - acc: 0.9602 -
val_loss: 0.1474 - val_acc: 0.9564
Epoch 14/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.1366 - acc: 0.9618 -
val_loss: 0.1424 - val_acc: 0.9571
Epoch 15/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.1311 - acc: 0.9636 -
val_loss: 0.1413 - val_acc: 0.9574
Epoch 16/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.1259 - acc: 0.9649 -
val_loss: 0.1350 - val_acc: 0.9590
Epoch 17/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.1211 - acc: 0.9660 -
val_loss: 0.1317 - val_acc: 0.9595
Epoch 18/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.1166 - acc: 0.9677 -
val_loss: 0.1276 - val_acc: 0.9609
Epoch 19/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.1125 - acc: 0.9689 -
val_loss: 0.1260 - val_acc: 0.9610
Epoch 20/20
60000/60000 [==============================] - 3s 46us/step - loss: 0.1087 - acc: 0.9698 -
val_loss: 0.1225 - val_acc: 0.9627
```

In [105]:

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

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

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

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

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

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

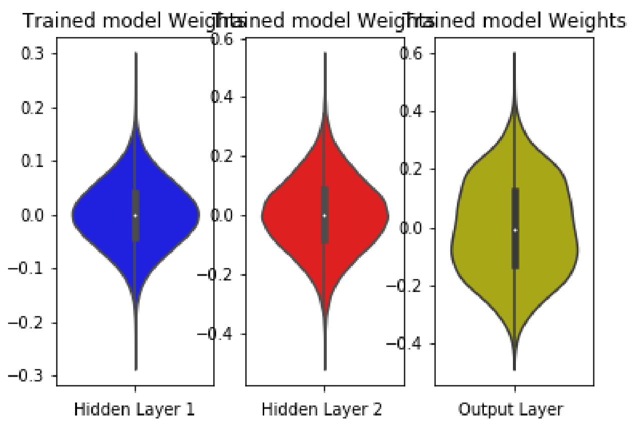
```
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.12253610298782587
Test accuracy: 0.9627
```



In [106]:

```
w_after = model_relu.get_weights()

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


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

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

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



MLP + ReLU + ADAM

In [107]:

```
model_relu = Sequential()
model_relu.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNor
mal(mean=0.0, stddev=0.062, seed=None)))
model_relu.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125
, seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

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

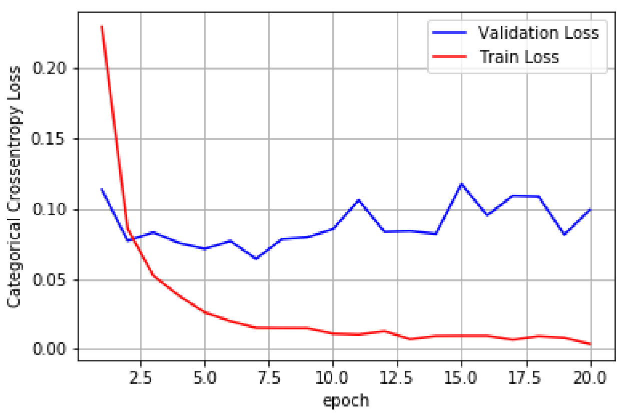
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_112 (Dense)            (None, 512)               401920    
_________________________________________________________________
dense_113 (Dense)            (None, 128)               65664     
_________________________________________________________________
dense_114 (Dense)            (None, 10)                1290      
=================================================================
Total params: 468,874
Trainable params: 468,874
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.2288 - acc: 0.9326 -
val_loss: 0.1132 - val_acc: 0.9655
Epoch 2/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0854 - acc: 0.9737 -
val_loss: 0.0770 - val_acc: 0.9761
Epoch 3/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.0522 - acc: 0.9837 -
val_loss: 0.0829 - val_acc: 0.9741
Epoch 4/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.0380 - acc: 0.9879 -
val_loss: 0.0755 - val_acc: 0.9764
Epoch 5/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0262 - acc: 0.9918 -
val_loss: 0.0712 - val_acc: 0.9793
Epoch 6/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0200 - acc: 0.9940 -
val_loss: 0.0768 - val_acc: 0.9788
Epoch 7/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0152 - acc: 0.9951 -
val_loss: 0.0642 - val_acc: 0.9828
Epoch 8/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0151 - acc: 0.9950 -
val_loss: 0.0781 - val_acc: 0.9796
Epoch 9/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0151 - acc: 0.9950 -
val_loss: 0.0795 - val_acc: 0.9795
Epoch 10/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.0111 - acc: 0.9962 -
val_loss: 0.0853 - val_acc: 0.9806
Epoch 11/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.0105 - acc: 0.9965 -
val_loss: 0.1058 - val_acc: 0.9755
Epoch 12/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0129 - acc: 0.9956 -
val_loss: 0.0834 - val_acc: 0.9799
Epoch 13/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0072 - acc: 0.9977 -
val_loss: 0.0840 - val_acc: 0.9814
Epoch 14/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0095 - acc: 0.9968 -
val_loss: 0.0819 - val_acc: 0.9814
Epoch 15/20
```

```
60000/60000 [==============================] - 3s 52us/step - loss: 0.0096 - acc: 0.9967 -
val_loss: 0.1173 - val_acc: 0.9756
Epoch 16/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0096 - acc: 0.9966 -
val_loss: 0.0950 - val_acc: 0.9800
Epoch 17/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0068 - acc: 0.9978 -
val_loss: 0.1090 - val_acc: 0.9777
Epoch 18/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0093 - acc: 0.9972 -
val_loss: 0.1086 - val_acc: 0.9780
Epoch 19/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0082 - acc: 0.9971 -
val_loss: 0.0813 - val_acc: 0.9820
Epoch 20/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0038 - acc: 0.9988 -
val_loss: 0.0990 - val_acc: 0.9826
```

In [108]:

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

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

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

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

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

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


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

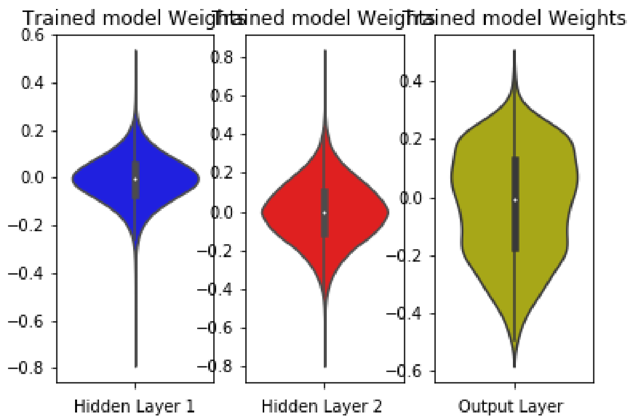
```
Test score: 0.09899708161438571
Test accuracy: 0.9826
```



In [109]:

```
w_after = model_relu.get_weights()

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

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


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

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

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



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

In [110]:

```
# Multilayer perceptron

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

from keras.layers.normalization import BatchNormalization

model_batch = Sequential()

model_batch.add(Dense(512, activation='sigmoid', input_shape=(input_dim,), kernel_initializer=Rando
mNormal(mean=0.0, stddev=0.039, seed=None)))
model_batch.add(BatchNormalization())

model_batch.add(Dense(128, activation='sigmoid', kernel_initializer=RandomNormal(mean=0.0, stddev=0
.55, seed=None)) )
model_batch.add(BatchNormalization())

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


model_batch.summary()
```

____

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_115 (Dense)            (None, 512)               401920
_________________________________________________________________
batch_normalization_45 (Batc (None, 512)               2048
_________________________________________________________________
dense_116 (Dense)            (None, 128)               65664
_________________________________________________________________
batch_normalization_46 (Batc (None, 128)               512
_________________________________________________________________
dense_117 (Dense)            (None, 10)                1290
=================================================================
Total params: 471,434
Trainable params: 470,154
Non-trainable params: 1,280
_________________________________________________________________
```

In [111]:

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

history = model_batch.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, vali
dation_data=(X_test, Y_test))
```

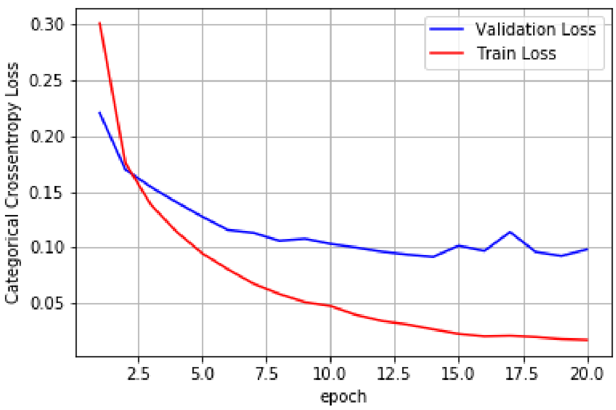
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 10s 163us/step - loss: 0.3013 - acc: 0.9113 - val_l
oss: 0.2207 - val_acc: 0.9347
Epoch 2/20
60000/60000 [==============================] - 5s 79us/step - loss: 0.1757 - acc: 0.9487 -
val_loss: 0.1698 - val_acc: 0.9502
Epoch 3/20
60000/60000 [==============================] - 5s 79us/step - loss: 0.1382 - acc: 0.9593 -
val_loss: 0.1542 - val_acc: 0.9546
Epoch 4/20
60000/60000 [==============================] - 5s 80us/step - loss: 0.1140 - acc: 0.9656 -
val_loss: 0.1405 - val_acc: 0.9581
Epoch 5/20
60000/60000 [==============================] - 5s 79us/step - loss: 0.0947 - acc: 0.9720 -
val_loss: 0.1276 - val_acc: 0.9625
Epoch 6/20
60000/60000 [==============================] - 5s 80us/step - loss: 0.0804 - acc: 0.9753 -
val_loss: 0.1157 - val_acc: 0.9654
Epoch 7/20
60000/60000 [==============================] - 5s 80us/step - loss: 0.0674 - acc: 0.9795 -
val_loss: 0.1131 - val_acc: 0.9656
Epoch 8/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.0582 - acc: 0.9826 -
val_loss: 0.1060 - val_acc: 0.9675
Epoch 9/20
60000/60000 [==============================] - 5s 81us/step - loss: 0.0509 - acc: 0.9842 -
val_loss: 0.1078 - val_acc: 0.9687
Epoch 10/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0474 - acc: 0.9853 -
val_loss: 0.1032 - val_acc: 0.9681
Epoch 11/20
60000/60000 [==============================] - 5s 81us/step - loss: 0.0395 - acc: 0.9876 -
val_loss: 0.0999 - val_acc: 0.9705
Epoch 12/20
60000/60000 [==============================] - 5s 81us/step - loss: 0.0341 - acc: 0.9889 -
val_loss: 0.0963 - val_acc: 0.9719
Epoch 13/20
60000/60000 [==============================] - 5s 81us/step - loss: 0.0308 - acc: 0.9902 -
val_loss: 0.0936 - val_acc: 0.9710
Epoch 14/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0265 - acc: 0.9917 -
val_loss: 0.0916 - val_acc: 0.9733
Epoch 15/20
60000/60000 [==============================] - 5s 81us/step - loss: 0.0224 - acc: 0.9933 -
val_loss: 0.1015 - val_acc: 0.9717
Epoch 16/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0202 - acc: 0.9936 -
val_loss: 0.0969 - val_acc: 0.9737
Epoch 17/20
60000/60000 [==============================] - 5s 81us/step - loss: 0.0208 - acc: 0.9932 -
val_loss: 0.1138 - val_acc: 0.9698
```

```
Epoch 18/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0196 - acc: 0.9936 -
val_loss: 0.0960 - val_acc: 0.9750
Epoch 19/20
60000/60000 [==============================] - 5s 81us/step - loss: 0.0177 - acc: 0.9941 -
val_loss: 0.0924 - val_acc: 0.9749
Epoch 20/20
60000/60000 [==============================] - 5s 81us/step - loss: 0.0169 - acc: 0.9944 -
val_loss: 0.0983 - val_acc: 0.9734
```

In [112]:

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

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

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

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

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

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

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

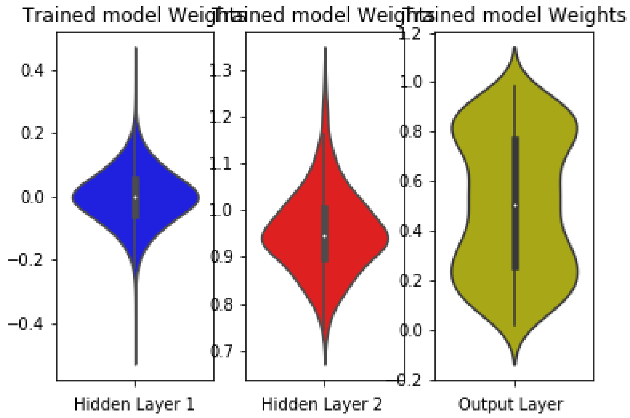
```
Test score: 0.09825632937119808
Test accuracy: 0.9734
```



In [113]:

```
w_after = model_batch.get_weights()

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


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
```

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

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

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



5. MLP + Dropout + AdamOptimizer

In [114]:

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

from keras.layers import Dropout

model_drop = Sequential()

model_drop.add(Dense(512, activation='sigmoid', input_shape=(input_dim,), kernel_initializer=Random
Normal(mean=0.0, stddev=0.039, seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(128, activation='sigmoid', kernel_initializer=RandomNormal(mean=0.0, stddev=0.
55, seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

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


model_drop.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_118 (Dense)            (None, 512)               401920    
_________________________________________________________________
batch_normalization_47 (Batc (None, 512)               2048      
_________________________________________________________________
dropout_35 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_119 (Dense)            (None, 128)               65664     
_________________________________________________________________
batch_normalization_48 (Batc (None, 128)               512       
_________________________________________________________________
dropout_36 (Dropout)         (None, 128)               0         
```

```
_________________________________________________________________
dense_120 (Dense)            (None, 10)                1290      
=================================================================
Total params: 471,434
Trainable params: 470,154
Non-trainable params: 1,280
_________________________________________________________________
```

In [115]:

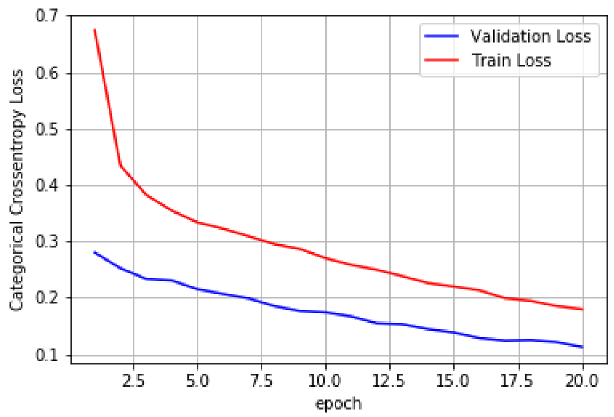
```
model_drop.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 11s 177us/step - loss: 0.6745 - acc: 0.7926 - val_l
oss: 0.2803 - val_acc: 0.9194
Epoch 2/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.4344 - acc: 0.8685 -
val_loss: 0.2526 - val_acc: 0.9237
Epoch 3/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.3827 - acc: 0.8839 -
val_loss: 0.2336 - val_acc: 0.9312
Epoch 4/20
60000/60000 [==============================] - 6s 98us/step - loss: 0.3548 - acc: 0.8927 -
val_loss: 0.2309 - val_acc: 0.9323
Epoch 5/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.3339 - acc: 0.8993 -
val_loss: 0.2153 - val_acc: 0.9359
Epoch 6/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.3227 - acc: 0.9015 -
val_loss: 0.2070 - val_acc: 0.9396
Epoch 7/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.3092 - acc: 0.9063 -
val_loss: 0.1990 - val_acc: 0.9406
Epoch 8/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.2952 - acc: 0.9118 -
val_loss: 0.1854 - val_acc: 0.9429
Epoch 9/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.2866 - acc: 0.9136 -
val_loss: 0.1767 - val_acc: 0.9462
Epoch 10/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.2704 - acc: 0.9189 -
val_loss: 0.1742 - val_acc: 0.9467
Epoch 11/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.2585 - acc: 0.9223 -
val_loss: 0.1670 - val_acc: 0.9505
Epoch 12/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.2499 - acc: 0.9249 -
val_loss: 0.1550 - val_acc: 0.9541
Epoch 13/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.2384 - acc: 0.9281 -
val_loss: 0.1533 - val_acc: 0.9534
Epoch 14/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.2262 - acc: 0.9318 -
val_loss: 0.1447 - val_acc: 0.9560
Epoch 15/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.2201 - acc: 0.9342 -
val_loss: 0.1387 - val_acc: 0.9570
Epoch 16/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.2133 - acc: 0.9354 -
val_loss: 0.1286 - val_acc: 0.9619
Epoch 17/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.1993 - acc: 0.9406 -
val_loss: 0.1242 - val_acc: 0.9622
Epoch 18/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.1947 - acc: 0.9415 -
val_loss: 0.1251 - val_acc: 0.9626
Epoch 19/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.1854 - acc: 0.9446 -
val_loss: 0.1217 - val_acc: 0.9649
Epoch 20/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.1799 - acc: 0.9449 -
val_loss: 0.1129 - val_acc: 0.9685
```

In [116]:

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

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

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

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

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

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

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

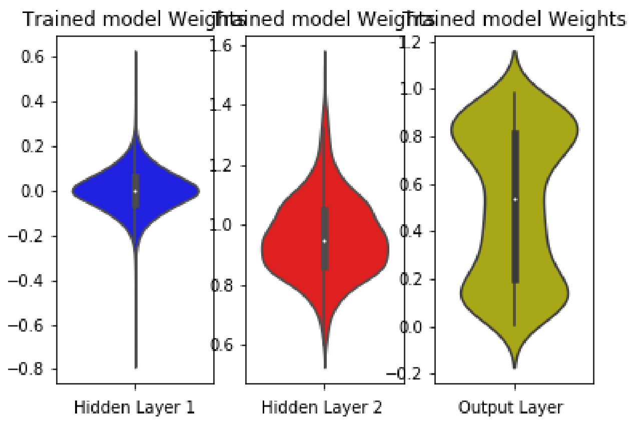
```
Test score: 0.11289959549605846
Test accuracy: 0.9685
```



In [117]:

```
w_after = model_drop.get_weights()

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


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

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

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

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



Hyper-parameter tuning of Keras models using Sklearn

In [118]:

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

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


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

    return model
```

In [119]:

```
# https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras
/

activ = ['sigmoid','relu']

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

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

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

grid = GridSearchCV(estimator=model, param_grid=param_grid)
grid_result = grid.fit(X_train, Y_train)
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/model_selection/_split.py:1978: FutureWarning: The
default value of cv will change from 3 to 5 in version 0.22. Specify it explicitly to silence this
warning.
  warnings.warn(CV_WARNING, FutureWarning)
```

In [120]:

```
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
```

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

```
Best: 0.976300 using {'activ': 'relu'}
0.975617 (0.001652) with: {'activ': 'sigmoid'}
0.976300 (0.002729) with: {'activ': 'relu'}
```

6. Model 1 with two hidden layers + Batch Normalization + Dropout:

In [121]:

```
model_relu = Sequential()
model_relu.add(Dense(256, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNor
mal(mean=0.0, stddev=0.062, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.3))
model_relu.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125
, seed=None)) )
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.3))
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

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

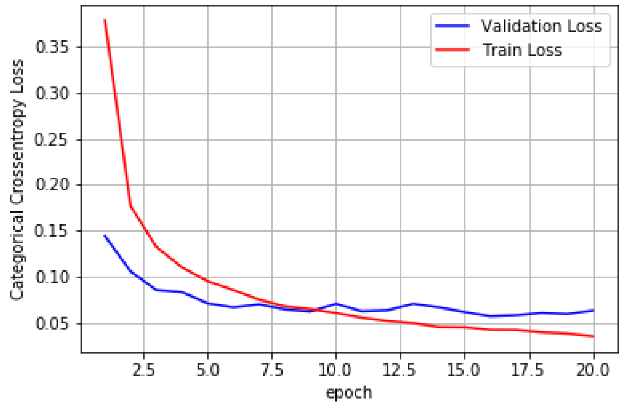
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_142 (Dense)            (None, 256)               200960    
_________________________________________________________________
batch_normalization_49 (Batc (None, 256)               1024      
_________________________________________________________________
dropout_37 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_143 (Dense)            (None, 128)               32896     
_________________________________________________________________
batch_normalization_50 (Batc (None, 128)               512       
_________________________________________________________________
dropout_38 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_144 (Dense)            (None, 10)                1290      
=================================================================
Total params: 236,682
Trainable params: 235,914
Non-trainable params: 768
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 11s 192us/step - loss: 0.3786 - acc: 0.8847 - val_l
oss: 0.1445 - val_acc: 0.9560
Epoch 2/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.1768 - acc: 0.9470 -
val_loss: 0.1059 - val_acc: 0.9682
Epoch 3/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.1326 - acc: 0.9583 -
val_loss: 0.0858 - val_acc: 0.9728
Epoch 4/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.1104 - acc: 0.9652 -
val_loss: 0.0835 - val_acc: 0.9741
Epoch 5/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0954 - acc: 0.9700 -
val_loss: 0.0715 - val_acc: 0.9782
Epoch 6/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0858 - acc: 0.9732 -
val_loss: 0.0671 - val_acc: 0.9801
Epoch 7/20
```

```
60000/60000 [==============================] - 5s 88us/step - loss: 0.0755 - acc: 0.9759 - val_loss: 0.0704 - val_acc: 0.9786
Epoch 8/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0683 - acc: 0.9783 - val_loss: 0.0650 - val_acc: 0.9805
Epoch 9/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0652 - acc: 0.9794 - val_loss: 0.0627 - val_acc: 0.9810
Epoch 10/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0610 - acc: 0.9803 - val_loss: 0.0710 - val_acc: 0.9803
Epoch 11/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0559 - acc: 0.9811 - val_loss: 0.0630 - val_acc: 0.9816
Epoch 12/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0525 - acc: 0.9830 - val_loss: 0.0640 - val_acc: 0.9814
Epoch 13/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0500 - acc: 0.9840 - val_loss: 0.0710 - val_acc: 0.9798
Epoch 14/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0459 - acc: 0.9850 - val_loss: 0.0672 - val_acc: 0.9807
Epoch 15/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0456 - acc: 0.9844 - val_loss: 0.0621 - val_acc: 0.9820
Epoch 16/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0428 - acc: 0.9859 - val_loss: 0.0575 - val_acc: 0.9826
Epoch 17/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0427 - acc: 0.9861 - val_loss: 0.0585 - val_acc: 0.9826
Epoch 18/20
60000/60000 [==============================] - 5s 87us/step - loss: 0.0401 - acc: 0.9862 - val_loss: 0.0611 - val_acc: 0.9833
Epoch 19/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0387 - acc: 0.9876 - val_loss: 0.0600 - val_acc: 0.9818
Epoch 20/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0358 - acc: 0.9880 - val_loss: 0.0637 - val_acc: 0.9827
```

In [122]:

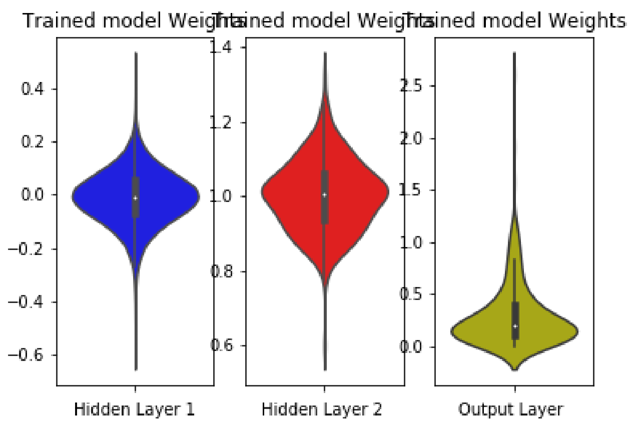
```
score = model_relu.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

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

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


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

```
Test score: 0.06367886031113303
Test accuracy: 0.9827
```



In [123]:

```
w_after = model_relu.get_weights()

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


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

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

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



7. Model 2 with three hidden layers + Batch Normalization + Dropout:

In [124]:

```
model_relu = Sequential()
model_relu.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.3))
```

```
model_relu.add(Dense(256, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNor
mal(mean=0.0, stddev=0.062, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.3))
model_relu.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125
, seed=None)) )
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.3))
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

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

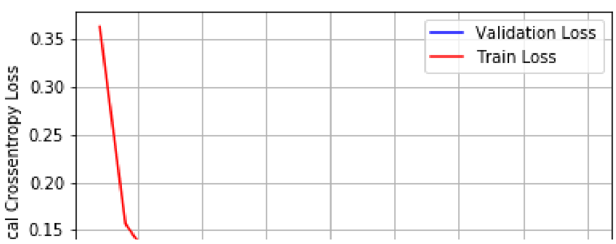
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_145 (Dense)            (None, 512)               401920    
_________________________________________________________________
batch_normalization_51 (Batc (None, 512)               2048      
_________________________________________________________________
dropout_39 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_146 (Dense)            (None, 256)               131328    
_________________________________________________________________
batch_normalization_52 (Batc (None, 256)               1024      
_________________________________________________________________
dropout_40 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_147 (Dense)            (None, 128)               32896     
_________________________________________________________________
batch_normalization_53 (Batc (None, 128)               512       
_________________________________________________________________
dropout_41 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_148 (Dense)            (None, 10)                1290      
=================================================================
Total params: 571,018
Trainable params: 569,226
Non-trainable params: 1,792
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 13s 224us/step - loss: 0.3627 - acc: 0.8891 - val_l
oss: 0.1164 - val_acc: 0.9641
Epoch 2/20
60000/60000 [==============================] - 6s 106us/step - loss: 0.1567 - acc: 0.9530 -
val_loss: 0.0919 - val_acc: 0.9695
Epoch 3/20
60000/60000 [==============================] - 6s 108us/step - loss: 0.1184 - acc: 0.9636 -
val_loss: 0.0804 - val_acc: 0.9751
Epoch 4/20
60000/60000 [==============================] - 6s 106us/step - loss: 0.1020 - acc: 0.9687 -
val_loss: 0.0829 - val_acc: 0.9746
Epoch 5/20
60000/60000 [==============================] - 6s 107us/step - loss: 0.0844 - acc: 0.9733 -
val_loss: 0.0787 - val_acc: 0.9756
Epoch 6/20
60000/60000 [==============================] - 6s 107us/step - loss: 0.0770 - acc: 0.9754 -
val_loss: 0.0699 - val_acc: 0.9785
Epoch 7/20
60000/60000 [==============================] - 6s 107us/step - loss: 0.0689 - acc: 0.9780 -
val_loss: 0.0686 - val_acc: 0.9770
Epoch 8/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.0626 - acc: 0.9799 -
val_loss: 0.0690 - val_acc: 0.9794
Epoch 9/20
60000/60000 [==============================] - 7s 111us/step - loss: 0.0568 - acc: 0.9819 -
val_loss: 0.0632 - val_acc: 0.9808
Epoch 10/20
60000/60000 [==============================] - 6s 108us/step - loss: 0.0511 - acc: 0.9834 -
val_loss: 0.0666 - val_acc: 0.9802
```

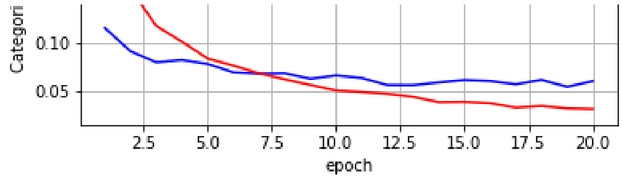
```
Epoch 11/20
60000/60000 [==============================] - 6s 108us/step - loss: 0.0495 - acc: 0.9839 -
val_loss: 0.0640 - val_acc: 0.9800
Epoch 12/20
60000/60000 [==============================] - 7s 109us/step - loss: 0.0475 - acc: 0.9845 -
val_loss: 0.0567 - val_acc: 0.9833
Epoch 13/20
60000/60000 [==============================] - 6s 107us/step - loss: 0.0444 - acc: 0.9856 -
val_loss: 0.0566 - val_acc: 0.9836
Epoch 14/20
60000/60000 [==============================] - 6s 108us/step - loss: 0.0388 - acc: 0.9875 -
val_loss: 0.0595 - val_acc: 0.9823
Epoch 15/20
60000/60000 [==============================] - 6s 107us/step - loss: 0.0391 - acc: 0.9874 -
val_loss: 0.0619 - val_acc: 0.9820
Epoch 16/20
60000/60000 [==============================] - 6s 107us/step - loss: 0.0378 - acc: 0.9879 -
val_loss: 0.0609 - val_acc: 0.9828
Epoch 17/20
60000/60000 [==============================] - 6s 108us/step - loss: 0.0332 - acc: 0.9890 -
val_loss: 0.0576 - val_acc: 0.9844
Epoch 18/20
60000/60000 [==============================] - 6s 107us/step - loss: 0.0349 - acc: 0.9884 -
val_loss: 0.0621 - val_acc: 0.9821
Epoch 19/20
60000/60000 [==============================] - 6s 107us/step - loss: 0.0323 - acc: 0.9894 -
val_loss: 0.0549 - val_acc: 0.9857
Epoch 20/20
60000/60000 [==============================] - 6s 108us/step - loss: 0.0317 - acc: 0.9896 -
val_loss: 0.0609 - val_acc: 0.9843
```

In [125]:

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

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

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

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

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

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


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

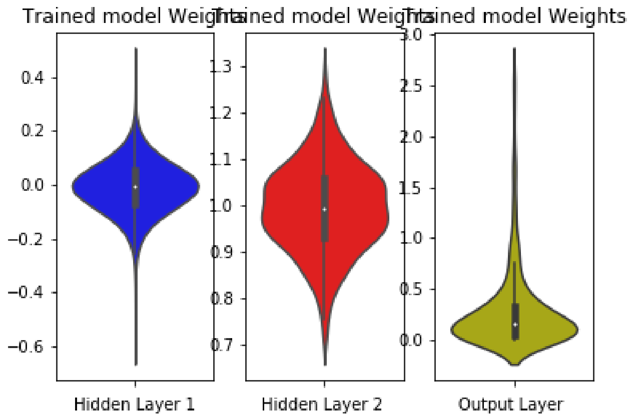
```
Test score: 0.060851101654174275
Test accuracy: 0.9843
```





In [126]:

```
w_after = model_relu.get_weights()

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


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

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

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



8. Model 3 with five hidden layers + Batch Normalization + Dropout:

In [127]:

```
model_relu = Sequential()
model_relu.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNor
mal(mean=0.0, stddev=0.062, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.3))
model_relu.add(Dense(256, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNor
mal(mean=0.0, stddev=0.062, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.3))
model_relu.add(Dense(128, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNor
mal(mean=0.0, stddev=0.062, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.3))
model_relu.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125,
seed=None)) )
model_relu.add(BatchNormalization())
```

```
model_relu.add(Dense(32, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125,
seed=None)) )
model_relu.add(BatchNormalization())
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

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

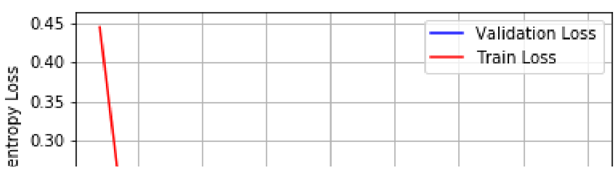
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_149 (Dense)            (None, 512)               401920    
_________________________________________________________________
batch_normalization_54 (Batc (None, 512)               2048      
_________________________________________________________________
dropout_42 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_150 (Dense)            (None, 256)               131328    
_________________________________________________________________
batch_normalization_55 (Batc (None, 256)               1024      
_________________________________________________________________
dropout_43 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_151 (Dense)            (None, 128)               32896     
_________________________________________________________________
batch_normalization_56 (Batc (None, 128)               512       
_________________________________________________________________
dropout_44 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_152 (Dense)            (None, 64)                8256      
_________________________________________________________________
batch_normalization_57 (Batc (None, 64)                256       
_________________________________________________________________
dense_153 (Dense)            (None, 32)                2080      
_________________________________________________________________
batch_normalization_58 (Batc (None, 32)                128       
_________________________________________________________________
dense_154 (Dense)            (None, 10)                330       
=================================================================
Total params: 580,778
Trainable params: 578,794
Non-trainable params: 1,984
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 16s 273us/step - loss: 0.4450 - acc: 0.8676 - val_l
oss: 0.1393 - val_acc: 0.9595
Epoch 2/20
60000/60000 [==============================] - 9s 144us/step - loss: 0.1794 - acc: 0.9467 -
val_loss: 0.1054 - val_acc: 0.9682
Epoch 3/20
60000/60000 [==============================] - 9s 143us/step - loss: 0.1373 - acc: 0.9597 -
val_loss: 0.0938 - val_acc: 0.9715
Epoch 4/20
60000/60000 [==============================] - 9s 144us/step - loss: 0.1131 - acc: 0.9658 -
val_loss: 0.0848 - val_acc: 0.9747
Epoch 5/20
60000/60000 [==============================] - 9s 142us/step - loss: 0.0990 - acc: 0.9699 -
val_loss: 0.0750 - val_acc: 0.9779
Epoch 6/20
60000/60000 [==============================] - 9s 144us/step - loss: 0.0866 - acc: 0.9733 -
val_loss: 0.0688 - val_acc: 0.9805
Epoch 7/20
60000/60000 [==============================] - 8s 141us/step - loss: 0.0795 - acc: 0.9754 -
val_loss: 0.0703 - val_acc: 0.9788
Epoch 8/20
60000/60000 [==============================] - 9s 143us/step - loss: 0.0718 - acc: 0.9781 -
val_loss: 0.0624 - val_acc: 0.9816
Epoch 9/20
60000/60000 [==============================] - 9s 143us/step - loss: 0.0664 - acc: 0.9800 -
val_loss: 0.0672 - val_acc: 0.9805
Epoch 10/20
```

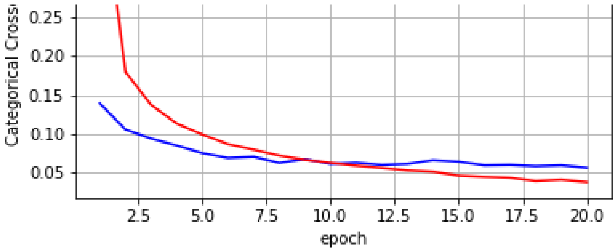
```
Epoch 10/20
60000/60000 [==============================] - 9s 142us/step - loss: 0.0625 - acc: 0.9800 -
val_loss: 0.0611 - val_acc: 0.9813
Epoch 11/20
60000/60000 [==============================] - 9s 144us/step - loss: 0.0588 - acc: 0.9817 -
val_loss: 0.0627 - val_acc: 0.9824
Epoch 12/20
60000/60000 [==============================] - 9s 142us/step - loss: 0.0559 - acc: 0.9827 -
val_loss: 0.0598 - val_acc: 0.9830
Epoch 13/20
60000/60000 [==============================] - 9s 143us/step - loss: 0.0528 - acc: 0.9834 -
val_loss: 0.0611 - val_acc: 0.9839
Epoch 14/20
60000/60000 [==============================] - 9s 143us/step - loss: 0.0510 - acc: 0.9841 -
val_loss: 0.0659 - val_acc: 0.9822
Epoch 15/20
60000/60000 [==============================] - 9s 143us/step - loss: 0.0460 - acc: 0.9860 -
val_loss: 0.0639 - val_acc: 0.9822
Epoch 16/20
60000/60000 [==============================] - 9s 144us/step - loss: 0.0444 - acc: 0.9861 -
val_loss: 0.0593 - val_acc: 0.9834
Epoch 17/20
60000/60000 [==============================] - 8s 141us/step - loss: 0.0433 - acc: 0.9865 -
val_loss: 0.0599 - val_acc: 0.9835
Epoch 18/20
60000/60000 [==============================] - 9s 142us/step - loss: 0.0391 - acc: 0.9880 -
val_loss: 0.0582 - val_acc: 0.9850
Epoch 19/20
60000/60000 [==============================] - 9s 142us/step - loss: 0.0407 - acc: 0.9875 -
val_loss: 0.0593 - val_acc: 0.9835
Epoch 20/20
60000/60000 [==============================] - 9s 143us/step - loss: 0.0376 - acc: 0.9880 -
val_loss: 0.0561 - val_acc: 0.9843
```

In [128]:

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

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

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

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

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

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


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

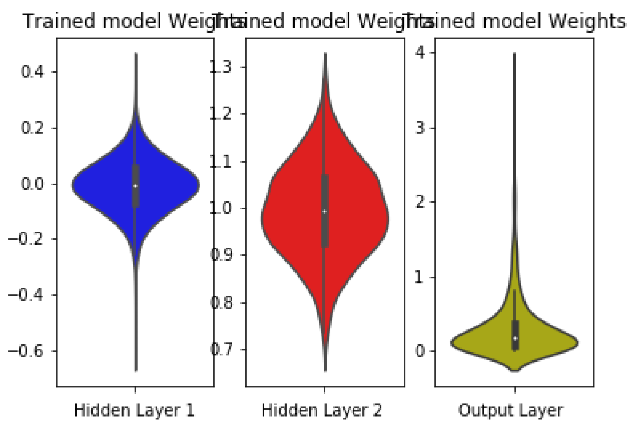
```
Test score: 0.05613952756321523
Test accuracy: 0.9843
```





In [129]:

```
w_after = model_relu.get_weights()

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


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

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

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



Conclusion:

In [150]:

```
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["# Layers", "Epoch", "Accuracy"]
x.add_row(["2", 20, 0.9827])
x.add_row(["3", 20, 0.9843])
x.add_row(["5", 20, 0.9843])
print(x)
```

```
+----------+-------+----------+
| # Layers | Epoch | Accuracy |
+----------+-------+----------+
|    2     |   20  |  0.9827  |
```

```
|     3     |   20  |  0.9843  |
|     5     |   20  |  0.9843  |
+-----------+-------+----------+
```

9. Model 1: MLP + BatchNormalization + Dropout (0.30)

- #layers: 5
- activation: ReLU
- Weight Initializer: RandomNormal
- Optimizer: ADAM

In [131]:

```
model_relu = Sequential()
model_relu.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNor
mal(mean=0.0, stddev=0.050, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.3))
model_relu.add(Dense(256, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNor
mal(mean=0.0, stddev=0.062, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.3))
model_relu.add(Dense(128, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNor
mal(mean=0.0, stddev=0.088, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.3))
model_relu.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125,
seed=None)) )
model_relu.add(BatchNormalization())
model_relu.add(Dense(32, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.176,
seed=None)) )
model_relu.add(BatchNormalization())
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_155 (Dense)            (None, 512)               401920
_________________________________________________________________
batch_normalization_59 (Batc (None, 512)               2048
_________________________________________________________________
dropout_45 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_156 (Dense)            (None, 256)               131328
_________________________________________________________________
batch_normalization_60 (Batc (None, 256)               1024
_________________________________________________________________
dropout_46 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_157 (Dense)            (None, 128)               32896
_________________________________________________________________
batch_normalization_61 (Batc (None, 128)               512
_________________________________________________________________
dropout_47 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_158 (Dense)            (None, 64)                8256
_________________________________________________________________
batch_normalization_62 (Batc (None, 64)                256
_________________________________________________________________
dense_159 (Dense)            (None, 32)                2080
_________________________________________________________________
batch_normalization_63 (Batc (None, 32)                128
_________________________________________________________________
dense_160 (Dense)            (None, 10)                330
=================================================================
Total params: 580,778
```

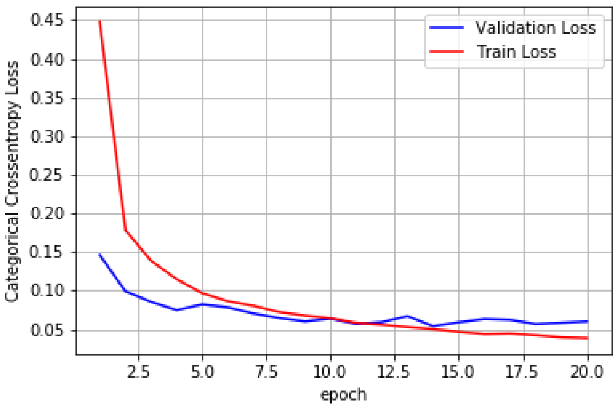
```
Trainable params: 578,794
Non-trainable params: 1,984
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 17s 278us/step - loss: 0.4477 - acc: 0.8703 - val_l
oss: 0.1462 - val_acc: 0.9558
Epoch 2/20
60000/60000 [==============================] - 8s 142us/step - loss: 0.1781 - acc: 0.9478 -
val_loss: 0.0991 - val_acc: 0.9717
Epoch 3/20
60000/60000 [==============================] - 9s 158us/step - loss: 0.1386 - acc: 0.9588 -
val_loss: 0.0854 - val_acc: 0.9741
Epoch 4/20
60000/60000 [==============================] - 9s 144us/step - loss: 0.1147 - acc: 0.9652 -
val_loss: 0.0748 - val_acc: 0.9775
Epoch 5/20
60000/60000 [==============================] - 9s 144us/step - loss: 0.0966 - acc: 0.9700 -
val_loss: 0.0822 - val_acc: 0.9767
Epoch 6/20
60000/60000 [==============================] - 8s 141us/step - loss: 0.0863 - acc: 0.9734 -
val_loss: 0.0783 - val_acc: 0.9772
Epoch 7/20
60000/60000 [==============================] - 9s 143us/step - loss: 0.0803 - acc: 0.9758 -
val_loss: 0.0704 - val_acc: 0.9790
Epoch 8/20
60000/60000 [==============================] - 9s 142us/step - loss: 0.0723 - acc: 0.9777 -
val_loss: 0.0646 - val_acc: 0.9802
Epoch 9/20
60000/60000 [==============================] - 8s 142us/step - loss: 0.0678 - acc: 0.9787 -
val_loss: 0.0602 - val_acc: 0.9831
Epoch 10/20
60000/60000 [==============================] - 8s 142us/step - loss: 0.0644 - acc: 0.9802 -
val_loss: 0.0638 - val_acc: 0.9814
Epoch 11/20
60000/60000 [==============================] - 9s 143us/step - loss: 0.0583 - acc: 0.9819 -
val_loss: 0.0568 - val_acc: 0.9834
Epoch 12/20
60000/60000 [==============================] - 9s 145us/step - loss: 0.0559 - acc: 0.9826 -
val_loss: 0.0593 - val_acc: 0.9831
Epoch 13/20
60000/60000 [==============================] - 9s 142us/step - loss: 0.0529 - acc: 0.9838 -
val_loss: 0.0670 - val_acc: 0.9802
Epoch 14/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.0502 - acc: 0.9838 -
val_loss: 0.0541 - val_acc: 0.9845
Epoch 15/20
60000/60000 [==============================] - 8s 141us/step - loss: 0.0468 - acc: 0.9851 -
val_loss: 0.0590 - val_acc: 0.9848
Epoch 16/20
60000/60000 [==============================] - 9s 143us/step - loss: 0.0438 - acc: 0.9863 -
val_loss: 0.0633 - val_acc: 0.9835
Epoch 17/20
60000/60000 [==============================] - 9s 142us/step - loss: 0.0444 - acc: 0.9860 -
val_loss: 0.0622 - val_acc: 0.9836
Epoch 18/20
60000/60000 [==============================] - 9s 142us/step - loss: 0.0423 - acc: 0.9872 -
val_loss: 0.0568 - val_acc: 0.9834
Epoch 19/20
60000/60000 [==============================] - 9s 143us/step - loss: 0.0396 - acc: 0.9871 -
val_loss: 0.0584 - val_acc: 0.9849
Epoch 20/20
60000/60000 [==============================] - 9s 142us/step - loss: 0.0387 - acc: 0.9882 -
val_loss: 0.0600 - val_acc: 0.9832
```

In [132]:

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

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

# list of epoch numbers
```

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

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

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

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


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

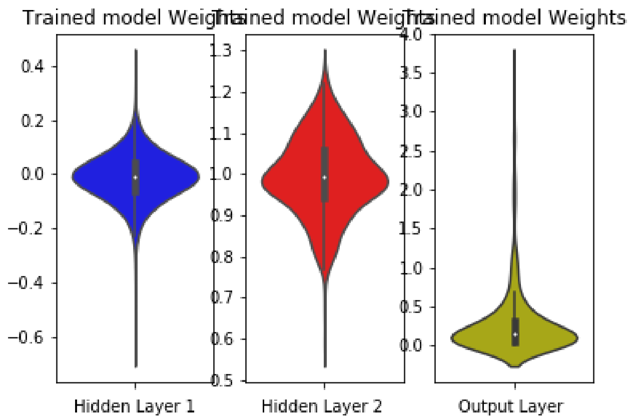
```
Test score: 0.06003061625857372
Test accuracy: 0.9832
```



In [133]:

```
w_after = model_relu.get_weights()

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


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

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

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

```
/opt/conda/lib/python3.6/site-packages/matplotlib/pyplot.py:514: RuntimeWarning: More than 20
figures have been opened. Figures created through the pyplot interface
(`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume too much memory.
(To control this warning, see the rcParam `figure.max_open_warning`).
  max_open_warning, RuntimeWarning)
```



10. Model 2: MLP + Dropout (0.30)

- #layers: 5
- activation: ReLU
- Weight Initializer: He Normal
- Optimizer: ADAM

In [134]:

```
model_relu = Sequential()
model_relu.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal
()))
model_relu.add(Dropout(0.3))
model_relu.add(Dense(256, activation='relu', kernel_initializer=he_normal()))
model_relu.add(Dropout(0.3))
model_relu.add(Dense(128, activation='relu', kernel_initializer=he_normal()))
model_relu.add(Dropout(0.3))
model_relu.add(Dense(64, activation='relu', kernel_initializer=he_normal()) )
model_relu.add(Dense(32, activation='relu', kernel_initializer=he_normal()) )
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_161 (Dense)            (None, 512)               401920    
_________________________________________________________________
dropout_48 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_162 (Dense)            (None, 256)               131328    
_________________________________________________________________
dropout_49 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_163 (Dense)            (None, 128)               32896     
_________________________________________________________________
dropout_50 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_164 (Dense)            (None, 64)                8256      
_________________________________________________________________
dense_165 (Dense)            (None, 32)                2080      
_________________________________________________________________
dense_166 (Dense)            (None, 10)                330       
=================================================================
Total params: 576,810
Trainable params: 576,810
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
```

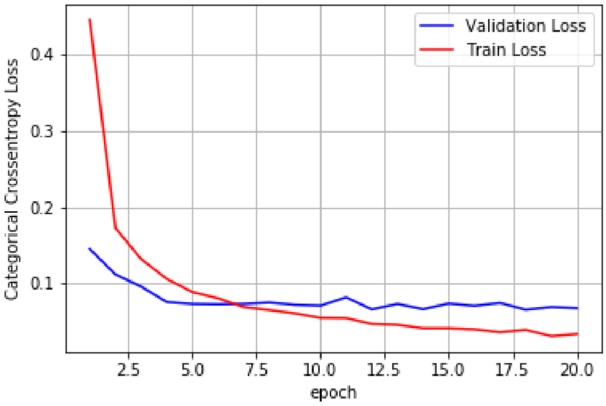
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 12s 194us/step - loss: 0.4457 - acc: 0.8628 - val_l
oss: 0.1451 - val_acc: 0.9572
Epoch 2/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.1724 - acc: 0.9502 -
val_loss: 0.1117 - val_acc: 0.9678
Epoch 3/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.1319 - acc: 0.9628 -
val_loss: 0.0957 - val_acc: 0.9693
Epoch 4/20
60000/60000 [==============================] - 4s 75us/step - loss: 0.1059 - acc: 0.9694 -
val_loss: 0.0755 - val_acc: 0.9762
Epoch 5/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0885 - acc: 0.9738 -
val_loss: 0.0729 - val_acc: 0.9785
Epoch 6/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.0804 - acc: 0.9756 -
val_loss: 0.0725 - val_acc: 0.9783
Epoch 7/20
60000/60000 [==============================] - 4s 75us/step - loss: 0.0689 - acc: 0.9796 -
val_loss: 0.0730 - val_acc: 0.9782
Epoch 8/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.0650 - acc: 0.9807 -
val_loss: 0.0749 - val_acc: 0.9788
Epoch 9/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.0606 - acc: 0.9816 -
val_loss: 0.0718 - val_acc: 0.9794
Epoch 10/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.0548 - acc: 0.9834 -
val_loss: 0.0708 - val_acc: 0.9813
Epoch 11/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.0544 - acc: 0.9832 -
val_loss: 0.0816 - val_acc: 0.9791
Epoch 12/20
60000/60000 [==============================] - 4s 75us/step - loss: 0.0469 - acc: 0.9856 -
val_loss: 0.0659 - val_acc: 0.9825
Epoch 13/20
60000/60000 [==============================] - 5s 76us/step - loss: 0.0457 - acc: 0.9866 -
val_loss: 0.0730 - val_acc: 0.9813
Epoch 14/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0411 - acc: 0.9875 -
val_loss: 0.0662 - val_acc: 0.9829
Epoch 15/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.0410 - acc: 0.9877 -
val_loss: 0.0734 - val_acc: 0.9827
Epoch 16/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.0397 - acc: 0.9886 -
val_loss: 0.0705 - val_acc: 0.9828
Epoch 17/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.0360 - acc: 0.9892 -
val_loss: 0.0742 - val_acc: 0.9814
Epoch 18/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.0390 - acc: 0.9885 -
val_loss: 0.0655 - val_acc: 0.9829
Epoch 19/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0309 - acc: 0.9904 -
val_loss: 0.0689 - val_acc: 0.9833
Epoch 20/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.0335 - acc: 0.9901 -
val_loss: 0.0675 - val_acc: 0.9833
```

In [135]:

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

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

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

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

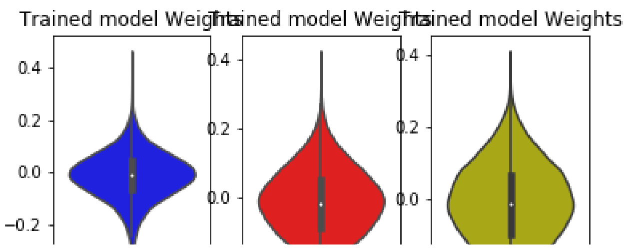
```
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, va
lidation_data=(X_test, Y_test))

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

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


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

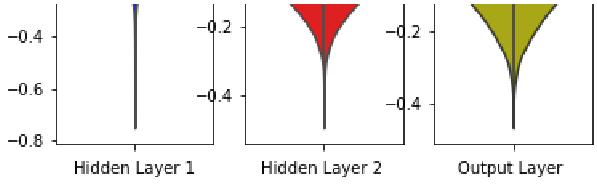
```
Test score: 0.06747149933629462
Test accuracy: 0.9833
```



In [136]:

```
w_after = model_relu.get_weights()

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


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

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

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





11. Model 3: MLP + BatchNormalization + Dropout (0.40)

- #layers: 5
- activation: ReLU
- Weight Initializer: RandomNormal
- Optimizer: ADAM

In [137]:

```
model_relu = Sequential()
model_relu.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNor
mal(mean=0.0, stddev=0.050, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.4))
model_relu.add(Dense(256, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNor
mal(mean=0.0, stddev=0.062, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.4))
model_relu.add(Dense(128, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNor
mal(mean=0.0, stddev=0.088, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.4))
model_relu.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125,
seed=None)) )
model_relu.add(BatchNormalization())
model_relu.add(Dense(32, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.176,
seed=None)) )
model_relu.add(BatchNormalization())
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_167 (Dense)            (None, 512)               401920    
_________________________________________________________________
batch_normalization_64 (Batc (None, 512)               2048      
_________________________________________________________________
dropout_51 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_168 (Dense)            (None, 256)               131328    
_________________________________________________________________
batch_normalization_65 (Batc (None, 256)               1024      
_________________________________________________________________
dropout_52 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_169 (Dense)            (None, 128)               32896     
_________________________________________________________________
batch_normalization_66 (Batc (None, 128)               512       
_________________________________________________________________
dropout_53 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_170 (Dense)            (None, 64)                8256      
_________________________________________________________________
batch_normalization_67 (Batc (None, 64)                256       
_________________________________________________________________
dense_171 (Dense)            (None, 32)                2080      
_________________________________________________________________
batch_normalization_68 (Batc (None, 32)                128       
```

```
_________________________________________________________________
dense_172 (Dense)            (None, 10)                330       
=================================================================
Total params: 580,778
Trainable params: 578,794
Non-trainable params: 1,984
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 17s 291us/step - loss: 0.5734 - acc: 0.8261 - val_l
oss: 0.1564 - val_acc: 0.9542
Epoch 2/20
60000/60000 [==============================] - 9s 145us/step - loss: 0.2228 - acc: 0.9344 -
val_loss: 0.1220 - val_acc: 0.9634
Epoch 3/20
60000/60000 [==============================] - 9s 153us/step - loss: 0.1739 - acc: 0.9496 -
val_loss: 0.1009 - val_acc: 0.9690
Epoch 4/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.1416 - acc: 0.9580 -
val_loss: 0.0903 - val_acc: 0.9737
Epoch 5/20
60000/60000 [==============================] - 9s 145us/step - loss: 0.1279 - acc: 0.9622 -
val_loss: 0.0767 - val_acc: 0.9766
Epoch 6/20
60000/60000 [==============================] - 9s 145us/step - loss: 0.1174 - acc: 0.9654 -
val_loss: 0.0806 - val_acc: 0.9762
Epoch 7/20
60000/60000 [==============================] - 9s 145us/step - loss: 0.1058 - acc: 0.9683 -
val_loss: 0.0778 - val_acc: 0.9760
Epoch 8/20
60000/60000 [==============================] - 9s 144us/step - loss: 0.0971 - acc: 0.9710 -
val_loss: 0.0673 - val_acc: 0.9793
Epoch 9/20
60000/60000 [==============================] - 9s 144us/step - loss: 0.0893 - acc: 0.9730 -
val_loss: 0.0737 - val_acc: 0.9785
Epoch 10/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0840 - acc: 0.9748 -
val_loss: 0.0634 - val_acc: 0.9824
Epoch 11/20
60000/60000 [==============================] - 9s 144us/step - loss: 0.0816 - acc: 0.9757 -
val_loss: 0.0611 - val_acc: 0.9824
Epoch 12/20
60000/60000 [==============================] - 9s 145us/step - loss: 0.0751 - acc: 0.9776 -
val_loss: 0.0667 - val_acc: 0.9811
Epoch 13/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0715 - acc: 0.9783 -
val_loss: 0.0640 - val_acc: 0.9810
Epoch 14/20
60000/60000 [==============================] - 9s 143us/step - loss: 0.0658 - acc: 0.9798 -
val_loss: 0.0632 - val_acc: 0.9824
Epoch 15/20
60000/60000 [==============================] - 9s 150us/step - loss: 0.0665 - acc: 0.9799 -
val_loss: 0.0613 - val_acc: 0.9835
Epoch 16/20
60000/60000 [==============================] - 9s 145us/step - loss: 0.0623 - acc: 0.9809 -
val_loss: 0.0619 - val_acc: 0.9819
Epoch 17/20
60000/60000 [==============================] - 9s 145us/step - loss: 0.0597 - acc: 0.9822 -
val_loss: 0.0571 - val_acc: 0.9836
Epoch 18/20
60000/60000 [==============================] - 9s 144us/step - loss: 0.0577 - acc: 0.9822 -
val_loss: 0.0557 - val_acc: 0.9836
Epoch 19/20
60000/60000 [==============================] - 9s 145us/step - loss: 0.0558 - acc: 0.9826 -
val_loss: 0.0590 - val_acc: 0.9833
Epoch 20/20
60000/60000 [==============================] - 9s 143us/step - loss: 0.0522 - acc: 0.9837 -
val_loss: 0.0556 - val_acc: 0.9836
```

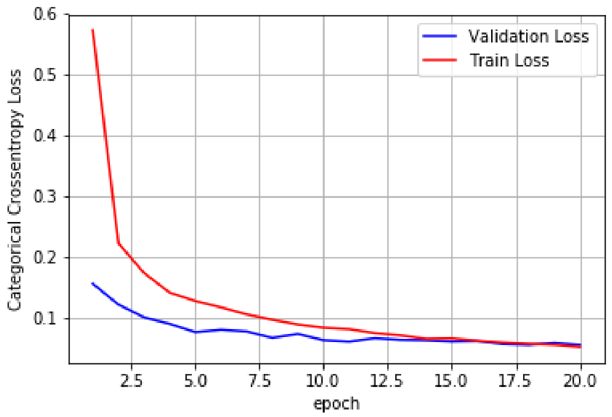
In [138]:

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

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

```
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, va
lidation_data=(X_test, Y_test))

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

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


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

```
Test score: 0.055616749329864976
Test accuracy: 0.9836
```

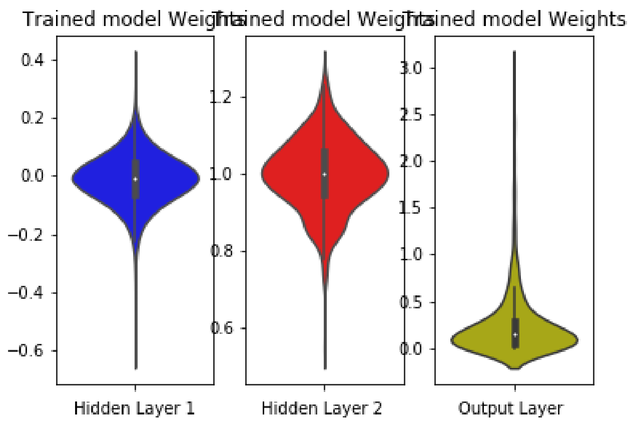
```
/opt/conda/lib/python3.6/site-packages/matplotlib/pyplot.py:514: RuntimeWarning: More than 20
figures have been opened. Figures created through the pyplot interface
(`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume too much memory.
(To control this warning, see the rcParam `figure.max_open_warning`).
  max_open_warning, RuntimeWarning)
```



In [139]:

```
w_after = model_relu.get_weights()

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


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

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

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

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



12. Model 4: MLP + BatchNormalization + Dropout (0.30)

- #layers: 5
- activation: sigmoid
- Weight Initializer: RandomNormal
- Optimizer: ADAM

In [140]:

```
model_relu = Sequential()
model_relu.add(Dense(512, activation='sigmoid', input_shape=(input_dim,), kernel_initializer=Random
Normal(mean=0.0, stddev=0.039, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.3))
model_relu.add(Dense(256, activation='sigmoid', kernel_initializer=RandomNormal(mean=0.0, stddev=0.
051, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.3))
model_relu.add(Dense(128, activation='sigmoid', kernel_initializer=RandomNormal(mean=0.0, stddev=0.
072, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.3))
model_relu.add(Dense(64, activation='sigmoid', kernel_initializer=RandomNormal(mean=0.0, stddev=0.1
02, seed=None)) )
model_relu.add(BatchNormalization())
model_relu.add(Dense(32, activation='sigmoid', kernel_initializer=RandomNormal(mean=0.0, stddev=0.1
44, seed=None)) )
model_relu.add(BatchNormalization())
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_173 (Dense)            (None, 512)               401920
_________________________________________________________________
batch_normalization_69 (Batc (None, 512)               2048
_________________________________________________________________
dropout_54 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_174 (Dense)            (None, 256)               131328
_________________________________________________________________
batch_normalization_70 (Batc (None, 256)               1024
_________________________________________________________________
dropout_55 (Dropout)         (None, 256)               0
```

```
dropout_55 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_175 (Dense)            (None, 128)               32896
_________________________________________________________________
batch_normalization_71 (Batc (None, 128)               512
_________________________________________________________________
dropout_56 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_176 (Dense)            (None, 64)                8256
_________________________________________________________________
batch_normalization_72 (Batc (None, 64)                256
_________________________________________________________________
dense_177 (Dense)            (None, 32)                2080
_________________________________________________________________
batch_normalization_73 (Batc (None, 32)                128
_________________________________________________________________
dense_178 (Dense)            (None, 10)                330
=================================================================
Total params: 580,778
Trainable params: 578,794
Non-trainable params: 1,984
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 18s 303us/step - loss: 0.4027 - acc: 0.8826 - val_l
oss: 0.2217 - val_acc: 0.9337
Epoch 2/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.2438 - acc: 0.9262 -
val_loss: 0.1567 - val_acc: 0.9532
Epoch 3/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.1918 - acc: 0.9430 -
val_loss: 0.1241 - val_acc: 0.9624
Epoch 4/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.1679 - acc: 0.9493 -
val_loss: 0.1181 - val_acc: 0.9643
Epoch 5/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.1455 - acc: 0.9564 -
val_loss: 0.1064 - val_acc: 0.9688
Epoch 6/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.1307 - acc: 0.9600 -
val_loss: 0.0982 - val_acc: 0.9722
Epoch 7/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.1157 - acc: 0.9651 -
val_loss: 0.0866 - val_acc: 0.9719
Epoch 8/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.1076 - acc: 0.9669 -
val_loss: 0.0824 - val_acc: 0.9745
Epoch 9/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.0948 - acc: 0.9706 -
val_loss: 0.0808 - val_acc: 0.9747
Epoch 10/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.0923 - acc: 0.9713 -
val_loss: 0.0797 - val_acc: 0.9771
Epoch 11/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.0816 - acc: 0.9751 -
val_loss: 0.0727 - val_acc: 0.9778
Epoch 12/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0795 - acc: 0.9761 -
val_loss: 0.0691 - val_acc: 0.9799
Epoch 13/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0761 - acc: 0.9766 -
val_loss: 0.0670 - val_acc: 0.9799
Epoch 14/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.0684 - acc: 0.9792 -
val_loss: 0.0659 - val_acc: 0.9800
Epoch 15/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0652 - acc: 0.9793 -
val_loss: 0.0674 - val_acc: 0.9807
Epoch 16/20
60000/60000 [==============================] - 10s 161us/step - loss: 0.0631 - acc: 0.9801 - val_l
oss: 0.0662 - val_acc: 0.9812
Epoch 17/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0575 - acc: 0.9820 -
val_loss: 0.0635 - val_acc: 0.9819
Epoch 18/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0574 - acc: 0.9819 -
val_loss: 0.0643 - val_acc: 0.9805
```

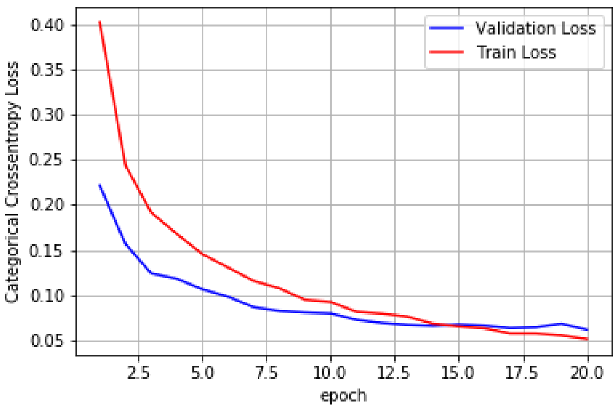
```
val_loss: 0.0645 - val_acc: 0.9805
Epoch 19/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.0553 - acc: 0.9826 -
val_loss: 0.0682 - val_acc: 0.9813
Epoch 20/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.0513 - acc: 0.9840 -
val_loss: 0.0617 - val_acc: 0.9820
```

In [141]:

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

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

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

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

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

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


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

```
Test score: 0.06166264152140356
Test accuracy: 0.982
```

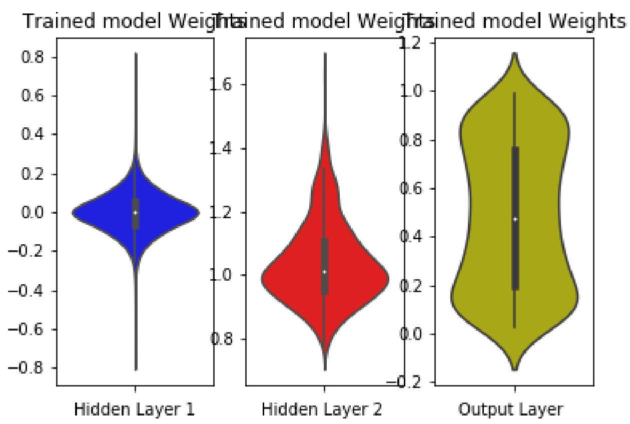
```
/opt/conda/lib/python3.6/site-packages/matplotlib/pyplot.py:514: RuntimeWarning: More than 20
figures have been opened. Figures created through the pyplot interface
(`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume too much memory.
(To control this warning, see the rcParam `figure.max_open_warning`).
  max_open_warning, RuntimeWarning)
```



In [142]:

```
w_after = model_relu.get_weights()

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

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

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

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



13. Model 5: MLP + BatchNormalization + Dropout (0.30)

- #layers: 5
- activation: sigmoid
- Weight Initializer: RandomNormal
- Optimizer: adadelta

In [143]:

```
model_relu = Sequential()
model_relu.add(Dense(512, activation='sigmoid', input_shape=(input_dim,), kernel_initializer=Random
Normal(mean=0.0, stddev=0.039, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.3))
model_relu.add(Dense(256, activation='sigmoid', kernel_initializer=RandomNormal(mean=0.0, stddev=0.
051, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.3))
model_relu.add(Dense(128, activation='sigmoid', kernel_initializer=RandomNormal(mean=0.0, stddev=0.
072, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.3))
model_relu.add(Dense(64, activation='sigmoid', kernel_initializer=RandomNormal(mean=0.0, stddev=0.1
02, seed=None)) )
model_relu.add(BatchNormalization())
model_relu.add(Dense(32, activation='sigmoid', kernel_initializer=RandomNormal(mean=0.0, stddev=0.1
44, seed=None)) )
model_relu.add(BatchNormalization())
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_179 (Dense)            (None, 512)               401920    
_________________________________________________________________
batch_normalization_74 (Batc (None, 512)               2048      
_________________________________________________________________
dropout_57 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_180 (Dense)            (None, 256)               131328    
_________________________________________________________________
batch_normalization_75 (Batc (None, 256)               1024      
_________________________________________________________________
dropout_58 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_181 (Dense)            (None, 128)               32896     
_________________________________________________________________
batch_normalization_76 (Batc (None, 128)               512       
_________________________________________________________________
dropout_59 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_182 (Dense)            (None, 64)                8256      
_________________________________________________________________
batch_normalization_77 (Batc (None, 64)                256       
_________________________________________________________________
dense_183 (Dense)            (None, 32)                2080      
_________________________________________________________________
batch_normalization_78 (Batc (None, 32)                128       
_________________________________________________________________
dense_184 (Dense)            (None, 10)                330       
=================================================================
Total params: 580,778
Trainable params: 578,794
Non-trainable params: 1,984
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 19s 316us/step - loss: 0.4114 - acc: 0.8797 - val_l
oss: 0.2380 - val_acc: 0.9287
Epoch 2/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.2509 - acc: 0.9246 - 
val_loss: 0.1653 - val_acc: 0.9520
Epoch 3/20
60000/60000 [==============================] - 9s 157us/step - loss: 0.2036 - acc: 0.9381 - 
val_loss: 0.1451 - val_acc: 0.9578
Epoch 4/20
60000/60000 [==============================] - 9s 154us/step - loss: 0.1751 - acc: 0.9468 - 
val_loss: 0.1324 - val_acc: 0.9632
Epoch 5/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.1528 - acc: 0.9545 - 
val_loss: 0.1179 - val_acc: 0.9650
Epoch 6/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.1415 - acc: 0.9574 - 
val_loss: 0.1063 - val_acc: 0.9698
Epoch 7/20
60000/60000 [==============================] - 9s 154us/step - loss: 0.1299 - acc: 0.9610 - 
val_loss: 0.0977 - val_acc: 0.9719
Epoch 8/20
60000/60000 [==============================] - 9s 155us/step - loss: 0.1196 - acc: 0.9635 - 
val_loss: 0.0920 - val_acc: 0.9732
Epoch 9/20
60000/60000 [==============================] - 9s 155us/step - loss: 0.1147 - acc: 0.9656 - 
val_loss: 0.0883 - val_acc: 0.9744
Epoch 10/20
60000/60000 [==============================] - 9s 155us/step - loss: 0.1083 - acc: 0.9668 - 
val_loss: 0.0860 - val_acc: 0.9748
Epoch 11/20
60000/60000 [==============================] - 9s 155us/step - loss: 0.0996 - acc: 0.9697 - 
val_loss: 0.0860 - val_acc: 0.9753
Epoch 12/20
60000/60000 [==============================] - 9s 153us/step - loss: 0.0948 - acc: 0.9703 - 
val_loss: 0.0866 - val_acc: 0.9762
Epoch 13/20
60000/60000 [==============================] - 9s 155us/step - loss: 0.0896 - acc: 0.9730 - 
val_loss: 0.0821 - val_acc: 0.9782
```

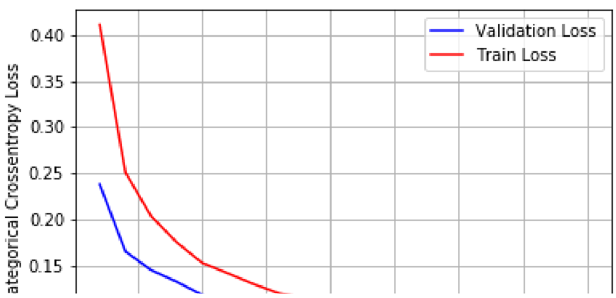
```
Epoch 14/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.0857 - acc: 0.9733 -
val_loss: 0.0798 - val_acc: 0.9774
Epoch 15/20
60000/60000 [==============================] - 9s 155us/step - loss: 0.0828 - acc: 0.9744 -
val_loss: 0.0741 - val_acc: 0.9786
Epoch 16/20
60000/60000 [==============================] - 9s 155us/step - loss: 0.0790 - acc: 0.9758 -
val_loss: 0.0696 - val_acc: 0.9793
Epoch 17/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.0752 - acc: 0.9759 -
val_loss: 0.0695 - val_acc: 0.9803
Epoch 18/20
60000/60000 [==============================] - 9s 154us/step - loss: 0.0721 - acc: 0.9767 -
val_loss: 0.0684 - val_acc: 0.9796
Epoch 19/20
60000/60000 [==============================] - 9s 157us/step - loss: 0.0682 - acc: 0.9791 -
val_loss: 0.0729 - val_acc: 0.9788
Epoch 20/20
60000/60000 [==============================] - 9s 155us/step - loss: 0.0675 - acc: 0.9789 -
val_loss: 0.0698 - val_acc: 0.9804
```

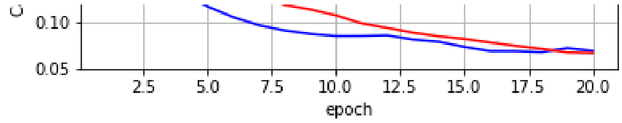
In [144]:

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

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

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

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

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

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


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

```
Test score: 0.06982594733461737
Test accuracy: 0.9804
```

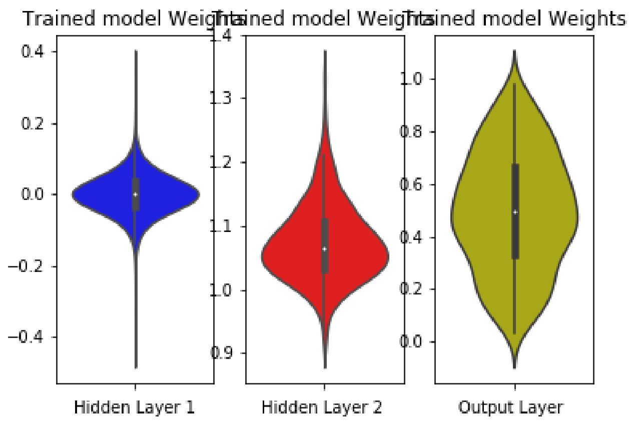
```
/opt/conda/lib/python3.6/site-packages/matplotlib/pyplot.py:514: RuntimeWarning: More than 20
figures have been opened. Figures created through the pyplot interface
(`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume too much memory.
(To control this warning, see the rcParam `figure.max_open_warning`).
  max_open_warning, RuntimeWarning)
```





In [145]:

```
w_after = model_relu.get_weights()

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


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

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

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



14. Model 6: MLP + BatchNormalization + Dropout (0.30)

- #layers: 5
- activation: tanh
- Weight Initializer: glorot_normal
- Optimizer: ADAM

In [146]:

```
model_relu = Sequential()
model_relu.add(Dense(512, activation='tanh', input_shape=(input_dim,), kernel_initializer=glorot_no
rmal()))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.3))
model_relu.add(Dense(256, activation='tanh', kernel_initializer=glorot_normal()))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.3))
model_relu.add(Dense(128, activation='tanh', kernel_initializer=glorot_normal()))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.3))
model_relu.add(Dense(64, activation='tanh', kernel_initializer=glorot_normal()))
model_relu.add(BatchNormalization())
```

```
model_relu.add(BatchNormalization())
model_relu.add(Dense(32, activation='tanh', kernel_initializer=glorot_normal()))
model_relu.add(BatchNormalization())
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_185 (Dense)            (None, 512)               401920    
_________________________________________________________________
batch_normalization_79 (Batc (None, 512)               2048      
_________________________________________________________________
dropout_60 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_186 (Dense)            (None, 256)               131328    
_________________________________________________________________
batch_normalization_80 (Batc (None, 256)               1024      
_________________________________________________________________
dropout_61 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_187 (Dense)            (None, 128)               32896     
_________________________________________________________________
batch_normalization_81 (Batc (None, 128)               512       
_________________________________________________________________
dropout_62 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_188 (Dense)            (None, 64)                8256      
_________________________________________________________________
batch_normalization_82 (Batc (None, 64)                256       
_________________________________________________________________
dense_189 (Dense)            (None, 32)                2080      
_________________________________________________________________
batch_normalization_83 (Batc (None, 32)                128       
_________________________________________________________________
dense_190 (Dense)            (None, 10)                330       
=================================================================
Total params: 580,778
Trainable params: 578,794
Non-trainable params: 1,984
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 19s 312us/step - loss: 0.4344 - acc: 0.8725 - val_l
oss: 0.2088 - val_acc: 0.9388
Epoch 2/20
60000/60000 [==============================] - 9s 145us/step - loss: 0.2380 - acc: 0.9296 -
val_loss: 0.1496 - val_acc: 0.9580
Epoch 3/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.1820 - acc: 0.9455 -
val_loss: 0.1261 - val_acc: 0.9636
Epoch 4/20
60000/60000 [==============================] - 9s 158us/step - loss: 0.1496 - acc: 0.9563 -
val_loss: 0.1042 - val_acc: 0.9712
Epoch 5/20
60000/60000 [==============================] - 9s 149us/step - loss: 0.1287 - acc: 0.9619 -
val_loss: 0.1016 - val_acc: 0.9722
Epoch 6/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.1195 - acc: 0.9648 -
val_loss: 0.0928 - val_acc: 0.9740
Epoch 7/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.1040 - acc: 0.9697 -
val_loss: 0.0841 - val_acc: 0.9737
Epoch 8/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0968 - acc: 0.9712 -
val_loss: 0.0780 - val_acc: 0.9763
Epoch 9/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.0900 - acc: 0.9734 -
val_loss: 0.0769 - val_acc: 0.9776
Epoch 10/20
```

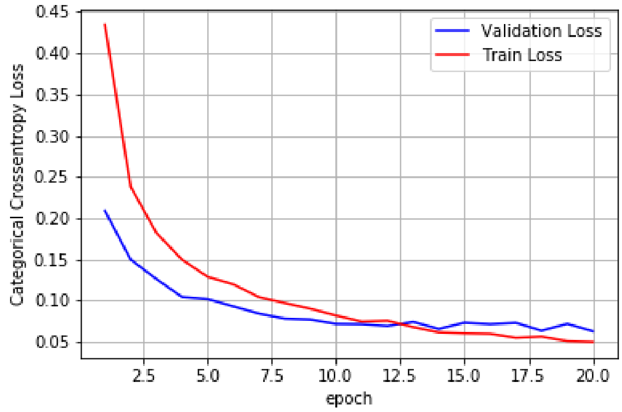
```
60000/60000 [==============================] - 9s 146us/step - loss: 0.0818 - acc: 0.9751 -
val_loss: 0.0716 - val_acc: 0.9796
Epoch 11/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0744 - acc: 0.9770 -
val_loss: 0.0713 - val_acc: 0.9801
Epoch 12/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0755 - acc: 0.9772 -
val_loss: 0.0691 - val_acc: 0.9818
Epoch 13/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.0677 - acc: 0.9797 -
val_loss: 0.0741 - val_acc: 0.9814
Epoch 14/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.0613 - acc: 0.9815 -
val_loss: 0.0655 - val_acc: 0.9832
Epoch 15/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0602 - acc: 0.9821 -
val_loss: 0.0731 - val_acc: 0.9819
Epoch 16/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.0596 - acc: 0.9817 -
val_loss: 0.0713 - val_acc: 0.9821
Epoch 17/20
60000/60000 [==============================] - 9s 149us/step - loss: 0.0548 - acc: 0.9834 -
val_loss: 0.0730 - val_acc: 0.9814
Epoch 18/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.0562 - acc: 0.9825 -
val_loss: 0.0634 - val_acc: 0.9829
Epoch 19/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.0508 - acc: 0.9847 -
val_loss: 0.0717 - val_acc: 0.9822
Epoch 20/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.0498 - acc: 0.9849 -
val_loss: 0.0628 - val_acc: 0.9843
```

In [147]:

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

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

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

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

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

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


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

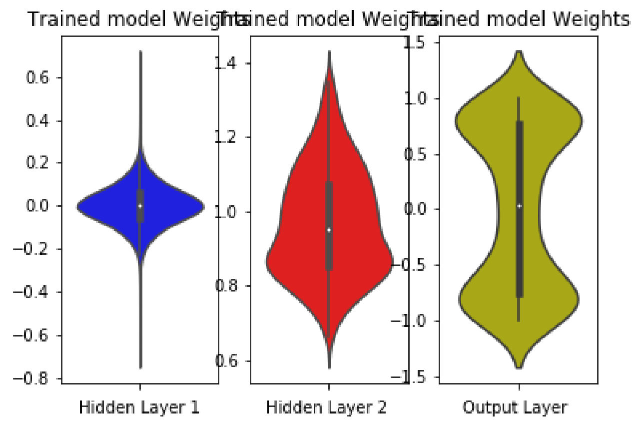
```
Test score: 0.06275533262640237
Test accuracy: 0.9843
```

```
/opt/conda/lib/python3.6/site-packages/matplotlib/pyplot.py:514: RuntimeWarning: More than 20
figures have been opened. Figures created through the pyplot interface
(`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume too much memory.
(To control this warning, see the rcParam `figure.max_open_warning`).
  max_open_warning, RuntimeWarning)
```



In [148]:

```
w_after = model_relu.get_weights()

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


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

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

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



Feedback Conclusion:

- I have used Kaggle platform to do this assignment as I found that kaggle is much much faster than Google colab.
- I have trained MLP Models with 2, 3 and 5 layers.
- I have used RandomNormal, He Normal and Glorot Normal weight initialization.
- I have used ReLU, sigmoid and tanh activation function.
- I have used AdaDelta and ADAM as optimizer.
- ADAM is faster than AdaDelta.
- I have also used BatchNormalization and Dropout.

In [149]:

```
table = PrettyTable()
table.field_names = ['Model #', 'Batch Normalization', 'Dropout + Value', 'Activation', 'Initialize
r', 'Optimizer', 'Accuracy']
table.add_row([1, "Yes", "Yes, 0.3", "ReLU", "RandomNormal", "ADAM", 0.9832])
table.add_row([2, "No", "Yes, 0.3", "ReLU", "He Normal", "ADAM", 0.9833])
table.add_row([3, "Yes", "Yes, 0.4", "ReLU", "RandomNormal", "ADAM", 0.9836])
table.add_row([4, "Yes", "Yes, 0.3", "sigmoid", "RandomNormal", "ADAM", 0.9820])
table.add_row([5, "Yes", "Yes, 0.3", "sigmoid", "RandomNormal", "AdaDelta", 0.9804])
table.add_row([6, "Yes", "Yes, 0.3", "tanh", "Glorot Normal", "ADAM", 0.9848])
print(table)
```

```
+---------+---------------------+-----------------+------------+---------------+----------+-------
+
| Model # | Batch Normalization | Dropout + Value | Activation |  Initializer  | Optimizer |
Accuracy |
+---------+---------------------+-----------------+------------+---------------+----------+-------
+
|    1    |         Yes         |     Yes, 0.3    |    ReLU    |  RandomNormal |   ADAM   |  0.983
|
|    2    |          No         |     Yes, 0.3    |    ReLU    |   He Normal   |   ADAM   |  0.983
|
|    3    |         Yes         |     Yes, 0.4    |    ReLU    |  RandomNormal |   ADAM   |  0.983
|
|    4    |         Yes         |     Yes, 0.3    |  sigmoid   |  RandomNormal |   ADAM   |  0.982
|
|    5    |         Yes         |     Yes, 0.3    |  sigmoid   |  RandomNormal | AdaDelta |  0.980
|
|    6    |         Yes         |     Yes, 0.3    |    tanh    | Glorot Normal |   ADAM   |  0.984
|
+---------+---------------------+-----------------+------------+---------------+----------+-------
+
```