Keras -- MLPs on MNIST

In [0]:

```
# 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
%matplotlib notebook
```

In [0]:

```
%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 [6]:

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

```
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 1s 0us/step
```

In [7]:

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

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

In [0]:

```
# 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 [9]:

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

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

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

In [10]:

```
# 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 [0]:

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

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

In [12]:

```
# 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 [13]:

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

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

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

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

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

Softmax classifier

In [0]:

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

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

# 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_un
iform',
# bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=None, activity_regu
larizer=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 a
rgument supported by all forward layers:

# from keras.layers import Activation, Dense

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

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

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


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

In [0]:

```
# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 20
```

In [16]:

```
# 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'))
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_b
ackend.py:66: The name tf.get_default_graph is deprecated. Please use tf.compat.v1.get_de
fault_graph instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_b
ackend.py:541: The name tf.placeholder is deprecated. Please use tf.compat.v1.placeholder
instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_b
ackend.py:4432: The name tf.random_uniform is deprecated. Please use tf.random.uniform in
stead.
```

In [17]:

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

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


# Note: when using the categorical_crossentropy loss, your targets should be in categoric
al format
# (e.g. if you have 10 classes, the target for each sample should be a 10-dimensional vec
tor 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, validat
ion_split=0.0,
# validation_data=None, shuffle=True, class_weight=None, sample_weight=None, initial_epoc
h=0, steps_per_epoch=None,
# validation_steps=None)

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

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

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

history1 = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1
, validation_data=(X_test, Y_test))
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimizers.py:793: T
he name tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead
.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_b
ackend.py:3576: The name tf.log is deprecated. Please use tf.math.log instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/ops
/math_grad.py:1424: where (from tensorflow.python.ops.array_ops) is deprecated and will b
e removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_b
ackend.py:1033: The name tf.assign_add is deprecated. Please use tf.compat.v1.assign_add
instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_b
ackend.py:1020: The name tf.assign is deprecated. Please use tf.compat.v1.assign instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_b
ackend.py:3005: The name tf.Session is deprecated. Please use tf.compat.v1.Session instea
d.

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_b
ackend.py:190: The name tf.get_default_session is deprecated. Please use tf.compat.v1.get
_default_session instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_b
ackend.py:197: The name tf.ConfigProto is deprecated. Please use tf.compat.v1.ConfigProto
instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_b
ackend.py:207: The name tf.global_variables is deprecated. Please use tf.compat.v1.global
_variables instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_b
ackend.py:216: The name tf.is_variable_initialized is deprecated. Please use tf.compat.v1
.is_variable_initialized instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_b
ackend.py:223: The name tf.variables_initializer is deprecated. Please use tf.compat.v1.v
ariables_initializer instead.

60000/60000 [==============================] - 11s 188us/step - loss: 1.3074 - acc: 0.684
8 - val_loss: 0.8180 - val_acc: 0.8364
Epoch 2/20
60000/60000 [==============================] - 1s 24us/step - loss: 0.7217 - acc: 0.8404
- val_loss: 0.6102 - val_acc: 0.8618
Epoch 3/20
60000/60000 [==============================] - 1s 25us/step - loss: 0.5905 - acc: 0.8580
- val_loss: 0.5276 - val_acc: 0.8735
Epoch 4/20
60000/60000 [==============================] - 2s 25us/step - loss: 0.5279 - acc: 0.8671
- val_loss: 0.4818 - val_acc: 0.8802
Epoch 5/20
60000/60000 [==============================] - 2s 27us/step - loss: 0.4898 - acc: 0.8738
- val_loss: 0.4517 - val_acc: 0.8860
Epoch 6/20
60000/60000 [==============================] - 2s 25us/step - loss: 0.4637 - acc: 0.8789
- val_loss: 0.4301 - val_acc: 0.8895
Epoch 7/20
60000/60000 [==============================] - 2s 27us/step - loss: 0.4443 - acc: 0.8830
- val_loss: 0.4138 - val_acc: 0.8918
Epoch 8/20
60000/60000 [==============================] - 1s 23us/step - loss: 0.4292 - acc: 0.8853
- val_loss: 0.4012 - val_acc: 0.8937
Epoch 9/20
60000/60000 [==============================] - 2s 25us/step - loss: 0.4170 - acc: 0.8881
- val_loss: 0.3906 - val_acc: 0.8962
Epoch 10/20
```

```
60000/60000 [==============================] - 1s 24us/step - loss: 0.4070 - acc: 0.8899
- val_loss: 0.3817 - val_acc: 0.8979
Epoch 11/20
60000/60000 [==============================] - 2s 25us/step - loss: 0.3984 - acc: 0.8918
- val_loss: 0.3745 - val_acc: 0.8985
Epoch 12/20
60000/60000 [==============================] - 2s 25us/step - loss: 0.3910 - acc: 0.8933
- val_loss: 0.3678 - val_acc: 0.9004
Epoch 13/20
60000/60000 [==============================] - 2s 26us/step - loss: 0.3846 - acc: 0.8950
- val_loss: 0.3624 - val_acc: 0.9016
Epoch 14/20
60000/60000 [==============================] - 2s 25us/step - loss: 0.3789 - acc: 0.8961
- val_loss: 0.3573 - val_acc: 0.9032
Epoch 15/20
60000/60000 [==============================] - 1s 24us/step - loss: 0.3738 - acc: 0.8973
- val_loss: 0.3532 - val_acc: 0.9045
Epoch 16/20
60000/60000 [==============================] - 1s 24us/step - loss: 0.3693 - acc: 0.8980
- val_loss: 0.3492 - val_acc: 0.9055
Epoch 17/20
60000/60000 [==============================] - 1s 24us/step - loss: 0.3651 - acc: 0.8995
- val_loss: 0.3454 - val_acc: 0.9071
Epoch 18/20
60000/60000 [==============================] - 2s 25us/step - loss: 0.3613 - acc: 0.9001
- val_loss: 0.3421 - val_acc: 0.9075
Epoch 19/20
60000/60000 [==============================] - 2s 26us/step - loss: 0.3579 - acc: 0.9011
- val_loss: 0.3389 - val_acc: 0.9093
Epoch 20/20
60000/60000 [==============================] - 2s 26us/step - loss: 0.3547 - acc: 0.9019
- val_loss: 0.3363 - val_acc: 0.9093
```

In [0]:

```
%matplotlib inline
```

In [0]:

```
# 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.grid()
  fig.canvas.draw()
```

In [20]:

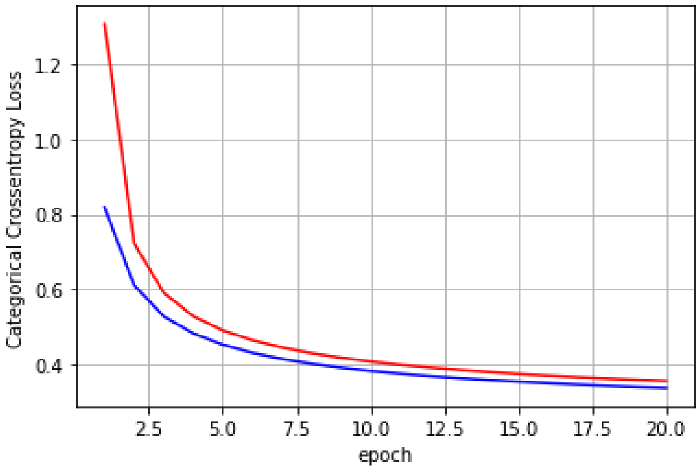
```
print(len(history1.history['val_loss']))
```

```
20
```

In [21]:

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

fig,ax1 = plt.subplots(1,1)
ax1.set_xlabel('epoch') ; ax1.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x1 = list(range(1,nb_epoch+1))

vy1 = history1.history['val_loss']
ty1 = history1.history['loss']
ax1.plot(x1, vy1, 'b', label="Validation Loss")
ax1.plot(x1, ty1, 'r', label="Train Loss")
plt.grid()
fig.canvas.draw()
plt.show();
```

```
Test score: 0.3363473850727081
Test accuracy: 0.9093
```



MLP + Sigmoid activation + SGDOptimizer

In [22]:

```
# Multilayer perceptron

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

model_sigmoid.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_2 (Dense)              (None, 512)               401920    
_________________________________________________________________
dense_3 (Dense)              (None, 128)               65664     
_________________________________________________________________
dense_4 (Dense)              (None, 10)                1290      
=================================================================
Total params: 468,874
Trainable params: 468,874
Non-trainable params: 0
_________________________________________________________________
```

In [23]:

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

history = model_sigmoid.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ve
rbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 2s 32us/step - loss: 2.2707 - acc: 0.2212
- val_loss: 2.2289 - val_acc: 0.3618
Epoch 2/20
60000/60000 [==============================] - 2s 30us/step - loss: 2.1886 - acc: 0.4307
- val_loss: 2.1386 - val_acc: 0.5356
Epoch 3/20
60000/60000 [==============================] - 2s 29us/step - loss: 2.0813 - acc: 0.5460
- val_loss: 2.0068 - val_acc: 0.5419
Epoch 4/20
60000/60000 [==============================] - 2s 29us/step - loss: 1.9258 - acc: 0.6055
- val_loss: 1.8199 - val_acc: 0.6352
Epoch 5/20
60000/60000 [==============================] - 2s 27us/step - loss: 1.7172 - acc: 0.6523
```

```
- val_loss: 1.5887 - val_acc: 0.6586
Epoch 6/20
60000/60000 [==============================] - 2s 30us/step - loss: 1.4841 - acc: 0.6941
- val_loss: 1.3559 - val_acc: 0.7194
Epoch 7/20
60000/60000 [==============================] - 2s 31us/step - loss: 1.2701 - acc: 0.7367
- val_loss: 1.1605 - val_acc: 0.7719
Epoch 8/20
60000/60000 [==============================] - 2s 32us/step - loss: 1.0970 - acc: 0.7690
- val_loss: 1.0090 - val_acc: 0.7783
Epoch 9/20
60000/60000 [==============================] - 2s 29us/step - loss: 0.9639 - acc: 0.7894
- val_loss: 0.8934 - val_acc: 0.8062
Epoch 10/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.8622 - acc: 0.8055
- val_loss: 0.8050 - val_acc: 0.8169
Epoch 11/20
60000/60000 [==============================] - 2s 29us/step - loss: 0.7834 - acc: 0.8178
- val_loss: 0.7349 - val_acc: 0.8304
Epoch 12/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.7211 - acc: 0.8277
- val_loss: 0.6792 - val_acc: 0.8370
Epoch 13/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.6708 - acc: 0.8368
- val_loss: 0.6346 - val_acc: 0.8457
Epoch 14/20
60000/60000 [==============================] - 2s 29us/step - loss: 0.6294 - acc: 0.8438
- val_loss: 0.5963 - val_acc: 0.8522
Epoch 15/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.5951 - acc: 0.8514
- val_loss: 0.5647 - val_acc: 0.8592
Epoch 16/20
60000/60000 [==============================] - 2s 30us/step - loss: 0.5659 - acc: 0.8566
- val_loss: 0.5381 - val_acc: 0.8633
Epoch 17/20
60000/60000 [==============================] - 2s 30us/step - loss: 0.5410 - acc: 0.8615
- val_loss: 0.5145 - val_acc: 0.8693
Epoch 18/20
60000/60000 [==============================] - 2s 30us/step - loss: 0.5194 - acc: 0.8657
- val_loss: 0.4942 - val_acc: 0.8716
Epoch 19/20
60000/60000 [==============================] - 2s 29us/step - loss: 0.5006 - acc: 0.8698
- val_loss: 0.4775 - val_acc: 0.8746
Epoch 20/20
60000/60000 [==============================] - 2s 30us/step - loss: 0.4842 - acc: 0.8727
- val_loss: 0.4614 - val_acc: 0.8786
```

In [0]:

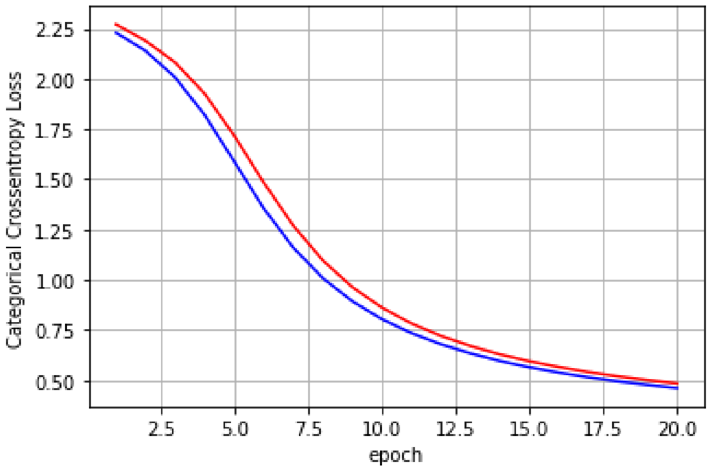
```
# list of epoch numbers


# 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, verb
ose=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 epoc
hs
```

In [24]:

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


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

```
Test score: 0.4614129983663559
Test accuracy: 0.8786
```



In [0]:

```

```

In [0]:

```
import matplotlib.pyplot as plt
```

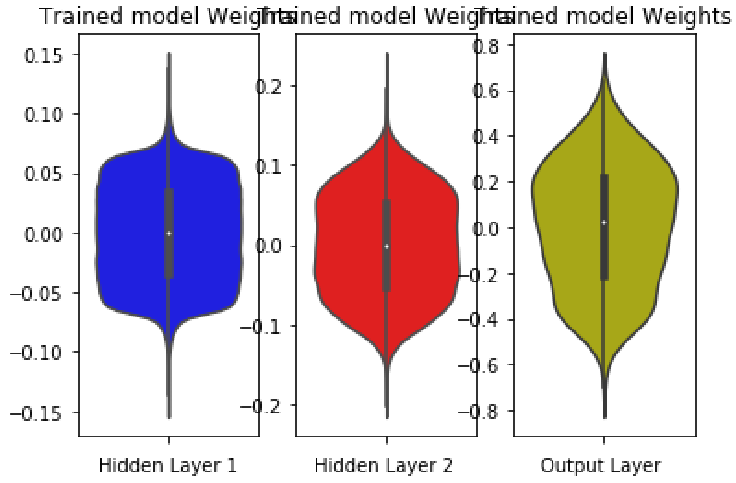
In [0]:

```
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)
```

In [27]:

```
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 [28]:

```
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=['accur
acy'])

history = model_sigmoid.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ve
rbose=1, validation_data=(X_test, Y_test))
```

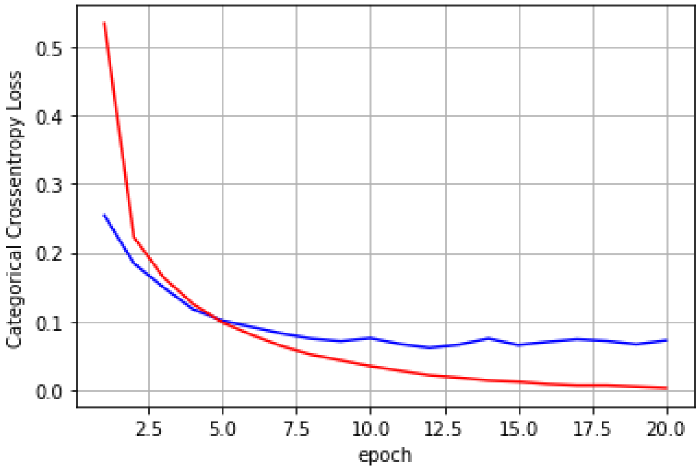
```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 512)               401920
_________________________________________________________________
dense_6 (Dense)              (None, 128)               65664
_________________________________________________________________
dense_7 (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 [==============================] - 2s 40us/step - loss: 0.5336 - acc: 0.8603
- val_loss: 0.2545 - val_acc: 0.9255
Epoch 2/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.2227 - acc: 0.9348
- val_loss: 0.1845 - val_acc: 0.9438
Epoch 3/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.1642 - acc: 0.9505
- val_loss: 0.1497 - val_acc: 0.9532
Epoch 4/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.1257 - acc: 0.9629
- val_loss: 0.1179 - val_acc: 0.9645
Epoch 5/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.0986 - acc: 0.9709
- val_loss: 0.1013 - val_acc: 0.9690
Epoch 6/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0803 - acc: 0.9763
- val_loss: 0.0919 - val_acc: 0.9718
Epoch 7/20
```

```
Epoch 7/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0641 - acc: 0.9813
- val_loss: 0.0826 - val_acc: 0.9736
Epoch 8/20
60000/60000 [==============================] - 2s 37us/step - loss: 0.0519 - acc: 0.9848
- val_loss: 0.0752 - val_acc: 0.9771
Epoch 9/20
60000/60000 [==============================] - 2s 38us/step - loss: 0.0435 - acc: 0.9873
- val_loss: 0.0713 - val_acc: 0.9784
Epoch 10/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.0352 - acc: 0.9899
- val_loss: 0.0759 - val_acc: 0.9762
Epoch 11/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.0283 - acc: 0.9920
- val_loss: 0.0672 - val_acc: 0.9802
Epoch 12/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0217 - acc: 0.9947
- val_loss: 0.0616 - val_acc: 0.9820
Epoch 13/20
60000/60000 [==============================] - 2s 37us/step - loss: 0.0181 - acc: 0.9956
- val_loss: 0.0662 - val_acc: 0.9799
Epoch 14/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.0142 - acc: 0.9968
- val_loss: 0.0753 - val_acc: 0.9780
Epoch 15/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.0123 - acc: 0.9968
- val_loss: 0.0656 - val_acc: 0.9805
Epoch 16/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.0090 - acc: 0.9980
- val_loss: 0.0703 - val_acc: 0.9806
Epoch 17/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.0071 - acc: 0.9986
- val_loss: 0.0742 - val_acc: 0.9789
Epoch 18/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.0071 - acc: 0.9982
- val_loss: 0.0716 - val_acc: 0.9804
Epoch 19/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.0053 - acc: 0.9990
- val_loss: 0.0666 - val_acc: 0.9825
Epoch 20/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.0035 - acc: 0.9993
- val_loss: 0.0725 - val_acc: 0.9815
```

In [29]:

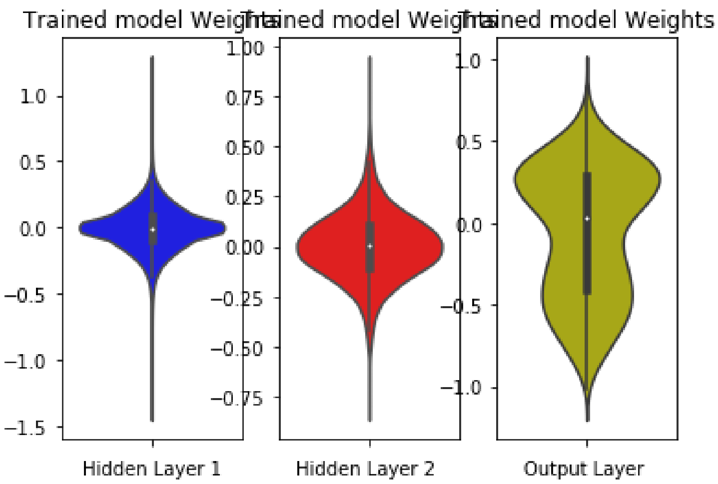
```
score = model_sigmoid.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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, verb
ose=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 epoc
hs

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

```
Test score: 0.07249096132973064
Test accuracy: 0.9815
```



In [30]:

```
w_after = model_sigmoid.get_weights()

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


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

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

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



MLP + ReLU +SGD

In [31]:

```
# 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
=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_relu.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, st
ddev=0.125, seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))

model_relu.summary()
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_b
ackend.py:4409: The name tf.random_normal is deprecated. Please use tf.random.normal inst
ead.

Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_8 (Dense)              (None, 512)               401920    
_________________________________________________________________
dense_9 (Dense)              (None, 128)               65664     
_________________________________________________________________
dense_10 (Dense)             (None, 10)                1290      
=================================================================
Total params: 468,874
Trainable params: 468,874
Non-trainable params: 0
_________________________________________________________________
```

In [32]:

```
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, verbo
se=1, validation_data=(X_test, Y_test))
```

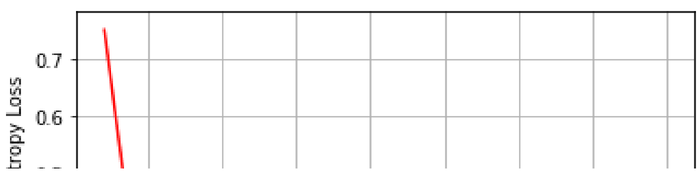
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.7522 - acc: 0.7912
- val_loss: 0.3851 - val_acc: 0.8953
Epoch 2/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.3552 - acc: 0.8993
- val_loss: 0.2979 - val_acc: 0.9158
Epoch 3/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.2924 - acc: 0.9173
- val_loss: 0.2600 - val_acc: 0.9264
Epoch 4/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.2585 - acc: 0.9269
- val_loss: 0.2405 - val_acc: 0.9308
Epoch 5/20
60000/60000 [==============================] - 2s 29us/step - loss: 0.2352 - acc: 0.9335
- val_loss: 0.2202 - val_acc: 0.9365
Epoch 6/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.2171 - acc: 0.9380
- val_loss: 0.2085 - val_acc: 0.9404
Epoch 7/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.2023 - acc: 0.9427
- val_loss: 0.1957 - val_acc: 0.9426
Epoch 8/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.1903 - acc: 0.9455
- val_loss: 0.1857 - val_acc: 0.9452
Epoch 9/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.1795 - acc: 0.9487
- val_loss: 0.1765 - val_acc: 0.9487
```

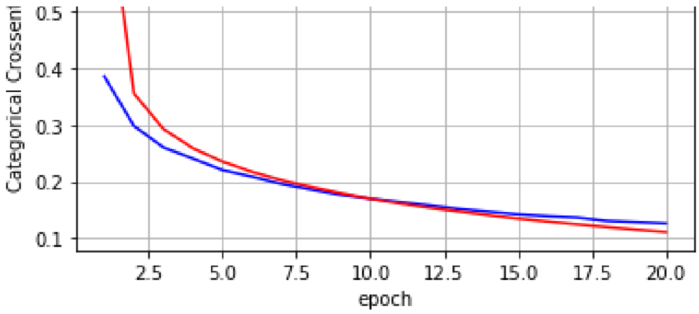
```
Epoch 10/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.1699 - acc: 0.9512
- val_loss: 0.1699 - val_acc: 0.9502
Epoch 11/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.1616 - acc: 0.9540
- val_loss: 0.1640 - val_acc: 0.9506
Epoch 12/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.1539 - acc: 0.9566
- val_loss: 0.1583 - val_acc: 0.9531
Epoch 13/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.1471 - acc: 0.9583
- val_loss: 0.1519 - val_acc: 0.9546
Epoch 14/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.1407 - acc: 0.9599
- val_loss: 0.1470 - val_acc: 0.9571
Epoch 15/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.1347 - acc: 0.9622
- val_loss: 0.1421 - val_acc: 0.9572
Epoch 16/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.1294 - acc: 0.9639
- val_loss: 0.1390 - val_acc: 0.9580
Epoch 17/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.1245 - acc: 0.9650
- val_loss: 0.1366 - val_acc: 0.9603
Epoch 18/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.1198 - acc: 0.9663
- val_loss: 0.1305 - val_acc: 0.9620
Epoch 19/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.1153 - acc: 0.9681
- val_loss: 0.1283 - val_acc: 0.9613
Epoch 20/20
60000/60000 [==============================] - 2s 30us/step - loss: 0.1112 - acc: 0.9693
- val_loss: 0.1261 - val_acc: 0.9617
```

In [33]:

```
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, verb
ose=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 epoc
hs

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

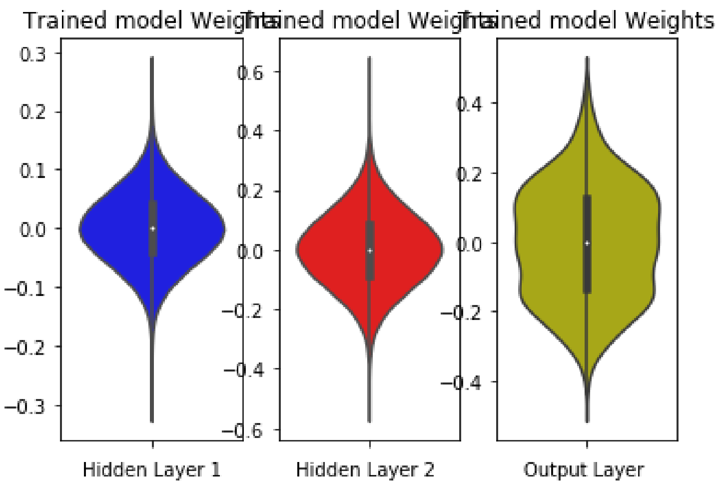
```
Test score: 0.12610071545392273
Test accuracy: 0.9617
```





In [34]:

```
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 [35]:

```
model_relu = Sequential()
model_relu.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer
=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_relu.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, st
ddev=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, verbo
se=1, validation_data=(X_test, Y_test))
```

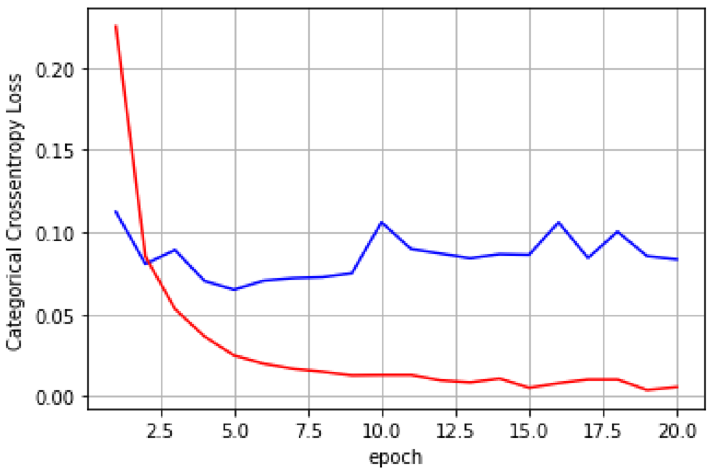
```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_11 (Dense)             (None, 512)               401920
_________________________________________________________________
dense_12 (Dense)             (None, 128)               65664
_________________________________________________________________
dense_13 (Dense)             (None, 10)                1290
=================================================================
Total params: 468,874
Trainable params: 468,874
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.2252 - acc: 0.9319
- val_loss: 0.1124 - val_acc: 0.9659
Epoch 2/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0852 - acc: 0.9742
- val_loss: 0.0806 - val_acc: 0.9741
Epoch 3/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.0533 - acc: 0.9832
- val_loss: 0.0892 - val_acc: 0.9715
Epoch 4/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.0365 - acc: 0.9883
- val_loss: 0.0703 - val_acc: 0.9764
Epoch 5/20
60000/60000 [==============================] - 2s 38us/step - loss: 0.0250 - acc: 0.9919
- val_loss: 0.0650 - val_acc: 0.9814
Epoch 6/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0201 - acc: 0.9935
- val_loss: 0.0704 - val_acc: 0.9804
Epoch 7/20
60000/60000 [==============================] - 2s 37us/step - loss: 0.0169 - acc: 0.9942
- val_loss: 0.0720 - val_acc: 0.9790
Epoch 8/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0151 - acc: 0.9952
- val_loss: 0.0727 - val_acc: 0.9800
Epoch 9/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0129 - acc: 0.9957
- val_loss: 0.0751 - val_acc: 0.9805
Epoch 10/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.0131 - acc: 0.9956
- val_loss: 0.1060 - val_acc: 0.9747
Epoch 11/20
60000/60000 [==============================] - 2s 38us/step - loss: 0.0131 - acc: 0.9954
- val_loss: 0.0898 - val_acc: 0.9775
Epoch 12/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.0099 - acc: 0.9969
- val_loss: 0.0870 - val_acc: 0.9798
Epoch 13/20
60000/60000 [==============================] - 2s 37us/step - loss: 0.0085 - acc: 0.9970
- val_loss: 0.0841 - val_acc: 0.9793
Epoch 14/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.0110 - acc: 0.9961
- val_loss: 0.0866 - val_acc: 0.9813
Epoch 15/20
60000/60000 [==============================] - 2s 37us/step - loss: 0.0053 - acc: 0.9984
- val_loss: 0.0862 - val_acc: 0.9809
Epoch 16/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0081 - acc: 0.9972
- val_loss: 0.1060 - val_acc: 0.9780
Epoch 17/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0104 - acc: 0.9966
- val_loss: 0.0843 - val_acc: 0.9818
Epoch 18/20
```

```
60000/60000 [==============================] - 2s 37us/step - loss: 0.0104 - acc: 0.9964
- val_loss: 0.1003 - val_acc: 0.9798
Epoch 19/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.0039 - acc: 0.9987
- val_loss: 0.0854 - val_acc: 0.9833
Epoch 20/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.0057 - acc: 0.9983
- val_loss: 0.0834 - val_acc: 0.9843
```

In [36]:

```
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, verb
ose=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 epoc
hs


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

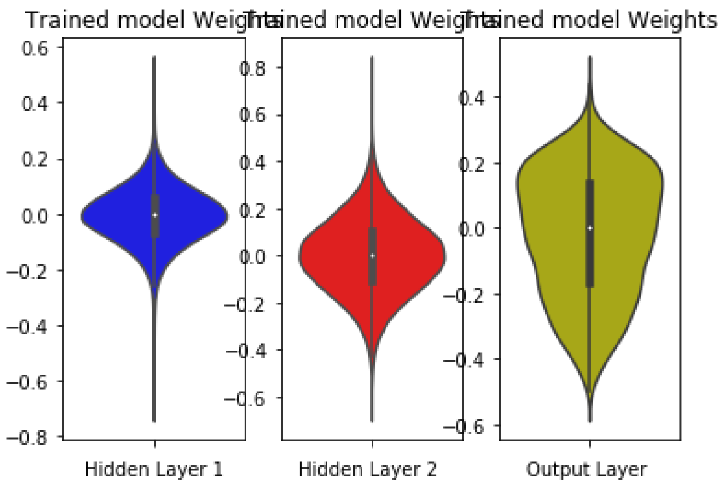
```
Test score: 0.08342993713870042
Test accuracy: 0.9843
```



In [37]:

```
w_after = model_relu.get_weights()

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


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

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

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

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



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

In [38]:

```
# 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_initial
izer=RandomNormal(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()
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_b
ackend.py:148: The name tf.placeholder_with_default is deprecated. Please use tf.compat.v
1.placeholder_with_default instead.

Model: "sequential_6"
_________________________________________________________________
```

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_14 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_1 (Batch (None, 512)               2048
_________________________________________________________________
dense_15 (Dense)             (None, 128)               65664
_________________________________________________________________
batch_normalization_2 (Batch (None, 128)               512
_________________________________________________________________
dense_16 (Dense)             (None, 10)                1290
=================================================================
Total params: 471,434
Trainable params: 470,154
Non-trainable params: 1,280
_________________________________________________________________
```

In [39]:

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

history = model_batch.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verb
ose=1, validation_data=(X_test, Y_test))
```

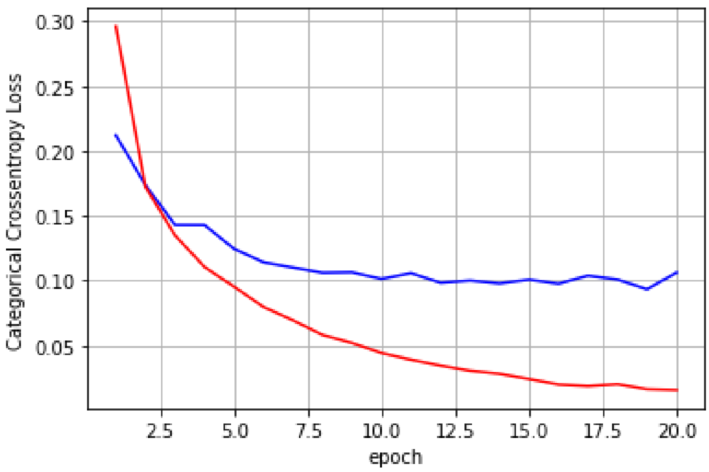
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.2960 - acc: 0.9113
- val_loss: 0.2120 - val_acc: 0.9373
Epoch 2/20
60000/60000 [==============================] - 4s 59us/step - loss: 0.1723 - acc: 0.9495
- val_loss: 0.1736 - val_acc: 0.9471
Epoch 3/20
60000/60000 [==============================] - 4s 62us/step - loss: 0.1349 - acc: 0.9607
- val_loss: 0.1430 - val_acc: 0.9574
Epoch 4/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.1106 - acc: 0.9673
- val_loss: 0.1429 - val_acc: 0.9568
Epoch 5/20
60000/60000 [==============================] - 3s 58us/step - loss: 0.0953 - acc: 0.9712
- val_loss: 0.1245 - val_acc: 0.9624
Epoch 6/20
60000/60000 [==============================] - 3s 56us/step - loss: 0.0797 - acc: 0.9758
- val_loss: 0.1142 - val_acc: 0.9648
Epoch 7/20
60000/60000 [==============================] - 4s 61us/step - loss: 0.0694 - acc: 0.9791
- val_loss: 0.1101 - val_acc: 0.9648
Epoch 8/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.0581 - acc: 0.9823
- val_loss: 0.1061 - val_acc: 0.9697
Epoch 9/20
60000/60000 [==============================] - 4s 59us/step - loss: 0.0519 - acc: 0.9839
- val_loss: 0.1064 - val_acc: 0.9675
Epoch 10/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0442 - acc: 0.9860
- val_loss: 0.1013 - val_acc: 0.9693
Epoch 11/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.0390 - acc: 0.9878
- val_loss: 0.1058 - val_acc: 0.9690
Epoch 12/20
60000/60000 [==============================] - 4s 61us/step - loss: 0.0346 - acc: 0.9891
- val_loss: 0.0983 - val_acc: 0.9709
Epoch 13/20
60000/60000 [==============================] - 3s 58us/step - loss: 0.0305 - acc: 0.9898
- val_loss: 0.1001 - val_acc: 0.9730
Epoch 14/20
60000/60000 [==============================] - 4s 61us/step - loss: 0.0282 - acc: 0.9910
- val_loss: 0.0978 - val_acc: 0.9735
Epoch 15/20
60000/60000 [==============================] - 3s 57us/step - loss: 0.0241 - acc: 0.9928
- val_loss: 0.1007 - val_acc: 0.9724
Epoch 16/20
```

```
60000/60000 [==============================] - 3s 57us/step - loss: 0.0199 - acc: 0.9938
- val_loss: 0.0976 - val_acc: 0.9736
Epoch 17/20
60000/60000 [==============================] - 3s 57us/step - loss: 0.0187 - acc: 0.9937
- val_loss: 0.1039 - val_acc: 0.9731
Epoch 18/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.0201 - acc: 0.9930
- val_loss: 0.1009 - val_acc: 0.9730
Epoch 19/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.0163 - acc: 0.9951
- val_loss: 0.0933 - val_acc: 0.9766
Epoch 20/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.0156 - acc: 0.9950
- val_loss: 0.1062 - val_acc: 0.9737
```

In [40]:

```
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, verb
ose=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 epoc
hs

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

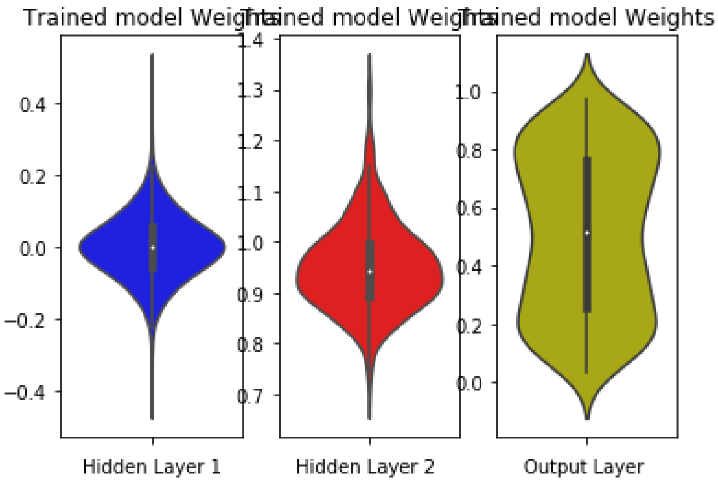
```
Test score: 0.10622683503271255
Test accuracy: 0.9737
```



In [41]:

```
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 [42]:

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

from keras.layers import Dropout

model_drop = Sequential()

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

model_drop.add(Dense(128, activation='sigmoid', kernel_initializer=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()
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_b
ackend.py:3733: calling dropout (from tensorflow.python.ops.nn_ops) with keep_prob is dep
recated and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_17 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_3 (Batch (None, 512)               2048
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_18 (Dense)             (None, 128)               65664
_________________________________________________________________
batch_normalization_4 (Batch (None, 128)               512
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_19 (Dense)             (None, 10)                1290
=================================================================
Total params: 471,434
Trainable params: 470,154
Non-trainable params: 1,280
_________________________________________________________________
```

In [43]:

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

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

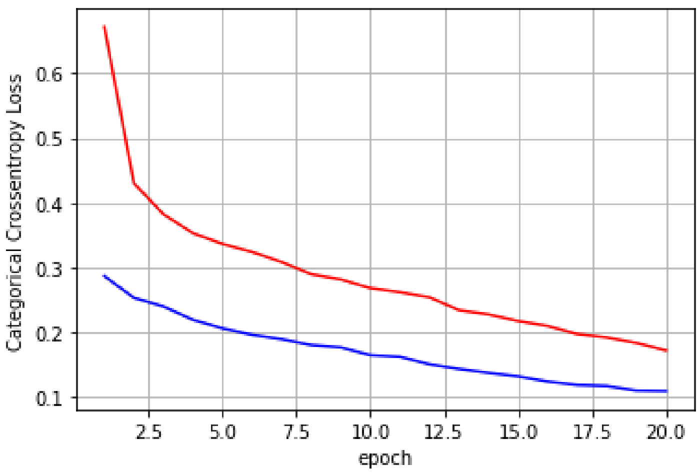
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.6715 - acc: 0.7927
- val_loss: 0.2873 - val_acc: 0.9125
Epoch 2/20
60000/60000 [==============================] - 4s 62us/step - loss: 0.4303 - acc: 0.8689
- val_loss: 0.2536 - val_acc: 0.9257
Epoch 3/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.3826 - acc: 0.8837
- val_loss: 0.2402 - val_acc: 0.9292
Epoch 4/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.3532 - acc: 0.8934
- val_loss: 0.2198 - val_acc: 0.9346
Epoch 5/20
60000/60000 [==============================] - 4s 59us/step - loss: 0.3368 - acc: 0.8979
- val_loss: 0.2066 - val_acc: 0.9399
Epoch 6/20
60000/60000 [==============================] - 4s 61us/step - loss: 0.3244 - acc: 0.9024
- val_loss: 0.1965 - val_acc: 0.9419
Epoch 7/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.3088 - acc: 0.9073
- val_loss: 0.1898 - val_acc: 0.9443
Epoch 8/20
60000/60000 [==============================] - 3s 57us/step - loss: 0.2899 - acc: 0.9126
- val_loss: 0.1807 - val_acc: 0.9448
Epoch 9/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.2821 - acc: 0.9149
- val_loss: 0.1770 - val_acc: 0.9481
Epoch 10/20
60000/60000 [==============================] - 4s 61us/step - loss: 0.2687 - acc: 0.9208
- val_loss: 0.1653 - val_acc: 0.9505
Epoch 11/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.2623 - acc: 0.9203
- val_loss: 0.1628 - val_acc: 0.9495
Epoch 12/20
60000/60000 [==============================] - 3s 58us/step - loss: 0.2544 - acc: 0.9241
- val_loss: 0.1512 - val_acc: 0.9556
Epoch 13/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.2346 - acc: 0.9299
- val_loss: 0.1439 - val_acc: 0.9576
Epoch 14/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.2281 - acc: 0.9305
- val_loss: 0.1379 - val_acc: 0.9587
Epoch 15/20
```

```
Epoch 15/20
60000/60000 [==============================] - 4s 62us/step - loss: 0.2179 - acc: 0.9335
- val_loss: 0.1325 - val_acc: 0.9601
Epoch 16/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.2102 - acc: 0.9363
- val_loss: 0.1243 - val_acc: 0.9633
Epoch 17/20
60000/60000 [==============================] - 4s 61us/step - loss: 0.1977 - acc: 0.9408
- val_loss: 0.1192 - val_acc: 0.9651
Epoch 18/20
60000/60000 [==============================] - 4s 59us/step - loss: 0.1924 - acc: 0.9423
- val_loss: 0.1176 - val_acc: 0.9645
Epoch 19/20
60000/60000 [==============================] - 4s 62us/step - loss: 0.1840 - acc: 0.9444
- val_loss: 0.1107 - val_acc: 0.9661
Epoch 20/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.1724 - acc: 0.9487
- val_loss: 0.1100 - val_acc: 0.9664
```

In [44]:

```
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, verb
ose=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 epoc
hs

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

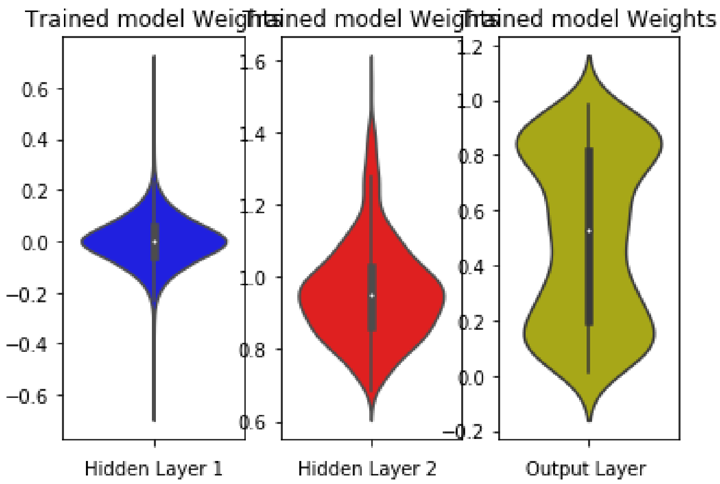
```
Test score: 0.10995881164651364
Test accuracy: 0.9664
```



In [45]:

```
w_after = model_drop.get_weights()
```

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


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

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

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



Hyper-parameter tuning of Keras models using Sklearn

In [0]:

```
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=R
andomNormal(mean=0.0, stddev=0.062, seed=None)))
    model.add(Dense(128, activation=activ, kernel_initializer=RandomNormal(mean=0.0, stdd
ev=0.125, seed=None)) )
    model.add(Dense(output_dim, activation='softmax'))


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

    return model
```

In [0]:

```
# https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-pyt
hon-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, verbose=0)
param_grid = dict(activ=activ)

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

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

In [48]:

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

```
Best: 0.978583 using {'activ': 'relu'}
0.977783 (0.001954) with: {'activ': 'sigmoid'}
0.978583 (0.001541) with: {'activ': 'relu'}
```

In [0]:

In [0]:

In [0]:

Two Hidden layer Architecture

In [0]:

Using RELU Activation and Adam Optimizer

In [49]:

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

from keras.layers import Dropout

model1 = Sequential()

model1.add(Dense(352, activation='relu', input_shape=(input_dim,), kernel_initializer=Ran
domNormal(mean=0.0, stddev=0.039, seed=None)))


model1.add(Dense(52, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev
=0.55, seed=None)) )


model1.add(Dense(output_dim, activation='softmax'))


model1.summary()
```

```
Model: "sequential_19"
_________________________________________________________________
```

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_53 (Dense)             (None, 352)               276320
_________________________________________________________________
dense_54 (Dense)             (None, 52)                18356
_________________________________________________________________
dense_55 (Dense)             (None, 10)                530
=================================================================
Total params: 295,206
Trainable params: 295,206
Non-trainable params: 0
_________________________________________________________________
```

In [50]:

```
model1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model1.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1
, validation_data=(X_test, Y_test))
```

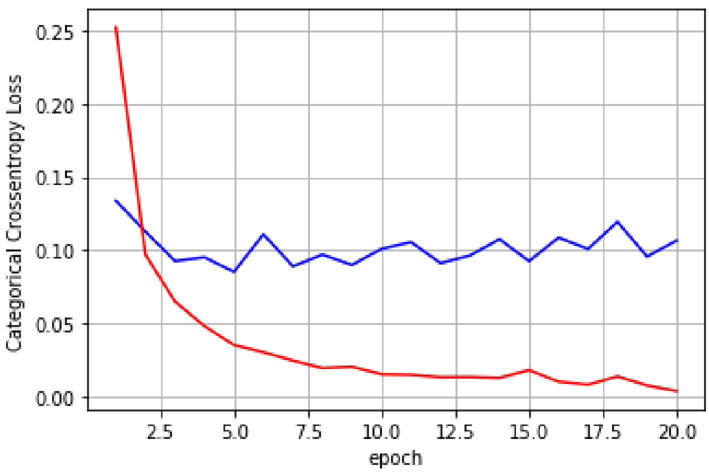
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 3s 58us/step - loss: 0.2520 - acc: 0.9256
- val_loss: 0.1338 - val_acc: 0.9580
Epoch 2/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.0972 - acc: 0.9699
- val_loss: 0.1128 - val_acc: 0.9665
Epoch 3/20
60000/60000 [==============================] - 2s 42us/step - loss: 0.0652 - acc: 0.9801
- val_loss: 0.0927 - val_acc: 0.9717
Epoch 4/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0484 - acc: 0.9845
- val_loss: 0.0953 - val_acc: 0.9694
Epoch 5/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.0355 - acc: 0.9885
- val_loss: 0.0854 - val_acc: 0.9766
Epoch 6/20
60000/60000 [==============================] - 2s 38us/step - loss: 0.0306 - acc: 0.9901
- val_loss: 0.1109 - val_acc: 0.9700
Epoch 7/20
60000/60000 [==============================] - 2s 37us/step - loss: 0.0249 - acc: 0.9919
- val_loss: 0.0891 - val_acc: 0.9743
Epoch 8/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.0199 - acc: 0.9933
- val_loss: 0.0971 - val_acc: 0.9757
Epoch 9/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.0207 - acc: 0.9930
- val_loss: 0.0901 - val_acc: 0.9752
Epoch 10/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.0156 - acc: 0.9950
- val_loss: 0.1010 - val_acc: 0.9753
Epoch 11/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.0153 - acc: 0.9948
- val_loss: 0.1056 - val_acc: 0.9764
Epoch 12/20
60000/60000 [==============================] - 2s 40us/step - loss: 0.0137 - acc: 0.9953
- val_loss: 0.0913 - val_acc: 0.9780
Epoch 13/20
60000/60000 [==============================] - 2s 42us/step - loss: 0.0137 - acc: 0.9954
- val_loss: 0.0965 - val_acc: 0.9779
Epoch 14/20
60000/60000 [==============================] - 2s 37us/step - loss: 0.0132 - acc: 0.9956
- val_loss: 0.1077 - val_acc: 0.9769
Epoch 15/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0184 - acc: 0.9938
- val_loss: 0.0926 - val_acc: 0.9795
Epoch 16/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.0106 - acc: 0.9966
- val_loss: 0.1087 - val_acc: 0.9790
Epoch 17/20
60000/60000 [==============================] - 2s 38us/step - loss: 0.0085 - acc: 0.9974
```

```
- val_loss: 0.1010 - val_acc: 0.9792
Epoch 18/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0141 - acc: 0.9957
- val_loss: 0.1195 - val_acc: 0.9775
Epoch 19/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0080 - acc: 0.9974
- val_loss: 0.0957 - val_acc: 0.9801
Epoch 20/20
60000/60000 [==============================] - 2s 38us/step - loss: 0.0041 - acc: 0.9987
- val_loss: 0.1067 - val_acc: 0.9769
```

In [51]:

```
score = model1.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, verb
ose=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 epoc
hs

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

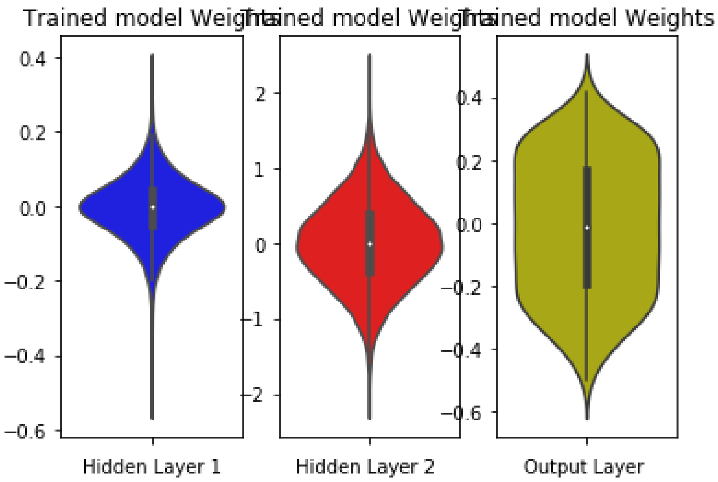
```
Test score: 0.10668754959471503
Test accuracy: 0.9769
```



In [52]:

```
w_after = model1.get_weights()

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


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

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

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

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



In [0]:

```

```

In [0]:

```

```

In [0]:

```

```

With Dropout

In [53]:

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

from keras.layers import Dropout

model_drop = Sequential()

model_drop.add(Dense(352, activation='relu', input_shape=(input_dim,), kernel_initializer
=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_drop.add(Dropout(0.5))

model_drop.add(Dense(52, activation='relu', kernel_initializer=RandomNormal(mean=0.0, std
dev=0.55, seed=None)) )
model_drop.add(Dropout(0.7))

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


model_drop.summary()
```

WARNING:tensorflow:Large dropout rate: 0.7 (>0.5). In TensorFlow 2.x, dropout() uses drop

```
out rate instead of keep_prob. Please ensure that this is intended.
Model: "sequential_20"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_56 (Dense)             (None, 352)               276320    
_________________________________________________________________
dropout_3 (Dropout)          (None, 352)               0         
_________________________________________________________________
dense_57 (Dense)             (None, 52)                18356     
_________________________________________________________________
dropout_4 (Dropout)          (None, 52)                0         
_________________________________________________________________
dense_58 (Dense)             (None, 10)                530       
=================================================================
Total params: 295,206
Trainable params: 295,206
Non-trainable params: 0
_________________________________________________________________
```

In [54]:

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

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

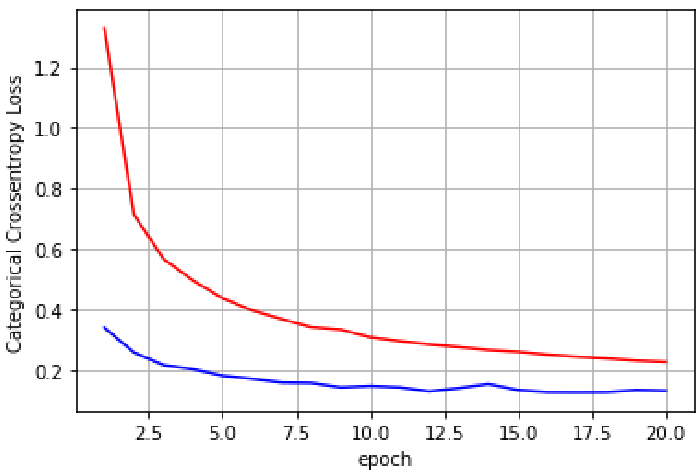
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 4s 60us/step - loss: 1.3294 - acc: 0.5665 - val_loss: 0.3412 - val_acc: 0.9163
Epoch 2/20
60000/60000 [==============================] - 2s 37us/step - loss: 0.7147 - acc: 0.7711 - val_loss: 0.2594 - val_acc: 0.9316
Epoch 3/20
60000/60000 [==============================] - 2s 40us/step - loss: 0.5678 - acc: 0.8272 - val_loss: 0.2178 - val_acc: 0.9427
Epoch 4/20
60000/60000 [==============================] - 2s 40us/step - loss: 0.4968 - acc: 0.8510 - val_loss: 0.2039 - val_acc: 0.9473
Epoch 5/20
60000/60000 [==============================] - 2s 40us/step - loss: 0.4382 - acc: 0.8706 - val_loss: 0.1824 - val_acc: 0.9531
Epoch 6/20
60000/60000 [==============================] - 2s 42us/step - loss: 0.3978 - acc: 0.8834 - val_loss: 0.1722 - val_acc: 0.9550
Epoch 7/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.3694 - acc: 0.8918 - val_loss: 0.1599 - val_acc: 0.9575
Epoch 8/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.3429 - acc: 0.8998 - val_loss: 0.1586 - val_acc: 0.9588
Epoch 9/20
60000/60000 [==============================] - 2s 40us/step - loss: 0.3345 - acc: 0.9029 - val_loss: 0.1439 - val_acc: 0.9614
Epoch 10/20
60000/60000 [==============================] - 2s 38us/step - loss: 0.3093 - acc: 0.9097 - val_loss: 0.1484 - val_acc: 0.9633
Epoch 11/20
60000/60000 [==============================] - 2s 38us/step - loss: 0.2965 - acc: 0.9162 - val_loss: 0.1440 - val_acc: 0.9642
Epoch 12/20
60000/60000 [==============================] - 2s 37us/step - loss: 0.2857 - acc: 0.9177 - val_loss: 0.1309 - val_acc: 0.9665
Epoch 13/20
60000/60000 [==============================] - 2s 40us/step - loss: 0.2770 - acc: 0.9205 - val_loss: 0.1416 - val_acc: 0.9662
Epoch 14/20
60000/60000 [==============================] - 2s 40us/step - loss: 0.2674 - acc: 0.9227 - val_loss: 0.1549 - val_acc: 0.9646
```

```
Epoch 15/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.2615 - acc: 0.9252
- val_loss: 0.1351 - val_acc: 0.9676
Epoch 16/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.2521 - acc: 0.9278
- val_loss: 0.1285 - val_acc: 0.9686
Epoch 17/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.2452 - acc: 0.9278
- val_loss: 0.1281 - val_acc: 0.9691
Epoch 18/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.2397 - acc: 0.9307
- val_loss: 0.1285 - val_acc: 0.9693
Epoch 19/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.2326 - acc: 0.9328
- val_loss: 0.1348 - val_acc: 0.9710
Epoch 20/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.2283 - acc: 0.9345
- val_loss: 0.1325 - val_acc: 0.9708
```

In [55]:

```
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, verb
ose=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 epoc
hs

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

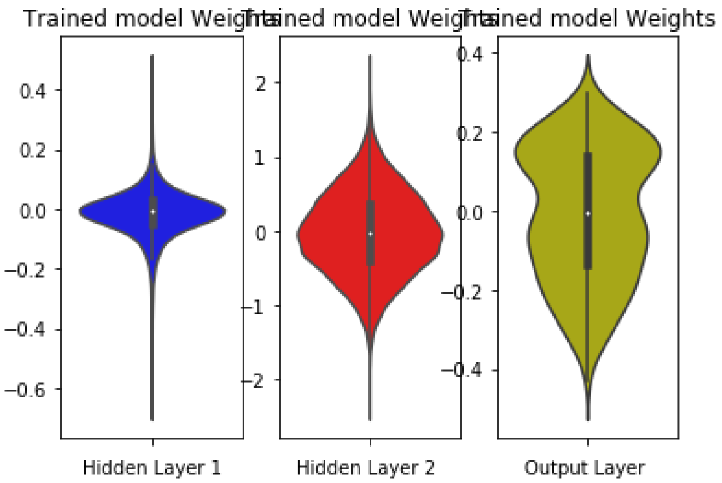
```
Test score: 0.13253417925792746
Test accuracy: 0.9708
```



In [56]:

```
w_after = model_drop.get_weights()
```

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


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

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

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



In [0]:

```

```

With Batch Normalization

In [0]:

```

```

In [57]:

```
# 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(352, activation='relu', input_shape=(input_dim,), kernel_initialize
r=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_batch.add(BatchNormalization())
```

```
model_batch.add(Dense(52, activation='relu', kernel_initializer=RandomNormal(mean=0.0, s
tddev=0.55, seed=None)) )
model_batch.add(BatchNormalization())

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


model_batch.summary()
```

```
Model: "sequential_21"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_59 (Dense)             (None, 352)               276320    
_________________________________________________________________
batch_normalization_5 (Batch (None, 352)               1408      
_________________________________________________________________
dense_60 (Dense)             (None, 52)                18356     
_________________________________________________________________
batch_normalization_6 (Batch (None, 52)                208       
_________________________________________________________________
dense_61 (Dense)             (None, 10)                530       
=================================================================
Total params: 296,822
Trainable params: 296,014
Non-trainable params: 808
_________________________________________________________________
```

In [58]:

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

history = model_batch.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verb
ose=1, validation_data=(X_test, Y_test))
```

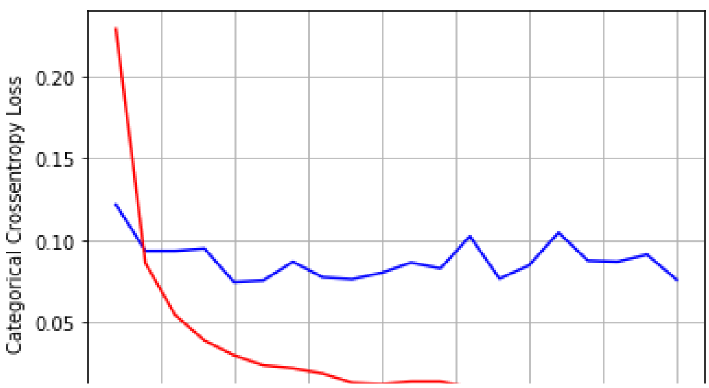
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 87us/step - loss: 0.2292 - acc: 0.9376
- val_loss: 0.1217 - val_acc: 0.9638
Epoch 2/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.0858 - acc: 0.9756
- val_loss: 0.0932 - val_acc: 0.9694
Epoch 3/20
60000/60000 [==============================] - 3s 58us/step - loss: 0.0542 - acc: 0.9838
- val_loss: 0.0933 - val_acc: 0.9707
Epoch 4/20
60000/60000 [==============================] - 3s 58us/step - loss: 0.0384 - acc: 0.9882
- val_loss: 0.0948 - val_acc: 0.9696
Epoch 5/20
60000/60000 [==============================] - 4s 59us/step - loss: 0.0294 - acc: 0.9907
- val_loss: 0.0742 - val_acc: 0.9777
Epoch 6/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.0233 - acc: 0.9927
- val_loss: 0.0752 - val_acc: 0.9781
Epoch 7/20
60000/60000 [==============================] - 3s 57us/step - loss: 0.0214 - acc: 0.9930
- val_loss: 0.0866 - val_acc: 0.9742
Epoch 8/20
60000/60000 [==============================] - 4s 62us/step - loss: 0.0183 - acc: 0.9942
- val_loss: 0.0771 - val_acc: 0.9781
Epoch 9/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.0125 - acc: 0.9964
- val_loss: 0.0759 - val_acc: 0.9776
Epoch 10/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0116 - acc: 0.9966
- val_loss: 0.0798 - val_acc: 0.9784
Epoch 11/20
60000/60000 [==============================] - 3s 57us/step - loss: 0.0132 - acc: 0.9959
- val_loss: 0.0861 - val_acc: 0.9777
Epoch 12/20
60000/60000 [==============================] - 3s 57us/step - loss: 0.0132 - acc: 0.9957
```

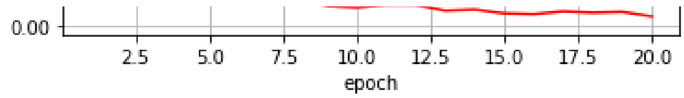
```
- val_loss: 0.0826 - val_acc: 0.9777
Epoch 13/20
60000/60000 [==============================] - 4s 61us/step - loss: 0.0096 - acc: 0.9971
- val_loss: 0.1023 - val_acc: 0.9731
Epoch 14/20
60000/60000 [==============================] - 4s 61us/step - loss: 0.0103 - acc: 0.9966
- val_loss: 0.0762 - val_acc: 0.9803
Epoch 15/20
60000/60000 [==============================] - 3s 56us/step - loss: 0.0079 - acc: 0.9976
- val_loss: 0.0845 - val_acc: 0.9804
Epoch 16/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.0074 - acc: 0.9975
- val_loss: 0.1045 - val_acc: 0.9756
Epoch 17/20
60000/60000 [==============================] - 4s 62us/step - loss: 0.0092 - acc: 0.9970
- val_loss: 0.0873 - val_acc: 0.9767
Epoch 18/20
60000/60000 [==============================] - 3s 58us/step - loss: 0.0084 - acc: 0.9972
- val_loss: 0.0867 - val_acc: 0.9782
Epoch 19/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.0089 - acc: 0.9970
- val_loss: 0.0911 - val_acc: 0.9778
Epoch 20/20
60000/60000 [==============================] - 4s 61us/step - loss: 0.0060 - acc: 0.9981
- val_loss: 0.0757 - val_acc: 0.9806
```

In [59]:

```
# 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, verb
ose=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 epoc
hs

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))

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

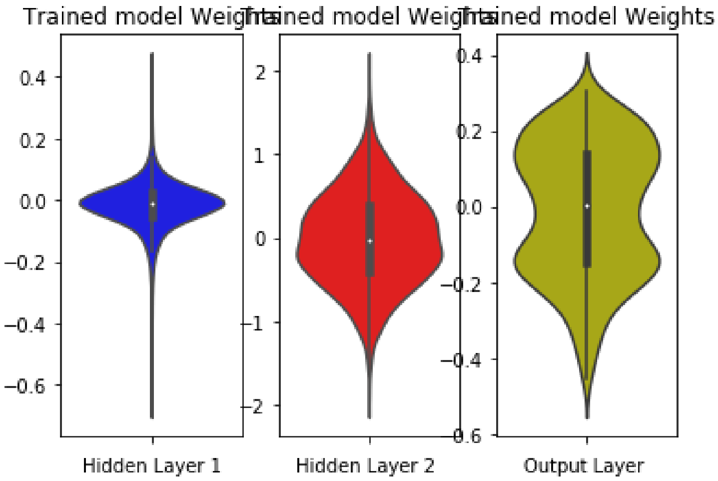
```
Test score: 0.07565803507223973
Test accuracy: 0.9806
```





In [84]:

```
w_after = model_drop.get_weights()

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


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

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

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



In [0]:

```
```

In [0]:

```
```

Batch Normalization + Dropout

In [61]:

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

from keras.layers import Dropout

model3 = Sequential()

model3.add(Dense(352, activation='relu', input_shape=(input_dim,), kernel_initializer=Ran
```

```
domNormal(mean=0.0, stddev=0.039, seed=None)))
model3.add(BatchNormalization())
model3.add(Dropout(0.5))

model3.add(Dense(52, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev
=0.55, seed=None)) )
model3.add(BatchNormalization())
model3.add(Dropout(0.5))



model3.add(Dense(output_dim, activation='softmax'))


model3.summary()
```

```
Model: "sequential_22"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_62 (Dense)             (None, 352)               276320
_________________________________________________________________
batch_normalization_7 (Batch (None, 352)               1408
_________________________________________________________________
dropout_5 (Dropout)          (None, 352)               0
_________________________________________________________________
dense_63 (Dense)             (None, 52)                18356
_________________________________________________________________
batch_normalization_8 (Batch (None, 52)                208
_________________________________________________________________
dropout_6 (Dropout)          (None, 52)                0
_________________________________________________________________
dense_64 (Dense)             (None, 10)                530
=================================================================
Total params: 296,822
Trainable params: 296,014
Non-trainable params: 808
_________________________________________________________________
```

In [62]:

```
model3.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model3.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1
, validation_data=(X_test, Y_test))
```

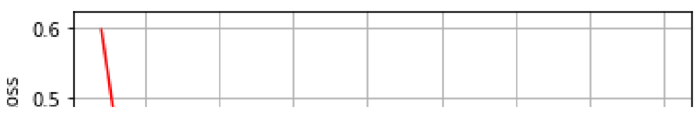
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.5979 - acc: 0.8192
- val_loss: 0.1928 - val_acc: 0.9417
Epoch 2/20
60000/60000 [==============================] - 4s 61us/step - loss: 0.3125 - acc: 0.9085
- val_loss: 0.1482 - val_acc: 0.9546
Epoch 3/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.2578 - acc: 0.9251
- val_loss: 0.1200 - val_acc: 0.9635
Epoch 4/20
60000/60000 [==============================] - 4s 59us/step - loss: 0.2192 - acc: 0.9368
- val_loss: 0.1104 - val_acc: 0.9688
Epoch 5/20
60000/60000 [==============================] - 3s 57us/step - loss: 0.1983 - acc: 0.9427
- val_loss: 0.0974 - val_acc: 0.9715
Epoch 6/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.1774 - acc: 0.9498
- val_loss: 0.0945 - val_acc: 0.9706
Epoch 7/20
60000/60000 [==============================] - 3s 56us/step - loss: 0.1635 - acc: 0.9526
- val_loss: 0.0923 - val_acc: 0.9713
Epoch 8/20
60000/60000 [==============================] - 4s 59us/step - loss: 0.1518 - acc: 0.9556
- val_loss: 0.0861 - val_acc: 0.9737
Epoch 9/20
```

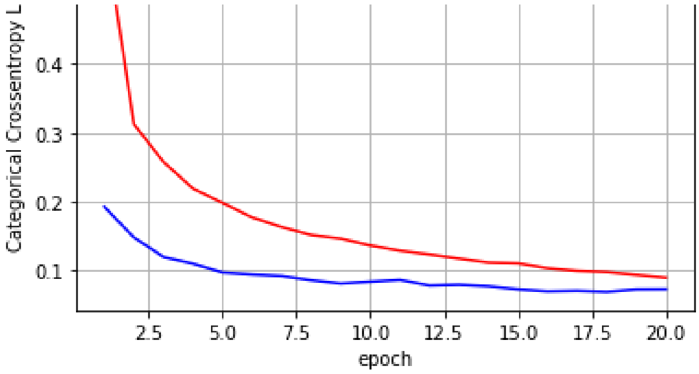
```
60000/60000 [==============================] - 4s 65us/step - loss: 0.1465 - acc: 0.9576
- val_loss: 0.0813 - val_acc: 0.9757
Epoch 10/20
60000/60000 [==============================] - 4s 61us/step - loss: 0.1367 - acc: 0.9603
- val_loss: 0.0840 - val_acc: 0.9755
Epoch 11/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.1290 - acc: 0.9627
- val_loss: 0.0866 - val_acc: 0.9748
Epoch 12/20
60000/60000 [==============================] - 4s 61us/step - loss: 0.1237 - acc: 0.9642
- val_loss: 0.0786 - val_acc: 0.9757
Epoch 13/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.1177 - acc: 0.9649
- val_loss: 0.0797 - val_acc: 0.9781
Epoch 14/20
60000/60000 [==============================] - 4s 61us/step - loss: 0.1120 - acc: 0.9663
- val_loss: 0.0774 - val_acc: 0.9775
Epoch 15/20
60000/60000 [==============================] - 4s 58us/step - loss: 0.1108 - acc: 0.9671
- val_loss: 0.0731 - val_acc: 0.9782
Epoch 16/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.1037 - acc: 0.9692
- val_loss: 0.0702 - val_acc: 0.9794
Epoch 17/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.0999 - acc: 0.9703
- val_loss: 0.0710 - val_acc: 0.9794
Epoch 18/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.0981 - acc: 0.9704
- val_loss: 0.0694 - val_acc: 0.9789
Epoch 19/20
60000/60000 [==============================] - 3s 58us/step - loss: 0.0940 - acc: 0.9723
- val_loss: 0.0730 - val_acc: 0.9797
Epoch 20/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.0900 - acc: 0.9736
- val_loss: 0.0732 - val_acc: 0.9790
```

In [63]:

```
score = model3.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, verb
ose=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 epoc
hs

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

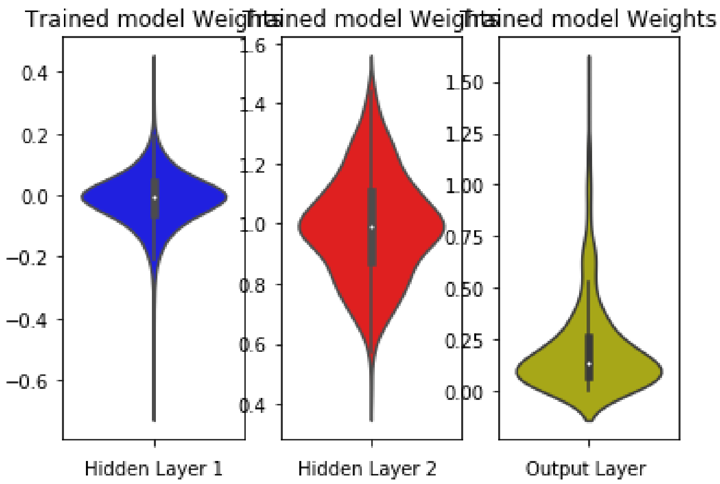
```
Test score: 0.07315509025455104
Test accuracy: 0.979
```





In [64]:

```
w_after = model3.get_weights()

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


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

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

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



In [0]:

```
```

In [0]:

```
```

Three Hidden Layer Architecture

Using RELU Activation and Adam Optimizer

In [0]:

```

```

In [65]:

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

from keras.layers import Dropout

model1 = Sequential()

model1.add(Dense(352, activation='relu', input_shape=(input_dim,), kernel_initializer=Ran
domNormal(mean=0.0, stddev=0.039, seed=None)))


model1.add(Dense(52, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev
=0.55, seed=None)) )

model1.add(Dense(102, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev
=0.6, seed=None)) )

model1.add(Dense(output_dim, activation='softmax'))


model1.summary()
```

```
Model: "sequential_23"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_65 (Dense)             (None, 352)               276320    
_________________________________________________________________
dense_66 (Dense)             (None, 52)                18356     
_________________________________________________________________
dense_67 (Dense)             (None, 102)               5406      
_________________________________________________________________
dense_68 (Dense)             (None, 10)                1030      
=================================================================
Total params: 301,112
Trainable params: 301,112
Non-trainable params: 0
_________________________________________________________________
```

In [66]:

```
model1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model1.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 [==============================] - 4s 63us/step - loss: 0.3633 - acc: 0.9046
- val_loss: 0.1766 - val_acc: 0.9475
Epoch 2/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.1396 - acc: 0.9590
- val_loss: 0.1287 - val_acc: 0.9616
Epoch 3/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0965 - acc: 0.9714
- val_loss: 0.1168 - val_acc: 0.9675
Epoch 4/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.0773 - acc: 0.9760
- val_loss: 0.1324 - val_acc: 0.9637
Epoch 5/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0646 - acc: 0.9796
- val_loss: 0.1348 - val_acc: 0.9646
Epoch 6/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0517 - acc: 0.9835
```

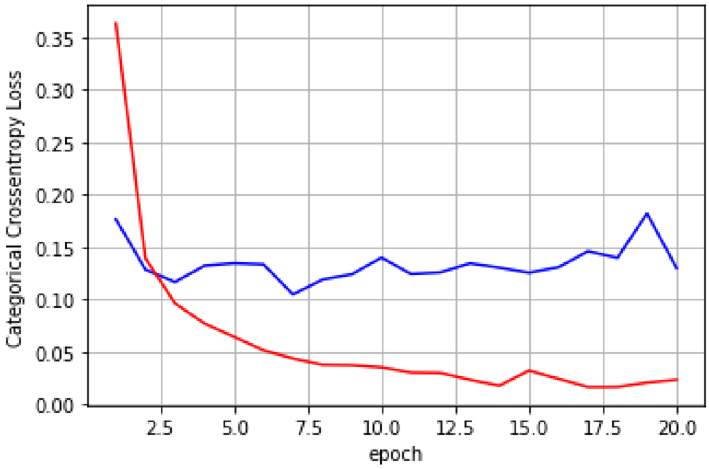
```
- val_loss: 0.1337 - val_acc: 0.9676
Epoch 7/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.0440 - acc: 0.9863
- val_loss: 0.1049 - val_acc: 0.9724
Epoch 8/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.0380 - acc: 0.9880
- val_loss: 0.1191 - val_acc: 0.9725
Epoch 9/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.0377 - acc: 0.9880
- val_loss: 0.1243 - val_acc: 0.9726
Epoch 10/20
60000/60000 [==============================] - 2s 40us/step - loss: 0.0356 - acc: 0.9892
- val_loss: 0.1401 - val_acc: 0.9703
Epoch 11/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0305 - acc: 0.9908
- val_loss: 0.1245 - val_acc: 0.9731
Epoch 12/20
60000/60000 [==============================] - 2s 38us/step - loss: 0.0302 - acc: 0.9907
- val_loss: 0.1260 - val_acc: 0.9717
Epoch 13/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0236 - acc: 0.9926
- val_loss: 0.1347 - val_acc: 0.9727
Epoch 14/20
60000/60000 [==============================] - 2s 40us/step - loss: 0.0180 - acc: 0.9943
- val_loss: 0.1305 - val_acc: 0.9731
Epoch 15/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.0325 - acc: 0.9904
- val_loss: 0.1257 - val_acc: 0.9747
Epoch 16/20
60000/60000 [==============================] - 2s 40us/step - loss: 0.0245 - acc: 0.9924
- val_loss: 0.1310 - val_acc: 0.9757
Epoch 17/20
60000/60000 [==============================] - 2s 38us/step - loss: 0.0167 - acc: 0.9946
- val_loss: 0.1460 - val_acc: 0.9708
Epoch 18/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.0168 - acc: 0.9949
- val_loss: 0.1398 - val_acc: 0.9743
Epoch 19/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.0209 - acc: 0.9938
- val_loss: 0.1822 - val_acc: 0.9692
Epoch 20/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.0236 - acc: 0.9928
- val_loss: 0.1301 - val_acc: 0.9767
```

In [67]:

```python
score = model1.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, verb
ose=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 epoc
hs

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

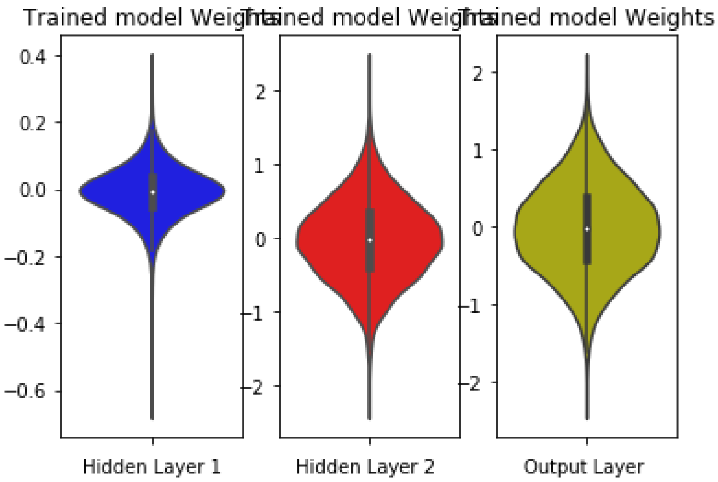
```
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.13014392414744216
Test accuracy: 0.9767
```



In [68]:

```
w_after = model1.get_weights()

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


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

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

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



In [0]:

```

```

In [0]:

```

```

In [0]:

```

```

With Dropout

In [69]:

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

from keras.layers import Dropout

model_drop = Sequential()

model_drop.add(Dense(352, activation='relu', input_shape=(input_dim,), kernel_initializer
=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_drop.add(Dropout(0.5))

model_drop.add(Dense(52, activation='relu', kernel_initializer=RandomNormal(mean=0.0, std
dev=0.55, seed=None)) )
model_drop.add(Dropout(0.5))

model1.add(Dense(102, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev
=0.6, seed=None)) )
model_drop.add(Dropout(0.5))

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


model_drop.summary()
```

```
Model: "sequential_24"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_69 (Dense)             (None, 352)               276320    
_________________________________________________________________
dropout_7 (Dropout)          (None, 352)               0         
_________________________________________________________________
dense_70 (Dense)             (None, 52)                18356     
_________________________________________________________________
dropout_8 (Dropout)          (None, 52)                0         
_________________________________________________________________
dropout_9 (Dropout)          (None, 52)                0         
_________________________________________________________________
dense_72 (Dense)             (None, 10)                530       
=================================================================
Total params: 295,206
Trainable params: 295,206
Non-trainable params: 0
_________________________________________________________________
```

In [70]:

```
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, verbo
se=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 4s 74us/step - loss: 1.5708 - acc: 0.4904
- val_loss: 0.3985 - val_acc: 0.9120
Epoch 2/20
```

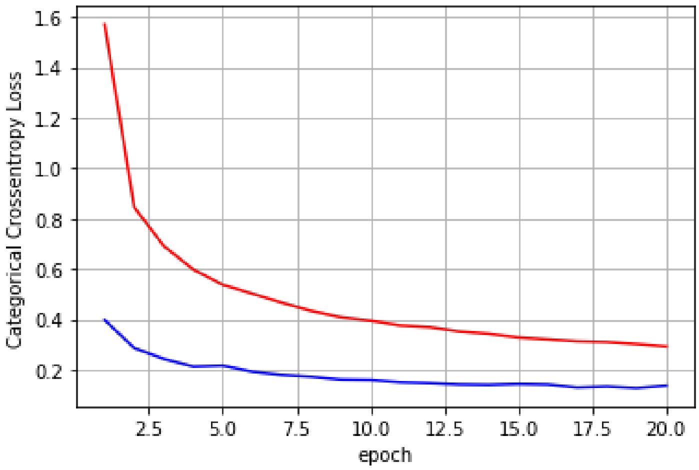
```
60000/60000 [==============================] - 3s 42us/step - loss: 0.8458 - acc: 0.7161
- val_loss: 0.2862 - val_acc: 0.9282
Epoch 3/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.6915 - acc: 0.7740
- val_loss: 0.2436 - val_acc: 0.9348
Epoch 4/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.5985 - acc: 0.8076
- val_loss: 0.2135 - val_acc: 0.9435
Epoch 5/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.5376 - acc: 0.8304
- val_loss: 0.2173 - val_acc: 0.9464
Epoch 6/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.5022 - acc: 0.8433
- val_loss: 0.1925 - val_acc: 0.9527
Epoch 7/20
60000/60000 [==============================] - 2s 40us/step - loss: 0.4663 - acc: 0.8554
- val_loss: 0.1797 - val_acc: 0.9539
Epoch 8/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.4336 - acc: 0.8655
- val_loss: 0.1726 - val_acc: 0.9543
Epoch 9/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.4093 - acc: 0.8730
- val_loss: 0.1612 - val_acc: 0.9604
Epoch 10/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.3952 - acc: 0.8772
- val_loss: 0.1597 - val_acc: 0.9603
Epoch 11/20
60000/60000 [==============================] - 2s 40us/step - loss: 0.3762 - acc: 0.8839
- val_loss: 0.1507 - val_acc: 0.9640
Epoch 12/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.3696 - acc: 0.8870
- val_loss: 0.1475 - val_acc: 0.9633
Epoch 13/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.3523 - acc: 0.8898
- val_loss: 0.1421 - val_acc: 0.9658
Epoch 14/20
60000/60000 [==============================] - 2s 40us/step - loss: 0.3429 - acc: 0.8965
- val_loss: 0.1406 - val_acc: 0.9657
Epoch 15/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.3285 - acc: 0.8975
- val_loss: 0.1440 - val_acc: 0.9649
Epoch 16/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.3209 - acc: 0.9019
- val_loss: 0.1418 - val_acc: 0.9668
Epoch 17/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.3132 - acc: 0.9041
- val_loss: 0.1299 - val_acc: 0.9703
Epoch 18/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.3103 - acc: 0.9060
- val_loss: 0.1339 - val_acc: 0.9683
Epoch 19/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.3021 - acc: 0.9073
- val_loss: 0.1277 - val_acc: 0.9683
Epoch 20/20
60000/60000 [==============================] - 2s 40us/step - loss: 0.2925 - acc: 0.9094
- val_loss: 0.1372 - val_acc: 0.9668
```

In [71]:

```python
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, verb
```

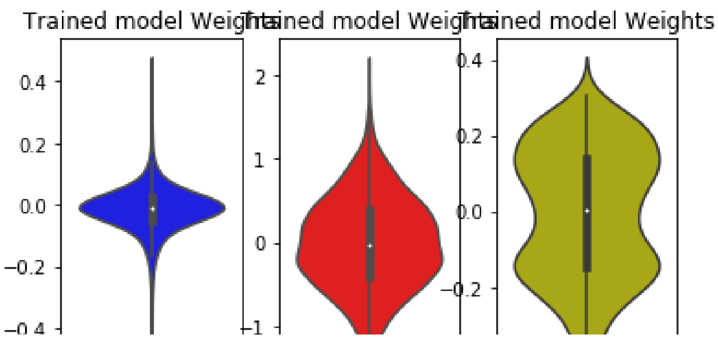
```
ose=1, validation_data=(X_test, Y_test))

# 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 epoc
hs

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

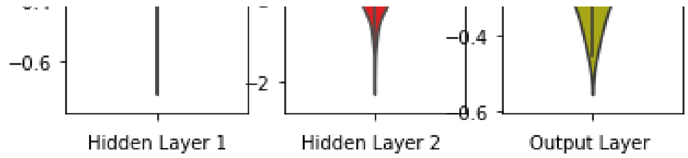
```
Test score: 0.1372257164807059
Test accuracy: 0.9668
```



In [72]:

```
w_after = model_drop.get_weights()

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


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

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

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





In [0]:

With Batch Normalization

In [0]:

In [73]:

```
# 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(352, activation='relu', input_shape=(input_dim,), kernel_initialize
r=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_batch.add(BatchNormalization())

model_batch.add(Dense(52, activation='relu', kernel_initializer=RandomNormal(mean=0.0, s
tddev=0.55, seed=None)) )
model_batch.add(BatchNormalization())

model_batch.add(Dense(102, activation='relu', kernel_initializer=RandomNormal(mean=0.0, s
tddev=0.55, seed=None)) )
model_batch.add(BatchNormalization())


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


model_batch.summary()
```

```
Model: "sequential_25"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_73 (Dense)             (None, 352)               276320
_________________________________________________________________
batch_normalization_9 (Batch (None, 352)               1408
_________________________________________________________________
dense_74 (Dense)             (None, 52)                18356
_________________________________________________________________
batch_normalization_10 (Batc (None, 52)                208
_________________________________________________________________
dense_75 (Dense)             (None, 102)               5406
_________________________________________________________________
batch_normalization_11 (Batc (None, 102)               408
_________________________________________________________________
dense_76 (Dense)             (None, 10)                1030
=================================================================
Total params: 303,136
Trainable params: 302,124
Non-trainable params: 1,012
```

In [74]:

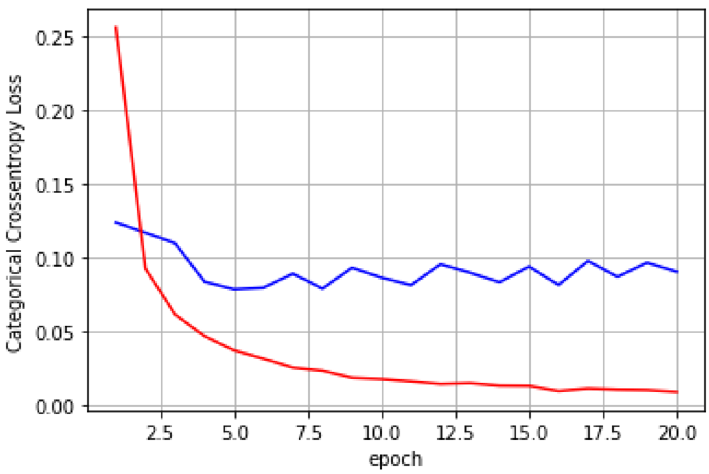
```
model_batch.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accurac
y'])

history = model_batch.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verb
ose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 109us/step - loss: 0.2560 - acc: 0.9258
- val_loss: 0.1237 - val_acc: 0.9627
Epoch 2/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.0924 - acc: 0.9721
- val_loss: 0.1167 - val_acc: 0.9627
Epoch 3/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0614 - acc: 0.9813
- val_loss: 0.1097 - val_acc: 0.9666
Epoch 4/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0465 - acc: 0.9855
- val_loss: 0.0835 - val_acc: 0.9754
Epoch 5/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0369 - acc: 0.9876
- val_loss: 0.0784 - val_acc: 0.9750
Epoch 6/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0313 - acc: 0.9897
- val_loss: 0.0795 - val_acc: 0.9753
Epoch 7/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.0253 - acc: 0.9918
- val_loss: 0.0890 - val_acc: 0.9735
Epoch 8/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0231 - acc: 0.9925
- val_loss: 0.0789 - val_acc: 0.9787
Epoch 9/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0183 - acc: 0.9942
- val_loss: 0.0930 - val_acc: 0.9735
Epoch 10/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0174 - acc: 0.9940
- val_loss: 0.0863 - val_acc: 0.9759
Epoch 11/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.0159 - acc: 0.9945
- val_loss: 0.0812 - val_acc: 0.9790
Epoch 12/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0141 - acc: 0.9953
- val_loss: 0.0953 - val_acc: 0.9759
Epoch 13/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0147 - acc: 0.9951
- val_loss: 0.0896 - val_acc: 0.9769
Epoch 14/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0130 - acc: 0.9957
- val_loss: 0.0832 - val_acc: 0.9799
Epoch 15/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.0128 - acc: 0.9961
- val_loss: 0.0938 - val_acc: 0.9764
Epoch 16/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.0093 - acc: 0.9973
- val_loss: 0.0813 - val_acc: 0.9786
Epoch 17/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0111 - acc: 0.9964
- val_loss: 0.0976 - val_acc: 0.9755
Epoch 18/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0103 - acc: 0.9964
- val_loss: 0.0868 - val_acc: 0.9791
Epoch 19/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0100 - acc: 0.9965
- val_loss: 0.0964 - val_acc: 0.9784
Epoch 20/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0088 - acc: 0.9971
- val_loss: 0.0903 - val_acc: 0.9787
```

In [75]:

```
# 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, verb
ose=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 epoc
hs

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))

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

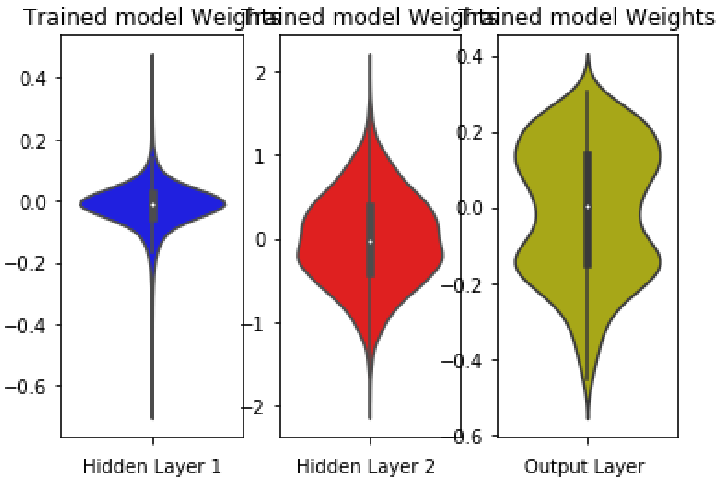
```
Test score: 0.09027723879703226
Test accuracy: 0.9787
```



In [76]:

```
w_after = model_drop.get_weights()

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


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

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

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



In [0]:

```

```

In [0]:

```

```

Batch Normalization + Dropout

In [77]:

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

from keras.layers import Dropout

model3 = Sequential()

model3.add(Dense(352, activation='relu', input_shape=(input_dim,), kernel_initializer=Ran
domNormal(mean=0.0, stddev=0.039, seed=None)))
model3.add(BatchNormalization())
model3.add(Dropout(0.5))

model3.add(Dense(52, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev
=0.55, seed=None)) )
model3.add(BatchNormalization())
model3.add(Dropout(0.5))

model3.add(Dense(102, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev
=0.55, seed=None)) )
model3.add(BatchNormalization())
model3.add(Dropout(0.5))


model3.add(Dense(output_dim, activation='softmax'))


model3.summary()
```

```
Model: "sequential_26"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_77 (Dense)             (None, 352)               276320    
_________________________________________________________________
```

```
batch_normalization_12 (Batc (None, 352)               1408
_________________________________________________________________
dropout_10 (Dropout)         (None, 352)               0
_________________________________________________________________
dense_78 (Dense)             (None, 52)                18356
_________________________________________________________________
batch_normalization_13 (Batc (None, 52)                208
_________________________________________________________________
dropout_11 (Dropout)         (None, 52)                0
_________________________________________________________________
dense_79 (Dense)             (None, 102)               5406
_________________________________________________________________
batch_normalization_14 (Batc (None, 102)               408
_________________________________________________________________
dropout_12 (Dropout)         (None, 102)               0
_________________________________________________________________
dense_80 (Dense)             (None, 10)                1030
=================================================================
Total params: 303,136
Trainable params: 302,124
Non-trainable params: 1,012
_________________________________________________________________
```

In [78]:

```
model3.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model3.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1
, validation_data=(X_test, Y_test))
```

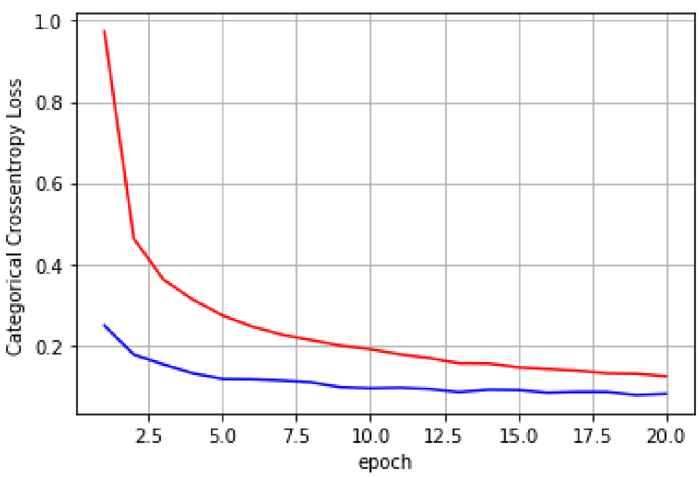
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.9735 - acc: 0.6879
- val_loss: 0.2488 - val_acc: 0.9267
Epoch 2/20
60000/60000 [==============================] - 4s 75us/step - loss: 0.4623 - acc: 0.8618
- val_loss: 0.1765 - val_acc: 0.9454
Epoch 3/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.3619 - acc: 0.8964
- val_loss: 0.1529 - val_acc: 0.9550
Epoch 4/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.3122 - acc: 0.9128
- val_loss: 0.1309 - val_acc: 0.9603
Epoch 5/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.2734 - acc: 0.9225
- val_loss: 0.1171 - val_acc: 0.9671
Epoch 6/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.2464 - acc: 0.9304
- val_loss: 0.1163 - val_acc: 0.9658
Epoch 7/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.2261 - acc: 0.9372
- val_loss: 0.1132 - val_acc: 0.9670
Epoch 8/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.2128 - acc: 0.9397
- val_loss: 0.1087 - val_acc: 0.9709
Epoch 9/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.1991 - acc: 0.9437
- val_loss: 0.0964 - val_acc: 0.9732
Epoch 10/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.1903 - acc: 0.9469
- val_loss: 0.0937 - val_acc: 0.9731
Epoch 11/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.1773 - acc: 0.9502
- val_loss: 0.0951 - val_acc: 0.9741
Epoch 12/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.1682 - acc: 0.9536
- val_loss: 0.0923 - val_acc: 0.9747
Epoch 13/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.1556 - acc: 0.9568
- val_loss: 0.0845 - val_acc: 0.9756
Epoch 14/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.1551 - acc: 0.9568
```

```
- val_loss: 0.0906 - val_acc: 0.9748
Epoch 15/20
60000/60000 [==============================] - 5s 76us/step - loss: 0.1458 - acc: 0.9595
- val_loss: 0.0898 - val_acc: 0.9771
Epoch 16/20
60000/60000 [==============================] - 4s 75us/step - loss: 0.1416 - acc: 0.9611
- val_loss: 0.0830 - val_acc: 0.9779
Epoch 17/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.1376 - acc: 0.9626
- val_loss: 0.0852 - val_acc: 0.9775
Epoch 18/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.1312 - acc: 0.9634
- val_loss: 0.0850 - val_acc: 0.9772
Epoch 19/20
60000/60000 [==============================] - 5s 79us/step - loss: 0.1302 - acc: 0.9644
- val_loss: 0.0773 - val_acc: 0.9788
Epoch 20/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.1235 - acc: 0.9659
- val_loss: 0.0802 - val_acc: 0.9792
```

In [79]:

```
score = model3.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, verb
ose=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 epoc
hs

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

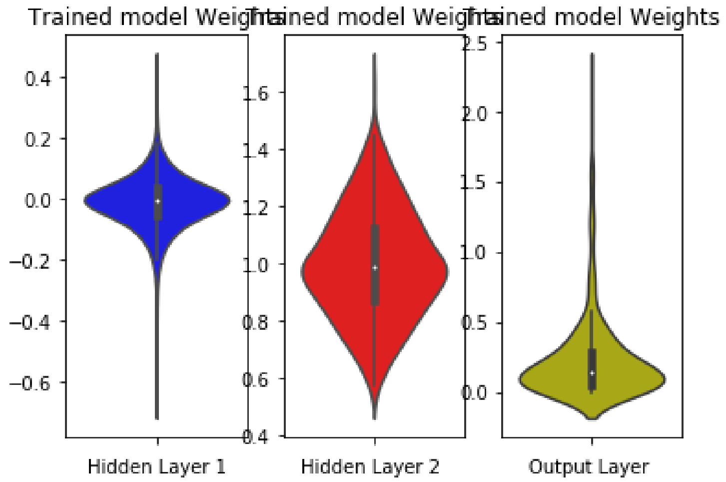
```
Test score: 0.08018281710293376
Test accuracy: 0.9792
```



In [80]:

```
w_after = model3.get_weights()
```

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


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

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

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



In [0]:

```

```

In [0]:

```

```

In [0]:

```

```

Five Layer Architecture

Using RELU Activation and Adam Optimizer

In [0]:

```

```

In [91]:

```
model_relu = Sequential()
model_relu.add(Dense(250, activation='relu', input_shape=(input_dim,), kernel_initializer
=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
```

```
model_relu.add(Dense(150, activation='relu', kernel_initializer=RandomNormal(mean=0.0, st
ddev=0.125, seed=None)) )
model_relu.add(Dense(146, activation='relu', kernel_initializer=RandomNormal(mean=0.0, st
ddev=0.15, seed=None)) )
model_relu.add(Dense(60, activation='relu', kernel_initializer=RandomNormal(mean=0.0, std
dev=0.25, seed=None)) )
model_relu.add(Dense(40, activation='relu', kernel_initializer=RandomNormal(mean=0.0, std
dev=0.5, seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())
```

```
Model: "sequential_29"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_93 (Dense)             (None, 250)               196250    
_________________________________________________________________
dense_94 (Dense)             (None, 150)               37650     
_________________________________________________________________
dense_95 (Dense)             (None, 146)               22046     
_________________________________________________________________
dense_96 (Dense)             (None, 60)                8820      
_________________________________________________________________
dense_97 (Dense)             (None, 40)                2440      
_________________________________________________________________
dense_98 (Dense)             (None, 10)                410       
=================================================================
Total params: 267,616
Trainable params: 267,616
Non-trainable params: 0
_________________________________________________________________
None
```

In [0]:

```

```

In [92]:

```
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, verbo
se=1, validation_data=(X_test, Y_test))
```

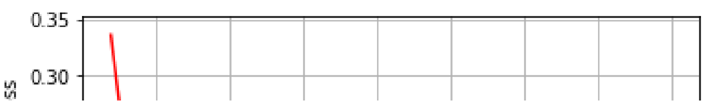
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.3360 - acc: 0.9028
- val_loss: 0.1423 - val_acc: 0.9570
Epoch 2/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.1223 - acc: 0.9639
- val_loss: 0.1142 - val_acc: 0.9638
Epoch 3/20
60000/60000 [==============================] - 3s 45us/step - loss: 0.0848 - acc: 0.9738
- val_loss: 0.0979 - val_acc: 0.9721
Epoch 4/20
60000/60000 [==============================] - 3s 45us/step - loss: 0.0639 - acc: 0.9802
- val_loss: 0.1029 - val_acc: 0.9688
Epoch 5/20
60000/60000 [==============================] - 3s 46us/step - loss: 0.0529 - acc: 0.9832
- val_loss: 0.0985 - val_acc: 0.9701
Epoch 6/20
60000/60000 [==============================] - 3s 46us/step - loss: 0.0453 - acc: 0.9850
- val_loss: 0.0847 - val_acc: 0.9762
Epoch 7/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.0388 - acc: 0.9875
- val_loss: 0.0940 - val_acc: 0.9746
Epoch 8/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.0314 - acc: 0.9896
- val_loss: 0.1085 - val_acc: 0.9718
Epoch 9/20
```

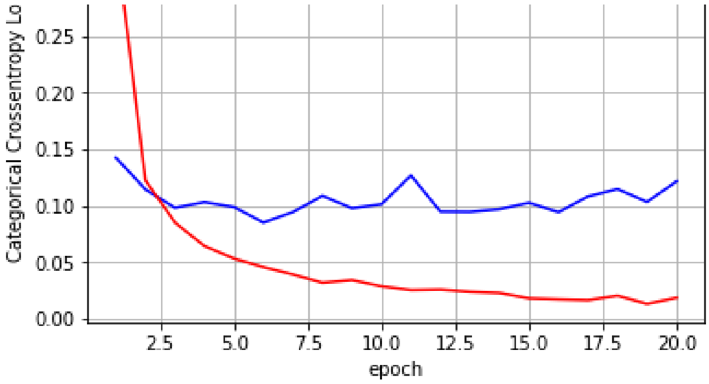
```
60000/60000 [==============================] - 3s 52us/step - loss: 0.0338 - acc: 0.9891
- val_loss: 0.0975 - val_acc: 0.9740
Epoch 10/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.0282 - acc: 0.9908
- val_loss: 0.1011 - val_acc: 0.9754
Epoch 11/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.0249 - acc: 0.9920
- val_loss: 0.1265 - val_acc: 0.9700
Epoch 12/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.0253 - acc: 0.9918
- val_loss: 0.0944 - val_acc: 0.9781
Epoch 13/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.0233 - acc: 0.9924
- val_loss: 0.0943 - val_acc: 0.9780
Epoch 14/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.0223 - acc: 0.9929
- val_loss: 0.0967 - val_acc: 0.9768
Epoch 15/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.0175 - acc: 0.9942
- val_loss: 0.1023 - val_acc: 0.9771
Epoch 16/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.0166 - acc: 0.9946
- val_loss: 0.0940 - val_acc: 0.9784
Epoch 17/20
60000/60000 [==============================] - 3s 46us/step - loss: 0.0159 - acc: 0.9951
- val_loss: 0.1080 - val_acc: 0.9759
Epoch 18/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.0198 - acc: 0.9937
- val_loss: 0.1146 - val_acc: 0.9758
Epoch 19/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0127 - acc: 0.9959
- val_loss: 0.1031 - val_acc: 0.9774
Epoch 20/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.0180 - acc: 0.9943
- val_loss: 0.1213 - val_acc: 0.9743
```

In [93]:

```
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, verb
ose=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 epoc
hs

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

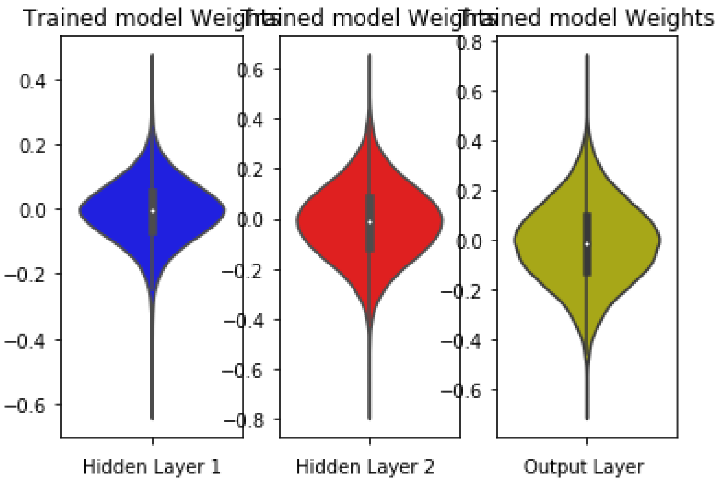
```
Test score: 0.12134343287702913
Test accuracy: 0.9743
```





In [94]:

```
w_after = model_relu.get_weights()

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



With Dropout

In [85]:

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

from keras.layers import Dropout

model_drop = Sequential()
```

```
model_drop.add(Dense(250, activation='relu', input_shape=(input_dim,),kernel_initializer=
RandomNormal(mean=0.0, stddev=0.125, seed=None)))
model_drop.add(Dropout(0.5))

model_drop.add(Dense(150, activation='relu',
                     kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)
))
model_drop.add(Dropout(0.5))

model_drop.add(Dense(146, activation='relu',
                     kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)
))
model_drop.add(Dropout(0.5))
model_drop.add(Dense(60, activation='relu',
                     kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)
))
model_drop.add(Dropout(0.5))

model_drop.add(Dense(40, activation='relu',
                     kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)
))
model_drop.add(Dropout(0.5))

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

```
Model: "sequential_28"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_87 (Dense)             (None, 250)               196250
_________________________________________________________________
dropout_13 (Dropout)         (None, 250)               0
_________________________________________________________________
dense_88 (Dense)             (None, 150)               37650
_________________________________________________________________
dropout_14 (Dropout)         (None, 150)               0
_________________________________________________________________
dense_89 (Dense)             (None, 146)               22046
_________________________________________________________________
dropout_15 (Dropout)         (None, 146)               0
_________________________________________________________________
dense_90 (Dense)             (None, 60)                8820
_________________________________________________________________
dropout_16 (Dropout)         (None, 60)                0
_________________________________________________________________
dense_91 (Dense)             (None, 40)                2440
_________________________________________________________________
dropout_17 (Dropout)         (None, 40)                0
_________________________________________________________________
dense_92 (Dense)             (None, 10)                410
=================================================================
Total params: 267,616
Trainable params: 267,616
Non-trainable params: 0
_________________________________________________________________
None
```

In [86]:

```
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, verbo
se=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 6s 92us/step - loss: 2.3843 - acc: 0.1383
- val_loss: 1.8483 - val_acc: 0.4532
```

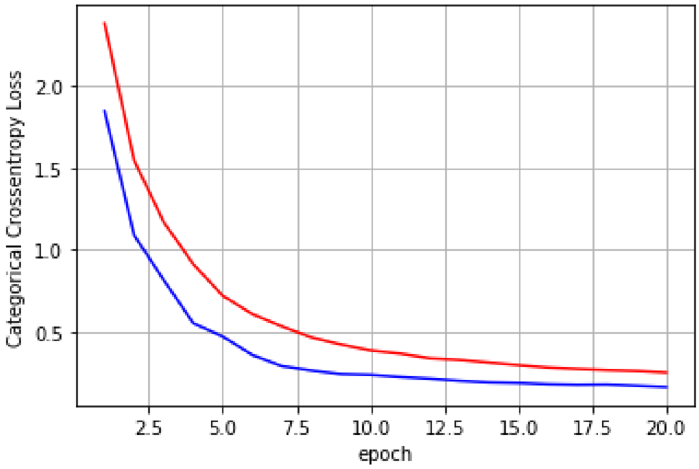
```
Epoch 2/20
60000/60000 [==============================] - 3s 58us/step - loss: 1.5475 - acc: 0.4169
- val_loss: 1.0906 - val_acc: 0.6451
Epoch 3/20
60000/60000 [==============================] - 3s 54us/step - loss: 1.1712 - acc: 0.5643
- val_loss: 0.8176 - val_acc: 0.7626
Epoch 4/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.9148 - acc: 0.6899
- val_loss: 0.5543 - val_acc: 0.8483
Epoch 5/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.7204 - acc: 0.7777
- val_loss: 0.4719 - val_acc: 0.8741
Epoch 6/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.6096 - acc: 0.8303
- val_loss: 0.3595 - val_acc: 0.9196
Epoch 7/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.5343 - acc: 0.8612
- val_loss: 0.2917 - val_acc: 0.9379
Epoch 8/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.4666 - acc: 0.8813
- val_loss: 0.2648 - val_acc: 0.9429
Epoch 9/20
60000/60000 [==============================] - 3s 58us/step - loss: 0.4232 - acc: 0.8957
- val_loss: 0.2428 - val_acc: 0.9481
Epoch 10/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.3869 - acc: 0.9049
- val_loss: 0.2383 - val_acc: 0.9479
Epoch 11/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.3681 - acc: 0.9101
- val_loss: 0.2254 - val_acc: 0.9515
Epoch 12/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.3393 - acc: 0.9177
- val_loss: 0.2145 - val_acc: 0.9558
Epoch 13/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.3302 - acc: 0.9222
- val_loss: 0.2015 - val_acc: 0.9576
Epoch 14/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.3135 - acc: 0.9253
- val_loss: 0.1920 - val_acc: 0.9582
Epoch 15/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.2979 - acc: 0.9295
- val_loss: 0.1886 - val_acc: 0.9617
Epoch 16/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.2828 - acc: 0.9327
- val_loss: 0.1806 - val_acc: 0.9624
Epoch 17/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.2742 - acc: 0.9351
- val_loss: 0.1774 - val_acc: 0.9638
Epoch 18/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.2672 - acc: 0.9374
- val_loss: 0.1788 - val_acc: 0.9645
Epoch 19/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.2624 - acc: 0.9383
- val_loss: 0.1715 - val_acc: 0.9653
Epoch 20/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.2526 - acc: 0.9402
- val_loss: 0.1637 - val_acc: 0.9662
```

In [88]:

```
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())
```

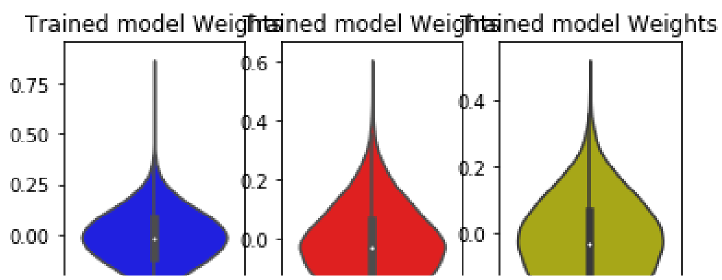
```
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verb
ose=1, validation_data=(X_test, Y_test))

# 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 epoc
hs

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

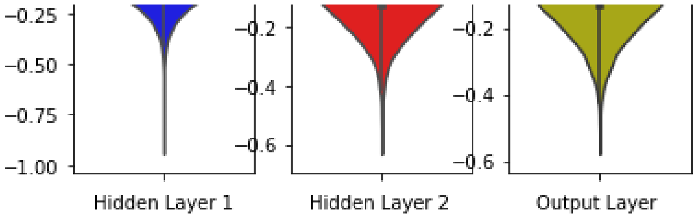
```
Test score: 0.16367745212987064
Test accuracy: 0.9662
```



In [90]:

```
w_after = model_drop.get_weights()

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


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

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

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





In [0]:

```

```

In [0]:

```

```

In [0]:

```

```

With Batch Normalization

In [0]:

```

```

In [95]:

```
# 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(250, activation='relu', input_shape=(input_dim,), kernel_initialize
r=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_batch.add(BatchNormalization())

model_batch.add(Dense(150, activation='relu', kernel_initializer=RandomNormal(mean=0.0, s
tddev=0.55, seed=None)) )
model_batch.add(BatchNormalization())

model_batch.add(Dense(146, activation='relu', kernel_initializer=RandomNormal(mean=0.0, s
tddev=0.15, seed=None)) )
model_batch.add(BatchNormalization())

model_drop.add(Dense(60, activation='relu', kernel_initializer=RandomNormal(mean=0.0, std
dev=0.2, seed=None)) )
model_batch.add(BatchNormalization())

model_drop.add(Dense(40, activation='relu', kernel_initializer=RandomNormal(mean=0.0, std
dev=0.6, seed=None)) )
model_batch.add(BatchNormalization())


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


model_batch.summary()
```

```
Model: "sequential_30"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_99 (Dense)             (None, 250)               196250
_________________________________________________________________
batch_normalization_15 (Batc (None, 250)               1000
_________________________________________________________________
dense_100 (Dense)            (None, 150)               37650
_________________________________________________________________
batch_normalization_16 (Batc (None, 150)               600
_________________________________________________________________
dense_101 (Dense)            (None, 146)               22046
_________________________________________________________________
batch_normalization_17 (Batc (None, 146)               584
_________________________________________________________________
batch_normalization_18 (Batc (None, 146)               584
_________________________________________________________________
batch_normalization_19 (Batc (None, 146)               584
_________________________________________________________________
dense_104 (Dense)            (None, 10)                1470
=================================================================
Total params: 260,768
Trainable params: 259,092
Non-trainable params: 1,676
_________________________________________________________________
```

In [96]:

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

history = model_batch.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verb
ose=1, validation_data=(X_test, Y_test))
```

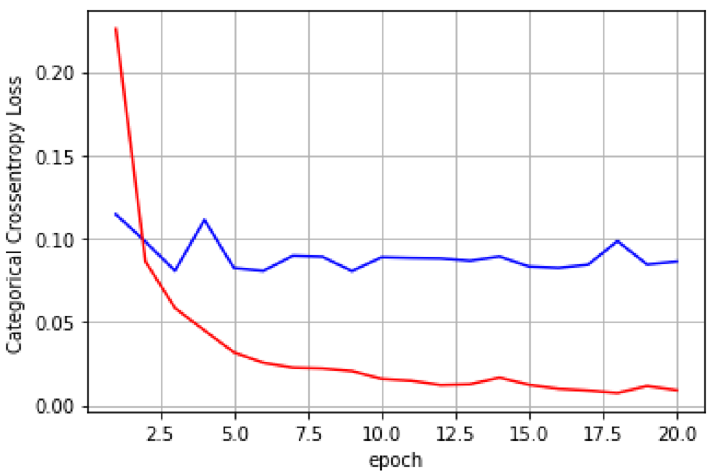
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.2259 - acc: 0.9315
- val_loss: 0.1145 - val_acc: 0.9643
Epoch 2/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0863 - acc: 0.9736
- val_loss: 0.0981 - val_acc: 0.9702
Epoch 3/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.0583 - acc: 0.9819
- val_loss: 0.0806 - val_acc: 0.9726
Epoch 4/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.0449 - acc: 0.9854
- val_loss: 0.1113 - val_acc: 0.9627
Epoch 5/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.0316 - acc: 0.9900
- val_loss: 0.0823 - val_acc: 0.9744
Epoch 6/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0255 - acc: 0.9918
- val_loss: 0.0806 - val_acc: 0.9775
Epoch 7/20
60000/60000 [==============================] - 6s 98us/step - loss: 0.0225 - acc: 0.9923
- val_loss: 0.0897 - val_acc: 0.9726
Epoch 8/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0221 - acc: 0.9930
- val_loss: 0.0891 - val_acc: 0.9734
Epoch 9/20
60000/60000 [==============================] - 6s 103us/step - loss: 0.0205 - acc: 0.9927
- val_loss: 0.0805 - val_acc: 0.9770
Epoch 10/20
60000/60000 [==============================] - 6s 99us/step - loss: 0.0158 - acc: 0.9947
- val_loss: 0.0888 - val_acc: 0.9737
Epoch 11/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0147 - acc: 0.9954
- val_loss: 0.0883 - val_acc: 0.9769
Epoch 12/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.0120 - acc: 0.9959
- val_loss: 0.0881 - val_acc: 0.9772
Epoch 13/20
60000/60000 [==============================] - 6s 102us/step - loss: 0.0125 - acc: 0.9961
- val_loss: 0.0867 - val_acc: 0.9764
```

```
- val_loss: 0.0867 - val_acc: 0.9764
Epoch 14/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0166 - acc: 0.9942
- val_loss: 0.0892 - val_acc: 0.9759
Epoch 15/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.0122 - acc: 0.9958
- val_loss: 0.0832 - val_acc: 0.9788
Epoch 16/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.0098 - acc: 0.9968
- val_loss: 0.0823 - val_acc: 0.9788
Epoch 17/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.0088 - acc: 0.9971
- val_loss: 0.0844 - val_acc: 0.9784
Epoch 18/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.0073 - acc: 0.9975
- val_loss: 0.0985 - val_acc: 0.9755
Epoch 19/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0115 - acc: 0.9963
- val_loss: 0.0845 - val_acc: 0.9784
Epoch 20/20
60000/60000 [==============================] - 6s 101us/step - loss: 0.0090 - acc: 0.9970
- val_loss: 0.0862 - val_acc: 0.9792
```

In [97]:

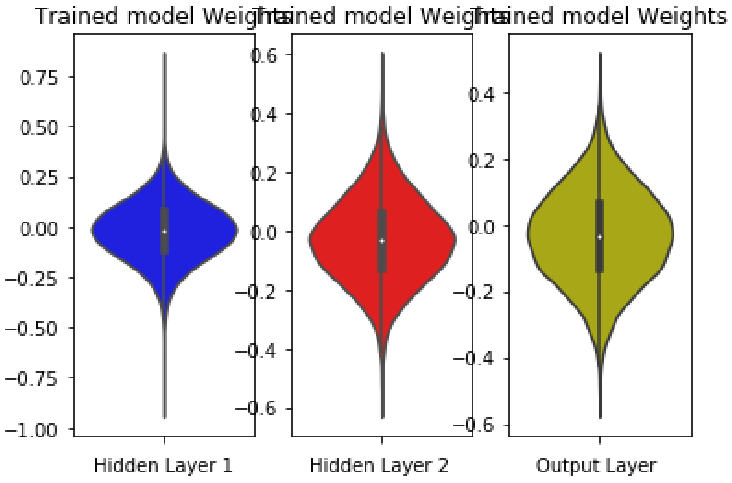
```python
# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verb
ose=1, validation_data=(X_test, Y_test))

# 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 epoc
hs

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))

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

```
Test score: 0.08622197609168743
Test accuracy: 0.9792
```



In [98]:

```
w_after = model_drop.get_weights()

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


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

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

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



In [0]:

```
```

In [0]:

```
```

Batch Normalization + Dropout

In [99]:

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

from keras.layers import Dropout

model3 = Sequential()

model3.add(Dense(250, activation='relu', input_shape=(input_dim,), kernel_initializer=Ran
domNormal(mean=0.0, stddev=0.039, seed=None)))
model3.add(BatchNormalization())
model3.add(Dropout(0.5))
```

```
model3.add(Dense(150, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev
=0.55, seed=None)) )
model3.add(BatchNormalization())
model3.add(Dropout(0.5))

model3.add(Dense(146, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev
=0.15, seed=None)) )
model3.add(BatchNormalization())
model3.add(Dropout(0.5))

model3.add(Dense(60, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev
=0.2, seed=None)) )
model3.add(BatchNormalization())
model3.add(Dropout(0.5))

model3.add(Dense(40, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev
=0.6, seed=None)) )
model3.add(BatchNormalization())
model3.add(Dropout(0.5))


model3.add(Dense(output_dim, activation='softmax'))


model3.summary()
```

```
Model: "sequential_31"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_105 (Dense)            (None, 250)               196250    
_________________________________________________________________
batch_normalization_20 (Batc (None, 250)               1000      
_________________________________________________________________
dropout_18 (Dropout)         (None, 250)               0         
_________________________________________________________________
dense_106 (Dense)            (None, 150)               37650     
_________________________________________________________________
batch_normalization_21 (Batc (None, 150)               600       
_________________________________________________________________
dropout_19 (Dropout)         (None, 150)               0         
_________________________________________________________________
dense_107 (Dense)            (None, 146)               22046     
_________________________________________________________________
batch_normalization_22 (Batc (None, 146)               584       
_________________________________________________________________
dropout_20 (Dropout)         (None, 146)               0         
_________________________________________________________________
dense_108 (Dense)            (None, 60)                8820      
_________________________________________________________________
batch_normalization_23 (Batc (None, 60)                240       
_________________________________________________________________
dropout_21 (Dropout)         (None, 60)                0         
_________________________________________________________________
dense_109 (Dense)            (None, 40)                2440      
_________________________________________________________________
batch_normalization_24 (Batc (None, 40)                160       
_________________________________________________________________
dropout_22 (Dropout)         (None, 40)                0         
_________________________________________________________________
dense_110 (Dense)            (None, 10)                410       
=================================================================
Total params: 270,200
Trainable params: 268,908
Non-trainable params: 1,292
_________________________________________________________________
```

In [100]:

```
model3.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model3.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1
```

```
, validation_data=(X_test, Y_test))
```

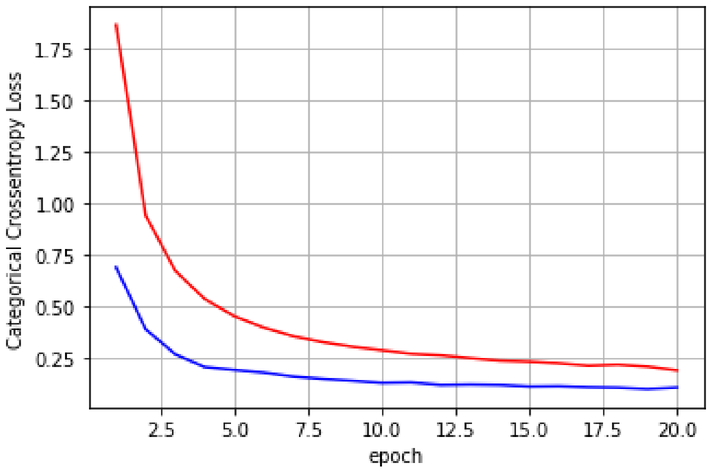
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 11s 176us/step - loss: 1.8646 - acc: 0.372
4 - val_loss: 0.6914 - val_acc: 0.8356
Epoch 2/20
60000/60000 [==============================] - 7s 113us/step - loss: 0.9432 - acc: 0.6832
- val_loss: 0.3925 - val_acc: 0.9030
Epoch 3/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.6758 - acc: 0.7879
- val_loss: 0.2720 - val_acc: 0.9294
Epoch 4/20
60000/60000 [==============================] - 7s 114us/step - loss: 0.5389 - acc: 0.8472
- val_loss: 0.2094 - val_acc: 0.9448
Epoch 5/20
60000/60000 [==============================] - 7s 110us/step - loss: 0.4560 - acc: 0.8775
- val_loss: 0.1955 - val_acc: 0.9483
Epoch 6/20
60000/60000 [==============================] - 6s 108us/step - loss: 0.4007 - acc: 0.8953
- val_loss: 0.1827 - val_acc: 0.9521
Epoch 7/20
60000/60000 [==============================] - 7s 113us/step - loss: 0.3588 - acc: 0.9081
- val_loss: 0.1637 - val_acc: 0.9573
Epoch 8/20
60000/60000 [==============================] - 7s 112us/step - loss: 0.3302 - acc: 0.9183
- val_loss: 0.1518 - val_acc: 0.9612
Epoch 9/20
60000/60000 [==============================] - 7s 115us/step - loss: 0.3078 - acc: 0.9257
- val_loss: 0.1429 - val_acc: 0.9649
Epoch 10/20
60000/60000 [==============================] - 6s 108us/step - loss: 0.2910 - acc: 0.9299
- val_loss: 0.1329 - val_acc: 0.9658
Epoch 11/20
60000/60000 [==============================] - 7s 109us/step - loss: 0.2740 - acc: 0.9345
- val_loss: 0.1351 - val_acc: 0.9684
Epoch 12/20
60000/60000 [==============================] - 7s 111us/step - loss: 0.2668 - acc: 0.9364
- val_loss: 0.1231 - val_acc: 0.9694
Epoch 13/20
60000/60000 [==============================] - 7s 110us/step - loss: 0.2528 - acc: 0.9393
- val_loss: 0.1252 - val_acc: 0.9697
Epoch 14/20
60000/60000 [==============================] - 7s 125us/step - loss: 0.2406 - acc: 0.9432
- val_loss: 0.1233 - val_acc: 0.9690
Epoch 15/20
60000/60000 [==============================] - 7s 112us/step - loss: 0.2359 - acc: 0.9446
- val_loss: 0.1157 - val_acc: 0.9732
Epoch 16/20
60000/60000 [==============================] - 7s 117us/step - loss: 0.2280 - acc: 0.9461
- val_loss: 0.1172 - val_acc: 0.9729
Epoch 17/20
60000/60000 [==============================] - 6s 108us/step - loss: 0.2165 - acc: 0.9489
- val_loss: 0.1124 - val_acc: 0.9727
Epoch 18/20
60000/60000 [==============================] - 7s 112us/step - loss: 0.2213 - acc: 0.9493
- val_loss: 0.1105 - val_acc: 0.9736
Epoch 19/20
60000/60000 [==============================] - 7s 110us/step - loss: 0.2119 - acc: 0.9508
- val_loss: 0.1037 - val_acc: 0.9759
Epoch 20/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.1936 - acc: 0.9545
- val_loss: 0.1110 - val_acc: 0.9735
```

In [101]:

```
score = model3.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')
```

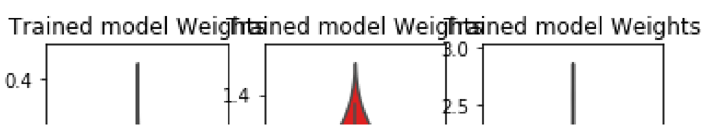
```
# list of epoch numbers
x = list(range(1,nb_epoch+1))

# 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, verb
ose=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 epoc
hs

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

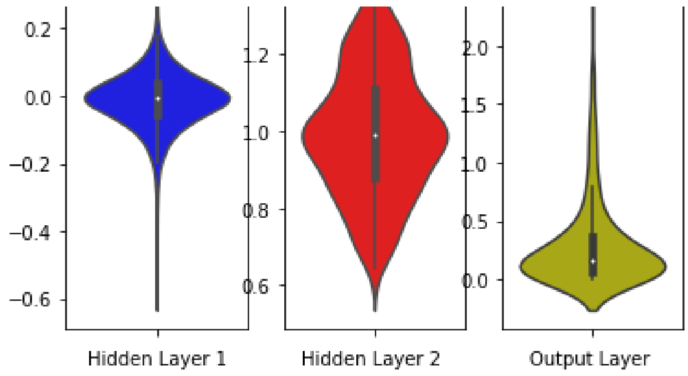
```
Test score: 0.11095908558527008
Test accuracy: 0.9735
```



In [102]:

```
w_after = model3.get_weights()

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


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

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

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





In [0]:

```

```

In [0]:

```

```

CONCLUSION

In [103]:

```
from prettytable import PrettyTable

x=PrettyTable()

x.field_names=(['No.of.Layers','Layers in MLP','Model Type','Test Score','Test accuracy'
])

x.add_row(['2-Layered','352-52','Without Dropout and BN',0.10,0.97])
x.add_row(['2-Layered','352-52','With Dropout',0.13,0.96])
x.add_row(['2-Layered','352-52','With BN',0.08,0.98])
x.add_row(['2-Layered','352-52','Dropout+BN',0.072,0.97])


print(x)
```

```
+--------------+---------------+------------------------+------------+---------------+
| No.of.Layers | Layers in MLP |       Model Type       | Test Score | Test accuracy |
+--------------+---------------+------------------------+------------+---------------+
|  2-Layered   |     352-52    | Without Dropout and BN |    0.1     |      0.97     |
|  2-Layered   |     352-52    |      With Dropout      |    0.13    |      0.96     |
|  2-Layered   |     352-52    |        With BN         |    0.08    |      0.98     |
|  2-Layered   |     352-52    |       Dropout+BN       |   0.072    |      0.97     |
+--------------+---------------+------------------------+------------+---------------+
```

In [104]:

```
y=PrettyTable()

y.field_names=(['No.of.Layers','Layers in MLP','Model Type','Test Score','Test Value'])

y.add_row(['3-Layered','352-52-102','Without Dropout and BN',0.15,0.975])
y.add_row(['3-Layered','352-52-102','With Dropout',0.14,0.967])
y.add_row(['3-Layered','352-52-102','With BN',0.09,0.979])
y.add_row(['3-Layered','352-52-102','Dropout+BN',0.07,0.978])


print(y)
```

```
+--------------+---------------+------------------------+------------+------------+
| No.of.Layers | Layers in MLP |       Model Type       | Test Score | Test Value |
+--------------+---------------+------------------------+------------+------------+
|  3-Layered   |   352-52-102  | Without Dropout and BN |    0.15    |   0.975    |
|  3-Layered   |   352-52-102  |      With Dropout      |    0.14    |   0.967    |
|  3-Layered   |   352-52-102  |        With BN         |    0.09    |   0.979    |
|  3-Layered   |   352-52-102  |       Dropout+BN       |    0.07    |   0.978    |
```

```
+--------------+---------------+-----------------------+------------+------------+
```

In [108]:

```python
z=PrettyTable()

z.field_names=(['No.of.Layers','Layers in MLP','Model Type','Test Score','Test Value'])

z.add_row(['5-Layered','250-150-146-60-40','With Dropout and BN',0.12,0.974])
z.add_row(['5-Layered','250-150-146-60-40','With Dropout',0.16,0.966])
z.add_row(['5-Layered','250-150-146-60-40','With BN',0.08,0.979])
z.add_row(['5-Layered','250-150-146-60-40','Dropout+BN',0.11,0.973])


print(z)
```

```
+--------------+-------------------+---------------------+------------+------------+
| No.of.Layers |   Layers in MLP   |      Model Type     | Test Score | Test Value |
+--------------+-------------------+---------------------+------------+------------+
|  5-Layered   | 250-150-146-60-40 | With Dropout and BN |    0.12    |   0.974    |
|  5-Layered   | 250-150-146-60-40 |     With Dropout    |    0.16    |   0.966    |
|  5-Layered   | 250-150-146-60-40 |       With BN       |    0.08    |   0.979    |
|  5-Layered   | 250-150-146-60-40 |      Dropout+BN     |    0.11    |   0.973    |
+--------------+-------------------+---------------------+------------+------------+
```

1)Here we have use Mutli-Layered perceptrons Architecture where we have used 2-Layered, 3-Layered and 5-Layered Structures, with different number of neurons.
2)By using 2-layered Architecture with 352-52 neurons, we have seen that by adding Batch Normalization gave highest accuracy.
3)By using 3-Layered Architecture with 352-52-102 neurons, we have seen that batch normalization and both droput and batch normalization gave highest accuracy
4)By using 5-Layered Architecture with 250-150-146-60-40 neurons, adding batch normalization gave highest accuracy.

In [0]:

```

```

In [0]:

```

```