Keras -- MLPs on MNIST

In [1]:

```
# 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
```

```
Using TensorFlow backend.
```

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_plot(x, vy, ty, ax, colors=['b']):
    fig , ax = plt.subplots(1,1)
    ax.plot(x, vy, 'b', label="Validation Loss")
    ax.plot(x, ty, 'r', label="Train Loss")
    plt.legend()
    plt.grid()
    fig.canvas.draw()
```

In [3]:

```
# 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 [4]:

```
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 [6]:

```
# 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 [8]:

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

```
[   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   3  18  18  18 126 136 175  26 166 255
  247 127   0   0   0   0   0   0   0   0   0   0   0   0  30  36  94 154
  170 253 253 253 253 253 225 172 253 242 195  64   0   0   0   0   0   0
    0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251  93  82
   82  56  39   0   0   0   0   0   0   0   0   0   0   0   0  18 219 253
  253 253 253 253 198 182 247 241   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0  14   1 154 253  90   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0  11 190 253  70   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  35 241
  225 160 108   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0  81 240 253 253 119  25   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0  45 186 253 253 150  27   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252 253 187
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0 249 253 249  64   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253
  253 207   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0  39 148 229 253 253 253 250 182   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253
  253 201  78   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0  23  66 213 253 253 253 253 198  81   2   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0  18 171 219 253 253 253 253 195
   80   9   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   55 172 226 253 253 253 253 244 133  11   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0 136 253 253 253 212 135 132  16
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
```

In [0]:

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

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

In [10]:

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

```
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
```

```
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         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.         0.97647059 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 [11]:

```
# 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 constructor:

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

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

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

###

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

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

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

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

####

# https://keras.io/activations/

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

# from keras.layers import Activation, Dense

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

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

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


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

In [0]:

```
# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 20
```

In [0]:

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

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

# output_dim represent the number of nodes need in that layer
```

```
# here we have 10 nodes

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

In [0]:

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

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


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

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

# that is why we converted out labels into vectors

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

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

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

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

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

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

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 3s 49us/step - loss: 1.2935 - acc: 0.6829 -
val_loss: 0.8171 - val_acc: 0.8312
Epoch 2/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.7213 - acc: 0.8385 -
val_loss: 0.6099 - val_acc: 0.8623
Epoch 3/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.5904 - acc: 0.8584 -
val_loss: 0.5275 - val_acc: 0.8741
Epoch 4/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.5278 - acc: 0.8682 -
val_loss: 0.4815 - val_acc: 0.8814
Epoch 5/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.4897 - acc: 0.8747 -
val_loss: 0.4515 - val_acc: 0.8870
Epoch 6/20
34432/60000 [================>.............] - ETA: 0s - loss: 0.4697 - acc: 0.877360000/60000 [==
============================] - 2s 35us/step - loss: 0.4635 - acc: 0.8799 - val_loss: 0.4303 - val
_acc: 0.8898
Epoch 7/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.4442 - acc: 0.8837 -
val_loss: 0.4138 - val_acc: 0.8917
Epoch 8/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.4290 - acc: 0.8866 -
val_loss: 0.4010 - val_acc: 0.8952
Epoch 9/20
```

```
Epoch 9/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.4169 - acc: 0.8894 -
val_loss: 0.3909 - val_acc: 0.8971
Epoch 10/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.4067 - acc: 0.8911 -
val_loss: 0.3818 - val_acc: 0.8994
Epoch 11/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.3982 - acc: 0.8926 -
val_loss: 0.3743 - val_acc: 0.9006
Epoch 12/20
 1792/60000 [..............................] - ETA: 1s - loss: 0.4077 - acc: 0.889560000/60000 [==
============================] - 2s 35us/step - loss: 0.3908 - acc: 0.8948 - val_loss: 0.3681 - val
_acc: 0.9020
Epoch 13/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.3844 - acc: 0.8960 -
val_loss: 0.3624 - val_acc: 0.9039
Epoch 14/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.3786 - acc: 0.8972 -
val_loss: 0.3575 - val_acc: 0.9046
Epoch 15/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.3735 - acc: 0.8981 -
val_loss: 0.3528 - val_acc: 0.9058
Epoch 16/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.3689 - acc: 0.8993 -
val_loss: 0.3490 - val_acc: 0.9062
Epoch 17/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.3648 - acc: 0.9004 -
val_loss: 0.3455 - val_acc: 0.9066
Epoch 18/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.3610 - acc: 0.9016 -
val_loss: 0.3419 - val_acc: 0.9077
Epoch 19/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.3575 - acc: 0.9024 -
val_loss: 0.3389 - val_acc: 0.9088
Epoch 20/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.3544 - acc: 0.9032 -
val_loss: 0.3362 - val_acc: 0.9093
```

In [0]:

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

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

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

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

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

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

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

```
Test score: 0.3362289469957352
Test accuracy: 0.9093
```

MLP + Sigmoid activation + SGDOptimizer

In [0]:

```
# Multilayer perceptron

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

model_sigmoid.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_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 [0]:

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

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 3s 42us/step - loss: 2.2708 - acc: 0.2169 -
val_loss: 2.2197 - val_acc: 0.2768
Epoch 2/20
60000/60000 [==============================] - 2s 42us/step - loss: 2.1739 - acc: 0.4237 -
val_loss: 2.1143 - val_acc: 0.4877
Epoch 3/20
60000/60000 [==============================] - 2s 42us/step - loss: 2.0506 - acc: 0.5485 -
val_loss: 1.9659 - val_acc: 0.5524
Epoch 4/20
60000/60000 [==============================] - 3s 42us/step - loss: 1.8796 - acc: 0.6135 -
val_loss: 1.7673 - val_acc: 0.6481
Epoch 5/20
60000/60000 [==============================] - 3s 42us/step - loss: 1.6674 - acc: 0.6659 -
val_loss: 1.5414 - val_acc: 0.7205
Epoch 6/20
 5376/60000 [=>............................] - ETA: 2s - loss: 1.5430 - acc: 0.698160000/60000 [==
============================] - 2s 41us/step - loss: 1.4449 - acc: 0.7125 - val_loss: 1.3233 - val
_acc: 0.7513
Epoch 7/20
60000/60000 [==============================] - 2s 41us/step - loss: 1.2442 - acc: 0.7466 -
val_loss: 1.1406 - val_acc: 0.7599
Epoch 8/20
60000/60000 [==============================] - 2s 41us/step - loss: 1.0815 - acc: 0.7722 -
val_loss: 0.9974 - val_acc: 0.7968
Epoch 9/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.9544 - acc: 0.7940 -
val_loss: 0.8867 - val_acc: 0.8060
Epoch 10/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.8556 - acc: 0.8082 -
val_loss: 0.7984 - val_acc: 0.8227
Epoch 11/20
31232/60000 [==============>...............] - ETA: 1s - loss: 0.7920 - acc: 0.820760000/60000 [==
============================] - 2s 42us/step - loss: 0.7776 - acc: 0.8225 - val_loss: 0.7292 - val
_acc: 0.8335
Epoch 12/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.7154 - acc: 0.8333 -
val_loss: 0.6741 - val_acc: 0.8422
Epoch 13/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.6650 - acc: 0.8412 -
val_loss: 0.6282 - val_acc: 0.8500
Epoch 14/20
```

```
60000/60000 [==============================] - 2s 41us/step - loss: 0.6237 - acc: 0.8485 -
val_loss: 0.5906 - val_acc: 0.8585
Epoch 15/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.5893 - acc: 0.8546 -
val_loss: 0.5591 - val_acc: 0.8616
Epoch 16/20
34432/60000 [================>.............] - ETA: 0s - loss: 0.5691 - acc: 0.857860000/60000 [==
============================] - 2s 41us/step - loss: 0.5606 - acc: 0.8599 - val_loss: 0.5329 - val
_acc: 0.8672
Epoch 17/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.5361 - acc: 0.8641 -
val_loss: 0.5095 - val_acc: 0.8703
Epoch 18/20
60000/60000 [==============================] - 2s 42us/step - loss: 0.5151 - acc: 0.8676 -
val_loss: 0.4904 - val_acc: 0.8736
Epoch 19/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.4969 - acc: 0.8710 -
val_loss: 0.4732 - val_acc: 0.8782
Epoch 20/20
60000/60000 [==============================] - 2s 42us/step - loss: 0.4810 - acc: 0.8739 -
val_loss: 0.4583 - val_acc: 0.8816
```

In [0]:

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

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

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

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

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

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

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

```
Test score: 0.4582893396139145
Test accuracy: 0.8816
```

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)


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()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWarning: remove_na is dep
recated and is a private function. Do not use.
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarning: remove_na is dep
recated and is a private function. Do not use.
  violin_data = remove_na(group_data)
```

MLP + Sigmoid activation + ADAM

In [0]:

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

model_sigmoid.summary()

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

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

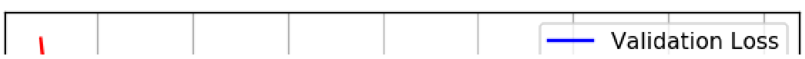
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_5 (Dense)              (None, 512)               401920    
_________________________________________________________________
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 [==============================] - 3s 55us/step - loss: 0.5308 - acc: 0.8636 - 
val_loss: 0.2550 - val_acc: 0.9259
Epoch 2/20
53120/60000 [=========================>....] - ETA: 0s - loss: 0.2244 - acc: 0.934060000/60000 [==
============================] - 3s 51us/step - loss: 0.2205 - acc: 0.9351 - val_loss: 0.1946 - val
_acc: 0.9417
Epoch 3/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.1650 - acc: 0.9512 - 
val_loss: 0.1421 - val_acc: 0.9570
Epoch 4/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.1279 - acc: 0.9614 - 
val_loss: 0.1238 - val_acc: 0.9645
Epoch 5/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.1010 - acc: 0.9704 - 
val_loss: 0.1029 - val_acc: 0.9693
Epoch 6/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0801 - acc: 0.9763 - 
val_loss: 0.0877 - val_acc: 0.9725
Epoch 7/20
 4480/60000 [=>............................] - ETA: 2s - loss: 0.0632 - acc: 0.979560000/60000 [==
============================] - 3s 51us/step - loss: 0.0645 - acc: 0.9809 - val_loss: 0.0831 - val
_acc: 0.9751
Epoch 8/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0519 - acc: 0.9842 - 
val_loss: 0.0724 - val_acc: 0.9780
Epoch 9/20
```

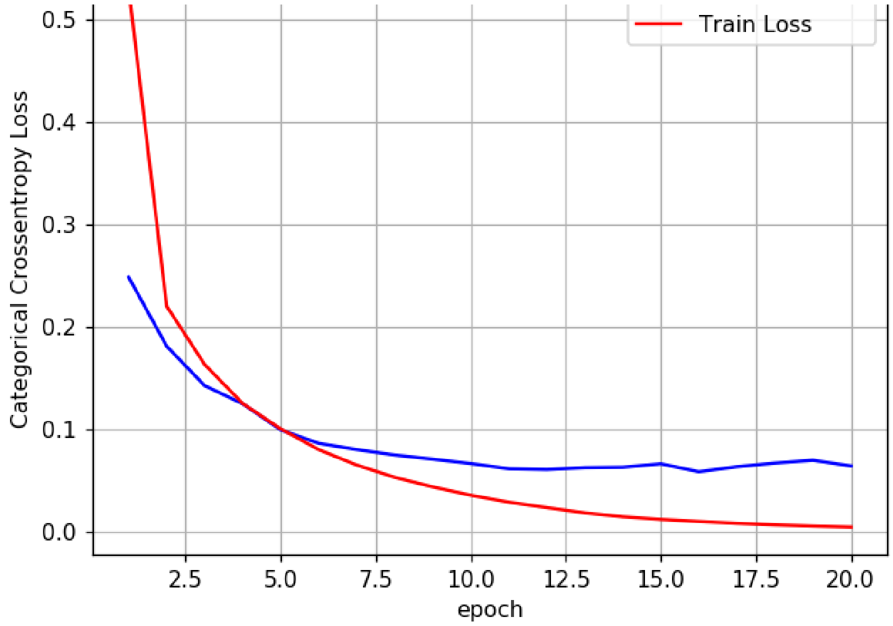
```
60000/60000 [==============================] - 3s 51us/step - loss: 0.0433 - acc: 0.9872 -
val_loss: 0.0714 - val_acc: 0.9786
Epoch 10/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0347 - acc: 0.9898 -
val_loss: 0.0695 - val_acc: 0.9776
Epoch 11/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0268 - acc: 0.9930 -
val_loss: 0.0659 - val_acc: 0.9796
Epoch 12/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0219 - acc: 0.9944 -
val_loss: 0.0642 - val_acc: 0.9809
Epoch 13/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0180 - acc: 0.9953 -
val_loss: 0.0677 - val_acc: 0.9794
Epoch 14/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0133 - acc: 0.9970 -
val_loss: 0.0647 - val_acc: 0.9803
Epoch 15/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0114 - acc: 0.9975 -
val_loss: 0.0628 - val_acc: 0.9812
Epoch 16/20
57344/60000 [===========================>..] - ETA: 0s - loss: 0.0085 - acc: 0.998260000/60000 [==
============================] - 3s 50us/step - loss: 0.0085 - acc: 0.9982 - val_loss: 0.0666 - val
_acc: 0.9806
Epoch 17/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0070 - acc: 0.9986 -
val_loss: 0.0643 - val_acc: 0.9822
Epoch 18/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0061 - acc: 0.9986 -
val_loss: 0.0656 - val_acc: 0.9818
Epoch 19/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0055 - acc: 0.9988 -
val_loss: 0.0811 - val_acc: 0.9774
Epoch 20/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0038 - acc: 0.9992 -
val_loss: 0.0723 - val_acc: 0.9818
```

In [0]:

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

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

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

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

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

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

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

```
Test score: 0.06385514608082886
Test accuracy: 0.9824
```





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)


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()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWarning: remove_na is dep
recated and is a private function. Do not use.
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarning: remove_na is dep
recated and is a private function. Do not use.
  violin_data = remove_na(group_data)
```

MLP + ReLU +SGD

In [0]:

```
# 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)
```

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

model_relu.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_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 [0]:

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

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

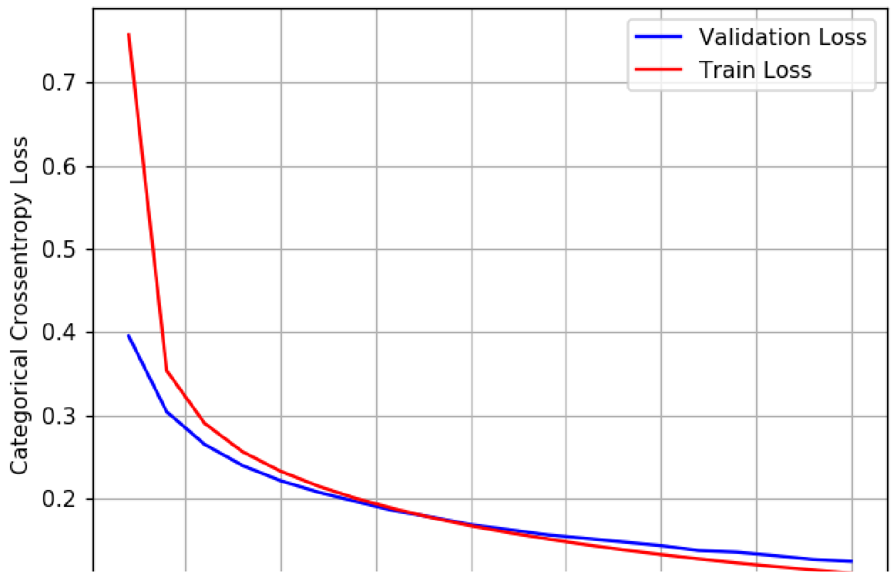
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.7579 - acc: 0.7812 -
val_loss: 0.3951 - val_acc: 0.8921
Epoch 2/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.3535 - acc: 0.8998 -
val_loss: 0.3040 - val_acc: 0.9153
Epoch 3/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.2900 - acc: 0.9172 -
val_loss: 0.2648 - val_acc: 0.9253
Epoch 4/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.2558 - acc: 0.9269 -
val_loss: 0.2393 - val_acc: 0.9316
Epoch 5/20
60000/60000 [==============================] - 4s 58us/step - loss: 0.2324 - acc: 0.9340 -
val_loss: 0.2210 - val_acc: 0.9371
Epoch 6/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.2144 - acc: 0.9391 -
val_loss: 0.2072 - val_acc: 0.9400
Epoch 7/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.1995 - acc: 0.9443 -
val_loss: 0.1957 - val_acc: 0.9444
Epoch 8/20
60000/60000 [==============================] - 4s 61us/step - loss: 0.1872 - acc: 0.9476 -
val_loss: 0.1848 - val_acc: 0.9456
Epoch 9/20
60000/60000 [==============================] - 3s 57us/step - loss: 0.1763 - acc: 0.9507 -
val_loss: 0.1771 - val_acc: 0.9488
Epoch 10/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.1668 - acc: 0.9539 -
val_loss: 0.1682 - val_acc: 0.9506
Epoch 11/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.1585 - acc: 0.9560 -
val_loss: 0.1623 - val_acc: 0.9518
Epoch 12/20
60000/60000 [==============================] - 7s 113us/step - loss: 0.1511 - acc: 0.9577 -
val_loss: 0.1560 - val_acc: 0.9543
Epoch 13/20
60000/60000 [==============================] - 7s 115us/step - loss: 0.1443 - acc: 0.9596 -
val_loss: 0.1517 - val_acc: 0.9557
Epoch 14/20
60000/60000 [==============================] - 7s 111us/step - loss: 0.1379 - acc: 0.9615 -
val_loss: 0.1474 - val_acc: 0.9572
Epoch 15/20
```

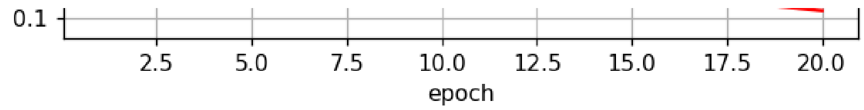
```
60000/60000 [==============================] - 4s 66us/step - loss: 0.1323 - acc: 0.9628 -
val_loss: 0.1429 - val_acc: 0.9580
Epoch 16/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.1270 - acc: 0.9645 -
val_loss: 0.1371 - val_acc: 0.9598
Epoch 17/20
60000/60000 [==============================] - 7s 110us/step - loss: 0.1221 - acc: 0.9661 -
val_loss: 0.1351 - val_acc: 0.9602
Epoch 18/20
60000/60000 [==============================] - 4s 62us/step - loss: 0.1177 - acc: 0.9671 -
val_loss: 0.1309 - val_acc: 0.9618
Epoch 19/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.1136 - acc: 0.9685 -
val_loss: 0.1263 - val_acc: 0.9631
Epoch 20/20
60000/60000 [==============================] - 5s 79us/step - loss: 0.1094 - acc: 0.9694 -
val_loss: 0.1241 - val_acc: 0.9631
```

In [0]:

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

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

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

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

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

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

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

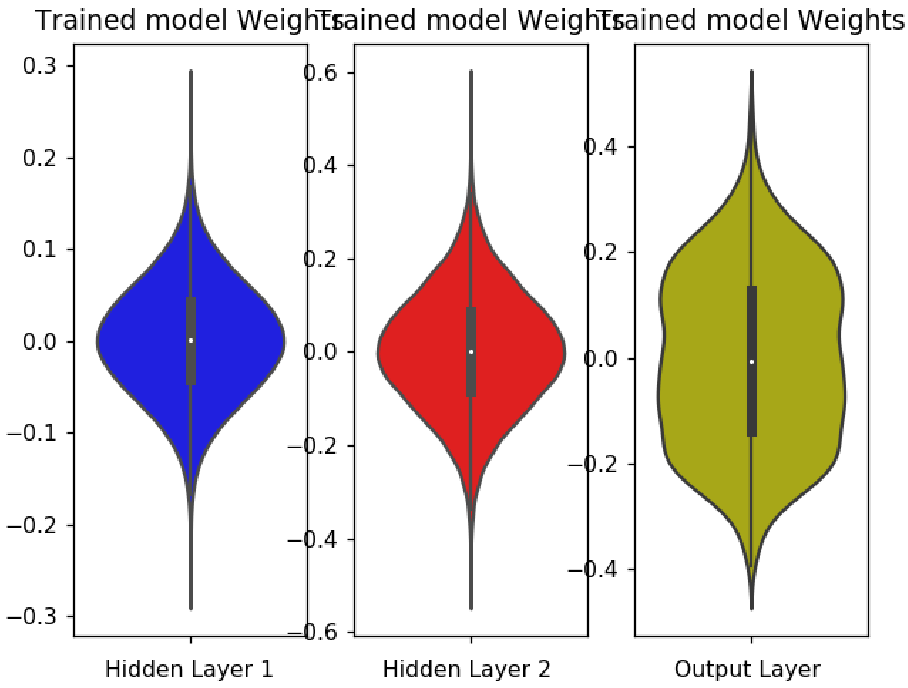
```
Test score: 0.12405014228336513
Test accuracy: 0.9631
```





In [0]:

```
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 [0]:

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

```python
print(model_relu.summary())

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

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

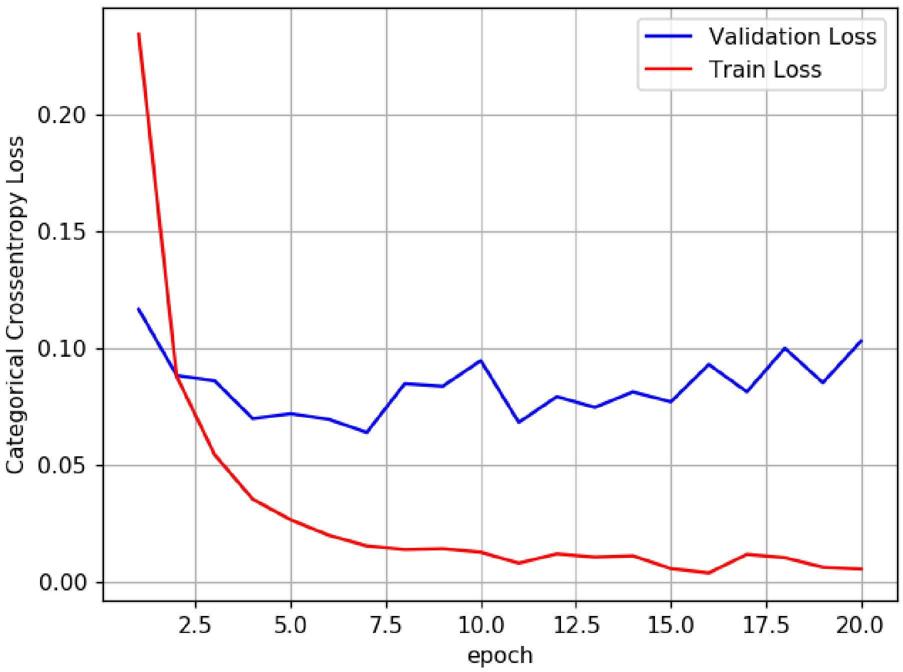
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_11 (Dense)             (None, 512)               401920    
_________________________________________________________________
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 [==============================] - 7s 121us/step - loss: 0.2341 - acc: 0.9295 -
val_loss: 0.1165 - val_acc: 0.9652
Epoch 2/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0878 - acc: 0.9729 -
val_loss: 0.0883 - val_acc: 0.9720
Epoch 3/20
60000/60000 [==============================] - 5s 75us/step - loss: 0.0544 - acc: 0.9825 -
val_loss: 0.0860 - val_acc: 0.9729
Epoch 4/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0354 - acc: 0.9885 -
val_loss: 0.0699 - val_acc: 0.9797
Epoch 5/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0266 - acc: 0.9914 -
val_loss: 0.0720 - val_acc: 0.9788
Epoch 6/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0200 - acc: 0.9941 -
val_loss: 0.0696 - val_acc: 0.9803
Epoch 7/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0155 - acc: 0.9951 -
val_loss: 0.0640 - val_acc: 0.9829
Epoch 8/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0140 - acc: 0.9952 -
val_loss: 0.0848 - val_acc: 0.9792
Epoch 9/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0143 - acc: 0.9952 -
val_loss: 0.0837 - val_acc: 0.9796
Epoch 10/20
60000/60000 [==============================] - 7s 115us/step - loss: 0.0128 - acc: 0.9958 -
val_loss: 0.0946 - val_acc: 0.9782
Epoch 11/20
60000/60000 [==============================] - 7s 125us/step - loss: 0.0081 - acc: 0.9974 -
val_loss: 0.0682 - val_acc: 0.9826
Epoch 12/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.0121 - acc: 0.9959 -
val_loss: 0.0793 - val_acc: 0.9816
Epoch 13/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.0107 - acc: 0.9963 -
val_loss: 0.0746 - val_acc: 0.9820
Epoch 14/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.0113 - acc: 0.9960 -
val_loss: 0.0813 - val_acc: 0.9816
Epoch 15/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.0058 - acc: 0.9982 -
val_loss: 0.0770 - val_acc: 0.9842
Epoch 16/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0040 - acc: 0.9987 -
val_loss: 0.0930 - val_acc: 0.9808
Epoch 17/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.0119 - acc: 0.9959 -
val_loss: 0.0813 - val_acc: 0.9819
Epoch 18/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0105 - acc: 0.9966 -
val_loss: 0.1000 - val_acc: 0.9803
```

```
Epoch 19/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0064 - acc: 0.9981 -
val_loss: 0.0852 - val_acc: 0.9831
Epoch 20/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.0056 - acc: 0.9982 -
val_loss: 0.1029 - val_acc: 0.9805
```

In [0]:

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

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

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

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

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

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


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

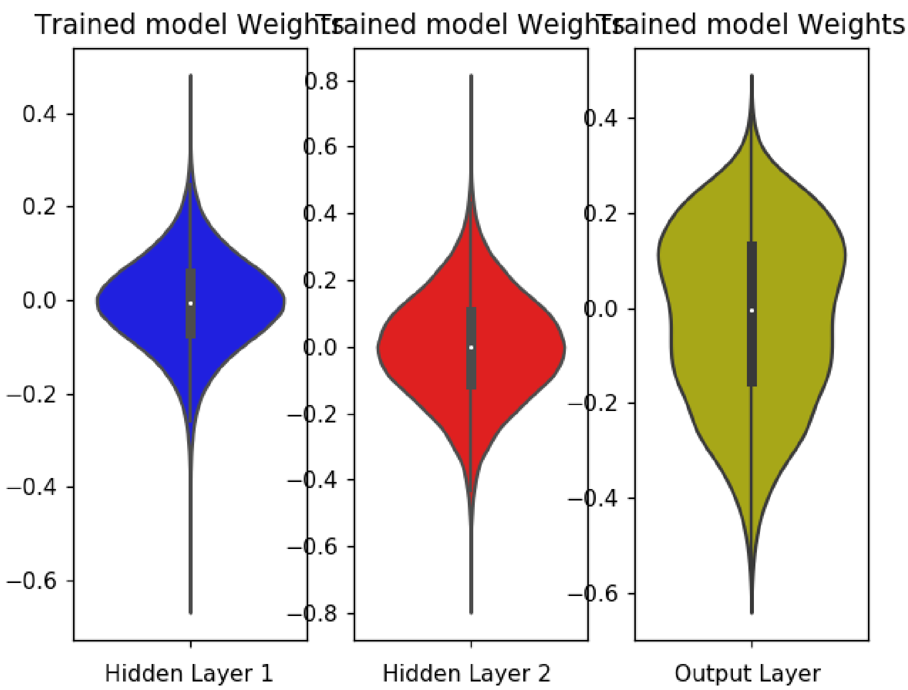
```
Test score: 0.10294274219236926
Test accuracy: 0.9805
```



In [0]:

```
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 [0]:

```
# Multilayer perceptron

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

from keras.layers.normalization import BatchNormalization

model_batch = Sequential()

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

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

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


model_batch.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_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 [0]:

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

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

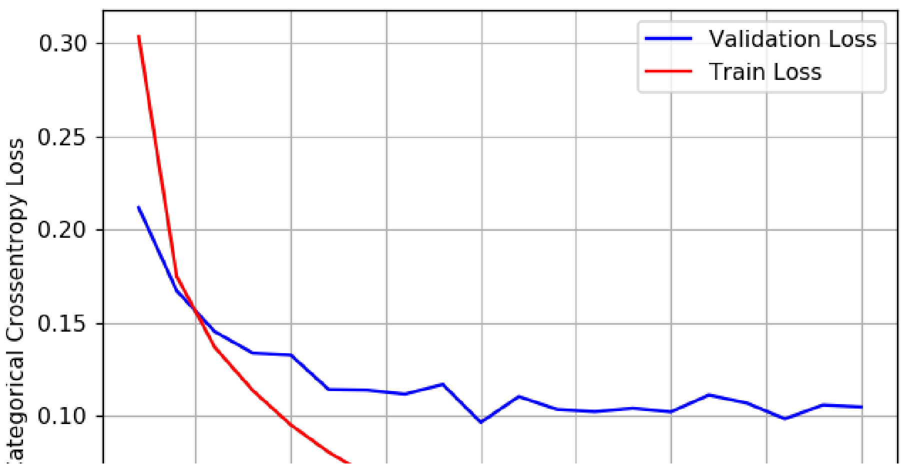
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.3036 - acc: 0.9104 -
val_loss: 0.2116 - val_acc: 0.9376
Epoch 2/20
60000/60000 [==============================] - 10s 170us/step - loss: 0.1747 - acc: 0.9483 - val_l
oss: 0.1670 - val_acc: 0.9505
Epoch 3/20
60000/60000 [==============================] - 13s 220us/step - loss: 0.1367 - acc: 0.9599 - val_l
oss: 0.1451 - val_acc: 0.9567
Epoch 4/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.1134 - acc: 0.9666 -
val_loss: 0.1335 - val_acc: 0.9603
Epoch 5/20
60000/60000 [==============================] - 13s 211us/step - loss: 0.0949 - acc: 0.9703 - val_l
oss: 0.1325 - val_acc: 0.9589
Epoch 6/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0802 - acc: 0.9758 -
val_loss: 0.1139 - val_acc: 0.9652
Epoch 7/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.0682 - acc: 0.9787 -
val_loss: 0.1136 - val_acc: 0.9666
Epoch 8/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.0608 - acc: 0.9815 -
val_loss: 0.1114 - val_acc: 0.9666
Epoch 9/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.0532 - acc: 0.9837 -
val_loss: 0.1167 - val_acc: 0.9666
Epoch 10/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.0455 - acc: 0.9856 -
val_loss: 0.0962 - val_acc: 0.9718
Epoch 11/20
60000/60000 [==============================] - 7s 112us/step - loss: 0.0376 - acc: 0.9880 -
val_loss: 0.1102 - val_acc: 0.9673
Epoch 12/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.0350 - acc: 0.9889 -
val_loss: 0.1033 - val_acc: 0.9710
Epoch 13/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.0308 - acc: 0.9903 -
val_loss: 0.1020 - val_acc: 0.9712
```

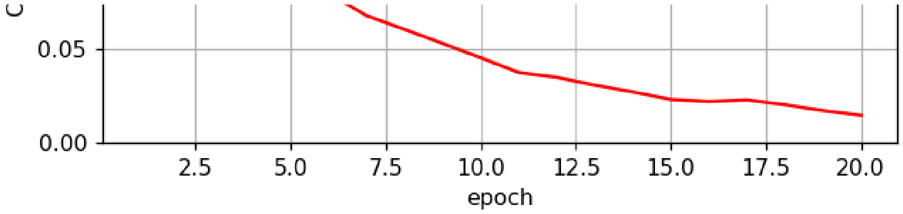
```
Epoch 14/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.0271 - acc: 0.9913 -
val_loss: 0.1038 - val_acc: 0.9727
Epoch 15/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.0231 - acc: 0.9926 -
val_loss: 0.1019 - val_acc: 0.9717
Epoch 16/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.0220 - acc: 0.9928 -
val_loss: 0.1110 - val_acc: 0.9703
Epoch 17/20
60000/60000 [==============================] - 7s 114us/step - loss: 0.0229 - acc: 0.9928 -
val_loss: 0.1067 - val_acc: 0.9739
Epoch 18/20
60000/60000 [==============================] - 8s 128us/step - loss: 0.0203 - acc: 0.9935 -
val_loss: 0.0982 - val_acc: 0.9738
Epoch 19/20
60000/60000 [==============================] - 7s 125us/step - loss: 0.0171 - acc: 0.9944 -
val_loss: 0.1056 - val_acc: 0.9706
Epoch 20/20
60000/60000 [==============================] - 11s 182us/step - loss: 0.0146 - acc: 0.9952 - val_l
oss: 0.1046 - val_acc: 0.9732
```

In [0]:

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

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

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

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

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

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

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

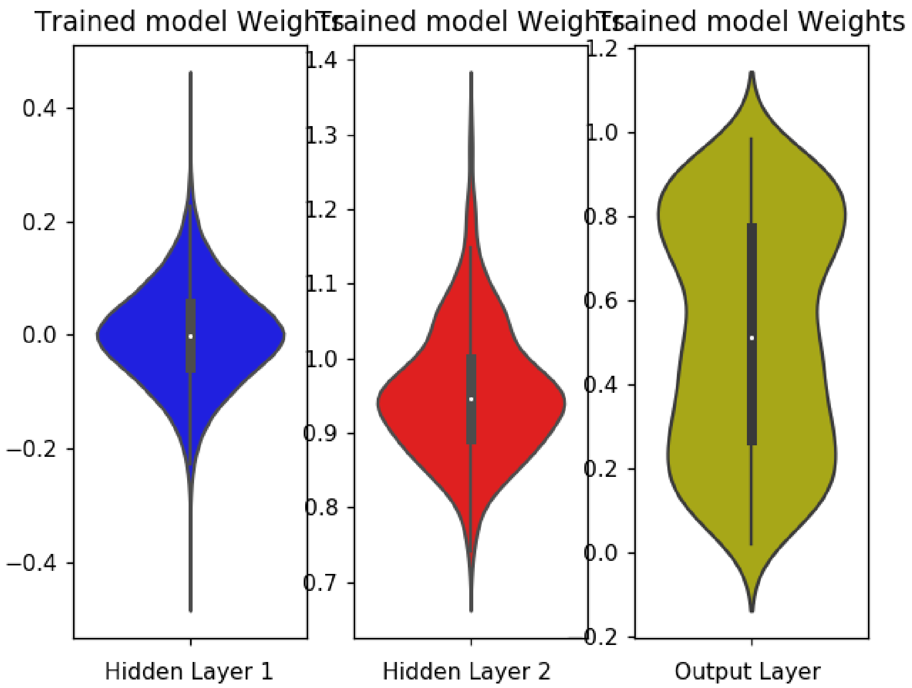
```
Test score: 0.10456635547156475
Test accuracy: 0.9732
```





In [0]:

```
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 [0]:

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

```python
from keras.layers import Dropout

model_drop = Sequential()

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

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

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


model_drop.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_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 [0]:

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

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

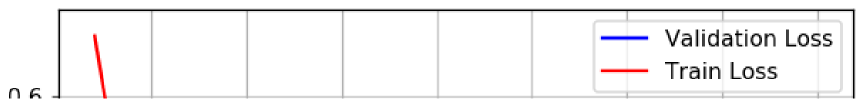
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 14s 227us/step - loss: 0.6612 - acc: 0.7951 - val_l
oss: 0.2860 - val_acc: 0.9166
Epoch 2/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.4250 - acc: 0.8710 -
val_loss: 0.2545 - val_acc: 0.9252
Epoch 3/20
60000/60000 [==============================] - 12s 198us/step - loss: 0.3841 - acc: 0.8846 - val_l
oss: 0.2391 - val_acc: 0.9298
Epoch 4/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.3551 - acc: 0.8927 -
val_loss: 0.2279 - val_acc: 0.9325
Epoch 5/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.3355 - acc: 0.8986 -
val_loss: 0.2127 - val_acc: 0.9356
Epoch 6/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.3234 - acc: 0.9031 -
val_loss: 0.2029 - val_acc: 0.9387: 1s - loss:
Epoch 7/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.3068 - acc: 0.9077 -
val_loss: 0.1927 - val_acc: 0.9421
Epoch 8/20
60000/60000 [==============================] - 11s 185us/step - loss: 0.2933 - acc: 0.9113 - val_l
oss: 0.1836 - val_acc: 0.9453
Epoch 9/20
60000/60000 [==============================] - 13s 222us/step - loss: 0.2850 - acc: 0.9131 - val_l
```

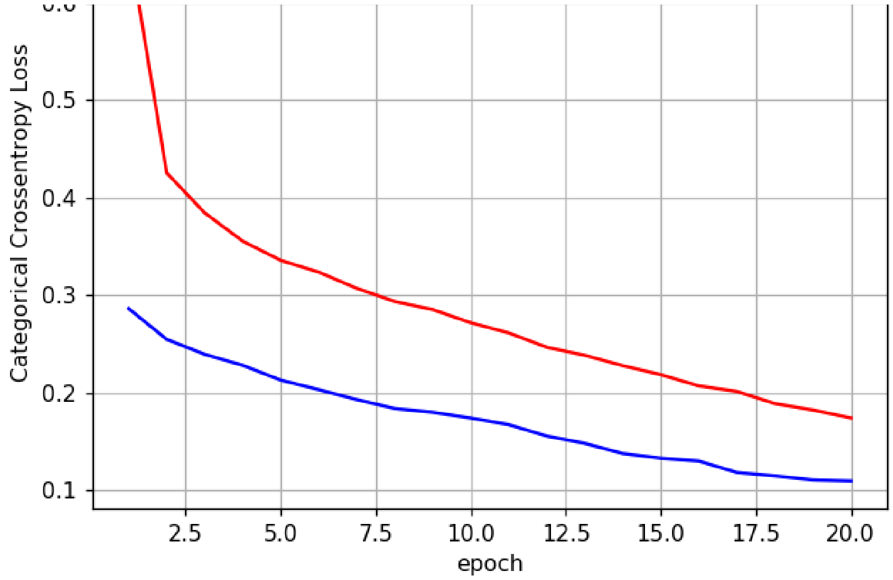
```
oss: 0.1797 - val_acc: 0.9451
Epoch 10/20
60000/60000 [==============================] - 14s 236us/step - loss: 0.2715 - acc: 0.9187 - val_l
oss: 0.1738 - val_acc: 0.9465
Epoch 11/20
60000/60000 [==============================] - 8s 141us/step - loss: 0.2611 - acc: 0.9214 -
val_loss: 0.1671 - val_acc: 0.9506
Epoch 12/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.2464 - acc: 0.9252 -
val_loss: 0.1554 - val_acc: 0.9525
Epoch 13/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.2382 - acc: 0.9278 -
val_loss: 0.1479 - val_acc: 0.9554
Epoch 14/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.2275 - acc: 0.9313 -
val_loss: 0.1375 - val_acc: 0.9580
Epoch 15/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.2183 - acc: 0.9337 -
val_loss: 0.1326 - val_acc: 0.9599
Epoch 16/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.2068 - acc: 0.9384 -
val_loss: 0.1297 - val_acc: 0.9613 loss: 0.2066 - ac
Epoch 17/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.2011 - acc: 0.9395 -
val_loss: 0.1181 - val_acc: 0.9646
Epoch 18/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.1886 - acc: 0.9435 -
val_loss: 0.1145 - val_acc: 0.9658
Epoch 19/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.1821 - acc: 0.9451 -
val_loss: 0.1104 - val_acc: 0.9662
Epoch 20/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.1739 - acc: 0.9473 -
val_loss: 0.1093 - val_acc: 0.9679
```

In [0]:

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

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

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

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

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

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

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

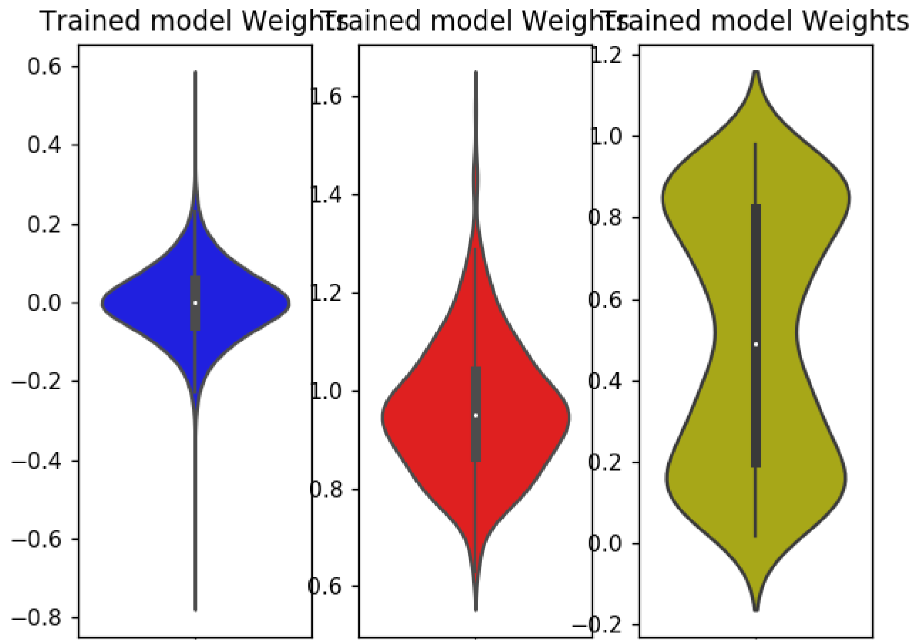
```
Test score: 0.1093290721397847
Test accuracy: 0.9679
```





In [0]:

```
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()
```



Hidden Layer 1 Hidden Layer 2 Output Layer

Hyper-parameter tuning of Keras models using Sklearn

Assignment

1. Architecture change with 2 hidden layers
2. Architecture change with 3 hidden layers
3. Architecture change with 5 hidden layers

Task 1

In [0]:

```
from keras.optimizers import Adam,RMSprop,SGD
def best_hyperparametersforTask1(activ):

    model = Sequential()
    model.add(Dense(456, activation=activ, input_shape=(input_dim,), kernel_initializer=RandomNorma
l(mean=0.0, stddev=0.062, seed=None)))
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(102, activation=activ, kernel_initializer=RandomNormal(mean=0.0, stddev=0.125,
seed=None)) )
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(output_dim, activation='softmax'))


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

    return model
```

Task 2

In [0]:

```
from keras.optimizers import Adam,RMSprop,SGD
def best_hyperparametersforTask2(activ):

    model = Sequential()
    model.add(Dense(484, activation=activ, input_shape=(input_dim,), kernel_initializer=RandomNorma
l(mean=0.0, stddev=0.062, seed=None)))
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(324, activation=activ, kernel_initializer=RandomNormal(mean=0.0, stddev=0.125,
seed=None)) )
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(128, activation=activ, kernel_initializer=RandomNormal(mean=0.0, stddev=0.125,
seed=None)) )
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(output_dim, activation='softmax'))


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

    return model
```

Task 3

In [0]:

```python
from keras.optimizers import Adam,RMSprop,SGD
def best_hyperparametersforTask3(activ):

    model = Sequential()
    model.add(Dense(512, activation=activ, input_shape=(input_dim,), kernel_initializer=RandomNorma
l(mean=0.0, stddev=0.062, seed=None)))
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(456, activation=activ, kernel_initializer=RandomNormal(mean=0.0, stddev=0.125,
seed=None)) )
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(348, activation=activ, kernel_initializer=RandomNormal(mean=0.0, stddev=0.125,
seed=None)) )
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(256, activation=activ, kernel_initializer=RandomNormal(mean=0.0, stddev=0.125,
seed=None)) )
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(124, activation=activ, kernel_initializer=RandomNormal(mean=0.0, stddev=0.125,
seed=None)) )
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(output_dim, activation='softmax'))


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

    return model
```

Example code

In [0]:

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

activ = ['sigmoid','relu']

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

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

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

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

In [0]:

```python
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: 0.975633 using {'activ': 'relu'}
0.974650 (0.001138) with: {'activ': 'sigmoid'}
0.975633 (0.002812) with: {'activ': 'relu'}
```

Task 1 execution

In [33]:

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


activ = ['sigmoid','relu']

from keras.layers.normalization import BatchNormalization
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV
from keras.layers import Dropout

model = KerasClassifier(build_fn=best_hyperparametersforTask1, epochs=nb_epoch, batch_size=batch_si
ze, 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)
```

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

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:3733: calling dropout (from
tensorflow.python.ops.nn_ops) with keep_prob is deprecated 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`.
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:4432: The name tf.random_uniform is deprecated. Pleas
e use tf.random.uniform instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimizers.py:793: The name t
f.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_backend.py:3576: The name tf.log is deprecated. Please use tf.ma
th.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 be 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_backend.py:1033: The name tf.assign_add is deprecated. Please us
e tf.compat.v1.assign_add instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.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_backend.py:3005: The name tf.Session is deprecated. Please use t
f.compat.v1.Session instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:190: The name tf.get_default_session is deprecated. P
lease use tf.compat.v1.get_default_session instead.

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

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:207: The name tf.global_variables is deprecated. Plea
se use tf.compat.v1.global_variables instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
```

```
packages/keras/backend/tensorflow_backend.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_backend.py:223: The name tf.variables_initializer is deprecated.
Please use tf.compat.v1.variables_initializer instead.
```

In [34]:

```python
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: 0.978167 using {'activ': 'relu'}
0.970483 (0.000768) with: {'activ': 'sigmoid'}
0.978167 (0.001281) with: {'activ': 'relu'}
```

In [36]:

```python
from keras.layers.normalization import BatchNormalization

model_task1 = Sequential()

model_task1.add(Dense(456, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.039, seed=None)))
model_task1.add(BatchNormalization())

model_task1.add(Dense(102, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.55
, seed=None)) )
model_task1.add(BatchNormalization())

model_task1.add(Dense(output_dim, activation='softmax'))


model_task1.summary()
```

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_26 (Dense)             (None, 456)               357960    
_________________________________________________________________
batch_normalization_17 (Batc (None, 456)               1824      
_________________________________________________________________
dense_27 (Dense)             (None, 102)               46614     
_________________________________________________________________
batch_normalization_18 (Batc (None, 102)               408       
_________________________________________________________________
dense_28 (Dense)             (None, 10)                1030      
=================================================================
Total params: 407,836
Trainable params: 406,720
Non-trainable params: 1,116
_________________________________________________________________
```

In [37]:

```python
model_task1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_task1.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, vali
dation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.1970 - acc: 0.9424 -
val_loss: 0.1066 - val_acc: 0.9681
Epoch 2/20
```

```
60000/60000 [==============================] - 5s 86us/step - loss: 0.0753 - acc: 0.9778 -
val_loss: 0.0932 - val_acc: 0.9711
Epoch 3/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.0477 - acc: 0.9859 -
val_loss: 0.0774 - val_acc: 0.9763
Epoch 4/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.0359 - acc: 0.9890 -
val_loss: 0.0763 - val_acc: 0.9763
Epoch 5/20
60000/60000 [==============================] - 5s 87us/step - loss: 0.0256 - acc: 0.9923 -
val_loss: 0.0717 - val_acc: 0.9779
Epoch 6/20
60000/60000 [==============================] - 5s 87us/step - loss: 0.0220 - acc: 0.9931 -
val_loss: 0.0808 - val_acc: 0.9747
Epoch 7/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.0188 - acc: 0.9941 -
val_loss: 0.0764 - val_acc: 0.9790
Epoch 8/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.0148 - acc: 0.9951 -
val_loss: 0.0696 - val_acc: 0.9802
Epoch 9/20
60000/60000 [==============================] - 5s 87us/step - loss: 0.0120 - acc: 0.9965 -
val_loss: 0.0767 - val_acc: 0.9784
Epoch 10/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.0146 - acc: 0.9950 -
val_loss: 0.0753 - val_acc: 0.9799
Epoch 11/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.0123 - acc: 0.9961 -
val_loss: 0.0770 - val_acc: 0.9792
Epoch 12/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.0105 - acc: 0.9968 -
val_loss: 0.0730 - val_acc: 0.9791
Epoch 13/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.0096 - acc: 0.9968 -
val_loss: 0.0763 - val_acc: 0.9799
Epoch 14/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.0093 - acc: 0.9970 -
val_loss: 0.0893 - val_acc: 0.9780
Epoch 15/20
60000/60000 [==============================] - 5s 87us/step - loss: 0.0093 - acc: 0.9968 -
val_loss: 0.0776 - val_acc: 0.9796
Epoch 16/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.0083 - acc: 0.9974 -
val_loss: 0.0764 - val_acc: 0.9814
Epoch 17/20
60000/60000 [==============================] - 5s 87us/step - loss: 0.0060 - acc: 0.9982 -
val_loss: 0.0909 - val_acc: 0.9795
Epoch 18/20
60000/60000 [==============================] - 5s 87us/step - loss: 0.0073 - acc: 0.9975 -
val_loss: 0.0904 - val_acc: 0.9800
Epoch 19/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.0077 - acc: 0.9973 -
val_loss: 0.0759 - val_acc: 0.9816
Epoch 20/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.0060 - acc: 0.9980 -
val_loss: 0.0806 - val_acc: 0.9804
```

In [77]:

```
import mpld3
from mpld3 import plugins

score_task1 = model_task1.evaluate(X_test, Y_test, verbose=0)

print('Test score:', score_task1[0])
print('Test accuracy:', score_task1[1])

ax_task1.set_xlabel('epoch') ; ax_task1.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x_task1 = list(range(1,nb_epoch+1))

vy_task1 = history.history['val_loss']
ty_task1 = history.history['loss']

print(vy_task1)
```

```
print(ty_task1)

plt_dynamic_plot(x_task1, vy_task1, ty_task1, ax_task1)
mpld3.display()
```

```
Test score: 0.08062241236004375
Test accuracy: 0.9804
[0.10663776563256978, 0.09317723634988069, 0.07740170887410641, 0.07626734635229222,
0.07168362509161234, 0.08082462112847716, 0.07638109823968262, 0.06958439976067748,
0.07670358981615864, 0.07533570332399103, 0.07699054275283124, 0.07295610949172406,
0.07633888032212853, 0.08933395910579711, 0.07756036461495677, 0.07637776467328658,
0.09086646485980746, 0.09044575633805507, 0.07594511643507867, 0.08062241120594554]
[0.19695996968746185, 0.07529902205864589, 0.0477438554495573, 0.035873308963576954,
0.025566030979156495, 0.022009523358444374, 0.018790008194496235, 0.014786293017243346,
0.012021565819283327, 0.014606639272719622, 0.012285229281087717, 0.010518471387246002,
0.009619418952443327, 0.009341397868438314, 0.009349660067384441, 0.008259907729923726,
0.006044347383423398, 0.007345158197854956, 0.0076754562556743625, 0.006027616090932861]
```

Out[77]:

Task 2 execution

In [79]:

```
activ = ['sigmoid','relu']

from keras.layers.normalization import BatchNormalization
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV
from keras.layers import Dropout

model2 = KerasClassifier(build_fn=best_hyperparametersforTask2, epochs=nb_epoch, batch_size=batch_s
ize, 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_task2 = GridSearchCV(estimator=model2, param_grid=param_grid)
grid_result2 = grid_task2.fit(X_train, Y_train)
```

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

In [80]:

```
print("Best: %f using %s" % (grid_result2.best_score_, grid_result2.best_params_))
means = grid_result2.cv_results_['mean_test_score']
stds = grid_result2.cv_results_['std_test_score']
params = grid_result2.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.977517 using {'activ': 'relu'}
0.970433 (0.001496) with: {'activ': 'sigmoid'}
0.977517 (0.000931) with: {'activ': 'relu'}
```

In [82]:

```
from keras.layers.normalization import BatchNormalization

model_task2 = Sequential()

model_task2.add(Dense(484, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.062, seed=None)))
model_task2.add(BatchNormalization())
```

```
model_task2.add(Dropout(0.5))
model_task2.add(Dense(324, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.12
5, seed=None)) )
model_task2.add(BatchNormalization())
model_task2.add(Dropout(0.5))
model_task2.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.12
5, seed=None)) )
model_task2.add(BatchNormalization())
model_task2.add(Dropout(0.5))
model_task2.add(Dense(output_dim, activation='softmax'))

model_task2.summary()
```

```
Model: "sequential_22"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_60 (Dense)             (None, 484)               379940    
_________________________________________________________________
batch_normalization_42 (Batc (None, 484)               1936      
_________________________________________________________________
dropout_38 (Dropout)         (None, 484)               0         
_________________________________________________________________
dense_61 (Dense)             (None, 324)               157140    
_________________________________________________________________
batch_normalization_43 (Batc (None, 324)               1296      
_________________________________________________________________
dropout_39 (Dropout)         (None, 324)               0         
_________________________________________________________________
dense_62 (Dense)             (None, 128)               41600     
_________________________________________________________________
batch_normalization_44 (Batc (None, 128)               512       
_________________________________________________________________
dropout_40 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_63 (Dense)             (None, 10)                1290      
=================================================================
Total params: 583,714
Trainable params: 581,842
Non-trainable params: 1,872
_________________________________________________________________
```

In [83]:

```
model_task2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history2 = model_task2.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, val
idation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 12s 202us/step - loss: 0.6564 - acc: 0.7996 - val_l
oss: 0.1881 - val_acc: 0.9407
Epoch 2/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.2767 - acc: 0.9178 -
val_loss: 0.1322 - val_acc: 0.9582
Epoch 3/20
60000/60000 [==============================] - 7s 117us/step - loss: 0.2130 - acc: 0.9364 -
val_loss: 0.1096 - val_acc: 0.9663
Epoch 4/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.1776 - acc: 0.9472 -
val_loss: 0.1004 - val_acc: 0.9694
Epoch 5/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.1581 - acc: 0.9528 -
val_loss: 0.0898 - val_acc: 0.9735
Epoch 6/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.1379 - acc: 0.9582 -
val_loss: 0.0820 - val_acc: 0.9749
Epoch 7/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.1308 - acc: 0.9608 -
val_loss: 0.0794 - val_acc: 0.9760
Epoch 8/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.1180 - acc: 0.9654 -
val_loss: 0.0776 - val_acc: 0.9758
Epoch 9/20
```

```
Epoch 9/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.1101 - acc: 0.9668 -
val_loss: 0.0803 - val_acc: 0.9776
Epoch 10/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.1000 - acc: 0.9702 -
val_loss: 0.0722 - val_acc: 0.9782
Epoch 11/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0989 - acc: 0.9698 -
val_loss: 0.0686 - val_acc: 0.9794
Epoch 12/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.0867 - acc: 0.9732 -
val_loss: 0.0682 - val_acc: 0.9806
Epoch 13/20
60000/60000 [==============================] - 7s 117us/step - loss: 0.0855 - acc: 0.9740 -
val_loss: 0.0670 - val_acc: 0.9804
Epoch 14/20
60000/60000 [==============================] - 7s 117us/step - loss: 0.0774 - acc: 0.9763 -
val_loss: 0.0649 - val_acc: 0.9803
Epoch 15/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.0781 - acc: 0.9760 -
val_loss: 0.0643 - val_acc: 0.9814
Epoch 16/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.0748 - acc: 0.9772 -
val_loss: 0.0682 - val_acc: 0.9818
Epoch 17/20
60000/60000 [==============================] - 7s 115us/step - loss: 0.0711 - acc: 0.9781 -
val_loss: 0.0662 - val_acc: 0.9799
Epoch 18/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.0720 - acc: 0.9773 -
val_loss: 0.0627 - val_acc: 0.9831
Epoch 19/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.0668 - acc: 0.9789 -
val_loss: 0.0583 - val_acc: 0.9827
Epoch 20/20
60000/60000 [==============================] - 7s 117us/step - loss: 0.0632 - acc: 0.9801 -
val_loss: 0.0645 - val_acc: 0.9825
```

In [84]:

```
score_task2 = model_task2.evaluate(X_test, Y_test, verbose=0)

print('Test score:', score_task2[0])
print('Test accuracy:', score_task2[1])

ax_task1.set_xlabel('epoch') ; ax_task1.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x_task1 = list(range(1,nb_epoch+1))

vy_task1 = history2.history['val_loss']
ty_task1 = history2.history['loss']

print(vy_task1)
print(ty_task1)

plt_dynamic_plot(x_task1, vy_task1, ty_task1, ax_task1)
mpld3.display()
```

```
Test score: 0.06452691478281632
Test accuracy: 0.9825
[0.18811061244010926, 0.1321807575829327, 0.1096077186241746, 0.10037248664498329,
0.08980758393295109, 0.08201991958636791, 0.0793870957940817, 0.07755979443769902,
0.08031285183485598, 0.07217185893226415, 0.06863237277567387, 0.0682252861815039,
0.06699350098753348, 0.06486856495265383, 0.06426630284576677, 0.0682348092280794,
0.06615413975138217, 0.06274196430703159, 0.05828299154554254, 0.06452691504568793]
[0.6564425819079082, 0.27673976113796234, 0.21299197249412535, 0.17755161137183506,
0.1581129201332728, 0.1378718542456627, 0.13079650530020395, 0.11802167086998622,
0.11007206436594327, 0.09997802500327428, 0.09885259782274564, 0.08671309248606364,
0.08551792087952297, 0.07736172651052475, 0.07810872188607852, 0.07475954596996308,
0.07110208432475726, 0.07195941407283148, 0.06680011056760947, 0.063217876030008]
```

Out[84]:

Task 3 Execution

In [86]:

```
activ = ['sigmoid','relu']

from keras.layers.normalization import BatchNormalization
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV
from keras.layers import Dropout

model3 = KerasClassifier(build_fn=best_hyperparametersforTask3, epochs=nb_epoch, batch_size=batch_s
ize, 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_task3 = GridSearchCV(estimator=model3, param_grid=param_grid)
grid_result3 = grid_task3.fit(X_train, Y_train)
```

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

In [87]:

```
print("Best: %f using %s" % (grid_result3.best_score_, grid_result3.best_params_))
means = grid_result3.cv_results_['mean_test_score']
stds = grid_result3.cv_results_['std_test_score']
params = grid_result3.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.976133 using {'activ': 'relu'}
0.965667 (0.001093) with: {'activ': 'sigmoid'}
0.976133 (0.001047) with: {'activ': 'relu'}
```

In [90]:

```
model_task3 = Sequential()

model_task3.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.062, seed=None)))
model_task3.add(BatchNormalization())
model_task3.add(Dropout(0.5))
model_task3.add(Dense(456, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.12
5, seed=None)) )
model_task3.add(BatchNormalization())
model_task3.add(Dropout(0.5))
model_task3.add(Dense(348, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.12
5, seed=None)) )
model_task3.add(BatchNormalization())
model_task3.add(Dropout(0.5))
model_task3.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.12
5, seed=None)) )
model_task3.add(BatchNormalization())
model_task3.add(Dropout(0.5))
model_task3.add(Dense(124, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.12
5, seed=None)) )
model_task3.add(BatchNormalization())
model_task3.add(Dropout(0.5))
model_task3.add(Dense(output_dim, activation='softmax'))

model_task3.summary()
```

```
Model: "sequential_38"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
```

```
dense_139 (Dense)            (None, 512)               401920
_________________________________________________________________
batch_normalization_106 (Bat (None, 512)               2048
_________________________________________________________________
dropout_102 (Dropout)        (None, 512)               0
_________________________________________________________________
dense_140 (Dense)            (None, 456)               233928
_________________________________________________________________
batch_normalization_107 (Bat (None, 456)               1824
_________________________________________________________________
dropout_103 (Dropout)        (None, 456)               0
_________________________________________________________________
dense_141 (Dense)            (None, 348)               159036
_________________________________________________________________
batch_normalization_108 (Bat (None, 348)               1392
_________________________________________________________________
dropout_104 (Dropout)        (None, 348)               0
_________________________________________________________________
dense_142 (Dense)            (None, 256)               89344
_________________________________________________________________
batch_normalization_109 (Bat (None, 256)               1024
_________________________________________________________________
dropout_105 (Dropout)        (None, 256)               0
_________________________________________________________________
dense_143 (Dense)            (None, 124)               31868
_________________________________________________________________
batch_normalization_110 (Bat (None, 124)               496
_________________________________________________________________
dropout_106 (Dropout)        (None, 124)               0
_________________________________________________________________
dense_144 (Dense)            (None, 10)                1250
=================================================================
Total params: 924,130
Trainable params: 920,738
Non-trainable params: 3,392
_________________________________________________________________
```

In [91]:

```
model_task3.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history3 = model_task3.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, val
idation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 23s 390us/step - loss: 1.1447 - acc: 0.6384 - val_l
oss: 0.2758 - val_acc: 0.9174
Epoch 2/20
60000/60000 [==============================] - 10s 170us/step - loss: 0.4032 - acc: 0.8816 - val_l
oss: 0.1789 - val_acc: 0.9472
Epoch 3/20
60000/60000 [==============================] - 10s 168us/step - loss: 0.2911 - acc: 0.9179 - val_l
oss: 0.1489 - val_acc: 0.9575
Epoch 4/20
60000/60000 [==============================] - 10s 171us/step - loss: 0.2391 - acc: 0.9324 - val_l
oss: 0.1174 - val_acc: 0.9652
Epoch 5/20
60000/60000 [==============================] - 10s 168us/step - loss: 0.2065 - acc: 0.9418 - val_l
oss: 0.1093 - val_acc: 0.9695
Epoch 6/20
60000/60000 [==============================] - 10s 170us/step - loss: 0.1834 - acc: 0.9487 - val_l
oss: 0.1016 - val_acc: 0.9715
Epoch 7/20
60000/60000 [==============================] - 10s 169us/step - loss: 0.1627 - acc: 0.9542 - val_l
oss: 0.0974 - val_acc: 0.9731
Epoch 8/20
60000/60000 [==============================] - 10s 168us/step - loss: 0.1483 - acc: 0.9581 - val_l
oss: 0.0838 - val_acc: 0.9758
Epoch 9/20
60000/60000 [==============================] - 10s 168us/step - loss: 0.1375 - acc: 0.9604 - val_l
oss: 0.0855 - val_acc: 0.9757
Epoch 10/20
60000/60000 [==============================] - 10s 169us/step - loss: 0.1313 - acc: 0.9628 - val_l
oss: 0.0840 - val_acc: 0.9772
```

```
Epoch 11/20
60000/60000 [==============================] - 10s 171us/step - loss: 0.1233 - acc: 0.9659 - val_l
oss: 0.0799 - val_acc: 0.9780
Epoch 12/20
60000/60000 [==============================] - 10s 171us/step - loss: 0.1184 - acc: 0.9660 - val_l
oss: 0.0814 - val_acc: 0.9772
Epoch 13/20
60000/60000 [==============================] - 10s 171us/step - loss: 0.1148 - acc: 0.9685 - val_l
oss: 0.0722 - val_acc: 0.9809
Epoch 14/20
60000/60000 [==============================] - 10s 170us/step - loss: 0.1076 - acc: 0.9695 - val_l
oss: 0.0735 - val_acc: 0.9803
Epoch 15/20
60000/60000 [==============================] - 10s 171us/step - loss: 0.1047 - acc: 0.9703 - val_l
oss: 0.0677 - val_acc: 0.9808
Epoch 16/20
60000/60000 [==============================] - 10s 172us/step - loss: 0.1014 - acc: 0.9719 - val_l
oss: 0.0690 - val_acc: 0.9817
Epoch 17/20
60000/60000 [==============================] - 10s 169us/step - loss: 0.0950 - acc: 0.9736 - val_l
oss: 0.0694 - val_acc: 0.9818
Epoch 18/20
60000/60000 [==============================] - 10s 170us/step - loss: 0.0910 - acc: 0.9749 - val_l
oss: 0.0644 - val_acc: 0.9836
Epoch 19/20
60000/60000 [==============================] - 10s 172us/step - loss: 0.0930 - acc: 0.9741 - val_l
oss: 0.0714 - val_acc: 0.9819
Epoch 20/20
60000/60000 [==============================] - 10s 169us/step - loss: 0.0826 - acc: 0.9762 - val_l
oss: 0.0692 - val_acc: 0.9825
```

In [92]:

```python
score_task3 = model_task3.evaluate(X_test, Y_test, verbose=0)

print('Test score:', score_task3[0])
print('Test accuracy:', score_task3[1])

ax_task1.set_xlabel('epoch') ; ax_task1.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x_task1 = list(range(1,nb_epoch+1))

vy_task1 = history3.history['val_loss']
ty_task1 = history3.history['loss']

print(vy_task1)
print(ty_task1)

plt_dynamic_plot(x_task1, vy_task1, ty_task1, ax_task1)
mpld3.display()
```

```
Test score: 0.06919898579865694
Test accuracy: 0.9825
[0.2757951403617859, 0.17889915275275708, 0.14887760931998492, 0.11736068408563734,
0.10926629482582212, 0.10164899545609951, 0.09743993790000677, 0.08383604726046323,
0.08553117757583968, 0.08398494211179204, 0.07987379637137056, 0.08139870572201907,
0.07221250504720957, 0.07348079082481564, 0.06771763956397772, 0.06896937229735776,
0.06935005094539375, 0.06440088580437005, 0.07141120507828891, 0.06919898539423011]
[1.144697232834498, 0.40316257503827413, 0.29111041218439737, 0.23914790218671164,
0.20650551250775656, 0.18335089112122854, 0.16265787526369094, 0.14834913578828177,
0.13748974556128185, 0.13125558004975318, 0.12330681715607643, 0.11839671297967434,
0.11480624519586563, 0.10761370351711909, 0.10465052956342698, 0.10139177598158519,
0.09499700264135996, 0.09102033909161886, 0.09296152342955272, 0.08258540489971637]
```

Out[92]:

Conclusion

In [93]:

```python
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Architecture", "Activation", "Optimizer", "Test score", "Test Accuracy"]


x.add_row(["2 Layer NN", "ReLU","Adam", 0.0806, 0.9804 ])
x.add_row(["3 Layer NN", "ReLU","Adam", 0.0645, 0.9825])
x.add_row(["5 Layer NN", "ReLU","Adam", 0.0691, 0.9825])

print(x)
```

```
+--------------+------------+-----------+------------+---------------+
| Architecture | Activation | Optimizer | Test score | Test Accuracy |
+--------------+------------+-----------+------------+---------------+
|  2 Layer NN  |    ReLU    |    Adam   |   0.0806   |     0.9804    |
|  3 Layer NN  |    ReLU    |    Adam   |   0.0645   |     0.9825    |
|  5 Layer NN  |    ReLU    |    Adam   |   0.0691   |     0.9825    |
+--------------+------------+-----------+------------+---------------+
```