1. Keras -- MLPs on MNIST Assignment

1.1 Importing required Libraries

```
# if you keras is not using tensorflow as backend set "KERAS_BACKEND=tensorflow" u
import tensorflow as tf
from keras.utils import np_utils
from keras.datasets import mnist
import seaborn as sns
from keras.initializers import RandomNormal
from keras.layers.normalization import BatchNormalization
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout
```

The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you upgrade now or ensure your notebook will continue to use TensorFlow 1.x via the `%t` `1.x` magic: more info.

```
Using TensorFlow backend.
```

1.2 Function to plot a dynamic plot

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

1.3 High level overview of data set

```
# 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 [==============================] - 2s 0us/step
```

```
print("Number of training examples :", X_train.shape[0], "and each image is of sha
```

```
print("Number of training examples :", X_test.shape[0], "and each image is of shap
```

```
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)
```

```
# 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])
```

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

print("Number of training examples :", X_train.shape[0], "and each image is of sha
print("Number of training examples :", X_test.shape[0], "and each image is of shap
```

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

```
# 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]
```

1.4 Normalizing the train and test sets

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

X_train = X_train/255
X_test = X_test/255


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

```
0.58823529 0.10588235 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.0627451  0.36470588 0.98823529 0.99215686 0.73333333
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.97647059 0.99215686 0.97647059 0.25098039 0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.18039216 0.50980392 0.71764706 0.99215686
0.99215686 0.81176471 0.00784314 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.15294118 0.58039216
0.89803922 0.99215686 0.99215686 0.99215686 0.98039216 0.71372549
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.09411765 0.44705882 0.86666667 0.99215686 0.99215686 0.99215686
0.99215686 0.78823529 0.30588235 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.09019608 0.25882353 0.83529412 0.99215686
0.99215686 0.99215686 0.99215686 0.77647059 0.31764706 0.00784314
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.07058824 0.67058824
0.85882353 0.99215686 0.99215686 0.99215686 0.99215686 0.76470588
0.31372549 0.03529412 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.21568627 0.6745098  0.88627451 0.99215686 0.99215686 0.99215686
0.99215686 0.95686275 0.52156863 0.04313725 0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.53333333 0.99215686
0.99215686 0.99215686 0.83137255 0.52941176 0.51764706 0.0627451
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.        ]
```

1.5 One-Hot Encoding the class label

```
# 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.]
```

1.6 A simple 2 layer model with Softmax classifier

```
# 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 co

# 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='gl
# bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=None, activi
# kernel_constraint=None, bias_constraint=None)

# Dense implements the operation: output = activation(dot(input, kernel) + bias) w
# activation is the element-wise activation function passed as the activation argu
# 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 activ

# 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 tensorflow.python.keras.layers import Dense
from tensorflow.python.keras.layers import BatchNormalization
from tensorflow.python.keras import Sequential



# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size =  256
nb_epoch = 20


# start building a model
model = Sequential()

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

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

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

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_cor
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
```

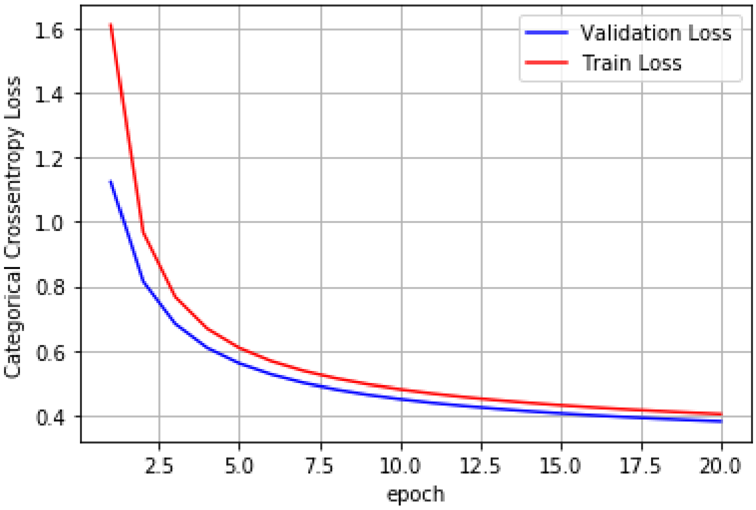
```
# Before training a model, you need to configure the learning process, which is do

# It receives three arguments:
# An optimizer. This could be the string identifier of an existing optimizer , htt
# A loss function. This is the objective that the model will try to minimize., htt
# A list of metrics. For any classification problem you will want to set this to m


# Note: when using the categorical_crossentropy loss, your targets should be in ca
# (e.g. if you have 10 classes, the target for each sample should be a 10-dimensio
# 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_data=None, shuffle=True, class_weight=None, sample_weight=None, initi
# validation_steps=None)

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

# it returns A History object. Its History.history attribute is a record of traini
# metrics values at successive epochs, as well as validation loss values and valid

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

history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verb
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 2s 39us/sample - loss: 1.6113
Epoch 2/20
60000/60000 [==============================] - 1s 13us/sample - loss: 0.9672
Epoch 3/20
60000/60000 [==============================] - 1s 13us/sample - loss: 0.7665
Epoch 4/20
60000/60000 [==============================] - 1s 13us/sample - loss: 0.6677
Epoch 5/20
60000/60000 [==============================] - 1s 13us/sample - loss: 0.6077
Epoch 6/20
60000/60000 [==============================] - 1s 14us/sample - loss: 0.5667
Epoch 7/20
60000/60000 [==============================] - 1s 14us/sample - loss: 0.5367
Epoch 8/20
60000/60000 [==============================] - 1s 13us/sample - loss: 0.5137
Epoch 9/20
60000/60000 [==============================] - 1s 13us/sample - loss: 0.4952
Epoch 10/20
60000/60000 [==============================] - 1s 13us/sample - loss: 0.4800
Epoch 11/20
60000/60000 [==============================] - 1s 14us/sample - loss: 0.4672
Epoch 12/20
60000/60000 [==============================] - 1s 14us/sample - loss: 0.4562
Epoch 13/20
60000/60000 [==============================] - 1s 13us/sample - loss: 0.4468
Epoch 14/20
60000/60000 [==============================] - 1s 13us/sample - loss: 0.4384
Epoch 15/20
60000/60000 [==============================] - 1s 13us/sample - loss: 0.4310
Epoch 16/20
60000/60000 [==============================] - 1s 13us/sample - loss: 0.4244
Epoch 17/20
60000/60000 [==============================] - 1s 13us/sample - loss: 0.4184
Epoch 18/20
60000/60000 [==============================] - 1s 13us/sample - loss: 0.4130
Epoch 19/20
60000/60000 [==============================] - 1s 13us/sample - loss: 0.4080
Epoch 20/20
60000/60000 [==============================] - 1s 13us/sample - loss: 0.4034
```

```
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_epoc

# 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

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

```
Test score: 0.38070512788295746
Test accuracy: 0.9
```



2 Model A : 2 Hidden Layes.

2.1 MLP + ReLU activation + Adam Optimizer

```
# Multilayer perceptron

model_relu = Sequential()
model_relu.add(Dense(256, activation='relu', input_shape=(input_dim,)))
model_relu.add(Dense(128, activation='relu'))
model_relu.add(Dense(output_dim, activation='softmax'))

model_relu.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 256)               200960    
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896     
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290      
=================================================================
Total params: 235,146
Trainable params: 235,146
Non-trainable params: 0
_________________________________________________________________
```

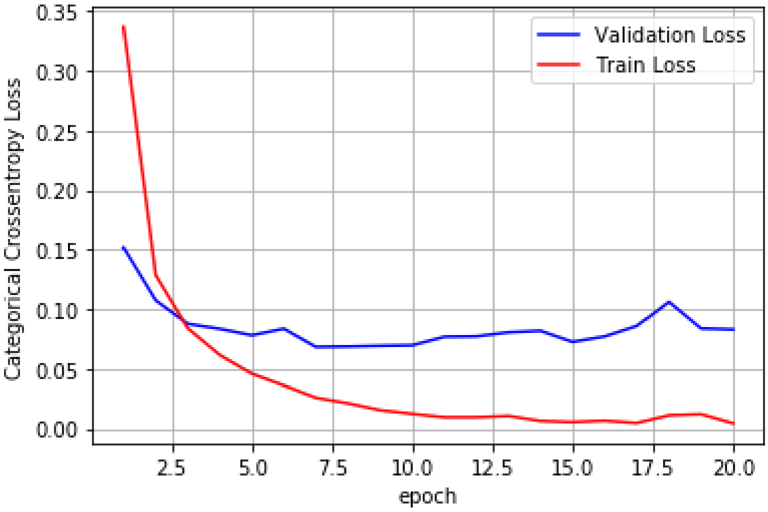
```
model_relu.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['ac

history = model_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 1s 17us/sample - loss: 0.3368
Epoch 2/20
60000/60000 [==============================] - 1s 15us/sample - loss: 0.1285
Epoch 3/20
60000/60000 [==============================] - 1s 16us/sample - loss: 0.0840
Epoch 4/20
60000/60000 [==============================] - 1s 16us/sample - loss: 0.0615
Epoch 5/20
60000/60000 [==============================] - 1s 15us/sample - loss: 0.0459
Epoch 6/20
60000/60000 [==============================] - 1s 15us/sample - loss: 0.0360
Epoch 7/20
60000/60000 [==============================] - 1s 15us/sample - loss: 0.0257
Epoch 8/20
60000/60000 [==============================] - 1s 15us/sample - loss: 0.0211
Epoch 9/20
60000/60000 [==============================] - 1s 15us/sample - loss: 0.0154
Epoch 10/20
60000/60000 [==============================] - 1s 16us/sample - loss: 0.0123
Epoch 11/20
60000/60000 [==============================] - 1s 15us/sample - loss: 0.0095
Epoch 12/20
60000/60000 [==============================] - 1s 16us/sample - loss: 0.0095
Epoch 13/20
60000/60000 [==============================] - 1s 15us/sample - loss: 0.0106
Epoch 14/20
60000/60000 [==============================] - 1s 16us/sample - loss: 0.0064
Epoch 15/20
60000/60000 [==============================] - 1s 15us/sample - loss: 0.0054
Epoch 16/20
60000/60000 [==============================] - 1s 16us/sample - loss: 0.0066
Epoch 17/20
60000/60000 [==============================] - 1s 15us/sample - loss: 0.0046
Epoch 18/20
60000/60000 [==============================] - 1s 15us/sample - loss: 0.0111
Epoch 19/20
60000/60000 [==============================] - 1s 16us/sample - loss: 0.0120
Epoch 20/20
60000/60000 [==============================] - 1s 16us/sample - loss: 0.0044
```

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

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

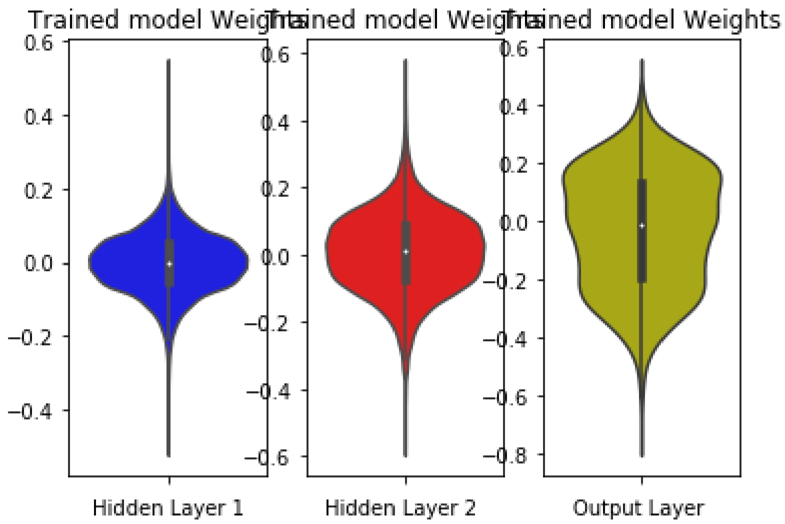
```
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

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

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

# 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

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

```
Test score: 0.08297917389905415
Test accuracy: 0.9815
```




```
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()
```



2.2 MLP + ReLU activation + Adam Optimizer (Batch Normalization)

```
model_relu = Sequential()
model_relu.add(Dense(256, activation='relu', input_shape=(input_dim,), kernel_init
model_relu.add(BatchNormalization())

model_relu.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=
model_relu.add(BatchNormalization())

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


model_relu.summary()
```

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

Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 256)               200960
_________________________________________________________________
batch_normalization (BatchNo (None, 256)               1024
```

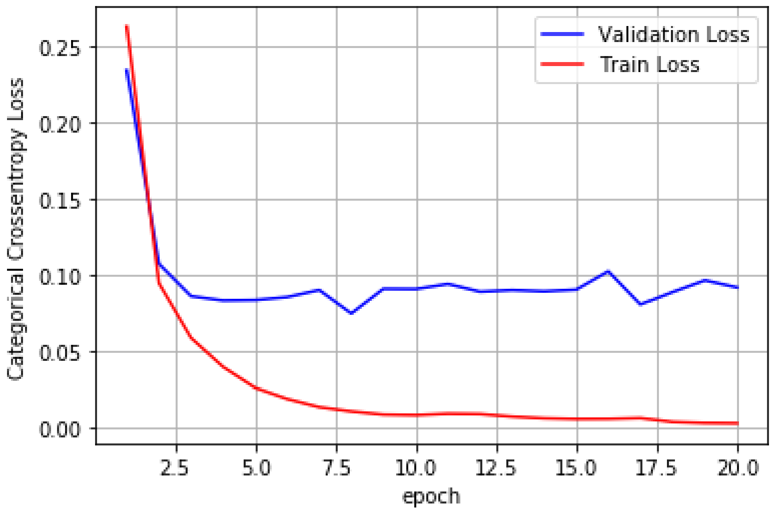
```
pile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

l_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, va
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 2s 27us/sample - loss: 0.2629
Epoch 2/20
60000/60000 [==============================] - 1s 24us/sample - loss: 0.0949
Epoch 3/20
60000/60000 [==============================] - 1s 24us/sample - loss: 0.0590
Epoch 4/20
60000/60000 [==============================] - 1s 24us/sample - loss: 0.0401
Epoch 5/20
60000/60000 [==============================] - 1s 24us/sample - loss: 0.0264
Epoch 6/20
60000/60000 [==============================] - 1s 23us/sample - loss: 0.0192
Epoch 7/20
60000/60000 [==============================] - 1s 23us/sample - loss: 0.0139
Epoch 8/20
60000/60000 [==============================] - 1s 23us/sample - loss: 0.0111
Epoch 9/20
60000/60000 [==============================] - 1s 24us/sample - loss: 0.0090
Epoch 10/20
60000/60000 [==============================] - 1s 23us/sample - loss: 0.0087
Epoch 11/20
60000/60000 [==============================] - 1s 23us/sample - loss: 0.0096
Epoch 12/20
60000/60000 [==============================] - 1s 23us/sample - loss: 0.0094
Epoch 13/20
60000/60000 [==============================] - 1s 23us/sample - loss: 0.0077
Epoch 14/20
60000/60000 [==============================] - 1s 24us/sample - loss: 0.0066
Epoch 15/20
60000/60000 [==============================] - 1s 23us/sample - loss: 0.0062
Epoch 16/20
60000/60000 [==============================] - 1s 23us/sample - loss: 0.0062
Epoch 17/20
60000/60000 [==============================] - 1s 23us/sample - loss: 0.0067
Epoch 18/20
60000/60000 [==============================] - 1s 23us/sample - loss: 0.0042
Epoch 19/20
60000/60000 [==============================] - 1s 24us/sample - loss: 0.0036
Epoch 20/20
60000/60000 [==============================] - 1s 23us/sample - loss: 0.0034
```

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

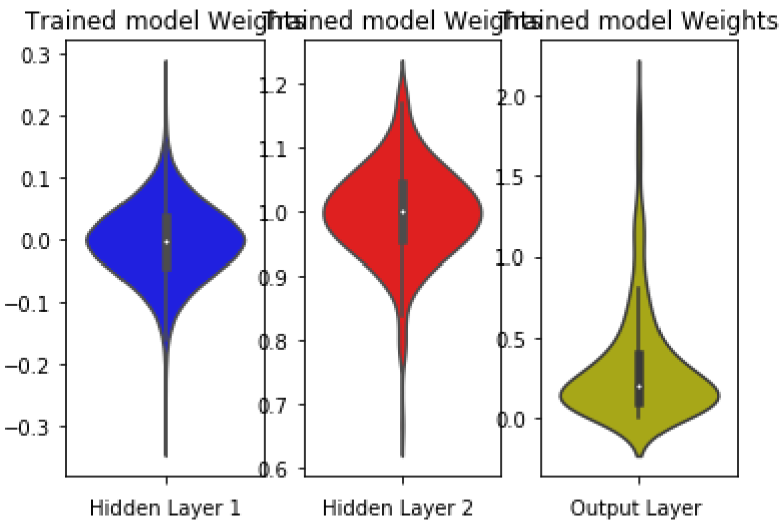
```
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# 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_epoc

# 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

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

```
Test score: 0.0921328787419421
Test accuracy: 0.9782
```




```
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()
```



2.3 MLP + ReLU activation + Adam Optimizer (Dropout)

```
from tensorflow.keras.layers import Dense, Dropout
model_relu = Sequential()
model_relu.add(Dense(256, activation='relu', input_shape=(input_dim,)))
model_relu.add(Dropout(0.5))
model_relu.add(Dense(128, activation='relu'))
model_relu.add(Dropout(0.5))
model_relu.add(Dense(output_dim, activation='softmax'))
```

```
model_relu.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

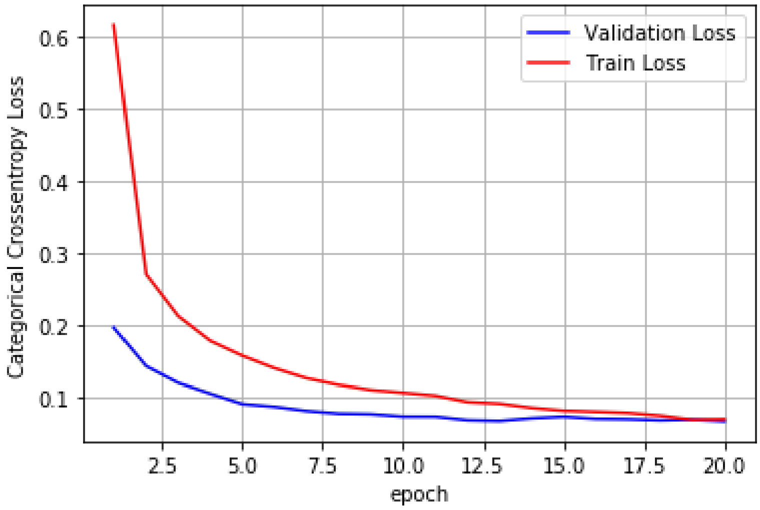
```
model_relu.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['ac

history = model_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,
```

```
validate on 10000 samples
===================] - 1s 16us/sample - loss: 0.6157 - acc: 0.8078 - val_loss:
===================] - 1s 14us/sample - loss: 0.2708 - acc: 0.9227 - val_loss:
===================] - 1s 14us/sample - loss: 0.2128 - acc: 0.9383 - val_loss:
===================] - 1s 14us/sample - loss: 0.1786 - acc: 0.9478 - val_loss:
===================] - 1s 14us/sample - loss: 0.1581 - acc: 0.9539 - val_loss:
===================] - 1s 14us/sample - loss: 0.1410 - acc: 0.9586 - val_loss:
===================] - 1s 14us/sample - loss: 0.1271 - acc: 0.9635 - val_loss:
===================] - 1s 15us/sample - loss: 0.1178 - acc: 0.9646 - val_loss:
===================] - 1s 15us/sample - loss: 0.1107 - acc: 0.9669 - val_loss:
===================] - 1s 14us/sample - loss: 0.1069 - acc: 0.9681 - val_loss:
===================] - 1s 14us/sample - loss: 0.1029 - acc: 0.9698 - val_loss:
===================] - 1s 14us/sample - loss: 0.0941 - acc: 0.9719 - val_loss:
===================] - 1s 15us/sample - loss: 0.0919 - acc: 0.9721 - val_loss:
===================] - 1s 15us/sample - loss: 0.0861 - acc: 0.9743 - val_loss:
===================] - 1s 14us/sample - loss: 0.0822 - acc: 0.9754 - val_loss:
===================] - 1s 15us/sample - loss: 0.0807 - acc: 0.9754 - val_loss:
===================] - 1s 14us/sample - loss: 0.0792 - acc: 0.9753 - val_loss:
===================] - 1s 14us/sample - loss: 0.0754 - acc: 0.9769 - val_loss:
===================] - 1s 14us/sample - loss: 0.0697 - acc: 0.9780 - val_loss:
===================] - 1s 14us/sample - loss: 0.0705 - acc: 0.9783 - val_loss:
```

```
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)
```

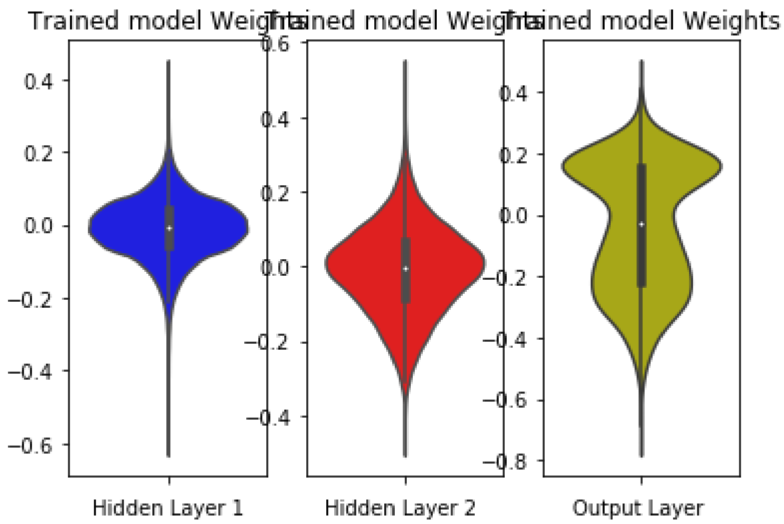
```
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# 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_epoc

# 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

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

```
Test score: 0.06762538701674711
Test accuracy: 0.9807
```




```
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()
```



2.4 MLP + ReLU activation + Adam Optimizer (Batch Normalization

```
# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormalizat
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.layers import BatchNormalization

model_relu = Sequential()
model_relu.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_init
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(output_dim, activation='softmax'))

model_relu.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_25 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_12 (Batc (None, 512)               2048
_________________________________________________________________
dropout_8 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_26 (Dense)             (None, 256)               131328
_________________________________________________________________
batch_normalization_13 (Batc (None, 256)               1024
_________________________________________________________________
dropout_9 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_27 (Dense)             (None, 10)                2570
=================================================================
Total params: 538,890
Trainable params: 537,354
Non-trainable params: 1,536
_________________________________________________________________
```

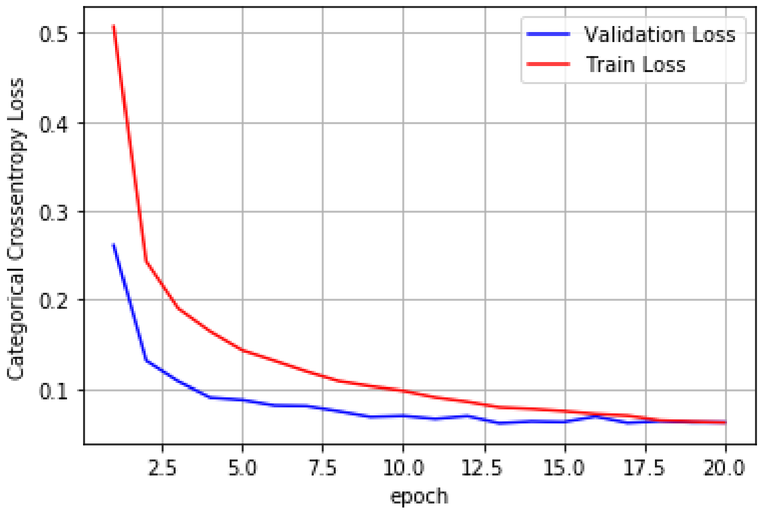
```
model_relu.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['ac

history = model_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 2s 29us/sample - loss: 0.5073
Epoch 2/20
60000/60000 [==============================] - 1s 24us/sample - loss: 0.2434
Epoch 3/20
60000/60000 [==============================] - 1s 22us/sample - loss: 0.1902
Epoch 4/20
60000/60000 [==============================] - 1s 22us/sample - loss: 0.1643
Epoch 5/20
60000/60000 [==============================] - 1s 23us/sample - loss: 0.1431
Epoch 6/20
60000/60000 [==============================] - 1s 23us/sample - loss: 0.1314
Epoch 7/20
60000/60000 [==============================] - 1s 22us/sample - loss: 0.1193
Epoch 8/20
60000/60000 [==============================] - 1s 23us/sample - loss: 0.1085
Epoch 9/20
60000/60000 [==============================] - 1s 22us/sample - loss: 0.1028
Epoch 10/20
60000/60000 [==============================] - 1s 23us/sample - loss: 0.0978
Epoch 11/20
60000/60000 [==============================] - 2s 26us/sample - loss: 0.0905
Epoch 12/20
60000/60000 [==============================] - 1s 22us/sample - loss: 0.0856
Epoch 13/20
60000/60000 [==============================] - 1s 23us/sample - loss: 0.0793
Epoch 14/20
60000/60000 [==============================] - 1s 22us/sample - loss: 0.0776
Epoch 15/20
60000/60000 [==============================] - 1s 23us/sample - loss: 0.0751
Epoch 16/20
60000/60000 [==============================] - 2s 25us/sample - loss: 0.0719
Epoch 17/20
60000/60000 [==============================] - 1s 24us/sample - loss: 0.0700
Epoch 18/20
60000/60000 [==============================] - 1s 25us/sample - loss: 0.0646
Epoch 19/20
60000/60000 [==============================] - 1s 21us/sample - loss: 0.0633
Epoch 20/20
60000/60000 [==============================] - 1s 22us/sample - loss: 0.0623
```

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

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

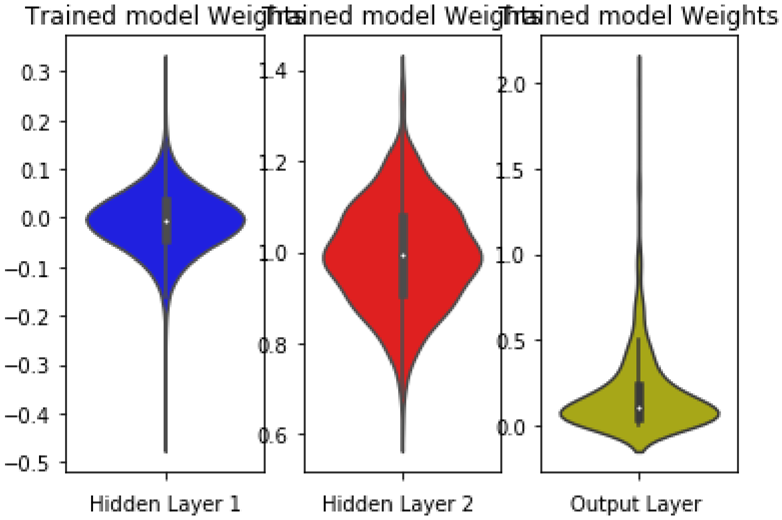
```
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

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

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

# 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

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

```
Test score: 0.06247625293707533
Test accuracy: 0.9828
```




```
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()
```



3 Model B : 3 Hidden Layers

3.1 MLP + ReLU activation + Adam Optimizer

```
# Multilayer perceptron

model_relu = Sequential()
model_relu.add(Dense(512, activation='relu', input_shape=(input_dim,)))
model_relu.add(Dense(256, activation='relu'))
model_relu.add(Dense(128, activation='relu'))
model_relu.add(Dense(output_dim, activation='softmax'))

model_relu.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_10 (Dense)             (None, 512)               401920
```

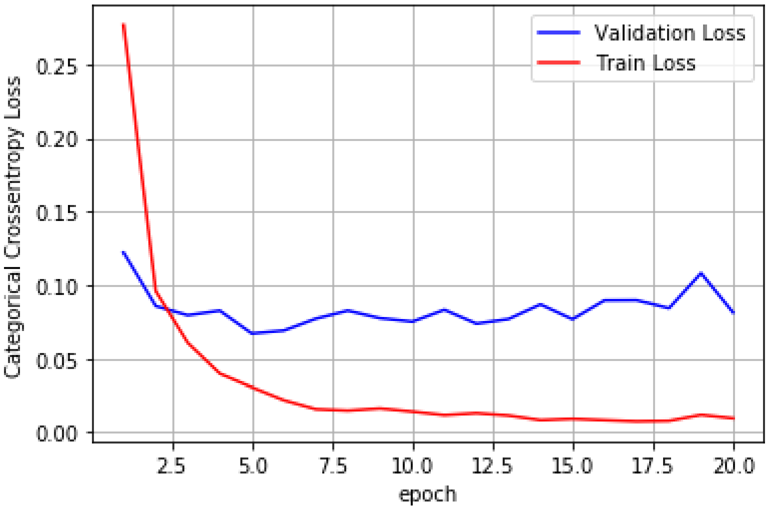
```
model_relu.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['ac

history = model_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,
```

```
validate on 10000 samples
===================] - 1s 20us/sample - loss: 0.2768 - acc: 0.9198 - val_loss:
===================] - 1s 18us/sample - loss: 0.0959 - acc: 0.9713 - val_loss:
===================] - 1s 17us/sample - loss: 0.0607 - acc: 0.9811 - val_loss:
===================] - 1s 18us/sample - loss: 0.0398 - acc: 0.9880 - val_loss:
===================] - 1s 17us/sample - loss: 0.0306 - acc: 0.9901 - val_loss:
===================] - 1s 18us/sample - loss: 0.0218 - acc: 0.9931 - val_loss:
===================] - 1s 18us/sample - loss: 0.0158 - acc: 0.9949 - val_loss:
===================] - 1s 18us/sample - loss: 0.0148 - acc: 0.9952 - val_loss:
===================] - 1s 18us/sample - loss: 0.0163 - acc: 0.9944 - val_loss:
===================] - 1s 18us/sample - loss: 0.0141 - acc: 0.9954 - val_loss:
===================] - 1s 18us/sample - loss: 0.0118 - acc: 0.9959 - val_loss:
===================] - 1s 18us/sample - loss: 0.0131 - acc: 0.9953 - val_loss:
===================] - 1s 18us/sample - loss: 0.0115 - acc: 0.9962 - val_loss:
===================] - 1s 18us/sample - loss: 0.0084 - acc: 0.9975 - val_loss:
===================] - 1s 18us/sample - loss: 0.0092 - acc: 0.9969 - val_loss:
===================] - 1s 18us/sample - loss: 0.0083 - acc: 0.9972 - val_loss:
===================] - 1s 18us/sample - loss: 0.0075 - acc: 0.9973 - val_loss:
===================] - 1s 18us/sample - loss: 0.0079 - acc: 0.9972 - val_loss:
===================] - 1s 18us/sample - loss: 0.0119 - acc: 0.9962 - val_loss:
===================] - 1s 18us/sample - loss: 0.0096 - acc: 0.9965 - val_loss:
```

```
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)
```

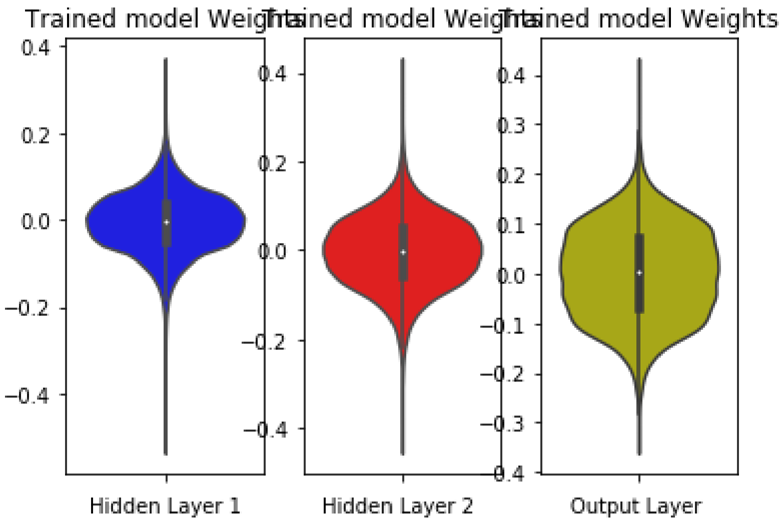
```
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# 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_epoc

# 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

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

```
Test score: 0.08133461526854412
Test accuracy: 0.9824
```




```
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()
```



3.2 MLP + ReLU activation + Adam Optimizer (Batch Normalization)

```
model_relu = Sequential()
model_relu.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_init
model_relu.add(BatchNormalization())

model_relu.add(Dense(256, activation='relu', input_shape=(input_dim,), kernel_init
model_relu.add(BatchNormalization())

model_relu.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=
model_relu.add(BatchNormalization())

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


model_relu.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_14 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_2 (Batch (None, 512)               2048
_________________________________________________________________
dense_15 (Dense)             (None, 256)               131328
_________________________________________________________________
batch_normalization_3 (Batch (None, 256)               1024
_________________________________________________________________
dense_16 (Dense)             (None, 128)               32896
_________________________________________________________________
batch_normalization_4 (Batch (None, 128)               512
_________________________________________________________________
dense_17 (Dense)             (None, 10)                1290
=================================================================
Total params: 571,018
Trainable params: 569,226
Non-trainable params: 1,792
_________________________________________________________________
```

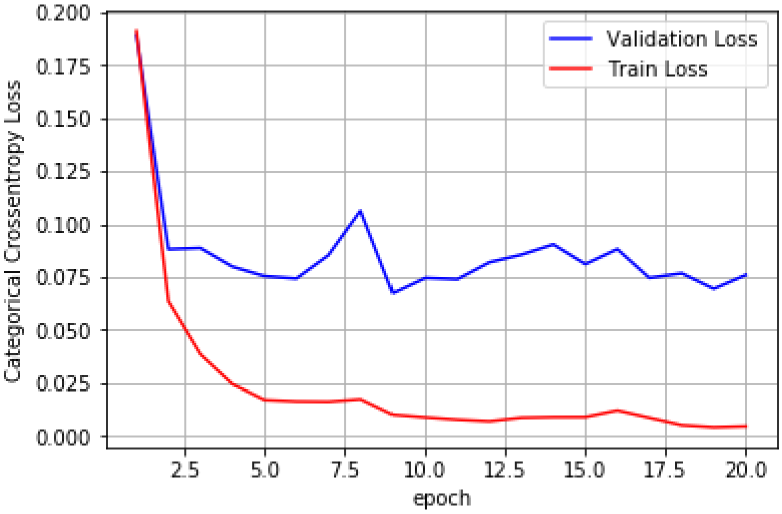
```
model_relu.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['ac

history = model_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 2s 33us/sample - loss: 0.1911
Epoch 2/20
60000/60000 [==============================] - 2s 29us/sample - loss: 0.0636
Epoch 3/20
60000/60000 [==============================] - 2s 27us/sample - loss: 0.0386
Epoch 4/20
60000/60000 [==============================] - 2s 28us/sample - loss: 0.0245
Epoch 5/20
60000/60000 [==============================] - 2s 28us/sample - loss: 0.0170
Epoch 6/20
60000/60000 [==============================] - 2s 27us/sample - loss: 0.0163
Epoch 7/20
60000/60000 [==============================] - 2s 28us/sample - loss: 0.0162
Epoch 8/20
60000/60000 [==============================] - 2s 27us/sample - loss: 0.0174
Epoch 9/20
60000/60000 [==============================] - 2s 28us/sample - loss: 0.0101
Epoch 10/20
60000/60000 [==============================] - 2s 28us/sample - loss: 0.0088
Epoch 11/20
60000/60000 [==============================] - 2s 28us/sample - loss: 0.0078
Epoch 12/20
60000/60000 [==============================] - 2s 29us/sample - loss: 0.0070
Epoch 13/20
60000/60000 [==============================] - 2s 28us/sample - loss: 0.0087
Epoch 14/20
60000/60000 [==============================] - 2s 29us/sample - loss: 0.0090
Epoch 15/20
60000/60000 [==============================] - 2s 29us/sample - loss: 0.0090
Epoch 16/20
60000/60000 [==============================] - 2s 29us/sample - loss: 0.0121
Epoch 17/20
60000/60000 [==============================] - 2s 26us/sample - loss: 0.0086
Epoch 18/20
60000/60000 [==============================] - 2s 27us/sample - loss: 0.0051
Epoch 19/20
60000/60000 [==============================] - 2s 26us/sample - loss: 0.0042
Epoch 20/20
60000/60000 [==============================] - 2s 29us/sample - loss: 0.0046
```

```
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)
```

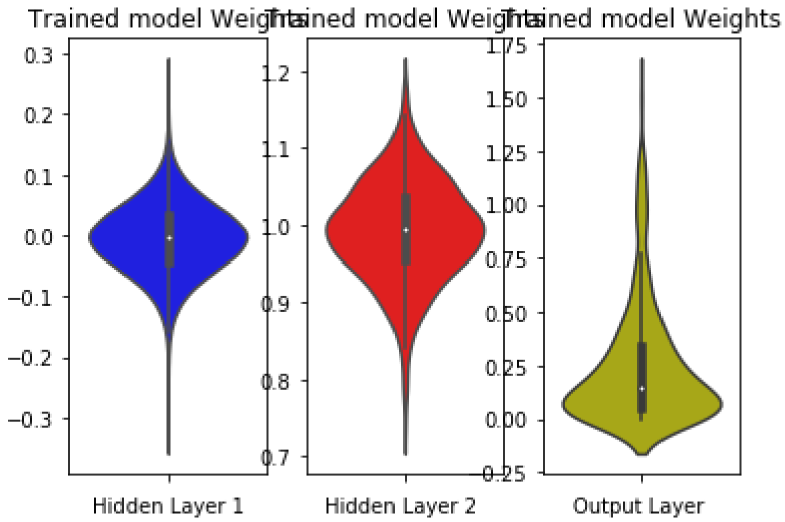
```
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# 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_epoc

# 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

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

```
Test score: 0.07600605861382428
Test accuracy: 0.9801
```




```
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()
```



3.3 MLP + ReLU activation + Adam Optimizer (Dropout)

```
from tensorflow.keras.layers import Dense, Dropout
model_relu = Sequential()
model_relu.add(Dense(512, activation='relu', input_shape=(input_dim,)))
model_relu.add(Dropout(0.5))
model_relu.add(Dense(256, activation='relu', input_shape=(input_dim,)))
model_relu.add(Dropout(0.5))
model_relu.add(Dense(128, activation='relu'))
model_relu.add(Dropout(0.5))
model_relu.add(Dense(output_dim, activation='softmax'))


model_relu.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_22 (Dense)             (None, 512)               401920
_________________________________________________________________
dropout_5 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_23 (Dense)             (None, 256)               131328
```

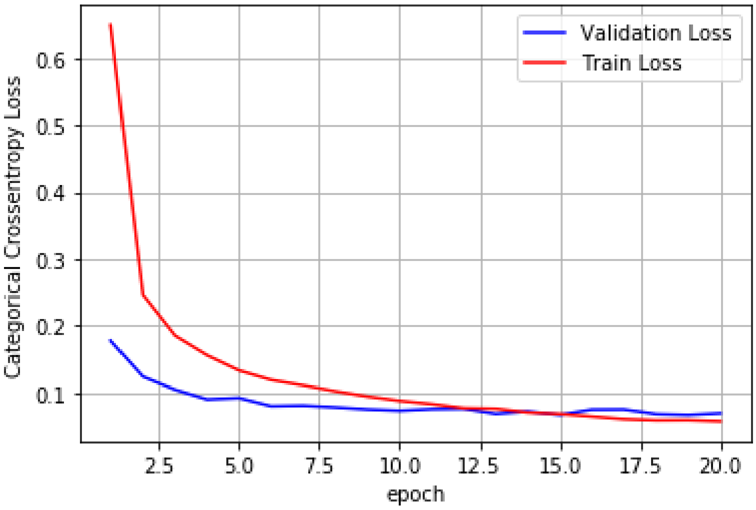
```
adam', loss='categorical_crossentropy', metrics=['accuracy'])

in, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_
```

```
validate on 10000 samples
===================] - 1s 25us/sample - loss: 0.6499 - acc: 0.7962 - val_loss:
===================] - 1s 19us/sample - loss: 0.2462 - acc: 0.9301 - val_loss:
===================] - 1s 19us/sample - loss: 0.1854 - acc: 0.9484 - val_loss:
===================] - 1s 19us/sample - loss: 0.1562 - acc: 0.9573 - val_loss:
===================] - 1s 19us/sample - loss: 0.1335 - acc: 0.9629 - val_loss:
===================] - 1s 19us/sample - loss: 0.1195 - acc: 0.9667 - val_loss:
===================] - 1s 20us/sample - loss: 0.1113 - acc: 0.9682 - val_loss:
===================] - 1s 20us/sample - loss: 0.1023 - acc: 0.9716 - val_loss:
===================] - 1s 20us/sample - loss: 0.0945 - acc: 0.9729 - val_loss:
===================] - 1s 20us/sample - loss: 0.0880 - acc: 0.9739 - val_loss:
===================] - 1s 19us/sample - loss: 0.0833 - acc: 0.9755 - val_loss:
===================] - 1s 19us/sample - loss: 0.0772 - acc: 0.9777 - val_loss:
===================] - 1s 19us/sample - loss: 0.0764 - acc: 0.9781 - val_loss:
===================] - 1s 19us/sample - loss: 0.0708 - acc: 0.9792 - val_loss:
===================] - 1s 19us/sample - loss: 0.0686 - acc: 0.9798 - val_loss:
===================] - 1s 19us/sample - loss: 0.0648 - acc: 0.9811 - val_loss:
===================] - 1s 19us/sample - loss: 0.0609 - acc: 0.9823 - val_loss:
===================] - 1s 19us/sample - loss: 0.0591 - acc: 0.9825 - val_loss:
===================] - 1s 19us/sample - loss: 0.0594 - acc: 0.9825 - val_loss:
===================] - 1s 19us/sample - loss: 0.0577 - acc: 0.9833 - val_loss:
```

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

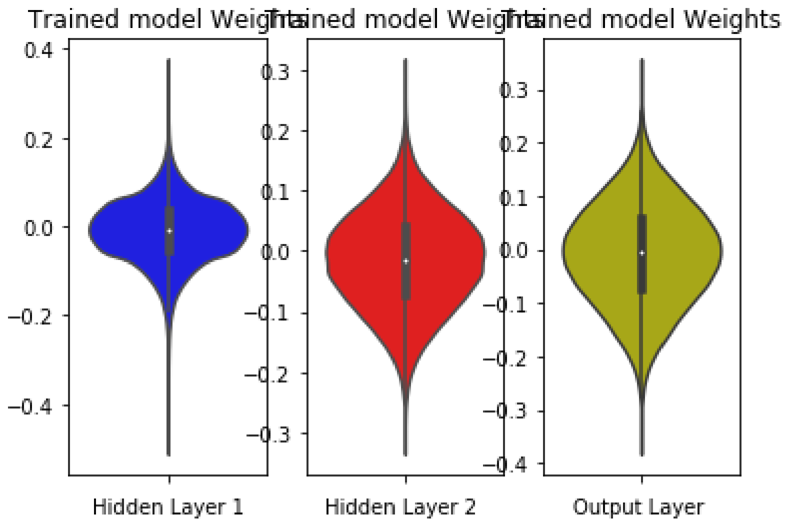
```
print('Test accuracy:', score[1])

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_epoc

# 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

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

```
Test score: 0.06959878407377218
Test accuracy: 0.9819
```




```
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()
```



3.4 MLP + ReLU activation + Adam Optimizer (Batch Normalization

```
# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormalizat
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.layers import BatchNormalization

model_relu = Sequential()
model_relu.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_init
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(256, activation='relu', input_shape=(input_dim,), kernel_init
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(output_dim, activation='softmax'))

model_relu.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_26 (Dense)             (None, 512)               401920    
_________________________________________________________________
batch_normalization_5 (Batch (None, 512)               2048      
_________________________________________________________________
dropout_8 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_27 (Dense)             (None, 256)               131328    
_________________________________________________________________
batch_normalization_6 (Batch (None, 256)               1024      
_________________________________________________________________
dropout_9 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_28 (Dense)             (None, 128)               32896     
_________________________________________________________________
batch_normalization_7 (Batch (None, 128)               512       
_________________________________________________________________
dropout_10 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_29 (Dense)             (None, 10)                1290      
=================================================================
Total params: 571,018
Trainable params: 569,226
Non-trainable params: 1,792
_________________________________________________________________
```

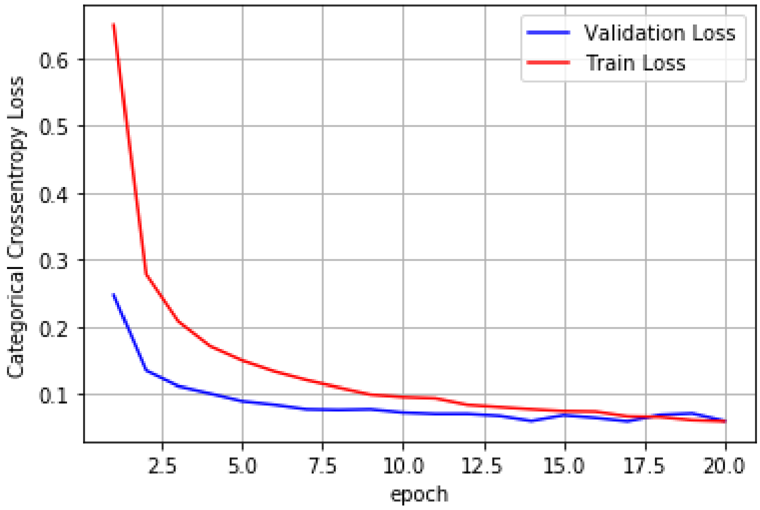
```
model_relu.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['ac

history = model_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,
```

```
validate on 10000 samples

===================] - 2s 39us/sample - loss: 0.6492 - acc: 0.8044 - val_loss:

===================] - 2s 30us/sample - loss: 0.2785 - acc: 0.9188 - val_loss:

===================] - 2s 29us/sample - loss: 0.2083 - acc: 0.9402 - val_loss:

===================] - 2s 31us/sample - loss: 0.1712 - acc: 0.9503 - val_loss:

===================] - 2s 30us/sample - loss: 0.1500 - acc: 0.9558 - val_loss:

===================] - 2s 31us/sample - loss: 0.1338 - acc: 0.9602 - val_loss:

===================] - 2s 31us/sample - loss: 0.1210 - acc: 0.9644 - val_loss:

===================] - 2s 32us/sample - loss: 0.1101 - acc: 0.9674 - val_loss:

===================] - 2s 29us/sample - loss: 0.0997 - acc: 0.9694 - val_loss:

===================] - 2s 31us/sample - loss: 0.0960 - acc: 0.9713 - val_loss:

===================] - 2s 30us/sample - loss: 0.0943 - acc: 0.9716 - val_loss:

===================] - 2s 29us/sample - loss: 0.0846 - acc: 0.9743 - val_loss:

===================] - 2s 29us/sample - loss: 0.0812 - acc: 0.9750 - val_loss:

===================] - 2s 31us/sample - loss: 0.0780 - acc: 0.9760 - val_loss:

===================] - 2s 30us/sample - loss: 0.0752 - acc: 0.9771 - val_loss:

===================] - 2s 30us/sample - loss: 0.0745 - acc: 0.9771 - val_loss:

===================] - 2s 29us/sample - loss: 0.0674 - acc: 0.9788 - val_loss:

===================] - 2s 28us/sample - loss: 0.0665 - acc: 0.9796 - val_loss:

===================] - 2s 30us/sample - loss: 0.0618 - acc: 0.9811 - val_loss:

===================] - 2s 29us/sample - loss: 0.0600 - acc: 0.9815 - val_loss:
```

```
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)
```

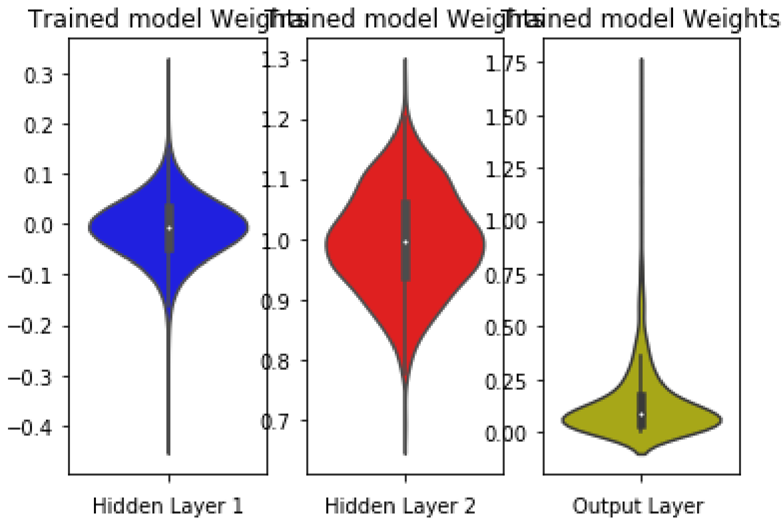
```
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# 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_epoc

# 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

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

```
Test score: 0.06073868394402962
Test accuracy: 0.9836
```




```
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()
```



4 Model C : 5 Hidden Layers

4.1 MLP + ReLU activation + Adam Optimizer

```
# Multilayer perceptron

model_relu = Sequential()
model_relu.add(Dense(1024, activation='relu', input_shape=(input_dim,)))
model_relu.add(Dense(512, activation='relu'))
model_relu.add(Dense(256, activation='relu'))
model_relu.add(Dense(128, activation='relu'))
model_relu.add(Dense(64, activation='relu'))
model_relu.add(Dense(output_dim, activation='softmax'))

model_relu.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_41 (Dense)             (None, 1024)              803840
_________________________________________________________________
dense_42 (Dense)             (None, 512)               524800
_________________________________________________________________
dense_43 (Dense)             (None, 256)               131328
_________________________________________________________________
dense_44 (Dense)             (None, 128)               32896
```

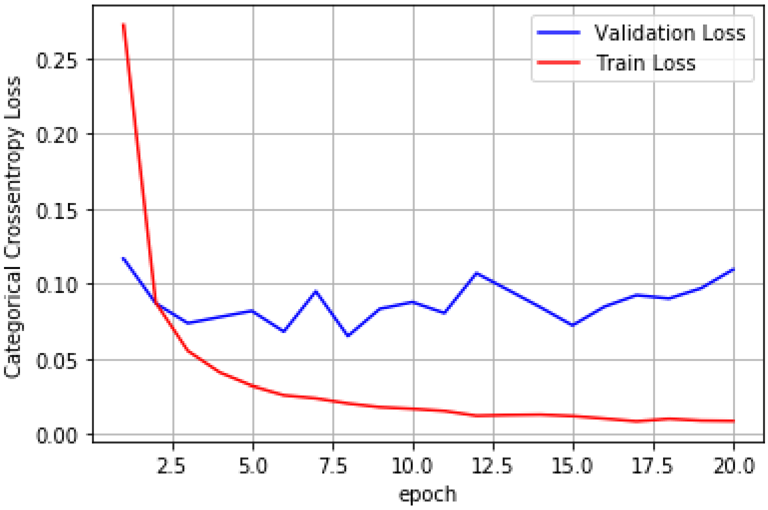
```
model_relu.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['ac

history = model_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 2s 29us/sample - loss: 0.2723
Epoch 2/20
60000/60000 [==============================] - 1s 23us/sample - loss: 0.0874
Epoch 3/20
60000/60000 [==============================] - 1s 22us/sample - loss: 0.0553
Epoch 4/20
60000/60000 [==============================] - 1s 22us/sample - loss: 0.0409
Epoch 5/20
60000/60000 [==============================] - 1s 22us/sample - loss: 0.0321
Epoch 6/20
60000/60000 [==============================] - 1s 22us/sample - loss: 0.0260
Epoch 7/20
60000/60000 [==============================] - 1s 22us/sample - loss: 0.0239
Epoch 8/20
60000/60000 [==============================] - 1s 22us/sample - loss: 0.0205
Epoch 9/20
60000/60000 [==============================] - 1s 22us/sample - loss: 0.0180
Epoch 10/20
60000/60000 [==============================] - 1s 22us/sample - loss: 0.0169
Epoch 11/20
60000/60000 [==============================] - 1s 23us/sample - loss: 0.0155
Epoch 12/20
60000/60000 [==============================] - 1s 22us/sample - loss: 0.0123
Epoch 13/20
60000/60000 [==============================] - 1s 22us/sample - loss: 0.0128
Epoch 14/20
60000/60000 [==============================] - 1s 22us/sample - loss: 0.0130
Epoch 15/20
60000/60000 [==============================] - 1s 22us/sample - loss: 0.0121
Epoch 16/20
60000/60000 [==============================] - 1s 22us/sample - loss: 0.0104
Epoch 17/20
60000/60000 [==============================] - 1s 22us/sample - loss: 0.0086
Epoch 18/20
60000/60000 [==============================] - 1s 22us/sample - loss: 0.0103
Epoch 19/20
60000/60000 [==============================] - 1s 22us/sample - loss: 0.0091
Epoch 20/20
60000/60000 [==============================] - 1s 22us/sample - loss: 0.0088
```

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

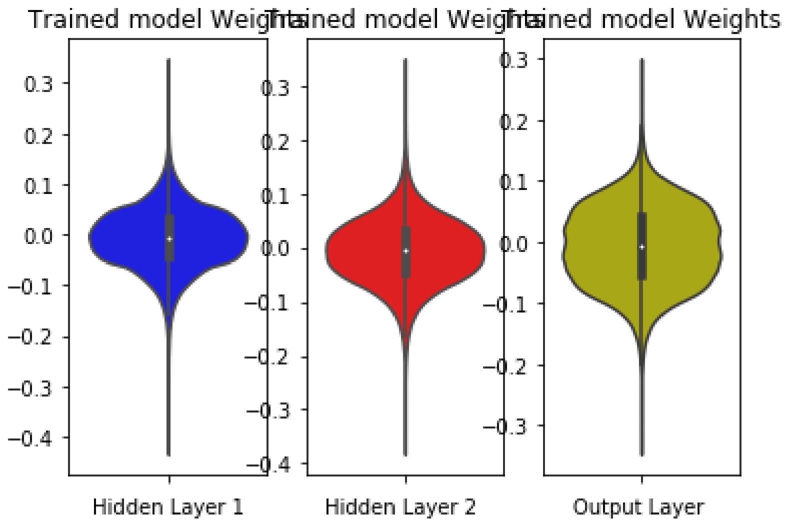
```
print('Test score:', score[0])
print('Test accuracy:', score[1])

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_epoc

# 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

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

```
Test score: 0.10951780684681689
Test accuracy: 0.9812
```




```
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()
```



4.2 MLP + ReLU activation + Adam Optimizer (Batch Normalization)

```
model_relu = Sequential()
model_relu.add(Dense(1024, activation='relu', input_shape=(input_dim,), kernel_ini
model_relu.add(BatchNormalization())

model_relu.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_init
model_relu.add(BatchNormalization())

model_relu.add(Dense(256, activation='relu', input_shape=(input_dim,), kernel_init
model_relu.add(BatchNormalization())

model_relu.add(Dense(128, activation='relu', input_shape=(input_dim,), kernel_init
model_relu.add(BatchNormalization())

model_relu.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0
model_relu.add(BatchNormalization())

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


model_relu.summary()
```

```
Model: "sequential_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_53 (Dense)             (None, 1024)              803840    
_________________________________________________________________
batch_normalization_13 (Batc (None, 1024)              4096      
_________________________________________________________________
dense_54 (Dense)             (None, 512)               524800    
_________________________________________________________________
batch_normalization_14 (Batc (None, 512)               2048      
_________________________________________________________________
dense_55 (Dense)             (None, 256)               131328    
_________________________________________________________________
batch_normalization_15 (Batc (None, 256)               1024      
_________________________________________________________________
dense_56 (Dense)             (None, 128)               32896     
_________________________________________________________________
batch_normalization_16 (Batc (None, 128)               512       
_________________________________________________________________
dense_57 (Dense)             (None, 64)                8256      
_________________________________________________________________
batch_normalization_17 (Batc (None, 64)                256       
_________________________________________________________________
dense_58 (Dense)             (None, 10)                650       
=================================================================
Total params: 1,509,706
Trainable params: 1,505,738
Non-trainable params: 3,968
_________________________________________________________________
```

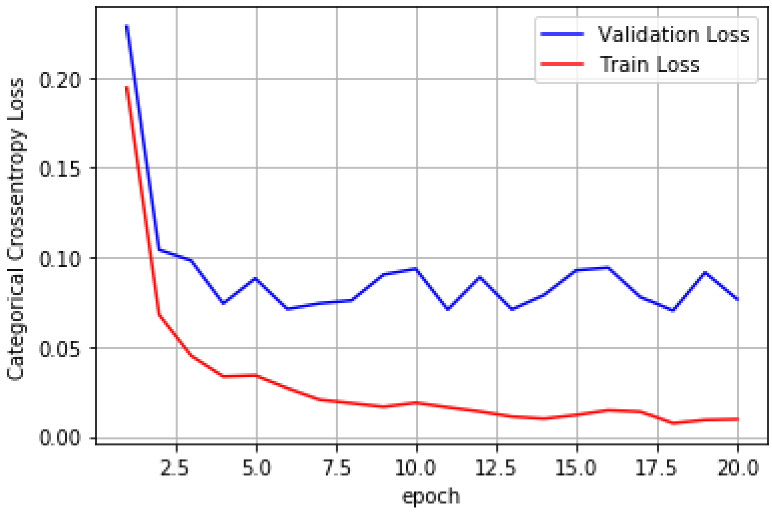
```
model_relu.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['ac

history = model_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 3s 53us/sample - loss: 0.1942
Epoch 2/20
60000/60000 [==============================] - 2s 36us/sample - loss: 0.0679
Epoch 3/20
60000/60000 [==============================] - 2s 38us/sample - loss: 0.0450
Epoch 4/20
60000/60000 [==============================] - 2s 40us/sample - loss: 0.0334
Epoch 5/20
60000/60000 [==============================] - 2s 38us/sample - loss: 0.0340
Epoch 6/20
60000/60000 [==============================] - 2s 38us/sample - loss: 0.0269
Epoch 7/20
60000/60000 [==============================] - 2s 38us/sample - loss: 0.0206
Epoch 8/20
60000/60000 [==============================] - 2s 40us/sample - loss: 0.0186
Epoch 9/20
60000/60000 [==============================] - 2s 36us/sample - loss: 0.0166
Epoch 10/20
60000/60000 [==============================] - 2s 36us/sample - loss: 0.0189
Epoch 11/20
60000/60000 [==============================] - 2s 38us/sample - loss: 0.0163
Epoch 12/20
60000/60000 [==============================] - 2s 38us/sample - loss: 0.0140
Epoch 13/20
60000/60000 [==============================] - 2s 36us/sample - loss: 0.0112
Epoch 14/20
60000/60000 [==============================] - 2s 36us/sample - loss: 0.0100
Epoch 15/20
60000/60000 [==============================] - 2s 37us/sample - loss: 0.0121
Epoch 16/20
60000/60000 [==============================] - 2s 39us/sample - loss: 0.0147
Epoch 17/20
60000/60000 [==============================] - 2s 39us/sample - loss: 0.0139
Epoch 18/20
60000/60000 [==============================] - 2s 39us/sample - loss: 0.0075
Epoch 19/20
60000/60000 [==============================] - 2s 39us/sample - loss: 0.0093
Epoch 20/20
60000/60000 [==============================] - 2s 40us/sample - loss: 0.0097
```

```
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)
```

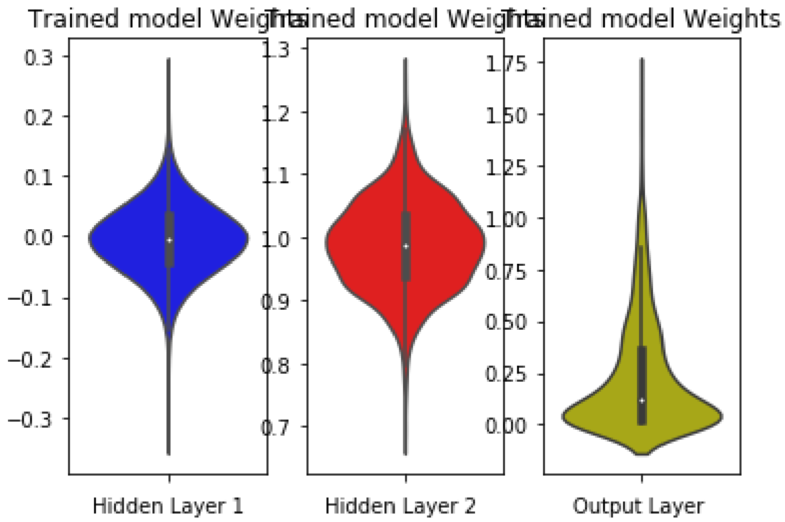
```
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# 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_epoc

# 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

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

```
Test score: 0.07663915940060397
Test accuracy: 0.9817
```




```
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()
```



4.3 MLP + ReLU activation + Adam Optimizer (Dropout)

```
from tensorflow.keras.layers import Dense, Dropout
model_relu = Sequential()
model_relu.add(Dense(1024, activation='relu', input_shape=(input_dim,)))
model_relu.add(Dropout(0.5))
model_relu.add(Dense(512, activation='relu', input_shape=(input_dim,)))
model_relu.add(Dropout(0.5))
model_relu.add(Dense(256, activation='relu', input_shape=(input_dim,)))
model_relu.add(Dropout(0.5))
model_relu.add(Dense(128, activation='relu', input_shape=(input_dim,)))
model_relu.add(Dropout(0.5))
model_relu.add(Dense(64, activation='relu'))
model_relu.add(Dropout(0.5))
model_relu.add(Dense(output_dim, activation='softmax'))


model_relu.summary()
```

```
Model: "sequential_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_59 (Dense)             (None, 1024)              803840    
_________________________________________________________________
dropout_11 (Dropout)         (None, 1024)              0         
_________________________________________________________________
dense_60 (Dense)             (None, 512)               524800    
_________________________________________________________________
dropout_12 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_61 (Dense)             (None, 256)               131328    
_________________________________________________________________
dropout_13 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_62 (Dense)             (None, 128)               32896     
_________________________________________________________________
dropout_14 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_63 (Dense)             (None, 64)                8256      
_________________________________________________________________
dropout_15 (Dropout)         (None, 64)                0         
_________________________________________________________________
dense_64 (Dense)             (None, 10)                650       
=================================================================
Total params: 1,501,770
Trainable params: 1,501,770
Non-trainable params: 0
_________________________________________________________________
```

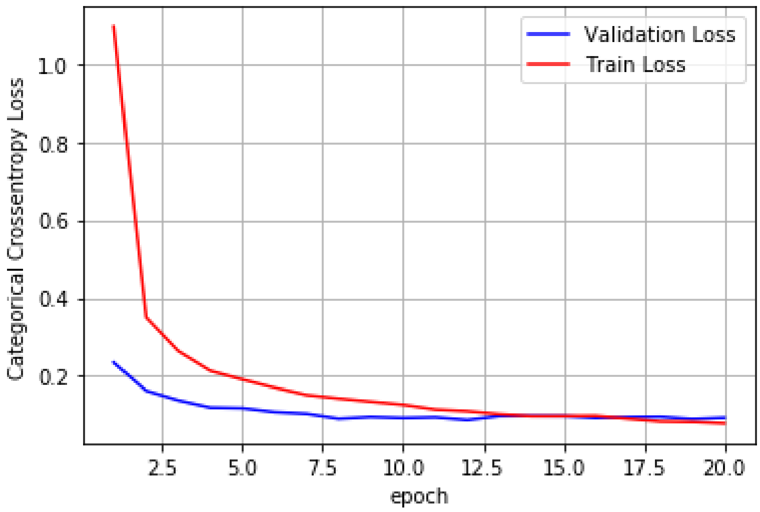
```
model_relu.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['ac

history = model_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,
```

```
validate on 10000 samples
===================] - 2s 35us/sample - loss: 1.0954 - acc: 0.6175 - val_loss:
===================] - 1s 24us/sample - loss: 0.3498 - acc: 0.9142 - val_loss:
===================] - 1s 24us/sample - loss: 0.2637 - acc: 0.9396 - val_loss:
===================] - 1s 24us/sample - loss: 0.2130 - acc: 0.9511 - val_loss:
===================] - 1s 24us/sample - loss: 0.1910 - acc: 0.9578 - val_loss:
===================] - 1s 23us/sample - loss: 0.1701 - acc: 0.9613 - val_loss:
===================] - 1s 24us/sample - loss: 0.1511 - acc: 0.9658 - val_loss:
===================] - 1s 23us/sample - loss: 0.1421 - acc: 0.9680 - val_loss:
===================] - 1s 24us/sample - loss: 0.1342 - acc: 0.9695 - val_loss:
===================] - 1s 24us/sample - loss: 0.1263 - acc: 0.9712 - val_loss:
===================] - 1s 24us/sample - loss: 0.1143 - acc: 0.9741 - val_loss:
===================] - 1s 23us/sample - loss: 0.1100 - acc: 0.9744 - val_loss:
===================] - 1s 23us/sample - loss: 0.1024 - acc: 0.9773 - val_loss:
===================] - 1s 23us/sample - loss: 0.0975 - acc: 0.9770 - val_loss:
===================] - 1s 23us/sample - loss: 0.0977 - acc: 0.9772 - val_loss:
===================] - 1s 23us/sample - loss: 0.0984 - acc: 0.9773 - val_loss:
===================] - 1s 23us/sample - loss: 0.0910 - acc: 0.9785 - val_loss:
===================] - 1s 24us/sample - loss: 0.0841 - acc: 0.9800 - val_loss:
===================] - 1s 23us/sample - loss: 0.0829 - acc: 0.9807 - val_loss:
===================] - 1s 24us/sample - loss: 0.0793 - acc: 0.9816 - val_loss:
```

```
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)
```

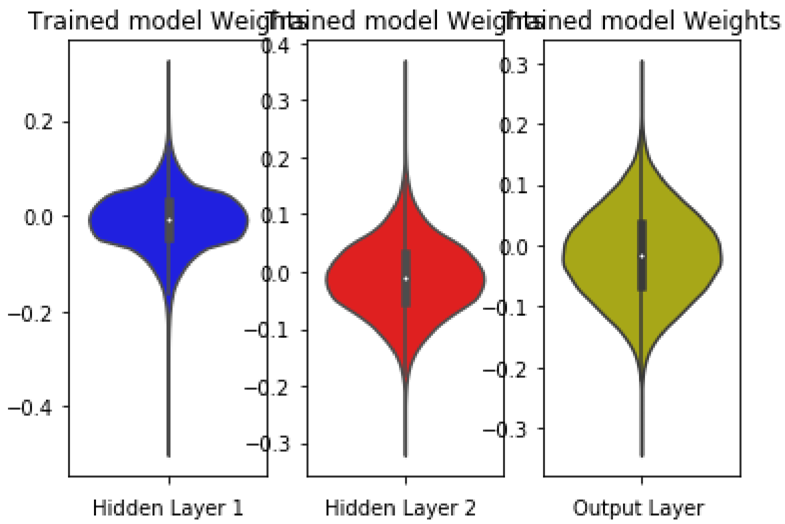
```
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# 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_epoc

# 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

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

```
Test score: 0.09344992985821433
Test accuracy: 0.9832
```




```
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()
```



4.4 MLP + ReLU activation + Adam Optimizer (Batch Normalization

```
# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormalizat
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.layers import BatchNormalization

model_relu = Sequential()
model_relu.add(Dense(1024, activation='relu', input_shape=(input_dim,), kernel_ini
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_init
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(256, activation='relu', input_shape=(input_dim,), kernel_init
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(64, activation='relu', input_shape=(input_dim,), kernel_initi
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(output_dim, activation='softmax'))

model_relu.summary()
```

```
Model: "sequential_15"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_65 (Dense)             (None, 1024)              803840
_________________________________________________________________
batch_normalization_18 (Batc (None, 1024)              4096
_________________________________________________________________
dropout_16 (Dropout)         (None, 1024)              0
_________________________________________________________________
dense_66 (Dense)             (None, 512)               524800
_________________________________________________________________
batch_normalization_19 (Batc (None, 512)               2048
_________________________________________________________________
dropout_17 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_67 (Dense)             (None, 256)               131328
_________________________________________________________________
batch_normalization_20 (Batc (None, 256)               1024
_________________________________________________________________
dropout_18 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_68 (Dense)             (None, 128)               32896
_________________________________________________________________
batch_normalization_21 (Batc (None, 128)               512
_________________________________________________________________
dropout_19 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_69 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_22 (Batc (None, 64)                256
_________________________________________________________________
dropout_20 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_70 (Dense)             (None, 10)                650
=================================================================
Total params: 1,509,706
Trainable params: 1,505,738
Non-trainable params: 3,968
_________________________________________________________________
```

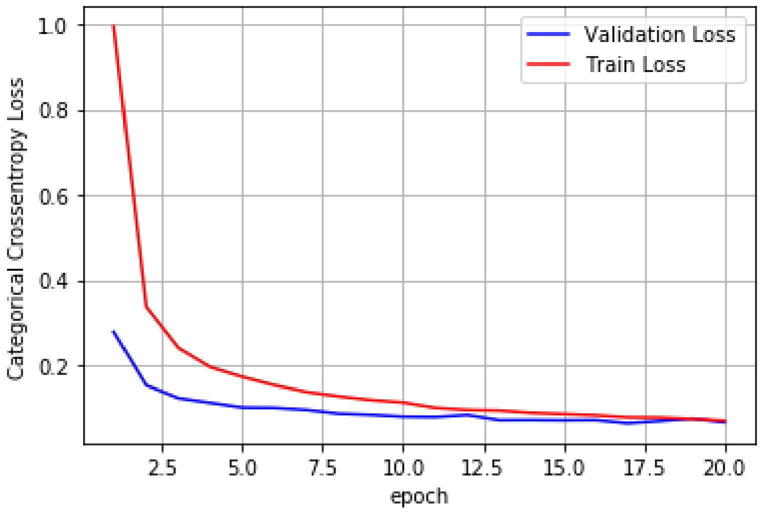
```
adam', loss='categorical_crossentropy', metrics=['accuracy'])

n, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_
```

```
validate on 10000 samples
===================] - 3s 57us/sample - loss: 0.9945 - acc: 0.6910 - val_loss:
===================] - 2s 39us/sample - loss: 0.3369 - acc: 0.9088 - val_loss:
===================] - 2s 40us/sample - loss: 0.2403 - acc: 0.9369 - val_loss:
===================] - 2s 40us/sample - loss: 0.1955 - acc: 0.9483 - val_loss:
===================] - 2s 39us/sample - loss: 0.1726 - acc: 0.9548 - val_loss:
===================] - 2s 41us/sample - loss: 0.1533 - acc: 0.9606 - val_loss:
===================] - 2s 40us/sample - loss: 0.1369 - acc: 0.9646 - val_loss:
===================] - 2s 39us/sample - loss: 0.1267 - acc: 0.9661 - val_loss:
===================] - 2s 40us/sample - loss: 0.1183 - acc: 0.9694 - val_loss:
===================] - 2s 41us/sample - loss: 0.1125 - acc: 0.9707 - val_loss:
===================] - 2s 41us/sample - loss: 0.1001 - acc: 0.9740 - val_loss:
===================] - 2s 40us/sample - loss: 0.0952 - acc: 0.9755 - val_loss:
===================] - 2s 40us/sample - loss: 0.0937 - acc: 0.9750 - val_loss:
===================] - 2s 42us/sample - loss: 0.0884 - acc: 0.9761 - val_loss:
===================] - 2s 40us/sample - loss: 0.0856 - acc: 0.9776 - val_loss:
===================] - 2s 40us/sample - loss: 0.0828 - acc: 0.9790 - val_loss:
===================] - 2s 41us/sample - loss: 0.0779 - acc: 0.9797 - val_loss:
===================] - 3s 43us/sample - loss: 0.0773 - acc: 0.9801 - val_loss:
===================] - 2s 41us/sample - loss: 0.0740 - acc: 0.9804 - val_loss:
===================] - 2s 41us/sample - loss: 0.0699 - acc: 0.9814 - val_loss:
```

```
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_epoc

# 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

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

```
Test score: 0.06646238846067572
Test accuracy: 0.9837
```




```
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')
```