mnist

```
# if you keras is not using tensorflow as backend set "KERAS_BACKEND=tensorflow" use this
from keras.utils import np_utils
from keras.datasets import mnist
import seaborn as sns
%matplotlib notebook
import matplotlib.pyplot as plt
from keras.initializers import he_normal
```

```
Using TensorFlow backend.
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 %te
```

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

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

```
# to get number of train and test points
print("Number of training examples :", X_train.shape[0], "and each image is of shape (%d, 
print("Number of training examples :", X_test.shape[0], "and each image is of shape (%d, %
```

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

```
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]
```

```
# 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
```

```
# 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.]
```

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

```
# some model parameters

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

batch_size = 128 # batch size
nb_epoch = 20# epoch
```

```
X_train.shape[1]
```

```
784
```

2 layer without dropout and batchnormal

```
#he_normal(seed=None)
model_relu = Sequential()
model_relu.add(Dense(500, activation='relu', input_shape=(input_dim,), kernel_initializer=
model_relu.add(Dense(224, activation='relu', kernel_initializer=he_normal(seed=None)) )
#model_relu.add(Dense(50, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))
```

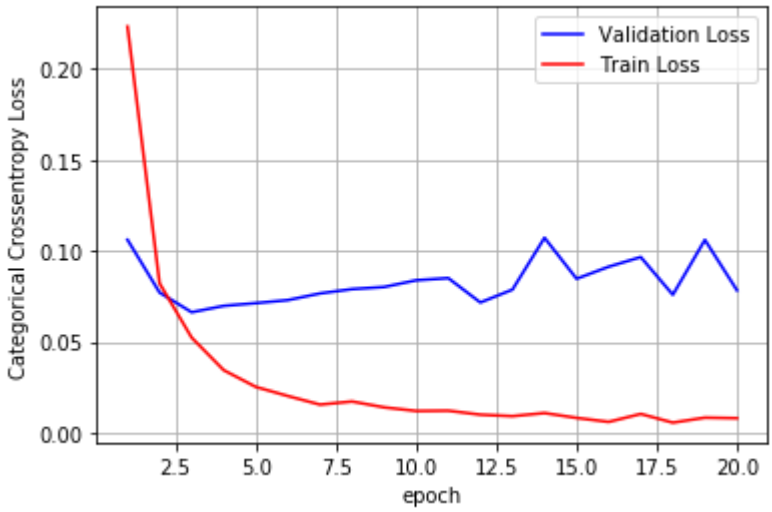
```
model_relu.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy']

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.2231 - acc: 0.
Epoch 2/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0821 - acc: 0.
Epoch 3/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0523 - acc: 0.
Epoch 4/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0345 - acc: 0.
Epoch 5/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0253 - acc: 0.
Epoch 6/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0203 - acc: 0.
Epoch 7/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.0156 - acc: 0.
Epoch 8/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0173 - acc: 0.
Epoch 9/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0141 - acc: 0.
Epoch 10/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0121 - acc: 0.
Epoch 11/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0122 - acc: 0.
Epoch 12/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0100 - acc: 0.
Epoch 13/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0092 - acc: 0.
Epoch 14/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0109 - acc: 0.
Epoch 15/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0083 - acc: 0.
Epoch 16/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0061 - acc: 0.
Epoch 17/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0105 - acc: 0.
Epoch 18/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.0057 - acc: 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, verbo

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

```
Test score: 0.07829980347480105
Test accuracy: 0.9832
```



2 layer without drop out with batch norm

```
from keras.layers.normalization import BatchNormalization

model_batch = Sequential()

model_batch.add(Dense(500, activation='relu', input_shape=(input_dim,), kernel_initializer
model_batch.add(BatchNormalization())

model_batch.add(Dense(224, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_batch.add(BatchNormalization())



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

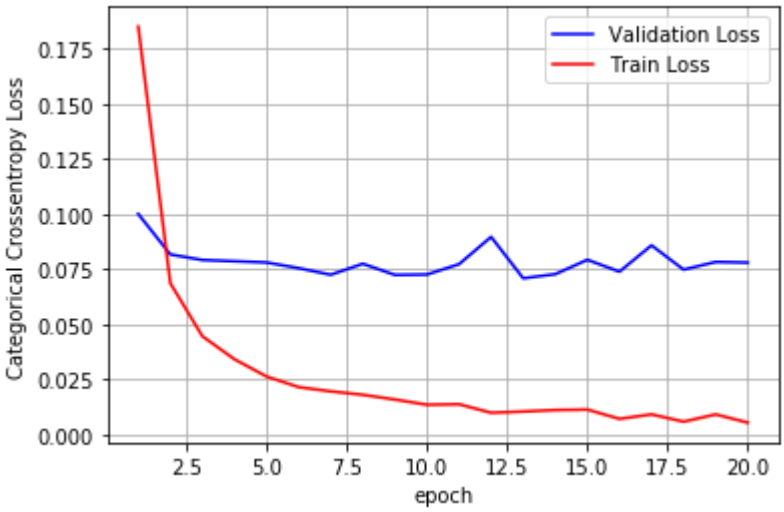
```
model_batch.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 9s 153us/step - loss: 0.1852 - acc: 0.
Epoch 2/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.0686 - acc: 0.
Epoch 3/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.0445 - acc: 0.
Epoch 4/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.0340 - acc: 0.
Epoch 5/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.0262 - acc: 0.
Epoch 6/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.0214 - acc: 0.
Epoch 7/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.0195 - acc: 0.
Epoch 8/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0179 - acc: 0.
Epoch 9/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.0158 - acc: 0.
Epoch 10/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.0134 - acc: 0.
Epoch 11/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.0136 - acc: 0.
Epoch 12/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.0097 - acc: 0.
Epoch 13/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.0103 - acc: 0.
Epoch 14/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.0110 - acc: 0.
Epoch 15/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.0112 - acc: 0.
Epoch 16/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.0070 - acc: 0.
Epoch 17/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0090 - acc: 0.
Epoch 18/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.0057 - acc: 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, verbo

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

```
Test score: 0.0779731295770489
Test accuracy: 0.982
```



2 layer with drop out without batch norm

```
model_drop = Sequential()

model_drop.add(Dense(500, activation='relu', input_shape=(input_dim,), kernel_initializer=
#model_drop.add(BatchNormalization())
model_drop.add(Dropout(rate=0.5))

model_drop.add(Dense(224, activation='relu', kernel_initializer=he_normal(seed=None)) )
#model_drop.add(BatchNormalization())
model_drop.add(Dropout(rate=0.5))



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

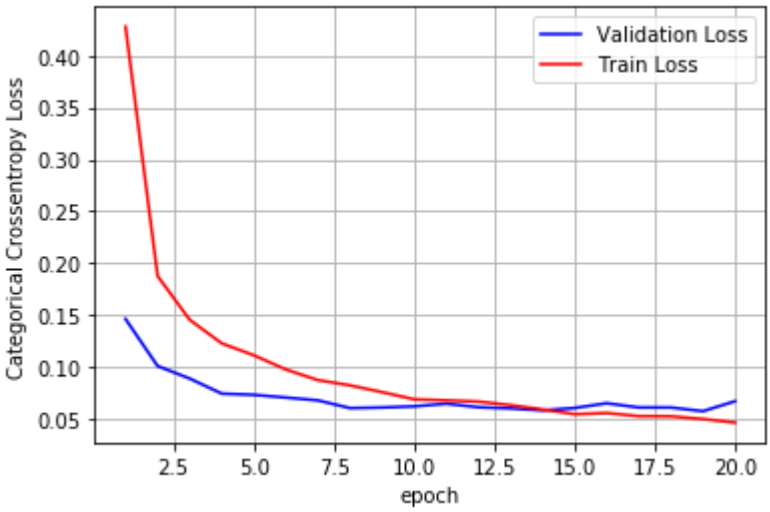
```
model_drop.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy']

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 9s 149us/step - loss: 0.4279 - acc: 0.
Epoch 2/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.1877 - acc: 0.
Epoch 3/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.1456 - acc: 0.
Epoch 4/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.1229 - acc: 0.
Epoch 5/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.1114 - acc: 0.
Epoch 6/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.0977 - acc: 0.
Epoch 7/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.0875 - acc: 0.
Epoch 8/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.0824 - acc: 0.
Epoch 9/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.0759 - acc: 0.
Epoch 10/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.0690 - acc: 0.
Epoch 11/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.0679 - acc: 0.
Epoch 12/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.0668 - acc: 0.
Epoch 13/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.0634 - acc: 0.
Epoch 14/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0593 - acc: 0.
Epoch 15/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.0546 - acc: 0.
Epoch 16/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.0557 - acc: 0.
Epoch 17/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.0528 - acc: 0.
Epoch 18/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.0526 - acc: 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, verbo

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

```
Test score: 0.06712127507654286
Test accuracy: 0.9829
```



2 layer with dropout and batch norm

```
model_drop = Sequential()

model_drop.add(Dense(500, activation='relu', input_shape=(input_dim,), kernel_initializer=
model_drop.add(BatchNormalization())
model_drop.add(Dropout(rate=0.5))

model_drop.add(Dense(224, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(rate=0.5))

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

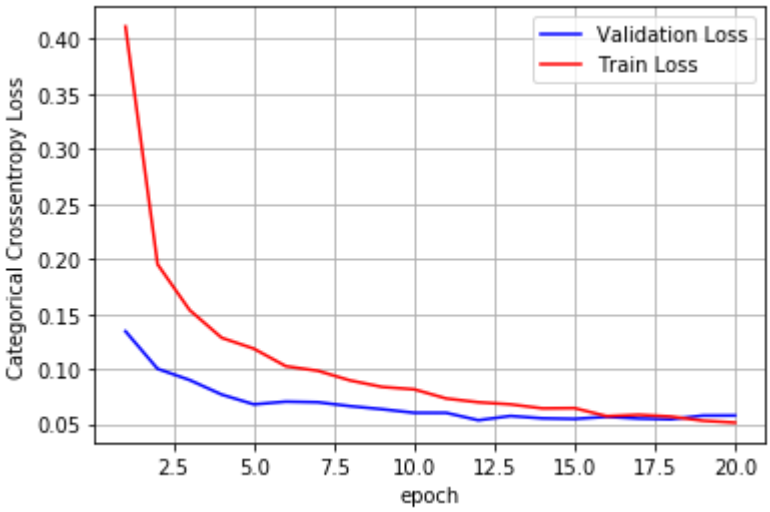
```
model_drop.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy']

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 11s 175us/step - loss: 0.4112 - acc: 0
Epoch 2/20
60000/60000 [==============================] - 9s 150us/step - loss: 0.1951 - acc: 0.
Epoch 3/20
60000/60000 [==============================] - 9s 150us/step - loss: 0.1535 - acc: 0.
Epoch 4/20
60000/60000 [==============================] - 9s 149us/step - loss: 0.1283 - acc: 0.
Epoch 5/20
60000/60000 [==============================] - 9s 150us/step - loss: 0.1185 - acc: 0.
Epoch 6/20
60000/60000 [==============================] - 9s 150us/step - loss: 0.1025 - acc: 0.
Epoch 7/20
60000/60000 [==============================] - 9s 150us/step - loss: 0.0984 - acc: 0.
Epoch 8/20
60000/60000 [==============================] - 9s 149us/step - loss: 0.0896 - acc: 0.
Epoch 9/20
60000/60000 [==============================] - 9s 149us/step - loss: 0.0837 - acc: 0.
Epoch 10/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.0815 - acc: 0.
Epoch 11/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.0732 - acc: 0.
Epoch 12/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.0698 - acc: 0.
Epoch 13/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.0679 - acc: 0.
Epoch 14/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.0643 - acc: 0.
Epoch 15/20
60000/60000 [==============================] - 9s 150us/step - loss: 0.0645 - acc: 0.
Epoch 16/20
60000/60000 [==============================] - 9s 150us/step - loss: 0.0572 - acc: 0.
Epoch 17/20
60000/60000 [==============================] - 9s 151us/step - loss: 0.0584 - acc: 0.
Epoch 18/20
60000/60000 [==============================] - 9s 153us/step - loss: 0.0568 - acc: 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, verbo

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

```
Test score: 0.057800181532799616
Test accuracy: 0.9826
```



three layered mlp relu activation

```
# Multilayer perceptron

model_sigmoid = Sequential()
model_sigmoid.add(Dense(500, activation='relu', input_shape=(input_dim,)))# first layer 50
model_sigmoid.add(Dense(124, activation='relu'))# second layer 124 neurons
model_sigmoid.add(Dense(50, activation='relu'))# third layer 50 neurons
model_sigmoid.add(Dense(output_dim, activation='softmax'))
```

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

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

```
model_sigmoid.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 500)               392500    
_________________________________________________________________
dense_2 (Dense)              (None, 124)               62124     
_________________________________________________________________
dense_3 (Dense)              (None, 50)                6250      
_________________________________________________________________
dense_4 (Dense)              (None, 10)                510       
=================================================================
Total params: 461,384
Trainable params: 461,384
Non-trainable params: 0
_________________________________________________________________
```

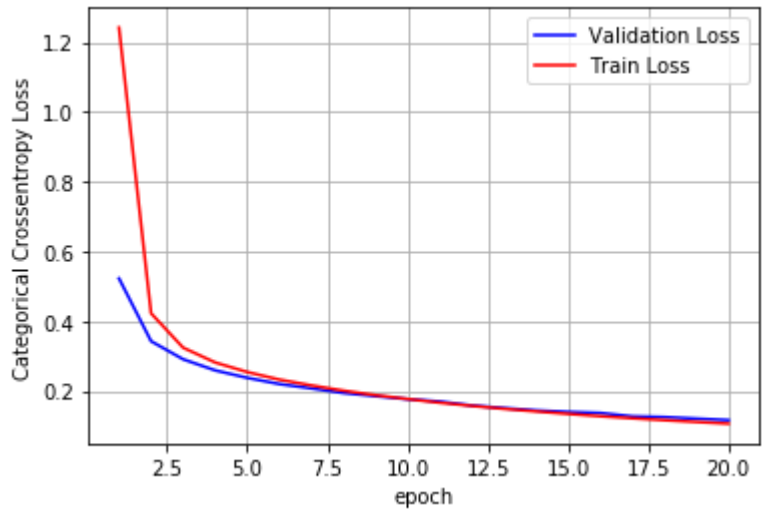
```
model_sigmoid.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy

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

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimizers.py:79

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

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python
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/tensorfl

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

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

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

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

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

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

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

60000/60000 [==============================] - 7s 113us/step - loss: 1.2421 - acc: 0.
Epoch 2/20
60000/60000 [==============================] - 7s 114us/step - loss: 0.4223 - acc: 0.
Epoch 3/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.3238 - acc: 0.
Epoch 4/20
60000/60000 [==============================] - 6s 104us/step - loss: 0.2812 - acc: 0.
Epoch 5/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.2538 - acc: 0.9
Epoch 6/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.2321 - acc: 0.9
Epoch 7/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.2154 - acc: 0.9
Epoch 8/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.2004 - acc: 0.9
Epoch 9/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.1874 - acc: 0.9
Epoch 10/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.1765 - acc: 0.9
Epoch 11/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.1659 - acc: 0.9
Epoch 12/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.1568 - acc: 0.9
Epoch 13/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.1483 - acc: 0.9
Epoch 14/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.1410 - acc: 0.9
Epoch 15/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.1338 - acc: 0.9
Epoch 16/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.1272 - acc: 0.9
Epoch 17/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.1212 - acc: 0.9
Epoch 18/20
60000/60000 [==============================] - 6s 98us/step - loss: 0.1158 - acc: 0.9
Epoch 19/20
```

```
60000/60000 [==============================] - 6s 97us/step - loss: 0.1107 - acc: 0.9
Epoch 20/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.1059 - acc: 0.9
```

```
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, verbo

# 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 epoch

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

```
Test score: 0.11591833184994757
Test accuracy: 0.9646
```



5 layered mlp with relu activation

```
model_relu5 = Sequential()
model_relu5.add(Dense(500, activation='relu', input_shape=(input_dim,)))# first layer 500
model_relu5.add(Dense(324, activation='relu'))# second layer 324 neurons
```

```
model_relu5.add(Dense(250, activation='relu'))# third layer 250 neurons
model_relu5.add(Dense(150, activation='relu'))# forth layer 150 neurons
model_relu5.add(Dense(50, activation='relu'))# fifth layer 50 neurons
model_relu5.add(Dense(output_dim, activation='softmax'))
```

```
model_relu5.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 500)               392500
_________________________________________________________________
dense_6 (Dense)              (None, 324)               162324
_________________________________________________________________
dense_7 (Dense)              (None, 250)               81250
_________________________________________________________________
dense_8 (Dense)              (None, 150)               37650
_________________________________________________________________
dense_9 (Dense)              (None, 50)                7550
_________________________________________________________________
dense_10 (Dense)             (None, 10)                510
=================================================================
Total params: 681,784
Trainable params: 681,784
Non-trainable params: 0
_________________________________________________________________
```

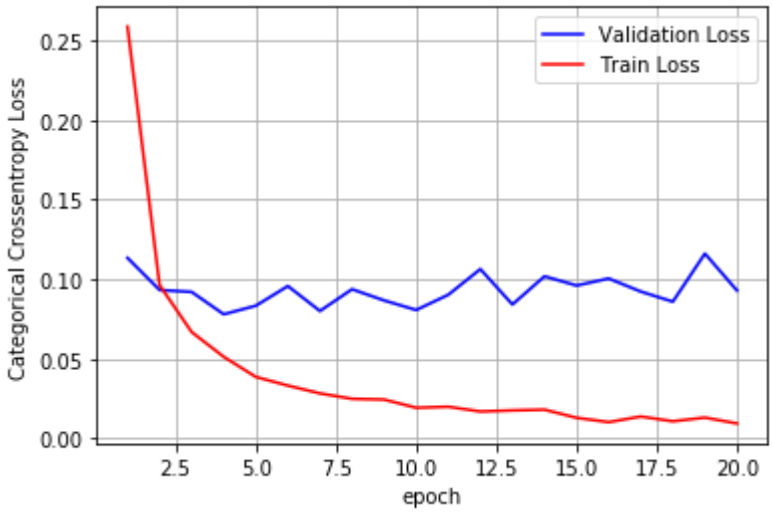
```
model_relu5.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'

history = model_relu5.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbos
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 11s 186us/step - loss: 0.2587 - acc: 0
Epoch 2/20
60000/60000 [==============================] - 11s 176us/step - loss: 0.0963 - acc: 0
Epoch 3/20
60000/60000 [==============================] - 10s 171us/step - loss: 0.0669 - acc: 0
Epoch 4/20
60000/60000 [==============================] - 11s 177us/step - loss: 0.0513 - acc: 0
Epoch 5/20
60000/60000 [==============================] - 11s 177us/step - loss: 0.0387 - acc: 0
Epoch 6/20
60000/60000 [==============================] - 10s 174us/step - loss: 0.0332 - acc: 0
Epoch 7/20
60000/60000 [==============================] - 10s 174us/step - loss: 0.0283 - acc: 0
Epoch 8/20
60000/60000 [==============================] - 10s 174us/step - loss: 0.0249 - acc: 0
Epoch 9/20
60000/60000 [==============================] - 11s 181us/step - loss: 0.0244 - acc: 0
Epoch 10/20
60000/60000 [==============================] - 11s 176us/step - loss: 0.0193 - acc: 0
Epoch 11/20
60000/60000 [==============================] - 11s 180us/step - loss: 0.0199 - acc: 0
Epoch 12/20
60000/60000 [==============================] - 10s 171us/step - loss: 0.0169 - acc: 0
Epoch 13/20
60000/60000 [==============================] - 10s 172us/step - loss: 0.0176 - acc: 0
Epoch 14/20
60000/60000 [==============================] - 11s 176us/step - loss: 0.0181 - acc: 0
Epoch 15/20
60000/60000 [==============================] - 11s 177us/step - loss: 0.0130 - acc: 0
Epoch 16/20
60000/60000 [==============================] - 11s 180us/step - loss: 0.0103 - acc: 0
Epoch 17/20
60000/60000 [==============================] - 11s 182us/step - loss: 0.0138 - acc: 0
Epoch 18/20
60000/60000 [==============================] - 11s 183us/step - loss: 0.0108 - acc: 0
Epoch 19/20
60000/60000 [==============================] - 11s 180us/step - loss: 0.0131 - acc: 0
Epoch 20/20
60000/60000 [==============================] - 11s 175us/step - loss: 0.0094 - acc: 0
```

```
score = model_relu5.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, verbo

# 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 epoch

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

```
Test score: 0.09293281461207743
Test accuracy: 0.9813
```



3 layered mlp with he_initializer without any batch_normalizer and d

```
#he_normal(seed=None)
model_relu = Sequential()
model_relu.add(Dense(500, activation='relu', input_shape=(input_dim,), kernel_initializer=
model_relu.add(Dense(124, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_relu.add(Dense(50, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))

model_relu.summary()
```

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

Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_11 (Dense)             (None, 500)               392500
_________________________________________________________________
dense_12 (Dense)             (None, 124)               62124
_________________________________________________________________
dense_13 (Dense)             (None, 50)                6250
_________________________________________________________________
dense_14 (Dense)             (None, 10)                510
=================================================================
Total params: 461,384
Trainable params: 461,384
Non-trainable params: 0
_________________________________________________________________
```

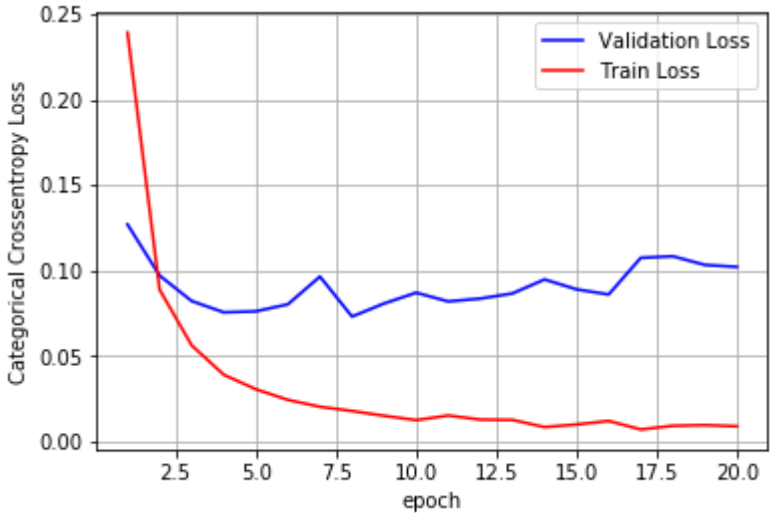
```
model_relu.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy']

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.2391 - acc: 0.
Epoch 2/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.0889 - acc: 0.
Epoch 3/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0563 - acc: 0.
Epoch 4/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.0392 - acc: 0.
Epoch 5/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0306 - acc: 0.
Epoch 6/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0244 - acc: 0.
Epoch 7/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.0204 - acc: 0.
Epoch 8/20
60000/60000 [==============================] - 7s 117us/step - loss: 0.0179 - acc: 0.
Epoch 9/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.0151 - acc: 0.
Epoch 10/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.0126 - acc: 0.
Epoch 11/20
60000/60000 [==============================] - 7s 114us/step - loss: 0.0153 - acc: 0.
Epoch 12/20
60000/60000 [==============================] - 7s 117us/step - loss: 0.0129 - acc: 0.
Epoch 13/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.0127 - acc: 0.
Epoch 14/20
60000/60000 [==============================] - 7s 115us/step - loss: 0.0085 - acc: 0.
Epoch 15/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.0100 - acc: 0.
Epoch 16/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.0121 - acc: 0.
Epoch 17/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.0071 - acc: 0.
Epoch 18/20
60000/60000 [==============================] - 7s 117us/step - loss: 0.0093 - acc: 0.
Epoch 19/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.0096 - acc: 0.
Epoch 20/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.0090 - acc: 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, verbo

# 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 epoch

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

```
Test score: 0.10216689538143828
Test accuracy: 0.9803
```



5 layered mlp with initializer = he_normal and relu activation withou

```
#he_normal(seed=None)
model_relu5 = Sequential()
model_relu5.add(Dense(500, activation='relu', input_shape=(input_dim,), kernel_initializer
model_relu5.add(Dense(324, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_relu5.add(Dense(250, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_relu5.add(Dense(150, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_relu5.add(Dense(50, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_relu5.add(Dense(output_dim, activation='softmax'))
```

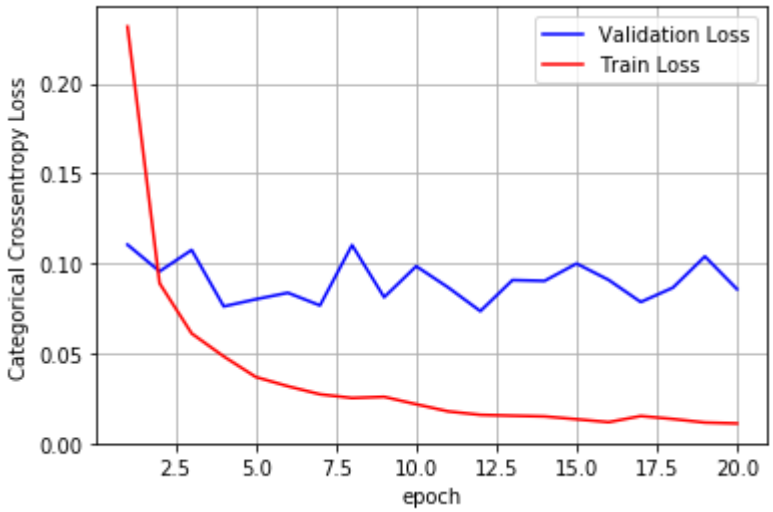
```
model_relu5.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'

history = model_relu5.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbos
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 11s 185us/step - loss: 0.2317 - acc: 0
Epoch 2/20
60000/60000 [==============================] - 11s 180us/step - loss: 0.0889 - acc: 0
Epoch 3/20
60000/60000 [==============================] - 10s 174us/step - loss: 0.0611 - acc: 0
Epoch 4/20
60000/60000 [==============================] - 11s 179us/step - loss: 0.0483 - acc: 0
Epoch 5/20
60000/60000 [==============================] - 10s 174us/step - loss: 0.0368 - acc: 0
Epoch 6/20
60000/60000 [==============================] - 10s 172us/step - loss: 0.0317 - acc: 0
Epoch 7/20
60000/60000 [==============================] - 11s 176us/step - loss: 0.0272 - acc: 0
Epoch 8/20
60000/60000 [==============================] - 11s 176us/step - loss: 0.0252 - acc: 0
Epoch 9/20
60000/60000 [==============================] - 10s 172us/step - loss: 0.0258 - acc: 0
Epoch 10/20
60000/60000 [==============================] - 11s 176us/step - loss: 0.0217 - acc: 0
Epoch 11/20
60000/60000 [==============================] - 11s 178us/step - loss: 0.0177 - acc: 0
Epoch 12/20
60000/60000 [==============================] - 11s 179us/step - loss: 0.0158 - acc: 0
Epoch 13/20
60000/60000 [==============================] - 11s 176us/step - loss: 0.0153 - acc: 0
Epoch 14/20
60000/60000 [==============================] - 11s 178us/step - loss: 0.0149 - acc: 0
Epoch 15/20
60000/60000 [==============================] - 11s 176us/step - loss: 0.0134 - acc: 0
Epoch 16/20
60000/60000 [==============================] - 11s 176us/step - loss: 0.0117 - acc: 0
Epoch 17/20
60000/60000 [==============================] - 11s 182us/step - loss: 0.0151 - acc: 0
Epoch 18/20
60000/60000 [==============================] - 11s 180us/step - loss: 0.0135 - acc: 0
Epoch 19/20
60000/60000 [==============================] - 11s 178us/step - loss: 0.0115 - acc: 0
Epoch 20/20
60000/60000 [==============================] - 11s 183us/step - loss: 0.0110 - acc: 0
```

```
score = model_relu5.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, verbo

# 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 epoch

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

```
Test score: 0.08569526613150633
Test accuracy: 0.9824
```



3 layered mlp with batch normalizer without dropout activation is re

```
from keras.layers.normalization import BatchNormalization

model_batch = Sequential()

model_batch.add(Dense(500, activation='relu', input_shape=(input_dim,), kernel_initializer
model_batch.add(BatchNormalization())

model_batch.add(Dense(124, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_batch.add(BatchNormalization())

model_batch.add(Dense(50, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_batch.add(BatchNormalization())

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

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

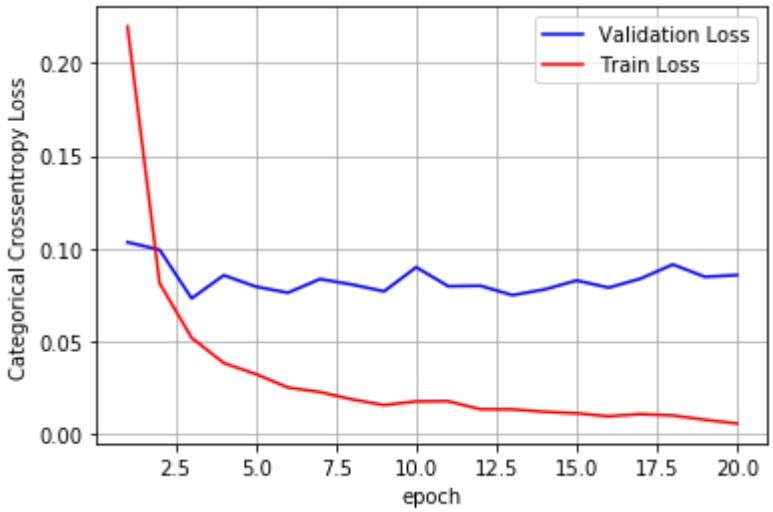
```
model_batch.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 10s 160us/step - loss: 0.2196 - acc: 0
Epoch 2/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.0815 - acc: 0.
Epoch 3/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0521 - acc: 0.
Epoch 4/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.0385 - acc: 0.
Epoch 5/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.0326 - acc: 0.
Epoch 6/20
60000/60000 [==============================] - 8s 140us/step - loss: 0.0254 - acc: 0.
Epoch 7/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.0229 - acc: 0.
Epoch 8/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.0189 - acc: 0.
Epoch 9/20
60000/60000 [==============================] - 9s 142us/step - loss: 0.0159 - acc: 0.
Epoch 10/20
60000/60000 [==============================] - 9s 143us/step - loss: 0.0179 - acc: 0.
Epoch 11/20
60000/60000 [==============================] - 9s 145us/step - loss: 0.0179 - acc: 0.
Epoch 12/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0136 - acc: 0.
Epoch 13/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.0136 - acc: 0.
Epoch 14/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0123 - acc: 0.
Epoch 15/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.0115 - acc: 0.
Epoch 16/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0098 - acc: 0.
Epoch 17/20
60000/60000 [==============================] - 9s 142us/step - loss: 0.0110 - acc: 0.
Epoch 18/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.0103 - acc: 0.
Epoch 19/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.0080 - acc: 0.
Epoch 20/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.0060 - acc: 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, verbo

# 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 epoch

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

```
Test score: 0.08583917959941027
Test accuracy: 0.9802
```



5 layered mlp with batch normalizer without dropout activation is re

```
model_batch = Sequential()

model_batch.add(Dense(500, activation='relu', input_shape=(input_dim,), kernel_initializer
model_batch.add(BatchNormalization())

model_batch.add(Dense(324, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_batch.add(BatchNormalization())

model_batch.add(Dense(250, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_batch.add(BatchNormalization())

model_batch.add(Dense(150, activation='relu', input_shape=(input_dim,), kernel_initializer
model_batch.add(BatchNormalization())


model_batch.add(Dense(50, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_batch.add(BatchNormalization())
```

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

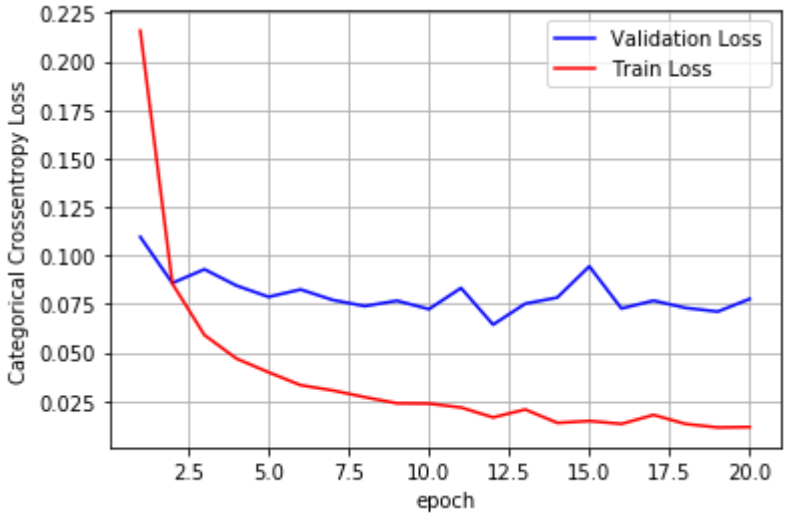
```
model_batch.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 15s 242us/step - loss: 0.2159 - acc: 0
Epoch 2/20
60000/60000 [==============================] - 13s 214us/step - loss: 0.0855 - acc: 0
Epoch 3/20
60000/60000 [==============================] - 13s 219us/step - loss: 0.0591 - acc: 0
Epoch 4/20
60000/60000 [==============================] - 13s 224us/step - loss: 0.0469 - acc: 0
Epoch 5/20
60000/60000 [==============================] - 13s 225us/step - loss: 0.0399 - acc: 0
Epoch 6/20
60000/60000 [==============================] - 13s 219us/step - loss: 0.0332 - acc: 0
Epoch 7/20
60000/60000 [==============================] - 13s 219us/step - loss: 0.0305 - acc: 0
Epoch 8/20
60000/60000 [==============================] - 13s 218us/step - loss: 0.0271 - acc: 0
Epoch 9/20
60000/60000 [==============================] - 13s 216us/step - loss: 0.0239 - acc: 0
Epoch 10/20
60000/60000 [==============================] - 13s 216us/step - loss: 0.0237 - acc: 0
Epoch 11/20
60000/60000 [==============================] - 13s 219us/step - loss: 0.0217 - acc: 0
Epoch 12/20
60000/60000 [==============================] - 14s 225us/step - loss: 0.0166 - acc: 0
Epoch 13/20
60000/60000 [==============================] - 13s 220us/step - loss: 0.0207 - acc: 0
Epoch 14/20
60000/60000 [==============================] - 13s 221us/step - loss: 0.0138 - acc: 0
Epoch 15/20
60000/60000 [==============================] - 13s 223us/step - loss: 0.0148 - acc: 0
Epoch 16/20
60000/60000 [==============================] - 13s 216us/step - loss: 0.0133 - acc: 0
Epoch 17/20
60000/60000 [==============================] - 14s 230us/step - loss: 0.0179 - acc: 0
Epoch 18/20
60000/60000 [==============================] - 13s 216us/step - loss: 0.0132 - acc: 0
Epoch 19/20
60000/60000 [==============================] - 13s 211us/step - loss: 0.0114 - acc: 0
Epoch 20/20
60000/60000 [==============================] - 13s 214us/step - loss: 0.0116 - acc: 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, verbo

# 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 epoch

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

```
Test score: 0.07765144463920151
Test accuracy: 0.9805
```



3 layered mlp without batch normalizer with dropout activation is re

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

```
from keras.layers import Dropout

model_drop = Sequential()

model_drop.add(Dense(500, activation='relu', input_shape=(input_dim,), kernel_initializer=
#model_drop.add(BatchNormalization())
model_drop.add(Dropout(rate=0.5))

model_drop.add(Dense(124, activation='relu', kernel_initializer=he_normal(seed=None)) )
#model_drop.add(BatchNormalization())
model_drop.add(Dropout(rate=0.5))

model_drop.add(Dense(50, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_drop.add(Dropout(rate=0.5))
```

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

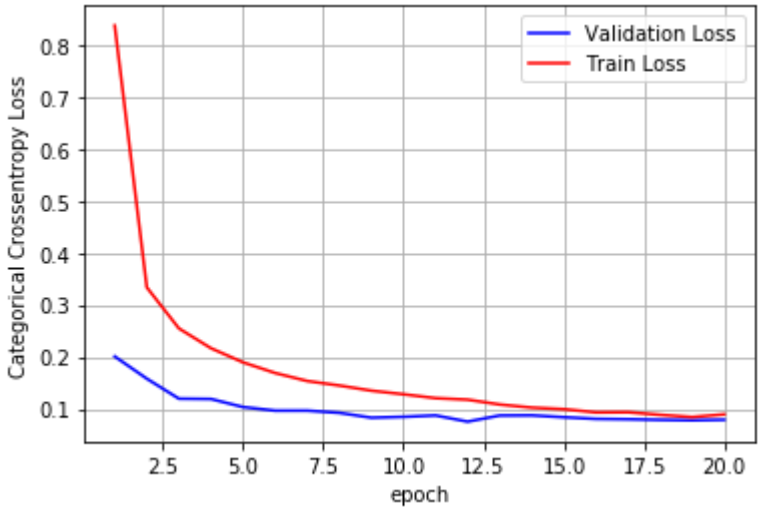
```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob
```

```
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
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 10s 161us/step - loss: 0.8373 - acc: 0
Epoch 2/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.3337 - acc: 0.
Epoch 3/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.2548 - acc: 0.
Epoch 4/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.2166 - acc: 0.
Epoch 5/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.1895 - acc: 0.
Epoch 6/20
60000/60000 [==============================] - 8s 140us/step - loss: 0.1690 - acc: 0.
Epoch 7/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.1536 - acc: 0.
Epoch 8/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.1450 - acc: 0.
Epoch 9/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.1350 - acc: 0.
Epoch 10/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.1281 - acc: 0.
Epoch 11/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.1206 - acc: 0.
Epoch 12/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.1177 - acc: 0.
Epoch 13/20
60000/60000 [==============================] - 8s 140us/step - loss: 0.1084 - acc: 0.
Epoch 14/20
60000/60000 [==============================] - 8s 128us/step - loss: 0.1024 - acc: 0.
Epoch 15/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0994 - acc: 0.
Epoch 16/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.0933 - acc: 0.
Epoch 17/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.0936 - acc: 0.
Epoch 18/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.0885 - acc: 0.
Epoch 19/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.0843 - acc: 0.
Epoch 20/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.0894 - acc: 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, verbo

# 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 epoch

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

```
Test score: 0.07902954334445494
Test accuracy: 0.9818
```



5 layered mlp without batch normalizer with dropout activation is re

```
model_drop = Sequential()

model_drop.add(Dense(500, activation='relu', input_shape=(input_dim,), kernel_initializer=
#model_drop.add(BatchNormalization())
model_drop.add(Dropout(rate=0.5))

model_drop.add(Dense(324, activation='relu', kernel_initializer=he_normal(seed=None)) )
#model_drop.add(BatchNormalization())
model_drop.add(Dropout(rate=0.5))
```

```
model_drop.add(Dense(250, activation='relu',kernel_initializer=he_normal(seed=None)))
#model_drop.add(BatchNormalization())
model_drop.add(Dropout(rate=0.5))

model_drop.add(Dense(150, activation='relu', kernel_initializer=he_normal(seed=None)) )
#model_drop.add(BatchNormalization())
model_drop.add(Dropout(rate=0.5))

model_drop.add(Dense(50, activation='relu', kernel_initializer=he_normal(seed=None)) )
#model_drop.add(BatchNormalization())
model_drop.add(Dropout(rate=0.5))

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

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

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

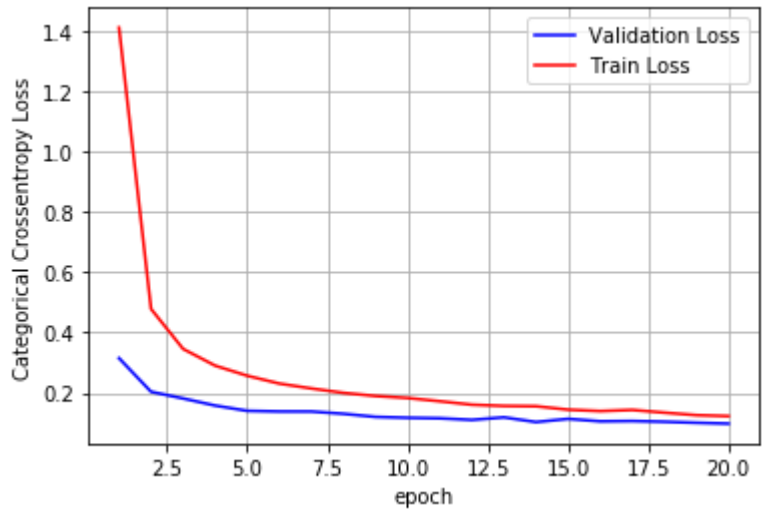
```
model_drop.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy']

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

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimizers.py:79

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

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python
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/tensorfl

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

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

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

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

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

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

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

60000/60000 [==============================] - 14s 232us/step - loss: 1.4113 - acc: 0
Epoch 2/20
60000/60000 [==============================] - 13s 212us/step - loss: 0.4782 - acc: 0
Epoch 3/20
60000/60000 [==============================] - 13s 212us/step - loss: 0.3456 - acc: 0
Epoch 4/20
60000/60000 [==============================] - 13s 211us/step - loss: 0.2901 - acc: 0
Epoch 5/20
60000/60000 [==============================] - 13s 213us/step - loss: 0.2567 - acc: 0
Epoch 6/20
60000/60000 [==============================] - 13s 210us/step - loss: 0.2308 - acc: 0
Epoch 7/20
60000/60000 [==============================] - 13s 214us/step - loss: 0.2145 - acc: 0
Epoch 8/20
60000/60000 [==============================] - 13s 213us/step - loss: 0.1999 - acc: 0
Epoch 9/20
60000/60000 [==============================] - 13s 214us/step - loss: 0.1896 - acc: 0
Epoch 10/20
60000/60000 [==============================] - 13s 213us/step - loss: 0.1829 - acc: 0
Epoch 11/20
60000/60000 [==============================] - 13s 209us/step - loss: 0.1721 - acc: 0
Epoch 12/20
60000/60000 [==============================] - 13s 214us/step - loss: 0.1607 - acc: 0
Epoch 13/20
60000/60000 [==============================] - 13s 212us/step - loss: 0.1567 - acc: 0
Epoch 14/20
60000/60000 [==============================] - 13s 214us/step - loss: 0.1556 - acc: 0
Epoch 15/20
60000/60000 [==============================] - 13s 213us/step - loss: 0.1442 - acc: 0
Epoch 16/20
60000/60000 [==============================] - 13s 212us/step - loss: 0.1396 - acc: 0
Epoch 17/20
60000/60000 [==============================] - 13s 214us/step - loss: 0.1433 - acc: 0
Epoch 18/20
60000/60000 [==============================] - 13s 209us/step - loss: 0.1344 - acc: 0
Epoch 19/20
```

```
60000/60000 [==============================] - 13s 211us/step - loss: 0.1261 - acc: 0
Epoch 20/20
60000/60000 [==============================] - 13s 209us/step - loss: 0.1229 - acc: 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, verbo

# 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 epoch

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

```
Test score: 0.09795631028999714
Test accuracy: 0.9796
```



3 layered mlp with batch normalizer with dropout activation is relu a

```
model_drop = Sequential()

model_drop.add(Dense(500, activation='relu', input_shape=(input_dim,), kernel_initializer=
```

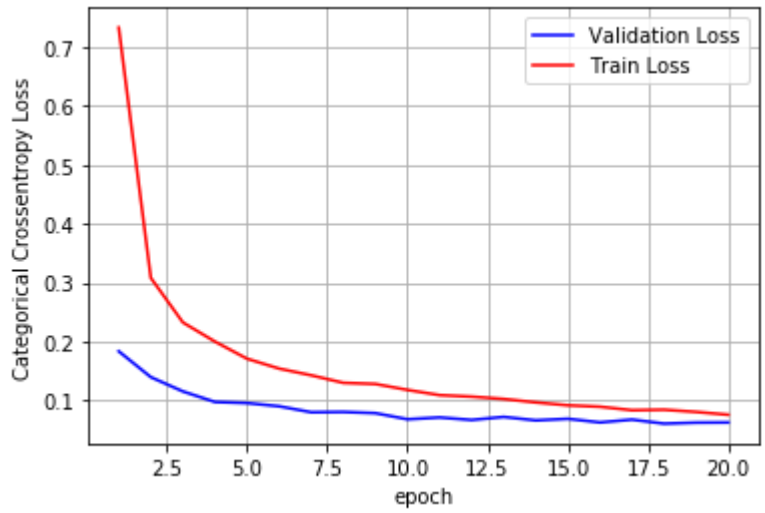
```
model_drop.add(BatchNormalization())
model_drop.add(Dropout(rate=0.5))

model_drop.add(Dense(124, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(rate=0.5))

model_drop.add(Dense(50, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(rate=0.5))

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

```
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
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 12s 199us/step - loss: 0.7329 - acc: 0
Epoch 2/20
60000/60000 [==============================] - 9s 157us/step - loss: 0.3083 - acc: 0.
Epoch 3/20
60000/60000 [==============================] - 10s 159us/step - loss: 0.2327 - acc: 0
Epoch 4/20
60000/60000 [==============================] - 10s 166us/step - loss: 0.2002 - acc: 0
Epoch 5/20
60000/60000 [==============================] - 10s 160us/step - loss: 0.1712 - acc: 0
Epoch 6/20
60000/60000 [==============================] - 10s 162us/step - loss: 0.1545 - acc: 0
Epoch 7/20
60000/60000 [==============================] - 9s 155us/step - loss: 0.1431 - acc: 0.
Epoch 8/20
60000/60000 [==============================] - 10s 161us/step - loss: 0.1303 - acc: 0
Epoch 9/20
60000/60000 [==============================] - 10s 169us/step - loss: 0.1284 - acc: 0
Epoch 10/20
60000/60000 [==============================] - 10s 166us/step - loss: 0.1183 - acc: 0
Epoch 11/20
60000/60000 [==============================] - 10s 161us/step - loss: 0.1094 - acc: 0
Epoch 12/20
60000/60000 [==============================] - 10s 159us/step - loss: 0.1069 - acc: 0
Epoch 13/20
60000/60000 [==============================] - 9s 157us/step - loss: 0.1029 - acc: 0.
Epoch 14/20
60000/60000 [==============================] - 10s 160us/step - loss: 0.0971 - acc: 0
Epoch 15/20
60000/60000 [==============================] - 10s 161us/step - loss: 0.0922 - acc: 0
Epoch 16/20
60000/60000 [==============================] - 9s 158us/step - loss: 0.0898 - acc: 0.
Epoch 17/20
60000/60000 [==============================] - 10s 167us/step - loss: 0.0839 - acc: 0
Epoch 18/20
60000/60000 [==============================] - 10s 159us/step - loss: 0.0847 - acc: 0
Epoch 19/20
60000/60000 [==============================] - 10s 165us/step - loss: 0.0809 - acc: 0
Epoch 20/20
60000/60000 [==============================] - 10s 158us/step - loss: 0.0762 - acc: 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, verbo

# 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 epoch

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

```
Test score: 0.06307774550883914
Test accuracy: 0.9816
```



5 layered mlp with batch normalizer with dropout activation is relu a

```
model_drop = Sequential()

model_drop.add(Dense(500, activation='relu', input_shape=(input_dim,), kernel_initializer=
model_drop.add(BatchNormalization())
model_drop.add(Dropout(rate=0.5))

model_drop.add(Dense(324, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_drop.add(BatchNormalization())
```

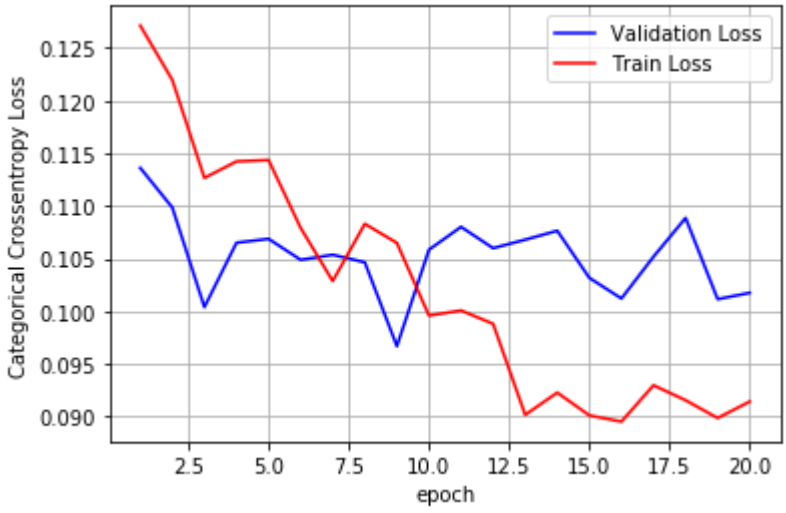
```
model_drop.add(Dropout(rate=0.5))



model_drop.add(Dense(250, activation='relu',kernel_initializer=he_normal(seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(rate=0.5))

model_drop.add(Dense(150, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(rate=0.5))

model_drop.add(Dense(50, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(rate=0.5))

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

```
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
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 14s 225us/step - loss: 0.1271 - acc: 0
Epoch 2/20
60000/60000 [==============================] - 13s 215us/step - loss: 0.1220 - acc: 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, verbo

# 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 epoch

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

```
Test score: 0.10175361420742388
Test accuracy: 0.983
```




```
# Please compare all your models using Prettytable library
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["layer", "type", "log_loss","accuracy"]
```

```
x.add_row(['2_layer','without dropout and batch_norm',0.07829980347480105,0.9832])
x.add_row(['2_layer','without dropout and with batch_norm',0.0779731295770489,0.982])
x.add_row(['2_layer','with dropout and without batch_norm',0.06712127507654286,0.9829])
x.add_row(['2_layer','with dropout and with batch_norm',0.057800181532799616,0.9826])
x.add_row(['3_layer','without dropout and batch_norm',0.10216689538143828,0.9803])
x.add_row(['5_layer','without dropout and batch_norm',0.08569526613150633,0.9824])
x.add_row(['3_layer','without dropout and with batch_norm',0.08583917959941027,0.9802])
x.add_row(['5_layer','without dropout and with batch_norm',0.07765144463920151,0.9805])
x.add_row(['3_layer','with dropout and without batch_norm',0.07902954334445494,0.9818])
x.add_row(['5_layer','with dropout and without batch_norm',0.09795631028999714,0.9796])
x.add_row(['3_layer','with dropout and with batch_norm',0.06307774550883914,0.9816])
x.add_row(['5_layer','with dropout and with batch_norm',0.10175361420742388,0.983])
```

```
print(x)
```

```
+---------+-------------------------------------+----------------------+----------+
|  layer  |                type                 |       log_loss       | accuracy |
+---------+-------------------------------------+----------------------+----------+
| 2_layer |    without dropout and batch_norm   | 0.07829980347480105  |  0.9832  |
| 2_layer | without dropout and with batch_norm |  0.0779731295770489  |  0.982   |
| 2_layer | with dropout and without batch_norm | 0.06712127507654286  |  0.9829  |
| 2_layer |   with dropout and with batch_norm  | 0.057800181532799616 |  0.9826  |
| 3_layer |    without dropout and batch_norm   | 0.10216689538143828  |  0.9803  |
| 5_layer |    without dropout and batch_norm   | 0.08569526613150633  |  0.9824  |
| 3_layer | without dropout and with batch_norm | 0.08583917959941027  |  0.9802  |
| 5_layer | without dropout and with batch_norm | 0.07765144463920151  |  0.9805  |
| 3_layer | with dropout and without batch_norm | 0.07902954334445494  |  0.9818  |
| 5_layer | with dropout and without batch_norm | 0.09795631028999714  |  0.9796  |
| 3_layer |   with dropout and with batch_norm  | 0.06307774550883914  |  0.9816  |
| 5_layer |   with dropout and with batch_norm  | 0.10175361420742388  |  0.983   |
+---------+-------------------------------------+----------------------+----------+
```