Keras -- MLPs on MNIST

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

```
Using TensorFlow backend.
```

```
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()
    plt.show()
```

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

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

```
print("Number of training examples :", X_train.shape[0], "and each image is of shape (%d, %d)
print("Number of training examples :", X_test.shape[0], "and each image is of shape (%d, %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 3 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)"%(X
print("Number of training examples :", X_test.shape[0], "and each image is of shape (%d)"%(X_
```

```
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
```

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

```
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.01176471 0.07058824 0.07058824 0.07058824
 0.49411765 0.53333333 0.68627451 0.10196078 0.65098039 1.
 0.96862745 0.49803922 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.11764706 0.14117647 0.36862745 0.60392157
 0.66666667 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
 0.88235294 0.6745098  0.99215686 0.94901961 0.76470588 0.25098039
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.19215686
 0.93333333 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
 0.99215686 0.99215686 0.99215686 0.98431373 0.36470588 0.32156863
 0.32156863 0.21960784 0.15294118 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.07058824 0.85882353 0.99215686
 0.99215686 0.99215686 0.99215686 0.99215686 0.77647059 0.71372549
 0.96862745 0.94509804 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.31372549 0.61176471 0.41960784 0.99215686
 0.99215686 0.80392157 0.04313725 0.         0.16862745 0.60392157
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.05490196 0.00392157 0.60392157 0.99215686 0.35294118
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.54509804 0.99215686 0.74509804 0.00784314 0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
```

```
       0.         0.         0.         0.         0.         0.04313725
```

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

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

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

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

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

Softmax classifier

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



# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 156
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'))
```

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

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


# Note: when using the categorical_crossentropy loss, your targets should be in categorical fo
# (e.g. if you have 10 classes, the target for each sample should be a 10-dimensional vector t
# 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_s
# validation_data=None, shuffle=True, class_weight=None, sample_weight=None, initial_epoch=0,
# validation_steps=None)

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

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

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

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

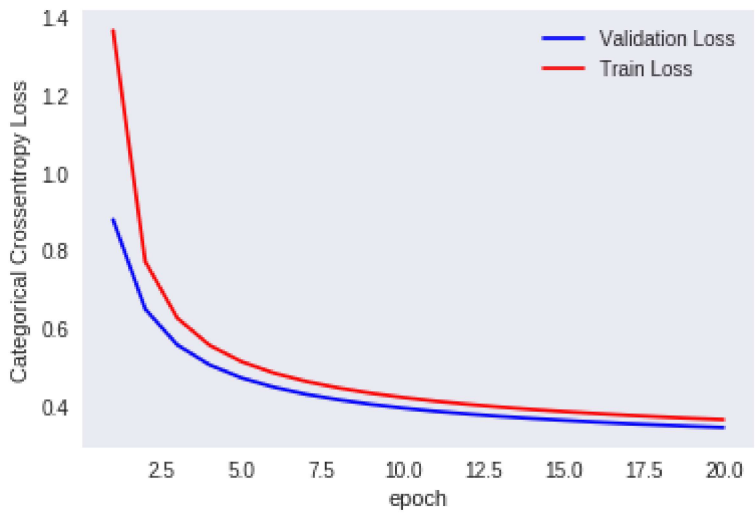
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
```

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

```
Test score: 0.3455672636508942
Test accuracy: 0.9051
```

```
Epoch 5/20
```

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

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

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

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

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

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



MLP + ReLU +Adam

```
# Multilayer perceptron

# https://arxiv.org/pdf/1707.09725.pdf#page=95
# for relu layers
# If we sample weights from a normal distribution N(0,σ) we satisfy this condition with σ=√(2/
# h1 =>  σ=√(2/(fan_in) = 0.059  => N(0,σ) = N(0,0.059)
# h2 =>  σ=√(2/(fan_in) = 0.113  => N(0,σ) = N(0,0.113)
```

```
model_relu = Sequential()
model_relu.add(Dense(556, activation='relu', input_shape=(input_dim,), kernel_initializer=Ran
model_relu.add(Dense(156, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev
model_relu.add(Dense(output_dim, activation='softmax'))

model_relu.summary()
```

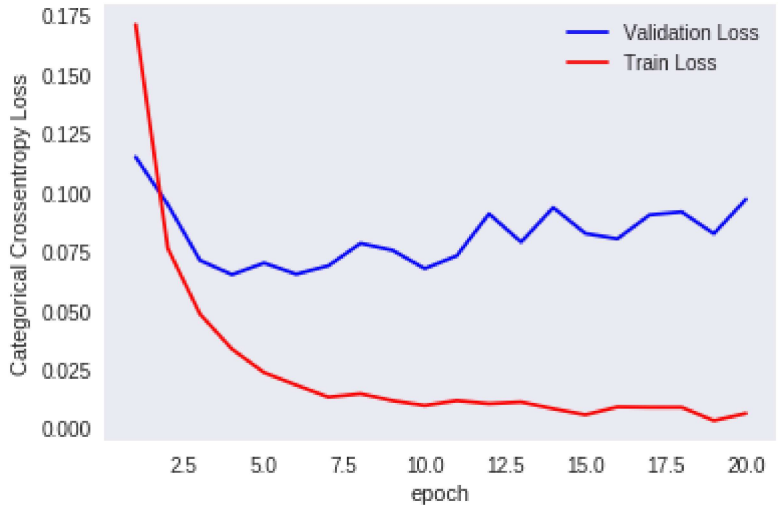
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 556)               436460
_________________________________________________________________
dense_3 (Dense)              (None, 156)               86892
_________________________________________________________________
dense_4 (Dense)              (None, 10)                1570
=================================================================
Total params: 524,922
Trainable params: 524,922
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=1,
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.1708 - acc: 0.9
Epoch 2/20
60000/60000 [==============================] - 7s 113us/step - loss: 0.0760 - acc: 0.9
Epoch 3/20
60000/60000 [==============================] - 7s 113us/step - loss: 0.0483 - acc: 0.9
Epoch 4/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.0334 - acc: 0.9
```

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

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

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

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

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

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

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

```
Test score: 0.09683581481487936
Test accuracy: 0.9802
```



Input(784)+ReLu(580)+ReLu(325)+ReLu(125)+Softm

```
# h1 =>  σ=√(2/(fan_in) = 0.058  => N(0,σ) = N(0,0.058)
# h2 =>  σ=√(2/(fan_in) = 0.078  => N(0,σ) = N(0,0.078)
# h3 =>  σ=√(2/(fan_in) = 0.126  => N(0,σ) = N(0,0.126)

from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout


model_relu = Sequential()
model_relu.add(Dense(580, activation='relu', input_shape=(input_dim,), kernel_initializer=Ran
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))

model_relu.add(Dense(325, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))

model_relu.add(Dense(125, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))

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

model_relu.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (None, 580)               455300
_________________________________________________________________
batch_normalization_2 (Batch (None, 580)               2320
_________________________________________________________________
dropout_1 (Dropout)          (None, 580)               0
_________________________________________________________________
dense_8 (Dense)              (None, 325)               188825
_________________________________________________________________
batch_normalization_3 (Batch (None, 325)               1300
_________________________________________________________________
dropout_2 (Dropout)          (None, 325)               0
_________________________________________________________________
dense_9 (Dense)              (None, 125)               40750
_________________________________________________________________
batch_normalization_4 (Batch (None, 125)               500
_________________________________________________________________
dropout_3 (Dropout)          (None, 125)               0
_________________________________________________________________
dense_10 (Dense)             (None, 10)                1260
=================================================================
Total params: 690,255
Trainable params: 688,195
Non-trainable params: 2,060
_________________________________________________________________
```

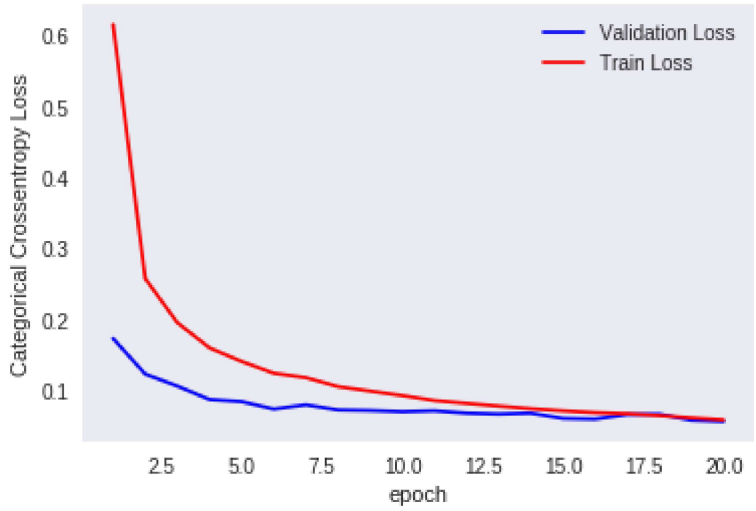
```
model_relu.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 14s 229us/step - loss: 0.6143 - acc: 0.
Epoch 2/20
60000/60000 [==============================] - 13s 209us/step - loss: 0.2568 - acc: 0.
Epoch 3/20
60000/60000 [==============================] - 12s 206us/step - loss: 0.1946 - acc: 0.
Epoch 4/20
60000/60000 [==============================] - 12s 207us/step - loss: 0.1592 - acc: 0.
Epoch 5/20
60000/60000 [==============================] - 12s 207us/step - loss: 0.1401 - acc: 0.
Epoch 6/20
60000/60000 [==============================] - 13s 209us/step - loss: 0.1235 - acc: 0.
Epoch 7/20
60000/60000 [==============================] - 13s 211us/step - loss: 0.1173 - acc: 0.
Epoch 8/20
60000/60000 [==============================] - 12s 204us/step - loss: 0.1047 - acc: 0.
Epoch 9/20
60000/60000 [==============================] - 12s 207us/step - loss: 0.0983 - acc: 0.
Epoch 10/20
60000/60000 [==============================] - 12s 205us/step - loss: 0.0920 - acc: 0.
Epoch 11/20
60000/60000 [==============================] - 12s 205us/step - loss: 0.0849 - acc: 0.
Epoch 12/20
60000/60000 [==============================] - 12s 205us/step - loss: 0.0810 - acc: 0.
Epoch 13/20
60000/60000 [==============================] - 12s 205us/step - loss: 0.0771 - acc: 0.
Epoch 14/20
60000/60000 [==============================] - 12s 204us/step - loss: 0.0735 - acc: 0.
Epoch 15/20
60000/60000 [==============================] - 13s 213us/step - loss: 0.0705 - acc: 0.
Epoch 16/20
60000/60000 [==============================] - 12s 205us/step - loss: 0.0679 - acc: 0.
Epoch 17/20
60000/60000 [==============================] - 12s 205us/step - loss: 0.0663 - acc: 0.
Epoch 18/20
60000/60000 [==============================] - 12s 207us/step - loss: 0.0638 - acc: 0.
Epoch 19/20
60000/60000 [==============================] - 12s 203us/step - loss: 0.0606 - acc: 0.
Epoch 20/20
60000/60000 [==============================] - 12s 205us/step - loss: 0.0578 - 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, verbose=1

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

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

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

```
Test score: 0.055547566109968464
Test accuracy: 0.9837
```



Input(784)+ReLu(658)+ReLu(525)+ReLu(410)+ReLu(350)+ReLu(150)+Output(10

```
# h1 =>  σ=√(2/(fan_in) = 0.055  => N(0,σ) = N(0,0.055)
# h2 =>  σ=√(2/(fan_in) = 0.061  => N(0,σ) = N(0,0.061)
# h3 =>  σ=√(2/(fan_in) = 0.069  => N(0,σ) = N(0,0.069)
# h4 =>  σ=√(2/(fan_in) = 0.075  => N(0,σ) = N(0,0.075)
# h5 =>  σ=√(2/(fan_in) = 0.115  => N(0,σ) = N(0,0.115)


model_relu = Sequential()
model_relu.add(Dense(658, activation='relu', input_shape=(input_dim,), kernel_initializer=Ran
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))

model_relu.add(Dense(525, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))

model_relu.add(Dense(410, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))

model_relu.add(Dense(350, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))

model_relu.add(Dense(150, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev
```

```
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))

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

model_relu.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_11 (Dense)             (None, 658)               516530
_________________________________________________________________
batch_normalization_5 (Batch (None, 658)               2632
_________________________________________________________________
dropout_4 (Dropout)          (None, 658)               0
_________________________________________________________________
dense_12 (Dense)             (None, 525)               345975
_________________________________________________________________
batch_normalization_6 (Batch (None, 525)               2100
_________________________________________________________________
dropout_5 (Dropout)          (None, 525)               0
_________________________________________________________________
dense_13 (Dense)             (None, 410)               215660
_________________________________________________________________
batch_normalization_7 (Batch (None, 410)               1640
_________________________________________________________________
dropout_6 (Dropout)          (None, 410)               0
_________________________________________________________________
dense_14 (Dense)             (None, 350)               143850
_________________________________________________________________
batch_normalization_8 (Batch (None, 350)               1400
_________________________________________________________________
dropout_7 (Dropout)          (None, 350)               0
_________________________________________________________________
dense_15 (Dense)             (None, 150)               52650
_________________________________________________________________
batch_normalization_9 (Batch (None, 150)               600
_________________________________________________________________
dropout_8 (Dropout)          (None, 150)               0
_________________________________________________________________
dense_16 (Dense)             (None, 10)                1510
=================================================================
Total params: 1,284,547
Trainable params: 1,280,361
Non-trainable params: 4,186
_________________________________________________________________
```

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

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 25s 412us/step - loss: 0.8993 - acc: 0.
Epoch 2/20
60000/60000 [==============================] - 23s 378us/step - loss: 0.3052 - acc: 0.
Epoch 3/20
60000/60000 [==============================] - 23s 379us/step - loss: 0.2283 - acc: 0.
Epoch 4/20
60000/60000 [==============================] - 23s 377us/step - loss: 0.1902 - acc: 0.
Epoch 5/20
60000/60000 [==============================] - 23s 379us/step - loss: 0.1588 - acc: 0.
Epoch 6/20
60000/60000 [==============================] - 23s 384us/step - loss: 0.1435 - acc: 0.
Epoch 7/20
60000/60000 [==============================] - 23s 381us/step - loss: 0.1310 - acc: 0.
Epoch 8/20
60000/60000 [==============================] - 23s 379us/step - loss: 0.1262 - acc: 0.
Epoch 9/20
60000/60000 [==============================] - 23s 381us/step - loss: 0.1143 - acc: 0.
Epoch 10/20
60000/60000 [==============================] - 23s 378us/step - loss: 0.1075 - acc: 0.
Epoch 11/20
60000/60000 [==============================] - 23s 378us/step - loss: 0.1022 - acc: 0.
Epoch 12/20
60000/60000 [==============================] - 23s 377us/step - loss: 0.0978 - acc: 0.
Epoch 13/20
60000/60000 [==============================] - 23s 378us/step - loss: 0.0929 - acc: 0.
Epoch 14/20
60000/60000 [==============================] - 23s 379us/step - loss: 0.0907 - acc: 0.
Epoch 15/20
60000/60000 [==============================] - 23s 378us/step - loss: 0.0828 - acc: 0.
Epoch 16/20
60000/60000 [==============================] - 23s 378us/step - loss: 0.0807 - acc: 0.
Epoch 17/20
```

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

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

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

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

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

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

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

```
Test score: 0.062496218884619884
Test accuracy: 0.9846
```