# various MLP architectures for MNIST

August 3, 2018

In [1]:
```
# if you keras is not using tensorflow as backend set "KERAS_BACKEND=tensorflow" use thi
from keras.utils import np_utils
from keras.datasets import mnist
import seaborn as sns
from keras.initializers import RandomNormal
```

```
/glob/intel-python/versions/2018u2/intelpython3/lib/python3.6/site-packages/h5py/__init__.py:34:
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

In [2]:
```
%matplotlib notebook
import matplotlib.pyplot as plt
import numpy as np
import time
def plt_dynamic(x, vy, ty, ax, colors=['b']):
    ax.plot(x, vy, 'b', label="Validation Loss")
    ax.plot(x, ty, 'r', label="Train Loss")
    plt.legend()
    plt.grid()
    fig.canvas.draw()
```

In [3]:
```
# the data, shuffled and split between train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

In [4]:
```
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)
```

In [5]:
```
# 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])
```

In [6]:
```
# after converting the input images from 3d to 2d vectors

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

In [7]:
```
# if we observe the above matrix each cell is having a value between 0-255
# before we move to apply machine learning algorithms lets try to normalize the data
# X => (X - Xmin)/(Xmax-Xmin) = X/255

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

In [8]:
```
# 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.]
```

In [9]:
```
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout
from keras.layers import Input, Dense
from keras.models import Model
from keras import optimizers
from keras.layers import Activation
```

In [10]:
```
import tensorflow as tf
```

### 0.0.1 2 layer network

**784 - 392 - 64 - 10 with all relu and softmax output -- Keras**

In [32]:
```
# input
inputs = Input(shape=(784,))
```

```
        # hidden layer1
        x = Dense(392, activation='relu')(inputs)
        #hidden layer 2
        x = Dense(64, activation='relu')(x)
        #output
        out = Dense(10, activation='softmax')(x)
        #model
        model = Model(inputs=inputs, outputs=out)
        print(model.summary())
        adam = optimizers.Adam(lr=0.0005, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, am
        #compile
        model.compile(optimizer=adam, loss='categorical_crossentropy', metrics=['accuracy'])
        #training
        history = model.fit(X_train, Y_train, batch_size=128, epochs=20, verbose=1, validation_
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_10 (InputLayer)        (None, 784)               0
_________________________________________________________________
dense_28 (Dense)             (None, 392)               307720
_________________________________________________________________
dense_29 (Dense)             (None, 64)                25152
_________________________________________________________________
dense_30 (Dense)             (None, 10)                650
=================================================================
Total params: 333,522
Trainable params: 333,522
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.3310 - acc: 0.9113 - val_l
Epoch 2/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.1306 - acc: 0.9626 - val_l
Epoch 3/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0864 - acc: 0.9750 - val_l
Epoch 4/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0632 - acc: 0.9820 - val_l
Epoch 5/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0480 - acc: 0.9855 - val_l
Epoch 6/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0372 - acc: 0.9892 - val_l
Epoch 7/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0287 - acc: 0.9921 - val_l
Epoch 8/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0213 - acc: 0.9944 - val_l
```

```
Epoch 9/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0183 - acc: 0.9948 - val_l
Epoch 10/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0136 - acc: 0.9966 - val_l
Epoch 11/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.0102 - acc: 0.9977 - val_l
Epoch 12/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0086 - acc: 0.9980 - val_l
Epoch 13/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0075 - acc: 0.9982 - val_l
Epoch 14/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.0084 - acc: 0.9976 - val_l
Epoch 15/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.0065 - acc: 0.9983 - val_l
Epoch 16/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0035 - acc: 0.9994 - val_l
Epoch 17/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.0021 - acc: 0.9996 - val_l
Epoch 18/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.0062 - acc: 0.9983 - val_l
Epoch 19/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.0081 - acc: 0.9972 - val_l
Epoch 20/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.0052 - acc: 0.9985 - val_l
```

```
In [33]: score = model.evaluate(X_test, Y_test, verbose=0)
         print('Test score:', score[0])
         print('Test accuracy:', score[1])

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

         # list of epoch numbers
         x = list(range(1,21))

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

```
Test score: 0.07724876616429055
Test accuracy: 0.9811
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

**784 - 392 - 64 - 10 with all relu and softmax output -- tensorflow**

In [11]:
```
### Creating place holders
tf.reset_default_graph()
with tf.name_scope('place_holders'):
    x = tf.placeholder(tf.float32, [None, 784],name = 'x_input_data')
    y_true = tf.placeholder(tf.float32, [None, 10],name = 'y_labled_true_data')
with tf.name_scope('Weights'):
    weights = {
        'w1': tf.get_variable('w1',shape=[784,392],initializer=tf.contrib.layers.xavier
        'w2': tf.get_variable('w2',shape=[392,64],initializer=tf.contrib.layers.xavier_
        'w3': tf.get_variable('w3',shape=[64,10],initializer=tf.contrib.layers.xavier_i
    }
    biases = {
    'b1': tf.get_variable('b1',shape=[392],initializer=tf.zeros_initializer()),
    'b2': tf.get_variable('b2',shape=[64],initializer=tf.zeros_initializer()),
    'b3': tf.get_variable('b3',shape=[10],initializer=tf.zeros_initializer())
    }

with tf.name_scope('layer1'):
    layer_1 = tf.add(tf.matmul(x, weights['w1']), biases['b1'],name = 'Dense1')
    layer_1 = tf.nn.relu(layer_1,name='RActivation1')
with tf.name_scope('layer2'):
    layer_2 = tf.add(tf.matmul(layer_1, weights['w2']), biases['b2'],name = 'Dense2')
    layer_2 = tf.nn.relu(layer_2,name='RActivation2')
with tf.name_scope('output'):
    out = tf.add(tf.matmul(layer_2, weights['w3']), biases['b3'],name = 'output')
    out = tf.nn.softmax(out,name='Softmax')
with tf.name_scope('cost'):
    cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits = out, labe
with tf.name_scope('backprop'):
    optimizer_adam = tf.train.AdamOptimizer(learning_rate=0.0005).minimize(cost)
with tf.name_scope('accuracy'):
    correct_prediction = tf.equal(tf.argmax(out,1), tf.argmax(y_true,1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

# create the training datasets
dx_train = tf.data.Dataset.from_tensor_slices(X_train)
dy_train = tf.data.Dataset.from_tensor_slices(Y_train)
# zip the x and y training data together and shuffle, batch etc.
train_dataset = tf.data.Dataset.zip((dx_train, dy_train)).shuffle(500).batch(128)
#iterator
iterator = train_dataset.make_initializable_iterator()
#get next element
next_element = iterator.get_next()
display_step = 1
with tf.Session() as sess:
    tf.global_variables_initializer().run()
```

```
            xs, ytrs, ytes = [], [], []
            train_acc = 0.0
            test_acc = 0.0
            for epoch in range(20):
                train_avg_cost = 0.0
                test_avg_cost = 0.0
                total_batch = int(X_train.shape[0]/128)
                sess.run(iterator.initializer)
                for i in range(total_batch):
                    batch_xs, batch_ys = sess.run(next_element)
                    _, c, w = sess.run([optimizer_adam, cost,weights], feed_dict={x: batch_xs,
                    train_avg_cost += c / total_batch
                    c = sess.run(cost, feed_dict={x: X_test, y_true: Y_test})
                    test_avg_cost += c / total_batch
                train_acc = (sess.run([accuracy],feed_dict={x:X_train, y_true: Y_train}))[0]
                test_acc = (sess.run([accuracy],feed_dict={x:X_test, y_true: Y_test}))[0]
                xs.append(epoch)
                ytrs.append(train_avg_cost)
                ytes.append(test_avg_cost)
                if epoch%display_step == 0:
                    print("Epoch:", '%04d' % (epoch+1), "cost={:.9f}".format(train_avg_cost),"a
                                                        "test cost={:.9f}".format(test_avg_cost),"te
```

```
Epoch: 0001 cost=1.665115213 acc=0.863699973 test cost=1.657961059 test acc=0.864000022
Epoch: 0002 cost=1.553161122 acc=0.955449998 test cost=1.552255132 test acc=0.952400029
Epoch: 0003 cost=1.506821749 acc=0.968283355 test cost=1.508401569 test acc=0.963100016
Epoch: 0004 cost=1.495825695 acc=0.975149989 test cost=1.500182503 test acc=0.967400014
Epoch: 0005 cost=1.490031286 acc=0.979099989 test cost=1.496287779 test acc=0.970399976
Epoch: 0006 cost=1.485262905 acc=0.980199993 test cost=1.493258562 test acc=0.969500005
Epoch: 0007 cost=1.481818107 acc=0.982616663 test cost=1.491085784 test acc=0.970200002
Epoch: 0008 cost=1.479243853 acc=0.986783326 test cost=1.489379456 test acc=0.974500000
Epoch: 0009 cost=1.477010255 acc=0.986616671 test cost=1.488232427 test acc=0.974799991
Epoch: 0010 cost=1.475809584 acc=0.986800015 test cost=1.487429235 test acc=0.972500026
Epoch: 0011 cost=1.474063572 acc=0.987299979 test cost=1.486677387 test acc=0.971899986
Epoch: 0012 cost=1.473106978 acc=0.990249991 test cost=1.486154201 test acc=0.973900020
Epoch: 0013 cost=1.472180222 acc=0.991316676 test cost=1.485470061 test acc=0.976800025
Epoch: 0014 cost=1.471389278 acc=0.990566671 test cost=1.484757890 test acc=0.975899994
Epoch: 0015 cost=1.470774428 acc=0.991866648 test cost=1.484183233 test acc=0.978900015
Epoch: 0016 cost=1.470356798 acc=0.992483318 test cost=1.484162278 test acc=0.978100002
Epoch: 0017 cost=1.469466509 acc=0.990700006 test cost=1.483700292 test acc=0.976100028
Epoch: 0018 cost=1.469017527 acc=0.992983341 test cost=1.483824379 test acc=0.977699995
Epoch: 0019 cost=1.469232037 acc=0.992399991 test cost=1.483476739 test acc=0.977699995
Epoch: 0020 cost=1.468447580 acc=0.993816674 test cost=1.483311839 test acc=0.979399979
```

```
In [13]: fig,ax = plt.subplots(1,1)
         ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')
         plt_dynamic(xs, ytes, ytrs, ax)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

### 784 - 392 - 64 - 10 with all relu, Batchnorm on hidden layes and softmax output - keras

```
In [36]: # input
         inputs = Input(shape=(784,),name='Input_layer')
         # hidden layer 1
         x = Dense(392,name='Dlayer1')(inputs)
         x = BatchNormalization(name='Blayer1')(x)
         x = Activation('relu',name='Alayer1')(x)
         #hidden layer2
         x = Dense(64,name='Dlayer2')(x)
         x = BatchNormalization(name='Blayer2')(x)
         x = Activation('relu',name='Alayer2')(x)
         #out
         out = Dense(10, activation='softmax',name='Output_layer')(x)
         #model
         model = Model(inputs=inputs, outputs=out)
         print(model.summary())
         adam = optimizers.Adam(lr=0.0005, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, am
         #compile
         model.compile(optimizer=adam, loss='categorical_crossentropy', metrics=['accuracy'])
         #training
         history = model.fit(X_train, Y_train, batch_size=128, epochs=20, verbose=1, validation_
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input_layer (InputLayer)     (None, 784)               0
_________________________________________________________________
Dlayer1 (Dense)              (None, 392)               307720
_________________________________________________________________
Blayer1 (BatchNormalization) (None, 392)               1568
_________________________________________________________________
Alayer1 (Activation)         (None, 392)               0
_________________________________________________________________
Dlayer2 (Dense)              (None, 64)                25152
_________________________________________________________________
Blayer2 (BatchNormalization) (None, 64)                256
_________________________________________________________________
Alayer2 (Activation)         (None, 64)                0
_________________________________________________________________
Output_layer (Dense)         (None, 10)                650
=================================================================
```

```
Total params: 335,346
Trainable params: 334,434
Non-trainable params: 912
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 6s 99us/step - loss: 0.2926 - acc: 0.9254 - val_l
Epoch 2/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0977 - acc: 0.9737 - val_l
Epoch 3/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0588 - acc: 0.9847 - val_l
Epoch 4/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0384 - acc: 0.9904 - val_l
Epoch 5/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0266 - acc: 0.9932 - val_l
Epoch 6/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0200 - acc: 0.9949 - val_l
Epoch 7/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0154 - acc: 0.9962 - val_l
Epoch 8/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0119 - acc: 0.9973 - val_l
Epoch 9/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0128 - acc: 0.9966 - val_l
Epoch 10/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0099 - acc: 0.9975 - val_l
Epoch 11/20
60000/60000 [==============================] - 5s 87us/step - loss: 0.0091 - acc: 0.9978 - val_l
Epoch 12/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0059 - acc: 0.9986 - val_l
Epoch 13/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0079 - acc: 0.9978 - val_l
Epoch 14/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0100 - acc: 0.9971 - val_l
Epoch 15/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0050 - acc: 0.9987 - val_l
Epoch 16/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0071 - acc: 0.9979 - val_l
Epoch 17/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0067 - acc: 0.9981 - val_l
Epoch 18/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0062 - acc: 0.9983 - val_l
Epoch 19/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0041 - acc: 0.9990 - val_l
Epoch 20/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0045 - acc: 0.9987 - val_l
```

```
In [37]: score = model.evaluate(X_test, Y_test, verbose=0)
         print('Test score:', score[0])
         print('Test accuracy:', score[1])

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

         # list of epoch numbers
         x = list(range(1,21))

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

```
Test score: 0.08478823226970271
Test accuracy: 0.9789
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

**784 - 392 - 64 - 10 with all relu, Batchnorm on idden layes and softmax output - Tensorflow**

```
In [15]: ### Creating place holders
         tf.reset_default_graph()
         epsilon = 1e-3
         with tf.name_scope('place_holders'):
             x = tf.placeholder(tf.float32, [None, 784],name = 'x_input_data')
             y_true = tf.placeholder(tf.float32, [None, 10],name = 'y_labled_true_data')
         with tf.name_scope('Weights'):
             weights = {
                 'w1': tf.get_variable('w1',shape=[784,392],initializer=tf.contrib.layers.xavier
                 'w2': tf.get_variable('w2',shape=[392,64],initializer=tf.contrib.layers.xavier_
                 'w3': tf.get_variable('w3',shape=[64,10],initializer=tf.contrib.layers.xavier_i
             }
             biases = {
             'b1': tf.get_variable('b1',shape=[392],initializer=tf.zeros_initializer()),
             'b2': tf.get_variable('b2',shape=[64],initializer=tf.zeros_initializer()),
             'b3': tf.get_variable('b3',shape=[10],initializer=tf.zeros_initializer())
             }

         with tf.name_scope('layer1'):
             layer_1 = tf.add(tf.matmul(x, weights['w1']), biases['b1'],name = 'Dense1')
             batch_mean_1, batch_var_1 = tf.nn.moments(layer_1,[0])
             scale_1 = tf.Variable(tf.ones([392]))
```

```
    beta_1 = tf.Variable(tf.zeros([392]))
    layer_1 = tf.nn.batch_normalization(layer_1, batch_mean_1, batch_var_1, beta_1, sca
    layer_1 = tf.nn.relu(layer_1,name='RActivation1')
with tf.name_scope('layer2'):
    layer_2 = tf.add(tf.matmul(layer_1, weights['w2']), biases['b2'],name = 'Dense2')
    batch_mean_2, batch_var_2 = tf.nn.moments(layer_2,[0])
    scale_2 = tf.Variable(tf.ones([64]))
    beta_2 = tf.Variable(tf.zeros([64]))
    layer_2 = tf.nn.batch_normalization(layer_2, batch_mean_2, batch_var_2, beta_2, sca
    layer_2 = tf.nn.relu(layer_2,name='RActivation2')
with tf.name_scope('output'):
    out = tf.add(tf.matmul(layer_2, weights['w3']), biases['b3'],name = 'output')
    out = tf.nn.softmax(out,name='Softmax')
with tf.name_scope('cost'):
    cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits = out, labe
with tf.name_scope('backprop'):
    optimizer_adam = tf.train.AdamOptimizer(learning_rate=0.0005).minimize(cost)
with tf.name_scope('accuracy'):
    correct_prediction = tf.equal(tf.argmax(out,1), tf.argmax(y_true,1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

# create the training datasets
dx_train = tf.data.Dataset.from_tensor_slices(X_train)
dy_train = tf.data.Dataset.from_tensor_slices(Y_train)
# zip the x and y training data together and shuffle, batch etc.
train_dataset = tf.data.Dataset.zip((dx_train, dy_train)).shuffle(500).batch(128)
#iterator
iterator = train_dataset.make_initializable_iterator()
#get next element
next_element = iterator.get_next()
display_step = 1
with tf.Session() as sess:
    tf.global_variables_initializer().run()
    xs, ytrs, ytes = [], [], []
    train_acc = 0.0
    test_acc = 0.0
    for epoch in range(20):
        train_avg_cost = 0.0
        test_avg_cost = 0.0
        total_batch = int(X_train.shape[0]/128)
        sess.run(iterator.initializer)
        for i in range(total_batch):
            batch_xs, batch_ys = sess.run(next_element)
            _, c, w = sess.run([optimizer_adam, cost,weights], feed_dict={x: batch_xs,
            train_avg_cost += c / total_batch
            c = sess.run(cost, feed_dict={x: X_test, y_true: Y_test})
            test_avg_cost += c / total_batch
        train_acc = (sess.run([accuracy],feed_dict={x:X_train, y_true: Y_train}))[0]
```

```
                test_acc = (sess.run([accuracy],feed_dict={x:X_test, y_true: Y_test}))[0]
                xs.append(epoch)
                ytrs.append(train_avg_cost)
                ytes.append(test_avg_cost)
                if epoch%display_step == 0:
                    print("Epoch:", '%04d' % (epoch+1), "cost={:.9f}".format(train_avg_cost),"a
                                                        "test cost={:.9f}".format(test_avg_cost),"te
        fig,ax = plt.subplots(1,1)
        ax.set_xlabel('epoch') ; ax.set_ylabel('Softmax Crossentropy Loss')
        plt_dynamic(xs, ytes, ytrs, ax)
```

```
Epoch: 0001 cost=1.591900543 acc=0.966816664 test cost=1.582991538 test acc=0.961899996
Epoch: 0002 cost=1.504018855 acc=0.980383337 test cost=1.502923975 test acc=0.972000003
Epoch: 0003 cost=1.489843388 acc=0.985450029 test cost=1.493284358 test acc=0.974500000
Epoch: 0004 cost=1.482457173 acc=0.988499999 test cost=1.489334382 test acc=0.977400005
Epoch: 0005 cost=1.478255027 acc=0.990999997 test cost=1.486667594 test acc=0.978100002
Epoch: 0006 cost=1.474952367 acc=0.991216660 test cost=1.485367718 test acc=0.977199972
Epoch: 0007 cost=1.472775489 acc=0.993533313 test cost=1.484200543 test acc=0.979300022
Epoch: 0008 cost=1.471633176 acc=0.994183362 test cost=1.483711184 test acc=0.981000006
Epoch: 0009 cost=1.470122601 acc=0.994816661 test cost=1.482396444 test acc=0.980799973
Epoch: 0010 cost=1.469446362 acc=0.994700015 test cost=1.482001757 test acc=0.981000006
Epoch: 0011 cost=1.468605584 acc=0.995999992 test cost=1.481875223 test acc=0.980400026
Epoch: 0012 cost=1.467851525 acc=0.995949984 test cost=1.481454936 test acc=0.979900002
Epoch: 0013 cost=1.467832543 acc=0.995883346 test cost=1.481672273 test acc=0.981100023
Epoch: 0014 cost=1.467221644 acc=0.996233344 test cost=1.480686287 test acc=0.980499983
Epoch: 0015 cost=1.466710021 acc=0.996433318 test cost=1.480526888 test acc=0.980300009
Epoch: 0016 cost=1.466327307 acc=0.996500015 test cost=1.480685591 test acc=0.980899990
Epoch: 0017 cost=1.466502902 acc=0.996416688 test cost=1.481014947 test acc=0.981299996
Epoch: 0018 cost=1.466187922 acc=0.996866643 test cost=1.480490149 test acc=0.981000006
Epoch: 0019 cost=1.465592796 acc=0.997250021 test cost=1.480371399 test acc=0.982200027
Epoch: 0020 cost=1.465670914 acc=0.996666670 test cost=1.480636943 test acc=0.980400026
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

### 784 - 392 - 64 - 10 with all relu ,Batchnorm and dropout on hidden layes and softmax output - Keras

```
In [40]: # input
         inputs = Input(shape=(784,),name='Input_layer')
         # hidden layer 1
         x = Dense(392,name='Dlayer1')(inputs)
         x = BatchNormalization(name='Blayer1')(x)
         x = Activation('relu',name='Alayer1')(x)
```

```
        x = Dropout(rate=0.4,name='Drlayer1')(x)
        #hidden layer2
        x = Dense(64,name='Dlayer2')(x)
        x = BatchNormalization(name='Blayer2')(x)
        x = Activation('relu',name='Alayer2')(x)
        x = Dropout(rate=0.4,name='Drlayer2')(x)
        #out
        out = Dense(10, activation='softmax',name='Output_layer')(x)
        #model
        model = Model(inputs=inputs, outputs=out)
        print(model.summary())
        adam = optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, ams
        #compile
        model.compile(optimizer=adam, loss='categorical_crossentropy', metrics=['accuracy'])
        #training
        history = model.fit(X_train, Y_train, batch_size=128, epochs=32, verbose=1, validation_
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input_layer (InputLayer)     (None, 784)               0
_________________________________________________________________
Dlayer1 (Dense)              (None, 392)               307720
_________________________________________________________________
Blayer1 (BatchNormalization) (None, 392)               1568
_________________________________________________________________
Alayer1 (Activation)         (None, 392)               0
_________________________________________________________________
Drlayer1 (Dropout)           (None, 392)               0
_________________________________________________________________
Dlayer2 (Dense)              (None, 64)                25152
_________________________________________________________________
Blayer2 (BatchNormalization) (None, 64)                256
_________________________________________________________________
Alayer2 (Activation)         (None, 64)                0
_________________________________________________________________
Drlayer2 (Dropout)           (None, 64)                0
_________________________________________________________________
Output_layer (Dense)         (None, 10)                650
=================================================================
Total params: 335,346
Trainable params: 334,434
Non-trainable params: 912
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/32
60000/60000 [==============================] - 7s 113us/step - loss: 0.4604 - acc: 0.8700 - val_
```

```
Epoch 2/32
60000/60000 [==============================] - 6s 97us/step - loss: 0.2308 - acc: 0.9335 - val_l
Epoch 3/32
60000/60000 [==============================] - 6s 97us/step - loss: 0.1809 - acc: 0.9471 - val_l
Epoch 4/32
60000/60000 [==============================] - 6s 97us/step - loss: 0.1475 - acc: 0.9552 - val_l
Epoch 5/32
60000/60000 [==============================] - 6s 97us/step - loss: 0.1303 - acc: 0.9619 - val_l
Epoch 6/32
60000/60000 [==============================] - 6s 97us/step - loss: 0.1157 - acc: 0.9650 - val_l
Epoch 7/32
60000/60000 [==============================] - 6s 97us/step - loss: 0.1056 - acc: 0.9670 - val_l
Epoch 8/32
60000/60000 [==============================] - 6s 97us/step - loss: 0.0975 - acc: 0.9700 - val_l
Epoch 9/32
60000/60000 [==============================] - 6s 98us/step - loss: 0.0926 - acc: 0.9715 - val_l
Epoch 10/32
60000/60000 [==============================] - 6s 98us/step - loss: 0.0890 - acc: 0.9724 - val_l
Epoch 11/32
60000/60000 [==============================] - 6s 97us/step - loss: 0.0808 - acc: 0.9747 - val_l
Epoch 12/32
60000/60000 [==============================] - 6s 97us/step - loss: 0.0753 - acc: 0.9760 - val_l
Epoch 13/32
60000/60000 [==============================] - 6s 97us/step - loss: 0.0727 - acc: 0.9779 - val_l
Epoch 14/32
60000/60000 [==============================] - 6s 97us/step - loss: 0.0695 - acc: 0.9781 - val_l
Epoch 15/32
60000/60000 [==============================] - 6s 97us/step - loss: 0.0651 - acc: 0.9797 - val_l
Epoch 16/32
60000/60000 [==============================] - 6s 97us/step - loss: 0.0623 - acc: 0.9811 - val_l
Epoch 17/32
60000/60000 [==============================] - 6s 97us/step - loss: 0.0603 - acc: 0.9807 - val_l
Epoch 18/32
60000/60000 [==============================] - 6s 97us/step - loss: 0.0588 - acc: 0.9811 - val_l
Epoch 19/32
60000/60000 [==============================] - 6s 97us/step - loss: 0.0563 - acc: 0.9821 - val_l
Epoch 20/32
60000/60000 [==============================] - 6s 97us/step - loss: 0.0523 - acc: 0.9833 - val_l
Epoch 21/32
60000/60000 [==============================] - 6s 97us/step - loss: 0.0499 - acc: 0.9838 - val_l
Epoch 22/32
60000/60000 [==============================] - 6s 97us/step - loss: 0.0503 - acc: 0.9843 - val_l
Epoch 23/32
60000/60000 [==============================] - 6s 97us/step - loss: 0.0472 - acc: 0.9848 - val_l
Epoch 24/32
60000/60000 [==============================] - 6s 97us/step - loss: 0.0464 - acc: 0.9850 - val_l
Epoch 25/32
60000/60000 [==============================] - 6s 97us/step - loss: 0.0459 - acc: 0.9853 - val_l
```

```
Epoch 26/32
60000/60000 [==============================] - 6s 97us/step - loss: 0.0453 - acc: 0.9861 - val_l
Epoch 27/32
60000/60000 [==============================] - 6s 97us/step - loss: 0.0411 - acc: 0.9864 - val_l
Epoch 28/32
60000/60000 [==============================] - 6s 97us/step - loss: 0.0445 - acc: 0.9860 - val_l
Epoch 29/32
60000/60000 [==============================] - 6s 97us/step - loss: 0.0388 - acc: 0.9871 - val_l
Epoch 30/32
60000/60000 [==============================] - 6s 98us/step - loss: 0.0407 - acc: 0.9869 - val_l
Epoch 31/32
60000/60000 [==============================] - 6s 97us/step - loss: 0.0389 - acc: 0.9875 - val_l
Epoch 32/32
60000/60000 [==============================] - 6s 97us/step - loss: 0.0370 - acc: 0.9879 - val_l
```

```
In [41]: score = model.evaluate(X_test, Y_test, verbose=0)
         print('Test score:', score[0])
         print('Test accuracy:', score[1])

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

         # list of epoch numbers
         x = list(range(1,33))

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

```
Test score: 0.0590697706670966
Test accuracy: 0.9843
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

**784 - 392 - 64 - 10 with all relu ,Batchnorm and dropout on hidden layes and softmax output - Tensorflow**

```
In [19]: ### Creating place holders
         tf.reset_default_graph()
         epsilon = 1e-3

         with tf.name_scope('place_holders'):
             x = tf.placeholder(tf.float32, [None, 784],name = 'x_input_data')
```

```
    y_true = tf.placeholder(tf.float32, [None, 10],name = 'y_labled_true_data')
with tf.name_scope('Weights'):
    weights = {
        'w1': tf.get_variable('w1',shape=[784,392],initializer=tf.contrib.layers.xavier
        'w2': tf.get_variable('w2',shape=[392,64],initializer=tf.contrib.layers.xavier_
        'w3': tf.get_variable('w3',shape=[64,10],initializer=tf.contrib.layers.xavier_i
    }
    biases = {
    'b1': tf.get_variable('b1',shape=[392],initializer=tf.zeros_initializer()),
    'b2': tf.get_variable('b2',shape=[64],initializer=tf.zeros_initializer()),
    'b3': tf.get_variable('b3',shape=[10],initializer=tf.zeros_initializer())
    }

with tf.name_scope('layer1'):
    layer_1 = tf.add(tf.matmul(x, weights['w1']), biases['b1'],name = 'Dense1')
    batch_mean_1, batch_var_1 = tf.nn.moments(layer_1,[0])
    scale_1 = tf.Variable(tf.ones([392]))
    beta_1 = tf.Variable(tf.zeros([392]))
    layer_1 = tf.nn.batch_normalization(layer_1, batch_mean_1, batch_var_1, beta_1, sca
    layer_1 = tf.nn.relu(layer_1,name='RActivation1')
    layer_1 = tf.nn.dropout(layer_1,0.4,name='Dropout1')
    tf.summary.histogram('activations', layer_1)
with tf.name_scope('layer2'):
    layer_2 = tf.add(tf.matmul(layer_1, weights['w2']), biases['b2'],name = 'Dense2')
    batch_mean_2, batch_var_2 = tf.nn.moments(layer_2,[0])
    scale_2 = tf.Variable(tf.ones([64]))
    beta_2 = tf.Variable(tf.zeros([64]))
    layer_2 = tf.nn.batch_normalization(layer_2, batch_mean_2, batch_var_2, beta_2, sca
    layer_2 = tf.nn.relu(layer_2,name='RActivation2')
    layer_2 = tf.nn.dropout(layer_2,0.4,name='Dropout2')
    tf.summary.histogram('activations', layer_2)
with tf.name_scope('output'):
    out = tf.add(tf.matmul(layer_2, weights['w3']), biases['b3'],name = 'output')
    out = tf.nn.softmax(out,name='Softmax')
with tf.name_scope('cost'):
    cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits = out, labe
tf.summary.scalar('Cost', cost)
with tf.name_scope('backprop'):
    optimizer_adam = tf.train.AdamOptimizer(learning_rate=0.001).minimize(cost)
with tf.name_scope('accuracy'):
    correct_prediction = tf.equal(tf.argmax(out,1), tf.argmax(y_true,1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    tf.summary.scalar('Accuracy', accuracy)

# create the training datasets
dx_train = tf.data.Dataset.from_tensor_slices(X_train)
dy_train = tf.data.Dataset.from_tensor_slices(Y_train)
# zip the x and y training data together and shuffle, batch etc.
```

```
        train_dataset = tf.data.Dataset.zip((dx_train, dy_train)).shuffle(500).batch(128)
        #iterator
        iterator = train_dataset.make_initializable_iterator()
        #get next element
        next_element = iterator.get_next()

        merged = tf.summary.merge_all()

        display_step = 1
        with tf.Session() as sess:
            tf.global_variables_initializer().run()
            #writing
            train_writer = tf.summary.FileWriter('graph_2layer3_train',sess.graph)
            test_writer = tf.summary.FileWriter('graph_2layer3_test')
            xs, ytrs, ytes = [], [], []
            train_acc = 0.0
            test_acc = 0.0
            for epoch in range(35):
                train_avg_cost = 0.0
                test_avg_cost = 0.0
                total_batch = int(X_train.shape[0]/128)
                sess.run(iterator.initializer)
                for i in range(total_batch):
                    batch_xs, batch_ys = sess.run(next_element)
                    _, c, w = sess.run([optimizer_adam, cost,weights], feed_dict={x: batch_xs,
                    train_avg_cost += c / total_batch
                    c = sess.run(cost, feed_dict={x: X_test, y_true: Y_test})
                    test_avg_cost += c / total_batch
                train_acc,summ = (sess.run([accuracy,merged],feed_dict={x:X_train, y_true: Y_tr
                train_writer.add_summary(summ, epoch)
                test_acc,summ = (sess.run([accuracy,merged],feed_dict={x:X_test, y_true: Y_test
                test_writer.add_summary(summ, epoch)

                xs.append(epoch)
                ytrs.append(train_avg_cost)
                ytes.append(test_avg_cost)
                if epoch%display_step == 0:
                    print("Epoch:", '%04d' % (epoch+1), "cost={:.9f}".format(train_avg_cost),"a
                                                        "test cost={:.9f}".format(test_avg_cost),"te
        fig,ax = plt.subplots(1,1)
        ax.set_xlabel('epoch') ; ax.set_ylabel('Softmax Crossentropy Loss')
        plt_dynamic(xs, ytes, ytrs, ax)
```

```
Epoch: 0001 cost=1.731791468 acc=0.876433313 test cost=1.720432235 test acc=0.881900012
Epoch: 0002 cost=1.594874881 acc=0.899550021 test cost=1.587469975 test acc=0.900200009
Epoch: 0003 cost=1.570556228 acc=0.915566683 test cost=1.563623525 test acc=0.914300025
Epoch: 0004 cost=1.557318421 acc=0.922883332 test cost=1.551386922 test acc=0.922900021
Epoch: 0005 cost=1.546317148 acc=0.929516673 test cost=1.543476427 test acc=0.929600000
```

```
Epoch: 0006 cost=1.540039581 acc=0.933866680 test cost=1.537321969 test acc=0.932600021
Epoch: 0007 cost=1.535750647 acc=0.938883305 test cost=1.532651953 test acc=0.937200010
Epoch: 0008 cost=1.530833601 acc=0.939883351 test cost=1.529633949 test acc=0.936399996
Epoch: 0009 cost=1.528817173 acc=0.943633318 test cost=1.526847145 test acc=0.939300001
Epoch: 0010 cost=1.526183478 acc=0.946116686 test cost=1.523876807 test acc=0.941100001
Epoch: 0011 cost=1.522906059 acc=0.948650002 test cost=1.522001660 test acc=0.940500021
Epoch: 0012 cost=1.521075729 acc=0.950100005 test cost=1.519879014 test acc=0.944500029
Epoch: 0013 cost=1.518931892 acc=0.951200008 test cost=1.518086605 test acc=0.945699990
Epoch: 0014 cost=1.516853496 acc=0.954100013 test cost=1.517055204 test acc=0.945400000
Epoch: 0015 cost=1.514760702 acc=0.954583347 test cost=1.515320760 test acc=0.947600007
Epoch: 0016 cost=1.515990383 acc=0.955833316 test cost=1.514777072 test acc=0.949199975
Epoch: 0017 cost=1.512584188 acc=0.956250012 test cost=1.513634486 test acc=0.947300017
Epoch: 0018 cost=1.510994484 acc=0.956683338 test cost=1.512367457 test acc=0.953100026
Epoch: 0019 cost=1.510604214 acc=0.959616661 test cost=1.511501221 test acc=0.953100026
Epoch: 0020 cost=1.509356168 acc=0.959800005 test cost=1.510266374 test acc=0.950900018
Epoch: 0021 cost=1.509201920 acc=0.959800005 test cost=1.510043586 test acc=0.952499986
Epoch: 0022 cost=1.508768132 acc=0.961816669 test cost=1.509550504 test acc=0.952099979
Epoch: 0023 cost=1.507526323 acc=0.961566687 test cost=1.509230568 test acc=0.952899992
Epoch: 0024 cost=1.507512023 acc=0.961316645 test cost=1.508298181 test acc=0.953899980
Epoch: 0025 cost=1.506277260 acc=0.962849975 test cost=1.507925041 test acc=0.952700019
Epoch: 0026 cost=1.505088601 acc=0.963083327 test cost=1.507867122 test acc=0.955399990
Epoch: 0027 cost=1.504501718 acc=0.963549972 test cost=1.507384900 test acc=0.957199991
Epoch: 0028 cost=1.503900969 acc=0.964166641 test cost=1.507170258 test acc=0.955399990
Epoch: 0029 cost=1.504030114 acc=0.964766681 test cost=1.506718311 test acc=0.957599998
Epoch: 0030 cost=1.502311161 acc=0.965566695 test cost=1.505914480 test acc=0.957599998
Epoch: 0031 cost=1.503058030 acc=0.965416670 test cost=1.505317406 test acc=0.955799997
Epoch: 0032 cost=1.502071810 acc=0.966050029 test cost=1.505280256 test acc=0.958000004
Epoch: 0033 cost=1.501793542 acc=0.965049982 test cost=1.504613776 test acc=0.959800005
Epoch: 0034 cost=1.500869437 acc=0.967299998 test cost=1.504613255 test acc=0.956799984
Epoch: 0035 cost=1.501697483 acc=0.966083348 test cost=1.504886840 test acc=0.957400024


<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>
```

#### 0.0.2 3 layer network

**784 - 512 - 256 - 128 - 10 with all relu on hidden layes and softmax output**

```
In [42]: # input
         inputs = Input(shape=(784,),name='Input_layer')
         # hidden layer 1
         x = Dense(512,name='Dlayer1')(inputs)
         x = Activation('relu',name='Alayer1')(x)
         #hidden layer2
         x = Dense(256,name='Dlayer2')(x)
```

```
        x = Activation('relu',name='Alayer2')(x)
        #hidden layer3
        x = Dense(128,name='Dlayer3')(x)
        x = Activation('relu',name='Alayer3')(x)
        #out
        out = Dense(10, activation='softmax',name='Output_layer')(x)
        #model
        model = Model(inputs=inputs, outputs=out)
        print(model.summary())
        adam = optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, ams
        #compile
        model.compile(optimizer=adam, loss='categorical_crossentropy', metrics=['accuracy'])
        #training
        history = model.fit(X_train, Y_train, batch_size=128, epochs=25, verbose=1, validation_
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input_layer (InputLayer)     (None, 784)               0
_________________________________________________________________
Dlayer1 (Dense)              (None, 512)               401920
_________________________________________________________________
Alayer1 (Activation)         (None, 512)               0
_________________________________________________________________
Dlayer2 (Dense)              (None, 256)               131328
_________________________________________________________________
Alayer2 (Activation)         (None, 256)               0
_________________________________________________________________
Dlayer3 (Dense)              (None, 128)               32896
_________________________________________________________________
Alayer3 (Activation)         (None, 128)               0
_________________________________________________________________
Output_layer (Dense)         (None, 10)                1290
=================================================================
Total params: 567,434
Trainable params: 567,434
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/25
60000/60000 [==============================] - 7s 109us/step - loss: 0.2322 - acc: 0.9309 - val_
Epoch 2/25
60000/60000 [==============================] - 6s 97us/step - loss: 0.0847 - acc: 0.9739 - val_l
Epoch 3/25
60000/60000 [==============================] - 6s 99us/step - loss: 0.0538 - acc: 0.9828 - val_l
Epoch 4/25
60000/60000 [==============================] - 6s 99us/step - loss: 0.0391 - acc: 0.9875 - val_l
```

```
Epoch 5/25
60000/60000 [==============================] - 6s 98us/step - loss: 0.0297 - acc: 0.9902 - val_l
Epoch 6/25
60000/60000 [==============================] - 6s 99us/step - loss: 0.0270 - acc: 0.9911 - val_l
Epoch 7/25
60000/60000 [==============================] - 6s 98us/step - loss: 0.0224 - acc: 0.9926 - val_l
Epoch 8/25
60000/60000 [==============================] - 6s 98us/step - loss: 0.0173 - acc: 0.9941 - val_l
Epoch 9/25
60000/60000 [==============================] - 6s 98us/step - loss: 0.0203 - acc: 0.9932 - val_l
Epoch 10/25
60000/60000 [==============================] - 6s 99us/step - loss: 0.0139 - acc: 0.9957 - val_l
Epoch 11/25
60000/60000 [==============================] - 6s 98us/step - loss: 0.0129 - acc: 0.9956 - val_l
Epoch 12/25
60000/60000 [==============================] - 6s 98us/step - loss: 0.0161 - acc: 0.9948 - val_l
Epoch 13/25
60000/60000 [==============================] - 6s 99us/step - loss: 0.0103 - acc: 0.9968 - val_l
Epoch 14/25
60000/60000 [==============================] - 6s 98us/step - loss: 0.0110 - acc: 0.9966 - val_l
Epoch 15/25
60000/60000 [==============================] - 6s 98us/step - loss: 0.0130 - acc: 0.9956 - val_l
Epoch 16/25
60000/60000 [==============================] - 6s 97us/step - loss: 0.0077 - acc: 0.9974 - val_l
Epoch 17/25
60000/60000 [==============================] - 6s 99us/step - loss: 0.0101 - acc: 0.9970 - val_l
Epoch 18/25
60000/60000 [==============================] - 6s 98us/step - loss: 0.0097 - acc: 0.9969 - val_l
Epoch 19/25
60000/60000 [==============================] - 6s 99us/step - loss: 0.0083 - acc: 0.9974 - val_l
Epoch 20/25
60000/60000 [==============================] - 6s 98us/step - loss: 0.0093 - acc: 0.9970 - val_l
Epoch 21/25
60000/60000 [==============================] - 6s 99us/step - loss: 0.0093 - acc: 0.9970 - val_l
Epoch 22/25
60000/60000 [==============================] - 6s 99us/step - loss: 0.0056 - acc: 0.9982 - val_l
Epoch 23/25
60000/60000 [==============================] - 6s 100us/step - loss: 0.0085 - acc: 0.9976 - val_
Epoch 24/25
60000/60000 [==============================] - 6s 98us/step - loss: 0.0081 - acc: 0.9978 - val_l
Epoch 25/25
60000/60000 [==============================] - 6s 99us/step - loss: 0.0049 - acc: 0.9985 - val_l
```

```
In [43]: score = model.evaluate(X_test, Y_test, verbose=0)
         print('Test score:', score[0])
         print('Test accuracy:', score[1])
```

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

        # list of epoch numbers
        x = list(range(1,26))

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

```
Test score: 0.09119474123555606
Test accuracy: 0.9814
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

**784 - 512 - 256 - 128 - 10 with all relu,batchnorm on hidden layes and softmax output**

```
In [44]: # input
         inputs = Input(shape=(784,),name='Input_layer')
         # hidden layer 1
         x = Dense(512,name='Dlayer1')(inputs)
         x = BatchNormalization(name='Blayer1')(x)
         x = Activation('relu',name='Alayer1')(x)
         #hidden layer2
         x = Dense(256,name='Dlayer2')(x)
         x = BatchNormalization(name='Blayer2')(x)
         x = Activation('relu',name='Alayer2')(x)
         #hidden layer2
         x = Dense(128,name='Dlayer3')(x)
         x = BatchNormalization(name='Blayer3')(x)
         x = Activation('relu',name='Alayer3')(x)
         #out
         out = Dense(10, activation='softmax',name='Output_layer')(x)
         #model
         model = Model(inputs=inputs, outputs=out)
         print(model.summary())
         adam = optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, ams
         #compile
         model.compile(optimizer=adam, loss='categorical_crossentropy', metrics=['accuracy'])
         #training
         history = model.fit(X_train, Y_train, batch_size=128, epochs=25, verbose=1, validation_
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

```
=================================================================
Input_layer (InputLayer)     (None, 784)               0
-----------------------------------------------------------------
Dlayer1 (Dense)              (None, 512)               401920
-----------------------------------------------------------------
Blayer1 (BatchNormalization) (None, 512)               2048
-----------------------------------------------------------------
Alayer1 (Activation)         (None, 512)               0
-----------------------------------------------------------------
Dlayer2 (Dense)              (None, 256)               131328
-----------------------------------------------------------------
Blayer2 (BatchNormalization) (None, 256)               1024
-----------------------------------------------------------------
Alayer2 (Activation)         (None, 256)               0
-----------------------------------------------------------------
Dlayer3 (Dense)              (None, 128)               32896
-----------------------------------------------------------------
Blayer3 (BatchNormalization) (None, 128)               512
-----------------------------------------------------------------
Alayer3 (Activation)         (None, 128)               0
-----------------------------------------------------------------
Output_layer (Dense)         (None, 10)                1290
=================================================================
Total params: 571,018
Trainable params: 569,226
Non-trainable params: 1,792
-----------------------------------------------------------------
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/25
60000/60000 [==============================] - 10s 158us/step - loss: 0.1805 - acc: 0.9471 - val
Epoch 2/25
60000/60000 [==============================] - 8s 139us/step - loss: 0.0690 - acc: 0.9790 - val_
Epoch 3/25
60000/60000 [==============================] - 8s 139us/step - loss: 0.0447 - acc: 0.9857 - val_
Epoch 4/25
60000/60000 [==============================] - 8s 139us/step - loss: 0.0314 - acc: 0.9899 - val_
Epoch 5/25
60000/60000 [==============================] - 8s 139us/step - loss: 0.0285 - acc: 0.9908 - val_
Epoch 6/25
60000/60000 [==============================] - 8s 139us/step - loss: 0.0222 - acc: 0.9928 - val_
Epoch 7/25
60000/60000 [==============================] - 8s 139us/step - loss: 0.0199 - acc: 0.9937 - val_
Epoch 8/25
60000/60000 [==============================] - 8s 139us/step - loss: 0.0156 - acc: 0.9949 - val_
Epoch 9/25
60000/60000 [==============================] - 8s 139us/step - loss: 0.0152 - acc: 0.9951 - val_
Epoch 10/25
```

```
60000/60000 [==============================] - 8s 139us/step - loss: 0.0149 - acc: 0.9951 - val_
Epoch 11/25
60000/60000 [==============================] - 8s 139us/step - loss: 0.0133 - acc: 0.9955 - val_
Epoch 12/25
60000/60000 [==============================] - 8s 139us/step - loss: 0.0097 - acc: 0.9968 - val_
Epoch 13/25
60000/60000 [==============================] - 8s 139us/step - loss: 0.0120 - acc: 0.9959 - val_
Epoch 14/25
60000/60000 [==============================] - 8s 138us/step - loss: 0.0105 - acc: 0.9967 - val_
Epoch 15/25
60000/60000 [==============================] - 8s 139us/step - loss: 0.0083 - acc: 0.9971 - val_
Epoch 16/25
60000/60000 [==============================] - 8s 139us/step - loss: 0.0104 - acc: 0.9968 - val_
Epoch 17/25
60000/60000 [==============================] - 8s 138us/step - loss: 0.0085 - acc: 0.9973 - val_
Epoch 18/25
60000/60000 [==============================] - 8s 139us/step - loss: 0.0070 - acc: 0.9978 - val_
Epoch 19/25
60000/60000 [==============================] - 8s 140us/step - loss: 0.0061 - acc: 0.9981 - val_
Epoch 20/25
60000/60000 [==============================] - 8s 138us/step - loss: 0.0102 - acc: 0.9967 - val_
Epoch 21/25
60000/60000 [==============================] - 8s 139us/step - loss: 0.0068 - acc: 0.9980 - val_
Epoch 22/25
60000/60000 [==============================] - 8s 139us/step - loss: 0.0046 - acc: 0.9987 - val_
Epoch 23/25
60000/60000 [==============================] - 8s 139us/step - loss: 0.0054 - acc: 0.9984 - val_
Epoch 24/25
60000/60000 [==============================] - 8s 139us/step - loss: 0.0063 - acc: 0.9977 - val_
Epoch 25/25
60000/60000 [==============================] - 8s 139us/step - loss: 0.0082 - acc: 0.9972 - val_
```

```
In [45]: score = model.evaluate(X_test, Y_test, verbose=0)
         print('Test score:', score[0])
         print('Test accuracy:', score[1])

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

         # list of epoch numbers
         x = list(range(1,26))

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

```
Test score: 0.07790960603984477
```

```
Test accuracy: 0.9802


<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>
```

**784 - 512 - 256 - 128 - 10 with all relu,batchnorm and dropout on hidden layes and softmax output**

```
In [46]: # input
         inputs = Input(shape=(784,),name='Input_layer')
         # hidden layer 1
         x = Dense(512,name='Dlayer1')(inputs)
         x = BatchNormalization(name='Blayer1')(x)
         x = Activation('relu',name='Alayer1')(x)
         x = Dropout(rate=0.5,name='Drlayer1')(x)
         #hidden layer2
         x = Dense(256,name='Dlayer2')(x)
         x = BatchNormalization(name='Blayer2')(x)
         x = Activation('relu',name='Alayer2')(x)
         x = Dropout(rate=0.5,name='Drlayer2')(x)
         #hidden layer3
         x = Dense(64,name='Dlayer3')(x)
         x = BatchNormalization(name='Blayer3')(x)
         x = Activation('relu',name='Alayer3')(x)
         x = Dropout(rate=0.5,name='Drlayer3')(x)
         #out
         out = Dense(10, activation='softmax',name='Output_layer')(x)
         #model
         model = Model(inputs=inputs, outputs=out)
         print(model.summary())
         adam = optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, ams
         #compile
         model.compile(optimizer=adam, loss='categorical_crossentropy', metrics=['accuracy'])
         #training
         history = model.fit(X_train, Y_train, batch_size=128, epochs=32, verbose=1, validation_
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input_layer (InputLayer)     (None, 784)               0
_________________________________________________________________
Dlayer1 (Dense)              (None, 512)               401920
_________________________________________________________________
Blayer1 (BatchNormalization) (None, 512)               2048
```

```
_________________________________________________________________
Alayer1 (Activation)         (None, 512)               0
_________________________________________________________________
Drlayer1 (Dropout)           (None, 512)               0
_________________________________________________________________
Dlayer2 (Dense)              (None, 256)               131328
_________________________________________________________________
Blayer2 (BatchNormalization) (None, 256)               1024
_________________________________________________________________
Alayer2 (Activation)         (None, 256)               0
_________________________________________________________________
Drlayer2 (Dropout)           (None, 256)               0
_________________________________________________________________
Dlayer3 (Dense)              (None, 64)                16448
_________________________________________________________________
Blayer3 (BatchNormalization) (None, 64)                256
_________________________________________________________________
Alayer3 (Activation)         (None, 64)                0
_________________________________________________________________
Drlayer3 (Dropout)           (None, 64)                0
_________________________________________________________________
Output_layer (Dense)         (None, 10)                650
=================================================================
Total params: 553,674
Trainable params: 552,010
Non-trainable params: 1,664
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/32
60000/60000 [==============================] - 10s 172us/step - loss: 0.6480 - acc: 0.8113 - val
Epoch 2/32
60000/60000 [==============================] - 9s 150us/step - loss: 0.3009 - acc: 0.9178 - val_
Epoch 3/32
60000/60000 [==============================] - 9s 150us/step - loss: 0.2358 - acc: 0.9351 - val_
Epoch 4/32
60000/60000 [==============================] - 9s 150us/step - loss: 0.2023 - acc: 0.9446 - val_
Epoch 5/32
60000/60000 [==============================] - 9s 149us/step - loss: 0.1747 - acc: 0.9518 - val_
Epoch 6/32
60000/60000 [==============================] - 9s 150us/step - loss: 0.1577 - acc: 0.9571 - val_
Epoch 7/32
60000/60000 [==============================] - 9s 149us/step - loss: 0.1450 - acc: 0.9601 - val_
Epoch 8/32
60000/60000 [==============================] - 9s 150us/step - loss: 0.1310 - acc: 0.9635 - val_
Epoch 9/32
60000/60000 [==============================] - 9s 150us/step - loss: 0.1258 - acc: 0.9644 - val_
Epoch 10/32
```

```
60000/60000 [==============================] - 9s 150us/step - loss: 0.1143 - acc: 0.9681 - val_
Epoch 11/32
60000/60000 [==============================] - 9s 149us/step - loss: 0.1109 - acc: 0.9691 - val_
Epoch 12/32
60000/60000 [==============================] - 9s 150us/step - loss: 0.1027 - acc: 0.9714 - val_
Epoch 13/32
60000/60000 [==============================] - 9s 150us/step - loss: 0.0983 - acc: 0.9722 - val_
Epoch 14/32
60000/60000 [==============================] - 9s 150us/step - loss: 0.0943 - acc: 0.9734 - val_
Epoch 15/32
60000/60000 [==============================] - 9s 150us/step - loss: 0.0909 - acc: 0.9748 - val_
Epoch 16/32
60000/60000 [==============================] - 9s 150us/step - loss: 0.0865 - acc: 0.9757 - val_
Epoch 17/32
60000/60000 [==============================] - 9s 150us/step - loss: 0.0828 - acc: 0.9768 - val_
Epoch 18/32
60000/60000 [==============================] - 9s 151us/step - loss: 0.0795 - acc: 0.9774 - val_
Epoch 19/32
60000/60000 [==============================] - 9s 150us/step - loss: 0.0758 - acc: 0.9789 - val_
Epoch 20/32
60000/60000 [==============================] - 9s 149us/step - loss: 0.0755 - acc: 0.9791 - val_
Epoch 21/32
60000/60000 [==============================] - 9s 150us/step - loss: 0.0705 - acc: 0.9805 - val_
Epoch 22/32
60000/60000 [==============================] - 9s 149us/step - loss: 0.0679 - acc: 0.9811 - val_
Epoch 23/32
60000/60000 [==============================] - 9s 150us/step - loss: 0.0645 - acc: 0.9826 - val_
Epoch 24/32
60000/60000 [==============================] - 9s 150us/step - loss: 0.0664 - acc: 0.9819 - val_
Epoch 25/32
60000/60000 [==============================] - 9s 150us/step - loss: 0.0683 - acc: 0.9808 - val_
Epoch 26/32
60000/60000 [==============================] - 9s 150us/step - loss: 0.0602 - acc: 0.9827 - val_
Epoch 27/32
60000/60000 [==============================] - 9s 149us/step - loss: 0.0610 - acc: 0.9825 - val_
Epoch 28/32
60000/60000 [==============================] - 9s 150us/step - loss: 0.0592 - acc: 0.9829 - val_
Epoch 29/32
60000/60000 [==============================] - 9s 150us/step - loss: 0.0530 - acc: 0.9849 - val_
Epoch 30/32
60000/60000 [==============================] - 9s 150us/step - loss: 0.0584 - acc: 0.9832 - val_
Epoch 31/32
60000/60000 [==============================] - 9s 150us/step - loss: 0.0560 - acc: 0.9838 - val_
Epoch 32/32
60000/60000 [==============================] - 9s 150us/step - loss: 0.0561 - acc: 0.9841 - val_
```

```
In [47]: score = model.evaluate(X_test, Y_test, verbose=0)
```

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

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

        # list of epoch numbers
        x = list(range(1,33))

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

```
Test score: 0.06090575837812812
Test accuracy: 0.984
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

#### 0.0.3 5 layer network

**784 - 512 - 256 - 128 - 64 - 32 - 10 with all relu and softmax output**

```
In [10]: # input
         inputs = Input(shape=(784,),name='Input_layer')
         # hidden layer 1
         x = Dense(512,name='Dlayer1')(inputs)
         x = Activation('relu',name='Alayer1')(x)
         #hidden layer2
         x = Dense(256,name='Dlayer2')(x)
         x = Activation('relu',name='Alayer2')(x)
         #hidden layer3
         x = Dense(128,name='Dlayer3')(x)
         x = Activation('relu',name='Alayer3')(x)
         #hidden layer4
         x = Dense(64,name='Dlayer4')(x)
         x = Activation('relu',name='Alayer4')(x)
         #hidden layer5
         x = Dense(32,name='Dlayer5')(x)
         x = Activation('relu',name='Alayer5')(x)
         #out
         out = Dense(10, activation='softmax',name='Output_layer')(x)
         #model
         model = Model(inputs=inputs, outputs=out)
         print(model.summary())
```

```
        adam = optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, ams
        #compile
        model.compile(optimizer=adam, loss='categorical_crossentropy', metrics=['accuracy'])
        #training
        history = model.fit(X_train, Y_train, batch_size=128, epochs=25, verbose=1, validation_
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input_layer (InputLayer)     (None, 784)               0
_________________________________________________________________
Dlayer1 (Dense)              (None, 512)               401920
_________________________________________________________________
Alayer1 (Activation)         (None, 512)               0
_________________________________________________________________
Dlayer2 (Dense)              (None, 256)               131328
_________________________________________________________________
Alayer2 (Activation)         (None, 256)               0
_________________________________________________________________
Dlayer3 (Dense)              (None, 128)               32896
_________________________________________________________________
Alayer3 (Activation)         (None, 128)               0
_________________________________________________________________
Dlayer4 (Dense)              (None, 64)                8256
_________________________________________________________________
Alayer4 (Activation)         (None, 64)                0
_________________________________________________________________
Dlayer5 (Dense)              (None, 32)                2080
_________________________________________________________________
Alayer5 (Activation)         (None, 32)                0
_________________________________________________________________
Output_layer (Dense)         (None, 10)                330
=================================================================
Total params: 576,810
Trainable params: 576,810
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/25
60000/60000 [==============================] - 6s 106us/step - loss: 0.3022 - acc: 0.9073 - val_
Epoch 2/25
60000/60000 [==============================] - 6s 103us/step - loss: 0.0992 - acc: 0.9696 - val_
Epoch 3/25
60000/60000 [==============================] - 6s 103us/step - loss: 0.0660 - acc: 0.9792 - val_
Epoch 4/25
60000/60000 [==============================] - 6s 104us/step - loss: 0.0500 - acc: 0.9843 - val_
Epoch 5/25
```

```
60000/60000 [==============================] - 6s 104us/step - loss: 0.0359 - acc: 0.9879 - val_
Epoch 6/25
60000/60000 [==============================] - 6s 103us/step - loss: 0.0302 - acc: 0.9905 - val_
Epoch 7/25
60000/60000 [==============================] - 6s 102us/step - loss: 0.0285 - acc: 0.9906 - val_
Epoch 8/25
60000/60000 [==============================] - 6s 104us/step - loss: 0.0215 - acc: 0.9929 - val_
Epoch 9/25
60000/60000 [==============================] - 6s 104us/step - loss: 0.0219 - acc: 0.9931 - val_
Epoch 10/25
60000/60000 [==============================] - 6s 103us/step - loss: 0.0161 - acc: 0.9950 - val_
Epoch 11/25
60000/60000 [==============================] - 6s 103us/step - loss: 0.0191 - acc: 0.9941 - val_
Epoch 12/25
60000/60000 [==============================] - 6s 103us/step - loss: 0.0150 - acc: 0.9954 - val_
Epoch 13/25
60000/60000 [==============================] - 6s 104us/step - loss: 0.0119 - acc: 0.9964 - val_
Epoch 14/25
60000/60000 [==============================] - 6s 104us/step - loss: 0.0165 - acc: 0.9948 - val_
Epoch 15/25
60000/60000 [==============================] - 6s 103us/step - loss: 0.0104 - acc: 0.9971 - val_
Epoch 16/25
60000/60000 [==============================] - 6s 103us/step - loss: 0.0129 - acc: 0.9962 - val_
Epoch 17/25
60000/60000 [==============================] - 6s 103us/step - loss: 0.0136 - acc: 0.9957 - val_
Epoch 18/25
60000/60000 [==============================] - 6s 103us/step - loss: 0.0117 - acc: 0.9963 - val_
Epoch 19/25
60000/60000 [==============================] - 6s 102us/step - loss: 0.0069 - acc: 0.9980 - val_
Epoch 20/25
60000/60000 [==============================] - 6s 103us/step - loss: 0.0132 - acc: 0.9961 - val_
Epoch 21/25
60000/60000 [==============================] - 6s 104us/step - loss: 0.0077 - acc: 0.9978 - val_
Epoch 22/25
60000/60000 [==============================] - 6s 103us/step - loss: 0.0090 - acc: 0.9972 - val_
Epoch 23/25
60000/60000 [==============================] - 6s 104us/step - loss: 0.0092 - acc: 0.9974 - val_
Epoch 24/25
60000/60000 [==============================] - 6s 102us/step - loss: 0.0058 - acc: 0.9983 - val_
Epoch 25/25
60000/60000 [==============================] - 6s 102us/step - loss: 0.0084 - acc: 0.9974 - val_
```

```
In [11]: score = model.evaluate(X_test, Y_test, verbose=0)
         print('Test score:', score[0])
         print('Test accuracy:', score[1])

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

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

        # list of epoch numbers
        x = list(range(1,26))

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

```
Test score: 0.09583575944229597
Test accuracy: 0.9826
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

With step decay of learning rate

```
In [23]: from keras.callbacks import LearningRateScheduler
         import math
         def step_decay(epoch):
             initial_lrate = 0.0005
             drop = 0.5
             epochs_drop = 3.0
             lrate = initial_lrate * math.pow(drop,
                    math.floor((1+epoch)/epochs_drop))
             return lrate
         lrate = LearningRateScheduler(step_decay)
```

```
In [24]: # input
         inputs = Input(shape=(784,),name='Input_layer')
         # hidden layer 1
         x = Dense(512,name='Dlayer1')(inputs)
         x = Activation('relu',name='Alayer1')(x)
         #hidden layer2
         x = Dense(256,name='Dlayer2')(x)
         x = Activation('relu',name='Alayer2')(x)
         #hidden layer3
         x = Dense(128,name='Dlayer3')(x)
         x = Activation('relu',name='Alayer3')(x)
         #hidden layer4
         x = Dense(64,name='Dlayer4')(x)
         x = Activation('relu',name='Alayer4')(x)
         #hidden layer5
         x = Dense(32,name='Dlayer5')(x)
         x = Activation('relu',name='Alayer5')(x)
```

```
        #out
        out = Dense(10, activation='softmax',name='Output_layer')(x)
        #model
        model = Model(inputs=inputs, outputs=out)
        print(model.summary())
        adam = optimizers.Adam(lr=0.0005, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, am
        #compile
        model.compile(optimizer=adam, loss='categorical_crossentropy', metrics=['accuracy'])
        #training
        history = model.fit(X_train, Y_train, batch_size=128, epochs=25, verbose=1,
                            validation_data=(X_test, Y_test),callbacks=[lrate])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input_layer (InputLayer)     (None, 784)               0
_________________________________________________________________
Dlayer1 (Dense)              (None, 512)               401920
_________________________________________________________________
Alayer1 (Activation)         (None, 512)               0
_________________________________________________________________
Dlayer2 (Dense)              (None, 256)               131328
_________________________________________________________________
Alayer2 (Activation)         (None, 256)               0
_________________________________________________________________
Dlayer3 (Dense)              (None, 128)               32896
_________________________________________________________________
Alayer3 (Activation)         (None, 128)               0
_________________________________________________________________
Dlayer4 (Dense)              (None, 64)                8256
_________________________________________________________________
Alayer4 (Activation)         (None, 64)                0
_________________________________________________________________
Dlayer5 (Dense)              (None, 32)                2080
_________________________________________________________________
Alayer5 (Activation)         (None, 32)                0
_________________________________________________________________
Output_layer (Dense)         (None, 10)                330
=================================================================
Total params: 576,810
Trainable params: 576,810
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/25
60000/60000 [==============================] - 7s 109us/step - loss: 0.3254 - acc: 0.9069 - val_
Epoch 2/25
```

```
60000/60000 [==============================] - 6s 103us/step - loss: 0.1124 - acc: 0.9674 - val_
Epoch 3/25
60000/60000 [==============================] - 6s 101us/step - loss: 0.0599 - acc: 0.9821 - val_
Epoch 4/25
60000/60000 [==============================] - 6s 104us/step - loss: 0.0453 - acc: 0.9868 - val_
Epoch 5/25
60000/60000 [==============================] - 6s 104us/step - loss: 0.0341 - acc: 0.9903 - val_
Epoch 6/25
60000/60000 [==============================] - 6s 104us/step - loss: 0.0208 - acc: 0.9943 - val_
Epoch 7/25
60000/60000 [==============================] - 6s 104us/step - loss: 0.0161 - acc: 0.9963 - val_
Epoch 8/25
60000/60000 [==============================] - 6s 104us/step - loss: 0.0131 - acc: 0.9970 - val_
Epoch 9/25
60000/60000 [==============================] - 6s 103us/step - loss: 0.0088 - acc: 0.9984 - val_
Epoch 10/25
60000/60000 [==============================] - 6s 104us/step - loss: 0.0072 - acc: 0.9989 - val_
Epoch 11/25
60000/60000 [==============================] - 6s 103us/step - loss: 0.0062 - acc: 0.9992 - val_
Epoch 12/25
60000/60000 [==============================] - 6s 104us/step - loss: 0.0048 - acc: 0.9994 - val_
Epoch 13/25
60000/60000 [==============================] - 6s 104us/step - loss: 0.0043 - acc: 0.9995 - val_
Epoch 14/25
60000/60000 [==============================] - 6s 104us/step - loss: 0.0038 - acc: 0.9996 - val_
Epoch 15/25
60000/60000 [==============================] - 6s 103us/step - loss: 0.0032 - acc: 0.9997 - val_
Epoch 16/25
60000/60000 [==============================] - 6s 103us/step - loss: 0.0030 - acc: 0.9998 - val_
Epoch 17/25
60000/60000 [==============================] - 6s 103us/step - loss: 0.0028 - acc: 0.9998 - val_
Epoch 18/25
60000/60000 [==============================] - 6s 104us/step - loss: 0.0026 - acc: 0.9998 - val_
Epoch 19/25
60000/60000 [==============================] - 6s 104us/step - loss: 0.0025 - acc: 0.9998 - val_
Epoch 20/25
60000/60000 [==============================] - 6s 105us/step - loss: 0.0024 - acc: 0.9998 - val_
Epoch 21/25
60000/60000 [==============================] - 6s 103us/step - loss: 0.0022 - acc: 0.9998 - val_
Epoch 22/25
60000/60000 [==============================] - 6s 103us/step - loss: 0.0022 - acc: 0.9998 - val_
Epoch 23/25
60000/60000 [==============================] - 6s 104us/step - loss: 0.0021 - acc: 0.9998 - val_
Epoch 24/25
60000/60000 [==============================] - 6s 103us/step - loss: 0.0021 - acc: 0.9999 - val_
Epoch 25/25
60000/60000 [==============================] - 6s 103us/step - loss: 0.0020 - acc: 0.9998 - val_
```

```
In [25]: score = model.evaluate(X_test, Y_test, verbose=0)
         print('Test score:', score[0])
         print('Test accuracy:', score[1])

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

         # list of epoch numbers
         x = list(range(1,26))

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

```
Test score: 0.07307920760185134
Test accuracy: 0.9819
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

**784 - 512 - 256 - 128 - 64 - 32 - 10 with all relu,batchnorm on hidden layers and softmax output**

```
In [26]: # input
         inputs = Input(shape=(784,),name='Input_layer')
         # hidden layer 1
         x = Dense(512,name='Dlayer1')(inputs)
         x = BatchNormalization(name='Blayer1')(x)
         x = Activation('relu',name='Alayer1')(x)
         #hidden layer2
         x = Dense(256,name='Dlayer2')(x)
         x = BatchNormalization(name='Blayer2')(x)
         x = Activation('relu',name='Alayer2')(x)
         #hidden layer3
         x = Dense(128,name='Dlayer3')(x)
         x = BatchNormalization(name='Blayer3')(x)
         x = Activation('relu',name='Alayer3')(x)
         #hidden layer4
         x = Dense(64,name='Dlayer4')(x)
         x = BatchNormalization(name='Blayer4')(x)
         x = Activation('relu',name='Alayer4')(x)
         #hidden layer5
         x = Dense(32,name='Dlayer5')(x)
         x = BatchNormalization(name='Blayer5')(x)
         x = Activation('relu',name='Alayer5')(x)
```

```
        #out
        out = Dense(10, activation='softmax',name='Output_layer')(x)
        #model
        model = Model(inputs=inputs, outputs=out)
        print(model.summary())
        adam = optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, ams
        #compile
        model.compile(optimizer=adam, loss='categorical_crossentropy', metrics=['accuracy'])
        #training
        history = model.fit(X_train, Y_train, batch_size=128, epochs=32, verbose=1, validation_
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input_layer (InputLayer)     (None, 784)               0
_________________________________________________________________
Dlayer1 (Dense)              (None, 512)               401920
_________________________________________________________________
Blayer1 (BatchNormalization) (None, 512)               2048
_________________________________________________________________
Alayer1 (Activation)         (None, 512)               0
_________________________________________________________________
Dlayer2 (Dense)              (None, 256)               131328
_________________________________________________________________
Blayer2 (BatchNormalization) (None, 256)               1024
_________________________________________________________________
Alayer2 (Activation)         (None, 256)               0
_________________________________________________________________
Dlayer3 (Dense)              (None, 128)               32896
_________________________________________________________________
Blayer3 (BatchNormalization) (None, 128)               512
_________________________________________________________________
Alayer3 (Activation)         (None, 128)               0
_________________________________________________________________
Dlayer4 (Dense)              (None, 64)                8256
_________________________________________________________________
Blayer4 (BatchNormalization) (None, 64)                256
_________________________________________________________________
Alayer4 (Activation)         (None, 64)                0
_________________________________________________________________
Dlayer5 (Dense)              (None, 32)                2080
_________________________________________________________________
Blayer5 (BatchNormalization) (None, 32)                128
_________________________________________________________________
Alayer5 (Activation)         (None, 32)                0
_________________________________________________________________
Output_layer (Dense)         (None, 10)                330
=================================================================
```

```
Total params: 580,778
Trainable params: 578,794
Non-trainable params: 1,984
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/32
60000/60000 [==============================] - 10s 173us/step - loss: 0.2759 - acc: 0.9339 - val
Epoch 2/32
60000/60000 [==============================] - 9s 156us/step - loss: 0.0889 - acc: 0.9741 - val_
Epoch 3/32
60000/60000 [==============================] - 9s 156us/step - loss: 0.0620 - acc: 0.9813 - val_
Epoch 4/32
60000/60000 [==============================] - 9s 156us/step - loss: 0.0480 - acc: 0.9849 - val_
Epoch 5/32
60000/60000 [==============================] - 9s 156us/step - loss: 0.0364 - acc: 0.9888 - val_
Epoch 6/32
60000/60000 [==============================] - 9s 156us/step - loss: 0.0339 - acc: 0.9895 - val_
Epoch 7/32
60000/60000 [==============================] - 9s 156us/step - loss: 0.0259 - acc: 0.9920 - val_
Epoch 8/32
60000/60000 [==============================] - 9s 156us/step - loss: 0.0248 - acc: 0.9917 - val_
Epoch 9/32
60000/60000 [==============================] - 9s 155us/step - loss: 0.0202 - acc: 0.9937 - val_
Epoch 10/32
60000/60000 [==============================] - 9s 156us/step - loss: 0.0200 - acc: 0.9932 - val_
Epoch 11/32
60000/60000 [==============================] - 9s 156us/step - loss: 0.0180 - acc: 0.9939 - val_
Epoch 12/32
60000/60000 [==============================] - 9s 156us/step - loss: 0.0190 - acc: 0.9939 - val_
Epoch 13/32
60000/60000 [==============================] - 9s 153us/step - loss: 0.0159 - acc: 0.9949 - val_
Epoch 14/32
60000/60000 [==============================] - 9s 156us/step - loss: 0.0133 - acc: 0.9955 - val_
Epoch 15/32
60000/60000 [==============================] - 9s 155us/step - loss: 0.0144 - acc: 0.9953 - val_
Epoch 16/32
60000/60000 [==============================] - 9s 157us/step - loss: 0.0119 - acc: 0.9961 - val_
Epoch 17/32
60000/60000 [==============================] - 9s 156us/step - loss: 0.0142 - acc: 0.9953 - val_
Epoch 18/32
60000/60000 [==============================] - 9s 156us/step - loss: 0.0103 - acc: 0.9967 - val_
Epoch 19/32
60000/60000 [==============================] - 9s 156us/step - loss: 0.0107 - acc: 0.9966 - val_
Epoch 20/32
60000/60000 [==============================] - 9s 156us/step - loss: 0.0092 - acc: 0.9969 - val_
Epoch 21/32
60000/60000 [==============================] - 9s 156us/step - loss: 0.0126 - acc: 0.9959 - val_
```

```
Epoch 22/32
60000/60000 [==============================] - 9s 156us/step - loss: 0.0122 - acc: 0.9960 - val_
Epoch 23/32
60000/60000 [==============================] - 9s 155us/step - loss: 0.0060 - acc: 0.9979 - val_
Epoch 24/32
60000/60000 [==============================] - 9s 156us/step - loss: 0.0086 - acc: 0.9971 - val_
Epoch 25/32
60000/60000 [==============================] - 9s 156us/step - loss: 0.0103 - acc: 0.9968 - val_
Epoch 26/32
60000/60000 [==============================] - 9s 156us/step - loss: 0.0097 - acc: 0.9965 - val_
Epoch 27/32
60000/60000 [==============================] - 9s 153us/step - loss: 0.0070 - acc: 0.9977 - val_
Epoch 28/32
60000/60000 [==============================] - 9s 154us/step - loss: 0.0065 - acc: 0.9980 - val_
Epoch 29/32
60000/60000 [==============================] - 9s 155us/step - loss: 0.0065 - acc: 0.9981 - val_
Epoch 30/32
60000/60000 [==============================] - 9s 156us/step - loss: 0.0070 - acc: 0.9976 - val_
Epoch 31/32
60000/60000 [==============================] - 9s 156us/step - loss: 0.0067 - acc: 0.9977 - val_
Epoch 32/32
60000/60000 [==============================] - 9s 156us/step - loss: 0.0048 - acc: 0.9987 - val_
```

```
In [28]: score = model.evaluate(X_test, Y_test, verbose=0)
         print('Test score:', score[0])
         print('Test accuracy:', score[1])

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

         # list of epoch numbers
         x = list(range(1,33))

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

```
Test score: 0.059243739600876144
Test accuracy: 0.9858
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

**784 - 512 - 256 - 128 - 64 - 32 - 10 with all relu,batchnorm and dropout on hidden layers and softmax output**

```
In [31]: # input
         inputs = Input(shape=(784,),name='Input_layer')
         # hidden layer 1
         x = Dense(512,kernel_initializer='he_normal',name='Dlayer1')(inputs)
         x = BatchNormalization(name='Blayer1')(x)
         x = Activation('relu',name='Alayer1')(x)
         x = Dropout(rate=0.6,name='Drlayer1')(x)
         #hidden layer2
         x = Dense(256,kernel_initializer='he_normal',name='Dlayer2')(x)
         x = BatchNormalization(name='Blayer2')(x)
         x = Activation('relu',name='Alayer2')(x)
         x = Dropout(rate=0.5,name='Drlayer2')(x)
         #hidden layer3
         x = Dense(128,kernel_initializer='he_normal',name='Dlayer3')(x)
         x = BatchNormalization(name='Blayer3')(x)
         x = Activation('relu',name='Alayer3')(x)
         x = Dropout(rate=0.5,name='Drlayer3')(x)
         #hidden layer4
         x = Dense(64,kernel_initializer='he_normal',name='Dlayer4')(x)
         x = BatchNormalization(name='Blayer4')(x)
         x = Activation('relu',name='Alayer4')(x)
         x = Dropout(rate=0.4,name='Drlayer4')(x)
         #hidden layer5
         x = Dense(32,kernel_initializer='he_normal',name='Dlayer5')(x)
         x = BatchNormalization(name='Blayer5')(x)
         x = Activation('relu',name='Alayer5')(x)
         x = Dropout(rate=0.4,name='Drlayer5')(x)
         #out
         out = Dense(10,kernel_initializer='he_normal', activation='softmax',name='Output_layer'
         #model
         model = Model(inputs=inputs, outputs=out)
         print(model.summary())
         adam = optimizers.Adam(lr=0.005, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, ams
         #compile
         model.compile(optimizer=adam, loss='categorical_crossentropy', metrics=['accuracy'])
         #training
         history = model.fit(X_train, Y_train, batch_size=128, epochs=32, verbose=1, validation_
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input_layer (InputLayer)     (None, 784)               0
_________________________________________________________________
Dlayer1 (Dense)              (None, 512)               401920
_________________________________________________________________
Blayer1 (BatchNormalization) (None, 512)               2048
```

```
_________________________________________________________________
Alayer1 (Activation)         (None, 512)               0
_________________________________________________________________
Drlayer1 (Dropout)           (None, 512)               0
_________________________________________________________________
Dlayer2 (Dense)              (None, 256)               131328
_________________________________________________________________
Blayer2 (BatchNormalization) (None, 256)               1024
_________________________________________________________________
Alayer2 (Activation)         (None, 256)               0
_________________________________________________________________
Drlayer2 (Dropout)           (None, 256)               0
_________________________________________________________________
Dlayer3 (Dense)              (None, 128)               32896
_________________________________________________________________
Blayer3 (BatchNormalization) (None, 128)               512
_________________________________________________________________
Alayer3 (Activation)         (None, 128)               0
_________________________________________________________________
Drlayer3 (Dropout)           (None, 128)               0
_________________________________________________________________
Dlayer4 (Dense)              (None, 64)                8256
_________________________________________________________________
Blayer4 (BatchNormalization) (None, 64)                256
_________________________________________________________________
Alayer4 (Activation)         (None, 64)                0
_________________________________________________________________
Drlayer4 (Dropout)           (None, 64)                0
_________________________________________________________________
Dlayer5 (Dense)              (None, 32)                2080
_________________________________________________________________
Blayer5 (BatchNormalization) (None, 32)                128
_________________________________________________________________
Alayer5 (Activation)         (None, 32)                0
_________________________________________________________________
Drlayer5 (Dropout)           (None, 32)                0
_________________________________________________________________
Output_layer (Dense)         (None, 10)                330
=================================================================
Total params: 580,778
Trainable params: 578,794
Non-trainable params: 1,984
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/32
60000/60000 [==============================] - 12s 203us/step - loss: 0.8933 - acc: 0.7258 - val
Epoch 2/32
```

```
60000/60000 [==============================] - 11s 176us/step - loss: 0.4372 - acc: 0.8903 - val
Epoch 3/32
60000/60000 [==============================] - 11s 176us/step - loss: 0.3601 - acc: 0.9145 - val
Epoch 4/32
60000/60000 [==============================] - 11s 177us/step - loss: 0.3187 - acc: 0.9263 - val
Epoch 5/32
60000/60000 [==============================] - 11s 176us/step - loss: 0.2943 - acc: 0.9318 - val
Epoch 6/32
60000/60000 [==============================] - 11s 177us/step - loss: 0.2689 - acc: 0.9371 - val
Epoch 7/32
60000/60000 [==============================] - 11s 176us/step - loss: 0.2537 - acc: 0.9409 - val
Epoch 8/32
60000/60000 [==============================] - 11s 177us/step - loss: 0.2426 - acc: 0.9444 - val
Epoch 9/32
60000/60000 [==============================] - 11s 177us/step - loss: 0.2342 - acc: 0.9458 - val
Epoch 10/32
60000/60000 [==============================] - 11s 177us/step - loss: 0.2216 - acc: 0.9473 - val
Epoch 11/32
60000/60000 [==============================] - 11s 176us/step - loss: 0.2146 - acc: 0.9500 - val
Epoch 12/32
60000/60000 [==============================] - 11s 177us/step - loss: 0.2109 - acc: 0.9516 - val
Epoch 13/32
60000/60000 [==============================] - 11s 176us/step - loss: 0.2040 - acc: 0.9539 - val
Epoch 14/32
60000/60000 [==============================] - 11s 175us/step - loss: 0.1982 - acc: 0.9551 - val
Epoch 15/32
60000/60000 [==============================] - 11s 176us/step - loss: 0.1953 - acc: 0.9551 - val
Epoch 16/32
60000/60000 [==============================] - 10s 174us/step - loss: 0.1877 - acc: 0.9571 - val
Epoch 17/32
60000/60000 [==============================] - 11s 176us/step - loss: 0.1825 - acc: 0.9580 - val
Epoch 18/32
60000/60000 [==============================] - 11s 175us/step - loss: 0.1809 - acc: 0.9572 - val
Epoch 19/32
60000/60000 [==============================] - 11s 176us/step - loss: 0.1741 - acc: 0.9602 - val
Epoch 20/32
60000/60000 [==============================] - 11s 177us/step - loss: 0.1741 - acc: 0.9600 - val
Epoch 21/32
60000/60000 [==============================] - 11s 178us/step - loss: 0.1659 - acc: 0.9624 - val
Epoch 22/32
60000/60000 [==============================] - 11s 175us/step - loss: 0.1602 - acc: 0.9624 - val
Epoch 23/32
60000/60000 [==============================] - 11s 176us/step - loss: 0.1593 - acc: 0.9633 - val
Epoch 24/32
60000/60000 [==============================] - 11s 176us/step - loss: 0.1652 - acc: 0.9619 - val
Epoch 25/32
60000/60000 [==============================] - 11s 176us/step - loss: 0.1605 - acc: 0.9626 - val
Epoch 26/32
```

```
60000/60000 [==============================] - 11s 176us/step - loss: 0.1504 - acc: 0.9660 - val
Epoch 27/32
60000/60000 [==============================] - 11s 177us/step - loss: 0.1552 - acc: 0.9650 - val
Epoch 28/32
60000/60000 [==============================] - 11s 177us/step - loss: 0.1531 - acc: 0.9652 - val
Epoch 29/32
60000/60000 [==============================] - 11s 177us/step - loss: 0.1471 - acc: 0.9671 - val
Epoch 30/32
60000/60000 [==============================] - 11s 177us/step - loss: 0.1440 - acc: 0.9661 - val
Epoch 31/32
60000/60000 [==============================] - 10s 174us/step - loss: 0.1391 - acc: 0.9674 - val
Epoch 32/32
60000/60000 [==============================] - 11s 177us/step - loss: 0.1391 - acc: 0.9673 - val
```

```
In [32]: score = model.evaluate(X_test, Y_test, verbose=0)
         print('Test score:', score[0])
         print('Test accuracy:', score[1])

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

         # list of epoch numbers
         x = list(range(1,33))

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

```
Test score: 0.07434172039366786
Test accuracy: 0.9828
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```