MLP_on_MNIST (4)

March 15, 2019

0.1 ASSIGNMENT 12

0.1.1 OBJECTIVE: To Build Various MLP architecture for MNIST Dataset

In [0]:
```
from keras.utils import np_utils
from keras.datasets import mnist
import seaborn as sns
from keras.initializers import RandomNormal
from keras.layers import Dropout
from keras.layers.normalization import BatchNormalization
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.initializers import he_normal
```

In [0]:
```
# loading both training and testing Dataset
(X_train , y_train) , (X_test , y_test) = mnist.load_data()
```

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

In [0]:
```
X_train.shape
```

Out[0]:
```
(60000, 28, 28)
```

In [0]:
```
print('Number of training points: {}'.format(X_train.shape[0]),' and each image is {} X
print('Number of training points: {}'.format(X_test.shape[0]),' and each image is {} X
```

```
Number of training points: 60000  and each image is 28 X 28 pixel
Number of training points: 10000  and each image is 28 X 28 pixel
```

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

```
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 [0]: print(type(X_train[0]))
```

```
<class 'numpy.ndarray'>
```

```
In [0]: # normalizing
        X_train = X_train/255
        X_test = X_test/255
```

```
In [0]: # 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)           # one hot encoding
        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 [0]: import matplotlib.pyplot as plt
        import numpy as np
        import time
        # https://gist.github.com/greydanus/f6eee59eaf1d90fcb3b534a25362cea4
        # https://stackoverflow.com/a/14434334
        # this function is used to update the plots for each epoch and error
        def plt_dynamic(x, vy, ty, ax, colors=['b']):
            ax.plot(x, vy, 'b', label="Validation Loss")
            ax.plot(x, ty, 'r', label="Train Loss")
            plt.legend()
            plt.grid()
            fig.canvas.draw()
```

0.2 SOFTMAX CLASSIFIER with 2 hidden layers

```
In [0]: # model parameters
        output_dim = 10
        input_dim = X_train.shape[1]

        batch_size = 128
```

```
nb_epoch = 20

# start building a model
model_2_layers = Sequential()


model_2_layers.add(Dense(364, activation='relu', input_shape=(input_dim,), kernel_initi
model_2_layers.add(BatchNormalization())            # batch normalization
model_2_layers.add(Dropout(0.25))                    #dropout


model_2_layers.add(Dense(52, activation='relu', kernel_initializer=he_normal(seed = No
model_2_layers.add(BatchNormalization())
model_2_layers.add(Dropout(0.25))

model_2_layers.add(Dense(output_dim, activation='softmax'))

# model Summary
print("Model Summary :- \n",model_2_layers.summary())

# Compiling the model
model_2_layers.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['ac

#training model
history_2_layers = model_2_layers.fit(X_train, Y_train, batch_size=batch_size, epochs=
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 364)               285740
_________________________________________________________________
batch_normalization_3 (Batch (None, 364)               1456
_________________________________________________________________
dropout_3 (Dropout)          (None, 364)               0
_________________________________________________________________
dense_5 (Dense)              (None, 52)                18980
_________________________________________________________________
batch_normalization_4 (Batch (None, 52)                208
_________________________________________________________________
dropout_4 (Dropout)          (None, 52)                0
_________________________________________________________________
dense_6 (Dense)              (None, 10)                530
=================================================================
Total params: 306,914
Trainable params: 306,082
Non-trainable params: 832
_________________________________________________________________
Model Summary :-
```

```
 None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 6s 100us/step - loss: 0.3387 - acc: 0.9021 - val
Epoch 2/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.1513 - acc: 0.9552 - val_
Epoch 3/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.1128 - acc: 0.9653 - val_
Epoch 4/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0917 - acc: 0.9719 - val_
Epoch 5/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0806 - acc: 0.9748 - val_
Epoch 6/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0680 - acc: 0.9784 - val_
Epoch 7/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0629 - acc: 0.9802 - val_
Epoch 8/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0565 - acc: 0.9819 - val_
Epoch 9/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.0512 - acc: 0.9839 - val_
Epoch 10/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.0485 - acc: 0.9847 - val_
Epoch 11/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0445 - acc: 0.9856 - val_
Epoch 12/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0409 - acc: 0.9871 - val_
Epoch 13/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.0410 - acc: 0.9869 - val_
Epoch 14/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.0409 - acc: 0.9870 - val_
Epoch 15/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0358 - acc: 0.9880 - val_
Epoch 16/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.0337 - acc: 0.9885 - val_
Epoch 17/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0314 - acc: 0.9898 - val_
Epoch 18/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0320 - acc: 0.9896 - val_
Epoch 19/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0311 - acc: 0.9900 - val_
Epoch 20/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0273 - acc: 0.9913 - val_
```

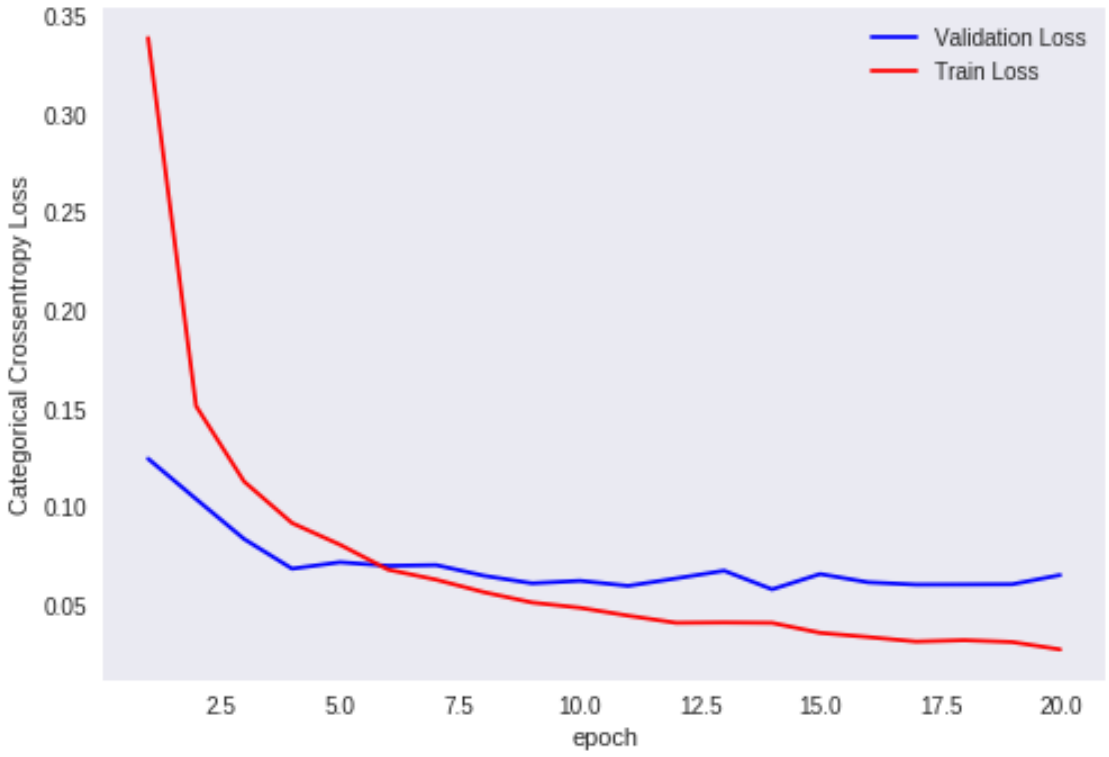
```
In [0]: score = model_2_layers.evaluate(X_test, Y_test, verbose=0)
        print('Test score:', score[0])
        print('Test accuracy:', score[1])
```

```
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))


vy = history_2_layers.history['val_loss']  # validation loss
ty = history_2_layers.history['loss']      # train loss
plt_dynamic(x, vy, ty,ax)
```

```
Test score: 0.06521892226291966
Test accuracy: 0.9811
```



3 hidden layers

```
In [0]: # start building a model
        model_3_layers = Sequential()


        model_3_layers.add(Dense(400, activation='relu', input_shape=(input_dim,), kernel_initi
        model_3_layers.add(BatchNormalization())
```

```
        model_3_layers.add(Dropout(0.5))                    #dropout

        model_3_layers.add(Dense(300, activation='relu', kernel_initializer=he_normal()))
        model_3_layers.add(BatchNormalization())
        model_3_layers.add(Dropout(0.5))

        model_3_layers.add(Dense(100, activation='relu', kernel_initializer=he_normal()))
        model_3_layers.add(BatchNormalization())
        model_3_layers.add(Dropout(0.5))

        model_3_layers.add(Dense(output_dim, activation='softmax'))

        # model Summary
        print("Model Summary :- \n",model_3_layers.summary())

        # Compiling the model
        model_3_layers.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['ac

        #training model
        history_3_layers = model_3_layers.fit(X_train, Y_train, batch_size=batch_size, epochs=

        score = model_3_layers.evaluate(X_test, Y_test)
        print('Test loss:', score[0])
        print('Test accuracy:', score[1])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_37 (Dense)             (None, 400)               314000
_________________________________________________________________
batch_normalization_21 (Batc (None, 400)               1600
_________________________________________________________________
dropout_28 (Dropout)         (None, 400)               0
_________________________________________________________________
dense_38 (Dense)             (None, 300)               120300
_________________________________________________________________
batch_normalization_22 (Batc (None, 300)               1200
_________________________________________________________________
dropout_29 (Dropout)         (None, 300)               0
_________________________________________________________________
dense_39 (Dense)             (None, 100)               30100
_________________________________________________________________
batch_normalization_23 (Batc (None, 100)               400
_________________________________________________________________
dropout_30 (Dropout)         (None, 100)               0
_________________________________________________________________
dense_40 (Dense)             (None, 10)                1010
=================================================================
```

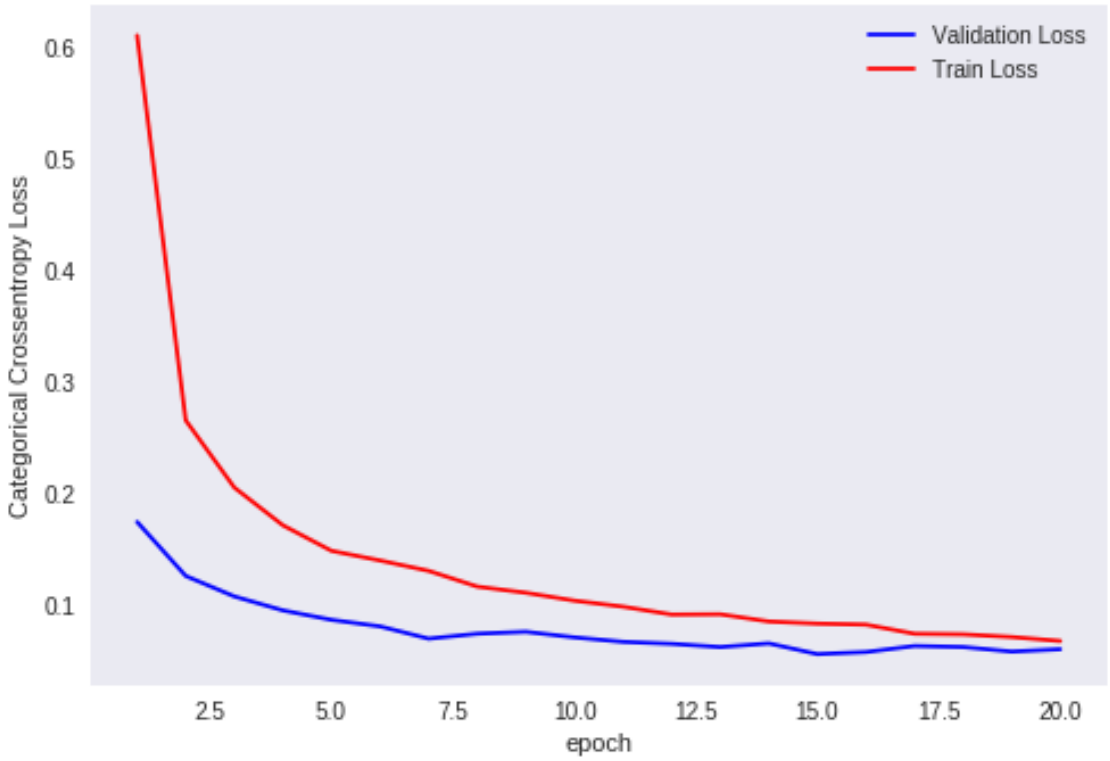
```
Total params: 468,610
Trainable params: 467,010
Non-trainable params: 1,600
_________________________________________________________________
Model Summary :-
 None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 11s 177us/step - loss: 0.6104 - acc: 0.8148 - va
Epoch 2/20
60000/60000 [==============================] - 9s 144us/step - loss: 0.2658 - acc: 0.9213 - val
Epoch 3/20
60000/60000 [==============================] - 9s 145us/step - loss: 0.2056 - acc: 0.9401 - val
Epoch 4/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.1718 - acc: 0.9491 - val
Epoch 5/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.1489 - acc: 0.9567 - val
Epoch 6/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.1403 - acc: 0.9588 - val
Epoch 7/20
60000/60000 [==============================] - 9s 145us/step - loss: 0.1310 - acc: 0.9613 - val
Epoch 8/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.1169 - acc: 0.9653 - val
Epoch 9/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.1114 - acc: 0.9670 - val
Epoch 10/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.1043 - acc: 0.9690 - val
Epoch 11/20
60000/60000 [==============================] - 9s 144us/step - loss: 0.0989 - acc: 0.9704 - val
Epoch 12/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0919 - acc: 0.9731 - val
Epoch 13/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.0921 - acc: 0.9729 - val
Epoch 14/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0857 - acc: 0.9748 - val
Epoch 15/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0838 - acc: 0.9746 - val
Epoch 16/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0828 - acc: 0.9762 - val
Epoch 17/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0747 - acc: 0.9774 - val
Epoch 18/20
60000/60000 [==============================] - 9s 144us/step - loss: 0.0742 - acc: 0.9778 - val
Epoch 19/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.0718 - acc: 0.9784 - val
Epoch 20/20
60000/60000 [==============================] - 9s 149us/step - loss: 0.0682 - acc: 0.9791 - val
10000/10000 [==============================] - 1s 84us/step
```

```
Test loss: 0.060828073866979686
Test accuracy: 0.9821
```

```
In [0]: fig,ax = plt.subplots(1,1)
        ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

        x = list(range(1,nb_epoch+1))


        vy = history_3_layers.history['val_loss']  # validation loss
        ty = history_3_layers.history['loss']      # train loss
        plt_dynamic(x, vy, ty,ax)
```



5 Hidden Layers

```
In [0]: from keras.initializers import he_normal
        model_5_layers = Sequential()

        model_5_layers.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initi
        model_5_layers.add(Dropout(0.25))

        model_5_layers.add(Dense(256 , activation = 'relu' , kernel_initializer = he_normal(se
        model_5_layers.add(Dropout(0.25))
```

```
        # batch Normalization
        model_5_layers.add(BatchNormalization())

        model_5_layers.add(Dense(128 , activation = 'relu' , kernel_initializer = he_normal(see
        model_5_layers.add(Dropout(0.25))

        model_5_layers.add(Dense(64 , activation = 'relu' , kernel_initializer = he_normal(seed
        model_5_layers.add(Dropout(0.25))

        model_5_layers.add(BatchNormalization())

        model_5_layers.add(Dense(32 , activation = 'relu' , kernel_initializer = he_normal(seed
        model_5_layers.add(Dropout(0.25))

        model_5_layers.add(BatchNormalization())

        model_5_layers.add(Dense(output_dim, activation='softmax'))

        # model Summary
        print("Model Summary :- \n",model_5_layers.summary())

        # Compiling the model
        model_5_layers.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['ac

        #training model
        history_5_layers = model_5_layers.fit(X_train, Y_train, batch_size=batch_size, epochs=

        score = model_5_layers.evaluate(X_test, Y_test)
        print('Test loss:', score[0])
        print('Test accuracy:', score[1])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_53 (Dense)             (None, 512)               401920
_________________________________________________________________
dropout_41 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_54 (Dense)             (None, 256)               131328
_________________________________________________________________
dropout_42 (Dropout)         (None, 256)               0
_________________________________________________________________
batch_normalization_28 (Batc (None, 256)               1024
_________________________________________________________________
dense_55 (Dense)             (None, 128)               32896
_________________________________________________________________
dropout_43 (Dropout)         (None, 128)               0
```

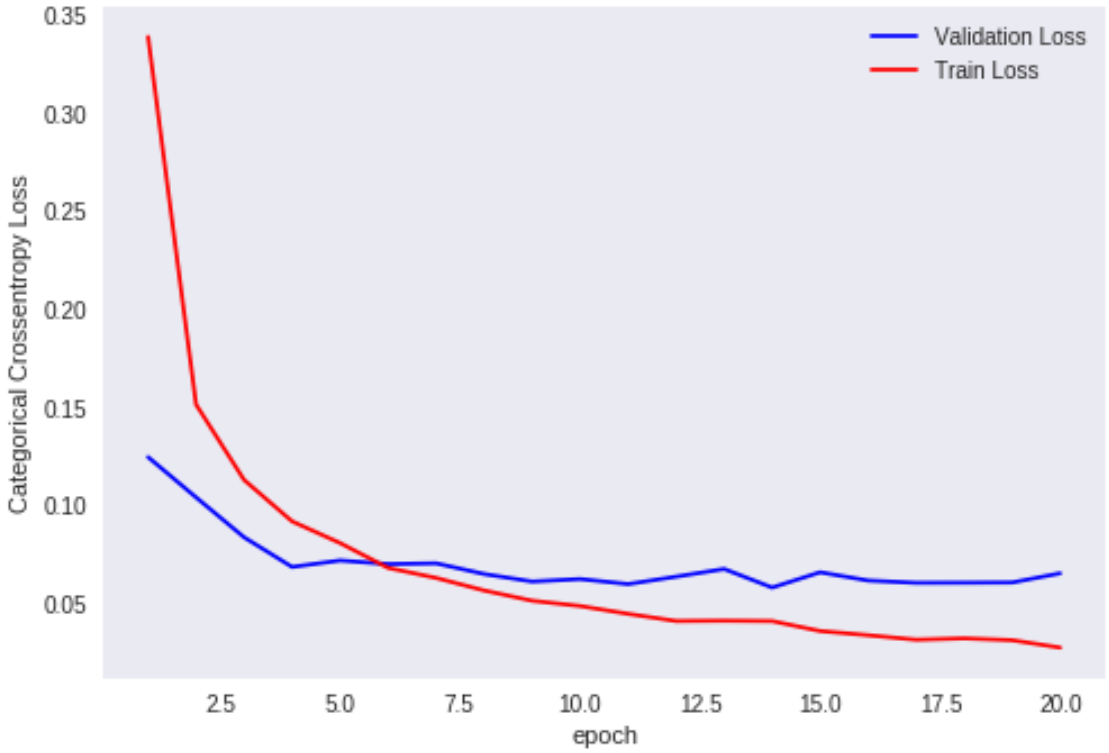
```
_________________________________________________________________
dense_56 (Dense)             (None, 64)                8256
_________________________________________________________________
dropout_44 (Dropout)         (None, 64)                0
_________________________________________________________________
batch_normalization_29 (Batc (None, 64)                256
_________________________________________________________________
dense_57 (Dense)             (None, 32)                2080
_________________________________________________________________
dropout_45 (Dropout)         (None, 32)                0
_________________________________________________________________
batch_normalization_30 (Batc (None, 32)                128
_________________________________________________________________
dense_58 (Dense)             (None, 10)                330
=================================================================
Total params: 578,218
Trainable params: 577,514
Non-trainable params: 704
_________________________________________________________________
Model Summary :-
 None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 13s 209us/step - loss: 0.7485 - acc: 0.7832 - va
Epoch 2/20
60000/60000 [==============================] - 10s 162us/step - loss: 0.2655 - acc: 0.9332 - va
Epoch 3/20
60000/60000 [==============================] - 10s 165us/step - loss: 0.1964 - acc: 0.9526 - va
Epoch 4/20
60000/60000 [==============================] - 10s 165us/step - loss: 0.1590 - acc: 0.9621 - va
Epoch 5/20
60000/60000 [==============================] - 10s 167us/step - loss: 0.1342 - acc: 0.9674 - va
Epoch 6/20
60000/60000 [==============================] - 10s 165us/step - loss: 0.1195 - acc: 0.9713 - va
Epoch 7/20
60000/60000 [==============================] - 10s 162us/step - loss: 0.1085 - acc: 0.9738 - va
Epoch 8/20
60000/60000 [==============================] - 10s 163us/step - loss: 0.0981 - acc: 0.9767 - va
Epoch 9/20
60000/60000 [==============================] - 10s 161us/step - loss: 0.0929 - acc: 0.9782 - va
Epoch 10/20
60000/60000 [==============================] - 10s 161us/step - loss: 0.0820 - acc: 0.9798 - va
Epoch 11/20
60000/60000 [==============================] - 10s 164us/step - loss: 0.0763 - acc: 0.9814 - va
Epoch 12/20
60000/60000 [==============================] - 10s 168us/step - loss: 0.0727 - acc: 0.9828 - va
Epoch 13/20
60000/60000 [==============================] - 10s 165us/step - loss: 0.0674 - acc: 0.9836 - va
```

```
Epoch 14/20
60000/60000 [==============================] - 10s 165us/step - loss: 0.0638 - acc: 0.9850 - va
Epoch 15/20
60000/60000 [==============================] - 10s 167us/step - loss: 0.0587 - acc: 0.9857 - va
Epoch 16/20
60000/60000 [==============================] - 10s 166us/step - loss: 0.0559 - acc: 0.9867 - va
Epoch 17/20
60000/60000 [==============================] - 10s 164us/step - loss: 0.0557 - acc: 0.9866 - va
Epoch 18/20
60000/60000 [==============================] - 10s 163us/step - loss: 0.0530 - acc: 0.9871 - va
Epoch 19/20
60000/60000 [==============================] - 10s 164us/step - loss: 0.0508 - acc: 0.9873 - va
Epoch 20/20
60000/60000 [==============================] - 10s 165us/step - loss: 0.0478 - acc: 0.9885 - va
10000/10000 [==============================] - 1s 89us/step
Test loss: 0.07591400257602218
Test accuracy: 0.9822
```

```
In [0]: fig,ax = plt.subplots(1,1)
        ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

        x = list(range(1,nb_epoch+1))


        vy = history_2_layers.history['val_loss']  # validation loss
        ty = history_2_layers.history['loss']      # train loss
        plt_dynamic(x, vy, ty,ax)
```



OBSERVATION: Deep MLPs are more prone to overfitting hence we can see that validation loss is more compare to train loss as no. of epoch is increasing

```
In [0]: from prettytable import PrettyTable

In [0]: x = PrettyTable()

In [0]: x.field_names = ["No. of layers", "Test Loss", "Test Accuracy"]
        x.add_row(['2' , '0.06521','9811'])
        x.add_row(['3' , '0.06082','0.9821'])
        x.add_row(['5' , '0.0759','0.9822'])

In [0]: print(x)
```

```
+---------------+-----------+---------------+
| No. of layers | Test Loss | Test Accuracy |
+---------------+-----------+---------------+
|       2       |  0.06521  |      9811     |
|       3       |  0.06082  |     0.9821    |
|       5       |   0.0759  |     0.9822    |
+---------------+-----------+---------------+
```

Conclusion

1. Increasing number of hidden layers does not always improve the performance of model
2. Deep MLP are prone to overfitting while shallow MLP are prone to underfitting
3. Overfitting can be avoid using dropout