MLP_Architectures_On_MNIST_Data

December 13, 2018

1 Different MLP Architecture On MNIST Dataset

```
In [0]: # if you keras is not using tensorflow as backend set "KERAS_BACKEND=tensorflow" use th
        from keras.utils import np_utils
        from keras.datasets import mnist
        import seaborn as sns
        from keras.initializers import RandomNormal
```

```
In [0]: # the data, shuffled and split between train and test sets
        (X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
In [0]: 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 (28, 28)
Number of training examples : 10000 and each image is of shape (28, 28)
```

```
In [0]: # 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 [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]: # An example data point
        print(X_train[0])
```

```
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   3  18  18  18 126 136 175  26 166 255
 247 127   0   0   0   0   0   0   0   0   0   0   0   0  30  36  94 154
 170 253 253 253 253 253 225 172 253 242 195  64   0   0   0   0   0   0
   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251  93  82
  82  56  39   0   0   0   0   0   0   0   0   0   0   0   0  18 219 253
 253 253 253 253 198 182 247 241   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0  14   1 154 253  90   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0  11 190 253  70   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  35 241
 225 160 108   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0  81 240 253 253 119  25   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  45 186 253 253 150  27   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252 253 187
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0 249 253 249  64   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253
 253 207   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0  39 148 229 253 253 253 250 182   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253
 253 201  78   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  23  66 213 253 253 253 253 198  81   2   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0  18 171 219 253 253 253 253 195
  80   9   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  55 172 226 253 253 253 253 244 133  11   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0 136 253 253 253 212 135 132  16
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
```

In [0]:
```
# if we observe the above matrix each cell is having a value between 0-255
# before we move to apply machine learning algorithms lets try to normalize the data
```

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

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

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

```
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.01176471 0.07058824 0.07058824 0.07058824
 0.49411765 0.53333333 0.68627451 0.10196078 0.65098039 1.
 0.96862745 0.49803922 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.11764706 0.14117647 0.36862745 0.60392157
 0.66666667 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
 0.88235294 0.6745098  0.99215686 0.94901961 0.76470588 0.25098039
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.19215686
 0.93333333 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
 0.99215686 0.99215686 0.99215686 0.98431373 0.36470588 0.32156863
 0.32156863 0.21960784 0.15294118 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.07058824 0.85882353 0.99215686
 0.99215686 0.99215686 0.99215686 0.99215686 0.77647059 0.71372549
```

```
0.96862745 0.94509804 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.31372549 0.61176471 0.41960784 0.99215686
0.99215686 0.80392157 0.04313725 0.         0.16862745 0.60392157
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.05490196 0.00392157 0.60392157 0.99215686 0.35294118
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.54509804 0.99215686 0.74509804 0.00784314 0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.04313725
0.74509804 0.99215686 0.2745098  0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.1372549  0.94509804
0.88235294 0.62745098 0.42352941 0.00392157 0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.31764706 0.94117647 0.99215686
0.99215686 0.46666667 0.09803922 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.17647059 0.72941176 0.99215686 0.99215686
0.58823529 0.10588235 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.0627451  0.36470588 0.98823529 0.99215686 0.73333333
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.97647059 0.99215686 0.97647059 0.25098039 0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.18039216 0.50980392 0.71764706 0.99215686
0.99215686 0.81176471 0.00784314 0.         0.         0.
```

```
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.15294118 0.58039216
 0.89803922 0.99215686 0.99215686 0.99215686 0.98039216 0.71372549
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.09411765 0.44705882 0.86666667 0.99215686 0.99215686 0.99215686
 0.99215686 0.78823529 0.30588235 0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.09019608 0.25882353 0.83529412 0.99215686
 0.99215686 0.99215686 0.99215686 0.77647059 0.31764706 0.00784314
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.07058824 0.67058824
 0.85882353 0.99215686 0.99215686 0.99215686 0.99215686 0.76470588
 0.31372549 0.03529412 0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.21568627 0.6745098  0.88627451 0.99215686 0.99215686 0.99215686
 0.99215686 0.95686275 0.52156863 0.04313725 0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.53333333 0.99215686
 0.99215686 0.99215686 0.83137255 0.52941176 0.51764706 0.0627451
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.        ]
```

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)
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]: # this function is used draw Categorical Crossentropy Loss VS No. of epochs plot
        def plt_dynamic(x, vy, ty):
          plt.figure(figsize=(10,5))
          plt.plot(x, vy, 'b', label="Validation Loss")
          plt.plot(x, ty, 'r', label="Train Loss")
          plt.xlabel('Epochs')
          plt.ylabel('Categorical Crossentropy Loss')
          plt.title('\nCategorical Crossentropy Loss VS Epochs')
          plt.legend()
          plt.grid()
          plt.show()
```

1.1 (1). Softmax Classifier with 2 hidden layers

1.1.1 (1.a) Without dropout and Batch normalization

```
In [0]: from keras.models import Sequential
        from keras.layers import Dense, Activation
        from keras.initializers import he_normal

        # some model parameters

        output_dim = 10
        input_dim = X_train.shape[1]

        batch_size = 128
        nb_epoch = 20

        # Initialising model
        model_2 = Sequential()

        # Adding first hidden layer
        model_2.add(Dense(364, activation='relu', input_shape=(input_dim,), kernel_initializer=

        # Adding second hidden layer
        model_2.add(Dense(52, activation='relu', kernel_initializer=he_normal(seed=None)))
```

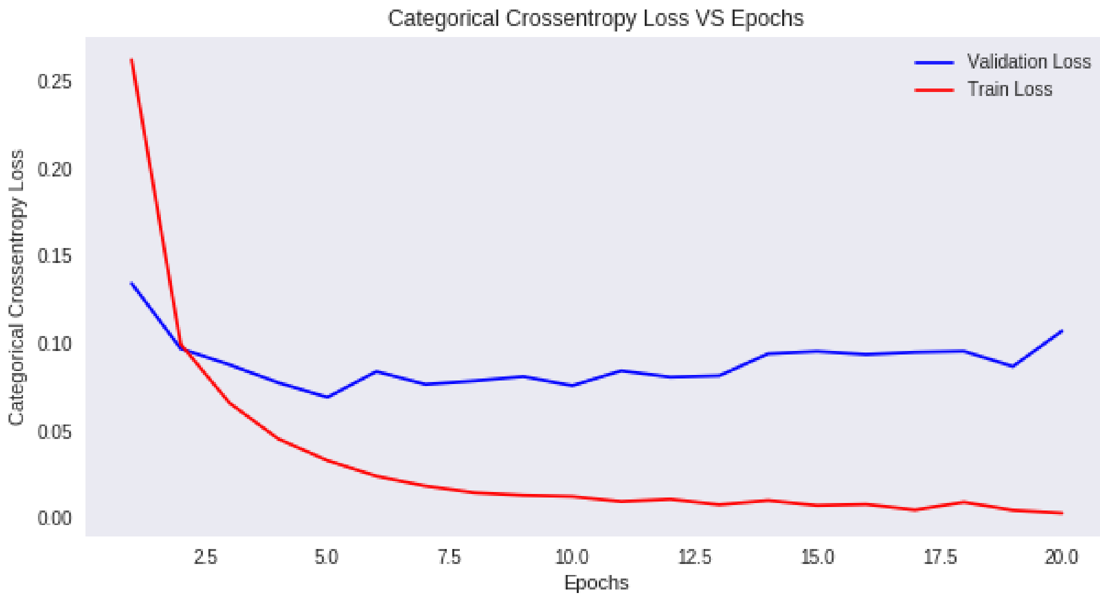
```python
# Adding output layer
model_2.add(Dense(output_dim, activation='softmax'))

# Printing model Summary
print("Model Summary :- \n",model_2.summary())

# Compiling the model
model_2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'

# Fitting the data to the model
history_2 = model_2.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verb
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_10 (Dense)             (None, 364)               285740
_________________________________________________________________
dense_11 (Dense)             (None, 52)                18980
_________________________________________________________________
dense_12 (Dense)             (None, 10)                530
=================================================================
Total params: 305,250
Trainable params: 305,250
Non-trainable params: 0
_________________________________________________________________
Model Summary :-
 None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 111us/step - loss: 0.2625 - acc: 0.9249 - va
Epoch 2/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0996 - acc: 0.9705 - val
Epoch 3/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0660 - acc: 0.9807 - val
Epoch 4/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.0454 - acc: 0.9862 - val
Epoch 5/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0331 - acc: 0.9902 - val
Epoch 6/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.0241 - acc: 0.9927 - val
Epoch 7/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.0184 - acc: 0.9942 - val
Epoch 8/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0146 - acc: 0.9954 - val
Epoch 9/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0131 - acc: 0.9959 - val
Epoch 10/20
```

```
60000/60000 [==============================] - 6s 92us/step - loss: 0.0124 - acc: 0.9964 - val_
Epoch 11/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.0095 - acc: 0.9969 - val_
Epoch 12/20
60000/60000 [==============================] - 5s 92us/step - loss: 0.0109 - acc: 0.9965 - val_
Epoch 13/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.0077 - acc: 0.9976 - val_
Epoch 14/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.0101 - acc: 0.9963 - val_
Epoch 15/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0073 - acc: 0.9975 - val_
Epoch 16/20
60000/60000 [==============================] - 5s 92us/step - loss: 0.0079 - acc: 0.9972 - val_
Epoch 17/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.0047 - acc: 0.9986 - val_
Epoch 18/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.0092 - acc: 0.9967 - val_
Epoch 19/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0045 - acc: 0.9984 - val_
Epoch 20/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0029 - acc: 0.9991 - val_
```

In [0]:
```python
import matplotlib.pyplot as plt
# Evaluating the model
score = model_2.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

# Test and train accuracy of the model
model_2_test = score[1]
model_2_train = history_2.history['acc']

# Plotting Train and Test Loss VS no. of epochs
# list of epoch numbers
x = list(range(1,nb_epoch+1))

# Validation loss
vy = history_2.history['val_loss']
# Training loss
ty = history_2.history['loss']

# Calling the function to draw the plot
plt_dynamic(x, vy, ty)
```

```
Test score: 0.10713244834685151
Test accuracy: 0.9788
```



1.1.2 (1.b) With dropout and Batch Normalization

```
In [0]: from keras.layers.normalization import BatchNormalization
        from keras.layers import Dropout

        # Initialising model
        model_2d = Sequential()

        # Adding first hidden layer
        model_2d.add(Dense(364, activation='relu', input_shape=(input_dim,), kernel_initialize
        # Adding Batch Normalization
        model_2d.add(BatchNormalization())
        # Adding dropout to first hidden layer
        model_2d.add(Dropout(0.5))

        # Adding second hidden layer
        model_2d.add(Dense(52, activation='relu', kernel_initializer=he_normal(seed=None)))
        # Adding Batch Normalization
        model_2d.add(BatchNormalization())
        # Adding dropout to second hidden layer
        model_2d.add(Dropout(0.5))

        # Adding output layer
        model_2d.add(Dense(output_dim, activation='softmax'))

        # Printing model Summary
```

```python
print("Model Summary :- \n",model_2d.summary())

# Compiling the model
model_2d.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy

# Fitting the data to the model
history_2d = model_2d.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ve
```

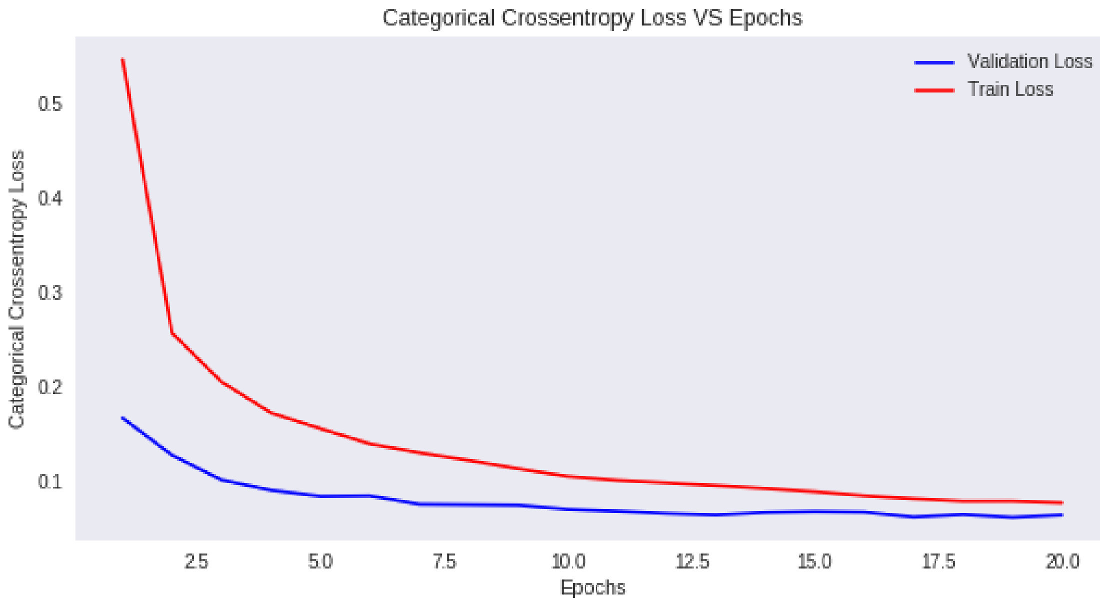
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_16 (Dense)             (None, 364)               285740
_________________________________________________________________
batch_normalization_3 (Batch (None, 364)               1456
_________________________________________________________________
dropout_3 (Dropout)          (None, 364)               0
_________________________________________________________________
dense_17 (Dense)             (None, 52)                18980
_________________________________________________________________
batch_normalization_4 (Batch (None, 52)                208
_________________________________________________________________
dropout_4 (Dropout)          (None, 52)                0
_________________________________________________________________
dense_18 (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 [==============================] - 10s 169us/step - loss: 0.5473 - acc: 0.8364 - va
Epoch 2/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.2572 - acc: 0.9249 - val
Epoch 3/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.2055 - acc: 0.9410 - val
Epoch 4/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.1726 - acc: 0.9498 - val
Epoch 5/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.1558 - acc: 0.9552 - val
Epoch 6/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.1397 - acc: 0.9592 - val
Epoch 7/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.1302 - acc: 0.9625 - val
Epoch 8/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.1221 - acc: 0.9650 - val
```

```
Epoch 9/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.1134 - acc: 0.9668 - va
Epoch 10/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.1052 - acc: 0.9693 - va
Epoch 11/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.1010 - acc: 0.9696 - va
Epoch 12/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.0983 - acc: 0.9712 - va
Epoch 13/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.0954 - acc: 0.9724 - va
Epoch 14/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.0923 - acc: 0.9728 - va
Epoch 15/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.0889 - acc: 0.9735 - va
Epoch 16/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.0846 - acc: 0.9746 - va
Epoch 17/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.0814 - acc: 0.9758 - va
Epoch 18/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.0789 - acc: 0.9763 - va
Epoch 19/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.0789 - acc: 0.9759 - va
Epoch 20/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.0771 - acc: 0.9773 - va
```

In [0]:
```python
# Evaluating the model
score = model_2d.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

# Test and train accuracy of the model
model_2d_test = score[1]
model_2d_train = history_2d.history['acc']

# Plotting Train and Test Loss VS no. of epochs
# list of epoch numbers
x = list(range(1,nb_epoch+1))

# Validation loss
vy = history_2d.history['val_loss']
# Training loss
ty = history_2d.history['loss']

# Calling the function to draw the plot
plt_dynamic(x, vy, ty)
```

```
Test score: 0.06424971719455207
```

```
Test accuracy: 0.9803
```



1.2 (2). Softmax Classifier with 3 hidden layers

1.2.1 (2.a) Without Dropout and Batch Normalization

```
In [0]: # Initialising model
        model_3 = Sequential()

        # Adding first hidden layer
        model_3.add(Dense(392, activation='relu', input_shape=(input_dim,), kernel_initializer=

        # Adding second hidden layer
        model_3.add(Dense(196, activation='relu', kernel_initializer=he_normal(seed=None)))

        # Adding third hidden layer
        model_3.add(Dense(98, activation='relu', kernel_initializer=he_normal(seed=None)))

        # Adding output layer
        model_3.add(Dense(output_dim, activation='softmax'))

        # Printing model Summary
        print(model_3.summary())

        # Compiling the model
        model_3.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy']
```

```python
# Fitting the data to the model
history_3 = model_3.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbo
```

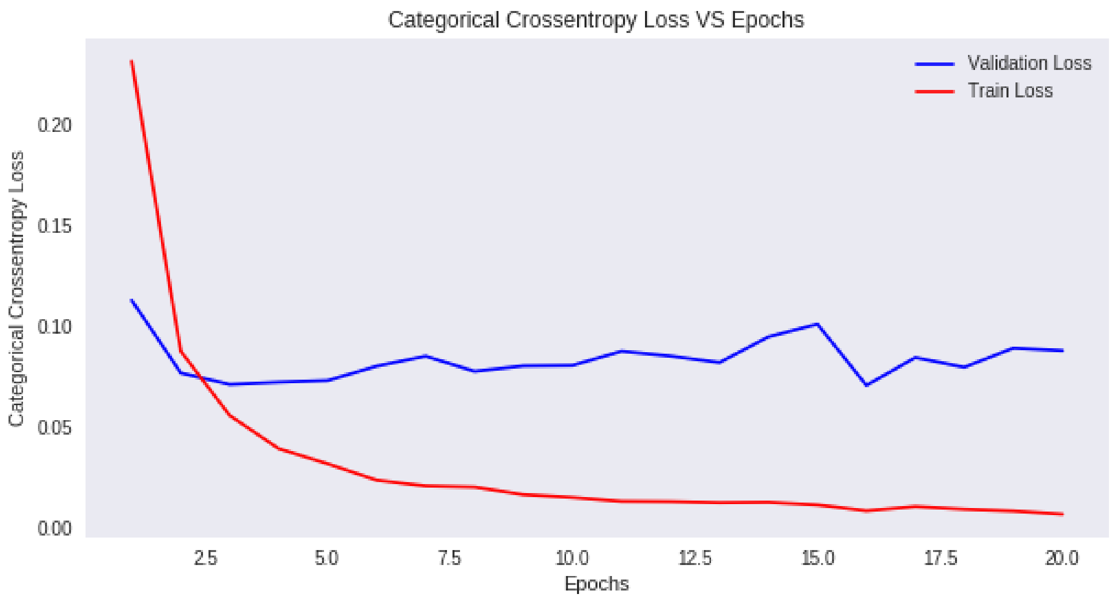
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_19 (Dense)             (None, 392)               307720
_________________________________________________________________
dense_20 (Dense)             (None, 196)               77028
_________________________________________________________________
dense_21 (Dense)             (None, 98)                19306
_________________________________________________________________
dense_22 (Dense)             (None, 10)                990
=================================================================
Total params: 405,044
Trainable params: 405,044
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 8s 128us/step - loss: 0.2318 - acc: 0.9323 - va
Epoch 2/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.0880 - acc: 0.9733 - va
Epoch 3/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0562 - acc: 0.9825 - va
Epoch 4/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0397 - acc: 0.9873 - va
Epoch 5/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.0322 - acc: 0.9895 - va
Epoch 6/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.0241 - acc: 0.9923 - va
Epoch 7/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0212 - acc: 0.9927 - va
Epoch 8/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.0206 - acc: 0.9932 - va
Epoch 9/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0169 - acc: 0.9947 - va
Epoch 10/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.0155 - acc: 0.9944 - va
Epoch 11/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.0136 - acc: 0.9955 - va
Epoch 12/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0134 - acc: 0.9953 - va
Epoch 13/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0129 - acc: 0.9955 - va
Epoch 14/20
```

```
60000/60000 [==============================] - 7s 118us/step - loss: 0.0131 - acc: 0.9962 - val
Epoch 15/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.0118 - acc: 0.9963 - val
Epoch 16/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0089 - acc: 0.9974 - val
Epoch 17/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0109 - acc: 0.9967 - val
Epoch 18/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.0096 - acc: 0.9973 - val
Epoch 19/20
60000/60000 [==============================] - 7s 117us/step - loss: 0.0087 - acc: 0.9971 - val
Epoch 20/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0072 - acc: 0.9977 - val
```

In [0]:

```
# Evaluating the model
score = model_3.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

# Test and train accuracy of the model
model_3_test = score[1]
model_3_train = history_3.history['acc']

# Plotting Train and Test Loss VS no. of epochs
# list of epoch numbers
x = list(range(1,nb_epoch+1))

# Validation loss
vy = history_3.history['val_loss']
# Training loss
ty = history_3.history['loss']

# Calling the function to draw the plot
plt_dynamic(x, vy, ty)
```

```
Test score: 0.08825439285551874
Test accuracy: 0.984
```



1.2.2 (2.b) With Droput and Batch Normalization

In [0]:
```
model_3d = Sequential()

# Adding first hidden layer
model_3d.add(Dense(392, activation='relu', input_shape=(input_dim,), kernel_initialize
# Adding Batch Normalization
model_3d.add(BatchNormalization())
# Adding dropout
model_3d.add(Dropout(0.5))

# Adding second hidden layer
model_3d.add(Dense(196, activation='relu', kernel_initializer=he_normal(seed=None)))
# Adding Batch Normalization
model_3d.add(BatchNormalization())
# Adding dropout
model_3d.add(Dropout(0.5))

# Adding third hidden layer
model_3d.add(Dense(98, activation='relu', kernel_initializer=he_normal(seed=None)))
# Adding Batch Normalization
model_3d.add(BatchNormalization())
# Adding dropout
model_3d.add(Dropout(0.5))

# Adding output layer
```

```python
model_3d.add(Dense(output_dim, activation='softmax'))

# Printing model Summary
print(model_3d.summary())

# Compiling the model
model_3d.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy

# Fitting the data to the model
history_3d = model_3d.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ve
```

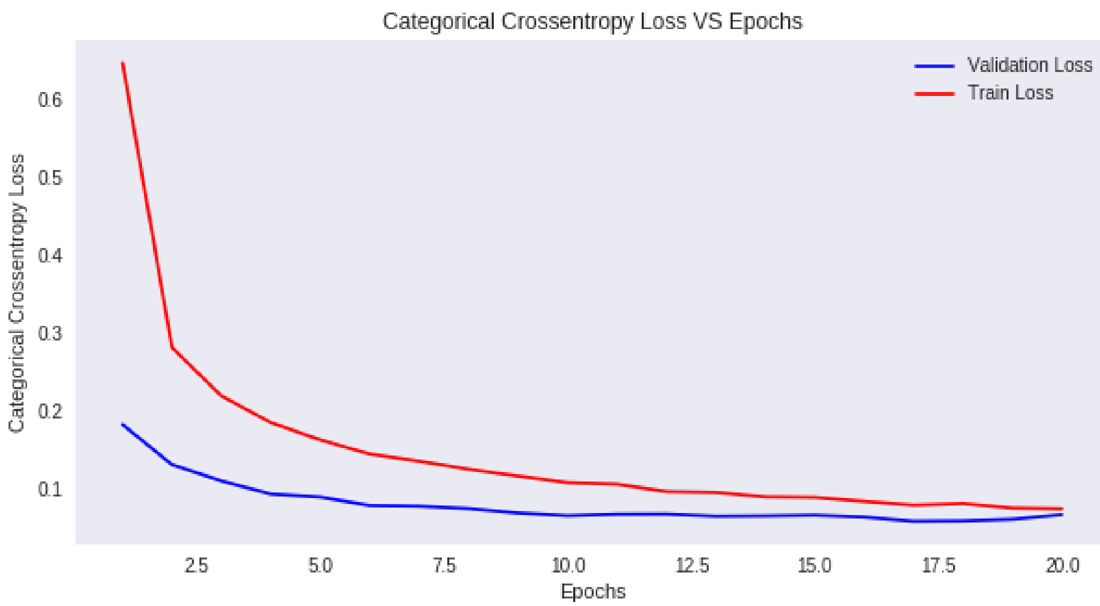
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_23 (Dense)             (None, 392)               307720
_________________________________________________________________
batch_normalization_5 (Batch (None, 392)               1568
_________________________________________________________________
dropout_5 (Dropout)          (None, 392)               0
_________________________________________________________________
dense_24 (Dense)             (None, 196)               77028
_________________________________________________________________
batch_normalization_6 (Batch (None, 196)               784
_________________________________________________________________
dropout_6 (Dropout)          (None, 196)               0
_________________________________________________________________
dense_25 (Dense)             (None, 98)                19306
_________________________________________________________________
batch_normalization_7 (Batch (None, 98)                392
_________________________________________________________________
dropout_7 (Dropout)          (None, 98)                0
_________________________________________________________________
dense_26 (Dense)             (None, 10)                990
=================================================================
Total params: 407,788
Trainable params: 406,416
Non-trainable params: 1,372
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 14s 237us/step - loss: 0.6456 - acc: 0.8009 - va
Epoch 2/20
60000/60000 [==============================] - 14s 225us/step - loss: 0.2815 - acc: 0.9186 - va
Epoch 3/20
60000/60000 [==============================] - 13s 213us/step - loss: 0.2195 - acc: 0.9367 - va
Epoch 4/20
60000/60000 [==============================] - 12s 204us/step - loss: 0.1855 - acc: 0.9470 - va
```

```
Epoch 5/20
60000/60000 [==============================] - 12s 205us/step - loss: 0.1633 - acc: 0.9535 - va
Epoch 6/20
60000/60000 [==============================] - 12s 201us/step - loss: 0.1453 - acc: 0.9578 - va
Epoch 7/20
60000/60000 [==============================] - 12s 193us/step - loss: 0.1361 - acc: 0.9613 - va
Epoch 8/20
60000/60000 [==============================] - 12s 206us/step - loss: 0.1257 - acc: 0.9630 - va
Epoch 9/20
60000/60000 [==============================] - 12s 193us/step - loss: 0.1170 - acc: 0.9654 - va
Epoch 10/20
60000/60000 [==============================] - 12s 199us/step - loss: 0.1086 - acc: 0.9689 - va
Epoch 11/20
60000/60000 [==============================] - 12s 196us/step - loss: 0.1068 - acc: 0.9688 - va
Epoch 12/20
60000/60000 [==============================] - 11s 189us/step - loss: 0.0972 - acc: 0.9713 - va
Epoch 13/20
60000/60000 [==============================] - 11s 191us/step - loss: 0.0961 - acc: 0.9718 - va
Epoch 14/20
60000/60000 [==============================] - 11s 184us/step - loss: 0.0904 - acc: 0.9737 - va
Epoch 15/20
60000/60000 [==============================] - 12s 193us/step - loss: 0.0898 - acc: 0.9734 - va
Epoch 16/20
60000/60000 [==============================] - 12s 195us/step - loss: 0.0847 - acc: 0.9747 - va
Epoch 17/20
60000/60000 [==============================] - 11s 188us/step - loss: 0.0799 - acc: 0.9757 - va
Epoch 18/20
60000/60000 [==============================] - 12s 202us/step - loss: 0.0819 - acc: 0.9757 - va
Epoch 19/20
60000/60000 [==============================] - 12s 202us/step - loss: 0.0761 - acc: 0.9773 - va
Epoch 20/20
60000/60000 [==============================] - 11s 188us/step - loss: 0.0752 - acc: 0.9769 - va
```

In [0]:
```
# Evaluating the model
score = model_3d.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

# Test and train accuracy of the model
model_3d_test = score[1]
model_3d_train = history_3d.history['acc']

# Plotting Train and Test Loss VS no. of epochs
# list of epoch numbers
x = list(range(1,nb_epoch+1))

# Validation loss
```

```
vy = history_3d.history['val_loss']
# Training loss
ty = history_3d.history['loss']

# Calling the function to draw the plot
plt_dynamic(x, vy, ty)
```

```
Test score: 0.06754246267098933
Test accuracy: 0.9816
```



1.3 3). Softmax Classifier with 5 hidden layers

1.3.1 3.a) Without Dropout and Batch Normalization

```
In [0]: # Initialising model
        model_5 = Sequential()

        # Adding first hidden layer
        model_5.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=

        # Adding second hidden layer
        model_5.add(Dense(256, activation='relu', kernel_initializer=he_normal(seed=None)))

        # Adding third hidden layer
        model_5.add(Dense(128, activation='relu', kernel_initializer=he_normal(seed=None)))
```

```python
# Adding fourth hidden layer
model_5.add(Dense(64, activation='relu', kernel_initializer=he_normal(seed=None)))

# Adding fifth hidden layer
model_5.add(Dense(32, activation='relu', kernel_initializer=he_normal(seed=None)))

# Adding output layer
model_5.add(Dense(output_dim, activation='softmax'))

# Printing model Summary
print(model_5.summary())

# Compiling the model
model_5.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy']

# Fitting the data to the model
history_5 = model_5.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbo
```

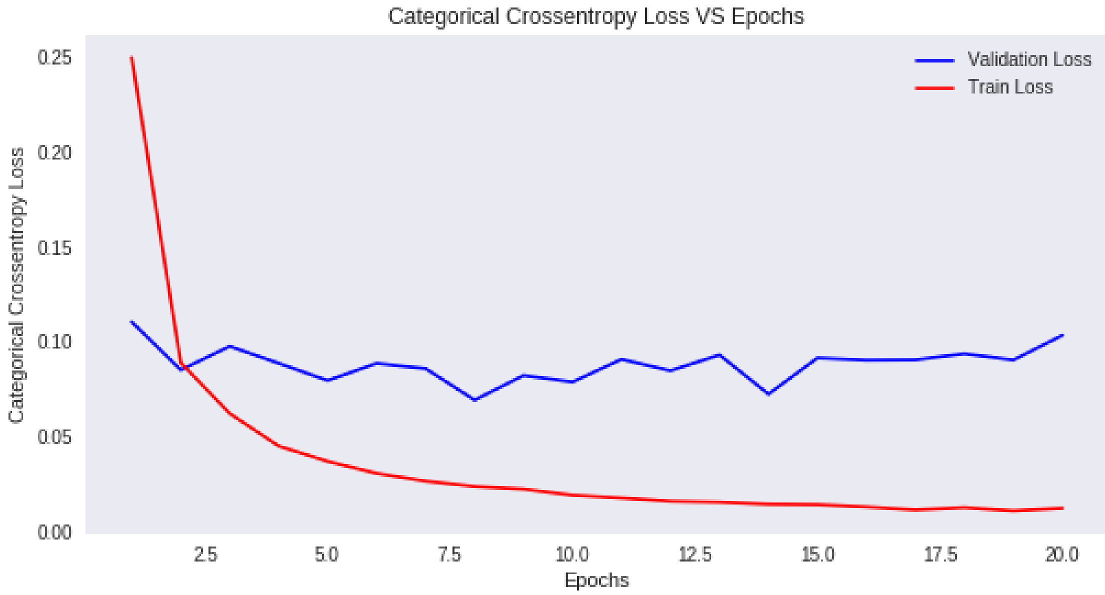
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_27 (Dense)             (None, 512)               401920    
_________________________________________________________________
dense_28 (Dense)             (None, 256)               131328    
_________________________________________________________________
dense_29 (Dense)             (None, 128)               32896     
_________________________________________________________________
dense_30 (Dense)             (None, 64)                8256      
_________________________________________________________________
dense_31 (Dense)             (None, 32)                2080      
_________________________________________________________________
dense_32 (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 1/20
60000/60000 [==============================] - 13s 213us/step - loss: 0.2497 - acc: 0.9237 - va
Epoch 2/20
60000/60000 [==============================] - 11s 191us/step - loss: 0.0890 - acc: 0.9730 - va
Epoch 3/20
60000/60000 [==============================] - 11s 188us/step - loss: 0.0623 - acc: 0.9803 - va
Epoch 4/20
60000/60000 [==============================] - 12s 192us/step - loss: 0.0451 - acc: 0.9857 - va
Epoch 5/20
```

```
60000/60000 [==============================] - 11s 191us/step - loss: 0.0371 - acc: 0.9882 - va
Epoch 6/20
60000/60000 [==============================] - 11s 189us/step - loss: 0.0307 - acc: 0.9905 - va
Epoch 7/20
60000/60000 [==============================] - 11s 188us/step - loss: 0.0266 - acc: 0.9911 - va
Epoch 8/20
60000/60000 [==============================] - 13s 212us/step - loss: 0.0238 - acc: 0.9919 - va
Epoch 9/20
60000/60000 [==============================] - 11s 191us/step - loss: 0.0224 - acc: 0.9931 - va
Epoch 10/20
60000/60000 [==============================] - 12s 193us/step - loss: 0.0192 - acc: 0.9941 - va
Epoch 11/20
60000/60000 [==============================] - 11s 192us/step - loss: 0.0178 - acc: 0.9944 - va
Epoch 12/20
60000/60000 [==============================] - 11s 191us/step - loss: 0.0160 - acc: 0.9949 - va
Epoch 13/20
60000/60000 [==============================] - 11s 191us/step - loss: 0.0156 - acc: 0.9954 - va
Epoch 14/20
60000/60000 [==============================] - 12s 206us/step - loss: 0.0145 - acc: 0.9956 - va
Epoch 15/20
60000/60000 [==============================] - 11s 191us/step - loss: 0.0143 - acc: 0.9956 - va
Epoch 16/20
60000/60000 [==============================] - 12s 193us/step - loss: 0.0131 - acc: 0.9959 - va
Epoch 17/20
60000/60000 [==============================] - 12s 192us/step - loss: 0.0115 - acc: 0.9966 - va
Epoch 18/20
60000/60000 [==============================] - 12s 193us/step - loss: 0.0127 - acc: 0.9963 - va
Epoch 19/20
60000/60000 [==============================] - 12s 193us/step - loss: 0.0110 - acc: 0.9966 - va
Epoch 20/20
60000/60000 [==============================] - 12s 193us/step - loss: 0.0124 - acc: 0.9965 - va
```

In [0]:
```
# Evaluating the model
score = model_5.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

# Test and train accuracy of the model
model_5_test = score[1]
model_5_train = history_5.history['acc']

# Plotting Train and Test Loss VS no. of epochs
# list of epoch numbers
x = list(range(1,nb_epoch+1))

# Validation loss
vy = history_5.history['val_loss']
```

```
# Training loss
ty = history_5.history['loss']

# Calling the function to draw the plot
plt_dynamic(x, vy, ty)
```

```
Test score: 0.10349378544007559
Test accuracy: 0.9775
```



1.3.2 (3.b) With Dropout and Batch Normalisation

```
In [0]: # Initialising model
        model_5d = Sequential()

        # Adding first hidden layer
        model_5d.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initialize
        # Adding Batch Normalization
        model_5d.add(BatchNormalization())
        # Adding dropout
        model_5d.add(Dropout(0.5))

        # Adding second hidden layer
        model_5d.add(Dense(256, activation='relu', kernel_initializer=he_normal(seed=None)))
        # Adding Batch Normalization
        model_5d.add(BatchNormalization())
        # Adding dropout
```

```
model_5d.add(Dropout(0.5))

# Adding third hidden layer
model_5d.add(Dense(128, activation='relu', kernel_initializer=he_normal(seed=None)))
# Adding Batch Normalization
model_5d.add(BatchNormalization())
# Adding dropout
model_5d.add(Dropout(0.5))

# Adding fourth hidden layer
model_5d.add(Dense(64, activation='relu', kernel_initializer=he_normal(seed=None)))
# Adding Batch Normalization
model_5d.add(BatchNormalization())
# Adding dropout
model_5d.add(Dropout(0.5))

# Adding fifth hidden layer
model_5d.add(Dense(32, activation='relu', kernel_initializer=he_normal(seed=None)))
# Adding Batch Normalization
model_5d.add(BatchNormalization())
# Adding dropout
model_5d.add(Dropout(0.5))

# Adding output layer
model_5d.add(Dense(output_dim, activation='softmax'))

# Printing model Summary
print(model_5d.summary())

# Compiling the model
model_5d.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy

# Fitting the data to the model
history_5d = model_5d.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ve
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_33 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_8 (Batch (None, 512)               2048
_________________________________________________________________
dropout_8 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_34 (Dense)             (None, 256)               131328
_________________________________________________________________
batch_normalization_9 (Batch (None, 256)               1024
_________________________________________________________________
```

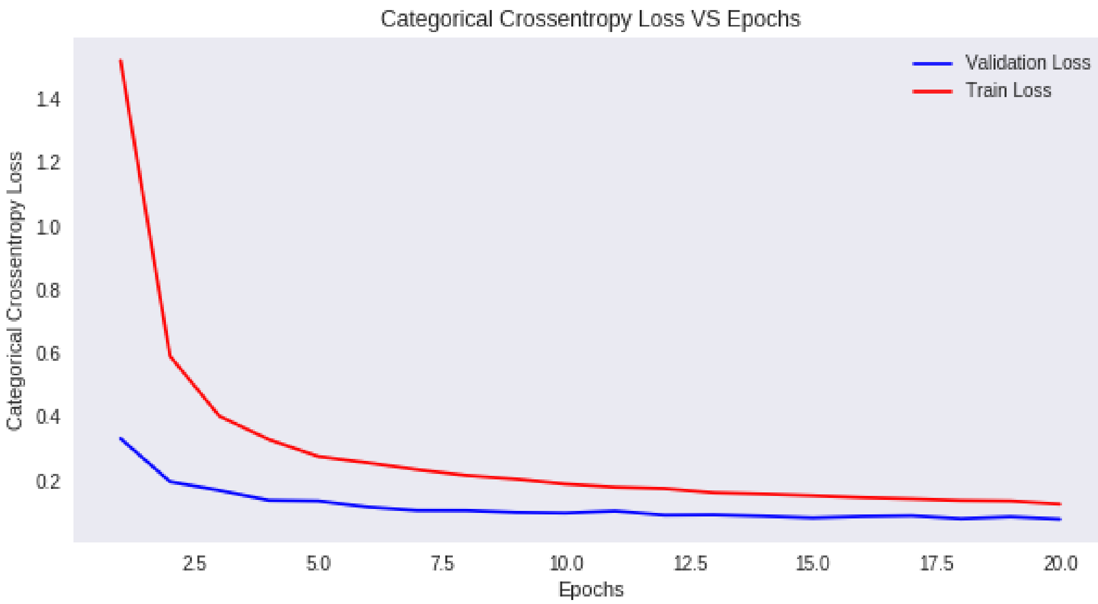
```
dropout_9 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_35 (Dense)             (None, 128)               32896
_________________________________________________________________
batch_normalization_10 (Batc (None, 128)               512
_________________________________________________________________
dropout_10 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_36 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_11 (Batc (None, 64)                256
_________________________________________________________________
dropout_11 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_37 (Dense)             (None, 32)                2080
_________________________________________________________________
batch_normalization_12 (Batc (None, 32)                128
_________________________________________________________________
dropout_12 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_38 (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 1/20
60000/60000 [==============================] - 18s 297us/step - loss: 1.5213 - acc: 0.5067 - va
Epoch 2/20
60000/60000 [==============================] - 15s 250us/step - loss: 0.5912 - acc: 0.8356 - va
Epoch 3/20
60000/60000 [==============================] - 15s 246us/step - loss: 0.4023 - acc: 0.8970 - va
Epoch 4/20
60000/60000 [==============================] - 15s 249us/step - loss: 0.3287 - acc: 0.9200 - va
Epoch 5/20
60000/60000 [==============================] - 15s 249us/step - loss: 0.2755 - acc: 0.9340 - va
Epoch 6/20
60000/60000 [==============================] - 14s 241us/step - loss: 0.2556 - acc: 0.9399 - va
Epoch 7/20
60000/60000 [==============================] - 15s 242us/step - loss: 0.2341 - acc: 0.9455 - va
Epoch 8/20
60000/60000 [==============================] - 14s 241us/step - loss: 0.2154 - acc: 0.9501 - va
Epoch 9/20
60000/60000 [==============================] - 15s 244us/step - loss: 0.2041 - acc: 0.9530 - va
Epoch 10/20
60000/60000 [==============================] - 15s 244us/step - loss: 0.1888 - acc: 0.9567 - va
```

```
Epoch 11/20
60000/60000 [==============================] - 16s 262us/step - loss: 0.1787 - acc: 0.9598 - va
Epoch 12/20
60000/60000 [==============================] - 15s 248us/step - loss: 0.1743 - acc: 0.9607 - va
Epoch 13/20
60000/60000 [==============================] - 15s 247us/step - loss: 0.1614 - acc: 0.9633 - va
Epoch 14/20
60000/60000 [==============================] - 15s 244us/step - loss: 0.1572 - acc: 0.9643 - va
Epoch 15/20
60000/60000 [==============================] - 15s 252us/step - loss: 0.1521 - acc: 0.9661 - va
Epoch 16/20
60000/60000 [==============================] - 15s 252us/step - loss: 0.1461 - acc: 0.9671 - va
Epoch 17/20
60000/60000 [==============================] - 15s 245us/step - loss: 0.1420 - acc: 0.9683 - va
Epoch 18/20
60000/60000 [==============================] - 15s 244us/step - loss: 0.1369 - acc: 0.9696 - va
Epoch 19/20
60000/60000 [==============================] - 15s 248us/step - loss: 0.1352 - acc: 0.9696 - va
Epoch 20/20
60000/60000 [==============================] - 15s 249us/step - loss: 0.1258 - acc: 0.9717 - va
```

In [0]:

```python
# Evaluating the model
score = model_5d.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

# Test and train accuracy of the model
model_5d_test = score[1]
model_5d_train = history_5d.history['acc']

# Plotting Train and Test Loss VS no. of epochs
# list of epoch numbers
x = list(range(1,nb_epoch+1))

# Validation loss
vy = history_5d.history['val_loss']
# Training loss
ty = history_5d.history['loss']

# Calling the function to draw the plot
plt_dynamic(x, vy, ty)
```

```
Test score: 0.07800791863687337
Test accuracy: 0.9824
```



1.4 Table (Different models with their train and test accuracies):

In [4]:
```
# Installing the library prettytable
!pip install prettytable

from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["S.No.", "Model", "Training Accuracy", "Test Accuracy"]

x.add_row([1.,"MLP(2 Hidden Layer) Without Dropout and Normalization", 0.99,0.97])
x.add_row([2.,"MLP(2 Hidden Layer) With Dropout and Normalization", 0.97,0.98])

x.add_row([3.,"MLP(3 Hidden Layer) Without Dropout and Normalization", 0.99,0.98])
x.add_row([4.,"MLP(3 Hidden Layer) With Dropout and Normalization", 0.97,0.98])

x.add_row([5.,"MLP(5 Hidden Layer) Without Dropout and Normalization", 0.99,0.97])
x.add_row([6.,"MLP(5 Hidden Layer) With Dropout and Normalization", 0.97,0.98])

print(x)
```

```
Requirement already satisfied: prettytable in /usr/local/lib/python3.6/dist-packages (0.7.2)
+-------+-------------------------------------------------------+-------------------+----------
| S.No. |                         Model                         | Training Accuracy | Test Accu
+-------+-------------------------------------------------------+-------------------+----------
```

1.0	MLP(2 Hidden Layer) Without Dropout and Normalization	0.99	0.97
2.0	MLP(2 Hidden Layer) With Dropout and Normalization	0.97	0.98
3.0	MLP(3 Hidden Layer) Without Dropout and Normalization	0.99	0.98
4.0	MLP(3 Hidden Layer) With Dropout and Normalization	0.97	0.98
5.0	MLP(5 Hidden Layer) Without Dropout and Normalization	0.99	0.97
6.0	MLP(5 Hidden Layer) With Dropout and Normalization	0.97	0.98

1.5 Conclusion:-

1.5.1 Procedure followed:-

1. Load MNIST dataset
2. Split the dataset into train and test
3. Normalize the train and test data
4. Convert class variable into categorical data vector
5. Implement Softmax classifier with 2 , 3 and 5 hidden layers without Dropout and Batch Normalization .
6. Then Implemented with Dropout and Batch Normalization to the hidden layers .
7. Draw Categorical Crossentropy Loss VS No.of Epochs plot