Assignment - TensorFlow and Keras Build various MLP architectures for MNIST dataset

In this assignment we have to try three different MLP architecture on MNIST data set. We have input as 784 and output as 10.

The task is to try 3 different MLP architecture, with Batch Normalization, Dropout as Regularizer, ReLU as Activation function and Adam as Optimizer.

We will try following architectures

Architecture 1 : MLP+ Adam + BN + ReLu + Dropout (0.5) + 2 Hidden Layer

Architecture 2 : MLP+ Adam + BN + ReLu + Dropout (0.3) + 3 Hidden Layer

Architecture 3 : MLP+ Adam + BN + ReLu + Dropout (0.2) + 5 Hidden Layer

Output: The expected output for this assignment is the plot between Train/test loss vs Epoch and Accuracy for all the models we train .

In [1]:
```
# if you keras is not using tensorflow as backend set "KERAS_BACKEND=tensorflow" use this command
import warnings
warnings.filterwarnings('ignore')
from keras.utils import np_utils
from keras.datasets import mnist
import seaborn as sns
from keras.initializers import RandomNormal
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout
```

```
Using TensorFlow backend.
```

In [2]:
```
%matplotlib notebook
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()
```

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, %d)"%(X_train.shape[1], X_train.shape[2]
print("Number of training examples :", X_test.shape[0], "and each image is of shape (%d, %d)"%(X_test.shape[1], X_test.shape[2]))
```

```
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 2 dimensional vector
# for each image we have a (28*28) vector
# we will convert the (28*28) vector into single dimensional vector of 1 * 784

X_train = X_train.reshape(X_train.shape[0], X_train.shape[1]*X_train.shape[2])
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1]*X_test.shape[2])
```

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)"%(X_train.shape[1]))
print("Number of training examples :", X_test.shape[0], "and each image is of shape (%d)"%(X_test.shape[1]))
```

```
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.]
```

Softmax classifier

In [9]:

```
# https://keras.io/getting-started/sequential-model-guide/

# The Sequential model is a linear stack of layers.
# you can create a Sequential model by passing a list of layer instances to the constructor:

# model = Sequential([
#     Dense(32, input_shape=(784,)),
#     Activation('relu'),
#     Dense(10),
#     Activation('softmax'),
# ])

# You can also simply add layers via the .add() method:

# model = Sequential()
# model.add(Dense(32, input_dim=784))
# model.add(Activation('relu'))

###

# https://keras.io/layers/core/

# keras.layers.Dense(units, activation=None, use_bias=True, kernel_initializer='glorot_uniform',
# bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None,
# kernel_constraint=None, bias_constraint=None)

# Dense implements the operation: output = activation(dot(input, kernel) + bias) where
# activation is the element-wise activation function passed as the activation argument,
# kernel is a weights matrix created by the layer, and
# bias is a bias vector created by the layer (only applicable if use_bias is True).

# output = activation(dot(input, kernel) + bias)  => y = activation(WT. X + b)

####
```

```
# https://keras.io/activations/

# Activations can either be used through an Activation layer, or through the activation argument supported by all forward layers:

# from keras.layers import Activation, Dense

# model.add(Dense(64))
# model.add(Activation('tanh'))

# This is equivalent to:
# model.add(Dense(64, activation='tanh'))

# there are many activation functions ar available ex: tanh, relu, softmax


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

In [10]:

```
output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 20
```

Architecture 1 : MLP + Adam + BN + ReLu + Dropout (0.5) with 2 Hidden Layer (512 - 256)

In [11]:

```
MLP_model_2Layers = Sequential()
MLP_model_2Layers.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.0
MLP_model_2Layers.add(BatchNormalization())
MLP_model_2Layers.add(Dropout(0.5))

MLP_model_2Layers.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)))
MLP_model_2Layers.add(BatchNormalization())
MLP_model_2Layers.add(Dropout(0.5))



MLP_model_2Layers.add(Dense(output_dim, activation='softmax'))
MLP_model_2Layers.summary()
MLP_model_2Layers.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history_2layers_b_d = MLP_model_2Layers.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(
```

```
WARNING:tensorflow:From C:\Anaconda\lib\site-packages\tensorflow\python\ops\resource_variable_ops.py:435: colocate_with (from tens
orflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 512)               401920
_________________________________________________________________
batch_normalization_1 (Batch (None, 512)               2048
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
batch_normalization_2 (Batch (None, 256)               1024
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                2570
=================================================================
Total params: 538,890
Trainable params: 537,354
Non-trainable params: 1,536
_________________________________________________________________
WARNING:tensorflow:From C:\Anaconda\lib\site-packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from tensorflow.python.op
s.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.4514 - accuracy: 0.8629 - val_loss: 0.1454 - val_accuracy:
0.9548
Epoch 2/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.2115 - accuracy: 0.9352 - val_loss: 0.1080 - val_accuracy:
0.9652
Epoch 3/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.1638 - accuracy: 0.9498 - val_loss: 0.0898 - val_accuracy:
0.9713
Epoch 4/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.1388 - accuracy: 0.9567 - val_loss: 0.0823 - val_accuracy:
0.9740
Epoch 5/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.1222 - accuracy: 0.9626 - val_loss: 0.0767 - val_accuracy:
0.9758
Epoch 6/20
```

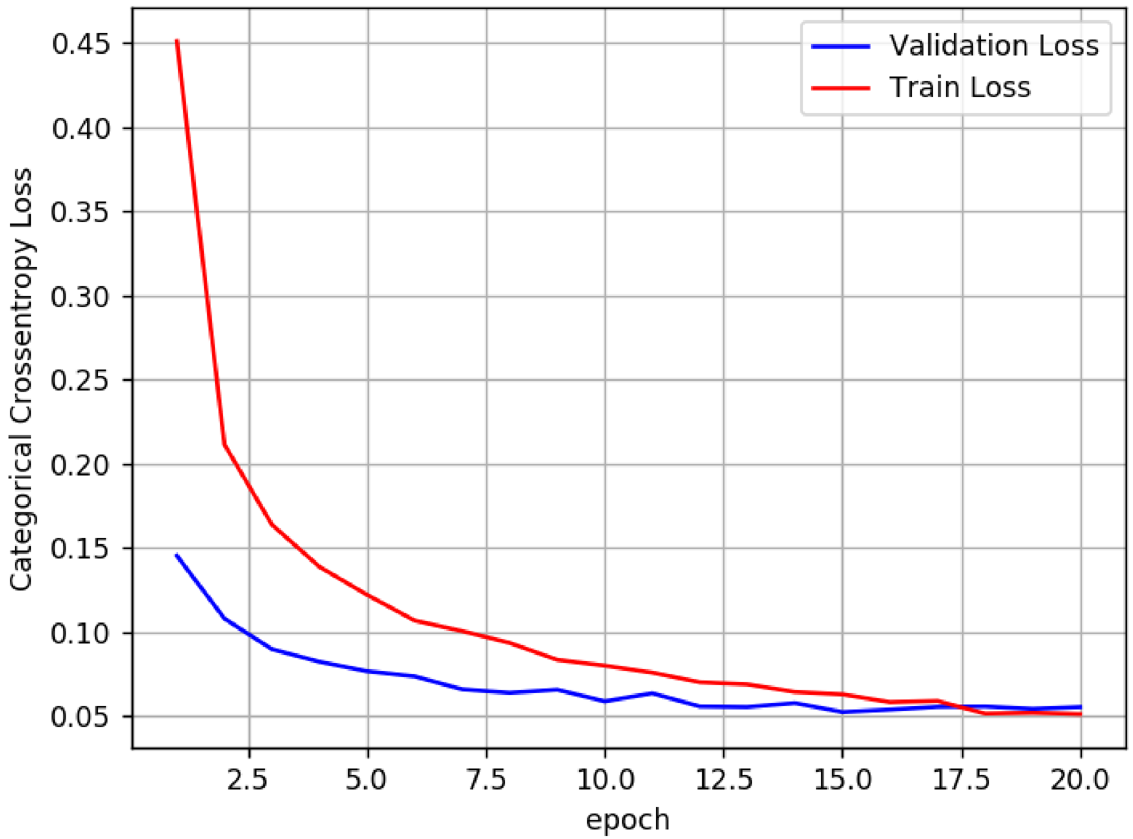
```
60000/60000 [==============================] - 7s 121us/step - loss: 0.1068 - accuracy: 0.9665 - val_loss: 0.0737 - val_accuracy:
0.9767
Epoch 7/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.1006 - accuracy: 0.9680 - val_loss: 0.0659 - val_accuracy:
0.9784
Epoch 8/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0936 - accuracy: 0.9702 - val_loss: 0.0639 - val_accuracy:
0.9805
Epoch 9/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.0835 - accuracy: 0.9736 - val_loss: 0.0657 - val_accuracy:
0.9799
Epoch 10/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.0800 - accuracy: 0.9736 - val_loss: 0.0588 - val_accuracy:
0.9812
Epoch 11/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0759 - accuracy: 0.9757 - val_loss: 0.0636 - val_accuracy:
0.9821
Epoch 12/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0702 - accuracy: 0.9775 - val_loss: 0.0558 - val_accuracy:
0.9825
Epoch 13/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.0689 - accuracy: 0.9784 - val_loss: 0.0555 - val_accuracy:
0.9826
Epoch 14/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0644 - accuracy: 0.9795 - val_loss: 0.0578 - val_accuracy:
0.9829
Epoch 15/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0629 - accuracy: 0.9792 - val_loss: 0.0524 - val_accuracy:
0.9844
Epoch 16/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0583 - accuracy: 0.9812 - val_loss: 0.0539 - val_accuracy:
0.9828
Epoch 17/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.0590 - accuracy: 0.9807 - val_loss: 0.0556 - val_accuracy:
0.9824
Epoch 18/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0515 - accuracy: 0.9830 - val_loss: 0.0558 - val_accuracy:
0.9833cy:
Epoch 19/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0520 - accuracy: 0.9831 - val_loss: 0.0543 - val_accuracy:
0.9838
Epoch 20/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0512 - accuracy: 0.9833 - val_loss: 0.0554 - val_accuracy:
0.9822
```

In [12]:

```
score = MLP_model_2Layers.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
```

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

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

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

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

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

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

vy = history_2layers_b_d.history['val_loss']
ty = history_2layers_b_d.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.0554369732551655
Test accuracy: 0.982200026512146
```



Architecture 2 : MLP+ Adam + BN + ReLu + Dropout (0.3) with 3 Hidden Layer (512- 256 -128)

In [13]:
```
MLP_model_3Layers = Sequential()
MLP_model_3Layers.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.0
MLP_model_3Layers.add(BatchNormalization())
MLP_model_3Layers.add(Dropout(0.3))

MLP_model_3Layers.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.051, seed=None)))
MLP_model_3Layers.add(BatchNormalization())
MLP_model_3Layers.add(Dropout(0.3))
```

```
MLP_model_3Layers.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.072, seed=None)))
MLP_model_3Layers.add(BatchNormalization())
MLP_model_3Layers.add(Dropout(0.3))

MLP_model_3Layers.add(Dense(output_dim, activation='softmax'))
MLP_model_3Layers.summary()
MLP_model_3Layers.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history_b_d = MLP_model_3Layers.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_test,
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 512)               401920
_________________________________________________________________
batch_normalization_3 (Batch (None, 512)               2048
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_5 (Dense)              (None, 256)               131328
_________________________________________________________________
batch_normalization_4 (Batch (None, 256)               1024
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_6 (Dense)              (None, 128)               32896
_________________________________________________________________
batch_normalization_5 (Batch (None, 128)               512
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_7 (Dense)              (None, 10)                1290
=================================================================
Total params: 571,018
Trainable params: 569,226
Non-trainable params: 1,792
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 9s 145us/step - loss: 0.3151 - accuracy: 0.9037 - val_loss: 0.1213 - val_accuracy:
0.9620
Epoch 2/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.1488 - accuracy: 0.9545 - val_loss: 0.0886 - val_accuracy:
0.9715
Epoch 3/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.1134 - accuracy: 0.9653 - val_loss: 0.0776 - val_accuracy:
```

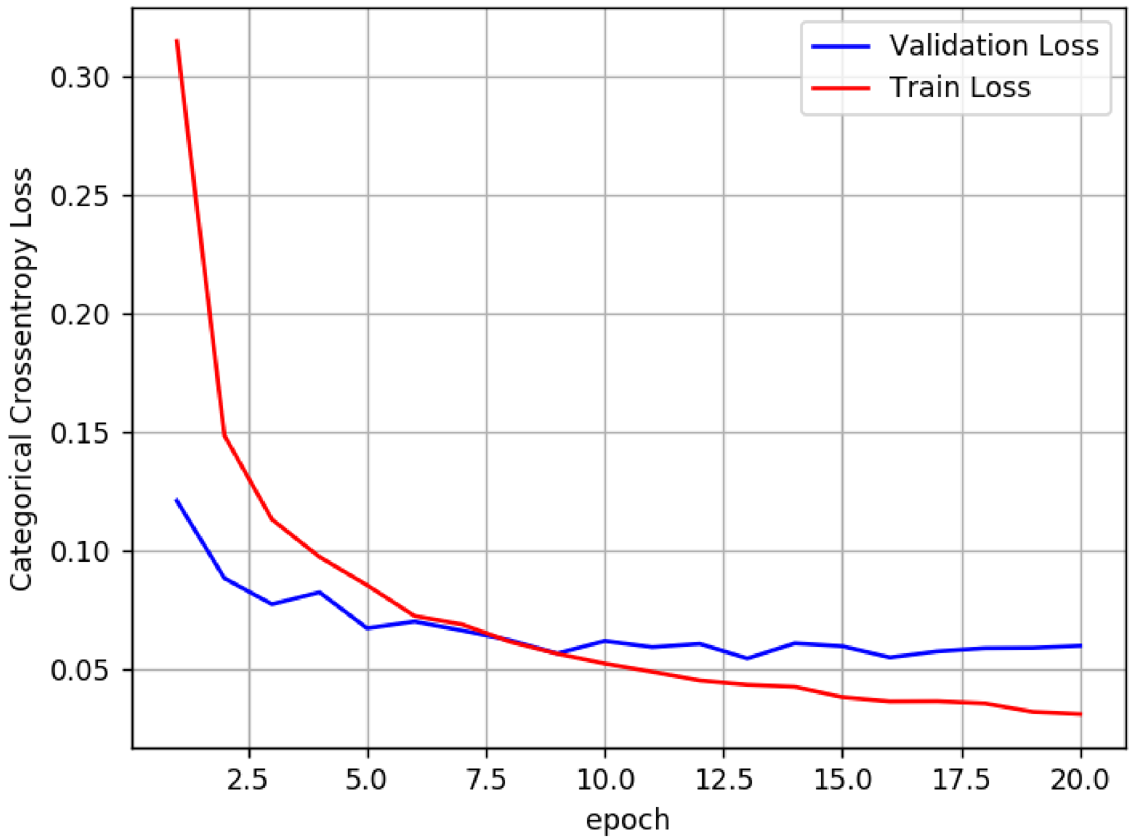
```
0.9762
Epoch 4/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0976 - accuracy: 0.9704 - val_loss: 0.0827 - val_accuracy:
0.9742
Epoch 5/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.0857 - accuracy: 0.9736 - val_loss: 0.0675 - val_accuracy:
0.9783
Epoch 6/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0727 - accuracy: 0.9768 - val_loss: 0.0704 - val_accuracy:
0.9808
Epoch 7/20
60000/60000 [==============================] - 9s 144us/step - loss: 0.0691 - accuracy: 0.9782 - val_loss: 0.0666 - val_accuracy:
0.9789
Epoch 8/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.0619 - accuracy: 0.9799 - val_loss: 0.0625 - val_accuracy:
0.9804
Epoch 9/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.0568 - accuracy: 0.9820 - val_loss: 0.0569 - val_accuracy:
0.9840
Epoch 10/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.0526 - accuracy: 0.9831 - val_loss: 0.0622 - val_accuracy:
0.9811
Epoch 11/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.0491 - accuracy: 0.9842 - val_loss: 0.0596 - val_accuracy:
0.9824
Epoch 12/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.0455 - accuracy: 0.9858 - val_loss: 0.0610 - val_accuracy:
0.9825
Epoch 13/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.0437 - accuracy: 0.9861 - val_loss: 0.0548 - val_accuracy:
0.9839
Epoch 14/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.0428 - accuracy: 0.9859 - val_loss: 0.0612 - val_accuracy:
0.9818
Epoch 15/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.0384 - accuracy: 0.9877 - val_loss: 0.0600 - val_accuracy:
0.9830
Epoch 16/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.0367 - accuracy: 0.9882 - val_loss: 0.0552 - val_accuracy:
0.9835
Epoch 17/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.0367 - accuracy: 0.9882 - val_loss: 0.0578 - val_accuracy:
0.9830
Epoch 18/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.0359 - accuracy: 0.9885 - val_loss: 0.0591 - val_accuracy:
0.9843
Epoch 19/20
```

```
60000/60000 [==============================] - 8s 131us/step - loss: 0.0323 - accuracy: 0.9897 - val_loss: 0.0592 - val_accuracy: 0.9836
Epoch 20/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.0313 - accuracy: 0.9899 - val_loss: 0.0601 - val_accuracy: 0.9835
```

In [14]:

```
score = MLP_model_3Layers.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

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

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

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

```
Test score: 0.06012513352438982
Test accuracy: 0.9835000038146973
```



Architecture 3 : MLP+ Adam + BN + ReLu + Dropout (0.2) with 5 Hidden Layer (512- 256 -128 – 64 - 32)

In [15]:
```
MLP_model_5Layers = Sequential()
MLP_model_5Layers.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.0
MLP_model_5Layers.add(BatchNormalization())
MLP_model_5Layers.add(Dropout(0.2))

MLP_model_5Layers.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.051, seed=None)))
MLP_model_5Layers.add(BatchNormalization())
MLP_model_5Layers.add(Dropout(0.2))
```

```
MLP_model_5Layers.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.072, seed=None)))
MLP_model_5Layers.add(BatchNormalization())
MLP_model_5Layers.add(Dropout(0.2))

MLP_model_5Layers.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.102, seed=None)))
MLP_model_5Layers.add(BatchNormalization())
MLP_model_5Layers.add(Dropout(0.2))

MLP_model_5Layers.add(Dense(32, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.144, seed=None)))
MLP_model_5Layers.add(BatchNormalization())
MLP_model_5Layers.add(Dropout(0.2))

MLP_model_5Layers.add(Dense(output_dim, activation='softmax'))
MLP_model_5Layers.summary()
MLP_model_5Layers.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history_5layers_b_d = MLP_model_5Layers.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_8 (Dense)              (None, 512)               401920
_________________________________________________________________
batch_normalization_6 (Batch (None, 512)               2048
_________________________________________________________________
dropout_6 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_9 (Dense)              (None, 256)               131328
_________________________________________________________________
batch_normalization_7 (Batch (None, 256)               1024
_________________________________________________________________
dropout_7 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_10 (Dense)             (None, 128)               32896
_________________________________________________________________
batch_normalization_8 (Batch (None, 128)               512
_________________________________________________________________
dropout_8 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_11 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_9 (Batch (None, 64)                256
_________________________________________________________________
dropout_9 (Dropout)          (None, 64)                0
_________________________________________________________________
```

```
dense_12 (Dense)             (None, 32)                2080
_________________________________________________________________
batch_normalization_10 (Batc (None, 32)                128
_________________________________________________________________
dropout_10 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_13 (Dense)             (None, 10)                330
=================================================================
Total params: 580,778
Trainable params: 578,794
Non-trainable params: 1,984
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 10s 166us/step - loss: 0.4479 - accuracy: 0.8729 - val_loss: 0.1385 - val_accuracy: 0.9582
Epoch 2/20
60000/60000 [==============================] - 9s 150us/step - loss: 0.1802 - accuracy: 0.9499 - val_loss: 0.1045 - val_accuracy: 0.9696
Epoch 3/20
60000/60000 [==============================] - 9s 151us/step - loss: 0.1341 - accuracy: 0.9627 - val_loss: 0.0975 - val_accuracy: 0.9713
Epoch 4/20
60000/60000 [==============================] - 9s 151us/step - loss: 0.1159 - accuracy: 0.9672 - val_loss: 0.0801 - val_accuracy: 0.9766
Epoch 5/20
60000/60000 [==============================] - 9s 150us/step - loss: 0.0986 - accuracy: 0.9722 - val_loss: 0.0839 - val_accuracy: 0.9768
Epoch 6/20
60000/60000 [==============================] - 9s 152us/step - loss: 0.0881 - accuracy: 0.9752 - val_loss: 0.0788 - val_accuracy: 0.9772
Epoch 7/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.0832 - accuracy: 0.9764 - val_loss: 0.0723 - val_accuracy: 0.9787
Epoch 8/20
60000/60000 [==============================] - 9s 149us/step - loss: 0.0757 - accuracy: 0.9788 - val_loss: 0.0857 - val_accuracy: 0.9770
Epoch 9/20
60000/60000 [==============================] - 9s 149us/step - loss: 0.0678 - accuracy: 0.9803 - val_loss: 0.0665 - val_accuracy: 0.9812
Epoch 10/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0599 - accuracy: 0.9830 - val_loss: 0.0753 - val_accuracy: 0.9785
Epoch 11/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.0594 - accuracy: 0.9824 - val_loss: 0.0671 - val_accuracy: 0.9821
Epoch 12/20
```

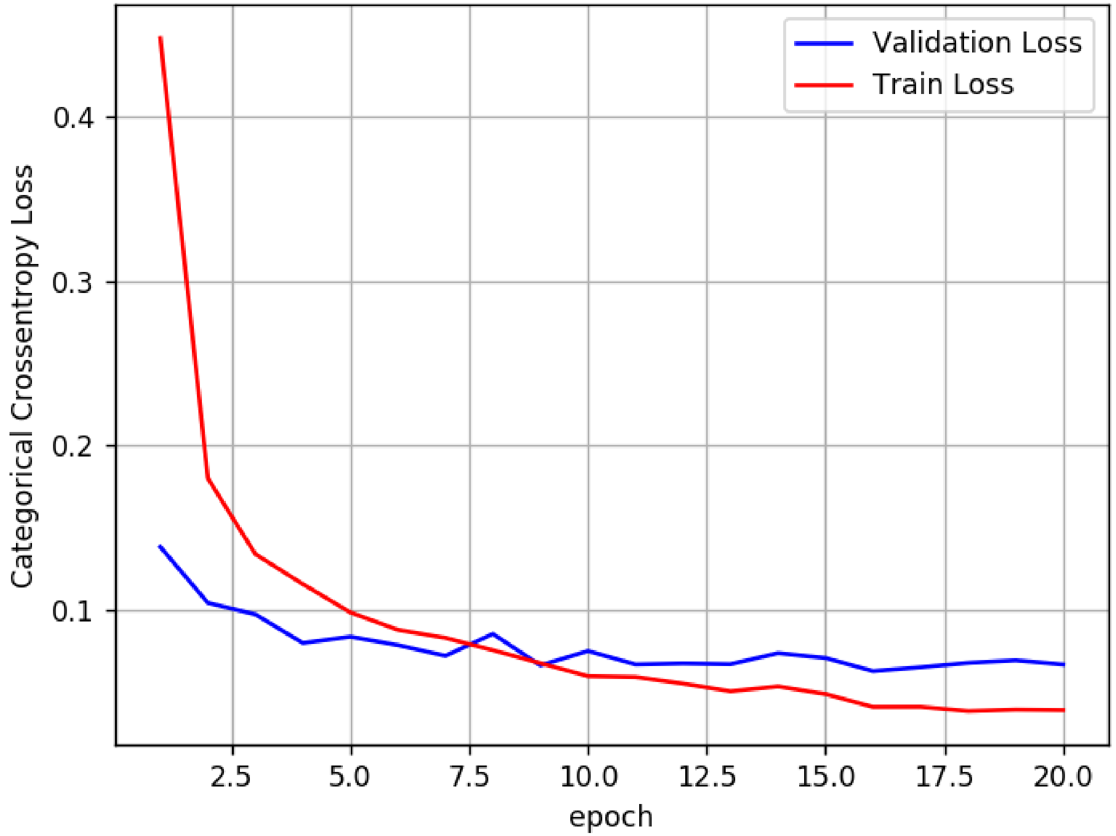
```
60000/60000 [==============================] - 9s 147us/step - loss: 0.0554 - accuracy: 0.9836 - val_loss: 0.0676 - val_accuracy:
0.9814
Epoch 13/20
60000/60000 [==============================] - 9s 150us/step - loss: 0.0507 - accuracy: 0.9855 - val_loss: 0.0672 - val_accuracy:
0.9816
Epoch 14/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.0537 - accuracy: 0.9847 - val_loss: 0.0739 - val_accuracy:
0.9809
Epoch 15/20
60000/60000 [==============================] - 9s 153us/step - loss: 0.0490 - accuracy: 0.9858 - val_loss: 0.0710 - val_accuracy:
0.9800
Epoch 16/20
60000/60000 [==============================] - 9s 151us/step - loss: 0.0412 - accuracy: 0.9880 - val_loss: 0.0630 - val_accuracy:
0.9831
Epoch 17/20
60000/60000 [==============================] - 9s 151us/step - loss: 0.0413 - accuracy: 0.9876 - val_loss: 0.0653 - val_accuracy:
0.9834
Epoch 18/20
60000/60000 [==============================] - 9s 151us/step - loss: 0.0387 - accuracy: 0.9892 - val_loss: 0.0680 - val_accuracy:
0.9827
Epoch 19/20
60000/60000 [==============================] - 9s 154us/step - loss: 0.0396 - accuracy: 0.9887 - val_loss: 0.0695 - val_accuracy:
0.9824
Epoch 20/20
60000/60000 [==============================] - 9s 151us/step - loss: 0.0393 - accuracy: 0.9889 - val_loss: 0.0670 - val_accuracy:
0.9829
```

In [16]:

```
score = MLP_model_5Layers.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

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

# loss : training loss
```

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

vy = history_5layers_b_d.history['val_loss']
ty = history_5layers_b_d.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.06700490548466333
Test accuracy: 0.9829000234603882
```



Conclusion

In [17]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Sr. No", "Model", "Hidden Layer Info", "Test Accuracy (%)"]
x.add_row(["Architecture 1 ",  "MLP+ Adam + BN + ReLu + Dropout (0.5) + 2  Layer   ", "512 - 256", 98.22])
x.add_row(["Architecture 2 ",  " MLP+ Adam + BN + ReLu + Dropout (0.3) + 3  Layer     ","512 - 256 -128" , 98.35])
x.add_row(["Architecture 3 ", "MLP+ Adam + BN + ReLu + Dropout (0.2) + 5  Layer   ","512- 256 -128 - 64 - 32", 98.29])
print(x)
```

```
+------------------+---------------------------------------------------------+--------------------------+--------------------+
|      Sr. No      |                          Model                          |    Hidden Layer Info     | Test Accuracy (%) |
+------------------+---------------------------------------------------------+--------------------------+--------------------+
| Architecture 1   |  MLP+ Adam + BN + ReLu + Dropout (0.5) + 2  Layer       |        512 - 256         |       98.22        |
| Architecture 2   |  MLP+ Adam + BN + ReLu + Dropout (0.3) + 3  Layer       |      512 - 256 -128      |       98.35        |
| Architecture 3   |  MLP+ Adam + BN + ReLu + Dropout (0.2) + 5  Layer       | 512- 256 -128 - 64 - 32 |       98.29        |
+------------------+---------------------------------------------------------+--------------------------+--------------------+
```