Keras -- MLPs on MNIST

In [0]:

```
# if you keras is not using tensorflow as backend set "KERAS_
BACKEND=tensorflow" use this command
from keras.utils import np_utils
from keras.datasets import mnist
import seaborn as sns
from keras.initializers import RandomNormal
```

```
Using TensorFlow backend.
```

In [0]:

```
%matplotlib notebook
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import time
# https://gist.github.com/greydanus/f6eee59eaf1d90fcb3b534a25
362cea4
# https://stackoverflow.com/a/14434334
# this function is used to update the plots for each epoch an
d 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 [0]:

```
# the data, shuffled and split between train and test sets
```

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
Downloading data from https://s3.amazonaws.com
/img-datasets/mnist.npz
11493376/11490434 [==========================
===] - 2s 0us/step
```

In [0]:

```
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.sha
pe[2]))
```

```
Number of training examples : 60000 and each i
mage is of shape (28, 28)
Number of training examples : 10000 and each i
mage is of shape (28, 28)
```

In [0]:

```
# 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_te
st.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 (%d)"%(X_train.shape[1]))
```

```python
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 i
mage is of shape (784)
Number of training examples : 10000 and each i
mage is of shape (784)
```

In [0]:

```python
# 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 2
53 253 253 253 251  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 2
53 253 205  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 2
53 253 119  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 1
82   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 1
 71 219 253 253 253 253 195
   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 2
 53 253 253 212 135 132  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]
```

Normalize the data

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 tr
y 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.]
```

Softmax classifier

In [0]:

```
# 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 laye
r 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, k
ernel_initializer='glorot_uniform',
# bias_initializer='zeros', kernel_regularizer=None, bias_reg
ularizer=None, activity_regularizer=None,
# kernel_constraint=None, bias_constraint=None)

# Dense implements the operation: output = activation(dot(inp
```

```
ut, kernel) + bias) where
# activation is the element-wise activation function passed a
s 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 = acti
vation(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 [0]:

```
# some model parameters
```

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

batch_size = 128
nb_epoch = 20
```

Model 1 : MLP + ReLU + ADAM

In [0]:

```
model_relu = Sequential()
model_relu.add(Dense(324, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_relu.add(Dense(108, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

history = model_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
WARNING: Logging before flag parsing goes to stderr.
W0728 09:41:24.063638 139911311832960 deprecation_wrapper.py:119] From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:74: The name tf.get_default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.

W0728 09:41:24.105846 139911311832960 deprecation_wrapper.py:119] From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:517: The name tf.placeholder is deprec
```

```
ated. Please use tf.compat.v1.placeholder inst
ead.

W0728 09:41:24.116193 139911311832960 deprecat
ion_wrapper.py:119] From /usr/local/lib/python
3.6/dist-packages/keras/backend/tensorflow_bac
kend.py:4115: The name tf.random_normal is dep
recated. Please use tf.random.normal instead.

W0728 09:41:24.149391 139911311832960 deprecat
ion_wrapper.py:119] From /usr/local/lib/python
3.6/dist-packages/keras/backend/tensorflow_bac
kend.py:4138: The name tf.random_uniform is de
precated. Please use tf.random.uniform instead
.

W0728 09:41:24.171610 139911311832960 deprecat
ion_wrapper.py:119] From /usr/local/lib/python
3.6/dist-packages/keras/optimizers.py:790: The
 name tf.train.Optimizer is deprecated. Please
 use tf.compat.v1.train.Optimizer instead.

W0728 09:41:24.203595 139911311832960 deprecat
ion_wrapper.py:119] From /usr/local/lib/python
3.6/dist-packages/keras/backend/tensorflow_bac
kend.py:3295: The name tf.log is deprecated. P
lease use tf.math.log instead.

W0728 09:41:24.345604 139911311832960 deprecat
ion.py:323] From /usr/local/lib/python3.6/dist
-packages/tensorflow/python/ops/math_grad.py:1
250: add_dispatch_support.<locals>.wrapper (fr
om tensorflow.python.ops.array_ops) is depreca
ted and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadc
ast rule as np.where
```

```
______________________________________________
___________________
Layer (type)                 Output Shape
         Param #
==============================================
===================
dense_1 (Dense)              (None, 324)
         254340
______________________________________________
___________________
dense_2 (Dense)              (None, 108)
         35100
______________________________________________
___________________
dense_3 (Dense)              (None, 10)
         1090
==============================================
===================
Total params: 290,530
Trainable params: 290,530
Non-trainable params: 0
______________________________________________
___________________
None
Train on 60000 samples, validate on 10000 samp
les
Epoch 1/20
60000/60000 [==============================] -
 6s 108us/step - loss: 0.2468 - acc: 0.9263 -
val_loss: 0.1157 - val_acc: 0.9659
Epoch 2/20
60000/60000 [==============================] -
 3s 46us/step - loss: 0.0939 - acc: 0.9720 - v
al_loss: 0.0882 - val_acc: 0.9732
Epoch 3/20
60000/60000 [==============================] -
 3s 46us/step - loss: 0.0610 - acc: 0.9809 - v
```

```
al_loss: 0.0834 - val_acc: 0.9738
Epoch 4/20
60000/60000 [==============================] -
 3s 45us/step - loss: 0.0428 - acc: 0.9869 - v
al_loss: 0.0710 - val_acc: 0.9790
Epoch 5/20
60000/60000 [==============================] -
 3s 46us/step - loss: 0.0298 - acc: 0.9912 - v
al_loss: 0.0791 - val_acc: 0.9756
Epoch 6/20
60000/60000 [==============================] -
 3s 46us/step - loss: 0.0242 - acc: 0.9925 - v
al_loss: 0.0730 - val_acc: 0.9788
Epoch 7/20
60000/60000 [==============================] -
 3s 46us/step - loss: 0.0185 - acc: 0.9942 - v
al_loss: 0.0702 - val_acc: 0.9792
Epoch 8/20
60000/60000 [==============================] -
 3s 45us/step - loss: 0.0152 - acc: 0.9953 - v
al_loss: 0.0994 - val_acc: 0.9744
Epoch 9/20
60000/60000 [==============================] -
 3s 45us/step - loss: 0.0129 - acc: 0.9960 - v
al_loss: 0.0846 - val_acc: 0.9778
Epoch 10/20
60000/60000 [==============================] -
 3s 45us/step - loss: 0.0115 - acc: 0.9962 - v
al_loss: 0.0825 - val_acc: 0.9791
Epoch 11/20
60000/60000 [==============================] -
 3s 45us/step - loss: 0.0110 - acc: 0.9962 - v
al_loss: 0.0720 - val_acc: 0.9799
Epoch 12/20
60000/60000 [==============================] -
 3s 45us/step - loss: 0.0112 - acc: 0.9962 - v
al_loss: 0.0788 - val_acc: 0.9791
```

```
Epoch 13/20
60000/60000 [==============================] -
 3s 46us/step - loss: 0.0116 - acc: 0.9960 - v
al_loss: 0.0926 - val_acc: 0.9787
Epoch 14/20
60000/60000 [==============================] -
 3s 46us/step - loss: 0.0084 - acc: 0.9973 - v
al_loss: 0.0793 - val_acc: 0.9822
Epoch 15/20
60000/60000 [==============================] -
 3s 45us/step - loss: 0.0103 - acc: 0.9966 - v
al_loss: 0.1006 - val_acc: 0.9769
Epoch 16/20
60000/60000 [==============================] -
 3s 45us/step - loss: 0.0072 - acc: 0.9976 - v
al_loss: 0.0944 - val_acc: 0.9797
Epoch 17/20
60000/60000 [==============================] -
 3s 45us/step - loss: 0.0028 - acc: 0.9990 - v
al_loss: 0.0888 - val_acc: 0.9804
Epoch 18/20
60000/60000 [==============================] -
 3s 46us/step - loss: 0.0114 - acc: 0.9962 - v
al_loss: 0.1006 - val_acc: 0.9778
Epoch 19/20
60000/60000 [==============================] -
 3s 46us/step - loss: 0.0087 - acc: 0.9970 - v
al_loss: 0.0947 - val_acc: 0.9809
Epoch 20/20
60000/60000 [==============================] -
 3s 46us/step - loss: 0.0071 - acc: 0.9975 - v
al_loss: 0.1004 - val_acc: 0.9794
```

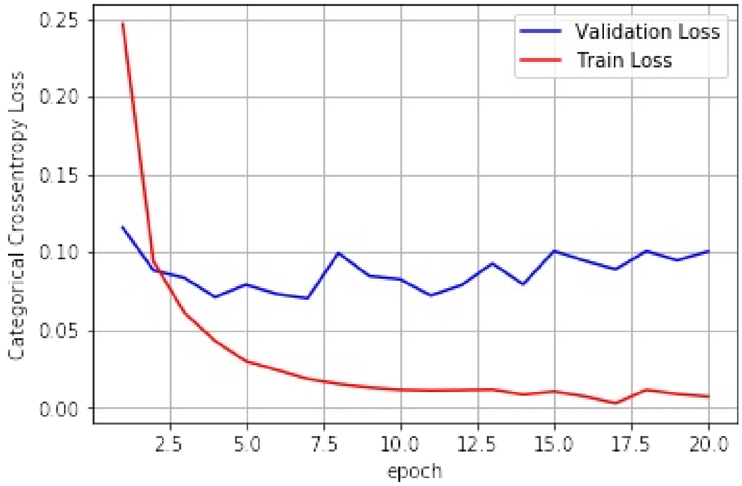
Train Accuracy = 99.75%

In [0]:

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

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

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

model_test_score = score[0]
model_test_acc = score[1]
model_train = history.history['acc']

# 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 par
amter 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 le
ngth equal to number of epochs


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

```
Test score: 0.10035708108908771
```

Test accuracy: 0.9794



MLP + Batch-Norm on hidden Layers + AdamOptimizer </2>

In [0]:

```
from keras.layers.normalization import BatchNormalization

model_batch = Sequential()

model_batch.add(Dense(324, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_batch.add(BatchNormalization())

model_batch.add(Dense(108, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.55, seed=None)) )
model_batch.add(BatchNormalization())

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

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

history = model_batch.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.2165 - acc: 0.9363 - val_loss: 0.1146 - val_acc: 0.9670
```

```
Epoch 2/20
60000/60000 [==============================] -
 4s 74us/step - loss: 0.0809 - acc: 0.9754 - v
al_loss: 0.0907 - val_acc: 0.9728
Epoch 3/20
60000/60000 [==============================] -
 4s 74us/step - loss: 0.0523 - acc: 0.9845 - v
al_loss: 0.0789 - val_acc: 0.9754
Epoch 4/20
60000/60000 [==============================] -
 4s 74us/step - loss: 0.0367 - acc: 0.9889 - v
al_loss: 0.0728 - val_acc: 0.9779
Epoch 5/20
60000/60000 [==============================] -
 4s 75us/step - loss: 0.0301 - acc: 0.9908 - v
al_loss: 0.0721 - val_acc: 0.9773
Epoch 6/20
60000/60000 [==============================] -
 4s 75us/step - loss: 0.0237 - acc: 0.9928 - v
al_loss: 0.0747 - val_acc: 0.9777
Epoch 7/20
60000/60000 [==============================] -
 5s 76us/step - loss: 0.0216 - acc: 0.9929 - v
al_loss: 0.0773 - val_acc: 0.9770
Epoch 8/20
60000/60000 [==============================] -
 5s 76us/step - loss: 0.0137 - acc: 0.9959 - v
al_loss: 0.0931 - val_acc: 0.9746
Epoch 9/20
60000/60000 [==============================] -
 4s 75us/step - loss: 0.0149 - acc: 0.9955 - v
al_loss: 0.0835 - val_acc: 0.9765
Epoch 10/20
60000/60000 [==============================] -
 4s 74us/step - loss: 0.0125 - acc: 0.9961 - v
al_loss: 0.0849 - val_acc: 0.9767
Epoch 11/20
```

```
60000/60000 [==============================] -
 4s 75us/step - loss: 0.0129 - acc: 0.9957 - v
al_loss: 0.0830 - val_acc: 0.9770
Epoch 12/20
60000/60000 [==============================] -
 4s 75us/step - loss: 0.0109 - acc: 0.9966 - v
al_loss: 0.0794 - val_acc: 0.9797
Epoch 13/20
60000/60000 [==============================] -
 4s 74us/step - loss: 0.0086 - acc: 0.9973 - v
al_loss: 0.0820 - val_acc: 0.9789
Epoch 14/20
60000/60000 [==============================] -
 4s 74us/step - loss: 0.0089 - acc: 0.9971 - v
al_loss: 0.0794 - val_acc: 0.9783
Epoch 15/20
60000/60000 [==============================] -
 4s 74us/step - loss: 0.0091 - acc: 0.9968 - v
al_loss: 0.0839 - val_acc: 0.9790
Epoch 16/20
60000/60000 [==============================] -
 4s 74us/step - loss: 0.0086 - acc: 0.9973 - v
al_loss: 0.0779 - val_acc: 0.9810
Epoch 17/20
60000/60000 [==============================] -
 4s 73us/step - loss: 0.0081 - acc: 0.9976 - v
al_loss: 0.0760 - val_acc: 0.9814
Epoch 18/20
60000/60000 [==============================] -
 4s 73us/step - loss: 0.0064 - acc: 0.9979 - v
al_loss: 0.0837 - val_acc: 0.9792
Epoch 19/20
60000/60000 [==============================] -
 4s 74us/step - loss: 0.0081 - acc: 0.9974 - v
al_loss: 0.0788 - val_acc: 0.9804
Epoch 20/20
60000/60000 [==============================] -
```

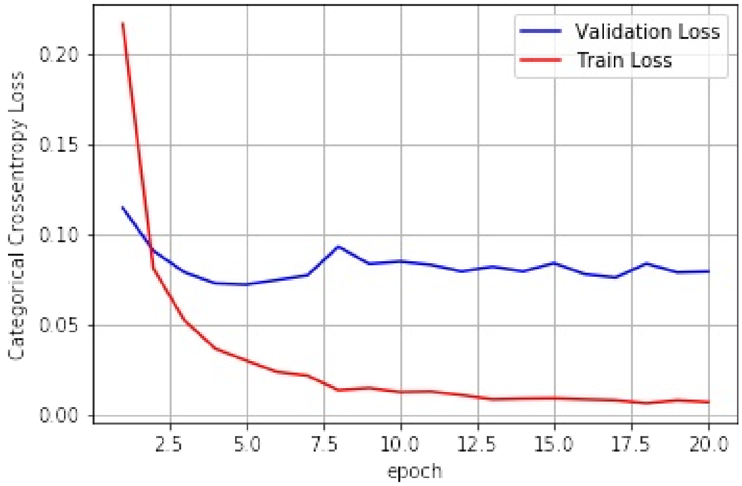
```
 4s 75us/step - loss: 0.0071 - acc: 0.9977 - v
al_loss: 0.0793 - val_acc: 0.9802
```

Train Accuracy = 99.77%

In [0]:

```
score = model_batch.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 Crossentr
opy 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 par
amter 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 le
ngth equal to number of epochs

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

```
Test score: 0.07929278038347275
Test accuracy: 0.9802
```



5. MLP + Dropout + AdamOptimizer

In [0]:

```
# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormalization-function-in-keras

from keras.layers import Dropout

model_drop = Sequential()

model_drop.add(Dense(324, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(108, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.55, seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

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



model_drop.summary()
```

```
W0728 10:01:15.132498 139911311832960 deprecation.py:506] From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:3445: calling dropout (from tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed in a future version.
Instructions for updating:
```

```
Please use `rate` instead of `keep_prob`. Rate
 should be set to `rate = 1 - keep_prob`.
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (None, 324)               254340
_________________________________________________________________
batch_normalization_3 (Batch (None, 324)               1296
_________________________________________________________________
dropout_1 (Dropout)          (None, 324)               0
_________________________________________________________________
dense_8 (Dense)              (None, 108)               35100
_________________________________________________________________
batch_normalization_4 (Batch (None, 108)               432
_________________________________________________________________
dropout_2 (Dropout)          (None, 108)               0
_________________________________________________________________
dense_9 (Dense)              (None, 10)                1090
=================================================================
```

```
Total params: 292,258
Trainable params: 291,394
Non-trainable params: 864
______________________________________________
___________________
```

In [0]:

```
model_drop.compile(optimizer='adam', loss='categorical_crosse
ntropy', metrics=['accuracy'])

history = model_drop.fit(X_train, Y_train, batch_size=batch_s
ize, epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_t
est))
```

```
Train on 60000 samples, validate on 10000 samp
les
Epoch 1/20
60000/60000 [==============================] -
 5s 90us/step - loss: 0.5430 - acc: 0.8353 - v
al_loss: 0.1843 - val_acc: 0.9431
Epoch 2/20
60000/60000 [==============================] -
 5s 78us/step - loss: 0.2851 - acc: 0.9163 - v
al_loss: 0.1363 - val_acc: 0.9570
Epoch 3/20
60000/60000 [==============================] -
 5s 78us/step - loss: 0.2328 - acc: 0.9307 - v
al_loss: 0.1152 - val_acc: 0.9632
Epoch 4/20
60000/60000 [==============================] -
 5s 78us/step - loss: 0.2003 - acc: 0.9401 - v
al_loss: 0.1024 - val_acc: 0.9694
Epoch 5/20
60000/60000 [==============================] -
 5s 78us/step - loss: 0.1782 - acc: 0.9463 - v
al_loss: 0.0930 - val_acc: 0.9711
Epoch 6/20
```

```
60000/60000 [==============================] -
 5s 79us/step - loss: 0.1635 - acc: 0.9519 - v
al_loss: 0.0893 - val_acc: 0.9735
Epoch 7/20
60000/60000 [==============================] -
 5s 78us/step - loss: 0.1514 - acc: 0.9537 - v
al_loss: 0.0866 - val_acc: 0.9745
Epoch 8/20
60000/60000 [==============================] -
 5s 79us/step - loss: 0.1431 - acc: 0.9576 - v
al_loss: 0.0796 - val_acc: 0.9765
Epoch 9/20
60000/60000 [==============================] -
 5s 79us/step - loss: 0.1302 - acc: 0.9608 - v
al_loss: 0.0728 - val_acc: 0.9777
Epoch 10/20
60000/60000 [==============================] -
 5s 79us/step - loss: 0.1252 - acc: 0.9621 - v
al_loss: 0.0777 - val_acc: 0.9761
Epoch 11/20
60000/60000 [==============================] -
 5s 78us/step - loss: 0.1177 - acc: 0.9654 - v
al_loss: 0.0776 - val_acc: 0.9775
Epoch 12/20
60000/60000 [==============================] -
 5s 78us/step - loss: 0.1114 - acc: 0.9659 - v
al_loss: 0.0704 - val_acc: 0.9788
Epoch 13/20
60000/60000 [==============================] -
 5s 78us/step - loss: 0.1067 - acc: 0.9667 - v
al_loss: 0.0730 - val_acc: 0.9774
Epoch 14/20
60000/60000 [==============================] -
 5s 77us/step - loss: 0.1051 - acc: 0.9673 - v
al_loss: 0.0735 - val_acc: 0.9775
Epoch 15/20
60000/60000 [==============================] -
```

```
 5s 78us/step - loss: 0.0994 - acc: 0.9693 - v
al_loss: 0.0686 - val_acc: 0.9800
Epoch 16/20
60000/60000 [==============================] -
 5s 78us/step - loss: 0.0928 - acc: 0.9717 - v
al_loss: 0.0639 - val_acc: 0.9805
Epoch 17/20
60000/60000 [==============================] -
 5s 78us/step - loss: 0.0937 - acc: 0.9712 - v
al_loss: 0.0722 - val_acc: 0.9779
Epoch 18/20
60000/60000 [==============================] -
 5s 78us/step - loss: 0.0915 - acc: 0.9725 - v
al_loss: 0.0681 - val_acc: 0.9808
Epoch 19/20
60000/60000 [==============================] -
 5s 78us/step - loss: 0.0827 - acc: 0.9741 - v
al_loss: 0.0636 - val_acc: 0.9808
Epoch 20/20
60000/60000 [==============================] -
 5s 78us/step - loss: 0.0864 - acc: 0.9739 - v
al_loss: 0.0689 - val_acc: 0.9796
```

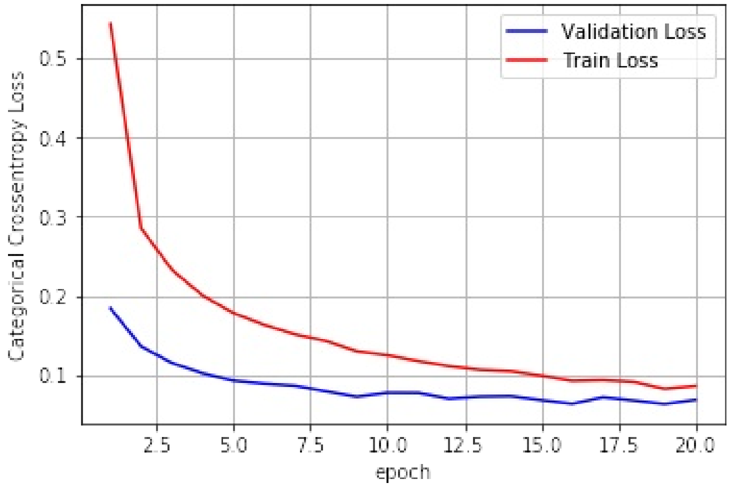
Train Accuracy = 97.39%

In [0]:

```
score = model_drop.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 Crossentr
opy 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 par
amter 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 le
ngth equal to number of epochs

vy = history.history['val_loss']
ty = history.history['loss']
```

```
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.06887181369569152
Test accuracy: 0.9796
```



Model 2 : (3 Layered) MLP + ReLU + ADAM</2>

In [0]:

```
from keras.initializers import he_normal

model_relu = Sequential()
model_relu.add(Dense(356, activation='relu', input_shape=(inp
ut_dim,), kernel_initializer=he_normal(seed=None)))
model_relu.add(Dense(105, activation='relu', kernel_initializ
er=he_normal(seed=None)) )
model_relu.add(Dense(51, activation='relu', kernel_initialize
r=he_normal(seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

model_relu.compile(optimizer='adam', loss='categorical_crosse
ntropy', metrics=['accuracy'])

history = model_relu.fit(X_train, Y_train, batch_size=batch_s
ize, epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_t
est))
```

```
______________________________________________
__________________
Layer (type)                 Output Shape
         Param #
==============================================
==================
dense_10 (Dense)             (None, 356)
         279460
```

```
_________________________________________________________________
dense_11 (Dense)             (None, 105)
         37485
_________________________________________________________________
dense_12 (Dense)             (None, 51)
         5406
_________________________________________________________________
dense_13 (Dense)             (None, 10)
         520
==============================================================
===================
Total params: 322,871
Trainable params: 322,871
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samp
les
Epoch 1/20
60000/60000 [==============================] -
 4s 58us/step - loss: 0.2576 - acc: 0.9233 - v
al_loss: 0.1140 - val_acc: 0.9645
Epoch 2/20
60000/60000 [==============================] -
 3s 50us/step - loss: 0.0955 - acc: 0.9711 - v
al_loss: 0.0838 - val_acc: 0.9735
Epoch 3/20
60000/60000 [==============================] -
 3s 50us/step - loss: 0.0624 - acc: 0.9809 - v
al_loss: 0.0809 - val_acc: 0.9764
Epoch 4/20
60000/60000 [==============================] -
 3s 50us/step - loss: 0.0447 - acc: 0.9861 - v
```

```
al_loss: 0.0752 - val_acc: 0.9768
Epoch 5/20
60000/60000 [==============================] -
 3s 50us/step - loss: 0.0322 - acc: 0.9899 - v
al_loss: 0.0765 - val_acc: 0.9768
Epoch 6/20
60000/60000 [==============================] -
 3s 50us/step - loss: 0.0268 - acc: 0.9914 - v
al_loss: 0.0789 - val_acc: 0.9777
Epoch 7/20
60000/60000 [==============================] -
 3s 50us/step - loss: 0.0198 - acc: 0.9934 - v
al_loss: 0.0770 - val_acc: 0.9798
Epoch 8/20
60000/60000 [==============================] -
 3s 51us/step - loss: 0.0193 - acc: 0.9935 - v
al_loss: 0.0658 - val_acc: 0.9817
Epoch 9/20
60000/60000 [==============================] -
 3s 50us/step - loss: 0.0163 - acc: 0.9946 - v
al_loss: 0.0895 - val_acc: 0.9768
Epoch 10/20
60000/60000 [==============================] -
 3s 51us/step - loss: 0.0133 - acc: 0.9957 - v
al_loss: 0.0781 - val_acc: 0.9804
Epoch 11/20
60000/60000 [==============================] -
 3s 50us/step - loss: 0.0156 - acc: 0.9949 - v
al_loss: 0.1048 - val_acc: 0.9751
Epoch 12/20
60000/60000 [==============================] -
 3s 50us/step - loss: 0.0130 - acc: 0.9955 - v
al_loss: 0.0855 - val_acc: 0.9815
Epoch 13/20
60000/60000 [==============================] -
 3s 50us/step - loss: 0.0122 - acc: 0.9958 - v
al_loss: 0.0836 - val_acc: 0.9814
```

```
Epoch 14/20
60000/60000 [==============================] -
 3s 50us/step - loss: 0.0109 - acc: 0.9965 - v
al_loss: 0.0883 - val_acc: 0.9810
Epoch 15/20
60000/60000 [==============================] -
 3s 50us/step - loss: 0.0119 - acc: 0.9964 - v
al_loss: 0.0764 - val_acc: 0.9834
Epoch 16/20
60000/60000 [==============================] -
 3s 50us/step - loss: 0.0084 - acc: 0.9975 - v
al_loss: 0.0988 - val_acc: 0.9787
Epoch 17/20
60000/60000 [==============================] -
 3s 50us/step - loss: 0.0139 - acc: 0.9954 - v
al_loss: 0.0918 - val_acc: 0.9795
Epoch 18/20
60000/60000 [==============================] -
 3s 50us/step - loss: 0.0064 - acc: 0.9980 - v
al_loss: 0.0766 - val_acc: 0.9841
Epoch 19/20
60000/60000 [==============================] -
 3s 50us/step - loss: 0.0035 - acc: 0.9990 - v
al_loss: 0.0956 - val_acc: 0.9806
Epoch 20/20
60000/60000 [==============================] -
 3s 50us/step - loss: 0.0105 - acc: 0.9964 - v
al_loss: 0.0864 - val_acc: 0.9830
```

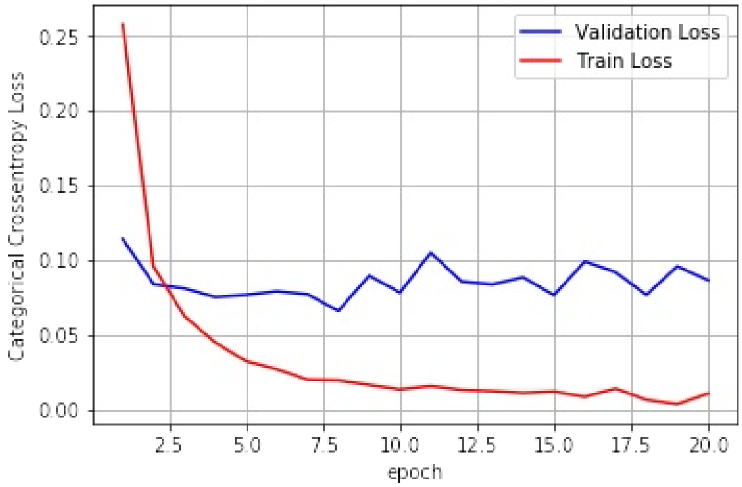
Train Accuracy=99.64%

In [0]:

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

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

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

model_test_score = score[0]
model_test_acc = score[1]
model_train = history.history['acc']

# 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 par
amter 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 le
ngth equal to number of epochs
```

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

```
Test score: 0.08639832216318823
Test accuracy: 0.983
```



MLP + Batch-Norm on hidden Layers + AdamOptimizer </2>

In [0]:

```
from keras.layers.normalization import BatchNormalization

model_batch = Sequential()

model_batch.add(Dense(356, activation='relu', input_shape=(in
put_dim,), kernel_initializer=he_normal(seed=None)))
model_batch.add(BatchNormalization())

model_batch.add(Dense(105, activation='relu', kernel_initiali
zer=he_normal(seed=None)) )
model_batch.add(BatchNormalization())

model_batch.add(Dense(51, activation='relu', kernel_initializ
er=he_normal(seed=None)) )
model_batch.add(BatchNormalization())

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

model_batch.compile(optimizer='adam', loss='categorical_cross
entropy', metrics=['accuracy'])

history = model_batch.fit(X_train, Y_train, batch_size=batch_
size, epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_
test))
```

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

```
60000/60000 [==============================] -
 7s 120us/step - loss: 0.2287 - acc: 0.9336 -
val_loss: 0.1145 - val_acc: 0.9643
Epoch 2/20
60000/60000 [==============================] -
 6s 96us/step - loss: 0.0844 - acc: 0.9748 - v
al_loss: 0.0870 - val_acc: 0.9730
Epoch 3/20
60000/60000 [==============================] -
 6s 96us/step - loss: 0.0545 - acc: 0.9829 - v
al_loss: 0.0788 - val_acc: 0.9755
Epoch 4/20
60000/60000 [==============================] -
 6s 95us/step - loss: 0.0419 - acc: 0.9863 - v
al_loss: 0.0909 - val_acc: 0.9719
Epoch 5/20
60000/60000 [==============================] -
 6s 97us/step - loss: 0.0343 - acc: 0.9892 - v
al_loss: 0.0789 - val_acc: 0.9767
Epoch 6/20
60000/60000 [==============================] -
 6s 95us/step - loss: 0.0272 - acc: 0.9913 - v
al_loss: 0.0786 - val_acc: 0.9774
Epoch 7/20
60000/60000 [==============================] -
 6s 97us/step - loss: 0.0231 - acc: 0.9927 - v
al_loss: 0.0800 - val_acc: 0.9768
Epoch 8/20
60000/60000 [==============================] -
 6s 97us/step - loss: 0.0231 - acc: 0.9921 - v
al_loss: 0.0824 - val_acc: 0.9745
Epoch 9/20
60000/60000 [==============================] -
 6s 96us/step - loss: 0.0176 - acc: 0.9942 - v
al_loss: 0.0811 - val_acc: 0.9785
Epoch 10/20
60000/60000 [==============================] -
```

```
 6s 97us/step - loss: 0.0186 - acc: 0.9938 - v
al_loss: 0.0731 - val_acc: 0.9793
Epoch 11/20
60000/60000 [==============================] -
 6s 96us/step - loss: 0.0145 - acc: 0.9955 - v
al_loss: 0.0718 - val_acc: 0.9806
Epoch 12/20
60000/60000 [==============================] -
 6s 96us/step - loss: 0.0154 - acc: 0.9949 - v
al_loss: 0.0859 - val_acc: 0.9774
Epoch 13/20
60000/60000 [==============================] -
 6s 96us/step - loss: 0.0144 - acc: 0.9953 - v
al_loss: 0.0721 - val_acc: 0.9817
Epoch 14/20
60000/60000 [==============================] -
 6s 96us/step - loss: 0.0117 - acc: 0.9963 - v
al_loss: 0.0764 - val_acc: 0.9804
Epoch 15/20
60000/60000 [==============================] -
 6s 98us/step - loss: 0.0128 - acc: 0.9954 - v
al_loss: 0.0859 - val_acc: 0.9783
Epoch 16/20
60000/60000 [==============================] -
 6s 96us/step - loss: 0.0118 - acc: 0.9957 - v
al_loss: 0.0766 - val_acc: 0.9819
Epoch 17/20
60000/60000 [==============================] -
 6s 96us/step - loss: 0.0097 - acc: 0.9968 - v
al_loss: 0.0654 - val_acc: 0.9833
Epoch 18/20
60000/60000 [==============================] -
 6s 96us/step - loss: 0.0088 - acc: 0.9970 - v
al_loss: 0.0834 - val_acc: 0.9814
Epoch 19/20
60000/60000 [==============================] -
 6s 96us/step - loss: 0.0093 - acc: 0.9967 - v
```

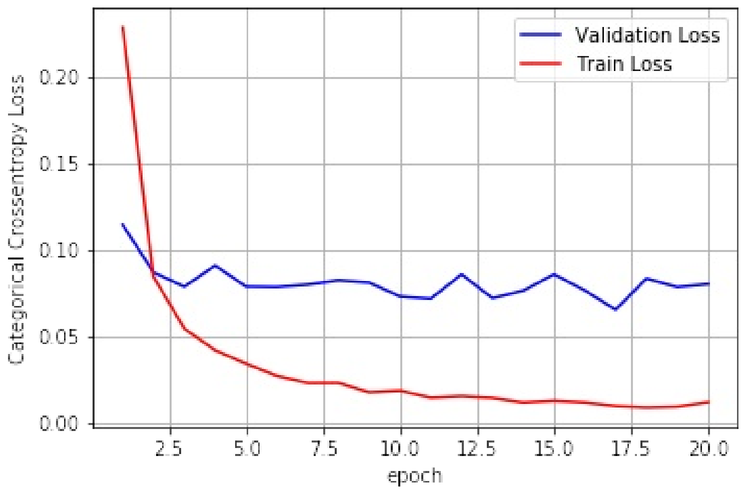
```
al_loss: 0.0786 - val_acc: 0.9807
Epoch 20/20
60000/60000 [==============================] -
 6s 96us/step - loss: 0.0119 - acc: 0.9960 - v
al_loss: 0.0803 - val_acc: 0.9810
```

Train Accuracy=99.60%

In [0]:

```
score = model_batch.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 Crossentr
opy 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 par
amter 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 le
ngth equal to number of epochs

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

```
Test score: 0.08030926787399022
Test accuracy: 0.981
```



5. MLP + Dropout + AdamOptimizer

In [0]:

```
# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormalization-function-in-keras

from keras.layers import Dropout

model_drop = Sequential()

model_drop.add(Dense(356, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal(seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(105, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(51, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

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

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

history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samp
les
Epoch 1/20
60000/60000 [==============================] -
 8s 129us/step - loss: 0.7857 - acc: 0.7579 -
val_loss: 0.2015 - val_acc: 0.9392
Epoch 2/20
60000/60000 [==============================] -
 6s 101us/step - loss: 0.3409 - acc: 0.9055 -
val_loss: 0.1470 - val_acc: 0.9549
Epoch 3/20
60000/60000 [==============================] -
 6s 100us/step - loss: 0.2589 - acc: 0.9297 -
val_loss: 0.1247 - val_acc: 0.9633
Epoch 4/20
60000/60000 [==============================] -
 6s 101us/step - loss: 0.2181 - acc: 0.9413 -
val_loss: 0.1096 - val_acc: 0.9680
Epoch 5/20
60000/60000 [==============================] -
 6s 101us/step - loss: 0.1917 - acc: 0.9476 -
val_loss: 0.1011 - val_acc: 0.9707
Epoch 6/20
60000/60000 [==============================] -
 6s 100us/step - loss: 0.1708 - acc: 0.9537 -
val_loss: 0.0910 - val_acc: 0.9740
Epoch 7/20
60000/60000 [==============================] -
 6s 101us/step - loss: 0.1654 - acc: 0.9554 -
val_loss: 0.0871 - val_acc: 0.9738
Epoch 8/20
60000/60000 [==============================] -
 6s 100us/step - loss: 0.1531 - acc: 0.9587 -
val_loss: 0.0799 - val_acc: 0.9772
Epoch 9/20
60000/60000 [==============================] -
 6s 100us/step - loss: 0.1383 - acc: 0.9612 -
```

```
val_loss: 0.0847 - val_acc: 0.9743
Epoch 10/20
60000/60000 [==============================] -
 6s 100us/step - loss: 0.1316 - acc: 0.9635 -
val_loss: 0.0778 - val_acc: 0.9778
Epoch 11/20
60000/60000 [==============================] -
 6s 102us/step - loss: 0.1238 - acc: 0.9659 -
val_loss: 0.0838 - val_acc: 0.9760
Epoch 12/20
60000/60000 [==============================] -
 6s 101us/step - loss: 0.1150 - acc: 0.9683 -
val_loss: 0.0754 - val_acc: 0.9783
Epoch 13/20
60000/60000 [==============================] -
 6s 101us/step - loss: 0.1154 - acc: 0.9688 -
val_loss: 0.0772 - val_acc: 0.9785
Epoch 14/20
60000/60000 [==============================] -
 6s 101us/step - loss: 0.1121 - acc: 0.9684 -
val_loss: 0.0745 - val_acc: 0.9796
Epoch 15/20
60000/60000 [==============================] -
 6s 101us/step - loss: 0.1095 - acc: 0.9712 -
val_loss: 0.0755 - val_acc: 0.9793
Epoch 16/20
60000/60000 [==============================] -
 6s 101us/step - loss: 0.1032 - acc: 0.9716 -
val_loss: 0.0758 - val_acc: 0.9803
Epoch 17/20
60000/60000 [==============================] -
 6s 101us/step - loss: 0.1026 - acc: 0.9721 -
val_loss: 0.0680 - val_acc: 0.9811
Epoch 18/20
60000/60000 [==============================] -
 6s 101us/step - loss: 0.0977 - acc: 0.9728 -
val_loss: 0.0704 - val_acc: 0.9800
```

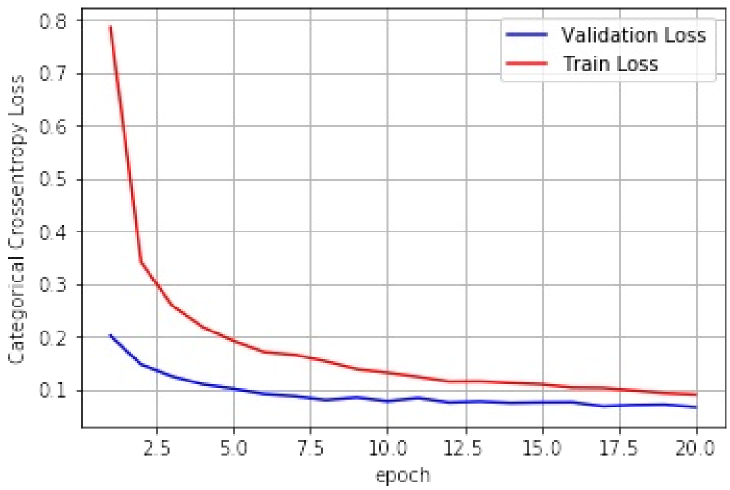
```
Epoch 19/20
60000/60000 [==============================] -
 6s 101us/step - loss: 0.0926 - acc: 0.9738 -
val_loss: 0.0712 - val_acc: 0.9814
Epoch 20/20
60000/60000 [==============================] -
 6s 101us/step - loss: 0.0899 - acc: 0.9751 -
val_loss: 0.0664 - val_acc: 0.9822
```

Train Accuracy=97.51%

In [0]:

```
score = model_batch.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 Crossentr
opy 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 par
amter 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 le
ngth equal to number of epochs

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

```
Test score: 0.08030926787399022
Test accuracy: 0.981
```



Model 3 : (5 Layered) MLP + ReLU + ADAM</2>

In [0]:

```
from keras.initializers import he_normal

model_relu = Sequential()
model_relu.add(Dense(507, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal(seed=None)))
model_relu.add(Dense(312, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_relu.add(Dense(212, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_relu.add(Dense(126, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_relu.add(Dense(84, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

history = model_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

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

```
==============================================
===================
dense_22 (Dense)             (None, 507)
         397995
______________________________________________
___________________
dense_23 (Dense)             (None, 312)
         158496
______________________________________________
___________________
dense_24 (Dense)             (None, 212)
         66356
______________________________________________
___________________
dense_25 (Dense)             (None, 126)
         26838
______________________________________________
___________________
dense_26 (Dense)             (None, 84)
         10668
______________________________________________
___________________
dense_27 (Dense)             (None, 10)
         850
==============================================
===================
Total params: 661,203
Trainable params: 661,203
Non-trainable params: 0
______________________________________________
___________________
None
Train on 60000 samples, validate on 10000 samp
les
Epoch 1/20
60000/60000 [==============================] -
 5s 87us/step - loss: 0.2285 - acc: 0.9311 - v
```

```
al_loss: 0.1176 - val_acc: 0.9640
Epoch 2/20
60000/60000 [==============================] -
 4s 70us/step - loss: 0.0901 - acc: 0.9722 - v
al_loss: 0.0963 - val_acc: 0.9717
Epoch 3/20
60000/60000 [==============================] -
 4s 70us/step - loss: 0.0604 - acc: 0.9811 - v
al_loss: 0.0991 - val_acc: 0.9703
Epoch 4/20
60000/60000 [==============================] -
 4s 69us/step - loss: 0.0468 - acc: 0.9852 - v
al_loss: 0.0948 - val_acc: 0.9725
Epoch 5/20
60000/60000 [==============================] -
 4s 69us/step - loss: 0.0349 - acc: 0.9888 - v
al_loss: 0.0800 - val_acc: 0.9759
Epoch 6/20
60000/60000 [==============================] -
 4s 71us/step - loss: 0.0338 - acc: 0.9895 - v
al_loss: 0.0943 - val_acc: 0.9751
Epoch 7/20
60000/60000 [==============================] -
 4s 70us/step - loss: 0.0270 - acc: 0.9914 - v
al_loss: 0.0792 - val_acc: 0.9791
Epoch 8/20
60000/60000 [==============================] -
 4s 68us/step - loss: 0.0291 - acc: 0.9907 - v
al_loss: 0.0827 - val_acc: 0.9791
Epoch 9/20
60000/60000 [==============================] -
 4s 70us/step - loss: 0.0198 - acc: 0.9933 - v
al_loss: 0.0813 - val_acc: 0.9803
Epoch 10/20
60000/60000 [==============================] -
 4s 69us/step - loss: 0.0192 - acc: 0.9940 - v
al_loss: 0.0825 - val_acc: 0.9805
```

```
Epoch 11/20
60000/60000 [==============================] -
 4s 70us/step - loss: 0.0187 - acc: 0.9942 - v
al_loss: 0.0992 - val_acc: 0.9739
Epoch 12/20
60000/60000 [==============================] -
 4s 69us/step - loss: 0.0185 - acc: 0.9941 - v
al_loss: 0.0894 - val_acc: 0.9785
Epoch 13/20
60000/60000 [==============================] -
 4s 69us/step - loss: 0.0154 - acc: 0.9954 - v
al_loss: 0.0873 - val_acc: 0.9803
Epoch 14/20
60000/60000 [==============================] -
 4s 69us/step - loss: 0.0163 - acc: 0.9949 - v
al_loss: 0.0815 - val_acc: 0.9817
Epoch 15/20
60000/60000 [==============================] -
 4s 69us/step - loss: 0.0109 - acc: 0.9969 - v
al_loss: 0.1100 - val_acc: 0.9771
Epoch 16/20
60000/60000 [==============================] -
 4s 69us/step - loss: 0.0127 - acc: 0.9962 - v
al_loss: 0.0869 - val_acc: 0.9807
Epoch 17/20
60000/60000 [==============================] -
 4s 69us/step - loss: 0.0153 - acc: 0.9952 - v
al_loss: 0.0987 - val_acc: 0.9795
Epoch 18/20
60000/60000 [==============================] -
 4s 69us/step - loss: 0.0151 - acc: 0.9956 - v
al_loss: 0.0793 - val_acc: 0.9826
Epoch 19/20
60000/60000 [==============================] -
 4s 69us/step - loss: 0.0097 - acc: 0.9971 - v
al_loss: 0.0884 - val_acc: 0.9817
Epoch 20/20
```

```
60000/60000 [==============================] -
 4s 68us/step - loss: 0.0091 - acc: 0.9973 - v
al_loss: 0.0729 - val_acc: 0.9835
```

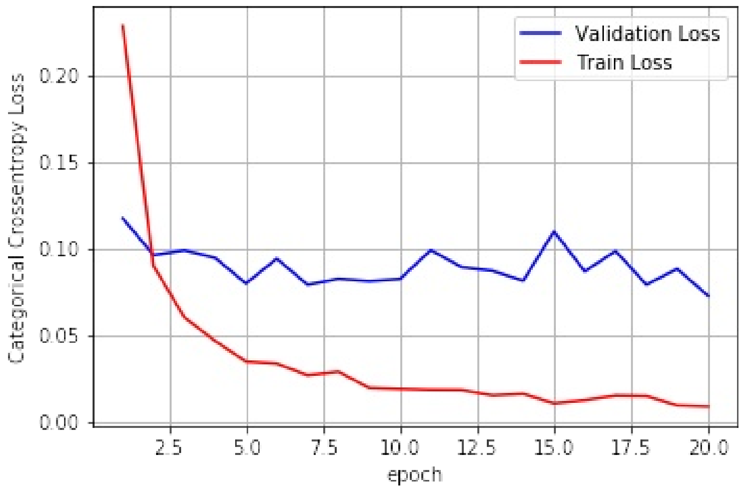
Train Accuracy=99.73%

In [0]:

```
score = model_batch.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 Crossentr
opy 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 par
amter 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 le
ngth equal to number of epochs

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

Test score: 0.08030926787399022

Test accuracy: 0.981



MLP + Batch-Norm on hidden Layers + AdamOptimizer </2>

In [0]:

```
from keras.layers.normalization import BatchNormalization

model_batch = Sequential()

model_batch.add(Dense(507, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal(seed=None)))
model_batch.add(BatchNormalization())

model_batch.add(Dense(312, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_batch.add(BatchNormalization())

model_batch.add(Dense(212, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_batch.add(BatchNormalization())

model_batch.add(Dense(126, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_batch.add(BatchNormalization())

model_batch.add(Dense(84, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_batch.add(BatchNormalization())

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

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

```
history = model_batch.fit(X_train, Y_train, batch_size=batch_
size, epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_
test))
```

```
Train on 60000 samples, validate on 10000 samp
les
Epoch 1/20
60000/60000 [==============================] -
 11s 183us/step - loss: 0.2148 - acc: 0.9356 -
 val_loss: 0.1066 - val_acc: 0.9677
Epoch 2/20
60000/60000 [==============================] -
 9s 147us/step - loss: 0.0867 - acc: 0.9731 -
val_loss: 0.0811 - val_acc: 0.9742
Epoch 3/20
60000/60000 [==============================] -
 9s 148us/step - loss: 0.0587 - acc: 0.9815 -
val_loss: 0.0947 - val_acc: 0.9702
Epoch 4/20
60000/60000 [==============================] -
 9s 149us/step - loss: 0.0505 - acc: 0.9831 -
val_loss: 0.0926 - val_acc: 0.9711
Epoch 5/20
60000/60000 [==============================] -
 9s 149us/step - loss: 0.0407 - acc: 0.9867 -
val_loss: 0.0816 - val_acc: 0.9761
Epoch 6/20
60000/60000 [==============================] -
 9s 148us/step - loss: 0.0365 - acc: 0.9879 -
val_loss: 0.0848 - val_acc: 0.9742
Epoch 7/20
60000/60000 [==============================] -
 9s 149us/step - loss: 0.0302 - acc: 0.9899 -
val_loss: 0.0784 - val_acc: 0.9779
Epoch 8/20
60000/60000 [==============================] -
```

```
 9s 148us/step - loss: 0.0270 - acc: 0.9908 -
val_loss: 0.0772 - val_acc: 0.9766
Epoch 9/20
60000/60000 [==============================] -
 9s 148us/step - loss: 0.0248 - acc: 0.9919 -
val_loss: 0.0743 - val_acc: 0.9766
Epoch 10/20
60000/60000 [==============================] -
 9s 150us/step - loss: 0.0207 - acc: 0.9933 -
val_loss: 0.0866 - val_acc: 0.9755
Epoch 11/20
60000/60000 [==============================] -
 9s 149us/step - loss: 0.0235 - acc: 0.9920 -
val_loss: 0.0823 - val_acc: 0.9757
Epoch 12/20
60000/60000 [==============================] -
 9s 148us/step - loss: 0.0221 - acc: 0.9923 -
val_loss: 0.0769 - val_acc: 0.9786
Epoch 13/20
60000/60000 [==============================] -
 9s 148us/step - loss: 0.0178 - acc: 0.9942 -
val_loss: 0.0716 - val_acc: 0.9803
Epoch 14/20
60000/60000 [==============================] -
 9s 149us/step - loss: 0.0171 - acc: 0.9946 -
val_loss: 0.0803 - val_acc: 0.9796
Epoch 15/20
60000/60000 [==============================] -
 9s 148us/step - loss: 0.0164 - acc: 0.9944 -
val_loss: 0.0763 - val_acc: 0.9790
Epoch 16/20
60000/60000 [==============================] -
 9s 149us/step - loss: 0.0158 - acc: 0.9948 -
val_loss: 0.0727 - val_acc: 0.9808
Epoch 17/20
60000/60000 [==============================] -
 9s 148us/step - loss: 0.0148 - acc: 0.9951 -
```

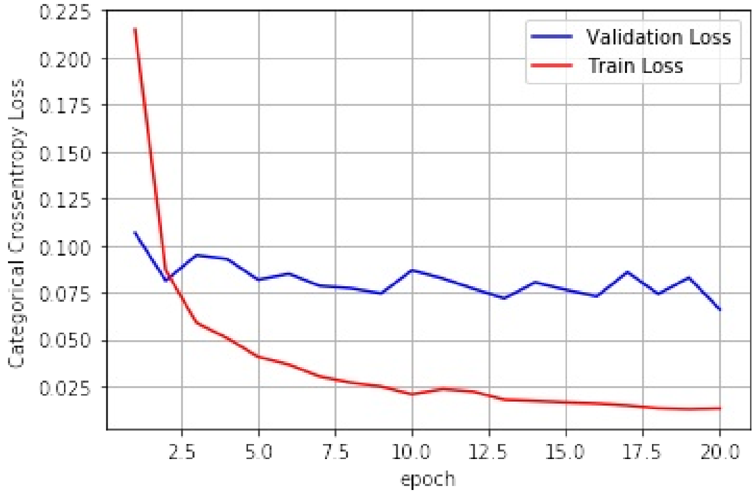
```
val_loss: 0.0857 - val_acc: 0.9761
Epoch 18/20
60000/60000 [==============================] -
 9s 149us/step - loss: 0.0132 - acc: 0.9955 -
val_loss: 0.0741 - val_acc: 0.9820
Epoch 19/20
60000/60000 [==============================] -
 9s 149us/step - loss: 0.0128 - acc: 0.9958 -
val_loss: 0.0828 - val_acc: 0.9802
Epoch 20/20
60000/60000 [==============================] -
 9s 149us/step - loss: 0.0132 - acc: 0.9957 -
val_loss: 0.0658 - val_acc: 0.9831
```

Train Accuracy=99.57%

In [0]:

```
score = model_batch.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 Crossentr
opy 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 par
amter 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 le
ngth equal to number of epochs

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

```
Test score: 0.0658267695394985
Test accuracy: 0.9831
```



5. MLP + Dropout + AdamOptimizer

In [0]:

```
# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormalization-function-in-keras

from keras.layers import Dropout

model_drop = Sequential()

model_drop.add(Dense(507, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal(seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(312, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(212, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(126, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(84, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_drop.add(BatchNormalization())
```

```
model_drop.add(Dropout(0.5))

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

model_drop.compile(optimizer='adam', loss='categorical_crosse
ntropy', metrics=['accuracy'])

history = model_drop.fit(X_train, Y_train, batch_size=batch_s
ize, epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_t
est))
```

```
Train on 60000 samples, validate on 10000 samp
les
Epoch 1/20
60000/60000 [==============================] -
 12s 201us/step - loss: 1.1379 - acc: 0.6438 -
 val_loss: 0.2566 - val_acc: 0.9249
Epoch 2/20
60000/60000 [==============================] -
 9s 156us/step - loss: 0.3986 - acc: 0.8896 -
val_loss: 0.1720 - val_acc: 0.9503
Epoch 3/20
60000/60000 [==============================] -
 9s 156us/step - loss: 0.2922 - acc: 0.9221 -
val_loss: 0.1444 - val_acc: 0.9598
Epoch 4/20
60000/60000 [==============================] -
 9s 156us/step - loss: 0.2413 - acc: 0.9353 -
val_loss: 0.1243 - val_acc: 0.9661
Epoch 5/20
60000/60000 [==============================] -
 9s 157us/step - loss: 0.2142 - acc: 0.9436 -
val_loss: 0.1076 - val_acc: 0.9711
Epoch 6/20
60000/60000 [==============================] -
 9s 157us/step - loss: 0.1873 - acc: 0.9498 -
val_loss: 0.1088 - val_acc: 0.9701
```

```
Epoch 7/20
60000/60000 [==============================] -
 9s 157us/step - loss: 0.1751 - acc: 0.9537 -
val_loss: 0.0930 - val_acc: 0.9749
Epoch 8/20
60000/60000 [==============================] -
 9s 156us/step - loss: 0.1660 - acc: 0.9568 -
val_loss: 0.0918 - val_acc: 0.9758
Epoch 9/20
60000/60000 [==============================] -
 9s 157us/step - loss: 0.1491 - acc: 0.9610 -
val_loss: 0.0863 - val_acc: 0.9770
Epoch 10/20
60000/60000 [==============================] -
 9s 156us/step - loss: 0.1431 - acc: 0.9621 -
val_loss: 0.0819 - val_acc: 0.9774
Epoch 11/20
60000/60000 [==============================] -
 9s 158us/step - loss: 0.1352 - acc: 0.9640 -
val_loss: 0.0807 - val_acc: 0.9787
Epoch 12/20
60000/60000 [==============================] -
 9s 157us/step - loss: 0.1289 - acc: 0.9667 -
val_loss: 0.0772 - val_acc: 0.9796
Epoch 13/20
60000/60000 [==============================] -
 9s 155us/step - loss: 0.1282 - acc: 0.9663 -
val_loss: 0.0778 - val_acc: 0.9804
Epoch 14/20
60000/60000 [==============================] -
 9s 157us/step - loss: 0.1235 - acc: 0.9679 -
val_loss: 0.0799 - val_acc: 0.9793
Epoch 15/20
60000/60000 [==============================] -
 9s 157us/step - loss: 0.1164 - acc: 0.9699 -
val_loss: 0.0785 - val_acc: 0.9803
Epoch 16/20
```

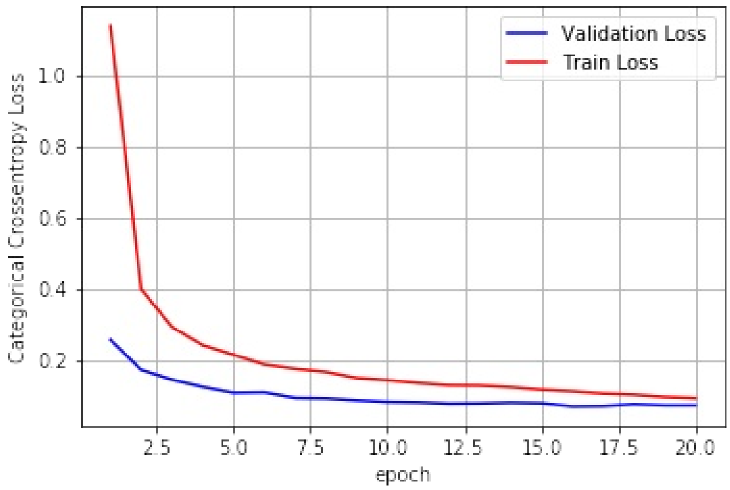
```
60000/60000 [==============================] -
 9s 156us/step - loss: 0.1118 - acc: 0.9702 -
val_loss: 0.0692 - val_acc: 0.9820
Epoch 17/20
60000/60000 [==============================] -
 9s 156us/step - loss: 0.1055 - acc: 0.9726 -
val_loss: 0.0700 - val_acc: 0.9823
Epoch 18/20
60000/60000 [==============================] -
 9s 156us/step - loss: 0.1022 - acc: 0.9731 -
val_loss: 0.0752 - val_acc: 0.9812
Epoch 19/20
60000/60000 [==============================] -
 9s 156us/step - loss: 0.0957 - acc: 0.9750 -
val_loss: 0.0721 - val_acc: 0.9820
Epoch 20/20
60000/60000 [==============================] -
 10s 158us/step - loss: 0.0923 - acc: 0.9760 -
 val_loss: 0.0721 - val_acc: 0.9820
```

Train Accuracy=97.60%

In [0]:

```
score = model_batch.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 Crossentr
opy 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 par
amter 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 le
ngth equal to number of epochs

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

```
Test score: 0.0658267695394985
Test accuracy: 0.9831
```



In [1]:

```
# Please compare all your models using Prettytable library
# http://zetcode.com/python/prettytable/

from prettytable import PrettyTable

#If you get a ModuleNotFoundError error , install prettytable
 using: pip3 install prettytable

x = PrettyTable()
x.field_names = ["Hidden Layers", "Activation Unit","Optimise
r", "Batch Normalisation", "DropOut(0.5)"," Train Accuracy","
Test_Accuracy"]
x.add_row(["2", "ReLU","Adam","-","-" ,"99.75%", "97.94%"])
x.add_row(["2", "ReLU","Adam","Yes","-" ,"99.77%", "98.02%"])
x.add_row(["2", "ReLU","Adam","Yes","0.5" ,"97.39%", "97.96%"
])

x.add_row(["2", "ReLU","Adam","-","-" ,"99.64%", "98.3%"])
x.add_row(["2", "ReLU","Adam","Yes","-" ,"99.60%", "98.1%"])
x.add_row(["2", "ReLU","Adam","Yes","0.5" ,"97.51%", "98.1%"]
```

```
)

x.add_row(["2", "ReLU","Adam","-","-" ,"99.73%", "98.1%"])
x.add_row(["2", "ReLU","Adam","Yes","-" ,"99.57%", "97.31%"])
x.add_row(["2", "ReLU","Adam","Yes","0.5" ,"97.60%", "98.31%"
])

print(x)
```

```
+---------------+-----------------+-----------
+---------------------+--------------+--------
---------+---------------+
| Hidden Layers | Activation Unit | Optimiser
| Batch Normalisation | DropOut(0.5) |  Train
Accuracy | Test_Accuracy |
+---------------+-----------------+-----------
+---------------------+--------------+--------
---------+---------------+
|       2       |       ReLU      |    Adam
|          -          |      -       |      99
.75%     |     97.94%    |
|       2       |       ReLU      |    Adam
|         Yes         |      -       |      99
.77%     |     98.02%    |
|       2       |       ReLU      |    Adam
|         Yes         |     0.5      |      97
.39%     |     97.96%    |
|       2       |       ReLU      |    Adam
|          -          |      -       |      99
.64%     |     98.3%     |
|       2       |       ReLU      |    Adam
|         Yes         |      -       |      99
.60%     |     98.1%     |
|       2       |       ReLU      |    Adam
|         Yes         |     0.5      |      97
.51%     |     98.1%     |
|       2       |       ReLU      |    Adam
```

```
|          -          |      -       |      99
.73%     |     98.1%     |
|       2       |       ReLU      |    Adam
|         Yes         |      -       |      99
.57%     |     97.31%    |
|       2       |       ReLU      |    Adam
|         Yes         |     0.5      |      97
.60%     |     98.31%    |
+---------------+-----------------+-----------
+---------------------+--------------+--------
---------+---------------+
```

Conclusion:

1. While using Batch Normalisation and Dropouts together,we get better results with good accuracy.
2. Model 3 with many layeres gave better results compared to Model 1 and Model 2.