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
```

In [0]:

```
import matplotlib.pyplot as plt
%matplotlib inline
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, colors=['b']):
    plt.plot(x, vy, color = 'b', label='Validation Loss')
    plt.plot(x, ty, color = 'r', label='Train Loss')
    plt.xlabel('epoch')
    plt.ylabel('Categorical Crossentropy Loss')
    plt.legend()
    plt.grid()
    plt.show();
```

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 [==============================] - 1s 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.s
hape[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 [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_test.shape[2])
```

In [0]:

```
# after converting the input images from 3d to 2d vectors

print("Number of training examples :", X_train.shape[0], "and each image is of shape (%d)"%(X_train.sha
pe[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 [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  49 238 253 253 253 253 253 253 253 253 251  93  82
  82  56  39   0   0   0   0   0   0   0   0   0   0   0   0  18 219 253
 253 253 253 253 198 182 247 241   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0  14   1 154 253  90   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0  11 190 253  70   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  35 241
 225 160 108   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0  81 240 253 253 119  25   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  45 186 253 253 150  27   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252 253 187
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0 249 253 249  64   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253
 253 207   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0  39 148 229 253 253 253 250 182   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253
 253 201  78   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  23  66 213 253 253 253 253 198  81   2   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0  18 171 219 253 253 253 253 195
  80   9   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  55 172 226 253 253 253 253 244 133  11   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0 136 253 253 253 212 135 132  16
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
```

In [0]:

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

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

In [0]:

```python
# 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.          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.          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 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 suppor
ted 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, Dropout, BatchNormalization, Activation
```

In [0]:

```
# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 20
```

1. two hidden layers

In [0]:

```
# start building a model
model = Sequential()
```

```
# The model needs to know what input shape it should expect.
# For this reason, the first layer in a Sequential model
# (and only the first, because following layers can do automatic shape inference)
# needs to receive information about its input shape.
# you can use input_shape and input_dim to pass the shape of input

# output_dim represent the number of nodes need in that layer
# here we have 10 nodes

model.add(Dense(units=256, input_shape=(input_dim,), activation='relu'))
model.add(Dropout(0.25))

model.add(Dense(128))
model.add(BatchNormalization())
model.add(Activation('relu'))

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

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:541:
The name tf.placeholder is deprecated. Please use tf.compat.v1.placeholder instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:4432
: The name tf.random_uniform is deprecated. Please use tf.random.uniform instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:148:
The name tf.placeholder_with_default is deprecated. Please use tf.compat.v1.placeholder_with_default in
stead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:3733
: 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`.
```

In [0]:

```
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 256)               200960
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_4 (Dense)              (None, 128)               32896
_________________________________________________________________
batch_normalization_1 (Batch (None, 128)               512
_________________________________________________________________
activation_1 (Activation)    (None, 128)               0
_________________________________________________________________
dense_5 (Dense)              (None, 10)                1290
=================================================================
Total params: 235,658
Trainable params: 235,402
Non-trainable params: 256
_________________________________________________________________
```

In [0]:

```
# Before training a model, you need to configure the learning process, which is done via the compile me
thod

# It receives three arguments:
# An optimizer. This could be the string identifier of an existing optimizer , https://keras.io/optimiz
ers/
# A loss function. This is the objective that the model will try to minimize., https://keras.io/losses/
# A list of metrics. For any classification problem you will want to set this to metrics=['accuracy'].
https://keras.io/metrics/
```

```
# Note: when using the categorical_crossentropy loss, your targets should be in categorical format
# (e.g. if you have 10 classes, the target for each sample should be a 10-dimensional vector that is al
l-zeros except
# for a 1 at the index corresponding to the class of the sample).

# that is why we converted out labels into vectors

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

# Keras models are trained on Numpy arrays of input data and labels.
# For training a model, you will typically use the  fit function

# fit(self, x=None, y=None, batch_size=None, epochs=1, verbose=1, callbacks=None, validation_split=0.0,

# validation_data=None, shuffle=True, class_weight=None, sample_weight=None, initial_epoch=0, steps_per
_epoch=None,
# validation_steps=None)

# fit() function Trains the model for a fixed number of epochs (iterations on a dataset).

# it returns A History object. Its History.history attribute is a record of training loss values and
# metrics values at successive epochs, as well as validation loss values and validation metrics values
(if applicable).

# https://github.com/openai/baselines/issues/20

history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_dat
a=(X_test, Y_test))
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimizers.py:793: The name tf.tra
in.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:3576
: The name tf.log is deprecated. Please use tf.math.log instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/ops/math_grad.py:
1424: where (from tensorflow.python.ops.array_ops) is deprecated and will be removed in a future versio
n.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:1033
: The name tf.assign_add is deprecated. Please use tf.compat.v1.assign_add instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:1020
: The name tf.assign is deprecated. Please use tf.compat.v1.assign instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:3005
: The name tf.Session is deprecated. Please use tf.compat.v1.Session instead.

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:190:
The name tf.get_default_session is deprecated. Please use tf.compat.v1.get_default_session instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:197:
The name tf.ConfigProto is deprecated. Please use tf.compat.v1.ConfigProto instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:207:
The name tf.global_variables is deprecated. Please use tf.compat.v1.global_variables instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:216:
The name tf.is_variable_initialized is deprecated. Please use tf.compat.v1.is_variable_initialized inst
ead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:223:
The name tf.variables_initializer is deprecated. Please use tf.compat.v1.variables_initializer instead.

60000/60000 [==============================] - 13s 214us/step - loss: 0.2877 - acc: 0.9166 - val_loss:
0.1143 - val_acc: 0.9649
Epoch 2/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.1233 - acc: 0.9627 - val_loss: 0.
0876 - val_acc: 0.9733
Epoch 3/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0895 - acc: 0.9723 - val_loss: 0.
0763 - val_acc: 0.9774
Epoch 4/20
```

```
60000/60000 [==============================] - 4s 65us/step - loss: 0.0726 - acc: 0.9771 - val_loss: 0.
0655 - val_acc: 0.9796
Epoch 5/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0631 - acc: 0.9797 - val_loss: 0.
0640 - val_acc: 0.9802
Epoch 6/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0536 - acc: 0.9831 - val_loss: 0.
0668 - val_acc: 0.9797
Epoch 7/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0470 - acc: 0.9845 - val_loss: 0.
0676 - val_acc: 0.9798
Epoch 8/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.0454 - acc: 0.9854 - val_loss: 0.
0634 - val_acc: 0.9798
Epoch 9/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.0395 - acc: 0.9870 - val_loss: 0.
0683 - val_acc: 0.9790
Epoch 10/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0393 - acc: 0.9868 - val_loss: 0.
0569 - val_acc: 0.9826
Epoch 11/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.0311 - acc: 0.9894 - val_loss: 0.
0608 - val_acc: 0.9811
Epoch 12/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0309 - acc: 0.9894 - val_loss: 0.
0626 - val_acc: 0.9826
Epoch 13/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.0283 - acc: 0.9902 - val_loss: 0.
0618 - val_acc: 0.9818
Epoch 14/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.0274 - acc: 0.9903 - val_loss: 0.
0610 - val_acc: 0.9811
Epoch 15/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0273 - acc: 0.9907 - val_loss: 0.
0597 - val_acc: 0.9826
Epoch 16/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0250 - acc: 0.9916 - val_loss: 0.
0637 - val_acc: 0.9823
Epoch 17/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0237 - acc: 0.9915 - val_loss: 0.
0576 - val_acc: 0.9833
Epoch 18/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.0219 - acc: 0.9925 - val_loss: 0.
0646 - val_acc: 0.9830
Epoch 19/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0206 - acc: 0.9929 - val_loss: 0.
0611 - val_acc: 0.9822
Epoch 20/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0229 - acc: 0.9921 - val_loss: 0.
0633 - val_acc: 0.9821
```

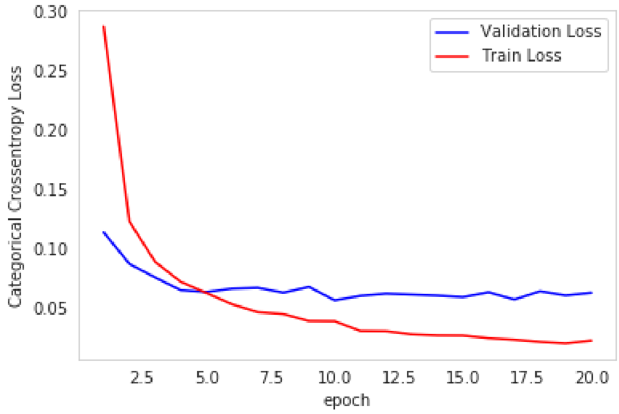
In [0]:

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

# 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, validat
ion_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.history['val_loss']
ty = history.history['loss']
```

```
Test score: 0.063285964876238
Test accuracy: 0.9821
```

In [0]:

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



2. three hidden layer

In [0]:

```
# start building a model
model = Sequential()

# The model needs to know what input shape it should expect.
# For this reason, the first layer in a Sequential model
# (and only the first, because following layers can do automatic shape inference)
# needs to receive information about its input shape.
# you can use input_shape and input_dim to pass the shape of input

# output_dim represent the number of nodes need in that layer
# here we have 10 nodes

model.add(Dense(units=256, input_shape=(input_dim,), activation='relu'))
model.add(Dropout(0.25))

model.add(Dense(units=256, activation='relu'))
model.add(Dropout(0.25))

model.add(Dense(128))
model.add(BatchNormalization())
model.add(Activation('relu'))

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

In [0]:

```
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 256)               200960
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_7 (Dense)              (None, 256)               65792
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_8 (Dense)              (None, 128)               32896
```

```
_________________________________________________________________
batch_normalization_2 (Batch (None, 128)               512
_________________________________________________________________
activation_2 (Activation)    (None, 128)               0
_________________________________________________________________
dense_9 (Dense)              (None, 10)                1290
=================================================================
Total params: 301,450
Trainable params: 301,194
Non-trainable params: 256
_________________________________________________________________
```

In [0]:

```python
# Before training a model, you need to configure the learning process, which is done via the compile me
thod

# It receives three arguments:
# An optimizer. This could be the string identifier of an existing optimizer , https://keras.io/optimiz
ers/
# A loss function. This is the objective that the model will try to minimize., https://keras.io/losses/
# A list of metrics. For any classification problem you will want to set this to metrics=['accuracy'].
https://keras.io/metrics/


# Note: when using the categorical_crossentropy loss, your targets should be in categorical format
# (e.g. if you have 10 classes, the target for each sample should be a 10-dimensional vector that is al
l-zeros except
# for a 1 at the index corresponding to the class of the sample).

# that is why we converted out labels into vectors

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

# Keras models are trained on Numpy arrays of input data and labels.
# For training a model, you will typically use the  fit function

# fit(self, x=None, y=None, batch_size=None, epochs=1, verbose=1, callbacks=None, validation_split=0.0,

# validation_data=None, shuffle=True, class_weight=None, sample_weight=None, initial_epoch=0, steps_per
_epoch=None,
# validation_steps=None)

# fit() function Trains the model for a fixed number of epochs (iterations on a dataset).

# it returns A History object. Its History.history attribute is a record of training loss values and
# metrics values at successive epochs, as well as validation loss values and validation metrics values
(if applicable).

# https://github.com/openai/baselines/issues/20

history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_dat
a=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 80us/step - loss: 0.3157 - acc: 0.9054 - val_loss: 0.
1099 - val_acc: 0.9668
Epoch 2/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.1349 - acc: 0.9589 - val_loss: 0.
0921 - val_acc: 0.9710
Epoch 3/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.1037 - acc: 0.9682 - val_loss: 0.
0806 - val_acc: 0.9756
Epoch 4/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0863 - acc: 0.9732 - val_loss: 0.
0690 - val_acc: 0.9798
Epoch 5/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.0741 - acc: 0.9765 - val_loss: 0.
0660 - val_acc: 0.9807
Epoch 6/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0650 - acc: 0.9792 - val_loss: 0.
0669 - val_acc: 0.9801
Epoch 7/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0567 - acc: 0.9817 - val_loss: 0.
```

```
60000/60000 [==============================] - 4s 71us/step - loss: 0.0567 - acc: 0.9817 - val_loss: 0.
0671 - val_acc: 0.9807
Epoch 8/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0533 - acc: 0.9826 - val_loss: 0.
0682 - val_acc: 0.9808
Epoch 9/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.0489 - acc: 0.9838 - val_loss: 0.
0673 - val_acc: 0.9811
Epoch 10/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0440 - acc: 0.9864 - val_loss: 0.
0680 - val_acc: 0.9807
Epoch 11/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0422 - acc: 0.9864 - val_loss: 0.
0669 - val_acc: 0.9813
Epoch 12/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0398 - acc: 0.9870 - val_loss: 0.
0680 - val_acc: 0.9820
Epoch 13/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.0363 - acc: 0.9883 - val_loss: 0.
0602 - val_acc: 0.9836
Epoch 14/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.0322 - acc: 0.9890 - val_loss: 0.
0620 - val_acc: 0.9831
Epoch 15/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0344 - acc: 0.9882 - val_loss: 0.
0583 - val_acc: 0.9817
Epoch 16/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0303 - acc: 0.9899 - val_loss: 0.
0648 - val_acc: 0.9828
Epoch 17/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.0299 - acc: 0.9900 - val_loss: 0.
0686 - val_acc: 0.9819
Epoch 18/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.0286 - acc: 0.9904 - val_loss: 0.
0619 - val_acc: 0.9829
Epoch 19/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0266 - acc: 0.9913 - val_loss: 0.
0627 - val_acc: 0.9835
Epoch 20/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0270 - acc: 0.9912 - val_loss: 0.
0567 - val_acc: 0.9850
```

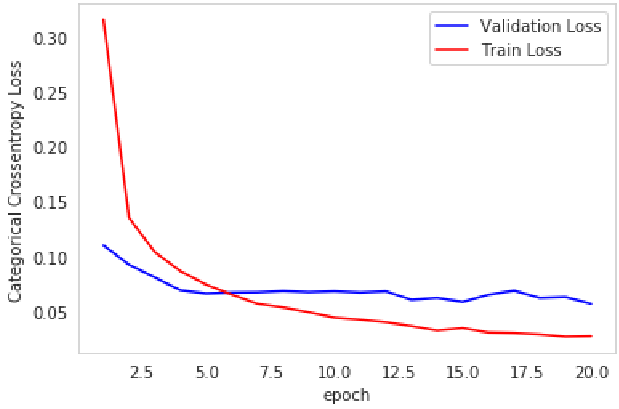
In [0]:

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

# 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, validat
ion_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.history['val_loss']
ty = history.history['loss']
```

```
Test score: 0.05666124046119221
Test accuracy: 0.985
```

In [0]:

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



3. five hidden layer

In [0]:

```
# start building a model
model = Sequential()

# The model needs to know what input shape it should expect.
# For this reason, the first layer in a Sequential model
# (and only the first, because following layers can do automatic shape inference)
# needs to receive information about its input shape.
# you can use input_shape and input_dim to pass the shape of input

# output_dim represent the number of nodes need in that layer
# here we have 10 nodes

model.add(Dense(units=256, input_shape=(input_dim,), activation='relu'))
model.add(Dropout(0.25))

model.add(Dense(units=256, activation='relu'))
model.add(Dropout(0.25))

model.add(Dense(units=512, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(units=256, activation='relu'))
model.add(Dropout(0.25))

model.add(Dense(128))
model.add(BatchNormalization())
model.add(Activation('relu'))

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

In [0]:

```
model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_16 (Dense)             (None, 256)               200960
_________________________________________________________________
dropout_8 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_17 (Dense)             (None, 256)               65792
_________________________________________________________________
dropout_9 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_18 (Dense)             (None, 512)               131584
_________________________________________________________________
dropout_10 (Dropout)         (None, 512)               0
_________________________________________________________________
```

```
dense_19 (Dense)             (None, 256)               131328    
_________________________________________________________________
dropout_11 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_20 (Dense)             (None, 128)               32896     
_________________________________________________________________
batch_normalization_4 (Batch (None, 128)               512       
_________________________________________________________________
activation_4 (Activation)    (None, 128)               0         
_________________________________________________________________
dense_21 (Dense)             (None, 10)                1290      
=================================================================
Total params: 564,362
Trainable params: 564,106
Non-trainable params: 256
_________________________________________________________________
```

In [0]:

```
# Before training a model, you need to configure the learning process, which is done via the compile me
thod

# It receives three arguments:
# An optimizer. This could be the string identifier of an existing optimizer , https://keras.io/optimiz
ers/
# A loss function. This is the objective that the model will try to minimize., https://keras.io/losses/
# A list of metrics. For any classification problem you will want to set this to metrics=['accuracy'].
https://keras.io/metrics/


# Note: when using the categorical_crossentropy loss, your targets should be in categorical format
# (e.g. if you have 10 classes, the target for each sample should be a 10-dimensional vector that is al
l-zeros except
# for a 1 at the index corresponding to the class of the sample).

# that is why we converted out labels into vectors

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

# Keras models are trained on Numpy arrays of input data and labels.
# For training a model, you will typically use the  fit function

# fit(self, x=None, y=None, batch_size=None, epochs=1, verbose=1, callbacks=None, validation_split=0.0,

# validation_data=None, shuffle=True, class_weight=None, sample_weight=None, initial_epoch=0, steps_per
_epoch=None,
# validation_steps=None)

# fit() function Trains the model for a fixed number of epochs (iterations on a dataset).

# it returns A History object. Its History.history attribute is a record of training loss values and
# metrics values at successive epochs, as well as validation loss values and validation metrics values
(if applicable).

# https://github.com/openai/baselines/issues/20

history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_dat
a=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 6s 103us/step - loss: 0.4301 - acc: 0.8676 - val_loss: 0
.1490 - val_acc: 0.9553
Epoch 2/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.1758 - acc: 0.9497 - val_loss: 0.
1054 - val_acc: 0.9684
Epoch 3/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.1336 - acc: 0.9618 - val_loss: 0.
0877 - val_acc: 0.9751
Epoch 4/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.1079 - acc: 0.9682 - val_loss: 0.
0876 - val_acc: 0.9758
Epoch 5/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0968 - acc: 0.9719 - val_loss: 0.
0837 - val_acc: 0.9764
```

```
Epoch 6/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.0852 - acc: 0.9759 - val_loss: 0.
0800 - val_acc: 0.9777
Epoch 7/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.0804 - acc: 0.9762 - val_loss: 0.
0677 - val_acc: 0.9805
Epoch 8/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.0717 - acc: 0.9795 - val_loss: 0.
0703 - val_acc: 0.9804
Epoch 9/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.0652 - acc: 0.9808 - val_loss: 0.
0771 - val_acc: 0.9790
Epoch 10/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.0639 - acc: 0.9817 - val_loss: 0.
0739 - val_acc: 0.9799
Epoch 11/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.0552 - acc: 0.9834 - val_loss: 0.
0820 - val_acc: 0.9794
Epoch 12/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.0530 - acc: 0.9847 - val_loss: 0.
0704 - val_acc: 0.9801
Epoch 13/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.0513 - acc: 0.9850 - val_loss: 0.
0687 - val_acc: 0.9817
Epoch 14/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.0450 - acc: 0.9866 - val_loss: 0.
0743 - val_acc: 0.9808
Epoch 15/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.0464 - acc: 0.9861 - val_loss: 0.
0696 - val_acc: 0.9816
Epoch 16/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.0464 - acc: 0.9862 - val_loss: 0.
0671 - val_acc: 0.9831
Epoch 17/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0391 - acc: 0.9888 - val_loss: 0.
0779 - val_acc: 0.9796
Epoch 18/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.0399 - acc: 0.9878 - val_loss: 0.
0690 - val_acc: 0.9835
Epoch 19/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.0390 - acc: 0.9882 - val_loss: 0.
0624 - val_acc: 0.9841
Epoch 20/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.0326 - acc: 0.9904 - val_loss: 0.
0728 - val_acc: 0.9827
```

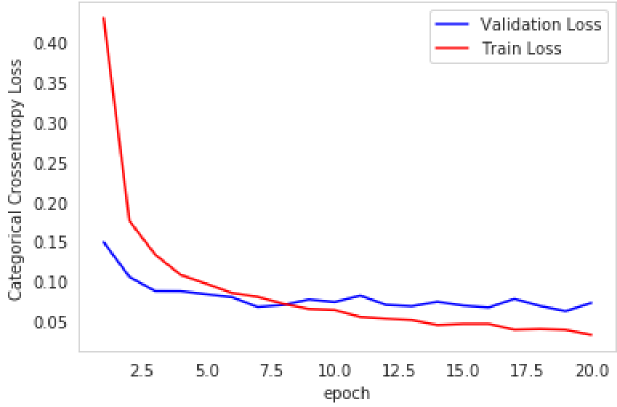
In [0]:

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

# 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, validat
ion_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.history['val_loss']
ty = history.history['loss']
```

```
Test score: 0.07284883902525763
Test accuracy: 0.9827
```

In [0]:

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



In [0]:

```
from prettytable import PrettyTable
```

In [0]:

```
x = PrettyTable()
x.field_names = ["#layers", "Train Loss/Acc","Test Loss/Acc"]
```

In [0]:

```
x.add_row(['2 layers (256(p=25%),128(BN))','2.3% / 99.21%','6.3% / 98.21%'])
x.add_row(['3 layers (256(p=25%),256(p=25%),128(BN))','2.7% / 99.12%','5.7% / 98.5%'])
x.add_row(['5 layers (256(p=25%),256(p=25%),512(p=50%),256(p=25%),128(BN))','3.3% / 99.04%','7.3% / 98.
27%'])
```

In [99]:

```
print(x)
```

```
+------------------------------------------------------------------+----------------+---------------+
|                             #layers                              | Train Loss/Acc | Test Loss/Acc |
+------------------------------------------------------------------+----------------+---------------+
|                  2 layers (256(p=25%),128(BN))                   | 2.3% / 99.21%  | 6.3% / 98.21% |
|             3 layers (256(p=25%),256(p=25%),128(BN))             | 2.7% / 99.12%  |  5.7% / 98.5% |
| 5 layers (256(p=25%),256(p=25%),512(p=50%),256(p=25%),128(BN)) | 3.3% / 99.04%  | 7.3% / 98.27% |
+------------------------------------------------------------------+----------------+---------------+
```

In [0]: