1. API (NeuralNetwork)

It's in *neural_network.py*. In it you can find 5 methods, *__init__(), getLayer(), backward(), updateParams* and *forward().*

In *__init__(),* the weights between different layers are initialized using *torch.distributions.normal(),* and the mean is 0, with standard deviation 1/sqrt(layer_size). The input size is 2xn, and output size is 1xn.

forward() method accepts input values, and after a series calculations, output a single predict value.

backward() method accepts target value as input, and calculate the error between its output and the target. Then through backpropagation, the error is transferred backward, and the value how error gets affected by weights *dE_dTheta* is calculated.

updateParams() method updates the weights between 2 adjacent layers of the network using the output of *backward()* method. It accepts eta as input to decide how aggressive the weights would be changed.

getLayer() gets the index of weight matrix as input and output the weight matrix.

2. API (MyImg2Num)

It's in file *my_img2num.py*. It has 2 public methods, *train()* and *forward([28x28 ByteTensor] img)*.

__init()__. It downloads the MNIST dataset into training and validating. It also sets parameters and the network structure using the above mentioned API *NeuralNetwork*. The parameters it sets are

max_epoch	50
input_size (input layer nodes number)	28*28
class_no (# of classes)	10
learning_rate	3.0
train_batch_size	60
validation_batch_size	1000
network structure	[28*28, 300, 60, 10]
loss function	mean squared error
non-linear function	sigmoid
shuffle before each epoch	True

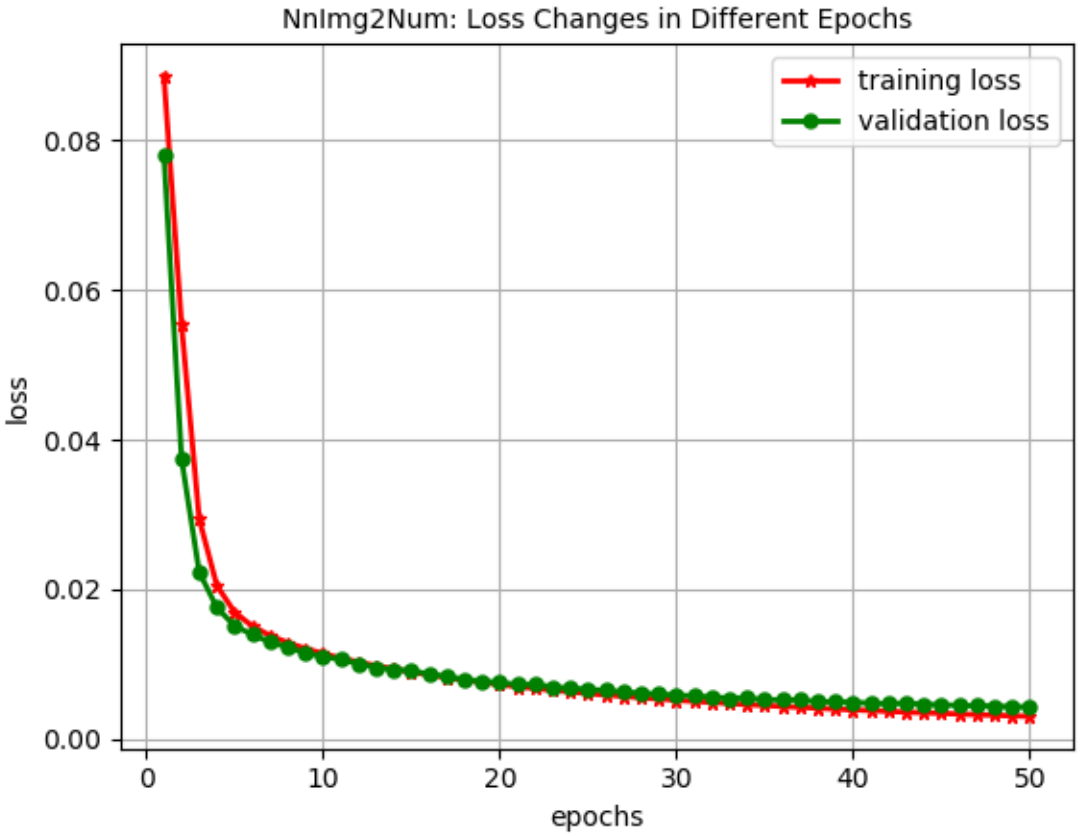
train(). It performs training and validation of the network. The process is, for each epoch, to perform *forward -> backward -> update* for each training batch of all training batches, and then perform validation for each validation batch. The whole process is repeated for *max_epoch* times. The whole process uses onehot labeling method to label the image and train the network.

forward(). It reads in one image of 28x28, converts it to 1D and output a predicted label for it.

3. API (NnImg2Num)

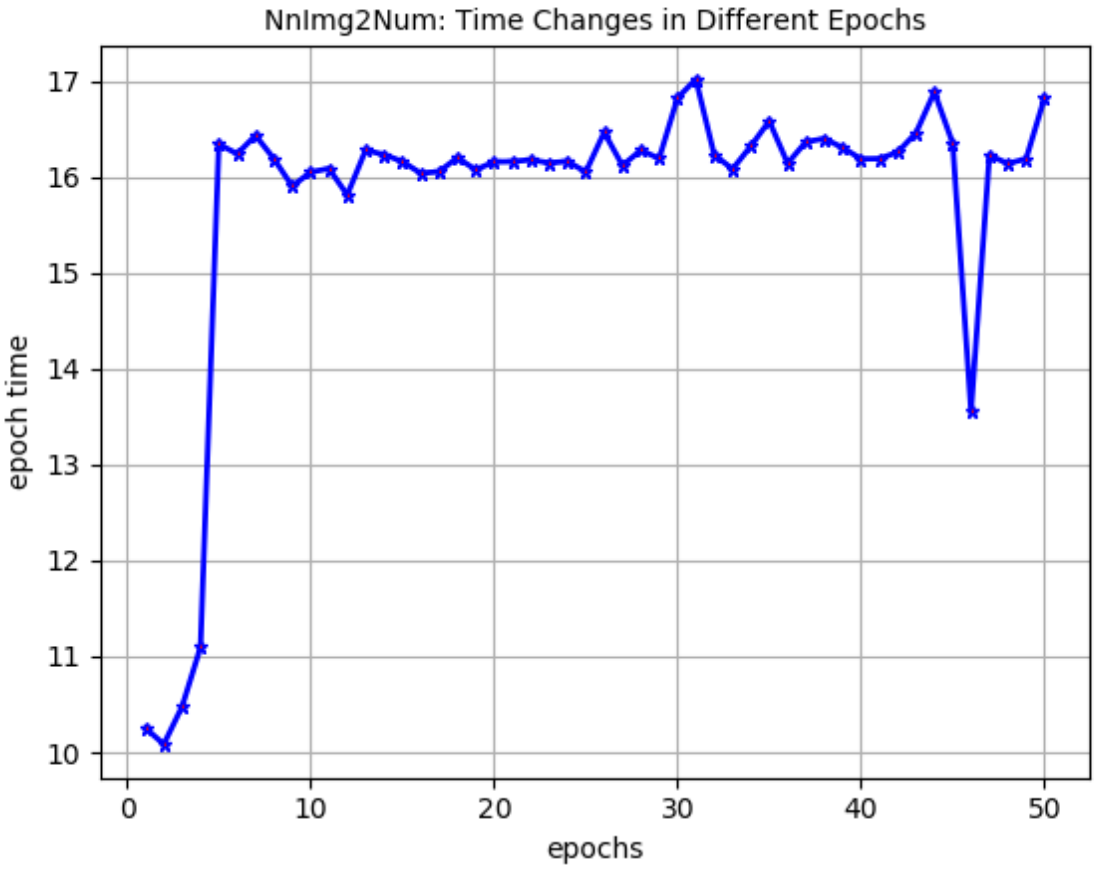
It's in *nn_img2num.py*. It performs the same function as API *MyImg2Num*, except that it uses the pyTorch nn packages. So the flow is the same as the previous API.

The training/validation loss vs epochs chart is



Because the way we calculate the training loss (average each batch loss after each forward-backward-update), the training loss is a little bit higher than validation loss, but eventually, the training loss is less than the validation loss. And at the end of *50th* epoch we have an accuracy of *97.27%.*

The time vs epochs chart is

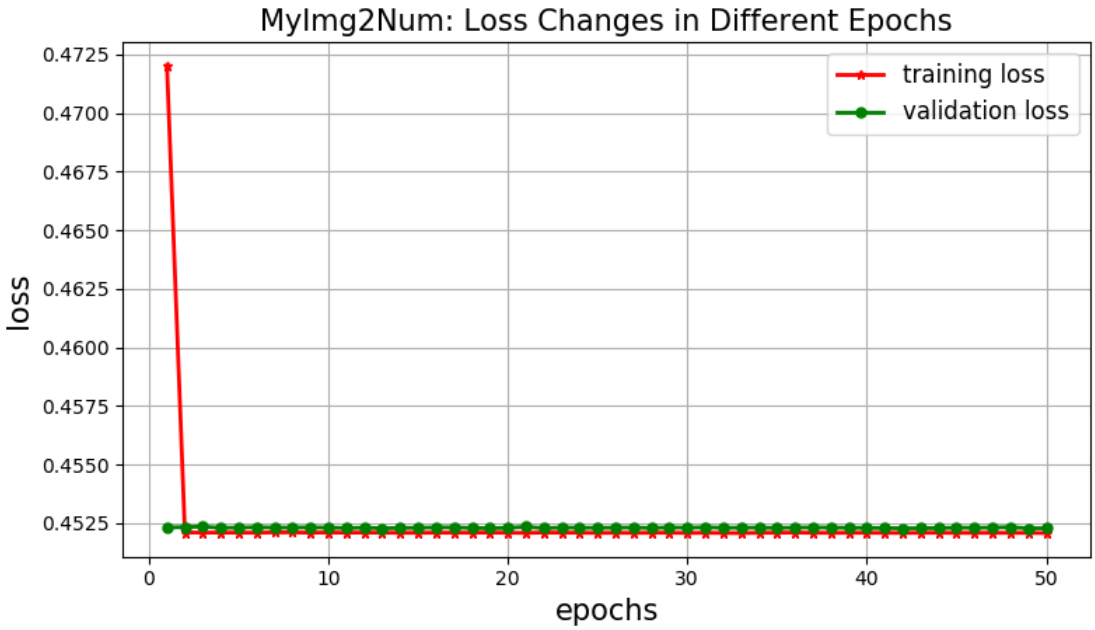


From the picture, we can see that the time taken for each epoch is rises after the first few epochs, and then stays in a small range (except for 1 fluctuation). The total time taken is *824.27s*.

4. test.py

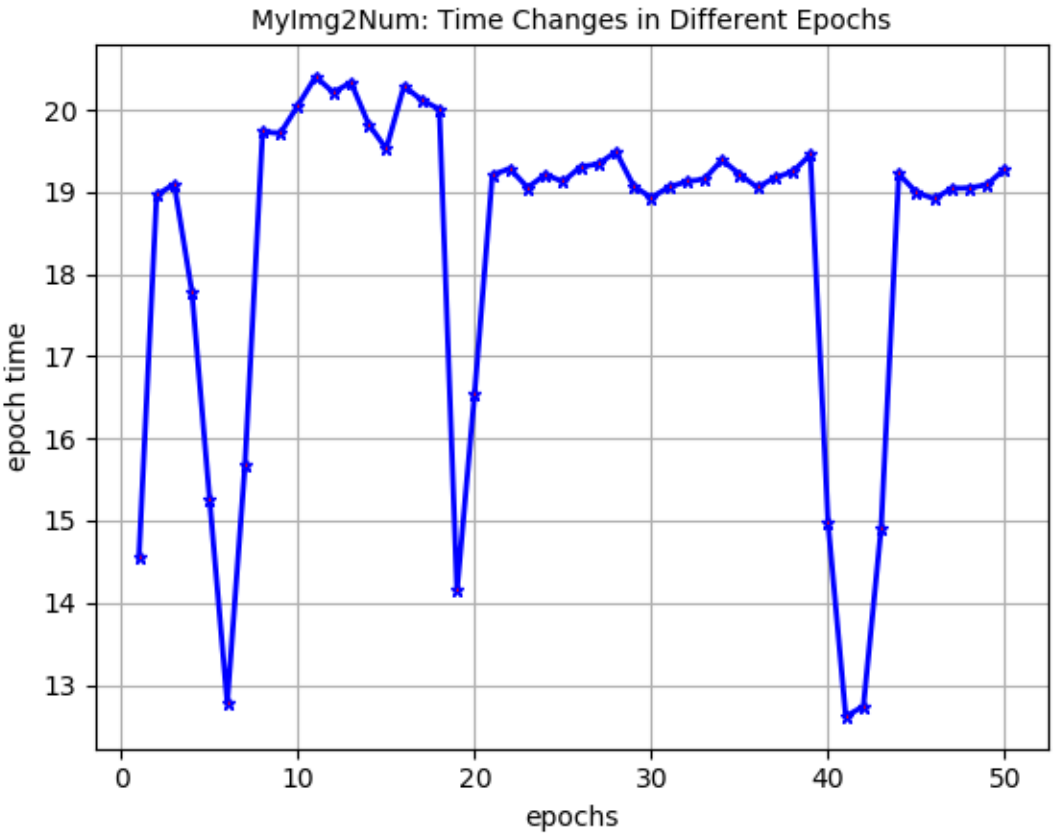
It contains code to run the training process using *MyImg2Num* API or *NnImg2Num* API, and output the total time to run 50 epochs.

For *MyImg2Num* API, the loss vs epoch plot is



The 2 loss values barely change during 50 epochs and stay high. The accuracy is around 10%. I checked that I implemented the right formulas, and changed different learning rates from 0.0000001 to 100.0, but still the accuracy couldn't improve much.

The time vs epoch plot is



It’s slower than API using torch.nn packages.