IMPLEMENTING SINGLE LAYER NEURAL NETWORK ON MNIST DATA

LINGALAN

Setup. X is the input vector size $d = 784$.

We have L neurons in the hidden layer.

Input to the i^{th} neuron in the hidden layer (1st layer) is given by

$$Z_i^1 = W_{ij}^1 X_j + b_i^1, \qquad i \in \{1, 2, \ldots, L\}.$$

(We are using Einstein summation convention.) The weights W^1 and biases b^1 we do not know yet.

The neurons activate according to the Rectified linear unit:

$$\sigma(c) \overset{\text{def}}{=} \max\{c, 0\}.$$

So, output from the neurons are

$$a_i^2 = \sigma(Z_i^1), \qquad i \in \{1, 2, \ldots, L\}.$$

Outer layer neurons are $K = 10$ in number (one for each digit). The input to each of the outer layer neurons (2^{nd} layer) is given by

$$Z_i^2 = W_{ij}^2 a_j^2 + b_i^2, \qquad i \in \{1, 2, \ldots, L\}.$$

The weights W^2 and biases b^2 we do not know yet.

The Z^2 values are fed to the soft-max function:

$$g : \mathbb{R}^L \to \mathbb{R}^L, \qquad (\, g(v)\,)_k = \frac{e^{v_k}}{\sum_j e^{v_j}}, \quad k \in \{1, 2, \ldots, K\}.$$

The digit corresponding to the input X should be declared as $\texttt{argmax}_k(g(Z^2))_k$.

Learning the parameters W^1, b^1, W^2, b^2. We would select those parameters that would minimize the KL distance between observed and theoretical densities. This amounts to maximizing

$$\sum_{p=1}^{N} \mathcal{L}^{(p)} \overset{\text{def}}{=} \sum_{p=1}^{N} \log(g(Z^2))_{y_p}$$

where the summation is over the training data, y_p is the class label of the p^{th} instance of the training data.

We would use the stochastic gradient ascent algorithm. For this purpose, we need to calculate the gradients of the above objective function. We get the following with everything evaluated with p^{th} training instance.

$$\frac{\partial \mathcal{L}^{(p)}}{\partial Z_j^2} = 1_{\{y_p = j\}} - (g(Z^2))_j$$

$$\frac{\partial \mathcal{L}^{(p)}}{\partial b_j^2} = \frac{\partial \mathcal{L}^{(p)}}{\partial Z_j^2}, \qquad \frac{\partial \mathcal{L}^{(p)}}{\partial W_{ij}^2} = \frac{\partial \mathcal{L}^{(p)}}{\partial Z_i^2}\, a_j^2$$

$$\frac{\partial \mathcal{L}^{(p)}}{\partial Z^1_j} = \sigma'(Z^1_j)\, W^2_{ij}\, \frac{\partial \mathcal{L}^{(p)}}{\partial Z^2_i}$$

$$\frac{\partial \mathcal{L}^{(p)}}{\partial b^1_j} = \frac{\partial \mathcal{L}^{(p)}}{\partial Z^1_j}, \qquad \frac{\partial \mathcal{L}^{(p)}}{\partial W^1_{ij}} = \frac{\partial \mathcal{L}^{(p)}}{\partial Z^1_i} X_j.$$

The above can be implemented using `numpy`'s functions `dot`, `outer`, `multiply`.

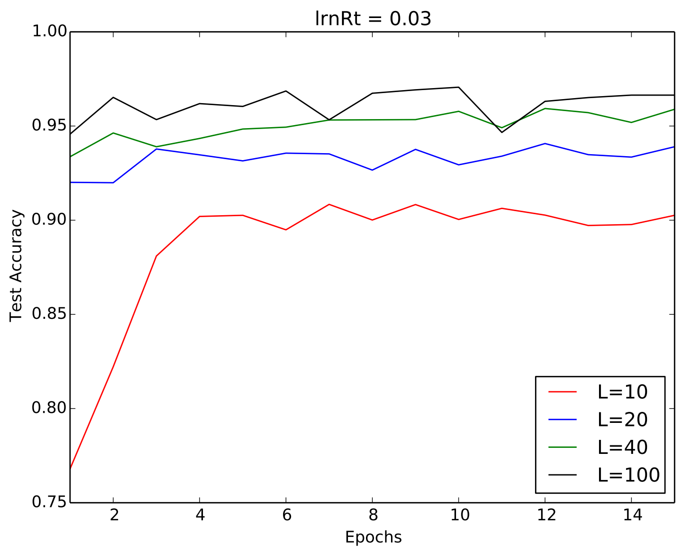
The update rule for W^2 is

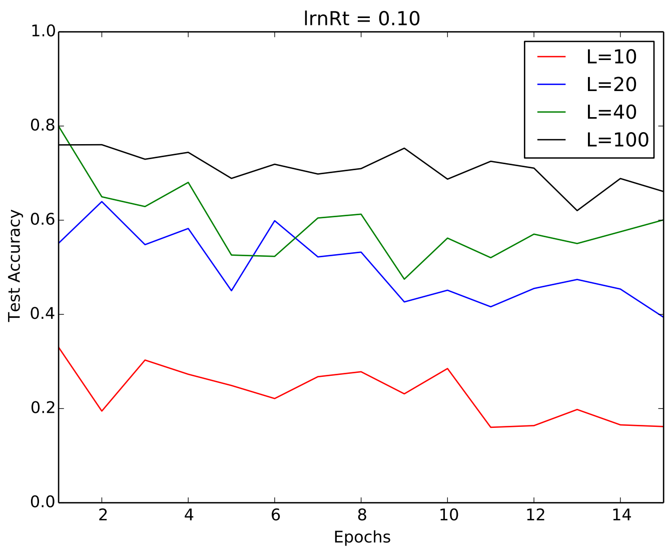
$$W^2_{ij} \leftarrow W^2_{ij} + \texttt{lrnRt} \frac{\partial \mathcal{L}^{(p)}}{\partial W^2_{ij}}$$

where `lrnRt` is the learning rate. Similar are the equations for W^1, b^1, b^2.

The code is written in the file `MNISTslnn.py`

The following two figures show that accuracy improves with the number of neurons in the hidden layer. Smaller the learning rate better the accuracy.


lrnRt = 0.03
Test Accuracy
Epochs
L=10
L=20
L=40
L=100


lrnRt = 0.10
Test Accuracy
Epochs
L=10
L=20
L=40
L=100