CSE 398/498: Deep Learning

Programming Assignment 2

Due on Thursday October 12, by midnight

In this assignment, you will learn the details of constructing a multiple layer neural network and using the back propagation algorithm to train the network for specific tasks. You will also experiment with various optimization techniques to speed up training and improve accuracy. You will start with the example code "network.py" posted on course site. The datasets to use are the MNIST and CIFAR 10. Your tasks are listed as follows.

Task 1. Multi-layer Neural Network on MNIST dataset

1.1. Answer the question: what kind of unit is used and what is the activation function implemented in "network.py"?

1.2. Read "mnist_loader.py", and use it to write a wrapper test program to load the training data, validation data, and test data from the MNIST dataset (mnist.pkl.gz). Use the data to test the network with one input layer (784 dimensional), one hidden layer (30 neurons), and one output layer (10 neurons). Set the initial hyper parameters to: number of epochs for training = 30, mini-batch size=10, and learning rate=3.0. Print the progress and accuracy report (as displayed in the terminal/console window) in a file.

1.3. Change each hyper parameter individually (number of hidden layers, number of units in each hidden layer, number of epochs for training, mini-batch size, and learning rate) and print the progress and accuracy report into a file each time you change a hyper parameter. Submit at least 3 such files with comments on what you observe.
   - Try a baseline with no hidden layers!
   - Try setting the learning rate too small or too big and see what happens

1.4. Use the validation data to tune the hyper parameters. That is, try different hyper parameter settings and test them on the validation data. Report what setting gives you the best performance on the validation data. Then, apply that parameter setting on the test data, and report accuracy on the test data.

1.5. Implement the Softmax group and Cross-entropy cost function for the last layer (i.e. output) layer. Note that both the activations of the last layer and cost derivatives need to be calculated according to Softmax. Run the program using MNIST dataset and the same hyper parameters as specified in Task 1.2. Print the Softmax implementation's progress and accuracy report in a file. Compare its performance with that from Task 1.2.
   - You can have a separate network module to implement the Softmax, or you can modify the existing network.py to add an additional argument to specify which cost function you want to use (QuadraticCost or CrossEntropyCost)

1.6. Implement the ReLu (rectified linear units) for all hidden layers. Use Softmax for the last output layer and cross entropy as the cost function. Run the program using MNIST dataset and print the progress and accuracy report in a file.

1.7. (Extra Credits) Do research on "Leaky" ReLu. Write a paragraph to explain what is Leaky ReLu, what problem of ReLu that Leaky ReLu attempts to address. Implement Leaky ReLu. Write a paragraph (or attach output files) to compare the performance by Leaky ReLu with that by ReLu.

Task 2. Optimization to make learning faster and better

2.1. Modify "network.py" to implement a different initialization scheme for weights. More specifically, initialize the weights to a hidden unit to be proportional to the inverse of sqrt(fan-in). Compare the results with the results by the original implementation (i.e. random initialization of weights) and report what you observe.

2.2. Pre-process the training data from MNIST, to (1) shift the inputs so that the inputs have zero mean over the whole training set, (2) scale the inputs so that the inputs have unit variance over the whole training set. Choose a favorite configuration for your network, and re-train the network using the pre-processed training data. Then, pre-process the test data in the same way that you pre-processed the training data, and run the network on the pre-processed test data. Print the progress and accuracy report in a file. Compare this result with that by using the original training and test datasets without pre-processing.

2.3. Implement the standard momentum method for updating weights. Implement the Nesterov method for updating weights. Compare these three sets of results: (1) using standard momentum method, (2) using Nesterov method, (3) baseline model using directly the gradients to change weights.

2.4. (Extra credits) Implement "Drop Out" to regularize the weights in the fully connected layers. Dropout means that a certain percentage (e.g. 50%) of the hidden units are randomly removed (or not used) for each training example. Compare the results with a baseline model without dropout.

2.5. (Extra credits) Implement "Model averaging", which means that you train multiple models/networks, and then for each test data sample, make the final prediction based on the consensus (e.g. majority voting) of the multiple models. Report whether any improvement is observed by using model averaging.

2.6. (Extra credits) Implement a scheme to adaptively set the learning rate.

- Try to set a faster learning rate at the beginning of the training, and then tune down the learning rate as training progresses
- Try to implement the scheme described in Hinton's lecture notes. That is, set the learning rate for a weight as a global learning rate (set by hand and fixed) multiplied by an appropriate local gain that is determined empirically for this weight. Start with a local gain of 1 for every weight. Then, increase the local gain for a weight if the gradient for that weight does not change sign; use small additive increases. Decrease the local gain for a weight if the gradient for that weight changes sign; use multiplicative decreases.

Task 3. Training and Testing using the CIFAR 10 dataset

3.1. Download the CIFAR 10 dataset (https://www.cs.toronto.edu/~kriz/cifar.html).
3.2. Choose a favorite configuration for your multi-layer neural network, and adapt it to make it work on the CIFAR 10 dataset. Print the progress and accuracy report in a file.
3.3. Explain how you handled the color channels since CIFAR 10 images are RGB color images.

Task 4. (Extra credits) **Backpropagation with weight constraints and implementation of a convolutional neural network**

For implementation details, please read this page:
http://cs231n.github.io/convolutional-networks/

4.1. Implement a convolutional layer, a max pooling layer.
4.2. Construct a convolutional neural network that is similar to the AlexNet.
4.3. Test your CNN network on the MNIST and CIFAR 10 datasets.
4.4. Write several paragraphs to explain your implementation and the performance of your network on the two datasets.

What to hand in

1. Your source code implementing all the required tasks.
2. A README file, to (1) answer questions, (2) summarize which scripts (and output files, etc.) complete which tasks, and (3) briefly explain what you did if you implemented any extra credit requirements.

Your completed work should be submitted via Course Site.