CS189: Introduction to Machine Learning

Homework 6

Due: 11:59 pm on Monday, April 21, 2014

Neural Networks for MNIST digit recognition

In this homework, you will implement neural networks for classifying handwritten digits, using raw pixels as features. You will use the MNIST dataset provided in Homework 1.

The file train_small.mat contains 7 different subsets, containing 100, 200, 500, 1000, 2000, 5000 and 10000 training points respectively. The full training set containing 60000 images is in train.mat. The test set is in test.mat.

The state-of-art error rate on this dataset using deep convolutional neural networks is roughly 0.5%. In this homework, you should, with appropriate parameter settings, get approximately (or better than) 8% error using a network with no hidden layers, and 5.5% using a network with two hidden layers.

You should expect 2 hours of training time per network. Please start this assignment early!

Single layer neural network

A single layer neural network is shown in Fig 1. This network only has an input layer and an output layer. In order to do digit classification as multiclass classification, the number of output units is set to be the number of classes.

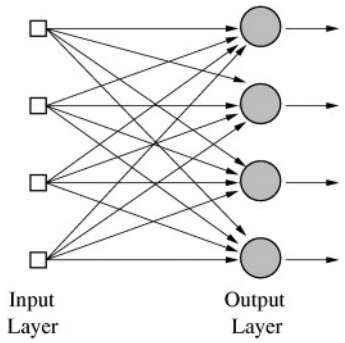

Figure 1: Single layer neural network.

The parameters of this network are the weight matrix $\mathbf{W}$, an n_{in}-by-n_{out} matrix, and the bias $\mathbf{b}$, an n_{out}-dimensional vector.

Suppose the input is an image x (a vector of dimension n_{in}) and output is y (a vector of dimension $n_{out} = 10$, with a single 1 at the position of the true class, and 0's elsewhere). You should use the sigmoid function σ as the nonlinear activation function, so that the output of the k^{th} neuron in the output layer is

$$y_k = \sigma\left(\sum_j W_{jk} x_j + b_k\right)$$

Now suppose t is an n_{out}-dimensional vector that represents the ground truth label (using the same 1-of-n_{out} encoding as y). You must consider the following two loss functions for y:

1. Mean squared error
$$J = \frac{1}{2}\sum_{k=1}^{n_{out}} (t_k - y_k)^2$$

2. Cross-entropy error
$$J = -\sum_{k=1}^{n_{out}} \left[t_k \ln y_k + (1 - t_k)\ln(1 - y_k)\right]$$

(i) Derive the stochastic gradient update for weight matrix $\mathbf{W}$ and bias $\mathbf{b}$ for the two loss functions.

(ii) Train this single layer network using stochastic gradient descent with minibatches of size 200. To do this, you must make many (hundreds) of passes, called *epochs*, over the training set. During each epoch, you must randomly shuffle the training set and split it into minibatches (if there are N training examples, then you should have about $N/200$ minibatches). Then, for each minibatch, you must compute the gradient of the loss summed over that minibatch, and then perform a step according to the gradient.

You should use the full training set, and you should initialize the parameters with random values. Perform training and testing for the two loss functions given above, and report results for both.

You must implement the backpropagation algorithm to compute gradients.

There are a variety of parameters for you to tune: the learning rate (you can consider decay the learning rate as epoch goes), the number of epochs of training. Please report your results, running time, plot the total training error and classification accuracy on training set and test set as it evolves each epoch. If you find that evaluating error takes a long time, you may do so once every 10 epochs.

Multilayer feed forward neural network

You may have figured out the error rate of single layer neural net is not good enough. Now you will investigate the effects of 'deeper' networks. Fig 2 is an example of multilayer feed forward neural network with one hidden layer.

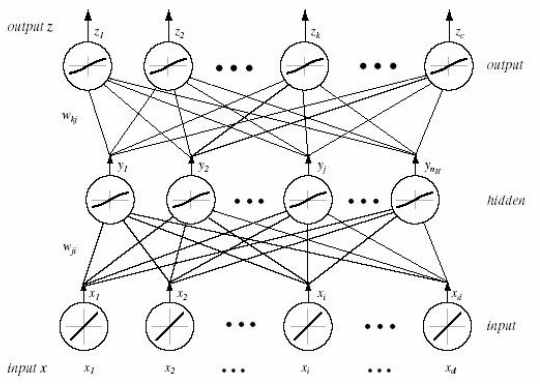

Figure 2: Example of a multiple layer neural network with one hidden layer.

In this section, you will implement a multilayer neural network with *two* hidden layers. The number of hidden units for two layers are set to $n^1_{hid} = 300$ and $n^2_{hid} = 100$. All the hidden units must use the tanh activation function, and the output units should use the sigmoid function. The parameters of this network are the following:

- Weights: $\mathbf{W^1_{hid}}$, a n_{in}-by-n^1_{hid} matrix; $\mathbf{W^2_{hid}}$, a n^1_{hid}-by-n^2_{hid} matrix; and $\mathbf{W_{out}}$, a n^2_{hid}-by-n_{out} matrix.
- Bias: $\mathbf{b^1_{hid}}$, a n^1_{hid}-dimensional vector of hidden biases; $\mathbf{b^2_{hid}}$, a n^2_{hid}-dimensional vector of hidden biases, $\mathbf{b_{out}}$, a n_{out}-dimensional vector of output biases.

(i) Derive the parameter update equations for the two loss functions mentioned in the previous section. (Hint: the derivative of tanh is $1 - \tanh^2$).

(ii) Train this multilayer neural network on full training data using stochastic gradient descent with mini-batches of size 200. Try both loss functions, and report the same information that you did for the previous section.

Tips

Numerical gradient checking

Backpropagation is quite tricky to implement correctly. We strongly encourage you to verify the gradients computed by your backpropagation code by comparing its output to gradients computed by finite differences. Let $f : \mathbb{R}^d \to \mathbb{R}$ be an arbitrary function

(for example, the loss of a neural network with d weights and biases), and let a be a d-dimensional vector. You can approximately evaluate the partial derivatives of f as:

$$\frac{\partial f}{\partial x_k}(a) \approx \frac{f(\dots, a_k + \epsilon, \dots) - f(\dots, a_k - \epsilon, \dots)}{2\epsilon}$$

where ϵ is a small constant. As ϵ approaches 0, this approximation theoretically becomes exact, but becomes prone to numerical precision issues. The choice $\epsilon = 10^{-5}$ usually works well.

This technique is extremely slow, so don't forget to remove it from your code after you've used it to verify your backpropagation implementation!

Checkpointing

Since training these neural networks takes lots of time, we encourage you to write your code so that it saves its progress every few epochs. This way, if your program somehow gets terminated, you can resume computation where it left off.

Convergence checking

With inappropriate step sizes, stochastic gradient descent on neural networks easily gets trapped in bad local minima. We encourage you to examine the magnitude of the gradient every epoch. Extremely small gradients indicate that gradient descent has converged, and if convergence happens at a point that gives poor validation set error, consider adjusting the step sizes so that this does not happen.

Submission

As deliverables for this homework, you need to submit your code, a README and a report explaining the features you implemented, the results you obtained, references to all external sources you used (code, articles, papers, books etc.) and anything else you might want to include (analysis of results, performance curves etc.). Submissions are through **bSpace**.

Reference

[1] Y. LeCun et.al. Gradient-based Learning Applied to Document Recognition.