CPSC 540 Assignment 5 (due October 29)

Neural Networks, Convergence Rates

Please put your name and student number on the assignment, staple the assignment together, and you will get 1 bonus point if the assignment is typed in LaTeX.

1 Visualizing a neural net for 1D regression

The file `nnet.m` contains a script to train a basic neural net. It is set up with a 1D example, i.e. where the neural net is used to learn a function mapping $\mathbb{R} \to \mathbb{R}$. When you run the script, you should be able to see training progress as the network begins to fit the data. However, in its current form it doesn't fit the data very well. Try to improve the performance of the method by changing the structure of the network (*nHidden* is a vector giving the number of hidden units in each layer) and the training procedure (e.g., change the sequence of step sizes, add momentum, or use *findMin* from the previous assignment). Hand in your plot after changing the code to have better performance, and list the changes you made.

2 Neural net tuning competition

In this problem we will investigate handwritten digit classification on the USPS data set. The inputs are 16 by 16 grayscale images of handwritten digits (0 through 9), and the goal is to predict the number given the image. If you run `nnet2.m` it will load this dataset and train a neural network with stochastic gradient descent, printing the test error as it goes. To handle the 10 classes, there are 10 output units (using a $\{-1, 1\}$ encoding of each of the ten labels) and the squared error is used during training. Your task in this question is to modify this training procedure/architecture to optimize performance. Report the best test error you are able to achieve on this dataset, and report the modifications you made to achieve this. (Your mark on this question will be a multiple of $(1 - \epsilon_i + \epsilon_{\min})$, where ϵ_i is the test error you achieved and $\epsilon_{\min}$ is the lowest test error achieved across the class, be prepared to submit your code to me if you think you achieved the best performance!).

Below are additional features you could try to incorporate into your neural network to improve performance (the options are approximately in order of increasing difficulty). You do not have to implement all of these, the modifications you make to try to improve performance are up to you and you can even try things that are not on this list. But, let's stick with neural networks models and only using one neural network (no ensembles)..

- Change the network structure (e.g., *nHidden* or the training procedure, as in the question above. Recall that *momentum* uses the update

$$w^{t+1} = w^t - \alpha_t \nabla f(w^t) + \beta_t (w^t - w^{t-1}),$$

  where α_t is the learning rate (step size) and β_t is the momentum strength. A common value of β_t is a constant 0.9.

- You could vectorize evaluating the loss function (e.g., try to express as much as possible in terms of matrix operations), to allow you to do more training iterations.
- Add ℓ_2 regularization of the weights to your loss function. For neural networks this is called *weight decay*. An alternate form of regularization that is sometimes used is *early stopping*, which is stopping training when the error on a validation set stops decreasing.
- Instead of using the squared error, use a softmax (multinomial logistic) layer at the end of the network so that the 10 outputs can be interpreted as probabilities of each class. Recall that the softmax function is

$$p(y_i) = \frac{\exp(z_i)}{\sum_{j=1}^{J} \exp(z_j)},$$

  and you can squared error with the negative log-likelihood of the true label under this loss, $-\log p(y_i)$.
- Instead of just having a bias variable at the beginning, make one of the hidden units in each layer a constant, so that each layer has a bias.
- Implement "dropout", in which hidden units are dropped out with probability p during training, and weights are multiplied by $1-p$ during testing. A common choice is $p = 0.5$.
- You can do 'fine-tuning' of the last layer. Fix the parameters of all the layers except the last one, and solve for the parameters of the last layer exactly as a convex optimization problem. E.g., treat the input to the last layer as the features and use techniques from earlier in the course (this is particularly fast if you use the squared error, since it has a closed-form solution).
- You can artificially create more training examples, by applying small transformations (translations, rotations, resizing, etc.) to the original images.
- Replace the first layer of the network with a 2D convolutional layer. You will need to reshape the USPS images back to their original 16 by 16 format. The Matlab `conv2` function implements 2D convolutions. Filters of size 5 by 5 are a common choice.

3 Convergence Rate of Coordinate Descent

Consider the optimization problem

$$\min_{x \in \mathbb{R}^d} f(x),$$

where f is twice-differentiable. We'll assume that f is *strongly-convex* and *coordinate-wise strongly-smooth*, so that

$$\mu I \preceq \nabla^2 f(x), \quad \nabla^2_{ii} f(x) \leq L,$$

for some $\mu > 0$ and some finite L, for all x and i. Consider a *coordinate descent* algorithm where we use the iteration

$$x^{t+1} = x^t - \frac{1}{L} \nabla_{i_t} f(x^t) e_{i_t},$$

where i_t is the coordinate we choose on iteration t and e_i is a vector of zeros but with a single 1 at location i (sometimes e_i are called *elementary vectors*). In this section we will use the proof technique from class to give a convergence rate for this algorithm based on different ways to choose the coordinate i_t.

3.1 Greedy Coordinate Descent

The *Gauss-Southwell* (or greedy) coordinate descent method chooses the coordinate with largest directional derivative, which is

$$i_t = \arg\max_i |\nabla_i f(x^t)|.$$

Show that under this choice, the algorithm satisfies

$$f(x^{t+1}) - f(x^*) \leq \left(1 - \frac{\mu}{Ld}\right)[f(x^t) - f(x^*)],$$

which implies a linear convergence rate if you apply this recursively.

(The implication of this is that, since L above is smaller than the L for the gradient method, that the coordinate descent method is faster than the gradient method if you can afford to do d iterations of coordinate descent for the cost of one gradient descent iteration.)

Hint: From strong-convexity you can use this bound we showed in class:

$$f(x^*) \geq f(x^t) - \frac{1}{2\mu}\|\nabla f(x^t)\|^2.$$

Also, since x^{t+1} and x^t differ only in the coordinate i_t you can use a coordinate-wise Taylor expansion around x^{t+1} as

$$f(x^{t+1}) = f(x^t) + \nabla_{i_t} f(x^t)(x^{t+1} - x^t)_{i_t} + \frac{1}{2}\nabla^2_{ii} f(z)(x^{t+1} - x^t)^2_{i_t}, \quad (1)$$

for some z and where $(x^{t+1} - x^t)_{i_t}$ is coordinate i_t of the difference. To connect these two equations, you can use that under this selection strategy $|\nabla_{i_t} f(x^t)| = \|\nabla f(x^t)\|_\infty$ and the equivalence between norms (question 1 of the previous assignment).

3.2 Randomized Coordinate Descent

The above algorithm/analysis is very old, but in 2010 Nesterov showed that if you pick a *random* i_t, so that

$$p(i_t = i) = 1/d,$$

for all i, then conditioned on x^t you still get the expected convergence rate

$$\mathbb{E}[f(x^{t+1})] - f(x^*) \leq \left(1 - \frac{\mu}{Ld}\right)[f(x^t) - f(x^*)].$$

Show that you obtain this convergence rate under this choice.

Hint: take the expectation with respect to i_t (remember that we assume x^t is given) after using the strong-smoothness assumption in (1).

3.3 Non-Uniform Randomized Coordinate Descent

Consider a variant where you have a different strong-smoothness constant with respect to each parameter,

$$\nabla^2_{ii} f(x) \leq L_i,$$

you change the step-size to $(1/L_i)$, and you sample proportional to the L_i,

$$p(i_t = i) = \frac{L_i}{\sum_{j=1}^d L_j}.$$

Show that this leads to a faster convergence rate.

4 Gradient and Proximal-Gradient

Recall the gradient method with a constant step-size α,

$$x^{t+1} = x^t - \alpha \nabla f(x^t).$$

A second common strategy used to analyze the convergence rate of gradient-style methods is to analyze the squared distance to the optimal solution, $\|x^{t+1} - x^*\|^2$. These arguments often begin by using the definition of x^{t+1} and then expanding as

$$\begin{aligned}
\|x^{t+1} - x^*\|^2 &= \|(x^t - \alpha \nabla f(x^t)) - x^*\|^2 \\
&= \|(x^t - x^*) - \alpha \nabla f(x^t)\|^2 \\
&= \|x^t - x^*\|^2 - 2(\alpha \nabla f(x^t))^T (x^t - x^*) + \|\alpha \nabla f(x^t)\|^2 \\
&= \|x^t - x^*\|^2 - 2\alpha(\nabla f(x^t))^T (x^t - x^*) + \alpha^2 \|\nabla f(x^t)\|^2
\end{aligned}$$

4.1 Faster convergence rate of gradient method

Assume that f is twice-differentiable, strongly-convex, and strongly-smooth, so that

$$\mu I \preceq \nabla^2 f(x) \preceq LI,$$

for all x.

Show that using a step-size of $\alpha_k = \frac{2}{L+\mu}$ gives a faster convergence than the one we showed in class,

$$\|x^{t+1} - x^*\|^2 \leq \left(\frac{L-\mu}{L+\mu}\right)^2 \|x^t - x^*\|^2.$$

(This is in fact the optimal constant step-size under these assumptions. And note that by strong-smoothness and $\nabla f(x^*) = 0$ this also implies that $f(x^{t+1}) - f(x^*) \leq \frac{L}{2}\left(\frac{L-\mu}{L+\mu}\right)^2 [f(x^t) - f(x^*)]$.)

Hint: to get this faster rate you need to use that $\nabla f(x^*) = 0$ and to use that strong-convexity and strong-smoothness imply

$$-(\nabla f(x) - \nabla f(y))^T (x - y) \leq -\frac{\mu L}{\mu + L}\|x - y\|^2 - \frac{1}{\mu + L}\|\nabla f(x) - \nabla f(y)\|^2,$$

which is a tighter inequality than you get if you use strong-convexity and strong-smoothness independently.

4.2 Convergence rate of proximal-gradient method

Consider the more general problem

$$\min_x f(x) + r(x),$$

where f satisfies the assumptions of the previous section and r is a general convex function (not necessarily differentiable, as in the ℓ_1-norm). Show that the proximal-gradient method from class

$$x^{t+1} = \text{prox}_{\alpha_k r(\cdot)}[x^t - \alpha_k \nabla f(x^t)],$$

obtains the same convergence rate as in the previous section,

$$\|x^{t+1} - x^*\|^2 \leq \left(\frac{L-\mu}{L+\mu}\right)^2 \|x^t - x^*\|^2.$$

Hint: you will need to use that proximal operators are *non-expansive* (they can't move x and y further apart),

$$\|\text{prox}[x] - \text{prox}[y]\| \leq \|x - y\|,$$

and that x^* is a *fixed-point* of the proximal-gradient iteration (applying the iteration to x^* just gives x^*),

$$x^* = \text{prox}_{\alpha_k r(\cdot)}[x^* - \alpha_k \nabla f(x^*)],$$

Note that, because of the presence of r, for this problem you cannot assume $\nabla f(x^*) = 0$.

4.3 Converting from Convergence Rates to Number of iterations

Instead of measuring the error on iteration t, we may be interested in the number of iterations required before $(f(x^t) - f(x^*)) \leq \epsilon$ for some small ϵ. This allows direct comparison of the runtimes of different iterative algorithms to each other and to non-iterative methods. Show the following:

1. If $f(x^t) - f(x^*) = O(1/t)$, then we require $t = \Omega(1/\epsilon)$ iterations.
2. If $f(x^t) - f(x^*) = O(1/t^2)$, then we require $t = \Omega(1/\sqrt{\epsilon})$ iterations.
3. If $f(x^t) - f(x^*) = O(\rho^t)$ for some $\rho < 1$, then we require $t = \Omega(\log(1/\epsilon))$ iterations.
   (Hint: $\log(\rho) < 0$ and $\log(1) = 0$.)

Hint: The argument roughly goes like this: If $f(x^t) - f(x^*) = O(1/\sqrt{t})$ (e.g., stochastic/sub-gradient method for weakly-convex functions), then this means $f(x^t) - f(x^*) \leq c/\sqrt{t}$ for some constant $c > 0$ (by definition of the $O(\cdot)$ notation) and in the optimization literature this means it holds for all $t > 1$. So if we want $f(x^t) - f(x^*) \leq \epsilon$ we need to have a t large enough that $c/\sqrt{t} \leq \epsilon$, or re-arranging that $t \geq c^2/\epsilon^2$ which means $t = \Omega(1/\epsilon^2)$ by the definition of $\Omega(\cdot)$.