

Perceptron learning

Sebastian Seung

9.641 Lecture 3: September 12, 2002

1 Learning as optimization

Suppose that we are given a training set of input-output pairs (x^μ, y^μ) as examples, $\mu = 1, \dots, m$. We would like to find a weight vector w so that the approximation

$$y^\mu \approx f(w \cdot x^\mu)$$

is as good as possible for all μ . The bias term has been omitted from these equations, but one can be added by increasing the dimensionality of the input vector by one, and setting the extra input to be equal to one for all μ .

One approach to training a perceptron is to minimize the squared error cost function

$$E(w) = \frac{1}{2} \sum_{\mu} [y^\mu - f(w \cdot x^\mu)]^2$$

with respect to the weight vector w . This cost function is a measure of the performance of the perceptron at producing the desired output y^μ when given the input x^μ . The best we can hope for is to find a w such that $E(w) = 0$, which would correspond to a perceptron that performs perfectly on the examples. More realistically, the optimal w will have small but nonzero $E(w)$, in which case the corresponding perceptron will perform well but not perfectly.

The formulation of learning as an optimization problem accords with the intuitive notion that the goal of learning is to improve performance.

2 Optimization by gradient following

What is the best way of optimizing the above cost function? In general, optimization is a difficult computational problem, and many books are devoted to the complexities of the subject. A useful reference is *Nonlinear Programming* by D. Bertsekas.

Perhaps the simplest method of minimizing a smooth cost function is that of gradient descent. Recall that the gradient of a multivariate function is the vector formed of all of its partial derivatives.

$$\nabla E = \left(\frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_N} \right)$$

Of all the possible directions in w -space, the gradient is that special direction along which E increases most rapidly. If E is a function of two variables, it can be visualized as a three-dimensional graph in which $E(w_1, w_2)$ is the height above the plane defined by the w_1 and w_2 axes. Alternatively, it can be depicted on a contour plot, which shows lines of constant E in the (w_1, w_2) plane.

If we are given a weight vector w , and would like to find another vector w' that has lower cost E , it makes sense to step in the direction of the negative of the gradient, because this is the direction along which E decreases most rapidly,

$$w' = w - \eta \nabla E(w)$$

Iteration of this formula is the gradient descent method of minimizing a function. If the learning rate η is chosen properly, this method will generally converge to a local minimum of $E(w)$, at which $\nabla E(w) = 0$. Of course η should be positive, so that the step goes in the proper direction. If η is chosen too large, then the step might be so large that $E(w')$ ends up larger than $E(w)$, which we don't want. We're only guaranteed that $E(w') < E(w)$ for sufficiently small η , provided that the gradient is nonzero. So it's important to find an intermediate value of η that is small enough to make the cost function decrease, but not so small that convergence is painfully slow. In practice, this takes some trial and error.

The above discussion was for minimization of a cost function by gradient descent. Sometimes the goal is to maximize an objective function, in which case, the gradient ascent method is used, which just involves changing a sign in the above formula. Both gradient ascent and descent are called gradient-following methods.

Gradient following has some deficiencies as a method of optimization. First of all, it can become trapped in a local optimum, rather than converging to the global optimum. One can hope that the local optimum is almost as good as the global optimum, or at least good enough for the purposes of the particular application, but this hope may not be realistic. If this is not the case, then one can run the gradient method many times from random initial conditions, and hope that one of these runs yields a good local optimum.

Even for finding a local optimum, gradient following does not converge as fast as other methods. The gradient generally deviates from the direction of a local optimum, even in its immediate vicinity. This deviation tends to be more severe, when the optimum is highly anisotropic. This can be quantified by the condition number of the Hessian. If this is a concern, there are more sophisticated methods, such as conjugate gradient, for speeding up convergence.

In spite of these deficiencies, gradient following is popular because of its simplicity.

3 Learning by gradient following

Let's apply gradient descent to the squared error cost function introduced above.

$$\begin{aligned} \Delta w &= -\eta \nabla E & (1) \\ &= \eta \sum_{\mu} f'(w \cdot x^{\mu}) [y^{\mu} - f(w \cdot x^{\mu})] x^{\mu} & (2) \end{aligned}$$

According to this formula, the gradient is the sum of many terms, one for each example. The contribution of the μ th example to the gradient is a vector that is proportional to x^μ . This vector is multiplied by the slope of the activation function. This is always positive, assuming that the activation function is monotone increasing. The vector is also multiplied by the difference between the desired and actual outputs of the perceptron, $y^\mu - f(w \cdot x^\mu)$. We'll refer to this difference as the error. If the actual output $f(w \cdot x^\mu)$ is less than the desired output y^μ , then the μ th term of the sum points in the same direction as x^μ . If the actual output $f(w \cdot x^\mu)$ is more than the desired output y^μ , then the μ th term of the sum points in the opposite direction as x^μ .

Note that the contribution to the gradient is small if the slope $f'(w \cdot x^\mu)$ of the activation function is small, because the output of the perceptron is insensitive to changes in this operating regime. For example, when f is the logistic function, the slope vanishes when the argument is either very positive or very negative.

Should we expect problems with local minima when performing gradient learning? There is a special case in which such problems are less severe, when there exists a w such that $E(w)$ is zero. But in many practical circumstances, there is no such w . In other words, there is no perceptron that will perform perfectly on all the examples. Doing well on one example may require doing poorly on another. This type of conflict can cause local minima problems. Nevertheless, gradient learning tends to work very well for perceptrons. As we will see later, it can have problems with more complicated networks.

4 Online learning

According to the formula given above, the gradient is computed by presenting all examples to the perceptron. Only after all examples are presented is w updated. This is known as batch learning, as the whole batch of examples is used before a single update is made.

An alternative is online learning, in which the update

$$\Delta w = \eta[y - f(w \cdot x)]f'(w \cdot x)x$$

is made after presentation of each example. Suppose that this update is made repeatedly, each time choosing an example randomly from the training set. (In practice, people often cycle through the training set in a fixed order, instead of going through it randomly). Then the update is a random variable. On average, the update points in the direction of $-\nabla E$. But any particular update might point in a very different direction. This is a special case of a general method known as stochastic gradient descent.

Intuitively, if the learning rate η is small, a significant change in w occurs only after many steps. This change will tend to be in the direction of the gradient, because it averages over so many steps. So it should be possible for stochastic gradient descent to converge to the same result as gradient descent.

If the learning rate η is held constant, then stochastic gradient descent may approach the neighborhood of a local optimum, but then the noise inherent in the method will prevent convergence. For true convergence, it is important to decrease the learning rate η with time. A popular choice is $\eta_t \propto 1/t$. Then as the local optimum is approached,

the learning rate η will decrease, causing noise to be suppressed by effectively averaging over many steps. Intuitively, the more knowledge the learner has acquired, the less impression new examples should make on the learner.

Batch learning requires that all examples be held explicitly in memory. In contrast, an online learner uses each example and then discards it. The only memory of past examples is implicit, in the weight vector w . For this reason, online learning is generally regarded as more biologically plausible.

5 Example: handwritten digit recognition

We can apply online gradient descent to the application of handwritten digit recognition. This is a

6 Stochastic approximation theory

A formal analysis of the properties of online gradient descent is provided by stochastic approximation theory. This is a complicated subject, so I'll just try to give a feel for the results.

It's convenient to rewrite the online update as

$$\Delta w = \eta \nabla_w e(w, x, y)$$

where

$$e(w, x, y) = \frac{1}{2} [y - f(w \cdot x)]^2$$

Furthermore, let's slightly modify the definition of the cost function to be

$$\bar{e}(w) = \langle e(w, x, y) \rangle = \frac{1}{m} \sum_{\mu=1}^m e(w, x^\mu, y^\mu)$$

The notation $\langle \rangle$ denotes an average over the randomly drawn example. Therefore the sum is normalized by the number of examples.

$$\nabla \bar{e}(w) = \langle \nabla_w e(w, x, y) \rangle = \frac{1}{m} \sum_{\mu=1}^m \nabla_w e(w, x, y)$$

Suppose that each trial is drawn at random from x

This is an example of stochastic gradient descent on the function $\bar{e}(w)$.

$$w_{t+1} = w_t + \eta_t \nabla e(w, z)$$

Suppose further that the learning rate satisfies the conditions

$$\sum_{t=0}^{\infty} \eta_t = \infty \quad \sum_{t=0}^{\infty} \eta_t^2 < \infty$$

Then every limit point of the sequence w_t is a stationary point of

$$E(w) = \langle e(w, z) \rangle$$

7 Perceptron learning rule

For convenience, the examples will be assumed to be labeled so that y takes on the values ± 1 , instead of 0, 1. Suppose that f is the step function, so that the output of the perceptron is binary. Then the cost function is not differentiable, so the above gradient following method cannot be implemented.

In this case, the classical perceptron learning rule can be applied. Each example is presented to the perceptron. If the perceptron gives the correct output, no change in w is made. If the perceptron makes an error, then the update $w = w + yx$ is made.

You can regard this as stochastic gradient descent on the cost function

$$\langle [-yw \cdot x]^+ \rangle_x$$

with a fixed learning rate $\eta = 1$. The previous results of stochastic approximation theory are not applicable here, since the learning rate is fixed. To show that the learning rule converges, we'll utilize another method.

8 Perceptron convergence theorem.

To prove convergence, we will assume that there exists a perceptron w^* with zero error on the examples. In other words, there exists a separating hyperplane. It turns out that this assumption is critical. If it is violated, then the perceptron learning rule may not converge.

The basic strategy is to prove that the number of errors N_{err} made by the perceptron learning rule remains finite. This implies there must be a last error, after which no updates to the weight vector are made. To do this, we'll show that the overlap $w \cdot w^*$ increases faster than $|w|$ as a function N_{err} . This yields a contradiction unless N_{err} remains finite.

For the sake of simplicity, we'll consider the special case that $y = 1$ for all examples. The general case is equivalent, if we make the reflection $x \rightarrow -x$ for all negative examples. Suppose further that all examples lie inside a ball of radius R , $|x| \leq R$. Without loss of generality, we set $|w^*| = 1$. Now define the margin M by

$$M = \min_{\mu} w^* \cdot x^{\mu}$$

Every time there is an error, the inner product $w \cdot w^*$ increases by at least

$$\Delta(w \cdot w^*) = x \cdot w^* \geq M$$

This means that $w \cdot w^* \geq N_{err}M$.

The change in the magnitude of w can be upper bounded by

$$\Delta|w|^2 = |w + x|^2 - |w|^2 = |x|^2 + 2w \cdot x \tag{3}$$

$$\leq |x|^2 \leq R^2 \tag{4}$$

Therefore $|w|^2 \leq N_{err}R^2$.

These two inequalities,

$$w \cdot w^* \geq N_{err} M \quad (5)$$

$$|w| \leq \sqrt{N_{err}} R \quad (6)$$

imply that

$$\cos \phi = \frac{w \cdot w^*}{|w| |w^*|} \leq \sqrt{N_{err}} \frac{M}{R}$$

where ϕ is the angle between w and w^* . Because $\cos \phi$ is upper bounded by 1, we find that

$$N_{err} \leq \frac{R^2}{M^2}$$

Can you construct a simple case where the perceptron learning rule does not converge?