COS 424: Interacting with Data

Lecturer: Léon Bottou — Lecture #9
Scribe: Rob Harrison — 3.11.2010

1 The Story

In slide 4, we motivate the discussion of neural networks by looking to the work by Norber Wiener in 1948. At the time, communications technologies and signal processing had become very mature while computing was still in a very nascent stage. Wiener drew a connection between the feedback loops that were seen in signal processing to the human brain, social processes, and information processes in general. Wiener's work motivated further study on what a computer should be.

In slides 5-6, we look at the two options for moving forward with a model for computation: the biological and the mathematical. The biological paradigm had some obvious engineering advantages, but the mathematical model offered far more guidance on how to move forward. Despite the obvious advantages of the mathematical approach, efforts to achieve computation by artificial neurons were still attempted. Slide 7 depicts an artificial neuron as a linear threshold unit (LTU). These early efforts have an obvious connection to neural networks by modeling a function with a vector x of inputs, weights w, and a threshold. In slide 8, we are presented with the NeoCognitron which was a circuit of LTUs that took advantage of symmetries and invariance of translation (throughout the field of vision) to do complicated tasks, but the question arose of how to effectively train such deep circuits.

Slide 9 finally kills the idea of using deep circuits of linear threshold elements to perform general computation. Problems easily solved by a general purpose computer required exceedingly deep circuits of LTUs and training these deep circuits seemed hopeless. Training in a discrete space resulted in combinatorial explosion, but this wasn't obvious at the time because complexity theory was also very immature. Although one can draw a network of LTUs that can represent anything, the difficulty is the combinatorial search involved in finding a small network of LTUs that achieves the desired task. This difficulty is also present (and even greater) when one searches a compact set of logical rules for the desired task. However, if we replace the hard threshold of LTUs with a sigmoid function, we now have a soft threshold by way of a continuous approximation. Such an approximation does not completely solve the problem, but it is sufficient for our purposes and allows us to leverage the gradient to perform any optimization we like.

2 Mechanics of Multi-Layer Networks

Slide 12 begins the discussion of modern multilayer networks. Consider a single generic brick akin to a lego. The brick is a modular version of a single layer neural network: a function $f_w(x)$ that takes a vector x of inputs, a parameter vector w of weights and produces output vector y. We can calculate some loss function based on the output of the module. We can combine these modules in any fashion we like. Slide 12 depicts a simple two layer network.

In some representations of neural networks, the non-linearity represented by the sigmoid (or other functions) can be called an activation function. Some texts will also combine the application of this activation function with the linear combination of the weights as a single layer. Note in our abstraction, we separate the application of the non-linearity from the linear combination of the weights because you do not always want both at the same time. This is largely a matter of convention and back-propagation will still work either way.

3 Back Propagation

After composing arbitrary sequences of bricks, we can use back-propagation to train the weights of the network. In order to do back-propagation, we need bricks for which we can calculate the derivative of the loss with respect to the parameters ($\frac{\partial L}{\partial w}$) and the derivative of the loss with respect to the inputs ($\frac{\partial L}{\partial x}$). We conduct back propagation in two passes, forward and backward. Slide 13 demonstrates the back-propagation algorithm in a two-layer network while slide 14 lists several different functions we could use in our generic brick as well as the equations for the forward and backward passes.

In the forward pass, we send the inputs through each brick in the network and send the outputs of each preceding brick as the input to the succeeding brick. On the final output, we calculate some resulting loss. In the backward pass, we compute the gradient of an individual brick and compute the stochastic gradient update to w at each brick. Because we ensured that each brick was differentiable, we know that this method will succeed.

In the stochastic update relation, γt is the gain or learning rate in our optimization (covered on Slides 21 and 27 of Optimization lecture). Too large a gain can cause divergence. See Figure 1 below for more explanation. In this example below, η is the equivalent of gamma.

4 Applications

Slide 15 presents several arbitrary combiations of modules demonstrating the flexibility of this abstraction for different applications. Slide 16 introduces convolutional networks (CNN) which are merely compositions of several modules that share a single matrix of weights W such that translational invariance is preserved. Slides 17-22 demonstrate the effectiveness of CNN applied to image processing. It is worth nothing that the code for facial recognition is 95% the same as the code for character recognition.

5 Optimization in Multi-Layer Networks: Challenges and Tricks

Slides 24-25 point out that the reason neural networks aren't more prevalent is because optimization can be difficult and may take a long time. In the graph on slide 25, one sees that there are several regions that are troublesome (listed next to the graph) and this graph is replicated symmetrically in each quadrant. The point is that if you choose your initial weights unwisely, your model may not converge to a solution.

On slide 26, we point out that if you pick small initial weights, then the linear portion of all the sigmoid curves will be exercised. As training progresses the weights will increase and the non-linearities of the function will slowly become more evident. In training such networks, one should continue to monitor both the training error and the validation error. When the validation error curve stops improving, then you should cease training.

A LITTLE THEORY

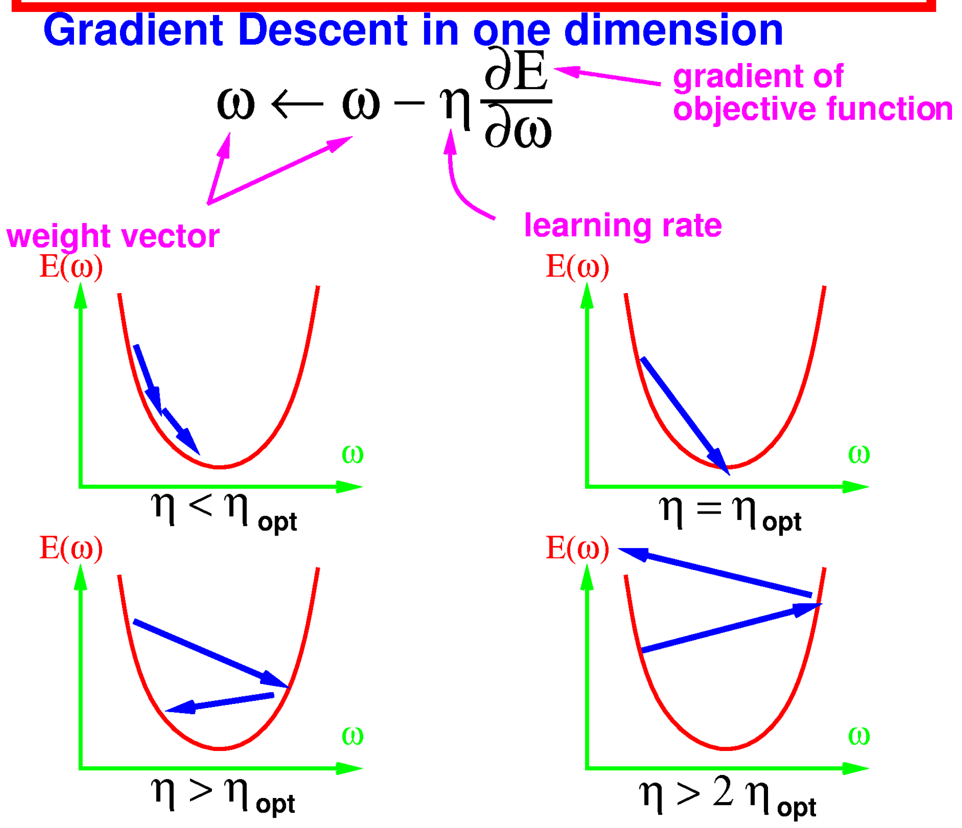
Gradient Descent in one dimension
$\omega \leftarrow \omega - \eta \frac{\partial E}{\partial \omega}$
gradient of objective function
weight vector
learning rate
$E(\omega)$
ω
$\eta < \eta_{opt}$
$\eta = \eta_{opt}$
$\eta > \eta_{opt}$
$\eta > 2\,\eta_{opt}$

Figure 1: Effects of learning rate or gain on convergence.