CS 5600/6600

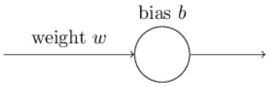
Lecture 4: Why NNs are Hard to Train and Tips on Training Them: Part 1

Vladimir Kulyukin
Department of Computer Science
Utah State University

Outline

Simple Learning Neuron (Version 1)

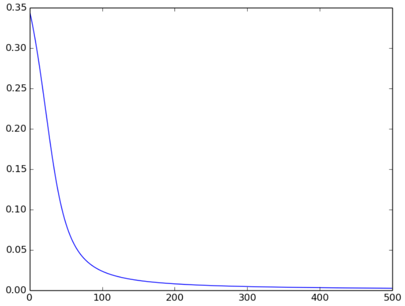
Let's create a simple neuron by setting $w - 1.28$, $b = -0.98$, $\eta = 0.15$. Our job is to train this neuron to take an input of 1 and convert to an input of 0. That simple! Let's use the quadratic cost function (MSE).



$\sigma(w \cdot x + b) = \sigma(-1.28 \cdot 1 - 0.98) \approx 0.09$ so this neuron must do some learning.

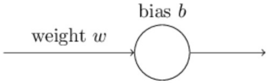
Simple Learning Neuron (Version 1): Epoch vs. Cost

Let's train the neuron defined on the previous slide for 500 epochs (each epoch consists of just one training input of 1) and graph the costs. The graph shows that the neurons learns rapidly a weight and a bias that drive down the cost to 0.



Simple Learning Neuron (Version 2)

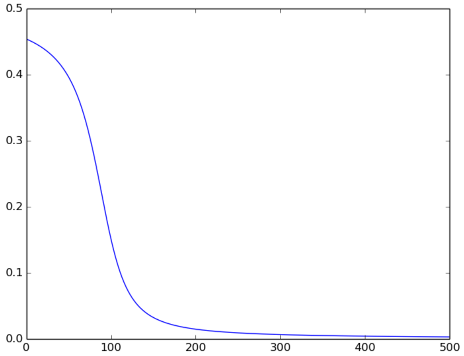
Let's create another version of the simple neuron by setting $w = b = 2.0$, and $\eta = 0.15$. We'll use the quadratic cost function.



$\sigma(w \cdot x + b) = \sigma(2.0 \cdot 1 + 2.0) \approx 0.98$ so this neuron must do even more learning than version 1.

Simple Learning Neuron (Version 2)

Let's train the neuron defined on the previous slide for 500 epochs and graph the costs. The graph shows that the neurons learns more slowly than neuron 1.



Why Does Learning Slow Down?

The simple neuron has more difficulty learning when it is more off the mark then when it is closer to the mark.

Let's try to understand why. The neuron learns by changing the weight and bias at a rate determined by two partial derivatives of the cost function $\partial C/\partial w$ and $\partial C/\partial b$.

Why is Learning Slow in the Simple Neuron?

Let's compute the partial derivatives. The cost function is

$$C = \frac{(y-a)^2}{2},$$

where a is the neuron's output when the training input $x = 1$ and $y = 0$ is the target output. Then

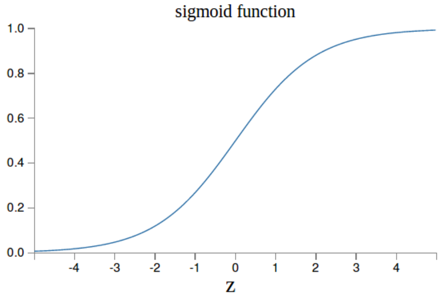
$$\frac{\partial C}{\partial w} = (a-y)\sigma'(z)x = a\sigma'(z)$$

$$\frac{\partial C}{\partial b} = (a-y)\sigma'(z) = a\sigma'(z),$$

because $x = 1$ and $y = 0$.

Why is Learning Slow in the Simple Neuron?

Recall the shape of the sigmoid function. When the neuron's output is close to 1 or 0, the curve gets flat, which makes $\sigma'(z)$ is very small. Therefore, $\partial C/\partial w$ and $\partial C/\partial b$ become very small.



Learning Slowdown

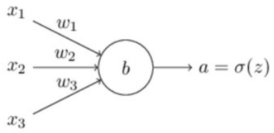
The learning slowdown is a problem for all neural networks.

A typical solution to address this problem is to replace the quadratic cost with a different cost function such as cross-entropy and softmax.

Outline

Cross-Entropy Formula

Suppose that we have a neuron with several inputs $x_1, x_2, ..., x_n$ and weights $w_1, w_2, ..., w_n$, a bias b, and output $a = \sigma(z)$, where $z = \sum_j w_j x_j + b$.



$$C = -\frac{1}{n} \sum_x [y ln(a) + (1 - y) ln(1 - a)],$$

where n is the total number of training data items (i.e., examples).

Cross-Entropy Formula

$$C = -\frac{1}{n}\sum_x [y ln(a) + (1-y) ln(1-a)],$$

Properties of Cross-Entropy:

1. $C > 0$;
2. $C \approx 0$ when $y = 0$ and $a \approx 0$ or $y = 1$ and $a \approx 1$.

In other words, the cross-entropy is positive and tends to 0 as the neuron gets closer to computing the desired output.

What's the big deal? The quadratic cost function also satisfies these properties. The big deal is that cross-entropy avoids the learning slowdown.

How Cross-Entropy Addresses Learning Slowdown

Recall that $a = \sigma(z)$. Then

$$\frac{\partial C}{\partial w_j} = -\frac{1}{n}\sum_x \left(\frac{y}{\sigma(z)} - \frac{(1-y)}{(1-\sigma(z))}\right)\frac{\partial \sigma(z)}{\partial w_j} =$$

$$-\frac{1}{n}\sum_x \left(\frac{y}{\sigma(z)} - \frac{(1-y)}{(1-\sigma(z))}\right)\sigma'(z)x_j =$$

$$\frac{1}{n}\sum_x \frac{\sigma'(z)x_j}{\sigma(z)(1-\sigma(z))}(\sigma(z)-y) =$$

$$\frac{1}{n}\sum_x x_j(\sigma(z)-y).$$

How Cross-Entropy Addresses Learning Slowdown

$$\frac{\partial C}{\partial w_j} = \frac{1}{n} \sum_x x_j(\sigma(z) - y).$$

This equation shows that the rate at which the weight learns is controlled by $\sigma(z) - y$, i.e., the error in the output. The larger the error, the faster the neuron learns.

A really cool thing is that when we use cross-entropy the term $\sigma'(z)$ cancels out. Thus, we are no longer dependent on it.

How Cross-Entropy Addresses Learning Slowdown

We can similarly compute the derivative of C with respect to b:

$$\frac{\partial C}{\partial b} = \frac{1}{n} \sum_x (\sigma(z) - y).$$

Again, when we use cross-entropy the term $\sigma'(z)$ cancels out.

Generalizing Cross-Entropy to Multilayer Networks

Let $y = y_1, y_2, \ldots$ be the target output values at the output neurons of a multilayer network. Let $a = a_1^L, a_2^L, \ldots$ be the actual output values of the same multilayer network. Then the cross-entropy C is defined as

$$C = -\frac{1}{n} \sum_x \sum_j [y_j ln(a_j^L) + (1 - y_j) ln(1 - a_j^L)].$$

This is the same as the previous formula except for the second embedded sum that sums over all outputs a_j^L.

Is Cross-Entropy Better than MSE?

Cross-entropy is almost always a better choice provided the output neurons are sigmoid.

Why? Since the initial weights are initialized randomly, it may happen that these initial choices result in the network that computes a really wrong output for some output. If this network uses cross-entropy, it will likely learn faster than if it uses MSE.

Outline

Another Approach to Learning Slowdown

The idea of softmax is to define a new type of output layer for neural networks.

Let the weighted inputs be $z_j^L = \sum_k w_{jk}^L a_k^{L-1} + b_j^L$.

However, instead of applying the sigmoid function σ to each z_j^L, then we apply the so-called softmax function to z_j^L, which is defined as

$$a_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}},$$

where in the denominator we sum over all the output neurons.

Softmax

As z_i^L increases, a_i^L increases and all other output activations decrease. As z_i^L decreases, a_i^L decreases, and all the other output activations increase. In other words,

$$\sum_j a_j^L = \frac{\sum_j e^{z_j^L}}{\sum_k e^{z_k^L}} = 1.$$

One can interpret a_j^L as the probability that the correct classification is j. One can think of softmax as a way of rescaling z^L to form a probability distribution.

Learning Slowdown Problem

How does the sofmax layer address the learning slowdown problem? Let's define the log-likelihood cost function.

$$C \equiv -ln(a_y^L).$$

Why is this a cost function? When a_y^L is close to 1, then C is small. When a_y^L is smaller, C is larger.

Learning Slowdown Problem

In a network with a softmax output and log-likelihood cost, we have

$$\frac{\partial C}{\partial b_j^L} = a_j^L - y_j$$

$$\frac{\partial C}{\partial w_{jk}^L} = a_k^{L-1}(a_j^L - y_j)$$

Practical Insight

Should you use a sigmoid output layer and cross-entropy or a softmax output layer and log-likelihood?

Both approaches work well. Many ANNs use sigmoid output layers with cross-entropy. Many convolutional neural networks use softmax with log-likelihood.

Softmax plus log-likelihood is used when we want to interpret the output activations as probabilities.