PRIMER ON FITTING WEIGHTS IN NEURAL NETWORKS

ROHAN KADAKIA

ABSTRACT. We consider a very simple neural network: two input nodes, a single dense hidden layer also with two nodes, and a single output node. The goal is to clearly explain the processes of forward and backward propagation. This exposition assumes knowledge of vector calculus. Since there is just one output, this is a (supervised) regression problem. For additional simplicity, we focus on the case of only one observation (i.e., only one pair of inputs).

1. INTRODUCTION

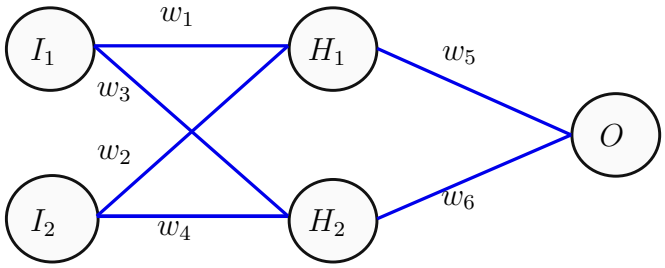

FIGURE 1. Simple neural network with weights labeled w_j

Figure 1 shows a neural network with input nodes $\vec{I} = (I_1, I_2)$, a hidden layer comprised of $\vec{H} = (H_1, H_2)$, and output node labeled O. The connections (sometimes called synapses) are labeled w_1, w_2 (feeding into H_1), w_3, w_4 (into H_2), and w_5, w_6.

For our loss/error function, we use mean squared error, and we take the rectified linear unit ('relu') as our activation function at each layer. This activation function $A : \mathbb{R} \to \mathbb{R}_{\geq 0}$ is given by

$$A(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0. \end{cases}$$

Popular alternatives to 'relu' include hyperbolic tangent or, more generally, any differentiable function with sigmoidal graph.

Start by fixing values for all weights $\vec{w} = (w_1, w_2, \ldots, w_6)$. The corresponding error E, in the case of n observations (i.e., n pairs of inputs), is

$$E(\vec{w}) = \frac{1}{n} \sum_{j=1}^{n} (p_j(\vec{w}) - a_j)^2,$$

Date: January 3, 2018.

the average of the squares of the differences between the predicted value $p_j(\vec{w})$ and the actual value a_j for each observation. The purpose of backward propagation is to systematically adjust the weights in order to minimize this error. For simplicity, we set $n = 1$ throughout this note, and we write $p(\vec{w}) = p_1(\vec{w})$, $a = a_1$.

Now fix values for the inputs $\vec{I} = (I_1, I_2)$ for our sole observation. As stated in the abstract, the activation function A is applied at each layer of the network. We therefore use the following notation:

$$\begin{aligned} \text{in}_{H_1} &= (I_1, I_2) \cdot (w_1, w_2) = I_1 w_1 + I_2 w_2 \\ \text{out}_{H_1} &= A(\text{in}_{H_1}) \end{aligned}$$

and similarly for H_2. For output node O,

$$\begin{aligned} \text{in}_O &= (\text{out}_{H_1}, \text{out}_{H_2}) \cdot (w_5, w_6) \\ \text{out}_O &= A(\text{in}_O). \end{aligned}$$

And notice that $\text{out}_O = p(\vec{w})$. Then, since we have already fixed the weights $\vec{w}$, we may employ *forward propagation* to determine out_O.

Example 1.1. Consider the neural network:

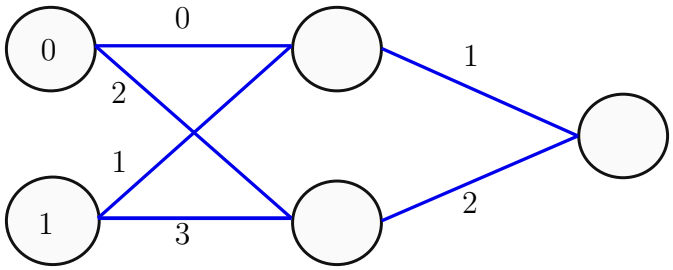

FIGURE 2. Network with numeric inputs and initial (untrained) weights

We wish to determine, by forward propagation, the predicted value: $p(\vec{w}) = \text{out}_O$ for $\vec{w} = (0, 1, 2, 3, 1, 2)$. First, we have

$$\begin{aligned} \text{in}_{H_1} &= (I_1, I_2) \cdot (w_1, w_2) = (0, 1) \cdot (0, 1) = 1 \\ \text{in}_{H_2} &= (I_1, I_2) \cdot (w_3, w_4) = (0, 1) \cdot (2, 3) = 3. \end{aligned}$$

More concisely,

$$\begin{bmatrix} \text{in}_{H_1} \\ \text{in}_{H_2} \end{bmatrix} = \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \end{bmatrix}.$$

Then $\text{out}_{H_1} = A(\text{in}_{H_1}) = 1$ and $\text{out}_{H_2} = A(\text{in}_{H_2}) = 3$. Finally,

$$\text{in}_O = (\text{out}_{H_1}, \text{out}_{H_2}) \cdot (w_5, w_6) = (1, 3) \cdot (1, 2) = 7,$$

so that $\text{out}_O = A(\text{in}_O) = 7$.

For concreteness, suppose the actual (target) value is $a = 4$. Then the error, after this first forward propagation using the initial weights $\vec{w}$, is

$$(p(\vec{w}) - a)^2 = (7 - 4)^2 = 9.$$

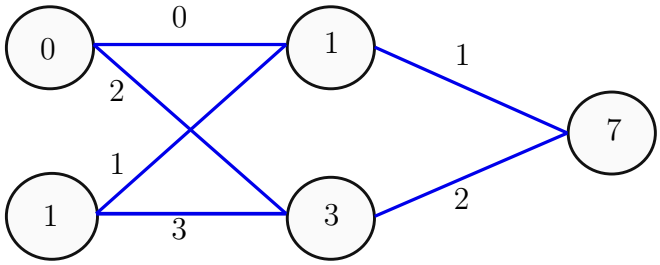

FIGURE 3. Network with predicted value after first forward propagation

2. Backward Propagation

With the goal of minimizing the error, we use the following theorem:

Theorem 2.1. *Let $f : \mathbb{R}^m \to \mathbb{R}$ be a differentiable function, and let $\vec{x}_0 \in \mathbb{R}^m$. Then the (directional) derivative of f at $\vec{x}_0$ is minimized in the direction given by $-\nabla f(\vec{x}_0)$ (and maximized in direction $\nabla f(\vec{x}_0)$).*

In our case $f = E$, the error function, $m = 6$ (as there are a total of six weights), and $\vec{x}_0 = \vec{w}$, the vector whose components are the weights of the neural network. This theorem justifies the *method of gradient descent*: the weights $\vec{w}$ are updated according to the formula

$$\vec{w}^+ = \vec{w} - \gamma \nabla E(\vec{w}), \tag{2.1}$$

where the entries of $\vec{w}^+$ are the new, adjusted weights, and γ is a small positive number called the *learning rate* (we are assuming a constant learning rate). The idea is that we are starting with some collection of weights $\vec{w}$ and, motivated by the theorem, we obtain $\vec{w}^+$ by traveling incrementally in the direction $-\nabla E(\vec{w})$. Commonly, $\gamma = 0.01$; in fact, this is the default learning rate when using `optimizer = 'sgd'` in Keras.

The operative term in (2.1) is $\nabla E(\vec{w})$, which measures the sensitivity of the error due to changes in each weight w_j. In the case of our model neural network (see Figure 1),

$$\nabla E(\vec{w}) = \left(\frac{\partial E}{\partial w_1}(\vec{w}), \ldots, \frac{\partial E}{\partial w_5}(\vec{w}), \frac{\partial E}{\partial w_6}(\vec{w}) \right).$$

Since it is backward propagation, we start on the right hand side on the network. To compute $\frac{\partial E}{\partial w_5}(\vec{w})$, notice that error is a function of out_O, in turn out_O is a function of in_O, and in_O is a linear function of w_5 and w_6:

$$\begin{aligned} E &= (\text{out}_O - a)^2 \\ \text{out}_O &= A(\text{in}_O) \\ \text{in}_O &= (\text{out}_{H_1}, \text{out}_{H_2}) \cdot (w_5, w_6). \end{aligned}$$

Then, by the chain rule,

$$\frac{\partial E}{\partial w_5} = \frac{dE}{d(\text{out}_O)} \times \frac{dA}{d(\text{in}_O)} \times \frac{\partial(\text{in}_O)}{\partial w_5}$$

$$= 2(\text{out}_O - a) \times A'(\text{in}_O) \times \text{out}_{H_1}.$$

In particular,

$$A'(\text{in}_O) = \begin{cases} 0, & \text{in}_O < 0 \\ 1, & \text{in}_O > 0. \end{cases}$$

(The 'relu' activation function is not differentiable at 0.) Similarly,

$$\frac{\partial E}{\partial w_6} = 2(\text{out}_O - a) \times A'(\text{in}_O) \times \text{out}_{H_2}.$$

Computing the remaining partial derivatives requires one additional step. For instance,

$$\begin{aligned} \frac{\partial E}{\partial w_1} &= \frac{\partial E}{\partial(\text{out}_{H_1})} \times \frac{dA}{d(\text{in}_{H_1})} \times \frac{\partial(\text{in}_{H_1})}{\partial w_1} \\ &= \left[\frac{dE}{d(\text{out}_O)} \times \frac{dA}{d(\text{in}_O)} \times \frac{\partial(\text{in}_O)}{\partial(\text{out}_{H_1})}\right] \times \frac{dA}{d(\text{in}_{H_1})} \times \frac{\partial(\text{in}_{H_1})}{\partial w_1} \\ &= 2(\text{out}_O - a) \times A'(\text{in}_O) \times w_5 \times A'(\text{in}_{H_1}) \times I_1. \end{aligned}$$

Likewise,

$$\begin{aligned} \frac{\partial E}{\partial w_2} &= 2(\text{out}_O - a) \times A'(\text{in}_O) \times w_5 \times A'(\text{in}_{H_1}) \times I_2 \\ \frac{\partial E}{\partial w_3} &= 2(\text{out}_O - a) \times A'(\text{in}_O) \times w_6 \times A'(\text{in}_{H_2}) \times I_1 \\ \frac{\partial E}{\partial w_4} &= 2(\text{out}_O - a) \times A'(\text{in}_O) \times w_6 \times A'(\text{in}_{H_2}) \times I_2. \end{aligned}$$

Example 2.2. Returning to the setting of our first example, we calculate

$$\begin{aligned} \frac{\partial E}{\partial w_5} &= 2(3) \times 1 \times 1 = 6 \\ \frac{\partial E}{\partial w_6} &= 2(3) \times 1 \times 3 = 18 \\ \frac{\partial E}{\partial w_1} &= 2(3) \times 1 \times 1 \times 1 \times 0 = 0 \\ \frac{\partial E}{\partial w_2} &= 2(3) \times 1 \times 1 \times 1 \times 1 = 6 \\ \frac{\partial E}{\partial w_3} &= 2(3) \times 1 \times 2 \times 1 \times 0 = 0 \\ \frac{\partial E}{\partial w_4} &= 2(3) \times 1 \times 2 \times 1 \times 1 = 12. \end{aligned}$$

Next, taking 0.01 as learning rate, we update the weights:

$$\vec{w}^+ = (0, 1, 2, 3, 1, 2) - (0.01)(0, 6, 0, 12, 6, 18) = (0, 0.94, 2, 2.88, 0.94, 1.82).$$

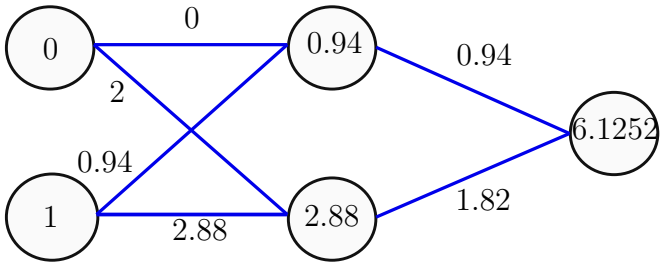

FIGURE 4. Network with predicted value determined by revised weights

Finally, we again perform forward propagation in the modified neural network with weights $\vec{w}^+$ in place of $\vec{w}$, resulting in Figure 4.

Indeed, the adjusted weights lead to an error $(p(\vec{w}^+) - a)^2 = (6.1252 - 4)^2 \approx 4.52$ which is nearly half the original error.

After revising weights and performing a second round of forward propagation, the procedure is repeated: backward propagation and weight adjustments, followed again by forward propagation, computing the error each time. Upon completing a sufficient number of iterations of this procedure, the value of the error will stabilize (that is, will approach its minimum).

DEPARTMENT OF MATHEMATICS, UNIVERSITY OF MICHIGAN, 530 CHURCH STREET, ANN ARBOR, MI 48109
E-mail address: kadakia@umich.edu