Backpropagation Algorithm

CSC357 Advanced Topic: Machine Learning

01 February 2020

1 Purpose of Training

Adjust the weights on connections in a neural network so that the network's computed values match predicted (expected) values as closely as possible.

2 The Key Data Structure

We will use a graph to model a neural network.

- A graph contains nodes and edges.
- The nodes represent artificial neurons.
- Nodes appear as bubbles that contain text in our picture of the graph.
- The edges represent connections between neurons.
- Edges appear as line segments or arcs with attached text in our picture of the graph.

The neural network in this example has three layers.

- an input layer
- a hidden (middle) layer
- an output layer

We distinguish among four kinds of nodes:

- Input nodes are in the network's first layer. They are data sources.

  They forward data to a subsequent layer. (They do not receive data—there is no previous layer for these nodes.)

- Bias nodes may be in an layer except for the last (output) layer. They are also data sources.

  They forward data to a subsequent layer. (They do not receive data—there is no previous layer for these nodes.)

- Hidden nodes are in the network's middle layers. They receive data from a previous layer and forward data to a subsequent layer.

- Output nodes are in the network's last layer. They are data sinks.

  They receive data from a previous layer. (They do not forward data—there is no subsequent layer for these nodes.)

  Output nodes hold a predicted (expected) value. They also compute a value using data that they receive from a previous layer.

3 The Key Equation

$$w \leftarrow w - \eta \frac{\partial E}{\partial w}$$

4 Nodes

The graph has 8 nodes.

The graph represents a network that has 3 layers numbered 0, 1, and 2.

- Layer 0 is the input layer. It contains 3 nodes, including one bias node. The first two nodes contain fixed input values.

- Layer 1 is the hidden layer. It contains 3 nodes, including one bias node.

- Layer 2 is the output layer. It contains 2 nodes. Each of these nodes contains an expected value (a value that we hope the machine will produce after training).

We will give the nodes names with the form Lij, where...

- L means "Layer"

- i indicates which layer contains the node

- j indicates which node within that layer

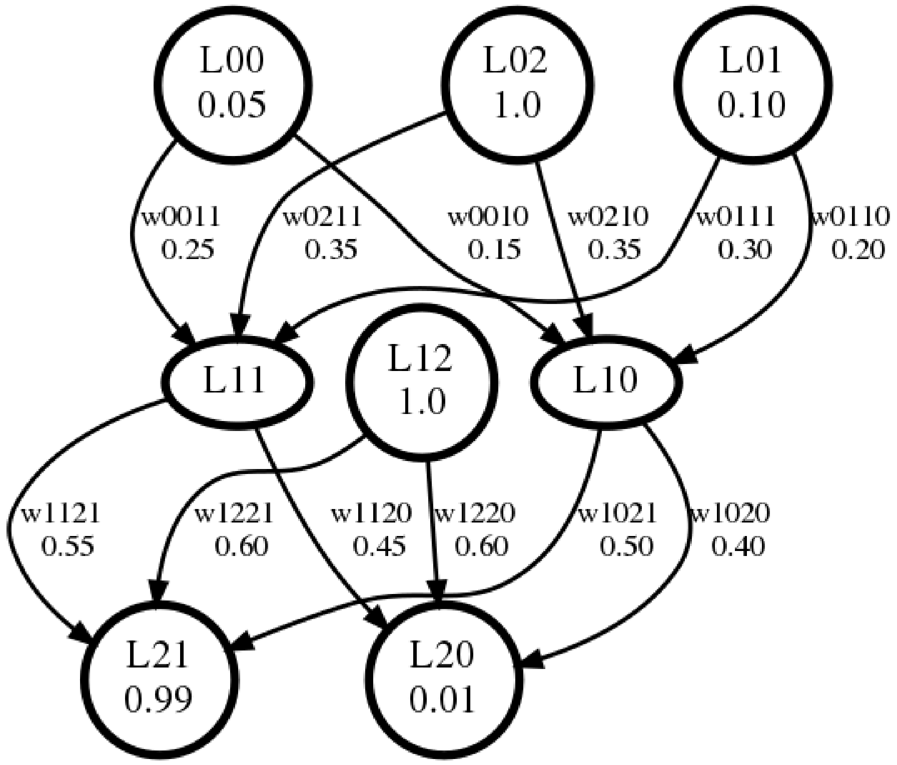

Figure 1: Multilayer network.

So, for example, $L10$ is node 0 in layer 1 (the second, hidden layer). Similary, $L21$ is node 1 in layer 2 (the output layer).

Mazur's name	**description**	**My name**	**value**
i1	Input 0	L00	0.05
i2	Input 1	L01	0.10
b1	Bias 0	L02	1.00
h1	Hidden 0	L10	—
h2	Hidden 1	L11	—
b2	Bias 1	L12	1.00
o1	Output 0	L20	0.01
o2	Output 1	L21	0.99

5 Edges

There are 12 edges in the graph:

- Six edges connect the nodes in Layer 0 to the nodes in Layer 1.
- Six edges connect the nodes in Layer 1 to the nodes in Layer 2.

We will give edges names with the form $wijmn$, where...

- w means "weighted edge"
- i denotes the layer of the node at which the edge begins
- j denotes a node in layer i
- m denotes the layer of the node at which the edge ends
- n denotes a node in layer m

Mazur's name	**My name**	**weight**	**description**
w1	w0010	0.15	$L00 \to L10$
w2	w0110	0.20	$L01 \to L10$
w3	w0011	0.25	$L00 \to L11$
w4	w0111	0.30	$L01 \to L11$
w5	w1020	0.40	$L10 \to L20$
w6	w1120	0.45	$L11 \to L20$
w7	w1021	0.50	$L10 \to L21$
w8	w1121	0.55	$L11 \to L21$
b1	w0210	0.35	$L02 \to L10$
b1	w0211	0.35	$L02 \to L11$
b2	w1220	0.60	$L12 \to L20$
b2	w1221	0.60	$L12 \to L21$

6 Forward pass

The input to a node n in layer $k+1$ is the dot product of two vectors:

- a vector whose elements are the values of the outputs of the nodes in layer k (that's the previous layer)
- a vector whose elements are the weights on the edges that connect the nodes in layer k to node n

Input nodes (nodes in the first layer of the network) and bias nodes are exceptions to this rule—their inputs are fixed constants.

The output of a node n is the value obtained by applying an activation function to its input.

We have a choice of activation functions. We will use the logistic function (also called the sigmoid function):

$$\sigma(t) = \frac{1}{1+e^{-t}}$$

Input nodes (nodes in the first layer of the network) and bias nodes are exceptions to this rule—their outputs are fixed constants.

7 Measurement of error

Each output node holds an expected value and produces a computed value. The difference between these two values is a measure of error. The square of the difference is also a measure of error.

The sum of the square of the differences at each output node is a measure of the error of the whole network. Of course, the squares of the differences are all non-negative values. Summing squares of differences (rather than summing just the differences) prevents positive errors from canceling negative errors.

The error E is a complicated function! It is a function of the expected values in the output nodes and the values computed in the output nodes. The values computed in the output nodes in turn depend upon the weights of the edges that lead into the output nodes and the values computed in the nodes in the previous layer. Those computed values in turn depend upon other weights and upon values computed in an even earlier layer, and so on.

Maybe we can use the error function E to find a better set of weights? A network with a better set of weights will produce less error. We will hold input values and biases constant but adjust weights on edges.

8 Backward pass

8.1 Weights on edges that feed into output nodes

Let's find out how the error E depends upon the weight of one edge that feeds into one output node.

Let's suppose that a network has output nodes number 0 to n.

Let's suppose that the values computed at those nodes (their outputs) are:

$$\text{computed values} \qquad = \{c_0, c_1, c_2, \ldots, c_n\}$$

Here, "c" stands for "computed value"—the value obtained by passing the value that is fed into a node through the activation function.

Let's suppose that the expected values at those nodes (the predicted values) are:

$$\text{predicted (expected) values} \qquad = \{p_0, p_1, p_2, \ldots, p_n\}$$

A measure of error is:

$$\begin{aligned} E &= \frac{1}{2}\sum_{i=0}^{n}(c_i - p_i)^2 \\ &= \frac{1}{2}\left[(c_0 - p_0)^2 + (c_1 - p_1)^2 + (c_2 - p_2)^2 + \cdots (c_n - p_n)^2\right] \end{aligned}$$

This measure is just the sum of squared differences multiplied by one half. The multiplication will simplify arithmetic in the next step. The expression is still a measure of error, since what counts are not absolute values but relative values. It is like converting a measure of height that is given in inches to a measure in centimeters (height in centimeters = 2.54 $\times$ height in inches)—inches or centimeters, it is still clear who is taller and who is shorter.

Let's suppose that we want to know how the error varies with small changes to weight on some edge that feeds into some output node.

- let E be the error function
- let w be the weight on that edge
- let c be the output (the computed value) at that output node

- let f be the value fed into that output node

The partial derivative of a function $g(x, y, z, \ldots)$ with respect to x tells how small changes to x affect the value of g. If might be that x is itself a function of some other variable—and maybe that other variable's value in turn depends upon the value of yet another variable. In that case, apply the chain rule of differentiation.

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial c} \cdot \frac{\partial c}{\partial f} \cdot \frac{\partial f}{\partial w}$$

Recall that c is the output of some particular output node. Let's say that we have chosen to focus on the first output node, so $c = c_0$.

$$\begin{aligned} E(c_0, c_1, c_2, \ldots, c_n) &= \frac{1}{2}\left[(c_0 - p_0)^2 + (c_1 - p_1)^2 + (c_2 - p_2)^2 + \cdots (c_n - p_n)^2\right] \\ \frac{\partial E}{\partial c_0} &= \frac{1}{2}\left[2(c_0 - p_0) + 0 + 0 + \ldots + 0\right] \\ &= (c_0 - p_0) \end{aligned}$$

The derivative of the sigmoid function is:

$$\begin{aligned} \sigma'(f) &= \frac{d\sigma(f)}{df} \\ &= \sigma(f)(1 - \sigma(f)) \end{aligned}$$

The output of node 0 is c_0. This is a value known to us. It is a value that we computed in the forward pass. The application of the activation function to the dot product f that the network fed into node 0 gave us c_0.

$$c_0 = \sigma(f)$$

The derivative $\sigma'(f)$ at the same point is therefore just:

$$\begin{aligned} \sigma'(f) &= \frac{\partial c}{\partial f} \\ &= c(1 - c) \end{aligned}$$

In the case of node 0, the value of the partial derivative is $c_0(1 - c_0)$.

Let's pause to look at what we have accomplished thus far:

$$\begin{aligned}\frac{\partial E}{\partial w} &= \frac{\partial E}{\partial c_0} \cdot \frac{\partial c_0}{\partial f} \cdot \frac{\partial f}{\partial w} \\ &= (c_0 - p_0) \cdot c_0(1 - c_0) \cdot \frac{\partial f}{\partial w}\end{aligned}$$

We want to express one derivative as the product of three other derivatives. We have expressions that give us the values of the first two of those three derivatives. We need only find an expression for the third derivative.

$$\frac{\partial f}{\partial w} = ?$$

The function f is a dot product of two vectors.

- there are m nodes in the previous layer of the network
- the outputs of those m nodes are $\{f_0, f_1, f_2, \ldots, f_m\}$ (these are the values that are *fed forward* from the previous layer)
- the weights on the edges that connect those nodes to our output node are $\{w_0, w_1, w_2, \ldots, w_m\}$

$$\begin{aligned}f &= \sum_{i=0}^{m} f_i w_i \\ &= f_0 w_0 + f_1 w_1 + f_2 w_2 + \cdots + f_m w_m\end{aligned}$$

We said that we were interested in the first edge that fed into our output node. Its weight is w_0.

$$\begin{aligned}\frac{\partial f}{\partial w_0} &= \frac{\partial}{\partial w_0}(f_0 w_0 + f_1 w_1 + \cdots + f_m w_m) \\ &= f_0\end{aligned}$$

Now we have a complete expression:

$$\begin{aligned}\frac{\partial E}{\partial w_0} &= \frac{\partial E}{\partial c_0} \cdot \frac{\partial c_0}{\partial f} \cdot \frac{\partial f}{\partial w_0} \\ &= (c_0 - p_0) \cdot c_0(1 - c_0) \cdot f_0\end{aligned}$$

Take this value, multiply it by a learning rate η, and subtract the product from w_0. Replace w_0 with this new edge weight.

Here's the whole recipe for updating the weight on an edge that feeds into an output node:

- w_j is the weight on edge j
- edge j connects node j on the previous layer to output node k
- f_j is the output (computed value) on node j on the previous layer
- the output (computed value) on output node k is c_k
- the predicted (expected) value on output node k is p_k

$$w_j \leftarrow w_j - (c_k - p_k) \cdot c_k(1 - c_k) \cdot f_j$$

8.2 Weights on edges that feed into hidden nodes

Again, let's look for a way to find out how much the weight on one edge that feeds into one node in the hidden layer affects the error.

- E is the error function
- h is the output (computed value) from the hidden node on which we are focused
- w is the weight on an edge that connects an input node to the hidden node on which we are focused
- f is the input into that hidden node

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial h} \cdot \frac{\partial h}{\partial f} \cdot \frac{\partial f}{\partial w}$$

The last two parts of this product are the same is before. The first part is different, because the output of our hidden node will affect the errors that come out of all of the nodes in the output layer.

In our example, there are two nodes in the output layer:

- let E_0 denote that part of the network's error that comes from output node 0

- let E_1 denote that part of the network's error that comes from output node 1

$$\frac{\partial E}{\partial h} = \frac{\partial E_0}{\partial h} + \frac{\partial E_1}{\partial h}$$

The calculation requires a few more steps in this middle layer than it did in the output layer.

- E_0 represents the contribution of output node 0 to the network's error
- f_0 represents the input into output node 0
- h_0 represents the output (computed value) from the hidden node on which we are focused
- c_0 represents the output (computed value) from output node 0
- c_1 represents the output (computed value) from output node 1

$$\frac{\partial E_0}{\partial h_0} = \frac{\partial E_0}{\partial f_0} \cdot \frac{\partial f_0}{\partial h_0}$$

We have seen the two derivatives on the r.h.s. of the equation before. In the first case, ...

$$\begin{aligned}\frac{\partial E_0}{\partial f_0} &= \frac{\partial E_0}{\partial c_0} \cdot \frac{\partial c_0}{\partial f_0} \\ &= (c_0 - p_0) * (c_0(1 - c_0))\end{aligned}$$

In the second case, recall that f_0 (the input into output node 0) is the dot product of two vectors. One vector contains the weights on all of the weights on the edges that connect hidden nodes to output nodes. The other vector contains the outputs (computed values) of all of the hidden nodes.

- let w_0 represent the weight on the edge that connnects the hidden node on which we are focused to output node 0

A dot product is a sum of products. Only one of the products in f_0 contains h_0. In the derivative with respect to h_0, all terms but that one are treated like constants—their derivatives are zero. The value of the derivative of f_0 with respect to h_0 is just the weight on one edge (in this case, the edge that connects our hidden node to output node 0).

$$f_0 = h_0 w_0 + h_1 w_1 + h_2 w_2 + \cdots$$

$$\begin{aligned}\frac{\partial f_0}{\partial h_0} &= w_0 + 0 + 0 + \cdots \\ &= w_0\end{aligned}$$

Here is how output node 0's contribution to the error depends upon the output (computed value) from our hidden node:

$$w_0 \cdot (c_0 - p_0) \cdot (c_0(1 - c_0))$$

We want to know how the total error depends on the output of our hidden node. We need to compute output node 1's contribution in the same way.

- let w_1 be the weight on the edge that connects our hidden node to output node 1
- let c_1 be the output (computed value) of output node 1
- let p_1 be the predicted (expected) value of output node 1

$$w_1 \cdot (c_1 - p_1) \cdot (c_1(1 - c_1))$$

Put these pieces together to get:

$$\frac{\partial E}{\partial h_0} = \frac{\partial E_0}{\partial h_0} + \frac{\partial E_1}{\partial h_0}$$

$$= [w_0 \cdot (c_0 - p_0) \cdot (c_0(1 - c_0))] + [w_1 \cdot (c_1 - p_1) \cdot (c_1(1 - c_1))]$$

Remember that we are trying to get an expression for $\frac{\partial E}{\partial w}$:

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial h} \cdot \frac{\partial h}{\partial f} \cdot \frac{\partial f}{\partial w}$$

We have the first part of the product on the right and there's good news! The second and third parts of the product on the right are easy.

$$\frac{\partial h}{\partial f} = h(1-h)$$
$$\frac{\partial f}{\partial w} = f_j$$

Now, put it all together...

- E is a measure of the error in the whole network (a sum of squared differences)
- w is weight on an edge that connects an input node to a hidden node—it is the value that we want to adjust
- w_0 is the weight on the edge that connects the hidden node to output node 0
- w_1 is the weight on the edge that connects the hidden node to output node 1
- c_0 is the output (computed value) on output node 0
- p_0 is the predicted (expected) value on output node 0
- c_1 is the output (computed value) on output node 1
- p_1 is the predicted (expected) value on output node 1
- h is the output (computed value) on the hidden node that lies at one end of the edge whose weight we are adjusting
- f_j is the output of the input node that lies at the other end of the edge whose weight we are adjusting

$$\frac{\partial E}{\partial w} = [(w_0 \cdot (c_0 - p_0) \cdot (c_0(1-c_0))) + (w_1 \cdot (c_1 - p_1) \cdot (c_1(1-c_1)))] \cdot (h(1-h)) \cdot f_j$$

This gives us the means to update the weight on an edge:

$$w \leftarrow w - \eta \frac{\partial E}{\partial w}$$