Statistical Mechanics, Neural Networks, and Artificial Intelligence: *Using Powerful Brain Strategies to Improve AI*

DRAFT Chapter 6:
Backpropagation:
The Classic Stochastic Gradient Descent Learning Method for Neural Networks

Alianna J. Maren
Northwestern University School of Professional Studies
Master of Science in Data Science Program

Draft: 2020-01-05

6.1 Introduction to Stochastic Gradient Descent and the Backpropagation Method

to be written

6.2 Backpropagation Overview: The Big Picture

to be written

6.3 The Sigmoid Transfer Function: A Quick Review

6.3.1 The Transfer Function and It's Derivative

For this chapter, we'll use the *sigmoid transfer function*,which we studied in the previous chapter. The reasons that we're using this particular function are:

- It has the requisite properties to be used in the backpropagation algorithm,
- It is relatively simple, and
- We have already (in the previous chapter) obtained its derivative.

As a quick review, we express the sigmoid transfer function as

$$y = \frac{1}{1 + exp(-\alpha x)} = (1 + exp(-\alpha x))^{-1}. \tag{6.1}$$

As a refresher, we show this function again in Fig. 6.1.

Once again, we'll note that it has the properties needed for our purposes, that is:

- It is continuous and smoothly differentiable,
- It is bounded, and
- It is monotonically increasing.

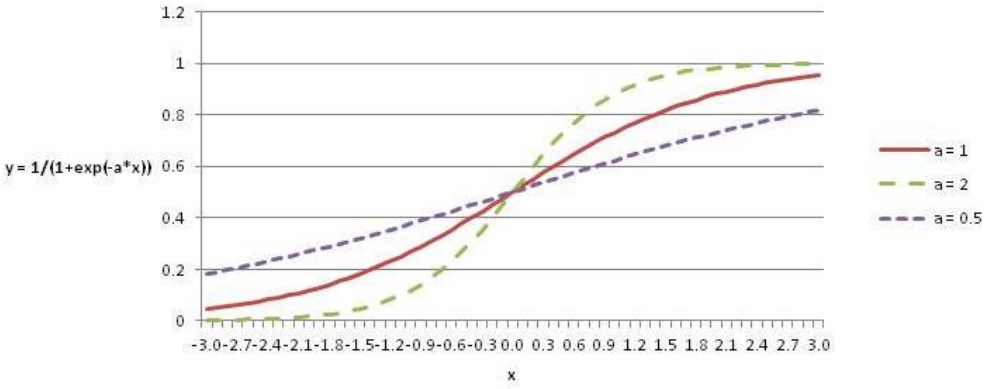

Figure 6.1: The simple transfer function.

6.3.2 The Derivative of the Sigmoid Transfer Function

The derivative of the transfer function plays an important role in backpropagation, so we'll collect our final result from the previous chapter, so we have it on hand for our work here.

We previously found derivative of the sigmoid transfer function as:

$$\partial y/\partial x = \alpha y\big[1 - y\big]. \tag{6.2}$$

In the previous chapter, we did our calculations in a formal mathematical sense. Now, we want to re-interpret our variables y and x in terms of the operations in a neural network.

For a given node, the output (or the activation) of that node corresponds to the y that we've been working with in our previous derivations. The value for y is what we get after we pass the summed, weighted inputs into the transfer function. We have previously constructed a variable for this *node input*.

Let's take the case where we are looking at a specific output node; let's call it the zero-th output node. (That means, counting Python-style, that we're dealing with the first output node.)

We'll use this same approach to deal with nodes on both the hidden and output layers, because each node on each of these two layers produceds an *output* based on a transfer function applied to the summed, weighted inputs into that particular node. The only difference is that:

- For any given node in the output layer, the inputs to that node come from the hidden layer nodes, i.e. from nodes $(H_0, ..., H_H)$, and
- For any given node in the hidden layer, the inputs to that node come from the input layer nodes, i.e. from nodes $(I_0, ..., I_I)$.

Thus, for a given output node o, we define the sum of the inputs into that output node o as $NdInpt_o$, and describe it mathematically as

$$NdInpt_o = \sum_{o=1}^{O} v_{h,o} * H_h, \tag{6.3}$$

where we recall that we use the convention of naming the hidden-to-output connection weights as the set of $v_{h,o}$ weights, where the subscripts h, o mean that we're referring to the connection weight from the h^{th} hidden node up to the o^{th} output node.

For any given output node, there are a set of h connection weights leading into it. The total set of of connection weights from the hidden to the output node is $h * o$. A similar logic applies to the set of input-to-hidden connection weights.

We refer now back to our calculation for the transfer function derivative. We used $\mathscr{F}$ to represent the transfer function. The input to the transfer function is the entire sum of the weighted inputs, or $NdInpt_o$ (for a given output node). This is the x in our previous derivation.

Similarly, the value for y is the output of the node, or the *activation* of this node, as A_o. Thus, we write

$$A_o = \mathscr{F}(NdInpt_o). \tag{6.4}$$

We have previulsy obtained the derivative for the sigmoidal transfer function. For reference, we see it again as Figure ...

Previously, we derived and discussed the derivative of the sigmoid transfer function, which we see again here in Fig. 6.2.

We're going to be embedding this derivative into the backpropagation learning method, which embodies a multiplication of several terms, using the chain rule from basic calculus.

As we glance at the derivative graph, and recall how the derivative of the sigmoid function goes to zero when the inputs are large numbers (whether those large numbers are negative or positive), we can see that if

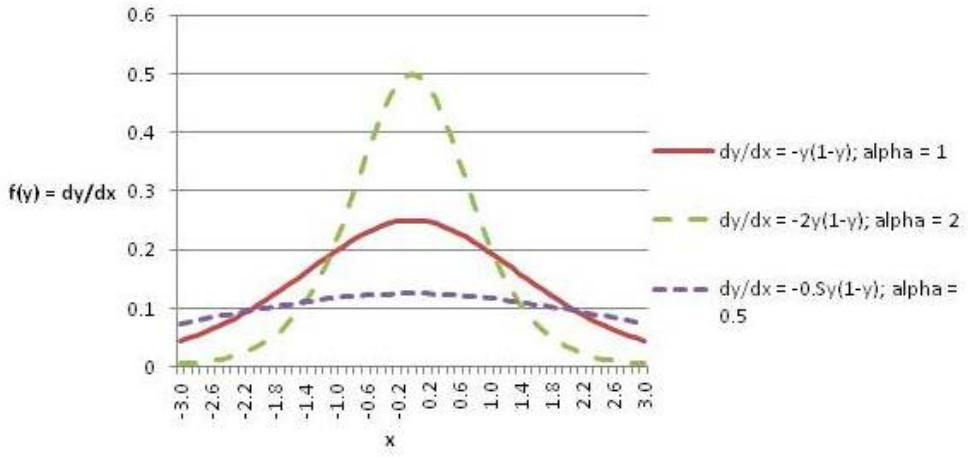

Figure 6.2: The derivative of the simple sigmoid transfer function.

we're introducing a term that is close to zero into a calculation - specifically if we are *multiplying* other terms by a term that is close to zero - then we're going to be getting a close-to-zero value as a result.

This means that if the inputs to the transfer function are large numbers (either large positive or large negative), then we'll have problems. We'll be introducing a result that is close to zero into a series of multiplications. Whatever good that series of multiplications is supposed to do for us, it will not be very effective if it's close to zero, right?

This is the heart and soul of what became known as the "vanishing gradient" problem. It's the primary reason that using backpropagation alone (or similar stochastic gradient descent methods) has limiations.

Note that this does not mean that we can't have large connection weights. The whole notion of keeping the $NdInpt_o$ from going to a large value (whether positive of negative) **does not** mean that we have to have small values for our connection weights. What it **does** mean is that the **sum** of the connection weights, times the values that they're multiplying, cannot get very large.

Obviously, there's no easy way to control this during network training. That's why sometimes a network will stall out during training, or take a *very* long time (tens of thousands of iterations) to train.

There are all sorts of strategies to deal with this, and we won't go into them just now. The key thing that we want to understand is how the derivative of the transfer function impacts the whole backpropagation training process.

With this little nugget in mind, let's go from the transfer function and its derivative to the entire backpropagation learning method.

6.4 Backpropagation of Error: Part 1

We adjust the weights so that the feedforward propagation of neuron activations yields results (in the output layer) that are close to the desired outputs.

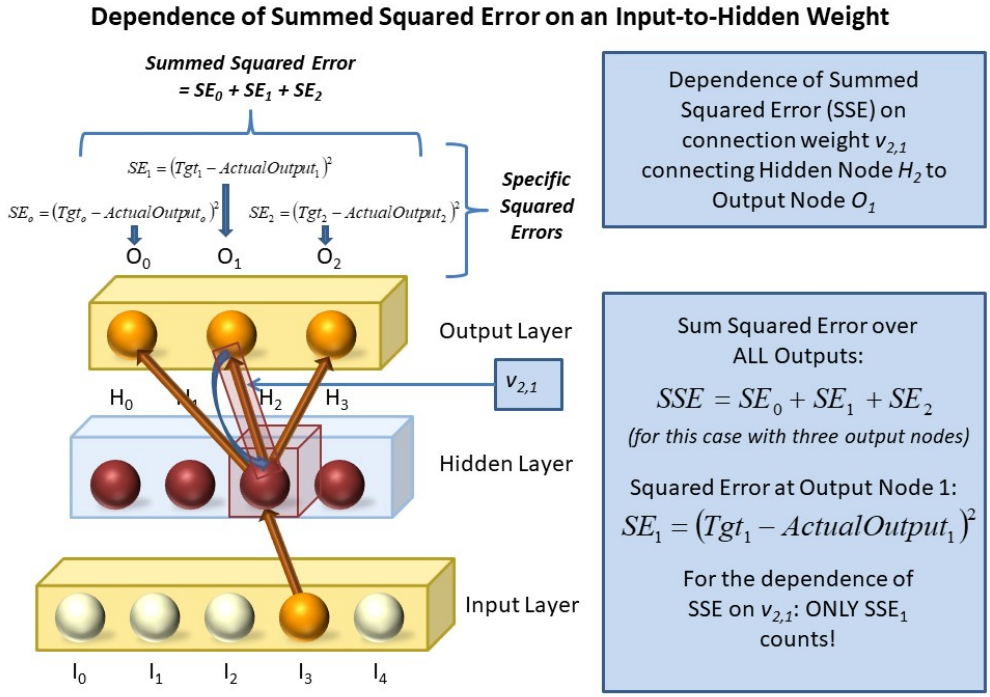

Figure 6.3: Dependence of the SSE on a single v weight.

We define the error term as:

$$SSE = \frac{1}{2}\sum_{o=1}^{O}(Desired_o - Actual_o)^2, \tag{6.5}$$

The multiplying factor of 1/2 is introduced so that when we take the derivative of this term, the factor of two that results from the derivative

of a squared term multiplies the factor of 1/2, neutralizing the impact of constants.

The specific error associated with any given output node is:

$$Error_o = Desired_o - Actual_o \tag{6.6}$$

Notice that we square these errors before summing across all the output nodes in the network.

Our goal is to adjust the weights to as to minimize the total error produced in the network. As we do so, we will also be minimizing the errors associated with any given specific output node.

Because the things that we can adjust are the actual weights themselves, what we want to compute is the dependence of the error on the weights. Specifically, we want two entirely different sets of dependencies:

- The dependence of the error on the weights connecting hidden to output nodes, and
- The dependence of the error on the weights connecting the input to the hidden nodes.

Both of these calculations use the chain rule from calculus. Also, since any one error term is a function of multiple different inputs (to the corresponding output node), any one error term has multiple dependency paths. Thus, we do our computations using *partial* derivatives; we seek to trace the unique and specific dependence that the error has on each contributing source.

We start by computing the dependence of the Summed Squared Error, SSE, on a specific hidden-to-output weight.

$$\frac{\partial SSE}{\partial v_{h,o}} = \frac{\partial SSE}{\partial E_o}\frac{\partial E_o}{\partial v_{h,o}} = \frac{\partial SSE}{\partial E_o}\frac{\partial E_o}{\partial A_o}\frac{\partial A_o}{\partial v_{h,o}} \tag{6.7}$$

So far, this equation tells us that to compute the dependence of the SSE on a given weight, $v_{h,o}$ (a weight connecting hidden node h to output node o), we compute:

- The dependence of the SSE on the specific error at output node o,
- The dependence of the specific *error* at output node o on the *actual output* A at output node o, and

- The dependence of the *actual output* A at output node o on the weight connecting the hidden node h to that output node o.

Before we plug in various values for each of these multiplying terms, there is one more step that we can take. We know that the actual output A at a given node is the transfer function applied to the sum of the weighted inputs going into that node.

$$A_o = \mathscr{F}(\sum_{h=1}^{H} v_{h,o} * H_h) \tag{6.8}$$

where the sum is being taken over all of the H hidden nodes, each contributing an input to a given output node o. In this expression, H_h is the output of the h^{th} hidden node.

This equation simply states that the actual output A_o at output node o equals the transfer function $\mathscr{F}$ (which we discussed in Subsection **??**, applied to the sum of the inputs to that output node. Further, each of the inputs to that output node is made up of the *weight* $v_{h,o}$ connecting a specific hidden node h to the specific output node o, multipled by the output from that specific hidden node H_h.

For simplicity in writing the longer chain rule expression, let's identify the sum of the inputs into a given output node o as $NdInpt_o$.

Thus we can write

$$NdInpt_o = \sum_{o=1}^{O} v_{h,o} * H_h \tag{6.9}$$

and

$$A_o = \mathscr{F}(NdInpt_o) \tag{6.10}$$

Now we are ready to go back to Eqn. 6.7 and introduce a substitution taken from Eqn. 6.10

$$\frac{\partial SSE}{\partial v_{h,o}} = \frac{\partial SSE}{\partial E_o}\frac{\partial E_o}{\partial A_o}\frac{\partial A_o}{\partial v_{h,o}} = \frac{\partial SSE}{\partial E_o}\frac{\partial E_o}{\partial \mathscr{F}(NdInpt_o)}\frac{\partial \mathscr{F}(NdInpt_o)}{\partial v_{h,o}} \tag{6.11}$$

We introduce one more substitution; we break down the dependence of the *Error* at node o ($Error_o$) on the results of transfer function applied to the weighted node inputs into two terms:

- The dependence of the *Error* at node o ($Error_o$) on the result of the transfer function itself, and
- The dependence of the result of the transfer function on the actual sum of the weighted inputs into that node.

This gives us

$$\begin{aligned}\frac{\partial SSE}{\partial v_{h,o}} &= \frac{\partial SSE}{\partial E_o}\frac{\partial E_o}{\partial \mathscr{F}(NdInpt_o)}\frac{\partial \mathscr{F}(NdInpt_o)}{\partial v_{h,o}} \\ &= \frac{\partial SSE}{\partial E_o}\frac{\partial E_o}{\partial \mathscr{F}}\frac{\partial \mathscr{F}(NdInpt_o)}{\partial NdInpt_o}\frac{\partial NdInpt_o}{\partial v_{h,o}}\end{aligned} \tag{6.12}$$

This means that we can compute the dependence of the summed squared error (that which we are using as our overall error term) on a specific weight connecting a specific hidden node to a specific output node as the multiplication of four distinct terms.

Now, all that we have to do is to compute each of these specific terms, substitute them in, and we have the backpropagation of error onto a given hidden-to-output connection weight.

Dependence of the Summed Squared Error on the Error Term

We begin with referencing Eqn. 6.5, and note that

$$SSE = \frac{1}{2}\sum_{q=1}^{O}(Desired_q - Actual_q)^2 = \frac{1}{2}\sum_{q=1}^{O}E_q^2. \tag{6.13}$$

Notice that we've changed the running index from o (as was used in Eqn. 6.5) to q. This is because we run the sum over all of the output errors, from all the output nodes, but want to find (in our next step) the dependence of the summed squared error on the error at a *specific* node.

We begin by substituting our expression for the summed squared error into the term for the dependence of the SSE on a specific error.

$$\frac{\partial SSE}{\partial E_o} = \frac{1}{2}\frac{\partial \sum_{q=1}^{O} E_q^2}{\partial E_o} \tag{6.14}$$

We notice right away that when we have a sum of different things, each dependent on a different node, that the only real dependence comes from the node that we are actually considering. That is, we get a dependence of the SSE on the error at node *o only* when we are dealing with the squared error from node *o*; none of the other squared error terms matter. That is, for the case of finding the dependence of the SSE on the error in the hidden-to-output connection weight dependence *only*, we can go from the sum to a single term.

We immediately simplify our equation by removing the sum, and just considering the node in question.

$$\frac{\partial SSE}{\partial E_o} = \frac{\partial SE}{\partial E_o} = \frac{1}{2}\frac{\partial E_{q=o}^2}{\partial E_o} \tag{6.15}$$

This is much easier now. We apply some basic calculus, recalling that

$$\frac{1}{2}\frac{dx^2}{dx} = 2 * \frac{1}{2}x = x \tag{6.16}$$

Thus we have

$$\frac{\partial SSE}{\partial E_o} = \frac{1}{2}\frac{\partial E_{q=o}^2}{\partial E_o} = E_o \tag{6.17}$$

Dependence of the Error on the Error Actual Output

We've just computed the first term from Eqn. 6.12, and are ready to compute the second, which is the dependence of the actual error on the activation at the output node. We want to find

$$\frac{\partial E_o}{\partial \mathscr{F}_o} \tag{6.18}$$

As we noticed earlier, the error term E_o is simply

$$E_o = Desired_o - Actual_o = Desired_o - \mathscr{F}_o, \tag{6.19}$$

where $\mathscr{F}_o$ is being used as shorthand for the entire activation at output node o, that is, $\mathscr{F}_o = \mathscr{F}(NdInpt_o)$.

We apply a rule of calculus, by which

$$\frac{d(a-x)}{dx} = -1 \tag{6.20}$$

to obtain

$$\frac{\partial E_o}{\partial \mathscr{F}_o} = -1. \tag{6.21}$$

Dependence of the Actual Output on the Weighted Summed Inputs

We previously computed the derivative of the transfer function in Subsection **??**, Eqn.6.2, to have

$$\partial y/\partial x = \alpha y\big[1 - y\big], \tag{6.22}$$

where y is identified as the transfer function $\mathscr{F}$, and x is the weighted summed input into a node.

We will use this to identify the value of the third term in Eqn. 6.12, so that

$$\frac{\partial \mathscr{F}(NdInpt_o)}{\partial NdInpt_o} = \alpha \mathscr{F}_o\big[1 - \mathscr{F}_o\big] \tag{6.23}$$

We have just one more term to compute in order to have the set of four terms that we need for backpropagating error from an output node to a weight connecting a hidden to that output node.

Dependence of the Weighted Summed Inputs on an Individual Connection Weight

Our final task, for this stage of backpropagation computation, is to determine

$$\frac{\partial NdInpt_o}{\partial v_{h,o}}. \tag{6.24}$$

We recall, from Eqn. 6.8, that $NdInpt_o$, the actual sum of weighted inputs to node o, is expressed as

$$NdInpt_o = \sum_{h=1}^{H} v_{h,o} * H_h. \tag{6.25}$$

Thus, we are seeking

$$\frac{\partial NdInpt_o}{\partial v_{h,o}} = \frac{\partial \sum_{q=1}^{H} v_{q,o} * H_h}{\partial v_{h,o}}. \tag{6.26}$$

Once again, we've changed the running index on the sum from h to q, to express that we are considering all possible contributions. Of these, however, only one term is dependent on $v_{h,o}$, so that we have

$$\frac{\partial NdInpt_o}{\partial v_{h,o}} = \frac{\partial v_{h,o} * H_h}{\partial v_{h,o}} = H_h. \tag{6.27}$$

We substitute all four terms into our previous Eqn. 6.12

$$\frac{\partial SSE}{\partial v_{h,o}} = \frac{\partial SSE}{\partial E_o}\frac{\partial E_o}{\partial \mathscr{F}}\frac{\partial \mathscr{F}}{\partial (NdInpt_o)}\frac{\partial (NdInpt_o)}{\partial v_{h,o}} \tag{6.28}$$

$$= E_o * (-1) * \alpha \mathscr{F}_o\big[1 - \mathscr{F}_o\big] * H_h \tag{6.29}$$

$$= -\alpha E_o \mathscr{F}_o\big[1 - \mathscr{F}_o\big] H_h \tag{6.30}$$

Before we move on, let's see if we can't interpret this equation just a bit. Let's keep in mind that we're discussing how the squared error at a single output node depends on one of the specific connection weights connecting a given hidden node to that output node.

First, the lead term is α. The higher the value for α, the greater the dependence of the error on the connection weight. We're going to adjust this connection weight by some small fraction of the overall dependence term; it's going to be like taking a linear approximation of a slope, and incrementing the connection weight by a small value based on this linear slope approximation. What this means is that the larger the alpha, the larger the (approximately linear slope) adjustment, which means a faster change to the connection weight. In short, increasing α means that we'll typically change the connection weight more than if we had a smaller alpha. The overall impact of this depends on the overall nature of the gradient curve for which we're attempting to find a minimum, and that is beyond this particular discussion. We'll simply summarize by saying, " bigger α means faster change."

The size of the error term has an impact also. A bigger error term will also induce a faster change. As we get closer to a workable set of connection weights, the error terms will decrease in size; the adjustments to the connection weights will become smaller.

To assess the value of the terms involving the transfer function, we consult Figure 6.2. Let us recall that the notation $\mathscr{F}_o$ means that the transfer function is applied to the set of summed inputs to the output node O_o. The more that this set of inputs is either a large positive or large negative number, the more that the total set of terms involving the transfer function is small. This means that for very large positive or negative values of the total summed inputs, we decrease the amount of change that we make to a specific connection weight.

When we consider this, it makes sense. The set of terms involving the transfer function reflect the input of all the hidden nodes into into that particular output node. If the sum of those weighted inputs is large (at least in magnitude), it means that either there are very strongly activated hidden nodes sending inputs to that output node, or the weights connecting those hidden nodes to the output node are strong, or both. In either case, we're looking at the influence of hidden nodes other than the one that has the particular connection to the output that we're seeking to modify. This means that we should probably not make any great changes, as we'd distort the input of those other hidden nodes.

On the other hand, when the sum of the weighted activations going into that transfer function are more moderate - centered more-or-less around zero - then we can say that either the overall inputs from these other hidden nodes are not that strong, or that they are canceling each other out. In either of these cases, we have more leeway in adjusting the connection weight in question.

Finally, we note the impact of the last term, H_h. This is the actual activation of the hidden node h. The stronger that this activation is, in either a positive or negative sense, the more that it will impact the extent to which we change the connection weight.

This concludes the derivation of the dependence of the squared error on a specific hidden-to-output connection weight. The next step, in the next subsection, is to consider the dependence of the squared error on a given input-to-hidden connection weight.

As previously mentioned, this method was originally developed by Paul Werbos and presented in his Ph.D. dissertation at Harvard University [1].

Bibliography

[1] P. J. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.