# Deep Learning: Ideas, Proofs, Algorithms, Examples

Gurtek Singh Gill [1]
Montek Singh Gill [2]

**Abstract**

We give an account of deep learning which is mathematically rigorous, clear in explanations and contains both the fundamental algorithms in pseudocode as well as hands-on examples. In particular, we give a detailed proof of the back-propagation formulae which are used in practice for gradient descent when training artificial neural networks.

## Contents



## 1 An Overview of Deep Learning

The goal of machine learning is to learn a correspondence

$$f\colon X \to Y$$

where $X$ is some subset of a euclidean space $\mathbb{R}^n$ and $Y$ is some subet of $\mathbb{R}$. If $Y$ is a continuous subset of $\mathbb{R}$, we say that we have a *regression problem*, as we are attempting to learn how a continuously varying quantity behaves, and if $Y$ is a discrete subset of $\mathbb{R}$, we say that we have a *classification problem*, as we are essentially learning a partition of the points of $\mathbb{R}^n$ into different categories. We must *learn* in the following sense: we shall be given example inputs $\mathbf{x}_i \in X$ and associated outputs $y_i = f(\mathbf{x}_i) \in Y$, and based on these examples, this *training* so to speak, we must construct a predictor function

$$f_{\text{pred}}\colon X \to Y$$

in such a manner that, given any new $\mathbf{x} \in X$, $f_{\text{pred}}(\mathbf{x})$ and $f(\mathbf{x})$ coincide often. (Here one must of course replace "coincide often" with a mathematically precise statement, and there are various alternatives in any given situation, using different loss functions, as we shall see below.)

In fact, the preceding description applies only to what has come to be called *supervised* machine learning. There exist other flavours of machine learning (unsupervised and reinforcement) involving different kinds of

[1] rickygill01@gmail.com
[2] montekgill@google.com

learning problems, but supervised machine learning is the most developed so far.

Now, for most problems in machine learning, in order to come up with a good predictor function $f_{\text{pred}}$, one first considers a general parametric class $\{f_{\theta_1,\ldots,\theta_p}\}$ of functions $f_{\theta_1,\ldots,\theta_p}$, and then finds optimal choices for the parameters $\theta_1,\ldots,\theta_p$. For example, one might consider the class comprising the linear functions $f_{w_1,\ldots,w_n,b}\colon \mathbb{R}^n \to \mathbb{R}\colon (x_1,\ldots,x_n) \mapsto w_1x_2 + \cdots + w_nx_n + b$ (this leads to the classical technique of linear regression). Deep learning is the subfield of machine learning which concerns the use of a special parametric class of functions $\mathbb{R}^n \to \mathbb{R}$, namely, the artificial neural networks, to construct predictor functions $f_{\text{pred}}$.

# 2 Artificial Neural Networks

## 2.1 The Structure of Artificial Neural Networks

**Definition 1.** An *artificial neural network* is a function $\mathrm{N}\colon \mathbb{R}^n \to \mathbb{R}$, for some $n \geq 1$, of the form

$$\mathrm{T}_{L-1} \circ a \circ \mathrm{T}_{L-2} \circ \cdots \circ \mathrm{T}_1 \circ a \circ \mathrm{T}_0$$

where $L \geq 0$, each $\mathrm{T}_l$ is a linear function between euclidean spaces and $a$ is a non-linear function $\mathbb{R} \to \mathbb{R}$, to be applied entrywise to any given input vector.

Note the following:

- We refer to $L$ as the *depth* of the network; it is from this terminology that the field of deep learning gets its name. The function $a$ is referred to as the *activation function*.
- There is only one artificial neural network with $L = 0$, which exists only in the case where $n = 1$ and is simply the identity function on $\mathbb{R}$.
- The artificial neural networks with $L = 1$ comprise exactly the linear functions $\mathbb{R}^n \to \mathbb{R}$.
- It is only when $L \geq 2$ that we get a class of non-linear functions.
- If the function $a$ were linear, regardless of depth, we would always only get the class of linear functions (as composites of linear functions are once again linear). This is why we require that $a$ be non-linear.
- It is possible to consider artificial neural networks from $\mathbb{R}^n$ to $\mathbb{R}^m$, for $m \geq 1$, as opposed to just the case $m = 1$ here. We shall however focus only on the $m = 1$ case here, for simplicity, and because the details of the $m > 1$ cases are analogous to those here.

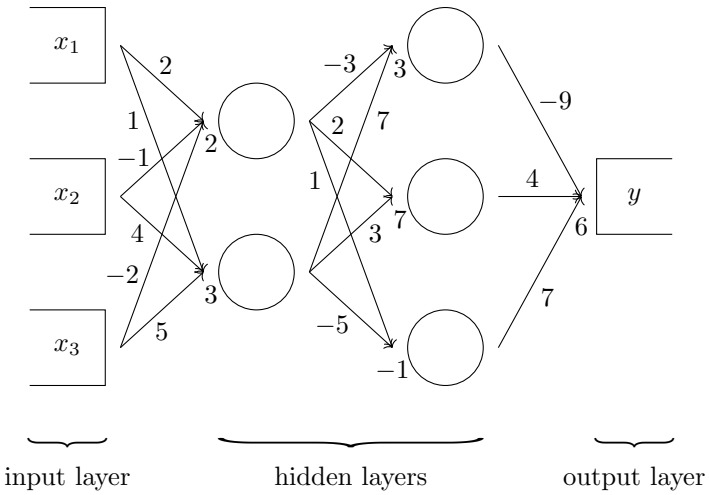
Now, each $\mathrm{T}_l$ is specified exactly by a matrix $\mathrm{W}^l = [w^l_{ij}]$ and a vector $\mathrm{B}^l = [b^l_i]$, each of the appropriate dimensions, where $\mathrm{T}_l(\mathbf{a}) = \mathrm{W}^l\mathbf{a} + \mathrm{B}^l$, for any input vector $\mathbf{a} \in \mathbb{R}^n$. To get a more concrete hold of the concept, let us consider an example artificial neural network, where say $n = 3$, $L = 3$, and:

$$\mathrm{W}^0 = \begin{bmatrix} 2 & 1 \\ -1 & 4 \\ -2 & 5 \end{bmatrix} \qquad \mathrm{B}^0 = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

$$\mathrm{W}^1 = \begin{bmatrix} -3 & 2 & 1 \\ 7 & 3 & -5 \end{bmatrix} \qquad \mathrm{B}^1 = \begin{bmatrix} 3 \\ 7 \\ -1 \end{bmatrix}$$

$$\mathrm{W}^2 = \begin{bmatrix} -9 \\ 4 \\ 7 \end{bmatrix} \qquad \mathrm{B}^2 = \begin{bmatrix} 6 \end{bmatrix}$$

We'll leave $a$ unspecified for now. Our neural network function $\mathrm{N}\colon \mathbb{R}^3 \to \mathbb{R}$ is given by $\mathrm{T}_2 \circ a \circ \mathrm{T}_1 \circ a \circ \mathrm{T}_0$. Given an input $(x_1, x_2, x_3)$, we see that the first rows of $\mathrm{W}^0$ and $\mathrm{B}^0$ tell us how to combine $x_1, x_2$ and $x_3$ to form the first entry of the output of $\mathrm{T}_0$. The second rows tell us how to form the second entry, and the third rows, the third entry. We then apply the activation function $a$ to each entry, and then proceed as before, now with $\mathrm{W}^1$ and $\mathrm{B}^1$. We can visualize this entire sequence of computations, and hence the artificial neural network itself, as follows:




Note the following:

- Numbers labelling arrows denote the weights (the entries of the $\mathrm{W}^l$) which we use to combine the input values; the numbers adjacent to the ○'s denote the values (the entries of the $\mathrm{B}^l$) which we add on to the combinations to form the outputs. Each ○ also carries with it an activation function which is applied to any input value.
- It is due to this visualization that we use the name "artificial neural network", as the displayed forward propagating computation has analogues in the human brain, via real networks of neurons, where a neuron corresponds to a ○ in the above picture. Moreover, this is also why we use the name "activation", as this too has analogues for neurons in the human brain, where $a$ corresponds to the manner in which a neuron passes on input electrical information to the other connected neurons. Moreover, as a result of this visualization and of this analogy, we have further terminology, as in the points below, associated with general artificial neural networks.
- Each ⊐, ○ and ⊏ is a *unit* or *cell* of the neural network; each ○ is a *neuron*.
- Each vertical layer of cells is a *layer* of the network. The first layer is the *input layer* and the last the *output layer*. Note that the cells in the input and output layers carry no activations ("are not neurons" one might say, but some other kinds of cell); they are simply placeholders for the inputs and outputs of artificial neural network functions (note that other authors may allow the output layer to carry an activation – we prefer to leave this final function as a hyperparameter of the deep learning algorithm). The layers between the input and output layers, if any, are termed the *hidden layers*, because they represent information and computations which are invisible when one simply feeds an input into an artificial neural network and inspects the output.
- The *architecture* of the artificial neural network comprises the depth, the numbers of hidden units in any hidden layer that exists, and also the choice for the activation $a$.

If we fix a depth, fix numbers of hidden units in any hidden layer that exists, and also fix a choice for the activation function $a$, which is to say if we fix an architecture for the artifical neural network, we then get a parametric class $\{\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}\}$ of functions

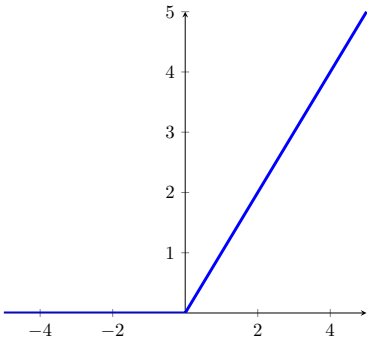
$$\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}} \colon \mathbb{R}^n \to \mathbb{R}$$

where $n$ is the number of input layer units. The parameters of this class are those of the linear functions of $\mathrm{T}_l$, that is the $w_{ij}^l$ and $b_i^l$. Constructing prediction functions via such parametric classes of functions is what constitutes the field of deep learning, where "deep" comes from the fact that sometimes networks with large depth are used.

## 2.2 Activation Functions

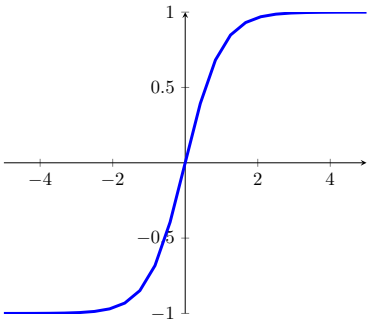
As part of the structure of an artificial neural network, we have the activation function $a$. So far, we have not specified any restrictions on $a$, beyond being non-linear. We of course must say something on exactly which non-linear functions we will typically want to use. The following are common choices:

- The ReLU function, aka the rectified linear unit. This is the function given by $x$ for $x \geq 0$ and 0 for $x < 0$. It's graph in the cartesian plane looks as follows:



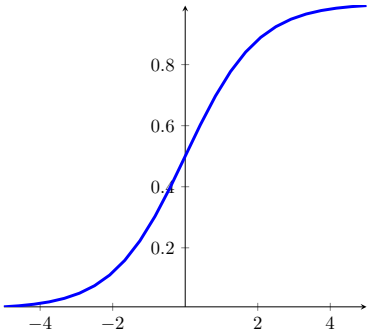

  It's effect is to discard any negative values input into a neuron (one could say that the neuron is activated only if the input signal is sufficiently strong).

- The hyperbolic tangent function. This is the function tanh where $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. It's graph in the cartesian plane looks as follows:




  It's effect is to compress all possible input values on the real line into the interval $(-1, 1)$, centred around 0.

- The sigmoid function. This is the function $\sigma$ where $\sigma(x) = \frac{1}{1+e^{-x}}$. It's graph in the cartesian plane looks as follows:




  It's effect is to compress all possible input values on the real line into the interval $(0, 1)$, centred around $1/2$.

# 3 The Loss Function

Let us return to the task of learning a correspondence $f\colon X \to Y$. As mentioned before, in deep learning, we begin by fixing an artificial neural network architecture, which provides us with a parametric class $\{\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}\}$ of functions $\mathbb{R}^n \to \mathbb{R}$. We wish to construct a predictor function $f_{\text{pred}}$ using this class. More precisely, we ask ourselves, if we restrict $f_{\text{pred}}$ to be one of the functions $\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}$ (or, as below, a slightly altered version of these in the case of classification problems), what is the best possible choice for the parameters $w_{ij}^l$ and $b_i^l$? That is, which choice of parameter values will yield the best predictor function? To answer this, we must have some means of measuring exactly how good any one possible choice $\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}$ is as a predictor. We do this by defining loss functions, which are designed to measure in fact just how bad a potential choice for the predictor is – and so we shall want to minimize the loss function.

## 3.1 Regression Problems: the $L^2$ Loss

Suppose that we have a regression problem, where we wish to learn a correspondence $f\colon X \to Y$, where $Y$ is some continuous subset of $\mathbb{R}$. We fix a choice of architecture for our artificial neural networks, yielding a class of potential predictor functions $\{\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}\}$.

Let us denote the training sample which we are provided by $\mathbf{x}_1, \ldots, \mathbf{x}_S, y_1, \ldots, y_S$, where $y_i = f(\mathbf{x}_i)$. Let $\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}$ be a general artificial neural network, of the specified architecture. The loss function, sometimes called the $L^2$ *loss* as it uses the $L^2$ norm, is as follows:

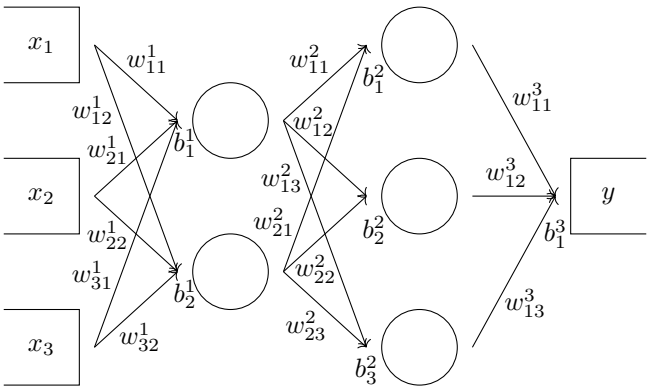
$$\mathcal{R}_{L^2}(\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}) := \frac{1}{S}\sum_{s=1}^{S}(\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}(\mathbf{x}_s) - y_s)^2$$

For brevity, for each $s = 1, \ldots, S$, one often sets $\widehat{y}_s := \mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}(\mathbf{x}_s)$, and then the loss function may be written as $(1/S)\sum_{s=1}^{S}(\widehat{y}_s - y_s)^2$. This is an appropriate choice for the loss function because it compares the guesses $\widehat{y}_s$ that our potential predictor function $\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}$ would make for the outputs when given the inputs $\mathbf{x}_s$, with the known outputs for these cases. The further apart these quantities, the larger the value $\mathcal{R}(\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i})$ of the loss function. Thus, to find optimal choices for the values of the parameters $w_{ij}^l$ and $b_i^l$, we seek to minimize this function.

Let's work out an example. Let's suppose that $n = 3$, so that we must construct a predictor function $\mathbb{R}^3 \to \mathbb{R}$. Let's also suppose that $S = 2$ and that our training sample is as follows:

$$\begin{aligned} \mathbf{x}_1 &= (2,4,3) \qquad & y_1 &= 5 \\ \mathbf{x}_2 &= (1,0,5) \qquad & y_2 &= -1 \end{aligned}$$

Next, let us work with artificial neural networks where $a = \sigma$ and where the remainder of the architecture is as follows:



For brevity, let just N denote the neural network function. Then, laborious but straightforward computatons demonstrate that:

$$\mathrm{N}(\mathbf{x}_1) = \frac{w_{11}^3}{1+e^{-\frac{w_{11}^2}{1+e^{-2w_{11}^1-4w_{12}^1-3w_{13}^1-b_1^1}}-\frac{w_{12}^2}{1+e^{-2w_{21}^1-4w_{22}^1-3w_{23}^1-b_2^1}}-b_1^2}} + \frac{w_{12}^3}{1+e^{-\frac{w_{21}^2}{1+e^{-2w_{11}^1-4w_{12}^1-3w_{13}^1-b_1^1}}-\frac{w_{22}^2}{1+e^{-2w_{21}^1-4w_{22}^1-3w_{23}^1-b_2^1}}-b_2^2}} + \frac{w_{13}^3}{1+e^{-\frac{w_{31}^2}{1+e^{-2w_{11}^1-4w_{12}^1-3w_{13}^1-b_1^1}}-\frac{w_{32}^2}{1+e^{-2w_{21}^1-4w_{22}^1-3w_{23}^1-b_2^1}}-b_3^2}} + b_1^3$$

$$\mathrm{N}(\mathbf{x}_2) = \frac{w_{11}^3}{1+e^{-\frac{w_{11}^2}{1+e^{-w_{11}^1-5w_{13}^1-b_1^1}}-\frac{w_{12}^2}{1+e^{-w_{21}^1-5w_{23}^1-b_2^1}}-b_1^2}} + \frac{w_{12}^3}{1+e^{-\frac{w_{21}^2}{1+e^{-w_{11}^1-5w_{13}^1-b_1^1}}-\frac{w_{22}^2}{1+e^{-w_{21}^1-5w_{23}^1-b_2^1}}-b_2^2}} + \frac{w_{13}^3}{1+e^{-\frac{w_{31}^2}{1+e^{-w_{11}^1-5w_{13}^1-b_1^1}}-\frac{w_{32}^2}{1+e^{-w_{21}^1-5w_{23}^1-b_2^1}}-b_3^2}} + b_1^3$$

Thus, the loss is as follows:

$$\begin{aligned}
&\mathcal{R}_{L^2}(w_{11}^1, w_{12}^1, w_{21}^1, w_{22}^1, w_{31}^1, w_{32}^1, b_1^1, b_2^1, w_{11}^2, w_{12}^2, w_{13}^2, w_{21}^2, w_{22}^2, w_{23}^2, b_1^2, b_2^2, b_3^2, w_{11}^3, w_{21}^3, w_{31}^3, b_1^3) \\
&\quad = \frac{1}{2}\left(\frac{w_{11}^3}{1+e^{-\frac{w_{11}^2}{1+e^{-2w_{11}^1-4w_{12}^1-3w_{13}^1-b_1^1}}-\frac{w_{12}^2}{1+e^{-2w_{21}^1-4w_{22}^1-3w_{23}^1-b_2^1}}-b_1^2}} + \frac{w_{12}^3}{1+e^{-\frac{w_{21}^2}{1+e^{-2w_{11}^1-4w_{12}^1-3w_{13}^1-b_1^1}}-\frac{w_{22}^2}{1+e^{-2w_{21}^1-4w_{22}^1-3w_{23}^1-b_2^1}}-b_2^2}} + \frac{w_{13}^3}{1+e^{-\frac{w_{31}^2}{1+e^{-2w_{11}^1-4w_{12}^1-3w_{13}^1-b_1^1}}-\frac{w_{32}^2}{1+e^{-2w_{21}^1-4w_{22}^1-3w_{23}^1-b_2^1}}-b_3^2}} + b_1^3 - 5\right)^2 \\
&\qquad + \frac{1}{2}\left(\frac{w_{11}^3}{1+e^{-\frac{w_{11}^2}{1+e^{-w_{11}^1-5w_{13}^1-b_1^1}}-\frac{w_{12}^2}{1+e^{-w_{21}^1-5w_{23}^1-b_2^1}}-b_1^2}} + \frac{w_{12}^3}{1+e^{-\frac{w_{21}^2}{1+e^{-w_{11}^1-5w_{13}^1-b_1^1}}-\frac{w_{22}^2}{1+e^{-w_{21}^1-5w_{23}^1-b_2^1}}-b_2^2}} + \frac{w_{13}^3}{1+e^{-\frac{w_{31}^2}{1+e^{-w_{11}^1-5w_{13}^1-b_1^1}}-\frac{w_{32}^2}{1+e^{-w_{21}^1-5w_{23}^1-b_2^1}}-b_3^2}} + b_1^3 + 1\right)^2
\end{aligned}$$

Of course, in practice, one doesn't use such giant, rather ugly, equations. Instead, one computes the loss in a step-by-step manner. We are explicitly writing out these equations only to illustrate that the loss really is just an ordinary, large but conceptually simple, multivariate function of the neural network parameters.

### 3.2 Classification Problems: the 0-1 Loss and the Cross-entropy Loss

Now suppose instead that we have a classification problem, where we wish to learn a correspondence $f\colon X \to Y$, where $Y$ is some discrete subset of $\mathbb{R}$. For simplicity, let us say that we have a binary classification problem, so that we can taken $Y$ to be $\{0,1\}$. (The more general classification case is very similar.). As before, let us fix an architecture for our artificial neural networks, yielding a class of potential predictor functions $\{\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}\}$. Now, any potential neural network function $\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}$ will in general output a continuous range of real values, whereas $f$, and a predictor for our classification problem, must output only values in $\{0,1\}$. For each possible choice of the parameters $w_{ij}^l$ and $b_i^l$, we take the associated potential predictor function to be, not $\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}$ itself, but rather $\mathrm{h} \circ \mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}$, where h is the Heaviside step function, which outputs 1 for non-negative inputs and 0 for negative inputs. Thus, for each possible choice of the parameters $w_{ij}^l$ and $b_i^l$, we are constructing a potential predictor as follows: given an input $\mathbf{x} \in \mathbb{R}^n$, consider the associated artificial neural network function $\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}$; if this function outputs a non-negative value, give $\mathbf{x}$ the label 1, if it outputs a negative value, given $x$ the label 0.

What about the loss function? One possible choice is the *0-1 loss*:

$$\mathcal{R}_{0\text{-}1}(\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}) := \sum_{s=1}^{S} \delta(\mathrm{h}(\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}(\mathbf{x}_s)), y_s)$$

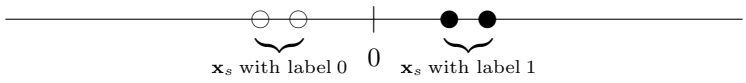
For brevity, for each $s = 1,\ldots,S$, one often sets $\widehat{y}_s := \mathrm{h}(\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}(\mathbf{x}_s))$, and then the loss function may be written as $\sum_{s=1}^{S} \delta(\widehat{y}_s, y_s)$. Here, for any inputs $a, b$, $\delta(a,b)$ is 1 if $a = b$, and 0 otherwise. Thus this loss function simply counts how many times the potential predictor function is correct on the given training sample. It is a very intuitive loss function, though not ideal for the purposes of optimization as it is not smooth with respect to the parameters $w_{ij}^l$ and $b_i^l$.

Another possible choice for the loss function, the *cross-entropy loss*, named as such due to a relation to the notion of cross-entropy in information theory, is as follows:

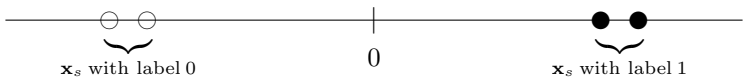
$$\mathcal{R}_{\mathrm{ce}}(\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}) := -\tfrac{1}{S}\sum_{s=1}^{S}\left[y_s \log\left(\sigma(\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}(\mathbf{x}_s))\right) + (1-y_s)\log\left(1-\sigma(\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}(\mathbf{x}_s))\right)\right]$$

Here $\sigma$ is the sigmoid function. For brevity, for each $s = 1,\ldots,S$, set $\widehat{p}_s = \sigma(\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}(\mathbf{x}_s))$ (we think of $\widehat{p}_s$ as an output probability). Then the loss function may be written as $(-1/S)\sum_{s=1}^{S}(y_s \log(\widehat{p}_s) + (1-y_s)\log(1-\widehat{p}_s))$.

At first sight, this may look like a strange loss function. The motivation for it is as follows. For our classifcation function, we have taken $\mathrm{h} \circ \mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}$, so that, for a given input $\mathbf{x}$, if the neural network function $\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}$ outputs a positive value (or zero), we label $\mathbf{x}$ by 1, whereas if $\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}$ outputs a negative value, we label $\mathbf{x}$ by 0. Rather than interpreting the outputs of the neural network function in a black and white manner, we can also consider the outputs as a spectrum of possibilities on the real line. Thus, if $\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}(\mathbf{x})$ is very large and positive, we interpret this as $\mathbf{x}$ being very likely to have the label 1, whereas if $\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}(\mathbf{x})$ is very small and positive, we think that there is a slightly greater than fifty-percent chance that $\mathbf{x}$ has the label 1. In short, we can sum this up as that $\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}$ ought to provide, not the labels, but rather probabilities for the labels. Note that then, when we optimize the parameters, it is this capability of the neural network that we will be optimizing, so that, for example, a neural network which separates the input training samples as follows




will have a larger associated cross-entropy loss than one which separates the input samples as follows




though both will have equivalent 0-1 loss.

We said that our neural network function $\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}$ ought to give probabilities. However, as things stand, we don't actually have probabilities, but rather output values on the real line $\mathbb{R}$. We make the probabilistic interpretation precise as follows. Given any input $\mathbf{x} \in \mathbb{R}^n$, we stipulate that $\sigma(\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}(\mathbf{x}))$ ought to give the probability of $\mathbf{x}$ having the label 1. Thus, we are really estimating a probability distribution now, as opposed to just the labels. Note that while $\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}$ maps into $\mathbb{R}$, $\sigma \circ \mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}$ maps into $(0,1)$, so that the probabilistic interpretation is valid. Moreover, $\sigma(\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}(\mathbf{x})) \geq 1/2$ exactly when $\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}(\mathbf{x}) \geq 0$, so that, according to our deterministic classifier $\mathrm{h} \circ \mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}$, $\mathbf{x}$ receives the label 1 exactly when, according to our probabilistic classifier, the probability of $\mathbf{x}$ having the label 1 is at least $1/2$. That is, the two classifiers, or the two interpretations of the neural network function outputs, are compatible.

Now, we can finally get to the actual loss function. Consider an arbitrary neural network, of the specified architecture, and the associated function $\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}$. Given any $\mathbf{x} \in \mathbb{R}^n$, according to the probabilistic interpretation, the probability that the associated label is 1 is $\sigma(\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}(\mathbf{x}))$, and so also the probability that the label is 0 is $1 - \sigma(\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}(\mathbf{x}))$. We can sum this up as:

$$\Pr(\text{label of } \mathbf{x} = y) = [\sigma(\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}(\mathbf{x}))]^y [1 - \sigma(\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}(\mathbf{x}))]^{1-y}$$

(Just check the two possible cases, $y = 0$ and $y = 1$, to see that this is correct.) For each of $\mathbf{x}_1, \ldots, \mathbf{x}_S$, we have already observed the labels, namely, $y_1, \ldots, y_S$. According to the above probabilistic interpretation, for each $s = 1, \ldots, S$, the probability that $\mathbf{x}_s$ has the label $y_s$ which we have already observed is

$$\Pr(\text{label of } \mathbf{x}_s = y_s) = [\sigma(\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}(\mathbf{x}_i))]^{y_s} [1 - \sigma(\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}(\mathbf{x}_i))]^{1-y_s}$$

As such, according to the probabilistic classifier derived from $\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}$, the probability of observing the labels $y_1, \ldots, y_S$ which have already been observed, is:

$$\prod_{s=1}^{S} [\sigma(\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}(\mathbf{x}_i))]^{y_s} [1 - \sigma(\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}(\mathbf{x}_i))]^{1-y_s}$$

This quantity can be taken as a measure of how well the classifier associated to the neural network function $\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}$ classifies. It is a quantity which we would wish to maximize. Our actual loss function results from applying $-\frac{1}{S}\log(-)$ to this quantity; we do this because, computationally, sums are easier to deal with than products, and because upon doing so we get a convex function. This completes the justification for the cross-entropy loss.

Note the following:

- The procedure we have outlined above to pass from the probabilistic interpretation to the loss function is an instance of *maximum likelihood estimation*, a common technique in probability density estimation.
- We won't do so here, but just as with the $L^2$ loss above, it is clear that we can explicitly flesh out and write out the 0-1 loss and the cross-entropy loss as functions of the neural network parameters (in the case of the 0-1 loss, the delta function notation of course cannot be fleshed out in terms of any other functions).

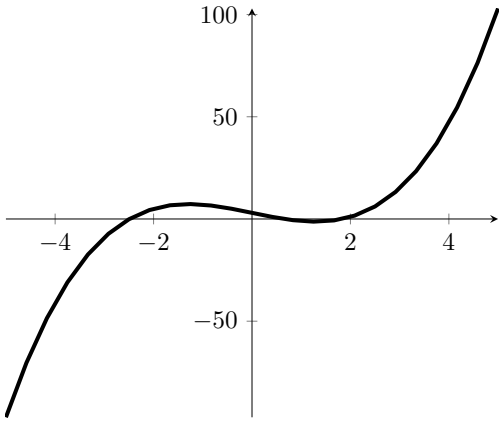
# 4 Gradient Descent

Let us return again to the task of learning a correspondence $f\colon X \to Y$. So far, having fixed an artificial neural network architecture, we have formed a parametric class of functions, and we have also formed loss functions which measure, for arbitrary values of the parameters, how well the corresponding putative predictor predicts. It remains to optimize these loss functions. That is, we must find values of the parameters that minimize these loss functions.

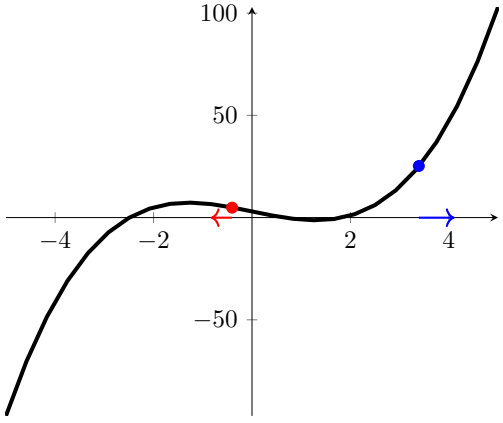
Prior to the discussion of how to optimize our loss functions, let us first consider gradient descent in the general context of wanting to minimize any smooth function $F\colon \mathbb{R}^n \to \mathbb{R}$, for some $n \geq 1$.

## 4.1 Generalities from Multivariate Calculus

We need to recall the notion of the *derivative* of a multivariate smooth function. There are multiple possible interpretations of the derivative (in the case where $n = 1$, and even more so when $n > 1$); we shall present the one which is the most useful for us in the present context. First, consider the case where $n = 1$. We can visualize $F$ as a curve in the cartesian plane:



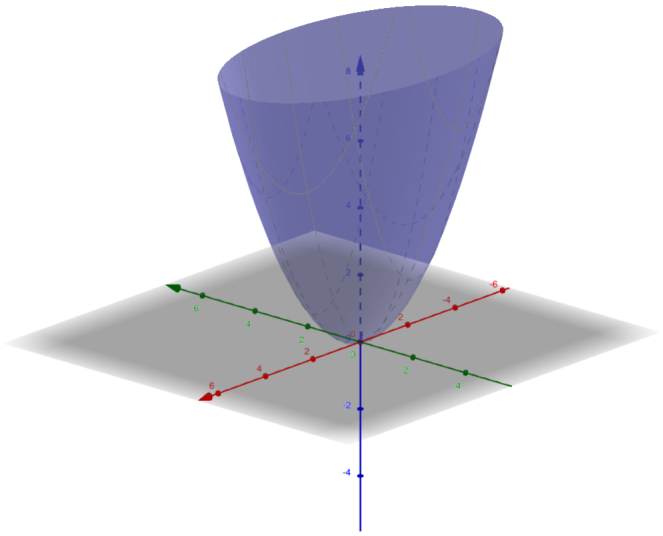

Here, the horizontal axis carries the inputs to $F$ and the vertical axis the outputs of $F$. In this case where $n = 1$, the derivative of $F\colon \mathbb{R} \to \mathbb{R}$ is typically denoted by $F'$ and is once again a function $\mathbb{R} \to \mathbb{R}$. The source copy of $\mathbb{R}$ for $F'$ is to be thought of as the same source copy of $\mathbb{R}$ for $F$. The output copy though is not the same as the output copy of $F$, and in fact we do not think of the outputs of $F'$ as scalars, but rather as one-dimensional vectors. Thus, we think of $F'$ as attaching a one-dimensional vector to each point in the domain of $F$. We might visualize this as follows:



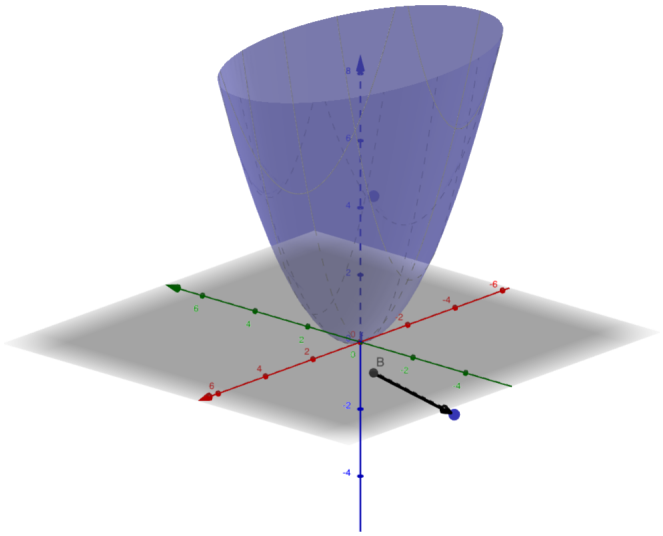

Fix some $a$ in the domain of $F$. The derivative vector $F'(a)$ has the following interpretation: the direction of $F'(a)$ corresponds to the direction, left or right, in which we should move along the horizontal axis, so as

to increase the output, in the vertical axis, of $F$ (if neither direction increases the value, $F'(a)$ will be the zero vector); the magnitude of $F'(a)$ represents the rate of change of the output, or height, with respect to the input, which is to say that, if, in the first figure above, we move, along the horizontal axis, from $a$ to $a + \varepsilon$ for some small $\varepsilon > 0$, we find that the output moves, in the vertical axis, from $F(a)$ to approximately $F(a) + F'(a) \cdot \varepsilon$ (and this approximation is more and more accurate as $\varepsilon \to 0$).

Now let us consider the case where $n = 2$. We can visualize $F$ as a surface in cartesian space as follows:



Here, the horizontal plane carries the inputs to $F$ and the vertical axis the outputs of $F$. In this case where $n = 2$, and more generally when $n > 1$, the derivative, or *gradient* as it is often called, is denoted by $\nabla F$, and is computed via the formula $\nabla F = (\mathrm{D}_1 F, \ldots, \mathrm{D}_n F)$, where $\mathrm{D}_i F$ denotes the $i^{\text{th}}$ partial derivative. Now, when $n = 2$, this is a function $\mathbb{R}^2 \to \mathbb{R}^2$ (and more generally, it is a function $\mathbb{R}^n \to \mathbb{R}^n$). As in the case where $n = 1$, here, the source copy of $\mathbb{R}^2$ is the same source copy of $\mathbb{R}^2$ for $F$. The output copy though is something different, and we think of the outputs as two-dimensional vectors. Thus, we think of $\nabla F$ as attaching a two-dimensional vector to each point in the domain of $F$. We might visualize this as follows:



Fix some point $(a, b)$ in the domain of $F$. The derivative vector $(\nabla F)(a, b)$ has the following interpretation: the direction of $(\nabla F)(a, b)$ corresponds to the direction in which we should move, in the horizontal plane, so as to maximally increase the output, the height in the vertical axis, of $F$ (if no such direction exists, $(\nabla F)(a, b)$ will be the zero vector); the magnitude of $(\nabla F)(a, b)$ represents the rate of change of the output, the height, with respect to the input, which is to say that, if, in the figure above, we move, in the horizontal plane, from $(a, b)$ to the point $\varepsilon$ units away in the direction of $(\nabla F)(a, b)$, for some small $\varepsilon > 0$, we find that the output height moves, in the vertical axis, from $F(a, b)$ to approximately $F(a, b) + (\nabla F)(a, b) \cdot \varepsilon$ (and this approximation is more and more accurate as $\varepsilon \to 0$).

Finally, let us consider the case of a general $n \geq 1$. Though we cannot physically plot the graph of $F$ in this case, the same intuitions carry over. The derivative is a function $\nabla F$ from $\mathbb{R}^n \to \mathbb{R}^n$, and, given $(a_1, \ldots, a_n)$ in the domain of $F$, $(\nabla F)(a_1, \ldots, a_n)$ is a vector attached to the point $(a_1, \ldots, a_n)$ in the domain, the direction of which represents the direction in the $n$-dimensional domain sapce in which we should move so as to maximally increase the output, the "height" (the $(n+1)^{\text{th}}$ coordinate in the $(n+1)$-dimensional cartesian space), and the magnitude of which represents the rate of the change of the "height" with respect to motion in the domain in this direction.

## 4.2 The Gradient Descent Algorithm

Now, recall that we have a goal of minimizing a smooth function $F \colon \mathbb{R}^n \to \mathbb{R}$, for some $n \geq 1$. The algorithm for this, *gradient descent*, is as follows:

GD0: Fix some scalar $\lambda \in \mathbb{R}_{>0}$.

GD1: Fix some input position $\mathbf{p} := (p_1, \ldots, p_n) \in \mathbb{R}^n$.

GD2: Compute the gradient vector $\mathbf{g} := (\nabla F)(\mathbf{p})$ at the current position $\mathbf{p}$.

GD3: Move in the direction of the gradient, by setting $\mathbf{p} := \mathbf{p} - \lambda(\nabla F)(\mathbf{p})$.

GD4: Return to GD1, replacing the original position with the new position.

The idea behind this algorithm is as follows. Given any current position $\mathbf{p} \in \mathbb{R}^n$, we know from our earlier discussion of higher dimensional derivatives that the direction of $(\nabla F)(\mathbf{p})$ is the direction in the domain in which the output of $F$ increases maximally. As a result, the opposite direction, that of $-(\nabla F)(\mathbf{p})$, is the direction in the domain in which the output of $F$ decreases maximally. Thus, as we wish to find a minimum value for the ouput, we move in this opposite direction; this is where the name "gradient descent" comes from. Though we know that we ought to move in the direction of $-(\nabla F)(\mathbf{p})$, we do not know exactly how large a step to take in this direction. The size of this step is dependent on what value we choose for $\lambda$ (and also the magnitude of the gradient). We cannot know in advance the optimal value for this constant, rather, we must simply experiment with various values. Because, by taking successive steps in various directions, we are, in a sense, learning which inputs yield a smaller output, the parameter $\lambda$, which controls how large each step is, is often referred to as the *learning rate*.

Note the following:

- We have not provided any guarantees for the above gradient descent algorithm. It is certainly possible to get stuck in infinite loops, or very slow descent, with poor choices of the parameter $\lambda$ or of the initial position $\mathbf{p}$. However, with sufficient experimentation, one can avoid this in practice.
- One could also get stuck in local minima which are not global minima. However, if one has a large dimensional input space, that is, if $n$ is large, which is often the case in practice, the likelihood of many local minima is not high. This is because, to have a local minimum, every single partial derivative needs to have the same sign (more specifically, be positive), and similarly for local maxima, so that if there are many inputs, saddle points, rather than local minima or maxima, tend to proliferate, and gradient descent can always escape saddle points along the directions with negative partial derivatives.

- Even in the low dimensional cases, at least when $F$ is convex, there exist theoretical guarantees for the convergence of gradient descent.

# 5 Training Artificial Neural Networks

Let us return now for the final time to the idea of constructing a predictor function $f_{\text{pred}}$ for a given correspondence $f\colon X \to Y$, where $X$ is a subset of $\mathbb{R}^n$ for some $n \geq 1$ and $Y$ is some subset of $\mathbb{R}$. To do so, as mentioned before, in deep learning, we begin by fixing an artificial neural network architecture, which provides us with a parametric class $\{\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}\}$ of functions $\mathbb{R}^n \to \mathbb{R}$. We wish to construct a predictor function $f_{\text{pred}}$ using this class. In particular, we saw above that in the case of regression problems, we take $f_{\text{pred}}$ to be exactly one of these neural network functions $\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}$, whereas in the case of classification problems, we takae $f_{\text{pred}}$ to be one of the functions $\mathrm{h} \circ \mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}$, where h is the Heaviside step function. In either case, we ask ourselves, given these possible choices, what is the best possible choice for the parameters $w_{ij}^l$ and $b_i^l$? Above, in either case, we constructed a loss function $\mathcal{R}(\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i})$ which, for a given choice of values for the parameters $w_{ij}^l$ and $b_i^l$, measured how well the corresponding predictor function function approximated $f$. In particular, the smaller the value of the loss, the better the predictor. As such, we are left with the task of minimizing the loss function. This is a smooth function $\mathbb{R}^P \to \mathbb{R}$, where $P$ denotes the total number of the parameters $w_{ij}^l$ and $b_i^l$. We have seen above how one can minimize of such functions, via gradient descent.

Whether we have a regression problem or a classification problem, note that the corresponding loss function is composed of the terms $\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}(\mathbf{x}_s)$, where $\mathbf{x}_s$ varies over the training sample inputs $\mathbf{x}_1,\ldots,\mathbf{x}_S$ (note that, for each $s = 1,\ldots,S$, the term $\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}(\mathbf{x}_s)$ is a function of the parameters $w_{ij}^l$ and $b_i^l$). Thus, to perform gradient descent on $\mathcal{R}(\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i})$, and in particular to compute the gradient of the loss function, we mostly just need to compute the gradients of the terms $\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}(\mathbf{x}_s)$. Such gradients can be computed via algorithms known as forward-propagation and back-propagation.

## 5.1 Forward-propagation and Back-propagation

Fix some neural network architecture, and an associated general neural network function $\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}$. Fix also some vector $\mathbf{x} = (x_1,\ldots,x_n) \in \mathbb{R}^n$. Consider the output $\widehat{y} := \mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}(\mathbf{x})$ as a function of the network parameters $w_{ij}^l$ and $b_i^l$. Suppose given some initial values for these network parameters. Our goal is to determine the value of the gradient $\nabla\widehat{y}$, or equivalently the values of the partial derivatives $\mathrm{D}_{w_{ij}^l}\widehat{y}$ and $\mathrm{D}_{b_i^l}\widehat{y}$, at the given values of the network parameters.

In order to do this, we shall introduce some convenient variables and notations (for these, it is helpful to keep in mind that the layers are indexed by $0,1,\ldots,L$, while the linear maps in between are indexed by $0,1,\ldots,L-1$):

- For $l = 1,\ldots,L-1$, let $\mathrm{Z}^l = [z_i^l]$ denote the pre-activated input at the $l^{\text{th}}$ layer; thus $\mathrm{Z}^1 = \mathrm{W}^0\mathbf{x} + \mathrm{B}^0$ and $\mathrm{Z}^l = \mathrm{W}^{l-1}a(\mathrm{Z}^{l-1}) + \mathrm{B}^{l-1}$ for $l = 2,\ldots L-1$. Let us also, for the same $l = 1,\ldots,L-1$, let $\mathrm{A}^l$ denote the activated input at the $l^{\text{th}}$ layer, namely $a(\mathrm{Z}^l)$. Note that $\widehat{y}$ may be considered a function of the variables in $\mathrm{W}^{L-1},\mathrm{B}^{L-1}$ and $\mathrm{A}^{L-1}$, via $\widehat{y} = \mathrm{W}^{L-1}\mathrm{A}^{L-1} + \mathrm{B}^{L-1}$, and we have $\mathrm{A}^{L-1} = a(\mathrm{Z}^{L-1})$. Similarly, for $l = 2,\ldots,L-1$, $\mathrm{Z}^l$ may be considered a function of the variables in $\mathrm{W}^{l-1},\mathrm{B}^{l-1}$ and $\mathrm{A}^{l-1}$ via $\mathrm{Z}^l = \mathrm{W}^{l-1}\mathrm{A}^{l-1} + \mathrm{B}^{l-1}$, and we have $\mathrm{A}^{l-1} = a(\mathrm{Z}^{l-1})$. Finally, $\mathrm{Z}^1$ may be considered a function of the variables in $\mathrm{W}^0$ and $\mathrm{B}^0$ via $\mathrm{Z}^1 = \mathrm{W}^0\mathbf{x} + \mathrm{B}^0$. These relations will allow us to inductively compute the desired derivatives.

- We let $\mathrm{D}_{\mathrm{W}^l}\widehat{y}$ denote the matrix where the $(i,j)$-entry is $\mathrm{D}_{w_{ji}^l}\widehat{y}$ (note that we switch around the $i,j$; this ensures that we are consistent with the convention for functions $\mathbb{R}^n \to \mathbb{R}$ that vectors in the source $\mathbb{R}^n$ are taken to be column vectors, whereas the corresponding gradients are row vectors). Similarly, for a fixed $l$, let us let $\mathrm{D}_{\mathrm{B}^l}\widehat{y}$ denote the row vector of the partial derivatives $\mathrm{D}_{b_i^l}\widehat{y}$, varying over $i$. We

also let $\mathrm{D}_{\mathrm{A}^l}\widehat{y}$ and $\mathrm{D}_{\mathrm{Z}^l}\widehat{y}$ denote the row vectors of the partial derivatives $\mathrm{D}_{a_i^l}\widehat{y}$ and $\mathrm{D}_{z_i^l}\widehat{y}$, respectively. Our goal is then to compute $\mathrm{D}_{\mathrm{W}^l}\widehat{y}$ and $\mathrm{D}_{\mathrm{B}^l}\widehat{y}$ for each $l = 0, \ldots, L-1$.

We have independent variables $w_{ij}^l$, $b_i^l$, and we are given initial values for these variables. We also have the dependent variable $\widehat{y}$, and have also introduced intermediate variables $z_i^l$ and $a_i^l$. Given the values for the network parameters $w_{ij}^l$, $b_i^l$, we can clearly pass through the network in a forward manner to compute corresponding values for the $z_i^l$, $a_i^l$ and $\widehat{y}$. Suppose then that these values have been computed. This computation of the values of these variables, from the initial values of the network parameters, is what we call *forward-propagation*.

Now, we are given some initial values for $w_{ij}^l$, $b_i^l$ and wish to find $\mathrm{D}_{\mathrm{W}^l}\widehat{y}$ and $\mathrm{D}_{\mathrm{B}^l}\widehat{y}$, for $l = 0, \ldots, L-1$, at these values. As above, we also have values for $\widehat{y}$ and the intermediate variables $z_i^l$, $a_i^l$. We shall now show how, given these values, one can then compute $\mathrm{D}_{\mathrm{W}^l}\widehat{y}$ and $\mathrm{D}_{\mathrm{B}^l}\widehat{Y}$. In fact, we shall find not only these but also $\mathrm{D}_{\mathrm{A}^l}\widehat{y}$ and $\mathrm{D}_{\mathrm{Z}^l}\widehat{y}$ for $l = 1, \ldots, D-1$. First, using $\widehat{y} = \mathrm{W}^{L-1}\mathrm{A}^{L-1} + \mathrm{B}^{L-1}$, an entry-by-entry check shows that:

$$\begin{aligned}
&\mathrm{D}_{\mathrm{W}^{L-1}}\widehat{y} = \mathrm{A}^{L-1} \\
&\mathrm{D}_{\mathrm{B}^{L-1}}\widehat{y} = 1 \\
&\mathrm{D}_{\mathrm{A}^{L-1}}\widehat{y} = \mathrm{W}^{L-1}
\end{aligned}$$

Note that $\mathrm{B}^{L-1}$ is necessarily just a scalar. Note also that values for the righthand sides are known, via the forward-propagation above. Moreover, using $\mathrm{A}^{L-1} = a(\mathrm{Z}^{L-1})$ and the chain rule, an entry-by-entry check shows that:

$$\mathrm{D}_{\mathrm{Z}^{L-1}}\widehat{y} = \mathrm{W}^{L-1} \odot [a'(\mathrm{Z}^{L-1})]^{\mathrm{T}}$$

Here, $\odot$ denotes the Hadamard, or entrywise, product; and $a'(-)$ is also applied entrywise. Also, once again, the value of the righthand side is known from the earlier forward-propagation (so long as we can compute the derivative of the activation). Next, we have that $\mathrm{Z}^{L-1} = \mathrm{W}^{L-2}\mathrm{A}^{L-2} + \mathrm{B}^{L-2}$. Using this and the chain rule, an entry-by-entry checks show that:

$$\begin{aligned}
&\mathrm{D}_{\mathrm{W}^{L-2}}\widehat{y} = \begin{bmatrix} | & & | \\ \mathrm{A}^{L-2} & \cdots & \mathrm{A}^{L-2} \\ | & & | \end{bmatrix} \odot \begin{bmatrix} - & \mathrm{D}_{\mathrm{Z}^{L-1}}\widehat{y} & - \\ & \vdots & \\ - & \mathrm{D}_{\mathrm{Z}^{L-1}}\widehat{y} & - \end{bmatrix} \\
&\mathrm{D}_{\mathrm{B}^{L-2}}\widehat{y} = \mathrm{D}_{\mathrm{Z}^{L-1}}\widehat{y} \\
&\mathrm{D}_{\mathrm{A}^{L-2}}\widehat{y} = (\mathrm{D}_{\mathrm{Z}^{L-1}}\widehat{y})\mathrm{W}^{L-2}
\end{aligned}$$

Moreover, using $\mathrm{A}^{L-2} = a(\mathrm{Z}^{L-2})$ and the chain rule, we have that:

$$\mathrm{D}_{\mathrm{Z}^{L-2}}\widehat{y} = [(\mathrm{D}_{\mathrm{Z}^{L-1}}\widehat{y})\mathrm{W}^{L-2}] \odot [a'(\mathrm{Z}^{L-2})]^{\mathrm{T}}$$

This pattern continues, and so more generally, by more or less the same argument, for $l = 1, \ldots, L-2$, another entry-by-entry check shows that:

$$\begin{aligned}
&\mathrm{D}_{\mathrm{W}^l}\widehat{y} = \begin{bmatrix} | & & | \\ \mathrm{A}^{l} & \cdots & \mathrm{A}^{l} \\ | & & | \end{bmatrix} \odot \begin{bmatrix} - & \mathrm{D}_{\mathrm{Z}^{l+1}}\widehat{y} & - \\ & \vdots & \\ - & \mathrm{D}_{\mathrm{Z}^{l+1}}\widehat{y} & - \end{bmatrix} \\
&\mathrm{D}_{\mathrm{B}^l}\widehat{y} = \mathrm{D}_{\mathrm{Z}^{l+1}}\widehat{y} \\
&\mathrm{D}_{\mathrm{A}^l}\widehat{y} = (\mathrm{D}_{\mathrm{Z}^{l+1}}\widehat{y})\mathrm{W}^{l} \\
&\mathrm{D}_{\mathrm{Z}^l}\widehat{y} = [(\mathrm{D}_{\mathrm{Z}^{l+1}}\widehat{y})\mathrm{W}^{l}] \odot [a'(\mathrm{Z}^{l})]^{\mathrm{T}}
\end{aligned}$$

Finally, we are left with computing $\mathrm{D}_{\mathrm{W}^0}\widehat{y}$ and $\mathrm{D}_{\mathrm{B}^0}\widehat{y}$. Using $\mathrm{Z}^1 = \mathrm{W}^0\mathbf{x} + \mathrm{B}^0$, these are as follows:

$$\mathrm{D}_{\mathrm{W}^0}\widehat{y} = \begin{bmatrix} | & & | \\ \mathbf{x} & \cdots & \mathbf{x} \\ | & & | \end{bmatrix} \odot \begin{bmatrix} - & \mathrm{D}_{\mathrm{Z}^1}\widehat{y} & - \\ & \vdots & \\ - & \mathrm{D}_{\mathrm{Z}^1}\widehat{y} & - \end{bmatrix}$$

$$\mathrm{D}_{\mathrm{B}^0}\widehat{y} = \mathrm{D}_{\mathrm{Z}^1}\widehat{y}$$

We have thus successfully computed all the necessary partial derivatives to construct the desired gradient $\nabla\widehat{y}$. Note that, if we set $\mathrm{A}^0 = \mathbf{x}$ and $\mathrm{Z}^L = \widehat{y}$, we can sum up the above computations as follows:

BP1: Set $\mathrm{D}_{\mathrm{Z}^L}\widehat{y} := 1$.

BP2: For $l = 0, \ldots, L-1$, set:

$$\mathrm{D}_{\mathrm{W}^l}\widehat{y} := \begin{bmatrix} | & & | \\ \mathrm{A}^l & \cdots & \mathrm{A}^l \\ | & & | \end{bmatrix} \odot \begin{bmatrix} - & \mathrm{D}_{\mathrm{Z}^{l+1}}\widehat{y} & - \\ & \vdots & \\ - & \mathrm{D}_{\mathrm{Z}^{l+1}}\widehat{y} & - \end{bmatrix}$$

$$\mathrm{D}_{\mathrm{B}^l}\widehat{y} := \mathrm{D}_{\mathrm{Z}^{l+1}}\widehat{y}$$

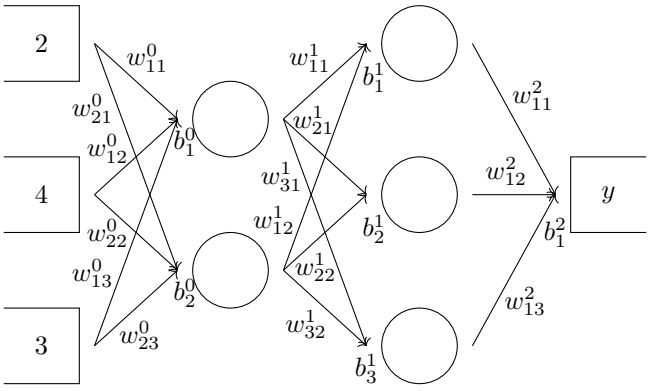
Break here if $l = 0$.

$$\mathrm{D}_{\mathrm{A}^l}\widehat{y} := (\mathrm{D}_{\mathrm{Z}^{l+1}}\widehat{y})\mathrm{W}^l$$

$$\mathrm{D}_{\mathrm{Z}^l}\widehat{y} := [(\mathrm{D}_{\mathrm{Z}^{l+1}}\widehat{y})\mathrm{W}^l] \odot [a'(\mathrm{Z}^l)]^{\mathrm{T}}$$

As we begin with partial derivatives with respect to variables in the final layer, and proceed backwards to get partial derivatives with respect to variables in earlier layers, this algorithm is referred to as *back-propagation*.

Let us consider an example. We'll consider artificial neural networks where $a = \sigma$ and where the remainder of the architecture is as follows:




As indicated, we take the sample input to be $\mathbf{x} = (x_1, x_2, x_3) = (2, 4, 3)$. We'll denote the neural network function by simply N. We are interested in $\widehat{y} = \mathrm{N}(2, 4, 3)$ as a function of the network parameters $w^l_{ij}$, $b^l_i$.

Recall from earlier that this is the following function:

$$\frac{w_{11}^3}{1+e^{-\frac{w_{11}^2}{1+e^{-2w_{11}^1-4w_{12}^1-3w_{13}^1-b_1^1}}-\frac{w_{12}^2}{1+e^{-2w_{21}^1-4w_{22}^1-3w_{23}^1-b_2^1}}-b_1^2}} + \frac{w_{12}^3}{1+e^{-\frac{w_{21}^2}{1+e^{-2w_{11}^1-4w_{12}^1-3w_{13}^1-b_1^1}}-\frac{w_{22}^2}{1+e^{-2w_{21}^1-4w_{22}^1-3w_{23}^1-b_2^1}}-b_2^2}} + \frac{w_{13}^3}{1+e^{-\frac{w_{31}^2}{1+e^{-2w_{11}^1-4w_{12}^1-3w_{13}^1-b_1^1}}-\frac{w_{32}^2}{1+e^{-2w_{21}^1-4w_{22}^1-3w_{23}^1-b_2^1}}-b_3^2}} + b_1^3$$

In particular, we wish to compute the partial derivatives $\mathrm{D}_{w_{ij}^l}(\mathrm{N}(2,3,4))$, $\mathrm{D}_{b_i^l}(\mathrm{N}(2,3,4))$ at some specified values for the network parameters. Though we can in principle derive analytic expressions for the partial derivatives, it is of course not desirable to work with such a large unwieldy expression. Rather, we will use the step-by-step algorithms described above.

Now, let us suppose that the network parameters are initialized as follows: each $w_{ij}^l$ is set to 1, while each $b_i^l$ is set to 0. We wish to compute the partial derivatives $\mathrm{D}_{w_{ij}^l}(\mathrm{N}(2,3,4))$, $\mathrm{D}_{b_i^l}(\mathrm{N}(2,3,4))$ at these initial values for the network parameters. As above, first, we forward-propagate. Upon doing so, we find the following:

$$z_1^1 = z_2^1 = 9$$
$$a_1^1 = a_2^1 \approx 0.99987660542$$
$$z_1^2 = z_2^2 = z_3^2 \approx 1.99975321085$$
$$a_1^2 = a_2^2 = a_3^2 \approx 0.88077116426$$
$$\widehat{y} \approx 2.6423134928$$

Next, we back-propagate. Upon doing so, we find the following:

$$\mathrm{D}_{\mathrm{Z}^3}\widehat{y} = 1$$

$$\begin{bmatrix} \mathrm{D}_{w_{11}^2}\widehat{y} \\ \mathrm{D}_{w_{12}^2}\widehat{y} \\ \mathrm{D}_{w_{13}^2}\widehat{y} \end{bmatrix} = \mathrm{D}_{\mathrm{W}^2}\widehat{y} = \mathrm{A}^2 \approx \begin{bmatrix} 0.88077116426 \\ 0.88077116426 \\ 0.88077116426 \end{bmatrix}$$

$$\mathrm{D}_{b_1^2}\widehat{y} = \mathrm{D}_{\mathrm{B}^2}\widehat{y} = \mathrm{D}_{\mathrm{Z}^3}\widehat{y} = 1$$

$$\begin{bmatrix} \mathrm{D}_{a_{11}^2}\widehat{y} & \mathrm{D}_{a_{12}^2}\widehat{y} & \mathrm{D}_{a_{13}^2}\widehat{y} \end{bmatrix} = \mathrm{D}_{\mathrm{A}^2}\widehat{y} = \mathrm{W}^2 = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \mathrm{D}_{z_{11}^2}\widehat{y} & \mathrm{D}_{z_{12}^2}\widehat{y} & \mathrm{D}_{z_{13}^2}\widehat{y} \end{bmatrix} = \mathrm{D}_{\mathrm{Z}^2}\widehat{y} = \mathrm{W}^2 \odot [a'(\mathrm{Z}^2)]^{\mathrm{T}}$$
$$\approx \begin{bmatrix} \sigma'(1.99975321085) & \sigma'(1.99975321085) & \sigma'(1.99975321085) \end{bmatrix}$$
$$\approx \begin{bmatrix} 0.10501332046 & 0.10501332046 & 0.10501332046 \end{bmatrix}$$

$$\begin{bmatrix} \mathrm{D}_{w_{11}^1}\widehat{y} & \mathrm{D}_{w_{21}^1}\widehat{y} & \mathrm{D}_{w_{31}^1}\widehat{y} \\ \mathrm{D}_{w_{12}^1}\widehat{y} & \mathrm{D}_{w_{22}^1}\widehat{y} & \mathrm{D}_{w_{32}^1}\widehat{y} \end{bmatrix} = \mathrm{D}_{\mathrm{W}^1}\widehat{y} = \begin{bmatrix} | & | & | \\ \mathrm{A}^1 & \mathrm{A}^1 & \mathrm{A}^l \\ | & | & | \end{bmatrix} \odot \begin{bmatrix} - & \mathrm{D}_{\mathrm{Z}^2}\widehat{y} & - \\ - & \mathrm{D}_{\mathrm{Z}^2}\widehat{y} & - \end{bmatrix}$$

$$\approx \begin{bmatrix} 0.99987660542 & 0.99987660542 & 0.99987660542 \\ 0.99987660542 & 0.99987660542 & 0.99987660542 \end{bmatrix} \odot \begin{bmatrix} 0.10501332046 & 0.10501332046 & 0.10501332046 \\ 0.10501332046 & 0.10501332046 & 0.10501332046 \end{bmatrix}$$
$$\approx \begin{bmatrix} 0.10500036238 & 0.10500036238 & 0.10500036238 \\ 0.10500036238 & 0.10500036238 & 0.10500036238 \end{bmatrix}$$

$$[\,\mathrm{D}_{b_1^1}\widehat{y}\quad \mathrm{D}_{b_2^1}\widehat{y}\quad \mathrm{D}_{b_3^1}\widehat{y}\,] = \mathrm{D}_{\mathrm{B}^1}\widehat{y} = \mathrm{D}_{\mathrm{Z}^2}\widehat{y} \approx [\,0.10501332046\quad 0.10501332046\quad 0.10501332046\,]$$

$$\begin{aligned}[\,\mathrm{D}_{a_1^1}\widehat{y}\quad \mathrm{D}_{a_2^1}\widehat{y}\,] = \mathrm{D}_{\mathrm{A}^1}\widehat{y} = (\mathrm{D}_{\mathrm{Z}^2}\widehat{y})\mathrm{W}^1 &\approx [\,0.10501332046\quad 0.10501332046\quad 0.10501332046\,]\begin{bmatrix}1 & 1\\ 1 & 1\\ 1 & 1\end{bmatrix}\\ &= [\,0.31503996138\quad 0.31503996138\,]\end{aligned}$$

$$\begin{aligned}[\,\mathrm{D}_{z_1^1}\widehat{y}\quad \mathrm{D}_{z_2^1}\widehat{y}\,] = \mathrm{D}_{\mathrm{Z}^1}\widehat{y} = [(\mathrm{D}_{\mathrm{Z}^2}\widehat{y})\mathrm{W}^l]\odot[a'(\mathrm{Z}^1)]^{\mathrm{T}} &\approx [\,0.31503996138\quad 0.31503996138\,]\big[\,\sigma'(9)\quad \sigma'(9)\,]\\ &\approx [\,0.00003886942\quad 0.00003886942\,]\end{aligned}$$

$$\begin{aligned}\begin{bmatrix}\mathrm{D}_{w_{11}^0}\widehat{y} & \mathrm{D}_{w_{21}^0}\widehat{y}\\ \mathrm{D}_{w_{12}^0}\widehat{y} & \mathrm{D}_{w_{22}^0}\widehat{y}\\ \mathrm{D}_{w_{13}^0}\widehat{y} & \mathrm{D}_{w_{23}^0}\widehat{y}\end{bmatrix} = \mathrm{D}_{\mathrm{W}^0}\widehat{y} &= \begin{bmatrix}| & |\\ \mathrm{A}^0 & \mathrm{A}^0\\ | & |\end{bmatrix}\odot\begin{bmatrix}- & \mathrm{D}_{\mathrm{Z}^1}\widehat{y} & -\\ - & \mathrm{D}_{\mathrm{Z}^1}\widehat{y} & -\\ - & \mathrm{D}_{\mathrm{Z}^1}\widehat{y} & -\end{bmatrix}\\ &\approx \begin{bmatrix}2 & 2\\ 4 & 4\\ 3 & 3\end{bmatrix}\odot\begin{bmatrix}0.00003886942 & 0.00003886942\\ 0.00003886942 & 0.00003886942\\ 0.00003886942 & 0.00003886942\end{bmatrix}\\ &\approx \begin{bmatrix}0.00007773884 & 0.00007773884\\ 0.00015547768 & 0.00015547768\\ 0.00011660826 & 0.00011660826\end{bmatrix}\end{aligned}$$

$$[\,\mathrm{D}_{b_1^0}\widehat{y}\quad \mathrm{D}_{b_2^0}\widehat{y}\,] = \mathrm{D}_{\mathrm{B}^0}\widehat{y} = \mathrm{D}_{\mathrm{Z}^1}\widehat{y} \approx [\,0.00003886942\quad 0.00003886942\,]$$

We have now found all the desired partial derivatives.

## 5.2 Training Artificial Neural Networks

We can now sum up all of our findings to illustrate how one can learn a correspondence $f\colon X \to Y$, where $X$ is a subset of $\mathbb{R}^n$ for some $n \geq 1$ and $Y \subseteq \mathbb{R}$, by training an artificial neural network. The final deep learning algorithm is as follows:

DL1: Collect training data $\mathbf{x}_1, \ldots, \mathbf{x}_S \in \mathbb{R}^n$ and $y_1, \ldots, y_S \in \mathbb{R}$.

DL2: Fix an artificial neural network architecture. Introduce variables $w_{ij}^l$ and $b_i^l$.

DL3: Fix a learning rate $\lambda$.

DL4: Choose initial values for the $w_{ij}^l$ and $b_i^l$.

DL5: For the current values of the $w_{ij}^l$ and $b_i^l$, use forward-propagation to compute $\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}(\mathbf{x}_s)$ for each $s = 1, \ldots, S$. Cache values for the $\mathrm{Z}^l$ and $\mathrm{A}^l$ along the way. Using these computations, compute the current loss $\mathcal{R}(\{w_{ij}^l\}_{l,i,j}, \{b_i^l\}_{l,i})$.

DL6: Use back-propagation to compute the values of the derivatives $\mathrm{D}_{w_{ij}^l}\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}(\mathbf{x}_s)$ and $\mathrm{D}_{b_i^l}\mathrm{N}_{\{w_{ij}^l\}_{l,i,j},\{b_i^l\}_{l,i}}(\mathbf{x}_s)$ for each $s = 1, \ldots, S$ and each $l, i, j$ at the current values of the $w_{ij}^l$ and $b_i^l$. Using these, compute the derivatives $\mathrm{D}_{w_{ij}^l}\mathcal{R}$ and $\mathrm{D}_{b_i^l}\mathcal{R}$ of the cost.

DL7: Update the values of the $w_{ij}^l$ and $b_i^l$ by setting $w_{ij}^l := w_{ij}^l - \lambda\,(\mathrm{D}_{w_{ij}^l}\mathcal{R})$ and $b_i^l := b_i^l - \lambda\,(\mathrm{D}_{b_i^l}\mathcal{R})$.

DL8: Repeat Steps 5 to 7. Repeat for however long is desired. If the loss is not decreasing sufficiently, go back to Step 2 and/or Step 3 and consider adjusting the architecture and/or the learning rate.