Introduction to Neural Networks

Building Blocks: Neurons

A neuron takes inputs, does some math with them, and produces one output.

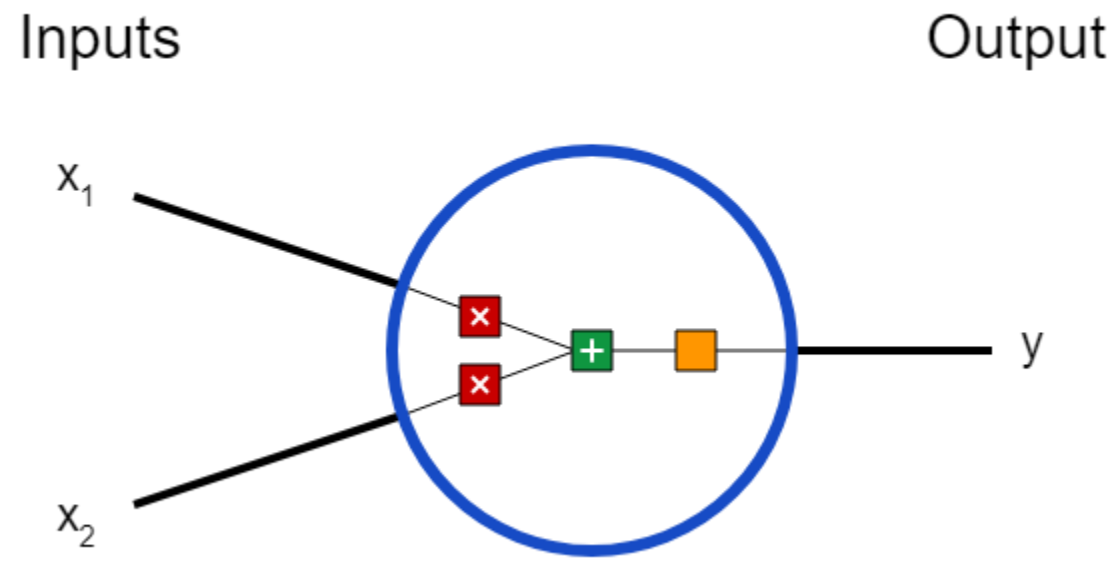


Fig. 2-input neuron

3 things are happening here. First, each input is multiplied by a weight:

$$x_1 \rightarrow x_1 * w_1$$

$$x_2 \rightarrow x_2 * w_2$$

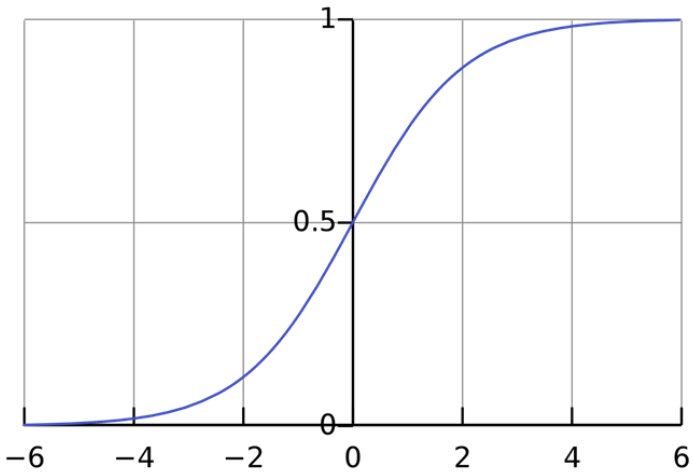
Next, all the weighted inputs are added together with a bias b:

$$(x_1 * w_1) + (x_2 * w_2) + b$$

Finally, the sum is passed through an activation function:

$$y = f(x_1 * w_1 + x_2 * w_2 + b)$$

The activation function is used to turn an unbounded input into an output that has a nice, predictable form. A commonly used activation function is the sigmoid function:



The sigmoid function only outputs numbers in the range (0,1). You can think of it as compressing (−∞,+∞) to (0,1) - big negative numbers become ~0, and big positive numbers become ~1.

$$Sigmoid\ function, f(x) = \frac{1}{1+e^{-x}}$$

A Simple Example

Assume we have a 2-input neuron that uses the sigmoid activation function and has the following parameters: $w = [0,1]\ and\ b = 4$

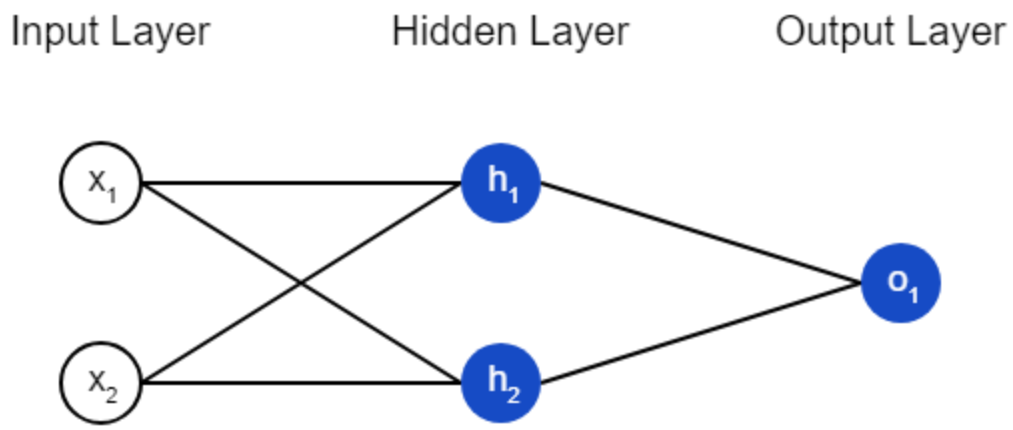
w=[0,1] is just a way of writing $w_1 = 0, w_2 = 1$ in vector form. Now, let's give the neuron an input of *x*=[2,3]. We'll use the dot product to write things more concisely:

$$(w \cdot x) + b = ((w1 * x1) + (w2 * x2)) + b = 0 * 2 + 1 * 3 + 4 = 7$$
$$y = f(w \cdot x + b) = f(7) = 0.999$$

The neuron outputs 0.999 given the inputs *x*=[2,3]. That's it! This process of passing inputs forward to get an output is known as **feedforward**.

Combining Neurons into a Neural Network

A neural network is nothing more than a bunch of neurons connected together. Here's what a simple neural network might look like:



This network has 2 inputs, a hidden layer with 2 neurons ($h1$ and $h2$), and an output layer with 1 neuron ($o1$). Notice that the inputs for $o1$ are the outputs from $h1$ and $h2$ - that's what makes this a network.

An Example: Feedforward

Let's use the network pictured above and assume all neurons have the same weights w= [0,1], the same bias b=0, and the same sigmoid activation function. Let h_1, h_2, o_1 denote the *outputs* of the neurons they represent.

What happens if we pass in the input x= [2,3]?

$$h_1 = h_2 = f(w \cdot x + b) = f((0 * 2) + (1 * 3) + 0) = f(3) = 0.9526$$
$$o_1 = f(w \cdot [h_1, h_2] + b) = f((0 * h_1) + (1 * h_2) + 0) = f(0.9526) = 0.7216$$

The output of the neural network for input x= [2,3] is 0.7216.

A neural network can have **any number of layers** with **any number of neurons** in those layers. The basic idea stays the same: feed the input(s) forward through the neurons in the network to get the output(s) at the end.

Training a Neural Network

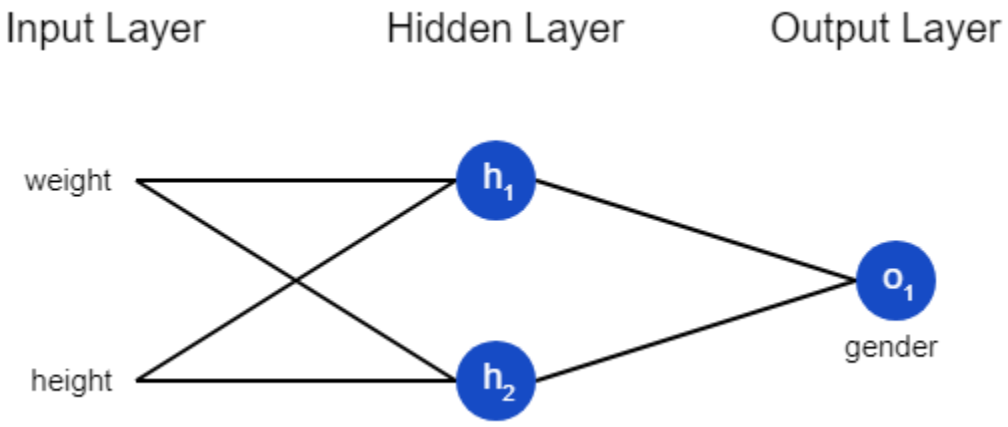
Name	Weight (lb)	Height (in)	Gender
Alice	133	65	F
Bob	160	72	M
Charlie	152	70	M
Diana	120	60	F

Let's train our network to predict someone's gender given their weight and height:

We'll represent Male with a 0 and Female with a 1, and we'll also shift the data to make it easier to use:

Name	Weight (minus 135)	Height (minus 66)	Gender
Alice	-2	-1	1
Bob	25	6	0
Charlie	17	4	0
Diana	-15	-6	1

I arbitrarily chose the shift amounts (135 and 66) to make the numbers look nice. Normally, you'd shift by the mean.



Loss

Before we train our network, we first need a way to quantify how "good" it's doing so that it can try to do "better". That's what the **loss** is.

We'll use the **mean squared error** (MSE) loss:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_{true} - y_{pred})^2$$

Let's break this down:

- n is the number of samples, which is 4 (Alice, Bob, Charlie, Diana).
- y represents the variable being predicted, which is Gender.
- y_{true} is the *true* value of the variable (the "correct answer"). For example, y_{true} for Alice would be 1 (Female).
- y_{pred} is the *predicted* value of the variable. It's whatever our network outputs.

$(y_{true} - y_{pred})^2$ is known as the **squared error**. Our loss function is simply taking the average over all squared errors (hence the name *means* squared error). The better our predictions are, the lower our loss will be!

Better predictions = Lower loss.

Training a network = trying to minimize its loss.

An Example Loss Calculation

Let's say our network always outputs 0 - in other words, it's confident all humans are Male. What would our loss be?

Name	y_{true}	y_{pred}	$(y_{true} - y_{pred})^2$
Alice	1	0	1
Bob	0	0	0
Charlie	0	0	0
Diana	1	0	1

MSE=(1+0+0+1)/4=0.5

Training a Neural Network (Minimize the loss)

We now have a clear goal: minimize the loss of the neural network. We know we can change the network's weights and biases to influence its predictions, but how do we do so in a way that decreases loss?

For simplicity, let's pretend we only have Alice in our dataset:

Name	Weight (minus 135)	Height (minus 66)	Gender
Alice	-2	-1	1

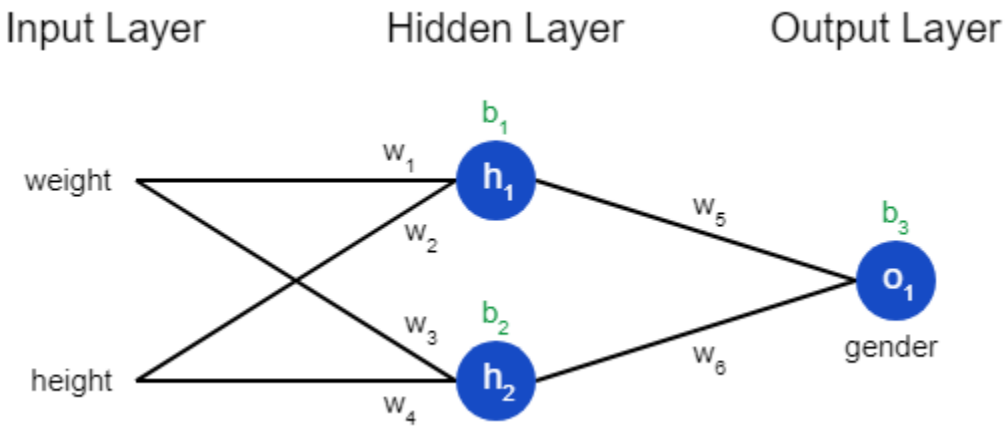
Then the mean squared error loss is just Alice's squared error:

$$MSE = \frac{1}{1}\sum_{i=1}^{n}\left(y_{true} - y_{pred}\right)^2$$

$$= \left(y_{true} - y_{pred}\right)^2$$

$$= (1 - y_{pred})^2$$

Another way to think about loss is as a function of weights and biases. Let's label each weight and bias in our network:



Then, we can write loss as a multivariable function:

$$L(w1, w2, w3, w4, w5, w6, b1, b2, b3)$$

Imagine we wanted to tweak *w*1. How would loss *L* change if we changed *w*1? That's a question the partial derivative $\frac{\delta L}{\delta w_1}$ can answer. How do we calculate it?

To start, let's rewrite the partial derivative in terms of $\frac{\delta y_{pred}}{\delta w_1}$ instead:

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_{pred}} * \frac{\partial y_{pred}}{\partial w_1}$$

This works because of the Chain Rule.

We can calculate $\frac{\delta L}{\delta y_{pred}}$ because we computed $L = (1 - y_{pred})^2$ above:

$$\frac{\partial L}{\partial y_{pred}} = \frac{\partial (1 - y_{pred})^2}{\partial y_{pred}} = \boxed{-2(1 - y_{pred})}$$

Now, let's figure out what to do with $\frac{\delta y_{pred}}{\delta w_1}$ Just like before, let *h*1,*h*2,*o*1 be the outputs of the neurons they represent. Then

$$y_{pred} = o_1 = f(w_5 h_1 + w_6 h_2 + b_3)$$

f is the sigmoid activation function, remember?

Since *w*1 only affects *h*1 (not *h*2), we can write.

$$\frac{\partial y_{pred}}{\partial w_1} = \frac{\partial y_{pred}}{\partial h_1} * \frac{\partial h_1}{\partial w_1}$$

$$\frac{\partial y_{pred}}{\partial h_1} = \boxed{w_5 * f'(w_5 h_1 + w_6 h_2 + b_3)}$$

More Chain Rule.

We do the same thing for $\frac{\delta h_1}{\delta w_1}$:

$$h_1 = f(w_1 x_1 + w_2 x_2 + b_1)$$

$$\frac{\partial h_1}{\partial w_1} = \boxed{x_1 * f'(w_1 x_1 + w_2 x_2 + b_1)}$$

You guessed it, Chain Rule.

*x*1 here is weight, and *x*2 is height. This is the second time we've seen $f(x)'$ (the derivate of the sigmoid function) now! Let's derive it:

$$f(x) = \frac{1}{1 + e^{-x}}$$

$$f'(x) = \frac{e^{-x}}{(1 + e^{-x})^2} = f(x) * (1 - f(x))$$

We'll use this nice form for $f(x)'$ later.

We're done! We've managed to break down $\frac{\delta L}{\delta w_1}$ into several parts we can calculate:

$$\boxed{\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_{pred}} * \frac{\partial y_{pred}}{\partial h_1} * \frac{\partial h_1}{\partial w_1}}$$

This system of calculating partial derivatives by working backwards is known as **backpropagation**, or "backprop".

Similarly,

Rest of the weights,

$$\frac{\delta L}{\delta w_2} = \frac{\delta L}{\delta y_{pred}} * \frac{\delta y_{pred}}{\delta h_1} * \frac{\delta h_1}{\delta w_2}$$

$$\frac{\delta L}{\delta w_3} = \frac{\delta L}{\delta y_{pred}} * \frac{\delta y_{pred}}{\delta h_2} * \frac{\delta h_2}{\delta w_3}$$

$$\frac{\delta L}{\delta w_4} = \frac{\delta L}{\delta y_{pred}} * \frac{\delta y_{pred}}{\delta h_2} * \frac{\delta h_2}{\delta w_4}$$

$$\frac{\delta L}{\delta w_5} = \frac{\delta L}{\delta y_{pred}} * \frac{\delta y_{pred}}{\delta w_5}$$

$$\frac{\delta L}{\delta w_6} = \frac{\delta L}{\delta y_{pred}} * \frac{\delta y_{pred}}{\delta w_6}$$

All the bias updates derivatives,

$$\frac{\delta L}{\delta b_1} = \frac{\delta L}{\delta y_{pred}} * \frac{\delta y_{pred}}{\delta h_1} * \frac{\delta h_1}{\delta b_1}$$

$$\frac{\delta L}{\delta b_2} = \frac{\delta L}{\delta y_{pred}} * \frac{\delta y_{pred}}{\delta h_2} * \frac{\delta h_2}{\delta b_2}$$

$$\frac{\delta L}{\delta b_3} = \frac{\delta L}{\delta y_{pred}} * \frac{\delta y_{pred}}{\delta b_3}$$

Example: Calculating the Partial Derivative

We're going to continue pretending only Alice is in our dataset:

Name	Weight (minus 135)	Height (minus 66)	Gender
Alice	-2	-1	1

Let's initialize all the weights to 1 and all the biases to 0. If we do a feedforward pass through the network, we get:

$$\begin{aligned} h_1 &= f(w_1 x_1 + w_2 x_2 + b_1) \\ &= f(-2 + -1 + 0) \\ &= 0.0474 \end{aligned}$$

$$h_2 = f(w_3 x_1 + w_4 x_2 + b_2) = 0.0474$$

$$\begin{aligned} o_1 &= f(w_5 h_1 + w_6 h_2 + b_3) \\ &= f(0.0474 + 0.0474 + 0) \\ &= 0.524 \end{aligned}$$

The network outputs y_{pred}=0.524, which doesn't strongly favor Male (0) or Female (1). Let's calculate $\frac{\delta L}{\delta w_1}$:

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_{pred}} * \frac{\partial y_{pred}}{\partial h_1} * \frac{\partial h_1}{\partial w_1}$$

$$\begin{aligned} \frac{\partial L}{\partial y_{pred}} &= -2(1 - y_{pred}) \\ &= -2(1 - 0.524) \\ &= -0.952 \end{aligned}$$

$$\begin{aligned}\frac{\partial y_{pred}}{\partial h_1} &= w_5 * f'(w_5 h_1 + w_6 h_2 + b_3) \\ &= 1 * f'(0.0474 + 0.0474 + 0) \\ &= f(0.0948) * (1 - f(0.0948)) \\ &= 0.249\end{aligned}$$

$$\begin{aligned}\frac{\partial h_1}{\partial w_1} &= x_1 * f'(w_1 x_1 + w_2 x_2 + b_1) \\ &= -2 * f'(-2 + -1 + 0) \\ &= -2 * f(-3) * (1 - f(-3)) \\ &= -0.0904\end{aligned}$$

$$\begin{aligned}\frac{\partial L}{\partial w_1} &= -0.952 * 0.249 * -0.0904 \\ &= \boxed{0.0214}\end{aligned}$$

Reminder: we derived $f'(x) = f(x) * (1 - f(x))$ for our sigmoid activation function earlier.

Similarly, an example for bias calculation,

$$\frac{\delta L}{\delta b_1} = \frac{\delta L}{\delta y_{pred}} * \frac{\delta y_{pred}}{\delta h_1} * \frac{\delta h_1}{\delta b_1}$$

$$\frac{\delta L}{\delta y_{pred}} = -0.952$$

$$\frac{\delta y_{pred}}{\delta h_1} = 0.249$$

$$\frac{\delta h_1}{\delta b_1} = \frac{\delta}{\delta b_1} f(w_1 x_1 + w_2 x_2 + b_1) \; where \; h_1 = w_1 x_1 + w_2 x_2 + b_1$$

$$\begin{aligned}\frac{\delta h_1}{\delta b_1} &= \frac{\delta}{\delta b_1} f(w_1 x_1 + w_2 x_2 + b_1) \\ &= \frac{\delta b_1}{\delta b_1} * f'(w_1 x_1 + w_2 x_2 + b_1) \\ &= 1 * f'(-2 - 1 + 0)\end{aligned}$$

$= f(-3) * (1 - f(-3))$ where $f'(x) = f(x) * (1 - f(x))$

$=0.047 * (1 - 0.047) = 0.044791$

Backpropagation

Question: The goal is to compute $J = cos(sin(x^2) + 3\,x^2)$ on the forward pass and derivative $\frac{dJ}{dx}$ on the backward pass.

Solution:

Forward Propagation: The input value is x.

Forward Pass:

(a) Compute $a = x^2$.

(b) Compute $b = sin(a)$.

(c) Compute $c = 3 * x^2$.

(d) Compute $d = b + c$.

(e) Compute $J = cos(d)$.

So, the sequence of calculations is:

$$x^2 \rightarrow sin(x^2) \rightarrow 3 * x^2 \rightarrow sin(x^2) + 3 * x^2 \rightarrow cos(sin(x^2) + 3 * x^2).$$

$$Steps: (a) \rightarrow (b) \rightarrow (c) \rightarrow (d) \rightarrow (e)$$

Backward Propagation:

Backpropagation calculates gradients with respect to the input (x) to perform gradient descent for optimization.

a. Derivative of J with respect to d: $\frac{dJ}{dd} = -sin(d)$ where the J come from last step of forward pass last step (e).
b. Derivative of d with respect to b and c where $d = b + c$ from forward pass step (d):
   Derivative of d with respect to b and c:
   $dd/db = 1$ (because d is directly dependent on b)
   $dd/dc = 1$ (because d is directly dependent on c)

Note: In the backward pass, when we're calculating derivatives using the chain rule, we need to compute the derivative of d with respect to both b and c since d depends on both b and c.

c. Derivative of b with respect to a: $db/da = cos(a)$
d. Derivative of a with respect to x: $da/dx = 2 * x$
e. Derivative of c with respect to x: $dc/dx = 6 * x$

Chain Rule: we can apply the chain rule to compute the overall derivative of J with respect to x:

$$\boldsymbol{dJ/dx = dJ/dd * dd/db * db/da * da/dx + dJ/dd * dd/dc * dc/dx}$$

Question: The goal is to compute cross entropy loss from logistic regression cost function on the forward pass and derivative on the backward pass.

Solution:

Forward Propagation:

Input: The input features are denoted as x, and the true binary label is denoted as y.

Forward Pass:

a. Compute the weighted sum of inputs: $z = w^T * x + b$.

b. Apply the sigmoid activation function to get the predicted probability:

$$h(x) = sigmoid(z) = 1 / (1 + exp(-z)).$$

Compute the Loss:

a. Calculate the logistic loss for this example:

$$J(y, h(x)) = -y * log(h(x)) - (1 - y) * log(1 - h(x)).$$

Backward Propagation:

The goal of backward propagation is to compute the gradients of the cost function with respect to the model parameters (w and b), so we can update the parameters using an optimization algorithm like gradient descent.

Derivative of Loss with respect to the predicted probability:

a. $dJ/dh = -(y / h(x)) + (1 - y) / (1 - h(x))$.

Derivative of the predicted probability with respect to the weighted sum:

a. $dh/dz = h(x) * (1 - h(x))$.

Derivative of the weighted sum with respect to the weights and bias:

a. $dz/dw = x$. Where z is from forward pass step (a), $z = w^T * x + b$.

b. $dz/db = 1$.

Chain Rule:

Apply the chain rule to compute the gradients of the loss with respect to the parameters:

a. $dJ/dw = dJ/dh * dh/dz * dz/dw = x * (h(x) - y)$.

b. $dJ/db = dJ/dh * dh/dz * dz/db = h(x) - y$.

Parameter Update:

Update the weights and bias using the calculated gradients and an optimization algorithm like gradient descent:

a. $w := w - learning\_rate * dJ/dw$.

b. $b := b - learning\_rate * dJ/db$.

These steps constitute one iteration of the optimization process. You repeat the forward and backward propagation steps for multiple iterations until the model's performance converges or reaches a satisfactory level.

In summary, forward propagation calculates the predicted probability and loss, while backward propagation calculates the gradients of the loss with respect to the model parameters, enabling the optimization algorithm to update the parameters for better predictions.