P05-06 Perceptron

Luis Sa-Couto[1] and Andreas Wichert[2]

INESC-ID, Instituto Superior Tecnico, Universidade de Lisboa
{luis.sa.couto,andreas.wichert}@tecnico.ulisboa.pt

1 Perceptron

The perceptron can be seen as an extremely simplified model of a biological neuron that is trained to fire when presented with elements of a given class.

More specifically, for an input vector $\mathbf{x} = \begin{pmatrix} x_1 & x_2 & \cdots & x_d \end{pmatrix}^T$, the perceptron will, ideally, output $o = +1$ if $\mathbf{x}$ is an element of the learned class and $o = -1$ if it is not.

The knowledge that helps the perceptron decide if an element belongs to the class is stored in a vector of weights $\mathbf{w} = \begin{pmatrix} w_1 & w_2 & \cdots & w_d \end{pmatrix}^T$. For each feature x_i in the input vector, there is a weight w_i that decides the importance of that feature to recognize the class, more specifically:

- A very positive weight w_i indicates that if $\mathbf{x}$ has a large x_i than it is likely an instance of the class;
- A very negative weight w_i indicates that if $\mathbf{x}$ has a large x_i than it probably does not belong to the class;
- A w_i close to zero, indicates that x_i is not very relevant for the decision.

To implement this reasoning, the perceptron sums the input features weighted by their importance:

$$w_1 x_1 + w_2 x_2 + \cdots + w_d x_d$$

Now, if the sum is large, then $\mathbf{x}$ is probably a member of the class. But how large? We need a threshold! Let's start by calling it T. If the sum is above T, the perceptron outputs $+1$. Otherwise, it outputs -1. More specifically, we want the perceptron to fire if:

$$perceptron\left(\mathbf{x}\right) = \begin{cases} +1 & w_1 x_1 + w_2 x_2 + \cdots + w_d x_d \geq T \\ -1 & w_1 x_1 + w_2 x_2 + \cdots + w_d x_d < T \end{cases}$$

Which is the same as:

$$perceptron\left(\mathbf{x}\right) = \begin{cases} +1 & w_1 x_1 + w_2 x_2 + \cdots + w_d x_d - T \geq 0 \\ -1 & w_1 x_1 + w_2 x_2 + \cdots + w_d x_d - T < 0 \end{cases}$$

The well-known sign function $sign\left(s\right) = \begin{cases} +1 & s \geq 0 \\ -1 & s < 0 \end{cases}$ can be used to write the perceptron computation in a more compact manner:

$$perceptron\left(\mathbf{x}\right) = sign\left(w_1 x_1 + w_2 x_2 + \cdots + w_d x_d - T\right)$$

In the literature, the threshold is usually treated as just another parameter refered to as the bias $w_0 = -T$, which leads to:

$$perceptron\left(\mathbf{x}\right) = sign\left(w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_d x_d\right)$$

To write the model in an even more compact manner, we can add a dimension to $\mathbf{x}$ called x_0 that corresponds to the bias weight. Naturally, we must have $x_0 = 1$. Having said that, given the weight vector (augmented with the bias) $\mathbf{w} = \begin{pmatrix} w_0 & w_1 & w_2 & \cdots & w_d \end{pmatrix}^T$, we expand all input vectors as follows:

$$\begin{pmatrix} x_1 & x_2 & \cdots & x_d \end{pmatrix}^T \rightarrow \begin{pmatrix} x_0 & x_1 & x_2 & \cdots & x_d \end{pmatrix}^T \rightarrow \begin{pmatrix} 1 & x_1 & x_2 & \cdots & x_d \end{pmatrix}^T$$

So, we can write the perceptron output as the sign of a dot product between input features and weights:

$$perceptron\left(\mathbf{x}\right) = sign\left(\sum_{i=0}^{d} w_i x_i\right) = sign\left(\mathbf{w} \cdot \mathbf{x}\right)$$

Now that we have covered how the perceptron makes decisions, we need to know how to learn the correct weights from training examples. To achieve that we sweep the data using the following procedure:

1. Initialize weights randomly
2. Get training example $\mathbf{x} \in \mathbb{R}^{\mathrm{d}}$ with target $t \in \{-1, +1\}$
3. Compute the perceptron output: $o = perceptron\left(\mathbf{x}\right)$
4. If it made a mistake: $o \neq t$, then for each feature x_i:
   (a) if output is positive $o = +1$ and it should be negative $t = -1$, subtract a fraction of x_i from w_i
   (b) if output is negative $o = -1$ and it should be positive $t = +1$, add a fraction of x_i to w_i
5. If there are more examples, go back to 2.

More specifically, if we define the size of the step (fraction of x_i) as $0 < \eta \leq 1$, we can write the learning rule for each feature as follows.

$$w_i = w_i + \eta\left(t - o\right)x_i$$

1) Consider the following linearly separable training set:

$$\left\{\mathbf{x}^1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \mathbf{x}^2 = \begin{pmatrix} 0 \\ 2 \end{pmatrix}, \mathbf{x}^3 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \mathbf{x}^4 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}\right\}$$

$$\{t_1 = -1, t_2 = +1, t_3 = +1, t_4 = -1\}$$

a) Initialize all weights to one (including the bias). Use a learning rate of one for simplicity. Apply the perceptron learning algorithm until convergence.

Solution:

According to the question, we start with $\eta = 1$ and $\mathbf{w} = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}^T$.

Now, we take the first data point x^1 and augment it with a bias dedicated dimension:

$$\begin{pmatrix} x_0^1 & x_1^1 & x_2^1 \end{pmatrix}^T \rightarrow \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}^T$$

and compute the perceptron output for it:

$$o^1 = sign\left(\mathbf{w} \cdot \mathbf{x}\right) = sign\left(1 \cdot 1 + 1 \cdot 0 + 1 \cdot 0\right) = +1$$

Since the perceptron made a mistake, we use the output to compute an update:

$$\mathbf{w} = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}^T + 1\left(-1 - 1\right)\begin{pmatrix} 1 & 0 & 0 \end{pmatrix}^T = \begin{pmatrix} -1 & 1 & 1 \end{pmatrix}^T$$

We can then repeat the same logic for the next data point:

$$o^2 = sign\left(\mathbf{w} \cdot \mathbf{x}\right) = sign\left(-1 \cdot 1 + 1 \cdot 0 + 1 \cdot 2\right) = +1$$

As no mistake was made, we move to the next point:

$$o^3 = sign\left(\mathbf{w} \cdot \mathbf{x}\right) = sign\left(-1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1\right) = +1$$

Again, no mistake was made, so we can move to the next point:

$$o^4 = sign\left(\mathbf{w} \cdot \mathbf{x}\right) = sign\left(-1 \cdot 1 + 1 \cdot 1 + 1 \cdot -1\right) = -1$$

No mistake was done again, which concludes our first sweep through the data (i.e. the first epoch). If we do another epoch, we see that the weights don't change. Thus, we have convergence.

---

b) Draw the separation hyperplane.

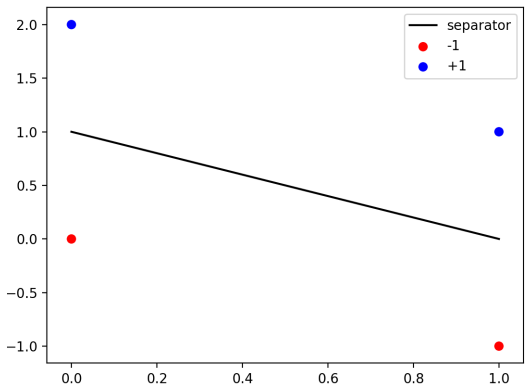
Solution:

Since we are working in two dimensions the weights define a separation line between the two classes. We get the equation for this line by checking the critical point where the decision changes from +1 to -1:

$$w_0 x_0 + w_1 x_1 + w_2 x_2 = 0$$

$$\begin{aligned} w_0 x_0 + w_1 x_1 + w_2 x_2 &= 0 \\ (-1)1 + x_1 + x_2 &= 0 \\ x_2 &= -x_1 + 1 \end{aligned}$$

Having the line's equation, we can draw it in a plot with our data points to see the separation:



---

c) Does the perceptron converge on the first epoch if we change the weight initialization to zeros?

Solution:

According to the question, we start with $\eta = 1$ and $\mathbf{w} = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}^T$.

Now, we take the first data point x^1 and augment it with a bias dedicated dimension:

$$\begin{pmatrix} x_0^1 & x_1^1 & x_2^1 \end{pmatrix}^T \rightarrow \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}^T$$

and compute the perceptron output for it:

$$o^1 = sign\left(\mathbf{w} \cdot \mathbf{x}\right) = sign\left(0 \cdot 1 + 0 \cdot 0 + 0 \cdot 0\right) = +1$$

Since the perceptron made a mistake, we use the output to compute an update:

$$\mathbf{w} = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}^T + 1\left(-1 - 1\right)\begin{pmatrix} 1 & 0 & 0 \end{pmatrix}^T = \begin{pmatrix} -2 & 0 & 0 \end{pmatrix}^T$$

We can then repeat the same logic for the next data point:

$$o^2 = sign\left(\mathbf{w} \cdot \mathbf{x}\right) = sign\left(-2 \cdot 1 + 0 \cdot 0 + 0 \cdot 2\right) = -1$$

Since the perceptron made a mistake, we use the output to compute an update:

$$\mathbf{w} = \begin{pmatrix} -2 & 0 & 0 \end{pmatrix}^T + 1\left(1 - (-1)\right)\begin{pmatrix} 1 & 0 & 2 \end{pmatrix}^T = \begin{pmatrix} 0 & 0 & 4 \end{pmatrix}^T$$

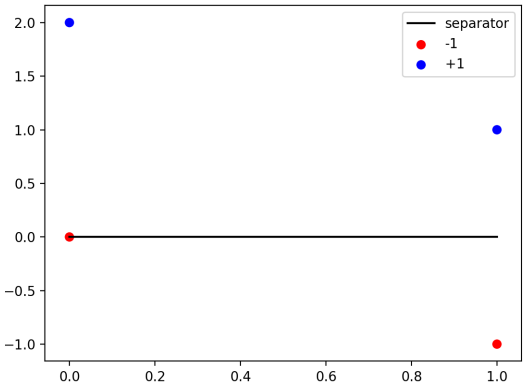
We can then repeat the same logic for the next data point:

$$o^3 = sign\left(\mathbf{w} \cdot \mathbf{x}\right) = sign\left(0 \cdot 1 + 0 \cdot 1 + 4 \cdot 1\right) = +1$$

No mistake was made, so we can move to the next point:

$$o^4 = sign\left(\mathbf{w} \cdot \mathbf{x}\right) = sign\left(0 \cdot 1 + 0 \cdot 1 + 4 \cdot -1\right) = +1$$

No mistake was made, so we finish the first epoch. We can plot the perceptron boundary and check that no convergence was achieved.



---

2) Consider the following linearly separable training set:

$$\left\{\mathbf{x}^1 = \begin{pmatrix}0\\0\\0\end{pmatrix}, \mathbf{x}^2 = \begin{pmatrix}0\\2\\1\end{pmatrix}, \mathbf{x}^3 = \begin{pmatrix}1\\1\\1\end{pmatrix}, \mathbf{x}^4 = \begin{pmatrix}1\\-1\\0\end{pmatrix}\right\}$$

$$\{t_1 = -1, t_2 = +1, t_3 = +1, t_4 = -1\}$$

a) Initialize all weights to one (including the bias). Use a learning rate of one for simplicity. Apply the perception learning algorithm for one epoch.

Solution:

According to the question, we start with $\eta = 1$ and $\mathbf{w} = \begin{pmatrix}1 & 1 & 1 & 1\end{pmatrix}^T$.

Now, we take the first data point x^1 and augment it with a bias dedicated dimension:

$$\begin{pmatrix}x_0^1 & x_1^1 & x_2^1 & x_3^1\end{pmatrix}^T \rightarrow \begin{pmatrix}1 & 0 & 0 & 0\end{pmatrix}^T$$

and compute the perceptron output for it:

$$o^1 = sign\left(\mathbf{w} \cdot \mathbf{x}\right) = sign\left(1 \cdot 1 + 1 \cdot 0 + 1 \cdot 0 + 1 \cdot 0\right) = +1$$

Since the perceptron made a mistake, we use the output to compute an update:

$$\mathbf{w} = \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix}^T + 1\,(-1-1)\begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix}^T = \begin{pmatrix} -1 & 1 & 1 & 1 \end{pmatrix}^T$$

We can then repeat the same logic for the next data point:

$$o^2 = sign\,(\mathbf{w} \cdot \mathbf{x}) = sign\,(-1 \cdot 1 + 1 \cdot 0 + 1 \cdot 2 + 1 \cdot 1) = +1$$

As no mistake was made, we move to the next point:

$$o^3 = sign\,(\mathbf{w} \cdot \mathbf{x}) = sign\,(-1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1) = +1$$

Again, no mistake was made, so we can move to the next point:

$$o^4 = sign\,(\mathbf{w} \cdot \mathbf{x}) = sign\,(-1 \cdot 1 + 1 \cdot 1 + 1 \cdot -1 + 1 \cdot 0) = -1$$

No mistake was done again, which concludes the epoch.

---

b) For an additional epoch, do the weights change?

Solution:
We take the weights at the end of the first epoch:

$$\mathbf{w} = \begin{pmatrix} -1 & 1 & 1 & 1 \end{pmatrix}^T$$

and compute the perceptron output for all points:

$$o^1 = sign\,(\mathbf{w} \cdot \mathbf{x}) = sign\,(-1 \cdot 1 + 1 \cdot 0 + 1 \cdot 0 + 1 \cdot 0) = -1$$

$$o^2 = sign\,(\mathbf{w} \cdot \mathbf{x}) = sign\,(-1 \cdot 1 + 1 \cdot 0 + 1 \cdot 2 + 1 \cdot 1) = +1$$

$$o^3 = sign\,(\mathbf{w} \cdot \mathbf{x}) = sign\,(-1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1) = +1$$

$$o^4 = sign\,(\mathbf{w} \cdot \mathbf{x}) = sign\,(-1 \cdot 1 + 1 \cdot 1 + 1 \cdot -1 + 1 \cdot 0) = -1$$

Since no mistakes occurred, we see that the perceptron converged.

---

c) What is the perceptron output for the query point $\begin{pmatrix} 0 & 0 & 1 \end{pmatrix}^T$?

Solution:
We start by augmenting the query with a bias dimension:

$$\begin{pmatrix} x_0 & x_1 & x_2 & x_3 \end{pmatrix}^T \rightarrow \begin{pmatrix} 1 & 0 & 0 & 1 \end{pmatrix}^T$$

We take the converged weights:

$$\mathbf{w} = \begin{pmatrix} -1 & 1 & 1 & 1 \end{pmatrix}^T$$

And compute the perceptron output for the point:

$$o = sign\left(\mathbf{w} \cdot \mathbf{x}\right) = sign\left(-1 \cdot 1 + 1 \cdot 0 + 1 \cdot 0 + 1 \cdot 1\right) = +1$$

We see that the point is on the boundary. So, because of the way we defined the sign function, the perceptron outputs +1, thus recognizing the point as a member of the learned class.

---

3) What happens if we replace the sign function by the step function?

$$\Theta\left(x\right) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$

Specifically, how would you change the learning rate to get the same results?

Solution:

Recall the lerning rule in question:

$$\mathbf{w} = \mathbf{w} + \eta\left(t - o\right)\mathbf{x}$$

The only term that depends on the perceptron output is the error $(y - o)$. With the sign function, the error is:

$$\delta_{sign} = t - o_{sign} = t - sign(\mathbf{w} \cdot \mathbf{x})$$

Which can be unfolded as:

$$\delta_{sign} = \begin{cases} +2 & t = +1, \mathbf{w} \cdot \mathbf{x} < 0 \\ -2 & t = -1, \mathbf{w} \cdot \mathbf{x} \geq 0 \end{cases}$$

Whereas with the step function, the error is:

$$\delta_{step} = t - o_{step} = t - \Theta(\mathbf{w} \cdot \mathbf{x})$$

Which can be unfolded as:

$$\delta_{step} = \begin{cases} +1 & t = +1, \mathbf{w} \cdot \mathbf{x} < 0 \\ -1 & t = -1, \mathbf{w} \cdot \mathbf{x} \geq 0 \end{cases}$$

Having said that, we notice that there is a relation between the two error terms $\delta_{sign} = 2\delta_{step}$.

So, since we had the update:

$$\mathbf{w} = \mathbf{w} + \eta_{sign}\delta_{sign}\mathbf{x}$$

If we replace the δ_{sign} by the error for the step function we get:

$$\mathbf{w} = \mathbf{w} + 2\eta_{sign}\delta_{step}\mathbf{x}$$

To get exactly the same update, we need to define a learning rate which is twice the one we used $\eta_{step} = 2\eta_{sign}$ and get:

$$\mathbf{w} = \mathbf{w} + \eta_{step}\delta_{step}\mathbf{x}$$

---

4) The perceptron can learn a relatively large number of functions. In this exercise, we focus on simple logical functions.

a) Show graphically that a perceptron can learn the logical NOT function. Give an example with specific weights.

Solution:

The NOT function receives as input a logical value $x \in \{-1, +1\}$ and outputs its logical negation $t \in \{-1, +1\}$. We can enumerate all possible inputs and their inputs:

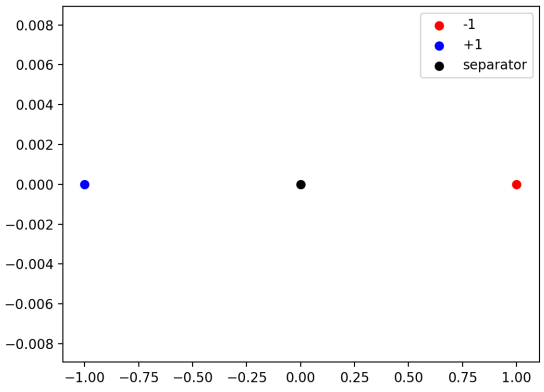
- For $x = -1$ the output is $t = +1$
- For $x = +1$ the output is $t = -1$

To show that a perceptron can learn a given function we just need to show that all points that require a positive output $(+1)$ can be separated from all points that require a negative output by an hyperplane.

Since we are working with 1 dimensional inputs, an hyperplane is, in this case, a point.

So, is there a point that accurately separates the points? Yes, any point between -1 and $+1$ will achieve this.

An example comes from setting the perceptron weights to zero:



b) Show graphically that a perceptron can learn the logical AND function for two inputs. Give an example with specific weights.

Solution:

The AND function receives as input a pair of logical values $\mathbf{x} \in \mathbb{R}^2$ and outputs its logical conjuntion $t \in \{-1, +1\}$. We can enumerate all possible inputs and their inputs:

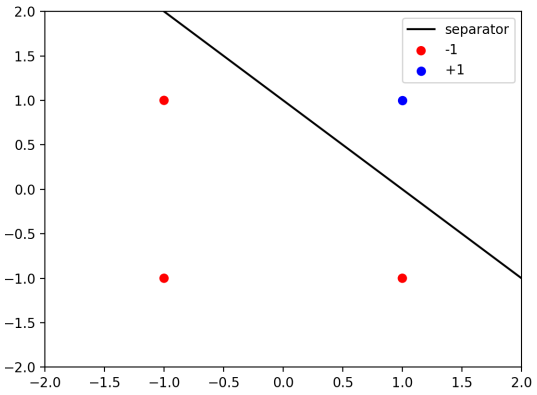
- For $\mathbf{x} = \begin{pmatrix} -1 & -1 \end{pmatrix}^T$ the output is $t = -1$
- For $\mathbf{x} = \begin{pmatrix} -1 & +1 \end{pmatrix}^T$ the output is $t = -1$
- For $\mathbf{x} = \begin{pmatrix} +1 & -1 \end{pmatrix}^T$ the output is $t = -1$
- For $\mathbf{x} = \begin{pmatrix} +1 & +1 \end{pmatrix}^T$ the output is $t = +1$

To show that a perceptron can learn a given function we just need to show that all points that require a positive output $(+1)$ can be separated from all points that require a negative output by an hyperplane.

Since we are working with 2 dimensional inputs, an hyperplane is, in this case, a line.

So, is there a weight vector that defines a boundary line that accurately separates the points?

Yes, for instance $\mathbf{w} = \begin{pmatrix} -1 & 1 & 1 \end{pmatrix}^T$ achieves:



c) Show graphically that a perceptron can learn the logical OR function for two inputs. Give an example with specific weights.

Solution:

The OR function receives as input a pair of logical values $\mathbf{x} \in \mathbb{R}^2$ and outputs its logical disjunction $t \in \{-1, +1\}$. We can enumerate all possible inputs and their inputs:

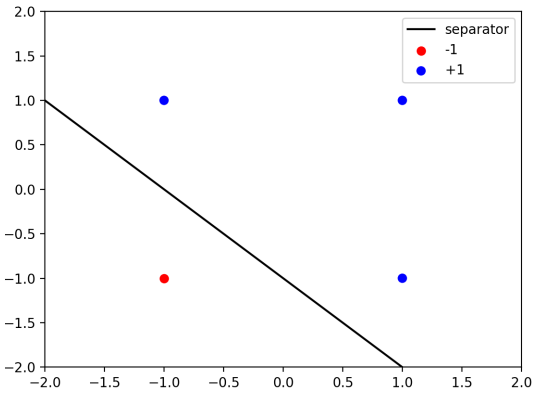
- For $\mathbf{x} = \begin{pmatrix} -1 & -1 \end{pmatrix}^T$ the output is $t = -1$
- For $\mathbf{x} = \begin{pmatrix} -1 & +1 \end{pmatrix}^T$ the output is $t = +1$
- For $\mathbf{x} = \begin{pmatrix} +1 & -1 \end{pmatrix}^T$ the output is $t = +1$
- For $\mathbf{x} = \begin{pmatrix} +1 & +1 \end{pmatrix}^T$ the output is $t = +1$

To show that a perceptron can learn a given function we just need to show that all points that require a positive output $(+1)$ can be separated from all points that require a negative output by an hyperplane.

Since we are working with 2 dimensional inputs, an hyperplane is, in this case, a line.

So, is there a weight vector that defines a boundary line that accurately separates the points?

Yes, for instance $\mathbf{w} = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}^T$ achieves:



---

d) Show graphically that a perceptron can not learn the logical XOR function for two inputs.

Solution:

The XOR function receives as input a pair of logical values $\mathbf{x} \in \mathbb{R}^2$ and outputs the exclusive disjunction $t \in \{-1, +1\}$. We can enumerate all possible inputs and their inputs:

- For $\mathbf{x} = \begin{pmatrix} -1 & -1 \end{pmatrix}^T$ the output is $t = -1$
- For $\mathbf{x} = \begin{pmatrix} -1 & +1 \end{pmatrix}^T$ the output is $t = +1$

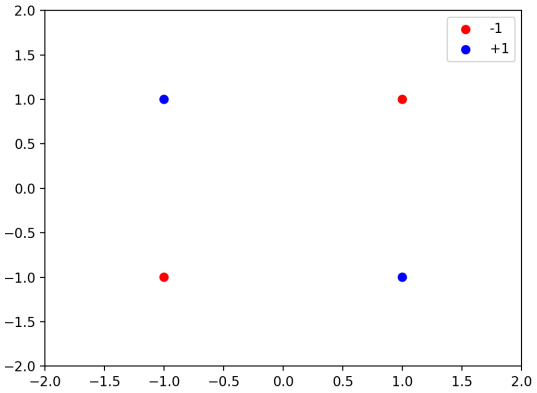
- For $\mathbf{x} = \begin{pmatrix} +1 & -1 \end{pmatrix}^T$ the output is $t = +1$
- For $\mathbf{x} = \begin{pmatrix} +1 & +1 \end{pmatrix}^T$ the output is $t = -1$

To show that a perceptron can learn a given function we need to show that all points that require a positive output (+1) can be separated from all points that require a negative output by an hyperplane.

Since we are working with 2 dimensional inputs, an hyperplane is, in this case, a line.

So, is there a weight vector that defines a boundary line that accurately separates the points?

No... If we plot the points we see right away that no line can separate the positive instances from the negative ones:



2 Gradient descent learning

We have already saw that the perceptron can work with other activation functions besides the sign. From that knowledge it is helpful to define what is called a generalized linear unit that receives an input vector $x = \begin{pmatrix} x_0 & \cdots & x_d \end{pmatrix}^T$, takes the dot product with its weight vector $\mathbf{w} = \begin{pmatrix} w_0 & \cdots & w_d \end{pmatrix}^T$ and passes it to a general activation function g. We can write the output of this generalized unit as follows:

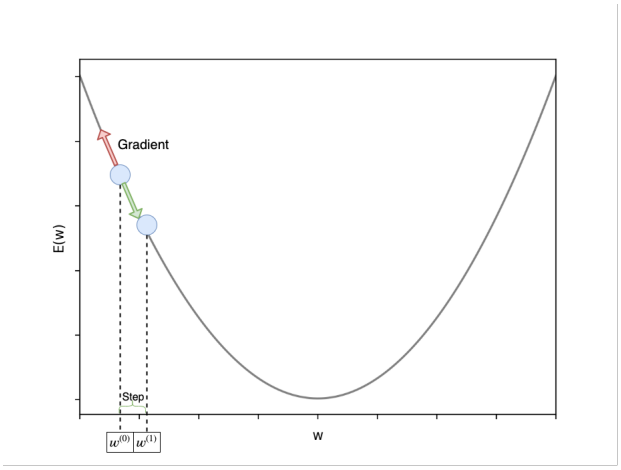
$$unit\left(\mathbf{x};\mathbf{w}\right) = g\left(\mathbf{w}^T\mathbf{x}\right)$$

In the original perceptron, g was the sign function. Afterwards, we worked with the Θ function. In fact, we can use, in principle, any function as we will se in the exercises below.

To perform learning in a generalized linear unit we also define an error function and try to minimize it. However, the process of minimization is not so easy as with linear regression. The activation function makes it impossible to get the same kind of closed form solution. So, we will have to use an iterative procedure, gradient descent!

We want to find the weight vector that minimizes an error function. However, we do not know a priori the error for all possible weight vectors. All we can do is: given an error function (like the sum of squared errors in the previous section) and training data, compute the error for a specific weight vector on that training data.

Illustrated in the figure below is the idea of gradient descent. Specifically, we start with a given weight vector $\mathbf{w}_0$ and compute the error function gradient for that vector $\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}\left(\mathbf{w}_0\right)$. This gradient will point away from the closest minimum value of error. So, we update our weight vector by taking a step into the reverse direction of the gradient $\mathbf{w}^{(1)} = \mathbf{w}^{(0)} - \eta \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}\left(\mathbf{w}^{(0)}\right)$.



So, in general, for a differentiable error function, we have the general learning rule:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \frac{\partial E\left(\mathbf{w}\right)}{\partial \mathbf{w}} \left(\mathbf{w}^{(t)}\right)$$

The error function gradient is computed on some training data. Sometimes, this process can take too long (if we have many examples) so we use stochastic gradient descent where the gradient of the error is computed for a single training example. So, instead of using all examples to take one update step we do one step per example. This makes learning faster. However, approximating the gradient with only one example can be very noisy so we can take very wrong steps sometimes.

Before moving to the exercises, let us make a quick note. When the error function is the cross-entropy loss and the unit's activation function is the sigmoid logistic function we call the model a logistic regression.

1) Consider the following training data:

$$\left\{\mathbf{x}^{(1)} = \begin{pmatrix}1\\1\end{pmatrix}, \mathbf{x}^{(2)} = \begin{pmatrix}2\\1\end{pmatrix}, \mathbf{x}^{(3)} = \begin{pmatrix}1\\3\end{pmatrix}, \mathbf{x}^{(4)} = \begin{pmatrix}3\\3\end{pmatrix}\right\}$$

$$\left\{t^{(1)} = 1, t^{(2)} = 1, t^{(3)} = 0, t^{(4)} = 0\right\}$$

In this exercise, we will work with a unit that computes the following function:

$$output\left(\mathbf{x}; \mathbf{w}\right) = \frac{1}{1 + \exp\left(-2\mathbf{w} \cdot \mathbf{x}\right)}$$

And we will use the half sum of squared errors as our error (loss) function:

$$E\left(\mathbf{w}\right) = \frac{1}{2} \sum_{k=1}^{N} \left(t^{(k)} - output\left(\mathbf{x}^{(k)}; \mathbf{w}\right)\right)^2$$

a) Determine the gradient descent learning rule for this unit.

Solution:

To apply gradient descent, we want an update rule that moves a step of size η towards the opposite direction from the gradient of the error function with respect to the weights:

$$\mathbf{w} = \mathbf{w} - \eta \frac{\partial E\left(\mathbf{w}\right)}{\partial \mathbf{w}}$$

To find the learning rule, we must compute the gradient. Before doing so, we should notice that the model is computing a sigmoid function, so:

$$output\left(\mathbf{x}; \mathbf{w}\right) = \frac{1}{1 + \exp\left(-2\mathbf{w} \cdot \mathbf{x}\right)} = \sigma\left(2\mathbf{w} \cdot \mathbf{x}\right)$$

Which has a well-known derivative that we will use later:

$$
\begin{aligned}
\frac{\partial \sigma(x)}{\partial x} &= \frac{\partial \frac{1}{1+\exp(-x)}}{\partial x} \\
&= \frac{\partial \frac{1}{1+\exp(-x)}}{\partial (1+\exp(-x))} \frac{\partial (1+\exp(-x))}{\partial (\exp(-x))} \frac{\partial (\exp(-x))}{\partial (-x)} \frac{\partial (-x)}{\partial x} \\
&= -\frac{1}{(1+\exp(-x))^2} \exp(-x)(-1) \\
&= \frac{1}{(1+\exp(-x))^2} \exp(-x) \\
&= \frac{1}{1+\exp(-x)} \frac{\exp(-x)}{1+\exp(-x)} \\
&= \frac{1}{1+\exp(-x)} \frac{\exp(-x)+1-1}{1+\exp(-x)} \\
&= \frac{1}{1+\exp(-x)} \left(\frac{\exp(-x)+1}{1+\exp(-x)} - \frac{1}{1+\exp(-x)} \right) \\
&= \frac{1}{1+\exp(-x)} \left(1 - \frac{1}{1+\exp(-x)} \right) \\
&= \sigma(x)(1-\sigma(x))
\end{aligned}
$$

Having made this auxiliar computation, it is now easier to compute the derivative of the error function with respect to the parameter vector.

$$
\begin{aligned}
\frac{\partial E\left(\mathbf{w}\right)}{\partial \mathbf{w}} &= \frac{\partial \frac{1}{2}\sum_{k=1}^{N}\left(t^{(k)} - output\left(\mathbf{x}^{(k)};\mathbf{w}\right)\right)^2}{\partial \mathbf{w}} \\
&= \frac{\partial \frac{1}{2}\sum_{k=1}^{N}\left(t^{(k)} - \sigma\left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right)^2}{\partial \mathbf{w}} \\
&= \frac{1}{2}\frac{\partial \sum_{k=1}^{N}\left(t^{(k)} - \sigma\left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right)^2}{\partial \mathbf{w}} \\
&= \frac{1}{2}\sum_{k=1}^{N}\frac{\partial \left(t^{(k)} - \sigma\left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right)^2}{\partial \mathbf{w}} \\
&= \frac{1}{2}\sum_{k=1}^{N}\left(\frac{\partial \left(t^{(k)} - \sigma\left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right)^2}{\partial \left(t^{(k)} - \sigma\left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right)}\frac{\partial \left(t^{(k)} - \sigma\left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right)}{\partial \sigma\left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)}\frac{\partial \sigma\left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)}{\partial \left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)}\frac{\partial \left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)}{\partial \mathbf{w}}\right) \\
&= \frac{1}{2}\sum_{k=1}^{N}\left(2\left(t^{(k)} - \sigma\left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right)\frac{\partial \left(t^{(k)} - \sigma\left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right)}{\partial \sigma\left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)}\frac{\partial \sigma\left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)}{\partial \left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)}\frac{\partial \left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)}{\partial \mathbf{w}}\right) \\
&= \frac{1}{2}\sum_{k=1}^{N}\left(2\left(t^{(k)} - \sigma\left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right)(-1)\frac{\partial \sigma\left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)}{\partial \left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)}\frac{\partial \left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)}{\partial \mathbf{w}}\right) \\
&= \frac{1}{2}\sum_{k=1}^{N}\left(2\left(t^{(k)} - \sigma\left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right)(-1)\left(\sigma\left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\left(1-\sigma\left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right)\right)\frac{\partial \left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)}{\partial \mathbf{w}}\right) \\
&= \frac{1}{2}\sum_{k=1}^{N}\left(2\left(t^{(k)} - \sigma\left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right)(-1)\left(\sigma\left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\left(1-\sigma\left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right)\right)\left(2\mathbf{x}^{(k)}\right)\right) \\
&= -2\sum_{k=1}^{N}\left(\left(t^{(k)} - \sigma\left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right)\left(\sigma\left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\left(1-\sigma\left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right)\right)\mathbf{x}^{(k)}\right)
\end{aligned}
$$

So, we can write our update rule as follows.

$$
\begin{aligned}
\mathbf{w} &= \mathbf{w} - \eta\frac{\partial E\left(\mathbf{w}\right)}{\partial \mathbf{w}} \\
&= \mathbf{w} - \eta\left(-2\sum_{k=1}^{N}\left(\left(t^{(k)} - \sigma\left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right)\left(\sigma\left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\left(1-\sigma\left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right)\right)\mathbf{x}^{(k)}\right)\right) \\
&= \mathbf{w} + 2\eta\sum_{k=1}^{N}\left(\left(t^{(k)} - \sigma\left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right)\left(\sigma\left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\left(1-\sigma\left(2\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right)\right)\mathbf{x}^{(k)}\right)
\end{aligned}
$$

---

b) Compute the first gradient descent update assuming an initialization of all ones .

Solution:

The original gradient descent does one update per epoch because its learning rule requires contributions from all data points to do one step. Let us compute it.

According to the problem statement, we start with weights $\mathbf{w} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$ and learning rate $\eta = 1.0$.

$$
\begin{aligned}
\mathbf{w} &= \mathbf{w} + 2\eta \sum_{k=1}^{4} \left(\left(t^{(k)} - \sigma\left(2\mathbf{w} \cdot \mathbf{x}^{(k)} \right) \right) \left(\sigma\left(2\mathbf{w} \cdot \mathbf{x}^{(k)} \right) \left(1 - \sigma\left(2\mathbf{w} \cdot \mathbf{x}^{(k)} \right) \right) \right) \mathbf{x}^{(k)} \right) \\
&= \mathbf{w} + \sum_{k=1}^{4} 2\eta \left(\left(t^{(k)} - \sigma\left(2\mathbf{w} \cdot \mathbf{x}^{(k)} \right) \right) \left(\sigma\left(2\mathbf{w} \cdot \mathbf{x}^{(k)} \right) \left(1 - \sigma\left(2\mathbf{w} \cdot \mathbf{x}^{(k)} \right) \right) \right) \mathbf{x}^{(k)} \right) \\
&= \mathbf{w} + 2\eta \left(\left(t^{(1)} - \sigma\left(2\mathbf{w} \cdot \mathbf{x}^{(1)} \right) \right) \left(\sigma\left(2\mathbf{w} \cdot \mathbf{x}^{(1)} \right) \left(1 - \sigma\left(2\mathbf{w} \cdot \mathbf{x}^{(1)} \right) \right) \right) \mathbf{x}^{(1)} \right) + \\
&+ 2\eta \left(\left(t^{(2)} - \sigma\left(2\mathbf{w} \cdot \mathbf{x}^{(2)} \right) \right) \left(\sigma\left(2\mathbf{w} \cdot \mathbf{x}^{(2)} \right) \left(1 - \sigma\left(2\mathbf{w} \cdot \mathbf{x}^{(2)} \right) \right) \right) \mathbf{x}^{(2)} \right) + \\
&+ 2\eta \left(\left(t^{(3)} - \sigma\left(2\mathbf{w} \cdot \mathbf{x}^{(3)} \right) \right) \left(\sigma\left(2\mathbf{w} \cdot \mathbf{x}^{(3)} \right) \left(1 - \sigma\left(2\mathbf{w} \cdot \mathbf{x}^{(3)} \right) \right) \right) \mathbf{x}^{(3)} \right) + \\
&+ 2\eta \left(\left(t^{(4)} - \sigma\left(2\mathbf{w} \cdot \mathbf{x}^{(4)} \right) \right) \left(\sigma\left(2\mathbf{w} \cdot \mathbf{x}^{(4)} \right) \left(1 - \sigma\left(2\mathbf{w} \cdot \mathbf{x}^{(4)} \right) \right) \right) \mathbf{x}^{(4)} \right) = \\
&= \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} + \\
&+ 2 \left(1 - \sigma \left(2 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \right) \right) \sigma \left(2 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \right) \left(1 - \sigma \left(2 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \right) \right) \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \\
&+ 2 \left(1 - \sigma \left(2 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} \right) \right) \sigma \left(2 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} \right) \left(1 - \sigma \left(2 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} \right) \right) \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} + \\
&+ 2 \left(0 - \sigma \left(2 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 3 \end{pmatrix} \right) \right) \sigma \left(2 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 3 \end{pmatrix} \right) \left(1 - \sigma \left(2 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 3 \end{pmatrix} \right) \right) \begin{pmatrix} 1 \\ 1 \\ 3 \end{pmatrix} + \\
&+ 2 \left(0 - \sigma \left(2 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 3 \\ 3 \end{pmatrix} \right) \right) \sigma \left(2 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 3 \\ 3 \end{pmatrix} \right) \left(1 - \sigma \left(2 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 3 \\ 3 \end{pmatrix} \right) \right) \begin{pmatrix} 1 \\ 3 \\ 3 \end{pmatrix} = \\
&= \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} + 2 \left(1 - \sigma \left(6 \right) \right) \left(\sigma \left(6 \right) \left(1 - \sigma \left(6 \right) \right) \right) \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} + 2 \left(1 - \sigma \left(8 \right) \right) \left(\sigma \left(8 \right) \left(1 - \sigma \left(8 \right) \right) \right) \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} + \\
&+ 2 \left(0 - \sigma \left(10 \right) \right) \left(\sigma \left(10 \right) \left(1 - \sigma \left(10 \right) \right) \right) \begin{pmatrix} 1 \\ 1 \\ 3 \end{pmatrix} + 2 \left(0 - \sigma \left(14 \right) \right) \left(\sigma \left(14 \right) \left(1 - \sigma \left(14 \right) \right) \right) \begin{pmatrix} 1 \\ 3 \\ 3 \end{pmatrix} = \\
&= \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 1.2197 \times 10^{-5} \\ 1.2197 \times 10^{-5} \\ 1.2197 \times 10^{-5} \end{pmatrix} + \begin{pmatrix} 2.2484 \times 10^{-7} \\ 4.4969 \times 10^{-7} \\ 2.2484 \times 10^{-7} \end{pmatrix} + \\
&+ \begin{pmatrix} -9.0787 \times 10^{-5} \\ -9.0787 \times 10^{-5} \\ -2.7236 \times 10^{-4} \end{pmatrix} + \begin{pmatrix} -1.6631 \times 10^{-6} \\ -4.9892 \times 10^{-6} \\ -4.9892 \times 10^{-6} \end{pmatrix} = \\
&= \begin{pmatrix} 0.99991997 \\ 0.99991687 \\ 0.99973507 \end{pmatrix}
\end{aligned}
$$

---

c) Compute the first stochastic gradient descent update assuming an initialization of all ones.

Solution:

In stochastic gradient descent we make one update for each training example. So, instead of summing accross all data points we adapt the learning rule for one example only:

$$\mathbf{w} = \mathbf{w} + 2\eta \left(\left(t - \sigma \left(2\mathbf{w} \cdot \mathbf{x} \right) \right) \left(\sigma \left(2\mathbf{w} \cdot \mathbf{x} \right) \left(1 - \sigma \left(2\mathbf{w} \cdot \mathbf{x} \right) \right) \right) \mathbf{x} \right)$$

We can now do the updates. Let us start with the first example:

$$\begin{aligned}
&\mathbf{w} = \mathbf{w} + 2\eta \left(\left(t - \sigma \left(2\mathbf{w} \cdot \mathbf{x} \right) \right) \left(\sigma \left(2\mathbf{w} \cdot \mathbf{x} \right) \left(1 - \sigma \left(2\mathbf{w} \cdot \mathbf{x} \right) \right) \right) \mathbf{x} \right) \\
&= \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} + \\
&+ 2 \left(1 - \sigma \left(2 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \right) \right) \sigma \left(2 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \right) \left(1 - \sigma \left(2 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \right) \right) \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \\
&= \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} + 2 \left(1 - \sigma \left(6 \right) \right) \left(\sigma \left(6 \right) \left(1 - \sigma \left(6 \right) \right) \right) \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \\
&= \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 1.2197 \times 10^{-5} \\ 1.2197 \times 10^{-5} \\ 1.2197 \times 10^{-5} \end{pmatrix} \\
&= \begin{pmatrix} 1.0000122 \\ 1.0000122 \\ 1.0000122 \end{pmatrix}
\end{aligned}$$

---

2) Consider the following training data:

$$\left\{ \mathbf{x}^{(1)} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \mathbf{x}^{(2)} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}, \mathbf{x}^{(3)} = \begin{pmatrix} 1 \\ 3 \end{pmatrix}, \mathbf{x}^{(4)} = \begin{pmatrix} 3 \\ 3 \end{pmatrix} \right\}$$

$$\left\{ t^{(1)} = 1, t^{(2)} = 1, t^{(3)} = 0, t^{(4)} = 0 \right\}$$

In this exercise, we will work with a unit that computes the following function:

$$output \left(\mathbf{x}; \mathbf{w} \right) = \frac{1}{1 + \exp \left(-\mathbf{w} \cdot \mathbf{x} \right)}$$

And we will use the cross-entropy loss function:

$$E\left(\mathbf{w}\right) = -\log\left(p\left(\mathbf{t} \mid \mathbf{w}\right)\right) = -\sum_{k=1}^{N}\left(t^{(k)}\log output^{(k)}\left(\mathbf{x}^{(k)};\mathbf{w}\right) + \left(1-t^{(k)}\right)\log\left(1-output^{(k)}\left(\mathbf{x}^{(k)};\mathbf{w}\right)\right)\right)$$

a) Determine the gradient descent learning rule for this unit.

Solution:

To apply gradient descent, we want an update rule that moves a step of size η towards the opposite direction from the gradient of the error function with respect to the weights:

$$\mathbf{w} = \mathbf{w} - \eta\frac{\partial E\left(\mathbf{w}\right)}{\partial \mathbf{w}}$$

To find the learning rule, we must compute the gradient. Before doing so, we should notice that the model is computing a sigmoid function, so:

$$output\left(\mathbf{x};\mathbf{w}\right) = \frac{1}{1+\exp\left(-\mathbf{w}\cdot\mathbf{x}\right)} = \sigma\left(\mathbf{w}\cdot\mathbf{x}\right)$$

Which has a well-known derivative that we computed earlier:

$$\frac{\partial\sigma\left(x\right)}{\partial x} = \sigma\left(x\right)\left(1-\sigma\left(x\right)\right)$$

Having made this auxiliar computation, it is now easier to compute the derivative of the error function with respect to the parameter vector.

$$
\begin{aligned}
\frac{\partial E\left(\mathbf{w}\right)}{\partial \mathbf{w}} &= \frac{\partial\left(-\sum_{k=1}^{N}\left(t^{(k)}\log output^{(k)}\left(\mathbf{x}^{(k)};\mathbf{w}\right)+\left(1-t^{(k)}\right)\log\left(1-output^{(k)}\left(\mathbf{x}^{(k)};\mathbf{w}\right)\right)\right)\right)}{\partial \mathbf{w}} \\
&= -\sum_{k=1}^{N}\frac{\partial\left(t^{(k)}\log output^{(k)}\left(\mathbf{x}^{(k)};\mathbf{w}\right)+\left(1-t^{(k)}\right)\log\left(1-output^{(k)}\left(\mathbf{x}^{(k)};\mathbf{w}\right)\right)\right)}{\partial \mathbf{w}} \\
&= -\sum_{k=1}^{N}\frac{\partial\left(t^{(k)}\log\sigma\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)+\left(1-t^{(k)}\right)\log\left(1-\sigma\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right)\right)}{\partial \mathbf{w}} \\
&= -\sum_{k=1}^{N}\left(\frac{\partial\left(t^{(k)}\log\sigma\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right)}{\partial \mathbf{w}}+\frac{\partial\left(\left(1-t^{(k)}\right)\log\left(1-\sigma\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right)\right)}{\partial \mathbf{w}}\right) \\
&= -\sum_{k=1}^{N}\left(t^{(k)}\frac{\partial\left(\log\sigma\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right)}{\partial \mathbf{w}}+\left(1-t^{(k)}\right)\frac{\partial\left(\log\left(1-\sigma\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right)\right)}{\partial \mathbf{w}}\right) \\
&= -\sum_{k=1}^{N}(t^{(k)}\frac{\partial\left(\log\sigma\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right)}{\partial\sigma\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)}\frac{\partial\sigma\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)}{\partial\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)}\frac{\partial\mathbf{w}\cdot\mathbf{x}^{(k)}}{\partial \mathbf{w}}+ \\
&+\left(1-t^{(k)}\right)\frac{\partial\left(\log\left(1-\sigma\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right)\right)}{\partial\left(1-\sigma\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right)}\frac{\partial\left(1-\sigma\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right)}{\partial\sigma\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)}\frac{\partial\sigma\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)}{\partial\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)}\frac{\partial\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)}{\partial \mathbf{w}}) \\
&= -\sum_{k=1}^{N}(t^{(k)}\frac{1}{\sigma\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)}\sigma\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\left(1-\sigma\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right)\mathbf{x}^{(k)}+ \\
&+\left(1-t^{(k)}\right)\frac{1}{1-\sigma\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)}\left(-1\right)\sigma\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\left(1-\sigma\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right)\mathbf{x}^{(k)}) \\
&= -\sum_{k=1}^{N}\left(t^{(k)}\left(1-\sigma\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right)\mathbf{x}^{(k)}+\left(1-t^{(k)}\right)\left(-1\right)\sigma\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\mathbf{x}^{(k)}\right) \\
&= -\sum_{k=1}^{N}\left(t^{(k)}\left(1-\sigma\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right)\mathbf{x}^{(k)}-\left(1-t^{(k)}\right)\sigma\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\mathbf{x}^{(k)}\right) \\
&= -\sum_{k=1}^{N}\mathbf{x}^{(k)}\left(t^{(k)}\left(1-\sigma\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right)-\left(1-t^{(k)}\right)\sigma\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right) \\
&= -\sum_{k=1}^{N}\mathbf{x}^{(k)}\left(t^{(k)}-t^{(k)}\sigma\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)-\left(\sigma\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)-t^{(k)}\sigma\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right)\right) \\
&= -\sum_{k=1}^{N}\mathbf{x}^{(k)}\left(t^{(k)}-t^{(k)}\sigma\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)-\sigma\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)+t^{(k)}\sigma\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right) \\
&= -\sum_{k=1}^{N}\mathbf{x}^{(k)}\left(t^{(k)}-\sigma\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\right)
\end{aligned}
$$

So, we can write our update rule as follows.

$$
\begin{aligned}
\mathbf{w} &= \mathbf{w} - \eta \frac{\partial E\left(\mathbf{w}\right)}{\partial \mathbf{w}} \\
&= \mathbf{w} - \eta \left(- \sum_{k=1}^{N} \mathbf{x}^{(k)} \left(t^{(k)} - \sigma \left(\mathbf{w} \cdot \mathbf{x}^{(k)} \right) \right) \right) \\
&= \mathbf{w} + \eta \sum_{k=1}^{N} \mathbf{x}^{(k)} \left(t^{(k)} - \sigma \left(\mathbf{w} \cdot \mathbf{x}^{(k)} \right) \right)
\end{aligned}
$$

---

b) Compute the first gradient descent update assuming an initialization of all ones .

Solution:

The original gradient descent does one update per epoch because its learning rule requires contributions from all data points to do one step. Let us compute it.

According to the problem statement, we start with weights $\mathbf{w} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$ and learning rate $\eta = 1.0$.

$$
\begin{aligned}
\mathbf{w} &= \mathbf{w} + \eta \sum_{k=1}^{N} \mathbf{x}^{(k)} \left(t^{(k)} - \sigma \left(\mathbf{w} \cdot \mathbf{x}^{(k)} \right) \right) \\
&= \mathbf{w} + \eta \sum_{k=1}^{4} \mathbf{x}^{(k)} \left(t^{(k)} - \sigma \left(\mathbf{w} \cdot \mathbf{x}^{(k)} \right) \right) \\
&= \mathbf{w} + \eta \mathbf{x}^{(1)} \left(t^{(1)} - \sigma \left(\mathbf{w} \cdot \mathbf{x}^{(1)} \right) \right) + \eta \mathbf{x}^{(2)} \left(t^{(2)} - \sigma \left(\mathbf{w} \cdot \mathbf{x}^{(2)} \right) \right) + \\
&+ \eta \mathbf{x}^{(3)} \left(t^{(3)} - \sigma \left(\mathbf{w} \cdot \mathbf{x}^{(3)} \right) \right) + \eta \mathbf{x}^{(4)} \left(t^{(4)} - \sigma \left(\mathbf{w} \cdot \mathbf{x}^{(4)} \right) \right) \\
&= \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \left(1 - \sigma \left(\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \right) \right) + \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} \left(1 - \sigma \left(\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} \right) \right) + \\
&+ \begin{pmatrix} 1 \\ 1 \\ 3 \end{pmatrix} \left(0 - \sigma \left(\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 3 \end{pmatrix} \right) \right) + \begin{pmatrix} 1 \\ 3 \\ 3 \end{pmatrix} \left(0 - \sigma \left(\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 3 \\ 3 \end{pmatrix} \right) \right) \\
&= \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} (1 - \sigma(3)) + \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} (1 - \sigma(4)) + \begin{pmatrix} 1 \\ 1 \\ 3 \end{pmatrix} (0 - \sigma(5)) + \begin{pmatrix} 1 \\ 3 \\ 3 \end{pmatrix} (0 - \sigma(7)) \\
&= \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 0.0474 \\ 0.0474 \\ 0.0474 \end{pmatrix} + \begin{pmatrix} 0.0179 \\ 0.0359 \\ 0.0179 \end{pmatrix} + \begin{pmatrix} -0.9933 \\ -0.9933 \\ -2.9799 \end{pmatrix} + \begin{pmatrix} -0.9991 \\ -2.9973 \\ -2.9973 \end{pmatrix} \\
&= \begin{pmatrix} -0.9269 \\ -2.9072 \\ -4.9118 \end{pmatrix}
\end{aligned}
$$

---

c) Compute the first stochastic gradient descent update assuming an initialization of all ones.

Solution:

In stochastic gradient descent we make one update for each training example. So, instead of summing accross all data points we adapt the learning rule for one example only:

$$\mathbf{w} = \mathbf{w} + \eta \mathbf{x} \left(t - \sigma \left(\mathbf{w} \cdot \mathbf{x} \right) \right)$$

We can now do the updates. Let us start with the first example:

$$
\begin{aligned}
\mathbf{w} &= \mathbf{w} + \eta \mathbf{x} \left(t - \sigma \left(\mathbf{w} \cdot \mathbf{x} \right) \right) \\
&= \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} + 1 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \left(1 - \sigma \left(\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \right) \right) \\
&= \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \left(1 - \sigma \left(3 \right) \right) \\
&= \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 0.0474 \\ 0.0474 \\ 0.0474 \end{pmatrix} \\
&= \begin{pmatrix} 1.0474 \\ 1.0474 \\ 1.0474 \end{pmatrix}
\end{aligned}
$$

---

3) Consider the following training data:

$$
\left\{ \mathbf{x}^{(1)} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \mathbf{x}^{(2)} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}, \mathbf{x}^{(3)} = \begin{pmatrix} 1 \\ 3 \end{pmatrix}, \mathbf{x}^{(4)} = \begin{pmatrix} 3 \\ 3 \end{pmatrix} \right\}
$$

$$
\left\{ t^{(1)} = 1, t^{(2)} = 1, t^{(3)} = 0, t^{(4)} = 0 \right\}
$$

In this exercise, we will work with a unit that computes the following function:

$$
output \left(\mathbf{x}; \mathbf{w} \right) = \exp \left(\left(\mathbf{w} \cdot \mathbf{x} \right)^2 \right)
$$

And we will use the half sum of squared errors as our error (loss) function:

$$
E \left(\mathbf{w} \right) = \frac{1}{2} \sum_{k=1}^{N} \left(t^{(k)} - output \left(\mathbf{x}^{(k)}; \mathbf{w} \right) \right)
$$

a) Determine the gradient descent learning rule for this unit.

Solution:

To apply gradient descent, we want an update rule that moves a step of size η towards the opposite direction from the gradient of the error function with respect to the weights:

$$
\mathbf{w} = \mathbf{w} - \eta \frac{\partial E \left(\mathbf{w} \right)}{\partial \mathbf{w}}
$$

To find the learning rule, we must compute the gradient of the error function with respect to the parameter vector.

$$\begin{aligned}
\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} &= \frac{\partial \frac{1}{2}\sum_{k=1}^{N}\left(t^{(k)} - output\left(\mathbf{x}^{(k)};\mathbf{w}\right)\right)^2}{\partial \mathbf{w}} \\
&= \frac{\partial \frac{1}{2}\sum_{k=1}^{N}\left(t^{(k)} - \exp\left(\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)^2\right)\right)^2}{\partial \mathbf{w}} \\
&= \frac{1}{2}\frac{\partial \sum_{k=1}^{N}\left(t^{(k)} - \exp\left(\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)^2\right)\right)^2}{\partial \mathbf{w}} \\
&= \frac{1}{2}\sum_{k=1}^{N}\frac{\partial\left(t^{(k)} - \exp\left(\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)^2\right)\right)^2}{\partial \mathbf{w}} \\
&= \frac{1}{2}\sum_{k=1}^{N}\frac{\partial\left(t^{(k)} - \exp\left(\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)^2\right)\right)^2}{\partial\left(t^{(k)} - \exp\left(\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)^2\right)\right)}\frac{\partial\left(t^{(k)} - \exp\left(\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)^2\right)\right)}{\partial \exp\left(\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)^2\right)} \\
&\quad \frac{\partial \exp\left(\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)^2\right)}{\partial\left(\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)^2\right)}\frac{\partial\left(\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)^2\right)}{\partial\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)}\frac{\partial\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)}{\partial \mathbf{w}} \\
&= \frac{1}{2}\sum_{k=1}^{N}\left(2\left(t^{(k)} - \exp\left(\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)^2\right)\right)(-1)\exp\left(\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)^2\right)2\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\mathbf{x}^{(k)}\right) \\
&= -2\sum_{k=1}^{N}\left(\left(t^{(k)} - \exp\left(\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)^2\right)\right)\exp\left(\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)^2\right)\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\mathbf{x}^{(k)}\right)
\end{aligned}$$

So, we can write our update rule as follows.

$$\begin{aligned}
\mathbf{w} &= \mathbf{w} - \eta\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} \\
&= \mathbf{w} - \eta\left(-2\sum_{k=1}^{N}\left(\left(t^{(k)} - \exp\left(\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)^2\right)\right)\exp\left(\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)^2\right)\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\mathbf{x}^{(k)}\right)\right) \\
&= \mathbf{w} + 2\eta\sum_{k=1}^{N}\left(\left(t^{(k)} - \exp\left(\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)^2\right)\right)\exp\left(\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)^2\right)\left(\mathbf{w}\cdot\mathbf{x}^{(k)}\right)\mathbf{x}^{(k)}\right)
\end{aligned}$$

---

b) Compute the stochastic gradient descent update for input $\mathbf{x} = \begin{pmatrix}1\\1\end{pmatrix}, t = 0$ initialized with $\mathbf{w} = \begin{pmatrix}0\\1\\0\end{pmatrix}$ and learning rate $\eta = 2$.

Solution:

In stochastic gradient descent we make one update for each training example. So, instead of summing accross all data points we adapt the learning rule for one example only:

$$\mathbf{w} = \mathbf{w} + 2\eta \left(\left(t - \exp\left((\mathbf{w} \cdot \mathbf{x})^2 \right) \right) \exp\left((\mathbf{w} \cdot \mathbf{x})^2 \right) (\mathbf{w} \cdot \mathbf{x}) \, \mathbf{x} \right)$$

We can now do the updates. Let us start with the first example:

$$\begin{aligned}
&\mathbf{w} = \mathbf{w} + 2\eta \left(\left(t - \exp\left((\mathbf{w} \cdot \mathbf{x})^2 \right) \right) \exp\left((\mathbf{w} \cdot \mathbf{x})^2 \right) (\mathbf{w} \cdot \mathbf{x}) \, \mathbf{x} \right) \\
&= \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + 2\,(2) \left(0 - \exp\left(\left(\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \right)^2 \right) \right) \\
&\exp\left(\left(\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \right)^2 \right) \left(\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \right) \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \\
&= \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + 4 \left((0 - \exp(1)) \exp(1) \, (1) \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \right) \\
&= \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} - 4 \begin{pmatrix} e^2 \\ e^2 \\ e^2 \end{pmatrix} \\
&= \begin{pmatrix} -4e^2 \\ 1 - 4e^2 \\ -4e^2 \end{pmatrix}
\end{aligned}$$

---

3 Thinking Questions

a) Think about the error functions we have seen. Do you think that one is clearly better than the other? What changes when one changes the error function?

b) Could you implement a statistical machine learning application that classifies a number into two classes, primes and non-primes? What is the main difference between this problem and the salmon and sea bass example from the lecture?