MLISP: Machine Learning in Signal Processing

Solutions to problem set 5

Prof. V. I. Morgenshtern

Solver: A. V. Corrales, G. Miller, V. I. Morgenshtern

Problem 1: Invariance of perceptron

The goal of this problem is to show that if all the weights and biases of a perceptron network are multiplied by a positive constant, $c > 0$, the behavior of the network does not change.

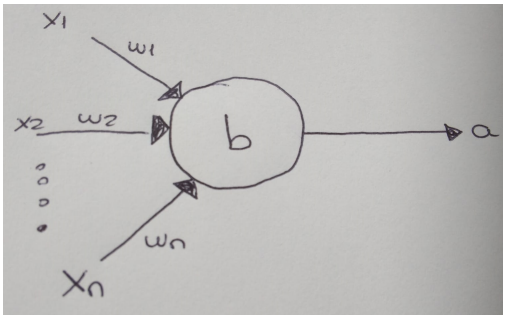


Figure 1: Perceptron neuron with its inputs and output

In order to prove this statement, first let's show that the behavior of a single perceptron does not change when it's weights and the bias is multiplied by constant a $c > 0$. First, recall the activation equation of a single neuron. Let

$$z = \mathbf{w}^\mathsf{T}\mathbf{x} + b, \tag{1}$$

where $\mathbf{w}$ and $\mathbf{x}$ are the vectors containing all the weights and inputs respectively, then

$$\text{output} = \begin{cases} 0 & \text{if } z \leq 0 \\ 1 & \text{if } z > 0 \end{cases} \tag{2}$$

Then, multiplying all weights and the bias by the constant c:

$$z' = (\mathbf{w}')^\intercal \mathbf{x} + b' \tag{3}$$

where $\mathbf{w}' = c \cdot \mathbf{w}$ and $b' = b \cdot c$. We see that $z' = c \cdot z,\ c \geq 0$, and therefore $\text{sign}(z') = \text{sign}(z)$ As z and z' have the same sign, according to (2), the output of the single neuron will not change. Finally, as every preceptron in the network do not change their output, the output of the whole network will also not change.

Problem 2: Emulating perceptron via sigmoidal neuron

The aim of this exercise is to prove that if all the weights and all the biases of a sigmoid network are multiplied by a positive constant, $c > 0$, the behavior of the network is the same as the one of a perceptron network with the same weights and same biases in the limit $c \to \infty$.

First, recall the behavior of these two types of neurons. Perceptrons have an activation function with the following property:

$$z = \mathbf{w}^\mathsf{T}\mathbf{x} + b$$

where $\mathbf{w}$ and $\mathbf{x}$ are the vectors containing all the weights and inputs respectively. If the value of z is positive, the neuron outputs 1; whereas if the value of z is negative, the neuron outputs 0.

For sigmoids, the output is obtained with the logistic function:

$$\text{output} = \frac{1}{1 + e^{-z}} \tag{4}$$

and

$$z = \mathbf{w}^\mathsf{T}\mathbf{x} + b. \tag{5}$$

Therefore, the output of sigmoids is not just 0 and 1, but it can take any intermediate value.

If the weights and bias of a sigmoid are multiplied by a positive constant c, (5) becomes:

$$z' = c \cdot (\mathbf{w}^\mathsf{T}\mathbf{x} + b).$$

and the output function in (4) becomes:

$$\text{output} = \frac{1}{1 + e^{-z'}} = \frac{1}{1 + e^{-(c \cdot z)}}.$$

To prove the point of the exercise, let's calculate the limit of the output of the neuron as $c \to \infty$:

$$\lim_{c \to \infty} \frac{1}{1 + e^{-(c \cdot z)}} = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z < 0. \end{cases} \tag{6}$$

From the last equation we see that the limit is equal to 1 if $z > 0$ and 0 if $z < 0$. Therefore, the output of the sigmoid is the same as the one of a perceptron.

As this property holds for every neuron, it follows that the behavior of a perceptron network would be the same as that of a sigmoid network with the same weights and biases, as long as all the weights and biases are multiplied by a positive arbitrary constant that tends to ∞.

Problem 3: From one-hot to binary

The goal of this exercise is to design an output layer that can convert a 10-sigmoid output layer, where the activated neuron shows which decimal digit from the MNIST dataset is being recognized, into a 4-sigmoid layer which corresponds to the binary representation (using 4 bits) of the previously mentioned decimal digits, which range from 0 to 9.

For this task we will implement the classical decimal to binary mapping, represented in the following picture:

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Figure 2: Mapping of the first 10 decimal numbers into binary

The four neurons located in the new output layer will consist of 10 input values x_i where $i \in \{0, 9\}$; the corresponding weights, w_i, for each of these inputs; and the bias of the output neuron, b.

Then, the output value of these neurons will be calculated with the usual equation $y = \sigma(z) = \sigma(\mathbf{w}^\mathsf{T}\mathbf{x} + b)$; where $\mathbf{w}$ and $\mathbf{x}$ correspond to the inputs and weights in vector form.

For simplicity, we will set $b = 0$ as the bias for all four output neurons. Furthermore, in order to propose some weights, we need to take into consideration that in this case the correct outputs have an activation of at least 0.99, whereas the wrong detections output an activation of less than 0.01.

As the difference between the correct and wrong outputs is so great, one of the input x_i at the last layer will be dominant over the rest. In order to use this, the weight vectors have been designed so that the bits which must be 1 are weighted with $w = 1$, whereas the bits which must be 0 use $w = 0$.

Therefore, the weight vectors used are the following:

		w_0	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9
LSB	Output Neuron 1	-1	1	-1	1	-1	1	-1	1	-1	1
	Output Neuron 2	-1	-1	1	1	-1	-1	1	1	-1	-1
	Output Neuron 3	-1	-1	-1	-1	1	1	1	1	-1	-1
MSB	Output Neuron 4	-1	-1	-1	-1	-1	-1	-1	-1	1	1

The table represents the used weights. Output neuron 1 calculates the least significant bit, which is 1 for the odd numbers. Therefore, a weight of 1 is applied to the inputs which correspond to

odd numbers and -1 to the even inputs. Analogous logic has been used for the other three neurons. The following example should further clarify this result.

Example: This example shows how the digit 9 can be converted into the sequence 1001 in the last layer of the neural network. The third layer of the network would output a vector with the following values: $x = (0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.99)$. Then,this vector is multiplied by the weights in each of the output neurons (as the bias is 0, there is no need to add anything) , obtaining the following values:

Output Neuron 1	=	0.98	$\sigma(0.98) > 0.5 \Rightarrow 1$
Output Neuron 2	=	-1	$\sigma(0.98) > 0.5 \Rightarrow 0$
Output Neuron 3	=	-1	$\sigma(0.98) > 0.5 \Rightarrow 0$
Output Neuron 4	=	0.92	$\sigma(0.98) > 0.5 \Rightarrow 1$

As expected, the final output of the network is the sequence 1001, which corresponds to 9 in binary.

Problem 4: Steepest descent direction

The goal of this exercise is to prove that the vector $\mathbf{v}$ that minimizes $(\nabla C)^{\mathsf{T}}\mathbf{v}$, under the constraint $\|\mathbf{v}\| = \epsilon$, in the gradient descent problem is given by $\mathbf{v} = -\eta \cdot \nabla C$, where $\eta = \epsilon/\|\nabla C\|$.

Recall,

$$\mathbf{v} = [v_1, ..., v_n]^{\mathsf{T}} \tag{7}$$

$$\nabla C = \left[\frac{\partial C}{\partial v_1}, ..., \frac{\partial C}{\partial v_n}\right]^{\mathsf{T}}. \tag{8}$$

Using the Cauchy-Schwarz inequality we obtain the inequality:

$$\|(\nabla C)^{\mathsf{T}}\mathbf{v}\| \leq \|\nabla C\|\|\mathbf{v}\|. \tag{9}$$

so that

$$\min_{\mathbf{v}} \left[(\nabla C)^{\mathsf{T}}\mathbf{v}\right] \geq -\|\nabla C\| \max_{\mathbf{v}} \|\mathbf{v}\| = -\|\nabla C\|\epsilon. \tag{10}$$

Can we achieve this bound with equality? The answer is 'yes'. Indeed, let $\mathbf{v} = -\eta \nabla C$, with $\eta = \epsilon/\|\nabla C\|$. Then,

$$(\nabla C)^{\mathsf{T}}\mathbf{v} = -\eta(\nabla C)^{\mathsf{T}} \nabla C = -\frac{\epsilon\|\nabla C\|^2}{\|\nabla C\|} = -\|\nabla C\|\epsilon. \tag{11}$$

We conclude that $\mathbf{v} = -\eta \nabla C$, with $\eta = \epsilon/\|\nabla C\|$ is indeed the minimizer, i.e. it defines the steepest direction of descent.

Problem 5: Proof of 3rd and 4th equations of backpropagation

The goal of this exercise is to prove the third and fourth fundamental equations of the backpropagation algorithm.

3rd equation of backpropagation: In this section we will give a proof for the third equation of backpropagation, which is used to calculate the rate of change of the cost function with respect to the bias:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{12}$$

Let's remember the definition of the error, δ:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} \tag{13}$$

where

$$\mathbf{z}^l = \mathbf{W}\mathbf{a}^{l-1} + \mathbf{b}^l. \tag{14}$$

The first step is to apply the chain rule to the partial derivate of the cost function:

$$\frac{\partial C}{\partial b_j^l} = \sum_k \frac{\partial C}{\partial z_k^l}\frac{\partial z_k^l}{\partial b_j^l}. \tag{15}$$

From (14), we see that the partial derivate of z_k^l with respect to b_j^l is equals to 1 when $k = j$ and it is 0 when $k \neq j$. Therefor we can rewrite (15) as:

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l}.$$

According to (13), the term on the right hand side is just δ_j^l, so that

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l.$$

as desired.

4th equation of backpropagation: In this section we will prove the fourth equation of backpropagation, which is used to calculate the rate of change with respect to any weight:

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \cdot \delta_j^l.$$

Again, begin by using chain rule:

$$\frac{\partial C}{\partial w_{jk}^l} = \sum_u \frac{\partial C}{\partial z_u^l}\frac{\partial z_u^l}{\partial w_{jk}^l} \tag{16}$$

Using the definition of $\mathbf{z}$ in (14) we can simplify the second term above as follows:

$$\frac{\partial z_u^l}{\partial w_{jk}^l} = \frac{\partial}{\partial w_{jk}^l}\left(\sum_v w_{uv}^l a_v^{l-1} + b_u^l\right) = \begin{cases} a_k^{l-1} & \text{if } j = u \\ 0 & \text{otherwise.} \end{cases} \tag{17}$$

so that (16) becomes:

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \cdot \frac{\partial C}{\partial z_j^l}. \tag{18}$$

Using (13), we see that the term in the right side of the equation is equal to δ_j^l so that we obtain the desired formula:

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \cdot \delta_j^l. \tag{19}$$

Problem 6: Cross-entropy as a cost function

The goal of this problem is to show that the cross-entropy function

$$C(a_1, \dots a_n) = -\frac{1}{n} \sum_x [y_x \ln a_x + (1 - a_x) \ln(1 - a_x)] \tag{20}$$

is minimized when $a_x = y_x$ for all x and that, in this case, the value of the function is:

$$C = -\frac{1}{n} \sum_x [y_x \cdot \ln(y_x) + (1 - y_x) \cdot \ln(1 - y_x)]. \tag{21}$$

In order to minimize the function, compute the first derivate and set the resulting expression to zero. As all the terms of the sum are nonnegative and follow the same expression, the calculations can be made for one of the terms without loss of generality. Consider:

$$C(a) = -[y \ln a + (1 - a) \ln(1 - a)]. \tag{22}$$

Then,

$$\frac{\partial C}{\partial a} = \frac{\partial}{\partial a}[-[y \cdot \ln(a) + (1 - y) \cdot \ln(1 - a)]] \tag{23}$$

$$\frac{\partial C}{\partial a} = \frac{y}{a} - \frac{1 - y}{1 - a}. \tag{24}$$

Hence, $\frac{\partial C}{\partial a} = 0$ iff $\frac{y}{a} = \frac{1-y}{1-a}$ or, equivalently, iff $a = y$

This proves that the cross-entropy has a minimum when $a = y$. Therefore, substituting a for y in the definition of cross-entropy, we conclude that the minimum is given by (21) as desired.

Problem 7: Cost function and learning slowdown

1. The goal of this part of the exercise is to prove the following equality:

$$\frac{\partial C}{\partial w_{jk}^L} = \frac{1}{n} \cdot \sum_x a_k^{L-1} \cdot (a_j^L - y_j) \cdot \sigma'(z_j^L) \tag{25}$$

Start with the 4th fundamental equation of backpropagation, and substitute δ by its definition:

$$\frac{\partial C}{\partial w_{jk}^L} = a_k^{L-1} \cdot \delta_j = a_k^{L-1} \cdot \frac{\partial C}{\partial z_j^L} \tag{26}$$

Next, recall that the quadratic cost function is given by:

$$C = \frac{1}{2n} \cdot \sum_x \left\| y(x) - a^L(x) \right\|^2$$

where

$$a^L = \sigma(z^L) = \frac{1}{1 + e^{-z^L}}.$$

Next, apply the chain rule to the partial derivative in (26).

$$\frac{\partial C}{\partial z_j^L} = a_k^{L-1} \cdot \sum_m \frac{\partial C}{\partial a_m^L} \cdot \frac{\partial a_m^L}{\partial z_j^L} = a_k^{L-1} \cdot \frac{\partial C}{\partial a_j^L} \cdot \frac{\partial a_j^L}{\partial z_j^L}$$

Above, the second equality follows because all the derivatives are zero unless $m = j$.

Substituting the values of the partial derivatives we obtain

$$\frac{\partial C}{\partial w_{jk}^L} = a_k^{L-1} \cdot -2 \cdot \frac{1}{2n} \sum_x (y_j - a_j^L) \cdot \sigma'(z_j^L)$$

By rearranging and simplifying the previous equation we obtain the required formula:

$$\frac{\partial C}{\partial w_{jk}^L} = \frac{1}{n} \sum_x a_k^{L-1} (a_j^L - y_j) \sigma'(z_j^L).$$

2. Now consider the cross-entropy cost function. The objective of this part of the exercise is to prove that in this case the partial derivate of the cost function is given by:

$$\frac{\partial C}{\partial w_{jk}^L} = \frac{1}{n} \sum_x a_k^{L-1} (a_j^L - y_j). \tag{27}$$

In particular, it does not depend on the derivative of the sigmoid.

The first use the chain rule:

$$\frac{\partial C}{\partial w_{jk}^L} = \sum_m \frac{\partial C}{\partial a_m^L} \cdot \frac{\partial a_m^L}{\partial w_{jk}^L}. \tag{28}$$

Observe, when $m \neq j$ the $\frac{\partial a_m^L}{\partial w_{jk}^L} = 0$, so we can rewrite the expression as:

$$\frac{\partial C}{\partial w_{jk}^L} = \frac{\partial C}{\partial a_j^L} \cdot \frac{\partial a_j^L}{\partial w_{jk}^L}. \tag{29}$$

Next, we substitute the cross-entropy cost function in the equation and evaluate the partial derivatives with respect to a_j^L obtaining the following result:

$$\frac{\partial C}{\partial w_{jk}^L} = \frac{\partial}{\partial a_j^L}\left[-\frac{1}{n}\cdot\sum_x\sum_u\left(y_u\cdot\ln(a_u^L)+(1-y_u)\cdot\ln(1-a_u^L)\right)\right]\cdot\frac{\partial a_j^L}{\partial w_{jk}^L} \tag{30}$$

$$\frac{\partial C}{\partial w_{jk}^L} = -\frac{1}{n}\cdot\sum_x\left(\frac{y_j}{a_j^L}-\frac{1-y_j}{1-a_j^L}\right)\cdot\frac{\partial a_j^L}{\partial w_{jk}^L}. \tag{31}$$

Next,

$$a_j^L = \sigma(z_j^L)$$

so that

$$\frac{\partial a_j^L}{\partial w_{jk}^L} = \sigma'(z_j^L)x_k^L = \sigma(z_j^L)(1-\sigma(z_j^L))x_k^L = a_j^L(1-a_j^L)x_k^L. \tag{32}$$

where we used that $\sigma'(z) = \sigma(z)\cdot(1-\sigma(z))$ as we have seen in the lecture. Substituting (32) into (31) we obtain:

$$\frac{\partial C}{\partial w_{jk}^L} = -\frac{1}{n}\cdot\sum_x\left(\frac{y_j}{a_j^L}-\frac{1-y_j}{1-a_j^L}\right)a_j^L(1-a_j^L)x_k^L = -\frac{1}{n}\cdot\sum_x\left(y_j-a_j^L\right)x_k^L. \tag{33}$$

Remember that x_k^L is the input of the layer L which is equals to the output of the layer $L-1$, which is a^{L-1}. Therefore,

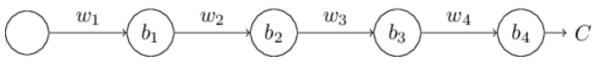
$$\frac{\partial C}{\partial w_{jk}^L} = -\frac{1}{n}\cdot\sum_x\left(\frac{y_j}{a_j^L}-\frac{1-y_j}{1-a_j^L}\right)a_j^L(1-a_j^L)x_k^L = -\frac{1}{n}\cdot\sum_x\left(y_j-a_j^L\right)a_k^{L-1}. \tag{34}$$

as desired.

Problem 8: Vanishing gradients

We now consider the problem of the vanishing gradient, with an activation function that is much larger than the example given in Lecture 21, i.e. $|\sigma'(z)| \gg 1/4$ for some z. Would this help us solve the unstable gradient problem?

Let's return to the simple example considered in the Lecture 21:



Comparing the derivatives of the cost with respect to biases at two different layers we find:

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1)\, w_2\sigma'(z_2)\; w_3\sigma'(z_3)\; \underbrace{w_4\sigma'(z_4)\frac{\partial C}{\partial a_4}}$$

$$\overbrace{\frac{\partial C}{\partial b_3} = \sigma'(z_3)\, w_4\sigma'(z_4)\frac{\partial C}{\partial a_4}}$$

common terms

The fundamental problem here isn't so much the vanishing gradient problem. It's that the gradient in early layers is the product of terms from all the later layers. When there are many layers, that's an intrinsically unstable situation. The only way all layers can learn at close to the same speed is if all those products of terms come close to balancing out. Without some mechanism or underlying reason for that balancing to occur, it's highly unlikely to happen simply by chance. In short, the real problem here is that neural networks suffer from an unstable gradient problem. As a result, if we use standard gradient-based learning techniques, different layers in the network will tend to learn at wildly different speeds.

Problem 10: Convex optimization problems (exam practice)

1. In this case the cost is not a differentiable function of the parameters (w's and b's). Non-differentiable functions cannot be optimized by gradient descent.

2. Vanishing or exploding gradients. If you do the calculation of the gradient of the cost function with respect to the parameters, you will see that the expression is a product of basic parts. The deeper you go into the network the more elements there are in the product. This leads to exponential decay or increase (depending on the initial conditions) of the gradient as a function of the depth of the network.