Introduction to Artificial Intelligence (INFO8006)

Exercise session 6

Multilayer perceptron

For an input vector $\mathbf{x} \in \mathbb{R}^d$ and activation function $\sigma(.)$, a MLP with L layers can be written as

$$\begin{aligned}\mathbf{h}_0 &= \mathbf{x} \\ \mathbf{h}_1 &= \sigma(\mathbf{W}_1^T \mathbf{h}_0 + \mathbf{b}_1) \\ &\vdots \\ \mathbf{h}_L &= \sigma(\mathbf{W}_L^T \mathbf{h}_{L-1} + \mathbf{b}_L) \\ \mathbf{y} &= \mathbf{h}_L\end{aligned}$$

with $\mathbf{h}_l \in \mathbb{R}^{q_l}$ defined as the the hidden vector, $\mathbf{W}_l \in \mathbb{R}^{q_{l-1} \times q_l}$ the weight matrix and $\mathbf{b}_l \in \mathbb{R}^{q_l}$ the bias vector of the l-th layer.

Training loop

For a given input-output training pair (x, y), a neural network $\Phi_\theta(.)$ parameterized by θ, a loss function $\mathcal{L}(.)$ and a learning rate λ:

1. Compute all intermediate values in the network for the given input x up to the output prediction $\hat{y} = \Phi_\theta(x)$. This is known as the forward pass.
2. Compute the symbolic gradient of the loss w.r.t. every parameters $g_i(.) = \frac{\partial \mathcal{L}}{\partial \theta_i}$ using the chain rule.
3. Evaluate the loss for your training point as $\mathcal{L}(y, \hat{y})$.
4. Backpropagate the loss and the activations through the network to compute the gradient values g_i. This is known as the backward pass.
5. Update every parameter using their gradient as

$$\theta_i \leftarrow \theta_i - \lambda g_i.$$

Universal approximation theorem

The universal approximation theorem states that a single-hidden-layer network with a finite number of hidden units can approximate any continuous function on a compact subset of $\mathbb{R}^d$ to arbitrary accuracy.

In session exercices: Ex. 1, Ex. 2, Ex. 5

Exercise 1 Escape game (January 2022)

A new virtual escape game came out, and you decide to play it. You arrive in a 5×5 grid world where each cell (x, y) is a room with doors leading to the adjacent rooms. The game's goal is to reach the exit room as fast as possible, but its position is unknown. Furthermore, some regions of the world are full of riddles, and crossing rooms in these regions takes longer. Fortunately, a leaderboard with the players' best times is provided, starting from a few different rooms. Due to rounding errors, you assume that the best times reported in the leaderboard are measurements affected by additive Gaussian noise $\mathcal{N}(0, 1)$.

i	Starting room	Measured best time
1	$(4, 5)$	2.0
2	$(5, 3)$	3.5
3	$(3, 3)$	4.5
4	$(4, 1)$	7.0
5	$(1, 2)$	8.5

From the leaderboard, you wish to learn a heuristic approximating the best time to get to the exit, starting from room (x, y). You decide to use a small neural network as approximator, described by the following parametric function,

$$h(x, y; \phi) = \mathrm{ReLU}(xw_1 + yw_2 + w_3) + \mathrm{ReLU}(xw_4 + yw_5 + w_6)$$
$$\mathrm{ReLU}(x) = \max(x, 0),$$

where $\phi = (w_1, w_2, w_3, w_4, w_5, w_6)$ is the set of parameters/weights of the neural network.

1. Among the following sets of parameters (A, B or C), which one would you use? Justify your answer.

Set	w_1	w_2	w_3	w_4	w_5	w_6
ϕ_A	-1.5	1	4	1	-1.5	6
ϕ_B	-1	1.5	3	0	-1	4
ϕ_C	-2	0.5	4.5	1.5	0	5

A set of parameters is better than another if it explains the data better, that is if the likelihood is higher. Our task is to find the set of parameters that maximizes the likelihood under the assumed probability model. In our case, the data d is the leaderboard and consists of independent position-time tuples (x_i, y_i, t_i). From the statement, we gather that the (best) time t is a function of the starting position (x, y), approximated by a neural network $h(x, y; \phi)$. We also learn that our time measurements are affected by Gaussian noise.

$$\begin{aligned}
\phi_{\mathrm{MLE}} &= \arg\max_{\phi} p(d|\phi) \\
&= \arg\max_{\phi} \prod_{i=1}^{5} p(x_i, y_i, t_i|\phi) \\
&= \arg\max_{\phi} \prod_{i=1}^{5} p(t_i|x_i, y_i, \phi)\, p(x_i, y_i)
\end{aligned}$$

$$
\begin{aligned}
&= \arg\max_{\phi} \prod_{i=1}^{5} \mathcal{N}(t; h(x, y; \phi), 1) \\
&= \arg\max_{\phi} \sum_{i=1}^{5} \log\left[\frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(t_i - h(x_i, y_i; \phi))^2}{2}\right)\right] \\
&= \arg\min_{\phi} \sum_{i=1}^{5} (t_i - h(x_i, y_i; \phi))^2
\end{aligned}
$$

Now that we have an objective $L(\phi)$ to minimize, we can evaluate it for each of the proposed sets and select the best one.

$$
\begin{aligned}
L(\phi_A) &= \sum_{i=1}^{5} (t_i - h(x_i, y_i; \phi_A))^2 = (2.0 - 5.5)^2 + (3.5 - 6.5)^2 + \cdots = 29.75 \\
L(\phi_B) &= 35.75 \\
L(\phi_C) &= 205.25
\end{aligned}
$$

The best set is ϕ_A.

2. You now assume a Gaussian prior $\mathcal{N}(0, 1)$ on each parameter. Which set of parameters in the table above would you now choose? Justify your answer.

   Now that we hold a prior belief on the parameters, our task is to find the most likely set of parameters given the data, that is maximum a posteriori (MAP) estimation.

$$
\begin{aligned}
\phi_{\text{MAP}} &= \arg\max_{\phi} p(\phi|d) \\
&= \arg\max_{\phi} p(d|\phi)p(\phi) \\
&= \arg\max_{\phi} \prod_{i=1}^{5} p(x_i, y_i, t_i|\phi) \prod_{j=1}^{6} p(w_j) \\
&= \arg\max_{\phi} \prod_{i=1}^{5} \mathcal{N}(t; h(x, y; \phi), 1) \prod_{j=1}^{6} \mathcal{N}(w_j; 0, 1) \\
&= \arg\min_{\phi} \sum_{i=1}^{5} (t_i - h(x_i, y_i; \phi))^2 + \underbrace{\sum_{j=1}^{6} (w_j)^2}_{\|\phi\|^2}
\end{aligned}
$$

   Here again, we have an objective $L'(\phi)$ to minimize, which we evaluate for each of the proposed sets.

$$
\begin{aligned}
L'(\phi_A) &= L(\phi_A) + \sum_{j=1}^{6} (w_j)^2 = 29.75 + (-1.5)^2 + (1)^2 + \cdots = 88.25 \\
L'(\phi_B) &= 65.0 \\
L'(\phi_C) &= 257.0
\end{aligned}
$$

   The best set is ϕ_B.

3. Discuss the procedure you would implement on a computer to find the optimal set of parameters, had the table above not been provided.

   If the table is not provided, it is intractable to evaluate and compare all possible sets of parameters. However, we still want to minimize the MLE (or MAP) objective $L(\phi)$. We

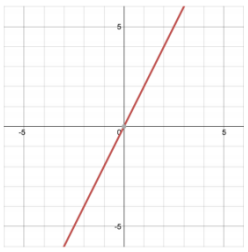
can improve a set of parameters ϕ by following the opposite of the objective's gradient $\nabla_\phi L(\phi)$, *i.e.* performing *gradient descent* steps

$$\phi \leftarrow \phi - \gamma \nabla_\phi L(\phi),$$

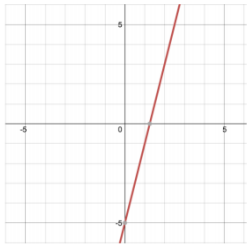
where γ is the learning rate.

Exercise 2 Neural network representations (CS188, Spring 2024)

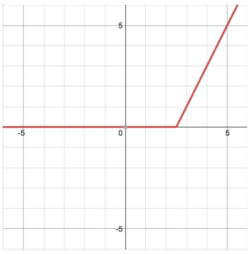
You are given a number of functions (a-h) of a single variable, x, which are graphed below. For each network structure proposed afterwards, indicate which functions they are able to represent. When possible, indicate appropriate values for the parameters.



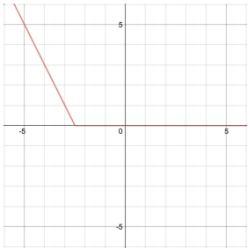
(a) $2x$



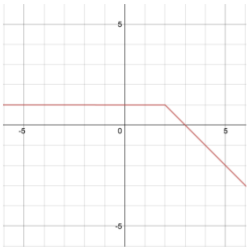
(b) $4x - 5$



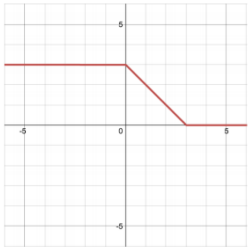
(c) $\begin{cases} 2x-5 & x \geq 2.5 \\ 0 & x < 2.5 \end{cases}$



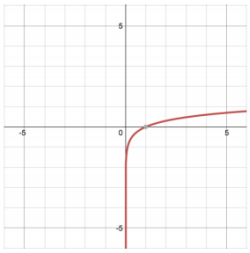
(d) $\begin{cases} -2x-5 & x \leq -2.5 \\ 0 & x > -2.5 \end{cases}$



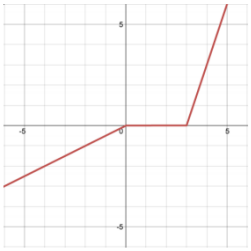
(e) $\begin{cases} -x+3 & x \geq 2 \\ 1 & x < 2 \end{cases}$



(f) $\begin{cases} 3 & x \leq 0 \\ 3-x & 0 < x \leq 3 \\ 0 & x > 3 \end{cases}$

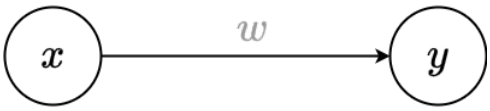


(g) $\log(x)$



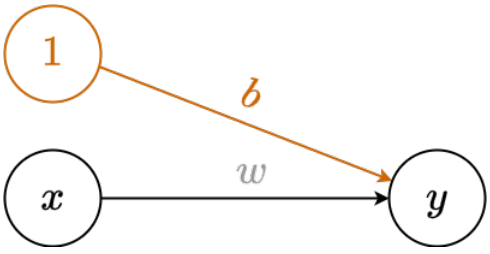
(h) $\begin{cases} 0.5x & x \leq 0 \\ 0 & 0 < x \leq 3 \\ 3x-9 & x > 3 \end{cases}$

1. We start with a linear transformation of the scalar input x, weight w, and output y, such that $y = wx$.



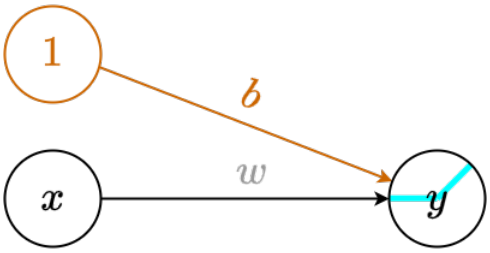
This network can only represent (a), with $w = 2$. Since there is no bias term, the line must pass through the origin.

2. We then introduce a bias term b, such that $y = wx + b$.



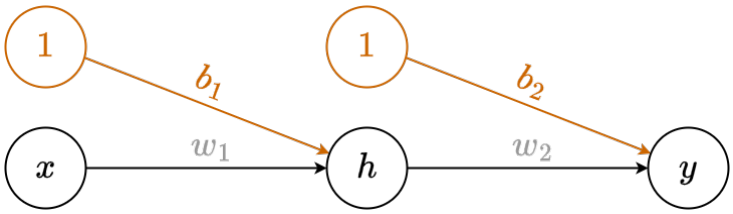
(a) with $w = 2$ and $b = 0$, and (b) with $w = 4$ and $b = -5$.

3. We now introduce a non-linearity $\sigma(.)$ into the network. We use the ReLU non-linearity, which has the form ReLU(x) = max(0, x).



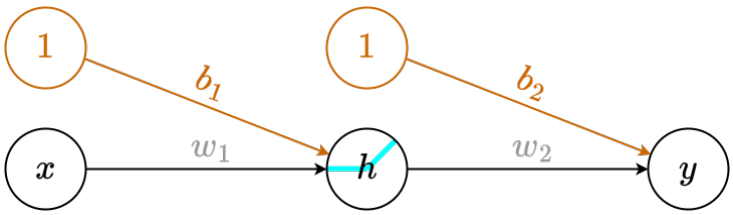
With the output coming directly from the ReLU, this cannot produce any values less than zero. It can produce (c) with $w = 2$ and $b = -5$, and (d) with $w = -2$ and $b = -5$.

4. Now we consider neural networks with multiple affine transformations, as depicted below.



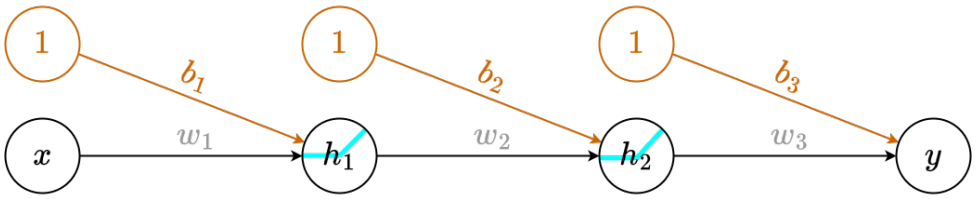
Applying multiple affine transformations (with no non-linearity in between) is not any more powerful than a single affine function: $w_2(w_1x + b_1) + b_2 = w_2w_1x + w_2b_1 + b_2$, so this is just a affine function with different coefficients. The functions we can represent are the same as in 1, if we choose $w_1 = w$, $w_2 = 1$, $b_1 = 0$, $b_2 = b$: (a) with $w_1 = 2$, $w_2 = 1$, $b_1 = 0$, $b_2 = 0$, and (b) with $w_1 = 4$, $w_2 = 1$, $b_1 = 0$, $b_2 = -5$.

5. We now add a ReLU non-linearity to the network between the affine transformations.



(c), (d), and (e). The affine transformation after the ReLU is capable of stretching (or flipping) and shifting the ReLU output in the vertical dimension. The parameters to produce these are: (c) with $w_1 = 2$, $b_1 = -5$, $w_2 = 1$, $b_2 = 0$, (d) with $w_1 = -2$, $b_1 = -5$, $w_2 = 1$, $b_2 = 0$, and (e) with $w_1 = 1$, $b_1 = -2$, $w_2 = -1$, $b_2 = 1$.

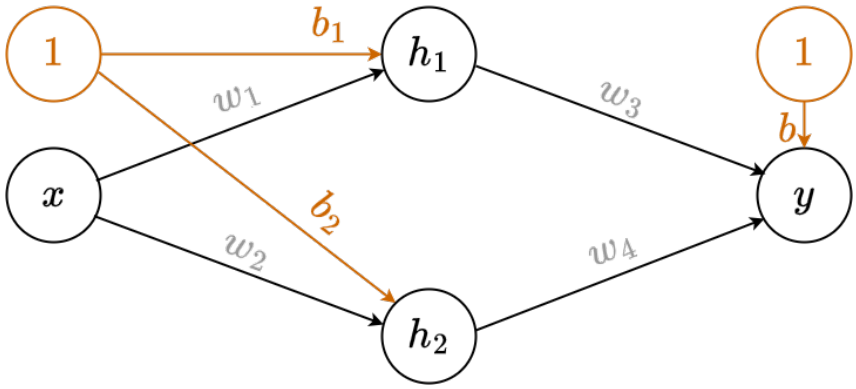
6. Now we add another hidden layer to the network, as depicted below. You do not have to guess parameters values anymore.



(c), (d), (e), and (f). The network can represent all the same functions as Q5 (because note that we could have $w_2 = 1$ and $b_2 = 0$). In addition it can represent (f): the first

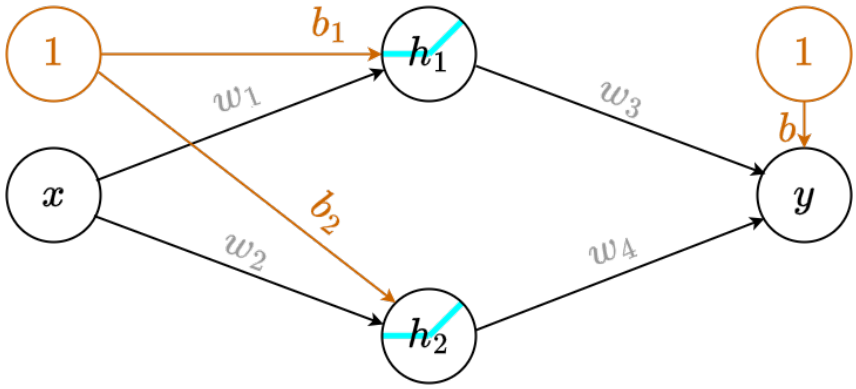
ReLU can produce the first flat segment, the affine transformation can flip and shift the resulting curve, and then the second ReLU can produce the second flat segment (with the final affine layer not doing anything). Note that (h) cannot be produced since its line has only one flat segment (and the affine layers can only scale, shift, and flip the graph in the vertical dimension; they can't rotate the graph).

7. We'd like to consider using a neural net with just one hidden layer, but larger. Let's first consider using just two affine functions, with no nonlinearity in between. You do not have to guess parameters values anymore.



(a) and (b). With no non-linearity, this reduces to a single affine function (in the same way as Q4).

8. Now we'll add a non-linearity between the two affine layers, to produce the neural network below with a hidden layer of size 2. You do not have to guess parameters values anymore.



All functions except for (g). Note that we can recreate any network from (5) by setting w_4 to 0, so this allows us to produce (c), (d) and (e). To produce the rest of the functions, note that h_1 and h_2 will be two independent functions with a flat part lying on the x-axis, and a portion with positive slope. The final layer takes a weighted sum of these two functions. To produce (a) and (b), the flat portion of one ReLU should start at the point where the other ends ($x = 0$ for (a), or $x = 1$ for (b). The final layer then vertically flips the ReLU sloping down and adds it to the one sloping up, producing a single sloped line. To produce (h), the ReLU sloping down should have its flat portion end (at $x = 0$ before the other's flat portion begins (at $x = 3$). The down-sloping one is again flipped and added to the up-sloping. To produce (f), both ReLUs should have equal slope, which will cancel to produce the first flat portion above the x-axis.

9. Are there functions that can't be represented by all proposed networks? If yes, explain why and what you would need to model them.

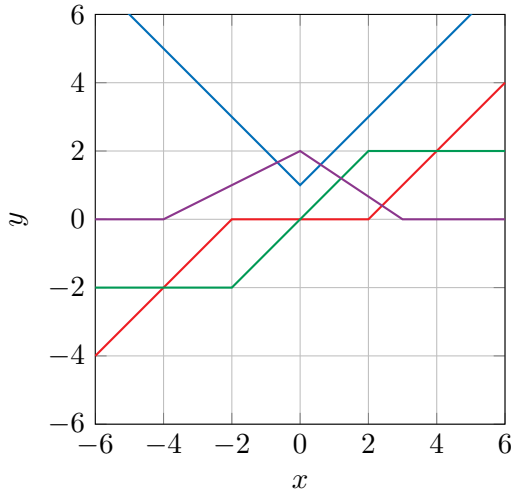
   Yes, function (g) cannot be modeled. We first see that the function is smooth and representing it only with few ReLU activations is not enough. We should probably use smoother activations like SiLU, tanh, ... (see slide 48 of the "Machine learning and neural networks" lecture).

   However, we remember the universal approximation theorem which states that a single-hidden-layer network with a finite number of hidden units can approximate any continuous function on a compact subset of $\mathbb{R}^d$ to arbitrary accuracy. Then, adding sufficient amount of hidden units of the 1-layer network of part 8 should be sufficient.

   Note that the logarithm function is only defined on $\mathbb{R}^+$. If our network is made of activation functions defined on $\mathbb{R}$, we will never be able to restrict the domain of the learned function to only real positive. However, you can be as accurate as you want on the desired domain, but it makes no sense to evaluate your network elsewhere for this problem.

Exercise 3 ReLU

For each of the piecewise-linear functions below, write a function $y = f(x)$ as a composition of sums $(+, -)$, ReLU ($\text{ReLU}(x) = \max(x, 0)$) non-linearities, and real parameters (weights and biases) that exactly matches the function over $\mathbb{R}$.

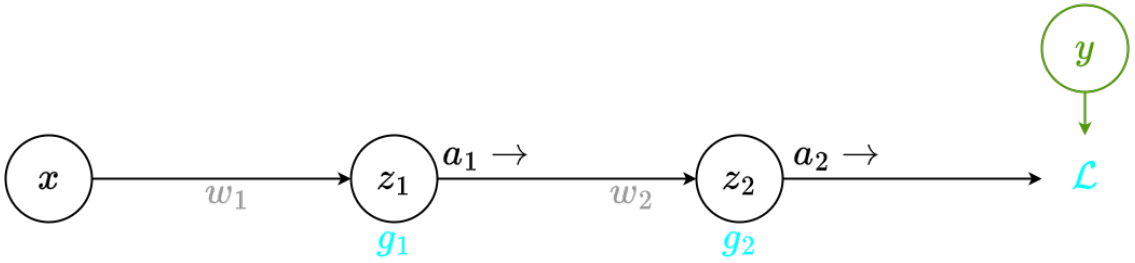


For example, $y = \text{ReLU}(x + 2) - \text{ReLU}(-2x)$ is a valid function.

1. $y = \text{ReLU}(x) - \text{ReLU}(-x) + 1$
2. $y = \text{ReLU}(x - 2) - \text{ReLU}(-x - 2)$
3. $y = \text{ReLU}(-\text{ReLU}(-x + 2) + 4) - 2$
4. $y = \text{ReLU}(-\text{ReLU}(\frac{2}{3}x) - \text{ReLU}(-\frac{1}{2}x) + 2)$

Exercise 4 Classification (CS188, Spring 2024)

Consider the following simple neural network for binary classification. Here x is a single real-valued input feature with an associated class y (0 or 1). There are two weight parameters w_1 and w_2, and non-linearity functions g_1 and g_2. The network will output a value a_2 between 0 and 1, representing the probability of being in class 1. We will be using a loss function $\mathcal{L}$ to compare the prediction a_2 with the true class y.



1. Write down the forward pass on this network, writing the output values for each node z_1, a_1, z_2 and a_2 in terms of the node's input values.

$$
\begin{aligned}
z_1 &= w_1 x \\
a_1 &= g_1(z_1) = g_1(w_1 x) \\
z_2 &= w_2 a_1 = w_2 g_1(w_1 x) \\
a_2 &= g_2(z_2) = g_2(w_2 g_1(w_1 x))
\end{aligned}
$$

2. Derive the arguments of the loss $\mathcal{L}(a_2, y)$ in terms of the input x, weights w_i, and activation functions g_i.

   The argument a_2 has been derived in the previous question. We have

   $$a_2 = g_2(w_2 g_1(w_1 x)).$$

   For the other input y, the latter is given as input of the problem. Indeed, in supervised learning, we assume access to a training dataset made of input-output pairs $\{(x_i, y_i)\}^N$ where y_i acts as a target for the network.

3. Using the chain rule, differentiate $\mathcal{L}$ w.r.t. w_2. Write your expression as a product of partial derivatives at each node: i.e. the partial derivative of the node's output with respect to its inputs. (Hint: the series of expressions you wrote in part 1 will be helpful; you may use any of those variables.)

   $$\frac{\partial \mathcal{L}}{\partial w_2} = \frac{\partial \mathcal{L}}{\partial a_2} \frac{\partial a_2}{\partial z_2} \frac{\partial z_2}{\partial w_2}$$

4. Motivate a choice for the output activation function g_2 given the nature of the problem. Motivate also a loss function. Give the expression of the gradients of both functions w.r.t. to their inputs.

   From the statement

   (a) we know that we are in a classification problem (specifically, a binary classification),

(b) we know that the output a_2 should be between 0 and 1 and represents the probability $p(Y = 1 \mid x)$.

From (b), we have $a_2 : \mathbb{R} \mapsto [0, 1]$. A typical choice that fulfill this requirement is the sigmoid function $\sigma(z) = \dfrac{1}{1 + e^{-z}}$. The gradient of the sigmoid w.r.t. its input is

$$\frac{\partial \sigma(z)}{\partial z} = \frac{e^{-z}}{(1 + e^{-z})^2} = \sigma(z)\sigma(-z) = \sigma(z)(1 - \sigma(z)).$$

For the loss function, either we remember the cross-entropy loss seen in the theoretical lecture, or we compute it from maximum likelihood estimation.

The likelihood of a pair (x, y) can be derived from our model as

$$\begin{cases} \text{if } y = 1 \quad \text{then} \quad P(Y = y \mid x) = \mathrm{NN}_\theta(x) \\ \text{if } y = 0 \quad \text{then} \quad P(Y = y \mid x) = 1 - \mathrm{NN}_\theta(x) \end{cases}$$

where NN_θ is the neural network parameterized by $\theta = [w_1, w_2]$. Denoting $\mathrm{NN}_\theta(x) = a_2$, the likelihood expression for one pair can be reformulated as

$$L(x, y; \theta) = y \log a_2 + (1 - y) \log(1 - a_2).$$

Since we want to maximize the likelihood, but minimize the loss, we can set the latter to

$$\mathcal{L}(a_2, y) = -y \log a_2 - (1 - y) \log(1 - a_2)$$

which corresponds to the binary cross-entropy loss. The gradient of the loss w.r.t. a_2 is

$$\frac{\partial \mathcal{L}(a_2, y)}{\partial a_2} = -\frac{y}{a_2} + \frac{(1 - y)}{(1 - a_2)} = \frac{a_2 - y}{a_2(1 - a_2)}.$$

5. We set the loss function to the binary cross-entropy

$$\mathcal{L}(a_2, y) = -y \log a_2 - (1 - y) \log(1 - a_2),$$

and g_1 and g_2 are both sigmoid functions $\sigma(z) = \frac{1}{1+e^{-z}}$. If you made this choice in the previous question, use your results to get the expression of $\frac{\partial \mathcal{L}}{\partial w_2}$. Otherwise, start by expressing $\frac{\partial \sigma(z)}{\partial z}$ and $\frac{\partial \mathcal{L}}{\partial a_2}$.

We first miss $\frac{\partial z_2}{\partial w_2}$ which is

$$\frac{\partial z_2}{\partial w_2} = \frac{\partial}{\partial w_2}(w_2 a_1) = a_1.$$

Plugging all the partial derivatives to the expression found in part 3, we have

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w_2} &= \frac{a_2 - y}{a_2(1 - a_2)} g_2(z_2)(1 - g_2(z_2)) a_1 \\ &= \frac{a_2 - y}{a_2(1 - a_2)} a_2(1 - a_2) a_1 \\ &= (a_2 - y) a_1 \end{aligned}$$

6. Now use the chain rule to express $\frac{\partial \mathcal{L}}{\partial w_1}$ as a product of partial derivatives at each node of interest.

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial a_2}\frac{\partial a_2}{\partial z_2}\frac{\partial z_2}{\partial a_1}\frac{\partial a_1}{\partial z_1}\frac{\partial z_1}{\partial w_1}$$

7. Finally, write $\dfrac{\partial \mathcal{L}}{\partial w_1}$ in terms of x, y, w_i, a_i, z_i.

   We first need to compute

$$\frac{\partial a_1}{\partial z_1} = g_1(z_1)(1 - g_1(z_1)) = a_1(1 - a_1)$$

   similarly to $\frac{\partial a_2}{\partial z_2}$, and

$$\frac{\partial z_1}{\partial w_1} = \frac{\partial}{\partial w_1}(w_1 x) = x.$$

   Then, we get

$$\frac{\partial \mathcal{L}}{\partial w_1} = (a_2 - y)w_2 a_1(1 - a_1)x.$$

8. What is the gradient descent update for w_1 with step-size λ in terms of the values computed above?

$$w_1 \leftarrow w_1 - \lambda(a_2 - y)w_2 a_1(1 - a_1)x.$$

   You see here the importance of the forward pass. Iteratively, when receiving the input x, we compute all the intermediate activations (here, a_1 and a_2) and the loss (given y). Then, we propagate backward (from the output to the input) the values following the chain rule for the gradient of the loss w.r.t. parameters of interest. Finally, we update the parameters using the computed gradient.

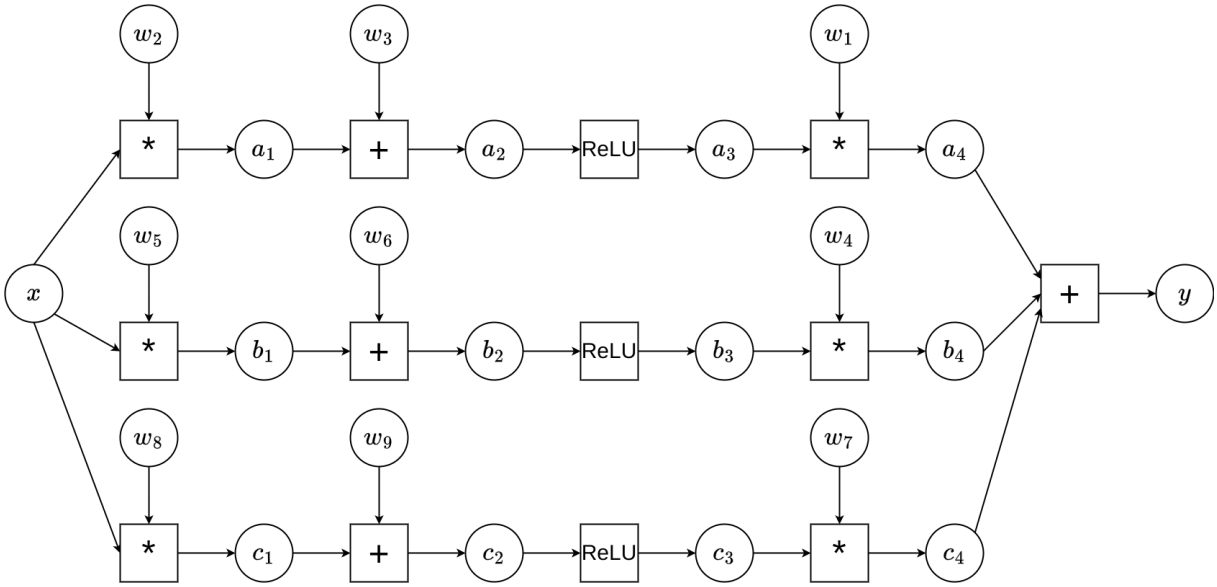
Exercise 5 Training loop (INFO8006, January 2024)

Let us consider a neural network f with one hidden layer taking as input a scalar $x \in \mathbb{R}$ and producing as output a scalar $y = f(x; \theta) \in \mathbb{R}$. The neural network is defined as

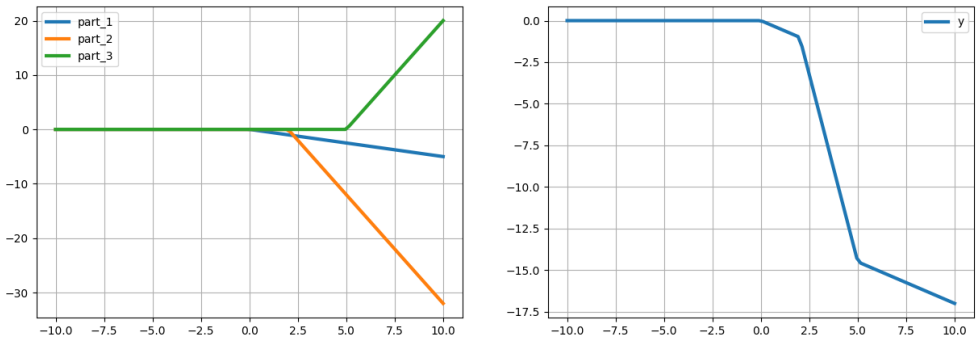
$$f(x; \theta) = w_1 \text{ReLU}(w_2 x + w_3) + w_4 \text{ReLU}(w_5 x + w_6) + w_7 \text{ReLU}(w_8 x + w_9),$$

where $\theta = (w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9)$ is the set of parameters and $\text{ReLU}(x) = \max(x, 0)$ is the rectified linear unit function.

1. Draw the computation graph representing the neural network and the flow of information from inputs to outputs. Your diagram should be a directed graph that follows the following conventions:
   - circled nodes correspond to variables (input, output, parameters or intermediate variables),
   - squared nodes correspond to primitive operations (addition, multiplication, ReLU) and produce an intermediate variable as output,
   - directed edges correspond to the flow of information, from inputs to outputs.



2. For $\theta = (-1, \frac{1}{2}, 0, -4, 1, -2, 4, 1, -5)$, draw the function $y = f(x; \theta)$ for $x \in [-10, 10]$.



3. Using the data point $(x, y) = (10, -15)$ and the value of θ given above, we want to fine-tune the parameter w_8 of the neural network such that $f(x; \theta)$ produces a more accurate prediction of y.

   (a) Evaluate the squared error loss at the data point $(x, y) = (10, -15)$ and the current value of θ.

$$f(10, \theta) = -17$$
$$\mathcal{L}(\hat{y} = -17, y = -15) = (17 - 15)^2 = 4$$

   (b) Derive an expression for the derivative of the squared error loss with respect to w_8.
   The loss is

$$\mathcal{L}(\hat{y}, y) = (\hat{y} - y)^2.$$

   We compute the gradient

$$\frac{\partial \mathcal{L}}{\partial w_8} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial c_4} \frac{\partial c_4}{\partial c_3} \frac{\partial c_3}{\partial c_2} \frac{\partial c_2}{\partial c_1} \frac{\partial c_1}{\partial w_8}$$
$$= 2(\hat{y} - y) w_7 \frac{\partial c_3}{\partial c_2} x.$$

   We need to compute forward activations to compute the gradient value. $\frac{\partial c_3}{\partial c_2}$ will be 1 if $c_2 > 0$ else 0 by definition of the ReLU. We have

$$\hat{y} = -17$$
$$y = -15$$
$$w_7 = 4$$
$$x = 10$$
$$c_2 = w_8 x + w_9 = 5 \Rightarrow \frac{\partial c_3}{\partial c_2} = 1$$

   which gives a gradient equal to -160.

   (c) Update the parameter w_8 using one step of gradient descent with a learning rate $\gamma = 0.0005$.

   The gradient step is given by

$$w_8 \leftarrow w_8 - \gamma \frac{\partial \mathcal{L}}{\partial w_8}$$

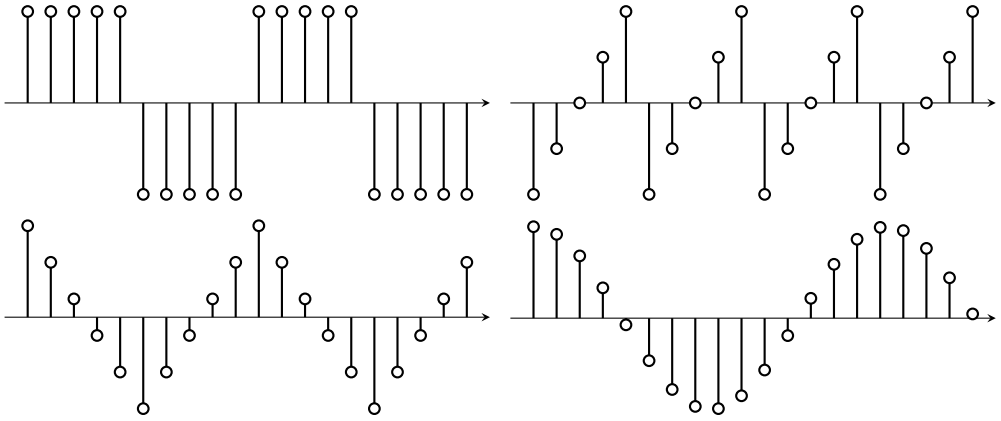
   which gives an updated w_8 equal to 1.08.

(d) Verify that the value of the loss function has decreased after the update.

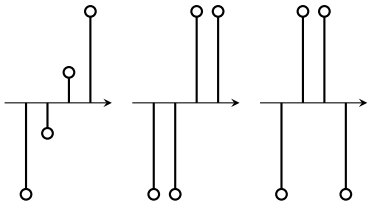
With the updated w_8, we have $\hat{y} = -13.8$ which gives a loss equal to 1.44. The previous loss value was 4, we indeed decreased the loss.

Exercise 6 Convolution

Represent the convolution $x \circledast u$ of each of the following signals x



with each of the following convolution kernels u.



Exercise 7 Learning to play Pacman (August 2020)

You observe a Grandmaster agent playing Pacman. How can you use the moves you observe to train your own agent?

1. Describe formally the data you would collect, the inference problem you would consider, and how you would solve it.
2. How would you design a neural network to control your agent? Define mathematically the neural network architecture, its inputs, its outputs, its parameters, as well as the loss you would use to train it.
3. Discuss the expected performance of the resulting agent when (a) the Grandmaster agent is optimal, and (b) the Grandmaster agent is suboptimal.

Quiz

We build a MLP with 2 hidden layers of 32 units, ReLU activations and squared error loss. We discard the biases from our linear layers. We know that our input is a vector of 2 elements and the output is a scalar. How much parameters do we have in the network?

- ☐ 1024.
- ■ 1120.
- ☐ 1185.
- ☐ Another value than the ones proposed above.

We add biases in the network proposed above. What is the number of parameters of the network?

- ☐ Another value than the ones proposed below.
- ☐ 1089.
- ■ 1185.
- ☐ 2048.
- ☐ 2240.

We have an image of shape (H, W), we want to pass it through a layer which reduces each dimension by a constant K, i.e. produces an output image of shape $(H - K, W - K)$.

- ☐ It can be done with a convolution layer with a kernel (K, K) or a linear layer with weight matrix $(HW, (H - K)(W - K))$.
- ■ It can be done with a convolution layer with a kernel $(K + 1, K + 1)$ or a linear layer with weight matrix $(HW, (H - K)(W - K))$.
- ☐ It can be done with a convolution layer with a kernel $(K - 1, K - 1)$ or a linear layer with weight matrix (HW, KK).
- ☐ It can be done with a convolution layer with a kernel $(K + 1, K + 1)$ or a linear layer with weight matrix (HW, KK).

From the previous question, we double the number of input pixels by setting the shape of the input image to (H', W) with $H' = 2H$. The latter implies that the output image should be of shape $(H' - K, W - K)$. If only look at weight parameters (no biases)...

- ☐ The number of parameters of both the linear and convolution layer are doubled.
- ☐ The linear layer will have twice more parameters and the convolution twice less.
- ■ The linear layer will have twice more parameters and the convolution will not change.
- ☐ Nothing changes neither for the linear layer nor the convolution one.

In deep learning, a layer in a multi-layer perceptron is defined as ...

- ☐ $\mathbf{h} = \sigma(\mathbf{W}^T\mathbf{x} + \mathbf{b})$, where σ is the standard deviation function.
- ■ $\mathbf{h} = \sigma(\mathbf{W}^T\mathbf{x} + \mathbf{b})$, where σ is an activation function, such as the sigmoid function.
- ☐ $\mathbf{h} = \sigma(\mathbf{W}^T + \mathbf{x} - \mathbf{b})$, where $\mathbf{W} \in \mathbb{R}^{d \times q}$ is matrix of weights.
- ☐ $\mathbf{h} = \sigma(\mathbf{W}^T\mathbf{x} + \mathbf{b})$, where $\mathbf{b} \in \mathbb{R}^d$ is the most likely vector of hidden states given $\mathbf{x}$.

Arnaud is trying to perform gradient descent on a function $f(x)$ using the update

$$x_{t+1} := x_t - \frac{\partial f}{\partial x}(x_t).$$

Is gradient descent guaranteed to converge to the global minimum of f?

- ☐ Yes, since he's updating using the gradient of x.
- ☐ Yes, but not for the reason above.
- ☐ No, since he is updating updating x in the wrong direction.
- ■ No, but not for the reason above.