Completed

Principles of Machine Learning

Exercises

Victor Verreet *victor.verreet@cs.kuleuven.be*
Laurens Devos *laurens.devos@cs.kuleuven.be*

Fall, 2021

Exercise Session 5: Artificial Neural Networks

5.1 Representation Power

Assume the inputs and outputs of a neural network are boolean values, represented by the numbers -1 and 1. If the activation function is the sign function

$$\sigma(x) = \begin{cases} -1 & x < 0 \\ 0 & x = 0 \\ 1 & x > 0 \end{cases}$$

then give a 2-input perceptron that represents the following functions, if possible.

1. $(A, B) \mapsto A \vee B$
2. $(A, B) \mapsto A \Leftrightarrow B$
3. $(A, B) \mapsto \neg A \wedge B$

Now consider a boolean function $f : \mathbb{B}^n \to \mathbb{B}$ in conjunctive normal form with k clauses

$$f(A_1, ..., A_n) = \bigwedge_{i=1}^{k} \bigvee_{j=1}^{n} f_{ij}(A_j)$$

where $f_{ij}(A) \in \{A, \neg A, \text{false}\}$ for all i and j. Give a neural network with two layers and n inputs that represents this function.

5.2 Decision Surfaces

Consider the decision surfaces in figure 1 below. For which of these surfaces can we find a 2-layer neural network that exactly represents this decision surface? The network takes 2-dimensional real input and uses the sign function as activation function in the hidden layer. Can you represent any convex polygon? Assume you know the equations of the individual lines.

5.3 Gradient Backpropagation

Consider the simple network in Figure 2 which has a single real input X, a hidden node H and an output Y. Let us use a sigmoid $s \mapsto (1 + e^{-s})^{-1}$ as activation function in the hidden layer and a linear function in the output layer. Write down H as a function of X and Y as a function of H. Using gradient backpropagation, calculate the gradient of Y with respect to every parameter in the network.

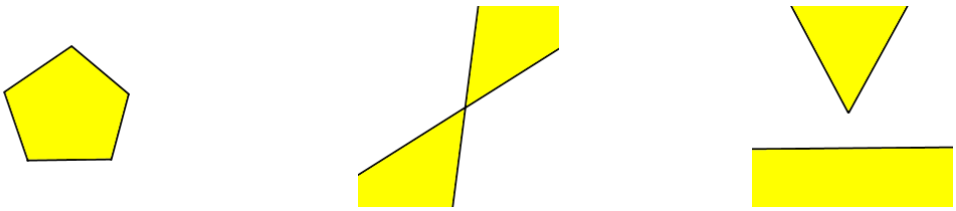


Figure 1: Three decision surfaces for 2-dimensional input.

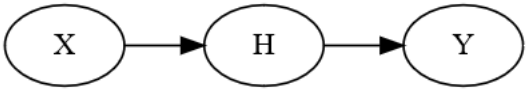


Figure 2: Simple neural network to illustrate backpropagation.

5.4 Stochastic Gradient Descent

The neural network in Figure 3 has a rectified linear unit ρ as activation function in each layer, which is defined as:

$$\rho(x) = \begin{cases} x & x > 0, \\ 0 & x \leq 0. \end{cases}$$

We will ignore the non-differentiable point at zero. The current values for the weights w are indicated on the edges and the offset terms q given in the nodes. We will perform stochastic gradient descent to learn the weights. The difference with regular gradient descent is that stochastic gradient descent only uses a randomly chosen subset of the data to calculate the gradient. In the next iteration, a new subset is again chosen at random. In our case, we will use only a single data point for one iteration. For input $(x_1, x_2) = (5, 3)$ we know the target value of $\hat{y} = 1$. Assume we have sampled this data point for stochastic gradient descent. Using the squared loss function $(y(w, q) - \hat{y})^2$, perform one update step on the weights w and offsets q, also called the bias term. Assume a learning rate η of unity.

Consider now the case that some weights are shared. If we require w_2 and w_3 to be the same and initialise this common weight to $w_2 = -1 = w_3$, what is the update to this weight?

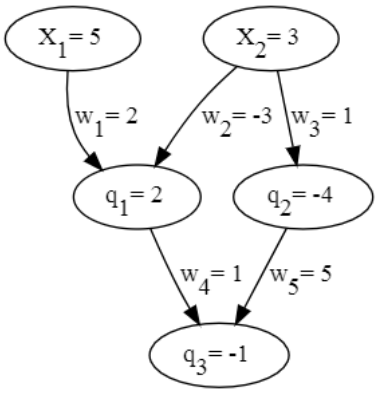


Figure 3: Neural network on which we can perform gradient descent.

5.5 Universal Approximation

Let us prove that neural networks are universal approximators for arbitrary continuous functions $f : [0, 1] \to \mathbb{R}$. For this purpose we make use of the known result that continuous functions on bounded closed intervals can be uniformly approximated to arbitrary precision by step functions of the form

$$S(x) = \sum_{n=1}^{N} S_n \chi[x \in [a_n, b_n)]$$

with $a_n < b_n$ for all n. In this notation χ is an indicator function, so $\chi[x \in [a,b)] = 1$ if $x \in [a,b)$ and zero otherwise. Convince yourself of this result. You do not need to prove it formally.

It thus sufficient to show that a neural network with a single in- and output and one hidden layer can approximate step functions to arbitrary precision. Consider neural networks with two layers, an activation function

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

in the hidden layer and a linear activation function in the output layer. Show that a suitable choice of the weights and offsets allows us to exactly represent any step function in the above form.

In fact, this proves the one-dimensional case of the universal approximation theorem. The general theorem states that any continuous function $f : \mathbb{R}^d \to \mathbb{R}$ in any dimension d can be approximated uniformly to arbitrary precision on compact sets by a neural network with only one hidden layer. The general proof is a bit more involved however. Furthermore notice that decision trees also have the power to represent arbitrary step functions and, by the same arguments, are universal approximators too.

5.6 Convolutional Neural Networks

We want to classify d by d images with k channels using a convolutional neural network. Our architecture consists of a convolutional layer λ_1 with n filters, each having kernel size $2s + 1$. The kernel size is the dimension of the square filter that is used in the convolution. Following this layer is a max pooling layer λ_2 with pool size l. If the input to our network is x, what are the dimensions of x, $\lambda_1(x)$ and $\lambda_2(\lambda_1(x))$? Assume that no padding is performed.

How many weights are present in this neural network? How many weights would there be if we used a fully connected layer instead of a convolutional layer, assuming the target dimensions remain the same? For a typical setting where $d = 64$, $k = 3$, $n = 5$, $s = 1$ and $l = 2$, what is the fractional reduction in the number of weights by choosing a convolutional layer over a fully connected layer?

5.7 Auto-Encoders

Consider an auto-encoder that encodes n-dimensional input into an encoded layer with $m < n$ nodes. If all the activation functions are linear, argue that training such an auto-encoder is equivalent to performing a principal component analysis on the data. How many components do you keep?

5.8 Hopfield Networks

Hopfield networks are special types of neural networks that try to mimic a memory. During training, patterns can be given, which will determine the connection weights of the network. When given a new input, the network finds the closest pattern to that input by running iterative updates on the input until it converges to some fixed state, hopefully one of the training patterns.

For example, if we want to recognize images representing digits, we can train a Hopfield network with one archetypical example for each digit. These will be stored as patterns. When given a new image, the network will evolve this input image until it converges to one of the archetype digits. This archetype can be used to classify the new image.

Like LSTM networks, Hopfield networks try to mimic memory. However, due to the update rule of Hopfield networks, so called spurious patterns can occur. These are unwanted extra patterns that are stored in the network. This is one of the reasons LSTM network have become more popular compared to Hopfield networks. Let us now look at how the evolution of a Hopfield network takes place.

For d-dimensional input, the constructed Hopfield network will have d fully connected neurons, each neuron representing one component of the input. The value of these neurons x will be updated from

time n to time $n+1$ according to the update rule

$$x^{(n+1)} = \alpha(Wx^{(n)} + q)$$

where α is the activation function, W is the weight matrix and q is the offset. For α one often chooses a saturated linear function

$$\alpha(x) = \begin{cases} -1 & x < -1 \\ 1 & x > 1 \\ x & \text{otherwise} \end{cases}$$

and the weights and offset are learned during the training phase. One way to set the weights is with the Hebbian learning rule, which states that for a set of patterns S to be learned, define the weights

$$W_{ij} = \sum_{s \in S} s_i s_j$$

and set the offset to zero.

Given a 2-dimensional input space and the patterns $S = \{(1,1),(1,-1),(-1,-1)\}$, apply the Hebbian learning rule to obtain the weight matrix W. Are the patterns fixed points of the update rule?

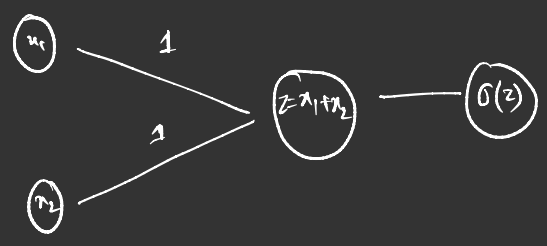
Now consider the input $(-1/2, 2/3)$ and perform three update steps on it. To what value does it converge? Can you explain this strange behaviour?

5.9 Keras

The file neural.py can be found online. Implement the missing method, used to define and train a convolutional neural network. You can then change the parameters and see what effect each has on the training process.

5.1

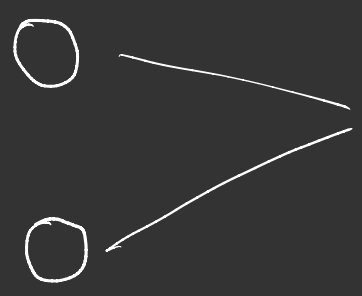
A	B	A ∨ B
0	0	0
1	0	1
0	1	1
1	1	1



$\rightarrow A \vee B$

(2)

A	B	A → B	B → A	A ⟺ B
1	1	1	1	1
1	0	0	1	0
0	1	1	0	0
0	0	1	1	1



$\therefore \omega_1 + \omega_2 + b > 0$

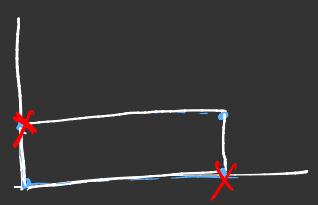
$b > 0$

$\therefore \omega_1 + b < 0$

$\therefore \omega_2 + b < 0$

$\Rightarrow b < -\omega_1$

$b < -\omega_2$

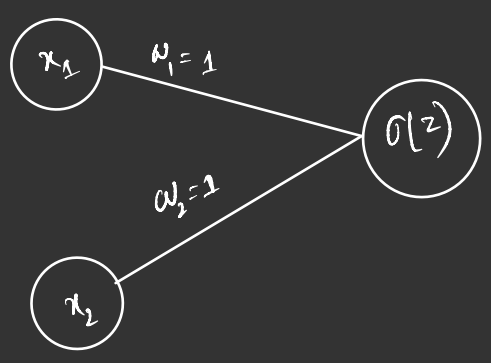


3)

A	B	A ∧ B
0	1	0
1	0	0
0	0	0
1	1	1

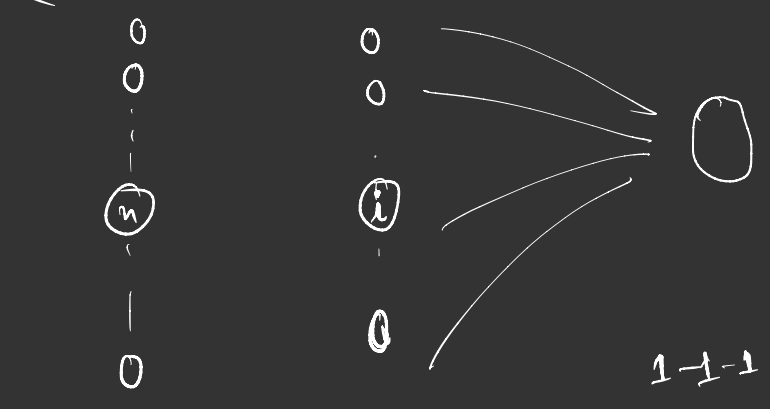
$z = \omega_1 + \omega_1 - 1.5$

$\sigma(z)$



$$f(A_1, A_2, \dots A_n) = \bigwedge_{i=1}^{K} \bigvee_{j=1}^{n} f_{ij}(A_j)$$

$(A_1 \vee A_2 \vee A_3) \wedge (A_1 \vee \neg A_2 \vee 0) \wedge (\neg A_1 \vee A_2 \vee A_3)$

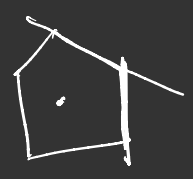


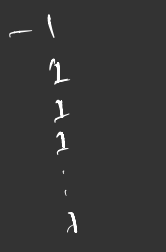
$1 - 1 - 1 + 2 \cdot 5$

$\sum w_i x_i + \left(n_i - \frac{1}{2}\right)$

$K - K + \frac{1}{2}$

5.2



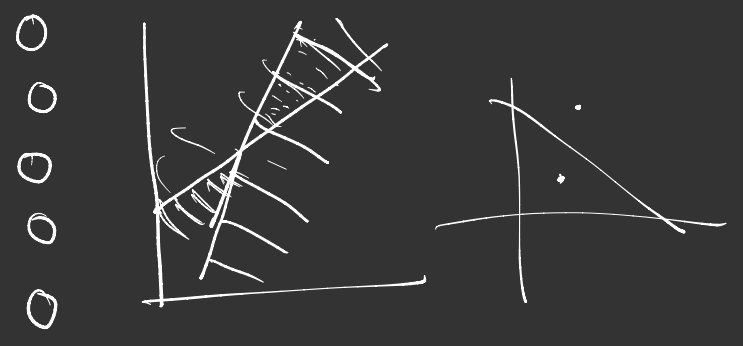


$2x + 3y = 0$

-5

$2 + 3 + 5 \Rightarrow 0$

$z_i = \sum w_i x_i + c$ 4-5



$\left(1+1+1+1+1\right) - \left(n - \frac{1}{2}\right)$

$\sum w_i x_i - \left(n - \frac{1}{2}\right)$

$A \Longleftrightarrow B$

$(\neg A \vee B) \wedge (A \vee \neg B)$

$ax + by + c$

$-ax - by - c$

5.3

$H = \sigma(W_1 x + b_1)$

$Y = W_2 H + b_2$

$H = \frac{1}{1+e^{-(W_1 x + b_1)}}$

$= \frac{e^{W_1 x + b}}{1+e^{W_1 x + b}}$

$= \frac{x e^{W_1 x + b_1}}{(1+e^{W_1 x + b})^2}$

$\therefore \frac{\partial Y}{\partial W_2} = H \qquad \frac{\partial Y}{\partial b_2} = 1$

$\therefore \frac{\partial Y}{\partial W_1} = \frac{\partial Y}{\partial H} \cdot \frac{\partial H}{\partial W_1}$

$= W_2 \cdot \left(\frac{x e^{W_1 x + b_1}}{(1+e^{W_1 x + b_1})^2} \right)$

$\frac{\partial Y}{\partial b} = W_2 \cdot \frac{e^{W_1 x + b_1}}{(1+e^{W_1 x + b_1})^2}$

5.4

$(x_1, x_2) = (5, 3)$

$\hat{y} = 1$

$\eta = 1$

$\therefore L = (y - \hat{y})^2$

$\therefore \hat{y} = \rho(w_4 h_1 + w_5 h_2 + q_3)$

$\therefore h_1 = \rho(w_1 x_1 + w_2 x_2 + q_1)$

$\therefore h_2 = \rho(w_3 x_2 + q_2)$

$h_1 = \rho(10 - 9 + 2)$

$= 3$

$h_2 = \rho(3 - 4) = 0$

$\hat{y} = 3 - 1 = 2$

$\frac{\partial L}{\partial \hat{y}} = -2(y - \hat{y})$

$\frac{\partial L}{\partial w_4}\Big|_{x_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_4}$

$= -2(y - \hat{y}) \cdot h_1 = 2 \cdot 3 = 6$

$\frac{\partial L}{\partial w_5} = -2(y - \hat{y}) \cdot h_2 = 0$

$\frac{\partial L}{\partial q_3} = -2(y - \hat{y}) = 2$

$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial h_1} \cdot \frac{\partial h_1}{\partial w_1}$

$= -2(y - \hat{y}) \cdot w_4 x_1$

$= 2 \cdot 1 \cdot 5 = 10$

$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial h_1} \cdot \frac{\partial h_1}{\partial w_2}$

$= -2(y - \hat{y}) \cdot x_2 \cdot w_4$

$= 2 \cdot 3 \cdot 1 = 6$

$\frac{\partial L}{\partial q_1} = 2$

$\frac{\partial L}{\partial w_3} = 0 \qquad \frac{\partial L}{\partial q_2} = 0$

$w_1 = w_1 - 10 = -8$

$w_2 = -3 - 6 = -9$

$w_3 = 1 \qquad w_4 = -5$

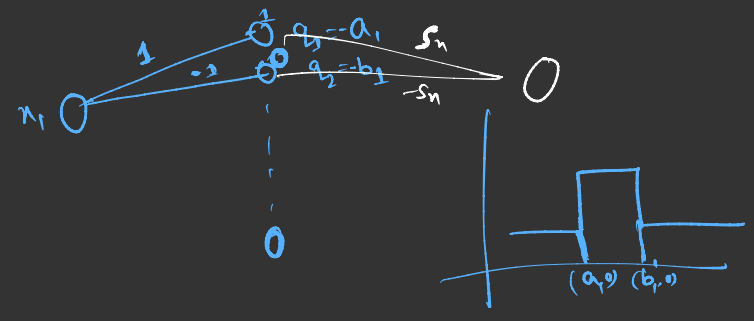
$w_5 = 5$

$q_3 = -3$

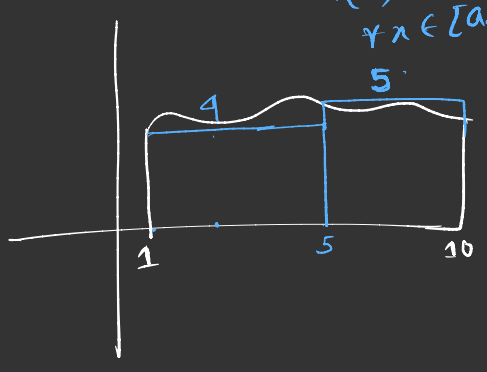
$q_1 = 0$

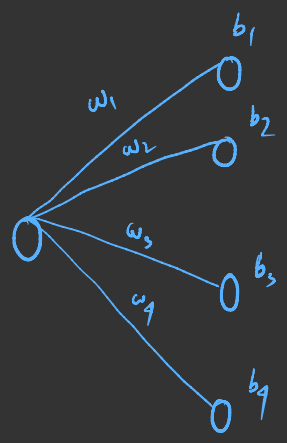
$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \left(\frac{\partial \hat{y}}{\partial h_1} \cdot \frac{\partial h_1}{\partial w_2} + \frac{\partial \hat{y}}{\partial h_2} \cdot \frac{\partial h_2}{\partial w_2} \right)$

5.5



$\delta(x) = \begin{cases} C_1(x) & \forall x \in [a_1, b_1) \\ C_n(x) & \forall x \in [a_n, b_n] \end{cases}$





5.6 dXdXK

Conv1 $(2S+1, 2S+1)$, num filters = n

MaxPool (k, k)

$d_1(n)_w = d - 2S$

$d_1(n)_h : d - 2S$

$d_1(n)_c : n$

$d_2(d_1(n))_w = \frac{d-2S}{k}$

$d_2(d_1(n))_h = \frac{d-2S}{k}$

$d_2(d_1(n)) : n$

Num weights

$n(2S+1)^2 K$

$3 \times 3 \times 32 \times 3$

$\Sigma w_i x_i$

$3 \times 3 - 2$

$\frac{31-2}{2}$

$\left\lceil \frac{29}{2} \right\rceil$

$\frac{Kd^2 \times n(d-2S)^2 - nK(2S+1)^2}{nKd^2(d-2S)^2}$

$= \frac{(d^2 - 2Sd)^2 - (2S+1)^2}{d^2(d-2S)^2}$

$0 \Longleftrightarrow 0$

$W = \begin{pmatrix} 1 \\ 1 \end{pmatrix}(1\ 1) + \begin{pmatrix} 1 \\ -1 \end{pmatrix}(1\ -1) + \begin{pmatrix} -1 \\ -1 \end{pmatrix}(-1\ -1)$

$= \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} + \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} + \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$

$= \begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix}$

$x^{(0)} = \left(-\frac{1}{2}, \frac{2}{3}\right)$

$$x_0^1 = \begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix}\begin{pmatrix} -1/2 \\ 2/3 \end{pmatrix}$$

$$= \begin{pmatrix} -\frac{3}{2} + \frac{2}{3} \\ -\frac{1}{2} + 2 \end{pmatrix} = \alpha \begin{pmatrix} -5/6 \\ 1.5 \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

$$x_1^1 = \alpha\left\{\begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix}\begin{pmatrix} -1 \\ 1 \end{pmatrix}\right\} = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

$$x_2^1 = \alpha\left\{\begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix}\begin{pmatrix} -1 \\ 1 \end{pmatrix}\right\} = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$