Correction: computational learning

Camille Gontier (camille.gontier@unibe.ch)

20/12/22

Exercise 1: Linear classifiers

The point of the first questions is to see how to use the derivative of an error function to determine an unknown parameter. Let us assume an unknown process which, to any input x, associates an output y : $x \to y$. We want to approximate it by a linear process $x \to \hat{y} = wx$, i.e. we want to find the optimal w that minimizes the difference between the actual output y and our estimate $\hat{y}$.

1. Recall the chain rule for calculus.

   Correction. Consider two functions f and g. Assuming that the conditions for derivating f, g, and $f \circ g$ are met, the chain rule yields

   $$(f \circ g)' = (f' \circ g) \cdot g'$$

   or, using a different notation,

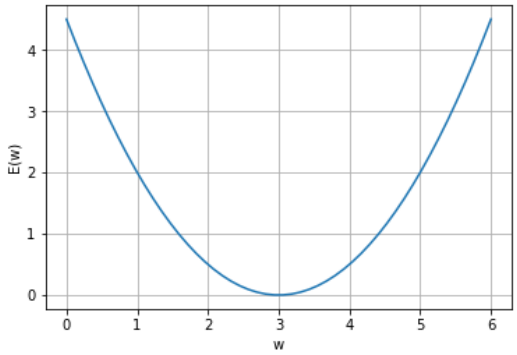
   $$(f(g(x)))' = f'(g(x)) \cdot g'(x)$$

   The chain rule thus allows to compute the derivative of the composition of functions. An intuitive way to understand it is to rewrite the derivatives as fractions (using Leibniz's notation):

   $$(f(g(x)))' = \frac{\Delta f(g(x))}{\Delta x} = \frac{\Delta f(g(x))}{\Delta g(x)} \frac{\Delta g(x)}{\Delta x} = f'(g(x)) \cdot g'(x)$$

2. The unknown process' output for $x = 1$ is $y = 3$. Plot the error function $E(w) = \frac{1}{2}(y - \hat{y})^2$ with respect to w for $y = 3$ and $x = 1$.

   Correction.



3. Prove that the derivative of E with respect to w is $E'(w) = (w - 3)$.

   Correction. $E'(w) = -\frac{1}{2}2(y - wx)x = (w - 3)$

   $w > 3 \implies E'(w) > 0$ and $w < 3 \implies E'(w) < 0$, which shows how $E(w)$ varies with w.

4. Explain why the optimal value of the parameter w^* is obtained for $E'(w^*) = 0$ and that $w^* = 3$.

   Correction.

   Solving for w^* gives $(w^* - 3) = 0 \implies w^* = 3$

5. In the general case, for a linear classifier (such that $x \to \hat{y} = wx$), show that the derivative of the error function $E(w) = \frac{1}{2}(y - \hat{y})^2$ is given by $E'(w) = -x(y - wx)$. Geometrically, what does $E'(w)$ represent?

   Correction. $E'(w) = -\frac{1}{2}2(y - wx)x = -x(y - wx)$

   $E'(w)$ quantifies the variation of E with respect to w, and eventually indicates towards which direction w should be moved to reduce E.

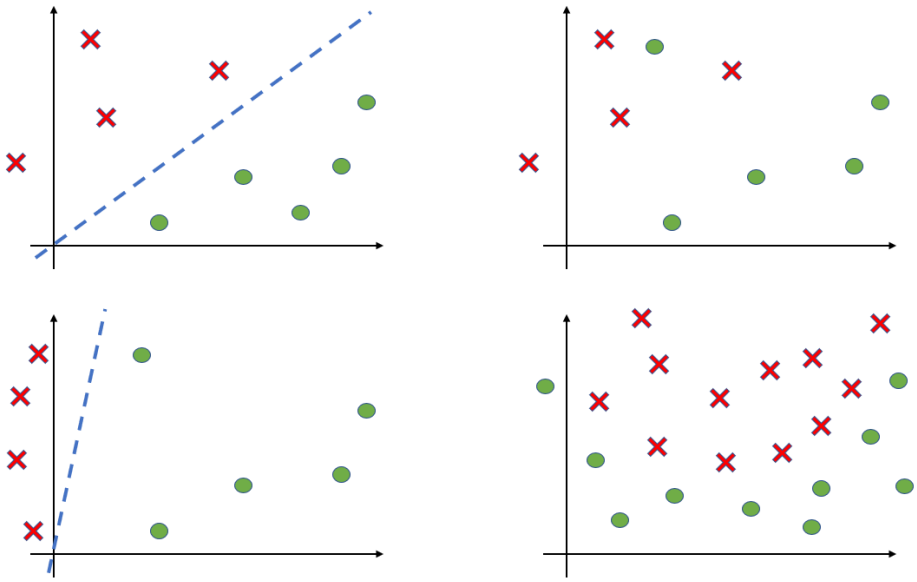
6. Derive a possible learning rule for w.

   Correction. $\Delta w = -\eta E'(w)$

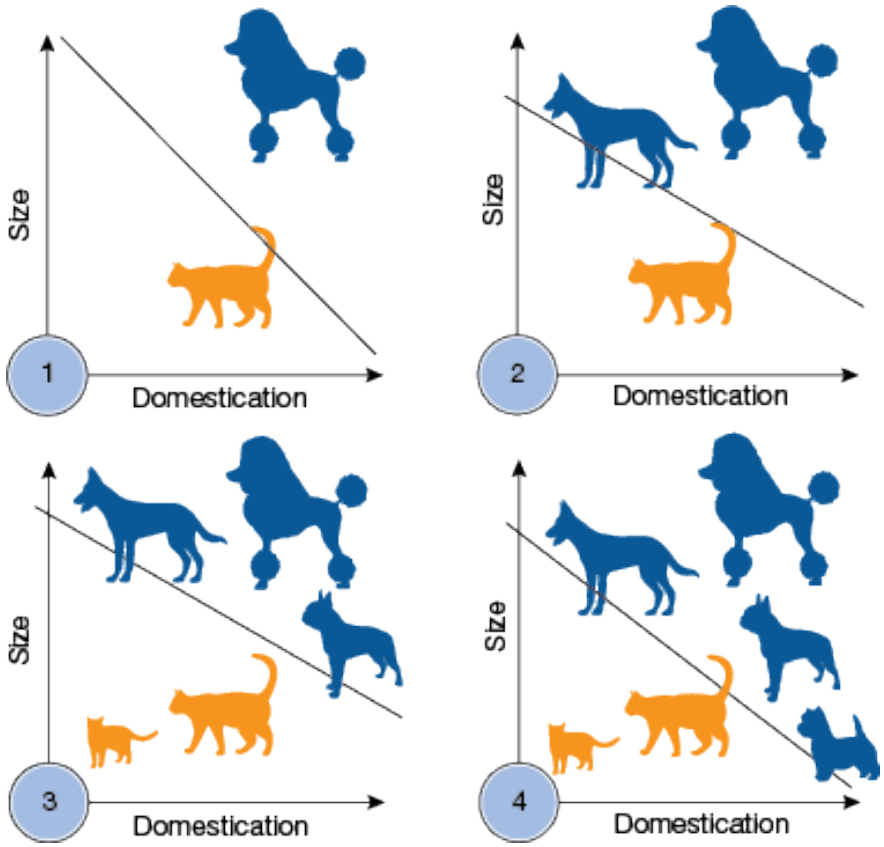
7. **Application to a 2 dimensional case**.

   (a) Which of the following data sets are linearly separable? For those which are, draw a possible linear separation.

   Correction.



In the previous questions, we saw how to set the parameter w of a function to minimize the error function $E(w)$: we are now going to apply it to a practical 2-dimensional example. Imagine we want to develop a linear classifier which determines whether an animal belong to the class "*dog*" or to the class "*cat*" depending on two features: its size and domestication level. As you can see on the picture below, the separation line (i.e. the vector $\mathbf{w}$) needs to be updated every time a new samples is obtained, to make sure that samples from each class are separated.



"A perceptron can learn to separate dogs and cats given size and domestication data. As more training examples are added, the perceptron updates its linear boundary." Source: `https://www.aiche.org/resources/publications/cep/2018/june/introduction-deep-learning-part-1`

Instead of working with dogs and cats, we will work with a simpler example: each sample $\mathbf{x}$ has two features (x_1 and x_2) and can belong either to class $y = 1$ or to class $y = -1$.

(b) Study the 2-dimensional classification task below:

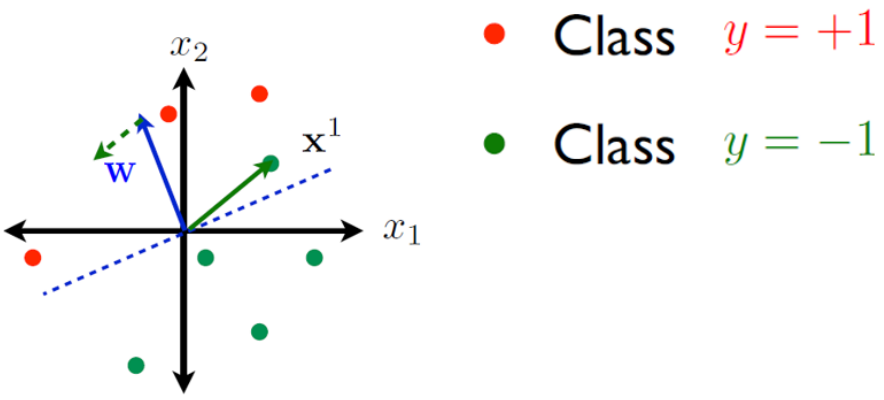
i. This classification task is performed by a 2-dimensional linear classifier with $\hat{y} = f(\mathbf{w} \cdot \mathbf{x})$. What is the function f ?
Correction. f is the sign function:

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

ii. Assume $\mathbf{x}^1 = (1, 1)$ and $\mathbf{w} = (-\frac{1}{2}, \frac{\sqrt{3}}{2})$. In this situation, the separation line goes through quadrants I and III (see figure below). Compute $\mathbf{w} \cdot \mathbf{x}^1$ and explain why $\mathbf{w}$ does not achieve perfect classification.
Correction. $\mathbf{w} \cdot \mathbf{x}^1 = -\frac{1}{2} \times 1 + \frac{\sqrt{3}}{2} \times 1 > 0$, while $y_{x^1} = -1$, $\mathbf{x}^1$ is thus misclassified.

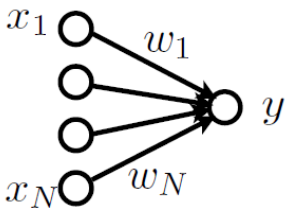
iii. Apply the correction $\Delta\mathbf{w} = \eta(y - \hat{y})\mathbf{x}^1$ with $\eta = 0.05$. Does the separation line turn clockwise or counter-clockwise ? Is $\mathbf{x}^1$ still misclassified?
Correction. $\Delta w_1 = 0.05 \times (-2) \times 1$ and $\Delta w_2 = 0.05 \times (-2) \times 1$, the updated value for $\mathbf{w}$ is thus $\mathbf{w} = [-0.6, 0.77]$. As expected, the separation line turns counter-clockwise. $\mathbf{w} \cdot \mathbf{x}^1 > 0$, so $\mathbf{x}^1$ is still misclassified.

iv. Repeat until finding a possible solution for $\mathbf{w}$.
Correction. Upon repeating the same correction, $\mathbf{w}$ becomes $[-0.7, 0.66]$. $\mathbf{w} \cdot \mathbf{x}^1 < 0$, so $\mathbf{x}^1$ is correctly classified. All points are correctly classified, so learning stops.



Exercise 2: Perceptron and Deep Learning

1. **Perceptron.** In the previous question, our linear classifier worked with only $N = 2$ features. In practice, classification problems can involve any number N of features. The data set classification task from the previous question can be formalized as a Neural Network Classifier with N inputs $(x_1, \dots, x_N)$, weights $(w_1, \dots, w_N)$, and an output $\hat{y} = f(\sum_{i=1}^{N} w_i x_i)$:



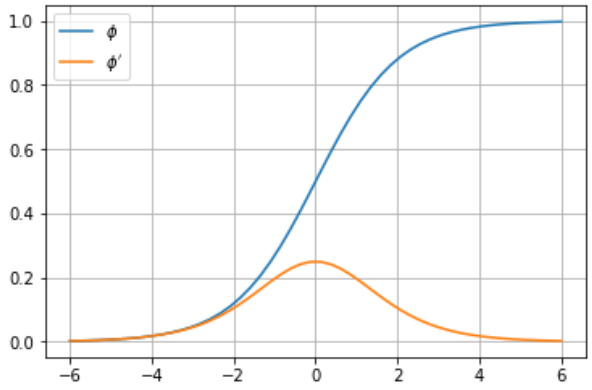
   (a) Prove that the behaviour of the Classifier is unchanged if all the weights $(w_1, \dots, w_N)$ are multiplied by the same value $\alpha > 0$. Geometrically, what does this property mean in a 2-dimensional case?

   Correction. For any $\alpha > 0$, $\sum_{i=1}^{N} w_i x_i > 0 \implies \alpha \sum_{i=1}^{N} w_i x_i > 0$ and $\sum_{i=1}^{N} w_i x_i < 0 \implies \alpha \sum_{i=1}^{N} w_i x_i < 0$, so the output of the perceptron (assuming the activation function is the Heaviside step function) remains unchanged.

   Geometrically, this means that whatever the length of the vector $\mathbf{w}$, the orientation of the separation plan will be the same.

   (b) A possible choice for f is the sigmoid function $\phi(u) = \frac{1}{1+\exp(-u)}$. Compute the derivative $\phi'(u)$ and plot it.

   Correction. $\phi'(u) = \frac{e^u}{(1+e^u)^2} = \phi(u)(1 - \phi(u))$



(c) With the help of exercise 1, use the mean-squared error to derive the learning rule for $\mathbf{w}$ if $\hat{y} = \phi(\sum_{i=1}^{N} w_i x_i)$.
Correction.

$$\Delta w_i = \eta(y - \phi(\mathbf{w} \cdot \mathbf{x}))\phi'(\mathbf{w} \cdot \mathbf{x})x_i$$

However, note that, depending on the choice of the transfer function ϕ, its derivative ϕ' may not exist or be usable. For instance, the perceptron was originally designed as a single layer neural network with a binary activation function $\phi(x) = \text{sign(x)}$, i.e.:

$$\begin{cases} x \geq 0 \implies \phi(x) = 1 \\ x < 0 \implies \phi(x) = -1 \end{cases}$$

In that case, $\phi'(x)$ is equal to 0 everywhere and undefined for $x = 0$, and is thus unusable for learning. We can however postulate the perceptron learning rule $\Delta w_i = \eta(y - \phi(\mathbf{w} \cdot \mathbf{x}))x_i$, for which the perceptron convergence theorem also holds.

(d) Give the conditions for which learning stops.
Correction. $y = \phi(\mathbf{w} \cdot \mathbf{x})$ (which can be intuitively understood as the condition in which data points are correctly classified; in that case, learning stops) or $\phi'(\mathbf{w} \cdot \mathbf{x}) \approx 0$

2. **Deep Learning.** For more complicated tasks (e.g. for non-linearly separable datasets), more complicated classifier involving hidden layers need to be used. In that case, computing the weights update by hands would become tedious: we will thus firstly let the computer do it for us. Go to `http://playground.tensorflow.org/` and answer the following questions:

(a) Can a classification task be performed on the *Gaussian* dataset using only 2 features and no hidden units? Justify.
Correction. As the dataset is linearly separable, only two features inputs (x_1 and x_2) are sufficient to separate the datasets.

(b) Is classification still possible for the *Circle* dataset? Why?
Correction. This dataset is not linearly separable, so x_1 and x_2 (that is to say, a simple perceptron) are not sufficient to separate it. Notice that this answer still holds even when using a non-linear transfer function (e.g. sigmoid or tanh).

(c) Describe the behaviour of the network for the *Circle* dataset with 2 features and one hidden layer made of 3 units.
Correction. In the hidden layer, the first unit checks whether the data point is on the left of the plane, the second checks whether the data point is on the bottom left of the plane, and the third whether the data point is on the top left of the plane.

(d) Design a network able to classify the *Spiral* dataset.
Correction. Use a second hidden layer.

3. **Neural networks and backpropagation.** Similarly to the simplified linear classifiers studied previously, a neural network can be optimized (i.e. the weights can be learned) by using the derivative of the error function with respect to the weights. But having hidden layers make the computation of the derivative slightly trickier: TensorFlow used an algorithm called backpropagation to efficiently update the weights. Here, we will derive the learning rule for a very simplified network having only one hidden layer with one neuron (although the principle remains the same for more complicated Deep Neural Networks).

We study here a neural network consisting of a first layer of input neurons of outputs $\mathbf{x}$, one hidden layer of outputs $\mathbf{h}$ (i.e $N = 1$), and an output layer of outputs $\hat{\mathbf{y}}$. W denotes the weights between the neurons of the input and the hidden layer, and V denotes the weights between the neurons of the hidden and the output layer. The transfers functions are defined as follows:

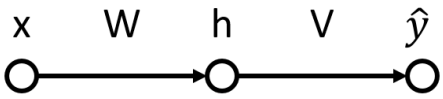
$$\mathbf{h} = f(W\mathbf{x})$$

and

$$\hat{\mathbf{y}} = g(V\mathbf{h})$$

We here only consider deterministic (i.e. non-probabilistic) networks.

(a) Recall the universal approximation theorem. Intuitively, how would you expect the performances of the network to vary when the number of hidden layers N increases ?
Correction. The universal approximation theorem states that standard multilayer feedforward networks with as few as one hidden layer (N=1) using arbitrary squashing functions are capable of approximating any Borel measurable function from one finite dimensional space to another to any desired degree of accuracy (and are thus universal approximators), provided sufficiently many hidden units are available.
However, when the number of neurons is limited, the performance increases when neurons are distributed among many hidden layers (i.e. $N > 1$).
For the next questions, we consider a toy example of a very simplified 1-dimensional NN (see figure below).



(b) Show that the learning rule for weights V is given by $\Delta V = \eta \Delta \mathbf{y} h$, where

$$\Delta \mathbf{y} = (\mathbf{y} - g(V\mathbf{h})) \cdot g'(V\mathbf{h})$$

Correction. The trick is to use the chain rule twice to compute the derivative of the mean-squared error $E = \frac{1}{2}(y - \hat{y})^2$ with respect to V:

$$\frac{\partial E}{\partial V} = \frac{\partial E}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial Vh} \cdot \frac{\partial Vh}{\partial V}$$

with

$$\hat{y} = g(Vh)$$

$$\frac{\partial Vh}{\partial V} = h$$

$$\frac{\partial \hat{y}}{\partial Vh} = \frac{\partial g(Vh)}{\partial Vh} = g'(Vh)$$

$$\frac{\partial E}{\partial \hat{y}} = \hat{y} - y$$

(c) Show that the learning rule for weights W is given by $\Delta W = \eta((V\Delta \mathbf{y}) \cdot f'(W\mathbf{x}))\mathbf{x}$
Correction.

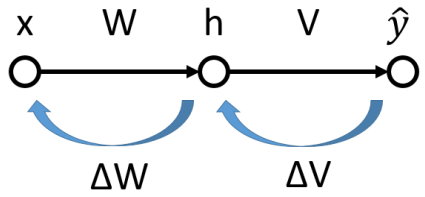
$$\frac{\partial E}{\partial W} = \frac{\partial E}{\partial f(Wx)} \cdot \frac{\partial f(Wx)}{\partial Wx} \cdot \frac{\partial Wx}{\partial W}$$

The demonstration is similar to the previous question, except for the derivative of the error with respect to $f(Wx) = h$. Here, in this simplified case, we can apply the chain rule again to compute it:

$$\frac{\partial E}{\partial h} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial Vh} V$$

(d) How does backpropagation learning differ from the perceptron learning rule when $N = 0$?
Correction. These learning rules are equivalent when $N = 0$ (no hidden layer).

(e) Explain why this learning algorithm is called backpropagation. Is it a local learning rule ?
Correction. We obtain a recursive expression (hence the appellation *backpropagation*) for the computation of the derivative of the error. It is a non-local learning rule, as information from the last layer is required (backpropagated) to update the weight of the previous layer.



Backpropagation is of particular importance in machine learning and artificial intelligence, as it is one of the most widely used learning algorithms for training neural networks. A nice intuitive graphical explanation of how it works for more complicated and realistic networks (with more than 1 hidden layer and several neurons per layer) is provided in this video: `https://www.youtube.com/watch?v=Ilg3gGewQ5U` (*What is backpropagation really doing? — Deep learning, chapter 3*).