Exercise: computational learning

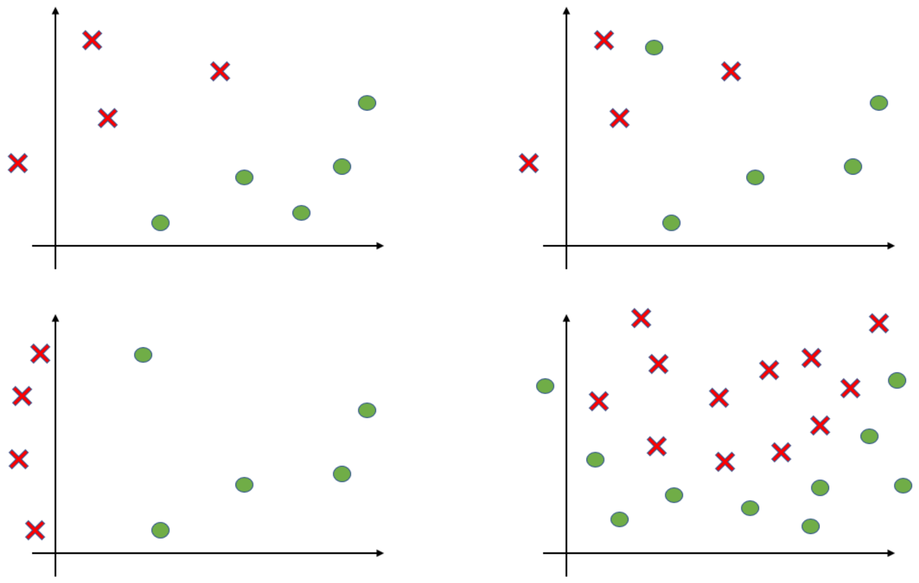
Camille Gontier (camille.gontier@unibe.ch)

20/12/22

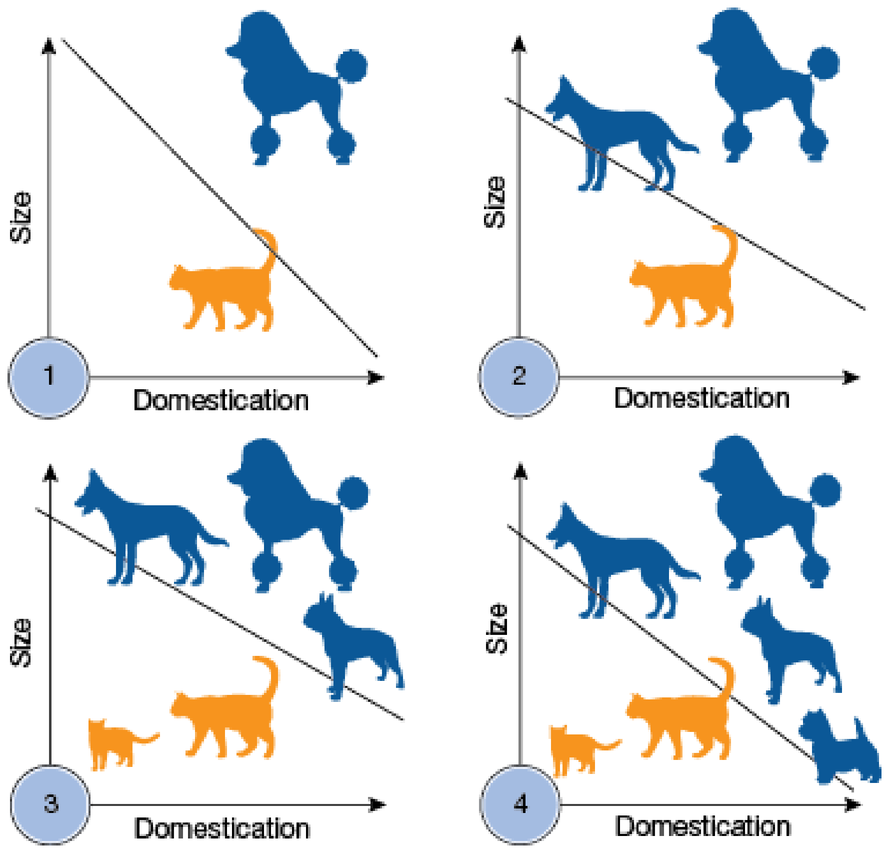
Exercise 1: Linear classifiers

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.
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 $x = 1$.
3. Prove that the derivative of E with respect to w is $E'(w) = (w - 3)$.
4. Explain why the optimal value of the parameter w^* is obtained for $E'(w^*) = 0$ and that $w^* = 3$.
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?
6. Derive a possible learning rule for 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.



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 $y = -1$.

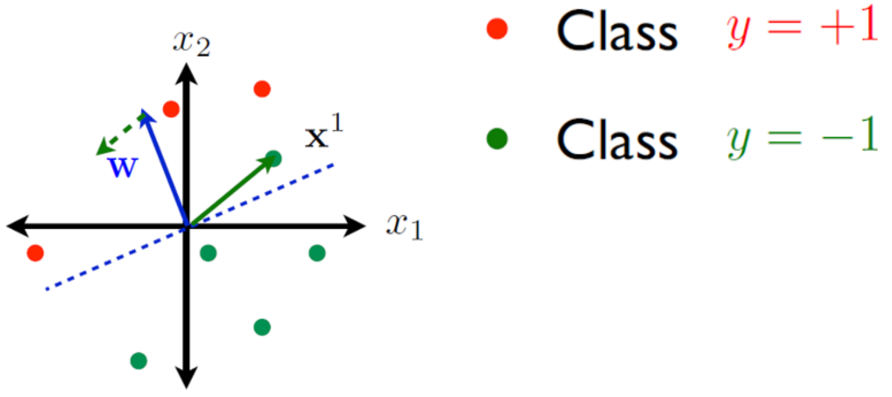
(b) Study the 2-dimensional classification task below:

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 ?

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.

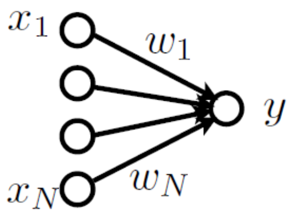
iii. Apply the correction $\Delta\mathbf{w} = \eta(y - \hat{y})\mathbf{x}^1$ with $\eta = 0.05$. Does the separation line turn clockwise or counter-clockwise ? Is $\mathbf{x}^1$ still misclassified?

iv. Repeat until finding a possible solution for $\mathbf{w}$.



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?

   (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.

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

   (d) Give the conditions for which learning stops.

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.

(b) Is classification still possible for the *Circle* dataset? Why?

(c) Describe the behaviour of the network for the *Circle* dataset with 2 features and one hidden layer made of 3 units.

(d) Design a network able to classify the *Spiral* dataset.

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:

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

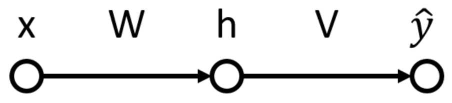
and

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

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 ?

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

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

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

(d) How does backpropagation learning differ from the perceptron learning rule when $N = 0$?

(e) Explain why this learning algorithm is called backpropagation. Is it a local learning rule ?