2.2.2 Neural Networks (for Classification)

Of course, neural networks can also be used for classification. In fact, the Perceptron $\hat{f}(x) = \mathrm{sign}(\langle w, x\rangle + b)$ was already a simple neural network
We again consider neural networks of the form

$$\hat{f}(x) = \Phi_L \circ \ldots \Phi_1(x)$$

($\circ$ means function composition: $(f \circ g)(x) = f(g(x))$, which means applying g first, then applying f to the result)

- In the context of neural networks
  - Each $\Phi_l(x)$ represents a layer transformation
  - The function $\hat{f}(x)$ is a composition of these transformation, applied sequentially!
  - So, the output of layer 1 becomes the input to layer 2 and so on!

  with $\Phi_l(x) := \sigma_l(W_l^T x + b_l)$. For classification of $C \geq 2$ classes (**2 or more classes!**, data of the form $(x_i, y_i) \in \mathbb{R}^d \times \mathbb{R}^C$ is usually used, where we assume that the y_i are in so-called one-hot encoding, i.e., they are vectors of the form $(0, \ldots, 0, 1, 0, \ldots, 0) \in \mathbb{R}^C$ with a 1 in the c-th position, where $c \in \{1, \ldots, C\}$ denotes the class. Choosing an appropriate loss function is very important for classification.
  One possible choice in $\ell(z, y) := \frac{1}{2}||z - y||^2$ for $y, z \in \mathbb{R}^C$. Much more common is the so-called cross-entropy. We assume the last activation function σ_L is the identity and that $n_L = C$. We then define the so-called Softmax operation

$$\mathrm{softmax} : \mathbb{R}^C \to \mathbb{R}^C, \quad \mathrm{softmax}(x)_c := \frac{\exp(x_c)}{\sum_{i=1}^C \exp(x_i)}$$

  Note that for all $x \in \mathbb{R}^C$, the vector $\mathrm{softmax}(x) \in \mathbb{R}^C$ is a discrete probability distribution, since it holds that

$$0 \leq \mathrm{softmax}(x)_c \leq 1 \forall c = 1, \ldots, C, \quad \sum_{c=1}^{C} \mathrm{softmax}(x)_c = 1$$

  Furthermore, we define the cross-entropy $H : \mathbb{R}^C \times \mathbb{R}^C \to \mathbb{R}$ as

$$H(x, y) := \begin{cases} -\sum_{c=1}^C y_c \mathrm{log}(x_c) & \text{if } x_c > 0 \ \forall c = 1, \ldots, C, \\ \infty & \text{otherwise} \end{cases}$$

  x_c is the predicted probability for class c (from softmax!), y_c is 1 for the correct class and 0 otherwise.
  If y is a one-hot encoding of the c-th class, then $H(x, y)$ is small if x_c is close to or equal to 1. (remember: $\log(1) = 0$ and if y_c has 1 in the correct class position, CE is computed solely

by $-\log(x_c)$. Low loss --> good prediction)
E.g. if the correct class has high probability x_c then $\log(x_c)$ is close to 0, meaning low loss. If the correct class has low probability, $\log(x_c)$ is large and negative, meaning high loss!
We obtain a suitable loss function by computing the Softmax operation and the cross-entropy

$$\ell(z, y) := H(\mathrm{softmax}(z), y) = -\sum_{c=1}^{C} y_c \log\left(\frac{\exp(z_c)}{\sum_{i=1}^{C} \exp(z_i)}\right)$$

To train a neural network with this loss function, we need the derivative with respect to z. This has a particularly simple form.

Proposition 2.2.2

Let $x, y \in \mathbb{R}^C$ and also $y_c \geq 0$ for all $c \in \{1, \ldots, C\}$ and $\sum_{c=1}^{C} y_c = 1$. Then for all $c \in \{1, \ldots, C\}$, it holds that

$$\frac{\partial \ell(z, y)}{\partial z_c} = \mathrm{softmax}(z)_c - y_c$$

The gradient is the difference between predicted probability and actual label! This makes gradient descent simple: If the prediction is too high, the weight decreases, and if it's too low, the weight increases!
Proof. The proof is carried out in the exercise