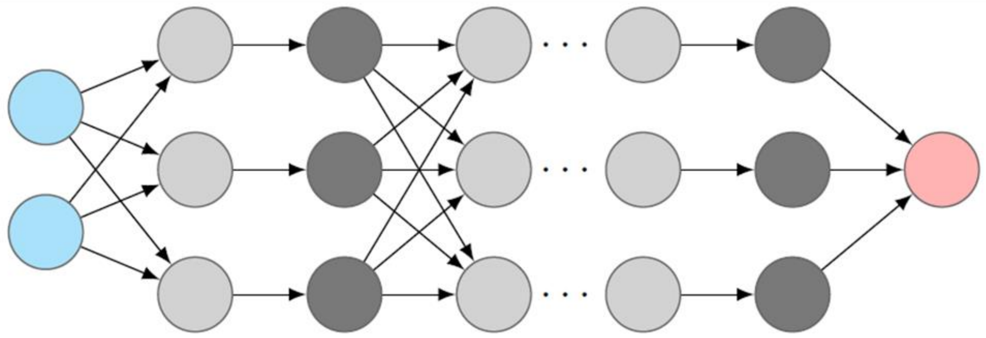
A linear regression “network”

Lecturer: Michael Möller – michael.moeller@uni-siegen.de
Exercises: Hartmut Bauermeister – hartmut.bauermeister@uni-siegen.de



Reminder

What (supervised) "Deep Learning" is:
A fancy word for function approximation

Assume there is an unknown function G that maps some kind of input data x to some kind of desired output y.

Assume we are given some evaluations of this (unknown) function G. This is what we will call ***training data***!

1. Choose a parameterized function $\mathcal{N}(x;\theta)$ in the hope that for the right choice of parameters θ it approximates the unknown function G well. We call $\mathcal{N}$ the ***network***, and sometimes refer to θ as the ***weights***.
2. Try to determine suitable weights θ in such a way that $\mathcal{N}(x_i;\theta) \approx y_i$ holds for all examples (x_i, y_i) from your training data set. This is referred to as ***training the network***.
3. Make try to ensure that both, the architecture as well as the training are chosen in such as way that the network makes good predictions during inference, i.e. on previously unseen data x: $\mathcal{N}(x;\theta) \approx G(x)$. We refer to this property as ***generalization***.

Wine rating regression

https://www.kaggle.com/kashnitsky/mlcourse#winequality-white.csv

We assume that the wine quality and taste can be characterized chemically, based on the following 11 descriptors:

1. fixed acidity
2. volatile acidity
3. citric acid
4. residual sugar
5. chlorides
6. free sulfur dioxide
7. total sulfur dioxide
8. density
9. pH
10. sulphates
11. alcohol

$x \in \mathbb{R}^{11}$

$$\xrightarrow{\quad G \quad}$$

Wine quality
(number between 0 and 10)

$y \in \mathbb{R}$

GOAL: Choose and train a network, i.e. a parameterized function $\mathcal{N} : \mathbb{R}^{11} \times \mathbb{R}^n \to \mathbb{R}$ to approximate G .

Linear regression

GOAL: Choose and train a network, i.e. a parameterized function $\mathcal{N} : \mathbb{R}^{11} \times \mathbb{R}^n \to \mathbb{R}$ to approximate G .

Simples choice for the architecture: Affine linear function in the input x

$$\mathcal{N} : \mathbb{R}^{11} \times \mathbb{R}^{12} \to \mathbb{R}$$

$$\mathcal{N}(x;\theta) = \left\langle \theta, \begin{pmatrix} x \\ 1 \end{pmatrix} \right\rangle := \sum_{i=1}^{11} \theta_i x_i \; + \; \theta_{12}$$

We need *training data*, i.e., a large number M of pairs (x, y = G(x)), that is, pairs of chemical descriptors and expert ratings. In the exercise example we'll have M = 4000.

Next, we need a way to *train* the parameters θ of our network. The common way to do this is to ensure that

$$\mathcal{N}(x_j;\theta) \approx y_j \qquad \forall \text{ training examples } (x_i, y_i)$$

Linear regression

How do we ensure/encourage $\mathcal{N}(x_j;\theta) \approx y_j \qquad \forall \text{ training examples } (x_i, y_i)$??

One commonly chooses a *loss function* $\mathcal{L}$ to measure how well $\mathcal{N}(x_j;\theta) \approx y_j$ is met, e.g. the quadratic loss

$$\mathcal{L}(\mathcal{N}(x_j,\theta), y_j) = \|\mathcal{N}(x_j,\theta) - y_j\|^2$$

The overall quality of the current parameters θ is then measured by summing the loss function over all training examples:

$$E(\theta) = \sum_j \|\mathcal{N}(x_j,\theta) - y_j\|^2$$

Finally, one tries to determine the optimal parameters θ as the *argument that minimizes the training costs*:

$$\hat{\theta} = \arg\min_\theta E(\theta)$$

During *inference*, one uses $\mathcal{N}(x;\hat{\theta})$ to make predictions.

Optimality conditions

How do we solve $\hat{\theta} = \arg\min_{\theta} E(\theta)$?

For a function $E : \mathbb{R}^n \to \mathbb{R}$ we define the *partial derivative* $\frac{\partial E}{\partial \theta_j}$ as the usual derivative of a function from $\mathbb{R}$ to $\mathbb{R}$ (which I assume you know) by treating all variables $\theta_i, \; i \neq j$, as constants.

Example on the board: $E(\theta_1, \theta_2) = \theta_1^2 \theta_2 + \theta_2 + \sin(\theta_2)\theta_1$

Optimality conditions

If all partial derivatives of a function $E : \mathbb{R}^n \to \mathbb{R}$ exist and are continuous, we call E *continuously differentiable*, and call

$$\nabla E := \begin{pmatrix} \frac{\partial E}{\partial \theta_1} \\ \vdots \\ \frac{\partial E}{\partial \theta_n} \end{pmatrix} : \mathbb{R}^n \to \mathbb{R}^n$$

the *gradient* of E. Evaluating the gradient at a point θ yields a vector in $\mathbb{R}^n$.

Necessary condition for local minima: If $E : \mathbb{R}^n \to \mathbb{R}$ has a local minimum at some point θ, then it holds that

$$\nabla E(\theta) = 0$$

Optimality conditions

But how do we ensure that a point actually is a minimizer?

As we will see, there are different answers…

1. Most of the time (for complex deep learning): We do not ensure it!
2. In particular cases, e.g. in our linear wine regregression example, the costs are *convex*! In this case $\nabla E(\theta) = 0$ ensures that θ is a global minimizer!

Network architecture:

$$\mathcal{N}(x;\theta) = \left\langle \theta, \begin{pmatrix} x \\ 1 \end{pmatrix} \right\rangle := \sum_{i=1}^{11} \theta_i x_i \; + \; \theta_{12} = \begin{pmatrix} x \\ 1 \end{pmatrix}^T \theta$$

Final cost function for training examples (x_j, y_j):

$$E(\theta) = \sum_j \|\mathcal{N}(x_j,\theta) - y_j\|^2 = \sum_j \left\| \begin{pmatrix} x_j \\ 1 \end{pmatrix}^T \theta - y_j \right\|^2$$

Computing gradients

In our linear wine regression example, we need to compute the gradient of

$$E(\theta) = \sum_j \left\| \begin{pmatrix} x_j \\ 1 \end{pmatrix}^T \theta - y_j \right\|^2$$

and solve $\nabla E(\theta) = 0$ for θ.

It might be handy to learn some general rules for computing gradients

Sum rule: The gradient of the sum of two (continuously differentiable) terms is the sum of their gradients.

And what about nested function?

Chain rule

<u>Chain rule:</u> Let $E = f \circ g$, i.e. $E(x) = f(g(x))$, for $f : \mathbb{R}^n \to \mathbb{R}$ and $g : \mathbb{R}^m \to \mathbb{R}^n$. Let all partial derivatives of f and g_i, $i \in \{1, \dots, n\}$ exist and be continuous. Then it holds that

$$\nabla E(x) = (Jg(x))^T \cdot \nabla f(g(x))$$

where

$$Jg(x) = \begin{pmatrix} \frac{\partial g_1}{\partial x_1}(x) & . & . & \frac{\partial g_1}{\partial x_m}(x) \\ . & . & . & . \\ . & . & . & . \\ \frac{\partial g_n}{\partial x_1}(x) & . & . & \frac{\partial g_n}{\partial x_m}(x) \end{pmatrix} \in \mathbb{R}^{n \times m}$$

is the *Jacobi-matrix* of g. For convenience we define the gradient of a vector valued function as the transposed Jacobi-matrix, $\nabla g(x) := (Jg(x))^T$. The chain rule then becomes

$$\nabla E(x) = \nabla g(x) \cdot \nabla f(g(x))$$

Computing gradients

Let us return to the gradient of our linear wine regression example

$$E(\theta) = \sum_j \left\| \begin{pmatrix} x_j \\ 1 \end{pmatrix}^T \theta - y_j \right\|^2$$

for which we wanted to solve $\nabla E(\theta) = 0$ for θ.

Board: The gradient of E is given by

$$\nabla E(\theta) = 2 \sum_j \begin{pmatrix} x_j \\ 1 \end{pmatrix} \left(\begin{pmatrix} x_j \\ 1 \end{pmatrix}^T \theta - y_j \right) = 0$$

$$\Rightarrow \theta_{opt} = \left(\sum_j \begin{pmatrix} x_j \\ 1 \end{pmatrix} \begin{pmatrix} x_j \\ 1 \end{pmatrix}^T \right)^{-1} \left(\sum_j y_i \begin{pmatrix} x_j \\ 1 \end{pmatrix} \right)$$

No worries – we will detail the chain rule several times throughout the course!

Summary: Linear wine regressor

Network architecture:

$$\mathcal{N}(x;\theta) = \begin{pmatrix} x \\ 1 \end{pmatrix}^T \theta$$

How we proceed:

* **First exercise**: Set up your system and familiarize yourself with Python and NumPy.

* **First homework**: Implement a linear wine regression! Details will be provided on the courses website!

* **Up next**: Beyond linear regression – how to go deeper! *Fully connected* neural networks.

Inference: Predict the scores of a new wine sample by simply computing $\mathcal{N}(x;\theta_{opt})$