Deep Learning: Applications in Biology

Alexander Schönhuth

UNIVERSITÄT BIELEFELD
Faculty of Technology

Bielefeld University
October 12, 2022

Contents Today

- Organizational Matters
- Introduction
  - None of today's topics plays an explicit role in assignments/exercises or the exam
  - But they may reappear in other topics, and then play an implicit role
  - Goal today is to get fundamental ideas about the following crucial topics

Organizational Matters

Prerequisites, Lectures, Exercises

- Course prerequisites: None; knowledge about optimization and machine learning helpful
- Lectures: Wednesdays, 12-14. In general hybrid meetings; full online meetings possible.
- Exercises: 5 assignments + 1 exam preparation session

Assignments, Exam

- *Tutorials/Assignments:*
  - New exercise sheets provided on Wednesdays October 19, November 2, November 16, November 30, December 14, January 4
  - Exercises to be submitted by Monday, **23:59** eleven days thereafter, discussion on Wednesday, 10-12
  - Submission of exercises in groups of 2 people possible
  - Every one is supposed to present at least one exercise in the tutorials (ideal scenario)
  - Upload to corresponding folder in the "Lernraum Plus"
  - First exercise sheet uploaded on 19th of October (next week)
- *Exam:*
  - Presence exam planned for **Wednesday, February 1, 2022 between 10:00 and 14:00** (may be subject to changes due to situation; we will communicate changes as timely as possible)
  - Admitted: everyone exceeding 50% of total exercise points
  - *Preparatory Session:* Wednesday, January 25, 10-12

Tutorials

- Every **Wednesday, 10-12, U10-146**
- Tutor: Johannes Schlüter
- Tutorials in English; in German if applicable
- Hybrid meetings (links will be provided in time); presence desirable and encouraged
- Presentation of individual solutions during the online meeting, individually

COURSE MATERIAL

- Course website:

  https://gds.techfak.uni-bielefeld.de/teaching/2022winter/bioadl

  - Slides and pointers to literature
  - Excercise sheets
- Lernraum Plus:

  https://lernraumplus.uni-bielefeld.de/course/view.php?id=15150

  - Submission of exercise solutions
  - Self-managed forum

Literature and Links

- Michael Nielsen. *Neural Networks and Deep Learning*: `http://neuralnetworksanddeeplearning.com`
- Ian Goodfellow, Yoshua Bengio, Aaron Courville. *Deep Learning*: `https://www.deeplearningbook.org/`
- Aston Zhang, Zack C. Lipton, Mu Li, Alex J. Smola. *Dive into Deep Learning*: `http://d2l.ai/`
- *Further Links:* To be provided during course.

Tentative Course Curriculum

Part 1: Foundations

- Introduction
  - Supervised Learning
  - Neural Networks (NNs)
- Deep Learning
  - Motivation
  - Training / Challenges
- Training
  - Back Propagation
  - Regularization, Dropout
- Convolutional NNs I
  - Filters, Pooling
  - Hyperparameters

Part 2: Foundations II + Applications

- Convolutional NNs II
  - Training Revisited
  - Vanishing Gradients
- Deep NN Architectures I
  - Recurrent Neural Networks
  - CNNs: Going Deep
- Graph Neural Networks
- Transformers
- Capsule Networks

Introduction

Supervised Learning

Supervised Learning

- There is a functional relationship

$$f^* : \mathbb{R}^d \to V$$

  we would like to understand, or *learn*.
  - *Regression*: $V = \mathbb{R}$
  - *Classification*: $V = \{1, ..., k\}$
- To learn it, we are given *m data points*

$$(x_i, f^*(x_i) = y_i)_{i=1,...,m}$$

  that reflect this functional relationship.

Final goal: Predict $f^*(x)$ on unknown data points x.

Supervised Learning

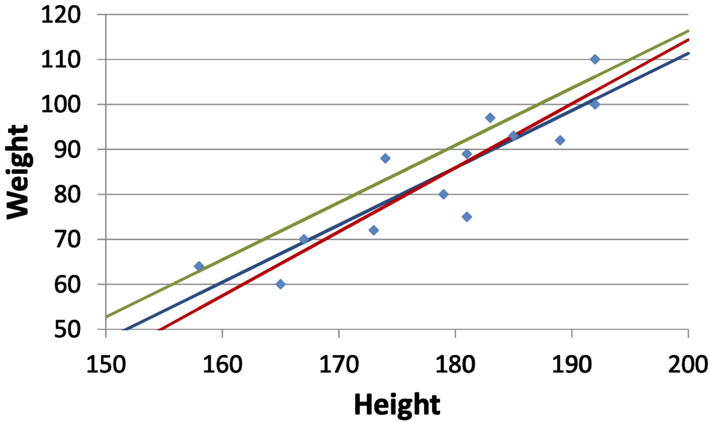
- The idea is to set up a *training procedure* (an algorithm) that *learns* f^* from the training data.
- Learning f^* means to *approximate* it by $f : \mathbb{R}^d \to V$ sufficiently well, where $f \in \mathcal{M}$ for a certain class of functions $\mathcal{M}$.
- In most cases, $f \in \mathcal{M}$ are parameterized by parameters $\mathbf{w}$.
- ☞ Learning means to determine an appropriate choice of parameters $\mathbf{w}$.

Supervised Learning

- We need to determine a *cost (or loss) function* C where $C(f, f^*)$ measures how well $f \in \mathcal{M}$ approximates f^*.
- *Optimization*: Pick $f \in \mathcal{M}$ (by picking the right set of parameters) that yields small (possibly minimal) cost $C(f, f^*)$
- *Generalization*: Optimization procedure should address that f is to approximate f^* well on *unknown data points*.

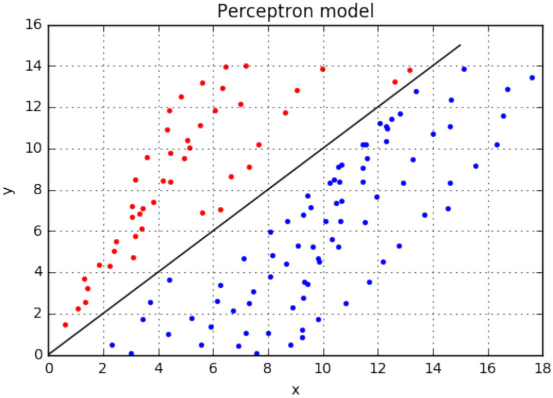
Linear Regression

Example: $f : \mathbb{R} \to \mathbb{R}$



Classification: Perceptron

Example: $f : \mathbb{R}^2 \to \{0, 1\}$



$$
\begin{aligned}
f \quad \mathbb{R}^2 \quad &\longrightarrow \quad \{0 = \text{blue}, 1 = \text{red}\} \\
(x_1, x_2) \quad &\mapsto \quad \begin{cases} 1 & x_2 - x_1 > 0 \\ 0 & x_2 - x_1 \leq 0 \end{cases}
\end{aligned} \tag{1}
$$

We need to specify:

- How to set up the data being used for training
- A model class $\mathcal{M}$, for example linear functions
- A cost function $C(f, f^*)$ that evaluates the goodness of $f \in \mathcal{M}$
- An optimization procedure that picks f such that $C(f, f^*)$ is minimal, or very small
- Keep in mind that f is to perform well on previously unseen data

Supervised Learning
Notation

- The dataset is given by a *design matrix* $\mathbf{X} \in \mathbb{R}^{m \times d}$ where m is the number of data points and d is the number of *features*
- *Example:* A row $\mathbf{X}_i$ corresponds to a data point

$$(x_{i1}, ..., x_{1d})$$

  where $x_{ij}, j = 1, ..., d$ are the features of $\mathbf{X}_i$
- Each data point $\mathbf{X}_i$ is assigned to a *label* y_i that reflects the true functional relationship $y_i = f^*(x_i)$
- $\mathbf{y} = (y_1, ..., y_m) \in V^m$ is the *label vector*.

Generalization

Enabling Generalization: Data

Training, Test and Validation

- Split $(\mathbf{X}, \mathbf{y})$ into
  - training data $(\mathbf{X}^{(\text{train})}, \mathbf{y}^{(\text{train})})$
  - validation data $(\mathbf{X}^{(\text{val})}, \mathbf{y}^{(\text{val})})$
  - test data $(\mathbf{X}^{(\text{test})}, \mathbf{y}^{(\text{test})})$
- While *training data* is to pick the optimal set of parameters (which specify elements from $\mathcal{M}$), using training and *validation data* in combination is for picking *hyperparameters*
- Hyperparameters can refer to choosing subsets of $\mathcal{M}$
- For example, depth of a neural network, and widths of hidden layers. They may also refer to specifications of cost function or optimization procedure

Enabling Generalization: Data

Training, Test and Validation

- $(\mathbf{X}^{(\text{test})}, \mathbf{y}^{(\text{test})})$ are never touched during training
- The final goal is to minimize the cost on the test data

☞ But we are not allowed to use it to reach that goal!

Over- and Underfitting

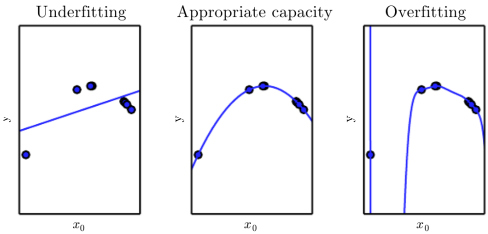
- *Underfitting:* Function underperforms on training data
  ☞ cannot be expected to perform well on test data either
- *Overfitting:* Function performs perfectly on training data
  ☞ Volatilities, noise affecting data lead to suboptimal performance on test data
- *Appropriate:* Leave room for deviations during training
  ☞ Can be expected to perform optimally on test data

What does that mean in practice, how to enable it?

***Principled Idea:* Prefer little complex functions**

Enabling Generalization: Model

Capacity, Under- and Overfitting



Left: Linear functions underfit
Center: Polynomials of degree 2 neither under- nor overfit
Right: Polynomials of degree 9 overfit

- Choose a class of models that has the right *capacity*
- Capacity too large: *overfitting*
- Capacity too small: *underfitting*

Enabling Generalization: Cost Function

Regularization

Let $C(f, f^*)$ be the cost function. Let $\mathbf{w} = (w_1, ..., w_k)$ be the parameters specifying elements of $f_{\mathbf{w}} \in \mathcal{M}$.

- Usually, C refers to only known data points. So, C evaluates as

$$C(f, f^*) = \sum_i C(f(x_i), y_i = f^*(x_i)) \tag{2}$$

  where x_i runs over all training data points.

- Add a *regularization term* to cost function, and choose $f_{\mathbf{w}}$ that yields minimal

$$C(f_{\mathbf{w}}, f^*) + \lambda\Omega(\mathbf{w}) \tag{3}$$

- λ is a hyperparameter

Enabling Generalization: Cost Function

Regularization

- Prominent examples:
  - *L_1 norm*: $\Omega(\mathbf{w}) := \sum_i |w_i|$
  - *L_2 norm*: $\Omega(\mathbf{w}) := \sum_i w_i^2$
- Rationale: Penalize too many non-zero weights
- Virtually less complex model, hence virtually less capacity

Enabling Generalization: Optimization

Early Stopping, Dropout

Optimization can be an iterative procedure.

- *Early stopping*: Stop the optimization procedure before cost function reaches an optimum on the training data.

  ☞ Function does not pick up all details, so is less complex

- *Dropout*: Neural network specific. Randomly remove neurons and optimize parameters for neurons remaining.

  ☞ Suppresses many weights to zero, reduces complexity

Prominent Model Examples

Supervised Learning

Example: Linear Regression

- Design matrix $\mathbf{X} \in \mathbb{R}^{m \times d}$, label vector $\mathbf{y} \in \mathbb{R}^m$
- Model class: Let $\mathbf{w} \in \mathbb{R}^d$

$$f_{\mathbf{w}} = f(\mathbf{x}; \mathbf{w}) : \begin{array}{ccc} \mathbb{R}^d & \longrightarrow & \mathbb{R} \\ \mathbf{x} & \mapsto & \mathbf{w}^T \mathbf{x} \end{array} \tag{4}$$

- *Remark*: Note that the case $\mathbf{w}^T \mathbf{x} + b$ can be treated as a special case to be included in $\mathcal{M}$, by augmenting vectors $\mathbf{x}_i$ by an entry 1 (think about this...)
- Cost function (recall $y_i = f^*(\mathbf{x}_i)$)

$$C(f, f^*) := \frac{1}{m} ||(f(\mathbf{x}_1), ..., f(\mathbf{x}_m)) - \mathbf{y}||_2^2 = \frac{1}{m} \sum_{i=1}^{m} (f(\mathbf{x}_i) - \mathbf{y}_i)^2 \tag{5}$$

Optimization

- Solve for

$$\nabla_{\mathbf{w}} C(f_{\mathbf{w}}, f^*) = 0 \tag{6}$$

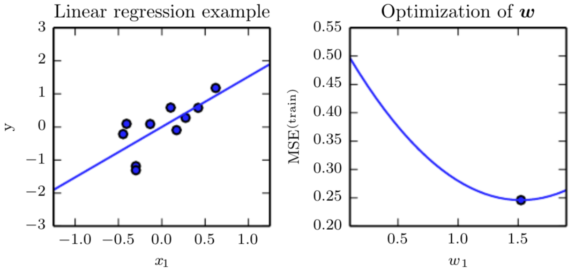
  to achieve a minimum. This yields the *normal equations*

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} \tag{7}$$

- *Global optimum* if $\mathbf{X}^T\mathbf{X}$ is invertible
- Do this on *training data* (so $\mathbf{X} = \mathbf{X}^{(\text{train})}, \mathbf{y} = \mathbf{y}^{(\text{train})}$) only. Hope that cost on test data is small.

Supervised Learning

Linear Regression: Normal Equations



- *Left*: Data points, and the linear function $y = w_1 x$ that approximates them best
- *Right*: Mean squared error (MSE) depending on w_1
- *Remark on Perceptrons*: Optimizing is different, but also supported by a very easy optimization scheme (the *perceptron algorithm*)

Supervised Learning

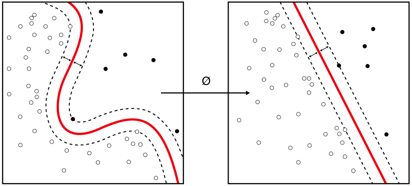
Popular Models: Support Vector Machines

- *Realization*: From (7), write

$$\mathbf{w}^T\mathbf{x} = \sum_{i=1}^{m} \alpha_i \mathbf{x}^T\mathbf{x}_i = \sum_{i=1}^{m} \alpha_i \langle \mathbf{x}, \mathbf{x}_i \rangle \tag{8}$$

- Replace $\langle ., . \rangle$ by different *kernel* (i.e. scalar product) $k(., .)$, that is by computing $\langle \phi(.), \phi(.) \rangle$ for appropriate ϕ

☞ Optimize for choosing good α's: still easy to optimize both for regression and classification!



Supervised Learning

Popular Models: Nearest Neighbor Classification

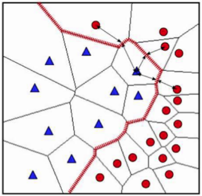
- Consider appropriate distance measure

$$D : \mathbb{R}^d \times \mathbb{R}^d \longrightarrow \mathbb{R}_+ \tag{9}$$

- For unknown data point $\mathbf{x}$, determine the closest given data point

$$\mathbf{x}_{i^*} := \mathrm{argmin}_i(D(\mathbf{x}, \mathbf{x}_i)) \tag{10}$$

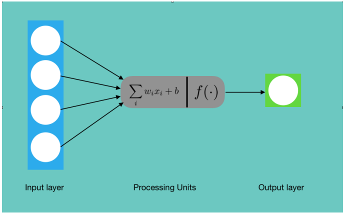
- Predict label of $\mathbf{x}$ as y_{i^*}



Neural Networks

NEURONS

LINEAR + ACTIVATION FUNCTION



$$\text{output} = a(w^T \cdot x + b)$$

Note: replace f in Figure by a!

Neuron: linear function followed by activation function

Examples

- Linear regression:

$$a = \mathrm{Id}$$

a is identity function

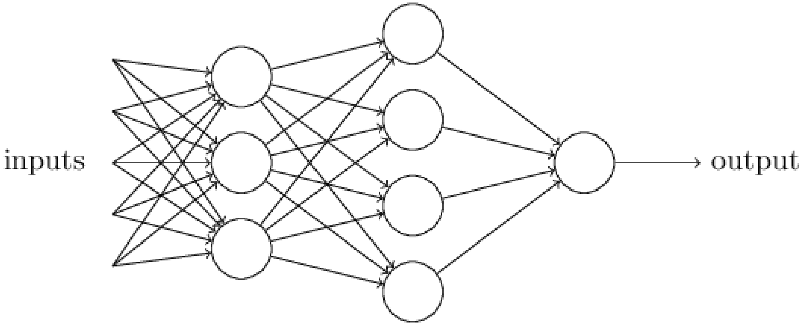
- Perceptron:

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

a is step function

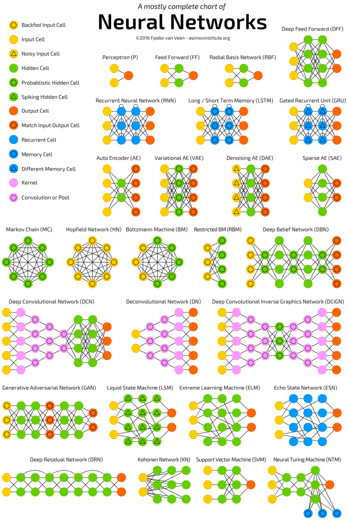
Neural Networks

Concatenating Neurons



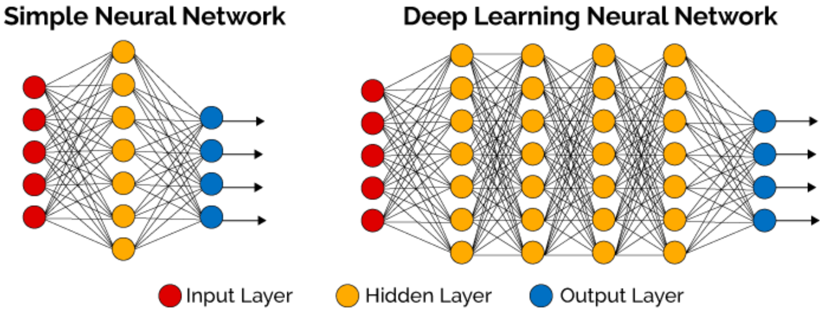
Neural Networks

Architectures



UNIVERSITÄT
BIELEFELD

Deep Neural Networks



Width = Number of nodes in a hidden layer

Depth = Number of hidden layers

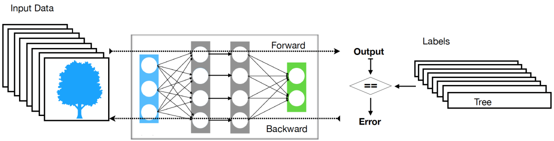
Deep = depth ≥ 8 (for historical reasons)

Neural Networks

Formal Definition

- Let $\mathbf{x}^l \in \mathbb{R}^{d(l)}$ be all outputs from neurons in layer l, where $d(l)$ is the *width* of layer l.
- Let $y \in V$ be the output.
- Let $\mathbf{x} =: \mathbf{x}^0$ be the input.
- Then
$$\mathbf{x}^l = \mathbf{a}^l(\mathbf{W}^{(l)}\mathbf{x}^{l-1} + \mathbf{b}^l)$$
where $\mathbf{a}^l(.) = (a_1^l(.), ..., a_{d(l)}^l(.))$, $\mathbf{W}^{(l)} \in \mathbb{R}^{d(l) \times d(l-1)}$, $\mathbf{b}^l \in \mathbb{R}^{d(l)}$
- The function f representing a neural network with L layers (with depth L) can be written
$$y = f(\mathbf{x}^0) = f^{(L)}(f^{(L-1)}(...(f^{(1)}(\mathbf{x}^{(0)}))...))$$
where $\mathbf{x}^l = f^{(l)}(\mathbf{x}^{l-1}) = \mathbf{a}^{\mathbf{l}}(\mathbf{W}^{(\mathbf{l})}\mathbf{x}^{l-1} + \mathbf{b}^{\mathbf{l}})$

Training: Backpropagation



- *E.g.* let X be a set of images, labels 1 and 0: tree or not
- Let
$$f_{(\mathbf{w},\mathbf{b})} : X \to \{0,1\} \quad \text{and} \quad \hat{f} : X \to \{0,1\}$$
network function ($f_{\mathbf{w},\mathbf{b}}$) and true function ($\hat{f}$)
- $L(f_{(\mathbf{w},\mathbf{b})}, \hat{f})$ loss function, differentiable in network parameters **w**, **b**
- *Back Propagation*: Minimize $L(f, \hat{f})$ through gradient descent
  - ☞ Heavily parallelizable!
- Decisive: Ratio number of parameters and training data

Materials / Outlook

- `http://neuralnetworksanddeeplearning.com`:
  Chapter 1, up to 'Perceptrons'
- `https://www.deeplearningbook.org/`:
  5.1, 5.2, 5.3, 5.7
- Next lecture:
  - Why "deep"?
  - Training challenges