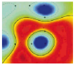
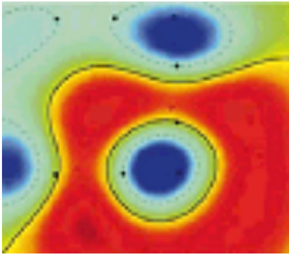


WiSe 2023/24

Machine Learning 1/1-X

Technische Universität Berlin



Lecture 9

Neural Networks 2

Outline

- Recap:
  - The Perceptron
  - Artificial Neural Networks
  - Forward / Backward Propagation
- Optimizing Neural Networks:
  - Bad Local Minima and Pathological Curvature
  - Initialization / Centering / Momentum
- Regularizing Neural Networks:
  - Hinge Loss
  - Perturbations

Recap: The Perceptron

F. Rosenblatt (1928–1971)

- Proposed by F. Rosenblatt in 1958.
- Classifier that perfectly separates training data (if the data is linearly separable).
- Trained using a simple and cheap iterative procedure.
- The perceptron gave rise to artificial neural networks.

Recap: The Perceptron Algorithm

Consider the linear model $y = \boldsymbol{w}^\top \boldsymbol{x} + b$, where $\text{sign}(y)$ gives the classification decision. Let $t \in \{-1, +1\}$ be the binary class label associated to $\boldsymbol{x}$.

Algorithm

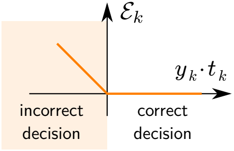
- Iterate over all data points $\{(\boldsymbol{x}_k, t_k)\,;\ k = 1\ldots, N\}$ (multiple times).
  - Compute $y_k = \boldsymbol{w}^\top \boldsymbol{x}_k + b$
  - If $\boldsymbol{x}_k$ is correctly classified (i.e. $\text{sign}(y_k) = t_k$), continue.
  - If $\boldsymbol{x}_k$ is wrongly classified (i.e. $\text{sign}(y_k) \neq t_k$), apply:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \gamma \cdot \boldsymbol{x}_k t_k \quad \text{and} \quad b \leftarrow b + \gamma \cdot t_k$$

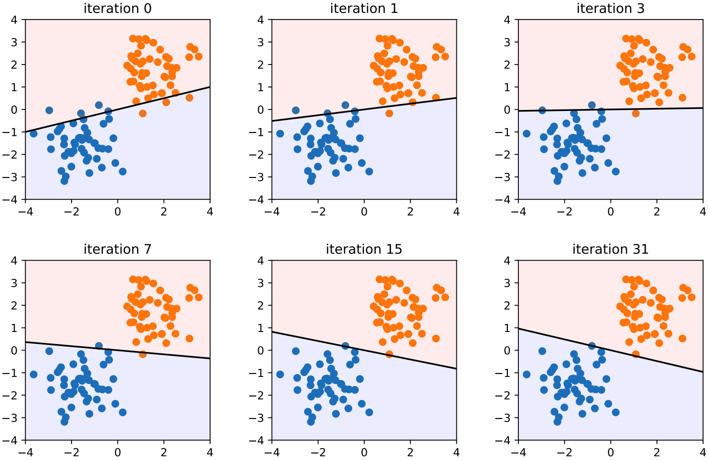
- Stop once all examples are correctly classified.

The perceptron can also be seen as a stochastic gradient descent of the error function

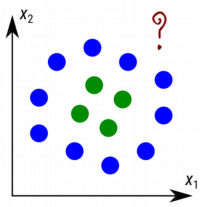
$$\mathcal{E}(\boldsymbol{w}, b) = \frac{1}{N} \sum_{k=1}^{N} \underbrace{\max(0, -y_k t_k)}_{\mathcal{E}_k(\boldsymbol{w}, b)}$$



Recap: Perceptron at Work



Recap: Nonlinear Classification



Observation:

- The perceptron builds a decision function which is linear in input space.
- In practice, the data may not be linearly separable.

Key Idea:

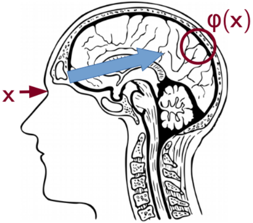
- Transform the data nonlinearly through some function Φ before applying the linear decision function.

$$f(\boldsymbol{x}) = \boldsymbol{w}^\top \Phi(\boldsymbol{x}) + b$$

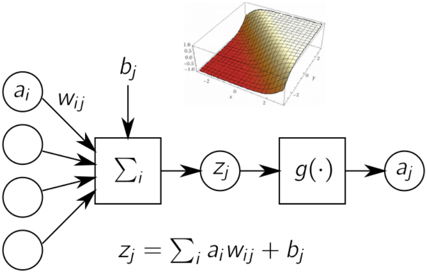
- *Example:* $\Phi(\boldsymbol{x}) = [x_1, x_2, x_1^2, x_2^2, x_1 x_2]$ and $\boldsymbol{w} \in \mathbb{R}^5$.

Recap: Artificial Neural Networks

- Models that are inspired by the way the brain represents sensory input and learn from repeated stimuli.
- Neuron activations can be seen as a nonlinear transformation of the sensory input (similar to $\Phi(\boldsymbol{x})$).
- The neural representation adapts itself after repeated exposure to certain stimuli (plasticity).



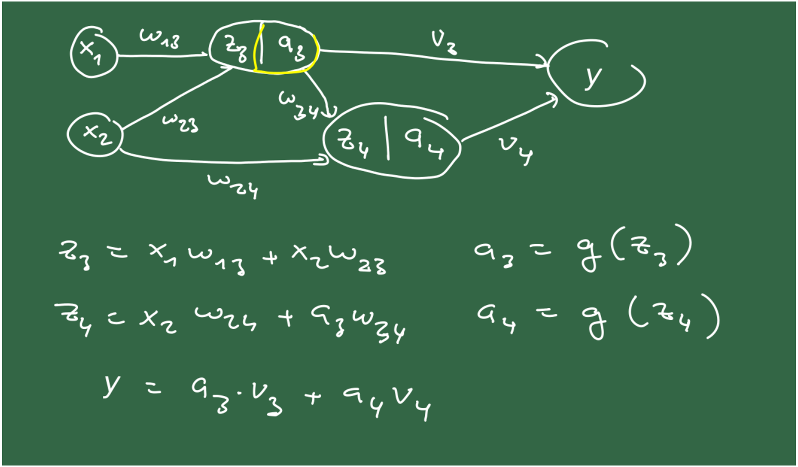
Recap: The Artificial Neuron



$$z_j = \sum_i a_i w_{ij} + b_j$$
$$a_j = g(z_j)$$

- Simple multivariate, nonlinear and differentiable function.
- Ultra-simplification of the biological neuron that retains two key properties: (1) ability to produce complex nonlinear representations when many neurons are interconnected (2) ability to learn from the data.

Recap: Artificial Neural Networks



$$z_3 = x_1 w_{13} + x_2 w_{23} \qquad a_3 = g(z_3)$$

$$z_4 = x_2 w_{24} + a_3 w_{34} \qquad a_4 = g(z_4)$$

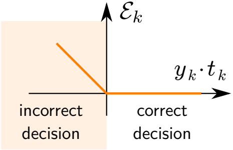
$$y = a_3 \cdot v_3 + a_4 v_4$$

Recap: Training a Neural Network

Idea:

- Use the same error function as the perceptron, but replace the perceptron output by the neural network output:

$$\mathcal{E}(\theta) = \frac{1}{N} \sum_{k=1}^{N} \underbrace{\max(0, -y_k t_k)}_{\mathcal{E}_k(\theta)}$$



and compute the gradient of the error function w.r.t. the parameters θ of the neural network.

Question:

- How to compute the gradient of the error function?

Recap: Error Backpropagation

Idea:

- The gradient can be expressed in terms of gradient in the higher layers using the multivariate chain rule.

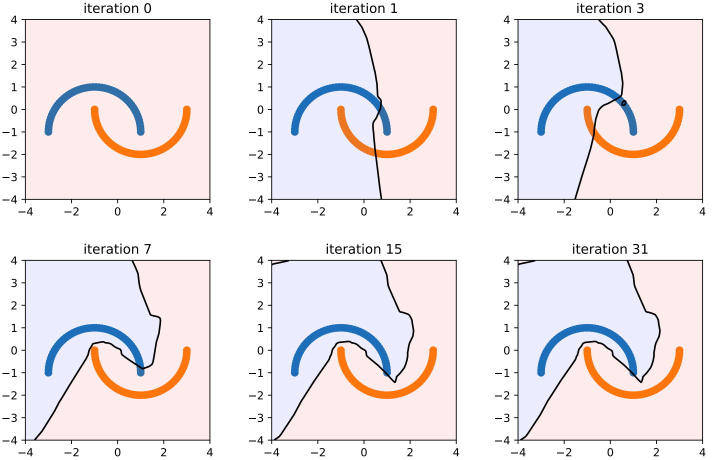
$$\frac{\partial \mathcal{E}}{\partial z_i} = \sum_j \frac{\partial z_j}{\partial z_i} \frac{\partial \mathcal{E}}{\partial z_j}$$

- Similarly, one can then extract the gradient w.r.t. the parameters of the model as:

$$\frac{\partial \mathcal{E}}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}} \frac{\partial \mathcal{E}}{\partial z_j}$$

- Gradients can be computed one after the other in a message passing fashion in O(forward pass). The algorithm is known as *error backpropagation*.

Recap: Neural Network at Work



Today's Lecture

Part 1:

- Optimizing Neural Networks
  (making sure neural networks can be trained effectively and efficiently).

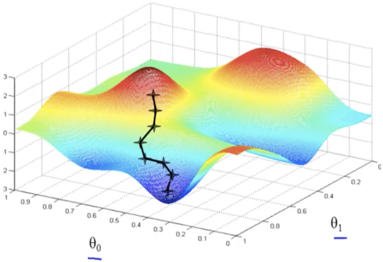
Part 2:

- Regularizing Neural Networks
  (making sure the learned models are good and generalize well).

Part 1: Optimizing Neural Networks

Questions:

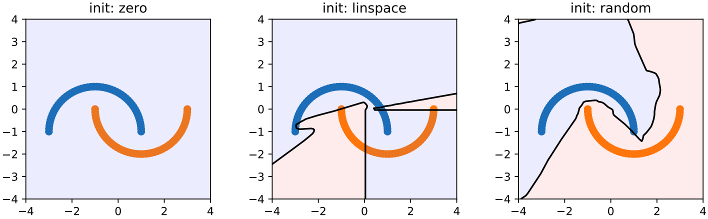
- How hard is it to optimize a neural network?
- Are we guaranteed to converge to a good local minimum?
- How quickly do we converge to this local minimum?



Neural Networks: Initialization

Experiment:

- Train the network with different weight initializations:



Observations:

- Some runs do not converge to a solution that perfectly classifies the data (e.g. when weights are initialized to zero, training is stuck on a plateau corresponding to a constant classifier).

Neural Networks: Initialization

- Weights should be initialized *randomly* (it breaks symmetries in parameter space and reduces the risk of landing in bad local minima or getting stuck on plateaus).
- If necessary, train the network *multiple times* using different random seeds, and retain the network with the *lowest error*.

He-et-al[1] random initialization heuristic:

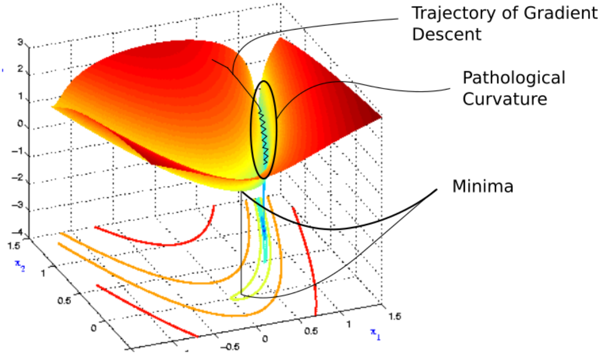
For neural networks with ReLU neurons, it is recommended to use: $w_{ij} \sim \mathcal{N}(0, \sigma^2)$ with $\sigma = \sqrt{2/\#\text{ input connections}}$ and where $\mathcal{N}$ is the Gaussian distribution.

Example: For a two-layer neural network with 50 *input features,* 200 *hidden neurons, and* 1 *output, weights in the first layer should be initialized using* $\sigma = \sqrt{2/50}$*, and in the second layer using* $\sigma = \sqrt{2/200}$*.*

[1] He et al. (2015) Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification

Neural Networks: Pathological Curvature

Another problem: The neural network may converge to some good local optimum but slowly due to *pathological curvature* of the error function.



Source: Martens. Deep Learning via Hessian-free Optimization U Toronto, 2010.

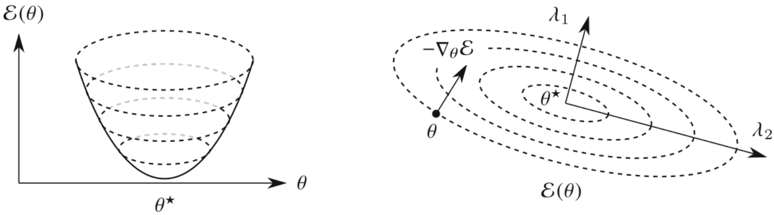
Neural Networks: Pathological Curvature

The curvature of the error function can be characterized by looking at the eigenvalues of the Hessian of the error function:

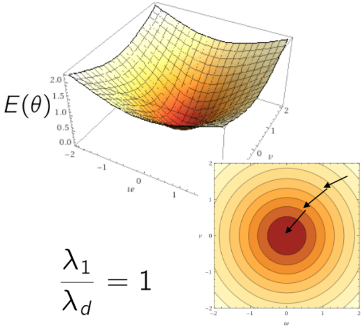
$$H = \left(\frac{\partial^2 \mathcal{E}(\theta)}{\partial\theta_i \partial\theta_j}\right)_{ij} \qquad\qquad \lambda_1, \dots, \lambda_{|\theta|} = \text{eigvals}(H)$$

The higher the ratio between the highest and the lowest eigenvalue (aka. condition number), the slower the convergence.

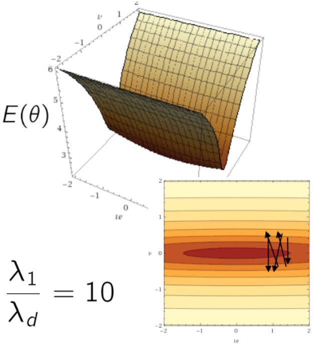
Two-dimensional example:



Neural Networks: Pathological Curvature



$$\frac{\lambda_1}{\lambda_d} = 1$$

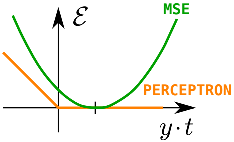


$$\frac{\lambda_1}{\lambda_d} = 10$$

The lower the condition number, the better.

Hessian Analysis (Linear Case, MSE)

Consider the simplest case of a homogeneous linear model $y = \boldsymbol{w}^\top \boldsymbol{x}$. Consider $\mathcal{E}(\boldsymbol{w}) = \mathbb{E}[0.5 \cdot (1 - y \cdot t)^2]$, the mean square error (MSE), which is easier to analyze than the perceptron loss.



The Hessian of the error function is given by:

$$H = \frac{\partial^2 \mathcal{E}}{\partial \boldsymbol{w}^2} = \mathbb{E}[\boldsymbol{x}\boldsymbol{x}^\top]$$

i.e. the data uncentered covariance.

Condition Number: Assuming $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \sigma^2 I)$,the Hessian reduces to $H = \sigma^2 I + \boldsymbol{\mu}\boldsymbol{\mu}^\top$, and the condition number is

$$\frac{\lambda_1}{\lambda_d} = 1 + \frac{\|\boldsymbol{\mu}\|^2}{\sigma^2}.$$

(show this in the homework). In other words, the condition number can be reduced by **centering the data** before training.

Hessian Analysis (General Case, MSE)

- Hessian of a neural network (LeCun et al. 1998):

$$H = \frac{\partial^2 \mathcal{E}}{\partial \theta^2} = \frac{\partial Y}{\partial \theta}^\top \frac{\partial^2 \mathcal{E}}{\partial Y^2} \frac{\partial Y}{\partial \theta} + \frac{\partial \mathcal{E}}{\partial Y} \frac{\partial^2 Y}{\partial \theta^2} \tag{1}$$

  where Y is the vector containing the predictions for all data points.

- Consider the part of the Hessian relating parameters of a specific neuron k. From Eq. (1), and assuming MSE, we get:

$$[H_k]_{jj'} = \frac{\partial^2 \mathcal{E}}{\partial w_{jk} \partial w_{j'k}} = \mathbb{E}[a_j a_{j'} \delta_k^2] + \mathbb{E}[a_j \frac{\partial \delta_k}{\partial w_{j'k}} \cdot (y - t)]$$

  where δ_k is a shortcut notation for $\partial \mathcal{E} / \partial z_k$.

- Assuming $a_j a_{j'}$ and δ_k^2 to be independent, and δ_k insensitive to $w_{j'k}$, we get:

$$H_k \approx \mathbb{E}[\boldsymbol{a}\boldsymbol{a}^\top] \cdot \mathbb{E}[\delta_k^2]$$

  which has a similar structure as in the analysis for the linear case.

- This motivates **centering activations** as well.

Hessian Analysis: Takeaway Message

To lower the condition number and consequently ease optimization of the neural network:

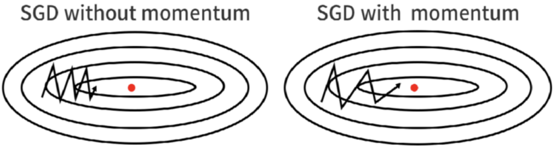
Center the input data, and also activations at each layer.

Note: Exactly centering activations can be difficult in practice, however, reasonably centered activations can be promoted by choosing an activation function satisfying $g(0) = 0$.

- Examples of **suitable** activation functions (where $g(0) = 0$) are the *ReLU* ($\max(0, z)$), the *centered softplus* ($\log(1 + \exp(z)) - \log(2)$), or the *hyperbolic tangent* ($\tanh(z)$).
- Examples of **less suitable** activation functions (where $g(0) \neq 0$) are the *standard softplus* ($\log(1 + \exp(z))$), or the *logistic sigmoid* ($\exp(z)/(1 + \exp(z))$).

Improving Gradient Descent

- Centering may significantly reduce the pathological curvature, but does not completely eliminate it (especially for deep neural networks).
- A complementary approach to centering is to modify the optimization procedure to move faster along directions of low curvature.
- This can be achieved by introducing a '*momentum*' in the gradient descent:



Improving Gradient Descent

(Stochastic) Gradient Descent:

$$\theta = \theta - \gamma \cdot \frac{\partial \mathcal{E}_k}{\partial \theta} \qquad \text{with} \quad k \sim \{1, \dots, N\}$$

where γ is the learning rate.

(Stochastic) Gradient Descent + Momentum:

$$\Delta_{\text{new}} = \mu \Delta_{\text{old}} + \frac{\partial \mathcal{E}_k}{\partial \theta} \qquad \text{with} \quad k \sim \{1, \dots, N\}$$

$$\theta = \theta - \gamma \Delta_{\text{new}}$$

where γ is the learning rate and μ is the momentum.

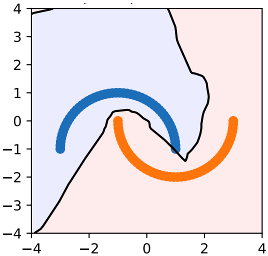
Heuristics:

- Typical learning rate and momentum: $\gamma = 0.01$ and $\mu = 0.9$ (need to be reduced if training diverges).

Part 2: Regularizing Neural Networks

Questions:

- Do neural networks trained with gradient descent and the perceptron loss yield good models?
- How to help neural networks to learn solutions that generalize well to new data?



Learning Well-Generalizing Solutions

So far:

- We have used the perceptron loss:

$$\mathcal{E}_k(\theta) = \max(0, -y_k t_k)$$

which becomes zero as soon as the current data point is being correctly classified.

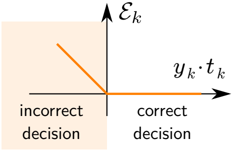
Idea:

- Impose a penalty to points that lie too close to the decision boundary (i.e. add a margin), aka. the Hinge Loss:

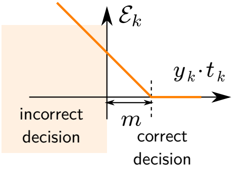
$$\mathcal{E}_k(\theta) = \max(0, 1 - y_k t_k)$$

This is similar to the slack variables penalties in the soft-margin SVM formulation.

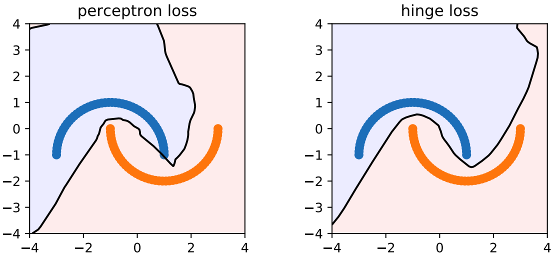
without margin



with margin



Hinge Loss at Work



Observation:

- Using the hinge loss, the decision function of the neural network moves away from the data.
- This leads to a better generalization performance.

More Loss Functions

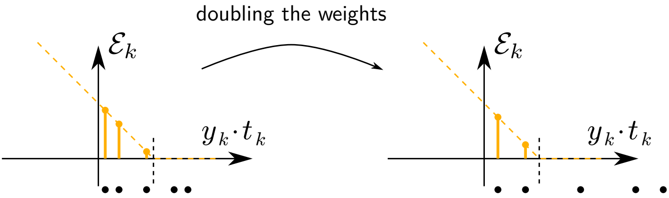
Name	Formula	Margin	Outliers[2]
Perceptron loss	$\max(0, -y \cdot t)$	no	yes
Hinge loss	$\max(0, 1 - y \cdot t)$	yes	yes
Squared hinge loss	$\max(0, 1 - y \cdot t)^2$	yes	no
Log-loss	$\log(1 + \exp(-y \cdot t))$	yes	yes

[2] Indicates whether the loss function is robust to outliers (e.g. mislabelings).

What is Still Missing?

Problem:

- If multiplying the weights by some factor, the hinge loss can be trivially reduced until it reaches zero, without actually changing the decision boundary.



- In the SVM, this was avoided by constraining $\|\boldsymbol{w}\|^2$ to be small.

Idea:

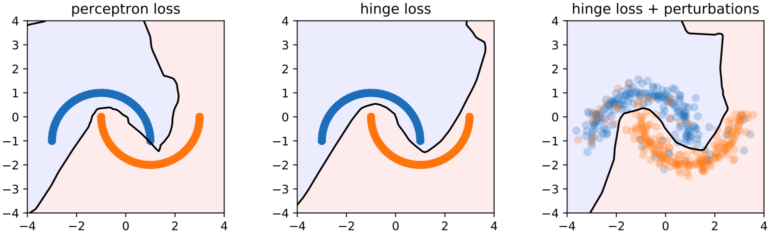
- To arrive at a well-generaling model, we also need to regularize the neural network itself.

Regularizing the Neural Network

Many approaches have been proposed and are often combined in practice. Some of the most popular ones are:

- **Weight decay:** Add a term $\lambda \cdot \|\theta\|^2$ to the objective. This is equivalent to multiplying the weights at each training iteration by some constant factor.
- **Data perturbation:** Apply random perturbations to the input examples (e.g. Gaussian additive noise), and to the labels (random flips).
- **Representation perturbation:** The Dropout method (Srivastava'14) extends the idea of data perturbation to intermediate representations, by randomly turning on and off neurons while training.
- **Training noise:** Stochastic gradient descent was found to have a regularizing effect compared to standard gradient descent.

Data Perturbation at Work



Observation:

- Data perturbations pull the decision boundary further away from the class manifolds, and typically further improves generalization.

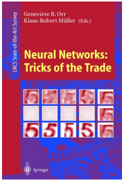
Neural Networks vs. SVMs/Kernels

SVMs/Kernels:

- Relatively straightforward to optimize and regularize.
- Doesn't scale well to very large datasets with millions of examples.

Neural Networks:

- Can be trained on very large distributions (with GPUs).
- Challenging to optimize and regularize neural networks. Lots of tricks and heursistics involved. Can make results harder to reproduce and less transparent.



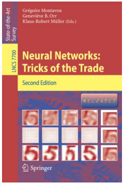
LNCS State-of-the-Art Survey

Genevieve B. Orr
Klaus-Robert Müller (Eds.)

Neural Networks:
Tricks of the Trade

Springer

,



State-of-the-Art Survey

LNCS 7700

Grégoire Montavon
Geneviève B. Orr
Klaus-Robert Müller (Eds.)

Neural Networks:
Tricks of the Trade

Second Edition

RELOADED

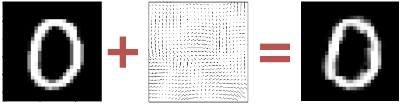
Springer

, etc.

Neural Networks: Futher Topics

Further Topics:

- **Incorporate prior knowledge:** Data extension, weight sharing, unsupervised pretraining, self-supervised pretraining, transfer learning, multitask learning.



- **Predicting structured data:** Convolutional neural networks, recurrent neural networks, transformers, graph neural networks (→ ML2).
- **Beyond classification:** Neural networks for regression, mixture density networks, structured prediction (→ ML2).
- **Adversarial robustness:** Adversarial training, model certification.
- **Explainability:** Extracting insights from a neural network, Clever Hans effect (later this semester).

Summary

Previous lecture:

- Neural networks derive their **high representation power** from a large number of simple interconnected components (neurons), and the gradient of a neural network w.r.t. its parameters can be efficiently extracted (using **backpropagation**).

Today's lecture:

- In practice, there are two major challenges with neural networks (1) how to **optimize** them, and (2) how to **regularize** them so that they generalize well.
- The problem of optimization can be studied analytically by computing the **Hessian** of the error function and the **condition number**.
- Well-generalizing models can be obtained by introducing **margin losses** and **perturbations**.