Backpropagation

Leander Kurscheidt

Hack & Söhne

2018

Backpropagation

- We've got the forward propagation, but how to train the neural network?
- many approaches, but Backward-Propagation is enabling the whole Deep-Learning Hype

for BP to work, **everything** needs to be differentiable!!!!

Intuition: Learning as an Optimization-Problem

in theory, it's easy!

1. assign every instance of neural network a score through a differentiable function (error function), e.g. derivation from target
2. optimize the score

Intuition: Learning as an Optimization-Problem

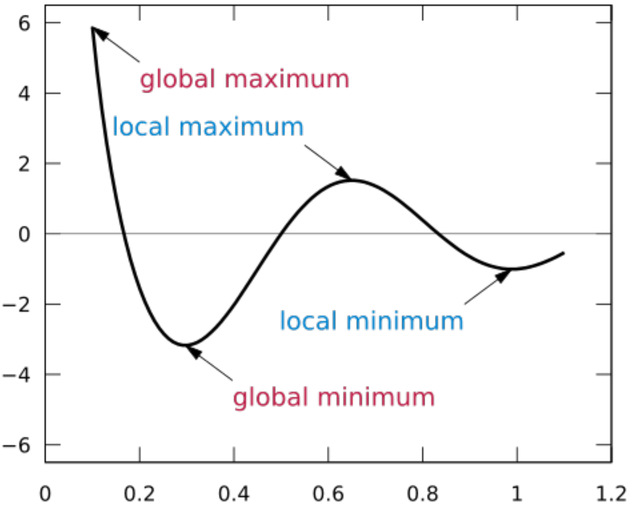


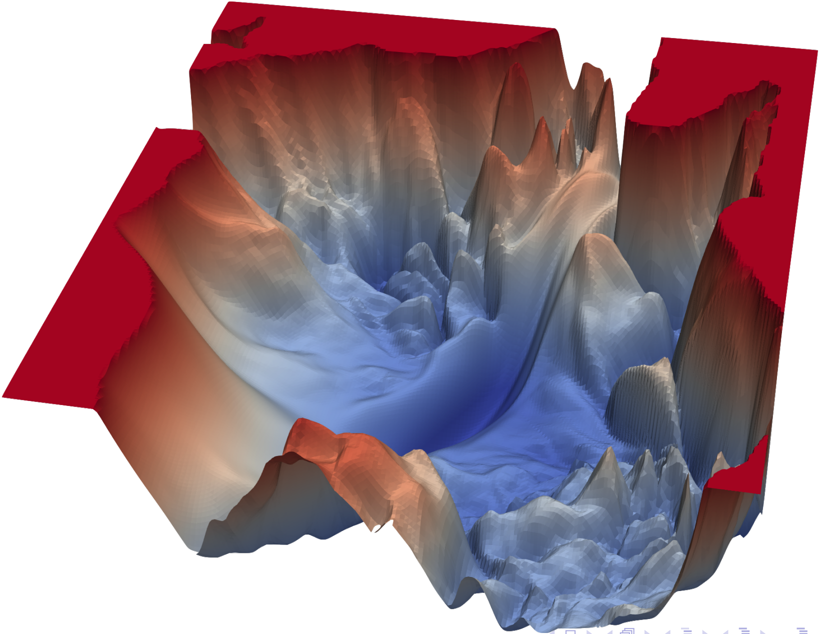
Figure: [Com07]

Intuition: Learning as an Optimization-Problem

common vocabulary:

1. a **Loss-Function** L is something I want to minimize
2. an **Error-Function** $E : Output_{NN} \times Target \to \mathbb{R}$
   $\Rightarrow$ in our context: Loss-Function $=$ Error-Function

Intuition: Learning as an Optimization-Problem



The BP-Algorithm

essentially: nothing more than the chain rule!

A naive formulation

w_{ijz} is the weight z of neuron j in layer z
Gradients of our Error Function: $\nabla E = \{\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, ..., \frac{\partial E}{\partial w_l}\}$

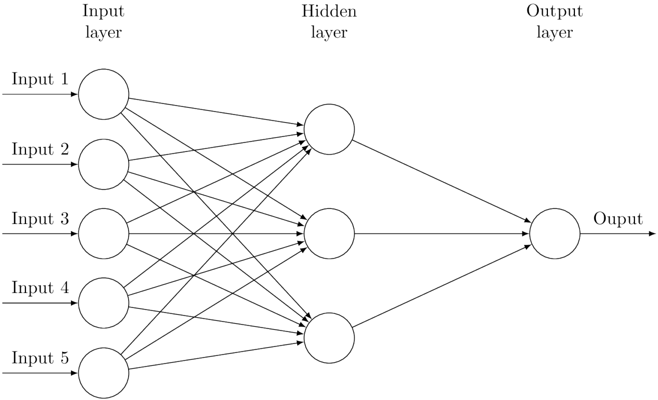
A naive formulation

w_{ijz} is the weight z of neuron j in layer z
Gradients of our Error Function: $\nabla E = \{\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, ..., \frac{\partial E}{\partial w_l}\}$

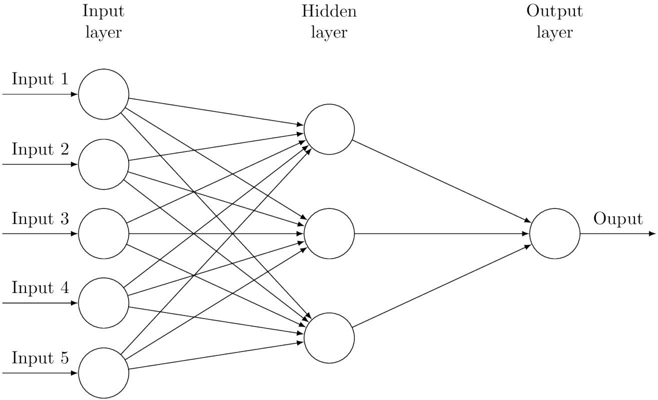
Definition

update weight: $\Delta w_{ij} = -\lambda \cdot \frac{\partial E}{\partial w_{ij}}$, *where* λ *is the learning rate ("step size")*

This is expensive!

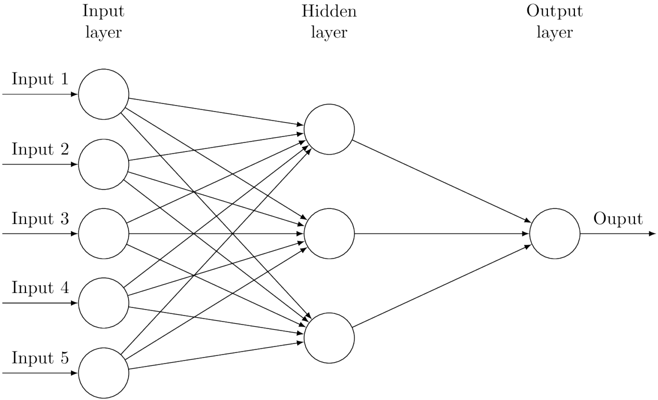


This is expensive!



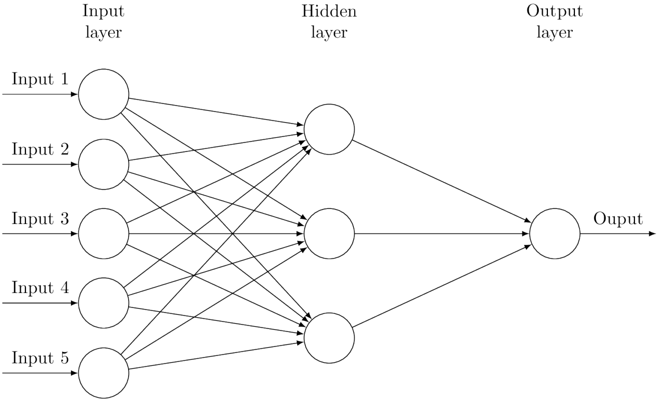
- $5 \cdot 3 + 5 + 3 + 1 = 24$ weights!

This is expensive!



- $5 \cdot 3 + 5 + 3 + 1 = 24$ weights!
- often hundreds of thousands of weights in real scenarios!

This is expensive!



- $5 \cdot 3 + 5 + 3 + 1 = 24$ weights!
- often hundreds of thousands of weights in real scenarios!
- and this is **one** update

Recap: Generalization of the Chain-Rule

single variable: $[f \circ g(x)]' = (f' \circ g)(x) \cdot g'(x)$
different notation: for $y = f(t)$, $t = g(x)$: $\frac{dy}{dx} = \frac{dy}{dt}\frac{dt}{dx}$

generalized to multivar.:
For $y = f(m_1, m_2)$,$m_1 = g(x_1, x_2)$ and $m_2 = h(x_1, x_2)$:
$\frac{\partial y}{\partial x_1} = \frac{\partial y}{\partial m_1}\frac{\partial m_1}{\partial x_1} + \frac{\partial y}{\partial m_2}\frac{\partial m_2}{\partial x_1}$

Chain rule to the rescue!

w_{ij} is a weight i of layer j, l_i is the number of weights in layer i

$$\Delta w_{ij} = -\lambda \cdot \frac{\partial E}{\partial w_{ij}}$$

Chain rule to the rescue!

w_{ij} is a weight i of layer j, l_i is the number of weights in layer i

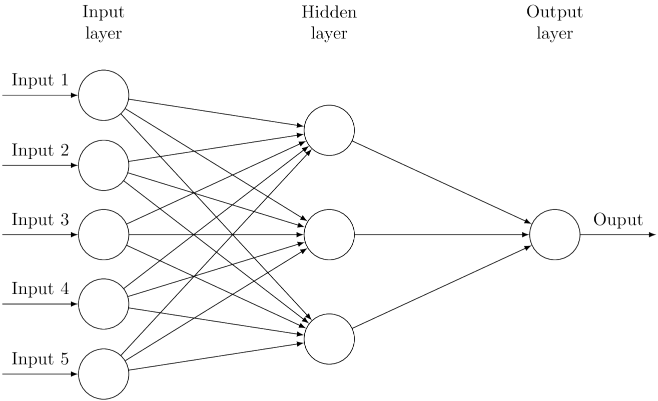
$$\begin{aligned}
\Delta w_{ij} &= -\lambda \cdot \frac{\partial E}{\partial w_{ij}} \\
&= -\lambda \cdot \sum_{n=0}^{l_{j+1}} \frac{\partial E}{\partial w_{n(j+1)}} \frac{\partial w_{n(j+1)}}{\partial w_{ij}}
\end{aligned}$$

Chain rule to the rescue!

w_{ij} is a weight i of layer j, l_i is the number of weights in layer i

$$\begin{aligned}
\Delta w_{ij} &= -\lambda \cdot \frac{\partial E}{\partial w_{ij}} \\
&= -\lambda \cdot \sum_{n=0}^{l_{j+1}} \frac{\partial E}{\partial w_{n(j+1)}} \frac{\partial w_{n(j+1)}}{\partial w_{ij}} \\
&= -\lambda \cdot \sum_{n_1=0}^{l_{j+1}} \Big(\sum_{n_2=0}^{l_{j+2}} \frac{\partial E}{\partial w_{n_2(j+2)}} \frac{\partial w_{n_2(j+2)}}{\partial w_{n_1(j+1)}} \Big) \frac{\partial w_{n_1(j+1)}}{\partial w_{ij}}
\end{aligned}$$

Chain rule to the rescue!



Scetch of the algorithm

TODO

Wikimedia Commons. *Extrema example original*. 2007. URL: https://commons.wikimedia.org/wiki/File:Extrema_example_original.svg.

Hao Li et al. "Visualizing the Loss Landscape of Neural Nets". In: *CoRR* abs/1712.09913 (2017). arXiv: 1712.09913. URL: http://arxiv.org/abs/1712.09913.