

Convolutional Neural Networks

Julius Taylor

0000000

stud3@email.com

Shawn Cala

4921431

shawn.cala@gmail.com

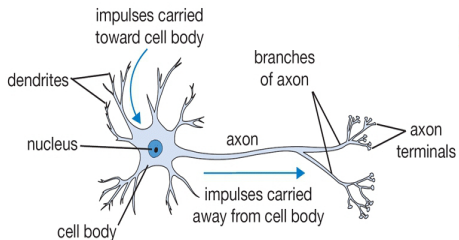
February 5, 2017

Table of Contents

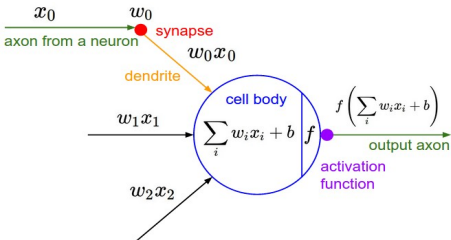
- 1 Background
 - Neural Networks
 - Structure
- 2 Convolutional Neural Networks
 - Concept
 - Kernel function
 - Application
- 3 Components
 - Activation function
 - Backpropagation
- 4 Problems
 - Overfitting
 - Underfitting
- 5 Code

Neural Networks

Primary motivation: Neural Networks mathematically simulate biological functionalities of the human brain



(a) biological model



(b) mathematical representation

Figure: neuronal model and computational abstraction

Structure

Neural Networks generally contain:

- an n-dimensional input
- one or many layers of interconnected neurons
- an output-layer

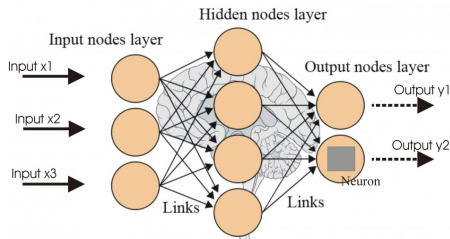


Figure: Basic concept of a Neural Network

Convolutional Neural Networks (CNNs) are a subtype of Neural Networks:

- all neurons in a layer are identical
- layers are interconnected through a kernel function
- different types of layers are used

Background and Motivation - Convolutional Neural Networks

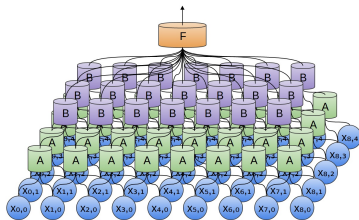


Figure: 2-dimensional CNN

- 1 Using identical copies of the same neuron allows for complex models with few parameters
- 2 Convolutional layers are not fully connected; each neurons is locally connected with a subsection of the previous layer. This is represented mathematically through a kernel function.

Kernel function

Kernel functions are an important component in many Neural Networks.

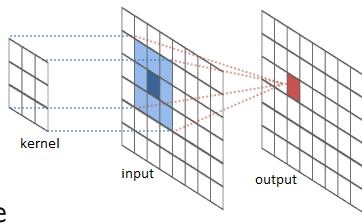


Figure: Forward propagation in a complex neural network
todo: besseres Bild, vll gif mit kernelfunktion

Application

Visual Object Recognition



Figure: ImageNet Classification with Deep Convolutional Neural Networks

ImageNet by Krizhevsky et al (2012) classified 1.2 million high-resolution images in the ImageNet LSVRC-2010 into 1000 different classes. It achieves error rates of 37.5% for the top result and 17.0% for the top-5 results

Application

Text Classification

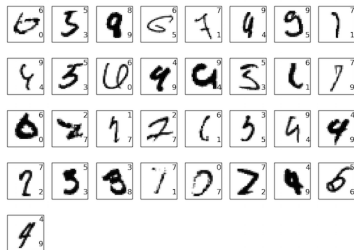


Figure: Excerpt of mnist

fehlt: etwas Erklärung hierzu

Components

hier Bild eines CNNs unseres Typs mit Komponenten

- 1 Neuron
- 2 Kernel-Function
- 3 Sigmoid function
- 4 softmax

Activation Function

Typically, an activation function is used to add nonlinearity to the network

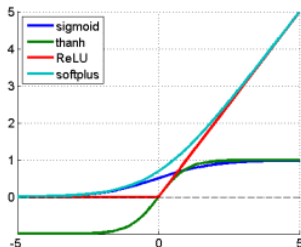


Figure: Different activation functions

Activation functions must be non linear and differentiable.
The employed network uses Sigmoid:

$$\sigma(x) = \frac{1}{(1+e^{(-x)})}$$

Components - Neuron in fully connected layer

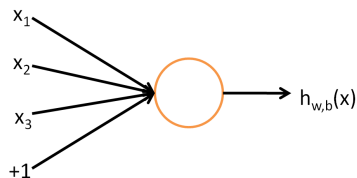


Figure: Single neuron of a fully connected layer

Input: connected neurons from the previous layer

Output: $f(\sum_i w_i x_i)$

with

w_i :

Components - Neuron in convolutional layer

To understand the concept behind Convolution we first demonstrate it in an example:

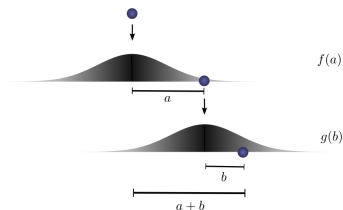


Figure: Probability distribution of ball thrown twice

We want to calculate the likelihood of the ball traveling a distance c after being thrown twice in a row.

Input layer

The input layer consists of an n -dimensional matrix, which contain the information to be processed

- Our example uses a 2-dimensional input which represents the pixels of the images
- asd

Input:

a, b : length of first and second throw $f(a), g(b)$: probability distribution of the balls location If we want $c = a + b$ the probability of this event is $f(a) \cdot g(b)$

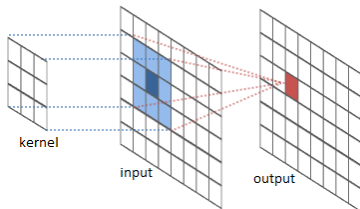
Since only the actual outcome is relevant to us, the actual values of a and b can be varied.

The total likelihood of the ball landing at c can also be summed over all probability distributions where $a + b = c$:

$$f * g(c) = \sum_{a+b=c} f(a) \cdot g(a)$$

The result of this sum is a convolution between a and b

Image processing with a kernel function can also be described as a



convolution:

Figure: Convolution between an input image and a kernel function

$$out = \sum_a \sum_b w_{ab} \cdot x$$

Backpropagation

Backpropagation is the key algorithm which makes training Neural Networks feasible by greatly increasing the efficiency of learning algorithms.

To understand Backpropagation we first introduce the concept of forward propagation:

Forward propagation

Forward propagation is the process of computing the output of a network for a given input

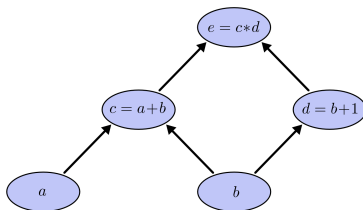


Figure: Computational graph of $e = (a + b) \cdot (b + 1)$

Forward propagation

We now arbitrarily set $a = 2$ and $b = 1$ for our example:

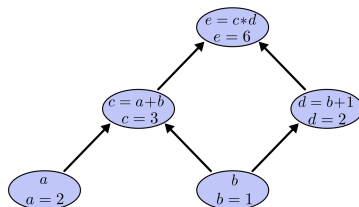


Figure: Computational graph of $e = (a + b) \cdot (b + 1)$

Forward propagation

If we want to train our network weights we must know how the output of the CNN is influenced by its layer weights. In our example our interest is how the result e is influenced by a and b . This is achieved by finding the respective partial derivatives

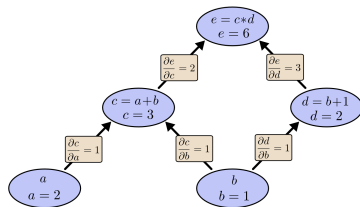


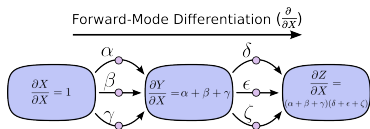
Figure: Partial derivatives

c is only indirectly influenced by a and b . To derive the exact influence, the chain rule can be applied: $\frac{\partial e}{\partial a} = \frac{\partial e}{\partial c} \cdot \frac{\partial c}{\partial a}$

$$\frac{\partial e}{\partial b} = \frac{\partial e}{\partial d} \cdot \frac{\partial e}{\partial c} \cdot \frac{\partial d}{\partial b} \cdot \frac{\partial c}{\partial b}$$

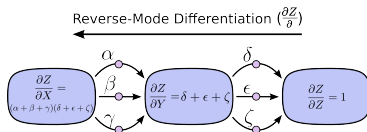
Forward propagation

In principle, this approach works for training a network. There are, however, obvious problems with exponential amounts of calculations if too many possible paths between input and output exist.



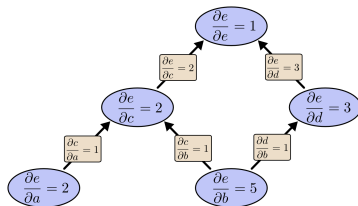
Backpropagation

Instead of determining the influence of one input on the output we calculate how the output is influenced by the previous layer.



Backpropagation

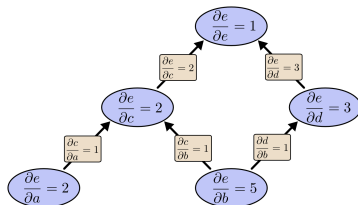
In our previous example executing a backward differentiation immediately gives us the derivative of the output relative to every input.



= massive speed in networks with many inputs/layers

Backpropagation

In our previous example executing a backward differentiation immediately gives us the derivative of the output relative to every input.



= massive speed in networks with many inputs/layers

Backpropagation

Lets reiterate our network:

Convolution: $conv = x \cdot W_1 + b_1$

Convolutional layer output: $out = \tanh(conv)$

Fully connected layer output: $\hat{y} = out \cdot W_2 + b_2$

Loss function: $L = - \sum_N y \cdot \ln \hat{y}$

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial W_2}$$

$$\frac{\partial L}{\partial \hat{y}} = \sum_N y \cdot \frac{\partial \ln \hat{y}}{\partial \hat{y}} = \hat{y} - y$$

$$\frac{\partial \hat{y}}{\partial W_2} = \frac{\partial out \cdot W_2 + b_2 = out^T}{\partial W_2} \frac{\partial L}{\partial W_2} = (\hat{y} - y) * out^T$$

$$\begin{aligned}\frac{\partial L}{\partial b_2} &= \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial b_2} \\ &= (\hat{y} - y) \cdot 1\end{aligned}$$

$$\begin{aligned}\frac{\partial L}{\partial W_1} &= \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial out} \cdot \frac{\partial out}{\partial conv} \cdot \frac{\partial conv}{\partial W_1} \\ \frac{\partial \hat{y}}{\partial conv} &= \frac{\partial out \cdot W_2 + b_2}{\partial out} = W_2 \frac{\partial out}{\partial conv} = \frac{\partial \tanh(conv)}{\partial conv} = 1 - \tanh^2 conv \\ \frac{\partial out}{\partial W_1} &= x^T \\ \frac{\partial L}{\partial W_1} &= x^T \cdot (1 - \tanh^2 conv \cdot (y - \hat{y}) \cdot W_2 \\ \frac{\partial L}{\partial b_1} &= \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial out} \cdot \frac{\partial out}{\partial conv} \cdot \frac{\partial conv}{\partial b_1} = \\ &= (1 - \tanh^2 conv \cdot (y - \hat{y}) \cdot W_2 \cdot 1\end{aligned}$$

Overfitting

Overfitting occurs when the complexity of the model relative to the training size is too high.

The model begins memorizing the training data rather than the underlying principle and easily loses its predictive power when the input is slightly altered.

Overfitting is prevented in a number of ways:

- 1 max-pooling layers
- 2 dropout layers
- 3

Code



Leslie Lamport, *L^AT_EX: a document preparation system*, Addison Wesley, Massachusetts, 2nd edition, 1994. @miscbworl, author = Christian Perone, title = Deep learning – Convolutional neural networks and feature extraction with Python, howpublished = "http://blog.christianperone.com/2015/08/convolutional-neural-networks-and-feature-extraction-with-year = 2015, note = "[Online; accessed 21.01.2017]"
<http://cs231n.github.io/convolutional-networks/>
<http://colah.github.io/posts/2014-07-Conv-Nets-Modular/>
<http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf>