Tutorial: Deep Learning (with Tensorflow)

Ole Salscheider

08.06.2018

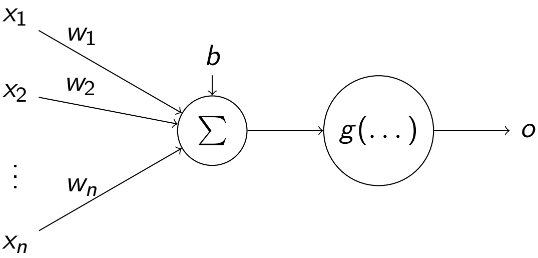
Overview

Introduction

History of Artificial Neural Networks

1940s	First ideas, Hebbian rule
1950s–1960s	First successful networks
1970s	Back-propagation algorithm
2010s	Deep neural networks

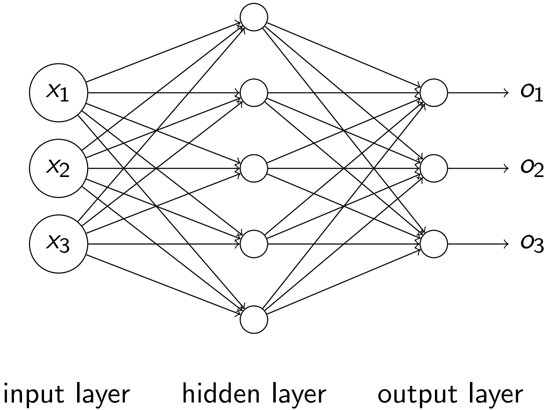
Artificial neuron



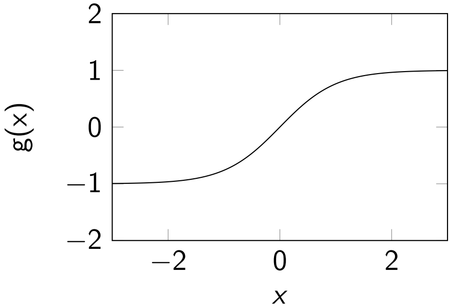
$$o = g\left(\sum_{i=1}^{n} w_i \cdot x_i + b\right) = g\left(\mathbf{w}^T \mathbf{x} + b\right)$$

$$\mathbf{o} = g\left(\mathbf{W}\mathbf{x} + \mathbf{b}\right)$$

Artificial neural network

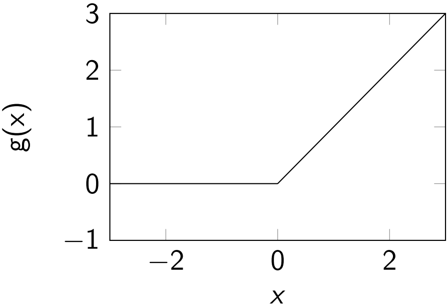


Activation function: tanh (or sigmoid)



- $g(x) = \tanh(x)$
- Was popular in the past...

Activation function: ReLU



- $g(x) = \max(0, x)$
- Converges faster than tanh or sigmoid
- Cheap to compute

Universal approximation theorem

A feed-forward network with a single hidden layer can approximate arbitrary continuous functions (under mild assumptions on the activation function).
A feed-forward network with two hidden layers can approximate arbitrary functions.

But how many neurons do we need? Can we find the weights?

Training

- Training: Minimise cost function $C \Rightarrow$ Learn weights and biases

- Stochastic gradient descent (with batch of size m)

$$w_k \rightarrow w_k - \frac{\eta}{m} \sum_{j=1}^{m} \frac{\partial C_j}{\partial w_k}$$

$$b_l \rightarrow b_l - \frac{\eta}{m} \sum_{j=1}^{m} \frac{\partial C_j}{\partial b_l}$$

Back-propagation

- Efficient calculation of $\frac{\partial C}{\partial w}$ and $\frac{\partial C}{\partial b}$

- Consists of forward pass and backward pass
  - *Forward pass*: Inference, remember weighted inputs and outputs at each layer
  - *Backward pass*: Propagate the error from the output layer to the earlier layers

Cost function

- Regression
  - L2 loss: $C = \sum_i (y_i - \hat{y}_i)^2$
- Classification (with one-hot vector, e. g. $\begin{pmatrix} 0 & 0 & 1 & 0 \end{pmatrix}$)
  - Cross entropy loss: $C = \sum_i -y_i \ln(\hat{y}_i) - (1 - y_i) \ln(1 - \hat{y}_i)$

Weight initialisation

- From pre-trained networks
- Gaussian noise
  - MSRA (for ReLU): $\sigma = \frac{2}{\sqrt{X \cdot Y \cdot (C_{in} + C_{out})}}$
  - Keeps variance of input and output the same

Regularisation

- L2 regularisation
  - Add squared weights as penalty to the cost function

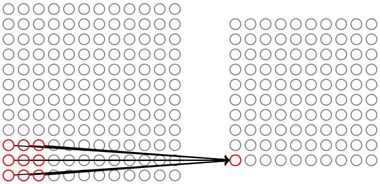
  $\Rightarrow$ Prevents weights from becoming too high
- Dropout
  - Drop nodes randomly

Deep learning - What is it?

- Deep learning $\equiv$ multiple hidden layers
- Became very popular during the last few years
  - CNNs
  - Efficient GPU implementations

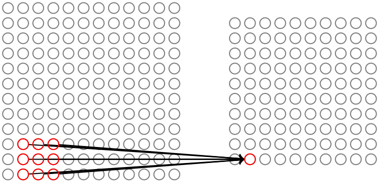
Convolutional Neural Networks

Convolution layer



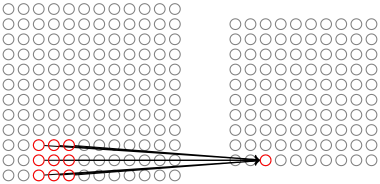
Weights are shared $\Rightarrow$ Prevents the curse of dimensionality

Convolution layer



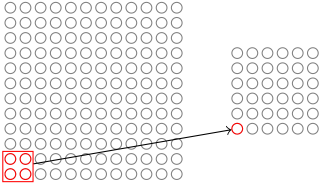
Weights are shared $\Rightarrow$ Prevents the curse of dimensionality

Convolution layer



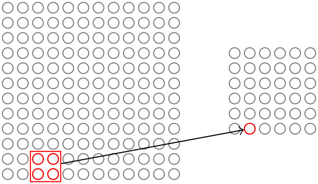
Weights are shared $\Rightarrow$ Prevents the curse of dimensionality

Pooling



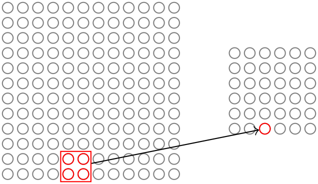
- Reduces resolution
  $\Rightarrow$ computationally cheaper, larger receptive field
- Adds shift invariance

Pooling



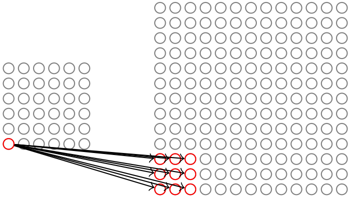
- Reduces resolution
  ⇒ computationally cheaper, larger receptive field
- Adds shift invariance

Pooling



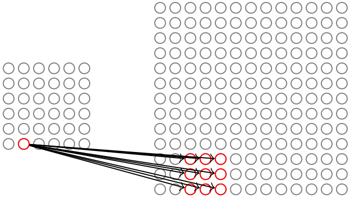
- Reduces resolution
  $\Rightarrow$ computationally cheaper, larger receptive field
- Adds shift invariance

Deconvolution layer



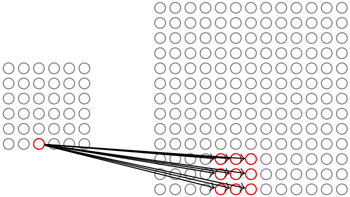
- Transposed convolution with stride > 1
- Used for upsampling

Deconvolution layer



- Transposed convolution with stride > 1
- Used for upsampling

Deconvolution layer



- Transposed convolution with stride > 1
- Used for upsampling

ResNet

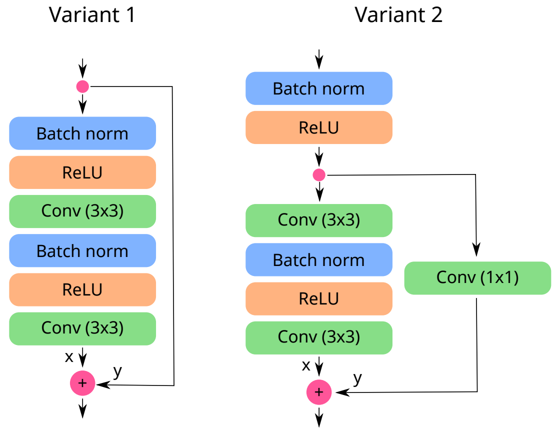
- Residual neural network
- Has skip connections / shortcuts

$\Rightarrow$ Avoids the *vanishing gradient problem*

$\Rightarrow$ Allows to train networks with hundreds to thousands of layers

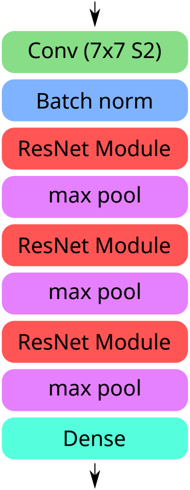
ResNet modules

- Computes $g(x) = x + f(x)$
- Gives the name to the module



Simple ResNet

- *Very* simple ResNet
  - We will implement this in the hands-on tutorial
- Usually, ResNets are much deeper and solve more complicated problems
- But this is fast to train on the CPU...



Tensorflow

Tensorflow - Overview

- Developed by Google
- Published as Open Source in November 2015
- Currently one of the most widely used deep learning frameworks

- Core implemented in C++
- Training code is (usually) written in Python
- Supports CPUs, (Nvidia) GPUS and Google TPUs

Tensorflow - Execution modes

- Eager execution
  - New...
  - Imperative
  - Evaluates operations immediately
  - ⇒ Easy to debug
  - ⇒ Natural control flow

Tensorflow - Execution modes

- Graph Execution
  - Python code defines a graph
  - Later, graph is evaluated repeatedly with different input data
  - ⇒ Loops have to be defined in graph
  - ⇒ Graph can be optimized
  - ⇒ Avoids constant switches between Python and C++
  - ⇒ Faster

Hands-on tutorial

Online resources

- http://neuralnetworksanddeeplearning.com
  Good introduction to neural networks
- https://github.com/ChristosChristofidis/awesome-deep-learning
  Huge collection of links
- https://github.com/kjw0612/awesome-deep-vision
  Collection of recent papers that cover computer vision applications using deep learning
- https://www.tensorflow.org/tutorials/
  Tutorials for Tensorflow

Hands-on tutorial

Hands-on tutorial: Traffic sign classifier with Tensorflow

Thank you for your kind attention!

Questions?