Business Intelligence
SS 2017

Neural Networks and Deep Learning

W. Grossmann

Content

- Introduction
- Backpropagation
- Tuning a network
- Autoencoders
- Deep Learning

Introduction

- The following presentation is based on Chapter 18 in: B. Efron, T. Hastie: Computer Age Statistical Inference, Cambridge University Press 2016
- Neural networks can be used to learn any smooth predictive relationship
- We focus on the use as a classifier, i.e. learning how to predict class membership from a number of attributes

Introduction

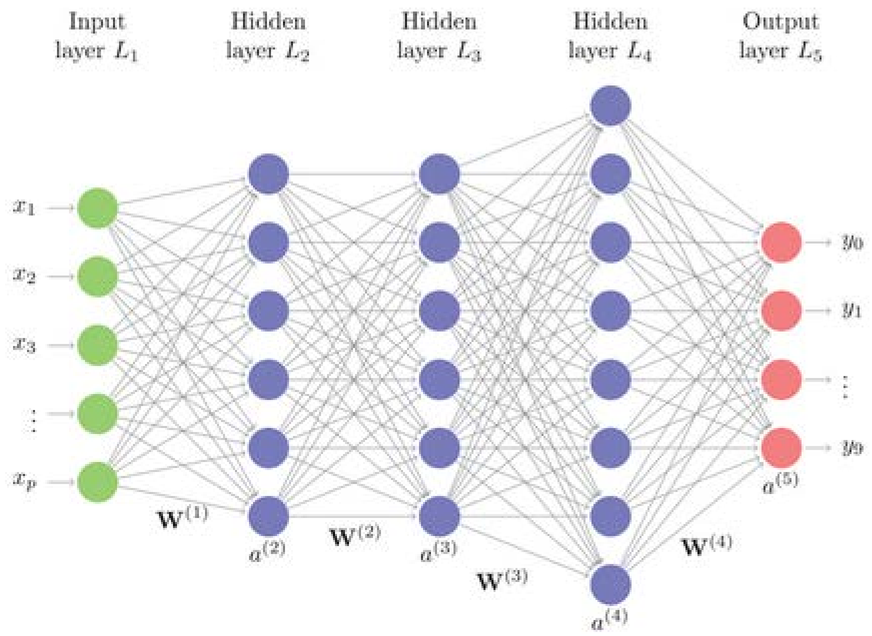
- A classical example of early successful application is the recognition of handwritten digits based on the MNIST dataset:
  - The full dataset consists of 60.000 training images for the digits 0,…,10 and 10.000 test data
  - Each digit is represented as a 28x28 gray-scale image

Introduction

- Examples of digits:

Introduction

- For learning the class membership the following network was used:



Introduction

- A network has
  - An input layer with nodes corresponding to the number of variables, $x_1, x_2, \ldots, x_p$
  - A number of hidden layers $L_1, L_2, \ldots, L_K$ with p_k nodes in layer k; the variables in the nodes are denoted by $a_1^{(k)}, a_2^{(k)}, \ldots, a_{p_k}^{(k)}$
  - An output layer with nodes corresponding to the number of classes
  - For the edges between the nodes variables $w_{lj}^{(k)}$ for weights are defined

Introduction

- The network operates in the following way
  - Propagate the input values through the layer to the output values according to the formula:

$$z_l^{(k)} = w_{\ell 0}^{(k-1)} + \sum_{j=1}^{p_{k-1}} w_{\ell j}^{(k-1)} a_j^{(k-1)}$$

$$a_l^{(k)} = g^{(k)}(z_l^{(k)}) \qquad k = 1, \dots K$$

  - Matrixnotation: $z^{(k)} = W^{(k-1)} a^{(k-1)}$

$$a^{(k)} = g^{(k)}(z^{(k)}) \qquad k = 1, \dots K$$

Introduction

– The functions $g^{(k)}$ are called activation functions
– The activation functions in the last layer are normalized in such a way that the results are probabilities:

$$g^{(K)}(z_m^{(K)}) = \frac{e^{z_m^{(K)}}}{\sum_{\ell=1}^{M} e^{z_\ell^{(K)}}}$$

Backpropagation

- The parameters of the network are the weights $W = (w_{lj}^{(k)})$
- These weights are chosen in such a way that a penalized loss with respect to the prediction is minimized:

$$Loss = \left\{ \frac{1}{n} \sum_{i=1} L[y_i, f(x_i, W)] + \lambda J(W) \right\}$$

$$J(W) = \frac{1}{2} \sum_{k=1}^{K} \sum_{j=1}^{p_k} \sum_{\ell=1}^{p_{k+1}} \left\{ w_{\ell j}^{(k)} \right\}^2$$

Backpropagation

- The standard loss function L is the quadratic loss function
- This minimization is obtained by adaptation of the weights according to gradient descent with respect to the weights
- Calculation of the gradients can be done in a stepwise manner:
  - Starting from the outputs we compute the gradients in each layer and obtain a change direction for the weights in the layers

Backpropagation

– Formula:

$$W^{(k)} = W^{(k)} - \alpha(\Delta W^{(k)} + \lambda W^{(k)})$$

$$\Delta W^{(k)} = \frac{1}{n}\sum_{i=1}^{n} \frac{\partial L[y_i, f(x_i; W)]}{\partial W^{(k)}}$$

– The parameter $\alpha \in (0,1]$ is called learning rate

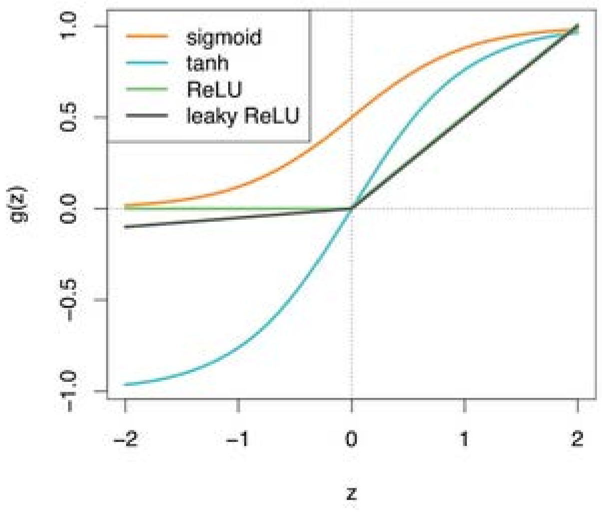
- The performance of the algorithm depends essentially on the choice for the learning rate and on specific ways for calculation of the gradients

Backpropagation

- Further tuning parameters:
  - Number of hidden layers: In general a large number of hidden layers is preferred and the complexity of the model is reduced by choosing a larger weight regularization
  - Using different penalizations: instead of the quadratic penalty one can use the absolute values of the weights

Backpropagation

– Choice of the activation function: The most popular is the sigmoid function; other functions can be used as shown in the figure:

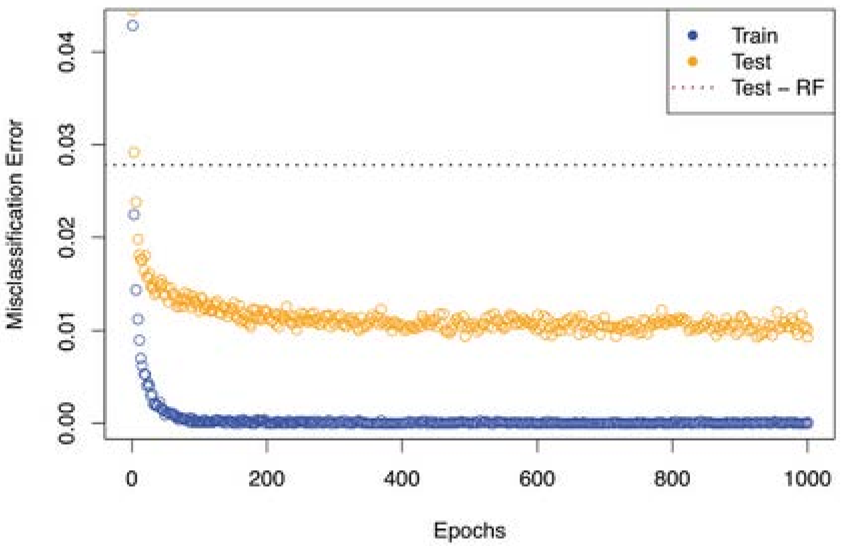


Backpropagation

- With the network shown in the introduction a misclassification rate in the test set of 0,093 was achieved. Misclassified cases:

Backpropagation

- Misclassification of training and test set:



Autoencoders

- Autoencoders are neural networks which compute a type of nonlinear principal component analysis
- Autoencoders need no training data and allow extraction of important features of the variables
- Given p vectors of inputs $x_1, x_2, \ldots, x_p$ a single layer auto-encoder finds a pxq weight matrix W which minimizes

$$\sum_{i=1}^{n} \left\| x_i - W' g(W x_i) \right\|^2$$

Deep Learning

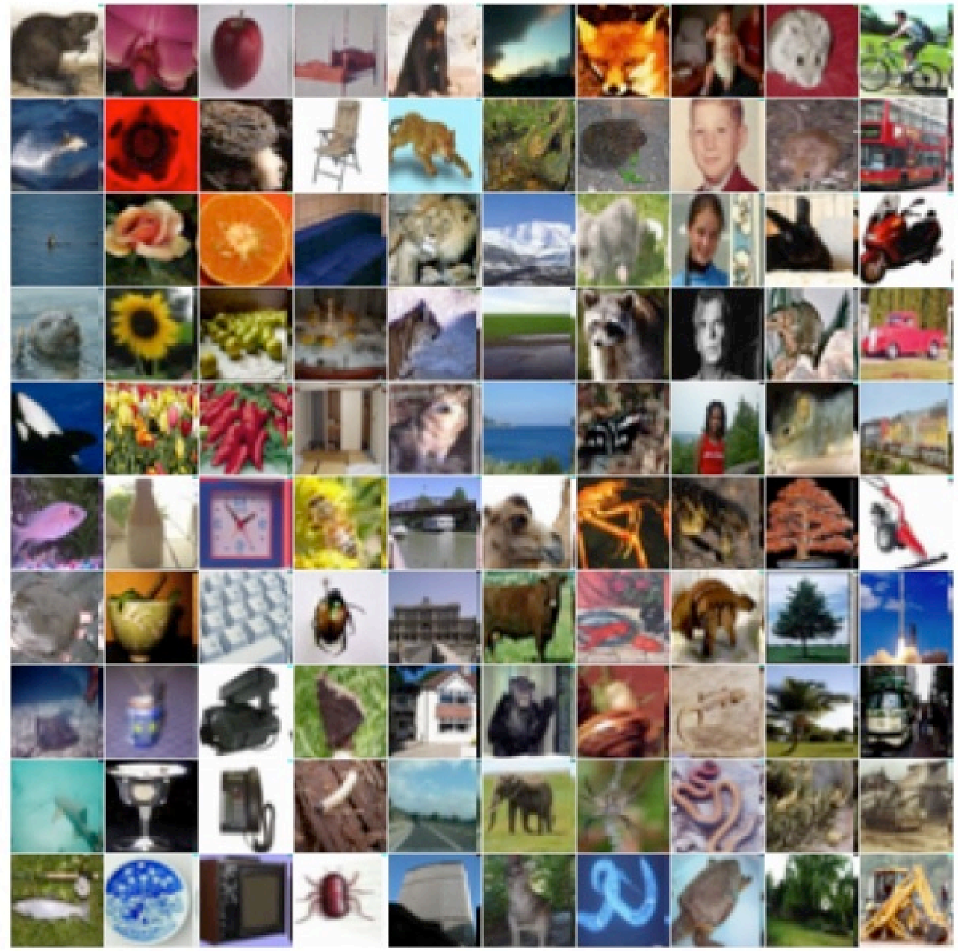
- Deep learning compute models that are composed of multiple processing layers for learning representations of data with multiple abstraction levels
- Examples:
  - Images can be viewed at different levels of pixel resolution and different colors (RGB)
  - Audio signals with multiple structures
  - 2d images for audio spectrograms

Deep Learning

- A basic model is the convolution network which is used for data forming multiple arrays
- There are four key ideas behind convolution networks :
  - Local connections: Information in local variables is often highly correlated
  - Shared weights: information in different parts of the signal show similar structures, hence they are processed in a similar way
  - Pooling: Condensing the information in local patches to a single value
  - Using many layers: Different layers allow the analysis of the data at different levels of abstraction

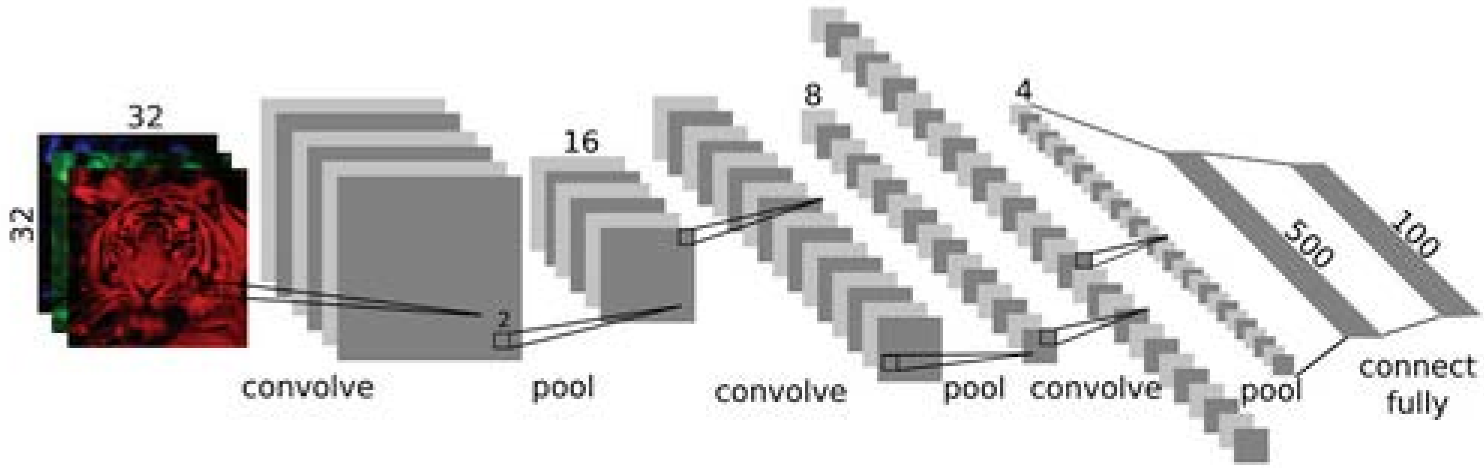
Deep Learning

- Example: The CIFAR-100 data: 32x32x3 natural images
- 600 examples, 100 classes, hierarchically organized: 20 coarse classes, with 5 subclasses



Deep Learning

- Architecture of a convolution network for analysis



Deep Learning

- Configuration of the Deep learning network for CIFAR-100

Algorithm 18.2 CONFIGURATION PARAMETERS FOR DEEP-LEARNING NETWORK USED ON THE CIFAR-100 DATA.

Layer 1: 100 convolution maps each with $2 \times 2 \times 3$ kernel (the 3 for three colors). The input image is padded from 32×32 to 40×40 to accommodate input distortions.

Layers 2 and 3: 100 convolution maps each $2 \times 2 \times 100$. Compositions of convolutions are roughly equivalent to convolutions with a bigger bandwidth, and the smaller ones have fewer parameters.

Layer 4: Max pool 2×2 layer, pooling nonoverlapping 2×2 blocks of pixels, and hence reducing the images to size 20×20.

Layer 5: 300 convolution maps each $2 \times 2 \times 100$, with dropout learning with rate $\phi_5 = 0.05$.

Layer 6: Repeat of Layer 5.

Layer 7: Max pool 2×2 layer (down to 10×10 images).

Layer 8: 600 convolution maps each $2 \times 2 \times 300$, with dropout rate $\phi_8 = 0.10$.

Layer 9: 800 convolution maps each $2 \times 2 \times 600$, with dropout rate $\phi_9 = 0.10$.

Layer 10: Max pool 2×2 layer (down to 5×5 images).

Layer 11: 1600 convolution maps, each $1 \times 1 \times 800$. This is a pixelwise weighted sum across the 800 images from the previous layer.

Layer 12: 2000 fully connected units, with dropout rate $\phi_{12} = 0.25$.

Layer 13: Final 100 output units, with softmax activation, and dropout rate $\phi_{13} = 0.5$.