Pattern Recognition

Part 10: (Artificial) Neural Networks

Gerhard Schmidt

Christian-Albrechts-Universität zu Kiel
Faculty of Engineering
Institute of Electrical and Information Engineering
Digital Signal Processing and System Theory
Contents

- Motivation and literature
- Structure of a (basic) neural network
- Applications of neural networks
- Types of neural networks
- Basic training of neural networks
- Example applications
Contents

- Motivation and literature
  - Neural networks
  - Deep learning
  - Literature

- Structure of a (basic) neural network
- Applications of neural networks
- Types of neural networks
- Basic training of neural networks
- Example applications
Motivation and Literature

**Neural networks:**

- Neural networks are a very popular machine learning technique.
- They simulate the mechanisms of learning in biological systems such as the human brain.
- The human brain/the nervous system contains cells which are called neurons. The neurons are connected using axons and dendrites. While learning the connections between neurons are changed.
- Within this lecture we will talk about artificial neural networks that mimic the processes in the human brain. The adjective “artificial” will be omitted for reasons of brevity.

Neural Networks

Motivation and Literature

**Deep learning:**

- The advantage of neuronal structures is their ability to be adapted to several types of problems by changing their size and internal structure.

- A few years ago so-called deep approaches appeared. This was one of the main factors for the success of neural networks.

- “Deep” means here to have on the one hand several/many hidden layers. On the other hand it means that specific training procedures are used.

- Compared to conventional (shallow) structures deep approaches are specially suited if a large amount of training data is available.
Neural Networks

Motivation and Literature

**Literature:**

- C. C. Aggarwal: *Neural Networks and Deep Learning*, Springer, 2018
- A. Géron: *Machine Learning mit Scikit-Learn & Tensorflow*, O’Reilly, 2018 (in German and English)
- I. Goodfellow, Y. Bengio, A. Courville: *Deep Learning*, Mitp, 2018 (in German and English)
Neural Networks

Contents

- Motivation
- Structure of a (basic) neural network
- Applications of neural networks
- Types of neural networks
- Basic training of neural networks
- Example applications
**Neural Networks**

**Structure of a Neural Network – Basics**

**Basic structure during runtime and training:**

1. **Training**
   - Database with input features
   - Database with input and output features
   - Neural network
   - Training algorithm
   - Output features: \( \hat{y}(n) \)
   - Error computation: \( e(n) \)

2. **Runtime**
   - Input features: \( x(n) \)
   - Output features: \( y(n) \)
Structure of a Neural Network – Basics

**Network structure:**

1. Database with input features
2. Database with input and output features
3. Input layer
4. Hidden layer
5. Hidden layer
6. Output layer
7. Training algorithm
8. Neural network
9. Distance comp.
10. $\hat{y}(n)$
11. $y(n)$
12. $x(n)$

Diagram:

- Input layer to Hidden layer to Hidden layer to Output layer
- $x(n)$ flows through the network
- $\hat{y}(n)$ is the output
Neural Networks

Structure of a Neural Network – Basics

**Input layer:**

- Sometimes only a “pass through” layer
  \[ h_0(n) = x(n). \]

- Sometimes also a *mean compensation* and a *normalization* is performed:
  \[ h_{0,i}(n) = \frac{x_i(n) - \mu_{x_i}}{\sigma_{x_i}}. \]

Afterwards all individually normalized inputs are **combined to a vector**:

\[ h_0(n) = [h_{0,0}(n), \ldots, h_{0,N_0-1}(n)]^T \]
Structure of a Neural Network – Basics

**Hidden layer:**

- **Linear weighting** of inputs with bias

\[ x_{m,i}(n) = w_{m,i}^T h_m(n) + b_{m,i} \]

with

\[ w_{m,i} = [w_{m,0}, \ldots, w_{m,N_{m-1}-1}]^T. \]

- **Nonlinear activation function:**

\[ y_{m,i}(n) = f_{\text{act},m}(x_{m,i}(n)). \]

- **Combination** of all results to a vector:

\[ h_{m+1}(n) = [y_{m,0}(n), \ldots, y_{m,N_{m-1}}(n)]^T. \]
Neural Networks

Structure of a Neural Network – Basics

**Activation functions – part 1:**

- The sum of the weighted inputs plus the bias will be **abbreviated** with
  \[ x(n) = w^T h(n) + b. \]

- Several **activation functions** exist, such as
  - the **identity** function
    \[ f_{\text{act}}(x(n)) = x(n), \]
  - the **sign** function, or
    \[ f_{\text{act}}(x(n)) = \text{sign}(x(n)), \]
  - the **sigmoid** function
    \[ f_{\text{act}}(x(n)) = \frac{1}{1 + e^{-x(n)}}. \]
Neural Networks

Structure of a Neural Network – Basics

Activation functions – part 2:

- Further activation functions:
  - the \textit{tanh} function
    \[ f_{\text{act}}(x(n)) = \frac{e^{2x(n)} - 1}{e^{2x(n)} + 1}, \]
  - the \textit{rectified linear} function (or unit, ReLU)
    \[ f_{\text{act}}(x(n)) = \max\{0, x(n)\}, \]
  - the “\textit{hard tanh}” function
    \[ f_{\text{act}}(x(n)) = \max\{\min\{1, x(n)\}, -1\}. \]
Neural Networks

Structure of a Neural Network – Basics

**Output layer:**

- Sometimes only a **“pass through” layer**
  \[
  \hat{y}(n) = h_M(n).
  \]

- Sometimes also a **limitation**
  \[
  \hat{y}_{\text{lim},i}(n) = \max \left\{ \hat{y}_{\text{min},i}, \min \{ \hat{y}_{\text{max},i}, h_{M,i}(n) \} \right\}
  \]

  and a **normalization** is performed:
  \[
  \hat{y}_i(n) = \frac{\hat{y}_{\text{lim},i}(n)}{\sum_{i=0}^{N_{M-1}} \hat{y}_{\text{lim},i}(n)}.
  \]

  The limited and normalized outputs are **combined to a vector**
  \[
  \hat{y}(n) = [\hat{y}_0(n), ..., \hat{y}_{N_{M-1}}(n)]^T.
  \]
Neural Networks

Structure of a Neural Network – Basics

Layer sizes:

- The **input and the output layer size** is usually given by the application. The input layer size is equal to the feature vector size and the output layer size is determined by the amount of output features.

  Sometimes *more outputs than required* are computed in order to modify the cost function.

- The entire **size of the network** (sum of all layer sizes) should be adjusted to the **size of the available data**.

- In some applications so-called *bottle neck layers* are helpful.
Neural Networks

Contents

- Motivation
- Structure of a (basic) neural network
- Applications of neural networks
  - Real-time video object recognition
  - Improving Image Resolution
  - Automatic image colorization
- Types of neural networks
- Basic training of neural networks
- Example applications
Neural Networks

Applications of Neural Networks – Sources

**Tesla:**
- https://vimeo.com/272696002?cjevent=c27333cefa3511e883d900650a18050f

**Pixel Recursive Super Resolution:**

**Image colorization:**
Video object recognition for Tesla cars:

- Tesla uses cameras, radar and ultrasonic sensors to detect objects in the surrounding area. However, they rely mostly on computer vision by cameras.

- Their current system uses (mostly) a so-called convolutional network (details later on) for object recognition. New approaches use “CodeGen” (also the structure [not only the weights] of the network are adapted during the training).

- The main system for autonomous driving is a deep neural network.

The following video is a full self driving demo by Tesla, where this legend is used:
Applications of Neural Networks – Real-time Video Object Recognition
Applications of Neural Networks – Improving Image Resolution

“Super resolution is the problem of artificially enlarging a low resolution photograph to recover a plausible high resolution. [...]”

Neural network types used:

- New probabilistic deep network architectures are used that are based on log-likelihood objectives.
- Extension of “PixelCNNs” (conv. net.) and “ResNet” (residual net.)
- Basically two networks are used:
  - A “prior network” that captures serial dependencies of pixels (auto-regressive part of model) [PixelCNN] and
  - a “conditioning network” that captures the global structure of images (DCNN, similar to “SRResNet”, feed-forward convolutional neural networks).

Problems:

- As magnification increases the neural network needs to predict missing information such as:
  - complex variations of objects, viewpoints, illumination, ...
  - Underspecified problem → many plausible high resolution images
Applications of Neural Networks – Automatic Image Colorization with Simultaneous Classification

Coloration of greyscale images:

- A convolutional network using low-level features to compute global features for classifying the image (rough type of image, what are the surroundings).

- A parallel network uses the same low-level features to compute mid-level features.

- Fusion of global features (e.g. indoor or outdoor photo) and mid-level features are used for colorization of the image.

- Greyscale image is then used for luminance.

Figure 2: Overview of our model for automatic colorization of greyscale images.
Applications of Neural Networks – Automatic Image Colorization with Simultaneous Classification

Other examples:

(a) Cranberry Picking, Sep. 1911  
(b) Burns Basement, May 1910  
(c) Miner, Sep. 1937  
(d) Scott’s Run, Mar. 1937

Typical failure cases:

Input  
Ground truth  
Proposed
Neural Networks

Contents

- Motivation
- Structure of a (basic) neural network
- Applications of neural networks
- Types of neural networks
  - Convolutional neural networks
  - Recurrent neural networks
- Basic training of neural networks
- Example application
Neural Networks

Types of Neural Networks

Convolutional neural networks (CNNs):

- CNNs were part of the *early times in deep approaches*.
- They are often applied in *image* and *video applications*.
- Often *three-dimensional layers* with special *ReLU activation functions* followed by *pooling* (next slide) are used.
- The weights of the layers are used as in a “conventional” convolution, meaning that the *same weights* are used very often (e.g. for edge detection).

Source: Adopted from Charu C Aggarwal, *Neural Networks and Deep Learning*, Springer, 2018
Convolutional neural networks (CNNs):

- Pooling can be realized e.g. by computing the maximum over an overlapping and moving part of the input:

  \[ h_{i,u,v}(n) = f_{\text{pool}}(X_{i-1}(n)) \]
  \[ = \max_{l \in \{-N,N\}} \left\{ \max_{k \in \{-N,N\}} \left\{ x_{i-1,u+l,v+k}(n) \right\} \right\} \]

- The basic idea behind pooling is that it is important that a specific pattern is found in a certain area, but it’s not important where exactly.
Neural Networks

Types of Neural Networks

Recurrent neural networks (RNNs):

- **Recursive branches** are added to the network to allow for efficient modelling of temporal memory.
- **Stability** (during operation) is not really an issue (in contrast to IIR filters), since usually the activation functions include limitations.
- Very often the **delay element is not depicted** in literature of RNNs.
Neural Networks

Types of Neural Networks

**Recurrent neural networks (RNNs):**

- **Training** could be done easily if the network is unfolded.

- Afterwards again a “standard” network with extended in- and outputs as well as with coefficient limitations can be trained.
Neural Networks

Contents

- Motivation
- Structure of a (basic) neural network
- Applications of neural networks
- Types of neural networks
- Basic training of neural networks
  - Backpropagation
  - Generative adversarial networks
- Examples application
Neural Networks

Training of Neural Networks – Basics

**Preliminary items – part 1:**

- In order to be mathematically correct, *several indices* are necessary:
  - *Time* or frame index $n$.
  - *Layer index* $m$.
  - *Parameter index* $i$.
  - *Training index* $p$.

- However, some of the indices will be *dropped* in the following slides for the reason of *better readability*. 
Training of Neural Networks – Basics

**Preliminary items – part 2:**

- For a simpler description, **extended parameter vectors** and **extended signal vectors** will be used in the following:

\[
\tilde{h}_m(n) = \begin{bmatrix} h_m^T(n), 1 \end{bmatrix}^T,
\]
\[
\tilde{w}_{m,i}(n) = \begin{bmatrix} w_{m,i}^T(n), b_{m,i} \end{bmatrix}^T.
\]

- The input of the activation function will be denoted with

\[
x_{m,i}(n) = w_{m,i}^T h_m(n) + b_{m,i} = \tilde{w}_{m,i}^T \tilde{h}_m(n).
\]
Neural Networks

Training of Neural Networks – Back Propagation

**Back-propagation algorithm:**

- A popular training algorithm for neural networks is the so-called *back-propagation algorithm*.
- The algorithm is minimizing a cost function by means of *gradient descent* steps.
- The *chain rule* in differentiation plays an important role and it is necessary that the *activation functions* are *continuous* and differentiable.
- While the network is computed during run-time from the input layer to the output layer, the back-propagation algorithm works *from the output layer to the input* one.
Neural Networks

Training of Neural Networks – Back Propagation

Cost function:

- A basic goal of the network might be to minimize the average norm of the difference between the desired and the estimated feature vectors:

\[ C = \sum_{n=0}^{N-1} \| y(n) - \hat{y}(n) \|^2 \quad \rightarrow \min. \]

- In order to achieve this goal all parameters of the neural network are corrected in negative gradient direction (method of steepest descent):

\[ -\nabla_{\tilde{w}_{m,i}} C = -\frac{\partial C}{\partial \tilde{w}_{m,i}}. \]

Training of Neural Networks – Back Propagation

**Back-propagation algorithm:**

- The *cost function* is “refined” as follows:
  \[ C^{(p)} = \sum_{n=0}^{N-1} \left\| y(n) - \hat{y}^{(p)}(n) \right\|_2^2 = \sum_{n=0}^{N-1} e^{(p)}(n) \rightarrow \min. \]

- The *gradient* of the cost function consists of several *partial differentiations*:
  \[ \nabla_{\tilde{w}^{(p)}_{m,i}} C^{(p)} = \left[ \frac{\partial C^{(p)}}{\partial \tilde{w}^{(p)}_{m,i,0}}, \frac{\partial C^{(p)}}{\partial \tilde{w}^{(p)}_{m,i,1}}, \frac{\partial C^{(p)}}{\partial \tilde{w}^{(p)}_{m,i,2}}, \ldots \right]^T. \]

- The *parameters are updated* during the training process according to:
  \[ \tilde{w}^{(p+1)}_{m,i} = \tilde{w}^{(p)}_{m,i} - \frac{\alpha}{2} \frac{\partial C^{(p)}}{\partial \tilde{w}^{(p)}_{m,i}}. \]

Training index

Step-size parameter
Neural Networks

Training of Neural Networks – Back Propagation

**Back-propagation algorithm:**

- We will focus now on a *single differentiation* (with respect to only one parameter). Here, we *insert the details of the cost function* and we *omit the training index for better readability*:

\[
\frac{\partial C}{\partial \tilde{w}_{m,i,j}} = \frac{\partial}{\partial \tilde{w}_{m,i,j}} \sum_{n=0}^{N-1} e(n) = \sum_{n=0}^{N-1} \frac{\partial e(n)}{\partial \tilde{w}_{m,i,j}}.
\]

- Keep the structure of the individual neurons in mind ....

\[
\tilde{h}_m(n) \xrightarrow{\times} x_{m,i}(n) \xrightarrow{\text{activation function}} \tilde{h}_{m+1,i}(n) = f_{\text{act},m}(\tilde{w}_{m,i}^T \tilde{h}_m(n)) = f_{\text{act},m}(x_{m,i}(n)).
\]
Neural Networks

Training of Neural Networks – Back Propagation

**Back-propagation algorithm:**

- First, we will compute the update of the weights in the output layer \( (m = M) \):

\[
\frac{\partial C}{\partial \tilde{w}_{M,i,j}} = \frac{\partial}{\partial \tilde{w}_{M,i,j}} \sum_{n=0}^{N-1} e(n) = \sum_{n=0}^{N-1} \frac{\partial e(n)}{\partial \tilde{w}_{M,i,j}}.
\]

- All individual gradients (individual for all input frames \( n \)) can be summed and then an update is performed or an update can be performed after each gradient computation. For reasons of brevity we will compute now only individual gradients. In order to compute the gradient, we *split the global gradient into a product of two simpler gradients*:

\[
\frac{\partial e(n)}{\partial \tilde{w}_{M,i,j}} = \frac{\partial e(n)}{\partial x_{M,i}(n)} \frac{\partial x_{M,i}(n)}{\partial \tilde{w}_{M,i,j}}.
\]

- This “trick” will be repeated but now for the multivariate case to compute the gradients for the weights of the hidden layers:

\[
\frac{\partial e(n)}{\partial \tilde{w}_{M-1,i,j}} = \sum_{k} \frac{\partial e(n)}{\partial x_{M,k}(n)} \frac{\partial x_{M,k}(n)}{\partial x_{M-1,i}(n)} \frac{\partial x_{M-1,i}(n)}{\partial \tilde{w}_{M-1,i,j}}.
\]
**Back-propagation algorithm:**

- Let’s start now with the gradient for the weights of the output layer:

\[
\frac{\partial e(n)}{\partial \tilde{w}_{M,i,j}} = \frac{\partial e(n)}{\partial x_{M,i}(n)} \frac{\partial x_{M,i}(n)}{\partial \tilde{w}_{M,i,j}}.
\]

\[
\frac{\partial x_{M,i}(n)}{\partial \tilde{w}_{M,i,j}} = \frac{\partial}{\partial \tilde{w}_{M,i,j}} \sum_d \tilde{w}_{M,i,d} \tilde{h}_{M,d}(n) = \tilde{h}_{M,j}(n),
\]

\[
\frac{\partial e(n)}{\partial x_{M,i}} = \frac{\partial}{\partial x_{M,i}} \sum_d (y_d(n) - \hat{y}_d(n))^2
\]

\[
= -2 \sum_d (y_d(n) - \hat{y}_d(n)) \frac{\partial \hat{y}_d(n)}{\partial x_{M,i}(n)}
\]

\[
= -2 \left(y_i(n) - \hat{y}_i(n)\right) f'_{\text{act},M}(x_{M,i}(n)) \tilde{h}_{M,j}(n)
\]
Back-propagation algorithm:

- For the second last layer we can do the same for the first and the last term:

\[
\frac{\partial e(n)}{\partial \tilde{w}_{M-1,i,j}} = \sum_k \frac{\partial e(n)}{\partial x_{M,k}(n)} \frac{\partial x_{M,k}(n)}{\partial x_{M-1,i}(n)} \frac{\partial x_{M-1,i}(n)}{\partial \tilde{w}_{M-1,i,j}}.
\]

- Now only the center term is missing:

\[
\frac{\partial x_{M,k}(n)}{\partial x_{M-1,i}(n)} = \ldots
\]
Neural Networks

Training of Neural Networks – Back Propagation

**Back-propagation algorithm:**

- The *missing term*:

\[
\frac{\partial x_{M,k}(n)}{\partial x_{M-1,i}(n)} = \frac{\partial}{\partial x_{M-1,i}(n)} \sum_d \tilde{h}_{M,d}(n) \tilde{w}_{M,k,d}
\]

\[
= \frac{\partial}{\partial x_{M-1,i}(n)} \sum_d f_{act,M-1}(x_{M-1,d}(n)) \tilde{w}_{M,k,d}
\]

\[
= f'_{act,M-1}(x_{M-1,i}(n)) \tilde{w}_{M,k,i}
\]

- Putting *everything together* leads to:

\[
\frac{\partial e(n)}{\partial \tilde{w}_{M-1,i,j}} = -2 \tilde{h}_{M-1,j}(n) \sum_k (y_k(n) - \hat{y}_k(n)) f'_{act,M}(x_{M,k}(n)) f'_{act,M-1}(x_{M-1,i}(n)) \tilde{w}_{M,k,i}
\]
**Training of Neural Networks – Back Propagation**

*Back-propagation algorithm:*

- Two more layers to see the structure:

\[
\frac{\partial e(n)}{\partial \tilde{w}_{M,i,j}} = \frac{\partial e(n)}{\partial x_{M,i}(n)} \frac{\partial x_{M,i}(n)}{\partial \tilde{w}_{M,i,j}},
\]

\[
\frac{\partial e(n)}{\partial \tilde{w}_{M-1,i,j}} = \sum_k \frac{\partial e(n)}{\partial x_{M,k}(n)} \frac{\partial x_{M,k}(n)}{\partial x_{M-1,i}(n)} \frac{\partial x_{M-1,i}(n)}{\partial \tilde{w}_{M-1,i,j}},
\]

\[
\frac{\partial e(n)}{\partial \tilde{w}_{M-2,i,j}} = \sum_k \frac{\partial e(n)}{\partial x_{M,k}(n)} \sum_\ell \frac{\partial x_{M,k}(n)}{\partial x_{M-1,\ell}(n)} \frac{\partial x_{M-1,\ell}(n)}{\partial x_{M-2,i}(n)} \frac{\partial x_{M-2,i}(n)}{\partial \tilde{w}_{M-2,i,j}},
\]

\[
\frac{\partial e(n)}{\partial \tilde{w}_{M-3,i,j}} = \sum_k \frac{\partial e(n)}{\partial x_{M,k}(n)} \sum_\ell \frac{\partial x_{M,k}(n)}{\partial x_{M-1,\ell}(n)} \sum_m \frac{\partial x_{M-1,\ell}(n)}{\partial x_{M-2,m}(n)} \frac{\partial x_{M-2,m}(n)}{\partial x_{M-3,i}(n)} \frac{\partial x_{M-3,i}(n)}{\partial \tilde{w}_{M-3,i,j}}.
\]
Training of Neural Networks – Back Propagation

**Back-propagation algorithm:**

- Interesting is, that the individual differentiations can be **computed recursively**. Let’s have a first look on the results (the third last layer was not derived before, but it’s straight forward). Let’s **start with the last layer**:

  \[
  \frac{\partial e(n)}{\partial \tilde{w}_{M,i,j}} = -2 \left( y_i(n) - \hat{y}_i(n) \right) f'_{\text{act},M} \left( x_{M,i}(n) \right) \tilde{h}_{M,j}(n)
  \]

- Here we introduce the following **“helping” variables**:

  \[
  \delta_{M,i}(n) = \left( y_i(n) - \hat{y}_i(n) \right) f'_{\text{act},M} \left( x_{M,i}(n) \right).
  \]

- To be a bit more precise, we **add also the iteration index**:

  \[
  \delta^{(p)}_{M,i}(n) = \left( y_i(n) - \hat{y}_i^{(p)}(n) \right) f'_{\text{act},M} \left( x_{M,i}^{(p)}(n) \right).
  \]

- Now the **update of the parameters of the last layer** (change in negative gradient direction) can be written as

  \[
  \tilde{w}_{M,i,j}^{(p+1)} = \tilde{w}_{M,i,j}^{(p)} + \alpha \sum_{n=0}^{N-1} \delta_{M,i}(n) \tilde{h}_{M,j}(n).
  \]
Neural Networks

Training of Neural Networks – Back Propagation

**Back-propagation algorithm:**

- Visualization – *last layer*:

  Compute in forward direction $\tilde{h}^{(p)}_{M,i}(n)$ and $x^{(p)}_{M,i}(n)$!

  Initialize helping variables in backward direction

  \[
  \delta^{(p)}_{M,i}(n) = (y_i(n) - \tilde{y}_i^{(p)}(n)) f'_{act,M}(x^{(p)}_{M,i}(n))
  \]

  and update the parameter of the last layer

  \[
  \bar{w}^{(p+1)}_{M,i,j} = \bar{w}^{(p)}_{M,i,j} + \alpha \sum_{n=0}^{N-1} \delta^{(p)}_{M,i}(n) \tilde{h}^{(p)}_{M,j}(n).
  \]
Training of Neural Networks – Back Propagation

**Back-propagation algorithm:**

- **Now the second last layer:**

\[
\frac{\partial e(n)}{\partial \tilde{w}_{M-1,i,j}} = -2 \tilde{h}_{M-1,i,j}(n) \sum_k \left( y_k(n) - \hat{y}_k(n) \right) f'_{act,M}(x_{M,k}(n)) f'_{act,M-1}(x_{M-1,i}(n)) \tilde{w}_{M,k,i}.
\]

- **Here we can insert the “helping” variables from the last layer:**

\[
\frac{\partial e(n)}{\partial \tilde{w}_{M-1,i,j}} = -2 \tilde{h}_{M-1,i,j}(n) \sum_k \left( y_k(n) - \hat{y}_k(n) \right) f'_{act,M}(x_{M,k}(n)) \frac{f'_{act,M-1}(x_{M-1,i}(n)) \tilde{w}_{M,k,i}}{\delta_{M,k}(n)}.
\]

\[
= -2 \tilde{h}_{M-1,i,j}(n) \sum_k \delta_{M,k}(n) f'_{act,M-1}(x_{M-1,i}(n)) \tilde{w}_{M,k,i}.
\]
Training of Neural Networks – Back Propagation

**Back-propagation algorithm:**

- Result of last slide:
  \[
  \frac{\partial e(n)}{\partial \tilde{w}_{M-1,i,j}} = -2 \tilde{h}_{M-1,j}(n) \sum_k \delta_{M,k}(n) f'_{\text{act},M-1}(x_{M-1,i}(n)) \tilde{w}_{M,k,i}. 
  \]

- Again, this could be separated in two steps. First a *helping variable* is *updated* (again, now with the training index):
  \[
  \delta_{M-1,i}^{(p)}(n) = f'_{\text{act},M-1}(x_{M-1,i}^{(p)}(n)) \sum_k \delta_{M,k}^{(p)}(n) \tilde{w}_{M,k,i}^{(p)}. 
  \]

- Now, the *update of the parameters of the second last layer* can be performed according to
  \[
  \tilde{w}_{M-1,i,j}^{(p+1)} = \tilde{w}_{M-1,i,j}^{(p)} + \alpha \sum_{n=0}^{N-1} \delta_{M-1,i}^{(p)}(n) \tilde{h}_{M-1,j}^{(p)}(n). 
  \]
Neural Networks

Training of Neural Networks – Back Propagation

**Back-propagation algorithm:**

- **Visualization – second last layer:**

  > Compute in forward direction: $\tilde{h}_{M-1,i}^{(p)}(n)$ and $x_{M-1,i}^{(p)}(n)$

  > Update helping variables in backward direction

  \[
  \delta_{M-1,i}^{(p)}(n) = f'_{\text{act},M-1}(x_{M-1,i}^{(p)}(n)) \sum \delta_{M,k}^{(p)}(n) \tilde{w}_{M,k,i}^{(p)}
  \]

  and update the parameter of the second last layer

  \[
  \tilde{w}_{M-1,i,j}^{(p+1)} = \tilde{w}_{M-1,i,j}^{(p)} + \alpha \sum_{n=0}^{N-1} \delta_{M-1,i,j}^{(p)}(n) \tilde{h}_{M-1,j}^{(p)}(n).
  \]
Neural Networks

Training of Neural Networks – Back Propagation

**Back-propagation algorithm:**

- This goes on until the first layer is reached. First an update of the helping variables:

\[
\delta_{m-1,i}(n) = f_{act,m-1}'(x_{m-1,i}(n)) \sum_{k} \delta_{m,k}(n) \tilde{w}_{m,k,i}^{(p)}.
\]

- And then an update of the network parameters:

\[
\tilde{w}_{m-1,i,j}^{(p+1)} = \tilde{w}_{m-1,i,j}^{(p)} + \alpha \sum_{n=0}^{N-1} \delta_{m-1,i}(n) \tilde{h}_{m-1,j}^{(p)}(n).
\]

- As in the case of codebooks, GMMs, HMMs it is checked by using test and validation data, if the cost function does increase. In that case the **training is stopped**. Furthermore, several **variants of this basic update strategies** have been published. Details can be found in the references.
Neural Networks

Contents

- Motivation
- Structure of a (basic) neural network
- Applications of neural networks
- Types of neural networks
- Basic training of neural networks
  - Backpropagation
  - Generative adversarial networks
- Examples application
Neural Networks

Training of Neural Networks – Generative Adversarial Networks

**Basics of generative adversarial networks (GANs):**

- GANs are **not a new network type**, it’s more a special way of training.
- During **runtime** a single “standard” neural network is used. This network is called the **generator network**.
- During **training** a second network is additionally used, called the **discriminator network**.
- The job of the second network is to **estimate**, whether the input (of the decision network) stems from **true (desired) data** or is the output of the **generator network**.
- During the training the generator and the discriminator network are **trained in an alternating fashion**.
Motivation of GANs:

- Example from *image-to-image translations* (creation of realistically looking images from label maps).
- GANs are good candidates if *smoothed results are undesired*.
- *Conditional GANs* were compared to conventionally trained networks.
- *Cost function* is not the mean squared error (or variants of it) any more.

Neural Networks

Training of Neural Networks – Generative Adversarial Networks

Structure of the training procedure:

- Training of the generator network:
  - The discriminator network is kept fixed.
  - A weighted sum of the average norm of the error of the generator network
    \[ \|e_y(n)\| = \|y(n) - \hat{y}(n)\|^2 \]
    and the inverse of the average classification error is
    \[ \frac{1}{e_d^2(n)} = \frac{1}{(d(n) - \hat{d}(n))^2} \]
    minimized (as one variant).
Neural Networks

Training of Neural Networks – Generative Adversarial Networks

*Structure of the training procedure:*

- Training of the *discriminator network:*
  - The *generator network* is kept *fixed.*
  - The average *power of the error* $e_d(n) = d(n) - \hat{d}(n)$ (as one variant) of the discriminator network is *minimized.*
Bandwidth extension:

- For bandwidth extension, GANs are also an interesting alternative (especially conditional GANs).
- The spectral envelope is estimated using GANs, the excitation signal is created by spectral repetition of the narrowband excitation signal.

Neural Networks

Contents

- Motivation
- Structure of a (basic) neural network
- Applications of neural networks
- Types of neural networks
- Basic training of neural networks
- Example application
Matlab example on (handwritten) digit recognition:

- Preprocessing and training for digit recognition in Matlab.
Neural Networks

Summary and Outlook

**Summary:**
- Motivation
- Structure of a (basic) neural network
- Applications of neural networks
- Types of neural networks
- Basic training of neural networks
- Example application

**Next week:**
- Your talks