Artificial Intelligence

11. Artificial Neural Networks

Bojana Dalbelo Bašić, Marko Čupić, Jan Šnajder

University of Zagreb
Faculty of Electrical Engineering and Computing

Acc. year. 2019./2020.

v3.10

Motivation

- Automated information processing is mostly performed by digital computers.
- Still, there is an abundance of information that are not processed automatically, but rather by nervous systems of living organisms!
- The development of a branch in computing is motivated by considering the most common way of information processing in nature.
- We are looking for a different concept of data processing which would better reflect the way a biological brain works.
- An A.I. system successfully replicating the workings of the brain might be intelligent.

Contents

Contents

Motivations for the development of neural-computing

- We know that the brain consists of a large number of neurons that work in parallel.
- We also know the following about it:
  - There are more than 100 different types of biological neurons
  - Each type performs a very simple kind of processing
  - Processing type for a neuron is ≈ 2 milliseconds
  - The number of neurons in a human brain is 10^{11}
  - On average, each neuron receives information from 10^3 do 10^4 other neurons
  - Information is being processed both sequentially and in parallel
  - Information is analog
  - The processing is fault tolerant

Motivations for the development of neural-computing

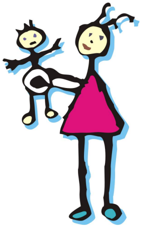
- We would like to build a computer system that would process information in the same way!
- A new paradigm:
  artificial neural networks
- A branch of computing concerned with this type of information processing: neural-computing
  - It is one of the branches of a larger computing subearea called soft-computing

Development directions for artificial intelligence

- Since the first days of AI (early '50s) there have been two main approaches to developing intelligent systems:
  - The symbolic approach: domain knowledge is modelled using atomic semantic objects (symbols) and these symbols are manipulated using algorithmic rules
  - The connectivist approach: based on building systems similar to the architecture of the brain which, instead of being programmed, **learn based on experience**
- The symbolic approach is excellent for many problems (it became especially popular with the development of expert systems), but has failed to deliver on rather optimistic early promises
- The failure lies in the the faulty assumption that any piece of knowledge is easily formalized and that the brain works by manipulating these formalized representations using rules.

Connectivist approach

- Many every day tasks are too complex for for symbolic representation, e.g. pattern recognition:
- Our mother we can recognize within 0.1 s
- Neurons in the brain are activated approximately every millisecond
- It follows that in this time at most 100 neurons can activate
- Processing must be parallel!



Artificial neural network - definition

In general: an artificial replica of the human brain designed to simulate the process of learning and data processing.

Artificial neural network

Artificial neural network is a set of interconnected simple processing elements (neurons) whose functionality is based on the biological neuron and which are used for distributed parallel information processing.

- Enable robust data processing
- Can be used for classification or regression problems.
- Can learn from data.

Training an artificial neural network

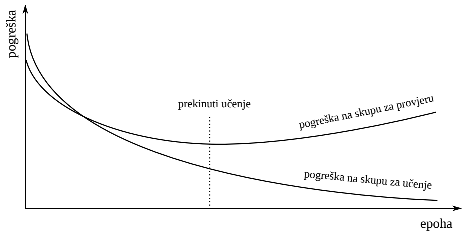
- Two phases of using an ANN:
  1. The training phase
  2. The data processing phase(exploitation, predicting on unseen data).
- Training is an iterative procedure of presenting the network with input examples (data or experience) often paired with desired network outputs and gradually modifying the weights based to achieve desired behaviour of the ANN.
- One pass through all the training examples is called an epoch
- Types of training:
  1. On-line: weights are modified after each training example
  2. mini-batch: weights are modified after a subset of training examples
  3. batch: weights are modified after all training examples (at the end of an epoch)
- The knowledge of how to map inputs to outputs is stored in the ANN implicitly in the weights on the neuron connections.

Training an artificial neural network

- Training of neural networks can be:
  1. supervised (learning with a teacher)
  2. unsupervised (learning without a teacher)
  3. reinforcement
- The set of examples available for learning is often divided into:
  1. Training set:
     e.g. 70% of available examples,
     used for the iterative updates of weights
  2. Validation set:
     e.g. 15% of available examples,
     we check the generalization power of the ANN on this set
  3. Test set:
     e.g. 15% of available examples,the final evaluation of
     the network and comparison to different models

Training an artificial neural network

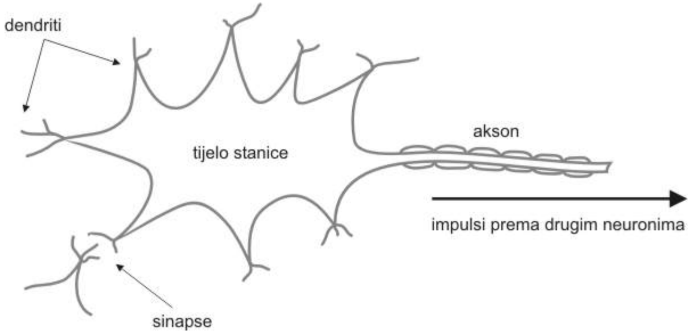
- Training is done on the train set keeping track of model performance (an error measure is introduced).
- Performance is also tracked on the validation set (but the network is not trained on this set).
- Overfitting: the network loses the ability to generalize to unseen data and becomes too adjusted to train data.
- Training is stopped when the error on the validation set starts rising.



Contents

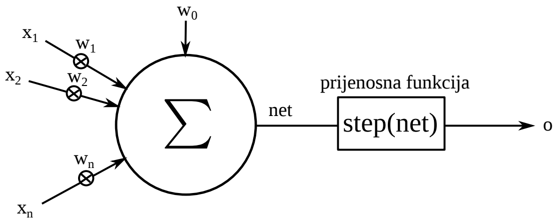
Biological neuron

- The biological neuron consists of the body (soma), dendrites, axon and axon terminals.
- In the human brain each neuron is connected to 1000 – 10000 other neurons, on average



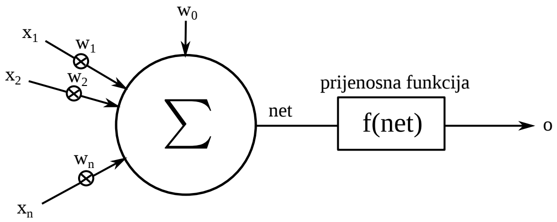
Artificial neuron

- McCulloch-Pitts define a simple model of a biological neuron (1943.): TLU-perceptron (engl. *Threshold Logic Unit*)
  - The value of each input x_i is multiplied by the sensitivity of the input w_i and accumulated in the body.
  - A bias term w_0 is added to the total sum (also sometimes denoted b). This defines the accumulated value $net = \left(\sum_{i=1}^{n} x_i \cdot w_i\right) + w_0$.
  - This value is transformed by the transfer function, producing the final output of the neuron: $o = step(net)$.



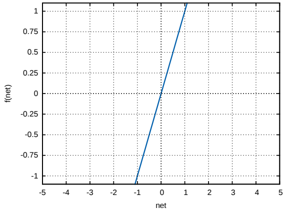
Artificial neuron

- In general, the model of an artificial neuron allows the accumulated value to be passed through an arbitrary transfer function f

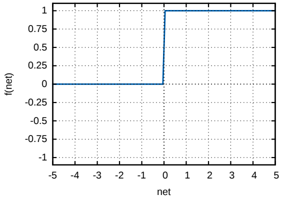


- Some often used transfer functions:
  - The identity (ADALINE-neuron)
  - The step function (TLU-perceptron)
  - The sigmoid function
  - Hyperbolic tangent
  - The hinge function (*Rectified Linear Unit*, ReLU)
  - Leaky hinge function (*Leaky Rectified Linear Unit*, LReLU)

Transfer functions



(a) Identity function



(b) Step function

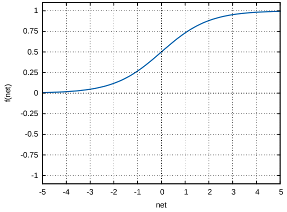
- Identity

  $f(net) = net$

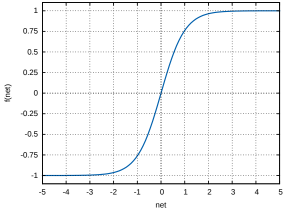
- Step function

  $$f(net) = step(net) = \begin{cases} 0 & net < 0 \\ 1 & \text{otherwise} \end{cases}$$

Transfer functions



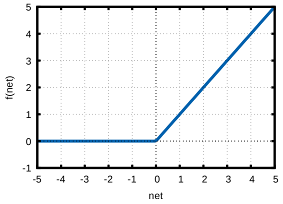
(c) Sigmoid function



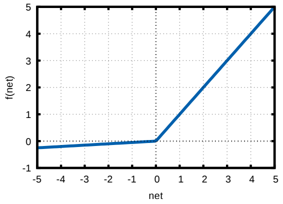
(d) Hyperbolic tangent

- Sigmoid function
  $f(net) = sigm(net) = \frac{1}{1+e^{-net}}$
- Hyperbolic tangent
  $f(net) = tanh(net) = \frac{2}{1+e^{-2 \cdot net}} - 1 = 2 \cdot sigm(2x) - 1$

Transfer functions



(e) Hinge



(f) Leaky hinge ($\alpha = 0.05$)

- Hinge
  $$f(net) = \max(0, net)$$
- Leaky hinge
  $$f(net) = \begin{cases} net & net >= 0 \\ \alpha \cdot net & \text{otherwise} \end{cases}$$

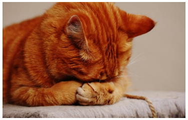
Classification

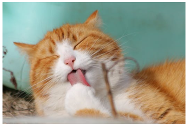
- Classification is the process of assigning labels to examples based on their features (e.g., color, shape, weight, ...).
- We would like to build a classifier that, based on examples from the training set, learn to classify new unseen examples.
- If there are only two classes, we are dealing with binary classification.
    - We can use a system that will have only one output with two clearly separated state (for example: 0 and 1; or alternatively -1 and 1) which define the class that is assigned to the input example.
- If there are multiple classes (but each example can be assigned into exactly one of them), it is common practice to use one-hot encoding: there are as many outputs as there are classes, with output i being 1 if the input example currently under consideration belongs to class i and 0 otherwise.

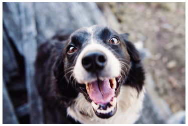
Binary classification

Let's assume we are building a binary classifier that classifies input images as images of dogs and images of cats. Suppose the "dog" class is encoded as 0 at the model output and the "cat" class is encoded as 1. Then for the images below we would expect outputs as denoted below the images.

(a) 0

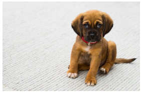

(b) 1

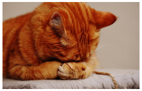

(c) 1

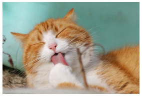

(d) 0

Multiclass classification

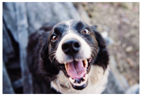
Let us no consider a system that would classify images into one of three classes: images of dogs, images of cats, and images of parrots. We could then ask the output of the network to be a one-hot encoded vector: 100 - for dog, 010 - for cat, and 001 for parrot.


(a) 100


(b) 010


(c) 010

(d) 001


(e) 100

(f) 001

Binary classification: an example

During the hiring procedure in the ACME company:

- each candidate goes through two independent rounds of interviews and obtains a score between 1 and 5 for each round
- based on those two scores, the HR manager provides a positive or negative opinion
- candidates that have obtained a positive opinion pass into subsequent phases of the interview process

To optimise business processes the company is considering automating a part of this hiring process. Specifically, possibilities of building a computer system to give a positive or negative opinion are being explored.
To make this possible a data set containing past decisions done by the HR manager has been prepared.

Binary classification: an example

- Denote by x_1 the score of a candidate in the first round and by x_2 the score of the candidate in the second round
- The company archives contain documents providing the following information about scores and corresponding decisions for four candidates:
  - $(x_2, x_1) = (2, 5)$: opinion was positive
  - $(x_2, x_1) = (5, 2)$: opinion was positive
  - $(x_2, x_1) = (1, 5)$: opinion was negative
  - $(x_2, x_1) = (5, 1)$: opinion was negative
- Based on this information we would like to train a classifier which could make decisions for future candidates.

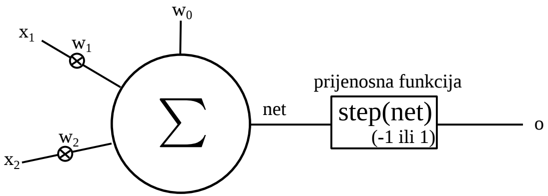
Binary classification: an example

- Since we have only two classes, as our classifier we will use a single neuron with a step transfer function, which has an output of -1 or 1, depending on the input.
  - we will encode the class *negative opinion* as -1
  - we will encode the class *positive opinion* as 1
- This yields the following set of training examples:

x_2	x_1	Class	t
2	5	positive opinion	1
5	2	positive opinion	1
1	5	negative opinion	-1
5	1	negative opinion	-1

Binary classification: an example

Our chose neuron:



- At the input x_1 we put the candidate score from the first round
- At input x_2 we put the candidate score from the second round
- The accumulated sum is $net = x_2 \cdot w_2 + x_1 \cdot w_1 + w_0$
- The output of the neuron is $o = step(net)$

Binary classification: an example

Let us assume the weights were chosen at random and are $w_2 = 1$, $w_1 = 1.3$, and $w_0 = -5.85$. How would the neuron classify examples from the training set?

(x_2, x_1, x_0)	t	(w_2, w_1, w_0)	***net***	o	**Correct**
$(2, 5, 1)$	1	$(1, 1.3, -5.85)$	2.65	1	yes
$(5, 2, 1)$	1	$(1, 1.3, -5.85)$	1.75	1	yes
$(1, 5, 1)$	-1	$(1, 1.3, -5.85)$	1.65	1	no
$(5, 1, 1)$	-1	$(1, 1.3, -5.85)$	0.01	1	no

Binary classification: an example

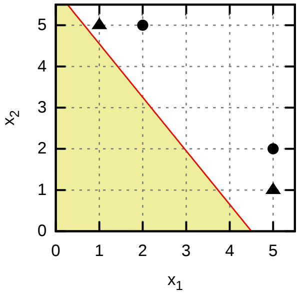
The decision boundary for a TLU neuron is linear:

- $o = step(net)$ changes its value when the sign of net changes
- therefore, we are interesting in the case where $net = 0$, which is: $x_2 \cdot w_2 + x_1 \cdot w_1 + w_0 = 0$
- this is a line in a two dimensional space with axes x_1 and x_2: $x_2 = -\frac{w_1}{w_2} x_1 - \frac{w_0}{w_2}$ where $-\frac{w_1}{w_2}$ is the slope of the line
- This line splits the two dimensional space into two subspaces: one of them contains al examples for which the classifier outputs -1 and the other contains all examples for which the classifier outputs 1
- In three dimensions the decision boundary is a plane and in more dimensions it is a hyperplane

Binary classification: an example

Lets consider our example where we calculate $net = 1 \cdot x_2 + 1.3 \cdot x_1 - 5.85$.

- The decision boundary is given in the image.
- Examples for which the classifier outputs -1 are in the yellow subspace (e.g., example (1,1) is in that subspace).
- Examples for which the classifier outputs 1 are in the white subspace (e.g., example (5,5) is in that subspace).
- The image shows that the classifier is not perfect and makes mistakes. It assigns the label 1 to all examples from our training set.



Learning from data

- in 1949. Hebb while studying biological neurons gains the following insight: *learning means changing the strength of connections between neurons*
- in 1958. Rosenblatt combines Hebbs idea with the McCulloch-Pitts neuron model and defines *the perceptron learning rule*

The perceptron learning rule

1. Cyclically iterate through all N training examples, one at a time.
2. Classify the current example.
    1. If the classification is correct, do not change the weights and
        1. if this is the N-th consecutive correctly classified example, stop learning,
        2. otherwise go to the next example.
    2. If the classification is incorrect, modify the neuron weights using the following expression:
       $w_i(k+1) \leftarrow w_i(k) + \eta \cdot (t - o) \cdot x_i$

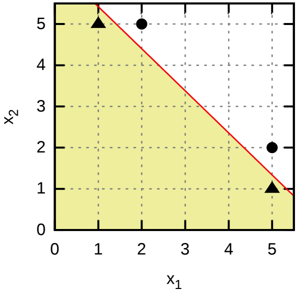
Learning from data

- The parameter η (eta) is called the learning rate
  - It is a small positive number (e.g., between 0.001 i 0.5) that determines the intensity of changes to the values of neuron weights
  - If η is too small, learning will be very slow
  - If η is too big, the algorithm might diverge
- Let's run the learning algorithm with $\eta = 0.02$ (see the materials "Umjetne neuronske mreže" at `http://java.zemris.fer.hr/nastava/ui/`, chapter 2 for the entire example).
- The algorithm finishes with these weights: $w_2 = 0.92,\ w_1 = 0.94,\ w_0 = -5.93.$

Binary classification: an example

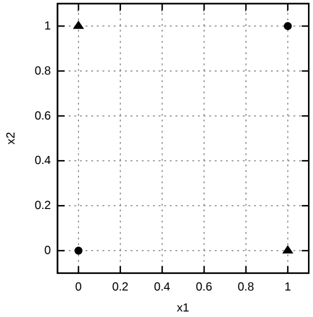
Now the TLU-perceptron calculates $net = 0.92 \cdot x_2 + 0.94 \cdot x_1 - 5.93$.

- All examples are correctly classified.
- The system has learned to generalize to some extent: examples like (1,1) are labeled as -1 and examples like (5,5) are labeled as 1, in line with what we would expect of a human to decide



Limitations of the TLU-perceptron

- The TLU-perceptron has a linear decision boundary
- Such a neuron can not solve classification problems where the classes are not linearly-separable
- An example of classes that are not linearly separable is shown in the image (the XOR function)
- To solve such more complex problems we will consider systems built of several neurons: artificial neural networks



Matrix notation

For implementation, especially when dealing with neural networks, it is convenient to use matrix notation.

- We will stick with the following conventions:
  - vector will be assumed to be a single column matrix
  - notation $\vec{y} = f(\vec{x})$ where f is a scalar function of a single variable will denote a new vector with values that are obtained by applying f to each element of $\vec{x}$, i.e., $y(i) = f(x(i))$.
- Let a neuron have n inputs; we denote them as $\vec{x} = (x_1, x_2, \ldots, x_n)$
- This means there are n weights $\vec{w} = (w_1, w_2, \ldots, w_n)$ and one bias term b (sometimes denoted w_0)
- The output of a neuron is now:

$$o = f(net) = f(\vec{w}^T \cdot \vec{x} + b)$$

- If we have access to an appropriate library for matrix and vector operations, our code can be very short and efficient.

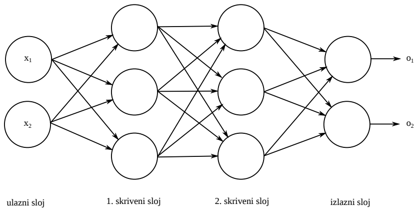
Contents

Architectures

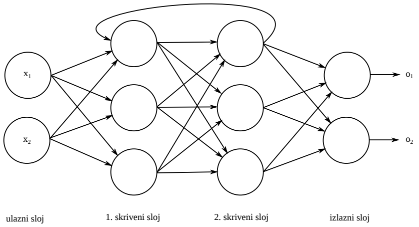
To enable modeling of more complex relations in classification and regression tasks, we will use more neurons.

- The notion of neural network *architecture* tells us how many neurons there are and how they are interconnected

Layered neural network $2 \times 3 \times 3 \times 2$



Non-layered neural network

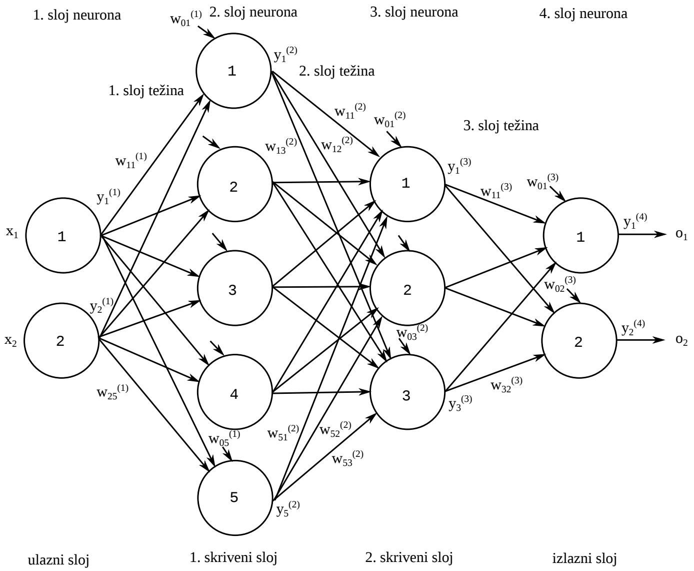


- The problem with the network on the right is the recurrent connection which makes it no longer be feed-forward making the job of the learning algorithm more difficult.

The need for non-linear transfer functions

- Assume we have build a multi layered neural network where all neurons use the identity function as the transfer function.
- This entire network is equally expressive as a single neuron of this type: a linear combination of linear combinations is again a linear combination.
- To increase the expressive power of the network, and allow the modelling of non-linear relationships, neurons must have transfer functions that are non-linear.
- In feedforward neural networks:
  - until recently it was common practice to use sigmoidal transfer functions
  - today ReLU functions are preferred, as they make easier training of deeper (more layers) neural networks

Feedforward layered neural network: an example



Slika: Feedforward layered fully-connected neural network

Feedforward layered neural network: an example

- our network has two hidden layers and one output layer
- it performs the following mapping $\mathbb{R} \times \mathbb{R} \to \mathbb{R} \times \mathbb{R}$
- conventions:
  - we will denote the input of the network as a two component vector $\vec{x}$
  - the output of each subsequent layer will also be a vector of dimensionality that corresponds to the number of neurons in that layer
  - the outputs of hidden layers are usually denoted as h (for *hidden*), so we will have $\vec{h}_1$ (dim=5), $\vec{h}_2$ (dim=3), and $\vec{y} = \vec{h}_3$ (dim=2)

Feedforward layered neural network: an example

- Let's consider the first neuron in the first hidden layer. It computes:

$$y_1^{(2)} = f\left(\begin{bmatrix} w_{11}^{(1)} & w_{21}^{(1)} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + w_{01}^{(1)}\right)$$

- The second neuron in the first layer computes:

$$y_2^{(2)} = f\left(\begin{bmatrix} w_{12}^{(1)} & w_{22}^{(1)} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + w_{02}^{(1)}\right)$$

- If we use matrix notation, we can write the entire output of the layer as:

$$\begin{bmatrix} y_1^{(2)} \\ y_2^{(2)} \\ y_3^{(2)} \\ y_4^{(2)} \\ y_5^{(2)} \end{bmatrix} = f\left(\begin{bmatrix} w_{11}^{(1)} & w_{21}^{(1)} \\ w_{12}^{(1)} & w_{22}^{(1)} \\ w_{13}^{(1)} & w_{23}^{(1)} \\ w_{14}^{(1)} & w_{24}^{(1)} \\ w_{15}^{(1)} & w_{25}^{(1)} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} w_{01}^{(1)} \\ w_{02}^{(1)} \\ w_{03}^{(1)} \\ w_{04}^{(1)} \\ w_{05}^{(1)} \end{bmatrix}\right)$$

Feedforward layered neural network: an example

- A more concise notation of the previous expression:

$$\vec{h}_1 = f(\mathbf{W}_1 \cdot \vec{x} + \vec{b}_1)$$

  where $\vec{h}_1$ is the vector of outputs from the first hidden layer, $\mathbf{W}_1$ is a weight matrix (each line represents one neuron), $\vec{x}$ is the input vector, and $\vec{b}_1$ is the vector of biases (one for each neuron)
- further generalisation yields the following expression:

$$\vec{h}_i = f(\mathbf{W}_i \cdot \vec{h}_{i-1} + \vec{b}_i)$$

  with $\vec{h}_0 = \vec{x}$ and $\vec{y} = \vec{h}_3$, which is easily implemented
- note: for each layer we must have a weight matrix, a bias vector, and a transfer function

Learning: modifying weights

- Values into which the network will map the input vector are determined by weights and biases of the neural network.
- During supervised learning of a neural network, we have available a training set of examples which are pairs of the form (input $\rightarrow$ desired output): $\{(x_{1,1},\ldots,x_{1,N_i}) \rightarrow (t_{1,1},\ldots,t_{1,N_o}),\ldots,(x_{N,1},\ldots,x_{N,N_i}) \rightarrow (t_{N,1},\ldots,t_{N,N_o})\}$ where N_i is the input dimensionality and N_o the output dimensionality
- To get to an algorithm for modifying the weights, we must first define an error function E. A popular choice is the halved sum of mean square errors over outputs:

$$E = \frac{1}{2}\sum_{s=1}^{N} E(s) = \frac{1}{2}\sum_{s=1}^{N}\frac{1}{N}\sum_{i=1}^{N_o}(t_{s,i}-o_{s,i})^2. \tag{1}$$

Learning: modifying weights

- One an error function is defined, and it is differentiable, its possible to write an optimisation algorithm based on gradients (partial derivatives of the error function with respect to the weights and biases)
- partial derivatives tell us how the error function will change if we slightly increase the corresponding weight
- we can exploit this information to change the weights in such a way to make the value of the error function E decrease

Error Backpropagation

An algorithm for training neural networks based on efficient computations of partial derivatives with respect to the network weights and applying them to determine the amounts by which to correct the weights is called Error Backpropagation.

We will not do a derivation, but will give the final result.

Error Backpropagation: the algorithm

1. Initial all neural network weights to random values
2. Repeat until the stopping criterion is met
    1. For each example $s : (x_{s,1}, \dots, x_{s,N_i}) \to (t_{s,1}, \dots, t_{s,N_o})$ do:
        1. Set the example $(x_{s,1}, \dots, x_{s,N_i})$ as the network input.
        2. Calculate outputs of all neurons in all layers, starting from the first layer to the last; denote the output of the last layer as: $(o_{s,1}, \dots, o_{s,N_o})$.
        3. Determine errors of neurons in the *output* layer

$$\delta_i^K = o_{s,i} \cdot (1 - o_{s,i}) \cdot (t_{s,i} - o_{s,i}).$$

        4. Go back layer by layer towards the first layer. For neuron i in layer k the error is:

$$\delta_i^{(k)} = y_i^{(k)} \cdot (1 - y_i^{(k)}) \cdot \sum_{d \in \mathrm{Downstream}} w_{i,d} \cdot \delta_d^{(k+1)}$$

        5. Apply corrections to all weights. Weight $w_{i,j}^{(k)}$ is modified as follows:

$$w_{i,j}^{(k)} \leftarrow w_{i,j}^{(k)} + \eta \cdot y_i^{(k)} \cdot \delta_j^{(k+1)}$$

and biases as follows:

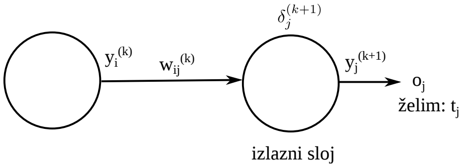
$$w_{0,j}^{(k)} \leftarrow w_{0,j}^{(k)} + \eta \cdot \delta_j^{(k+1)}.$$

Error Backpropagation: the algorithm

- In this course we will not cover the derivation of the Error Backpropagation algorithm
- A more detailed description of the algorithm is given in the additional materials
- The expressions in the given pseudocode assume that sigmoidal transfer functions were used
- In case of different transfer functions the parts of expressions that are there as the derivative of the sigmoidal function $(f \cdot (1 - f))$ should be replaced with the derivative of the alternative transfer function.
- The expressions for weight correction can be easily remembered "visually": demonstrated on the next couple of slides

Error Backpropagation: the algorithm

Calculating the error for a neuron of the output layer.

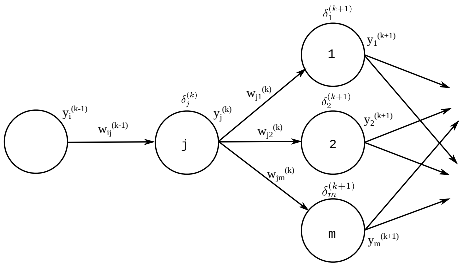


Error: product of the transfer function derivative $(y_j^{(k+1)} \cdot (1 - y_j^{(k+1)}))$ and the actual error $(t_j - o_j)$.

$$\delta_j^{k+1} = o_{s,j} \cdot (1 - o_{s,j}) \cdot (t_{s,j} - o_{s,j}).$$

Error Backpropagation: the algorithm

Calculating the error for a neuron of the hidden layer: $\delta_j^{(k)}$.

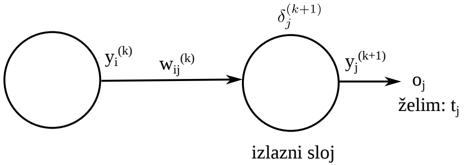


Pogreška: product of the transfer function derivative of the neuron and a weighted sum of errors of all neurons to which it sends its output:

$$\delta_j^{(k)} = y_j^{(k)} \cdot (1 - y_j^{(k)}) \cdot (w_{j,1}^{(k)} \cdot \delta_1^{(k+1)} + \cdots + w_{j,m}^{(k)} \cdot \delta_m^{(k+1)}).$$

Error Backpropagation: the algorithm

Corrections: proportional to the product of the learning rate η, the output of the neuron to the left of the weight and the error of the neuron to the right of the weight:



$$\Delta w_{ij}^k = \eta \cdot y_i^{(k)} \cdot \delta_j^{(k+1)}$$

$$w_{ij}^k \leftarrow w_{ij}^k + \Delta w_{ij}^k$$

For bias weights "to the left" is the constant 1, so we have:

$$\Delta w_{0j}^k = \eta \cdot \delta_j^{(k+1)}$$

Examples

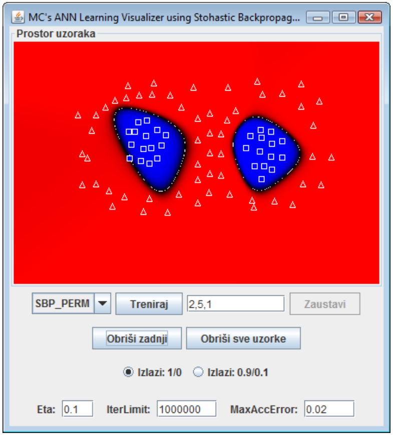
At the following url `http://java.zemris.fer.hr/nastava/ui/` under the *Umjetne neuronske mreže* section. There are implementations and descriptions of several examples:

- Classification of xamples in 2D
- Learning a function using a regression model
- Gesture classification - classifying a movement of the mouse as one of several possible gestures

We encourage you to download the examples and try them yourselves!

Example: classification and regression

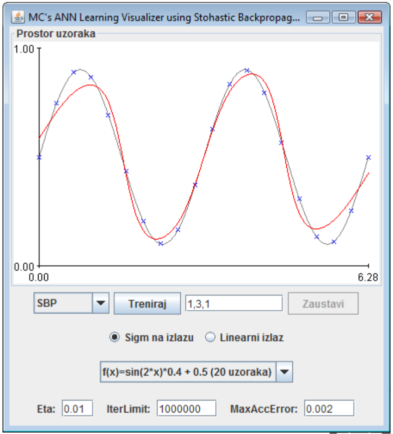
- We consider using a neural network for classification of 2D examples (left) or regression to learn a function (right)



2,5,1

[✓ Izlazi: 1/0] Izlazi: 0.9/0.1

Eta: 0.1 IterLimit: 1000000 MaxAccError: 0.02



1,3,1

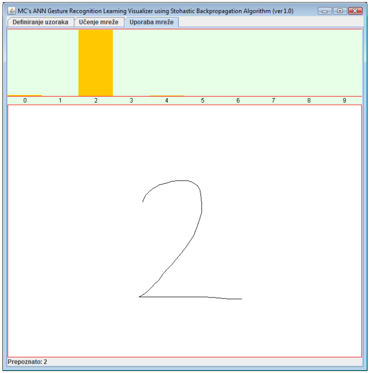
[✓ Sigm na izlazu] Linearni izlaz

f(x)=sin(2*x)*0.4 + 0.5 (20 uzoraka)

Eta: 0.01 IterLimit: 1000000 MaxAccError: 0.002

Example: recognizing digits as gestures

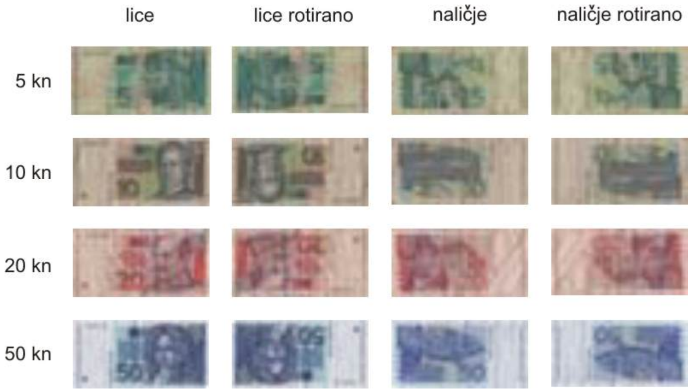
- Problem: create a program that uses a neural network for digit recognition (0-9) based on a gesture that the user performs in a single move of the mouse
- The program enables the gathering of training examples, training the network , and using the trained network to classify new examples


MC's ANN Gesture Recognition Learning Visualizer using Stohastic Backpropagation Algorithm (ver 1.0)
Definiranje uzoraka
Učenje mreže
Uporaba mreže
0
1
2
3
4
5
6
7
8
9

Prepoznato: 2

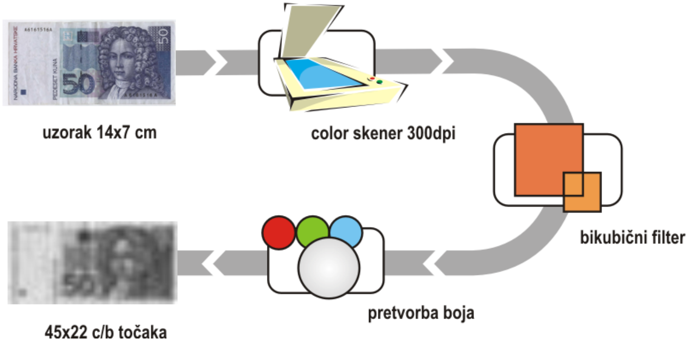
Example: bank note classification

- Problem: classify four types of paper bank notes regardless of orientation
- The training set consists of 16 examples sampled with 45x22 image elements



Example: bank note classification

- Before being presented to the neural network examples are preporcessed

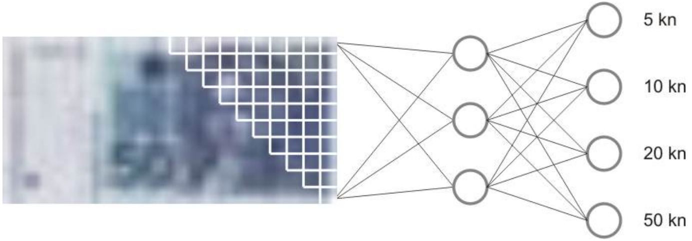


Example: bank note classification

- Examples are in reality digital images, so they will differ in intensity of corresponding image elements (a consequence of wear, crumpling, and other damage to the bank note).
- In the example a generator of artificial examples is supplied that can generate and simulate various types of damage to the bank note, in order to better test the robustness of the trained neural network.

Example: bank note classification

- Network parameters: acyclical fully connected multi layered neural network $990 \times 3 \times 4$
- Training: Error Backpropagation, learning rate=0.02, moment=0.02, performance is further checked on the validation set every 2500 epochs



More on this topic?

- Artificial neural networks have numerous applications nowadays (image, sound, text processing)
- Several types that were not mentioned today are heavily used: convolutional neural networks, recurrent/recursive neural networks
- More information on this is available in your graduate studies
  - Machine learning
  - Fuzzy, evolutionary and neurocomputing
  - Deep learning