Perceptrons and Backpropagation

Fabio Zachert
Cognitive Modelling
WiSe 2014/15

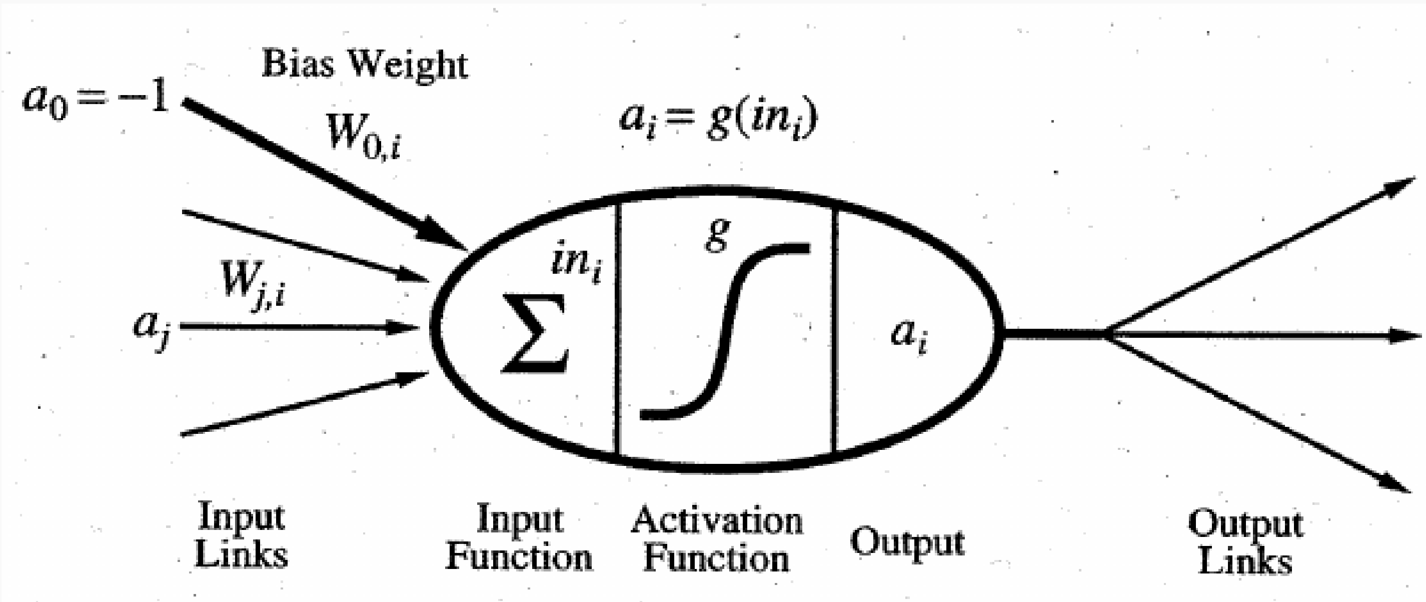
Content

- History
- Mathematical View of Perceptrons
- Network Structures
- Gradient Descent
- Backpropagation (Single-Layer-, Multilayer-Networks)
- Learning The Structure
- Questions

History

- Inspired by the brains' information processing
- 1943: Warren McCulloch and Walter Pitts create computational model
- 1974: Paul Werbos creates backpropagation-algorithm
- Generally used for regression and classification in AI

Mathematical Model of Perceptrons

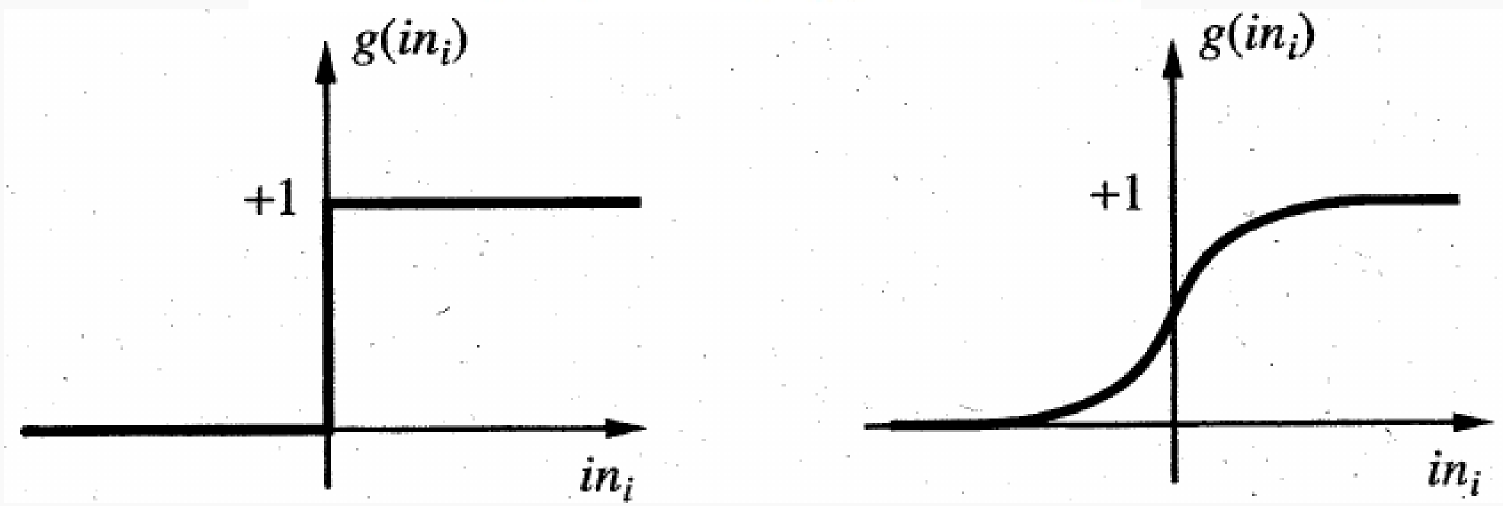


$$in_i = \sum_{j=0}^{n} W_{j,i} a_j$$

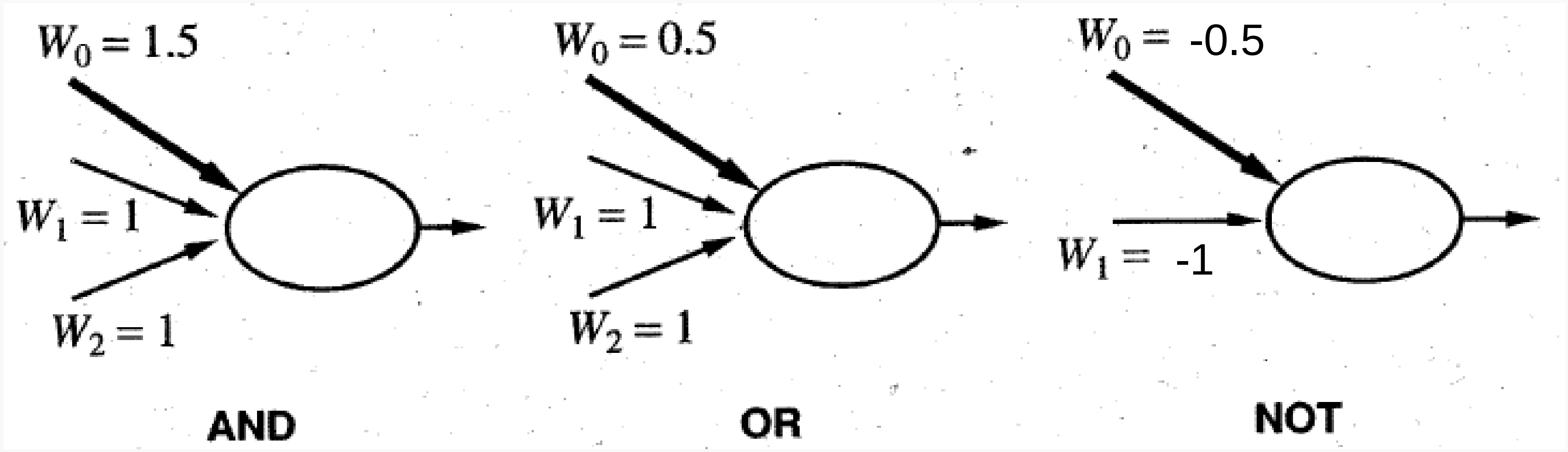
$$a_i = g(in_i) = g\left(\sum_{j=0}^{n} W_{j,i} a_j\right)$$

Mathematical Model of Perceptrons

- Activation function
- Non-linear function with output between 0 and 1
- Often a thresholdhold function or sigmoid
- Sigmoid: $\operatorname{sig}(t) = \frac{1}{1+e^{-t}} = \frac{1}{2} \cdot \left(1 + \tanh \frac{t}{2}\right)$



Boolean Function

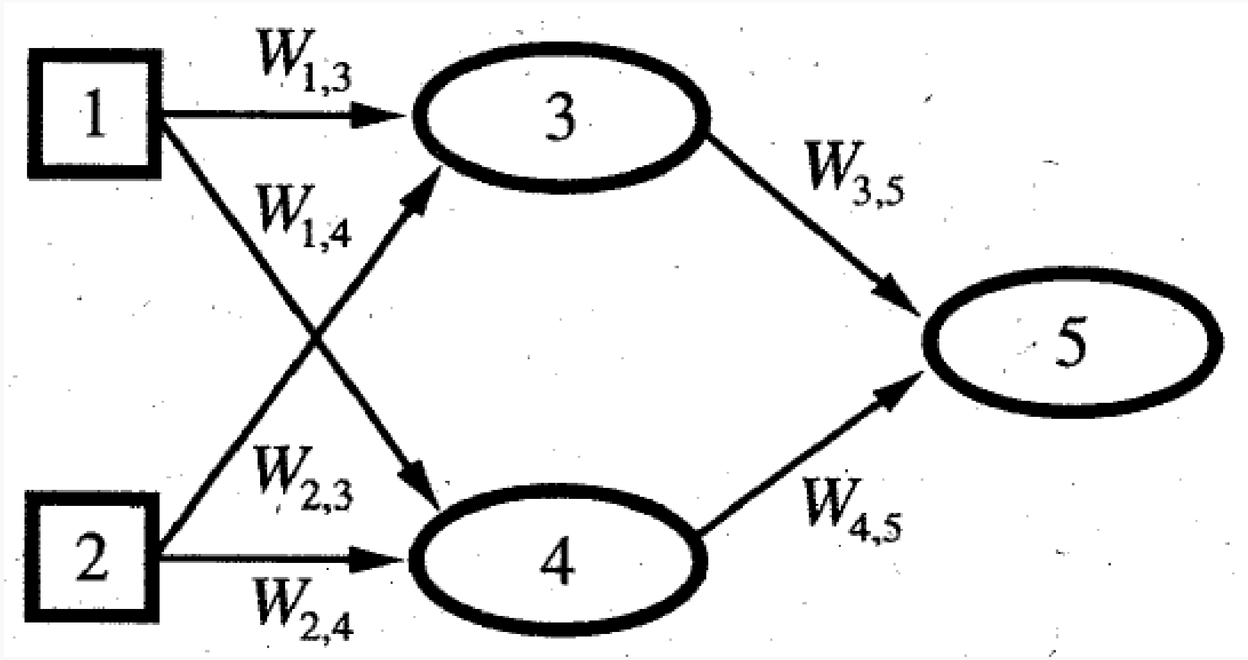


But how to create the XOR operator?
→ Use a network of units

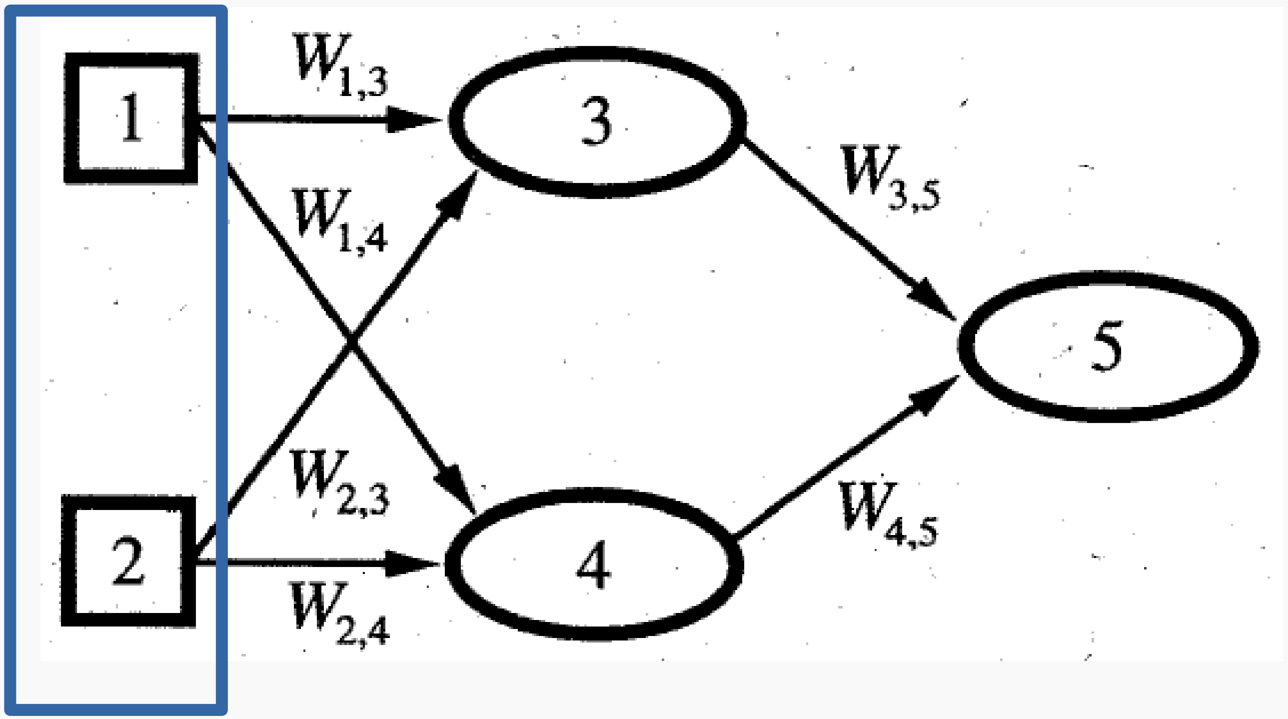
Network Structures

- Combine multiple units to generate more complex functions
- Feed-forward networks
  - No loops allowed
- Recurrent networks
  - Loops allowed
  - More complex functions
  - Possibility to simulate memory
  - No convergence guaranteed

Network Structures

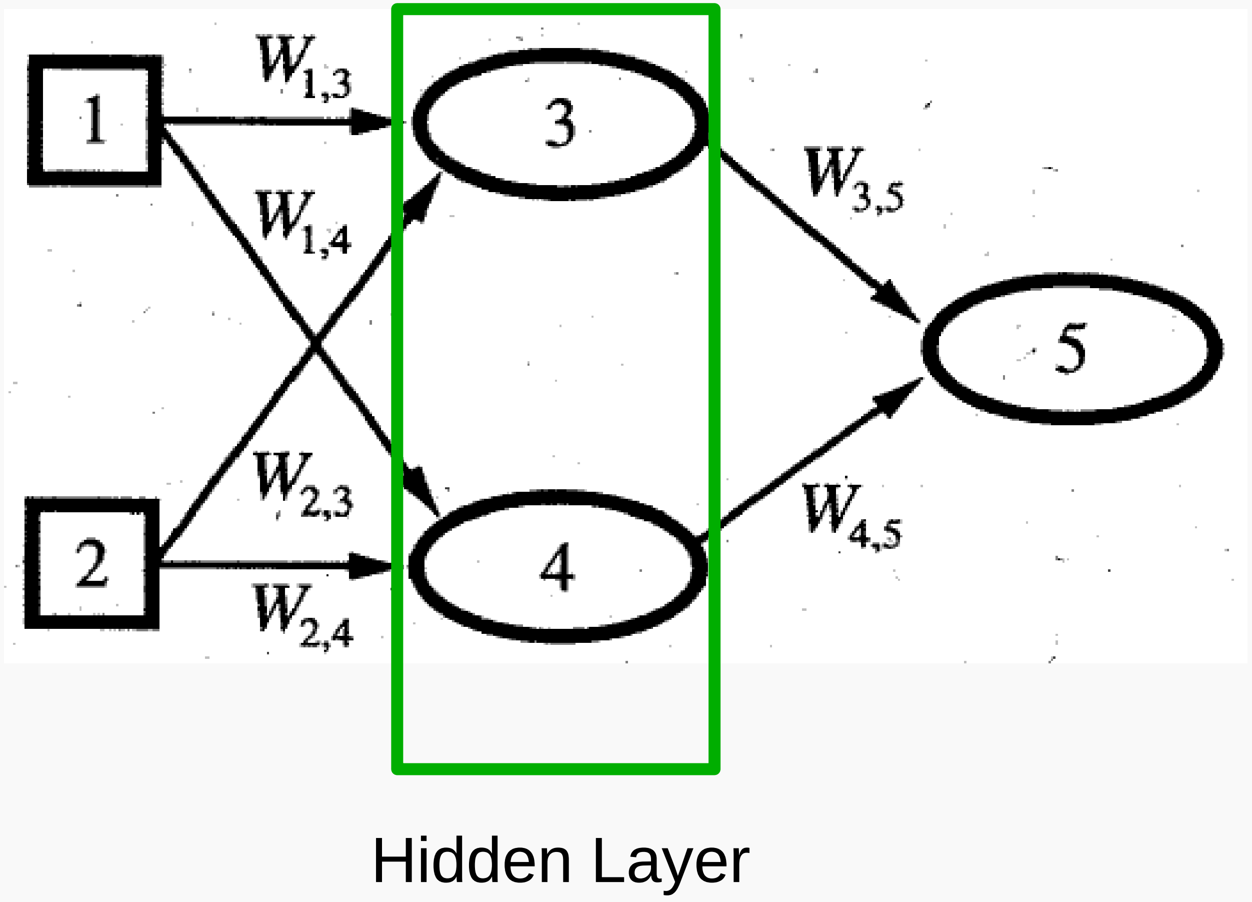


Network Structures

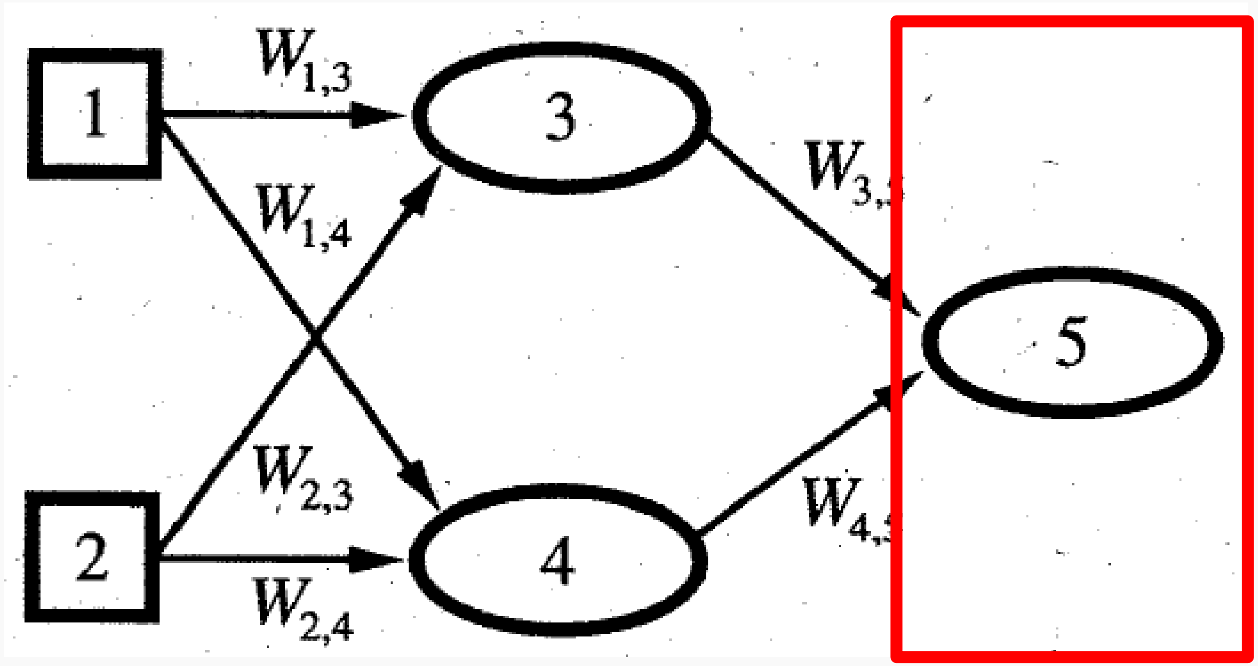


Input Units

Network Structures

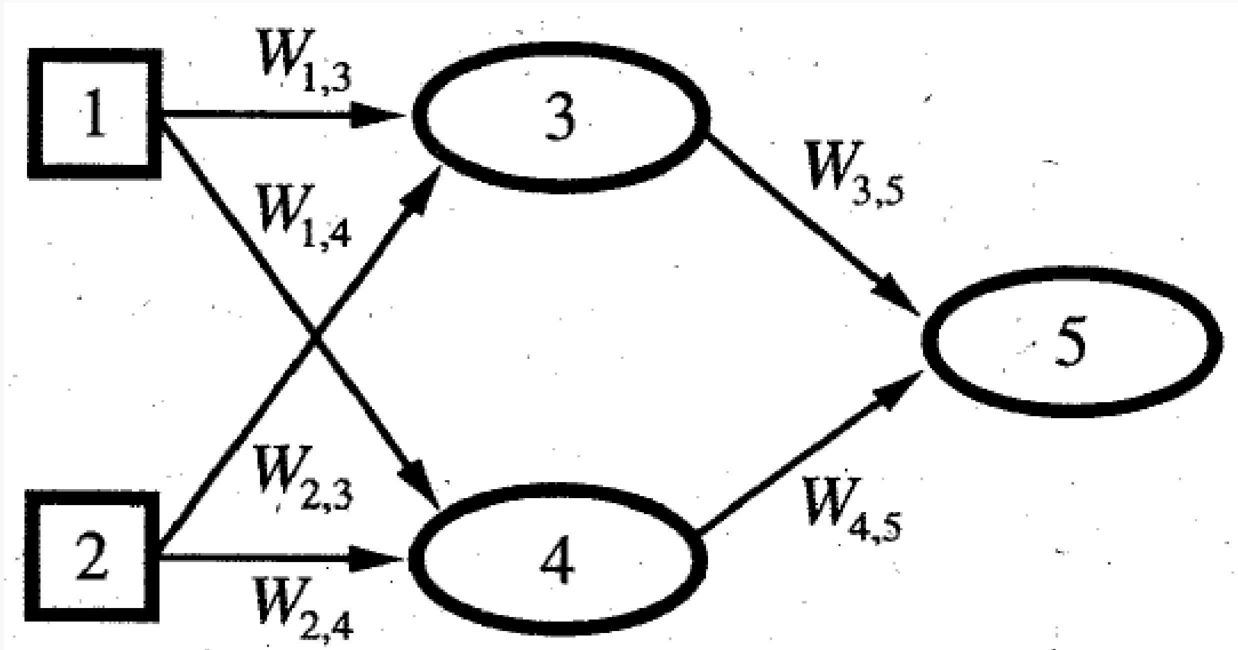


Network Structures



Output unit

Network Structures



$$
\begin{aligned}
a_5 &= g(W_{3,5}a_3 + W_{4,5}a_4) \\
&= g(W_{3,5}g(W_{1,3}a_1 + W_{2,3}a_2) + W_{4,5}g(W_{1,4}a_1 + W_{2,4}a_2))
\end{aligned}
$$

Gradient Descent

- Problem: Find $\text{argmin}_x f(x)$
- Minimum cannot be calculated directly
- Gradient defines the direction of the highest slope at a given point
- Use gradient of f(x) to search downhill in the direction of the minimum

- Algorithm:

  Init x

  Until convergence

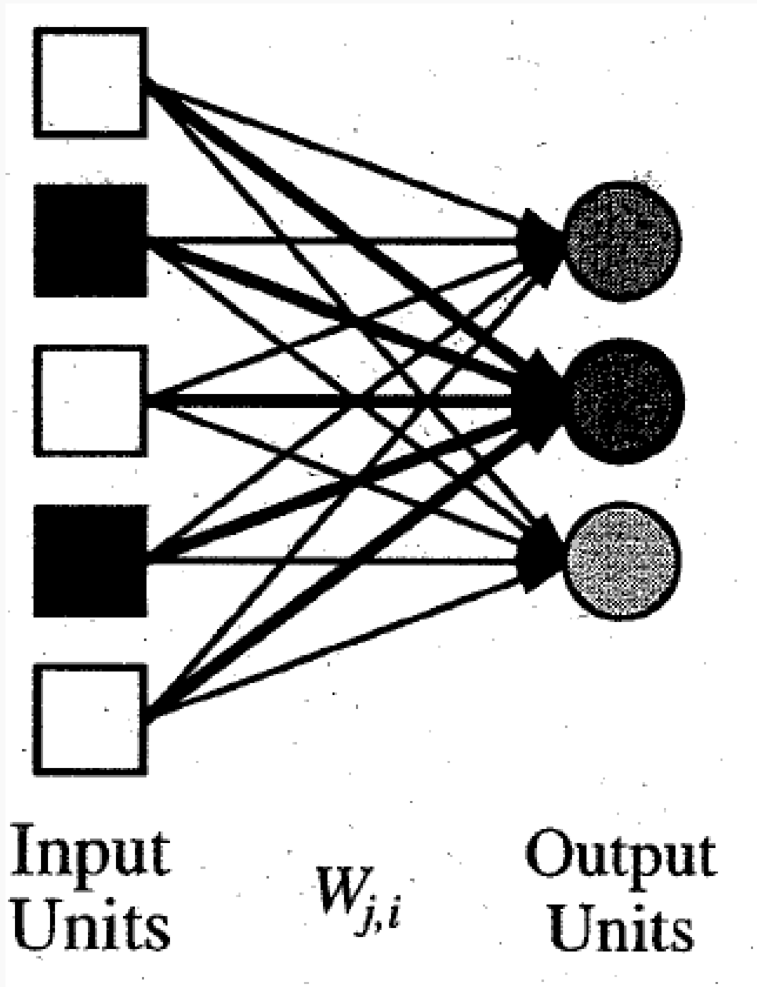
  $x \leftarrow x - \alpha * f'(x)$

- α is the learning rate

Gradient Descent

- When to converge?
  - After n steps
  - Gradient threshold
  - Gain threshold
- How to set learn rate α?
  - Experience of the programmer
  - Adaptive algorithms
- No guarantee to find global minimum

Single-Layer Backpropagation



Single-Layer Backpropagation

- A supervised learning task
- Given (x,y) pairs
  - Learn general function for all x

- Minimize sum of square error (SSE) for network-function h(x) with weight-vector w

$$E = \frac{1}{2} Err^2 \equiv \frac{1}{2}(y - h_{\mathbf{w}}(\mathbf{x}))^2$$

- Use gradient descent to find optimal w
  - Calculate partial derivation of the error over the weight-vector

Single-Layer Backpropagation

- Recall: Derivation
- Partial derivation $\frac{\partial f}{\partial x_i}$: derivation of a function with multiple input above one single input
- Rules:

Sum-rule: $(g+h)'(x) = g'(x) + h'(x)$

Chain-rule: $(g \circ h)'(x) = (g(h(x)))' = g'(h(x)) * h'(x)$

Product-rule: $(g * h)' = g' * h + g * h'$

Power-rule: $(x^n)' = n * x^{n-1}$

Constant-factor-rule: $f'(k * x) = k * f'(x)$

Single-Layer Backprogation

- Calculate partial derivation

$$
\begin{aligned}
\frac{\partial E}{\partial W_j} &= Err \times \frac{\partial Err}{\partial W_j} \\
&= Err \times \frac{\partial}{\partial W_j} g\left(y - \sum_{j=0}^{n} W_j x_j\right) \\
&= -Err \times g'(in) \times x_j \ ,
\end{aligned}
$$

- For $\mathrm{g(in)} = \mathrm{sig(in)} \rightarrow g'(in) = sig(in)(1 - sig(in))$

Single-Layer Backpropagation

Algorithm:

- Calculate perceptron output and error for each training sample
- Update weights, based on error-gradient

repeat

  for each e **in** $examples$ **do**

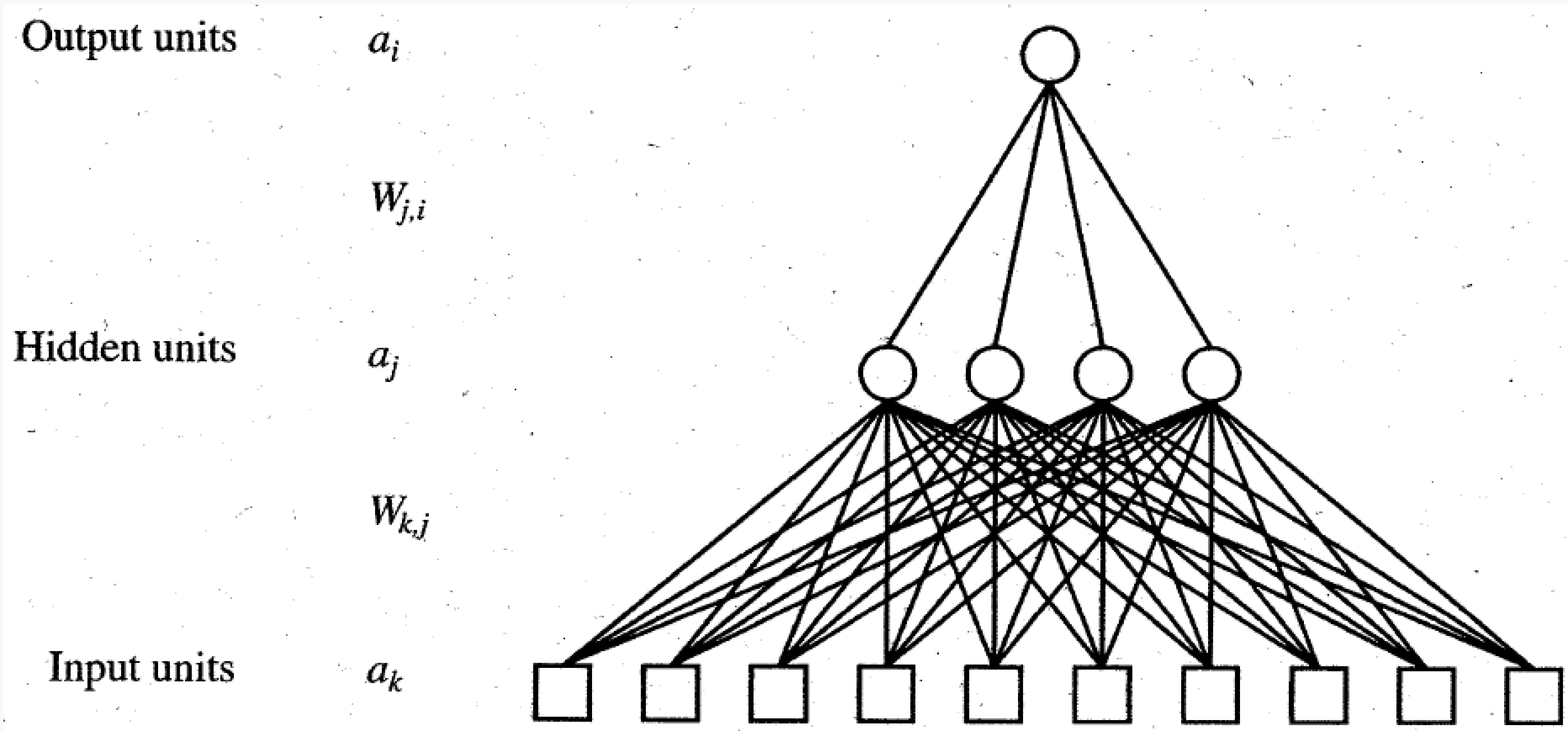
    $in \leftarrow \sum_{j=0}^{n} W_j \, x_j[e]$

    $Err \leftarrow y[e] - g(in)$

    $W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j[e]$

until some stopping criterion is satisfied

Multi-Layer Backpropagation



Multi-Layer Backpropagation

- Extend network with a hidden-layer
- Update-rule for output-layer based on output of hidden-layer

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i \quad \text{with} \quad \Delta_i = Err_i \times g'(in_i)$$

- Propagate the error back on the hidden layer
- Update-rule for hidden-layer is based on its output
- One hidden-layer unit is partly responsible for the errors of the connected output-units

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j \quad \text{with} \quad \Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i$$

Multi-Layer Backpropagation

Algorithm:

repeat
    for each e **in** *examples* **do**
        for each node j in the input layer **do** $a_j \leftarrow x_j[e]$
        for $\ell = 2$ **to** M **do**
            $in_i \leftarrow \sum_j W_{j,i}\, a_j$
            $a_i \leftarrow g(in_i)$
        for each node i in the output layer **do**
            $\Delta_i \leftarrow g'(in_i) \times (y_i[e] - a_i)$
        for $\ell = M - 1$ **to** 1 **do**
            for each node j in layer ℓ **do**
                $\Delta_j \leftarrow g'(in_j) \sum_i W_{j,i}\, \Delta_i$
                for each node i in layer $\ell + 1$ **do**
                    $W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$
until some stopping criterion is satisfied

Multi-Layer Backpropagation

Algorithm:

repeat
    for each e **in** *examples* **do**
        for each node j in the input layer **do** $a_j \leftarrow x_j[e]$
        for $\ell = 2$ **to** M **do**
            $in_i \leftarrow \sum_j W_{j,i}\, a_j$
            $a_i \leftarrow g(in_i)$
        for each node i in the output layer **do**
            $\Delta_i \leftarrow g'(in_i) \times (y_i[e] - a_i)$
        for $\ell = M - 1$ **to** 1 **do**
            for each node j in layer ℓ **do**
                $\Delta_j \leftarrow g'(in_j) \sum_i W_{j,i}\, \Delta_i$
                for each node i in layer $\ell + 1$ **do**
                    $W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$
until some stopping criterion is satisfied

- Init input-layer
- Feed-forward input through layers

Multi-Layer Backpropagation

Algorithm:

repeat
  for each e **in** *examples* **do**
    for each node j in the input layer **do** $a_j \leftarrow x_j[e]$
    for $\ell = 2$ **to** M **do**
      $in_i \leftarrow \sum_j W_{j,i}\, a_j$
      $a_i \leftarrow g(in_i)$
    for each node i in the output layer **do**
      $\Delta_i \leftarrow g'(in_i) \times (y_i[e] - a_i)$
    for $\ell = M - 1$ **to** 1 **do**
      for each node j in layer ℓ **do**
        $\Delta_j \leftarrow g'(in_j) \sum_i W_{j,i}\, \Delta_i$
        for each node i in layer $\ell + 1$ **do**
          $W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$
until some stopping criterion is satisfied

- Calculate error of output-layer

Multi-Layer Backpropagation

Algorithm:

repeat

  for each e **in** *examples* **do**

    for each node j in the input layer **do** $a_j \leftarrow x_j[e]$

    for $\ell = 2$ **to** M **do**

      $in_i \leftarrow \sum_j W_{j,i}\, a_j$

      $a_i \leftarrow g(in_i)$

    for each node i in the output layer **do**

      $\Delta_i \leftarrow g'(in_i) \times (y_i[e] - a_i)$

    for $\ell = M - 1$ **to** 1 **do**

      for each node j in layer ℓ **do**

        $\Delta_j \leftarrow g'(in_j) \sum_i W_{j,i}\, \Delta_i$

        for each node i in layer $\ell + 1$ **do**

          $W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$

until some stopping criterion is satisfied

- Propagate error backwards through the layers

Multi-Layer Backpropagation

Algorithm:

repeat
    for each e **in** *examples* **do**
        for each node j in the input layer **do** $a_j \leftarrow x_j[e]$
        for $\ell = 2$ **to** M **do**
            $in_i \leftarrow \sum_j W_{j,i}\, a_j$
            $a_i \leftarrow g(in_i)$
        for each node i in the output layer **do**
            $\Delta_i \leftarrow g'(in_i) \times (y_i[e] - a_i)$
        for $\ell = M - 1$ **to** 1 **do**
            for each node j in layer ℓ **do**
                $\Delta_j \leftarrow g'(in_j) \sum_i W_{j,i}\, \Delta_i$
                for each node i in layer $\ell + 1$ **do**
                    $W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$
until some stopping criterion is satisfied

- Correct weights based on error-gradient

Learning the Structure

- But, how to learn the size/structure of the network?
- Oversized nets tend to overfitting
  - Learn the general function and not the noise
- Try to estimate the optimal size:
  - Optimal-brain-damage-algorithm
  - Cross-validation

N-Fold-Crossvalidation

- Separate data for learning and evaluation
- Split dataset in n parts
- n-1 parts for training, 1 part for evaluation

1	2	3	....	n

- Use different parts for evaluation
- Calculate average evaluation error
- Use network-size with the lowest evaluation-error

Questions

Questions

- How to initialize weights?

Sources

- Russell, S., & Norvig, P. (2003). Artificial intelligence: A modern approach. Upper Saddle River, NJ: Pearson Education
- Madsen, K., Nielsen, H.B., Tingleff, O., (2004, 2nd Edition), Methods for non-linear least squares problems
- Murphy, Kevin P. (2012). Machine learning: a probabilistic perspective. MIT press
- http://en.wikipedia.org/wiki/Differentiation_rules

Thank you for your attention!