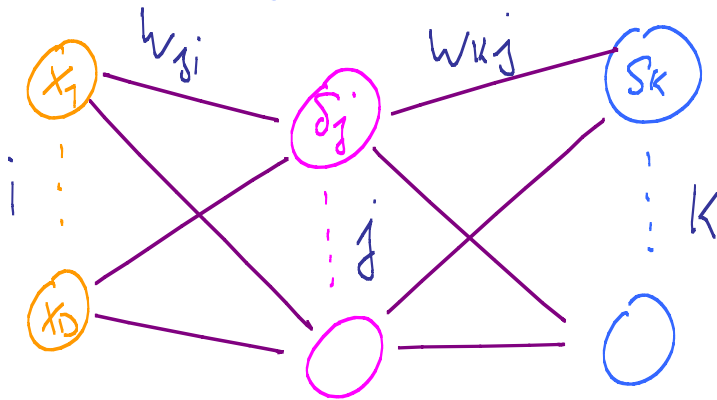


Backpropagation:



$$o_j = \sum_{i \in \text{pre}(j)} w_{ji} \cdot z_i \quad \dots \text{Input for Aktivierungsfunktion}$$

$$O_i = h_i(o_i)$$

$$z_i = \begin{cases} O_i \dots j \dots \text{Output} \\ x_i \dots j \dots \text{Hidden} \end{cases}$$

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial o_j} \cdot \frac{\partial o_j}{\partial w_{ji}} \rightarrow z_i$$

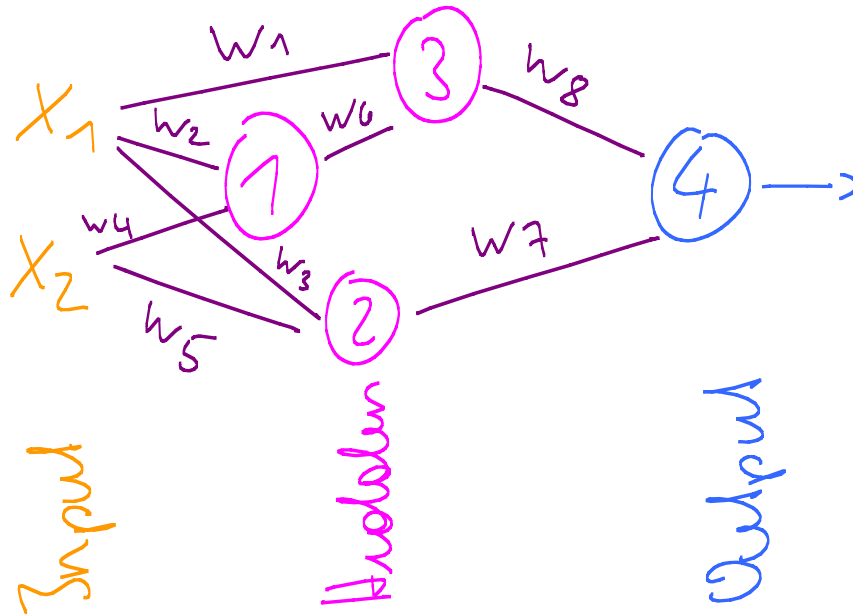
\downarrow
 δ_j

Outputlayer: $\delta_k = \frac{z}{2} (h(o_k) - \gamma_k) \cdot h'(o_k)$

Hiddenlayer: $\delta_j = h'(o_j) \sum_{k \in \text{post}(j)} w_{kj} \delta_k$

2. Schritte:
- Forwardpropagation
 \Rightarrow Berechne alle a_i, a_j
 - Backwardpropagation
 \Rightarrow Berechne δ_k, δ_j

Beispiel:



• Forwardpropagation

$$\begin{aligned}
 a_1 &= x_1 \cdot w_2 + x_2 \cdot w_4 \Rightarrow O_1 = h(a_1) \\
 a_2 &= x_1 \cdot w_3 + x_2 \cdot w_5 \Rightarrow O_2 = h(a_2) \\
 a_3 &= x_1 \cdot w_1 + O_1 \cdot w_6 \Rightarrow O_3 = h(a_3) \\
 a_4 &= O_3 \cdot w_8 + O_2 \cdot w_7 \Rightarrow O_4 = h(a_4)
 \end{aligned}$$

• Backwardpropagation

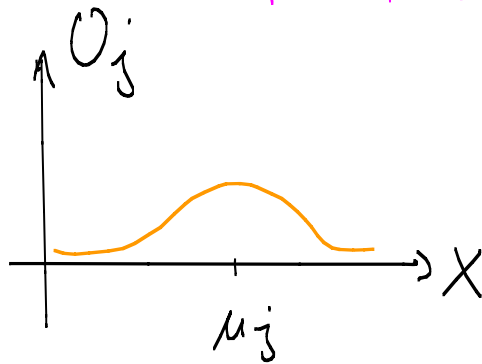
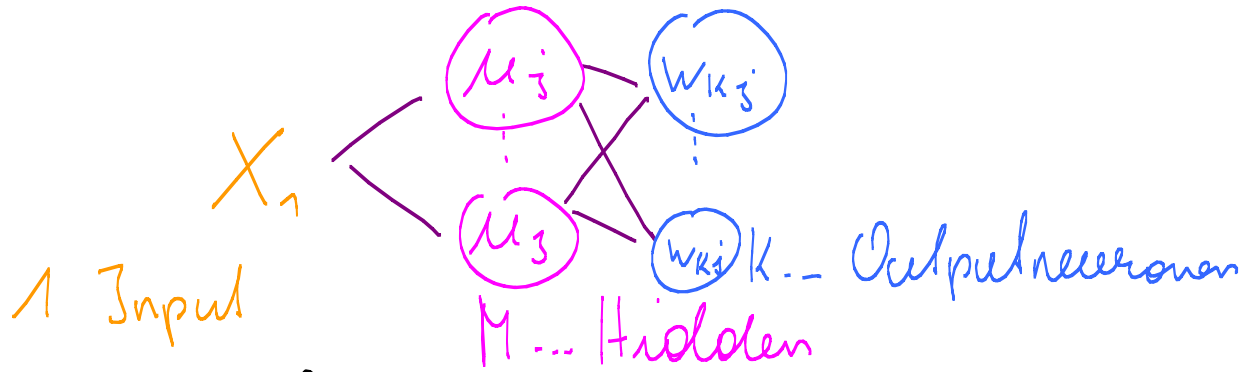
$$\begin{aligned}
 \delta_4 &= 2 \cdot (O_4 - y) \cdot h'(a_4) \\
 \delta_3 &= h'(a_3) \cdot \delta_4 \\
 \delta_2 &\dots \text{ nicht notwendig für } \frac{\partial E}{\partial w_2} \\
 \delta_1 &= h'(a_1) \cdot w_6 \cdot \delta_3
 \end{aligned}$$

$$\frac{\partial E}{\partial w_2} = \delta_1 \cdot x_1$$

zur Hausübung:

$$y_k = \tanh \left(\sum_{j=1}^M w_{jk} \exp(-(x - \mu_j)^2) \right)$$

Parameter des Netzes



Aktivierungsfunktion
ist glockenkurven-
ähnlich

Anzahl der Neuronen darf nicht zu hoch (overfitting) oder zu niedrig (underfitting) gewählt werden; allgemein ist sie abhängig von der Anzahl der Trainingssets

```
net.trainparam.epochs = 5; %#trainingsschritte  
net.trainparam.show = NAN;
```

```
net = init(net)
```

```
[net, tr_2hu] = train(net, inputdaten_training,  
outputdaten_training, [], [], [], []);
```

```
sim(net,x); %simuliert das netz mit dem Vektor x als input
```

```
mvnrand ... Normalverteilung
```

```
fuer Klassifikation:
```

```
K = #Klassen
```

```
full(ind2vec(t_train)) ... gibt die Klassen vor fuer das Netzwerk  
vor (one of n coding) zum training
```

```
net = newff([Wertebereich input als Vector], [#Neuron im Hidden,  
#Neuronen im Output], {[logsig ... sigmide gatter fuer hidden,  
logsig ... sigmide gatter Neuron fuer output, trainscg ...  
Trainingsfunktion]}) ... legt das Netzerk an
```

Hausuebung 2 wird eine Bildererkennung sein:
Jedes Pixel ist ein Input