Learning by

Error Back-Propagation

Training Multi-Layered Networks

How can we train a multi layer network?

- Perceptron Learning
  Requires that we know in which direction the weights should be changed to come nearer the solution.
  Does not work!
- Delta Rule
  Requires that we can measure the error before thresholding, but this only works for the last layer.
  Does not work!

Training Multi-Layered Networks

Dilemma:

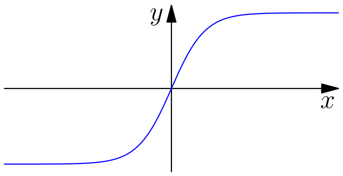
- Thresholding destroys information needed for learning
- Without thresholding we loose the advantage of multiple layers

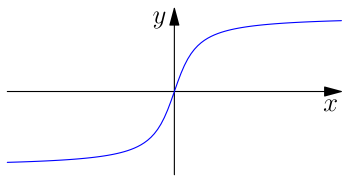
Solution

Use threshold-like but differentiable squashing functions

Using Smooth Functions

Two commonly used squashing functions $\varphi(\sum)$





$$\varphi(x) = \frac{1 - e^{-x}}{1 + e^{-x}} \qquad \varphi(x) = \arctan(x)$$

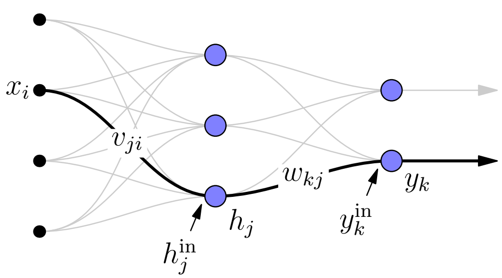
Generalization of the Delta Rule:

1. Choose a cost function ε

$$\varepsilon = \frac{1}{2}||\vec{t} - \vec{y}||^2 = \frac{1}{2}\sum_k (t_k - y_k)^2$$

2. Minimize it using Steepest Decent
Compute the gradient, i.e.

$$\frac{\partial \varepsilon}{\partial v_{ji}} \quad \text{and} \quad \frac{\partial \varepsilon}{\partial w_{kj}}$$



First case: derivative w.r.t. a weight w_{kj} in the second layer

$$\begin{aligned}
\frac{\partial \varepsilon}{\partial w_{kj}} &= \frac{\partial \varepsilon}{\partial y_k} \cdot \frac{\partial y_k}{\partial w_{kj}} \\
&= -(t_k - y_k) \cdot \frac{\partial \varphi(y_k^{\text{in}})}{\partial w_{kj}} \\
&= -(t_k - y_k) \cdot \varphi'(y_k^{\text{in}}) \cdot \frac{\partial y_k^{\text{in}}}{\partial w_{kj}} \\
&= -(t_k - y_k) \cdot \varphi'(y_k^{\text{in}}) \cdot h_j \\
&= -\delta_k h_j
\end{aligned}$$

Here we have introduced $\delta_k = (t_k - y_k) \cdot \varphi'(y_k^{\text{in}})$

Second case: derivative w.r.t. a weight v_{kj} in the first layer

$$\begin{aligned}
\frac{\partial \varepsilon}{\partial v_{ji}} &= \sum_k \frac{\partial \varepsilon}{\partial y_k} \cdot \frac{\partial y_k}{\partial v_{ji}} \\
&= -\sum_k (t_k - y_k) \cdot \frac{\partial y_k}{\partial v_{ji}} \\
&= -\sum_k (t_k - y_k) \cdot \varphi'(y_k^{\text{in}}) \cdot \frac{\partial y_k^{\text{in}}}{\partial v_{ji}} \\
&= -\sum_k \delta_k \cdot \frac{\partial y_k^{\text{in}}}{\partial v_{ji}} \\
&= -\sum_k \delta_k \cdot w_{kj} \cdot \frac{\partial h_j}{\partial v_{ji}}
\end{aligned}$$

We continue. . .

$$\begin{aligned}
\frac{\partial \varepsilon}{\partial v_{ji}} &= -\sum_k \delta_k \cdot w_{kj} \cdot \frac{\partial h_j}{\partial v_{ji}} \\
&= -\sum_k \delta_k \cdot w_{kj} \cdot \varphi'(h_j^{\text{in}}) \cdot \frac{\partial h_j^{\text{in}}}{\partial v_{ji}} \\
&= -\sum_k \delta_k \cdot w_{kj} \cdot \varphi'(h_j^{\text{in}}) \cdot x_i \\
&= -\delta_j x_i
\end{aligned}$$

Here we have introduced $\delta_j = \sum_k \delta_k \cdot w_{kj} \cdot \varphi'(h_j^{\text{in}})$

Summary

$$\frac{\partial \varepsilon}{\partial w_{kj}} = -\delta_k h_j \qquad \text{where } \delta_k = (t_k - y_k) \cdot \varphi'(y_k^{\text{in}})$$

$$\frac{\partial \varepsilon}{\partial v_{ji}} = -\delta_j x_i \qquad \text{where } \delta_j = \sum_k \delta_k \cdot w_{kj} \cdot \varphi'(h_j^{\text{in}})$$

Gradient Decent

$$\Delta w_{kj} = \eta \delta_k h_j$$

$$\Delta v_{ji} = \eta \delta_j x_i$$

Error Back-Propagation

1. Forward Pass: Compute all h_j and y_k

$$h_j = \varphi(\sum_i v_{ji} x_i) \qquad y_k = \varphi(\sum_j w_{kj} h_j)$$

2. Backward Pass: Compute all δ_k and δ_j

$$\delta_k = (t_k - y_k) \cdot \varphi'(y_k^{\text{in}}) \qquad \delta_j = \sum_k \delta_k \cdot w_{kj} \cdot \varphi'(h_j^{\text{in}})$$

3. Weight Updating:

$$\Delta w_{kj} = \eta \delta_k h_j \qquad \Delta v_{ji} = \eta \delta_j x_i$$

Things to Be Aware Of

Problems with BackProp

- Does not always converge (gets stuck in local minima)
- Slow convergence
- Many parameters need to be tuned
- Bad scaling behavior for large problems
- Biologically unrealistic
  - Backward propagating signal
  - Requires known target values

Things to Be Aware Of

Tips when using BackProp

- Use an antisymmetric $\varphi(x)$
- Put the target values $\vec{t}$ inside the domain interval of φ
- Order or weigh the training examples so the hard examples dominate
- Choose smart initial weights
- Introduce momentum in the weight updating
- Add random noise to the weights during training
- Remove the squashing-function ($\varphi(x)$) for the output units

Things to Be Aware Of

Preprocessing of input patterns

1. Subtract the average
2. Decorrelate
3. Normalize the variance