Machine Learning

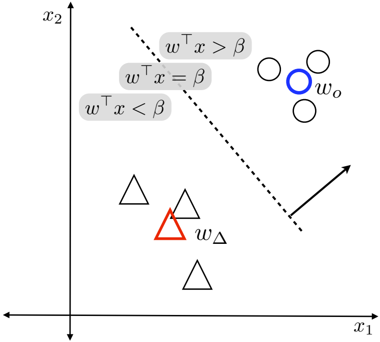
Neural Networks

Felix Bießmann

Beuth University & Einstein Center for Digital Future

June 25, 2019

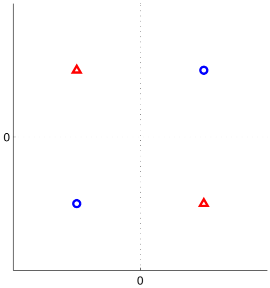
Linear Classification

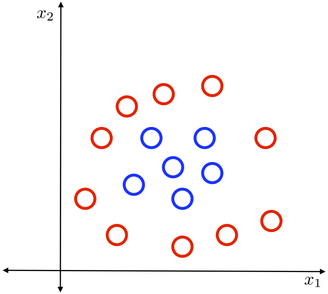


$$\phi(\mathbf{w}^\top \mathbf{x} - \beta) = \begin{cases} > 0 & \text{if } \mathbf{x} \text{ is from class } o \\ < 0 & \text{if } \mathbf{x} \text{ is from class } \Delta \end{cases}$$

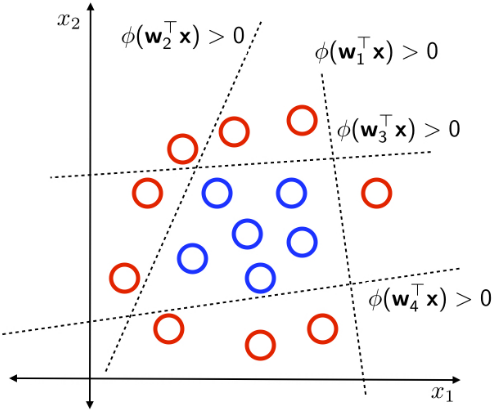
Problems with Perceptrons

Perceptrons can only learn linearly separable problems.



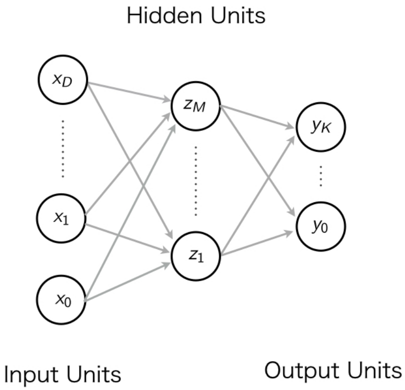


Deep Neural Networks



Deep Neural Networks

Combinations of Perceptrons (Multi Layer Perceptrons):



Neurons (Units), that are neither output nor input are called **Hidden Units**.

A Short History of Deep Learning

1943 First mathematical Neuron Model (Mcculloch and Pitts, 1943)

1957 Perceptron Algorithm (Rosenblatt, 1958)

1969 Perceptrons cannot solve non-linearen Problems (Minsky and Papert, 1969)

1970 Backpropagation: Efficient gradient computations (Linnainmaa, 1970)

1980 Computer Hardware $\approx$ 10,000 faster compared to 1960/1970 – Automatic Differentiation (Speelpenning, 1980)

1986 Backpropagation learns meaningful representations (Rumelhart et al., 1986), NETtalk (Sejnowski and Rosenberg, 1986)

1992 Support-Vector Machines (SVMs) (Boser et al., 1992)

2000 Computer Hardware (GPUs) $\approx$ 10,000 faster compared to 1980/1990, Bigger datasets render kernel SVMs computationally infeasible

2012 Deep Convolutional Networks wins ImageNet (Krizhevsky et al., 2012)

2014 Neural Machine Translation surpasses traditional methods

2017 Neural Networks for Reinforcement Learning excell at Go (AlphaGo Zero)

2018 ImageNet Moment for Neural Language Models (BERT / ELMO)

Sources: Juergen Schmidhuber's page and others

Universal Approximation Theorem

[Cybenko, 1989]

Multilayer Perceptrons with one hidden layer and a finite number of hidden units can approximate any function.

Training of Deep Neural Networks

- Training: Gradient Descent
- Problem: Gradient Computations
  - Mathematically challenging for complex models
  - Computationally challenging

→ Solution: **Backpropagation**
  - Elegant formulation
  - Efficient implementation

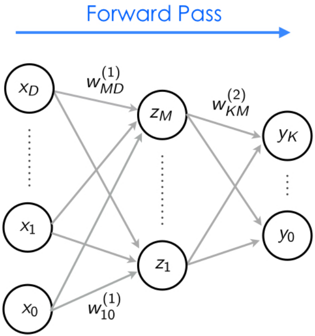
Backpropagation Algorithm

Algorithm 1 Backpropagation Algorithm - Pseudocode

Require: Data $\mathbf{X} \in \mathbb{R}^{D \times N}$, labels $\mathbf{Y} \in \mathbb{R}^{K \times N}$, untrained network
Ensure: Network parameters $\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(V)}$

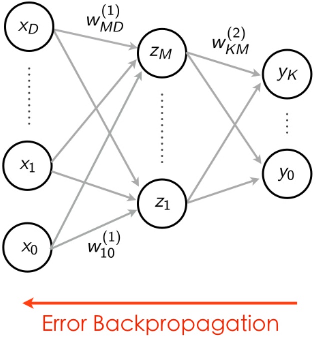
```
while Not converged do
    # Compute network predictions
    # Evaluate error function
    # Propagate error from output layer back to input layer
    # Take gradient descent step
end while
```

Learning with Backpropagation in Neural Networks



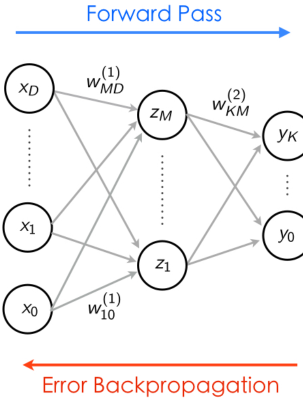
Computation of network predictions is called **Forward Propagation**.

Learning with Backpropagation in Neural Networks



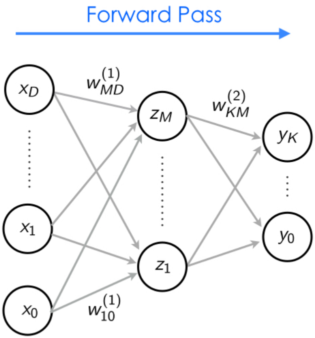
Backpropagation refers to efficient computation of error function gradients for all connections.

Learning with Backpropagation in Neural Networks



After a forward and backward pass a gradient step is performed.

Forward Pass



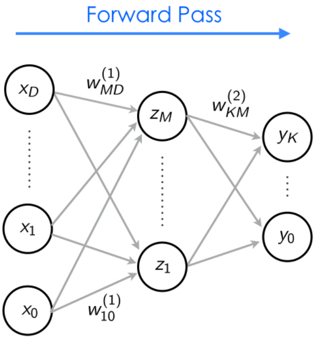
Each neuron computes a weighted sum a_j of its inputs

$$a_j = \sum_i w_{ji}^{(v)} z_i \tag{1}$$

and transforms a_j with some non-linear function $\phi(.)$

$$z_j = \phi(a_j). \tag{2}$$

Forward Pass



Each neuron computes a weighted sum a_j of its inputs

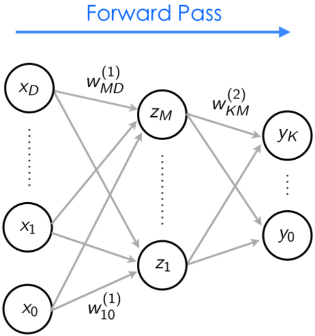
$$a_j = \sum_i w_{ji}^{(v)} z_i \qquad (1)$$

and transforms a_j with some non-linear function $\phi(.)$

$$z_j = \phi(a_j). \qquad (2)$$

Input Layer: $z_i \equiv x_i$
Output Layer: $z_i \equiv \hat{y}_i$

Forward Pass



Each neuron computes a weighted sum a_j of its inputs

$$a_j = \sum_i w_{ji}^{(v)} z_i \tag{1}$$

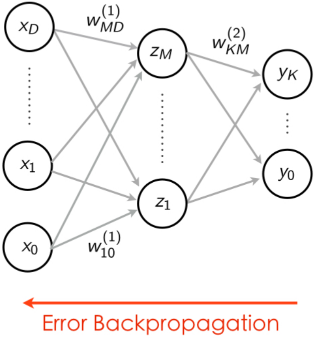
and transforms a_j with some non-linear function $\phi(.)$

$$z_j = \phi(a_j). \tag{2}$$

After a forward pass the error function is evaluated:

$$J(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{2}(\hat{\mathbf{y}} - \mathbf{y})^2 \tag{3}$$

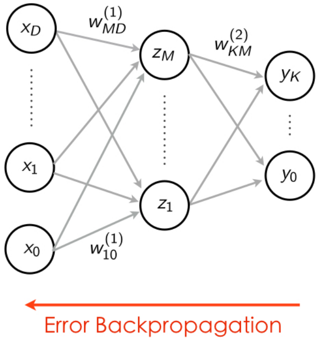
Error Backpropagation



Goal:
Computation of gradient of error function J

$$\frac{\partial J(\mathbf{y}, \mathbf{x}, \mathbf{W}^{(1)}, \mathbf{W}^{(2)})}{\partial w_{ji}^{(v)}} \tag{4}$$

Error Backpropagation



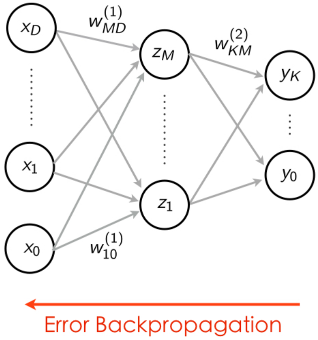
Backpropagation Idea

$w_{ji}^{(v)}$ changes J
only through summed up inputs a_j

Gradient of error function (chain rule):

$$\frac{\partial J}{\partial w_{ji}^{(v)}} = \frac{\partial J}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}}$$

Error Backpropagation



Error function gradient:

$$\frac{\partial J}{\partial w_{ji}^{(v)}} = \frac{\partial J}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}^{(v)}} \tag{4}$$

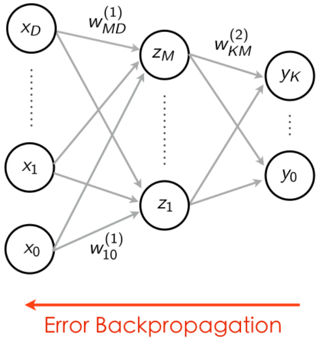
From 1:

$$a_j = \sum_i w_{ji}^{(v)} z_i$$

$$\frac{\partial a_j}{\partial w_{ji}^{(v)}} = \frac{\partial \sum_i w_{ji}^{(v)} z_i}{\partial w_{ji}^{(v)}} \tag{5}$$

$$= z_i$$

Error Backpropagation



Error function gradient:

$$\frac{\partial J}{\partial w_{ji}^{(v)}} = \frac{\partial J}{\partial a_j} \underbrace{\frac{\partial a_j}{\partial w_{ji}}}_{z_i}$$

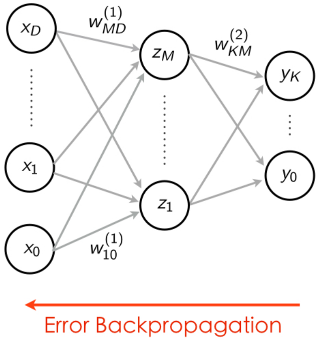
at output units ($\mathbf{z} \equiv \mathbf{a} \equiv \hat{\mathbf{y}}$):

Outer derivative of J, e.g.

$$J = \frac{1}{2}(\hat{\mathbf{y}} - \mathbf{y})^2 \tag{4}$$

$$\frac{\partial J}{\partial a_j} = (\hat{\mathbf{y}} - \mathbf{y}) \equiv \delta_j \tag{5}$$

Error Backpropagation



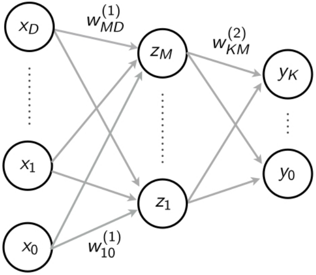
Error function gradient:

$$\frac{\partial J}{\partial w_{ji}^{(v)}} = \underbrace{\frac{\partial J}{\partial a_j}}_{\delta_j} \underbrace{\frac{\partial a_j}{\partial w_{ji}}}_{z_i}$$

δ_j is the error signal of the **receiving** neurons j

z_i is the activation of the **sending** neuron i

Error Backpropagation



Error Backpropagation

Error function gradient:

$$\frac{\partial J}{\partial w_{ji}^{(v)}} = \underbrace{\frac{\partial J}{\partial a_j}}_{\delta_j} \underbrace{\frac{\partial a_j}{\partial w_{ji}}}_{z_i}$$

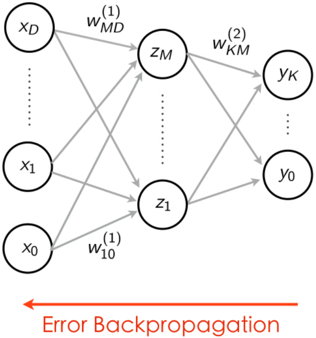
at hidden units:

Backpropagation Idea

a_j changes J **only** via outputs to a_k

$$\delta_j \equiv \frac{\partial J}{\partial a_j} = \sum_k \frac{\partial J}{\partial a_k} \frac{\partial a_k}{\partial a_j} \qquad (4)$$

Error Backpropagation



Error function gradient:

$$\frac{\partial J}{\partial w_{ji}^{(v)}} = \underbrace{\frac{\partial J}{\partial a_j}}_{\delta_j} \underbrace{\frac{\partial a_j}{\partial w_{ji}}}_{z_i}$$

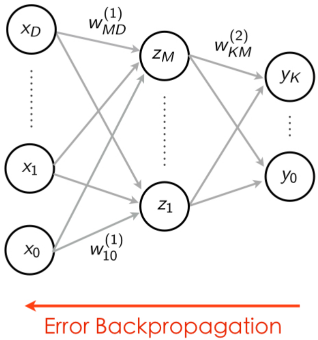
δ_j at hidden units:

$$\delta_j \equiv \frac{\partial J}{\partial a_j} = \sum_k \frac{\partial J}{\partial a_k} \frac{\partial a_k}{\partial a_j} \tag{4}$$

$$\frac{\partial a_k}{\partial a_j} = \frac{\partial w_{kj}^{(v)} \phi(a_j)}{\partial a_j}$$

$$= w_{kj}^{(v)} \phi'(a_j) \tag{5}$$

Error Backpropagation



Error function gradient:

$$\frac{\partial J}{\partial w_{ji}^{(v)}} = \underbrace{\frac{\partial J}{\partial a_j}}_{\delta_j} \underbrace{\frac{\partial a_j}{\partial w_{ji}}}_{z_i}$$

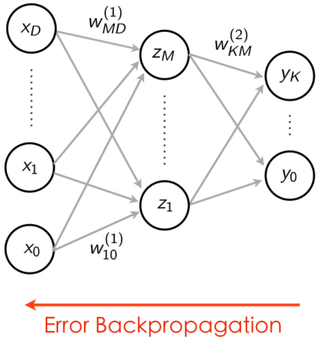
δ_j at hidden units:

$$\delta_j \equiv \frac{\partial J}{\partial a_j} = \sum_k \frac{\partial J}{\partial a_k} \frac{\partial a_k}{\partial a_j} \tag{4}$$

$$\frac{\partial a_k}{\partial a_j} = w_{kj}^{(v)} \phi'(a_j)$$

$$\frac{\partial J}{\partial a_k} = \delta_k$$

Error Backpropagation



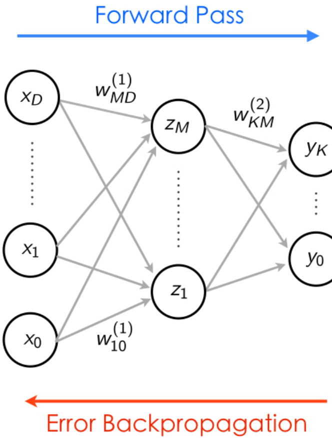
Error function gradient:

$$\frac{\partial J}{\partial w_{ji}^{(v)}} = \underbrace{\frac{\partial J}{\partial a_j}}_{\delta_j} \underbrace{\frac{\partial a_j}{\partial w_{ji}}}_{z_i}$$

δ_j at hidden units:

$$\delta_j \equiv \frac{\partial J}{\partial a_j} = \phi'(a_j) \sum_k w_{kj}^{(v)} \delta_k \tag{4}$$

Error Backpropagation - Simple Example



Network with one hidden layer and linear output function

$$\hat{\mathbf{y}} = \mathbf{a}$$

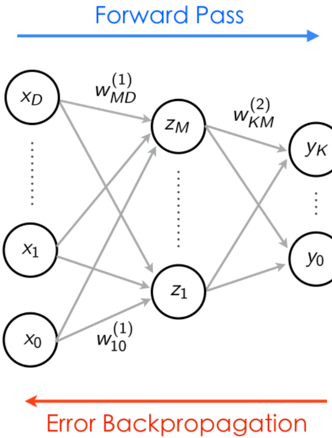
$\phi(z)$ is the logistic function

$$\phi(z) = \frac{1}{1 + e^{-z}}$$

J is the quadratic error

$$J = \frac{1}{2}(\hat{\mathbf{y}} - \mathbf{y})^2$$

Error Backpropagation - Simple Example



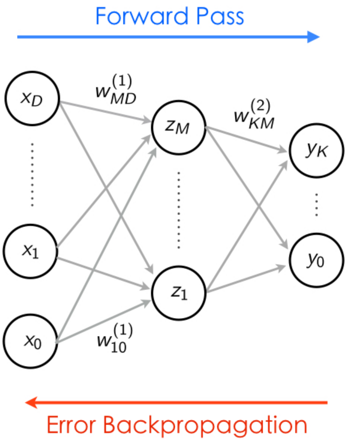
For each datapoint $\mathbf{x}_i$ the prediction $\hat{\mathbf{y}}$ is computed

$$a_j = \sum_{i=0}^{D} w^{(1)}_{ji} x_i$$

$$z_j = \phi(a_j)$$

$$\hat{y}_k = \sum_{j=0}^{M} w^{(2)}_{kj} z_j$$

Error Backpropagation - Simple Example



The error signal δ is:

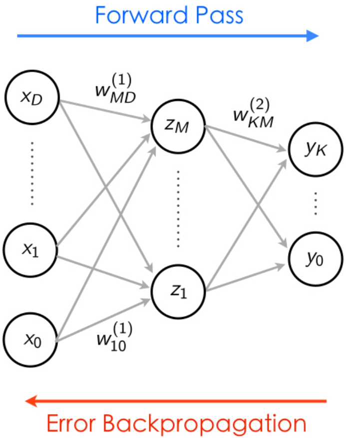
at the output layer

$$\delta_k = \hat{y}_k - y_k$$

at hidden units

$$\begin{aligned}\delta_j &= \phi'(z_j)\sum_{k=1}^{K} w^{(2)}_{kj}\delta_k \\ &= \phi(z_j)(1-\phi(z_j))\sum_{k=1}^{K} w^{(2)}_{kj}\delta_k\end{aligned}$$

Error Backpropagation - Simple Example



Full gradient

$$\frac{\partial J}{w_{kj}^{(2)}} = \delta_k z_j$$
$$\frac{\partial J}{w_{ji}^{(1)}} = \delta_j x_i$$

Other Loss Functions

Error Function	Used in
$\frac{1}{2}(y - \mathbf{w}^\top \mathbf{x})^2$	Adaline [Widrow and Hoff, 1960]
$\max(0, -y\mathbf{w}^\top \mathbf{x})$	Perceptron [Rosenblatt, 1958]
$-\sum_{k=1}^{K} y_{\text{true}} \log(y_{\text{predicted}})$	Most classification neural networks

Cross-Entropy

$$-\sum_{k=1}^{K} y_{\text{true}} \log(y_{\text{predicted}}) \tag{5}$$

Where

- K is the number of classes
- y_{true} is the one-hot encoded label
- $\mathbf{z}_k$ is the activity of the kth neuron in the last layer and

$$y_{\text{predicted}} = \frac{e^{\mathbf{z}_k}}{\sum_{k=1}^{K} e^{\mathbf{z}_k}} \tag{6}$$

Backpropagation Algorithm

Algorithm 2 Backpropagation Algorithm

Require: Data $\mathbf{X} \in \mathbb{R}^{D \times N}$, labels $\mathbf{Y} \in \mathbb{R}^{K \times N}$, untrained network
Ensure: network parameters $\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(V)}$

while Not converged **do**
    # **Forward Propagation**
    # Input Layer:
    $\mathbf{z}_0 = \phi(\mathbf{W}^{(0)}\mathbf{x}_i)$
    for Layer $v = 1, \ldots, V$ **do**
        $\mathbf{z}_v = \phi(\mathbf{W}^{(v)}\mathbf{z}_{v-1})$
    end for
    # Error Computation at Output Layer (quadratic error)
    $\delta_{V+1} = \mathbf{z}_V - \mathbf{y}_i$
    # **Backpropagation**
    for Layer $v = V, \ldots, 1$ **do**
        # Error Signal in Layer v
        $\delta_v = \phi'(\mathbf{z}_v)^\top \delta_{v+1}^\top \mathbf{W}^{(v)}$
        # Gradient Step
        $\mathbf{W}^{(v)} = \mathbf{W}^{(v)} - \eta \delta_v \mathbf{z}_{v-1}^\top$
    end for
end while

Summary

- Perceptrons cannot separate linearly non-separable problems
- Using combinations and stacking of standard Perceptrons, Multi Layer Perceptrons (MLPs) can approximate any function with one hidden layer
- Gradient descent for MLPs is challenging, mathematically and computationally
- Backpropagation: Efficient Gradient computation

References

C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer US, 2007.

G. Cybenko. Approximations by superpositions of sigmoidal functions. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. 2003.

K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. Adaptive Computation and Machine Learning. The MIT Press, 1 edition, 2012. ISBN 0262018020,9780262018029.

F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, Nov. 1958.

B. Widrow and M. E. Hoff. Adaptive switching circuits. In *1960 IRE WESCON Convention Record, Part 4*, pages 96–104, New York, 1960. IRE.