# What is backpropagation?


Shpresim Sadiku

(Technische Universität Berlin & Zuse Institute Berlin)

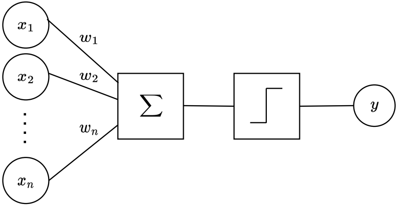

What is ...? Seminar · February 10, 2023

# Outline

- The Perceptron Algorithm
- Perceptron via gradient descent
- Gradients of a Neural Network
- Numerical gradient computation
- Backpropagation algorithm
  - Chain rule and multivariate chain rule
  - Backpropagation through example
  - Formalization of backpropagation
  - Vanishing gradients
  - Choice of nonlinear activation functions
  - Automatic differentiation

# The Perceptron

**Structure:**




- Weighted sum of input features

$$z = \sum_{i=1}^{n} w_i x_i + b = \mathbf{w}^T \mathbf{x} + b$$

- Followed by the sign function

$$y = \text{sign}(z)$$

**Learning task:** Given input data

$$\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(m)} \in \mathbb{R}^n$$

of corresponding labels $t^{(1)}, t^{(2)}, ..., t^{(m)} \in \{-1, 1\}$

- Goal is to learn a collection of parameters $(\mathbf{w}, b)$ such that

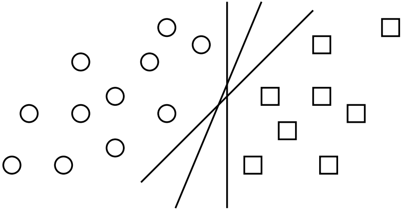
$$\min_{\mathbf{w}, b} \sum_{j=1}^{m} \mathcal{L}(t^j, \mathbf{w}^T \mathbf{x}^j + b)$$

- $\mathcal{L}(\mathbf{w}, b)$ denotes the error function

# The Perceptron

- Predictions of the perceptron for each datapoint

$$\begin{aligned} z^{(j)} &= \mathbf{w}^T \mathbf{x}^{(j)} + b \\ y^{(j)} &= \text{sign}(z^{(j)}) \end{aligned}$$



## Question:

Can all the points be correctly classified

$$\exists(\mathbf{w}, b) : y^{(j)} = t^{(j)}, \forall_{j=1}^{m}?$$

# The Perceptron Algorithm

## Perceptron Algorithm

- Initialize $\mathbf{w} = \mathbf{0}$ and $b = 0$
- Repeat for $j = 1, ..., m$
  - If $\mathbf{x}^{(j)}$ is correctly classified ($y^{(j)} = t^{(j)}$), continue
  - If $\mathbf{x}^{(j)}$ is wrongly classified ($y^{(j)} \neq t^{(j)}$), update

$$\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} + \eta \cdot \mathbf{x}^{(j)} t^{(j)} \\ b &\leftarrow b + \eta \cdot t^{(j)} \end{aligned}$$

  for some learning rate $\eta$
- Until all examples are classified correctly

# Optimization View of Perceptron

## Proposition

The perceptron is equivalent to the gradient descent of the so-called *Hinge Loss*

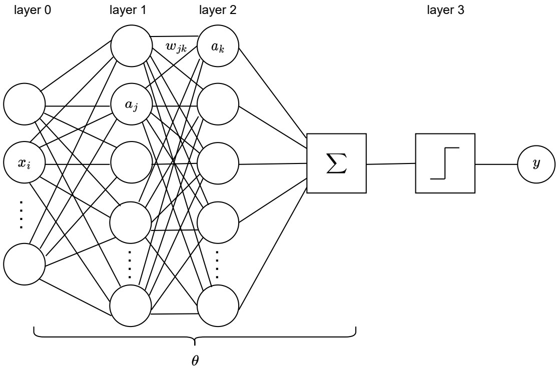
$$\mathcal{L}(\mathbf{w}, b) = \frac{1}{m}\sum_{j=1}^{m} \underbrace{\max(0, -z^{(j)} t^{(j)})}_{\mathcal{L}_j(\mathbf{w}, b)}$$

## Proof.

$$\begin{aligned}
\mathbf{w} - \eta \frac{\partial \mathcal{L}_j}{\partial \mathbf{w}} &= \mathbf{w} - \eta \cdot 1_{-z^{(j)} t^{(j)} > 0} \cdot \left( -\frac{\partial z^{(j)}}{\partial \mathbf{w}} t^{(j)} \right) \\
&= \mathbf{w} - \eta \cdot 1_{y^{(j)} \neq t^{(j)}} \cdot \left( -\frac{\partial z^{(j)}}{\partial \mathbf{w}} t^{(j)} \right) \\
&= \mathbf{w} + \eta \cdot 1_{y^{(j)} \neq t^{(j)}} \cdot \mathbf{x}^{(j)} t^{(j)}
\end{aligned}$$

- Proceed similarly for the parameter $b$

# From Perceptron to Deep Neural Networks




## Idea:

Stack multiple perceptrons together to generalize the formulation where $z$ is the output of a multilayer neural network with parameters $\theta$

$\hookrightarrow$ Updated error function $\mathcal{L}(\theta)$

# Numerical Differentiation

## Question:

How hard is it to compute the gradient of the error function w.r.t. the model parameters

$$\theta = \theta - \eta \frac{\partial \mathcal{L}}{\partial \theta} \text{ ?}$$

## Idea:

Use the definition of the derivative

$$\forall_t : \frac{\partial \mathcal{L}}{\partial \theta_t} = \lim_{\varepsilon \to 0} \frac{\mathcal{L}(\theta + \varepsilon \cdot \delta_t) - \mathcal{L}(\theta)}{\varepsilon}$$

- $\delta_t$ denotes an indicator vector for the parameter $t$

**Properties:**

- Applicable to any error function $\mathcal{L}$
- Re-evaluate the function as many times as there are parameters ($\hookrightarrow$ slow for a large number of parameters)
- Neural networks typically have between $10^3$ and $10^9$ parameters ($\hookrightarrow$ numerical differentiation unfeasible)
- Need to use high-precision due to small $\varepsilon$ and numerator

# Non-convex error function

**Problems:**

- $\mathcal{L}(\theta)$ is non-convex and non-linear
- For complex functions, the computation of $\nabla_\theta \mathcal{L}$ is tricky to be done by hand

**Question:**

Can we do this automatically?

- A general rule to find the weights $\theta$ was not discovered until 1974 (Paul Werbos) / 1985 (LeCun) / 1986 (Rumelhart et al.)

**Idea:**

Need to compute the gradient $\partial\mathcal{L}/\partial w_{jk}$
$\hookrightarrow$ Compute the error at the output, and propagate that back to the neurons in the earlier layers
$\hookrightarrow$ Compute the gradient

# The Chain Rule

- Assume some parameter of interest $\theta_q$ and the output of the network $z$ are linked through a sequence of functions

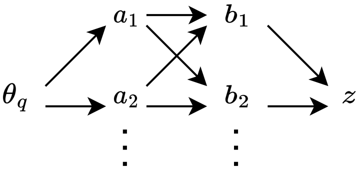
$$\boldsymbol{\theta_q} \longrightarrow \boldsymbol{a} \longrightarrow \boldsymbol{b} \longrightarrow \boldsymbol{z}$$

- Applying the chain rule for derivatives, the derivative w.r.t. the parameter of interest is the product of local derivatives along the path connecting $\theta_q$ to $z$

$$\frac{\partial z}{\partial \theta_q} = \frac{\partial a}{\partial \theta_q} \frac{\partial b}{\partial a} \frac{\partial z}{\partial b}$$

# The Multivariate Chain Rule

- The parameter of interest may be linked to the output of the network via multiple paths, formed by all neurons in layers between $\theta_q$ and $z$

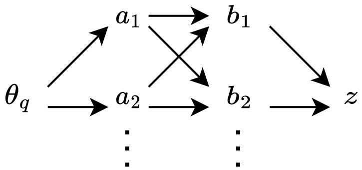



- Multivariate scenario $\Rightarrow$ the chain rule enumerates all the paths between $\theta_q$ and $z$

$$\frac{\partial z}{\partial \theta_q} = \sum_i \sum_j \frac{\partial a_i}{\partial \theta_q} \frac{\partial b_j}{\partial a_j} \frac{\partial z}{\partial b_j}$$

  where $\sum_i$ and $\sum_j$ run over all indices of the nodes in the corresponding layers
- Nested sum - complexity grows exponentially with the number of layers

# Factor Structure in the Multivariate Chain Rule

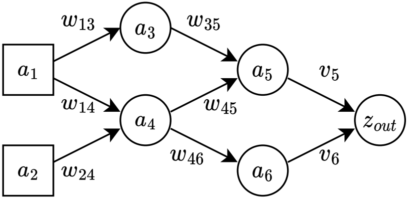



- Re-write the computation - perform the summing operation incrementally
- Re-use intermediate computation for different paths and parameters for which we would like to compute the gradient

$$\frac{\partial z}{\partial \theta_q} = \sum_i \frac{\partial a_i}{\partial \theta_q} \underbrace{\sum_j \frac{\partial b_j}{\partial a_j} \underbrace{\frac{\partial z}{\partial b_j}}_{\delta_j}}_{\delta_i}$$

- The resulting gradient computation w.r.t. all parameters in the network is linear with the size of the network ($\Rightarrow$ fast!)

# Backpropagation through Example

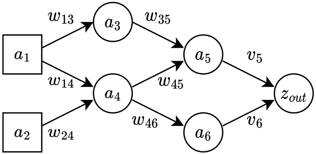



**Forward pass:**

$$a_1 = x_1$$
$$a_2 = x_2$$

$$z_3 = a_1 w_{13} \qquad a_3 = \tanh(z_3)$$
$$z_4 = a_1 w_{14} + a_2 w_{24} \qquad a_4 = \tanh(z_4)$$
$$z_5 = a_3 w_{35} + a_4 w_{45} \qquad a_5 = \tanh(z_5)$$
$$z_6 = a_4 w_{46} \qquad a_6 = \tanh(z_6)$$
$$z_{out} = a_5 v_5 + a_6 v_6$$
$$\mathcal{L} = \max(0, -z_{out} \cdot t)$$

# Backpropagation through Example



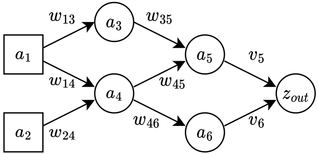

$$
\begin{aligned}
z_3 &= a_1 w_{13} \\
z_4 &= a_1 w_{14} + a_2 w_{24} \\
z_5 &= a_3 w_{35} + a_4 w_{45} \\
z_6 &= a_4 w_{46} \\
z_{out} &= a_5 v_5 + a_6 v_6 \\
\mathcal{L} &= \max(0, -z_{out} \cdot t)
\end{aligned}
$$

$$
\begin{aligned}
a_1 &= x_1 \\
a_2 &= x_2 \\
a_3 &= \tanh(z_3) \\
a_4 &= \tanh(z_4) \\
a_5 &= \tanh(z_5) \\
a_6 &= \tanh(z_6)
\end{aligned}
$$

## Backward pass:

$$
\begin{aligned}
\delta_{out} \quad = \quad & \frac{\partial \mathcal{L}}{\partial z_{out}} = 1_{\{-z_{out} \cdot t > 0\}} \cdot (-t) \\
& \frac{\partial \mathcal{L}}{\partial v_6} = \frac{\partial z_{out}}{\partial v_6} \frac{\partial \mathcal{L}}{\partial z_{out}} = a_6 \cdot \delta_{out} \\
& \frac{\partial \mathcal{L}}{\partial v_5} = \frac{\partial z_{out}}{\partial v_5} \frac{\partial \mathcal{L}}{\partial z_{out}} = a_5 \cdot \delta_{out}
\end{aligned}
$$

# Backpropagation through Example



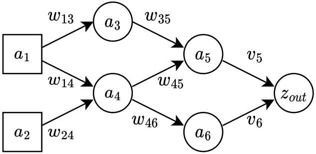

$$
\begin{aligned}
z_3 &= a_1 w_{13} \\
z_4 &= a_1 w_{14} + a_2 w_{24} \\
z_5 &= a_3 w_{35} + a_4 w_{45} \\
z_6 &= a_4 w_{46} \\
z_{out} &= a_5 v_5 + a_6 v_6 \\
\mathcal{L} &= \max(0, -z_{out} \cdot t)
\end{aligned}
$$

$$
\begin{aligned}
a_1 &= x_1 \\
a_2 &= x_2 \\
a_3 &= \tanh(z_3) \\
a_4 &= \tanh(z_4) \\
a_5 &= \tanh(z_5) \\
a_6 &= \tanh(z_6)
\end{aligned}
$$

## Backward pass:

$$
\begin{aligned}
\delta_{out} &= \frac{\partial \mathcal{L}}{\partial z_{out}} = 1_{\{-z_{out} \cdot t > 0\}} \cdot (-t) \\
\delta_6 &= \frac{\partial \mathcal{L}}{\partial a_6} = \frac{\partial z_{out}}{\partial a_6} \frac{\partial \mathcal{L}}{\partial z_{out}} = v_6 \cdot \delta_{out} \\
\delta_5 &= \frac{\partial \mathcal{L}}{\partial a_5} = \frac{\partial z_{out}}{\partial a_5} \frac{\partial \mathcal{L}}{\partial z_{out}} = v_5 \cdot \delta_{out}
\end{aligned}
$$

# Backpropagation through Example



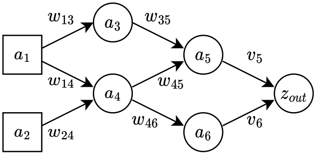

$$
\begin{aligned}
z_3 &= a_1 w_{13} \\
z_4 &= a_1 w_{14} + a_2 w_{24} \\
z_5 &= a_3 w_{35} + a_4 w_{45} \\
z_6 &= a_4 w_{46} \\
z_{out} &= a_5 v_5 + a_6 v_6 \\
\mathcal{L} &= \max(0, -z_{out} \cdot t)
\end{aligned}
$$

$$
\begin{aligned}
a_1 &= x_1 \\
a_2 &= x_2 \\
a_3 &= \tanh(z_3) \\
a_4 &= \tanh(z_4) \\
a_5 &= \tanh(z_5) \\
a_6 &= \tanh(z_6)
\end{aligned}
$$

**Backward pass:**

$$
\begin{aligned}
\delta_6 &= \frac{\partial \mathcal{L}}{\partial a_6} = \frac{\partial z_{out}}{\partial a_6} \frac{\partial \mathcal{L}}{\partial z_{out}} = v_6 \cdot \delta_{out} \\
\delta_5 &= \frac{\partial \mathcal{L}}{\partial a_5} = \frac{\partial z_{out}}{\partial a_5} \frac{\partial \mathcal{L}}{\partial z_{out}} = v_5 \cdot \delta_{out} \\
& \frac{\partial \mathcal{L}}{\partial w_{46}} = \frac{\partial z_6}{\partial w_{46}} \frac{\partial a_6}{\partial z_6} \frac{\partial \mathcal{L}}{\partial a_6} = a_4 \cdot \tanh'(z_6) \cdot \delta_6 \\
& \frac{\partial \mathcal{L}}{\partial w_{45}} = \frac{\partial z_5}{\partial w_{45}} \frac{\partial a_5}{\partial z_5} \frac{\partial \mathcal{L}}{\partial a_5} = a_4 \cdot \tanh'(z_5) \cdot \delta_5 \\
& \frac{\partial \mathcal{L}}{\partial w_{35}} = \frac{\partial z_5}{\partial w_{35}} \frac{\partial a_5}{\partial z_5} \frac{\partial \mathcal{L}}{\partial a_3} = a_5 \cdot \tanh'(z_5) \cdot \delta_5
\end{aligned}
$$

# Backpropagation through Example



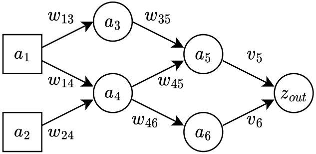

$$\begin{aligned} z_3 &= a_1 w_{13} \\ z_4 &= a_1 w_{14} + a_2 w_{24} \\ z_5 &= a_3 w_{35} + a_4 w_{45} \\ z_6 &= a_4 w_{46} \\ z_{out} &= a_5 v_5 + a_6 v_6 \\ \mathcal{L} &= \max(0, -z_{out} \cdot t) \end{aligned}$$

$$\begin{aligned} a_1 &= x_1 \\ a_2 &= x_2 \\ a_3 &= \tanh(z_3) \\ a_4 &= \tanh(z_4) \\ a_5 &= \tanh(z_5) \\ a_6 &= \tanh(z_6) \end{aligned}$$

## Backward pass:

$$\begin{aligned} \delta_6 &= \frac{\partial \mathcal{L}}{\partial a_6} = \frac{\partial z_{out}}{\partial a_6} \frac{\partial \mathcal{L}}{\partial z_{out}} = v_6 \cdot \delta_{out} \\ \delta_5 &= \frac{\partial \mathcal{L}}{\partial a_5} = \frac{\partial z_{out}}{\partial a_5} \frac{\partial \mathcal{L}}{\partial z_{out}} = v_5 \cdot \delta_{out} \\ \delta_4 &= \frac{\partial \mathcal{L}}{\partial a_4} = \frac{\partial z_6}{\partial a_4} \frac{\partial a_6}{\partial z_6} \frac{\partial \mathcal{L}}{\partial a_6} + \frac{\partial z_5}{\partial a_4} \frac{\partial a_5}{\partial z_5} \frac{\partial \mathcal{L}}{\partial a_5} = w_{46} \cdot \tanh'(z_6) \cdot \delta_6 + w_{45} \cdot \tanh'(z_5) \cdot \delta_5 \\ \delta_3 &= \frac{\partial \mathcal{L}}{\partial a_3} = \frac{\partial z_5}{\partial a_3} \frac{\partial a_5}{\partial z_5} \frac{\partial \mathcal{L}}{\partial a_5} = w_{35} \cdot \tanh'(z_5) \cdot \delta_5 \end{aligned}$$

# Backpropagation through Example




$$\begin{aligned} z_3 &= a_1 w_{13} \\ z_4 &= a_1 w_{14} + a_2 w_{24} \\ z_5 &= a_3 w_{35} + a_4 w_{45} \\ z_6 &= a_4 w_{46} \\ z_{out} &= a_5 v_5 + a_6 v_6 \\ \mathcal{L} &= \max(0, -z_{out} \cdot t) \end{aligned}$$

$$\begin{aligned} a_1 &= x_1 \\ a_2 &= x_2 \\ a_3 &= \tanh(z_3) \\ a_4 &= \tanh(z_4) \\ a_5 &= \tanh(z_5) \\ a_6 &= \tanh(z_6) \end{aligned}$$

## Backward pass:

$$\begin{aligned} \delta_4 &= \frac{\partial \mathcal{L}}{\partial a_4} = \frac{\partial z_6}{\partial a_4}\frac{\partial a_6}{\partial z_6}\frac{\partial \mathcal{L}}{\partial a_6} + \frac{\partial z_5}{\partial a_4}\frac{\partial a_5}{\partial z_5}\frac{\partial \mathcal{L}}{\partial a_5} = w_{46} \cdot \tanh'(z_6) \cdot \delta_6 + w_{45} \cdot \tanh'(z_5) \cdot \delta_5 \\ \delta_3 &= \frac{\partial \mathcal{L}}{\partial a_3} = \frac{\partial z_5}{\partial a_3}\frac{\partial a_5}{\partial z_5}\frac{\partial \mathcal{L}}{\partial a_5} = w_{35} \cdot \tanh'(z_5) \cdot \delta_5 \\ & \frac{\partial \mathcal{L}}{\partial w_{24}} = \frac{\partial z_4}{\partial w_{24}}\frac{\partial a_4}{\partial z_4}\frac{\partial \mathcal{L}}{\partial a_4} = a_2 \cdot \tanh'(z_4) \cdot \delta_4 \\ & \frac{\partial \mathcal{L}}{\partial w_{14}} = \frac{\partial z_4}{\partial w_{14}}\frac{\partial a_4}{\partial z_4}\frac{\partial \mathcal{L}}{\partial a_4} = a_1 \cdot \tanh'(z_4) \cdot \delta_4 \\ & \frac{\partial \mathcal{L}}{\partial w_{13}} = \frac{\partial z_3}{\partial w_{13}}\frac{\partial a_3}{\partial z_3}\frac{\partial \mathcal{L}}{\partial a_3} = a_1 \cdot \tanh'(z_3) \cdot \delta_3 \end{aligned}$$

# Formalization for a Standard Neural Network

- Propagate the gradient of the error from layer to layer using the chain rule

$$\underbrace{\frac{\partial \mathcal{L}}{\partial a_j}}_{\delta_j} = \sum_k \underbrace{\frac{\partial a_k}{\partial a_j}}_{w_{jk} g'(z_k)} \cdot \underbrace{\frac{\partial \mathcal{L}}{\partial a_k}}_{\delta_k}$$

- Extract gradients w.r.t. parameters at each layer as

$$\frac{\partial \mathcal{L}}{\partial w_{jk}} = \sum_k \underbrace{\frac{\partial a_k}{\partial w_{jk}}}_{a_j g'(z_k)} \cdot \underbrace{\frac{\partial \mathcal{L}}{\partial a_k}}_{\delta_k}$$

- Re-write equations as matrix-vector products

$$\begin{aligned} \delta^{(l-1)} &= W^{(l-1,l)} \cdot (g'(\mathbf{z}^{(l)}) \odot \delta^{(l)}) \\ \frac{\partial \mathcal{L}}{\partial W^{(l-1,l)}} &= \mathbf{a} \cdot (g'(\mathbf{z}^{(l)}) \odot \delta^{(l)})^T \end{aligned}$$

# Vanishing gradient

- In general

$$\partial\mathcal{L}/\partial W^{(l-1,l)} \gg \partial\mathcal{L}/\partial W^{(l-2,l-1)}$$

$\Rightarrow$ the more left you get in the network, the more the gradient vanishes

- tanh has gradients in the range $(0, 1]$
  $\Rightarrow$ in an $n-$layer network the gradient decreases exponentially with $n$

Ways to circumvent vanishing gradients

- Use many labeled data (e.g., well possible for images)
- Train "longer" (possible with GPUs)
- Better weight initialization (e.g., Xavier/Glorot)
- Regularize with “dropout“
- Other activation functions: ReLU

# Choice of Nonlinear Activation Function

Choose the nonlinear function such that

- Its gradient is defined (almost) everywhere
- A significant portion of the input domain has a non-zero gradient
- Its gradient is informative, i.e., indicate decrease/increase of the activation function

Commonly used activation functions:

- **Sigmoid:** $g(z) = \exp(z)/(1 + \exp(z))$
- **tanh:** $g(z) = \tanh(z)$
- **ReLU:** $g(z) = \max(0, z)$

Problematic activation functions:

- $g(z) = \max(0, z - 100)$
- $g(z) = 1_{z>0}$
- $g(z) = \sin(100 \cdot z)$

# Automatic Differentiation

- Automatically generate backpropagation equations from the forward equations
- Automatic differentiation widely available in deep learning libraries (PyTorch, Tensorflow, JAX, etc.)

**Consequences:**

- No need to do backpropagation, just program the forward pass
  $\hookrightarrow$ backward pass comes for free
- Motivated the development of neural networks that are way more complex, and with much more heterogeneous structures (e.g. ResNet, Yolo, transformers, etc.)
- In few cases, it is still useful to express the gradient analytically (e.g. to analyze theoretically the stability of a gradient descent procedure)

# Training Neural Networks

## Basic gradient descent algorithm

- Initialize $\theta$ at random
- Repeat for $T$ steps
  - Compute the forward pass
  - Use backpropagation to extract $\partial\mathcal{L}/\partial\theta$
  - Perform a gradient step

$$\theta = \theta - \gamma \frac{\partial\mathcal{L}}{\partial\theta}$$

    for some learning rate $\gamma$

# Summary

- Gradient descent to minimize the error of a classifier (e.g. Perceptron, neural network + backpropagation)
- Error backpropagation provides a computationally efficient way of computing the gradient compared to the formula for numerical differentiation
- Error backpropagation is a direct application of the multivariate chain rule, where the different terms can be factored due to the structure of the neural network graph
- Use certain techniques to circumvent vanishing gradients
- No need to program error backpropagation manually, use automatic differentiation techniques instead

# THANK YOU!

Slides available at:

`www.shpresimsadiku.com`

Check related information on Twitter at:

`@shpresimsadiku`