

# Neural Networks I

---

Machine Learning 1 — Lecture 7

30<sup>th</sup> April 2023

Robert Peharz

Institute of Theoretical Computer Science  
Graz University of Technology

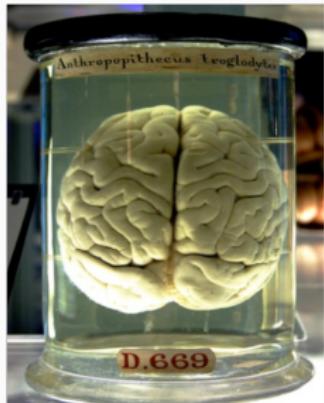
## **Brief Appetizer Video**

<https://www.youtube.com/watch?v=Suevq-kZdIw>

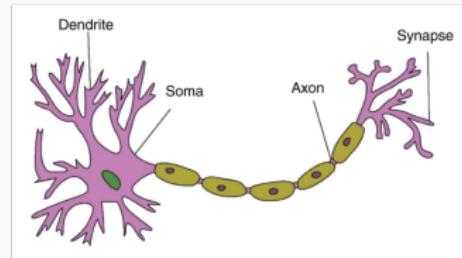
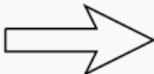
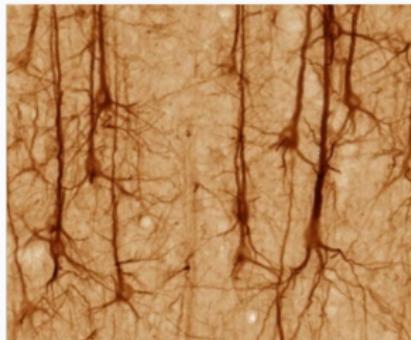
## **Excellent Tutorial by 3Blue1Brown [optional, recommended]**

[https://www.youtube.com/playlist?list=PLZHQB0WTQDNU6R1\\_67000Dx\\_ZCJB-3pi](https://www.youtube.com/playlist?list=PLZHQB0WTQDNU6R1_67000Dx_ZCJB-3pi)

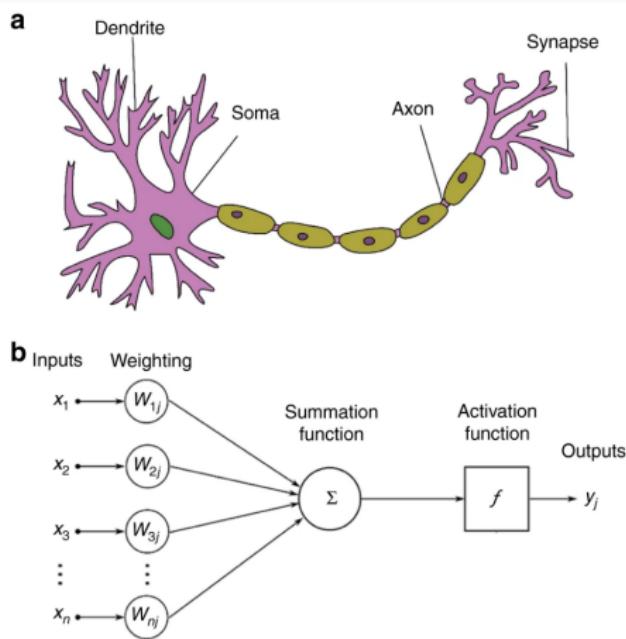
# Biological Neural Networks



- The brain: still the only “truly” intelligent device on earth
- A biological neural network of 100 billion **neurons**, connected via 100 trillions of **synapses**
- Extremely complex and parallel



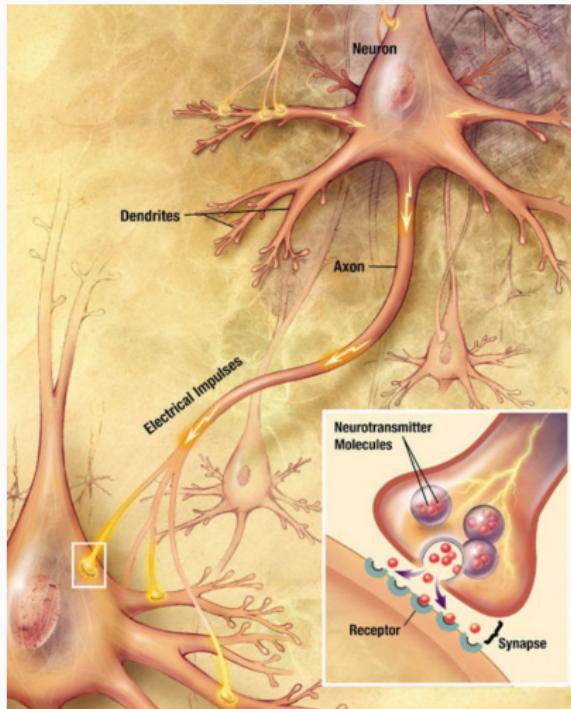
# Biological vs. Artificial Neuron



- **dendrite**: receive input spikes from previous neurons
- **soma**: accumulate input, fires/spikes if excited enough ( $i$ )
- **axon**: long way transmission of spikes
- **synapses**: connection to next neurons (connection strength)
- **In ANNs:**
  - accumulation is  $\approx \sum$
  - spiking is  $\approx$  non-linear activation function
  - synapse strength is  $\approx$  weights

Image: Zhang et al., 2019

# Synapses

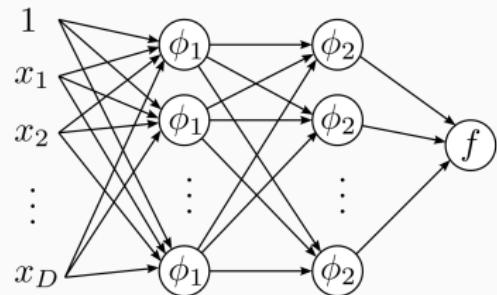


- Synapses are essential for the communication between neurons
- They are basically a link transmitting spikes from the pre-synaptic to the post-synaptic neuron
- **Synaptic strength** can be adapted; an essential mechanism for learning

# Biological vs. Artificial Neural Networks



Biological neural network



Artificial neural network

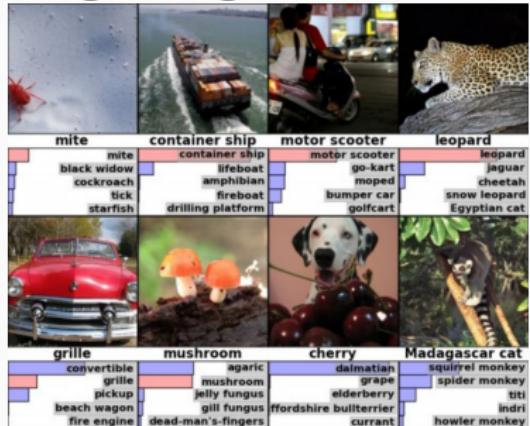
- Artificial neural networks (ANNs): learn from biology
- Two way street: Use ANNs to understand biological neural systems better
  - Develop biologically accurate neuron models
  - Simulations of biological neural networks
  - Spiking ANNs
- For ML purposes: **powerful function approximators**

# Brief History

- 1943 – McCulloch and Pitts – *simple electrical circuit for connectionism*
- 1949 – Donald Hebb – *Hebbian learning*
- 1958 – Frank Rosenblatt – *Perceptron*
- ≈ 1970-1980 – “AI Winter”
- 1982 – Jon Hopfield – *Hopfield net*
- 1986 – Rumelhart, Hinton and Williams – *Backpropagation* (re-invented from 1960, actually autodiff/chain rule of calculus)
- 1989 – Yann LeCun – *Convolutional Neural Networks* (image models)
- 1997 – Sepp Hochreiter and Jürgen Schmidhuber – *Long Short Term Memory Networks*
- ≈ 1995–2006 – Neural networks were not popular (lighter models were preferred, e.g. support vector machines)
- 2006 – Hinton, Osindero, Teh – *Deep Learning, Deep Belief Networks*
- 2012 – Krizhevsky, Sutskever, Hinton: New state of the art on ImageNet
- After 2012 – extremely fast growth in popularity due to their success in real-world computer vision problems, natural language processing, etc.

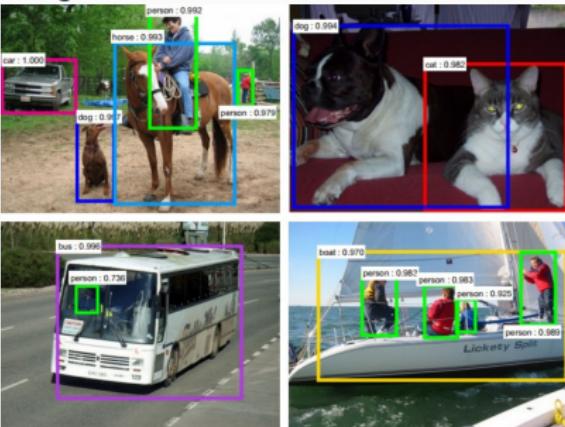
# Deep Learning in Computer Vision

## Image recognition



Krizhevsky et al., 2012

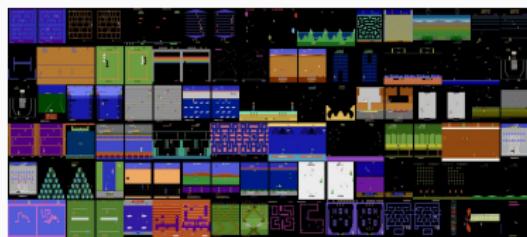
## Object Detection



Ren et al., 2015

# Deep Learning in Game Playing

## Atari Games



Mnih et al., 2015

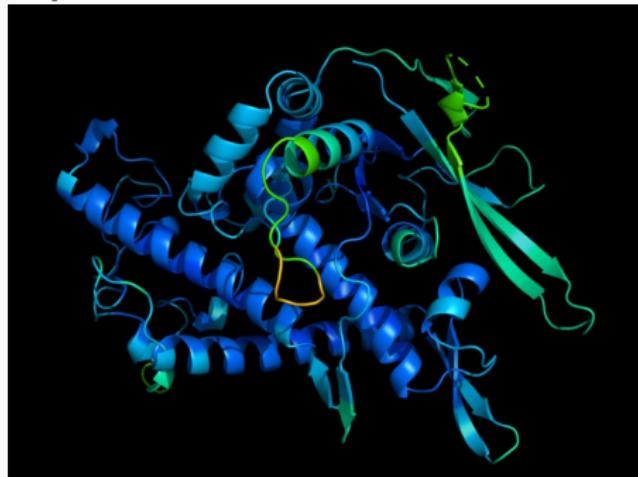
## Board Games



Silver et al., 2017

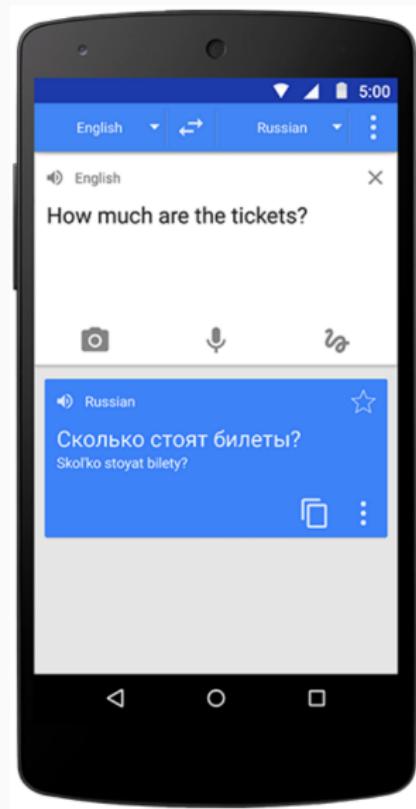
# Deep Learning for Protein Folding

Alpha Fold



Senior et al., 2020

# Deep Learning for Machine Translation



- **Long-short term memories**  
[Hochreiter and Schmidhuber, 1997]
- **Transformers** [Vaswani et al., 2017]

## Neural Networks: Basics

---

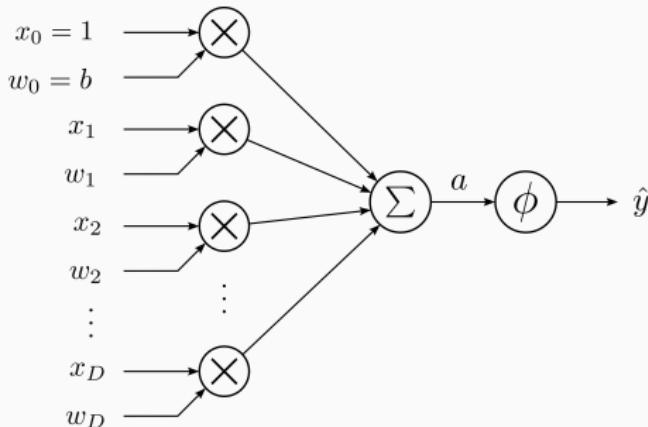
# Neuron, Unit

The **neuron (unit)** is the basic building block of neural networks:

$$\hat{y} = \phi\left(\underbrace{\sum_{i=0}^D w_i x_i}_a\right)$$

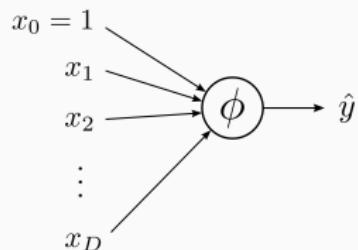
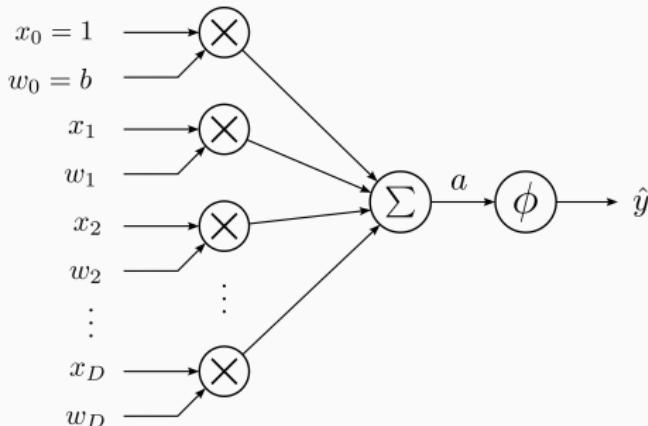
- $D$ : number of inputs
- $x_i$ : inputs (features or outputs from previous neurons)
- $w_i$ : weights
- $w_0$ : bias term ( $x_0 \equiv 1$ )
- $a$ : **activation**  $\sum_{i=0}^D w_i x_i$
- $\phi$ : non-linear **activation function** (also called **non-linearity**)
- $\hat{y}$ : output  $\phi(a)$
- **Affine function of  $x$ , followed by non-linearity  $\phi$**

# Neuron as Computational Graph



**computational graph:** directed acyclic graph whose leaves are inputs, roots are outputs, and internal nodes are computational operations; note that weights  $w_d$  can be seen as inputs as well

# Neuron as Computational Graph

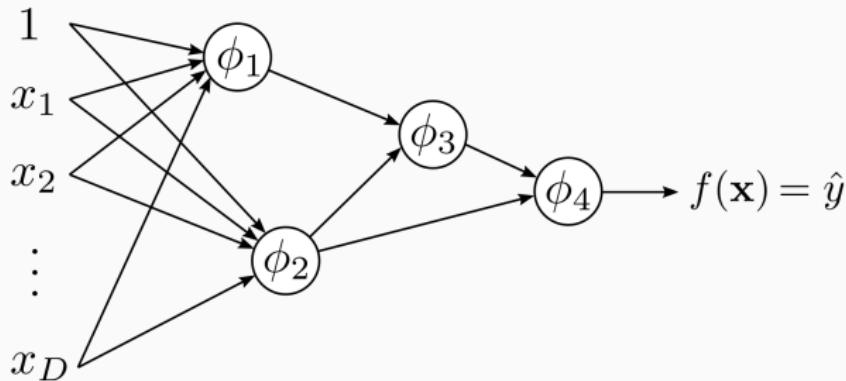


**computational graph:** directed acyclic graph whose leaves are inputs, roots are outputs, and internal nodes are computational operations; note that weights  $w_d$  can be seen as inputs as well

**simplified graph:** multiplication with weights  $w_d$  absorbed into edges, activation  $a$  absorbed into  $\phi$  node

## Neural Networks: Network of Neurons

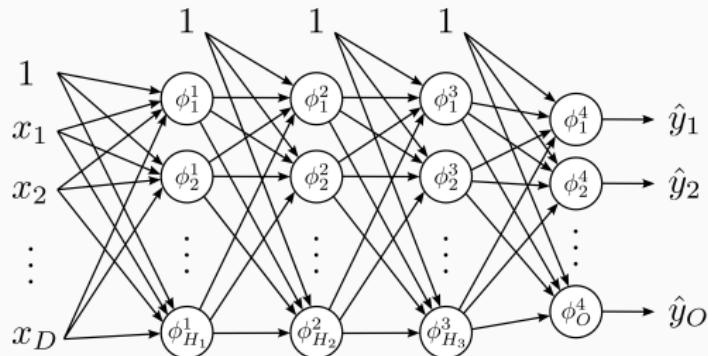
- Like in biological neural nets, ANNs are networks of neurons
- Each neuron is simple, but when combined, they can represent complicated functions  $f(\mathbf{x})$ , to be used for
  - Classification
  - Regression
  - Advanced applications
- Training by optimizing the weights (discussed later)



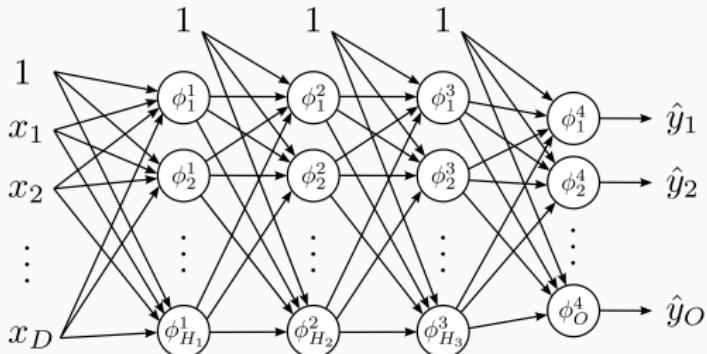
A regular architecture is the **multilayer perceptron (MLPs)**:

- consists of  $L + 1$  **layers** of neurons
- each neuron has all neurons of the previous layer as input
- $L + 1^{\text{st}}$  layer is the **output layer (output neurons)**
- layers  $1 \dots L$  are often called **hidden layers (hidden neurons)**
- inputs  $\mathbf{x} = (1, x_1, \dots, x_D)^T$  interpreted as Layer 0 (**input layer**)

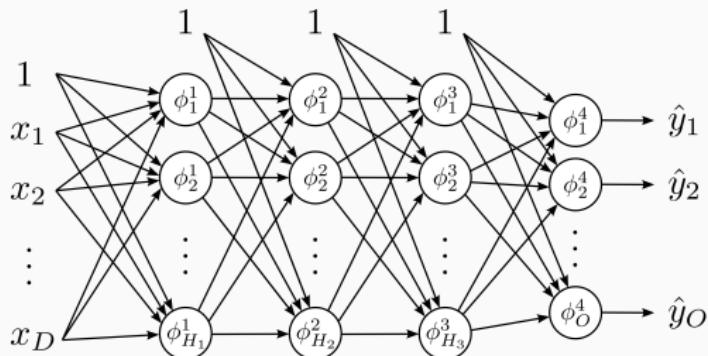
Example for  $L = 3$



- $D$ : number of inputs
- $O$ : number of outputs
- $H_l$ : number of neurons in  $l^{\text{th}}$  hidden layer
- $\phi_k^l$ : activation function of  $k^{\text{th}}$  neuron in  $l^{\text{th}}$  layer
- usually the same activation function across each layer, i.e.  $\phi_k^l = \phi^l$
- $L, H_1, \dots, H_L$  are **hyper-parameters** of the architecture



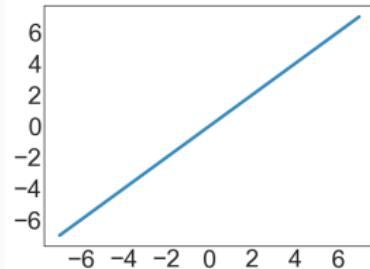
- $w_{i,j}^l$ : weight for connection between  $i^{\text{th}}$  neuron in  $l - 1^{\text{st}}$  layer to  $j^{\text{th}}$  neuron in  $l^{\text{th}}$  layer
- let  $H_0 := D$  and  $h_i^0 := x_i$
- let  $h_0^l \equiv 1$  (dummy features for biases)
- neuron activation  $a_j^l = \sum_{i=0}^{H_{l-1}} w_{i,j}^l h_i^{l-1}$
- neuron output  $\phi_j^l(a_j^l)$
- MLP output  $\hat{y}_i := h_i^{L+1}$



# Common Activation Functions $\phi$

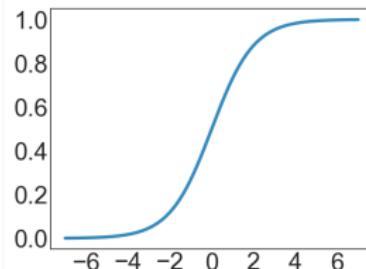
**linear** (identity function)

$$\phi(x) = x$$



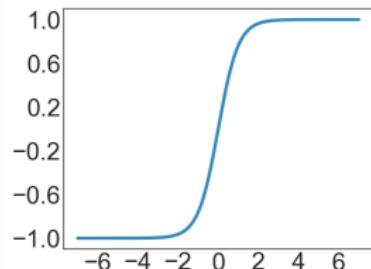
**sigmoid** (logistic func.)

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



**hyperbolic tangent**

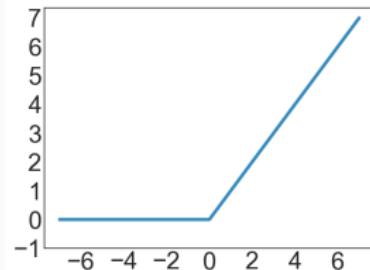
$$\phi(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



**rectified linear unit**

(ReLU)

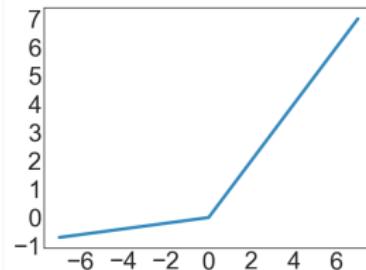
$$\phi(x) = \max(x, 0)$$



**leaky ReLU**

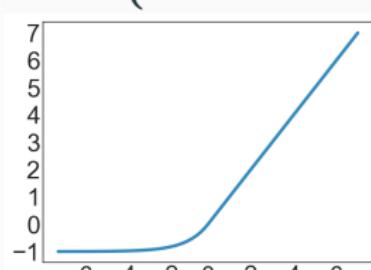
$$\phi(x) = \max(x, ax)$$

(e.g.  $a = 0.1$ )



**exponential linear unit**

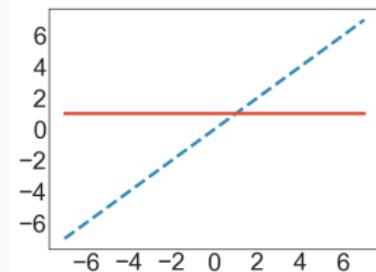
$$\phi(x) = \begin{cases} x & x \geq 0 \\ a(e^x - 1) & x < 0 \end{cases}$$



# Derivatives of Common Activation Functions $\phi$

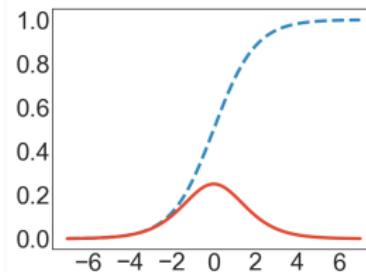
**linear** (identity function)

$$\phi'(x) = 1$$



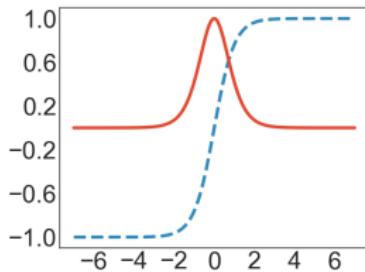
**sigmoid** (logistic func.)

$$\phi'(x) = \sigma(x)\sigma(-x)$$



**hyperbolic tangent**

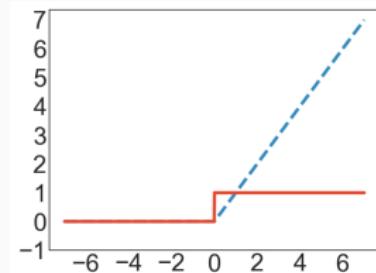
$$\phi'(x) = 1 - \tanh^2(x)$$



**rectified linear unit**

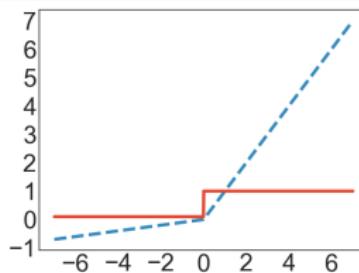
**(ReLU)**

$$\phi'(x) = \mathbb{1}(x \geq 0)$$



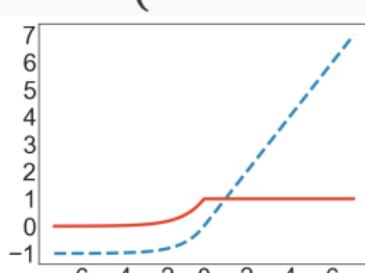
**leaky ReLU**

$$\phi'(x) = \mathbb{1}(x \geq 0) + a\mathbb{1}(x < 0)$$



**exponential linear unit**

$$\phi'(x) = \begin{cases} 1 & x \geq 0 \\ ae^x & x < 0 \end{cases}$$



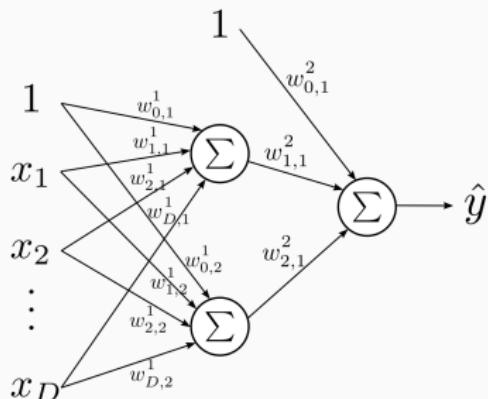
**Why do we need non-linear activation functions  $\phi$ ?**

## Why do we need non-linear activation functions $\phi$ ?

1. Linear functions often not adequate for real-world datasets.
2. Plug-and-play philosophy of ANNs: Construct complicated functions out of simple ones.

**This does not work with linear neurons!**

Assume **linear units**, i.e. using  $\phi(x) = x$ :

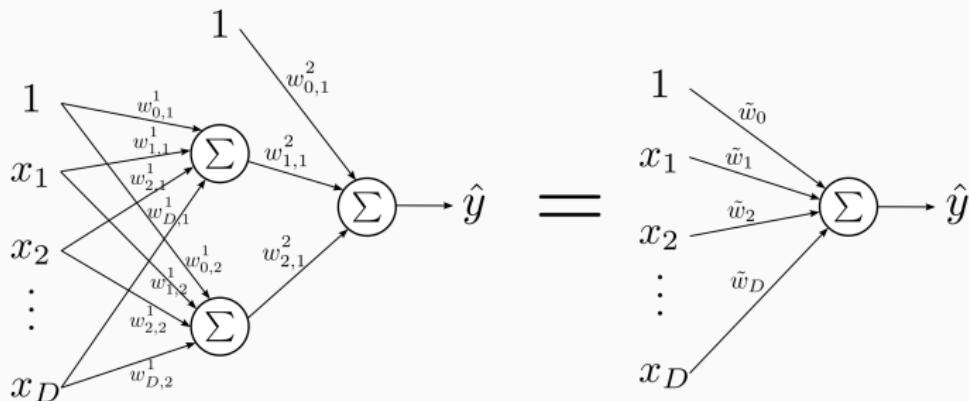


$$\hat{y} = w_{0,1}^2 + \sum_{i=1}^2 w_{i,1}^2 \left( \sum_{j=0}^D w_{j,i}^1 x_j \right)$$

# Neural Network of Linear Units

Example

Assume **linear units**, i.e. using  $\phi(x) = x$ :



$$\hat{y} = w_{0,1}^2 + \sum_{i=1}^2 w_{i,1}^2 \left( \sum_{j=0}^D w_{j,i}^1 x_j \right) = \sum_{j=0}^D \tilde{w}_j x_j$$

$$\tilde{w}_0 = w_{0,1}^2 + w_{0,1}^1 w_{1,1}^2 + w_{0,2}^1 w_{2,1}^2$$

$$\tilde{w}_j = w_{j,1}^1 w_{1,1}^2 + w_{j,2}^1 w_{2,1}^2$$

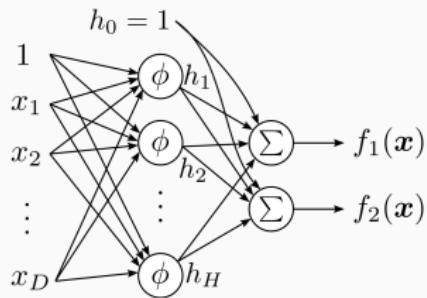
**No matter how many linear units, or how they are connected:  
Neural network of linear units = single linear unit!**

**No matter how many linear units, or how they are connected:  
Neural network of linear units = single linear unit!**

**Does non-linearity save us?**

# Universal Approximation Theorem

Consider **single-layer neural nets**, i.e. MLPs with  $L = 1$ . Further assume that all hidden units have the same non-linearity  $\phi$  and that all outputs are linear units.



**Theorem**

[Cybenko (1989), Pinkus (1999)]

Any continuous function  $f: \mathbb{R}^D \mapsto \mathbb{R}^O$  can be arbitrarily well approximated (on any compact set) by a single-layer neural network, for a wide range of non-linearities  $\phi$  (specifically,  $\phi$  just needs to be non-polynomial).

A **single-layer net** is already a **universal approximator!**

Why should we then even consider MLPs with more than one hidden layer?

# Expressive Efficiency

- **Shallow networks** (1 hidden layer)
  - **universal approximators**
  - However, some function classes require **exponentially** many neurons, in the number of inputs
- **Deep networks** (many hidden layers)
  - can typically represent the same function classes with **only polynomial size**
  - This phenomenon is called **expressive efficiency** – Deep networks can represent non-linear function more efficiently than shallow ones
- Classical example: **parity function** – requires exponential size in shallow networks, but only polynomial size in deep networks

## Expressive Efficiency – Intuition

- Non-linearities  $\phi$  are “simple”, like sigmoid, ReLU, etc.
- Real-world problems like “predict apple/pear from an image” are “highly non-linear”
- Cascading non-linear layers improves their flexibility to express non-linear functions
- Lower layers tend to extract low-level features, while higher levels tend to learn more abstract features

## Deep Vs. Shallow Networks

- **Traditional downside** of deep networks: harder to train
- **“Neural network winter” (~1990-2005):** “Neural networks with more than two layers cannot reliably be trained.”
- **Deep learning (2006–):** Tricks of the trade to train deep networks
- Neural networks can nowadays be scaled to hundreds and thousands of layers

# From Linear Models to Neural Networks

- In **Lecture 4**, we introduced **linear models**:

$$\hat{y} = \sum_{i=0}^D w_i x_i$$

- In **Lecture 5**, we generalized them by considering fixed **non-linear** features:

$$\hat{y} = \sum_{i=0}^H w_i \phi_i(\mathbf{x})$$

- Single-layer neural nets can be seen as linear models (output layer) with **learnable** non-linear features (hidden layer):

$$\hat{y} = \sum_{j=0}^H w_{j,1}^2 \phi \left( \sum_{i=0}^D w_{i,j}^1 x_i \right)$$

- **Deep neural networks = stacked feature extractors**

# Automatic Differentiation

---

Neural networks are powerful function approximators, with hundreds to hundreds of billions of parameters.

**How do we learn these parameters?**

Neural networks are powerful function approximators, with hundreds to hundreds of billions of parameters.

**How do we learn these parameters?**

**Gradient descent!**

## Training Setup

- We collect all network parameters in one vector  $\theta$
- Let  $\hat{y} = f(\mathbf{x}, \theta)$  be the network's output for input  $\mathbf{x}$  and parameters  $\theta$
- Training loss:  $\mathcal{L}_{train}(\theta) = \frac{1}{N} \sum_{i=1}^N \ell(y^{(i)}, f(\mathbf{x}^{(i)}, \theta))$
- $\ell$  is some **sample-wise loss**, e.g. quadratic loss for regression, or cross-entropy for classification
- Minimize with gradient descent, using  $\nabla_{\theta} \mathcal{L}_{train}$
- However, the structure of neural nets is complicated...
  - How to compute the gradient? 

## A Little Simplification

$$\mathcal{L}_{train}(\theta) = \frac{1}{N} \sum_{i=1}^N \ell(y^{(i)}, f(\mathbf{x}^{(i)}, \theta))$$

First simplification: gradient of sum is sum of gradients:

$$\nabla_{\theta} \mathcal{L}_{train}(\theta) = \nabla_{\theta} \frac{1}{N} \sum_{i=1}^N \ell(y^{(i)}, f(\mathbf{x}^{(i)}, \theta)) = \frac{1}{N} \sum_{i=1}^N \nabla_{\theta} \ell(y^{(i)}, f(\mathbf{x}^{(i)}, \theta))$$

Thus, we just need to understand how to **compute the gradient for single sample  $\mathbf{x}, y$ :**

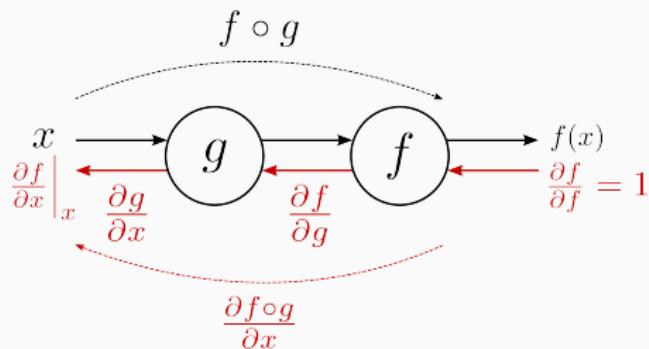
$$\nabla_{\theta} \ell(y, f(\mathbf{x}, \theta))$$

# Automatic Differentiation in Computational Graphs

- A neural network  $f(\mathbf{x}, \boldsymbol{\theta})$  is really just a special case of a **computational graph** with  $\mathbf{x}$  and  $\boldsymbol{\theta}$  as inputs
- **Automatic Differentiation (AD)** is a family of techniques to compute the partial derivatives (gradient) in computations graphs—thus, AD is immediately applicable to neural nets
- In neural networks a special case of AD, namely **reverse mode AD** is usually called **backpropagation of error (backprop)**
- Backprop is often attributed to Rumelhart, Hinton and Williams (1986), but actually also they just rediscovered the method for neural nets

# Automatic Differentiation = Chain Rule of Calculus

- Recall that the chain rule says:  $\frac{\partial f \circ g}{\partial x} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial x}$
- This can be interpreted as **message passing** on a (chain-shaped) **computational graph**:

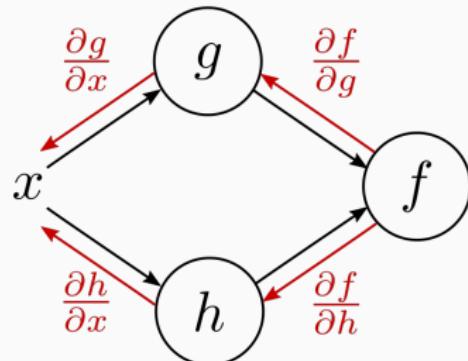


- Backwards pass starts with the trivial derivative  $\frac{\partial f}{\partial f} = 1$
- Recursion: if derivatives of a node's output is known, the derivative of the node itself is just a multiplicative update with a **local derivative** (usually easy, by construction)

# Automatic Differentiation = Chain Rule of Calculus

- More generally, we need a chain rule in multiple dimensions
- Consider a function of the form  $f(x) = f(g(x), h(x))$  which is a composition of a  $\mathbb{R} \mapsto \mathbb{R}^2$  ( $g, h$ ) and a  $\mathbb{R}^2 \mapsto \mathbb{R}$  ( $f$ ) function
- Derivatives of several output branches simply add up:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial x} + \frac{\partial f}{\partial h} \frac{\partial h}{\partial x}$$



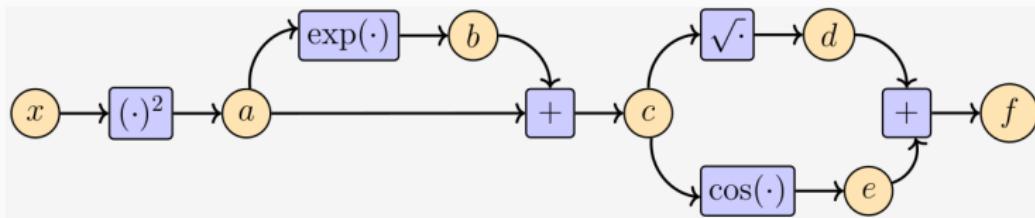
- In general, if  $f(x) = f(g_1(x), g_2(x), \dots, g_K(x))$  then

$$\frac{\partial f}{\partial x} = \sum_{k=1}^K \frac{\partial f}{\partial g_k} \frac{\partial g_k}{\partial x}$$

# Computational Graph

# Example

- Consider following computational graph (Deisenroth et al.):



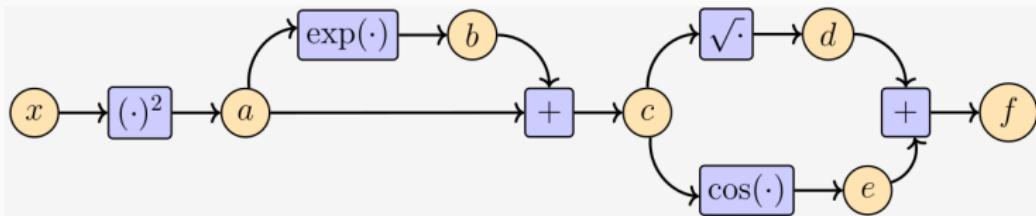
- Here, the blue nodes are **operations**, while the orange nodes are **intermediate results**
- The computational graph implements the function

$$f(x) = \sqrt{x^2 + \exp(x^2)} + \cos(x^2 + \exp(x^2))$$

- Note that there might be many graphs implementing the same function (graph is not unique for given  $f$ )

## Forward Pass

## Example



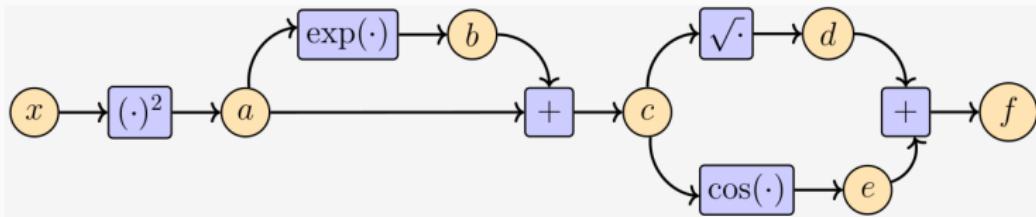
What is  $f(x)$  for  $x = 2$ ?

- $a = x^2 = 4$
- $b = \exp(a) = 54.598$
- $c = a + b = 58.598$
- $d = \sqrt{c} = 7.6549$
- $e = \cos(c) = -0.46$
- $f = d + e = 7.194$

Let's store forward messages  $a, b, c, d, e, f$  in the orange nodes

# Backwards Pass (Autodiff, Backprop)

Example



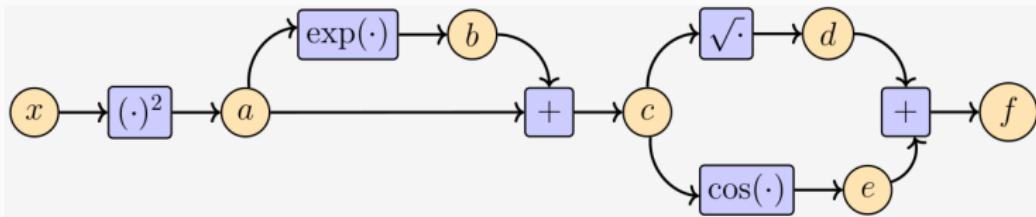
What is  $\frac{\partial f}{\partial x}$ ?

(more precisely,  $\frac{\partial f}{\partial x} \Big|_{x=2}$ )

- $\frac{\partial f}{\partial f} =$
- $\frac{\partial f}{\partial e} =$
- $\frac{\partial f}{\partial d} =$
- $\frac{\partial f}{\partial c} =$
- $\frac{\partial f}{\partial b} =$
- $\frac{\partial f}{\partial a} =$
- $\frac{\partial f}{\partial x} =$

# Backwards Pass (Autodiff, Backprop)

Example



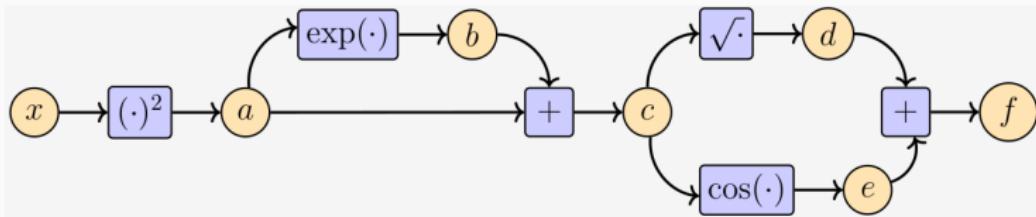
What is  $\frac{\partial f}{\partial x}$ ?

(more precisely,  $\frac{\partial f}{\partial x} \Big|_{x=2}$ )

- $\frac{\partial f}{\partial f} = 1$
- $\frac{\partial f}{\partial e} =$
- $\frac{\partial f}{\partial d} =$
- $\frac{\partial f}{\partial c} =$
- $\frac{\partial f}{\partial b} =$
- $\frac{\partial f}{\partial a} =$
- $\frac{\partial f}{\partial x} =$

# Backwards Pass (Autodiff, Backprop)

Example



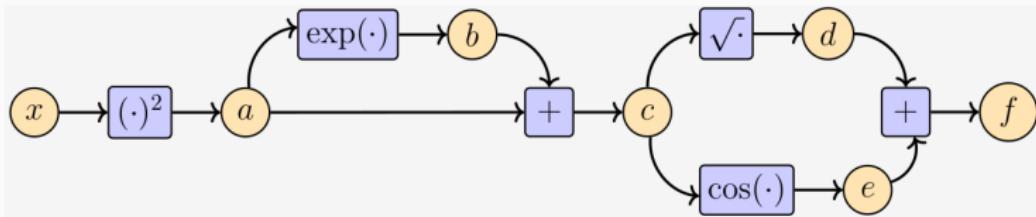
What is  $\frac{\partial f}{\partial x}$ ?

(more precisely,  $\frac{\partial f}{\partial x} \Big|_{x=2}$ )

- $\frac{\partial f}{\partial f} = 1$
- $\frac{\partial f}{\partial e} = \frac{\partial f}{\partial f} \frac{\partial f}{\partial e} = 1 \times 1 = 1$
- $\frac{\partial f}{\partial d} =$
- $\frac{\partial f}{\partial c} =$
- $\frac{\partial f}{\partial b} =$
- $\frac{\partial f}{\partial a} =$
- $\frac{\partial f}{\partial x} =$

# Backwards Pass (Autodiff, Backprop)

Example



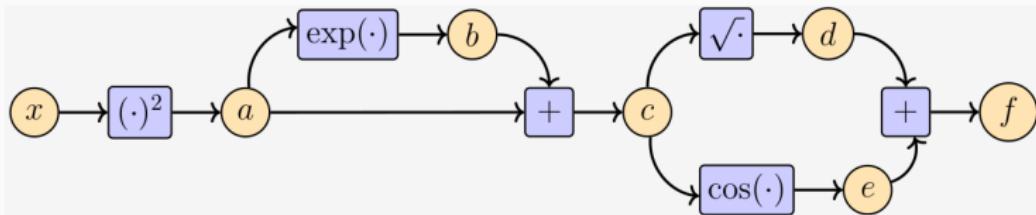
What is  $\frac{\partial f}{\partial x}$ ?

(more precisely,  $\frac{\partial f}{\partial x} \Big|_{x=2}$ )

- $\frac{\partial f}{\partial f} = 1$
- $\frac{\partial f}{\partial e} = \frac{\partial f}{\partial f} \frac{\partial f}{\partial e} = 1 \times 1 = 1$
- $\frac{\partial f}{\partial d} = \frac{\partial f}{\partial f} \frac{\partial f}{\partial d} = 1 \times 1 = 1$
- $\frac{\partial f}{\partial c} =$
- $\frac{\partial f}{\partial b} =$
- $\frac{\partial f}{\partial a} =$
- $\frac{\partial f}{\partial x} =$

# Backwards Pass (Autodiff, Backprop)

Example



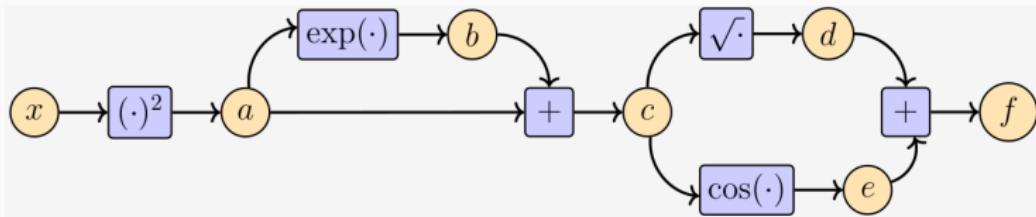
What is  $\frac{\partial f}{\partial x}$ ?

(more precisely,  $\frac{\partial f}{\partial x} \Big|_{x=2}$ )

- $\frac{\partial f}{\partial f} = 1$
- $\frac{\partial f}{\partial e} = \frac{\partial f}{\partial f} \frac{\partial f}{\partial e} = 1 \times 1 = 1$
- $\frac{\partial f}{\partial d} = \frac{\partial f}{\partial f} \frac{\partial f}{\partial d} = 1 \times 1 = 1$
- $\frac{\partial f}{\partial c} = \frac{\partial f}{\partial e} \frac{\partial e}{\partial c} + \frac{\partial f}{\partial d} \frac{\partial d}{\partial c} = 1 \times -\sin(c) + 1 \times \frac{1}{2}c^{-\frac{1}{2}} = -0.822$
- $\frac{\partial f}{\partial b} =$
- $\frac{\partial f}{\partial a} =$
- $\frac{\partial f}{\partial x} =$

# Backwards Pass (Autodiff, Backprop)

Example



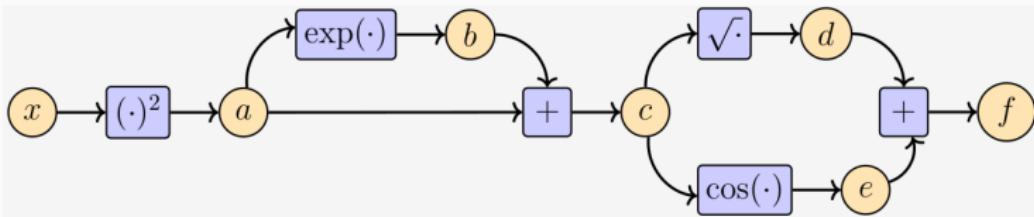
What is  $\frac{\partial f}{\partial x}$ ?

(more precisely,  $\frac{\partial f}{\partial x} \Big|_{x=2}$ )

- $\frac{\partial f}{\partial f} = 1$
- $\frac{\partial f}{\partial e} = \frac{\partial f}{\partial f} \frac{\partial f}{\partial e} = 1 \times 1 = 1$
- $\frac{\partial f}{\partial d} = \frac{\partial f}{\partial f} \frac{\partial f}{\partial d} = 1 \times 1 = 1$
- $\frac{\partial f}{\partial c} = \frac{\partial f}{\partial e} \frac{\partial e}{\partial c} + \frac{\partial f}{\partial d} \frac{\partial d}{\partial c} = 1 \times -\sin(c) + 1 \times \frac{1}{2}c^{-\frac{1}{2}} = -0.822$
- $\frac{\partial f}{\partial b} = \frac{\partial f}{\partial c} \frac{\partial c}{\partial b} = -0.822 \times 1 = -0.822$
- $\frac{\partial f}{\partial a} =$
- $\frac{\partial f}{\partial x} =$

# Backwards Pass (Autodiff, Backprop)

Example



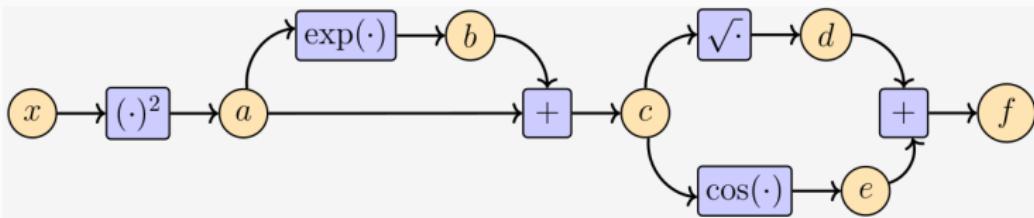
What is  $\frac{\partial f}{\partial x}$ ?

(more precisely,  $\frac{\partial f}{\partial x}|_{x=2}$ )

- $\frac{\partial f}{\partial f} = 1$
- $\frac{\partial f}{\partial e} = \frac{\partial f}{\partial f} \frac{\partial f}{\partial e} = 1 \times 1 = 1$
- $\frac{\partial f}{\partial d} = \frac{\partial f}{\partial f} \frac{\partial f}{\partial d} = 1 \times 1 = 1$
- $\frac{\partial f}{\partial c} = \frac{\partial f}{\partial e} \frac{\partial e}{\partial c} + \frac{\partial f}{\partial d} \frac{\partial d}{\partial c} = 1 \times -\sin(c) + 1 \times \frac{1}{2}c^{-\frac{1}{2}} = -0.822$
- $\frac{\partial f}{\partial b} = \frac{\partial f}{\partial c} \frac{\partial c}{\partial b} = -0.822 \times 1 = -0.822$
- $\frac{\partial f}{\partial a} = \frac{\partial f}{\partial c} \frac{\partial c}{\partial a} + \frac{\partial f}{\partial b} \frac{\partial b}{\partial a} = -0.822 \times 1 + -0.822 \times \exp(a) = -45.717$
- $\frac{\partial f}{\partial x} =$

# Backwards Pass (Autodiff, Backprop)

Example



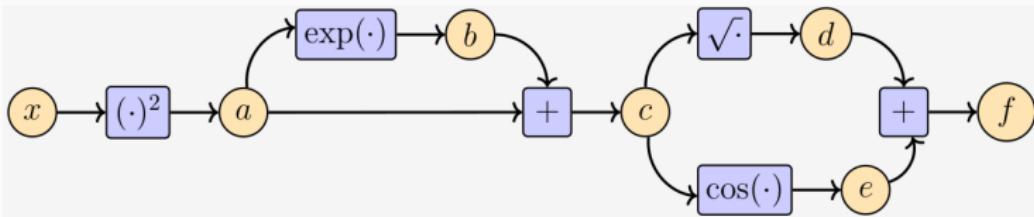
What is  $\frac{\partial f}{\partial x}$ ?

(more precisely,  $\frac{\partial f}{\partial x} \Big|_{x=2}$ )

- $\frac{\partial f}{\partial f} = 1$
- $\frac{\partial f}{\partial e} = \frac{\partial f}{\partial f} \frac{\partial f}{\partial e} = 1 \times 1 = 1$
- $\frac{\partial f}{\partial d} = \frac{\partial f}{\partial f} \frac{\partial f}{\partial d} = 1 \times 1 = 1$
- $\frac{\partial f}{\partial c} = \frac{\partial f}{\partial e} \frac{\partial e}{\partial c} + \frac{\partial f}{\partial d} \frac{\partial d}{\partial c} = 1 \times -\sin(c) + 1 \times \frac{1}{2}c^{-\frac{1}{2}} = -0.822$
- $\frac{\partial f}{\partial b} = \frac{\partial f}{\partial c} \frac{\partial c}{\partial b} = -0.822 \times 1 = -0.822$
- $\frac{\partial f}{\partial a} = \frac{\partial f}{\partial c} \frac{\partial c}{\partial a} + \frac{\partial f}{\partial b} \frac{\partial b}{\partial a} = -0.822 \times 1 + -0.822 \times \exp(a) = -45.717$
- $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial a} \frac{\partial a}{\partial x} = -45.717 \times 2x = -182.87$

# Backwards Pass (Autodiff, Backprop)

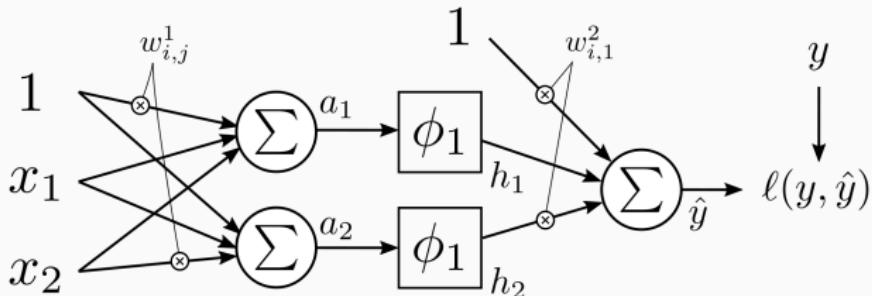
Example



What is  $\frac{\partial f}{\partial x}$ ?

(more precisely,  $\frac{\partial f}{\partial x} \Big|_{x=2}$ )

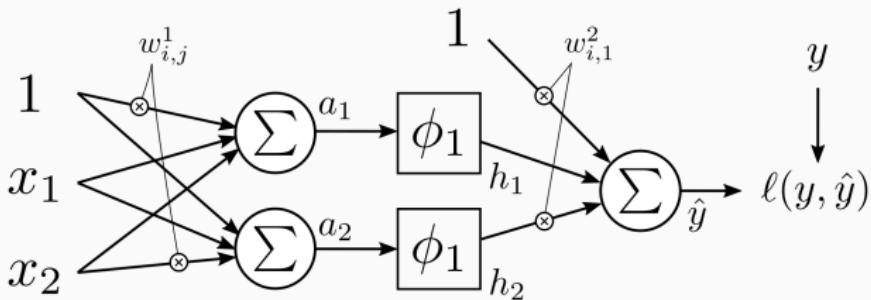
- $\frac{\partial f}{\partial f} = 1$
- $\frac{\partial f}{\partial e} = \frac{\partial f}{\partial f} \frac{\partial f}{\partial e} = 1 \times 1 = 1$
- $\frac{\partial f}{\partial d} = \frac{\partial f}{\partial f} \frac{\partial f}{\partial d} = 1 \times 1 = 1$
- $\frac{\partial f}{\partial c} = \frac{\partial f}{\partial e} \frac{\partial e}{\partial c} + \frac{\partial f}{\partial d} \frac{\partial d}{\partial c} = 1 \times -\sin(c) + 1 \times \frac{1}{2}c^{-\frac{1}{2}} = -0.822$
- $\frac{\partial f}{\partial b} = \frac{\partial f}{\partial c} \frac{\partial c}{\partial b} = -0.822 \times 1 = -0.822$
- $\frac{\partial f}{\partial a} = \frac{\partial f}{\partial c} \frac{\partial a}{\partial a} + \frac{\partial f}{\partial b} \frac{\partial b}{\partial a} = -0.822 \times 1 + -0.822 \times \exp(a) = -45.717$
- $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial a} \frac{\partial a}{\partial x} = -45.717 \times 2x = -182.87$
- Note we get derivatives for **all nodes**, not only  $x$ !



- Neural nets are simply computational graphs
- Inputs to the graph:
  - features  $x_1, x_2, \dots, x_D$
  - weights  $w_{i,j}^l$
  - targets  $y$
- Forward pass computes all  $a_i, h_i, \hat{y}$
- The loss  $\ell$  is also a node in the graph!

# Backprop in Neural Networks

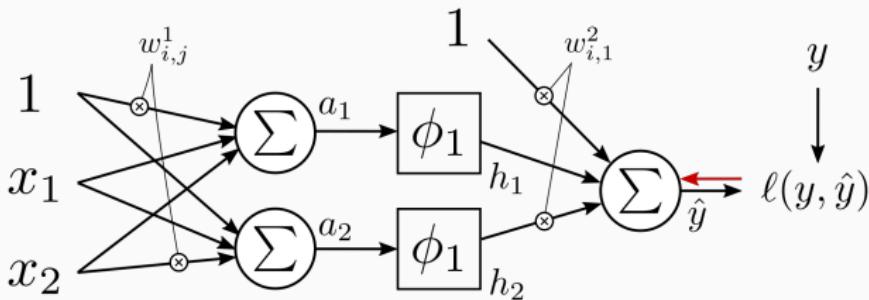
Example



- $\frac{\partial \ell}{\partial \ell} = 1$

# Backprop in Neural Networks

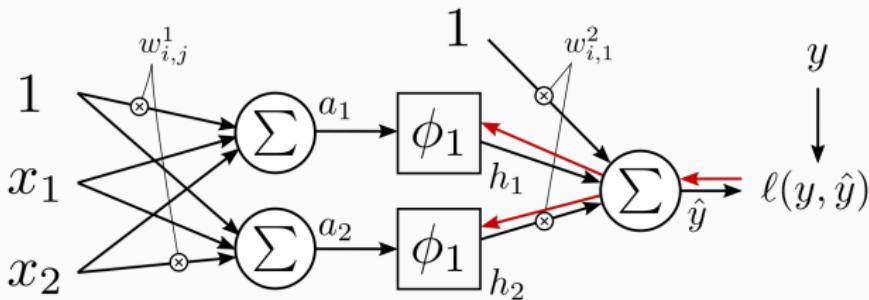
Example



- $\frac{\partial \ell}{\partial \ell} = 1$
- $\frac{\partial \ell}{\partial y} = \frac{\partial \ell}{\partial \ell} \frac{\partial \ell}{\partial y} = 1 \frac{\partial \ell}{\partial \hat{y}} =: B_1$  (closed form for most losses)

# Backprop in Neural Networks

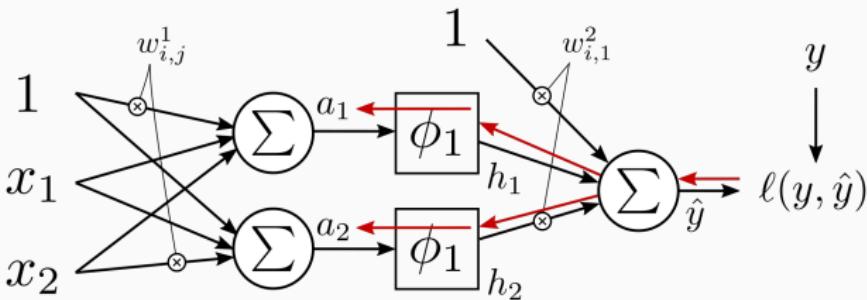
Example



- $\frac{\partial \ell}{\partial \ell} = 1$
- $\frac{\partial \ell}{\partial \hat{y}} = \frac{\partial \ell}{\partial \ell} \frac{\partial \ell}{\partial \hat{y}} = 1 \frac{\partial \ell}{\partial \hat{y}} =: B_1$  (closed form for most losses)
- $\frac{\partial \ell}{\partial h_i} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_i} = B_1 w_{i,1}^2 =: B_{2,i}$  ( $\hat{y} = \sum_{i=0}^2 w_{i,1}^2 h_i$ )

# Backprop in Neural Networks

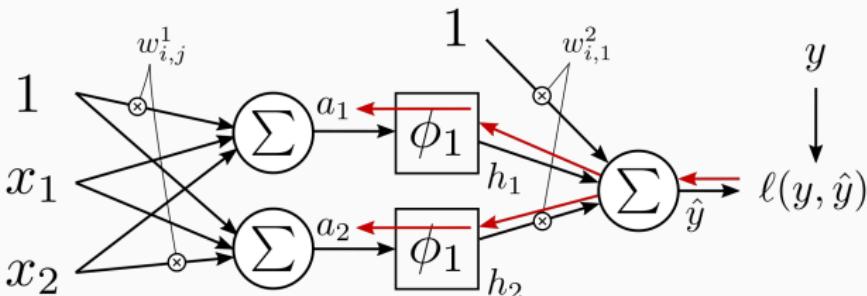
Example



- $\frac{\partial \ell}{\partial \ell} = 1$
- $\frac{\partial \ell}{\partial \hat{y}} = \frac{\partial \ell}{\partial \ell} \frac{\partial \ell}{\partial \hat{y}} = 1 \frac{\partial \ell}{\partial \hat{y}} =: B_1$  (closed form for most losses)
- $\frac{\partial \ell}{\partial h_i} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_i} = B_1 w_{i,1}^2 =: B_{2,i}$  ( $\hat{y} = \sum_{i=0}^2 w_{i,1}^2 h_i$ )
- $\frac{\partial \ell}{\partial a_i} = \frac{\partial \ell}{\partial h_i} \frac{\partial h_i}{\partial a_i} = B_{2,i} \phi'_1(a_i) =: B_{3,i}$  (known, closed form  $\phi'_1$ )

# Backprop in Neural Networks

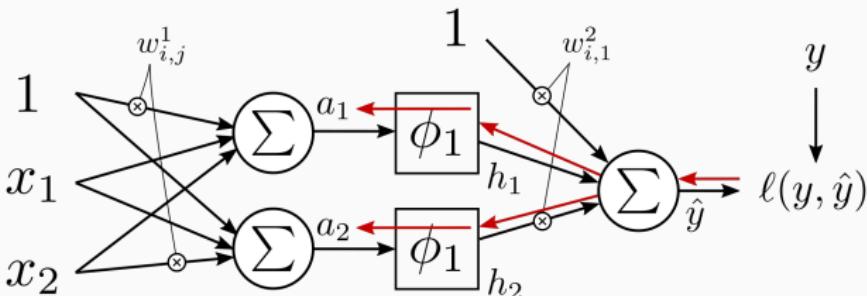
Example



- $\frac{\partial \ell}{\partial \ell} = 1$
- $\frac{\partial \ell}{\partial \hat{y}} = \frac{\partial \ell}{\partial \ell} \frac{\partial \ell}{\partial \hat{y}} = 1 \frac{\partial \ell}{\partial \hat{y}} =: B_1$  (closed form for most losses)
- $\frac{\partial \ell}{\partial h_i} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_i} = B_1 w_{i,1}^2 =: B_{2,i}$  ( $\hat{y} = \sum_{i=0}^2 w_{i,1}^2 h_i$ )
- $\frac{\partial \ell}{\partial a_i} = \frac{\partial \ell}{\partial h_i} \frac{\partial h_i}{\partial a_i} = B_{2,i} \phi'_1(a_i) =: B_{3,i}$  (known, closed form  $\phi'_1$ )
- $\frac{\partial \ell}{\partial w_{i,1}^2} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_{i,1}^2} = B_1 h_i$  ( $\hat{y} = \sum_{i=0}^2 w_{i,1}^2 h_i$ )

# Backprop in Neural Networks

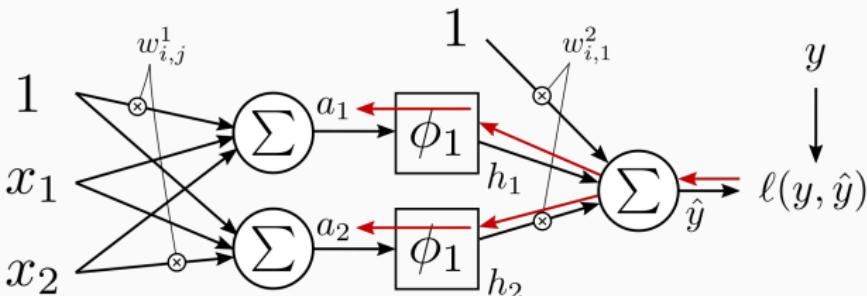
Example



- $\frac{\partial \ell}{\partial \ell} = 1$
- $\frac{\partial \ell}{\partial \hat{y}} = \frac{\partial \ell}{\partial \ell} \frac{\partial \ell}{\partial \hat{y}} = 1 \frac{\partial \ell}{\partial \hat{y}} =: B_1$  (closed form for most losses)
- $\frac{\partial \ell}{\partial h_i} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_i} = B_1 w_{i,1}^2 =: B_{2,i}$  ( $\hat{y} = \sum_{i=0}^2 w_{i,1}^2 h_i$ )
- $\frac{\partial \ell}{\partial a_i} = \frac{\partial \ell}{\partial h_i} \frac{\partial h_i}{\partial a_i} = B_{2,i} \phi'_1(a_i) =: B_{3,i}$  (known, closed form  $\phi'_1$ )
- $\frac{\partial \ell}{\partial w_{i,1}^2} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_{i,1}^2} = B_1 h_i$  ( $\hat{y} = \sum_{i=0}^2 w_{i,1}^2 h_i$ )
- $\frac{\partial \ell}{\partial w_{i,j}^1} = \frac{\partial \ell}{\partial a_j} \frac{\partial a_j}{\partial w_{i,j}^1} = B_{3,j} x_i$  ( $a_j = \sum_{i=0}^2 w_{i,j}^1 x_i$ )

# Backprop in Neural Networks

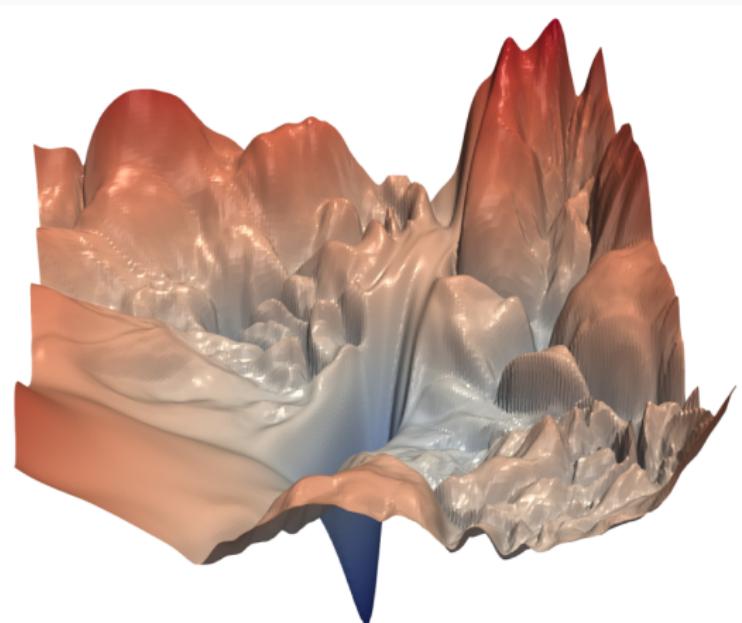
Example



- $\frac{\partial \ell}{\partial \ell} = 1$
- $\frac{\partial \ell}{\partial \hat{y}} = \frac{\partial \ell}{\partial \ell} \frac{\partial \ell}{\partial \hat{y}} = 1 \frac{\partial \ell}{\partial \hat{y}} =: B_1$  (closed form for most losses)
- $\frac{\partial \ell}{\partial h_i} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_i} = B_1 w_{i,1}^2 =: B_{2,i}$  ( $\hat{y} = \sum_{i=0}^2 w_{i,1}^2 h_i$ )
- $\frac{\partial \ell}{\partial a_i} = \frac{\partial \ell}{\partial h_i} \frac{\partial h_i}{\partial a_i} = B_{2,i} \phi'_1(a_i) =: B_{3,i}$  (known, closed form  $\phi'_1$ )
- $\frac{\partial \ell}{\partial w_{i,1}^2} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_{i,1}^2} = B_1 h_i$  ( $\hat{y} = \sum_{i=0}^2 w_{i,1}^2 h_i$ )
- $\frac{\partial \ell}{\partial w_{i,j}^1} = \frac{\partial \ell}{\partial a_j} \frac{\partial a_j}{\partial w_{i,j}^1} = B_{3,j} x_i$  ( $a_j = \sum_{i=0}^2 w_{i,j}^1 x_i$ )
- $\frac{\partial \ell}{\partial x_i} = \sum_j \frac{\partial \ell}{\partial a_j} \frac{\partial a_j}{\partial x_i} = \sum_j B_{3,j} w_{i,j}^1$  (if needed)

## Loss Landscape of Neural Networks (Non-Convex)

Training loss in neural networks is highly non-convex, and we can find only local minima. However, in practice they work very well.



Source: Li et al., *Visualizing the Loss Landscape of Neural Nets*, NeurIPS 2018.

- ANN: network of artificial neurons
- Non-linearity is key, otherwise ANNs “collapse” to a linear function
- Universal function approximators, even for one hidden layer
- Deep networks are more expressive efficient
- Learning via gradient descent
- Gradients computed via **automatic differentiation**  
**(backpropagation, chain rule)**