Natural Language Processing with Deep Learning
Neural Networks – a Walkthrough

Navid Rekab-Saz

navid.rekabsaz@jku.at

JOHANNES KEPLER UNIVERSITY LINZ

Institute of Computational Perception

Agenda

- Introduction
- Non-linearities
- Forward pass & backpropagation
- Softmax & loss function
- Optimization & regularization

Agenda

- **Introduction**
- Non-linearities
- Forward pass & backpropagation
- Softmax & loss function
- Optimization & regularization

Notation

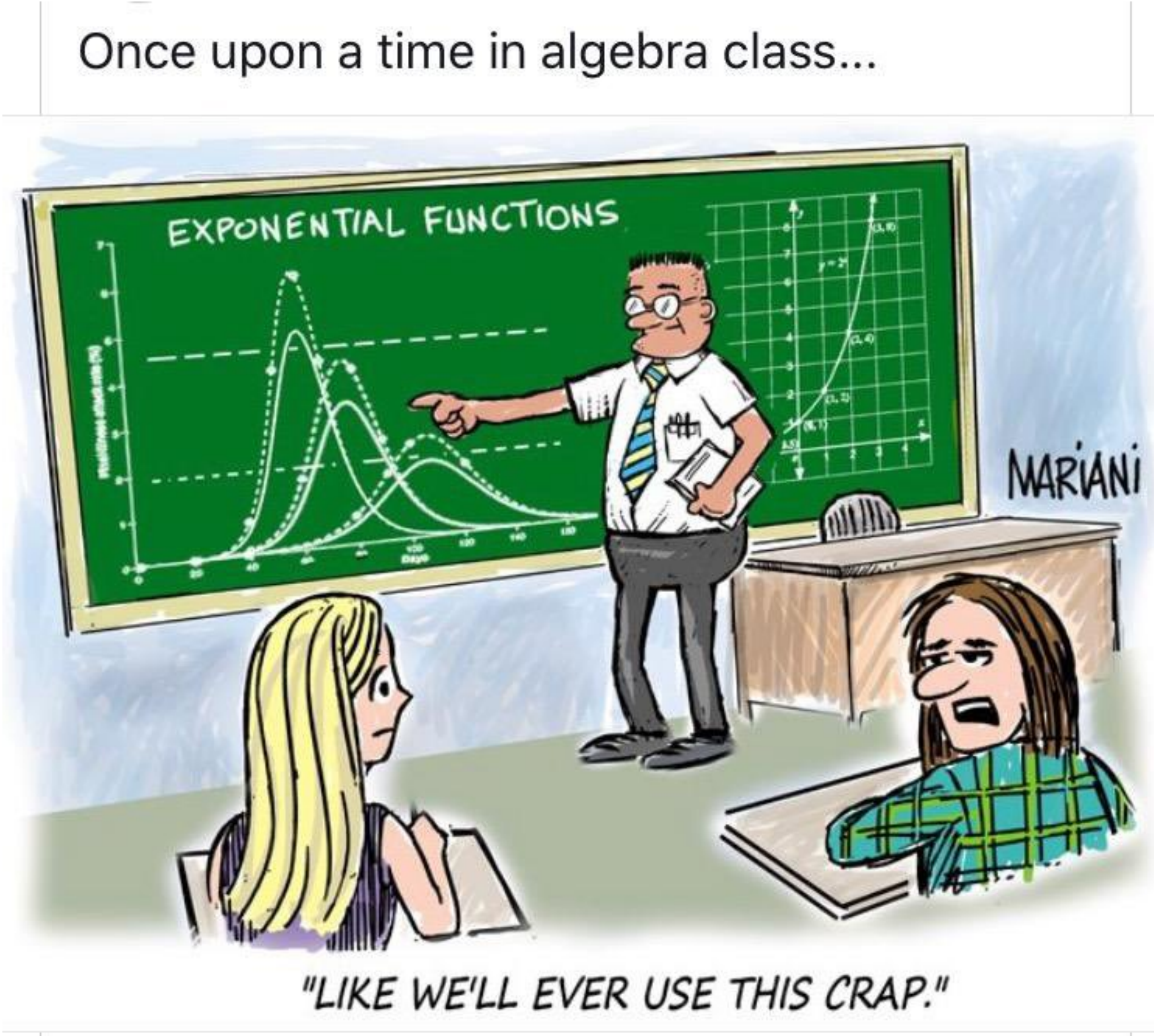
- $a \longrightarrow$ scalar

- $\boldsymbol{b} \longrightarrow$ vector
  - i^{th} element of $\boldsymbol{b}$ is the scalar b_i

- $\boldsymbol{C} \longrightarrow$ matrix
  - i^{th} vector of $\boldsymbol{C}$ is $\boldsymbol{c}_i$
  - j^{th} element of the i^{th} vector of $\boldsymbol{C}$ is the scalar $c_{i,j}$

- Tensor: generalization of scalar, vector, matrix to any arbitrary dimension

Linear Algebra



Linear Algebra – Transpose

- $\boldsymbol{a}$ is in $1 \times d$ dimensions $\rightarrow$ $\boldsymbol{a}^{\mathrm{T}}$ is in $d \times 1$ dimensions

- $\boldsymbol{A}$ is in $e \times d$ dimensions $\rightarrow$ $\boldsymbol{A}^{\mathrm{T}}$ is in $d \times e$ dimensions

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

Linear Algebra – Dot product

- $\boldsymbol{a} \cdot \boldsymbol{b}^T = c$
  - dimensions: $1 \times d \cdot d \times 1 = 1$

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix} = 5$$

- $\boldsymbol{a} \cdot \boldsymbol{B} = \boldsymbol{c}$
  - dimensions: $1 \times d \cdot d \times e = 1 \times e$

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 2 & 3 \\ 0 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 5 & 2 \end{bmatrix}$$

- $\boldsymbol{A} \cdot \boldsymbol{B} = \boldsymbol{C}$
  - dimensions: $l \times m \cdot m \times n = l \times n$

$$\begin{bmatrix} 1 & 2 & 3 \\ 1 & 0 & 1 \\ 0 & 0 & 5 \\ 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} 2 & 3 \\ 0 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 5 & 2 \\ 3 & 2 \\ 5 & -5 \\ 8 & 13 \end{bmatrix}$$

- Linear transformation: dot product of a vector to a matrix

Probability

- Conditional probability

$$p(y|x)$$

- Probability distribution
  - For a discrete random variable $\boldsymbol{z}$ with K states
    - $0 \leq p(z_i) \leq 1$
    - $\sum_{i=1}^{K} p(z_i) = 1$
  - E.g. with $K = 4$ states: $[0.2 \quad 0.3 \quad 0.45 \quad 0.05]$

Probability

- Expected value

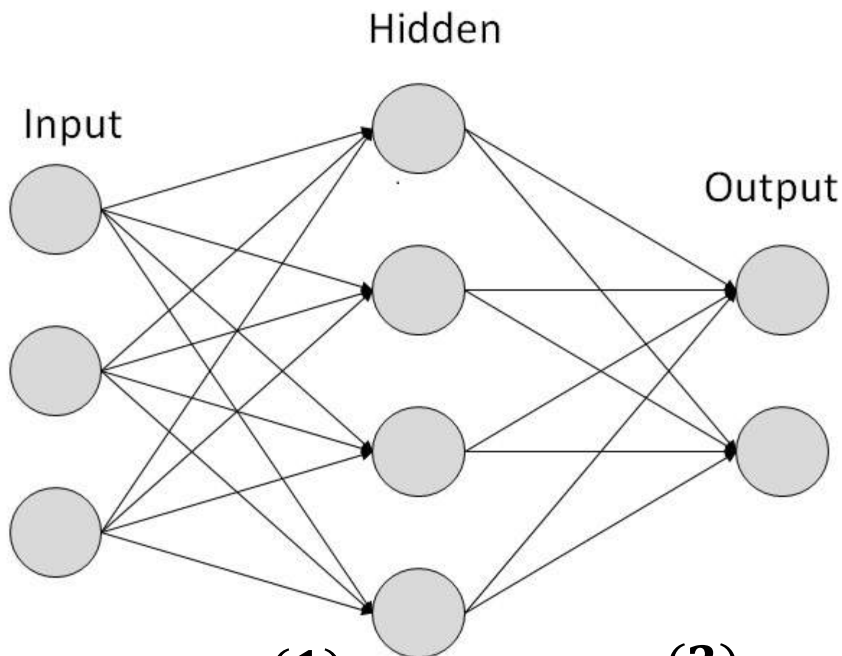
$$\mathbb{E}_{x \sim X}[f] = \frac{1}{|X|} \sum_{x \in X} f(x)$$

  - Note: This is an imprecise definition. Though, it suffices for our use in this lecture

Artificial Neural Networks

- Neural Networks are non-linear functions and universal approximators
- They composed of several simple (non-)linear operations
- Neural networks can readily be defined as probabilistic models which estimate $p(y|\boldsymbol{x};\boldsymbol{W})$
  - Given input vector $\boldsymbol{x}$ and the set of parameters $\boldsymbol{W}$, estimate the probability of the output class y

A Feedforward network



input vector
$\boldsymbol{x}$

output probability distribution
$p(y|\boldsymbol{x};\boldsymbol{W})$

$\boldsymbol{W}^{(1)}$
size 3x4

$\boldsymbol{W}^{(2)}$
size 4x2

parameter matrices

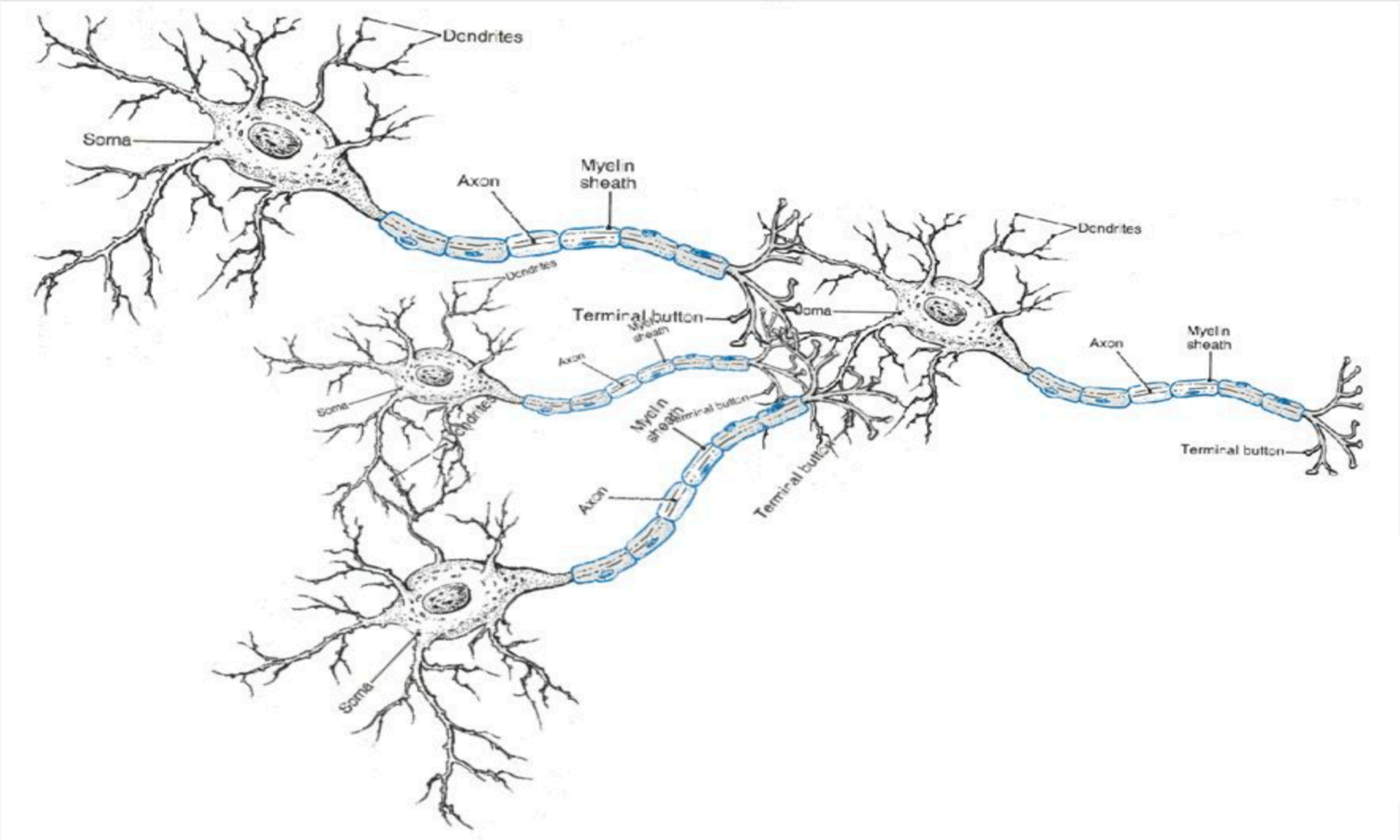
Learning with Neural Networks

- Design the network's architecture
- Consider proper **regularization** methods
- Initialize parameters
- Loop until some **exit criteria** are met
  - Sample a **minibatch** from training data $\mathcal{D}$
  - Loop over data points in the minibatch
    - **Forward pass**: given input $\boldsymbol{x}$ predict output $p(y|\boldsymbol{x};\boldsymbol{W})$
  - Calculate **loss** function
  - Calculate the **gradient** of each parameter regarding the loss function using the **backpropagation** algorithm
  - **Update** parameters using their gradients

Agenda

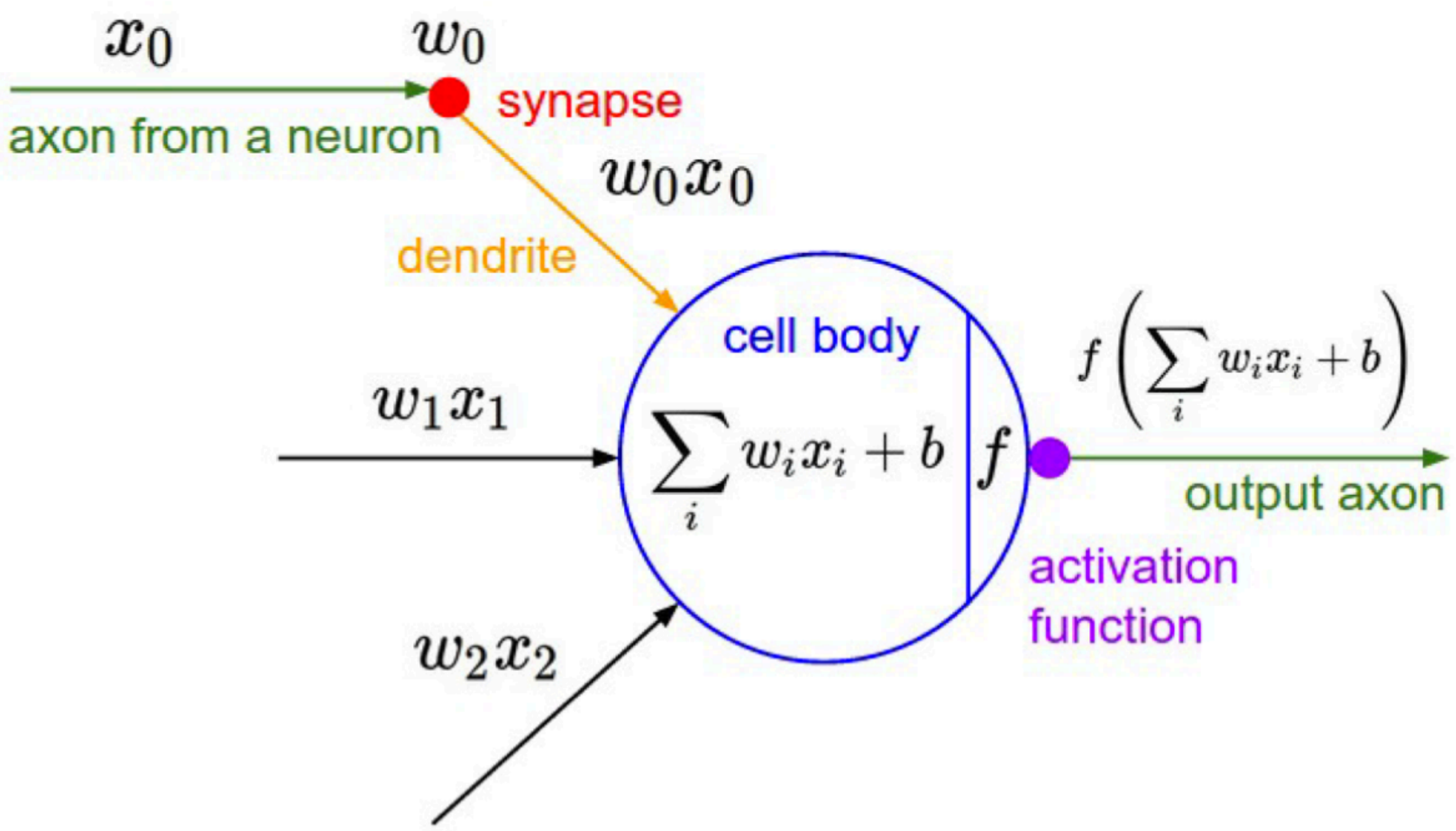
- Introduction
- **Non-linearities**
- Forward pass & backpropagation
- Softmax & loss function
- Optimization & regularization

Neural Computation



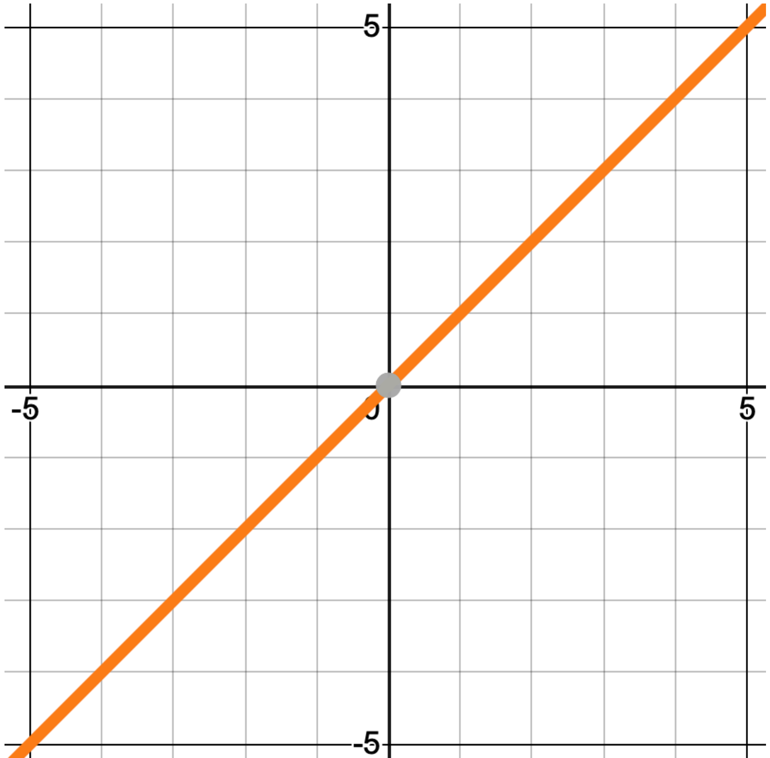
source

An Artificial Neuron



Linear

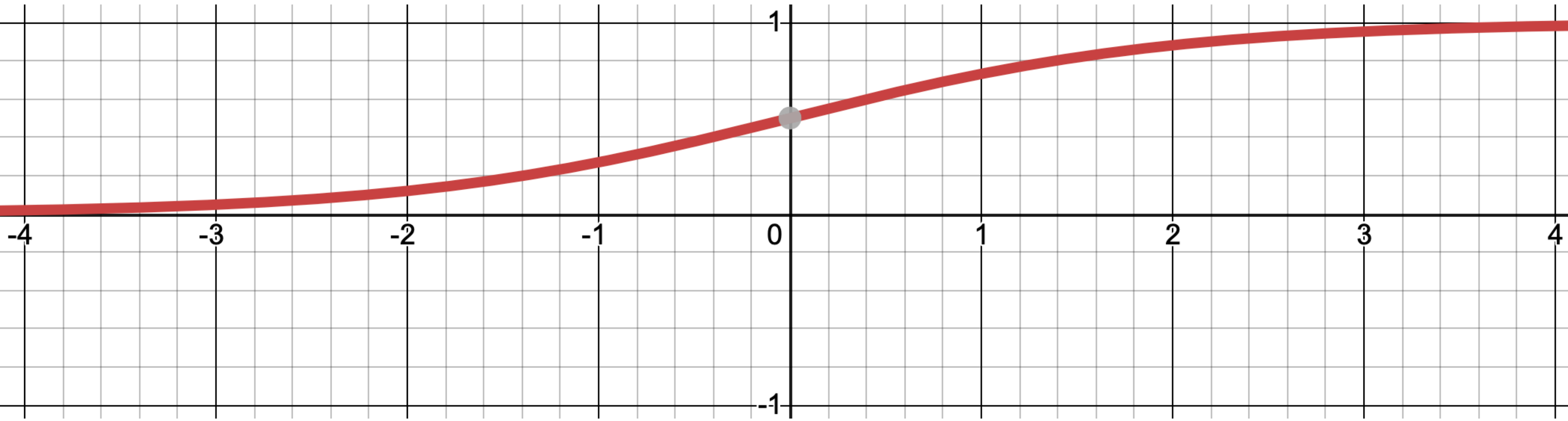
$$f(x) = x$$



Sigmoid

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

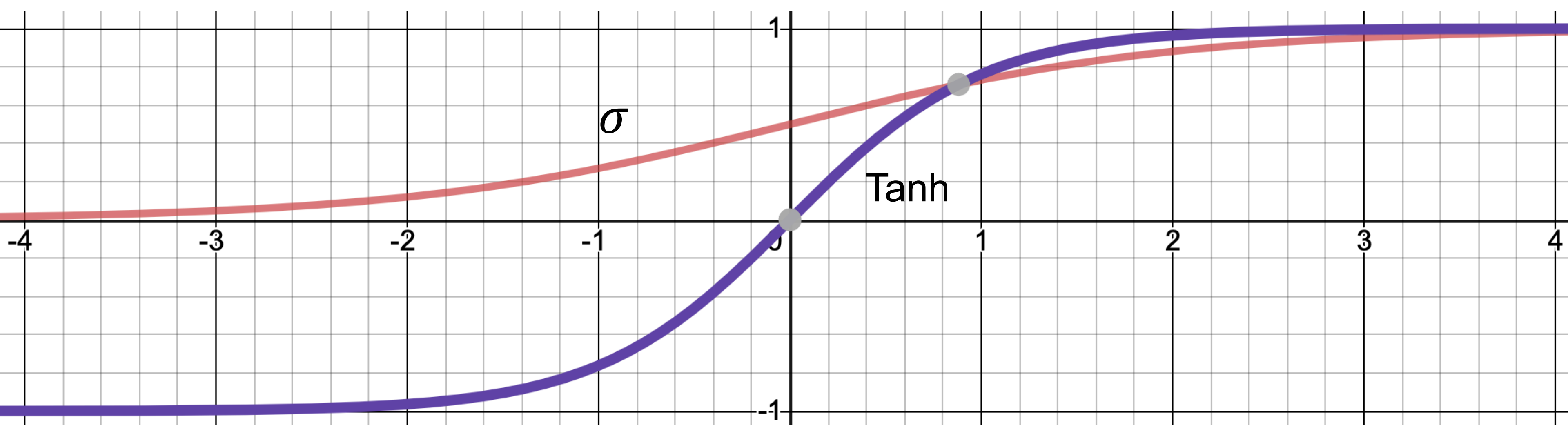
- squashes input between 0 and 1
- Output becomes like a probability value



Hyperbolic Tangent (Tanh)

$$f(x) = \tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

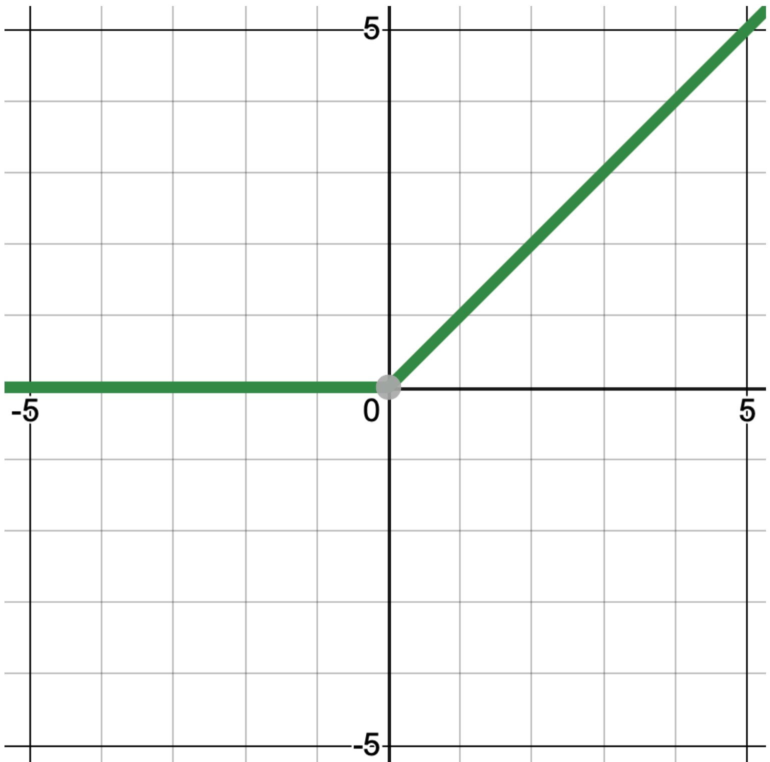
- squashes input between -1 and 1



Rectified Linear Unit (ReLU)

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

- Good for deep architectures, as it prevents vanishing gradient



Examples

$$\boldsymbol{x} = \begin{bmatrix} 1 & 3 \end{bmatrix} \qquad \boldsymbol{W} = \begin{bmatrix} 0.5 & -0.5 & 2 & 0 & 0 \\ 0 & 0 & 0 & 4 & -1 \end{bmatrix}$$

- Linear transformation $\boldsymbol{xW}$:

$$\boldsymbol{xW} = \begin{bmatrix} 1 & 3 \end{bmatrix} \begin{bmatrix} 0.5 & -0.5 & 2 & 0 & -1 \\ 0 & 0 & 0 & 4 & -1 \end{bmatrix} = \begin{bmatrix} \mathbf{0.5} & \mathbf{-0.5} & \mathbf{2} & \mathbf{12} & \mathbf{-4} \end{bmatrix}$$

- Non-linear transformation $\mathrm{ReLU}(\boldsymbol{xW})$:

$$\mathrm{ReLU}(\begin{bmatrix} 0.5 & -0.5 & 2 & 12 & -3 \end{bmatrix}) = \begin{bmatrix} \mathbf{0.5} & \mathbf{0.0} & \mathbf{2} & \mathbf{12} & \mathbf{0.0} \end{bmatrix}$$

- Non-linear transformation $\sigma(\boldsymbol{xW})$:

$$\sigma(\begin{bmatrix} 0.5 & -0.5 & 2 & 12 & -3 \end{bmatrix}) = \begin{bmatrix} \mathbf{0.62} & \mathbf{0.37} & \mathbf{0.88} & \mathbf{0.99} & \mathbf{0.018} \end{bmatrix}$$

- Non-linear transformation $\tanh(\boldsymbol{xW})$:

$$\tanh(\begin{bmatrix} 0.5 & -0.5 & 2 & 12 & -3 \end{bmatrix}) = \begin{bmatrix} \mathbf{0.46} & \mathbf{-0.46} & \mathbf{0.96} & \mathbf{0.99} & \mathbf{-0.99} \end{bmatrix}$$

Agenda

- Introduction
- Non-linearities
- **Forward pass & backpropagation**
- Softmax & loss function
- Optimization & regularization

Forward pass

- Consider this calculation:

$$z(x;\boldsymbol{w}) = 2 * {w_2}^2 + x * w_1 + w_0$$

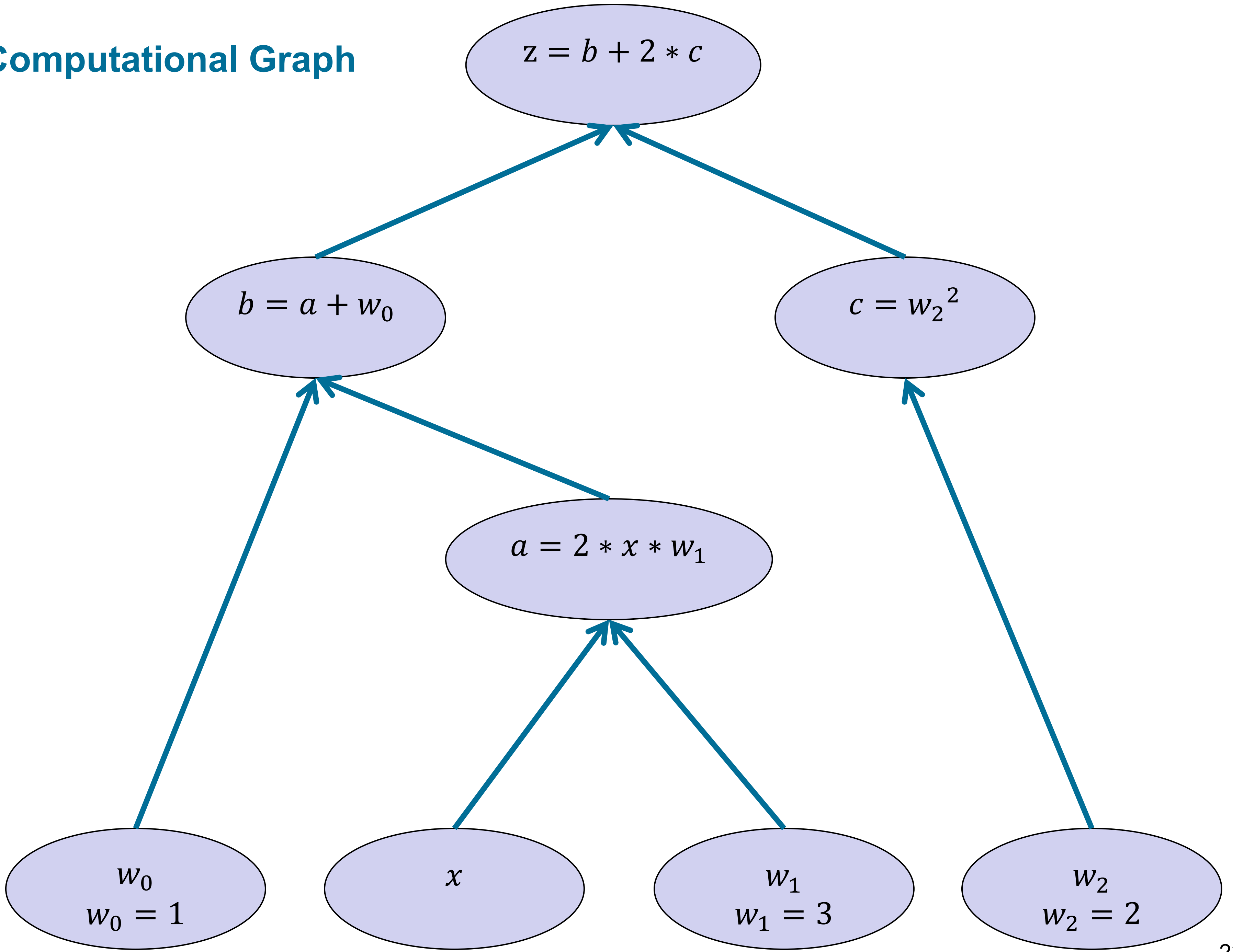
where x is input and $\boldsymbol{w}$ is the set of parameters with the initialization

$w_0 = 1 \quad w_1 = 3 \quad w_2 = 2$

- Let's break it into intermediary variables:

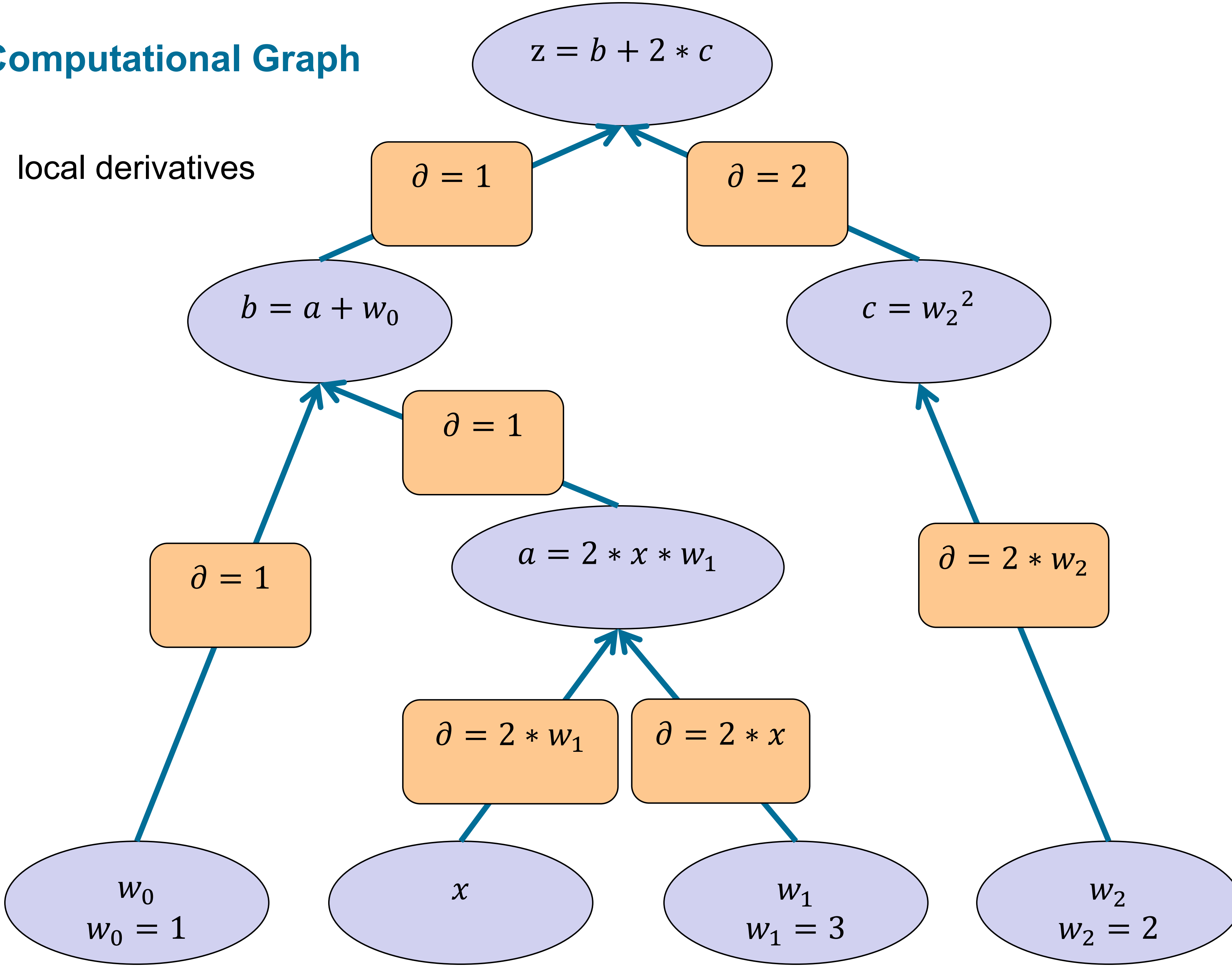
$$a = x * w_1$$
$$b = a + w_0$$
$$c = {w_2}^2$$
$$z = b + 2 * c$$

Computational Graph



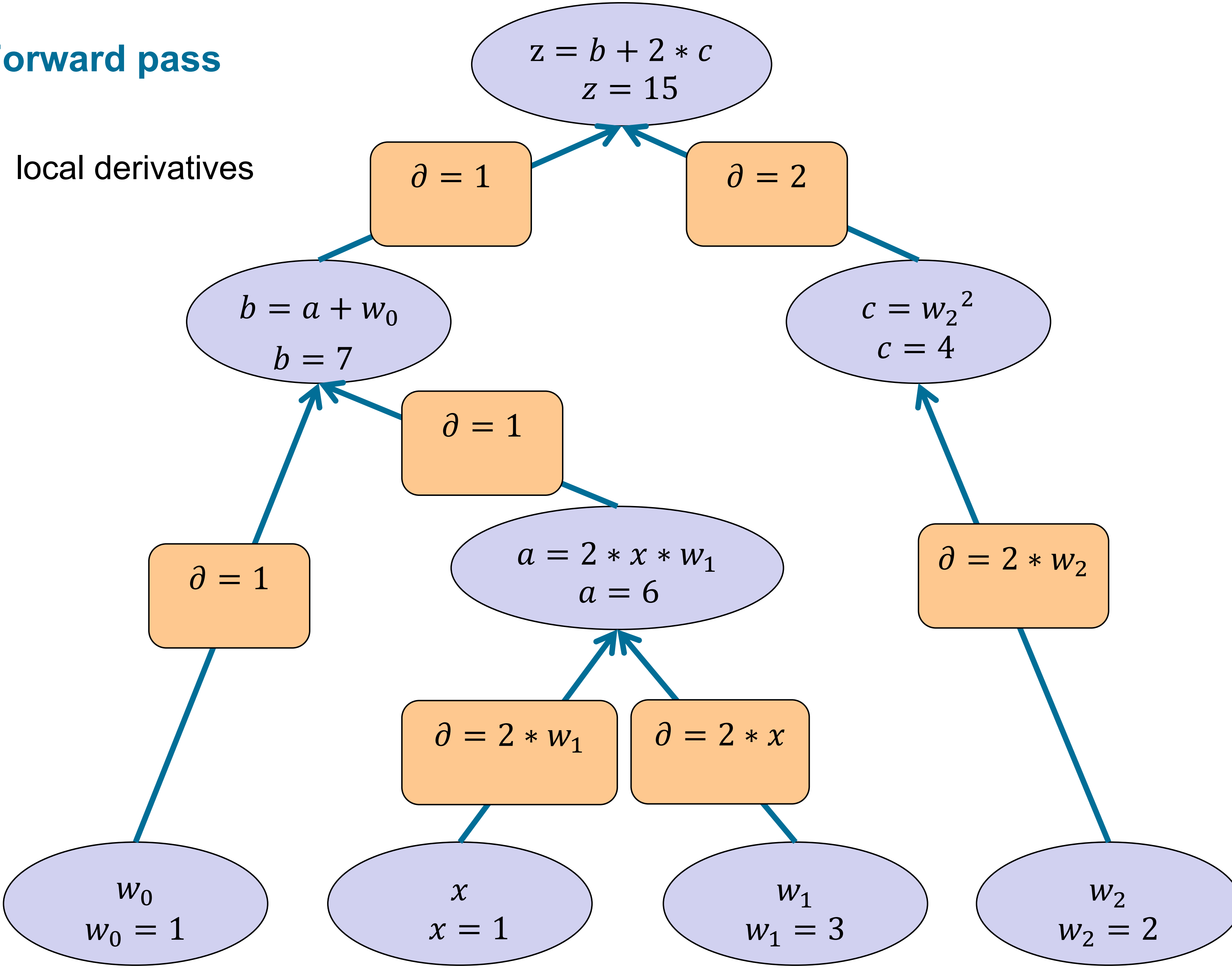
Computational Graph

∂ local derivatives



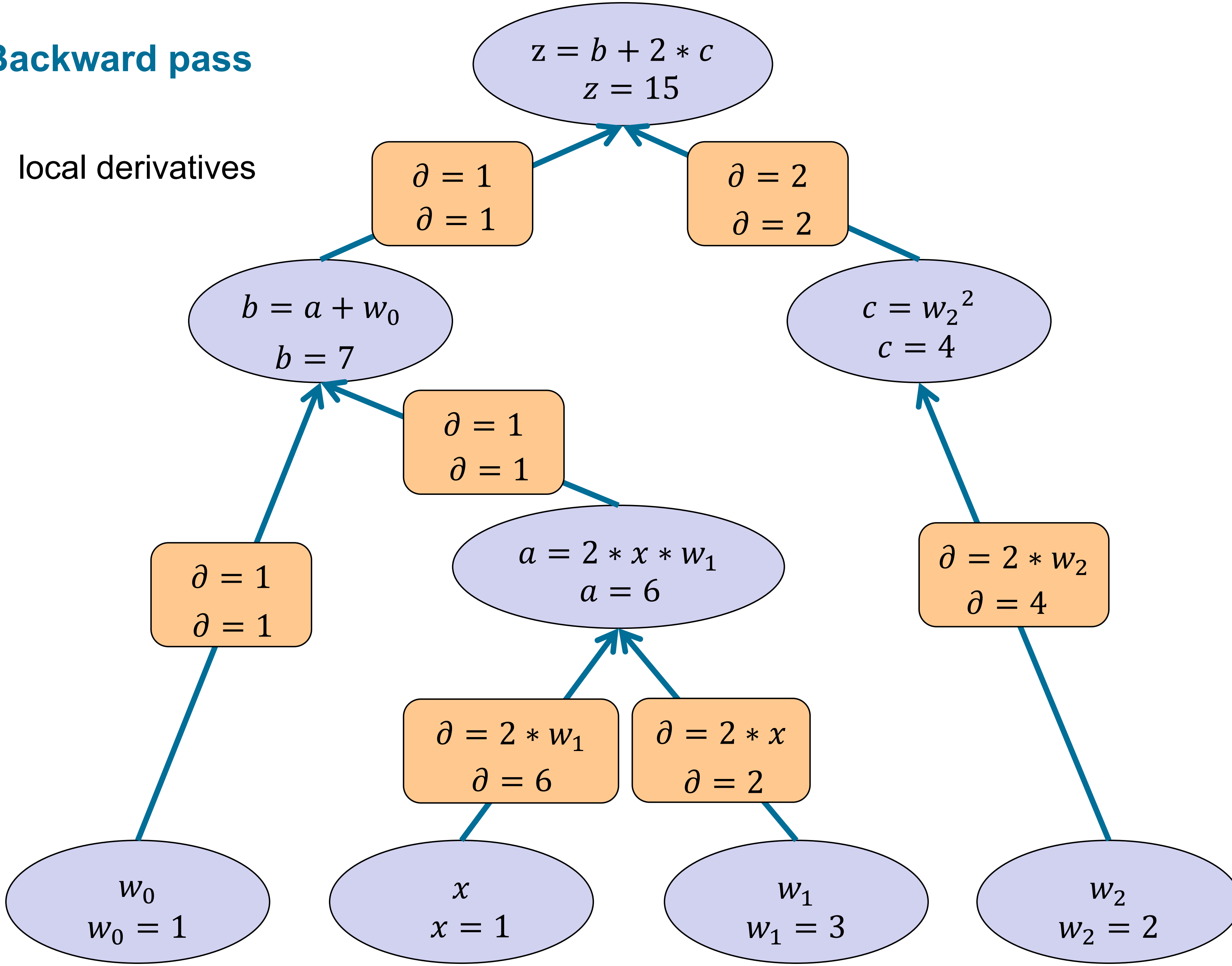
Forward pass

∂ local derivatives



Backward pass

∂ local derivatives



Gradient & Chain rule

- We need the gradient of z regarding $\boldsymbol{w}$ for optimization

$$\nabla_{\boldsymbol{w}} z = \left[\frac{\partial z}{\partial w_0} \quad \frac{\partial z}{\partial w_1} \quad \frac{\partial z}{\partial w_2}\right]$$

- We calculate it using chain rule and local derivates:

$$\frac{\partial z}{\partial w_0} = \frac{\partial z}{\partial b}\frac{\partial b}{\partial w_0}$$

$$\frac{\partial z}{\partial w_1} = \frac{\partial z}{\partial b}\frac{\partial b}{\partial a}\frac{\partial a}{\partial w_1}$$

$$\frac{\partial z}{\partial w_2} = \frac{\partial z}{\partial c}\frac{\partial c}{\partial w_2}$$

Backpropagation

$$\frac{\partial z}{\partial w_0} = \frac{\partial z}{\partial b}\frac{\partial b}{\partial w_0} = 1 * 1 = 1$$

$$\frac{\partial z}{\partial w_1} = \frac{\partial z}{\partial b}\frac{\partial b}{\partial a}\frac{\partial a}{\partial w_1} = 1 * 1 * 2 = 2$$

$$\frac{\partial z}{\partial w_2} = \frac{\partial z}{\partial c}\frac{\partial c}{\partial w_2} = 2 * 4 = 8$$

Agenda

- Introduction
- Non-linearities
- Forward pass & backpropagation
- **Softmax & loss function**
- Optimization & regularization

Softmax

- Given the output vector $\boldsymbol{z}$ of a neural networks model with K output classes

- softmax turns the vector to a probability distribution

$$\text{softmax}(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

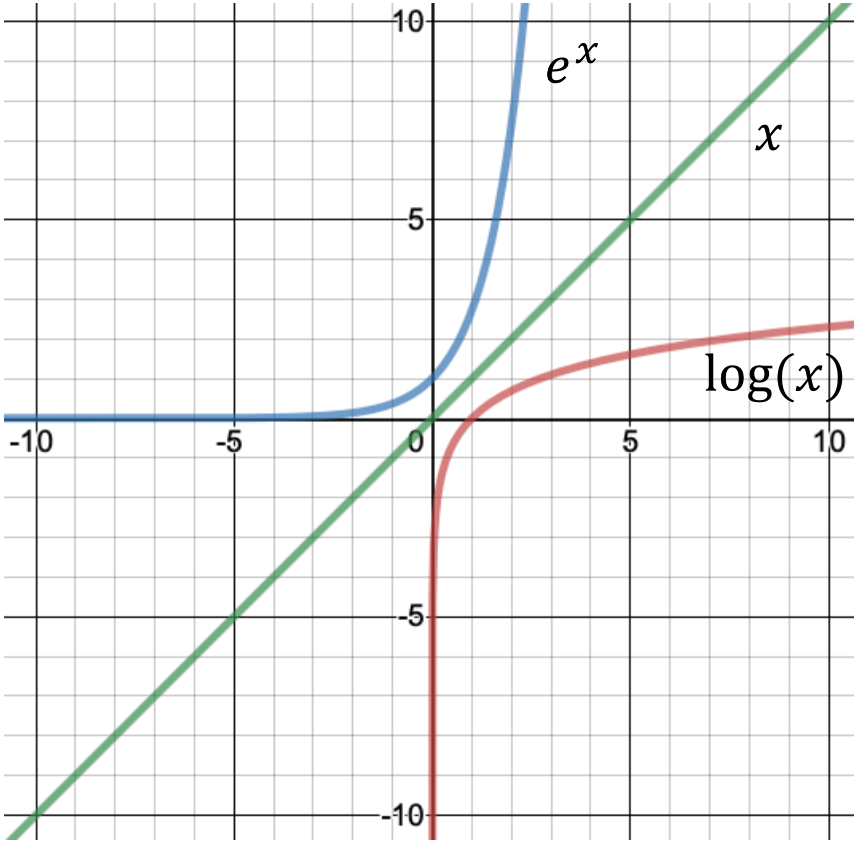
normalization term

Softmax – numeric example

- $K = 4$ classes

$$\boldsymbol{z} = \begin{bmatrix} 1 \\ 2 \\ 5 \\ 6 \end{bmatrix}$$

$$\mathrm{softmax}(\boldsymbol{z}) = \begin{bmatrix} 0.004 \\ 0.013 \\ 0.264 \\ 0.717 \end{bmatrix}$$



Softmax characteristics

- The exponential function in softmax makes the highest value becomes separated from the others
- Softmax identifies the "*max*" but in a "*soft*" way!
- Softmax makes competition between the predicted output values, so that in the extreme case, "*winner takes all*"
  - Winner-takes-all: one output is 1 and the rest are 0
  - This resembles the competition between nearby neurons in the cortex

Negative Log Likelihood (NLL) Loss

- NLL loss function is commonly used in neural networks to optimize classification tasks:

$$\mathcal{L} = -\mathbb{E}_{\boldsymbol{x},y\sim\mathcal{D}} \log p(y|\boldsymbol{x}; \mathbb{W})$$

  - $\mathcal{D}$ the set of (training) data
  - $\boldsymbol{x}$ input vector
  - y correct output class

- NLL is a form of cross entropy loss

NLL + Softmax

- The choice of output function (such as softmax) is highly related to the selection of loss function. These two should fit with each other!

- Softmax and NLL are a *good* pair

- To see why, let's calculate the final NLL loss function when softmax is used at output layer (next page)

NLL + Softmax

- Loss function for one data point: $\mathcal{L}(f(\boldsymbol{x};\boldsymbol{w}), y)$
- $\boldsymbol{z}$ the output vector of network before applying softmax
- y the index of the correct class

$$\mathcal{L}(f(\boldsymbol{x};\boldsymbol{w}), y) = -\log p(y|\boldsymbol{x};\mathbb{W})$$

$$= -\log \frac{e^{z_y}}{\sum_{j=1}^{K} e^{z_j}}$$

$$= -z_y + \log \sum_{j=1}^{K} e^{z_j}$$

NLL + Softmax – example 2

$$\boldsymbol{z} = \begin{bmatrix} 1 \\ 2 \\ 0.5 \\ 6 \end{bmatrix}$$

- If the correct class is the first one, $y = 0$:

  $\mathcal{L} = -1 + \log(e^1 + e^2 + e^{0.5} + e^6) = -1 + 6.02 = \mathbf{5.02}$

- If the correct class is the third one, $y = 2$:

  $\mathcal{L} = -0.5 + \log(e^1 + e^2 + e^{0.5} + e^6) = -0.5 + 6.02 = \mathbf{5.52}$

- If the correct class is the fourth one, $y = 3$:

  $\mathcal{L} = -6 + \log(e^1 + e^2 + e^{0.5} + e^6) = -6 + 6.02 = \mathbf{0.02}$

NLL + Softmax – example 1

$$\boldsymbol{z} = \begin{bmatrix} 1 \\ 2 \\ 5 \\ 6 \end{bmatrix}$$

- If the correct class is the first one, $y = 0$:

  $\mathcal{L} = -1 + \log(e^1 + e^2 + e^5 + e^6) = -1 + 6.33 = \mathbf{5.33}$

- If the correct class is the third one, $y = 2$:

  $\mathcal{L} = -5 + \log(e^1 + e^2 + e^5 + e^6) = -5 + 6.33 = \mathbf{1.33}$

- If the correct class is the fourth one, $y = 3$:

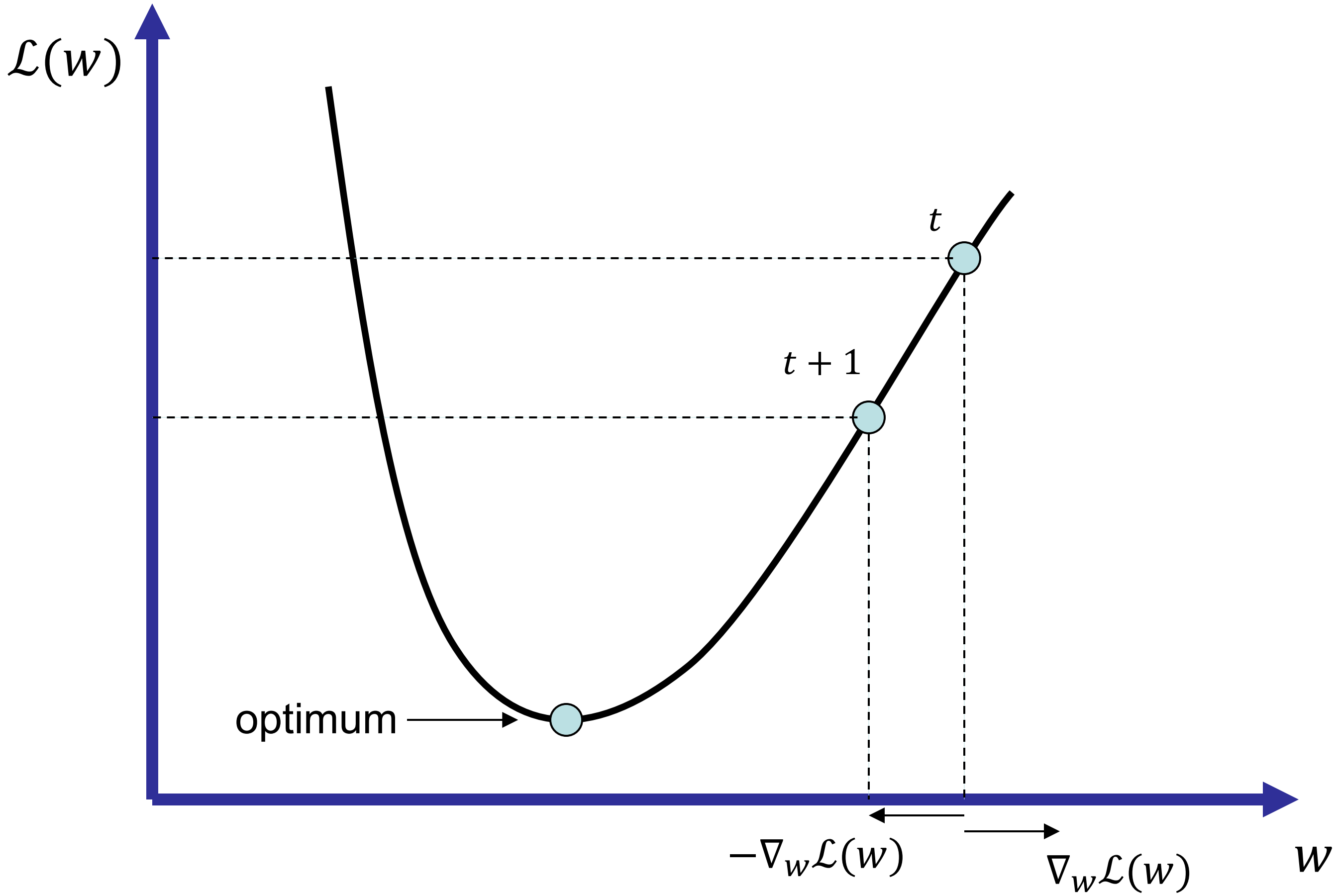
  $\mathcal{L} = -6 + \log(e^1 + e^2 + e^5 + e^6) = -6 + 6.33 = \mathbf{0.33}$

Agenda

- Introduction
- Non-linearities
- Forward pass & backpropagation
- Softmax & loss function
- **Optimization & regularization**

Stochastic Gradient Descent (SGD)

- For every $w \in \mathbb{W}$ and for m training data points



Stochastic Gradient Descent algorithm

- A set of parameters $\boldsymbol{w}$
- A **learning rate** η
- Loop until some exit criteria are met
  - Sample a **minibatch** of m data points from $\mathcal{D}$
  - Compute gradient (vectors) of parameters:

$$\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{w}} \sum_i \mathcal{L}(f(\boldsymbol{x}^{(i)}; \boldsymbol{w}), y^{(i)})$$

  - Update the parameters by taking a step in the opposite direction of the corresponding gradients:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \boldsymbol{g}$$

  - Reduce learning rate (**annealing**) if some criteria are met or based on a schedule

Sampling size

- If only one data point is used in every step; $m = 1$
  - Fast
  - learns **online**
  - Training can become unstable with a lot of fluctuations

- If all data points are used in every step; $m = N$
  - Also called **Batch Gradient Descent**
  - Training can take very long time

- If m is between these
  - Also called **Mini-Batch Gradient Descent**
  - Typical setting for training deep learning models

Other gradient-based optimizations

- Limitations of the mentioned SGD algorithms
  - Choosing learning rate is hard
  - Choosing annealing method/rate is hard
  - Same learning rate is applied to all parameters
  - Can get trapped in non-optimal local minima and saddle points

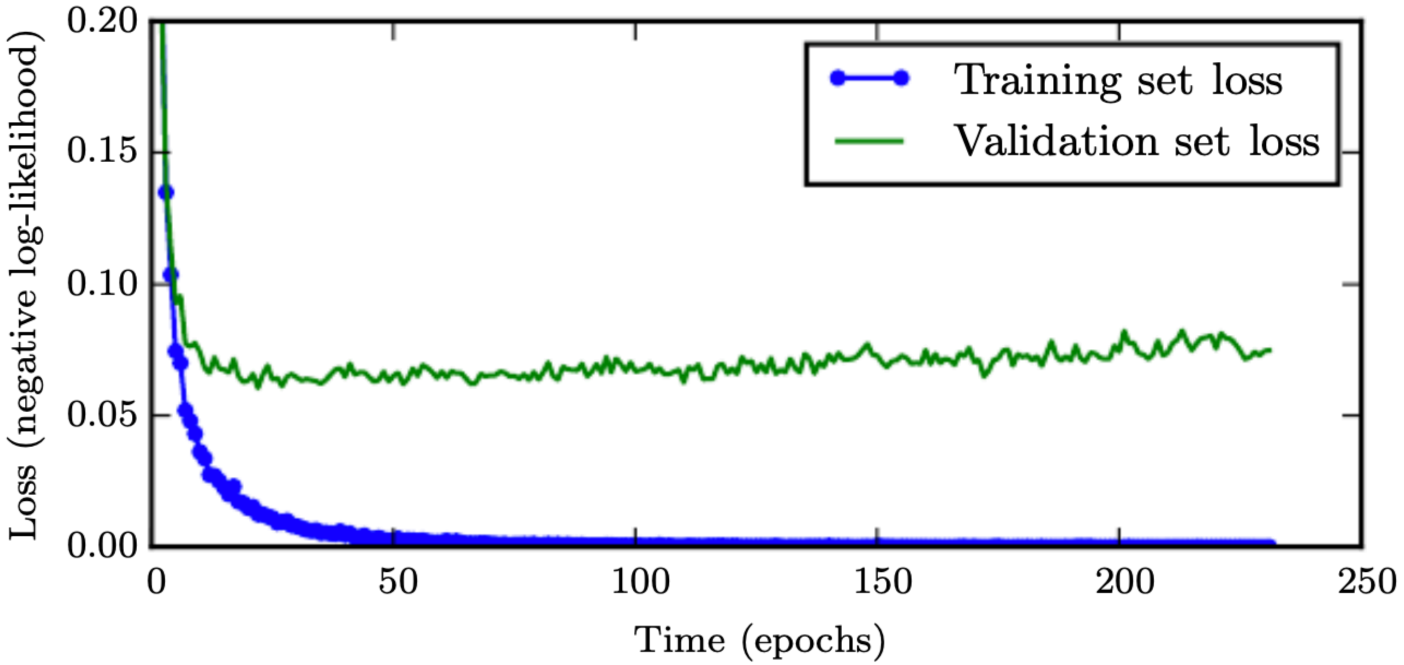
- Some other commonly used algorithms:
  - Nestrov accelerated gradient
  - Adagrad
  - Adam

Regularization techniques for neural networks and deep learning

- Parameter norm penalties (discussed in previous lecture)
- **Early stopping**
- **Dropout**
- Batch normalization
- Transfer learning
- Multitask learning
- Unsupervised / Semi-supervised pre-training
- Noise robustness
- Dataset augmentation
- Ensemble
- Adversarial training

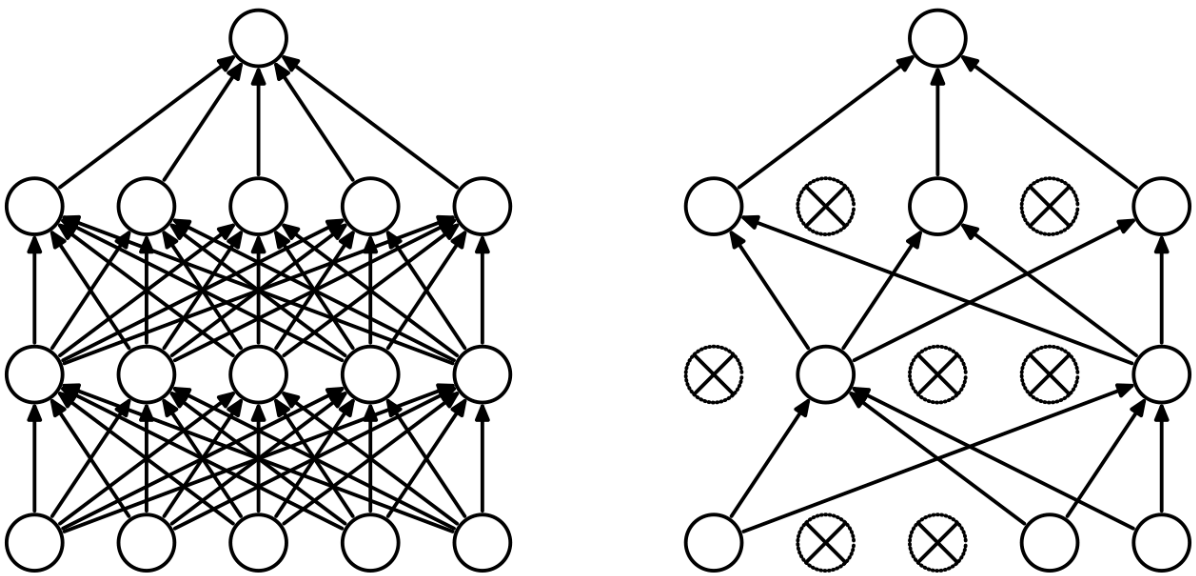
Early Stopping

- Run the model for several steps (epochs), and in each step evaluate the model on the validation set
- Store the model if the evaluation results improve
- At the end, take the stored model (with best validation results) as the final model



Dropout

- Key idea: prune neural network by removing some hidden units stochastically
- At training time for each data point:
  - Each hidden unit's output is multiplied to zero based on a dropout probability (like 0.6)

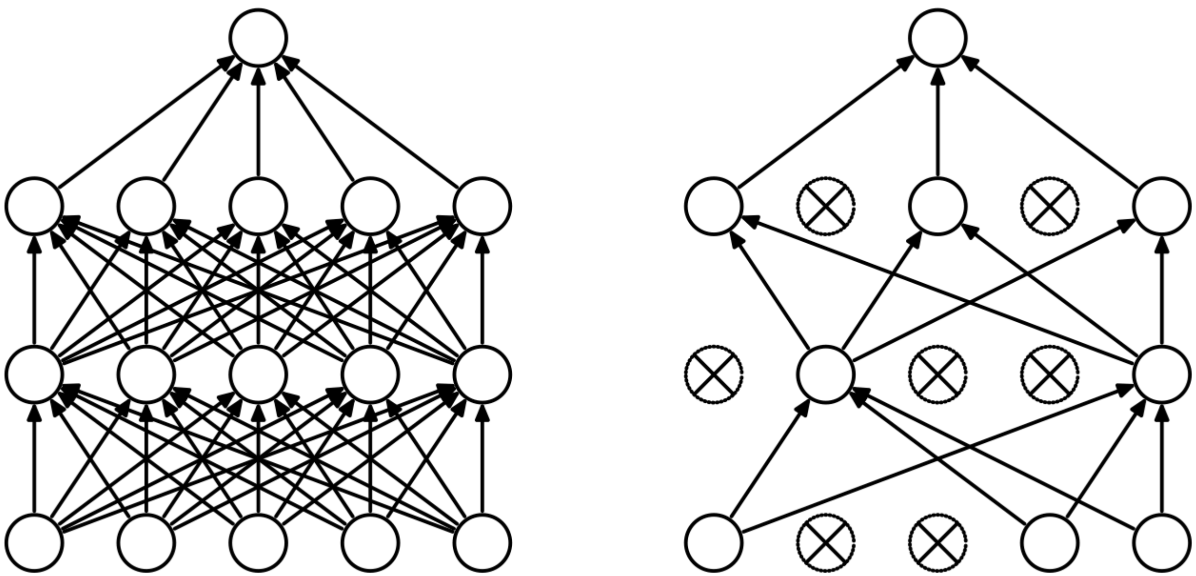


(a) Standard Neural Net (b) After applying dropout.

Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting", JMLR 2014

Dropout

- At test time:
  - All hidden units are used
  - The output of each hidden is multiplied to the dropout probability



(a) Standard Neural Net (b) After applying dropout.

Dropout – characteristics

- Computationally inexpensive but a powerful method
- Dropout can be viewed as a geometric average of an exponential number of networks → Ensemble
- Dropout prevents hidden units from forming co-dependencies amongst each other
- Every hidden unit learns to perform well regardless of other units