344.175 VL: Natural Language Processing
Neural Networks – a Walkthrough

Navid Rekab-saz

navid.rekabsaz@jku.at

JʏU
JOHANNES KEPLER
UNIVERSITY LINZ

Institute of
Computational
Perception

Agenda

- Artificial Neural Networks

- Forward pass and backpropagation

- Non-linearities, softmax, and loss

- Optimization and regularization

Notation – recap

- $a \rightarrow$ scalar

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

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

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

Probability

- Conditional probability, given two random variables X and Y:

$$P(Y|X)$$

- Probability distribution
  - For a discrete random variable Y with K states (classes)
    - $0 \leq P(Y_i) \leq 1$
    - $\sum_{i=1}^{K} P(Y_i) = 1$
  - E.g. with $K = 4$ states: $\begin{bmatrix} 0.2 & 0.3 & 0.45 & 0.05 \end{bmatrix}$

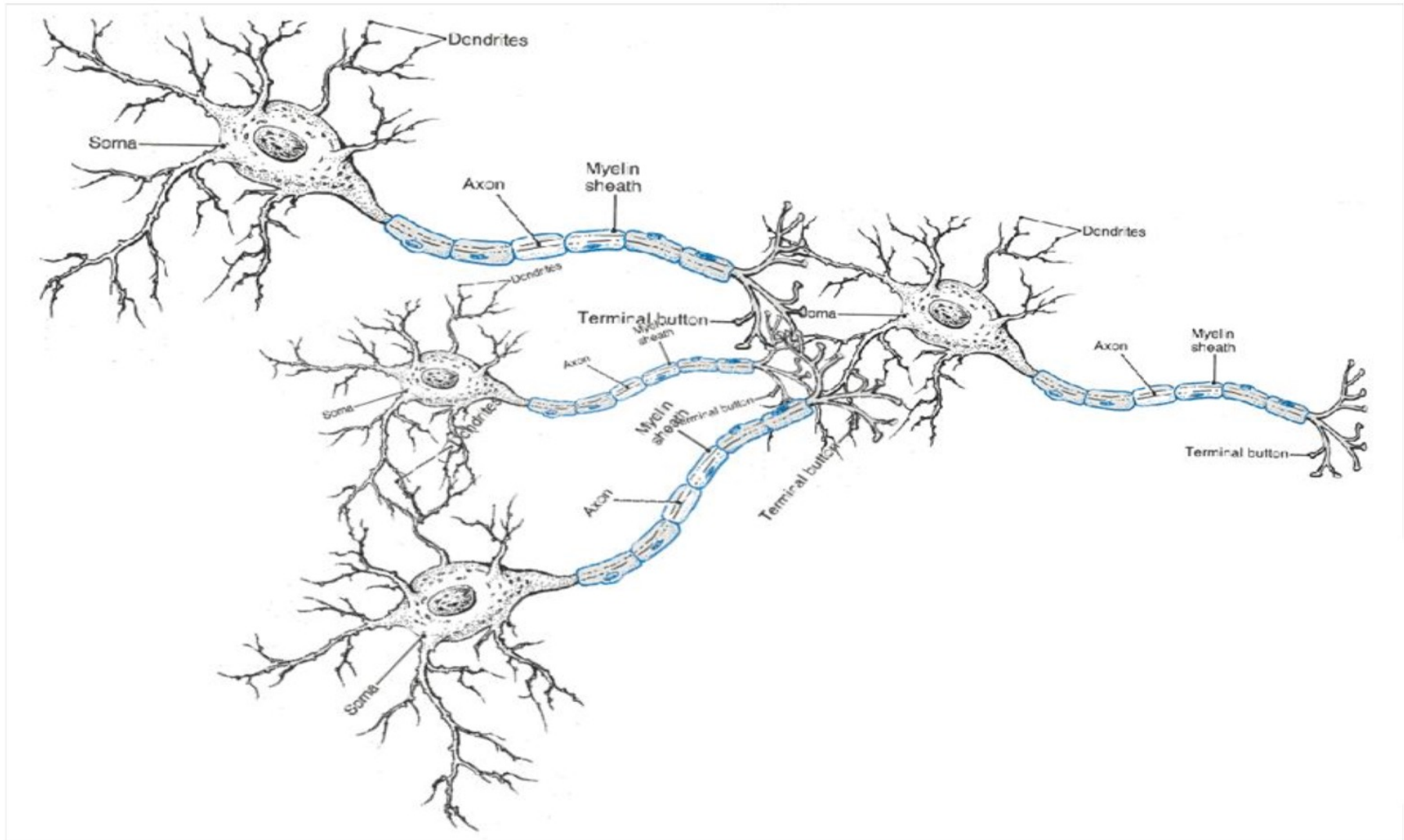- Expected value over a set $\mathcal{D}$

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

Note: The definition of expected value is not completely precise. Though, it suffices for our use in this lecture
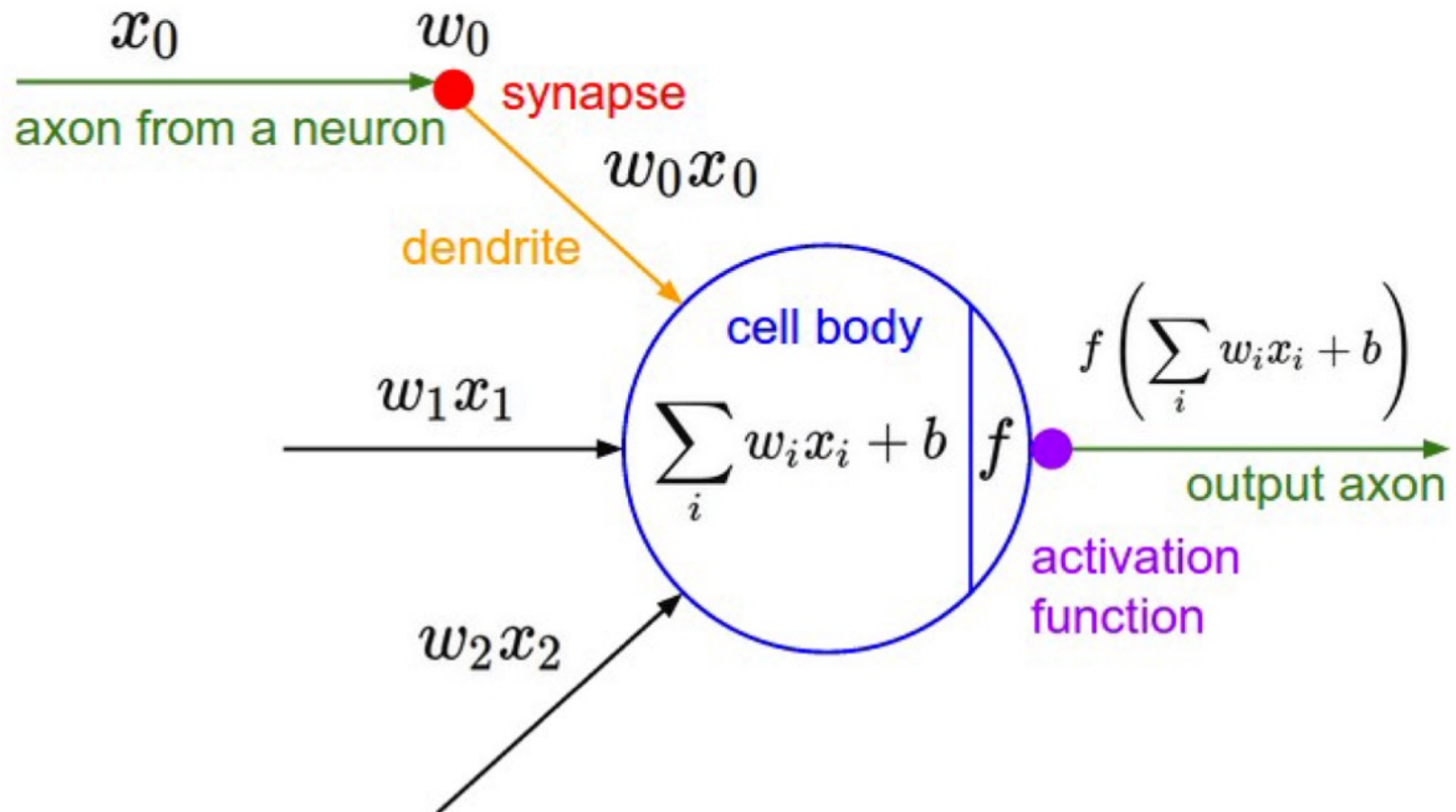
Agenda

- **Artificial Neural Networks**
- Forward pass and backpropagation
- Non-linearities, softmax, and loss
- Optimization and regularization
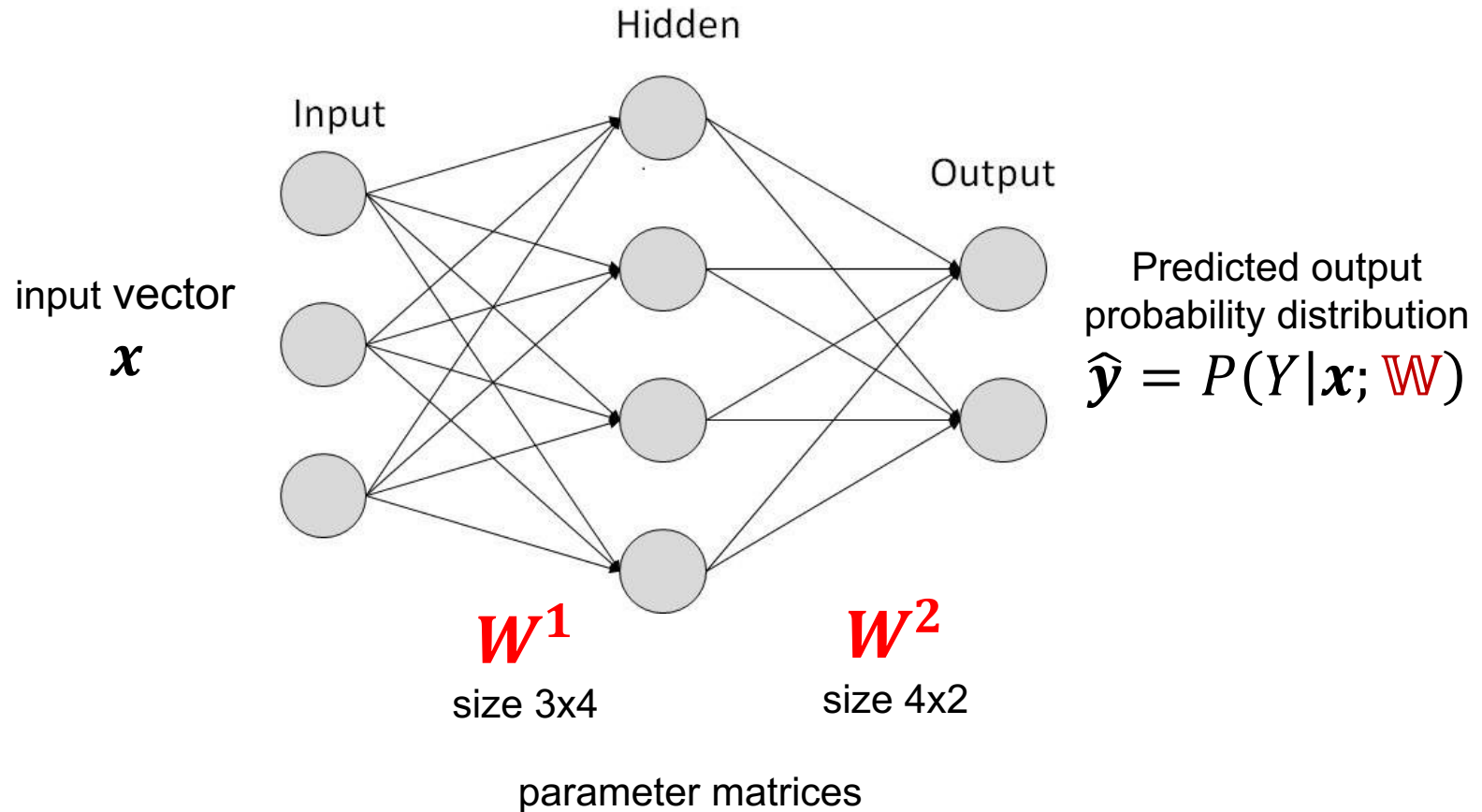
Neural Computation

An Artificial Neuron

x_0

w_0

synapse

axon from a neuron

$w_0 x_0$

dendrite

cell body

$w_1 x_1$

$$\sum_i w_i x_i + b$$

f

$f\left(\sum_i w_i x_i + b\right)$

output axon

activation function

$w_2 x_2$

Artificial Neural Networks

- Neural Networks are non-linear functions and universal approximators

- Neural networks can readily be defined as probabilistic models which estimate $P(Y|X)$

- Considering model parameter, $P(Y|X)$ can be written as $P(Y|\boldsymbol{x}; \mathbb{W})$
  - $\boldsymbol{x}$ is an input vector and $\mathbb{W}$ is the set of model parameters
  - The model's predicted probability distribution is:

$$\widehat{\boldsymbol{y}} = P(Y|\boldsymbol{x}; \mathbb{W})$$

A sample neural network (Multi Layer Perceptron)

Hidden

Input

Output

input vector
x

Predicted output
probability distribution
$\widehat{y} = P(Y|x; \mathbb{W})$

W^1

size 3x4

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 **(mini)batch** from training data $\mathcal{D}$

  - For each data point in the minibatch

    - **Forward pass**: given input $\boldsymbol{x}$ predict output distribution $\hat{\boldsymbol{y}} = P(Y|\boldsymbol{x}; \mathbb{W})$

  - Calculate **loss** function of the (mini)batch

  - Calculate the **gradient** of each parameter regarding the loss function using the **backpropagation** algorithm

  - **Update** parameters using their gradients

Agenda

- Artificial Neural Networks
- **Forward pass and backpropagation**
- Non-linearities, softmax, and loss
- Optimization and regularization

Forward pass

- Let's see how a neural network calculates the following function:

$$f(x; \mathbb{W}) = 2 * {w_2}^2 + 2 * x * w_1 + w_0$$

  - x is input and $\mathbb{W}$ is the tensor of parameters
  - Parameters are initialized with

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

- A neural network splits the function to subfunctions on each of basic operations, and rewrites it using new intermediary variables*:
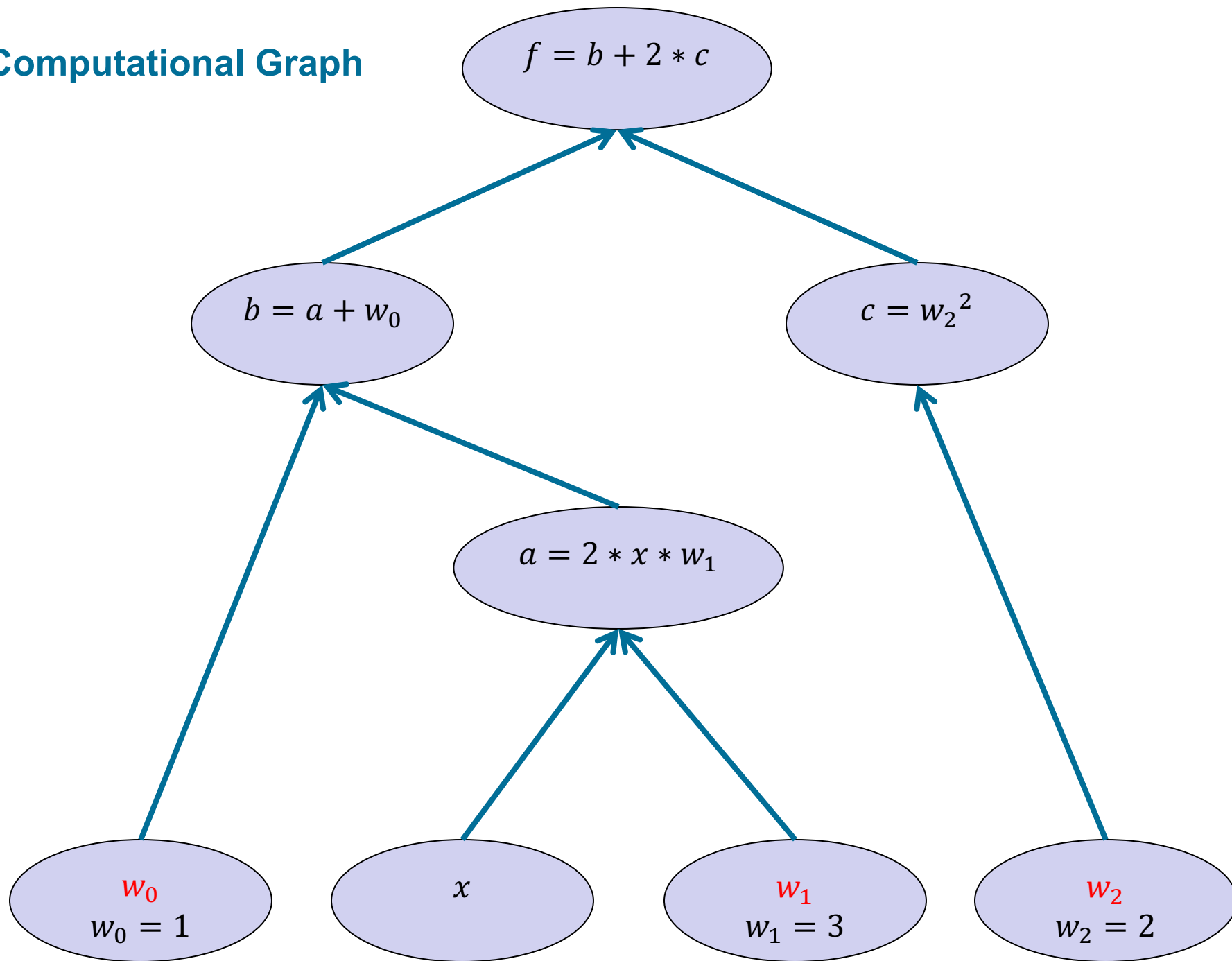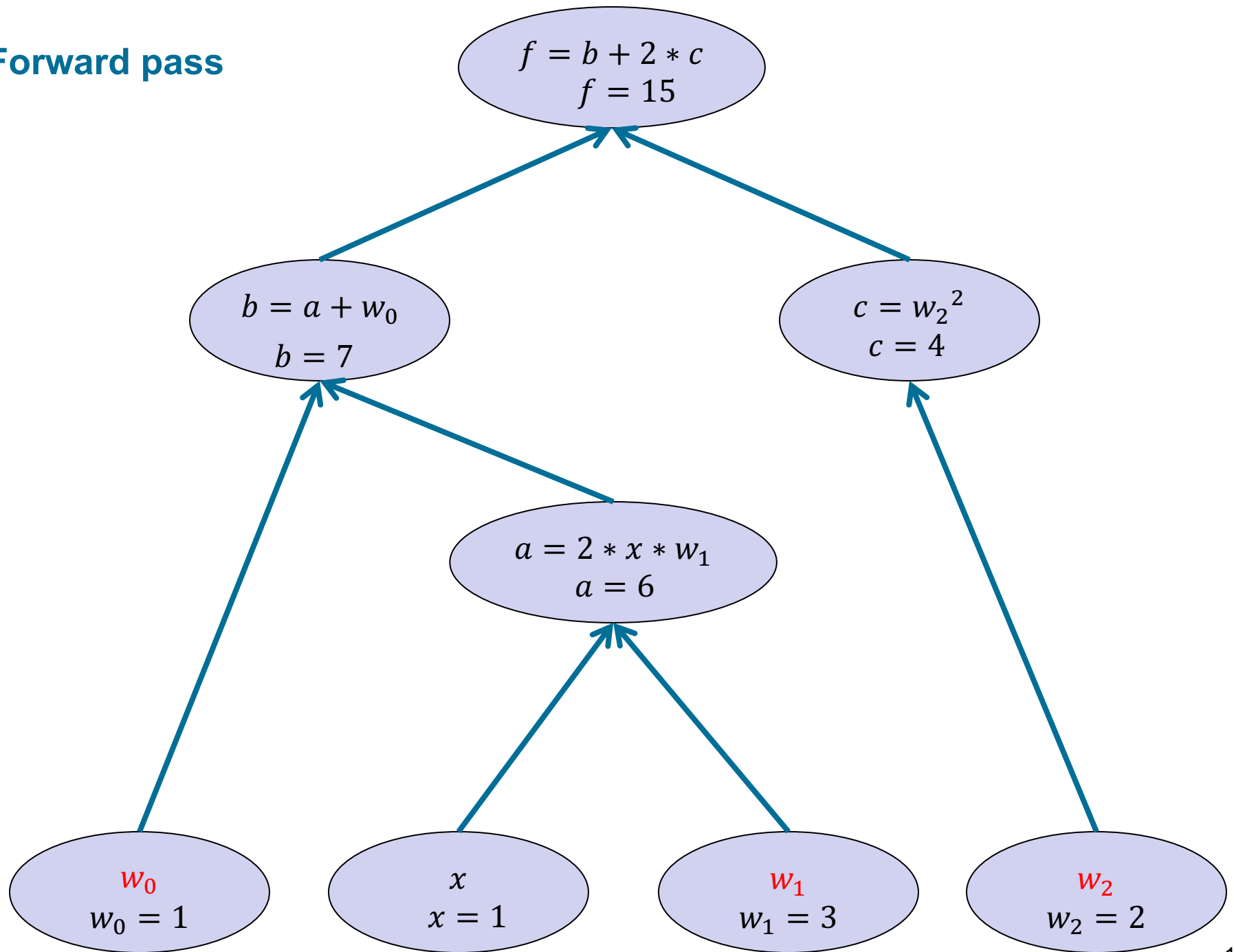
$$a = 2 * x * w_1$$
$$b = a + w_0$$
$$c = {w_2}^2$$
$$f = b + 2 * c$$

* To keep the example simple, the splitting is not applied to all basic operation

Computational Graph

$$f = b + 2 * c$$

$$b = a + w_0$$

$$c = w_2{}^2$$

$$a = 2 * x * w_1$$

w_0
$w_0 = 1$

x

w_1
$w_1 = 3$

w_2
$w_2 = 2$

13

Forward pass

$$f = b + 2 * c$$
$$f = 15$$

$$b = a + w_0$$
$$b = 7$$

$$c = w_2{}^2$$
$$c = 4$$

$$a = 2 * x * w_1$$
$$a = 6$$

w_0
$$w_0 = 1$$

x
$$x = 1$$

w_1
$$w_1 = 3$$

w_2
$$w_2 = 2$$

14

Towards backpropagation – Gradient vector

- To optimize the model's parameters, we need to calculate the gradient vector of f regarding parameters $\boldsymbol{w}$:

$$\nabla_{\boldsymbol{w}} f = \left[\frac{\partial f}{\partial w_0} \quad \frac{\partial f}{\partial w_1} \quad \frac{\partial f}{\partial w_2} \right]$$
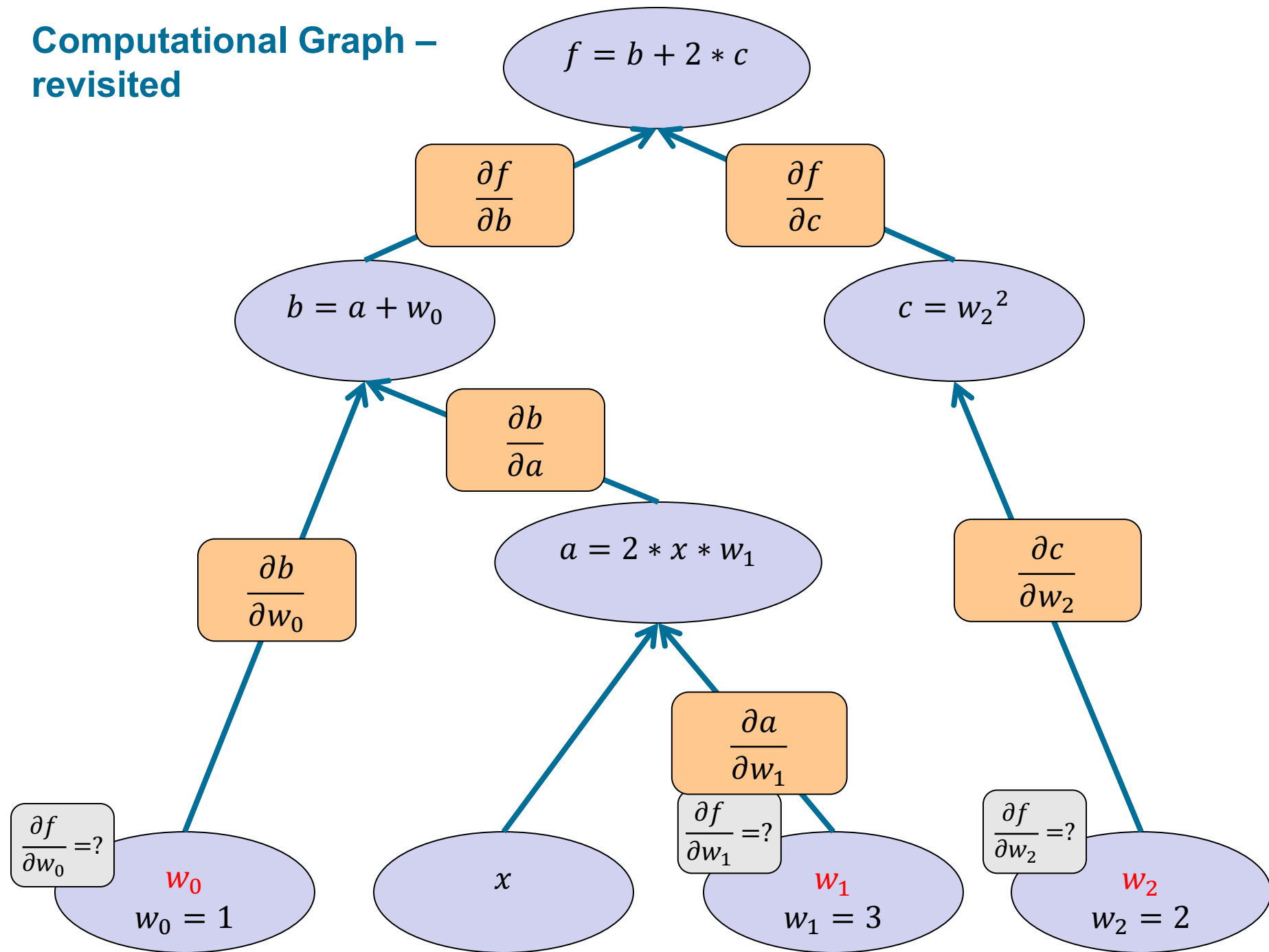
- The elements of the gradient vector are the partial derivatives of f to each parameter:

$$\frac{\partial f}{\partial w_0} = ?$$

$$\frac{\partial f}{\partial w_1} = ?$$

$$\frac{\partial f}{\partial w_2} = ?$$

Computational Graph – revisited

$$f = b + 2 * c$$

$$\frac{\partial f}{\partial b}$$

$$\frac{\partial f}{\partial c}$$

$$b = a + w_0$$

$$c = w_2{}^2$$

$$\frac{\partial b}{\partial a}$$

$$a = 2 * x * w_1$$

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

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

$$\frac{\partial a}{\partial w_1}$$

$$\frac{\partial f}{\partial w_0} = ?$$

$$\frac{\partial f}{\partial w_1} = ?$$

$$\frac{\partial f}{\partial w_2} = ?$$

w_0
$w_0 = 1$

x

w_1
$w_1 = 3$

w_2
$w_2 = 2$

16

Chain rule

- Gradient vector: $\nabla_{\boldsymbol{w}} f = \begin{bmatrix} \frac{\partial f}{\partial w_0} & \frac{\partial f}{\partial w_1} & \frac{\partial f}{\partial w_2} \end{bmatrix}$

- We can calculate partial derivatives using local derivates and chain rule:
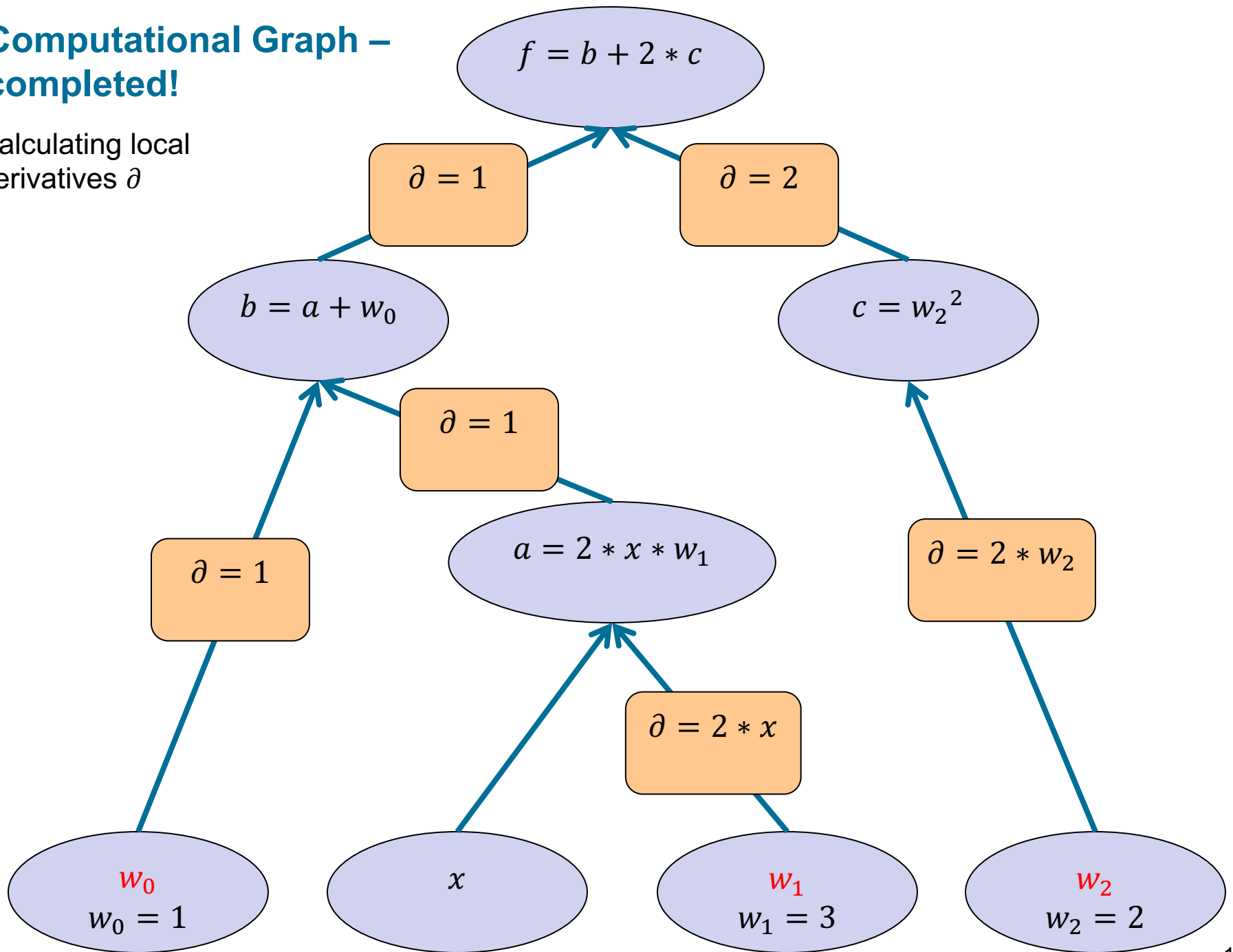
$$\frac{\partial f}{\partial w_0} = \frac{\partial f}{\partial b} \frac{\partial b}{\partial w_0}$$

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

$$\frac{\partial f}{\partial w_2} = \frac{\partial f}{\partial c} \frac{\partial c}{\partial w_2}$$
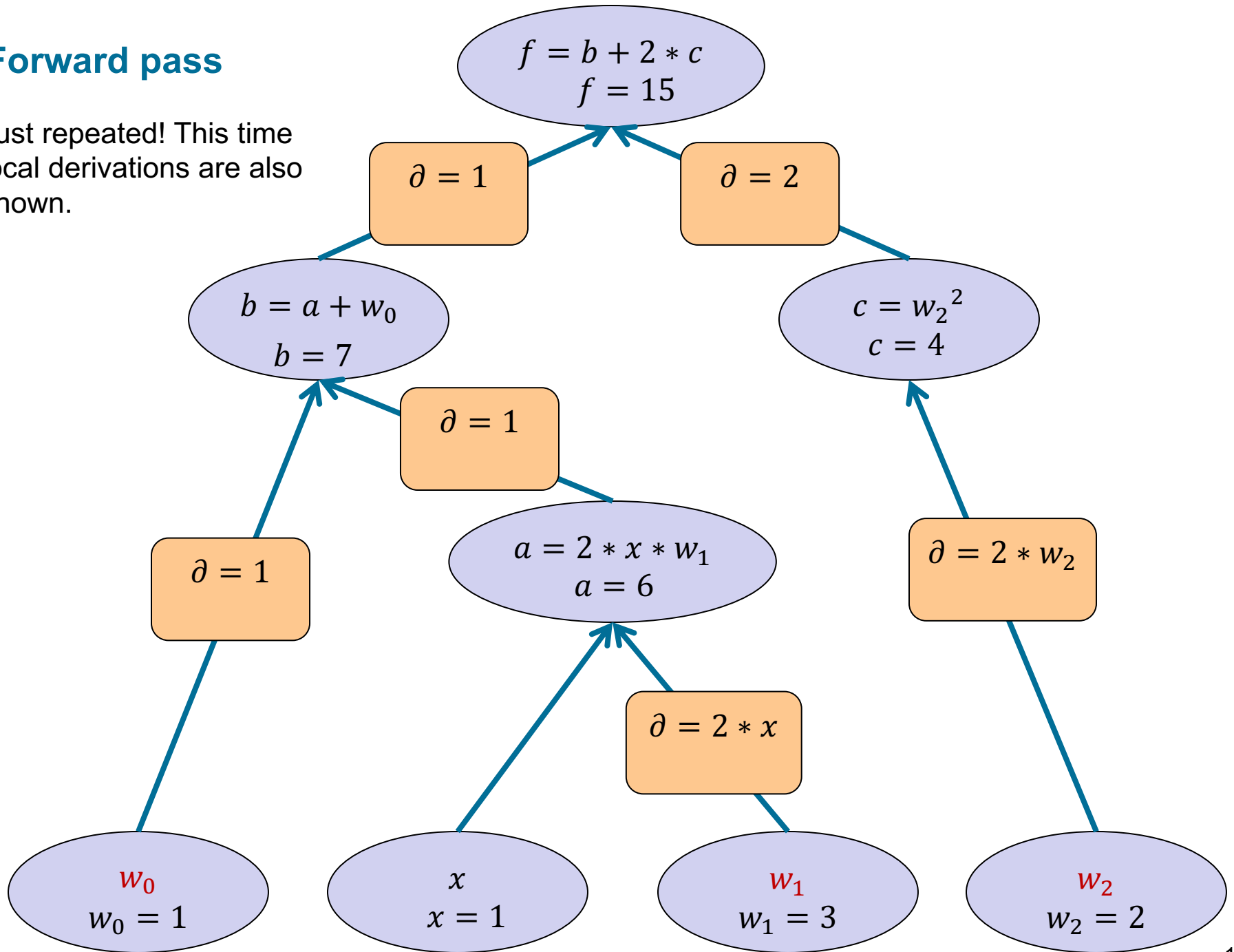
Computational Graph – completed!
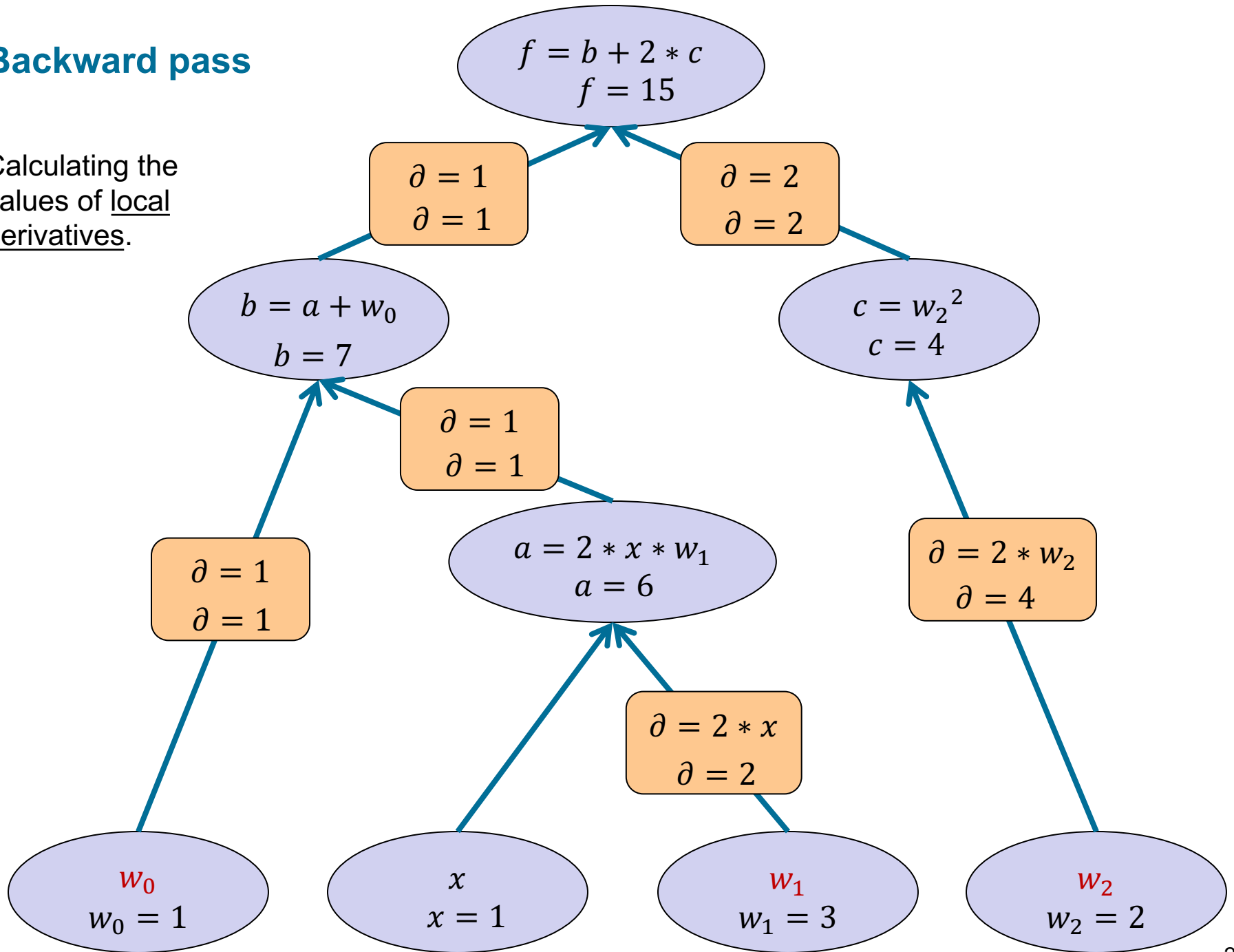
Calculating local derivatives ∂

$$f = b + 2 * c$$

$$\partial = 1$$

$$\partial = 2$$

$$b = a + w_0$$

$$c = {w_2}^2$$

$$\partial = 1$$

$$\partial = 1$$

$$a = 2 * x * w_1$$

$$\partial = 2 * w_2$$

$$\partial = 2 * x$$

$$w_0$$
$$w_0 = 1$$

$$x$$

$$w_1$$
$$w_1 = 3$$

$$w_2$$
$$w_2 = 2$$

Forward pass

Just repeated! This time local derivations are also shown.

$$f = b + 2 * c$$
$$f = 15$$

$$\partial = 1$$

$$\partial = 2$$

$$b = a + w_0$$
$$b = 7$$

$$c = {w_2}^2$$
$$c = 4$$

$$\partial = 1$$

$$\partial = 1$$

$$a = 2 * x * w_1$$
$$a = 6$$

$$\partial = 2 * w_2$$

$$\partial = 2 * x$$

$$w_0$$
$$w_0 = 1$$

$$x$$
$$x = 1$$

$$w_1$$
$$w_1 = 3$$

$$w_2$$
$$w_2 = 2$$

Backward pass

Calculating the values of <u>local</u> <u>derivatives</u>.

$$f = b + 2 * c$$
$$f = 15$$

$\partial = 1$
$\partial = 1$

$\partial = 2$
$\partial = 2$

$$b = a + w_0$$
$$b = 7$$

$$c = w_2{}^2$$
$$c = 4$$

$\partial = 1$
$\partial = 1$

$\partial = 1$
$\partial = 1$

$$a = 2 * x * w_1$$
$$a = 6$$

$\partial = 2 * w_2$
$\partial = 4$

$\partial = 2 * x$
$\partial = 2$

w_0
$w_0 = 1$

x
$x = 1$

w_1
$w_1 = 3$

w_2
$w_2 = 2$

Backward pass

Calculating the values of <u>partial</u> <u>derivatives</u>.

$$f = b + 2 * c$$
$$f = 15$$

$$\partial = 1$$
$$\partial = 1$$

$$\partial = 2$$
$$\partial = 2$$

$$b = a + w_0$$
$$b = 7$$

$$c = w_2{}^2$$
$$c = 4$$

$$\partial = 1$$
$$\partial = 1$$

$$\partial = 1$$
$$\partial = 1$$

$$a = 2 * x * w_1$$
$$a = 6$$

$$\partial = 2 * w_2$$
$$\partial = 4$$

$$\partial = 2 * x$$
$$\partial = 2$$

$$\frac{\partial f}{\partial w_0} = 1$$

$$w_0$$
$$w_0 = 1$$

$$x$$
$$x = 1$$

$$\frac{\partial f}{\partial w_1} = 2$$

$$w_1$$
$$w_1 = 3$$

$$\frac{\partial f}{\partial w_2} = 8$$

$$w_2$$
$$w_2 = 2$$

21
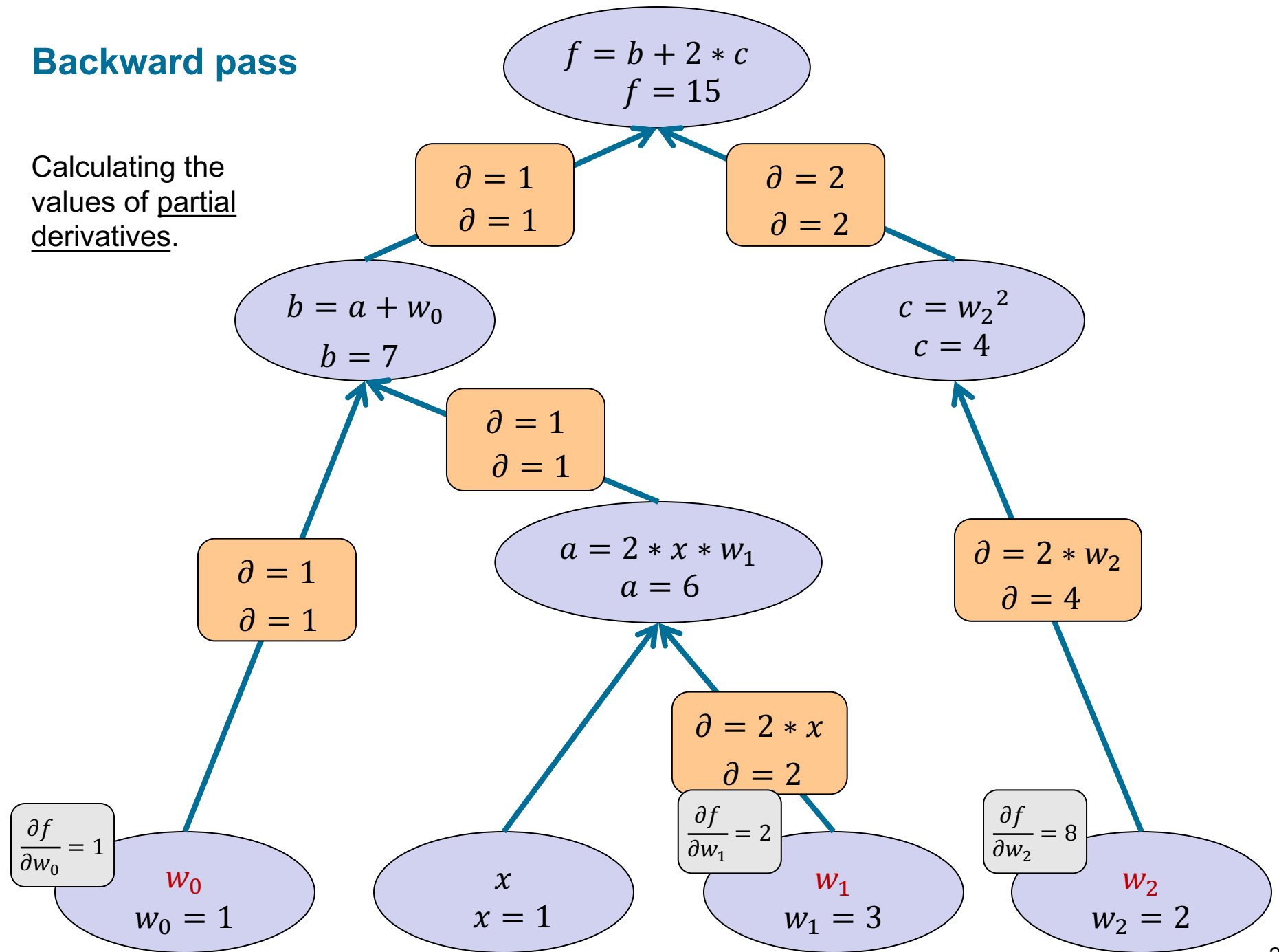
Backpropagation

Calculating partial derivatives:

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

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

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

Agenda

- Artificial Neural Networks
- Forward pass and backpropagation
- **Non-linearities, softmax, and loss**
- Optimization and regularization

An Artificial Neuron
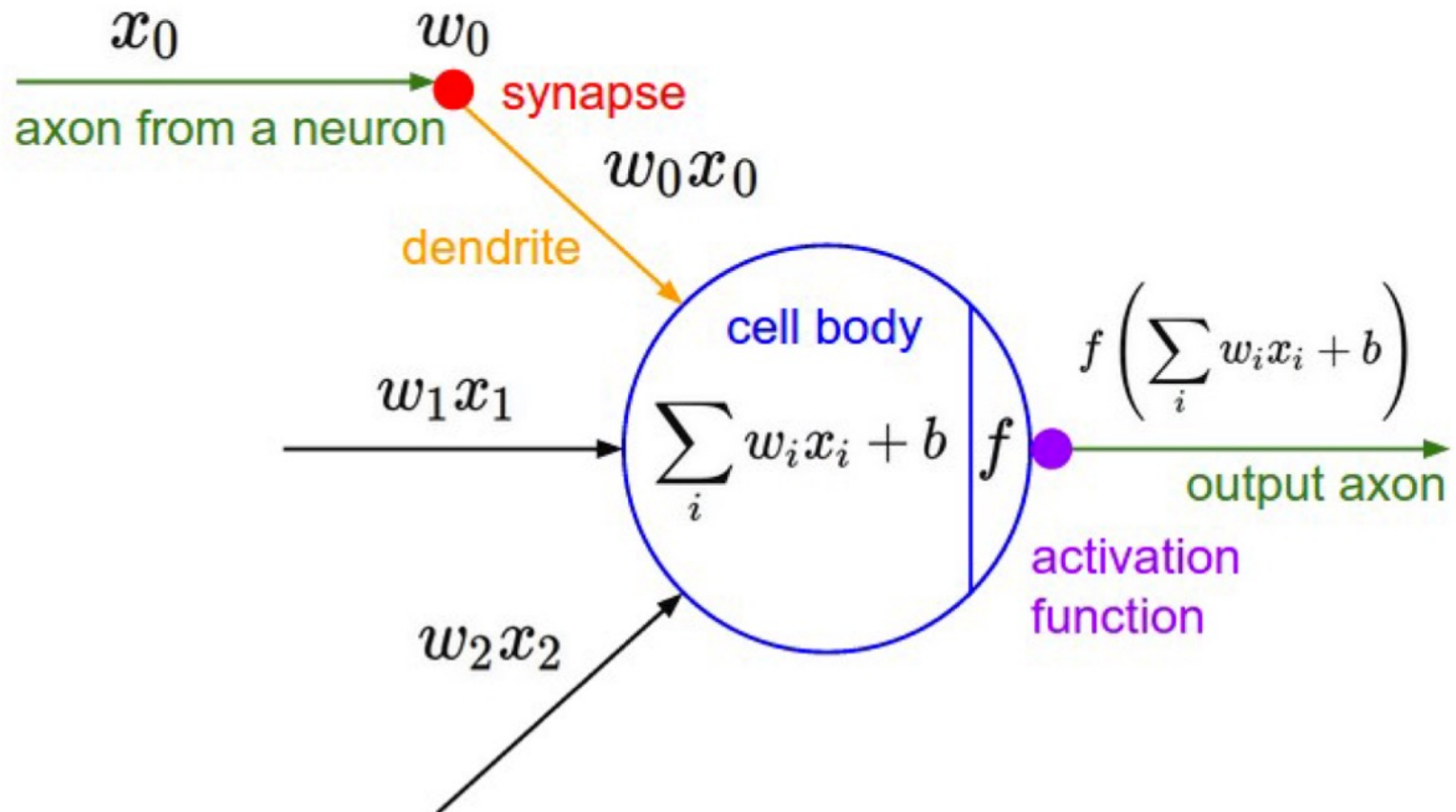
Linear

$$f(x) = x$$
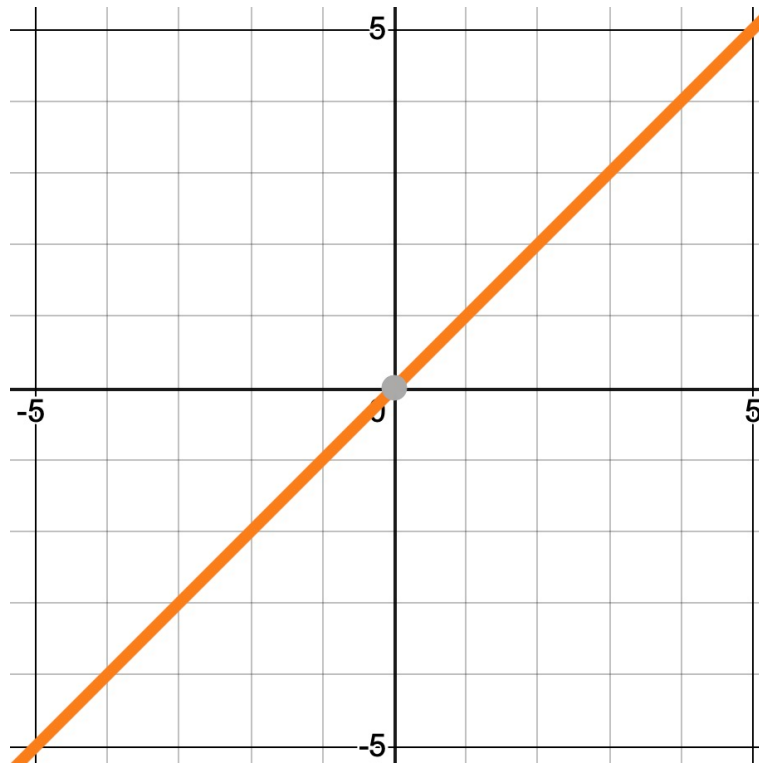
Sigmoid

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

- squashes input between 0 and 1
- Output becomes like a probability value
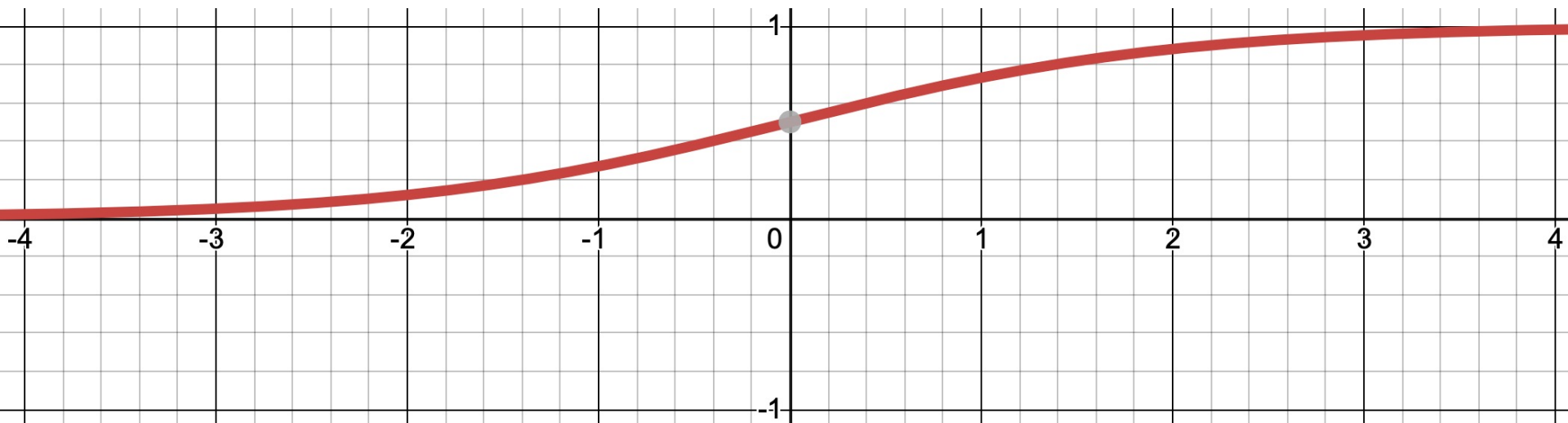
Hyperbolic Tangent (Tanh)

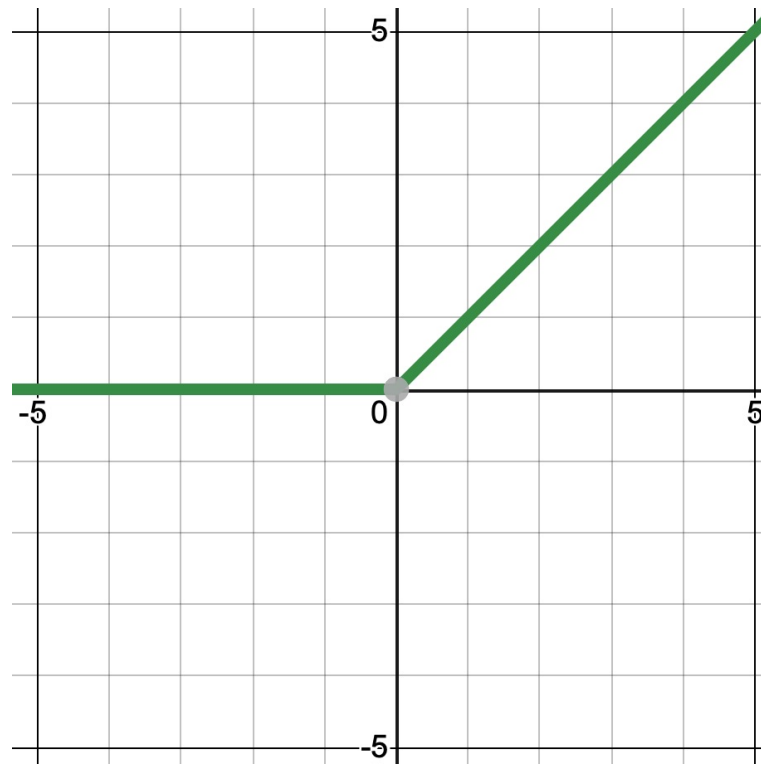$$f(x) = \tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

- squashes input between -1 and 1
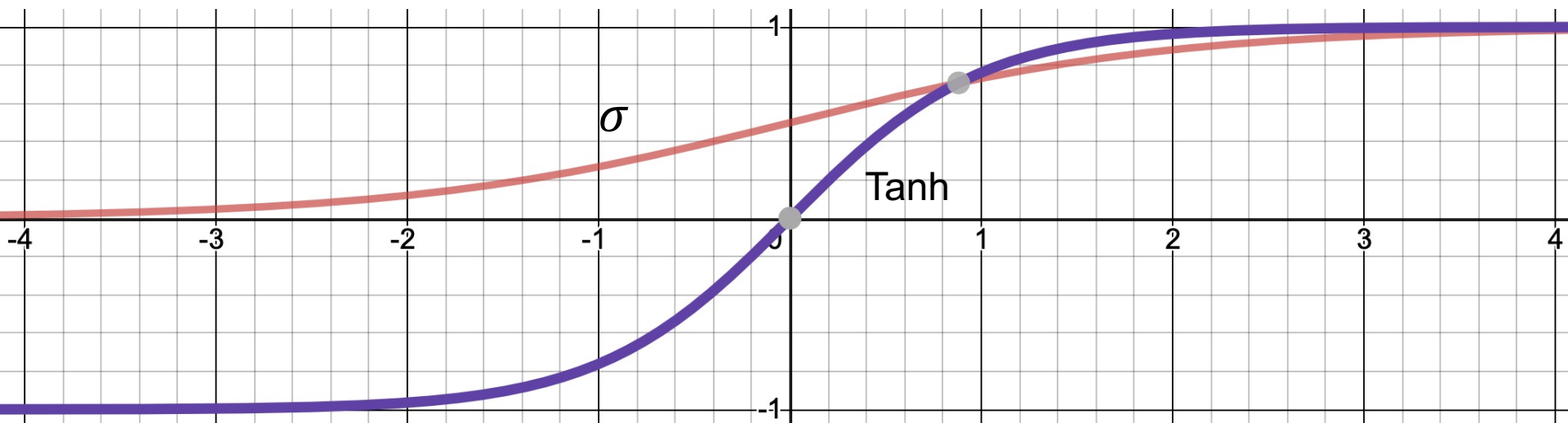
Rectified Linear Unit (ReLU)

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

- fits to deep architectures, as it prevents vanishing gradient

Examples

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

- Linear transformation xW:

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

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

$$\mathrm{ReLU}(\begin{bmatrix} 0.5 & -0.5 & 2 & 12 & -4 \end{bmatrix}) = \begin{bmatrix} \mathbf{0.5} & \mathbf{0.0} & \mathbf{2} & \mathbf{12} & \mathbf{0.0} \end{bmatrix}$$

- Non-linear transformation $\sigma(xW)$:

$$\sigma(\begin{bmatrix} 0.5 & -0.5 & 2 & 12 & -4 \end{bmatrix}) = \begin{bmatrix} \mathbf{0.62} & \mathbf{0.37} & \mathbf{0.88} & \mathbf{0.99} & \mathbf{0.11} \end{bmatrix}$$

- Non-linear transformation $\tanh(xW)$:

$$\tanh(\begin{bmatrix} 0.5 & -0.5 & 2 & 12 & -4 \end{bmatrix}) = \begin{bmatrix} \mathbf{0.46} & \mathbf{-0.46} & \mathbf{0.96} & \mathbf{0.99} & \mathbf{-0.99} \end{bmatrix}$$

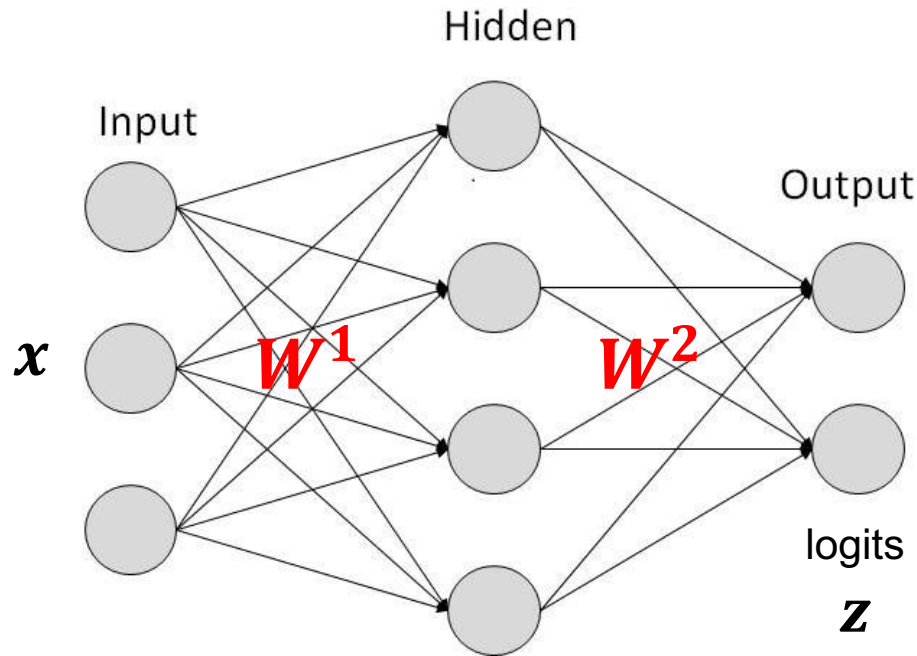# Softmax

- As discussed, neural networks can readily turn to probabilistic models

- To do it, we need to transform the output vector z of a neural network with K output classes to a probability distribution
  - In the context of neural networks, z is usually called **logits**

- softmax turns a vector to a probability distribution
  - z could be the output vector of a neural network

$$\text{softmax}(z)_l = \frac{e^{z_l}}{\sum_{i=1}^{K} e^{z_i}}$$

normalization term

A sample neural network

Hidden

Input

Output

x

$\textcolor{red}{W^1}$

$\textcolor{red}{W^2}$

Predicted output
probability distribution

$\hat{y} = P(Y|x; \textcolor{red}{\mathbb{W}})$

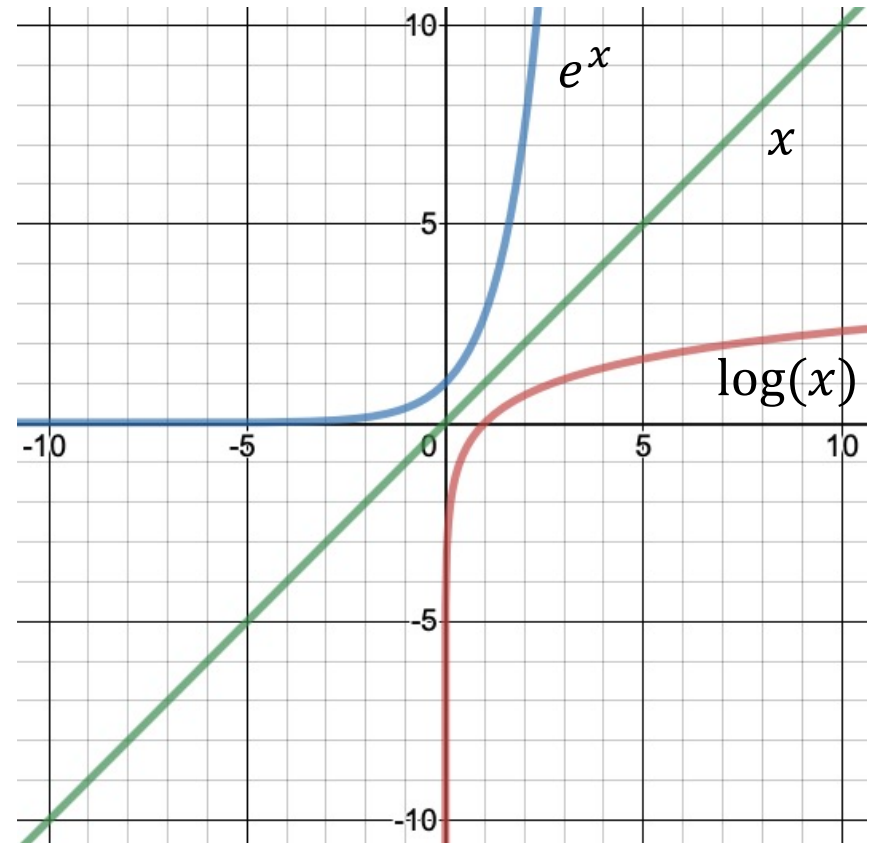$= \text{softmax}(z)$

logits

z

Softmax – example

$$K = 4 \text{ classes}$$

$$\text{softmax}(\mathbf{z})_l = \frac{e^{z_l}}{\sum_{i=1}^{K} e^{z_i}}$$

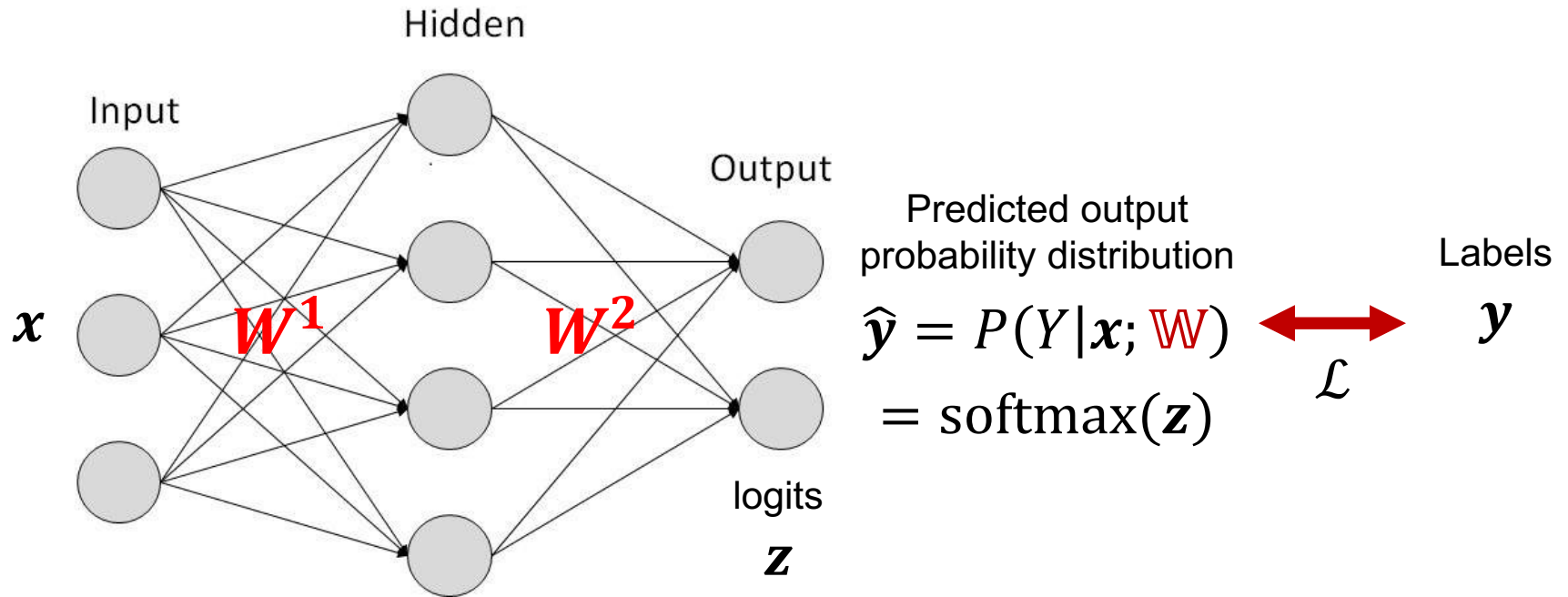$$\mathbf{z} = \begin{bmatrix} 1 \\ 2 \\ 5 \\ 6 \end{bmatrix}$$

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

Softmax characteristics

- The exponential function in softmax makes the maximum becomes much higher than the others

- Softmax identifies the "*max*" but in a "*soft*" way!

- Softmax imposes competition between the predicted output values, as in fact "*winner takes (almost) all!*"
  - Winner-takes-all is the case when one value is 1 and the rest are 0
  - Softmax provides a soft distribution of winner-takes-all
  - This resembles the competition between nearby neurons in the cortex

Sample neural network

Hidden

Input

Output

x

$\boldsymbol{W^1}$ $\boldsymbol{W^2}$

Predicted output
probability distribution

$\hat{\boldsymbol{y}} = P(Y|\boldsymbol{x}; \mathbb{W})$

$= \mathrm{softmax}(\boldsymbol{z})$

logits

$\boldsymbol{z}$

Labels

$\boldsymbol{y}$

$\mathcal{L}$

Cross Entropy Loss

- Given a classification task with K classes
  - known as multi-class classification
- $\widehat{\boldsymbol{y}} \rightarrow$ predicted probability distribution of the classes
- $\boldsymbol{y} \rightarrow$ actual probability distribution of the classes (labels)
- Cross Entropy loss is defined as:

$$\mathcal{L} = -\mathbb{E}_{\mathcal{D}} \sum_{i=1}^{K} y_i \log \hat{y}_i$$

  - $\mathcal{D} \rightarrow$ the set of training data
- In neural networks, we can write it as:

$$\mathcal{L}(\mathbb{W}) = -\mathbb{E}_{\mathcal{D}} \sum_{i=1}^{K} y_i \log P(Y_i | \boldsymbol{x}; \mathbb{W})$$

Cross Entropy Loss – example 1

- A multi-label scenario:

$$\widehat{\boldsymbol{y}} = \begin{bmatrix} 0.004 \\ 0.013 \\ 0.264 \\ 0.717 \end{bmatrix} \qquad \boldsymbol{y} = \begin{bmatrix} 0 \\ 0.25 \\ 0 \\ 0.75 \end{bmatrix}$$

$$\mathcal{L} = -\sum_{i=1}^{K} y_i \log \widehat{y}_i$$

$$\mathcal{L} = -(0 \times \log 0.004 + 0.25 \times \log 0.013 + 0 \times \log 0.264 + 0.75 \times \log 0.717)$$

$$\mathcal{L} = -(0 - 0.471 + 0 - 0.108)$$

$$\mathcal{L} = 0.579$$

Cross Entropy Loss – example 2

- A single-label scenario:

$$\hat{\boldsymbol{y}} = \begin{bmatrix} 0.004 \\ 0.013 \\ 0.264 \\ 0.717 \end{bmatrix} \qquad \boldsymbol{y} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\mathcal{L} = -\sum_{i=1}^{K} y_i \log \hat{y}_i$$

$$\mathcal{L} = -(0 \times \log 0.004 + 0 \times \log 0.013 + 0 \times \log 0.264 + 1 \times \log 0.717)$$

$$\mathcal{L} = -(0 + 0 + 0 - 0.144)$$

$$\mathcal{L} = 0.144$$

Negative Log Likelihood (NLL) Loss

- Single-label classification is the most common scenario

- In this case, we can simplify Cross Entropy formulation to

$$\mathcal{L}(\mathbb{W}) = -\mathbb{E}_{\mathcal{D}} \sum_{i=1}^{K} y_i \log P(Y_i | \boldsymbol{x}; \mathbb{W}) = -\mathbb{E}_{\mathcal{D}} \log P(Y_l | \boldsymbol{x}; \mathbb{W})$$

  - where l is the index of the correct class

- This loss function is known as Negative Log Likelihood (NLL)
  - NLL is a special case of Cross Entropy

NLL + softmax

- What happens when we use NLL and softmax in the output layer of a neural network?

$$\mathcal{L}(\mathbb{W}) = -\mathbb{E}_{\mathcal{D}} \log P(Y_l | \boldsymbol{x}; \mathbb{W}) = -\mathbb{E}_{\mathcal{D}} \log \text{softmax}(\boldsymbol{z})_l$$

$\boldsymbol{z} \rightarrow$ output vector before softmax (logits)

$$\mathcal{L}(\mathbb{W}) = -\mathbb{E}_{\mathcal{D}} \log \frac{e^{z_l}}{\sum_{i=1}^{K} e^{z_i}} = -\mathbb{E}_{\mathcal{D}} \left[\log e^{z_l} - \log \sum_{i=1}^{K} e^{z_i} \right]$$

$$\mathcal{L}(\mathbb{W}) = -\mathbb{E}_{\mathcal{D}} \left[z_l - \log \sum_{i=1}^{K} e^{z_i} \right]$$

This term is (almost) equal to $\max(\boldsymbol{z})$

NLL + softmax – example 1

$$\mathcal{L} = -\left[z_l - \log \sum_{i=1}^{K} e^{z_i} \right]$$

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

- If the correct class is the first one, $l = 1$:

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

- If the correct class is the third one, $l = 3$:

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

- If the correct class is the fourth one, $l = 4$:

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

NLL + softmax – example 2

$$\mathcal{L} = -\left[z_l - \log \sum_{i=1}^{K} e^{z_i} \right]$$

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

- If the correct class is the first one, $l = 1$:

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

- If the correct class is the third one, $l = 3$:

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

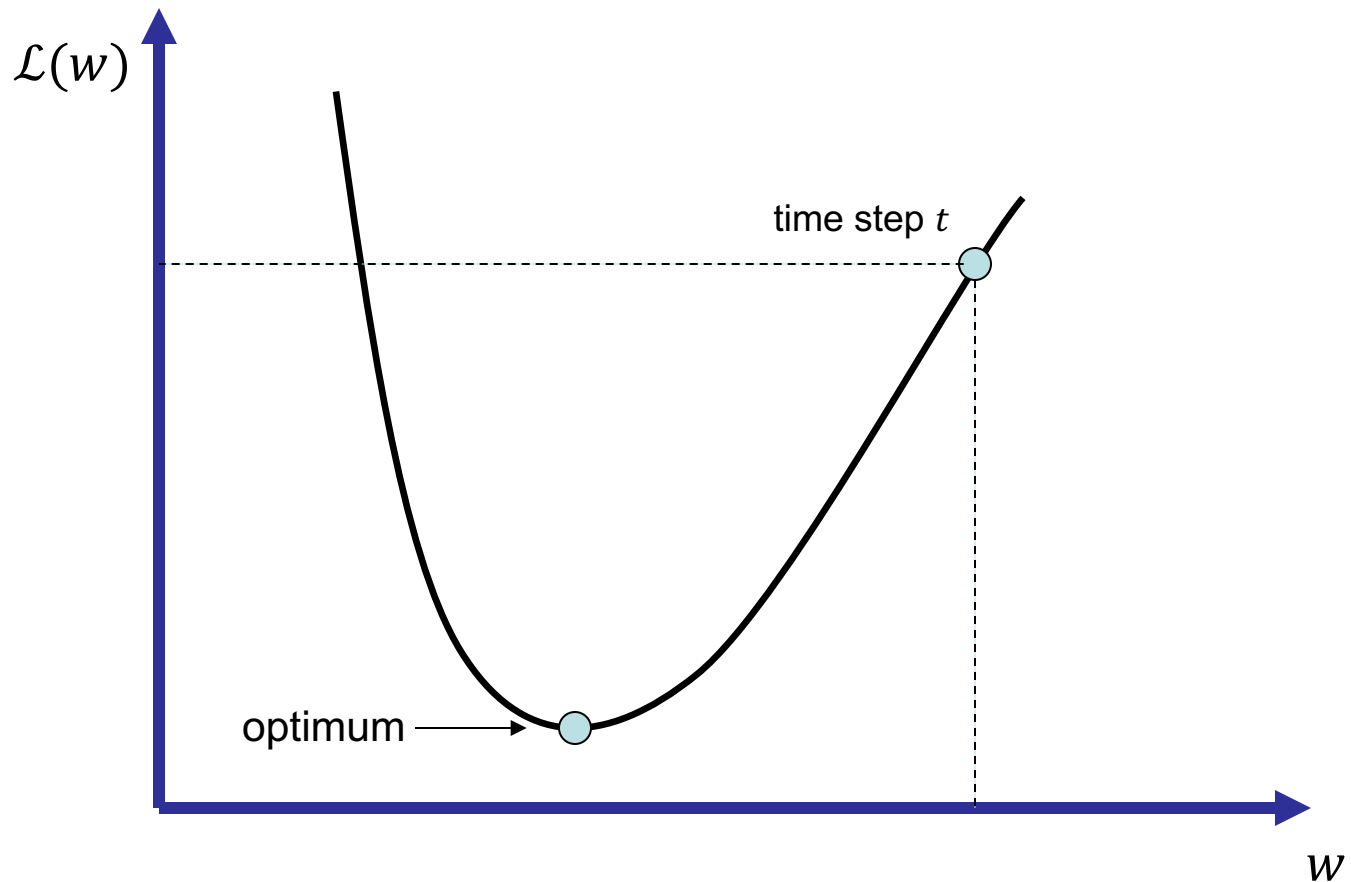- If the correct class is the fourth one, $l = 4$:

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

Agenda

- Artificial Neural Networks
- Forward pass and backpropagation
- Non-linearities, Softmax, and loss
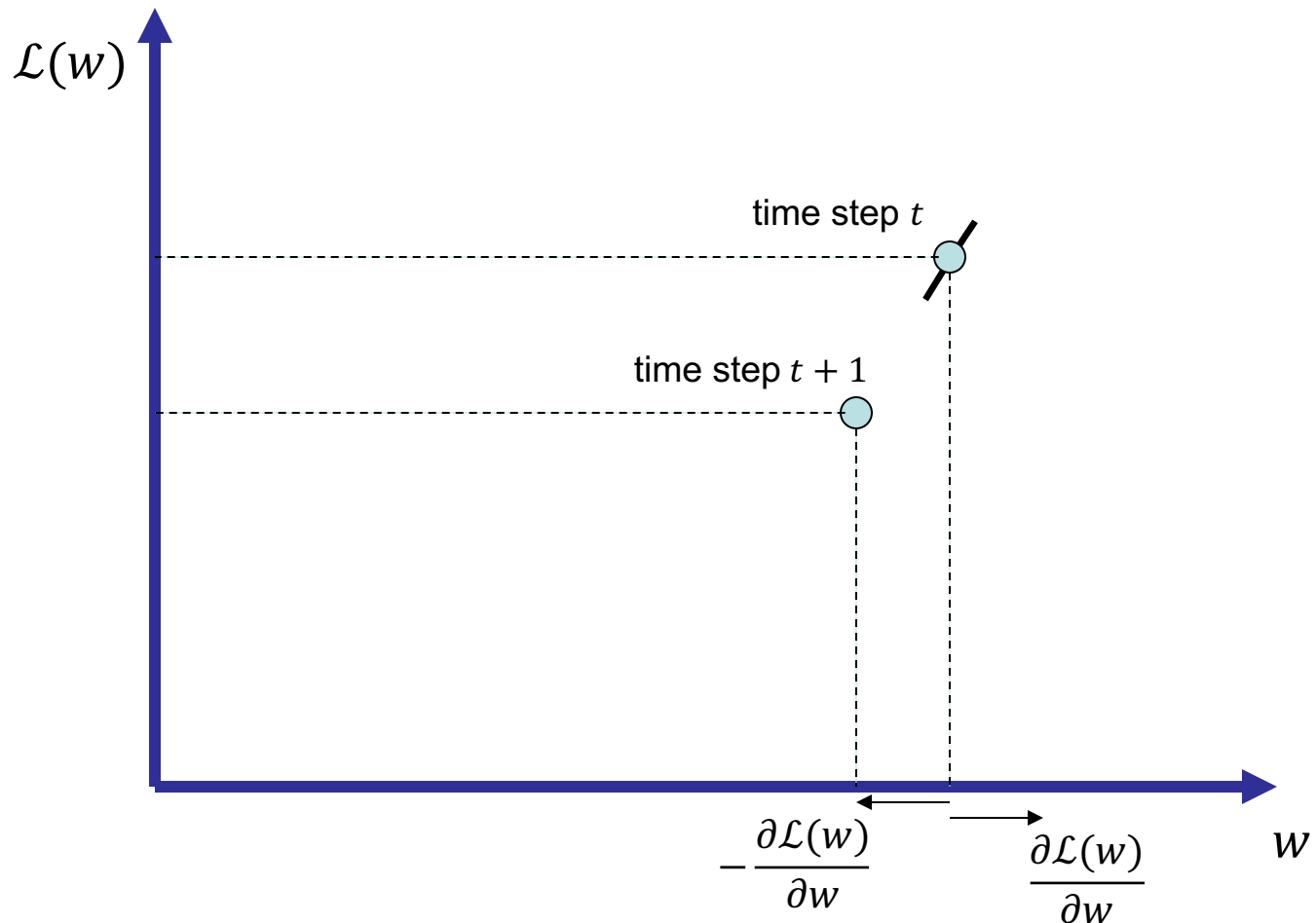- **Optimization & regularization**

Stochastic Gradient Descent (SGD)

- For every $w \in \mathbb{W}$ and after calculating loss of one/some/all of data points …

Stochastic Gradient Descent (SGD)

- For every $w \in \mathbb{W}$ and after calculating loss of one/some/all of data points …

Stochastic Gradient Descent algorithm

- A model with a set of parameters $\mathbb{W}$ at time step $t \rightarrow \mathbb{W}^{(t)}$

- A **learning rate** η

- Loop until some exit criteria are met

  - $\widehat{\mathcal{D}}$ is a **minibatch** containing S data points, sampled from $\mathcal{D}$

  - Compute gradient tensor of parameters $\mathbb{G}$:

  $$\mathbb{G} \leftarrow \frac{1}{S} \nabla_{\mathbb{W}} \sum_{(\boldsymbol{x},y) \in \widehat{\mathcal{D}}} \mathcal{L}(\boldsymbol{x}, y; \mathbb{W})$$

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

  $$\mathbb{W}^{(t+1)} \leftarrow \mathbb{W}^{(t)} - \eta \mathbb{G}$$

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

Sampling (batch) size in (Stochastic) Gradient Descent

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

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

- If S is between these
  - Also called **Mini-Batch (Stochastic) Gradient Descent**
  - Typical setting for training deep learning models

Other gradient-based optimizations

- Some 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 penalty**
- **Early stopping**
- Dropout
- Batch normalization
- Transfer learning
- Multitask learning
- Unsupervised / Semi-supervised pre-training
- Noise robustness
- Dataset augmentation
- Ensemble
- Adversarial training

Parameter norm penalty

- Adds the norm of parameters to the loss function
- For instance, the squared L2 norm of parameters: $\|\mathbb{W}\|_2{}^2$

$$\|\mathbb{W}\|_2{}^2 = \left(\sqrt{\sum_{w \in \mathbb{W}} w^2}\right)^2 = \sum_{w \in \mathbb{W}} w^2$$

- Norm penalty in NLL loss:

$$\mathcal{L}(\mathbb{W}) = -\log P(Y_l|\boldsymbol{x}; \mathbb{W}) + \|\mathbb{W}\|_2{}^2$$

- This constraint forces the model to punish (decrease the values of) parameters with high values
  - Read more about L1, L2 norms here section 2.5

Early Stopping

- Run the model for several steps (epochs), and in each step evaluate the model on the <u>validation set</u>

- Store the model if the evaluation results improve

- At the end, take the stored model (with best validation results) as the final model