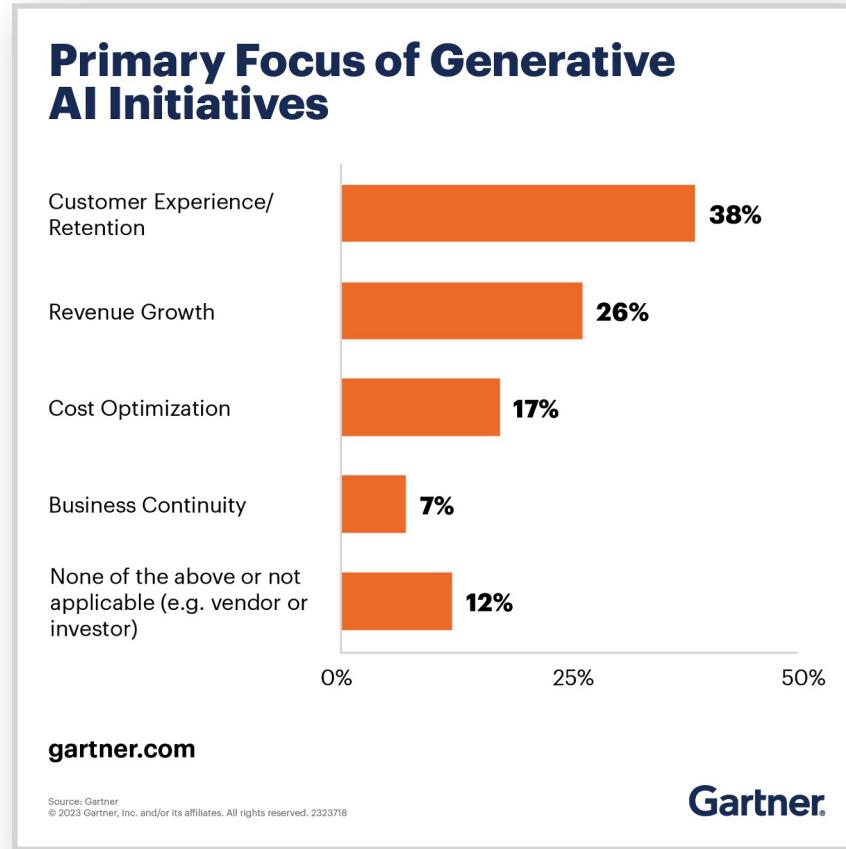




Generative AI

Introduction

Why Generative AI?

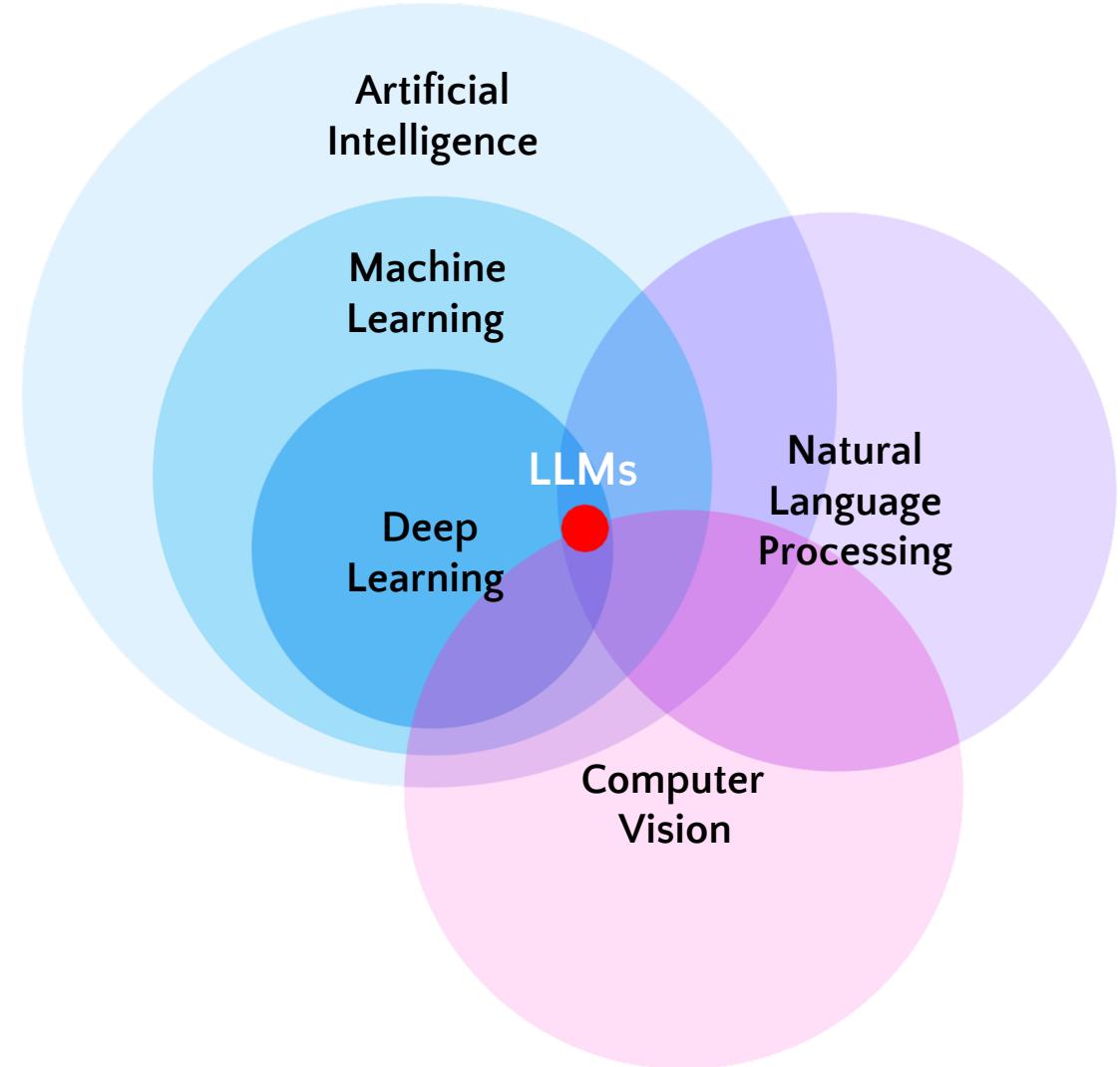


“ Gartner sees generative AI becoming a **general-purpose technology** with an impact similar to that of the **steam engine, electricity and the internet**. The hype will subside as the reality of implementation sets in, but the **impact of generative AI will grow** as people and enterprises discover more innovative applications for the technology in daily work and life.”

<https://www.gartner.com/en/topics/generative-ai>

Large Language Models (LLMs)

- Deep-learning models (neural networks) that generate text
- All accept text as input
 - "Write a poem about deep learning"
- Multimodal LLMs (MLLMs) accept text, images, and sometimes audio and video as input
 - "Describe what you see on this map"
- Billions of parameters trained on trillions of words



Generating Text



Write a short story that begins with "It was a dark and stormy night."

It was a dark and stormy night. The wind was howling through the streets, making it difficult for anyone to stay on their feet. The rain was coming down in sheets, drenching anyone who dared to step outside. Inside, a young woman sat huddled in a corner of her small apartment, listening to the storm rage on outside.

She had always been afraid of storms, ever since she was a child. But tonight, there was something different about this one. It seemed to be getting closer and closer, and she could feel the panic rising in her chest. She tried to shake it off, telling herself that it was just a normal storm, but she couldn't shake the feeling that something was coming.

...

Translating Text

Translate the following into French: "What time does the train leave for Paris?"

"À quelle heure part le train pour Paris?"

Generating Code

Create a Python function that accepts an array of numbers as input, bubble sorts the numbers, and returns a sorted array

```
def bubble_sort(numbers):
    # Get the length of the array
    n = len(numbers)

    # Traverse through all the array elements
    for i in range(n-1):
        # Last i elements are already in place
        for j in range(0, n-i-1):
            # Traverse the array from 0 to n-i-1
            # Swap if the element found is greater than the next element
            if numbers[j] > numbers[j+1]:
                numbers[j], numbers[j+1] = numbers[j+1], numbers[j]

    return numbers
```

Translating Code

Translate the bubble_sort function into FORTRAN

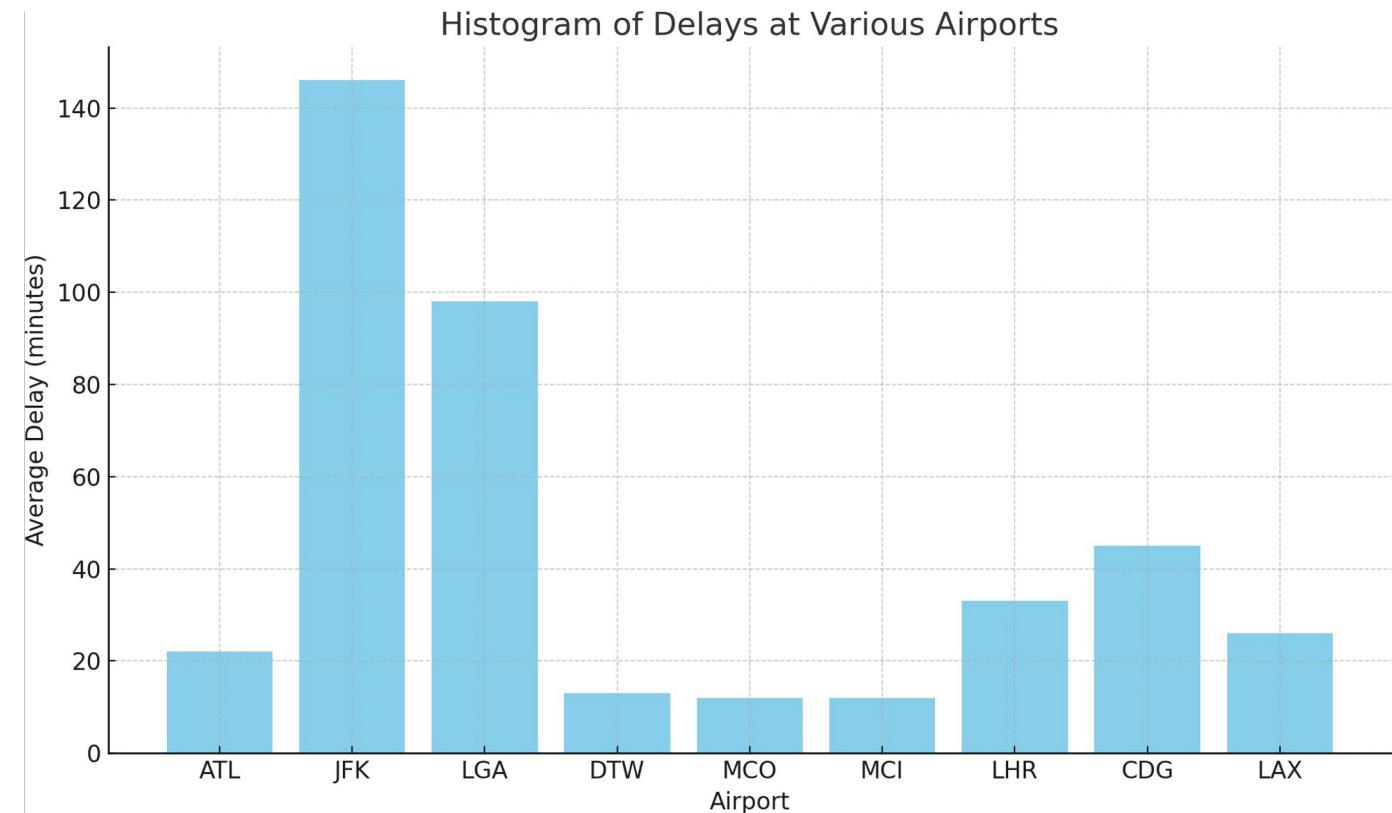
```
FUNCTION bubble_sort(arr, n) RESULT(result)
    ...
    DO i = 1, n
        DO j = 1, n-i-1
            IF (arr(j) > arr(j+1)) THEN
                temp = arr(j)
                arr(j) = arr(j+1)
                arr(j+1) = temp
            END IF
        END DO
    END DO

    result = arr
END FUNCTION bubble_sort
```

Generating Charts and Graphs

Generate a histogram from the data below showing delays at various airports

Airport	Average Delay
ATL	22
JFK	146
LGA	98
DTW	13
MCO	12
MCI	12
LHR	33
CDG	45
LAX	26



Analyzing Charts

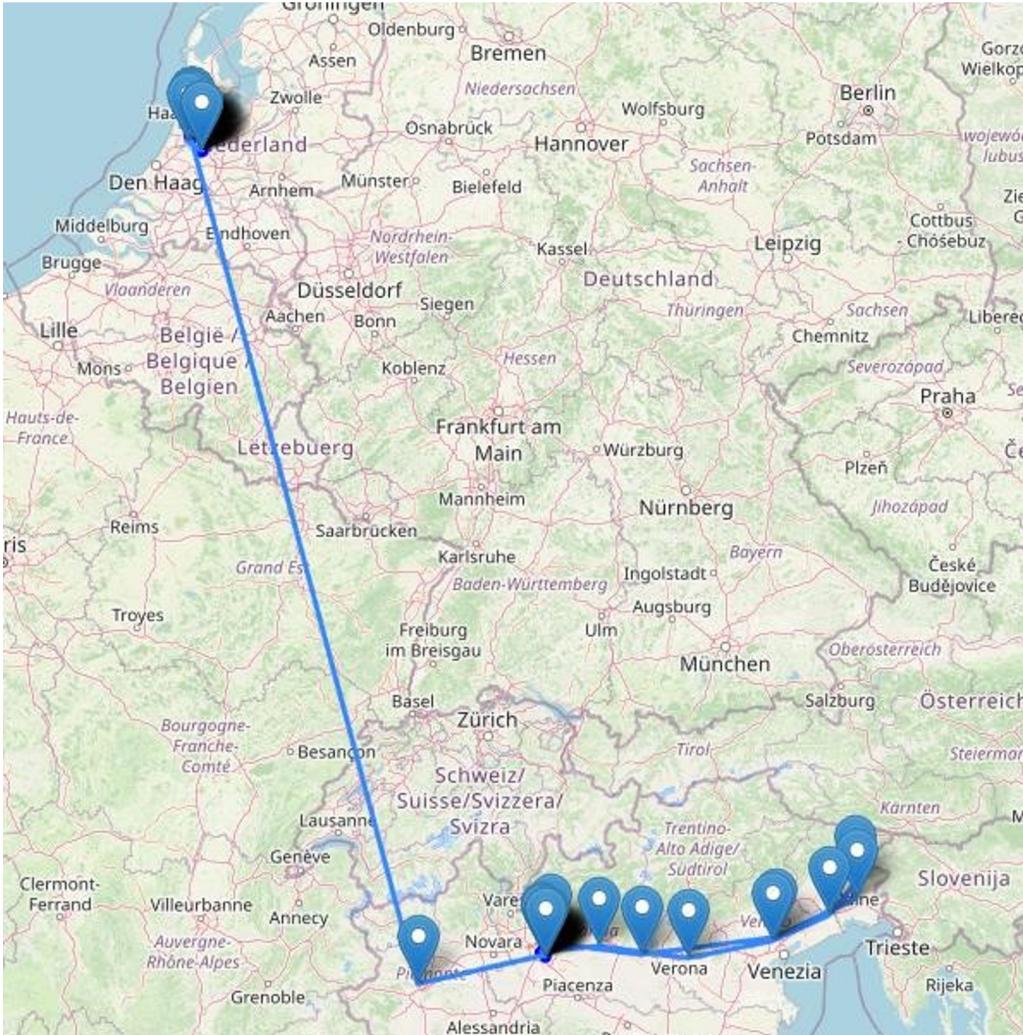
Explain the chart on the preceding slide

This chart is a histogram depicting the average delays (in minutes) at various airports. Here are some key observations:

- JFK has the highest average delay, approaching 140 minutes.
- LGA follows with a slightly lower average delay, around 100 minutes.
- Most other airports, such as ATL, DTW, MCO, and MCI, have much lower average delays, typically around 20 minutes or lower.
- CDG and LHR, two major international airports, show moderate delays (CDG with around 70 minutes and LHR with about 50 minutes)

The light blue bars indicate the magnitude of the delay for each airport, and the title of the chart indicates it's a "Histogram of Delays at Various Airports."

Multimodal LLMs (MLLMs)



Describe the shipment plotted on this map.

The shipment began in the Netherlands and made its way to northeastern Italy using a combination of air and ground transportation.

The initial segment went by air, originating from a location near the coast of the Netherlands, possibly near Rotterdam, and ending in northeastern Italy, likely near Milan, as the shipment appears to fly across the Netherlands, over Germany, and into Italy.

The subsequent legs of the journey were traveled by road. The shipment passed through several cities in northern Italy, eventually reaching its destination near Venice.

The map does not specify exact highways, but the route within Italy likely involved major roads such as the A4 motorway, which connects Milan to Venice.

Image generation

"Create an image of cupcake made of a cute corgi dog, in cartoon style"



"Create an image that looks like it's made out of felt, having a crafty, textured look"



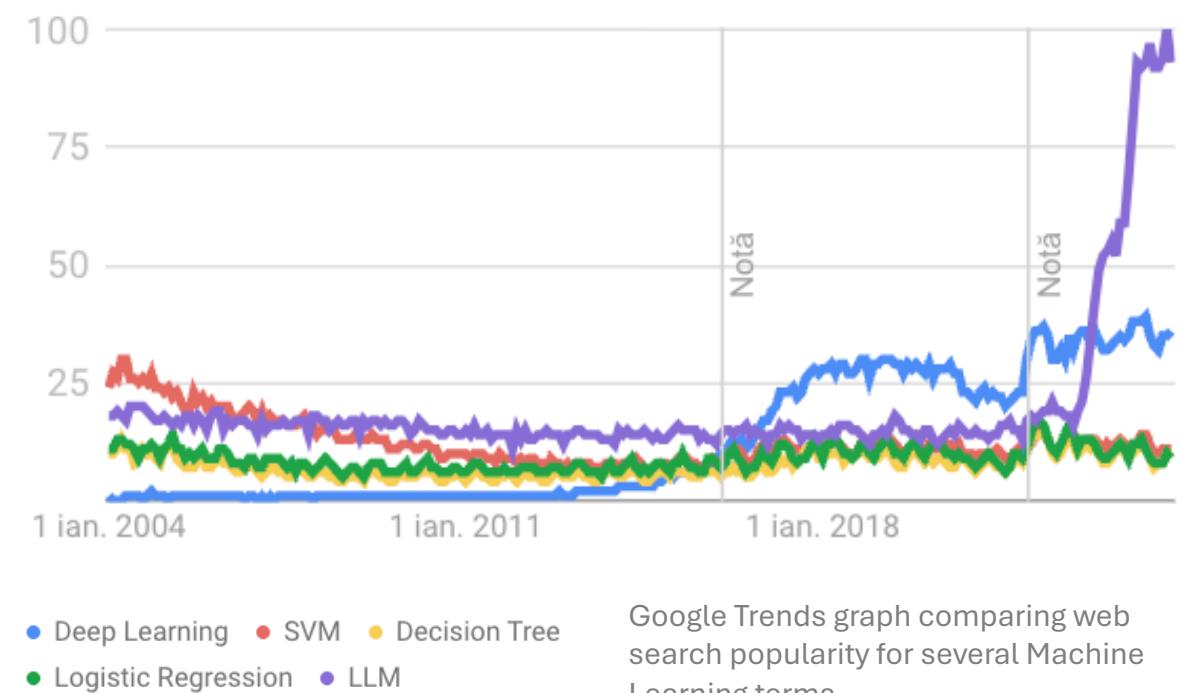


Neural Networks

Basics

Neural Networks – Motivation

- Deep Learning, through Neural Networks, is considered to be one of the most powerful and high-profile approaches to Artificial Intelligence.
- Natural language processing/generation, image processing/generation, speech recognition, face recognition, autonomous driving, recommender algorithms etc.
- Development of high-capacity storage options, Internet, Cloud and high-power CPUs and GPUs have all contributed to the popularity of Deep Learning.



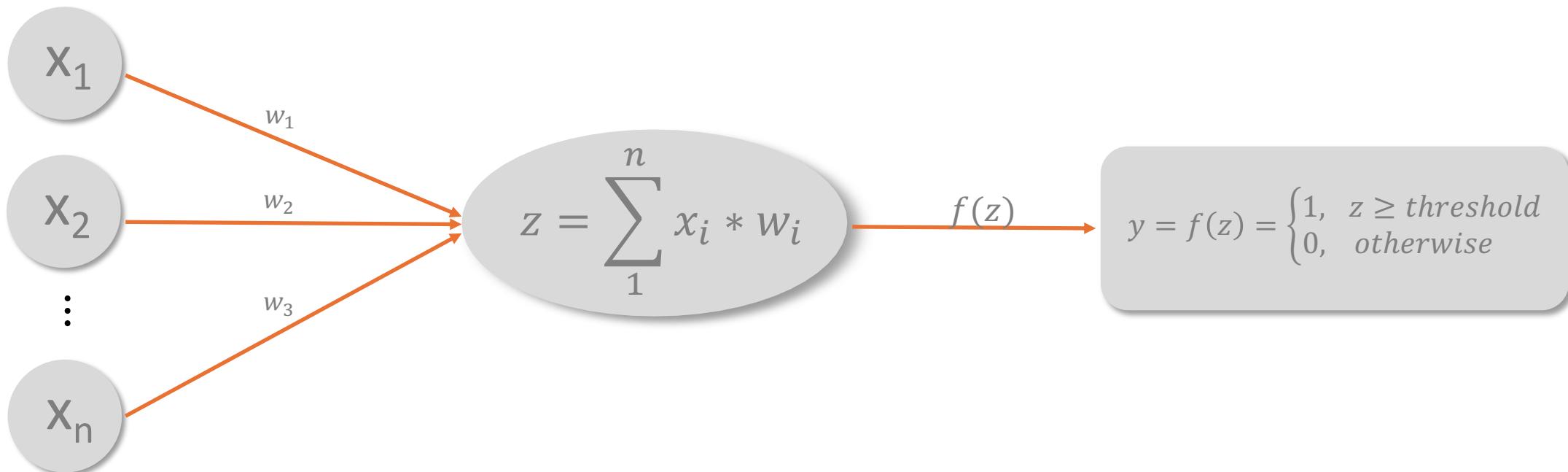
Neural Networks – Use Cases



- **Recommendation Systems**
 - E.g., Netflix, YouTube, and Amazon for personalized content or product suggestions.
- **Customer Support**
 - Chatbots and virtual assistants like Siri or Alexa.
- **Fraud Detection**
 - Identifying suspicious transactions in banking and e-commerce.
- **Marketing and Advertising**
 - Targeted advertising and customer segmentation.
- **Quality Control**
 - Visual inspection systems for defect detection in production lines.
- **Energy Optimization**
 - Managing and forecasting energy demands in utilities.
- **Medical Imaging**
 - Diagnosis using X-rays, MRIs, or CT scans (e.g., detecting cancer or fractures).
- **Autonomous Vehicles**
 - Powering self-driving cars and drones.
- **Finance**
 - Stock price prediction and portfolio management.
- **Security**
 - Facial recognition and surveillance systems.

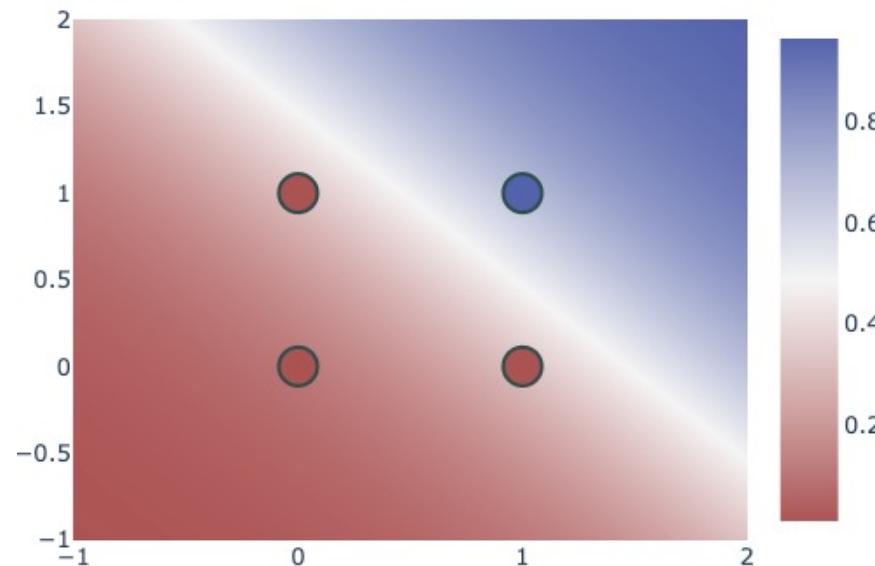
Perceptron

- The **Perceptron** – simplest kind of Neural Networks
- A single layer, containing only one neuron
- Each input is fed proportionally by a weight into this neuron:
 - The sum of the products between each input and its respective weight is computed
 - If the sum is above a threshold (usually **0**) the neuron outputs the activated value (usually **1**)
 - Otherwise, it outputs a deactivated value (**0** or **-1**)

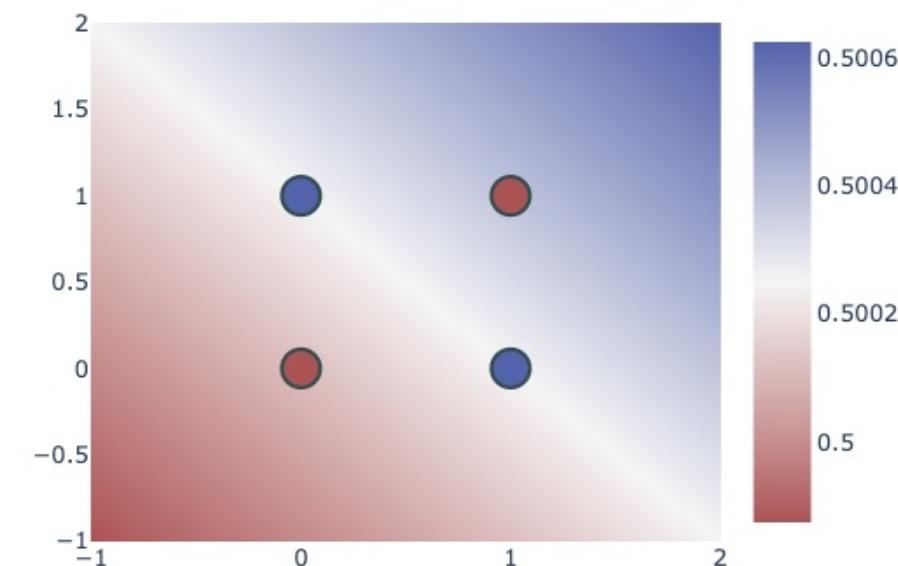


Perceptron

- The output of the Perceptron, $y = f(z)$, is the output of a function, called step function, applied to the sum of the product of weights and inputs – **activation function**.
- Because it is described by a linear equation, the Perceptron is a **linear classifier**, therefore, it can only classify **linearly separable** datasets.
- **2D space** – Perceptron is described by a line.
- **3D space** – Perceptron is described by a plane.



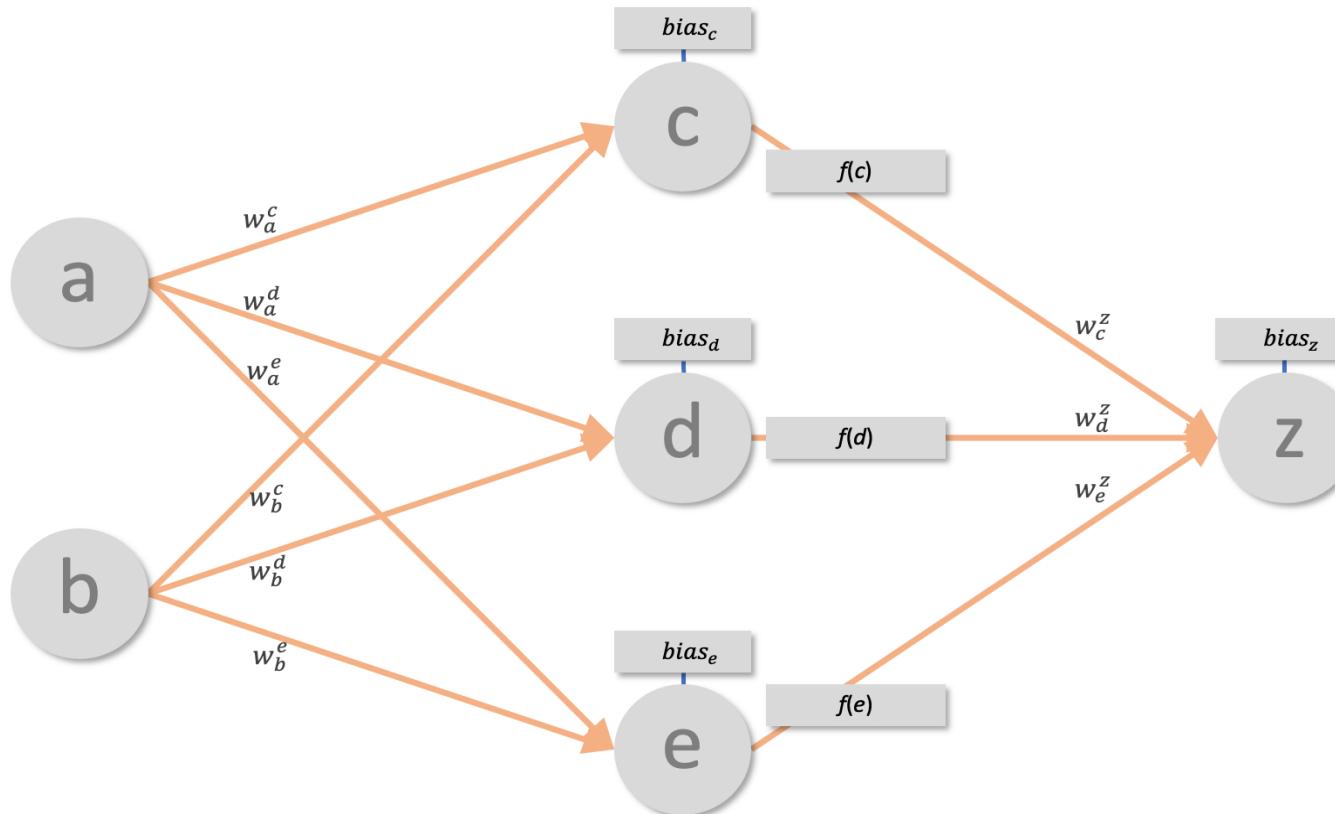
The Perceptron, being a linear classifier, successfully learns the **AND** function, as its graph is **linearly separable**.



As for the **XOR** function, because its graph is **not linearly separable**, the Perceptron is unable to classify it correctly.

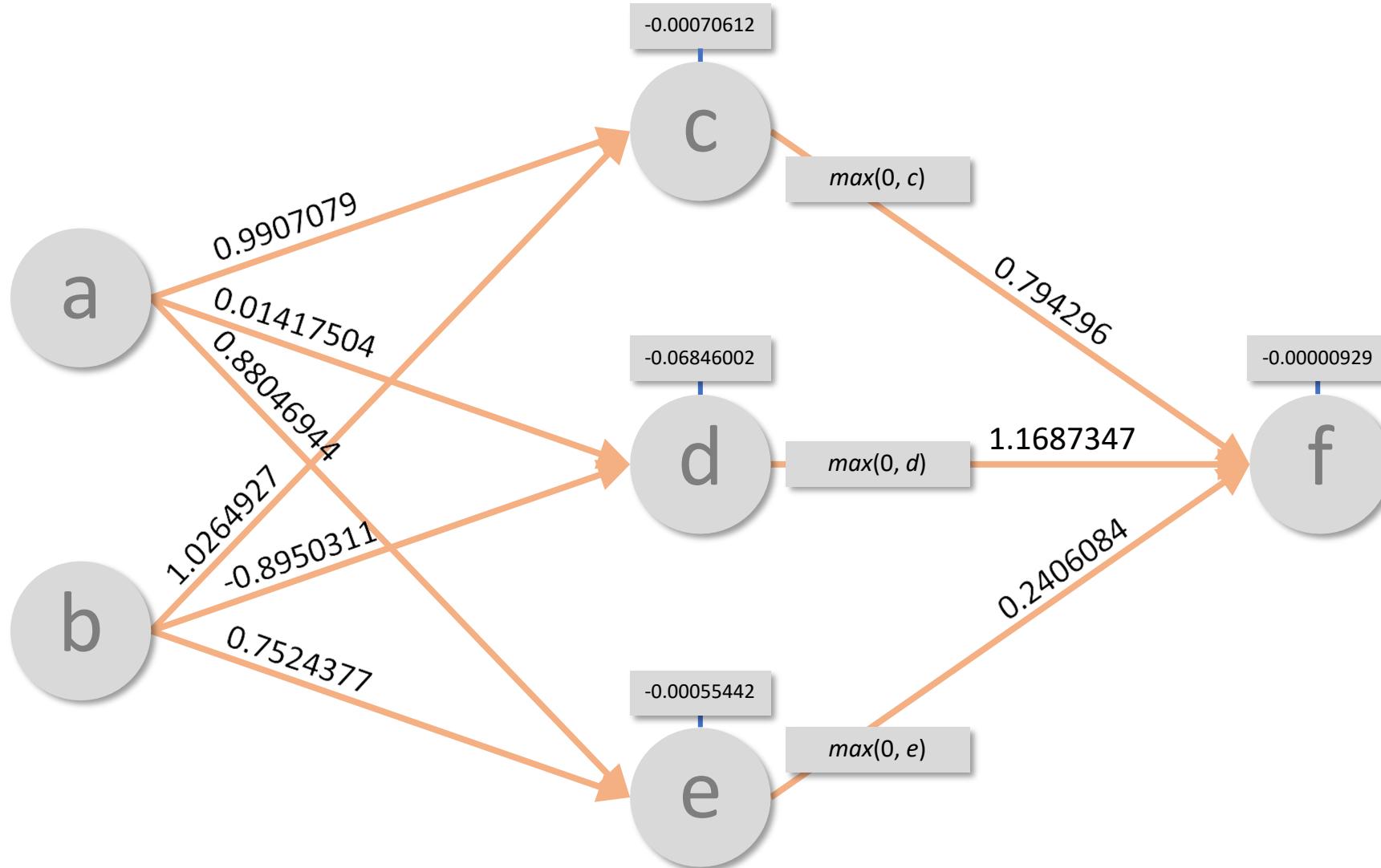
Multi-Layer Perceptron

- Multiple Perceptrons on multiple, fully-connected layers → classify more complex datasets.
- These configurations are the heart of Deep Learning: **Dense Neural Networks**
- Having multiple layers results in basically a composition of functions: each layer in the network applies a function to the input it receives and the output of one layer becomes the input to the next.



- Depicted as directed graphs
- Node = neuron
- Edge = weighted link
- a, b – input layer
- c, d, e – hidden layer
- z – output layer
- $f(x)$ – output of neuron x
- $bias_x$
 - bias of neuron x , a value (\pm) added to the sum of products between adjacent input nodes and their respective weights
 - i.e. the **threshold** of the previous example of single Perceptron
- Example: $c = a * w_a^c + b * w_b^c + bias_c$

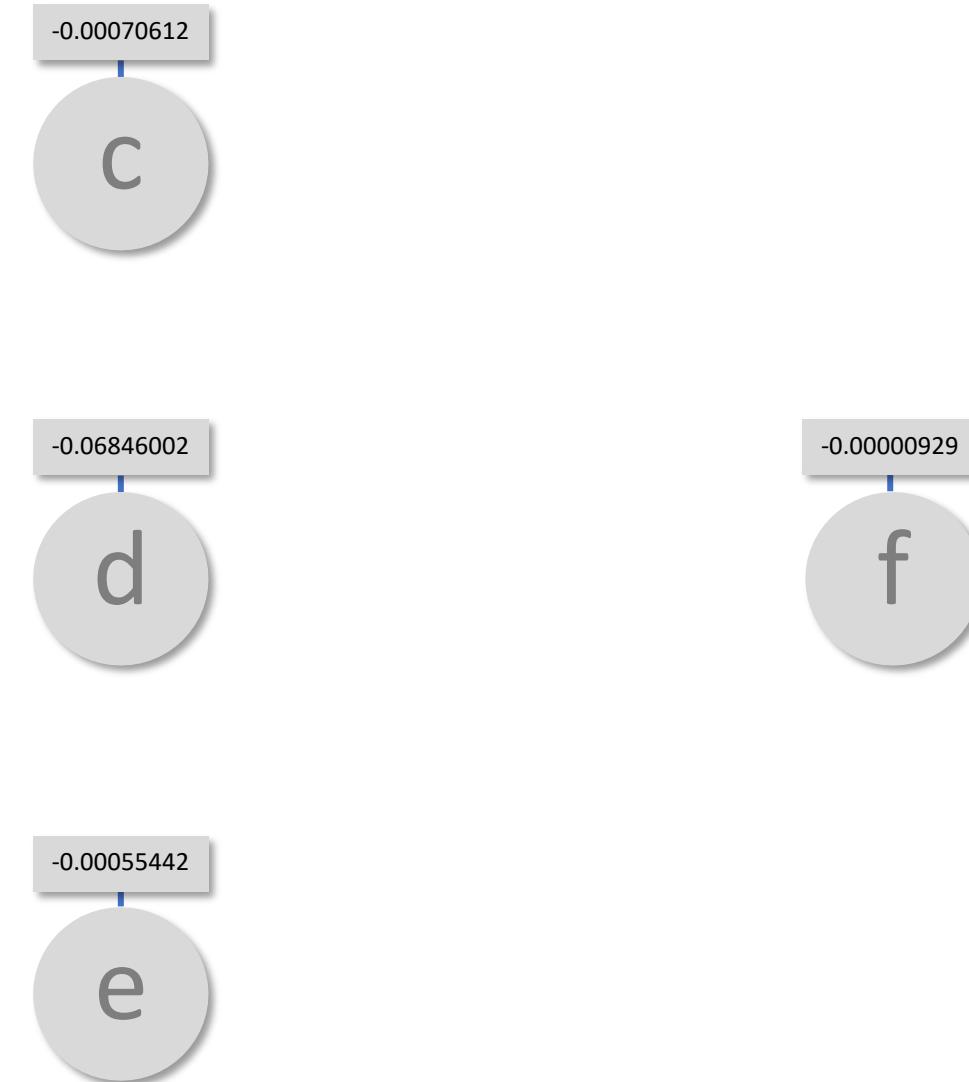
Feed Forward, Cont.



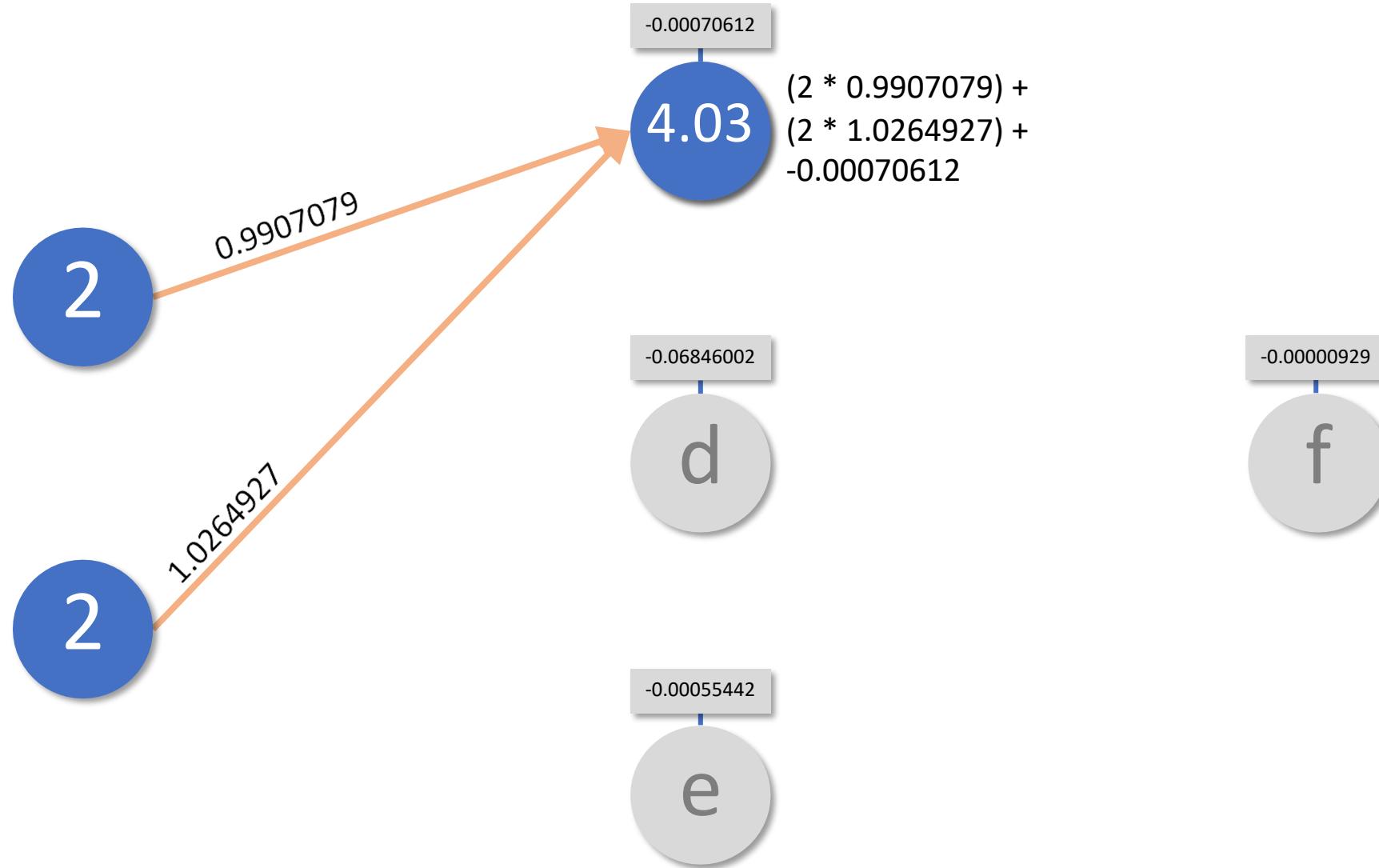
Feed Forward, Cont.

2

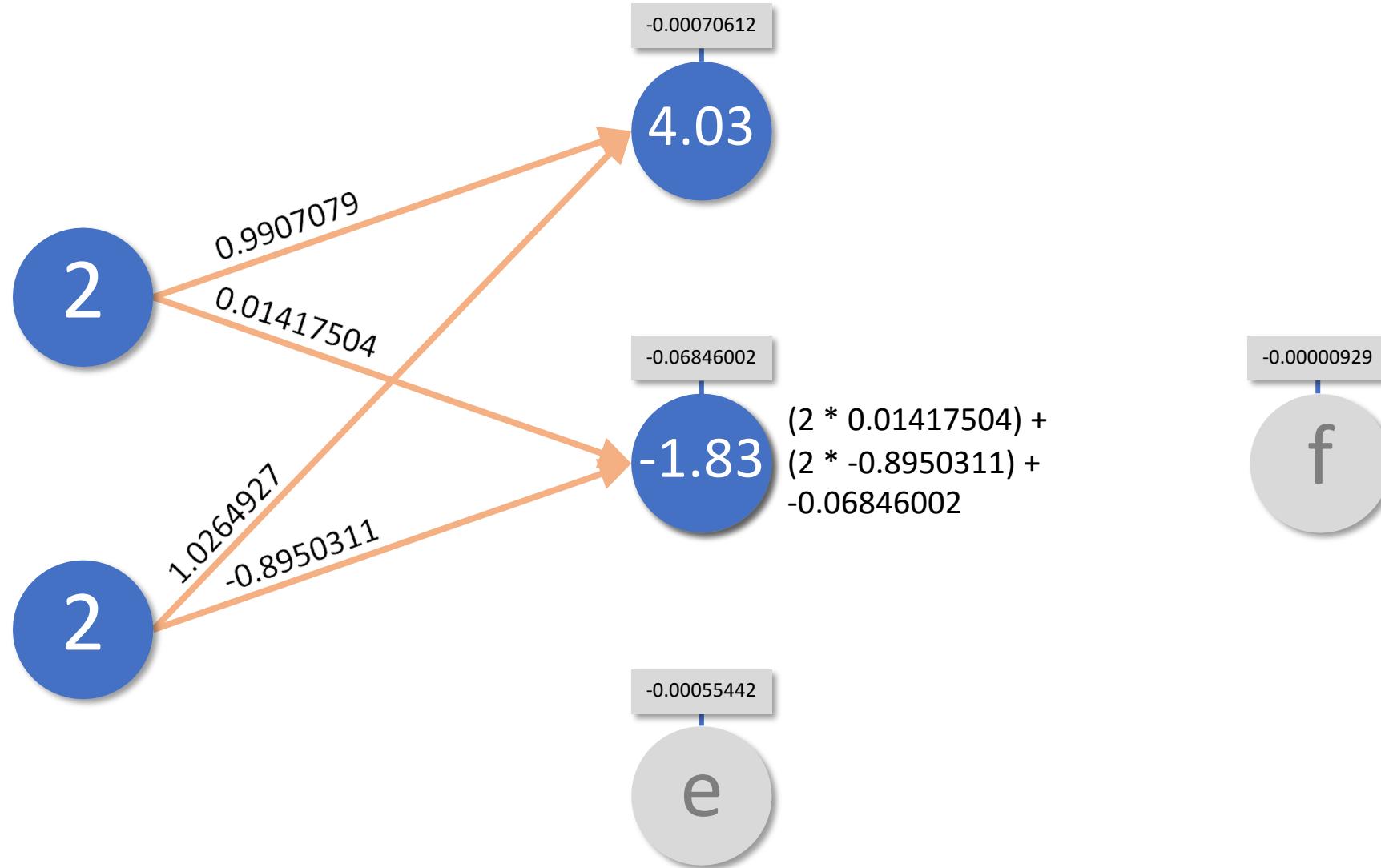
2



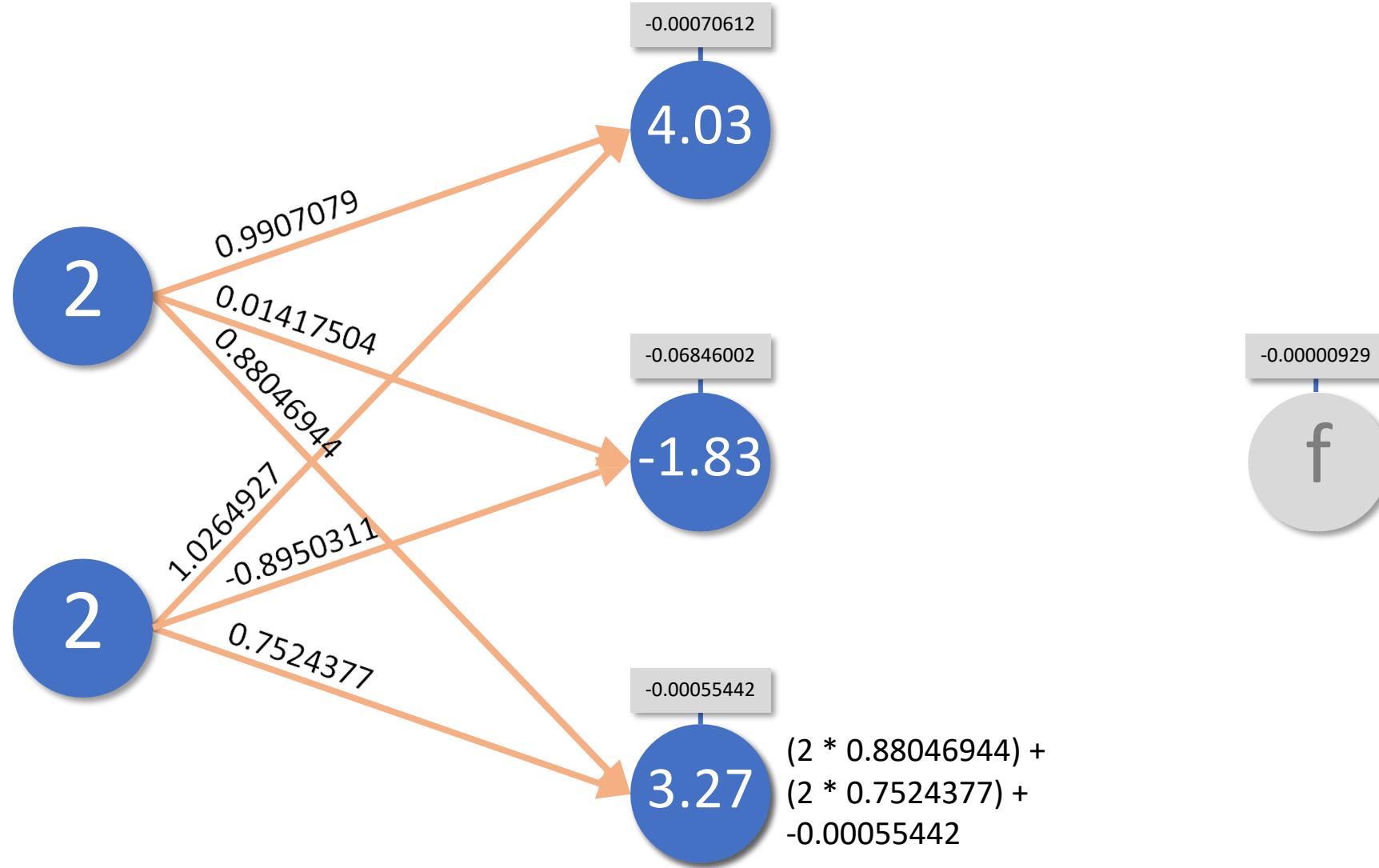
Feed Forward, Cont.



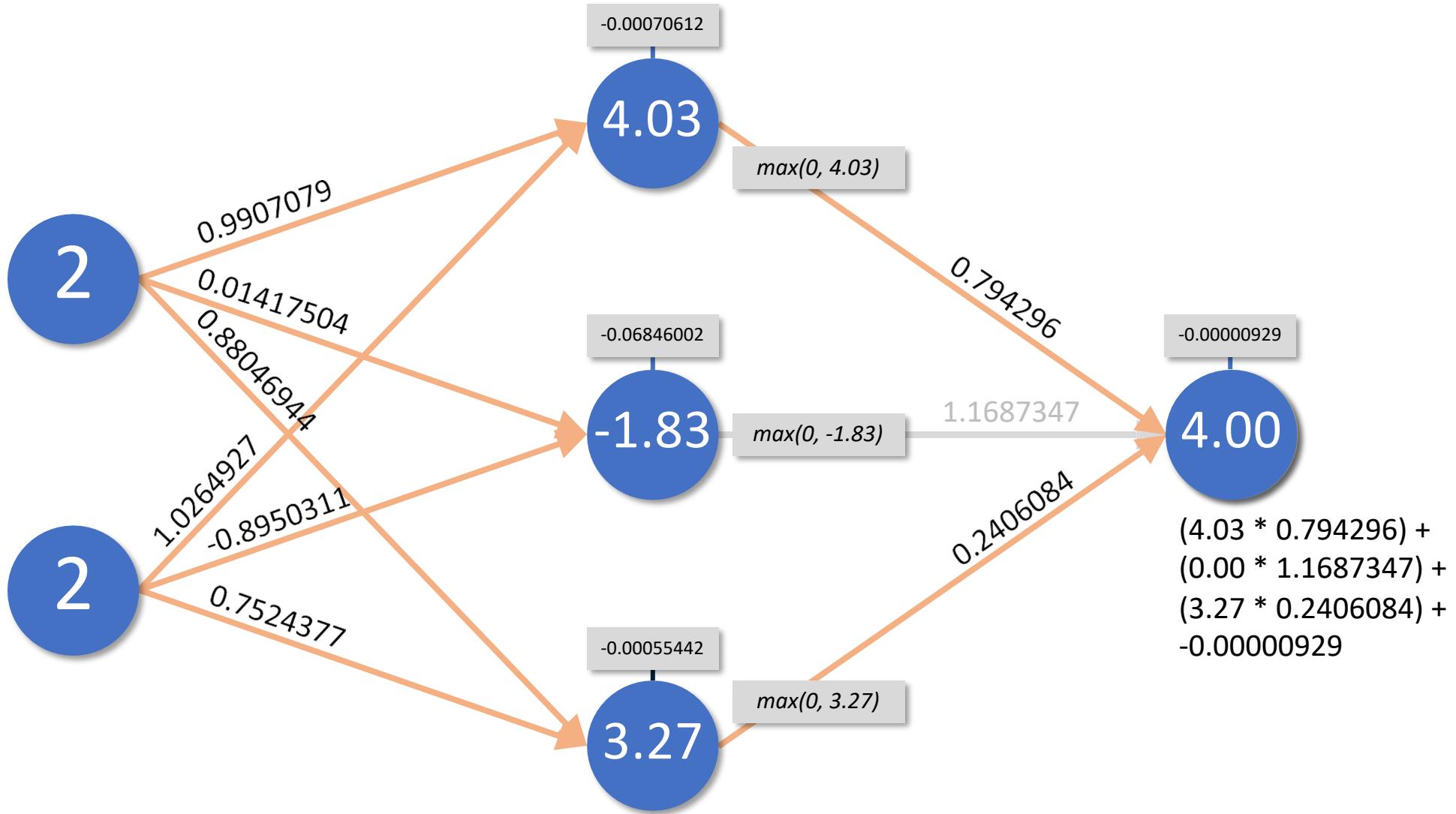
Feed Forward, Cont.



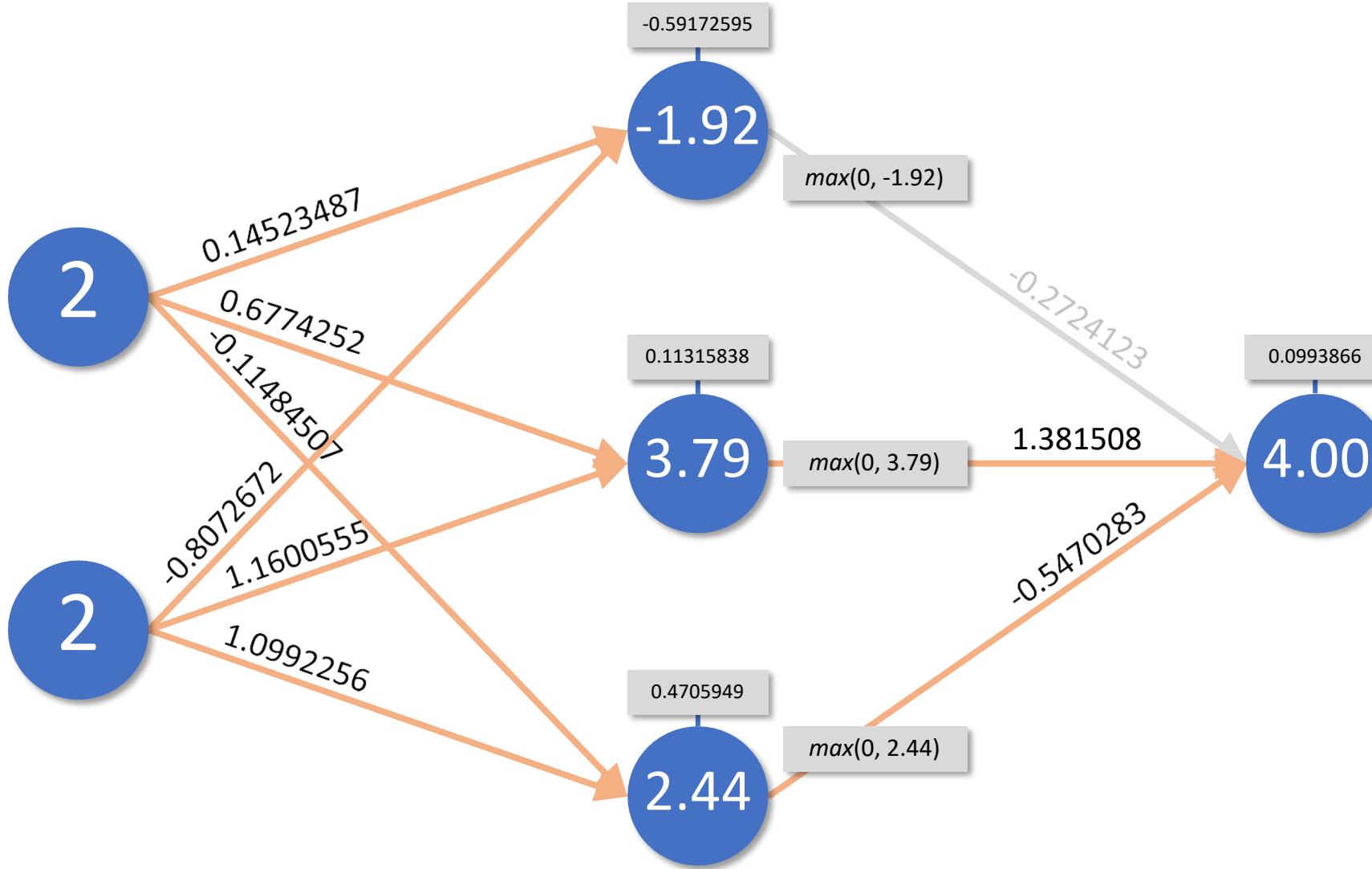
Feed Forward, Cont.



Feed Forward, Cont.



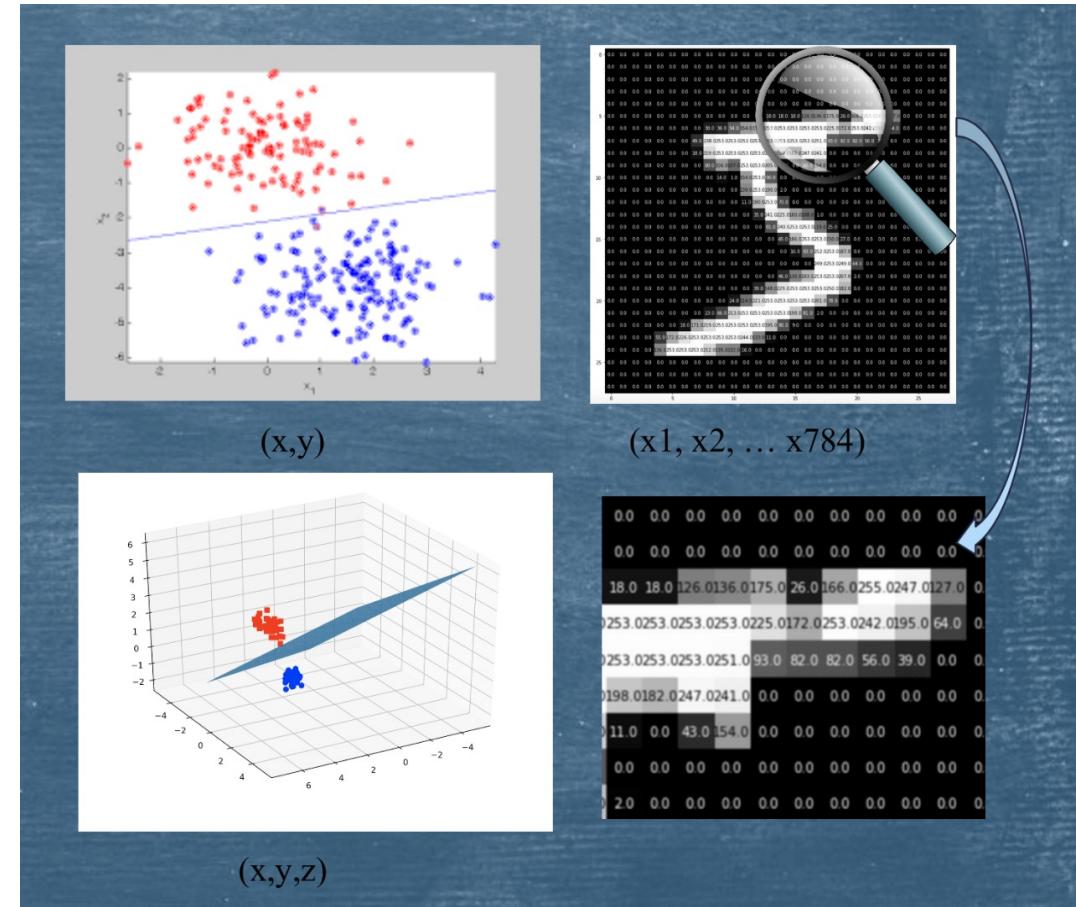
Different Weights and Biases, Same Result



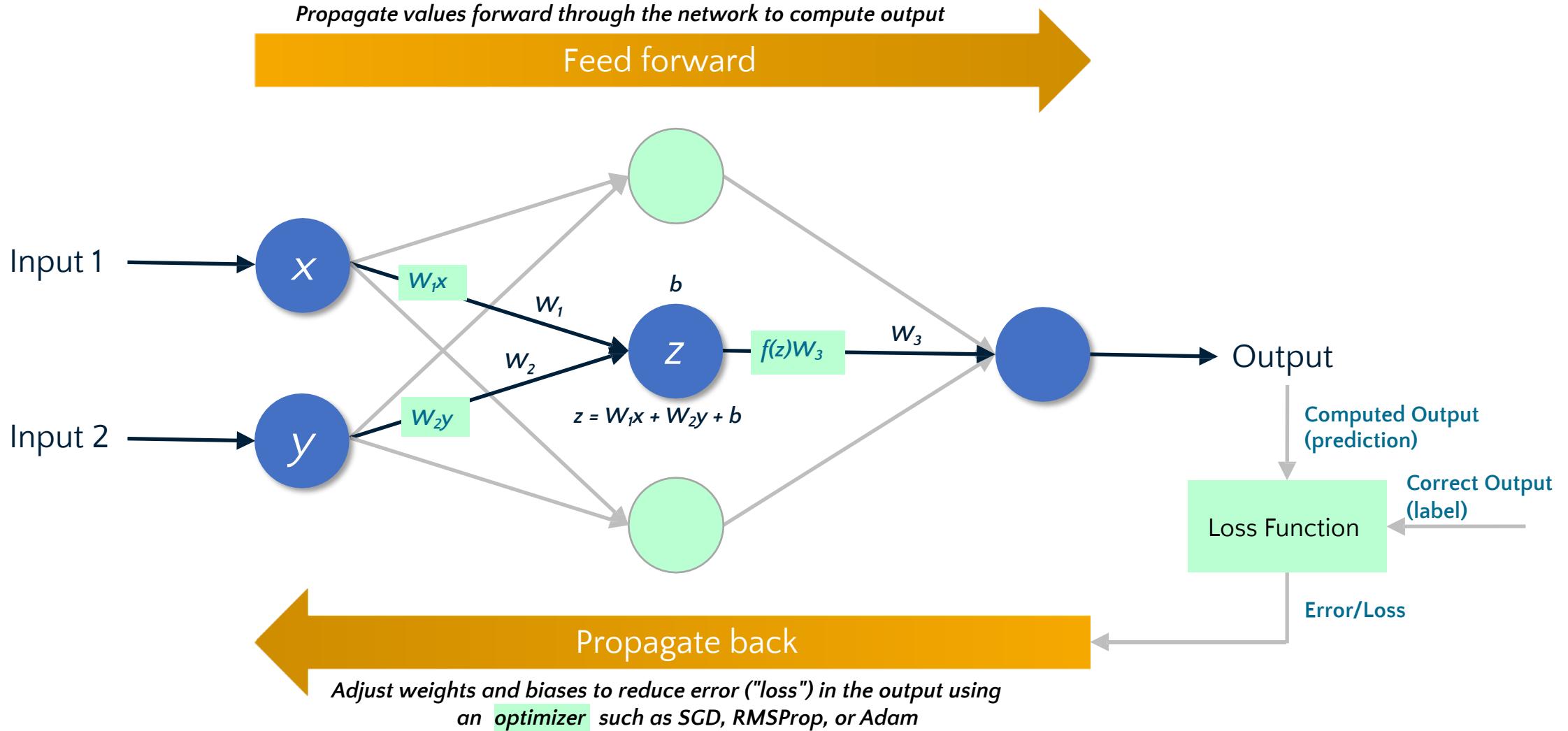
Concepts

- **Dataset**

- Collection of elements used for training
- Represented as a set of
 - **feature vectors:** (x, y) for bidimensional datasets, $(x, y, \text{color_code})$ for images etc.
 - **labels:** for each element of the dataset (i.e. a feature vector), a label is assigned (positive/negative, an encoding for a class in the case of multi-class classification, etc.)
- It should be split in
 - **Training subset** – used for training and only for training
 - **Testing subset** – used for evaluating the model's capacity to predict on new data (that wasn't used for training)

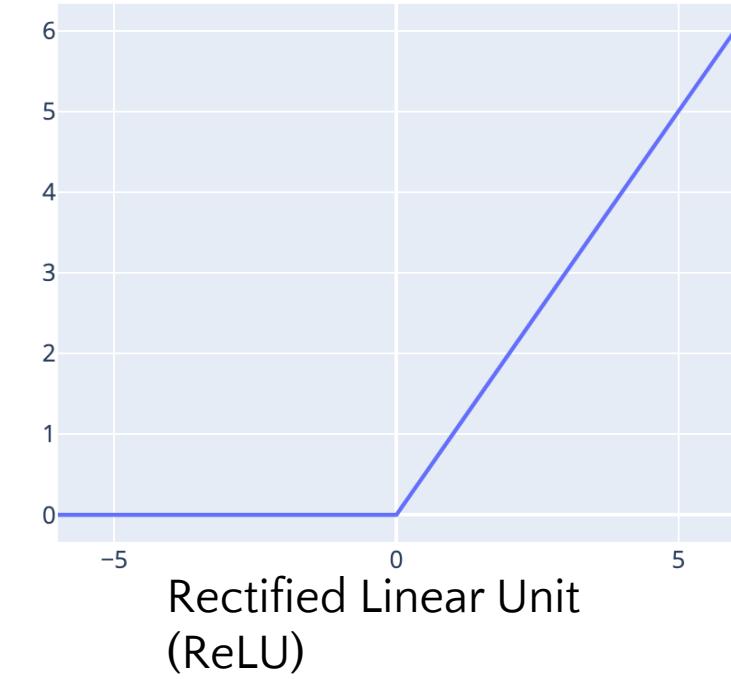
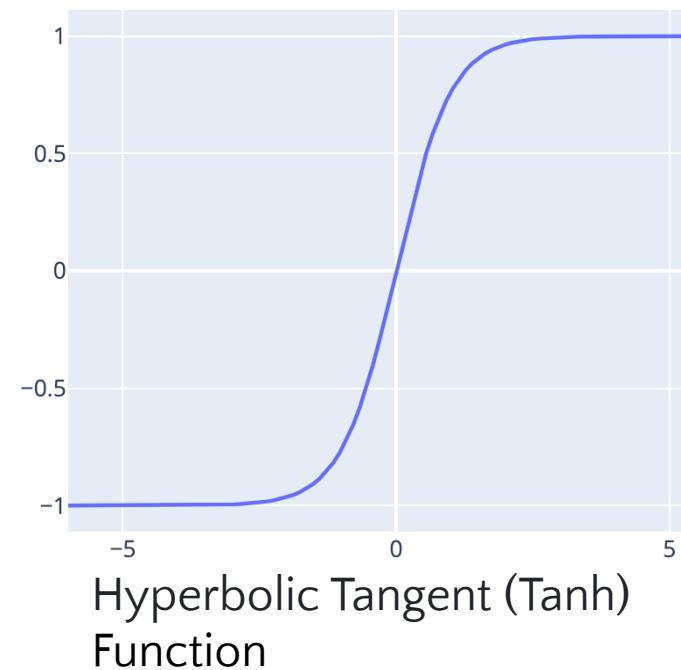
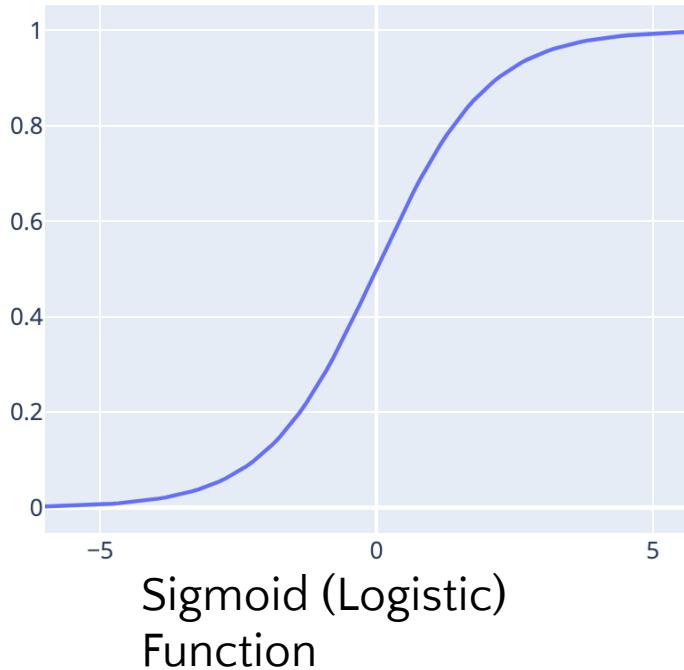


Training



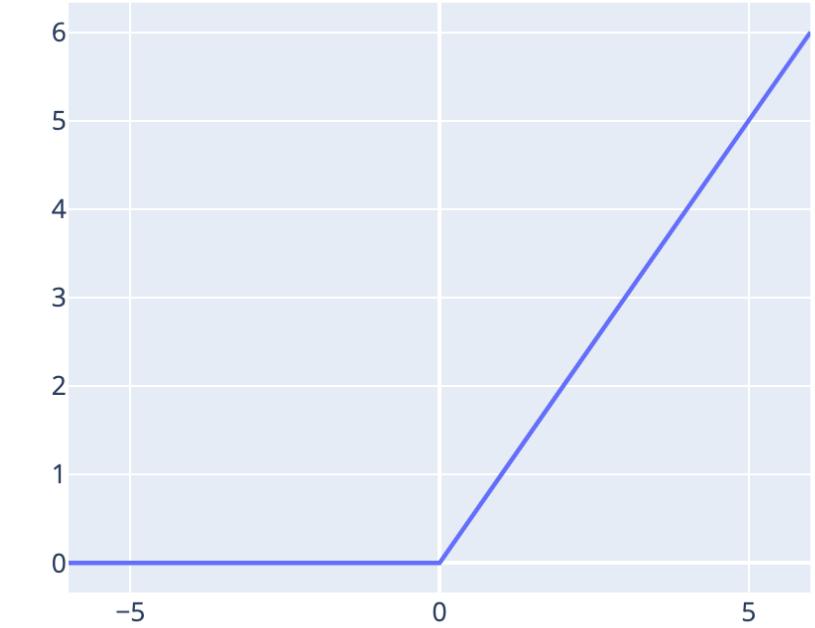
Activation Functions

- These functions are applied to the sums of the products between inputs and respective weights of a neuron → the **output of a neuron**.
- Their role is to introduce **non-linearity**.
- They should be **differentiable**, as their **gradients** (derivatives) will be used in the learning process.
- Some are used to control the output (e.g. the Logistic function has the outputs in the $[0,1]$ range).



Rectified Linear Unit (ReLU) Function

- **Input:** any real number
- **Output:** $y \in [0, +\infty)$
- Together with other variants \rightarrow the most used act. function
- **Advantages**
 - Useful derivative
 - Computationally efficient
 - Faster learning
 - Constant gradients \rightarrow no vanishing gradient problem
 - 0 gradient for negative values \rightarrow sparsity in the network

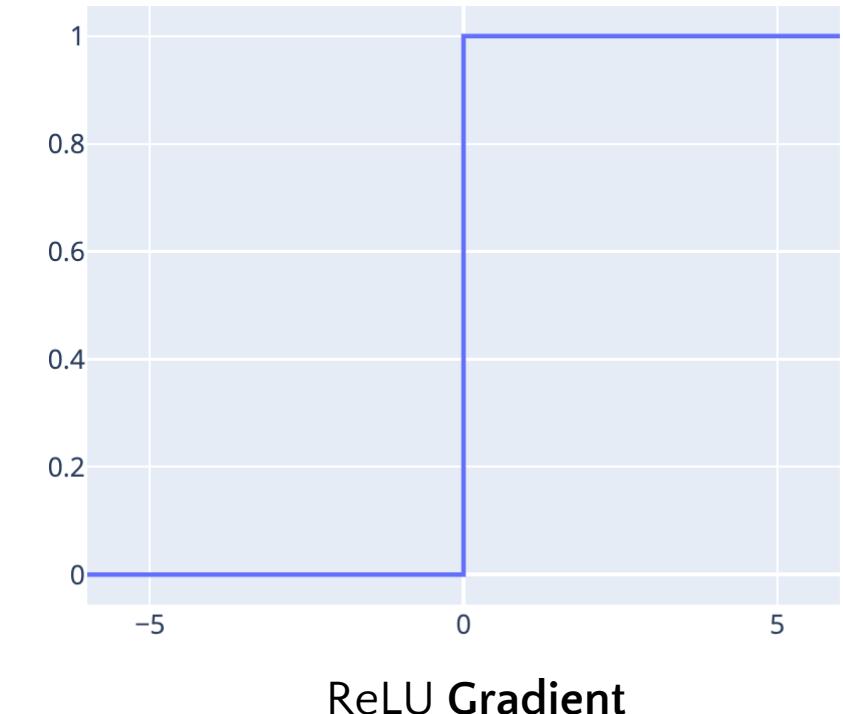


ReLU Function

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

Rectified Linear Unit (ReLU) Function cont.

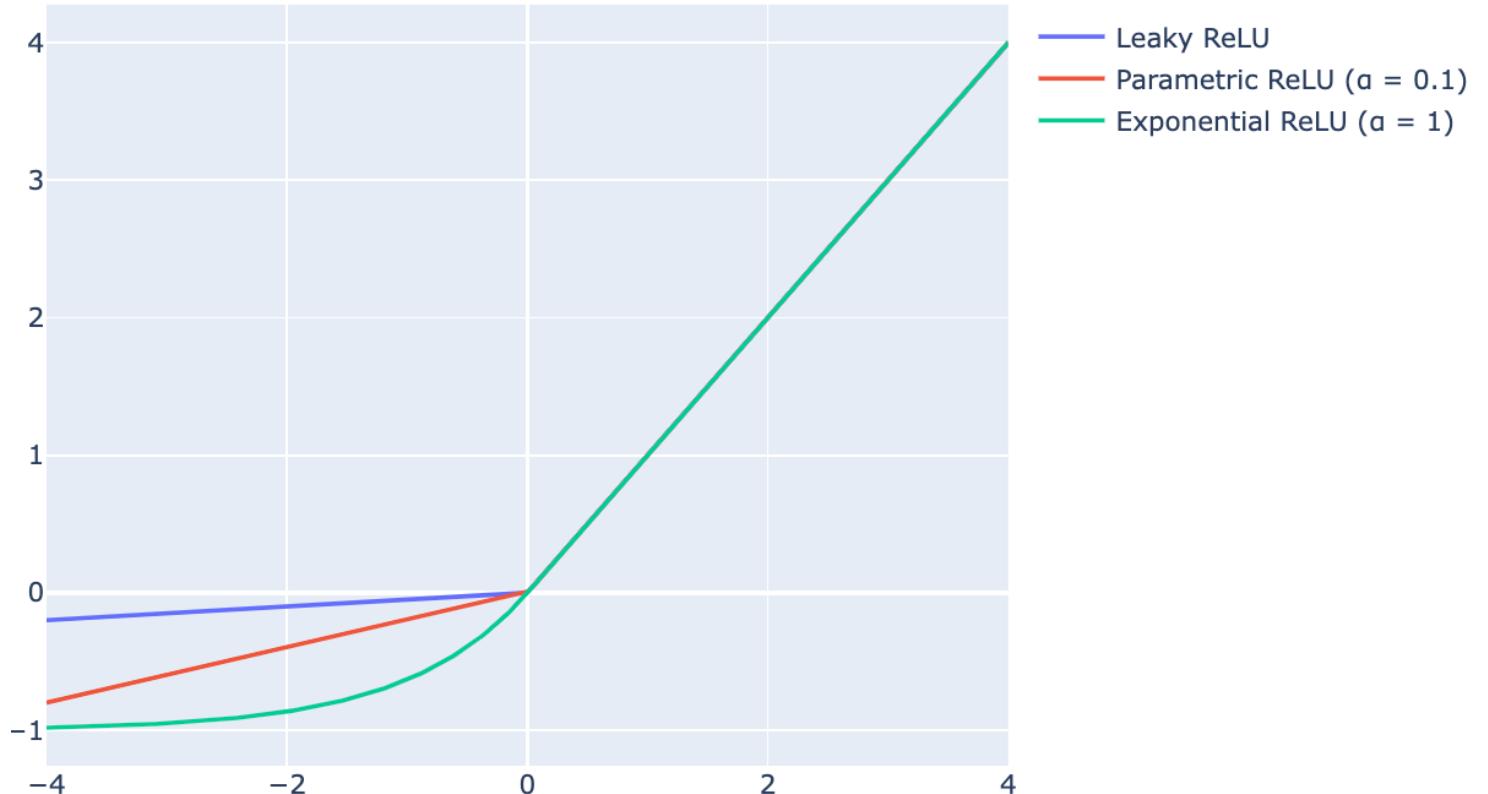
- **Disadvantages**
 - When the output of a neuron is always negative, causing the gradient to be 0 during backpropagation, its weights stop updating and the neuron effectively becomes “dead” → **Dying ReLU Problem**
 - Since the output is unbounded, it can lead to exploding gradients
- **Alternatives**
 - Leaky ReLU
 - Parametric ReLU



$$relu'(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

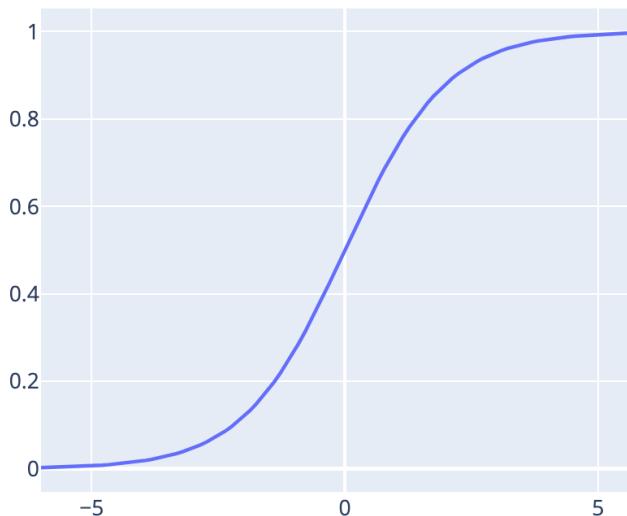
ReLU Variants

- Leaky ReLU
 - $x \cdot 0.1 \text{ if } x < 0$
- Parametric ReLU
 - $x \cdot \alpha \text{ if } x < 0$
- Exponential ReLU
 - $\alpha(e^x - 1) \text{ if } x < 0$



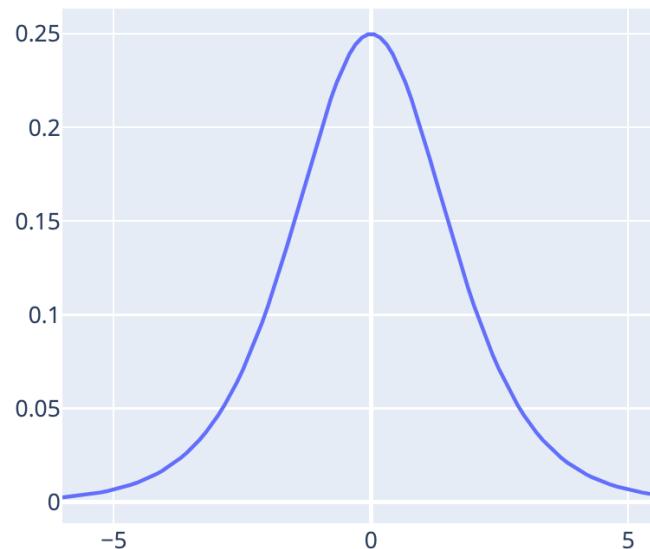
Sigmoid (Logistic) Function

- **Input:** any real number
- **Output:** $y \in (0,1)$
- Usually used in **binary classification**
- **Advantages**
 - Produces a smooth gradient.
- **Disadvantages**
 - Pre-processing of the input labels needed (e.g. negative 0, positive 1)
 - Small gradients for inputs outside of $[-3,3]$ range → **Vanishing Gradient Problem** → slow learning



Sigmoid

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

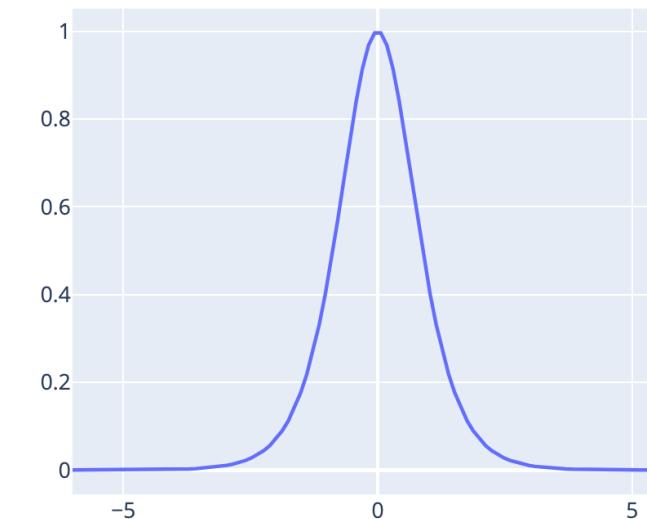
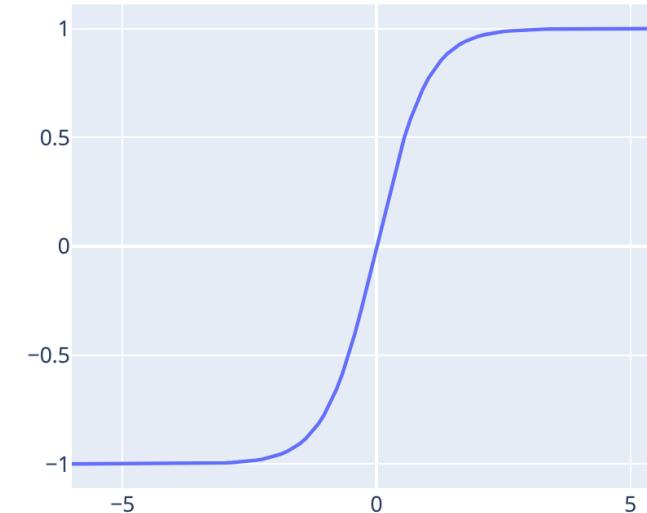


Sigmoid Gradient

$$\sigma'(x) = \sigma(x) * (1 - \sigma(x))$$

Hyperbolic Tangent (Tanh) Function

- **Input:** any real number
- **Output:** $y \in (-1,1)$
- **Advantages**
 - Same advantages as Sigmoid
 - It is 0-centered
 - Has steeper gradients in $[-1,1]$ -> **faster learning**
 - Outputs both **positive** and **negative** values
- **Disadvantages**
 - Pre-processing of the input labels needed (e.g. negative -1 , positive 1)
 - Small gradients for inputs outside of $[-2,2]$ range -> **Vanishing Gradient Problem** -> slow learning



Tanh

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

Tanh Gradient

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

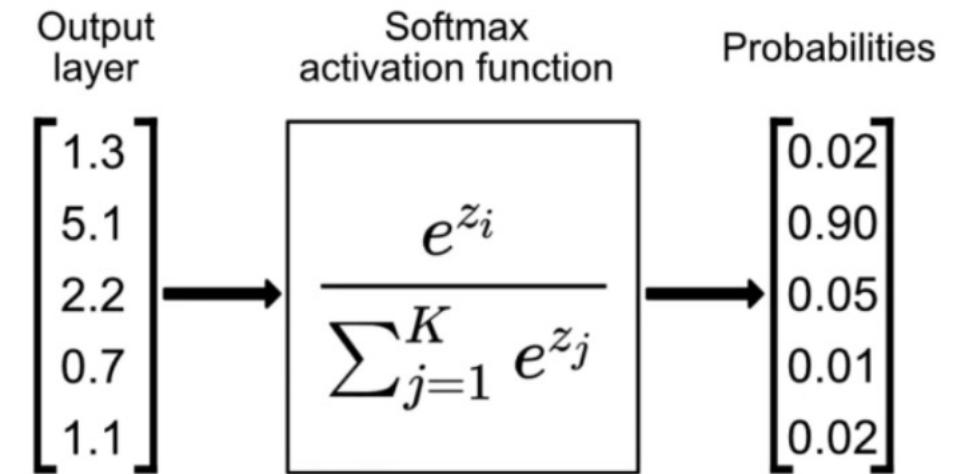
Softmax Function

- **Advantages:**

- Transforms raw output scores into **probabilities** that sum up to 1
- Commonly used in the case of **multi-class classification** tasks
- Each value from the output vector represents the probability that the input element belongs to each class

- **Disadvantages:**

- Preprocessing is needed for the class labels in the training dataset. E.g. Labels 1, 2 and 3 need to be converted to **one-hot encodings**: [1,0,0], [0,1,0], [0,0,1]



Where:

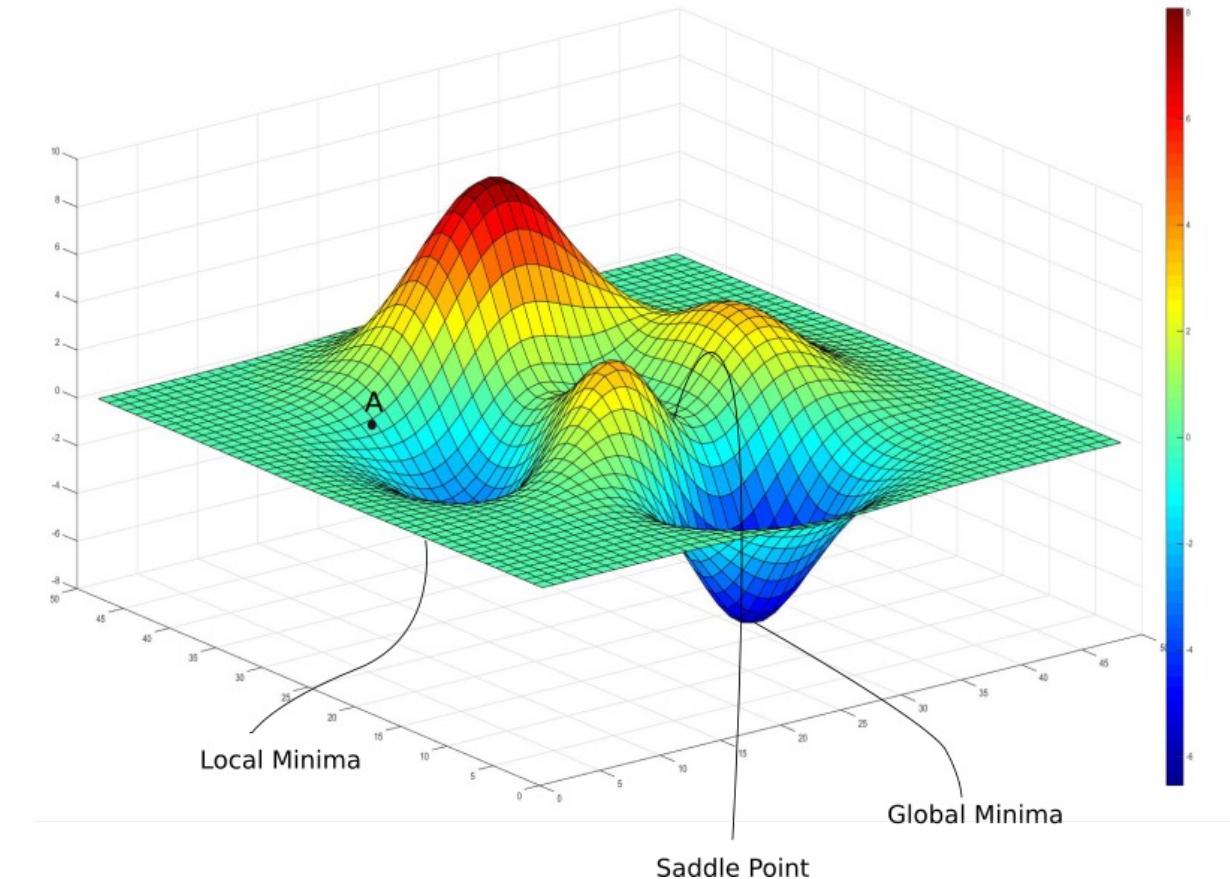
- z_i is the weighted sum of the inputs for neuron i
- K is the number of neurons on the given (usually output) layer

Activation Functions – Summary

Activation Function	When to use
ReLU	The most popular activation for hidden layers. It is simple, fast to compute, and works well for most models. No vanishing gradient.
Sigmoid	For binary classification , when you need outputs in (0,1) and logistic regression.
Softmax	Used in the output layer, especially for multi-class classification . It converts raw scores into probabilities for each class, ensuring that the output sums to 1.
Tanh	When you need outputs in $(-1,1)$, that are zero-centered
Parametric ReLU (PReLU)	Useful where ReLU's zero output for negative inputs could cause dead neurons .
Exponential ReLU (ELU)	When you want a smoother , more continuous activation function that avoids dead neurons and is less prone to the vanishing gradient problem .

Loss (Cost) Functions

- It quantifies how well or poorly a neural network is performing in terms of making predictions.
- It expresses the loss (error) by computing the discrepancy between the network's **predicted outputs** and the **actual target values** (ground truth).
- Training the network aims at **minimizing** the loss function
- Minimum loss = **maximum accuracy**



Mean Squared Error (MSE)

- Computes the average of squared differences between the **predicted** labels and the **actual** labels
- Resembles the formula for variance, which computes how far some elements are from the mean

$$L_{MSE} = \frac{1}{2n} \sum_i^n (t_i - y_i)^2$$

Where:

- n – nb. of training instances
- t_i – label of instance i
- y_i – output (predicted label) for instance i

Mean Squared Error (MSE)

- **Advantages:**
 - It is continuous and easily differentiable
 - Penalizes larger errors: MSE squares the error, so larger differences have a disproportionately larger effect on the loss.
 - MSE is a convex function, which means that it has a single global minimum
- **Disadvantages**
 - Very sensitive to outliers
 - A single error can lead to a model that perform poorly on the majority of data

$$L'_{MSE} = -\frac{1}{n} \sum_i^n (t_i - y_i)y'_i$$

Where y'_i is the gradient of the activation function

Cross-Entropy

- Best used together with the **Softmax** activation function
- Suited for **multi-class** classification tasks
- Can also be adapted to **binary-class** classifications:
Binary Cross-Entropy
- Takes probabilities and measures the “distance” from the actual values (actual labels from the training set)
- Output is **logarithmic**:
 - Values close to 1 for large differences
 - Values close to 0 for small differences

$$L_{CE} = -\frac{1}{n} \sum_i^n \sum_j^m t_{ij} \ln(y_{ij})$$

Where:

- n – nb. of training instances
- m – nb. of classes
- t_{ij} – label of instance i from class j
- y_{ij} – output (predicted label) for instance i from class j

Cross-Entropy

- **Advantages**
 - It is designed to provide large gradients for large errors and small gradients from small errors
 - Cross Entropy is a convex function, which means that it has a single global minimum
- **Disadvantages**
 - Sensitive to imbalanced sets

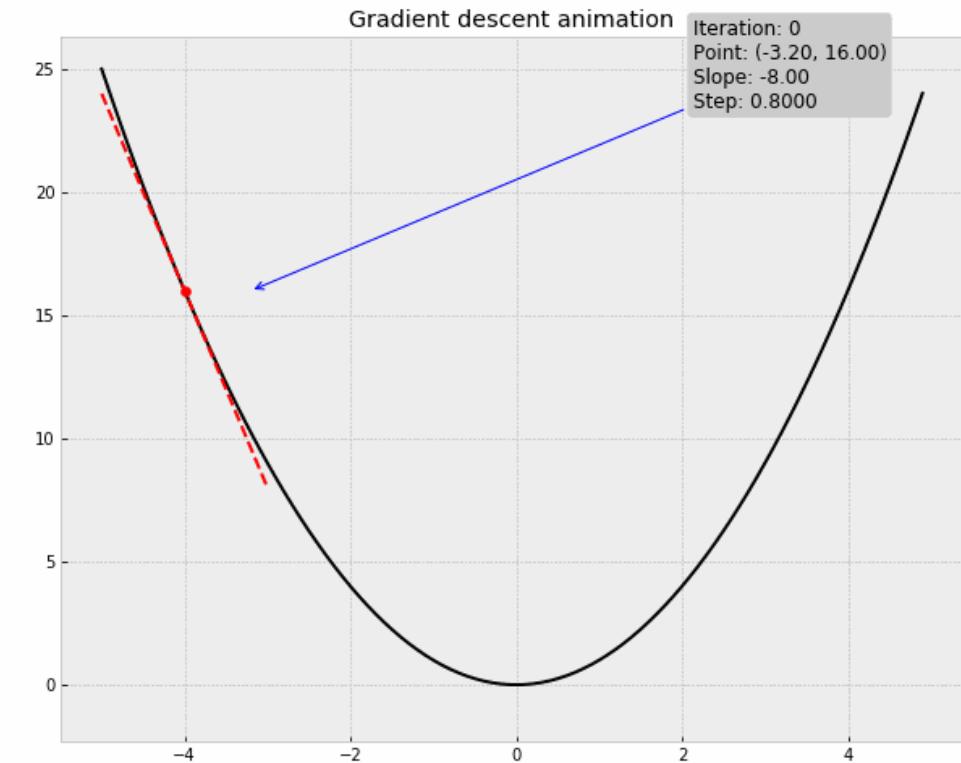
$$L_{CE} = -\frac{1}{n} \sum_i^n \sum_j^m t_{ij} \ln(y_{ij})$$

Where:

- n – nb. of training instances
- m – nb. of classes
- t_{ij} – label of instance i from class j
- y_{ij} – output (predicted label) for instance i from class j

Gradient

- Gradient
 - Another name for the **first derivative**
 - Gradient determines the **slope of tangent** of the graph for a function
 - In other words, it points to the direction of the **greatest rate of increase**
 - By changing the input of the function towards the **opposite direction**, we can reach a **global minimum**



<https://www.kaggle.com/code/trolukovich/animating-gradient-descent>

Backpropagation with Stochastic Gradient Descent



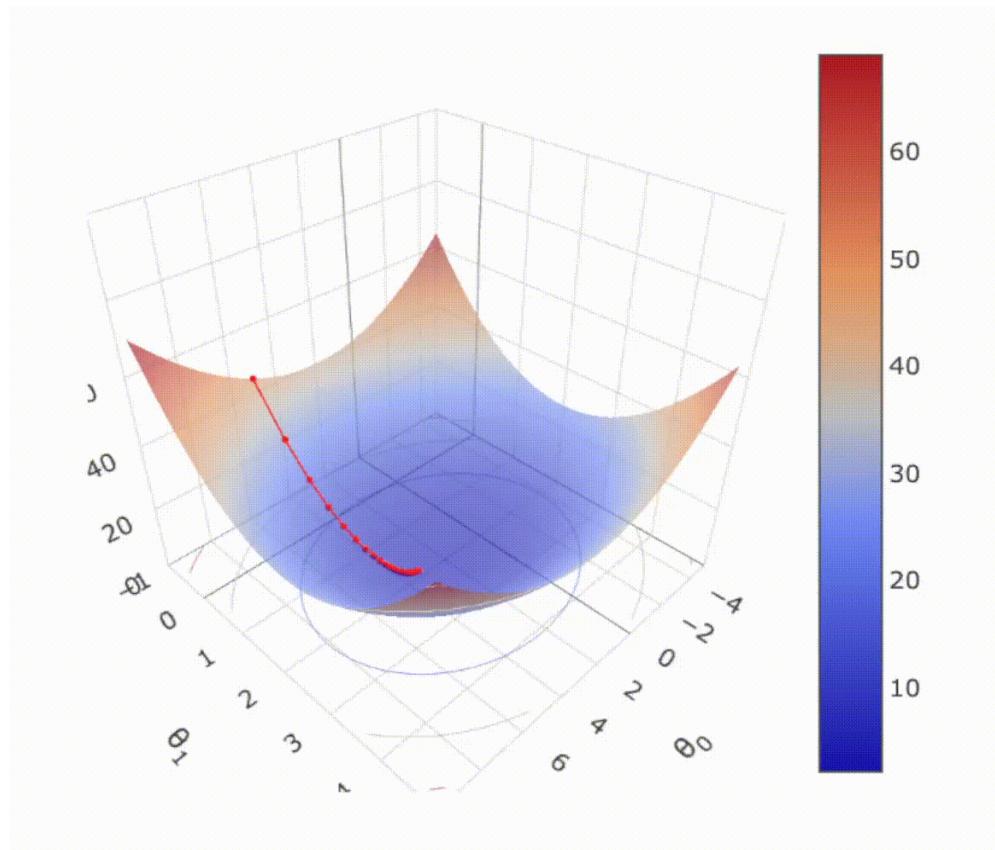
- Given that we want the output of the neural network to be similar to the target, we will perform **Gradient Descent** on the cost (loss) function.
- So, the cost function C will depend the neural network's weights: $C(w)$
- Minimizing the cost function = **adjusting the weights** such that the outputs are close to the target values.
- Adjusting the weights is done by a small rate (usually between 0 and 0.5) value η , called **learning rate**.
- The value of η is set empirically, based on each situation
 - Too small -> slow learning
 - Too big -> fluctuation in cost function -> the global minimum is missed (overshooting)

- **Training steps**
 1. Load an element from the training dataset (feature vector x_i and label t_i)
 2. Obtain prediction y_i by performing feed-forward
 3. Calculate the loss (error), using the prediction y_i and the label t_i
 4. Backpropagate the error using gradients and update the parameters (weights and biases)
- After doing this for each element from the training dataset we complete a **training epoch**
- The training procedure involves repeating epochs for a number of times, depending on the task.

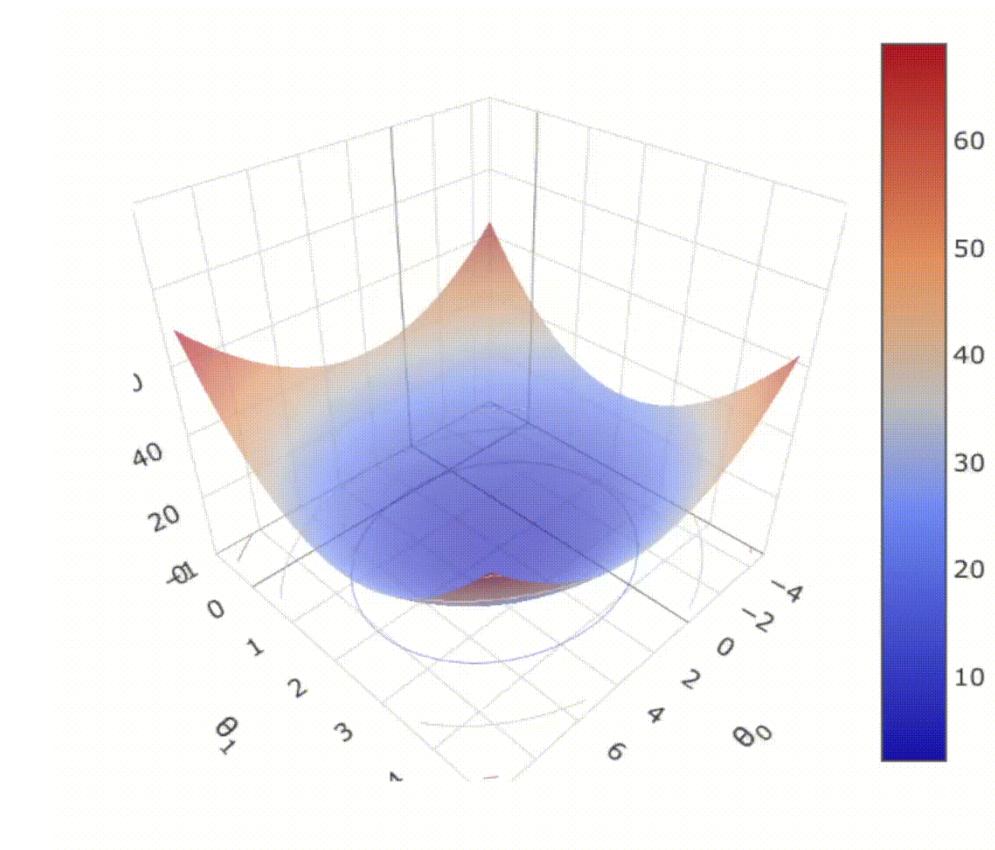
Concepts – Mini-Batch Training

- Classic training
 - Processes **element** at a time, computing the gradient and updating the weights after each
 - Con: a small number of **noisy data** points can throw off the training
- **Mini-batch** training
 - Splits the dataset into **smaller chunks**
 - Each whole **mini-batch** is used to compute the gradient and update weights
 - Pro:
 - A lot **more stable** than classic training, since large gradients generated by noisy data points are **averaged** over the batch
 - It can be slower on CPU, but on **GPU** it makes use of **parallel computing**
 - Practical for large datasets, as it is more **memory efficient**

Gradient Descent Animation



Gradient descent with adequate learning rate

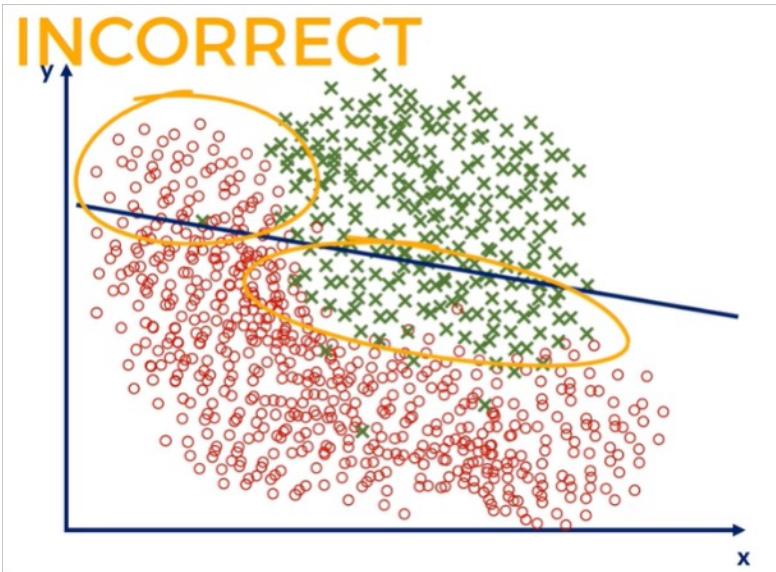


Gradient descent with large learning rate

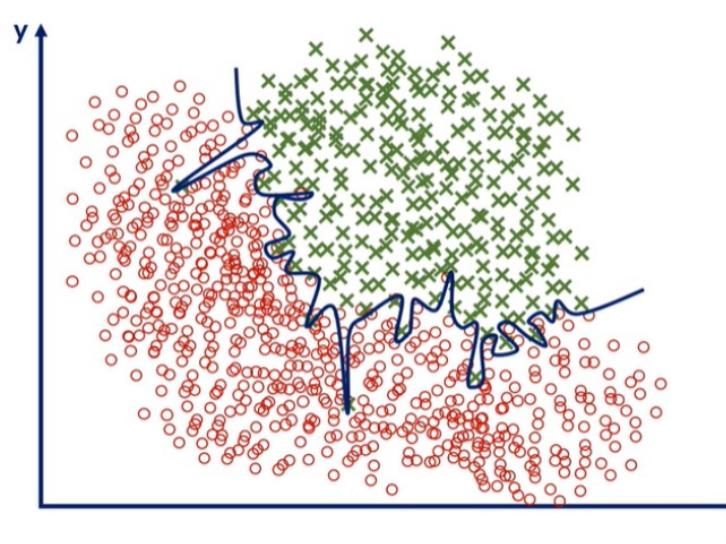
Concepts – Overfitting

- **Overfitting**
 - The purpose of training is to obtain neural networks capable of accurately **predicting new data**
 - A neural network that performs well on training data, but poorly on new, unseen data, suffers from **overfitting** (i.e. high accuracy on training data, low accuracy on new data)
 - Main cause of overfitting – training dataset is **too small** and **not varied enough**
 - **Solutions**
 - Increase dataset
 - Get more data
 - **Generate new data** from existing dataset (e.g. for images, generate new pictures by applying different transformations such as rotation, skews, blurring and translation)
 - May become impractical (larger dataset = longer training)
 - Reduce network size/complexity
 - **Reduce** number of neurons/layers
 - Implement **Dropout**: at each epoch, select a random batch of neurons to be “ignored”
 - Limit weights values
 - **Regularization** is a technique to penalize large weight values

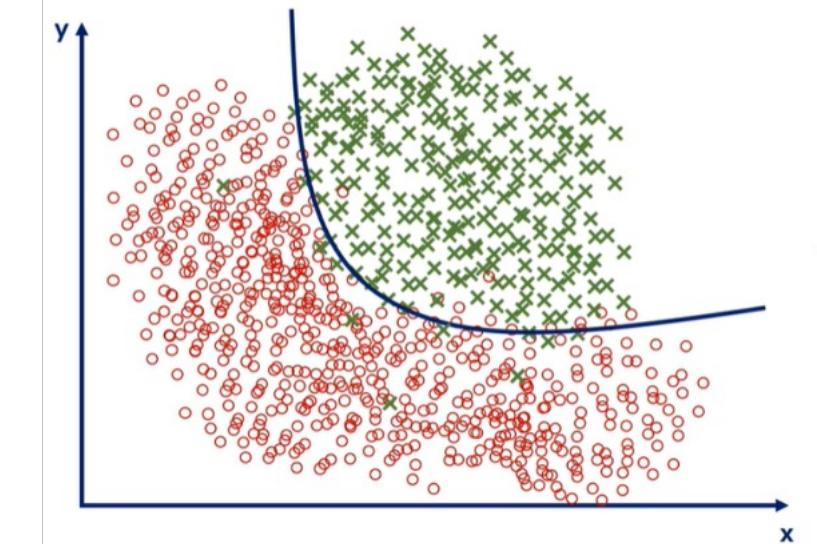
Overfitting cont.



- **Underfit model**
 - Doesn't capture any logic
 - High training loss
 - Low accuracy on new data

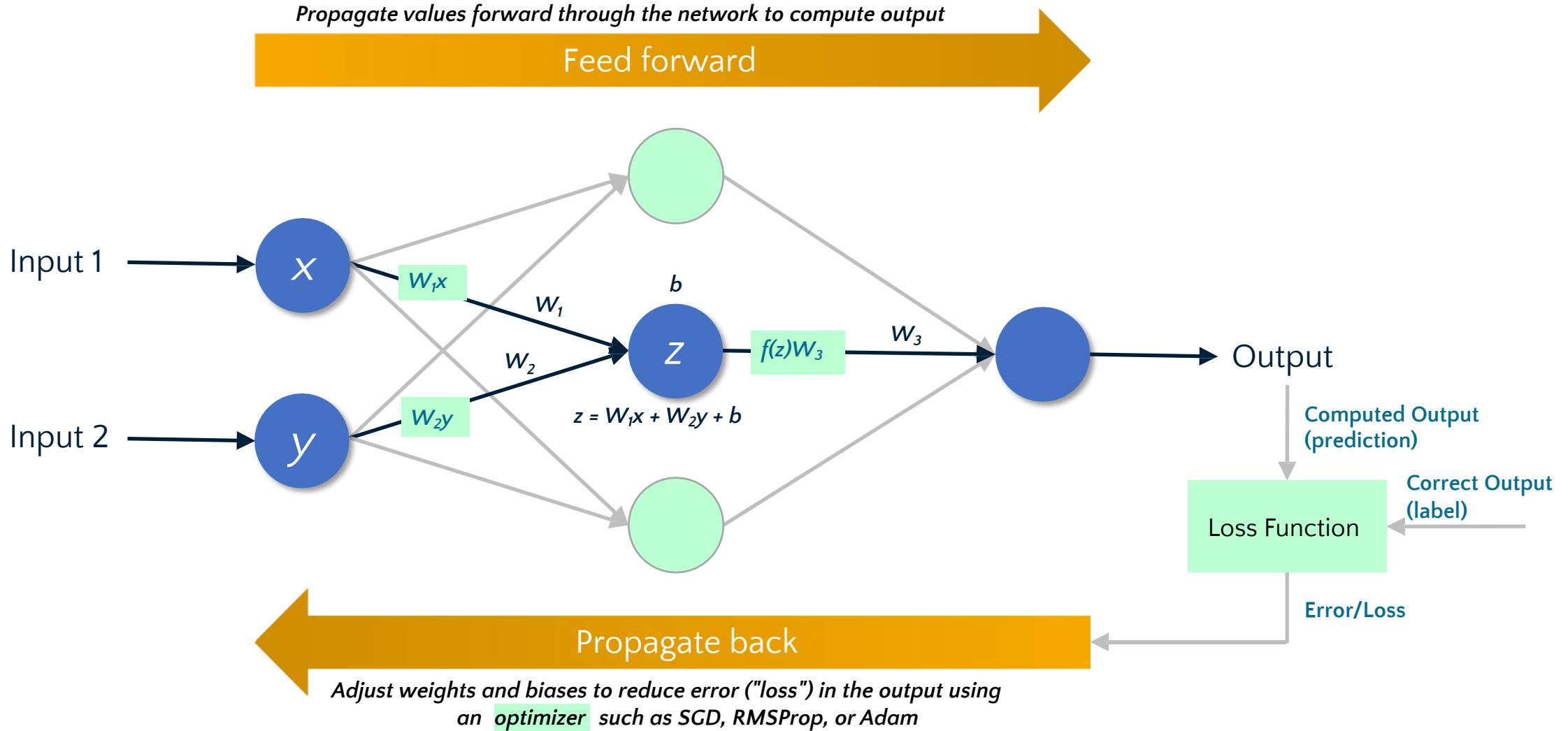


- **Overfit model**
 - Captures all of the noise
 - "Misses the point"
 - Low training loss
 - Low accuracy on new data



- **Right fit**
 - Captures the underlying logic of the dataset
 - Low training loss
 - High accuracy on new data

Training



- **Parameters**
 - Values that are modified after each epoch in order to improve accuracy and minimize loss: **weights** and **biases**
- **Hyperparameters** – configurations that define the behaviour of the training process
 - Number of training **epochs**
 - **Learning rate** – adjusting of parameters is done by a small rate value, usually between 0 and 0.5, called learning rate η
 - **Mini-batch size** – the number of training points used in one iteration to update the parameters
 - Number of **hidden layers** and **layer size** – defines the overall architecture of the model
 - **Activation functions** – the function applied to each layer's output
 - **Optimizer** – the algorithm that is used to update the parameters

Optimizers cont.

- Besides Stochastic Gradient Descent (SGD), there are other optimizers that aim to overcome some of its shortcomings: slow training, sometimes can get stuck on local minima of loss function
- **Adagrad**
 - **Custom learning rate** for each weight, based on the **update magnitude** for each weight
 - this is done by accumulating **all** past gradients for each weight
 - **Frequent/large** updates → **smaller** learning rate
 - **Rare/small** updates → **larger** learning rate
 - **Pro:** performs well on **sparse data**
 - **Con:** can slow down on plateaus

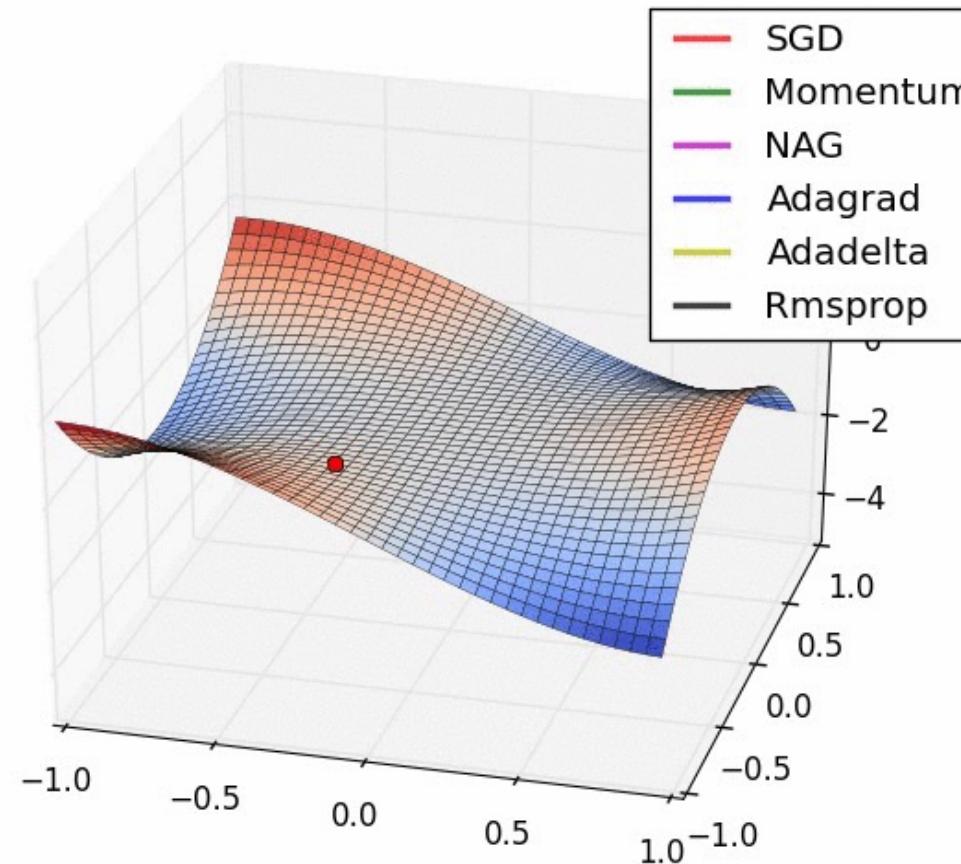
Optimizers cont.

- **RMSProp** (Root Mean Square Propagation)
 - Improvement for Adagrad
 - Accumulates gradients by placing more emphasis on more **recent** values and not all of them
 - Prevents slowing down
- **Adadelta**
 - Improves RMSProp by adapting more intelligently to the past weight updates
 - Compared to RMSProp, it does not require a fixed, initial learning rate, as it is adapted internally
- **Adam**
 - Modified version of RMSProp, with faster convergence
 - Combines also **Momentum** (adapts learning rate for each parameter by not only the history of update values, but also based on the update signs)
 - Also works with sparse networks (when some neurons become inactive)
 - Requires less tuning of hyperparameters
 - Most popular

Optimizers Summary

Optimizer	When to use
Adam	General-purpose, noisy data, sparse gradients.
RMSProp	Sequential data.
Adagrad	Sparse data, adaptively adjusting learning rate.
Adadelta	Stability in training, sparse data, no manual learning rate tuning needed.
SGD	Simple models, didactic.

Optimizers Comparison Animation



<https://imgur.com/a/visualizing-optimization-algos-Hqolp>

Concepts - Weight Initialization

- Choosing the right **weight initialization strategy** is crucial for the performance of neural networks.
- It affects the **speed of convergence, stability** during training, and the **ability to escape poor local minima**.
- Proper initialization helps mitigate problems like **vanishing** or **exploding gradients**.
- Most of the time, when using specialized frameworks for Deep Learning, it is ok to use the default weight initialization strategy. But keep in mind that there might occur situations where tweaking is needed.

Concepts - Weight Initialization

- Most of the initialization techniques use **random**, small values, between 0 and 1, following **distributions** with different parameters for **mean** and **variance** based on the size of the **input** and/or **output** of the layers.
- Since these values are initialized using random distributions, the training procedure usually follows a **non-deterministic** path.
- In situations where we want more predictable results, we can limit the randomness by applying **random seeds** (i.e. the weights are still randomly generated, but with the same values every time)

Concepts - Weight Initialization

Technique	Pros	Cons
He Initialization	Ideal for ReLU, helps with gradient issues	Less effective for sigmoid/tanh
Xavier/Glorot Initialization	Balances gradients for sigmoid/tanh	Not optimal for ReLU
Random Initialization (Uniform/Normal distribution)	Simple to implement	Can lead to vanishing/exploding gradients.
Zero/constant Initialization	Simple to implement	Equal weights

Neural Network Types

Neural Network Type	Description	Uses
Feedforward Neural Networks (FNN)	Simplest neural network with data flowing in one direction.	- Classification and regression tasks. - Didactic
Recurrent Neural Networks (RNN)	Designed for sequential data, with loops allowing information to persist.	- Natural language processing (e.g., text generation). - Time-series analysis. - Speech recognition and translation.
Long Short-Term Memory Networks (LSTMs)	A type of RNN excelling at capturing long-term dependencies.	- Sentiment analysis. - Predictive text and chatbots. - Music composition and sequential data modeling.
Transformer Networks	Utilizes self-attention mechanisms, foundational to modern NLP.	- Large language models (e.g., GPT, BERT).

Neural Network Types

Neural Network Type	Description	Uses
Generative Adversarial Networks (GANs)	Composed of a generator and discriminator, trained adversarially to generate data.	<ul style="list-style-type: none">- Image generation (e.g., deepfakes).- Data augmentation.- Image upscaling (e.g. DLSS)
Convolutional Neural Networks (CNN)	Specialized for grid-like data (e.g., images), using convolutional layers to extract features.	<ul style="list-style-type: none">- Image classification and recognition (e.g., object detection).
Autoencoders	Networks for unsupervised learning, encoding and decoding data to reconstruct input.	<ul style="list-style-type: none">- Dimensionality reduction.- Anomaly detection.- Denoising images or signals.
Graph Neural Networks (GNNs)	Processes graph-structured data, learning node, edge, and graph-level representations.	<ul style="list-style-type: none">- Social network analysis.- Molecular property prediction.- Recommendation systems.- 3D data



Neural Networks – Demo

Thank You!

Danke

Multumesc

Hvala!