Artificial Intelligence

Michal Vašinek

VŠB-Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science,
Department of Computer Science

March 25, 2024

Demonstrations

- **Supporting code:** All demonstrations are available at Google Colab

What are Neural Networks?

Neural Network

A system of interconnected neurons that can be trained and used for pattern recognition, prediction, classification, and other tasks.

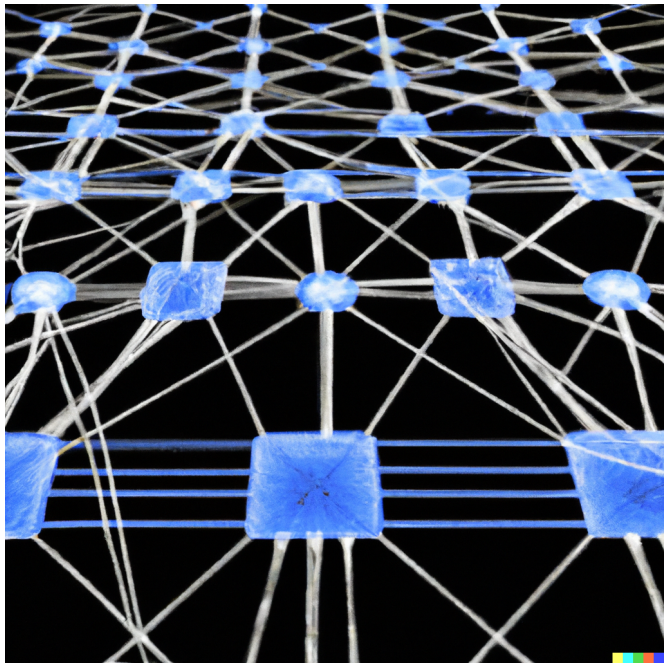


Figure: OpenAI - DALL-E 2 generated picture. Query: artificial neural network.

Basic concepts - Neuron

Neuron

Basic building block of neural networks, simplified model of a human brain cell.

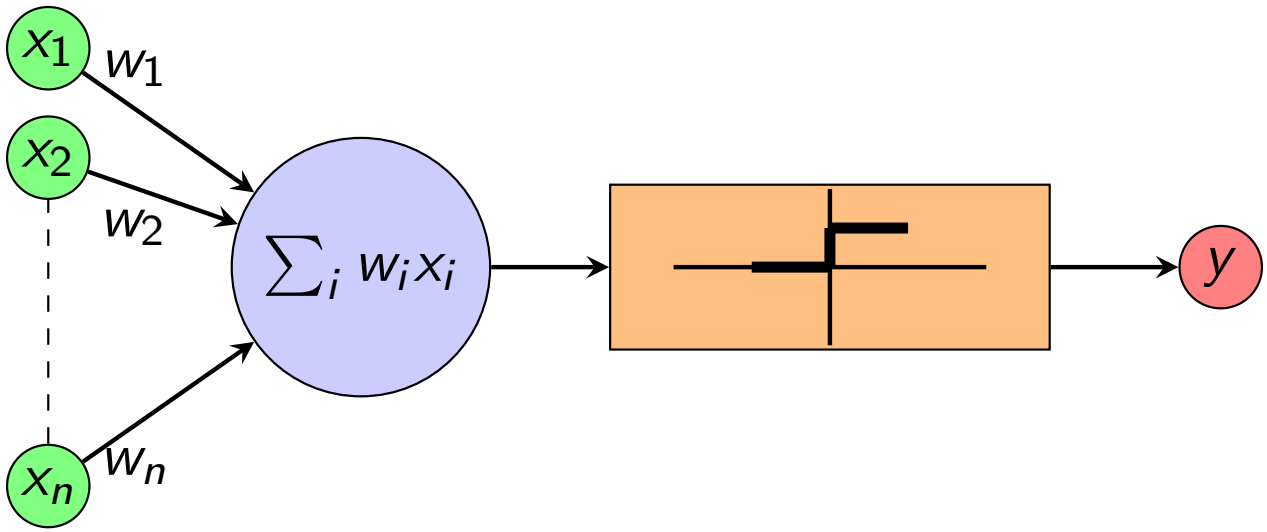


Figure: McCulloch-Pitts artificial neuron model.

Basic concepts - Architectures

Architectures

Neural networks can have different ways of organizing neurons into functional hierarchies, including single-layer perceptrons and complex deep neural networks.

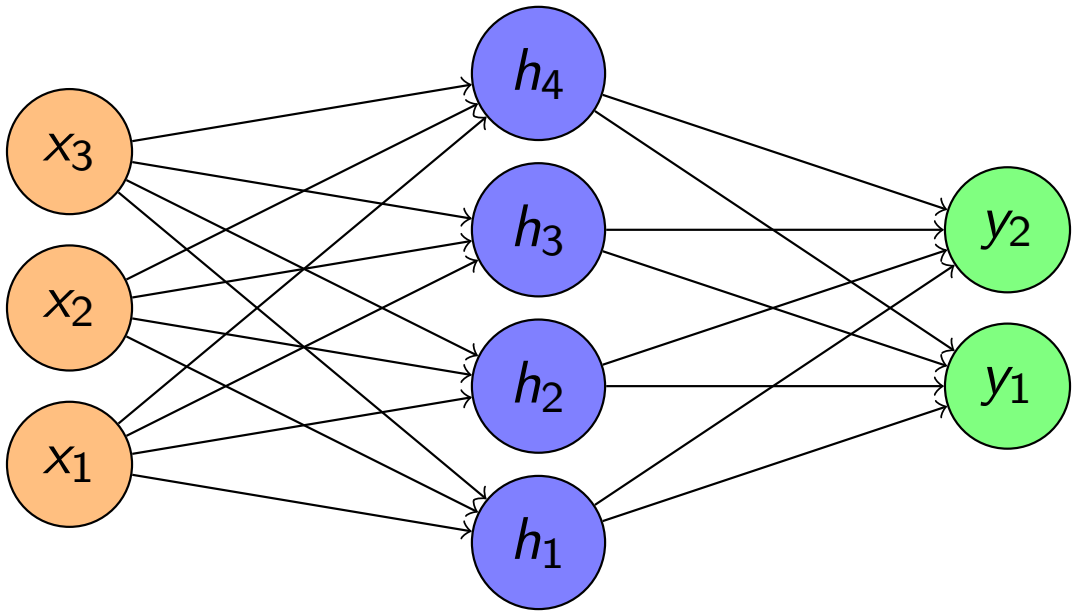


Figure: Fully connected neurons with one hidden layer and two neurons at the output.

Basic concepts - Learning

Learning

- Neural networks learn from training data using learning algorithms such as Backpropagation.
- Neural network learning is based on adapting weights to reduce the difference between expected and predicted output.

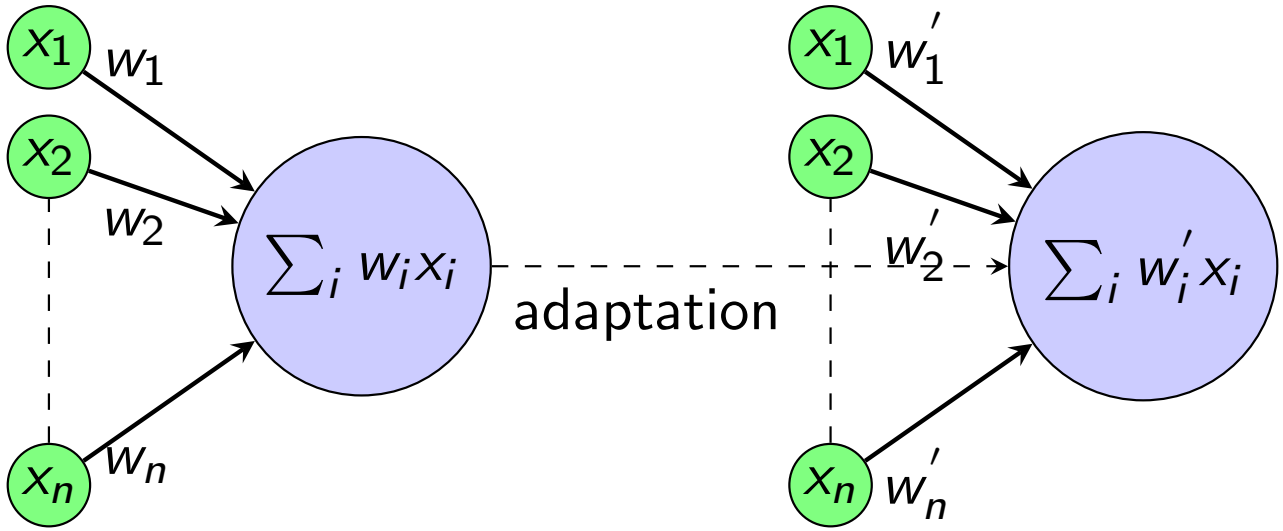


Figure: Learning is a process of neural network weight adaptation.

Basic concepts - Activation function

Activation function

Each neuron uses an activation function to determine the output based on weighted inputs and a threshold.

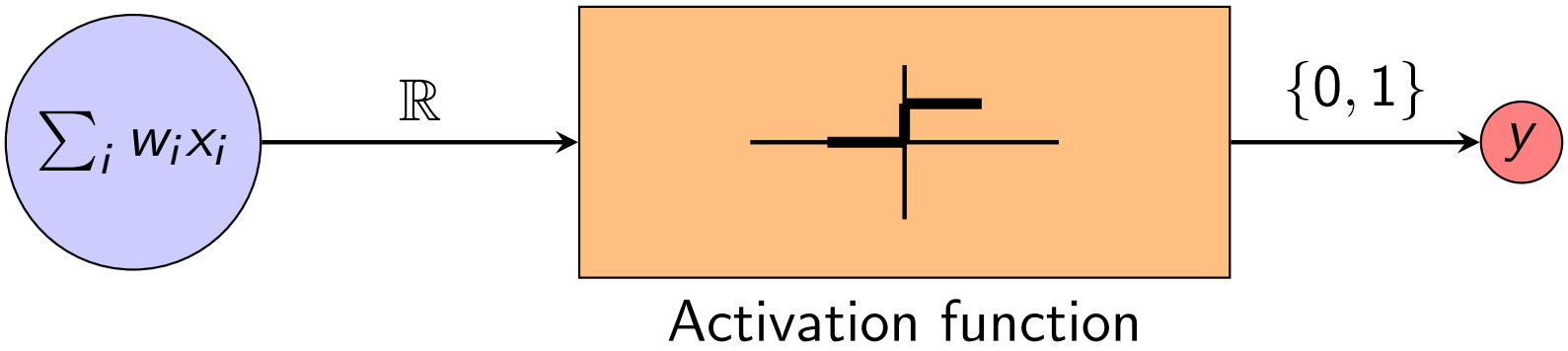


Figure: The activation function maps the input value space to the output value space. In this case, it maps the set of real numbers $\mathbb{R}$ to the set $\{0, 1\}$.

Neural Networks - Historical Perspective

- **1943**: McCulloch-Pitts neuron model
- **1957**: Frank Rosenblatt introduced perceptron
- **1980s**: Backpropagation
- **1990s**: SVMs (Support Vector Machines)
- **2000s**: Deep Neural Networks - Jeffrey Hinton
- **2010s**: Deep Learning (DL), winner in ImageNet competition.
- **2020s**: Neural Networks everywhere, self-driving vehicles, large language models(OpenAI - ChatGPT, Google - Bard), DL based language translation(DeepL).

Applications in Management

- **Financial Forecasting:** Neural networks can be used to predict stock prices and market trends.
- **Customer Relationship Management:** Analyzing customer data to improve customer satisfaction and retention.
- **Supply Chain Optimization:** Predicting demand and optimizing inventory levels.
- **Employee Performance Analysis:** Evaluating employee performance and identifying areas for improvement.
- **Fraud Detection:** Detecting fraudulent activities and transactions.

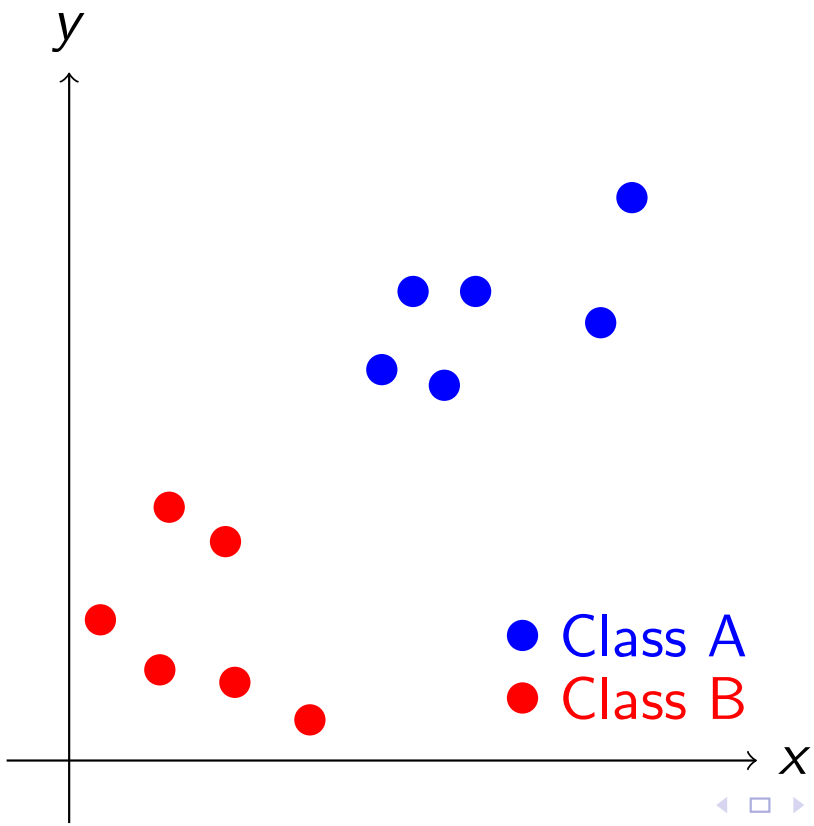
Challenges and Considerations

- **Data Quality:** Neural networks require high-quality and relevant data for accurate predictions.
- **Interpretability:** Neural networks can be complex and difficult to interpret, making it challenging for managers to understand the reasoning behind predictions.
- **Ethical Considerations:** Addressing ethical concerns related to data privacy, bias, and fairness in decision-making.

Motivation

Binary classification

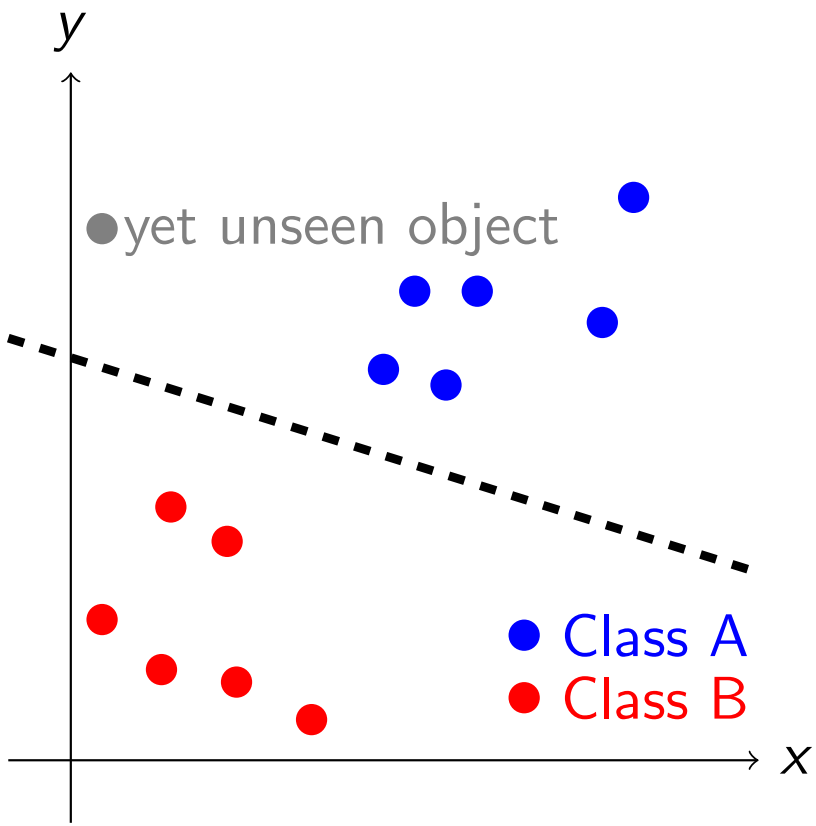
Suppose we have records representing two classes of objects and that we have measured two values of x and y describing each object. Objective is to find a line that best separates the two classes.



Motivation

Binary classification

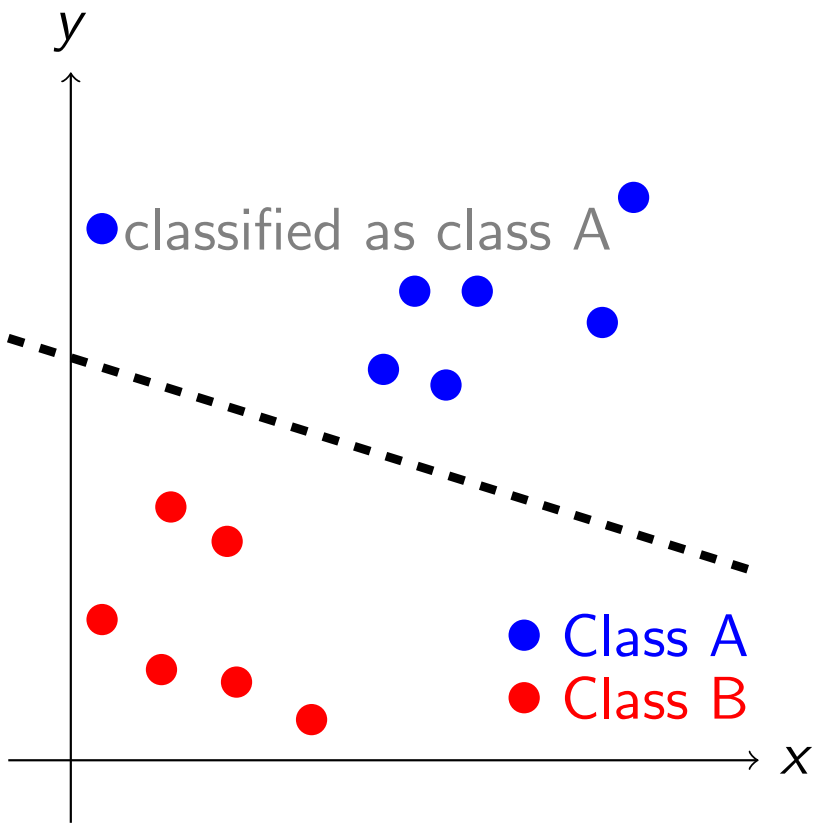
If we can find a separating line, then we might be able to automatically classify records by x and y values.



Motivation

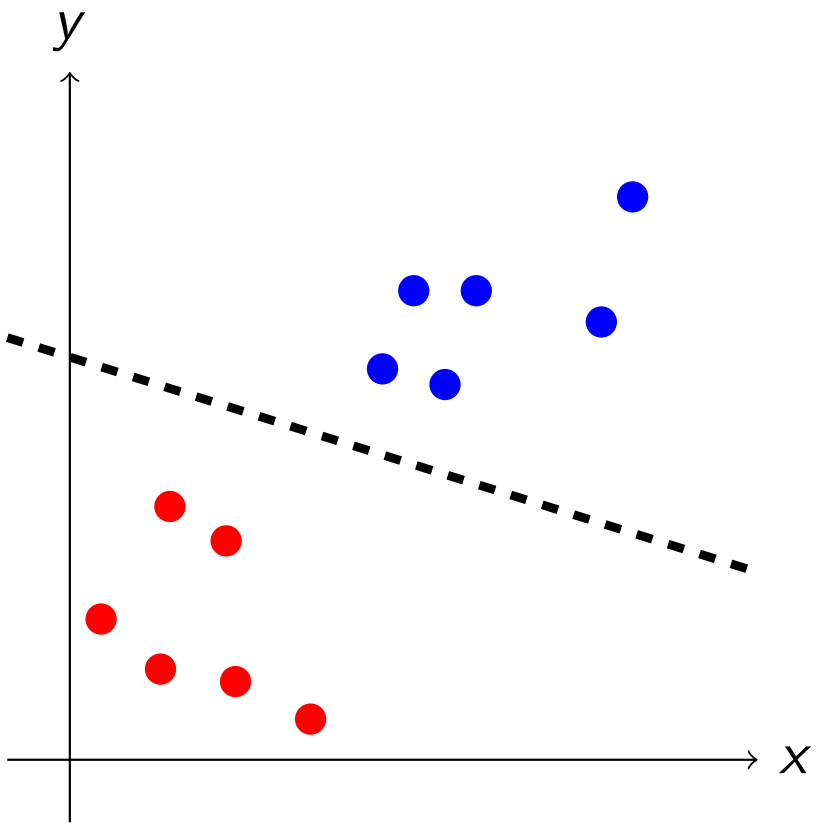
Binary classification

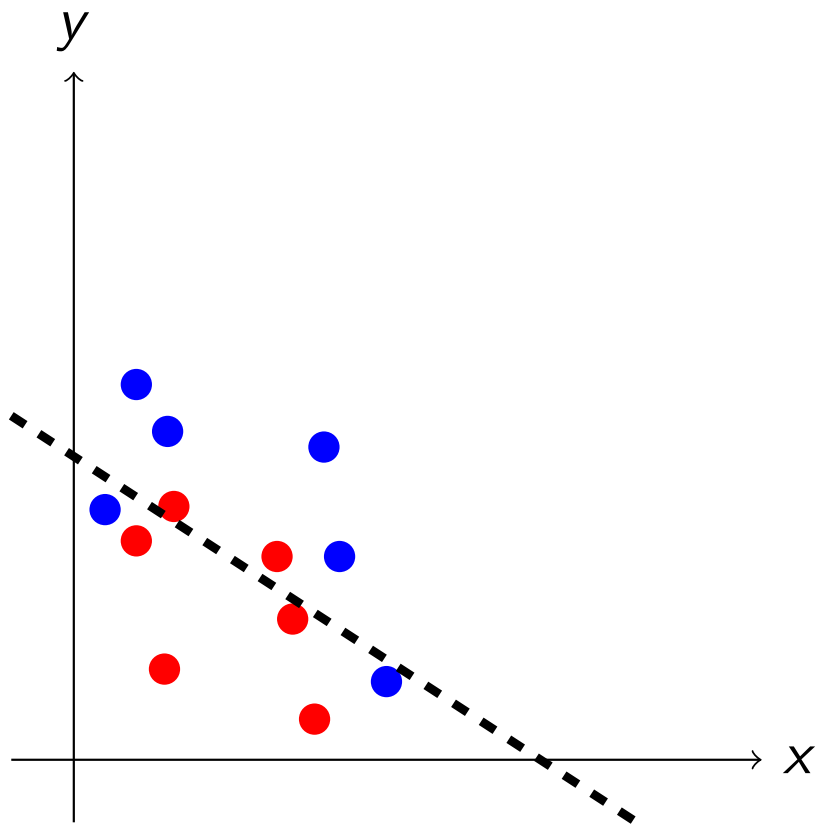
If the object is above the separating line then it is a class A object otherwise it is a class B object.



Non-linearly Separable Dataset

- **Linearly separable** - we are able to find the separating line.
- **Non-linearly separable** - we are unable to find the separating line.

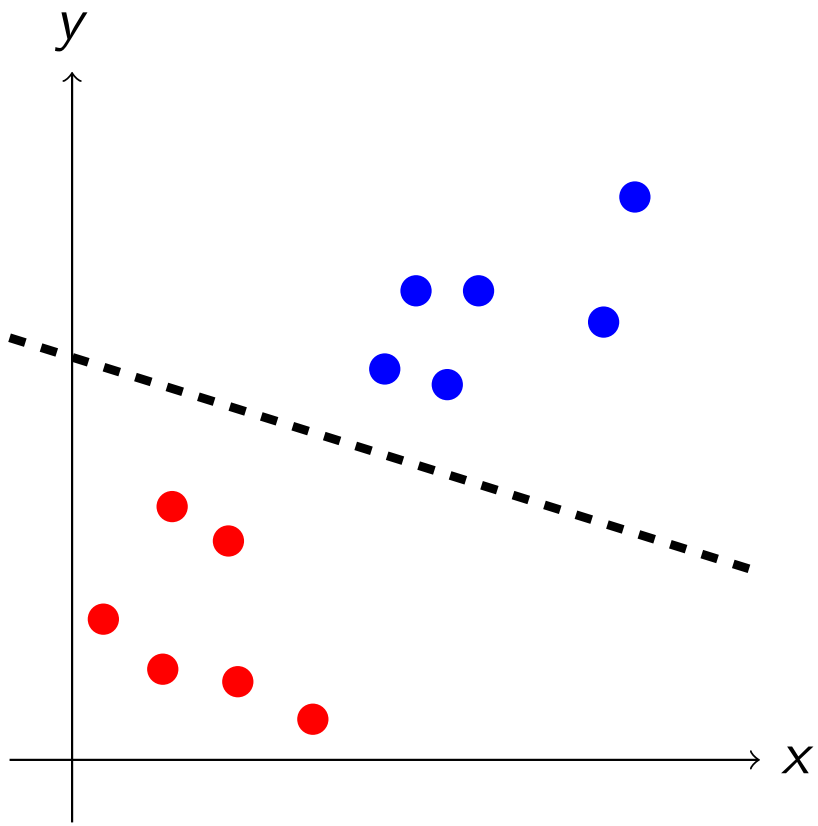


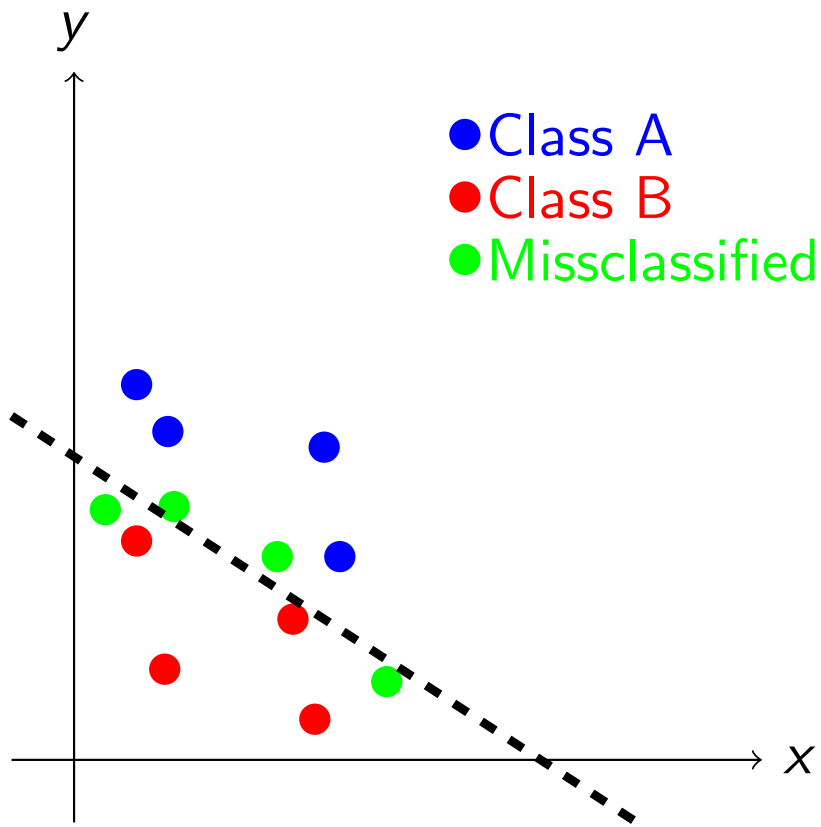


Non-linearly Separable Dataset

Misclassified records

If we try to separate objects from a non-linearly separable dataset with a straight line, we get misclassified objects.





Perceptron

Definition

A perceptron is the simplest form of a neural network unit, capable of binary classification tasks.

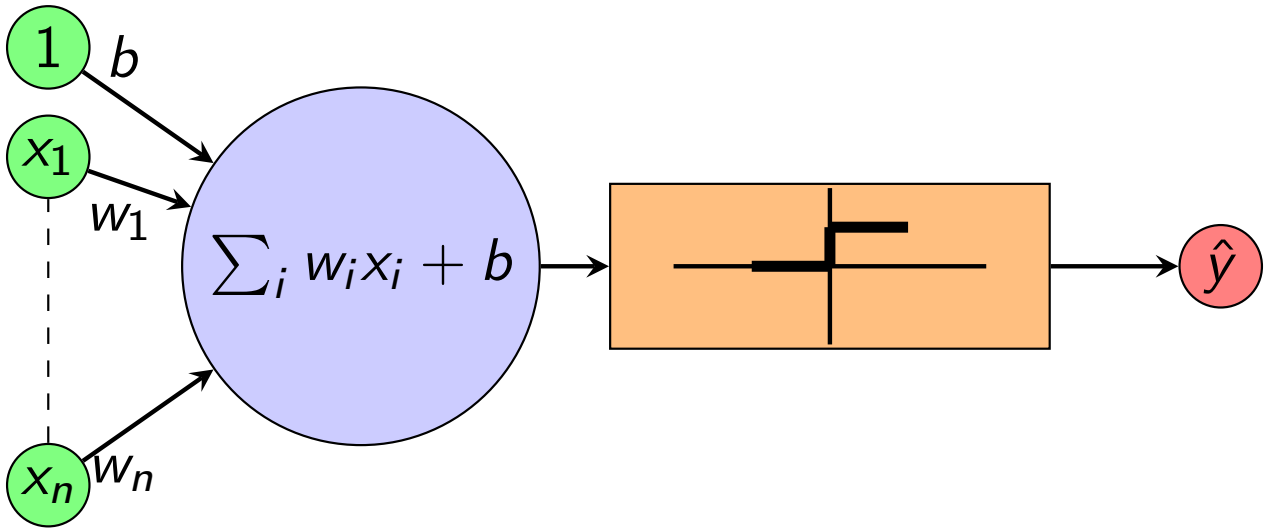


Figure: Perceptron model.

Perceptron

- **Mathematical Expression:** The output $\hat{y}$ of a perceptron for an input vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and weights $\mathbf{w} = (w_1, w_2, \ldots, w_n)$ is calculated using the following formula:

$$\hat{y} = \text{Step}\left(\sum_{i=1}^{n} x_i \cdot w_i + b\right)$$

where b is the bias term, and $\text{Step}(z)$ is the step function defined as:

$$\text{Step}(z) = \begin{cases} 1, & \text{if } z > 0 \\ 0, & \text{otherwise} \end{cases}$$

- **Decision Boundary:** The perceptron classifies inputs based on a linear decision boundary in the input space.

Perceptron Forward Pass: Example

Example - default parameters

Let's consider a perceptron with two input features $x_1 = 1.2$ and $x_2 = -0.7$, weights w_1 and w_2, and bias b. Weights and bias: $w_1 = 0.6$, $w_2 = -0.8$, $b = -0.5$.

- **Forward Pass Calculation:** The weighted sum of inputs plus bias is computed as follows:

$$z = x_1 \cdot w_1 + x_2 \cdot w_2 + b$$

- **Activation:** Applying the step function, the output $\hat{y}$ is determined:

$$\hat{y} = \text{Step}(z) = \begin{cases} 1, & \text{if } z > 0 \\ 0, & \text{otherwise} \end{cases}$$

- **Example Calculation:** For $x_1 = 1.2$ and $x_2 = -0.7$, compute $\hat{y} = \mathit{Step}(1.2 \cdot 0.6 + (-0.7) \cdot (-0.8) - 0.5 = \mathit{Step}(0.78) = 1$.

Perceptron Training: Weight and Bias Update

Objective

Train the perceptron to correctly classify input patterns by adjusting its weights ($\mathbf{w}$) and bias (b).

- **Update Rule:** The weights and bias are updated using the perceptron learning algorithm. For a misclassified input $(\mathbf{x}, y)$ where y is the target output (0 or 1) and $\hat{y}$ is the predicted output, the update rule is:

$$w_i \leftarrow w_i + \alpha \cdot (y - \hat{y}) \cdot x_i$$
$$b \leftarrow b + \alpha \cdot (y - \hat{y})$$

  where α is the learning rate and x_i is the ith component of the input vector $\mathbf{x}$.

- **Effect:** The update rule pushes the decision boundary closer to the misclassified point, improving the perceptron's classification accuracy.

Perceptron Weight Update: Example

- **Objective:** Train a perceptron to classify inputs correctly. Let's consider a misclassified input $\mathbf{x} = (1.2, -0.7)$ with target output $y = 0$.
- **Initial Weights and Bias:** $w_1 = 0.6$, $w_2 = -0.8$, $b = -0.5$.
- **Prediction:** Compute the weighted sum plus bias $z = 1.2 \times 0.6 + (-0.7) \times (-0.8) + (-0.5) = 0.78$.
- **Misclassification:** Predicted output $\hat{y} = \text{Step}(0.78) = 1$ does not match the target $y = 0$.
- **Weight Update:** Update the weights using learning rate $\alpha = 0.1$ and the perceptron learning rule:

$$
\begin{aligned}
w_1 &\leftarrow 0.6 + 0.1 \cdot (0 - 1) \cdot 1.2 = 0.48 \\
w_2 &\leftarrow -0.8 + 0.1 \cdot (0 - 1) \cdot (-0.7) = -0.73 \\
b &\leftarrow -0.5 + 0.1 \cdot (0 - 1) = -0.6
\end{aligned}
$$

Perceptron Weight Update: Example

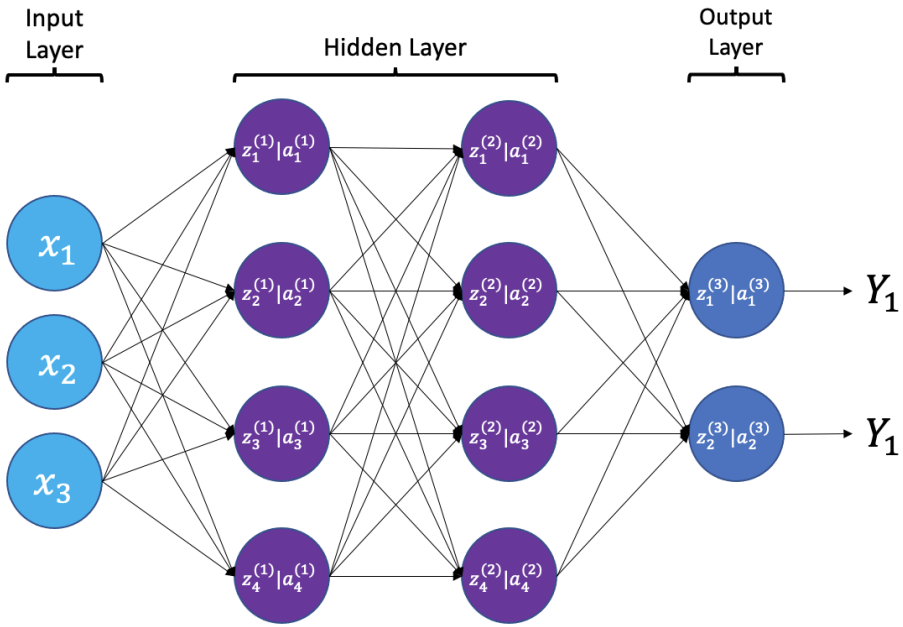
- **Weight Update:** Update the weights using learning rate $\alpha = 0.1$ and the perceptron learning rule:

$$w_1 \leftarrow 0.6 + 0.1 \times (0 - 1) \times 1.2 = 0.48$$
$$w_2 \leftarrow -0.8 + 0.1 \times (0 - 1) \times (-0.7) = -0.73$$
$$b \leftarrow -0.5 + 0.1 \times (0 - 1) = -0.6$$

- **Next prediction:**
  $z = 1.2 \times 0.48 + (-0.7) \times (-0.73) + (-0.6) = 0.78 = 0.487.$
- Output of weighted sum before weights update was 0.78, after adjustment of weights 0.487.
- It is still a misclassification, but it is closer to the expected result.

Dense Neural Networks (DNNs)

- DNNs, also known as fully connected networks, consist of densely interconnected layers of nodes.
- Each node in one layer is connected to every node in the subsequent layer.



Dense Neural Networks (DNNs) - Applications, Demonstration

- DNNs are the foundation of deep learning and can model complex patterns in data.
- They are widely used in various applications, including image and speech recognition, natural language processing, and game playing.
- DNNs employ activation functions like ReLU (Rectified Linear Unit) or sigmoid to introduce non-linearity, enabling them to learn intricate relationships in the data.

Activation Functions

Role

Activation functions introduce non-linearity into the neural network, enabling it to learn complex patterns in the data.

Proper choice may improve learning speed.

Vanishing gradient problem - for very large values of x the output gets very small and it can cause very slow weights update, potentially even stop learning.

Visualization of Sigmoid Function

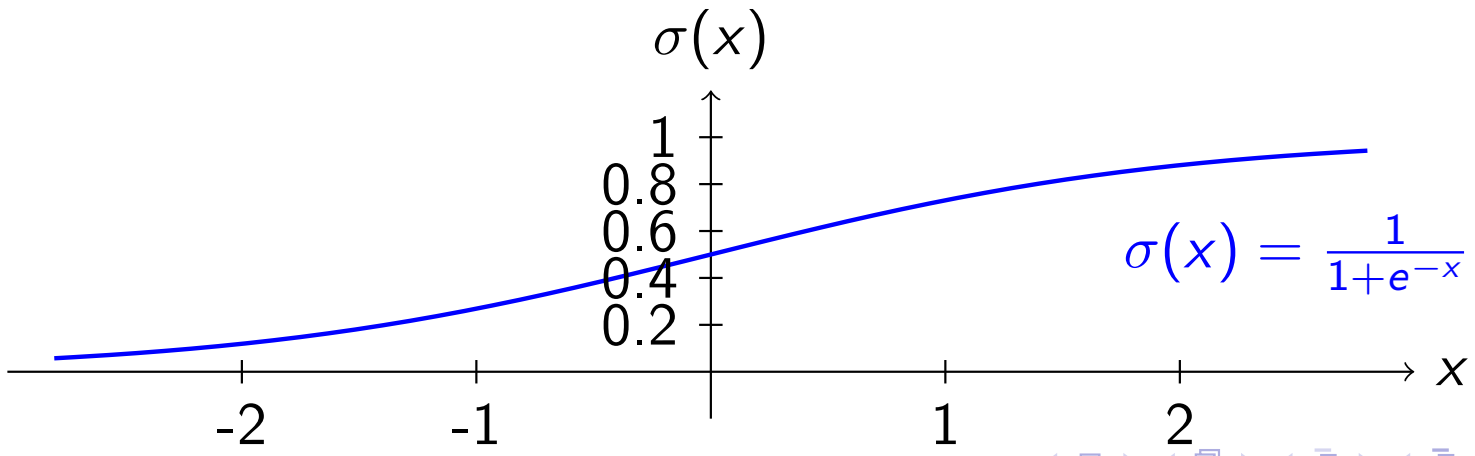
Sigmoid

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

Range: (0, 1)

Output: Squashes input to values between 0 and 1.

Used in older networks, problem with vanishing gradient.



Visualization of Tanh Function

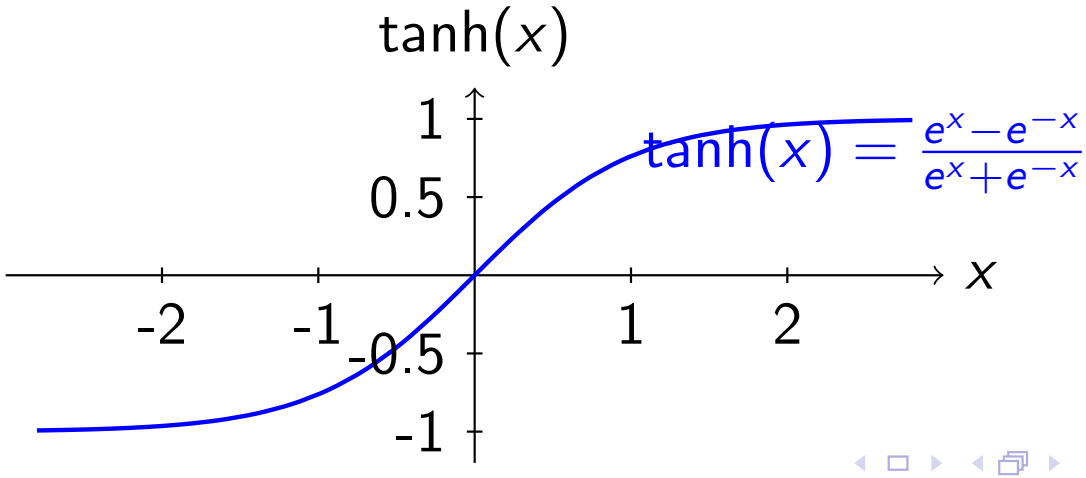
Tanh

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

Range: (-1, 1)

Output: Squashes input to values between -1 and 1, centered around 0.

Prevalently used in recurrent neural networks.



Visualization of ReLU Function

ReLU (Rectified Linear Unit)

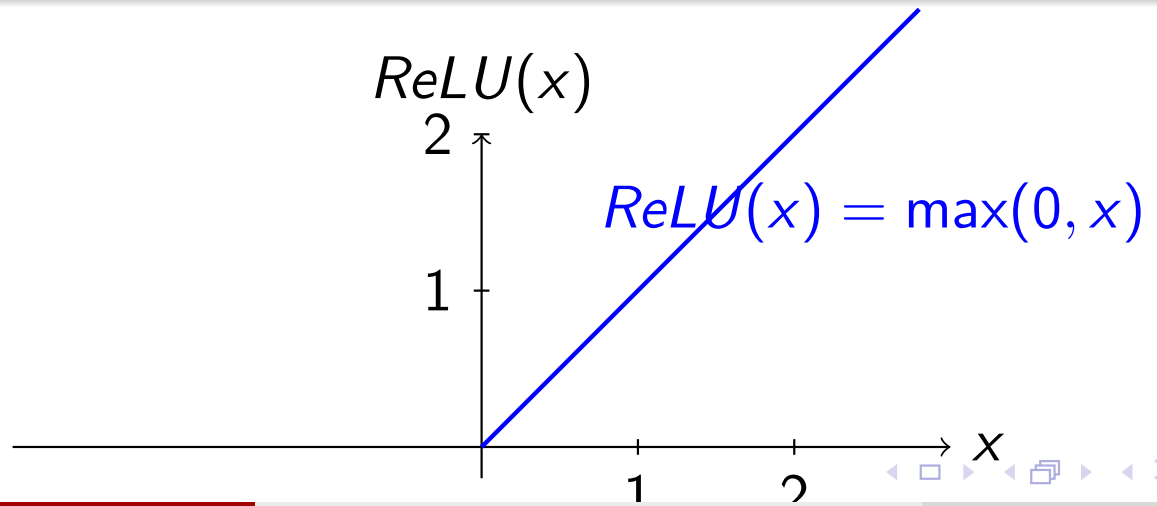
$ReLU(x) = \max(0, x)$

Range: $[0, \infty)$

Output: Keeps positive values unchanged, sets negative values to 0.

Used to solve vanishing gradient problem.

Part of modern deep learning networks.



Visualization of Leaky ReLU Function

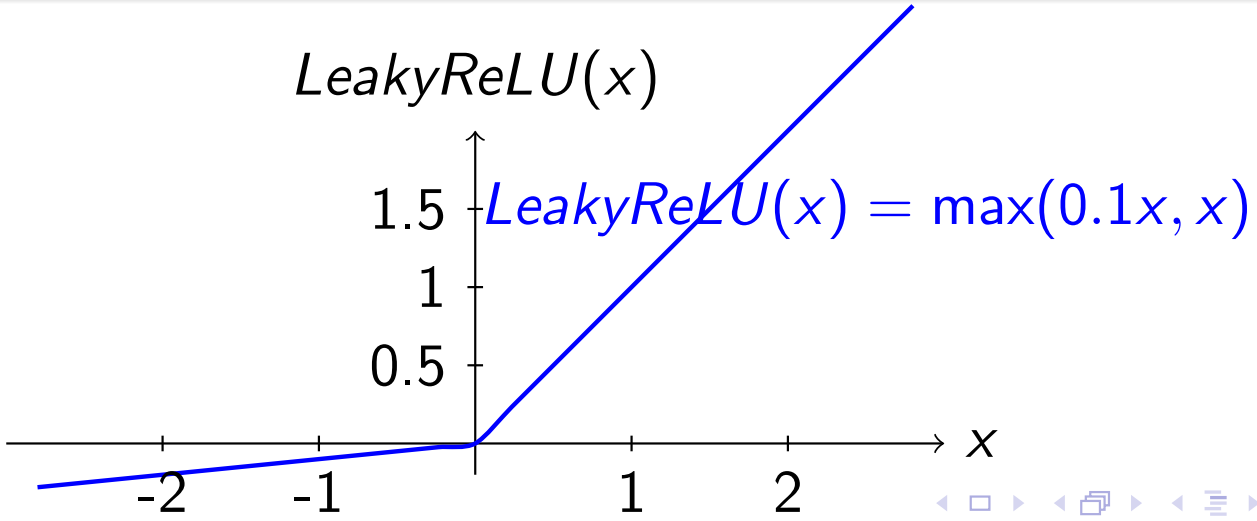
Leaky ReLU

$LeakyReLU(x) = \max(\alpha x, x)$, where α is a small positive constant.

Range: $(-\infty, \infty)$

Output: Similar to ReLU but allows a small gradient for negative values.

Eliminates problem of dying ReLU.



Loss Functions

Role

Loss functions measure the difference between the predicted values (output of the neural network) and the actual values (ground truth) in the training data.

- **Mean Squared Error (MSE):** $\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$
  Measures the average squared difference between predictions ($\hat{y}_i$) and actual values (y_i).
- **Binary Crossentropy:** $-\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$
  Used for binary classification tasks, penalizes deviations from the true class labels.
- **Categorical Crossentropy:** $-\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij}\log(\hat{y}_{ij})$
  Used for multi-class classification tasks, where C is the number of classes.

Backpropagation

Definition

Backpropagation is a supervised learning algorithm used for training artificial neural networks. It is a method for efficiently computing gradients of the loss function with respect to the weights of the network.

- **Steps:**
  1. **Forward Pass:** Compute the predicted output of the neural network using the current weights.
  2. **Loss Computation:** Calculate the loss between the predicted output and the actual target values.
  3. **Backward Pass (Backpropagation):** Compute the gradients of the loss with respect to the network's parameters (weights and biases) by applying the chain rule.
  4. **Gradient Descent:** Update the weights and biases of the network using the computed gradients to minimize the loss.

Backpropagation

- **Key Concepts:**
  - **Chain Rule:** Backpropagation relies on the chain rule of calculus to compute gradients efficiently in a neural network with multiple layers.
  - **Learning Rate:** Learning rate is a hyperparameter that determines the size of the steps taken during gradient descent.
  - **Iterations:** Backpropagation involves multiple iterations (epochs) over the entire training dataset to optimize the network's parameters.

Backpropagation: Mathematical Expression

Objective:

Minimize the loss function L by adjusting the weights and biases of the neural network using gradient descent.

- **Derivation:** Using the chain rule, the derivative of the loss with respect to the weights in layer l is calculated as follows:

$$\frac{\partial L}{\partial W_{ij}^{(l)}} = \frac{\partial L}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial W_{ij}^{(l)}}$$

$$\frac{\partial L}{\partial b_i^{(l)}} = \frac{\partial L}{\partial z_i^{(l+1)}}$$

where $z_i^{(l+1)}$ is the input to neuron i in layer $l+1$.

Backpropagation: Mathematical Expression

- **Weight Update:** After computing gradients, the weights and biases are updated using gradient descent:

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial L}{\partial W_{ij}^{(l)}}$$

$$b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial L}{\partial b_i^{(l)}}$$

where α is the learning rate.

Backpropagation: Derivative in Loss Function

Loss Function: $L = \frac{1}{2}(y - \hat{y})^2$ *(Mean Squared Error)*

Output Layer:

- **Predicted Output:** $\hat{y}$
- **Target Output:** y

Derivative of Loss with Respect to Predicted Output:

$$\frac{\partial L}{\partial \hat{y}} = \hat{y} - y$$

Backpropagating the Error:

- **Gradient at Output Layer:** $\delta^{(output)} = \frac{\partial L}{\partial \hat{y}}$
- **Using Chain Rule for Further Layers:** Derivatives are calculated backwards through the network using the chain rule.

Backpropagation: Derivative Using Chain Rule

Hidden Layer Neuron: $z = \sum_i (w_i \cdot x_i) + b$
Activation Function: $a = \sigma(z)$ *(e.g., Sigmoid)*
Derivative of Loss with Respect to Activation Output:

$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial a}$$

Derivative of Activation Function:

$$\frac{\partial a}{\partial z} = a \cdot (1 - a)$$

(for Sigmoid function)
Derivative of Loss with Respect to Neuron Output:

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z}$$

Backpropagation: Derivative Using Chain Rule

Derivative of Loss with Respect to Weights and Bias:

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial z} \cdot x_i$$

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

Note:

- Derivatives are computed using the chain rule to propagate the error backwards through the network.

Data Preprocessing for Neural Networks

- **Data Cleaning:** Handling missing values, outliers, and noise in the dataset to ensure quality and consistency.
- **Feature Scaling:** Scaling features to a similar range (e.g., normalization) to prevent certain features from dominating others during training.
- **Feature Engineering:** Creating new features from existing ones or transforming features to better represent the underlying patterns in the data.
- **Data Splitting:** Splitting the dataset into training, validation, and test sets for model training, tuning, and evaluation, respectively.
- **Label Encoding/One-Hot Encoding:** Converting categorical variables into numerical representations that can be processed by neural networks.

Normalizing Numerical Data

- **Mean Normalization:** Subtracting the mean of the feature from each data point, then dividing by the standard deviation. Centers the data around zero.
- **Min-Max Scaling:** Scaling features to a specific range, often [0, 1] or [-1, 1], by subtracting the minimum value and dividing by the range.
- **Robust Scaling:** Scaling features using statistics that are robust to outliers, such as the interquartile range (IQR).
- **Log Transformation:** Applying a logarithm to the data to stabilize variance and make the data more symmetric.

Encoding Categorical Data

- **Label Encoding:** Assigning a unique numerical label to each category. Useful for ordinal categorical data.
  Example: Customers feedback
  - Excellent (3): Customers were delighted and very satisfied with their experience.
  - Average (2): Customers had a neutral experience, nothing extraordinary, nothing bad.
  - Poor (1): Customers were very dissatisfied and had a bad experience with the company.
- **One-Hot Encoding:** Creating binary columns for each category. Each column indicates the presence or absence of the category.
- **Embedding Layers:** For large categorical features, embedding layers can be used in neural networks to learn dense vector representations of categories.

Underfitting

Definition

Occurs when a machine learning model is too simple to capture the underlying patterns in the data. It performs poorly not only on the training data but also on unseen (test) data.

- **Causes of Underfitting:**
  - Too few features or too simplistic model architecture.
  - Insufficient training or too aggressive regularization.
- **Solutions:** Consider increasing the model's complexity by adding more layers or neurons, introduce more relevant features, or extend the training duration. Additionally, gathering more diverse and representative data for training can improve the model's ability to learn the problem.

Overfitting

Definition

Occurs when a machine learning model learns the training data too well, including its noise and outliers. It performs well on the training data but poorly on unseen data due to its excessive focus on training data specifics.

- **Causes of Overfitting:**
  - Too complex model with too many features.
  - Limited training data or lack of proper regularization.
- **Solutions:** Finding the right balance between model complexity and dataset size. Techniques like cross-validation, regularization, and feature selection help mitigate underfitting and overfitting problems.

Simple Recurrent Neural Networks (RNN)

Definition:

A Simple RNN is a type of recurrent neural network that maintains a hidden state to capture information about previous time steps. It can process sequential data and is suitable for tasks involving temporal dependencies.

- **Architecture:**
  - Input Layer: Accepts input features at each time step.
  - Hidden Layer: Maintains a hidden state capturing information from previous time steps.
  - Output Layer: Produces predictions or representations based on the hidden state.
- **Recurrent Connection:** The hidden state at time step t serves as an input to the network at time step $t + 1$, creating a recurrent connection.

Block of Simple RNN

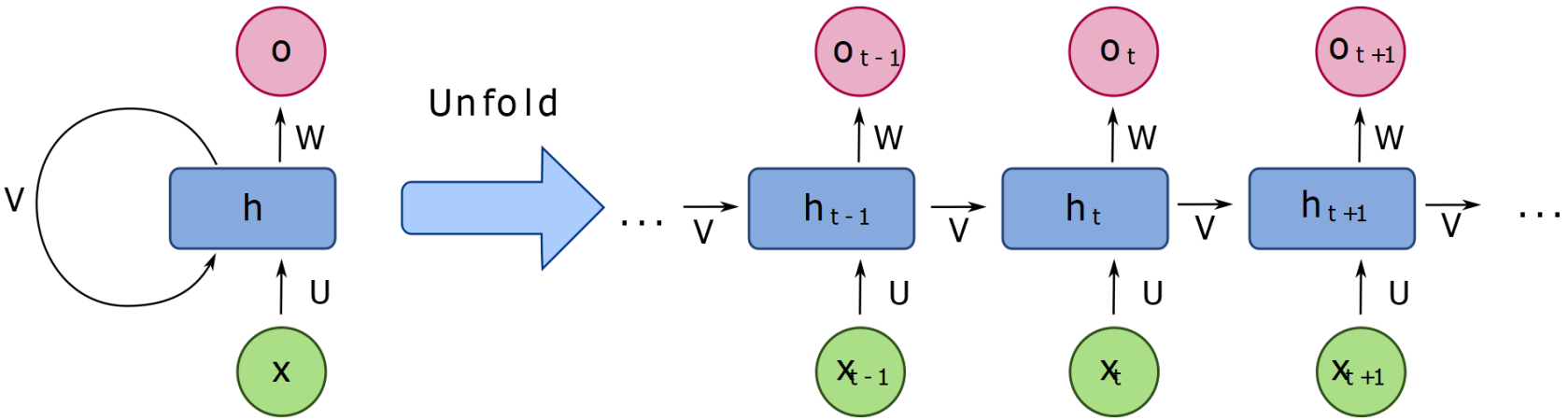


Figure: Recurrent Neural Network, By fdeloche - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=60109157

Computing Hidden State and Output in Simple RNN

Input Sequence: $[x_1, x_2, x_3, ..., x_n]$

Hidden State Computation:

- **Weights:** W_{hx} (Input to Hidden), W_{hh} (Hidden to Hidden), b_h (Bias)
- **Hidden State at Time Step t:**
  $h_t = \tanh(W_{hx} \cdot x_t + W_{hh} \cdot h_{t-1} + b_h)$

Output Computation:

- **Weights:** W_{oh} (Hidden to Output), b_o (Bias)
- **Output at Time Step t:** $y_t = \sigma(W_{oh} \cdot h_t + b_o)$

Note:

- $\tanh(x)$ is commonly used as the activation function for the hidden state, and $\sigma(x)$ (e.g., sigmoid) for the output.
- Adjust the weights (W_{hx}, W_{hh}, W_{oh}) and biases (b_h, b_o) during training to learn from data.

Computing Output in a Simple RNN

Input Sequence: [0.5, 0.3]
Hidden Layer:

- **Weights:** $W_{hx} = 0.6,\ W_{hh} = -0.4,\ b_h = 0.1$
- **Initial Hidden State:** $h_0 = 0$
- **Activation Function:** $\tanh(x)$

Calculation:

- **Time Step 1:**
  $a_1 = W_{hx} \cdot 0.5 + W_{hh} \cdot 0 + b_h = 0.1$
  $h_1 = \tanh(a_1) = 0.0997$
- **Time Step 2:**
  $a_2 = W_{hx} \cdot 0.3 + W_{hh} \cdot 0.0997 + b_h = 0.0788$
  $h_2 = \tanh(a_2) = 0.0786$

Output: The hidden state at the last time step (h_2) can be used as the output of the Simple RNN for the given input sequence.

Simple RNN - applications, demonstration

- **Applications:** Simple RNNs are used in natural language processing, speech recognition, and various other sequence modeling tasks.
- **Limitations:** Simple RNNs struggle with capturing long-term dependencies due to the vanishing gradient problem.

Gated Recurrent Unit (GRU)

Definition:

GRU is a type of recurrent neural network designed to capture long-term dependencies in sequential data. GRU uses gating mechanisms to control the flow of information, allowing it to selectively update its memory content.

- **Components:**
  - **Update Gate:** Determines how much of the past information to retain.
  - **Reset Gate:** Decides how much of the past information to forget.
  - **Hidden State:** Captures the current state of the network and is used to compute the output.
- **Simpler Architecture:** GRU has a simpler architecture compared to LSTM, making it computationally more efficient and easier to train.

Block of GRU

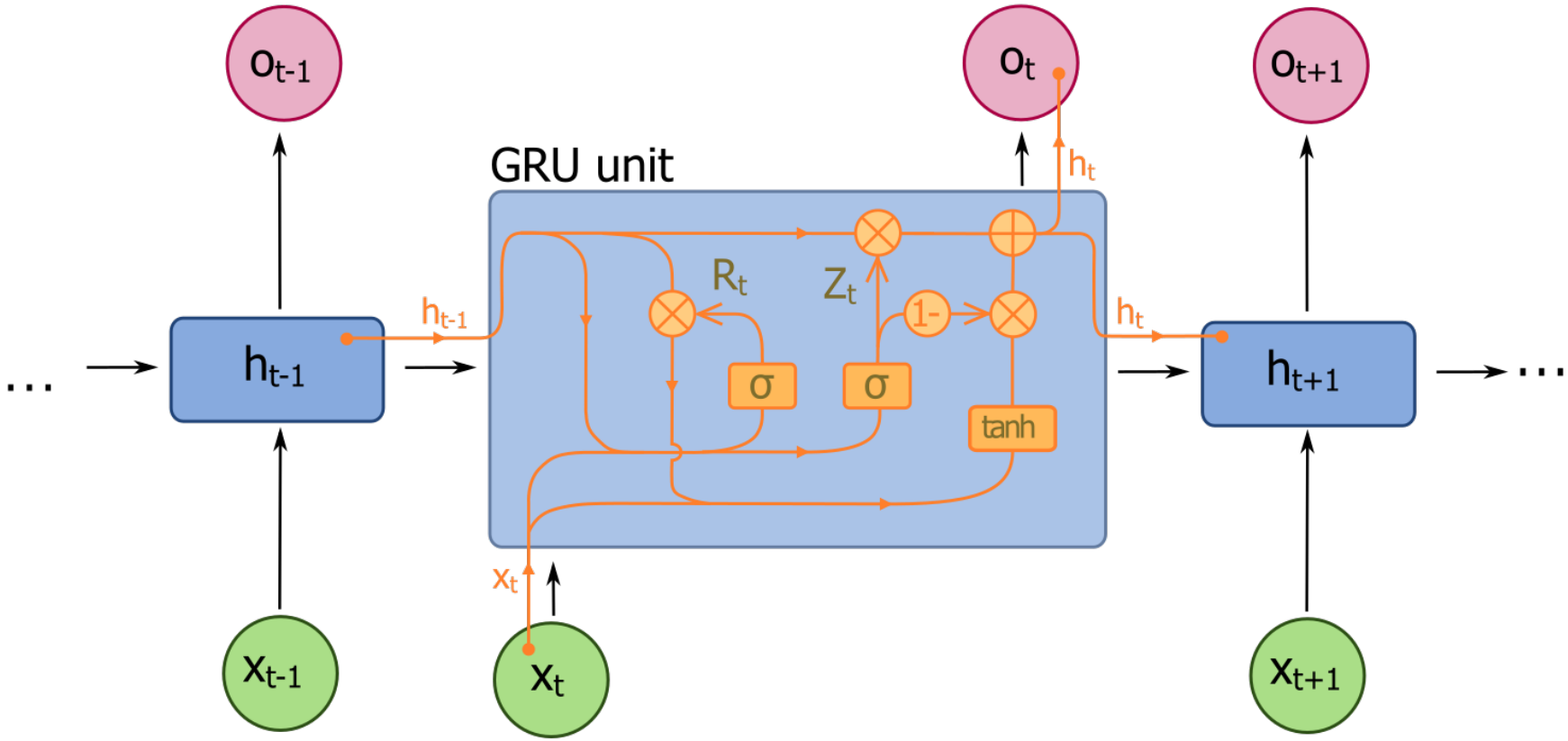


Figure: Gated Recurrent Unit, By fdeloche - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=60466441

Gated Recurrent Unit (GRU) - Applications, Demonstration

- **Applications:** Used in natural language processing, speech recognition, and other sequential data tasks where capturing long-term dependencies is crucial.

Long Short-Term Memory (LSTM)

Definition:

LSTM is a type of recurrent neural network designed to capture long-term dependencies in sequential data. It uses special units called memory cells to store and retrieve information over extended time intervals.

- **Components:**
  - **Memory Cell:** Core component of LSTM, capable of storing and processing information over multiple time steps.
  - **Forget Gate:** Determines what information from the cell's state should be discarded.
  - **Input Gate:** Updates the cell's state with new information.
  - **Output Gate:** Produces the output based on the updated cell's state.
- **Long-Term Dependencies:** Simple LSTM addresses the vanishing gradient problem, enabling the network to capture and learn long-term dependencies in the data.

Block of LSTM

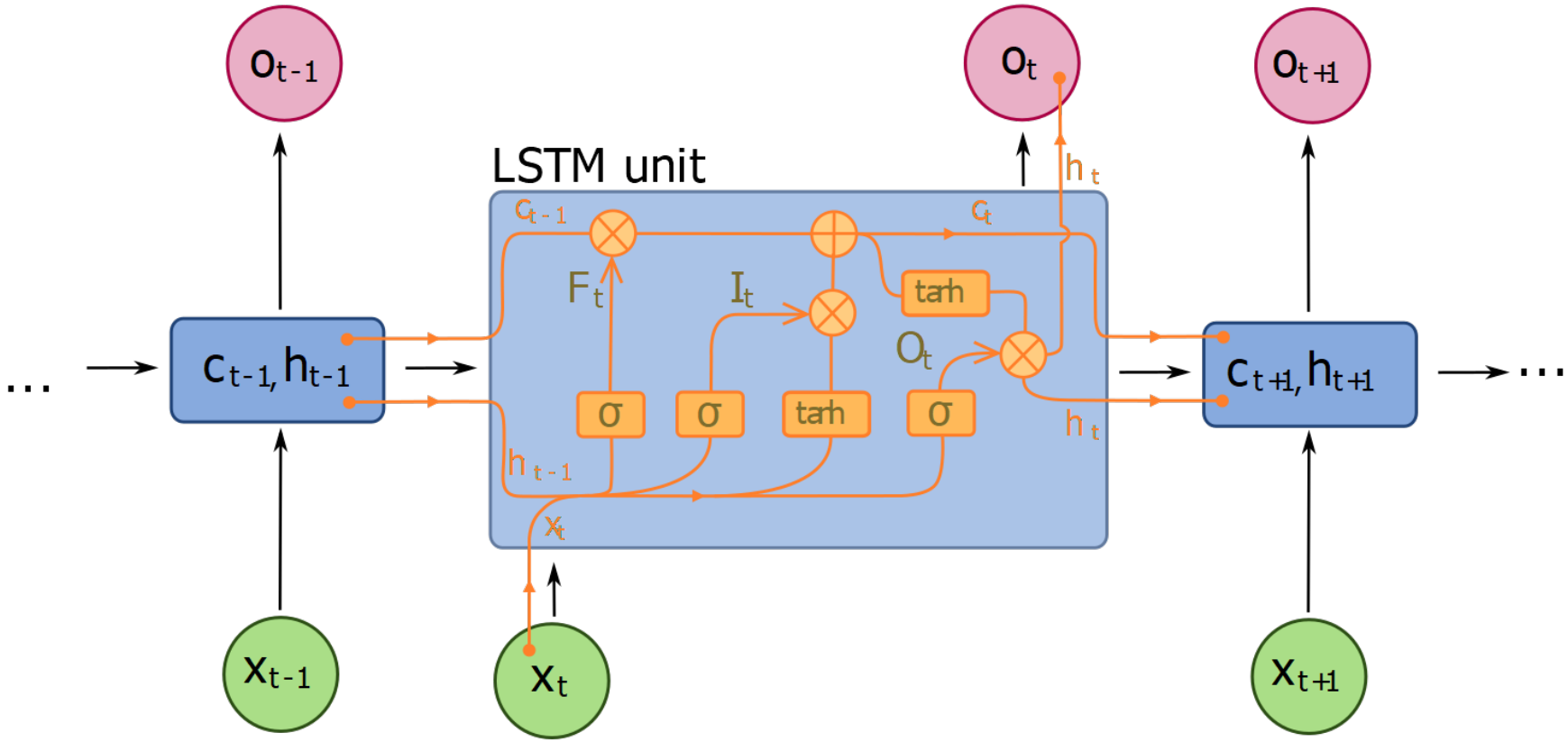


Figure: Long Short-Term Memory, By fdeloche - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=60466441

Long Short-Term Memory (LSTM)

- **Applications:** Used in machine translation, speech recognition, and tasks involving sequential data with extended contexts.

Attention Mechanism

Definition:

Attention mechanism is a concept in neural networks that allows the model to focus on specific parts of the input sequence when making predictions or generating output. It assigns different weights to different parts of the input, emphasizing the relevant information.

- **Components:**
  - **Query, Key, and Value Vectors:** Attention mechanisms use query, key, and value vectors to calculate the attention scores between input elements.
  - **Attention Scores:** Attention scores quantify the importance of each input element. Softmax function is often used to compute these scores.
  - **Weighted Sum:** Input elements are combined using their attention scores to create a weighted sum, which becomes the input for the next layer.

Attention Mechanism

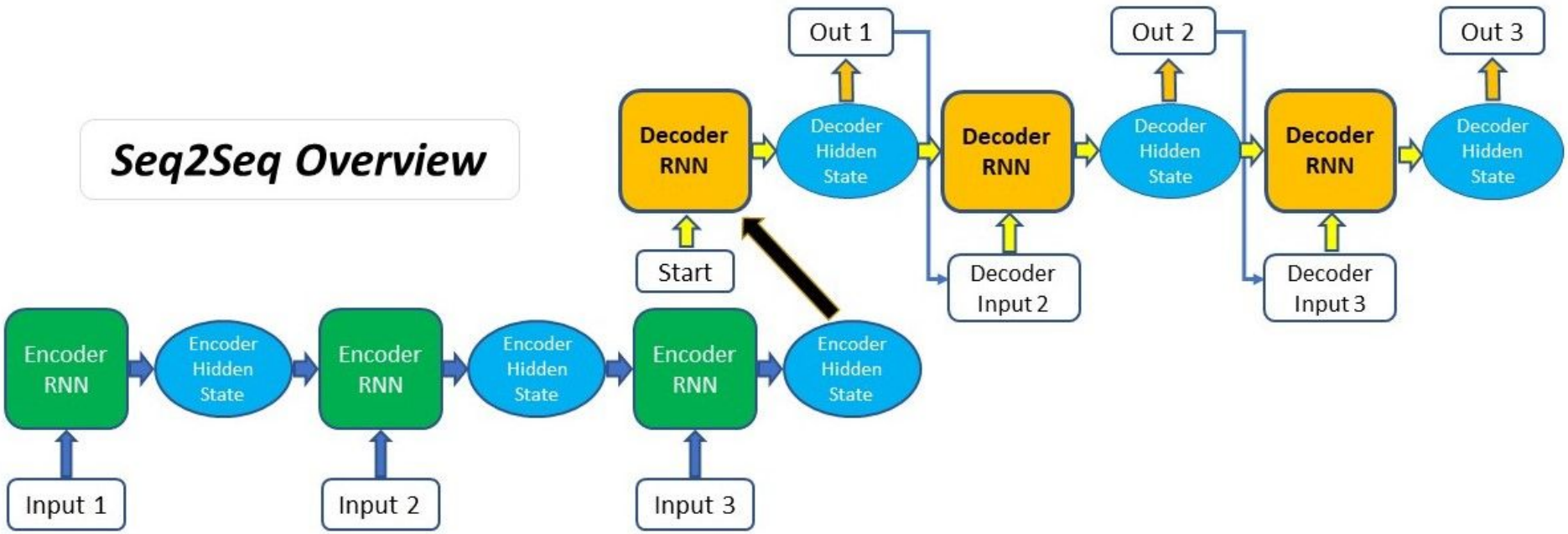


Figure: Source: https://blog.floydhub.com/attention-mechanism/

Attention Mechanism - Applications, Demonstration

- **Applications:** Attention mechanisms are widely used in machine translation, image captioning, and natural language processing tasks where sequential or spatial relationships are important.
- **Transformers:** Core part of general purpose transformers (GPT).

Temporal Convolutional Networks (TCNs)

- TCNs are a class of neural networks designed for modeling sequential data, where the order of the input elements matters.
- TCNs utilize dilated convolutions, which have exponentially growing receptive fields, enabling them to capture long-term dependencies in the input sequences.

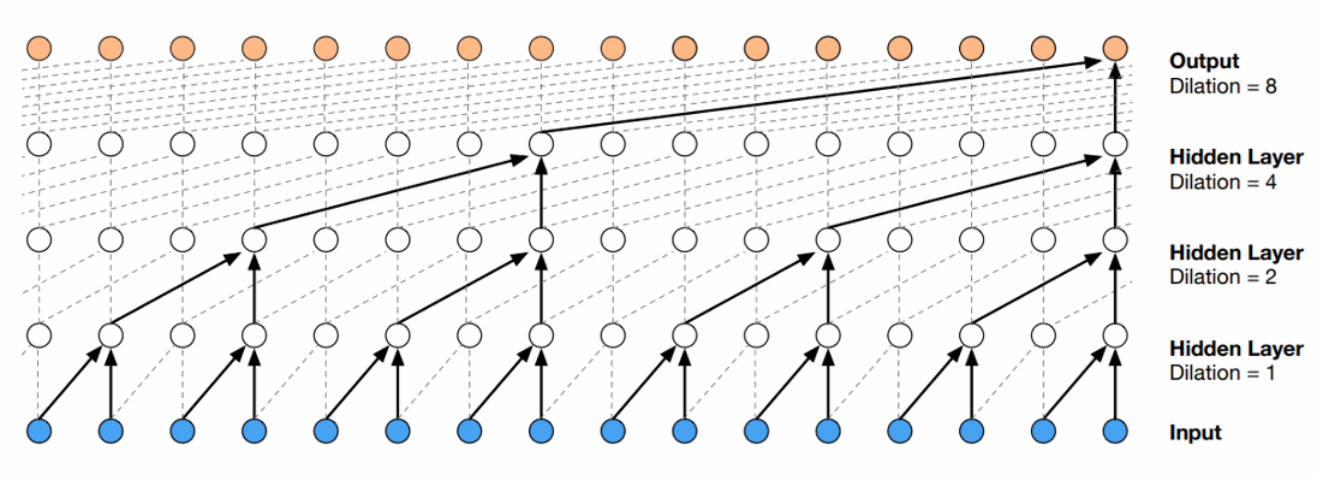


Figure: Source: WaveNet: A Generative Model for Raw Audio

Temporal Convolutional Networks (TCNs) - Applications, Demonstration

- By stacking multiple layers of dilated convolutions, TCNs can effectively model complex temporal patterns while maintaining parallelism and computational efficiency.
- TCNs have been successfully applied in various tasks, including language modeling, time series prediction, and natural language processing, showcasing their versatility and effectiveness in capturing sequential dependencies.

Discovering AI

- Deep Learning with Python, François Chollet.
- Coursera - Deep Learning Specialization by Andrew Ng from Stanford University.
- Use Google Colab to write code that you can test without having to install anything on your computer.
- Ask ChatGPT for help.

- Thank you for your attention.
- Questions and/or discussion?