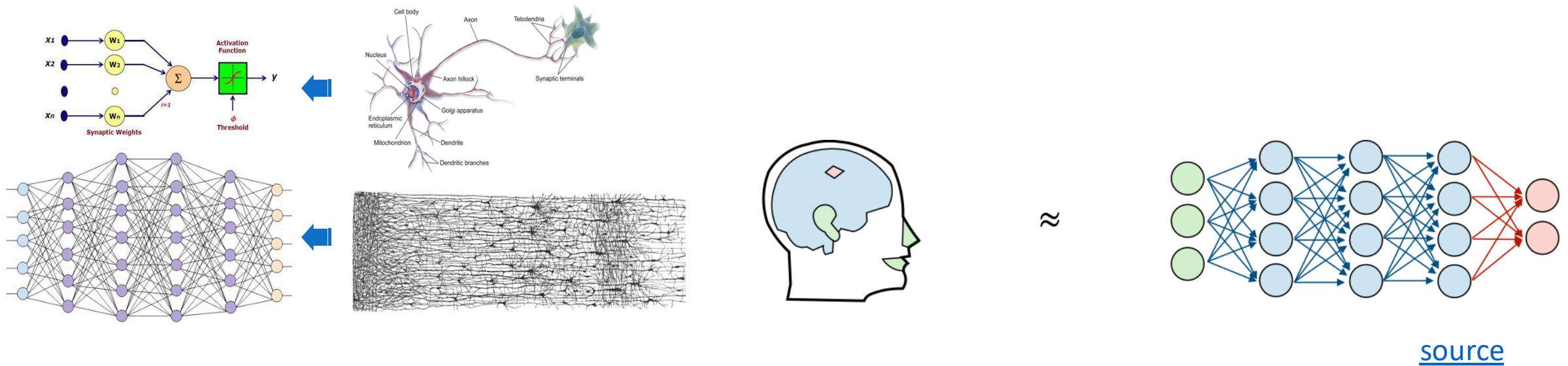


Neural Networks

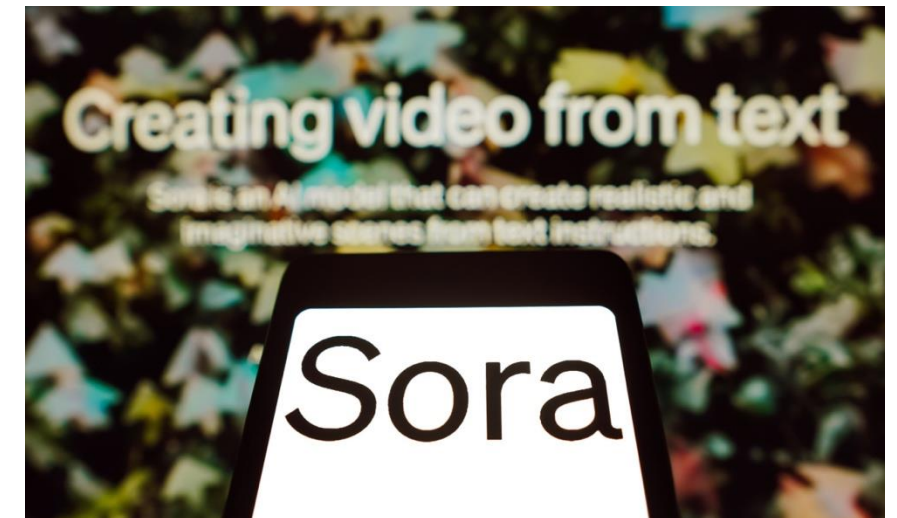
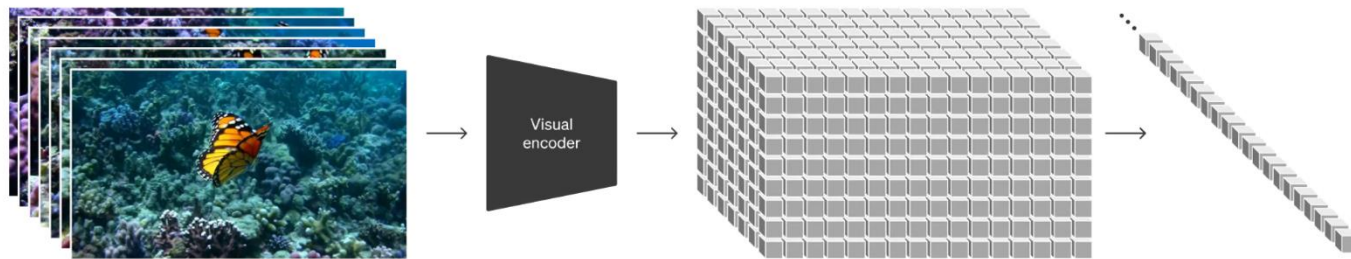
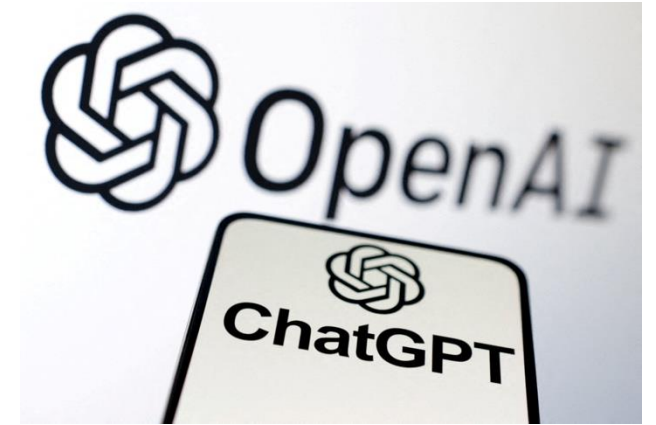
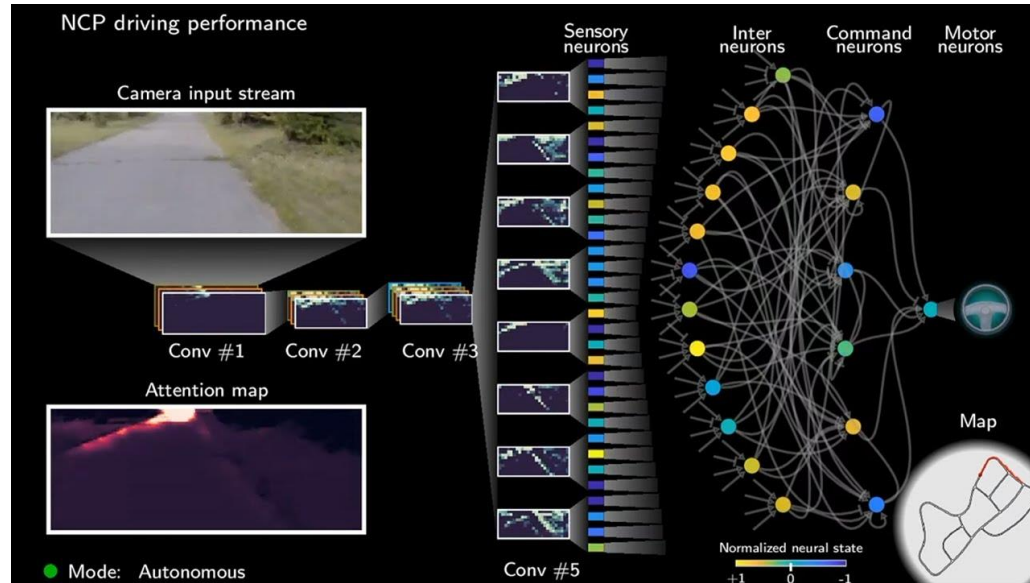
Mehdi Ataei

Neural Networks

- Inspired by how the brain works
- Neurons are connected after each other, like neurons in the brain

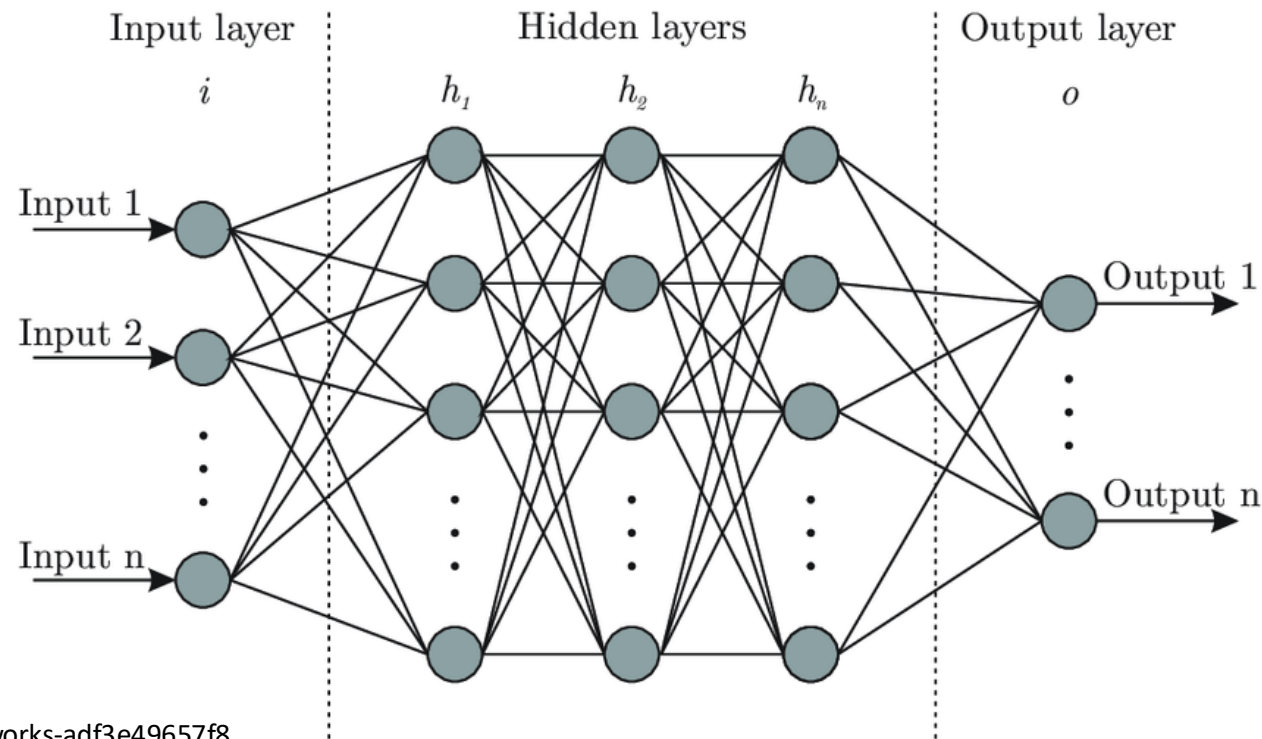


Neural Networks are Powerful AI Models

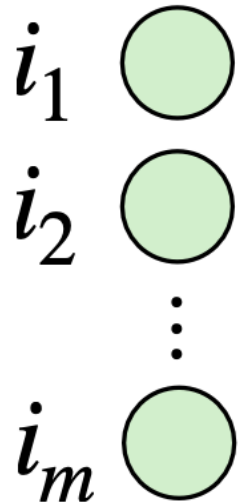


Neural Networks

- A neural network consists of an input layer, one or more of hidden layers, and an output layer



Components: Input

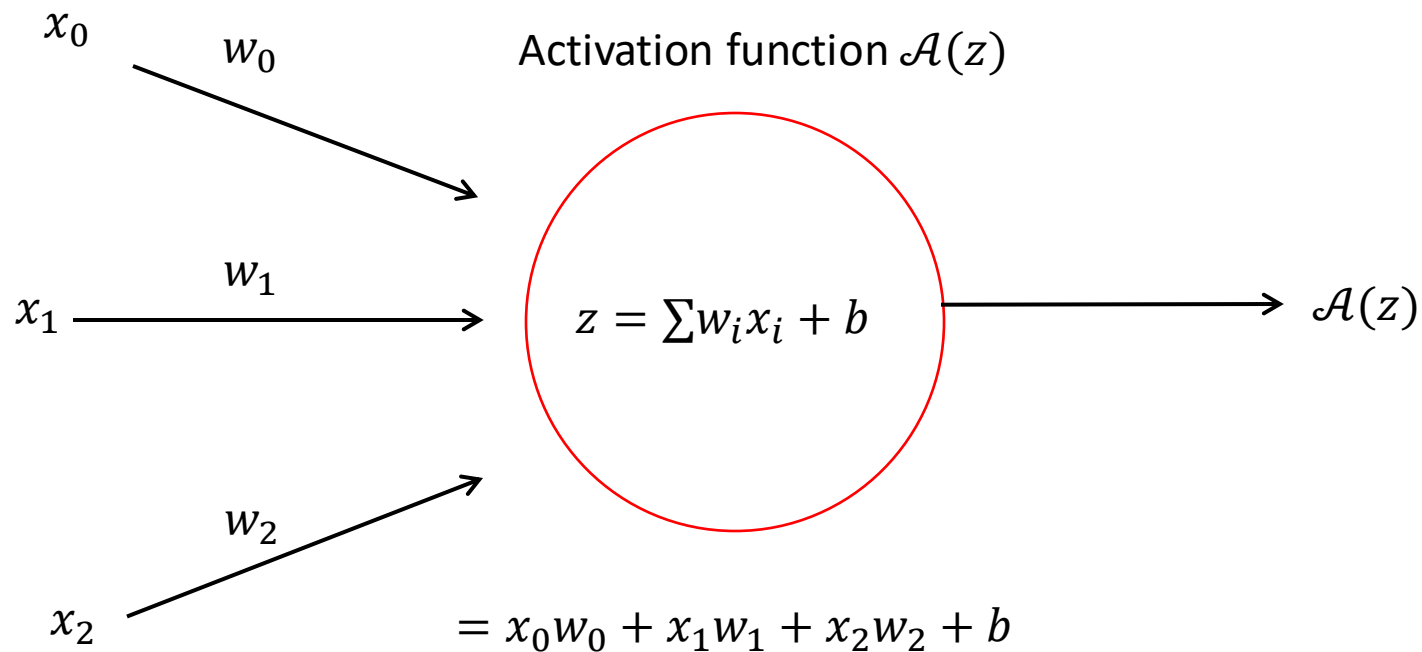


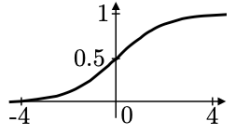
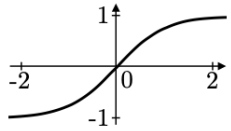
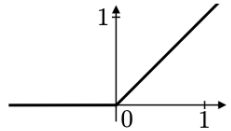
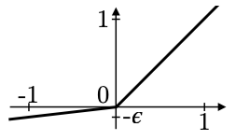
Modality	Description	Illustration
Text	The input text can be divided into entities that are each represented by a vector. <i>Example:</i> one-hot encoding	<p>The illustration shows the text 'a cute teddy bear' split into three segments: 'a' (dashed box), 'cute' (solid box), and 'teddy bear' (dashed box). Arrows from each segment point to a corresponding light green circle, representing a vector.</p>
Audio	The input audio for speech recognition can be divided into small time segments, each represented by features extracted from the sound wave. <i>Example:</i> Mel-spectrograms	<p>The illustration shows a sound wave divided into three segments by dashed vertical lines. The third segment is highlighted with a solid black box. An arrow points from this segment to a corresponding light green circle, representing a vector.</p>
Image	Each pixel of the image can be represented by a vector. <i>Example:</i> RGB encoding	<p>The illustration shows a pixelated image of a brown teddy bear on a white background. A small square region of the image is highlighted with a solid black box. An arrow points from this region to a corresponding light green circle, representing a vector.</p>

Components: Hidden Layer

- A hidden layer (hl) enables the network to capture and represent patterns within the data. It consists of the following components:
 - Weights (w): These are model-learned parameters that are applied to the input through multiplication
 - Biases (b): These are also learned parameters added to the result of the weight multiplication
- These are the **learnable** parameters of the network by gradient descent (recall linear regression lecture)

Hidden Layer and Activation Function



Name	Function $\mathcal{A}(z)$	Output range	Illustration
Sigmoid σ <i>Sigmoid function</i>	$\frac{1}{1 + e^{-z}}$	$]0, 1[$	
Tanh <i>Hyperbolic Tangent</i>	$\frac{e^z - e^{-z}}{e^z + e^{-z}}$	$] - 1, 1[$	
ReLU <i>Rectified Linear Unit</i>	$\max(0, z)$	$[0, +\infty[$	
Leaky ReLU <i>Leaky Rectified Linear Unit</i>	$\max(\epsilon z, z)$ with $\epsilon \ll 1$	$] - \infty, +\infty[$	

[source](#)

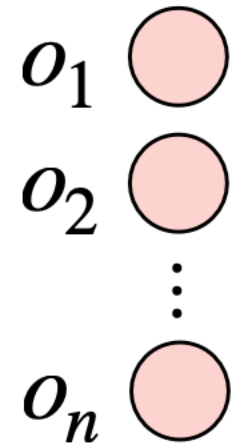
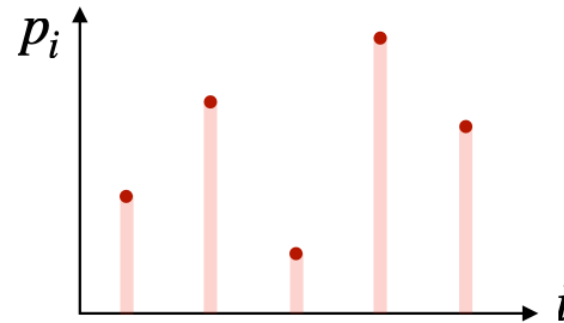
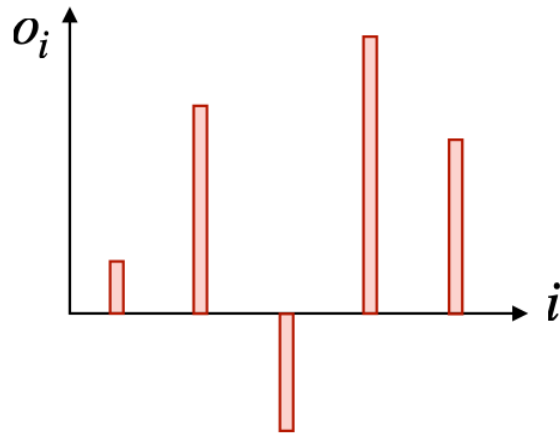
Example

- Let's consider a neuron with two input values (x_1 and x_2) and two corresponding weights (w_1 and w_2).
- The output of the neuron is calculated as follows:
- **$\text{output} = \text{activation_function}(w_1 * x_1 + w_2 * x_2 + \text{bias})$**
- For simplicity, let's assume that the activation function is the ReLU function.
- Suppose the values of the input and weights are:
- **$x_1 = 0.5, x_2 = -0.8, w_1 = 0.3, w_2 = 0.7, \text{bias} = -0.2$**
- Then the output of the neuron can be calculated as:
- **$\text{output} = \text{ReLU}(0.3 * 0.5 + 0.7 * (-0.8) - 0.2) = \text{ReLU}(-0.61) = 0.0$**
- The output value can be interpreted as the neuron's activation level, representing the degree to which it has been activated by the input values and weights.

Output Layer: Classification

- **Classification:** A *softmax layer* converts the result into a probability distribution

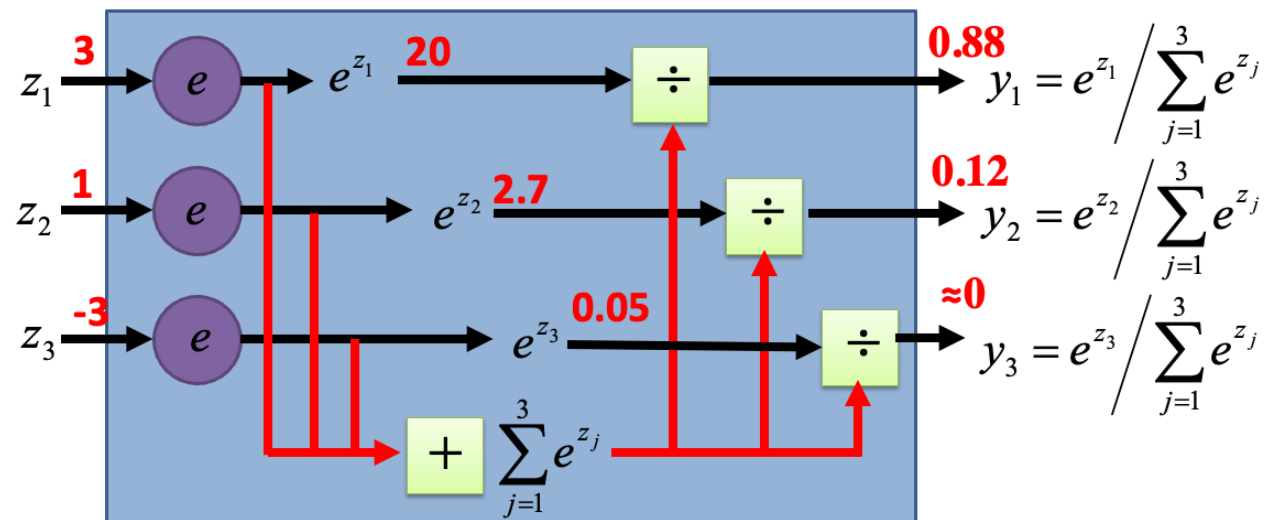
$$p_i = \text{softmax}(o)_i = \frac{e^{o_i}}{\sum_{j=1}^n e^{o_j}}$$



[source](#)

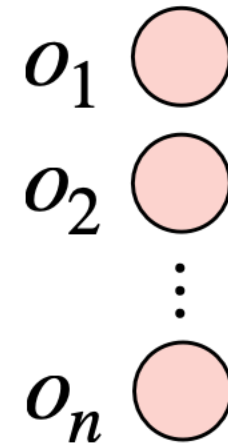
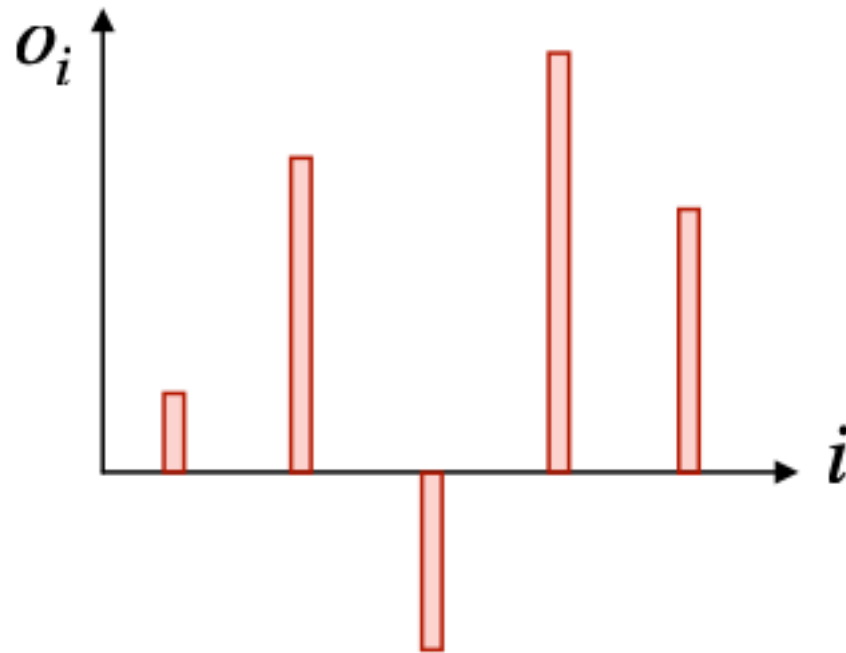
Softmax Function

- The **softmax** activations are added to output a probability value in the range $[0, 1]$
- What don't we use sigmoid for output? except for binary classification (two classes) the probabilities may not add up to 1.0



Output Layer: Regression

- In regression, O is returned as is (for having positive or negative outputs), or depending on the output expected, one of the activation functions are used (see the next slide)



Regression (examples)

Example	Output activation	Why?
Predicting temperature (in Celsius)	Linear activation	Temperature can be both positive and negative
Predicting temperature (in Kelvin)	ReLU	Temperature can only be positive
Normalized data between [0,1]	Sigmoid	Any output above 1 and below 0 is meaningless

Learning Process of a Neural Network

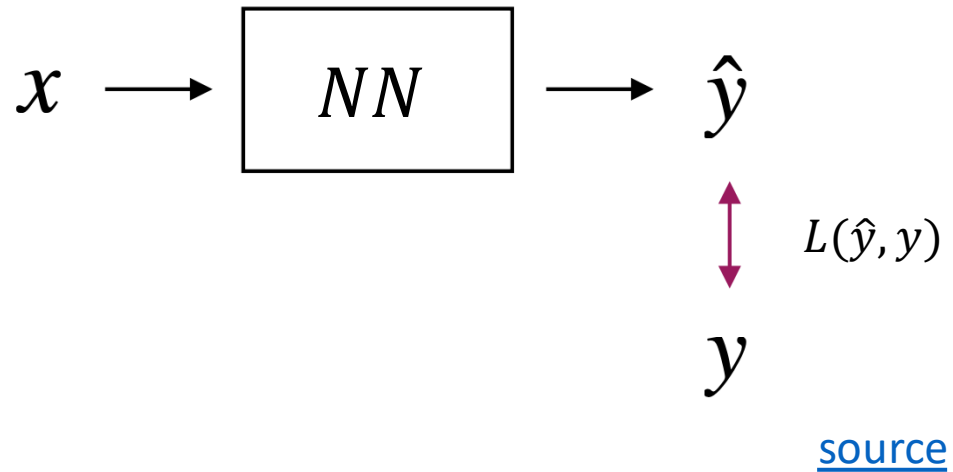
- The goal of training a neural network is to **find** a set of **weights** and **biases** for all the neurons that correspond to the desired output
- A neural network learns from data by adjusting its weights and biases to minimize a **loss function**, which measures how well the network's predictions match the actual outcomes

Training Neural Networks

- We define the following parameters:
- **Training size (N):** The total number of observations included in the training dataset.
- **Batch size (b):** The number of observations the model processes in each iteration.
- **Steps per epoch (s):** The number of iterations required for the model to go through the entire training dataset once.
- These parameters are related to each other with $N = b \times s$

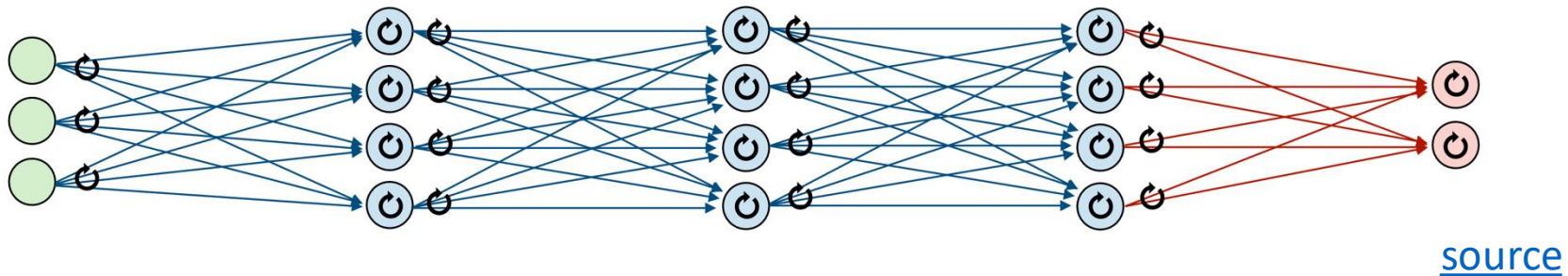
Loss Function

- The loss function $L(\hat{y}, y)$ provides us with a metric on how close the prediction of the model \hat{y} is compared to our target values y (e.g., MSE, MAE)



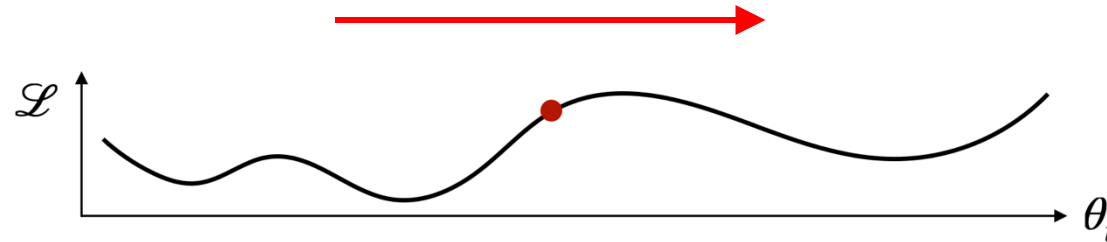
Training Neural Networks

- Backpropagation is a training method used to reduce the loss function L by adjusting the model parameters θ through the computation of gradients $\nabla L(\theta)$.
- The concept is to update the neural network's parameters θ in a way that ensures its predicted output \hat{y} closely aligns with the target label y .

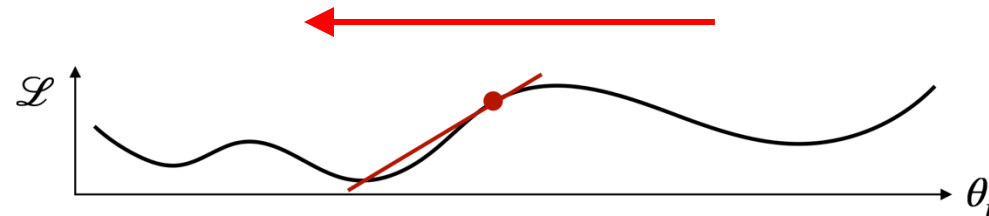


Training Steps

1. Forward step: Take a batch of training data, perform forward propagation to generate the model's predictions, and calculate the loss $L(\hat{y}, y)$ based on the predicted and actual values



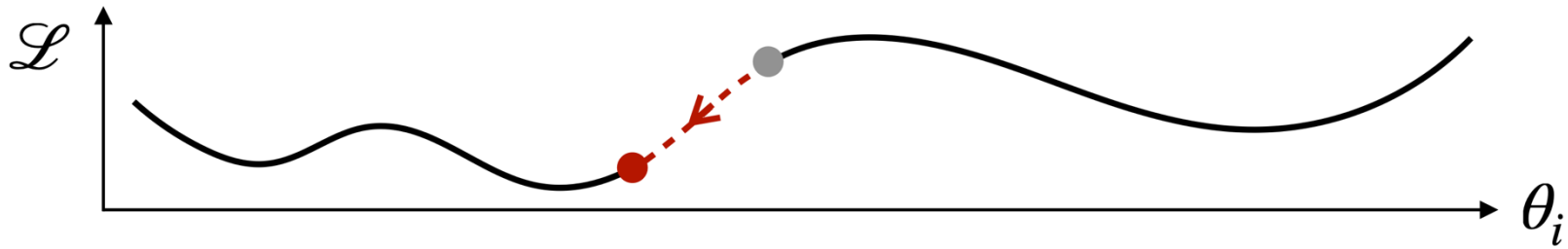
2. Backward step: Compute the gradient of the loss $\frac{\partial L}{\partial \theta_i}$ with respect to each network parameter θ_i



[source](#)

Training Steps

3. Update step: Use the computed gradients to update each parameter θ of the network towards a direction that best decreases the loss L

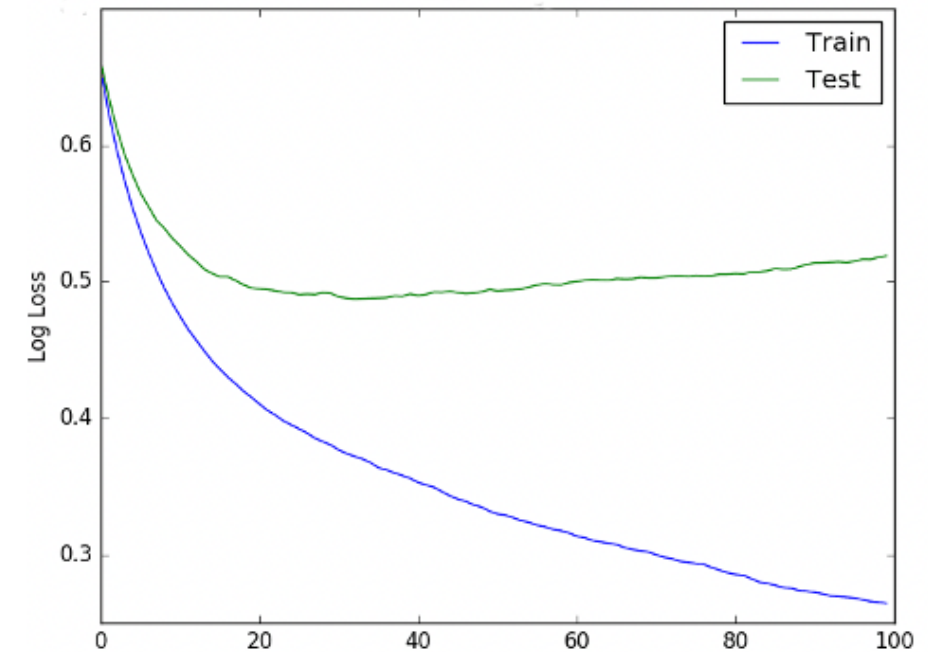


4. Repeat!

[source](#)

Testing a Neural Network Model

- After training, the performance of the model must be tested against the **test dataset**
- A comparable performance is expected on the test dataset
- If the performance of the model is worse on the test dataset, it corresponds to **overfitting**
- **One solution:** Reduce network trainable parameters
 - Reduce number of neurons in each hidden layer
 - Reduce the number of hidden layers




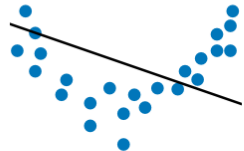
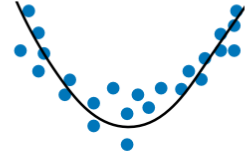
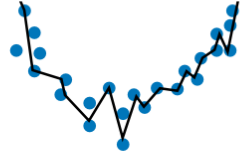



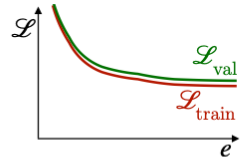
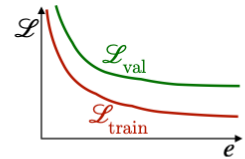
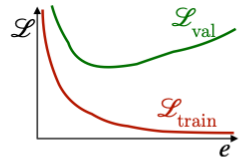


Overfitting and Underfitting

- **Overfitting** happens when a model learns the training data too well, including noise and details that don't generalize to new, unseen data. As a result, the model performs very well on the training set but poorly on new data because it is too specific to the training examples
- **Underfitting** occurs when a model is too simple and doesn't capture the underlying patterns in the data. It performs poorly on both the training set and new data because it hasn't learned enough from the information available

In short:

- **Overfitting** = too specific, good on training data but bad on new data.
- **Underfitting** = too simple, bad on both training and new data.

	Underfitting	Just right	Overfitting
Symptoms	<ul style="list-style-type: none"> • High training error • Training error close to test error • High bias 	Training error slightly lower than test error	<ul style="list-style-type: none"> • Low training error • Training error much lower than test error • High variance
Intuition			
Regression			
Classification			
Evolution of the loss			
Possible remedies	<ul style="list-style-type: none"> • Complexify model • Add more features • Train longer 		<ul style="list-style-type: none"> • Regularize • Get more data

Advanced Neural Network Architectures

So far we have covered the simplest neural network architecture known as Multi-Layer Perceptron (MLP) or fully-connected neural network

There are more advanced architectures tailored towards different data types

- **Convolutional Neural Networks (CNNs)** are particularly effective for image data. They are designed to automatically learn spatial features from images, and have been used for tasks like object detection and image classification.
- **Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM), and Transformers** are often used for sequential data, such as text or time series data. These architectures are designed to learn dependencies and patterns over time, and have been used for tasks like language translation, sentiment analysis, and speech recognition.
- GANS and Diffusion Models are used for tasks such as image generation

Mandatory watch!

https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi

