Neural Networks: Structure, Learning and Biological Intuition

UNIVERSITY OF OXFORD
Mathematical Institute

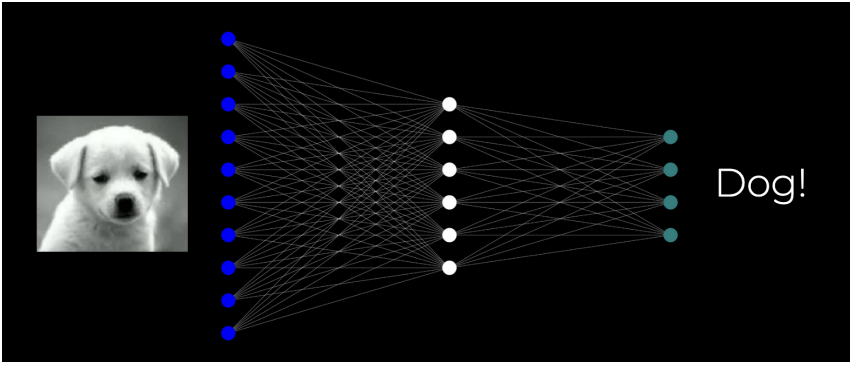
Lida Kanari

Mathematical Institute
University of Oxford

Introduction to Machine Learning, November 2025

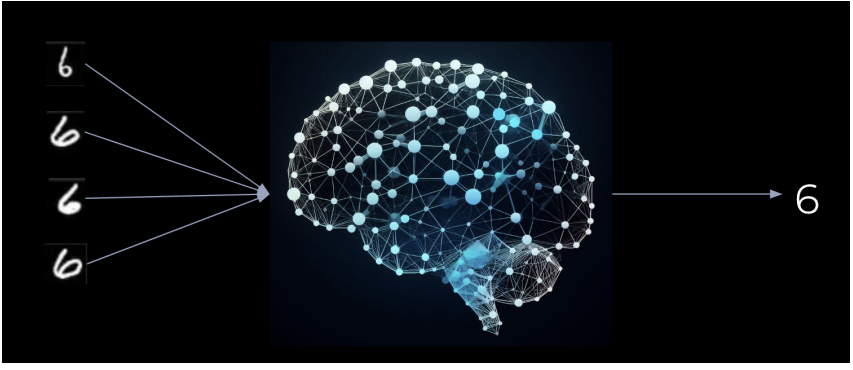
Oxford Mathematics

Table of Contents

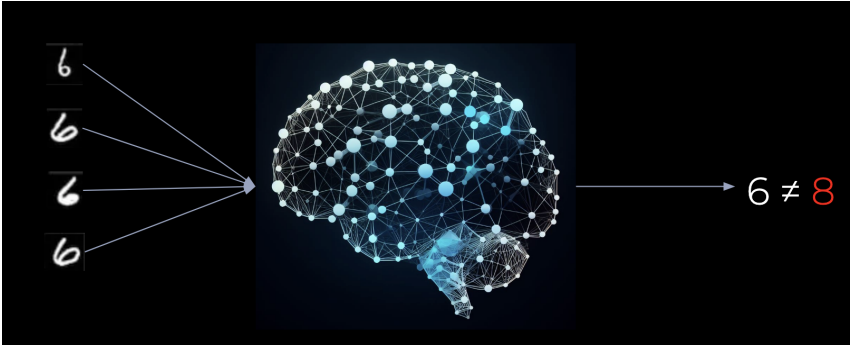


Objective of a neural network architecture.

Introduction

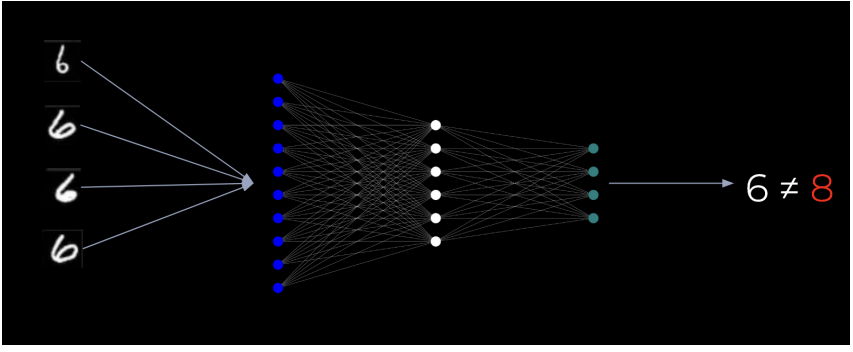


For our brain this task is obvious.



For our brain this task is obvious.

Introduction



Can we reproduce this with an NN.

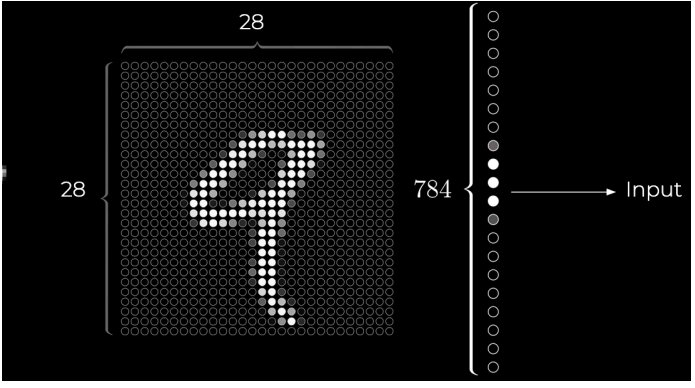
Dataset: MNIST Digits

- Images: handwritten digits $0, \ldots, 9$ with **labels**.
- Each image: 28×28 pixels $\Rightarrow$ 784 features if flattened.
- Grayscale values (e.g. in $[0, 1]$) representing intensity.
- Training set: 30000 digits
- Test set: 10000 digits

MNIST Gallery

MNIST: dataset of handwritten digits.

MNIST Gallery



MNIST: dataset of handwritten digits.

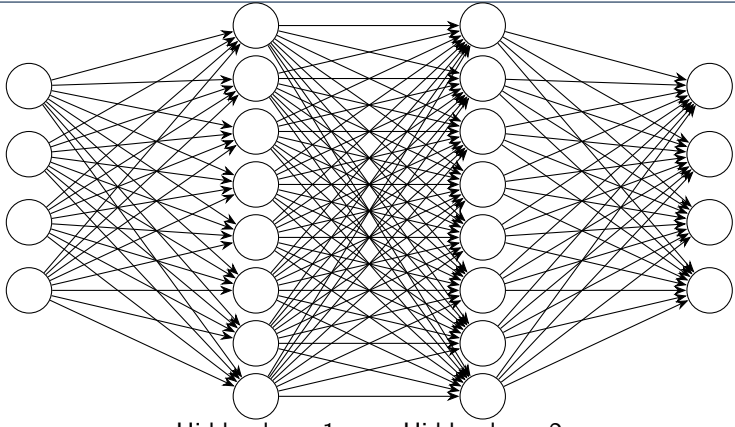
What is a Neural Network?

Multilayer perceptron is the simplest form of a neural network, that consist of multiple layers.
Each layer holds information about the input data and consists of multiple neurons.
Neurons hold one value, for example a value between $(0, 1)$.
The **perceptron** has connections between all neurons in a layer to all neurons in the next layer. The information that are processed depend on the strength of the connections.

What is a Neural Network?

Definition: A feedforward neural network is a parametrized composition of affine maps and nonlinearities:
Parameters: $\theta = \{W^{(l)}, b^{(l)}\}_{l=1}^{L}$.
Goal: learn the θ so that the network approximates the mapping $\mathcal{X} \mapsto \dagger$.

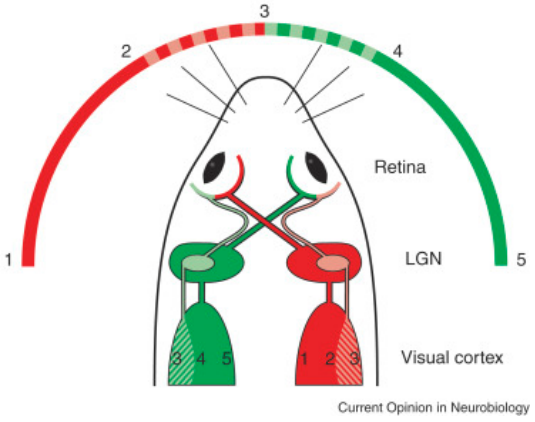
Neural Network



Neural Networks Inspired by the Brain

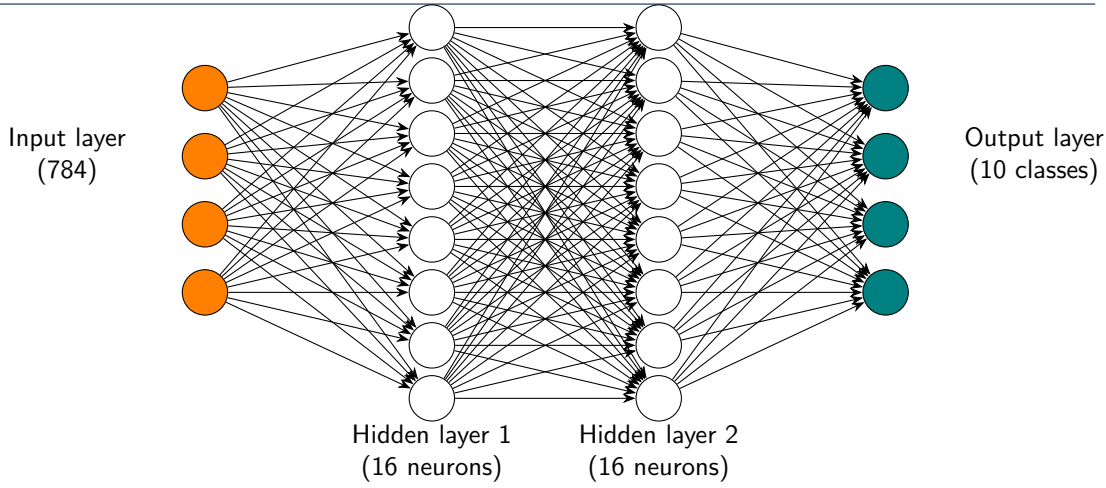
- Artificial neurons mimic (loosely) biological neurons: integrate inputs, apply nonlinearity, send outputs.
- Biological learning: synaptic plasticity.
- Artificial learning: adjust weights via gradients from a loss function.
- Not a literal model of the brain, but a functional inspiration.

Neural Network inspiration

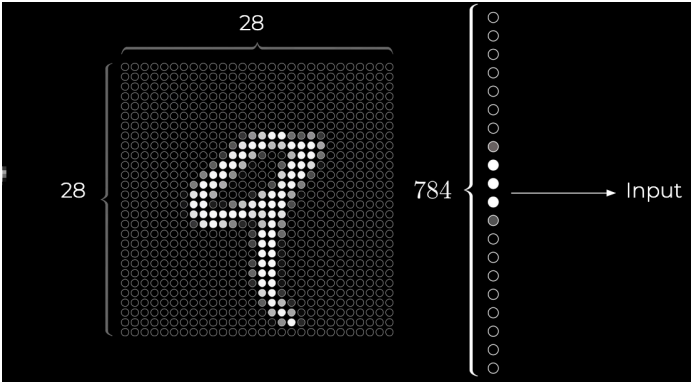


Processing images in the rodent brain (Hübener et al. 2003)

Neural Network: Input - Output

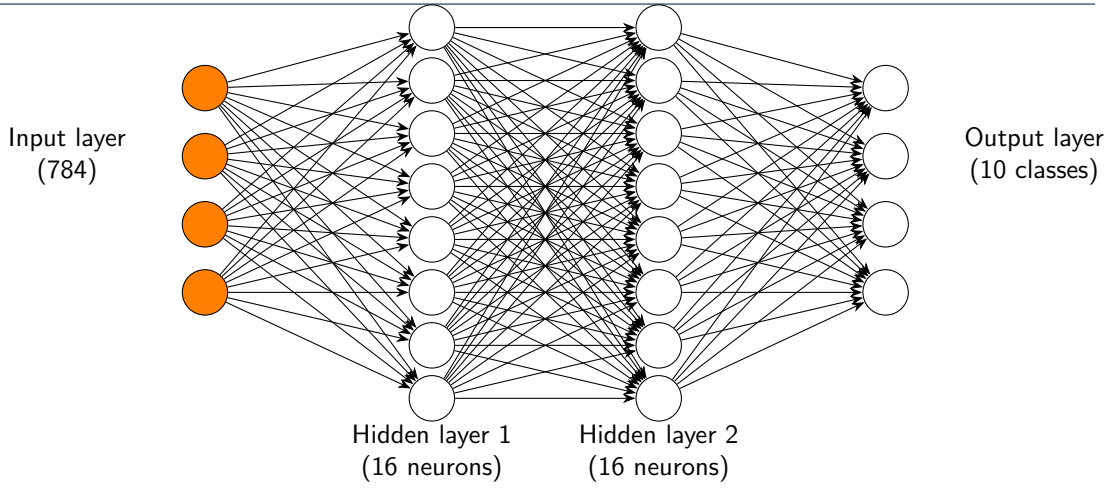


Input: MNIST data



MNIST: dataset of handwritten digits.

Neural Network: Input - Output



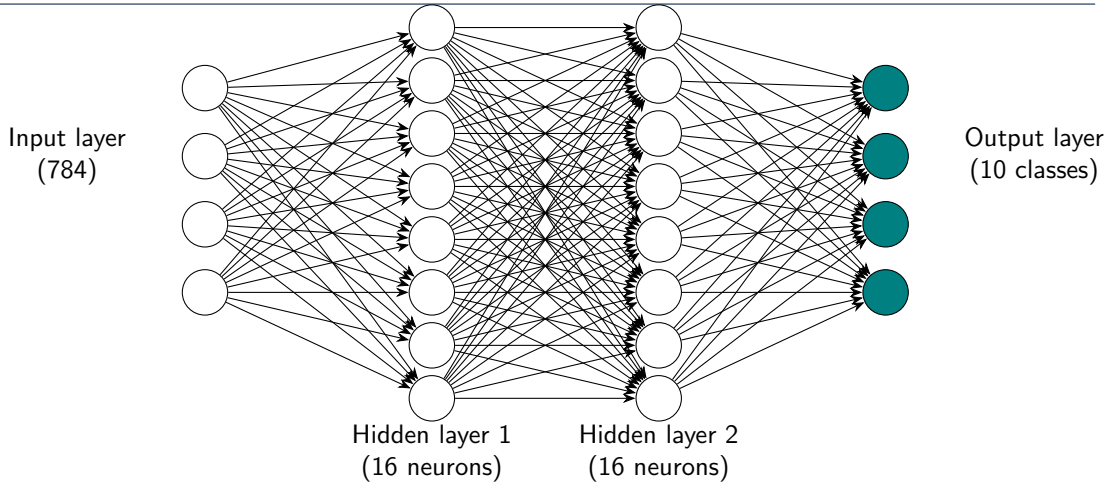
Output: digit labels

Output layer: 10 nodes, one per digit class (0–9).
Network raw output (logits): $z^{(L)} \in \mathbb{R}^{10}$.
Conversion to probabilities via softmax:

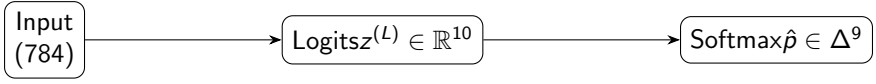
$$\hat{p}_k = \frac{e^{z_k}}{\sum_{j=0}^{9} e^{z_j}}, \quad k = 0, \ldots, 9.$$

Prediction: $\hat{y} = \arg\max_k \hat{p}_k$.

Neural Network: Input - Output

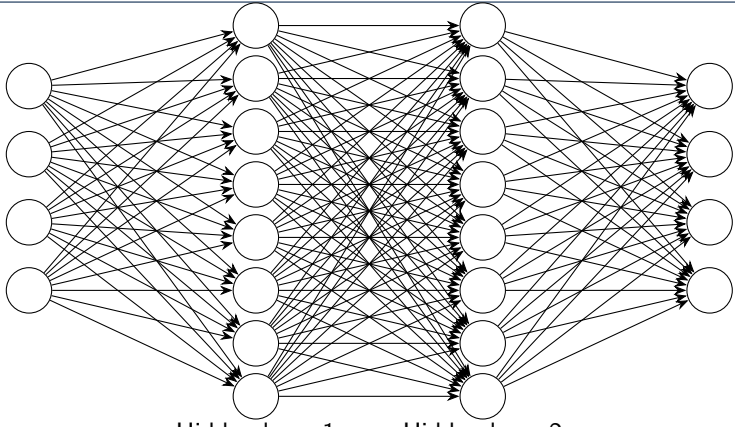


Output Activations



Predicted class $= \arg\max \hat{p}_k$

Neural Network: 2-layer architecture



Example architecture

Input $\to$ NN (layer-1) $\to$ NN (layer-2) $\to$ Output

$$x \in \mathbb{R}^{784} \xrightarrow{W^{(1)},b^{(1)}} a^{(1)} \in \mathbb{R}^{n_1} \xrightarrow{W^{(2)},b^{(2)}} a^{(2)} \in \mathbb{R}^{n_2} \to y \in \mathbb{R}^{10}$$

Two hidden layers with 16 neurons

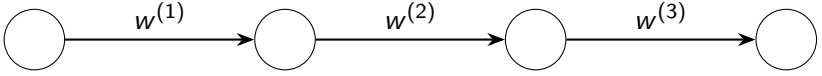
Concrete sizes:

$$n_0 = 784, \quad n_1 = 16, \quad n_2 = 16, \quad n_3 = 10.$$

Number of parameters (approx):

$$\text{params} \approx 784 \cdot 16 + 16 + 16 \cdot 16 + 16 + 16 \cdot 10 + 10 \approx 13{,}000.$$

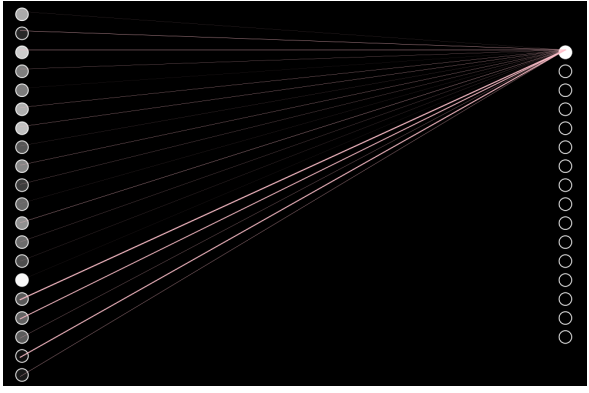
Schematic of Neural Network



For the MNIST example: **784** $\rightarrow$ **16** $\rightarrow$ **16** $\rightarrow$ **10** (13 k parameters).

Neural Networks: notation and linear algebra

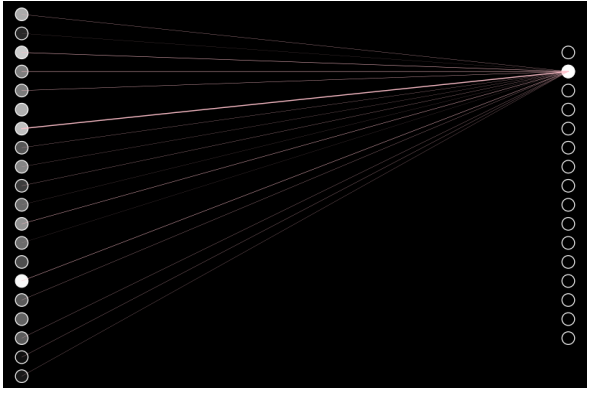
$$a_0^{(1)} = \sigma(w_{00}\, a_0^{(0)} + w_{01}\, a_1^{(0)} + w_{02}\, a_2^{(0)} + \ldots + w_{0n}\, a_n^{(0)} + b_0)$$



The activation of the first neuron $a_0^{(1)}$ is a combination of the activations of the input layer $a_i^{(0)}$, multiplied by the respective weights.

Neural Networks: notation and linear algebra

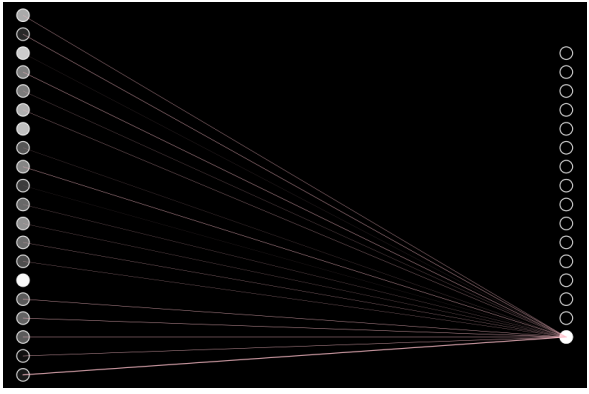
$$a_1^{(1)} = \sigma(w_{10}\, a_0^{(0)} + w_{11}\, a_1^{(0)} + w_{12}\, a_2^{(0)} + \ldots + w_{1n}\, a_n^{(0)} + b_1)$$



The activation of the second neuron $a_1^{(1)}$ is a combination of the activations of the input layer $a_i^{(0)}$, multiplied by the respective weights.

Neural Networks: Notation and linear algebra

$$a_n^{(1)} = \sigma(w_{n0}\, a_0^{(0)} + w_{n1}\, a_1^{(0)} + w_{n2}\, a_2^{(0)} + \ldots + w_{nn}\, a_n^{(0)} + b_n)$$



The activation of the last neuron $a_n^{(1)}$ is a combination of the activations of the input layer $a_i^{(0)}$, multiplied by the respective weights.

Neural Networks: notation and linear algebra

$$
\begin{aligned}
a_0^{(1)} &= \sigma(w_{00}\, a_0^{(0)} + w_{01}\, a_1^{(0)} + w_{02}\, a_2^{(0)} + \ldots + w_{0n}\, a_n^{(0)} + b_0) \\
a_1^{(1)} &= \sigma(w_{10}\, a_0^{(0)} + w_{11}\, a_1^{(0)} + w_{12}\, a_2^{(0)} + \ldots + w_{1n}\, a_n^{(0)} + b_1) \\
a_2^{(1)} &= \sigma(w_{20}\, a_0^{(0)} + w_{21}\, a_1^{(0)} + w_{22}\, a_2^{(0)} + \ldots + w_{2n}\, a_n^{(0)} + b_2) \\
&\ldots \\
a_n^{(1)} &= \sigma(w_{n0}\, a_0^{(0)} + w_{n1}\, a_1^{(0)} + w_{n2}\, a_2^{(0)} + \ldots + w_{nn}\, a_n^{(0)} + b_n)
\end{aligned}
$$

Putting all the equations together.

$$\begin{bmatrix} a_0^{(1)} \\ a_1^{(1)} \\ a_2^{(1)} \\ \dots \\ a_n^{(1)} \end{bmatrix} = \begin{bmatrix} w_{00} & w_{01} & w_{00} & \dots & w_{0n} \\ w_{10} & w_{11} & w_{10} & \dots & w_{1n} \\ w_{20} & w_{21} & w_{20} & \dots & w_{2n} \\ & & & \dots & \\ w_{n0} & w_{n1} & w_{n0} & \dots & w_{nn} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ a_2^{(0)} \\ \dots \\ a_n^{(0)} \end{bmatrix}$$

Using matrices we have from all the equations.

$$\begin{bmatrix} a_0^{(1)} \\ a_1^{(1)} \\ a_2^{(1)} \\ \dots \\ a_n^{(1)} \end{bmatrix} = \begin{bmatrix} w_{00} & w_{01} & w_{00} & \dots & w_{0n} \\ w_{10} & w_{11} & w_{10} & \dots & w_{1n} \\ w_{20} & w_{21} & w_{20} & \dots & w_{2n} \\ & & & \dots & \\ w_{n0} & w_{n1} & w_{n0} & \dots & w_{nn} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ a_2^{(0)} \\ \dots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \dots \\ b_n \end{bmatrix}$$

Using matrices we have from all the equations.

Two important points!!!

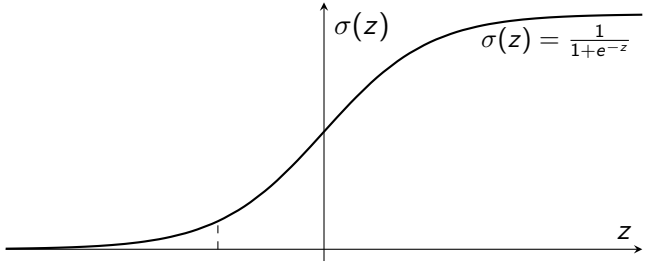
- The sigmoid function σ
- The significance of the bias b_i

Sigmoid Activation

The sigmoid function is defined as:

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

Sigmoid Activation



Sigmoid Activation

The sigmoid function is defined as:

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

The sigmoid function maps the real numbers to the $(0, 1)$ interval. Therefore it is important to control the activity of each neuron so that the output is always between $(0, 1)$.

Neural Network Bias

The bias is important to control when a neuron is active.

$$a_n^{(1)} = \sigma(w_{n0}\, a_0^{(0)} + w_{n1}\, a_1^{(0)} + w_{n2}\, a_2^{(0)} + \ldots + w_{nn}\, a_n^{(0)} + b_n)$$

Neural Network Bias

The bias is important to control when a neuron is active.

$$a_n^{(1)} = \sigma(w_{n0}\, a_0^{(0)} + w_{n1}\, a_1^{(0)} + w_{n2}\, a_2^{(0)} + \ldots + w_{nn}\, a_n^{(0)} + b_n)$$

For a bias $= 0$ the neuron is only active for a positive weighted sum.

Neural Network Bias

The bias is important to control when a neuron is active.

$$a_n^{(1)} = \sigma(w_{n0}\, a_0^{(0)} + w_{n1}\, a_1^{(0)} + w_{n2}\, a_2^{(0)} + \ldots + w_{nn}\, a_n^{(0)} + b_n)$$

For a bias $= 0$ the neuron is only active for a positive weighted sum.
But maybe we need the neuron to be active above a certain value.
The bias is added to control when the neuron is active

Bias effect on Sigmoid Activation

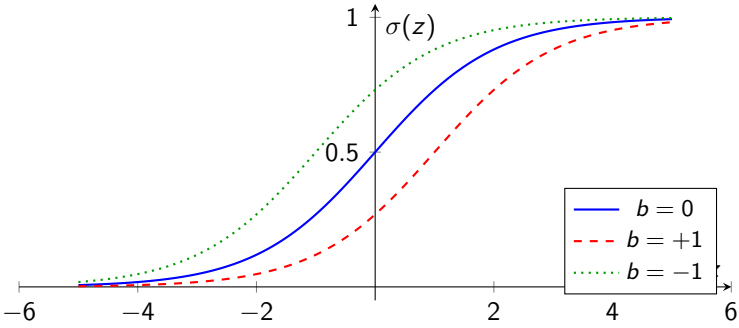
Bias b translates the pre-activation:

$$z = w^\top x + b.$$

Changing b shifts the sigmoid horizontally:

- a positive b makes neuron activate more easily
- a negative b requires larger input for activation

Sigmoid with Different Biases



Increasing the bias b shifts the sigmoid curve to the left (neuron activates more easily); decreasing b shifts it to the right (requires stronger input).

Activation and Weighted Sum

For neuron j in layer l:

$$z_j^{(l)} = \sum_i w_{ji}^{(l)} a_i^{(l-1)} + b_j^{(l)}.$$

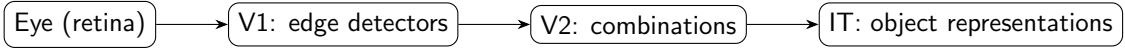
Activation:

$$a_j^{(l)} = \sigma\left(z_j^{(l)}\right).$$

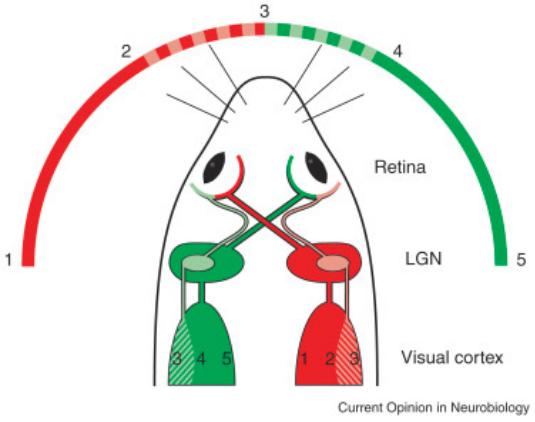
Why we need layers

- Layers capture hierarchical structure: from simple features to complex features.
- In vision: low-level layers detect edges; higher layers detect curves, loops; final layers detect digits.
- Layers enable compositional representation.

Why we need layers



Why we need layers



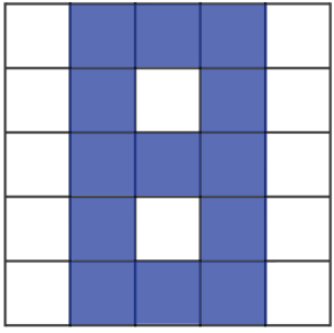
Processing images in the rodent brain (Hübener et al. 2003)

Layer Functionality

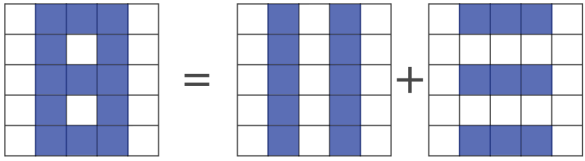
Hidden layers can respond to particular features: loop detection, line detection etc.

- Digit '8': two loops or a combination of line features.
- Digit '6': loop + vertical stroke + horizontal line.

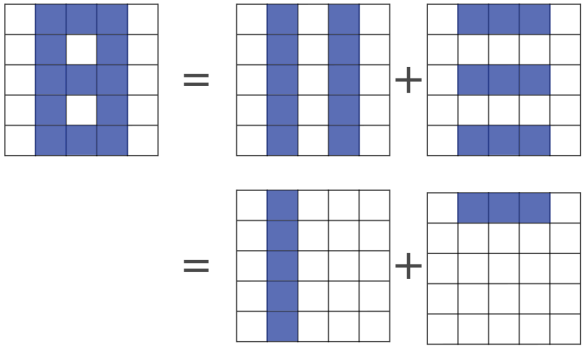
Layer Functionality



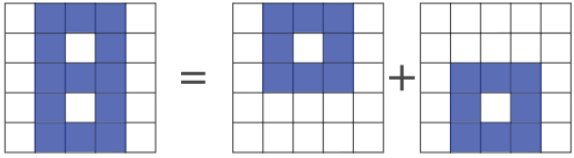
Layer Functionality: line decomposition



Layer Functionality: line decomposition



Layer Functionality: loop decomposition



Layers of Abstraction

- Vision: pixels $\rightarrow$ edges $\rightarrow$ shapes $\rightarrow$ objects.
- Speech: waveform $\rightarrow$ phonemes $\rightarrow$ words $\rightarrow$ meaning.
- Text: characters $\rightarrow$ tokens $\rightarrow$ phrases $\rightarrow$ semantics.

Weights and Bias

- Each connection has weight $w_{ji}^{(l)}$ from neuron i in layer $l-1$ to neuron j in layer l.
- Bias $b_j^{(l)}$ adjusts threshold of neuron j.

Weights and Bias

- Each connection has weight $w_{ji}^{(l)}$ from neuron i in layer $l-1$ to neuron j in layer l.
- Bias $b_j^{(l)}$ adjusts threshold of neuron j.
- All parameters $\theta = \{W^{(l)}, b^{(l)}\}$ are learned from data.

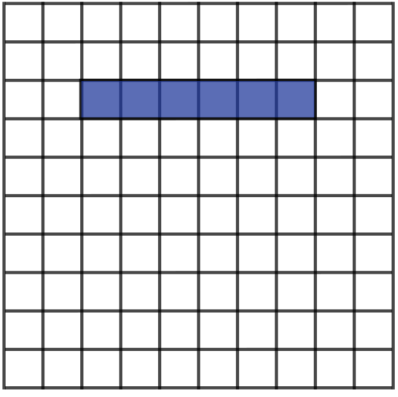
Vertical-Line Detector via Weights

Imagine a weight pattern that emphasizes a column of pixels:

$$w_{ji} \approx \begin{cases} +1 & \text{if pixel } i \text{ is in the central column,} \\ 0 & \text{otherwise.} \end{cases}$$

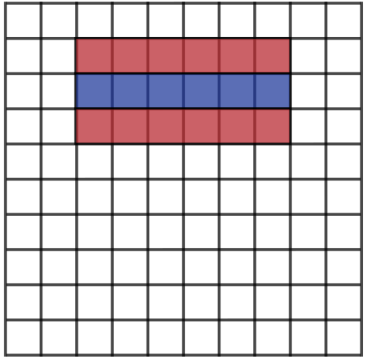
Then z_j will be large when that vertical column has large values in the correct pixels, and $a_j = \sigma(z_j)$ will indicate presence of the vertical line.

Vertical Line Detector



Weights can be chosen for line detection.

Vertical Line Detector



Weights can be chosen for line detection.

Vertical-Line Detector via Weights

Imagine a weight pattern that emphasizes a column of pixels:

$$w_{ji} \approx \begin{cases} +1 & \text{if pixel } i \text{ is in the central column,} \\ 0 & \text{otherwise,} \\ -1 & \text{if pixel } i \text{ above or below the central column.} \end{cases}$$

Then z_j will be large when that vertical column has large values in the correct pixels, negative values in adjacent pixels, and $a_j = \sigma(z_j)$ will identify the presence of the vertical line.

Learning in Neural Networks

How can we use the weights and biases to learn the different patterns?

Learning Cost Function

For a single training example (x, y) and squared error:

$$C = \frac{1}{2}\|\hat{y} - y\|^2 = \frac{1}{2}\sum_k (\hat{y}_k - y_k)^2.$$

Where $\hat{y}$ is network output.

Cost over the Dataset (MSE)

MSE over dataset of size N:

$$C_{\text{MSE}} = \frac{1}{2N} \sum_{n=1}^{N} \|\hat{y}^{(n)} - y^{(n)}\|^2.$$

Gradient Descent

We define a cost (loss) function $C(\theta)$ measuring how well the network predicts. Gradient descent updates parameters θ by moving opposite to the gradient:

$$\theta \leftarrow \theta - \eta \nabla_\theta C(\theta),$$

where $\eta > 0$ is the learning rate.
Small η: slow but stable.
Large η: faster but may diverge.

Gradient Descent

For weight $w_{ji}^{(l)}$:

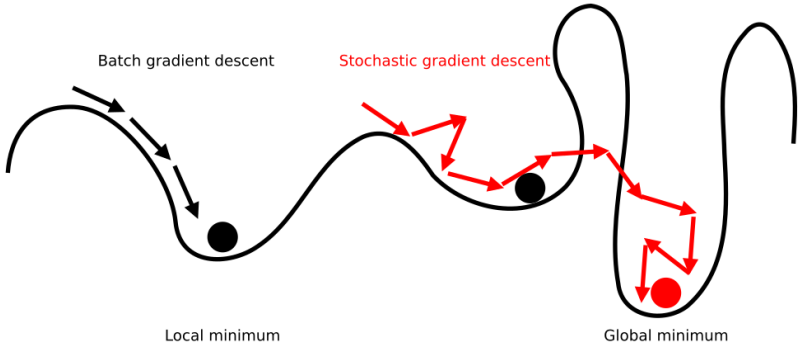
$$w_{ji}^{(l)} \leftarrow w_{ji}^{(l)} - \eta \frac{\partial C}{\partial w_{ji}^{(l)}}.$$

We compute $\frac{\partial C}{\partial w_{ji}^{(l)}}$ via backpropagation.

Convergence: Local vs Global Minima

- Cost surfaces in neural networks are non-convex.
- Gradient descent finds a local minimum or stationary point (or saddle).
- Initialization and learning dynamics influence the final solution.

Convergence: Local vs Global Minima



Parameter Space

All weights and biases form a high-dimensional vector $\theta \in \mathbb{R}^P$ where P is total parameters. Training searches for θ^* minimizing $C(\theta)$:

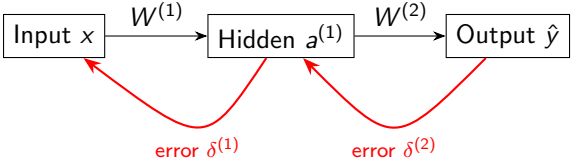
$$\theta^* \in \arg\min_{\theta} C(\theta).$$

Backprop provides $\nabla_\theta C$ efficiently across layers.

Backpropagation: High-Level Idea

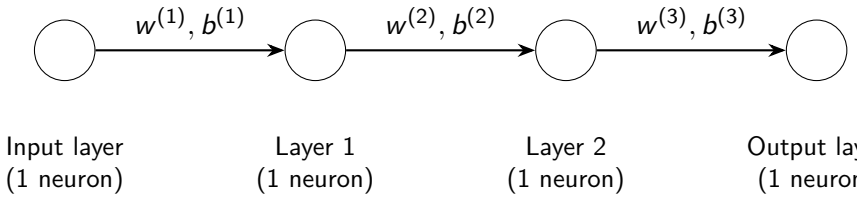
Compute gradients of cost w.r.t. parameters by propagating error from the output layer and back through each layer using the chain rule.

Backpropagation Schematic



Backprop propagates gradients from output to input layers

Backpropagation in one neuron layers



Backpropagation with one neuron layers

We start backwards, from the output layer. The cost is defined as:

$$C = (\hat{y} - y)^2$$

Backpropagation with one neuron layers

We start backwards, from the output layer. The cost is defined as:

$$C = (\hat{y} - y)^2$$

But $\hat{y}$ is the activation of the last layer $a^{(L)}$ which is defined as:

$$a^{(L)} = \sigma(w^{(L)} a^{(L-1)} + b^{(L)})$$

$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

Therefore, the cost is:

$$C = (a^{(L)} - y)^2$$

Where:

$$a^{(L)} = \sigma(z^{(L)})$$

$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

Backpropagation with one neuron layers

Therefore, the cost is:

$$C = (a^{(L)} - y)^2$$

Where:

$$a^{(L)} = \sigma(z^{(L)}), \quad z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

And similarly:

$$a^{(L-1)} = \sigma(z^{(L-1)}), \quad z^{(L-1)} = w^{(L-1)} a^{(L-2)} + b^{(L-1)}$$

Backpropagation with one neuron layers

Therefore, the cost is:

$$C = (a^{(3)} - y)^2$$

Where:

$$a^{(3)} = \sigma(z^{(3)}), \quad z^{(3)} = w^{(3)}a^{(2)} + b^{(3)}$$

And:

$$a^{(2)} = \sigma(z^{(2)}), \quad z^{(2)} = w^{(2)}a^{(1)} + b^{(2)}$$

Backpropagation with one neuron layers

Therefore, the cost is:

$$C = (a^{(3)} - y)^2$$

Where:

$$a^{(3)} = \sigma(z^{(3)}), \quad z^{(3)} = w^{(3)}a^{(2)} + b^{(3)}$$

And:

$$a^{(2)} = \sigma(z^{(2)}), \quad z^{(2)} = w^{(2)}a^{(1)} + b^{(2)}$$

And:

$$a^{(1)} = \sigma(z^{(1)}), \quad z^{(1)} = w^{(1)}a^{(0)} + b^{(1)}$$

$a^{(0)}$ is the input layer.

Backpropagation with one neuron layers

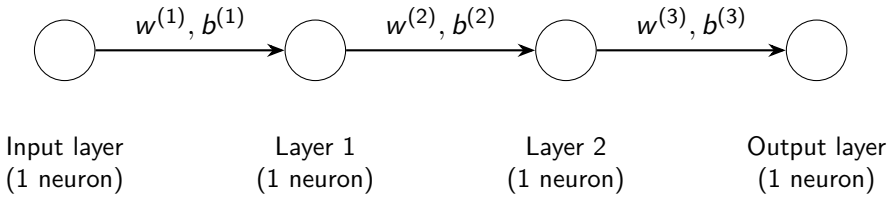
Therefore from the cost

$$C = (a^{(3)} - y)^2$$

We need to learn:

$$\theta = (w^{(3)}, b^{(3)}, w^{(2)}, b^{(2)}, w^{(1)}, b^{(1)})$$

Backpropagation with one neuron layers



Chain Rule

For the cost C:

$$\frac{\partial C}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C}{\partial a^{(L)}}$$

Understanding $\partial C \,/\, \partial a^{(L)}$

For

$$C = (a^{(L)} - y)^2$$

$$\frac{\partial C}{\partial a^{(L)}} = 2(a^{(L)} - y)$$

So differences between the output and the expected value have a significant impact.

Understanding $\partial a^{(L)} \,/\, \partial z^{(L)}$

For

$$a^{(L)} = \sigma(z^{(L)})$$

$$\frac{\partial a^{(L)}}{\partial z^{(L)}} = \sigma'(z^{(L)})$$

The sigmoid function controls the impact that changes in $z^{(L)}$ will have to $a^{(L)}$.

Understanding $\partial z^{(L)} \ / \ \partial w^{(L)}$

For

$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

$$\frac{\partial z^{(L)}}{\partial w^{(L)}} = a^{(L-1)}$$

The amount that the weight influences the last layer, depends on how strong the previous layer is.

Chain Rule: all together

For the cost C:

$$\frac{\partial C}{\partial w^{(L)}} = a^{(L-1)} \quad \sigma'(z^{(L)}) \quad 2(a^{(L)} - y)$$

And of course this needs to be averaged over all training examples.

Learning: How to Reduce the Cost

Given a high cost for a sample, we can:

- Increase/decrease bias b (shift activation).
- Increase/decrease weights connected to the activated features.
- Change previous-layer activations through weight updates upstream.

Backpropagation computes the needed adjustments iteratively.

Link to Hebbian learning

Hebbian learning comes from the idea that neurons that "fire" together "wire" together.
Therefore, linking back to the backpropagation error, we can see that:

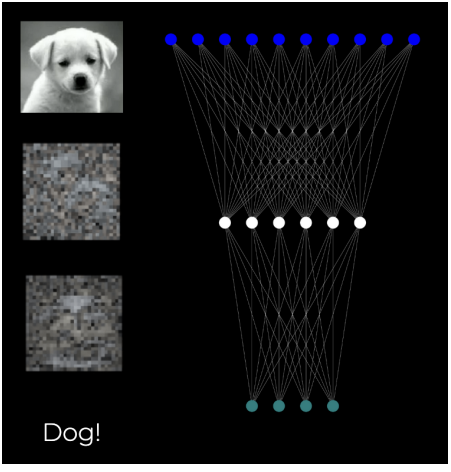
$$\frac{\partial z^{(L)}}{\partial w^{(L)}} = a^{(L-1)}$$

Indicates that the weights will change if both values for $a^{(L)}$ and $a^{(L-1)}$ are activated together, reminiscent of Hebbian co-activation.
However, backpropagation uses a global supervised error whereas Hebbian rules are local and unsupervised.

Intuition vs Real Neurons

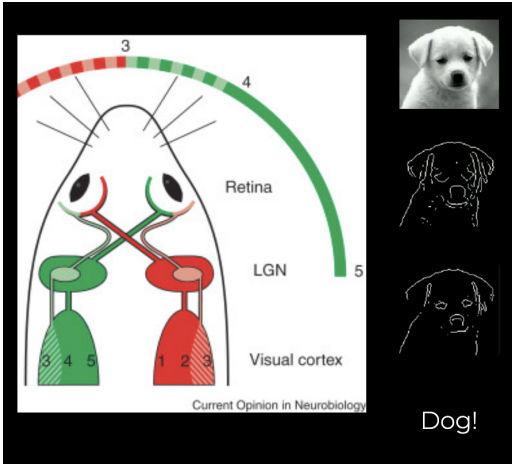
Hidden-layer patterns may not resemble simple features humans expect — they can appear noisy yet produce high performance. Biological neurons and networks implement distinct learning rules; ANNs use global supervised error signals.

Neural Network hidden layers



Processing images in a neural network.

Processing images in the brain



Processing images in the rodent brain (Hübener et al. 2003)

Why Patterns Look Noisy

- High-dimensional optimization yields many minima and solutions.
- Networks with $\sim$ 13k parameters can represent many functions; learned features are distributed.

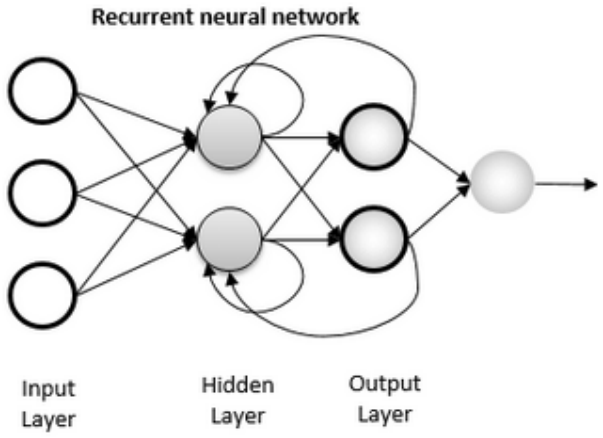
Random Input Behaviour

Networks can output high-confidence predictions on noise, such as a random input, because parameters map inputs into confident regions. Confidence of the neural network output does not imply semantic meaning.

Alternative network architectures

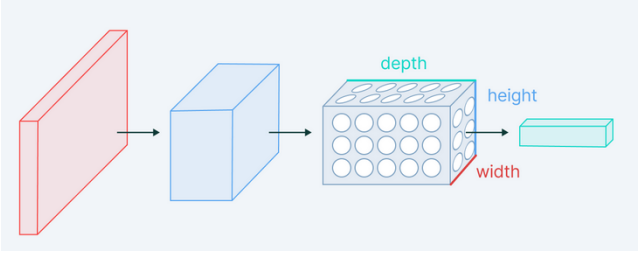
Multi-layer neural networks can get much more complicated if we modify the architectures. Two common examples are recurrent and convolutional neural networks.

Alternative architectures: RNN



Recurrent Neural Network.

Alternative architectures: CNN



Convolutional Neural Network.

Summary and Recap

- Neural networks consist of layers that are connected through weights.
- Weights and biases are the main parameters that are learned by minimizing a cost via gradient descent learning.
- Backpropagation uses the chain rule to compute gradients efficiently.
- Hidden layers provide hierarchical "feature" extraction.

Summary and Recap

- Features are not learned like in biological networks due to local minima.
- Hidden layer patterns can be non-intuitive.
- Random inputs can be interpreted with high confidence.
- There are links to biological Hebbian co-activation but also important differences.

Thank you — Questions?