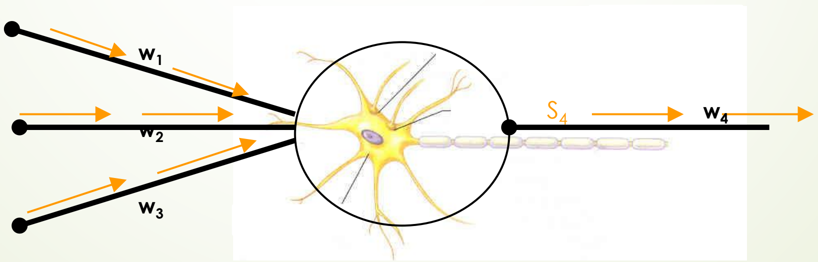
# Artificial Neural Networks, ANN

Mikael Pontarp

Lund University



# How ANNs Are Used in Biology

Inference vs. Mechanistic Modeling

# Two Fundamental Uses of ANNs

1. Statistical Inference (Most Common)

2. Forward Mechanistic Modeling (Less Common)

# ANNs as Statistical Inference Tools

Used to detect patterns, classify samples, and predict outcomes:

- Crow hybrid classification
  - E.g. identify and classify individuals
- Cancer prognosis prediction
  - E.g. Identify tumor subtypes
- Protein structure prediction (AlphaFold)
  - E.g. infer 3D structure from amino acid sequence
- Gene expression prediction
  - E.g. predict expression based on promoter sequences

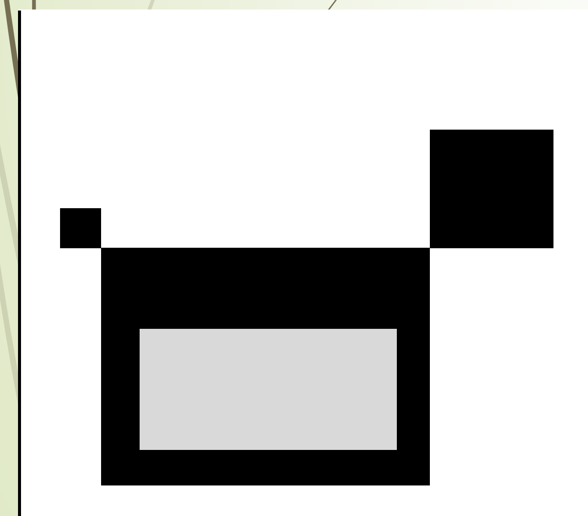
# Hybridization in the crow

- Carrion crow and hooded crow hybridize across Europe.
- Hybrid zone produces intermediate phenotypes.
- Classification task: determine species from image data.
- ANN needed because classification relies on pixel patterns.

# Input signals from "pixel" images

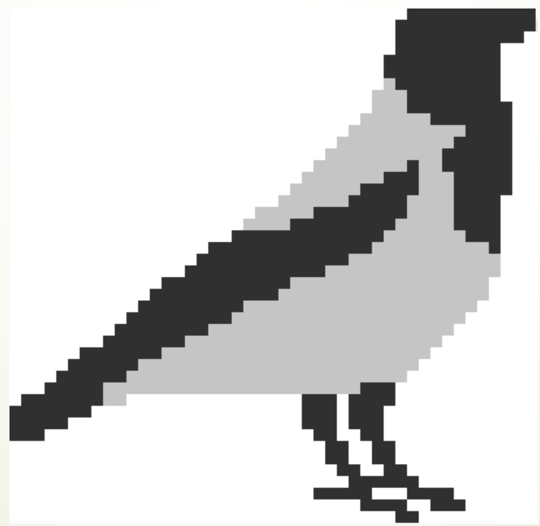
$S_1 = -1 =$ $S_2 = 0 =$ $S_3 = 1 =$

14 x 14



196

45 x 45

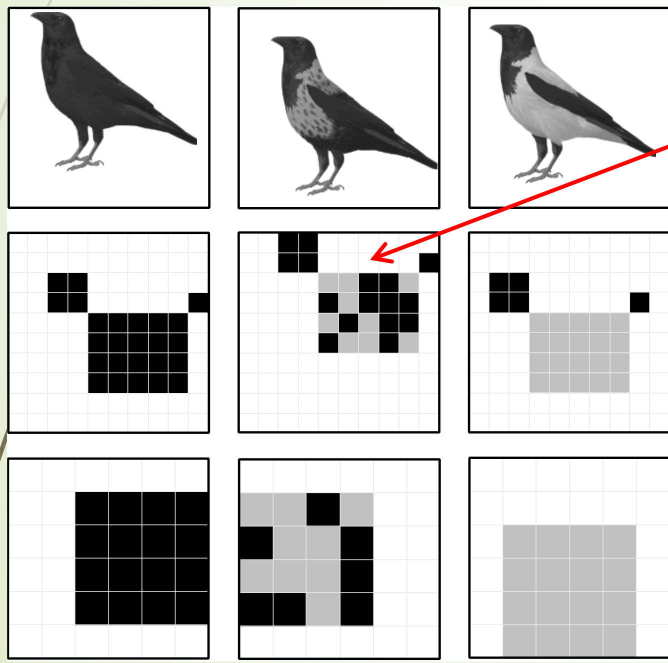


2025

200 x 200

40000

# What information is the network learning?

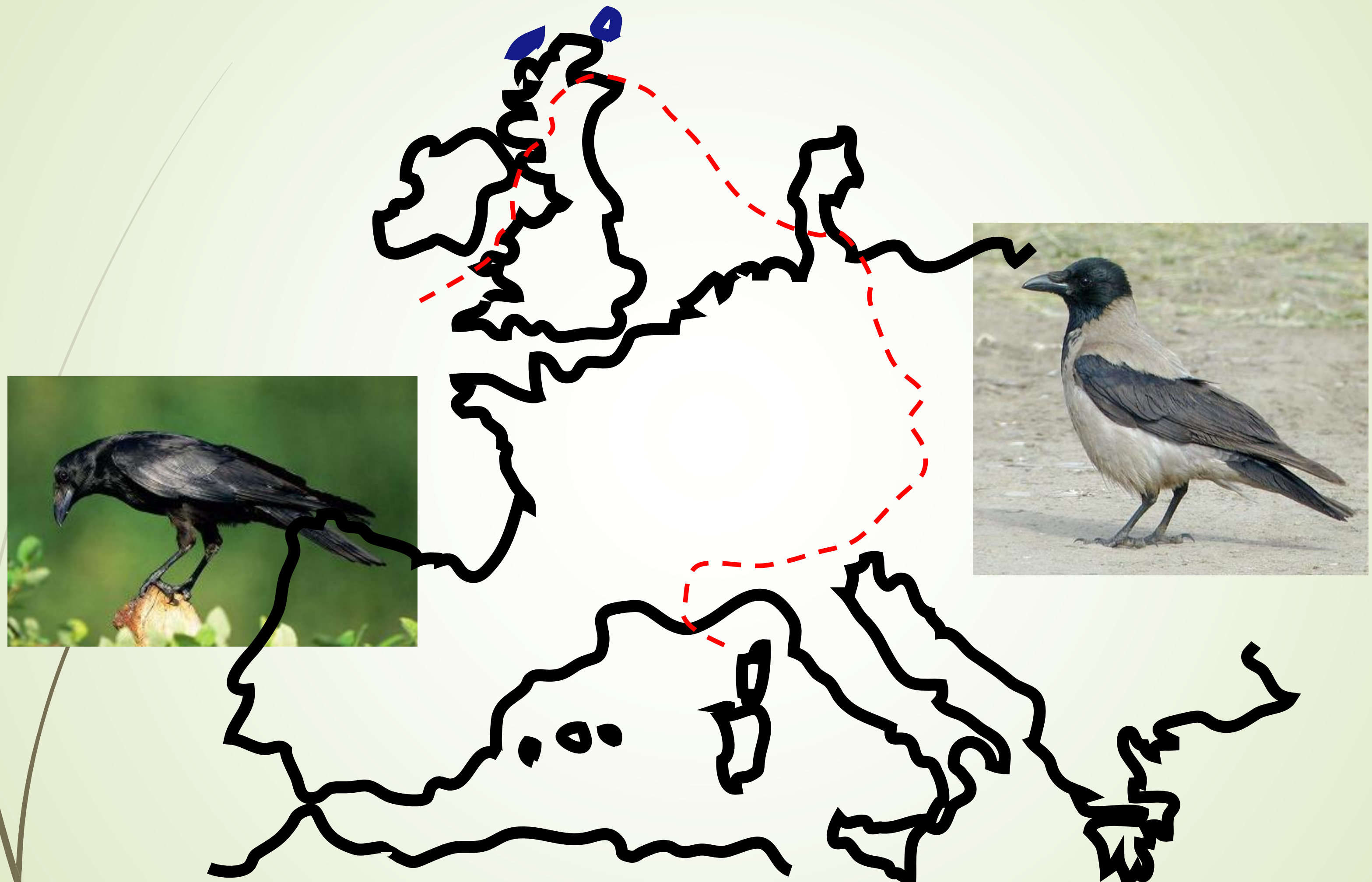


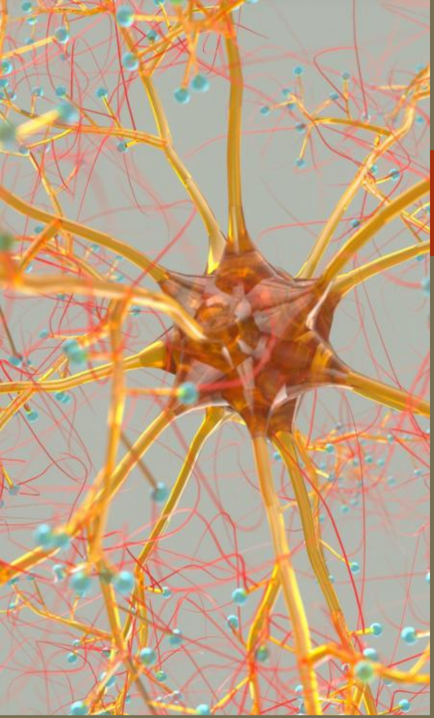
0 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 -1 -1 ...

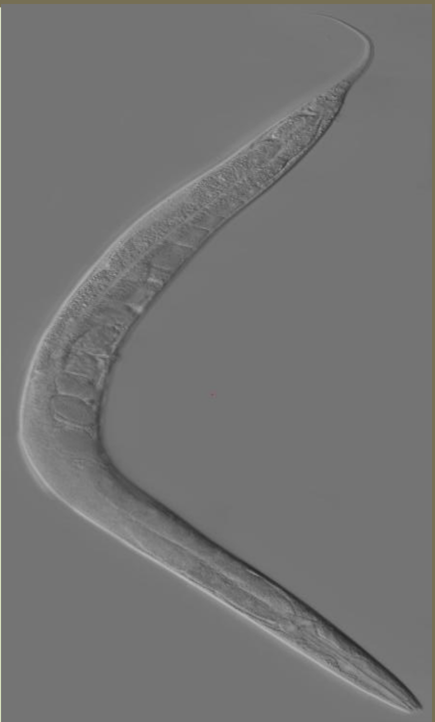
0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0 ...

# Images as ANN Inputs: Pixels Become Signals

- Images are converted into numerical input values.
- Each pixel → a signal (e.g., black=1, gray=0, white=−1).
- Higher resolution → more input neurons:
- – 14×14 → 196 pixels
- – 45×45 → 2,025 pixels
- – 200×200 → 40,000 pixels
- ANN must learn patterns in these high-dimensional vectors.





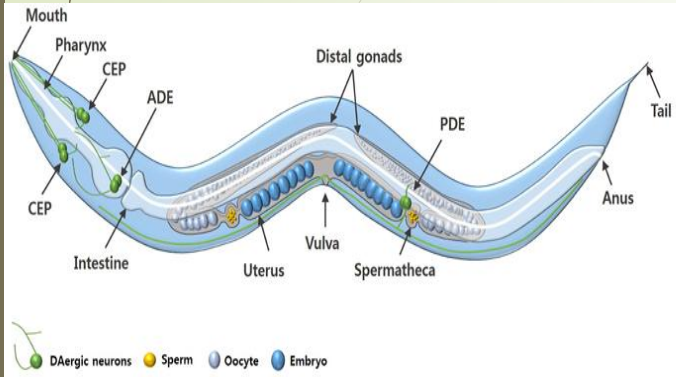


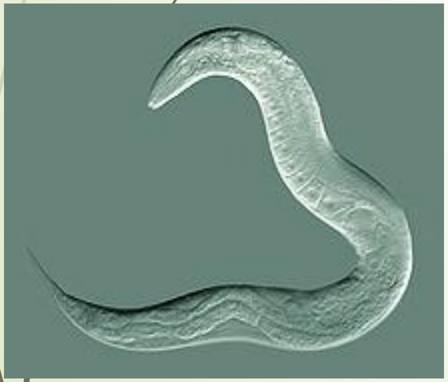
# ANNs as Forward Models

Used to simulate biological processes or decisions:

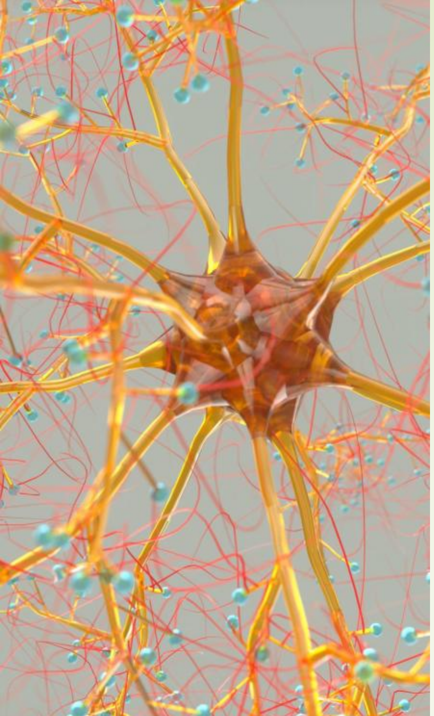
- Neuronal logic (AND/OR/XOR)
  - E.g. gene regulation, immune response, sex determination
- Simulate movement, sensory response, etc
  - E.g. C. elegans neural circuit
- Retinal feature detection models
  - E.g. edges, motion, light
- Models of animal behavior
  - E.g. navigation, foraging, defense systems

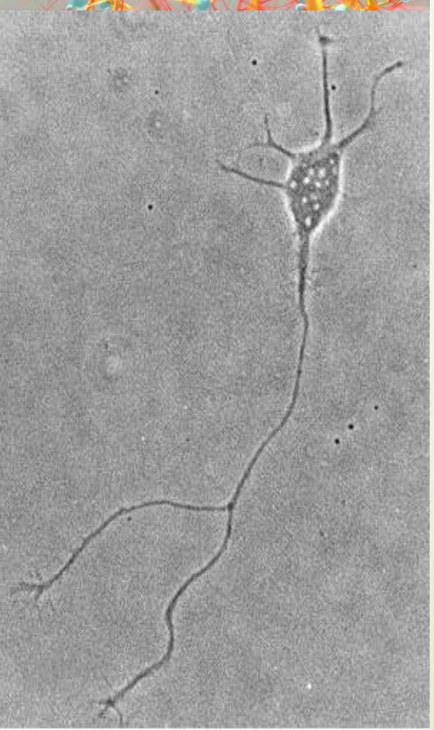
# ANNs as Forward Models



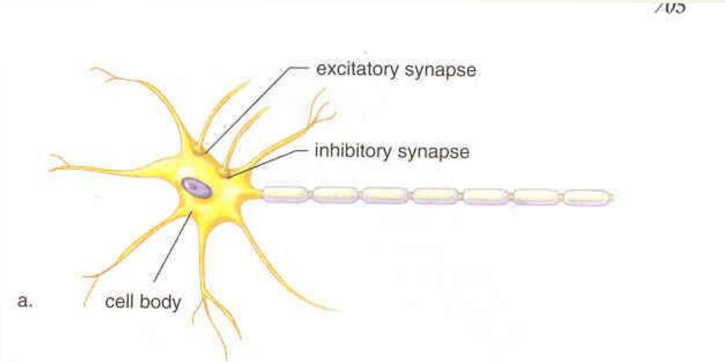



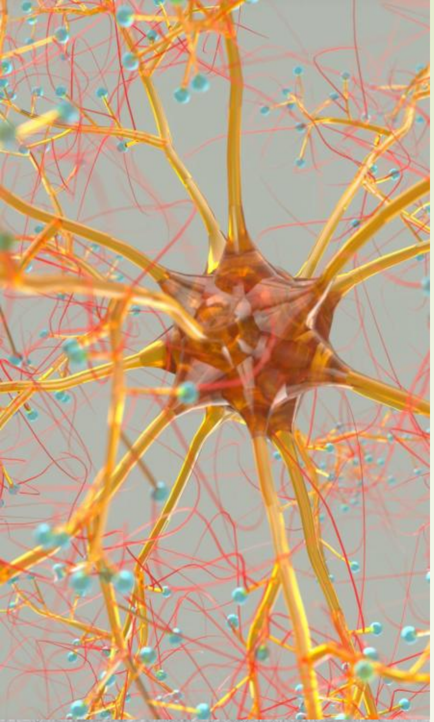
- C. elegans has only 302 neurons — a fully mapped biological neural network.
- Its connectome inspires ANN models that simulate behavior from neural wiring.
- ANNs can mimic sensory input → internal processing → motor output.
- Forward simulation: given stimuli, the ANN predicts organism responses.
- Enables testing hypotheses about behavior, decision-making, and neural control.
- Demonstrates how artificial networks can approximate real biological systems.

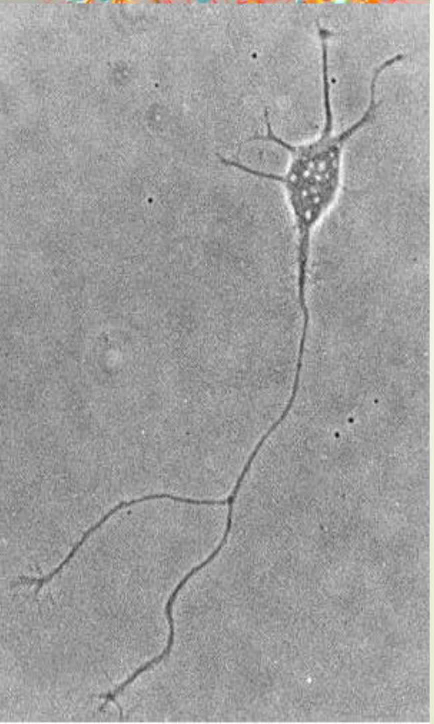




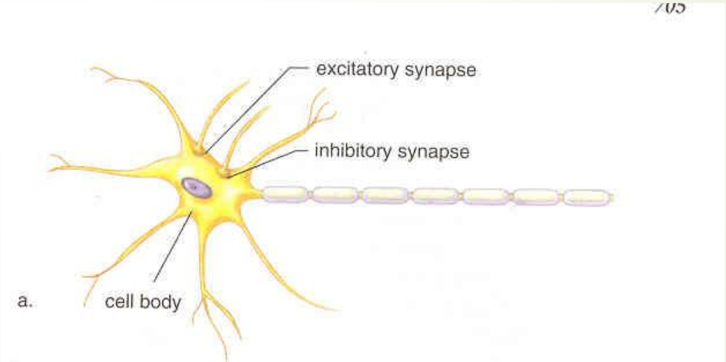
- ANN architecture is biologically inspired
- ANN usage is usually statistical
- ANN interpretation can be mechanistic
- Sits between ML, systems biology, and neuroscience







- a neuron accumulates signals from synapses on dendrites and soma
- fires at some threshold
- spike that propagates to other neurons in axon



# How Simple Neurons Create Complex Decisions

- A single artificial neuron can only draw one straight decision boundary.
- It can learn simple rules like AND or OR, but not XOR.

- When neurons are connected in layers, each learns a simple rule (a simple boundary).
- By combining these simple boundaries, the network forms complex, nonlinear decisions.
- This is why multilayer networks can solve XOR and other nonlinear problems—many simple neurons working together create powerful behavior.

# Summary: AND, OR, and XOR Logic

- AND (“and”): Output = 1 only when BOTH inputs are 1.

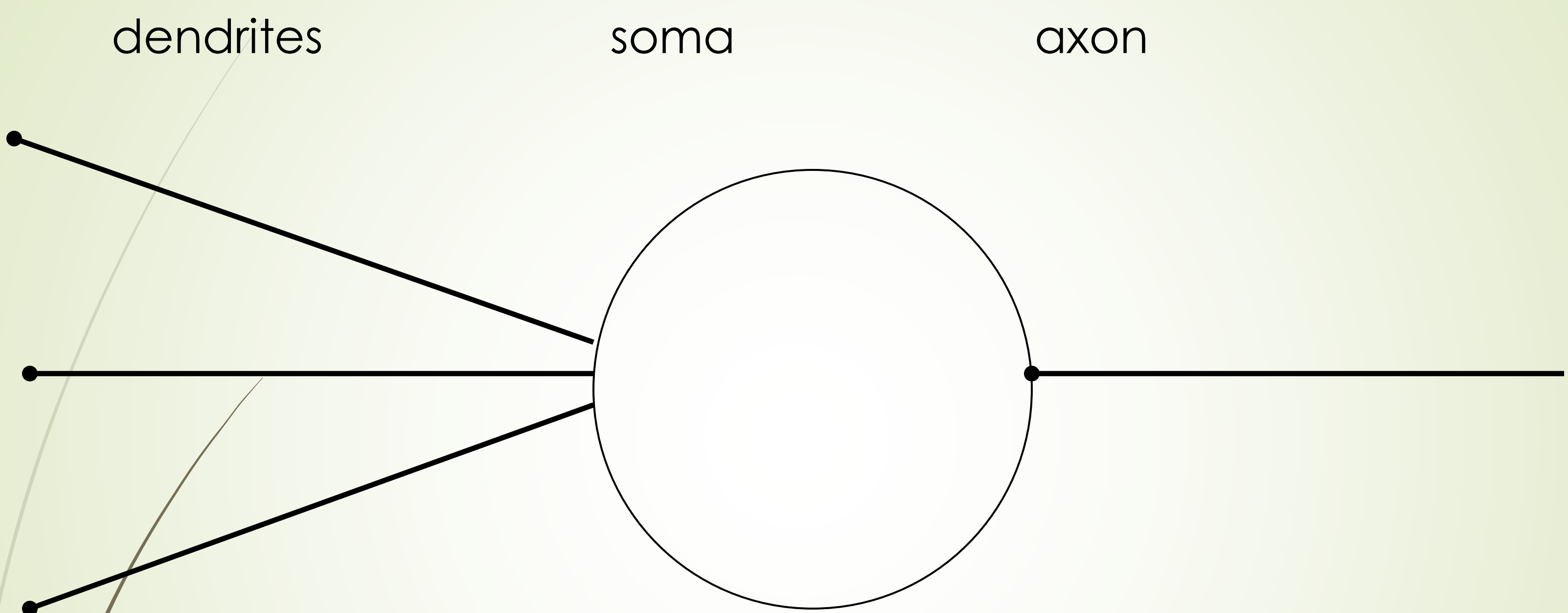
 – Linearly separable → a single neuron can learn it.

- OR (“or”): Output = 1 when AT LEAST ONE input is 1.
  – Also linearly separable → a single neuron can learn it.
- XOR (“exclusive or”): Output = 1 when EXACTLY ONE input is 1.
  – NOT linearly separable → requires multiple neurons (hidden layer).

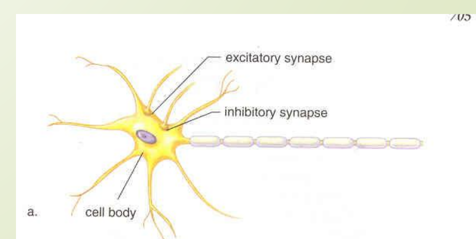
# Biological Examples of AND, OR, and XOR Logic

- AND: Gene expression requires two transcription factors
- – Example: A gene activates only when TF1 AND TF2 bind.
- OR: Immune cells activate if any danger signal is detected
  – Example: viral RNA OR fungal β-glucan triggers inflammation.
- XOR: Output depends on exactly one of two signals
  – Example: Reptiles sex determination, extreme (high or low) temperature → female while intermediate temperature → male.

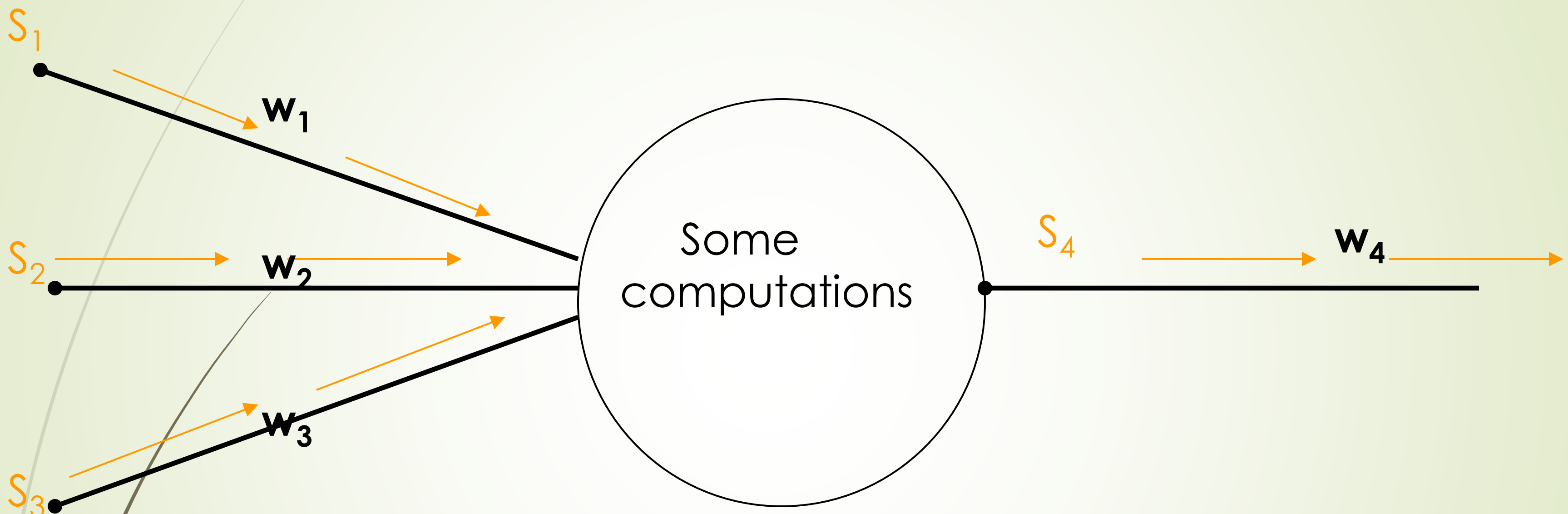
# Artificial neuron:



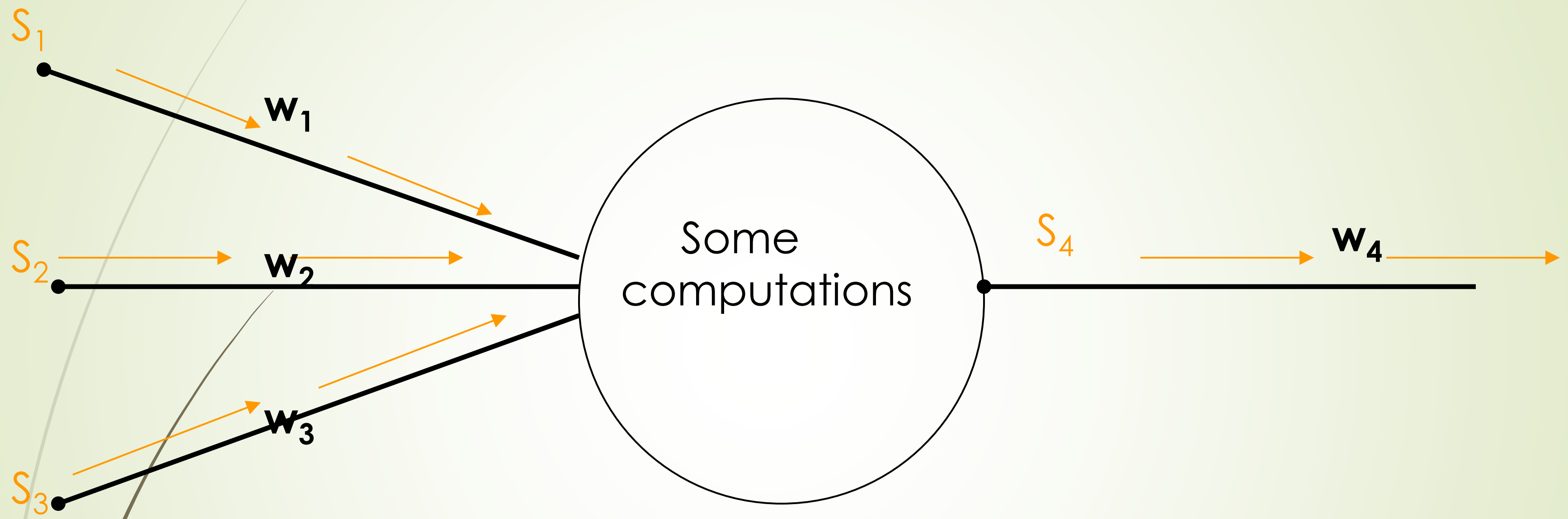

dendrites = axons!



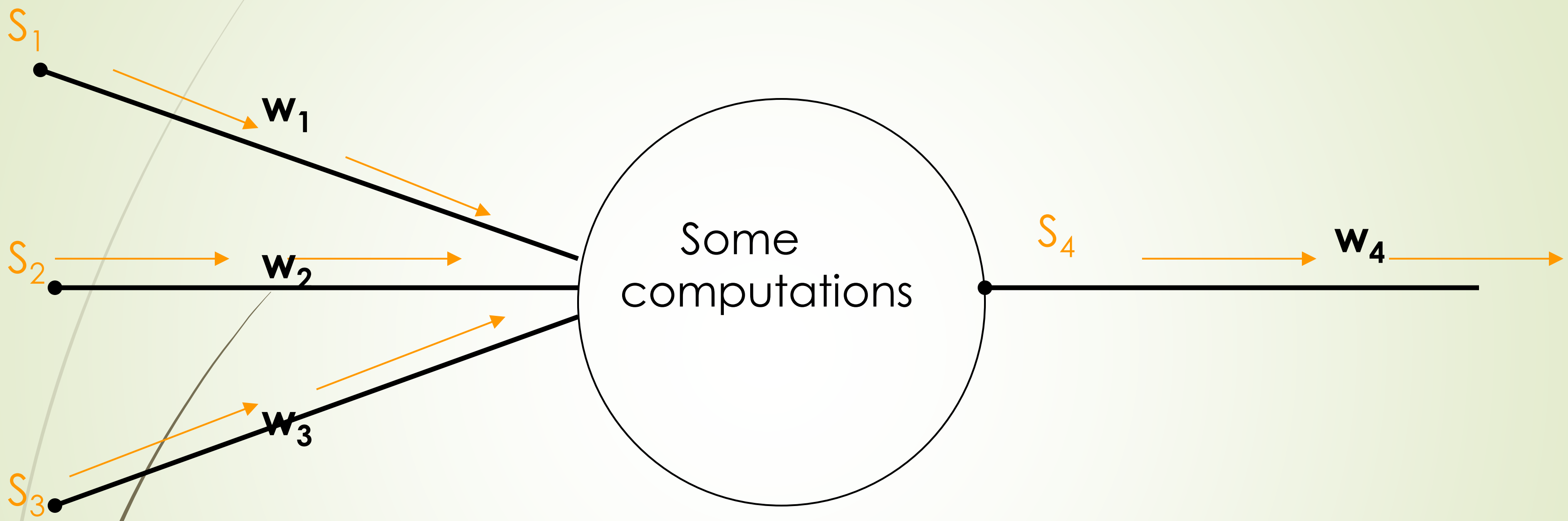
# Dendrites and axons in real neurons, weights (w) in artificial ones:



# Input: Three input signals, $S_1, S_2, S_3$



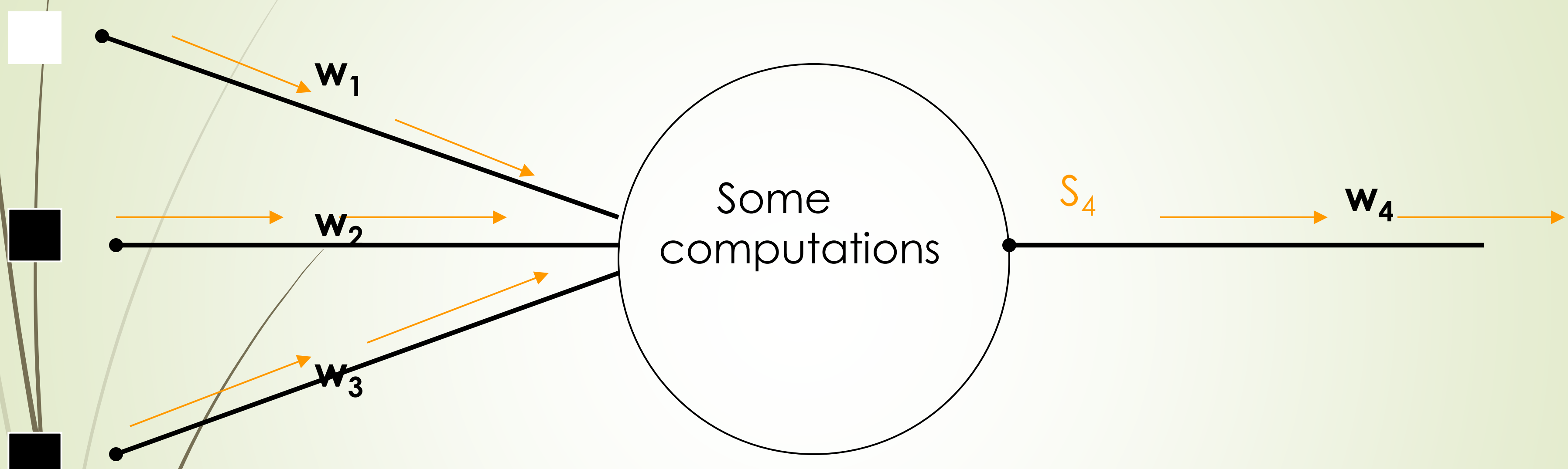
# Input: Three input signals, $S_1$,$S_2$,$S_3$




Output: one signal, $S_4$

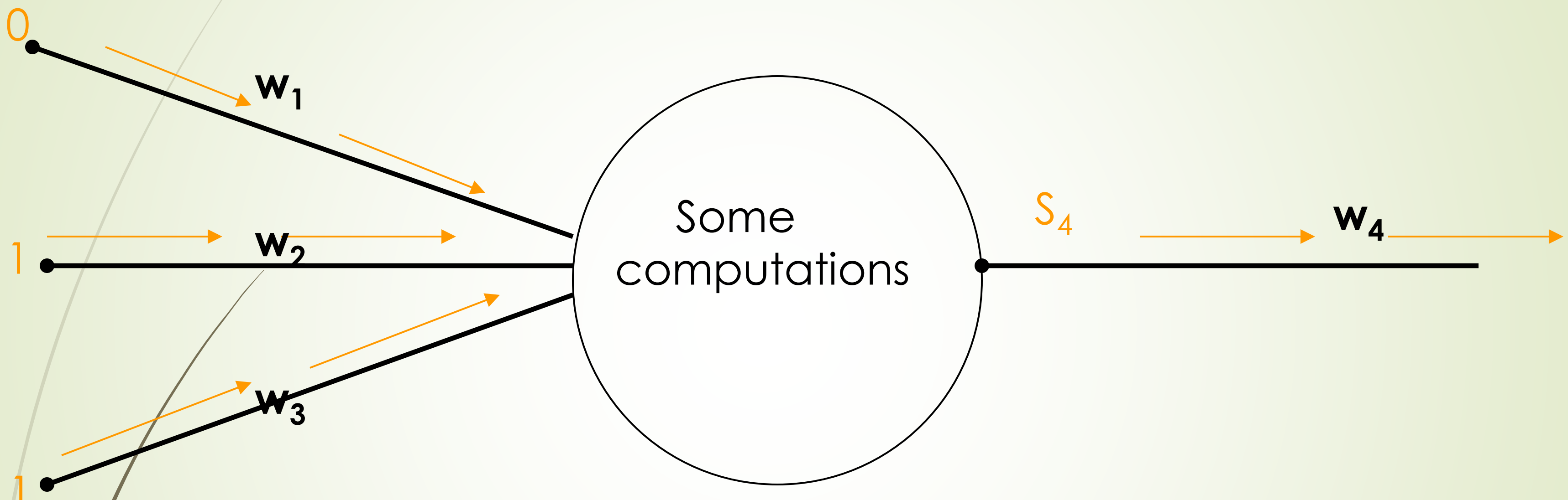
Input: Three input signals, $S_1, S_2, S_3$

Could represent white - black – black…



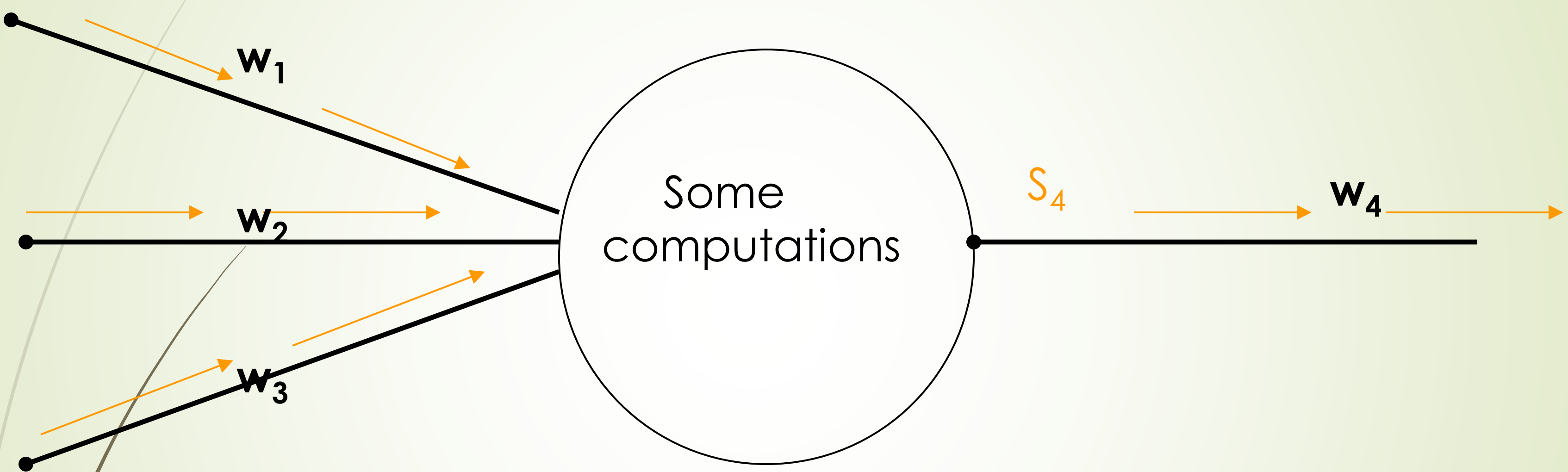
# Input: Three input signals, $S_1$,$S_2$,$S_3$

…or 0 1 1

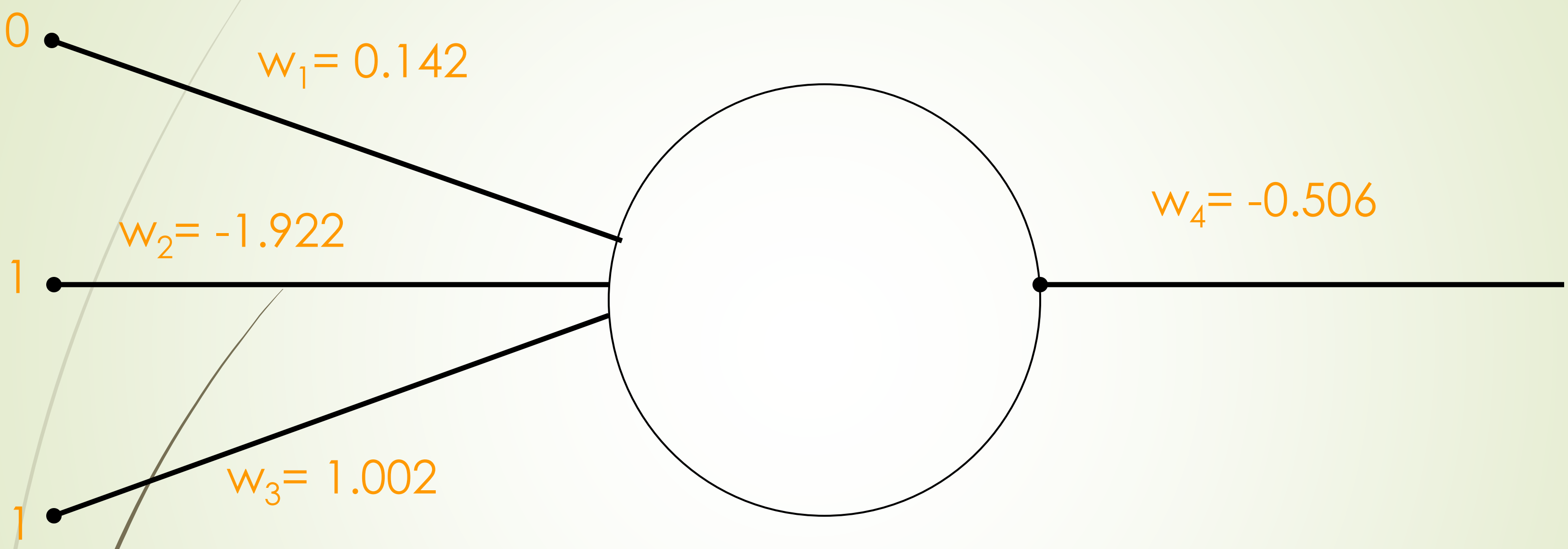


# Input: Three input signals, $S_1,S_2,S_3$

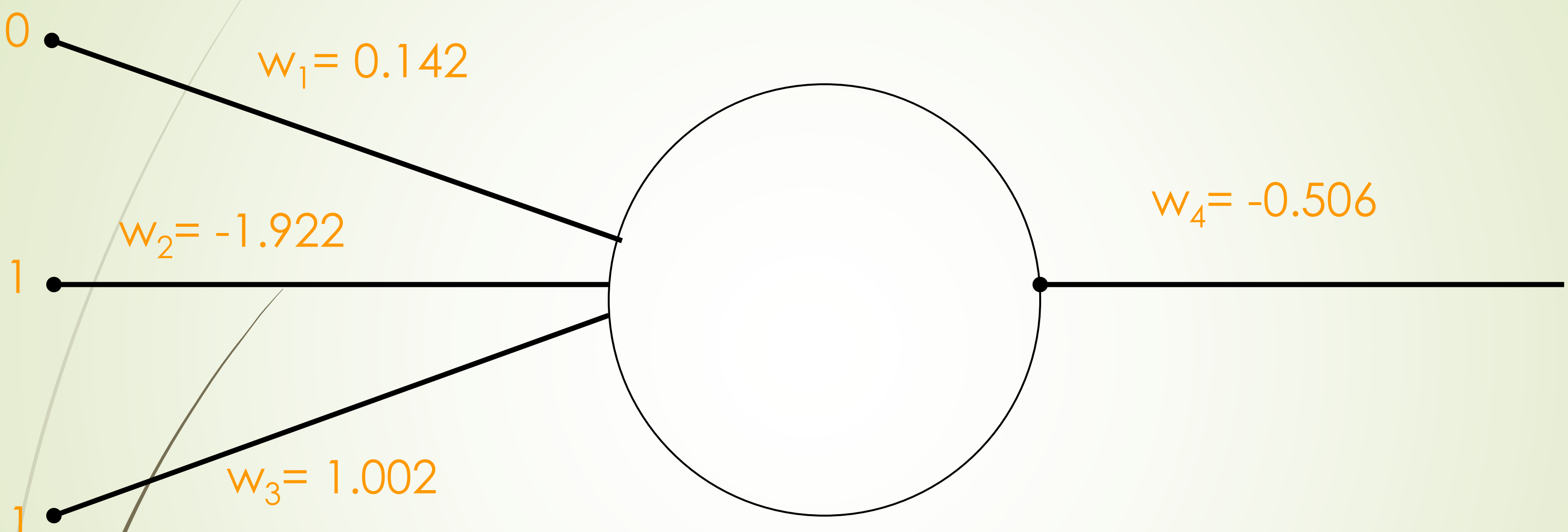
…or more often continous values



# Set weights to random decimal numbers

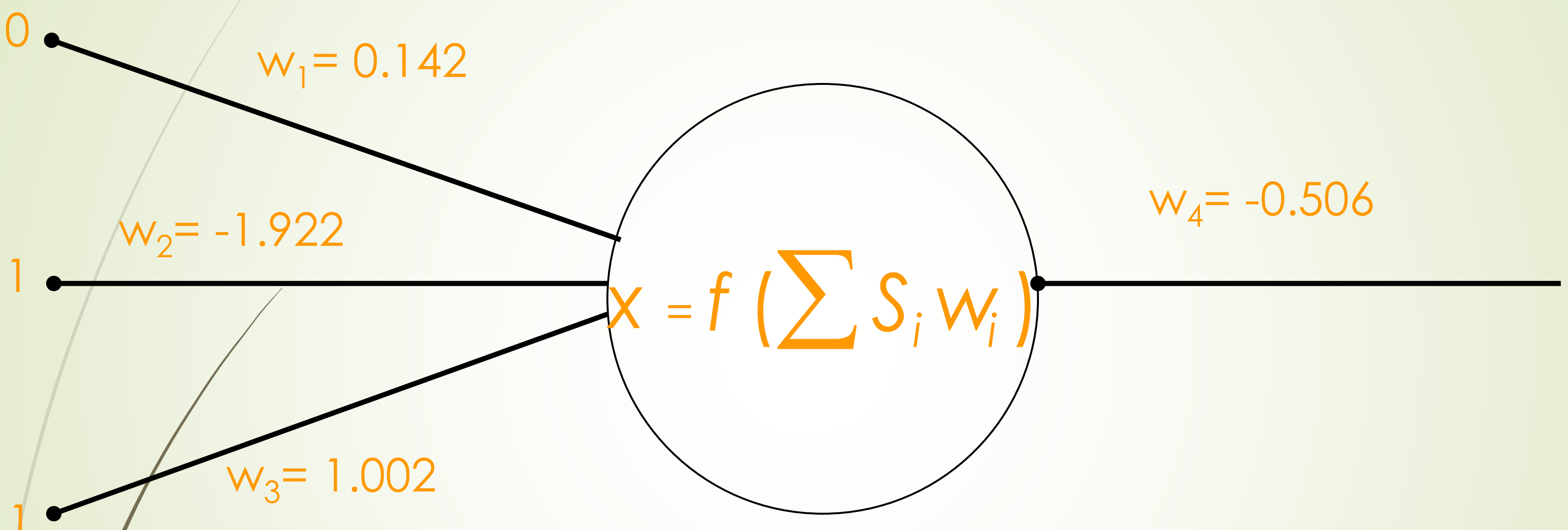


# Multiply with input signals



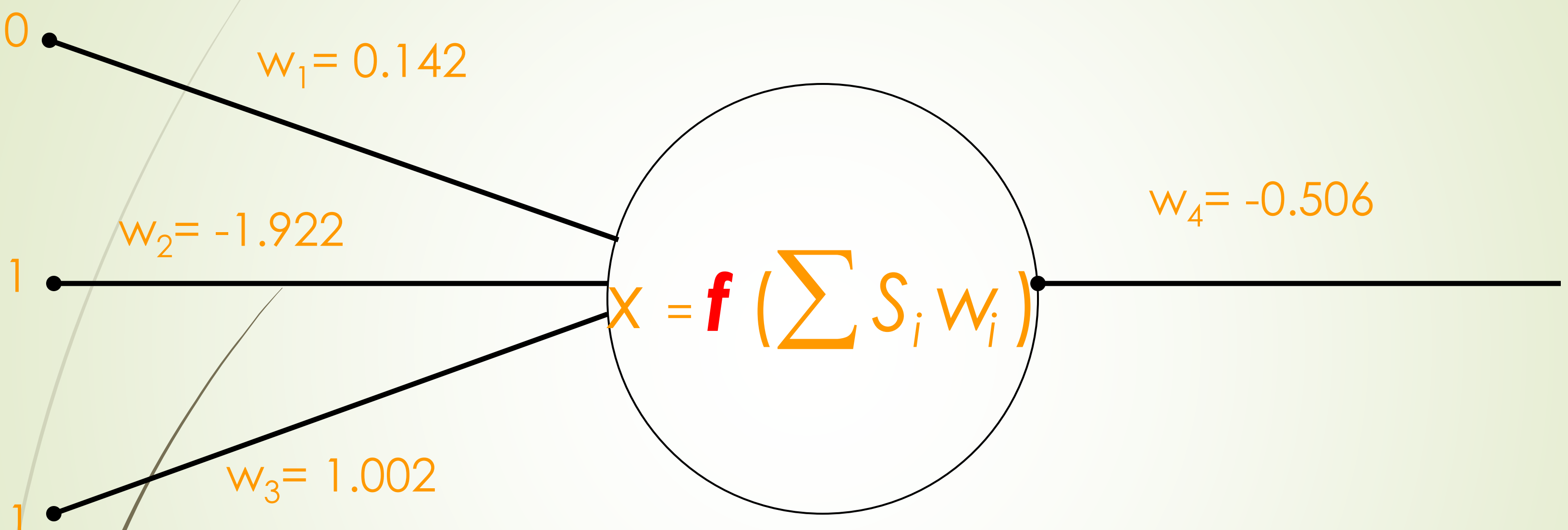

$$\sum S_i w_i = 0 * 0.142 + 1 * (-1.922) + 1 * 1.002$$

# This happens in the "soma" = node = neuron body

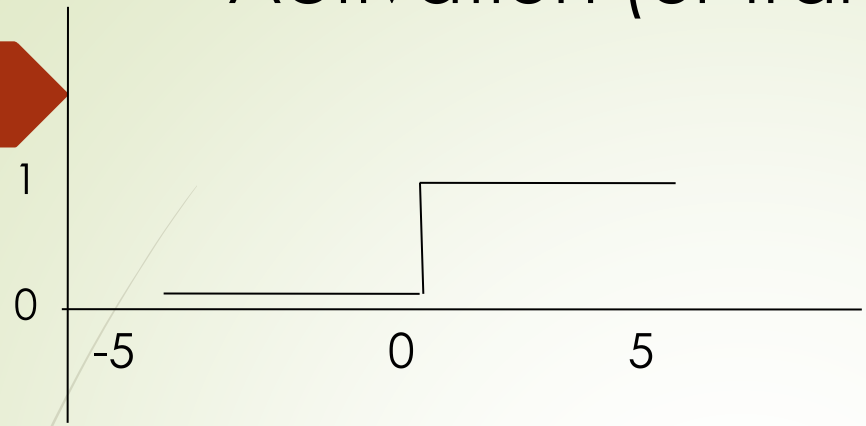



$$\sum S_i w_i = 0 * 0.142 + 1 * (-1.922) + 1 * 1.002$$

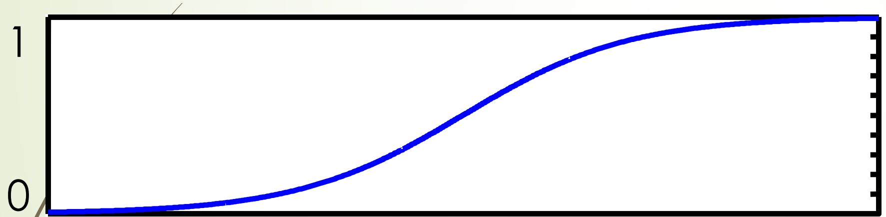
# Activation function decides how signal propagates



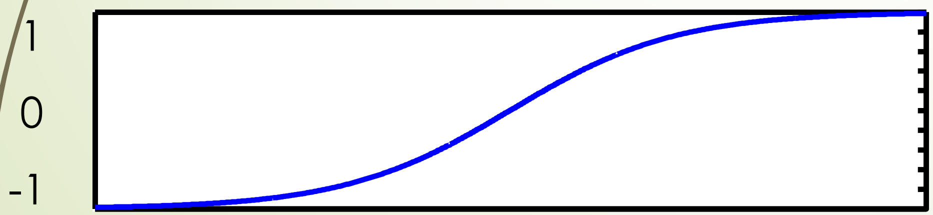
# Activation (or transfer) functions




$$f(T) = \begin{cases} 0 \text{ if } T \leq 0 \\ 1 \text{ if } T > 0 \end{cases}$$

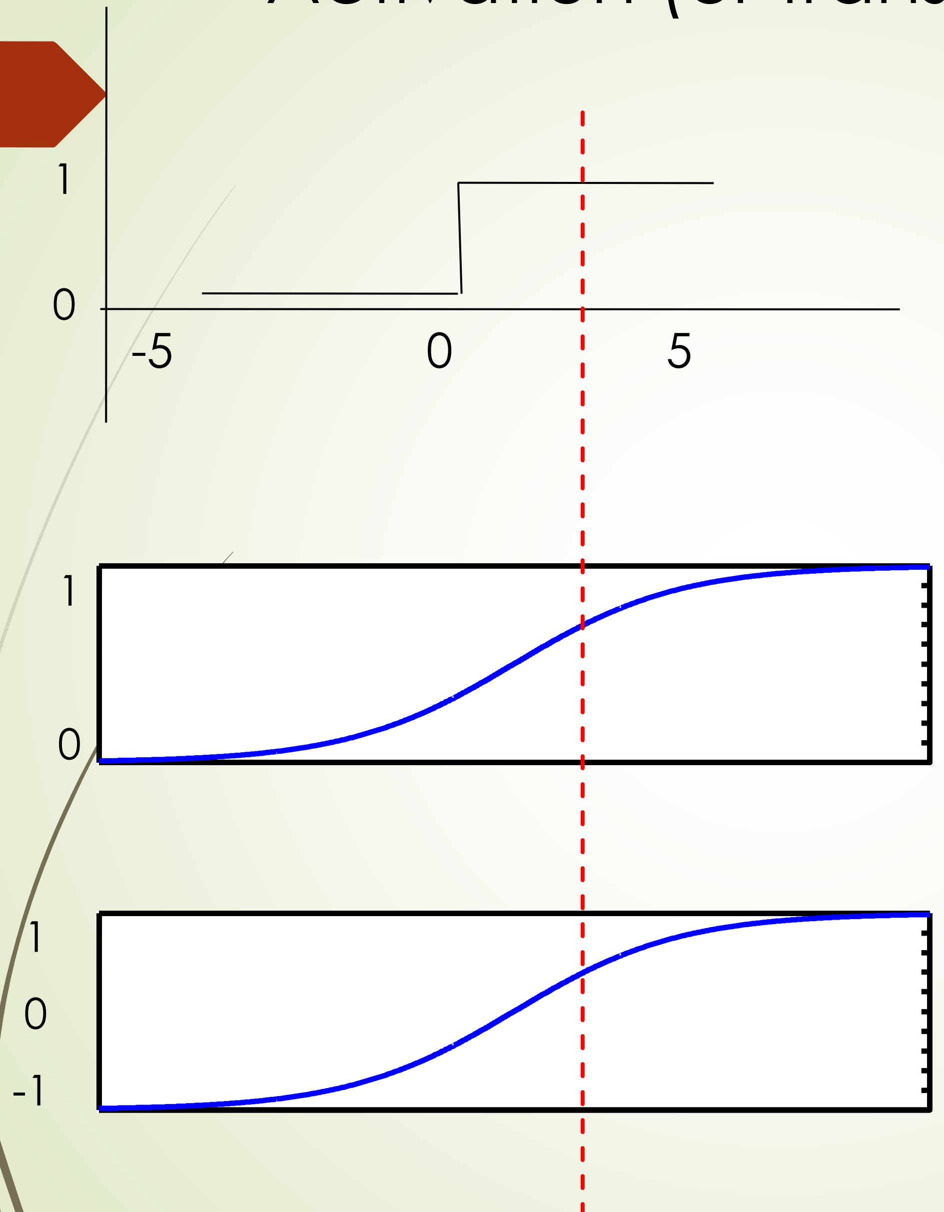



$$f(T) = \frac{1}{1 + e^{-T}}$$




$$f(T) = \tanh(x)$$

# Activation (or transfer) functions

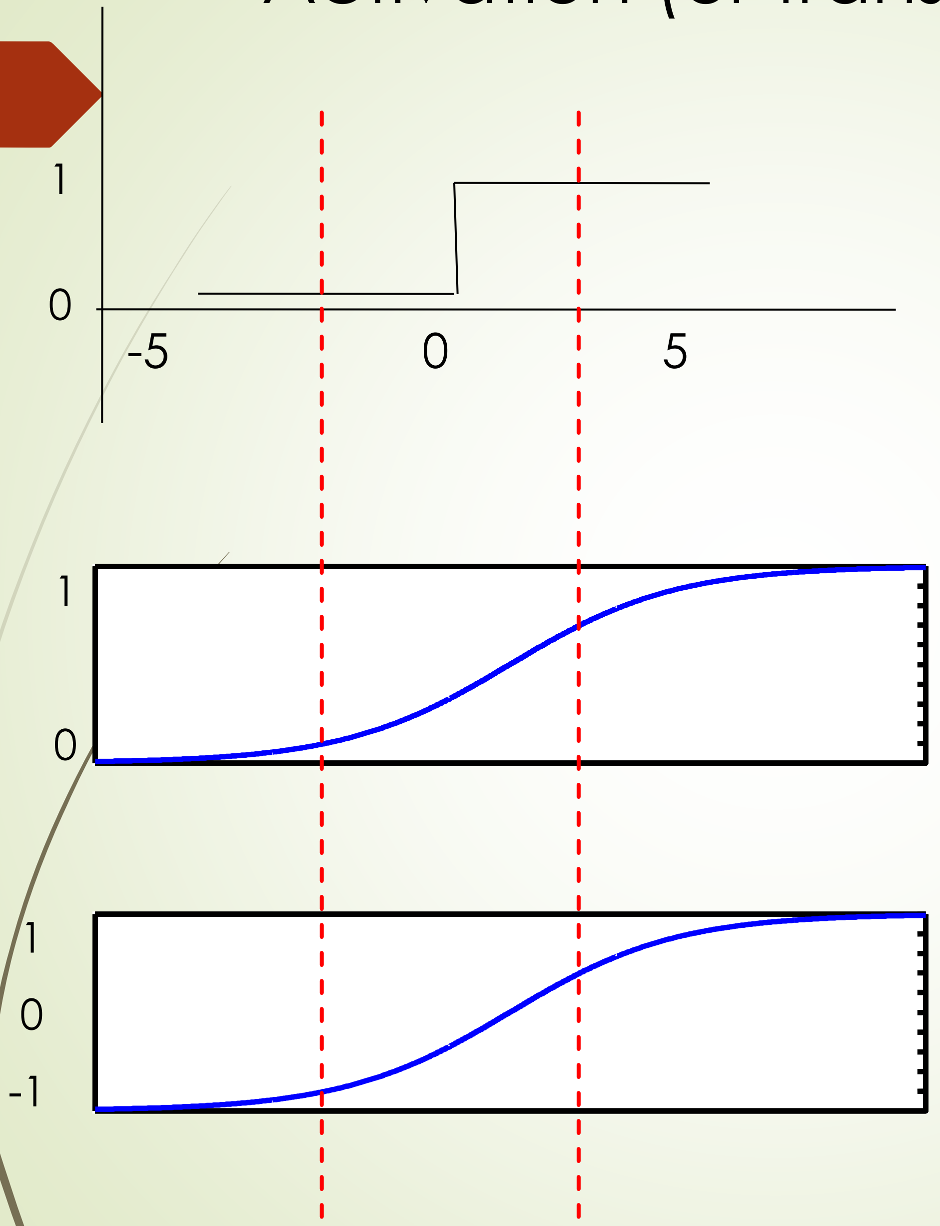



$$f(T) = \begin{cases} 0 \ \text{if} \ T \le 0 \\ 1 \ \text{if} \ T > 0 \end{cases}$$

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

$$f(T) = \tanh(x)$$

# Activation (or transfer) functions

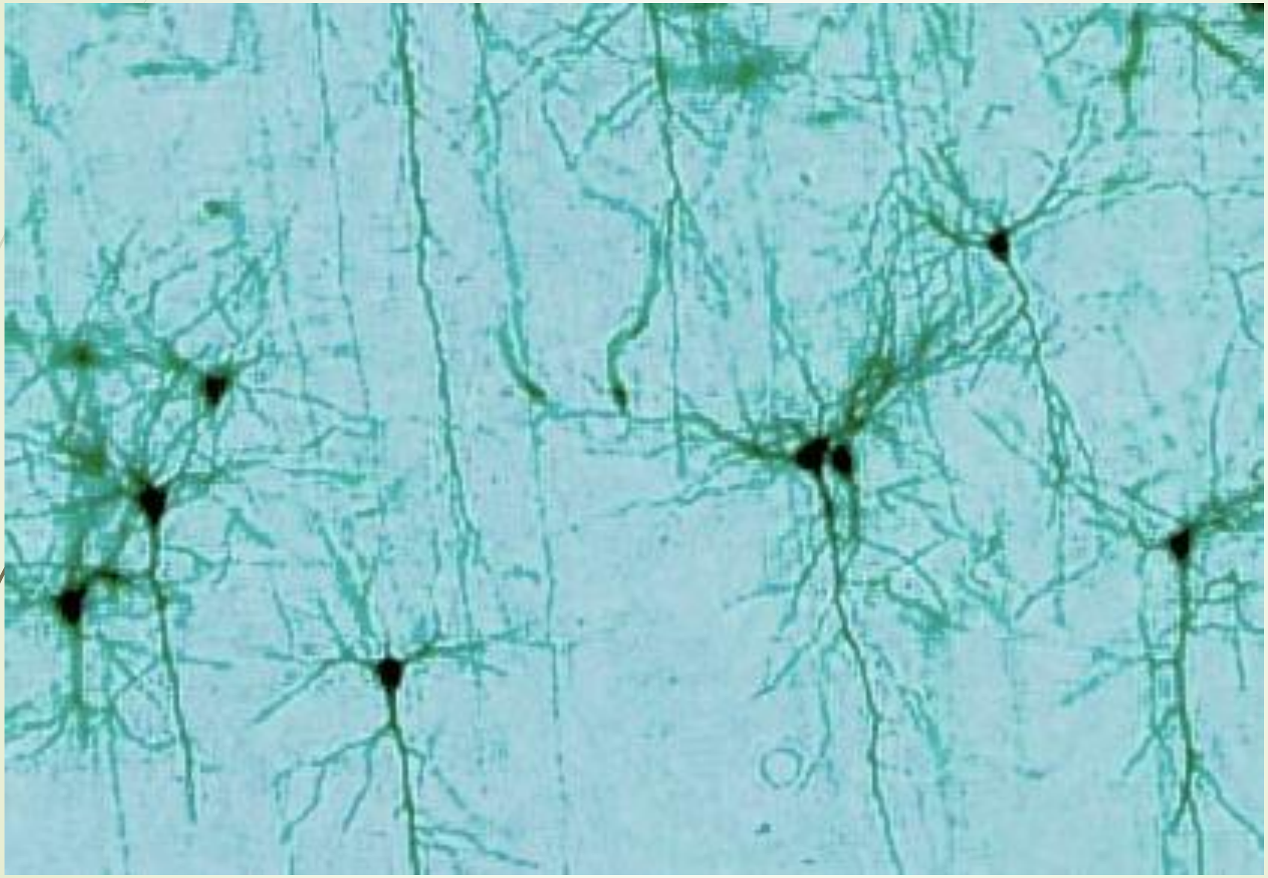



$$f(T)=\begin{cases}0 & \text{if } T\le 0\\ 1 & \text{if } T>0\end{cases}$$

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

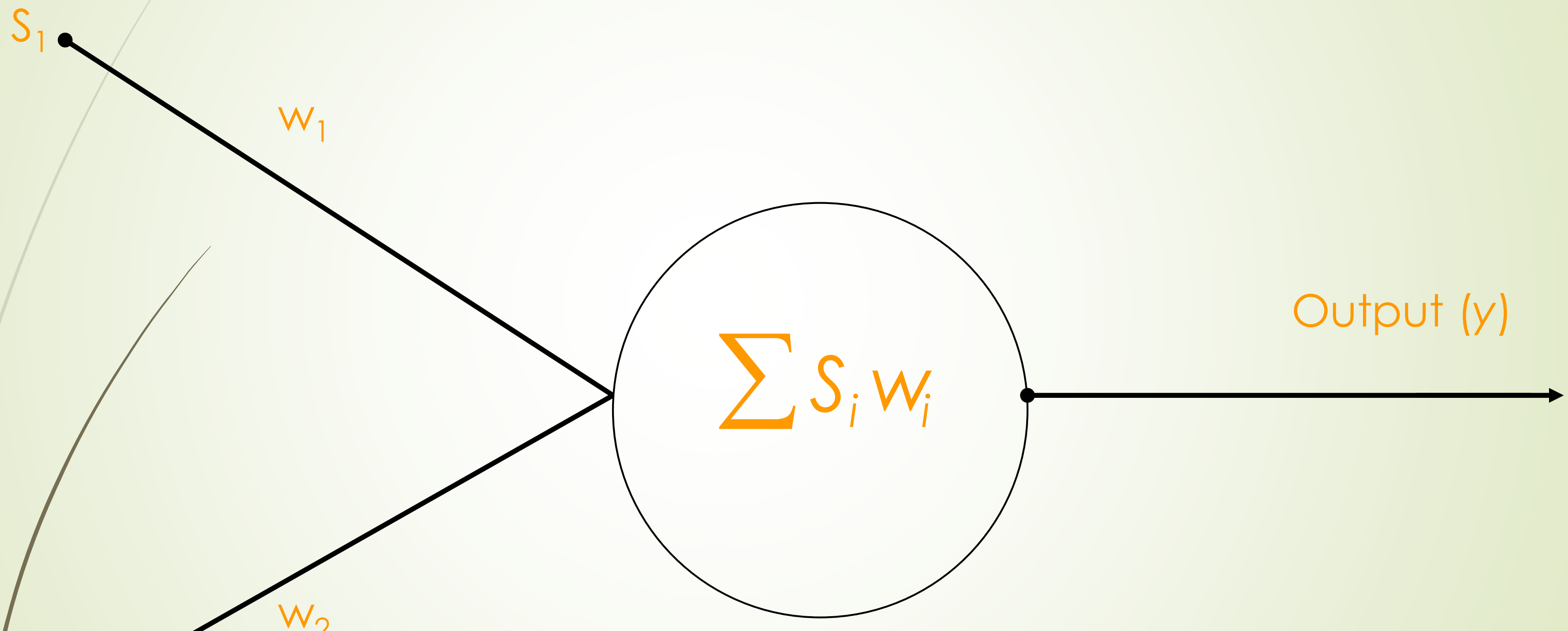
$$f(T)=\tanh(x)$$

# Real network:



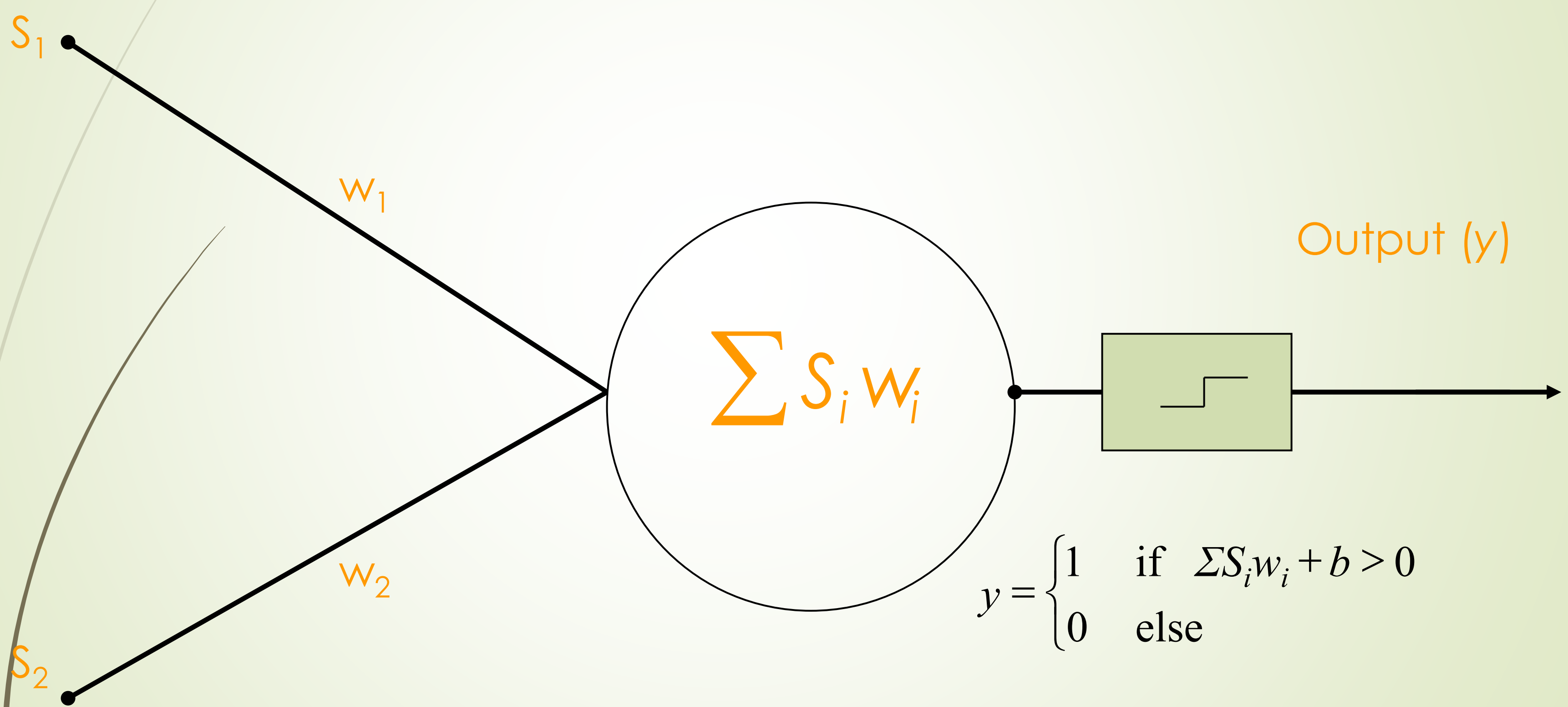
# The simplest network:

# a single-layer perceptron



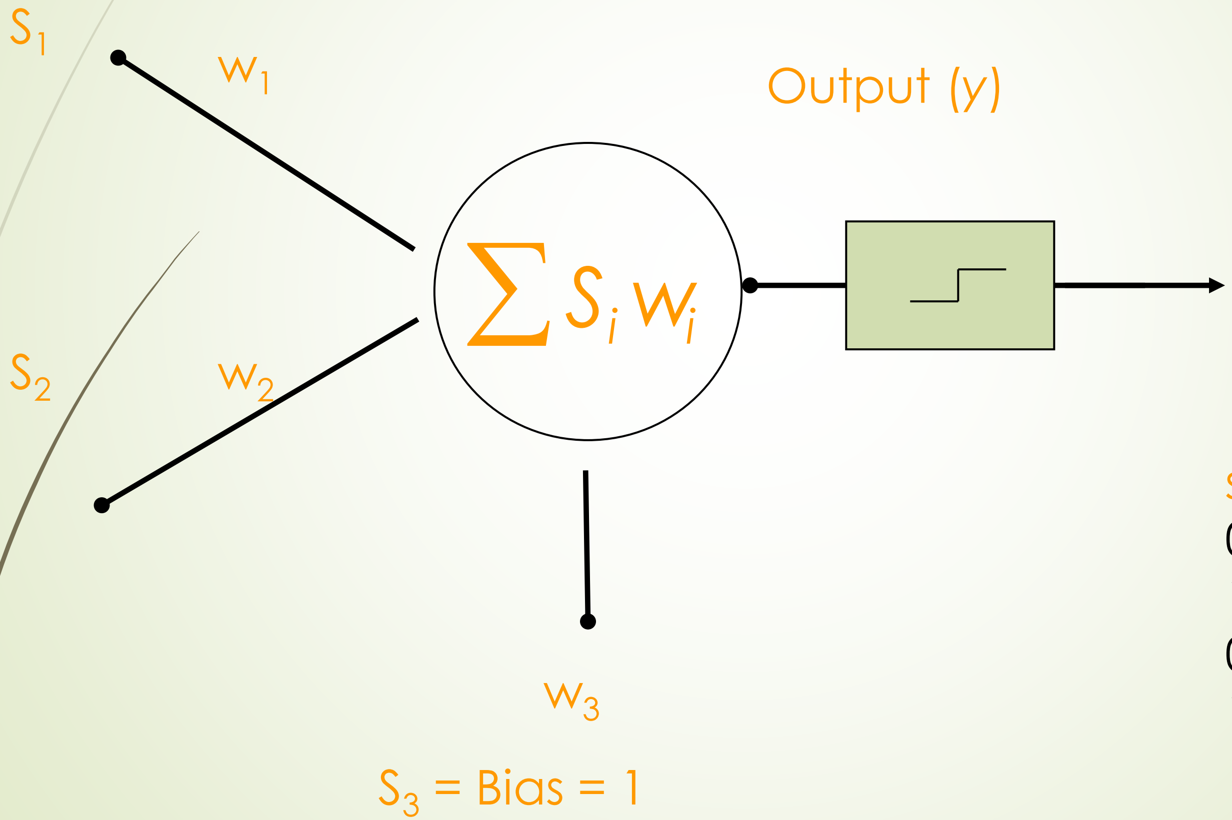

A single-layer perceptron is a neural network that:
• Has one layer of trainable weights and no hidden layers
• Contains one or more neurons receiving all input features directly
• Computes a weighted sum and applies an activation function
• Can represent only linear decision boundaries

# Activation function




$$y = \begin{cases} 1 & \text{if } \Sigma S_i w_i + b > 0 \\ 0 & \text{else} \end{cases}$$

# Bias and truth table




| $S_1$ | $S_2$ | out |
|---|---|---|
| 0 | 0 | ? |
| 1 | 0 | ? |
| 0 | 1 | ? |
| 1 | 1 | ? |

# Bias in a Perceptron

- The bias acts like a constant input always equal to 1
- It allows shifting the decision boundary left or right
- Equivalent to a neuron's threshold
- Without bias, many functions cannot be learned

# Truth Table Use in Training

- The truth table lists all possible inputs and desired outputs
- For each row, compute: $T = \Sigma(S_i w_i)+b$
- Apply activation function → prediction
- If prediction ≠ target → update weights
- Example (AND):
-   (0,0→0), (1,0→0), (0,1→0), (1,1→1)

# Training it on logical operators:

## AND:

| $s_1$ | $s_2$ | out |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

## OR:

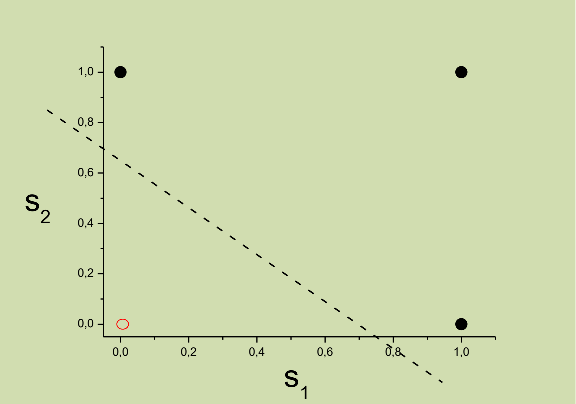
| $s_1$ | $s_2$ | out |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 1 |

## XOR:

| $s_1$ | $s_2$ | out |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 0 |

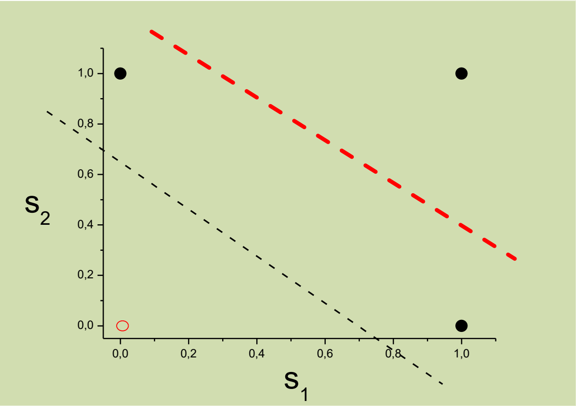
# Graphical Representation of the OR Function

- Inputs $s_1$ and $s_2$ plotted in 2-D plane
- Black dots = OR = 1 (1,0), (0,1), (1,1)
- Red open dot = OR = 0 (0,0)
- Dashed line = perceptron decision boundary
- Line separates false from true cases
- OR is linearly separable → single-layer perceptron can learn it



# Graphical Representation of the AND Function

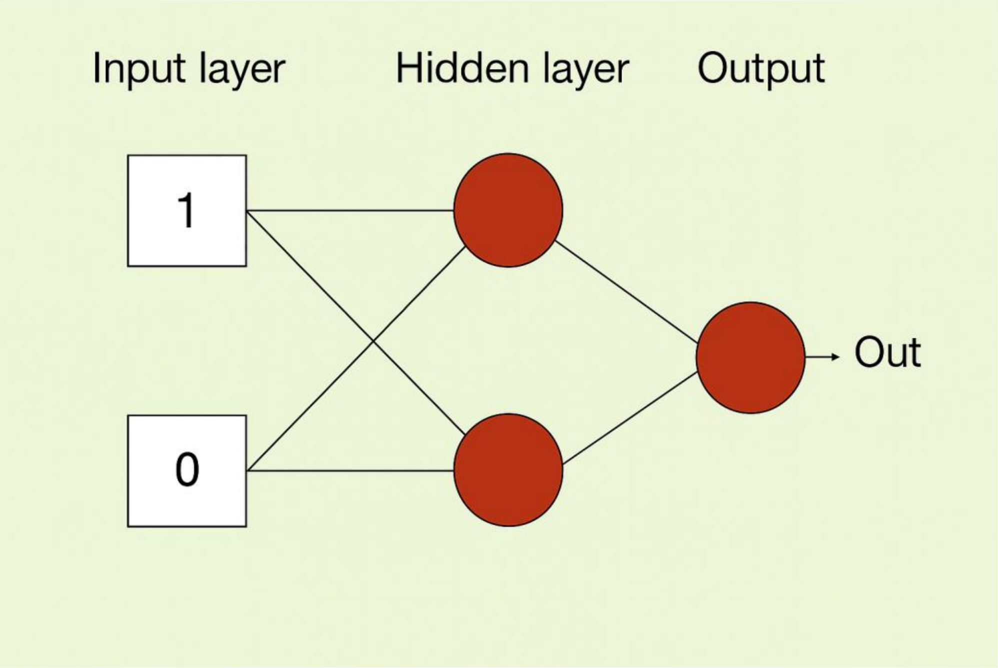
- Inputs $s_1$ and $s_2$ plotted in 2-D plane
- Dashed red line = perceptron decision boundary
- Boundary separates true from false cases
- AND is linearly separable → perceptron can learn it



# Why XOR Cannot Be Learned by a Single-Layer Perceptron

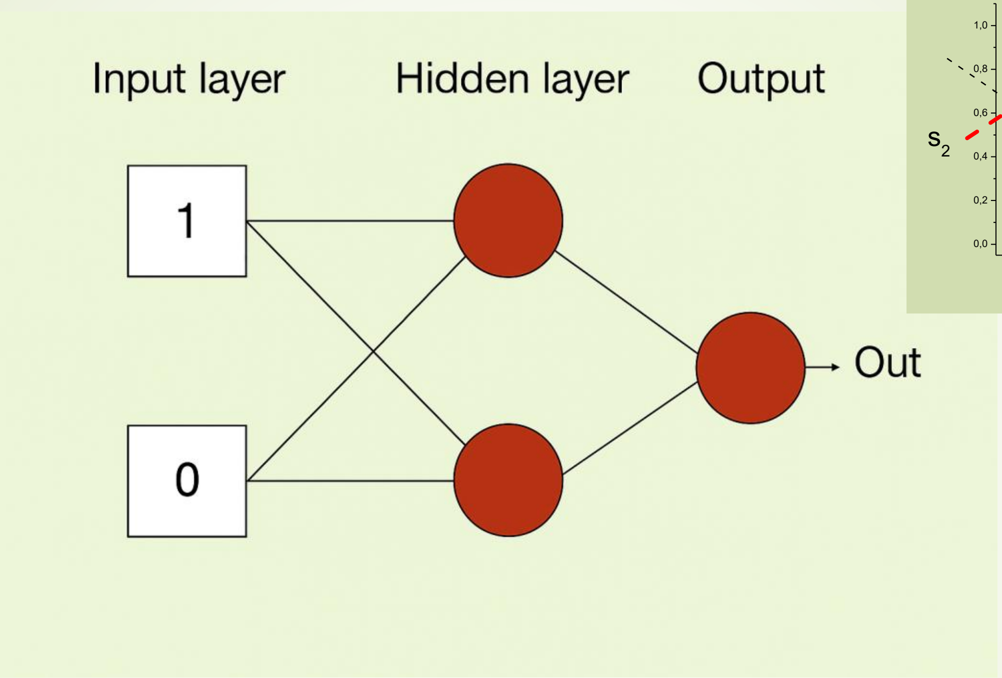
- XOR truth table: (0,1) and (1,0) → 1; (0,0) and (1,1) → 0
- Points cannot be separated by a single straight line
- No linear decision boundary separates true and false cases
- Therefore: XOR is NOT linearly separable
- A perceptron can only learn linearly separable functions
- → XOR requires a multi-layer network (with hidden layer)

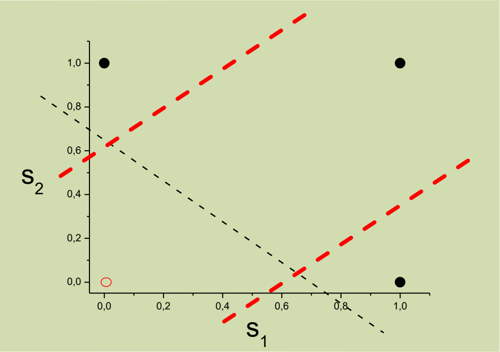
# The more complex network:




- \- Hidden Neuron 1 learns one boundary.
- \- Hidden Neuron 2 learns the other boundary.
- \- The output neuron combines the two boundaries into the XOR rule.
- \- This transforms the input so XOR becomes linearly separable at the output layer.
- \- Smallest architecture that solves XOR: 2 input → 2 hidden → 1 output.

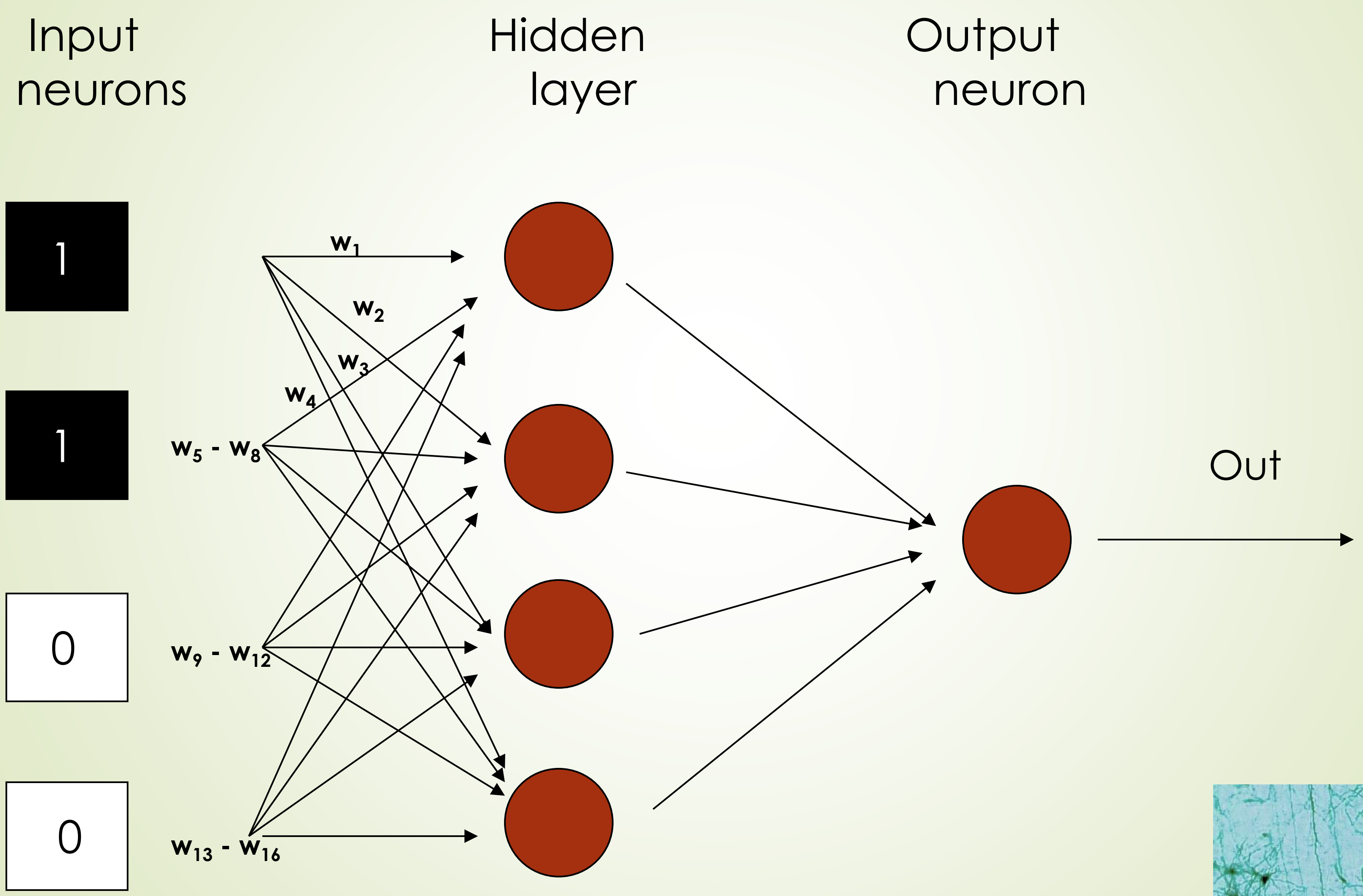
# The more complex network:

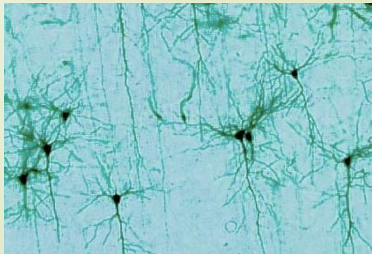


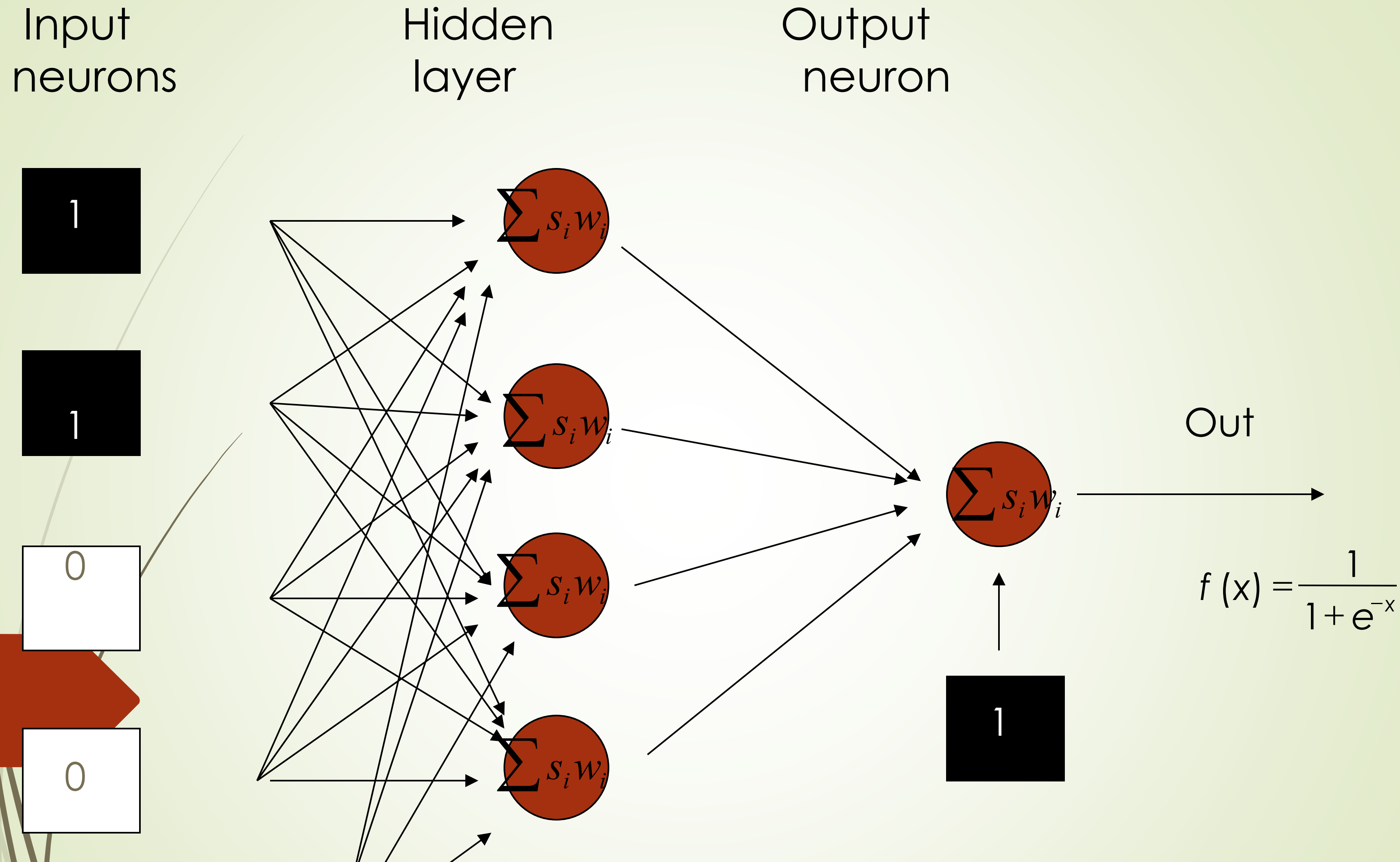

- - Hidden Neuron 1 learns one boundary.
- - Hidden Neuron 2 learns the other boundary.
- - The output neuron combines the two boundaries into the XOR rule.
- - This transforms the input so XOR becomes linearly separable at the output layer.
- - Smallest architecture that solves XOR: 2 input → 2 hidden → 1 output.

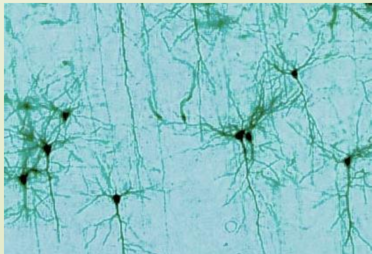
# A larger network



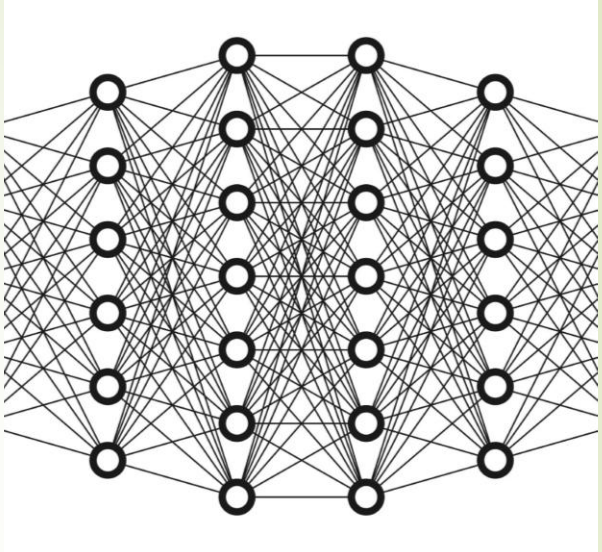
Adding activation functions:

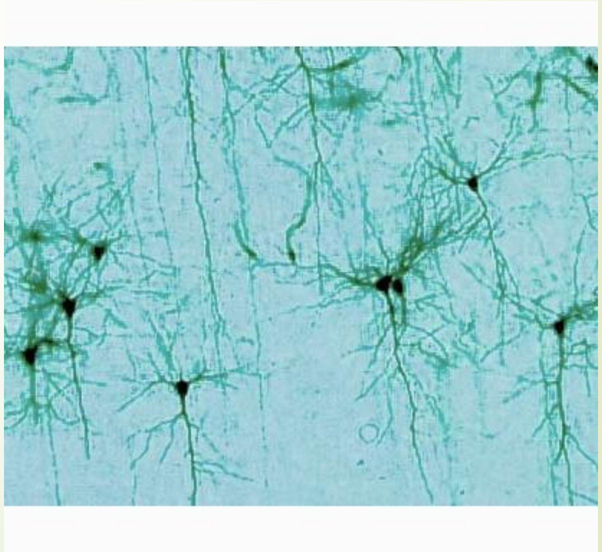


# How Hidden Layers Learn Complex Nonlinear Problems

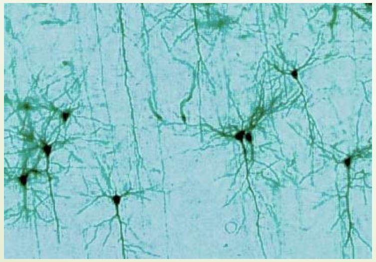
- Single-layer perceptrons can only learn linear boundaries.
- Hidden layers introduce nonlinear transformations.
- Each layer progressively extracts more abstract patterns from input data.
- Enables solving problems far beyond linear separability.

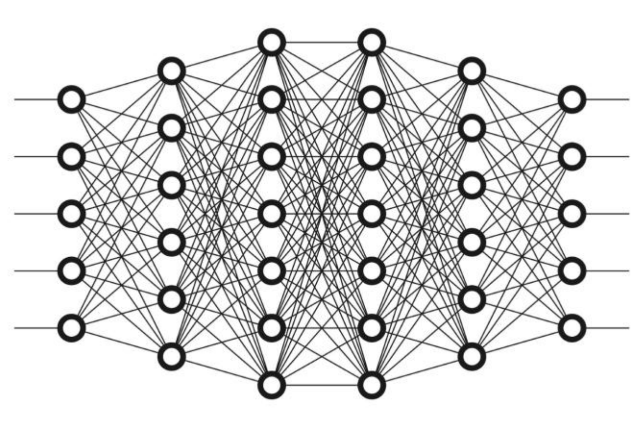




# Step 1: Hidden Neurons Create Multiple Linear Boundaries

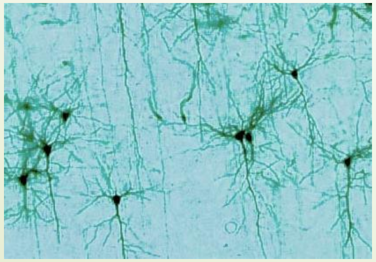
- Each hidden neuron computes a weighted sum of inputs.
- This creates a separate linear decision boundary (line/plane/hyperplane).
- With H hidden neurons, the network forms H linear partitions.
- These partitions divide input space into multiple regions.

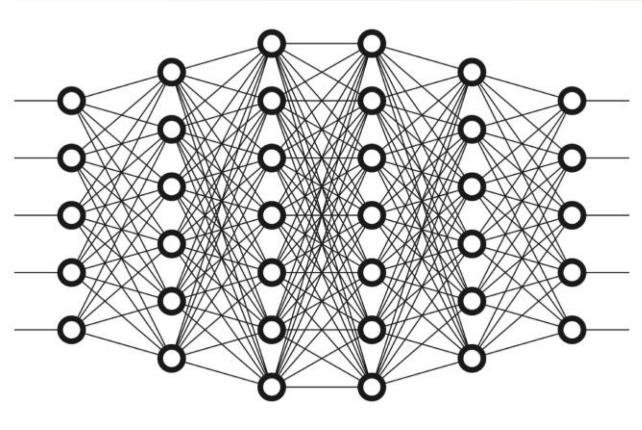




# Step 2: Nonlinear Activations Bend and Combine Regions

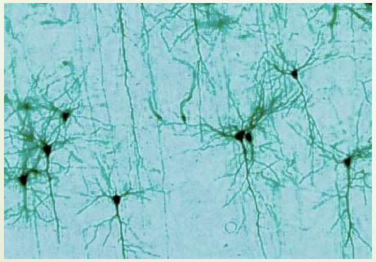
- Activations like sigmoid, tanh, and ReLU introduce nonlinearity.
- They allow the network to warp boundaries into curves.
- Networks can represent nested, curved, or disjoint regions.
- The result: flexible and powerful transformation of input space.

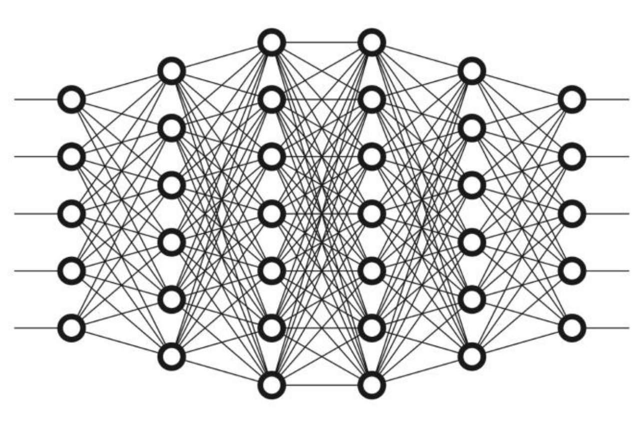




# Step 3: Layer-by-Layer Composition Builds Complexity

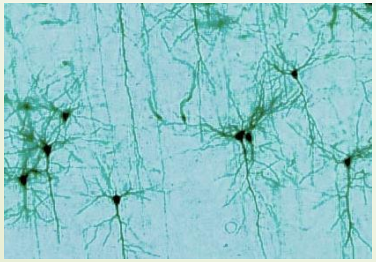
- Each deeper layer receives transformed outputs from previous layers.
- Early layers learn simple features (edges, directions, contrasts).
- Later layers combine them into complex concepts.
- This hierarchical composition allows deep networks to approach universal function approximation.

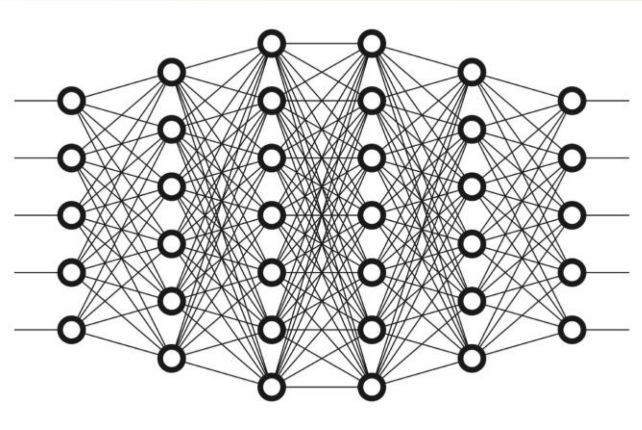




# Summary: Why Hidden Layers Enable Advanced Learning

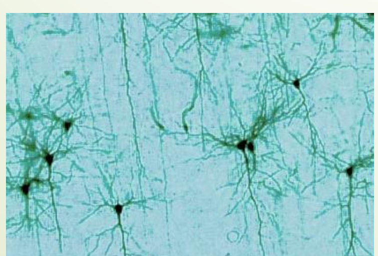
- Multiple linear boundaries + nonlinear activations = flexible regions.
- Hidden layers transform data into linearly separable representations.
- Deep networks can approximate any continuous function.
- Enables learning of complex patterns in vision, language, biology, and more.

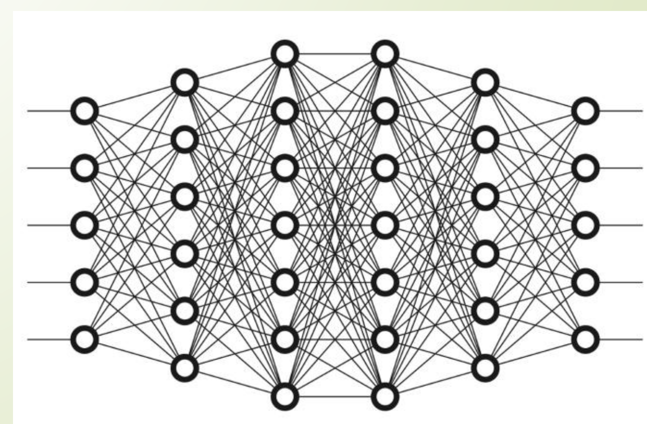




# Where is Knowledge Stored in an ANN?

- Knowledge in an artificial neural network is stored entirely in its weights and biases.
  - Weights encode the strength of connections between neurons
  - Biases shift activation thresholds to improve flexibility
  - Learning = adjusting weights and biases to reduce error
  - Neurons, activation functions, and architecture do NOT store learned knowledge





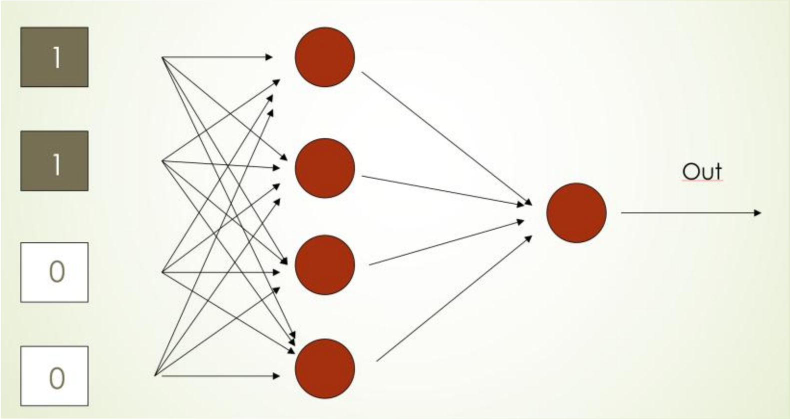
# What about learning?

# Learning Rules in Neural Networks

- Hebbian learning (still relevant conceptually): $\Delta w \propto x_i \cdot x_j$ (unsupervised correlation learning).
- Gradient-based learning (modern standard):
  - Backpropagation: uses chain rule to compute $\partial L/\partial w$ for all weights.
  - Gradient descent update: $\Delta w = -\eta \cdot \partial L/\partial w$.
  - Variants: SGD, Momentum, RMSProp, Adam (most widely used today).
- Perceptron learning rule: historically important, rarely used today except for teaching.
- Genetic algorithms: used in specialized cases (neuroevolution)
- Summary: Modern neural networks are trained almost exclusively with gradient-based optimization + backpropagation.

# Feedforward Network

- Signals flow left → right only
- Input layer feeds hidden layer; hidden feeds output
- No loops or feedback connections
- Used for prediction (classification/regression)
- Computation: output = f(Σ w·x)
- Trained by back propagation and gradient descent



# Backpropagation Phase

- Error signal flows backward from output → hidden → input weights
- Each weight updated based on its contribution to the error
- Arrows in hidden layer show gradient flow
- Allows hidden neurons to adjust internal representations
- Full learning cycle = forward pass + backward pass

# Algorithmic Order of Backpropagation

- 1. Forward Pass: Compute activations layer-by-layer (Input → Hidden → Output).
- 2. Output Error: Compare prediction to target and compute output-layer error.
- 3. Backward Pass Step 1: Propagate error from output layer to previous hidden layer.
- 4. Backward Pass Step 2: Continue propagating error one layer at a time toward the input.
- 5. Gradient Calculation: Compute gradients for each weight using the layer-specific error.
- 6. Weight Update: After all gradients are computed, update all weights simultaneously.

# Gradients in Backpropagation (Chain Rule Overview)

- Forward relations (single neuron): $z = \Sigma\, w_i x_i + b$ , $\hat{y} = f(z)$.
- Interpretation: z is the weighted input; $\hat{y}$ is the neuron's output after activation f.
- Goal: Compute how much changing a weight $w_i$ changes the loss L → $\partial L/\partial w_i$.
- Chain rule: $\partial L/\partial w_i = (\partial L/\partial \hat{y}) \cdot (\partial \hat{y}/\partial z) \cdot (\partial z/\partial w_i)$.
  - $\partial L/\partial \hat{y}$ — Change in loss when the prediction $\hat{y}$ changes (from loss function).
  - $\partial \hat{y}/\partial z$ — Change in neuron output when input z changes (activation derivative).
  - $\partial z/\partial w_i$ — Change in weighted sum z when weight $w_i$ changes (equals input $x_i$).
- Gradient descent update: $w_i \leftarrow w_i - \eta \cdot \partial L/\partial w_i$.
- In multi-layer networks: this gradient computation is applied recursively backward through every layer so each neuron adjusts its weights based on its contribution to the final error.

# From Supervised to Unsupervised Learning

- Supervised learning: model learns from labelled data (input → target).
- Training uses loss function to measure prediction error.
- Backpropagation + gradient descent adjust weights to minimise loss.
- Unsupervised learning: no explicit labels or target values.
- Goal: discover structure or patterns directly from the input data.
- Transition concept: instead of predicting targets, the ANN learns internal structure.

# Backpropagation for Unsupervised Learning

- Backpropagation still works if we provide a suitable objective.
- Common approach: self-supervision (model creates its own learning signal).
- Loss function compares outputs to reconstructed or transformed versions of input data.
- Gradients computed normally → update weights without external labels.
- Thus: backprop is a general optimisation tool, not limited to supervised tasks.

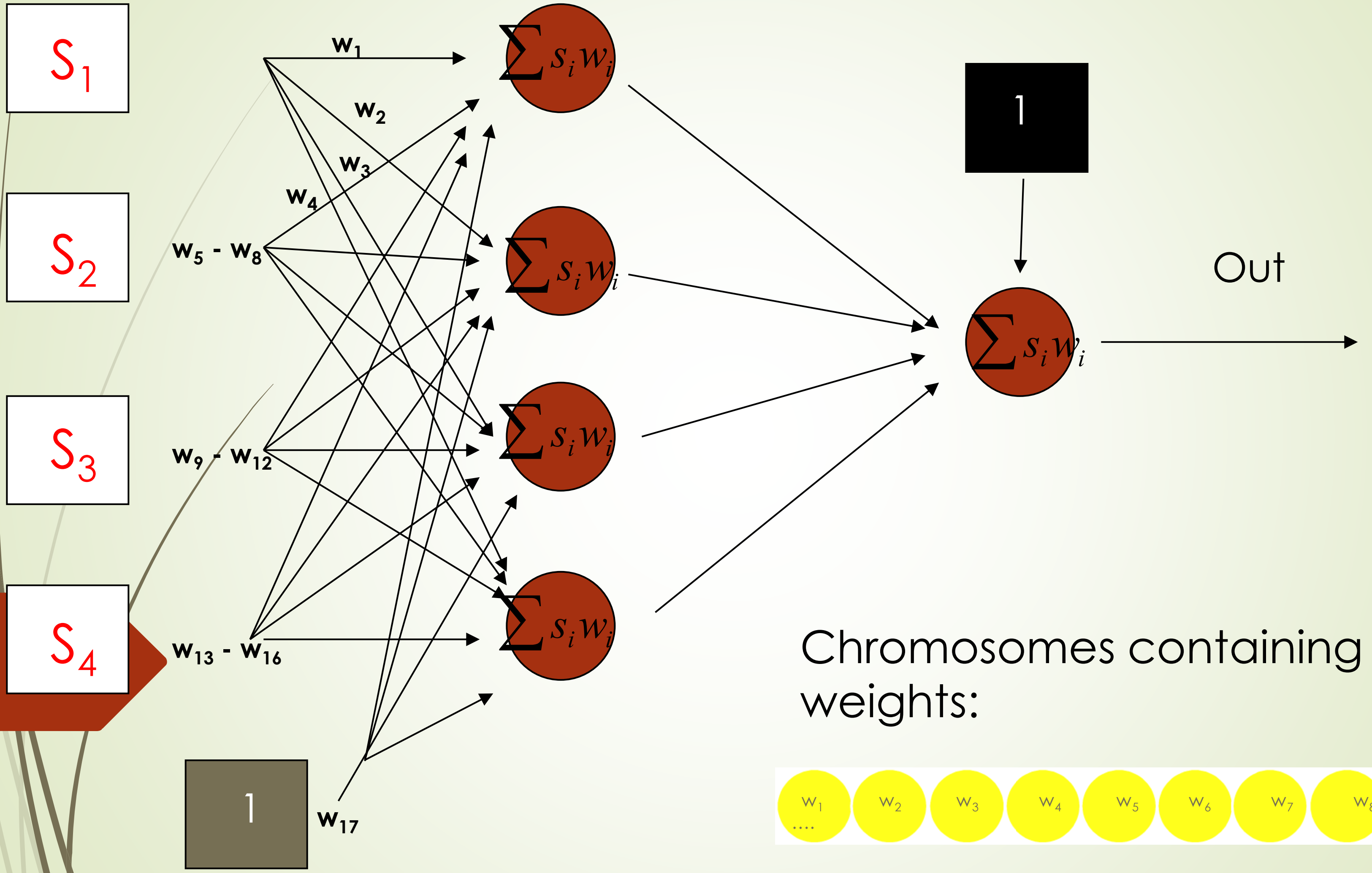
# Hebbian Learning in Unsupervised Systems

- Hebbian rule: “cells that fire together, wire together.”
- Weight update proportional to correlation between neuron activations.
- $\Delta w_{ij} \propto x_i \cdot x_j$ (or variants such as Oja’s rule).
- No loss function: learning emerges from co-activation statistics.
- Useful for feature discovery, clustering, and self-organisation.

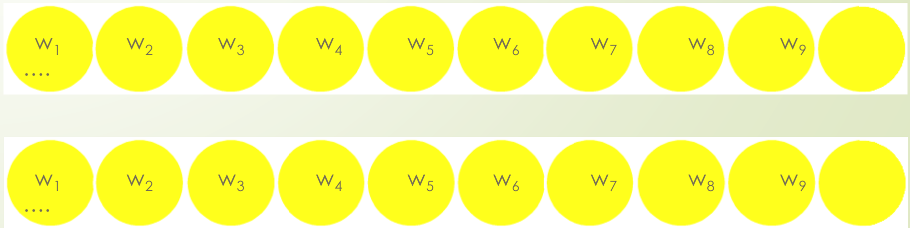
# Applications of Supervised and Unsupervised Learning

- Supervised learning:
-  – Image classification (e.g., crow species).
-  – Speech recognition, translation.
-  – Medical diagnosis, regression tasks.
- Unsupervised learning:
-  – Clustering (grouping similar individuals).
-  – Dimensionality reduction (PCA, autoencoders).
-  – Pattern discovery, anomaly detection.
- Both approaches complement each other in modern ML systems.

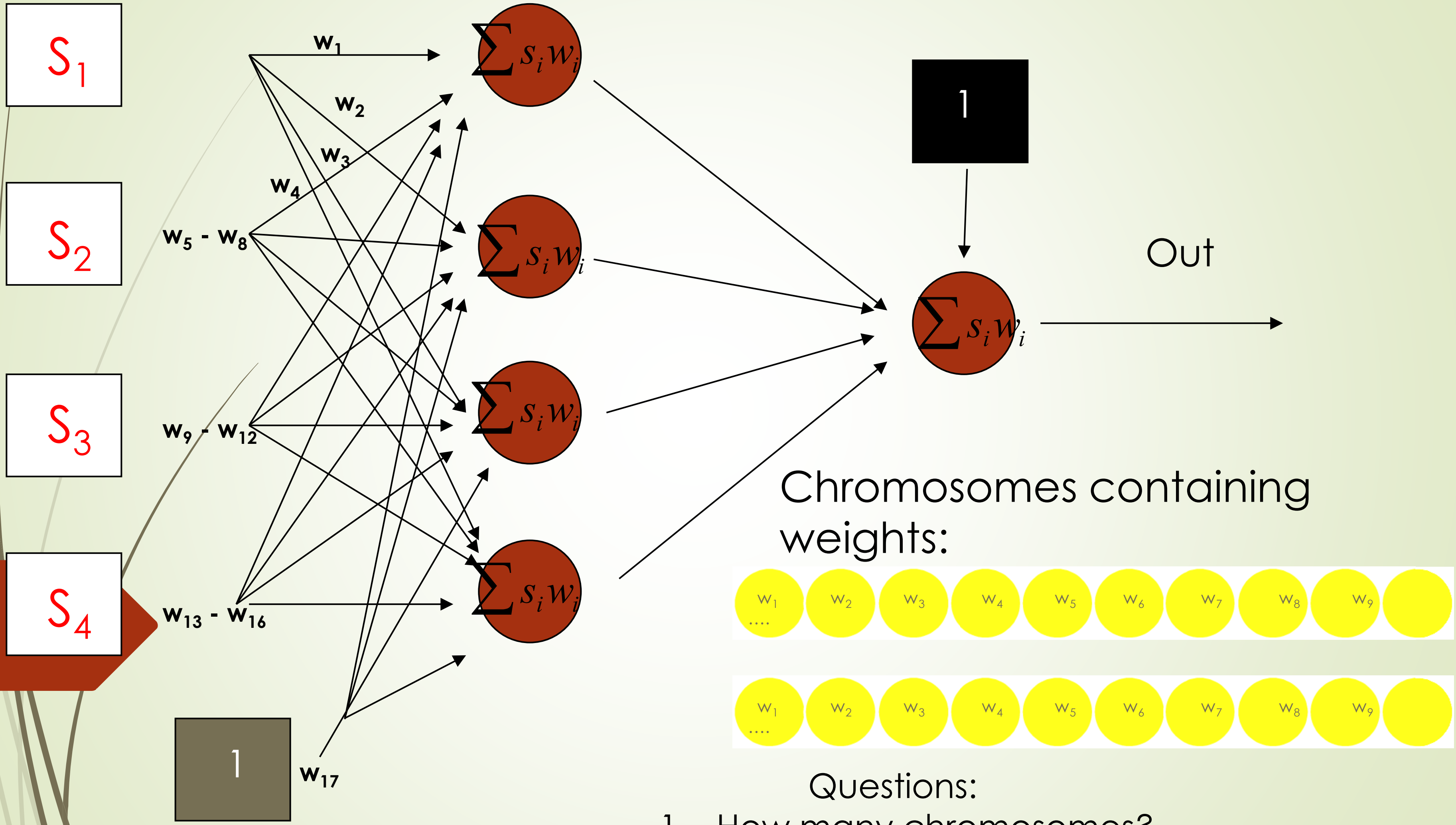
# Extra: Neural network optimised by genetic algorithm:




Chromosomes containing weights:



# Extra: Neural network optimised by genetic algorithm:




Chromosomes containing weights:

Questions:

1. How many chromosomes?
2. How many genes in each?
3. What type of reproduction?