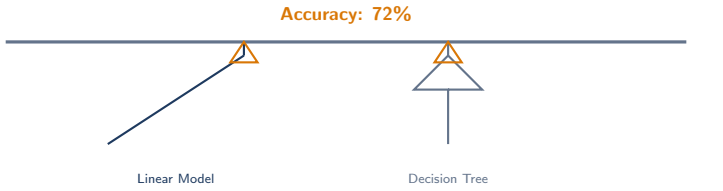
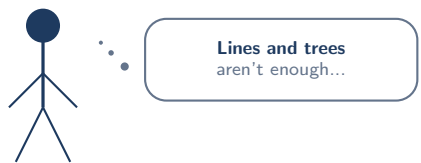
Deep Learning: The Big Idea

Data Science with Python – BSc Course

25–30 Minutes


Accuracy: 72%
Linear Model
Decision Tree

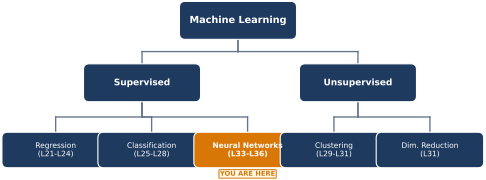

Lines and trees
aren't enough...

When Simple Models Fail

Beyond lines and trees

- Linear regression and decision trees work well for simple patterns
- Complex relationships (images, text, nonlinear interactions) need more power
- **Deep learning** uses **neural networks** (layers of simple computing units wired together) for complex patterns

Read the chart: Deep learning sits inside ML, inside AI. It is a specialization, not a replacement.



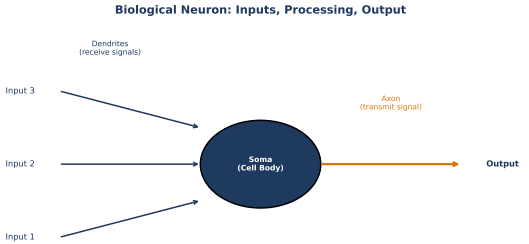
A neural network with 3 layers learns to detect fraudulent trades from 50 raw features.

From Biology to Math: The Neuron

Biological inspiration

- Biological neurons receive signals, process them, fire if threshold is met
- Artificial neurons: weighted sum of inputs, pass through an **activation function** (a rule that decides whether the neuron "fires")
- Same principle: aggregate evidence, then decide

Read the chart: Left: biological neuron with dendrites and axon. Right: mathematical model with weights and activation.



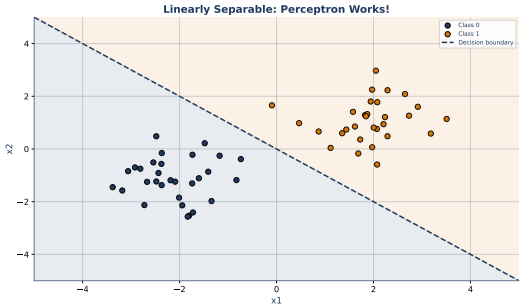
Inputs [0.6, 0.3], weights [0.5, 0.8], sum = 0.54; activation fires if sum ¿ 0.5.

The Perceptron: Simplest Neural Network

One neuron, one boundary

- One neuron, one linear decision boundary
- Can separate data that is **linearly separable** (a straight line divides classes)
- Weights determine the boundary orientation; **bias** shifts the boundary position (like an intercept)

Read the chart: The line separates two classes. Points above = class 1, below = class 0.



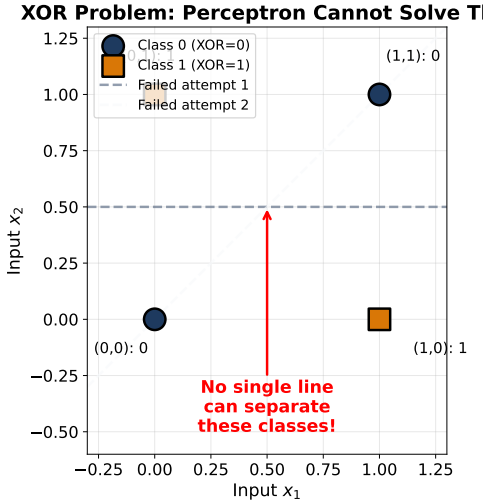
Weight = 0.7 on volume, weight = 0.3 on price, bias = −0.5 classifies buy/sell.

The XOR Problem: One Neuron Is Not Enough

The limitation

- XOR: output is 1 when inputs differ, 0 when they match
- No single straight line can separate the two classes
- This limitation motivated the development of multi-layer networks

Read the chart: The four XOR points cannot be split by any single straight line. Try drawing one – impossible.



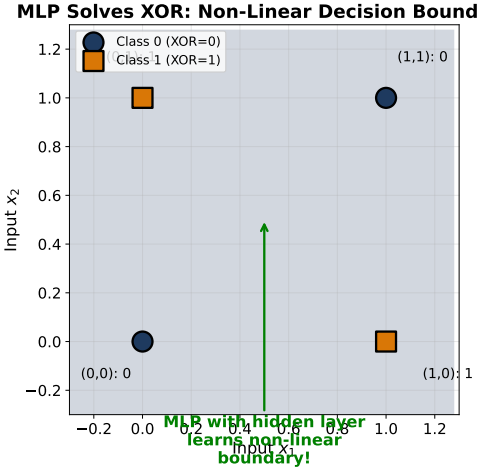
XOR cannot be solved by a single perceptron – we need multiple layers.

Stack Layers: The Multi-Layer Perceptron

Depth solves XOR

- Add **hidden layers** (layers between input and output that are not directly observed) between input and output
- Each layer learns increasingly abstract features
- Even one hidden layer can approximate any continuous function

Read the chart: Two hidden neurons create two lines. Together they carve out the XOR region.



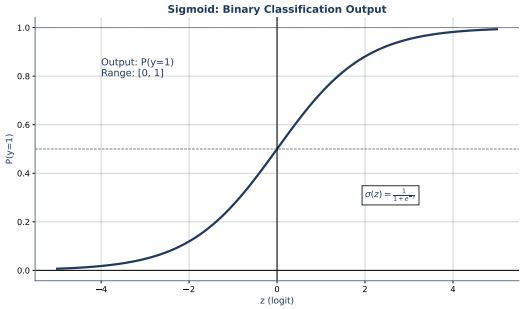
Input (10 features) → hidden (64 neurons) → hidden (32 neurons) → output (1).

Activation Functions: Adding Nonlinearity

Why activation matters

- Without activation functions, stacked layers collapse to a single linear layer
- Sigmoid squashes output to (0, 1) – useful for probabilities
- **ReLU** (output = max(0, input)) is the modern default: fast, avoids **vanishing gradients** (signal shrinking to near-zero in deep layers)

Read the chart: The sigmoid curve squashes any input to [0,1]. Flat tails cause vanishing gradients.



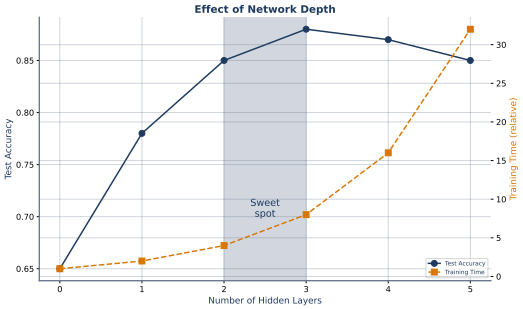
ReLU(2.5) = 2.5, ReLU(−1.3) = 0. Sigmoid would squash both toward 0.5.

Depth vs Width: How to Size a Network

Architecture decisions

- Deeper networks learn hierarchical features (edges, shapes, objects)
- Wider networks memorize more but may not generalize well
- Start small, increase complexity only if performance demands it

Read the chart: Multiple network sizes compared. Deeper networks reach lower error with fewer total neurons.



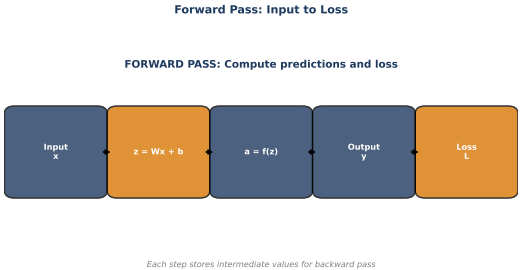
3 layers of 64 neurons outperforms 1 layer of 192 neurons on image tasks.

Forward Pass: Data In, Prediction Out

Making a prediction

- Input data flows through each layer: multiply by weights, add bias, activate
- The final layer produces a prediction (regression value or class probability)
- One forward pass = one prediction

Read the chart: Arrows flow left to right through layers. Each layer transforms the data.



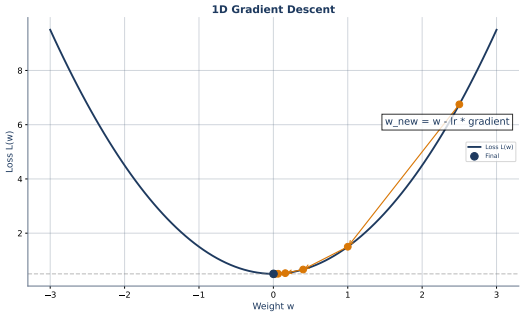
Input [stock price, volume] → layer 1 → layer 2 → output: predicted return.

Gradient Descent: Rolling Downhill to Minimize Loss

Learning from errors

- The **loss function** (e.g. mean squared error) measures how wrong predictions are
- Gradient descent computes the slope and takes a step downhill
- Repeated steps move weights toward the loss minimum

Read the chart: The ball rolls down the curve toward the lowest point. Each arrow is one gradient step.



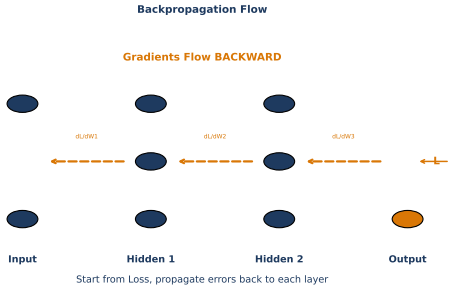
Predicted return = 0.05, actual = 0.02, squared error = 0.0009.

Backpropagation: Error Flows Backward

Computing gradients

- After the forward pass, compare prediction to the true value (loss)
- Backpropagation sends the error signal backward through each layer
- Each weight gets a **gradient** (the slope telling it which direction reduces the loss)

Read the chart: Arrows now flow right to left. Error signal propagates backward through each layer.



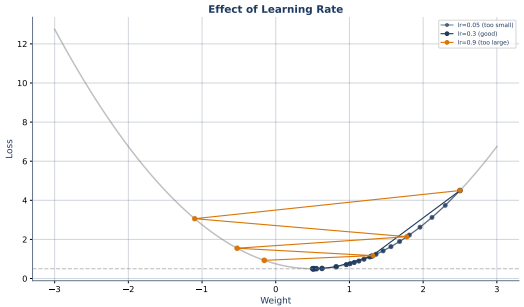
Gradient = -0.3 for weight w_1 means "increase w_1 to reduce the loss."

Learning Rate: The Step Size Tradeoff

Tuning step size

- Too large: overshoots the minimum, loss oscillates or diverges
- Too small: converges painfully slowly, may get stuck
- The learning rate is a **hyperparameter** (a setting chosen before training, not learned from data)

Read the chart: Three curves show convergence at different rates. Middle (0.01) converges smoothly.



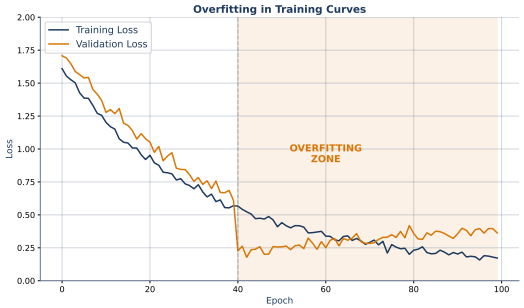
Learning rate = 0.01 converges smoothly; 1.0 overshoots; 0.00001 takes forever.

Overfitting: When the Network Memorizes

The danger of depth

- Training loss drops, but **validation loss** (error on unseen held-out data) starts rising
- The network memorizes training data instead of learning general patterns
- The gap between training and validation performance signals overfitting

Read the chart: Two curves diverge after epoch 50. The widening gap is the overfitting signature.



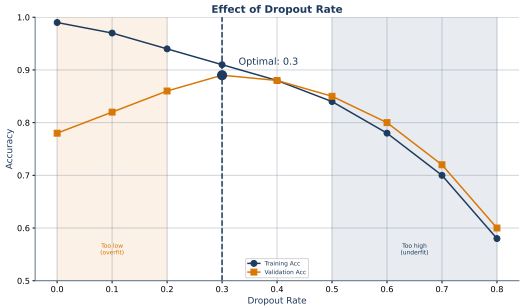
After epoch 50, train loss = 0.01 but validation loss = 0.15 – the model memorized training data.

Dropout: Random Amnesia During Training

Preventing memorization

- Randomly deactivate a fraction of neurons during each training step
- Forces the network to not rely on any single neuron
- Acts like training an **ensemble** (a committee of diverse models whose votes are averaged)

Read the chart: With dropout, training and validation curves stay closer together.



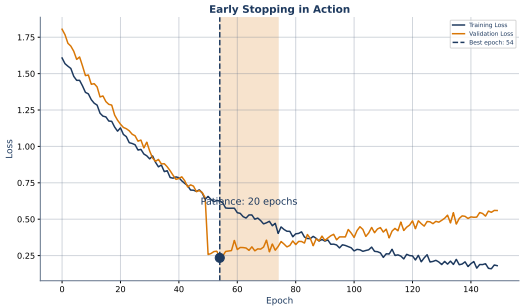
Dropout rate = 0.3 means each neuron has a 30% chance of being switched off each step.

Early Stopping: Quit While You Are Ahead

Knowing when to stop

- Monitor validation loss during training
- Stop training when validation loss stops improving for several **epochs** (one epoch = one full pass through all training data)
- Saves the best model weights before overfitting takes hold

Read the chart: The vertical dashed line marks the best epoch. Training continued but the model got worse.



Best model at epoch 42; stopped at epoch 52 after 10 epochs of no improvement.

When to Use Deep Learning

Criterion	Traditional ML	Deep Learning
Data size needed	Hundreds–Thousands	Thousands–Millions
Feature engineering	Manual, domain-driven	Automatic, learned
Interpretability	High (coefficients, splits)	Low (black box)
Compute cost	Low (CPU fine)	High (GPU recommended)
Tabular data	Often better	Comparable
Images / text / audio	Limited	Excels
Training time	Minutes	Hours–Days

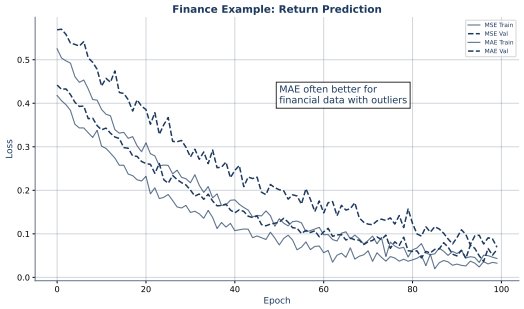
Feature engineering = manually creating inputs (e.g. moving averages). Deep learning learns features automatically.

Finance: Return Prediction with Neural Networks

Deep learning in finance

- Feed in technical indicators, macro variables, sentiment scores
- The network discovers nonlinear interactions traditional models miss
- Challenge: financial data is noisy – regularization is critical

Read the chart: Predicted vs actual returns. The scatter around the diagonal shows prediction quality.



Neural networks can model complex return patterns, but overfitting on noisy financial data is the key risk.

Key Takeaways

1. A single perceptron draws a line – stacking layers creates **nonlinear** boundaries
2. Activation functions inject the nonlinearity that makes depth meaningful
3. Forward pass predicts, backpropagation learns – the learning loop
4. Learning rate is the most important hyperparameter to tune first
5. Overfitting is the central challenge – use dropout and early stopping
6. Deep learning excels on large datasets with complex, unstructured patterns

Deep learning extends ML with layered architectures that learn features automatically.

Accuracy: 94%