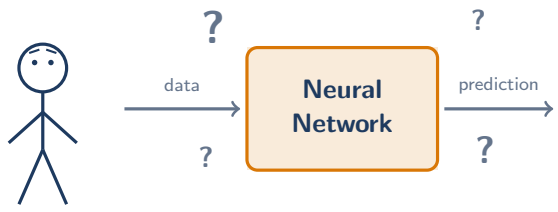
Deep Learning: The Complete Picture

Data Science with Python – BSc Course

90–120 Minutes

What's inside the black box?



Learning Objectives

After this session you will be able to:

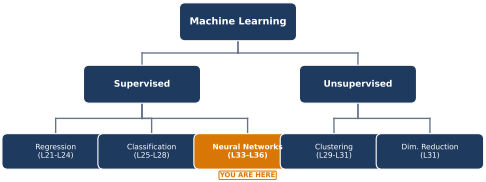
1. **Explain** the perceptron as a single computational neuron
2. **Trace** forward pass computations through a multi-layer perceptron
3. **Compare** activation functions (step, sigmoid, ReLU, softmax)
4. **Interpret** gradient descent and backpropagation intuitively
5. **Evaluate** overfitting prevention strategies (dropout, early stopping, L2)

Covers L33–L36: Perceptron, MLP, Backpropagation, Overfitting Prevention

The Big Picture: Where Deep Learning Fits

- Deep learning = models with **multiple layers** that learn hierarchical representations
- Subset of machine learning, which is a subset of AI
- Key difference: features are **learned**, not hand-engineered

Read the chart: Nested ovals show that DL is a specialisation within ML, not a replacement for it.

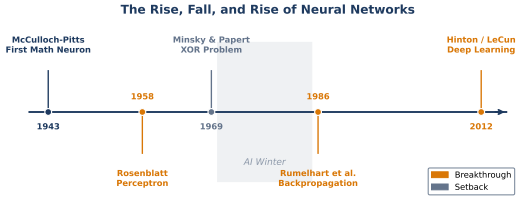


Deep learning automates feature extraction – like a quant fund that discovers its own alpha signals

From Logistic Regression to Deep Learning

- Logistic regression (L25) is a **single neuron** with sigmoid activation
- Deep learning simply **stacks many neurons** into layers
- Same gradient-based optimization, just more parameters

Read the chart: The timeline shows key breakthroughs – from the 1958 perceptron to modern transformers.

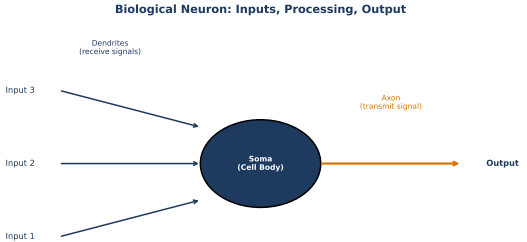


Every deep network is built from the same simple unit you already know

Biological Inspiration

- Neurons receive signals, process them, fire if threshold exceeded — the artificial version is called a **perceptron**
- Artificial neurons mimic this: weighted inputs, activation, output
- The analogy is loose – brains are far more complex

Read the chart: Dendrites receive input, the soma sums it, and the axon fires output – exactly the perceptron flow.

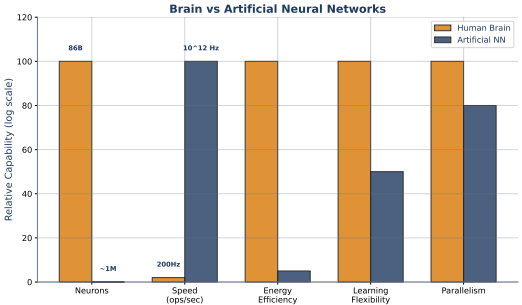


Inspiration, not imitation – like how planes borrow from birds but do not flap wings

Brain vs. Artificial Neural Networks

- Brain: ∼86 billion neurons, massively parallel, low energy
- ANNs: millions of parameters, sequential GPU computation, high energy
- Key similarity: both learn by adjusting connection strengths

Read the chart: The comparison table highlights where biology and silicon diverge most.



ANNs borrow the concept of weighted connections – not the biology

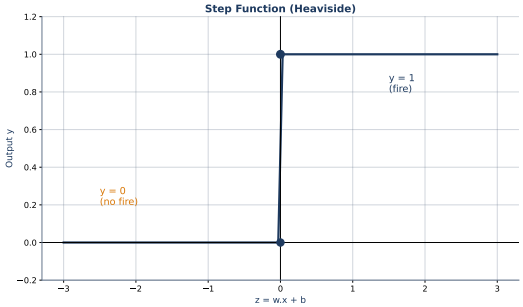
The Step Function

The simplest **activation function** — binary on/off:

$$\hat{y} = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

where $z = \mathbf{w}^\top \mathbf{x} + b$ is the weighted sum plus bias.

Read the chart: The sharp jump at $z = 0$ is the non-differentiable point that breaks gradient descent.



Problem: not differentiable at $z = 0$ – like a light switch with no dimmer

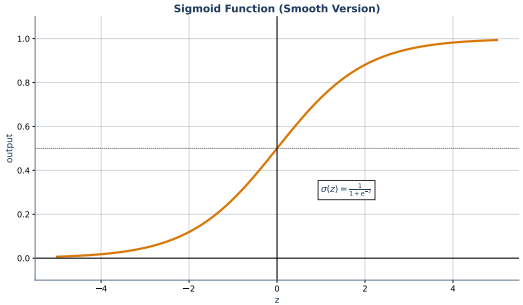
Sigmoid Activation

Smooth, differentiable replacement for the step function:

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

- Output always between 0 and 1 – interpretable as probability
- This is exactly the logistic regression activation from L25

Read the chart: The S-curve squashes any input to $(0, 1)$, with the steepest gradient near $z = 0$.



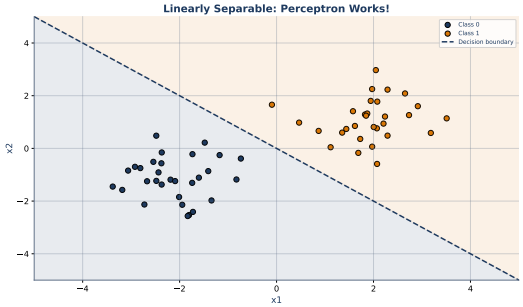
Sigmoid bridges logistic regression and neural networks – same function, deeper architecture

Linear Separability

A single perceptron can solve any **linearly separable** problem:

- Finds a hyperplane that perfectly divides two classes
- Works for AND, OR, NAND gates
- Guaranteed to converge if data is linearly separable

Read the chart: The line cleanly splits blue from orange – this is what one perceptron achieves.

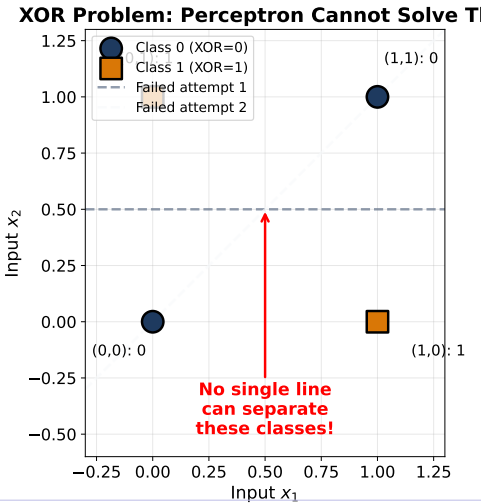


High-return vs low-return stocks split by a single volatility threshold are linearly separable

The XOR Problem

- **XOR** (exclusive or): output 1 when inputs differ – no single line separates the classes
- This limitation motivated multi-layer networks

Read the chart: Try drawing one straight line to separate the two colours – you cannot.

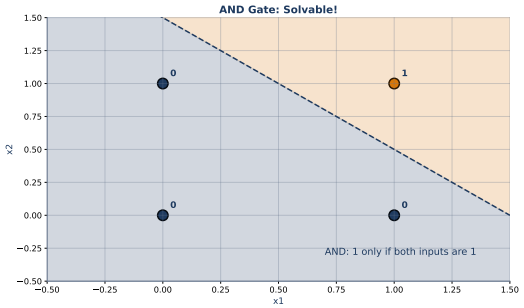


Minsky & Papert (1969) proved this – stalling neural network research for a decade

AND Gate – Solvable by Perceptron

- AND: output is 1 only when both inputs are 1
- Linearly separable – one line does the job
- Weights: $w_1 = 1, w_2 = 1$, bias $b = -1.5$ works

Read the chart: The diagonal line puts the single $(1, 1)$ point above the boundary, all others below.

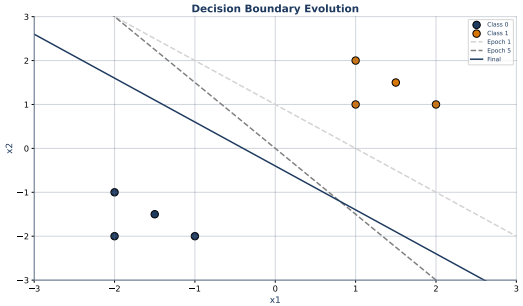


Logic gates are the simplest test cases – like unit tests for a perceptron

Perceptron Learning Rule

- Start with random weights, classify each sample
- If wrong: nudge the decision boundary toward the correct answer
- Watch the boundary evolve over iterations

Read the chart: Each frame shows the boundary rotating and translating as errors are corrected.

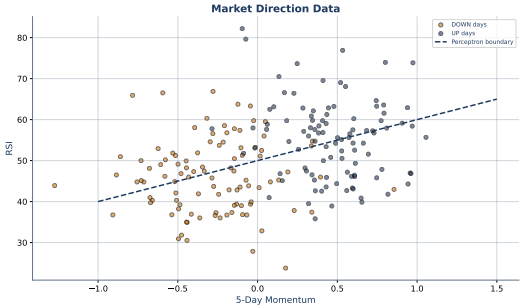


The perceptron learning rule converges in finite steps for linearly separable data

Finance Application: Market Direction Prediction

- Binary classification: will the market go up or down tomorrow?
- Features: previous returns, volume, volatility
- A perceptron draws a single linear boundary in feature space

Read the chart: The scatter shows up/down days in feature space – no clean linear cut exists.

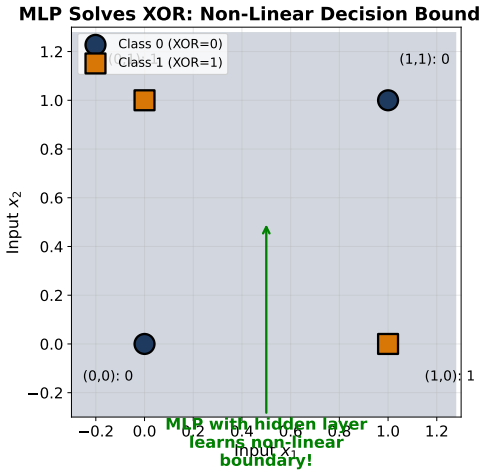


Markets are rarely linearly separable – motivation for deeper models

MLP Solves XOR

- Add a **hidden layer** (neurons between input and output that learn intermediate features): each neuron learns a different linear boundary
- The output neuron combines them into a nonlinear decision region

Read the chart: Two hidden neurons each draw a line; the output neuron intersects them to carve out the XOR region.

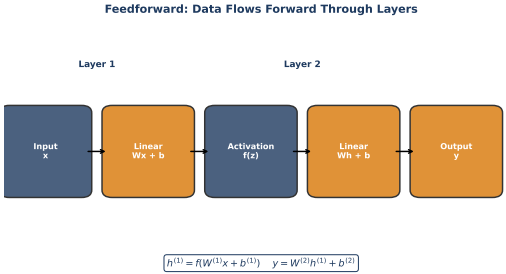


One hidden layer is enough to solve XOR – more layers solve harder problems

Feedforward Computation

- **Feedforward**: data flows in one direction only, input to output, with no loops — left to right: input → hidden → output
- Each layer: $\mathbf{a} = f(\mathbf{W}\mathbf{x} + \mathbf{b})$
- No loops or backward connections (that would be recurrent)

Read the chart: Arrows flow strictly left-to-right through input, hidden, and output layers.

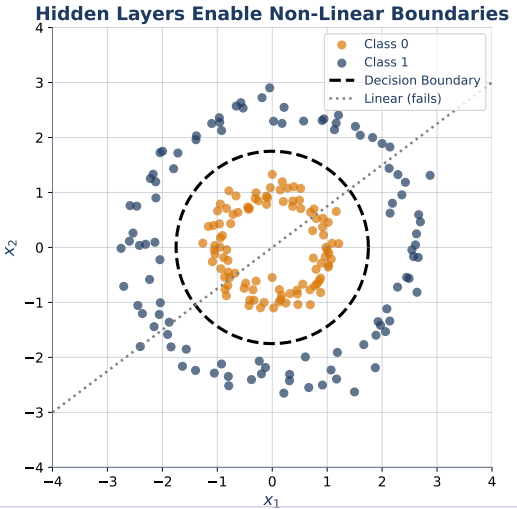


Feedforward = one-directional data flow, the simplest neural network architecture

Why Hidden Layers Matter

- Layer 1: simple patterns; Layer 2: combines them into complex features
- More layers = more abstract, hierarchical representations

Read the chart: Each layer transforms the representation – raw pixels become edges become shapes become objects.

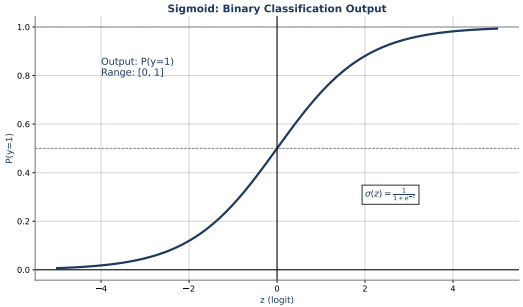


Depth lets networks build representations that shallow models cannot

Sigmoid Output: Binary Classification

- Output layer uses sigmoid when the task is binary (yes/no)
- Output $\in (0, 1)$ interpreted as $P(\text{class} = 1|\mathbf{x})$
- Threshold at 0.5 to get a hard prediction

Read the chart: The network's final sigmoid node outputs 0.82, interpreted as 82% confidence in class 1.



Same sigmoid from L25 logistic regression, now at the output of a multi-layer network

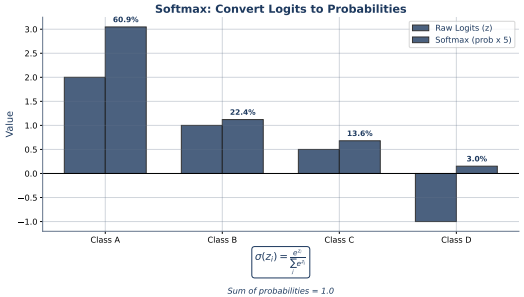
Softmax Output: Multiclass Classification

For K classes, softmax converts raw scores into probabilities:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

- All outputs sum to 1, each $\in (0, 1)$
- Generalizes sigmoid to $K > 2$ classes

Read the chart: Raw logits $[2.0, 1.0, 0.1]$ become calibrated probabilities $[0.66, 0.24, 0.10]$.

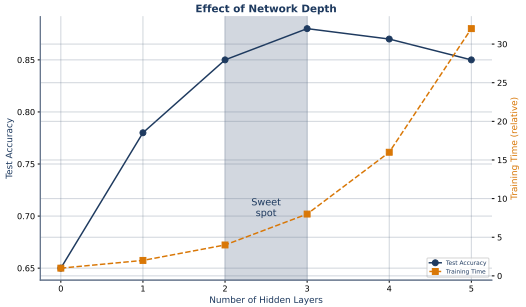


Softmax is how a fund's model assigns probabilities to bull, bear, and sideways regimes

Effect of Network Depth

- Deeper networks can represent more complex decision boundaries
- But deeper also means harder to train (**vanishing gradients**: gradients shrink toward zero in early layers, stalling learning)
- Diminishing returns: 2–3 hidden layers suffice for most tabular data

Read the chart: Accuracy improves with depth up to a point, then plateaus or drops.

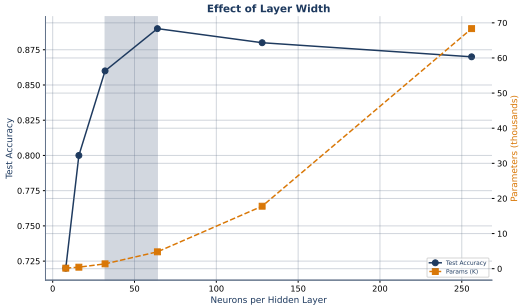


Depth is powerful but must be balanced against trainability – like leverage in finance

Effect of Layer Width

- Width = number of neurons per hidden layer
- Wider layers capture more features per level of abstraction
- Too wide risks overfitting; too narrow loses information

Read the chart: Decision boundaries become smoother as width increases, but the gap between train/test widens.



Width and depth are complementary – both matter for model capacity

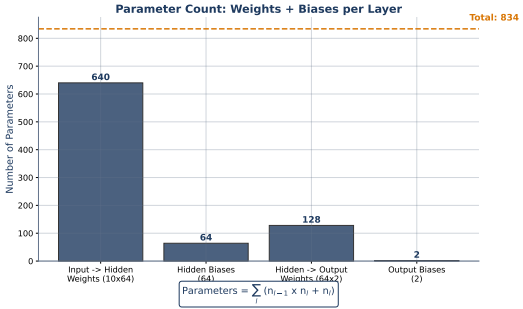
Counting Parameters

For a single fully connected layer:

$$\text{params} = (n_{\text{in}} + 1) \times n_{\text{out}}$$

- n_{in}: number of input features (plus 1 for bias per neuron)
- n_{out}: number of neurons in the layer
- Total network params = sum across all layers

Read the chart: The diagram labels each connection – count them to verify the formula.

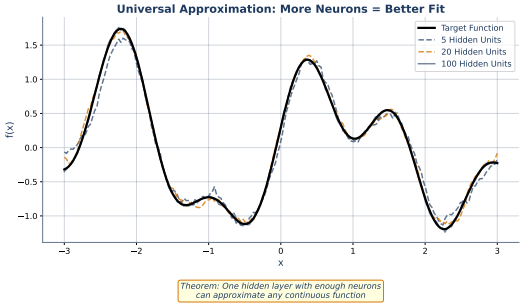


Parameter count determines model capacity and memory – a 3-layer MLP with 64 neurons has ∼5K params

Universal Approximation Theorem

- A single hidden layer with enough neurons can approximate **any** continuous function
- But "enough" may mean billions of neurons – impractical
- The theorem guarantees existence, not learnability

Read the chart: The approximation improves with more neurons, but never reaches perfection in practice.

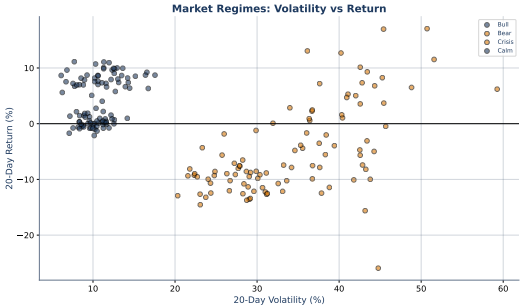


Like saying "enough LEGO bricks can build anything" – true, but the instruction booklet may be impossibly long

Finance Application: Market Regime Classification

- Multiclass problem: bull, bear, or sideways market?
- MLP with softmax output assigns probabilities to each regime
- Features: volatility, volume trends, momentum indicators

Read the chart: Clusters overlap in feature space – the MLP draws curved boundaries between regimes.

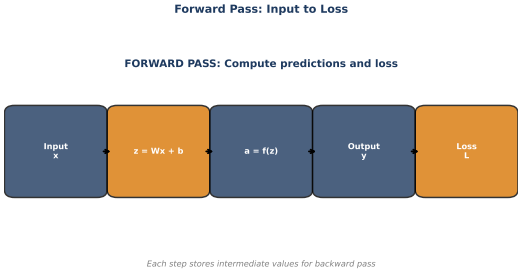


Market regimes are nonlinear – MLPs capture boundaries logistic regression cannot

The Forward Pass

- The **forward pass** is one complete trip through the network: input enters, prediction exits
- Each layer applies: $\mathbf{a} = f(\mathbf{W}\mathbf{a}_{\text{prev}} + \mathbf{b})$
- The final output is compared to the true label to compute loss

Read the chart: Data enters at left, transforms at each layer, and produces a prediction at right.

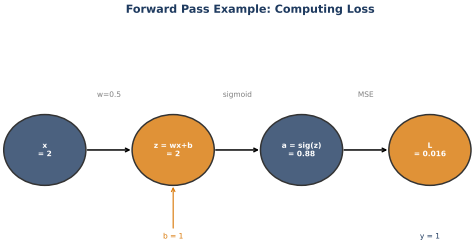


Forward pass = making a prediction; backward pass = learning from the error

Forward Pass: Numerical Example

- Trace concrete numbers through each layer
- Multiply, add bias, apply activation – step by step
- Seeing the math removes the mystery

Read the chart: Follow the numbers: input 0.5 $\times$ weight 0.8 $+$ bias 0.1 $=$ 0.5, then sigmoid(0.5) $=$ 0.62.



Every "intelligent" prediction is just repeated multiply-add-activate

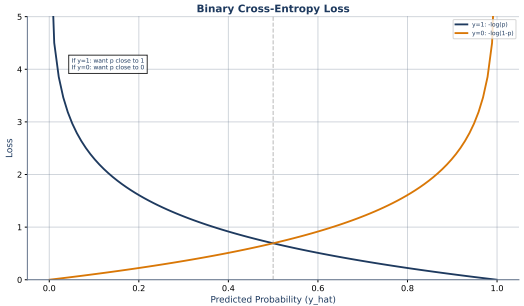
Loss Functions: Measuring Error

Binary cross-entropy for classification:

$$\text{BCE} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)\right]$$

- Penalizes confident wrong predictions heavily
- MSE used for regression; BCE for classification

Read the chart: The loss curve spikes when the model is confident and wrong (near $\hat{y} = 0$ when $y = 1$).



The loss function is the network's report card – a trading model's P&L is its ultimate loss function

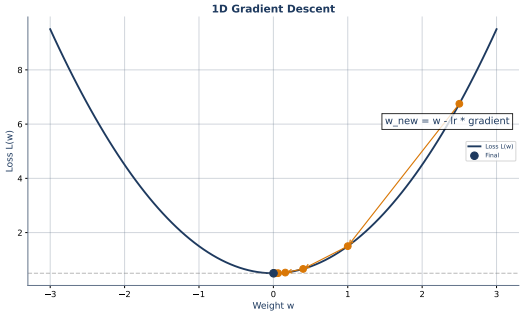
Gradient Descent in 1D

The weight update rule:

$$w_{\text{new}} = w_{\text{old}} - \eta \frac{\partial L}{\partial w}$$

- η = learning rate (step size)
- Derivative = slope of the loss at current position
- Move downhill toward the minimum

Read the chart: The ball rolls down the curve, taking steps proportional to the slope.

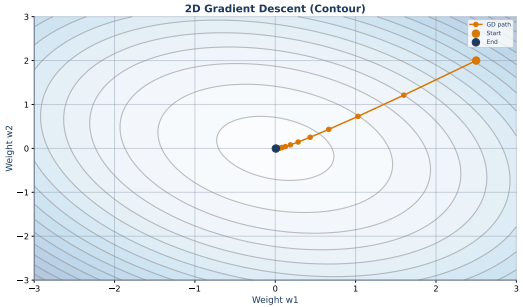


Gradient descent: repeatedly take small steps in the steepest downhill direction

Gradient Descent: Loss Landscape

- With 2 parameters, the loss forms a surface
- Gradient points uphill; we move opposite to it
- Real networks: millions of dimensions, but same principle

Read the chart: Contour lines show equal-loss regions; the path spirals inward toward the minimum.

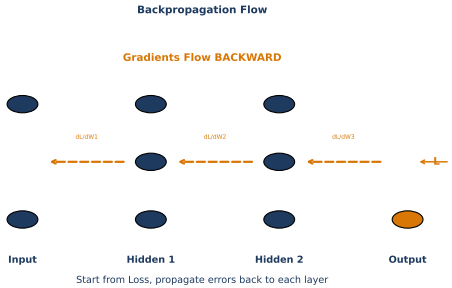


Visualizing 2D helps intuition, even though real landscapes are high-dimensional

Backpropagation: Chain of Blame

- **Backpropagation**: the chain rule applied layer by layer to compute how each weight contributed to the loss
- Error at the output is propagated backward through the network
- Each layer receives its share of the blame via the chain rule
- Derivative = slope: tells each weight how to change to reduce loss

Read the chart: Red arrows flow right-to-left, carrying error signals back through each layer.

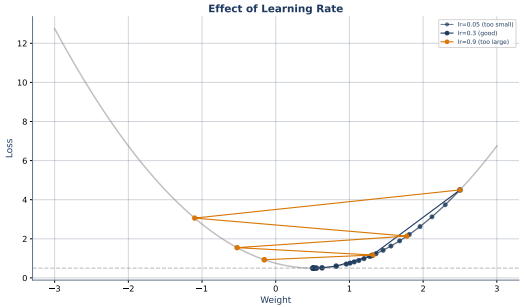


Backprop assigns credit: which weights contributed most to the error?

Effect of Learning Rate

- Too large: overshoots the minimum, loss oscillates or diverges
- Too small: converges very slowly, may get stuck
- Just right: steady decrease toward a good minimum

Read the chart: Three loss curves show diverging (high LR), slow (low LR), and ideal (tuned LR) trajectories.



Learning rate is the single most important hyperparameter – like position sizing in trading

What Good Training Looks Like

- Training loss decreases smoothly over epochs
- Validation loss tracks training loss (no overfitting)
- Convergence: loss plateaus, further training adds little value

Read the chart: Both curves decline together and flatten – the gap between them stays small.

Always monitor both training and validation loss – divergence signals trouble

Finance Application: Return Prediction Training

- Train an MLP to predict next-day stock returns
- Monitor training and validation loss over epochs
- Early stopping prevents memorizing noise in financial data

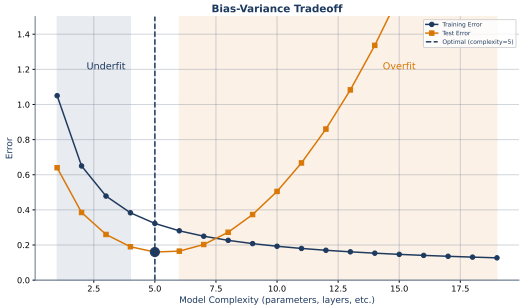
Read the chart: Training loss keeps dropping but validation loss curves up – classic overfitting signature.

Financial time series are noisy – overfitting is the primary risk

Bias-Variance Tradeoff

- **Bias**: error from oversimplified assumptions (underfitting)
- **Variance**: error from sensitivity to training data (overfitting)
- Goal: find the sweet spot that minimizes total error

Read the chart: The U-shaped total error curve has a minimum where bias and variance balance.

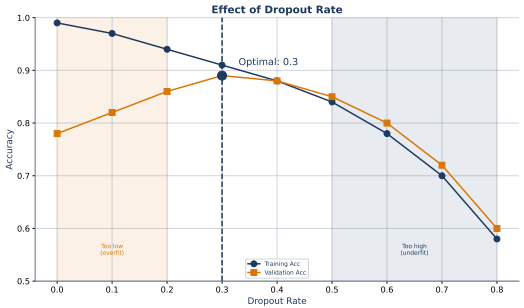


All regularization techniques shift the balance toward less variance – like diversifying a portfolio

Dropout: Random Deactivation

- During training, randomly set neurons to zero with probability p
- Forces the network to not rely on any single neuron
- At test time, all neurons active but outputs scaled by $(1 - p)$

Read the chart: Grey neurons are "dropped" – the remaining ones must compensate, building redundancy.

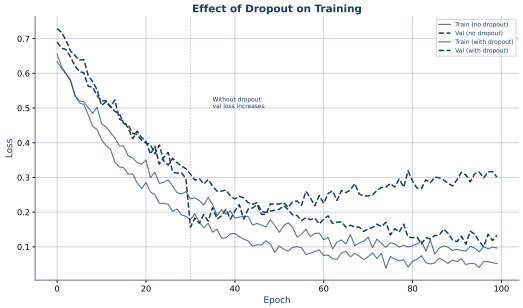


Dropout is like studying with random flash cards – you cannot rely on seeing the same ones each time

Effect of Dropout Rate

- $p = 0$: no dropout, full capacity, risk of overfitting
- $p = 0.2$–0.5: typical range, good regularization
- $p \to 1$: too aggressive, network cannot learn

Read the chart: Validation accuracy peaks at moderate dropout, then drops as too many neurons are silenced.

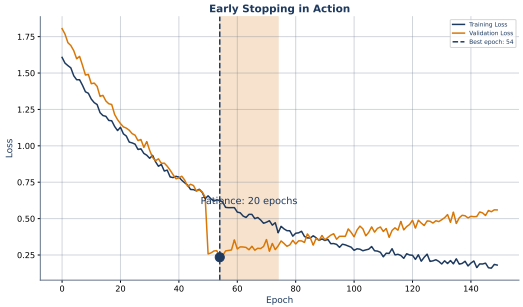


Start with $p = 0.2$ for input layers and $p = 0.5$ for hidden layers

Early Stopping

- Monitor validation loss during training
- Stop when validation loss stops improving (patience parameter)
- Restore the weights from the best epoch

Read the chart: The vertical dashed line marks the optimal stopping point where validation loss is lowest.

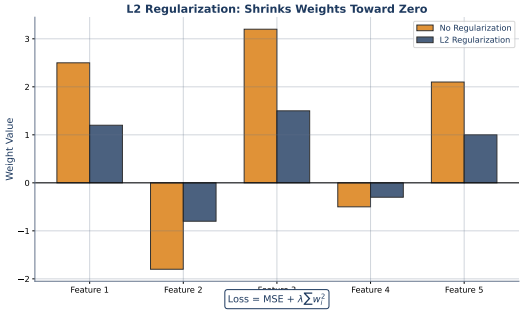


Early stopping is the simplest regularizer – like a stop-loss order on training

L2 Regularization (Weight Decay)

- Add penalty $\lambda \sum w_i^2$ to the loss function
- Encourages smaller weights, smoother decision boundaries
- Same idea as Ridge regression from L22 – applied to neural networks

Read the chart: High λ shrinks decision boundaries toward smooth curves; low λ allows jagged overfitting.

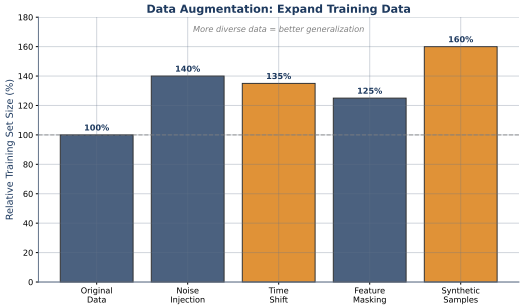


L2 regularization bridges classical statistics and deep learning – same λ tradeoff as Ridge

Data Augmentation for Finance

- Create new training samples from existing data
- Finance: add noise to returns, resample time windows, synthetic scenarios
- More diverse training data reduces overfitting

Read the chart: Original samples (solid) spawn augmented variants (dashed) that fill gaps in the training distribution.

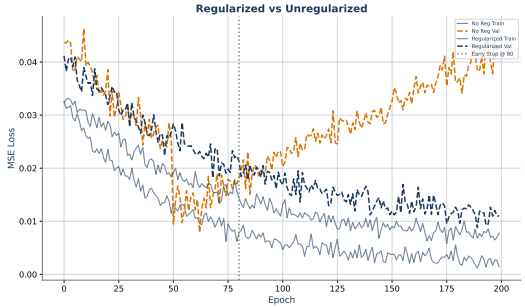


Data augmentation is especially valuable when financial datasets are small – like bootstrapping in statistics

Finance Application: Backtest vs. Live Performance

- Unregularized model: perfect backtest, poor live trading
- Regularized model: slightly worse backtest, robust live performance
- This is overfitting in action – and regularization in action

Read the chart: The unregularized equity curve collapses out-of-sample; the regularized one holds steady.



A strategy that only works on historical data is worthless – the market always moves forward

Traditional ML vs. Deep Learning

Aspect	Traditional ML	Deep Learning
Features	Hand-engineered	Learned automatically
Data needs	Hundreds–thousands	Thousands–millions
Interpretability	High (coefficients)	Low (black box)
Training time	Seconds–minutes	Minutes–hours
Hardware	CPU sufficient	GPU recommended
Tabular data	Often best choice	Competitive, not dominant
Images/text	Needs feature extraction	End-to-end learning
Overfitting risk	Moderate	High (more parameters)

Choose the simplest model that solves your problem – for tabular finance data, XGBoost often beats deep learning

Formula Reference: Perceptron and MLP

Weighted Sum

$$z = \sum_{i=1}^{n} w_i x_i + b = \mathbf{w}^\top \mathbf{x} + b$$

Step Function

$$\hat{y} = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases}$$

Sigmoid

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

ReLU

$$\text{ReLU}(z) = \max(0,\, z)$$

Softmax

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Parameter Count (per layer)

$$\text{params} = (n_{\text{in}} + 1) \times n_{\text{out}}$$

Binary Cross-Entropy

$$\text{BCE} = -\frac{1}{n} \sum \left[y \log \hat{y} + (1-y) \log(1-\hat{y}) \right]$$

All deep learning formulas build on these fundamental operations

Formula Reference: Backprop and Regularization

Gradient Update

$$w_{\text{new}} = w_{\text{old}} - \eta \frac{\partial L}{\partial w}$$

Chain Rule (backprop core)

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_1}$$

Mean Squared Error

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Dropout Mask

$$\tilde{\mathbf{a}} = \mathbf{a} \odot \mathbf{m}, \quad m_i \sim \text{Bernoulli}(1 - p)$$

L2 Penalty

$$L_{\text{total}} = L_{\text{data}} + \lambda \sum_i w_i^2$$

Early Stopping Rule

$$\text{Stop if } L_{\text{val}}^{(t)} > L_{\text{val}}^{(t-k)} \ \forall\ k \in [1, \text{patience}]$$

The chain rule is the mathematical engine behind all neural network learning

Common Misconceptions (1–4)

1. **"Deep = always better"**
   Reality: For tabular data, gradient boosting often outperforms deep learning.
2. **"More layers always help"**
   Reality: Beyond 2–3 hidden layers, gains diminish and training becomes harder.
3. **"Neural networks understand meaning"**
   Reality: They learn statistical patterns, not semantic understanding.
4. **"Backpropagation IS the learning algorithm"**
   Reality: Backprop computes gradients; gradient descent does the actual learning.

Clear thinking about what networks actually do prevents costly investment decisions

Common Misconceptions (5–8)

5. **"Universal approximation = guaranteed solution"**
   Reality: The theorem says nothing about finding the right weights.
6. **"Dropout hurts performance"**
   Reality: It reduces training accuracy but improves generalization.
7. **"Learning rate doesn't matter much"**
   Reality: It is the single most impactful hyperparameter.
8. **"Validation loss always decreases"**
   Reality: It eventually increases – that is overfitting, and why we use early stopping.

Understanding these misconceptions separates practitioners from beginners

Self-Assessment

Q1: A perceptron has weights $[0.5, -0.3]$ and bias 0.1. Given input $[2, 4]$, compute z and the sigmoid output.

Self-Assessment

Q1: A perceptron has weights $[0.5, -0.3]$ and bias 0.1. Given input $[2, 4]$, compute z and the sigmoid output.

$z = 0.5 \times 2 + (-0.3) \times 4 + 0.1 = 1.0 - 1.2 + 0.1 = -0.1$

$\sigma(-0.1) = \frac{1}{1+e^{0.1}} \approx 0.475$ (class 0, since < 0.5)

Q2: A network has layers: input $= 10$, hidden1 $= 64$, hidden2 $= 32$, output $= 3$. Count the total parameters.

Self-Assessment

Q1: A perceptron has weights $[0.5, -0.3]$ and bias 0.1. Given input $[2, 4]$, compute z and the sigmoid output.

$z = 0.5 \times 2 + (-0.3) \times 4 + 0.1 = 1.0 - 1.2 + 0.1 = -0.1$

$\sigma(-0.1) = \frac{1}{1+e^{0.1}} \approx 0.475$ (class 0, since < 0.5)

Q2: A network has layers: input $= 10$, hidden1 $= 64$, hidden2 $= 32$, output $= 3$. Count the total parameters.

Layer 1: $(10 + 1) \times 64 = 704$

Layer 2: $(64 + 1) \times 32 = 2{,}080$

Layer 3: $(32 + 1) \times 3 = 99$

Total: $704 + 2{,}080 + 99 = \mathbf{2{,}883}$ parameters

If you got both right, you understand the core mechanics of deep learning

Not magic. Just math – layer by layer.