Multi-Layer Perceptron Architecture

Neural Networks for Finance

Neural Networks for Finance

BSc Lecture Series

November 30, 2025

Where We Left Off

Module 1 Summary
We learned that a single perceptron:

- Takes weighted inputs
- Applies a threshold
- Outputs a binary decision
- Can only draw **linear** boundaries

The Perceptron Equation:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

The Problem
The perceptron cannot solve XOR or any non-linearly separable problem.

The AI Winter:

- Minsky-Papert (1969) critique
- Funding dried up
- "Neural networks don't work"

Today's Question:
What if we stack multiple perceptrons together?

The perceptron: powerful but limited

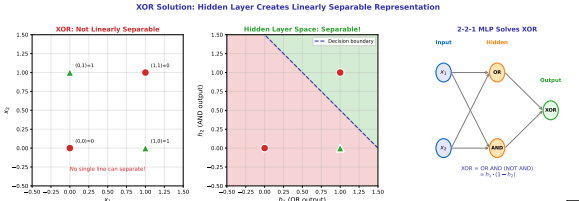
The XOR Problem Revisited

Why One Line Isn't Enough

x_1	x_2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

The Geometry:

- Opposite corners have same label
- No single line can separate them
- We need *multiple* boundaries



../.

xor_solution_ml

Some patterns require more than a single line

The Finance Parallel

Single Analyst (Perceptron)
One junior analyst screening stocks:

- Looks at a few metrics
- Applies simple rules
- Makes direct decisions
- Limited perspective

Limitation:
"Buy if P/E < 15 AND momentum > 0"
This is a single linear rule.

Investment Team (MLP)
A hierarchical team:

- Junior analysts find patterns
- Senior analysts synthesize
- CIO makes final call
- Complex reasoning emerges

Capability:
"Consider value metrics, momentum signals, AND market regime together"
Multiple non-linear patterns.

Key Insight: Hierarchical processing enables complex pattern recognition.

A single analyst sees simple patterns. A team sees complex ones.

Module 2 Roadmap

What We'll Cover

1. **Historical Context**
   - AI Winter survival
   - Backprop rediscovery (1986)
2. **MLP Architecture**
   - Intuition: The firm analogy
   - Math: Matrix notation
3. **Activation Functions**
   - Why non-linearity matters
   - Sigmoid, Tanh, ReLU
4. **Universal Approximation**
   - The fundamental theorem
   - Implications and limits
5. **Loss Functions**
   - MSE for regression
   - Cross-entropy for classification

Learning Objectives:

- Understand MLP architecture
- Master matrix notation
- Know when to use which activation
- Appreciate universal approximation

From single perceptron to universal function approximation

The AI Winter (1969-1982)

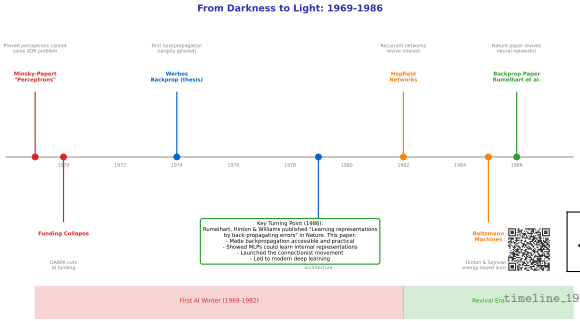
After Minsky-Papert

The neural network winter:

- Government funding cut
- Researchers moved to other fields
- “Connectionism is dead”
- Symbolic AI dominated

The Mood:

- Perceptrons can’t solve XOR
- Multi-layer networks exist but...
- No efficient training algorithm
- Why bother?



../.

After Minsky-Papert, neural network research nearly died

Underground Progress

Paul Werbos (1974)
PhD thesis at Harvard:

- Derived backpropagation
- For general non-linear systems
- Applied to neural networks
- Largely ignored

Why Ignored?

- Published in economics, not CS
- AI winter was at its coldest
- No computational power to test
- No community to spread ideas

Parallel Discoveries

1970s:

- Linnainmaa: automatic differentiation
- Control theory: similar ideas

1980s:

- Parker (1982): rediscovery
- LeCun (1985): independent work
- Rumelhart/Hinton/Williams (1986): fame

Lesson: Good ideas can be discovered multiple times before they "take off."

The key ideas existed but were ignored

1982: Hopfield Networks

John Hopfield
A physicist (not AI researcher) revived interest:

- Connected neural networks to physics
- Energy-based formulation
- Published in PNAS (prestigious)
- Showed neural nets could store memories

The Impact:

- Legitimized neural network research
- Attracted physicists to the field
- New mathematical tools
- Funding started returning

Why Physics Helped

Physics Connection:

- Neurons $\leftrightarrow$ spins in magnets
- Learning $\leftrightarrow$ energy minimization
- Networks $\leftrightarrow$ statistical mechanics

Finance Parallel:
Physicists would later apply similar ideas to:

- Option pricing
- Market dynamics
- Risk modeling
- Quantitative finance

John Hopfield: Physicist rediscovers neural networks

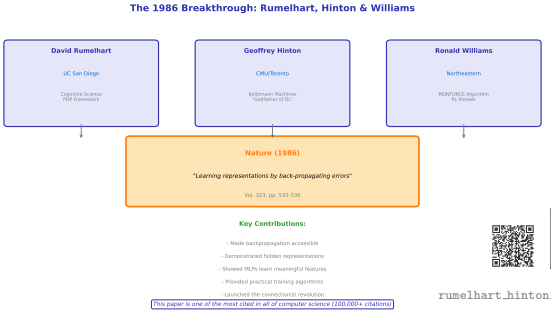
1986: The Backpropagation Paper

The Paper That Changed Everything
Rumelhart, Hinton, Williams in Nature (1986):
"Learning representations by back-propagating errors"

Key Contributions:

- Clear algorithm presentation
- Demonstrated on real problems
- Published in high-impact journal
- Well-communicated to broad audience

The Result:
Neural network renaissance begins.



../.

rumelhart_hinton_william

Nature paper: "Learning representations by back-propagating errors"

What Made 1986 Different?

Werbos (1974)

+ Correct algorithm
+ General framework
- Wrong field (economics)
- No demonstrations
- No community
- No computers

Rumelhart et al. (1986)

+ Correct algorithm
+ Clear presentation
+ Compelling demos
+ High-profile venue (Nature)
+ Growing community
+ Computers available

Lesson for Researchers:
Being right isn't enough. You need:

- The right timing
- The right communication
- The right audience
- The right technology

The right idea at the right time with the right people

Discussion Question

"Backpropagation was discovered multiple times (1974, 1982, 1986). Why do some discoveries get ignored while others take off? What role did timing play?"

Consider:

- Publication venue matters
- Community readiness
- Computational infrastructure
- Demonstration quality
- Today: transformers (2017) exploded
- LSTMs existed since 1997
- What changed?

Think-Pair-Share: 3 minutes

The Neural Network Renaissance

After 1986
Neural networks were back:

- Funding returned
- New conferences (NIPS, now NeurIPS)
- "Connectionism" movement
- Real applications emerged

Key Milestones:

- 1989: LeNet for digit recognition
- 1990s: Speech recognition
- 1990s: Financial applications begin

But Challenges Remained

Not everything worked:

- Deep networks hard to train
- Vanishing gradients
- Limited compute power
- Another "winter" in 2000s

True Revolution: 2012
AlexNet on ImageNet marked the deep learning era. (Module 4)

But first, we need to understand the architecture...

Neural networks are back - and this time they can learn

The Investment Firm Analogy

Hierarchical Decision Making

Level 1: Junior Analysts (Hidden Layer 1)

- Look at raw data
- Find basic patterns
- "This looks like a value stock"
- "This has momentum"

Level 2: Senior Analysts (Hidden Layer 2)

- Combine junior reports
- Higher-level synthesis
- "Value + momentum = quality"

Level 3: CIO (Output Layer)

- Final buy/sell decision
- Combines all analyses
- Single decision point

Key Properties:

1. Information flows upward
2. Each level adds abstraction
3. Later layers see patterns in patterns
4. Final layer integrates everything

This is an MLP!

Hierarchical decision making

Input Layer: The Data Gatherers

The Input Layer
What it does:

- Receives raw data
- One neuron per feature
- No computation
- Just passes data forward

In Finance:

- P/E ratio
- Momentum (returns)
- Volume
- Volatility
- Sector indicators
- Market cap

Notation

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

where:

- n = number of features
- x_i = value of feature i

Example (n=4):

$$\mathbf{x} = \begin{pmatrix} 15 \\ 0.08 \\ 1.2M \\ 0.25 \end{pmatrix} = \begin{pmatrix} \text{P/E} \\ \text{Return} \\ \text{Volume} \\ \text{Vol} \end{pmatrix}$$

The input layer receives raw information

Hidden Layers: The Pattern Finders

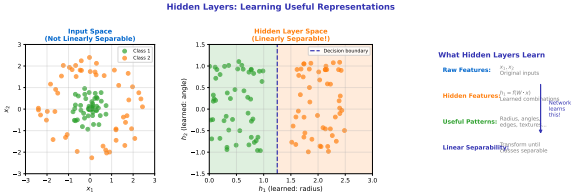
What Hidden Layers Do

They discover intermediate patterns:

- Not explicitly programmed
- Emerge from training
- Often uninterpretable
- But highly useful

Each Hidden Neuron:

- Receives weighted inputs
- Applies activation function
- Outputs a single number
- "Detects" a specific pattern



../.

hidden_layer_representation

"They see things in the data you didn't explicitly ask for"

Finance Example: What Hidden Layers Find

Hypothetical Hidden Neurons

Hidden Neuron 1: "Value Detector"

- Positive weight on low P/E
- Positive weight on high book value
- Activates for value stocks

Hidden Neuron 2: "Momentum Detector"

- Positive weight on recent returns
- Positive weight on volume
- Activates for trending stocks

Hidden Neuron 3: "Risk Detector"

- Positive weight on volatility
- Positive weight on debt
- Activates for risky stocks

The Output Layer

Combines hidden neuron outputs:

$$\text{Buy} = f(w_1 \cdot \text{Value} + w_2 \cdot \text{Momentum} - w_3 \cdot \text{Risk})$$

Key Insight:

We never told the network what "value" or "momentum" means. It *discovered* these concepts from data.

Caveat:

Real hidden neurons may not be this interpretable. They might detect patterns we can't name.

Hidden neurons learn abstract concepts

Output Layer: The Final Decision

The Output Layer
Takes hidden representations and produces:

- Classification: probability of class
- Regression: continuous prediction
- Multiple outputs possible

For Binary Classification:
Single output neuron with sigmoid:

$$\hat{y} = \sigma(w^T h + b)$$

Output $\in (0, 1)$ interpreted as probability.

For Regression:
Single output neuron with no activation (or linear):

$$\hat{y} = w^T h + b$$

Output is predicted value.

Finance Examples

Buy/Sell Classification:

- Output: $P(\text{Buy})$
- If > 0.5: recommend Buy
- If < 0.5: recommend Sell

Return Prediction:

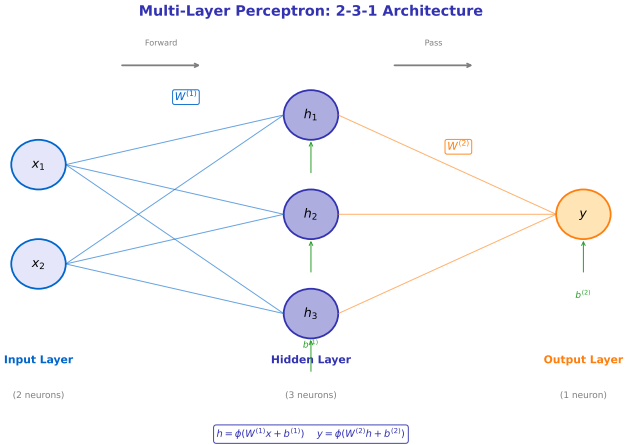
- Output: predicted return
- Could be next-day, next-month
- Continuous value

Multi-Class (Sector):

- Multiple output neurons
- Softmax activation
- Each output = probability of sector

The output layer synthesizes everything into a decision

The Full MLP Architecture



../.

mlp_architecture_2_3_

A complete multi-layer perceptron

Why Are They Called "Hidden"?

We Don't Observe Them Directly
Observable:

- Input layer: the features we provide
- Output layer: the prediction we get

Hidden:

- Internal representations
- Not directly specified
- Learned automatically
- "Hidden" from us

We Don't Tell Them What to Learn

Traditional ML:
"Here are features: P/E, momentum, volume"
We engineer the features.

Deep Learning Philosophy:
"Here is raw data. Find useful patterns."
Network discovers features.

Trade-off:
More automatic, but less interpretable.

Hidden layers discover features automatically

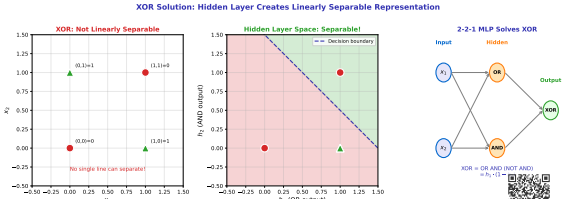
How MLPs Solve XOR

The Two-Hidden-Neuron Solution

Hidden Neuron 1:
Learns: "Is it in the upper-right region?"
$h_1 = \sigma(w_{11}x_1 + w_{12}x_2 + b_1)$

Hidden Neuron 2:
Learns: "Is it in the lower-left region?"
$h_2 = \sigma(w_{21}x_1 + w_{22}x_2 + b_2)$

Output Neuron:
Combines: "If h_1 XOR h_2, output 1"
Each hidden neuron draws *one* line. Together, they create a non-linear boundary.



xor_solution_ml

Multiple decision boundaries working together

Discussion Question

"If hidden layers find features automatically, why do we still need feature engineering in finance?"

Consider:

Arguments for Feature Engineering:

- Domain knowledge helps
- Less data needed
- More interpretable
- Faster training

Reality: In finance, hybrid approaches often work best.

Arguments Against:

- Human biases
- Miss non-obvious patterns
- Deep learning works on raw data
- ImageNet revolution

Think-Pair-Share: 3 minutes

Universal Approximation: The Big Promise

A Remarkable Theorem

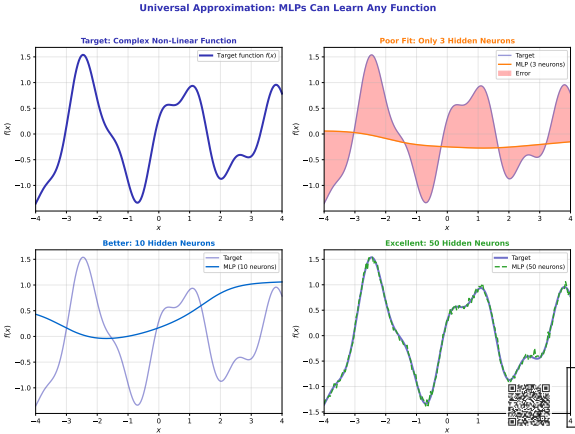
With just *one* hidden layer and enough neurons, an MLP can approximate **any** continuous function to arbitrary accuracy.

Implications:

- MLPs are universal function approximators
- No pattern is too complex (in theory)
- The architecture is not the bottleneck

Caveats:

- "Enough neurons" may be exponential
- Finding the right weights is hard
- Theory vs practice gap



MLPs can learn ANY pattern (in theory)

Now Let's Formalize

What You Already Know
From the intuition section:

- Layers process sequentially
- Each layer transforms its input
- Hidden layers find patterns
- Output layer makes predictions

What's Next

- Matrix notation for efficiency
- Precise forward pass equations
- Parameter counting
- Worked numerical examples

Why Matrix Notation?

Without Matrices:
Write $n \times m$ separate equations for each weight.

With Matrices:

$$\mathbf{h} = f(\mathbf{W}\mathbf{x} + \mathbf{b})$$

One equation captures everything.

Benefits:

- Compact notation
- Efficient computation (GPUs)
- Easier to implement
- Clearer understanding

You understand the intuition. Let's write it precisely.

Matrix Notation: Why Matrices?

Single Neuron (Scalar)

$$h = f(w_1x_1 + w_2x_2 + w_3x_3 + b)$$

As Dot Product:

$$h = f(\mathbf{w}^T\mathbf{x} + b)$$

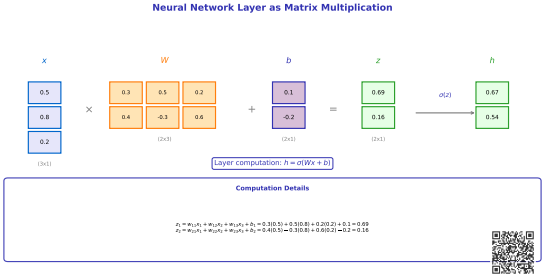
where $\mathbf{w}, \mathbf{x} \in \mathbb{R}^3$

Multiple Neurons (Matrix):

$$\mathbf{h} = f(\mathbf{W}\mathbf{x} + \mathbf{b})$$

where $\mathbf{W} \in \mathbb{R}^{m \times n}$

Each *row* of $\mathbf{W}$ is the weights for one hidden neuron.



../.

matrix_multiplication_visua

Matrices make neural network math elegant

The Weight Matrix

Weight Matrix $\mathbf{W}^{(l)}$
For layer l:

$$\mathbf{W}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$$

where:

- n_l = neurons in layer l
- n_{l-1} = neurons in layer $l-1$

Entry $W_{ij}^{(l)}$:
Weight from neuron j in layer $l-1$ to neuron i in layer l.

Bias Vector $\mathbf{b}^{(l)}$

$$\mathbf{b}^{(l)} \in \mathbb{R}^{n_l}$$

One bias per neuron in layer l.

Example: 4-3 Layer
Input: 4 neurons, Hidden: 3 neurons

$$\mathbf{W}^{(1)} = \begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \end{pmatrix}$$

Size: 3×4 (12 weights)
$\mathbf{b}^{(1)} \in \mathbb{R}^3$ (3 biases)

Each layer has its own weight matrix

Forward Pass: Layer by Layer

One Layer Computation

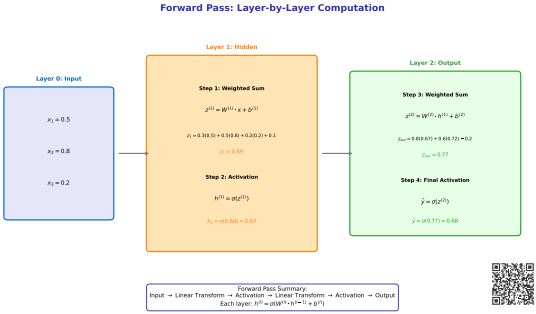
$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$$
$$\mathbf{a}^{(l)} = f(\mathbf{z}^{(l)})$$

where:

- $\mathbf{z}^{(l)}$: pre-activation (weighted sum)
- $\mathbf{a}^{(l)}$: activation (after f)
- $\mathbf{a}^{(0)} = \mathbf{x}$: input

The Steps:

1. Matrix multiply: $\mathbf{W}^{(l)}\mathbf{a}^{(l-1)}$
2. Add bias: $+\mathbf{b}^{(l)}$
3. Apply activation: $f(\cdot)$



../.

layer_by_layer_computatio

Computing outputs one layer at a time

The Complete Forward Pass

For an L-Layer Network

Input:

$$\mathbf{a}^{(0)} = \mathbf{x}$$

Hidden Layers ($l = 1, \ldots, L-1$):

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$$

$$\mathbf{a}^{(l)} = f(\mathbf{z}^{(l)})$$

Output Layer:

$$\mathbf{z}^{(L)} = \mathbf{W}^{(L)}\mathbf{a}^{(L-1)} + \mathbf{b}^{(L)}$$

$$\hat{\mathbf{y}} = g(\mathbf{z}^{(L)})$$

where g may differ from f.

Example: 2-Layer Network

Layer 1 (hidden):

$$\mathbf{z}^{(1)} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$$

$$\mathbf{a}^{(1)} = \text{ReLU}(\mathbf{z}^{(1)})$$

Layer 2 (output):

$$\mathbf{z}^{(2)} = \mathbf{W}^{(2)}\mathbf{a}^{(1)} + \mathbf{b}^{(2)}$$

$$\hat{y} = \sigma(\mathbf{z}^{(2)})$$

Compact Form:

$$\hat{y} = \sigma(\mathbf{W}^{(2)}\text{ReLU}(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)})$$

Chaining layer computations together

Dimensions Matter

Dimension Checking
For $\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$:

$\mathbf{W}$:	$(n_{\text{out}} \times n_{\text{in}})$
$\mathbf{x}$:	$(n_{\text{in}} \times 1)$
$\mathbf{W}\mathbf{x}$:	$(n_{\text{out}} \times 1)$
$\mathbf{b}$:	$(n_{\text{out}} \times 1)$
$\mathbf{z}$:	$(n_{\text{out}} \times 1)$

Rule:
Inner dimensions must match.
$(m \times \mathbf{n}) \times (\mathbf{n} \times p) = (m \times p)$

Example: 4-3-1 Network

Layer 1:

- $\mathbf{W}^{(1)}$: 3×4
- $\mathbf{x}$: 4×1
- $\mathbf{z}^{(1)}$: 3×1

Layer 2:

- $\mathbf{W}^{(2)}$: 1×3
- $\mathbf{a}^{(1)}$: 3×1
- $\mathbf{z}^{(2)}$: 1×1 (scalar)

Common Error: Transposed matrices. Always check dimensions!

Matrix dimensions must be compatible

Worked Example: 2-3-1 Network

Network Setup

Input: $\mathbf{x} = \begin{pmatrix} 0.5 \\ 0.8 \end{pmatrix}$

Layer 1 weights:

$$\mathbf{W}^{(1)} = \begin{pmatrix} 0.2 & 0.4 \\ 0.3 & 0.1 \\ 0.5 & 0.2 \end{pmatrix}$$

$$\mathbf{b}^{(1)} = \begin{pmatrix} 0.1 \\ -0.1 \\ 0.0 \end{pmatrix}$$

Layer 2 weights:

$$\mathbf{W}^{(2)} = \begin{pmatrix} 0.6 & 0.3 & 0.4 \end{pmatrix}$$

$$b^{(2)} = -0.2$$

Forward Pass

Layer 1:

$$\mathbf{z}^{(1)} = \begin{pmatrix} 0.2(0.5) + 0.4(0.8) + 0.1 \\ 0.3(0.5) + 0.1(0.8) - 0.1 \\ 0.5(0.5) + 0.2(0.8) + 0.0 \end{pmatrix} = \begin{pmatrix} 0.52 \\ 0.13 \\ 0.41 \end{pmatrix}$$

$$\mathbf{a}^{(1)} = \text{ReLU}(\mathbf{z}^{(1)}) = \begin{pmatrix} 0.52 \\ 0.13 \\ 0.41 \end{pmatrix}$$

Layer 2:

$$z^{(2)} = 0.6(0.52) + 0.3(0.13) + 0.4(0.41) - 0.2 = 0.315$$

$$\hat{y} = \sigma(0.315) = 0.578$$

Output: 57.8% probability of class 1

Following the numbers through the network

Counting Parameters

Parameters per Layer
For layer l with n_{l-1} inputs and n_l outputs:

Weights: $n_l \times n_{l-1}$
Biases: n_l
Total: $n_l \times n_{l-1} + n_l = n_l(n_{l-1} + 1)$

Network Total:

$$\text{Params} = \sum_{l=1}^{L} n_l(n_{l-1} + 1)$$

Example: 4-10-5-1 Network

Layer 1 ($4 \to 10$):

$$10 \times 4 + 10 = 50$$

Layer 2 ($10 \to 5$):

$$5 \times 10 + 5 = 55$$

Layer 3 ($5 \to 1$):

$$1 \times 5 + 1 = 6$$

Total: 111 parameters

For 100 training samples: < 2 samples per parameter.
Risk of overfitting!

How many weights does your network have?

Discussion Question

"A 4-10-5-1 network has how many parameters? Calculate and discuss: is this a lot or a little for stock prediction?"

Answer: 111 parameters

Consider:

Stock Data Context:

- Daily data: ∼252 days/year
- 10 years = 2,520 samples
- 111 params: 23 samples/param
- Seems okay...

But Also Consider:

- Financial regimes change
- Not all data equally relevant
- Need train/val/test split
- Model complexity vs data size

Exercise: 3 minutes

Finance Example: Multi-Factor Stock Prediction

A Realistic Setup

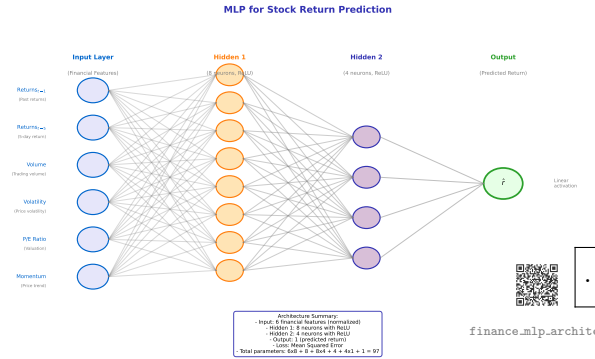
Input Features (10):

- P/E, P/B, EV/EBITDA (value)
- 1m, 3m, 6m returns (momentum)
- 20d volatility (risk)
- Volume ratio (liquidity)
- Sector one-hot (2 features)

Architecture:

- Hidden 1: 20 neurons (ReLU)
- Hidden 2: 10 neurons (ReLU)
- Output: 1 neuron (sigmoid)

Total: 441 parameters



Multiple factors combined through hidden layers