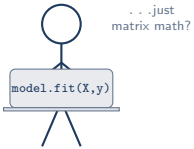
Deep Learning for Finance

Data Science with Python – BSc Course

90 Minutes

Deep Learning: Hype vs. Reality

The headline:
"Deep learning will replace everyone."



The reality:
Linear algebra + calculus + data.

Today we demystify deep learning – what it actually is, when it works, and when it fails.

Learning Objectives

By the end of this lecture, you will be able to:

1. **Explain** how neural networks transform inputs into predictions through layers of weighted sums and activations
2. **Analyze** why depth matters – how hidden layers learn feature hierarchies that linear models cannot
3. **Apply** backpropagation and gradient descent to train a network on structured financial data
4. **Evaluate** when deep learning outperforms classical ML and when it wastes resources
5. **Create** a regularized MLP architecture for a finance classification task with justified design choices

Bloom's taxonomy levels: Remember → Understand → Apply → Analyze → Evaluate → Create

Each objective maps to a section of this lecture. We build from foundations to practical applications.

From Supervised ML to Deep Learning

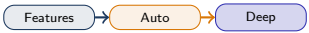
What you already know

- Linear/logistic regression
- Decision trees, random forests
- Train/test split, cross-validation
- Bias–variance tradeoff

These ideas carry over directly.

What this lecture adds

- Neurons as building blocks
- Stacking layers for depth
- Learning features automatically
- Gradient-based optimization



Deep learning is not a separate universe – it extends the supervised learning toolkit you already have.

Can a machine learn patterns that humans cannot define?

Three sub-questions drive this lecture:

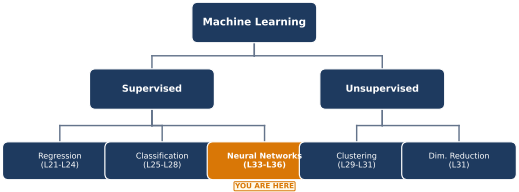
1. **Representation:** How do we encode complex, nonlinear relationships in a model?
2. **Optimization:** How does the model find good parameters among millions of possibilities?
3. **Generalization:** How do we prevent a powerful model from memorizing noise?

Classical ML answers these with feature engineering, grid search, and regularization.
Deep learning answers them with **layers**, **backpropagation**, and **dropout/early stopping**.

Every design choice in deep learning traces back to one of these three challenges.

Where Deep Learning Fits

What you see: ML family tree with deep learning highlighted.



- Deep learning is a **subset** of machine learning, not a replacement
- It excels when data is abundant and features are hard to engineer manually

Deep learning shares evaluation metrics, train/test splits, and regularization with all supervised ML.

Three Domains Where DL Changed the Game

Image Recognition

- ImageNet error: 26% $\rightarrow$ 3%
- Convolutional layers detect edges, textures, objects
- Medical imaging, autonomous vehicles

Language

- GPT, BERT: human-level text
- Transformers process sequences in parallel
- Translation, summarization, chatbots

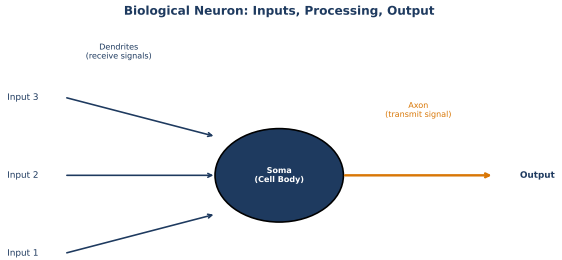
Finance

- Fraud detection in real time
- Sentiment from news text
- Portfolio regime detection
- Limit: interpretability gap

Finance adoption is slower because regulators demand explainability that DL struggles to provide.

Biological Inspiration

What you see: A biological neuron – dendrites receive signals, axon transmits output.



- Artificial neurons are a **loose analogy**, not a faithful model of biology
- Key idea: aggregate inputs, apply threshold, produce output

The brain has ∼86 billion neurons. A large MLP has millions of parameters – still vastly simpler.

When Automation Goes Wrong

Knight Capital Group, August 2012

- Automated trading system deployed with a bug
- Lost **$440 million in 45 minutes**
- Company went from profitable to near-bankrupt overnight

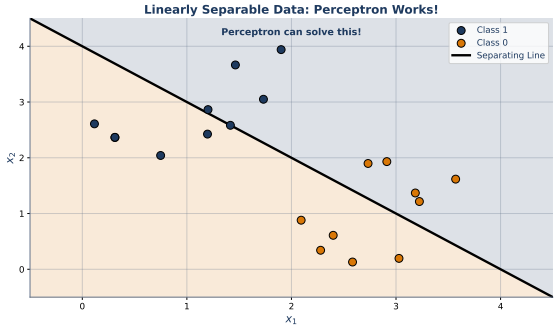
The paradox for DL in finance:

- More powerful models → harder to audit
- Faster decisions → faster failures
- Higher accuracy → higher overconfidence

Power without safeguards is dangerous. This applies to DL models as much as trading algorithms.

What Linear Models Can Do

What you see: Linearly separable data (left) vs. non-separable data (right).

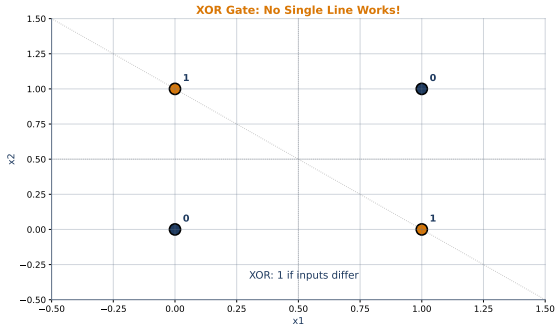


- A single neuron (perceptron) draws one straight decision boundary
- Many real problems are **not** linearly separable

If your data is linearly separable, logistic regression suffices. DL is for when it is not.

The XOR Problem: A Single Neuron Fails

What you see: XOR gate – no single line can separate the classes.

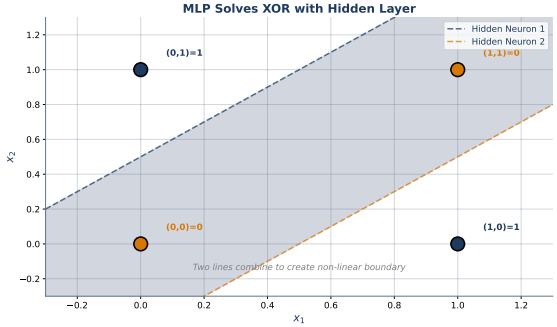


- XOR: output is 1 when inputs differ, 0 when they match
- No linear boundary exists → single perceptron fails completely

Minsky & Papert (1969) proved this limitation, triggering the first "AI winter."

The Fix: Add a Hidden Layer

What you see: A multi-layer perceptron (MLP) solving the XOR problem.



- Hidden layer transforms the space → problem becomes separable
- This is the core idea: **depth creates representational power**

One hidden layer with 2 neurons is enough for XOR. More complex problems need more layers.

Deep Learning: Honest Assessment

Strengths (+)

- Learns features automatically – no manual engineering
- Scales with data: more data $\rightarrow$ better performance
- State-of-the-art on images, text, speech
- Flexible architectures for any input type
- Can capture highly nonlinear patterns

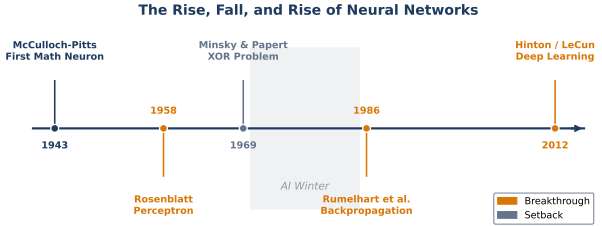
Weaknesses (−)

- Requires large datasets (often $>$10k samples)
- Computationally expensive to train
- Black-box: hard to explain predictions
- Prone to overfitting on small/tabular data
- Hyperparameter-sensitive: architecture, learning rate, regularization

For structured tabular data with $<$10k rows, gradient boosting often outperforms deep learning.

A Brief History of Neural Networks

What you see: Timeline of neural network development – winters and breakthroughs.



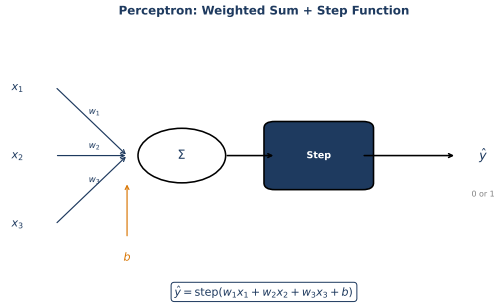
Socratic escalation: Why did it take 70 years from the first neuron model to AlphaGo?

- 1943: Mathematical neuron. *Could we compute with it?*
- 1986: Backpropagation. *Could we train deep networks?*
- 2012: GPUs + big data. *Could we finally afford to?*

Three ingredients aligned in 2012: algorithms (known since 1986), data (internet scale), compute (GPUs).

The Perceptron: Simplest Neural Network

What you see: A single perceptron with weights, bias, and activation.



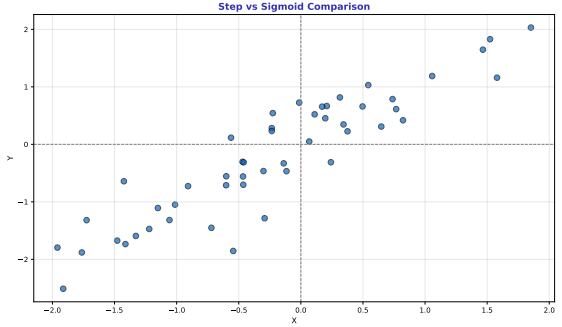
Plain English: Multiply each input by a weight, add them up, add a bias, then decide.

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

This is logistic regression if f is the sigmoid function. You already know this model.

Activation Functions: From Step to Smooth

What you see: Step function (binary) vs. sigmoid (smooth, differentiable).

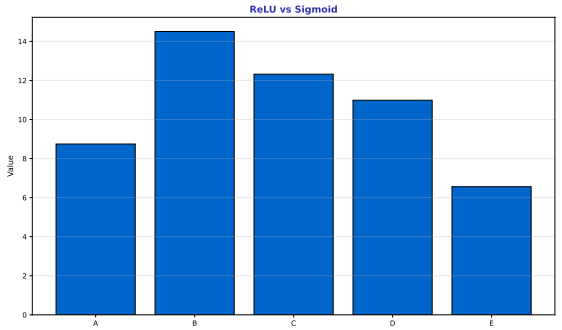


- Step function: all-or-nothing, not differentiable → cannot use gradient descent
- Sigmoid: smooth output in $[0, 1]$ → gradients flow, learning works

Differentiability is the key requirement – without it, backpropagation cannot compute updates.

ReLU: The Modern Default

What you see: ReLU vs. sigmoid – ReLU avoids the vanishing gradient problem.



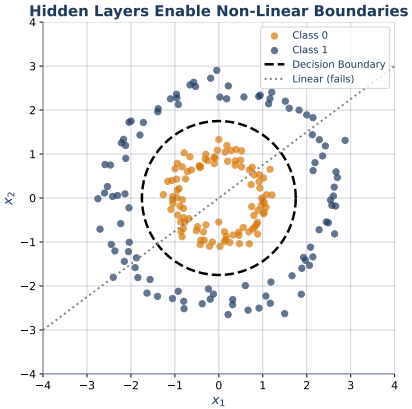
Analogy:

- Sigmoid is a **dimmer switch** – smooth but saturates at extremes (gradient $\to 0$)
- ReLU is a **valve** – fully open above zero, fully closed below (gradient $= 0$ or 1)

ReLU trains 6x faster than sigmoid in deep networks because gradients do not vanish.

Why Hidden Layers Matter

What you see: How hidden layers create increasingly complex decision boundaries.

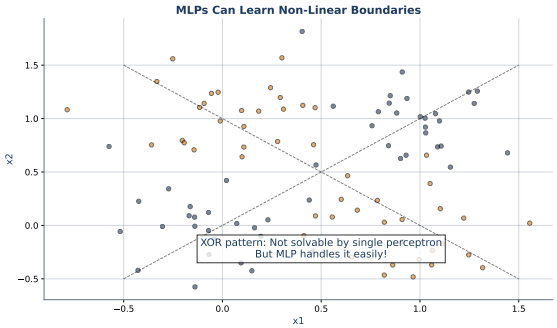


- Layer 1: simple features. Layer 2: combinations. Layer 3+: abstractions.
- Each layer transforms the representation, making the next layer's job easier

Depth lets networks build feature hierarchies – from raw input to high-level concepts.

MLPs Can Learn Anything (In Theory)

What you see: MLP approximating a complex function – universal approximation in action.



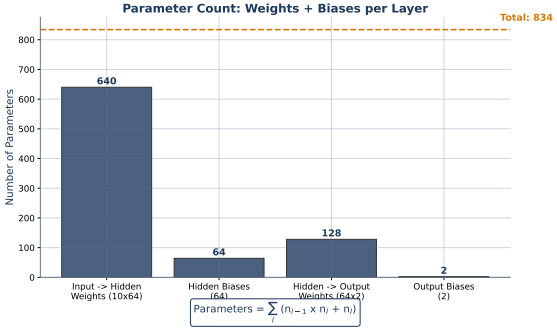
Universal Approximation Theorem (Cybenko, 1989):
A single hidden layer with enough neurons can approximate any continuous function to arbitrary precision.

Caveat: "enough neurons" might mean millions. Depth is more parameter-efficient than width.

The theorem guarantees existence, not that gradient descent will find the solution.

Counting Parameters

What you see: How the number of parameters grows with network architecture.



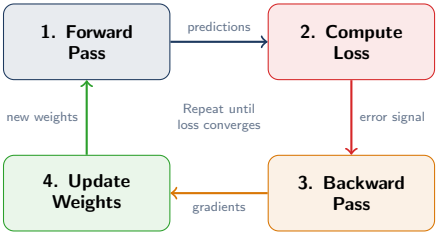
Formula for a dense layer:

$$\text{Parameters} = (\text{inputs} \times \text{neurons}) + \text{neurons (bias)}$$

Example: 100 inputs $\rightarrow$ 64 neurons $= 100 \times 64 + 64 = 6{,}464$ parameters in one layer.

More parameters = more capacity, but also more risk of overfitting. This tradeoff is central.

The Training Loop

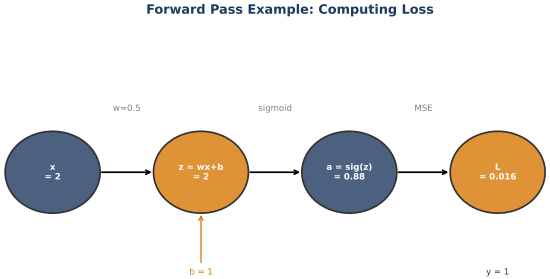


- **Forward:** Input $\rightarrow$ layers $\rightarrow$ prediction
- **Loss:** How wrong is the prediction?
- **Backward:** Which weights caused the error?
- **Update:** Adjust weights to reduce error

Every neural network – from a 3-neuron MLP to GPT – follows this exact same loop.

Forward Pass: A Worked Example

What you see: Numerical example – concrete values flowing through a small network.

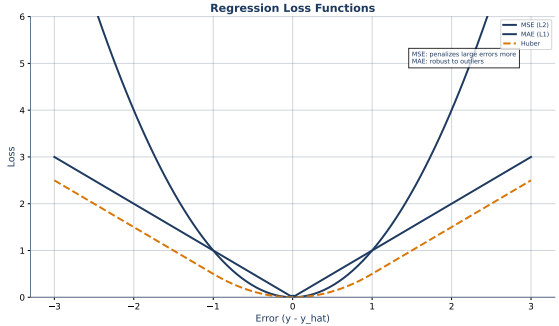


- Each neuron: multiply inputs by weights, add bias, apply activation
- Follow the numbers from left to right to see how prediction emerges

Understanding the forward pass with real numbers removes all mystery from "how neural networks work."

Loss Functions: Measuring Error

What you see: Common regression loss functions – MSE, MAE, and Huber.

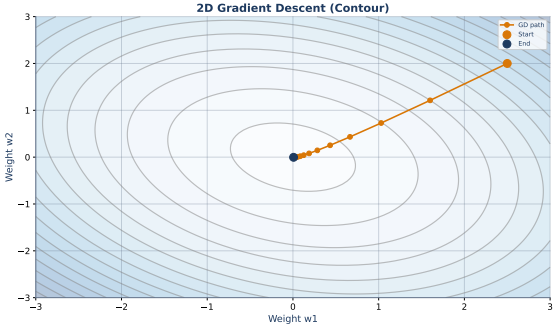


- **MSE:** Penalizes large errors heavily (squared). Standard for regression.
- **Cross-entropy:** Standard for classification (not shown – you know it from logistic regression)

The loss function defines what "good" means. The optimizer finds parameters that minimize it.

Gradient Descent: Walking Downhill in Fog

What you see: 2D loss surface with gradient descent path toward the minimum.



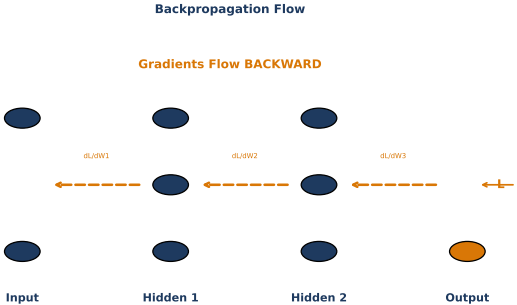
Fog analogy: You stand on a mountain in dense fog. You cannot see the valley.
Strategy: feel the slope under your feet, take a step downhill, repeat.

$$w \leftarrow w - \eta \cdot \frac{\partial \mathcal{L}}{\partial w}$$

η is the learning rate – step size. Too large: overshoot. Too small: takes forever.

Backpropagation: The Chain Rule at Scale

What you see: How gradients flow backward through the network – layer by layer.

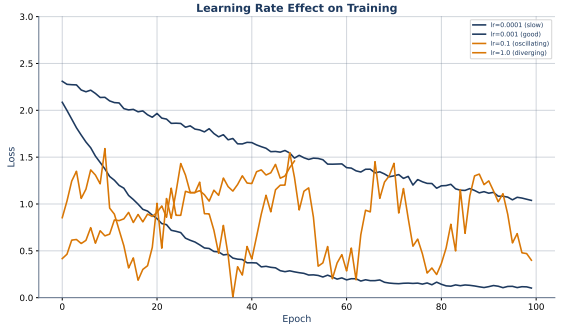


- Chain rule: $\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial a_3} \cdot \frac{\partial a_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial w_1}$
- Each layer passes its gradient to the previous layer

Backpropagation is just the chain rule applied systematically. Nothing more, nothing less.

Learning Rate: The Most Important Hyperparameter

What you see: Effect of different learning rates on training convergence.



- Too small: converges but painfully slowly (wasted compute)
- Too large: oscillates or diverges (never converges)
- Just right: fast convergence to good solution

In practice, use learning rate schedulers that start large and decay. Common default: 0.001.

Think-Pair-Share: Reading a Training Curve

What you see: Training history – loss and accuracy over epochs.

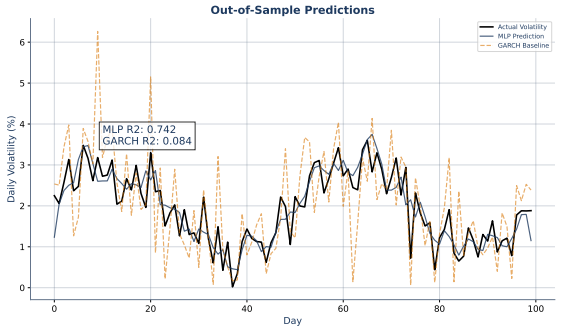
Activity (4 minutes total):

1. **Think** (1 min): At which epoch does training loss plateau? What does this mean?
2. **Pair** (1 min): Compare with your neighbor – do train and validation curves diverge?
3. **Share** (2 min): If validation loss rises while training loss falls, what is happening?

The gap between training and validation curves is your primary overfitting diagnostic.

Does It Generalize? Out-of-Sample Test

What you see: Model predictions on unseen test data – the ultimate test.



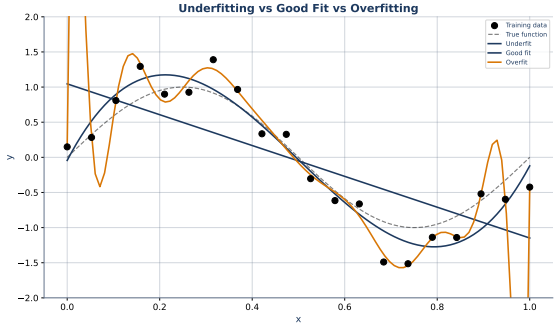
Finance context:

- In-sample R^2 of 0.95 means nothing if out-of-sample R^2 is 0.30
- Financial data is non-stationary: patterns shift over time
- Always evaluate on data the model has **never seen** during training

Backtesting without out-of-sample validation is curve fitting, not model building.

Underfitting: Model Too Simple

What you see: Underfitting (too simple) vs. good fit – the bias–variance spectrum.

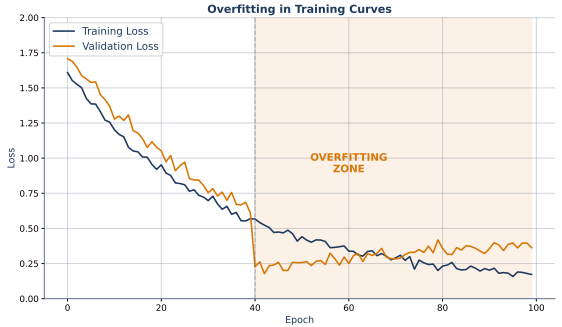


- **Underfitting:** Model cannot capture the underlying pattern (high bias)
- **Good fit:** Model captures the trend without memorizing noise

Solution for underfitting: more layers, more neurons, longer training, or better features.

Overfitting: Model Too Complex

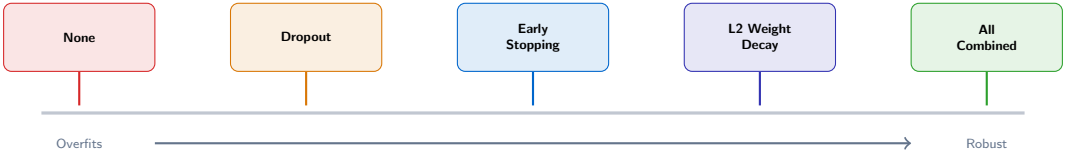
What you see: Overfitting in training – the model memorizes noise.



- Training loss keeps decreasing but validation loss **increases**
- The model is memorizing training data instead of learning general patterns
- Deep networks are especially prone due to high parameter counts

Overfitting is the central practical challenge of deep learning. The next slides address solutions.

Regularization Toolkit

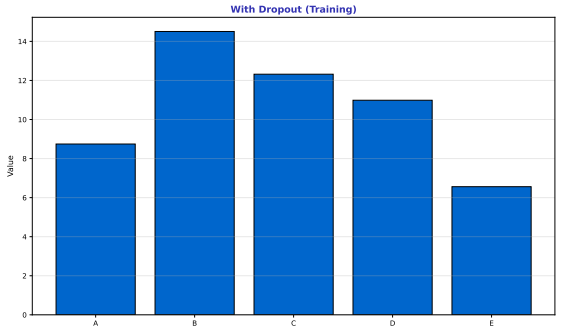


- **Dropout:** Randomly disable neurons during training
- **Early stopping:** Stop when validation loss stops improving
- **L2 decay:** Penalize large weights in the loss function
- **Combined:** Use all three – they complement each other

In practice, always use at least two regularization techniques. Start with dropout + early stopping.

Dropout: Training with Random Amnesia

What you see: Network with dropout – randomly zeroed neurons during training.

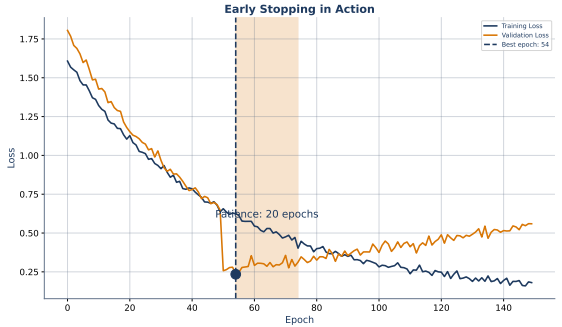


- Each training step randomly drops neurons with probability p (typically 0.2–0.5)
- Forces the network to learn redundant representations – no single neuron is critical
- At test time, all neurons are active (weights scaled by $1 - p$)

Dropout is like training an ensemble of sub-networks simultaneously. Srivastava et al., 2014.

Early Stopping: Know When to Quit

What you see: Validation loss curve with optimal stopping point marked.

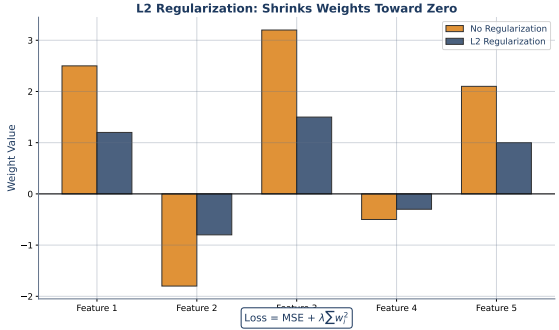


- Monitor validation loss after each epoch
- Stop training when validation loss has not improved for k epochs ("patience")
- Restore the weights from the best epoch

Early stopping is free regularization – it costs nothing and almost always helps.

L2 Regularization: Penalizing Large Weights

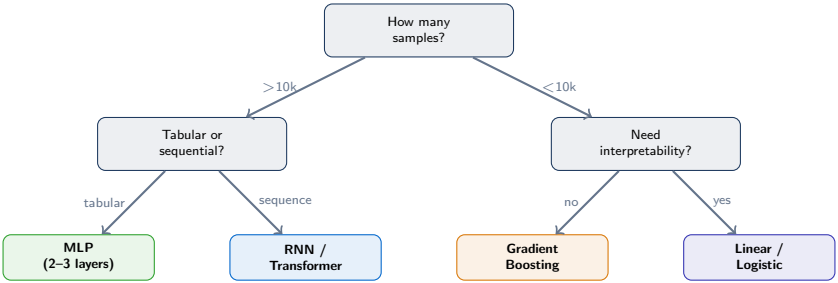
What you see: L2 regularization concept – weight magnitudes constrained.



- Modified loss: $\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{data}} + \lambda \sum w_i^2$
- Large weights $\rightarrow$ large penalty $\rightarrow$ model prefers simpler solutions
- Same idea as Ridge regression – you already know this

λ controls the strength. Too high: underfits. Too low: no effect. Tune via validation.

Choosing an Architecture

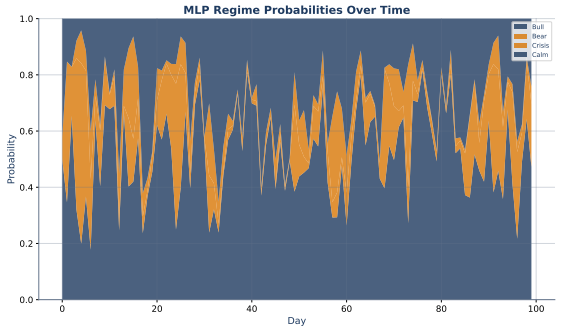


These are starting points, not rigid rules.
Always validate with your specific data.

Start simple. Only move to deeper architectures when simpler models demonstrably fail.

Finance Application: Market Regime Detection

What you see: MLP-predicted regime probabilities over time – bull, bear, and sideways.



- MLP takes rolling features (volatility, momentum, volume) as inputs
- Softmax output layer gives probability per regime
- Portfolio allocation adjusts based on predicted regime

Regime detection is one of the strongest use cases for DL in quantitative finance.

Discussion: Should a Bank Use DL for Credit Scoring?

Scenario: A bank deploys a deep learning model for consumer credit decisions.

The model's performance:

- 95% accuracy (vs. 89% for logistic regression)
- Cuts default losses by 18%
- Processes applications in 0.3 seconds

The problem:

- Cannot explain **why** an applicant was rejected
- EU AI Act classifies credit scoring as "high risk"
- GDPR requires "meaningful explanation" of automated decisions

Discuss in groups (3 minutes):

1. Should the bank deploy this model? Under what conditions?
2. What would you need to make it compliant?
3. Is 6% accuracy worth the legal and ethical risk?

There is no correct answer. The tension between accuracy and explainability is real and unresolved.

When DL Wins and When It Loses

Finance Task	Data Size	Best ML	Best DL	Winner
Credit scoring	50k rows	XGBoost (89%)	MLP (91%)	Tie (explainability)
Fraud detection	10M txns	Random Forest	LSTM (97%)	DL
Portfolio opt.	2k assets	Mean-Variance	Autoencoder	ML
Sentiment	500k texts	TF-IDF + SVM	BERT (94%)	DL

Pattern:

- DL wins with **large, unstructured** data (text, images, sequences)
- ML wins with **small, structured, tabular** data
- Tie when explainability requirements constrain model choice

"Use the simplest model that meets your requirements" – including regulatory requirements.

Industry Impact: Winners and Losers

DL creates value when:

- Data is abundant (>100k samples)
- Features are raw (pixels, text, audio)
- Manual feature engineering is prohibitive
- Real-time inference is needed at scale
- Accuracy improvement justifies compute cost

DL destroys value when:

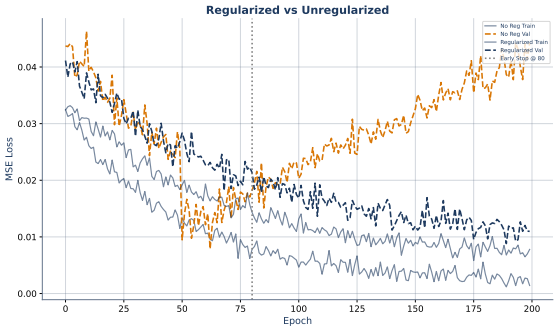
- Data is scarce (<5k samples)
- Features are already well-engineered
- Decisions require regulatory explanation
- Compute budget is limited
- Simpler model achieves similar accuracy

Most finance data is tabular with <50k rows. This is gradient boosting territory.

Knowing when NOT to use DL is as important as knowing how to use it.

The Overfitting Trap in Backtesting

What you see: Regularized vs. unregularized model – out-of-sample reality check.

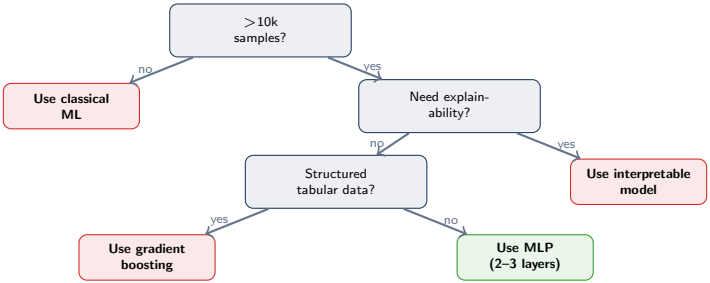


Finance warning:

- An unregularized DL model can "predict" past markets perfectly
- This is not skill – it is memorization of historical noise
- Always compare regularized vs. unregularized on held-out data

If your backtest looks too good to be true, it is. Regularization reveals the model's true skill.

Decision Framework: Should You Use Deep Learning?



Start simple. Escalate complexity only when simpler models fail.

This framework prevents the most common mistake: reaching for DL when simpler tools suffice.

Five Honest Limitations of Deep Learning

1. **Data hunger:** DL needs 10–100x more data than tree-based models to reach comparable performance on tabular tasks. Small datasets mean high variance.
2. **Computational cost:** Training a moderately sized MLP takes minutes; training a transformer takes days on GPU clusters costing thousands of dollars.
3. **Interpretability gap:** A logistic regression gives you coefficients. A random forest gives you feature importances. A deep network gives you a number.
4. **Brittleness:** Small input perturbations (adversarial examples) can flip predictions with high confidence. Financial data is noisy by nature.
5. **Reproducibility:** Non-deterministic GPU operations, random initialization, and sensitivity to hyperparameters make exact replication difficult.

Acknowledging limitations is not weakness – it is how professionals make informed model choices.

The Black Box Problem in Regulated Finance

Regulators require:

- Meaningful explanations for automated decisions (GDPR Art. 22)
- Model risk management documentation (SR 11-7)
- Bias testing and fairness audits
- Human oversight for high-stakes decisions
- Full audit trail of model behavior

DL provides:

- A prediction and a confidence score
- Post-hoc explanations (SHAP, LIME) – approximations, not ground truth
- No guaranteed fairness properties
- Difficulty tracing specific input $\rightarrow$ output logic
- EU AI Act: credit scoring is "high risk"

The gap between what regulators demand and what DL delivers is the main barrier to adoption in banking.

This is not a technical problem waiting for a better algorithm – it is a fundamental architectural property.

Five Takeaways

1. **Neural networks are stacked linear transformations + nonlinear activations.** Nothing more. The power comes from composition, not complexity.
2. **Backpropagation is just the chain rule applied systematically.** It computes how each weight contributed to the error, enabling gradient descent.
3. **Depth beats width for efficiency.** Two layers with 64 neurons each are more powerful than one layer with 128 neurons, with fewer parameters.
4. **Regularization is mandatory, not optional.** Dropout, early stopping, and weight decay are the difference between a model that works and one that memorizes.
5. **Start simple, escalate only when needed.** Logistic regression $\rightarrow$ gradient boosting $\rightarrow$ shallow MLP $\rightarrow$ deep MLP. Skip steps only with good reason.

If you remember nothing else: DL is powerful but not magic. The fundamentals from this course still apply.

What Comes Next

You now understand:

- How neurons compute
- How networks learn
- How to prevent overfitting

The real test:

- Always benchmark against simple baselines
- Complexity must earn its place

Next: Module 8 – NLP and Text Analysis

Deep learning is a tool in your toolkit. The skill is knowing when to reach for it.