# Introduction to Neural Networks

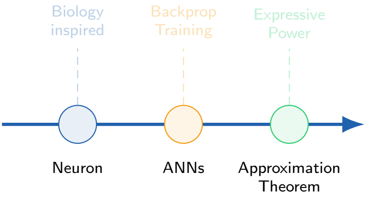
From Perceptron to Modern AI

Leila Kalhor
Shahid Beheshti University

**ICTP, Physics Without Frontiers**
October 2025

# Course Goals & Roadmap

- Understand the concept of an **artificial neuron**
- Grasp the **architecture of neural networks**
- See how ANNs **approximate complex functions**
- Survey key **applications** in AI

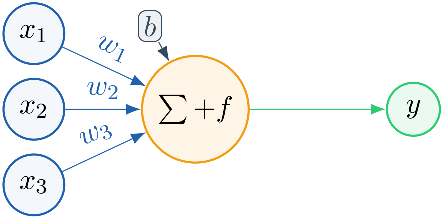



**Takeaway**

By the end of this intro you should be comfortable with the building blocks and why they work.

# Artificial Neuron (Perceptron)

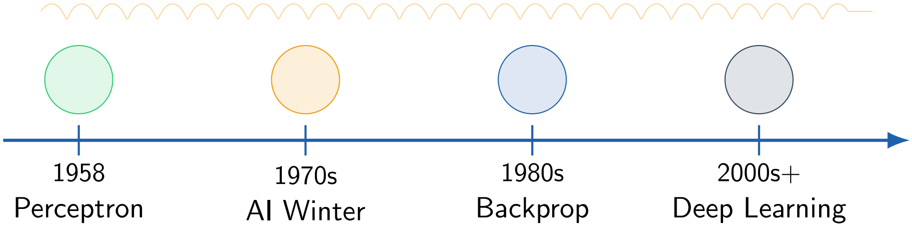
- Inputs $\{x_i\}$ with **weights** $\{w_i\}$ and **bias** $b$
- Linear combination: $z = \sum_i w_i x_i + b$
- Output: $y = f(z)$ where $f$ is an **activation**
- Introduced by **Frank Rosenblatt** (1958)




**Key idea**

Nonlinear activations let simple units build complex decision boundaries.

# A Short History (Visual Timeline)

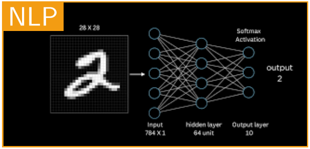



**Context**

Progress accelerated with data, compute (GPUs), and algorithms—enabling today's practical AI systems.

# Modern Applications (Visual Collage)

- **Computer Vision**: classification, detection, segmentation
- **NLP**: translation, question answering, sentiment analysis
- **Robotics/Control**: navigation, manipulation, autonomous driving

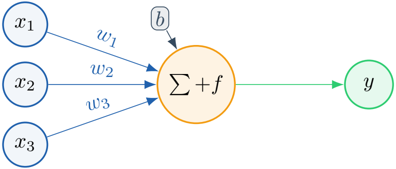




### Message

Neural networks are the *function approximators* behind many intelligent systems.

# Perceptron: Neuron Equation

- Linear part: $z = \sum_i w_i x_i + b$
- Nonlinearity: $y = f(z)$ (activation)
- Before $f$: **linear**; after $f$: **nonlinear**

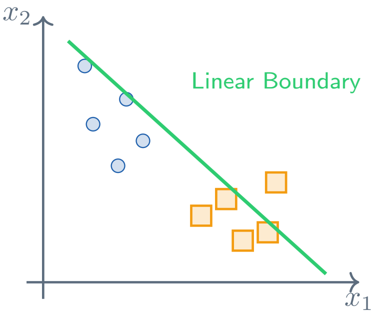



**Key Idea**

Simple units + nonlinear activations ⇒ expressive decision boundaries.

# Activation & Linear Separability

- If $f$ is **linear** $\Rightarrow$ the whole network remains linear.
- We need **nonlinear** activations (ReLU, Tanh, Sigmoid).
- Linear decision boundary works for linearly separable data.

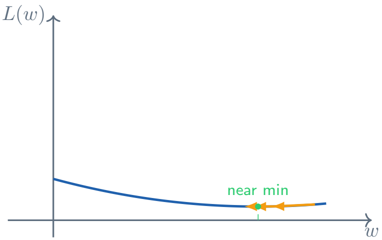



**Takeaway**

Nonlinearity unlocks complex decision surfaces.

# Perceptron Learning: Intuition

- Goal: **reduce loss** by nudging weights downhill.
- Update (intuition): $w \leftarrow w - \eta\, \partial L / \partial w$
- Learning rate $\eta$: too big $\Rightarrow$ oscillation; too small $\Rightarrow$ slow.




**Mental Model**

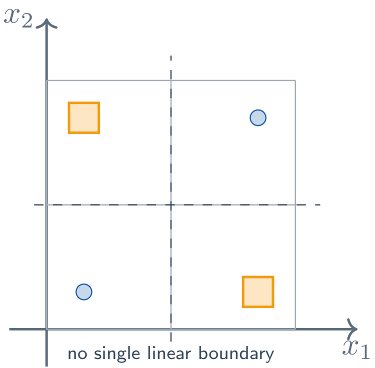
A ball rolling down the loss surface toward a minimum.

# Limitation: XOR is Not Linearly Separable

- Single perceptron fails on XOR — can't separate with one line.
- Motivation for **multi-layer** networks (MLPs).

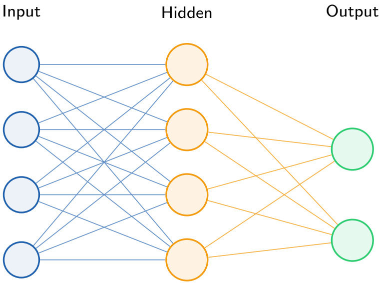
**Message**

Hidden layers combine simple boundaries to solve non-linear problems.



# MLP Architecture (Feedforward)

- Layers: Input → Hidden (nonlinear) → Output
- Nonlinear activations between linear layers
- Depth/width increase representational power

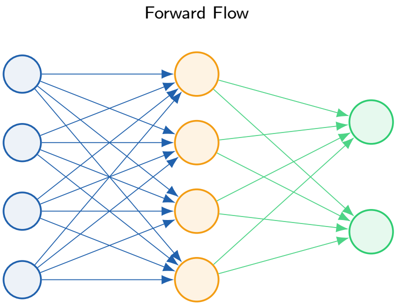



**Idea**

Composition of simple units builds complex features.

# Forward Pass & Loss

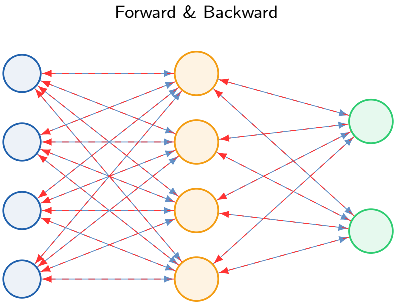
- Forward: $a^{(l)} = f(W^{(l)}a^{(l-1)} + b^{(l)})$
- Loss: Cross-Entropy (classification), MSE (regression)
- Goal: minimize loss over data




**Note**

Normalization and stable outputs (e.g., Softmax) help training.

- Compute output error $\Rightarrow$ propagate **backwards**
- Chain rule links each weight to loss
- Update: $W^{(l)} \leftarrow W^{(l)} - \eta \nabla_{W^{(l)}} L$

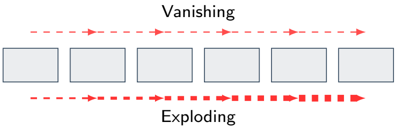



## Picture

Blue arrows forward (activations), red dashed arrows backward (errors).

# Gradient Flow: Vanishing vs Exploding

- Deep chains can **shrink** (vanish) or **blow up** (explode) gradients
- Remedies: ReLU-family activations, good initialization, normalization




**Visual**

Arrow thickness indicates gradient magnitude across layers.

# Optimization & Learning Rate

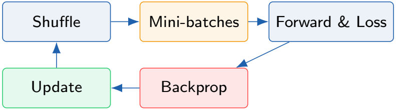
- SGD / Mini-batch GD for efficiency
- Learning rate $\eta$: small = slow, large = unstable
- Practical: schedulers, momentum/Adam


**Heuristic**

Start with a conservative $\eta$, increase if stable; otherwise decrease.

# Training Loop & Mini-batching

- Shuffle data → split into mini-batches
- For each batch: forward → loss → backprop → update
- Monitor train/val metrics; early stopping on plateau

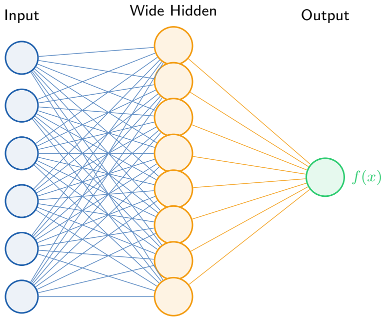



## Recipe

BatchNorm/LayerNorm, proper init, and regularization improve stability.

# Universal Approximation Theorem: Statement

- A feedforward network with one hidden layer and a **nonlinear** activation can approximate any **continuous** function on a compact domain, given enough neurons.
- UAT speaks about **expressive power**, not training ease.
- Depth is not required by the theorem, but often improves efficiency.

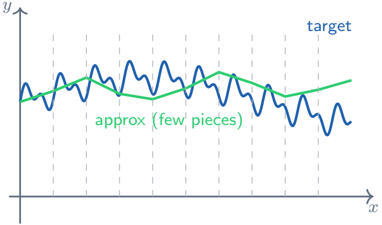



**Essence**

With sufficient width, shallow networks are universal approximators.

# UAT Intuition: Building Functions from Simple Pieces

- Partition the input domain and stitch **simple pieces** together: steps or piecewise-linear segments (e.g., sums of ReLUs).
- Increasing the number of hidden units $\Rightarrow$ finer partition $\Rightarrow$ better approximation.
- Depth can reduce the number of units needed for a given accuracy.

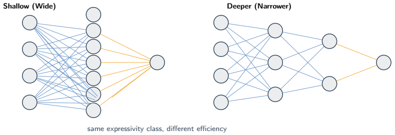



**Picture**

Target curve (blue) vs. shallow piecewise approximation (green).

# UAT: Misconceptions & Practical Limits

- **Expressivity $\neq$ Learnability**: UAT does not guarantee training success.
- Data, optimization, and regularization control **generalization**.
- Shallow universality may require **many** neurons; depth can be **parameter-efficient**.
- Choice of activation matters (ReLU/tanh vs. saturating sigmoids).

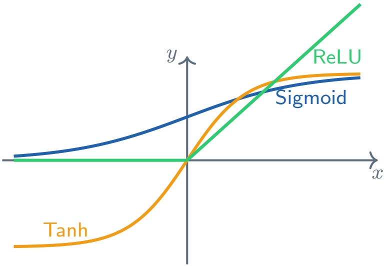



**Takeaway**

UAT explains *why* ANNs can represent complex functions—not *how* to train them well.

# Activation Zoo & Roles

- **Why**: Nonlinearity enables complex decision boundaries.
- **Sigmoid** $(0, 1)$: saturates; good for probabilities (binary).
- **Tanh** $(-1, 1)$: zero-centered; still saturates.
- **ReLU** $\max(0, x)$: simple, sparse, robust gradients.

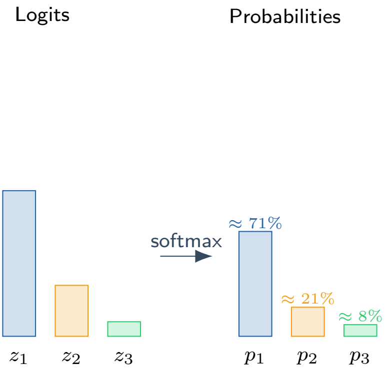



**Takeaway**

Hidden layers: ReLU-family as a strong default; outputs depend on task.

# Softmax & Cross-Entropy (Multi-class)

- **Softmax**: $p_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$ converts logits to a probability simplex.
- **Cross-Entropy**: $-\sum_i y_i \log p_i$ aligns predicted distribution with labels.
- Stable training: combine `Softmax` + `CE`; use log-sum-exp tricks in practice.

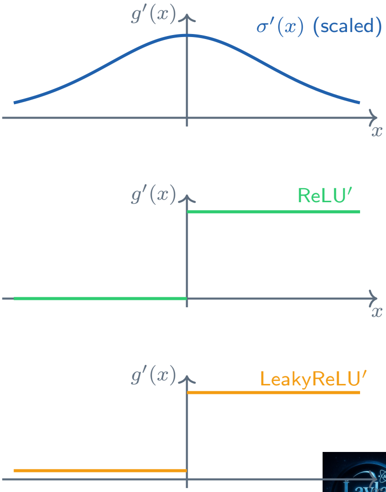



**Message**

Use Softmax at the output for single-label multi-class problems.

# Practical Tips & Pitfalls

- **Vanishing gradients**: Sigmoid/Tanh saturate $\Rightarrow$ use ReLU-family or normalization.
- **Dead ReLU**: neurons stuck at $x < 0$; mitigate with LeakyReLU/ELU/GELU.
- **Choices**:
  - Hidden: ReLU / LeakyReLU (safe defaults)
  - Output: Softmax (multi-class), Sigmoid (multi-label), Linear (regression)

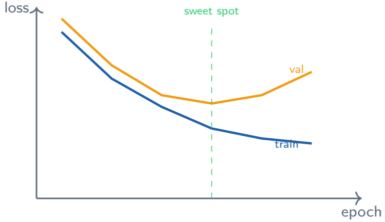



### Rule of Thumb

Start with ReLU (or LeakyReLU), change only if gradients/accuracy suggest otherwise.

# What is Overfitting?

- Model fits **noise** or idiosyncrasies of training data
- Training loss ↓ while **validation loss** eventually ↑
- Poor **generalization** to unseen data



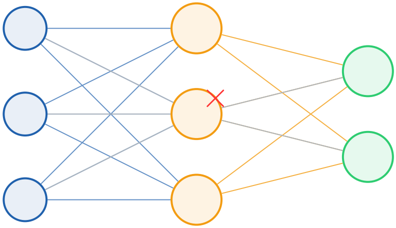

## Signal

Growing gap: train vs. validation metrics as epochs increase.

# Regularization Toolbox

- **Weight Decay (L2)**: penalize large weights
- **Dropout**: randomly deactivate units during training
- **Early Stopping**: stop at best validation performance
- (Also: Data Augmentation, Norm layers, Smaller models)

Dropout: random off during train



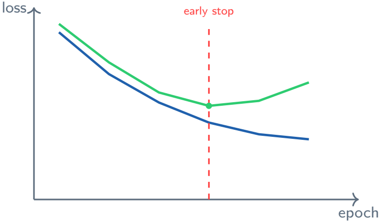
L2: keep weights small

## Goal

Reduce variance without adding too much bias.

# Validation & Early Stopping

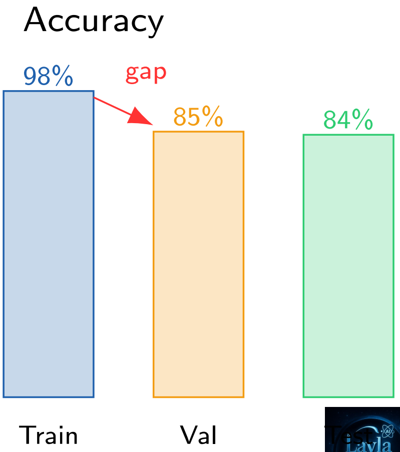
- Split data: **Train / Validation / Test**
- Monitor validation loss/accuracy each epoch
- Stop when validation no longer improves (**patience** $k$ epochs)




**Outcome**

Prevents over-training past the generalization sweet spot.

- **Large gap** train vs. val/test metrics
- Highly complex model vs. small dataset
- Unstable training, high variance across runs
- Fixes: regularization, more data/augmentation, simpler model

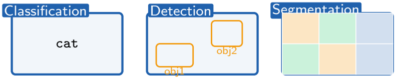



**Quick Checks**

Shuffle properly, hold-out a test set, verify labels/leakage.

# Applications: Computer Vision

- **Classification**: image-level labels
- **Detection**: bounding boxes for objects
- **Segmentation**: pixel-level understanding
- Pipelines: data $\rightarrow$ augment $\rightarrow$ CNN/MLP head $\rightarrow$ metrics

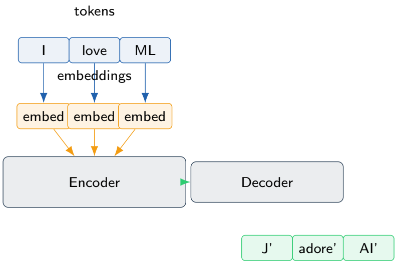



### Impact

From medical imaging to autonomous driving and retail analytics.

# Applications: Natural Language Processing

- **Machine Translation**, **Sentiment Analysis**, **Question Answering**
- Tokenization → Embeddings → Encoder/Decoder → Output
- Losses: Cross-Entropy, Label Smoothing; decoding: Greedy/Beam




**Note**

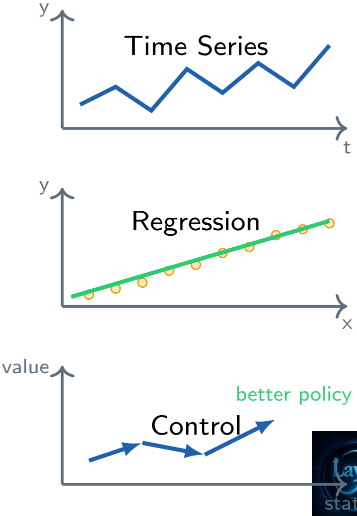
Context modeling is key; attention/transformers extend MLP fundamentals.

# Applications: Tabular/Time-Series & Robotics/Control

- **Regression/Forecasting**: prices, demand, sensors
- **Anomaly Detection**: monitoring, security
- **Control**: policy/value approximation for decision making

**Pattern**

Learned representations beat manual features when data is sufficient.



# Summary & Next Steps

- **Core**: Perceptron $\rightarrow$ MLP/Backprop $\rightarrow$ UAT
- **Practice**: Activations, optimization, regularization
- **Applications**: Vision, NLP, Tabular/Control
- **Next**: hands-on demo + try hyperparameter tweaks

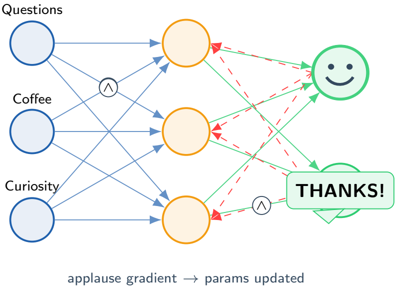

- ReLU/Softmax choices
- LR, batch size, schedulers
- L2/Dropout/Early stop

**Key Message**

Neural networks = powerful function approximators; training craft makes them useful.

## Training Stats

- final loss: $\approx 0$ (ish)
- smiles accuracy: $99.9\%$ (val)
- optimizer: Adam (caffeinated)
- batch size: **you**
- regularizer: coffee & great questions
- epoch: until **Q&A** converges




**Thank You!**
gradient of gratitude ↗