Module: Neural Networks — 10 Lessons

Designed for a fast, visual, hands-on intro. Each lesson includes learning goals, key ideas, and **key terms** (plus JavaScript sketches for a few).

Lesson 1 — Neurons & Perceptrons

Neural networks are built from simple computation units called **neurons**. Each neuron computes a weighted sum of inputs plus a bias and then applies an **activation function**. A single-layer perceptron can draw a linear decision boundary; it succeeds on linearly separable data but fails on problems like XOR without extra layers or nonlinearities.

Key terms: neuron, weight, bias, activation, perceptron, linear separability

---

Lesson 2 — The Neural Network Training Pipeline

A reliable workflow turns raw data into useful predictions: assemble a **dataset**, perform **preprocessing** (cleaning, encoding, scaling), construct a model, and **train** it by minimizing a loss with backpropagation on the training split. Evaluate on validation/test splits, monitor metrics, and finally run **inference** for deployment. Always compute preprocessing parameters on the training data only to avoid leakage.

Key terms: dataset, preprocessing, backpropagation, inference

---

Lesson 3 — Linear Layers & Affine Maps

A **linear (affine) layer** applies a matrix multiply plus bias. Stacking multiple linear layers without nonlinearities still yields one linear map; expressivity is limited to straight boundaries in feature space. Bias translates the boundary, while rank constrains what transformations are possible.

Key terms: linear layer, affine, bias, rank, expressivity

Lesson 4 — Nonlinearity & MLPs *(JS sketch)*

Adding nonlinear activations (ReLU, SiLU/Swish, Tanh) between linear layers creates **MLPs** that approximate complex functions. Depth builds hierarchies of features; width increases capacity within a layer. Choice of activation affects gradient flow, saturation, and training stability.

Key terms: MLP, hidden layer, ReLU, SiLU/Swish, Tanh, universal approximation

---

Lesson 5 — Losses, Gradients & Backprop *(JS sketch)*

Training minimizes a **loss** (e.g., cross-entropy for classification, MSE for regression) using **gradients** computed efficiently by the chain rule (**backpropagation**). Mini-batch **SGD** and its variants update parameters iteratively; learning rate and batch size strongly influence convergence speed and stability.

Key terms: loss function, cross-entropy, gradient, SGD, mini-batch, learning rate

---

Lesson 6 — Regularization & Generalization

Models must generalize beyond the training set. **Weight decay (L2)** penalizes large weights; **dropout** randomly masks units to reduce co-adaptation; **early stopping** halts training when validation loss worsens; **data augmentation** increases effective data diversity. The goal is balanced bias–variance and robust performance on unseen data.

Key terms: overfitting, underfitting, weight decay, dropout, early stopping, augmentation

---

Lesson 7 — Initialization, Normalization & Optimizers

Good **initialization** (Xavier/He) preserves signal scale at depth. **Normalization** layers (BatchNorm, LayerNorm) stabilize activations and speed training. Optimizers like **Adam/AdamW** adapt learning rates per parameter; schedules (warmup, cosine decay) improve convergence and final accuracy.

Key terms: Xavier, He, BatchNorm, LayerNorm, Adam, AdamW, LR schedule, warmup

---

Lesson 8 — Convolutional Neural Networks (CNNs)

CNNs exploit spatial structure with local receptive fields and **weight sharing**. Convolution → activation → pooling blocks extract edges, textures, and shapes; deeper layers compose these into high-level features. Stride and padding control resolution; channels track feature maps across depth.

Key terms: convolution, kernel, stride, padding, pooling, feature map

---

Lesson 9 — Sequence Models: RNNs to Transformers

Sequences (text, audio, time series) require models that track context. **RNNs** (LSTM/GRU) maintain hidden state step-by-step but struggle with long dependencies. **Transformers** use **self-attention** to relate all positions at once and rely on **positional encodings** to inject order, enabling strong performance on language and beyond.

Key terms: RNN, LSTM/GRU, self-attention, positional encoding, encoder/decoder

---

Lesson 10 — Transfer Learning, Fine-tuning & Deployment

Pretrained models offer a head start: **transfer learning** freezes early layers and **fine-tunes** later ones on your dataset. For production, consider latency and footprint via **quantization**, **distillation**, and batching. Monitor drift and re-train periodically as data distributions shift.

Key terms: transfer learning, fine-tuning, freezing, quantization, distillation

---

Visual “Big Picture” Recap

- From neurons to MLPs → nonlinearity gives expressive power.
- Pipeline discipline → split data, avoid leakage, evaluate correctly.
- Train with the right loss/optimizer → watch learning rate and batches.
- Generalize via regularization → weight decay, dropout, early stopping, augmentation.
- Choose architectures by data → CNNs for images; RNNs/Transformers for sequences.
- Scale and ship → init/norm/optimizers for stability; transfer and compress for deployment.

Lesson Quick Checks (MCQs)

Lesson 1 — Neurons & Perceptrons

Question: What does a standard neuron compute before activation?

- A. Product of inputs only
- B. Weighted sum plus bias
- C. Max over inputs
- D. Running average
  Answer: B

Lesson 2 — Training Pipeline

Question: To prevent data leakage, preprocessing parameters (e.g., scaler means) should be fit on:

- A. The full dataset before splitting
- B. Training split only, then applied to val/test
- C. Validation split only
- D. Test split only
  Answer: B

Lesson 3 — Linear Layers & Affine Maps

Question: Two stacked linear layers with no activation are equivalent to:

- A. A nonlinear map
- B. A single linear layer
- C. A decision tree
- D. A convolution with stride
  Answer: B

Lesson 4 — Nonlinearity & MLPs

Question: Which choice best explains why activations are essential?

- A. They reduce matrix multiply cost
- B. They enable non-linear decision boundaries
- C. They remove the need for gradients
- D. They always prevent overfitting
  Answer: B

Lesson 5 — Losses, Gradients & Backprop

Question: For multi-class classification with one-hot targets, the typical loss is:

- A. MAE

- B. Hinge
- C. Cross-entropy
- D. KL divergence
  Answer: C

Lesson 6 — Regularization & Generalization

Question: Which technique explicitly penalizes large weights?

- A. Early stopping
- B. L2 weight decay
- C. Dropout
- D. Data augmentation
  Answer: B

Lesson 7 — Init, Norm & Optimizers

Question: AdamW's key difference from Adam is:

- A. First use of momentum
- B. Decoupled weight decay from the gradient step
- C. Removal of adaptive learning rates
- D. Replacing bias correction
  Answer: B

Lesson 8 — CNNs

Question: CNN efficiency largely comes from:

- A. Fully connected layers on all pixels
- B. Local receptive fields and shared weights
- C. Random pixel sorting
- D. Massive kernels only
  Answer: B

Lesson 9 — Sequence Models

Question: Transformers capture long-range dependencies primarily via:

- A. Max pooling over time
- B. Recurrent hidden states
- C. Self-attention across tokens
- D. Very deep MLPs
  Answer: C

Lesson 10 — Transfer & Deployment

Question: Which technique reduces model size and latency by training a small model to match a larger one's behavior?

- A. Quantization
- B. Pruning
- C. Knowledge distillation
- D. Weight averaging
  Answer: C

Summary

We covered ten focused lessons that take you from the fundamentals of neurons and linear layers to expressive MLPs, principled training with losses and backprop, and practical strategies for stable optimization through initialization, normalization, and modern optimizers. You learned to recognize and combat overfitting using regularization and to select architectures matched to data—CNNs for spatial inputs and RNNs/Transformers for sequences. The module closes with transfer learning and deployment techniques so your models can move from notebooks to real-world systems while remaining efficient and reliable.

Next Steps

* Upcoming modules: **optimization & regularization** (L1/L2, early stopping) and **deployment & monitoring** (A/B tests, drift, feedback loops), plus a short unit on **Responsible AI** (fairness, privacy).