Machine Learning Algorithms & Deep Learning - Lecture + Cheatsheet

How training actually works. In supervised learning we observe inputs and labeled targets and learn a function that generalizes. You choose a **model family**, define a **loss**, and use an **optimizer** to update **parameters**. We monitor a validation set to avoid **overfitting** and finally **serialize** the trained model for reuse.

- **Parameters**: values learned from data (weights, biases).
- **Hyperparameters**: settings you decide (learning rate, batch size, epochs, regularization, depth).
- **Loss**: error measure (MSE, cross-entropy).
- **Optimizer**: update rule (Gradient Descent, SGD, Adam).

Epochs, batches, iterations. One **epoch** scans the whole training set. We split data into **mini-batches**; each batch produces a gradient step - an **iteration**. The **batch size** controls the noise/efficiency trade-off. Paired with **learning rate**, it shapes convergence speed and stability.

- **Small batches** -> noisier updates, regularization effect, lower memory.
- **Large batches** -> smoother updates, better hardware utilization.
- **LR too high** -> divergence; **too low** -> slow/plateau; consider warm-up/decay schedules.

Supervised algorithms - when to pick what

Use this as a quick selection guide; each item includes one-line intuition + key knobs.

- **Linear regression**: continuous targets; add **L2/L1** for stability/feature selection.
- **Logistic regression**: strong linear baseline for classification; interpretable.
- **k-NN**: prediction by neighbors; sensitive to scaling; choose **k** via validation.
- **SVM (RBF)**: maximum-margin with **non-linear** kernel; tune **C**, **gamma**.
- **Decision trees**: human-readable rules; prune to curb overfitting.
- **Random Forests**: ensemble of trees; robust default for tabular data.
- **Gradient Boosting** (XGBoost/LightGBM): sequential trees that fix residuals; tune **lr**, depth, estimators.
- **Naive Bayes**: simple probabilistic model; great for bag-of-words text.

Unsupervised algorithms - structure without labels

- **k-Means**: partitions into k clusters; pick k via elbow/silhouette.
- **Hierarchical clustering**: dendrogram for exploratory analysis.
- **DBSCAN**: density-based clusters and outliers; tune **eps**, **min_samples**.
- **PCA**: linear **dimensionality reduction** for visualization/denoising.

Deep Learning - what it is and how it works

Definition. Deep learning is a subfield of ML that learns hierarchical, non-linear representations using multi-layer **neural networks**. Unlike many classical models that rely on hand-engineered features, deep networks learn useful features from raw data end-to-end.

Neural network workflow - from input to output.

1) Neuron (node) computation. Each neuron takes a vector of inputs and computes a weighted sum plus a bias: z = w1*x1 + w2*x2 + ... + b. It then applies an activation function to produce its output a = activation(z). Common activations are **ReLU** (max(0, z)), **sigmoid** (1/(1+e^-z)), and **tanh**. Activations introduce non-linearity so stacked layers can model complex functions.

2) What each layer means.

- **Input layer**: holds raw features (pixels, words, tabular columns). No computation here; it just feeds the next layer.
- **Hidden layers**: each hidden layer transforms its inputs into progressively more abstract features. Earlier layers capture simple patterns; later layers combine them into higher-level concepts.
- **Output layer**: produces final predictions. For classification we often use a softmax layer; for regression a linear output.

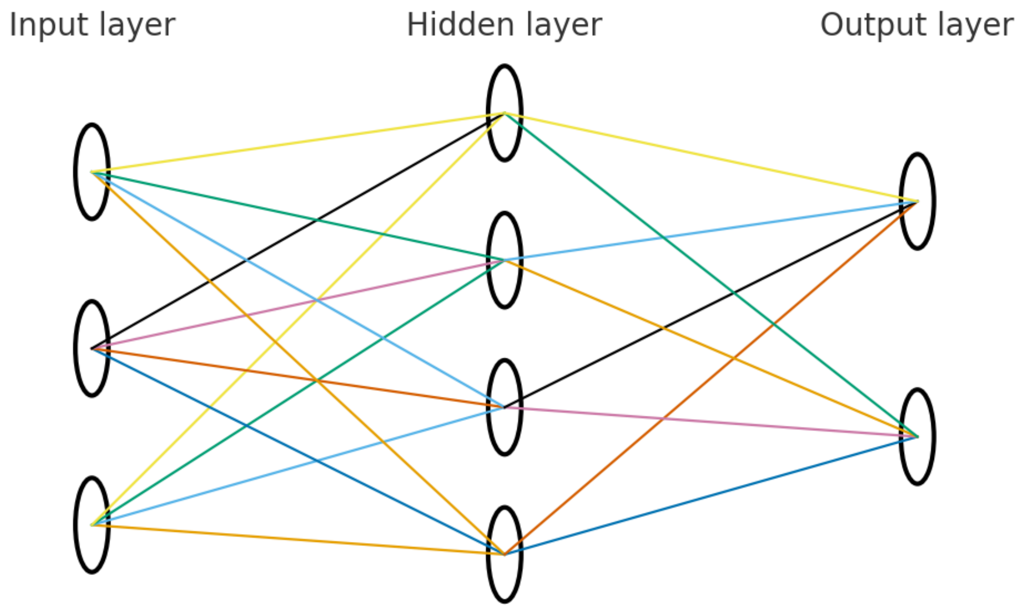
3) Forward pass. Data flows layer by layer: inputs -> hidden layers -> outputs. At each layer we compute weighted sums and activations to get the next representation. The result is a prediction y_hat.

4) Loss. We compare predictions y_hat with true targets y using a loss function (cross-entropy for classification, mean squared error for regression). The loss is a single number that summarizes how wrong the network is on the current batch.

5) Backpropagation. Using the chain rule, we propagate gradients of the loss backward through the network to compute how much each weight and bias contributed to the error.

6) Parameter update. An optimizer such as SGD or Adam updates weights using the gradients. Key dials are **learning rate**, **batch size**, and number of **epochs**.

7) Repeat over epochs. We repeat forward pass -> loss -> backprop -> update over many mini-batches and epochs until the validation metric stops improving (early stopping) or we hit a budget.



Common architectures.

- **MLP (Multilayer Perceptron)**: also called a feed-forward network; good for tabular or generic signals.
- **CNN (Convolutional Neural Network)**: uses convolution filters for images and spatial data; captures local patterns with weight sharing.
- **RNN (Recurrent Neural Network)**: processes sequences by carrying state over time; classic variants are **LSTM (Long Short-Term Memory)** and **GRU (Gated Recurrent Unit)**.
- **Transformer**: based on self-attention; dominant in **NLP (Natural Language Processing)** and increasingly used in vision and audio.

Training tips.

- **Initialization**: good weight initialization speeds up training and avoids early saturation.
- **Normalization**: batch norm or layer norm helps stabilize deep stacks.
- **Regularization**: dropout, weight decay, data augmentation, early stopping.

Put it together. Start with a simple baseline, measure a clear metric, iterate with disciplined validation. Once satisfied, save the model so it is reusable in code and deployment.

- **Workflow**: split (train/val/test) -> train -> validate -> tune -> test once.
- **Search**: coarse grid/random, then refine around good regions.
- **Serialize**: scikit-learn **joblib/pickle**; PyTorch **state_dict** (.pt/.pth); TensorFlow/Keras **SavedModel**.

Sources

• Infinite Codes - “All Machine Learning algorithms explained in 17 min” (YouTube, id: E0Hmnixke2g).

• Infinite Codes - “Epochs, Iterations and Batch Size | Deep Learning Basics” (YouTube, id: SftOqbMrGfE).

• scikit-learn Documentation - Model persistence; Hyper-parameter tuning; Cross-validation.