Machine Learning – Lecture Notes

Luu Minh Sao Khue

November 19, 2025

1. From Linear Models to Neural Networks

This lecture introduces a sequence of models that gradually increase in complexity: Logistic Regression, the Perceptron, the Multi-Layer Perceptron (MLP), and general feed-forward neural networks. The goal is to show how each model naturally evolves from the linear regression framework.

1.1 Motivation

Previously, we studied linear regression, which models the relationship between input features $\mathbf{x} \in \mathbb{R}^d$ and a continuous target y using:

$$\hat{y} = \theta^\top \mathbf{x} + b.$$

However, many real-world tasks involve **classification**, where the output is a discrete label (e.g., $y \in \{0, 1\}$). Linear regression is inappropriate for classification because its output is unbounded and not interpretable as a probability.

This motivates the introduction of models that transform the linear combination into a value between 0 and 1.

2. Logistic Regression

Logistic Regression is a classification model that converts the linear combination $z = \theta^\top \mathbf{x} + b$ into a probability using the sigmoid activation function:

$$\sigma(z) = \frac{1}{1 + e^{-z}}.$$

2.1 Model Definition

For an input $\mathbf{x}$, the model computes:

$$z = \theta^\top \mathbf{x} + b, \qquad \hat{y} = \sigma(z),$$

where $\hat{y}$ represents the estimated probability that the class label is 1.

2.2 Interpretation

The logistic model assumes:

$$P(y = 1 \mid \mathbf{x}) = \hat{y}, \qquad P(y = 0 \mid \mathbf{x}) = 1 - \hat{y}.$$

2.3 Loss Function: Cross-Entropy

For a training sample $(\mathbf{x}^{(i)}, y^{(i)})$, the **loss** is:

$$\ell^{(i)} = -\Big(y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})\Big).$$

The **cost function** (average loss) is:

$$J(\theta, b) = -\frac{1}{N} \sum_{i=1}^{N} \left[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})\right].$$

2.4 Training via Gradient Descent

Because the loss is differentiable, the parameters can be updated using gradient descent:

$$\theta \leftarrow \theta - \alpha \frac{\partial J}{\partial \theta}, \qquad b \leftarrow b - \alpha \frac{\partial J}{\partial b}.$$

Logistic regression can be viewed as a **single neuron** with a sigmoid activation.

3. The Perceptron

The Perceptron is the historical precursor to modern neural networks. It also computes a linear combination:

$$z = \theta^\top \mathbf{x} + b,$$

but uses a **step activation function**:

$$\hat{y} = \begin{cases} 1 & \text{if } z \geq 0, \\ 0 & \text{if } z < 0. \end{cases}$$

3.1 Decision Boundary

The perceptron defines a linear classifier with decision boundary:

$$\theta^\top \mathbf{x} + b = 0.$$

3.2 Perceptron Update Rule

Given a misclassified example $(\mathbf{x}^{(i)}, y^{(i)})$, the update is:

$$\theta \leftarrow \theta + \alpha\big(y^{(i)} - \hat{y}^{(i)}\big)\mathbf{x}^{(i)}, \qquad b \leftarrow b + \alpha\big(y^{(i)} - \hat{y}^{(i)}\big).$$

Unlike logistic regression, the perceptron is **not differentiable** because the step function has no gradient. It is limited to linearly separable problems.

4. Multi-Layer Perceptron (MLP)

Logistic regression and the perceptron both produce a linear decision boundary. To model nonlinear relationships, we introduce **hidden layers**.

4.1 Network Architecture

Given input $\mathbf{x} \in \mathbb{R}^d$, a one-hidden-layer MLP computes:

$$\mathbf{h} = \sigma\big(W^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\big),$$
$$\hat{y} = \sigma\big(W^{(2)}\mathbf{h} + b^{(2)}\big),$$

where:

- $W^{(1)} \in \mathbb{R}^{m \times d}$ are the weights of the hidden layer,
- $\mathbf{h} \in \mathbb{R}^m$ are the hidden activations,
- $W^{(2)} \in \mathbb{R}^{1 \times m}$ are the output weights,
- $\sigma(\cdot)$ is a nonlinear activation function (sigmoid, ReLU, tanh, etc.).

4.2 Nonlinear Decision Boundaries

Because σ is nonlinear, the network can represent complex patterns:

$$\hat{y} = \sigma\big(W^{(2)}\,\sigma\big(W^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\big) + b^{(2)}\big).$$

This enables MLPs to solve problems such as XOR that are impossible for strictly linear models.

4.3 Loss Function

For classification, we typically use binary cross-entropy:

$$J(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\Big[y^{(i)}\log\hat{y}^{(i)} + (1-y^{(i)})\log(1-\hat{y}^{(i)})\Big].$$

4.4 Training with Backpropagation

Training requires computing gradients through multiple layers. Using the chain rule, we propagate gradients from the output layer back to earlier layers:

$$\frac{\partial J}{\partial W^{(1)}} = \frac{\partial J}{\partial \mathbf{h}} \cdot \frac{\partial \mathbf{h}}{\partial W^{(1)}}, \qquad \frac{\partial J}{\partial W^{(2)}} = \frac{\partial J}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial W^{(2)}}.$$

This algorithm is called **backpropagation**. It allows efficient training of networks with many layers.

5. General Feed-Forward Neural Networks

A deep neural network generalizes the MLP by stacking multiple layers:

$$\mathbf{h}^{(1)} = \sigma\big(W^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\big),$$
$$\mathbf{h}^{(2)} = \sigma\big(W^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}\big),$$
$$\vdots$$
$$\hat{y} = \sigma\big(W^{(L)}\mathbf{h}^{(L-1)} + \mathbf{b}^{(L)}\big).$$

Each layer applies:

$$\text{linear transformation} \quad + \quad \text{nonlinear activation}.$$

5.1 Universal Approximation

A network with at least one sufficiently wide hidden layer can approximate any continuous function on a compact domain. This makes neural networks extremely powerful for modeling complex relationships.

5.2 Training Objective

The objective is to find parameters $\{W^{(l)}, \mathbf{b}^{(l)}\}$ that minimize:

$$J = \frac{1}{N} \sum_{i=1}^{N} \ell(\hat{y}^{(i)}, y^{(i)}),$$

where ℓ is typically cross-entropy (classification) or mean squared error (regression).

5.3 Optimization

Parameters are updated via gradient-based methods such as:

$$W^{(l)} \leftarrow W^{(l)} - \alpha \frac{\partial J}{\partial W^{(l)}}, \qquad \mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} - \alpha \frac{\partial J}{\partial \mathbf{b}^{(l)}}.$$

6. Summary

- Logistic Regression is a linear model with a sigmoid activation that outputs a probability.
- The Perceptron is a historical threshold-based classifier with a hard decision boundary.
- The Multi-Layer Perceptron introduces hidden layers and nonlinear activations, enabling nonlinear decision boundaries.
- Deep neural networks stack many such layers and are trained using backpropagation.