Lab 5: Introduction to Neural Networks

Ali Al-Zawqari

2024-2025

1 Lab objectives

The goal of this lab is to:

- Build a two–layer Artificial Neural Network (ANN) from scratch without using high-level libraries.
- Understand and implement forward and backward propagation for multi-class classification.
- Visualize decision boundaries and training progress.
- Evaluate the model using accuracy and confusion matrices.

2 Neural Networks

2.1 Lab Overview

In this exercise, you will implement a solution to **classify species of *Iris* flowers**—*Irissetosa*, *Irisversicolor*, and *Iris virginica*—from their morphological measurements. You will construct every component of the network yourself (initialization, forward pass, loss, backward pass, and parameter updates).

2.2 Architecture and Mathematical Formulation

Given an input matrix $\mathbf{X} \in \mathbb{R}^{m\times 4}$, where m is the number of samples and the four columns correspond to sepal length, sepal width, petal length, and petal width, the network computes class probabilities via two affine transformations separated by a non–linear activation:

$$\mathbf{Z}^{(1)} = \mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)}, \qquad \mathbf{W}^{(1)} \in \mathbb{R}^{4\times 8},\ \mathbf{b}^{(1)} \in \mathbb{R}^{1\times 8}, \tag{1}$$

$$\mathbf{A}^{(1)} = \sigma\left(\mathbf{Z}^{(1)}\right), \qquad \mathbf{A}^{(1)} \in \mathbb{R}^{m\times 8} \text{ (sigmoid activation)}, \tag{2}$$

$$\mathbf{Z}^{(2)} = \mathbf{A}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}, \qquad \mathbf{W}^{(2)} \in \mathbb{R}^{8\times 3},\ \mathbf{b}^{(2)} \in \mathbb{R}^{1\times 3}, \tag{3}$$

$$\mathbf{A}^{(2)} = \text{softmax}\left(\mathbf{Z}^{(2)}\right), \qquad \mathbf{A}^{(2)} \in \mathbb{R}^{m\times 3}. \tag{4}$$

The sigmoid activation function is defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \tag{5}$$

The softmax operator is defined component-wise by

$$[\text{softmax}(\mathbf{z})]_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{for } i = 1, 2, \ldots, K, \tag{6}$$

Loss function. With one-hot labels $\mathbf{Y} \in \{0, 1\}^{m \times 3}$, the categorical cross-entropy loss is

$$\mathcal{L} = -\frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{3} Y_{ik} \log A_{ik}^{(2)}. \tag{7}$$

Gradients. Using the chain rule, the gradients used in backpropagation are

$$\mathbf{dZ}^{(2)} = \mathbf{A}^{(2)} - \mathbf{Y}, \tag{8}$$

$$\mathbf{dW}^{(2)} = \frac{1}{m} \left(\mathbf{A}^{(1)}\right)^{\top} \mathbf{dZ}^{(2)}, \qquad \mathbf{db}^{(2)} = \frac{1}{m} \sum_{i=1}^{m} \mathbf{dZ}_{i,\cdot}^{(2)}, \tag{9}$$

$$\mathbf{dZ}^{(1)} = \left(\mathbf{dZ}^{(2)} \mathbf{W}^{(2)\top}\right) \odot \sigma'\left(\mathbf{Z}^{(1)}\right), \tag{10}$$

$$\mathbf{dW}^{(1)} = \frac{1}{m} \mathbf{X}^{\top} \mathbf{dZ}^{(1)}, \qquad \mathbf{db}^{(1)} = \frac{1}{m} \sum_{i=1}^{m} \mathbf{dZ}_{i,\cdot}^{(1)}, \tag{11}$$

where $\odot$ denotes element-wise multiplication and $\sigma'(z) = \sigma(z)(1 - \sigma(z))$.

Parameter update. For a learning rate $\eta > 0$ the parameters are updated using batch gradient descent:

$$\theta \leftarrow \theta - \eta \nabla_{\theta} \mathcal{L}, \quad \theta \in \mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(2)}. \tag{12}$$

2.3 Dataset

The *Iris* dataset consists of 150 labeled observations (50 per species). Each sample comprises four continuous features and a categorical class label. The primary source of the data is Fisher [1]. In this lab, the dataset is provided via `scikit-learn`:

```
from sklearn.datasets import load_iris
```

Prior to training, the features are standardized using `StandardScaler` and split into 80% training and 20% test subsets.

2.4 Working Procedure

1. **Setup.** Install the required libraries and execute the provided notebook to load the data.
2. **Implement the network.** Complete the functions for initialization, forward propagation, loss computation, backward propagation, and parameter updates.
3. **Train.** Train the network for `2000` epochs with a learning rate of $\eta = 0.1$ and a hidden size of 8.
4. **Monitor.** Record the loss every epoch and plot the training curve.
5. **Evaluate.** Compute accuracy and display a confusion matrix on both the training and the held-out test set.
6. **Visualise.** Plot 2-D decision boundaries for all six pairs of features.
7. **Experiment (optional).** Vary hyperparameters, add regularisation, or extend the network depth.

2.5 Theoretical background

In 1962, Arthur Samuel wrote an essay[1] proposing a new approach for computers to complete a task. His suggestion can be marked as the birth of machine learning. Samuel's idea was straightforward yet powerful: **instead of programming the computer to do the task step by step, show the computer examples of a problem that you wanted to solve and let it figure out how to solve it**. Figure 1 shows the updated version of Samuel's original idea of training a computer to complete a specific task, which you will work with to solve this lab's problem. Here, the neural network is your model, and by that, each block of Samuel's machine learning model will be explained in the context of using neural networks as your solution.

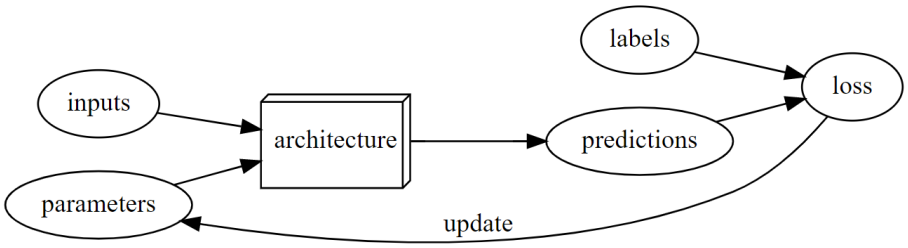

Figure 1: Training loop of Neural Networks

Your dataset provides the **inputs** and **labels**, usually called input u and output y in your electrical engineering courses. The inputs represent examples with features you want to

[1] Artificial Intelligence: A Frontier of Automation, Arthur L. Samuel

use to predict specific labels (output). For example, if you have 20 samples in your dataset, and each sample has 8 features, your inputs and labels can be represented mathematically as in Equation 13.

$$X = \begin{bmatrix} (x_{(1)})^T \\ \vdots \\ (x_{(20)})^T \end{bmatrix}; Y = \begin{bmatrix} (y_{(1)}) \\ \vdots \\ (y_{(20)}) \end{bmatrix} \tag{13}$$

Each sample of your **inputs** $x_{(n)}$ is a vector of 1×8. The **label** $y_{(n)}$ (output) is a specific scalar or one-hot vector that you aim to predict.

Neural networks are represented mathematically as the function shown in Equation 14.

$$\hat{y} = f_\theta(x) \tag{14}$$

In words, a neural network computes the prediction output $\hat{y}$ from an input x using the defined function f_θ. This function f_θ consists of a sequence of multiple stages connected by a non-linear process. Each stage has **parameters** w, b (weights and bias in deep learning language) used for multiplications and additions. The total number of these stages (layers), the number of individual units per stage (neurons), and the type of non-linear function are called the neural network **architecture**. The simplest neural network architecture should have at least three layers: an input layer, a hidden layer, and an output layer.

A toy example is shown in Figure 2 to explain how neural networks work. First, the feature values are organized as an input vector x. Then, a linear operation (a dot product) is performed with **input vector** x and **parameters** $w^{(1)}$. The neurons pass the resulting weighted sum to a **non-linear function**. In this example, the non-linear function is the Rectified Linear Unit (ReLU), defined as

$$\text{Re}(x) = \max(0, x). \tag{15}$$

The second layer repeats the linear operation with new parameters $w^{(2)}$ and produces a final prediction y. The **Mean Squared Error** (MSE) loss in Equation 16 is often used for regression; in this lab, we instead use the categorical cross-entropy loss defined earlier.

$$\text{Loss} = \text{MSE}(y, y) = \frac{1}{n}\sum_{i=1}^{n}(y_i - y_i^*)^2. \tag{16}$$

To minimize a loss function, an optimization algorithm such as **stochastic gradient descent** iteratively updates the parameters w. We strongly recommend the 3Blue1Brown neural-network series by Grant Sanderson for an intuitive visual explanation.

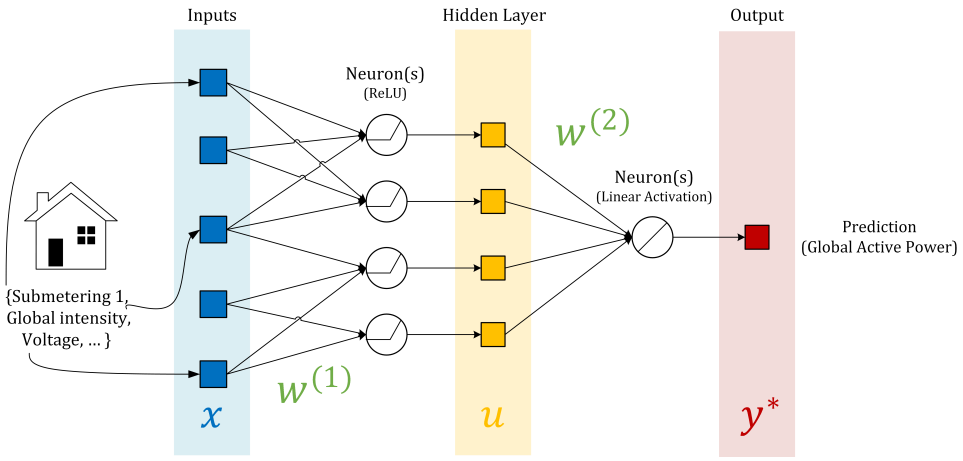

Figure 2: Two-layer Artificial Neural Network (illustrative example)

3 Minimum Requirements and Submission

3.1 Minimum Requirements

The minimum requirements are summarised by answering the following questions in your solution:

- How did you pre-process your dataset? How did you split your dataset?
- What is your neural-network architecture? How did you choose the number of layers/neurons?
- What activation function did you use and why?
- What is your cost function?
- How did you implement back-propagation? What optimiser did you use?
- How did you judge your model? What is the rationale for using the chosen metric(s)?

3.2 Submission

You do **not** have to submit code through the learning-management system. Bring your code and results to the oral exam; you will be asked to explain and discuss specific parts of your implementation.

References

[1] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, 1936.