INF721 - Deep Learning
L5: Multilayer Perceptron

Prof. Lucas N. Ferreira
Universidade Federal de Viçosa

2024/2

1 Introduction

This lecture introduces the multilayer perceptron (MLP), which is the first neural network architecture we will study in this deep learning course. The MLP builds upon concepts from linear and logistic regression to create more powerful models capable of learning complex nonlinear functions.

2 Linearly Separable Problems

We begin by reviewing linearly separable problems, which can be solved using simple linear models like logistic regression.

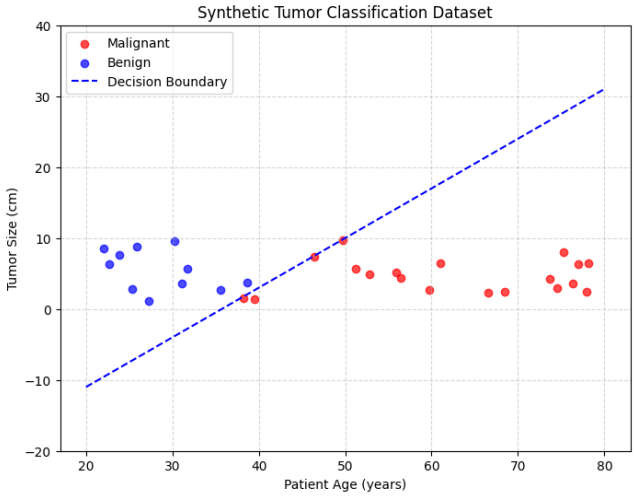

Figure 1: Example of a linearly separable binary classification problem

A problem is linearly separable if there exists a hyperplane (a line in 2D) that can perfectly separate the classes. For binary classification, this means we can find a decision boundary of the form:

$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

Where $\mathbf{w}$ is the weight vector, $\mathbf{x}$ is the input vector, and b is the bias term.

3 The Perceptron

The perceptron was one of the earliest trainable artificial neurons, proposed by Frank Rosenblatt in 1958. It's important historically, even though we don't commonly use it today due to limitations in its training procedure.

The perceptron classification rule is:

$$h(x) = \text{sgn}(\mathbf{w} \cdot \mathbf{x} + b)$$

Where the sign function is defined as:

$$\text{sgn}(z) = \begin{cases} +1, & \text{if } z \geq 0 \\ -1, & \text{if } z < 0 \end{cases}$$

Example: Consider a perceptron with weights $\mathbf{w} = [-0.7, 1]$ and bias $b = 25$.

For input $\mathbf{x}^{(1)} = [51, 8]$:

$$\begin{aligned} h(\mathbf{x}^{(1)}) &= \text{sgn}(-0.7 \cdot 51 + 1 \cdot 8 + 25) \\ &= \text{sgn}(-2.7) \\ &= -1 \end{aligned}$$

For input $x^{(2)} = [10, 30]$:

$$\begin{aligned} h(\mathbf{x}^{(2)}) &= \text{sgn}(-0.7 \cdot 10 + 1 \cdot 30 + 25) \\ &= \text{sgn}(48) \\ &= 1 \end{aligned}$$

Note that the perceptron is not trained using gradient descent because the sign function is not differentiable. Instead, it uses a simple update rule based on misclassifications.

4 Artificial Neurons

We now introduce the concept of an artificial neuron, which generalizes linear models like logistic regression and the perceptron. An artificial neuron computes a prediction $\hat{y} = g(z)$, where:

1. $z = w \cdot x + b$ is a linear combination of inputs and weights and;

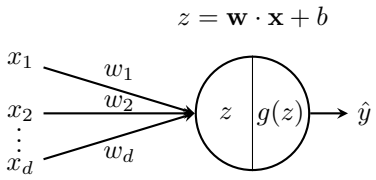

Figure 2: Structure of an artificial neuron with d input features

2. $g(z)$ is a (typically) non-linear activation function

Different activation functions lead to different models:

- Linear Regression: $g(z) = z$
- Logistic Regression: $g(z) = \frac{1}{1+e^{-z}}$
- Perceptron: $g(z) = \text{sgn}(z)$

5 Non-linearly Separable Problems

Many real-world problems are not linearly separable. A classic example is the XOR function:

x_1	x_2	$f(x_1, x_2) = x_1$ XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

Table 1: Truth table for XOR function

No single line can separate the positive (1) and negative (0) examples in the XOR problem.

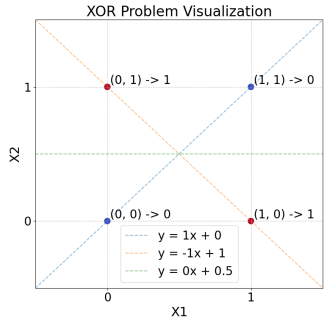

Figure 3: Plot of XOR function to show it is not linearly separable

This is where neural networks become powerful - they can learn new representations of the input data that make the problem linearly separable.

6 Multilayer Perceptron (MLP)

The multilayer perceptron, also known as a feedforward neural network, consists of multiple layers of artificial neurons. The MLP architecture includes:

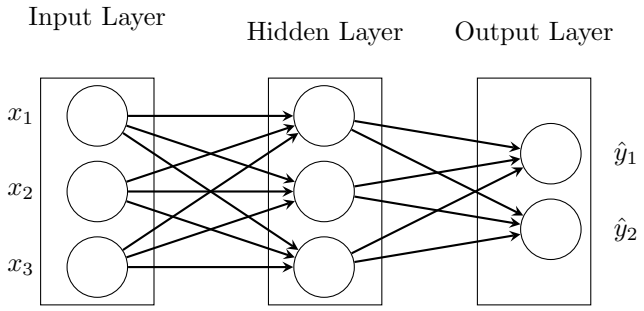

Figure 4: Architecture of a Multilayer Perceptron

- Input Layer: Receives the raw input features
- Hidden Layer(s): One or more layers of neurons that transform the input
- Output Layer: Produces the final prediction

7 Forward Pass

The forward pass is the process of computing the output of an MLP given an input. We'll walk through this process for a simple two-layer network. Consider an MLP with:

- Input layer: 2 features (x_1, x_2)
- Hidden layer: 2 neurons
- Output layer: 1 neuron

7.1 Hidden Layer Computation

For the hidden layer (layer 1):

$$a_1 = g^{[1]}(w_{11}^{[1]}x_1 + w_{21}^{[1]}x_2 + b_1^{[1]})$$
$$a_2 = g^{[1]}(w_{12}^{[1]}x_1 + w_{22}^{[1]}x_2 + b_2^{[1]})$$

We can write this in vector form: $a^{[1]} = g^{[1]}(W^{[1]}\mathbf{x} + \mathbf{b}^{[1]})$, where:

$$W^{[1]} = \begin{bmatrix} w_{11}^{[1]} & w_{21}^{[1]} \\ w_{12}^{[1]} & w_{22}^{[1]} \end{bmatrix}, \quad \mathbf{b}^{[1]} = \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

7.2 Output Layer Computation

For the output layer (layer 2):

$$\hat{y} = g^{[2]}(w_{11}^{[2]}a_1 + w_{21}^{[2]}a_2 + b_1^{[2]})$$

In vector form:

$$\hat{y} = g^{[2]}(W^{[2]}a^{[1]} + b^{[2]})$$

Where $W^{[2]} = [w_{11}^{[2]} \quad w_{21}^{[2]}]$ and $b^{[2]}$ is a scalar.

7.3 Vectorized Forward Pass

For efficiency, we typically process multiple examples simultaneously. Given a dataset X with m examples:

$$X = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(m)} \\ x_2^{(1)} & x_2^{(2)} & \cdots & x_2^{(m)} \end{bmatrix}$$

The forward pass becomes:

$$A^{[1]} = g^{[1]}(W^{[1]}X + b^{[1]})$$
$$\hat{\mathbf{y}} = g^{[2]}(W^{[2]}A^{[1]} + b^{[2]})$$

Where $\hat{\mathbf{y}} = [\hat{y}^{(1)} \quad \hat{y}^{(2)} \quad \cdots \quad \hat{y}^{(m)}]$

8 Hypothesis Space

The hypothesis space of an MLP is the set of all functions it can represent. For our two-layer network, this can be written as:

$$h(\mathbf{x}) = g^{[2]}(W^{[2]} \cdot g^{[1]}(W^{[1]}\mathbf{x} + \mathbf{b}^{[1]}) + b^{[2]})$$

This formulation shows that MLPs learn composite functions, which allows them to approximate complex nonlinear mappings.

9 Activation Functions

Activation functions introduce non-linearity into the network, allowing it to learn more complex patterns. Common activation functions include:

9.1 Logistic (Sigmoid)

$$g(z) = \frac{1}{1 + e^{-z}}$$
$$g'(z) = g(z)(1 - g(z))$$

9.2 Hyperbolic Tangent (tanh)

$$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$
$$g'(z) = 1 - g(z)^2$$

9.3 Rectified Linear Unit (ReLU)

$$g(z) = \max(0, z)$$
$$g'(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z < 0 \\ \text{undefined} & \text{if } z = 0 \end{cases}$$

9.4 Leaky ReLU

$$g(z) = \max(0.01z, z)$$
$$g'(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0.01 & \text{if } z < 0 \\ \text{undefined} & \text{if } z = 0 \end{cases}$$

10 Importance of Non-linear Activation Functions

Non-linear activation functions are crucial for the expressive power of neural networks. If we used only linear activations, the entire network would collapse into a single linear transformation:

$$\begin{aligned} h(\mathbf{x}) &= W^{[2]} \cdot (W^{[1]} \cdot \mathbf{x} + \mathbf{b}^{[1]}) + b^{[2]} \\ &= (W^{[2]} \cdot W^{[1]}) \cdot \mathbf{x} + (W^{[2]} \cdot \mathbf{b}^{[1]}) + b^{[2]} \\ &= W' \cdot \mathbf{x} + \mathbf{b}' \end{aligned}$$

This shows that without non-linear activations, a multi-layer network is no more powerful than a single-layer linear model.

11 Initializing MLP Weights

Proper weight initialization is crucial for training deep neural networks. Unlike in logistic regression, we cannot initialize all weights to zero in MLPs. If we did, all neurons in a layer would compute the same function, making the network no more powerful than a single neuron.

Instead, we typically initialize weights with small random values. This breaks symmetry and allows different neurons to learn different functions. Initializing close to zero is beneficial because it puts the neuron activations in the region where their gradients are largest, facilitating faster learning.

12 Deep Neural Networks

As we add more hidden layers, we create deeper neural networks. The general formulation for a layer l in a deep network is:

$$Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]}$$
$$A^{[l]} = g^{[l]}(Z^{[l]})$$

Where $A^{[0]} = X$ (the input) and $A^{[L]} = \hat{\mathbf{y}}$ (the output) for an L-layer network.

13 Output Layer Design

The design of the output layer depends on the type of problem we're solving:

13.1 Regression

For regression problems, we typically use a single output neuron with a linear activation function:

$$\hat{y} = z$$

13.2 Binary Classification

For binary classification, we use a single output neuron with a sigmoid activation function:

$$\hat{y} = g(z) = \frac{1}{1 + e^{-z}}$$

Here, $\hat{y}$ represents the probability of the positive class: $P(y = 1|x)$.

13.3 Multiclass Classification

For multiclass classification with C classes, we use C output neurons with the softmax activation function:

$$\hat{y}_c = g(z_c) = \frac{e^{z_c}}{\sum_{j=1}^{C} e^{z_j}}$$

The softmax function ensures that the outputs sum to 1, giving a valid probability distribution over the classes.

14 Loss Functions

14.1 Regression

For regression problems, we typically use the mean squared error loss:

$$L(h) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)})^2$$

Where $\hat{y}^{(i)}$ is the predicted value and $y^{(i)}$ is the true value for example i.

14.2 Binary Classification

For binary classification, we typically use the binary cross-entropy loss:

$$L(h) = -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$$

14.3 Multiclass Classification

For multiclass classification, we typically use the categorical cross-entropy loss:

$$L(h) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{c=1}^{C} y_c^{(i)} \log(\hat{y}_c^{(i)})$$

Where $y_c^{(i)}$ is 1 if example i belongs to class c, and 0 otherwise.

15 Example

Now that we have a good understanding of the multilayer perceptron, let's work through a concrete example to solidify our understanding.

- Model:

$$\begin{aligned} z^{[1]} &= W^{[1]}x + b^{[1]} \\ a^{[1]} &= relu(z^{[1]}) \\ z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\ \hat{y} &= \sigma(z^{[2]}) \end{aligned}$$

- Input: $x = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ and true label: $y = 1$
- Weights and biases:

$$\begin{aligned} W^{[1]} &= \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{bmatrix}, \quad b^{[1]} = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix} \\ W^{[2]} &= \begin{bmatrix} 0.5 & 0.6 \end{bmatrix}, \quad b^{[2]} = 0.3 \end{aligned}$$

- Loss function: Binary cross-entropy

$$L(\hat{y}, y) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

Let's compute the forward pass for this example:

1. Hidden Layer:

$$\begin{aligned} z^{[1]} &= W^{[1]}x + b^{[1]} \\ &= \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix} = \begin{bmatrix} 0.1(1) + 0.2(2) \\ 0.3(1) + 0.4(2) \end{bmatrix} + \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix} \\ &= \begin{bmatrix} 0.5 \\ 1.1 \end{bmatrix} + \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix} = \begin{bmatrix} 0.6 \\ 1.3 \end{bmatrix} \\ a^{[1]} &= g(z^{[1]}) = \max(0, z^{[1]}) = \begin{bmatrix} \max(0, 0.6) \\ \max(0, 1.3) \end{bmatrix} = \begin{bmatrix} 0.6 \\ 1.3 \end{bmatrix} \end{aligned}$$

2. Output layer:

$$\begin{aligned} z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\ &= \begin{bmatrix} 0.5 & 0.6 \end{bmatrix} \begin{bmatrix} 0.6 \\ 1.3 \end{bmatrix} + 0.3 = (0.5 \cdot 0.6 + 0.6 \cdot 1.3) + 0.3 \\ &= (0.3 + 0.78) + 0.3 = 1.38 \\ \hat{y} &= \sigma(z^{[2]}) = \frac{1}{1 + e^{-z^{[2]}}} = \frac{1}{1 + e^{-1.38}} \approx 0.7986 \end{aligned}$$

3. Compute the loss:

$$\begin{aligned} L(\hat{y}, y) &= -y \log \hat{y} - (1 - y) \log(1 - \hat{y}) \\ &= -(1) \log(0.7986) - (1 - 1) \log(1 - 0.7986) \\ &= -\log(0.7986) \\ &\approx 0.2247 \end{aligned}$$

16 Conclusion

The multilayer perceptron is a powerful model capable of learning complex non-linear functions. By stacking layers of artificial neurons and using non-linear activation functions, MLPs can approximate a wide range of mappings from inputs to outputs. In the next lecture, we will explore how to train these networks efficiently using the backpropagation algorithm.

Exercises

1. What is the main advantage of using a multilayer perceptron (MLP) over logistic regression?

   (a) MLPs are always faster to train

   (b) MLPs can learn non-linear decision boundaries

   (c) MLPs require less data for training

   (d) MLPs always have a lower error rate

2. Consider an MLP with input $x = [2, 1]^T$, hidden layer weights $W^{[1]} = \begin{bmatrix} 0.5 & 0.1 \\ 0.2 & 0.4 \end{bmatrix}$, hidden layer bias $b^{[1]} = [0.1, 0.2]^T$, ReLU activation for the hidden layer, output layer weights $W^{[2]} = [0.3, 0.7]$, output bias $b^{[2]} = 0.1$, and sigmoid activation for the output layer. Calculate the output of the hidden layer and the final output of the network. Which of the following is correct? (Round the final output to 4 decimal places)

   (a) $a^{[1]} = [1.2, 0.8]^T$, $\hat{y} \approx 0.8807$

   (b) $a^{[1]} = [1.1, 0.6]^T$, $\hat{y} \approx 0.8176$

   (c) $a^{[1]} = [1.3, 0.7]^T$, $\hat{y} \approx 0.8575$

   (d) $a^{[1]} = [1.2, 0.8]^T$, $\hat{y} \approx 0.8575$

3. Why is it important to use non-linear activation functions in hidden layers of an MLP?

   (a) They make the network train faster

   (b) They allow the network to approximate non-linear functions

   (c) They reduce the number of parameters in the network

   (d) They prevent overfitting

4. For a multiclass classification problem with 5 classes, how many neurons should the output layer have, and what activation function should be used?

   (a) 1 neuron with sigmoid activation

   (b) 5 neurons with sigmoid activation

   (c) 5 neurons with softmax activation

   (d) 5 neurons with ReLU activation

5. Consider a 3-class classification problem. For a batch of 2 examples, the true labels are $y^{(1)} = [1, 0, 0]^T$ and $y^{(2)} = [0, 0, 1]^T$, and the predicted probabilities are $\hat{y}^{(1)} = [0.7, 0.2, 0.1]^T$ and $\hat{y}^{(2)} = [0.1, 0.3, 0.6]^T$. Calculate the categorical cross-entropy loss for this batch. Which of the following is correct? (Round to 4 decimal places)

   (a) $L \approx 0.3567$

   (b) $L \approx 0.5108$

   (c) $L \approx 0.7133$

   (d) $L \approx 0.8979$