INF721 - Deep Learning
L4: Logistic Regression

Prof. Lucas N. Ferreira
Universidade Federal de Viçosa

2024/2

1 Introduction

This lecture covers logistic regression, an important algorithm in machine learning and a foundational concept for understanding neural networks. We'll start by revisiting linear regression with multiple features, then discuss vectorization techniques for efficient implementation, and finally dive into logistic regression.

2 Linear Regression with Multiple Features

2.1 Recap of Univariate Linear Regression

In the previous lecture, we discussed univariate linear regression, which models the relationship between a single input feature x and an output y:

$$h(x) = wx + b \tag{1}$$

Where w is the weight and b is the bias term.

2.2 Multiple Linear Regression

We can extend this model to handle multiple input features. Given a dataset D with d input features:

x_1 (Size in m^2)	x_2 N. of beds	x_3 Age in years	y (Price in 1000's USD)
152	4	24	1550
229	3	35	2286
84	1	10	293
95	3	14	196
⋮	⋮	⋮	⋮

The multiple linear regression model is:

$$h_{\mathbf{w},b}(\mathbf{x}) = w_1x_1 + w_2x_2 + \ldots + w_dx_d + b \tag{2}$$

We can write this more compactly using vector notation:

$$h_{\mathbf{w},b}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b \tag{3}$$

Where:

- $\mathbf{w} = [w_1, w_2, \ldots, w_d]$ is the weight vector
- $\mathbf{x} = [x_1, x_2, \ldots, x_d]$ is the input vector
- b is the scalar bias term

The dot product $\mathbf{w} \cdot \mathbf{x}$ is defined as:

$$\mathbf{w} \cdot \mathbf{x} = w_1x_1 + w_2x_2 + \ldots + w_dx_d \tag{4}$$

2.3 Loss Function and Gradients

The loss function for multiple linear regression remains the Mean Squared Error (MSE):

$$L(h_{\mathbf{w},b}) = \frac{1}{2m}\sum_{i=1}^{m}(h_{\mathbf{w},b}(\mathbf{x}^{(i)}) - y^{(i)})^2 \tag{5}$$

The gradients with respect to each weight w_j and the bias b are:

$$\frac{\partial L}{\partial w_j} = \frac{1}{m}\sum_{i=1}^{m}(h_{\mathbf{w},b}(\mathbf{x}^{(i)}) - y^{(i)})x_j^{(i)} \tag{6}$$

$$\frac{\partial L}{\partial b} = \frac{1}{m}\sum_{i=1}^{m}(h_{\mathbf{w},b}(\mathbf{x}^{(i)}) - y^{(i)}) \tag{7}$$

3 Vectorization

Vectorization is a programming technique that optimizes code to perform operations on entire vectors or matrices at once, rather than using explicit loops. This approach is crucial in machine learning for several reasons:

- Significantly faster execution (takes advantage of SIMD instructions)
- More concise and readable code
- Better utilization of modern CPU/GPU architectures
- Improved scalability for large datasets

3.1 Vectorizing Multiple Linear Regression

Let's compare a non-vectorized implementation with a vectorized one:

Without vectorization:

```
# Input features as a list
x = [152, 4, 24]
# Weights as a list
w = [0.1, 4.0, -2.0]
# Bias term as a float
b = 4

def model(x, w, b):
    y_hat = 0
    for i in range(len(x)):
        y_hat += w[i] * x[i]
    return y_hat + b
```

With vectorization (using NumPy):

```
import numpy as np

# Input features as a vector
x = np.array([152, 4, 24])
# Weights as a vector
w = np.array([0.1, 4.0, -2.0])
# Bias term as a float
b = 4

def model(x, w, b):
    return np.dot(w, x) + b
```

The vectorized version is not only more concise but also much faster, especially for large datasets.

3.2 Why Vectorization Speeds Up ML Code

Vectorization takes advantage of Single Instruction, Multiple Data (SIMD) operations in modern CPUs and GPUs. Instead of performing operations sequentially in a loop, vectorized operations can be executed in parallel.

For example, in the dot product calculation:

- Non-vectorized: Requires d cycles (where d is the number of features)
- Vectorized: Can be computed in a single cycle (or a few cycles, depending on the hardware)

This parallelism leads to significant speedups, especially for large datasets and complex models.

4 Logistic Regression

4.1 Problem: Binary Classification

Let's consider a binary classification problem: predicting whether a tumor is malignant based on its size. Our dataset D might look like this:

x (size cm)	y (malignant)
9.63	1
4.32	0
5.42	0
9.52	1
$\vdots$	$\vdots$

Where $y = 1$ indicates a malignant tumor, and $y = 0$ indicates a benign tumor.

4.2 Why Not Use Linear Regression?

One might be tempted to use linear regression for this problem, but there are several issues:

1. **Unbounded output:** Linear regression produces outputs in $\mathbb{R}$, not $[0, 1]$.
2. **Interpretation:** The output of linear regression doesn't have a clear probabilistic interpretation.
3. **Sensitivity to outliers:** Extreme values can significantly skew the decision boundary.

We could try to address the first issue by defining a prediction threshold:

$$\hat{y} = \begin{cases} 0, & \text{if } h(x) < 0.5 \\ 1, & \text{if } h(x) \geq 0.5 \end{cases} \tag{8}$$

However, this doesn't solve the other problems, and the model remains sensitive to outliers.

4.3 Logistic Regression Model

Logistic regression addresses these issues by using a logistic function (also called sigmoid function) to map the linear combination of inputs to a value between 0 and 1:

$$h_{\mathbf{w},b}(\mathbf{x}) = \sigma(\mathbf{w} \cdot \mathbf{x} + b) \tag{9}$$

Where σ is the sigmoid function:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{10}$$

The sigmoid function has several important properties:

- It's bounded between 0 and 1: $0 < \sigma(z) < 1$
- It's S-shaped, with a steeper slope near $z = 0$
- $\sigma(0) = 0.5$

4.4 Hypothesis Space

The hypothesis space for logistic regression consists of all functions of the form:

$$h_{\mathbf{w},b}(\mathbf{x}) = \frac{1}{1 + e^{-(\mathbf{w} \cdot \mathbf{x} + b)}} \tag{11}$$

Different values of $\mathbf{w}$ and b produce different shapes of the sigmoid curve:

- Changing $\mathbf{w}$ affects the steepness of the curve
- Changing b shifts the curve left or right

4.5 Probabilistic Interpretation

One of the key advantages of logistic regression is its probabilistic interpretation. The output $h_{\mathbf{w},b}(\mathbf{x})$ can be interpreted as the probability that the label y is 1, given the input $\mathbf{x}$:

$$h_{\mathbf{w},b}(\mathbf{x}) = P(y = 1|\mathbf{x}) \tag{12}$$

For example:

- If $h(3) = 0.12$, it means there's a 12
- If $h(7) = 0.94$, it means there's a 94

The probability of the tumor being benign ($y = 0$) is simply the complement:

$$P(y = 0|\mathbf{x}) = 1 - P(y = 1|\mathbf{x}) = 1 - h_{\mathbf{w},b}(\mathbf{x}) \tag{13}$$

4.6 Decision Boundary

To make predictions, we typically use a threshold of 0.5:

$$\hat{y} = \begin{cases} 0, & \text{if } h_{\mathbf{w},b}(\mathbf{x}) < 0.5 \\ 1, & \text{if } h_{\mathbf{w},b}(\mathbf{x}) \geq 0.5 \end{cases} \tag{14}$$

The decision boundary is the set of points where $h_{\mathbf{w},b}(\mathbf{x}) = 0.5$. For a two-dimensional input space, this forms a line (or hyperplane in higher dimensions) that separates the two classes. For example, if we have a trained model:

$$h_{\mathbf{w},b}(\mathbf{x}) = \sigma(x_1 + x_2 - 3) \tag{15}$$

The decision boundary would be the line $x_1 + x_2 = 3$.

4.7 Loss Function: Binary Cross-Entropy

To train a logistic regression model, we need a suitable loss function. While we could use Mean Squared Error (MSE), it's not the best choice for logistic regression because it results in a non-convex optimization problem. Instead, we use the Binary Cross-Entropy (BCE) loss function:

$$L(h) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(h(\mathbf{x}^{(i)})) + (1-y^{(i)})\log(1-h(\mathbf{x}^{(i)}))\right] \tag{16}$$

This loss function is derived from the principle of maximum likelihood estimation. Here's an intuitive explanation:

1. For a single example $(\mathbf{x}^{(i)}, y^{(i)})$, we want to maximize:
   - $P(y^{(i)} = 1|\mathbf{x}^{(i)}) = h(\mathbf{x}^{(i)})$ if $y^{(i)} = 1$
   - $P(y^{(i)} = 0|\mathbf{x}^{(i)}) = 1 - h(\mathbf{x}^{(i)})$ if $y^{(i)} = 0$
2. We can combine these into a single expression:
$$P(y^{(i)}|\mathbf{x}^{(i)}) = h(\mathbf{x}^{(i)})^{y^{(i)}} \cdot (1 - h(\mathbf{x}^{(i)}))^{1-y^{(i)}} \tag{17}$$
3. For the entire dataset, we want to maximize the product of these probabilities:
$$L(h) = \prod_{i=1}^{m} h(\mathbf{x}^{(i)})^{y^{(i)}} \cdot (1 - h(\mathbf{x}^{(i)}))^{1-y^{(i)}} \tag{18}$$
4. Taking the logarithm (to convert product to sum) and negating (to convert maximization to minimization) gives us the BCE loss function.

The BCE loss has several desirable properties:

- It's convex, making optimization easier
- It heavily penalizes confident and wrong predictions
- It has a probabilistic interpretation

5 Gradient Derivation for Logistic Regression

To optimize the parameters of our logistic regression model using gradient descent, we need to compute the gradients of the loss function with respect to the weights and bias. Surprisingly, these gradients have the same form as in linear regression:

5.1 Model and Loss Function

The logistic regression model is defined as:

$$h(x) = \sigma(wx + b) = \frac{1}{1 + e^{-(wx+b)}} \tag{19}$$

The binary cross-entropy loss function is:

$$L = -[y \log(h(x)) + (1 - y) \log(1 - h(x))] \tag{20}$$

5.2 Derivative with respect to w

We start by applying the chain rule:

$$\frac{\partial L}{\partial w} = -\left[y \frac{\partial}{\partial w} \log(h(x)) + (1 - y) \frac{\partial}{\partial w} \log(1 - h(x))\right] \tag{21}$$

$$= -\left[y \frac{1}{h(x)} \frac{\partial h(x)}{\partial w} + (1 - y) \frac{1}{1 - h(x)} \frac{\partial (1 - h(x))}{\partial w}\right] \tag{22}$$

$$= -\left[y \frac{1}{h(x)} \frac{\partial h(x)}{\partial w} - (1 - y) \frac{1}{1 - h(x)} \frac{\partial h(x)}{\partial w}\right] \tag{23}$$

Now, we calculate $\frac{\partial h(x)}{\partial w}$:

$$\frac{\partial h(x)}{\partial w} = \frac{\partial}{\partial w} \frac{1}{1 + e^{-(wx+b)}} \tag{24}$$

$$= \frac{e^{-(wx+b)}}{(1 + e^{-(wx+b)})^2} \cdot x \tag{25}$$

$$= h(x)(1 - h(x))x \tag{26}$$

Substituting back:

$$\frac{\partial L}{\partial w} = -\left[y \frac{1}{h(x)} h(x)(1 - h(x))x - (1 - y) \frac{1}{1 - h(x)} h(x)(1 - h(x))x\right] \tag{27}$$

$$= -[y(1 - h(x))x - (1 - y)h(x)x] \tag{28}$$

$$= -[(y - yh(x))x - (h(x) - yh(x))x] \tag{29}$$

$$= -[y - h(x)]x \tag{30}$$

$$= [h(x) - y]x \tag{31}$$

5.3 Derivative with respect to b

The derivation for b is similar, but $\frac{\partial(wx+b)}{\partial b} = 1$ instead of x:

$$\frac{\partial L}{\partial b} = [h(x) - y] \tag{32}$$

To summarize, the gradient for logistic regression is:

$$\frac{\partial L}{\partial w} = [h(x) - y]x \tag{33}$$

$$\frac{\partial L}{\partial b} = [h(x) - y] \tag{34}$$

5.4 Implementing Logistic Regression

Here's a basic implementation of logistic regression using gradient descent:

```
def optimize(X, y, lr, n_iter):
    m, n = X.shape
    w = np.zeros(n)
    b = 0

    for _ in range(n_iter):
        # Forward pass
        z = np.dot(X, w) + b
        y_hat = sigmoid(z)

        # Compute gradients
        dw = (1/m) * np.dot(X.T, (y_hat - y))
        db = (1/m) * np.sum(y_hat - y)

        # Update parameters
        w -= lr * dw
        b -= lr * db

    return w, b
```

This implementation uses vectorization for efficiency. The ‘optimize‘ function performs gradient descent to find the optimal weights and bias, while the ‘predict‘ function uses these parameters to make predictions on new data.

6 Conclusion

Logistic regression is a fundamental algorithm in machine learning, particularly for binary classification problems. It addresses the limitations of linear regression for classification tasks by using the sigmoid function to map inputs to probabilities. The key points to remember are:

- Logistic regression outputs probabilities between 0 and 1
- It uses the sigmoid function to squash the linear combination of inputs
- The decision boundary is linear, making it a linear classifier
- Binary Cross-Entropy is used as the loss function
- Despite its name, logistic regression is used for classification, not regression

Understanding logistic regression is crucial as it forms the basis for more complex models, including neural networks. In fact, each neuron in a neural network typically performs a logistic regression operation.

In the next lecture, we'll explore Multilayer Perceptrons (MLPs), which extend the ideas of logistic regression to create more powerful, non-linear classifiers.