INF721 - Deep Learning
L3: Linear Regression

Prof. Lucas N. Ferreira
Universidade Federal de Viçosa

2024/2

1 Introduction

Linear regression is a fundamental algorithm in machine learning and serves as a building block for neural networks. It is one of the simplest parametric algorithms and provides an excellent introduction to the concepts we'll use throughout this deep learning course.

1.1 Context in Deep Learning

While linear regression might seem basic compared to complex neural networks, understanding it is crucial because:

1. It introduces key concepts like hypothesis spaces, loss functions, and optimization.
2. The techniques used in linear regression, particularly gradient descent, are foundational for training neural networks.
3. A single neuron in a neural network can be viewed as performing linear regression.

2 Supervised Learning Review

Before diving into linear regression, let's briefly review supervised learning:

- Supervised learning involves learning from labeled data.
- The goal is to find a function that maps inputs to outputs based on example input-output pairs.
- We focus on two main types of supervised learning problems:
  - Regression: Predicting continuous values

  - Classification: Predicting discrete categories

- Linear regression is a regression algorithm, as the name suggests.

3 Linear Regression Problem Formulation

Let's consider a specific problem to illustrate linear regression: predicting house prices based on their size.

3.1 Dataset Description

We have a dataset of house sizes and their corresponding prices:

- Input feature (X): House size in square meters
- Output label (Y): House price in thousands of dollars
- The dataset contains 30 examples

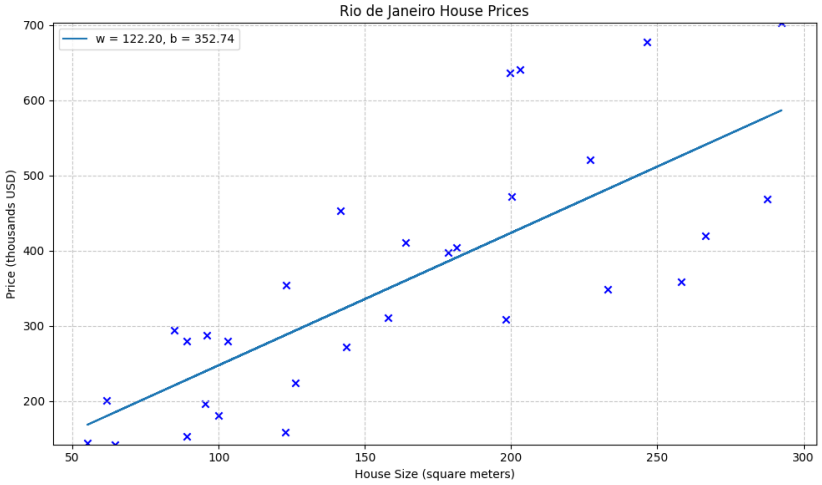

Figure 1: House size vs. price dataset

3.2 Goal of Linear Regression

The objective of linear regression is to find a linear function that best fits (see Figure 1) the given data and can predict the price of houses not in the dataset. This linear function will be used to estimate house prices based on their sizes.

4 Hypothesis Space

In linear regression, our hypothesis space H is defined as the set of linear functions:

$$h(x) = wx + b$$

Where:

- w is the weight (or slope)
- b is the bias (or y-intercept)
- x is the input feature (house size in our example)

4.1 Visualizing Linear Functions

To better understand the hypothesis space, let's look at some examples of linear functions:

- $h(x) = 0x + 1.5$: A horizontal line parallel to the x-axis at y = 1.5
- $h(x) = 0.5x + 0$: A line passing through the origin with a slope of 0.5
- $h(x) = 0.5x + 1$: A line with a y-intercept of 1 and a slope of 0.5

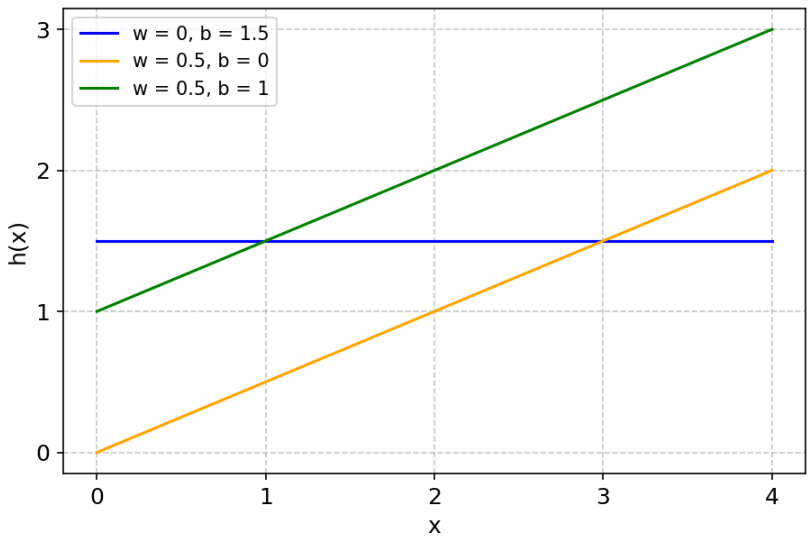

Figure 2: Examples of linear functions

5 Loss Function

To measure how well our linear function fits the data, we need a loss function. For linear regression, we use the Mean Squared Error (MSE) loss:

$$L(w, b) = \frac{1}{2m} \sum_{i=1}^{m} (h(x_i) - y_i)^2$$

Where:

- m is the number of examples in the dataset
- $h(x_i)$ is the predicted value for the i-th example
- y_i is the true value for the i-th example

The factor of $\frac{1}{2}$ is added for mathematical convenience when computing derivatives.

5.1 Intuition Behind MSE

The MSE loss function:

- Measures the average squared difference between predictions and true values
- Penalizes larger errors more heavily due to the squaring
- Always produces a non-negative value
- Reaches its minimum (zero) when predictions perfectly match true values

6 Optimization: Gradient Descent

Now that we have defined our hypothesis space and loss function, we need a way to find the optimal values for w and b that minimize the loss. This is where gradient descent comes in. Gradient descent is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function. It's one of the most important algorithms in deep learning, used for training neural networks.

6.1 Gradient Descent Algorithm

The basic idea of gradient descent is:

1. Start with initial values for the parameters (w and b)
2. Compute the gradient of the loss function with respect to each parameter
3. Update the parameters in the opposite direction of the gradient

4. Repeat steps 2-3 until convergence

Mathematically, for each iteration t:

$$w_{t+1} = w_t - \alpha \frac{\partial L}{\partial w}$$
$$b_{t+1} = b_t - \alpha \frac{\partial L}{\partial b}$$

Where α is the learning rate, a hyperparameter that controls the step size of each iteration.

6.2 Visualizing Gradient Descent

To better understand how gradient descent works, let's visualize it on a contour plot of the loss function:

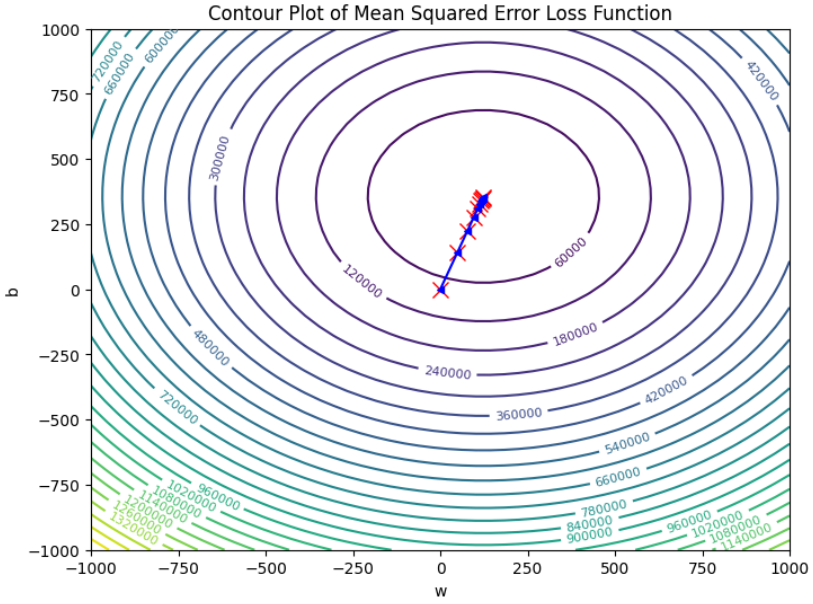

Figure 3: Gradient descent optimization

In this visualization:

- The x and y axes represent the parameters w and b
- The contour lines represent the loss function values
- The red crosses represent the loss values $L(w, b)$ at different parameter values w, b
- The blue arrows show the direction of steepest descent. We can see the optimization path converging towards the minimum.

6.3 Impact of Learning Rate

The learning rate α is a crucial hyperparameter in gradient descent that significantly affects the optimization process. It determines the step size at each iteration while moving toward a minimum of the loss function. Choosing the appropriate learning rate is essential for efficient and effective optimization.

- **Large learning rate**

  Pros: Faster initial convergence

  Cons: May overshoot the minimum, causing divergence or oscillation

- **Small learning rate**

  Pros: More precise convergence, less likely to overshoot

  Cons: Slower convergence, may get stuck in local minima

- **Optimal learning rate**

  Balances speed and precision. Allows for efficient convergence to the global minimum

7 Gradient Descent for Linear Regression

Now that we've introduced the gradient descent algorithms, let's apply it to our linear regression problem. To do that, we need derive the partial derivatives of the Mean Squared Error (MSE) loss function with respect to w and b. In this section we will do that step-by-step. If you need a calculus review, please refer to the appendix. First, recall our MSE loss function:

$$L(w,b) = \frac{1}{2m}\sum_{i=1}^{m}(wx^{(i)} + b - y^{(i)})^2]$$

7.1 Partial Derivative with respect to w

$$\begin{aligned}
\frac{\partial L}{\partial w} &= \frac{\partial}{\partial w}\left[\frac{1}{2m}\sum_{i=1}^{m}(wx^{(i)} + b - y^{(i)})^2\right] \\
&= \frac{1}{2m}\sum_{i=1}^{m}\frac{\partial}{\partial w}\left[(wx^{(i)} + b - y^{(i)})^2\right] \quad \text{(Linearity rule)} \\
&= \frac{1}{2m}\sum_{i=1}^{m}2(wx^{(i)} + b - y^{(i)})\frac{\partial}{\partial w}(wx^{(i)} + b - y^{(i)}) \quad \text{(Chain rule)} \\
&= \frac{1}{2m}\sum_{i=1}^{m}2(wx^{(i)} + b - y^{(i)})x^{(i)} \quad \text{(Cancel out 2)} \\
&= \frac{1}{m}\sum_{i=1}^{m}(wx^{(i)} + b - y^{(i)})x_i = \frac{1}{m}\sum_{i=1}^{m}(h(x_i) - y^{(i)})x_i
\end{aligned}$$

7.2 Partial Derivative with respect to b

$$\begin{aligned}
\frac{\partial L}{\partial b} &= \frac{\partial}{\partial b}\left[\frac{1}{2m}\sum_{i=1}^{m}(wx^{(i)}+b-y^{(i)})^2\right] \\
&= \frac{1}{2m}\sum_{i=1}^{m}\frac{\partial}{\partial b}\left[(wx^{(i)}+b-y^{(i)})^2\right] \quad \text{(Linearity rule)} \\
&= \frac{1}{2m}\sum_{i=1}^{m}2(wx^{(i)}+b-y^{(i)})\frac{\partial}{\partial b}(wx_i+b-y^{(i)}) \quad \text{(Chain rule)} \\
&= \frac{1}{2m}\sum_{i=1}^{m}2(wx^{(i)}+b-y^{(i)})\cdot 1 \quad \text{(Cancel out 2)} \\
&= \frac{1}{m}\sum_{i=1}^{m}(wx^{(i)}+b-y^{(i)}) = \frac{1}{m}\sum_{i=1}^{m}(h(x^{(i)})-y^{(i)})
\end{aligned}$$

To summarize, we have derived the partial derivatives $\frac{\partial L}{\partial w}$, $\frac{\partial L}{\partial b}$, which are the components of the gradient vector $\nabla L = (\frac{\partial L}{\partial w}, \frac{\partial L}{\partial b})$, that we use in the gradient descent algorithm to update our parameters w and b.

7.3 Gradient Descent Algorithm for Linear Regression

Here's the complete gradient descent algorithm for linear regression:

```
def optimize(x, y, num_iterations=1000, learning_rate=0.01):
    m = len(y)

    w, b = 0, 0
    for _ in range(num_iterations):
        # Compute predictions
        y_pred = w * x + b

        # Compute gradients
        dw = (1/m) * np.sum((y_pred - y) * x)
        db = (1/m) * np.sum(y_pred - y)

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

    return w, b
```

8 Conclusion

In this lecture, we've covered the fundamentals of linear regression:

- Problem formulation and its relevance to deep learning
- Hypothesis space of linear functions
- Mean Squared Error loss function
- Gradient descent optimization
- Implementation in Python

Understanding these concepts is crucial as we move forward in the course. Linear regression serves as a stepping stone to more complex models, including neural networks. The optimization techniques we've learned, particularly gradient descent, will be applied repeatedly as we delve deeper into deep learning algorithms.

In the next lecture, we'll explore logistic regression, which extends these ideas to classification problems and introduces the concept of non-linear activation functions.

A Appendix: Calculus Review

To understand how we compute the gradients for gradient descent, we need to review some key concepts from calculus.

A.1 Derivatives

The derivative of a function f(x) at a point x = a is defined as:

$$f'(a) = \lim_{h \to 0} \frac{f(a+h) - f(a)}{h}$$

Intuitively, the derivative represents the slope of the tangent line to the function f at that point a.

A.2 Partial Derivatives

For functions of multiple variables, we use partial derivatives. The partial derivative of $f(x, y)$ with respect to x is:

$$\frac{\partial f}{\partial x} = \lim_{h \to 0} \frac{f(x+h, y) - f(x, y)}{h}$$

A.3 Gradient Vector

The gradient of a multivariate function $f(x_1, x_2, ..., x_d)$ is a vector of its partial derivatives:

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, ..., \frac{\partial f}{\partial x_d} \right)$$

The gradient vector points in the direction of steepest ascent of the function f.

A.4 Derivative Rules

Some key derivative rules include:

- Derivative of a constant: $\frac{d}{dx} c = 0$
- Power rule: $\frac{d}{dx} x^n = nx^{n-1}$
- Derivative of a sum: $\frac{d}{dx}(f(x) + g(x)) = f'(x) + g'(x)$
- Derivative of a product: $\frac{d}{dx}(f(x)g(x)) = f'(x)g(x) + f(x)g'(x)$
- Derivative of a quotient: $\frac{d}{dx}\left(\frac{f(x)}{g(x)}\right) = \frac{f'(x)g(x) - f(x)g'(x)}{(g(x))^2}$
- Derivative of an exponential: $\frac{d}{dx} e^{ax} = ae^{ax}$

- Derivative of a logarithm: $\frac{d}{dx}\log_a(x) = \frac{1}{x\ln(a)}$
- Linearity rule: $\frac{d}{dx}(af(x) + bg(x)) = a\frac{d}{dx}f(x) + b\frac{d}{dx}g(x)$
- Chain rule: $\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}$
  This rule is fundamental in backpropagation for neural networks.