Lecture: Linear Regression in Machine Learning

1. Introduction to Linear Regression

Linear regression is one of the foundational techniques in statistics and machine learning. It is used to model and quantify the relationship between one or more **input variables (features)** and a **continuous output variable (label)**.

In a machine learning context, linear regression answers the question:

> *How does a change in the input features affect the predicted output, and how can we learn this relationship from data?*

Linear regression is widely used because it is:

- Interpretable
- Computationally efficient
- A building block for more advanced models

---

2. Problem Setup: Features and Labels

In supervised learning, data consists of:

- **Features (x)**: Input variables used for prediction
- **Label (y)**: The value we want to predict

Example: Predicting Fuel Efficiency

Suppose we want to predict a car's fuel efficiency (miles per gallon) based on its weight.

Pounds (in 1000s)	**Miles per Gallon**
3.5	18
3.69	15

3.44	18
3.43	16
4.34	15
4.42	14
2.37	24

When plotted, these points show a **negative linear relationship**: as weight increases, fuel efficiency decreases.

---

3. The Linear Regression Model

Algebraic Form

In basic algebra, a line is defined as:

y=mx+cy = mx + cy=mx+c

Where:

- mmm is the slope
- ccc is the y-intercept

Machine Learning Form

In machine learning, the linear regression model is written as:

y^=b+wx\hat{y} = b + wxy^=b+wx

Where:

- y^\hat{y}y^: Predicted label
- bbb: **Bias** (intercept)
- www: **Weight** (slope)
- xxx: Feature

The bias and weight are collectively called **model parameters**.

4. Interpretation of Bias and Weight

- **Weight (w)**
  Measures how much the prediction changes when the feature changes by one unit.
- **Bias (b)**
  Represents the predicted value when all feature values are zero.
  It shifts the regression line up or down.

Example Model

If the learned model is:

y^=34−4.6x\hat{y} = 34 - 4.6xy^=34−4.6x

Then:

- Bias = 34
- Weight = –4.6

A 4,000-pound car (x=4x = 4x=4) would have:

y^=34−4.6(4)=15.6\hat{y} = 34 - 4.6(4) = 15.6y^=34−4.6(4)=15.6

---

5. Training a Linear Regression Model

Training means **learning the best values of bias and weights** from data.

What Does “Best” Mean?

“Best” is defined using a **loss function**, which measures how far predictions are from actual values.

---

6. Loss Function

A loss function quantifies prediction error.

Common Loss Function: Mean Squared Error (MSE)

MSE=1n∑i=1n(y^i−yi)2\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2MSE=n1i=1∑n(y^i−yi)2

Properties:

- Penalizes large errors more heavily
- Differentiable, making it suitable for optimization

The objective of training is to **minimize the loss**.

---

7. Gradient Descent Optimization

Gradient descent is an iterative algorithm used to find parameter values that minimize the loss function.

How Gradient Descent Works

1. Initialize bias and weights randomly
2. Compute predictions
3. Calculate loss
4. Compute gradients (partial derivatives of loss w.r.t. parameters)
5. Update parameters in the opposite direction of the gradient

w=w−α∂L∂ww = w - \alpha \frac{\partial L}{\partial w}w=w−α∂w∂L b=b−α∂L∂bb = b - \alpha \frac{\partial L}{\partial b}b=b−α∂b∂L

Where:

- α\alphaα is the **learning rate**

---

8. Hyperparameters in Linear Regression

Hyperparameters are **not learned from data** but are set before training.

Key hyperparameters include:

- Learning rate
- Number of training iterations (epochs)
- Batch size (for batch or stochastic gradient descent)

Hyperparameter Tuning

Improper tuning can cause:

- **Underfitting** (learning rate too small, too few iterations)
- **Divergence** (learning rate too large)

Efficient tuning ensures faster convergence and better performance.

---

9. Linear Regression with Multiple Features

Most real-world problems involve multiple features.

Model Equation

y^=b+w1x1+w2x2+⋯+wnxn\hat{y} = b + w_1x_1 + w_2x_2 + \dots + w_nx_ny^=b+w1x1+w2x2+⋯+wnxn

Each feature has:

- Its own weight
- Its own contribution to the prediction

Example: Fuel Efficiency Prediction

Features may include:

- Weight
- Engine displacement
- Acceleration
- Number of cylinders

- Horsepower

Each feature captures a different linear relationship with the label.

---

10. Key Parameters Updated During Training

During training:

- **Bias and weights are updated**
- Feature values and labels remain fixed
- Predictions change as parameters change

---

11. Bias as a Distribution (Advanced Perspective)

Deterministic View (Basic ML)

In standard linear regression:

- Bias is treated as a single scalar value
- It is optimized via gradient descent
- Once trained, it is fixed

Probabilistic and Statistical View

In a probabilistic framework, linear regression is often written as:

y=b+wx+ϵy = b + wx + \epsilony=b+wx+ϵ

Where:

- ϵ\epsilonϵ is a random error term
- Typically assumed to follow a normal distribution:
  ϵ∼N(0,σ2)\epsilon \sim \mathcal{N}(0, \sigma^2)ϵ∼N(0,σ2)

Bias as a Random Variable

In **Bayesian linear regression**, the bias is no longer a fixed constant. Instead:

b~N(μb,σb2)b \sim \mathcal{N}(\mu_b, \sigma_b^2)b~N(μb,σb2)

Implications:

- Bias has a **distribution**, not a single value
- Reflects uncertainty in the intercept
- Especially useful when data is limited or noisy

Why This Matters

- Captures model uncertainty
- Produces predictive distributions instead of point estimates
- Allows regularization through priors
- Improves robustness and interpretability in real-world systems

Practical Interpretation

Instead of saying:

> "The bias is 34"

We say:

> "The bias is most likely around 34, but with some uncertainty governed by its variance."

This perspective is foundational for:

- Bayesian inference
- Probabilistic machine learning
- Advanced regression techniques

---

12. Summary

- Linear regression models the relationship between features and a continuous label
- The model is defined by **weights** and **bias**
- Training minimizes a loss function using gradient descent
- Hyperparameters control training efficiency
- Multiple features improve predictive power
- In advanced settings, bias can be modeled as a **distribution**, enabling uncertainty-aware predictions

Lecture: Loss Functions in Linear Regression

1. What Is Loss?

In machine learning, **loss** is a numerical value that quantifies how incorrect a model's predictions are compared to the actual values. It measures the **distance** between the predicted output and the true label.

The central objective of training a regression model is:

> **To minimize loss**, ideally bringing it as close to zero as possible.

Loss provides a precise, mathematical way to evaluate model performance during training and to guide parameter updates.

---

2. Loss as Distance, Not Direction

Loss measures **how far** a prediction is from the true value, not **whether it is above or below** it.

For example:

- Actual value: 5
- Predicted value: 2

The raw error is:

2−5=−32 - 5 = -32−5=−3

However, the sign is irrelevant. What matters is the **magnitude of the error**, which is 3.

To eliminate the sign, loss functions typically:

1. Take the **absolute value**, or
2. **Square** the difference

---

3. Visual Intuition for Loss

Geometrically, loss can be visualized as **vertical distances** between the data points and the regression line.

- Each data point contributes its own loss
- The total loss is the aggregation of these distances across all examples
- A better-fitting model produces shorter “loss lines”

---

4. Individual Loss vs Aggregated Loss

Loss can be computed:

- **Per example** (individual loss)
- **Across the dataset** (aggregate loss)

When training on multiple examples, it is standard practice to **average** the loss so that:

- Loss does not scale with dataset size
- Comparisons across models are meaningful

---

5. Common Loss Functions in Linear Regression

5.1 L1 Loss (Absolute Loss)

Definition
The absolute difference between the prediction and the actual value.

L1=∣y−y^∣L_1 = |y - \hat{y}|L1=∣y−y^∣

This measures error directly in the same units as the label.

5.2 Mean Absolute Error (MAE)

Definition
The average L1 loss across NNN examples.

MAE=1N∑i=1N∣yi−y^i∣\text{MAE} = \frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}_i|MAE=N1i=1∑N∣yi−y^i∣

Properties

- Robust to outliers
- Highly interpretable
- Linear penalty for errors

5.3 L2 Loss (Squared Loss)

Definition
The square of the difference between the prediction and the actual value.

L2=(y−y^)2L_2 = (y - \hat{y})^2L2=(y−y^)2

Squaring amplifies larger errors.

5.4 Mean Squared Error (MSE)

Definition
The average L2 loss across NNN examples.

MSE=1N∑i=1N(yi−y^i)2\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2MSE=N1i=1∑N(yi−y^i)2

Properties

- Penalizes large errors heavily
- Smooth and differentiable
- Commonly used during training

5.5 Root Mean Squared Error (RMSE)

Definition
The square root of MSE.

RMSE=MSE\text{RMSE} = \sqrt{\text{MSE}}RMSE=MSE

Purpose

- Converts error back to the original units of the label
- More interpretable than MSE

---

6. Effect of Squaring: L1 vs L2 Intuition

Consider two prediction errors:

- Error A = 1
- Error B = 2

Loss Type	Error A	Error B
L1	1	2
L2	1	4

Squaring makes larger errors **disproportionately more costly**, which has important implications for model behavior.

---

7. Example: Calculating L2 Loss

Suppose:

- Feature value: 2.37 (2,370 pounds)
- Actual fuel efficiency: 26 mpg
- Model prediction: 23.1 mpg

Step 1: Compute the error

26−23.1=2.926 - 23.1 = 2.926−23.1=2.9

Step 2: Square the error

(2.9)2=8.41(2.9)^2 = 8.41(2.9)2=8.41

The **L2 loss for this single data point is 8.41**.

---

8. Choosing a Loss Function

The choice of loss function determines **how the model reacts to errors**, especially outliers.

8.1 Understanding Outliers

Outliers can appear in two ways:

1. **Unusual feature values**
   Example: An 8,000-pound car
2. **Unusual prediction errors**
   Example: A typical car weight with extremely high or low fuel efficiency

---

8.2 Behavior of MSE

- Large errors are heavily penalized
- The model is pulled toward outliers
- Produces smoother gradients for optimization

Use MSE when:

- Outliers are meaningful and should influence the model
- Large errors are especially costly
- Training stability is important

8.3 Behavior of MAE

- All errors are treated linearly
- Outliers do not dominate training
- More robust to noisy data

Use MAE when:

- Dataset contains significant outliers
- You want an interpretable “average error”
- Robustness matters more than sensitivity

---

9. Comparative Example: MSE vs MAE

Scenario A

- 6 points lie exactly on the line
- 4 points are 1 unit away

Total MSE is relatively small.

Scenario B

- 8 points lie exactly on the line
- 2 points are 2 units away

Despite fewer errors, the **MSE is higher** because:

22=4(four times worse than an error of 1)2^2 = 4 \quad \text{(four times worse than an error of 1)}22=4(four times worse than an error of 1)

This demonstrates how MSE emphasizes **error magnitude over error frequency**.

---

10. Loss vs Metric

- **Loss** is used during training to optimize the model
- **Metrics** are often used for evaluation and reporting

In practice:

- Train with MSE (smooth optimization)
- Report MAE or RMSE (human interpretability)

---

11. Summary

- Loss quantifies how wrong a model's predictions are
- It measures distance, not direction
- MAE and MSE are the most common loss functions in linear regression
- MSE penalizes large errors more heavily than MAE
- Choice of loss function directly affects model behavior
- Understanding loss is essential for interpreting and training regression models

Lecture: Gradient Descent in Linear Regression

1. Introduction

Gradient descent is the core optimization technique used to train linear regression models. Its purpose is to **iteratively find the values of the model parameters—weights and bias—that minimize the loss function**.

Because linear regression uses convex loss functions, gradient descent is guaranteed to converge to the best possible solution when configured correctly.

---

2. Role of Gradient Descent

Gradient descent does **not**:

- Choose the loss function
- Remove outliers
- Modify the dataset

Instead, gradient descent:

> **Finds the weights and bias that minimize the chosen loss function.**

This is achieved through repeated, incremental updates based on the direction of steepest loss reduction.

---

3. Model Initialization

Training begins by assigning:

- Randomized weights near zero
- A bias near zero

This initial model is usually poor and produces high loss, but it provides a starting point for optimization.

---

4. The Gradient Descent Algorithm

Gradient descent follows a fixed iterative loop:

1. **Compute predictions** using current weights and bias
2. **Calculate loss** using the chosen loss function
3. **Compute gradients** (direction of steepest loss increase)
4. **Update parameters** in the opposite direction
5. **Repeat** for a predefined number of iterations or until convergence

Mathematically, the update rules are:

w=w−α∂L∂ww = w - \alpha \frac{\partial L}{\partial w}w=w−α∂w∂L b=b−α∂L∂bb = b - \alpha \frac{\partial L}{\partial b}b=b−α∂b∂L

Where:

- α\alphaα is the learning rate
- LLL is the loss function

---

5. Iterations and Learning Dynamics

Each pass through the training loop is called an **iteration**.

- Early iterations cause large reductions in loss
- Later iterations result in smaller improvements
- Eventually, parameter updates become negligible

This diminishing improvement signals **model convergence**.

---

6. Loss Curves and Convergence

Loss Curve

A **loss curve** plots:

- **Loss** on the y-axis

- **Iterations** on the x-axis

A typical loss curve shows:

- A steep initial decline
- A gradual flattening
- A near-horizontal line at convergence

When the loss curve flattens, additional training yields minimal improvement.

---

7. Visualizing Model Convergence

To understand convergence, consider snapshots of the regression line at different iterations:

Early Training

- Model tilts away from the data
- Loss lines are long
- Predictions are inaccurate

Mid Training

- Model begins aligning with the data
- Loss decreases significantly
- Weights and bias move closer to optimal values

Late Training

- Model fits the data well
- Loss lines are short
- Further updates are minimal

This progression directly links **parameter updates** to **loss reduction**.

---

8. Convex Loss Functions

Definition of Convexity

A function is **convex** if:

- It has a single global minimum
- Any line segment between two points lies above the function

Convexity in Linear Regression

For linear regression with losses such as MSE:

- The loss surface is convex
- There are no local minima
- Gradient descent always moves toward the global minimum

---

9. Loss Surface Visualization

For a model with one feature:

- x-axis: weight
- y-axis: bias
- z-axis: loss

The resulting surface resembles a **bowl**.

Gradient descent behaves like:

> A ball rolling downhill until it reaches the lowest point.

---

10. Convergence Behavior

A model is said to **converge** when:

- The gradients approach zero
- Parameter updates become very small
- Loss no longer decreases meaningfully

Important observations:

- The model almost never reaches the exact mathematical minimum
- Instead, it settles very close to it
- The minimum loss is usually **not zero**

---

11. Best-Fit Model

The weights and bias at convergence produce:

- The lowest achievable loss for the dataset
- The best linear approximation of the data

For example:

- Weight = –5.44
- Bias = 35.94

These values define the optimal model for that dataset.

---

12. Why Gradient Descent Works So Well for Linear Regression

Gradient descent is particularly effective because:

- The loss surface is convex

- Gradients are well-defined
- There is a single global optimum

This combination ensures:

- Stable training
- Predictable convergence
- Reliable results

---

13. Key Terms

- **Gradient descent**: Optimization algorithm that minimizes loss
- **Iteration**: One update step of the algorithm
- **Loss curve**: Plot showing loss over iterations
- **Convex function**: Function with a single global minimum
- **Convergence**: Point where further training yields negligible improvement

---

14. Summary

- Gradient descent is an iterative optimization method
- It updates weights and bias to reduce loss
- Loss curves help diagnose convergence
- Convex loss functions guarantee a global minimum
- Convergence produces the best possible linear model

Lecture: Hyperparameters in Linear Regression

1. Introduction

In machine learning, **hyperparameters** are external configuration values that control **how a model is trained**, rather than what the model represents. Unlike model parameters, hyperparameters are **set by the practitioner before training begins**.

In linear regression, hyperparameters strongly influence:

- Training speed
- Stability of convergence
- Final model quality

Selecting appropriate hyperparameters is essential for building effective models.

---

2. Hyperparameters vs Parameters

Parameters

- Learned from data
- Updated during training
- Define the model itself

Examples

- Weights
- Bias

Hyperparameters

- Set manually before training

- Control the learning process

Examples

- Learning rate
- Batch size
- Number of epochs

---

3. Core Hyperparameters in Linear Regression

The three most commonly used hyperparameters are:

1. Learning rate
2. Batch size
3. Epochs

Each influences training in a distinct way, but they interact closely.

---

4. Learning Rate

Definition

The **learning rate** is a positive real number that determines the **step size** taken during each gradient descent update.

θnew=θold−α∇L\theta_{\text{new}} = \theta_{\text{old}} - \alpha \nabla Lθnew=θold−α∇L

Where:

- α\alphaα is the learning rate
- ∇L\nabla L∇L is the gradient of the loss

Effect of Learning Rate

The learning rate controls how aggressively the model updates its parameters:

- **Small learning rate** → slow but stable learning
- **Large learning rate** → fast but unstable learning

Learning Rate Scenarios

Ideal Learning Rate

- Loss decreases rapidly at first
- Then gradually flattens
- Model converges efficiently

Learning Rate Too Small

- Loss decreases very slowly
- Requires many iterations
- Training becomes inefficient

Learning Rate Too Large

- Loss oscillates or diverges
- Model overshoots the minimum
- Training may never converge

Key Insight

There is **no universal ideal learning rate**.
The optimal learning rate is:

- Dataset-dependent
- Model-dependent

- Often found through experimentation

---

5. Batch Size

Definition

Batch size is the number of training examples used to compute the gradient and update the model parameters in one iteration.

Full-Batch Gradient Descent

- Batch size = entire dataset
- One update per epoch
- Stable but computationally expensive

Stochastic Gradient Descent (SGD)

- Batch size = 1
- One update per example
- Very fast updates
- Produces noisy loss curves

Noise arises because each update is based on a single random example.

Mini-Batch Gradient Descent

- Batch size between 1 and dataset size
- Gradients averaged over the batch
- Balance between stability and efficiency

Mini-batch SGD is the most commonly used approach in practice.

Batch Size and Noise

- Small batches → noisy but exploratory updates
- Large batches → smoother but less adaptive updates

A moderate amount of noise can:

- Improve generalization
- Help avoid poor local behavior in complex models

---

6. Epochs

Definition

An **epoch** is one complete pass through the entire training dataset.

Relationship Between Epochs and Iterations

Iterations per epoch=Number of training examplesBatch size\text{Iterations per epoch} = \frac{\text{Number of training examples}}{\text{Batch size}}Iterations per epoch=Batch sizeNumber of training examples

Example:

- Dataset size = 1,000
- Batch size = 100
- Iterations per epoch = 10

Role of Epochs

- More epochs allow the model to refine parameters

- Too few epochs → underfitting
- Too many epochs → wasted computation (or overfitting in complex models)

The number of epochs is chosen based on:

- Loss convergence
- Validation performance
- Computational constraints

---

7. Interaction Between Batch Size and Epochs

Batch size determines **how often** parameters are updated.

Batch Type	Updates per Epoch
Full batch	1
SGD	Number of examples
Mini-batch	Dataset size ÷ batch size

Example:

- Dataset = 1,000
- Batch size = 100
- Epochs = 20
  → Total updates = 200

---

8. Hyperparameter Selection Guidelines

Learning Rate

- Start small and increase gradually

- Monitor loss curves for divergence or slow convergence

Batch Size

- Smaller batches for limited memory
- Larger batches for smoother updates
- Choose based on dataset size and hardware

Epochs

- Train until loss converges
- Use early stopping if applicable

---

9. Common Misconceptions

- Doubling the learning rate can **slow down training** by causing instability
- Larger batches are not universally better
- There is no “best” batch size or learning rate for all problems

Hyperparameter tuning is inherently **problem-specific**.

---

10. Summary

- Hyperparameters control how training occurs
- Learning rate determines update magnitude
- Batch size determines update frequency and noise
- Epochs determine how many times the data is reused
- Proper tuning balances efficiency, stability, and performance

11. Key Terms

- **Hyperparameter**: External training configuration
- **Learning rate**: Step size in gradient descent
- **Batch size**: Number of examples per update
- **Epoch**: One full pass through the dataset
- **Mini-batch SGD**: Compromise between SGD and full-batch training