Medical Diagnostics

Tutorial 5 (1): Modelling data with simple regression models

Spring 2025

Faculty of Biotechnology and Food Engineering
Technion Israel Institute of Technology

TA: Mattan Hoory

Table of contents

0 From learning to machine learning

0.1 What is learning?

- **Learning (general):** Acquiring skill with experience accumulated from observations.
  - Observations → learning → skill
- **Machine learning (ML):** Acquiring skill with experience accumulated/computed from data.
  - Data → ML algorithm → ML skill
- ML automates the process of learning from data to make predictions or decisions.

0.2 What is a skill?

- Improves some performance measure (e.g. prediction accuracy)
- **Machine learning:** Improving some performance measure with experience computed from data
  - Data → ML algorithm → improved performance measure
- An application in diagnostics:
  - Patient data → ML → Early detection of disease

0.3 Why use machine learning?

- ‘Define’ trees and implement an algorithm: **difficult**
- Learn from data: **a 3-year-old can do it**
- ML can be easier to build (if data is available) and provide an alternative to complex algorithms

0.4 When to use machine learning?

1. Exists some ‘underlying pattern’ to be learned —so ‘performance measure’ can be improved
2. But no programmable (easy) definition—so ‘ML’ is needed
3. Somehow there is data about the pattern—so ML has some ‘inputs’ to learn from

1 Supervised learning setup

1.1 Basic notations

- **Input space:** $\mathcal{X}$
- **Output space:** $\mathcal{Y}$
- **Training examples:** $D = \{(x_i, y_i)\}_{i=1}^{N}$
- **Unknown target function:** $f : \mathcal{X} \to \mathcal{Y}$
- **Learning algorithm:** $\mathcal{A}$
- **Hypothesis set:** $\mathcal{H}$ (set of possible models)
- **Final hypothesis:** $g \approx f$

1.2 Statistics vs ML vs Artificial Intelligence (AI)

- **Statistics:**
  - Use data to make inferences about a population/process
- **ML:**
  - Use data to compute hypothesis g that approximates the target function f
- **AI:**
  - Compute something that shows intelligent behavior

2 Learning flow

1. **Collect data:** $D = \{(x_i, y_i)\}$
2. **Choose hypothesis set:** $\mathcal{H}$ (e.g., linear functions)
3. **Select learning algorithm:** $\mathcal{A}$
4. **Train:** Use $\mathcal{A}$ on D to find $g \in \mathcal{H}$
5. **Evaluate:** Assess g on new data
6. **Deploy:** Use g for predictions

2.1 Case study: Gene expression inference

Based on: Chen, Y., Li, Y., Narayan, R., Subramanian, A., & Xie, X. (2016). Gene expression inference with deep learning. Bioinformatics (Oxford, England), 32(12), 1832–1839.

- **Problem:** Sequencing of the human genome is expensive and time-consuming
- **Motivation:** Despite the large number of genes ($\approx 22k$) across the whole human genome, most of their expression profiles are known to be highly correlated.

2.1 Case study: Gene expression inference

- **Goal:** Predict the expression of thousands of target genes from a subset of landmark genes
- **Inputs:** $x \in \mathbb{R}^{943}$ (landmark gene expression levels)
- **Outputs:** $y \in \mathbb{R}^k$ (target gene expression levels, $k = 9,520$ to $21,290$)
- **Datasets:** GEO (microarray) and GTEx (RNA-seq)

3 A simple hypothesis set: linear regression (LR)

3.1 Simple LR (1-1)

- **Model:** $h_\theta(x) = \theta_0 + \theta_1 x$
- **Parameters(θ) :** $a = \theta_1$ (weights), $b = \theta_0$ (bias)
- How to choose θ_1 and θ_0?
- **Goal:** Find θ_1 and θ_0 that minimize the difference between predicted and true outputs

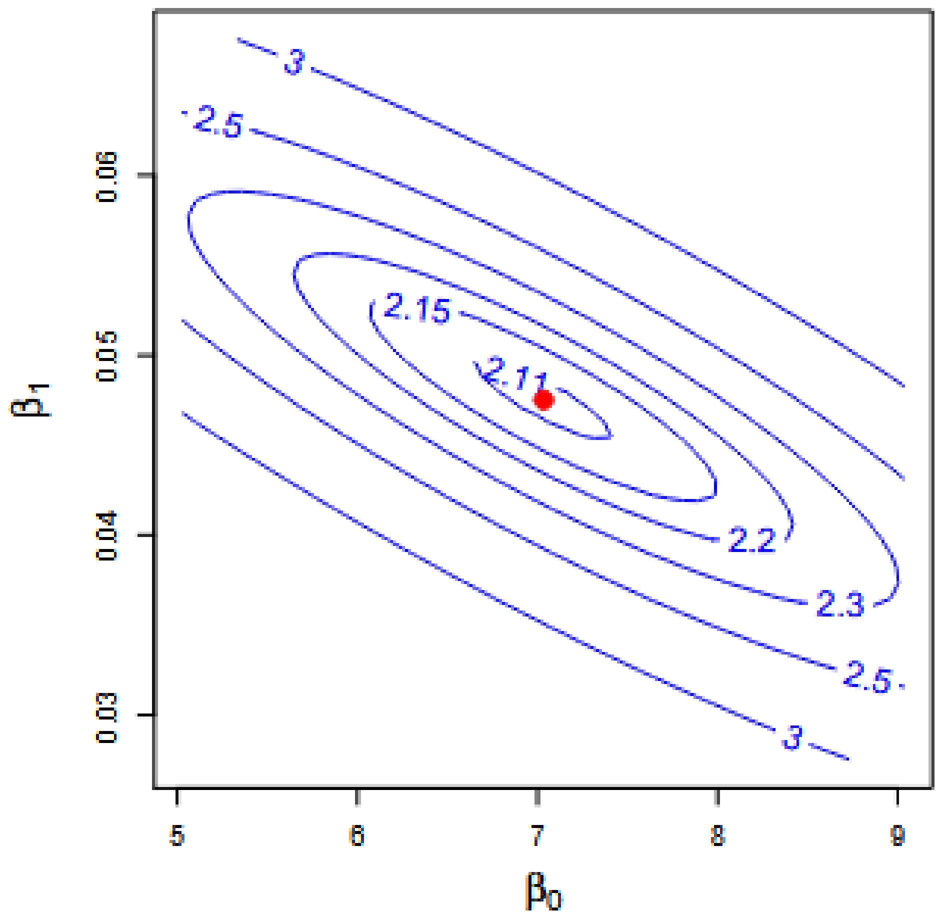
3.2 Loss function for LR: mean squared error (MSE)

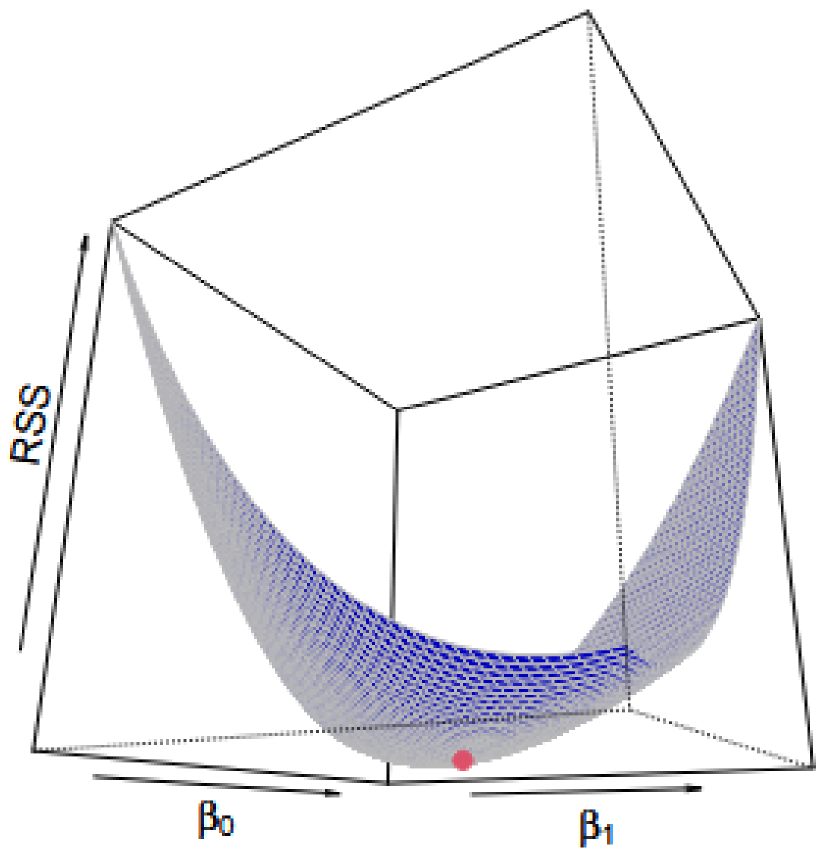
- **Loss function:** Quantifies how well the LR model fits the data.
- **Mean squared error (MSE):**

$$\mathcal{L}_{\text{MSE}}(\theta) = \frac{1}{N}\sum_{i=1}^{N}(h_\theta(x_i) - y_i)^2$$

- **Goal:** Find θ that minimizes $\mathcal{L}_{\text{MSE}}(\theta)$

3.3 MSE: visualization





- What is a good value for L ?

3.4 Multiple LR (many-1)

- **General case (multiple features):**

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_d x_d$$

- **Vector notation:**
  - Let $x = [x_1, x_2, \ldots, x_d]^T$ (feature vector)
  - Let $w = [\theta_1, \theta_2, \ldots, \theta_d]^T$ (weights)
  - θ_0 is the bias (intercept)

$$h_\theta(x) = w^T x + b$$

  - Here, $b = \theta_0$

3.5 Bias term in LR

- **Bias trick:**
  - Augment x with a constant 1: $x' = [1, x_1, x_2, \dots, x_d]^T$
  - Augment w with θ_0: $w' = [\theta_0, \theta_1, \dots, \theta_d]^T$
  - Now, $h_\theta(x) = (w')^T x'$
  - This allows us to write the model as a single dot product, including the bias term.
- What about Many-Many LR? (HW1!)

3.7 Multiple LR in practice

Multiple LR: overview

Scikit-learn API:

```python
from sklearn.linear_model import LinearRegression
model = LinearRegression()
# model.fit(X.numpy(), y.numpy())
# print(model.coef_)
```

Documentation: Scikit-learn Linear Regression

4 Gradient descent (GD)

4.1 Gradient descent: overview

- **Goal:** Minimize a loss function $\mathcal{L}(\theta)$ over parameters θ
- **Used for:** Models where closed-form solution is not feasible
- **Learning rate** η controls step size (static (?))

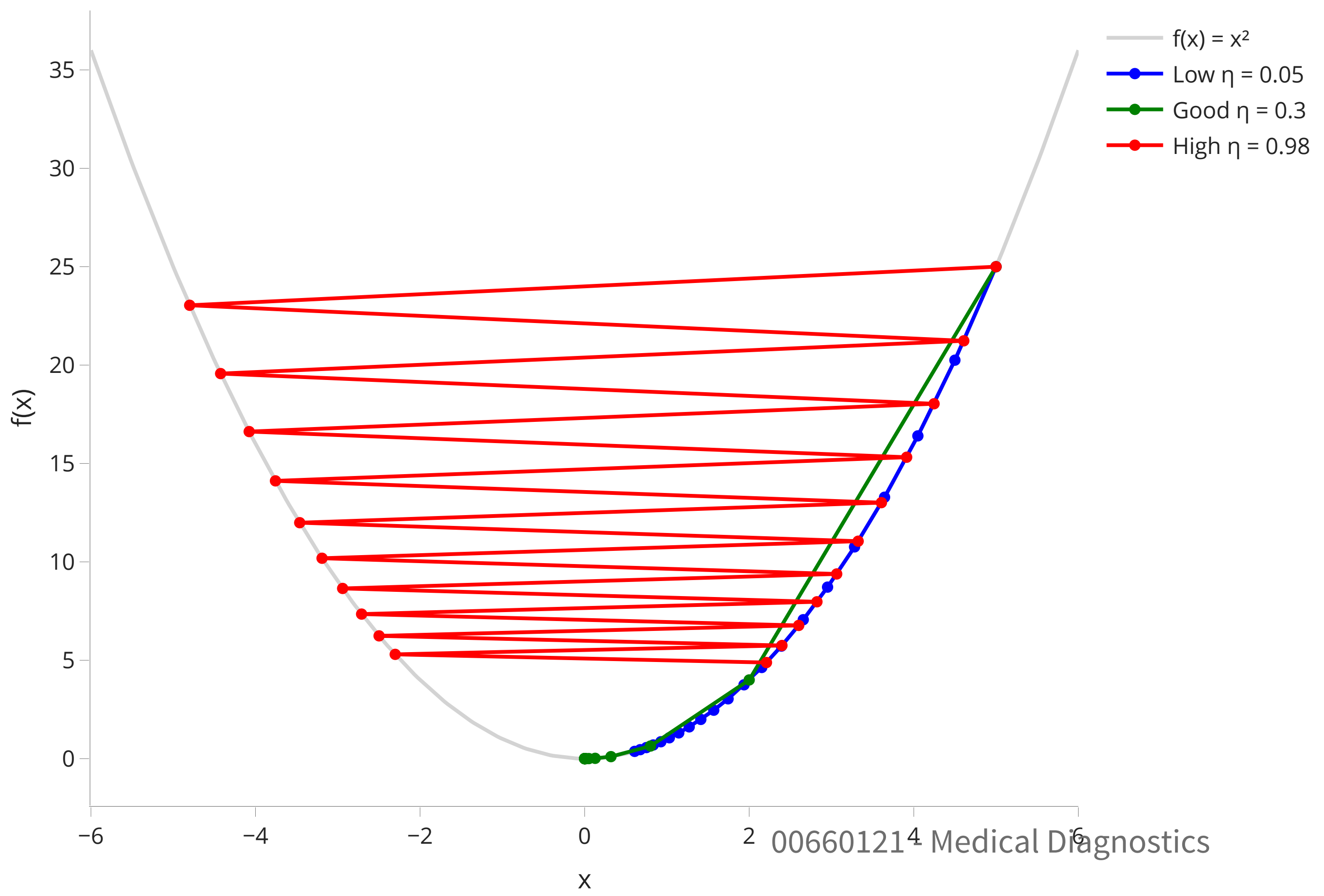
4.2 Gradient descent: algorithm

- **Algorithm:**
  1. Initialize θ
  2. Repeat:
     - Compute gradient $\nabla_\theta \mathcal{L}(\theta)$
     - Update: $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(\theta)$
  3. Until convergence

4.3 Gradient descent: the learning rate

- **Hyperparameter (η):** Step size in parameter space, affects convergence speed

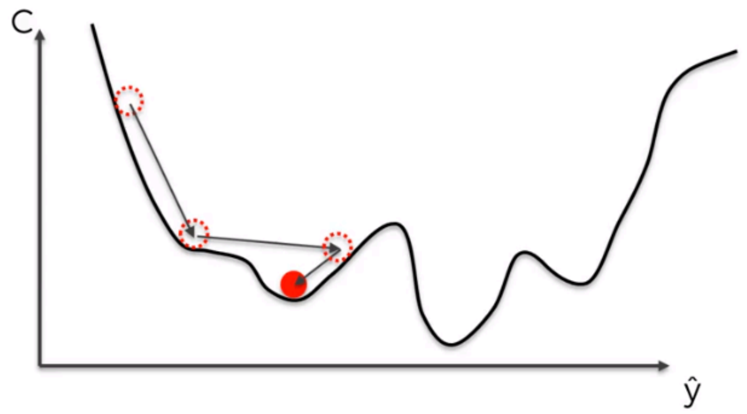
Gradient Descent Trajectories for Different Learning Rates



4.4 Stochastic/mini batch gradient descent

- **Mini batch gradient descent:** Update parameters using only a batch of training examples at a time.

- **Advantages:**
  - Faster convergence for large datasets.
  - Can escape local minima due to noisy updates.
- **Disadvantages:**
  - Noisy updates can lead to oscillations.
  - May require careful tuning of learning rate.



6 Training models

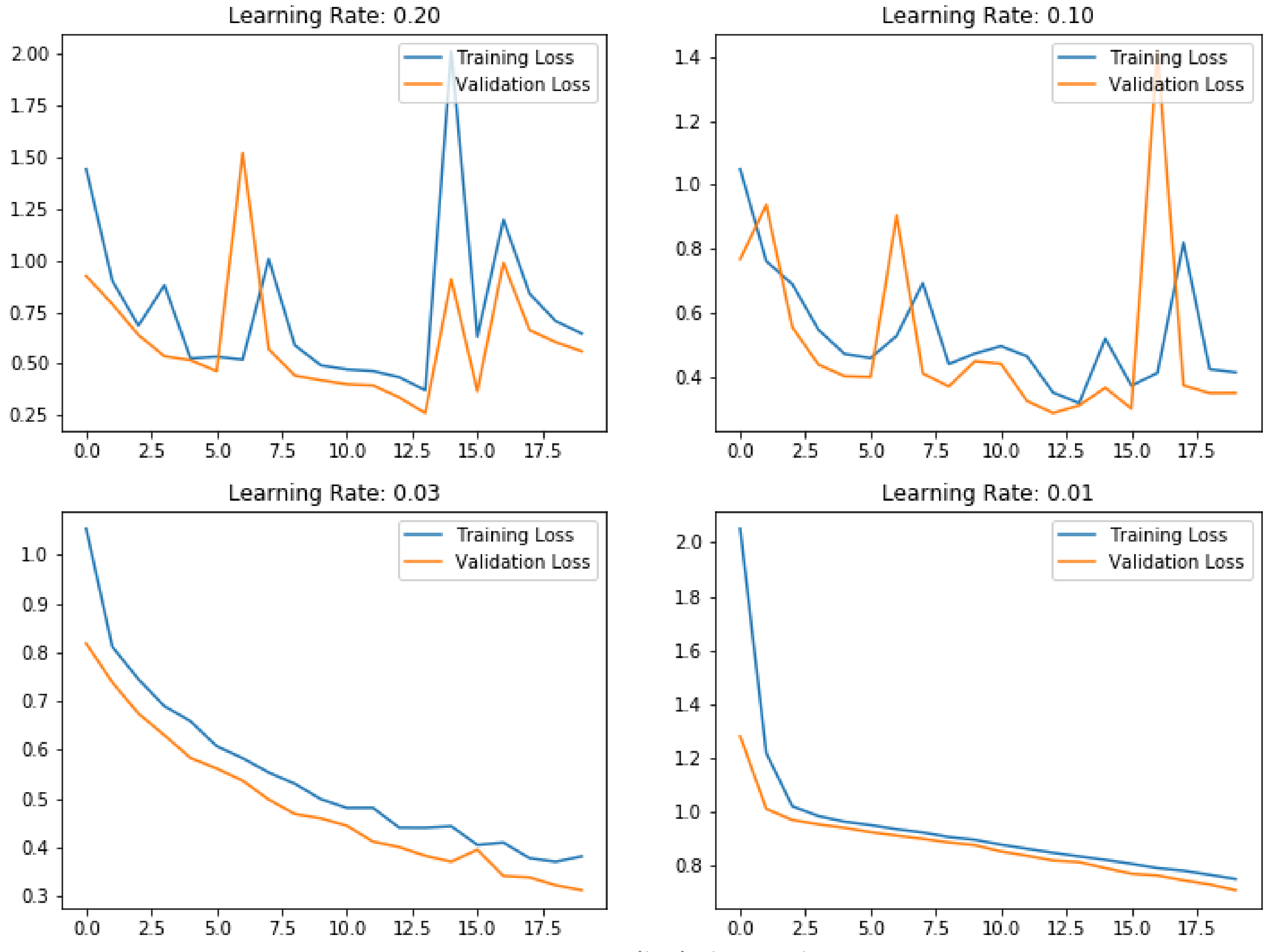
6.1 Train-test-validate

- **Training set:** Used to fit the model (e.g., find θ)
- **Validation set:** Used to tune hyperparameters
- **Test set:** Used to evaluate final model performance
- **Goal:** Ensure model generalizes well to unseen data
- **Partitioning:** Usually 60% train, 20% validation, 20% test
- **K-fold cross-validation:** Split into K subsets, train on K-1 and validate on the remaining one, repeat K times
- **Inference:** Using the trained model to make predictions on new data

6.2 The training process

```
initialize model parameters
for each epoch:
    for each batch in training_data:
        compute predictions using model
        compute loss using loss function
        compute gradients
        update model parameters using gradients and learning rate
    if validation set available:
        evaluate model on validation set
```

6.2 The learning curve



6.3 Model evaluation: in-sample vs. out-of-sample error

- **In-sample error E_{in}:**
  - Error measured on the training data
  - May underestimate true error if model overfits
- **Out-of-sample error E_{out}:**
  - Error measured on unseen (test/validation) data
  - Better estimate of model's real-world performance
- **Goal:** Achieve low E_{out} (good generalization)

6.3 Overfitting, underfitting, and generalization

- **Overfitting:**
  - Model fits training data too closely (captures noise)
  - Low in-sample error, high out-of-sample error
- **Underfitting:**
  - Model too simple to capture underlying pattern
  - High in-sample and out-of-sample error
- **Generalization:**
  - Model performs well on new, unseen data
  - Achieved by balancing model complexity and training
  - Goal: low out-of-sample error

8 Case study: linear regression for gene expression

8.1 Problem statement

- **Goal:** Predict the expression of thousands of target genes from a subset of landmark genes using linear regression.
- **Inputs:** $x \in \mathbb{R}^{943}$ (landmark gene expression levels)
- **Outputs:** $y \in \mathbb{R}^k$ (target gene expression levels, $k = 9,520$ to $21,290$)
- **Dataset:** GEO (microarray) and GTEx (RNA-seq)

8.2 Data preprocessing steps

- **Duplication removal:**
  - Removed duplicate samples to ensure each profile is unique.
- **Normalization:**
  - Each gene's expression was standardized (zero mean, unit variance) across samples.
- **Partitioning:**
  - Data split into training, validation, and test sets.

8.3 Linear regression model & notation

- **Model:** $h_\theta(x) = x^T w + b$
  - x: Input vector (landmark genes)
  - w: Weights (one per target gene)
  - b: Bias term
- **Training set:** $S = \{(x_i, y_i)\}_{i=1}^N$
- **Hypothesis:** h_θ parameterized by $\theta = (w, b)$

8.4 Loss function

- **Mean squared error (MSE):**

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N}\sum_{i=1}^{N} \|h_\theta(x_i) - y_i\|_2^2$$

  - Measures average squared difference between predicted and true target gene values.

- **Training objective:**

$$\theta^* = \arg\min_\theta \mathcal{L}_{\text{MSE}}(\theta|S)$$

8.5 Training: fitting the model

1. **Remove duplicates and normalize data**
2. **Split data** into train/validation/test
3. **Fit a separate linear regression** for each target gene:
    - For each gene j, solve:

$$w_j^*, b_j^* = \arg\min_{w_j, b_j} \frac{1}{N} \sum_{i=1}^{N} (x_i^T w_j + b_j - y_{ij})^2$$

8.6 Evaluation: measuring performance

- **Evaluate** on test set using MAE (mean absolute error):

$$\mathrm{MAE} = \frac{1}{N}\sum_{i=1}^{N} |h_\theta(x_i) - y_i|$$

- **Key findings:**
  - Linear regression provides a simple, interpretable baseline
  - Removing duplicates and normalization are critical
  - Performance is measured by MAE on held-out data
  - This baseline motivates the need for more complex models (not covered here)

9 Working with data: CSVs, tensors, and tensor operations

9.1 Loading data from CSV

- **Read CSV file:**
  - Open file, read each row as a list of values
  - Convert values to numbers (floats)
  - Store as a 2D array (matrix)

```
# Pseudocode for loading CSV data
open file.csv as f
for each line in f:
    split line by ',')
    convert each value to float
    append to data matrix
```

9.2 Tensors and tensor arithmetic

- **Tensor:** Generalization of vectors (1D) and matrices (2D) to higher dimensions
- **Create tensor:**
  - From list, array, or data matrix

```
# Pseudocode for tensor arithmetic
A = tensor([[1, 2], [3, 4]])
B = tensor([[5, 6], [7, 8]])
C = A + B        # elementwise addition
D = A * B        # elementwise multiplication
E = A @ B        # matrix multiplication
```

9.3 Broadcasting and advanced tensor operations

- **Broadcasting:**
  - Automatically expands shapes for elementwise operations
  - Example: Adding a vector to each row of a matrix

```
# Pseudocode for broadcasting
A = tensor([[1, 2], [3, 4]])   # shape (2, 2)
b = tensor([10, 20])           # shape (2,)
C = A + b                      # b is broadcast to (2, 2)
```

10 Putting it all together: a simple training loop

```python
# X, y are 1-D arrays of length N
theta0, theta1 = 0.0, 0.0
lr = 1e-2 # eta

for epoch in epochs:
    y_hat = theta1 * X + theta0
    residual = y_hat - y
    grad0 = 2 / len(X) * residual.sum()
    grad1 = 2 / len(X) * (X * residual).sum()
    theta0 -= alpha * grad0
    theta1 -= alpha * grad1
```

$$\mathcal{L}(\theta_0, \theta_1) = \frac{1}{N} \sum_{i=1}^{N} (\theta_1 x_i + \theta_0 - y_i)^2$$

$$\frac{\partial \mathcal{L}}{\partial \theta_0} = \frac{2}{N} \sum_{i=1}^{N} (\theta_1 x_i + \theta_0 - y_i)$$

$$\frac{\partial \mathcal{L}}{\partial \theta_1} = \frac{2}{N} \sum_{i=1}^{N} x_i (\theta_1 x_i + \theta_0 - y_i)$$

11 Model training workflow

1. **Prepare data:** Clean, preprocess, and split into training/validation/test sets
2. **Choose model:** Select hypothesis set (e.g., linear regression)
3. **Train model:** Fit parameters using training data
4. **Tune hyperparameters:** Use validation set to adjust settings (e.g., learning rate)
5. **Evaluate:** Assess performance on test set
6. **Deploy:** Use trained model for predictions