00660121 - Medical Diagnostics HW1: Multiple Linear Regression & Gradient Descent

TA in charge: Mattan Hoory*
Faculty of Biotechnology and Food Engineering
Technion - Israel Institute of Technology

Spring 2025

Submission Due: 24.6.25

Table of contents

1 Aim of the assignment **2**

2 Dataset & background **3**

2.1 Writing and running code 4

2.1.1 Anaconda (Instructions are also given in HW0, but here is a reminder) 4

2.1.2 Google Colab / Kaggle 4

3 Tasks **5**

3.1 Task 1 – Implement the model 5

3.2 Task 2 – Train the model 6

3.3 Task 3 – Evaluation & analysis 6

*Thanks to Prof. Roee Amit and the Technion Faculty of Biotechnology & Food Engineering for their support.

3.4 Task 4 – From Regression to a "Toy" Classifier 7
3.5 Task 5 – Theoretical questions 10

4 Submission Guidelines **10**

4.1 Written report guidelines 11

5 Grading rubric **13**

6 Academic integrity & help **13**

7 FAQ **13**

1 Aim of the assignment

In lecture 1 we defined supervised learning and saw the linear-regression model.

This homework lets you:

- Implement *multiple* (multi-input:one-output) linear regression **from scratch** with gradient descent
- Apply it to a **mini-GEO gene-expression subset** (landmark → a **single** target gene)
- Measure and reflect on the model's limits.

Some of the code you write here might be reused when we extend to neural networks in HW2.

Work in pairs. One submission per pair. **Due date:** 24.6.2025, 23:59. **Late submission:** 10% penalty per day. **Submission:** Through moodle, ZIP file with code and report, see details below.

- *Steps* you need to take are marked *A., B., C.* etc. in the sections below. This requires you to write code, and run it. You will not be evaluated on the code itself, but on the results you obtain and the report you write.
- *Questions/Plots* required for the report are marked numerically *3.1.1, 3.1.2,* etc. These require you to answer the questions or use code to create the plots, and then include them in your report. You can answer the questions as a paragraph, or as a list of answers, but make sure to include all the required information.
- Theoretical questions appear in section 5, the answers to these questions should be included in your report as well.

2 Dataset & background

File / folder	Purpose
gene_expression_regression.csv	mini-GEO gene-expression dataset containing the landmark genes and one target gene from the original D-GEX dataset. The target gene is in the last column.

Click the link above to download the dataset, you will need to unzip it to access the `gene_expression_regression.csv` file. Pay attention to the file format, it is a CSV file with a header row and column. The last column is the target gene expression value, and the other columns are the landmark genes' expression values. The data is already preprocessed and ready for use.

Reference:

Gene expression inference with deep learning

Yifei Chen, Yi Li, Rajiv Narayan, Aravind Subramanian, Xiaohui Xie

Bioinformatics, Volume 32, Issue 12, June 2016, Pages 1832–1839

https://doi.org/10.1093/bioinformatics/btw074

2.1 Writing and running code

To write and execute the code, you can use either **VScode** with an **Anaconda** environment, **Google Colab / Kaggle** , or any other environment you see fit. see HW0 for details.

2.1.1 Anaconda (Instructions are also given in HW0, but here is a reminder)

Environment is identicle to HW0, so you can use the same one.

```
git clone https://github.com/mathoory/00660121-medical-diagnostics.git

cd "00660121-medical-diagnostics\homeworks\02 HW1"

conda env create -f environment.yml

conda activate hw1-00660121
```

2.1.2 Google Colab / Kaggle

1. Create a new notebook.
2. Upload the `gene_expression_regression.csv` file to the notebook. For *Kaggle* you can use the datatset I've uploaded here

3 Tasks

A. Read the (1) Introduction, and (2) Dataset (up until and including 2.2) chapters from the publication "Gene expression inference with deep learning" by Chen et al. (2016) to understand the dataset and the task at hand.

B (**Sanity Check**). Using code, open the file `gene_expression_regression.csv`, print the first 5 rows, to check that the data is loaded correctly.

3.0.1. What do the columns represent? What do the rows represent?

3.0.2. How many samples and features are there?

3.1 Task 1 – Implement the model

A. Implement a function to fit a **linear regression model** using **mini-batch gradient descent** and **mean squared error (MSE)** loss. Use the following notation:

$$X \in \mathbb{R}^{n \times m}$$

$$Y \in \mathbb{R}^{n}$$

(Last column is the target gene)

Implement this part yourself (a full 'training-loop' like what we have seen in the tutorials), do **not** use imported optimisers. In the tutorials we have seen an example for a single-variable linear regression, here you will implement a **multiple linear regression** model, which is a generalization of the single-variable case.

3.1.1 What are the inputs and outputs of the model?

3.1.1.1 What parameters of the model are learned during training?

3.1.1.2 What parameters of the model are fixed during training?

3.1.1.3 What is the weight vector (denoted w), what is its shape/size?

3.1.2 What is the loss function used for training? Write down the equation for the loss function.

3.1.3 What is the gradient of the loss function with respect to the model parameters? Write down the equation for the gradient.

3.1.4 What is the update rule for the model parameters? Write down the equation for the update rule.

3.2 Task 2 – Train the model

A. Split data 80 % / 10 % / 10 % into **train / val / test**, for this part only you may use `sklearn.model_selection.train_test_split` (with `random_state=42`). You can read about using it here.

Note: The split is fixed for reproducibility

B. Decide on a learning rate and batch size.

C. Plot the losses vs epochs for both train and validation sets.

3.3 Task 3 – Evaluation & analysis

A. Experiment and compare different learning rates and batch sizes.

B. Evaluate the model on the test set using **mean squared error (MSE)** and **Pearson correlation** between your predictions ($h(X) = \hat{Y}$) and the true values (Y).

3.3.1 Report your results and findings. Which learning rate and batch size worked best?

How did you determine that? (Add plots if necessary.)

3.3.2 Report the best **final test MSE** and **Pearson correlation** (one number each). Mention any changes you made to the model or training process that improved the results.

3.3.3 Plot the **predicted vs true values** for the test set. Add a line $y = x$ to the plot, and explain what it means.

3.3.4 (Optional Bonus) What is the difference betweeen Pearson correlation and MSE? When would you use one over the other? How do they relate to each other?

3.4 Task 4 – From Regression to a "Toy" Classifier

One of the possible **downstream tasks** of gene-expression models is to predict disease risk based on gene expression. You have already built a **linear regression** model to predict a continuous target gene expression value (scalar), in real clinics you often predict a *binary* outcome ("disease / no disease" - 0/1) rather than a raw gene-expression value. This is called binary classification. This correlation between prediction and clinical state could be based on clinical guidelines or expert knowledge for the specific disease and it's biomarkers. In practice you would not use LR, and instead apply dedicated classification models (logistic regression, trees, NN + soft-max) and loss functions (cross-entropy) which are used to predict probabilities and **are not required** for this task.

Here we take a shortcut: we keep the linear-regression hypothesis class you just built and add on a simple *threshold* to obtain class labels. (Now, $Y \in 0, 1$ instead of $Y \in \mathbb{R}$.) The goal is to see how model choice + decision threshold jointly influence downstream accuracy.

The true target y_i you used so far is assumed to be a quantitative **biomarker** measuring

risk for a certain disease. We convert it to a binary label:

A. For all of the data, compute the **median** of the target gene (across all samples).

B. Define

$$\tilde{y}_{ij} = \begin{cases} 1 & \text{if } y_{\text{binary}} \geq \text{median} \\ \\ 0 & \text{otherwise.} \end{cases}$$

(In real datasets you would get 0/1 labels from clinicians.)

C. Write code to compute and save Y_{binary}, a vector of 0/1 labels for each sample, where $Y_{\text{binary}}[i] = \tilde{y}_i$ for the i-th sample.

D. Modify your model the predict the binary label you just created Y_{binary} which will be used as the ground-truth instead of the original continuous Y.

D.1. **Modify the model for binary labels**: Think what should be changed so that its predictions are 0 / 1 classes that match the labels you just defined.

D.2. **Predict on the existing test split**: for every sample output a prediction (0/1).

D.3. **Evaluate with three classification metrics** (See definitions below):

- **Accuracy** – fraction of correct predictions
- **Precision** – $\text{TP}/(\text{TP} + \text{FP})$
- **Recall** (sensitivity) – $\text{TP}/(\text{TP} + \text{FN})$

Definitions:

- **TP (True Positive):** Number of samples where the model predicted 1 (disease) and the true label is also 1.
- **FP (False Positive):** Number of samples where the model predicted 1 (disease) but the true label is 0 (no disease).

- **TN (True Negative):** Number of samples where the model predicted 0 (no disease) and the true label is also 0.
- **FN (False Negative):** Number of samples where the model predicted 0 (no disease) but the true label is 1 (disease).
- **Accuracy:** Fraction of correct predictions

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Precision:** Fraction of positive predictions that are correct

$$\text{Precision} = \frac{TP}{TP + FP}$$

- **Recall (Sensitivity):** Fraction of actual positives that are correctly identified

$$\text{Recall} = \frac{TP}{TP + FN}$$

3.4.1 How did you modify your model for binary prediction and why?

3.4.2 How do these metrics compare? Based on what our downstream task, which metric is the most important? why?

3.5 Task 5 – Theoretical questions

Answer concisely (1-2 lines) in your PDF report:

1. Did you encounter underfitting or overfitting? How do we detect overfitting and underfitting in machine learning?
2. What would happen if we included non-linear featuers (e.g. $x_1^2, log(x_2)$, etc.) in the model? Would it help or hurt the model performance? Why? (Theoretical question, no need to implement it.)
3. Suppose we double the number of input genes, how many parameters will the model have?
4. Suppose we double the number of output/target genes, how many parameters will the model have?
5. Suppose we double the number of input samples, how many parameters will the model have?
6. Suppose we have made all modifications in questions 3-5, write down the loss equation. In your answer state the dimensions of each term (scalar, vector (and what size) and matrix (and what size)).
7. Mention two limitations of linear regression models.
8. Does the test set allow us to estimate our in-sample error?
9. Would any split of the data into two disjoint subsets result in an equally useful train-test split?

4 Submission Guidelines

Submit a single ZIP file with the following structure:

```
HW1_ID1_ID2.zip
  src/                 # your python files
     main.py or main.ipynb
     any other code files for reproducibility...
  report_ID1_ID2.pdf   #  3 pages + charts
```

4.1 Written report guidelines

In essence, the goal of the report is to be a *concise* summary of your work, it should show your understanding of the model, results and reasoning behind your decisions. No need to include code, any equations or algorithms used should be mentioned if we saw them in class or explained otherwise.

- **Content**:
  - Include your names and IDs on the first page
  - Include answers and results of all parts (1 – 5) in the report.
  - Include all plots and figures that help explain your results.
  - Use clear markers for each task and question.
- **Length**: Maximum **3 pages** (excluding charts and figures, equations count towards length).
- **Font**: Readable (e.g., Arial) with a minimum size of 11pt and standard margins.
- **Explanations**: Provide clear, short explanations for your model choices, esp. if they diverge from what we've seen in class. Include any challenges you faced and how you solved them.
- **Charts**: Include relevant charts and figures to illustrate your results. Make sure they are clear and labeled, and provide a brief explanation of how it relates to your

results.

5 Grading rubric

Component	Pts
Written Report	30
Full implementation of the model (Sections 3.1-4)	25
Written answers (3.5)	25
Model performance (competitive factor)	15
Results reproducibility	5

6 Academic integrity & help

- Do not use any external libraries for the model implementation, except `numpy` and `pandas`.
- Do not use `sklearn` for the model implementation, only for data splitting.
- Use any online documentation or LLMs, but **do not share code or prompts** across pairs.
- Post questions in the **HW1 Moodle forum**. For Private issues email the TA.

Good luck!

7 FAQ

Q: Can I use Kaggle/Collab? A: You may use any platform for development, but your final submission should follow the required folder structure and include files as specified.

Q: What libraries can I use? A: You can use `numpy` and `pandas` for data manipulation, but do not use any libraries for the model implementation, except for data splitting with

`sklearn.model_selection.train_test_split`. For plotting, any library is fine (e.g., `matplotlib`, `seaborn`, etc.).

Q: Do I need a GPU to run this assignment? A: No, you can run this assignment on a CPU. The dataset is small enough to fit in memory and the model training will not take long.

Q: What language should I use for the report and code? A: Any language is fine! (English is preferred, but not required.)

Q: What should I do if my results are different each run? A: Make sure to set all random seeds (e.g., random_state=42 in train_test_split and for any random number generators you use) for reproducibility.

Q: What Loss funtcion should I use? A: You can you any loss function that is suitable, but the most common one for regression tasks is Mean Squared Error (MSE).