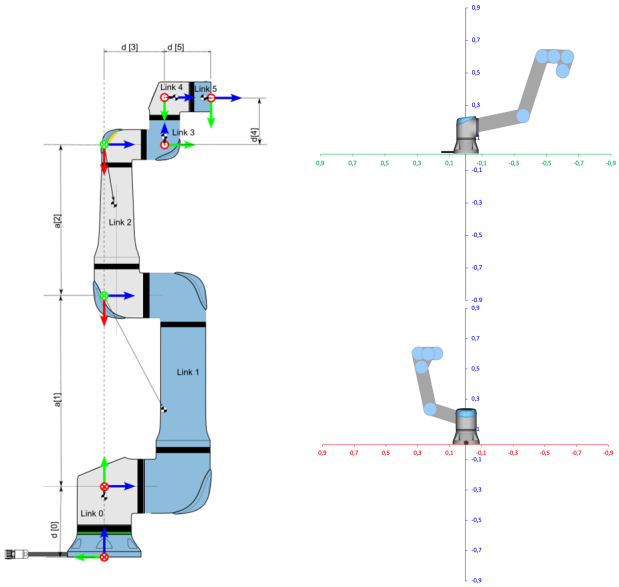
Machine Learning for Robotics RBE 577

Project 1: Joint Effort - Predicting the Pose

DUE: Tuesday February 4 at 5:00 pm.



All written components should be typeset using LaTeX. A template is provided on Overleaf. However, you are free to use your own format as long as it is in LaTeX.

Submit two files **a)** your code as a zip file **b)** the written report as a pdf file. If you work in pairs, only **one** of you should submit the assignment, and write as a comment the name of your partner in Canvas. Please follow this formatting structure and naming:

```
project_YOUR_NAME.pdf
project_YOUR_NAME.zip
    code
        train_linear_regression.py
        train_regression.py
        train_NN.py
        datasets.py
        linear_regression.pth
        NN.pth
        linear_regression_analytical.pth
        feature_engineering.py
        helpers
            data_transforms.py
            loss.py
```

```
└── metrics.py
```

Present your work and your work only. You must *explain* all of your answers. Answers without explanation will be given no credit.

Goal of This Project

This project is designed to assess your understanding of core concepts in regression, machine learning, and programming with PyTorch. By working on this assignment, you will:

Programming Component (70 points)

In this component, you will:

1. Explore regression techniques to model the forward kinematics of a UR10 robot arm.
2. Gain hands-on experience with PyTorch.
3. Learn how to preprocess data and train simple neural networks to solve real-world problems.

You will be provided with the following files:

1. `datasets.py` – Implement *load_dataset()* to read the dataset, extract features (X) and labels (y), and *prepare_dataset()* to split the data into training and testing sets.
2. `ur10dataset.csv` – This file contains 100,000 samples of joint angles $(\theta_1, \theta_2, \ldots, \theta_6)$ and the corresponding end-effector poses (x, y, z, r_x, r_y, r_z). The first 6 columns correspond to the joint angles (inputs), and the last 6 columns correspond to the end-effector poses (outputs).
3. `train_linear_regression.py` – Implement a Linear Regression class with methods for training, predicting, and evaluating the model on the provided dataset.
4. `train_NN.py` – You will need to implement a Multi-Layer Perceptron (MLP) to train and evaluate a neural network on the provided dataset.
5. `helpers` – You are required to implement the losses, additional metrics, and data transforms in this folder to support the training process.

We recommend using **PyTorch version 2.0 or higher**, specifically **PyTorch version 2.4.1**, for this assignment. To ensure a consistent and isolated environment, we suggest using **conda environments**.

You can find the official documentation for installing conda here. Once conda is installed, follow these steps to set up your environment:

First, let us create a conda environment, after which we will install the required libraries listed in the *requirements.txt* file.

```
conda create -n ml_env python=3.8.10 -y
conda activate ml_env
pip install -r requirements.txt #install the requirements
```

You are now ready to begin the coding part of the assignment

(30 points) Implement Linear Regression

For this part of the assignment, you are required to implement Linear Regression from scratch in two steps: first using the analytical solution, and then using an iterative optimization method like Stochastic Gradient Descent (SGD). You must write all necessary components from scratch and cannot rely on PyTorch functions.

Specifically, you need to:

1. **Analytical Solution:** Implement Linear Regression using the closed-form analytical solution. Your implementation should:
    - Compute the weights directly using the analytical method.
    - Train the model on the dataset provided and evaluate its performance on a test set.(*Note: You may not be able to train with all 100k samples using this approach. Instead, try using a subset of the dataset. Explain your reasoning for this.*)
    - Discuss the advantages and limitations of using this approach.
2. **Stochastic Gradient-Based Solution:** Implement Linear Regression using Stochastic Gradient Descent. Your implementation should:
    - Write a class-based implementation of Linear Regression.
    - Implement methods for predictions, loss computation, and weight updates. *(Please do not use PyTorch functions for the loss, weight updates and predictions)*
    - Train the model using the dataset provided and evaluate its performance on a test set.
    - Compute the different types of metrics in the `helpers/metrics.py` file, compare their performance during training, and submit graphs that showcase these comparisons.
3. **Comparison:** Compare the performance of the two approaches (analytical vs. iterative). In your discussion, consider the following:
    - Under what conditions might one approach be preferred over the other?
    - How do the computational requirements of the two methods compare?
    - What challenges might arise when using each approach on large datasets?

(15 points) Explore Feature Engineering

You are required to think critically about the type of features that might improve the training and performance of your Linear Regression model. Instead of directly using the raw joint angles as inputs, consider transforming or engineering features that better capture the relationships between the joint angles and the end-effector poses.

Specifically, you need to:

- Experiment with different feature representations of the joint angles that could improve model performance.
- Train your Linear Regression model using the engineered features and compare its performance to the model trained on raw angles.
- Reflect on the difference in performance between the two models. Discuss how and why the engineered features influenced the results.

The following equation represents the forward kinematics for a 2-DOF (Degrees of Freedom) RR robot. It computes the coordinates of the end-effector (x, y) relative to the robot base:

$$\mathbf{p}_{\text{end-effector}} = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} L_1 \cos(\theta_1) + L_2 \cos(\theta_1 + \theta_2) \\ L_1 \sin(\theta_1) + L_2 \sin(\theta_1 + \theta_2) \end{bmatrix}$$

In the dataset provided, the kinematics are for a 6-DOF RR robot. This equation serves as a hint for feature engineering—think about how to transform the raw joint angle values $(\theta_1, \theta_2, \ldots)$ into features that better capture the relationships between the input angles and the end-effector coordinates.

(25 points) Implement a Neural Network (MLP)

Using the features you engineered in the previous section, train a Multi-Layer Perceptron (MLP) to predict the end-effector poses given the joint angles. Unlike the Linear Regression task, you are allowed to use `PyTorch` functions to implement and train your Neural Network.

Specifically, you need to:

- Design a Multi-Layer Perceptron (MLP) architecture with at least one hidden layer. You can experiment with the number of layers, hidden units, and activation functions.
- Use `PyTorch` to implement your MLP, including the forward pass, loss computation, and backpropagation.
- Train your model using the dataset and evaluate its performance on the test set.
- Compute the different types of metrics in the `helpers/metrics.py` file, compare their performance during training, and submit graphs that showcase these comparisons.
- Compare the performance of the Neural Network to the Linear Regression model. Discuss whether the Neural Network outperformed the Linear Regression model and, if so, explain why.

Important: Ensure that your implementation includes all necessary steps, such as model definition, training loop, and evaluation. Additionally, include your reasoning in the performance comparison and provide any relevant visualizations or metrics to support your discussion.

Pro Tips

1. If you are new to PyTorch, consider starting with the official PyTorch Beginner Tutorials. They provide a step-by-step guide to building and training models.

2. When implementing the Multi-Layer Perceptron (MLP), the PyTorch documentation is an excellent starting point. It provides detailed explanations and examples of modules like `torch.nn.Linear`, `torch.nn.ReLU`, and `torch.optim`.

Theoretical Questions (30 points)

1. Importance of Loss Curves (10 points)

In the programming component, you might have plotted loss curves to observe how the loss decreases over time and to analyze the training and test components of the loss. Evaluating these curves can help you determine the quality of your model. You may have encountered terms such as overfitting, underfitting, bias, and variance. Refer to the graphs (Figures A and B in Figure 1) and comment on their behavior. Figure C represents the ideal fit we expect.

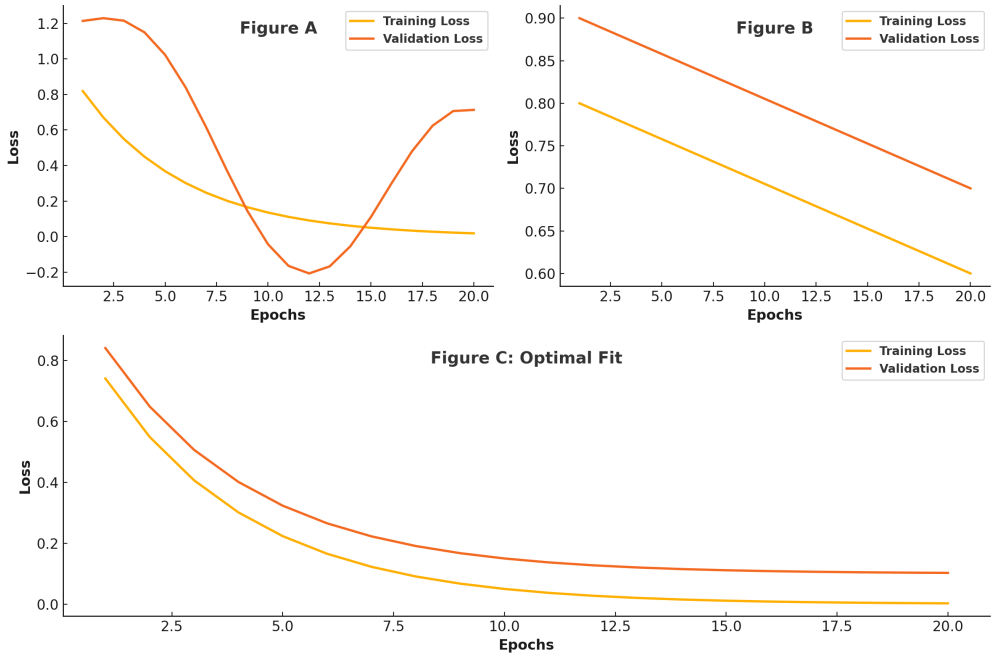


Figure 1: Loss Curves

1. Comment on the bias and variance of the training process for both graphs.

2. Identify which training iteration shows signs of overfitting or underfitting, and explain your reasoning in relation to the concepts of bias and variance.

3. Propose solutions to address the identified problems. List specific approaches to improve model performance.

2. Autoencoders and Variational Autoencoders (VAEs) (10 points)

Compare and contrast traditional autoencoders and variational autoencoders (VAEs) based on the following aspects:

1. Architecture
2. Loss functions
3. Latent space properties

The figure below (Figure 2) shows two latent spaces. Determine which one likely belongs to a VAE and which to a traditional autoencoder. Provide a detailed explanation of your reasoning.

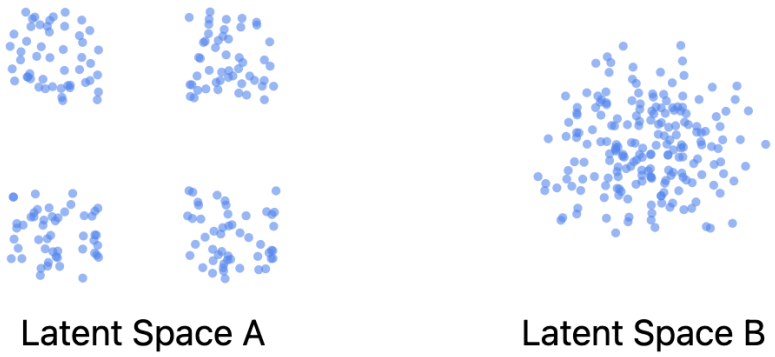


Figure 2: Latent Space of Autoencoders

3. Linear Algebra in Machine Learning (10 points)

Question: Let $\mathbf{A} \in \mathbb{R}^{d \times d}$ be a symmetric matrix representing a linear transformation. Consider the quadratic loss function:

$$L(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \mathbf{A}\mathbf{x}$$

Problem: Provide a detailed analysis of the following:

1. The relationship between the eigenvalues of $\mathbf{A}$ and the critical points of $L(\mathbf{x})$.
2. The conditions required for the existence of a minimum value of $L(\mathbf{x})$.
3. The relationship between the minimum value (if it exists) and the eigenvalues of $\mathbf{A}$.