ECE228 Machine Learning for Physical Applications (Spring 2025): Assignment 2

Instructor: Yuanyuan Shi, yyshi@ucsd.edu
Co-instructors and Teaching Assistants:
Yuexin Bian, yubian@ucsd.edu
Luke Bhan, lbhan@ucsd.edu
Rich Pai, cpai@ucsd.edu
Zekai Zhao, zez020@ucsd.edu

* Release Time: April 24, 10:00 AM
*** Deadline is May 7, 11:59 PM**

Submission format:

- For this assignment, you will be having 2 parts, the first part has theory questions and the second part is a coding one. All the questions should be done **Individually**.
- You need to submit two different files to Gradescope. One for the Written Questions(Part I) and one for Coding Questions(Part II).
- Please submit your HW as a typed PDF document (not handwritten). It is encouraged that you LaTeX all your work, though you may use another comparable typesetting method. The PDF document should include solutions to **both** the written questions and coding questions (results, discussions, plots, and so on).
- For Coding Questions, you should submit the ipynb (by running all cells in the notebook and including all the outputs) in Gradescope. Additional details have been given at the bottom.

1 Written Questions [60 Pts]

1.1 Neural Network Basics: Forward Propagation, Loss, and Backpropagation [18 pts]

Neural networks are the foundational building blocks of modern machine learning. They are composed of layers of simple functions, such as affine transformations and nonlinear activations like the ReLU function, which are stacked together to form highly expressive models. In this part of

the homework, we will work with a basic feedforward neural network consisting of one hidden layer with ReLU activation followed by a linear output layer. We will explicitly examine how the network maps an input vector to an output prediction, how the ReLU nonlinearity influences the structure of the loss function, and how to compute gradients with respect to the model parameters.

Let $x \in \mathbb{R}^k$ be an input vector and $y \in \mathbb{R}$ be a scalar output. The model is defined as follows:

$$\begin{aligned} z &= Wx + b \quad \text{(hidden layer pre-activation)} \\ h &= \text{ReLU}(z) \quad \text{(hidden layer activation)} \\ \hat{y} &= v^\top h + c \quad \text{(output)} \end{aligned}$$

where:

- $W \in \mathbb{R}^{m \times k}$, $b \in \mathbb{R}^m$: weights and biases of the hidden layer,
- $v \in \mathbb{R}^m$, $c \in \mathbb{R}$: weights and bias of the output layer,
- $\text{ReLU}(z) = \max(0, z)$: applied elementwise.

Assume that we are given a dataset $\{(x_i, y_i)\}_{i=1}^n$, and define the mean squared error loss as

$$L = \frac{1}{2n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2.$$

Q1.1 [2pts] Assume $m = 2$ and $k = 3$. Write out explicitly the dimensions of all matrices and vectors, and show how to compute $\hat{y}_i$ given an input $x_i \in \mathbb{R}^3$.

Q1.2 [2pts] Write the full expression for the loss L in terms of W, b, v, c, and the input data $\{x_i, y_i\}_{i=1}^n$. Show how the ReLU activation affects this expression.

Q1.3 [8pts] Derive the gradient of the loss L with respect to all parameters W, b, v, c, for a single data point (x, y). That is, compute:

$$\frac{\partial L}{\partial v}, \quad \frac{\partial L}{\partial c}, \quad \frac{\partial L}{\partial W}, \quad \frac{\partial L}{\partial b}.$$

Although the ReLU function is not differentiable at $z = 0$, it is differentiable almost everywhere, and in practice we can define its derivative as follows to enable gradient-based optimization methods:

$$\frac{d}{dz}\text{ReLU}(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$$

Q1.4 [6pts] Using the gradients derived in Q1.3, perform one step of gradient descent to update the parameters W, b, v, c for a single data point (x, y). Assume the data is

$$x = \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix}, y = 3$$

The parameters are

$$W = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 2 & 1 \end{bmatrix}, b = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, v = \begin{bmatrix} 2 \\ -1 \end{bmatrix}, c = 0,$$

and a learning rate $\eta = 0.02$. Write out the gradient values.

To help you check if your gradient derivation is correct and make the calculation easier, we've included the script in the provided code file.

After performing the updates, compute the new prediction $\hat{y}^{(1)}$ and the corresponding loss $L^{(1)}$ using the updated parameters. Compare the original loss $L^{(0)}$ and the new loss $L^{(1)}$ after one step of gradient descent. Discuss how the choice of learning rate η can influence the behavior of the optimization.

1.2 Universal Approximation Theorem for Neural Networks [8 pts]

The Universal Approximation Theorem states that a feedforward neural network with a single hidden layer and enough number of neurons can approximate any *continuous* function on a *compact* subset of $\mathbb{R}^n$, to arbitrary accuracy. In this question, we consider a simple case for 1D function approximation to build some intuitions in understanding neural network's universal approximation capability. Consider the task of approximating a continuous target function $f : [0, 1] \to \mathbb{R}$ using a one-hidden-layer feedforward neural network of the following form:

$$\hat{f}(x) = \sum_{i=1}^{m} v_i \, \sigma(w_i x + b_i) + c,$$

where $\sigma(\cdot)$ is a nonlinear activation function (e.g., Sigmoid, ReLU), and v_i, w_i, b_i, c are learnable parameters.

Q2.2 [2pts] Consider a simple piecewise constant function $f(x)$ defined by:

$$f(x) = \begin{cases} 0, & \text{if } 0 \le x < 0.5, \\ 1, & \text{if } 0.5 \le x \le 1. \end{cases}$$

Construct an explicit neural network $\hat{f}(x)$ with one neuron (i.e., $m = 1$) that approximates $f(x)$ as closely as possible using Sigmoid activation. Write down the expressions for w_i, b_i, v_i, and c.

Q2.3 [3pts] Consider another simple piecewise constant function $f(x)$ defined by:

$$f(x) = \begin{cases} 0, & \text{if } 0 \leq x < 1, \\ 1, & \text{if } 1 \leq x \leq 2, \\ 3, & \text{if } 2 \leq x \leq 3. \end{cases}$$

Construct an explicit neural network $\hat{f}(x)$ with two neurons (i.e., $m = 2$) that approximates $g(x)$ as closely as possible using sigmoidal activations. Write down the expressions for w_i, b_i, v_i, and c.

Q2.4 [3pts] Discuss why approximating step functions is important for approximating arbitrary continuous functions on $[0, 1]$.

1.3 Transformer basics: Single-head attention [15pts]

Assume we are working with a standard Transformer encoder block applied to a sequence of input vectors $x_1, x_2, \ldots, x_T \in \mathbb{R}^d$.

The self-attention operation produces output vectors $Y \in \mathbb{R}^d$ using the standard formula:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V,$$

where:

- $Q, K, V \in \mathbb{R}^{n \times d_k}$ are the query, key, and value matrices,
- d_k is the key/query dimension.

Assume we use single-head attention.

Q3.1 [3pts] Given input sequence $X = [x_1; x_2; x_3] \in \mathbb{R}^{3 \times d}$, and weight matrices $W_Q, W_K, W_V \in \mathbb{R}^{d \times d_k}$, write out explicitly (Show the matrix dimensions at each step.):

- The expressions for Q, K, V in terms of X and W_Q, W_K, W_V.
- The attention score matrix $S = QK^\top$.

Q3.2 [3pts] Assume $d = d_k = 2$, and given:

$$X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \quad W_Q = W_K = W_V = I_2$$

where I_2 is the 2×2 identity matrix.

Compute:

- The attention score matrix S.

- The softmax of each row of S.
- The output Y.

Q3.3 [3pts] In self-attention, why do we divide $QK^\top$ by $\sqrt{d_k}$? Assume that the elements of $q_i, k_j \in \mathbb{R}^{d_k}$ are independent random variables with mean 0 and variance 1. Can you derive an analytical expression for the variance of the dot product $q_i \cdot k_j$ in terms of d_k.

Q3.4 [2 pts] Now we compare the single-head attention with MLP. Assume the following two-layer MLP:

- First layer: $W_1 \in \mathbb{R}^{2\times 4}$ is:

$$W_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \quad b_1 = \mathbf{0}$$

- Activation: ReLU.
- Second layer: $W_2 \in \mathbb{R}^{4\times 2}$ is:

$$W_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad b_2 = \mathbf{0}$$

Can you compute the intermediate activation $H = \text{ReLU}(XW_1 + b_1)$ and compute the final output $Y_{\text{MLP}} = HW_2 + b_2$.

Q3.5 [4pts] **Comparison:**

(a) How does self-attention differ from the MLP in how it mixes input information?

(b) Why might self-attention better capture relationships between different tokens compared to an MLP?

1.4 Transformer basics: Multi-head attention [11pts]

To better understand how Transformers model complex relationships, we next explore how multi-head attention partitions the input space into separate subspaces for parallel processing. In a multi-head attention mechanism, the input vector $x \in \mathbb{R}^d$ is projected into multiple subspaces, with each head operating on a lower-dimensional space.

Suppose you are given:

- An input matrix $X = [x_1; x_2; x_3] \in \mathbb{R}^{3\times 4}$,
- Two heads (Head 1 and Head 2),

- Each head works on dimension $d_k = 2$.

That is, each head independently projects X to Q, K, V matrices of size 3×2.

Q4.1 [3pts] Suppose the input matrix is:

$$X = \begin{bmatrix} 1 & 2 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$

The projection matrices for Head 1 and Head 2 are:

$$W_Q^{(1)} = W_K^{(1)} = W_V^{(1)} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad W_Q^{(2)} = W_K^{(2)} = W_V^{(2)} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

For each head, compute the projected Q, K, and V matrices and write down the dimensions after projection.

Q4.2 [3pts] For each head separately, compute the attention score matrix $S = QK^\top$, and then apply scaling by $1/\sqrt{2}$, and apply softmax. Explain briefly: Which dimensions of the original input does each head focus on?

Q4.3 [3pts] Suppose now you concatenate the outputs of Head 1 and Head 2 (so you have dimension 4 per token again). Assume the output projection matrix $W_O = I_4$ (4×4 identity). Compute the final output matrix $Y \in \mathbb{R}^{3\times 4}$.

Q4.4 [2pts] Reflect:

(a) Why does splitting dimensions across heads (rather than sharing all dimensions) help the model?

(b) What is a potential downside if heads are not coordinated or trained properly?

1.5 Transformer basics: Positional Encoding [8pts]

Transformers lack recurrence and convolution, so they require a way to encode the order of tokens in a sequence. A standard method is to add a **positional encoding** vector to each token embedding.

The **sinusoidal positional encoding** is defined by:

$$\text{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right), \quad \text{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right),$$

where:

- pos is the token position (starting from 0),
- i is the dimension index,
- d is the total embedding dimension.

Q 5.1 [3pts] Suppose $d = 4$ (4-dimensional embeddings). Compute explicitly the positional encoding vectors for positions $pos = 0, 1, 2$. (You can use operation $\sin$ instead of compute the exact number).

Q5.2 [3 pts] (1) Find the period for dimension $i = 0$ and $i = 1$ when $d = 4$. (2) Explain how different dimensions vary at different frequencies. (3) Explain why, for a fixed dimension i, the function $pos \mapsto \text{PE}_{(pos,i)}$ varies smoothly and periodically.

Q5.3 [2 pts] Suppose you add positional encodings to input embeddings $X \in \mathbb{R}^{T \times d}$.

(a) Write the updated embeddings formula.

(b) Explain why positional encodings enable attention to capture relative position information even without explicit recurrence.

2 Coding Questions: MLP and Transformer Implementation [40 Points]

2.1 Dataset and Task

In this problem, we will work on a time-series dataset generated from a synthetic office room. The dataset includes measurements of room temperature, CO_2 concentration, number of occupants, ambient temperature, airflow rate and airflow temperature of the supply vent. Visualization of some sampled data from the dataset is shown in Fig 1.

The dataset provides historical data on these variables, which can be used to predict *future room temperature* and *CO_2 concentrations*.

For this task, we are going to

- Implement an MLP model for prediction.
- Implement a Transformer-based model for prediction.

Poor indoor air quality can lead to a range of health problems, including respiratory issues and allergies. By accurately predicting future room temperature and CO_2 concentration based on historical data, building managers can take proactive actions to improve thermal comfort and indoor air quality to ensure a comfortable and healthy environment for occupants.

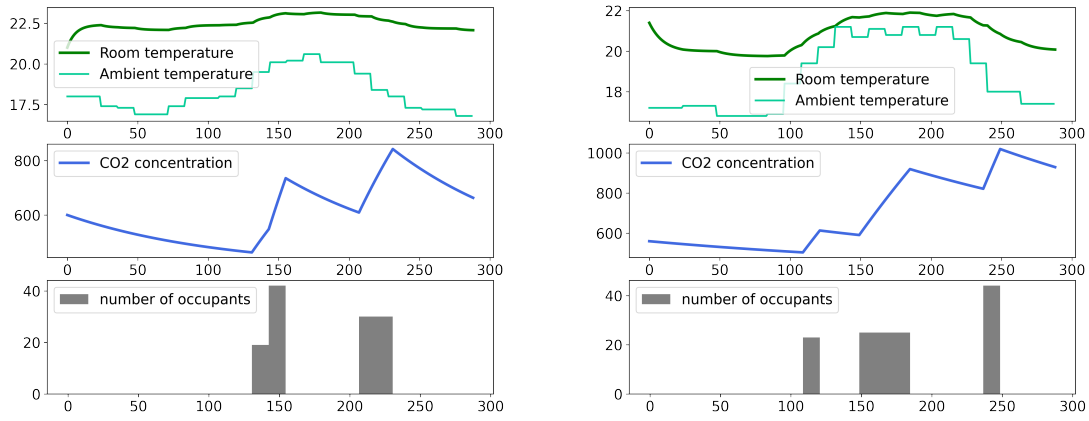

Figure 1: The temperature and CO2 concentration dynamics.

2.2 MLP Implementation

Multi-Layer Perceptron (MLP) is a type of feedforward neural network composed of one or more hidden layers of fully connected neurons. In regression tasks, the final layer typically outputs

a continuous value. Each hidden layer applies a linear transformation followed by a non-linear activation (such as ReLU). MLPs are powerful function approximators that learn hierarchical feature representations through backpropagation.

In this homework, you will complete the MLP Regressor class implementation including:

Complete the MLP Regressor class implementation including:

C 1.1 [2 pts] Implement an MLP.

C 1.2 [3 pts] Implement training function and evaluation function.

C 1.3 [5 pts] Conduct experiments using learning rates of 1×10^{-3} and 1×10^{-4} with hidden layer sizes of 32 and 128, and epochs to be 100. Visualize the validation loss for each parameter combination. What is your observation?

2.3 Transformer predictor

The transformer model is a type of deep learning architecture that has gained widespread popularity in natural language processing (NLP) tasks, and can also be used for time-series prediction. It is a neural network that is based on attention mechanisms, allowing it to focus on different parts of the input sequence at different time steps. This ability to attend to different parts of the sequence simultaneously makes it highly effective for processing long sequences and capturing complex temporal patterns in the data.

C 2.1 - C 2.5 [25 pts] Implementing a Transformer predictor following the instructions in the .ipynb file.

C 2.6 [5 pts] Conduct experiments using learning rates of 1×10^{-3} and 1×10^{-4} with hidden layer sizes of 32 and 128, and epochs to be 100. Visualize the validation loss for each parameter combination. What is your observation?

After implementing MLP and transformer, what do you think are some key differences between transformers and MLP in terms of architecture and performance, and in what situations might you prefer to use one over the other for time-series prediction tasks? If you have time, go wild tweaking hyperparameters and exploring the code!