BAX 452 HW 4

Course Instructor : Dr. Rahul Makhijani

Due Date : 18th Feb 2025

Please note: You are supposed to type up the solutions and submit it on canvas by **18th Feb 11:59 pm PST**. Refer to the course syllabus regarding late HW guidelines.

Question 1:

Linear Regression with PyTorch on the Auto MPG Dataset (30 points)

This task involves building, training, and evaluating a linear regression model using PyTorch on the Auto MPG dataset.

(a) Data Preprocessing (10 points)

1. **Dataset Loading**:
    - Load the Auto MPG dataset using Seaborn (sns.load_dataset('mpg')) or an equivalent library. Inspect the dataset to understand its structure and features.
2. **Handling Missing Values**:
    - Identify and handle any missing values in the dataset. For instance, you can:
        - Drop rows with missing values.
        - Use statistical imputation techniques (mean, median, or mode).
3. **Feature Selection**:
    - Select relevant features for predicting mpg. Include both numerical and categorical features (e.g., horsepower, weight, origin).
    - Justify your choice of features in a short explanation.
4. **Categorical Feature Encoding**:
    - Encode categorical features (e.g., origin) using techniques like one-hot encoding or label encoding to make them suitable for input into a PyTorch model.
5. **Normalization**:
    - Normalize numerical features to ensure they are on a similar scale. Use methods like min-max scaling or standardization (zero mean, unit variance).
6. **Dataset Splitting**:

- Split the dataset into training and testing sets (e.g., 80% training, 20% testing). Provide the code for shuffling and splitting the dataset.

(b) Model Implementation (10 points)

1. **Linear Regression Model**:
   - Design a simple linear regression model using PyTorch's torch.nn.Module. The model should consist of:
     - A single fully connected (nn.Linear) layer to predict mpg.
     - Weight initialization to set initial weights and biases using PyTorch's initialization methods (e.g., torch.nn.init).
2. **Loss Function**:
   - Select and use a loss function for this regression task.
     Explain your choice and why it is appropriate for this task.
3. **Optimizer**:
   - Implement the Stochastic Gradient Descent (SGD) optimizer from scratch without using PyTorch's built-in optimizers, such as torch.optim.SGD.
   - This involves directly updating the model parameters based on the gradients computed during backpropagation. Write the code for the update rule manually.
   - TIP: Ensure that gradients are reset (zeroed) after each update to avoid accumulation from previous iterations. param.grad.zero_()
4. **Model Summary**:
   - Print the model architecture and the number of trainable parameters.
5. **Utility Functions**:
   - Implement utility functions for:
     - Initializing model weights and biases.
     - Data batching (optional but recommended for large datasets).

(c) Training and Evaluation (5 points)

1. **Model Training**:
   - Train the model using the training set. Implement the following:
     - A training loop to update weights using the optimizer.
     - Track the training loss over epochs.
     - Use PyTorch's DataLoader for batching (optional).
2. **Loss Visualization**:

- Plot the training loss over epochs using Matplotlib or TensorBoard. Highlight trends such as convergence or overfitting.

3. **Model Evaluation**:
   - Evaluate the model on the test set and report the following metrics:
     - Mean Squared Error (MSE)
     - R^2 score (Coefficient of Determination)
   - Include a visualization comparing true vs. predicted mpg values (e.g., a scatter plot).

(d) Analysis (5 points)

1. **Performance Analysis**:
   - Discuss the performance of your model based on the reported metrics and visualization. Address questions such as:
     - How well did the model generalize to the test set?
     - Were there any underfitting or overfitting issues?
2. **Challenges**:
   - Describe any challenges faced during preprocessing, model training, or evaluation (e.g., feature scaling, convergence issues).
3. **Suggestions for Improvement**:
   - Suggest potential improvements for better performance, such as:
     - Feature engineering (e.g., polynomial features).
     - Hyperparameter tuning (learning rate, batch size).
     - Trying advanced models or adding regularization.

Question 2:

Image Compression using PCA (20 points)

In this question we will see the application of PCA for image compression.

The jpeg package is useful for reading and writing images.

- Using the above package, read in the image (https://drive.google.com/file/d/18-Nic-h4s6Cl35aZMtpz0Am9qp06K-Zu/view?usp=sharing) of the cat using the command readJPEG('tiger.jpeg')

  When you read in the image it would be a 3 Dimensional matrix of the form A(:,:,

1), A(:,:, 2), A(:,:, 3). The third dimension corresponds to each component of the RGB color scheme. Store all the three matrices into separate variables.

As this problem is focused on image compression and not description or interpretation of the variables, the data does not require centering (subtracting the variable means from the respective observation vectors), and the center argument is set to FALSE. If the argument is not set to FALSE, the returned image will not have the right RGB values due to having their respective means subtracted from each pixel color vector.

- Now use PCA on all three matrices to compress the data. The reduced matrix can be obtained by dropping the columns corresponding to the smaller eigen-values.

- Plot the fraction of variance as k increases

- Save the new image and note the size. In order to reconstruct the image from the principal components Plot the compression ratio (size of the new image/ size of original image) as a function of k where k is the number of principal components used to construct the compressed image. Choose k ∈ [3, 5, 10, 25, 50, 100, 150, 200, 250, 300, 350, p] where p is all the principal components.

Question 3:

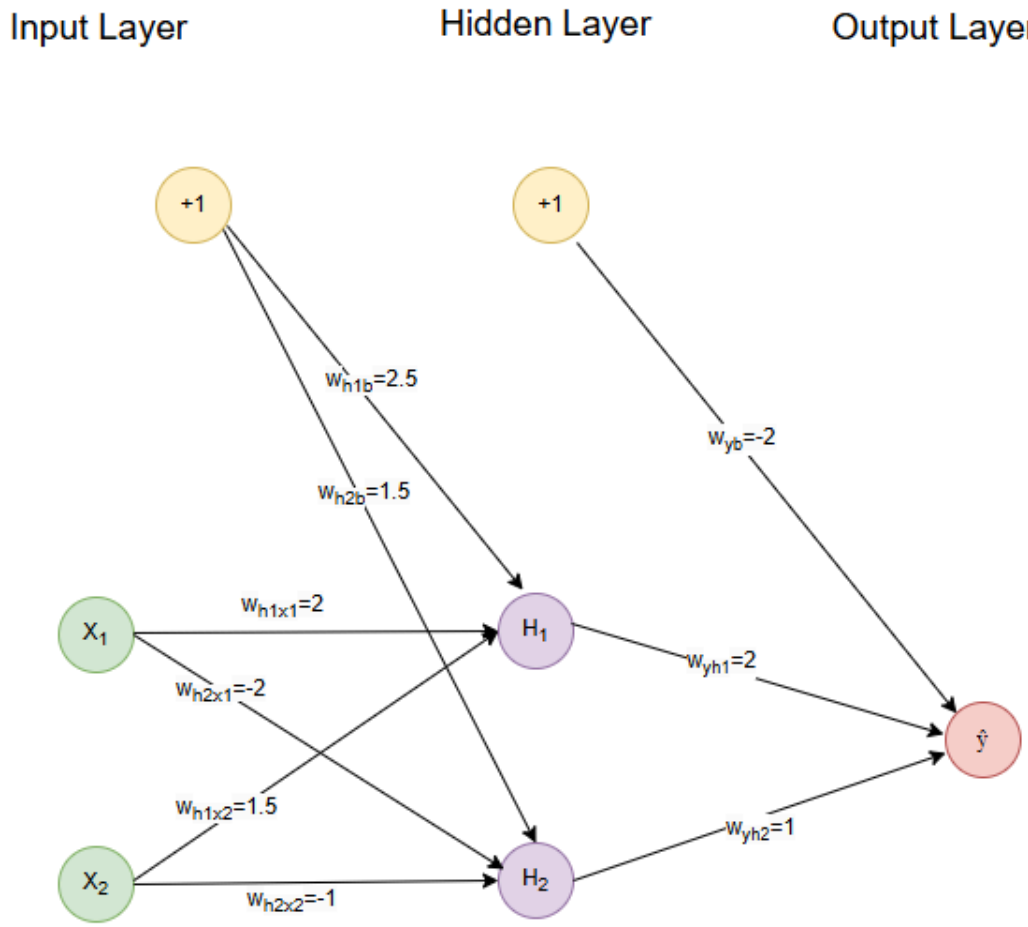
The figure below shows a 2-layer, feed-forward neural network with two hidden-layer nodes and one output node. x1 and x2 are the two inputs. For the following questions, assume the learning rate is α=0.1. Each node also has a bias input value of +1. Assume there is a sigmoid activation function at the hidden layer nodes and at the output layer node. A sigmoid activation function takes the form:

$$g(z) = \frac{1}{1 + e^{-z}}$$

where

$$z = \sum_{i=1}^{n} w_i x_i$$

and wi is the ith incoming weight to a node, xi is the ith incoming input value, and n is the number of incoming edges to the node.



(a) [10 points]

Calculate the output values at nodes h1, h2, and ŷ of this network for input {x1= 0, x2 = 1}.
Each unit produces as its output the real value computed by the unit's associated sigmoid function. Show all steps in your calculation.

(b) [20 points]

Compute one (1) step of the backpropagation algorithm for a given example with input {x1= 0, x2 = 1} and target output y=1.

The network output is the real-valued output of the sigmoid function, so the error on the given example is defined as:

$$E = \frac{1}{2}(y - O)^2$$

where O is the real-valued network output of that example at the output node, and y is the integer-valued target output for that example.

Compute the updated weights for both the hidden layer and the output layer (there are nine updated weights in total, i.e., the three incoming weights to node h1, the three incoming weights to node h2, and the three incoming weights to node $\hat{y}$) by performing **one** step of gradient descent. Show all steps in your calculation.

Question 4: Multi-Head Regression for Order Fulfillment Prediction (20 points)

This task involves building, training, and evaluating a **multi-head regression model using PyTorch** to predict multiple fulfillment-related metrics for a logistics company.

(a) Data Preprocessing (5 points)

Dataset Loading:

Use the **Supply Chain Order Fulfillment Dataset** from Kaggle: https://www.kaggle.com/datasets/shashwatwork/dataco-smart-supply-chain-for-big-data-analysis?resource=download

Load the dataset and inspect its structure.

- Identify and handle missing values:

Feature Selection:

Select relevant features for predicting the following targets:

1. **Estimated Fulfillment Time** (Days for shipping (real))
2. **Estimated Order Profit Per Order** (Order Profit Per Order)

3. **Likelihood of Delay** (Delivery Status)

Provide a short explanation of your feature selection choices.

Categorical Feature Encoding:

Encode categorical features using one-hot encoding or label encoding.

Normalization:

Normalize numerical features using **min-max scaling** or **standardization**.

Dataset Splitting:

Split the dataset into **training (80%)** and **testing (20%)** sets. Provide the code for dataset shuffling and splitting.

(b) Model Implementation (10 points)

Multi-Head Regression Model:

Design a **multi-head regression model in PyTorch** with:

- A **shared input layer** for feature extraction
- Three separate output heads for:
    1. **Fulfillment Time Prediction** (Linear output)
    2. **Estimated Order Profit Per Order** (Linear output)
    3. **Likelihood of Delay** (Sigmoid activation)

Loss Functions:

Choose and justify appropriate loss functions for each output:

- **Mean Squared Error (MSE)** for Fulfillment Time and Estimated Order Profit Per Order
- **Binary Cross-Entropy (BCE)** for Likelihood of Delay

Optimizer:

Use an optimizer such as **Adam** or **SGD** to train the model.

Model Summary:

Print the model architecture and number of trainable parameters.

(c) Training and Evaluation (5 points)

Model Training:

Train the model using the training dataset:

- Implement a training loop that updates weights
- Track the **loss separately** for each output head

Loss Visualization:

Plot **training loss curves** over epochs for all three targets using Matplotlib or TensorBoard.

Model Evaluation:

Evaluate the model on the test dataset and report:

- **Mean Absolute Error (MAE)** and **R^2 score** for Fulfillment Time and Estimated Order Profit Per Order
- **Binary Cross-Entropy Loss (BCE Loss)** for Likelihood of Delay

Include a **scatter plot** comparing true vs. predicted values for Fulfillment Time and Estimated Order Profit Per Order.