Deep Learning Architecture and Hardware Acceleration

Lab 1 Report

Jinay Panchal (jdpancha@iu.edu)

Objective: We are given a dataset with the meal inputs features as fish, chips, ketchup, and the goal is to predict the prize cost/money for the meal.

- The forward pass of the linear model, in getting the value is:
- Price = $X_{fish} * W_{fish} + X_{chips} * W_{chips} + X_{ketchup} * W_{ketchup}$
- Learning goal is $W = (W_{fish}, W_{chips}, W_{ketchup})$, with initial $W_0 = (50, 50, 50)$

This report details the implementation and evaluation of the **Delta Rule** and **Batch Delta Rule** for training a linear model. The primary objectives include adjusting model weights to minimize Mean Squared Error (MSE) and analysing the impact of different learning rates and training methods on model convergence.

Task 1:

➢ Delta Rule:

- **Data Definition:**
  The code begins by defining two classes: train_data and test_data, both subclasses of the Dataset class from PyTorch's torch.utils.data. These classes are designed to create custom datasets for training and testing.

- **train_data class:**
  - **__init__**(self, inputs, targets, learning_rates): Initializes the training dataset with input features, target values, and learning rates. These values are passed as arguments during dataset creation.
  - **__len__**(self): Returns the number of training samples.
  - **__getitem__**(self, idx): Returns a training sample's input features, target value, and learning rate based on the given index.

- **test_data class:**
  - **__init__**(self, inputs, targets): Initializes the testing dataset with input features and target values.
  - **__len__(**self): Returns the number of testing samples.
  - **__getitem__**(self, idx): Returns a testing sample's input features and target value based on the given index.

- **Implementation of Delta Rule:**

    - delta_rule is a function that performs the training using the Delta Rule. It takes two arguments: **train_data** and **test_data**.

    - It initializes the weights of the linear model to [50.0, 50.0, 50.0]. These weights represent the initial values for the model parameters.

    - It sets up lists **training_losses** and **testing_losses** to store training and testing losses over the course of training.

    - The function then iterates through each training sample:

        - It retrieves the input features, target value, and learning rate for the current sample from train_data.

        - It converts the input features to PyTorch tensors with the data type as torch.float32.

        - It computes the **prediction** using the current model weights and the input features.

        - It calculates the loss, which is the difference between the target value and the prediction.

        - It appends the loss to the training_losses list.

        - It updates the model weights using the Delta Rule formula: **weights = weights + learning_rate * inputs * loss**.

        - It calculates the testing loss using the **calculate_mse** function and appends it to the **testing_losses** list.

    - Finally, the function returns three values: training_losses (training losses), weights (final model weights), and testing_losses (testing losses during training).

Below is the output screenshot after running delta rule, the output consists of the weight changes happening after each of the iteration. Apart from that, it also prints the delta training loss and delta testing loss, and the final weights as well. The final weights are [134.0000, 52.0000, 114.0000]

```
Iteration:  0
tensor([100.,  70.,  90.])
Iteration:  1
tensor([130., 100., 120.])
Iteration:  2
tensor([130.0000,  50.0000, 110.0000])
Iteration:  3
tensor([134.0000,  52.0000, 114.0000])
Delta Rule Losses: [tensor(700.), tensor(120.), tensor(-270.), tensor(20.)]
Delta Weights:  tensor([134.0000,  52.0000, 114.0000])
Delta Test Losses:  [37000.0, 11800.0, 5133.33349609375, 1938.6666259765625]
Delta Test MSE:  1938.6666259765625
Final Testing MSE for Delta Rule: 1938.6666259765625
```

Fig: Weight changes after each iteration

- **Observations:**
  - **Weight Updates:** The Delta Rule updates the weights iteratively based on individual training samples. The learning rate controls the magnitude of weight adjustments.
  - **Convergence Behaviour:** The testing loss changes over iterations are visualized. Analysing the loss curve can provide insights into the model's convergence behaviour.
  - **Learning Rates:** Different learning rates (ε) lead to varying convergence speeds. Smaller learning rates result in slower but more stable convergence, while larger learning rates may cause oscillations or divergence.
  - **Final Weights:** The final weights obtained after training are dependent on the learning rate and data samples. These weights represent the optimized model for the given training samples.

➢ Batch Delta Rule:

- **Data Definition:**
  The task involves implementing the "batch delta rule" for training a linear model over **10 epochs** with a **batch size of 3**. The **learning rate** is uniformly set to **1/100**.

  The training data consists of input samples (batch_delta_train_inputs) and their corresponding target values (batch_delta_train_targets). Similarly, there is testing data (batch_delta_test_inputs and batch_delta_test_targets) for evaluating the model.

  Dataset classes (train_data and test_data): These classes are similar to the ones used in the previous "delta rule" implementation. They are custom dataset classes implemented

using PyTorch's Dataset class. They are used to structure and manage training and testing data effectively.

Note: Instead of using PyTorch functionality for Dataset or Dataloader class for Batch Delta Rule, I have directly taken the inputs for the delta function by straight away calling it.

- **Implementation of Batch Delta Rule:**
  1) Initialization:
     - Initialize model weights (weights) to [50.0, 50.0, 50.0].
     - Create empty lists to store training losses (losses) and testing losses (testing_losses).
  2) Epoch Loop: Iterate through 10 epochs.
  3) Batch Iteration:
     - Split the training data into each batch of size 3.
     - For each batch:
       - Compute the batch loss accumulating the weights changes.
       - Update the model weights using the accumulated changes.
  4) Epoch Loss Calculation:
     - Calculate and store the average training loss for the epoch.
     - Calculate and store the testing loss using the **calculate_mse** function.
5) Returning the training loss (losses), testing loss (testing_losses) and the updated weights (weights) to the function batch_delta_rule function assigning the respective values to batch_delta_losses, batch_delta_testing_losses and batch_delta_weights.

Below is the output screenshot after running batch delta rule, the output consists of the final updated weight changes after the last epoch which are [149.6477, 51.2920, 99.1688].

```
Current Epoch:  0
Current Epoch:  1
Current Epoch:  2
Current Epoch:  3
Current Epoch:  4
Current Epoch:  5
Current Epoch:  6
Current Epoch:  7
Current Epoch:  8
Current Epoch:  9

Batch Delta Final Updated weights:  tensor([149.6477,  51.2920,  99.1688])
```

Fig: Final Updated Weight changes after last epoch

- **Observations:**
    - The "batch delta rule" method allows the model to train more efficiently by processing data in batches, which can lead to faster convergence compared to the "delta rule."
    - The learning rate of 1/100 is consistent across all batches and epochs, ensuring stable training.
    - The training and testing losses are used to assess the model's performance, with the MSE serving as a quantitative measure.
    - The code provides a clear structure for implementing and evaluating the "batch delta rule" and allows for easy experimentation with different hyperparameters.

Task 2:

Validating the linear model collecting testing losses for batch delta and delta and plotting it versus epochs and iterations respectively.

1) **Iteration vs Testing Loss for Delta Rule:**
    - Testing loss decreases over iterations but with noticeable fluctuations as the model updates its weights.
    - The plot shows that the testing loss decreases over the time as the model learns from the training data, but it does not follow smooth convergence.
    - Reflects how well the model generalizes to testing data.

2) **Iteration vs Testing Loss for Batch Delta Rule:**
    - Testing loss significantly decreases more smoothly for batch delta rule with each epoch in contrast to the delta rule.
    - Each epoch includes multiple mini-batch iterations, but the testing loss is computed and plotted at the end of each epoch.
    - It also shows that batch delta can give more stable and consistent learning compared to delta rule.

- Batch Delta Rule Training Losses: [tensor(554.0490), tensor(25.9034), tensor(7.1928), tensor(1.7505), tensor(0.1590), tensor(-0.2376), tensor(-0.2819), tensor(-0.2367), tensor(-0.1799), tensor(-0.1316)]

- Batch Delta Testing Losses (from MSE): [7274.12646484375, 2984.542236328125, 1477.42138671875, 751.3836059570312, 381.6708984375, 193.2288055419922, 97.62451171875, 49.27012252807617, 24.853933334350586, 12.53415298461914]

- Delta Rule Training Losses: [tensor(700.), tensor(120.), tensor(-270.), tensor(20.)]
- Delta Rule Testing Losses: [37000.0, 11800.0, 5133.33349609375, 1938.6666259765625]

- Below are the plots for batch delta testing loss vs epochs and delta rule testing loss vs iterations.

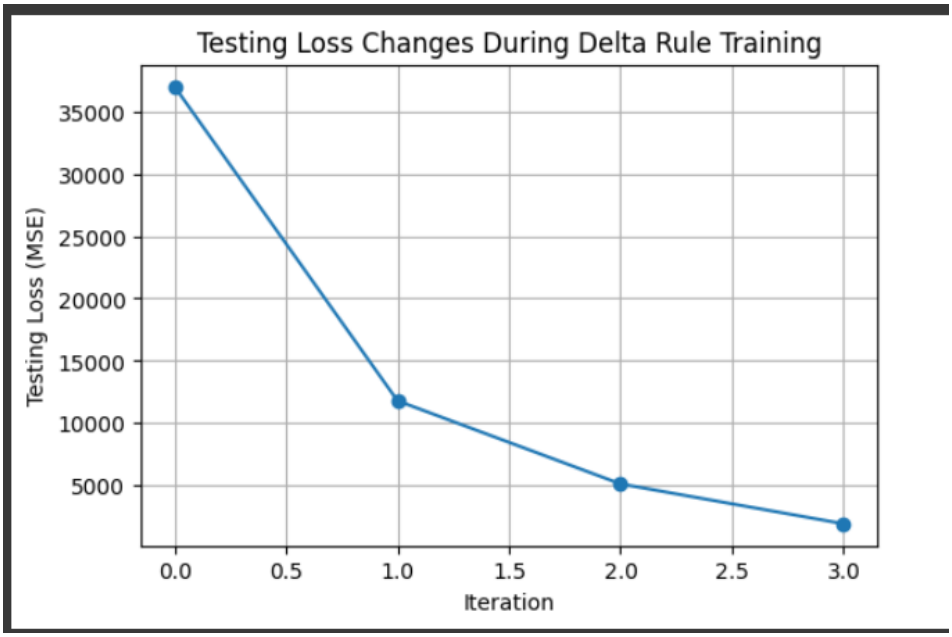

Fig: Testing Loss Changes Delta Rule vs Iterations

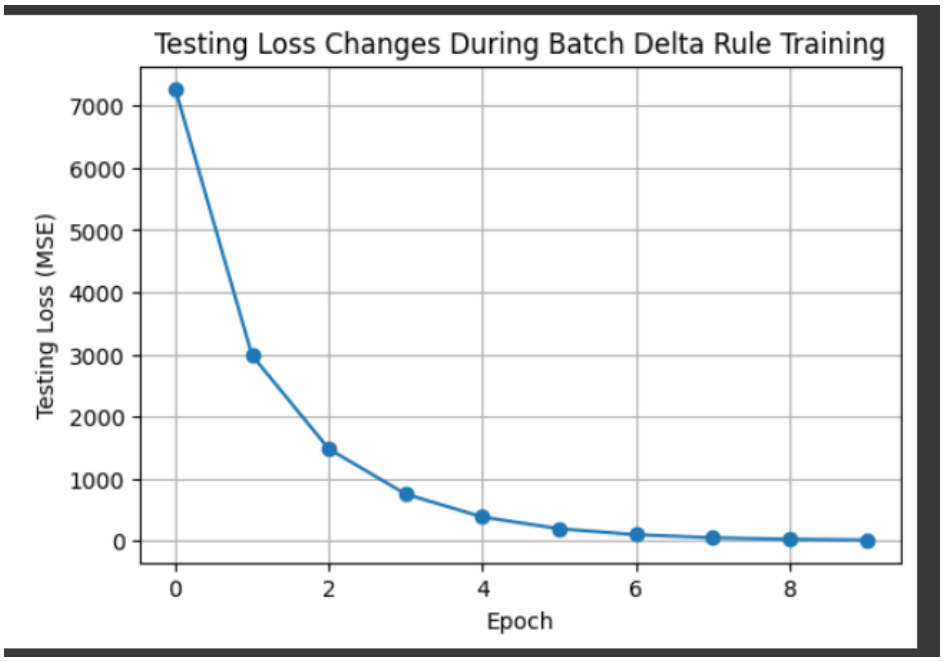

Fig: Testing Loss Changes Batch Delta Rule vs Epochs

Conclusions: Overall, the batch delta rule seems to offer more stable convergence in terms of testing loss compared to the delta rule. However, the choice between these two methods may depend on factors such as the dataset size, computational resources, and desired training characteristics.