Assignment #4 – GNN

https://github.com/Lemorita95/1FA006/tree/main/4_gnn

Written summary

Also available in https://github.com/Lemorita95/1FA006/blob/main/4_gnn/README.md

approach

1. hyperparameters:

   batch_size = 64

   learning_rate = 0.8e-4

   num_epochs = 50

   k_neighbors = 10

   n_layers = 2
2. Load data with awkward and normalize features and labels with get_normaized_data();
3. Data is normalized through z-score normalization;
4. Train, validation and test dataset is created with PyTorch Dataloader and a custom collate function collate_fn_gnn();
5. Training loss function is defined as MSE;
6. Gives the user the possibility to load a model (if found at models/model.pth);
7. Default model is defined with 2 dynamic edge convolution layers with a MLP kernel and a MLP output layer;
8. Edge convolution MLP kernel have a linear input layer, a fully connected hidden layer with 128 neurons and a linear output layer. Between each edgeconv layer the number of channels are kept constant at 128 neurons;
9. Train and validate model train_validade_model();
10. Test model test_model();
11. Model outputs the predicted labels GNNEncoder.forward();
12. Denormalize predictions and true values denormalize();
13. Plot scatter of averages and residuals;

results

1. the training-validation loop for the hyperparameters took around 14 seconds per epoch;
2. the best model was from the current hyperparameters and MLP architecture, both in terms of validation loss and computation time;
3. relatively simple model with MLP and EdgeConv layers flexibility for further accessments;
4. no signs of overfitting at train/validation losses for 50 epochs;
5. optimal number of neighbors (k_neighbors) and DynamicEdgeConv layers (n_layers) found was 10 and 2 respectively.

challenges

1. Awkard arrays are more complex then numpy arrays for debugging but the operations (normalization) ware quite straightforward;
2. Some hard-coded keys from data normalization -> not optimal;
3. changing batch_size (32->64) did not impact much on performance (losses and computation time) - maintaning other parameters constant;
4. 10 neighbors had lower validation loss and was more computation efficient, comparing to 5 and 10 neighbors - maintaning other parameters constant;
5. changing the number of DynamicEdgeConv layers did impact mostly on computation time, not at losses - maintaning other parameters constant;
6. DynamicEdgeConv fully connected layer number of channels had significant impact on both validation loss and computation time, more neurons yield less losses but more computation expensive (non-linear increse in time with linear-ish decrease in loss);

Result plots

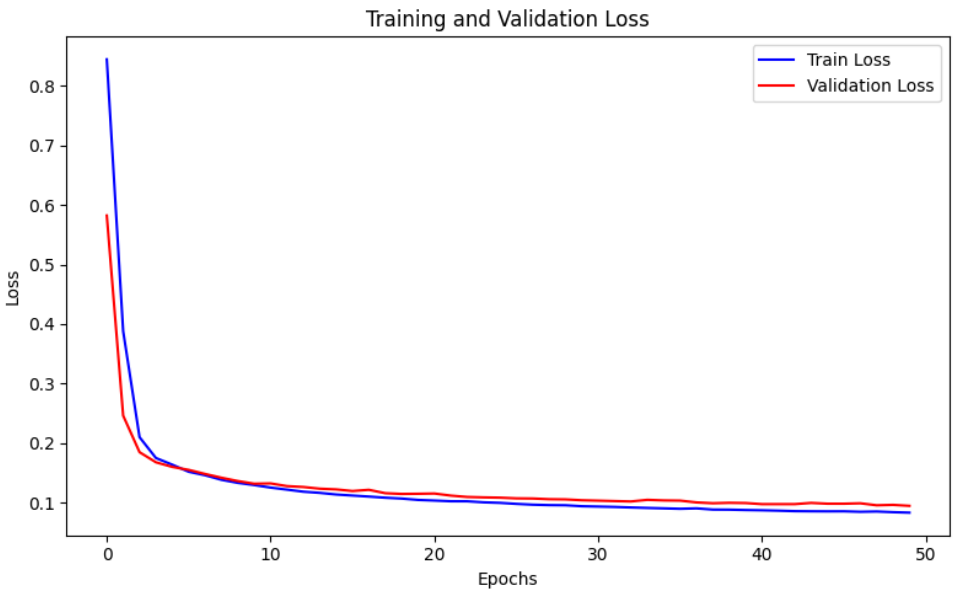

Figure 1: Train and validation loss per epoch.

From Figure 1 we see a decrease as the model goes through the epochs. This experiment used $Batch\ size = 64, Learning\ Rate = 8 \cdot 10^{-5}, Epochs = 50$. As seen in the figure and described in the previous section, the model did not presented signs of overfitting.

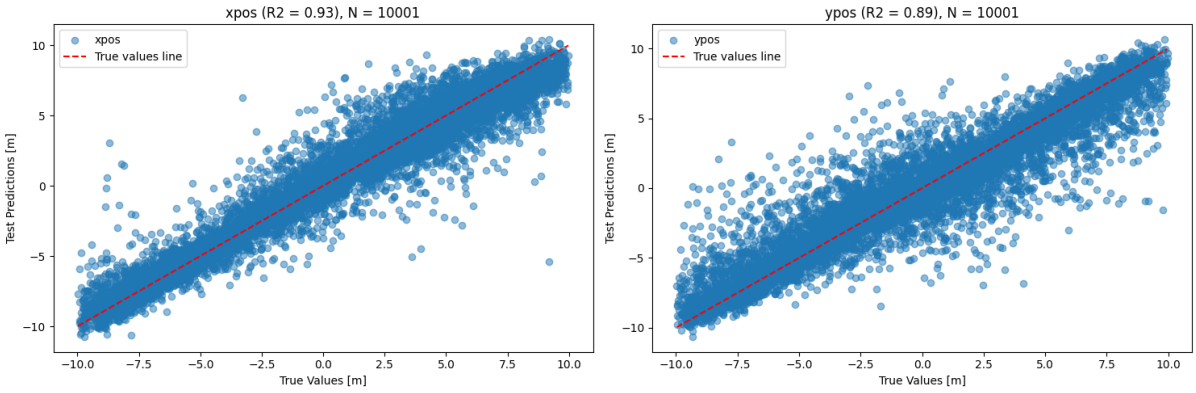

Figure 2: Scatter plot of predicted means.

From Figure 2, we can observe an elevated correlation from the model prediction to the true values. Outliers on `xpos` predictions seems to be extreme values (true values = +- 10) and a mirrored comportment is observed for both `xpos` and `ypos`, where a noticeable outliers are seen for negative values of true values, often overestimated (prediction > true), and positive values for true values, often underestimated (prediction < true).

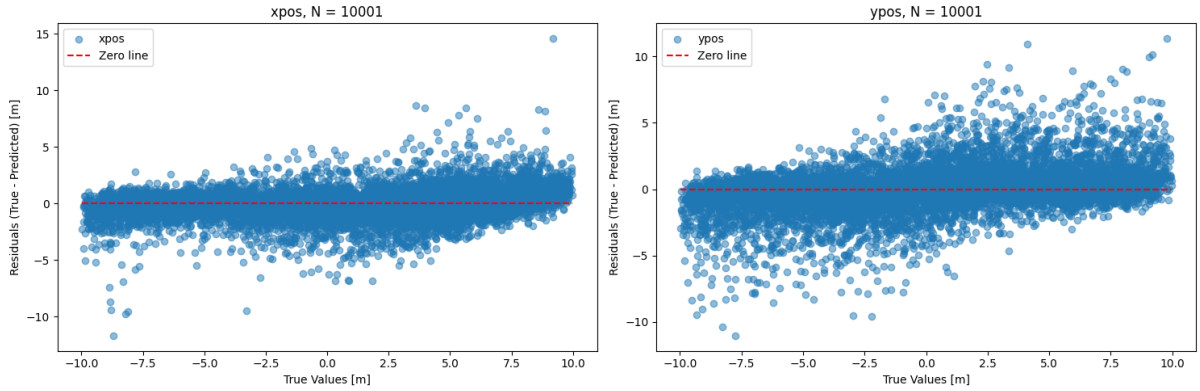

Figure 3: Scatter plot of predicted means residuals.

Figure 3 show a slight correlation of residuals and true values for `ypos`.

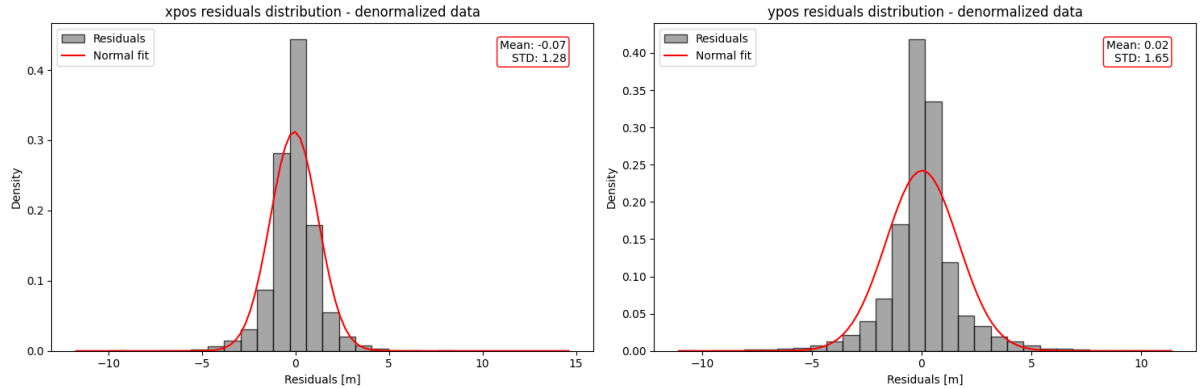

Figure 4: Distribution of predicted means residuals.

Figure 4 shows a good performance on the model's predictions regarding bias, when comparing the residual distribution to a normal distribution.