Classifying Speedup of Loop Optimizations

github.com/ryanbrooks816/CSC461_FinalProject

wandb.ai/ryanbrooks-uni/MLFinal/overview

Ryan Brooks

Introduction

The goal of the project was to classify the speedup/slowdown of compiler optimizations related to loops in source code.

My job was to improve the accuracy of the classification given the limitations of the dataset. It first required adopting to an existing codebase as part of a research project.

There are different benchmarks that make up the dataset and each benchmark has different programs. I worked on the NPB benchmark, which has 10 applications.

Each application has a series of 5 possible transformations applied to it, each with varying parameters.

An LLM is sent the reference and transformed source code to get the embeddings for the model to work with.

Dataset

Each applications (belonging to a benchmark) are reused throughout the dataset. Each loop targeted in these applications form a loop group.

Each data point has a different combination of transformations applied on the loop for that group. The target is the associated speedup value.

Problem 1

The dataset is highly skewed toward slow downs:
HSL: 61.1%, SP 14.6%,
SL: 21.6%, HSP: 2.6% (>1.5)

Easily possible to overfit the SP and HSP data points.

Problem 2

The sizes of the benchmarks, applications, and loop groups have high variance. Some loop groups and applications dominate the dataset.

For example, the NPB benchmark I was working with had applications with over 10,000 samples but others only 400 or 50.

Model Choices

- Using an MLP containing a series of linear layer, normalization layer (either batch or layer), a ReLU activation function and a dropout of variable percentage.
- The data loader, responsible for loading the embeddings, filtering out data based on config options, and performing train-test-split. Employs a stratified split to help combat the issues of the high-variance dataset.
- The goal is to have an even representation of all classes in the training, validation (and test) sets, and to avoid overfitting on speed up/high speed up data points, since they are so few.
- I would perform a series of tasks to study model performance of the validation set under different conditions and database splits.

Initial Testing

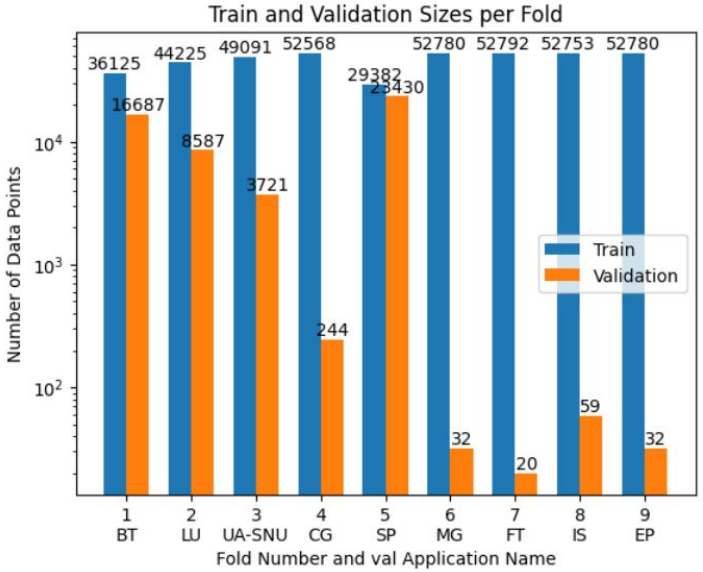
- To start, I ran some tests with basic model options. Batch size, learning rate, number of hidden layers, layer size, dropout, and weight decay.
- On average, the training accuracy was 91% and the validation accuracy was 85%

Next Steps

- From here, one of my first goals would be to experiment with narrowing the amount of data we're working with–both in benchmark and application levels.
- Through each, I'd compare the model's ability to fit training data, generalize the validation, and the in result variance between runs.

LOOCV

Given these next steps, the goal was to narrow down the dataset to just one benchmark (the largest one). I achieved this with leave-one-out cross-validation on the applications in the NPB benchmark.



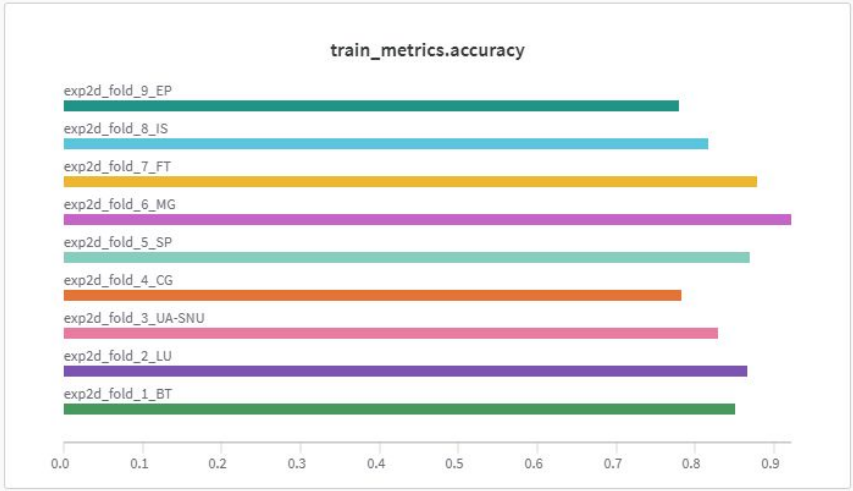
We can see that still here the the applications vary greatly in the size. (80-20 split).

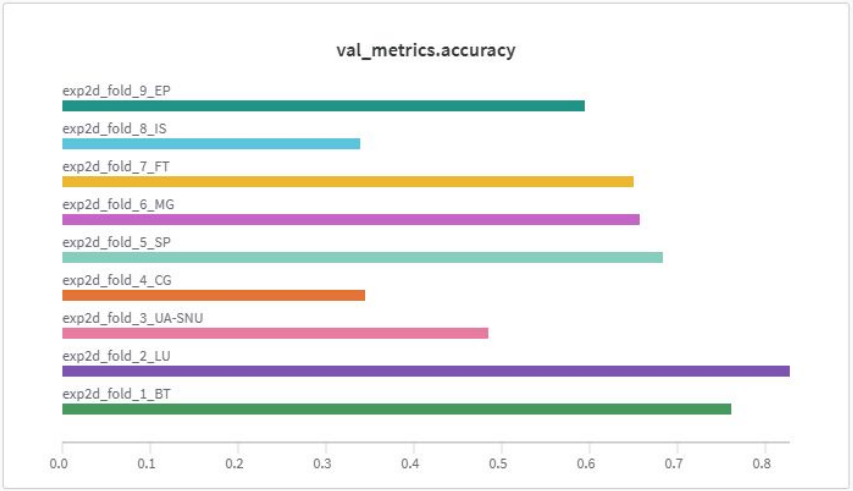
Led to extremely mixed results. Again, certain applications dominate the training space and causes issues when validating on smaller applications.

I had to research the nature of application. With some of the folds, we can expect it not to generalize well if for example we train on 9 applications for vectorization benchmarks and then test on an application for floating point calculation.

LOOCV

While narrowing down the largest benchmark helped a bit, the differing sizes in applications was the biggest factor in accuracy.





Further Limiting Applications

- Based on the results so far, I wanted to see the performance of the model under a simpler version of the problem once more.
- I switched to selecting the largest application in NPB (SP), which helped stabilize the variance.
- The initial runs compared random stratification against majority stratification (where the majority class of the loop group represents the entire loop group’s behavior).



In training, they both fit the model similarity, but in validation the random strat got ~89% and majority got ~93%. This run was across the default hyperparameters, run 10 times. Only ~0.01 standard deviation.

Further Limiting Applications

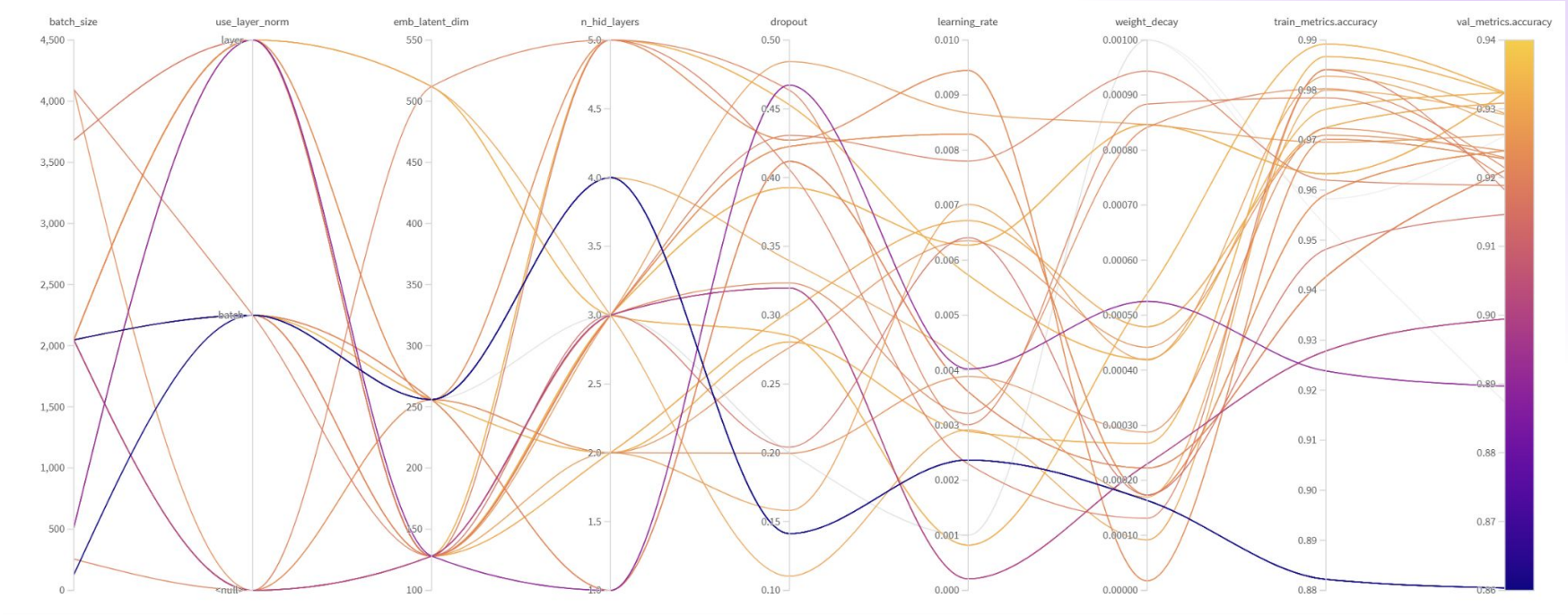
Given promising results from those tests, I performed many runs with different hyperparameters doing a random search.

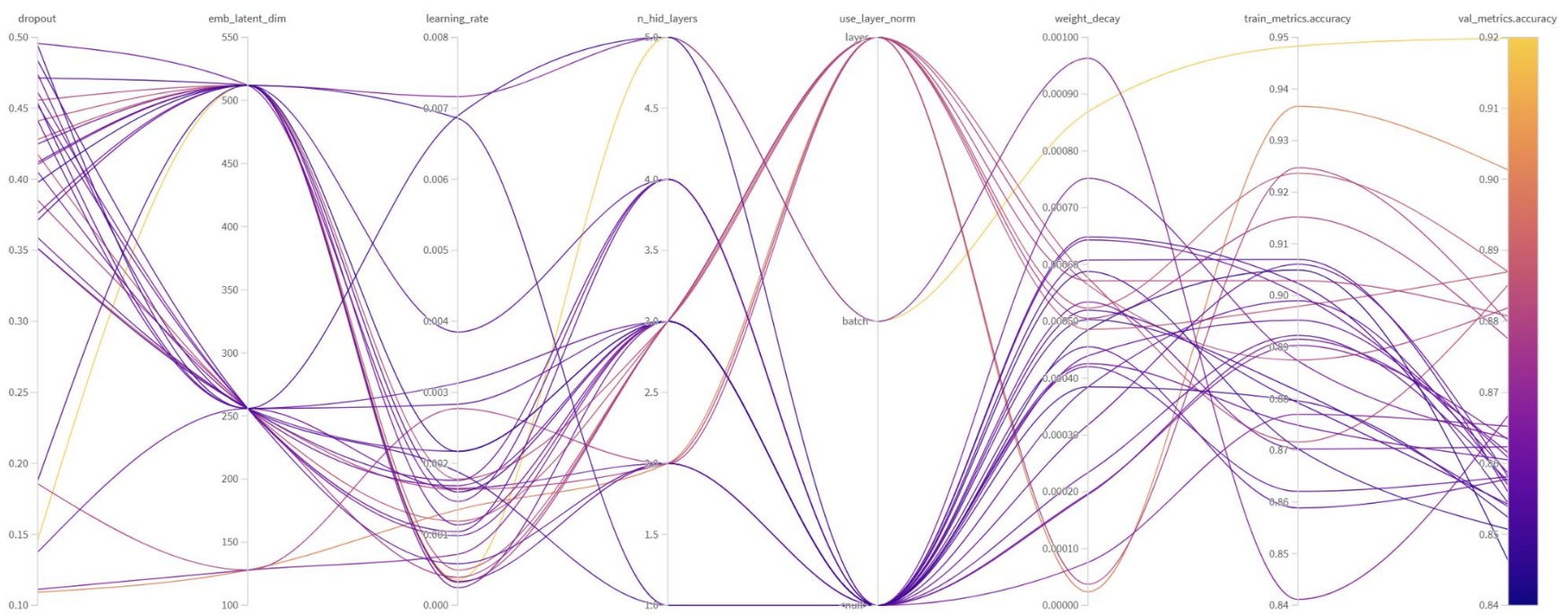
Conclusions

The best I got was 93.7% and the worst at 86%, so my conclusions were the hyperparameters weren't having much of an effect–I couldn't tank the performance of the model with some only minorly bad parameters.

I could make some minor conclusions:

- Got good results with both batch and layer norm as expected.
- Dropout at 0.2 or 0.3 worked best
- emb_latent_dim was best at 256
- Good results with the learning rate at 0.0001 and 0.0006
- Better to do hidden layers between 2 - 4


batch_size
use_layer_norm
emb_latent_dim
n_hid_layers
dropout
learning_rate
weight_decay
train_metrics.accuracy
val_metrics.accuracy
layer
batch
null


dropout
emb_latent_dim
learning_rate
n_hid_layers
use_layer_norm
weight_decay
train_metrics.accuracy
val_metrics.accuracy
layer
batch
0.50
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
550
500
450
400
350
300
250
200
150
100
0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.000
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.00100
0.00090
0.00080
0.00070
0.00060
0.00050
0.00040
0.00030
0.00020
0.00010
0.00000
0.95
0.94
0.93
0.92
0.91
0.90
0.89
0.88
0.87
0.86
0.85
0.84
0.92
0.91
0.90
0.89
0.88
0.87
0.86
0.85
0.84

Next Steps

- The research into improving these results is still ongoing.
- I had also looked into improving the embeddings themselves a bit as well as changing how the active transformations are represented (currently a long binary string). We'd change that to be something not as sparse when only a few optimizations are applied.
- Implementing 3-fold validation with BT, SP, and LU applications since they are the 3 largest and also share similarities.
- Eventually building up to combining more benchmarks and building up the dataset again while monitoring its results on the training and validation sets as before.

Thanks!

Any Questions?