Atmospheric Physics Climate Model

Author: Dominic Welsh
Student ID: 127909

Introduction

Fully understanding Earth's climate and all the different parts of it that combine and interact to make various phenomena happen is extremely difficult. There are vast numbers of inputs that affect Earth's systems in even just a small local region, so trying to expand that across the entire globe is no simple feat. But having a deeper knowledge of Earth's climate, both how it currently works and what could happen if there are changes to those inputs, is crucial for knowing the affects our actions today have on these systems.

In order to answer these questions, climate models have been created that allow scientists to research and better understand Earth's systems. Most climate models have grid sizes of 50 to 100 km, and use approximations for processes that occur at smaller scales than this, such as clouds, storms, rainfall, etc. However, these approximations are not always adequate, so multiscale models were developed to accurately calculate these subgrid processes. But these multiscale models are far more computationally intense, and as such are difficult to use all the time, which is becoming more and more of an issue due to climate change and how rapidly "normal" climate processes are changing. This is where machine learning models come in to play.

Once trained, machine learning (ML) models require far less processing power than multiscale models. The goal of this project is to create an ML model that is trained on data from the Energy Exascale Earth System Model - multiscale modeling framework (E3SM-MMF), a multi-scale climate model backed by the U.S. Department of Energy. By training an ML model on this data, we will be able to achieve results similar to E3SM-MMF but at a much more accessible level, allowing the model to be more useable in a variety of scenarios.

Data

For this project, the data is provided by Learning the Earth with Artificial Intelligence and Physics (LEAP), a Science and Technology Center that's part of Columbia University (Leap Columbia University, 2024). The dataset can be found through LEAP's Hugging Face page, https://huggingface.co/LEAP, although for the Kaggle competition only a portion of the low-resolution dataset is used.

In this dataset, each row "corresponds to the inputs and outputs of a cloud-resolving model (CRM) in E3SM-MMF at some location and timestep" (*LEAP*, n.d.). The input variables are various relevant measurements, such as air temperature, surface heat flux, and wind speed, and the target variables are the resulting effects, such as moistening tendency, rain rate, and wind acceleration (*LEAP*, n.d.). There are only 25 input variables and 14 target variables, but

because 9 input variables and 6 target variables span the entire 60 levels of an atmospheric column, there are a total of 556 input columns and 368 output columns (*LEAP*, n.d.).

Due to the large number of features, first a test was done to see which features had very small variance (standard deviation of less than 0.000001) in their data, as these features were most likely not useful to predict results. In the larger_model_features.ipynb notebook, this resulted in a list of 238 features, which was over half the input data. To further check the importance of these features, an additional test was done to see if any of these small variance features had a correlation coefficient greater than 0.1, which returned 153 features, leaving 82 features with very small variance and relatively no correlation with the output variables, and so they were removed. This reduced the number of features from 556 to 474.

While there is a variety of units in the dataframe, all of the data types are floats. In order to have the model not be confused by the different magnitudes of float ranges, a scaler was used to normalize the data and bring all the variables to the same range. To determine the best scaler, first a test was done on the 474 remaining features to see how many of them fail a normal distribution test, of which 473 of them rejected the null hypothesis of having normal distributions. So after splitting the data into train, validation, and test sets, a MinMaxScaler was used to fit and transform the training input data, and also used to transform the validation and test data.

Data Organization

The Kaggle competition does provide a test data set that is completely separate from the training data set. However, this test set is specifically for people to test the trained model on and submit their model's results to the competition, and therefore it does not have target data, only the input features. Because of this, data was only used from the provided train.csv file.

Due to the very large size of the train.csv file, just the first 1 million rows of data were used for the model. Originally the final model was trying to use 2 million rows, but memory issues were encountered and so the size had to be reduced. Using scikit-learn's train_test_split, this data was first split into two sets, with 70% being reserved for training. Using train_test_split again, the remaining 30% was split in half, to create both a validation data set and a testing data set.

Methods

The model is set up as a neural network comprising of linear layers, with 5 layers in total. After each full connection, the features are normalized using BatchNorm1d, followed by the Leaky ReLU activation function, before some of the features are dropped to reduce overfitting. Because this is a multi output regression model, mean squared error was chosen as the

criterion to calculate loss, via PyTorch's MSELoss(), and Adam was chosen as the optimizer algorithm.

For hyperparameters, GridSearchCV was used to try and identify optimal parameters in the notebook hyp_param_optimize.ipynb. While some of the test values were just factors around default values, the value for hidden layer size were a bit more calculated after finding an article on Medium. There, the author talked about a few rule-of-thumb methods for determining neurons in hidden layers, such as keep the size between the sizes of the input and output layer, or that "the number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer" (Krishnan, 2022). This gave a bit more focus on what values that hyperparameter should be tested at. The resulting values are as follows:

batch_size: 64
hidden_size: 694
learning_rate: 0.01
regularization_param: 0.0001

One issue that was encountered when optimizing hyperparameters was that the original model setup could not be used with GridSearchCV, and so it had to be reformatted. The reformatted model was also only calculating results based on training data, meaning that dropout percentage could not be optimized, which turned out to be pretty important. This is because the default dropout percentage of 0.5 lead the model to greatly overfit the training data and not perform well with the validation or test data. After some additional trial and error, it was discovered that a low dropout percentage of 0.1 allowed the model to perform well for both training and validation data sets.

Another method that was used to help optimize the model was using a scheduler, specifically PyTorch's ReduceLROnPlateau. This allowed the model to start reducing its learning rate once the validation loss value stopped increasing, helping prevent the model from overshooting an ideal position. A previously submitted competition notebook by user Egor Trushin was very helpful in helping understand how to use this scheduler (Trushin, 2024). If the validation loss had not improved after 5 epochs, the learning rate was reduced by a factor of 0.1. In addition, a custom check was added, and if the validation loss had not improved after 3 reductions in a row, or 15 epochs, the model automatically stopped. This scheduler was also why the model was given a high number of epochs, because the goal was for it to run until it was unable to improve the validation loss.

Because the Kaggle competition is evaluating submissions based on an R^2 metric, PyTorch's R2Score metric was used to indicate the performance of the model. This was also needed due to the multiple regression outputs, so the R^2 score gives one singular number to express how well the model is performing for all targets. However, based on the results of the model, which will be talked about more next section, it is likely that this metric is not accurately evaluating the model.

The large data set made it very important to move the model to the computer's GPU while running, in order to run more processes in parallel. As this model was run on MacOS, the

model (and relevant data sets) were moved to the GPU via Metal Performance Shaders (MPS). If running on non-Mac computers, or Mac computers without Apple silicon or AMD GPU's, the device would need to be changed to suit that computer.

Results

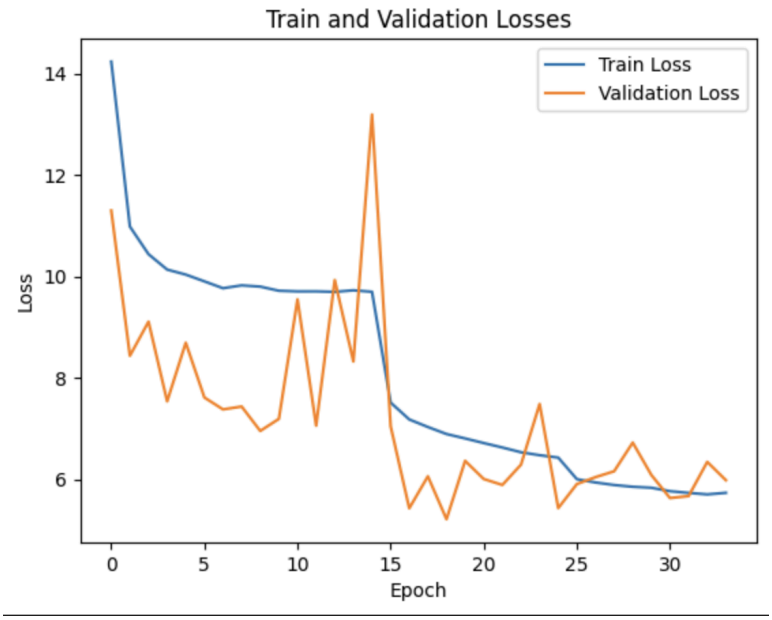

Figure 1 - Train and Validation Loss

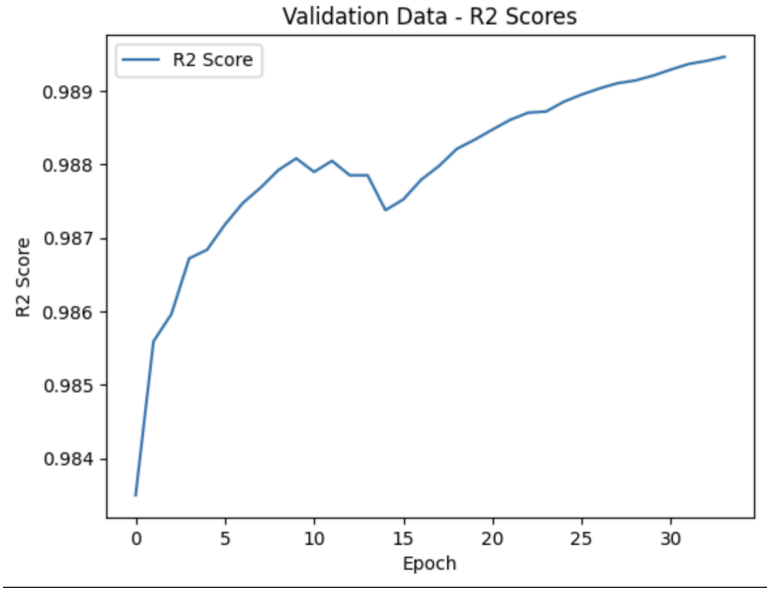

Figure 2 - R^2 Scores of Validation Data

Although the model ran for 34 epochs, the best model was at epoch 19, with the results:

Train Loss = 6.89
Validation Loss = 5.21
R^2 Score = 0.988

When running this model on the Test data set, the Test Loss = 6 with an R^2 Score = 0.99.

Analysis

As can be seen in Figure 2, even from just the first epoch the model had very high R^2 Score of over 0.98. But in Figure 1, the loss never goes below 5 for the training data, and fluctuates between 5 and 14 for the validation data. The extremely high R^2 Scores say that our model is practically perfectly accurate, which is an immediate red flag. This is confirmed when comparing with other models on Kaggle's competition page, where many of them have losses below 1 for training and validation data sets, but no R^2 score that is above 0.9. In fact, the number 1 performing model on the leaderboard currently has an R^2 score of 0.79 when using the test data set, compared to this model which returned an R^2 score of 0.99 on it's test data, indicating that there is probably an issue with how this metric is being used.

One problem could be not enough time spent selecting features. In another article on Medium, the author pointed out that R^2 value is sensitive to unnecessary features, and adding more variables will only make the R^2 value stay the same or increase (Wohlwend, 2023). So it could be that more work should be done on removing unnecessary features, to reduce noise and get a cleaner R^2 value. It is also possible that the metric was set up incorrectly, and while it returns values that look correct, they are not the actual results.

In terms of the loss not getting close to 1 for any of the data sets, it would probably be worth trying out a model with layers other than only fully connected layers. Reflecting on my work, I created a model that seemed to function well enough, but didn't spend time thinking or researching on what better layer connections might be, instead focusing on trying to figure out hyperparameter optimizations or other work. Which is especially notable due to even mentioning in my project proposal the idea of using a convolutional neural network for climate data, but needing to do further research on it's potential. As result, although the model appears to be evaluating extremely well, it most likely would not accurately predict output values at a statistically significant level.

Conclusion

Climate is an extremely complex topic, and so it makes sense that attempting to model it would be similarly so. Although the model may not accurately replicate the results of E3SM-MMF, it has been a very informative project that has taught me a lot on creating models for large data sets, providing me a strong foundation for the next model while also showing how much more there is to learn.

References

Krishnan, S. (2022, August 6). How to determine the number of layers and neurons in the hidden layer? *Medium*. https://medium.com/geekculture/introduction-to-neural-network-2f8b8221fbd3. Accessed 2024, July 02.

Leap Columbia University. (2024, June 19). *LEAP - an NSF science and technology center*. https://leap.columbia.edu/. Accessed 2024, July 04.

LEAP - Atmospheric Physics using AI (ClimSim) | *Kaggle*. (n.d.). https://www.kaggle.com/competitions/leap-atmospheric-physics-ai-climsim/data. Accessed 2024, July 04.

Trushin, E. (2024, May 16). *[LEAP] FFNN/PyTorch*. Kaggle. https://www.kaggle.com/code/egortrushin/leap-ffnn-pytorch. Accessed 2024, June 27.

Wohlwend, B. (2023, August 12). Regression model evaluation metrics: R-Squared, Adjusted R-Squared, MSE, RMSE, and MAE. *Medium*. https://medium.com/@brandon93.w/regression-model-evaluation-metrics-r-squared-adjusted-r-squared-mse-rmse-and-mae-24dcc0e4cbd3. Accessed 2024, July 04.