Optimizing Wind Forecasting with Graph Neural Networks

Author: Sigurd Almli Hanssen

UiT - Norges Arktiske Universitet

ML group, department of Physics and Tech

June-August 2024

1 Task

My summer project focused on extending the work initiated by Patricia Asemann in her master's thesis "Offshore Wind Prediction with Graph Neural Networks" Asemann [2024]. She had started developing a Graph Neural Network (GNN) model for wind forecasting around the Goliat station in the North Sea, using data from the CARRA reanalysis and Sentinel SAR images.

The primary objective was to investigate whether GNNs could accurately forecast wind using CARRA data as input and Sentinel data as ground truth for wind speed. A central aim was to experiment with various network architectures, exploring how modifying the number of layers or the size of the embeddings could affect model performance. Additionally, I aimed to implement skip connections in the network to mitigate oversmoothing, improving the viability of deeper Graph Convolutional Networks (GCNs).

A secondary goal of the project was to refine the code in preparation for future work.

2 Implementations and Changes

2.1 Pre-processing

For pre-processing, I was provided with a Jupyter notebook containing the necessary functions to generate the graph data from CARRA and Sentinel datasets required for training GCNs. However, the cells in the notebook were not in chronological order. My first task was to determine the correct sequence for running these functions and restructure the code accordingly.

In doing so, I introduced checks to prevent overwriting existing files, ensuring that pre-processed steps already completed would not be repeated unless necessary. This change allowed the addition of new images without disrupting the existing dataset and provided safeguards in case the code crashed mid-process.

Additionally, I adjusted certain parts of the code that previously opened and modified existing files to create new ones instead. Although this approach increased the overall dataset size, it eliminated the risk of corrupting the dataset if the code was run multiple times.

Lastly, I performed some general tidying up of the code. However, since I had limited background knowledge in this field, I refrained from making substantial changes to the core pre-processing functions.

2.2 WindNet

For the GCN named WindNet, I made several significant improvements. My first goal was to utilize a GPU to accelerate training, which required adapting the code for Google Colab and CUDA compatibility. This optimization reduced training time from 1-2 minutes per epoch to just a few seconds—an improvement of 20-60 times—making it feasible to experiment with different architectures more efficiently.

Once I had the code running on CUDA, I started working on the network itself. My first change here was the introduction of a scheduled learning rate, as I noticed that the loss plots from the master thesis did not settle well due to the use of a fixed learning rate.

I noticed that loading and creating the datasets took a long time every time you wanted to start a new run. I implemented dataloaders and the ability to save them and load them up again in following runs. This saved me about 20 minutes each time I had to restart my session. This might be a google colab related problem though, as the speed to load data from google drive is very slow.

Next, I made WindNet more flexible by generating the network architecture through a loop, using layer depth and embedding size as input parameters. This made it easier to modify architectures between runs and facilitated testing multiple architectures simultaneously by simply iterating over a list of configurations.

I also introduced skip connections, adding them as an optional parameter to test architectures both with and without skip connections.

2.3 Summary of Changes

- Reorganized and cleaned up the pre-processing code.
- Added functionality to skip previously completed steps during pre-processing.
- Prevented the code from overwriting and potentially corrupting important files.
- Adapted WindNet for compatibility with Google Colab and CUDA to enable GPU use.
- Implement ability to save and load up dataloaders.
- Enhanced WindNet's flexibility to accept varying architectures as input.
- Introduced skip connections to the network.
- Converted the code into a standard Jupyter notebook format for use in VS Code.

3 Results

3.1 Scheduled Learning Rate

The first modification I implemented after getting the code running was a scheduled learning rate. This adjustment helped the network settle into minima, preventing it from oscillating in and out. Before this change, the loss plots exhibited volatility, similar to the one shown in Figure 1 from Patricia's thesis. This volatility made it difficult to compare results after a fixed number of epochs, as performance could fluctuate significantly between epochs.

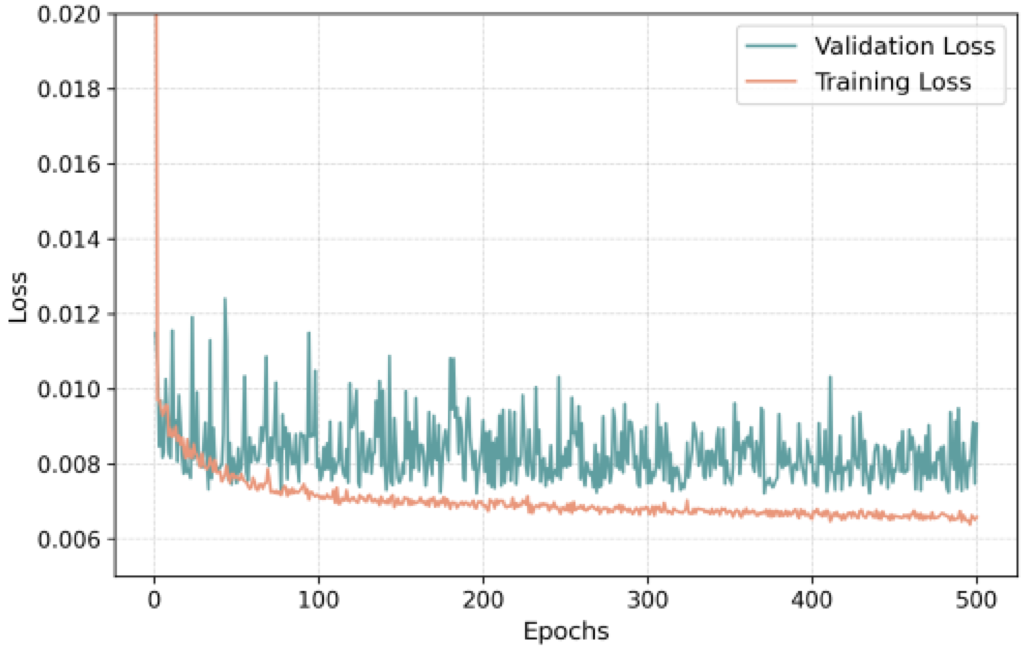


Figure 1: Plot from Patricia's master's thesis showing MSE without a scheduled learning rate.

After implementing the scheduled learning rate, the loss functions stabilized, making it possible to compare models more accurately.

3.2 Experimenting with Architectures

In my experiments, I did not observe a significant improvement in Mean Squared Error (MSE) from varying the layer depth, embedding size, or from introducing skip connections.

3.2.1 Varying Architecture without Skip Connections

I tested multiple variations in layer depth and embedding size. Figure 2 illustrates the loss functions for networks with varying layer depths from 2 to 40 with embedding size 32 over 80 epochs. 80

Epochs was around the point where many architectures began reaching their minimum.

We expected deeper networks to perform worse due to oversmoothing. Indeed, the 40-layer network performed poorly, as anticipated. Surprisingly, however, the second-deepest network, with 20 layers, converged faster than the others and performed well, achieving the third-best overall performance.

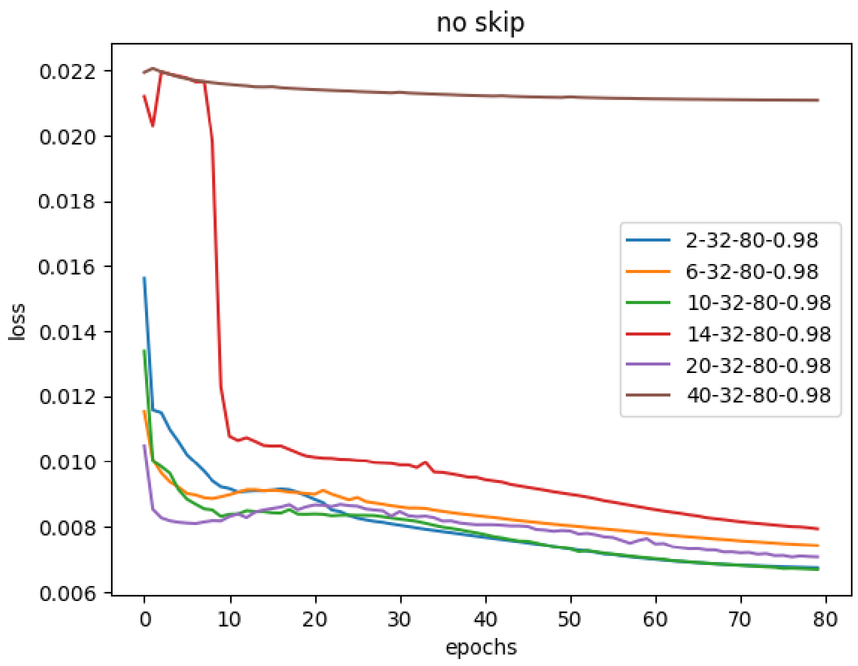


Figure 2: Loss functions for networks with varying depths, without skip connections.

3.2.2 Varying Architecture with Skip Connections

Skip connections were introduced to mitigate oversmoothing, allowing information to bypass certain layers. This approach was expected to improve the performance of deeper networks.

With skip connections enabled, the 40-layer network converged and became competitive, unlike in the previous tests. However, all models, excluding the very deep ones without skip connections, converged to an MSE within the range of 0.0062 to 0.007, as shown in Figure 3.

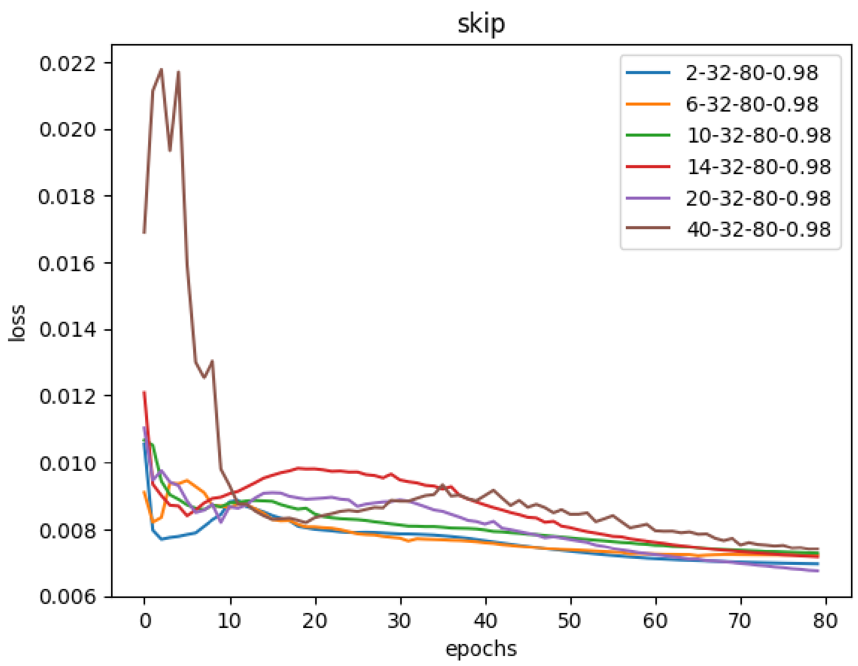


Figure 3: Loss functions for networks with varying depths, with skip connections.

A comparison of the epoch-80 loss values for networks with and without skip connections is shown in Figure 4. As the figure illustrates, most models converged within the expected range.

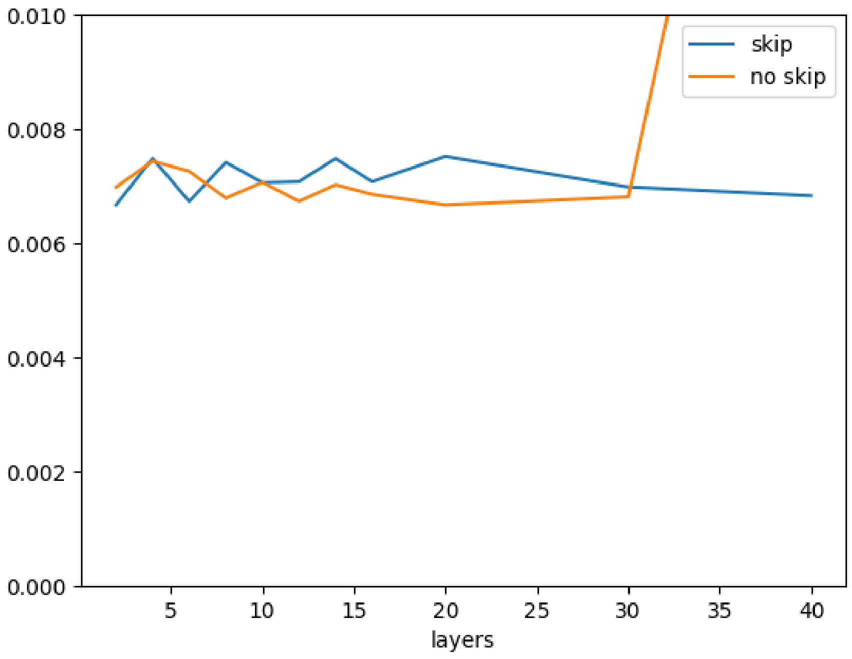
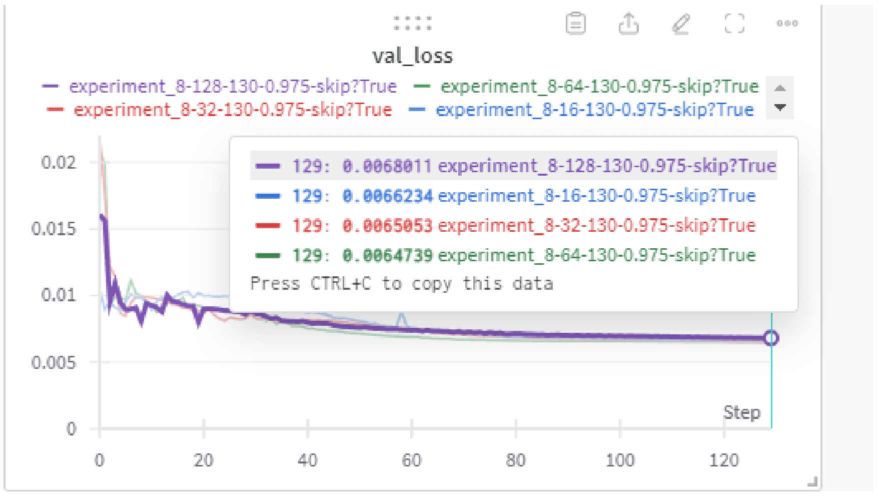


Figure 4: Comparison between runs with and without skip connections.

I also tested varying embedding sizes, from 16 to 32 with layer depth 8, over 120 epochs, but again, no significant improvements were observed.



val_loss
experiment_8-128-130-0.975-skip?True
experiment_8-64-130-0.975-skip?True
experiment_8-32-130-0.975-skip?True
experiment_8-16-130-0.975-skip?True
0.02
0.015
0.01
0.005
0
Step
0
20
40
60
80
100
120

```
129: 0.0068011 experiment_8-128-130-0.975-skip?True
129: 0.0066234 experiment_8-16-130-0.975-skip?True
129: 0.0065053 experiment_8-32-130-0.975-skip?True
129: 0.0064739 experiment_8-64-130-0.975-skip?True
Press CTRL+C to copy this data
```

Figure 5: Loss functions for networks with varying embedding sizes.

To verify these results, I also conducted multiple runs with the same architecture. As shown in Figure 6, the variation between runs of the same architecture was larger than the variation between runs with different architectures.

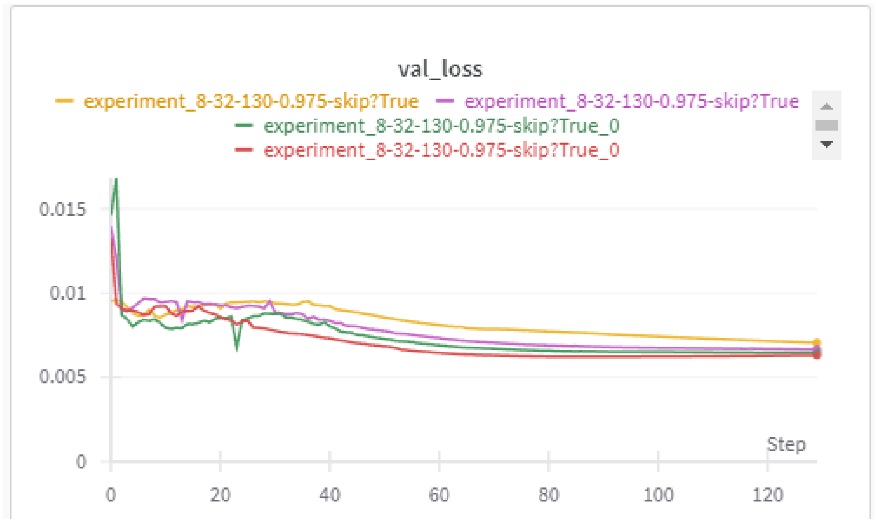


Figure 6: Multiple runs with the same architecture.

4 Conclusion

Given the larger variation between multiple runs of the same architecture compared to variations in architecture, it is difficult to conclude that any specific architecture was consistently better than another. The differences in performance appear to stem more from initial conditions and randomness than from architecture changes.

5 Challenges

The project had a slow start. I spent the first couple of weeks gathering data, reviewing the master's thesis, and attempting to get the code running. Uploading and downloading files, along with troubleshooting compatibility issues between my laptop and Google Colab, consumed much of my time. Additionally, I encountered issues with missing data in uploads, requiring re-uploads, which resulted in approximately 70 hours of waiting spread over a week.

Work on the pre-processing was also a slow and laborious task. As has been mentioned the code cells was not in the correct order, so I had to use trial and error to identify the correct sequence. Furthermore, running the pre-processing step took around 16 hours, and I was unable to make it CUDA-compatible, forcing me to rely on my laptop for the pre-processing part of this project.

References

Patricia Asemann. Offshore wind prediction with graph neural networks. Master's thesis, UiT Norges arktiske universitet, 2024.