CSCI 378 Final Project Writeup

Bram Schuijff

May 10, 2025

1 Background Research

Prior to implementing any networks, I originally tried to examine physics-informed neural networks (PINNs). I read both research papers which introduced the idea and parsed the PINN library itself. Unfortunately, I abandoned this idea due to the amount of physics and partial differential equation knowledge it required, both of which I am lacking in. I also lacked an interesting dataset on which I could actually train a PINN, and I did not know where to start looking for such a dataset.

The majority of my background research following my abandonment of PINNs was in trying to figure out how to implement recurrent neural networks. Most of this implementation occurred before we went over temporal convolutional networks in class. Therefore, I had to sift through Torch documentation to figure out how they worked. This started with trying LSTM networks and, when I realized they were overkill for the job and took far too long to train, moving to temporal convolutional networks.

Another piece of research I had to do was figuring out precisely which dataset to use. There were a few options such as NASA's Daymet dataset, but I eventually decided on UC Berekeley's temperature anomaly dataset. This is because I had a very hard time trying to adapt the Daymet dataset to a machine learning context due to the complex geospatial file type. Additionally, due to the dataset's size, training networks took a ridiculously long time even on Weftdrive.

2 The Problem

The main problem I am attempting to solve is to predict daily temperature anomalies. Based on the historic mean temperature of the planet, how much did the global temperature of that day deviate from that mean? UC Berkeley has a dataset with temperature anomaly information for each day between 1880 and September 2022. My goal was to use this dataset to train a network which could predict future temperature anomalies. In this sense, I was looking at a time series forecasting problem. This problem is of practical interest because it can demonstrate the impact of climate change on the average temperature of our planet over time. Additionally, this network could predict the future impacts of climate change.

3 Implementation

I implemented a temporal convolutional network. I did this as opposed to an LSTM network for a few reasons: the number of parameters in a moderately-complex LSTM caused horrific overfitting on my low-dimensional dataset, LSTM networks had ludicrous file sizes that made debugging them painful, and they took far too long to train. Furthermore, I saw very little difference in performance between an LSTM network and the much smaller temporal convolutional network, making the latter the obvious choice.

The network itself is relatively simple, accepting an argument `layers` in its constructor which is a list of positive integers. The network then contains a number of hidden temporal convolutional layers equal to the length of the list and where each layer has a number of output channels in accordance with its associated list element. Each layer also has a `ReLU` activation function and one-dimensional padding in accordance with what we went over in class. The layers also get increasingly dilated as we go down the hidden layers, allowing for the network to observe more of its context window at once. Finally, our classifier is just a kernel of size 1 which converts our data into a prediction for our time series with 1 channel corresponding to the real number we are trying to predict.

I have two implementations for what our predictions are supposed to model. Given a context window of size k, we either attempt to predict the element immediately following our context window or we attempt to predict the k elements following our context window. One of these problems is clearly harder than the other as the first is a sub-problem of the second. The way that I implemented this is with a global boolean variable called `predictive`. If it is `False`, we only attempt to predict the next element. If it is `True`, we attempt to predict the entire next context window. `predictive` is read in the `TempAnomalyDataset` object which is a subclass of the `torch.Dataset` object which is how we actually load our data.

I started with the next-element case but quickly found that it overfit with even tiny models. Additionally, it was very bad at producing predictions for future temporal anomaly data on even small cases because it often got into feedback loops as it was fed its own predictions to forecast the time series. I then moved to the full context window case which, while it took longer to train, produced significantly more desirable models for my use case. Overfitting became much less of an issue because of the increasing difficulty of the problem. Additionally, this framework did not require the network to be fed its own inputs in order to forecast the time series; instead, we gave it a tail of the input data which it used to produce an equally-sized forecast.

The final piece of our implementation is some simple `matplotlib` code which allows us to visualize our predictions. This can be found in the `vis.py` file where blue represents our real dataset, red represents our predictions over the dataset, and yellow represents our forecast.

4 Results

My models were able to pretty accurately predict the dataset with minimal overfitting even at very large network sizes. I tested this with Tensorboard and varied my architecture, learning rate and weight decay to achieve optimal results. For smaller architectures, it seems that high epoch ranged work best with a relatively high learning rate. These architectures are able to continually learn for long periods of time with their validation losses steadily decreasing. In the end, a MSE loss of approximately 0.08 was about as good as I could get. Figure 1 demonstrates that our network was able to accurately predict data within the dataset.

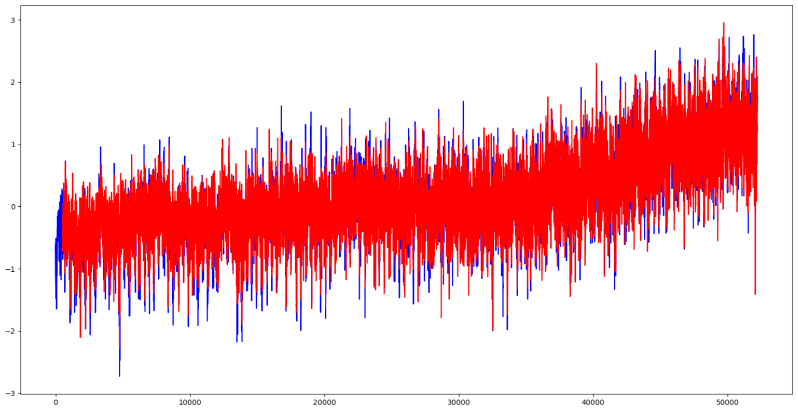

Figure 1: Relatively large architecture demonstrating accurate predictions

Unfortunately, despite this relatively good loss, my forecasting left a lot to be desired. It seems that no matter how small of a window I asked my networks to forecast, they were simply incapable of producing good results. I tried everything I could think of to get good results and nevertheless, my results were lacking. Larger networks had a tendency to squash they predictions in a very small range of values while smaller networks would have extremely erratic predictions. Evidence for both cases is provided in Figures 2 and 4.

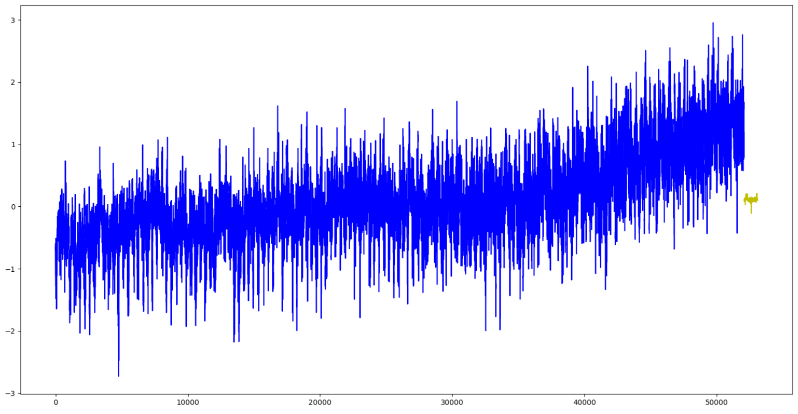

Figure 2: Large architecture demonstrating squashed predictions

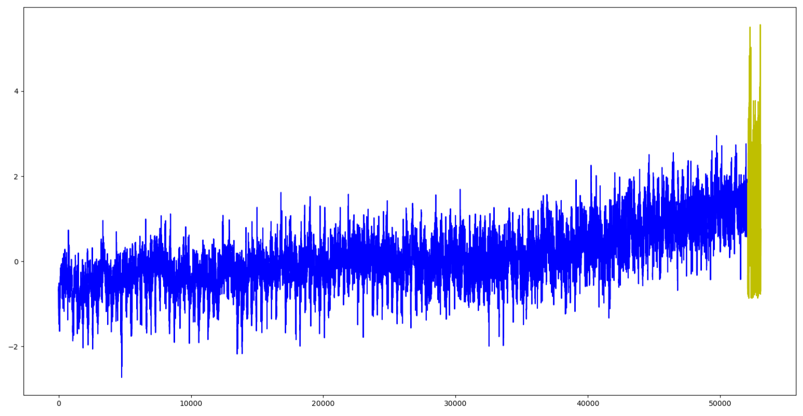

Figure 3: Small architecture demonstrating erratic predictions

In the above figures, the blue line represents our real dataset and the yellow line represents the model's forecast. These are both demands for the model to forecast the next 1000 days of temperature anomalies. The smaller model has predicitons which are all over the place while the larger model tends to squash its values just above 0. I cannot for the life of me figure out why this is the case and I have not been able to fix it. There appears to be no middle-ground between wildly erratic predictions and completely squashed predictions. Even when a model demonstrates good behavior over the validation set, this behavior persists. Perhaps it is because of how I am getting the model to forecast, but I have not been able to root out the source of the bug.