CS230

Daily Streamflow Prediction Using Deep Learning: A Case Study on Russian River, CA

Xinle Yao (Grace)
Stanford University
xinleyao@stanford.edu
SUID: xinleyao
Project Category: Generative Modeling

1 Introduction

The project objective is to explore the potential of using neural networks to predict average streamflow discharge of the next day using streamflow information from previous days together with other ancillary data. As demonstrated in the project proposal, accurate predictions of future streamflow are essential to water management and planning as well as for hazard prevention.

Conventional streamflow models, such as the autoregressive moving average, are commonly used for generating potential future streamflow time-series. However, due to the use of linear regression, these classic models are not able to represent the highly nonlinear distribution of streamflow patterns. This is where deep learning can be used to improve the prediction and the forecasting.

2 Dataset

To proceed with the project, I identified the Russian River as my region of interest. Russian River locates in Northern California crossing nine different counties. It is an important part of the community by providing recreational services and supporting the local tourism and business. The streamflow dataset I obtained is from the USGS WaterWatch program (url: `https://waterwatch.usgs.gov/`)[1]. From USGS, the discharge (cubic feet per second) at the Russian River Hopland gauging station is downloaded at the time step of every fifteen minutes. These data are later aggregated to average daily flow.

Other information including the precipitation, air temperature, and soil temperature were retrieved from the California Irrigation Management Information System (CIMIS) of the California Department of Water Resources (url: `https://cimis.water.ca.gov/`)[2]. CIMIS has a field monitoring station nearby the USGS Hopland gauging station, which measures hourly precipitation, temperature, and solar radiation. For the purpose of this project, I retrieved these hourly data (precipitation, air and soil temperatures) and aggregated them into daily data. After obtaining all the data from different sources, the available data time range ended up being from February 1991 to September 2020.

3 Methods and Approaches

Since my goal is to explore the potential of using neural networks to predict future streamflow, I decided to approach this project using three different methods. First, I trained an linear model to serve as the baseline comparison. Then, I built a fully connected model to try out its performance on streamflow prediction. Lastly, I also tested an RNN Long Short Term Memory (LSTM) model

considering it is one of the highly recommended architecture in the current literature for streamflow prediction [3,4,5].

3.1 Data Pre-processing for Baseline Model and Fully Connected Model

Before building the baseline (linear) model and the fully connected model, I processed the data in the following manner. For each sample, 17 input variables were generated. As an example, the following vector represents the input of Day i. The first row is the Julian day of Day i, which ranges from 0 to 365 or 0 to 366 depends on the Year. This input row is used to indicate the position of the day within a year considering streamflows vary a lot depending on the season. The second and third row of the sample vector are the averaged daily air and soil temperatures at the CIMIS station near the river. The fourth to eighth rows are the average daily discharge (aka flowrate) values of the previous five days in cubic feet per second. The ninth to eleventh rows are the average monthly flow rate one month ahead, two month ahead, and three month ahead. I included these three inputs hoping that they can be used as the indicator of whether the current year is a dry year or wet year. Lastly, the precipitation of Day i to i-5 are also included as the inputs. As a result, my input array has size of (m, 17), where m is the total number of available days. The corresponding label array has the size of (m, 1), where each label is the discharge rate of that day.

$$\begin{bmatrix} JulianDay \\ AvgAirTemperature(C) \\ AvgSoilTemperature(C) \\ QofDay_{i-1}(cfs) \\ QofDay_{i-2}(cfs) \\ QofDay_{i-3}(cfs) \\ QofDay_{i-4}(cfs) \\ QofDay_{i-5}(cfs) \\ AvgMonthlyQ_1(cfs) \\ AvgMonthlyQ_2(cfs) \\ AvgMonthlyQ_3(cfs) \\ AvgPrecip(mm) \\ AvgPrecipofDay_{i-1}(mm) \\ AvgPrecipofDay_{i-2}(mm) \\ AvgPrecipofDay_{i-3}(mm) \\ AvgPrecipofDay_{i-4}(mm) \\ AvgPrecipofDay_{i-5}(mm) \end{bmatrix}$$

To make the later training process more convenient, I decided to remove samples with Nan value for any of the 17 inputs. As a result, I ended up having only 7184 sets of samples and labels. One potential future step might be relaxing the constraint on the existence of Nan values so that more data will be available. (Note: if only removing samples without a label (discharge value), I can have around 10,000 samples.)

3.2 Data Pre-processing for RNN LSTM Model

The major step taken for the data pre-processing of the RNN model involves the normalization of input data by using the MinMaxScaler function from sklearn. Aside from that, I also tried to fill in the missing data points by using the historical average of that day of the year. By doing so, I obtained around 10,000 training samples. However, this approach might have introduced errors. To make the model more comparable to the baseline and fully connected models, I decided to use the time step of 90 days, meaning that the input of the model include the information of previous 90 days.

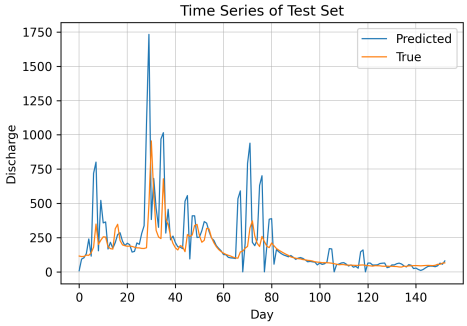
3.3 Test Set

Aside from the training set, I also reserved a set of test examples for the baseline model and fully connected model. This test set composed of the data from Oct 1st 2020 to May 30th 2021. These data points were not included in the training set due to the fact that they have not been officially 'approved' yet (Note: This is due to the processing procedure of USGS, where the organization will check all the data from each gauging station to make sure the validity of the data. Though the data itself is immediately available online, the certification process usually take some extra time.).

4 Things Tried and Results

4.1 Linear Baseline Model

Using a single Dense layer from Keras, I built this baseline model to show the performance of a linear model on daily streamflow prediction in this case. Below are two sample graphics showing the performance of the linear baseline model on the test set. From the time series plot, we can see that the linear model is missing evenly on predicting low and high flows. The scatter plot shows a fairly low R^2 value. Values from different metrics are shown in the table in next subsection.



4.2 Fully Connected Model

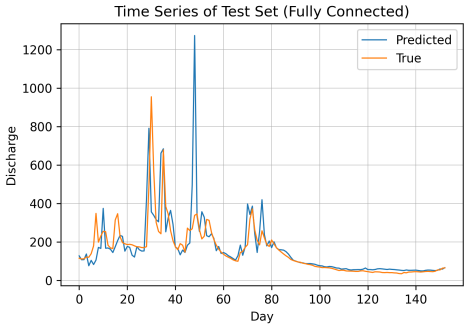
Continuing from the simple model built earlier for the milestone report, I kept the similar architecture and came up with this fully connected model.

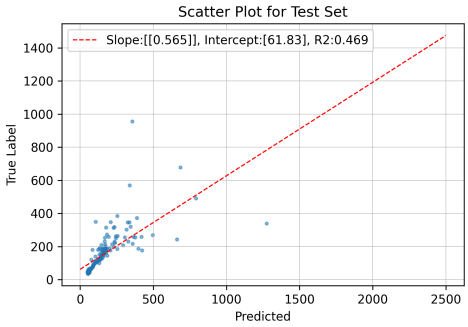
Layer (type)	Output Shape	Param #
dense_110 (Dense)	(None, 100)	1800
dense_111 (Dense)	(None, 90)	9090
dense_112 (Dense)	(None, 80)	7280
dense_113 (Dense)	(None, 70)	5670
dense_114 (Dense)	(None, 60)	4260
dense_115 (Dense)	(None, 40)	2440
dense_116 (Dense)	(None, 30)	1230
dense_117 (Dense)	(None, 20)	620
dense_118 (Dense)	(None, 10)	210
dense_119 (Dense)	(None, 1)	11

Total params: 32,611
Trainable params: 32,611
Non-trainable params: 0

As shown in the table above, the model consist of eight hidden layers (10 layers in total) using the Sequential class of Keras. I chose ReLU as my activation functions for all layers including the output layer. After trying with different optimizers from Keras, I ended up with Adam optimizer with learning rate of 0.001, β_1 of 0.9, β_2 of 0.999, and ϵ of 1e-07. For the purpose of the model, I decided to use mean squared error (MSE) as my loss function, as it measures the difference between true labels and predicted values. Lastly, I picked mean absolute percentage error (MAPE) and root mean squared error (RMSE) as my evaluation metrics. The use of these two metrics are also observed in literature of models with similar goals. Throughout the process of working on this fully connected model, I tested different number of layers. As the number of layers getting close to 9, the improvement on performance started to slow down. Therefore, I decided to settle with this 10-layer structure.

The table below summarizes the performances of this fully connected model and the linear baseline model. The graphics below demonstrate the sample time series and scatter plot for the test set.





	Linear Baseline Model	Fully Connected Model
$MAPE_{train}$	40.15	6.77
$MAPE_{val}$	40.15	6.78
$MAPE_{test}$	52.55	23.44
$RMSE_{train}$	487.53	120.08
$RMSE_{val}$	487.52	120.07
$RMSE_{test}$	201.29	114.55

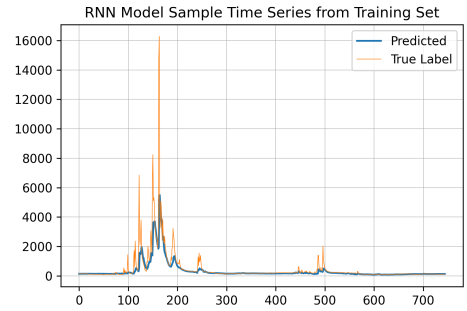
Though there's still room for major improvement, the simple fully connected model is already a huge improvement from the linear baseline model. As shown in the table, the performance of the fully connected model became worse for MAPE for the test set, whereas it improves for the RMSE, which is very interesting to me.

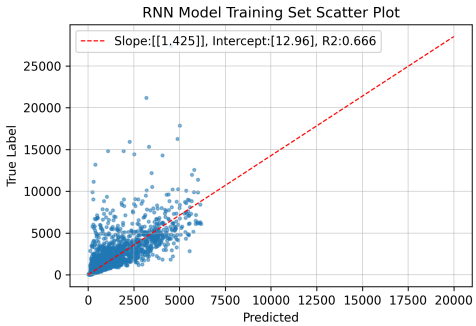
4.3 RNN LSTM Model

Lastly, I also tried a simple LSTM model using Keras with the architecture shown below.

```
Model: "sequential_32"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_31 (LSTM)               (None, 30)                4320
_________________________________________________________________
dense_31 (Dense)             (None, 1)                 31
=================================================================
Total params: 4,351
Trainable params: 4,351
Non-trainable params: 0
```

This simple model is inspired by the a paper published on IEEE in early 2020 [4]. Unfortunately, with the time limitation, I haven't gotten a chance to fully explore the architecture. The resulting performance of the model turned out not as good as the paper suggests. As shown in the sample graphics below (from training set), we can see that the RNN model performs different from the other two in a sense that it is under predicting the streamflow discharges, but doing fairly well on the low flow.





	LSTM
$MAPE_{train}$	21.11
$RMSE_{train}$	822.08

5 Conclusions and Next Steps

In general, I had a lot of fun exploring these models for streamflow prediction. As shown by the result, neural networks definitely improve the performance of streamflow prediction as compare to the simple linear model. I am looking forward to continuing working on the project to see where it will bring me.

References

[1] U.S. Geological Survey, 2016, National Water Information System data available on the World Wide Web (USGS Water Data for the Nation), accessed [April 18, 2012], at URL [http://waterdata.usgs.gov/nwis/].

[2] California Irrigation Management Information System. California Department of Water Resources. https://cimis.water.ca.gov/

[3] Hussain, D., Hussain, T., Khan, A. A., Naqvi, S. A. A., Jamil, A. (2020). A deep learning approach for hydrological time-series prediction: A case study of Gilgit river basin. Earth Science Informatics, 13(3), 915-927.

[4] Fu, M., Fan, T., Ding, Z. A., Salih, S. Q., Al-Ansari, N., Yaseen, Z. M. (2020). Deep learning data-intelligence model based on adjusted forecasting window scale: application in daily streamflow simulation. IEEE Access, 8, 32632-32651.

[5] Liu, D., Jiang, W., Mu, L., Wang, S. (2020). Streamflow prediction using deep learning neural network: case study of Yangtze River. IEEE Access, 8, 90069-90086.