Application of transformer and LSTM model for predicting Snow Water Equivalent

Students: Tenzin Worden tw2834 | Columbia University
Instructor: Pierre Gentine
Course: EAEE E4000 | Fall 2022

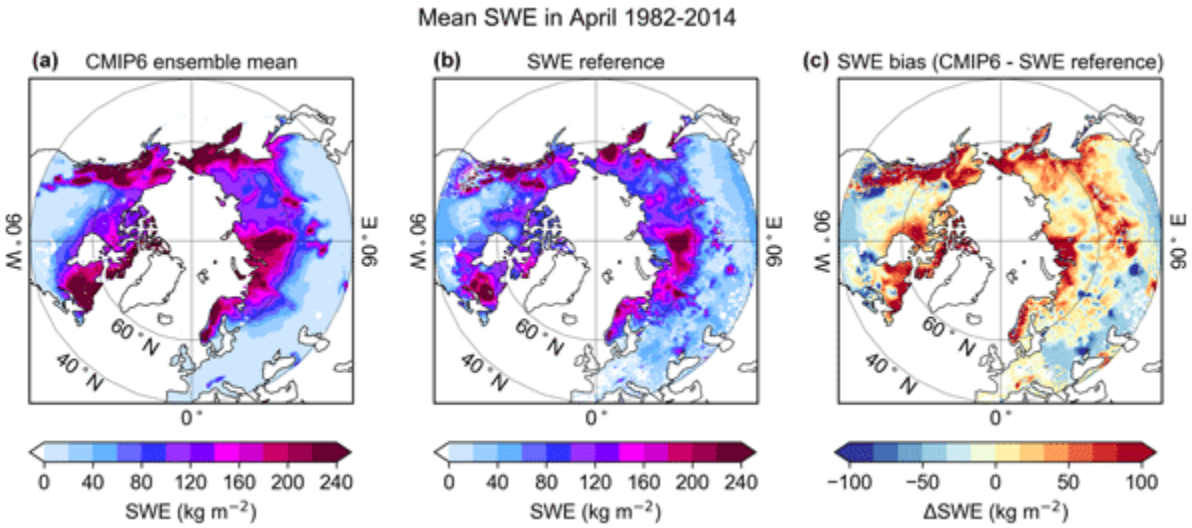


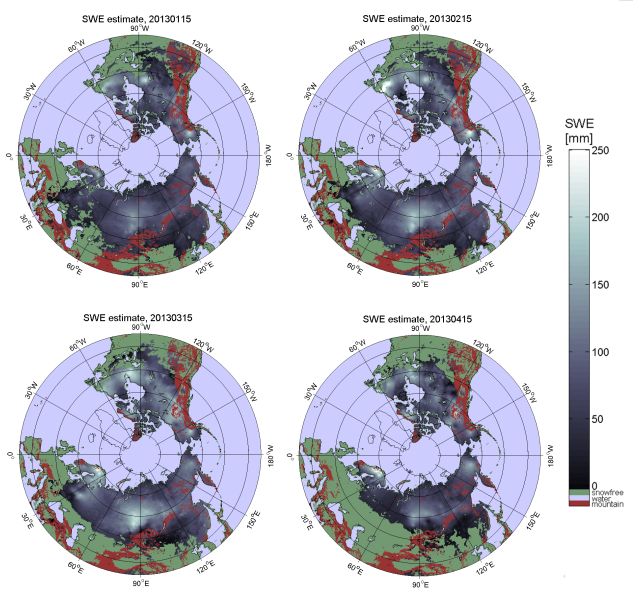
Table of Contents:

Key Prior Art/Reference(s): Konstantin Franz Fotios Ntokas1, Jean Odry, Marie-Amélie Boucher1 , and Camille Garnaud Using an ensemble of artificial neural networks to convert snow depth to snow water equivalent over Canada

1. Introduction

1.1. Background

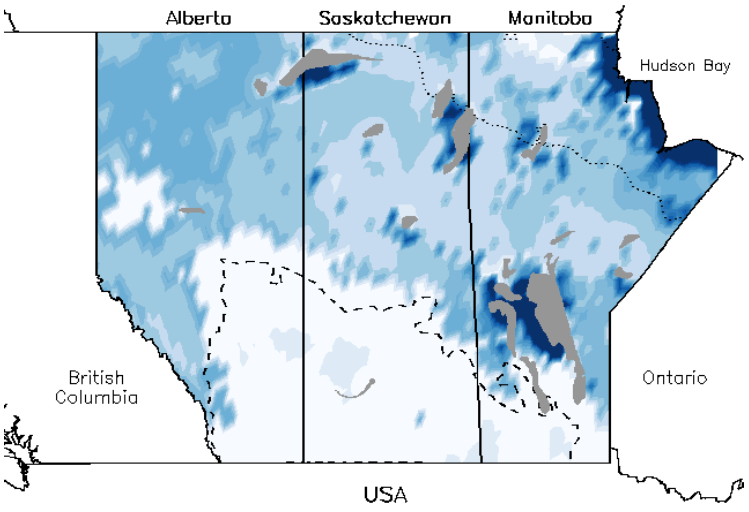
In the Northern Hemisphere, the seasonal snow cover is a significant factor in the global climate system. Thus snow cover is an essential variable in climate models. Previously, they found substantial uncertainties in Northern Hemisphere snow water equivalent (SWE) estimates. Snowmelt plays a significant role in the hydrological cycle of many world regions. For example, Casson et al. (2018) found that snow accumulation and melt are the main drivers of the spring freshet in the Canadian subarctic area. In addition, Pomeroy et al. (2011) showed that the snowmelt is the source of over 80% of the annual runoff in the Canadian Prairies. Therefore, an accurate prediction of the accumulation and melting of snow is vital in various applications, such as the management of reservoirs for hydroelectric power generation, irrigation and water supply, and climate impact studies. Therefore, the most commonly used parameter in hydrology is Snow water equivalent (SWE), which defines the total amount of water (solid and liquid) stored in the snowpack. However, SWE measurements are expensive and not continuously accessible in real-time; thus, the Machine learning models aid us in finding an alternative approach to predicting SWE from more widely available and continuous data.



We show the method's general applicability through extensive data of 234779 snow depth-density-SWE records from 2878 non-uniformly distributed sites across Canada. These data cover almost four decades of snowfall. First, the project focuses on predicting SWE using the LSTM model as a base model. Later I used a custom transformer model to predict SWE. In this implementation, we are modeling simple time series data; hence, I choose LSTM and Transformer.

1.2. Motivation

Snow water equivalent (SWE) determines the amount of water available in the snow. However, measuring water retained in an inch of snow is difficult to estimate since the temperature of the air controls the amount. For example, one inch of rain can produce 2 to 50 inches or more of snow, depending on the air temperature. And also, Different storms bring different types of snow that can hold different amounts of water. Throughout winter, other storms bring different types of snow, so snow depth does not translate directly to the amount of water held in snow. Because of this variability, SWE helps to understand how much water the snow contains.



The Transformer widely used in GPT-3, BERT, and Codex is creating a new wave in NLP, time series prediction, etc. The "Attention is all you need." introduces a transformer as an alternate and more efficient model for language translation tasks. The advantages include transformers being parallelizable and requiring less time to train. The Author also states that transformers' performance improved over the best results. The paper centers around the attention mechanism allowing models to attain information from input and output without regard to the distance between input and output sequence. Thus, my scope includes using a transformer and also modifying some parts of the transformer to make the model suitable for the SWE prediction task.

On the other hand, LSTM is a special type of RNN which was introduced to mitigate the vanishing or exploding gradient problems. LSTM's ability to retain long term information makes them suitable for time series forecasting. Thus, in my project the LSTM model is a base model to check the performance of a Transformer.

2. Data

Havard Dataverse is the data source of my project. The paper "Using an ensemble of artificial neural networks to convert snow depth to snow water equivalent over Canada. " uses the same data. This dataset contains most of Canada's snow measurement and meteorological data. The information includes 2685 other stations or geographical locations and has sixteen features, such as SWE(Snow Water Equivalent), Snow depth, Density, Day without snow, etc. The details of the data are below:

Predictand	Snow Water Equivalent
Predictor	Snow depth, Snow density, Elevation, Latitude, Longitude, Day of the year, Day without snow, Number of frost-defrost, Accum pos degree, Average age SC, Number of layers, Accum solid precipitation, Accum solid precipitation in last 10 days, Total precipitation in last 10 days, Avg temperature last 6 days

	SWE	Den	SD	Elev	Lat	Long	Day of year	days without snow	number frost-defrost	accum pos degrees	average age SC	number layer	accum solid precip	accum solid precip in last 10 days	total precip last 10 days	average temp last 6 days
0	176.0	217.284	81.0	730.0	60.58	-129.18	209	194.0	27.0	240.650236	108.729412	186.0	207.549863	5.404499	5.545973	-5.175614
1	112.0	224.000	50.0	700.0	55.30	-123.13	122	113.0	42.0	497.870618	34.347938	94.0	131.999957	1.589837	1.671648	-3.368384
2	31.0	193.750	16.0	122.0	45.25	-76.25	170	164.0	35.0	822.255478	43.855456	148.0	168.140066	33.889766	39.922995	-5.420656
3	1534.0	427.298	359.0	1340.0	50.15	-123.43	224	184.0	78.0	300.042397	87.843641	97.0	1480.637285	112.914425	151.454664	-0.532150
4	117.0	300.000	39.0	168.0	47.15	-67.25	227	211.0	44.0	684.101630	87.965087	161.0	350.782801	0.507681	1.699459	3.038674
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
77382	254.0	362.857	70.0	152.0	48.37	-68.47	210	193.0	24.0	499.712647	86.437214	166.0	372.623211	2.898494	6.883417	-2.427680
77383	46.0	139.394	33.0	265.0	44.63	-78.83	136	134.0	25.0	641.947880	34.382868	122.0	79.903544	14.675475	15.242455	-14.685496
77384	56.0	254.545	22.0	290.0	43.98	-81.43	213	199.0	37.0	770.777052	74.341494	169.0	286.034687	4.295342	24.652693	-1.325837
77385	183.0	225.926	81.0	518.0	52.15	-68.78	179	159.0	19.0	265.993418	72.789019	162.0	243.670871	11.968676	12.125827	-21.351218
77386	62.0	344.444	18.0	76.0	45.95	-67.33	211	196.0	43.0	811.786131	70.905378	158.0	287.718139	13.123810	41.837195	1.815218

77133 rows × 16 columns

Training Data:

Input Data Size: (77133, 10, 15), Output Data Size: (77133, 10, 1)

Validation Data:

Input Data Size: (76827, 10, 15), Output Data Size: (76827, 10, 1)

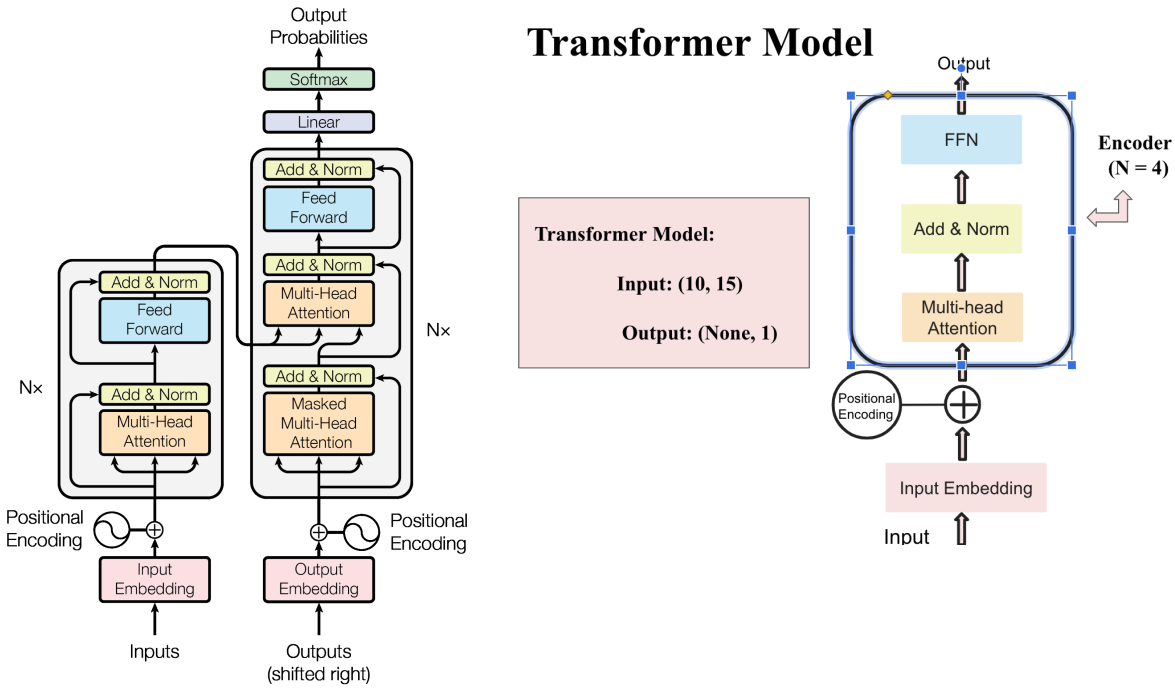
Test Data:

Input Data Size- (76899, 10, 15), Output DataSize: (76899, 10, 1)

3. Method

3.1. Transformer:

The Transformer is composed of an encoder and a decoder. The encoder processes each item in the input sequence and compiles the information it captures into a context vector. Then, based on the context, the decoder produces the output sequence. Both the Encoder and Decoder consist of Embedding layers and Attention layers. The Embedding layer converts the input into embedding vectors used by the attention layer. Attention allows the model to focus on the relevant parts of the input sequence as needed.

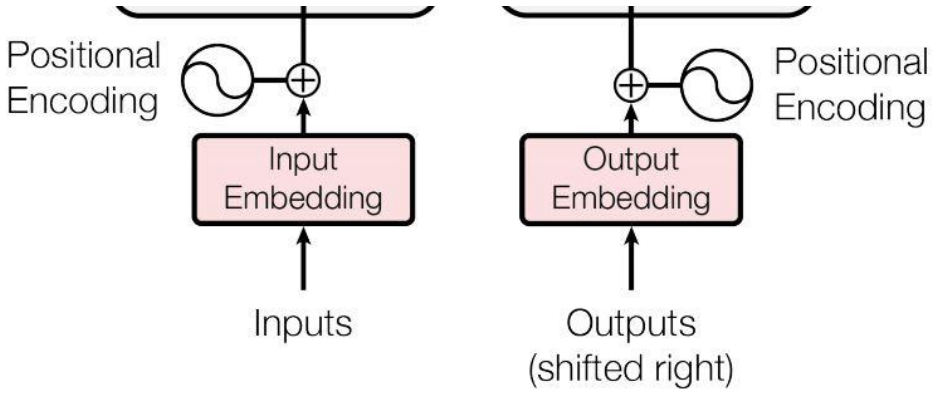


Custom Transformer: Transformer architecture is constructed based on the "Attention is all I need." However, the original model is for natural language processing tasks. Thus, some unnecessary parts of the architecture are removed or replaced for the time series prediction problem. For example, the embedding layer is a simple dense layer since the input values are already numeric. I removed the decoder because the problem is a simple time series prediction.

Embedding Layers: The NLP tasks require input vectors of continuous values; thus, plain text strings need vector transformation. For example, words or phrases from a vocabulary are mapped to a corresponding vector of real numbers using word Embedding. The vector representation has two essential advantages: Dimensionality Reduction, which leads to more efficient representation, and Contextual Similarity, which is a more expressive representation. Word Embedding creates a vector representation with a much lower dimensional space called Word Vectors. These vectors are used for semantic parsing to extract information such as the Contextual similarity of words. For example, tigers, horses, and elephants are similar since they all are animals, but we do not expect them to have proximity to dishwashers, washing machines, etc. Hence, the word vectors maintain these similarities, so words that regularly occur nearby in the text will also be nearby in vector space.

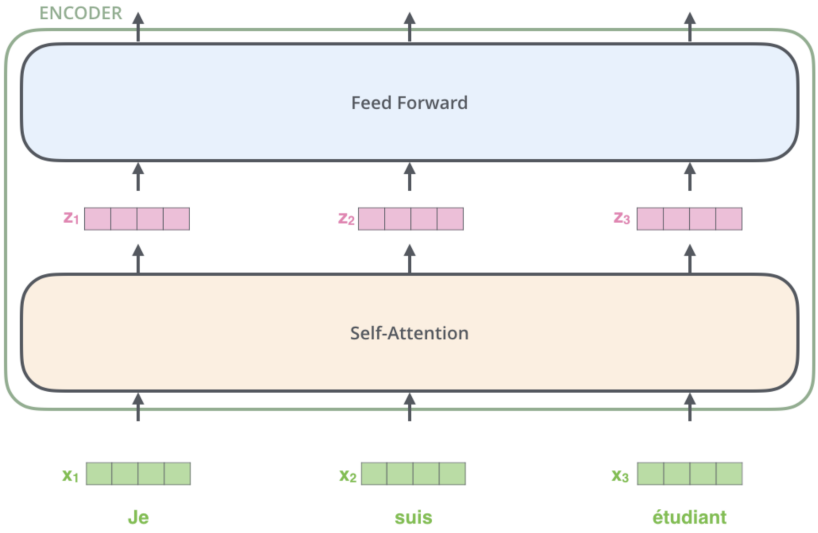
In this implementation, I use one dense layer as an embedding layer to do the dimension conversion for the next layer. Since the inputs are numeric, thus, the transformer model does not require the Keras.Embedding layer in this case.

Positional encoding: The model's attention layers perceive input as a set of vectors with no order. Thus, "Let's eat, grandma" is the same as "Let's eat grandma," Thus, the lack of recurrent or convolution layers needs some way to identify word order. Without the context of order in the input sequence, it will lead to the wrong information or no information from the input. Thus, Positional Encoding is required; The author uses a set of sines and cosines at different frequencies (across the sequence). The positional encoding ensures that nearby elements will have similar encodings.



The custom transformer model uses the same positional encoding to get the context.

Encoder: The encoder mainly consists of N number of identical layers. In this attention paper, N = 6. Each layer has two sub-layer: a multi-head self-attention layer and a position-wise fully connected feed-forward network. The sub-layers contain residual connections and normalization layers. The residual connection in the sub-layer provides a direct path for the gradient and ensures that the attention layer updates vectors. The ResNet uses the same concept to address the gradient degradation problem in deep neural networks. Therefore, the residual connection provides better gradient flow, easy-to-optimization, and less training error in ResNet. In addition, layer normalization maintains a suitable scale for the output.

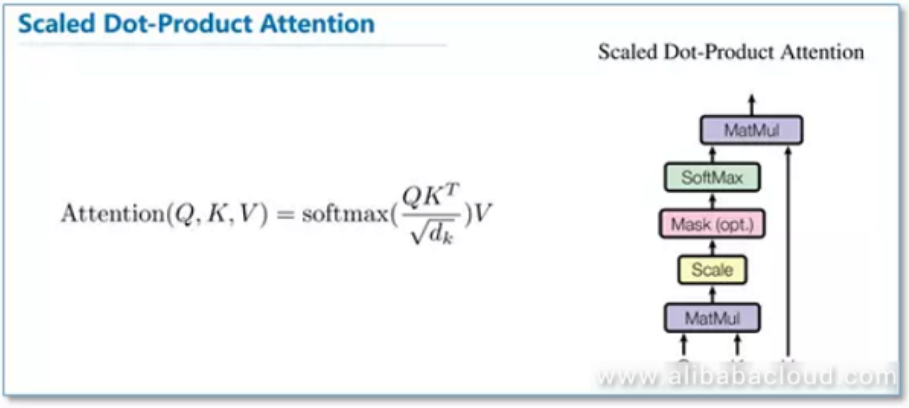


The custom transformer model uses N = 4 of Encoder layers. The number of the layer is a hyperparameter. Thus, we can adjust the number of layers accordingly.

Decoder: The decoder in this paper also consists of six identical layers. Each layer has two sub-layer: a masked multi-head self-attention layer and a position-wise fully connected feed-forward network. The sub-layers contain residual connections and normalization layers. The masking in the attention layer ensures that the output sequence at position p depends only on the output sequence at positions less than p.

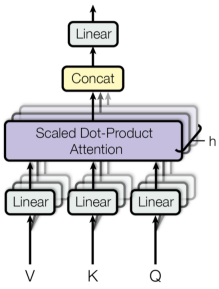
This implementation does not require the decoder part since the task at our hand is a simple time series prediction task. In the future, we can add the decoder to analyze the input and output relationship with time.

Attention: The attention layer consists of a Multihead attention block which consists of self-attention layers that maps a query and set of key-value pairs to an output. Each attention function is a "Scaled Dot-Product Attention" in this paper. The input consists of queries and key-value pairs. The author suspects that large values of dimension of the key will lead the dot products to grow large, thus pushing the softmax function into regions with minimal gradients. Hence, to counteract this effect, scaling is done.



The custom transformer model uses the same Scaled Dot-Product Attention to calculate the attention score.

Multi-Head Attention: The attention mechanism is context based on their relativeness. Thus, instead of performing a single attention function, they found it more beneficial to perform multiple attention functions in parallel. The Multi-head attention allows the model to jointly attend to information from different representation subspaces at various positions. The author used h = 8 parallel attention layers or heads. For each of these, they use dimension = 64. Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality.

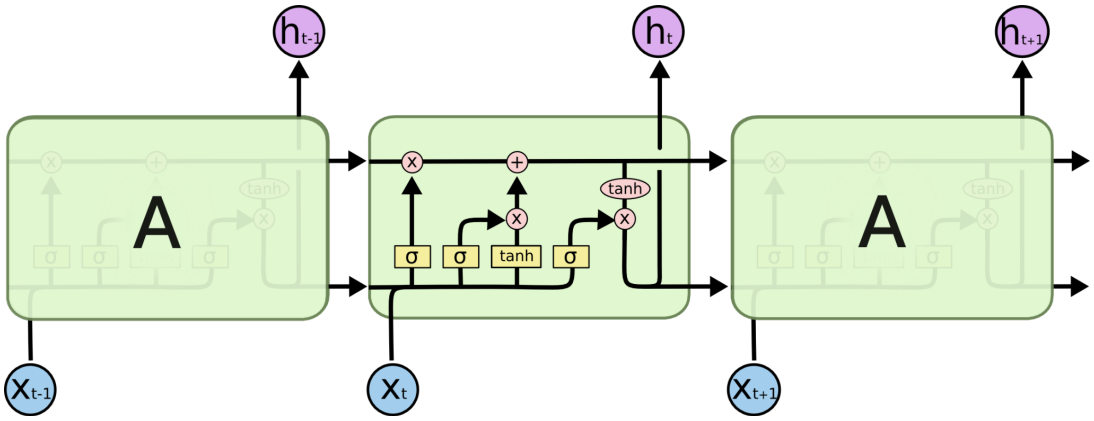


The custom transformer model uses N = 4 of Encoder layers, hence it has

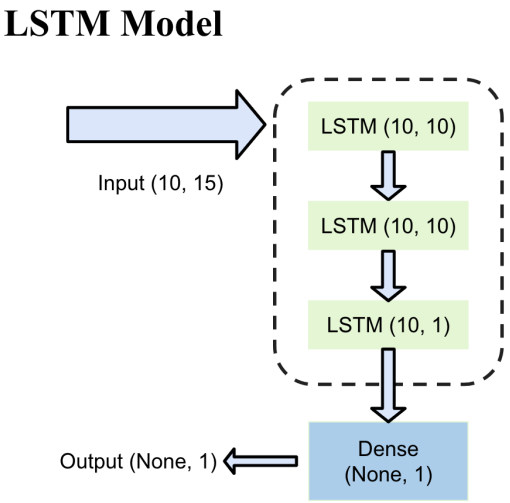
N = 4 attention layer.

3.2. LSTM:

RNNs capability to connect previous information to the present task, like - time series forecasting, makes them suitable for many similar tasks. But sometimes, these tasks require current context and earlier information that RNNs cannot retain. The major problem of RNN is its need for more ability to maintain long-term details. Long Short-Term Memory networks, LSTMs, are a special kind of RNN introduced to mitigate the vanishing and exploding gradient problem of RNN and can learn long-term dependencies. Hence, LSTM's capability to connect previous information to the present task, like - time series forecasting, makes them suitable for many similar tasks.



I used a simple model with three LSTM layers and one dense layer. The input is a time series, and the output is SWE values. Input to the model includes features such as - snow density, snow depth, etc in ten-time stamps. This implementation is simple time series forecasting using the LSTM model. My model contains three LSTM layers with 10 neurons and a dense layer.



4. Results & Discussion

4.1 Transformer Performance

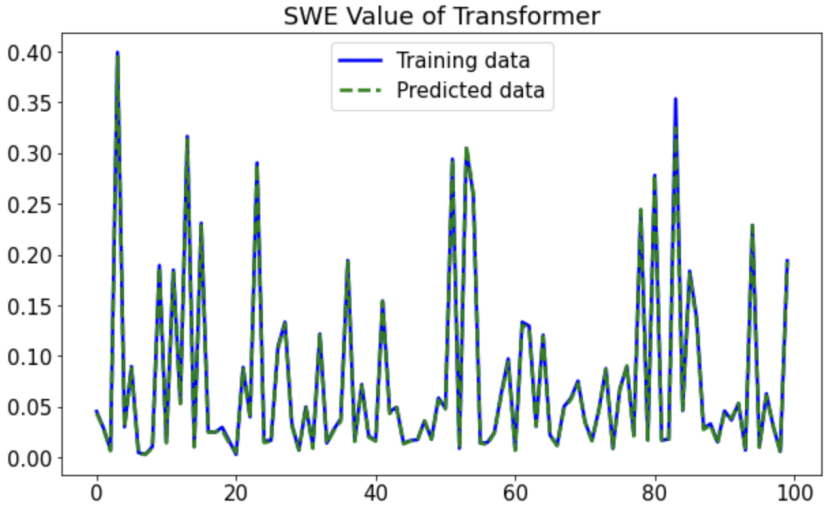


Figure - 4.11: Transformer performance of training data

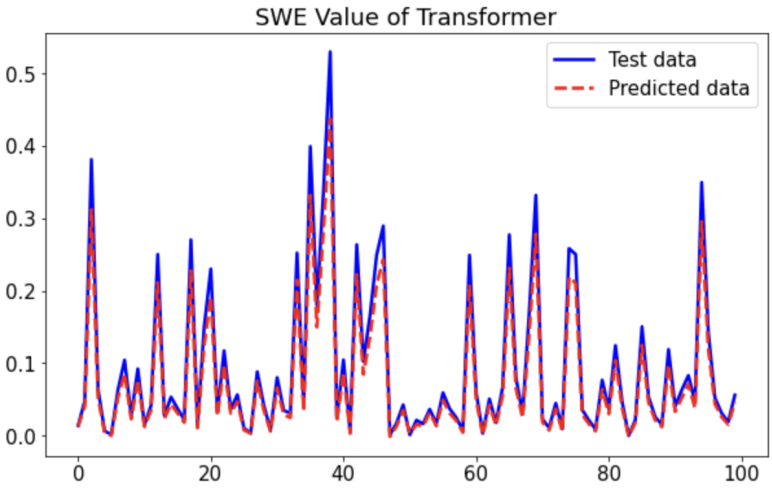


Figure - 4.12: Transformer performance of test data

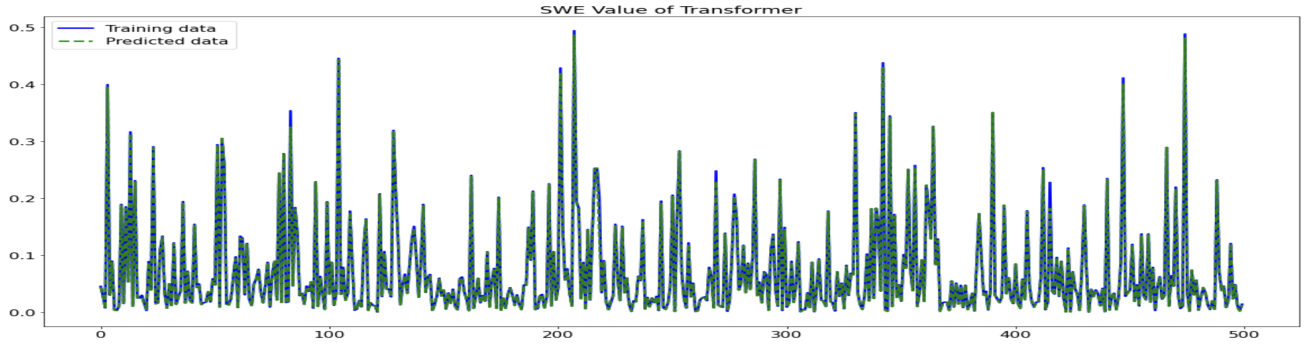


Figure - 4.13: Overall Transformer performance

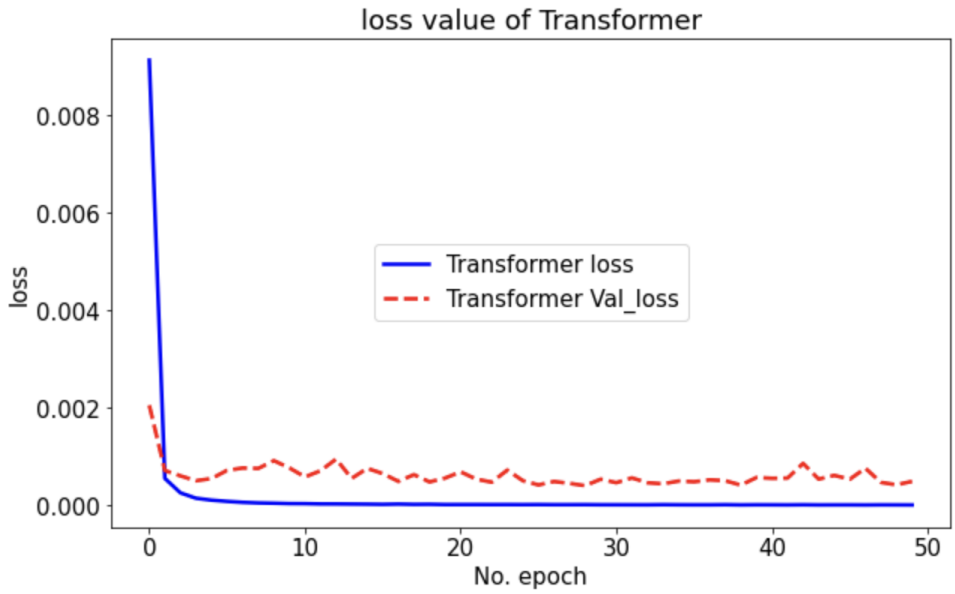


Figure - 4.14: Loss value of Transformer model

Observations:

No. of Neurons	No of Attention layers	Epoch	Learning rate	loss	Val loss	Training time
10	4	50	0.01	6.0696e-06	4.8880e-04	2239
10	5	50	0.01	8.3850e-06	4.8760e-06	1802
10	15	50	0.01	1.2096e-05	3.7678e-06	1560

A transformer with four attention layers gives the best performance. However, when I increased the attention layer, it did not increase efficiency. Initially, I used some part of the validation data set; thus, every time, the validation loss was less than the training loss. Therefore, I suspected that my model was overfitting. However, upon using the whole validation set, I discovered that

the transformer was not overfitting. I got a loss around 6.0696e-06 and a validation loss of 4.8880e-04.

4.2 LSTM Performance

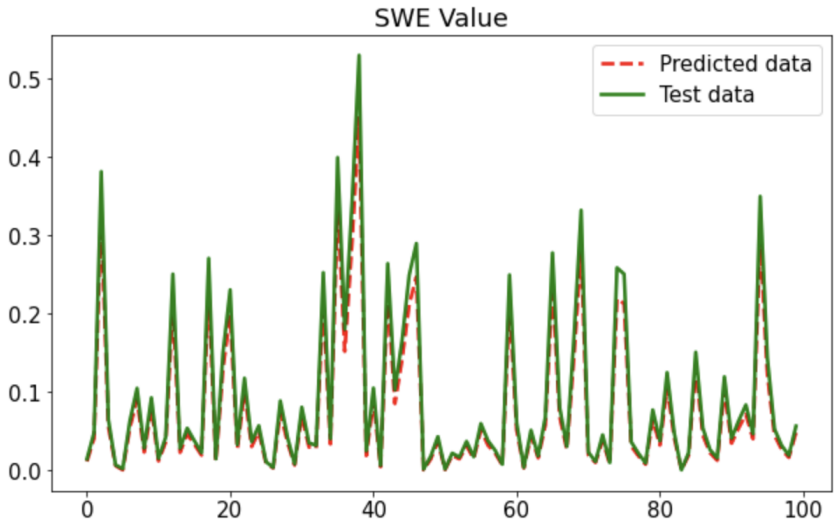


Figure - 4.21: LSTM performance of training data

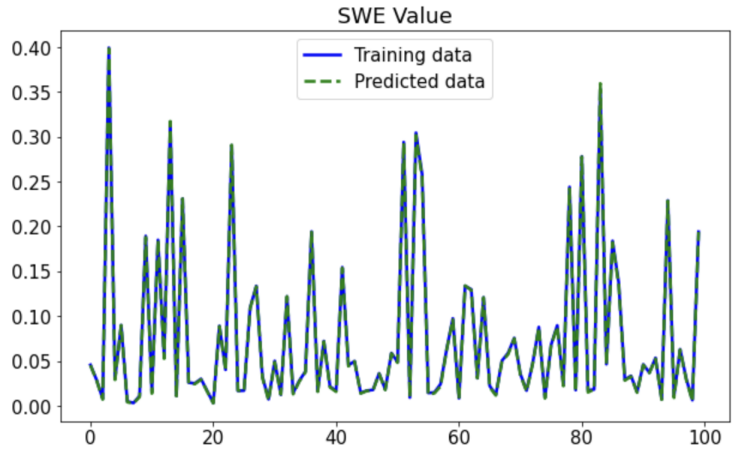


Figure - 4.22: LSTM performance of test data

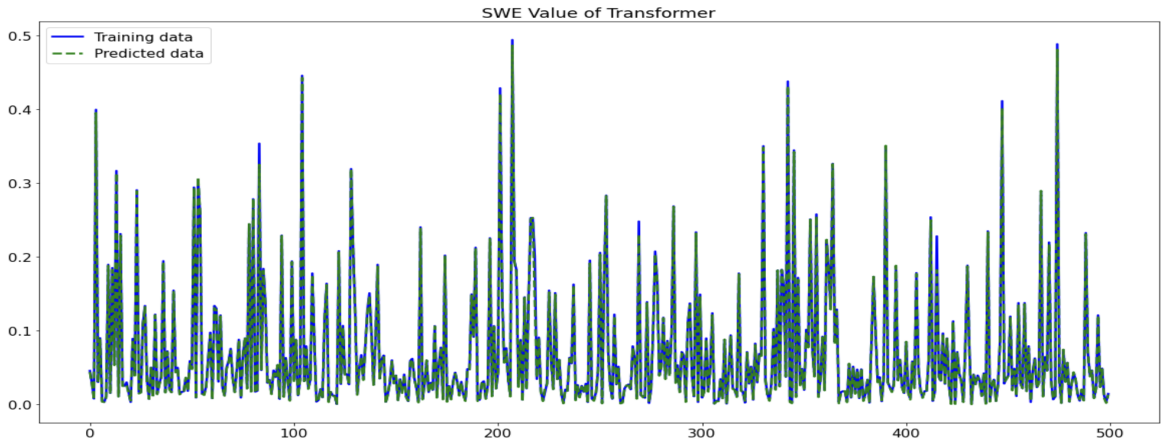


Figure - 4.23: Overall LSTM performance of training data

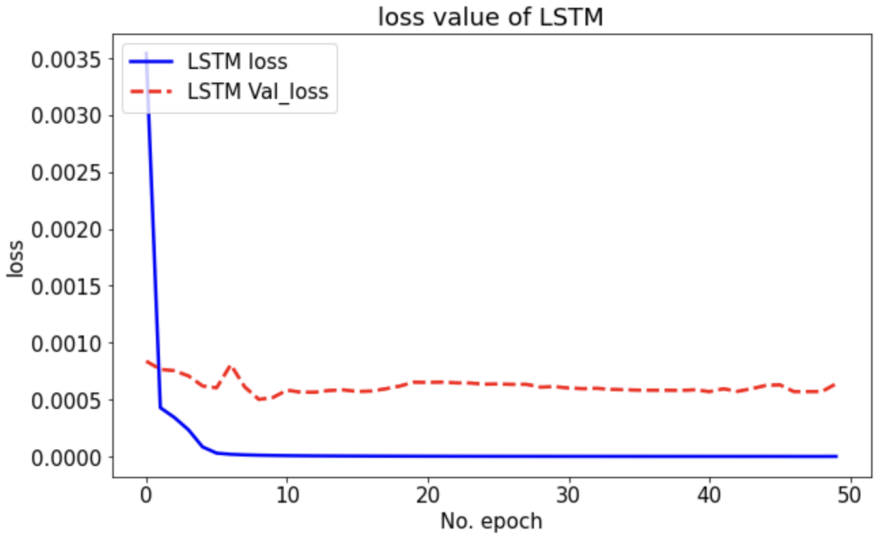


Figure - 4.24: Loss value of LSTM model

Observations:

No. of Neurons	Epoch	Learning rate	loss	Val loss	Training time	
10	50	0.01	1.7187e-06	6.3943e-04	2739	
30	20	0.01	4.4523e-06	5.7003e-06	915 sec	
64	20	0.01	0.0131	0.0132	915 secs	

I tried multiple LSTM models with different numbers of layers and numbers of neurons. I found that simple LSTM with ten neurons and three layers gives a generalized and low-value loss function. However, when I increased the number of neurons to sixty-four, the loss value was 0.01. And also, the underlying model was not generalized. Figure- 4.25 shows that the predicted value is constant.

LSTM with ten neurons is a suitable choice for predicting the SWE. The model is simple and generalized, thus, giving us a context of how the time series prediction works for this data set.

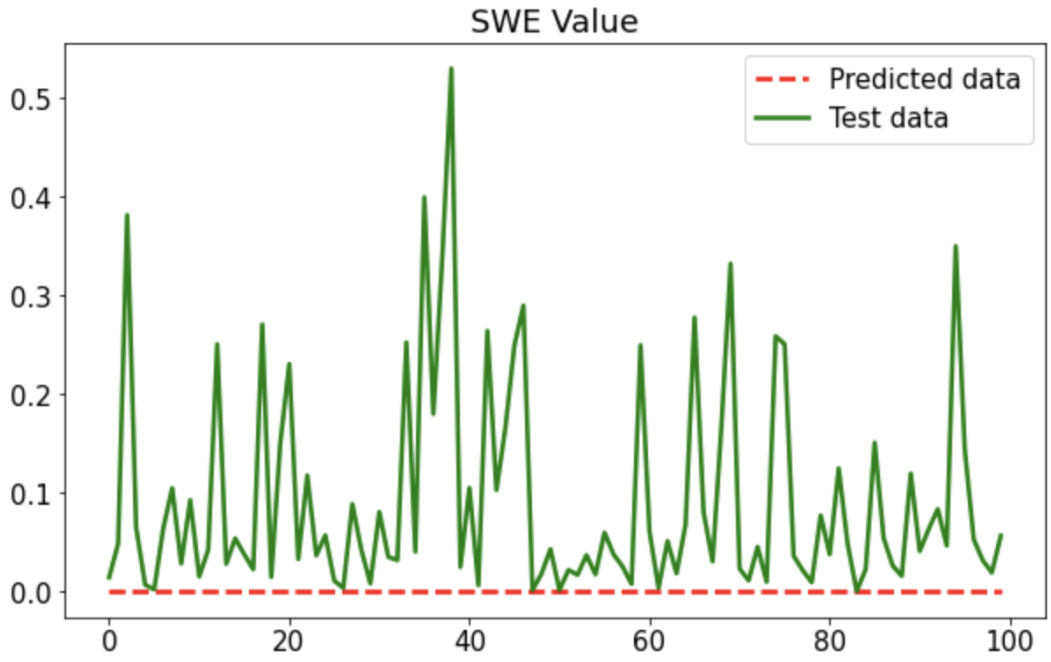


Figure - 4.25: Performance of LSTM with 64 neurons

5. Conclusion

Model	Epoch	Learning rate	loss	Val loss	Training time
LSTM	50	0.01	1.7187e-06	6.3943e-04	41 minutes
Transformer	50	0.01	5.0212e-06	1.9381e-06	23 minutes

In the implementation of the LSTM, the loss = 1.7187e-06 with val_loss = 6.3943e-04. The model captured the overall nonlinear relationship between the predictors and predictand. The model's loss and deviation are slightly lower than the transformer model. In the implementation of the transformer, the loss = 6.0696e-06 with val_loss = 4.8880e-04. The model captured the overall nonlinear relationship between the predictors and predictand. The model's loss and deviation are slightly higher than the transformer model.Due to the parallel nature of the transformer, the transformer takes less time than LSTM to train.

The relatively lower efficiency of the transformer model is because I changed the overall model. The decoder, in general, has two multi-head attention layers. Thus, they can extract context from the output-to-output and input-to-output relationship. My future scope is to explore the decoder part and incorporate two more attention mechanisms to increase the efficiency of the current approach. Nevertheless, the transformer is still a good model for time series prediction.

6. Code Availability

Link - https://github.com/tenworden/e4000_project_tw2834_SWE

The code is available on a private repository since I am not authorized to share the data. I will, however, have a separate public repository without the data.

7. References

1. https://hess.copernicus.org/preprints/hess-2020-566/hess-2020-566.pdf
2. https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/T46ANR
3. https://www.climatehubs.usda.gov/hubs/northwest/topic/snow-water-equivalent-swe-its-importance-northwest#:~:text=Snow%20water%20equivalent%20(SWE)%20determines,broad%20impacts%20on%20water%20resources.
4. https://arxiv.org/abs/1706.03762
5. http://colah.github.io/posts/2015-08-Understanding-LSTMs/
6. http://jalammar.github.io/illustrated-transformer/
7. https://www.tensorflow.org/text/tutorials/transformer#the_embedding_and_positional_encoding_layer
8. https://github.com/ukairia777/tensorflow-transformer/blob/main/Transformer_Korean_Chatbot.ipynb