ECSE552 Homework 4

Group 1

Romain Bazin
McGill ID: 261087142
romain.bazin@mail.mcgill.ca

Yuang Hou
McGill ID: 260847786
yuang.hou@mail.mcgill.ca

***Abstract*—In this work, we propose TimeFormer, a transformer-based model for weather time series forecasting, that outperforms two classic approaches to weather time series forecasting.**

***Index Terms*—Deep learning, Time series forecasting, Transformer**

I. Introduction

Time series forecasting is a critical task in many domains such as weather prediction, finance, and business data analysis. Quick and accurate forecasting allows decision makers in various fields to allocate resources and manage risks more effectively. In the literature, popular deep learning architectures utilized for time series forecasting tasks include convolutional neural networks (CNN) [1], and several forms of recurrent neural networks such as Gated Recurrent Unit (GRU) [2]–[4], Long Short-Term Memory (LSTM) [5], and transformer [6], [7]. Hybrid models that leverage the strengths of several architectures have also achieved great performance [8].

In this work, we designed a deep learning based time series forecasting model that took 14 input features from four past time steps to predict four labels (atmospheric pressure, air temperature, relative humidity, and wind velocity) for the following time step. Data distribution over these four features are displayed in Figure 1. Our proposed solution is a straightforward application of the *transformer* architecture [9] to weather time series forecasting. More precisely, it consists of one *embedding layer*, one *transformer encoder* block and one *linear decoder*. To put this model's performance in terms of prediction accuracy into context, we have trained two baseline models whose hyperparameters aligned with our own model's. The results are clear, **our model outperforms the two baseline models by multiple orders of magnitude**, with the same fixed number of hidden layers and fixed window size of past observations.

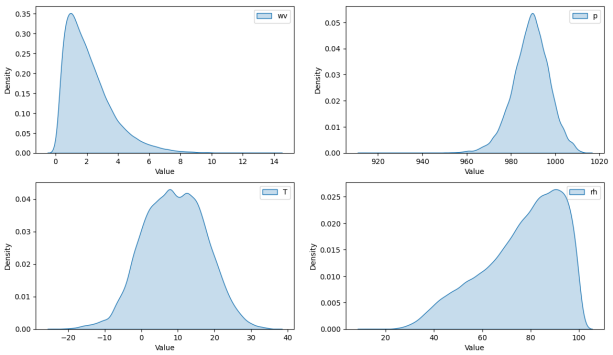


Fig. 1: Data distribution over four features. From left to right and up to down they are : wind velocity, pressure, temperature, relative humidity.

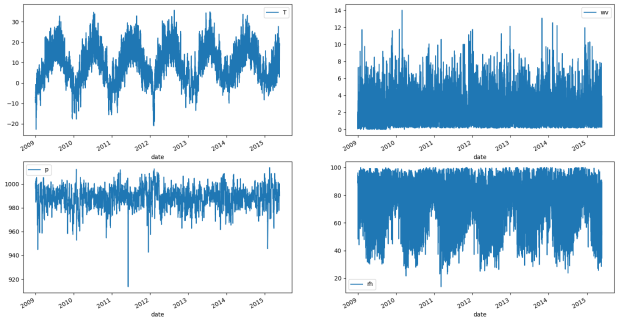


Fig. 2: Evolution of the four labels over time. From left to right and up to down they are : temperature, wind velocity, pressure, relative humidity.

Experimentes were performed on the weather time series data set from the Max Planck Institute for Biogeochemistry [10]. This data set was prepared by Francois Chollet, and it originally included 14 weather metrics in ten-minute intervals from 2009 to 2016. Figure 2 illustrates the evolution of each of the labels through time. Temporally, we only used these time series data in one-hour intervals which struck a reasonable balance between the data set size and resolution. With a sliding window of five time steps, 56068 and 14019 samples were extracted from the data set to form respectively the train set and the test set. We also provide a study of the error propagation for the baseline models, as well as for the transformer model.

All models were implemented with PyTorch [11] and trained on the free tier of Google Colab, which offers an access to a T4 GPU. The code for this project is available here.

II. Methodology

All models had the same task of predicting four continuous labels : the atmospheric pressure **p**, the air temperature **T**, the relative humidity **rh**, and the wind velocity **wv** for a given time

step t. All models took historical inputs from time $t-k$ to time $t-1$, where t was the time step for which the labels had to be predicted and k was the number of historical time steps that served as the inputs of the models. The historical values given as an input were from 14 different features (comprising the 4 labels used as features when they are from past time steps, plus 10 other atmospheric features).

To compare the performance of the different models, we fixed two major hyperparameters : the window size of historical values k to use as an input and the number of hidden layers of the models h. **In all experiments, k was fixed to 4 and h to 3**.

As described earlier, all models solve a **multi-label regression task**, where the number of continuous values to predict is 4. The loss used is the very classic *mean squared error* (MSE), heavily used in all types of regression tasks. Both baseline models were trained using *stochastic gradient descent* (SGD) and a batch size of 1000. All three hidden layers of the baseline models have 4 nodes. As for our proposed transformer model, which we couldn't resist to name *TimeFormer*, it conists of three hidden layers : a fully connected *linear embedding layer* that maps the into features to a 250 dimensional space, the embeddings are then enhanced by a *positional embedding* [9], then comes a *transformer encoder* block of dimension 250, with 10 heads, which is finally connected to a fully connected *linear decoder* that generates the target labels.

A. Fully connected model (Baseline 1)

A fully connected neural network is a type of artificial neural network that consists of a series of dense layers where any neuron in a given layer is connected to all neurons in the adjacent layers, forming a dense network of connections. Fully connected neural networks have been used in time series forecasting tasks in the literatrue [12]. In this work, features from each past time step was a vector of length 14, and four of these feature vectors were concatenated into a vector of length 56 to form the input feature vector for the fully connected network. There were three hidden layers, sharing the same dimension of 4. The model was trained to minimize the mean squared error between its four outputs $\hat{\mathbf{y}}$ and their ground truth values $\mathbf{y}$ at time t.

B. LSTM model (Baseline 2)

The Long Short Term Memory network (LSTM) is a type of recurrent neural network architecture designed to solve the vanishing/exploding gradient problem to effectively capture long-term dependencies within time series data [13]. The application of LSTM on time series tasks is heavily researched in the literature [5], [14], [15]. The LSTM architecture in this work consisted of three LSTM hidden layers where each hidden state contained 4 nodes. Sharing the same dimension as the hidden states was the output at time $t-1$ which was passed into a dense layer that mapped the last LSTM layer's output at time $t-1$ into $\hat{\mathbf{y}}$, the output of the model. At the training stage, input features from the past 4 time steps were fed into the model sequentially from time $t-k$ to time $t-1$, and the training objective was to minimize MSE between $\hat{\mathbf{y}}$ and $\mathbf{y}$.

C. TimeFormer (our proposed solution)

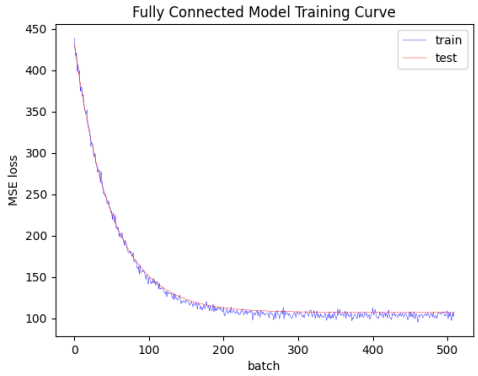
First democratized in the field of *Natural Language Processing* (NLP), the transformer architecture [9] has rapidly been adopted in many other fields of deep learning, such as object detection [16], video restoration [17] or automatic speech recognition [18].

Multiple reasons favored the massive adoption of this architecture. First, transformers are much better than LSTM and RNN models at capturing long relationships between two vectors distant in a sequence, because all vectors are compared to all other vectors of the sequence at the same time, rather than through a long chain of reccurent units like in LSTM or RNN models. This is why transformers represent the foundations of all *Large Language Models* (LLMS) [19] [20], as words in sentences often have distant, yet important, relationships. Second, the training of transformers is much more parallelizable than the one for LSTM networks, for which the units have to process vectors in a sequential order, whereas the *attention heads* in the transformer architecture can be trained all at the same time. This is propably what allowed the third benefit of using transformers, which is that the architecture is much more scalable, in the sense of it's faster to train due to the easy parallelization of the training, but also because large transformer models are very *data efficient*, which means that they achieve better results than other models without training them as much [21].

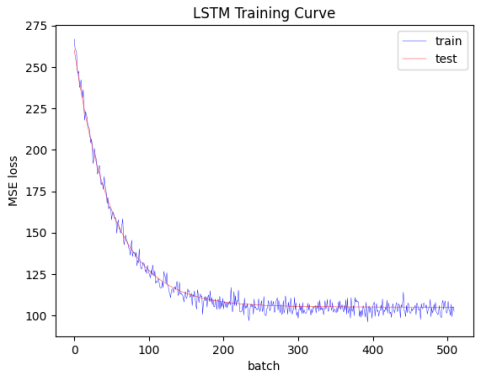
For all these reasons, we believe that the transformer architecture is very appropriate to the task of weather time series forecasting, as most of the features seem to have time dependencies (see figure 2) which transformers could exploit. This is why we propose a simple and small architecture, with at its core, a *transformer encoder layer*. To train this model, the features vectors of time steps between $t-4$ and $t-1$ are fed to an linear embedding layer, the embeddings of dimension 250 are retrieved and completed with a simple *cosine positional embedding* (like in the original transformer paper [9]), the completed embeddings are then fed to the transformer encoder layer which should learn the temporal relationships between features vectors, and hopefully generate embeddings meaningful enough for the linear decoder layer to generate appropriate label values at time step t. Then, the MSE loss computes the L_2 distance between the generated labels and the groundtruth. This loss is used to update the weights of the network with the *Adam* optimizer [22].

III. Experimental results

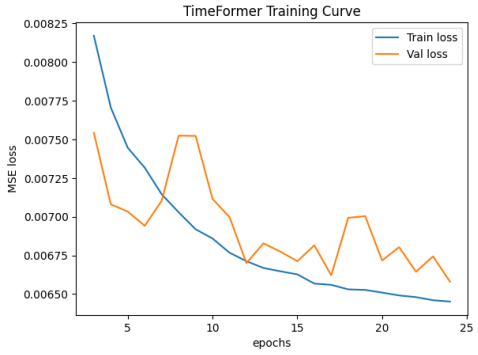
With a fixed learning rate of 0.01, both baseline models were trained with stochastic gradient descent where the batch size was 1000 samples. Both models converged within less than five epoches, or around 250 iterations. TimeFormer was trained with a learning rate of 0.001, on 25 epochs and with a batch of size 64.



(a)



(b)



(c)

Fig. 3: (a) Fully Connected network (b) LSTM network (c) TimeFormer

The metric used to compare the models was the MSE. It was used globally on all metrics during a **10-fold cross validation**, and for the evaluation on the test set. Additionnally, we show the performance of our model compared to the baselines, for each feature, using a per-feature MSE. Finally, we propose an evaluation of the *error propogation* of all models on the test set.

Due to time constraints, we did not perform any type of hyperparameter tunning for any of the models. Still, we observed that the distribution of our labels was showing a small number of outliers 1, that we thought were deteriorating the performance of our model, maybe by skewing the scaling of the features, so we tried to remove them in hope that it would boost performance. However, removing those outliers just decreased the performance of all models on the test set, so we continued our experiments without touching the outliers.

A. MSE over all features (10-fold cross validation)

For every model, test MSE on unseen data for all 10 folds (cross validation) are displayed below to illustrate this model's overall prediction accuracy and stability.

Model	MSE	std
Fully connected	104.85	11.10
LSTM	104.06	9.97
TimeFormer (ours)	8.39	3.28

TABLE I: Performance comparison of all models using MSE during 10-fold cross validation, using **unscaled features**.

Model	MSE	std
Fully connected	104.06	9.97
LSTM	105.07	10.97
TimeFormer (ours)	0.00647	0.00024

TABLE II: Performance comparison of all models using MSE during 10-fold cross validation, using **scaled features**.

We observe that our model very clearly outperforms the two other models by a large margin, on the unscaled data (table I), but even more so on the scaled data (table II). The reason for that might be that it's easier for the model to learn the time dependencies between vectors of features when the features are scaled, this has probably something to do with the position embedding, which is a very small number added summed to each vector, to encode its position in the input sequence of vectors.

B. MSE per-feature (test set)

We also evaluated the performance of all models at predicting each label on the test, by computing the per-feature MSE. The following tables summarize the results.

Model	**p**	**T**	**rh**	**wv**
Fully connected	68.28	72.70	275.80	2.34
LSTM	68.42	72.70	275.79	2.34
TimeFormer (ours)	6.25	1.95	17.25	210

TABLE III: Performance comparison with per-feature MSE, using **unscaled features**.

Model	**p**	**T**	**rh**	**wv**
Fully connected	88.62	42.16	259.63	2.74
LSTM	88.39	42.30	160.10	2.73
TimeFormer (ours)	0.0004	0.0008	0.008	108

TABLE IV: Performance comparison with per-feature MSE, using **scaled features**.

Once again, we observe that **our model clearly outperforms the other baseline models** by multiple orders of magnituted for most labels, and that the scaling undeniably boosts the performance of our model, but not the performance of the two others.

Interestingly, in both tables it appears that our model **fails** at correctly predicting the wind velocity (wv), when the two other models predict this exact label **exceptionnally well**. This behavior **did not** appear on the validation set, which led us to think that the distribution of the wind velocity from the training set was not representative of the one from the test set, we hypothesized that it could come from the previously highlighted outliers. However, as stated earlier, we didn't see any (stastically relevant) boost in performance for this specific label when removing the outliers.

C. Error propagation

In this section, the trained models were deployed to predict values from time t to $t+4$ with input data from time $t-4$ to $t-1$. Predictions on four features are graphically illustrated below. While the models iteratively took its own output $\hat{\mathbf{y}}$ to make further prediction, the models' output dimension of 4 did not align with thier input dimension of 14. As a result, we had to fill the 10 missing features at each input feature vector with their ground truth values.

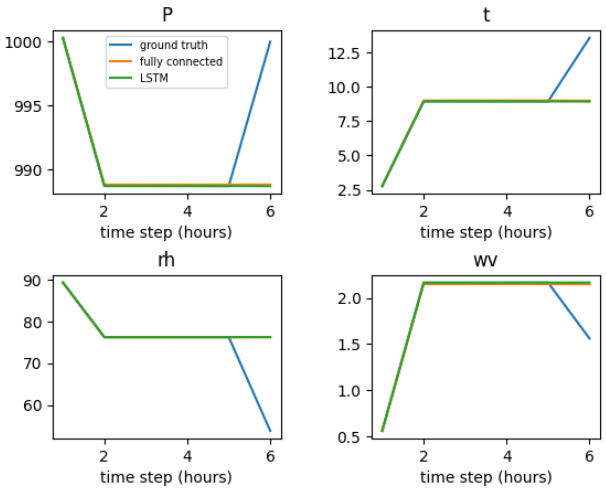


Fig. 4: Illustration of the error propagation for the baseline models.

As an illustration of what error propagation is, you can see on figure 4 how the baseline models globally fail at predicting the correct label values when using their own predictions. For the sake of the illustration, t was taken as 56010 and k was 4. Ground truth data for all time steps shown on the figure was unseen during training.

We quantified the error propagation, and how it worsens when increasing how much ahead the model has to predict by computing the average error over the whole test set for different predictions ahead. For example, for a value of *steap ahead* equal to 2, the model would first predict the labels for the time step ahead of one, then reuse its own prediction to predict the labels for the step ahead of 2. The error would then only be computed between the predicteed label ahead of 2 and the corresponding groundtruth. The results are showed on figure 5. As expected, the error propagation greatly increases by a *compound effect*, only the wind velocity label doesn't follow this rule, which corroborates the idea that the model was failing at predicting this value from the start.

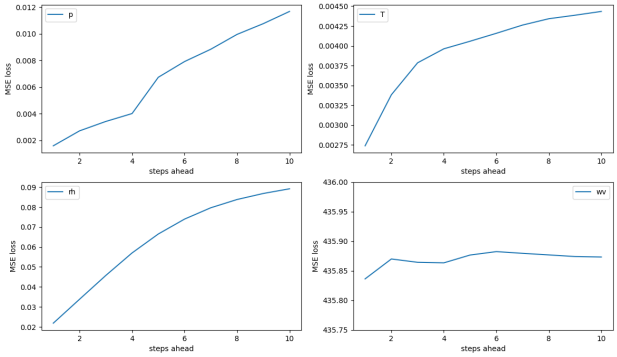


Fig. 5: Error propagation for TimeFormer, horizontal axis is the number of steps ahead for the evaluated prediction, y axis is the MSE loss.

IV. Conclusion

During this project, we compared different approaches for weather time series forecasting. Our proposed model, TimeFormer, based on the transformer architecture, **greatly outperformed the baseline models**, with the same number of hidden layers and input features. However, we noted that it particularly fails at predicting the "wind velocity" label, for unknown reasons.

For future work, we think that increasing further the performance of our model could be easily achieved by a number of ways : increasing the number of transformer encoder layers, changing the linear decoder layer by multiple transformer decoder layers.

REFERENCES

[1] Wensi Tang, Guodong Long, Lu Liu, Tianyi Zhou, Jing Jiang, and Michael Blumenstein. Rethinking 1d-cnn for time series classification: A stronger baseline. *arXiv preprint arXiv:2002.10061*, pages 1–7, 2020.

[2] Peter T Yamak, Li Yujian, and Pius K Gadosey. A comparison between arima, lstm, and gru for time series forecasting. In *Proceedings of the 2019 2nd International Conference on Algorithms, Computing and Artificial Intelligence*, pages 49–55, 2019.

[3] Xu Zhang, Furao Shen, Jinxi Zhao, and GuoHai Yang. Time series forecasting using gru neural network with multi-lag after decomposition. In *Neural Information Processing: 24th International Conference, ICONIP 2017, Guangzhou, China, November 14–18, 2017, Proceedings, Part V 24*, pages 523–532. Springer, 2017.

[4] Wendong Zheng and Gang Chen. An accurate gru-based power time-series prediction approach with selective state updating and stochastic optimization. *IEEE Transactions on Cybernetics*, 52(12):13902–13914, 2021.

[5] Vinay Kumar Reddy Chimmula and Lei Zhang. Time series forecasting of covid-19 transmission in canada using lstm networks. *Chaos, Solitons & Fractals*, 135:109864, 2020.

[6] Qingsong Wen, Tian Zhou, Chaoli Zhang, Weiqi Chen, Ziqing Ma, Junchi Yan, and Liang Sun. Transformers in time series: A survey. *arXiv preprint arXiv:2202.07125*, 2022.

[7] George Zerveas, Srideepika Jayaraman, Dhaval Patel, Anuradha Bhamidipaty, and Carsten Eickhoff. A transformer-based framework for multivariate time series representation learning. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 2114–2124, 2021.

[8] Fazle Karim, Somshubra Majumdar, Houshang Darabi, and Shun Chen. Lstm fully convolutional networks for time series classification. *IEEE access*, 6:1662–1669, 2017.

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[10] Francois Chollet. Max planck institute for biogeochemistry weather time-series dataset.

[11] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[12] Anastasia Borovykh, Cornelis W Oosterlee, and Sander M Bohté. Generalization in fully-connected neural networks for time series forecasting. *Journal of Computational Science*, 36:101020, 2019.

[13] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[14] Steven Elsworth and Stefan Güttel. Time series forecasting using lstm networks: A symbolic approach. *arXiv preprint arXiv:2003.05672*, 2020.

[15] Alaa Sagheer and Mostafa Kotb. Time series forecasting of petroleum production using deep lstm recurrent networks. *Neurocomputing*, 323:203–213, 2019.

[16] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[17] Rui Liu, Hanming Deng, Yangyi Huang, Xiaoyu Shi, Lewei Lu, Wenxiu Sun, Xiaogang Wang, Jifeng Dai, and Hongsheng Li. Fuseformer: Fusing fine-grained information in transformers for video inpainting, 2021.

[18] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision, 2022.

[19] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[20] Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, YaGuang Li, Hongrae Lee, Huaixiu Steven Zheng, Amin Ghafouri, Marcelo Menegali, Yanping Huang, Maxim Krikun, Dmitry Lepikhin, James Qin, Dehao Chen, Yuanzhong Xu, Zhifeng Chen, Adam Roberts, Maarten Bosma, Vincent Zhao, Yanqi Zhou, Chung-Ching Chang, Igor Krivokon, Will Rusch, Marc Pickett, Pranesh Srinivasan, Laichee Man, Kathleen Meier-Hellstern, Meredith Ringel Morris, Tulsee Doshi, Renelito Delos Santos, Toju Duke, Johnny Soraker, Ben Zevenbergen, Vinodkumar Prabhakaran, Mark Diaz, Ben Hutchinson, Kristen Olson, Alejandra Molina, Erin Hoffman-John, Josh Lee, Lora Aroyo, Ravi Rajakumar, Alena Butryna, Matthew Lamm, Viktoriya Kuzmina, Joe Fenton, Aaron Cohen, Rachel Bernstein, Ray Kurzweil, Blaise Aguera-Arcas, Claire Cui, Marian Croak, Ed Chi, and Quoc Le. Lamda: Language models for dialog applications, 2022.

[21] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020.

[22] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.