TRANSFORMER IN TIME SERIES FORECASTING

Hengda Shi
Department of Computer Science
University of California, Los Angeles
Los Angeles, CA 90095, USA
hengda.shi@cs.ucla.edu

ABSTRACT

Time series forecasting has been an important research in academia and real world applications. While many machine learning models assume data points are independent from each other, times series data has strong temporal dependency among different data points. Time series forecasting models are required to be capable of capturing the temporal dynamics hidden in the data. Recent studies have shown the potential of Transformer in modeling long sequence data in natural language processing. In this capstone project report, we discuss the popular Transformer architecture and its application in time series data in detail. [1].

1 INTRODUCTION

With the recent success of deep learning in many scientific fields, many different domains that heavily rely on time series data, such as earthquake prediction (Johnson et al., 2021), weather forecasting (Ren et al., 2021), and influenza-like illness (ILI) forecasting (Wu et al., 2020), have been benefited from the active development of deep learning. In these problems, the historical data can be leveraged to make a future prediction. These predictions can be crucial in various scenarios, such as real-time disease monitoring and real-time disaster warning.

A variety of methods have been proposed to make prediction on time series data. Traditional statistical methods including auto-regression (AR), autoregressive integrated moving average (ARIMA) have proven to be effective in time series forecasting. On the other hand, modern machine learning methods learn hidden temporal information purely based on the data. The capability of machine learning models has grown as the data and computing power increase.

Since the deep learning revolution in 2012, deep learning has gained tremendous interest in the academia and industry. Many different domains, such as image classification (Krizhevsky et al., 2012), natural language processing (Devlin et al., 2019), reinforcement learning (Silver et al., 2016), and graph representation learning (Kipf & Welling, 2017), have been making progress with the use of deep learning algorithms. Deep neural networks are capable of learning complex data representations by building multi-layer networks with various techniques and objectives. It mitigates the need of extensive feature engineering to some extent.

However, vanilla neural networks are limited in capturing the underlying data relationship because it treats each feature in the data point equally. Convolutional neural networks and recurrent neural networks which was developed for image data and sequence data respectively can be applied to time series data and work better than the vanilla neural networks. With the recent advance of self-attention mechanism in natural language processing (Vaswani et al., 2017), transformer-based architecture has been gaining popularity in even many other domains, such as computer vision (Dosovitskiy et al., 2020) and graph representation learning (Shi et al., 2021).

The problem definition will be provided in the preliminary section. In the prior work section, we will discuss the common deep learning approaches before transformer-based architecuture. In the transformer section, we will detailedly discuss the transformer architecture and transformer variant that is designed specifically for time series data. In the experiment section, dataset and experimental

[1] The code can be found at https://github.com/hengdashi/capstone_project

setup will be introduced and further discussed. Finally, we conclude our discussion on transformer-based architecture in time series data in the conclusion.

2 PRELIMINARY

Time series forecasting makes prediction on a target $\boldsymbol{y}_i^t$ for a set of observations $\boldsymbol{y}_i$ at time t. The time series models take the form:

$$\hat{\boldsymbol{y}}_i^{t+1:t+m} = f(\boldsymbol{y}_i^{t-k:t}, \boldsymbol{x}_i^{t-k:t}), \tag{1}$$

where $\hat{\boldsymbol{y}}_i^{t+1:t+m}$ is the model prediction. When $m > 1$, the time series prediction is said to be a multi-step prediction. $\boldsymbol{x}_i^{t-k:t}$ could be empty if we aim to forecast all the features in the observation. $f(\cdot)$ is the prediction function learned by the model. When $\boldsymbol{y}_i^t \in \mathbb{R}^{L_y}$ and $L_y > 1$, the prediction is said to be multivariate as the model aims to predict more than one variable.

3 PRIOR WORK

Before the transformer-based models, two neural network architectures are popular in time series data — Convolutional Neural Networks and Recurrent Neural Networks. They are still popular choices in time series prediction. In the following sections, we will introduce the overall concepts of each architecture and discuss their advantages and disadvantages.

3.1 CONVOLUTIONAL NEURAL NETWORKS

Convolutional neural networks (CNNs) LeCun & Bengio (1998) extract spatial relationship in the input data by connecting a subset of the neurons in the previous layer. The subset of neurons are typically a sequence of features in the input data and could be a rolling window of univariate or multivariate time series data. CNNs can therefore capture information of the rolling window, or receptive field, and make forecasts based on the learned information.

To specifically dealing with time series data, dilated causal convolution van den Oord et al. (2016) is typically employed. Unlike traditional convolution which takes output from its surrounding neighbor for a given node, causal convolutions ensure the convolutional filter can only take input from the past. For time series data, it is important to make sure the convolutional filter does not pick up information that is supposed to be in the future. A network comparison is shown in Figure 1. For long sequence time series prediction, dilation is often needed to lower the resolution of the distant observations. The dilated causal convolution layer takes in the form:

$$\boldsymbol{h}_t^{l+1} = \sigma\Big((\boldsymbol{W} * \boldsymbol{h})(l, t)\Big), \tag{2}$$

$$(\boldsymbol{W} * \boldsymbol{h})(l, t, d_l) = \sum_{\tau=0}^{\lfloor k/d_l \rfloor} \boldsymbol{W}(l, \tau)\boldsymbol{h}_{t-d_l\tau}^l, \tag{3}$$

where $\boldsymbol{h}_t^l$ is the output of a hidden layer l at time t, $\sigma(\cdot)$ is the activation function, $*$ is the convolution operator, and $\boldsymbol{W}(l, \tau)$ is the weight of the filter at layer l.

However, CNNs are also limited by its receptive field and the fixed filters. The fixed filters will be applied to any window of time series data, and is therefore incapable of capturing seasonal changes hidden in the data. It is also limited by the rolling window because it is only capable of capturing the information within the receptive field.

3.2 RECURRENT NEURAL NETWORKS

Recurrent neural networks (RNNs) Rumelhart et al. (1988), on the other hand, are designed for sequence modeling, and were commonly used in various natural language processing tasks. Given the similar nature of text sequence and time series data, RNNs seem to be natural to apply to time series related tasks. RNNs contain connections between nodes and stores a summary of the past

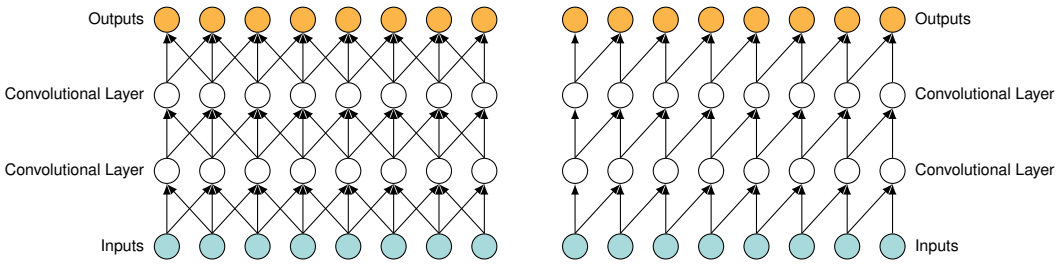

Figure 1: Image CNN (Left) v.s. Causal CNN (Right)

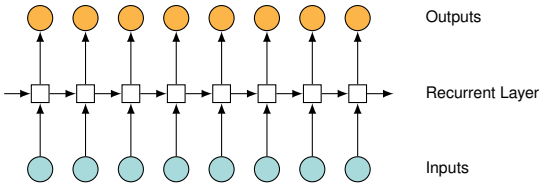

Figure 2: Basic RNN architecture

information as a memory state. RNNs can learn more past information because they do not have restrictions on the rolling window. A typical RNN architecture is shown in Figure 2.

For a typical RNN architecture, the hidden state $\boldsymbol{a}^t$ and the ouput $\boldsymbol{y}^t$ at time step t is in the form:

$$\boldsymbol{a}^t = \sigma_1(\boldsymbol{W}_a \boldsymbol{a}^{t-1} + \boldsymbol{W}_x \boldsymbol{x}^t + \boldsymbol{b}_a) \tag{4}$$

$$\boldsymbol{y}^y = \sigma_2(\boldsymbol{W}_y \boldsymbol{a}^t + \boldsymbol{b}_y) \tag{5}$$

where σ_1, σ_2 are the activation functions, and $\boldsymbol{W}_a, \boldsymbol{W}_x, \boldsymbol{W}_y, \boldsymbol{b}_a, \boldsymbol{b}_y$ are weights shared temporally.

However, exploding and vanishing gradients problem presist in RNNs and its variant architecture. This problem limits the ability of RNN to store long term memory in the memory states. With the addition of forget state in RNN variant like LSTM Hochreiter & Schmidhuber (1997), the exploding and vanishing gradients problem can be mitigated, but it is far from completely solving the problem. Another issue with RNNs is that the dependency between nodes makes it harder to parallelize as the computation of the current node can only be started when the previous node finishes its computation.

4 TRANSFORMER

With the attention machanism Bahdanau et al. (2015) born to help the encoder-decoder RNN network preserving long term memory, the use of attention in various networks have become popular in recent years. In particular, transformer model Vaswani et al. (2017) that was introduced in 2017 is entirely built upon the self-attention machanism without the use of any recurrent architecture. In the following subsections, we will discuss the Transformer architecture in detail.

4.1 POSITIONAL ENCODING

In the case of natural language processing, position and order of the words are important in interpreting the language. In recurrent neural networks, the order of the word has been naturally taken into consideration as the words are processed in a sequential manner. However, by removing the recurrent-like architecture, Transformer itself does not have the ability to capture the position and order of each word in the sentence. Therefore, it becomes essential to preprocess the input so that it incorporates positional information when entering the Transformer model.

Even though one can incorporate the position index into the data, it corrupts the data when the sequence is extremely long as the data at the last position will be added with a large number. The author instead proposed to encode the position of the data in the sequence using sine and cosine

functions in the form:

$$\begin{cases} \boldsymbol{p}^t_{2i} & = \sin(w_i \cdot t) \\ \boldsymbol{p}^t_{2i+1} & = \cos(w_i \cdot t) \end{cases} \tag{6}$$

where

$$w_i = \frac{1}{10000^{2i/d_{\text{model}}}} \tag{7}$$

t is the position and i is the feature dimension. The benefit of such positional encoding is obvious. For the data within the sequence length, each data will be equiped with a different encoding provided by the $\sin$ and $\cos$ functions.

Temporal Encoding: In time series data, positional encoding seems to be useful in capturing the temporal relationship. However, the relative positional encoding within the sequence length limits the Transformer to only capture information in the range similar to CNNs. Zhou et al. (2021) proposed a method of encoding the temporal information in the input. Specifically, it leverages the time information in the data to create the encoding. For example, for a data point with month, day, and hour information, the temporal embedding will be a sum of the month embedding, day embedding, and hour embedding. Each embedding is a positional encoding with fixed sequence size. The sequence size corresponds to the finest granularity of each type of embedding. The finest granularity of the month embedding will likely be 12 months. The finest granularity of the day embedding will likely be 30 days. This temporal encoding could be useful in incorporating additional temporal information in the input data. However, it requires every data entry is associated with a time stamp, which might sometimes not be the case.

4.2 Multi-Head Self-Attention

The self attention machanism is the major component in the Transformer. The attention takes the encoded input representation in the form of (query, key, value) and performs the scaled dot-product as a variant of the dot-product attention Luong et al. (2015):

$$\mathcal{A}(\boldsymbol{W}, \boldsymbol{K}, \boldsymbol{V}) = \text{Softmax}(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d}})\boldsymbol{V} \tag{8}$$

where $\boldsymbol{Q} \in \mathbb{R}^{L_Q \times d}$, $\boldsymbol{K} \in \mathbb{R}^{L_K \times d}$, $\boldsymbol{V} \in \mathbb{R}^{L_V \times d}$ and d is the input dimension.

To further increase the expressive power of the model, multiple attention heads are used to run the self-attention machanism in parallel. The author claims that the multi-head attention will allow the model to learn attention from different position of the input representation. Specifically, the multi-head attention takes in the form:

$$\text{MultiHead}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = [\text{head}_1 : \ldots : \text{head}_h]\boldsymbol{W}^O \tag{9}$$

where

$$\text{head}_i = \mathcal{A}(\boldsymbol{Q}\boldsymbol{W}^Q_i, \boldsymbol{K}\boldsymbol{W}^K_i, \boldsymbol{V}\boldsymbol{W}^V_i) \tag{10}$$

h is the number of heads, and $\boldsymbol{W}^Q_i, \boldsymbol{W}^K_i, \boldsymbol{W}^V_i, \boldsymbol{W}^O$ are learnable parameters.

***ProbSparse* Self-Attention**: A major problem in Transformer is the quadratic computation time complexity and the $O(L_Q L_K)$ memory comsumption. The self-attention machanism has potential sparsity and could be utilized to reduce the computational cost. Previous studies (Child et al., 2019; Li et al., 2019; Beltagy et al., 2020) have shown multiple attempts in reducing the computational complexity by using heuristic methods to search for the sparsity of the self-attention. In Zhou et al. (2021), they propose to extract the top-u queries under a sparsity measurement in the form:

$$M(\boldsymbol{q}_i, \boldsymbol{K}) = \ln \sum_{j=1}^{L_K} e^{\frac{\boldsymbol{q}_i \boldsymbol{k}_j^\top}{\sqrt{d}}} - \frac{1}{L_K} \sum_{j=1}^{L_K} \frac{\boldsymbol{q}_i \boldsymbol{k}_j^\top}{\sqrt{d}}, \tag{11}$$

The u is set to be $u = c \cdot \ln L_Q$ where c is a constant sampling factor. By only utilizing the top-u queries in the dot product computation, the self-attention time complexity can be reduced from quadratic time complexity to $O(\ln L_Q)$. The memory usage will also drop from $O(L_K L_Q)$ to $O(L_K \ln L_Q)$. However, the $u = c \cdot \ln L_Q$ is not always smaller than L_Q when $c \geq 3$. Therefore, when L_Q is small and $c \geq 3$, it is equivalent to the traditional self-attention mechanism.

4.3 ENCODER AND DECODER

The encoder and docoder are built upon the self-attention layer and standard techniques like layer normalization, dropout, and residual connection. Each encoder layer typically consists of a **multi-head attention** layer, a **dropout** layer, a **residual** connection, and a **layernorm** layer. The input of the encoder at sequence t is shaped to $\boldsymbol{X}_{en}^{t} \in \mathbb{R}^{L_x \times d_{\text{model}}}$ after the positional encoding. L_x is the size of the rolling window up to the prediction point. In the decoder, the input is feeded as the concatenation of the start token and the prediction sequence:

$$\boldsymbol{X}_{de}^{t} = \text{Concat}(\boldsymbol{X}_{\text{token}}^{t}, \boldsymbol{X}_{\boldsymbol{O}}^{t}) \in \mathbb{R}^{(L_{\text{token}}+L_t) \times d_{\text{model}}} \tag{12}$$

It is commonly found that utilizing a earlier slide of the sequence just before the prediction sequence would help the decoder in capturing the sequence relationship. $\boldsymbol{X}_{\text{token}}^{t}$ would be the last L_{token} of the $\boldsymbol{X}_{en}^{t}$ sequence.

In later literature (Devlin et al., 2019; Radford et al., 2019), an important observation is that good performance can be obtained by using only the encoder or the decoder architecture. It is also observed in the experiment section that using encoder-decoder or decoder only architecture can lead to similar performance.

Self-Attention Distilling: Inspired by dilated convolution, Zhou et al. (2021) chose to distill their self-attention layer by performing 1D convolution and max pooling on the output. Specifically, the output from j-th layer into $(j+1)$-th layer is in the form:

$$\boldsymbol{X}_{j+1}^{t} = \text{MaxPool}\Big(\text{ELU}\big(\text{Conv1d}([\boldsymbol{X}_{j}^{t}]_{AB})\big)\Big) \tag{13}$$

where $[\cdot]_{AB}$ is the output of the multi-head ProbSparse self-attention and $\text{Conv1d}(\cdot)$ is an 1-D convolutional filter with kernel width 3. The max pooling layer will downsample the output to half of the original size and thus reduce the memory usage.

5 EXPERIMENT

5.1 DATASET

In the experiment, a rather noisy dataset is chosen to test the performance of the Transformer as well established datasets have been already benchmarked in many papers.

The dataset is provided in the kaggle competition "$\text{Jane Street Market Prediction}$". A total of 130 anonymized features representing real stock market data is provided in the dataset. Each data entry represents a trading opportunity, and the model shall predict whether to attempt a trade or not to maximize the return. In addition to the trading features, 5 returns over different time horizons are provided to guide the model in maximizing the return. However, only one of the return specified by the organizer will be used in evaluating the overall model performance. Each training data is also associated with a weight to be used in evaluating the overall return. Specifically, the evaluation is measured on a utility score with the following form:

$$p_i = \sum_j (\text{weight}_{i,j} \cdot \text{resp}_{i,j} \cdot \text{action}_{i,j}), \tag{14}$$

$$t = \frac{\sum p_i}{\sqrt{\sum p_i^2}} \cdot \sqrt{\frac{250}{|i|}}, \tag{15}$$

$$u = \min(\max(t, 0), 6) \sum p_i \tag{16}$$

where p_i is the overall weighted sum of the specified return 'resp' for each date i, $|i|$ is the number of days available in the test dataset, and the utility score u is the overall return $\sum p_i$ multiplying a weighted number clipped between the range $[0, 6]$. A relative date where the trading opportunity is available is given for each training entry with a total of 500 days available in the training dataset, and a relative ordering of the training entry is also given.

5.2 DATA ANALYSIS

A notable characteristic of the dataset is that the time interval between adjacent trading opportunities is dynamic. Unlike many traditional time series dataset that is recorded on a fixed time interval, the trading dataset provided by jane street is recorded anytime during the trading period. The trading frequency for a given date can be as high as ten thousand times and as low as 22 times.

This characteristic becomes the main obstacle in training time series model like Transformer. In image or text data, the distance between two features in the input is mostly fixed. For image data, the neighbor pixel has a fixed length of one to a center pixel. In text data, each pair of adjacent words has a fixed length of one as well. Such information is implicited provided to the model as the input data being computed because adjacent data must exhibit similar characteristic.

However, in the trading prediction, only trading opportunities are given, and the adjacent set of features can likely be two distant moments that have a time interval up to a few hours or as short as one second.

5.3 EXPERIMENT SETUP

The experiment is conducted on one 8GB RTX 2070 Super Graphics Card and AMD 3600 CPU. Many different setups have been experimented. Different number of encoder and decoder layers in the range of $[0, 4]$, prediction window size L_t {3, 12, 24, 48, 96, 128}, different depth of the model d_{model}, feed forward network d_{ff}, the number of heads in the self attention, and the attention mechanism [prob, full] have been experimented to tune the performance of the model. Two different loss functions are experimented in the experiment. One metric is the binary cross entropy loss which the problem is framed to be a binary classification problem. The action will be taken if the resp is greater than zero meaning that under a certain time horizon, the return is positive. Another metric is the Smooth L1 loss or the Huber loss. The problem in this case is framed to be a regression problem where the gold is to predict the return as close as possible with the groundtruth resp. Other traditional hyperparameters including learning rate, dropout rate, batch size, and early stopping patience are also experimented.

5.4 RESULTS AND ANALYSIS

The main difference in performance comes from the loss function. Framing the problem to be a binary classification problem generally will perform poorly compared to framing the it to be a regression problem. Choosing a large d_{model} and d_{ff} can cause learning problem as well. For example, the original architecture chose d_{model} and d_{ff} to be 512 and 2048 respectively. The number of features in the dataset is only 130, and in their dataset, the number of features is only 7. The wide model layer makes the feature spreading in the output and sparser than the original feature space. The convolutional layer with receptive field of 3 can therefore only capture a small area of features and degrade the model performance by a large margin.

The default configuration is shown in Table 1. L_x represents the input sequence length or the rolling window length of the input into the encoder. L_{token} represents the rolling window length of the input to the decoder. L_t is the number of prediction output. When $L_t > 1$, the prediction is a multistep prediction meaning that the model tries to predict multiple future consecutive events. L_e represents the number of the encoder layers, L_d represents the number of the decoder layers, and L_h represents the number of attention heads. lr represents the learning rate, which is commonly seen in deep learning applications.

Under the default hyperparameters setting, the model can achieve a utility score of 2143.54. With only two layers of decoders and no encoders, the model is still able to achieve 2199.20 which is even higher than the naive transformer model. Based on the experiment, the encoder does not seem to contribute much to the final performance, and this result is also constantly observed in other NLP literature. The change in L_{token} does not lead to high performance gain as well. For a decoder only architecture, $L_{\text{token}} = 48$ is sufficient to achieve the best performance in all tested Transformer architecture. *ProbSparse* self-attention does save tremendous amount of memory. Under the same standard setting with default hyperparameters configuration, the batch size can be two times larger when using *ProbSparse* other than the Full attention.

Attention	Loss	L_x	L_{token}	L_t	d_{model}	d_{ff}	dropout	L_e	L_d	L_h	lr
Full	Huber	96	48	1	128	128	0.2	1	1	8	0.001

Table 1: Default Hyperparameters

6 CONCLUSION

In this capstone project report, we discuss different deep learning architecture that could be applied to time series data. We also further investigate the Transformer and its state of the art application in the times series data. In the experiment section, we investigate the usage of Transformer in a particular noisy dataset, and pinpoint some of the drawbacks with the Transformer architecture. It would be interesting to see further improvement in time series forecasting and more mathematically proven methods in dealing with time series data.

REFERENCES

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1409.0473.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv:2004.05150*, 2020.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *URL https://openai.com/blog/sparse-transformers*, 2019.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://www.aclweb.org/anthology/N19-1423.

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, 2020.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8): 1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

Paul A. Johnson, Bertrand Rouet-Leduc, Laura J. Pyrak-Nolte, Gregory C. Beroza, Chris J. Marone, Claudia Hulbert, Addison Howard, Philipp Singer, Dmitry Gordeev, Dimosthenis Karaflos, Corey J. Levinson, Pascal Pfeiffer, Kin Ming Puk, and Walter Reade. Laboratory earthquake forecasting: A machine learning competition. *Proceedings of the National Academy of Sciences*, 118(5), 2021. ISSN 0027-8424. doi: 10.1073/pnas.2011362118. URL https://www.pnas.org/content/118/5/e2011362118.

Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the 5th International Conference on Learning Representations*, ICLR '17, 2017. URL https://openreview.net/forum?id=SJU4ayYgl.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

Yann LeCun and Yoshua Bengio. *Convolutional Networks for Images, Speech, and Time Series*, pp. 255–258. MIT Press, Cambridge, MA, USA, 1998. ISBN 0262511029.

Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/6775a0635c302542da2c32aa19d86be0-Paper.pdf.

Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 1412–1421, Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1166. URL https://www.aclweb.org/anthology/D15-1166.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

Xiaoli Ren, Xiaoyong Li, Kaijun Ren, Junqiang Song, Zichen Xu, Kefeng Deng, and Xiang Wang. Deep learning-based weather prediction: A survey. *Big Data Research*, 23:100178, 2021. ISSN 2214-5796. doi: https://doi.org/10.1016/j.bdr.2020.100178. URL https://www.sciencedirect.com/science/article/pii/S2214579620300460.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. *Learning Representations by Back-Propagating Errors*, pp. 696–699. MIT Press, Cambridge, MA, USA, 1988. ISBN 0262010976.

Yunsheng Shi, Zhengjie Huang, shikun feng, Hui Zhong, Wenjin Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification, 2021. URL https://openreview.net/forum?id=B9t708KMr9d.

David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016. URL http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Neo Wu, Bradley Green, Xue Ben, and Shawn O'Banion. Deep transformer models for time series forecasting: The influenza prevalence case, 2020.

Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *The Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021*, pp. online. AAAI Press, 2021.