Transformer Kalman Networks: Introducing Uncertainty to Transformers

Jeffrey Bao
The University of Texas at Austin
Austin, Texas
jeffreyxbao@gmail.com

ABSTRACT

In Natural Language Processing (NLP), Transformers have largely replaced models based on Recurrent Neural Networks (RNNs), such as Long Short Term Memory Recurrent Neural Networks (LSTMs) and Gated Recurrent Unit Recurrent Neural Networks (GRUs). The Transformer has many inherent benefits due to its use of attention instead of recurrence or convolution, but has seen limited use outside of NLP tasks, such as translation. One reason for this is its inability to analyze uncertainty. The goal of this work is to implement uncertainty estimates in Transformer outputs. This feature is demonstrated by predicting confidence intervals for the next values in a time series.

1 INTRODUCTION

Time series are studied in many different disciplines due to their applicability to a great variety of real world situations. Virtually any process that occurs over time can be modeled by a time series; among the fields that use this are computer vision, quantitative finance, meteorology, and epidemiology. Time series can be used to control robots, develop investment strategies, forecast the weather, predict the spread of disease [4].

The Recurrent Neural Network and its derivatives have dominated analysis of time series, but are not without drawbacks. Because Recurrent Neural Networks are recurrent, they can take inordinately long to train and suffer from vanishing/exploding gradients. While these issues are alleviated in some degree with techniques such as Truncated Backpropagation Through Time [5], the Transformer's inherent design differences eliminate these problems. The Transformer is able to train in a fraction of the time compared to other recurrent architectures [6].

However, the Transformer was not designed with time series in mind, which is evident in many of the design choices proposed by Vashwani et al. For example, Transformers expect a discrete output space (each word or token has a discrete value). The output dimension of a Transformer is equal to the number of possible values in the output space and the Transformer outputs a probability for each value [6]. Unfortunately, time series analysis generally requires a continuous output space, so this strategy will not work well.

In time series analysis, the ability to estimate uncertainty in a continuous output space is very common, such as with the Kalman Filter and the related Recurrent Kalman Network (RKN) [1]. The Transformer Kalman Network (TKN) I propose incorporates elements of the RKN and the Kalman Filter and implements them in conjunction with the Transformer network proposed by Vashwani et al. This implementation can prove useful in situations where there is epistemic uncertainty in the measurements and/or aleatoric uncertainty in the system.

2 DESIGN

2.1 Overview

The goal of this design is to generate a prediction sequence from some input sequence. Like other sequence to sequence networks, the TKN uses an encoder-decoder structure [6].

In the encoding stage, the input is processed. The encoder maps the input sequence $(x_0, ..., x_n)$ to some vector z. This vector is used as an input for all subsequent decoder steps.

The decoding stage predicts the next values in the series. To initialize this prediction, Vashwani et al. uses a start token [6], but since we are not using discrete values, we must instead use the last known value in the series, x_n, and an uncertainty value of 0. The value z, sequence (x_n), and sequence (0) are fed into the decoder, which outputs a prediction $(\hat{y}_0)$ and uncertainty estimate $(\hat{\sigma}_0)$. To predict the next value, the decoder takes in the value z, sequence $(x_n, \hat{y}_0)$, and sequence $(0, \hat{\sigma}_0)$ to output a prediction sequence $(\hat{y}_0, \hat{y}_1)$ and uncertainty estimate sequence $(\hat{\sigma}_0, \hat{\sigma}_1)$. This is repeated until eventually, the value z, sequence $(x_n, \hat{y}_0, ..., \hat{y}_{m-1})$, and sequence $(0, \hat{\sigma}_0, ..., \hat{\sigma}_{m-1})$ are fed into the decoder, which outputs a final prediction sequence $(\hat{y}_0, ..., \hat{y}_m)$ and uncertainty estimate sequence $(\hat{\sigma}_0, ..., \hat{\sigma}_m)$.

2.2 Encoder and Decoder

Figure 1 shows the overall structure of the TKN.

The encoder starts with an input pseudo-embedding and a positional encoding. This output is fed into a stack of N_{enc} identical layers. Each of these sub-layers consists of a multi-head attention layer, feedforward layer (with hidden dimension d_{ff}), as well as residual connections [3]. The sub-layers all use input and output dimension d_{model}.

Similarly, the decoder starts with an output pseudo-embedding and a positional encoding, which feed into a stack of N_{dec} identical layers. These layers are very similar to the layers used in the encoder, except they use a look-ahead mask [1] and use z to generate query and value vectors. After these sub-layers is a feedforward layer (with hidden dimension d_{ff}), which outputs a sequence of prediction vectors and uncertainty estimate vectors.

2.3 Pseudo-Embedding

The main purpose of embedding is to gain semantic information from a token by giving it some vector representation. Unfortunately,

[1] a matrix used during training to prevent the decoder from looking at points in the future

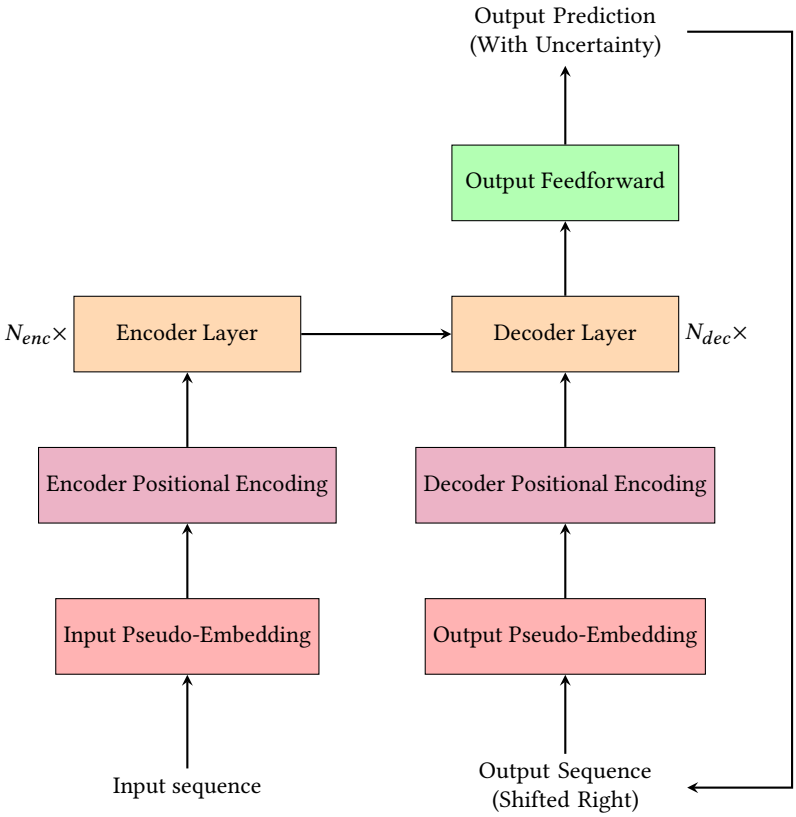

Figure 1: The TKN's architecture

the input and output embeddings used in Transformers will not work for time series because the inputs and outputs are not token indices, as they are in NLP tasks.

Another important function of embedding is determining d_{model}, the input and output dimension of the encoder and decoder sub-layers. In many cases, the dimension of the input data, d_{in} is not sufficient to convey meaningful information between parts of the network. For this reason, it is useful to raise the dimension of the input vector, d_{in}, using a "pseudo-embedding" to some higher dimension d_{model}.

$$v' = Av \tag{1}$$

We define A as a $d_{model} \times d_{in}$ matrix, where $d_{model} > d_{in}$. We feed a sequence of vectors $(v'_0, ..., v'_n)$ into the encoder rather than $(v_0, ..., v_n)$ in order to preserve more information about the sequences between layers of the encoder as well as between the encoder and decoder.

The output pseudo-embedding is identical except that it uses d_{out} as the input dimension rather that d_{in}.

2.4 Positional Encoding

Because the TKN does not use convolution or recurrence, positional encoding provides information about the order of the sequence. The TKN uses a learned positional encoding (also called positional embedding) as it has been shown to perform well in models such as the Bidirectional Encoder Representations from Transformers (BERT) [2].

A positional encoding vector is generated by the *PosEmbedding* function using the positional index i of vector v'. This function is a true embedding layer, which maps each index to some vector. This vector is added to the scaled vector v'. It is important to scale v' by $\sqrt{d_{model}}$ in order to reduce variance in the positional encoding and improve stability during training.

$$PosEncLayer(v', i) = \sqrt{d_{model}}v' + PosEmbedding(i) \tag{2}$$

2.5 Query, Key, and Value

Query, key, and value vectors are used in attention layers to analyze relationships between elements of the input and output sequences, as each point is mapped to a query, key, and value vector. The motivation behind this mechanism is similar that of retrieval, like in search engines. Queries are compared with keys, and this comparison is used to retrieve appropriate values.

For the TKN, query, key, and value vectors are generated using learned mappings from vector space $\mathbb{R}^{d_{model}}$ to $\mathbb{R}^{d_k}$, $\mathbb{R}^{d_k}$, and $\mathbb{R}^{d_v}$ respectively. In this case, $d_k = d_v = d_{model}$ was chosen for simplicity.

The query, key, and value generators for this work are feed-forward neural networks with N_{QKV} layers and ReLU activation functions.

$$(q_i, k_i, v_i) = QKVGen(x_i) \tag{3}$$

Equation 3 shows how query, key, and value vector are generated from each element of sequence $(x_0, ..., x_n)$.

2.6 Attention

Attention is the core of the TKN and is what differentiates it from other architectures. RNNs look at each point in a sequence individually, but attention analyzes the relationships between every permutation of two points in the sequence all at the same time. This dramatically decreases computational time but increases memory used.

First query, key, and value matrices are generated.

$$Q = \begin{bmatrix} q_1 \\ \vdots \\ q_n \end{bmatrix} \tag{4}$$

Equation 4 shows how a Q matrix is generated by concatenating q_i row vectors. K and V matrices are generated identically except using k_i and v_i row vectors respectively.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{5}$$

The TKN uses multi-head attention, as proposed by Vashwani et al [6]. This allows the learning of different features without significantly increasing computational cost.

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O \tag{6}$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{7}$$

Equations 6 and 7 show how multi-head attention is performed by concatenating the results of individual heads. The W matrices are used to project between subspaces. Because we chose $d_k = d_v = d_{model}$, these matrices do not have a significant function. As such, a detailed explanation is beyond the scope of this work. For more information, refer to "Attention Is All You Need" [6].

2.7 Training

For training, the Adam optimizer was chosen in conjunction with a ReduceLROnPlateau scheduler. A masked mean squared error loss function was used so that the uncertainty estimate could optionally be ignored during training.

Initially, the uncertainty estimate should be ignored during training to improve stability. A function $p(\epsilon)$, using epoch ϵ and constant α, is used to determine the probability that a given epoch will include include the uncertainty estimate in training.

$$p(\epsilon) = 1 - e^{-\alpha\epsilon} \tag{8}$$

The target uncertainty values are calculated by calculating the mean squared errors of the point estimates.

Additionally, the TKN cannot be robustly trained using only the traditional method of Transformer training, which involves feeding the ground truth target sequence, $(x_n, y_0, ..., y_m)$, into the decoder. The reason for this is that the traditional training method assumes that the decoder will perfectly predict the next values.

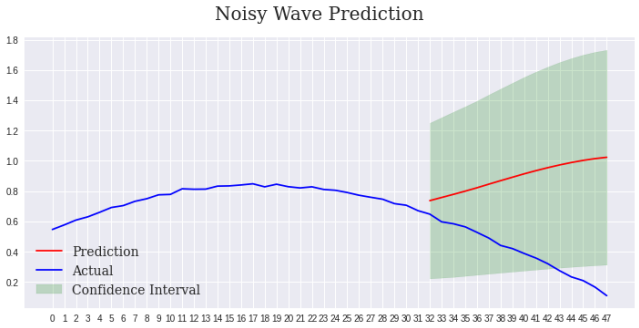

Figure 2: A sample prediction after training traditionally. Noticeable drifting occurs and the prediction accuracy is poor.

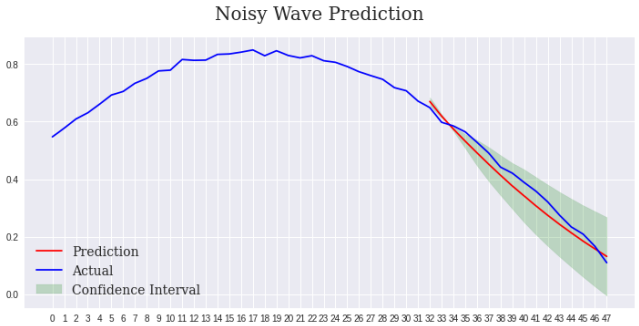

Figure 3: A sample prediction after training traditionally and iteratively. This prediction is much improved compared to training only traditionally.

In real world situations, the ground truth for the output sequence is not known and the prediction sequence is generated iteratively, predicting $\hat{y}_i$ based on previous predictions $(x_n, \hat{y}_0, ..., \hat{y}_{i-1})$. This is usually not an issue with discrete output spaces, but causes problems in continuous output spaces. If the predicted output differs even slightly from the ground truth, the prediction sequence will likely drift away from the ground truth sequence.

To solve this problem, the TKN can first be trained using the traditional Transformer training method for E_t epochs to save time. Then, the TKN is trained for E_i epochs with the decoder output generated iteratively, much like the RNN (and its derivatives). Unlike the RNN, the TKN encoder does not need to be trained iteratively because the same amount of information about the input sequence is always known.

3 RESULTS

To demonstrate the application of uncertainty estimates, the TKN was used to predict the next points in a noisy wave.

Parameter	Definition	Value
n	Input Sequence Length	32
m	Output Sequence Length	16
N_{enc}	Number of Encoder Layers	2
N_{dec}	Number of Decoder Layers	2
h	Number of Attention Heads	8
N_{QKV}	Query/Key/Value Feedforward Layers	2
d_{in}	Input Dimension	1
d_{out}	Output Dimension	2
d_{model}	Encoder/Decoder Hidden Dimension	256
d_{ff}	Feedforward Hidden Dimension	256
P_{drop}	Dropout Probability	0.01
E_t	Traditional Training Epochs	20
E_i	Iterative Training Epochs	40
α	Training Probability Constant	0.01

Figure 4: Network parameters

$$f(t) = 0.1sin(0.005t) + 0.5sin(0.5t) + 0.2sin(0.01t) + 0.01Rand(t) \quad (9)$$

The function $Rand(t)$ maps each value t to some random value from a normal distribution with $\mu = 0$ and $\sigma = 1$.

A sequence of 10240 values was generating using values from $t = 0$ to $t = 1024$ using steps of size 0.1. From the first 80% of the sequence, 1234 random samples were used for training. From the last 20% of the sequence, 271 random samples were used for validation. The model used for this demonstration uses parameters from Figure 4. Training and testing was done on one NVIDIA 2060 SUPER.

Traditional training took 0.8s per epoch on average and a best validation MSE of 0.00033 was achieved. When a sample prediction was performed with the ground truth output unknown, drifting occurred, as shown in Figure 2.

Iterative training took 5.9s per epoch on average and achieved a best validation MSE of 0.00029. After iterative training, drifting does not occur, and the prediction is much more accurate, as shown in Figure 3. As expected, uncertainty increases as the prediction goes further forward in time.

4 CONCLUSION

In this work, I presented the Transformer Kalman Network, an attention based sequence to sequence prediction model designed for time series with non-negligible uncertainty. Not only is the TKN efficient, it is also highly configurable and can accept any length of input and output sequences.

For noisy wave prediction, the TKN performed well. It's lack of recurrence or convolution allowed it to train very quickly. The TKN also generated excellent confidence intervals.

A major downside to the TKN is the drifting that occurs when training with the traditional method. A possible solution would be to use attention in the encoder and recurrence in the decoder, which could be especially effective when the input sequence is long but the output sequence is short.

I am very excited about the future of attention beyond NLP. In future work, I plan to apply the TKN to higher dimensional input and output spaces, such as the quad link pendulum done by Becker et al. [1]. In addition, I hope to incorporate elements of newer Transformer networks, such as Muppet networks [7].

ACKNOWLEDGMENTS

I would like to thank Dr. Rudolf Lioutikov for his invaluable support and advice throughout this project.

REFERENCES

[1] Philipp Becker, Harit Pandya, Gregor H. W. Gebhardt, Cheng Zhao, C. James Taylor, and Gerhard Neumann. 2019. Recurrent Kalman Networks: Factorized Inference in High-Dimensional Deep Feature Spaces. *CoRR* abs/1905.07357 (2019). arXiv:1905.07357 http://arxiv.org/abs/1905.07357

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR* abs/1810.04805 (2018). arXiv:1810.04805 http://arxiv.org/abs/1810.04805

[3] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735 arXiv:https://doi.org/10.1162/neco.1997.9.8.1735

[4] Ratnabali Pal, Arif Ahmed Sekh, Samarjit Kar, and Dilip K. Prasad. 2020. Neural Network Based Country Wise Risk Prediction of COVID-19. *Applied Sciences* 10, 18 (Sep 2020), 6448. https://doi.org/10.3390/app10186448

[5] Corentin Tallec and Yann Ollivier. 2017. Unbiasing Truncated Backpropagation Through Time. *CoRR* abs/1705.08209 (2017). arXiv:1705.08209 http://arxiv.org/abs/1705.08209

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. *CoRR* abs/1706.03762 (2017). arXiv:1706.03762 http://arxiv.org/abs/1706.03762

[7] Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu. 2019. ERNIE: Enhanced Language Representation with Informative Entities. *CoRR* abs/1905.07129 (2019). arXiv:1905.07129 http://arxiv.org/abs/1905.07129