DMSN Final Project: Improve LESSR Model Structure

LI-CHANG TENG[*]
Department of Electrical Engineering
National Cheng Kung University
n26091194@gs.ncku.edu.tw

TZU-JIUN SU[*]
Department of Electrical Engineering
National Cheng Kung University
n26100359@gs.ncku.edu.tw

GUAN-RU CHEN[*]
Department of Electrical Engineering
National Cheng Kung University
n26091136@gs.ncku.edu.tw

SHIH-FU SONG[*]
Department of Electrical Engineering
National Cheng Kung University
n26092116@gs.ncku.edu.tw

Abstract

In our final project, we tried to improve performance of LESSR[1], which is a session-based recommendation model based on Graph Neural Networks(GNNs)[2]. We found there is a GRU[3] layer in EOPA layer in LESSR[1]. Because GRU[3] has a problem with parallel processing, here we tried to replace it by 1) self-attention[4] 2) Transformer Encoder[4] to handle this problem. In experiments section, we conducted multi-head experiment for self-attention, and 1) feed-forward dimension 2) multi-head 3) number of layers experiments for Transformer due to it has more parameters than self-attention. Also we tried different attention mechanisms in SGAT layer and Readout layer in LESSR[1]. Finally, we found that replacing GRU[3] by TransformerEncoder[4] could increase performance in HR, MRR, NDCG metrics. The best result of total(sum) improvement in 3 metric is up to 1.03% without grid search hyperprameters turning.

1 INTRODUCTION

Due to the highly practical value, session-based recommendation attracted researchers' great attention. Most of the methods proposed earlier are based on Markov chains or recurrent neural networks (RNNs). Recently, Graph Neural Networks(GNNs)[2] have become increasingly popular and achieved state-of-the-art performance in many tasks. There are also some attempts to apply GNNs to session-based recommendation.

Although these GNN-based methods obtained exciting results and offered a new and promising direction for session-based recommendation, we observe that there are two information loss problems in these methods. The first information loss problem in the existing GNN-based methods is called the lossy session encoding problem. The second information loss problem is called the ineffective long-range dependency capturing problem where these GNN-based methods cannot effectively capture all long-range dependencies.

To solve the above problems, author propose a novel GNN model called LESSR (Lossless Edge-order preserving aggregation and Shortcut graph attention for Session-based Recommendation)[1]. Through the structure of the original paper, we found that there are one places with GRU and two place with attention mechanism. We hope to increase the speed of the model by changing the part of GRU. We first change GRU to self-attention[4], then change the model to Transformer[4], increase the number of layers and increase pos encoding, and finally see if the time and accuracy increase.

[*]Equal contribution. Listing order is random.

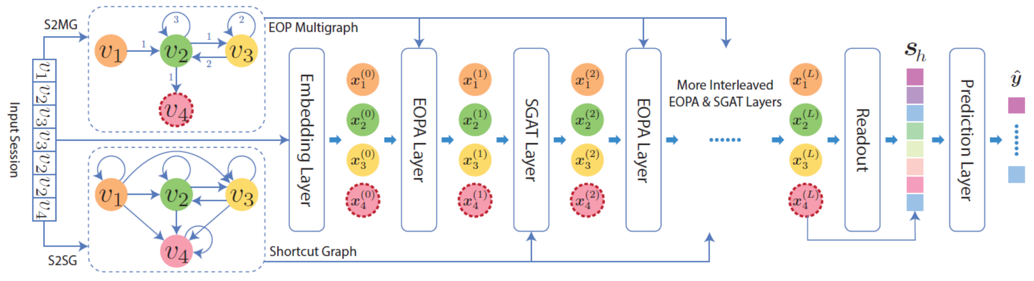

Figure 1: LESSR structure

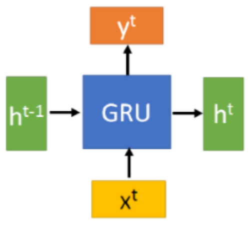

Figure 2: GRU structure

2 RELATED WORK

A given input session is first converted to a losslessly encoded graph called edge-order preserving (EOP) multigraph and a shortcut graph where the EOP multigraph could address the lossy session encoding problem and the shortcut graph could address the ineffective long-range dependency capturing problem. Then, the graphs along with the item embeddings are passed to multiple edge-order preserving aggregation (EOPA) and shortcut graph attention (SGAT) layers to generate latent features of all nodes. The EOPA layers capture local context information using the EOP multigraph and the SGAT layers effectively capture long-range dependencies using the shortcut graph, and the architecture diagram is as figure 1.

3 PRELIMINARY

3.1 GRU

GRU (Gate Recurrent Unit) is a type of Recurrent Neural Network (RNN). It is proposed to solve the problems of long-term memory and gradients in back propagation. GRU input and output structure: The input-output structure of GRU is the same as that of ordinary RNN. There is a current input x^t and the hidden state h^{t-1} passed down from the previous node. This hidden state contains information about the previous node. The GRU structure as shown in figure 2.

3.2 Self-Attention

For self-attention[4], the three matrices Q(Query), K(Key), and V(Value) are all from the same input. First, we have to calculate the dot product between Q and K, and then in order to prevent the result from being too large, we will Divide by $\sqrt{d_k}$, then pass the basis activity function, and finally multiply by V, it can be expressed by the following formula 1.

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})v \tag{1}$$

And multi-head means that we can have different Q, K, V representations, and finally combine the results.

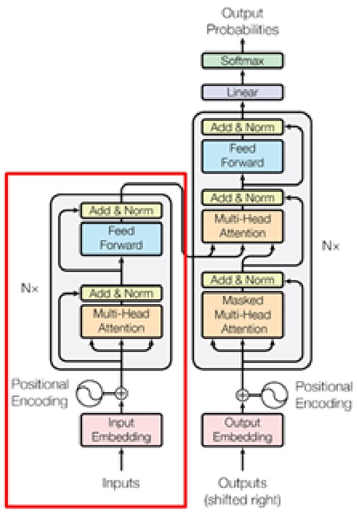

Figure 3: Transformer structure

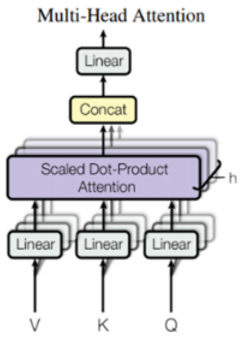

Figure 4: Multi-Head Attention

4 METHODOLOGY

4.1 Transformer Encoder

In this section, we will introduce the structure of transformer encoder, which will be used to replace the GRU layer in EOPA in the following experiments.

Each transformer encoder has two sub-layers. The first one is Multi-Head Attention layer, and the second one is a fully connected feed forward layer. After both layer, we employ residual connection layers and layer normalization layers. We will specific each layer later. The Transformer structure is shown in figure 3.

4.2 Multi-Head Attention

The first one is the Multi-Head Attention layer. In this layer our goal is to get attention score from the graphs. In Multi-Head Attention layer, we have three groups of variables, Q, k, V, represent query, keys, value respectively. The number i in these groups (ex: $Q\{q_1, q_2, \ldots, q_i\}$) is decided by how many heads we use. After that, we will use the following formula to compute the attention weight, where d_k represents vector dimensionality of k, v and Z represents final attention weight. The Multi-Head Attention structure is shown in figure 4.

4.3 Add & Norm

In these layers we use residual connection and layers normalization to help our model training. With residual connection, we can let gradient flow directive through the network, preventing gradient vanishing. As for layers normalization, it's effective at stabling the hidden state dynamic our model. In mathematical definition, residual connection can be simply denoted by the following formula 2.

$$\text{add}(x) = x + \text{sublayer}(x) \tag{2}$$

On the other hand, layers normalization is much complex. Assume we give input x over a mini batch size of m, which is $B = x_1, x_2, \ldots, x_m$. Each sample x contain K elements. We first compute the mean and variance of each sample x, denoted by the following formula 3.

$$\mu_i = \frac{1}{K}\Sigma x_{i,k}\sigma_i^2 = \frac{1}{K}\Sigma(x_{i,k} - \mu_i)^2 \tag{3}$$

Then we normalize each sample such that the elements in the sample have zero mean and unit variance as formula 4. ϵ is for numerical stability in case the denominator becomes zero by chance.

$$x_{i,k} = \frac{(x_{i,k} - \mu_i)}{\sqrt{\sigma_i^2 - \epsilon}} \tag{4}$$

Combining these two layers we simplify its as formula 5

$$\text{LayerNorm}\,(x + \text{sublayer}(x)) \tag{5}$$

Recall that sublayers is Multi-Head Attention and feed forward network.

4.4 Feedforward Network

In addition to attend sub-layers, we employ a fully connected feed-forward network shown as formula 6 in our encoder. This consist of two linear transformation with a ReLU activation function.

$$\text{FFN}\,(x) = \max{(0, xW_1 + b_1)}W_2 + b_2 \tag{6}$$

4.5 Positional Encoding

Since we change the original GRU to Transformer Encoder, which contains no recurrence. That is to say, we lost the time information of our graph datasets. In order to solve this problem, we use a technique called "Positional Encoding" to help us to regain those information back. More specifically, we inject a Positional Encoding matrix, which has the same dimension as the input embedding, so we can sum them and inject the specific time information to corresponding data.

As shown in Figure 5, every column of the matrix represents a unique time, so by concatenating this to the embedding we are able to handle time information while using self-attention and Transformer Encoder. The positional encoding formulas shown in following:

$$\text{PE}_(\text{pos}, 2i) = \sin(\frac{\text{pos}}{10000^(2_i/d_{model})}) \tag{7}$$

$$\text{PE}_(\text{pos}, 2i+1) = \cos(\frac{\text{pos}}{10000^(2_i/d_{model})}) \tag{8}$$

Where d_{model} represents that embedding dimension, and pos is the position.

5 EXPERIMENTS

In this section, we will introduce experiment setting, dataset, and analyze the experiment result. We conducted several experiments to check our hypotheses and evaluate our model with chosen metric.

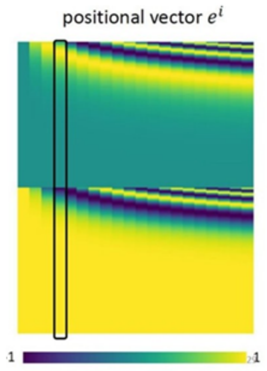

Figure 5: Positional Encoding

Table 1: Statistics of dataset

Diginetica	
No. of Clicks	981,620
No. of Sessions	777,029
No. of Items	42,596
Average length	4.80

5.1 Dataset

We choose Diginetica dataset[2] following LESSR [1] paper, which is the CIKM cup 2016 dataset provided by DIGINETICA Crop. There are 6 files in Diginetica dataset, but we only need the transaction one. As [1], we used last week sessions as test data. We got the same training and test set by following preprocessing method described in [1]. Statistics of Diginetica dataset is shown in Table 1.

5.2 Baseline and metrics

We choose [1] as out baseline model, then we tried to improve model structure in [1] by some changes. Comparing the metrics to [1], we could know the change is positive or negative influence. Following [1], the metrics we used are HR@20 (Hit Rate) and MRR@20 (Mean Reciprocal Rank).

5.3 MultiheadAttention layer layer

MultiheadAttention layer[3] is a official implemented self-attention layer by pytorch. Here we replace GRU[4] layer in EOPA block in [1] by MultiheadAttention layer layer. All settings are the same but GRU now replaced by MultiheadAttention layer. We adjusted num of heads parameter in MultiheadAttention layer layer to see the influence of multi-head attention.

The pytorch official did not implemented positional encoding in MultiheadAttention layer layer, so there is no position information within layer. To handle this problem we need to do position encoding manually. We found a official tutorial[5] that manually implemented position encoding, so we followed the encoding method here.

Table 2 and Table 3 are the experiment result without and with positional encoding respectively. It turns out no matter multi-head or positional encoding, can not improve the result. So in next section we decided to using more complex layer.

[2]`https://competitions.codalab.org/competitions/111610`
[3]`https://pytorch.org/docs/stable/generated/torch.nn.MultiheadAttentionlayer.html`
[4]`https://pytorch.org/docs/stable/generated/torch.nn.GRU.html`
[5]`https://pytorch.org/tutorials/beginner/transformer_tutorial.html`

Table 2: Multi-head w/o pos encoding

AGG.TYPE	HR@20	MRR@20	NDCG@20	Total impv.
baseline	52.82	18.3	25.93	-
Head=1	52.65	18.25	25.85	-0.90
Head=2	52.58	18.27	25.84	-0.97
Head=4	52.6	18.28	25.85	-0.83
Head=8	52.63	18.28	25.87	-0.70
Head=16	52.62	18.28	25.86	-0.76
Head=32	52.64	18.29	25.87	-0.63

Table 3: Multi-head with pos encoding

AGG.TYPE	HR@20	MRR@20	NDCG@20	Total impv.
baseline	52.82	18.3	25.93	-
Head=1	52.57	18.26	25.83	-1.08
Head=2	52.55	18.29	25.85	-0.87
Head=4	52.59	18.3	25.88	-0.63
Head=8	52.62	18.29	25.87	-0.66
Head=16	52.54	18.31	25.86	-0.75
Head=32	52.57	18.32	25.89	-0.52

5.4 TransformerEncoder layer

In this section, we use transformer encoder [6] to replace GRU. TransformerEncoder has a lot of hyperparameter, so we conducted 3 main experiments to tuning the model: 1) dim_feedforward 2) nhead 3) encoder_layer. Also, each main experiments have two sub experiments: 1) w/o pos encoding 2) with pos encoding.

5.4.1 Dim_Feedforward Experiment

In this experiment we fix all hyperprameters but dim_feedforward. Table 4 shown the result without positional encoding. Table 5 shown the result with positional encoding.

From Table 4 and Table 5, we found that the best dim_feedforward is setting 512, whether with pos encoding or not, dimension 512 in both case has a good result, so we choose dimension 512 for our model in the later experiments.

[6]https://pytorch.org/docs/stable/generated/torch.nn.TransformerEncoder.html

Table 4: dim exp w/o pos encoding

AGG.TYPE	HR@20	MRR@20	NDCG@20	Total impv.
baseline	52.82	18.3	25.93	-
Dim = 2048	52.73	18.25	25.83	-0.83
Dim = 1024	52.9	18.31	25.86	-0.06
Dim = 512	52.97	18.35	26	0.83
Dim = 256	52.67	18.28	25.88	-0.59
Dim = 128	52.88	18.34	25.94	0.37
Dim = 64	52.84	18.39	26.01	0.84
Dim = 32	52.7	18.24	25.85	-0.86
Dim = 16	52.68	18.3	25.88	-0.46

Table 5: dim exp with pos encoding

AGG.TYPE	HR@20	MRR@20	NDCG@20	Total impv.
baseline	52.82	18.3	25.93	-
Dim = 2048	52.74	18.28	25.88	-0.45
Dim = 1024	52.86	18.3	25.88	-0.12
Dim = 512	52.85	18.36	25.97	0.54
Dim = 256	52.7	18.26	25.86	-0.72
Dim = 128	52.89	18.37	25.95	0.59
Dim = 64	52.79	18.34	25.95	0.24
Dim = 32	52.74	18.29	25.9	-0.32
Dim = 16	52.6	18.36	25.92	-0.13

Table 6: multi-head exp w/o pos encoding

AGG.TYPE	HR@20	MRR@20	NDCG@20	Total impv.
baseline	52.82	18.3	25.93	-
nhead=1	52.97	18.35	26	0.83
nhead=2	52.77	18.37	25.95	0.36
nhead=4	52.98	18.35	26	0.85
nhead=8	52.87	18.37	25.96	0.59
nhead=16	52.78	18.37	25.97	0.46
nhead=32	52.92	18.41	25.97	0.95

5.4.2 Multi-Head Experiment

Here we fixed all hyperprameters but nhead to see the influence. Also, The dim_feedforward set to 512. Result without positional encoding shown in Table 6 and result with positional encoding shown in Table 7.

Comparing Table 6 and Table 7, We found the metrics without positional encoding are usually better than the other one. So positional information might not a critical info in this scenario.

Note that best performance appeared when nhead set to 8 with positional encoding.

5.4.3 Num-Layers Experiment

Here all hyperprameters was fixed but num_layers will be change. The dim_feedforward was set to 512 and nhead was set to 1. Table 8 and Table 9 shown the result without and with positional encoding respectively.

We found that whether transformer encoder with positional encoding or not, it has similar trend that performance decreased as layers increased. And we found as layers increased, model's inference time also increased.

Table 7: multi-head exp with pos encoding

AGG.TYPE	HR@20	MRR@20	NDCG@20	Total impv.
baseline	52.82	18.3	25.93	-
nhead=1	52.85	18.36	25.97	0.54
nhead=2	52.7	18.31	25.91	-0.25
nhead=4	52.88	18.38	25.98	0.74
nhead=8	52.87	18.41	26.03	1.08
nhead=16	52.82	18.35	25.96	0.39
nhead=32	52.75	18.35	25.95	0.22

Table 8: num-layers exp w/o pos

AGG.TYPE	HR@20	MRR@20	NDCG@20	Total impv.
baseline	52.82	18.3	25.93	-
layer=1	52.97	18.35	26	0.83
layer=2	52.72	18.41	25.93	0.41
layer=3	52.69	18.4	25.95	0.38
layer=4	52.69	18.33	25.89	-0.24
layer=6	52.65	18.13	25.72	-2.06
layer=8	52.24	17.96	25.48	-4.69
layer=16	52.11	17.77	25.34	-6.52

Table 9: num-layers exp with pos

AGG.TYPE	HR@20	MRR@20	NDCG@20	Total impv.
baseline	52.82	18.3	25.93	-
layer=1	52.85	18.36	25.97	0.54
layer=2	52.77	18.41	25.96	0.62
layer=3	52.72	18.34	25.88	-0.16
layer=4	52.84	18.32	25.9	0.03
layer=6	52.8	18.18	25.78	-1.27
layer=8	52.3	17.95	25.46	-4.71
layer=16	52.04	17.81	25.37	-6.31

5.5 Multi-Head Attention + EOPA layer

After changing GRU to another network, we comprehended more methods trying to surpass the baseline.We figured out to concatenate networks orderly rather than replacing it. Then, the results showed as Table 10. Beyond our expectations, we thought the results would turn out to be better as head numbers count up.

5.6 Readout Layer Exploration

Above all of experiments, we only focus on EOPA layer, but there are three of layers which can be changed using Self-Attention. The two of last layers are Readout and SGAT layer. In this section, we want to discuss how can we do with Readout layer. According to the [1], readout layer computes a graph-level representation by aggregating node representations using the attention mechanism. [1] let $X_{last}^{(L)}$ denote the final node representation of the last item in the session. The graph-level representation h_G is defined as follows:

$$h_G = \sum_{i \in V} \beta_i X_i^{(L)} \tag{9}$$

$$\beta = softmax(\epsilon) \tag{10}$$

Table 10: MULTI-HEAD ATTENTION PLUS GRU

AGG.TYPE	HR@20	MRR@20	NDCG@20	Total impv.
baseline	52.82	18.3	25.93	-
Head = 1	52.39	17.93	25.52	-4.22
Head = 2	52.42	17.94	25.55	-3.99
Head = 4	52.46	17.96	25.56	-3.77
Head = 8	52.47	17.97	25.59	-3.58
Head = 16	52.46	18.00	25.59	-3.44

Table 11: Setting 1 to 7

Setting counts	Mechanism of Attention
Original	$\epsilon_i = q^T \sigma(W_1 x_i^L + W_2 x_{last}^L + r)$
Setting 1	$\epsilon_i = q^T \sigma(W_1 x_i^L \odot W_2 x_{last}^L + r)$
Setting 2	$\epsilon_i = q^T \textbf{tanh}(W_1 x_i^L + W_2 x_{last}^L + r)$
Setting 3	$\epsilon_i = q^T W_3([(W_1 x_i^L + r)\|\|(W_2 x_{last}^L)])$
Setting 4	$\epsilon_i = q^T \textbf{tanh}(W_3([(W_1 x_i^L + r)\|\|(W_2 x_{last}^L)]))$
Setting 5	setting 4 with $h_G = \sum_{i \in V} \beta_i \boldsymbol{W_v} X_i^{(L)}$
Setting 6	$\epsilon_i = q^T \textbf{tanh}(W_1 x_i^L + W_2 x_{last}^L + r)$
Setting 7	setting 6 with $h_G = \sum_{i \in V} \beta_i \boldsymbol{W_v} X_i^{(L)}$

Table 12: READOUT ATTENTION

baseline	HR@20	MRR@20	NDCG@20	Total impv.
baseline	52.82	18.3	25.93	-
Setting 1	52.77	18.27	25.89	-0.41
Setting 2	52.81	18.26	25.9	-0.35
Setting 3	52.58	18.21	25.78	-1.52
Setting 4	52.6	18.15	25.76	-1.89
Setting 5	52.62	18.16	25.72	-1.95
Setting 6	52.65	18.2	25.78	-1.45
Setting 7	52.64	18.14	25.77	-1.83

$$\epsilon_i = q^T \sigma(W_1 x_i^L + W_2 x_{last}^L + r) \tag{11}$$

where $q, r \in R^d$ and $W_1, W_2 \in R^{d \times d}$ are learnable parameters.

Apparently, we are able to revise the eqn. 9 to 11 because the mechanism of combining two features is the sign of '+'. As the consequence, changing mechanism with their own built model is our next topic to discover. There are lots of things we can do about the model, but one constraint is numbers of dimension. Except that changing the mechanism, the activation function and linear transformation are two main methods we choose to use in the model. Table 11 below is the 7 settings we modify in the function and Table 12 is the results after modifying.

5.7 SGAT Layer Exploration

Let's introduce basic information about how self-attention works before starting. Defining a original feature set as $\{x^i | x^i \in ; i \in N\}$, we can get q, k,and v vectors after passing x^i through linear transform.Define q, k,and v as follows :

$$q^i = W^q X^i \tag{12}$$

$$k^i = W^k X^i \tag{13}$$

$$v^i = W^v X^i \tag{14}$$

where q means query (to match others), k means key (to be matched), and v means information to be extracted. Making vector q and k match is our next step. Therefore, we use the function defined as 'Scaled Dot-Product Attention' to get attention weight vector:

$$\alpha_{1,i} = \frac{q_1 \cdot k}{\sqrt{d}} \tag{15}$$

where $\cdot$ means dot product, d is the dimension of q and k. Softmax function is attached to avoid over-fitting, so $\alpha_{1,i}$ is going to be like :

$$\hat{\alpha}_{1,i} = \frac{exp(\alpha_{1,i})}{\sum_j exp(\alpha_{1,i})} \tag{16}$$

Multiply attention weight with v and we'll get attention output vector:

$$\text{attn\_out} = \sum_i \hat{\alpha}_{1,i} v^i \tag{17}$$

Above all are basic attention mechanism, and the main purpose is to built our own self-attention among SGAT layer's features of q and k. Due to limitation of dimension and little knowledge of dgl library which is used in GNN network, we can't directly use $\boldsymbol{multi-headattention}$ function built in pytorch library. As the result, the aim of self-attention is to learn merely more information on q and k before sending them to SGAT layer.

Beyond our expectation, nonetheless, the results show that the machine can't even learn a thing. We conclude that cause happened because of complicating model architecture after adding self-attention among SGAT layer.

6 CONCLUSION

In our experiments, no matter with positional encoding or not, the performance of MultiheadAttention layer is worse than GRU, although using MultiheadAttention layer is slightly fast. In multi-head experiment of TransformerEncoder, we found there is no distinct trend about num of head, also we found when TransformerEncoder stacking more layers, model's performance will decreased, this means the model is over-fitting, moreover, evaluate time will increased, this cause training time getting longer.

After several experiments, we found that TransformerEncoder can improve model performance by replacing GRU in EOPA layer, but the parameter of TransformerEncoder need in proper setting. In SGAT and Readout layer, we tried to change the attention mechanisms, but we didn't got a good result. This might because attention mechanisms is dependent on model structure and data, it won't be useful if we just replace attention mechanisms from different paper.

Finally the best result we got is when layer=1 and nhead=8 with positional encoding, total improvement of three metrics is 1.08%, although there is still a space for improvements in this model. The result is a acceptable to us because we didn't change the model structure so much but just change a layer inside EOPA block. If we could modify SGAT and Readout layer's attention mechanisms to multi-head attention, we might get a better result.

References

[1] Tianwen Chen and Raymond Chi-Wing Wong. Handling information loss of graph neural networks for session-based recommendation. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20)*, pages 1172—1180, 2020.

[2] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009. doi: 10.1109/TNN.2008.2005605.

[3] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014.

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