A write-up on Self-Attentive Sequential Recommendation

Aniruddha M. Mishra

411650, National Institute of Technology, Andhra Pradesh

Abstract - Transformer model is purely based on attention mechanism (self-attention)(Alammar, 2018b). Here self-attention mechanism is applied to sequential recommendation system. The goal is to draw context from all actions in the past (RNN) and frame prediction in terms of small number of action (MC).Self-Attentive Sequential Recommendation (SASRec) (Kang & McAuley, 2018) is faster than CNN/RNN based alternatives (as Self-attention block is suitable for parallel acceleration).This model is based on Transformer model,where self-attention is used. Transformer model has achieved state-of-the-art performance in NLP and machine translation. Here, an attempt to use this model in recommendation system is done.

1 Introduction

Basic two approaches for recommendation system includes -

1. Markov Chain (MC)
   (a) Assumes users next action can be predicted on the basis of just their last(or last few) actions.
   (b) Performs best in sparse datasets where model parsimony is critical.
2. RNNs
   (a) Allows long-term semantics to be uncovered.
   (b) Performs best in denser datasets where higher complexity is affordable.

To balance the shortcomings of both these models **SASRec** is introduced.

1. Allows to capture long term semantics like RNN.
2. Uses Attention Mechanism (makes prediction on relatively few actions like MC).

2 Kinds of Recommendation

1. *General Recommendation* - Recommender models the compatibility between user and items using history feedback.

   User Feedbacks

   - *Explicit* - Ratings
   - *Implicit* - Clicks, purchases, search history, etc.

   (a) *Matrix Factorization (MF)* - Uncovers latent dimensions to represent user's preferences and item properties.

   (b) *Item Similarity Model (ISM)* - Doesn't explicitly model each user with latent factors. Instead learns an item-item similarity matrix and measures users' preference towards an item via measuring its similarities with the items the user has interacted before.

2. *Temporal Recommendation* - Models timestamps of users' activities. **TimeSVD++** splits time into several segments and then models users and items in each segment. Sequential recommendation differs slightly as it only considers order of actions and models sequential patterns independent of time.

3. *Sequential Recommendation* - First-order MC assumes next action is related to the previous one. Since the last visited is often the key factor affecting users' next action, first-order MC shows strong performance. High-order MC takes several previous actions into consideration.
   GRU4Rec uses Gated Recurrent Units (GRU) to model click sequences for session-based recommendation. In each step RNNs take the state from last step and current action as its input. These dependencies make RNNs less efficient, though techniques like 'session-parallelism' have been proposed

4. *Attention Mechanism* - Basic idea is that sequential outputs each depend on 'relevant' parts of some input that model should focus on successively. Attention technique used above is essentially and additional component to the original model.

3 Transformer Model

To understand self-attention let's take quick look at transformer model(Alammar, 2018a).

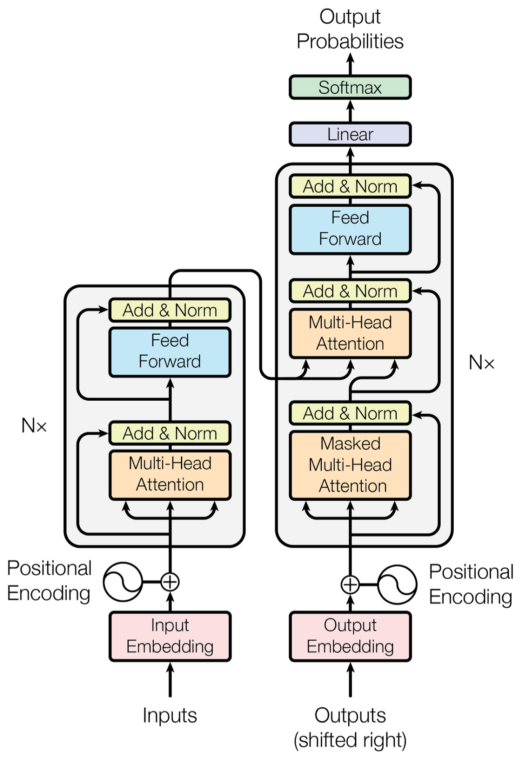

Figure 1: The Transformer - model architecture.

The transformer model consists of -

1. A pack of six encoders (left block in the fig.1) and decoders (right block in fig.2).
2. Each encoder consists of a self-attention layer and a feed-forward layer.
   - Self-attention layer helps encoder look at other activities in the sequence.
   - Output of self-attention layer is fed to feed-forward network
3. Decoder too, has both these layers but between then is a self-attention layer that helps decoder focus on relevant part of the input sequence.

Let's see how the input flows between these components to generate output.

3.1 Embedding Layer

1. Transform the input sequence $(S_1^u, S_2^u, S_3^u, ...S_{|S^u|-1}^u)$ to fixed-length sequence $s = (s_1, s_2, s_3, ..., s_n)$.
    - If sequence length is greater than n, take the n recent actions.
    - If sequence length is less than n, repeatedly add a 'padding item' to left.
2. Create Item-Embedding matrix M.
3. Retrieve Input-Embedding matrix E

$$E_i = M_{s_i}$$

Position Embedding - To account for order of actions in input sequence,the transformer adds a vector to each input embedding. The intuition here is that adding these values to the embeddings provides meaningful distances between the embedding vectors once they're projected into Q/K/V vectors and during dot-product attention

$$\hat{E} = \begin{bmatrix} M_{s_1} + P_1 \\ M_{s_2} + P_2 \\ ... \\ M_{s_n} + P_n \end{bmatrix}$$

After embedding the sequence, each of them flows through each of the two layers of encoder. Word in each position flows through its own path in encoder. There are dependencies between these words in self-attention layer. The feed-forward layer does not have these dependencies and can be executed in parallel.

3.2 Encoding

List of vectors is received as input, which is passed through self-attention layer then through feed-forward network.

3.2.1 Self-Attention

1. First step is to create 3 vectors from encoder's input. So, for each embedding a Query vector, a Key vector and a Value vector is created. These vectors are created by multiplying the embedding by 3 matrices (W^Q, W^K, W^V) that we trained during the training process.

2. Second step is to calculate a score. It is done by calculating the matrix product of the query vector Q and key vector K.

3. Divide the score by square root of dimension d of key vector. Pass this result to SoftMax. SoftMax normalizes the scores, so that they add to 1.

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d}}\right)V$$

4. Now, multiply each value vector by the SoftMax score.

5. Sum up the weighted value vectors. This is the output of self-attention layer. This is sent along to feed-forward network.

In our case, the self-attention operation takes the embedding $\hat{E}$ as input, converts it to three matrices through linear projections, and feeds them into an attention layer:

$$S = SA(\hat{E}) = Attention(\hat{E}W^Q, \hat{E}W^K, \hat{E}W^V)$$

Multi-head attention is used here. This improves the performance of attention layer in 2 ways:

1. It expands the model's ability to focus on different positions.

2. It gives the attention layer multiple "representation subspaces" (i.e. multiple sets of Query, Key, Value matrices, each set is used to project input embeddings into different subspaces).

Now, all the Z matrix produced by different attention-heads are concatenated and multiplied by W^O to give resultant matrix Z which is sent to FFN.

3.2.2 Feed-forward Network

Though the self-attention is able to aggregate all previous items' embeddings with adaptive weights, ultimately it is still a linear model.

To endow the model with nonlinearity and to consider interactions between different latent dimensions, we apply a point-wise two-layer feed-forward network to all Si identically (sharing parameters):

$$F_i = FFN(S_i) = ReLU(S_iW^{(1)} + b^{(1)})W^{(2)} + b^{(2)}$$

where $W^{(1)}$,$W^{(2)}$ are $d \times d$ matrices and $b^{(1)}$,$b^{(2)}$ are d-dimensional vectors.

3.3 Stacking Self-Attention Blocks

After the first self-attention block, F_i aggregates all previous items' embeddings. However, it might be useful to learn complex item transition by another self-attention on F_i . So we stack a layer of self-attention blocks. (i.e. self-attention layer and feed-forward network). If network goes deeper some problems arises viz.

- Overfitting
- Training process becomes unstable (due to vanishing gradients)
- Model with more parameters require more time for training

To alleviate these problems,

$$g(x) = x + Dropout(g(LayerNorm(x)))$$

g is *self-attention* or *feed-forward network.* Suppose, for layer g in each block, we apply layer normalization on the input x before feeding into g, apply *dropout* on $g's$ output, and add the input x to the final output

3.4 Residuals

- Each sub-layer in encoder has a residual connection, followed by layer-normalization(keitakurita, 2018).
- In batch normalization, the statistics are computed across the batch and are the same for each example in the batch.
- In contrast, in layer normalization, the statistics are computed across each feature and are independent of other examples. Unlike batch normalization, statistics used in layer norm are independent of other samples in the same batch.

  $$g(x) = x + Dropout(g(LayerNorm(x)))$$

Dropout - To alleviate overfitting problem dropout regularization technique is used. Basic idea is to turn off neurons with probability p while training and used all of them when testing. It can be viewed as a form of ensemble learning

3.5 Decoder

- The output of the top encoder is then transformed into a set of attention vectors K and V. These are to be used by each decoder in its "encoder-decoder attention" layer which helps the decoder focus on appropriate places in the input sequence.

- The output of each step is fed to the bottom decoder in the next time step, and the decoders bubble up their decoding results just like the encoders did. Also, we embed and add positional encoding to those decoder inputs to indicate position of each word.

- In the decoder, the self-attention layer is only allowed to attend to earlier positions in the output sequence. This is done by masking future positions (setting them to $-inf$) before the $softmax$ step in the self-attention calculation

3.6 Final *Linear* and *Softmax* Layer

- Linear Layer is just a fully connected neural network that projects the vector produced by the stack of decoders, into a much, much larger vector called *logits* vector.

- Let's assume that our model knows 5,000 movies that it learned from its training datasets. This would make the *logits* vector 5,000 cells wide - each cell corresponding to the score of each movie. The *softmax* layer then turns those scores to probabilities and the cell with the highest probability is chosen.

Explicit User Modelling - To provide personalized recommendation an additional user embedding matrix can be added to the input embedding matrix. An explicit user embedding represents users' preferences and implicit user embedding consists of users' previous actions and visited items, etc.

4 Complexity

Space Complexity
The learned parameters in our model are from the embeddings and parameters in the self-attention layers, feed-forward networks and layer normalization.The total number of parameters is $O(\mid I \mid d + nd + d2)$, which is

moderate compared to other methods (e.g. $O(\mid U \mid d+ \mid I \mid d)$ for FPMC) since it does not grow with the number of users, and d is typically small in recommendation problems.
I – Item Set
U – User Set
d – Latent vector dimension

Time Complexity
Mainly due to the self-attention layer and the feed-forward network, which is $O(n^2d + nd^2)$. The dominant term is typically $O(n^2d)$ from the self-attention layer. A convenient property in our model is that the computation in each self-attention layer is fully parallelizable, which is amenable to GPU acceleration.

5 Conclusion

- The proposed Self Attention Sequential Recommendation model (SASRec) models the entire user sequences and adaptively considers consumed items of prediction.

- SASRec model has worked well with both sparse and dense datasets. This model outperforms state-of-the-art baselines.

- SASRec model is faster than conventional RNN/CNN based approaches due to more scope of parallelization.

References

Alammar, J. (2018a). *The Illustrated Transformer.* http://jalammar.github.io/illustrated-transformer/.

Alammar, J. (2018b). *Mechanics of seq2seq model with Attention.* https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/.

Kang, W.-C., & McAuley, J. (2018). Self-attentive sequential recommendation. In *2018 ieee international conference on data mining (icdm)* (pp. 197–206).

keitakurita. (2018). *Weight Normalization and Layer Normalization explained.* https://mlexplained.com/2018/01/13/weight-normalization-and-layer-normalization-explained-normalization-in-deep-learning-part-2/.