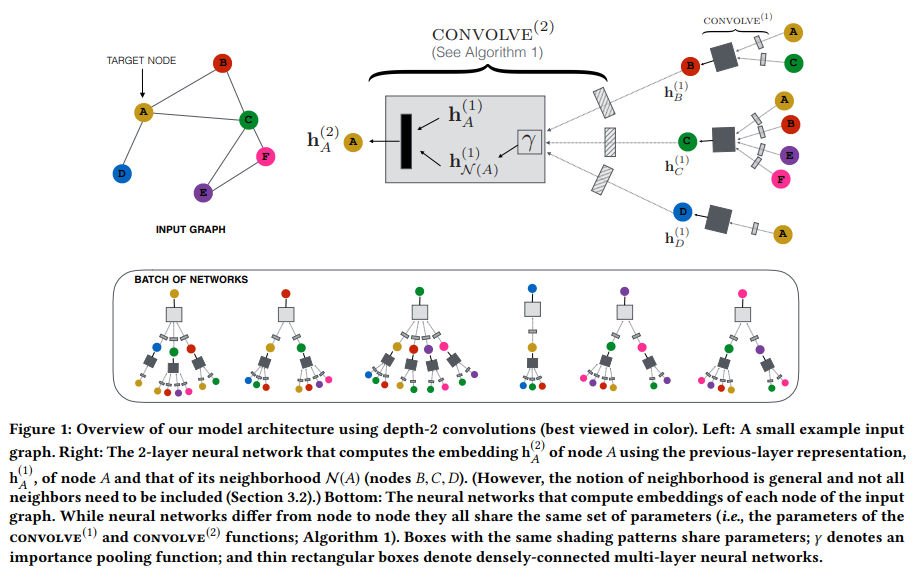
**Paper Summary 9 by Yingtao Luo**

**Title: “Graph Convolutional Neural Networks for Web-Scale Recommender Systems”.**

The paper practices a large-scale deep recommendation engine deployed at Pinterest. The major challenge is the efficient training of large graphs with billions of nodes and 18 billions edges. The authors propose a random-walk-based GCN named PinSage. The On-the-fly convolution in PinSage performs efficient and localized convolutions, which samples the nerighbors around a node and dynamically construct computation graph, as visualized in the figure below. The producer-consumer minibatch construction can ensure maximal GPU utilization during model training. A large-memory, CPU-bound producer samples neighborhoods and fetches the features to define local convolutions, while a GPU-bound model consumes these computation graphs to run StochasticGD.

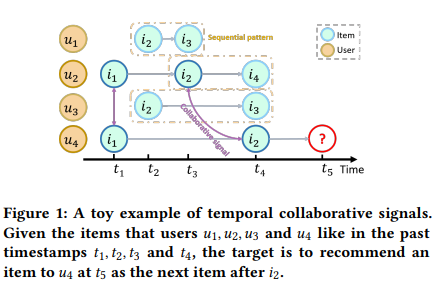


Random walks are used for constructing convolutions so each node has an importance score to perform pooling and aggregation, which replaces the random sampling. They design a curriculum training scheme where the algorithm is fed harder examples. Also, they use MapReduce to project all pins to a low-dimensional latent space to perform the aggregation. Another MapReduce job is used to join the representation with the ids of the boards they occur in. The board embedding is computed by pooling the features of the sampled neighbors. This avoids redundant computations since the latent vector for each node is computed once. After the offine training that obtains the embeddings, the online lookup operation can be used to efficiently inference by approximating KNN via locality sensitive hashing. The experiments show that PinSage outperforms other methods in hit-rate and MRR, and PinSage wins over other methods in head-to-head comparison of which image is more relevant to the recommended query. The production A/B test also shows that the metric of interest, i.e. repin rate measuring the percentage of homefeed recommendations that have been saved by the users, improves by 10-30%. The running time also shows that PinSage can stably and efficiently work for different batch sizes. The trade-off between metrics (such as Hit-rate and MRR) and the training time with respect to the number of neighbors sampled is also shown.

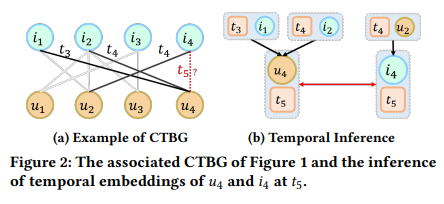
**Title: “Continuous-Time Sequential Recommendation with Temporal**

**Graph Collaborative Transformer”.**

**Introduction and motivation:** The author argue that most sequential recommendation models in the literature ignore temporal collaborative signals. The authors use an example to illustrate that. In below, the user 4 should not only use the most recent interacted item to predict next item, but also use the temporal collaborative signals between users of similar patterns (which is the definition of collaborative signals that similar patterns of both items and users should be considered).



The authors propose to construct continuous-time bipartitie graph (CTBG). The graph consists of user/item nodes and interacted user/item are connected by a timestamp edge, as shown below. Except that the sequence of interacted items of u4 can have a sequential pattern that is usefor, the probability of recommending i4 for u4 in t5 is also affected by the i4 of u2 at t4, as u2 has collaborative temporal signals that are useful for u4.



**Methods:**

The methods part is the core methodology the authors practice. We will focus on this part and give summary of each subsection to comment on the proposed method.

1. **Preliminary:**

The authors define the continuous-time bipartitie graph as follows. A continuous time bipartite graph with 𝑁 nodes and 𝐸 edges for recommendations is defined as B = {U, I, }, where U and I are two disjoint node sets of users and items, respectively. Every edge is denoted as a tuple 𝑒 = (𝑢,𝑖,𝑡), where 𝑢 U, 𝑖 I, and 𝑡 as the edge attribute. Each triplet (𝑢,𝑖, 𝑡) denotes the interaction of a user 𝑢 with item 𝑖 at timestamp 𝑡. Let denotes the set of items interacted with the user u before t. For a specific user 𝑢, given a set of future timestamps , the model recommends every timestamp .

**Summary of 1:** The authros define bipartitie graph from the user/item sequence. That is the foundation of what is discussed in the following subsections.

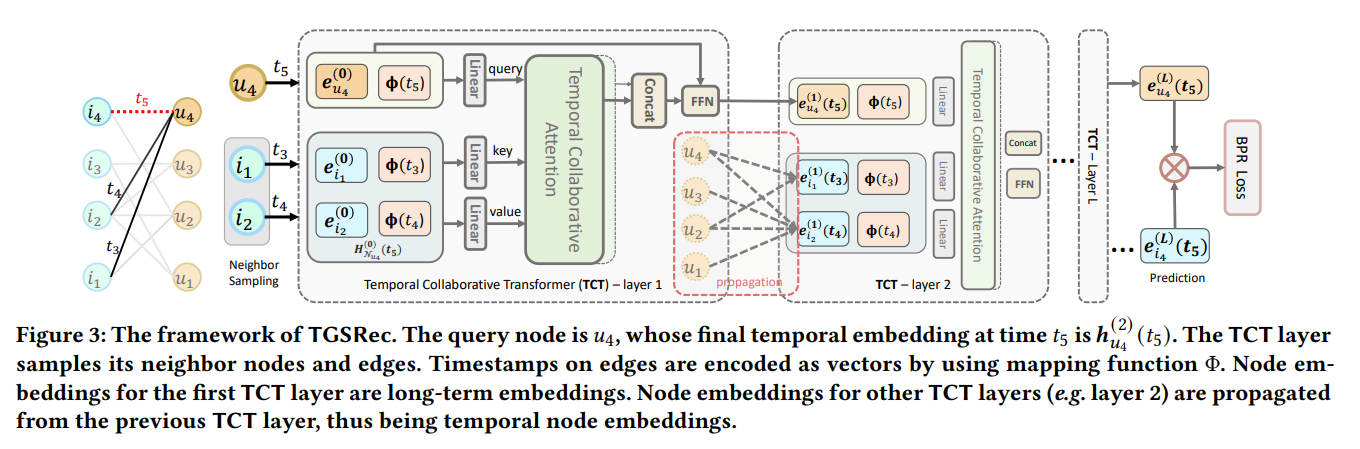
1. **Neural Architecture:**

The authors propose Temporal Graph Sequential Recommender (TGSRec).

There are three major components of the proposed neural architecture:

1. **Embedding layer,** which encodes nodes and timestamps in a consistent way to connect the SR problem with graph embedding method;
2. **Temporal Collaborative Transformer (TCT) layer**, which employs a novel temporal collaborative attention mechanism to discriminate temporal impacts of neighbors, and aggregates both node and time embeddings to infer the temporal node embedding;
3. **Prediction layer**, which utilizes output embeddings from the final TCT layer to calculate the score.

The architecture is visualized below.



* 1. **Embedding layer:**

The embedding layer are two types of embeddings:

1. **the long-term embeddings of nodes**

We define user/item node embedding as



We define user/item embedding table as



1. **the continuous time embeddings of timestamps on edges.**

It behaves as a function that maps those scalar timestamps into vectors, i.e. maps those scalar timestamps of interacted items into vectors. It also represents the time span as the dot product of corresponding encoded time embeddings. Given a pair of interactions (𝑢,𝑖, 𝑡1) and (𝑢,𝑗, 𝑡2) of the same user, we have



Here, K is the temporal kernel. The temporal effect 𝜓 (𝑡1 − 𝑡2) measures the temporal correlation between two timestamps. The continuous time encoding is a function that maps those scalar timestamps into vectors. It also represents the time span as the dot product of corresponding encoded time embeddings. The temporal effect 𝜓 (𝑡1 − 𝑡2) measures the temporal correlation between two timestamps.



**Summary of 2.1:** The authros define direct user/item embedding and then introduce the core continuous-time embedding of timestamps to consider temporal effect. The way to model this embedding is to first encodes two timestamps and then multiply them as a kernel to measure the relative time difference effect that can generalize to any time. The kernel is continuous and translation-invariant based on Bochner’s Theorem. The last equation shows how the time encoding works, which is to use sine function of many dimensions of different weighted t and put a normalized term in the front to prevent the overall value of the dense representation from growing too quickly with dimensions. So far, everything makes sense and similar embeddings have been seen in the literature, e.g. papers in sequential recommendation and the papers we read in the class.

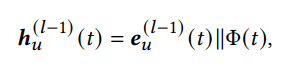
**2.2 Temporal collaborative transformer**

The proposed temporal collaborative transformer consists of **information construction, information propagation, temporal collaborative attention,** and **information aggregation**. Two strengths of a TCT layer are:

1. constructing information from both user/item embeddings and temporal embedding, which characterizes temporal effects;
2. a collaborative attention models the importance of user-item interactions, which is able to explicitly recognize collaborative signals.

**2.2.1 Information construction**

The *query* input information at the 𝑙-th layer for user 𝑢 at time 𝑡 is



𝑙 = 1, 2, . . . , L.



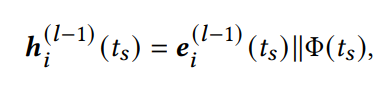
When 𝑙 > 1, the temporal embedding is from the previous TCT layer.

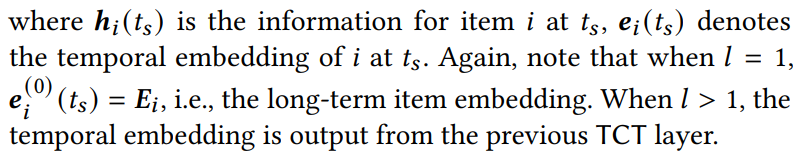
We also propagate temporal collaborative information from its neighbors.

We randomly sample 𝑆 different interactions of 𝑢 before time 𝑡 as



The input information at the 𝑙-th layer for each (𝑖, ) pair is denoted as

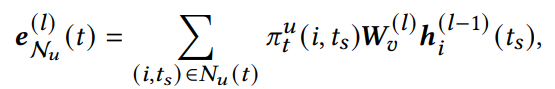


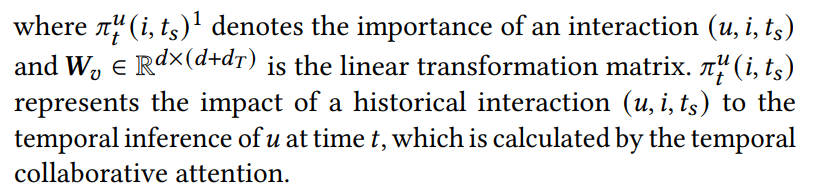


**Summary of 2.2.1:** After concatenating embedding of i at ts and the time encoding of time ts, we obtain the information for item i at time ts. This contains both the item info and time info.

**2.2.2 Information propagation**

We propagate the information of sampled neighbors to infer the temporal embeddings. We compute the linear combination of information from all samples as

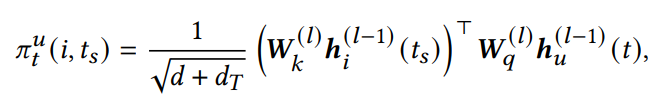


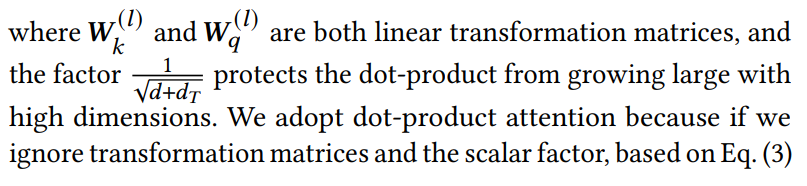


**Summary of 2.2.2:** The propagation is to infer the embedding from sampled neighbors of the graph. The info from each neighbor is weighted by learnable parameters. This simply aims at propagating the info from related nodes in the graph for further process.

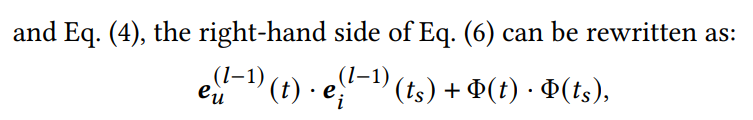
**2.2.3 Temporal collaborative attention**

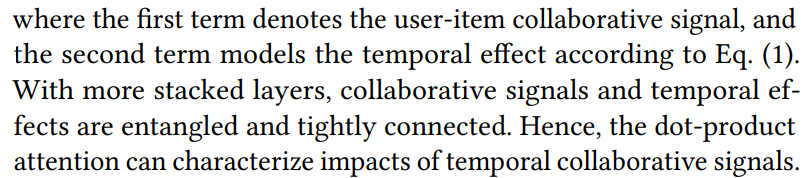
For temporal collaborative attention, the key question is how to measure the attention weights of both neighboring interactions and the temporal information on edges. We denote the attention scores as follows

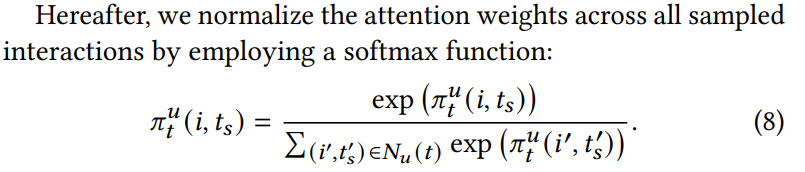
\



This can be interpreted in an alternative way. If we ignore transformation matrics, what the model is doing so far can be interpreted as



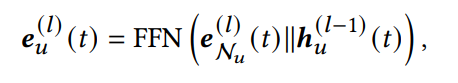


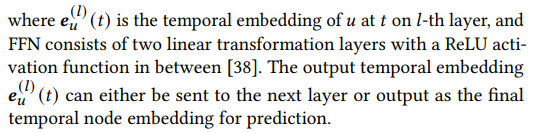


**Summary of 2.2.3:** This means that the user-item collaborative signal of the first term and the temporal effect of the second term are considered simultaneously by the model, which is one of the core claims of major challenges in the paper. The authors manage to entangle the two info with stacked layers since everytime we pass the overall obtained info via a layer, the two info will be calculated into a single representation of mixed signals. The following calculations are been easily introduced as follows: after getting the attention scores/weights, we use softmax function and follow the normal attention mechanism to multiply the softmaxed attention score (of query and key) with the value. Here, the query is the user info concatenated by the corresponding time info, while the key and value are both the info of sampled neighbors concatenated by the corresponding time. The interpretation is that we first calculate how we use the info from sampled neighbors to aggregate to get the latent representation for the prediction of next item for the user, and then we multiply this “weight factor” with the sampled neighbor information and aggreagate them together to consider all the impact from possibly related interaction. So far, this attention is still the common used one for most papers in the literature. The contribution here is that the author manage to consider temporal effect and use stacked layers to make sure both user/item info and temporal info are mixedly considered.

**2.2.4 Information Aggregation**

To aggregate the query information and neighbor information, we concatenate and use feed-forward neural network to get the output temporal embedding:

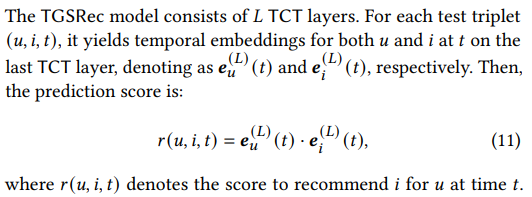




We can also use a similar feedforward network to calculate output item embedding.

**Summary of 2.2.4:** The feedforward network to calculate the final output of a layer is also commonly used in the attention-based sequential recommendation.

* 1. **Model prediction**

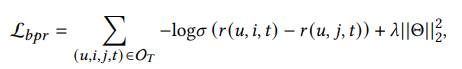
****

**Summary of 2.3:** The final model prediction is to multiply the output of item and temporal embedding to form the overall prediction. It means the probability/score to recommend item i for user u at time t. Still, so far we know that the major novelty is that the author consider the temporal effect at every single step even including the final model prediction step. The relative temporal effect is naturally considered in the attention mechanism as explained in 2.2.3 as we multiply the time enconding of time t and the times before t.

* 1. **Model optimization**

Two losses are used;

1. Pairwise BPR loss



1. Binary Cross-Entropy loss



**Summary of 2.4:** Both losses are regularized by L2 norm to prevent overfitting. The pairwise BPR loss wants to increase the probability of recommending the label item and decrease the probability of recommending false items in a straightforward way, as we can infer from the equation that minus log of sigmoid is a decreasing function. (If we want to make the loss decreases, we need to make the overall value in the bracket increases, which is to increase the probability/score of correct items and decrease the counterparts) The binary cross-entropy loss is also frequently used, which pushes the prediction distribution towards the correct item distribution to ensure the prediction will more likely be the label/correct item. Both losses make sense.

**Experiments:**

The authors compare the proposed model with baselines. The improvement is very obvious although the authors admit in their GitHub page that the data processing is wrong. Therefore, the experimental analysis is somewhat concerning. The authors analyze hyperparameter sensitivity and find that there are optimal hyperparameters existed for different datasets. They also find that the sequential patterns are important and the proposed embedding and attention components demonstrate the necessity to consider the collborative attention and temporal effect.