Experiment Summary

April 2023

1 TSNE

1.1 Purpose

This experiment aims to visualize the output of hidden layer and use different metrics to visualize the decision boundary including KNN, Decision Tree and SVM.

We input the hidden layer output into TSNE function to reduce their dimensions to 2 and normalize the result Then we plot these 2-dimension normalized vectors and use different machine learning algorithms (KNN, Decition Tree, SVM) to predict the boundary and visualize.

1.2 Script

python train_self_supervised.py –use_memory –prefix tgn-attn –n_runs 1 –tsne

1.3 Result

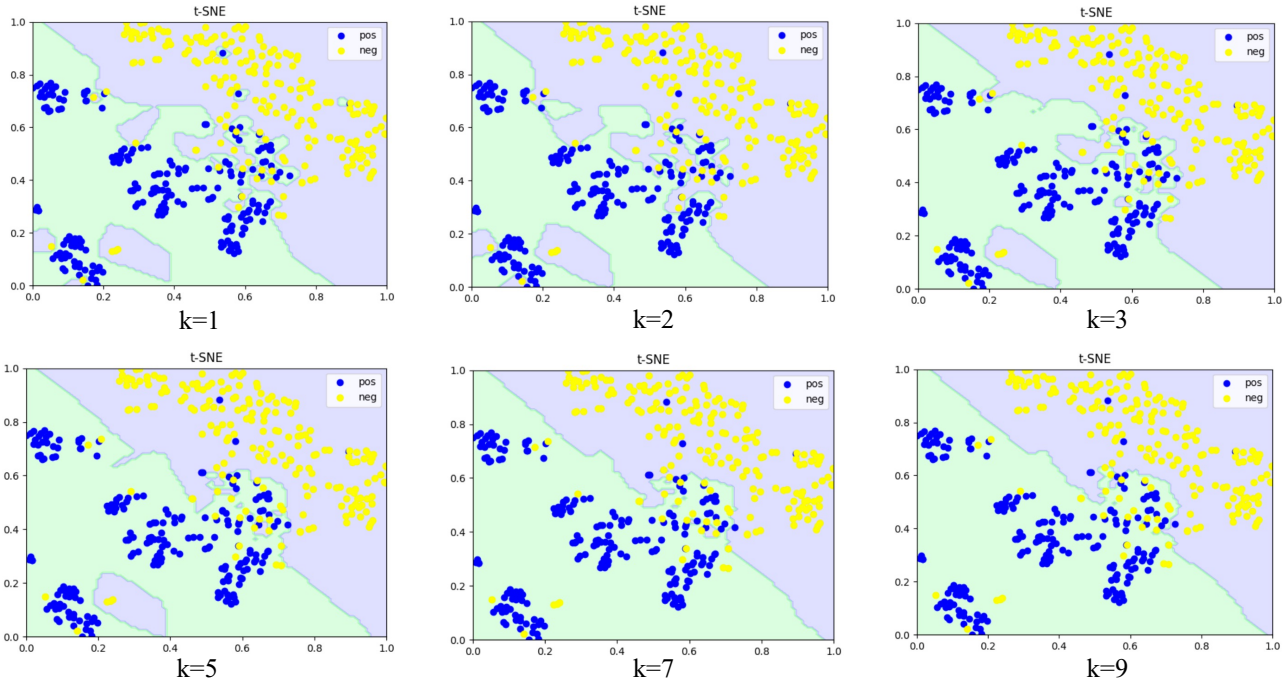

Figure 1: KNN

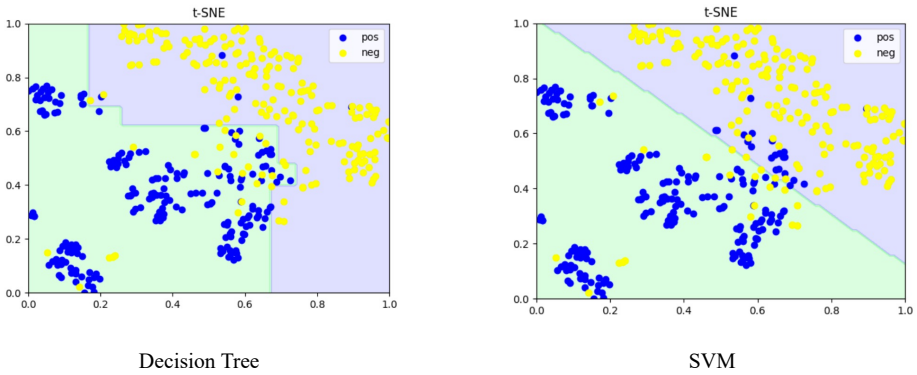

Figure 2: Decision Tree and SVM

2 Attention

2.1 Purpose

This experiment aims to visualize the attention weight which is obtained while aggregating nodes and their neighbours.

2.2 Script

python train_self_supervised.py –use_memory –prefix tgn-attn –n_runs 1 –visual_att

2.3 Result

Each node has 10 neighbors, so the x-axis represents the attention weight of neighbors of each node. The y-axis represents different nodes.

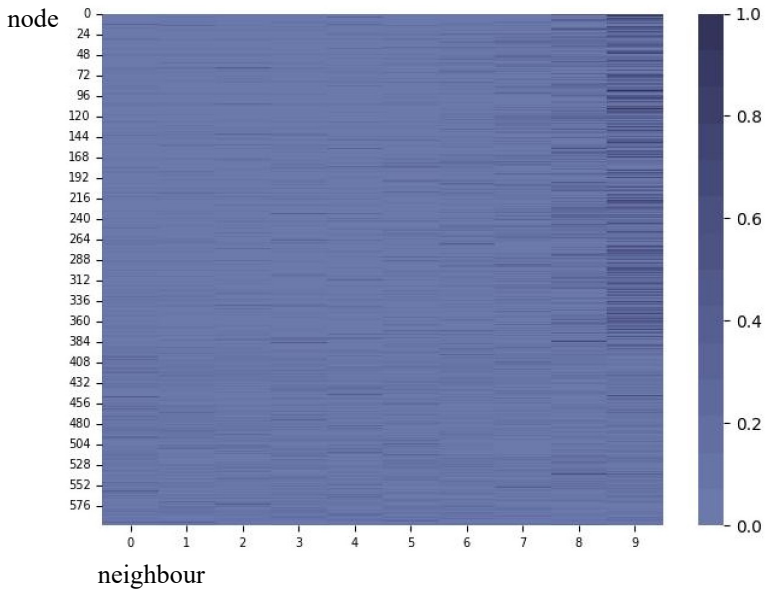

Figure 3: Visualization of Attention Weight

3 Define batch as the length of time interval

3.1 Purpose & Procedure

This experiment aims to explore whether using length of time interval as batch will affect the performance of model.

Instead of measuring batch size as number of events, define batch as the length of time interval. For example, suppose the training set is in the time interval [0,T], evenly divide [0,T] into **m** batch, then, the return batch are events in [0, T/m], [T/m, 2T/m],. . . ,[m-1 T/m, T].

3.2 Script

python train_self_supervised.py –use_memory –prefix tgn-attn –n_runs 1 –interval_as_batch

3.3 Result

m	Epoch Time	AP (Transductive)	AP (Inductive)
5000	103.24	0.9848	0.9786
10000	74.67	0.9819	0.9760
20000	59.85	0.9797	0.9718
40000	30.55	0.9768	0.9716

4 The Effect of Degree

4.1 Purpose & Procedure

This experiment aims to explore whether the degree of node would affect the performance.

For nodes of small degree, we extract the first 20 vertices with smallest degree. For nodes of large degree, we extract the last 20 vertices with smallest degree. For nodes of average degree, we extract the vertices with degree in [avg_degree-1, avg_degree+1]. Then we test each vertex's accuracy with respect to the rest of vertices.

4.2 Script

python train_self_supervised.py –use_memory –prefix tgn-attn –n_runs 1 –test_degree

4.3 Result

Degree	Test AP	Test AUC	Pos Acc	Neg Acc
small	0.7390	0.7275	0.4	0.9
average	0.8968	0.8821	0.665	0.9
large	0.9533	0.9464	0.795	0.96

5 Time-series Data Smoothing

5.1 Purpose & Procedure

This experiment aims to explore whether smoothing the time-series data can improve the performance of TGN.

We use two methods to smooth the memory vector. The first one is Backward Moving Average (see Figure 4 for detailed).

Figure 4: Backward Moving Average

The second one is Exponential Smoothing. Its fomula is shown below:

$$y_0^{'} = y_0 \tag{1}$$

$$y_{t+1}^{'} = \alpha * y_t + (1 - \alpha) * y_t^{'} \tag{2}$$

where α is a smoothing factor.

5.2 Scripts

1. **Back Moving Average:** python train_self_supervised.py –use_memory –prefix tgn-attn –n_runs 1 –back_smooth
2. **Exponential Smoothing:** python train_self_supervised.py –use_memory –prefix tgn-attn –n_runs 1 –exp_smooth

5.3 Result

Smoothing	Transductive AUC	Transductive AP	Inductive AUC	Inductive AP
No Smoothing	0.9781	0.9793	0.9729	0.9744
Backward Moving Average	0.9670	0.9709	0.9626	0.9660
Exponential Smoothing	0.9583	0.9646	0.9570	0.9623

6 Visualization of Memory Vector

6.1 Purpose & Procedure

This experiment aims to explore the statistic of memory vector and cosine similarity.

We append memory in each iteration to a list and after an epoch, we compute their statistic and visualize them. In terms of computing cosine similarity, we compute the similarity between memory in t iteration and $t + 1$ iteration, and append the computed similarity to a list, then plot it.

6.2 Script

python train_self_supervised.py –use_memory –prefix tgn-attn –n_runs 1 –compute_statistic

6.3 Result

7 Visualization of Mean of Memory Vector (Improved)

7.1 Purpose & Procedure

This experiment re-compute the mean of memory vector and visualize it. For a memory vector of dimension [9228, 172], we compute the mean of each column, then we will obtain a vector of [1, 172]. Afterwards, we use

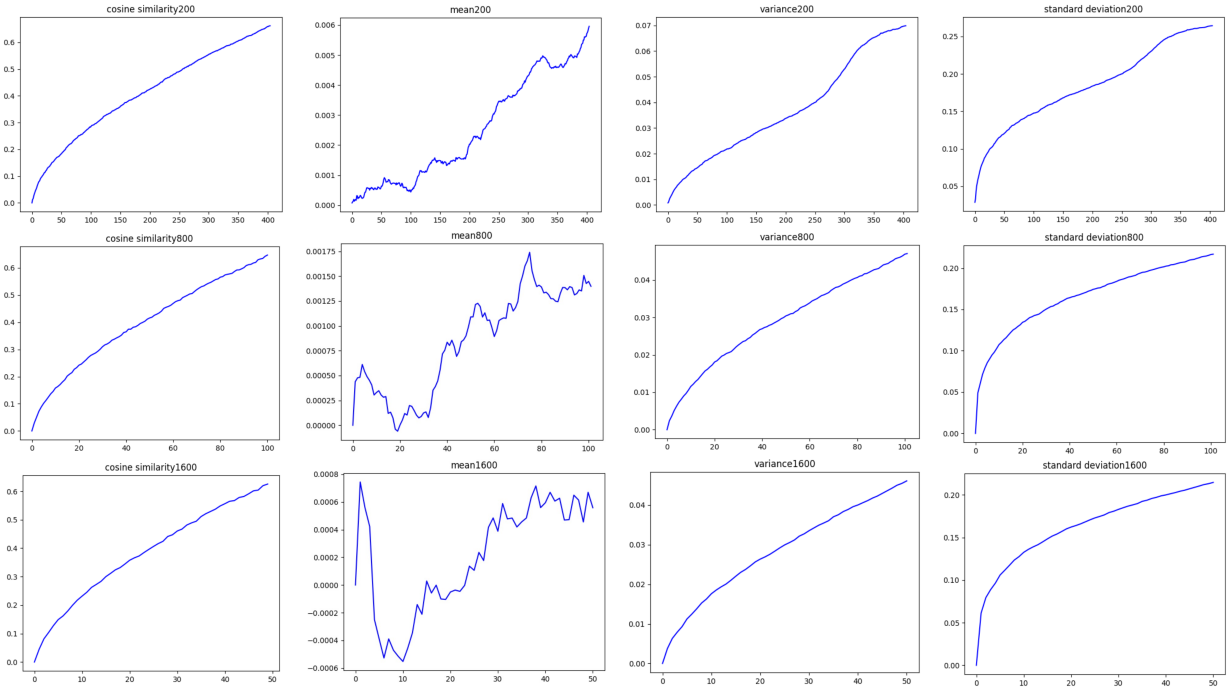

Figure 5: Visualization of Statistic of Memory Vector

t-SNE to reduce the dimension of mean value, and normalize it to [0,1].

7.2 Script

python train_self_supervised.py –use_memory –prefix tgn-attn –n_runs 1 –compute_statistic

7.3 Result

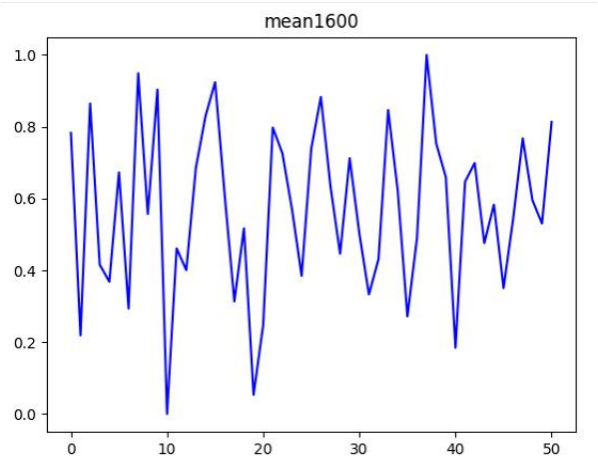

Figure 6: Visualization of Mean of Memory Vector (re-compute)