Experiment Summary

December 2022

1 TGN

1.1 Data Preprocess

Create Train data:

1. Random choose 10% data as new nodes which appears at test time (to test inductiveness) and remove them
2. Timestamp <= val timestamp

Create Validation data:

1. val timestamp <Timestamp <= test timestamp

Create Test data:

1. Timestamp >test timestamp

Create Unseen Validation Data:

1. val timestamp <Timestamp <= test timestamp
2. In 10% data that is distracted from train data

Create Unseen Test Data:

1. Timestamp >test timestamp
2. In 10% data that is distracted from train data

1.2 Training Procedure

1. Sampling batch according to the time sequence
2. The current message is calculated using the raw messages of the previous batch to avoid information leakage.
3. Update the states and calculate the node embeddings.
4. Transfer node embeddings to edge probabilities
5. Calculate the loss and update TGN

1.3 Evaluation

AP	Precision	F1	Accuracy	Specificity	Sensitivity
0.9846	0.9852	0.9329	0.9326	0.9282	0.9370

2 Influence of Batch Size

2.1 Purpose

This experiment aims to explore the relationship of batch size and training time.

2.2 Script

python train_self_supervised.py –use_memory –prefix tgn-attn –n_runs 10 –bs 200

2.3 Result

bs	Epoch Time	AP (Transductive)	AP (Inductive)
10	788.04	0.9821	0.9729
50	278.00	0.9847	0.9774
100	110.89	0.9825	0.9754
200	62.72	0.9846	0.9781
400	39.84	0.9825	0.9763
500	37.98	0.9787	0.9731
1000	21.94	0.9818	0.9760
2000	18.55	0.9778	0.9724
4000	14.50	0.9750	0.9691

2.4 Implication

From the result shown above, we can see that the training time increases as the batch size increases, that is say large batch size can save the time in training procedure. In terms of AP score, there is a slight drop in performance of models due to the increase of batch size (from bs 200 to bs 4000).

3 Influence of Memory

3.1 Purpose

This experiment aims to explore the influence of memory used in TGN.

3.2 Script

with memory: python train_self_supervised.py –use_memory –prefix tgn-attn –n_runs 10
without memory python train_self_supervised.py –prefix tgn-attn –n_runs 10

3.3 Result

model	Test AP (Transductive)	Test AP (Inductive)
TGN (w/ memory)	0.9846	0.9781
TGN (w/o memory)	0.9479	0.9468
TGN (zero-shot)	0.6337	0.6408

3.4 Implication

From the result shown above, we can see that model with memory has better performance than that without memory. This implicates that memory does contribute to the performance of model.

The result of zero-shot experiment implicates that the model's performance improves dramatically after it has been trained.

4 Sample

4.1 Purpose

This experiment aims to explore whether the reduction in data volume substantially affects the performance of the model and also explore the influence of distributed training.

4.2 Procedure

Partition the dataset into two datasets D1, D2 by sampling two data points from every 2/4/8/16/32 data points.

Algorithm 1

$new\_node \leftarrow random\_sample(dataset)$
$val\_timestamp \leftarrow 0.7$
$test\_timestamp \leftarrow 0.85$
$train\_set \leftarrow datasets[timestamp < val\_timestamp] - new\_node$
$val\_set \leftarrow datasets[val\_timestamp < timestamp < test\_timestamp]$
$test\_set \leftarrow datasets[timestamp > test\_timestamp]$
$new\_val\_set \leftarrow val\_set[edges\ contain\ new\_node]$
$new\_test\_set \leftarrow test\_set[edges\ contain\ new\_node]$

$s \leftarrow [4, 8, 16, 32]$
$n \leftarrow len(train\_set)/s$
$random\_indexs \leftarrow sorted(random.sample(range(0, s), 2))$
for i in range(n) **do**
    $sample\_train\_set.extend(train\_set[i * s : (i + 1) * s])[random\_indexs]$
end for

Rerun the training and average the parameters of M1 and M2 for every iteration, record the performance of resulting model. (combine)

Algorithm 2

$TGN1 \leftarrow initial\_model()$
$TGN2 \leftarrow initial\_model()$

for each epoch **do**
    $p^1_{pos}, p^1_{neg} \leftarrow TGN1(train\_set1)$
    $p^2_{pos}, p^2_{neg} \leftarrow TGN2(train\_set2)$
    $l_1 \leftarrow BCE(p^1_{pos}, p^1_{neg})$
    $l_2 \leftarrow BCE(p^2_{pos}, p^2_{neg})$
    $l_1.backward()$
    $l_2.backward()$

    $TGN1.load\_state\_dict((TGN1.state\_dict() + TGN2.state\_dict())/2)$
    $TGN2.load\_state\_dict(deepcopy(TGN1.state\_dict()))$
end for

4.3 Script

D1:
python train_tbatch.py –use_memory –prefix tgn-attn –sample_type uniform –skip 16 –id 0
D2:
python train_tbatch.py –use_memory –prefix tgn-attn –sample_type uniform –skip 16 –id 1
combine:
python train_sample.py –use_memory –prefix tgn-attn –sample_type uniform –skip 16

4.4 Result

4.4.1 Wikipedia

Sample	Test AP (Transductive)		Test AP (Inductive)	
		w/o memory		w/o memory
4(0)	0.9419	0.9820	0.9417	0.9752
4(1)	0.9415	0.9823	0.9411	0.9769
4(average)	0.9167	0.9783	0.9163	0.9715
4(combine)	0.9477	0.9843	0.9474	0.9791
8(0)	0.9395	0.9728	0.9384	0.9666
8(1)	0.9371	0.9759	0.9359	0.9675
8(average)	0.9277	0.9739	0.9260	0.9686
8(combine)	0.9432	0.9763	0.9429	0.9715
16(0)	0.9244	0.9615	0.9235	0.9557
16(1)	0.9183	0.9601	0.9181	0.9556
16(average)	0.9214	0.9656	0.9194	0.9618
16(combine)	0.9161	0.9570	0.9152	0.9541
32(0)	0.9024	0.9384	0.8997	0.9371
32(1)	0.9024	0.9418	0.8946	0.9402
32(average)	0.9047	0.9441	0.9011	0.9418
32(combine)	0.9181	0.9235	0.9162	0.9234

4.4.2 Reddit

Sample	Test AP (Transductive)		Test AP (Inductive)	
		w/o memory		w/o memory
4(0)	0.9538	0.9849	0.9318	0.9727
4(1)	0.9525	0.9828	0.9299	0.9707
4(average)	0.9378	0.9808	0.9057	0.9669
4(combine)	0.9546	0.9870	0.9325	0.9754
8(0)	0.9418	0.9822	0.9165	0.9696
8(1)	0.9437	0.9816	0.9189	0.9683
8(average)	0.9292	0.9735	0.8951	0.9606
8(combine)	0.9483	0.9830	0.9229	0.9702
16(0)	0.9346	0.9735	0.9067	0.9607
16(1)	0.9338	0.9758	0.9078	0.9636
16(average)	0.9208	0.9705	0.8862	0.9536
16(combine)	0.9307	0.9678	0.9067	0.9475
32(0)	0.9174	0.9599	0.8862	0.9425
32(1)	0.9171	0.9655	0.8858	0.9478
32(average)	0.9114	0.9637	0.8753	0.9463
32(combine)	0.9204	0.9623	0.8888	0.9416

4.4.3 Enron-Email

Sample	Test AP (Transductive)		Test AP (Inductive)	
	w/o memory		w/o memory	
4(0)	0.8901	0.9684	0.7819	0.9118
4(1)	0.8931	0.9533	0.7834	0.8896
4(average)	0.8762	0.9578	0.7611	0.8926
4(combine)	0.8910	0.9712	0.7803	0.9198
8(0)	0.8824	0.9450	0.7697	0.8726
8(1)	0.8904	0.9575	0.7794	0.8932
8(average)	0.8723	0.9545	0.7578	0.8839
8(combine)	0.8813	0.9705	0.7708	0.9271
16(0)	0.8808	0.9474	0.7748	0.8734
16(1)	0.8829	0.9521	0.7739	0.8765
16(average)	0.8576	0.9523	0.7366	0.8725
16(combine)	0.8872	0.9393	0.7823	0.8394
32(0)	0.8686	0.9428	0.7534	0.8567
32(1)	0.8702	0.9446	0.7553	0.8578
32(average)	0.8659	0.9467	0.7548	0.8613
32(combine)	0.8404	0.9492	0.7533	0.8692

4.5 Implication

From the result shown above, the reduction of train set reduces the accuracy of the model. And the performance of model under distributed training is close to original model.

5 t-batch

5.1 Purpose

This experiment aims to explore whether the reduction in data volume substantially affects the performance of the model and the training time.

5.2 Procedure

Divide the dataset into m (32) t-batch, for each epoch, randomly sampled k = {1,2,4,8,16} t-batch.

Algorithm 3

$s \leftarrow [1, 2, 4, 8, 16]$
$m \leftarrow 32$
$n \leftarrow len(train\_set)/m$
$random\_indexs\_list \leftarrow sorted(random.sample(range(0, m), s))$

for i in random_indexs_list **do**
    $sample\_train\_set.extend(train\_set[i * n : (i + 1) * n])$
end for

5.3 Script

python train_tbatch.py –use_memory –prefix tgn-attn –sample_type t-batch –skip 4 –id 0

5.4 Result

5.4.1 Wikipedia

Sample	Test AP (Transductive)		Test AP (Inductive)		Time	
	w/o memory	w/ memory	w/o memory	w/ memory	w/o memory	w/ memory

1	0.9286	0.9277	0.9272	0.9234	0.2523	1.4092
2	0.9200	0.9224	0.9197	0.9189	0.4866	2.4025
4	0.9211	0.9614	0.9186	0.9539	1.0660	5.5749
8	0.9362	0.9733	0.9335	0.9674	1.9456	12.6140
16	0.9465	0.9806	0.9461	0.9750	3.6586	22.5398

5.4.2 Reddit

Sample	**Test AP (Transductive)**		**Test AP (Inductive)**		**Time**	
	w/o memory	w/ memory	w/o memory	w/ memory	w/o memory	w/ memory
1	0.9282	0.9637	0.9006	0.9213	1.6961	5.7906
2	0.9345	0.9689	0.9097	0.9480	3.5920	15.2807
4	0.9407	0.9789	0.9158	0.9629	7.7395	36.8422
8	0.9484	0.9833	0.9254	0.9710	14.8441	76.5402
16	0.9551	0.9848	0.9332	0.9714	27.9549	159.3442

5.4.3 Enron-Email

Sample	**Test AP (Transductive)**		**Test AP (Inductive)**		**Time**	
	w/o memory	w/ memory	w/o memory	w/ memory	w/o memory	w/ memory
1	0.8494	0.9416	0.7612	0.8560	0.9507	4.4313
2	0.8593	0.9481	0.7821	0.8717	2.0002	9.9783
4	0.8499	0.9526	0.7600	0.8756	4.3263	23.9450
8	0.8908	0.9431	0.7971	0.8724	8.0765	55.4182
16	0.8890	0.9655	0.7884	0.9087	15.0158	131.7370

5.5 Implication

From the result shown above, the reduction of train set reduces the accuracy of the model and save the time in training procedure. However, the reduction of accuracy is slight compared to the reduction of time. We also figure out that the usage of memory increases the training time dramatically.

6 Exploration of Always Predicted Wrongly Data

6.1 Purpose

This experiment aims to figure out the characteristic of test data that is always predicted wrongly and compare them with those always predicted correctly.

6.2 Result

The table shown below shows the number of test data always predicted wrong (or correct) and the number of test data not only predicted wrong (or correct) but also has historical event in training procedure.

Dataset			**# wrong/correct**	**# wrong/correct & appear in train**	**%**
Wikipedia	positive	wrong	569	60	10.5%
		correct	21274	9408	44.2%
	negative	wrong	926	219	23.7%
		correct	20693	8904	43.03%
Reddit	positive	wrong	3637	821	22.6%
		correct	93828	68153	72.64%
	negative	wrong	4028	2226	55.3%
		correct	92488	65564	70.89%

For an event and its corresponding historical event, we calculate the time span between each event and compute their average. We divide the time span into $(0,10^4)$, $[10^4,10^5)$, $[10^5,10^6)$ and $[10^6,10^5)$, and count the number of events whose average time span falls within a certain interval. The following two tables show the results of data predicted wrongly and correctly respectively.

Wrong

Dataset		**Mean time span** $(0,10^4)$	$[10^4, 10^5)$	$[10^5, 10^6)$	$[10^6, 10^7)$
Wikipedia	positive	0%	18.33%	58.33%	23.33%
	negative	1.83%	30.59%	54.34%	13.24%
Reddit	positive	0.00%	2.19%	66.50%	31.18%
	negative	1.21%	28.08%	60.38%	10.33%

Correct

Dataset		**Mean time span** $(0,10^4)$	$[10^4, 10^5)$	$[10^5, 10^6)$	$[10^6, 10^7)$
Wikipedia	positive	7.13%	55.36%	34.61%	2.90%
	negative	7.29%	55.57%	34.15%	2.99%
Reddit	positive	10.21%	52.39%	34.37%	3.04%
	negative	10.51%	52.52%	33.74%	3.22%

6.3 Implication

Those test data always predicted correctly is more likely to occur in training procedure than those always predicted wrongly.

From the last two tables, the majority of those always predicted correctly has the average time span of $[10^4, 10^5)$ while the majority of those always predicted wrongly has the average time span of $[10^5, 10^6)$. From this result, we can infer that if an event occurs frequently, it is likely to be predicted correctly.