PreTraM: Self-Supervised Pre-training via Connecting Trajectory and Map Supplementary Material

Chenfeng Xu⋆[1], Tian Li⋆[2], Chen Tang⋆⋆[1], Lingfeng Sun[1], Kurt Keutzer[1], Masayoshi Tomizuka[1], Alireza Fathi[3], and Wei Zhan[1]

[1] University of California, Berkeley
[2] University of California, San Diego
[3] Google Research

1 Data Efficiency

In section 4.3, we demonstrate with experiments that PreTraM boosts data efficiency of the prediction model. Apart from the results shown there, we perform experiments at other percentages of trajectory data, ranging from 80%, 70%, 60% to 20% and 10%. Besides, to ensure reliable results for lower percentages including 20% and 10%, we repeat experiments for 5 and 10 times respectively. For the other settings, we repeat 3 times as in section 4.3. We report the mean performance as well as the variance in table 1. Note that these experiments are all based on AgentFormer with ResNet18.

On this full set of experiment results, we observe that across all settings of experiments, PreTraM mitigates prediction error and significantly reduces variance of performance. For the mean value of using different random seeds with using 70% data, PreTraM-AgentFormer gives 2.451 ADE-5 and 5.343 FDE-5, which outperforms the baseline with 100% data. Note that for one specific experiment with 10 % data, we observe that PreTraM-Agentformer can achieve 0.32 ADE-5 and 0.77 FDE-5 improvement, as reported in the main text.

Therefore, we conclude that PreTraM indeed enhances data efficiency.

2 Ablation Study on Augmentation Operation in MCL

MCL uses dropout as minimal augmentation. In section 3.2, we briefly discuss why dropout works better for map representation learning (MCL) than the commonly used data augmentation applied on contrastive learning in computer vision.

To verify the argument, we conduct experiments with different augmentation operations including rotation, flip, color jitter, and gaussian noise. The experiments are based on PreTraM-AgentFormer with ResNet18, and the results are reported in Table 2. We observe that MCL via any of the alternative

⋆ Equal contribution
⋆⋆ Corresponding author

data augmentations even deteriorates the performance of AgentFormer, rather than boosting it. They are worse than the baseline AgentFormer by at least 0.019 ADE-5 and 0.076 FDE-5. In contrast, PreTraM with MCL via dropout drastically improves the baseline by 0.1 ADE-5 and 0.218 FDE-5. This sharp contrast indicates that conventional augmentations do destroy the semantic and the topology in the HD-map, while dropout acting as minimal augmentation enhances the representation learning of HD-maps and consequently promotes TMCL.

3 Implementation Details

All of our experiments are developed based on Pytorch and conducted on Intel Haswell CPU platform with one single V100 GPU.

Reproduced AgentFormer. We refer to the official code of AgentFormer to reproduce a new version that supports parallel computing. Specifically, original AgentFormer only uses one single scene in each iteration due to the variant agent number in different scenes. Therefore, to support parallel computing, we pad the agent number to 20, and the padding agents will be masked in the models as well as during loss computation so that the whole process is not influenced by the padding agents. On the other hand, we do not use the step decay learning rate scheduler in the original AgentFormer, but use the linear scheduler with warmup provided by HuggingFace [2], which is commonly used in language-targeted transformer models [1]. During training, we set the batch size as 8, the initial learning rate as 10^{-4}, the warmup rate as 0.1, and we train the model with Adam optimizer for 100 epochs. During inference, we keep the same setting as the original AgentFormer.

PreTraM on top of AgentFormer and Trajectron++. We simply conduct PreTraM on map encoder and past encoder of AgentFormer and Trajectron++ in the pre-training phase. Trajectron++ is slightly different from AgentFormer. The past feature encoding in Trajectron++ includes past trajectory en-

Table 1. Experiment results with part of the trajectory data. For percentages above 30%, experiments are repeated 3 times, while for 20% and 10% data, the experiments are repeated 5 and 10 times respectively.

Percentage of	AgentFormer		PreTraM-AgentFormer	
Trajectory	ADE-5	FDE-5	ADE-5	FDE-5
100%	2.472	5.401	2.372	5.183
80%	2.556±0.035	5.599±0.071	2.439±0.017	5.329±0.059
70%	2.570±0.038	5.609±0.075	2.451±0.030	5.343±0.072
60%	2.586±0.051	5.621±0.123	2.524±0.031	5.477±0.103
50%	2.707±0.037	5.889±0.089	2.599±0.016	5.652±0.030
40%	2.780±0.079	6.055±0.195	2.721±0.018	5.888±0.081
30%	3.055±0.061	6.672±0.101	2.890±0.065	6.281±0.140
20%	3.388±0.115	7.339±0.256	3.180±0.089	6.840±0.181
10%	3.818±0.141	8.038±0.292	3.618±0.114	7.607±0.229

Table 2. We use different augmentation operations to conduct MCL for comparison. MCL via dropout is our proposed method.

PreTraM-AgentFormer with different MCL	ADE-5	FDE-5
AgentFormer (baseline)	2.472	5.401
PreTraM-AgentFormer (MCL via dropout)	2.372(-0.100)	5.183(-0.218)
PreTraM-AgentFormer (MCL via random rotation)	2.508(+0.036)	5.477(+0.076)
PreTraM-AgentFormer (MCL via random flip)	2.491(+0.019)	5.499(+0.098)
PreTraM-AgentFormer (MCL via color jitter)	2.505(+0.033)	5.505(+0.104)
PreTraM-AgentFormer (MCL via gaussian noise)	2.541(+0.069)	5.601(+0.200)

coding and edge encoding, and we concatenate them as the trajectory features in PreTraM.

We avoid taking great pains to tune hyperparameters of the pre-training phase and use the same training recipe as the finetuning phase. The only difference of our experiment on AgentFormer and Trajectron++ is that we pre-train the latter with fewer epochs (20 epochs for AgentFormer and 5 epochs for Trajectron++). Besides, we set λ in equation (4) to 1. These choices, though simple, turn out to work well for PreTraM.

We provide a numpy-like pseudocode of PreTraM, as shown below.

```
# past_trajectory_encoder: Transformer Encoder in AgentFormer
    or LSTM in Trajectron++
# map_encoder: Map_CNN or ResNet with dropout
# s[N_traj, T, D]: input trajectories
# m[N_traj, C, C, 3]: agent-centric map patches paired with s
# m_mcl[N_map, C, C, 3]: trajectory-decoupled maps
# W_traj[d_s, d_e]: learned projection of trajectory to embed
# W_map[d_m, d_e]: learned projection of map to embed (TMCL)
# W_mcl[d_m, d_e]: learned projection of map to embed (MCL)
# t_traj: learned temperature parameter for TMCL
# t_map: learned temperature parameter for MCL
# lamda: hyperparameter for balancing MCL and TMCL

####################
## For TMCL
####################
# extract feature
h_traj_timed = past_trajectory_encoder(s) # [N_traj, T, d_t]
h_map = map_encoder(m) # [N_traj, d_m]

# mean pooling over time dimension
h_traj = h_traj_timed.mean(axis=1) # [N_traj, d_t]

# linear projection to embedding space [N_traj, d_e]
h_traj_norm = l2_normalize(np.dot(h_traj, W_traj), axis=1)
h_map_norm = l2_normalize(np.dot(h_map, W_map), axis=1)

```

```
# compute pairwise similarities [N_traj, N_traj]
logits_tmcl = np.dot(h_traj_norm, h_map_norm.T) / t_traj

# symmetric loss
labels = np.arange(N_traj)
loss_traj = cross_entropy_loss(logits_tmcl, labels)
loss_map = cross_entropy_loss(logits_tmcl.T, labels)
loss_TMCL = (loss_traj + loss_map)/2
####################
## For MCL
####################
# extract feature [N_map, d_m]
h_dp1 = map_encoder(m_mcl)
h_dp2 = map_encoder(m_mcl) # different dropout mask

# shared linear projection [N_map, d_e]
h_dp1_norm = l2_normalize(np.dot(h_dp1, W_mcl), axis=1)
h_dp2_norm = l2_normalize(np.dot(h_dp2, W_mcl), axis=1)

# compute pairwise similarities [N_map, N_map]
logits_mcl = np.dot(h_dp1_norm, h_dp2_norm.T) / t_map

# contrastive loss
labels = np.arange(N_map)
loss_MCL = cross_entropy_loss(logits_mcl, labels)

# Total loss
loss_total = loss_TMCL + lamda * loss_MCL
```

Listing 1.1. Numpy-like pseudocode of PreTraM.

References

1. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers). pp. 4171–4186. Association for Computational Linguistics, Minneapolis, Minnesota (Jun 2019). https://doi.org/10.18653/v1/N19-1423, https://aclanthology.org/N19-1423 2
2. Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T.L., Gugger, S., Drame, M., Lhoest, Q., Rush, A.M.: Transformers: State-of-the-art natural language processing. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. pp. 38–45. Association for Computational Linguistics, Online (Oct 2020), https://www.aclweb.org/anthology/2020.emnlp-demos.6 2