Scalable Spatiotemporal Graph Neural Networks

Andrea Cini*[1] Ivan Marisca*[1] Filippo Maria Bianchi[23] Cesare Alippi[14]

[1]The Swiss AI Lab IDSIA, Università della Svizzera italiana [2]UiT the Arctic University of Norway
[3]NORCE Norwegian Research Centre [4]Politecnico di Milano *Equal contribution

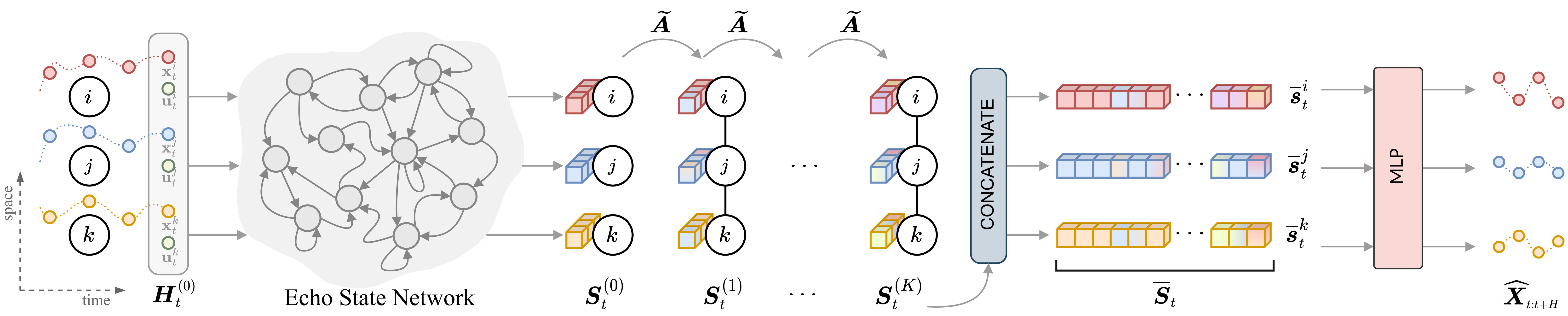


Figure 1. Scalable Graph Predictor

SGP: Scalable Graph Predictor

A novel approach based on an **encode-decoder** architecture with a **training-free spatiotemporal encoding** scheme and where the only learned parameters are in the **node-level trainable decoder** (an MLP).

- ▶ Representations for each point in time and space can be **precomputed**.
- ▶ The decoder can be trained by **sampling uniformly time and space**.
- ▶ This enables **scalability at training time**: $\mathcal{O}(ET) \rightarrow \mathcal{O}(1)$.

Motivation

Scalablity in Spatiotemporal GNNs is challenging to achieve due to the large amounts of input data.

- ▶ Standard STGNNs have **time and space complexity** of $\mathcal{O}(ET)$.
- ▶ If an attention component is included, **complexity becomes quadratic** in either time or space.
- ▶ **Subsampling strategies** can break long-range spatiotemporal dependencies and are prone to failure.

Spatiotemporal Encoder

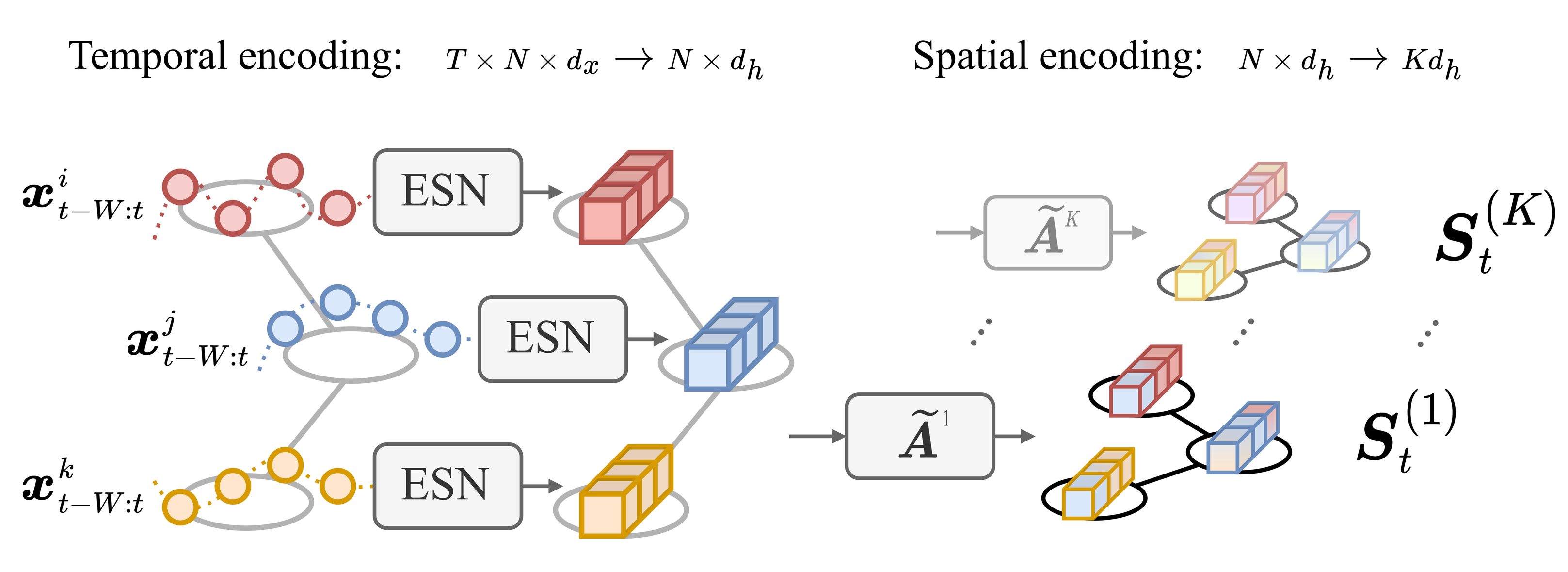


Figure 2. Spatiotemporal encoding scheme.

The spatiotemporal encoder relies on two modules.

- ▶ A randomized recurrent neural network for encoding sequences.
- ▶ A propagation process through the graph structure.

TEMPORAL ENCODING

We adopt the **deep echo state network** framework [1].

$$\boldsymbol{h}_t^{i,(0)} = \left[\boldsymbol{x}_t^i \| \boldsymbol{u}_t^i\right],$$

$$\hat{\boldsymbol{h}}_t^{i,(l)} = \tanh\left(\boldsymbol{W}_u^{(l)}\boldsymbol{h}_t^{i,(l-1)} + \boldsymbol{W}_h^{(l)}\boldsymbol{h}_{t-1}^{i,(l)} + \boldsymbol{b}^{(l)}\right),$$

$$\boldsymbol{h}_t^{i,(l)} = (1-\gamma_l)\boldsymbol{h}_{t-1}^{i,(l)} + \gamma_l \hat{\boldsymbol{h}}_t^{i,(l)}. \qquad l = 1, \dots, L$$

- ▶ Weight matrices are **randomly generated**.
- ▶ Proper normalization makes the system **stable**.
- ▶ The concatenated multilayer state representations **encode rich, multi-scale, dynamics**

$$\overline{\boldsymbol{H}}_t = \left(\boldsymbol{H}_t^{(0)} \| \boldsymbol{H}_t^{(1)} \| \dots \| \boldsymbol{H}_t^{(L)}\right).$$

SPATIAL PROPAGATION

The extracted temporal encodings are propagated by using **powers of a graph shift operator**.

$$\boldsymbol{S}_t^{(0)} = \overline{\boldsymbol{H}}_t = \left(\boldsymbol{H}_t^{(0)} \| \boldsymbol{H}_t^{(1)} \| \dots \| \boldsymbol{H}_t^{(L)}\right),$$

$$\boldsymbol{S}_t^{(k)} = \widetilde{\boldsymbol{A}}\boldsymbol{S}_t^{(k-1)} = \left(\widetilde{\boldsymbol{A}}^k\boldsymbol{H}_t^{(0)} \| \widetilde{\boldsymbol{A}}^k\boldsymbol{H}_t^{(1)} \| \dots \| \widetilde{\boldsymbol{A}}^k\boldsymbol{H}_t^{(L)}\right),$$

$$\overline{\boldsymbol{S}}_t = \left(\boldsymbol{S}_t^{(0)} \| \boldsymbol{S}_t^{(1)} \| \dots \| \boldsymbol{S}_t^{(K)}\right).$$

- ▶ This can be done either **recursively** or **in parallel**.

Multi-Scale Decoder

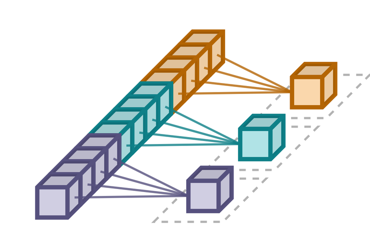
Could be a standard MLP, however, we can exploit the structure of the embedding.

- ▶ We make the connectivity of the first MLP layer sparse.

$$\boldsymbol{Z}_t^{(k)} = \sigma\left(\widetilde{\boldsymbol{A}}^k\boldsymbol{H}_t^{(0)}\boldsymbol{\Theta}_k^{(0)} \| \dots \| \widetilde{\boldsymbol{A}}^k\boldsymbol{H}_t^{(L)}\boldsymbol{\Theta}_k^{(L)}\right)$$

$$= \sigma\left(\boldsymbol{S}_t^{(k)}\begin{bmatrix}\boldsymbol{\Theta}_k^{(0)} & & \boldsymbol{0}\\ & \ddots & \\ \boldsymbol{0} & & \boldsymbol{\Theta}_k^{(L)}\end{bmatrix}\right),$$

$$\overline{\boldsymbol{Z}}_t = \left(\boldsymbol{Z}_t^{(0)} \| \boldsymbol{Z}_t^{(1)} \| \dots \| \boldsymbol{Z}_t^{(K)}\right).$$



- ▶ Standard fully connected layers can then be used to obtain predictions.
- ▶ Trained by sampling minibatches of representations over time and space as if they were iid.

Some Results

We tested the scalability of the approach by imposing a limit on training time (1 hour) and capping GPU memory utilization (12 GB).

		Prediction error (MAE)			Resource utilization		
		30 mins	7 hours 30 mins	11 hours	Batch/s	Memory	Batch size
PV-US 100	DCRNN	1.39 ± 0.09	3.34 ± 0.22	**3.54 ± 0.48**	2.04 ± 0.01	9.63 GB	2
	GWNet	1.45 ± 0.13	5.09 ± 0.63	5.26 ± 1.34	2.01 ± 0.02	11.64 GB	2
	GatedGN	1.33 ± 0.08	**2.94 ± 0.05**	**3.12 ± 0.14**	8.41 ± 0.09	11.46 GB	5
	SGP	**1.09 ± 0.01**	**3.14 ± 0.21**	**3.16 ± 0.19**	**116.58 ± 8.74**	**2.21 GB**	4096
PV-US Full	DCRNN	1.59 ± 0.17	4.10 ± 0.27	4.93 ± 0.60	1.37 ± 0.00	11.59 GB	1*
	GWNet	1.65 ± 0.23	6.93 ± 0.58	7.93 ± 0.17	0.77 ± 0.00	11.35 GB	2
	GatedGN	1.61 ± 0.06	3.25 ± 0.04	**3.04 ± 0.05**	8.83 ± 0.10	11.14 GB	1*
	SGP	**1.09 ± 0.00**	**3.06 ± 0.11**	**3.13 ± 0.13**	**118.64 ± 8.35**	**2.21 GB**	4096

Table 1. Results on large-scale datasets.

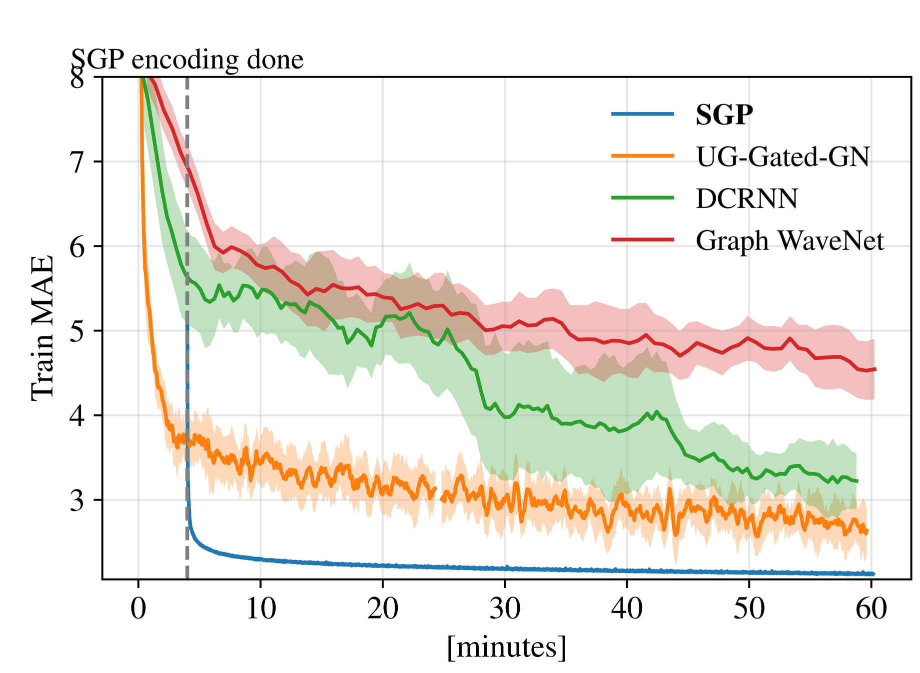


Figure 3. Learning curve on PV-US.

References

[1] C. Gallicchio, A. Micheli, and L. Pedrelli. Deep reservoir computing: A critical experimental analysis. *Neurocomputing*, 268: 87–99, 2017.

Our library for spatiotemporal data processing:

TorchSpatiotemporal/tsl