Graph Networks for Wind Nowcasting

EE-452. Graph Networks offer a flexible way of handling non-regular dataset such as sparse measures in the 3D space. In this work we investigate wether different types of graphs and nodes features can help modelling the wind speed at cruising altitude.

EDIC
Idiap Research Institute
Machine Learning Group

Arnaud Pannatier
arnaud.pannatier@epfl.ch
arnaudpannatier.ch

1 Introduction

Air Traffic Controllers (ATC) need to have access to reliable wind speed forecasts to organize the airspace efficiently. At cruising altitudes, the only measures that one has access to are measured by airplanes that record the wind along their trajectories, and fortunately, as there are a lot of planes flying in the airspace ATCs have recorded a lot of data. Their interest is to be able to a good prediction during the window of time they operate which is rather short (from 30min up to a few hours). In that range, called nowcasting, one gets the most accurate forecast by extrapolating from the latest measures available rather than trying to solve expensive numerical equations solvers. It is in that range that deep learning methods thrive, as meteorological data is often highly available. Sadly these data are recorded along trajectories and are sparse in space, so we cannot use standard methods such as CNN to deal with it. We noticed that it could be a good candidate for Graph Networks are their inductive bias are a good fit for non-regular dataset.

This work will use graph networks to forecast wind speed at cruising altitudes. We will use the *SkySoft ATM MALAT Wind Speed Dataset* [Berling et al., 2021] and starting from a Graph Element Network (GEN) [Alet et al., 2019] implementation, we will analyze the effect of the topology of the graphs and different nodes and edge features on the performance of the model. The implementation of the different models is publicly available. [1]

1.1 Problem formulation

Let $\mathbb{X}, \mathbb{I}, \mathbb{O}$ be three bounded metric spaces, and define two functions $f : \mathbb{X} \to \mathbb{I}$, mapping from data to the input space, and $g : \mathbb{X} \to \mathbb{O}$ mapping to the output space. In the case of wind nowcasting, $\mathbb{X} = \mathbb{R}^3 \times \mathbb{R}^+$, which corresponds to 3 spatial coordinates and one temporal and $\mathbb{I} = \mathbb{O} = \mathbb{R}^2$ which encode the wind in the (x, y) plane. We want to learn the transformation which maps a function f to g. We only have access to f by data $(x, i) \in \mathbb{X} \times \mathbb{I}$ of measures i at position x and we know g only through a set $(x', o) \in \mathbb{X} \times \mathbb{O}$ of measurements o at position x'. So we learn this mapping from data using multiple sets of measures corresponding to many (f, g) pairs to minimize the distance in the output metric space $\mathbb{O}$ between predictions and references.

2 Related works

2.1 Graph Element Networks (GENs)

Graph Element Networks [Alet et al., 2019] aims to model field transformations using non-regular graphs with nodes in the underlying space $\mathbb{X}$. Each measurement (x, i) is encoded using a small MLP, the initial latent feature of one node is then computed by averaging the embeddings of the measurements based on their distance to the node. The model then processes this latent variable using T steps of message passing, using the following equations:

$$m_{ij}^{t+1} = m_e(l_i^t, l_j^t) \tag{1}$$

$$l_j^{t+1} = m_n(l_i^t, \sum_{(i,j)\in\mathcal{N}} m_{ij}^{t+1}) \tag{2}$$

Where m_e, m_n are multilayer perceptrons and l_i^t is the latent state of nodes i at the t-th step of message passing. In that representation, no edge features are used. To predict a value at a new query position, the model linearly extrapolates in latent space and decodes using a small MLP modeling the transformation from latent to output space.

[AP] advantages – drawbacks

[1] The code of this work is available at `https://github.com/ArnaudPannatier/nml-project`

Figure 1: GEN architecture [Alet et al., 2019]. The measures are first encoded to a latent space, then the initial node feature is computed by aggregating the closer encoded measures, the processing is done by message passing and the final measures are retrieved by first interpolating the features of the nodes at the query position and then decoding.

2.2 Finite Element Networks

Traditional PDEs solvers model the domain using regular grids, but this has some drawbacks as some parts of the space might be more complicated to model than the others. One way of dealing with this problem is to use a Finite Element method approach [Hughes, 2012], which uses a non-regular graph that can be denser in the more complex regions and coarser in the smoother regions.

[AP] advantages – drawbacks

3 Exploration

3.1 Graph Types

Usually, the mesh for FEM simulations is created beforehand with denser regions where we expect the field to behave more complicatedly way. Meshes for this simulation are usually a triangulation of the region of interest. The triangulation subdivides the space into nonoverlapping triangles, such that each triangle have either a side or a vertex in common or is disjoint. A density parameter defines the finesse of the triangulation. The graph generated by this triangulation scheme is such that there is an edge between two nodes only if there are close in the Euclidean sense. The degree distribution of these graphs has only small values corresponding to the local neighborhood.

Figure 2: Typical FEM meshing [Pfaff et al., 2020]

For GENs, the original way of creating graphs is to create a grid mesh that covers the whole space, and encode the node position as parameters of the model, so that it can be optimized by gradient descent during training. In that case, the graph structure remains similar to the initial grid structure.

In both apporaches, only those graph structures have been used. In this work, we will consider three approaches for our graph topology. The first one, considered as a baseline is constructed as follows: First, we will take the k-means of the measurement positions and we will add edges between the nearest neighbors. The second one will be a random network where the nodes' position and their edges are taken at random, we will try different parameters p and analyze their effect on the quality of the model.

For the last graph structure, we will use a Barabási-Albert model scheme for growing a graph to a given number of nodes, we will then associate to each node one of the k-means positions. FEM and GEN are traditionally only using edges between neighbors, but we think that having

Figure 3: Generalization of 1d positional encoding (right) to 2d/3d with the choice of concatenating (first two columns) or adding (last columns.)

long-range connections and hubs can help the models process the information more efficiently. A sketch of the graphs can be found in Figure 4.

3.2 Features

In this work, we will focus on nodes and edges features as we are using the graph network to do regression over the whole space. We won't need to aggregate the features over the whole graph.

3.3 Nodes features

As described in section 2.1, GEN is only using node features that are computed by aggregating the learned embeddings of the measures based on their distance. We will visualize the effect of the message passing on the features. This will help us to choose the message passing steps and guide us through the analysis of the different graph structures.

We assume that this part is the most important to get a good model. So we will explore different strategies and measure their impact on the model. First, we will try to use no encoder and aggregate the normalized measured values. With this approach, we delegate the whole processing to the graph networks, and we think this should be already a strong baseline as the outputs values are strongly related to these initial values. We will then try to use fixed sine-cosine positional encodings [Vaswani et al., 2017] and 3D positional fixed sine-cosine positional encodings [Chu et al., 2021]. The idea is to use a simple 1D positional encoding that takes as input the position x, y or z and maps into a feature space of larger dimensionality. These encodings can be added or concatenated and we will try both approaches to see which one helps during training. A sketch of these positional encodings can be found in Figure 3.

3.4 Edge features

A contribution of this work is to add edge features to GENs. We think that good graph networks for space modeling should be able to compute an approximation of the derivatives of values in the graph. We will take the Euler scheme as an inspiration and have an explicit design for edge features that allows this kind of computation. We will make an ablation with learned features and see their effect on the quality of the model.

The inspiration is that the evolution of the wind speed is governed by the local movement

Model		Nodes		
		10	100	1000
Random Network, $p = 1\%$	FIXED	0.80 $\pm$ 0.04	0.71 $\pm$ 0.02	0.70 $\pm$ 0.02
	ADAPT	0.69 $\pm$ 0.01	0.65 $\pm$ 0.02	**0.64** $\pm$ **0.02**
Random Network, $p = 10\%$	FIXED	0.76 $\pm$ 0.03	0.70 $\pm$ 0.02	0.76 $\pm$ 0.02
	ADAPT	0.69 $\pm$ 0.01	0.66 $\pm$ 0.01	0.66 $\pm$ 0.02
k**-means, 3 nn.**	FIXED	0.68 $\pm$ 0.01	0.69 $\pm$ 0.02	0.68 $\pm$ 0.01
	ADAPT	0.68 $\pm$ 0.01	0.66 $\pm$ 0.01	0.65 $\pm$ 0.01
k**-means, BA**	FIXED	0.69 $\pm$ 0.01	0.68 $\pm$ 0.01	0.66 $\pm$ 0.02
	ADAPT	0.68 $\pm$ 0.02	0.65 $\pm$ 0.02	**0.64** $\pm$ **0.01**

Table 1: We tried different graph topologies for our model that use no embeddings. First, as a baseline, we tried Random Networks, where the position of each node is drawn from $\mathcal{U}(-1, 1)$ and where there is an edge between two nodes with probability p. Then we tried to use k-means to get the position of the nodes and added edges either between the 3 nearest neighbors in the graph or following a Barabási-Albert procedure. The results are the MSE error on the validation set, we ran ten different seeds to get the standard deviation. All measures are normalized.

equation [Meister, 2004]:

$$\rho \mathbf{g} - \nabla p = \rho \frac{d\mathbf{v}}{dt} \tag{3}$$

Roughly when using a traditional solver we will approximate the spatial and temporal derivatives using finite differences [Wikipedia contributors, 2004]:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = F_i^n(u, x, t, \frac{\partial u}{\partial x}, i\frac{\partial^2 u}{\partial x^2}) \tag{4}$$

So we think that adding edge features that hold the distance between two nodes can help model the PDEs.

4 Exploitation

4.1 Graph Types

The four different graphs that we tried are depicted in Figure 4. The first-row show the case where the nodes are fixed during training and the second row contains graphs where the node position is optimized during training with gradient descent. The performance of the different graph types is displayed in Table 1. We ran experiments using ten different seeds, for each node embedding and the position of the graph could be optimized in the case where the structure is fixed. We see that allowing positions to move during training (ADAPT in the table) are always performing better than their fixed counterpart. Random Graphs are the baseline and we expect them to perform poorly as they are initialized randomly without taking into account additional insight that we can have about the dataset. We see that indeed their performance is worse than the other models, but when the position of the nodes can be optimized Random Networks are on par with the best network structure.

Traditional approaches are using only edges between neighbors, Their idea might have been that for modeling the dynamics of the wind the information should only influence its neighborhood. However, we found that the best performing graphs (Random Networks with optimized position and Barabási-Albert graphs) achieve the best performance. This might be due to the presence of hubs and long-range interaction which might helps the models to organize the information hierarchically.

(a) Random network with $p = 1\%$ (b) Random network with $p = 10\%$ (c) k-means, 3nn (d) k-means, BA

(e) Random network with $p = 1\%$ (f) Random network with $p = 10\%$ (g) k-means, 3nn (h) k-means, BA

Figure 4: Graph structure for the different models. The first row (a-d) show graphs where each node has a fixed position, in the second row (e-h) we see how this position has evolved during training.

Model		Nodes		
		10	100	1000
No Emb	FIXED	0.68 $\pm$ 0.01	0.69 $\pm$ 0.02	0.68 $\pm$ 0.01
	ADAPT	0.68 $\pm$ 0.01	0.66 $\pm$ 0.01	0.65 $\pm$ 0.01
MLP	FIXED	0.66 $\pm$ 0.01	0.66 $\pm$ 0.02	0.67 $\pm$ 0.02
	ADAPT	0.66 $\pm$ 0.01	0.64 $\pm$ 0.01	0.64 $\pm$ 0.03
Added Pos-Enc,MLP	FIXED	1.03 $\pm$ 0.04	1.06 $\pm$ 0.02	1.04 $\pm$ 0.02
	ADAPT	1.07 $\pm$ 0.07	1.04 $\pm$ 0.03	1.03 $\pm$ 0.02
Concat Pos-Enc,MLP	FIXED	0.65 $\pm$ 0.01	0.63 $\pm$ 0.01	0.65 $\pm$ 0.02
	ADAPT	0.65 $\pm$ 0.02	**0.63** $\pm$ **0.01**	0.64 $\pm$ 0.01

Table 2: We tried different types of node embeddings for our model. The first one is the baseline is to use no embedding layer and keep the aggregated wind value as the node state. The second one uses a small MLP encoder to project the wind value to a higher-dimensional space. Finally, we explored two versions of the positional encoding layer. The results are the MSE error on the validation set, we ran ten different seeds to get the standard deviation. All measures are normalized.

4.2 Nodes and Edge features

We tried four different ways of embeddings the information in the nodes. The results of the different experiments can be found in Table 4. We ran ten experiments, changing the seed, for each node embedding and the position of the graph could be optimized in the case where the structure is fixed. We first use no embeddings at all, using directly the measurements values. This allowed us to see the processing performed by the network [Fig.5]. We found that using no embeddings and delegating all the processing to the message passing layers was already performing on par with the best models. We think that the models are powerful enough to process the information and that the other approach needs to learn an additional encoding/decoding network which might require longer training to reach the best accuracy. We trained all the models for 10 epochs (but each epoch contains roughly 33 M measurements) so in that range, the simpler model might be easier to learn. The best performing embedding was using an MLP on top of concatenated positional encodings. This might help the model process the information at a different scale and apparently, it helps a little. Finally, we found that adding the positional encodings altogether was decreasing a lot the accuracy of the model. Maybe under this representation distinguishing between positions is harder for the model.

Figure 5: Using no-embeddings allows to visualize the model processing the info layer after layer. The first column shows the graph structure, the second represents a minibatch of data that is then encoded (third column), then we see two steps of message passing and the decoded values.

4.3 Edge Features

We added edge features to the graph that encode the distance between the nodes or the finite difference between the initial node features. These values are computed with the following formulae:

$$d_{ij} = \|p_i - p_j\|_2 \tag{5}$$

$$F_{ij} = \frac{p_i - p_j}{d_{ij}} \tag{6}$$

Edge Features		Nodes		
		10	100	1000
Encoded distance	ADAPT	0.64 $\pm$ 0.02	0.63 $\pm$ 0.01	0.63 $\pm$ 0.01
Encoded Finite difference	ADAPT	0.64 $\pm$ 0.02	0.63 $\pm$ 0.01	**0.62** $\pm$ **0.01**

Table 3: We tried to add edge embeddings corresponding to the distance or the finite-difference between the nodes to our best-performing model. These values are computed before message passing and additionally, they are encoded using a small MLP. With these edge embeddings, we reach the best performance. The results are the MSE error on the validation set, we ran ten different seeds to get the standard deviation. All measures are normalized.

where p_i corresponds to the position of the i-th node. We compute these values before the message passing scheme and we encode them using a small MLP. In theory, the graph network shouldn't need them as, by the universal approximation theorem, it should be able to compute it from the node features, but in practice, we see that precomputing these values helps the model as it doesn't have to learn these relations.

5 Conclusion

We used graph networks to forecast the wind speed at cruising altitudes on a horizon of 30min. Graph Networks were able to efficiently process measurements sparsely distributed in the space. This architecture has a good inductive bias for this problem as each node is capable of approximating a part of the space. As the dataset at our disposal is sparse with most of the measurements clustered along some main routes, graph networks are more efficient than other grid-based solvers such as traditional PDEs solvers or CNN. We started by reimplementing the architecture presented in [Alet et al., 2019]. We used several graph structures and different node embeddings. We found that using graphs with hubs and edges between far-away nodes helps the model process the information efficiently. We found that the model doesn't need to have special embeddings as the message-passing architecture is powerful enough, but adding a small MLP encoder with concatenated positional encodings helps a bit. And finally adding edge features to the graph network encoding the finite difference between the node features helped the model to reach the best performance. We think that precomputing these values helps the model as it doesn't need to learn them and they are needed as they are an approximation of the first derivative. Further work might be done exploring these features and their effect on the performance.

References

[Alet et al., 2019] Alet, F., Jeewajee, A. K., Villalonga, M. B., Rodriguez, A., Lozano-Perez, T., and Kaelbling, L. (2019). Graph element networks: adaptive, structured computation and memory. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 212–222. PMLR.

[Berling et al., 2021] Berling, D., Pannatier, A., and Fleuret, F. (2021). Skysoft atm malat wind speed.

[Chu et al., 2021] Chu, X., Tian, Z., Zhang, B., Wang, X., Wei, X., Xia, H., and Shen, C. (2021). Conditional positional encodings for vision transformers. *arXiv preprint arXiv:2102.10882*.

[Hughes, 2012] Hughes, T. J. (2012). *The finite element method: linear static and dynamic finite element analysis*. Courier Corporation.

[Meister, 2004] Meister, J.-J. (2004). *Physique des Fluides*.

[Pfaff et al., 2020] Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A., and Battaglia, P. (2020). Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

[Wikipedia contributors, 2004] Wikipedia contributors (2004). Crank-nicolson method — Wikipedia, the free encyclopedia. [Online; accessed 08-June-2022].

6 Appendix

		Arnaud	**Yasmin**
Homeworks	HW1	100 %	
	HW2	1-version + review	
	HW3	100 %	
	HW4	100 %	
Project	Code	100 %	
	Report	100 %	
	Slides		
	Presentation tool		

Table 4: Workload repartition between the two teammates TODO : Yasmin write your contributions here