ALTEGRAD Final Project

Samuel Hurault
Ecole Normale Superieure Paris-Saclay
samuel.hurault@ens-paris-saclay.fr

Hannah Bull
Ecole Normale Superieure Paris-Saclay
hannah.bull@ens-paris-saclay.fr

1. Introduction

The goal of the data challenge is to implement a graph regression problem with the constraint of using a Hierarchical Attention Network (HAN) architecture. The graph is first represented as a document whose words are the nodes and whose sentences are random walks sampled from the network. Then, the documents are vectorized as in standard NLP algorithms. The HAN is build on top of this embedding layer. The dataset is made up of 93,719 undirected, unweighted graphs. Each graph is associated to four target values. We build a regression model for each of the target variable. A basic HAN implemented in keras is given. The objective is to optimize the different levels of the overall architecture, from the random walks to the final regression layer. The optimization realized independently for each of the four target variable. Moreover all the methods are evaluated by cross-validation on a validation set containing of 20% of the training data.

During this challenge, we tried to test as many methods as possible to improve the network, given time and computational power constraints. There are many situations where we would have liked to construct a larger grid search, but did not want to waste too many google console credits. Thus, we decided to limit our grid search of optimal parameters in order to better understand and apply a wider range of techniques. We gained a better understanding of the HAN architecture and the techniques involved in using data at different scales (documents, sentences, words).

2. Relevant literature

The type of problem in this data challenge is a graph regression problem, which is obvious strongly related to graph classification problems. The HAN architecture is commonly used in situations where data has a 'nested panel' form, such as in the reference article provided [10] and in the article [1]. The method is not just for documents/sentences/words data. For example [9] uses this data for action recognition in videos in order to incorporate static frames, short motions and long-term video structure.

We particularly look for techniques used in graph classification. Embeddings are commonly used in order to transform words, sentences, documents, graphs or other objects in 'difficult formats' into vectors. In particular, node2vec ([3]) and graph2vec ([5]) are commonly used techniques.

Aside from these embeddings, another alternative for dealing with such types of data is to use graph kernels. This allows the calculation of distance in a feature space. However, it is not entirely obvious how to exploit graph kernels in a neural network framework, as in general they are usually used in combination with classifiers such as SVM, for example in [6].

Attention networks have recently become very popular and an article that has made a bit of buzz is [8]. This article argues that methods such as RNN/LSTM can be replaced with attention modules. We thus hope that using attention modules in an optimal way on the data challenge will lead to good results.

It is unfortunately difficult to consult domain specific papers in this data challenge, as we do not know what the data represents.

3. HAN architecture

The original architecture is provided in [10] and illustrated in figure 1. The HAN layer is described in figure 2.

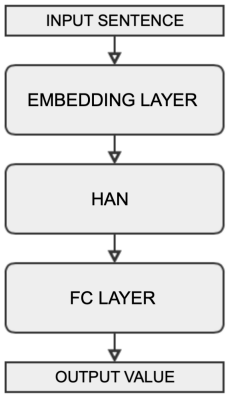


Figure 1. Global model structure

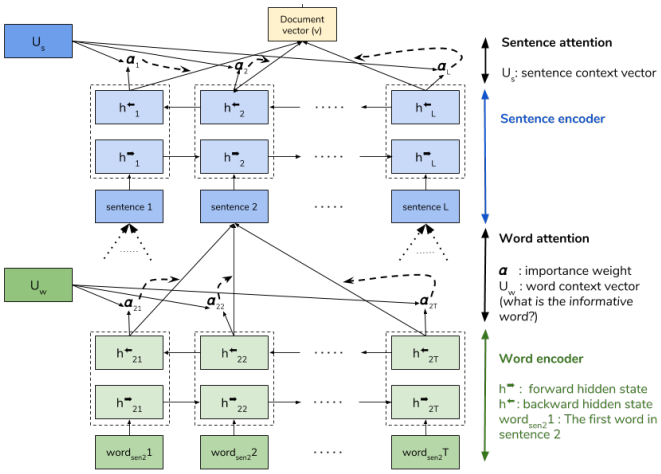


Figure 2. Hierarchical Attention structure

We first use this architecture with the following original parameters :

- number of hidden units in the RNN layers $n\_unit = 50$
- dropout after the attention layers $dropout = 0.5$
- $adam$ opimizer

We observe that the target values are normalized with zero mean and unit standard deviation. However the given model has a fully connected final layer with a sigmoïd activation function. Therefore it outputs values constrained between 0 and 1. We propose two different method to modify this:

- Replace this final layer with two TC layers followed by respectively ReLU and linear activations.
- Normalize the target training values y_i in the following way to clip them between 0 and 1 : $y_i \longleftarrow \frac{y_i + |min\{y_i\}_i|}{|max\{y_i\}_i - min\{y_i\}_i|}$. Then add a final layer to the network where the output $y \longleftarrow y * |max\{y_i\}_i - min\{y_i\}_i| - |min\{y_i\}_i|$

We try both strategies and average the mean square error (MSE) obtained for the four datasets. We found that the method one gives better results. It is explained by the fact that the normalization min-max implies the approximation that the min and max of the training targets are close to the ones of the testing targets. We now keep the first method.

4. Sampling

The graph is first represented as a document whose words are the nodes and whose sentences are random walks sampled from the network. Given a graph, for each node we create num_walks of size $walk\_length$, initialized at this node. Originally, $num\_walks = 5$ and $walk\_length = 10$. Each generated document is limited to $max\_doc\_size = 70$ sentences. We could not augment this limit due to memory constraints.

4.1. Biased random walks

We first try to bias the random walks like it is done in Node2vec [3]. The random walk scheme is illustrated in figure 3. The parameter p controls the likelihood of immediately returning to a node in the walk. The parameter q is the in-out parameter that controls inward or outward exploration. If $q > 1$ then the random walk is biased towards nodes close to the current node. If $q < 1$ then the random walk is biased towards nodes that are further away. We find in table 1 that a small value of p and a large value of q works well. The walk thus remains local. This suggests that local information is important for the labelling of the graphs.

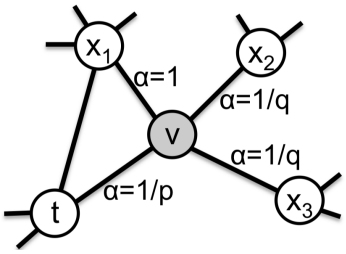


Figure 3. Illustration of the random walk procedure in node2vec. The walk just transitioned from t to v and is now evaluating its next step out of node v. Edge labels indicate search biases α.

	MSE 1	MSE 2	MSE 3	MSE 4
Original	0.320	0.228	0.565	**0.258**
$p = 0.25$, $q = 4$	**0.295**	**0.147**	**0.518**	0.262
$p = 4$, $q = 0.25$	0.341	0.265	0.946	0.260
$p = 1$, $q = 2$	0.363	0.337	0.516	0.254

Table 1. Biased in random walk parameters

4.2. Random walk parameters

As we find that the values $p = 0.25$, $q = 4$ work relatively well, we test a different number of walks. Increasing the number of walks from 5 to 20 does not improve the result, except for the fourth case.

	MSE 1	MSE 2	MSE 3	MSE 4
$p = 0.25$, $q = 4$, $n = 5$	**0.295**	**0.147**	**0.518**	0.262
$p = 0.25$, $q = 4$, $n = 20$	0.373	0.263	0.602	**0.235**

Table 2. Number of random walks

5. Node embeddings

The Embedding layer is the first layer of the model. This layer attributes a unique vector representation to each node of dimension d. Each document of size $n\_walks \times walk\_length$ becomes $n\_walks \times walk\_length \times d$.

5.1. Initial embedding

The challenge provides an initial pretrained node Embedding. It assigns to each node a 13-dimensional attribute vector. All the dimensions except the 5th and 6th have binary values. Dimension 5 has integers and dimension 6 floats. We observe in table 3, the gain in MSE when normalizing these 2 layers.

	MSE 1	**MSE 2**	**MSE 3**	**MSE 4**
Original	0.527	0.360	1.03	1.927
Dimensions 5 and 6 normalized	**0.308**	**0.179**	**0.518**	**0.232**

Table 3. Normalization initial node embedding

5.2. Enriching node attribute vectors

With adjacency matrix columns

We now try to add additional dimension to node embedding. We have to find new features representing each node. A first straightforward node feature is the column of the adjacency matrix of the graph this node belongs to. This adjacency matrix are padded with 0 values to match the bigger graph dimension of 29 nodes (padding an adjacency matrix is equivalent to adding unconnected nodes). We extract for each node a 29 dimensional vector and concatenate this vector to the previous node embedding. The new node embedding is in dimension $29 + 13 = 42$.

We show in table 4 the final MSE results with and without adding this new feature to the Embedding matrix. We observe that this augmentation only works for the regression 4.

	MSE 1	**MSE 2**	**MSE 3**	**MSE 4**
without adjacency feature	**0.308**	**0.179**	**0.518**	0.232
with adjacency feature	0.312	0.198	0.851	**0.165**

Table 4. Adding adjacency matrix columns to the node attribute vectors

With Node2Vec vectors

The paper [3] constructs node embeddings to learn representations of graphs. The types of problems looked at in this paper are node classification and link prediction, so it is not entirely obvious that these embeddings will be useful for our task of regression on graphs. The paper [7] uses node2vec embeddings for a graph classification task as input to a CNN. They show that it leads to state-of-the-art performance.

We replicate this method on our data. The problem with this method is that node2vec is stochastic and treats the graphs one-by-one, then a feature dimension for one graph does not necessarily correspond to the same feature dimension for another graph. Thus, it is is necessary to reduce the dimensions to a common dimension with PCA. We choose to keep the 5 first PCA dimension.

The node2vec embeddings takes several hours to compute with 24 CPUs. It restricted the amount of parameters that we could test.

Value of (p, q)	**MSE 1**	**MSE 2**	**MSE 3**	**MSE 4**
Original	**0.308**	**0.179**	**0.518**	0.232
(1,1)	0.390	0.267	0.719	**0.230**
(1,2)	0.394	0.377	0.547	0.251
(1,0.5)	0.373	0.262	0.602	0.235

Table 5. Node embeddings tuning

Unfortunately, this method does not work well. We think that the embedding is not adapted to our specific problem. Indeed, as nodes are labelled differently, node2vec is applied one-by-one to each graph. It prevents from capturing efficiently the similarity across documents. It might be efficient to relabel the nodes with the WL-kernel before conducting this node2vec embedding.

6. Graph embeddings

In the network, the sentence attention layer builds a vectorial representation of a document. We try to enrich this layer using graph embeddings. We use the graph embeddings proposed in [5]. These graph embeddings achieve state-of-the-art performance on graph classification datasets, and so we think that they should be able to help in our task.

For each document, we extract 20 graph features. We append these to the document vector at the end of the network. There is a marginal improvement for the third label.

	MSE 1	**MSE 2**	**MSE 3**	**MSE 4**
Original	**0.308**	**0.179**	0.518	**0.232**
With graph2vec embeddings	0.312	0.275	**0.494**	0.239

Table 6. Adding graph2vec document embeddings

We did not find an article that uses graph2vec embeddings in conjunction with a HAN, so it is difficult to know exactly where and how they should be implemented into the network. Perhaps with a better placement strategy, our results would have been better.

7. Model architecture

Given the data sampling and the previous pre-trained embedding, we now optimize the architecture.

We first try to modify the initial bi-directional GRU layers by bi-directional LSTM layers.

RNN	MSE 1	MSE 2	MSE 3	MSE 4
GRU	0.290	**0.194**	**0.516**	0.284
LSTM	0.287	0.215	0.522	**0.255**

Table 7. RNN choice

Given these results we keep GRU for regressions 1, 2 and 3. We use LSTM for regression 4.

We also propose two augmentations of the network :

- add, after the Attention layer, two fully connected layers with $\frac{n\_unit}{2}$ hidden neurons.
- add another bi-RNN (bi-GRU or bi-LSTM) layer on top of the existing ones at sentence and word levels.

We show in table 8 the results of such augmentations. We observe that adding another bi-RNN layer on top of the previous one is useful, contrary to adding FC layers at the end of the network.

Architecture	MSE 1	MSE 2	MSE 3	MSE 4
Original	0.287	0.141	0.516	0.127
Additional FC layers	0.3156	0.182	0.553	0.168
Additional bi-RNN layer	**0.251**	**0.133**	**0.505**	**0.056**

Table 8. Architecture augmentation

Adding context vectors

As shown in figure 1, the attention layers at sentence and word levels use unique context vectors u_s and u_w. We use the notations of [10]. u_s is a 1-D vector of lenght n_walks. With the outputs h_i of the bi-RNN layer :

$$u_i = tanh(W_w h_i + b_w)$$

$$\alpha_i = softmax(u_i^T u_s)$$

$$v = \sum_i \alpha_i h_i$$

In [4], the authors explain that this vector representation u_s usually focuses on a specific component of the sentence, so it is expected to reflect an aspect, or component of the semantics in a sentence. However, there can be multiple components in a sentence that together forms the overall semantics of the whole sentence, especially for long sentences. Thus we need to perform multiple hops of attention.

If we want r different parts to be extracted from the sentence, we introduce r different $(u_s^k)_k$ and we now have for k in $1, .., r$:

$$\alpha_i^k = softmax(u_i^T u_s^k)$$

We finally average all the different representations :

$$\alpha_i = \frac{\sum_k \alpha_i^k}{r}$$

$$v = \sum_i \alpha_i h_i$$

We also can realize the same extension for the attention at word level. We observe in table 9, the effect of using multiple context vectors at sentence and word levels. We observe that adding context vectors does not improve the score. It might be explained by the fact that our sentences don't have the complexity of a true language document. The context is easier to understand and one context vector is enough. Adding other vectors may over-fit and thus worsen the result.

Nb context vectors	MSE 1	MSE 2	MSE 3	MSE 4
1	**0.289**	**0.158**	**0.525**	**0.101**
2	0.298	0.181	0.527	0.245
3	0.303	0.197	0.526	0.150

Table 9. Adding attention vectors

8. Training

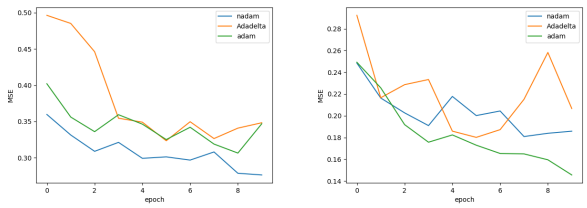
The past experiments were conducted with the adam optimizer with standard parameters $lr = 0.001$, $beta\_1 = 0.9$, $beta\_2 = 0.999$. We also experimented with the optimizers NAdam [2] and Adadelta [11]n without fine-tuning the involved parameters like the learning rate. In figure 4, we show, for each optimizer the evolution of the validation loss during training.

Given this curves, we now use the optimizer Nadam for the regressions 1 and 3 and we keep the Adam optimizer for regressions 2 and 4.

9. Conclusion

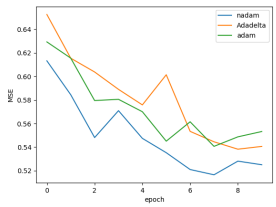
Our final method obtains an MSE of 0.207 on the Kaggle public test set. We optimized different hyper-parameters for each of the 4 graph label. This optimization could be further processed with a complete grid search. Indeed, cross-validation takes time for little educational gain, then we limited our choices.

Given more time, we would have liked to understand how to use kernel methods in conjunction with the Hierachical Attentional Network. Perhaps it is possible to use graph kernels in order to cluster graphs/nodes in the feature space, and then use these clustering methods in order

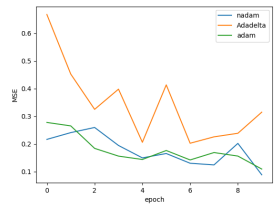


(a) Regression 1

(b) Regression 2



(c) Regression 3



(d) Regression 4

Figure 4. Evolution of the validation loss during training for different optimizers and for the 4 types of regressions.

to add similarity features to the model. We would have also liked to better understand how to incorporate node2vec and graph2vec in the model. The node2vec embeddings can clearly be used as inputs to the neural model, but it is unclear how to ensure that the node2vec features can be aligned across distinct graphs with differently labelled nodes. Perhaps it is necessarily to firstly relabel the nodes before constructing the node2vec features, for example using the first part of the Weisfeiler-Lehman kernel method. It remains unclear how to incorporate graph2vec features after the sentence encoder.

Finally, we would have liked to spend more time constructing the structure of the network, experimenting further with skip-connections and choosing the best optimizer parameters.

References

[1] K. Al-Sabahi, Z. Zuping, and M. Nadher. A hierarchical structured self-attentive model for extractive document summarization (hssas). *IEEE Access*, 6:24205–24212, 2018.

[2] T. Dozat. Incorporating nesterov momentum into. 2015.

[3] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.

[4] Z. Lin, M. Feng, C. N. d. Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio. A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130*, 2017.

[5] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal. graph2vec: Learning distributed representations of graphs. *arXiv preprint arXiv:1707.05005*, 2017.

[6] N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep):2539–2561, 2011.

[7] A. J.-P. Tixier, G. Nikolentzos, P. Meladianos, and M. Vazirgiannis. Graph classification with 2d convolutional neural networks. *arXiv preprint arXiv:1708.02218*, 2017.

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[9] Y. Wang, S. Wang, J. Tang, N. O'Hare, Y. Chang, and B. Li. Hierarchical attention network for action recognition in videos. *arXiv preprint arXiv:1607.06416*, 2016.

[10] Z. Yang, D. Yang, C. Dyer, X. He, A. J. Smola, and E. H. Hovy. Hierarchical attention networks for document classification. In *HLT-NAACL*, 2016.

[11] M. D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.