Assignment 2. Recurrent Neural Networks and Graph Neural Networks

Kai Liang
University of Amsterdam
kai.liang@student.uva.nl

1 Vanilla RNN versus LSTM

Question 1.1

It is considered for this question that the gradient of a scalar with respect to a column vector is a row vector. Firstly, for $\frac{\partial \mathcal{L}^{(T)}}{\partial \mathbf{W}_{ph}}$:

$$
\begin{aligned}
\frac{\partial \mathcal{L}^{(T)}}{\partial \mathbf{W}_{ph}} &= \frac{\partial \mathcal{L}^{(T)}}{\partial \hat{\mathbf{y}}^{(t)}} \frac{\partial \hat{\mathbf{y}}^{(t)}}{\partial \mathbf{p}^{(t)}} \frac{\partial \mathbf{p}^{(t)}}{\partial \mathbf{W}_{ph}} \\
\frac{\partial \mathcal{L}^{(T)}}{\partial \hat{\mathbf{y}}_i^{(t)}} &= \frac{\partial - \sum_{k=1}^{K} \mathbf{y}_k^{(t)} \log \hat{\mathbf{y}}_k^{(t)}}{\partial \hat{\mathbf{y}}_i^{(t)}} \\
&= -\frac{\mathbf{y}_i^{(t)}}{\hat{\mathbf{y}}_i^{(t)}} \\
\frac{\partial \hat{\mathbf{y}}_i^{(t)}}{\partial \mathbf{p}_j^{(t)}} &= \frac{\partial \frac{\exp\left(\mathbf{p}_i^{(t)}\right)}{\sum_k \exp\left(\mathbf{p}_k^{(t)}\right)}}{\partial \mathbf{p}_j^{(t)}} \\
&= \frac{\delta_{ij} \exp\left(\mathbf{p}_j^{(t)}\right) \sum_k \exp\left(\mathbf{p}_k^{(t)}\right) - \exp\left(\mathbf{p}_i^{(t)}\right) \exp\left(\mathbf{p}_j^{(t)}\right)}{\left(\sum_k \exp\left(\mathbf{p}_k^{(t)}\right)\right)^2} \\
&= \delta_{ij} \hat{\mathbf{y}}_j^{(t)} - \hat{\mathbf{y}}_i^{(t)} \hat{\mathbf{y}}_j^{(t)} \\
&= \left(\delta_{ij} - \hat{\mathbf{y}}_i^{(t)}\right) \hat{\mathbf{y}}_j^{(t)} \\
\frac{\partial L^{(t)}}{\partial \mathbf{p}_j^{(t)}} &= -\frac{\mathbf{y}_j^{(t)}}{\hat{\mathbf{y}}_j^{(t)}} \left(1 - \hat{\mathbf{y}}_j^{(t)}\right) \hat{\mathbf{y}}_j^{(t)} - \sum_{i \neq j} \frac{\mathbf{y}_i^{(t)}}{\hat{\mathbf{y}}_i^{(t)}} \left(-\hat{\mathbf{y}}_i^{(t)} \hat{\mathbf{y}}_j^{(t)}\right) \\
&= -\mathbf{y}_j^{(t)} \left(1 - \hat{\mathbf{y}}_j^{(t)}\right) + \sum_{i \neq j} \mathbf{y}_i^{(t)} \hat{\mathbf{y}}_j^{(t)} \\
&= -\mathbf{y}_j^{(t)} + \hat{\mathbf{y}}_j^{(t)} \\
\frac{\partial L^{(t)}}{\partial \mathbf{p}^{(t)}} &= \left(-\mathbf{y}^{(t)} + \hat{\mathbf{y}}^{(t)}\right)^T
\end{aligned}
$$

$$\frac{\partial \mathbf{p}_i^{(t)}}{\partial \left(\mathbf{W}_{ph}\right)_{jk}} = \frac{\partial \left(\left(\mathbf{W}_{ph}\right)_{i,:} \mathbf{h}^{(t)} + \left(\mathbf{b}_p\right)_i\right)}{\partial \left(\mathbf{W}_{ph}\right)_{jk}}$$
$$= \begin{cases} 0, & \text{if } i \neq j \\ \mathbf{h}_k^{(t)}, & \text{if } i = j \end{cases}$$
$$\frac{\partial \mathcal{L}^{(T)}}{\partial \mathbf{W}_{ph}} = \mathbf{h}^{(t)} \left(-\mathbf{y}^{(t)} + \hat{\mathbf{y}}^{(t)}\right)^T$$

Then, for $\frac{\partial \mathcal{L}^{(T)}}{\partial \mathbf{W}_{hh}}$, assume that $\mathbf{h}^{(t)} = \tanh\left(\hat{\mathbf{h}}^{(t)}\right)$:

$$\frac{\partial \mathcal{L}^{(T)}}{\partial \mathbf{W}_{hh}} = \frac{\partial \mathcal{L}^{(T)}}{\partial \mathbf{p}^{(t)}} \frac{\partial \mathbf{p}^{(t)}}{\partial \mathbf{h}^{(t)}} \frac{\partial \mathbf{h}^{(t)}}{\partial \hat{\mathbf{h}}^{(t)}} \frac{\partial \hat{\mathbf{h}}^{(t)}}{\partial \mathbf{W}_{hh}}$$
$$\frac{\partial \mathbf{p}_i^{(t)}}{\partial \mathbf{h}_j^{(t)}} = \frac{\partial \left(\left(\mathbf{W}_{ph}\right)_{i,:} \mathbf{h}^{(t)} + \left(\mathbf{b}_p\right)_i\right)}{\partial \mathbf{h}_j^{(t)}}$$
$$= \left(\mathbf{W}_{ph}\right)_{ij}$$
$$\frac{\partial \mathbf{p}^{(t)}}{\partial \mathbf{h}^{(t)}} = \mathbf{W}_{ph}$$
$$\frac{\partial \mathcal{L}^{(T)}}{\partial \mathbf{h}^{(t)}} = \frac{\partial L^{(t)}}{\partial \mathbf{p}^{(t)}} \frac{\partial \mathbf{p}^{(t)}}{\partial \mathbf{h}^{(t)}}$$
$$= \left(-\mathbf{y}^{(t)} + \hat{\mathbf{y}}^{(t)}\right)^T \mathbf{W}_{ph}$$
$$\frac{\partial \mathbf{h}_i^{(t)}}{\partial \hat{\mathbf{h}}_j^{(t)}} = \delta_{ij} \left(1 - \left(\mathbf{h}_j^{(t)}\right)^2\right)$$
$$\frac{\partial \mathcal{L}^{(T)}}{\partial \hat{\mathbf{h}}^{(t)}} = \left(-\mathbf{y}^{(t)} + \hat{\mathbf{y}}^{(t)}\right)^T \mathbf{W}_{ph} \circ \left(\mathbf{1} - \left(\mathbf{h}^{(t)}\right)^2\right)^T$$
$$\frac{\partial \hat{\mathbf{h}}_i^{(t)}}{\partial \left(\mathbf{W}_{hh}\right)_{jk}} = \frac{\partial \left(\left(\mathbf{W}_{hx}\right)_{i,:} \mathbf{x}^{(t)} + \left(\mathbf{W}_{hh}\right)_{i,:} \mathbf{h}^{(t-1)} + \left(\mathbf{b}_h\right)_i\right)}{\partial \left(\mathbf{W}_{hh}\right)_{jk}}$$
$$= \delta_{ij} \left(\mathbf{h}^{(t-1)}\right)_k + \left(\mathbf{W}_{hh}\right)_{i,:} \frac{\partial \mathbf{h}^{(t-1)}}{\partial \left(\mathbf{W}_{hh}\right)_{jk}}$$
$$\frac{\partial \mathcal{L}^{(T)}}{\partial \mathbf{W}_{hh}} = \left(\mathbf{h}^{(t-1)} + \mathbf{W}_{hh} \frac{\partial \mathbf{h}^{(t-1)}}{\partial \left(\mathbf{W}_{hh}\right)}\right) \cdot \left[\left(-\mathbf{y}^{(t)} + \hat{\mathbf{y}}^{(t)}\right)^T \mathbf{W}_{ph} \circ \left(\mathbf{1} - \left(\mathbf{h}^{(t)}\right)^2\right)^T\right]$$

We can see that $\frac{\partial \mathcal{L}^{(T)}}{\partial \mathbf{W}_{ph}}$ only depends on the hidden state at time step t and the final output, while $\frac{\partial \mathcal{L}^{(T)}}{\partial \mathbf{W}_{hh}}$ depends on the hidden state at the previous time step $t-1$ which again depends on its previous time step $t-2$ etc. Hence, a problem of the latter gradient is that it involves cumulative multiplication of $t-1$ times (for time step t), which will easily lead to either vanishing gradient or gradient exploding depending on whether the magnitude of $\mathbf{W}_{hh}$ is larger or smaller than 1.

Question 1.2

The detailed implementation of the vanilla recurrent neural network (RNN) could be referred in `vanilla_rnn.py` and `part1/train.py`. The reason of using `clip_grad_norm` is that, vanilla RNN suffers from the vanishing gradient/gradient exploding problem, and this function clips/scales the gradient to a reasonable range.

Question 1.3

To test the performance of the RNN, experiments are conducted for palindromes of increasing length from 5 to 60. For each experiment, the final result is recorded for the average of 50 best accuracies. The reasons is that, measuring the 'maximum' accuracy is a good metric of showing model performance, while it might also sometimes reflect randomness. Thus, the average of 50 best accuracies are taken to avoid randomness and to indicate the degree of model convergence. The result is shown in Figure 1.

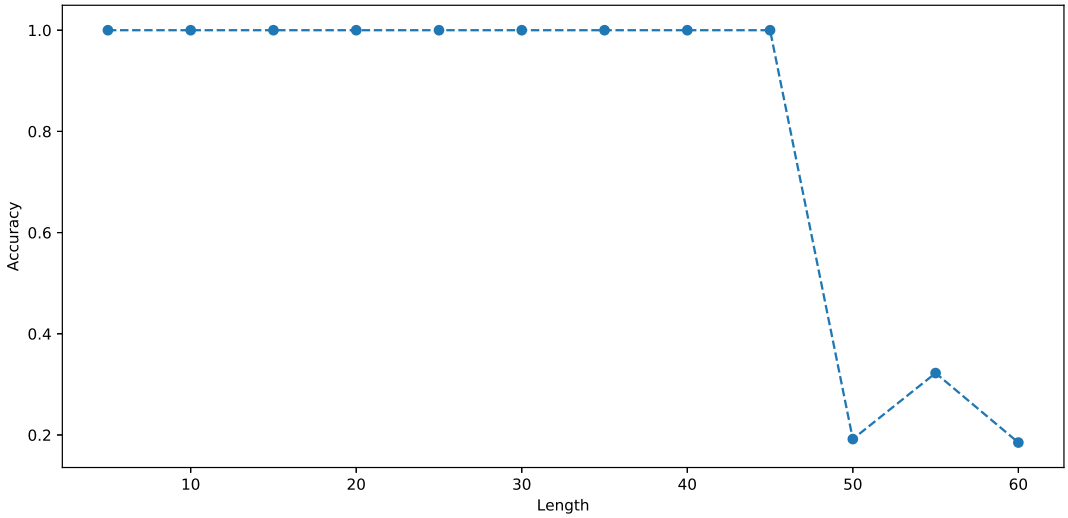

Figure 1: *The evolution of accuracy versus palindrome length using the vanilla RNN.*

Question 1.4

A drawback of vanilla stochastic gradient descent is that the path taken by SGD when updating the parameters oscillates too much which leads to slow convergence, so the benefit of adding **momentum** (e.g. Adam) is that it cancels out the oscillating gradients by averaging over the gradients of previous steps, which ensures more robust gradients and faster convergence.

Another drawback of vanilla SGD is that the learning rate needs to be pre-defined either as a fixed value or a scheduler and is applied to all parameters of the model, which is difficult as it varies for different types of models and data, so the benefit of using **adaptive learning rate** (e.g. RMSProp and Adam) is that the amplitude of the update of a parameter adjusts automatically according to the magnitude of its gradients. Specifically, if the gradients are large, the updates will be tamed, while if the gradients are small, the updates will be more aggressive.

Question 1.5

(a) All of the 4 gates take $\mathbf{h}^{(t-1)}$ and $\mathbf{x}^{(t)}$ as input and pass through some activation function, yet their purposes are different. For *input modulation gate*, it proposes information to add to the current cell state, and the advantage of using the tanh activation function is that its domain centers around zero and ensures a faster convergence. For *input gate*, it controls the amount (percentage) of information to 'remember' from the input of the current time step, and the advantage of using the sigmoid activation function is that its domain ranges from 0 to 1, which acts like a probability. For *forget gate*, it controls the amount (percentage) of information to 'unlearn' from the previous hidden state, and the advantage of it using the sigmoid activation function is the same as for *input gate*. For *output gate*, it controls the amount (percentage) of information to store in the current hidden state from the current cell state, and the advantage of using the sigmoid activation function is the same as for *input gate*.

(b) If not considering the linear layer (i.e. output layer), the formula for the total number of trainable parameters in the LSTM cell is given by $4nd + 4n^2 + 4n$, where:

- $4nd$: 4 *input-to-hidden* weight matrices.
- $4n^2$: 4 *hidden-to-hidden* weight matrices.
- $4n$: 4 bias vectors.

Question 1.6

The detailed implementation of the LSTM network can be referred in `lstm.py` and `train.py`. To reach a satisfactory performance of the model in the palindrome problem, `learning_rate` is changed from default to 0.01. Again, as similar in Question 1.3, the accuracy is measured for the average of 50 best ones among 10000 iterations. The result is shown in Figure 2.

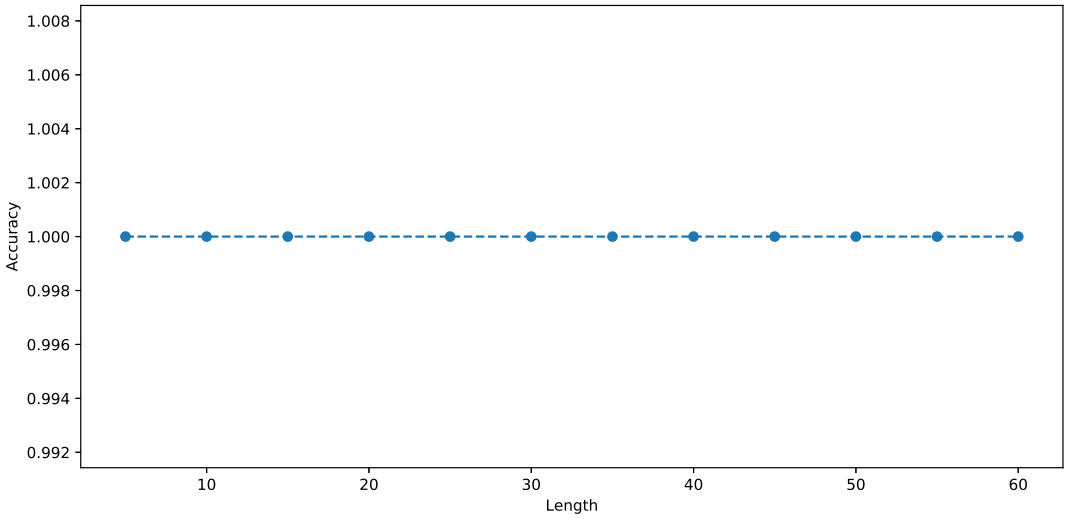

Figure 2: *The evolution of accuracy versus palindrome length using the LSTM.*

2 Recurrent Nets as Generative Model

Question 2.1

(a) The implementation of the two-layer LSTM network could be referred in `train.py` and `model.py`.

(b) The network is trained for 20000 steps in total on Grimm's Fairytales, and the evolution of accuracy and loss versus the training step is shown in Figure 3. Some examples of the generated sentences of length $T = 30$ during different phases of training are shown in Listing 1.

Listing 1: *Part of the generated sentences of length $T = 30$ by greedy sampling.*

```
--------------
Training Step: 100
--------------
We the the the the the the the

--------------
Training Step: 4100
--------------
Mand said, 'I have something t

--------------
```

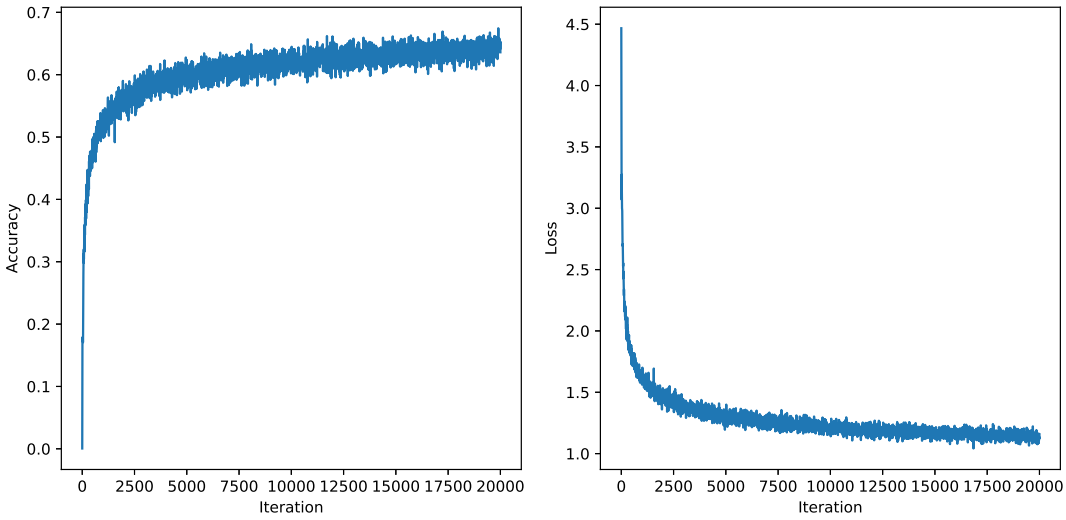

Figure 3: *The evolution of accuracy and loss versus the training step of the generative LSTM.*

```
Training Step: 8100
--------------
I will give you a little tailo

--------------
Training Step: 12100
--------------
Maston, and the princess was a

--------------
Training Step: 16100
--------------
 the work and said, 'I am sure
```

As could be inferred from Listing 1, when the training process evolves, the generated sentences become more grammatical and meaningful, which means that the network gradually learns to capture the (long-term) dependencies between words and characters from the corpus.

(c) Listing 2, Listing 3 and Listing 4 below display 5 text samples generated by the network over evenly spaced intervals during training under temperatures of $[0.5, 1.0, 2.0]$ respectively.

Listing 2: *Part of the generated sentences of length $T = 30$ by sampling with temperature = 0.5.*

```
--------------
Training Step: 100
--------------
qand wo aind the sin on was he

--------------
Training Step: 4100
--------------
Cheep states they went to the

--------------
Training Step: 8100
--------------
p and all the giant wished two

--------------
```

```
Training Step: 12100
--------------
jumped up and said, 'I am so t

--------------
Training Step: 16100
--------------
good time a great time and dra
```

Listing 3: *Part of the generated sentences of length $T = 30$ by sampling with temperature = 1.0.*

```
--------------
Training Step: 100
--------------
ead wfanry sir the avas ang hi

--------------
Training Step: 4100
--------------
Vane and running or time, that

--------------
Training Step: 8100
--------------
Now which has slept, for he sa

--------------
Training Step: 12100
--------------
Rap with that, you ask the pea

--------------
Training Step: 16100
--------------
Many which he took the reag sh
```

Listing 4: *Part of the generated sentences of length $T = 30$ by sampling with temperature = 2.0.*

```
--------------
Training Step: 100
--------------
H1t3'd gAlnwir rhoett meyt'bem

--------------
Training Step: 4100
--------------
Bath upon same, a;leccitys, 'i

--------------
Training Step: 8100
--------------
(eaf thelele? You, my geast an

--------------
Training Step: 12100
--------------
J$E EVEn GRAENBER DABHEMANAGE

--------------
Training Step: 16100
--------------
W8DOS,p%groum; fult as qui
```

It could be inferred from the above results that, when increasing the temperature (e.g. to 2), the sampling distribution of characters becomes more uniform, which results in more diverse while meaningless sequences regardless of the training stage. On the other hand, when decreasing the temperature (e.g. to 0.5), the sampling distribution of characters becomes more discriminative, which leads to less diverse but more meaningful sentences as the training evolves.

Bonus Question 2.2

To make the network finish sentences longer than $T = 30$ characters, firstly the beginning is fed as input to the network, after which the hidden state and memory state is iteratively passed as input to continue the generation until the desired length is reached. For instance, Listing 5 displays some generated sentences of $T = 64$ characters (excluding the given beginning) under different sampling strategies and temperatures.

Listing 5: *Part of the generated sentences of length $T = 64$ with different sampling strategies.*

```
--------------
Greedy Sampling
--------------
Sleeping beauty is all the world, when the soldier said, 'I will give you
any of t

--------------
Random Sampling (Temperature = 0.5)
--------------
Sleeping beauty is on the heart.' Then the man came back to her finger, and
set th

--------------
Random Sampling (Temperature = 1.0)
--------------
Sleeping beauty is the fire in their brother.' Hansel stayed together in
their lit
```

As could be inferred from the outputs, the model can almost always place punctuation symbols correctly. However, the model sometimes fails to learn (generate) the correct verb tense.

3 Graph Neural Networks

Question 3.1

(a) This layer exploits the structural information in the graph data by the adjacency matrix $\tilde{A}$, which encodes the nodes and the edges in the graph using values of, e.g. 0 and 1. A GCN layer can be seen as performing message passing over the graph by the following. Firstly, a node collects all the features from its neighbours with the adjacency matrix $\tilde{A}$. Then, it aggregates these feature vectors by multiplying the activation matrix $H^{(l)}$ with $\tilde{A}$ and passes through some activation function σ. Finally, the weight matrix $W^{(l)}$ can be updated accordingly, which finishes a loop of message passing.

(b) We at most need 3 GCN layers to stack to propagate to every node's embedding the information (features) from nodes 3 hops away in the graph [Zhou et al., 2018].

Question 3.2

GNNs could be applied to many real-world applications in which the datasets are in the form of graphs or network, which include [Zhou et al., 2018, Wu et al., 2018]:

- Social networks (for social relationship understanding etc.).
- Knowledge graphs (for generation and completion etc.).
- Recommender systems (for user-action prediction etc.).
- Chemical compounds (for protein interface prediction etc.).

Question 3.3

(a) For typical sequential data (e.g. videos and texts), it is expected that the RNN-based models will outperform the GNNs, which is for three reasons. Firstly, comparing the size of available datasets, it is usually hard to find ones which annotate sequential data in the graph form, while there are much more datasets of sequential data in the normal form. Obviously, more data likely mean better generalizability and better performance. Secondly, it is still an open question of defining the optimal graph representations for non-structural data (e.g. texts) [Zhou et al., 2018]. Thirdly, GNNs are always shallow (i.e. not deeper than three layers), while the RNN-based models can be stacked many layers. Again, deeper networks usually denote better expressivity [Zhou et al., 2018].

As for native graph-like data (e.g. social network, chemical compounds and knowledge graph), it is expected that GNNs will outperform the RNN-based models, which is for the following reasons. Firstly, GNNs are more flexible in terms of the idea of 'time step' and 'sequence', as edges and paths in a graph can be undirected, which the RNN-based models cannot properly model. Secondly, GNNs explicitly store dependencies between nodes in the graph by edges and perform the propagation accordingly, while RNN-based models only encode such dependencies within the nodes.

(b) Two examples of the combination between RNNs and GNNs are the following:

- Tree-LSTM [Tai et al., 2015], where the input to the network is not simply sequences of words, but their dependency trees (a.k.a syntactic trees), which is able to capture both word-sense and grammatical dependencies. Some classic tasks where Tree-LSTM can be applied are sentiment analysis and sentence disambiguation.
- Graph Convolutional Recurrent Network (GCRN) [Seo et al., 2016], which combines GCN to identify spatial structures and RNN to find dynamic patterns. GCRN can be applied to tasks such as modeling natural language and predicting dynamic graphical data.

References

Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph Neural Networks: A Review of Methods and Applications. *arXiv e-prints*, art. arXiv:1812.08434, Dec 2018.

Shu Wu, Yuyuan Tang, Yanqiao Zhu, Liang Wang, Xing Xie, and Tieniu Tan. Session-based Recommendation with Graph Neural Networks. *arXiv e-prints*, art. arXiv:1811.00855, Oct 2018.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1556–1566, Beijing, China, July 2015. Association for Computational Linguistics. doi: 10.3115/v1/P15-1150. URL `https://www.aclweb.org/anthology/P15-1150`.

Youngjoo Seo, Michaël Defferrard, Pierre Vand ergheynst, and Xavier Bresson. Structured Sequence Modeling with Graph Convolutional Recurrent Networks. *arXiv e-prints*, art. arXiv:1612.07659, Dec 2016.