Deep Learning Assignment 2

Yke Rusticus
Student nr.: 11306386
University of Amsterdam
yke.rusticus@student.uva.nl
28/11/2019

1 Vanilla RNN versus LSTM

Question 1.1

Given:

$$h^{(t)} = \tanh(W_{hx}x^{(t)} + W_{hh}h^{(t-1)} + b_h) \tag{1}$$

$$p^{(t)} = W_{ph}h^{(t)} + b_p \tag{2}$$

$$\hat{y}^{(t)} = \text{softmax}(p^{(t)}) \tag{3}$$

$$\mathcal{L} = -\sum_{k=1}^{K} y_k \log \hat{y}_k, \qquad L := \mathcal{L} \text{ for convenience} \tag{4}$$

Derivatives:

$\frac{\partial L^{(T)}}{\partial W_{ph}}$:

$$\frac{\partial L^{(T)}}{\partial W_{ph}} = \frac{\partial L^{(T)}}{\partial \hat{y}^{(T)}} \frac{\partial \hat{y}^{(T)}}{\partial p^{(T)}} \frac{\partial p^{(T)}}{\partial W_{ph}}, \tag{5}$$

where

$$\frac{\partial L^{(T)}}{\partial \hat{y}^{(T)}} = \begin{bmatrix} \frac{\partial L^{(T)}}{\partial \hat{y}_1^{(T)}} & \frac{\partial L^{(T)}}{\partial \hat{y}_2^{(T)}} & \cdots & \frac{\partial L^{(T)}}{\partial \hat{y}_K^{(T)}} \end{bmatrix} = -[y^{(T)} \oslash \hat{y}^{(T)}]^\top \tag{6}$$

$$\frac{\partial \hat{y}^{(T)}}{\partial p^{(T)}} = \text{diag}(\hat{y}^{(T)}) - \hat{y}^{(T)}(\hat{y}^{(T)})^\top \tag{7}$$

$$\frac{\partial p^{(T)}}{\partial W_{ph}} = \begin{bmatrix} \frac{\partial p_1^{(T)}}{\partial W_{ph}} \\ \frac{\partial p_2^{(T)}}{\partial W_{ph}} \\ \vdots \\ \frac{\partial p_K^{(T)}}{\partial W_{ph}} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} h^{(T)} & 0 & 0 & \cdots & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & h^{(T)} & 0 & \cdots & 0 \end{bmatrix} \\ \vdots \\ \begin{bmatrix} 0 & \cdots & 0 & 0 & h^{(T)} \end{bmatrix} \end{bmatrix} \tag{8}$$

$\text{diag}(\cdot)$ in Eq. 7 is defined such that for $x \in \mathbb{R}^N$, $\text{diag}(x)$ is the $N \times N$ matrix with on its diagonal the elements of x, and zeros elsewhere. Eq. 7 was derived in previous assignment. In Eq. 8, the zeros represent zero vectors with the same size as $h^{(T)}$. The shape of the derivative in Eq. 8 is a bit complex, since we are taking the derivative of a vector with respect to a matrix.

$\frac{\partial L^{(T)}}{\partial W_{hh}}$:

$$\frac{\partial L^{(T)}}{\partial W_{hh}} = \frac{\partial L^{(T)}}{\partial \hat{y}^{(T)}} \frac{\partial \hat{y}^{(T)}}{\partial p^{(T)}} \frac{\partial p^{(T)}}{\partial h^{(T)}} \frac{\partial h^{(T)}}{\partial W_{hh}} \tag{9}$$

$$= \sum_{t=1}^{T} \frac{\partial L^{(T)}}{\partial \hat{y}^{(T)}} \frac{\partial \hat{y}^{(T)}}{\partial p^{(T)}} \frac{\partial p^{(T)}}{\partial h^{(T)}} \frac{\partial h^{(T)}}{\partial h^{(t)}} \frac{\partial h^{(t)}}{\partial W_{hh}}, \tag{10}$$

where

$$\frac{\partial p^{(T)}}{\partial h^{(T)}} = W_{ph} \tag{11}$$

$$\frac{\partial h^{(T)}}{\partial h^{(t)}} = \prod_{j=t+1}^{T} \frac{\partial h^{(j)}}{\partial h^{(j-1)}} \tag{12}$$

$$\frac{\partial h^{(t)}}{\partial W_{hh}} = \begin{bmatrix} \frac{\partial h_1^{(t)}}{\partial W_{hh}} \\ \frac{\partial h_2^{(t)}}{\partial W_{hh}} \\ \vdots \end{bmatrix} = \text{sech}^2(W_{hx}x^{(t)} + W_{hh}h^{(t-1)} + b_n) \odot \begin{bmatrix} \begin{bmatrix} h^{(T)} & 0 & 0 & \cdots & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & h^{(T)} & 0 & \cdots & 0 \end{bmatrix} \\ \vdots \\ \begin{bmatrix} 0 & \cdots & 0 & 0 & h^{(T)} \end{bmatrix} \end{bmatrix} \tag{13}$$

Eq. 12 was given in the slides. Eq. 13 is again a bit complex, because we are taking the derivative of a vector w.r.t. a matrix. Since $\frac{\partial}{\partial x}\tan(x) = \text{sech}^2(x)$ we get the hyperbolic secant in Eq. 13. Each element of this hyperbolic secant (it is a vector in our case) is multiplied by the corresponding row vector of the matrix in the last term of the equation, as indicated by $\odot$.

The two derivatives, $\frac{\partial L^{(T)}}{\partial W_{ph}}$ and $\frac{\partial L^{(T)}}{\partial W_{hh}}$, are different in their temporal dependence. The former is only dependent on time step T, while the latter depends on all $t = 1, \ldots, T$. For a large number of time steps this may result in the vanishing or exploding of gradients (for the latter). For deep layers (towards the input), Eq. 12 becomes a product with many factors. If all these factors are slightly smaller or larger than 1, the product will quickly become very small resp. very large, with a result that these layers will stop learning.

Question 1.2

The vanilla recurrent neural network as specified by the equations is implemented without using high-level PyTorch building blocks in the file `vanilla_rnn.py`. A for-loop is used in the forward pass to step through time and for the backward pass we rely on automatic differentiation. The RMSProp optimizer was used for tuning the weights.

Question 1.3

Given palindromes of length T, we let the model learn to predict the last digit of the palindromes. The results are shown in Figure 1. Table 1 gives the parameter values that were used for this model. The figures show loss and accuracy of the model over the training process. The loss is calculated based on training batches, and the accuracy is measures based on an evaluation set of 4000 examples. For each run, training was stopped when an accuracy $> 99.95\%$ (rounded 100%) was achieved. If this was not achieved, the training stopped after 1000 iterations. Several seeds were tested, and these led to similar results as shown here. We see that for palindromes with $T > 13$, the model seems to fluctuate more and more for increasing T and for $T = 33$, it cannot learn the task perfectly anymore given the used set of parameters. This could be due to the fact that for large T, the models needs to "remember" a (too) long sequence of digits, which leads to problems as described in Question 1.1.

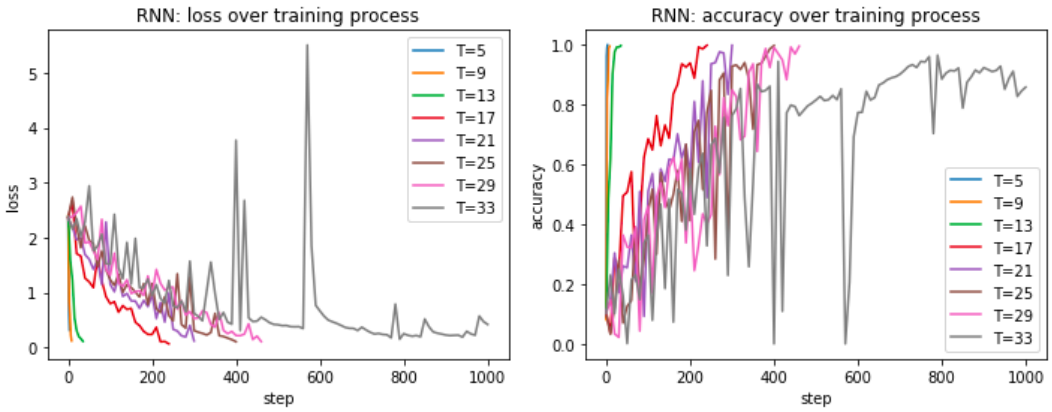

Figure 1: RNN performance of predicting the last digit of palindromes of length T. The parameters that were used for the model are given in Table 1. A validation set of size 4000 was used to measure accuracy over the training process.

Parameter	Value
input length	T
input dimension	10
number of classes	10
number of hiddens	128
batch size	128
learning rate	0.001
max norm	10

Table 1: The parameter values used for both the RNN and LSTM in the first part of the assignment. The input dimension was set to 10, since we are using one-hot vectors as input. This parameter should not be changed, as an embedding layer is not implemented in the model.

Question 1.4

A problem that might occur when using vanilla stochastic gradient descent (SGD), is that the model could fluctuate around a local minimum fairly easily. Weight updates are performed based on gradients. However, the pre-set learning rate might cause the update to overshoot the local minimum, resulting in a decreased or unchanged performance of the model. Using an adaptive learning rate this problem is reduced. As the learning rate gets smaller and smaller over the number of steps taken, the model finally converges more closely to the local minimum. The RMSProp optimizer makes use of an adaptive learning rate and is therefore able to converge faster and obtain better local minima than SGD. Another problem that might still arise is the constant change of direction of the update. This occurs when the previous update is not taken into account. By taking into account the previous update, or the current direction of descend, the model does not change direction as easily as before. You could speak of it as momentum: just like an object keeps momentum when it is pushed into another direction than it is propagating, a model could converge to a local minimum like a rolling ball converges to the bottom of a convex well, when it makes use of its previous updates. Adam optimizer has this implemented and is therefore also able to converge faster and obtain better local minima than SGD.

Question 1.5a

- *input modulation gate* $\mathbf{g}^{(t)}$
  Creates a vector of new candidate values to store in the cell state. The $\tanh$ activation function makes sure the values are between -1 and 1. This is useful, as the values together are centered around zero and this remains when adding new input to the cell state.

- *input gate* $\mathbf{i}^{(t)}$
  In this gate it is decided what to store in the cell state. The sigmoid activation function outputs values between 0 and 1 in order to distinguish what is kept and what is not. If a value from the input gate is 0, then the candidate value from the input modulation gate is not stored on the cell state. If the value from the input gate is 1, the value is fully stored. For any value between 0 and 1, the candidate value is scaled by this value and stored in the cell state.
- *forget gate* $\mathbf{f}^{(t)}$
  This gate specifies what will be forgotten from cell state. Again, a sigmoid is used, where a value of 1 means that the corresponding feature on the cell state will be fully kept, and a value of 0 means that it will be fully forgotten.
- *output gate* $\mathbf{o}^{(t)}$
  The output gate holds values that specify how much of each element in the cell state will be used for output. Again, a sigmoid is used where the values between 0 and 1 scale the cell state elements for output. In other words, if the value is 1, it will be fully used for output, and for 0 not at all.

Question 1.5b

Only the four gates described above hold trainable parameters in the LSTM cell. $\mathbf{p}^{(t)}$ is only the linear output mapping, but is not part of the core of the LSTM cell. First of all, the number of trainable parameters is independent of batch size and sequence length. Now, for each gate we have one parameter matrix that maps input $x^{(t)} \in \mathbb{R}^d$ to $\mathbb{R}^n$, where n is the size of the hidden and the cell states. For each gate we also have a parameter matrix that just transforms the hidden state, without reshaping it. Adding the bias term of the same size, i.e. n, we get that each gate has $(n \times d) + (n \times n) + n$ trainable parameters. We have 4 gates (that have trainable parameters), so:

$$\#\text{trainable parameters} = 4(n \times d + n \times n + n) \tag{14}$$
$$= 4n(d + n + 1) \tag{15}$$

Question 1.6

The LSTM network as specified by the equations in the assignment is implemented in the file `lstm.py`. The same experiments were performed as for the RNN model. The results are shown in Figure 2. Without changing any of the parameters (see Table 1), we found more promising results than for the RNN model. Where the RNN stuggled to get to perfection for large T, the LSTM seems to converge without problems to 100% accuracy. It is also impressive that it does so for each T within a maximum of 160 steps. This relatively large difference between the behaviour of the RNN and LSTM lies in the fact that LSTM makes use of multiple gates, whereas the RNN only uses 1. Due to the problem of vanishing or exploding gradients, the RNN performs badly for large sequences. The LSTM has a more stable backpropagation due to the use of a separate cell state.

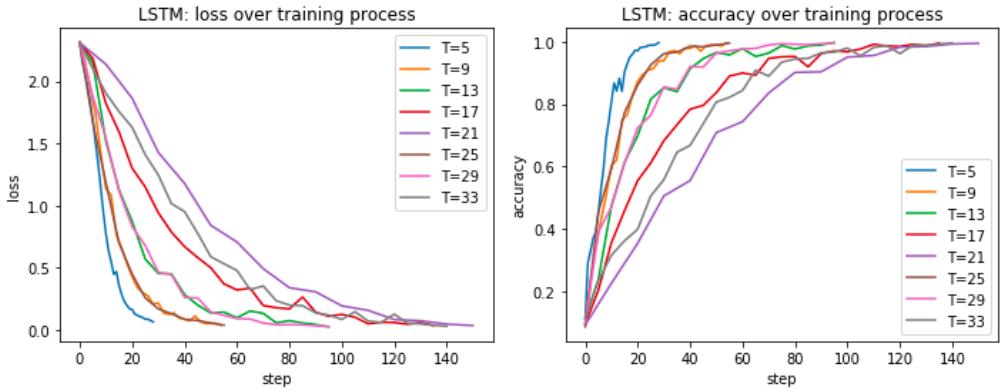

Figure 2: RNN performance of predicting the last digit of palindromes of length T. The parameters that were used for the model are given in Table 1. A validation set of size 4000 was used to measure accuracy over the training process.

Question 1.7

(As I left this question for last, I did not have much time to structure it better than this, sorry.) To obtain the gradients between time steps, parts of the code had to be altered a little bit to make it work. As such, I have used the altered code once for the purpose of this question, which was later changed back to its original. However, implemented parts for this are still included in the files, but as comments. Although I was convinced the implementation was correct, the results were surprising. In a bit of a redundant manner, I performed the forward pass of the models multiple times without updating, while keeping track of the gradients of a different time step each time. The norm of the gradient matrix was calculated for each time step. The results for both models are shown in Figure 3. The results show the opposite of what was expected. The gradients of the LSTM are mostly near zero, while those of the RNN do not seem to vanish or explode. So in this case, I expect either my implementation or my interpretation to be incorrect.

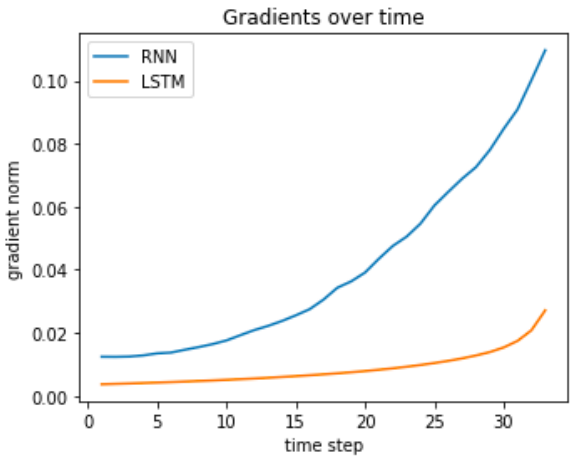

Figure 3: The norm of the gradient matrices between time steps.

2 Recurrent Nets as Generative Model

Question 2.1a

A two-layer LSTM network was implemented in the file `model.py`. The model was trained on Grimm's Fairy Tales, with text sequences of $T = 30$ at a time. The network was trained to predict the next character in the text sequence. When calculating the loss, the build-in CrossEntropyLoss function in pytorch was used. The model's loss and accuracy during training are showed in Figure 4. In Table 2 the model's parameters are given. The parameter values are left exactly as they were given as default values, since these values already provided results sufficient for the purpose of this question. As can be seen from the figure, the model scored around 70% in predicting characters at the end of its training.

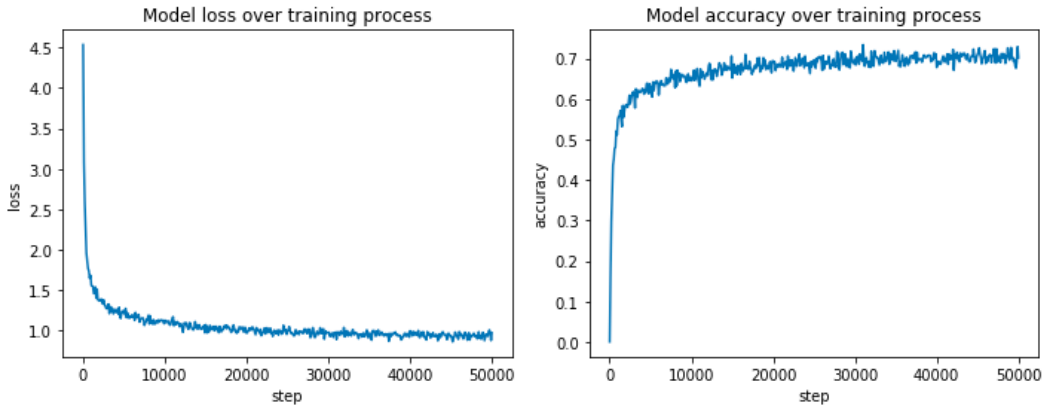

Figure 4: Loss and accuracy of the model, where both loss and accuracy are measured over the same batch for each specific step. The used model is specified by both Question 2.1a and the parameters shown in Table 2.

Parameter	Value
sequence length	30
input dimension	93
number of classes	93
number of hiddens	128
batch size	64
learning rate	0.002
max norm	5
learning rate decay	0.96
learning rate step	5000

Table 2: The parameter values used for both the model in the second part of the assignment. The input dimension and the number of classes are both equal to the vocabulary size of our book, since we are using one-hot vectors as input.

Question 2.1b

We used the model to generate sentences given a random first capital letter of the alphabet. Several generated sequences of text are are given below for different steps during training:

```
Step 0     : Jee
Step 250   : He the the the the the the the the the the the the the the t
Step 1750  : Good with the works and said, ‘‘We will be a strange of the s
Step 30000 : Foundation and feared her back into the forest, and the seco
Step 50000 : Fox is not safe for your servants.  Every day they were sitti
```

In step 0 the model had not learned anything yet, but produced a pronounceable "Jee" and a lot of following spaces. In step 250, we can already see some progression. Its first words! Further in training, big steps are taken as the model becomes less repetitive and uses more difficult words. As the training progresses no big steps are taken anymore, but possibly some rare words are formed more often. From the structure of the texts it is no surprise that the model learned with sequences of only 30 characters long. Within the first 30 characters, sentences make up somewhat of a structure, that we still cannot call correct, but if we look further, sentences start to make even less sense. However, the example of the quotation marks followed by "... and said, " shows that the model does sometimes produce realistic structures even further in the sentence.

Question 2.1c

(In this part I was struggling with saving my model, so as a temporary solution I let the model train, after which I let it produce text for the given temperatures, hard-coded in the file.) With random sampling using a temperature parameter τ, the output probabilities of the model are scaled down or up depending on the value of τ. More specific, if τ is high, the model becomes more certain of its choice as the highest probability class gets pushed away even further from the other classes. The smaller τ, the closer the probabilities will lie to each other, where for $\tau = 0$ the sampling process is completely random. The random sampling method as specified is the assignment was implemented in the code and the results for sequences of 100 samples are shown below for different temperatures:

- $\tau = 0.5$:

```
N"


After
that to’dom, clob-seed, Fam Endine careplay rasel!
Before they glass lovion again.  Lint
```

- $\tau = 1.0$:

```
RISENTIVE HAIR]


On the shoe!
‘‘_Three-Eyes, when the King commanded throw a
false clack, which sh
```

- $\tau = 2.0$:

```
Project Gutenberg-tm electronic works
and fruit down below.  So how it happen,’’ and how nut for the s
```

The results correspond with the explanation of the effect of the temperature on the sampling above, as the smaller τ, the more errors the model makes.

Question 2.2 (Bonus)

For this question, the model was given the sequence "Sleeping beauty is ", and it generated the following for a temperature of $\tau = 2.0$:

```
Sleeping beauty is only one of the country where the King came into the
wood, and they would have the commanded out the maiden, and the true
Bride.''

At last the man anger who are too alone.

The girl had all set off the town and drink out of the well and pears
desired.


[Illustration]

THE DONKEY CABBAGES
```

Although the model did not prove to be an expert on the story of sleeping beauty, it did try to fit in an illustration of donkey cabbages, which deserves some credit.

Question 2.3 (Bonus)

For beam search, the most promising characters are evaluated further, as a "what if" case.

3 Graph Neural Networks

Question 3.1a

As input a GCN takes a feature matrix, in this case $H^{(0)}$, and an adjacency matrix A, which describes the connections in the graph. At each layer, each node recieves feature respresentations as input from its neighbouring nodes, which are transformed to obtain a new representation for the node in question. The transformation is defined by Eq. 14 (in the assignment), where weight matrix W is learned and shared between nodes (not between layers). As it is possible that some nodes are not directly connected, it could take some transformations for the information of one node to reach the other. For this reason, a GCN layer can be seen as performing message passing over the graph.

Question 3.1b

The way some GCNs, and also the one in this assignment, are defined implicitly assume equal importance of self-connections versus neigbour-connections. This limits the flexibility of the model. However, we can easily overcome this by introducing a learnable trade-off parameter λ in Eq. 16, such that it becomes (1):

$$\tilde{A} = A + \lambda I_N \tag{16}$$

Another limitation, and maybe more of a drawback, is that for a GCN it is preferable to perform full-batch gradient descent, in order to cover all connections in a graph while training. However, this requires a lot of memory, sometimes more than a GPU fits. To overcome this, one could train the network on the CPU at the cost of longer, but still feasible, runtime.

Question 3.2a

The adjacency matrix A is an $N \times N$ matrix, where N is the number of nodes in the graph. $A_{ij} = 1$ if node i is connected to j, $A_{ij} = 0$ otherwise. Here the values $1, 2, \ldots, N$ for i and j correspond to nodes $A, B, \ldots, F$ respectively. A does not include self-connections, $\tilde{A}$ does include

self-connections.

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \tag{17}$$

$$\tilde{A} = A + I_N = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \tag{18}$$

Question 3.2b

It takes at least three steps (node to node) to get from C to E, so it takes three updates for information from C to reach E.

Question 3.3

Real-world applications where GNNs/GCNs have been applied:

- Text classification (2)
- Road networks (3)
- Ranking web pages (4)
- rgbd semanting segmantation (5)

Question 3.4a

As RNN-based models work with sequence representations, they are well suited in tasks where a certain order is present, such as predicting words in a sequence of text. GNNs work with graph representations where such an order is not necessarily present. GNNs will therefore work well in, for example, classification tasks of papers in a citation network, where order does not matter. In these two examples the model in question will probably outperform the other.

Sequential representation is more expressive for data where order matters most, and graph representation is more expressive for data where relations matter most.

Question 3.4b

In case both order and relations matter, for example in a social network (a graph that changes over time), it could be beneficial to use GNNs and RNNs in a combined model. The GNN is used to learn relations between nodes, while the RNN is used to learn the change of the graph over time.

References

[1] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *CoRR*, vol. abs/1609.02907, 2016.

[2] L. Yao, C. Mao, and Y. Luo, "Graph convolutional networks for text classification," *CoRR*, vol. abs/1809.05679, 2018.

[3] T. S. Jepsen, C. S. Jensen, and T. D. Nielsen, "Graph convolutional networks for road networks," *arXiv preprint arXiv:1908.11567*, 2019.

[4] F. Scarselli, S. L. Yong, M. Gori, M. Hagenbuchner, A. C. Tsoi, and M. Maggini, "Graph neural networks for ranking web pages," in *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence*, pp. 666–672, IEEE Computer Society, 2005.

[5] X. Qi, R. Liao, J. Jia, S. Fidler, and R. Urtasun, "3d graph neural networks for rgbd semantic segmentation," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 5199–5208, 2017.