Assignment 2: Recurrent Neural Networks and Graph Neural Networks

Florian Wolf
University of Amsterdam
Amsterdam, 1012 WX Amsterdam
`florian.wolf@student.uva.nl`

Abstract

This is the report for the second assignment of the the course 'Deep Learning' for the Master program 'Artificial Intelligence' at University of Amsterdam. Enjoy reading.

1 Vanilla RNN versus LSTM

1.1 Vanilla RNN

- Therory

Question 1.1
The Derivative of the loss over the shared output Weight matrix.

$$\frac{\partial L}{\partial W_{ph}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial p} \frac{\partial p}{\partial W_{ph}}$$

$$\frac{\partial L}{\partial \hat{y}_i} = -\sum y_i \frac{1}{\hat{y}_i}$$

This derivative returns a vector of size of the i entries.

$$\frac{\partial \hat{y}_i}{\partial p_{3i}} = \hat{y}_i(\delta_{ij} - \hat{[}y_j)$$

$$\delta_{ij} = \begin{cases} 1 & \text{for } i = j \\ 0 & \text{for } i \neq j \end{cases}$$

This is the derivative of the sofmax function using the Kronecka delta. It returns a Matrix of the final size of i and j.

$$\frac{\partial L}{\partial p_i} = p_i - y_i$$

$$\frac{\partial p}{\partial W_{ph}} = h$$

This derivative is a three dimensional Matrix with h along the diagonal.

Putting all the equations together we get:

$$\frac{\partial L}{\partial W_{ph}} = (p - y_i)h_j \tag{1}$$

Now we derive the Loss over the shared hidden layer weight matrix:

$$\frac{\partial L}{\partial W_{ph}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial p} \frac{\partial p}{\partial h} \frac{\partial h}{\partial W_{hh}} \tag{2}$$

$$\frac{\partial p}{\partial h^T} = W_{ph} \tag{3}$$

$$\frac{\partial h^T}{\partial W_{hh}} = (1 - \tanh(W_{hx}x^{(T)} + W_{hh}h^{(T-1)} + b_h))(\mathbb{I} \times h^{T-1} + W_{hh}\frac{\partial h^{T-1}}{\partial W_{hh}}) \tag{4}$$

$$\frac{\partial h^2}{\partial W_{hh}} = (1 - \tanh(W_{hx}x^{(T)} + W_{hh}h^{(1)} + b_h)(\mathbb{I} \times h^1) \tag{5}$$

The same applies for the gradient of W_{hx}.

$$\frac{\partial L}{\partial W_{hx}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial p} \frac{\partial p}{\partial h} \frac{\partial h}{\partial W_{hx}} \tag{6}$$

$$\frac{\partial h^T}{\partial W_{hx}} = (1 - \tanh(W_{hx}x^{(T)} + W_{hh}h^{(T-1)} + b_h))(\mathbb{I} \times x^{T-1}) \tag{7}$$

For the derivative of the $\tanh$ we use the chain rule and for further derivation the product rule. This derivation loops on until the initial hidden vector is reached, which does not depend on W_{hh}. The expression $\mathbb{I} \times h^{T-1}$ specifies that a four dimensional matrix of ones is matrix multiplied with the hidden vector h^{T-1}. The final result is of the same shape as the weight matrix W_{hh}.
The gradient of W_{ph} and W_{ph} depend only on the same timestep and are independent of the timesteps before. The gradient of W_{hh} depends on all previous timesteps. This means that the calculation is much heavier and much more variables from the forward pass need to be stored. One problem that might occur because of having to backtrack too many timesteps is 'vanishing gradient'.

- Implementation in PyTorch

Question 1.2

The Vanilla RNN was implemented using PyTorch.

Question 1.3

The sanity check with an input length of 5 resulted in an mean prediction accuracy over 10000 training epochs of 96.355% with a standard deviation of $9,034\%$.

For significant results each experiment was run 3 times. The following figure shows the loss over timesteps. To smooth the plot the the data was averaged every 100 timesteps.

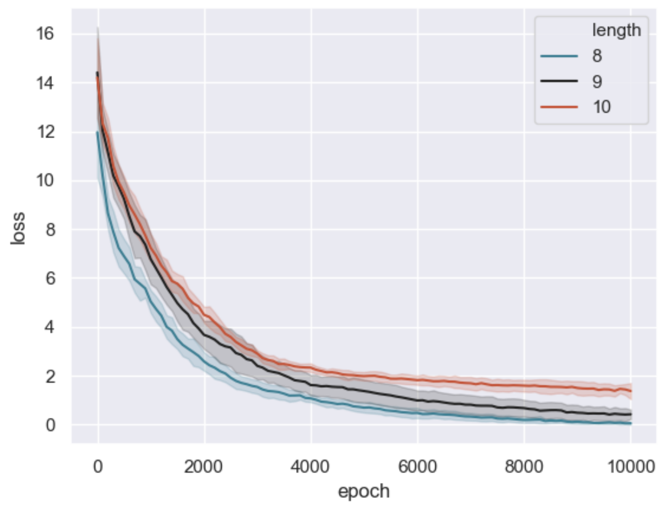

Figure 1: Loss over timesteps for RNNs trained on different palindrome length.

The loss converges at about 4000 timesteps. Therefore the presented accuracy only shows results above this timestep threshold. The violin plot shows each experiment for the input length of 5 to 13 in steps of two.

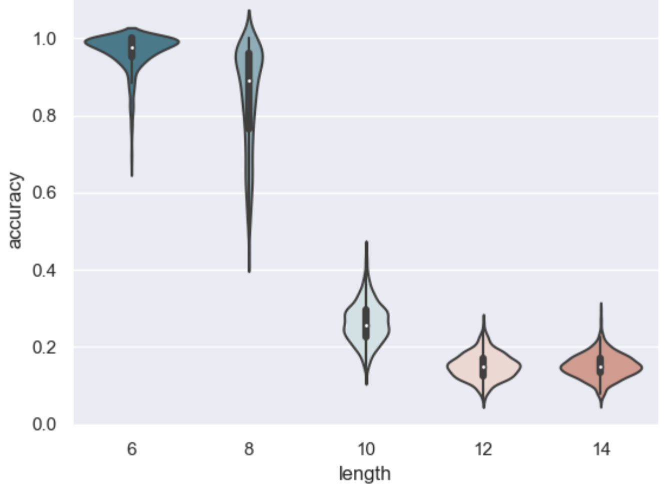

Figure 2: Accuracy distribution over length of palindrome for the RNN.

Question 1.4
One way to find the minimum of the loss function is to use Stochastic Gradient Decent (SGD). The value α is the factor of the update term. For higher values the model learn faster but doesn't converge to a local minimum of the loss, Lower values slow the learning down so much that the minimum is never reached. The adaptive learning rate adjusts the learning rate over time. There are various methods, one of them is to divide the learning rate by the number of epochs.

$$W^{k+1} = W^k - \frac{\alpha}{1 + i_{epoch}} \nabla f\left(W^k\right) \tag{8}$$

Every epoch of SGD takes a step in direction of the gradient. This causes oscillate as it approaches the local minimum and restricts the use of lager learning rates. The Momentum technique minimizes these oscillations. Instead of updating each step with the current gradient the Momentum technique proposes to update using the moving average of the previous gradients. This way the oscillations get averaged out. The rate between previous and current gradient is defined by the hyperparameter β. A common value is $\beta = 0.9$. One variation of the Momentum technique is:

$$\begin{aligned} Z^{k+1} &= \beta Z^k + (1-\beta)\nabla f\left(W^k\right) \\ W^{k+1} &= W^k - \alpha Z^{k+1} \end{aligned} \tag{9}$$

1.2 Long-Short Term Network (LSTM) in PyTorch

- Understanding

Question 1.5

a) Explain:
input modulation gate g: This gate returns candidate data to add to the information stream. It uses the tangens-hyperbolicus function.

input gate i: This gate scales the data returned by the above gate to match the amount of added information with the amount of forgotten information in the forget gate.

forget gate f: The forget gate f uses the sigmoid activation function to decide how much of the old information is passed on or forgotten.

output gate o: This gate also uses the sigmoid function to scale the output of the layer.

Discuss non-linearity:
The range of the sigmoid function is $0 - 1$ this is similar to "all or nothing". Multiplying with the sigmoid function fits the purpose of scaling down the product.
The tangens-hyperbolicus function ranges from -1 to 1 and the mean is 0. This fits the purpose of adding or subtracting. If this activation function was used as multiplication the chances of multiplying by a number close to o would be high and the problem of vanishing gradient would remain unsolved.

b) Calculate number of parameters:
$\mathbf{x} \in \mathbb{R}^{T \times d}$
T: sequence length
d: feature dimension
n: number of units
m: batch size
c: number of classes

The output of gate p is $c \times 1$. This specifies the first dimension of the weight matrix. The second dimension will be derived later on. Therefore:
W_{gx}: $c \times n$
b_{gx}: $c \times 1$
Each LSTM layer takes one part of the input sequence length T.
Because of the summation in gate c the hidden gate h has the output dimension $n \times 1$. The dimensions of gate g, i, f and o match.
Therefore the remaining weights and bias have the dimensions:
W_{xx}: $n \times d$
W_{xh}: $n \times n$
b_x: $n \times 1$

Total amount of parameters: $c \times (n+1) + 4n \times (n+d+1)$

- Implementation in PyTorch

Similar to the RNN model a version of LSTM was implement using low-level PyTorch features. The model performed better than the RNN in terms of memorizing palindromes of higher length. The loss converges after 2000 steps but shows severe outliners.

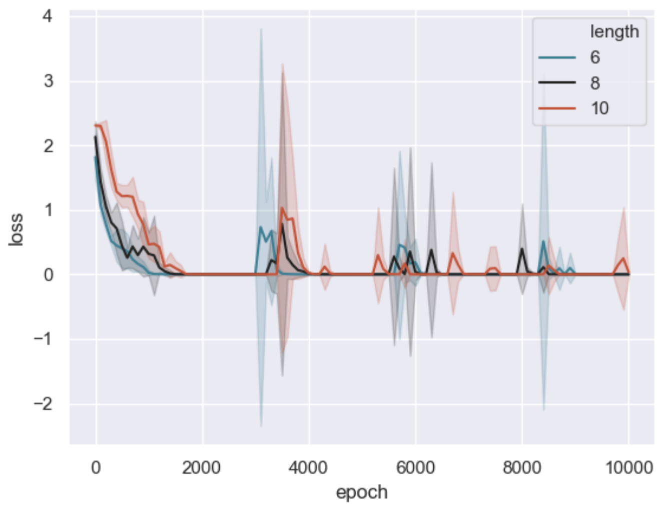

Figure 3: Loss of the LSTM model for palindromes of different length over timesteps.

For the accuracy violin plot only the results up to the threshold of 2000 timesteps have not been considered. Different then expected the LSTM model outperforms the RNN model only by two palindrome digits. This might be caused by errors in the implementation.

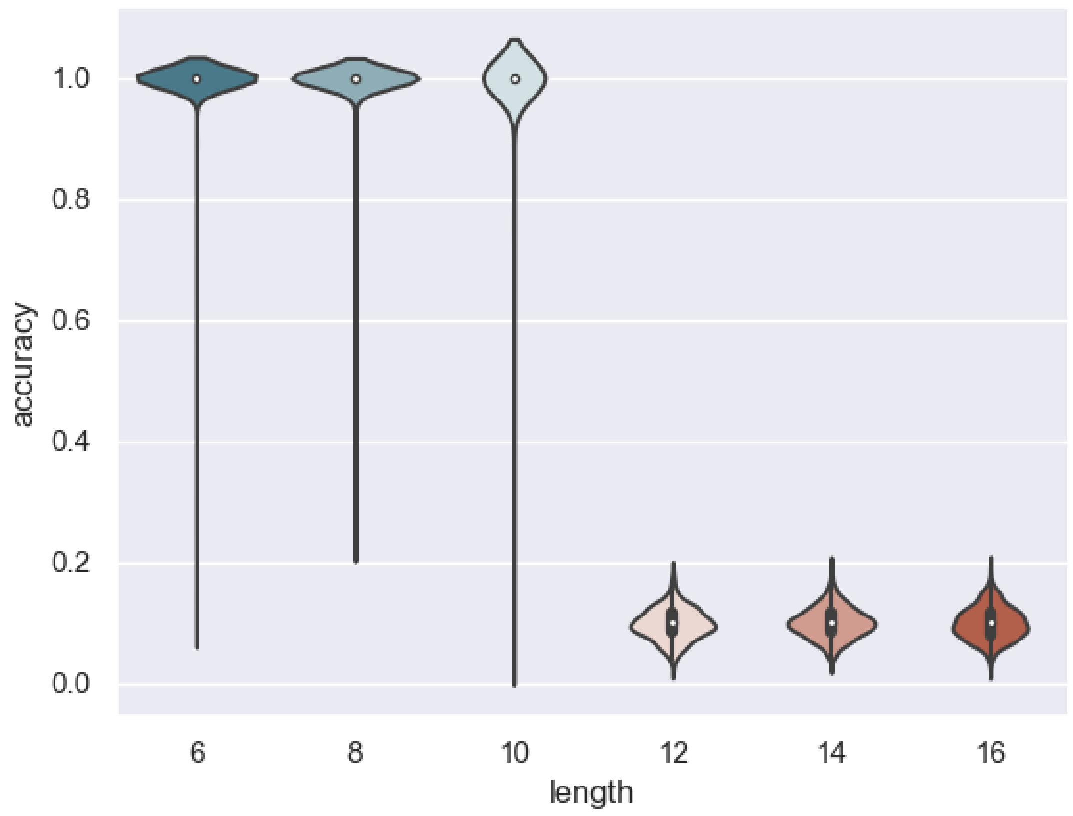

Figure 4: Accuracy distribution of the LSTM model for palindrome of different length.

2 Recurrent Nets as Generative Model

2.1 Experiments

Question 2.1

a) The LSTM was implemented using the torch module `nn.LSTM()`. The module provides input arguments for the networks number of layers. This parameter was set to 2.
To calculate the Loss the target as well as the input batch vectors were concatenated. The torch module `nn.CrossEntropyLoss()` reduces as default the various inputs by taking the mean. Inside the forward pass of the LSTM module the input is transformed to a one-hot vector of the size of the character vocabulary. As optimized the torch `nn.Adam()` optimizer is used. Further the learning rate is optimized using learning rate decay over the epochs. The model is trained on the book 'Alice's Adventures in Wonderland'.
With this setup and the default hyperparameters the model reaches an accuracy of 80%.

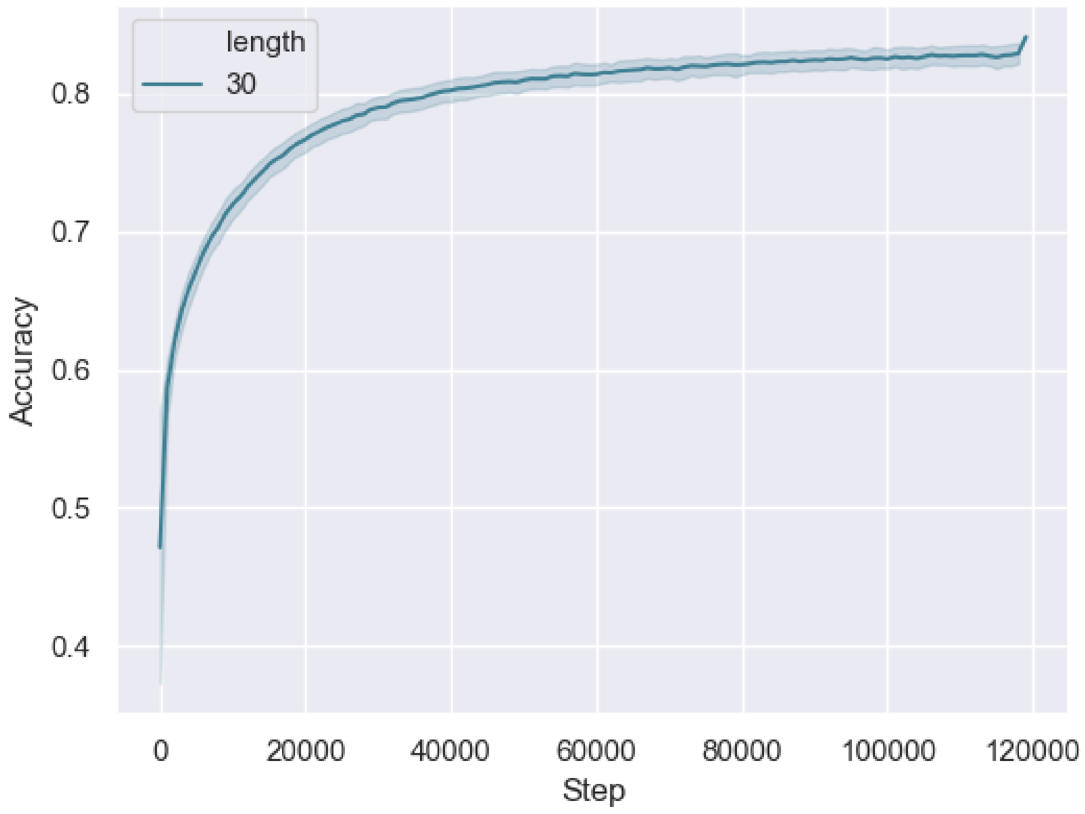

Figure 5: Accuracy of the LSTM model trained with default hyperparameters.

b) By using a random input the model returns a new character. Iterating this process by recursively feeding the output and the hidden state of the LSTM as input new text is generated. For this experiment, text is generated at different timesteps during the training. This gives a more applied insight on the models performance as the training accuracy. Text was generated every 100 steps during 50 epochs The results are:

epoch	Generated Text
1	`e the the the the the the the`
2	`ueth, and the Mock Turtle a li`
5	`I won’t indeed!’ said the Mock`
33	`use doing out here? Run home`
50	`ing said, turning to Alice for`

The learning curve of the model is clearly visible. The first few epochs it learns how to generate words and putting words together. Towards the end the generated text is not only grammatically correct but also makes sense. The first letter has been randomly generated and the experimental setup missed out on printing this character as well. For the sentence generated in epoch 50 the initializing character was with very high probability a ’K’. Which would make the sequence:

```
King said, turning to Alice for
```

This seems to be memorized from the original training data. In fact it is not and was made up by the model. The highest similarity found was this sentence:

```
King said, turning to the jury.
```

c) The following sentences were generated using random multinomial sampling with different temperature values.

Temperature	Generated Text
0.5	`lway; no,' adding lat.  T`
1	`!'  (Think of enough!'  all late`
2	`Then the reason is a good deal`

For lower temperature values the output is complete non sense. For a temperature of 2, meaning less randomness the results return to be use full and in contrast to the greedy policy show less similarity to the book.

2.2 Bonus Questions

For this section the model has been tuned. The number of hidden layers and unit were rises as well as the batch size in order to increase the model complexity and training time. Fine tuning in from of dropout and learning rate decay has been applied as well. The tuned model outperformed the default model by an increase of accuracy by 10% as shown in the following plot.

LSTM.png

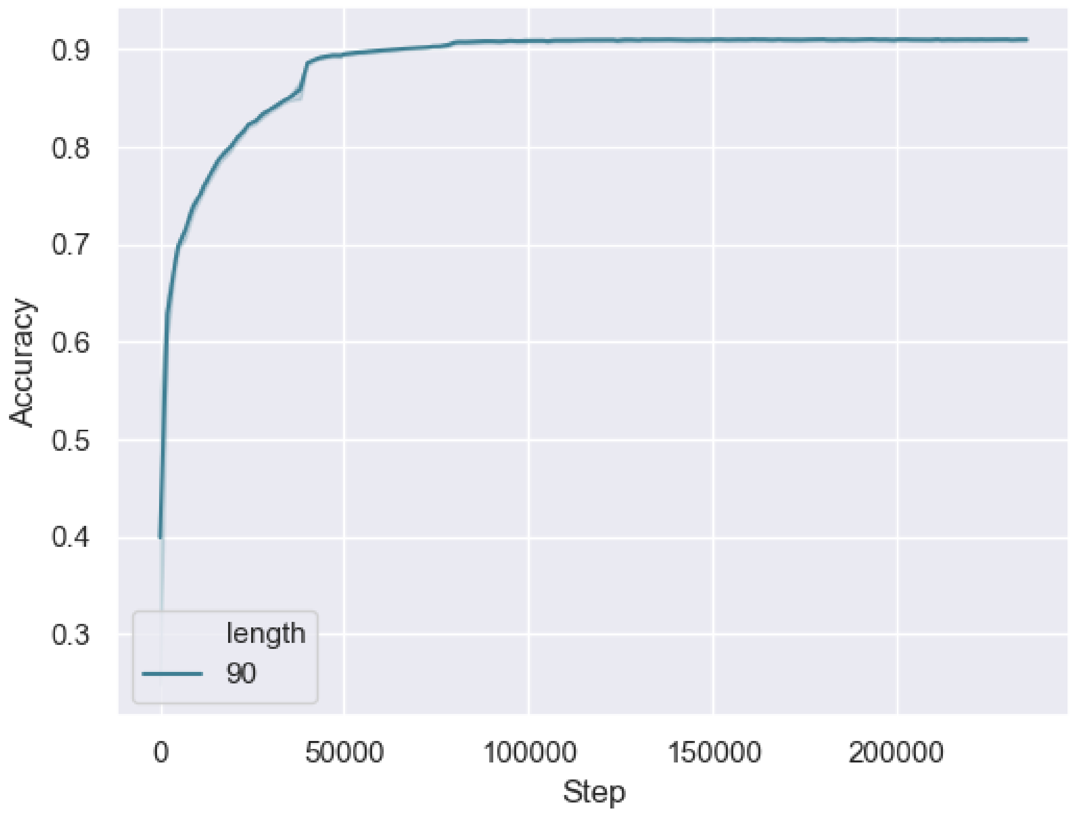

Figure 6: Accuracy of the tuned LSTM model over timesteps.

The model generated strings after getting the initial input sequence `Alice`. Out of interest the generated sequence length has been set to 111 characters.

Generated Text initiated with 'Alice'
Alice said to herself, 'to be going messages for a rabbit! I suppose Dinah'll be sending me on messages after all
Alice took up the fan and gloves, and, that buted out it haste on the book of the hall, but her head was so full o
Alice along in a minute. Alice began to feel voice, and shook the sompinges, and conesily itself, with the coo

The generated sentences show that the model learned to apply grammar and context understanding. Some words are made up and the punctuation shows errors. Overall an impressive result.

3 Graph Neural Networks

3.1 GCN Forward Layer

GCN are Convolution Neural Networks which use Graphs instead of of Matrix input. Graphs are defined by nodes and edged. A node carries specific information as feature and has edges which are the connections to other nodes. The information can be converted to a one-hot graph and the connections representd in a connection matrix. Each layer updates the value of the nodes by applying a weight matrix to itself and all its neighbors, summing them up and adding them to itself.

Question 3.1

a) Question: `Describe how this layer exploits the structural information in the graph data, and how a GCN layer can be seen as performing message passing over the graph.`

$$H^{(l+1)} = \sigma\left(\hat{A}H^{(l)}W^{(l)}\right) \tag{10}$$

The matrix $\hat{A}$ specifies the connections between the nodes. The matrix H contains all the current state feature of each node. H is initialized with the features of each node and then updates over each layer. The layer individual weight W is applied H which means to each node value and each node is summed up with its neighbors. The next state is reached by applying a non-linearity. In each layer, to each node the information of all its neighbors is added, this can be compared to 'message passing'.

b) Question: `How many GCN layers at most would you need to stack to propagate to every node's embedding the information (features) from nodes 3 hops away in the graph?`

To propagate the information of a node three hops away a three layer network is required. Each layer passes on the information of the each node to all its neighbors. To pass the information on over three node, three steps and therefore three layers are required. More than two layers are uncommon since the information by multiplying a higher number of weight matrices and non-linear activation functions is most likely to turn into noise.

3.2 Applications of GCN

This section discusses the applications of GCN in real-world. Further the differences and possible combinations with RNN are presented. Note that GNN is a generalizing term for all variations of Graph networks not restricting to the model presented in 3.

Question 3.2

Social-network information:
Social media platforms offer a graph structure of human relationships. Each account contains various usefull information features and even the connections between users can be weighted differently. Neither Instagram nor Facebook are open source but it would come as a surprise if the idea idea of GNN would not have been exploited yet.

Chemistry:
GNN have been used to find new molecules with similar properties as the input molecules. Another application is the prediction of attributes like side-effects with the new molecule as input.

Stock-exchange:
GNN can be used to exploit the dependencies of different different stocks or markets beween each other.

Computer Science:
GNN has been trained to debug code. GNN can capture the relation of variables, classes, methods and attributes between each other.

3.3 Comparing to RNN

Question 3.3

a) Question: `Consider using RNN-based models to work with sequence representations, and GNNs to work with graph representations of some data. Discuss what are the benefits of choosing either one of the two approaches in different situations. For example, in what tasks (or datasets) you would see one of the two outperform the other? What representation is more expressive for what kind of data?`

   The main difference between RNN and GNN is the type of data they can be applied on. RNN relys on sequential data. Everytimestep uses previous data to predict upcoming data. Every input is only connected to its the previous and the following timestep. It can be represented as one string of information.
   GNN relies on dependencies not represantable in eulidean space. Each node can have multiple connections. The network is trained to capture the dependencies between each node. Instead of a sequentall information flow GNN uses a dataset of parallel information features. Catching up on the example of Facebook, one person could be a criminal and has friendship connection with certain other people. A netork trained on social networks of previous criminals could predict which friends are likely to be criminal as well. Or to lighten up this example the feature 'talented Data Scientist' can be used instead.

b) Question: `Think about how GNNs and RNNs models could be used in a combined model, and for what tasks this model could be used.`

   A useful combination of both would be for stock market prediction. The GNN could forecast the interaction of different markets and the RNN predict a rise or fall according to historical data of the market.

4

5 Conclusion

YES
YOU
CAN

References