Assignment 2. Recurrent Neural Networks and Graph Neural Networks

Max Bos
10669027
University of Amsterdam
maxn.bos@gmail.com

1 Vanilla RNN versus LSTM

1.2 Vanilla RNN in PyTorch

Question 1.1

$$\begin{aligned}\frac{\partial \mathcal{L}^{(T)}}{\partial \mathbf{W}_{ph}} &= \frac{\partial -\sum_{k=1}^{K} \mathbf{y}_k \log \hat{\mathbf{y}}_k}{\partial \mathbf{W}_{ph}} \\ &= \frac{\partial -\sum_{k=1}^{K} \mathbf{y}_k \log \text{softmax}\left(\mathbf{W}_{ph}\mathbf{h}^{(t)} + \mathbf{b}_p\right)}{\partial \mathbf{W}_{ph}} \\ &= \frac{\partial \mathcal{L}^{(T)}}{\partial \hat{\mathbf{y}}^{(t)}} \frac{\partial \hat{\mathbf{y}}^{(t)}}{\partial \mathbf{p}^{(t)}} \frac{\partial \mathbf{p}^{(t)}}{\partial \mathbf{W}_{ph}} \\ &= \frac{\partial \mathcal{L}^{(T)}}{\partial \hat{\mathbf{y}}^{(t)}} \frac{\partial \hat{\mathbf{y}}^{(t)}}{\partial \mathbf{p}^{(t)}} \mathbf{h}^{(t)T}\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{L}^{(T)}}{\partial \mathbf{W}_{hh}} &= \frac{\partial -\sum_{k=1}^{K} \mathbf{y}_k \log \text{softmax}\left(\mathbf{W}_{ph} \tanh\left(\mathbf{W}_{hx}\mathbf{x}^{(t)} + \mathbf{W}_{hh}\mathbf{h}^{(t-1)} + \mathbf{b}_h\right) + \mathbf{b}_p\right)}{\partial \mathbf{W}_{hh}} \\ &= \frac{\partial \mathcal{L}^{(T)}}{\partial \hat{\mathbf{y}}^{(t)}} \frac{\partial \hat{\mathbf{y}}^{(t)}}{\partial \mathbf{p}^{(t)}} \frac{\partial \mathbf{p}^{(t)}}{\partial \mathbf{h}^{(t)}} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{W}_{hh}} \\ &= \frac{\partial \mathcal{L}^{(T)}}{\partial \hat{\mathbf{y}}^{(t)}} \frac{\partial \hat{\mathbf{y}}^{(t)}}{\partial \mathbf{p}^{(t)}} \frac{\partial \mathbf{p}^{(t)}}{\partial \mathbf{h}^{(t)}} \mathbf{h}^{(t-1)T}\end{aligned}$$

The former gradient depends on the hidden state of the current timestep, while the latter gradient depends on the hidden state of the previous timestep. The latter gradient could lead to an exploiding gradient given the accumulation of gradients unrolled over hundreds of input time steps. Also, the latter gradient could lead to a vanishing gradient when propagating the loss through a large number of timesteps.

Question 1.2

I have implemented the vanilla recurrent neural network in the file `part1/vanilla_rnn.py`. The weight and the bias parameters are initialized as tensors sampled from a continuous uniform distribution using the `torch.Tensor.uniform_` functionality. The initial hidden state matrix is filled with zeros.

The implementation of `part1/train.py` was finished for running the optimization procedure. The file contained the line `torch.nn.utils.clip_grad_norm(model.parameters(),`

`max_norm=config.max_norm)`. This functionality is applied to counteract the exploding gradient that happens to occur in vanilla recurrent neural networks.

Question 1.3

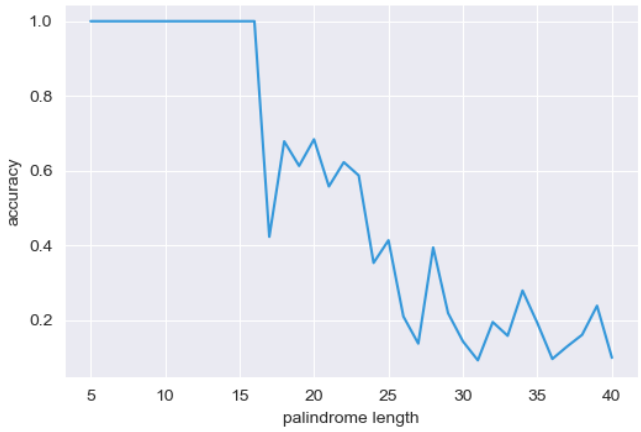

Figure 1: Accuracy for training a vanilla RNN on palindromes ranging from lengths 5 to 40

Experiments have been performed for palindromes of lengths ranging from `T=5` to `T=40`. For each palindrome length the network was trained 3 times until convergence, where convergence is implemented as a loss lower than 0.001. The accuracy for each training run was averaged over the last 10 batches, and the overall accuracy per palindrome length was averaged over the returned accuracy per training run. Figure 1 depicts the achieved accuracies for each palindrome length. Up until a palindrome length of 16 the RNN achieves an accuracy of 1.0, after which the RNN starts performing gradually worse.

Question 1.4

Vanilla stochastic gradient descent does not guarantee good convergence, since it does not work well with finding the global minima in a non-convex function, it uses a fixed learning rate throughout the optimization process, which can cause the loss function to fluctuate around the minimum or even to diverge if one chooses a learning rate that is too large, and the same learning rate applies to all parameter updates while it could be that some parameters require different updating. One can utilize learning rate schedulers to schedule learning rate changes, but the changes have to be manually designed in advance and do not automatically adapt to the dataset's characteristics.

Now, vanilla SGD has trouble navigating through areas of local optima, when the local optimum is an area where the surface curves much more steeply in one dimension than in another, the use of momentum helps accelerate vanilla SGD in the relevant direction and dampens oscillations. RMSprop is an adaptive learning rate method that utilizes momentum and that uses and adapts a different learning rate for each parameter based on the slope of the gradient. The learning rate adapts to slow the updating down when necessary, e.g. in the case of reaching a minimum. Adaptive Moment Estimation (Adam) is another method that computes adaptive learning rates for each parameter, with the difference that Adam uses a method similar to momentum that prefers flat minima in the error surface.

1.3 Long-Short Term Network (LSTM) in PyTorch

Question 1.5

a) **forget gate** $f^{(t)}$ The forget gate decides what information to forget from the cell state. What information and how much of it to forget is calculated by multiplying a dedicated learned parameter matrix by the current input and the previous hidden state. The result is propagated

through a sigmoid activation function, yielding a number between 0 and 1 for each number in the cell state, which is perfect for quantifying the proportion to keep of a current value in the cell state.

input modulation gate $g^{(t)}$ The input modulation gate calculates new candidate values for the cell state. The tanh activation function is used here because the cell state includes values that range from -1 to 1, and is used to overcome the vanishing gradient problem.

input gate $i^{(t)}$ The input gate decides what information from the input modulation gate and how much of it is actually added to the cell state. This is achieved by calculating a matrix of values between 0 and 1, result from using the sigmoid activation function, to quantify the proportion to keep from each number in the input modulation gate.

output gate $o^{(t)}$ After the previous cell state is mutated by the forget gate and increased by the result of the input gate, the output gate decides what information of the current cell state and how much of it to output. It uses a sigmoid activation function to yield values between 0 and 1, which act as the proportion to keep of each number in the current cell state.

b) The formula for the total number of trainable parameters in the LSTM cell as defined in the assignment is: $4(nd + n^2 + n)$, where n is the number of hidden units, and d is the input dimension. If we would also take into account the layer for calculating the probabilities, the formula would be $4(nd + n^2 + n) + (cn + c)$, where c is the number of output classes.

Question 1.6

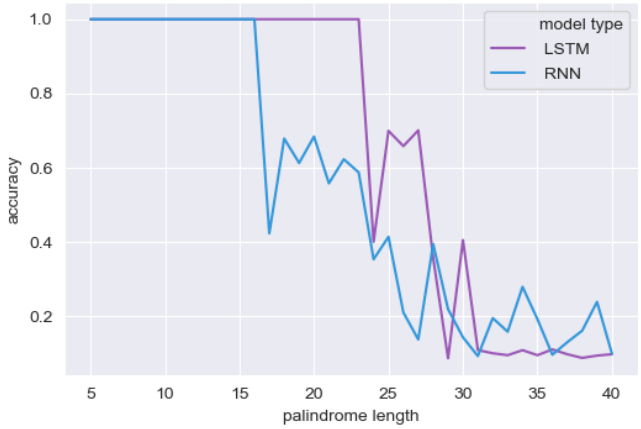

Figure 2: Accuracy for training a vanilla RNN and LSTM on palindromes ranging from lengths 5 to 40

I have implemented the LSTM network as specified in the `lstm.py` and `train.py` files.

Experiments have been performed for palindromes of lengths ranging from `T=5` to `T=40`. For each palindrome length the network was trained 3 times until convergence, where convergence is implemented as a loss lower than 0.001. The accuracy for each training run was averaged over the last 10 batches, and the overall accuracy per palindrome length was averaged over the returned accuracy per training run. The learning rates were set to 0.001, 0.0007, and 0.0004 for the length ranges 5-14, 15-29, and 30-40, respectively.

Figure 2 depicts the achieved accuracies for each palindrome length for the LSTM network and the vanilla RNN. Noticably, the LSTM outperformed the RNN for longer palindrome lengths, the LSTM was able to achieve an accuracy of 1.0 until a palindrome length of 23, while the RNN until 16. This would be due to the LSTM being able to learn more long term dependencies. The LSTM yielded lower accuracies for palindrome lengths than the vanilla RNN in the range 30-40, this could be due to the low learning rate that was set for the LSTM in that range. It could be that the LSTM will perform better if it uses a higher learning rate of 0.0007 in that range.

2 Recurrent Nets as Generative Model

Question 2.1

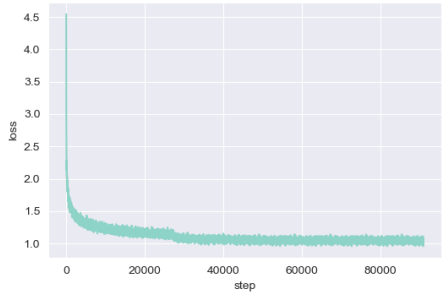

Figure 3: Loss curve for training a two-layer LSTM network on sentences of length 30

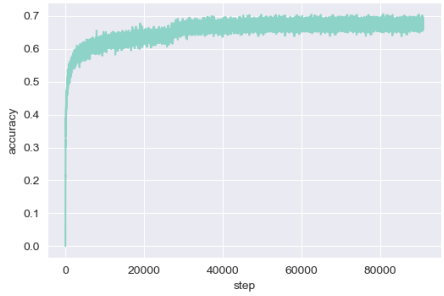

Figure 4: Accuracy curve for training a two-layer LSTM network on sentences of length 30

a) I have implemented a two-layer LSTM network to predict the next character in a sentence by training on sentences from a book. The input batches are first forwarded through an embedding layer, after which the output from the embedding layer is forwarded through the LSTM network. It uses the RMSProp optimizer for tuning the weights, and a learning rate scheduler to decay the learning rate of each parameter group by 0.1 every 2 epochs.

   In specific, I have trained the network on sentences of length `T=30` from the book 'The Adventures of Sherlock Holmes by Arthur Conan Doyle'. The network was trained for 10 epochs, where each epoch has a maximum step size of 9090. After every second epoch, the learning rate was decayed. Figure 3 and 4 show the loss and accuracy curves, respectively. The network improves significantly after the first learning rate adjustment in the third epoch. After which, the network converges at around an accuracy of 0.7.

b) I have implemented code for generating sample sentences of length `T=30` every 100 steps. Below is a sample of the generated sentencens:

   Step 0 I
   Step 100 On the the the the the the the
   Step 900 But the poor and the poor and
   Step 3300 Very strange and should be a s
   Step 11400 7 I have had a small paper and
   Step 21400 . It is a small fact that he h
   Step 49800 xperience that I have no doubt
   Step 56200 quite concerned in the street.
   Step 88700 5 friends with a strong presum
   Step 91000 ve the deep the streets of the

   At the beginning of the training, the network seems to have learned the most common single characters in the book, which are in this case the pronoun `I`, and the space character. At step 100 it has learned the most common combination of characters, which would be determiners, such as `the`, and prepositions, such as `on`. At step 3300, it has learned the structure of a sentence. The network starts of learning short term dependencies and learns more long term dependencies as the learning progresses.

c) The sentence generation functionality is extended to support random character sampling with a temperature parameter that regulates the greediness. I have generated sentences for the temperature values $T \in \{0.5, 1.0, 2.0\}$. The network starts out with generating the following sentences at step 0:

   `T=0.5` "Gb2»""N7YDhnXR0hsWe' yXDs
   "
   `T=1.0` iHm'j-A(%ecOrE)1Oq$NX!i4ly/
   `T=2.0` "K/v@

UtX8/bZeH?I©uN¿[bïR(rï©"

At step 100 the network seems to have learned most common words, which in this case are determiners, such as `the`, pronouns, such as `I`, and prepositions, such as `to`. Also, it seem to have knowledge of starting a new sentence with a capital letter after a question mark:

`T=0.5` ? The fad to the drere I de to
`T=1.0` "?Jtgale dit the, ans wo out-st"
`T=2.0` "? 1jyxIv.!""-LYexfCorbTW3es mit"

At step 500, the network seems to have improved its knowledge to such an extend that the generation at a higher temperature starts to yield more realistic sentences:

`T=0.5` que which I was the set that I
`T=1.0` "quls the
and he untore,
and th"
`T=2.0` "q(Ed; Wake;'xacoFEN
I IK[FrnIq"

At step 6500:

`T=0.5` Lord St. Simon has almost comp
`T=1.0` "Lask that it were point and
fr"
`T=2.0` "Look anxoyy feared my monogo."""

At step 91075, we see that the generated sentences at temperature 2.0 samples include more realistic characters:

`T=0.5` "ve the truth, to wait upon the"
`T=1.0` ve been a little certaining in
`T=2.0` "ven awn immecilue!"" He streets"

Concludingly, as the learning progresses, the network learns increasingly more about dependencies, resulting in higher sampling temperatures yielding more realistic combinations.

Bonus Question 2.2

An experiment has been performed for generating a text of 1000 characters, by starting with the text with the sentence: `"You are sure that she has not sent it yet?"`.

Generated, T=0.5 "You are sure that she has not sent it yet?"said he.

"He is silked, and then he was at the time and of the morning, and the matter which led the little of the lady to him, and that he had saved it. It must be the dead of his father, and the lady cold in the corner of the business, but I could see that the room. ""It is hident here and the doctor had been seen a long and seen a little to me."

"What seemed to be a shadow pass be too much sense to see you who had brought in a man with a shade of a sum by trying here. He was only so. I shall be the seats and bustle at the bottom the ground or distribution of the door which he had suffered to find the ship to me that he was a little bald in the facts are really into the chairman of which you have had a man who is most recovered the problem, and so small that I am not explain the great work is come to him to remain from the facts which makes his son, that is a man who ission that I am afraid that you will be the matter. I heard the man who will leave an admirable cry out of the sort "

Actual "You are sure that she has not sent it yet?"

"I am sure."

"And why?"

"Because she has said that she would send it on the day when the betrothal was publicly proclaimed. That will be next Monday."

"Oh, then we have three days yet," said Holmes with a yawn. "That is very fortunate, as I have one or two matters of importance to look into just at present. Your Majesty will, of course, stay in London for the present?"

"Certainly. You will find me at the Langham under the name of the Count Von Kramm."

"Then I shall drop you a line to let you know how we progress."

"Pray do so. I shall be all anxiety."

"Then, as to money?"

"You have carte blanche."

"Absolutely?"

"I tell you that I would give one of the provinces of my kingdom to have that photograph."

From the generated text, we can conclude that the network places punctuation symbols correctly. It has learned when a new sentence starts, after a dot symbol, and to start a new sentence with a capital letter. Only the generated sentences in this example are long compared to the acutal sentences in the text snippet. In specific, it has generated a sentence that counts 441 characters. The model has learned short term dependencies and moderately long term dependencies, to complete quoted sentences for example. The model has not learned correct usage of semantics, or how to create coherent text.

3 Graph Neural Networks

3.1 GCN Forward Layer

Question 3.1

a) The layer in the Graph Convolutional Network architecture as defined in the assignment consists of an adjacency matrix A, that describes the structure of a graph, an input matrix H, that contains feature values per node in the graph, and a matrix W, that contains weights and acts as a mapping from the current input dimension to the input dimension of the next layer. This layer could thus be seen as performing message passing over the graph, since the result of AH is an $N \times d$ matrix that contains the sum of the values at each input dimension from all direct parent nodes plus the node itself. Subsequently, this data is transformed to the input dimension of the next layer to be able to pass it through to the next layer. So one calculation of the layer can be seen as input data from each node in the graph being forwarded to their direct neighbors.

b) Forwarding over one GCN layer constitutes 1 hop of information through the network. So, to be able to forward information 3 hops away in the network one would need to stack 3 GCN layers.

3.2 Application of GNNs

Question 3.2

GNNs could be applied to real-word problems with datasets that come in the form of graphs or networks, such as social networks, knowledge graphs, and protein-interaction networks. For example, interaction networks can be trained to reason about the interactions of objects in a complex physical system, or be applied to learn about existing molecular structures aswell as learning new chemical structures. Also, GNNs can be applied in various NLP tasks such as text classification, and machine translation. Lastly, GNNs are also being applied to image classification tasks and combinatorial optimization problems.

3.3 Comparing and Combining GNNs and RNNs

Question 3.3

a) RNNs are useful in training sequential data and it is hard to train RNNs on arbitrarily structured graphs. RNNs require the input graphs to be directed and acyclic while GNNs can process arbitrarily structured graphs. GNNs can actually also be applied to sequential data, but I would think that the implementation of an RNN is more flexible for constructing an architecture to train for arbitrarily long timesteps. If you would use a GNN to train on

propagating through a sentence, word-for-word, one would need 1 GNN layer per (word) propagation, as opposed to 1 RNN layer to propagate through a whole sentence.

I would see the GNN outperform the RNN on image classification tasks, combinatorial optimization problems and protein-interaction networks, since a GNN can efficiently represent arbitrarily structured graphs in its adjacency matrix. An RNN would maybe outperform a GNN on tasks that require short and long term dependency over a long interval.

b) I would think that GNNs and RNNs could be used in a combined model for various NLP tasks, such as machine translation or text summarization. A GNN would be well suited to be trained and keep track on the dependencies within a text, such as the structure of a sentence or combination of sentences.