Assignment 2: Recurrent Neural Networks and Graph Neural Networks

Volodymyr Medentsiy
id: 12179078
volodymyr.medentsiy@student.uva.nl

1 Vanilla RNN versus LSTM

Question 1.1
In the following exercise I use results derived in the assignment 1. l denotes $argmax(y)$

$$\mathbf{1)}\frac{\partial L^{(T)}}{\partial W_{ph}} = \frac{\partial L^{(T)}}{\partial p^{(T)}}\frac{\partial p^{(T)}}{\partial W_{ph}} \quad \text{with} \quad (\frac{\partial L^{(T)}}{\partial W_{ph}})_{jk} = \sum_{i=1}^{K}(\frac{\partial L}{\partial p^{(T)}})_i(\frac{\partial p^{(T)}}{\partial W_{ph}})_{ijk} = \sum_{i=1}^{K}(-\delta_{il}+\tilde{y}_i^{(T)})\delta_{ij}h_k^{(T)} = (-\delta_{jl}+\tilde{y}_j^{(T)})h_k^{(T)}$$

$$\mathbf{2)}\frac{\partial L^{(T)}}{\partial W_{hh}} = \frac{\partial L^{(T)}}{\partial \tilde{y}^{(T)}}\frac{\partial \tilde{y}^{(T)}}{\partial h^{(T)}}\frac{\partial h^{(T)}}{\partial W_{hh}} = \frac{\partial L^{(T)}}{\partial \tilde{y}^{(T)}}\frac{\partial \tilde{y}^{(T)}}{\partial h^{(T)}}\sum_{\tau=0}^{T}\frac{\partial h^{(\tau)}}{\partial W_{hh}}\prod_{j=\tau+1}^{T}\frac{\partial h^{(j)}}{\partial h^{(j-1)}},$$

where

$$\frac{\partial L}{\partial \tilde{x}^{(T)}} \in R^{1*10}, \quad \text{with} \quad (\frac{\partial L}{\partial \tilde{x}^{(T)}})_i = -\frac{1}{\tilde{y}_l^{(T)}}\delta_{il}$$

$$\frac{\partial \tilde{y}^{(T)}}{\partial h^{(T)}} = \frac{\partial \tilde{y}^{(T)}}{\partial p^{(T)}}\frac{\partial p^{(T)}}{\partial h^{(T)}} = (diag(softmax(p^{(T)})) - softmax(p^{(T)})softmax(p^{(T)})^T)W_{ph}$$

$$\frac{\partial h^{(\tau)}}{\partial W_{hh}} = (1-(h^{(\tau)})^2)\frac{\partial (W_{hh}h^{(\tau-1)})}{\partial W_{hh}}$$

The issue with taking $\frac{\partial (W_{hh}h^{(\tau-1)})}{\partial W_{hh}}$ is that $h^{(\tau-1)})$ depends on the W_{hh} so the gradient should be decomposed in two terms: $(\frac{\partial W_{hh}}{\partial W_{hh}}h^{(\tau-1)} + W_{hh}\frac{\partial h^{(\tau-1)}}{\partial W_{hh}}$. And again $\frac{\partial h^{(\tau-1)}}{\partial W_{hh}} = (1-(h^{(\tau-1)})^2)\frac{\partial (W_{hh}h^{(\tau-2)})}{\partial W_{hh}}$, so we will need to decompose gradient up to the term $h^{(0)}$.

$$\frac{\partial h^{(j)}}{\partial h_{(j-1)}} = (1-(h^{(j)})^2)W_{hh}$$

I take both gradients using chain rule, however in the second case (**2**)) the term $\frac{\partial h^{(T)}}{\partial W_{hh}}$ should be decomposed up to the state $h^{(0)}$ because $h^{(T)}$ is defined recursively, and we should also consider all previous states $h^{(j<T)}$, which results in aggregating temporal information in the second gradient. With large T the effect of individual terms in the product $\prod_j \frac{\partial h^{(j)}}{\partial h_{(j-1)}}$ is accumulated and it could become either very large ($||\frac{\partial h^{(j)}}{\partial h_{(j-1)}}|| > 1 \Rightarrow ||\prod_j \frac{\partial h^{(j)}}{\partial h_{(j-1)}}|| >> 1$) or very small ($||\frac{\partial h^{(j)}}{\partial h_{(j-1)}}|| < 1 \Rightarrow ||\prod_j \frac{\partial h^{(j)}}{\partial h_{(j-1)}}|| << 1$), which causes the problem of exploding or vanishing gradients respectively.

Submitted to the Spring 2019 Amsterdam Deep Learning course

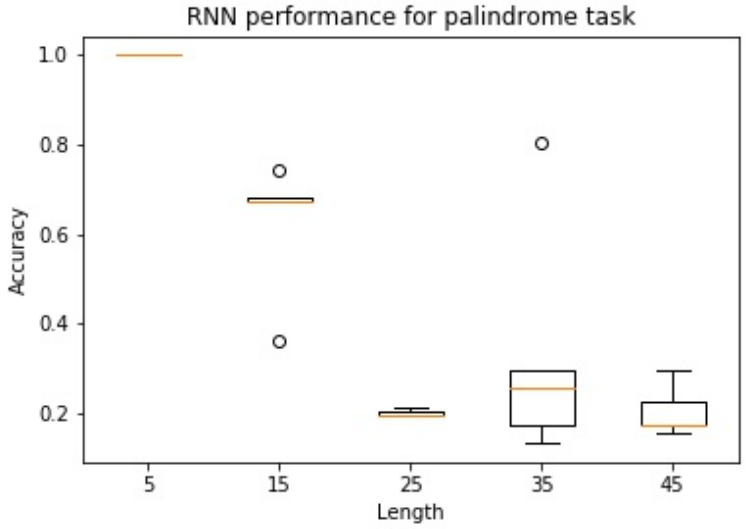


Figure 1: RNN performance for the palindrome task

Question 1.2
The implementation and training of vanilla RNN could be found in the part1/vanilla_rnn.py and part1/train.py file. During training we use torch.nn.utils.clip_grad_norm() function. It is used to bound the norm of the gradient within a certain threshold (specified by the max_norm argument), this technique can help with the problem of exploding gradients and makes training stable (avoid too large steps when update parameters). The weights are initialised using normal distribution with 0.0001 standard deviation.

Question 1.3
To compare the accuracy for different palindrome sizes, I train each model 5 times and then compute accuracy on the data of 10 batches size. In the Figure 1, bar plots for accuracy for various length palindromes are reported. The model was trained with default parameters. From these experiments I can conclude that RNN is capable of learning short term structure of palindrome of small length (for T = 5 we get perfect accuracy). However with an increase in the number of iterations performance deteriorates, which indicates RNN limitations in capturing long term dependencies.

Question 1.4
The loss surface in the parameters space could be a non-convex function with complex curvature, in such cases vanilla SGD could easily be stuck in local optima, oscillate near it or take too much time to converge. These problems could be addressed by adjusting the speed(learning rate) and direction(momentum) of updates. The technique of momentum helps to determine the direction of updates and speeds up convergence in the direction of local minima. It takes into an account the information of the previous gradients by introducing parameter 'velocity'. 'Velocity' is comprised of the new gradient g and the exponential moving average of previous gradients αv. The influence of past gradients in the 'velocity' v is controlled by the hyperparameter $\alpha \in [0, 1]$ (ϵ - is the learning rate):

$$\begin{cases} v_{k+1} = \alpha v_k - \epsilon g \\ \theta_{k+1} = \theta_k + v \end{cases}$$

The momentum algorithm was proven to be useful for functions with pathological curvature, where vanilla SGD oscillates, which slows down optimization.

Loss surface could rapidly change along one direction while staying almost the same in another; in such a case, it makes sense to use different learning rates along a different axis. The technique of adaptive learning rate leverages this idea by considering the learning rate to be a vector so that each learnable parameter has a corresponding component in it. Learning rates could be adjusted in various

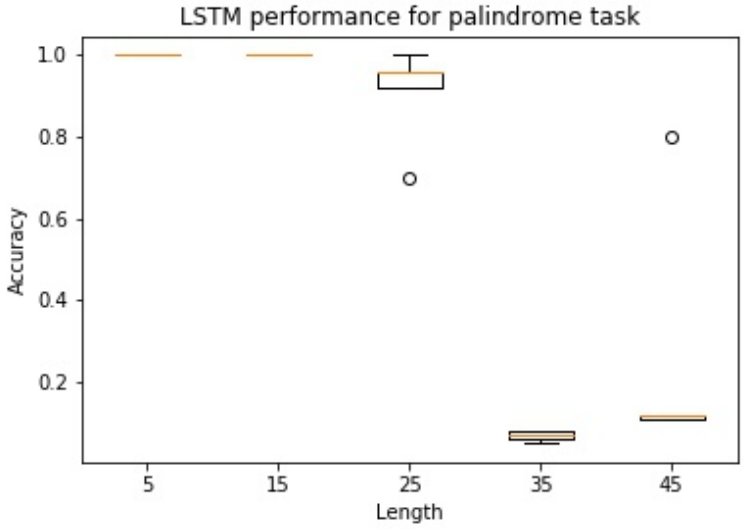


Figure 2: LSTM performance for the palindrome task

ways, for example, the RMSProp algorithm updates them with respect to the inverse of the square root of the exponential average of squared gradients.

The RMSProp and ADAM are modifications of vanilla SGD. While RMSProp uses only the technique of adaptive learning rate, ADAM uses both momentum and adaptive learning rate and is considered as one of the best optimizers.

Question 1.5
A Forget gate $f(t)$. Given input x_t and previous hidden state h_{t-1} defines components which will be discarded from the previous cell state c_{t-1} and which will be kept. It uses sigmoid activation, so that 0 corresponds to discard the component from c_{t-1} and 1 to keep it in c_{t-1}.

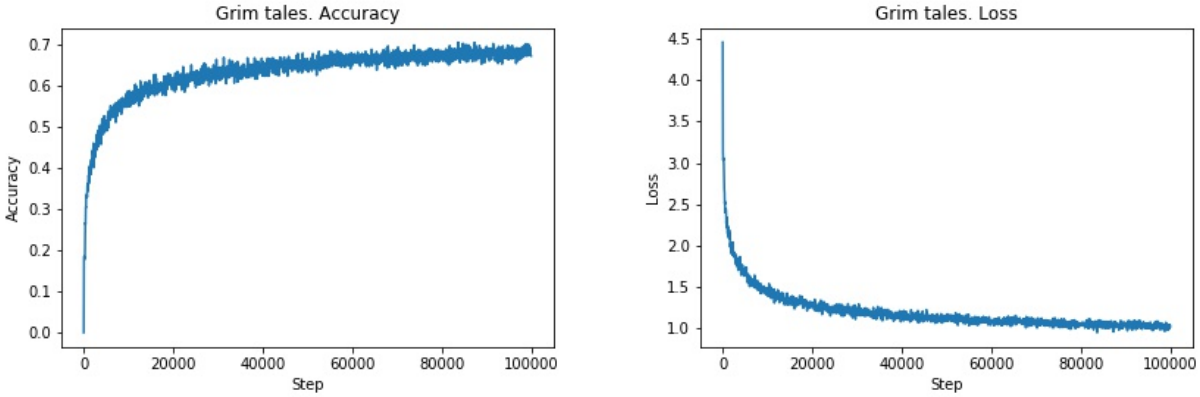
Input modulation gate $g(t)$. Given input x_t and previous hidden state h_{t-1} defines candidate values to update the cell state c_{t-1} with. It has tanh activation, so that only values within range [-1; 1] could be added to the components of the cell state.

Input gate $i(t)$. Given input x_t and previous hidden state h_{t-1} defines which components of the previous cell state c_{t-1} will be updated. It has sigmoid activation, with 1 corresponding to updating the corresponding component of c_{t-1}. The final vector with values which will be added to the c_{t-1} is defined as element wise multiplication $i(t) * g(t)$.

Output gate $o(t)$. Given input x_t and previous hidden state h_{t-1} defines components of the updated cell state c_t which should be kept in it for the next forward step.

B Each unit consists of four gates and each gate has d*T (weight matrices W_x) + d*d (weight matrices W_h) + d (bias term) parameters. So the total number of parameters is equal to: $n*4*(d*T+d*d+d)$.

Question 1.6
In Figure 2, bar plots for accuracy for various length palindromes are reported (the experiments were conducted in a similar fashion as in question 1.3). All parameters were set to default values except the learning rate which was reduced to 0.0001. The choice of learning rate has a huge impact on the LSTM's performance on medium and long palindromes, with default parameter LSTM performed almost like a random model on them (mean accuracy around 0.1). The weights were initialized with xavier normal. We see that LSTM outperforms RNN on medium length palindromes, but its performance significantly drops on the palindromes longer than 35 symbols. I hypothesize that using more complex architecture with multiple LSTM layers would improve performance on long palindromes.



(a) Accuracy (b) Loss

Figure 3: Accuracy and Loss curves of the LSTM

2 Recurrent Nets as Generative Model

Question 2.1
A To train the network some of the parameters were left with default values (2 LSTM layers with 128 hidden layer size and the batch size 64), and some changed (learning rate 0.001 with ADAM optimizer for 10^5 steps). We see that after 20000 steps the convergence become slower. The loss and accuracy curves for the final are represented in Figure 3. Minimum achieved loss is 0.94 and maximum achieved accuracy is 0.7.

B Because the loss stabilizes after 20000 steps to get a more detailed illustration of generated text and a better understanding of the performance of NN, the model was saved each 20000 steps and also on the 100, 500, 1500, 4000 steps (saved models are in the results_grim folder). The code used to generate the sequence is given in the gen.py file, I alos added generate() method to the model class. The results of the generated text for a different number of steps are represented in the table:

Steps	Accuracy	Loss	Generated text
0	0	4.5	O6aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
100	0.18	3.05	a
500	0.3	2.5	Z the the the the the the the t
1500	0.41	2.08	on the the the the the the the
4000	0.5	1.68	?’ ‘I will the will with the wi
20000	0.6	1.28	the world was so that the morn
40000	0.63	1.18	X The peasant said: ‘I am not t
60000	0.67	1.06	y the third day, and the soldie
80000	0.68	1.05	Just let he saw that the third
Final model	0.67	1.0	quite satisfied, and the wolf w

At 0 step network did not learn anything, at 100 step model produces the most common symbol - space, at 500 - the most common word and at 4000 the most common phrase. After 20000 steps generated sentences are quite real and at least have a correct grammatical structure and represent the style of a text.

C The increase in T corresponds to a more random choice of characters, which results in less plausible sentences. It happens because bigger T smoothes distribution and as T goes to ∞ we will perform random sampling.

T = 0.5: Just like the princess, the sol
T = 1: //p. Grisly.’ So he heard this,
T = 2: TROS Recas!’ hown!’ awl bair ha

The effect of adding the temperature for longer sequences is illustrated in question 2.2

Question 2.2
Below are some examples of sentences with a length of 180 symbols:
S1(greedy): 'Kywitt, what a beautiful princess, and the soldier was a good dinner, and the soldier was all the world were so streamon, however, were set out, and the soldier was a good dinner, a'
S2(greedy): 'My went down to the wood, and the soldier was all the world were so streamon, however, were set out, and the soldier was a good dinner, and the soldier was all the world were so str'
S3(greedy): 't the work was a good dinner, and the soldier was all the world were so streamon, however, were set out, and the soldier was a good dinner, and the soldier was all the world were so'
S4(temperature = 0.5): 't work. His way they were all of water, and fell asleep; and the stones were as the tree, and the third day the wolf knows a beauty, ate a man who had something back to the town to '
S5(temperature = 0.5): ' and the soldier, who knocks at the dish, and said to him, 'I will give you a good man for a man and drank the garden after the window, and said: 'If you will do nothing like this w'
Last two sentences are more versatile in comparison with the first three, where certain phrases and words prevail. Generated sentences have correct grammatical structure and generated punctuation also looks plausible. Some phrases, if we took them out of the context of the whole sentence, could make sense, but the whole sentences are meaningless. This could be explained by the fact that the neural network was trained on the text of 30 chars at most and thus can not capture longer dependencies. So the generating capacity is limited by the way the neural network was trained. To create a sequence with given beginning I have implemented functions generate_greedy_given() and generate_temperature_given(). Again greedy version is too confident and generates specific phrases and words more common than others. I found the temperature version more interesting because its output was more randomized and unexpected. However, both sentences are nonsense from the perspective of lexical meaning:

S1 (greedy): 'Sleeping beauty is too much to the bottom of the work with the top of the tree, and the soldier was a good dinner, and the soldier was all the world were so streamon, however, were set out, and the so'
S2 (temperature = 0.5): 'Sleeping beauty is there, and the man said to himself, 'I will give you a good man met the forest, and said to him, 'Where are you going,' said the country and said: 'I will give you a defect Gutenber'.

3 Graph Neural Networks

Question 3.1
A The structural information is incorporated in the adjacency matrix H, and we exploit this information when we multiply it by A and W, which corresponds to taking weighted average of the feature vectors of neighboring nodes. Thus new feature representation of the node is constructed as a weighted average of features of the node itself and its neighbors. When we apply a GCN layer on the graph, we update each feature vector with the aforementioned weighted sum, which could be seen as passing messages (feature vectors) from neighboring nodes to get the updated feature of the node.

B 3 layer GCN will convolve each node with all other nodes which are three hops away from it.

Question 3.2
As described in Zonghan Wu [2018] GCN could be effective in the field of computer vision (point clouds classification and segmentation, action recognition), chemistry (learning molecular fingerprints and predicting molecular properties), recommender systems, traffic (taxi demand prediction and forecasting traffic speed).

Question 3.3
A RNN applies the same weights across all consecutive data points, unlike GNN which applies the weight matrix over all connected nodes at once. So RNN learns temporal dependencies and GNN structural dependencies. Also, an important distinction is that RNN could be applied on sequences of variable length, unlike GNN where the number of nodes in the graph should be fixed. So I hypothesize that RNN is more useful when we aim to learn dynamical patterns in data or work with data which updates real-time, while GNN is useful when we need to model structural or spatial dependencies in the data. Thus I think that GNN could outperform RNN when we work with social or transportation

networks and images, but RNN could outperform GNN when working with time series and natural language data.

B Combining both GNN and RNN makes sense when we want to model spatial and temporal information in the data. For example, the paper Youngjoo Seo [2017] suggests to first apply graph convolutional network on the data $x_t^{GCNN} = GCNN(x)$ and then run LSTM on top of it $x_t^{LSTM} = LSTM(x_t^{GCNN})$. This approach was proposed for the task of predicting moving MNIST and natural language modeling.

References

Member IEEE Fengwen Chen Guodong Long Chengqi Zhang Senior Member IEEE Philip S. Yu Fellow IEEE Zonghan Wu, Shirui Pan. A comprehensive survey on graph neural networks. 2018. URL `https://arxiv.org/pdf/1901.00596.pdf`.

Pierre Vandergheynst Xavier Bresson Youngjoo Seo, Michael Defferrard. Structured sequence modeling with graph convolutional recurrent networks. 2017. URL `https://arxiv.org/pdf/1612.07659.pdf`.