Deep Learning - Assignment 2

Andrei Pauliuc
11596619
andupauliuc@gmail.com

1 Vanilla RNN versus LSTM

Question 1.1

$$\mathcal{L}^{(T)} = -\sum_{k=1}^{K} \boldsymbol{y}_k^{(T)} \log \hat{\boldsymbol{y}}_k^{(T)}$$

$$\Rightarrow \frac{\partial \mathcal{L}^{(T)}}{\partial \boldsymbol{W}_{ph}} = \frac{\partial \mathcal{L}^{(T)}}{\partial \hat{\boldsymbol{y}}^{(T)}} \cdot \frac{\partial \hat{\boldsymbol{y}}^{(T)}}{\partial \boldsymbol{p}^{(T)}} \cdot \frac{\partial \boldsymbol{p}^{(T)}}{\partial \boldsymbol{W}_{ph}}$$

In this derivation, matrix $\boldsymbol{W}_{ph}$ influences $\mathcal{L}^{(T)}$ directly and only once through $\hat{\boldsymbol{y}}^{(T)}$, so the derivation is straight-forward and contains multiplication of vectors and matrices.

$$\frac{\partial \mathcal{L}^{(T)}}{\partial \boldsymbol{W}_{hh}} = \frac{\partial \mathcal{L}^{(T)}}{\partial \hat{\boldsymbol{y}}^{(T)}} \cdot \frac{\partial \hat{\boldsymbol{y}}^{(T)}}{\partial \boldsymbol{p}^{(T)}} \cdot \frac{\partial \boldsymbol{p}^{(T)}}{\partial \boldsymbol{h}^{(T)}} \cdot \frac{\partial \boldsymbol{h}^{(T)}}{\partial \boldsymbol{W}_{hh}}$$

We can observe that $\boldsymbol{h}^{(T)}$ depends on $\boldsymbol{W}_{hh}$, but also on $\boldsymbol{h}^{(T-1)}$, which also depends on $\boldsymbol{W}_{hh}$ and $\boldsymbol{h}^{(T-2)}$, and so on. Thus, $\boldsymbol{W}_{hh}$ influences $\mathcal{L}^{(T)}$ through the hidden state at all timesteps, and the derivative becomes:

$$\begin{aligned}\frac{\partial \mathcal{L}^{(T)}}{\partial \boldsymbol{W}_{hh}} &= \sum_{i=1}^{T} \frac{\partial \mathcal{L}^{(T)}}{\partial \hat{\boldsymbol{y}}^{(T)}} \cdot \frac{\partial \hat{\boldsymbol{y}}^{(T)}}{\partial \boldsymbol{p}^{(T)}} \cdot \frac{\partial \boldsymbol{p}^{(T)}}{\partial \boldsymbol{h}^{(T)}} \cdot \frac{\partial \boldsymbol{h}^{(T)}}{\partial \boldsymbol{h}^{(i)}} \cdot \frac{\partial \boldsymbol{h}^{(i)}}{\partial \boldsymbol{W}_{hh}} \\ &= \sum_{i=1}^{T} \frac{\partial \mathcal{L}^{(T)}}{\partial \boldsymbol{h}^{(T)}} \cdot \frac{\partial \boldsymbol{h}^{(T)}}{\partial \boldsymbol{h}^{(i)}} \cdot \frac{\partial \boldsymbol{h}^{(i)}}{\partial \boldsymbol{W}_{hh}}\end{aligned}$$

It can be noticed that the derivative of $\boldsymbol{h}^{(T)}$ with respect to $\boldsymbol{h}^{(i)}$ can be rewritten using the chain rule as

$$\begin{aligned}\frac{\partial \boldsymbol{h}^{(T)}}{\partial \boldsymbol{h}^{(i)}} &= \frac{\partial \boldsymbol{h}^{(T)}}{\partial \boldsymbol{h}^{(T-1)}} \cdot \frac{\partial \boldsymbol{h}^{(T-1)}}{\partial \boldsymbol{h}^{(T-2)}} \cdots \frac{\partial \boldsymbol{h}^{(i+1)}}{\partial \boldsymbol{h}^{(i)}} \\ &= \prod_{k=i}^{T-1} \frac{\partial \boldsymbol{h}^{(k+1)}}{\partial \boldsymbol{h}^{(k)}}\end{aligned}$$

$$\Rightarrow \frac{\partial \mathcal{L}^{(T)}}{\partial \boldsymbol{W}_{hh}} = \sum_{i=1}^{T} \frac{\partial \mathcal{L}^{(T)}}{\partial \boldsymbol{h}^{(T)}} \cdot \left(\prod_{k=i}^{T-1} \frac{\partial \boldsymbol{h}^{(k+1)}}{\partial \boldsymbol{h}^{(k)}} \right) \cdot \frac{\partial \boldsymbol{h}^{(i)}}{\partial \boldsymbol{W}_{hh}}$$

The big difference between the two gradients involving $\boldsymbol{W}_{ph}$ and $\boldsymbol{W}_{hh}$ is the summation and product over all hidden states in the latter one. The second gradient depends on all timesteps, since each hidden state is computed using also the previous states.

If the number of steps becomes large, the computation of the gradient of loss with respect to $\boldsymbol{W}_{hh}$ involves the product of many factors which increases the risk of having a gradient with value 0. This in turn will not update the weight matrix and thus training is stuck at a point in the manifold.

Question 1.2 and 1.3

Using the default parameters given for this part (128 hidden units, 0.1 learning rate, 10000 training steps) the vanilla RNN resulted in the following accuracy performance on different sequences.

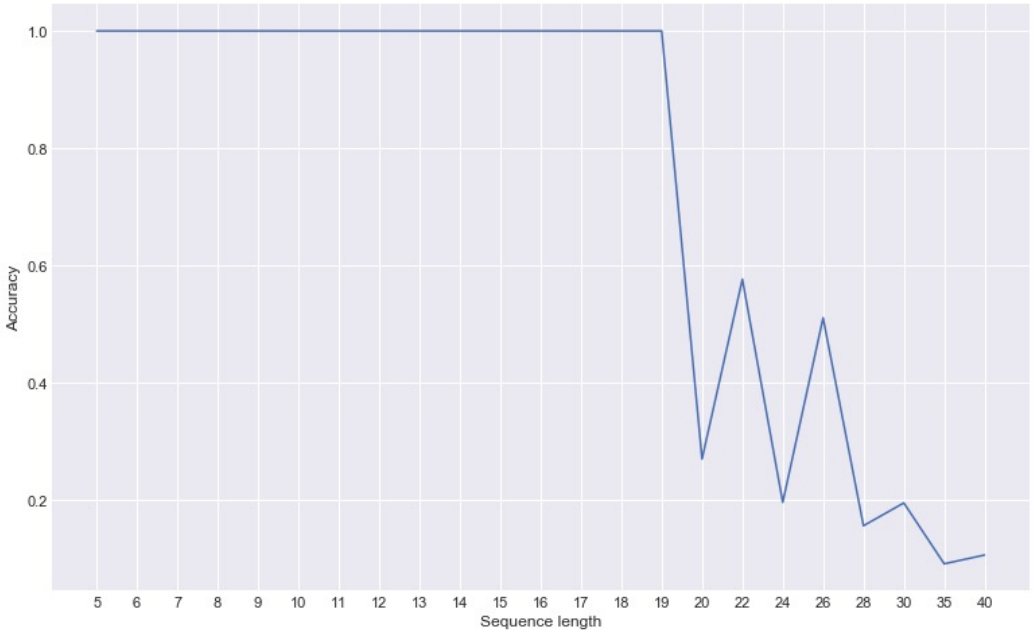

Figure 1: Vanilla RNN accuracy

As figure 1 shows, the RNN performs really well on sequences of length below 19. After that, its performance drops and cannot pass 20% after sequences of length 30.

Question 1.4

One of the problems posed by the traditional stochastic gradient descent method is the convergence of the training to local minima and/or saddle points. Sometimes, the algorithm cannot detect that the found minima is local and move out of it. Even more, when training a model with mini-batches, the methods actually estimate the derivatives of the loss function with respect to weights, which results in noisy gradients. More optimal methods for updating the model's weights take into account past gradients and make use of learning rates for each individual parameter. Moreover, the use of first and second order moments create a more powerful learning dynamic, speeding up and improving the descent of the gradient towards the global minimum.

For example, RMSprop makes use the history of squared gradients, exponentially decaying them at every time step. These gradients are then involved in adjusting the learning rates of each weight of the network. This method is widely known as adaptive learning rate and is effective even when the gradients are very large or very small.

Besides using adaptive learning rates, Adam also uses the history of gradients, which estimates the mean. Because of this, gradients pointing in the same direction are further pushed in that direction, while the gradients which point in opposite directions (which happens in a ravine, for example) can be cancelled out. This way, the momentum of the gradients is taken into account.

Question 1.5

a)

- *Input modulation gate*, $\boldsymbol{g}^{(t)}$, deals with creating actual values that can be stored in the cell state at time (t). It is the only gate with $tanh$ activation, which creates values between -1

and 1. The reason behind this non-linearity is to overcome the vanishing gradient problem. $Tanh$ is good for this job because its second derivative has the property of sustaining for a long range before it goes to 0.

- *Input gate*, $\boldsymbol{i}^{(t)}$, decides which values coming from the input modulation gate will be actually used in the current cell state. The $sigmoid$ activation creates values between 0 and 1 for every corresponding value, and indicates how much from the new information will be stored. The $sigmoid$ activation is suitable here because its output can oscillate between "no information is passed" ($= 0$) and "all information is passed" ($= 1$)
- The *forget gate*, $\boldsymbol{f}^{(t)}$, is in charge of throwing away information which already exists in the cell state. Based on the values of the input and the hidden state, together with the $sigmoid$ activation function, the gate plays an important role in decoupling old and irrelevant information from what comes next. Similar to the input gate, it is a good choice to use the $sigmoid$ non-linearity here to create values in the range $[0, 1]$.
- Finally, the *output gate*, $\boldsymbol{o}^{(t)}$, plays the role of deciding what information stored in the cell state is relevant as output for the current timestep t. The $sigmoid$ function is also used here for the same role, to determine which cell state values are actually used as output and for the next hidden state, and is the correct choice for this role.

b) The LSTM cell has as trainable parameters 8 weight matrices and 4 bias vectors. The matrices that are multiplied with the input have dimension $(hidden\_size \times input)$, and the ones connecting to the hidden states have dimension $(hidden\_size \times hidden\_size)$. The bias vectors have dimension $(hidden\_size \times 1)$. Thus, the total number of trainable parameters in the LSTM cell is:

$$\begin{aligned} &= 4 \cdot n \cdot d + 4 \cdot n \cdot n + 4 \cdot n \\ &= 4n(d + n + 1) \end{aligned}$$

Question 1.6

Training the LSTM implementation on the same set of sequences lengths and with the same parameters, the accuracy obtained is shown in figure 2. Only the learning rate is declined for the experiments made on sequence length above 20.

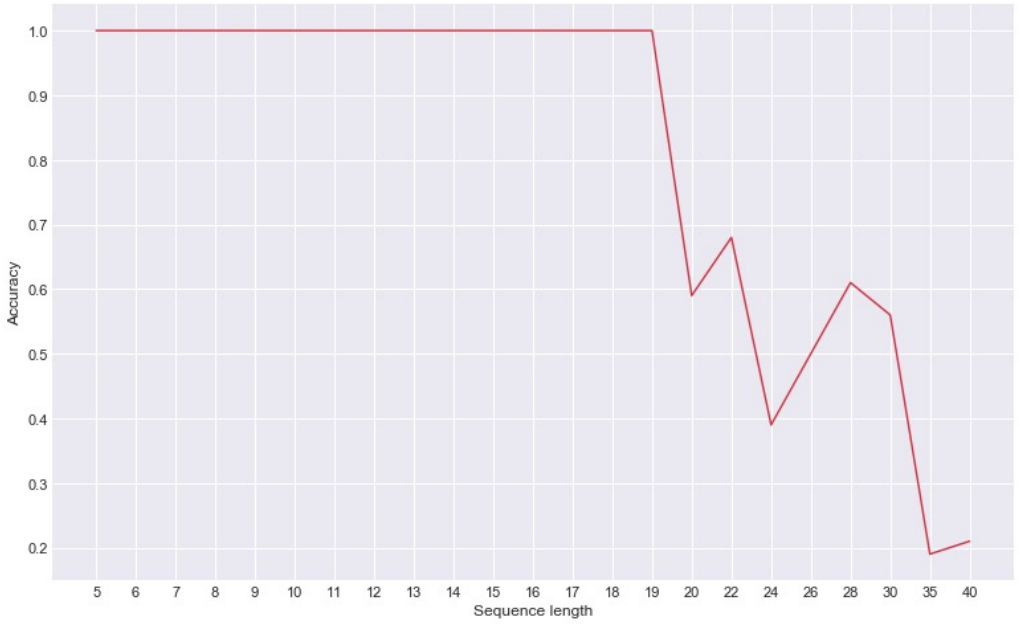

Figure 2: LSTM accuracy

Like the vanilla RNN, the LSTM model reaches accuracy 100% for the lengths from 5 to 19, after which the performance drops. However, the drop in performance is not as bad as the vanilla RNN

displays, with an accuracy which stays above 20% for most cases. A comparison of both models can be directly observed in figure 3. As expected, the LSTM model is better at modelling longer sequences than the vanilla RNN version on all tested scenarios. The main reason behind this difference is the architecture that LSTM has, including a cell/memory state which can model long term dependencies.

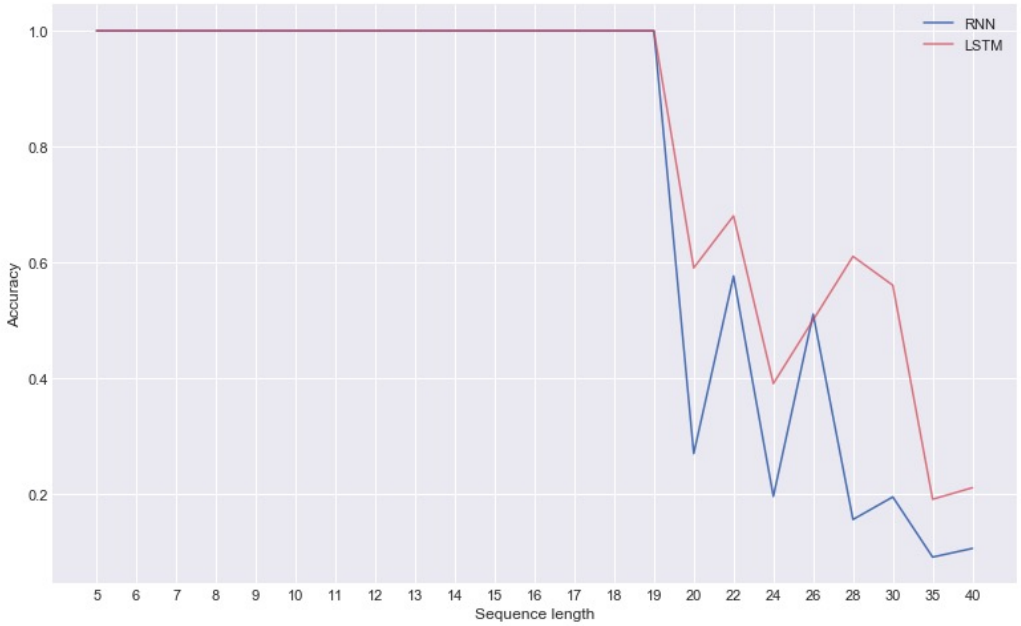

Figure 3: RNN vs. LSTM accuracies

2 Modified LSTM Cell

Question 2.1

The change of the temporal gate $k(t)$ is displayed in figure 4. The value of $k(t)$ oscillates in the interval $[0, 1]$, based on the values of the three parameters.

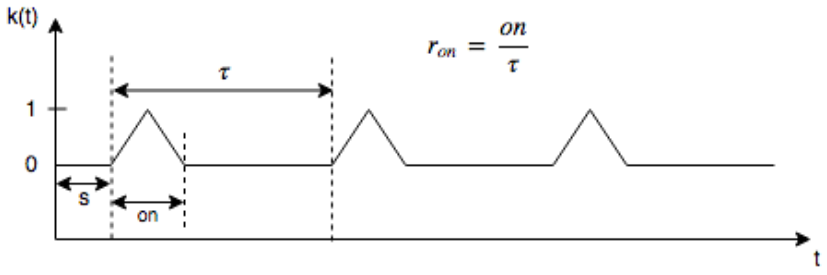

Figure 4: Change in $k(t)$ with respect to time t

Question 2.2

The introduction of the cell $k^{(t)}$ leads to updates to the cell and hidden state at intervals of time dictated by parameters of the cell. Because of this, the cell and hidden states are not updated at each timestep, but only when the gate is opened.

The new LSTM cell provides advantages when working with event-driven input data which might be asynchronously sampled. This is really usefull when the input comes from different real sensors, for example, each with its unique sampling rate.

Question 2.3

The rhythmic oscillation is controlled by the 3 parameters:

- Parameter τ is the total time of the oscillation and equals open_phase + closed_phase.
- Parameter r_{on} equals the ratio of the open_phase over the whole period.
- Parameter s is a phase shift, indicating a delay between $t = 0$ and the start of the first oscillation.

All three parameters can be learned from the data and do not need to be specified as hyperparameters. This is even more advisable if the data presents temporal dynamics.

3 Recurrent Nets as Generative Model

Question 3.1

a) The generative model I implemented was trained once on the book *Grimms' Fairy Tales* by Jacob Grimm, and once on the Dutch version of *Anna Karenina* by Leo Tolstoy. In both cases, the LSTM network has 2 layers with 128 hidden units, including a dropout in the LSTM cell with value of 0.3. Starting the training with a learning rate of $2e-3$, decaying it every $10,000$ steps with a rate of 0.95 and training the models for about $500,000$ training steps both models reach a loss value around 1.020 and accuracy over 60%. However, the model trained on *Grimms' Fairy Tales* has better accuracy, with the highest accuracy being 69.58%, while the other model reaches only 64.97%.

b) Using the model trained on *Grimms' Fairy Tales* book and sampling from the model by randomly selecting the first character in the sequence, we get the following sequences:

Step	Sample
$5,000$	% to the world, and the stars w
$105,000$	Jorindel said: 'There is a gold
$205,000$	r the world, and the second son
$305,000$	But the second son said: 'I wil
$405,000$	y and said: 'I have been drinki

As we can see, the model learns a structure from the training data, and it becomes better and better as the model is trained. It's interesting to see that when comparing the first and the third sample, after the word 'world', a comma is introduced, followed by a space and the word 'the'. This shows the model actually learnt to correctly separate the words and create continuations with 'and'. Also, we can notice a change in the pattern it gives after the sequence 'said:', since in the last sample, this is followed by something completely different than in the second sample. All these illustrate that the LSTM is capable of learning structure, and based on the history of inputs, it can create meaningful sequences.

c) Implementing temperature to control between fully-greedy and fully-random sampling, we observe interesting changes to the learning dynamic of the network. Here are some sequences of length 30 generated for different temperatures.

Temperature = 0.5

- "I will take her and said, 'I w
- 01. The will be seen my father
- joy at the same time she was so
- " and the second son said, 'I w

Temperature = 1

- On the stars and said: 'I will
- UNDER OF THIS PROJECT GUTENBERG
- !' said the fisherman, 'that is

- was to be seen and said: ‘I hav

Temperature = 2

- the second stones and the seco
- For a little peasant said: ‘I w
- the strength to see the stren
- the wolf was so beautiful that

Bonus questions

To test the capabilities of the network, I have tried multiple scenarios where the model predicts sequences longer than 30 characters. This is based on sequences selected by hand, and even by letting the network create a sequence of arbitrary length and the using that as input to the network.

Because the model was trained on the book *Grimms' Fairy Tales*, I first tried to predict a continuation to the sequence "The sleeping beauty is". This produced the following sentences:

- Length 52: The sleeping beauty is to be mayor.’ The man went ou
- Length 142: The sleeping beauty is to be mayor.’ The man went out together, and the second son said: ‘I have not the back again! but the table was a great

Also interesting is to randomly sample a sequence of 30 characters, and then let the model continue it with another 30/60 characters. This produced results like these:

- You must go out of my brothers’ lair the work of the world, and the princess was already sat down on the stairs, and said: ‘I have
- Little Red-Cap, however, who was already sat down and said: ‘I have not
- was the first place when he had gone out of the water, and said: ‘I have not the back again! but the table was a great de
- he sparrow asked him what he was so fast as they were sittin
- Me Snow-white and Rose-red went to bed, and said, ‘I will giv

As a conclusion, the model successfully learns some patterns between words, but only in a window of 30 characters. Moreover, if the sample starts with a specific ordering of the characters, than the continuation will most likely be the same for any generated sequence (as is the continuation of the sequence "The sleeping beauty is").