Deep learning Assignment 2

Raymond Koopmanschap - 11925582
`raymond.koopmanschap@student.uva.nl`

1 Vanilla RNN versus LSTM

Vanilla RNN in PyTorch

Question 1.1

First the softmax result from the first assignment is written out in matrix form

$$\frac{\partial \hat{\mathbf{y}}_i}{\partial \mathbf{p}_j} = \hat{\mathbf{y}}_i(\mathbb{1}_{i=j} - \hat{\mathbf{y}}_j) = \hat{\mathbf{y}}_i \mathbb{1}_{i=j} - \hat{\mathbf{y}}_i \hat{\mathbf{y}}_j)$$

$$\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{p}} = diag(\hat{\mathbf{y}}) - \hat{\mathbf{y}}\hat{\mathbf{y}}^T$$

Where diag means that the vector $\hat{\mathbf{y}}$ is on the diagonal of the matrix.
Second we will derive the loss with respect to each index i,j of the weight matrix.

$$\frac{\partial \mathcal{L}^{(T)}}{\partial \mathbf{W}_{ph}^{ij}} = \frac{\partial \mathcal{L}^{(T)}}{\partial \hat{\mathbf{y}}^{(T)}} \frac{\partial \hat{\mathbf{y}}^{(T)}}{\partial \mathbf{p}^{(T)}} \frac{\partial \mathbf{p}^{(T)}}{\partial \mathbf{W}_{ph}^{ij}} = -\frac{\mathbf{y}^{(T)}}{\hat{\mathbf{y}}^{(T)}}(diag(\hat{\mathbf{y}}^{(T)}) - \hat{\mathbf{y}}^{(T)}(\hat{\mathbf{y}}^{(T)})^T) \begin{bmatrix} 0 \\ \vdots \\ h_j \\ \vdots \\ 0 \end{bmatrix} \tag{1}$$

$$= (\hat{\mathbf{y}}^{(T)} - \mathbf{y}^{(T)}) \begin{bmatrix} 0 \\ \vdots \\ h_j \\ \vdots \\ 0 \end{bmatrix} \tag{2}$$

$$= (\hat{y}_i^{(T)} - y_i^{(T)}) h_j^{(T)} \tag{3}$$

$$\frac{\partial \mathcal{L}^{(T)}}{\partial \mathbf{W}_{ph}} = (\hat{\mathbf{y}}^{(T)} - \mathbf{y}^{(T)})(\mathbf{h}^{(T)})^T \tag{4}$$

The divide $\div$ is elementwise division. In the second step you get $(\hat{\mathbf{y}} - \mathbf{y})$ because y only has 1 entry that is non-zero. The h_j is in the i-th row, therefore you get y_i as index.

The second derivative to calculate is $\frac{\partial \mathcal{L}^{(T)}}{\partial \mathbf{W}_{hh}}$

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

Now the first two terms are already calculated in the previous equation and $\frac{\partial \mathbf{p}^{(T)}}{\partial \mathbf{h}^{(T)}} = \mathbf{W}_{ph}$. Now since $\frac{\partial \mathbf{h}^{(T)}}{\partial \mathbf{W}_{hh}}$ depends on $\mathbf{h}^{(T-1)}$ and $\mathbf{h}$ also depends on $\mathbf{h}^{(T-1)}$ and this is recursive we get:

$$\frac{\partial \mathcal{L}^{(T)}}{\partial \mathbf{W}_{hh}} = \sum_{t=0}^{T} \frac{\partial \mathcal{L}^{(T)}}{\partial \hat{\mathbf{y}}^{(T)}} \frac{\partial \hat{\mathbf{y}}^{(T)}}{\partial \mathbf{p}^{(T)}} \frac{\partial \mathbf{p}^{(T)}}{\partial \mathbf{h}^{(T)}} \frac{\partial \mathbf{h}^{(T)}}{\partial \mathbf{h}^{(t)}} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{W}_{hh}}$$

Now note that the gradient $\frac{\partial \mathbf{h}^{(T)}}{\partial \mathbf{h}^{(t)}}$ will be computed by the chain rule:

$$\frac{\partial \mathbf{h}^{(T)}}{\partial \mathbf{h}^{(t)}} = \prod_{t}^{T} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}} \tag{5}$$

Now if the amount of timesteps gets large, the product defined in 5 will be a long multiplication of Jacobians. Now if you are using the tanh or sigmoid activation function the norm of the Jacobians all have an upper-bound of 1 (1). Because there will each time be multiplied by a number less than 1 the gradients will vanish really quickly, which causes the model to not update the gradients anymore, which is a huge problem for training.

Question 1.2

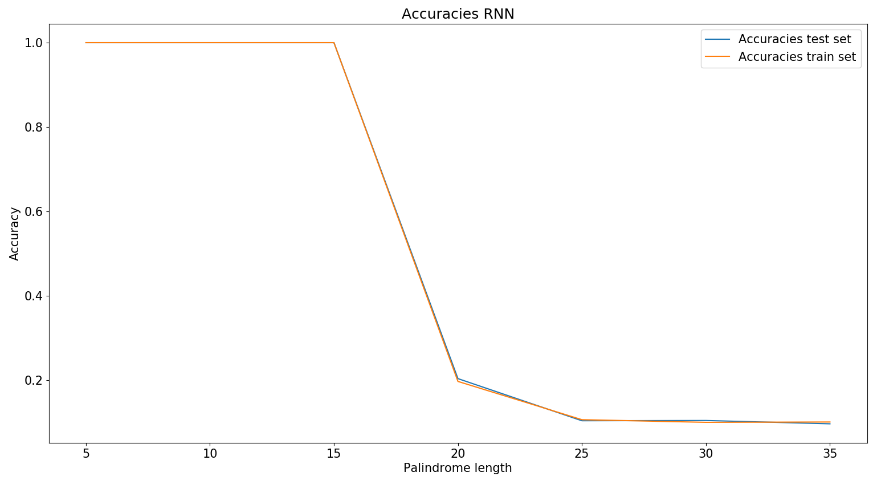
The RNN is implemented with orthogonal initalization, furthermore an adam optimizer is used with a learning rate of 0.001 and the cross entropy loss is used for evaluation. For further details see the code. Also the answer to the question: what happens here and why?

```
(torch.nn.utils.clip_grad_norm(model.parameters(),␣max_norm=config.max_norm))
```

Answer: It makes sure the gradients are not too high (thus preventing exploding gradients)

Question 1.3

To evaluate the accuracy of the model, there is trained on the training set. After that there is evaluated on both a test set of 10.000 samples, where the average accuracy is taken and the average accuracy of the last 10 training batches is taken. This is done for a sequence length of 5, 10, 15, 20, 25, 30 and 35.



A few things are noticable:

- The train and test set have roughly the same accuracy
- The accuracy up to a length of 15 is 1, it predicts everything correctly
- After a length of 15, there is a very sharp drop in accuracy, until it reaches 0.1, which is the same as random guessing.

Question 1.4

Lets first start with momentum. The formula for momentum is given by

$$u_{t+1} = \gamma u_t - \eta_t g_t$$
$$w_{t+1} = w_t + u_{t+1}$$

So instead of updating the parameters by the gradient there is updated with the gradient plus the previous gradient, which again depends on the previous gradient etc. This can be seen as an exponential moving average, where the most recent gradient have the biggest influence on the next gradient and this influence is decaying. There are two noticeable effect that this change has with respect to SGD.

- If the gradient stays consistent, e.g. always the same number then each time u will decrease (since we are doing gradient **descent**) and thus the gradient will become more and more negative.
- If the gradient constantly changes, e.g. [-1, 1, -1, 1 ...] then u each time subtract and add the same value, staying around 0 and thus the gradient will average out, smoothening the effect of rapidly changing gradients.

The second gradient optimization technique is RMSProp. The formula is given by

$$\begin{gathered} r_t = \alpha r_{t-1} + (1-\alpha) \odot g_t^2 \\ u_t = -\frac{\eta}{\sqrt{r_t + \varepsilon}} \odot g_t \\ w_{t+1} = w_t + \eta_t u_t \end{gathered}$$

Here you can see that the gradient get scaled by a factor $\frac{1}{\sqrt{r_t+\epsilon}}$. This factor can be seen as a normalization, if the gradients are small, they will become bigger, if they where big they become smaller. Basically it is trying to roughly give all gradients the same magnitude. This scaling factor also depends on the previous gradient, just as by momentum, only this time squared.

The last technique which combines momentum with RMSProp is called Adam and is given by

$$\begin{gathered} m_t = \beta_1 m_{t-1} + (1-\beta_1)\, g_t \\ v_t = \beta_2 v_{t-1} + (1-\beta_2)\, g_t^2 \\ \widehat{m}_t = \frac{m_t}{1-\beta_1^t}, \hat{v}_t = \frac{v_t}{1-\beta_2^t} \\ u_t = -\eta_t \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon} \\ w_{t+1} = w_t + u_t \end{gathered}$$

Here m_t can be seen as the momentum term, the gradient gets scaled by the exponential moving average of the gradients. v_t can be seen as the RMSProp term, where the gradient gets normalized by a factor $\frac{1}{\sqrt{v_t}+\epsilon}$. The $\hat{m}_t$ and $\hat{v}_t$ are calculated, those correct factors are especially important in the beginning of the training where the gradients are biased towards our initial value of m_0 and v_0, to correct for this the bias correction is implemented. Some important things to notice are:

- A sign difference doesn't matter for v_t, since you take the square. So a gradient of [1, -1, 1, -1 ..], will still result in an increasing v_t and will thus increase the gradient, however since momentum will decrease the gradient, it will still be low.
- If β_1 and β_2 are the same (e.g. both 0.9) and the gradient is 1, v_t amd m_t will be the same but the $\frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}$ still doesn't cancel each other out. Since the square root is only taken at the update step. Therefore if the gradient increases consistently (higher m_t and v_t but still at the same rate), then momentum gets the upper hand and will have a bigger influence on the resulting gradient.

Long-Short Term Network (LSTM) in PyTorch

Question 1.5 a

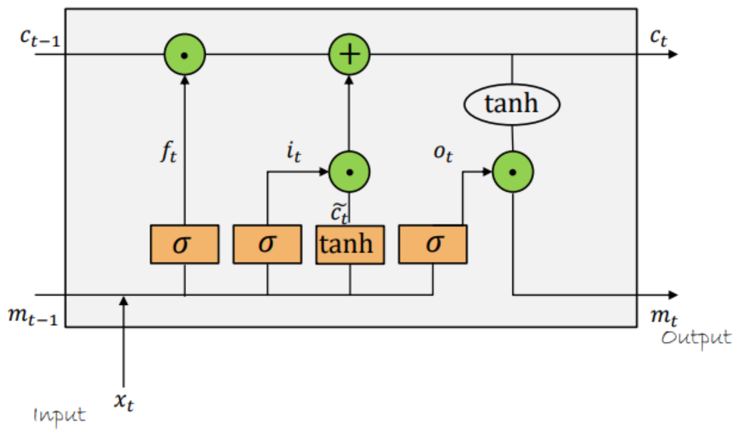


Figure 1: Graphical picture of LSTM

The purpose of the LSTM network is to provide a solution to the vanishing gradient problem that occurs in RNNs. It does this by letting the cell state c_t be connected to c_{t-1} by addition instead of multiplication. Thus not all gradients will be multiplied by a low number, if one gradient is low. Furthermore it has 4 gates to determine which things to put in the cell state. This can also be viewed as determining which things you want to remember and which things do you want to forget. Those four gates are the forget gate f_t, the input gate i_t, the input modulation gate g_t (in the picture $\tilde{c}_t$) and the output gate o_t.
All those gates except the input modulation gate have a sigmoid activation function as non-linearity. Since the sigmoid has a range between 0 and 1, it can be seen as a gate that determines what is important. 0 means not important and 1 means important. The intuition here is that multiplying something by 0 results in 0 and adding 0 doesn't change anything. Another thing that the sigmoid does is not letting the weights/gradients get out of control, since they will at most be multiplied by 1. The input modulation has an activation of tanh, because this gate can be seen as proposing new candidates and thus you also want to store candidates that have a negative value, since you might want to explicitly remember that those words where not useful.
First the cell state is updated by the forget gate f_t, which determines which things to forget. This is then multiplied by the cell state. Next the input and input modulation gate are combined to propose new candidates to the cell state, this new candidates are than added. Finally the output gate o_t decides together with the cell state what the final prediction will be.

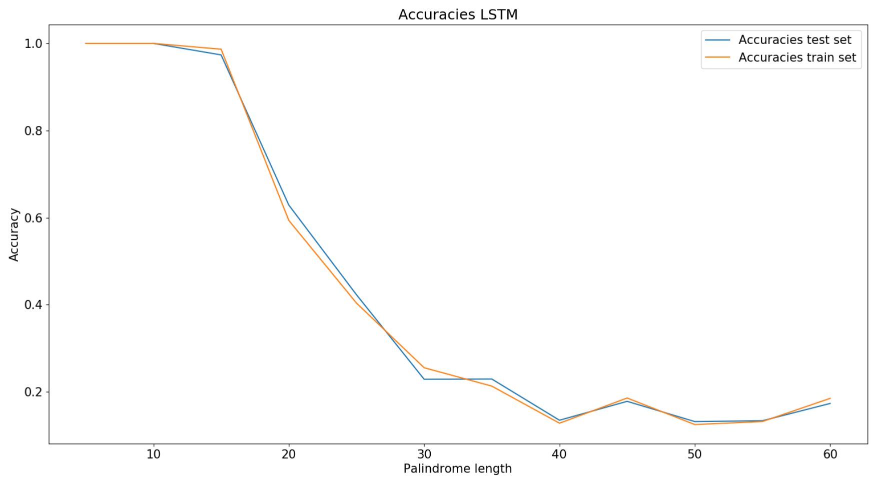
Question 1.5 b

Each of the four gates $g^{(t)}, i^{(t)}, f^{(t)}, o^{(t)}$ have one weight matrix of dimensions n*d, where d is the number of features and n the hidden units. The second weight matrix has dimensions n*n and then a bias term of n. This makes a total of n*d + n*n + n parameters for each gate. Finally the weight matrix $\mathbf{W}_{ph}$ has dimensions c*n, where c is the number of classes (which is not defined in the exercise, but I am pretty certain that this is the case) and n is the number of hidden units. So the total amount of trainable parameters (pt) is:

$$pt = 4 * (n * d + n * n + n) + c * n$$

Question 1.6

The LSTM uses a random initialization from a normal distribution, also an adam optimizer is used this time with a learning rate of 0.01 and the cross entropy loss is used for evaluation. For further details see the code.



There is evaluated on both a test set with roughly 10.000 samples (128*78), and the average train accuracy of the last 10 steps are taken. This is done for a sequence length of 5 until 60, with increments of 5.

The LSTM performs slightly better than the RNN, in the RNN we saw a sharp dropoff in accuracy after a length of 20, in the LSTM a length of 20 still performs relatively well, 0.6 accuracy, after a length of 35 the LSTM performs the same as random guessing.

2 Recurrent Nets as Generative Model

I have chosen for the 2000 page counted book: Rationality from AI to Zombies, which is a book that tries to summaries a couple years of blog post from the website Lesswrong, it contains among other things also Bayesian formulas and has more characters than a normal text book. For example it includes the signs: $\infty, \rightarrow, \leq$ and things like $P(H|\neg E) < P(H)$.

Question 2.1 a

First the batch input and targets are converted in matrix form (as a tensor) before it is feed into the LSTM. Furthermore this model is also trained using the Adam optimizer and the cross-entropy loss. Once in a while a text is generated with sequence length of 30. Also a file called generation.py is added where you can generate a sequence of arbitrarily length and also finish a beginning of a sentence. For more information see the README in the assignment 2 folder.

Question 2.1 b

First the model is trained and a couple of models are saved which are trained to different steps: 1000, 2000, 5000, 10000, 40000. A couple of sentences that the model has as output are given below. With number of steps reported after the sentence.

- (A that the same the same the s (1000 steps)
- ing the same and the same and t (2000 steps)
- What is a single theory of the (5000 steps)
- How is a state of the same thin (5000 steps)
- Real Relativity is a statement (10000 steps)
- $-(Y) + (1) \times P(Z1) = P(Z1$ (10000 steps)
- I am not a single probability o (10000 steps)
- Knowledge of Science and Social (40000 steps)

- When you can be able to be able (40000 steps)

A few things that were noticable is that the words "the same" and "probability" occurred very often. Also the model nearly always outputted the above sentence at step 1000. At later steps the variability was larger and it also was less likely to loop (e.g. the same the same the same...). However sometimes it still has loops at even 40000 steps. Also the accuracy of the model didn't really improved after step 5000, but the sentences got slightly better over time, better in the sense of more variability in the words and less loops. Another funny thing is that it also has learned the correct syntax for formulas.

Question 2.1 c

Below the results for a model trained until step size 40000 and different temperature values. T = 0.5

- Universes. If you extend that a
- 't have to try to be one of the
- When you don't design the consc
- Knowledge of an answer of the m
- For were be able to concept and

The model diversity improved without hurting the correctness of the sentences T = 1

- when I would come possible!
- ows of "human," says. Result, I
- Standredenlade of Cluessed Co
- Returns starting wrong! Albert
- #7, no. D isn't say, everything

The randomness increases and the correctness of the sentences decreases, they start to make less sense. Sometimes I also get words that are not correct. T = 2

- 'rapwlegualization tigerage
- way, i, noce+tialpressedacely
- ! of ideam kablzolime -60 lie?
- fisn's unougibbling lanques's h
- nvenay, Dashemotely E". Philory

The results are very random and the model doesn't put correct words out anymore. The conclusion from this is that T=0.5, seems to work best. It worked better than fully greedy (always picking the max), but still has enough diversity.

Bonus Question 2.2

First we will analyse a few longer sentences with a sequence length of 200.

- $(A \leftarrow \neg A, C, Y) + P(Z2Y|Z|i) \times P(Y|Z|i))$ is the true and for the territory experimental own general process, and the explanation of the word "more consciousness" and the concept the self-magic is
- Evolutionary Decision Practices belief, and Perhaps and I regardless the same constitute the consciousness that has an experiment with their life cause of the core the beliefs and the world, and a self
- In the word and some minds who claim that were mind to confirmed the one of the prior world that seems that a human member and all the same called that if the probability of scientists are that the r

As can be seen the English words are correct and short sentences do make some sense. Also it seems to correctly place comma's and quotes, but doesn't use any end of sentence points ('.'). Also it rapidly switches to different contexts and the final meaning of the sentence is garbage. This is mostly likely due to training on short sequences of length 30.
Next we will test if it can finish some homemade sentences. I have feeded in a sequence of predefined characters and each time re-use the hidden state produced by those characters to make a next prediction. Below are some interesting examples.

Input	Completed sentence
1982)	10. Philosophical Art of Slovic Cany of Separate and Science Because People are start of
pr	probability of evidence that seem that the mass experimental supported in the first probability is not
P(X|Y)	$= P(A, B) \leq P(A,$ B)
physisicts who says:	"I have a reasoning people are logically being the real problem, and all a replied that what it woul d
Daniel Kah	Daniel Kahneman is the System Artificial Intelligence I am not already be mysterious would be important of the

A few things to notice are

- It can correctly complete a word. E.g. pr is probability. Also Daniel Kah always becomes Daniel Kahneman, often followed by the word is or at least a verb.
- It correctly completed formulas like $P(X|Y)$, this example is a bit cherry picked and it doesn't always give the right syntax, but it does reasonably often.
- After the sentence "who says:" it nearly always start with quotes, because someone is saying something, although it often forget to close them.
- Years like 1982 followed by a bracket are nearly always end of chapter citations, so what follows is a new chapter.

3 Graph Neural Networks

Question 3.1a

$$H^{(l+1)} = \sigma\left(\hat{A}H^{(l)}W^{(l)}\right) \tag{6}$$

$$\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}} \tag{7}$$

$$\tilde{A} = A + I_N \tag{8}$$

$$\tilde{D}_{ii} = \sum_j \tilde{A}_{ij} \tag{9}$$

Each update 6 can be seen as a forward pass through the network. In each forward pass the features are updated. matrix H is a feature representation of the whole network where each row represent an embedding for a particular node, this H will each time be updated by each forward propagation. W, the weight matrix basically determines how important each connection in the graph is, how much weight should it give. By multiplying the adjacency matrix (which describes the structure of the graph) each node is effectively aggregating the information from its neighboring nodes because the adjacency matrix has a 1 in a particular position if there is a connection and a 0 if there is no connection. This aggregating of information from neighboring nodes can be seen as a form of message passing.

Question 3.1b

You will need to stack 3 GCN layers to propagate the information from nodes 3 hops away in the graph. Every forward propagation collect the information of its direct neighbors. So the first forward pass collects everything from 1 hop away, the second from 2 hops and the third from 3 hops away.

3.2 Applications of GNNs

Modeling physical systems. Objects that are for example connected through springs or in any other way inherently influence each other and can thus are in a certain relationship to each other which can be modelled by a GNN which could then make predictions about the future trajectory of certain objects or even infer the dynamics of a particular system by looking at the trajectories.

Discovering new chemical structures. Molecules posses also a graphical structure that can be exploited by GNNs. By training on existing molecule graphs there can be reasoned about new molecules which can help to discover new drugs or other interesting chemical structures.

Classical combinatorial optimization problems such as the traveling salesman, where you need to find the shortest path, this problem is also graphical in nature.

Finally GNNs are also being used for problems that are not inherently graphical like text classification, object detection and machine translation (2).

3.3 Comparing and Combining GNNs and RNNs

Question 3.3a

Both GNN and RNN using a weight matrix that is shared across the system. However RNNs have self-loops and are therefore able to exploit information from previous time steps, GNNs on the other hand have particular connections to some nodes and not to others thus exploiting the relationship between certain things. This differences reflects the kind of applications that are useful for one of them. For example in time-series prediction, where you have a signal across time and need to predict the next timestep an RNN will most likely outperform a GNN because the next time step depends a lot on the previous time steps. A GNN on the other hand will outperform in domains like modeling physical systems as mentioned above, since those objects do have close relationships and encoding those relationship will increase the performance of future predictions.

Question 3.3b

A classical example that can be seen as a combination of RNN and Graphs are Tree LSTMs. Since a tree can be seen as a graph. This allows the model to both consider previous timestep as well as the dependencies of the sentence which provide useful extra information in order to do all kind of tasks like sentiment analysis, POS-tagging or text generation.

References

[1] R. Pascanu, T. Mikolov, and Y. Bengio, “On the difficulty of training recurrent neural networks,” in *International conference on machine learning*, pp. 1310–1318, 2013.

[2] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun, “Graph neural networks: A review of methods and applications,” *arXiv preprint arXiv:1812.08434*, 2018.