DL: Assignment 2

Jonathan Mitnik
MSc Artificial Intelligence
University of Amsterdam
jonathan@student.uva.nl

Abstract

In this report, the main focus is to go through the assignment

1 Recurrent Neural Networks

1.1

1.1.a

The gradient can be calculated as follows:

$$\frac{\delta L^{(t)}}{\delta \mathbf{W}_{ph}} : \boxed{\begin{aligned}\frac{\delta L^{(t)}}{\delta \mathbf{W}_{ph}} &= \frac{\delta L^{(t)}}{\delta \hat{\mathbf{y}}^{(t)}} \frac{\delta \hat{\mathbf{y}}^{(t)}}{\delta \mathbf{p}^{(t)}} \frac{\delta \mathbf{p}^{(t)}}{\delta \mathbf{W}_{ph}} \\ &= \frac{\delta L^{(t)}}{\delta \hat{\mathbf{y}}^{(t)}} \frac{\delta \hat{\mathbf{y}}^{(t)}}{\delta \mathbf{p}^{(t)}} \frac{\delta}{\delta \mathbf{W}_{ph}} \mathbf{W}_{ph} * h^{(t)} + b_p \quad = \frac{\delta L^{(t)}}{\delta \hat{\mathbf{y}}^{(t)}} \frac{\delta \hat{\mathbf{y}}^{(t)}}{\delta \mathbf{p}^{(t)}} h^{(t)}\end{aligned}}$$

1.1.b

For the next gradient, we need to keep in mind the temporal and recursive nature. As such, we can define it as: $\frac{\delta L^{(t)}}{\delta \mathbf{W}_{hh}}$:

$$\boxed{\begin{aligned}\frac{\delta L^{(t)}}{\delta \mathbf{W}_{hh}} &= \frac{\delta L^{(t)}}{\delta \hat{\mathbf{y}}^{(t)}} \frac{\delta \hat{\mathbf{y}}^{(t)}}{\delta \mathbf{p}^{(t)}} \frac{\delta \mathbf{p}^{(t)}}{\delta \mathbf{h}^{(t)}} \\ &= \sum_{i=0}^{T=t} \frac{\delta L^{(t)}}{\delta \hat{\mathbf{y}}^{(t)}} \frac{\delta \hat{\mathbf{y}}^{(t)}}{\delta \mathbf{p}^{(t)}} \frac{\delta \mathbf{p}^{(t)}}{\delta \mathbf{h}^{(t)}} \frac{\delta \mathbf{h}^{(t)}}{\delta \mathbf{h}_i} \frac{\delta \mathbf{h}^{(i)}}{\delta \mathbf{W}_{hh}} \\ &= \sum_{i=0}^{T=t} \frac{\delta L^{(t)}}{\delta \hat{\mathbf{y}}^{(t)}} \frac{\delta \hat{\mathbf{y}}^{(t)}}{\delta \mathbf{p}^{(t)}} \frac{\delta \mathbf{p}^{(t)}}{\delta \mathbf{h}^{(t)}} \prod_{j=i+1}^{T} \Big(\frac{\delta \mathbf{h}^{(j)}}{\delta \mathbf{h}_{j-1}} \Big) \frac{\delta \mathbf{h}^{(i)}}{\delta \mathbf{W}_{hh}}\end{aligned}}$$

1.1.c

There is a very clear distinction between the former formula and the latter: the derivative of the latter does not depend on the previous time-steps. Essentially, the matrix mapping the output only depends on the last hidden state. When you compare this to the recursive nature of $\frac{\delta L^{(t)}}{\delta \mathbf{W}_{hh}}$, one important practical difference becomes clear: how "far" the gradient has to traverse to update the weights. For

W_{ph}, the gradient depends only on the latest hidden state, which means that the gradient and updates generally are stronger. With W_{hh}, however, the jacobian product intuitievly either "explodes" the gradient, or it weakens it, "vanishing" the gradient in the process the more time-steps we go back, due to its recursive nature.

1.2

1.2.a

Gates, go as follow:

1. Input modulation gate: this gate is responsible for actually calculating new values that are going to be eventually added to the cell-state. These values are modulated further with tanh, as to ensure that gradients remain stable by not exceeding 1 / -1 (as tanh does).
2. Input gate: this gate acts as "differentiable" mask for the input modulation gate. It decides how much of the input modulation gate will be added on top of the current cell state. The sigmoid allows this "mask-like" functionality to be applied, ranging between 0 and 1.
3. Forget gate: this gate is responsible for deciding how much of the previous cell state still remains (based on the current input and hidden state). It does so, again, by using sigmoids' range of 0 and 1 to decide whether to maintain nothing (0) or all (1).
4. Output gate: this gate uses another "sigmoid-mask" to decide how much of the cell-state is used as hidden-state, possibly more short-term oriented. Before that, the cell-state is put through tanh to ensure that the value after its previous update is more stable again, max 1.

1.2.b

The number of parameters are $4 * \Big((N_{hidden} * N_{hidden}) + (N_{hidden} * N_{input}) + N_{hidden} \Big) + (N_{hidden} * N_{output})$.

1.3

During training, the model was trained a number of times. To be more specific, the model was trained on three seeds (42, 1337 and 1994), with varying sequence lengths eventually becoming 41 and 81 ((10, 20) * 4 + 2 -1). Both the training loss and training accuracy can be seen in the figure below, where the standard deviation for every 10th step defines the "error-boundary", across seeds. Two noticeable differences occur. Whereas sequences of 41 lenght eventually reach a proper convergence (across all seeds), sequences twice as long still have a very high variance in both loss and and accuracy. However, this might be a simple shift of increased variance that sequences of length 41 had (from timestep 500(=50*10) to 2000(=200*10)); if the model were to continue to train, the model might reach similar convergence.

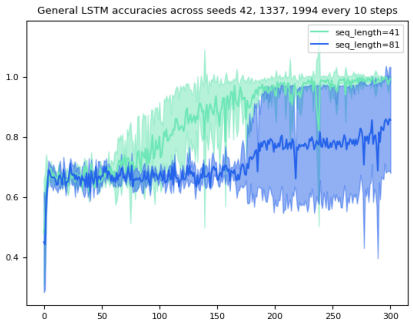
1.4

Now, with PeepLSTM, the results across sequence lengths look more similar to one another. The mean performance for both the longer sequence and shorter sequence seem to be much closer now, hinting at the idea that the peepholes may make models more robust in general for longer sequences (even if the variance is still relatively high). What's more, the signal averages seem more noisy, indicating slightly less stable predictions in general (notice both losses and accuracies seem to be contain more discord). The true power of the peephole LSTM is mostly visible when looking at the general increase in performance speed for lenght 81: where the vanilla seemed to plateau for a while (likely due to the output gate being closed), Peephole LSTMS bypass some of the problems by provind this extra signal.

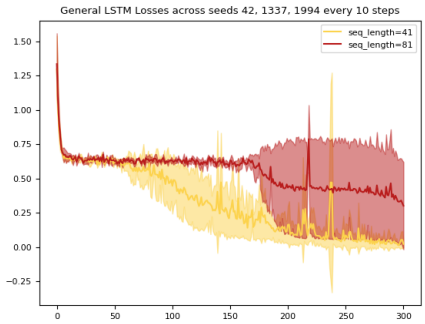
2 Recurrent Nets as Generative Model

2.1a

The parameters that were experimented with (independently) were the learning rate (0.0002, 0.002, 0.02), as this can schedule what performance can be reached under different levels learning rate.

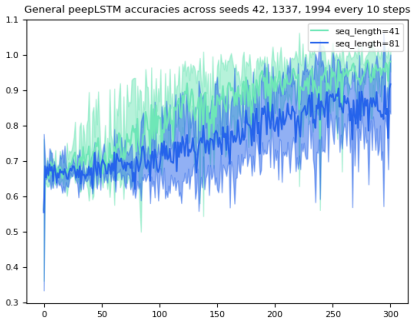


(a) LSTM accuracy averaged

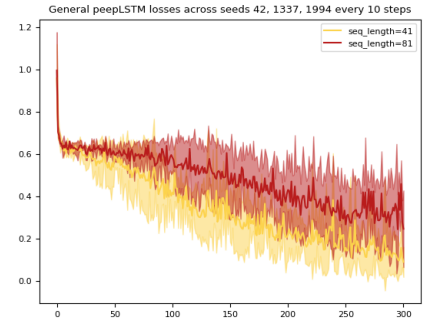


(b) LSTM loss averaged

Figure 1: LSTM Performance over time: note, these are estimates stored every 10 steps of (3000), thus showing 300 recordings



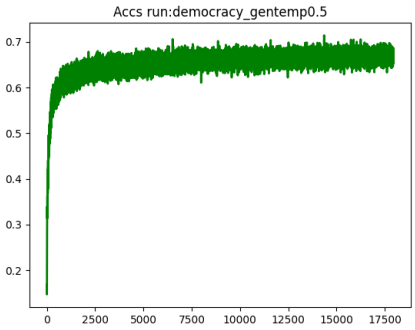
(a) PeepLSTM accuracy averaged



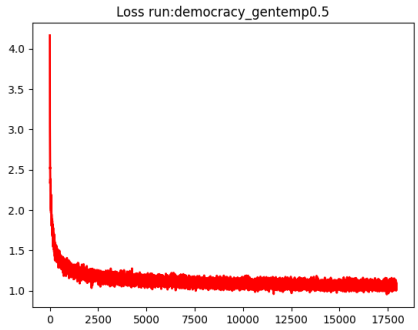
(b) PeepLSTM loss averaged

Figure 2: PeepLSTM Performance over time: note, these are estimates stored every 10 steps of (3000), thus showing 300 recordings

Furthermore, the sequence length of our data was tested (30, 60, 80), as shown earlier, to test how well it performs on longer dependencies. Finally, also differnet amount of layers are tested(1,2,3) as well as number of hidden neuron (64,128,256). These were chosen simply to test the effect of deeper / wider LSTM networks. Finally, this was trained primarily on the provided Democracy in the US book. The optimal paramters (independently) were found to be $lr = 0.02, seqlength = 30, nrlayers = 2, nrhidden = 256$, which was run separately al-together. However, it is important to note that within the same dataset, given sufficient training time, the differnece in training is neglible, and learning-rate becomes the most essential parameter. Because of this, any figure would show similar pattern, converging sooner or later at around an Accuracy of 0.65.



(a) Accuracy for best model



(b) Loss for best model

Figure 3: Loss and accuracy for best model

2.1b

LEN=30
After seeing 100 samples (starting with J) J on the the the the the the t
After seeing 3900 samples (starting with 6) 60,000 miles of the Union the
RAfter seeing 12000 samples (starting with R) Republican courts of the State
After seeing 16000 samples (starting with X) XVIII: Future Condition is alw
AFter seeing 18000 samples (starting with Y) You may be adopted by the cont
LEN=50
Seeing 100 samples (starting with J) Gthe the the the the the the the the the the t
Seeing 3900 samples (starting with g) great an individual and the present the present th
Seeing 12000 samples (starting with W) When the principal constitution of the United Stat
Seeing 16000 samples (starting with t) the principle of the United States the same power
LEN=20
Seeing 99 samples (seeing Q): Qrrrrr
Seeing 3900 samples (seeing I): In the United States
Seeing 12000 samples (seeing J): Jects of the country
Seeing 16000 samples (seeing 6): 6, and the present t
Seeing 18000 samples (seeing V): Vigrania is the cons

In above box, a few sentense excerpts are picked apart, based on previously "best" model trained on the Democracy in the US book. What became very apparent during running and generating sentence, is a strong and clear bias for particular phrases (principal, United States, principle, constitution, country). Some of these excerpt project that, but this becomes more paprent with logner sentences, as the model has a higher likelihood to generate onf oits biases. Fortunately, after 1000 iterations or so, the model becomes successful at generating readable and coherent sentences (even if somewhat repetitive). However, no mattter how readable, it is clear that the logner sentences get, the more the model seems to have the tendency to repeat certain phrases (great an individual and the present the present). These phrases seem to be most typical and indicative of this dataset, so if these words were to be reweighed, it might be possible that more unique sentences could be generated.

2.1c

Temperature is added to the exponents in a softmax to "punish" larger numbers more. What that essentially means, is that the original probabilities in favour of common words (such as "United States") previously should get scaled back a little bit, and allow the network to make slightly less predictable predictions. However, important to note is that in this case, we use the 'inverted' temperature $(1/T)$, so the lower temperature, the more high probabilities get punished. When applied to the democracy dataset using the same settings as were chosen in the previous example, we can absolutely recognize this in the generated sentences:

For example, these are the sentences that are sampled for our "democracy" data-set:
Temperature 2
- Majority which are - 0 from the law with the admin - European nations - Zenehwer the exerciss
Temperature 0.5
- lafue loat, asing tadelent has - Q Fedent parvey, nome. From ways yir thellet, ... hanized
For the dutch dataset about Darwin's travels around the world:
Temperature 2: - pleken wij daarop het hoofd do - in den grond niet gescheiden - p van de boomen en schildwacht - één klein manier is met de
Temperature 0.5:
- kogla melf, Lier noindent - Fraai. Eenkel - äwonia 20: - Oze fron, Wnaslage Nuargo

These sentences show the difference in temperature more than enough. Both for the English and Dutch language, a low temperature produces nonsenical but syntactically similar sentences. The vocabulary is non-existant, but the usage of punctuation works similar to the more coherent (but boring) predictions when temperature = 2. Greedy sampling, however, was most predictable, as it seemed as if sentences somehow kept reocccuring, even if only paraphrased.

3 Graph Neural Networks

3.1 GCN Forward Layer

3.1.a

The structure of the graph can be found represented in $\hat{A}$, which is composed of the Diagonals $\tilde{D}$ (meaning how many neighbours a node has) and $\tilde{A}$, representing the edges of the nodes (the adjacency matrix). This multiplication will instruct which neighbouring nodes will receive the message formed by the $W^{(l)}$-projection of $H^{(l)}$ from the prior layer, and how much of it (this is the normalization step). This message as such is shared among neighbours, and propagated through the layers.

3.1.b

One major problem, is when nodes have the same neighbours: because the input for all nodes are shared equally across neighbours, including a node's own input, the output will lose the distinguishability of a node's meaning. An alternative to this, would be to use *attention* to let a model learn the weighed averages of neighbours rather than treating them equally.

3.2.a

$$\tilde{A} = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$

3.2.b

The number of updates this would take is 4. The first update would go from C to D, then this would be passed both to F and B, then on the third update they both pass some part of C to A, which on the fourth update will be passed to E.

3.2 Graph Attention Networks

What we need to add to the existing formula, is obviously some coefficient to express the attention relationship between i and j. This is introduces an MLP a which maps the concatenation of i and j to a relationship, a LeakyRelu which ensures that attention remains dependent on the query, and a softmax to scale the attention to become a probability distribution.

$$h_i^{(l+1)} = \sigma\Big(\sum_{j \in N(i)} \frac{exp(LeakyRelu(a * [Wh_i || Wh_j]))}{\sum_{k \in N_i} exp(LeakyRelu(a * [Wh_i || Wh_k]))} * W^{(l)} h_j^{(l)} \Big)$$

3.3 Applications of GNNs

A first example could be for instance improving a recommender system with knowledge graph information, as detailed in [2]. For instance, when recommending movies to a user, a movies can utilize as connections such as other movies a leading actor played in, or perhaps movies a friend enjoyed as well. This is in general an edge-level task. A node-level use-case could be enhancing embeddings via concatenation, such as is done in [3]. When classifying books (nodes), one could classify these with BERT embeddings, but also use related structural information such as author.

3.4 Comparing and Combining GNNs and RNNs

3.4.a

Any spatial information that is not 1-dimensional likely benefits more form GNNs than RNNs. In its traditional form, RNNs do not consider more than a linear dimension. Therefore, images will work better with a GNN that accounts for its entire neighbourhood. However, GNNs are less robust when it comes to particular order, such as an important sequence of text: RNNs traverse these sequences with an emphasis on order, therefor being able to parse translation between languages better.

3.4.b

In [1], a summarization task is discussed, where a bit of text is processed using traditional RNN encoder style. This is then enriched by inserting these token representation into a gated GNN, which borrows gating techniques as proposed by the LSTMs. The GNNs can then utilize supervised relationships of these tokens/embeddings, and pool the resulting node-transformations into a decodable summary.

References

[1] Patrick Fernandes, Miltiadis Allamanis, and Marc Brockschmidt. Structured neural summarization, 2020.

[2] Qingyu Guo, Fuzhen Zhuang, Chuan Qin, Hengshu Zhu, Xing Xie, Hui Xiong, and Qing He. A survey on knowledge graph-based recommender systems, 2020.

[3] Malte Ostendorff, Peter Bourgonje, Maria Berger, Julián Moreno Schneider, Georg Rehm, and Bela Gipp. Enriching BERT with knowledge graph embeddings for document classification. *CoRR*, abs/1909.08402, 2019.