Recurrent Neural Networks and Graph Neural Networks

Ivo Verhoeven
13013319
Deep Learning
University of Amsterdam
Amsterdam, 1098 XH
`ivo.verhoeven@student.uva.nl`

1 Recurrent Neural Networks

1.1 Vanilla RNNs

Question 1.1

(a) The gradient for the prediction layers weights follow as,

$$\begin{aligned}\frac{\partial \mathcal{L}^{(T)}}{\partial \mathbf{W}_{ph}} &= \sum_{t=1}^{T} \frac{\partial \mathcal{L}^{(t)}}{\partial \mathbf{W}_{ph}} \\ &= \sum_{t=1}^{T} \frac{\partial \mathcal{L}^{(t)}}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{p}^{(t)}} \frac{\partial \mathbf{p}^{(t)}}{\partial \mathbf{W}_{ph}} \end{aligned} \tag{1}$$

(b) The gradient for the hidden state propagation weights follow as,

$$\begin{aligned}\frac{\partial \mathcal{L}^{(T)}}{\partial \mathbf{W}_{hh}} &= \sum_{t=1}^{T} \frac{\partial \mathcal{L}^{(t)}}{\partial \mathbf{W}_{hh}} \\ &= \sum_{t=1}^{T} \frac{\partial \mathcal{L}^{(t)}}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{p}^{(t)}} \frac{\partial \mathbf{p}^{(t)}}{\partial \mathbf{h}^{(t)}} \frac{\partial \mathbf{h}^{(t)}}{\partial \text{tanh}} \frac{\partial \mathbf{W}_{hh}\mathbf{h}^{(t-1)}}{\partial \mathbf{W}_{hh}} \\ &= \sum_{t=1}^{T} \frac{\partial \mathcal{L}^{(t)}}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{p}^{(t)}} \frac{\partial \mathbf{p}^{(t)}}{\partial \mathbf{h}^{(t)}} \prod_{i=1}^{t} \frac{\partial \mathbf{h}^{(t-i+1)}}{\partial \text{tanh}} \frac{\partial \mathbf{W}_{hh}\mathbf{h}^{(t-i)}}{\partial \mathbf{W}_{hh}} \end{aligned} \tag{2}$$

(c) In the first the gradients are independent of previous time-steps. As such, the information passed to the parameters are summed.

In the second the recurrent terms become understandably problematic. The nested weights require unrolling the computation graph, and multiplying the gradients through each of the successive time-steps. Thus, small gradients are likely to become smaller and smaller until vanishing, or vice versa. Worse than that, these gradients are also attenuated by derivative of (in this case) the hyperbolic tangent activation function. This function is know to cause issues with vanishing gradients, *especially* in deep networks.

1.2 LSTM Network

Question 1.2

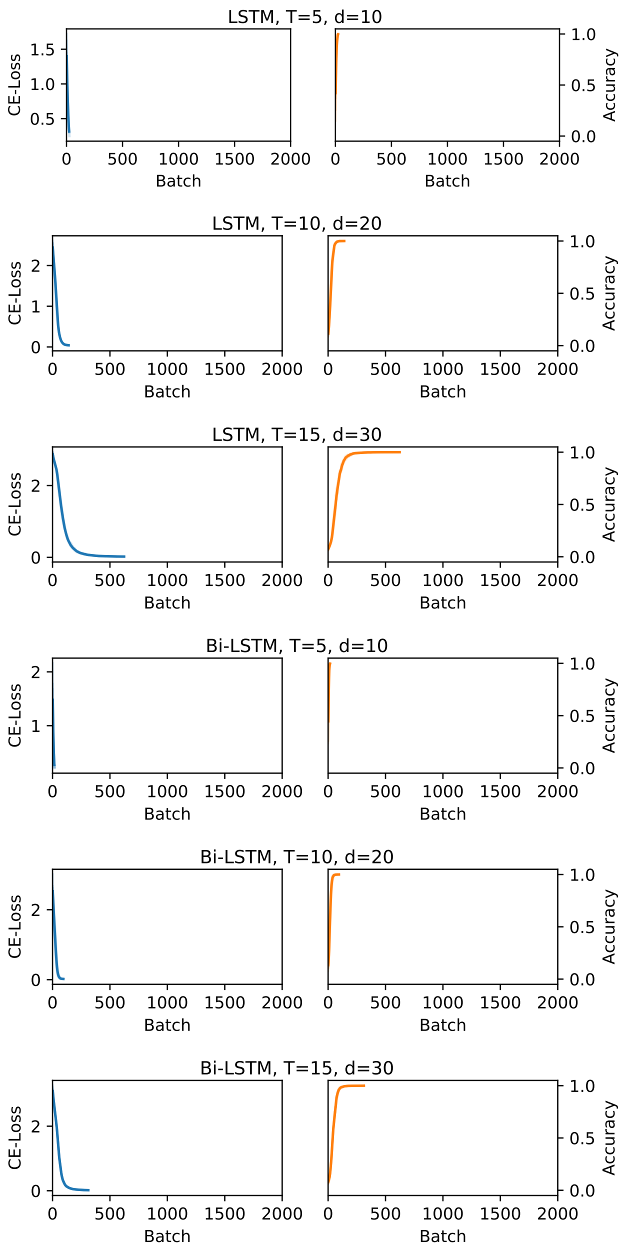

Figure 1: The loss plots for the various (Bi-)LSTM models. The dark lines give the mean loss or accuracy, over the different seeds, smoothed for clarity's sake. Early stopping was applied, given that all models converged to 100% accuracy well before the end of training.

(a) **Input Modulation**: raw input cannot possibly be dumped onto the current cell state. Instead, an update step for the cell state is proposed. This is done via the input modulation gate, which uses a tanh activation, and outputs values between $[-1, 1]$.

Forget: logically, some information in memory is no longer relevant to the current context. The forget gate uses a sigmoid activation and outputs a value between $[0, 1]$ for each of the stored values, multiplying with the previous cell state.

Input: not all information in the proposed update is relevant for the current cell state. The input gate performs a similar operation as the forget gate, but instead applies it to the modulated input. It uses a sigmoid activation and thus outputs values between $[0, 1]$.

Table 1: Means and standard devations of the models over three random seeds.

Model	Sequence-Length	Loss	Accuracy	Batches till Convergence
LSTM	5	0.29 ± 2.7e-02	1.0±0.0	25.0 ± 2.2
LSTM	10	0.04 ± 1.0e-03	1.0±0.0	124.7 ± 15.1
LSTM	15	0.02 ± 5.9e-03	1.0±0.0	544.7 ± 71.0
biLSTM	5	0.23 ± 7.5e-03	1.0±0.0	18.3 ± 0.5
biLSTM	10	0.02 ± 4.6e-03	1.0±0.0	80.7 ± 12.4
biLSTM	15	0.02 ± 3.8e-03	1.0±0.0	298.3 ± 21.5

Output: The output gate determines which aspects of the cell state are pulled from "memory" for current time-step prediction, and which information is solely kept for future time-steps. This again uses a sigmoid, and gives values between $[0, 1]$.

(b) Note, the dimensions here are written as they are used in the implementation. Since that considers mini-batch updates, the weight multiplications are swapped. Ultimately, this has no effect on the total number of parameters.

$$\mathbf{W}_{?x} \in \mathbb{R}^{N_{\text{input}} \times N_{\text{hidden}}}, \mathbf{W}_{?h} \in \mathbb{R}^{N_{\text{hidden}} \times N_{\text{hidden}}}, \mathbf{b}_{?} \in \mathbb{R}^{N_{\text{hidden}} \times 1}, \mathbf{W}_{ph} \in \mathbb{R}^{N_{\text{hidden}} \times N_{\text{output}}}, \mathbf{b}_{p} \in \mathbb{R}^{N_{\text{output}} \times 1}$$

$$\text{N}_{\text{params}} = \underbrace{4(N_{\text{input}} \times N_{\text{hidden}}) + 4N_{\text{hidden}}}_{\text{Hidden layer}} + \underbrace{(N_{\text{hidden}} \times N_{\text{output}}) + N_{\text{output}}}_{\text{Output layer}}$$

1.3 LSTMs in PyTorch

Question 1.3

For this question, the vanilla LSTM model was implemented and trained on the random combinations data-set for sequence lengths of 5, 10 and 15. All models were allowed to train for a maximum of 2 000 batches, but were stopped early if convergence was likely (100% accuracy on the last 3 batches). Performance was measured on a large (4 096) independent validation set. All models were trained using a batch size of 128, a learning rate of 0.001 and gradient clipping using a max-norm of 10. The most impactful hyper-parameter was the dimensionality of the embeddings, with more dimensions resulting in faster (and guaranteed) convergence[1]. Ultimately, I settled on the rule of dims $= 2 \times$ seq-length.

The loss and accuracy curves for the various LSTMs are shown in the top 3 rows of Figure 3. The results are summarised in Table 1.

Question 1.4

Bidirectional LSTMs were trained using the same set-up to compare their effectiveness. Due to the simplicity of the problem, it is hard to make a direct comparison on measures such as loss and accuracy. Instead, the number of batches until convergence was used (last column Table 1). Here, it did point to biLSTMs having an advantage over plain LSTMs, as these converged marginally faster. Their benefit likely only becomes useful for truly long sequences, with effectively two passes over the data being available by the classification layer.

[1]See Appendix for embeddings of dimension 1. Convergence was difficult, even for the simplest tasks

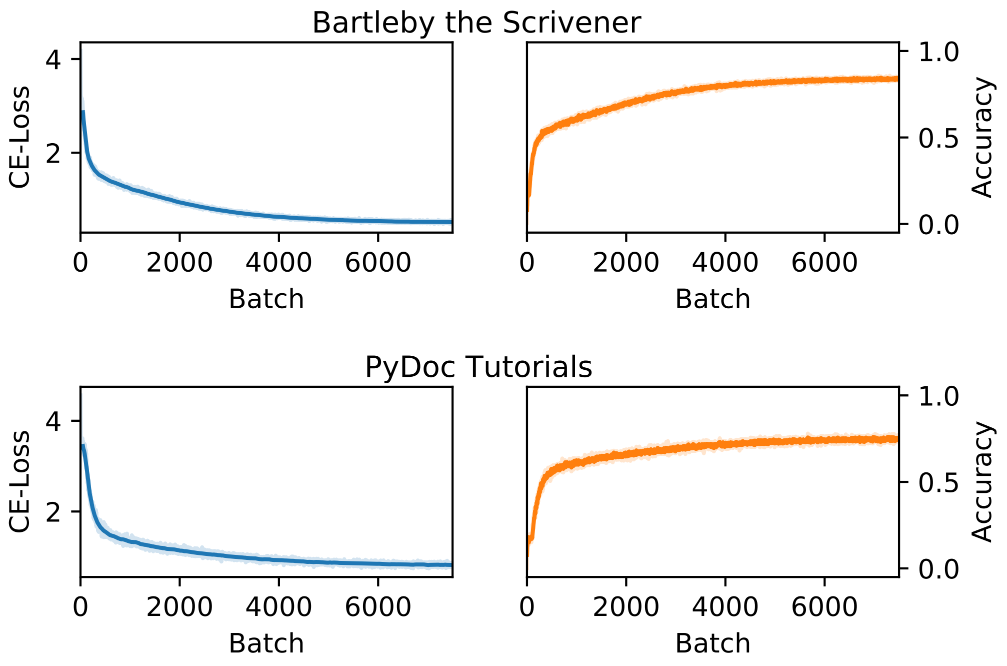

Figure 2: Loss curves for the 2-layer LSTMs trained as next-character prediction models on two different corpora. As usual, dark lines are smoothed (50 batches) and the light are not.

2 Recurrent Nets as Generative Models

Question 2.1 (a)

A two-layer, 1024 hidden unit, LSTM network was trained as a next-character prediction model for sequences of $T = 30$. Explicit teacher forcing was employed, with loss calculated at every time-step by comparing to ground-truth. The same set-up was used for both corpora, namely Melville's "Bartleby the Scrivener"[2] and the combined texts from the Python documentation (PyDoc) tutorials[3] (see Figure 2. Models were trained for 7 500 batches at batch size 64, using the Adam optimizer with an initial learning rate of 0.005 and weight decay of 0.01. At every 75 batches, the learning rate was decayed by 0.995. Ultimately, the "Bartleby the Scrivener" generator achieved a maximum accuracy of 84.22%, whereas the PyDoc tutorials achieved 74.84%. The discrepancy likely comes from the increased complexity from the PyDoc tutorials data-set, with 95 unique characters to 65. In an effort to avoid overfitting a dropout-layer with dropout 0.5 was added.

Question 2.1 (b)

The trained LSTMs can also be used as generative models. Here the a random string is fed into the model at $t = 0$ with the initial hidden state matrix, and the model produces the most likely subsequent characters. These are displayed in Table 2. For both corpora, the model appears to overfit quickly. While impressive that it produces coherent *words* at 5000 steps, which are mostly grammatically correct, they remain nonsensical. By 10000 steps and beyond, it is clear the models start to simply repeat sentences it has seen in the data, evidenced by artefacts like "[A STOR]Y OF WALL STREET." or various headings (e.g. "$\sim\sim\sim$..."" or "* * *..."), but also the semantic clarity suddenly present within the sentences. This was especially prominent in longer sequences ($T > 30$), with whole dialogue being regurgitated. However, it is remarkable that the model output retains some notion of coherence despite including context greater than which it was trained on.

Question 2.1 (c)

[2]`http://www.gutenberg.org/ebooks/11231`
[3]`https://docs.python.org/3/download.html`

Table 2: Some samples from the LSTM models. All strings are of length 30, with trailing spaces removed. For the PyDoc Unicode special characters are left to highlight use of Pythonic syntax.

Sampling Scheme	**Batch**	**Bartleby the Scrivener**	**PyDoc Tutorials**
Greedy	2 500	Produce me, will you do any thi	6797115888337711588833771121181
Greedy	2 500	zed me. I can see that his own	6.0625\n\t»> fruits\n\t['a', '
Greedy	2 500	Of a cold morning, with a grin.	General and specific Python int
Greedy	2 500	My business with a his eyes imp	<stdin>", line 1, in <module> \n
Greedy	2 500	Bartleby was not insensible to	$ass DerivedClassName(Base2, Ba
Greedy	5 000	For a certain melancholy merge	1000000000000555111512312578270
Greedy	5 000	Y OF WALL-STREET.	7.2. Reading and Encoding\n=====
Greedy	5 000	Every copyists or scriveners.	)\n\t»> for i in range(10)
Greedy	5 000	which he delivered in the after	∼∼∼∼∼∼∼∼∼∼∼∼∼∼ etc.
Greedy	5 000	xamine his papers. When I retu	The "str.format()" method of th
Greedy	7 500	* * * * * *	\n\t»> from fibo import fibo
Greedy	7 500	There was no pause for digestio	ZIP for char in reverse('golf')
Greedy	7 500	was the fact, that exactly when	. The "finally" clause is exec
Greedy	7 500	Vyresent I would prefer not to	on of the file will be availabl
Greedy	7 500	Ginger Nut munched his noon app	LookupError', 'UnicodeTranslate
$\tau = 0.5$	7 500	";I not write,?"\n\nI heard a swe	\ \'] =i> 'm','] -> steri'
$\tau = 1.0$	7 500	–copying? Nay\nagain, whatever	You can follows:\n\n »> tel\n
$\tau = 2.0$	7 500	As Nippers once observed, Turke	Python interpreter starts with

Table 3: Sentence completion of the LSTM models from the input string to $T = 100$.

Input String	**Bartleby the Scrivener**	**PyDoc Tutorials**
"I would prefer not to."	"I would prefer not to."\n\n "Prefer not to," echoed I, rising in high excitement, and gently disappear	"I would prefer not to.""\n\tprint("– This parrot wouldn't", action='VOOOOOM', voltage=1000
for i in range	for i in rangely mild, firm voice, replied, "I would prefer not to."\n\n"Prefer not to," echoed I, ris	for i in range()\n\tprint("– This parrot wouldn't", action='voom'):\n\t\tprint("– This p
Do Re Mi	Do Re Might at the prison, and especially in the corner. At about eleven o'clock though, in the aft	Do Re Mided to the file is that supports a number of a function object is called "f.read()" function

The temperature coefficient in a (tempered) random sampling scheme controls the degree of "randomness" by which the samples are drawn. A value of $\tau < 1$ decays the values and smooths the distribution. In the extreme cases, $\tau = 0$, all values are chosen with equal probability. A value of $\tau > 1$ does the opposite, it highlights the differences. In the extreme case $\tau \to \infty$ this becomes equal to the greedy sampling scheme.

This effect is present in the samples generated (see Table 2, bottom 3 rows), with $\tau = 0.5$ producing total nonsense, $\tau = 2$ resembling the structure of using greedy sampling and $\tau = 1$ falling somewhere in between these extremes. Using a tempered $\tau = 1$, at least for this small sample, seems to be a good trade-off between generating accurate sentences and "creativity", with the provided samples being both morphologically and syntactically close to something that might be present in the texts.

Question 2.2

Some sentence completions are displayed in Table 3. While it is remarkable how well the models are capable of picking up on the text styles, with correct structure of dialogue and Python functions, it is clear that it struggles with long inputs, reverting to text it has seen often before rather than playing off of what was given. Regardless, the sentences read like their source material.

3 Graph Neural Networks

3.1 GCN Forward Layer

Question 3.1

(a) This formulation leverages the graph Laplacian, combining both the node neighbours and the node degrees, in matrix $\hat{A}$. The multiplication with the weighted hidden state results in every node updating by subtracting the values in the neighbouring nodes from the up-scaled state in the current node (thus, i.e. a message passing algorithm). Every additional layer allows each node within the network to extend its receptive field to more nodes.

(b) Currently, the re-normalized adjacency matrix is (rather arbitrarily) defined as $\mathbf{A}_N + \mathbb{I}_N$, making connections to neighbours equally important as self-connections. A hyper-parameter $\lambda\mathbb{I}_N$ might allow for overcoming this limitation.

Question 3.2

(a) The adjacency matrix $\widetilde{\mathbf{A}}$ of the graph in the Assignment's Figure 3

$$\begin{array}{c|cccccc} & A & B & C & D & E & F \\ \hline A & 1 & 1 & 0 & 0 & 1 & 1 \\ B & 1 & 1 & 0 & 1 & 0 & 0 \\ C & 0 & 0 & 1 & 1 & 0 & 0 \\ D & 0 & 1 & 1 & 1 & 0 & 1 \\ E & 1 & 0 & 0 & 0 & 1 & 0 \\ F & 1 & 0 & 0 & 1 & 0 & 1 \end{array} \tag{3}$$

(b) The shortest path(s) between nodes C and E is 4 hops. Hence, it will also take 4 updates before information from node C is in E.

3.2 Graph Attention Networks

Question 3.3

Attention, may be induced by incorporating a parameterised function α, for example,

$$\tilde{h}_i^{(l+1)} = \sigma\left(\sum_{j\in\mathcal{N}(i)} \alpha_{i,j}\mathbf{W}^{(l)}h_j^l\right). \tag{4}$$

Specifically, α is a single layer feed-forward network, that combines the computed hidden layer $h_j^{(l+1)}\forall j \in \mathcal{N}(i)$ using two weight matrices, to compute a single value between 0 and 1 (softmax normalized) for all $j \in \mathcal{N}(i)$. This value specifies how important j is for i in $\tilde{h}_i^{(l+1)}$.

3.3 Applications of GNNs

Question 3.4

Polypharmacy:[**zitnik2018modeling**] use a GCN as a link predictor, where links encode side-effects due to combinations of pharmaceutical drugs using a graph of protein-protein and drug-protein interactions. Not only did this outperform similar methods substantially, it also managed to make predictions not in the dataset but later verified in recent publications.

Document Classification: [**yao2019graph**] apply NLP-motivated feature engineering to the adjacency matrix of a document-word co-occurrence graph. Simple, but effective, as it achieves SoTA.

3.4 Comparing and Combining GNNs and RNNs

Question 3.5

(a) RNNs will likely manage to outperform GNNs in cases where is there is a clear, directed flow of information. Models working with a time dimension, where each successive time-step is clearly related to its history but not its future, should be most natural for RNNs. On the ohter hand, cases where there is strong inter-dependency between different relations and states should see GNNs benefit. These will be able to generalise to cases where there is no clear sense of "future"/"past" or "up"/"down", but only that of connectivity.

(b) Sentiment classification of scientific texts remains difficult, likely due to the high register within these documents. As such, detecting the attitude of the author to another author is likely not possible from the text alone. On the other hand, citations form natural networks, with papers referring to related papers. Here, a pure-RNN solution or pure-GNN solution will not succeed, but combined, it might leverage and synergise the information available at both scales.

References

Kipf, T. N., Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv*:1609.02907.

Yao, L., Mao, C., Luo, Y. (2019, July). Graph convolutional networks for text classification. *In Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 33, pp. 7370-7377).

Zitnik, M., Agrawal, M., Leskovec, J. (2018). Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics*, 34(13), i457-i466.

Appendix

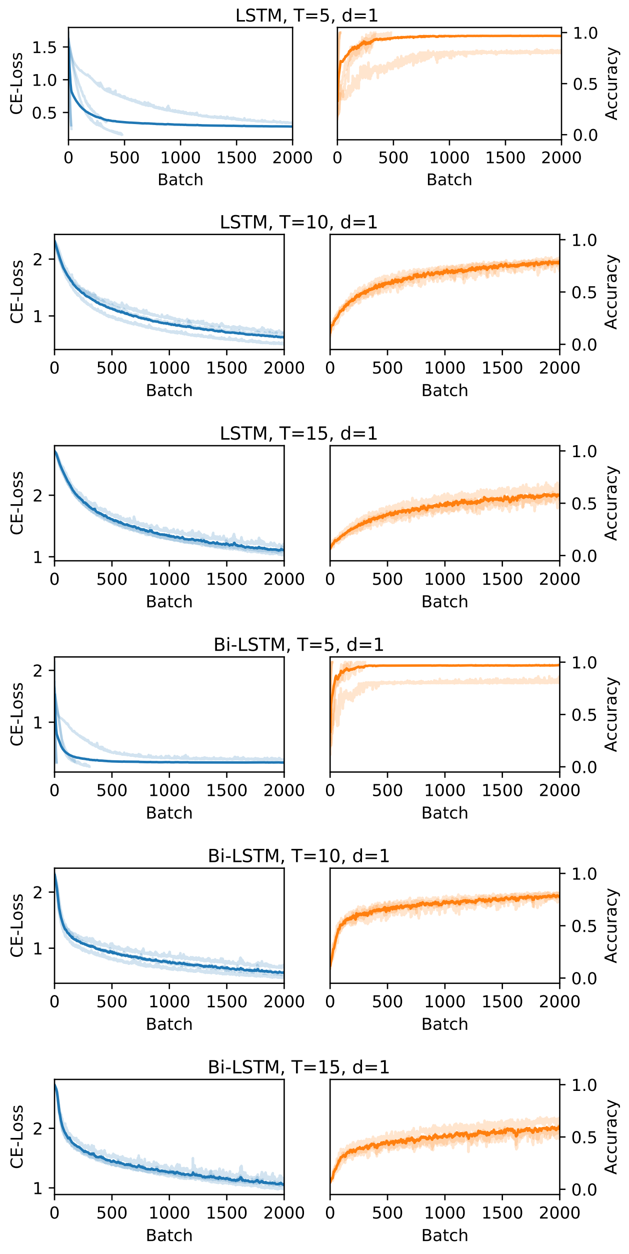

Figure 3: The loss plots for the various (Bi-)LSTM models. The dark lines give the mean loss or accuracy, over the different seeds, smoothed for clarity's sake. Early stopping was applied, given that all models converged to 100% accuracy well before the end of training.