Deep Learning Assignment 2, Fall 2020

Blazej Manczak
University of Amsterdam
`blazej.manczak@student.uva.nl`

1 Recurrent neural network

1.1 Vanilla RNNs

$\frac{\partial \mathcal{L}}{\partial W_{ph}} = \frac{\mathcal{L}}{\partial \hat{y}^{(T)}} \frac{\partial \hat{y}^{(T)}}{\partial p^{(T)}} \frac{\partial p^{(T)}}{\partial W_{ph}}$

$\frac{\partial \mathcal{L}}{\partial W_{hh}} = \frac{\mathcal{L}}{\partial \hat{y}^{(T)}} \frac{\partial \hat{y}^{(T)}}{\partial p^{(T)}} \frac{\partial p^{(T)}}{\partial h^{(T)}} \sum_{t=0}^{T} \frac{\partial h^{(T)}}{\partial h^{(t)}} \frac{\partial h^{(t)}}{W_{hh}} = \frac{\mathcal{L}}{\partial \hat{y}^{(T)}} \frac{\partial \hat{y}^{(T)}}{\partial p^{(T)}} \frac{\partial p^{(T)}}{\partial h^{(T)}} \sum_{t=0}^{T} \Pi_{j=i+1}^{t} \frac{\partial h^{(j)}}{\partial h^{(j-1)}} \frac{\partial h^{(t)}}{W_{hh}}$

We see that $\frac{\partial \mathcal{L}}{\partial W_{ph}}$ depends only on timestep T whereas $\frac{\partial \mathcal{L}}{\partial W_{hh}}$ depends on all the prior timesteps as well. In the formula for $\frac{\partial \mathcal{L}}{\partial W_{hh}}$ the multiplication of gradients can lead to exploding (spectral norm > 1) or vanishing gradients (spectral norm < 1). Dealing with exploding gradients is more straight-forward as we can just clip them. However, the vanishing gradient problem is not that easy to solve. Most importantly, the vanishing gradient problem causes the contribution of the subsequent terms to decrease exponentially (fast) which makes it hard to model long-term dependencies.

1.2 Long Short-Term Memory (LSTM) Network

1.2.1 What do all of these gates do?

- Input modulation gate $\mathbf{g}^{(\mathbf{t})}$: The task of the modulation gate is to add a non-linearity to the data and form a new, potentially relevant memory that could be added to the state. It is later combined with the input gate.
- Input gate $\mathbf{i}^{(\mathbf{t})}$: this gate decides which dimensions of the input are important (more important information has weight closer to 1 and less important closer to 0). It is later combined with the input modulation gate.
- Forget gate $\mathbf{f}^{(\mathbf{t})}$: decides what and how much of the past state should be forgotten (the less important dimensions will decrease towards 0, the more important to 1)
- Output gate $\mathbf{o}^{(\mathbf{t})}$: decides what dimensions of the next hidden state are important

1.2.2 Number of parameters in LSTM

We have 4 gates where each gate has two weight matrices: one that is of a size $N_{hidden} \times N_{input}$ and one that is of size $N_{hidden} \times N_{hidden}$. Additionally, for each of these 4 we have a bias term of size N_{hidden}. Finally we have an additional matrix $W_p h$ for calculating the output that has the size $N_{output} \times N_{hidden}$ and again a bias vector. Putting things together, we have:

$$4(N_{hidden}N_{input} + N_{hidden}N_{hidden} + N_{hidden}) + N_{output}N_{hidden} + N_{output}$$

1.3 LSTMS in PyTorch

Below please find the plots for accuracy and loss for three different seeds (1,2 and 3). Note that the sequence length in the figure descriptions refers to the command line argument $input\_length$ which results in a sequence of length $input\_length - 1$ and a scalar label. All the runs were run with the parameters as specified in Table 2 in the assignment. As it was unclear what dimension for the embedding to use, per Phillip's suggestion, I fixed the embedding dimension to be a quarter of the hidden dimension for both LSTM and biLSTM (32 for settings from the table). I've stopped training when the accuracy was higher than 99.5%. I've experimented to see if the performance deteriorates after these 99.5% are reached but it did not, hence I stop the training once the first perfect accuracy is reached not to clutter the plots. Per Phillip's suggestion on Piazza, upon termination I test the performance of the converged model on 200 batches (having 128 example each). Plots are generated with TensorBoard with 0.6 smoothing (default). Accuracy and loss are logged on every step.

1.3.1 Random Combinations and LSTM

All the models reach perfect accuracy (up to 2 decimal places, evaluated on 200 batches). Hence, the mean accuracy is 1 and std is 0 (the training was less stable for lower dimension of the embedding). Additionally, I report the mean termination step over 3 random seeds (1,2,3) to illustrate the difference between lstm and bi-lstm:

- For T=6: 86
- For T=11: 204
- For T=16: 375

Below we see the accuracy and the loss curve for different sequence lengths. We see three distinct patterns corresponding to sequence lengths 6,11 and 16. The shorter the sequence, the faster the convergence:

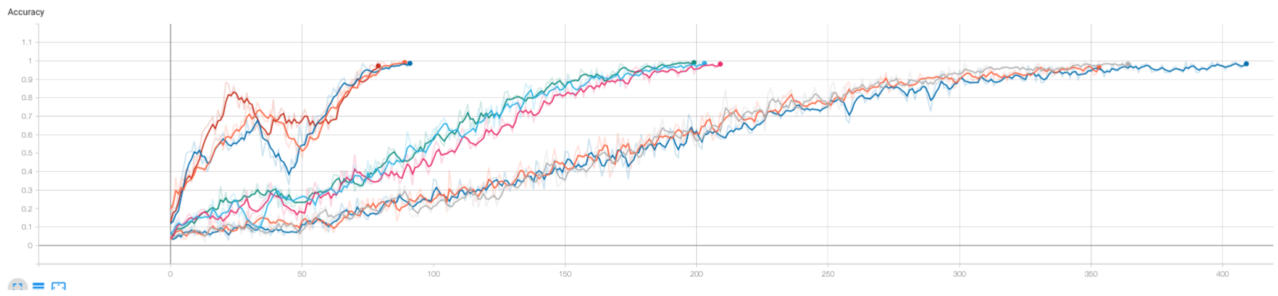

Figure 1: LSTM accuracy for sequence lengths 6,11 and 16 over 3 random seeds.

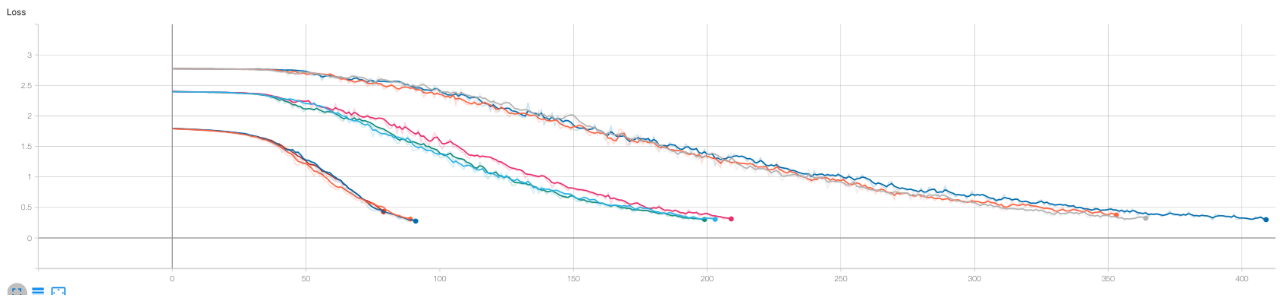

Figure 2: LSTM loss for sequence lengths 6,11 and 16 over 3 random seeds.

1.3.2 Random Combinations and bi-LSTM

All the models reach perfect accuracy (up to 2 decimal places, evaluated on 200 batches). Hence, the mean accuracy is 1 and std is 0 (the training was less stable for lower dimension of the embedding). Additionally, I report the mean termination step over 3 random seeds (1,2,3) to illustrate the difference between lstm and bi-lstm:

- For T=6: 76

- For T=11: 119
- For T=16: 266

For bi-LSTM we again see that the shorter the sequence, the faster the convergence. However, here the results have higher variance of convergence between different seeds. One can still see three distinct patterns however they are more ragged. For example we see that the grey line corresponding to one of the seeds for sequence length 11 reaches perfect accuracy almost as fast as the two of the seeds for sequence length 6.

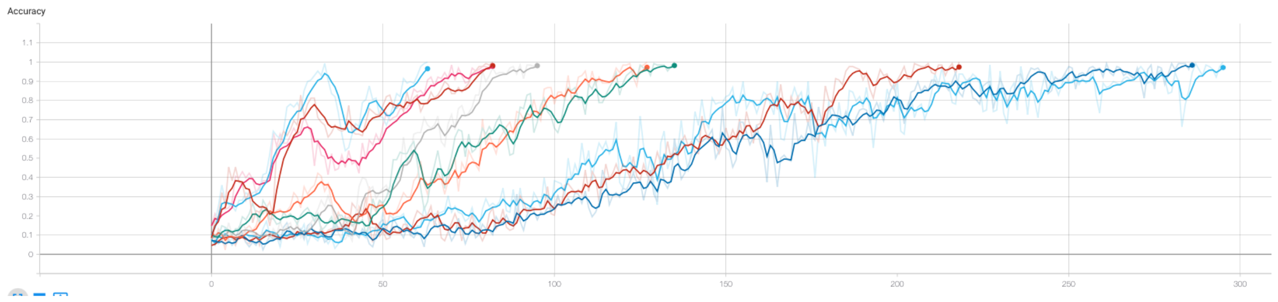

Figure 3: bi-LSTM accuracy for sequence lengths 6,11 and 16 over 3 random seeds.

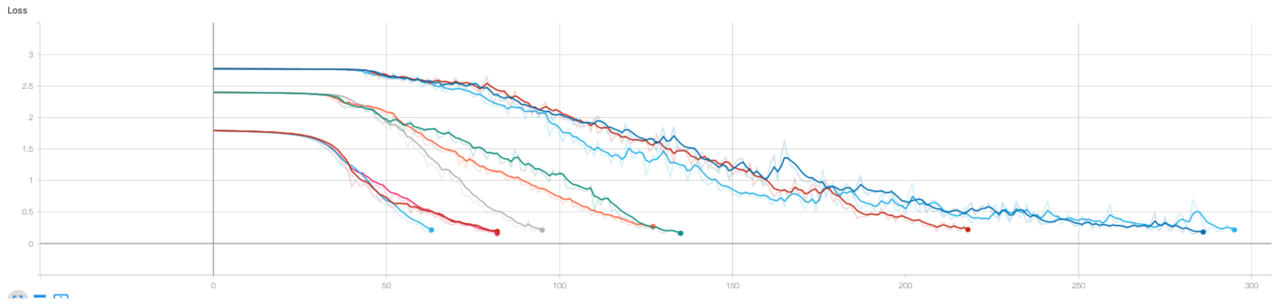

Figure 4: bi-LSTM loss for sequence lengths 6,11 and 16 over 3 random seeds.

Comparing the plots for LSTM and bi-LSTM we see two differences: (1) bi-LSTM converges faster than regular LSTM but (2) has more ragged loss landscape. (2) can be considered a short-coming that can be mitigated by tunning the parameters for bi-LSTM. For example, I noticed that with embedding of higher dimension this problem is significantly smaller.

2 Recurrent Nets as Generative Model

2.1 Two-layer LSTM loss and accuracy curve

I've trained a two-layer LSTM with the following parameters: batch size 256, number of steps 1500, number of hidden units 1024, embedding dimension of 256 and Adam optimizer with the learning rate 5e-4. I've increased the number of parameters for a model to be more expressive. It lead to increase in performance, however as we don't have a validation set, this expressiveness comes down, to some extent, to remembering the training corpus (which result in more coherent generated sentences). I've experimented with dropout and weight decay but they did not offer significant improvements (it probably would on a validation set). Please find the loss and accuracy plots for the democracy dataset below. As I had some troubles with Lisa refusing connection I used google colab to train the model. You can see all the outputs I described below under this link.

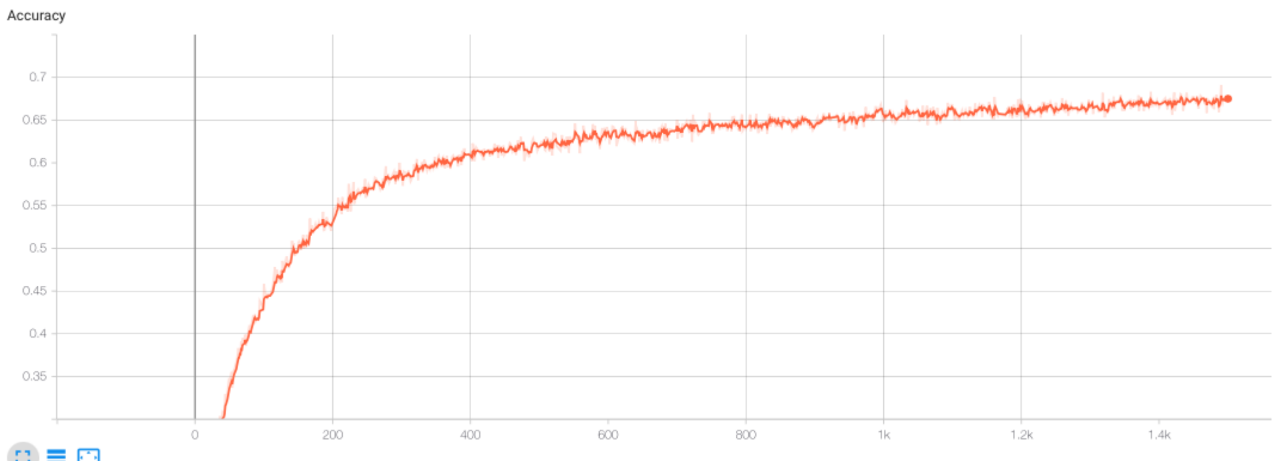

Figure 5: Accuracy for the two-layer LSTM.

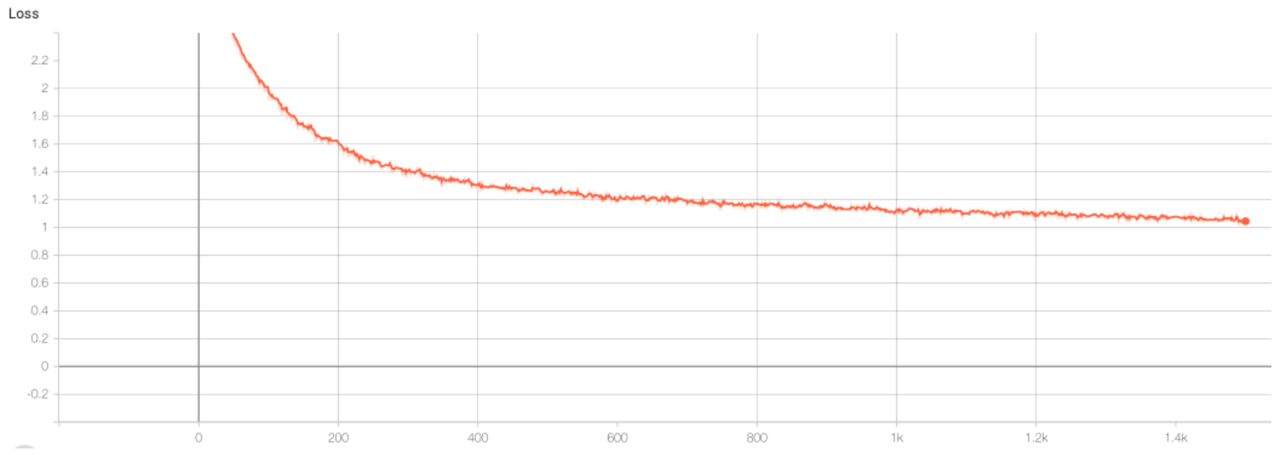

Figure 6: Loss for the two-layer LSTM.

2.2 Generated sentences in course of training

Here is a sample of the sentences generated in 1/3, 2/3 and at the end of the training using the model described in the previous sub-section:

1/3 of training: "? The Americans of the constitu", "West the constitution of the co", "he constitution of the constitu", "on the constitution of the cons", "9 the constitution of the const"

2/3 of training: "] "In the United States are alwa", "% the property of the United St", "Restation of the United States ", "$ the property of the United St", "Mong the property of the United"

End of training: "Government in the United State", "Ken the principle of the peopl", "States which is the people in", "Restraining the United States a", " the present time the present "

As the training process evolves, the dictionary used by the model expands. Moreover,the sentences become less repetitive. The vocabulary is enriched and the coherency increases as more of the training corpus has been iterated over. Even though we see some syntactic and semantic structure for those short sentences, it quickly disappears when we generate longer sequences. However, the disappearance of structure for sentences longer than 30 is to be expected as the gradients are not propagated more than 30 steps back.

2.3 Random sampling

The smaller the temperature parameter, the more 'random' our predictions are as we flatten out the softmax probabilities. Conversely, the bigger the temperature, the elements with big probabilities become even bigger, resulting in the sentences of which the model is more sure.

Here is a sample of the sentences with a fully trained model on the democracy text:

Temperature 0.5: "Dennmond, CEvieol. ehut@ziY a", "nmedwe toworving edZen, they", "y pnaiciol3."]D.V.], per3I' Le"
Temperature 1: "(so circumstances, which have", "Lic. The leader tomely fliend", "Xgent than is more cwovacy is"
Temperature 2: "zend the wealth and which the ", "States is to be a felt and the", "% and the supposition of the c"

We see that a low value for temperature of 0.5 flattens the distribution so much that what is produced can barley lacks meaning and is almost unpronounceable. The temperature of 1 is already a big step up producing mostly proper English words, though some spelling mistakes are still there. For temperature we have almost no spelling mistakes and the text is more significantly more coherent (though still far from perfect).

2.4 Bonus

I've trained on the Democracy book and tested the models capability on a couple of sentences. Here are the results with generated sequence length being 30 (the part in italic has been given, the rest is the model):

The heart of democracy is not always be added to the sa
Liberty is the most powerful influence o
People of the Union is a serious and mo
Equality of the people are not always
Slavery is a mere difficult to discus
America is the most powerful influence o
North America the power of the Unio

We see that there is local semantic and syntactic structure, that diminishes as we look at a broader context. However, even given that the model is simple it's still impressive that the generated text has a seed of sense.

3 Graph Neural Networks

3.1 GCN Forward Layer

3.1.1

The GCN layer exploits the structural information in the graph by using the augmented adjacency matrix $\hat{A}$ during a node feature update step. We see that the term $H^{(l)}W^{(l)}$ is the projection of the node features. This projection stores the (latent) features of each node which we want to share with other nodes, hence this projection is our message. Then, by matrix multiplication with $\hat{A}$, we average the projected features per node with all the neighboring nodes, which can be seen as passing a message from one node to the adjacent rows. To pass messages in bigger neighborhoods one should stack the GCN layers.

One issue for scalability of the GCN layer is that when averaging messages of adjacent nodes we can lose node-specific information. This happens when two nodes have the same adjacent nodes, counting the node itself. Then, even though these nodes started with different feature values, they will end up having the same values. One solution to this issue is to weigh the self connections differently than the connections with the neighbours.

3.1.2

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

It will take 4 updates to propagate the information from node C to E as this is the length of the (shortest) path from C to E.

3.2 Graph Attention Networks

Equation 49 is lacking the attention weight between the node i and j, α_{ij}. The updated equation has the following form:

$$h_i^{(l+1)} = \sigma(\sum_{j \in \mathcal{N}(i)} \alpha_{ij} W^{(l)} h_j^{(l)})$$

Without this weight, we would weigh every neighbor the same which is likely to be less beneficial for the model.

3.3 Applications of GNNs

In generak GNNs can be applied to problems which representations can be portrayed as graphs. The GNN can produce node-level, edge-level or graph-level outputs.

First use case of GNNs is relationship classification/prediction (edge-level task) in a social network. Given peoples relationships and their attributes one might use a GNN to classify the relationship or recommend new friendship suggestions based on a how 'likely/strong' the possible edge would be.

Another example usage of GNNs would be in zero-shot learning (either NLP or CV), where GNNs are used to extract information from knowledge graphs to enable classification on unseen classes.

3.4 Comparing and Combining GNNs and RNNs

In general, GNN's will be more expressive for data in which the order of the connections does not matter. For example, in social networks, one person can be linked to many others and there is no apparent notion of order. However, the algorithms to translate graph to a sequence such as DFS or BFS would impose some artificial order that is really not there. This problem would be amplified if the nodes in the graph have high degrees as there would be more artificial ordering. Conversely, RNNs work better in the cases where order matters, such as language. Since GNNs are local permutation equivariant, it is not straightforward to accommodate learning n-ary (n-children) parse trees.

One example of combining RNNs and GNNs is to use RNNs to aggregate information from neighboring nodes, instead of just using linear operation. This has proven to significantly boos performance in zero-shot learningzero-shot learning paper I mentioned above. Another combined model approach has been taken by this paper. The authors portray the patients records separately as a GNN and as a LSTM and then they combine the output to predict the next prescription.