Deep Learning Assignment 2

Jochem Loedeman
12995282

1 Recurrent Neural Networks

1.1 Vanilla RNNs

Question 1.1

(a)

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

In the computational graph, there is no path from outputs of recurrent hidden states to the loss at T. Therefore, no sum over previous states is present in the expression for the gradient.

(b)

$$\frac{\partial \mathcal{L}^{(T)}}{\partial \boldsymbol{W}_{hh}} = \frac{\partial \mathcal{L}^{(T)}}{\partial \boldsymbol{p}^{(T)}} \frac{\partial \boldsymbol{p}^{(T)}}{\partial \boldsymbol{h}^{(T)}} \frac{\partial \boldsymbol{h}^{(T)}}{\partial \boldsymbol{W}_{hh}} = \sum_{i=0}^{T} \frac{\partial \mathcal{L}^{(T)}}{\partial \boldsymbol{p}^{(T)}} \frac{\partial \boldsymbol{p}^{(T)}}{\partial \boldsymbol{h}^{(T)}} \frac{\partial \boldsymbol{h}^{(T)}}{\partial \boldsymbol{h}^{(i)}} \frac{\partial \boldsymbol{h}^{(i)}}{\partial \boldsymbol{W}_{hh}}$$

$$= \sum_{i=0}^{T} \frac{\partial \mathcal{L}^{(T)}}{\partial \boldsymbol{p}^{(T)}} \frac{\partial \boldsymbol{p}^{(T)}}{\partial \boldsymbol{h}^{(T)}} \left(\prod_{j=i+1}^{T} \frac{\partial \boldsymbol{h}^{(j)}}{\partial \boldsymbol{h}^{(j-1)}} \right) \frac{\partial \boldsymbol{h}^{(i)}}{\partial \boldsymbol{W}_{hh}}$$

In this case, there are paths from nodes in previous timesteps computed with $\boldsymbol{W}_{hh}$ to the loss at T. Therefore, we should account for these dependencies through the chain rule, giving rise to sums and products within the expression for the gradient.

(c) The first and second gradients are respectively independent and dependent on previous timesteps, as was already explained through the connectivity of the computational graph. When training this network for a large number of timesteps, we have to account for very long-term dependencies of the recurrent states through the product in the expression for the second gradient. This product of Jacobians can cause the total gradient to become very small or very large, making learning difficult.

1.2 Long Short-Term Memory (LSTM) network

Question 1.2

(a)

- *Input modulation gate:*
  Proposes an updated cell state. This gate has a tanh activation, which is important to keep the activations of the internal state zero-centered. If for example we used the sigmoid, these states would increase over time.

- *Input gate:*
  Regulates how much of the new value proposed by the input modulation gate is actually used in the updated cell state. Uses a sigmoid to obtain a gating value between 0 and 1.

- *Forget gate:*
  Regulates how much of the previous cell state is retained in the updated cell state. Uses a sigmoid to obtain a gating value between 0 and 1, which corresponds to completely forgetting or strongly remembering the previous state.

- *Output gate:*
  Regulates how much of the nonlinearized current cell state is passed as output of the cell. Uses a sigmoid to obtain a gating value between 0 and 1.

(b)

$$N_{\text{total}} = 4\underbrace{\left(N_{\text{hidden}} \cdot N_{\text{input}} + N_{\text{hidden}} \cdot N_{\text{hidden}} + N_{\text{hidden}}\right)}_{\text{gate input weights, recurrent weights and biases}}$$
$$+\underbrace{N_{\text{output}} \cdot N_{\text{hidden}}}_{\text{output weights}} + \underbrace{N_{\text{output}}}_{\text{output bias}}$$

1.3 LSTMs in PyTorch

Question 1.3

The required LSTM model was implemented in PyTorch, and evaluated using the supplied `train` function and the binary palindrome data. Accuracies were obtained for sequence lengths of 10 and 20, where each hyperparameter setting was evaluated for 3 different random seeds (0, 1 and 2). From these results, the mean and standard deviation for each accuracy value was obtained. This yielded two accuracy curves, one for each sequence length setting, together with the corresponding standard deviations. They are shown in Figure 1, where the shaded areas correspond to the accuracy values that lie within 1 standard deviation on either side of the obtained accuracy value. Notice that for $T = 20$ the standard deviation increases strongly after 1000 steps. At this point, some training runs make a large jump in accuracy, whereas others fall behind. This behaviour seems to be characteristic of the model on this data; the accuracy is stable around 0.55 when suddenly, the model improves relatively quickly to perfect accuracy.

Question 1.4

The peepLSTM model was implemented in PyTorch, using the same hyperparameter settings as before. The results are shown in Figure 2. For $T = 20$ the model converges to perfect accuracy already around 500 steps, which is significantly faster than the original LSTM model. Furthermore, the uncertainty of the accuracy values is much smaller. Apparently, setting the gate values based on the cell states rather than the LSTM outputs is beneficial for the performance of the model on this data set.

2 Recurrent Nets as Generative Model

Question 2.1

(a) The text generation model was implemented using the following hyperparameters:

- Embedding dimension: 64. This value was chosen as it was suggested to use an embedding dimension of $\frac{1}{4}$ of the cell state size.
- Batch size: 64. The observed fluctuations in loss and accuracy are quite large for this value, but we still have steady convergence, and we keep the benefit of being able to escape local minima.

Running the model on Grimms fairy tale dataset for 1500 steps, the loss and accuracy curves are obtained as shown in Figure 3:

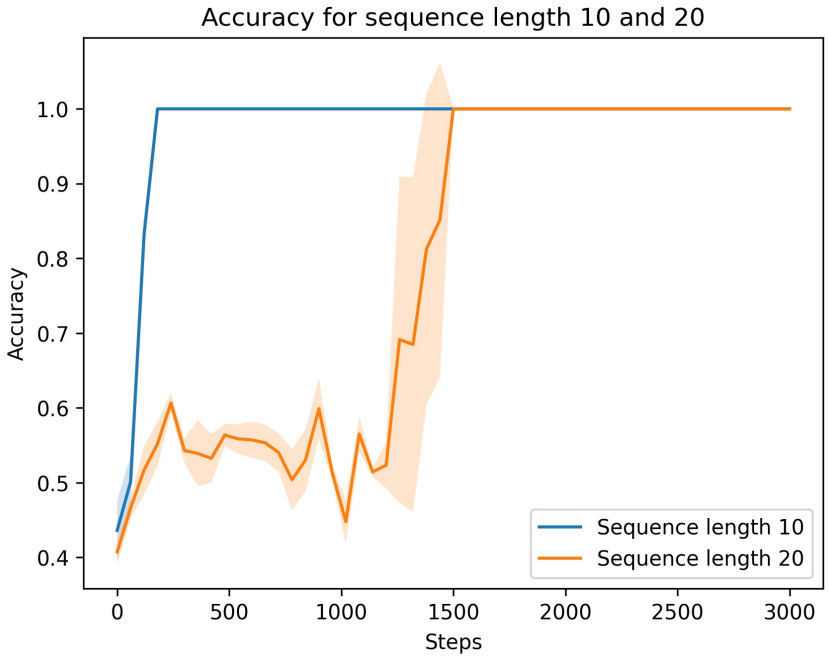

Figure 1: Accuracy of the LSTM on the binary palindrome dataset for $T = 10$ and $T = 20$

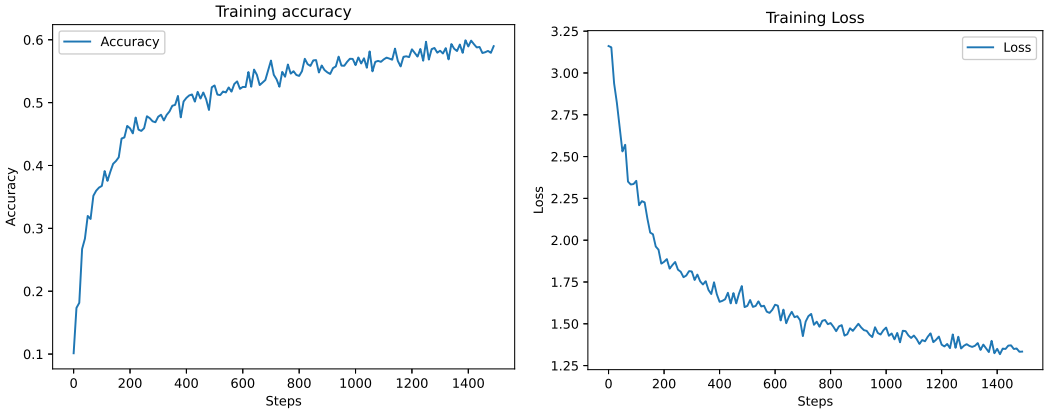

Figure 3: Accuracy and loss on the text generation task. The model is trained on the Grimms fairy tale dataset

Notice that the model has not completely converged at 1500 steps. When the model is run for approximately 5000 steps, it converges to an accuracy of approximately 0.65.

(b) Next, the function `generate_sequence` was implemented. Below, we list 5 sets of 3 sequences that were generated at 1/3, 2/3 and 3/3 of the 1500 training iterations. Each set has a different initial character from which the sequences were generated.

- **Initial character: t**
  1/3: the forest to the forest to the
  2/3: the stood the stread to the sto
  3/3: the world with his father that

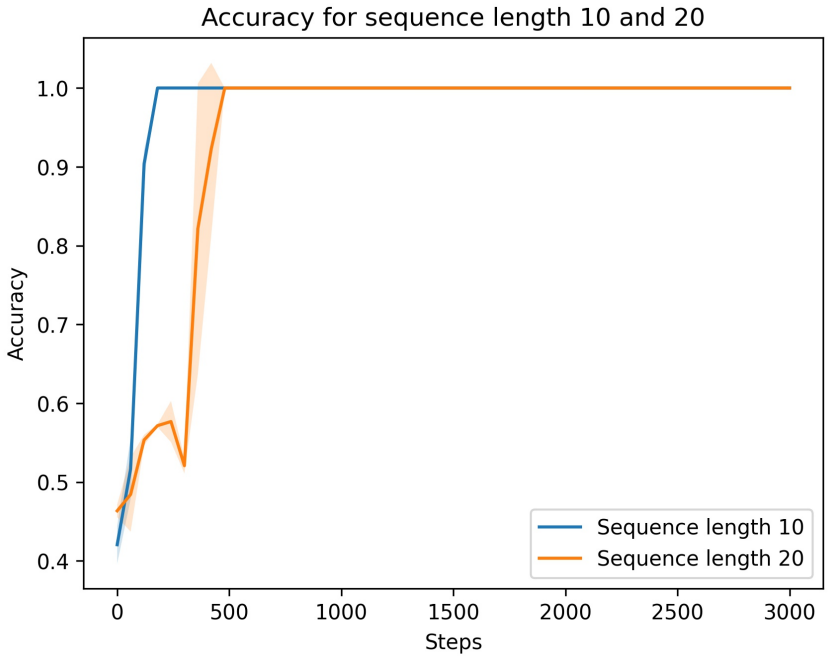

Figure 2: Accuracy of the peepLSTM on the binary palindrome dataset for $T = 10$ and $T = 20$

- **Initial character: a**
  1/3: and the wasted the wasted the w
  2/3: and the bear the bear the bear
  3/3: and the sparrow what had not go
- **Initial character: u**
  1/3: ut the was some to the was some
  2/3: ut the fire and said: 'I will s
  3/3: ut the straw and said, 'I will
- **Initial character: v**
  1/3: ve her her have her her have he
  2/3: ver the searly and the searly a
  3/3: ve the stores and said, 'I will
- **Initial character: g**
  1/3: g to the stoor and said the sto
  2/3: g to the wood with the wood wit
  3/3: g the fire and said, 'I will go

From the sequences, it becomes clear that the model is quite vulnerable for getting stuck in loops of words that is has seen with high frequency. Especially prepositions, conjunctions and articles like "in, the, and" are repeated frequently. As the training progresses, longer and more complex dependencies that are frequent in the data become apparent. Notice that the phrase "and said, 'I will" occurs in 3 out of 5 sets of sequences. When we increase the length of generated sequences, we generally observe many repetitions of the same few phrases. This is an artifact of using the argmax to select the next character, because each character will have a unique successor. This makes it easy to get stuck in a loop containing the same characters.

(c) When the temperature is large, a very large part of the probability density will be assigned to the character with the largest logit, due to the exponential. In other words, the softmax becomes increasingly more like the argmax that was used

previously to generate characters. When the temperature is very low, the distribution flattens, and becomes the uniform distribution in the limit of $\tau \to 0$.

Next, sentences were generated for $\tau \in \{0.5, 1.0, 2.0\}$ using the previous model that was trained to 0.65 accuracy:

- $\boldsymbol{\tau = 0.5}$: g. Yefuitly; if thought now-Devildres-bacly warm muSif; 'Blick Stying yon ire! Kul_Czeby! Is nor pair!' 'Heive spife rrau; but flight a prettil, and te
- $\boldsymbol{\tau = 1.0}$: great men in his people they did not read the ball, he got into the golden cushied. The road behindful of pure streamed a white doves danced with you,
- $\boldsymbol{\tau = 2.0}$: gave his wife sitting and have something and said, 'I will go to the princesses on the castle given her hand, and he got on and the son begged there wa

For $\tau = 0.5$, the generated words are nonsensical most of the time. Clearly, the distribution is too flat, giving too much probability mass to characters that break the correctness of words. For $\tau = 1.0$, the spelling is correct, but the generated sentences lack syntactic structure. Finally, when $\tau = 2.0$, some syntactic structure can be observed. The span over which this structure is present is small however, which can be attributed to the sequence length that is used. Notice however that the sentences contain few repetitions of phrases, which we observed frequently when using greedy sampling. For generating longer sentences, using the temperature-softmax is clearly the better sampling approach.

3 Graph Neural Networks

3.1 GCN Forward Layer

Question 3.1

(a) The structural information in the graph data is exploited by propagating data only between graph nodes that are connected by an edge. This operation is accomplished by multiplication with the self-connected adjacency matrix, as it assigns to each node the sum of the messages of its graph neighbours (including itself). Note that by messages, I mean the feature vectors multiplied with the weight matrix. For this reason, the process can be seen as message passing.

(b) The self-connected adjacency matrix becomes intractably large and therefore memory-inefficient when the graph consists of many nodes. This problem can be overcome by using a more efficient data structure for storing the graph structure, such as a sparse matrix or edge list.

Question 3.2

(a) Let each node's index in the matrix correspond to its label's position in the alphabet. Then,

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

(b) It will take 4 updates, since the shortest path from C to E has 4 edges. Note that the shortest path is not unique.

3.2 Graph Attention Networks

Question 3.3

In order to perform attention, we have to replaces the average that is taken over the incoming messages with a weighted average, where the weights are determined by comparison of input queries and input keys. This would modify the forward function as follows:

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

where α_{ij} is the attention weight corresponding to node j, when attending over neighbours of node i. Notice that in general, this is not a symmetric quantity, i.e., $\alpha_{ij} \neq \alpha_{ji}$

3.3 Applications of GNNs

Question 3.4

- **Text classification:** Graph convolutional networks can be used to classify text documents, by learning a graph that has words and documents as nodes, and is built from word occurence and co-occurence. According to the authors the model has state-of-the-art performance. (`https://ojs.aaai.org//index.php/AAAI/article/view/4725`)
- **Few-shot image classification**: Graph convolutional networks can be used to increase the performance of image classification problems that have classes for which a very small number of examples is available. This problem can be addressed by using meta-learning, a framework that uses collections of examples and corresponding label similarity measures instead of example-label pairs. By learning graph representations of images in the collections and corresponding similarity measures, state-of-the-art performance is attained, with added benefits with respect to other meta-learning approaches. (`https://arxiv.org/pdf/1711.04043.pdf`)

3.4 Comparing and Combining GNNs and RNNs

Question 3.5

One would most likely choose RNNs when the data has an intrinsic sequential structure, i.e., when the data could be represented as an directed acyclic graph. Usually, this is data that has a clear linear structure, often together with a temporal component (time series data, text). A GNN would be chosen when the data has more complex dependencies than the situation described above. These dependencies can take on many forms, such as spatial (molecules, materials, images), semantic (knowledge graphs), social (social networks) and others.