Assignment 2: Recurrent Neural Networks and Graph Neural Networks

Steven van de Graaf
12234036
steven.vandegraaf@student.uva.nl

All code and output for this assignment are available here.

1 Vanilla RNN versus LSTM

1.1 Toy Problem: Palindrome Numbers

1.2 Vanilla RNN in PyTorch

Question 1.1

For this gradient, we consider:

$$\frac{\partial \mathcal{L}^{(T)}}{\partial \mathbf{W}_{ph}} = \frac{\partial \mathcal{L}^{(T)}}{\partial \hat{\mathbf{y}}^{(T)}} \frac{\partial \hat{\mathbf{y}}^{(T)}}{\partial \mathbf{p}^{(T)}} \frac{\partial \mathbf{p}^{(T)}}{\partial \mathbf{W}_{ph}}, \tag{1}$$

where

$$\mathcal{L}^{(T)} = -\sum_{k=1}^{K} \mathbf{y}_k^{(T)} \log \hat{\mathbf{y}}_k^{(T)}, \tag{2}$$

$$\hat{\mathbf{y}}^{(T)} = \mathrm{softmax}(\mathbf{p}^{(T)}), \tag{3}$$

$$\text{and } \mathbf{p}^{(T)} = \mathbf{W}_{ph}\, \mathbf{h}^{(T)} + \mathbf{b}_p \tag{4}$$

As such, let's consider this gradient in parts, starting with:

$$\frac{\partial \mathcal{L}^{(T)}}{\partial \hat{\mathbf{y}}^{(T)}} = \frac{\partial}{\partial \hat{\mathbf{y}}^{(T)}} \left(-\sum_{k=1}^{K} \mathbf{y}_k^{(T)} \log \hat{\mathbf{y}}_k^{(T)} \right), \tag{5}$$

where

$$\left(\frac{\partial}{\partial \hat{\mathbf{y}}^{(T)}} \left(-\sum_{k=1}^{K} \mathbf{y}_k^{(T)} \log \hat{\mathbf{y}}_k^{(T)} \right) \right)_j = \frac{\partial}{\partial \hat{\mathbf{y}}_j^{(T)}} \left(-\sum_{k=1}^{K} \mathbf{y}_k^{(T)} \log \hat{\mathbf{y}}_k^{(T)} \right) = -\frac{\mathbf{y}_j^{(T)}}{\hat{\mathbf{y}}_j^{(T)}}, \tag{6}$$

as for all $j \neq k$, $\hat{\mathbf{y}}_k^{(T)}$ does not depend on $\hat{\mathbf{y}}_j^{(T)}$, and, as such, its derivative with respect to $\hat{\mathbf{y}}_j^{(T)}$ will be zero. As such, this part of the gradient is given by:

$$\frac{\partial \mathcal{L}^{(T)}}{\partial \hat{\mathbf{y}}^{(T)}} = \left[-\frac{\mathbf{y}_j^{(T)}}{\hat{\mathbf{y}}_j^{(T)}} \right]_{j=1}^{K} = \begin{bmatrix} -\frac{\mathbf{y}_1^{(T)}}{\hat{\mathbf{y}}_1^{(T)}} & -\frac{\mathbf{y}_2^{(T)}}{\hat{\mathbf{y}}_2^{(T)}} & \cdots & -\frac{\mathbf{y}_K^{(T)}}{\hat{\mathbf{y}}_K^{(T)}} \end{bmatrix} \in \mathbb{R}^{1 \times K} \tag{7}$$

Submitted to the Spring 2019 Amsterdam Deep Learning course

Let's now consider the following part of the gradient:

$$\frac{\partial \hat{\mathbf{y}}^{(T)}}{\partial \mathbf{p}^{(T)}} = \frac{\partial}{\partial \mathbf{p}^{(T)}} \left(\text{softmax}(\mathbf{p}^{(T)}) \right), \tag{8}$$

where

$$\begin{aligned} \left(\frac{\partial \hat{\mathbf{y}}^{(T)}}{\partial \mathbf{p}^{(T)}} \right)_{i,j} &= \frac{\partial \hat{\mathbf{y}}_i^{(T)}}{\partial \mathbf{p}_j^{(T)}} = \frac{\partial}{\partial \mathbf{p}_j^{(T)}} \left(\text{softmax}(\mathbf{p}_i^{(T)}) \right) \\ &= \frac{\partial}{\partial \mathbf{p}_j^{(T)}} \left(\frac{\exp \mathbf{p}_i^{(T)}}{\sum_{k=1}^{K} \exp \mathbf{p}_k^{(T)}} \right) \end{aligned} \tag{9}$$

For $i = j$, we get:

$$\left(\frac{\partial \hat{\mathbf{y}}^{(T)}}{\partial \mathbf{p}^{(T)}} \right)_{i,j} = \hat{\mathbf{y}}_i^{(T)} \cdot \left(1 - \hat{\mathbf{y}}_j^{(T)} \right) \tag{10}$$

For $i \neq j$, we get:

$$\left(\frac{\partial \hat{\mathbf{y}}^{(T)}}{\partial \mathbf{p}^{(T)}} \right)_{i,j} = - \hat{\mathbf{y}}_i^{(T)} \cdot \hat{\mathbf{y}}_j^{(T)} \tag{11}$$

As such, this part of the gradient is given by:

$$\begin{aligned} \frac{\partial \hat{\mathbf{y}}^{(T)}}{\partial \mathbf{p}^{(T)}} &= \left[\hat{\mathbf{y}}_i^{(T)} \cdot \left(\mathbb{1}(i = j) - \hat{\mathbf{y}}_j^{(T)} \right) \right]_{i,j=1}^{K,K} \\ &= \begin{bmatrix} \hat{\mathbf{y}}_1^{(T)} \cdot \left(1 - \hat{\mathbf{y}}_1^{(T)} \right) & \hat{\mathbf{y}}_1^{(T)} \cdot \left(- \hat{\mathbf{y}}_2^{(T)} \right) & \cdots & \hat{\mathbf{y}}_1^{(T)} \cdot \left(- \hat{\mathbf{y}}_K^{(T)} \right) \\ \hat{\mathbf{y}}_2^{(T)} \cdot \left(- \hat{\mathbf{y}}_1^{(T)} \right) & \hat{\mathbf{y}}_2^{(T)} \cdot \left(1 - \hat{\mathbf{y}}_2^{(T)} \right) & \cdots & \hat{\mathbf{y}}_2^{(T)} \cdot \left(- \hat{\mathbf{y}}_K^{(T)} \right) \\ \vdots & \vdots & \ddots & \cdots \\ \hat{\mathbf{y}}_K^{(T)} \cdot \left(- \hat{\mathbf{y}}_1^{(T)} \right) & \hat{\mathbf{y}}_K^{(T)} \cdot \left(- \hat{\mathbf{y}}_2^{(T)} \right) & \cdots & \hat{\mathbf{y}}_K^{(T)} \cdot \left(1 - \hat{\mathbf{y}}_K^{(T)} \right) \end{bmatrix} \in \mathbb{R}^{K \times K}. \end{aligned} \tag{12}$$

Finally, let's consider the third and last part of the gradient:

$$\frac{\partial \mathbf{p}^{(T)}}{\partial \mathbf{W}_{ph}} = \begin{bmatrix} \frac{\partial \mathbf{p}_1^{(T)}}{\partial \mathbf{W}_{ph}} \\ \frac{\partial \mathbf{p}_2^{(T)}}{\partial \mathbf{W}_{ph}} \\ \vdots \\ \frac{\partial \mathbf{p}_K^{(T)}}{\partial \mathbf{W}_{ph}} \end{bmatrix} \in \mathbb{R}^{K \times 1}, \tag{13}$$

where

$$\frac{\partial \mathbf{p}_k^{(T)}}{\partial \mathbf{W}_{ph}} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{h}_1^{(T)} & \mathbf{h}_2^{(T)} & \cdots & \mathbf{h}_K^{(T)} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ (\mathbf{h}^{(T)})^T \\ \vdots \\ 0 \end{bmatrix} \tag{14}$$

As such, combining Eqns. 7 & 12 yields:

$$
\begin{aligned}
\frac{\partial \mathcal{L}^{(T)}}{\partial \hat{\mathbf{y}}^{(T)}} \frac{\partial \hat{\mathbf{y}}^{(T)}}{\partial \mathbf{p}^{(T)}} &= \begin{bmatrix} -\frac{\mathbf{y}_1^{(T)}}{\hat{\mathbf{y}}_1^{(T)}} \\ -\frac{\mathbf{y}_2^{(T)}}{\hat{\mathbf{y}}_2^{(T)}} \\ \cdots \\ -\frac{\mathbf{y}_K^{(T)}}{\hat{\mathbf{y}}_K^{(T)}} \end{bmatrix}^T \cdot \begin{bmatrix} \hat{\mathbf{y}}_1^{(T)} \cdot \left(1 - \hat{\mathbf{y}}_1^{(T)}\right) & \hat{\mathbf{y}}_1^{(T)} \cdot \left(-\hat{\mathbf{y}}_2^{(T)}\right) & \cdots & \hat{\mathbf{y}}_1^{(T)} \cdot \left(-\hat{\mathbf{y}}_K^{(T)}\right) \\ \hat{\mathbf{y}}_2^{(T)} \cdot \left(-\hat{\mathbf{y}}_1^{(T)}\right) & \hat{\mathbf{y}}_2^{(T)} \cdot \left(1 - \hat{\mathbf{y}}_2^{(T)}\right) & \cdots & \hat{\mathbf{y}}_2^{(T)} \cdot \left(-\hat{\mathbf{y}}_K^{(T)}\right) \\ \vdots & \vdots & \ddots & \cdots \\ \hat{\mathbf{y}}_K^{(T)} \cdot \left(-\hat{\mathbf{y}}_1^{(T)}\right) & \hat{\mathbf{y}}_K^{(T)} \cdot \left(-\hat{\mathbf{y}}_2^{(T)}\right) & \cdots & \hat{\mathbf{y}}_K^{(T)} \cdot \left(1 - \hat{\mathbf{y}}_K^{(T)}\right) \end{bmatrix} \\
&= \begin{bmatrix} -\frac{\mathbf{y}_1^{(T)}}{\hat{\mathbf{y}}_1^{(T)}} \cdot \left(\hat{\mathbf{y}}_1^{(T)} \cdot \left(1 - \hat{\mathbf{y}}_1^{(T)}\right) + \hat{\mathbf{y}}_1^{(T)} \cdot \left(-\hat{\mathbf{y}}_2^{(T)}\right) + \cdots + \hat{\mathbf{y}}_1^{(T)} \cdot \left(-\hat{\mathbf{y}}_K^{(T)}\right)\right) \\ -\frac{\mathbf{y}_2^{(T)}}{\hat{\mathbf{y}}_2^{(T)}} \cdot \left(\hat{\mathbf{y}}_2^{(T)} \cdot \left(-\hat{\mathbf{y}}_1^{(T)}\right) + \hat{\mathbf{y}}_2^{(T)} \cdot \left(1 - \hat{\mathbf{y}}_2^{(T)}\right) + \cdots + \hat{\mathbf{y}}_2^{(T)} \cdot \left(-\hat{\mathbf{y}}_K^{(T)}\right)\right) \\ \vdots \\ -\frac{\mathbf{y}_K^{(T)}}{\hat{\mathbf{y}}_K^{(T)}} \cdot \left(\hat{\mathbf{y}}_K^{(T)} \cdot \left(-\hat{\mathbf{y}}_1^{(T)}\right) + \hat{\mathbf{y}}_K^{(T)} \cdot \left(-\hat{\mathbf{y}}_2^{(T)}\right) + \cdots + \hat{\mathbf{y}}_K^{(T)} \cdot \left(1 - \hat{\mathbf{y}}_K^{(T)}\right)\right) \end{bmatrix}^T \\
&= \begin{bmatrix} -\frac{\mathbf{y}_1^{(T)}}{\hat{\mathbf{y}}_1^{(T)}} \cdot \left(\hat{\mathbf{y}}_1^{(T)} \cdot \left(1 - \hat{\mathbf{y}}_1^{(T)} - \hat{\mathbf{y}}_2^{(T)} - \cdots - \hat{\mathbf{y}}_K^{(T)}\right)\right) \\ -\frac{\mathbf{y}_2^{(T)}}{\hat{\mathbf{y}}_2^{(T)}} \cdot \left(\hat{\mathbf{y}}_2^{(T)} \cdot \left(1 - \hat{\mathbf{y}}_1^{(T)} - \hat{\mathbf{y}}_2^{(T)} - \cdots - \hat{\mathbf{y}}_K^{(T)}\right)\right) \\ \vdots \\ -\frac{\mathbf{y}_K^{(T)}}{\hat{\mathbf{y}}_K^{(T)}} \cdot \left(\hat{\mathbf{y}}_K^{(T)} \cdot \left(1 - \hat{\mathbf{y}}_1^{(T)} - \hat{\mathbf{y}}_2^{(T)} - \cdots - \hat{\mathbf{y}}_K^{(T)}\right)\right) \end{bmatrix}^T \\
&= \begin{bmatrix} -\mathbf{y}_1^{(T)} \cdot \left(1 - \hat{\mathbf{y}}_1^{(T)} - \hat{\mathbf{y}}_2^{(T)} - \cdots - \hat{\mathbf{y}}_K^{(T)}\right) \\ -\mathbf{y}_2^{(T)} \cdot \left(1 - \hat{\mathbf{y}}_1^{(T)} - \hat{\mathbf{y}}_2^{(T)} - \cdots - \hat{\mathbf{y}}_K^{(T)}\right) \\ \vdots \\ -\mathbf{y}_K^{(T)} \cdot \left(1 - \hat{\mathbf{y}}_1^{(T)} - \hat{\mathbf{y}}_2^{(T)} - \cdots - \hat{\mathbf{y}}_K^{(T)}\right) \end{bmatrix}^T \\
&= \begin{bmatrix} -\mathbf{y}_1^{(T)} \cdot \left(1 - \sum_{k=1}^K \hat{\mathbf{y}}_k^{(T)}\right) \\ -\mathbf{y}_2^{(T)} \cdot \left(1 - \sum_{k=1}^K \hat{\mathbf{y}}_k^{(T)}\right) \\ \vdots \\ -\mathbf{y}_K^{(T)} \cdot \left(1 - \sum_{k=1}^K \hat{\mathbf{y}}_k^{(T)}\right) \end{bmatrix}^T \\
&= \begin{bmatrix} \hat{\mathbf{y}}_1^{(T)} - \mathbf{y}_1^{(T)} \\ \hat{\mathbf{y}}_2^{(T)} - \mathbf{y}_2^{(T)} \\ \vdots \\ \hat{\mathbf{y}}_K^{(T)} - \mathbf{y}_K^{(T)} \end{bmatrix}^T = \left(\hat{\mathbf{y}}^{(T)} - \mathbf{y}^{(T)}\right)^T \in \mathbb{R}^{1 \times K}.
\end{aligned} \tag{15}
$$

Now, finally, plugging all of this (Eqns. 15 & 13) into Eqn. 1 yields:

$$
\begin{aligned}
\frac{\partial \mathcal{L}^{(T)}}{\partial \mathbf{W}_{ph}} &= \frac{\partial \mathcal{L}^{(T)}}{\partial \hat{\mathbf{y}}^{(T)}} \frac{\partial \hat{\mathbf{y}}^{(T)}}{\partial \mathbf{p}^{(T)}} \frac{\partial \mathbf{p}^{(T)}}{\partial \mathbf{W}_{ph}} \\
&= \begin{bmatrix} \hat{\mathbf{y}}_1^{(T)} - \mathbf{y}_1^{(T)} \\ \hat{\mathbf{y}}_2^{(T)} - \mathbf{y}_2^{(T)} \\ \vdots \\ \hat{\mathbf{y}}_K^{(T)} - \mathbf{y}_K^{(T)} \end{bmatrix}^T \cdot \begin{bmatrix} \frac{\partial \mathbf{p}_1^{(T)}}{\partial \mathbf{W}_{ph}} \\ \frac{\partial \mathbf{p}_2^{(T)}}{\partial \mathbf{W}_{ph}} \\ \vdots \\ \frac{\partial \mathbf{p}_K^{(T)}}{\partial \mathbf{W}_{ph}} \end{bmatrix}
\end{aligned}
$$

$$
\begin{aligned}
&= \sum_{k=1}^{K} \left(\hat{\mathbf{y}}_k^{(T)} - \mathbf{y}_k^{(T)} \right) \cdot \begin{bmatrix} 0 \\ \vdots \\ (\mathbf{h}^{(T)})^T \\ \vdots \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} \hat{\mathbf{y}}_1^{(T)} - \mathbf{y}_1^{(T)} \\ \hat{\mathbf{y}}_2^{(T)} - \mathbf{y}_2^{(T)} \\ \vdots \\ \hat{\mathbf{y}}_K^{(T)} - \mathbf{y}_K^{(T)} \end{bmatrix} \cdot (\mathbf{h}^{(T)})^T \\
&= \left(\hat{\mathbf{y}}^{(T)} - \mathbf{y}^{(T)} \right) \cdot (\mathbf{h}^{(T)})^T. \qquad (16)
\end{aligned}
$$

For this gradient, we consider:

$$
\frac{\partial \mathcal{L}^{(T)}}{\partial \mathbf{W}_{hh}} = \frac{\partial \mathcal{L}^{(T)}}{\partial \hat{\mathbf{y}}^{(T)}} \frac{\partial \hat{\mathbf{y}}^{(T)}}{\partial \mathbf{p}^{(T)}} \frac{\partial \mathbf{p}^{(T)}}{\partial \mathbf{h}^{(T)}} \frac{\partial \mathbf{h}^{(T)}}{\partial \mathbf{W}_{hh}}, \qquad (17)
$$

where $\frac{\partial \mathcal{L}^{(T)}}{\partial \hat{\mathbf{y}}^{(T)}}$ and $\frac{\partial \hat{\mathbf{y}}^{(T)}}{\partial \mathbf{p}^{(T)}}$ are already known from Eqns. 7 and 12. As such, let's now consider the third part of this gradient:

$$
\frac{\partial \mathbf{p}^{(T)}}{\partial \mathbf{h}^{(T)}} = \mathbf{W}_{ph}\,. \qquad (18)
$$

As such, all we are left with now is the fourth part of the gradient:

$$
\begin{aligned}
\frac{\partial \mathbf{h}^{(T)}}{\partial \mathbf{W}_{hh}} &= \frac{\partial}{\partial \mathbf{W}_{hh}} \left(\tanh \left(\mathbf{W}_{hx}\, \mathbf{x}^{(T)} + \mathbf{W}_{hh}\, \mathbf{h}^{(T-1)} + \mathbf{b}_h \right) \right) \\
&= \left(1 - \tanh^2 \left(\mathbf{W}_{hx}\, \mathbf{x}^{(T)} + \mathbf{W}_{hh}\, \mathbf{h}^{(T-1)} + \mathbf{b}_h \right) \right) \cdot \frac{\partial}{\partial \mathbf{W}_{hh}} \left(\mathbf{W}_{hx}\, \mathbf{x}^{(T)} + \mathbf{W}_{hh}\, \mathbf{h}^{(T-1)} + \mathbf{b}_h \right) \\
&= \left(1 - {\mathbf{h}^{(T)}}^2 \right) \cdot \frac{\partial}{\partial \mathbf{W}_{hh}} \left(\mathbf{W}_{hh}\, \mathbf{h}^{(T-1)} \right) \\
&= \left(1 - {\mathbf{h}^{(T)}}^2 \right) \cdot \left(\left(\frac{\partial}{\partial \mathbf{W}_{hh}} \mathbf{W}_{hh} \right) \mathbf{h}^{(T-1)} + \mathbf{W}_{hh} \left(\frac{\partial}{\partial \mathbf{W}_{hh}} \mathbf{h}^{(T-1)} \right) \right) \\
&= \left(1 - {\mathbf{h}^{(T)}}^2 \right) \cdot \left(\mathbf{h}^{(T-1)} + \mathbf{W}_{hh} \frac{\partial \mathbf{h}^{(T-1)}}{\partial \mathbf{W}_{hh}} \right). \qquad (19)
\end{aligned}
$$

As $\frac{\partial \mathbf{h}^{(T)}}{\partial \mathbf{W}_{hh}}$ depends on $\frac{\partial \mathbf{h}^{(T-1)}}{\partial \mathbf{W}_{hh}}$, and $\frac{\partial \mathbf{h}^{(T-1)}}{\partial \mathbf{W}_{hh}}$ depends on $\frac{\partial \mathbf{h}^{(T-2)}}{\partial \mathbf{W}_{hh}}$, andsoforth, the recursive nature of this gradient becomes quite clear.

As $\frac{\partial \mathcal{L}^{(T)}}{\partial \mathbf{W}_{ph}}$ depends solely on the inputs and hidden states of the current state, whereas $\frac{\partial \mathbf{h}^{(T)}}{\partial \mathbf{W}_{hh}}$ depends on all inputs and hidden states of the current and previous timesteps, the difference in temporal dependence of the two gradients becomes instantly clear.

When training this recurrent network for many timesteps, with the $\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{W}_{hh}}$ gradient a problem that might occur is that the gradient either decreases greatly ("vanishes"), or increases greatly ("explodes"). At a large enough timestep t, if $\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{W}_{hh}}$ is bigger (or smaller) than 1, then the recursive product of gradients might increase sharply (or decrease), such that it explodes (or collapses).

Question 1.2

Please see my `part1/vanilla_rnn.py` and `part1/train.py` for my implementation of the vanilla recurrent network and the optimization procedure.

To answer the in-code question:

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

addresses the previously mentioned issue of possibly exploding gradients by "clipping" them, which forces the gradient values (element-wise) to a predefined maximum, should they exceed this predefined maximum. When the gradient descent optimization method (RMSprop in our case) proposes to take a too large step, this `torch.nn.utils.clip_grad_norm` method reduces the step size.

Question 1.3

In order to investigate the memorization capability of the of the vanilla RNN, several experiments were conducted. In all of these experiments, the following *default* parameters were used:

```
model_type = 'RNN'
input_dim = 1
num_classes = 10
num_hidden = 128
batch_size = 128
learning_rate = 0.001
train_steps = 10000
max_norm = 10.0
```

Furthermore, the weights were initialized using the normalized initialization as introduced in Glorot and Bengio [2010], and the biases were initialized with zeroes.

For every experiment conducted, the `input_length` parameter was varied as follows:

```
input_lengths = [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20],
```

the results of which can be reviewed in Fig. 1 (for Figs. 1a & 1b, the accuracies were plotted for the odd and even palindrome lengths seperately, for clarity).

From Fig. 1, it becomes quite clear that for short palindrome lengths (from $T = 5$ till $T = 13$), the accuracies obtained are (close to) 100%, after which it starts to decrease with increasing palindrome length. What's interesting to note that, for longer palindrome lengths, the vanilla RNN model performs better for *odd* length palindromes, than for *even* length palindromes.

Question 1.4

The benefits of variants of stochastic gradient descent, such as the RMSprop and Adam optimizers and/or the use of momentum, in comparison to vanilla stochastic gradient descent, are such that the optimizer is able to escape from *pathological curvatures* (see Fig. 2) or *plateaus*, for example.

A *pathological curvature* is a region of the parameter space where the optimizer gets "stuck" such that, while it does converge to the (local) minimum, it does so very slowly, by taking a path that is not necessarily the "ideal" path, like in the ravine pictured in Fig. 2.

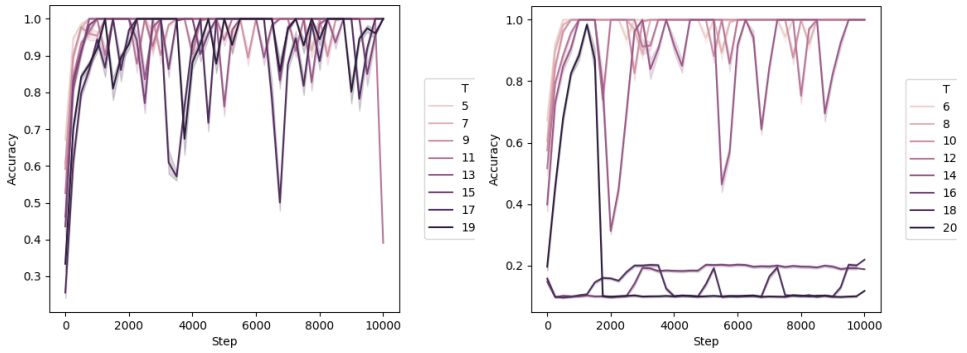
One of the techniques one could use to escape such a *pathological curvature* is called **momentum**, where, at every timestep t, you don't only use the gradient of the current timestep for gradient descent, but also accumulate those of the previous timesteps. In practice, this looks as follows:

$$v_j^{(t+1)} = \alpha \cdot v_j^{(t)} - \eta \cdot G^{(t)} \tag{20}$$

$$w_j^{(t+1)} = v_j^{(t+1)} + w_j^{(t)}, \tag{21}$$

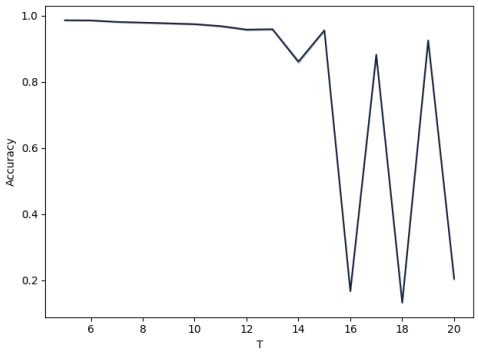
where $v_j^{(t+1)}$ is the weight update of the j^{th} weight $w_j^{(t)}$ at the t^{th} timestep, α is the *coefficient of momentum*, $v_j^{(t)}$ is the retained gradient from the previous timestep, η is the learning rate, and $G^{(t)}$ is the gradient.

By accumulating the gradients from the previous timestep, the optimizer is able to prevent the "zig-zag" behaviour displayed in Fig. 2, as the components of the gradients along the w_1 direction



(a) Accuracy of the vanilla RNN model for various *odd* values of `input_length` over various timesteps

(b) Accuracy of the vanilla RNN model for various *even* values of `input_length` over various timesteps



(c) Accuracy of the vanilla RNN model over various values of `input_length`

Figure 1: Plots of accuracies obtained during training of the vanilla RNN model for various values of `input_length`

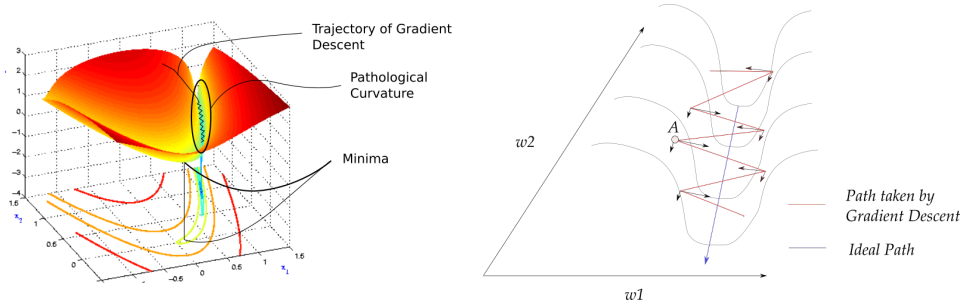


Figure 2: Visual representation of pathological curvature.

Image courtesy of Paperspace

will (roughly) cancel each other out, while reinforcing those components along the w_2 direction, thereby escaping the ravine more quickly.

Another possible way for an optimizer to escape such *pathological curvatures* or *plateaus* is to vary the learning rate, for example starting with a relatively high learning rate (in order to traverse the "bad" parts of the parameter space more quickly in the beginning) and decreasing it over time (allowing it to converge to the (local) optima as the optimizer approaches it).

This is exactly what optimizers like RMSprop and Adam do; they make use of an **adaptive learning rate**. For RMSprop, this looks like:

$$u_j^{(t+1)} = \alpha \cdot u_j^{(t)} + (1-\alpha) \cdot (G_j^{(t)})^2 \tag{22}$$

$$v_j^{(t+1)} = \frac{\eta}{\sqrt{u_j^{(t+1)} + \epsilon}} \cdot G_j^{(t)} \tag{23}$$

$$w_j^{(t+1)} = \eta \cdot v_j^{(t+1)} + w_j^{(t)}, \tag{24}$$

where $u_j^{(t)}$ is the exponential average of squares of gradients of the j^{th} weight $w_j^{(t)}$ at the t^{th} timestep, η is the initial learning rate and $G_j^{(t)}$ the gradient of this $w_j^{(t)}$. As can be seen from Eqn. 23, the learning rate in RMSprop is varied at each timestep by dividing it by $\sqrt{u_j^{(t+1)} + \epsilon}$.

For Adam, both the adaptive learning rate and momentum come together as follows:

$$u_j^{(t+1)} = \beta_1 \cdot u_j^{(t)} + (1-\beta_1) \cdot G_j^{(t+1)} \tag{25}$$

$$s_j^{(t+1)} = \beta_2 \cdot s_j^{(t)} + (1-\beta_2) \cdot (G_j^{(t+1)})^2 \tag{26}$$

$$\hat{u}_j^{(t+1)} = \frac{u_j^{(t+1)}}{1-\beta_1^t} \tag{27}$$

$$\hat{s}_j^{(t+1)} = \frac{s_j^{(t+1)}}{1-\beta_2^t} \tag{28}$$

$$w_j^{(t+1)} = w_j^{(t)} - \eta \cdot \frac{\hat{u}_j^{(t+1)}}{\sqrt{\hat{s}_j^{(t+1)} + \epsilon}}, \tag{29}$$

where $u_j^{(t+1)}$ and $s_j^{(t+1)}$ are the first and second momentum estimates, respectively, and $\hat{u}_j^{(t+1)}$ and $\hat{s}_j^{(t+1)}$ are bias corrections.

1.3 Long-Short Term Network (LSTM) in PyTorch

Question 1.5

(**a**) The *input modulation gate*

$$\mathbf{g}^{(t)} = \tanh\left(\mathbf{W}_{gx}\,\mathbf{x}^{(t)} + \mathbf{W}_{gh}\,\mathbf{h}^{(t-1)} + \mathbf{b}_g\right) \tag{30}$$

computes a "candidate" hidden state, based on the input at the current timestep and the hidden state at the previous timestep. It uses a $\tanh$ non-linearity, which is centered around 0, and therefore ensures that, when added to the old cell state, it does not significantly increase in magnitude.

The *input gate*

$$\mathbf{i}^{(t)} = \sigma(\mathbf{W}_{ix}\,\mathbf{x}^{(t)} + \mathbf{W}_{ih}\,\mathbf{h}^{(t-1)} + \mathbf{b}_i) \tag{31}$$

regulates what information from the *input modulation gate* $\mathbf{g}^{(t)}$ should be added to the cell state. It uses a σ non-linearity, which has a range of $[0, 1]$, thereby allowing certain information to flow (when its value is (near) 1), and certain information to halt (when its value is (near) 0).

The *forget gate*

$$\mathbf{f}^{(t)} = \sigma(\mathbf{W}_{fx}\,\mathbf{x}^{(t)} + \mathbf{W}_{fh}\,\mathbf{h}^{(t-1)} + \mathbf{b}_f) \tag{32}$$

regulates what information of the old cell state should be "forgotten". It uses a σ non-linearity, which has a range of $[0, 1]$, thereby allowing it to "remember" certain information (when its value is (near) 1), and to "forget" certain information (when its value is (near) 0).

The *output gate*

$$\mathbf{o}^{(t)} = \sigma(\mathbf{W}_{ox}\,\mathbf{x}^{(t)} + \mathbf{W}_{oh}\,\mathbf{h}^{(t-1)} + \mathbf{b}_o) \tag{33}$$

regulates what information of the current cell state should be passed along to the hidden state. It uses a σ non-linearity, which has a range of $[0, 1]$, thereby allowing certain information to flow to the output (when its value is (near) 1), and certain information to halt (when its value is (near) 0).

(**b**) We have

$$\mathbf{x} \in \mathbb{R}^{T \times d}, \tag{34}$$

where T is the sequence length and d is the feature dimensionality. As such, for the 4 gates with n hidden units each (which all boast the same linear mapping), we have:

$$\#\text{params}_{LSTM} = 4 \cdot (n \cdot d + n \cdot n + n) \tag{35}$$

Question 1.6

In order to investigate the memorization capability of the of the LSTM, several experiments were conducted. In all of these experiments, the following *default* parameters were used:

```
model_type = 'LSTM'
input_dim = 1
num_classes = 10
num_hidden = 128
batch_size = 128
learning_rate = 0.001
train_steps = 10000
max_norm = 10.0
```

Furthermore, the weights were initialized using the normalized initialization as introduced in Glorot and Bengio [2010], and the biases were initialized with zeroes.

For every experiment conducted, the `input_length` parameter was varied as follows:

```
input_lengths = [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20],
```

the results of which can be reviewed in Fig. 3 (for Figs. 3a & 3b, the accuracies were plotted for the odd and even palindrome lengths seperately, for clarity).

From Fig. 3, it becomes quite clear that the accuracies obtained for short and medium palindrome lengths (from $T = 5$ till $T = 15$), the accuracies obtained are (close to) 100%, after which it starts to decrease with increasing palindrome length. What's interesting to note that, as with the vanilla RNN model, for longer palindrome lengths, the LSTM model performs better for *odd* length palindromes, than for *even* length palindromes.

2 Recurrent Nets as Generative Model

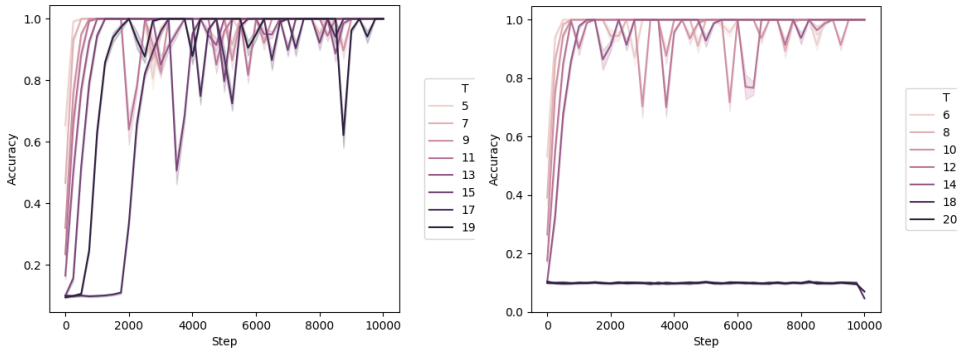
Question 2.1

(**a**) Please see my `part2/model.py` and `part2/train.py` for my implementation of the two-layer LSTM and the optimization procedure. Also see my `part2/utils.py` for a collection of some utility functions.

I trained the model on three different books (of different sizes and subjects), namely:

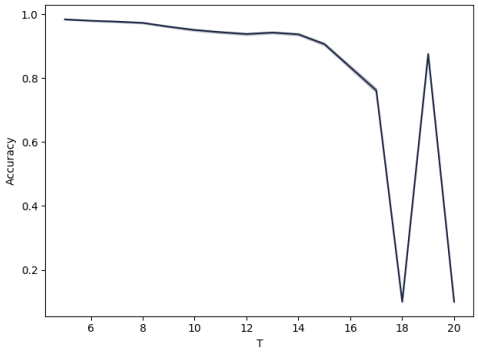
i The Communist Manifesto (15k words)
ii The Adventures of Sherlock Holmes (108k words)
iii The King James Bible (824k words)

All models were trained for 150k steps.



(a) Accuracy of the LSTM model for various *odd* values of `input_length` over various timesteps

(b) Accuracy of the LSTM model for various *even* values of `input_length` over various timesteps



(c) Accuracy of the LSTM model over various values of `input_length`

Figure 3: Plots of accuracies obtained during training of the LSTM model for various values of `input_length`

Step	**The Communist Manifesto**	**The Adventures of Sherlock Holmes**	**The King James Bible**
0	LLLLLLLLLLLLLLLLLLLLLLLLLLLLLL	ldccccccccccccccccccccccccccc	Nfffffffffffffffffffffffffff
2000	0 the proletariat and a politi	, and the station and the stat	. 1:12 And the LORD of the LO
4000	You must complete into the pro	Now, and the street to the str	Now the LORD said unto the LOR
6000	f the proletariat all associat	(81. I have a considerable th	$11:10 And the children of Isr
8000	Has not Christianity in propor	$ "I have a little state and	Now the son of Jerusalem the s
10000	@ DISTRIBUTOR UNDER THIS AGREE	& I have been the corner of th	the LORD hath sent unto the LO

Table 1: Table of generated sequence of length $T = 30$, using temperature $Temp = 0$ (greedy sampling) at evenly spaced intervals during training

(**b**) For all three books, you will find generated text sequences of length $T = 30$, using temperature $Temp = 0$ (greedy sampling) at evenly spaced intervals during training in table 1.

As can be judged from table 1, the two-layer LSTM is very well able to pick up on the particularities of a body of text. This is extremely evident with the model trained on the King James Bible, for example, with the biblical numbering, full-capital LORDs, etc. What's also interesting to note is that, as the Communist Manifesto is so short, the sequence generated at the 10000^{th} timestep looks quite a bit like the copyright notice at the front of the book, which is not something we see for the longer bodies of text.

Initially, the generated sequences boast a lot of repetition, both in terms of repetition of characters, but also of words. As the training process evolves, the sentences become less repetitive, and, in fact, more coherent and grammatical.

(**c**) For all three books, you'll find generated text sequences of length $T = 30$, using various temperature $Temp$ values at evenly spaced intervals during training in table 1

Temp	Step	The Communist Manifesto	The Adventures of Sherlock Holmes	The King James Bible
0.5	0	5fvtXQG:;[fiQLrJ8e-x]1y-j&kGuJ	k-l1'ââ@3OFU[YrL2CLMoesuâ2év8	1jSE81Yr;RqU9.W,kDkVlKtrI[b(,z
	2000	: and of the status and all de	-was friend, and that I have a	forth of the wall the prichild
	4000	ment to manabry, and to introd	be a long returned in the stry	Jesus sent the LORD thy spoken
	6000	IV. POSITION OF THE COMMU	#State which had been deep to	Moses the camp of the congrega
	8000	's agreement for free dis	very stranger, and of the stat	ORD of a sheep of the LORD is
	10000	MMUNIST MANIFESTO *** Tran	Holmes, he had left the table	the commandment of the peopl
1.0	0	)"XI[jE0AgYi7'w@lG#c&JFv MH[xi	/J3d]U%0C-59jg]vmfNy;1vP'(rE2x	1Vu.;zl2 xyoZr2Y(]#886[me-e!#m
	2000	ghties and discosation, and po	resce, all I am verown of ence	@beshing, they thouse noor to
	4000	n the old freedom into the pro	But yes, which some which cuzr	Y of vept; and when the peoft
	6000	0S. "UNTHIAL SACRAGES EVEN IF	he stop and been down," said t	oughful many discuples it city
	8000	livalised to use and self-cons	ver," side's talk he along me	s of Zion. 24:10 And his grea
	10000	f France into to give the grea	& sude 100 pounds the cords whil	s says ly forest thousand, 5:4
2.0	0	xWd!&-wzC4f'IYJ1SJgQpS ;gX5v:F	*l0EEd;(èéSIQ zry,L'oBR6 l6hW	Wfb1F.D!jm@pQe#$z 1;@z:n-E0vY
	2000	)-compee in Mezed of!ah" mevan	$XW, lk, be Mustordsae," .6 E	#2 Tilmidoent amsssl ridmchs o
	4000	7E,CSagetity Vovelopil't; Ipli	XI, mekerqy's, thr enor Giin,,	; Balock of Danl; asY thrinopt
	6000	gglisheves no. money: or, (as	;. "PIt repate: Hopmurwe Rin	@cyr:) labody up said aasuzea'
	8000	31.0. .t, Nepposus? 82. A	l céalitor,-baciClamow"s habed	:33 Ticcapbs? or, intiph. 8:4
	10000	7, Icadie weapold the ground a	& éou? Gook, squiek," said I7 no	(lrightþir, saw; and judg's im

Table 2: Table of generated sequence of length $T = 30$, using various temperature $Temp$ values at evenly spaced intervals during training

One thing that becomes instantly clear from table 2, is that pretty much all sequences generated using temperature $Temp = 2.0$ are gibberish at best. In no way, shape or form are those sequences coherent, nor grammatically correct. Personally, I find the sequences generated using temperature $T = 0.5$ to be of the highest quality: They're quite coherent and grammatically correct, and they lack repetition. In general, however, it becomes clear that, with increasing temperature, the originality (brand new sequences) does increase, but at a possible cost of coherence and grammatical correctness.

3 Graph Neural Networks

3.1 GCN Forward Layer

Question 3.1

(**a**) For this question, we consider the following layer:

$$H^{(l+1)} = \sigma\Big(\hat{A}H^{(l)}W^{(l)}\Big), \tag{36}$$

where matrix $\hat{A}$ encodes the connections between nodes (how "close" they are to each other), matrix $H^{(l)}$ is the feature representation of a node, and $W^{(l)}$ is a weight matrix that is shared by all nodes.

This layer exploits the structural information in the graph data as follows: It updates the feature representation $H^{(l)}$ of a node with a weighted average of the feature representations of its neighbours (depending on how "close" they are to each other, which is encoded in matrix $\hat{A}$). As such, essentially, Eqn. 14 acts as a *message passing* function.

(**b**) One would need to stack at most 3 GCN layers to propagate to every node's embedding the information (features) from nodes 3 hops away in the graph. As in a single GCN layer, the

information of a node is only sent to its direct neighbours (propagated over 1 hop), in order to bridge 3 hops, one would need 3 such GCN layers.

3.2 Applications of GNNs

Question 3.2

Many real-world applications exist in which GNNs could be applied. A few of them are listed below (taken from Wu et al. [2019]):

- **Computer vision**: Johnson et al. [2018]
- **Recommender systems**: Ying et al. [2018]
- **Chemistry**: De Cao and Kipf [2018]
- **Social networks**: Qiu et al. [2018]

3.3 Comparing and Combining GNNs and RNNs

Question 3.3

(**a**) RNN-based models assume sequential data, as in, that the data at hand can be ordered along some axis. An obvious example would be timeseries.

GNNs assume that the data at hand boasts some graph structure (i.e. involving nodes and edges), such as networks and trees. In fact, sequential data could be considered a special type of graph, which is directed, and chain-like (i.e. every node only has a *previous* and *next* node it is connected to).

(**b**) GNNs and RNNs could be combined for graph data which also has a temporal/sequential aspect. An obvious example of this would be natural language, which has graph-like properties (dependency trees, for example) and sequential properties (words are ordered into a sentence, and sentences are ordered into documents, etc.). Another example would be dynamic graphs Ma et al. [2018], such as social networks, which change (evolve) over time.

References

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019.

Justin Johnson, Agrim Gupta, and Li Fei-Fei. Image generation from scene graphs. *CoRR*, abs/1804.01622, 2018. URL `http://arxiv.org/abs/1804.01622`.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. *CoRR*, abs/1806.01973, 2018. URL `http://arxiv.org/abs/1806.01973`.

Nicola De Cao and Thomas Kipf. MolGAN: An implicit generative model for small molecular graphs. *ICML 2018 workshop on Theoretical Foundations and Applications of Deep Generative Models*, 2018.

Jiezhong Qiu, Jian Tang, Hao Ma, Yuxiao Dong, Kuansan Wang, and Jie Tang. Deepinf: Social influence prediction with deep learning. *CoRR*, abs/1807.05560, 2018. URL `http://arxiv.org/abs/1807.05560`.

Yao Ma, Ziyi Guo, Zhaochun Ren, Eric Zhao, Jiliang Tang, and Dawei Yin. Dynamic graph neural networks. *arXiv preprint arXiv:1810.10627*, 2018.