Assignment 2
Recurrent Neural Networks and Graph Neural Networks

Francesco Dal Canton
f.dal.canton@student.uva.nl
12404225

1 Vanilla RNN versus LSTM

1.1 Toy Problem: Palindrome Numbers

1.2 Vanilla RNN in PyTorch

1.1 Throughout this assignment I use the notation x_T in place of $x^{(T)}$ used in the assignment text. I also make the choice to ignore the $tanh$ function, which would only appear as an additional ∂-term in each application of the chain rule.

That said, we can easily write the expression for the gradient of the loss w.r.t. the output weights as follows.

$$\begin{aligned}\frac{\partial \mathcal{L}_T}{\partial W_{ph}} &= \frac{\partial \mathcal{L}_T}{\partial \hat{y}_T}\frac{\partial \hat{y}_T}{\partial p_T}\frac{\partial p_T}{\partial W_{ph}} \\ &= \frac{\partial \mathcal{L}_T}{\partial \hat{y}_T}\frac{\partial \hat{y}_T}{\partial p_T}h_T^T\end{aligned}$$

Defining the gradient w.r.t the hidden weights is trickier. We can start by writing out the standard chain rule formulation, which we can compress by omitting parts that we've defined in the previous assignment (i.e. cross-entropy, softmax, etc.).

$$\begin{aligned}\frac{\partial \mathcal{L}_T}{\partial W_{hh}} &= \frac{\partial \mathcal{L}_T}{\partial \hat{y}_T}\frac{\partial \hat{y}_T}{\partial p_T}\frac{\partial p_T}{\partial h_T}\frac{\partial h_T}{\partial W_{hh}} \\ &= \frac{\partial \mathcal{L}_T}{\partial h_T}\frac{\partial h_T}{\partial W_{hh}}\end{aligned}$$

We notice that any h_t depends recursively on h_{t-1} which itself depends on W_{hh}. We can then define the gradient of the last hidden state w.r.t. its weights, and develop it using the product rule.

$$\begin{aligned}\frac{\partial h_T}{\partial W_{hh}} &= \frac{\partial \left[W_{hx}x_T + W_{hh}h_{T-1} + b_h\right]}{\partial W_{hh}} \\ &= \frac{\partial \left[W_{hh}h_{T-1}\right]}{\partial W_{hh}} \\ &= \frac{\partial W_{hh}}{\partial W_{hh}}h_{T-1} + W_{hh}\frac{\partial h_{T-1}}{\partial W_{hh}} \\ &= h_{T-1} + W_{hh}\frac{\partial h_{T-1}}{\partial W_{hh}}\end{aligned}$$

Where, at the end of the recursive chain, we reach the gradient of h_0 w.r.t. the hidden weights, which is 0. With this knowledge, we can expand the chain in order to compress it again.

$$\begin{aligned}\frac{\partial h_T}{\partial W_{hh}} &= h_{T-1} + W_{hh}\left(h_{T-2} + W_{hh}\left(h_{T-3} + W_{hh}\left(\dots\right)\right)\right) \\ &= h_{T-1} + W_{hh}h_{T-2} + W_{hh}^2 h_{T-3} + \cdots + W_{hh}^{T-2}h_{T-(T-1)} \\ &= \sum_{i=1}^{T-1} W_{hh}^{i-1} h_{T-i}\end{aligned}$$

So that we can express the original gradient as:

$$\frac{\partial \mathcal{L}_T}{\partial W_{hh}} = \frac{\partial \mathcal{L}_T}{\partial h_T} \sum_{i=1}^{T-1} W_{hh}^{i-1} h_{T-i}$$

The most striking difference between the gradient of the loss w.r.t. the output weights and the one w.r.t. the hidden weights is that the latter involves a temporal dependency that extends all the way to the first time step of the cell's operation. In other words, after feeding the RNN a sequence of n inputs, in order to perform backpropagation we need to multiply the gradient by the hidden weight matrix a large number of times, which grows exponentially with the number of time steps.

This fact causes the well known problem of vanishing or exploding gradients. When performing an update over a large number of timesteps, the gradient that should pertain to the gradients further back in time either reduces to 0 or it increases to infinity, depending on whether the hidden weights are small or large. This makes it very hard to train such networks for long sequences, since those longer time dependencies can't be learned.

1.2

1.3

1.4

1.3 Long-Short Term Network (LSTM) in PyTorch

1.5 (a)

(b)

1.6

2 Recurrent Nets as Generative Model

2.1 (a)

(b)

(c)

2.2

3 Graph Neural Networks

3.1 GCN Forward Layer

3.1 (a)

(b)

3.2 Applications of GNNs

3.2

3.3 Comparing and Combining GNNs and RNNs

3.3 (a)

(b)