DL: Assignment 2

Jonathan Mitnik
MSc Artificial Intelligence
University of Amsterdam
jonathan@student.uva.nl

Abstract

In this report, the main focus is to go through the assignment

1 Recurrent Neural Networks

1.1

1.1.a

The gradient can be calculated as follows:

$$\frac{\delta L^{(t)}}{\delta \mathbf{W}_{ph}} : \boxed{\begin{aligned}\frac{\delta L^{(t)}}{\delta \mathbf{W}_{ph}} &= \frac{\delta L^{(t)}}{\delta \hat{\mathbf{y}}^{(t)}} \frac{\delta \hat{\mathbf{y}}^{(t)}}{\delta \mathbf{p}^{(t)}} \frac{\delta \mathbf{p}^{(t)}}{\delta \mathbf{W}_{ph}} \\ &= \frac{\delta L^{(t)}}{\delta \hat{\mathbf{y}}^{(t)}} \frac{\delta \hat{\mathbf{y}}^{(t)}}{\delta \mathbf{p}^{(t)}} \frac{\delta}{\delta \mathbf{W}_{ph}} \mathbf{W}_{ph} * h^{(t)} + b_p \quad = \frac{\delta L^{(t)}}{\delta \hat{\mathbf{y}}^{(t)}} \frac{\delta \hat{\mathbf{y}}^{(t)}}{\delta \mathbf{p}^{(t)}} h^{(t)}\end{aligned}}$$

1.1.b

For the next gradient, we need to keep in mind the temporal and recursive nature. As such, we can define it as: $\frac{\delta L^{(t)}}{\delta \mathbf{W}_{hh}}$:

$$\boxed{\begin{aligned}\frac{\delta L^{(t)}}{\delta \mathbf{W}_{hh}} &= \frac{\delta L^{(t)}}{\delta \hat{\mathbf{y}}^{(t)}} \frac{\delta \hat{\mathbf{y}}^{(t)}}{\delta \mathbf{p}^{(t)}} \frac{\delta \mathbf{p}^{(t)}}{\delta \mathbf{h}^{(t)}} \\ &= \sum_{i=0}^{T=t} \frac{\delta L^{(t)}}{\delta \hat{\mathbf{y}}^{(t)}} \frac{\delta \hat{\mathbf{y}}^{(t)}}{\delta \mathbf{p}^{(t)}} \frac{\delta \mathbf{p}^{(t)}}{\delta \mathbf{h}^{(t)}} \frac{\delta \mathbf{h}^{(t)}}{\delta \mathbf{h}_i} \frac{\delta \mathbf{h}^{(i)}}{\delta \mathbf{W}_{hh}} \\ &= \sum_{i=0}^{T=t} \frac{\delta L^{(t)}}{\delta \hat{\mathbf{y}}^{(t)}} \frac{\delta \hat{\mathbf{y}}^{(t)}}{\delta \mathbf{p}^{(t)}} \frac{\delta \mathbf{p}^{(t)}}{\delta \mathbf{h}^{(t)}} \prod_{j=i+1}^{T} \Big(\frac{\delta \mathbf{h}^{(j)}}{\delta \mathbf{h}_{j-1}} \Big) \frac{\delta \mathbf{h}^{(i)}}{\delta \mathbf{W}_{hh}}\end{aligned}}$$

1.1.c

There is a very clear distinction between the former formula and the latter: the derivative of the latter does not depend on the previous time-steps. Essentially, the matrix mapping the output only depends on the last hidden state. When you compare this to the recursive nature of $\frac{\delta L^{(t)}}{\delta \mathbf{W}_{hh}}$, one important practical difference becomes clear: how "far" the gradient has to traverse to update the weights. For

W_{ph}, the gradient depends only on the latest hidden state, which means that the gradient and updates generally are stronger. With W_{hh}, however, the jacobian product intuitievly either "explodes" the gradient, or it weakens it, "vanishing" the gradient in the process the more time-steps we go back, due to its recursive nature.

1.2

1.2.a

Gates, go as follow:

1. Input modulation gate: this gate is responsible for actually calculating new values that are going to be eventually added to the cell-state. These values are modulated further with tanh, as to ensure that gradients remain stable by not exceeding 1 / -1 (as tanh does).
2. Input gate: this gate acts as "differentiable" mask for the input modulation gate. It decides how much of the input modulation gate will be added on top of the current cell state. The sigmoid allows this "mask-like" functionality to be applied, ranging between 0 and 1.
3. Forget gate: this gate is responsible for deciding how much of the previous cell state still remains (based on the current input and hidden state). It does so, again, by using sigmoids' range of 0 and 1 to decide whether to maintain nothing (0) or all (1).
4. Output gate: this gate uses another "sigmoid-mask" to decide how much of the cell-state is used as hidden-state, possibly more short-term oriented. Before that, the cell-state is put through tanh to ensure that the value after its previous update is more stable again, max 1.

1.2.b

The number of parameters are $4 * \Big((N_{hidden} * N_{hidden}) + (N_{hidden} * N_{input}) + N_{hidden} \Big) + (N_{hidden} * N_{output})$.

2 Recurrent Nets as Generative Model

test

3 Graph Neural Networks

test