Homework 4

Deadline: Thursday, March 14, at 11:59pm.

Submission: You must submit your solutions as a PDF through MarkUs. You can produce the file however you like (e.g. LaTeX, Microsoft Word, scanner) as long as it is readable.

Late Submission: MarkUs will remain open until 3 days after the deadline, after which no late submissions will be accepted. The late penalty is 10% per day, rounded up.

Weekly homeworks are individual work. See the Course Information handout[1] for detailed policies.

1. **LSTM Gradient [4pts]** Here, you'll derive the Backprop Through Time equations for the univariate version of the Long-Term Short-Term Memory (LSTM) architecture.

   For reference, here are the computations it performs:

$$\begin{aligned}
i^{(t)} &= \sigma(w_{ix}x^{(t)} + w_{ih}h^{(t-1)}) \\
f^{(t)} &= \sigma(w_{fx}x^{(t)} + w_{fh}h^{(t-1)}) \\
o^{(t)} &= \sigma(w_{ox}x^{(t)} + w_{oh}h^{(t-1)}) \\
g^{(t)} &= \tanh(w_{gx}x^{(t)} + w_{gh}h^{(t-1)}) \\
c^{(t)} &= f^{(t)}c^{(t-1)} + i^{(t)}g^{(t)} \\
h^{(t)} &= o^{(t)}\tanh(c^{(t)})
\end{aligned}$$

   (a) **[3pts]** Derive the Backprop Through Time equations for the activations and the gates:

$$\begin{aligned}
\overline{h^{(t)}} &= \\
\overline{c^{(t)}} &= \\
\overline{g^{(t)}} &= \\
\overline{o^{(t)}} &= \\
\overline{f^{(t)}} &= \\
\overline{i^{(t)}} &=
\end{aligned}$$

   You don't need to vectorize anything or factor out any repeated subexpressions.

   (b) **[1pt]** Derive the BPTT equation for the weight w_{ix}:

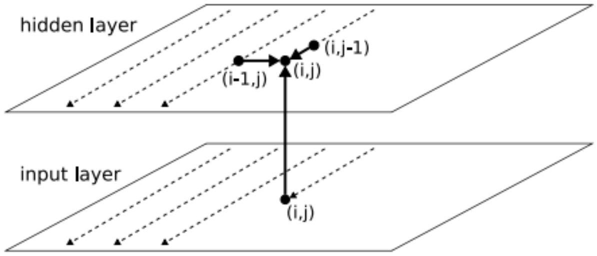
$$\overline{w_{ix}} =$$

   (The other weight matrices are basically the same, so we won't make you write those out.)

   (c) **[optional, no points]** Based on your answers above, explain why the gradient doesn't explode if the values of the forget gates are very close to 1 and the values of the input and output gates are very close to 0. (Your answer should involve both $\overline{h^{(t)}}$ and $\overline{c^{(t)}}$.)

[1] http://www.cs.toronto.edu/~rgrosse/courses/csc421_2019/syllabus.pdf

2. **Multidimensional RNN [3pts]** One of the predecessors to the PixelRNN architecture was the multidimensional RNN (MDRNN). This is like the RNNs we discussed in lecture, except that instead of a 1-D sequence, we have a 2-D grid structure. Analogously to how ordinary RNNs have an input vector and a hidden vector for every time step, MDRNNs have an input vector and hidden vector for every grid square. Each hidden unit receives bottom-up connections from the corresponding input square, as well as recurrent connections from its north and west neighbors as follows:



The activations are computed as follows:

$$\mathbf{h}^{(i,j)} = \phi\left(\mathbf{W}_{\text{in}}^{\top}\mathbf{x}^{(i,j)} + \mathbf{W}_{\text{W}}^{\top}\mathbf{h}^{(i-1,j)} + \mathbf{W}_{\text{N}}^{\top}\mathbf{h}^{(i,j-1)}\right).$$

For simplicity, we assume there are no bias parameters. Suppose the grid is $G \times G$, the input dimension is D, and the hidden dimension is H.

(a) **[1pt]** How many weights does this architecture have? How many arithmetic operations are required to compute the hidden activations? (You only need to give big-O, not an exact count.)

(b) **[1pt]** Suppose that in each step, you can compute as many matrix-vector multiplications as you like. How many steps are required to compute the hidden activations? Explain your answer.

(c) **[1pt]** Give one advantage and one disadvantage of an MDRNN compared to a conv net.

3. **Reversibility [3pts]** In lecture, we discussed reversible generator architectures, which enable efficient maximum likelihood training. In this question, we consider another (perhaps surprising) example of a reversible operation: gradient descent with momentum. Suppose the parameter vector $\boldsymbol{\theta}$ (and hence also the velocity vector $\mathbf{p}$) are both D-dimensional. Recall that the updates are as follows:

$$\begin{aligned}\mathbf{p}^{(k+1)} &\leftarrow \beta\mathbf{p}^{(k)} - \alpha\nabla\mathcal{J}(\boldsymbol{\theta}^{(k)})\\ \boldsymbol{\theta}^{(k+1)} &\leftarrow \boldsymbol{\theta}^{(k)} + \mathbf{p}^{(k+1)}\end{aligned}$$

If we denote $\mathbf{s}^{(k)} = (\boldsymbol{\theta}^{(k)}, \mathbf{p}^{(k)})$, then we can think of the above equations as defining a function $\mathbf{s}^{(k+1)} = f(\mathbf{s}^{(k)})$.

(a) **[1pt]** Show how to compute the inverse, $\mathbf{s}^{(k)} = f^{-1}(\mathbf{s}^{(k+1)})$.

(b) **[2pts]** Find the determinant of the Jacobian, i.e.

$$\det \partial\mathbf{s}^{(k+1)}/\partial\mathbf{s}^{(k)}.$$

Hint: first write the Jacobian as a product of two matrices, one for each step of the above algorithm.