Topics in Artificial Intelligence (CPSC 532S): Assignment #1

Due on Tuesday, September 20, 2021 at 11:59pm PST

In this assignment you will get hands on experience with the basic operations, processing and inner-workings of traditional deep learning libraries. There are many deep learning libraries that allow easy creation of complex neural network architectures. In future assignments, for example, we will use **PyTorch** to do this (other popular ones are Keras, TensorFlow and Theano). However, those libraries in an interest of ease of use often abstract a lot of detail. The basics learned in this assignment will give you hands on experience on how they work under the hood and give you skills necessary to implement new types of layers and whole architectures (if necessary) and to think through corresponding algorithmic and computational issues.

Problem 1 (40 points)

The key to learning in deep neural networks is ability to compute the derivative of a vector function $\mathbf{f}$ with respect to the parameters of the deep neural network. Computing such derivatives as closed-form expressions for complex functions $\mathbf{f}$ is difficult and computationally expensive. Therefore, instead, deep learning packages define computational graphs and use automatic differentiation algorithms (typically backpropagation) to compute gradients progressively through the network using the chain rule.

Let us define the following vector functions:

$$y_1 = \mathbf{f}_1(x_1, x_2) = e^{2x_2} + x_1 sin(3x_2^2) \quad (1)$$

$$y_2 = \mathbf{f}_2(x_1, x_2) = x_1 x_2 + \sigma(x_2), \quad (2)$$

where $\sigma(\cdot)$ denotes the standard sigmoid function. This is equivalent to a network with two inputs $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ and two outputs $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \mathbf{f}(\mathbf{x})$ and a set of intermediate layers (note that bold indicates vectors).

(a) Draw a computational graph. Computational graph should be at the level of elemental mathematical operations (multiplication, square, sine, etc.) and constants/variables.

(b) Draw backpropagation graph, based on computational graph above.

(c) Compute the value of $\mathbf{f}(\mathbf{x})$ at $\mathbf{x} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$ using forward propagation. Please do not just plug in numbers directly into Eq. (1) and Eq. (2), this is not the intent here.

(d) At $\mathbf{x} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$, compute Jacobian using forward mode auto-differentiation. Show all steps.

(e) At $\mathbf{x} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$, compute Jacobian using backward mode auto-differentiation. Show all steps.

Remember that Jacobian is defined as $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$.

Note #1: The goal here is **not** the final result, it could be obtained easily by any symbolic differentiation package (Mathematica, Matlab, etc.) and you will loose points if this is what you do. The goal is to show that you understand how backpropagation actually works and can do this by hand. This will be a bit painful, but in fact that is part of the point: ability of AutoDiff algorithms to do this automatically is a huge benefit.

Note #2: I am providing the LaTex template for the assignment. You are welcome to do this part directly in LaTex and hand in a PDF. However, this is not required. I have had students turn in photos of pages where the math is done by hand. As long as it is legible, that's OK.

Problem 2 (15 points)

Consider a neural network with N input units denoted by $\mathbf{x} \in \mathbb{R}^N$ vector, M output units denoted by $\mathbf{y} \in \mathbb{R}^M$ vector, and K hidden units denoted by $\mathbf{h} \in \mathbb{R}^K$ vector. The hidden layer has an activation function σ (e.g., a sigmoid or ReLU). The resulting equations that govern the behavior of this simple network are:

$$\begin{aligned} \mathbf{z} &= \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \\ \mathbf{h} &= \sigma(\mathbf{z}) \\ \mathbf{y} &= \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)} \end{aligned}$$

The loss function $\mathcal{L}$ involves an L2 error on prediction $\mathcal{E}$ plus a regularizer $\mathcal{R}$ as follows:

$$\begin{aligned} \mathcal{L} &= \mathcal{E} + \mathcal{R} \\ \mathcal{R} &= \mathbf{r}^T\mathbf{h} \\ \mathcal{E} &= \frac{1}{2}||\mathbf{y} - \mathbf{s}||^2 \end{aligned}$$

where $\mathbf{r}$ and $\mathbf{s}$ are given (e.g., $\mathbf{s}$ is a ground truth prediction for $\mathbf{x}$).

(a) Draw the computational graph relating $\mathbf{x}$, $\mathbf{z}$, $\mathbf{h}$, $\mathbf{y}$, $\mathcal{L}$, $\mathcal{E}$, $\mathcal{R}$. Note, in this case we expect nodes to be representing collections of values, not each neuron individually.

(b) Derive the backpropagation equations for computing $\frac{\partial \mathcal{L}}{\partial \mathbf{x}}$. Please use σ' to denote the derivative of the activation function.

Credit: Problem 2 is adopted from University of Toronto's CSC421/2516 Problem Set 1.

Problem 3 (80 points)

The second half of the assignment requires Google Colab and is distributed as a notebook. All instructions are given in the notebook itself. The assignment does not require any libraries beyond vanilla Python. Once the assignment is done hand in both the written portion and the programming portion using `Canvas`. When handing in programming assignment make sure you hand in all the code and the notebook contains the output of executed code.

Those who never used Python Jupyter, can find many tutorials online, including on YouTube,
e.g., `https://www.youtube.com/watch?v=HW29067qVWk`
or contact TAs for help.