CS479/679: Neural Networks
Assignment 2
Due: 11:59 PM (EST), Feb 14, 2023, submit on LEARN.
Include your name and student number!

Submit your writeup in pdf and all source code in a zip file (with proper documentation). Write a script for each programming exercise so that the TAs can easily run and verify your results. Make sure your code runs!

[Text in square brackets are hints that can be ignored.]

Question 1: Softmax/Categorical CE

[30 marks] Consider a classification problem in which you have K classes. Suppose you have a labelled dataset containing pairs of inputs and class labels, $(\mathbf{x}, \ell)$, where $\mathbf{x} \in \mathbb{R}^X$ and $\ell \in \{1, 2, \ldots, K\}$.

Your neural network's output is a probability vector based on the softmax activation function, so that if z_k is the input current for output node k, then the activation of output node y_k is

$$y_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}} \quad , \quad k = 1, \ldots, K\,.$$

Thus, $\mathbf{y} \in [0,1]^K$, and $y_k = P\left(k = \ell \,|\, \mathbf{x}\right)$.

Suppose that your loss function is categorical cross-entropy,

$$L(\mathbf{y}, \mathbf{t}\,) = -\sum_{k=1}^{K} t_k \ln y_k \ ,$$

where $\mathbf{t}$ is the one-hot indicator vector for class ℓ, so that $t_k = \delta_{k\ell}$ (Kronecker delta). Derive the simplest expression you can for $\nabla_{\mathbf{z}} L$, the gradient of the loss function with respect to the input currents to the output layer.

Make sure your derivation is organized, and explain your steps.

What to submit

For this question, you can:

- typeset your solutions using a word-processing application, such as Microsoft Word, LaTeX, Google docs, etc., or
- write your solutions using a tablet computer, or
- write your solutions on paper and take photographs.

In any case, it is **your responsibility** to make sure that your solutions are sufficiently legible.

Automatic Differentiation

Question 1: Auto-Differentiation by Hand

[20 marks]

Construct an expression graph for

$$L = w\,\sigma\left(x_1 m_1\right) + w\ \tanh\left(x_2 m_2\right) + b$$

where σ is the logistic function. Use your expression graph to compute an explicit formula for the partial derivatives of L with respect to each of its dependent variables (w, x_1, x_2, m_1, m_2, and b). **Label each operation in the graph with its derivative**, and **label each variable (including intermediate variables) with the partial derivative of L with respect to it**. You can (and should) use intermediate variables in your answer.

Question 2: Backprop using Auto-Differentiation

[50 marks]

Download the following and place in a single folder:

- `a03q2_YOU.ipynb`
- `matad.py`
- `utils.py`.

Note that the `MatOperation` functions return `Mat` objects, but their `backward` functions deal with NumPy arrays, not `Mat` arrays.

Theorem 1: Consider $L(Y) \in \mathbb{R}$, for $Y \in \mathbb{R}^{D\times N}$. That is, $L : \mathbb{R}^{D\times N} \to \mathbb{R}$. Let $Y = H \cdot W$, where $H \in \mathbb{R}^{D\times M}$, $W \in \mathbb{R}^{M\times N}$, and $\cdot$ refers to the matrix product. Then $\nabla_H L = \nabla_Y L \cdot W^{\mathrm{T}}$, and $\nabla_W L = H^{\mathrm{T}} \cdot \nabla_Y L$.

The jupyter notebook `a03q2_YOU.ipynb` creates and tries to train a neural network on a simple dataset. However, some critical parts of the code are incomplete. Complete the implementation by doing the following:

(a) `backward`: Complete the implementation of the `backward` method in the `Mul` class. As the documentation states, the `Mul` class implements matrix-matrix multiplication. The `backward` method takes a 2D NumPy array as input, applies its own term to the chain of derivatives, and sends those derivatives (NumPy arrays) to the `backward` function of each of its arguments. You will probably find theorem 1 useful.

(b) `__call__`: Complete the implementation of the `__call__` function in the `Connection` class. This class represents the connection weights and biases between two `Population` layers. The `__call__` function takes the activity of the layer below, multiplies it by the connection weights, and adds the bias, and returns the resulting input current (as a `Mat` array).

Hint: Take advantage of the properties of the `Mat` and `MatOperation` classes. If you do it properly, your solution to part (c) will be much easier.

(c) `learn`: Complete the `learn` function in the `Network` class. To get full marks, you must use the automatic-differentiation functionality of the `Mat` and `MatOperation` classes. Notice that the `Network` class has a method called `parameters()` that returns a list of all the `Mat` objects in the network that correspond to connection weights and biases.

There is some code at the end of the notebook that creates a network and runs `learn` on a the simple dataset. If your code works, you should see it learn to classify the dataset correctly, with accuracy better than 98%.

Be sure to make your code readable, and include informative comments.

What to submit

Question 1: You can:

- typeset your solutions using a word-processing application, such as Microsoft Word, LaTeX, Google docs, etc., or
- write your solutions using a tablet computer, or
- write your solutions on paper and take photographs.

In any case, it is **your responsibility** to make sure that your solutions are sufficiently legible.

Question 2:

Make sure you submit your solutions, and not just a copy of the supplied skeleton code. We suggest you rename the `ipynb` file by replacing "`YOU`" with your WatIAM ID (eg. `a03q2_jorchard.ipynb`). You should **not** submit `matad.py` or `utils.py`.