CMPUT 261, Fall 2026
Assignment #3

1. (**Linear Classification**; 4 points)
   Consider the following dataset of $[(x_1, x_2), y]$ for a binary classification problem:

$$\begin{aligned} S = \{&[(1,3),+1],\\ &[(2,1),-1],\\ &[(2,4),+1],\\ &[(4,1),-1],\\ &[(5,2),-1],\\ &[(4,5),+1]\} \end{aligned}$$

   (a) **[2 points]** What is the *0/1 loss* on S for the linear classification hypothesis

$$h(x_1, x_2; w_0, w_1, w_2) = \operatorname{sgn}(w_0 + w_1 x_1 + w_2 x_2)$$

   with weights $w_0 = 3, w_1 = -1, w_2 = 0$? Recall that

$$\operatorname{sgn}(z) = \begin{cases} +1 & \text{if } z \geq 0, \\ -1 & \text{otherwise.} \end{cases}$$

   (b) **[2 points]** What is the *L2 regularized* 0/1 loss on S for the linear classification hypothesis

$$h(x_1, x_2; w_0, w_1, w_2) = \operatorname{sgn}(w_0 + w_1 x_1 + w_2 x_2)$$

   with weights $w_0 = 0, w_1 = -1, w_2 = 1$ and regularization parameter $\lambda = 1$?

2. (**Neural Networks**; 54 points) The MNIST dataset is a set of images of handwritten digits labelled by their actual digit. We will operate on two versions of this dataset: one with the images shifted 2 pixels to the upper-left, and one with the images shifted 2 pixels to the bottom-right.

   This question requires the use of PyTorch. You may need to install PyTorch using the following command:

```
pip3 install --user torch torchvision
```

   Torch idiosyncrasies. In class, we specified the units in a dense feedforward network as performing a linear combination followed by an activation function application (e.g., relu of linear combination). However, PyTorch treats these operations as separate layers. See the implementation of `MLP1` for an example: The hidden layer is specified as a `Linear` layer followed by a `ReLU` layer. Similarly, to specify a convolutional layer with ReLU activation, you first specify a `Conv2d` layer followed by a `ReLU` layer.

(a) **[10 points]** Implement a fully-connected feed-forward neural network for classifying MNIST images according to the digit that they represent by editing the `MLP2` class in the provided `cnn.py` file.

The network should have two hidden layers: one dense linear layer with 128 rectified linear ('relu') activation, and one with 64 rectified linear units. The output should be a fully-connected linear layer of 10 units with the softmax activation. The `cnn.py` file contains an example implementation of a network with a single hidden layer in the `MLP1` class that you may template from. The `main` function will test your program for you; you may add any tests that you like.

(b) **[30 points]** Implement a convolutional neural network for classifying MNIST images by editing the `CNN` class in the provided `cnn.py` file.

The network should have the following architecture:

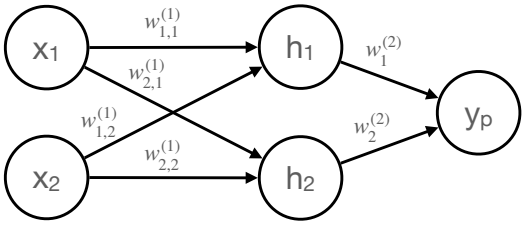
- A layer of 32 convolutional units with a kernel size of 5×5 and a stride of $1, 1$, with relu activation
- A max-pooling layer with a pool size of 2×2 and a stride of $2, 2$.
- A layer of 64 convolutional units with a kernel size of 5×5 and the default stride, with relu activation.
- A max-pooling layer with a pool size of 2×2 and the default stride.
- A `Flatten` layer (to reshape the image from a 2D matrix into a single long vector)
- A layer of 512 fully-connected linear units with relu activation
- A layer of 10 fully-connected linear units with log-softmax activation (the output layer). This gives an output of log-probabilities.

(c) **[2 points]** What was the accuracy of your trained 2-hidden-layer feedforward network on the two test sets?

(d) **[2 points]** What was the accuracy of your trained convolutional neural network on the two test sets?

(e) **[10 points]** Did one of your implementations perform substantially better on one of the test sets than the other implementation did? If so, why? If not, why not?

3. (**Automatic Differentiation**; 31 points)
   Consider the following feedforward neural network:



   - Nodes x_1 and x_2 are input nodes.
   - Nodes h_1, h_2, and y_p are rectified linear units; i.e., they use the activation $g(t) = \max\{0, t\}$.
   - The inputs to h_1 are x_1 and x_2, weighted by $w^{(1)}_{1,1}$ and $w^{(1)}_{1,2}$ respectively; h_2's inputs are x_1 and x_2 with weights $w^{(1)}_{2,1}$ and $w^{(1)}_{2,2}$ respectively.
   - The inputs to y_p are h_1 and h_2 weighted by $w^{(2)}_1$ and $w^{(2)}_2$ respectively. (Note that the output node is called y_p to distinguish it from the "true" value of y in a training example)
   - There are no offsets in this network; the weights are the only parameters.

   (a) **[5 points]** Let $y_p = f\left(x_1, x_2; w^{(1)}_{1,1}, w^{(1)}_{1,2}, w^{(1)}_{2,1}, w^{(1)}_{2,2}, w^{(2)}_1, w^{(2)}_2\right)$ be the prediction of the neural network on an input x_1, x_2. Give an expression for f.

   (b) **[3 points]** Let $L\left(w^{(1)}_{1,1}, w^{(1)}_{1,2}, w^{(1)}_{2,1}, w^{(1)}_{2,2}, w^{(2)}_1, w^{(2)}_2\right)$ be the mean-squared error of f on a single example (x_1, x_2, y). Give an expression for L.

   (c) **[10 points]** Give a *finite numerical algorithm* for computing L.

(d) **[13 points]** Give a *finite numerical algorithm* for computing the gradient of L with respect to $\left(w_{1,1}^{(1)}, w_{1,2}^{(1)}, w_{2,1}^{(1)}, w_{2,2}^{(1)}, w_{1}^{(2)}, w_{2}^{(2)}\right)$. You need not re-define the intermediate steps for the forward pass defined in (3c). By convention, you may assume that $\frac{d}{dz}\max\{z, 0\}\big|_{z=0} = 0$.

4. (**Self-Attention**; 10 points) Consider a self-attention unit with three inputs:

$$x_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad x_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \qquad x_3 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

For an input $x_i = (x_{i1}, x_{i2})^\top$, the unit computes scalar *queries*, *keys,* and *values* as follows:

$$q_i = 1x_{i1} \qquad k_i = (\ln 2)x_{i2} \qquad v_i = x_{i1} + 3x_{i2}.$$

The attention weights and outputs are

$$w_{ij} = \frac{\exp(q_j k_i)}{\sum_{\ell=1}^{3} \exp(q_j k_\ell)} \qquad y_j = \sum_{i=1}^{3} w_{ij} v_i.$$

Here w_{ij} is the weight placed on input i's value when computing output j.

(a) **[4 points]** Compute the queries, keys, and values for all three inputs (i.e., q_i, k_i, v_i for all i).

(b) **[4 points]** Compute the values of y_1 and y_2 (this will require you to first compute their attention weights).

(c) **[2 points]** Explain why the weights assigned to inputs for computing y_1 differ from the assigned to inputs for computing y_2, even though the keys and values for the inputs are the same in both computations.