Homework 2

Deadline: Wednesday, Jan. 24, at 11:59pm.

Submission: You must submit your solutions as a PDF file through MarkUs[1]. 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 2 days after the deadline; until that time, you should submit through MarkUs. If you want to submit the assignment more than 2 days late, please e-mail it to `csc321ta@cs.toronto.edu`. The reason for this is that MarkUs won't let us collect the homeworks until the late period has ended, and we want to be able to return them to you in a timely manner.

Weekly homeworks are individual work. See the Course Information handout[2] for detailed policies.

1. **Perceptron Algorithm.** Suppose we have the following training examples in a 2-dimensional input space, and *no bias term*. Following our discussion of the perceptron algorithm, we use $t = -1$ rather than $t = 0$ to denote the negative class.

x_1	x_2	t
1	-2	1
0	-1	-1

Recall the perceptron algorithm from lecture. It repeatedly applies the following procedure, stopping when the weights are strictly within the feasible region (i.e. not on the boundary):

For each training case $(\mathbf{x}^{(i)}, t^{(i)})$,
  $z^{(i)} \leftarrow \mathbf{w}^T \mathbf{x}^{(i)}$
  If $z^{(i)} t^{(i)} \leq 0$,
    $\mathbf{w} \leftarrow \mathbf{w} + t^{(i)} \mathbf{x}^{(i)}$

Here we work through the algorithm by hand.

   (a) **[2pts]** Sketch out the problem in weight space. In particular: draw and label the axes, draw the half-space corresponding to each of the two training examples, and shade the feasible region. (Remember that there is no bias term.)

   (b) **[2pts]** Simulate the perceptron algorithm by hand, starting from the initial weights $w_1 = 0, w_2 = -2$. On the plot from part (a), mark an X on every point in weight space visited by the algorithm until it stops. You do not need to provide anything for this part other than the X's on the plot. *Hint: the weights are updated 12 times. It will be tedious if you compute all the updates using arithmetic; instead, plot the first few updates as you go along, and you should start to notice a visual pattern.*

[1] `https://markus.teach.cs.toronto.edu/csc321-2018-01`
[2] `http://www.cs.toronto.edu/~rgrosse/courses/csc321_2018/syllabus.pdf`

2. **Feature Maps.** Suppose we have the following 1-D dataset for binary classification:

x	t
-1	1
1	0
3	1

   (a) **[1pt]** Argue briefly (at most a few sentences) that this dataset is not linearly separable. Your answer should mention convexity.

   Now suppose we apply the feature map

$$\boldsymbol{\psi}(x) = \begin{pmatrix} \psi_1(x) \\ \psi_2(x) \end{pmatrix} = \begin{pmatrix} x \\ x^2 \end{pmatrix}.$$

   Assume we have no bias term, so that the parameters are w_1 and w_2.

   (b) **[2pts]** Write down the constraint on w_1 and w_2 corresponding to each training example, and then find a pair of values (w_1, w_2) that correctly classifies all the examples. Remember that there is no bias term.

3. **Loss Functions. [3pts]** Suppose we have a prediction problem where the target t corresponds to an angle, measured in radians. A reasonable loss function we might use is

$$\mathcal{L}(y, t) = 1 - \cos(y - t).$$

   Suppose we make predictions using a linear model,

$$y = \mathbf{w}^\top \mathbf{x} + b.$$

   As usual, the cost is the average loss over the training set:

$$\mathcal{E} = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(y^{(i)}, t^{(i)}).$$

   Derive a sequence of vectorized mathematical expressions for the gradients of the cost with respect to $\mathbf{w}$ and b. As usual, the inputs are organized into a design matrix $\mathbf{X}$ with one row per training example. The expressions should be something you can translate into a Python program without requiring a `for`-loop. Your answer should look like:

$$\begin{aligned} \mathbf{y} &= \cdots \\ \frac{\partial \mathcal{E}}{\partial \mathbf{y}} &= \cdots \\ \frac{\partial \mathcal{E}}{\partial \mathbf{w}} &= \cdots \\ \frac{\partial \mathcal{E}}{\partial b} &= \cdots \end{aligned}$$

   You can use $\sin(\mathbf{A})$ to denote the sin function applied elementwise to $\mathbf{A}$. Remember that $\partial \mathcal{E}/\partial \mathbf{w}$ denotes the gradient vector,

$$\frac{\partial \mathcal{E}}{\partial \mathbf{w}} = \begin{pmatrix} \frac{\partial \mathcal{E}}{\partial w_1} \\ \vdots \\ \frac{\partial \mathcal{E}}{\partial w_D} \end{pmatrix}$$