Caregivers and Machine Learning 2023
Homework Assignment 4 - **Neural Networks**

Deadline: Friday, April 21 @ 11:59pm

This assignment contains three sections: conceptual questions **[25 pts]**, a coding problem **[25 pts]**, and a bonus math question **[+10 pts]** for a total of **[50 pts]**.
Note: For all written questions please be concise and limit the length of your answers

Submission: Submit the following through the https://learn.vectorinstitute.ai course page:

- Your written answers to Sections 1 and 3, and outputs/answers requested for Section 2, as a PDF file titled hw4_name_writeup.pdf. You can produce the file however you like (e.g. LaTex, Microsoft Word, Google docs, scan or picture of handwritten notes), as long as it is readable.

- Your code for Section 2, as a Python (.py) file or a Jupyter/Colab notebook file (.ipynb), titled as "hw4_name_python.ipynb/py" (For this assignment it is easiest to save a copy of the Colab notebook that is provided to your drive, work in that, and then submit your amended notebook)

1. **Conceptual Questions [25 pts]:**

   a. **[4 pts]** In your own words, what is a feed-forward neural network (FFNN)? Provide a graphical illustration for a feed-forward neural network with input, hidden and output layers. Supporting your answer with mathematical expressions is certainly encouraged.

   b. **[4 pts]** In your own words, what is a convolutional neural network (CNN)? What is the difference compared to a feed-forward neural network?

   c. **[4 pts]** Explain the principle of minimizing a cost function C using gradient descent.

   d. **[5 pts]** In your opinion, why is it useful to use non-linear activation functions in a neural network? Conceptual or technical answers are both accepted.
   Hint for a technical answer: You can try to show that a feed-forward neural network with linear activation functions and with one hidden layer is equivalent to a feedforward neural network with no hidden layer. Does the same conclusion hold when non-linear activation functions are used?
   Remark: In this context a linear activation function means the identity function.

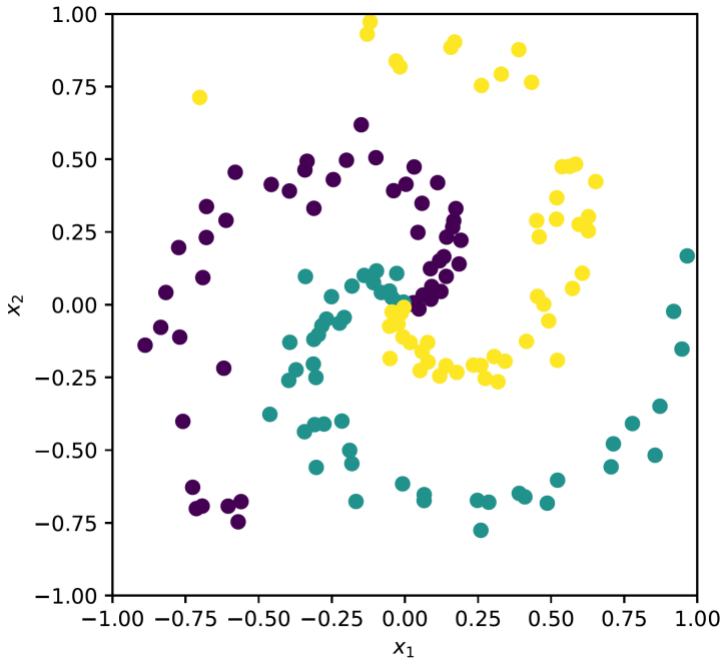
   e. **[5 pts]** In a multi-classification problem with K classes, a neural network has an output layer with size $K > 2$. Explain why having a size K instead of a size 1 at the output layer is useful for multi-classification. Second, explain why using 'Softmax' activation is better compared to 'Sigmoid' activation at the output layer with size K.
   Hint for first question: Consider the case of $K = 3$ classes, when the prediction we want to express is 50% to get class 0 and 50% chance to get class 2.

   f. **[3 pts]** Think of all the models and algorithms you've learned so far. Specifically, k-Nearest Neighbors (k-NN), Decision Trees and Linear Regression. What are some of the advantages and disadvantages of Neural Network models compared to those? What types of problems might Neural Networks be better or worse suited to than these algorithms (e.g. problems with more or less data, supervised vs unsupervised learning, etc.)?
   Remark: This is a more open-ended question. We just want to see how you think- you don't need to compare/contrast neural networks to each of those three models in detail, no need to cover all bases. Feel free to just pick one of the models you have seen (e.g. Decision Trees, although you are welcome to do more). Keep your answers as short as you can.

2. **Coding Question [25 pts + 3 bonus pts]**

The objective of this problem is to become comfortable with using the Python package PyTorch to create and train a simple feedforward neural network for supervised learning.

You will use and modify this Google Colab notebook. The first section of this code generates a random dataset of two-dimensional points with K branches. For example, when $K = 3$ this dataset might look as follows:



For each datapoint $\mathbf{x} = (x_1, x_2)$, the label is the branch index such that $y = 0, 1$ or 2 for the example above. Our goal is to implement a neural network capable of classifying the branches.

This network will compare its output with labels that have been converted to the so-called one-hot encoding. For a given label $y = k$, the corresponding one-hot encoding is a K-dimensional vector with all entries zero except for the k^{th} entry (which has value 1). So when $K = 3$ the one-hot encodings for the labels are

$$0 \to \begin{bmatrix}1\\0\\0\end{bmatrix}, \qquad 1 \to \begin{bmatrix}0\\1\\0\end{bmatrix}, \qquad 2 \to \begin{bmatrix}0\\0\\1\end{bmatrix}$$

The code first defines the structure of the neural network and then uses the dataset to train this network. The code can generate the following files: 'spiral_data.pdf' (a plot of the dataset such as the one above) 'spiral_results.pdf' (which displays three plots illustrating the results of training), and 'trainedNNaccuracy_vs_magNoise.pdf' (displays the accuracy of the trained

NN as a function of noise). As part of the problem you will then write/tweak a bit of code to be able to generate another two files, 'trainedNNaccuracy_vs_hiddenSize.pdf' (accuracy vs number of hidden neurons) and 'trainedNNaccuracy_vs_K.pdf' (accuracy vs number of labels). There are plenty of instructions in the notebook (in the form of comments) to guide you for whenever you have to tweak some code as part of the questions below, but please ask questions if you need further clarification and help, I will be happy to help.

a. Run the code and look at how it attempts to classify the two-dimensional space. You should find that the resulting classifier separates the two-dimensional space using lines, and thus does a poor job of representing the data. You don't need to understand every line of code. But feel free to ask me (Andrew) if you have questions about the details!

b. **[5 pts]** Look through the section of code labelled "Define the network architecture". On paper, draw the neural network corresponding to the one in the code for the case of K=3 branches (labels). Pay particular attention to the number of neurons in each layer.

c. **[5 pts]** Add in a hidden layer with 4 neurons and study how this hidden layer changes the output. On paper, draw the neural network in this case.

d. **[5 pts]** Replace the activation function in the first layer with a sigmoid and then a rectified linear unit (ReLU). Discuss how the choice of activation function changes your results.

e. **[5 pts]** Study the effects of increasing and decreasing the learning rate hyperparameter. Provide short comment on what you notice.

f. **[5 pts + 3 bonus pts]** Here you will plot accuracy of the final trained (optimized) neural network (with one hidden layer) as a function of three changing hyperparameters:

    i. **[2 pts] Hyperparameter 1**: mag_noise (the magnitude of noise in the data) - the code is already written here, you just need to tweak one line of code to include the hidden layer (instructions for this should be clear in the code, see relevant comment!) and then run it.

    ii. **[3 pts] Hyperparameter 2**: hidden_size (the number of neurons in the hidden layer) - you will need to copy paste the code used for mag_noise and tweak relevant lines of code to plot accuracy vs hidden_size. If in doubt, start with 1 hidden neuron and then increase to about 8 or 9 in increments of 1.

    iii. **[BONUS 3 pts] Hyperparameter 3**: K (the number of different labels) - again, you will need to copy paste the code used for mag_noise and tweak relevant lines of code to plot accuracy vs K. This is a bonus question. If in

doubt, you can start from $K = 2$ ($K = 1$ is trivial) to about 6 or 7 in increments of 1.

Note: If you feel that your network is not converging in any of the above questions, go ahead and increase N_epochs (the number of training steps, as you will see in the code) to see if you achieve better convergence (for example if N_epochs = 10000 doesn't quite cut it for a given set of hyperparameters, maybe try N_epochs = 20000 or 15000 and see if that works better). Also, while you will need to play with the learning rate in part(e) above, feel free to modify the learning rate for all the other parts of this problem as well to try to achieve better convergence (the default value we select in the code is learning_rate $= 1 \rightarrow$ this should work well enough but feel free to play with it).

3. **Bonus Math Question [+10 pts]**

Suppose that we have a multiclassification problem with K classes. Let us assume we have a neural network with D inputs and K outputs (so an input layer and an output layer, but no hidden layer) that is geared to learn the mapping between the D-dimensional inputs and the K classes. Let $\boldsymbol{x} = (x_1, x_2, \dots, x_D)$ be a D-dimensional vector of inputs. The output layer first performs a linear transformation of the inputs as

$$z_k = \sum_{j=1}^{D} w_{kj} x_j + b_k$$

for $k = 1, \dots, K$. Here w_{kj} are the output layer weights/parameters that form the $K \times D$ weight matrix $\mathbf{W}$, while b_k is the k^{th} element of the bias vector $\boldsymbol{b}$. This bias can be incorporated as part of $\mathbf{W}$ by appending it to the matrix as the leftmost column such that $\mathbf{W} \in \mathbb{R}^{K \times (D+1)}$ and adding a dummy variable $x_0 = 1$ such that $\boldsymbol{x} = (x_0, x_1, x_2, \dots, x_D)$. Thus, we obtain

$$z_k = \sum_{j=0}^{D} w_{kj} x_j$$

or in vectorized form,

$$\boldsymbol{z} = \mathbf{W}\boldsymbol{x}$$

Next, we use the Softmax function to compute the final probabilities associated with the K classes (i.e. K outputs) in our problem, giving us

$$y_k = \text{softmax}(z_1, \dots, z_K)_k = \frac{e^{z_k}}{\sum_{k'=1}^{K} e^{z_{k'}}}$$

Now the loss function is the cross-entropy given by

$$\mathcal{L}_{CE}(\boldsymbol{y}, \boldsymbol{t}) = -\sum_{k=1}^{K} t_k \log y_k = -\boldsymbol{t}^{\mathrm{T}} \log \boldsymbol{y}$$

where $\boldsymbol{t} = (0, \dots, 0, 1, 0, \dots, 0)$, with the k^{th} element equal to 1 and all other elements 0, is the target vector for an input whose true label is the k^{th} class.

For gradient descent updates, we must compute the derivative of the loss function with the respect to the network weights. If we do the math, we find

$$\frac{\partial \mathcal{L}_{CE}}{\partial w_{kj}} = \frac{\partial \mathcal{L}_{CE}}{\partial z_k} \cdot \frac{\partial z_k}{\partial w_{kj}} = (y_k - t_k) \cdot x_j$$

If we define $\boldsymbol{w_k} = (w_{k1}, w_{k2}, \ldots, w_{kD})$, i.e. the row vector containing the elements of the k^{th} row of **W**, we can define

$$\frac{\partial \mathcal{L}_{CE}}{\partial \boldsymbol{w_k}} = \frac{\partial \mathcal{L}_{CE}}{\partial z_k} \cdot \frac{\partial z_k}{\partial \boldsymbol{w_k}} = (y_k - t_k) \cdot \boldsymbol{x}$$

where $\frac{\partial \mathcal{L}_{CE}}{\partial \boldsymbol{w_k}} \equiv \left(\frac{\partial \mathcal{L}_{CE}}{\partial w_{k1}}, \frac{\partial \mathcal{L}_{CE}}{\partial w_{k2}}, \ldots, \frac{\partial \mathcal{L}_{CE}}{\partial w_{kD}}\right)$ and $\frac{\partial z_k}{\partial \boldsymbol{w_k}} \equiv \left(\frac{\partial z_k}{\partial w_{k1}}, \frac{\partial z_k}{\partial w_{k2}}, \ldots, \frac{\partial z_k}{\partial w_{kD}}\right)$

a. **[3 pts]** Show that

$$\frac{\partial z_k}{\partial \boldsymbol{w_k}} = \boldsymbol{x}$$

b. **[7 pts]** Show that

$$\frac{\partial \mathcal{L}_{CE}}{\partial z_k} = y_k - t_k$$

Hint: Start by writing

$$\frac{\partial \mathcal{L}_{CE}}{\partial z_k} = \sum_{k'=1}^{K} \frac{\partial \mathcal{L}_{CE}}{\partial y_{k'}} \cdot \frac{\partial y_{k'}}{\partial z_k}$$

and then go from there.