DS 4420: Machine Learning and Data Mining 2 **(Due: Jan. 23 2026)**

Homework Assignment #1

Professor: Eric Gerber *Name:* _____

Instructions: Please include the following information on the first page of **each file of** your completed homework:

1. Your name
2. DS 4420
3. Homework #1

You will submit **up to four files** to Gradescope for this homework:

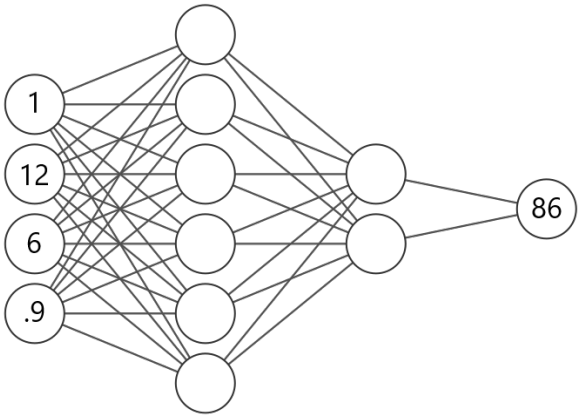
- Handwritten/latex typeset answers may be included in their own **.pdf** file or as part of either of the below
- A **.ipynb** file with all python code/output
- A **.pdf** file knitted from an included **.rmd** with all R code/output

Answers that are not supported by reasoning/work will not receive full credit. **Homework is due by 11:59 pm, via Gradescope, on the date above.** Late submissions will **not** be accepted, but you may receive extra credit for early submission (see syllabus for details).

You will also be graded on organization/neatness of the submitted files.

MLP for Regression (50 points)

(1) Below is a MLP defined for predicting the numeric grade of a student given a bias term and three input features (hours spent playing video games per week, hours spent studying per week, attendance proportion):



Use the above to answer the following questions.

(a) How many total trainable parameters does this MLP have? Show your calculation. How would this number change if we added one more node to the second hidden layer? Also, explain why we need non-linear activation functions between layers in a neural network. What would happen if we used only linear activation functions so that, for example in a network with two hidden layers, $f_w(x) = W^{(3)^T}(W^{(2)^T}(W^{(1)^T}x))$ is our prediction function?

(b) Given the $W^{(1)}$ matrix below, calculate the post-activation values for the first hidden layer $h^{(1)}$ assuming the *ReLU* activation function (i.e. find $h^{(1)} = \langle {W^{(1)}}^T x \rangle_+$):

$$W^{(1)} = \begin{bmatrix} 1 & -2 & 5 & 1 & -6 & 2 \\ 0 & 2 & 0 & -2 & 2 & 2 \\ 3 & -1 & -2 & -4 & 2 & 1 \\ -5 & 0 & 3 & 4 & -1 & 0 \end{bmatrix}$$

(c) Given the $W^{(2)}$ matrix below, and the post-activation values for the first hidden layer $h^{(1)}$, calculate the post-activation values for the second hidden layer $h^{(2)}$ assuming the sigmoid $\sigma(h) = \frac{1}{1+e^{-h}}$ activation function (i.e. find $h^{(2)} = \sigma({W^{(2)}}^T h^{(1)})$).

$$W^{(2)} = \begin{bmatrix} -2 & 1 \\ 0 & 3 \\ 1 & -1 \\ -4 & 6 \\ 1 & 3 \\ 0 & -1 \end{bmatrix}$$

(d) Given $W^{(3)} = \begin{bmatrix} -24 \\ 90 \end{bmatrix}$ and the post-activation values for the second hidden layer $h^{(2)}$, find the final predicted value for the student (use the linear activation function for the output node, i.e. $f_w(x) = {W^{(3)}}^T h^{(2)}$) under this MLP. What is the loss for this student (i.e. calculate the squared error)?

(e) Assume that we don't know any of the W matrices and want to fit the model by minimizing the MSE. Write out by hand the derivatives for use in updating the last two W matrices ($W^{(3)}$ and $W^{(2)}$) using backprop. That is, write out the functional forms for:

1. $\frac{d\mathcal{L}}{dW^{(3)}} = \frac{d\mathcal{L}}{df} \frac{df}{dw^{(3)}}$
2. $\frac{d\mathcal{L}}{dW^{(2)}} = \frac{d\mathcal{L}}{df} \frac{df}{dh^{(2)}} \frac{dh^{(2)}}{dW^{(2)}}$

Hints:

- The objective function is $\mathcal{L} = \frac{1}{n} \sum_i^n (f_w(x_i) - y_i)^2$, where $f_w(x_i) = {w^{(3)}}^T h^{(2)}$
- $h^{(2)} = \sigma({W^{(2)}}^T h^{(1)})$
- and the derivative of the function $\sigma(w^T h) = \frac{1}{1+e^{w^T h}}$ is $\frac{d\sigma}{dw} = \frac{h e^{-w^T h}}{(1+e^{-w^T h})^2} = h\sigma(w^T h)(1 - \sigma(w^T h))$.

MLP for Classification (50 points)

(2) On Canvas, there is the **evs.csv** file where each row is a single car (you can discard the **id** column before fitting the model). The data consist of 119 electric vehicles and their measurements. Our goal is to predict the y = drive (0 = Front, 1 = Rear) of the vehicle based on the 6 other features:

- Acceleration (meter per second squared: smaller values indicate *faster* acceleration)
- Top Speed (in MPH)
- Electric Range (in miles)
- Total Power (in horsepower)
- Battery Capacity (in ampere-hours (Ah))
- Charge Speed (in kilowatts (kW))

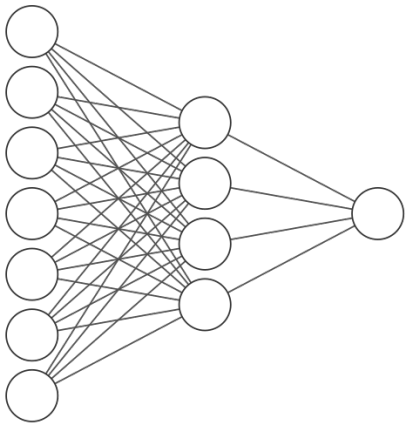
THE HOMEWORK CONTINUES ON THE NEXT PAGE

(a) In **both Python and R**, pre-process the data to prepare it for analysis by:

- Scaling the data using **Min-Max Scaling** (do **NOT** use standardization). We do this to make sure all x_i values are positive, since all the features should also always be positive.
- Separate the y feature into it's own NumPy array of appropriate 0's and 1's.
- Put all the x features into their own array, with an additional intercept/bias column.
- Split the data into training and test sets (we will skip validation, since the data set is relatively small).

Note: You should remember how to scale, one-hot encode categorical features, add bias terms, and cross validate (in Python) from DS 4400. If not, you might want to review the **preprocess_review.ipynb** file on Canvas. An equivalent **preprocess_r.R** file will give an example of how to accomplish the same tasks in R. Both, with an example data file, are provided in the **preprocessing review files.zip**.

(b) Manually implement (do **not** use keras, tensorflow, MLPregressor, or any other pre-packaged neural network function) an MLP using gradient descent based on the following architecture in **Python only**. For your choices of η and initial $W^{(1)}$ and $w^{(2)}$, ensure the algorithm is converging over 500 iterations/epochs (using the training data) and then print out the final estimates of $W^{(1)}$, $w^{(2)}$. Then predict using the test data and evaluate the accuracy of the algorithm.



Use, for a given observation's input $x \in \mathbb{R}^7$:

- The *ReLU* activation function for the hidden layer: $h = \langle W^{(1)^T} x \rangle_+$ where $W^{(1)} \in \mathbb{R}^{7\times 4}$
- The sigmoid activation function for the output layer:

$$f_w(x) = \sigma(w^{(2)^T} h) = \frac{1}{1 + e^{-w^{(2)^T} h}}, \text{ where } w^{(2)} \in \mathbb{R}^4$$

- The log loss objective function:

$$\mathcal{L} = \frac{1}{n} \sum_i^n -y_i \log(f_w(x_i)) - (1 - y_i) \log(1 - f_w(x_i))$$

Major Hints: The two updates require:

$$\frac{d\mathcal{L}}{dw^{(2)}} = \frac{1}{n} \sum_i^n (f_w(x) - y_i) h$$

$$\frac{d\mathcal{L}}{dW^{(1)}} = \frac{1}{n} \sum_i^n (f_w(x) - y_i)(x_i \otimes [w^{(2)} \odot \mathbb{1}_{\{W^{(1)^T} x > 0\}}]^T), \text{ where } \otimes \text{ is the Kronecker product}$$

THE HOMEWORK EXTRA CREDIT IS ON THE NEXT PAGE

Extra Credit: MLP in R (20 points)

For extra credit, implement an MLP with a single hidden layer in R and RMarkdown for predicting the hit points (**hp**) of a Pokémon given the six other features in the data (**excluding** pokedex and name). To do this, use the **pokedata_num.csv** file on Canvas.

(a) Within an RMarkdown file, in a code chunk, read in the data and pre-process it. Some of this is already done for you (for example, the categorical feature "Is Evolved" is already one-hot encoded). To proceed:

- separate the **hp** column into it's own vector and call it y (since it is the output)
- scale all the numeric input features using standardization
- add a bias column so that the resulting X matrix has 7 columns
- create training and test data sets for cross validation (I used approximately a 75-25 split)

Note: if you want some examples of data preprocessing in R, you can find them in the **preprocessing review files.zip** files. There are multiple ways to accomplish the above tasks, however, so feel free to use the way that makes most sense to you.

(b) In a code chunk, set up the infrastructure for your MLP. To proceed:

- define a reasonable step size η (depending on if your algorithm converges, you may want to fiddle with this later; I started with 0.01)
- define the width of your MLP (i.e. choose a number of nodes for the hidden layer)
- randomly initialize the two weight matrices based on the dimensions of the data/hidden layer
  - Make sure you set your seed each time you randomly initialize!

(c) A Pokémon's hit points (**hp**: the output feature) is bounded by 0 (i.e. it cannot be negative). As such, we will use the *ReLU* activation for **BOTH** the hidden layer **AND** the output layer. Below I have derived the two derivatives needed for completing the gradient descent steps assuming the MSE objective/loss function (aren't I nice? You'll still need to program them in the next part...). **Using LaTeXtypesetting in the R Markdown**, label the component parts of the chain rule used to derive these derivatives, and verify that they are the correct dimension:

$$\frac{d\mathcal{L}}{dW^{(2)}} = \frac{2}{n}\sum_{i=1}^{n}(f_w(x_i) - y_i)(h \times \mathbb{1}_{W^{(2)^T}h>0})$$

$$\frac{d\mathcal{L}}{dW^{(1)}} = \frac{2}{n}\sum_{i=1}^{n}\left((f_w(x_i) - y_i) \times \mathbb{1}_{W^{(2)^T}h>0}\right)\left[x\left(W^{(2)} \odot \mathbb{1}_{W^{(1)^T}x>0}\right)^T\right]$$

(d) Program the gradient descent algorithm and run it for 500 epochs (you will want to run it for a shorter amount while troubleshooting/until you are confident). Make sure you keep track of the loss to verify convergence with a plot. **Note:** you should be able to take the MLP code we've covered in class/you've written on the homework and update it using the derivatives provided in part (c).

(e) Once your MLP algorithm has converged, predict the observations in the test set and plot the predictions against the true values. Compute the R^2 (the proportion of variability in the observed test y values explained by the model*). Determine if your MLP is good for predicting a Pokémon's hit points.

* there are several ways to do this. If you are really dedicated to the Python Way®, there is a directly equivalent **R2_score()** (note the capital R) function in the **MLmetrics** package.