COMP 4437 Artificial Neural Networks, Spring'24
Homework 1

Lecturer: Arman Savran

Question:	1	2	3	Total
Points:	30	20	50	100

Late submission: 10 points less grade for each day late. If you have a valid excuse, explain it to me and your submission may be accepted without a penalty (have a health report) or with some mild penalty.

Code of Honor: By submitting this assignment, you accept the honor code: "I affirm that wherever I received any help on this assignment, I have cited the source or the person clearly, otherwise I affirm that I have not given or received any unauthorized help on this assignment, and that this work is my own." If you get help from your classmate but do not cite him/her, then it is cheating and you and your friend can both get a big penalty. If you reference your friend, then your friend will not have any penalty and you may get a partial penalty or may not get a penalty at all depending on the help received. Copy-paste is always subject to some penalty. Similarly, if you get help from the internet you should reference your source.

Submission:

- **Notebook.** All of your solutions must be included in one notebook which must be submitted after converting it to an HTML file. Clean your intermediate results before submitting, do not have a messy submission. You are given plenty of time, so prepare a neat submission. You can use markdown cells for text explanations, latex formulas, or image embeddings. Don't forget to put the title as HOMEWORK NO, your full name, and your student ID at the top of the notebook!
- **References.** Cite all the sources that you benefited from, e.g., for explanation or code. It could be the URL of a webpage, or a book, your classmate, or some other person.

1. **Standard normalization by vectorization.** Standard normalization (standard scaler) is commonly applied in neural networks. You will implement it with the method of vectorization. Vectorization is used to avoid very slow Python loops and to benefit from GPU and CPU boosting. You are given a dataset of N examples of D dimensional vectors, which is denoted by the matrix $\mathbf{X}_{N\times D}$, i.e., each row is for an example. Then a value of the element i from the dimension$j$ is

$$X_{i,j} \qquad \forall i,\ 0 \le i < N \quad and \quad \forall j,\ 0 \le j < D \tag{1}$$

We do the normalization by calculating the mean vector $\mathbf{m} = [m_j]_D$ and the standard deviation vector $\sigma = [\sigma_j]_D$ over the dataset.

$$Y_{i,j} = \frac{X_{i,j} - m_j}{\sigma_j} \qquad \forall i,\ 0 \le i < N \quad and \quad \forall j,\ 0 \le j < D \tag{2}$$

   (a) **(10 points) Understand broadcasting.** In the below code cell, the dataset is represented by `x_org`. As you see, it is created by an outer product of two vectors. Re-implement it using the element-wise multiplication operator, $*$, instead of the matrix multiplication operator, @, utilizing the broadcasting technique. The broadcasting allows doing operations with incompatible array shapes without using loops. You can read about it for instance at `https://numpy.org/doc/stable/user/basics.broadcasting.html`.

   (b) **(20 points) Standardization by broadcasting.** If you understand how broadcasting works, now implement the one-line code of standard normalization. Use the `np.mean` and `np.std` functions for the statistical calculations.
   Hint: You need to use the `axis` and `keepdims` arguments of these two functions to be able to implement a one-line code. Learn about these arguments from the documentation.

```python
import numpy as np

def stdnorm(X):
    """
    Inputs:
    - X: A numpy array of shape (N, D)

    Returns:
    A numpy array of shape (N, D) where Y[i] contains the same data as X[i],
    but normalized to have zero mean and unit standard deviation.

    Hint: Use the axis argument for calculations along the correct axis.
          Use the keepdims argument to be able to do broadcasting.
    """
    return # FILL HERE: one-line long code only, for part (b)

N = 10
D = 3
x_org = np.arange(1, N+1).reshape(N,1) @ np.arange(1, D+1).reshape(1,D)
x_org = # IMPLEMENT THE SAME BY THE OPERATOR *: one-line long code only, for part (a)
print(x_org.shape)
print(x_org)
x_norm = stdnorm(x_org)
print(x_norm)
print('Mean:', np.mean(x_norm, axis=0))
print('Std:', np.std(x_norm, axis=0))
```

2. **(20 points) The XOR problem.** Design and train a neural network with the minimum number of neurons that predicts XOR outputs with 100% accuracy. If you use more neurons than the minimum number you get **5 points** less. You must clearly show that your model works. Create the XOR dataset yourself. You are free to apply any learning method (shortly explain what you apply), but you MUST use only numpy for the computations. No other package is allowed for the computations, you can use other packages only for visualization or reporting. You are free to use any code from the lab sessions.

3. **Chance Prediction for graduate admission.**

   - **Dataset:** `admissionv2.csv`
   - **Task:** Regression of the admission chance given seven evaluation scores.
   - **Method:** Logistic regression by a two-layer neural network.
   - **Performance metrics:** R^2 score and root-mean-square-error (RMSE).
   - **Experimentation:** Hyperparameter tunning with grid-search.

   You will get points according to the below.

   (a) **(20 points) Implementation of the method.** We do logistic regression since the target is a probability value (admission chance). It is implemented by using the sigmoid activation at the output. Use sigmoid at the hidden layer as well. Use the MSE loss and L2 regularization. Implement stochastic gradient descent with momentum for the optimization. You MUST implement yourself only using the numpy package and can freely use any code from the lab practices.

   (b) **(10 points) Explanation of the performance metrics.** In your notebook, write the formulas of the R^2 and RMSE metrics, and briefly explain their descriptions (to learn about them search the internet for their definitions). What is the range of R^2 and RMSE (in general)? Note that a range can be unbounded as well. What values of R^2 and RMSE denote a perfect prediction, and what values denote terrible predictions? Explain your reasoning for those values, do not just write some numbers or descriptive words.

Note: You can use latex commands in a markup cell of your notebook for the formulas, or alternatively include their images in your notebook.

(c) **(20 points) Hyperparameter tunning via grid-search.** You will try to find the best model by repeating a three-step procedure many times for each different combination of the hyperparameters. The steps are:

1. Training on the training set (model fitting)
2. Prediction on the validation set
3. Performance evaluation on the validation set by R^2

- The most important hyperparameters are the layer size (the number of hidden layer neurons), and the learning rate. The others are the number of epochs, batch size, regularization strength, and momentum coefficient.
- One simple suggested strategy is to initially set these parameters to some common default values (you may search the internet, and may look at the ML packages, etc.). If they work fine (since you got a reasonable performance or you observed convergence at the training and validation curves), then you can proceed with hyperparameter tuning via grid search. You can also refine your grid at a later stage to hunt even better models. For the tunning, you must decide on a search range for each hyperparameter that you want to optimize. Decide on the range boundaries and step size for each one separately. Thus, each range makes one axis of your search grid. Then, you will run a loop for each axis, and will run nested loops for a multi-dimensional search grid.
- You can choose each range manually via a list or using suitable functions from numpy.
- You must at least use a 2-dimensional grid, for the layer size and the learning rate, after fixing the others. You can also use any higher-dimensional grid as well by incorporating some other parameters. You can look at the results on the grid and can refine the grid for a re-run.

Performance evaluation functions:

```
from sklearn.metrics import r2_score, root_mean_squared_error
```

Report. Report your best hyperparameters with the resulting R^2 and RMSE performances. Also, report your R^2 search-grid results via a heatmap (an example of heatmap usage is below). If you use a three or more-dimensional grid, only show the heatmap of the layer size and learning rate, using the reported best values for the others. Heatmap axis tick labels must be shown correctly so that a reader can see your search ranges and what values resulted in what performances.

Tip: Usually, a range chosen in a logarithmic scale is much more efficient than a linear scale.

Note: If you can't implement the regressor yourself, you can use the MLPRegressor below.

```
# Note: Use this only if you cannot implement the regressor yourself!
from sklearn.neural_network import MLPRegressor
# Read the meanings of the arguments at the documentation page.
regr = MLPRegressor(hidden_layer_sizes=???,
                    learning_rate='constant',
                    learning_rate_init=???,
                    activation='logistic',
                    solver='sgd',
                    random_state=1,
                    max_iter=???,
                    verbose=0,
                    batch_size=???,
                    alpha=???,
                    momentum=???,
                    nesterovs_momentum=False)
```

Starter code:
Data loading and exploration:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Read the dataset
dataset = pd.read_csv('admissionv2.csv')

# Explore the dataset
# Observe that the scales and centers are very different which needs to be fixed,
# and the column "Serial No." is actually just the row number so it must be removed.
dataset  # To print the dataset (In a jupyter notebook there is no need to use the print command)
dataset.head(10)  # Or, you can print only the first 10 lines
dataset.describe()  # The basic statistical description of the dataset


# Remove the useless "Serial No."
dataset.drop('Serial No.', axis=1, inplace=True)

# Let's look at the correlations across the columns.
dataset.corr()  # Print the correlations

# We can see easier when we draw a correlation heatmap
f,ax = plt.subplots(figsize=(15, 15))
sns.heatmap(dataset.corr(), annot=True, linewidths=0.5, linecolor="white", fmt= '.1f',ax=ax)
plt.show()
# Which score looks like the most related to the admission chance?
```

Data preparation:

```
# Let's prepare the dataset for the experimentation
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Separate the targets from the inputs
X = dataset.iloc[:,:-1].values
Y = dataset.iloc[:,7].values.reshape(-1,1) # Chance of admission
# Convert to numpy arrays
X = np.array(X)
Y = np.array(Y)

# Split the training and validation sets (ensuring the same split always)
x_train, x_valid, y_train, y_valid = train_test_split(X, Y, test_size=0.30,
                                                      shuffle=False, random_state=1)

# Preprocessing (by standard normalization since we observed quite different scales/centers)
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)  # We fit the scaler only on the training set
x_valid = scaler.transform(x_valid)  # We use the resulting fit on the validation set (always!)
print(x_train.shape, x_valid.shape)
```