CS 486/686: Assignment 3 (125 Points)

Instructors: Jesse Hoey & Victor Zhong
Due Date: Wednesday March 12, 2025 at 11:59 pm (ET; Waterloo Time)

Instructions

- Submit written solutions in a file named `writeup.pdf` and code solutions in two files (named `neural_network.py` and `operations.py`, respectively) to the A3 Dropbox on `Learn` (`https://learn.uwaterloo.ca`). Unlike Assignment 1, please **do NOT zip your files**.
- No late assignment will be accepted.
- This assignment is to be done individually.
- Use the latest Python 3.12 to implement the code.
- Lead TAs:
  - Negar Arabzadeh Ghahyazi (narabzadehghahyazi)
  - Hala Sheta (hsheta)

  The TA office hours will be scheduled on Piazza.

1 Naïve Bayes Learning (45 Points)

In assignment 2, you learned a decision tree to classify text documents in two sets given a labeled training set. Here you will learn a Naïve Bayes classifier for the same data. The data is made from a subset of Reddit posts sourced from `https://files.pushshift.io/reddit/` and processed it using Google BigQuery. The dataset includes the first 1500 comments of August 2019 of each of the r/books and r/atheism subreddits, cleaned by removing punctuation and some offensive language, and limiting the words to only those used more than 3 times among all posts. These 3000 comments are split evenly into training and testing sets (with 1500 documents in each).

To simplify your implementation, these posts have been pre-processed and converted to the *bag of words* model. More precisely, each post is converted to a vector of binary values such that each entry indicates whether the document contains a specific word or not. Each line of the files `trainData.txt` and `testData.txt` are formatted "docId wordId" which indicates that word wordId is present in document docId. The files `trainLabel.txt` and `testLabel.txt` indicate the label/category (1=`atheism` or 2=`books`) for each document (docId = line#). The file `words.txt` indicates which word corresponds to each wordId (denoted by the line#).

Implement code to learn a naïve Bayes model by maximum likelihood[1]. More precisely, learn a Bayesian network where the root node is the label/category variable with one child variable per word feature. The word variables should be binary and represent whether that word is present or absent in the document. Learn the parameters of the model by maximizing the likelihood of the training set only. This will set the class probability to the fraction of documents in the training set from each category, and the probability of a word given a document category as the fraction of documents in that category that contain that word. You should use a Laplace correction by adding 1 to numerator and 2 to the denominator, in order to avoid situations where both classes have probability of 0. Classify documents by computing the label/category with the highest posterior probability $\Pr(label \mid \text{words in document})$. Report the training and testing accuracy (i.e., percentage of correctly classified articles).

What to hand in:

- **[10 points]** A printout of your code.
- **[10 points]** A printout listing the 10 most discriminative word features measured by

$$\max_{word} |\log \Pr(word \mid label_1) - \log \Pr(word \mid label_2)|$$

  Since the posterior of each label is formulated by multiplying by the conditional probability $\Pr(word \mid label_i)$, a word feature should be more discriminative when the ratio

[1] For the precise equations for this, see the note on the course webpage `https://cs.uwaterloo.ca/~jhoey/teaching/cs486/naivebayesml.pdf`

$\frac{\Pr(word|label_1)}{\Pr(word|label_2)}$ is very large or very small, and, therefore, when the absolute difference between $\log \Pr(word \mid label_1)$ and $\log \Pr(word \mid label_2)$ is large. In your opinion, are the printed words good features in terms of discriminating between the two classes?

- **[10 points]** Training and testing accuracy (i.e., two numbers indicating the percentage of correctly classified articles for the training and testing set).
- **[5 points]** The naïve Bayes model assumes that all word features are independent. Is this a reasonable assumption? Explain briefly.
- **[5 points]** What could you do to extend the Naïve Bayes model to take into account dependencies between words?
- **[5 points]** What if, instead of using ML learning, you were to use MAP learning. Explain what you would need to add and how it would work.

2 Neural Networks for Classification and Regression (80 Points)

In this part of the assignment, you will implement a feedforward neural network from scratch. Additionally, you will implement activation functions, a loss function, and a performance metric. Lastly, you will train a neural network model to perform a regression problem.

Red Wine Quality - A Regression Problem

The task is to predict the quality of red wine from northern Portugal, given some physical characteristics of the wine. The target $y \in [0, 10]$ is a continuous variable, where 10 is the best possible wine, according to human tasters. This dataset was downloaded from the UCI Machine Learning Repository. The features are all real-valued. They are listed below:

- Fixed acidity
- Volatile acidity
- Citric acid
- Residual sugar
- Chlorides
- Free sulfur dioxide
- Total sulfur dioxide
- Density
- pH
- Sulphates
- Alcohol

Training a Neural Network

In Lecture 8b, you learn how to train a neural network using the backpropagation algorithm. In this assignment, you will apply the forward and backward pass to the entire dataset simultaneously (i.e. batch gradient descent). As a result, your forward and backward passes will manipulate tensors, where the first dimension is the number of examples in the training set, n. When updating an individual weight $W_{i,j}^{(l)}$, you will need to find the sum of gradients $\frac{\partial \mathcal{L}}{\partial W_{i,j}^{(l)}}$ (where $\mathcal{L}$ is the Error) across all examples in the training set to apply the update. Algorithm 1

gives the training algorithm in terms of functions that you will implement in this assignment. Further details can be found in the documentation for each function in the provided source code.

Algorithm 1 Training

Require: $\eta > 0$ ▷ Learning rate
Require: $n_{epochs} \in \mathbb{N}^+$ ▷ Number of epochs
Require: $X \in \mathbb{R}^{n \times f}$ ▷ Training examples with n examples and f features
Require: $y \in \mathbb{R}^n$ ▷ Targets for training examples
  Initiate weight matrices $W^{(l)}$ randomly for each layer. ▷ Initialize `net`
  for $i \in \{1, 2, \ldots, n_{epochs}\}$ **do** ▷ Conduct n_{epochs} epochs
    `A_vals, Z_vals` $\leftarrow$ `net.forward_pass(`X`)` ▷ Forward pass
    $\hat{Y} \leftarrow$ `Z_vals[-1]` ▷ Predictions
    $L \leftarrow \mathcal{L}(\hat{Y}, Y)$
    Compute $\frac{\partial}{\partial \hat{Y}} \mathcal{L}(\hat{Y}, Y)$ ▷ Derivative of error with respect to predictions
    `deltas` $\leftarrow$ `backward_pass(A_vals,` $\frac{\partial}{\partial \hat{Y}} \mathcal{L}(\hat{Y}, Y)$ `)` ▷ Backward pass
    `update_gradients()` ▷ $W_{i,j}^{(\ell)} \leftarrow W_{i,j}^{(\ell)} - \eta \sum_n \frac{\partial \mathcal{L}}{\partial W_{i,j}^{(\ell)}}$ for each weight
  end for
  return trained weight matrices $W^{(\ell)}$

Activation and Loss Functions

You will implement the following activation functions and their derivatives:

- **Sigmoid**

$$g(x) = \frac{1}{1 + e^{-kx}}$$

- **ReLU**

$$g(x) = \max(0, x)$$

You will implement the following loss function and its derivative for regression:

- **Mean Squared Error**

$$\mathcal{L}(\hat{Y}, Y) = \frac{1}{n} \sum_{i=1}^{n} (\hat{Y} - Y)^2$$

To evaluate the performance of the model, you will implement the following performance metric:

- **Mean Absolute Error**

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|\hat{Y} - Y|$$

Implementation

We have provided three Python files. Please read the detailed comments in the provided files carefully. Note that some functions have already been implemented for you.

1. `neural_network.py`: Contains an implementation of a `NeuralNetwork` class. You must implement the `forward_pass()`, `backward_pass()`, and `update_weights()` methods in the `NeuralNetwork` class. **Do not change the function signatures. Do not change anything else in this file!**

2. `operations.py`: Contains classes for activation functions, a loss function, and a performance metric. The activation functions extend a base `Activation` class and the loss function extends a base `Loss` class. You must implement all the blank functions as indicated in this file. **Do not change the function signatures. Do not change anything else in this file!**

3. `train_experiment.py`: Provides a demonstration of how to define a `NeuralNetwork` object and train it on one of the provided datasets. Feel free to change this file as you desire.

Please complete the following tasks:

1. **[60 Points]** Implement the empty functions in `neural_network.py` and `operations.py`.

   Please do not invoke any numpy random operations in `neural_network.py` and `operations.py`. This may throw the automatic grading off.

   Unit tests for `neural_network.py`:

   - `NeuralNetwork.forward_pass()`
     (1 public test + 2 secret tests) * 4 marks = 12 marks
   - `NeuralNetwork.backward_pass()`
     (1 public test + 2 secret tests) * 4 marks = 12 marks
   - `NeuralNetwork.update_weights()`
     (1 public test + 2 secret tests) * 5 marks = 15 marks

   Unit tests for `operations.py`:

- `Sigmoid.value()`
  (1 public test + 2 secret tests) * 1 mark = 3 marks
- `Sigmoid.derivative()`
  (1 public test + 2 secret tests) * 1 mark = 3 marks
- `ReLU.value()`
  (1 public test + 2 secret tests) * 1 mark = 3 marks
- `ReLU.derivative()`
  (1 public test + 2 secret tests) * 1 mark = 3 marks
- `MeanSquaredError.value()`
  (1 public test + 2 secret tests) * 1 mark = 3 marks
- `MeanSquaredError.derivative()`
  (1 public test + 2 secret tests) * 1 mark = 3 marks
- `mean_absolute_error`
  (1 public test + 2 secret tests) * 1 mark = 3 marks

We have included the public tests in the provided files under `p2.zip/tests/`, where `nets.json` specifies the initial network weights used for forward, backward, and weight updating tests. Feel free to run the tests locally to ensure your implementation is correct (at least on public tests). The secret tests will be used for grading.

Once you have implemented the functions, you can try training neural networks on the provided dataset. The wine quality dataset is in `p2.zip/data/wine_quality.csv`.

In `train_experiment.py`, we have provided some code to instantiate a neural network and train on an entire dataset. Experiment with different numbers of layers with different sizes and activation functions. You do not need to submit the results of this experimentation.

2. **[20 points]** Execute k-fold cross validation for the wine quality dataset with $k = 5$. Use a single node with the Identify activation function as your output layer. Other layers are your choice. Report the sizes of the layers that you used in your neural network, along with the activation functions you used for your hidden layers. Train for 500 epochs in each trial and use $\eta = 0.001$.

   To perform cross validation, randomly split the data into 5 folds. For each fold, train the model on the remaining data and determine the trained model's mean absolute error on the fold. You can use `NeuralNetwork.evaluate()` to determine the mean absolute error on the validation set (i.e. fold).

   Produce a plot where the x-axis is the epoch number and the y-axis is the average training loss across all experiments for the current epoch. Report the average and standard deviation of the mean absolute error across all folds after training for 500 epochs.

For example, for your first fold, 80% of the examples should be in the training set and 20% of the examples should be in the validation set (i.e. fold 1). You will require the loss obtained after executing the forward pass for each of the 500 epochs. After your model has trained, use the trained model to calculate the mean absolute error on the validation set. This is one experiment. You will need to run this experiment 5 times in total, plotting the average loss at epoch i for each epoch.