CS/SE 4AL3 Homework 3:
Neural Networks for Breast Cancer Prediction

TAs: Hamed Hasibi, Parisa Salmani

Fall 2025
Due: November 7th

1 Overview

This assignment will get you started working with neural networks in PyTorch. You will build a model to predict if tumors are benign or malignant and will experiment with different forms of regularization. This assignment has four parts and will be graded out of 25 points. You must work on this assignment individually. A starter notebook file has been provided to you, your task is to complete it by filling in the required sections.

2 Model Specification (2 points)

For this assignment, you will design a simple feedforward neural network that can perform binary classification using PyTorch. The network should accept an input vector of size $input\_size$, which represents the number of features in your dataset. The first layer will be a fully connected (linear) layer with 32 neurons, followed by a ReLU activation function to introduce non-linearity. The output from this layer should then pass into a second fully connected layer with 16 neurons, again followed by a ReLU activation. Finally, the network should include an output layer consisting of a single neuron, followed by an activation function that converts the final output into a probability between 0 and 1.

3 Problem

Challenge: Breast cancer prediction is a widely studied and challenging problem in the medical community. Our goal is to predict breast cancer in patients using a small data set of 569 samples.

Dataset: (`https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html`).

Part 1: Early Stopped SGD (7 points)

Goal: Build the simple feedforward neural network using PyTorch using early stopped SGD. Early stopping limits the number of iterations while training the model. It is implemented as follows:

1. Execute gradient descent for 1 iteration
2. Compute loss using the model's loss function
3. Compare the loss for the current and previous iterations
4. If the difference is beyond a threshold, repeat, else stop

For this part of the assignment, implement early stopping for the given classifier. The function `stochastic_gradient_descent()` must print the number of iterations your early stopping algorithm takes to achieve optimal performance. The output of this part should be the loss function plotted every 1/10th of the epoch (e.g. if your epoch size is 5000, plot the loss after every 500 iterations) for both the training and validation loss. You should also output the accuracy of your model on a held-out validation set that is 20% the size of the total data.

A successful early-stopping algorithm will reach the lowest possible train error before it plateaus or validation loss increases. This means that your training function must also plot your train and validation loss well beyond your stopping iterations to verify whether your early stopping algorithm is yielding appropriate performance. You must also carefully select the loss threshold that aligns with your loss function. This number is typically of the magnitude of 10^{-2} or 10^{-3}, but depends on how you compute the loss. You may also increase the window of comparison to improve your algorithm, i.e. instead of comparing with the previous iteration, you may compare with the 2nd previous, or some aggregate of previous iterations.

Part 2: MiniBatch SGD (6 points)

Goal: Build the simple feedforward neural network to reliably classify tumors using minibatch SGD. Instead of training 1 sample at a time, you should compute the gradient over a small batch size of n samples. You are free to select n as a hyperparameter. Remember that in minibatch SGD, the loss is the average of the samples in that batch. To optimize the performance of your minibatch, you may reconfigure other model hyperparameters if needed (this includes layer size, number of neurons, and activation functions).

Show the performance metrics for minibatch SGD compared with early stopped SGD (defined in Part 1), in terms of accuracy, precision, recall (you must implement these yourself), and plot the train and validation losses. The output should be the loss function plotted every 1/10th of an epoch, showing the training and validation loss curves for early stopped SGD and minibatches. Also show the final accuracy, precision, and recall on the held-out validation set. Use the same splits as you did for part 1.

Part 3: Dropouts (5 points)

Goal: In the next two sections, you will extend your neural network from Parts 1 and 2 to explore different techniques to prevent overfitting.

To make your neural network more robust and less prone to overfitting, you will add dropout layers between the hidden layers. Dropout randomly "turns off" a fraction of neurons during each training iteration, forcing the network to learn redundant, generalized patterns rather than memorizing the training data.

Modify the previous neural network class to include dropout layers between the hidden layer and other layers. You are asked to experiment with different dropout rates (0.1, 0.3, 0.5) and train your model on the same dataset as before and record how the training and validation accuracy change by plotting for different dropout rates, using the minibatch SGD of Part 2. Then, you should explain how dropout affects your model performance and generalization.

Part 4: L1/L2 Regularization (5 points)

Another effective way to prevent overfitting in neural networks is to use weight regularization, also known as L1/L2 penalties. While dropout works by randomly deactivating neurons during training, L1 and L2 regularization work by penalizing large weights in the loss function, encouraging the model to keep parameters small and generalizable.

Add L2 regularization by setting the *weight_decay* (λ) parameter in your optimizer. Train your model with different λ values (e.g.; 0, 10^{-4}, 10^{-3}) and observe how training and validation losses differ. As an extension, use L1 regularization (manually choosing the hyperparameter is fine) for $\lambda = 10^{-4}$ and compare its effect on model sparsity. You need to explain how L1 and L2 regularization affect the model's loss curves, and which regularization type yields better generalization for your dataset.

4 Deliverables

- Please submit your work as **a single Jupyter Notebook**. The starter Jupyter Notebook has been provided. Your code should be well-commented and described. There is no restriction on the notebook file name.
- **Please do not alter** the notebook structure. You should only fill in the blocks of the code that say "#CODE HERE" and explain in text blocks noted "#EXPLAIN HERE".
- You should include any disclaimers about the use of AI in the first text block of the Jupyter notebook, as dedicated in the notebook, citing which parts of code the AI have been used.
- The Jupyter Notebook should include all the **codes already executed and saved, before submission**. TAs won't run your code blocks. Hence, ensure that all code blocks are executed by clicking "Run All". Failure to provide a previously run Jupyter notebook will result in grade deductions.
- You are **only** permitted to use the libraries provided in the Jupyter notebook. You will receive a zero if any other libraries are used.
- Note that scikit learn is available for this assignment. You may add import statements within these packages for your convenience, e.g. `from sklearn.PACKAGE import CLASS`.
- You need to use Python version 3.12. Do not upload any datasets.

5 Help

Please use the Teams channel and tutorials to ask questions about the assignment when you need guidance or pointers on this homework. You are free to discuss your approach and ideas with classmates, but should not share code or reuse data. You may use generative AI tools if you find them helpful, but please clearly document how they were used as described above and follow the guidelines in the syllabus for what you **must** include when using generative AI. If you use generative AI and do not report it, you may receive a 0 for the assignment. You take full responsibility for the deliverables you submit.

You are welcome to use code snippets from examples in class, things you find online, or from AI code generation tools. Just make sure to give proper attribution to code you did not write. Follow the syllabus instructions for how to report the use of AI tools. However, you may not copy code that does the entire assignment (e.g. someone who did this assignment in a previous semester).