Part 1: Binary Classification Logistic Regression - Sigmoid Activation Implementation

Application

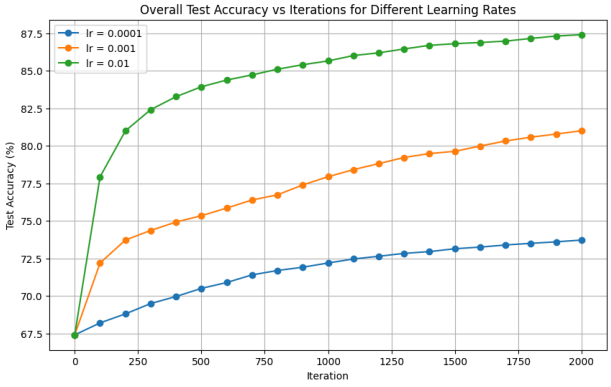
The MNIST dataset consists of 10-digit classes (0-9), but logistic regression is inherently a binary classifier that distinguishes between two categories. So, to adapt logistic regression for multi-classification, I am going to use the One-vs-All strategy, where I train 10 separate binary classifiers, each responsible for detecting a single digit. Each classifier will predict whether an input image belongs to a specific digit or not. For example, one classifier will determine whether an image is a '0' or 'not 0', another will determine '1' or 'not 1', and so on for all 10 digits. At prediction time, the test image will be passed through all 10 classifiers, and the digit with the highest probability will be chosen as the predicted class.

Methodology

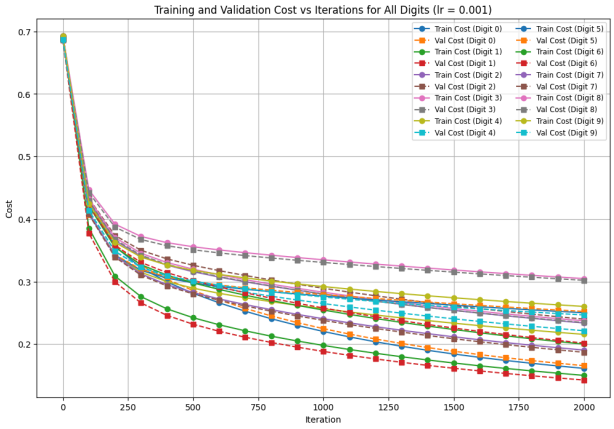
To prepare the data for training, I am going to first load the MNIST dataset using TensorFlow. Since pixel values range from 0 to 255, I will normalize them by dividing by 255 to scale the values between 0 and 1, which helps improve training stability. Next, I will reshape each 28x28 image into a 784-dimensional vector, as logistic regression operates on flattened feature vectors rather than images. Before splitting the data, I will randomly shuffle the training samples to ensure that both the training and validation sets are representative of the overall dataset. I will also split the dataset, reserving 10% of the training data as a validation set to monitor generalization performance. For each classifier, I will convert the validation labels into binary labels to accurately compute the validation cost and plot digit 0 for analysis. For demonstration in the plots, I specifically show the training and validation cost curves for the "digit 0" classifier, but the same process is repeated for digits 1–9. I also show the training and validation cost plot for all digits under the 0.001 learning rate. For training, as mentioned, I am going to implement the One-vs-All approach, which involves training 10 separate binary logistic regression models–one for each digit. Each model will learn to classify whether a given image belongs to its respective digit class or not. To optimize the model, I will use binary cross-entropy loss, which measures the difference between the predicted probabilities and the true binary labels. I will apply batch gradient descent to minimize the loss, ensuring that the model learns the correct decision boundaries. Additionally, I am going to experiment with three different learning rates (0.0001, 0.001, 0.01) to analyze their impact on convergence speed and accuracy. To evaluate performance, I will track test accuracy every 100 iterations. After training is completed, I will use the validation set metrics solely to view how well each digit's classifier generalizes. For final testing, I will classify each test image by passing it through all 10 classifiers. The digit with the highest probability will be selected as the predicted label. This method ensures that even though each classifier makes binary decisions, the final model is capable of handling multi-class classification. In addition, I will track and plot both training and validation costs at each 100 iterations, which will, in turn, allow me to detect potential overfitting or underfitting. Finally, in terms of evaluating performance, I will analyze and plot the training accuracy curve to determine how well the model is learning from the training data over time.

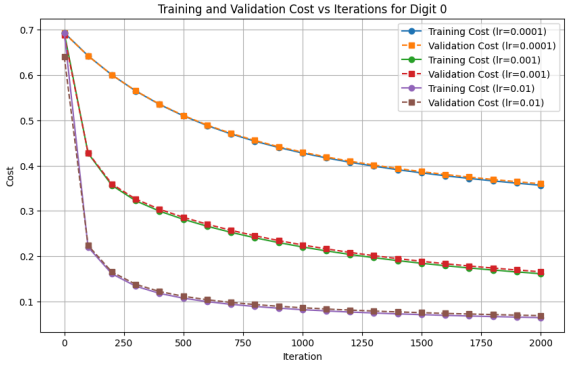
	Pros	Cons
Training	Each classifier learns a simple binary task (overall easier to optimize)	Requires training 10 separate models, increasing overall training time
Parallelization	Models can be trained independently in parallel, speeding up training	If trained sequentially, total training time increases significantly
Prediction Time	Each classifier runs a straightforward probability computation	Each test sample requires 10 model evaluations, making overall prediction slower

Results and Analysis + Bonus

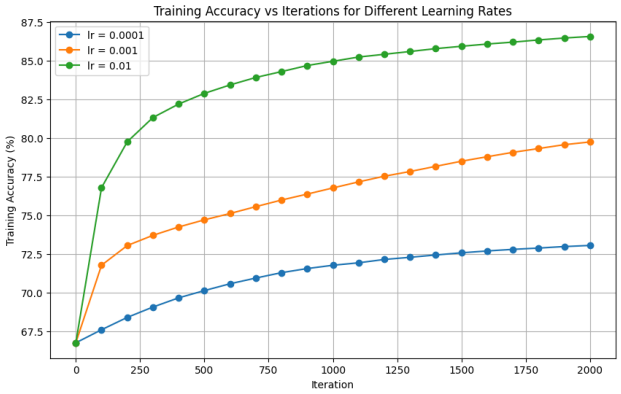


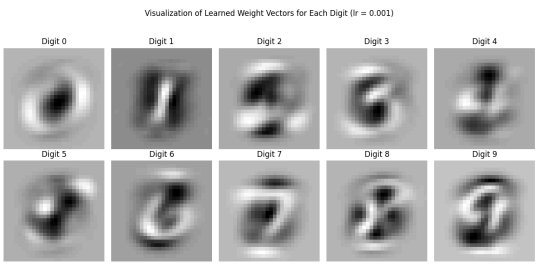
From the Test Accuracy vs Iterations for Different Learning Rates plot, it is coherent that the highest learning rate (0.01) improves test accuracy the fastest, surpassing 80% accuracy in fewer than 250 iterations and continuing to climb toward around 87.41% by iteration 2000. The moderate rate (0.001) also converges effectively, steadily reaching 81.01% accuracy over the same period. In contrast, the lowest rate (0.0001) rises more gradually, moving from 67.41% to 73.73% by the final iteration. These trends illustrate the impact of learning rate on convergence speed and final performance: higher rates can learn more quickly and achieve better accuracy within a fixed number of iterations, provided they remain stable and do not overshoot.





In the cost plot for digit 0, the highest learning rate (0.01) rapidly decreases both training and validation costs. The moderate rate (0.001) converges more steadily, while the lowest rate (0.0001) decreases very gradually, reflecting slower learning. These differences carry over to the training accuracy curves: the model trained with the highest rate achieves the fastest rise in accuracy, ultimately reaching the highest final performance, while the moderate rate also shows a strong but more controlled improvement. In contrast, the lowest rate lags behind both in speed and final accuracy. It is also evident from the different digits with Training and Validation sets that there is proper convergence. The slight gap is okay since the Validation is only 10% of the data.





Bonus:

Each visualization picture shows the weight vector for one digit, reshaped into a 28x28 grid to match the input image size. As training progresses, the model learns which parts of the image matter most for identifying each digit. For example, the weight image for digit '0' highlights the circular shape, while the weight image for digit '1' emphasizes the vertical line. Bright or dark spots in the weight image mean those pixels strongly influence the decision to label an image as that digit or not. This shows that the algorithm is picking out the main features of each digit's shape to tell them apart.

Part 2: Neural Network with Softmax Activation Implementation

Application

The MNIST dataset consists of 10 digital classes (0-9), and unlike logistic regression, which is inherently binary, a neural network with a softmax output layer is naturally suited for multi-classification. Instead of training 10 separate binary classifiers using the One-vs-All approach like before, I will construct a single neural network that takes an input image and directly outputs probabilities for all 10 digits. The softmax activation function ensures that the sum of all output probabilities equals 1, allowing the model to make a definitive classification decision by selecting the class with the highest probability. This approach streamlines training by consolidating the decision-making process into a single model.

Methodology

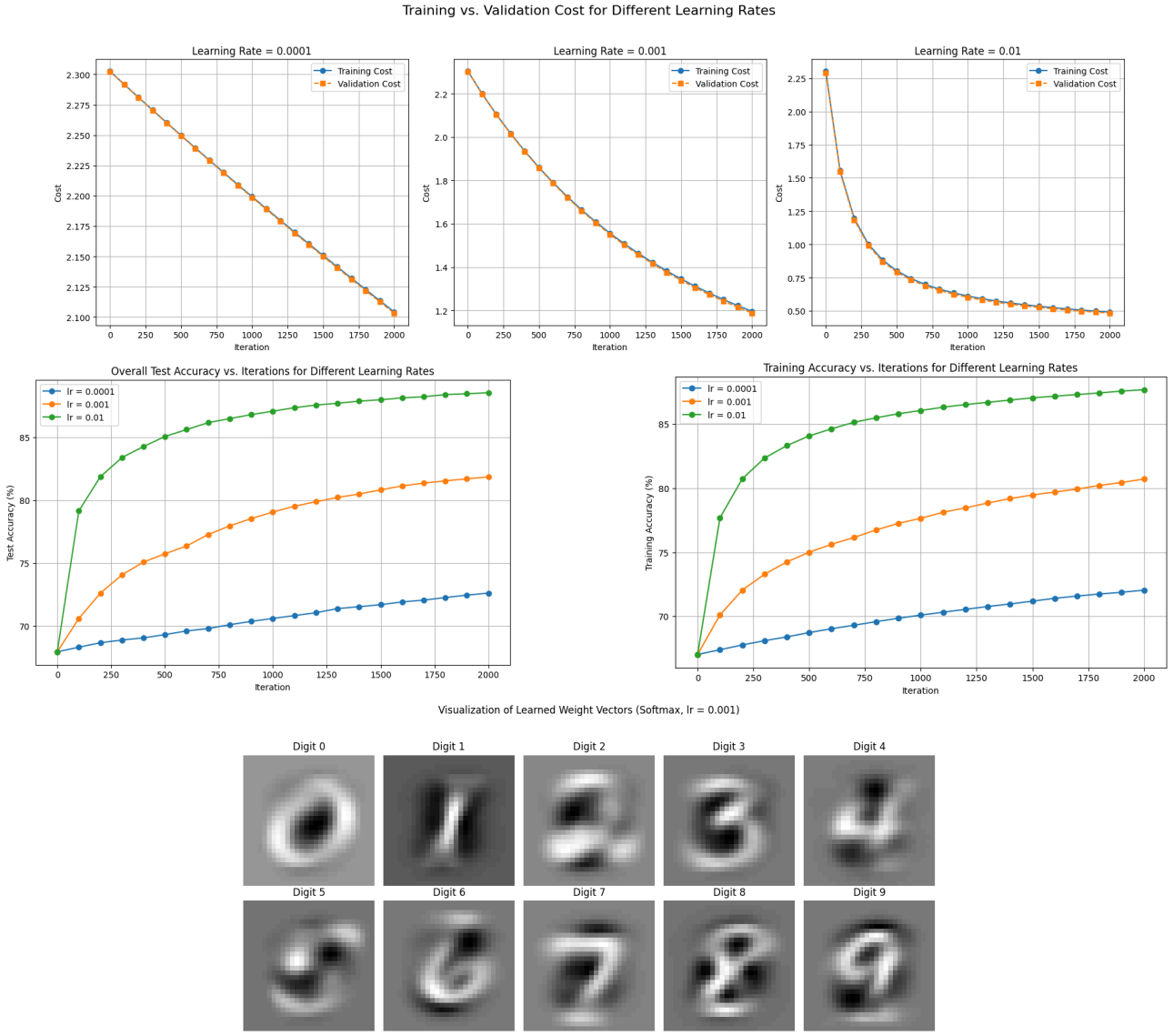
To prepare the dataset, I will first load MNIST images and labels using TensorFlow. Since pixel values range from 0 to 255, I will normalize them by dividing by 255, ensuring numerical stability and faster convergence. Each 28x28 image will be flattened into a 784-dimensional vector, allowing it to be processed by the model. The dataset will be shuffled to improve generalization, and I will split it into training and validation sets, reserving 10% of the training data for validation. The model is a single-layer neural network that directly maps the input features to output probabilities using the softmax activation function. The output layer consists of 10 neurons, each representing a digit class (0-9). The softmax function ensures that the sum of all outputs is 1, allowing the model to assign a probability to each digit and choose the most likely one. For training, I will use categorical cross-entropy loss. To minimize this loss, I will apply batch gradient descent, experimenting with three different learning rates (0.0001, 0.001, 0.01) to analyze their effects on training and speed accuracy.

To evaluate performance, I will:

- Track training and validation loss at every 100 iterations to detect overfitting or underfitting
- Monitor training accuracy to measure how well the model is learning
- Evaluate the model on unseen test images, selecting the digit with the highest softmax probability as the predicted class.

For plots, the first plot illustrates the overall test accuracy against iterations, allowing to observe how quickly and effectively the model converges to high accuracy for each learning rate. The second set of subplots compares training and validation costs over iterations for each learning rate, which helps in identifying potentially major overfitting or underfitting between the curves. The third plot offers a visualization of the learned weight vectors for each digit class (0.001), where each weight vector is reshaped into a 28x28 image to provide insight into what features the model is capturing. Finally, the training accuracy plot displays how the model's accuracy on the training data improves over time for each learning rate.

	Pros	Cons
Training	Trains a single model for all digits much faster	Can be less accurate than deeper networks
Parallelization	Fast classification in one step	Once again, it is not as fast as more advanced models, but a way more efficient way of doing it than Sigmoid.



Running the Softmax took 9.25 minutes, and running the Sigmoid took 25.5 minutes, so the Softmax implementation won on converging time. Also, there was a very minimal difference between the Softmax and the Sigmoid accuracies and learning rates, and Sigmoid was a lot easier to implement and train.

From the Training vs. Validation Cost plot for Different Learning Rates plot, it is evident that there's no underfitting or overfitting in the neural network and that the 0.0001 learning rate takes much much longer to converge and is a linear line while the 0.001 learning rate achieves a good balance between speed and stability and the 0.01 learning rate converges much quicker. This displays stable convergence in the model. The Overall Test Accuracy vs. Iterations for Different Learning Rates plot displays the best learning rate, 0.01, because it provides the highest accuracy and fastest convergence. The learning rate of 0.001 is also reasonable, but it requires more iterations to reach a lower final accuracy. The learning rate of 0.0001 is too slow, and it would require far more iterations to reach competitive accuracy. For the Training Accuracy vs. Iterations for Different Learning Rates, it is evident that the model learning rate of 0.01 is the best choice. The Bonus/Visualization of Learned Weight vectors confirms that the model correctly learns class-specific patterns.