FAST NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

REPORT

Name:	**Muhammad Saif (22F-3165)** **Sitara Shabbir (22F-3838)**
Assignment:	**5**
Instructor:	**Dr. Hashim Yasin**
Course:	**Machine Learning**

DELIVERABLES:

- Code report (PDF)
- Q1.ipynb
- Q2(a).ipynb
- Q2(b).ipynb
- Q3.ipynb

All codes are properly commented in the notebook files for better understanding of the implementation.

QUESTION 1:

Part (a)

Support vectors are basically the data points that sit closest to the decision boundary (imagine a line or plane that separates different classes of data). They're super important because they're the ones that help define where that boundary should be.

For linearly separable data (data that can be split by a straight line), here's how we get to SVM's objective function:

1. We want to find the widest possible street (margin) between two classes
2. The objective function looks like this:
   minimize: $(1/2)||w||^2$
   subject to: $y_i(w \cdot x_i + b) \geq 1$

where w is the normal vector to the boundary, b is the bias term, and yi are the class labels.

DIFFERENCE:

HARD MARGIN	SOFT MARGIN
Only works with linearly separable data	Works with both separable and non-separable data
No misclassifications allowed	Allows some misclassifications
More sensitive to outliers	More robust to outliers
Less practical in real-world scenarios	More practical for real applications

Part (b)

SVM with Linear Kernel:

In this part, we implemented a Support Vector Machine (SVM) model using a linear kernel to classify the image data.

The steps taken are as follows:

1. **Model Creation and Training:**
    - We created an SVM classifier (svm_clf_linear) using the linear kernel. This kernel is appropriate for linearly separable data, where the decision boundary is a straight line.
    - The model was trained on the training data (train_images_train and train_labels_train).
2. **Training Accuracy Report**
   The model performed exceptionally well on the training data, achieving a perfect 100% accuracy. This means it correctly classified all training images without any mistakes. However, on the test data, the accuracy dropped to 90.35%, which is still quite impressive for unscaled image data.

The model was trained on a subset of the dataset as training the model on complete 60000 images take extensive computational time, and this might cause our model to give 100% accuracy.

We also trained the model on complete training set and accuracy was reported 98.42%. But it took much longer than expected and computational time was around 19 minutes.

Part (c)

SVM with Linear Kernel (with Scaling):

In this version, we enhanced our previous SVM approach by adding StandardScaler preprocessing to our image data. The StandardScaler standardizes features by removing the mean and scaling to unit variance, which is often beneficial for SVM performance.

Training Accuracy Report:

After implementing scaling, here's what we found:

- Training accuracy: 100%
- Testing accuracy: 90.65% (a slight improvement from 90.35% in the unscaled version)

Comparing this to our previous unscaled model:

- The testing accuracy improved slightly by 0.3% (from 90.35% to 90.65%)

Utilizing Model Size:

We trained the model on a subset of data because training through 60000 images takes extensive computational time. Using this smaller dataset might explain our perfect 100% training accuracy, as the model can more easily memorize fewer examples. Despite this, the scaled version showed promising results with a test accuracy of 90.65%, slightly better than the unscaled version's 90.35%. This improvement, though modest, suggests that scaling the data helps the model perform better on unseen examples, even with a reduced dataset size

Part (d)

SVM with RBF Kernel:

We experimented with an SVM classifier using a Radial Basis Function (RBF) kernel, which is better suited for handling non-linear data patterns.

Training Accuracy Report:

- Working with the same subset of the MNIST dataset, the RBF kernel with a C value of 0.1 achieved a training accuracy of 93.96% and a test accuracy of 90.90%.
- Unlike the linear kernel, we no longer see perfect training accuracy, suggesting less overfitting.
- The smaller gap between training and testing accuracy (about 3%) indicates better generalization.
- This RBF implementation slightly outperformed both our scaled and unscaled linear SVM models on the test set, making it potentially the most robust choice among our SVM implementations.

Part (e)

REPORTING TESTING ACCURACIES:

Let me break down how our different SVM models performed on the test data:

Looking at the numbers, we got:

- Basic linear SVM (no scaling): 90.35%
- Linear SVM with scaling: 90.65%
- SVM with RBF kernel: 90.90%

The RBF kernel came out slightly ahead with a 90.90% test accuracy, though the difference isn't huge. The scaled linear SVM did better than the unscaled version, which makes sense since scaling helps the SVM algorithm work better with the data.

What's interesting is how close all three results are - they're all hovering around 90%. This tells us that even the simplest approach (unscaled linear SVM) did a decent job with our digit classification task. The improvements from scaling and using the RBF kernel were helpful but modest, suggesting our data might not be too complex to handle.

The RBF kernel's slight edge might make it the best choice here, but considering the small differences, the simpler linear models could also be viable depending on our computational constraints.

QUESTION 2:

Part (a)

Designing a Neural Network Architecture:

Let me explain the architecture choices for our MNIST digit classification network:

Input Layer:

- Shape (784,): This comes from flattening our 28x28 pixel images (28 * 28 = 784)
- We normalize pixel values by dividing by 255 to get values between 0-1, helping with training stability

Hidden Layers:

1. First Hidden Layer (128 neurons):

- Chose 128 neurons as a good starting point - enough capacity to learn important features but not too large to cause overfitting
- ReLU activation: Used because it helps prevent vanishing gradients and typically trains faster than sigmoid/tanh

2. Second Hidden Layer (64 neurons):

- Reduced to 64 neurons to create a bottleneck architecture, forcing the network to learn more compact representations
- ReLU activation again for consistency and good performance

Output Layer:

- 10 neurons: One for each digit (0-9)
- Softmax activation: Perfect for multi-class classification as it gives us probability distribution across all classes

Results Analysis:

- The neural network demonstrated strong performance on the MNIST dataset, improving from 86.32% to 99.36% training accuracy across 10 epochs.
- The model achieved a final test accuracy of 97.40%, which is notably close to the validation accuracy of 97.42%.
- These results, along with the steady decrease in loss values, indicate that our model successfully learned to recognize digits without overfitting.
- The performance is competitive with our SVM implementations while being more straightforward to train.

Part (b)

Architecture Design of the Convolutional Neural Network

1. **First Convolutional Layer**
    - 32 filters with 3x3 kernel size
    - ReLU activation to introduce non-linearity
    - Input shape: 28x28x1 (grayscale images)
2. **Second Convolutional Layer**
    - 64 filters with 3x3 kernel size
    - ReLU activation for better feature extraction
3. **Max Pooling Layer (2x2)**
    - Reduces spatial dimensions
    - Helps in reducing computation and preventing overfitting
4. **Dropout Layer (25%)**
    - Prevents overfitting by randomly deactivating 25% of neurons
    - Improves model generalization
5. **Flatten Layer**
    - Converts 2D feature maps to 1D vector for dense layers
6. **Dense Layer**
    - 128 neurons with ReLU activation
    - Learns high-level feature combinations

7. **Output Layer**
    - 10 neurons (one per class) with softmax activation
    - Provides probability distribution across classes

Training Configuration

- Optimizer: Adam (adaptive learning rate)
- Loss function: Categorical cross entropy
- Batch size: 32
- Epochs: 3
- Validation split: 20%

Results Analysis

- Training accuracy improved steadily:
    - Epoch 1: 91.86%
    - Epoch 2: 98.57%
    - Epoch 3: 99.07%
- Final test accuracy: 98.93%
- Model showed consistent performance between training and validation, indicating good generalization
- Training completed in reasonable time (54s, 53s, 347s per epoch)

QUESTION 3:

Implementation of Neural Network for Boolean Function

Part (a)

Neural Network Architecture

- Input Layer: 5 features (Boolean inputs normalized to the range [0, 1]).
- Hidden Layers:
    - First hidden layer with 8 neurons and various activation functions.
    - Second hidden layer with 4 neurons and various activation functions.
- Output Layer: Single neuron for binary classification using different activation functions.
- Optimizer: Adam with a learning rate of 0.01.
- Loss Function: Binary Cross-Entropy.
- Epochs: 50
- Batch Size: 5

Part (b)

Justification

- Two hidden layers were chosen to capture complex relationships in the Boolean dataset.
- The number of neurons in each layer was experimentally optimized for the dataset size and complexity.

Part (c)

Activation Function Analysis

Different combinations of activation functions were applied to the hidden and output layers, and their performance was evaluated.

Why This Combination Was Used

The combination of ReLU in hidden layers and Sigmoid in the output layer was selected because it consistently provided the highest accuracy during testing (1.0000). ReLU accelerates training by avoiding saturation, and Sigmoid ensures the output is a valid probability for classification.

Hidden Activation	Output Activation	Accuracy
Sigmoid	Sigmoid	0.7333
Tanh	Sigmoid	0.9333
ReLU	Sigmoid	1.0000
ReLU	Softmax	0.6000

Best Combination

ReLU for the hidden layers and Sigmoid for the output layer achieved the highest accuracy of **1.0000**.

Reasoning

ReLU effectively handles non-linearities in the data, while Sigmoid is well-suited for binary classification tasks. The use of Softmax was unsuitable for the output layer as the target shape did not align with its intended purpose.