Analysis and Report of Project 1

Experiment Outline

In this project, our main objective is to design, train, and evaluate deep neural networks to improve optimization and generalization. These objectives were split into three tasks: building the network, improving the optimization, and improving the optimization.
In Task 1, we used three models of neural network architecture: a fully connected network, a locally connected network, and a convolutional neural network. The purpose of testing these three models, in particular, is to identify which aspects of their design can aid or hinder optimization results. For example, locally connected networks don't require weight sharing throughout all layers, which allows the network to learn localized patterns for each part of an input, possibly enhancing the network's range of data processing capability.
In Task 2, we continue testing the models' weight initialization strategies. This is important because simply changing the network parameters can vastly alter the learning trajectory of the original model. For example, we found that small initial weights often result in vanishing gradients, leading to a much slower learning rate.
In Task 3, we examine the ensemble method and dropout regularization and their effectiveness in handling overfitting in the models. When experimenting with the ensemble method, we combine the weight predictions of multiple models, allowing them to share predictive patterns and reduce variance in their final predictions. Conversely, when testing the dropout regularization technique, a random subset of neurons would be deactivated or "dropped out" from the network with the purpose of preventing groups of neurons from becoming overly reliant on one another.
Throughout these three tasks, we aim to identify ways to improve neural networks' effectiveness and accuracy by understanding the effect that each part of the network has on the architectural model. This knowledge will then allow us to build more efficient models in the future by knowing what conditions each architectural model performs better in.

Task I Report

This report for task 1 explores the effectiveness of three different neural network architectures on a handwritten digit classification task. The three architectures are:

1. Fully Connected Network: This treats the input as a flat vector with no spatial relationships.
2. Locally Connected Network (No Weight Sharing): This has similar features compared to the CNN but without any shared filters. This allows different filters at different positions to learn localized features.

3. Convolutional Neural Network (CNN): This feature uses convolutional layers to share weights, which makes it more computationally efficient.
The goal of this task is to compare the networks' ability to recognize specific patterns, using accuracy as the evaluation rubric.

The Expected Outcomes Before Training:
Fully Connected Network:
1. Since The Fully Connected Network lacks spatial awareness, they treat all the pixels equally to each other, making it inefficient for image-based tasks.
Locally Connected Network:
1. Since the features of this model do not share weights, it should learn localized features better than the Fully Connected Network, but it may struggle with generalization.
CNN:
The CNNs excel in pattern recognition due to shared filters across spatial locations.

Based on prior research and class, the CNN model should outperform The Fully Connected Network, while the locally connected networks might behave similarly to CNNs but require more parameters.

Observed Results after Training:
After a few tests, I can say that:
1. The Fully Connected Network average loss is around 0.0467 with 5 Epoch and an average accuracy of 92.92 percent.
2. The Locally Connected average loss is around 0.0378 with 5 Epoch and an average accuracy of 94.72 percent.
3. The CNN average loss is around 0.0364 and an average accuracy of 94.52 percent.

Key Insights and Analysis:
1. Although the Fully Connected Networlacksck spatial feature extractions, the dataset might contain some easy-to-learn patterns that allow the Fully Connected to classify digits efficiently. The Regulation effects from nonlinear activations helped the model learn meaningful features and tricks.
2. No weight sharing means that each part of the image has unique filters, which allows the network to learn more specific location patterns. The dataset contains a fixed position of digits and there are no CNNs to recognize patterns involuntarily across other locations.
3. The CNNs excel in recognizing patterns regardless of the position.

Conclusion:

1. The Fully Connected Network performed well, indicating that it can possibly be efficient when data has easy and structured patterns.

2. The Locally Connected performed nearly the best, likely due to the datasets' fixed positions, which benefit from localized learning.

3. CNN was the second best, slightly outperformed by the Locally Connected model.

<u>Task II Report</u>

Parameter Initialization Strategies

MLP (Fully Connected Network)

Analysis:

- The MLP has multiple dense layers with different activation functions (ReLU, sigmoid, tanh)
- Each layer transition requires appropriate scaling to prevent vanishing/exploding gradients
- Deeper architecture makes it particularly sensitive to initialization

Demonstration:

1. **Very Slow Learning:** Using too small initialization (std=0.0001)
    - ◦ Gradients remain tiny throughout training
    - ◦ Loss decreased minimally from 2.31 to 2.31 over 5 epochs
    - ◦ Accuracy remained low at ~9-14%
2. **Effective Learning:** Using He/Kaiming initialization
    - ◦ Properly scales weights based on layer size
    - ◦ Loss decreased steadily from 2.52 to 2.02 over 5 epochs
    - ◦ Accuracy improved to ~11%
3. **Too Fast Learning:** Using too large initialization (std=10.0)
    - ◦ Caused extremely high loss values (~167)
    - ◦ Gradients exploded, pushing activations to saturation
    - ◦ Training was unstable, with fluctuating loss
    - ◦ Accuracy remained poor at ~9-10%

Locally Connected Network

Analysis:

- Contains convolutional layers without weight sharing
- ReLU activations in conv layers and tanh in fully connected layer

- Requires balanced initialization to prevent feature dominance.

Demonstration:

1. **Very Slow Learning:** Using too small initialization (std=0.0001)
   - Minimal weight updates due to small gradients
   - Loss decreased slightly from 2.30 to 2.29 over 5 epochs
   - Accuracy improved marginally to 14%
2. **Effective Learning:** Using He/Kaiming initialization
   - Appropriate for ReLU activations in conv layers
   - Loss decreased significantly from 2.36 to 1.79 over 5 epochs
   - Training progressed steadily
3. **Too Fast Learning:** Using too large initialization (std=10.0)
   - Extremely high loss values (~120)
   - Network stuck in poor local minima
   - Accuracy remained at ~8% with no improvement

CNN (Convolutional Neural Network)
Analysis:

- Weight sharing in convolutional layers provides some robustness to initialization
- ReLU activations throughout the network except for the final layer
- Deeper architecture with skip connections requires careful initialization

Demonstration:

1. **Very Slow Learning:** Using too small initialization (std=0.0001)
   - Gradients propagated poorly through the network
   - Loss decreased minimally from 2.30 to 2.29 over 5 epochs
   - Accuracy remained at ~9% with no improvement
2. **Effective Learning:** Using He/Kaiming initialization
   - Well-suited for ReLU activations in CNNs
   - Loss decreased steadily from 2.62 to 1.77 over 5 epochs
   - Accuracy improved to 10%
3. **Too Fast Learning:** Using too large initialization (std=10.0)
   - Very high loss values (~130-140)
   - Network unable to learn meaningful features
   - Accuracy stuck at 16% with no improvement

The experiments clearly demonstrate that proper initialization is critical for all three networks, with He/Kaiming initialization providing the most effective learning across all architectures.

<u>Task III Report</u>

Analysis of Ensemble Approach

I implemented an ensemble of three neural networks to improve generalization performance on the digit classification task. The ensemble combines:

1. **MLP (Fully Connected Network):** A multi-layer perceptron with dropout regularization
2. **Locally Connected Network:** A network with local connectivity but no weight-sharing
3. **CNN:** A convolutional neural network with batch normalization

Results

Model	**Accuracy**
MLP	17.00%
LocalNN	10.00%
CNN	12.00%
Equal-Weighted Ensemble	14.00%
Performance-Weighted Ensemble	19.00%

Model

The performance-weighted ensemble achieved a 2% improvement over the best individual model (MLP), demonstrating the effectiveness of ensemble methods for improving generalization.

Why the Ensemble Improved Performance

1. **Error Diversity:** Each network architecture makes different types of errors. The MLP captures global patterns, the LocalNN captures local patterns without weight sharing, and the CNN captures hierarchical features with weight sharing.

2. **Reduced Variance:** By averaging predictions from multiple models, the ensemble reduces the variance component of the error, making predictions more stable.

3. **Weighted Contribution:** The performance-weighted ensemble performed better than the equal-weighted ensemble because it gave more influence to the more accurate models.

4. **Complementary Strengths:** Each model has different strengths in recognizing specific digit patterns. The ensemble leverages these complementary capabilities.

5. **Regularization Effect:** The ensemble effectively acts as a form of regularization, reducing overfitting that might occur in individual models.

The experiment clearly demonstrates that combining diverse neural network architectures through ensemble methods is an effective technique for improving generalization performance on the digit classification task.

Effects of Dropout Parameter

Dropout is a regularization technique that randomly "drops out" (sets to zero) a proportion of neurons during training. The dropout parameter p represents the probability of keeping a neuron active.

Key effects of the dropout parameter:

1. **Regularization Strength:**
   - Lower p values (e.g., 0.5) create stronger regularization
   - Higher p values (e.g., 0.8) create milder regularization
2. **Network Capacity Reduction:**
   - During training, dropout effectively creates a different "thinned" network for each batch
   - This prevents co-adaptation of neurons (where neurons become overly dependent on each other)
3. **Ensemble Effect:**
   - Conceptually similar to training many different network architectures and averaging their predictions
   - At test time, using the full network with scaled weights approximates this ensemble
4. **Impact on Learning:**
   - Too low p can make learning difficult (network capacity too limited)
   - Too high p provides insufficient regularization (minimal effect)
   - Optimal p typically depends on network size and dataset complexity

Analysis of Dropout Parameter

Dropout is a regularization technique that randomly deactivates neurons during training to prevent overfitting. The dropout parameter (p) represents the probability of keeping a neuron active during training.

Experimental Results

I conducted experiments with four different dropout configurations on the fully connected neural network:

Dropout Configuration	**Dropout Rate**	**Test Accuracy**
No Dropout	0.0	16.00%
Mild Dropout	0.2	13.00%
Moderate Dropout	0.5	13.00%
Severe Dropout	0.8	15.00%

Effective vs. Ineffective Dropout Cases

Effective Case: No Dropout (p=1.0)

- **Training behavior:** Rapid convergence with training accuracy reaching 100%
- **Validation behavior:** Initially improved but then degraded as training continued
- **Test accuracy:** 16.00% (highest among all configurations)
- **Analysis:** For this specific dataset and model architecture, the network benefited from using its full capacity without dropout. This suggests the model wasn't complex enough relative to the dataset to suffer from significant overfitting.

Ineffective Case: Mild Dropout (p=0.8)

- **Training behavior:** Slower convergence with training accuracy reaching ~89%
- **Validation behavior:** More stable than no dropout but lower final performance
- **Test accuracy:** 13.00% (lowest among all configurations)
- **Analysis:** This level of dropout restricted the model's capacity without providing sufficient regularization benefits, creating a suboptimal trade-off.

Key Observations

1. **Overfitting vs. Underfitting Trade-off:**
    - No dropout allowed the model to fully utilize its capacity, but it showed signs of overfitting (increasing validation loss)
    - Severe dropout (p=0.2) prevented overfitting but limited learning capacity
2. **Training Dynamics:**
    - Higher dropout rates dramatically slowed down learning (compare training curves)
    - With severe dropout, training loss remained high throughout training

3. **Generalization Gap:**
    - No dropout: Large gap between training accuracy (100%) and test accuracy (16%)
    - Moderate dropout: Smaller gap between training accuracy (~21%) and test accuracy (13%)
4. **Model Capacity Utilization:**
    - For this specific task with limited data, the model benefited more from full capacity utilization than from regularization

The experiments demonstrate that dropout effectiveness is highly dependent on the specific model architecture and dataset characteristics. For this particular network and dataset, the model performed best without dropout, suggesting that model capacity was more critical than regularization for this task.

Experiment Results Conclusion

In summary, this experiment aims to create a systematic approach to refining deep neural networks to improve both their optimization during training and their capacity to generalize to unseen data. In examining the architecture in Task 1, we gained insight into how structural differences from fully connected layers to locally connected filters and convolutional layers shape the learning process. Following Task 2, we found that weight initialization fundamentally influences a network's eventual performance. Finally, in Task 3, we explored methods for combatting overfitting and further improving generalization. This was done by integrating ensemble techniques paired with dropout as a form of regularization, creating a combined strategy that balances maintaining high model capacity and preventing overfitting. These three tasks combine architectural design, parameter initialization, and advanced regularization techniques to optimize a neural network's performance and allow for a deeper exploration of the mechanics of optimizing learning models.