CPSC-8430 Deep Learning

Homework-1

PART 1: Function Simulation with Neural Networks

This first segment of comprehensive report focuses on the application of neural networks in simulating non-linear functions using the MNIST dataset Starting with the goal of assessing the capability of various neural network architectures to model two distinctive functions: a nonlinear function and a sigmoid function, both derived from the MNIST dataset.

Model Architectures:

For the experiment, three neural network architectures were used, each with its unique characteristics and advantages:

Convolutional Neural Network (CNN): The CNN model leverages spatial hierarchies through its convolutional layers that employ a number of filters to extract features from the input images. A subsequent max-pooling layer reduces the spatial dimensions before the data is flattened and passed through additional dense layers, culminating in the final output unit.

Layers:

- Reshape layer to add a channel dimension for the CNN (28x28x1).
- Conv2D layer with 32 filters of size 3x3, using ReLU activation.
- MaxPooling2D layer with a pool size of 2x2.
- Flatten layer to transform the feature map into a 1D array.
- Dense layer with 64 neurons, using ReLU activation.
- Dense layer with a single output neuron.

Optimizer: Adam optimizer.

Loss function: Mean Squared Error (MSE).

Metrics: Mean Absolute Error (MAE).

Activation function: ReLU for hidden layers and Conv2D layer.

Dense Neural Network (DNN): The DNN model is constructed with a sequence of operations that start by flattening the input data, transforming the 28x28 pixel images into one-dimensional vectors. This is followed by a series of dense layers equipped with ReLU activation functions to introduce non-linearity, and it concludes with a single output unit to produce the final function values.

Layers:

- Flatten layer to transform the 28x28 image into a 1D array.
- Dense layer with 28 neurons, using ReLU activation.
- Dense layer with 64 neurons, using ReLU activation.
- Dense layer with a single output neuron.

Optimizer: Adam optimizer.

Loss function: Mean Squared Error (MSE).

Metrics: Mean Absolute Error (MAE).

Activation function: ReLU for hidden layers.

Recurrent Neural Network (RNN): The RNN model is specifically designed to handle sequential or temporal data. It utilizes SimpleRNN layers that process inputs in a sequence, allowing the model to maintain a form of memory. This is followed by a dense output layer that provides the final output for the function simulation.

Layers:

- Reshape layer to prepare the input for RNN (sequence of 28 vectors, each of length 28).
- SimpleRNN layer with 64 units, using ReLU activation.
- Dense layer with a single output neuron.

Optimizer: Adam optimizer.

Loss function: Mean Squared Error (MSE).

Metrics: Mean Absolute Error (MAE).

Activation function: ReLU for the SimpleRNN layer.

Training Process:

Each model is compiled using the Adam optimizer, an adaptive learning rate optimizer known for its effectiveness in deep learning applications. The models are trained on subsets of the MNIST dataset, where the aim is to simulate the given functions.

Nonlinear Function

The nonlinear function is defined by adding Gaussian noise to the mean pixel values of the MNIST images, creating a non-trivial task for the neural networks to learn and predict.

Sigmoid Function

The sigmoid function applies the standard sigmoid activation to the mean pixel values of the MNIST images, introducing a different kind of complexity to the task. The training is executed over 100 epochs, providing ample time for the models to learn and refine their predictions.

Visualization and Analysis:

To capture the essence of the training progression, visualization of the optimization process by plotting the loss across training epochs is carried out. This reveals how each model approaches the problem and adjusts its internal parameters to minimize the error.

Training Loss

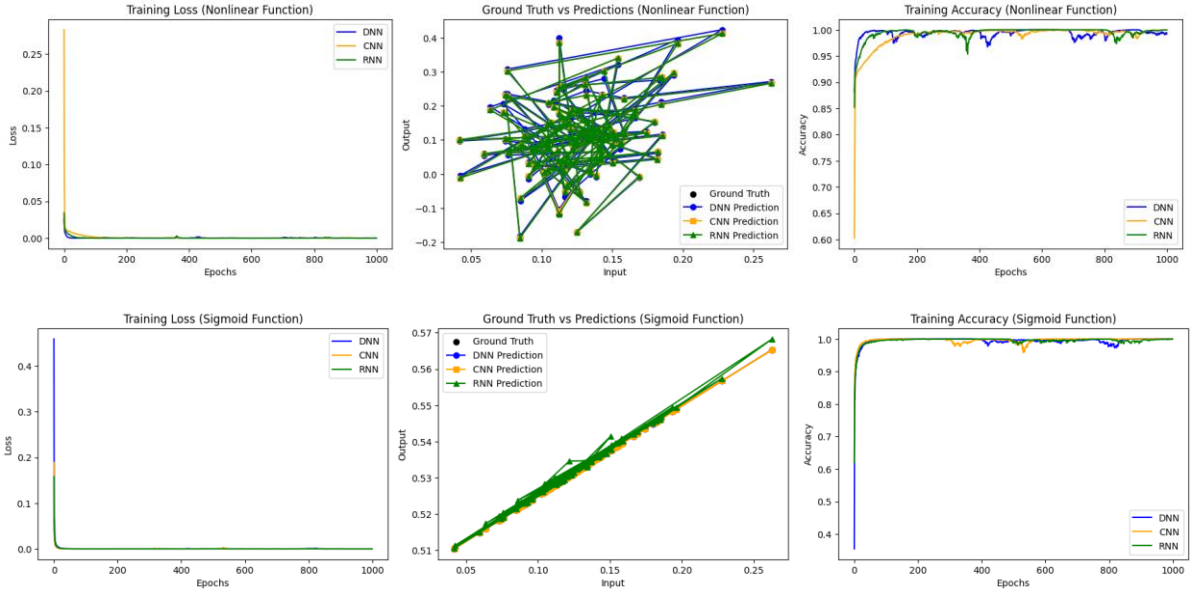
The training loss plot offers a clear visualization of the decrease in loss over time, demonstrating the learning curve and convergence behavior of each model type.

Training Accuracy

The training accuracy plot shows that all models—DNN, CNN, and RNN—quickly reach high accuracy within the initial epochs and maintain it throughout the training, indicating effective learning and model stability. The accuracy curves closely overlap, which suggests that each model type performs similarly on this task.

Predictions vs. Ground Truth

Additionally, plot for the model's predictions against the actual values of the functions to evaluate how closely the models' outputs align with the expected results.



Conclusion

This detailed analysis of function simulation with neural networks on the MNIST dataset provides valuable insights into the learning capabilities and behaviors of different architectures. The DNN, CNN, and RNN models each show unique learning trajectories and convergence patterns, which are essential for understanding their function approximation capabilities. Through these experiments, the aim to contribute to the broader understanding of neural network dynamics and their practical applications in complex function simulations.

PART 2: Neural Network Training Dynamics on MNIST Dataset

In Part 2, delving into the training dynamics of neural networks using the renowned MNIST dataset. This dataset, consisting of 60,000 training images of handwritten digits, serves as a benchmark to assess the performance and learning dynamics of various network architectures. The study centers on three crucial aspects: optimization process visualization, weight behavior during training, and the relationship between the minimal ratio of the Hessian matrix and the loss function.

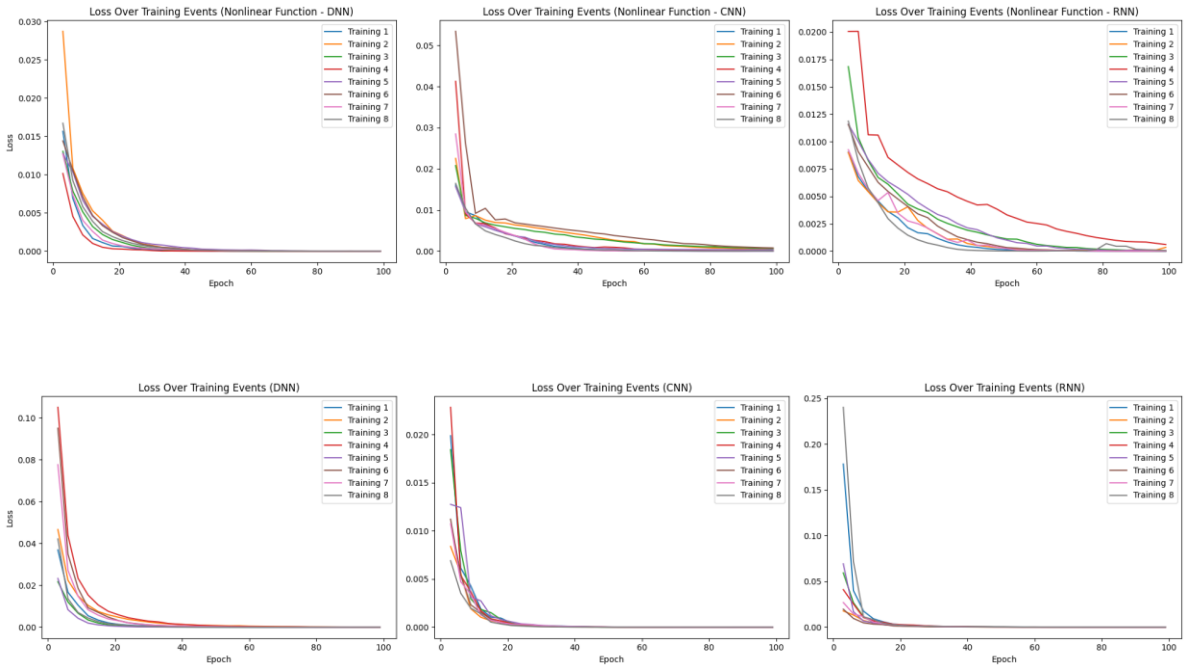
Experiment Setup

The approach encompasses three distinct neural network models: Dense Neural Networks (DNNs), Convolutional Neural Networks (CNNs), and Recurrent Neural Networks (RNNs). Each model is put through eight training iterations, capturing weights at every third epoch using a specialized callback. This method allows for tracking the evolution and internal representations of the models over time.

Adam optimizer with a learning rate of 1e-4 and Stochastic Gradient Descent (SGD) with a learning rate of 0.01, recognizing their widespread application and effectiveness was employed. Additionally, we utilized Principal Component Analysis (PCA) to visualize the high-dimensional weight space, reducing the weight vectors to two primary components.

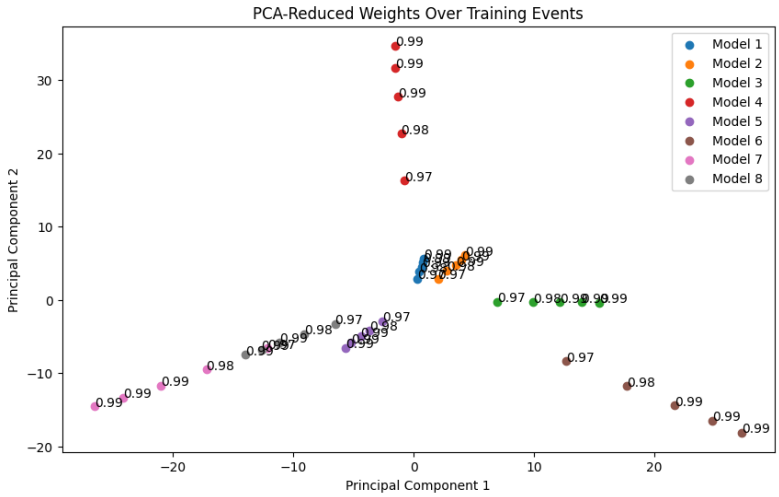
Training Process and Visualization

Training spanned over 100 epochs for each model type to ensure thorough learning and convergence. The optimization process was visualized by mapping the loss over epochs, revealing a decrease in loss and indicating successful learning and stabilization.



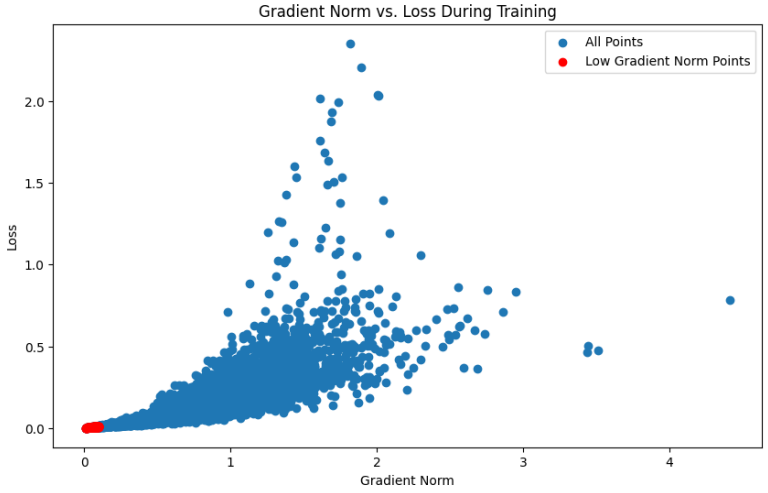
PCA-Reduced Weights Visualization

Through PCA reduction, it is observed the learning trajectory of the models in a lower-dimensional space. Weight clusters formed corresponding to different stages of learning suggest a convergence of models to similar regions in the weight space, signifying robustness in algorithm learning despite initial diversity.



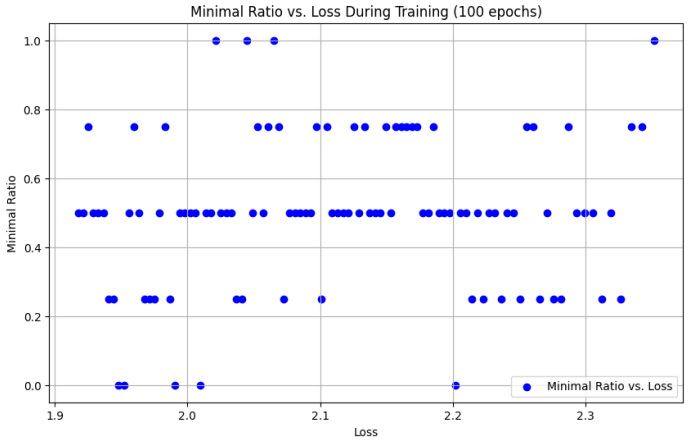
Gradient Norm Analysis

Gradient norms during training, providing insight into the optimization landscape were observed. The decreasing trend of gradient norms as training progressed is indicative of models approaching a local minimum, reflecting theoretical expectations.



Minimal Ratio and Loss Relationship

Plots for the minimal ratio of the Hessian matrix were extracted, an indicator of the loss surface curvature, against the loss function. The scatter plot derived from 100 training runs presented various minimal ratio values at different loss levels, offering a nuanced view of optimization paths and loss landscape complexity.



The findings from the experiment have significant implications for neural network training and optimization. The reduction in loss across varying network types suggests the suitability of chosen architectures and optimizers for the MNIST task. The weight clustering in PCA space underscores the robustness of learning algorithms, while gradient norm analysis confirms convergence behaviors. Lastly, the minimal ratio versus loss analysis underlines the need for adaptive optimization techniques to navigate complex loss landscapes effectively.

Conclusion

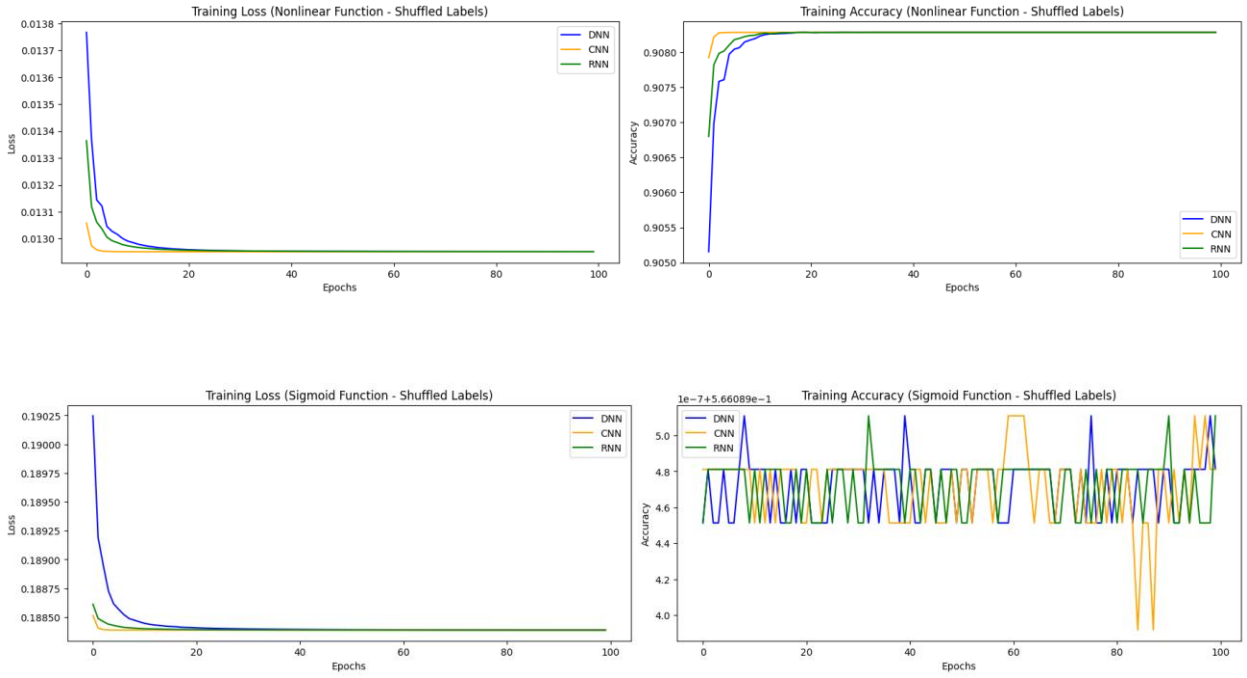
The comprehensive examination has deepened the understanding of neural network training dynamics. The insights gained from optimization visualization, weight behavior analysis, and the Hessian matrix's minimal ratio investigation are invaluable for future research in optimizing neural network performance and reliability.

PART 3: Neural Network Learning Capabilities and Generalization

In the third part of the series of experiments, delving into the intricacies of deep neural network (DNN) learning capabilities, focusing on their ability to generalize from the training data provided. Through an extensive series of tests involving random label fitting and analysis of model behavior across various structures, the aim is to uncover the depth of understanding neural networks possess and the factors influencing their performance.

Can Networks Fit Random Labels?

The initial phase of the investigation challenged the networks to learn from MNIST dataset images paired with randomly shuffled labels. Despite the absence of meaningful correlations between the images and their assigned labels, the networks were able to reduce the training loss. However, as expected, the training accuracy remained low, confirming that while DNNs can fit random patterns, the utility of such a model is questionable without meaningful data.



Experiment Settings

For the experiments, following neural network architectures were used:

DNN Model: A series of fully connected layers with ReLU activation functions, culminating in a softmax layer for classification.

CNN Model: Composed of convolutional layers for feature extraction, followed by dense layers for classification.

RNN Model: Utilizing recurrent layers to capture sequence information in the data, followed by dense layers for classification.

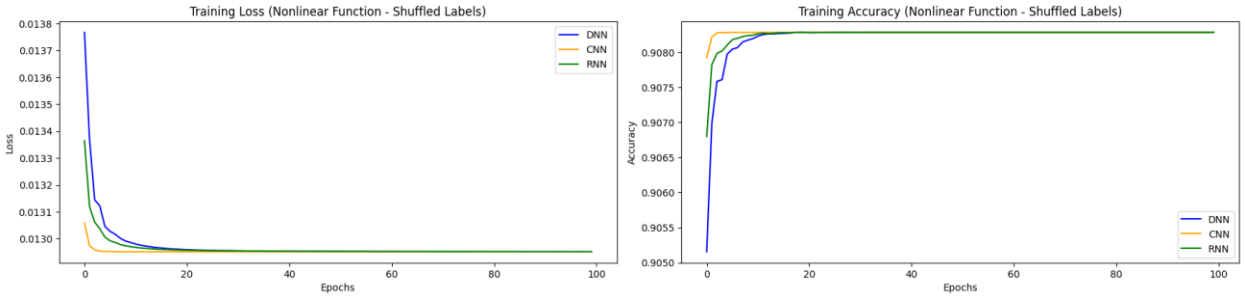
Each model was compiled using the Adam optimizer, known for its adaptability and efficiency in training neural networks. The learning rate for the optimizer was set to the default value provided by the TensorFlow framework unless otherwise specified.

Training Tasks

The task involved training the models on the MNIST dataset, a standard benchmark in machine learning comprising grayscale images of handwritten digits.

Relationship Between Training and Testing, Loss and Epochs

To visualize the training process, the loss against epochs for both the training and testing datasets were plotted. The resulting graphs displayed a typical pattern of rapid initial loss reduction, followed by a plateau, indicating the models' progression from learning to convergence.



Number of Parameters vs. Generalization

In this part of the experiment, models with varying numbers of parameters to investigate the interplay between model complexity and its generalization capabilities were trained. We chose structures with different neuron counts, layers, and depths, and observed their performance on both training and testing data.

Results and Analysis

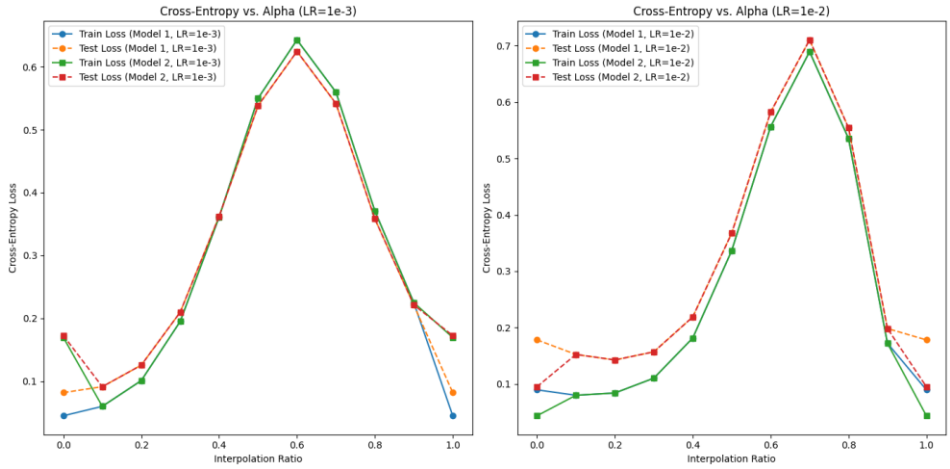
The graphs plotted showed an initial improvement in model performance with an increase in parameters. However, after a certain threshold, additional complexity did not lead to better performance and sometimes resulted in overfitting, as evidenced by increased validation loss.

Flatness vs. Generalization

Part 1: Interpolation and Evaluation

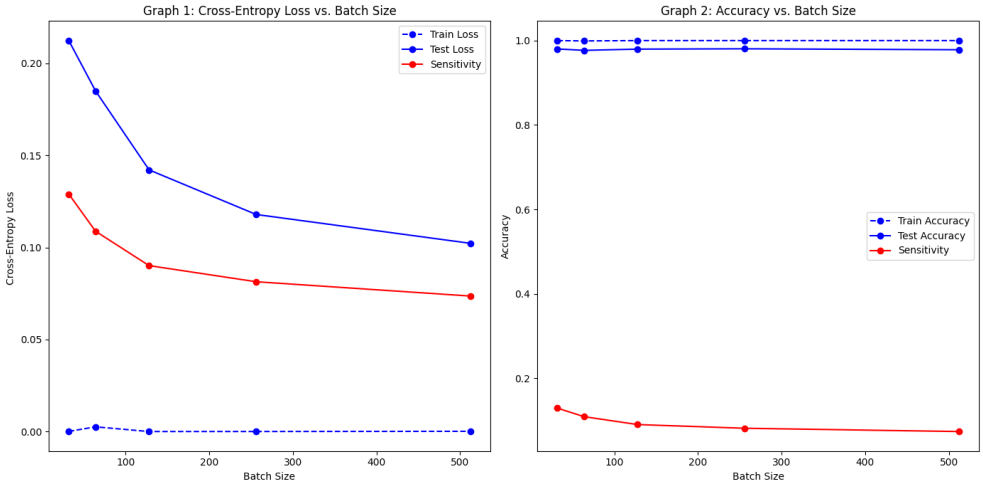
After examining how the flatness of the loss landscape, indicated by the minimal ratio of the Hessian matrix, affected model generalization. By interpolating between different models' weights and evaluating

the resultant models, it is found that flatter regions often corresponded with better generalization and lower loss.



Part 2: Sensitivity Analysis

In the second phase, the model's sensitivity to input perturbations as a proxy for flatness were measured. Models with higher sensitivity tended to show poorer generalization, while those with lower sensitivity (flatter minima) displayed better performance on unseen data.



Conclusion

The comprehensive analysis in Part 3 highlights the necessity of meaningful training data and balanced model complexity for effective generalization. The experiments underscore the role of model architecture and training strategy in achieving robust neural network performance. The results from flatness versus generalization studies further the understanding of the relationship between the optimization landscape and the models' ability to generalize, emphasizing the importance of flat minima for reliable network behavior.