Basics of Deep Learning - Binary and Multiclass classification

Etai Zilberman, Bar Ara

Submitted as mid-semester project report for Basics of Deep Learning course, Colman, 2024

1 Introduction

Deep learning has revolutionized the field of artificial intelligence, providing remarkable solutions to complex problems in computer vision, natural language processing, and autonomous systems. The ability of deep neural networks to automatically learn hierarchical features from data makes them particularly effective for tasks like image classification. In this project, we focused on recognizing hand gestures representing digits in sign language using neural networks. Our aim was to build models capable of classifying images of hand signs into the correct digit class (0-9) with high accuracy.

In the first part of the project, we implemented a binary classification neural network to distinguish between two selected sign language digits. In the second part, we extended this work to develop a multiclass classification model capable of recognizing all digits from 0 to 9 in sign language. Initially, we built a Base Model, followed by Experiment 1 and Experiment 2.

The main objective was to design, train, and evaluate models that could accurately classify hand gesture images into their corresponding digit classes. Through this process, we investigated how different architectural choices and hyperparameter settings impacted the learning behavior, with the goal of developing a robust and generalizable classifier for sign language digits.

1.1 Data

The dataset consists of 5000 grayscale images of hand signs representing digits from 0 to 9. Each image is 28x28 pixels, similar in structure to the MNIST dataset. The data set was provided for us via the gdown command. It was then preprocessed to normalize pixel values between 0 and 1. This normalization facilitates efficient training by preventing large input values from causing instability in gradient-based optimization.

For the binary classification task, the data was split into:

- 80% training data
- 20% test data

For the multiclass classification task, the data was split into training, validation, and test sets with the following ratio:

- 80% training data
- 20% of the training data used for validation
- 20% test data

This split ensures that the model has enough data to learn while allowing proper evaluation and tuning without overfitting. With each epoch, a batch of the validation set was sent to the validation model to make an assessment during training and to check for overfitting/underfitting.

1.2 Problem

The main challenge of this project was to accurately classify hand gesture images into the correct digit class. Initially, we tackled a binary classification problem, where the goal was to distinguish between two selected digit classes. This allowed us to focus on simpler decision boundaries and fundamental model evaluation techniques. In the second part, we extended our efforts to multiclass classification, which introduced the challenge of distinguishing between all ten digit classes. This is a classic multiclass classification problem, requiring the model to learn distinct features for each digit class while managing inter-class similarities. Our goal was to develop models that minimize classification errors and perform well across all digit classes.

To achieve this, we followed a structured approach:

- For the binary classification task, we implemented a straightforward neural network to distinguish between two digit classes. The design was relatively simple and aimed at understanding the fundamentals of binary classification.
- For the multiclass classification task, we implemented a simple Base Model to establish performance benchmarks.
- Experimented with architectural changes to improve accuracy.
- Tuned hyperparameters to optimize learning efficiency and prevent overfitting.

2 Solution

2.1 General Approach

Our solution approach was divided into distinct phases for binary and multiclass classification:

2.1.1 Binary Classification

1. **Data Preprocessing:** Normalize the images and split the dataset.
2. **Model Implementation:** Develop a simple neural network with a single hidden layer to classify between two digit classes.
3. **Training:** Train the model with a suitable learning rate and batch size.
4. **Evaluation:** Assess performance using accuracy and loss metrics, and a confusion matrix.

2.1.2 Multiclass Classification

1. **Data Preprocessing:** Normalize the images and split the dataset.
2. **Base Model:** Implement a simple neural network with minimal layers to establish a performance baseline.
3. **Model Improvements:** Experiment with architectures, by changing the number of layers and number of neurons in each layer, to improve accuracy.
4. **Hyperparameter Tuning:** Optimize learning rate, batch size, and number of epochs.
5. **Evaluation:** Use metrics like accuracy, loss, and classification report to measure performance.

2.2 Design

The model was implemented in Python using PyTorch within a Jupyter Notebook environment. The design choices for the architecture and training process were as follows:

- **Platform:** Jupyter Notebook with PyTorch for model implementation and training.
- **Device:** NVIDIA T4 GPU was utilized to accelerate training and inference, significantly reducing computation time.

- **Training Time:** Approximately 30 seconds for the whole notebook.
- **Loss Function:** BCEWithLogitsLoss() for the binary task and CrossEntropyLoss() for multi-class classification.
- **Optimizer:** Adam optimizer for adaptive learning rate adjustments.
- **Architecture:** A fully connected network starting with 784 input units and 1 output unit for the binary task, and 10 output units for the multiclass task.

2.2.1 BCE with Logits Loss Function

The binary cross-entropy with logits loss function (BCE with logits) was chosen for this project because it is a more numerically stable version of the binary cross-entropy loss. Unlike the standard BCE loss, which requires the model's outputs to be probabilities (i.e., values between 0 and 1), BCE with logits operates directly on the raw output logits of the model (i.e., unbounded real values). This function internally applies a sigmoid activation to the logits, converting them to probabilities before computing the binary cross-entropy.

This approach helps to avoid potential numerical instability that could arise when applying the sigmoid function separately and then computing the BCE. Additionally, BCE with logits is well-suited for binary classification tasks, as it measures the difference between the predicted probabilities (derived from the logits) and the true labels, penalizing incorrect predictions more heavily the further they are from the actual labels. This aligns with the model's goal of accurately distinguishing between two classes.

2.2.2 CrossEntropyLoss

The CrossEntropyLoss function was chosen for this project because it is widely used for multi-class classification. It expects raw logits from the model and internally applies the softmax function to convert them into probabilities. The loss is computed by comparing the predicted probabilities with the true labels, penalizing incorrect predictions.

2.2.3 Adam Optimizer

The Adam optimizer was chosen for this project due to its efficiency and effectiveness in training deep learning models. Adam, short for Adaptive Moment Estimation, combines the advantages of two other popular optimizers: AdaGrad and RMSProp. It adapts the learning rate for each parameter individually by maintaining both the first moment (mean) and the second moment (variance) of the gradients. This allows Adam to converge faster and perform better on problems with sparse gradients or noisy data. Its adaptive nature makes it particularly well-suited for tasks where hyperparameter tuning might be difficult, as it typically requires less manual adjustment of the learning rate.

2.2.4 ReLU Activation Function

The Rectified Linear Unit (ReLU) activation function was chosen for this project because of its simplicity and effectiveness in training deep neural networks. ReLU introduces non-linearity by outputting the input directly if it is positive, and zero otherwise. This simple operation helps mitigate the vanishing gradient problem, enabling faster convergence during training. Additionally, ReLU is computationally efficient, as it only involves a comparison operation. It is particularly effective in hidden layers, where its ability to allow gradients to flow without saturation helps the network learn complex patterns.

2.2.5 Hyperparameters

The hyperparameters, including the hidden layer size, learning rate, and number of epochs, were carefully selected to optimize network performance. The learning rate controlled the step size for parameter updates, balancing speed and stability in convergence.

3 Models

3.1 Model Architecture and Training Setup

In this subsection, we compare the architecture and training setup of the base model, experiment 1 (Exp 1), experiment 2 (Exp 2), and the binary classification model. The models vary in terms of their architecture, loss function, and learning rate. Table 1 shows a summary of these differences.

Aspect	Binary Classification	Base Model	Exp 1	Exp 2
Model Architecture	[784, 64, 1]	[784, 10]	[784, 64, 10]	[784, 64, 10]
Learning Rate	0.001	0.001	0.001	0.003
Loss Function	BCEWithLogitsLoss	CrossEntropyLoss	CrossEntropyLoss	CrossEntropyLoss
Optimizer	Adam	Adam	Adam	Adam

Table 1: Model Architecture and Training Setup Comparison

3.2 Binary Classification Model

For the binary classification model, the output layer was modified to represent a binary classification task instead of a multi-class classification task. The model architecture includes:

- Input Layer: 784 neurons (flattened 28x28 images)
- Output Layer: 1 neuron (representing the binary class output)

The model was trained using the BCEWithLogitsLoss loss function, suitable for binary classification problems, with no need to add an activation function for the output (e.g. sigmoid). It also employed the Adam optimizer for efficient training. The binary model was trained for 10 epochs with a batch size of 64, achieving an accuracy of 93%. This model serves as a benchmark for experiments focused on binary classification tasks.

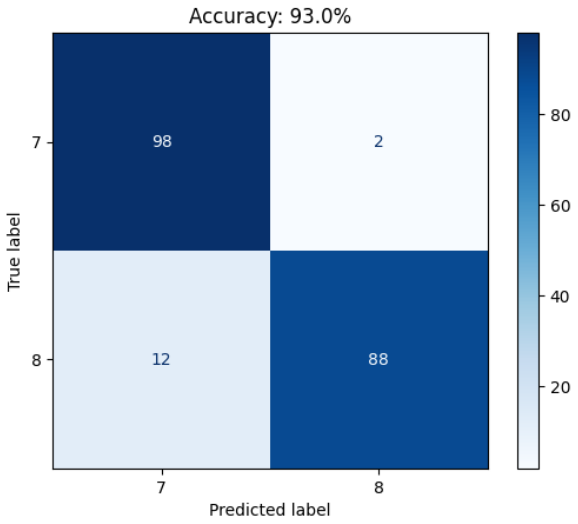

Figure 1: Confusion Matrix

Multiclass Classification:

The following sections describe experiments with the Base Model and its variations, designed for multiclass classification tasks.

3.3 Base Model

The base model consists of a simple feed-forward neural network with the following architecture:

- Input Layer: 784 neurons (flattened 28x28 images)
- Output Layer: 10 neurons (one for each digit class)

This model was trained using a batch size of 64, the Adam optimizer, and the CrossEntropyLoss function for 10 epochs. The accuracy achieved was 89.4%, serving as a benchmark for the experiments that follow. The simplicity of the architecture made it an ideal starting point to evaluate how additional changes in architecture and hyperparameters affect model performance.

3.3.1 Classification Report for Base Model

Class	**Precision**	**Recall**	**F1-Score**	**Support**
0	0.99	0.98	0.98	100
1	0.98	0.97	0.97	100
2	0.93	0.87	0.90	100
3	0.92	0.88	0.90	100
4	0.82	0.77	0.79	100
5	0.91	0.95	0.93	100
6	0.78	0.88	0.83	100
7	0.73	0.79	0.76	100
8	0.95	0.79	0.86	100
9	0.89	0.98	0.93	100
Accuracy		0.89		1000
Macro Avg	0.89	0.89	0.89	1000
Weighted Avg	0.89	0.89	0.89	1000

Table 2: Classification Report for Base Model

3.3.2 Classification Report Summary

The classification report for the Base Model shows an overall accuracy of 89%, with good precision, recall, and F1-scores across most classes. The model performs particularly well on classes 0, 1, 9, and 5, with scores above 0.90, while class 4 shows lower performance (precision 0.82, recall 0.77). There are no signs of significant overfitting, as the macro and weighted averages for precision, recall, and F1-score are all around 0.89, indicating good generalization. Overall, the Base Model serves as a strong baseline for further improvements.

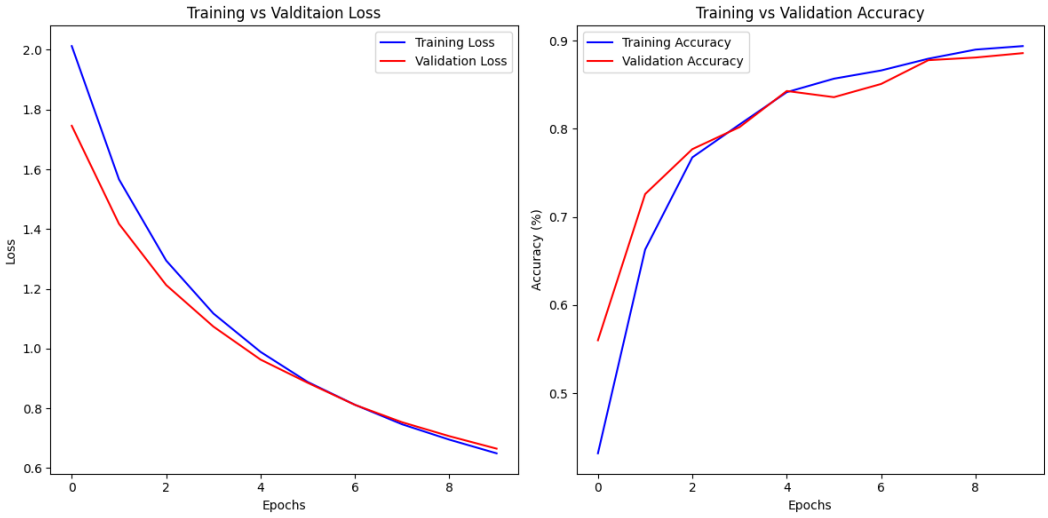

Figure 2: Base Model loss and accuracy

3.4 First Experiment: Changing Architecture

We expanded the model with one hidden layer of 64 neurons and ReLU activation. This allowed the model to learn more complex patterns in the data. Results improved noticeably, showing higher accuracy and better generalization.

3.4.1 Classification Report for Exp 1

Class	Precision	Recall	F1-Score	Support
0	1.00	1.00	1.00	100
1	1.00	0.99	0.99	100
2	0.97	0.97	0.97	100
3	0.98	0.90	0.94	100
4	0.91	0.90	0.90	100
5	0.95	0.99	0.97	100
6	0.92	0.92	0.92	100
7	0.86	0.94	0.90	100
8	0.99	0.89	0.94	100
9	0.93	1.00	0.97	100
Accuracy	0.95			1000
Macro Avg	0.95	0.95	0.95	1000
Weighted Avg	0.95	0.95	0.95	1000

Table 3: Classification Report for Exp 1

3.4.2 Classification Report Summary

The classification report for Experiment 1 indicates a strong performance with an accuracy of 95%. The model shows high precision, recall, and F1-scores, particularly for classes 0, 1, 5, and 9, where the precision and recall reach 1.00. There is some variation in performance, with class 3 showing slightly lower recall (0.90) compared to precision (0.98), and class 8 showing relatively lower recall (0.89) despite good precision (0.99). The macro and weighted averages for precision, recall, and F1-score all stand at 0.95, suggesting that the model generalizes well without significant overfitting.

Figure 3: Experiment 1 loss and accuracy

3.5 Second Experiment: Hyperparameter Tuning

In the second experiment, we fine-tuned the learning rate, batch size, and dropout rate, aiming for further performance improvements.

3.5.1 Classification Report for Exp 2

Class	Precision	Recall	F1-Score	Support
0	0.99	1.00	1.00	100
1	1.00	0.99	0.99	100
2	0.99	0.98	0.98	100
3	0.99	0.94	0.96	100
4	0.95	0.91	0.93	100
5	0.94	0.99	0.97	100
6	0.94	0.94	0.94	100
7	0.91	0.96	0.93	100
8	0.99	0.96	0.97	100
9	0.98	1.00	0.99	100
Accuracy		0.97		1000
Macro Avg	0.97	0.97	0.97	1000
Weighted Avg	0.97	0.97	0.97	1000

Table 4: Classification Report for Exp 2

3.5.2 Classification Report Summary

The classification report for Experiment 2 shows an impressive accuracy of 97%, with precision, recall, and F1-scores consistently high across all classes. Notably, classes 0, 1, and 9 achieved near-perfect scores, with recall reaching 1.00. The model's performance remains strong even for less prominent classes like 4 and 7, with F1-scores of 0.93 and 0.93, respectively. The macro and weighted averages for precision, recall, and F1-score are all 0.97, indicating excellent generalization and no signs of overfitting.

Figure 4: Experiment 2 loss and accuracy

3.6 Best Model Results and Metrics

After evaluating the performance of all the models, we selected **Experiment 2** as our best model due to its superior accuracy and overall performance. This model includes the following architecture:

- Input Layer: 784 neurons (flattened 28x28 images)
- Hidden Layer: 64 neurons with ReLU activation
- Output Layer: 10 neurons (one for each digit class)

The best model achieved an impressive accuracy of **97%** on the test set, significantly outperforming the base model and other experimental configurations. This improvement in performance is attributed to the addition of a hidden layer with 64 neurons, fine-tuning of the learning rate, and changing the batch size from 64 to 128.

A detailed classification report, including precision, recall, and F1-score for each class, further emphasizes the model's well-balanced performance across all digits.

This high accuracy and robust performance make experiment 2 the most effective choice for this task, demonstrating its ability to generalize well to unseen data while minimizing overfitting.

3.7 Performance Comparison

This subsection discusses the performance of the models in terms of training and validation loss and accuracy. The table below (Table 5) presents the results for training and validation metrics, including the batch size and number of epochs used for each experiment.

Aspect	Binary Classification	Base Model	Exp 1	Exp 2
Training Loss	0.134	0.65	0.22	0.13
Validation Loss	-	0.66	0.24	0.16
Training Accuracy	97.12%	89.4%	96.1%	97.6%
Validation Accuracy	-	88.6%	95%	97.3%
Batch Size	64	64	64	128
Number of Epochs	10	10	10	10

Table 5: Performance Comparison: Binary Classification, Base Model, Exp 1, and Exp 2

4 Discussion

Through iterative experimentation, we significantly improved the model's performance from the Base model up to the performance of experiment 2. The addition of a hidden layer, enhanced feature learning and generalization. Hyperparameter tuning such as changing of learning rate to 0.003 and the batch size of 128 further refined the model, leading to our best results.

Future improvements could involve convolutional neural networks (CNNs), which are better suited for image data. We learned about CNNs in lectures, but unfortunately, we could not implement them in this project. Additionally, data augmentation could increase the model's robustness and help the model generalize even better by exposing it to more varied training data.

5 Code

The complete code is available in the Colab notebook: Colab Notebook