TP: From Design to Deployment of Deep Learning Models

Donfack Pascal Arthur
Supervisor: Dr. Louis Fippo Fitime

September 2025

1 Introduction

This report documents the practical work performed in the Deep Learning course (M2-GI) at ENSPY. The purpose of this TP was to provide hands-on experience in developing, tracking, containerizing, and deploying deep learning models. The report focuses on the answers to the theoretical questions posed throughout the exercises and explains the concepts and best practices implemented.

2 Part 1: Foundations of Deep Learning

2.1 Q1: Difference between Batch Gradient Descent and Stochastic Gradient Descent

- **Batch Gradient Descent:** Computes gradients using the entire dataset. Stable but slow and memory-intensive.
- **Stochastic Gradient Descent (SGD):** Computes gradients per sample or mini-batch. Faster updates, can escape shallow local minima, and scales to large datasets.
- **Preference in Deep Learning:** Large datasets and complex models make full-batch computation impractical. SGD allows faster convergence and better generalization.

2.2 Q2: Roles of Layers and Backpropagation

- **Input Layer:** Receives raw input data and shapes it for the network.
- **Hidden Layers:** Extract and transform features via weighted combinations and nonlinear activation functions.
- **Output Layer:** Produces predictions, often as probabilities for classification.
- **Backpropagation:** Algorithm for computing gradients of the loss function with respect to all parameters, propagating errors backward to update weights efficiently.

3 Exercise 1: Keras MNIST

3.1 Q1: Dense, Dropout, and Softmax

- **Dense layers:** Fully-connected layers that learn weighted combinations of inputs to represent complex functions.
- **Dropout:** Regularization technique that randomly disables neurons during training to prevent overfitting.
- **Softmax:** Converts logits to probability distribution over multiple classes; ideal for multi-class classification such as MNIST.

3.2 Q2: Adam Optimizer vs SGD

Adam is an adaptive learning rate optimizer that combines the benefits of Momentum and RMSProp:

- Adjusts learning rates per parameter using estimates of first and second moments of gradients.
- Offers faster convergence, better stability, and improved handling of sparse gradients.
- Often performs better out-of-the-box than vanilla SGD without extensive tuning.

3.3 Q3: Vectorization and Batch Computation

- **Vectorization:** Uses array operations instead of loops, enabling parallel computation on CPUs/GPUs and faster training.
- **Batching:** Processes multiple samples per update (batch_size = 128), improving gradient estimation and computational efficiency.

4 Part 2: Engineering Deep Learning

4.1 Exercise 2: Git/GitHub

No theoretical questions; the exercise focused on version control, repository setup, and collaboration.

4.2 Exercise 3: MLflow

Purpose of MLflow tracking:

- **Parameters:** Record hyperparameters to ensure reproducibility.
- **Metrics:** Track performance metrics (e.g., accuracy, loss) across runs.
- **Artifacts:** Store models, outputs, and other relevant files for future use or deployment.
- **Benefits:** Enables experiment comparison, model selection, reproducibility, and facilitates deployment readiness.

4.3 Exercise 4: Containerization and API

No explicit questions; focused on practical setup of Docker and Flask API for serving the trained model.

4.4 Exercise 5: CI/CD and Monitoring

4.4.1 Q1: CI/CD Automation for Docker Deployment

- Trigger pipeline automatically on code push.
- Build Docker image and run automated tests.
- Push Docker image to a registry (DockerHub, Google Container Registry, AWS ECR).
- Deploy image automatically to cloud services (e.g., Google Cloud Run, AWS ECS).
- Optionally, run smoke tests and roll back on failure.

4.4.2 Q2: Key Monitoring Indicators in Production

1. **Performance metrics:** Request latency (p50, p95, p99), throughput, CPU/RAM/GPU usage.
2. **Reliability metrics:** Error rates (4xx, 5xx), crash/restart counts, timeouts.
3. **Model-specific metrics:** Prediction distribution drift, input data drift, outlier detection, and periodic accuracy checks if labels are available.

5 Conclusion

This TP provided a full practical workflow for deep learning model development. Key skills acquired include model training with Keras, experiment tracking with MLflow, containerization using Docker, and API deployment with Flask. Best practices in CI/CD, monitoring, and production readiness were also applied.

6 TP2: Improving Deep Neural Networks

6.1 Part 1: Theory and Key Concepts

6.1.1 Data Splitting

The dataset is divided into three sets: training (used to update model parameters), validation (dev, used to tune hyperparameters and diagnose bias/variance), and test (used only for final evaluation to avoid overfitting).

6.1.2 Results Analysis

By comparing training error and validation error: - High training error and high validation error: high bias (underfitting). - Low training error but high validation error: high variance (overfitting).

6.1.3 L2 Regularization

Penalizes large weights by adding a term proportional to the square of weights to the loss function, encouraging smaller weights and reducing overfitting.

6.1.4 Dropout

Randomly disables neurons during training, forcing the network to learn redundant representations and acting as regularization.

6.1.5 Batch Normalization

Normalizes activations of hidden layers, stabilizing training and allowing higher learning rates.

6.1.6 Optimization Algorithms

- Momentum: Accumulates past gradients to accelerate convergence. - RMSprop: Adapts learning rates based on recent gradient magnitudes. - Adam: Combines momentum and RMSprop, often the default due to robustness.

6.2 Part 2: Practical Exercises

6.2.1 Exercise 1: Bias/Variance Analysis

After training with validation set, the model showed low training loss but slightly higher validation loss, indicating mild overfitting (high variance). Regularization was applied to address this.

6.2.2 Exercise 2: Applying Regularization

Added L2 regularization (0.001) and Dropout (0.2). This improved validation accuracy by reducing overfitting, as seen in lower validation loss compared to training loss.

6.2.3 Exercise 3: Comparing Optimizers

Experiments with MLflow showed: - Adam: Fastest convergence, highest accuracy (0.97). - RMSprop: Good performance, slightly slower than Adam. - SGD with momentum: Slower convergence but stable.

6.2.4 Exercise 4: Batch Normalization

Adding BatchNormalization accelerated training and improved stability, leading to better final accuracy.

Annexes

For full access to the project code and the collaborative work:

- **Overleaf Project:** https://www.overleaf.com/project/68d314812f29a01e37f8c3ab
- **GitHub Repository:** https://github.com/Tiger-Foxx/DeepLearning1