Practical Work 2: Improving Deep Neural Networks

Louis Fippo Fitime, Claude Tinku, Kerolle Sonfack
Department of Computer Engineering, ENSPY

September 22, 2025

Abstract

This second Practical Work document focuses on advanced engineering practices for neural networks. The objective is to go beyond simple model training to understand and apply essential techniques such as hyperparameter tuning, regularization, and advanced optimization, in order to build more performant and robust models.

Learning Objectives

- Understand industry best practices for the development of DL models.
- Diagnose problems of **high bias** and **high variance** and know how to solve them.
- Master regularization techniques: **L2** and **Dropout**.
- Learn to use **Batch Normalization** to accelerate and stabilize training.
- Know how to implement and compare the main optimization algorithms (Momentum, RMSprop, Adam).

1 Part 1: Theory and Key Concepts (2h)

1.1 Performance Diagnosis: Bias vs. Variance

Performance diagnosis is a crucial step in the model lifecycle. A model suffers from **high bias** (underfitting) if it fails to capture the relationship between input and output data, even on the training set. A model suffers from **high variance** (overfitting) if it performs well on the training set but poorly on new data.

- **Data Splitting:** Explain the importance of dividing the dataset into three distinct sets: **training**, **validation (dev)**, and **test**. Describe the role of each.
- **Results Analysis:** How can simply examining the training error and the validation error allow you to diagnose whether your model suffers from bias or variance?

1.2 Regularization and Normalization

Regularization techniques help to reduce a model's variance and prevent overfitting.

- **L2 Regularization:** Describe the principle of L2 regularization (also known as "weight decay") and how it penalizes large weights.
- **Dropout:** Explain how dropout works and why it acts as a regularization technique.
- **Batch Normalization:** Explain how batch normalization helps to standardize the activations of hidden layers and to stabilize and accelerate training.

1.3 Advanced Optimization Algorithms

The chosen optimization algorithm has a major impact on the convergence speed. Briefly describe the principle of Momentum, RMSprop, and Adam, and explain why Adam is often considered the default choice.

2 Part 2: Practical Exercises (5h)

2.1 Exercise 1: Bias/Variance Analysis with Keras

In this exercise, you will take your model from the previous TP and modify it to analyze its performance.

Instructions:

1. Take the `train_model.py` file from your TP 1.
2. Modify the data loading to explicitly create training, validation (dev), and test sets.

```
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.
    load_data()

Use 90% for training and 10% for validation (dev)
x_val = x_train[54000:]
y_val = y_train[54000:]
x_train = x_train[:54000]
y_train = y_train[:54000]

Normalization and reshaping as in TP 1
...
history = model.fit(
x_train,
y_train,
epochs=5,
batch_size=128,
validation_data=(x_val, y_val) # Use the validation set
)
```

Listing 1: Creating the Datasets

3. Run the model. Observe the training history (loss and accuracy on the training and validation sets).
4. **Question:** Based on the observed results, diagnose whether your model suffers from bias or variance. Justify your answer.

2.2 Exercise 2: Applying Regularization

To solve overfitting problems (high variance), you will add regularization to your model.

Instructions:

1. Modify your model's definition in `train_model.py` to add:
   - An L2 regularization term on the dense layer (kernel_regularizer=keras.regularizers.l2(0.001)).
   - A Dropout layer after the input layer and before the output layer.
2. Train the model and compare the training and validation loss and accuracy curves with the non-regularized model.
3. **Question:** Did the regularization techniques improve your model's performance on the validation set? Explain why.

2.3 Exercise 3: Comparing Optimizers

The choice of optimizer can influence the convergence speed. You will test different algorithms.

Instructions:

1. Use MLflow to track your experiments. For each training run, log the name of the optimizer used.
2. Create a loop that trains your model with the following optimizers:
   - adam
   - sgd (with learning_rate and momentum)
   - rmsprop

```python
optimizers = {
'SGD_with_momentum': keras.optimizers.SGD(learning_rate=0.01,
   momentum=0.9),
'RMSprop': 'rmsprop',
'Adam': 'adam'
}

for opt_name, optimizer in optimizers.items():
with mlflow.start_run(run_name=f"Optimizer_Comparison_{opt_name}"):
model = # Recreate the model here with the same parameters
model.compile(optimizer=optimizer, ...)

history = model.fit(...)

# Log metrics and parameters with MLflow
mlflow.log_param("optimizer", opt_name)
mlflow.log_metric("final_test_accuracy", test_acc)
# You can also log loss curves for better analysis
```

Listing 2: Example of an Optimizer Loop

2.4 Exercise 4: Batch Normalization

Instructions:

1. Add a BatchNormalization layer between the Dense layer and the Dropout layer in your model.
2. Train the model and compare the convergence speed with the previous versions.

```python
model = keras.Sequential([
keras.layers.Dense(512, activation='relu', input_shape=(784,)),
keras.layers.BatchNormalization(),
keras.layers.Dropout(0.2),
keras.layers.Dense(10, activation='softmax')
])
```

Listing 3: Adding Batch Normalization

3 Conclusion

To submit: 29/09/2025

- The link to your GitHub or GitLab repository with the modifications from TP 2.
- A short report (`.pdf` file + overleaf link) summarizing your experiments with the different optimizers and regularization techniques.