Università degli Studi di Napoli Federico II

Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione

Neuro BackPropagation Lab

Giuliano Aiello

2025

Contents

1 Prolusion **1**
1.1 Goal . . . 1
1.2 Software Stack . . . 2
1.3 Project Structure . . . 3

2 Module Overview **5**
2.0.1 Model . . . 5
2.0.2 Tester . . . 5
2.0.3 Trainer . . . 5
2.0.4 Utils . . . 5

3 Resilient Backpropagation Techniques **7**
3.1 Implementations . . . 7
3.1.1 Rprop- . . . 7
3.1.2 Rprop+ . . . 7
3.1.3 Improved Rprop with Weight-Backtracking . . . 7
3.1.4 Rprop+ by PyTorch . . . 7
3.2 Comparisons . . . 7

Acronyms **11**

Chapter 1

Prolusion

1.1 Goal

This report provides a comprehensive overview of a Python project whose goal is to develop and compare different adaptive backpropagation techniques involved in a machine learning process, as Rprop (Resilient BackPropagation).

The project follows the "Empirical evaluation of the improved Rprop learning algorithms" article by Christian Igel and Michel Hüsken (2001).

1.2 Software Stack

- Python 3.9.6
- PyTorch 2.6.0

The project is equipped with a `requirements.txt` file which allows for seamless installation of dependencies, by executing `pip install -r requirements.txt`.

1.3 Project Structure

```
neuro-backprop-lab/
├── model/
├── tester/
│   ├── tester.py
│   └── <trained_model>.pt
├── trainer/
│   ├── irpropplus/
│   ├── rpropminus/
│   ├── rpropplus/
│   └── trainer.py
├── utils/
├── test_model.py
└── train_model.py
```

- model includes the neural network model architecture.
- tester handles the testing flow of the ready-to-use <trained_model>.pt.
- trainer handles the examined backpropagation techniques and the training flow of the model, saving it as <trained_model>.pt.
- utils offers utility functions designed to support the root project scripts.

Chapter 2

Module Overview

The part of the project which is shared across all the examined Rprop techniques is presented as follows.

2.0.1 Model

This class represents the artificial neural network model architecture to be trained and tested.

It is a shallow network based on `torch.nn.Module`[1] class. Its three layers are fully connected using `torch.nn.Linear(.)` and they feed forward as follows:

1. the first layer flattens the input MNIST image, by transforming it from a multidimensional vector to a 784-sized (since a 28×28-sized image is manipulated) one-dimensional vector;
2. the hidden layer receives the transformed vector and processes it into a 128-sized vector with a ReLU activation function to introduce non-linearity, a choice that was the result of empirical experiments;
3. the output layer extracts the final predictions by transforming the 128-sized vector into a 10-sized vector, which corresponds to the number of possible classes for classification.

2.0.2 Tester

This class is responsible for loading and running a pre-trained model on unseen data.

In fact, the “holdout” approach is adopted: the MNIST is separated into training set and test set. Pseudocode of the test flow follows.

2.0.3 Trainer

2.0.4 Utils

[1] *https://pytorch.org/docs/stable/generated/torch.nn.Module.html* (accessed 2025)

Chapter 3

Resilient Backpropagation Techniques

Rprop algorithms differ from the classical back-propagation algorithms by the fact that they are independent of the magnitude of the gradient, but depend on its sign only.

3.1 Implementations

3.1.1 Rprop-

This is Rprop-.

3.1.2 Rprop+

This is Rprop+.

3.1.3 Improved Rprop with Weight-Backtracking

This is IRprop+.

3.1.4 Rprop+ by PyTorch

This is Rprop+ by PyTorch.[1]

3.2 Comparisons

Here I will show comparisons.

[1] *https://pytorch.org/docs/stable/generated/torch.optim.Rprop.html* (accessed 2025)

Acronyms

MNIST Modified National Institute of Standards and Technology database 5

ReLU Rectified Linear Unit 5

Rprop Resilient BackPropagation 1, 5, 7