DATA130051.01

Computer Vision

Project I
Latte

Shao Yi

2022-04-11

Abstract

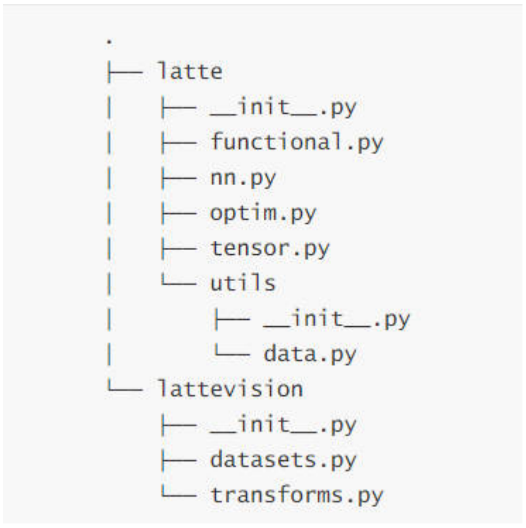
Latte (**L**et's **A**bsorb **T**orch **T**echnology **E**legantly) is a self-designed deep learning framework working on CPU, the package name shows tribute to Caffe while the inner structure is inspired by the PyTorch framework. This project focuses on the manual implementation of the most common deep learning package modules and solves the **MNIST** dataset classification problem.

GitHub Repo: https://github.com/Tequila-Sunrise/FDU-Computer-Vision

Best Model: https://github.com/Tequila-Sunrise/FDU-Computer-Vision/tree/main/Project/best_models

1 Introduction

The package structure in the tree view is as follows:



```
.
├── latte
│   ├── __init__.py
│   ├── functional.py
│   ├── nn.py
│   ├── optim.py
│   ├── tensor.py
│   └── utils
│       ├── __init__.py
│       └── data.py
└── lattevision
    ├── __init__.py
    ├── datasets.py
    └── transforms.py
```

Figure 1: Package Structure

Latte packages some basic modules, like **tensor** defines the basic data structure, **nn** together with **functional** provides necessary classes and functions for neural network, **optim** includes some classical optimization methods, and **utils.data** is for dataset and data loader. All the interfaces are referred to the official documentation of PyTorch.

Furthermore, what torchvision is to torch, lattevision is to latte. It includes some requisite modules for computer vision tasks. To be more specific, **transforms** contains some transformation functions for image processing, **datasets** implements the downloading and loading of MNIST dataset for now.

All the features above are implemented using **numpy** package.

For more information, please refer to the source code.

1.1 Module tensor

Tensor is the basic data structure in Latte which wraps the **numpy array** together with some other features such as **gradient**, **require grad**, and so on.

One important feature need to be noted is that tensor basically reproduces **Computational Graph** for every single operation in matrix computation.

1.2 Module nn & functional

These two modules are the core of the framework. They are used to define the neural network layers like **Linear**, **Sigmoid**, **ReLU**, **Dropout** and loss functions like **MSELoss**, **BCELoss**, **CrossEntropyLoss**.

1.3 Module optim

Manually implementation of **SGD** and **Adam**.

Also the **L2 regularization** is provided as the interface of **weight decay**.

1.4 Quick Start

This part is a quick start guide, we run a toy example to see whether the framework works. By the way, all the code implementations are included in **quick start** jupyter notebooks.

As an example, we use only **fully connected** layers to build a simple network, the activation function is **ReLU**. For criterion and optimizer, we simply use **CrossEntropyLoss** and **Adam** respectively.

The nerwork architecture is shown in the following figure:

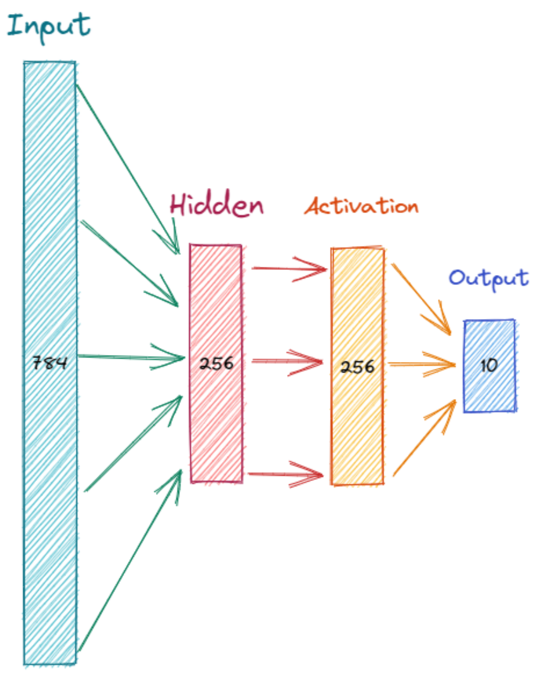


Figure 2: Simple Network

Finally we get the accuracy **96.59%**. In the next part, we will do some search on hyperparameters to find the best model.

2 Hyperparameter Search

Instead of making attempts in different learning rate, we apply a **decreasing scheme** for learning rate. The learning rate is set to **0.01** initially, and we decrease it by a factor of **0.9** when the validation loss does not improve.

Then we search for the best **hidden units** and **weight decay** in the space of **[64, 128, 256, 512]** and **[0, 1e-5]** respectively.

Here's the result:

Units	Weight Decay	Best Val Loss	Best Val Acc	Test Acc
64	0	0.0055	95.56%	96.50%
64	1e-5	0.0050	96.24%	96.44%
128	0	0.0047	96.60%	97.22%
128	1e-5	0.0044	96.68%	97.23%
256	0	0.0042	96.83%	97.47%
256	1e-5	0.0039	97.43%	97.61%
512	0	0.0041	97.36%	97.85%
512	1e-5	0.0039	97.38%	**97.92%**

Table 1: Number of Hidden Units

And the training process for the best model is shown in the following figure:

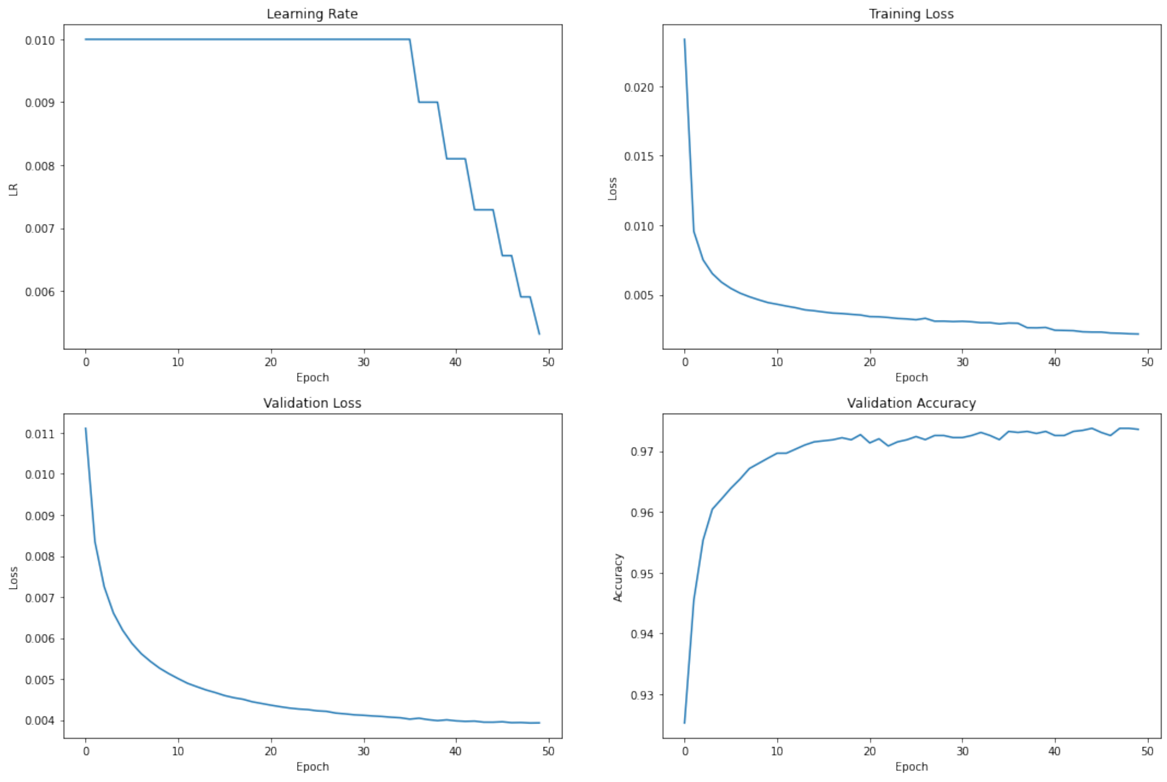


Figure 3: Training Process

More details for other models can be found in the **hyperparam search** jupyter notebook.

3 Parameter Visualization

The visualization of the best model's parameters in heatmap is as follows:

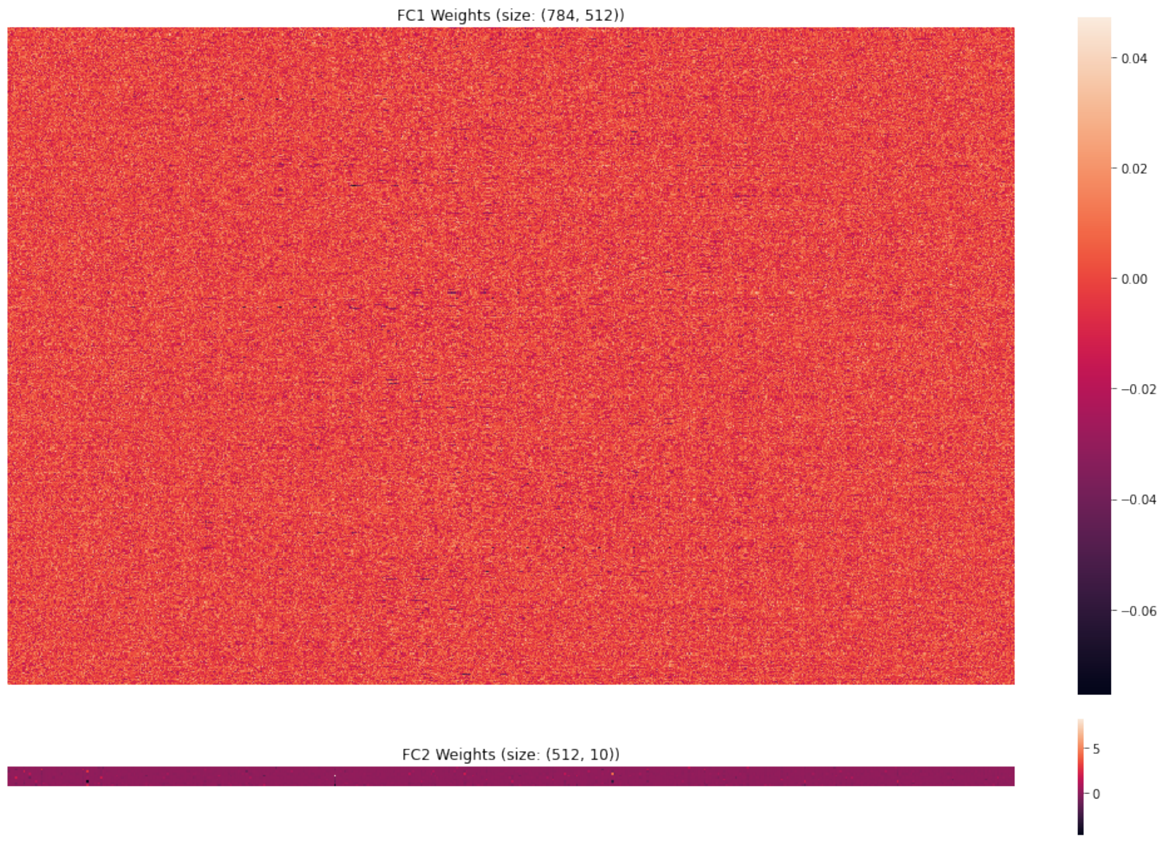


Figure 4: Heatmap Best

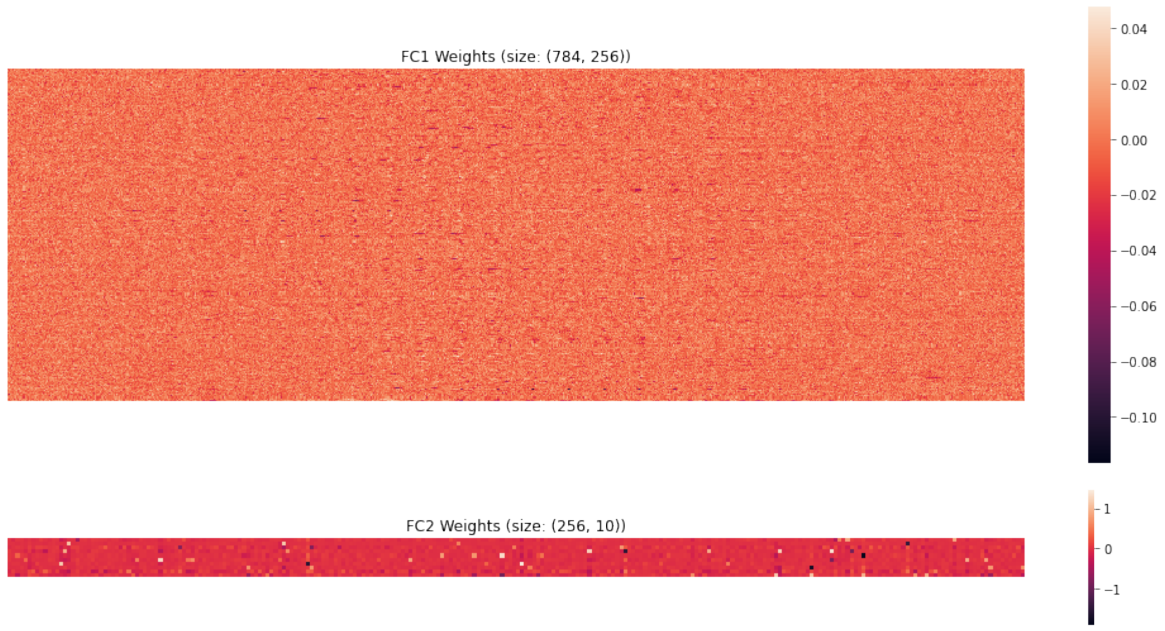


Figure 5: Heatmap 256

We can find some implicit grids in the heatmap regularly.