FINAL PROJECT REPORT

Course: Introduction to Machine Learning

Full name: Lê Ngọc Anh Khoa

Student's ID: 22127196

Class: 22KHMT2

DECEMBER 29, 2024

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ HIÊN – TP.HỒ CHÍ MINH
227 Đ. Nguyễn Văn Cừ, Phường 4, Quận 5, Hồ Chí Minh, Việt Nam

I. Overview

1. About Project

This project focuses on analyzing and comparing popular machine learning libraries and frameworks, specifically TensorFlow, Torch, Scikit-Learn, and JAX (with Flax). The analysis includes evaluating their performance in terms of training time, GPU memory usage, ease of use, and identifying their respective pros and cons.

2. Student Information

Full Name: Lê Ngọc Anh Khoa

Student ID: 22127196

Class: 22KHMT2

3. Self-Evaluation

No.	Details	Max Score	Score
1	Scikit-learn (MLPClassifier)	30%	25%
2	TensorFlow / Keras	30%	30%
3	PyTorch	30%	30%
4	Report	10%	10%
5	Others framework: JAX	20%	20%
	Total	120%	115%

II. Models Implementation

1. Model Design

For consistency, the same neural network architecture was implemented across all frameworks. The architecture includes:

- Input layers: 3072 (total features)
- Hidden layers: 3 layers with 1024, 512, 256 neurons respectively
- Activation: ReLU for all hidden layers
- Output layers: 10 with softmax activation for classifying CIFAR-10 dataset
- Optimizer: Adam with learning rate 0.0001
- Loss function: Cross Entropy

- Technique to prevent overfitting: dropout with rate 0.3 and early stopping based on validation accuracy with patience 5 epochs (iterations)

To get this architecture, I researched and applied hyperparameter tuning to choose the most suitable parameters for my project.

2. Implementation

a. TensorFlow

- Using the Sequential API from TensorFlow to implement following these steps:
    - Create a Sequential API model
    - Add layers with activations and dropout method for each layer
    - Add output layer
    - Compile model with optimizer and loss function
    - Fit the model with dataset (labels must be one-hot encoded before)

b. PyTorch

- Using the PyTorch torch.nn.Module class to implement model following steps:
    - Create a custom module class inheriting from torch.nn.Module
    - Define layers in the constructor, including activations and dropouts. Define forward propagation logic
    - Define the optimizer and loss function.
    - Implement the training loop for the model by myself (including batch loops and early stopping logics)
    - Train the model using a data loader (must convert data to DataLoader)

c. Scikit-Learn

- Using MLPClassifier class from Scikit-Learn framework to implement following steps:
    - Initialize model with the architecture
    - Fit the model with data which is flattened

d. JAX (with Flax)

- Using JAX and Flax modules to implement model following steps:
    - Define a custom MLP model class inherit from Flax's module class

- Add layers, activation and dropout in model definition
- Define loss function and optimizer
- Use JAX functions to initialize the model's parameters and apply them during training
- Define train function to train each batch of training loop (include dropout)
- Implement training loops for model (include batch loops and early stopping logic)

III. Analysis & Comparison

1. Training Time & Memory Usage

	Scikit-Learn	**Torch**	**TensorFlow**	**JAX**
Training time (s)	816	57	54	34
GPUs usage (MB)	0	1314	1157	10

- Training time:
  - JAX > Torch, TensorFlow >>>>> Scikit-Learn
- Memory usage:
  - JAX >> TensorFlow, Torch
  - Scikit-Learn framework doesn't use GPUs
- Summary:
  - **JAX** is an excellent choice for scenarios requiring high-performance training with minimal resource usage
  - **TensorFlow** and **Torch** are nearly equal in terms of speed and memory efficiency.
  - **Scikit-Learn** slow and limited in resource utilization for deep learning

2. Ease Of Use

- Here's my personal experience:
  - **Scikit-Learn**: most friendly framework for beginners but difficult to add techniques like Dropout or others to improve performance.

- **TensorFlow:** easy to build and train the model since this framework has a high-level API
- **PyTorch:** moderate level for me to understand and work with since this framework needs us to define training loop to train the model
- **JAX:** this is the most difficult for me since I need to redefine most necessary functions to train the model.

3. Pros & Cons

	Pros	Cons
Scikit-Learn	Easy to implement Best choice for small dataset	Slow Has no GPU support Limited support for MLP (like dropout)
TensorFlow	Strong community Good GPU support (speed) Easy to access training history	High memory usage
PyTorch	Strong community High performance (accuracy) Good GPU support (speed) Flexible	Moderate level to begin High memory usage
JAX	High performance Best GPU support Flexible	Difficult to begin Small community Lacks high-level APIs

IV. References

- Hyperparameter tuning
- TensorFlow
- PyTorch
- JAX