Solving HJB Equation in Multi-Class Queueing Control Systems Using Neural Network-Based BSDE Solver

Hongzhang "Steve" Shao

December 22, 2024

Abstract

This document details my study of Jiequn Han's neural network-based numerical method for solving Hamilton-Jacobi-Bellman (HJB) equations. I analyze his code implementation and extend the method to solve the HJB equation arising from a dynamic scheduling problem in multi-class queueing systems. The document outlines the high-level approach, presents pseudocode for new implementations, and discusses key technical considerations.

Contents

1 The High-level Plan

1. **Get Jiequn's Python code up and running: (Completed)**
   - **Task**: Set up Python environment per Jiequn's requirements. (✓)
   - **Task**: Clone Jiequn's code. Test it. Add testing code for consistency in later changes. (✓)
   - **Task**: Add comprehensive comments and docstrings to understand the code better. (✓)
   - **Task**: Reorganize project files for direct use as Python package or git submodule. (✓)
   - **Result**: Pushed the updated code to Github: DeepBSDE-Package
   - **Observation**: Everything works, except the code can only run on CPU, not GPU.
     Diagnosis: The code uses `TensorFlow 2.13` (without GPU support), thus can only run on CPU.
2. **Get Ebru's Python code up and running:**
   - **Task**: Test Ebru's Python code under Jiequn's project environment. (✓)
   - **Task**: Solve Ebru's model class with Jiequn's solver. Check consistency in results. **(pending)**
   - **Observation**: With Jiequn's code and CPU, the 2D test problem takes > an hour to run.
     It will be much easier to work with the code if it can run on GPU. So I added the next step.
3. **Get Jiequn's Python code running on GPU: (Failed)**
   - Note: There is another distribution of `TF 2.13` that have `CUDA` (Nvidia's GPU firmware) support. However, `TF 2.13` only supports `Cuda 11.3`, not `Cuda 12.4` used by Vast.ai, Mercury, etc., by default. Downgrading `Cuda` risks conflicts with other packages, so I avoid this option.
   - **Attempt 1**: For GPU, setup an env with latest `Cuda 12.4`-compatible `TensorFlow 2.18`. (✓)
     Result: Code cannot run due to syntax changes in TensorFlow from 2.13 to 2.18.
   - **Attempt 2**: Make minimal possible changes to code for compatiblility with both env. (✓)
     Result: Code runs on GPU with new env, but the training loss is not improving.
   - **Attempt 3**: To check error in Attempt 2, run new code in original env with `TF 2.13`. (✓)
     Result: Code runs on CPU as original. Loss converges correctly. Thus, the changes where correct.
   - **Diagnosis**: Two possibilities:
     (a) Compatibility of original code itself with GPU:
     (b) Compatibility of original code with new `TF 2.18`:
   - **Attempt 4**: To determine which is the issue, test new code and new env on CPU. (✓)
     Result: Code runs on CPU with new env, but the training loss is not improving, same as on GPU. So, it's (b).
   - **Attempt 5**: Discuss with colleagues for possible fixes. (✓)
     Result: There's two possible fixes:
     - (i) rewrite the code for `TF 2.18`
     - (ii) rewrite the code with `PyTorch`

     Dawei mentioned `TF` is known for inconsistency over versions. He recommended `PyTorch`.
4. **Get Jiequn's Python code running on GPU with `PyTorch` (with minimal changes possible):**
5. **Understand Ebru's model in detail. Put pseudocode in writing. Update model class:**
6. **Understand "tricks" in Ebru's newer Julia code. Update Python model class accordingly:**
7. **Attempt training Ebru's model with Jiequn's solver. Report results:**
8. **Iteratively improve training results:**