Homework1: Stochastic Hopfield Network

Hai Dinh (19960331-4494)

1. Code

To speed up the computation, I use the `multiprocessing` package, which supports thread-safe queues, where the main thread can give tasks to the worker threads via `tasks_queue` and the worker threads can write back the results via the `done_queue`.

A task for each worker thread is simply a single trial run, where the thread will compute the order parameter for a network that stores some random patterns. Once the order paramemters are computed for all experiments, the main thread will take the average and print out the final results.

```
import time
import random
import argparse
import numpy as np
import multiprocessing as mp
from progressbar import progressbar # pip install progressbar2

def noisy_sigmoid(value, beta=2):
    """
    Compute the noisy sigmoid activation in stochastic Hopfield network, with noise parameter beta.
    """
    return 1 / (1 + np.exp(-2 * beta * value))

def generate_random_patterns(p, N=200):
    """
    Generate `p` random patterns, each with `N` bits/neurons.
    """
    patterns = np.random.randint(2, size=(p, N))
    patterns[patterns == 0] = -1
    return patterns

def compute_weight_matrix(stored_patterns, row=None, zerodiag=True):
    """
    Create weight matrix for Hopfield network using Hebb's rule.

    Args:
        stored_patterns (ndarray): Tensor of shape `(p, N)` containing `p` stored patterns, each
            represented by `N` neurons/bits.

        row (int): Create weight matrix only for this row, such that the return value will have the
            shape of `(1, N)` (where N is the number of bits in each stored pattern). If no row is
            given, then the entire weight matrix of shape `(N, N)` will be computed.
```

```python
        zerodiag (bool): If true, the weights along the diagonal of the weight matrix will be zeroed
            out. Otherwise, the diagonal weights will be computed using the normal Hebb's rule.

    Returns:
        The weight matrix of shape `(1, N)` if row is given. Otherwise, the entire matrix of shape
        `(N, N)` will be returned.
    """
    N = stored_patterns.shape[1]
    W = stored_patterns if row is None else stored_patterns[:, row, None]
    W = (W.T @ stored_patterns) / N

    if zerodiag:
        if row is None:
            np.fill_diagonal(W, 0)
        else:
            W[:,row] = 0
    return W

def run_experiment(p, T=int(2e5)):
    """
    Run one trial by creating a stochastic Hopfield network and compute its order parameter.

    Args:
        p (int): Number of random patterns to store in the network.

        T (int): Number of asynchronous updates used for computing the order parameter.

    Returns:
        A number representing the order parameter of the stochastic Hopfield network.
    """
    order_param = 0
    random_patterns = generate_random_patterns(p)
    W = compute_weight_matrix(random_patterns)

    input_pattern = random_patterns[0]
    pattern = np.copy(input_pattern)
    N = len(pattern)

    for _ in range(T):
        random_neuron = random.randrange(N)
        prob = noisy_sigmoid(np.dot(W[random_neuron], pattern))
        pattern[random_neuron] = 1 if random.random() <= prob else -1
        order_param += np.dot(pattern, input_pattern) / N / T

    return order_param

def worker_thread(tasks_queue, done_queue):
    """
    Worker thread will take one task from the `tasks_queue`, run the experiment, and then put the
    final result (order parameter) into the `done_queue`.
    """
    while True:
        p = tasks_queue.get(block=True)
        order_parameter = run_experiment(p)
```

```python
        done_queue.put(order_parameter)

def main_thread(args):
    """
    Main thread will initialize worker threads and compute the average order parameter of all runs.
    """
    order_params = []
    tasks_queue = mp.Queue()
    done_queue = mp.Queue()
    workers = mp.Pool(args.n_workers, initializer=worker_thread, initargs=(tasks_queue, done_queue))

    for _ in range(args.n_trials):
        tasks_queue.put(args.p)

    for _ in progressbar(range(args.n_trials)):
        order_param = done_queue.get(block=True)
        order_params.append(order_param)

    workers.terminate()
    workers.join()
    print("Average order parameter: {:.3f}".format(sum(order_params) / len(order_params)))

if __name__ == "__main__":
    parser = argparse.ArgumentParser(
        description="Compute order parameter for stochastic Hopfield network"
    )
    parser.add_argument(
        "p",
        type=int,
        help="Number of stored patterns"
    )
    parser.add_argument(
        "--n-workers",
        "-w",
        default=12,
        type=int,
        help="Number of parallel workers"
    )
    parser.add_argument(
        "--n-trials",
        "-t",
        default=100,
        type=int,
        help="Number of trials to perform for computing the average order parameter"
    )
    start_time = time.time()
    main_thread(parser.parse_args())
    print("Total time taken: {} seconds".format(time.time() - start_time))
```

2. Results

With $p = 7$, I get the following results:

```
$ python3 StochasticHopfieldNetwork.py 7

100% (100 of 100) |####################################| Elapsed Time: 0:00:18 Time:  0:00:18
Average order parameter: 0.855
Total time taken: 19.351770401000977 seconds
```

With $p = 45$, I get the following results:

```
$ python3 StochasticHopfieldNetwork.py 45

100% (100 of 100) |####################################| Elapsed Time: 0:00:19 Time:  0:00:19
Average order parameter: 0.156
Total time taken: 19.952781200408936 seconds
```