IIE4122: Midterm Exam Screenshots

2021190002 Seohyun Jang

Question 1 (python code, prediction results, real-time stock price with timestamp)

```python
# ---- Load data ----
ticker = yf.Ticker("005930.KS")
price_data = ticker.history(period="1d", interval="1m")
n = 200
recent = price_data.tail(n)
prices = recent['Close'].values

# ---- Hyperparameters ----
window_size = 5
hidden_size = 10
param_size = window_size * hidden_size + hidden_size + hidden_size + 1

# ---- Prepare dataset (X: past window_size prices, y: next price) ----
X = []
y = []
for i in range(len(prices) - window_size):
    X.append(prices[i:i+window_size])
    y.append(prices[i+window_size])
X = np.array(X)
y = np.array(y)

# ---- Train/Test Split ----
split_idx = int(0.7 * len(X))
X_train = X[:split_idx]
y_train = y[:split_idx]
X_test = X[split_idx:]
y_test = y[split_idx:]

# ---- Define MLP Model ----
def forward(x, params, input_dim, hidden_dim):
    W1 = params[:input_dim * hidden_dim].reshape(input_dim, hidden_dim)
    b1 = params[input_dim * hidden_dim : input_dim * hidden_dim + hidden_dim]
    W2 = params[input_dim * hidden_dim + hidden_dim : input_dim * hidden_dim + hidden_dim + hidden_dim].reshape(hidden_dim, 1)
    b2 = params[-1]

    hidden = np.dot(x, W1) + b1
    hidden = np.maximum(hidden, 0)  # ReLU
    output = np.dot(hidden, W2) + b2
    return output.squeeze()

# ---- Loss Function (MSE) ----
def mse_loss(params, X, y, input_dim, hidden_dim):
    preds = np.array([forward(x, params, input_dim, hidden_dim) for x in X])
    return np.mean((preds - y)**2)

# ---- Genetic Algorithm Class ----
class GA:
    def __init__(self, population_size, dim, X, y, n_generations, input_dim, hidden_dim):
        self.population_size = population_size
        self.dim = dim
        self.X = X
        self.y = y
        self.n_generations = n_generations
        self.input_dim = input_dim
        self.hidden_dim = hidden_dim

        self.population = [np.random.randn(dim) for _ in range(population_size)]
        self.fitness = [self.evaluate(ind) for ind in self.population]

    def evaluate(self, individual):
        return mse_loss(individual, self.X, self.y, self.input_dim, self.hidden_dim)

    def selection(self):
        idx = np.argsort(self.fitness)
        return [self.population[i] for i in idx[:self.population_size//2]]

    def crossover(self, parent1, parent2):
        crossover_prob = np.random.beta(2, 5)  # Beta distribution
        if random.random() < crossover_prob:
            point = random.randint(1, self.dim-1)
            child1 = np.concatenate((parent1[:point], parent2[point:]))
            child2 = np.concatenate((parent2[:point], parent1[point:]))
            return child1, child2
        else:
            return parent1.copy(), parent2.copy()

    def mutation(self, individual):
        mutation_prob = np.clip(np.random.normal(0.1, 0.05), 0.01, 0.5)  # Normal Distribution
        for i in range(self.dim):
            if random.random() < mutation_prob:
                individual[i] += np.random.normal(0, 0.1)
        return individual

    def evolve(self):
        for gen in range(self.n_generations):
            selected = self.selection()
            next_population = selected.copy()

            while len(next_population) < self.population_size:
                parents = random.sample(selected, 2)
                child1, child2 = self.crossover(parents[0], parents[1])
                next_population.append(self.mutation(child1))
                if len(next_population) < self.population_size:
                    next_population.append(self.mutation(child2))

            self.population = next_population
            self.fitness = [self.evaluate(ind) for ind in self.population]

            if gen % 10 == 0:
                print(f"Generation {gen}, Best Loss: {min(self.fitness):.6f}")

        best_idx = np.argmin(self.fitness)
        return self.population[best_idx]

# ---- Run GA ----
print("GA")
ga = GA(population_size=30, dim=param_size, X=X_train, y=y_train,
        n_generations=100, input_dim=window_size, hidden_dim=hidden_size)
best_params = ga.evolve()

# ---- Train/Test Time Range ----
train_timestamps = recent.index[window_size:split_idx+window_size]
test_timestamps = recent.index[split_idx+window_size:]

train_start_time = train_timestamps[0]
train_end_time = train_timestamps[-1]
test_start_time = test_timestamps[0]
test_end_time = test_timestamps[-1]

print("\nDATA PERIODS")
print(f"Train set: {train_start_time.strftime('%H:%M')} ~ {train_end_time.strftime('%H:%M')}")
print(f"Test set: {test_start_time.strftime('%H:%M')} ~ {test_end_time.strftime('%H:%M')}")


# ---- Test Evaluation with timestamp ----
predictions = np.array([forward(x, best_params, window_size, hidden_size) for x in X_test])

test_timestamps = recent.index[split_idx+window_size:]

print("\nTEST PREDICTION")
print(f"{'Timestamp':<25} {'Real':<15} {'Prediction':<15}")
print("-"*55)
for ts, real, pred in zip(test_timestamps, y_test, predictions):
    print(f"{ts} {real:<15.6f} {pred:<15.6f}")

mse_test = np.mean((predictions - y_test)**2)
absolute_errors = np.abs(predictions - y_test)
mean_absolute_error = np.mean(absolute_errors)

print("\nTest MSE:", round(mse_test, 6))
print("Test Mean Absolute Error:", round(mean_absolute_error, 6))
```

```
GA
Generation 0, Best Loss: 5689942.023401
Generation 10, Best Loss: 19291.138665
Generation 20, Best Loss: 8440.889432
Generation 30, Best Loss: 5288.763019
Generation 40, Best Loss: 5209.068426
Generation 50, Best Loss: 5167.794624
Generation 60, Best Loss: 5162.149554
Generation 70, Best Loss: 5059.708639
Generation 80, Best Loss: 5059.429966
Generation 90, Best Loss: 5059.024253

DATA PERIODS
Train set: 09:05 ~ 10:42
Test set: 10:43 ~ 11:25

TEST PREDICTION
Timestamp                 Real            Prediction
-------------------------------------------------------
2025-04-28 10:43:00+09:00 55600.000000    55598.426376
2025-04-28 10:44:00+09:00 55600.000000    55598.426376
2025-04-28 10:45:00+09:00 55600.000000    55598.426376
2025-04-28 10:46:00+09:00 55650.000000    55598.426376
2025-04-28 10:47:00+09:00 55600.000000    55642.828665
2025-04-28 10:48:00+09:00 55700.000000    55578.809552
2025-04-28 10:49:00+09:00 55600.000000    55670.887991
2025-04-28 10:50:00+09:00 55600.000000    55587.005657
2025-04-28 10:51:00+09:00 55650.000000    55579.483565
2025-04-28 10:52:00+09:00 55650.000000    55698.454523
2025-04-28 10:53:00+09:00 55750.000000    55650.698073
2025-04-28 10:54:00+09:00 55700.000000    55695.673457
2025-04-28 10:55:00+09:00 55750.000000    55639.850449
2025-04-28 10:56:00+09:00 55750.000000    55684.926752
2025-04-28 10:57:00+09:00 55700.000000    55737.278749
2025-04-28 10:58:00+09:00 55750.000000    55676.206799
2025-04-28 10:59:00+09:00 55700.000000    55754.295725
2025-04-28 11:00:00+09:00 55700.000000    55720.362691
2025-04-28 11:01:00+09:00 55750.000000    55695.823623
2025-04-28 11:02:00+09:00 55700.000000    55770.638688
2025-04-28 11:03:00+09:00 55700.000000    55692.549762
2025-04-28 11:04:00+09:00 55800.000000    55682.080507
2025-04-28 11:05:00+09:00 55700.000000    55815.040978
2025-04-28 11:06:00+09:00 55700.000000    55672.932939
2025-04-28 11:07:00+09:00 55700.000000    55665.737544
2025-04-28 11:08:00+09:00 55800.000000    55754.049328
2025-04-28 11:09:00+09:00 55700.000000    55814.714280
2025-04-28 11:10:00+09:00 55800.000000    55659.189823
2025-04-28 11:11:00+09:00 55800.000000    55754.542123
2025-04-28 11:12:00+09:00 55800.000000    55803.620259
2025-04-28 11:13:00+09:00 55800.000000    55742.794707
2025-04-28 11:14:00+09:00 55800.000000    55770.934333
2025-04-28 11:15:00+09:00 55700.000000    55798.420564
2025-04-28 11:16:00+09:00 55800.000000    55709.615986
2025-04-28 11:17:00+09:00 55800.000000    55837.654212
2025-04-28 11:18:00+09:00 55750.000000    55831.106490
2025-04-28 11:19:00+09:00 55850.000000    55698.392417
2025-04-28 11:20:00+09:00 55800.000000    55834.953446
2025-04-28 11:21:00+09:00 55800.000000    55795.146704
2025-04-28 11:22:00+09:00 55800.000000    55754.264673
2025-04-28 11:23:00+09:00 55800.000000    55812.490377
2025-04-28 11:24:00+09:00 55800.000000    55812.163680
2025-04-28 11:25:00+09:00 55800.000000    55798.420564

Test MSE: 4626.737818
Test Mean Absolute Error: 54.001779
```

Question 2

```python
from math import exp, sin
import numpy as np

class AntColony:
    def __init__(
        self, distances,
        num_ants=20, num_best=10, num_iterations=200,
        alpha=1, beta=2, gamma=0.1,
    ):
        self.distances = distances
        self.pheromone = np.ones(self.distances.shape) / len(distances)
        self.all_inds = range(len(distances))
        self.n_ants = num_ants
        self.n_best = num_best
        self.n_iterations = num_iterations
        self.alpha = alpha
        self.beta = beta
        self.gamma = gamma

    def run(self):
        shortest_path = None
        all_time_shortest_path = ("placeholder", np.inf)
        for _ in range(self.n_iterations):
            all_paths = self.gen_all_paths()
            self.spread_pheronome(all_paths, self.n_best, self.gamma)
            shortest_path = min(all_paths, key=lambda x: x[1])
            if shortest_path[1] < all_time_shortest_path[1]:
                all_time_shortest_path = shortest_path
        return all_time_shortest_path

    def spread_pheronome(self, all_paths, n_best, gamma):
        self.pheromone *= (1 - self.gamma)
        sorted_paths = sorted(all_paths, key=lambda x: x[1])
        for path, _ in sorted_paths[:n_best]:
            for move in path:
                self.pheromone[move] += exp(-self.distances[move]) / (1 + sin(self.distances[move]) ** 2)

    def gen_path_dist(self, path):
        total_distance = 0
        for ele in path:
            total_distance += self.distances[ele]
        return total_distance

    def gen_all_paths(self):
        all_paths = []
        for _ in range(self.n_ants):
            path = self.gen_path(0)
            all_paths.append((path, self.gen_path_dist(path)))
        return all_paths

    def gen_path(self, start):
        path = []
        visited = set()
        visited.add(start)
        prev = start
        for _ in range(len(self.distances) - 1):
            move = self.pick_move(self.pheromone[prev], self.distances[prev], visited)
            path.append((prev, move))
            prev = move
            visited.add(move)
        path.append((prev, start))
        return path

    def pick_move(self, pheromone, dist, visited):
        pheromone = np.copy(pheromone)
        pheromone[list(visited)] = 0
        min_distance = np.inf
        move = None
        for i in self.all_inds:
            if i not in visited and dist[i] < min_distance:
                min_distance = dist[i]
                move = i
        return move

if __name__ == "__main__":
    np.random.seed(1)
    num_cities = 60
    city_coords = np.random.rand(num_cities, 2) * 1000
    distances = np.zeros((num_cities, num_cities))
    for i in range(num_cities):
        for j in range(num_cities):
            if i == j:
                distances[i, j] = np.inf
            else:
                distances[i, j] = np.linalg.norm(city_coords[i] - city_coords[j])

    ant_colony = AntColony(distances, num_ants=50, num_best=20, num_iterations=200, alpha=1, beta=2, gamma=0.1)
    shortest_path = ant_colony.run()

    city_names = [f"City {i+1}" for i in range(num_cities)]
    city_order = [shortest_path[0][0][0]]
    for move in shortest_path[0]:
        city_order.append(move[1])

    print("ANT COLONY OPTIMIZATION")
    print(f"\nShortest Path (corresponding distance: {shortest_path[1]:.2f}):")

    full_order = city_order

    for i in range(0, len(full_order), 10):
        line_cities = full_order[i:i+10]
        if i + 10 >= len(full_order):
            if len(line_cities) > 1:
                line = " -> ".join(city_names[idx] for idx in line_cities[:-1])
                line += " -> " + city_names[line_cities[-1]]
            else:
                line = city_names[line_cities[0]]
            print(line)
        else:
            line = " -> ".join(city_names[idx] for idx in line_cities)
            print(line, end=" ->\n")
```

```
ANT COLONY OPTIMIZATION

Shortest Path (corresponding distance: 7559.45):
City 1 -> City 6 -> City 12 -> City 22 -> City 43 -> City 7 -> City 15 -> City 37 -> City 58 -> City 20 ->
City 26 -> City 8 -> City 50 -> City 33 -> City 23 -> City 16 -> City 31 -> City 4 -> City 27 -> City 10 ->
City 56 -> City 3 -> City 14 -> City 25 -> City 38 -> City 48 -> City 29 -> City 28 -> City 60 -> City 18 ->
City 42 -> City 32 -> City 34 -> City 45 -> City 30 -> City 40 -> City 19 -> City 59 -> City 11 -> City 13 ->
City 21 -> City 57 -> City 53 -> City 41 -> City 49 -> City 35 -> City 17 -> City 39 -> City 52 -> City 24 ->
City 36 -> City 47 -> City 9 -> City 5 -> City 51 -> City 2 -> City 54 -> City 46 -> City 44 -> City 55 ->
City 1
```