Проєкт по застосуванню Генетичного алгоритму

Мета проєкту

Метою даного проєкту є розробка та дослідження моделі штучного інтелекту для розпізнавання рукописних цифр на основі датасету DATA_SET із використанням генетичного алгоритму для оптимізації параметрів. У рамках роботи було створено декілька архітектур, зокрема моделі із шаром клітинного автомата (cellular automata) та альтернативні підходи, що не використовують цей шар. Основна задача — порівняти ефективність розпізнавання та швидкодію моделей залежно від використання різних підходів до обробки зображень, зокрема визначити вплив клітинного автомата на якість класифікації у порівнянні з іншими реалізованими варіантами.

1 Огляд Структури Проєкту

```
genetic-algorithm/
 genetic/          # Модуль генетичного алгоритму
 handlers/         # Обробники подій та логування
 loader/           # Завантаження та підготовка даних
 nml/              # Нейромережева бібліотека
 project/          # Основна логіка проєкту
 test.py           # Тестовий файл
```

2 Опис Модулів

2.1 Модуль genetic/

Реалізація генетичного алгоритму.

Основні компоненти та їх алгоритми:

1. **BestSelection** — клас для відбору найкращих особин

   ```
   class BestSelection(Selection):
       def __call__(self, population: list[tuple[Any, float]]):
           sorted_candidates = sorted(population, key=lambda x: x[1])
           return sorted_candidates[: self.population_size]
   ```

   Алгоритм роботи:

   - Сортування популяції за значенням пристосованості (fitness)
   - Вибір top-N особин, де N - розмір популяції
   - Складність: $O(n \log n)$, де n - розмір популяції
   - Забезпечує елітарність - збереження найкращих рішень

2. **ChromosomePipeline** — обробка хромосом

```
class ChromosomePipeline:
    def __init__(self, crossover: Crossover, mutation: Mutation,
                 process_device: Device = Device.CPU):
        self.crossover = crossover
        self.mutation = mutation
        self.process_device = process_device
```

Функціональність:

- Координація операцій схрещування та мутації
- Управління обчислювальними пристроями (CPU/GPU)
- Оптимізація передачі даних між пристроями
- Паралельна обробка хромосом у межах популяції

3. **Crossover** — реалізація схрещування

```
class Crossover:
    def __init__(self, method):
        self.method = method
```

Методи схрещування:

- **Single-point**: Схрещування в одній точці (child = parent1[:point] + parent2[point:])
- **Two-point**: Схрещування у двох точках (child = parent1[:p1] + parent2[p1:p2] + parent1[p2:])
- **Uniform**: Рівномірне схрещування (child[i] = random.choice([parent1[i], parent2[i]]))

4. **GaussianMutation** — мутація за нормальним розподілом

```
class GaussianMutation:
    def __init__(self, sigma: float = 0.1, scale: float = 1.0):
        self.sigma = sigma
        self.scale = scale
```

Алгоритм мутації:

- Генерація шуму за нормальним розподілом $N(0, \sigma^2)$
- Масштабування шуму коефіцієнтом scale
- Додавання шуму до генів: $gene' = gene + noise$
- Адаптивне налаштування параметрів σ та scale

5. **GenomePipeline** — конвеєр обробки геномів

```
class GenomePipeline:
    def __init__(self, selection, pipelines, elitarism_selection):
        self.selection = selection
        self.pipelines = pipelines
        self.elitarism_selection = elitarism_selection
```

Процес обробки:

(a) Відбір батьківських особин

(б) Застосування операторів схрещування

(в) Мутація нащадків

(г) Збереження елітних особин

(д) Формування нового покоління

6. **RouletteSelection** — відбір методом рулетки

```
class RouletteSelection:
    def __call__(self, population: list[tuple[Any, float]])\
-> list[tuple[Any, float]]:
        total_fitness = sum(fitness for _, fitness in population)
        selected = []
        for _ in range(self.population_size):
            pick = random.uniform(0, total_fitness)
            current = 0
            for candidate, fitness in population:
                current += fitness
                if current >= pick:
                    selected.append((candidate, fitness))
                    break
        return selected
```

Принцип роботи:

- Розрахунок загальної пристосованості популяції
- Нормалізація значень пристосованості
- Випадковий вибір особин з ймовірністю, пропорційною їх пристосованості
- Формула ймовірності вибору: $P(i) = \frac{fitness(i)}{total\_fitness}$

7. **Додаткові особливості модуля**:

- Підтримка паралельних обчислень на GPU
- Оптимізація пам'яті через використання представлень тензорів
- Автоматичне налаштування параметрів операторів
- Моніторинг різноманітності популяції
- Адаптивні стратегії еволюції

2.2 Модуль handlers/

Система обробки подій та логування.

Ключові класи:

- PrintHandler — виведення інформації в консоль
- SaveHandler — збереження станів популяції
- TableHandler — створення табличних звітів

2.3 Модуль loader/

Завантаження та керування даними.

Компоненти:

```
class Downloader:
    """Завантаження та підготовка набору даних MNIST"""
    # Обробка файлів даних
    # Валідація завантажених даних
```

Класи:

- DataManager — керування наборами даних
- SklearnBalancedDataLoader — завантажувач збалансованих даних

2.4 Модуль nml/

Нейромережева бібліотека для обробки зображень.

Основні компоненти:

```
class Sequential:
    """Послідовна модель нейромережі"""
    def __init__(self, *layers):
        # Ініціалізація шарів
```

Шари:

- Input — вхідний шар
- CellularAutomata — шар клітинного автомата
- Linear — лінійний шар
- ReLU — шар активації
- Softmax — вихідний шар
- Flatten — шар вирівнювання
- Cast — шар перетворення типів

Опис шарів:

1. **Input Layer**: Вхідний шар приймає зображення розміром 8x8 пікселів у форматі uint8 та нормалізує їх значення до діапазону rule_bitwidth за формулою $x_{norm} = x/255$. Шар також виконує початкову валідацію розмірності вхідних даних та їх типу.

2. **CellularAutomata Layer**
   Клітинний автомат — це дискретна модель, що складається з квадратної сітки клітинок (8 на 8), кожна з яких може мати різний стан. Кожна клітинка змінює свій стан залежно від станів сусідів за певними правилами. Сітка "загортається"по краях, тобто крайні клітинки взаємодіють із протилежними. Клітинки можуть взаємодіяти з різною кількістю сусідів, і для цього в проєкті реалізовано кілька методів пошуку сусідів. Найпростіший — це околиця фон Неймана, де кожна клітинка враховує лише

чотирьох сусідів по вертикалі та горизонталі (зверху, знизу, зліва, справа). Околиця Мура розширює цю ідею, дозволяючи клітинці враховувати ще й діагональних сусідів, тобто всього вісім сусідів навколо. Також використовується розширена околиця, яка включає сусідів на більшій відстані, наприклад, другий рівень по прямим напрямкам, що дає до 12 сусідів. Принцип роботи кожного методу полягає у визначенні індексів сусідніх клітинок відносно поточної, з урахуванням замикання сітки по краях, і подальшому аналізі їхніх станів для прийняття рішення про зміну стану самої клітинки.

3. **Linear Layer**: Повнозв'язний шар, що виконує лінійне перетворення вхідного вектора x за формулою $y = Wx + b$, де W — матриця ваг, b — вектор зміщення. Параметри W та b оптимізуються під час навчання.

4. **ReLU Layer**: Шар активації, що застосовує функцію $f(x) = \max(0, x)$ до кожного елемента вхідного тензора. Дозволяє мережі моделювати складні залежності та запобігає проблемі затухаючих градієнтів.

5. **Softmax Layer**: Вихідний шар, що перетворює вектор логітів z у вектор ймовірностей за формулою $p_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$, де K — кількість класів. Сума ймовірностей всіх класів дорівнює 1.

6. **Flatten Layer**: Перетворює багатовимірний тензор у одновимірний вектор, зберігаючи всі значення для подальшої обробки лінійними шарами.

7. **Cast Layer**: Виконує приведення типів даних між шарами, наприклад, з uint8 у float32, забезпечуючи коректну передачу даних між різними компонентами мережі.

2.5 Модуль project/

Основна логіка проєкту.

Ключові класи:

```
class Manager:
    """Керування генетичним алгоритмом"""
    def __init__(self, sequential, fitness_evaluator, data_manager, ...):
        # Ініціалізація компонентів
        # Налаштування параметрів
```

Компоненти та їх функціональність:

1. **FitnessEvaluator (Оцінка пристосованості)** — реалізує різні метрики оцінки якості рішень.

   - **Accuracy** — частка правильних передбачень серед усіх:

   $$\text{Accuracy} = \frac{\text{Кількість правильних передбачень}}{\text{Загальна кількість}}$$

   - **Cross-Entropy** — логарифмічна функція втрат, яка оцінює відстань між справжніми й передбаченими розподілами ймовірностей:

   $$\text{CE} = -\sum_{i=1}^{n} y_i \log(\hat{y}_i)$$

- **Mean Probability** — середня ймовірність правильного класу серед усіх передбачень:

$$\frac{1}{n}\sum_{i=1}^{n} P(\text{правильний}_i)$$

- **Combined** — комбінована метрика, яка об'єднує точність та середню ймовірність із ваговими коефіцієнтами:

$$\text{Combined} = w_1 \cdot \text{Accuracy} + w_2 \cdot \text{MeanProbability}$$

- **Balanced Accuracy** — зважена точність для класифікацій з незбалансованими класами (враховує точність по кожному класу окремо).

$$\text{Balanced Accuracy} = \frac{1}{K}\sum_{k=1}^{K} \frac{TP_k}{TP_k + FN_k}$$

де K — кількість класів, TP_k — кількість істинно позитивних випадків для класу k, FN_k — хибно негативних.

2. **GenerationHandler (Обробка поколінь)** — система обробки подій життєвого циклу популяції, відстеження прогресу, збереження найкращих особин, аналіз різноманітності, профілювання продуктивності.

3. **Manager (Керування процесом)** — координує всі аспекти еволюції: ініціалізація популяції, оцінка пристосованості, керування генетичними операторами, контроль збіжності.

Процес еволюції:

1. Ініціалізація: створення початкової популяції, налаштування параметрів, підготовка обробників подій

2. Цикл еволюції: оцінка популяції, відбір найкращих, застосування операторів, створення нового покоління, оновлення статистики

3. Завершення: перевірка умов збіжності, збереження найкращих рішень, генерація фінального звіту

Параметри оптимізації:

- Розмір популяції
- Частота мутацій
- Тип відбору

3 Взаємодія Компонентів

3.1 Потік Даних

1. loader → завантаження даних
2. nml → обробка через нейромережу
3. genetic → оптимізація параметрів
4. handlers → логування результатів

4 Використані Технології

- NumPy для матричних обчислень
- CUDA для GPU-прискорення
- Python як основна мова програмування

5 Експерименти та їх результати

Для оцінки ефективності різних архітектур та налаштувань були проведені експерименти. Тут q означає рівень квантизації (rule_bitwidth), а CA — використання шару клітинного автомата.

- **CA 8x8 q2**: Модель з клітинним автоматом, розміром 8x8 та квантизацією 2 рівні демонструє наступні характеристики:
  - Максимальний фітнес досягає близько 0.15-0.17
  - Спостерігається стабільна поведінка протягом всіх поколінь
  - Характерний малий розрив між мінімальним та максимальним значеннями фітнесу
  - Виділяється як найбільш стабільна з усіх досліджених моделей

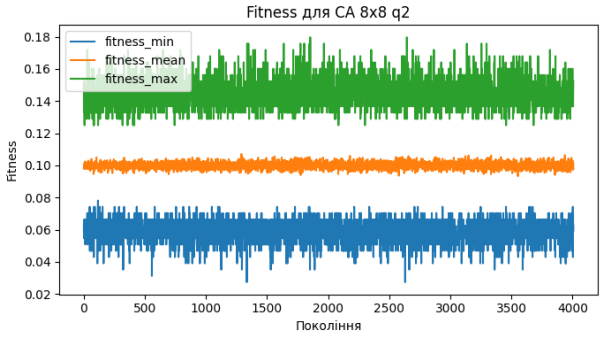

Рис. 1: Графіки fitness для CA 8x8 q2

- **8x8 q2**: Модель без клітинного автомата, з розміром 8x8 та квантизацією 2 рівні показує:
  - Максимальний фітнес близько 0.7-0.73
  - Значний розрив між мінімальним та максимальним фітнесом
  - Нестабільну поведінку протягом навчання
  - Суттєву варіацію у якості розпізнавання

- **28x28 q8**: Модель з високою роздільною здатністю та 8-рівневою квантизацією характеризується:
  - Максимальним фітнесом близько 0.35-0.4
  - Підвищеним початковим рівнем фітнесу
  - Помірним розривом між крайніми значеннями

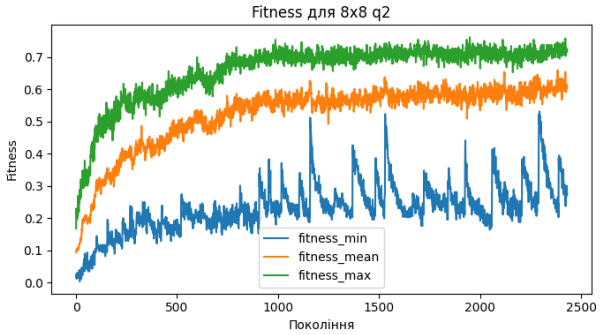

Рис. 2: Графіки fitness для 8x8 q2

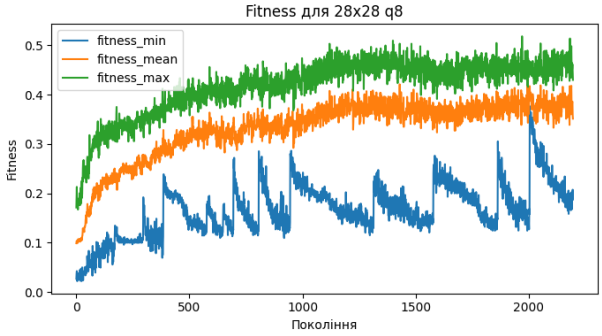

Рис. 3: Графіки fitness для 28x28 q8

– Покращеною деталізацією завдяки високій роздільній здатності

- **8x8 q8**: Модель з малим розміром та високою квантизацією демонструє:
  - Максимальний фітнес близько 0.73-0.74
  - Нестабільну поведінку під час навчання
  - Значний розрив між мінімальним та максимальним значеннями фітнесу

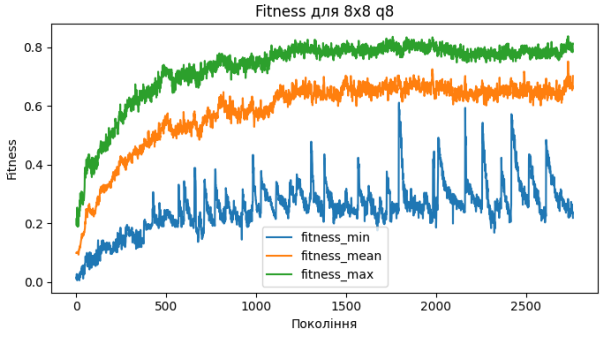

Рис. 4: Графіки fitness для 8x8 q8

5.0.1 Вплив Cellular Automata

- Забезпечує стабільність процесу навчання
- Зменшує варіацію між найкращими та найгіршими результатами
- Може обмежувати максимальну точність через додаткову обробку даних

5.0.2 Вплив розміру зображення

- Більший розмір (28x28) забезпечує кращий початковий фітнес
- Надає більше інформації для аналізу
- Потребує додаткових обчислювальних ресурсів

5.0.3 Вплив квантизації

- Підвищена квантизація (q8) зберігає більше деталей зображення
- Сприяє покращенню максимального фітнесу
- Знижує стабільність моделі

5.1 Висновок

Оптимальний баланс між стабільністю та точністю демонструє модель 28x28 q8, що пояснюється оптимальним співвідношенням між кількістю доступної інформації (висока роздільна здатність у поєднанні з високою квантизацією) та складністю її обробки. Використання клітинного автомата (CA) сприяє стабілізації процесу навчання, однак може створювати обмеження для досягнення максимальної точності класифікації.

Загалом, результати експериментів показали, що використання шару клітинного автомата (CA) суттєво підвищує стійкість та якість класифікації, особливо при низьких рівнях квантизації. Збільшення розміру вхідного зображення та квантизації також може покращити точність, але потребує більше обчислювальних ресурсів і ретельного налаштування для уникнення перенавчання.