Генетичні алгоритми в машинному навчанні

Звіт з проєкту

Розпізнавання рукописних цифр MNIST
за допомогою еволюційних алгоритмів

8 травня 2025 р.

Зміст

1 Огляд проєкту

Даний проєкт впроваджує симуляцію штучного життя за допомогою клітинних еволюційних автоматів (CEA) для розпізнавання рукописних цифр MNIST. Система моделює еволюційні процеси в середовищі на основі сітки, де нейронні мережі еволюціонують за допомогою генетичних алгоритмів. Проєкт спрямований на дослідження поведінки, що виникає, та методів оптимізації, натхненних природною еволюцією.

2 Автомат у проєкті

В основі проєкту лежить концепція **клітинного автомату** — дискретної математичної моделі, що складається з регулярної сітки клітин, кожна з яких може перебувати в одному з скінченної кількості станів. Формально, клітинний автомат можна визначити як систему:

$$A = (S, N, f) \tag{1}$$

де:

- S — множина можливих станів клітини
- N — функція, що визначає окіл клітини (сусідів)
- $f : S^{|N|} \to S$ — функція переходу, що визначає новий стан клітини на основі поточних станів її сусідів

У нашому випадку:

- **Кожна клітина автомату** — це повноцінна нейронна мережа з власними параметрами.
- **Стан клітини** — визначається вагами та біасами нейронної мережі, представленими як вектор параметрів $\theta \in \mathbb{R}^n$.
- **Правила переходу** — визначаються генетичними операторами (селекція, схрещування, мутація), які застосовуються на основі "пристосованості"(fitness) нейронних мереж.
- **Околи клітин** — визначають, які нейронні мережі взаємодіють між собою під час еволюції.

Автомат функціонує ітеративно, де кожна ітерація (покоління) включає:

1. Оцінку продуктивності кожної нейронної мережі в сітці
2. Визначення околів для кожної клітини
3. Вибір батьківських мереж на основі їхньої пристосованості в цих околах
4. Створення нової мережі шляхом схрещування та мутації батьківських мереж
5. Заміну поточної мережі в клітині на новостворену в залежності типу моделі

Особливість нашого автомату полягає в тому, що замість простих символів чи бінарних станів, кожна клітина містить складний об'єкт — нейронну мережу, що здатна розпізнавати рукописні цифри.

3 Архітектура та компоненти

3.1 Основні модулі

1. **Модуль CNN моделей** (`model/cnn.py`):
   - Реалізація згорткової нейронної мережі (CNN)
   - Функції для оцінки точності моделі
   - Методи для отримання та встановлення параметрів для генетичних операцій
   - Функції створення випадкових моделей для ініціалізації
2. **Модуль завантаження даних** (`model/data_load.py`):
   - Завантаження та обробка набору даних MNIST
   - Визначення `train_loader` та `test_loader` для навчання та тестування CNN моделей
3. **Модуль генетичних операторів** (`genetic/operators.py`):

- Реалізація операторів селекції: турнірний, рулетковий, Больцманівський та рангові методи
- Оператори схрещування для комбінування параметрів від батьківських моделей
- Оператори мутації для введення випадкових змін у параметри моделі

4. **Модуль клітинних автоматів** (`automata/fsm.py`):
   - Визначення фреймворку клітинних еволюційних автоматів
   - Реалізація синхронних (SyncCEA) та асинхронних (AsyncCEA) автоматів
   - Підтримка різних типів околів та методів селекції
5. **Веб-додаток** (`app.py`, `static/`):
   - Flask веб-сервер для взаємодії з моделями
   - Користувацький інтерфейс для малювання цифр та отримання передбачень
   - Можливість порівняння різних моделей

4 Ключові особливості

4.1 Нейронні мережі як особи популяції

У проєкті використовуються згорткові нейронні мережі (CNN) як окремі особи в популяції. Кожна CNN має таку архітектуру:

- Вхідний шар для зображень 28×28 пікселів (MNIST формат)
- Згорткові шари з пакетною нормалізацією та ReLU активацією
- Maxpooling шари для зменшення розмірності
- Повнозв'язні шари для класифікації
- Вихідний шар з 10 класами (цифри 0-9)

Рис. 1: Архітектура CNN мережі для розпізнавання MNIST

4.2 Генетичні оператори

Проєкт впроваджує різноманітні генетичні оператори:

4.2.1 Селекція

- **Турнірна селекція**: вибирає переможців серед випадково вибраних груп
- **Рулеткова селекція**: імовірність вибору пропорційна пристосованості
- **Рангова селекція** (лінійна та експоненційна): вибір на основі рангу в популяції

Математично, ймовірність вибору особини i при рулетковій селекції можна представити як:

$$P(i) = \frac{f_i}{\sum_{j=1}^{N} f_j} \tag{2}$$

де f_i — пристосованість особини i, а N — розмір популяції.

4.2.2 Схрещування

- **Стандартне схрещування**: випадковий вибір параметрів від кожного з батьків
- **Одноточкове та двоточкове схрещування**: перемикання між параметрами батьків у визначених точках
- **Blend-схрещування**: зважена комбінація параметрів
- **Mask-схрещування**: використання бінарної маски для вибору параметрів

При Blend-схрещуванні, параметри нащадка θ_c обчислюються як:

$$\theta_c = \alpha \cdot \theta_{p1} + (1 - \alpha) \cdot \theta_{p2} \tag{3}$$

де θ_{p1} та θ_{p2} — параметри батьків, а $\alpha \in [0, 1]$ — коефіцієнт змішування.

4.2.3 Мутація

- Застосування випадкових змін до ваг нейронних мереж з певною ймовірністю
- Регульована сила мутації для балансу дослідження та експлуатації

Математично, мутацію параметра θ_i можна представити як:

$$\theta_i' = \theta_i + \mathcal{N}(0, \sigma^2) \tag{4}$$

де $\mathcal{N}(0, \sigma^2)$ — нормальний розподіл із середнім 0 та дисперсією σ^2.

4.3 Клітинні еволюційні автомати

Проєкт реалізує два типи клітинних еволюційних автоматів:

4.3.1 Синхронний CEA (SyncCEA)

- Оновлює всю сітку одночасно в кожному поколінні
- Підтримує елітарну стратегію для збереження найкращих особин
- Усі клітини автомату змінюють свій стан одночасно на основі стану своїх сусідів у попередньому поколінні

Алгоритм оновлення при синхронному CEA:

4.3.2 Асинхронний CEA (AsyncCEA)

- Оновлює сітку асинхронно, клітина за клітиною
- Дозволяє виникати більш динамічним еволюційним патернам
- Кожна клітина оновлюється на основі поточного стану сусідів, що може включати вже оновлені клітини

Алгоритм оновлення при асинхронному CEA:

Обидва типи підтримують різні шаблони околів, включаючи:

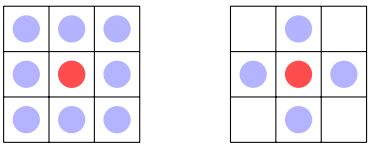
Algorithm 1 Синхронне оновлення CEA

$G \leftarrow$ поточна сітка
$G' \leftarrow$ нова сітка такого ж розміру
$E \leftarrow$ множина елітних клітин (клітини з найвищою пристосованістю)
for кожна позиція (i, j) в сітці **do**
    if $(i, j) \in E$ **then**
        $G'[i, j] \leftarrow G[i, j]$ ▷ Збереження елітних особин
    else
        $N \leftarrow$ отримати сусідів (i, j) в G
        $F \leftarrow$ обчислити пристосованість для кожного сусіда в N
        $(p_1, p_2) \leftarrow$ вибрати батьків з N на основі F і методу селекції
        $c \leftarrow$ схрестити(p_1, p_2)
        $c' \leftarrow$ мутувати(c)
        $G'[i, j] \leftarrow c'$
    end if
end for
return G'

Algorithm 2 Асинхронне оновлення CEA

$G \leftarrow$ поточна сітка
$E \leftarrow$ множина елітних клітин (клітини з найвищою пристосованістю)
for кожна позиція (i, j) в сітці **do**
    if $(i, j) \in E$ **then**
        пропустити ▷ Збереження елітних особин
    else
        $N \leftarrow$ отримати сусідів (i, j) в G
        $F \leftarrow$ обчислити пристосованість для кожного сусіда в N
        $(p_1, p_2) \leftarrow$ вибрати батьків з N на основі F і методу селекції
        $c \leftarrow$ схрестити(p_1, p_2)
        $c' \leftarrow$ мутувати(c)
        $G[i, j] \leftarrow c'$ ▷ Оновлення поточної сітки
    end if
end for
return G

- **Мур (M1, M2)**: усі сусідні клітини
- **Фон Нейман (VN1, VN2)**: сусідні клітини в основних напрямках
- **Власні шаблони околів**: користувацькі конфігурації сусідства



Мур (M1) Фон Нейман (VN1)

Рис. 2: Приклади шаблонів околів (червоний — центральна клітина, сині — сусіди)

5 Реалізація автомату

Наш автомат реалізовано в модулі `automata/fsm.py`, де визначено абстрактний клас `CellularEvolutionaryAutomata` та його конкретні реалізації `SyncCEA` і `AsyncCEA`. Ключові методи включають:

1. **Створення популяції** (`create_grid_population`): ініціалізує сітку нейронних мереж заданого розміру.
2. **Визначення околів** (`get_neighborhood_deltas`, `get_neighborhood`): визначає, які клітини взаємодіють між собою.
3. **Оцінка пристосованості** (`create_fitness_hash_map`): обчислює "пристосованість"кожної нейронної мережі на наборі даних.
4. **Створення нащадків** (`get_child_from_cell`): генерує нову нейронну мережу шляхом селекції, схрещування та мутації.
5. **Створення нового покоління** (`create_next_gen`): оновлює всю сітку відповідно до правил автомату (синхронно або асинхронно).

```
# SyncCEA:

def create_next_gen(self, elite=0.1, ...):
```

```
    new_grid = [[None] * self.width for _ in range(self.height)]
    # ...
                    ...
    for y in range(self.height):
        for x in range(self.width):
            coord = (y, x)
            if coord in top_k_coordinates:
                # ...
                  ...
            else:
                #

                child = self.get_child_from_cell(coord)
                new_grid[y][x] = child
    #
    self.grid = new_grid

# AsyncCEA:

def create_next_gen(self, elite=0.1):
    # ...

                                    ...
    for y in range(self.height):
        for x in range(self.width):
            coord = (y, x)
            if coord in top_k_coords_set:
                continue
            #

            child = self.get_child_from_cell(coord)
            self.grid[y][x] = child
            # ...
                                        ...
```

Лістинг 1: Різниця між синхронним та асинхронним автоматами

6 Технології та інструменти

Проєкт використовує наступні технології:

6.1 Бекенд

- Python 3.12
- PyTorch для реалізації нейронних мереж
- Flask для веб-сервера
- NumPy для обробки даних

6.2 Фронтенд

- HTML/CSS/JavaScript
- Canvas API для малювання цифр
- Chart.js для візуалізації результатів

7 Впровадження та використання

7.1 Навчання моделей

Проєкт підтримує два підходи до навчання:

7.1.1 Традиційне навчання CNN

- Використання оптимізатора Adam
- Налаштування швидкості навчання та кількості епох

7.1.2 Еволюційне навчання через CEA

- Налаштування розміру сітки (популяції)
- Вибір типу околу та методу селекції
- Налаштування кількості поколінь

Команда для навчання на синхронному CEA:

```
python main.py --method genetic --synchronous --grid_size 5 \
    --neighborhood "[[0,1,0],[1,2,1],[0,1,0]]" \
    --selection rank_exponential --genetic_epochs 100
```

7.2 Веб-інтерфейс

Проєкт включає веб-інтерфейс, який дозволяє:

- Малювати цифри на полотні
- Отримувати прогнози від різних моделей
- Порівнювати продуктивність моделей CNN, SyncCEA та AsyncCEA
- Візуалізувати розподіл імовірностей для кожної цифри

Рис. 3: Веб-інтерфейс для розпізнавання рукописних цифр

8 Результати та оцінка

З аналізу коду та наявних даних можна зробити висновок про результати проєкту:

Модель	**Точність**	**Час навчання**
CNN (Adam)	99.26%	Найшвидший
SyncCEA	44.65%	Найдовший
AsyncCEA	42.91%	Середній

Табл. 1: Порівняння моделей за основними показниками

9 Потенційні застосування та перспективи

Проєкт має широкий спектр потенційних застосувань та напрямків для подальшого розвитку:

9.1 Оптимізація нейронних мереж

- Автоматичний пошук оптимальних архітектур нейронних мереж
- Еволюційний підхід до налаштування гіперпараметрів

9.2 Моделювання складних систем

- Дослідження поведінки, що виникає в розподілених системах
- Моделювання біологічних процесів та систем

9.3 Навчання без градієнтів

- Альтернативні методи для задач, де градієнтні методи неефективні
- Паралельне та розподілене навчання

9.4 Подальші напрямки розвитку

- Впровадження більш складних архітектур нейронних мереж
- Гібридні підходи, що поєднують градієнтне та еволюційне навчання
- Розширення на інші набори даних та задачі, окрім MNIST

10 Висновки

Проєкт успішно демонструє застосування клітинних еволюційних автоматів для навчання нейронних мереж. Хоча традиційні методи (CNN з Adam) показують кращу точність та ефективність для розпізнавання MNIST, еволюційні підходи (SyncCEA та AsyncCEA) показують обнадійливі результати та відкривають нові можливості для дослідження.

Особливо цікавими є можливості асинхронних CEA для моделювання більш природних еволюційних процесів, які можуть бути корисними для більш складних задач або динамічних середовищ.

Проєкт закладає основу для подальших досліджень на перетині штучного життя, еволюційних алгоритмів та глибоких нейронних мереж.