Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Aproksymacja funkcji celu algorytmu ewolucyjnego aproksymatorem liniowym.

Uczenie maszynowe 2022L

Michał Brus, Michał Rejer

17 kwietnia 2022

Spis treści

1 Opis problemu **3**

2 Algorytm ewolucyjny **3**

2.1 Cel . . . 3
2.2 Ogólny opis działania . . . 3
2.3 Pseudokod algorytmu . . . 3
2.4 Przyjęte założenia . . . 4
2.5 Generacja zbiorów treninigowych . . . 4

3 Aproksymator liniowy **5**

3.1 Metoda najmniejszych kwadratów . . . 5
3.2 Generacja modelu . . . 5
3.3 Przykład obliczeniowy . . . 6
3.4 Podział dziedziny na części . . . 6
3.5 Szukanie minimum modelu . . . 7

4 Ocena jakości modeli **7**

4.1 Plan eksperymentów . . . 7
4.2 Określenie dziedziny . . . 7
4.3 Walidacja modelu . . . 7

5 Implementacja **7**

1 Opis problemu

Projekt polega na aproksymacji funkcji celu algorytmu ewolucyjnego aproksymatorem liniowym, dla każdej z 30 funkcji z CEC 2017. Jego celem będzie wyznaczanie minimów badanych funkcji oraz określenie jak często zaimplementowany aproksymator uzyskuje lepsze wyniki od AE. Dodatkowym zagadnieniem będzie również ocena jakości generowanych modeli.

2 Algorytm ewolucyjny

2.1 Cel

W projekcie zostanie wykorzystany algorytm ewolucyjny. Będzie użyty do znalezienia minimów funkcji CEC oraz będzie generatorem zbiorów danych uczących.

2.2 Ogólny opis działania

Idea algorytmu ewolucyjnego wywodzi się z ewolucji naturalnej. Rozpoczyna się od wylosowania punktów w przestrzeni, które staną się populacją początkową. Oceniania jest ich wartość i zapamiętany zostaje najlepszy punkt. Następnie zachodzi selekcja, podczas której z populacji wybierane są te punkty, które mają większe prawdopodobieństwo reprodukcji. Te punkty zostają rodzicami i (opcjonalnie) rozpoczyna się krzyżowanie części z nich. Następnie dzieci (nowe punkty z krzyżowania) oraz rodzice bez dzieci podlegają mutacji. Wygenerowane w ten sposób punkty są ponownie ocenianie i najlepszy z nich jest zapisywany jeśli jest lepszy niż najlepszy do tej pory. Na końcu zachodzi sukcesja podczas, której zachodzi decyzja, które osobniki ze starej populacji i nowych punktów przejdą do następnej generacji. Proces jest powtarzany, aż do zakończenia warunku pętli (zazwyczaj liczba generacji).

2.3 Pseudokod algorytmu

Data : $q(x), P_0, \mu, \sigma, p_m, p_c, t_{max}$
Result: $\hat{x}^*, \hat{o}^*$
$t \leftarrow 0$
$o \leftarrow$ ocena(q, P_0)
$\hat{x}^*, \hat{o}^* \leftarrow$ znajdz najlepszego (P_0, o)
while *kryterium stop np.* $t < t_{max}$ **do**
  $R \leftarrow$ reprodukcja (P_t, o, μ)
  $M \leftarrow$ operacje genetyczne (R, σ, p_m, p_c)
  $o_m \leftarrow (q, M)$
  $x_t^*, o_t^* \leftarrow$ znajdz najlepszego (M, o_m)
  if $o_t^* <= \hat{o}^*$ **then**
    $\hat{o}^* \leftarrow o_t^*$
    $\hat{x} \leftarrow x_t^*$
  $P_{t+1}, o \leftarrow$ sukcesja (P_t, M, o, o_m)
  $t \leftarrow t + 1$

Parametry

- $q(x)$ - funkcja celu, którą optymalizujemy
- P_0 - populacja początkowa
- μ - liczba osobników w populacji
- σ - siła mutacji. Mutacja polega na wygenerowaniu punktu z otoczenia punktu bieżącego, tak, że bardziej prawdopodobna jest generacja punktu bliskiego. Najczęściej stosuje się mutację gausowską,

gdzie do aktualnej wartości **x** dodaje się wektoe liczb losowych z rozkładu losowego przeskalowany przez $\boldsymbol{\sigma}$ - siłę mutacji:

$$x = x + \sigma * N(0,1) \tag{1}$$

- p_m - prawdopodobieństwo mutacji. Dla danego punktu jeśli $U(0,1) < p_m$ to mutujemy ten punkt.
- p_c - prawdopodobieństwo krzyżowania. Dla każdej pary punktów, jeśli $U(0,1) < p_c$ to stosujemy krzyżowanie np. z 2 rodziców powstaje 2 dzieci. Jeśli krzyżowanie nie zachodzi to rodzice przechodzą dalej.
- t_{max} - maksymalna liczba iteracji (pokoleń)

2.4 Przyjęte założenia

W niniejszej implementacji algorytmu ewolucyjnego przyjęto następujące założenia:

- Krzyżowanie jest opcjonalne (przy włączonym 2 rodziców = 2 dzieci)
- Zachodzi selekcja turniejowa o rozmiarze 2. Populacja jest sortowana względem jakości (najlepszy ma indeks 1). Losujemy z niej (ze zwracaniem) z rozkładem jednostajnym pary osobników, które "toczą ze sobą pojedynek". Prawdopodobieństwo wygranej, czyli reprodukcji zależy od jakości osobnika, ale też jego szansy na bycie wylosowanym:

$$p_s(P(t,j)) = \frac{1}{\mu^S}((\mu - j + 1)^S - (\mu - j)^S) \tag{2}$$

  gdzie S to rozmiar turnieju (tu 2), t - iteracja populacji, j - j-ty osobnik z populacji

- Zmiana populacji z $t \rightarrow t+1$ odbywa się przez sukcesję elitarną 1-osobnikową. Oznacza to, że ze starej populacji najlepszy osobnik zostaje a z M (potencjalna nowa populacja) wchodzą wszyscy. Najgorszy z sumy nie przechodzi.

$$P_{t+1} = \{\text{k najlepszych z } P(t)\} \cup M \setminus \{\text{k najgorszych z tego połączonego zbioru}\} \tag{3}$$

- Rozmiar populacji 20
- Liczba iteracji $\geqslant 500$

2.5 Generacja zbiorów treninigowych

Populacja początkowa P_0 po wylosowaniu i ocenieniu jej zostanie przekazana jako pierwszy zbiór treningowy do aproksymatora liniowego. Każda nowa populacja z AE zostanie dołożona do obecnego już zbioru treningowego (np. 20, 40, 60 ...).

$$U = \bigcup_{j=1}^{t} U_j \tag{4}$$

$$U_j = \{\langle x_i, y_i \rangle : i = 1, ..., \mu\} \tag{5}$$

$$y_i = q(x_i) \tag{6}$$

3 Aproksymator liniowy

3.1 Metoda najmniejszych kwadratów

Aproksymator liniowy dla funkcji $f : \mathbb{R}^n \to \mathbb{R}$ opisanej równaniem 8. będzie przyjmował postać hiperpłaszczyzny $\hat{f}$ o wymiarze n opisanej równaniem 11.

$$\mathbf{x} = [x_1, x_2, \ldots, x_n] \tag{7}$$

$$f(\mathbf{x}) = f(x_1, x_2, \ldots, x_n) \tag{8}$$

$$\hat{f}(\mathbf{x}) = \hat{f}(x_1, x_2, \ldots, x_n) \tag{9}$$

$$\mathbf{a} = [a_0, a_1, \ldots, a_n]^T \tag{10}$$

$$\hat{f}(\mathbf{x}) = a_0 + \sum_{i=1}^{n} a_i x_i = [1, \mathbf{x}]\mathbf{a} = 0 \tag{11}$$

Wektor współczynników $\mathbf{a}$ będzie szacowany przy użyciu metody najmniejszych kwadratów (MNK) dla zbioru treningowego. Zadaniem optymalizacji będzie minimalizacja wskaźnika opisanego równaniem 15.
$p = 0, 1, \ldots, k$ - liczność zbioru trenującego
$\mathbf{x}_k = [x_{k,1}, \ldots, x_{k,n}]$ - argument funkcji f ze zbioru trenującego,
$\mathbf{y}$ - wektor zbierający wartości funkcji f dla kolejnych elementów zbioru trenującego.

$$M = \begin{bmatrix} 1 & x_{0,1} & \cdots & x_{0,n} \\ 1 & x_{1,1} & \cdots & x_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{k,1} & \cdots & x_{k,n} \end{bmatrix} = \begin{bmatrix} 1 & \mathbf{x}_0 \\ 1 & \mathbf{x}_1 \\ \vdots & \vdots \\ 1 & \mathbf{x}_k \end{bmatrix} \tag{12}$$

$$\mathbf{y} = [y_0, y_1, \ldots, y_k] \tag{13}$$

$$\mathbf{y} = Ma \tag{14}$$

$$E(\mathbf{a}) = \frac{1}{k} \sum_{i=0}^{k} \|[1, \mathbf{x}_k]\mathbf{a} - f(\mathbf{x}_k)\|^2 \tag{15}$$

Rozwiązanie zadania minimalizacji ma postać podaną równaniem 16.

$$\mathbf{a} = (M^T M)^{-1} M^T \mathbf{y}^T \tag{16}$$

3.2 Generacja modelu

Po wygenerowaniu w danej iteracji modelu z wykorzystaniem aproksymatora liniowego dla każdego punktu liczony jest błąd dopasowania aproksymatora ze wzoru numer 17. Jeżeli w punkcie, dla którego błąd dopasowania jest największy, zostanie przekroczony dopuszczalny próg błędu następuje podział danego fragmentu dziedziny na dwie części. Dla każdej z części parametry aproksymatora wyliczane są ponownie, a błędy porównywane z progiem, aż do uzyskania satysfakcjonującego dopasowania modelu.

$$e(\mathbf{x_k}) = \|[1, \mathbf{x}_k]\mathbf{a} - f(\mathbf{x}_k)\|^2 \tag{17}$$

3.3 Przykład obliczeniowy

Przykład dla funkcji $f : \mathbb{R} \to \mathbb{R}$. Wyznaczone zostały punkty postaci $(x, f(x))$:

$$\{(1, 0.13820), (2, 0.2857), (3, 0.3964), (4, 0.4241), (5, 0.6179), (6, 0.6452)\} \tag{18}$$

Po zastosowaniu wzoru 16. wyznaczone został wektor współczynników:

$$\mathbf{a} = [0.1017, 0.0619] \tag{19}$$

Funkcja aproksymująca ma więc postać:

$$\hat{f}(x) = 0.1017 + 0.0619x \tag{20}$$

Uzyskany model generuje punkty $(x, \hat{f}(x))$ odpowiadające aproksymowanym danym:

$$\{(1, 0.1637), (2, 0.2654), (3, 0.3671), (4, 0.4688), (5, 0.5705), (6, 0.6722)\} \tag{21}$$

Co przekłada się na wektor błędów:

$$[0.0006, 0.0004, 0.0009, 0.0020, 0.0022, 0.0007] \tag{22}$$

Największy błąd wystąpił dla $x = 5$. Zakładając, że przekroczony został dopuszczalny próg błędu dopasowania aproksymatora, dziedzina w $x = 5$ zostanie podzielona na dwa przedziały, a w nich powstaną dwie nowe funkcje aproksymujące: $\hat{f}_1$ i $\hat{f}_2$. Opisane kroki będą powtarzane, aż do uzyskania złożenia liniowych aproksymatorów, dla których błąd dopasowania dla każdego elementu ze zbioru trenującego będzie akceptowalny lub podział nie będzie możliwy.

3.4 Podział dziedziny na części

W ogólności hiperpłaszczyznę o wymiarze n definiuje $n+1$ punktów, dlatego podział fragmentu dziedziny na części ma jedynie sens, gdy w każdej z części znajdzie się co najmniej n+1 punktów, gdzie n będzie wymiarem dziedziny aproksymowanej funkcji.

Przestrzeń o wymiarze n na dwie części dzieliła będzie hiperpłaszczyzna o wymiarze $n - 1$. Przyjmując, że fragmenty modelu będą wyznaczane dla wcześniej wydzielonych części dziedziny D, można zaproponować kilka sposobów dalszego podziału dziedziny aproksymowanej funkcji. Zakładając podział na dwie części tylko jednego fragmentu D w jednej iteracji. można spróbować dokonać podziału według jednej ze współrzędnych punktu.
Np. na początku największy błąd przekraczający próg został zidentyfikowany w punkcie: $\mathbf{x_j} = [1, 0, 0, 0, 0]$. Dokonujemy podziału dziedziny na dwie części według pierwszej współrzędnej:

- część pierwsza: $x_1 < 1$
- część druga: $x_1 \geqslant 1$

Drugi punkt podziału to: $\mathbf{x_{j+1}} = [0, 1, 0, 0, 0]$
Tym razem zmieniamy współrzędną dzielącą fragment dziedziny. Punkt $\mathbf{x_{j+1}}$ znalazł się we wcześniej wydzielonej części pierwszej, dlatego teraz ona podlega podziałowi:

- część pierwsza: $x_1 < 1 \wedge x_2 < 1$
- część druga: $x_1 < 1 \wedge x_2 \geqslant 1$
- część trzecia: $x_1 \geqslant 1$

Punkt, w którym dokonano podziału można włączyć do jednej z części dziedziny powstałych po podziale. Jeśli w powstałych częściach pozostała niewystarczająca liczba punktów, można zmienić współrzędną, po której dokonywany jest podział. Natomiast, jeżeli podział będzie niemożliwy można sprawdzić próg błędu dopasowania w innym fragmencie dziedziny.

Jest to propozycja rozwiązania problemu, która zostanie zweryfikowana w trakcie realizacji projektu, i ewentualnie może zostać zmieniona.

3.5 Szukanie minimum modelu

Dla modelu w formie aproksymarów liniowych na fragmentach dziedziny, minimum powinno znajdować się albo na przecięciu poszczególnych fragmentów modelu albo dla ograniczenia dziedziny.

4 Ocena jakości modeli

4.1 Plan eksperymentów

Dla każdej funkcji CEC algorytm ewolucyjny zostanie uruchomiony 30 razy, aby wyciągnąć średnie z otrzymanych wyników. Otrzymane minima w kolejnych generacjach będą porównane z minimami z aproksymatora liniowego (lepszy wynik pogrubiony w tabeli). Następnie zostanie policzony ich stosunek i sprawdzony w jakim procencie model okazuje się lepszy od AE.

4.2 Określenie dziedziny

Dziedzina, dla której zostaną przeprowadzane eksperymenty, będzie 10-wymiarowa, a jej zakres $[-x_{max}, x_{max}]$ dobrany w trakcie testów. Aby sprawdzić poprawność implementowanego algorytmu, zostanie on sprawdzony w niższych wymiarach (w celu lepszej wizualizacji).

4.3 Walidacja modelu

Jakość modelu będzie sprawdzana na 2 sposoby:

- Porównanie przybliżonego minimum funkcji, otrzymanym z modelu z faktycznym minimum.
- Sprawdzenie średniej straty modelu w k-krotnej (tutaj 3) walidacji krzyżowej. Średnia strata modelu ma postać wektora błędów (równanie (22)). Zbiór trenujący jest podzielony na 3 części, gdzie jedna z nich jest zbiorem testowym (T), a dwie pozostałe służą do uczenia (U).

Input : $U \neq \emptyset$: zbiór par uczących, k - liczba podziałów
Result: $\hat{f}, e$
Sortujemy zbiór U w losowej kolejności, a następnie dzielimy na k równych części.
for *i=1, ..., k* **do**
  $\hat{f}_i \leftarrow$ Uczymy model na zbiorze $U - U_i$
  $e_i \leftarrow$średnia strata modelu $\hat{f}_i$ na zbiorze U_i
$e \leftarrow \frac{1}{k} \sum_{i=1}^{k} e_i$
$\hat{f} \leftarrow$ Uczymy model na zbiorze U

---

5 Implementacja

Projekt zostanie zaimplementowany w języku Python przy wykorzystaniu dodatkowych bibliotek (numpy, matplotlib etc.). Wykorzystane do optymalizacji zostaną funkcje z repozytorium na githubie - CEC 2017.