AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej

Projekt dyplomowy

Optymalizacja hiperparametrów w głębokich sieciach
Hyperparameters optimization in deep networks.

Autor: *Jacek Tyszkiewicz*
Kierunek studiów: *Automatyka i Robotyka*
Opiekun pracy: *dr hab. inż. Wojciech Chmiel*

Kraków, 2025

Serdeczie dziękuję moim rodzicom, którzy zawsze mnie wspierają, promotorowi dr. hab. inż. Wojciechowi Chmielowi za dobór interesującego tematu i włożony wysiłek.

Spis treści

1. Wprowadzenie

Optymalizacja hiperparametrów w sieciach głębokich polega na stworzeniu architektury sieci neuronowej, która będzie odpowiednia dla wybranego zestawu danych. W niniejszej pracy przetestowano najpopularniejsze algorytmy optymalizacji. Testów dokonano na trzech zbiorach danych. Opracowano autorski algorytm optymalizacji hiperparametrycznej, który porównano z najpopularniejszymi algorytmami. Wynikiem pracy jest aplikacja AutoML system, która optymalizuje hiperparametry w automatyczny sposób.

1.1. Pojęcia podstawowe

Na samymy początku pracy, omówię różnicę między dwoma pojęciami: hiperparametry, parametry. Zdarza się, że oba pojęcia są mylnie nazywane parametrami. Jednak ich semantyczna interpretacja jest zupełnie inna.

- **Hiperparametry** są wartościami, które definiujemy przed procesem uczenia i nie zmieniają swoich wartości podczas tego procesu.
- **Parametry** są wartościami, które również definiujemy przed procesem uczenia, ale ich wartości ulegają zmianie podczas tego procesu.

Przykładowe hiperparametry dla sieci neuronowej to: ilość neuronów w danej warstwie, liczba warstw. Dokonujemy także wyboru: funkcji aktywacji w danej warstwie, schematu walidacji krzyżowej, algorytmu optymalizacji wag sieci neuronowej oraz metryki, której wartości chcemy optymalizować. Natomiast parametray dla sieci neuronowej to np. wagi połączeń, przesunięcia (ang. biases).

Parametry oraz hiperparametry pełnią odmienną rolę w modelach sztucznej inteligencji. Dla naszego przykładu, dzięki aktualizacji wag połączeń, model uczy się wzorców ze zbioru uczącego. Natomiast hiperparametry są odpowiedzialne za stworzenie architektury sieci neuronowej. Hiperparametry są także odpowiedzialne za eliminację wysokiego błędu obciążenia (ang.high bias) i wysokiej wariancji (ang.high variance).

Niekorzystnym zjawiskiem jest sytuacja, w której zbiór uczący jest prosty, natomiast model jest skomplikowany. Taki model będzie podatny na przeuczenie, co może prowadzić do sytuacji, w której nie będzie poprawnie generalizował (high variance). Natomiast sytuacja odwrotna, w której model jest zbyt prosty, a dane zawierają skomplikowane wzorce, również jest niekorzystna z punktu widzenia procesu uczenia, ponieważ model nie będzie w stanie nauczyć się danych (ang.high bias). Proces doboru odpowiednich hiperparametrów jest odpowiedzialny za wprowadzenie kompromisu, pomiędzy obciążeniem (bias), a wariancją (wariance).

Wykres efektywnego wymiaru (ang. effective dimension) obrazuje jak przeszukiwanie w danym kierunku wpływa na poprawienie wartości funkcji celu. W procesie uczenia, szczególnie przy ograniczonych zasobach obliczeniowych, ważne jest, aby w pierwszej kolejności przeszukiwać kierunki w których stopień oczekiwanej poprawy jest największy. Wartość ukazującę metrykę wydajnościową, dla odpowiednich próbek (tj. nauczonych modeli), możemy odczytać z wykresy efektywnego wymiaru.

Typ hiperparametru - hiperparametr może być zmienną typu ciągłego (np. współczynnik uczenia), dyskretnego (liczba neuronów w danej warstwie), kategorycznego (np. schemat walidacji krzyżowej).

Funkcja kosztu optymalizacji hiperparametrów

$$\lambda^{(*)} \approx \arg\min_{\lambda \in \Lambda} \text{mean}_{x \in \mathcal{X}^{(\text{valid})}} \mathcal{L}\left(x; \mathcal{A}_\lambda\left(\mathcal{X}^{(\text{train})}\right)\right) \tag{1}$$

$$\equiv \arg\min_{\lambda \in \Lambda} \Psi(\lambda) \tag{2}$$

$$\approx \arg\min_{\lambda \in \{\lambda^{(1)}, \ldots, \lambda^{(S)}\}} \Psi(\lambda) \equiv \hat{\lambda} \tag{3}$$

Źródło: J.Bergstra i wsp. [1]

gdzie:

- $\mathcal{A}_\lambda\left(\mathcal{X}^{(\text{train})}\right)$ - model wytrenowany na zbiorze $\mathcal{X}$.
- $\lambda^{(*)}$ - optymalny hiperparametr, który minimalizuje funkcję kosztu.
- $\mathcal{L}(x; \mathcal{A}_\lambda(\mathcal{X}))$ - funkcja kosztu dla punktu x i modelu $\mathcal{A}_\lambda$ wytrenowanym na zbiorze $\mathcal{X}$.
- $\Psi(\lambda)$ - funkcja odpowiedzi zależna od: algorytmu, hiperparametrów, zbioru danych, metryki.
- $\hat{\lambda}$ - najlepsza wartość hiperparametru λ wybrana z losowo wybranych próbnych punktów $\{\lambda^{(1)}, \ldots, \lambda^{(S)}\}$.

1.2. Etapy doboru hiperparametrów

Poczas doboru hiperparametrów, pierwszym etapem jest storzenie przestrzeni hiperparametrów (ang. Hyperparameter space). Przestrzeń hiperparparametryczna zawiera kierunki, które zamierzamy przeszukiwać oraz zakresy wartości tych kierunków. Następnym etapem jest wybranie metody walidacji krzyżowej oraz metryki którą chcemy minimalizować bądź maksymalizować. Na końcy wybieramy metodę do próbkowania wartości z przestrzeni hiperparametrycznej.

1.3. Omówienie problemu doboru hiperparametrów

Znalezienie odpowiednich wartości hiperparametrów w praktyce jest trudne, ponieważ nie można zdefiniować matematycznej formuły w celu znalezienia hiperparametrów. Oznacza to, że gradientowe algorytmy optymalizacji nie mogą zostać użyte do optymalizacji hiperparametrycznej. Należy więc spróbować różnych kombinacji i ocenić model. Nie oznacza to jednka, że nie istnieją inteligentne metody próbkowania w przestrzeni hiperparametrycznej. Wykorzystując statystykę i rozkład normalny jesteśmy w stanie wybrać kolejność hiperparametrów do sprawdzenia. Sam proces uczenia może trwać długo (zależy to między innymi od skomplikowania modelu, wielkości zbioru uczącego, wyboru schematu walidacji krzyżowej). Trzeba więc opracować taki algorytm, który będzie możliwie jak najlepiej wybierał następne próbki hiperparametrów, aby możliwie jak najszybciej uzyskać zbieżność. Można też skorzystać z gotowych algorytmów, które są zaimplementowane w bibliotekach pythona, takich jak: Hyperopt, Optuna, scikit-opt, Keras Tuner.

1.4. Schemat walidacji krzyżowej, a optymalizacja hiperparametryczna

Użycie schematu walidacji krzyżowej jest ważnym etapem optymalizacji hiperparametrycznej. Najczęstszym, a także podstawowym schematem walidacji krzyżowej w wielu bibliotekach (Optuna, Hyperopt) jest k-krotna walidacja krzyżowa (ang. KFold cross-validation). W przypadku, gdy nie przeprowadzalibyśmy walidacji krzyżowej, dobrane hiperparametry mogłyby być odpowiednie dla danego podziału tj. zbioru treningowego i testowego, ale dla innego istniałyby inne, lepsze hiperparametry. Wówczas nastąpiłby wyciek danych ogólnej struktury zbioru testowego i treningowego. Natomiast wprowadzając walidację krzyżową otrzymujemy ocenę, która jest średnią wszystkich ocen z różnych podziałów zbiorów przeprowadzonych w procesie walidacji krzyżowej. Również warto zauważyć, iż schematy walidacji krzyżowej znacząco

wpływają na czas uczenia modelu np. schemat k-krotnej walidacji krzyżowej generalnie jest dużo szybszy niż walidacja krzyżowa z opuszczaniem P elementów (ang. Leave-P-Out cross-validation), ponieważ potrzebuje na ogół mniej iteracji.

1.5. System automatycznego uczenia maszynowego (Automated Machine Learning)

Aby stworzyć algorytmy bazujące na metodach AI, które będą przeprowadzały poprawną predykcję, należy wybrać odpowiedni model ze zbioru dostępnych modeli uczenia maszynowego, a następnie dobrać do niego hiperparametry. Istnieją usługi chmurowe, nazywane AutoML, które wykonują te czynności automatycznie. Należy zaimportować dane, a na wyjściu dostaniemy model z dobranymi hiperparametrami, nauczony i gotowy do predykcji. Jedną z tych usług jest IBM Cloud.

2. Optymalizacja hiperparametryczna - teoria

W tym rozdziale omówię najpopularniejsze grupy algorytmów wykorzystywane do optymalizacji hiperparametrycznej.

2.1. Algorytmy z grupy wyczerpujących poszukiwań (ang. Exhaustive Search)

Grupa algorytmów wyczerpujących poszukiwań (ang. Exhaustive Search) zawiera trzy algorytmy optymalizacji hiperparametrycznej. Są to:

- **Wyszukiwanie ręczne (ang. Manual Search)**
- **Wyszukiwanie siatką (ang. Grid Search)**
- **Wyszukiwanie losowe (ang. Random Search)**

Opracowanie teoretyczne algorytmów wykonano na podstawie [1].
Wspólną cechą tych algorytmów jest to, że przeszukują przestrzeń w nieukierunkowany sposób, tzn. że nie uczą się z poprzednich iteracji (oprócz Manual Search). Są to najprostrze i najczęściej używanie algorytmy.

2.1.1. Wyszukiwanie ręczne

Wyszukiwanie ręczne jest metodą polegającą na doborze hiperparametrów przez programistę. Nie może być nazwana metodą nieukierunkowaną, ponieważ osoba odpowiedzialna za wybór hiperparametrów uczy się podczas kolejnych iteracji algorytmu. Jest to metoda, która nie ma postaci implementacyjnej. Czy jest więc metodą prostą w realizacji? Niekoniecznie, wynika to z faktu, iż osoba odpowiedzialna za dobór powinna mieć wyobrażenie, jak dany hiperparametr wpływa na model. Jest to więc metoda polegająca na przeprowadzeniu serii eksperymentów.

Zalety	Wady
Dla doświadczonego dewelopera połączenie tej metody z innymi metodami może pomóc skrócić czas eksperymentu.	Trudno jest zgadnąć dobrą wartość hiperparametru, nawet dla kogoś, kto naprawdę rozumie, jak działa model.
	Przeprowadzanie wyszukiwania ręcznego samodzielnie jest czasochłonne.

Tabela 2.1. Ręczne wyszukiwanie: zalety i wady [1]

2.1.2. Wyszukiwanie siatką

Wyszukiwanie siatką polega na zdefiniowaniu skończonej przestrzeni hiperparametrycznej, a następnie, dla każdej kombinacji hiperparametrów uczymy model i zapisujemy wartości metryki wydajnościowej. Jest to metoda prosta w implementacji, wystarczy wykorzystać w tym celu pętle zagnieżdżone. Ale istnieją gotowe biblioteki, które zrobią to za nas np. Optuna, Hyperopt.

Zalety	Wady
Bardzo łatwa do zaimplementowania od podstaw	Przekleństwo wymiarowości (COD)
Pozwala przetestować wszystkie możliwe kombinacje	Możliwość pominięcia lepszych kombinacji hiperparametrów poza zdefiniowaną przestrzenią wyszukiwania

Tabela 2.2. Wyszukiwanie siatką: zalety i wady [1]

COD (ang. Curse of Dimensionality) dodanie kolejnej wartości do przestrzeni hiperparametrów spowoduje eksponencjalny wzrost czasu eksperymentu.

2.1.3. Wyszukiwanie losowe

Wyszukiwanie losowe polega na losowym próbkowaniu hiperparametrów z przestrzeni hiperparametrycznej. W tym algorytmie, definiujemy liczbę iteracji oraz przestrzeń hiperparametrów. Zmienne w przestrzeni hiperparametrycznej mogą być typu ciągłego, a wybór wartości zmiennej odbywa się na podstawie wybranego rozkładu (np.jednostajnego).

Zalety	Wady
W porównaniu do wyszukiwania siatką, ta metoda jest bardziej wydajna pod względem kosztów obliczeniowych i uzyskania optymalnych hiperparametrów.	Powoduje wysoką wariancję podczas procesu.
Dobrze sprawdza się w odkrywaniu nieoczekiwanych, optymalnych kombinacji hiperparametrów.	Czasami potrzeba więcej czasu, aby uzyskać optymalne hiperparametry.

Tabela 2.3. Wyszukiwanie losowe: zalety i wady [1]

Mogłoby się wydawać, że algorytmy dokonujące losowego przeszukiwania są rzadziej wykorzystywane i gorsze w porównaniu do "inteligentnych algorytmów", takich jak algorytmy Bayesowskie czy heurystyczne. Nic bardziej mylnego, algorytmy wyczerpujących przeszukiwań mają tę przewagę nad algorytmami "inteligentnego poszukiwania", że mogą być wykonywane równolegle. Natomiast algorytmy Bayesowskie oraz heurystyczne nie mogą być przetwarzane równolegle.

2.2. Algorytmy Bayesowskie (ang. Bayesian Algorithms)

Generalnie algorytmy Bayesowskie można zaliczyć do grup:

– **algorytmów sekwencyjnego przeszukiwania (ang. Sequential Search lub SMBO)**. Jak sama nazwa wskazuje, algorytmy te działają w sposób sekwencyjny tzn. na podstawie kilku próbek, algorytm podejmuje decyzje o wyborze następnej próbki. Jest to spory minus algorytmów Bayesowskich, ponieważ uniemożliwia to wykonywanie ich równolegle. Użycie algorytmów „inteligentnego“ próbkowania ma sens wtedy, gdy proces próbkowania zajmuje mniej czasu niż proces oceny modelu.

– **algorytmów optymalizacji bezgradientowej** jest to grupa algorytmów, która jest zdolna do globalnej optymalizacji funkcji black-box, która nie przyjmuje żadnych form funkcjonalnych.

– **algorytmów informowanego wyszukiwania (ang. informed search)**, czyli grupy w której algorytmy uczą się na podstawie wcześniejszych iteracji.

Opracowanie teoretyczne algorytmów wykonano na podstawie [1], [2].

2.2.1. BOGP

BOGP jest algorytmem optymalizacji bayesowskiej. Wykorzystuje on procesy Gaussa (ang. Gaussian processes) do modelowania kosztownej w obliczeniach funkcji celu.

Opis słowny algorytmu:
W pierwszym kroku, algorytm wybiera hiperparametry ze zdefiniowanej przestrzeni hiperparametrów w sposób losowy (np.5 próbek), następnie wylicza dla nich wartość funkcji celu. Wybrane hiperpametry i wyliczona dla nich funkcja celu nazywamy rozkładem priori. Następnie algorytm tworzy model zastępczy funkcji celu. Model zastępczy funkcji celu nazywamy rozkład posteriori. Do utworzenia modelu zastępczego, algorytm wykorzystuje procesy Gaussa oraz rozkład priori. Na podstawie przygotowanego modelu, funkcja akwizycji (ang. acquisition function) wybiera następną próbkę do oceny i algorytm przechodzi do następnej iteracji, czyli aktualizacji rozkładu priori itd. Po wykonaniu zadanej liczby iteracji, algorytm wybiera najlepsze hiperparametry, które zostały zbadane i przyucza na nich model.

W odniesieniu do reguły Bayesa, algorytm ten można zapisać jako [1]:

$$P(\Theta \mid \text{data}) = \frac{P(\text{data} \mid \Theta) \cdot P(\Theta)}{P(\text{data})} \tag{4}$$

gdzie:

- $P(\Theta \mid \text{data})$ to rozkład a posteriori
- $P(\text{data} \mid \Theta)$ to model wiarygodności
- $P(\Theta)$ to rozkład a priori
- $P(\text{data})$ to stała, zapewniająca, że wartości wyniku jest w przedziale $[0, 1]$ (pomijana w algorytmie BOGP)

Reguła Bayesa pozwala na dynamiczną alokację przekonań w świetle nowych danych, dzięki czemu jest szeroko stosowana w analizie danych i modelowaniu predykcyjnym.

Wyjaśnienie pojęć matematycznych - kluczowych do zrozumienia sposobu tworzenia modelu zastępczego za pomocą Procesów Gaussa [2],[3]:

1. Wariancja i odchylenie standardowe są miarą rozproszenia danych dla konkretnej zmiennej.

2. Kowariancja to miara wspólnego prawdopodobieństwa np. dwóch zmiennych losowych lub miara zależności między tymi dwoma zmiennymi, miara korelacji. Jeżeli:

   – cov(x1,x2)=0 - zmienne nie są skorelowane

   – cov(x1,x2) > 0 - im większe x1, tym większe x2

   – cov(x1,x2) < 0 - im większe x1, tym mniejsze x2

3. Wzory matematyczne dla podpunktu pierwszego i drugiego:

$$\text{var} = \frac{\sum(x_i - \bar{x})^2}{N-1} \tag{5}$$

$$\sigma = \sqrt[2]{var} \tag{6}$$

$$\text{cov}_{x,y} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{N-1} \tag{7}$$

   – $\text{cov}_{x,y}$ - kowariancja zmiennych x, y

   – x_i - wartość danych dla x

   – y_i - wartość danych dla y

   – $\bar{x}$ - średnia z x

   – $\bar{y}$ - średnia z y

   – N - liczba wartości danych

   – var - wariancja zmiennej x (lub y)

   – σ - odchylenie standardowe zmiennej

4. Wielowymiarowy rozkad gaussa jest opisany przez:

   – macierz kowariancji - która zawiera wariancję każdej z tych zmiennych na diagonalnej (czyli kowariancję każdej zmiennej z samą sobą) oraz kowariancję między dwiema zmiennymi na przecięciu wiersza i kolumny dla tych zmiennych. Jest to macierz kwadratowa oraz symetryczna.

   – średnią - wektor n wymiarowy

5. Procesy gaussa to rozkład prawdopodobieństwa dla funkcji, jest to swego rodzaju uogólnienie wielowymiarowego rozkładu gaussa na nieskończenie wiele zmiennych losowych. Procesu gaussa używamy do oszacowania prawdopodobieństwo funkcji celu. Dzięki nim jesteśmy w stanie rozważać właściwości wysokiego poziomu funkcji, które mogłyby dopasować nasze dane.

Algorytm BOGP realizuje kroki w następujący sposób (omówienie konceptu matematycznego) [2]:

1) Na początku określamy właściwości wysokiego poziomu funkcji (rozkład prior), są to nasze przewidywanie co do funkcji, nie muszą być poprawne. Podczas tego etapu, powinniśmy zadać sobie pytania:czy te funkcje szybko się zmieniają, czy są okresowe lub obejmują warunkową niezależność.

2) Gdy uwzględnimy dane, możemy użyć prior do wywnioskowania rozkładu posterior dla funkcji, które mogą pasować do danych.

Właściwości procesu Gaussa, które wykorzystujemy do dopasowania danych, są ściśle kontrolowane przez tzw. funkcję kowariancji, znaną również jako jądro (kernel). Przykładem funkcji kowariancji może być funkcja radialna:

$$k_{\text{RBF}}(x, x') = \text{Cov}(f(x), f(x')) = a^2 \exp\left(-\frac{1}{2\ell^2}\|x - x'\|^2\right)$$

- $k_{\text{RBF}}(x, x')$ — funkcja jądra radialnego (RBF) - mierzy podobieństwo między punktami x i x'.
- $\text{Cov}(f(x), f(x'))$ — kowariancja między wartościami funkcji f w punktach x i x'.
- a^2 — parametr skali - kontroluje wielkość wariancji (kontroluje pionową skalę, w której funkcja się zmienia).
- $\exp\left(-\frac{1}{2\ell^2}\|x - x'\|^2\right)$ — część wykładnicza - decyduje o stopniu podobieństwa punktów x i x'.
- ℓ — parametr długości (zwany też „długością skali") - kontroluje zakres wpływu punktu na sąsiednie punkty. Im większa wartość ℓ, tym wolniej zanika kowariancja w miarę wzrostu odległości między punktami x i x'.

– $\|x - x'\|^2$ — kwadrat odległości euklidesowej między punktami x i x'.

Omówienie hiperparametrów dla RBF:
Analizując postać funkcji RBF, możemy dojść do wniosku, że gdy punkty są oddalone od siebie na więcej niż ℓ, to wartość funkcji w tych punktach stają się nieskorelowane. Oznacza to, że największy wpływ na prognozę mają punkty, które są odległe na mniej niż ℓ. Więc jądro kowariancji, pełni rolę miary prawdopodobieństwa dla przewidywanego punktu, która jest oparta na najbliższych punktach. Ważne jest aby poprawinie dobrać parametr ℓ, ponieważ jeżeli długość będzie zbyt mała, model będzie reagował tylko na lokalne dane. Natomiast przy zbyt dużej wartość, model utraci zdolność uchwycenia lokalnych szczegółów.

Skoro omawiany algorytm ma służyć optymalizacji hiperparametrów w siecicach głębokich, a sam zawiera hiperparametry, to jak je dobrać? Otóż możemy zrobić to w automatyczny sposób, przy użyciu funkcji wiarygodności brzegowej (ang. marginal likelihood). Metoda ta polega na maksymalizacji logarytmicznej funkcji prawdopodobieństwa brzegowego:

$$\log p(\mathbf{y}|\theta, X) = -\frac{1}{2}\mathbf{y}^\top \left[K_\theta(X,X) + \sigma^2 I\right]^{-1} \mathbf{y} - \frac{1}{2}\log|K_\theta(X,X)| + c \tag{8}$$

Innymi słowy, maksymalizacja logarytmicznej funkcji prawdopodobieństwa brzegowego sprowadza problem do poszukiwania takiego θ, które najlepiej dopasuje model GP do danych uczących X i y (gdzie θ to wektor zawierający hiperparametry jądra np. dla RBF, θ składa się z ℓ oraz a). Optymalizacji tej funkcji możemy dokonać przy użyciu algorytmów gradientowych.

Jak już wcześniej zaznaczone, proces gaussa jest uogólnieniem wielowymiarowego rozkładu Gaussa, możemy więc opisać go za pomocą macierzy kowariancji, opartej na jądrze kowariancji oraz za pomocą wektora wartości średnich. Wektor średnich μ tego rozkładu jest dany przez funkcję średniej (ang. mean function), którą zazwyczaj przyjmuje się jako stałą lub równą zero.

$$\begin{bmatrix} f(x) \\ f(x_1) \\ \vdots \\ f(x_n) \end{bmatrix} \sim \mathcal{N}\left(\mu, \begin{bmatrix} k(x,x) & k(x,x_1) & \cdots & k(x,x_n) \\ k(x_1,x) & k(x_1,x_1) & \cdots & k(x_1,x_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(x_n,x) & k(x_n,x_1) & \cdots & k(x_n,x_n) \end{bmatrix}\right) \tag{9}$$

Równanie 9 określa rozkład GP prior. Wykorzystując go, możemy obliczyć warunkowy rozkład $f(x)$ dla dowolnego x dla znanych $f(x_1), \ldots, f(x_n)$, czyli wartości funkcji, które wyznaczyliśmy na podstawie oceny modelu za pomocą funkcji celu (metryki). Wyznaczony rozkład warunkowy nazywamy rozkładem posteriori i właśnie tego rozkładu używamy do

dokonania predykcji.

Rozkład posteriori, można zapisać jako:

$$f(x) \mid f(x_1), \ldots, f(x_n) \sim \mathcal{N}(m, s^2) \tag{10}$$

wektor średnich algorytm można wyliczyć ze wzoru:

$$m = k(x, x_{1:n})k(x_{1:n}, x_{1:n})^{-1} f(x_{1:n}) \tag{11}$$

gdzie:

- $k(x, x_{1:n})$ jest wektorem $1 \times n$ kowariancji między nowym punktem x a obserwacjami $x_{1:n}$.
- $k(x_{1:n}, x_{1:n})$ jest macierzą $n \times n$ kowariancji między obserwacjami.
- $f(x_{1:n})$ jest wektorem obserwowanych wartości funkcji.

wariancję algorytm wylicza korzystając ze wzoru:

$$s^2 = k(x, x) - k(x, x_{1:n})k(x_{1:n}, x_{1:n})^{-1} k(x, x_{1:n}) \tag{12}$$

gdzie:

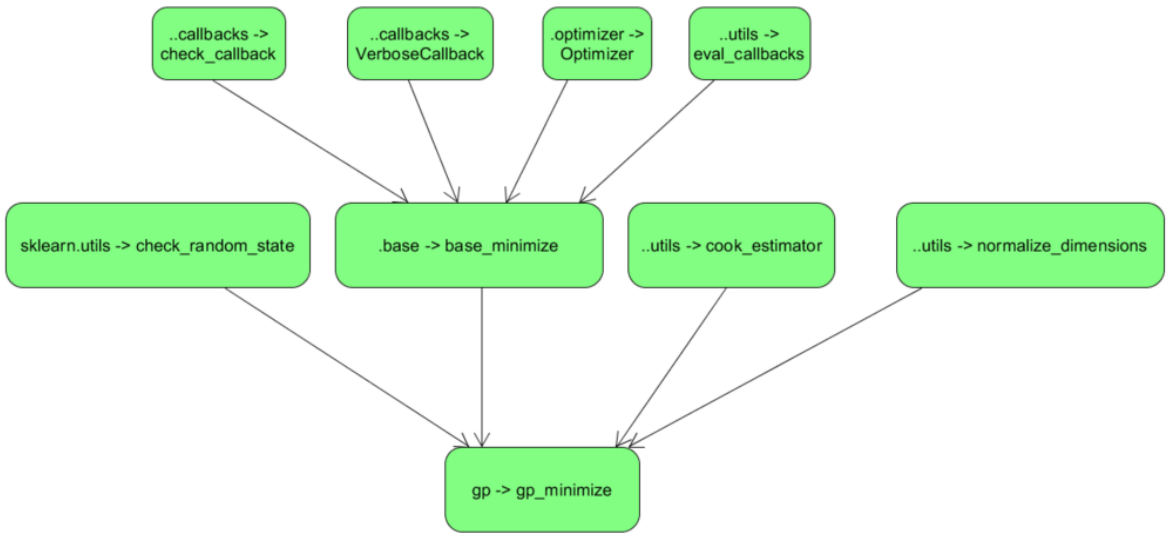
- $k(x, x)$ jest wariancją w punkcie x.
- Wyrażenie $k(x, x_{1:n}) \left[k(x_{1:n}, x_{1:n})\right]^{-1} k(x_{1:n}, x)$ reprezentuje redukcję niepewności w predykcji $f(x)$ dzięki informacjom z obserwacji.

Wzory te wynikają bezpośrednio z własności wielowymiarowego rozkładu normalnego. Jeśli chcemy stworzyć przedział, w którym z 95 % prawdopodobieństwem znajdzie się funkcja f(x), możemy wziąść przedział $m \pm 2s$.
Warto też zauważyć, że za pomocą równań 9-11 możemy wyznaczyć przewidywania dla dowolnego zestawu punktu x.

2.2.2. Analiza implementacji algorytmu optymalizacji Bayesowskiej z biblioteki scikit-optimize

Analizę implementacji algorytmu optymalizacji Bayesowskiej przerowadzono na podstawie kodu źródłowego oraz dokumentacji biblioteki scikit-optimize [4].Wykonana analiza została wykorzystana podczas tworzenia własnej implementacji autorskiego algorytmu, w której elementy optymalizacji Bayesowskiej zostały wykorzystane.

Schemat zależności funkcjonalnych algorytmu optymalizacji Bayesowskiej:



Rys. 2.1. Materiały własne: Schemat I zależności funcjonalnych

Opis bloków ze schematu I zależności funkcjonnalnych (Rys.2.1):

– gp -> gp_minimize - plik główny zawierający funkcję główną - gp_minimize, która jest odpowiedzialna za inicjalizalizację oraz zarządzanie procesem optymalizacji Bayesowskiej,

– sklearn.utils -> check_random_state - funkcja odpowiedzialna za inicjalizacja generatora losowego,

– .base -> base_minimize - Plik “base” zawiera funkcję base_minimize, która jest odpowiedzialna za zarządzanie optymalizacją,

– ..utils -> cook_estimator - funkcja przygotowuje estymator (np. estymator procesu gausa) do użycia w procesie optymalizacji,

– ..utils -> normalize_dimensions - Funkcja odpowiedzialna za normalizację wymiarów przestrzeni poszukiwań np. do [0,1],

– ..callbacks -> check_callback - funkcja sprawdzający czy callbacki zostały poprawie podane,

– ..callbacks -> VerboseCallback - callback używany do wyświetlania szczegółów podczas optymalizacji,

– .optimizer -> Optimizer - Klasa reprezentująca operacje optymalizacji bayesowskiej,

– ..utils -> eval_callbacks - Funkcja uruchamiająca callbacki w trakcie optymalizacji.

Opis słowny plików:

1. gp - plik główny zawierający funkcję główną - gp_minimize, który jest odpowiedzialnym za inicjalizalizację oraz zarządzanie procesem optymalizacji Bayesowskiej. Pełni on rolę koordynatora, który integruje ze sobą pliki odpowiedzialne za realizację poszczególnych funkcjonalności algorytmu. W pliku zdefiniowana jest funkcja gp_minimize, której parametry stanowią interfejs podczas doboru hiperparametrów. Interfejs jest opisany w dokumentacji [5].

   Kroki algorytmu gp_minimize:

   1. rng = check_random_state(random_state) - inicjalizacja generatora losowego.
   2. normalizacja przestrzeni przeszukiwań za pomocą normalize_dimensions().
   3. inicjalizacja base estimator przez użytkownika, jeżeli nie został on wcześniej określony. Domyślnie na procesy Gaussa, za pomocą funkcji cook_estimator.
   4. return base_minimize (parametry) - uruchomienie optymalizacji bayesowskiej dla zdefiniowanych parametrów w interfejsie użytkownika, które zostały uprzednio znormalizowane - punkty 1-3.

2. base - Plik “base” zawiera funkcję base_minimize, która jest odpowiedzialna za zarządzanie optymalizacją, dostaje ona paremetry od funkcji gp_minimize. Nazwy parametrów są identyczne jak w pliku gp.

   Plik base_minimize zawiera:
   Główną pętlę optymalizacji, która jest wykonywana n_calls razy:

   1. wybieramy następny punkt za pomocą metody ask() obiektu optimizer tj. next_x = optimizer.ask()
   2. wyliczamy funkcję celu dla następnego punktu tj. next_y = func(next_x)

3. aktualizujemy obiekt result o wybraną próbkę result = optimizer.tell(next_x, next_y)
4. Atrybutowi specs obiektu result zostaje przypisana wartość specs (specs przechowuje lokalne argumenty)
5. Jeżeli jakiś callback wymusi zatrzymanie przerywamy pętlę i zwracamy wynik result

Plik gp oraz base_minimize są odpowiedzialne za interfejs użytkownika oraz obsługę i realizację kolejnych kroków algorytmu Bayesowskiego.

3. Plik optimizer, w którym zawarto kolejne kroki optymalizacji Bayesowskiej zawiera klasę optimizer, posiadającą metody:

   – metoda ask i _ask - wybiera punkt z przestrzeni poszukiwań, który następnie zostać oceniony przez funkcję celu. Jej zadaniem jest utrzymanie balansu między eksploracją, a eksplatacją. Do tego wykorzystuje mechanizm Constant Liar (CL), który polega na tworzeniu kopii optymalizatora i „oszukiwaniu" go, poprzez przypisanie mu sztucznych wartości do zbioru punktów, na podstawie których tworzony jest model zastępczy. Istnieją trzy strategie wyboru sztucznej wartości dla następnych punktów cl_min (wybór minimum z dotychczasowych wartości funkcji celu), cl_mean (wyliczenie średniej z dotychczasowych wartości funkcji celu), cl_max (wybór maksimum z dotychczasowych wartości funkcji celu). Domyślnie w metodzie ask użyta jest strategia cl_min. Jednak liczba fałszywych punktów w metodzie ask, domyślnie jest ustawiona na None, a podczas wywołania nie jest zmieniana tzn. ten mechanizm nie jest użyty dla algorytmu BOGP. Metoda ask w fazie początkowej w losowy sposób wybiera punkty inicjalizacji, zgodnie z liczbą podaną przez użytkownika w API lub losowo wybiera punkt, gdy model zastępczy nie został jeszcze wybrany. W przeciwym razie punkt do oceny zostaje pobrany z poprzedniego wykonania metody _tell(), która wybiera następny punkt przy użyciu funkcji akwizycji. Sprawdzany jest też czy odległość od punktów jest mniejsza niż $1e-8$, wtedy wysyłany jest warning.

   – metoda tell i _tell - metoda tell podobnie jak metoda ask pełni rolę interfejsu tzn. sprawdza czy punkt znajduje się w zdefiniowanej przestrzeni hiperparametrów, sprawdza zgodność typów danych x i y oraz obsługuje specjalne funkcje akwizycji tj. zawierające "ps"w swojej nazwie - przekształca czas obliczeń w logarym naturalny w celu poprawnego skalowania dla tych funkcji. Natomiast metoda _tell jest

odpowiedzialna za aktualizucje stanu optymalizatora. Doadaje punkt x oraz odpowiadającą mu wartość funkcji celu do zbioru danych, aktualizuje model zastępczy i wybiera następny punkt do oceny. Wyboru dokonuje na podstawie funkcji akwizycji, domyślnie gp_hedge.

- metoda copy - odpowiedzialna za kopiowanie obiektu optimizer, ale bez przyuczania do danych. Wykorzystane np. podczas strategii "CL".
- metoda _check_y_is_valid odpowiada za sprawdzenie poprawności i spójności danych wejściowowych.
- metoda run - odpowiedzialna za wykonanie pętli optymalizacyjnej. Jednak w pliku base_minimize utworzona jest inna pętla optymalizacyjna, aby zachować większą kontrolę nad procesem. Więc metoda run nie uczestniczy w procesie optymalizacji.
- metoda get_result - funkcja pomocnicza, która umożliwia uzyskanie rezultatów optymalizacji w aktualnym stanie optymalizatora.

To w pliku optimizer realizowane są kolejne kroki optymalizacji Bayesowskiej. Jednak do tego pliku importowana jest klasa odpowiedzialna za stworzenie pliku zastępczego - cook_estimator.

Cook_estimator dla modelu oprateg o procesy gaussa tworzy model GP oparty o jądro Matern z parametrem $\nu = 2.5$, a dla przestrzeni kategorycznej Hamming Kernel.

W pliku optimizer zdefiniowane są funkcje akwizycji umożliwiające wybór następnego punktu do oceny na podstawie modelu zastępczego. Dostępne funkcje akwizycji to: "gp_hedge", "EI", "LCB", "PI", "EIps", "PIps". Gdzie: gp_hedge to strategia polegająca na wyborze jedenej strategi podczas kolejnych iteracji z pośród EI, LCB, PI. Wybór dokonywany jest na podstawie prawdopodobieństwa, które jest obliczane z zysków (zmienna gains_), które mierzą skuteczność każdej strategii w poprzednich iteracjach. Pozostałe strategie są opisane w literaturze [1].

2.2.3. SMAC

SMAC należy do grypy algorytmów optymalizacji Bayesowskiej . W odróżnóżnieniu od BOGP dla którego modelelem zastępczym są procesy gaussa, dla SMAC rolę modelu zastępczego pełni random forest. SMAC jest metodą zoptyamlizowaną do dyskretnych lub kategorycznych hiperparametrów. Generalnie poza tą różnicą SMAC działa analogicznie do BOGP [1] .

Las losowy jest to model uczenia maszynowego w którego skład wchodzi pewna grupa drzew decyzyjnych. Model ten zawiera kilkanaście hiperparametrów. [6]. Hiperparametry te można dostroić alogorytmem lasu losowego. W tym wypadku używamy drugiego modelu tego samego typu do poprawy wydajności modelu pierwszego. Jest to możliwe, ponieważ pierwszy model traktujemy jako model zastępczy, podczas gdy drugi jest rzeczywistym modelem dopasowanym do zmiennych niezależnych w celu przewidywania zmiennej zależnej. [1]

2.2.4. BOinG

BOinG (Bayesian Optimization inside a Grove) to podejście w optymalizacji bayesowskiej, które łączy dwa modele zastępcze: Random Forest (RF) na poziomie globalnym oraz Gaussian Process (GP) na poziomie lokalnym. [7]

Algorytm: Dwuetapowa optymalizacja bayesowska z BOinG

1. **Wejście:** funkcja typu „czarna skrzynka” f; przestrzeń poszukiwań $\mathcal{X}$; modele predykcyjne $\hat{f}_g$ i $\hat{f}_l$, które odpowiednio dopasowują globalne i lokalne rozkłady obserwacji; funkcje akwizycji α_g i α_l; budżet ewaluacyjny T.

2. **Wyjście:** globalny minimizer f:

$$x^* \in \arg\min_{x \in \mathcal{X}} f(x)$$

3. **Inicjalizacja:** Zainicjuj dane $\mathcal{X}^{(0)}$ początkowymi obserwacjami.

4. **Dla** $t = 1, 2, \ldots, T$ **wykonuj:**

   (a) Dopasuj globalny model $\hat{f}_g^{(t)}$ do danych $\mathcal{X}^{(t-1)}$.

   (b) Wybierz kandydacki punkt x_g za pomocą globalnej funkcji akwizycji α_g:

$$x_g \in \arg\max_{x \in \mathcal{X}} \alpha_g(x; \mathcal{X}^{(t-1)}, \hat{f}_g^{(t)})$$

(c) Wyekstrahuj podregion $\mathcal{X}_{\text{sub}} \subseteq \mathcal{X}$ na podstawie globalnego kandydata x_g i modelu $\hat{f}_g^{(t)}$.

(d) Dopasuj lokalny model $\hat{f}_l$ z punktami wewnątrz podregionu $\mathcal{X}_i^{(t-1)} \in \mathcal{X}_{\text{sub}}$ oraz z punktami spoza podregionu:

$$\mathcal{X}_o^{(t-1)} = \mathcal{X}^{(t-1)} \setminus \mathcal{X}_i^{(t-1)}$$

(e) Określ ostateczny punkt próbkowania na podstawie lokalnej funkcji akwizycji α_l:

$$x^{(t)} \in \arg\max_{x \in \mathcal{X}_{\text{sub}}} \alpha_l(x; \mathcal{X}_i^{(t-1)}, \mathcal{X}_o^{(t-1)}, \hat{f}_l^{(t)})$$

(f) Sprawdź $y^{(t)} := f(x^{(t)})$.

(g) Zaktualizuj dane:

$$\mathcal{X}^{(t)} \leftarrow \mathcal{X}^{(t-1)} \cup \{(x^{(t)}, f(x^{(t)}))\}$$

5. **Koniec.**

6. **Zwróć:**

$$x^* \in \arg\min_{x \in \mathcal{X}} f(x)$$

Algorytm przedstawia ogólną ideę BOinG. W każdej iteracji najpierw trenujemy globalny model zastępczy $\hat{f}_g$ na wszystkich obserwacjach $X^{(t-1)}$, aby oszacować możliwe regiony, które warto eksplorować. Używając funkcji akwizycji α_g na $\hat{f}_g$, wybieramy obiecującą konfigurację, która będzie kierować wyborem podregionu. Następnie nowy lokalny model $\hat{f}_l$ jest dopasowywany do obserwacji w wybranym podregionie $X_{\text{in}}^{(t-1)}$.

Na podstawie $\hat{f}_l$, maksimum lokalnej funkcji akwizycji decyduje o następnej konfiguracji do oceny na rzeczywistej funkcji kosztu f.

Aby podejście było możliwe do zastosowania w bardziej złożonych problemach z bardziej rozsądnymi budżetami ewaluacyjnymi T, model globalny musi skalować się do wielu obserwacji.

Z tego też względu w algorytmie BOinG modelem globalnym jest las losowy, a lokalnym Procesy Gaussa (GP).

2.3. Algorytmy heurystyczne(ang. Heuristic Search)

Algorytmy heurystyczne szukają suboptymalnego rozwiązania problemu poprzez próbkowanie przestrzeni rozwiązań, podczas, której każdy realizuje odpowiednią strategię. Przyjęta strategia ma za zadanie wprowadzić balans pomiędzy eksploracją nowych, obiecujących obszarów rozwiązań, a eksplatacją obszarów potencjalnie oznaczonych jako dobrze prognozujące.

2.3.1. Symulowane Wyżarzanie

Fizyczna inspiracja do stworzenia algorytmu:
Algorytm symulowanego wyżarzania jest inspirowany procesem wyżarzania metalu. Podczas tego procesu metal jest nagrzewany do bardzo wysokiej temperatury na pewien czas. Ma to na celu zwiększenie swobody w poruszania się atomów metalu, podczas tego procesu atomy dążą do lepszej konfiguracji. Następnie następuje schłodzenie w wyniku, którego uzyskujemy krystaliczną strukturę. Dzięki temu procesowi metal zwiększa swoją wytrzymałość. Rozdział opracowano na podstawie[1]

Koncepcja algorytmu symulowanego wyżarzania:
Algorytm symulowanego wyżarzania startuje z losowo wybranego punktu przestrzeni hiperparametrów. Parametrem sterującym jest temperatura, która wraz ze wzrostem licznika iteracji ulega spadkowi (chłodzenie metalu). Spadek temperatury, powoduje zmniejszenie tolerancji na akceptację gorszych rozwiązań od aktualnego. Za akceptację obecnie przetwarzanego rozwiązania odpowiedzialne jest kryterium akceptacji. Wybór następnego punktu do oceny dokonujemy w sposób losowy, spośród sąsiedztwa aktualnie rozważanego punktu. Kryterium stopu algorytmu może być spełnienie zdefiniowanych założeń lub czas obliczeń.

$$AC(T, \Delta f) = \begin{cases} \exp\left(-\frac{\Delta f}{T}\right) & \text{jeśli kandydujący punkt jest gorszy niż aktualny punkt,} \\ 1 & \text{w przeciwnym razie.} \end{cases} \tag{13}$$

gdzie:
$\Delta f = |f(\text{obecny\_punkt}) - f(\text{obecny\_punkt})|$,

Sterowanie temperaturą (chłodzenie metalu) algorytm może dokonać postępując zgodnie z poniższymi strategiami [1]:

– **Chłodzenie geometryczne:** W tej strategi temperatura jest zmniejszana przez współczynnik chłodzenia $0 < \alpha < 1$. W chłodzeniu geometrycznym początkowa temperatura T_0

jest mnożona przez α^{iter}, gdzie *iter* to aktualna liczba iteracji:

$$T = \alpha^{iter} \cdot T_0$$

– **Chłodzenie liniowe:** Zmniejszanie temperatury następuje w sposób liniowy za pomocą współczynnika chłodzenia β. Wartość β jest wybierana w taki sposób, aby T miało nadal dodatnią wartość po t_f iteracjach. Na przykład:

$$\beta = \frac{T_0 - T_f}{t_f}$$

Gdzie T_f to oczekiwana końcowa temperatura po t_f iteracjach:

$$T = T_0 - \beta \cdot iter$$

– **Cauchego SA:** Strategia chłodzenia działa poprzez zmniejszanie temperatury proporcjonalnie do aktualnej liczby iteracji, $iter$:

$$T = \frac{T_0}{iter}$$

Zalety	**Wady**
Posiada wszystkie zalety, jakie ma losowe przeszukiwanie.	Posiada wszystkie wady, jakie ma losowe przeszukiwanie.
Zdolność do większego skupienia na częściach przestrzeni, które prawdopodobnie zawierają optymalne hiperparametry.	Może pominąć części przestrzeni, które zawierają optymalne hiperparametry.
	Wyższe koszty obliczeniowe z powodu konieczności obliczania AC w każdej próbie.

Tabela 2.4. Porównanie zalet i wad.

2.3.2. Algorytm Genetyczny

Inspiracją do stworzenia algorytmu genetycznego była teoria Charlesa Darwina. Teoria zakładała ewolucję przez dobór naturalny tzn. wszystkie gatunki ewoluują w miarę upływu czasu w wyniku zmian w ich cechcach dziedzicznych.

Koncepcja algorytmu genetycznego:
Algorytm genetyczny jest oparty na populacji tzn. startujemy z losowo stworzonej populacji. W skład populacji wchodzą obiekty o określonych cechach, a siłę obiektu stanowi wyliczona dla niego funkcja celu. W kolejnej iteracji algorytm wykorzystuje dwa osobniki z poprzedniej populacji (rodzice) do stworzenia nowego osobnika (potomka). Proces wyboru rodziców opiera się o pewną strategię, największą szansę mają osobniki o najwyższej wartości funkcji celu. W Procesie tworzenia nowego osobnika wykorzystywany jest operatora krzyżowania, który determinuje jak cechy rodziców zostaną przetworzone. Operator mutacji natomiast wprowadza do cech osobnika, nowe cechy, które nie posiada żaden z rodziców. Nowa populacja składa się więc z potomków populacji poprzedniej. Jest też możliwość skopiowania osobników o największej funkcji celu (w przypadku maksymalizacji) do nowej populacji. Jednak liczba osobników w populacji między kolejnymi iteracjami zazwyczaj pozostaje niezmieniona. [8]

Strategie wyboru rodziców:

Ruletka (ang. Roulette Wheel Selection) - koncepcja polegająca na przypisaniu osobnikowi prawdopodobieństwa wyboru na rodzica, proporcjonalnie do wartości funkcji celu. Osobniki o wyższych wartościach funkcji celu mają większą szansę na bycie wybranymi.

1. Oblicz sumę wszystkich wartości funkcji celu w populacji.
2. Każdemu osobnikowi przypisz proporcjonalny „przedział” na wirtualnym kole ruletki.
3. Zakręć kołem ruletki i wybierz osobnikia na którym zatrzyma się ruletka.

Selekcja turniejowa (Tournament Selection) - strategia polegająca na losowym wyborze k osobników do tzw. turnieju. Następnie zwycięzca turnieju (osobnik z najwyższą funkcją celu) zostaje rodzicem.

1. Z populacji wybierz losowo k osobników.
2. Spośród tych osobników wybierz osobnika z najwyższą wartością funkcji celu.
3. Powtórz proces, aby wybrać kolejnego rodzica.

Elitarny wybór (Elitism) - Osobniki o najwyższej wartości funkcji celu są automatycznie przenoszeni do następnego pokolenia, a reszta rodziców jest wybierana inną strategią.

1. Wybierz najlepsze osobniki na podstawie funkcji celu.
2. Dodaj je bezpośrednio do nowej populacji.
3. Pozostałych rodziców wybierz dowolną strategią (np. ruletką, turniejem).

Celem operator mutacji jest wprowadzenie nowych cech do osobnika. Operator jest definiowany zależnie od struktury osobnika. Np. usunięcie 5% cech osobnika i zastąpienie ich losowymi.

Operator krzyżowania odpowiada za wybór cech, które rodzice przekażą potomstwu. Przykładami operatorów krzyżowania mogą być:

Genotyp jest zestawem genów danego osobnika.

Cycle Crossover (CX) – krzyżowanie cykliczne jest wykorzystywane w problemach dla, których genotypy są permutacjami. Mechanizm ten zapewnia, że wartości genów w dzieciach będą unikalne i obecne w rodzicach. [9]

1. Zidentyfikuj cykl między rodzicami, zaczynając od pierwszego rodzica.
2. W jednym dziecku zachowaj geny jednego cyklu od pierwszego rodzica, a resztę genów od drugiego.
3. Powtórz dla drugiego dziecka w odwrotnej konfiguracji.

3. Autorski algorytm optymalizacji hiperparametrycznej BOinEA

Niniejszy rozdział zawiera omówienie autorskiego algorytmu optymalizacji hiperparametrycznej. Inspiracją do jego stworzenia był algorytm BOinG [7]. Algorytm BOinG realizował optymalizację dwustopniową. Pierwszy stopień polegał na optymalizacji funkcji celu globalnie, korzystając z modelu random forest, natomiast drugi stopień na optymalizacji lokalnej, przeprowadzanej przy użyciu procesów gaussowskich. Koncepcja dwustopniowej optymalizacji, polega na użyciu jako optymalizatora globalnego modelu skalowalnego, a w przypadku optymalizacji lokalnej modelu o większej złożoności obliczeniowej, który może trafniej eksplatować pewną podprzestrzeń rozwązań.

3.1. BOinEA

Jedną z wielu zalet algorytm genetyczny jest jego zdolność do optymalizacji globalnej. Wykorzystując takie mechanizmy jak mutacje w genotypie, krzyżowanie, strategią wyboru rodziców, możemy sterować zakresem eksploracji przestrzeni.Koncepcja autorskiego algorytmu BOinEA polega na użyciu algorytmu genetycznego podczas optymalizacji globalnej oraz procesów Gaussa podczas optymalizacji lokalnej. Procesy Gaususa bardzo dobrze radzą sobie w modelowaniu funkcji celu na poziomie lokalnym, jednak ich największą wadą jest złożoność obliczniowa $O(n^3)$. Dlatego użycie algorytmu genetycznego jako optymalizatora globalnego eliminuę tą wadę i poprawia złożoność obliczeniową.

W tym rozdziale omówię koncepcję autorskiego algorytmu optymalizacji hiperparametrycznej

3.2. Algorytm genetyczny

Poniżej opisano algorytm genetyczny przystosowany do procesu optymalizacji hiperparametrycznej (koncepcja algorytmu genetycznego została opisana w rozdziale 2.3.2.):

Pętla główna algorytmu:

Parametry algorytmu to:

- *iteration* - liczba iteracji - domyślnie 10,
- *param_grid* - przestrzeń hiperparametryczna,
- *mutation_probability* - prawdopodobieństwo mutacji - domyślnie 0,1,
- *population_size* - rozmiar populacji - domyślnie 10,
- *pb* - prawdopodobieństwo mutacji bitu - domyślnie 0,1.
- *Pstart* - populacja startowa,

1. **Inicjalizacja**: Stworzenie *Pstart* przy użyciu rozkładu jednostajnego, próbkując *param_grid*. Liczba osobników, stworzonych w *Pstart* jest równa *population_size*. Zalecana wartość: 10 (wartość użyta podczas testów). Osobniki zostały zapisane w tablicy population.
2. **Pętla**: Dla każdej generacji od i=0 do i=*iteration* - wykonaj:
   a) Stwórz pustą tablicę, przeznaczoną na nowe populacje - new_population.
   b) **Pętla**: tworząca nową populacje, od j=0 do *population_size*//2 - wykonaj:
      i. Wybierz dwóch rodziców z tablicy population, korzystając z funkcji tournament_selection.
      ii. Stwórz dwójkę dzieci, na podstawie rodziców wybranych w poprzednim kroku, przy użyciu funkcji crossover.
      iii. Wylosuj liczbę z zakresu [0,1], jeżeli jest mniejsza od *mutation_probability* dokonaj mutacji potomka pierwszego, przy użyciu funkcji mutation.
      iv. Wylosuj liczbę z zakresu [0,1], jeżeli jest mniejsza od *mutation_probability* dokonaj mutacji potomka drugiego, przy użyciu funkcji mutation.
      v. Dla każdego dziecka, oblicz ocenę - accuracy - przy użyciu funkcji objective (funkcja oceny).
      vi. Dodaj dzieci do tablicy - new population.
   c) Zaktualizuj populację, poprzez przypisanie do tablicy population wartości z tablicy new_population.
3. **Wypisz** najlepsze hiperparametry.
4. **Wypisz** accuracy dla najlepszych hiperparametrów.

Szczegółowy opis funkcji crossover, touranment_selection, mutation:

1. **crossover** - funkcja odpowiedzialna za krzyżowanie dwóch osobników tzw. rodziców. W wyniku zwraca dwa nowe osobnik tzw. dzieci.

   Funkcja jako parametry przyjmuje: dwa osobniki parent_1, parent_2 oraz zmienną length domyślnie ustawioną na 20. Podczas operacji krzyżowania funkcja operuje na dwóch typach gentów tj. geny będące wartościami rzeczywistymi lub całkowitoliczbowymi:

   a) współczynnik uczenia,

   b) liczba neuronów w danej warstwie gęstej,

   c) liczba warstw gęstych,

   d) liczba warstw konwolucyjnych,

   e) wartość dropout

   lub mającymi postać kategoryczną:

   a) liczba filtrów używanych w warstwie konwolucyjnej,

   b) rozmiar kernela warstwy konwolucyjnej.

   Krzyżowanie genów w przypadku zmiennych kategorycznych polega na wzięciu genu kategorycznego naprzemian tzn. raz od rodzica pierwszego, raz od rodzica drugiego. Natomiast krzyżowanie w przypadku zmiennych ciągłych lub całkowitych, których przedział jest określony przez granicę dolną i górną, wygląda następująco:

   (a) zamieniamy zmienną ciągłą lub całkowitą na 20-bitową postać (parametr length=20) - za pomocą funkcji encode_variable - tworzona jest tablica zawierająca 20 bitów,

   (b) następnie realizowana jest strategia one-point-crossover tzn. wybierany jest punkt krzyżowania (index w tablicy 20 bitowej), zwany point-crossover. Index wybierany jest z zakresu 1-8, aby zwiększyć zmienność w cechach, ponieważ bity początkowe są bardzej znaczące. Następnie cecha ciągła pierwszego dziecka tworzona jest jako sklejenie fragmentów tablic, pochądzących od rodziców. To znaczy, pierwsze dziecko otrzymuje od rodzica pierwszego, fragment tablicy zaczynającej się indeksem zerowym, a kończącym indeksem równym point-crossover. Drugą część zaczynającą się od point-crossover do ostatniego bitu, dostaje od drugiego rodzica. W przypadku drugiego dziecka zamieniamy kolejnością rodziców i wykonujemy identyczną operację,

(c) następnie po utworzeniu bitowej reprezentacji cechy dla dwójki dzieci, są ode dekodowane do postaci ciągłej. Najczęściej w wyniku tej operacji powstaje liczba rzeczywista, dlatego w przypadku zmiennych całkowitych, dodatkwo należy przekonwertować wynik do zmiennej całkowitej.

Wynikiem operacji krzyżowania są dwa osobniki, które następnie dodawane są do bierzącej populacji.

2. **mutation** - funkcja odpowiedzialna za mutacje w genotypie osobnika. W przypadku zmiennych kategorycznych, pozostają one zachowane bez zmian. Natomiast zmienne ciągłe lub całkowite mutowane są według poniższego schematu:

   a) zamienimay zmienną na postać binarną 30-bitową, przy użyciu funkcji encode_variable,

   b) następnie iterujemy po tablicy bitów i negujemy bity, jeżeli wylosowano liczba z zakresu [0,1] jest mniejsza niż 0,1,

   c) dekodujemy liczbę z postaci bitowej, na postać ciągłą. W przypadku zmiennych całkowitych, konwertujemy otrzymane wyniki do postaci całkowitej.

3. **tournament_selection** - funkcja realizująca strategię wyboru rodziców. W stworzonym algorytmie wybrano strategię turniejową. Opis:

   a) turniej polega na losowym wybraniu dwóch osobników z populacji. Następnie wybraniu osobnika o większej wartości funkcji celu,

   b) powyższy krok powtarzamy, z zastrzeżeniem, iż znaleziony osobnik nie jest tym samym, który został znaleziony w kroku a,

   c) funkcja zwraca dwa osobniki.

Ponadto ważnym aspektem powyższego algorytmu były funkcje pomocnicze:

1. **encode** - funkcja zamieniająca liczbę z postaci dziesiętenj, na postacią bitową o zadanej precyzji. [10] Funkcja przyjmuje parametry:

   – liczbę w postaci dziesiętnej,

   – precyzję,

   – minimalną wartość liczby - dolne ograniczenie dla hiperparametru pobrane z zdefiniowanej przestrzeni hiperparametrycznej oraz długość liczby w postaci bitowej.

Funkcja encode otrzymuje jako parametr wartość zmiennej precision. Jest ona obliczana jako, iloraz zakresu kodowanych wartości i ilości liczb, które możemy zakodować na n-bitach. Możemy ją obliczyć korzystając ze wzoru:

$$precyzja = \frac{x_{\text{max}} - x_{\text{min}}}{2^n - 1}$$

gdzie:

- x_{max} jest górną granicą hiperparametru,
- x_{min} jest dolną granicą hiperparametru,
- n - liczba bitów na których zakodowana jest zmienna.

Precyzja wprowadza informacje do algorytmu, jak bardzo mogą różnić się dwie liczby zakodowane bitowo.
Kroki algorytmu konwersji:

a) Normalizacja danych, tj. przesunięcie liczby do zera i rozciągnięcie na cały przedział bitowy:

$$x_{\text{n}} = \frac{value - x_{\text{min}}}{precyzja}$$

gdzie:
x_{n} - znormalizowana liczba,
$value$ - liczba w postaci dziesiętnej.

b) zaokrąglenie liczby x_{n} do liczby całkowitej.

c) konwersja znormalizowanej liczby w krokach a,b, do reprezentacji binarnej.

Maksymalny błąd kwantyzacji jest równy połowie precyzji.

2. **decode** - funkcja zmieniająca liczbę z postaci bitowej, na liczbę do postaci dziesiętnej. Kroki algorytmu dekodującego można otrzymać z przekształceń wzorów dla encode. Maksymalny błąd kwantyzacji jest równy połowie precyzji.

3. **funkcja celu** - jest wskaźnikiem dokładności (accuracy) dla danej sieci neuronowej i stanowi o jakości danego osobnika.

4. **first_population_generator** - w *Pstart* tworzonych jest 10 osobników, których cechy są próbkowane z przestrzeni hiperparametrycznej na podstawie rozkładu jednostajnego.

5. **create_cnn** - funkcja budująca architekturę sieci neuronowej na podstawie zadanych hiperparametrów. Wykorzystywana jest przez funkcję celu do oceny hiperparametrów.
Algorytm genetyczny o ustawieniach domyślnych, będziemy nazywać algorytmem genetycznym w wersji podstawowej.

3.3. Algorytm BOinEA - integracja algorytmu genetycznego z procesami gaussa

W celu zwiększenia eksploracji przestrzeni rozwiązań przez algorytm genetyczny, dokonano zmian hiperparametrów algorytmu genetycznego:

– *mutation_probability* nastawiono na 0.3,

– *pb* nastawiono na 0.3,

– podczas krzyżowania, skrócono przedział losowania punktu point-crossover na przedział od 1 do 5.

Ponadto do pętli głównej algorytmu dodano funkcjonalność odpowiedzialną za optymalizację lokalną procesami gaussa, **pętla główna algorytmu BOinEA**:

1. **Inicjalizacja**: Stworzenie populacji startowej poprzez wygenerowanie osobników. Liczba osobników populacji jest równa 10. Osobniki zostały zapisane w tablicy population.
2. Ustaw flagę bayesian_flag na False.
3. **Pętla**: Dla każdej generacji od i=0 do i=*iteration* - wykonaj:
   a) Stwórz pustą tablicę, przeznaczoną na nowe populacje - new_population.
   b) **Pętla**: tworząca nową populacje, od j=0 do 5 - wykonaj:
      i. Wybierz dwóch rodziców z tablicy population, korzystając z funkcji tournament_selection.
      ii. Stwórz dwójkę potomków, na podstawie rodziców wybranych w poprzednim kroku, przy użyciu funkcji crossover.
      iii. Wylosuj liczbę z zakresu [0,1], jeżeli jest mniejsza od *mutation_probability* dokonaj mutacji dziecka pierwszego, przy użyciu funkcji mutation.
      iv. Wylosuj liczbę z zakresu [0,1], jeżeli jest mniejsza od *mutation_probability* dokonaj mutacji dziecka drugiego, przy użyciu funkcji mutation.
      v. Dla każdego dziecka, oblicz ocenę - accuracy - przy użyciu funkcji objective (funkcja oceny).
      vi. Dodaj dzieci do tablicy - new_population.
   c) Zaktualizuj populację, poprzez przypisanie do tablicy population wartości z tablicy new_population.

d) Jeśli bayesian_flag równa się True, wykonaj:

i. Znajdź najlepszego osobnika w danej populacji - najwyższa ocena,

ii. Utwórz podprzestrzeń przestrzeni hiperparametrycznej w której zawarty jest najlepszy osobnik (sąsiedztwo osobnika), za pomocą funkcji define_search_space_around_individual,

iii. Znormalizuj podprzestrzeń za pomocą funkcji normalize_dimensions,

iv. Utwórz estymator GP za pomocą funkcji cook_estimator,

v. Utwórz obiekt Optimizer (zarządzającym procesami Gaussa) jako parametry przekaż estymator GP, znormalizowaną podprzestrzeń, n_initial_points = 4, a pozostałe parametry ustaw na identyczne jak domyślne w BOGP (tj. acq_func = "EI"etc.).

vi. Przekaż do obiektu Optimizer, najlepszego osobnika z danej populacji, wraz z zanegowaną oceną (scikit-optimize minimalizuje),

vii. W pętli for od n=0 do n=9: wykonaj optymalizację Bayesowską w sąsiedztwie najlepszego osobnika. (obiekt Optimizer, metody ask, tell)

viii. Jeżeli w sąsiedztwie najlepszego osobnika znaleziono osobnika o wyższej ocenie, podmień osobnika z najwyższą oceną w populacji na osobnika o wyższej ocenie.

e) bayesian_flag ustaw na True

4. **Wypisz** najlepsze hiperparametry.

5. **Wypisz** accuracy dla najlepszych hiperparametrów.

Optymalizacja Bayesowska rozpoczyna się po trzeciej iteracji algorytmu genetycznego. To znaczy kolejność wywołań jest następująca:

– I iteracja: utworzenie populacji początkowej

– II iteracja: utworzenie drugiej populacji algorytmem genetycznym

– III iteracja: utworzenie trzeciej populacji algorytmem genetycznym

– IV iteracja: wywołanie procesów gaussowskich w sąsiedźtwie najlepszego osobnika z populacji

– ...

Następnie w kolejnych iteracjach, naprzemian wywoływany jest algorytm genetyczny i optymalizacja procesami gaussa. Za taki porządek odpowiada między innymi flaga: bayesian_flag.

Szczegóły dotyczące obiektu Optimizer, przeprowadzenia optymalizacji Bayesowskiej w pętli for, a także innych ważnych funkcji np. cook_estimator, normalize_dimensions, które są z dystrybucji biblioteki scikit-optimize, zostały zawarte w rozdziale 2.2.2.

Funkcja define_search_space_around_individual służy do tworzeniu sąsiedztwa wokół punktu z przestrzeni hiperparametrycznej. Sąsiedztwo zdefiniowana w następujący sposób:

- współczynnik uczenia $\pm 10^{\frac{x_1+x_2}{2}}$, gdzie: 10^{x_1} to granica dolna, a 10^{x_2} to granica górna, współczynnika uczenia, zakładamy, że $x_1, x_2 < 0$
- liczba neuronów w warstwie gęstej to: ± 20, z zachowaniem ograniczeń
- liczba warstw gęstych ± 1, z zachowaniem ograniczeń
- liczba warstw konwolucyjnych: const.
- liczba rozmiar kernelu: niezmienna
- liczba filtrów: const.
- liczba funkcja aktywacji: const.

gdzie:
const. oznacza takaą samaą wartość, jak dla punktu wokół, którego tworzymy sąsiedztwo.

4. Zbiory danych

W tym rozdziale zawarto spefyfikację zbiorów danych, które wykorzystano w przeprowadzonych testach algorytmów optymalizacji hiperparametrycznej. Do testów użyto trzy zbiory danych. Dwa zbiory zawierające obrazy i jeden zbiór numeryczny.

4.1. CIFAR-10

CIFAR-10 to zbiór zawierający zdjęcia dziesięciu klas obiektów. Klasy obiektów to: (Opracowano na podstawie [11])

Nr	Klasa (po polsku)	Klasa (po angielsku)
1	Samolot	Airplane
2	Samochód	Automobile
3	Ptak	Bird
4	Kot	Cat
5	Jelonek/Sarna	Deer
6	Pies	Dog
7	Żaba	Frog
8	Koń	Horse
9	Statek	Ship
10	Ciężarówka	Truck

Tabela 4.1. Klasy w zbiorze CIFAR-10.

CIFAR-10 (Kanadyjski Instytut Badań Zaawansowanych, 10 klas) to podzbiór zbioru danych Tiny Images, zawierający 60 000 kolorowych obrazów o rozdzielczości 32x32 piksele. Obrazy są oznaczone jedną z 10 wzajemnie wykluczających się klas. W każdej klasie znajduje się 6000 obrazów, z czego 5000 przeznaczonych jest do treningu, a 1000 do testowania.

4.2. Fashion-MNIST

Fashion MNIST to zbiór zawierający zdjęcia dziesięciu klas obiektów. Klasy obiektów to:(Opracowano na podstawie [12])

Nr	Klasa (po polsku)	Klasa (po angielsku)
1	T-shirt/Top	T-shirt/Top
2	Spodnie	Trouser
3	Sweter	Pullover
4	Sukienka	Dress
5	Płaszcz	Coat
6	Sandały	Sandal
7	Koszula	Shirt
8	Buty sportowe	Sneaker
9	Torebka	Bag
10	Botki	Ankle Boot

Tabela 4.2. Klasy w zbiorze Fashion MNIST.

Fashion-MNIST to zbiór danych zawierający obrazy artykułów odzieżowych Zalando — składający się z zestawu treningowego, który zawiera 60 000 przykładów oraz zestawu testowego z 10 000 przykładów. Każdy przykład to obraz w skali szarości o rozdzielczości 28x28 pikseli, przypisany do jednej z 10 klas. Każdej klasie przypisano 6000 obrazków jako zbiór treningowy oraz 100 jako zbiór testowy, jest więc to zbiór dobrze zbalansowanych.

4.3. NSL-KDD

Zestaw danych KDD99 powstał w ramach konkursu KDD Cup w 1999 roku. Zawiera on dane z symulowanego środowiska sieciowego, które mogą być analizowane w celu wykrycia normalnych zachowań i ataków hakerskich. Zestaw danych NSL-KDD został stworzony na podstawie KDD-99. Jest jego ulepszoną wersją w której usunięto powtarzające się dane.W zestawie danych NSL-KDD wprowadzono równowagę w próbkach poprzez zwiększenie proporcji rzadkich i trudnych do wykrycia próbek w zestawie testowym. Został on zaprojektowany, aby rozwiązać wiele problemów związanych z oryginalnym zestawem danych KDD, takich jak nadmiarowe rekordy, które prowadziły do uprzedzeń algorytmów uczenia maszynowego.

Każdy rekord zestawu treningowego zawiera 41 cech, które opisują sesję sieciową. Dane są etykietowane. Etykieta wskazuje czy jest to normalna aktywność, czy atak. W przypadku ataku podana jest typ. [13]

Ataki są podzielone na te same cztery główne kategorie, co w NSL-KDD: [14]

- **DOS (Denial of Service)** – Ataki blokujące usługi.
- **Probe** – Skanowanie i zbieranie informacji.
- **R2L (Remote to Local)** – Nieautoryzowany dostęp do zdalnego systemu.
- **U2R (User to Root)** – Eskalacja uprawnień.

Etykiety klas ataków to nazwy specyficznych typów (np.neptune, smurf, guess_passwd). W zbiorze są 23 etykiety.

5. Testy

W tym rozdziale przeprowadzono testy dla następujących algorytmów:

- Optymalizacja Bayesowska: BOGP, SMAC
- Autorskie algorytmy optymalizacyjne: EA, BOinEA

Testy przeprowadzono na zestawach danych:

- CIFAR-10
- Fashion-MNIST
- NSL-KDD

Specyfikacja tych zbiorów danych została zawarta w rozdziale czwartym.

5.1. Środowisko testowe

Do testów użyto środowisko testowe o następujących parametrach:

- Visual Studio Code: notatnik jupyter
- python 3.12.7 zainstalowany przy użyciu narzędzia anaconda

Użyte biblioteki podczas testów to:

NumPy	**Pandas**	**Matplotlib**	**Seaborn**	**Scikit-learn**	**Keras**	**TensorFlow**	**Skopt**
1.26.4	2.2.2	3.9.2	0.13.2	1.5.1	3.7.0	2.18.0	0.10.2

Tabela 5.1. Biblioteki użyte podczas testów

Testy wykonano na procesorze: AMD Ryzen 7 7840Hs

5.2. Optymalizowana architektura CNN dla zestawu Fashion-MNIST

Każdy algorytm optymalizacji hiperparametrycznej potrzebuje szkieletu architektury sieci neurnowej dla której zoptymalizuje jej budowę. W tym rozdziale przedstawiono architektury CNN dla każdego zestawu danych. Zdefiniowane architektury wykorzystano do przeprowadzenia testów algorytmów optymalizacji hiperparametrycznej. Niniejszy rozdział zawiera także zdefiniowane przestrzenie hiperparametryczne.

5.2.1. Fashion-MNIST

Architektura sieci CNN:

1. **Wejście do sieci:** dane wejściowe przetwarzane przez sieć to obrazy 28x28 w odcieniu szarości (jeden kanał).

2. **Warstwy konwolucyjne i pooling**
   Pierwsza warstwa konwolucyjna - zawiera następujące parametry:

   - Liczba filtrów: 32,
   - Rozmiar filtra: 5×5,
   - Stride: 1,
   - Padding: same,
   - Funkcja aktywacji - hiperparametr algorytmu.

   Pierwsza warstwa MaxPooling - parametry:

   - Rozmiar okna: 2×2,
   - Stride: 2.

   Druga warstwa konwolucyjna - parametry:

   - Liczba filtrów: 64,
   - Rozmiar filtra: 5×5,
   - Stride: 1,
   - Padding: same,
   - Funkcja aktywacji - hiperparametr algorytmu.

Druga warstwa MaxPooling - parametry:

- Rozmiar okna: 2×2,
- Stride: 2.

3. **Warstwa spłaszczająca** - przekształca dane wejściowe z postaci trójwymiarowej na jednowymiarowy wektor.
4. **Pętla tworząca warstwy gęste**:
   - Liczba warstw: hiperparametr algorytmu,
   - Liczba neuronów w każdej warstwie: hiperparametr algorytmu,
   - Funkcja aktywacji: hiperparametr algorytmu.
5. **Warstwa wyjściowa** - parametry:
   - Liczba neuronów: 10,
   - Funkcja aktywacji: sigmoid .
6. **Optymalizator i funkcja straty**
   - Optymalizator: Adam ze współczynnikiem uczenia - hiperparametr algorytmu,
   - Funkcja straty: Binary crossentropy ,
   - Metryka: dokładność.

W architekturze CNN dla zestawu Fashion-MNIST dostrajane hiperparametry to:

– współczynnik uczenia,

– rodzaj funkcji aktywacji w warstwie konwolucyjnej i ukrytych,

– liczba neuronów w danej warstwie ukrytej,

– liczba warstw ukrytych.

Poniżej, w tabeli zawarto przestrzeń hiperparametryczną:

Hiperparametr	**Typ**	**Zakres / Kategorie**
Współczynnik uczenia (tempo uczenia)	Ciągły	1×10^{-6} do 1×10^{-2}
Liczba warstw w pełni połączonych	Liczba całkowita	1 do 5
Liczba neuronów w każdej warstwie	Liczba całkowita	10 do 500
Funkcja aktywacji	Kategoryczny	`'relu'`, `'sigmoid'`

Tabela 5.2. Przestrzeń hiperparametryczna

Podczas testów na zbiorze Fashion-MNIST przyjęto stała wartość epok równą 5.

5.2.2. CIFAR-10

Architektura sieci CNN:

1. **Wejście do sieci:** dane wejściowe to obrazy o rozmiarze $32 \times 32 \times 3$ (RGB z trzema kanałami kolorów: czerwonym, zielonym i niebieskim).
2. **Pętla tworząca kolejne warstwy konwolucyjne**

   Liczba warstw konwolucyjnych jest hiperparametrem algorytmu. W pętli dodawane są kolejne warstwy, które posiadają następujące parametry:

   - **Warstwa konwolucyjna:**
     - Liczba filtrów: filters + $(15 \cdot i)$, gdzie i to numer warstwy,
     - Rozmiar filtra -hiperparametr algorytmu,
     - Stride: 1,
     - Padding: same,
     - Funkcja aktywacji - hiperparametr algorytmu.
   - **BatchNormalization:** warstwa dodawana po każdej warstwie konwolucyjnej
   - **MaxPooling:** warstwa dodawana po każdej warstwie BatchNormalization
     - Rozmiar okna: $(2, 2)$,
     - Stride: 2.
3. **Dropout:** Warstwa Dropout z parametrem 0.25.
4. **Warstwa spłaszczająca:** przekształca dane wejściowe z postaci trójwymiarowej na jednowymiarowy wektor.
5. **Pętla tworząca warstwy gęste**:
   - Liczba warstw: hiperparametr algorytmu,
   - Liczba neuronów w każdej warstwie: hiperparametr algorytmu,
   - Funkcja aktywacji: hiperparametr algorytmu.
6. **Warstwa wyjściowa:**
   - Liczba neuronów: 10,
   - Funkcja aktywacji: sigmoid.

7. **Optymalizator i funkcja straty:**
   - Optymalizator: Adam,
   - współczynnika uczenia - hiperparametr algorytmu,
   - Funkcja straty: Binary crossentropy,
   - Metryka: dokładność.

W architekturze CNN, stworzonej dla zestawu CIFAR-10, dostrajane hiperparametry to:

- współczynnik uczenia,
- rodzaj funkcji aktywacji w warstwie konwolucyjnej i ukrytych,
- liczba warstw ukrytych,
- liczba neuronów w danej warstwie ukrytej,
- liczba warstw konwolucyjncych występującymi wraz z warstwami normalizującą i maxpoolingu,
- liczba filtrów w warstwie konwolucyjnej,
- rozmiar filtra w warstwie konwolucyjnej.

Poniżej, w tabeli zawarto przestrzeń hiperparametryczną:

Hiperparametr	**Typ**	**Zakres / Kategorie**
Współczynnik uczenia (tempo uczenia)	Ciągły	1×10^{-6} do 1×10^{-2}
Liczba warstw w pełni połączonych	Liczba całkowita	1 do 5
Liczba neuronów w każdej warstwie	Liczba całkowita	10 do 500
Liczba warstw konwolucyjnych	Liczba całkowita	2 do 5
Liczba filtrów w pierwszej warstwie konwolucyjnej	Kategoryczny	16, 32, 64, 128
Rozmiar filtra w warstwie konwolucyjnej	Kategoryczny	(3×3), (5×5), (7×7)
Funkcja aktywacji	Kategoryczny	`'relu','sigmoid'`

Tabela 5.3. Przestrzeń hiperparametryczna

Podczas testów na zbiorze CIFAR-10 przyjęto stałą wartość epok równą 5.

5.2.3. NSL-KDD

Architektura sieci CNN:

1. **Wejście do sieci**: dane wejściowe przetwarzane przez sieć są wektorami o wymiarze równym liczbie cech.
2. **Warstwy gęste**:
   - **Pierwsza warstwa wejściowa (Input)** zawiera wejście o wymierze równym liczbie cech w danych wejściowych tj.41.
   - **Pętla tworząca warstwy gęste**:
     - Liczba warstw: hiperparametr algorytmu,
     - Liczba neuronów w każdej warstwie: hiperparametr algorytmu,
     - Funkcja aktywacji: hiperparametr algorytmu,
     - Współczynnik Dropout: hiperparametr algorytmu.
   - **Ostatnia warstwa gęsta (wyjściowa)**:
     - Liczba neuronów: liczba klas tj. 23,
     - Funkcja aktywacji: sigmoid.
3. **Optymalizator i kompilacja modelu**:
   - Optymalizator: Adam,
   - Współczynnik uczenia: hiperparametr algorytmu,
   - Funkcja straty: binary_crossentropy,
   - Metryka: dokładność.

W architekturze CNN, strzorzonej dla zestawu NSL-KDD, dostrajane hiperparametry to:

– współczynnik uczenia,

– rodzaj funkcji aktywacji w warstwach ukrytych,

– liczba neuronów w danej warstwie ukrytej,

– liczba warstw ukrytych.

Poniżej, w tabeli zawarto przestrzeń hiperparametryczną:

Hiperparametr	**Typ**	**Zakres / Kategorie**
Współczynnik uczenia	Ciągły	1×10^{-6} do 1×10^{-2}
Liczba warstw w pełni połączonych	Liczba całkowita	2 do 4
Liczba neuronów w każdej warstwie	Liczba całkowita	16 do 80
Funkcja aktywacji	Kategoryczny	`relu`, `sigmoid`, `tanh`
Współczynnik Dropout	Ciągły	0.0 do 0.5

Tabela 5.4. Przestrzeń hiperparametryczna

Podczas testów na zbiorze NSL-KDD przyjęto stałą wartość epok równą 10 oraz zastosowano technikę early_stoping.

5.2.4. Funkcja celu

Istnieje jedna funkcja celu używana w procesie optymalizacji hiperparametrów modelu CNN dla każdego z trzech zbiorów danych. Jej postać ma na celu:

1. **Zbudowanie modelu CNN** z określonym zestawem hiperparametrów.
2. **Trenowanie modelu** wraz z obliczeniem dokładności na zbiorze walidacyjnym. W przypadku zbioru NSL-KDD wprowadzono walidację krzyżową Stratified K-Fold.
   Uwaga: Podczas trenowania modelu zastosowano ReduceLROnPlateau. Zmniejsza on współczynnik uczenia o połowę, jeżeli dokładność na zbiorze walidacyjnym nie uległa poprawi przez dwie epoki. Minimalny współczynnik uczenia to 0.00001.

Komentarz: Funkcja zwraca negatywną dokładności dla algorytmów BOGP, SMAC (ponieważ scikit-optimize minimalizuje funkcję celu). Natomiast dla algorytmów EA, BOinEA funkcja celu zwraca dokładność.

5.3. Testy

W tym podrozdziale przeprowadzono testy dwóch algorytmów z biblioteki scikit-optimize: BOGP, SMAC oraz dwóch autorskich algorótmów EA w wersji podstawowej i BOinEA. Testów dokonano na trzech zbiorach danych, których specyfikacja jest zawarta w rozdziale czwartym.

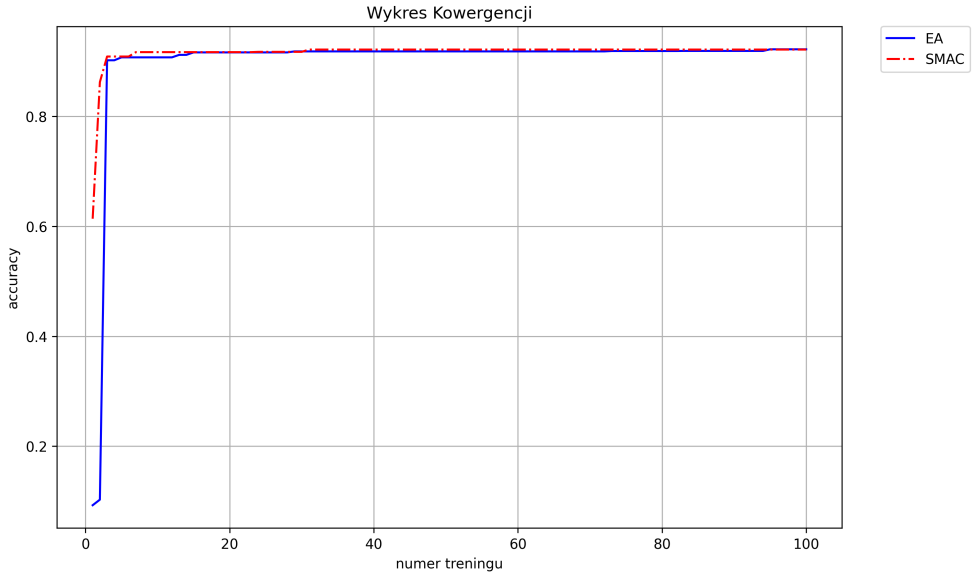
W ramach optymalizacji hiperparametrycznej nauczono 100 modeli sieci neuronowych o różnej architekturze, testując zadany algorytmu na danym zbiorze.

5.3.1. Fashion-MNIST

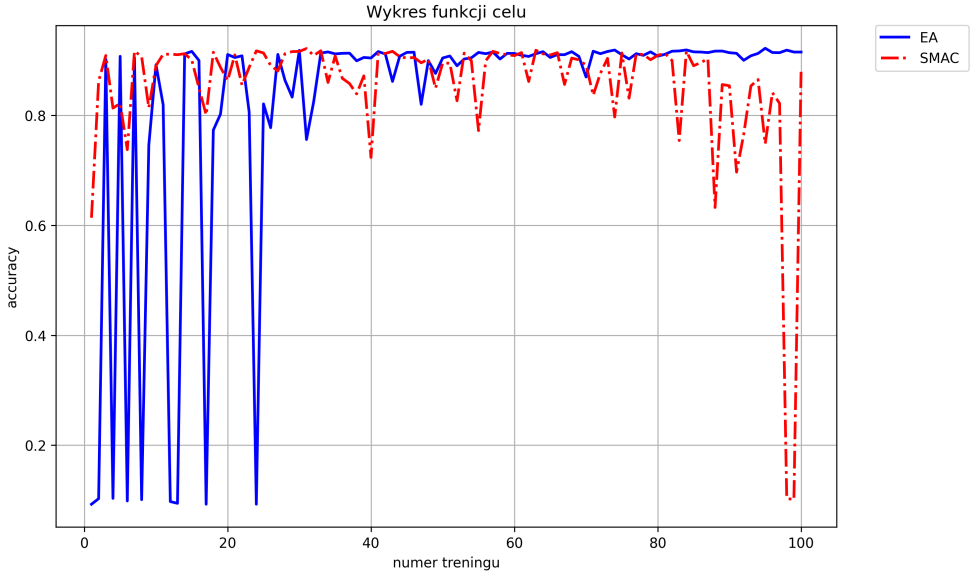
W tym podrozdziale przedstawiono wyniki procesu optymalizacji hiperparametrycznej, uzyskane przez testowane algorytmy dla zestawu Fashion-MNIST.

Algorytm	Współczynnik uczenia	Liczba warstw gęstych	Liczba neuronów w warstwie gęstej	Funkcja aktywacyjna	Accuracy zbiór treningowy [%]	Accuracy zbiór testowy [%]	Czas obliczeń (sekundy)
BOGP	0.000995	1	297	ReLU	92.37	91.83	7304
SMAC	0.000395	2	418	ReLU	92.17	91.20	6512
EA	0.001852	2	274	ReLU	92.21	91.72	6424
BOinEA	0.002700	1	330	ReLU	92.16	91.57	6099

Tabela 5.5. Wyniki optymalizacji hiperparametrycznej



Rys. 5.1. Wykres kowergencji algorytmów SMAC i EA wersja podstawowa.



Rys. 5.2. Wykres funkcji celu dla algorytmów SMAC i EA wersja podstawowa.

Wykres kowergencji dla algorytmów SMAC i EA wersja podstawowa (Rys. 5.1) jest bardzo zbliżony kształtem. Najlepsze z wytrenowanych modeli osiągnęły niemalże identyczną dokładność, różnica 0,04 % na korzyć algorytmu EA wersja podstawowa.
Wnioski z przebiegu funkcji celu dla algorytmów SMAC i EA wersja podstawowa:

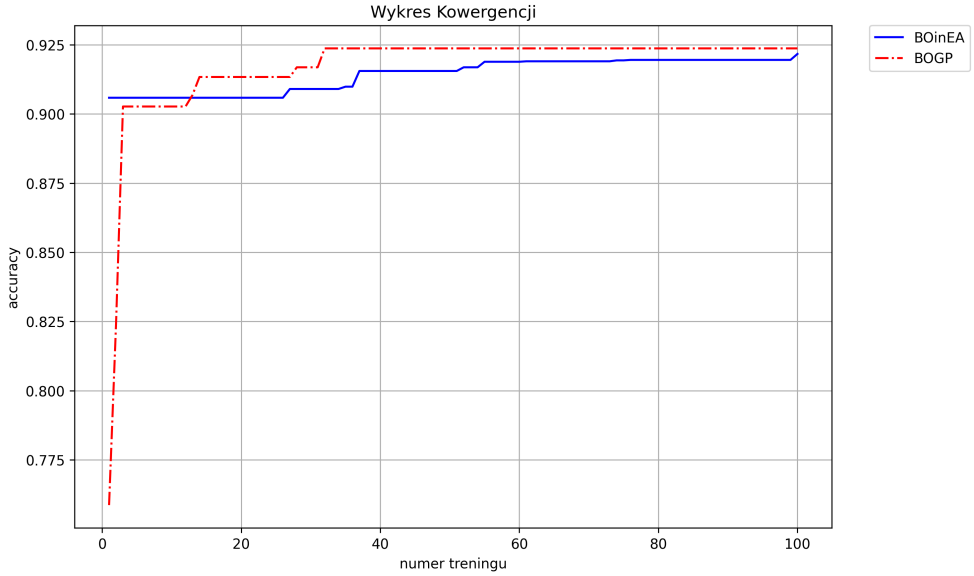
– algorytm genetyczny, przez pierwsze 25 treningów, eksploruje całą przestrzeń rozwiązań, o czym świadczy niska wartość accuracy niektórych modeli,

– po 30 treningach, cechy osobników już są tak wyselekcjonowane, że wskaźnik accuracy trenowanych sieci jest przekracza 80 % dla każdej sieci neuronowej,

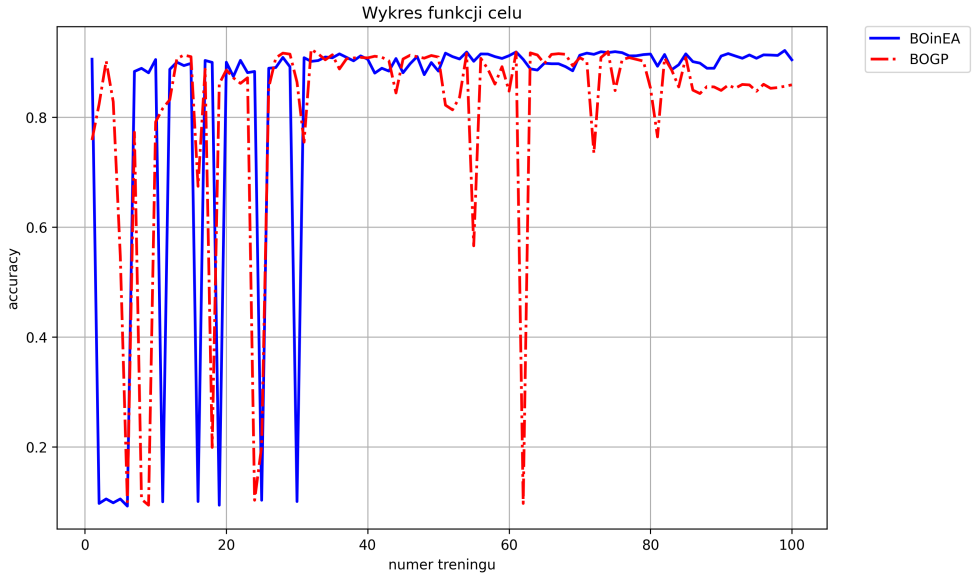
– w przypadku SMAC, algorytm do ok. 85 treningu testuje tylko modele o wskaźniku przekraczającym 70%,

– po przekroczeniu 85 treningu rozpoczyna ekploracje nowych, niezbadanych obszarów, o czym świadczy niska wartość accuracy dla trenowanych sieci,

– strategie eksploracji i eksplatacji nowych rozwiązań są różne dla obydwu algorytmów, jednak uzyskany wynik funkcji celu znalezionych jest nimalże identyczny.



Rys. 5.3. Wykres kowergencji algorytmów BOinEA i BOGP wersja podstawowa.



Rys. 5.4. Wykres funkcji celu dla algorytmów BOinEA i BOGP wersja podstawowa.

Z wykresu kowergencji dla algorytmów BOinEA oraz BOGP wynika, że algorytm BOGP znalazł minimalie lepsze rozwiązanie. Różnica wartości accuracy dla rozwiązań wynosi 0.21 %. Analizując wykres funkcji celu dla tych algorytmów nasuwają się wnioski:

– algorytm BOinEA przez pierwsze 30 treningów eksploruje przestrzeń w poszukiwaniu najlepszych rozwiązań,

– po około 30 iteracji, accuracy sieci neuronowych nie spada poniżej 85%, oznacza to, że w populacji zostały same dobre zestawy parametrów,

– BOGP podobnie jak BOinEA, przez pierwsze 30 treningów eksploruje przestrzeń rozwiązań, później wartości accuracy osiągają wartości bliskie 80-90%, jednak występują dwie niższe wartości

5.3.2. CIFAR-10

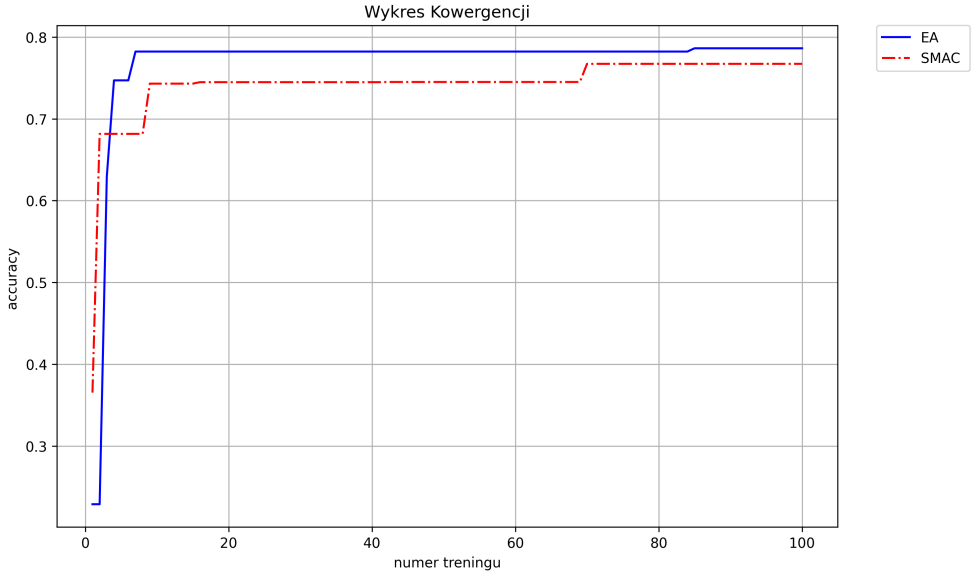
W tym podrozdziale przedstawiono wyniki procesu optymalizacji hiperparametrycznej, uzyskane przez testowane algorytmy dla zestawu CIFAR-10.

Algorytm	Współczynnik uczenia	Liczba warstw gęstych	Liczba neuronów w warstwie gęstej	Liczba warstw konwolucyjnych	Rozmiar kernela
BOGP	0.000090	1	500	2	3x3
SMAC	0.000652	5	494	2	3x3
EA	0.001380	1	343	4	3x3
BOinEA	0.001834	3	446	3	3x3

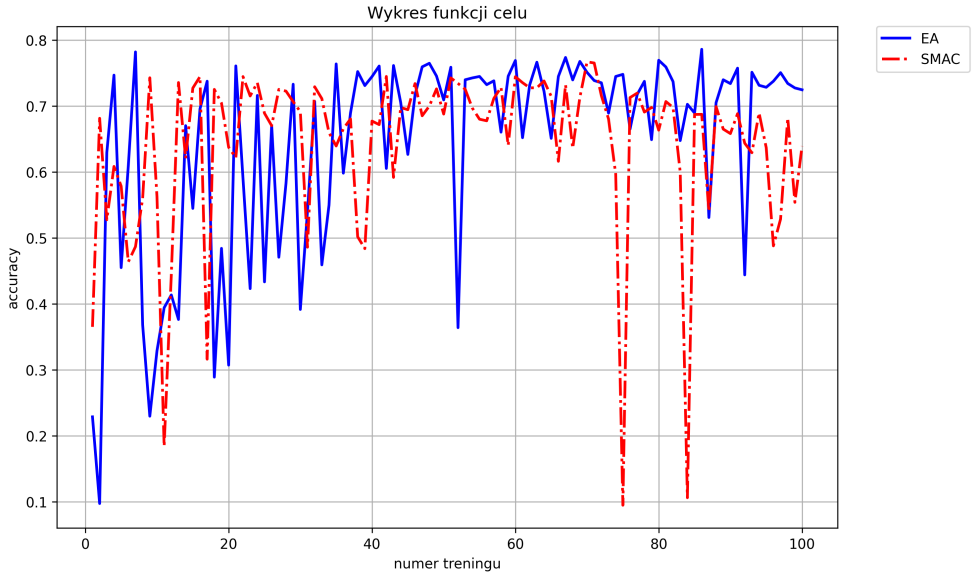
Tabela 5.6. Wyniki optymalizacji hiperparametrycznej (cz.1).

Algorytm	Liczba filtrów	Funkcja aktywacyjna	Accuracy zbiór treningowy [%]	Accuracy zbiór testowy [%]	Czas obliczeń (sekundy)
BOGP	128	ReLU	74.36	73.68	8674
SMAC	128	ReLU	76.72	74.97	31208
EA	128	ReLU	78.62	78.22	33041
BOinEA	32	ReLU	77.28	75	26262

Tabela 5.7. Wyniki optymalizacji hiperparametrycznej (cz.2).

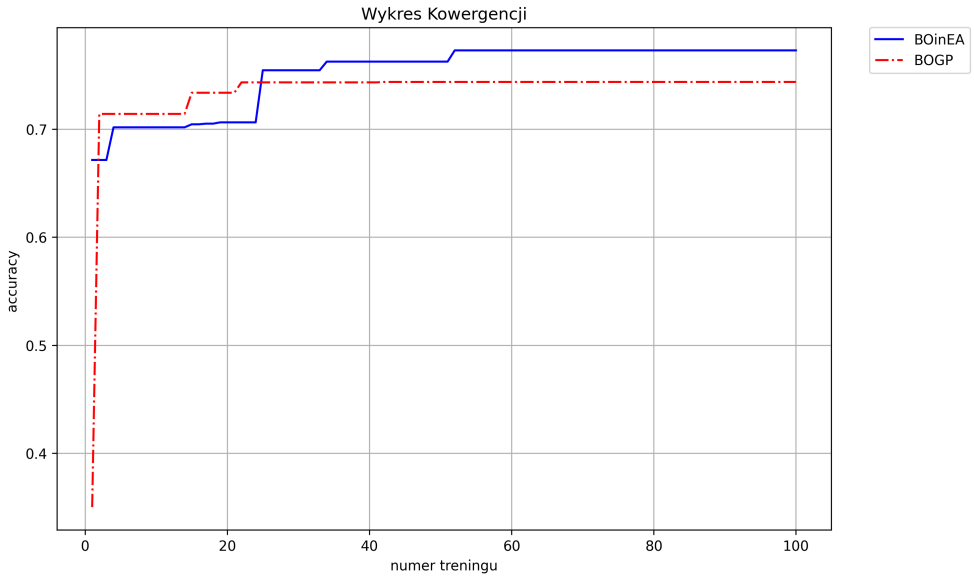


Rys. 5.5. Wykres kowergencji algorytmów SMAC i EA wersja podstawowa.

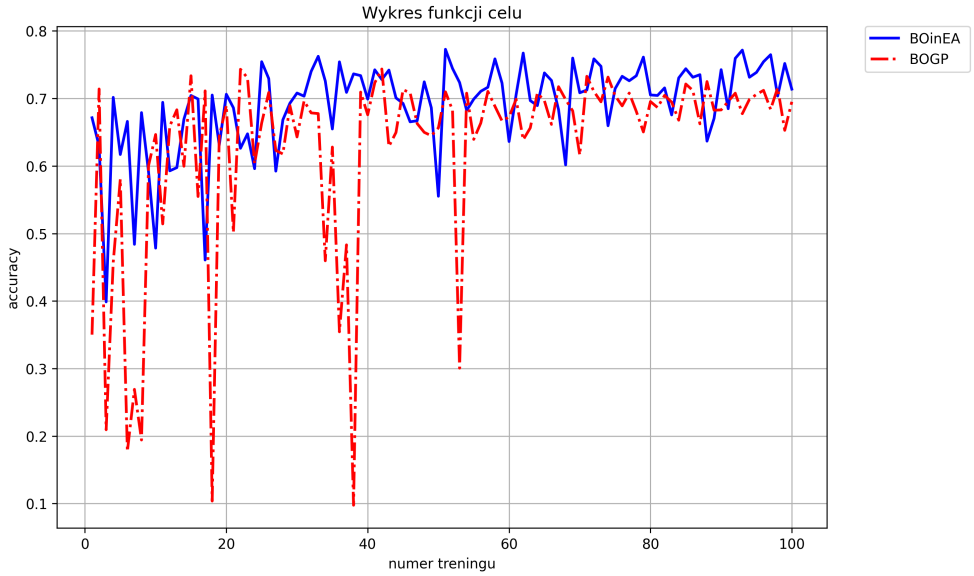


Rys. 5.6. Wykres funkcji celu dla algorytmów SMAC i EA wersja podstawowa.

Z wykresu kowergencji algorytmów SMAC i EA wersja podstawowa dla zestawu CIFAR-10, wynika, iż algorytm EA lepiej optymalizuje hiperparametry. Najlepszy znaleziony model ma wartość accuracy większą o 1.9 % w porównaniu do SMAC. Natomiast z wykresu funkcji celu, wynika, iż w algorytmie EA występuje duża zmienność wartości funkcji celu, między kolejnymi numerami treningu. W miarę wzrostu numeru treningu, wartość funkcji celu oscyluje wokół wysokich wartości accuracy.



Rys. 5.7. Wykres kowergencji algorytmów BOinEA i BOGP wersja podstawowa.



Rys. 5.8. Wykres funkcji celu dla algorytmów BOinEA i BOGP wersja podstawowa.

Na wykresie kowergencji dla algorytmów BOinEA oraz BOGP można zauważyć, iż algorytm BOinEA znalazł lepsze rozwiązanie. Różnica wyniosła, aż 2.92 punkta procentowego, mimo iż po pierwszych 23 treningach to model BOGP dawał lepsze wyniki. Analizując wykres funkcji celu nasuwają się następujące wnioski:

– rozwiązania algorytmu BoinEA oscylują wokół wysokich wartości tj. 70 %, dla algorytmu BOGP jest podobnie, jednak około dziesięciu modeli uzyskało bardzo niskie wyniki, a dla algorytmu BOinEA ani jeden.

5.3.3. NSL-KDD

W tym podrozdziale przedstawiono wyniki procesu optymalizacji hiperparametrycznej, uzyskane przez testowane algorytmy dla zestawu NSL-KDD.

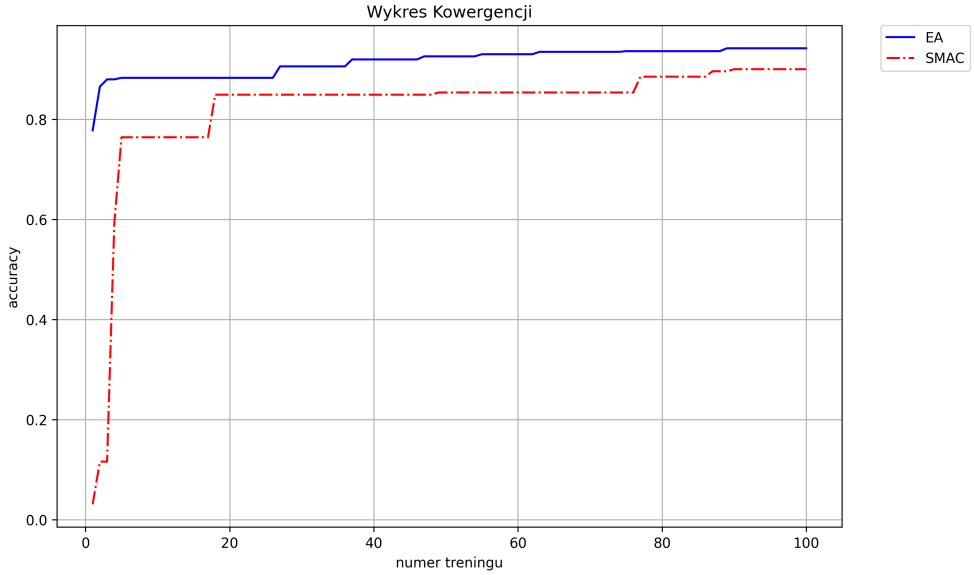
Algorytm	Współczynnik uczenia	Liczba warstw gęstych	Liczba neuronów w warstwie gęstej	Funkcja aktywacyjna
BOGP	0.001521	2	80	ReLU
SMAC	0.003499	2	74	tanh
EA	0.008572	1	491	sigmoid
BOinEA	0.007011	1	361	ReLU

Tabela 5.8. Wyniki optymalizacji hiperparametrycznej (cz.1).

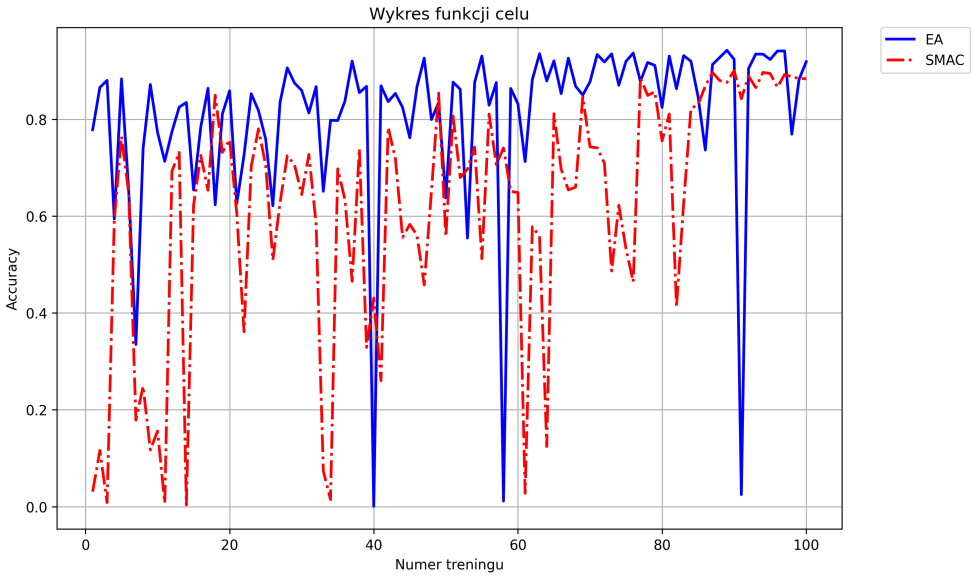
Algorytm	Dropout rate	Accuracy zbiór treningowy [%]	Accuracy zbiór testowy [%]	Czas obliczeń (sekundy)
BOGP	0	93.00	69.92	8595
SMAC	0	90.05	66.15	8363
EA	0.0584	94.22	70.17	7168.78
BOinEA	0.0557	92.83	69.56	6546

Tabela 5.9. Wyniki optymalizacji hiperparametrycznej (cz.2).

Różnica accuracy między zbiorem testowym, a treningowym, wynika z tego, iż zbiór testowy NSL-KDD został wzbogacony o obiekty, których klasy nie występują w zbiorze treningowym. Jeżeli zbiór treningowy zostanie podzielony na zbiór treningowy_1, stanowiący 80% danych treningowych i zbiór testowy_2, stanowiący 20% danych treningowego, to po przeprowadzeniu uczenia na zbiorze treningowym_1, model uzyskuje wartość accuracy na zbiorze testowym_2 przekraczającą 90%.

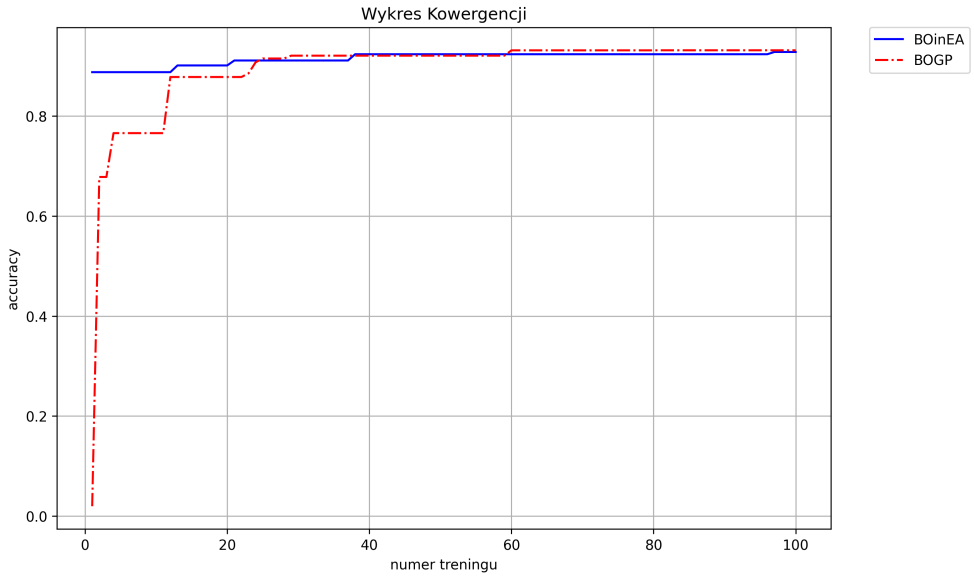


Rys. 5.9. Wykres kowergencji algorytmów SMAC i EA wersja podstawowa.

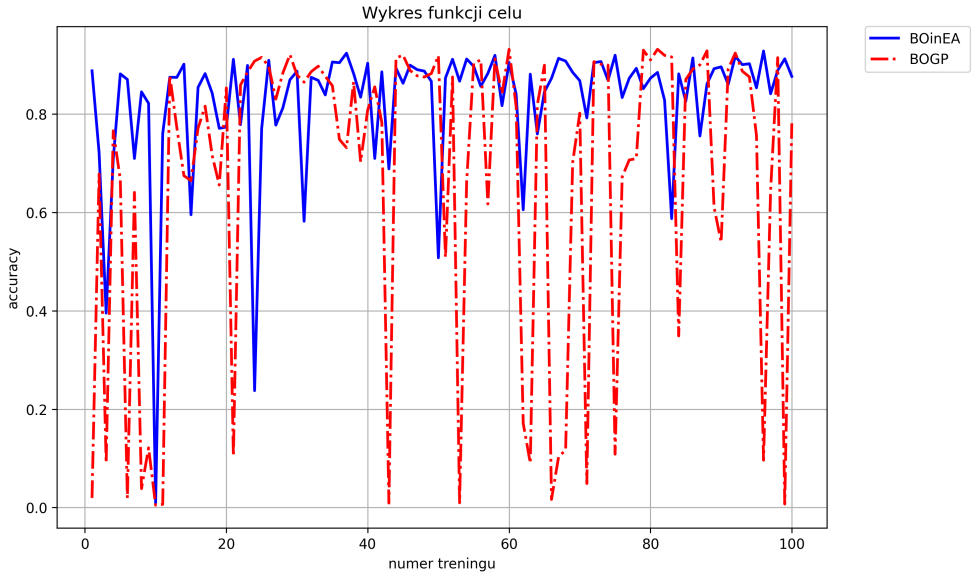


Rys. 5.10. Wykres funkcji celu dla algorytmów SMAC i EA wersja podstawowa.

Z wykresu kowergencji algorytmów SMAC i EA wynika, iż lepsze rozwiązanie znalazł algorytm EA w wersji podstawowej, różnica wynosi aż 2.78 punkta procentowego. Z wykresu funkcji celu, wynika natomiast, iż algorytm genetyczny częściej próbkował hiperparametry o wysokim wyniku funkcji celu, niż SMAC. Uzyskane niskie wyniki funkcji celu dla niektórych hiperparametrów, znalezionych przez EA, mogą świadczyć o zachodzących mutacjach.



Rys. 5.11. Wykres kowergencji algorytmów BOinEA i BOGP wersja podstawowa.



Rys. 5.12. Wykres funkcji celu dla algorytmów BOinEA i BOGP wersja podstawowa.

Z wykresu kowergencji algorytmu BOinEA oraz BOGP można odczytać, iż najlepsze rozwiązania uzyskane przez oba algorytmy, dają niemalże identyczną wartość funkcji celu. Natomiast patrząc na wykres funkcji celu można stwierdzić, że algorytm BOGP często próbkuje obszary o niskiej wartości funkcji celu, podczas gdy rozwiązania algorytm BOinEA zazwyczaj uzyskują wysokie wartości.

5.3.4. Podsumowanie testów

Z przeprowadzonych testów wynika, iż najlepszym algorytmem optymalizacji hiperparamertrycznej jest EA w wersji podstawowej. Dla zbioru NSL-KDD znalazł hiperparametry, których wartość funkcja celu jest wyższa o 1.22 punka procentowego, w porównaniu do wyniku uzyskanego przez sieć neuronową, dla której hiperparametry znalazła inna metoda. W zbiorze CIFAR-10 również zwyciężył różnicą 1.34 punkta procentowego nad BOinEA. Natomiast dla zestawu Fashion-MNIST dał wynik gorszy o 0.16 punkta procentowego, niż algorytm BOGP. Wynik EA w wersji podstawowej zajął drugie miejsce podczas testów na zestawie Fashion-MNIST. Po przeprowadzonych testach, można stwierdzić, iż algorytm EA w wersji podstawowej wypadł w nich nalepiej. Drugie miejsce zajął algorytm BOinEA przed BOGP, gdyż algorytmy dały porównywalne wyniki w NSL-KDD oraz Fashion-MNIST (minimalna różnica na korzyść BOGP), a w zestawie CIFAR-10, BOinEA uzyskał wynik lepszy o 2.92 punkta procentowego. Na ostatnim miejscu uplasował się SMAC, zajmując kolejno trzecie, trzecie i czwarte miejsce podczas realizowanych testów na zbiorach danych.

Bibliografia

[1] Owen Louis. „*Hyperparameter Tuning with Python: Boost your machine learning model's performance via hyperparameter tuning*". Birmingham: Wydawnictwo Packt Publishing, 2022.

[2] Aston Zhang et al. „*DIVE INTO DEEP LEARNING*". Cambridge: Wydawnictwo Cambridge University Press, 2023.

[3] Marc Peter, Deisenroth i A. Aldo Faisal Cheng Soon Ong. „*Matematyka w uczeniu maszynowym*". Gliwice: Helion, 2022.

[4] „*scikit-optimize*". URL: *https://github.com/scikit-optimize/scikit-optimize/blob/master/skopt/optimizer/gp.py* (term. wiz. 2024-12-05).

[5] „*API: gp_minimize*". URL: *https://scikit-optimize.github.io/stable/modules/generated/skopt.gp_minimize.html#skopt.gp_minimize* (term. wiz. 2024-12-05).

[6] Aurélien Géron. „*Uczenie maszynowe z użyciem Scikit-Learn, Keras i TensorFlow. Wydanie III*". Gliwice: Helion, 2023.

[7] Difan Deng i Marius Lindauer. „*Searching in the Forest for Local Bayesian Optimization*". 2021. arXiv: *2111.05834* [cs.LG].

[8] Tomasz Dominik Gwiazda. „*Algorytmy genetyczne : kompendium. T. 1, Operator krzyżowania dla problemów numerycznych*". Warszawa: Wydawnictwo Naukowe PWN, 2007.

[9] „*operatory krzyżowania w algorytmie genetycznym*". URL: *https://medium.com/geekculture/crossover-operators-in-ga-cffa77cdd0c8* (term. wiz. 2024-12-28).

[10] Marin Golub. „*AN IMPLEMENTATION OF BINARY AND FLOATING POINT CHROMOSOME REPRESENTATION IN GENETIC ALGORITHM*".

[11] „*Specyfikacja zbioru danych CIFAR-10*". URL: *https://paperswithcode.com/dataset/cifar-10* (term. wiz. 2024-12-19).

[12] „*Specyfikacja zbioru danych Fashion-MNIST*". URL: *https://github.com/zalandoresearch/fashion-mnist* (term. wiz. 2024-12-19).

[13] „*Specyfikacja zbioru danych NSL-KDD*". URL: *https://www.kaggle.com/datasets/hassan06/nslkdd?resource=download&select=KDDTest-21.txt* (term. wiz. 2024-12-19).

[14] „*Specyfikacja zestawu danych KDD*". URL: *https://kdd.ics.uci.edu/databases/kddcup99/task.html* (term. wiz. 2024-12-28).