

**Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej**

Praca dyplomowa

*Zastosowanie uczenia ze wzmocnieniem w celu zastąpienia regulatora instalacji grzewczych*

*Reinforcement Learning Applications to Control of Heating Systems*

Autor: inż. *Adrian Gacek*

Kierunek studiów: Automatyka i Robotyka

Opiekun pracy: *dr inż. Jakub Grela*

Kraków, 2024

Spis treści

[Wstęp 4](#_Toc171609334)

[1. Podstawy teoretyczne uczenia ze wzmocnieniem 5](#_Toc171609335)

[1.1. Uczenie maszynowe 5](#_Toc171609336)

[1.2. Podstawowe pojęcia w dziedzinie uczenia ze wzmocnieniem 6](#_Toc171609337)

[1.3. Proces decyzyjny Markova 7](#_Toc171609338)

[1.4. Funkcja wartości 8](#_Toc171609339)

[1.5. Eksploracja, a eksploatacja 9](#_Toc171609340)

[1.6. Klasyfikacja metod uczenia ze wzmocnieniem 9](#_Toc171609341)

[1.7. Uczenie offline 10](#_Toc171609342)

[1.8. Sieci neuronowe 10](#_Toc171609343)

[1.9. Sieci rekurencyjne 12](#_Toc171609344)

[2. Algorytmy uczenia ze wzmocnieniem 14](#_Toc171609345)

[2.1. Q-learning 14](#_Toc171609346)

[2.2. Twin Delayed Deep Deterministic Policy Gradient 15](#_Toc171609347)

[2.3. Soft Actor-Critic 16](#_Toc171609348)

[2.4. Conservative Q-Learning 16](#_Toc171609349)

[2.5. Inne algorytmy 17](#_Toc171609350)

[3. Regulacja instalacji grzewczych 19](#_Toc171609351)

[3.1. Dotychczasowe rozwiązania – metody tradycyjne 19](#_Toc171609352)

[3.2. Dotychczasowe rozwiązania – uczenie maszynowe 20](#_Toc171609353)

[4. Narzędzia implementacji 22](#_Toc171609354)

[4.1. Python 22](#_Toc171609355)

[4.2. Tensorflow 22](#_Toc171609356)

[4.3. Tf-Agents 22](#_Toc171609357)

[4.4. Optuna 23](#_Toc171609358)

[4.5. TCLab 23](#_Toc171609359)

[5. Implementacja środowiska 24](#_Toc171609360)

[5.1. Środowisko symulacyjne 24](#_Toc171609361)

[5.2. Analiza danych procesu sterowania 25](#_Toc171609362)

[5.3. Interfejs środowiska 28](#_Toc171609363)

[5.4. Akcje i nagrody 30](#_Toc171609364)

[6. Implementacja agentów 32](#_Toc171609365)

[6.1. DQN 32](#_Toc171609366)

[6.2. Algorytmy aktor-krytyk 33](#_Toc171609367)

[6.3. Optymalizacja hiperparametrów 33](#_Toc171609368)

[Bibliografia 34](#_Toc171609369)

# Wstęp

W ciągu ostatnich lat, można zaobserwować znaczące przyśpieszenie rozwoju sztucznej inteligencji, a jej potencjał znajduje zastosowanie w coraz szerszym zakresie problemów. Od generowania i identyfikacji obrazów, przez chatboty, po systemy sterowania. Algorytmy uczenia maszynowego coraz częściej odgrywają kluczową rolę w rozwiązywaniu zarówno prostych, jak i niezwykle złożonych wyzwań. Jednym z obszarów, w którym sztuczna inteligencja staje się coraz bardziej znacząca, jest skuteczne i zoptymalizowane sterowanie systemami grzewczymi.

Współczesne systemy grzewcze odgrywają istotną rolę w zapewnieniu komfortu i bezpieczeństwa w budynkach, a ich odpowiednia regulacja jest kluczowa dla efektywnego wykorzystania energii. Ze względu na powszechność tego problemu oraz duże znaczenie ogrzewania w kontekście kosztów eksploatacji budynku, nawet niewielkie poprawy w sterowaniu mogą mieć istotny wpływ zarówno na budżet, jak i na środowisko.

W świetle powyższego, niniejsza praca ma na celu przeprowadzenie analizy możliwości wykorzystania algorytmów uczenia ze wzmocnieniem w problemach sterowania systemami grzewczymi. Użycie tych zaawansowanych technik może prowadzić do bardziej efektywnego wykorzystania energii, poprawy komfortu użytkowników oraz redukcji negatywnego wpływu na środowisko naturalne.

Kolejne rozdziały będą miały na celu przedstawienie teoretycznych podstaw zagadnienia uczenia maszynowego, w szczególności przegląd metod uczenia ze wzmocnieniem i istotnych pojęć z nimi związanych oraz zaprezentowane zostaną przykłady zastosowania podobnych rozwiązań w literaturze. Następnie w części praktycznej przedstawiony zostanie proces przygotowania środowiska opartego o symulację systemu grzewczego i uczenia wybranych algorytmów jego sterowania. Na koniec opisano wyniki testów symulacyjnych oraz sprzętowych nauczonych modeli, wraz z wnioskami.

# Podstawy teoretyczne uczenia ze wzmocnieniem

Niniejszy rozdział ma za zadanie przedstawić podstawowe idee stojące za uczeniem maszynowym i metodami uczenia ze wzmocnieniem.

## Uczenie maszynowe

Uczenie maszynowe to obszerna dziedzina sztucznej inteligencji, zajmująca się rozwijaniem algorytmów i technik umożliwiających komputerom uczenie się z danych i doświadczenia, bez konieczności jawnego programowania. Polega ono na tworzeniu modeli, które analizując dane, wyszukują wzorców i podejmują decyzje lub dokonują prognoz na ich podstawie.

Uczenie maszynowe można podzielić na następujące obszary:

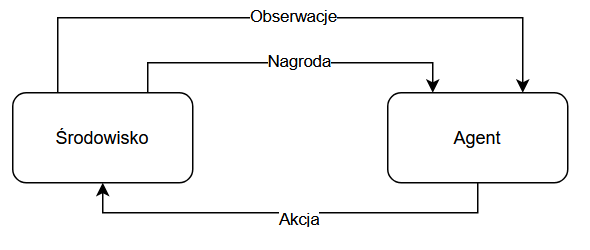
* Uczenie nadzorowane
* Uczenie nienadzorowane
* Uczenie ze wzmocnieniem

Algorytmy uczenia nadzorowanego otrzymują zestaw danych wejściowych, w którym każdy przykład posiada etykietę określającą pożądane wyjście. Celem jest znalezienie funkcji mapującej dane wejściowe na dane wyjściowe. Są one stosowane na przykład w problemach klasyfikacji i regresji.

Uczenie nienadzorowane zakłada, że algorytm otrzymuje dane wejściowe bez etykiet i ma za zadanie znajdować wzorce występujące w danych. Taki rodzaj algorytmów może być zastosowany do klasteryzacji czy wykrywania anomalii.

Uczenie ze wzmocnieniem (w skrócie RL, z ang. Reinforcement Learning) to proces uczenia się, w którym celem jest maksymalizacja nagrody poprzez przyporządkowanie sytuacji do akcji. Uczący się algorytm nie otrzymuje wskazówek, które akcje wykonać, lecz nabiera doświadczenia poprzez przeprowadzenie interakcji z systemem. Tworzy się w ten sposób model wskazujący, które akcje przynoszą największą nagrodę, biorąc pod uwagę nie tylko stan aktualny, ale również następujące po nim stany przyszłe. Poszukiwanie metodą prób i błędów oraz opóźniona nagroda to dwie główne cechy odróżniające uczenie ze wzmocnieniem od innych form uczenia. Problematyka uczenia ze wzmocnieniem jest formalizowana przy użyciu teorii systemów dynamicznych. [1]

Proces uczenia ze wzmocnieniem opiera się o interakcję między agentem a środowiskiem: agent wykonuje akcję, na którą reaguje środowisko, zmieniając swój stan i przyznając za to nagrodę. Następnie agent otrzymuje nagrodę oraz obserwacje zawierające informacje o stanie środowiska, które wykorzystuje do dalszej nauki i optymalizacji swoich przyszłych decyzji. Rys. 1. przedstawia ogólny schemat przepływu danych w takim procesie. Jego celem jest opracowanie strategii, która maksymalizuje sumę przyszłych nagród, które agent może otrzymać.



Rys. 1. Interakcja między środowiskiem i agentem. Opracowanie własne przy pomocy narzędzi draw.io

## Podstawowe pojęcia w dziedzinie uczenia ze wzmocnieniem

Dla pełnego zrozumienia procesu uczenia ze wzmocnieniem należy zdefiniować jego podstawowe elementy [1]:

**Obserwacje** – są to wszystkie informacje o stanie środowiska, które umożliwiają określenie następnej akcji podjętej przez agenta. Warto zauważyć, że nie w każdym systemie agent ma dostęp do wszystkich zmiennych stanu, co może utrudnić proces uczenia. W wielu źródłach można jednak spotkać stosowanie pojęcia obserwacji i stanu zamiennie.

**Akcja** – Reprezentuje działanie, jakie wykonuje agent, aby wpłynąć na zmianę stanu środowiska, np. wartość sygnału sterującego w systemach sterowania. Może być zdefiniowana w sposób zarówno ciągły, jak i dyskretny.

**Nagroda** - zmienna zwracana przez środowisko po wykonaniu kroku przez agenta, obrazuje to, jak korzystne było wykonanie akcji. Celem agenta jest maksymalizowanie sumarycznej nagrody gromadzonej podczas całego cyklu życia procesu.

**Trajektoria –** ciąg następujących po sobie trójek stan-akcja-nagroda reprezentujący interakcję agenta ze środowiskiem: () gdzie – stan w czasie t,   
 – akcja, – nagroda.

**Bufor doświadczeń** - Wiele algorytmów optymalizacyjnych zakłada, że próbki są niezależne i identycznie rozłożone. Pobieranie doświadczeń w sposób sekwencyjny, powoduje jednak, że sąsiednie stany są ze sobą ściśle skorelowane. Dodatkowo, aby efektywnie wykorzystać optymalizację sprzętową, istotne jest stosowanie batchy, a więc zestawów doświadczeń przetwarzanych w jednej iteracji algorytmu. W celu rozwiązania tego problemu można zastosować bufor powtórek. Jest to skończony bufor o ograniczonym rozmiarze przechowujący krotki (). Pobieranie zestawów np. w sposób losowy zapewnia niezależność próbek wymaganą między innymi w optymalizacji sieci neuronowych. [24]

**Polityka –** funkcja mapująca obserwacje ze środowiska na akcje. Ma za zadanie zdefiniować zachowanie agenta w sytuacjach, w jakich może znaleźć się środowisko. Znalezienie odpowiedniej polityki jest celem uczenia ze wzmocnieniem. Może być definiowana zarówno deterministycznie, jak i stochastycznie.

**Stopa dyskontowa** – współczynnik określający relację między natychmiastową nagrodą, a przewidywaną sumą przyszłych nagród. Wzór (1) wskazuje typowe zastosowanie:

|  |  |  |
| --- | --- | --- |
|  |  | (1) |

Gdzie:

* – maksymalizowana w procesie uczenia wartość sumy nagród oceniająca korzyść z wykonania danej akcji, nazywana również zwrotem
* numer stanu, dla którego wyznaczamy wartość sumy nagród
* – stopa dyskontowa, wartość pomiędzy zapewnia, że każdy następny przewidywany stan, ma mniejsze znaczenie względem natychmiastowej nagrody
* – numer przyszłego przewidywanego stanu
* – nagroda

## Proces decyzyjny Markova

U podstaw paradygmatu uczenia ze wzmocnieniem leży proces decyzyjny Markova (Markov Decision Process, MDP), z pomocą którego można formalnie zdefiniować wiele badanych w tej dziedzinie problemów. Mając na uwadze elementy wymienione w poprzednim podrozdziale warto omówić cechy takiego procesu.

Proces decyzyjny Markova jest zdefiniowany jako piątka (𝑆, 𝐴, 𝒯, 𝑟, 𝛾) [16].

Gdzie:

* 𝑆 – przestrzeń stanów
* 𝐴 – przestrzeń akcji.
* 𝒯 – dynamika przejść - model systemu, który opisuje zasady rządzące przejściami z jednego stanu do drugiego. W stanie 𝑠𝑡 podejmowana jest akcja 𝑎𝑡,   
  która dokonuje przejścia do nowego stanu systemu 𝑠𝑡+1∈𝑆, zgodnie z dynamiką przejść   
  środowiska 𝒯(𝑠𝑡+1|𝑠, 𝑎).
* 𝑟 – funkcja nagrody 𝑟: 𝑆×𝐴⟶ℝ
* 𝛾 – stopa dyskontowa

Bardzo istotną cechą takiego procesu jest to, że spełnia on własność Markova. Pomimo, że nagrody, jak i przejścia mogą być określone przez prawdopodobieństwo, zależą one wyłącznie od aktualnego stanu i akcji. Poprzednie wartości nie mają wpływu na przyszłość. Warto jednak nadmienić, że jest to cecha modelu opisującego proces. Czy dany proces będzie spełniał taki warunek zależy od sposobu w jaki została zdefiniowana przestrzeń stanu. [18]

Celem agenta jest maksymalizacja oczekiwanej zdyskontowanej nagrody poprzez wybieranie odpowiednich akcji. Agent powinien nauczyć się optymalnej polityki jak wskazano we wzorze (2) definiującym optymalną politykę [16] .

|  |  |  |
| --- | --- | --- |
|  |  | (2) |

Gdzie *Ea~π*  oznacza oczekiwaną wartość zwrotu, w wyniku wyboru akcji 𝑎, na podstawie polityki *π* .

## Funkcja wartości

Funkcja wartości określa, co jest korzystne w dłuższej perspektywie czasowej, w przeciwieństwie do sygnału nagrody, który ocenia tylko natychmiastowe efekty akcji. W przybliżeniu wartość stanu to całkowita suma nagród, jakiej agent może oczekiwać, zaczynając od danego stanu. Ma na celu wskazać długoterminową atrakcyjność stanów, biorąc pod uwagę stany oraz nagrody, które prawdopodobnie nastąpią. Wzór (3), przedstawia jak funkcja wartości określa oczekiwany zwrot przy zastosowaniu polityki π. [1]

|  |  |  |
| --- | --- | --- |
|  |  | (3) |

Podobnie określa się również funkcję wartości akcji [wzór (4)], czyli funkcja wartości przy założeniu, że w aktualnym stanie zostanie wykonana akcja : [1]

|  |  |  |
| --- | --- | --- |
|  |  | ((4) |

## Eksploracja, a eksploatacja

Istotnym problemem w dziedzinie RL jest problem balansu między eksploracją i eksploatacją. Wynika on z faktu, że zbieranie informacji i ich wykorzystywanie to często dwa wzajemnie wykluczające się działania. Z jednej strony, istotnym jest, aby agent eksplorował, czyli zdobywał informacje o swoim otoczeniu w celu poprawy przyszłych wyników. Ponosi jednak w ten sposób koszty zdobywania nowych informacji nie korzystając z posiadanej wiedzy. Z drugiej strony, eksploatując, czyli używając wyłącznie istniejącej wiedzy, ograniczamy zdolności adaptacji do istotnych zmian w otoczeniu i system może utknąć w suboptymalnej stabilnej równowadze. Dlatego optymalne zachowanie zwykle wymaga pewnej równowagi między działaniami eksploracyjnymi a eksploatacyjnymi. [2]

Przykładem rozwiązania tego problemu może być jedna z prostych metod balansowania między eksploracją a eksploatacją ε-greedy. Działa ona w następujący sposób [3]:

**Eksploatacja**: Większość czasu metoda ε-greedy działa zachłannie (greedy), czyli wybiera działanie, które w danej chwili wydaje się najlepsze na podstawie dostępnej wiedzy.

**Eksploracja**: Co pewien czas, z małym prawdopodobieństwem ε, metoda wybiera losowo jedno z dostępnych działań, zamiast wybierać działanie uważane za najlepsze.

## Klasyfikacja metod uczenia ze wzmocnieniem

Jednym z popularnych sposobów podziału metod uczenia ze wzmocnieniem jest ten, oparty o wiedzę agenta na temat modelu środowiska.

**Pierwszą** kategorią są metody **model-based**, czyli oparte na modelu środowiska. Model ten umożliwia przewidywanie przyszłych stanów i nagród na podstawie aktualnego stanu i podjętych akcji, dzięki symulacji zachowania środowiska. W ten sposób, można planować działania, rozważając możliwe przyszłe sytuacje przed ich faktycznym doświadczeniem. **Metody model-free**, są pozbawione modelu środowiska i uczą się na zasadzie prób i błędów. Są one prostsze i nie wykorzystują planowania, zamiast tego uczą się bezpośrednio na podstawie interakcji ze środowiskiem. Nowoczesne systemy RL mogą łączyć oba podejścia. [1]

Agenci w procesie treningu mogą być uczeni na dwa sposoby. W przypadku metod   
**on-policy**, algorytm posiada jedną politykę, która jest jednocześnie optymalizowana, oraz wykorzystywana do interakcji z otoczeniem. Metody **off-policy** posiadają osobną politykę do generowania danych ze środowiska i osobną tzw. docelową (target policy), która jest uczona. Przykładem mogą być metody, w których próbki generowane przez tzw. behavioral policy są przechowywane w buforze doświadczeń. W trakcie treningu próbki są pobierane z bufora np. w sposób losowy, aby na ich podstawie dokonać aktualizacji polityki docelowej. [22]

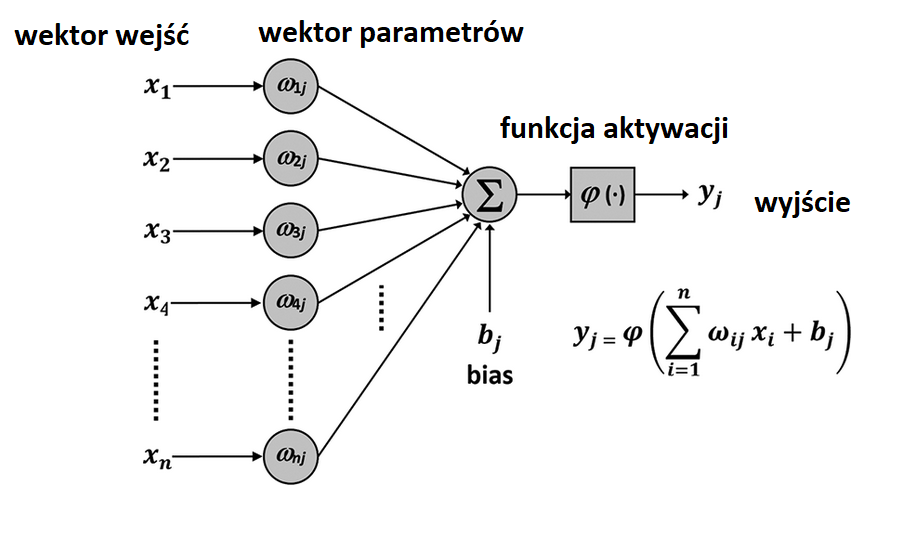
W literaturze można również znaleźć przykłady połączenia tych dwóch metod. Grupa takich rozwiązań nazywana jest aktor-krytyk (actor-critic), ponieważ w procesie treningu uczona jest zarówno polityka, jak i funkcja wartości, która jest używana do jej oceny. [22]

## Uczenie offline

Uczenie ze wzmocnieniem offline to podejście, w którym agent uczy się optymalnej polityki działania na podstawie zbioru danych zgromadzonych wcześniej, bez interakcji z rzeczywistym środowiskiem podczas fazy uczenia. To różni się od tradycyjnego (online) RL, gdzie agent uczy się przez iteracyjne działanie i obserwowanie wyników swoich działań w środowisku w czasie rzeczywistym. Rozwiązania offline mają szereg zalet takich jak bezpieczeństwo i koszt uczenia, ponieważ nie występuje konieczność odłączania działającej polityki od systemu, dzięki czemu nie narażamy systemu na wykonywanie nieoptymalnych działań w trakcie procesu uczenia, co może być w krytycznych aplikacjach (np. w medycynie) niebezpieczne, a eksploracja może być kosztowna. Ponadto jeden zestaw danych może być wielokrotnie wykorzystany do nauczenia różnych modeli i doskonalenia algorytmów. Z drugiej strony problemem jest duża zależność skuteczności agenta od danych uczących. Dane muszą być wysokiej jakości i dobrze reprezentować przestrzeń stanów i akcji, ponieważ modele mogą mieć problem z generalizacją danych. Algorytmy mogą mieć tendencję do przeszacowywania wartości akcji, co prowadzi do podejmowania nieoptymalnych decyzji.

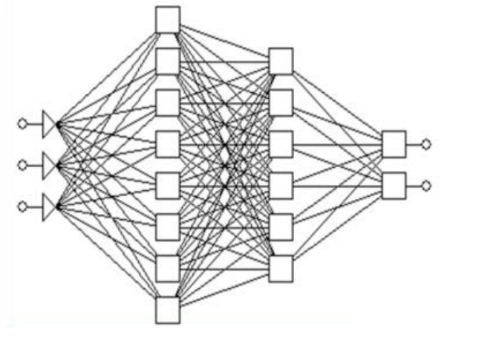
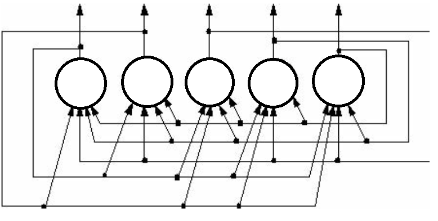
## Sieci neuronowe

Sieci neuronowe są systemem inspirowanym działaniem ludzkiego mózgu. Są one zbudowane z połączonych sztucznych neuronów, które przetwarzają dane wejściowe, przekazując je przez serię warstw, aby wygenerować pożądane wyjście. Podstawowymi jednostkami sieci neuronowych są sztuczne neurony. Rys. 2. przedstawia najprostszą sieć składającą się z jednego neuronu, tzw. perceptron. Każdy neuron przyjmuje wiele wejść, przetwarza je za pomocą funkcji aktywacji i generuje sygnał, który jest przekazywany do innych neuronów lub do wyjścia. Każde połączenie z wejściem ma przypisaną wagę, która modyfikuje wpływ sygnału. Wagi są parametrami, które sieć neuronowa uczy się dostosowywać podczas procesu uczenia. Funkcje aktywacji (np. softmax, ReLU, sigmoida) decydują, czy i w jakim stopniu neuron zostanie aktywowany. [31]



Rys. 2. Schemat perceptronu. [30]

W większości problemów sieci składają się z wielu neuronów, połączonych w warstwy. W takich przypadkach w procesie optymalizacji parametrów stosowany jest algorytm propagacji wstecznej, aby umożliwić dostosowywanie wag w warstwach oddalonych od wyjścia, dla którego wyznaczany jest błąd względem oczekiwanych rezultatów. Propagacja wsteczna (backpropagation) działa na zasadzie przenoszenia błędu od warstwy wyjściowej wstecz do warstw ukrytych, wykorzystując pochodne cząstkowe funkcji kosztu względem wag na podstawie reguły łańcuchowej.



Rys. 3. Przykłady różnych struktur połączeń neuronów. Sieci Hopfielda (po lewej), połączenia każdy z każdym   
(po prawej). [31]

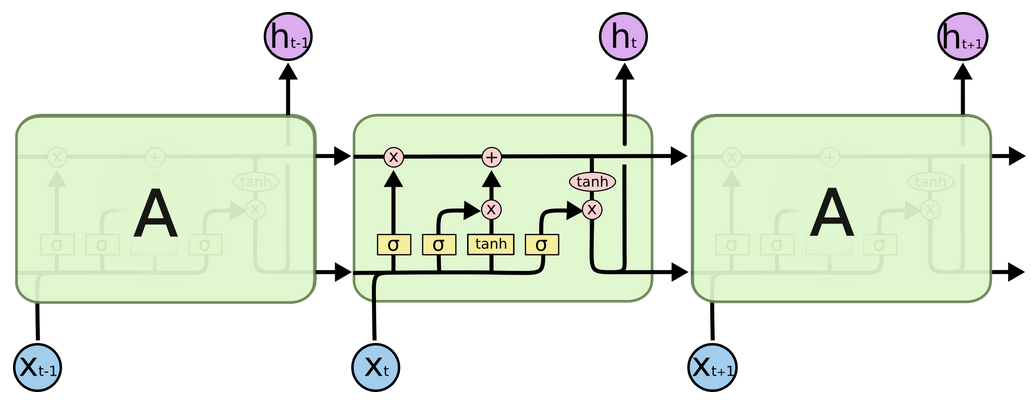
Na przestrzeni lat powstało wiele rodzajów sieci, różniących się strukturą i zastosowaniem (Rys. 3). Najprostszym przykładem mogą być wielowarstwowe perceptrony, w których sygnał porusza się w tylko jednym kierunku i każdy neuron kolejnej warstwy jest połączony ze wszystkimi neuronami warstwy poprzedniej. Sieci konwolucyjne składają się między innymi z warstw konwolucyjnych, które symulują działanie filtra przesuwnego, dzięki połączeniu neuronu z fragmentem poprzedniej warstwy należącym do pewnego otoczenia. Takie sieci dobrze sprawdzają się w analizie problemów, w których wejścia mają strukturę przestrzenną, jak obrazy czy szeregi czasowe. Sieci rekurencyjne wykorzystują dodatkowe połączenia wewnątrz warstw, tworząc pętle, które przechowują informacje o poprzednich stanach w jakich znajdowały się neurony. Szczególnie użyteczne są w analizie sekwencji. W RL różne sieci neuronowe znalazły zastosowanie między innymi przy aproksymacji funkcji wartości [13] (sieć rekurencyjna GRU), polityki [24] (wielowarstwowy perceptron) czy modeli środowiska [29] (wielowarstwowy perceptron).

## Sieci rekurencyjne

Sieci rekurencyjne (RNN) to typ sieci neuronowych, który powstał z myślą o analizie danych sekwencyjnych. Podstawowym elementem sieci rekurencyjnych jest komórka rekurencyjna, która działa na zasadzie sprzężenia zwrotnego. Każdy neuron w takiej komórce przekazuje swoje wyjście nie tylko do neuronów kolejnej warstwy, ale także do samego siebie. W ten sposób, w sieci powstaje pamięć, która pozwala uwzględniać poprzednie stany przy przetwarzaniu bieżącego wejścia. Ta zdolność do przechowywania kontekstu jest kluczowa przy rozwiązywaniu problemów, w których istotna jest kolejność i kontekst danych.

W praktyce istnieje kilka wariantów sieci rekurencyjnych, które różnią się strukturą i zastosowaniami. Najprostszym typem RNN są klasyczne sieci rekurencyjne, jednak ich efektywność bywa ograniczona przez problemy związane z zanikiem gradientu [38]. Aby temu zaradzić, opracowano bardziej zaawansowane struktury, takie jak długoterminowa pamięć krótkotrwała (LSTM) oraz sieci z jednostkami GRU (Gated Recurrent Unit). LSTM i GRU posiadają mechanizmy bramek, które regulują przepływ informacji, umożliwiając efektywne przechowywanie i wykorzystanie długoterminowych zależności.

Jednostka LSTM (przedstawiona schematycznie na Rys. 4) zawiera trzy główne rodzaje bramek: bramkę zapominania, bramkę wejściową oraz bramkę wyjściową. Bramka zapominania (Forget Gate) decyduje, które informacje z poprzedniego stanu komórki powinny zostać zapomniane. Bramka wejściowa (Input Gate) określa, które nowe informacje powinny być dodane do stanu komórki. Składa się ona z dwóch części: warstwy sigmoidalnej oraz warstwy z funkcją tangens hiperboliczny. Bramka wyjściowa (Output Gate) określa, które informacje ze stanu komórki będą używane jako wyjście [38]. Dzięki specjalnej architekturze z bramkami, LSTM mogą skutecznie przechowywać i wykorzystywać informacje przez dłuższy czas niż klasyczne sieci rekurencyjne, a także lepiej regulują przepływ gradientu ograniczając jego zanikanie.



Rys. 4. Schemat komórki LSTM, X: wektor wejściowy, h: wektor wyjściowy [39]

# Algorytmy uczenia ze wzmocnieniem

Niniejszy rozdział ma za zadanie przedstawić przegląd algorytmów uczenia ze wzmocnieniem. Zaprezentowane zostaną ich ogólne zasady działania oraz najważniejsze cechy.

## Q-learning

Q-learning jest algorytmem zaproponowanym przez Chrisa Witkinsa w 1989 roku [18]. Jest to metoda typu model-free, off-policy. Jest podstawą, na której opracowano wiele późniejszych rozwiązań, np. omówione w dalszej części TD3 i DDPG. Opiera się na funkcji wartości akcji , która ocenia jakość wykonania danej akcji w stanie . Algorytm aktualizuje wartości tej funkcji na podstawie doświadczeń zdobywanych podczas interakcji z otoczeniem. Działanie algorytmu można przedstawić następująco [19]:

W każdym kroku uczenia *t* agent:

1. Obserwuje aktualny stan *t*
2. Wybiera i wykonuje akcję *at*
3. Obserwuje następny stan
4. Otrzymuje natychmiastowy zwrot nagrody *rt*
5. Dostosowuje funkcję według wzoru (5), *αt* – współczynnik uczenia (learning rate)

|  |  |  |
| --- | --- | --- |
|  |  | (5) |
| Gdzie: | |  |
|  |  | (6) |

Z początku funkcja była reprezentowana przy pomocy tablicy look-up table, jednak dziś coraz popularniejsze staje się wykorzystywanie jej aproksymacji, co pozwala rozszerzyć obszar zastosowań do bardziej złożonych problemów, w tym także na problemy z ciągłą przestrzenią stanów. Szczególnie interesujące okazało się stosowanie głębokich sieci neuronowych. Takie algorytmy nazywane są Deep Q-Learning (DQN). Problemem takiego rozwiązania jest jednak, to, że nie ma pewności co do zbieżności polityki. Warunkiem zbieżności algorytmu Q-learning jest fakt, że może przejść przez każdą parę stan-akcja nieskończoną liczbę razy i learning rate jest odpowiednio redukowany [20]. Warunek ten jest spełniony tylko w przypadku dyskretnej przestrzeni stanów. Algorytmy te wymagają akcji dyskretnych, co w niektórych przypadkach może wymuszać dyskretyzację.

## Twin Delayed Deep Deterministic Policy Gradient

Algorytmy klasy Deterministic Policy Gradient (DPG) przedstawione przez Silvera i in. w pracy [23] są metodami opartymi o gradient polityki. Algorytmy DPG są pozbawione modelu środowiska. Zaliczają się do nich zarówno rozwiązania off-policy, jak i on-policy. Zostały opracowane, aby rozwiązywać problemy ciągłej przestrzeni akcji.

W przypadku metod stochastycznych polityka mapuje stany na prawdopodobieństwa wykonania danej akcji. Metody deterministyczne są szczególnym przypadkiem takich metod, gdy prawdopodobieństwo jednej z nich jest zawsze równe 1. Pozwala to na przyspieszenie wyznaczania , ponieważ nie trzeba sprawdzać wyników każdej akcji w następujących stanach. Gradient oczekiwanego zwrotu w tych metodach zależy przede wszystkim od gradientu polityki oraz gradientu funkcji wartości jak określono we wzorze (6) [23]:

|  |  |  |
| --- | --- | --- |
|  |  | (6) |

Gdzie:

– wektor parametrów deterministycznej polityki : S

- rozkład stanów dla polityki, opisuje, jak często agent znajduje się w różnych stanach środowiska podczas jego interakcji z nim

Rozwinięciem DPG jest zaprezentowana w [24] Deep DPG (DDPG) metoda, która wykorzystuje głębokie sieci neuronowe do skalowania rozwiązania na bardziej złożone problemy. DDPG wprowadza wykorzystanie sieci docelowych zarówno dla aktora, jak i krytyka w celu stabilizacji treningu. Sieci docelowe są powoli aktualizowane, aby zapobiec gwałtownym wahaniom w procesie uczenia się. Rozwiązanie przedstawione w [24] proponuje również wykorzystanie strategii eksploracji, poprzez dodanie szumu Ornsteina-Uhlenbecka do polityki deterministycznej, aby zapewnić wystarczającą eksplorację przestrzeni działania. Pozwala to otrzymać gładszy sygnał szumu, względem zupełnie losowych wartości, co umożliwia bardziej realistyczną eksplorację np. w środowiskach o pewnej bezwładności.

Twin Delayed Deep Deterministic Policy Gradient (TD3) to algorytm uczenia ze wzmocnieniem, który jest szczególnym przypadkiem metody Actor-Critic. TD3 został opracowany jako rozwinięcie algorytmu DDPG w celu poprawienia stabilności i wydajności, zwłaszcza pod kątem przeceniania wartości z powodu błędów aproksymacji. Istotną cechą algorytmu jest zastosowanie dwóch sieci krytyków, w ten sposób do aktualizacji parametrów funkcji aproksymujących może zostać wykorzystana minimalna wartość Q. Drugim ważnym elementem jest opóźnienie aktualizacji polityki. Algorytm rzadziej dokonuje jej aktualizacji względem zmian w parametrach krytyków. Ponadto do akcji otrzymywanych z docelowej polityki dodawany jest szum w trakcie trenowania, ograniczając eksploatację błędów. Zastosowanie takich rozwiązań ogranicza przeszacowania wartości Q. Algorytm TD3 pozwala na rozwiązywanie problemów z ciągłą przestrzenią akcji bez znajomości modelu środowiska.

## Soft Actor-Critic

Soft Actor-Critic (SAC) [40] jest zaawansowanym algorytmem uczenia ze wzmocnieniem, który opiera się na maksymalizacji entropii w celu poprawy eksploracji i stabilności polityki. SAC ma architekturę off-policy aktor-krytyk, działa w ciągłej przestrzeni akcji. Algorytm ocenia bieżącą politykę za pomocą iteracyjnego obliczania wartości soft Q, która do klasycznej wartości Q szacującej nagrody dodaje entropię polityki (wzór 7). Następnie polityka jest aktualizowana w kierunku eksponenty wartości soft Q.

W każdym stanie algorytm dąży do maksymalizacji entropii, co promuje większą różnorodność zachowań i lepszą eksplorację przestrzeni stanów i działań. Parametr temperatury, reprezentujący relację między istotnością składnika entropii i nagrody we wzorze na wartość Q, jest dostosowywany automatycznie. Umożliwia to utrzymanie równowagi między eksploatacją a eksploracją, eliminując konieczność ręcznego dostrajania hiperparametrów.

|  |  |  |
| --- | --- | --- |
|  |  | (7) |

Gdzie:

* - entropia,
* – parametr temperatury.

SAC wykorzystuje dwie sieci krytyków oraz jedną sieć aktora, co pozwala na redukcję wariancji w ocenie wartości Q i prowadzi do bardziej stabilnych aktualizacji polityki. Sieci krytyków uczą się przez minimalizację błędu Bellmana (3), natomiast sieć aktora optymalizuje politykę, maksymalizując wartości soft Q. Dzięki temu podejściu, SAC jest w stanie efektywnie radzić sobie z zadaniami o wysokim wymiarze przestrzeni akcji, które są trudne dla innych algorytmów.

## Conservative Q-Learning

Conservative Q-Learning [41] (CQL) to algorytm klasy batch RL, a więc przeznaczony do uczenia offline, który ma na celu rozwiązanie problemu przeszacowywania wartości Q-funkcji w wyniku przesunięcia dystrybucji między zbiorem danych a uczoną polityką (w zbiorze uczącym rozkład stanów i akcji może być istotnie różny względem środowiska, w którym docelowo ma działać). Algorytm CQL uczy konserwatywnej funkcji Q, tak aby oczekiwana wartość polityki była dolnym ograniczeniem jej rzeczywistej wartości. Minimalizuje wartości funkcji Q względem określonej dystrybucji par stan-akcja. Istotnym elementem CQL jest zmodyfikowany standardowy błąd Bellmana, uwzględniający regularyzację Q-wartości. Podczas trenowania Q-funkcji algorytm minimalizuje wartości Q względem dystrybucji danych oraz maksymalizuje wartości Q względem dystrybucji behawioralnej. CQL może być zaimplementowany na bazie istniejących algorytmów deep Q-learning i actor-critic poprzez dodanie warunków regularyzacji Q-funkcji. Jest szczególnie przydatny w sytuacjach, gdzie mamy do czynienia z ograniczoną ilością danych uczących lub gdy dane te pochodzą z różnych źródeł, co zwiększa ryzyko błędnych przewidywań funkcji Q.

## Inne algorytmy

We wcześniejszych rozdziałach przedstawiono kilka przykładów popularnych algorytmów uczenia ze wzmocnieniem, jednak w omawianej dziedzinie opracowano oczywiście znacznie więcej metod, różniących się zastosowaniem i skutecznością.

REINFORCE to klasa algorytmów, która dokonuje bezpośredniej optymalizacji polityki poprzez maksymalizację oczekiwanej nagrody. Algorytm ten działa w ramach podejścia on-policy, a więc uczy się na podstawie polityki, którą aktualnie stosuje agent. Algorytmy REINFORCE dostosowują wagi w kierunku leżącym wzdłuż gradientu oczekiwanego wzmocnienia zarówno w zadaniach z natychmiastową nagrodą, jak i w pewnych ograniczonych formach zadań ze zwrotem opóźnionym, i robią to bez jawnego obliczania szacunków gradientu lub nawet przechowywania informacji, na podstawie których można by dokonać takich szacunków [21]. Metoda nie wymaga znajomości modelu środowiska przez agenta. REINFORCE wykorzystuje do aktualizacji polityki oszacowane skumulowane zwroty uzyskane z zestawu próbkowanych trajektorii za pomocą metody Monte Carlo. Najczęściej używanym wariantem tego algorytmu jest forma z bazą, której celem jest zmniejszenie wariancji generowanej przy oszacowywaniu gradientu.

Proximal Policy Optimization (PPO) [25] jest jedną z nowoczesnych metod uczenia ze wzmocnieniem. Oparta jest na gradientach polityki i Trust Region Policy Optimization (TRPO), jednocześnie upraszczając implementację i zwiększając stabilność treningu. Jest to metoda off-policy, niewykorzystująca modelu środowiska. PPO wprowadza funkcję celu, która ogranicza zmiany w polityce. W przeciwieństwie do tradycyjnych metod, które mogą powodować zbyt gwałtowne modyfikacje, PPO je kontroluje, co prowadzi do bardziej stabilnego uczenia. Proces nauki polega na poszukiwaniu wektora wag polityki, dla której oczekiwana wartość funkcji celu będzie największa. PPO używa operacji "clipping" w celu ograniczenia stosunku prawdopodobieństw wykonania akcji na podstawie nowej polityki względem starszej. Operacja ta ogranicza wartości stosunku do określonego zakresu, co zapobiega zbyt dużym zmianom. Istnieje też drugi wariant PPO-penalty, który zamiast wprowadzania ograniczenia dodaje do funkcji celu karę za zbyt duże różnice. Współczynnik kary jest dostosowywany automatycznie w trakcie treningu. PPO jest stabilniejszy niż tradycyjne metody gradientów polityki. Może być stosowany do problemów zarówno w środowiskach dyskretnych, jak i ciągłych.

Do tej pory w pracy skupiono się na metodach, które nie potrzebowały modelu środowiska. Warto jednak pamiętać, że jest to skuteczne rozwiązanie tylko części problemów stawianych przed uczeniem ze wzmocnieniem. Przykładem algorytmu należącego do klasy model-based może być PILCO [27]. Aby modelować zachowanie dynamiczne środowiska użyto w nim modeli probabilistycznych. Pozwalają one na przewidywanie, jak zmiany w sterowaniu wpłyną na przyszłe stany środowiska. W procesie optymalizacji wykorzystuje on metody gradientowe do znalezienia optymalnego sterowania uwzględniając niepewności dynamiczne. Na podstawie kolejnych doświadczeń modele są aktualizowane poprawiając jakość predykcji.

Innym wartym uwagi algorytmem jest MBMF [28] (Model-Based Model-Free), który łączy dwa różne podejścia do uczenia ze wzmocnieniem, czyli metody oparte o model środowiska z tymi, które są go pozbawione. Model w MBMF wykorzystywany jest do przewidywania trajektorii na podstawie pojedynczego przejścia, natomiast polityka docelowa jest optymalizowana, dzięki oszacowaniu kosztów przy pomocy procesu Gaussowskiego.

# Regulacja instalacji grzewczych

Instalacje grzewcze odgrywają kluczową rolę w zapewnieniu komfortu cieplnego i efektywności energetycznej budynków. W dobie rosnącej świadomości ekologicznej oraz wzrastających kosztów energii, optymalna regulacja tych systemów staje się nie tylko pożądana, ale wręcz niezbędna. Odpowiednie sterowanie w takim systemie wymaga nie tylko osiągnięcia zamierzonych warunków cieplnych, ale również ograniczenia zużycia energii. Ponieważ jest to problem bardzo szeroko rozpowszechniony, potencjalne korzyści w zakresie poprawy jakości życia, ograniczenia kosztów, oraz zmniejszenia niekorzystnego wpływu człowieka na środowisko mogą być znaczące.

Docelowym środowiskiem, na którym zostanie zrealizowana część praktyczna, będzie system symulujący rozbiór ciepłej wody użytkowej (szczegółowy opis instalacji znajduje się w kolejnych rozdziałach). Systemy tego typu charakteryzują się stosunkowo dużą bezwładnością, istotne jest, także, aby utrzymywać temperaturę w odpowiednim zakresie, zwłaszcza w górnej granicy, gdyż zbyt wysoka temperatura może prowadzić do poparzeń. Powinny być w stanie szybko reagować na zmiany zapotrzebowania, które mogą być gwałtowne i nieprzewidywalne z uwagi np. na zmieniającą się liczbę użytkowników.

Niniejszy rozdział przedstawi przykładowe rozwiązania proponowane przez badaczy związanych z dziedziną regulacji ogrzewania, zwłaszcza w obszarze ciepłej wody użytkowej.

## Dotychczasowe rozwiązania – metody tradycyjne

Bardzo popularnymi metodami sterowania instalacjami grzewczymi są systemy wykorzystujące regulatory PID. Typowy regulator PID działa w pętli sprzężenia zwrotnego, poprzez trzy podstawowe mechanizmy. Część proporcjonalna (P) generuje sygnał proporcjonalny do bieżącego błędu sterowania. Część całkująca (I) uwzględnia sumę wszystkich wcześniejszych błędów w czasie, co eliminuje stały błąd. Część różniczkująca (D) reaguje na szybkość zmiany błędu, pomagając w tłumieniu oscylacji i poprawie stabilności systemu. Jest to bardzo proste i wydajne rozwiązanie, jednak w wielu przypadkach, wyniki mogą być niezadowalające. W nowych publikacjach, regulator PID często jest łączony z innymi metodami, aby osiągać lepsze wyniki. Jednym z przykładów może być praca [6], w której autorzy prezentują porównanie 3 kontrolerów w systemie elektrycznego zbiornika wody grzewczej. Dzięki zastosowaniu logiki rozmytej możliwa była znacząca poprawa wyników, zarówno pod względem czasu, jak i dokładności regulacji. Podobnych rozwiązań można znaleźć wiele więcej: [5], [6], [7].

Kontrolery oparte o logikę rozmytą (z ang.  *fuzzy logic controller*), również są bardzo popularną metodą sterowania, co pokazuje już część prac wymienionych powyżej. Takie regulatory wykorzystują logikę wielowartościową. Sygnały wejściowe takiego regulatora poddawane są w pierwszej kolejności fuzzifikacji, czyli określany jest ich stopień przynależności do zbiorów rozmytych sygnału wejściowego, by następnie stopień ten zmapować, na podstawie bazy reguł na stopień przynależności do zbiorów rozmytych sygnału wyjściowego. Na koniec dokonywana jest defuzzifikacja, otrzymanego sygnału, co jest operacją odwrotną do fuzzifikacji [8]. Zastosowanie takiego mechanizmu można wykorzystać zarówno samodzielnie ([4], [8]), jak i we współpracy z innymi metodami (z PID: [6], z uczeniem maszynowym: [9]). Do głównych zalet kontrolerów logiki rozmytej należy zdolność radzenia sobie z nieprecyzyjnymi i nieliniowymi systemami, oferując większą elastyczność i bardziej intuicyjne projektowanie dzięki użyciu reguł opartych na języku naturalnym. Jednakże, ich wady obejmują trudności w doborze odpowiednich funkcji przynależności i reguł oraz potencjalnie większe wymagania obliczeniowe w porównaniu do klasycznych regulatorów PID.

Powyższe rozwiązania mogą jednak być niewystarczające w przypadku bardziej złożonych systemów, a także gdy zależy nam na większych oszczędnościach energii. W takich sytuacjach przydatne mogą okazać się bardziej zaawansowane propozycje. Na przykład przedstawiony w pracy [10] algorytm pozwala na zaoszczędzenie od 8% do 28% energii, dzięki zastosowaniu algorytmu przewidującego obecność osób w pomieszczeniach. Autorzy porównywali wyniki z tradycyjnym systemem regulacji temperatury opartym o timer, który pozwala na dostosowywanie systemu do różnych wymagań w określonych porach dnia.

Thilker i in. w artykule [11] wprowadzili koncepcję predykcyjnego sterowania nieliniowego modelu na potrzeby inteligentnego ogrzewania budynków, w którym wykorzystuje się inteligentne termostaty i czujniki Internetu Rzeczy w celu umożliwienia precyzyjnego sterowania systemami grzewczymi za pomocą sterowania predykcyjnego. Podejście to oznacza przejście w stronę bardziej wyrafinowanych technik sterowania, które starają się odpowiednio przewidywać i dostosowywać do zmieniających się warunków.

## Dotychczasowe rozwiązania – uczenie maszynowe

W systemach ogrzewania, obiecującym podejściem do zwiększania efektywności energetycznej, optymalizacji wydajności i zapewniania komfortu użytkowników okazało się wykorzystanie uczenia maszynowego. W proponowanej w pracy [12] metodyce zastosowano model uczenia maszynowego, który wykorzystuje zarówno zmierzone dane (np. temperatura wewnętrzna/zewnętrzna, wilgotność względna), jak i dane prognozowane (np. dane meteorologiczne), aby trenować model regresji liniowej wielokrotnej do prognozowania temperatury w analizowanym pomieszczeniu. Następnie, przy użyciu metody optymalizacji algorytmem genetycznym, model ten ocenia różne strategie ogrzewania. Każdej strategii przypisuje się ocenę na podstawie zdefiniowanych przez użytkownika kryteriów, co pozwala na priorytetyzację i wybór najlepszej strategii.

W celu poprawy wydajności systemów HVAC (ang. heating, ventilation, air conditioning), w badaniu [13] zaproponowano metodę hybrydową WDQN-temPER, która łączy sieć deep Q-learning (DQN) z techniką priorytetowego odtwarzania doświadczeń (PER) oraz modelem jednostki rekurencyjnej (GRU). Model GRU przewiduje przyszłe temperatury zewnętrzne, które są wykorzystywane jako zmienne stanu w modelu RL, natomiast PER pozwala na efektywniejsze wykorzystywanie doświadczeń.

W artykule [14] autorzy proponują sformułowanie projektowania układu sterowania jako procesu decyzyjnego Markowa z użyciem głębokich sieci neuronowych i wykorzystanie algorytmu głębokiego uczenia przez wzmacnianie opartego na głębokich gradientach polityki deterministycznej (DDPG) do znalezienia optymalnej strategii sterowania systemami HVAC, która równoważy koszty energii elektrycznej i komfort użytkowników.

Warto również wskazać pracę [15], w której Blad i in. zaproponowali uczenie ze wzmocnieniem wykorzystujące wielu agentów, w systemie HVAC, aby osiągnąć minimalny czas trenowania, działając on-line. Zaprezentowany algorytm został zaprojektowany, aby posiadał zdecentralizowaną strukturę, w której tylko istotne stany są udostępniane między agentami, a akcje są sekwencyjnie koordynowane. Większość agentów dobiera akcje na podstawie stanu środowiska, a na koniec tzw. mixing agent wybiera akcje, na podstawie stanu środowiska i akcji z pozostałych agentów.

# Narzędzia implementacji

Niniejszy rozdział ma za zadanie przedstawić narzędzia implementacji, które zdecydowano się wykorzystać przy testowaniu wybranych algorytmów uczenia ze wzmocnieniem w ramach części praktycznej analizy ich zastosowań.

## Python

Python to interpretowany język programowania wysokiego poziomu. Jest to język obiektowy, jednak wspiera również paradygmaty programowania funkcjonalnego i proceduralnego [32]. Składnia oraz dostęp do wielu różnorodnych bibliotek takich jak Pandas, Keras, czy NumPy sprawia, że jest to język wszechstronny, a przy tym prosty i   
czytelny, umożliwiając zawarcie wielu skomplikowanych obliczeń w krótkim kodzie. Jest udostępniany w formacie open source, a jego interpreter jest dostępny na wielu systemach   
opartych o Unix oraz na systemie Windows. Do głównych zastosowań należą kolejno: analiza danych, tworzenie stron internetowych i uczenie maszynowe [33]. W pracy wykorzystano wersję 3.10.12.

## Tensorflow

TensorFlow to otwarta biblioteka opracowana przez Google. Pozwala programistom budować, trenować i wdrażać modele uczenia maszynowego, umożliwiając im tworzenie zaawansowanych aplikacji opartych na sztucznej inteligencji. Główną koncepcją w TensorFlow są tensory, które są wielowymiarowymi macierzami używanymi do reprezentowania danych w modelach uczenia maszynowego. Umożliwia realizację projektów w wielu środowiskach: na procesorach, kartach graficznych, TPU, w przeglądarkach internetowych i wielu więcej [34]. W pracy wykorzystano wersję 2.15.0.

## Tf-Agents

TF-Agents to biblioteka stworzona do rozwoju agentów uczenia ze wzmocnieniem w oparciu o TensorFlow. Jest to narzędzie do budowy, trenowania i ewaluacji różnych rodzajów agentów uczenia ze wzmocnieniem, takich jak agenty oparte na głębokim uczeniu czy actor-critic. Oferuje gotowe implementacje popularnych algorytmów uczenia ze wzmocnieniem, takich jak DQN, SAC (Soft Actor-Critic) i PPO. Biblioteka ta jest rozwijana w ścisłej integracji z TensorFlow, co umożliwia wydajne trenowanie agentów na różnych środowiskach uczenia. W pracy wykorzystano TF-Agents w wersji 0.19.0.

## Optuna

Optuna to biblioteka do optymalizacji hiperparametrów, która umożliwia automatyczne znalezienie najkorzystniejszych wartości dla modeli uczenia maszynowego. Jest to narzędzie wykorzystywane do automatyzacji procesu, która może znacznie przyspieszyć proces eksperymentowania i poprawić wydajność modeli. Optuna wykorzystuje algorytmy optymalizacji, takie jak algorytm Tree-structured Parzen Estimator (TPE) lub algorytm Random Search, aby przeszukać przestrzeń hiperparametrów w celu znalezienia najlepszych kombinacji [35]. Zastosowano wersję 3.2.0.

## TCLab

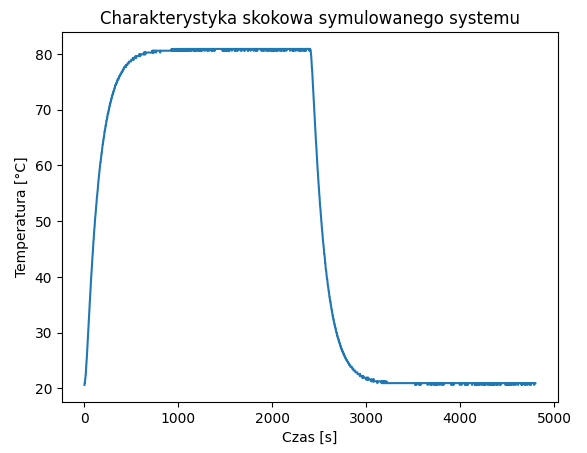
TCLab udostępnia interfejs Pythona do Laboratorium Kontroli Temperatury zaimplementowanego na mikrokontrolerze Arduino (Arduino Temperature Control Lab), komunikującym się przez interfejs USB [36]. Biblioteka umożliwia zarówno interakcję z rzeczywistym układem, jak również z zawartym w niej cyfrowym bliźniakiem umożliwiającym symulację systemu. Arduino Temperature Control Lab to modułowe, przenośne i niedrogie rozwiązanie do praktycznej nauki sterowania procesami. Moc cieplna jest regulowana przez modulację przepływu prądu do każdego z dwóch tranzystorów. Dwa termistory mierzą temperatury. Energia z wyjścia tranzystora jest przenoszona przez przewodzenie i konwekcję do czujnika temperatury. Laboratorium jest zintegrowane z małą płytką PCB, którą można zamontować na dowolnym Arduino lub kompatybilnym mikrokontrolerze. Zastosowano bibliotekę w wersji 1.0.0.

# Implementacja środowiska

Aby przyspieszyć uczenie agentów zdecydowano się zrealizować rozwiązanie problemu, przy pomocy technik zarówno uczenia offline, jak i online. W pierwszej kolejności wygenerowano zestaw danych historycznych opartych o regulator PI, dzięki czemu w początkowej fazie trenowania, agenci nie musieli wykonywać wielu błędnych prób manipulacji obiektem, zamiast tego obserwując trajektorie o poprawnym sterowaniu. Niniejszy rozdział przedstawi implementację środowiska i agentów oraz przygotowanie danych historycznych, które zostaną wykorzystane do uczenia modeli DQN, TD3, SAC i CQL-SAC.

## Środowisko symulacyjne

Środowisko na którym zostało przeprowadzone uczenie oraz pierwszy etap testów zostało oparte o cyfrowego bliźniaka płytki TCLab. Symulacja miała reprezentować system, w którym w pewnym niewielkim zbiorniku, należy utrzymać temperaturę wody na zmiennym zadanym poziomie. Sterowanie następowało poprzez zmiany mocy grzałki w zakresie od 0 do 100%. Dobór sygnału sterującego następował na podstawie zmierzonej za pomocą symulowanego czujnika temperatury oraz wygenerowanej wartości zadanej. Doświadczalnie sprawdzono, że temperatura w symulowanym zbiorniku może osiągać wartości z zakresu w przybliżeniu 20-80 stopni Celsjusza. Zmniejszenie różnicy o 95% między wartością minimalną a maksymalną trwa 440 sekund, a w przeciwną stronę 450 sekund. Charakterystyka skokowa takiego systemu została przedstawiona na Rys. 5.



Rys. 5. Charakterystyka skokowa symulowanego systemu.

Na podstawie powyższych danych zdecydowano zaimplementować generator nastaw, który z pomocą generatora liczb pseudo-losowych generuje wartość zadaną w danej chwili czasu w zakresie temperatur 30-70 °C, zmieniając ją nie częściej niż co 240 s i nie rzadziej niż co 600 s.

## Analiza danych procesu sterowania

Aby lepiej zrozumieć proces sterowania omawianym obiektem, przygotowano prosty regulator PID. Na podstawie charakterystyki skokowej (Rys. 1), wyznaczono czas osiągnięcia 10% (t0) oraz 90% (tk) wartości regulowanej. Na podstawie tych wartości i pierwszej metody Zieglera-Nicholsa [42] określono parametry regulatora, które przedstawiono w Tabeli 1.

Tabela 1. Parametry regulatora PID

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Parametr | t0 | tk | Kp=1,2 t0 / tk | Ti = 2 t0 | Td = 0,5 t0 |
| Wartość | 30s | 364s | 14,55 | 60,06 | 15,02 |

W oparciu o te parametry przygotowano kod w Pythonie, realizujący algorytm regulatora PID, przedstawiony jako fragment kodu 1.

*Fragment kodu 1. Regulator PID*

class PID():

    def \_\_init\_\_(self):

        self.Kc   = 5.95

        self.tauI = 63

        self.tauD = 0.0

        self.prev\_y = 0

        self.epsilon\_integral = 0

        self.prev\_t = 0

    def control(self, observation, time):

        y, sp = observation[0], observation[1]

        error = sp - y

        dt = time - self.prev\_t

        dy = y - self.prev\_y

        self.epsilon\_integral += error

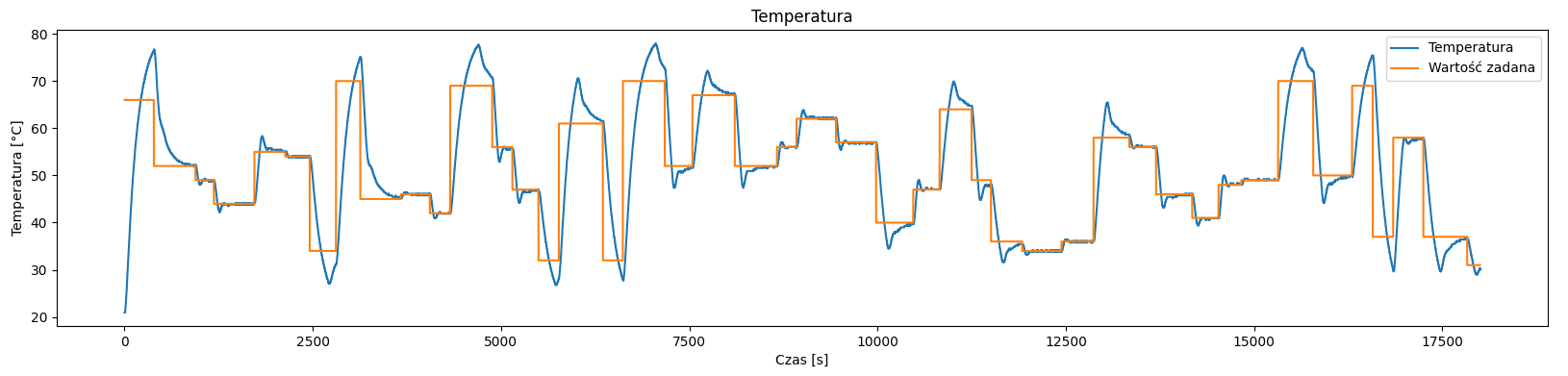
        derivative = dy / (dt + 1e-10)

        control = self.Kc \* (error + self.epsilon\_integral/self.tauI + self.tauD\*derivative)

        control = min(100, max(control, 0)) / 100

        return control

Z pomocą powyższego regulatora PID wygenerowano trajektorię trwającą 300 minut otrzymując wyniki przedstawione na Rys. 6. Na wykresie można zauważyć istotne przeregulowania, które można zredukować np. poprzez zastosowanie mechanizmu anti-windup, ponieważ jednak w przypadku uczenia ze wzmocnieniem algorytmy uczą się nie poprzez porównywanie wartości wyjściowych z oczekiwanym wynikiem, a poprzez analizę wpływu obserwacji i akcji na nagrodę, zdecydowano pozostawić regulator w takiej formie. Dzięki temu dane uczące są bardziej zróżnicowane i obejmują szerszy zakres obserwacji.



Rys. 6. Trajektoria danych historycznych.

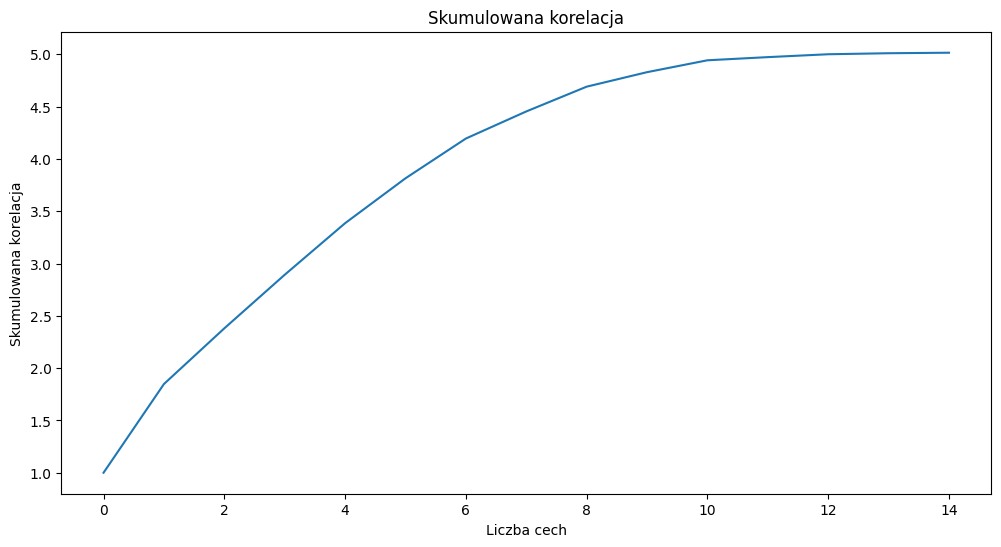
W następnym kroku dane o stanie środowiska zostały rozszerzone o zestaw nowych zmiennych, które przeanalizowano pod kątem nowych informacji jakie mogą wnieść do obserwacji. Wybrano następujące:

* Błąd sterowania
* Średnia temperatury z małym (10 próbek) i większym oknem (50 próbek)
* Wariancja temperatury z małym (10 próbek) i większym oknem (50 próbek)
* Wariancja błędu sterowania (10 próbek)
* Wariancja wartości zadanej (10 próbek)
* Pochodna temperatury
* Pochodna błędu

W pierwszej kolejności porównano korelację między zmiennymi, a wartościami nagród i akcji i odrzucono dane najmniej skorelowane . Tabela 1. przedstawia posortowane wartości korelacji, a Rys. 7 skumulowane wartości. Zdecydowano się odrzucić 7 najmniej istotnych cech przy wyborze akcji, pozostawiając 5, odpowiedzialnych za ponad 90% kumulatywnej korelacji.

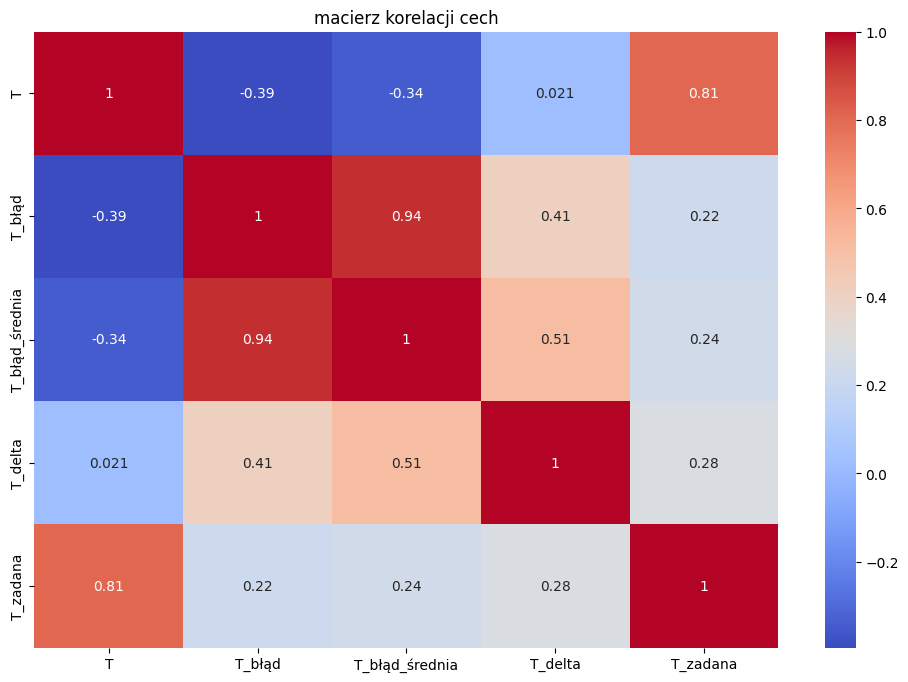
Tabela 2. Porównanie korelacji danych z akcjami i nagrodami.

|  |  |
| --- | --- |
| Korelacja z nagrodami | Korelacja z akcjami |
| Nagrody 1.000000  T\_wariancja 0.593321  T\_błąd\_wariancja 0.565889  T\_wariancja50 0.497708  T\_zadana\_wariancja 0.409474  T\_błąd 0.206179  T\_błąd\_średnia 0.183500  T\_zadana 0.178269  Akcje 0.138503  T\_delta 0.101496  T\_średnia50 0.057325  T\_średnia 0.024010  T 0.014561  T\_błąd\_delta 0.010564 | Akcje 1.000000  T\_zadana 0.847947  T\_delta 0.532413  T\_błąd\_średnia 0.512210  T\_błąd 0.492969  T 0.429218  T\_średnia 0.380073  T\_wariancja50 0.258904  T\_średnia50 0.236935  Nagrody 0.138503  T\_wariancja 0.113657  T\_zadana\_wariancja 0.030224  T\_błąd\_delta 0.027474  T\_błąd\_wariancja 0.009768 |



Rys. 7. Skumulowana korelacja cech z akcjami.

Następnie na podstawie macierzy korelacji (Rys. 7) zauważono wysoką zależność między temperaturą, błędem sterowania oraz średnią błędu, dlatego spośród nich wybrano tylko błąd sterowania, ponieważ poziom korelacji z nagrodą i akcjami jest na podobnym poziomie, co wartość uśredniona i wyższy niż temperatura, natomiast upraszcza to implementację środowiska. Pochodna temperatury również została odrzucona z uwagi na poziom korelacji z błędem na poziomie 0,41 oraz niską korelację z nagrodami.



Rys. 8.Macierz korelacji między cechami.

Na podstawie powyższej analizy, jako obserwacje z badanego środowiska przyjęto wartość zadaną oraz błąd sterowania.

## Interfejs środowiska

Opisany w podrozdziale 5.1 system sterowania został opakowany w klasę o interfejsie kompatybilnym z obiektami z biblioteki tf-agents, aby umożliwiać prosty sposób komunikacji z predefiniowanymi agentami i umożliwiać generowanie danych w odpowiednim formacie. Do wykonania jednego kroku algorytmu wymagane są następujące funkcje:

* Funkcja *reset* – przywraca środowisko do stanu początkowego. Inicjalizuje laboratorium oraz generator nastaw, a następnie zwraca pierwszy TimeStep, czyli obiekt zawierający aktualne obserwacje, nagrodę oraz rodzaj danego kroku w epizodzie (pierwszy, środkowy lub ostatni). Fragment kodu 2. przedstawia jej implementację.

*Fragment kodu 2. Funkcja reset*

    def \_reset(self):

...

            # zamyka poprzednie laboratorium jeśli istnieje

            try:

                self.lab.close()

            except AttributeError:

                pass

           # Otwiera nowe laboratorium z nowym zegarem i generatorem nastaw

            lab =  tclab.setup(connected=False, speedup=self.SPEEDUP)

            self.lab = lab()

            self.clk = tclab.clock(self.episode\_time)

            self.\_T\_gen = setpoint\_gen(self.clk, self.\_seed)

        self.\_\_set\_control(0)

        self.\_\_state\_update()

        self.\_episode\_ended = False

        self.\_current\_time\_step = ts.restart(self.state.as\_array())

        return self.\_current\_time\_step

* Funkcja *step* – reprezentuje kolejny krok procesu, przyjmuje akcję, którą aplikuje jako sterowanie. Ponadto sprawdza, czy czas epizodu się skończył, obserwuje aktualny stan środowiska oraz wyznacza nagrodę. Na koniec zwraca te informacje jako TimeStep. Poniżej przedstawiono implementację tej funkcji jako fragment kodu 3.

*Fragment kodu 3. Funkcja step*

def \_step(self, action):

    try:

        self.time = next(self.clk)

        self.done = (self.time >= self.episode\_time)

    except StopIteration: # Jeśli czas epizodu przekroczył limit

        self.done = True

    self.\_\_set\_control(action)

    self.\_\_state\_update()

    self.reward = self.\_\_calculate\_reward(action)

        current\_reward = np.array(self.reward, dtype=np.float32)

      current\_state = self.state.as\_array()

if self.done:

self.\_current\_time\_step = ts.termination(current\_state,

current\_reward)

else:

self.\_current\_time\_step = ts.transition(current\_state,

current\_reward)

    return self.\_current\_time\_step

## Akcje i nagrody

Sposób aplikowania akcji do symulowanej grzałki różni się w przypadku agenta DQN względem pozostałych, ponieważ operuje on na akcjach dyskretnych. Należało więc określić sposób dyskretyzacji, aby każdej wartości z zakresu 0-100 przypisać liczbę całkowitą. Zdecydowano wybrać najprostszy sposób dyskretyzacji, gdzie wartości sterowania rosną liniowo względem indeksu akcji. Dyskretyzacja została przedstawiona we wzorze (8), a proces odwrotny we wzorze (9).

|  |  |  |
| --- | --- | --- |
|  |  | (8) |

|  |  |  |
| --- | --- | --- |
|  |  | (9) |

Gdzie:

* – sterowanie grzałką
* – rozdzielczość dyskretyzacji
* – akcja ciągła
* – akcja dyskretna o indeksie

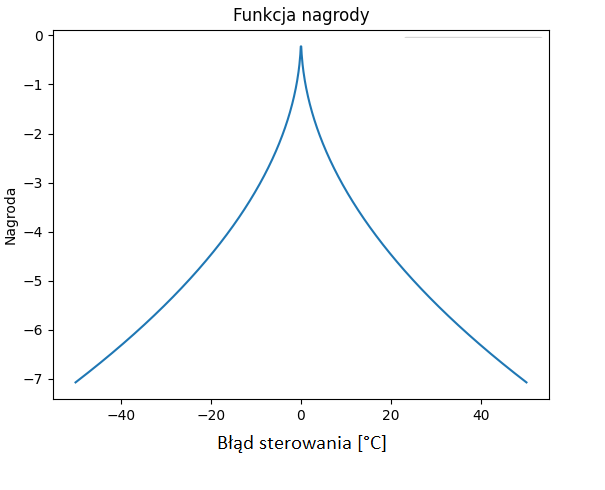
W przypadku agentów o ciągłej przestrzeni akcji wykorzystano normalizację akcji do zakresu 0-1, dlatego przed przekazaniem jako sygnał sterujący są przycinane, a normalizacja zostaje odwrócona.

Odpowiednia definicja nagrody jest jednym z kluczowych problemów uczenia ze wzmocnieniem, które ma wpływ na stabilność i zbieżność procesu uczenia. W badanym środowisku głównym celem sterowania jest minimalizacja uchybu temperatury, przy możliwie niskim koszcie energetycznym. W celu zniwelowania błędu sterowania zastosowano wzór (10) zwizualizowany na Rys. 9. Dzięki wykorzystaniu pierwiastka w definicji nagrody modele w większym stopniu skupiały się na nauce sterowania w otoczeniu wartości zadanej, były więc w stanie dokładniej śledzić zadaną temperaturę.

|  |  |  |
| --- | --- | --- |
|  |  | (10) |

Gdzie:

* – temperatura
* – wartość zadana temperatury



Rys. 9. Funkcja nagrody od błędu sterowania.

Minimalizacja wykorzystanej energii jest jednak bardziej problematyczna. Jeżeli ten element będzie zbyt istotny w definicji nagrody, algorytm nauczy się utrzymywać zbyt niską temperaturę lub w skrajnym przypadku nastawi stałe, zerowe sterowanie. Mając na uwadze powyższe, rozwinięto wzór (10) o dodatkowy człon dodający karę za wysokie wartości sterowania. Jest on uzależniony od wartości zadanej, dzięki czemu, kara jest mniejsza, gdy wymagana temperatura wymusza mocniejsze ogrzewanie. Wzór (11) przedstawia takie rozwinięcie.

|  |  |  |
| --- | --- | --- |
|  |  | (11) |

Gdzie:

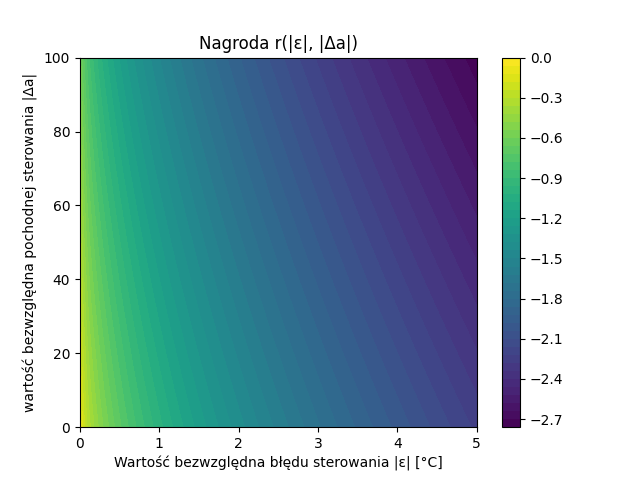
* – waga składnika minimalizującego błąd
* – waga składnika minimalizującego sterowanie

Ponadto zauważono, że nauczone sterowanie jak zostanie pokazane w kolejnych podrozdziałach charakteryzuje się dużymi wahaniami sygnału. Zamiast ustalić ogrzewanie na danym poziomie temperatura oscyluje w niewielkim zakresie wokół wartości zadanej. Rozważono, więc również dodanie członu ograniczającego zmiany sterowania, co zostało przedstawione we wzorze (12) oraz zwizualizowane na Rys. 10..

|  |  |  |
| --- | --- | --- |
|  |  | (12) |

Gdzie:

* – waga składnika minimalizującego błąd
* – waga składnika minimalizującego wahania sterowania

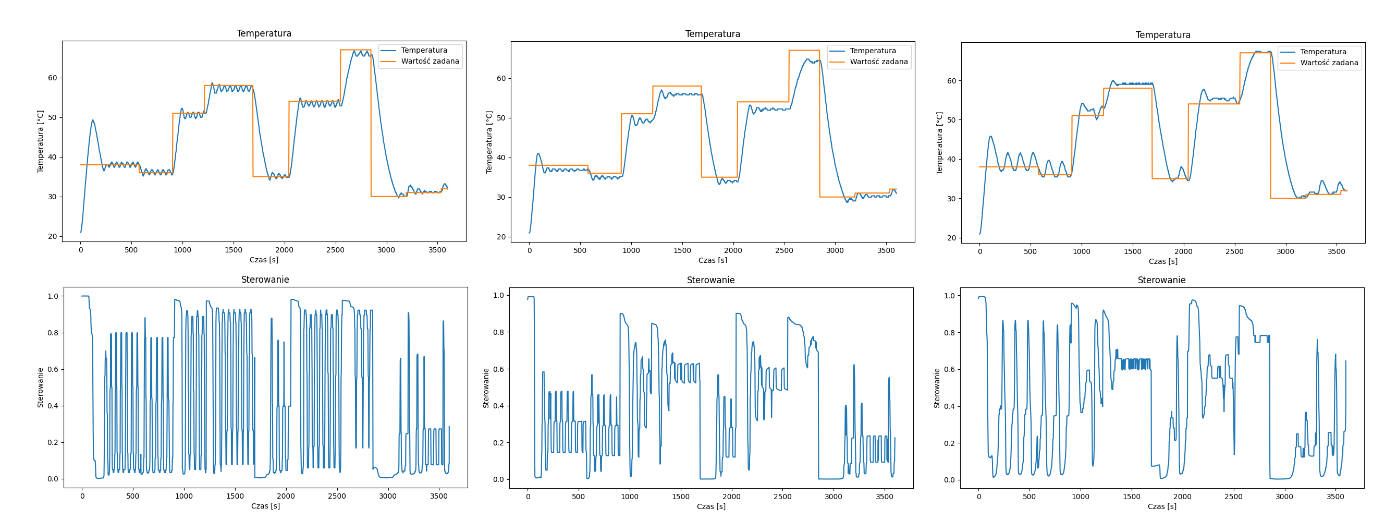


Rys. 10. Wykres funkcji nagrody dla Ce = 1,2,

Różne definicje nagrody zostały przetestowane na agencie SAC, na wspólnych danych, liczbie kroków i parametrach agenta określonych w Tabeli 6. Najistotniejsze wyniki przedstawia Tabela 3. oraz Rys. 10. Można zauważyć, że rozwiązanie ze wzoru (12) osiąga najgorsze wskaźniki, natomiast opcja ze wzoru (11) osiągnęła średni uchyb gorszy o 18% i niższe zużycie energii o 6% względem pierwszej metody. Ponieważ wyniki są porównywalne zdecydowano się wybrać rozwiązanie najprostsze wg. wzoru (10).

Tabela 3. Porównanie wpływu definicji nagrody na uczenie agenta SAC.

|  |  |  |  |
| --- | --- | --- | --- |
| Nagroda | Wg. wzoru (10) | Wg. wzoru (11)  =1, =14 | Wg. wzoru (12)  =1, =1.2 |
| Średnie zużycie energii % | 41.19 | 38.58 | 42.94 |
| Średnia nagroda | -1.32 | -1.65 | -1.51 |
| Średni uchyb bezwzględny temperatury [°C] | 3.23 | 3.80 | 3.62 |
| Czas utrzymany w komfortowym zakresie [s] | 2727.99 | 2259.08 | 2259.08 |



Rys. 11. Porównanie nauczonego sterowania dla nagrody wg. wzoru (10) po lewej, (11) po środku, (12) po prawej.

## Agent DQN

Agent DQN jest algorytmem opartym o funkcję wartości. W zaimplementowanym modelu do jej estymacji została wykorzystana klasa QRnnNetwork. Przygotowanie agenta wymaga podania następujących parametrów:

* Specyfikacja danych wejściowych i wyjściowych
* Struktura sieci neuronowej – liczba parametrów w poszczególnych warstwach
* Funkcja aktywacji wykorzystywana w warstwach fully-connected (FC)
* Optymalizator określający algorytm optymalizacji spadku gradientu, w tym przypadku zastosowano Adam.
* Współczynnik uczenia, określający szybkość optymalizacji.
* Funkcja strat używana do obliczania błędów tzw. Temporal Difference - różnicy między przewidywanymi a rzeczywistymi wartościami Q.
* Parametr epsilon w strategii epsilon-greedy. Określa prawdopodobieństwo, z jakim agent wybiera losową akcję zamiast najlepszej znanej akcji, co pozwala na eksplorację środowiska. Na etapie uczenia offline jest nieistotnym parametrem z uwagi na brak bezpośredniej interakcji ze środowiskiem.
* Parametr tau dla aktualizacji sieci docelowej (target network). Określa, jak szybko sieć docelowa jest aktualizowana w stosunku do sieci głównej.
* Okres aktualizacji sieci docelowej określający, co ile kroków symulacji sieć docelowa jest aktualizowana.
* Parametr dyskontowania gamma. Definiuje, jak bardzo przyszłe nagrody są dyskontowane w procesie uczenia się.
* Współczynnik skalowania nagród otrzymywanych przez agenta, wpływający na tempo i stabilność procesu uczenia się agenta.

Warto zauważyć, że agent dqn\_agent.DqnAgent posiada dwie sieci, podstawową oraz docelową, druga jest aktualizowana na podstawie zmian dokonanych w wielu krokach aktualizacji sieci podstawowej. W opracowanej implementacji, sieć docelowa jest kopią pierwszej.

## Algorytmy aktor-krytyk

Algorytm TD3 w przeciwieństwie do DQN opiera się na kooperacji modelu aktora i krytyka. Wykorzystano więc dwie sieci rekurencyjne ActorRnnNetwork oraz CriticRnnNetwork. Są one inicjalizowane analogicznie do QRnnNetwork. Każda z nich posiada własny optymalizator Adam. Istotnym parametrem, wprowadzonym zamiast epsilon, jest odchylenie standardowe szumu eksploracyjnego. Pozwala on na wprowadzanie eksploracji w ciągłej przestrzeni akcji. Wprowadzony zostaje także parametr umożliwiający przycinanie gradientów, co może poprawić stabilność uczenia.

W implementacji algorytmu SAC wykorzystano tę samą klasę dla sieci krytyka, natomiast dla aktora zastosowano klasę ActorDistributionRnnNetwork, generującą rozkład normalny akcji. Kluczowym elementem metody Soft Actor-Critic jest składnik entropii i związany z nim współczynnik α. Agent SAC optymalizuje go automatycznie przy pomocy trzeciego optymalizatora. Możliwe jest też określenie jego początkowej wartości.

CQL-SAC działa analogicznie do podstawowej wersji SAC, wprowadza jednak kilka istotnych nowych parametramów:

* cql\_alpha - kontroluje wpływ regularyzacji CQL na proces uczenia się.
* Liczba próbek używanych do obliczenia straty CQL
* Wariancja szumu dodawanego do nagrody

## Optymalizacja hiperparametrów

Kluczowym etapem prac nad modelami uczenia maszynowego jest optymalizacja hiperparametrów. W tym celu wykorzystana została biblioteka Optuna. Proces ten został przeprowadzony dwukrotnie, w pierwszym przypadku opierając się na danych historycznych, a następnie na interakcji ze środowiskiem.

Ogólny schemat optymalizacji offline wyglądał następująco:

1. Uzupełnienie bufora doświadczeń trajektorią z zastosowaniem regulatora PID
2. Utworzenie obiektu study, który wykona określoną liczbę prób:
   1. Wybór badanych w danej próbie wartości hiperparametrów agentów, oraz rozmiaru i kolejności przetwarzanych danych.
   2. Inicjalizacja agenta.
   3. Inicjalizacja iteratora do pobierania danych z bufora doświadczeń.
   4. Trening przez określoną liczbę kroków.
   5. Ewaluacja polityki po 2/3 i całości przeprowadzonego treningu.
   6. Zwrot najlepszej oceny jako wartości funkcji celu próby.

Ewaluacja polityki jest dokonywana online na 30 minutowych symulacjach, a maksymalizowaną funkcją celu jest suma nagród w trakcie epizodu.

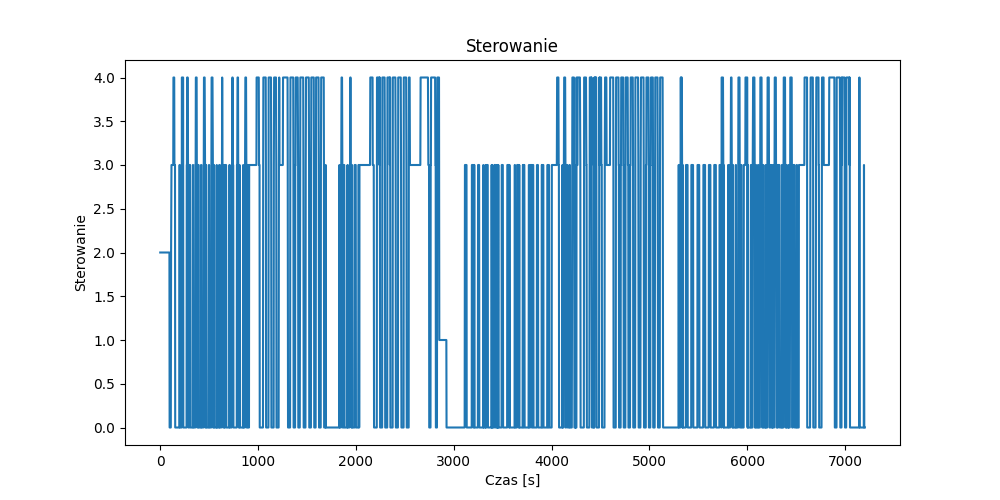
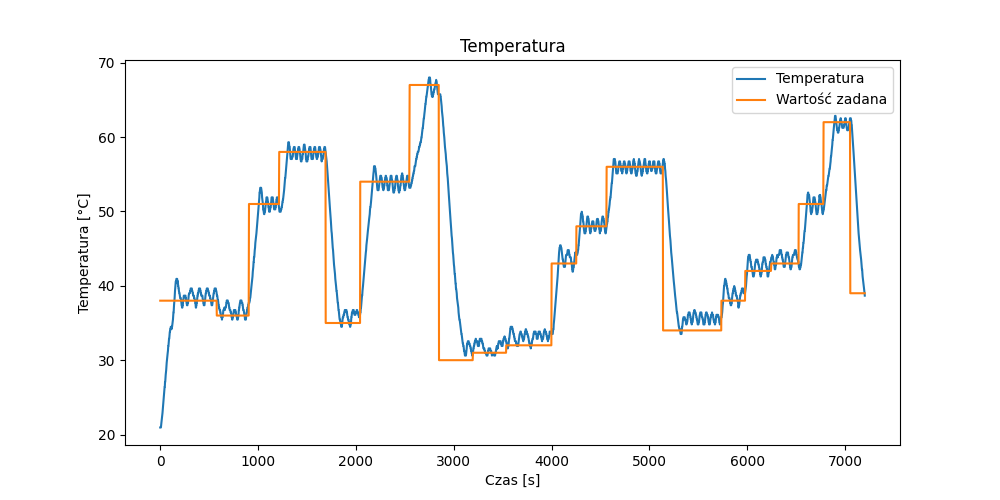
## Efekty uczenia offline

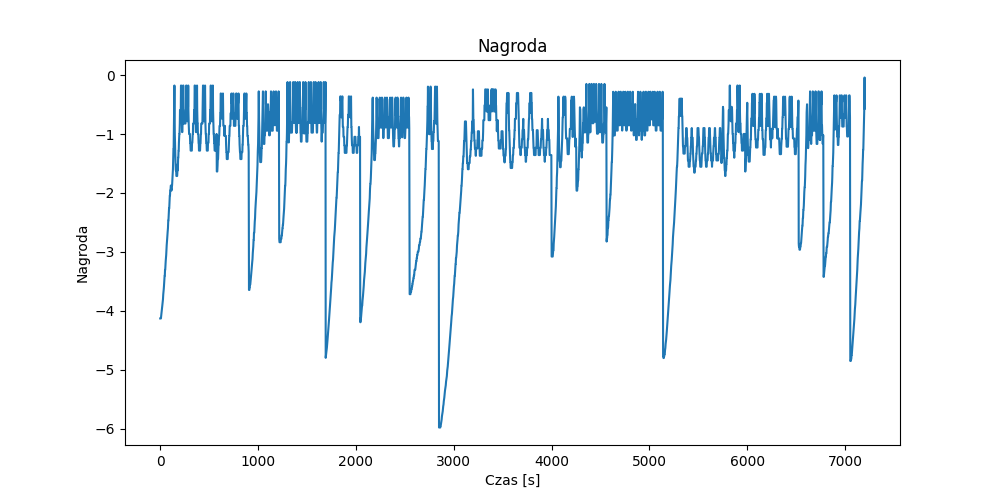
W Tabelach 4-7 przedstawiono hiperparametry agentów wybranych do porównania na etapie uczenia offline. Zostały one wykorzystane do wytrenowania strategii, które następnie zostały przetestowane w symulacji przez 120 minut.

**DQN**

Tabela 4. Parametry agenta DQN otrzymane w wyniku optymalizacji.

|  |  |
| --- | --- |
| Parametry sieci | |
| Rozmiar warstw wejściowych FC | (209, 148) |
| Rozmiar komórek LSTM | 77 |
| Rozmiar warstw wyjściowych FC | (216, 100) |
| Funkcja aktywacji | SELU |
| Współczynnik uczenia | 3,33e-4 |
| Parametry agenta | |
| Tau | 0,005 |
| Okres aktualizacji sieci docelowej | 1 |
| epsilon | 0,1 |
| gamma | 0,96 |
| Współczynnik skalowania nagrody | 0,99 |
| Parametry danych uczących | |
| Rozmiar batchy | 256 |
| Liczba kroków czasowych analizowanych jako sekwencja | 4 |
| Liczba akcji | 5 |

****

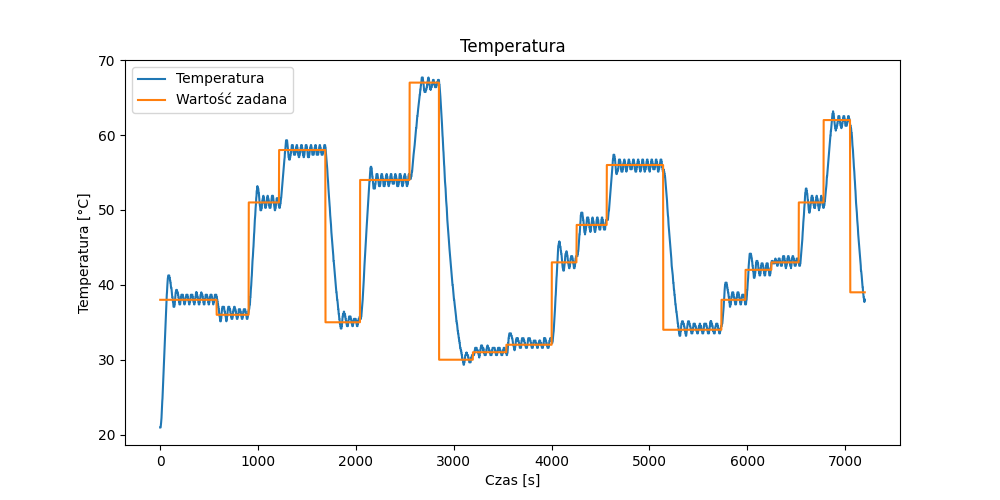


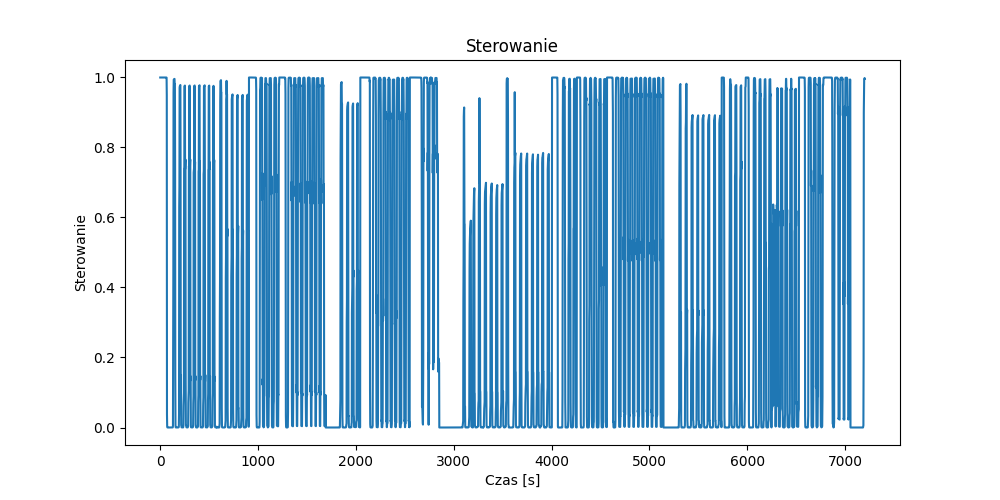
Rys. 12. Wykresy trajektorii dla DQN uczonego offline.

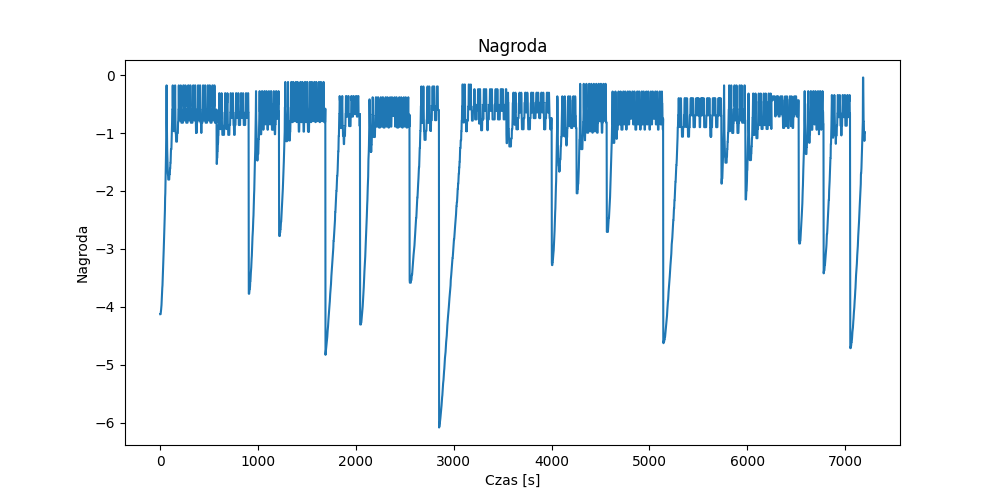
**TD3**

Tabela 5. Parametry agenta TD3 otrzymane w wyniku optymalizacji.

|  |  |
| --- | --- |
| Parametry sieci aktora | |
| Rozmiar warstw wejściowych FC | (209, 148) |
| Rozmiar komórek LSTM | (77,) |
| Rozmiar warstw wyjściowych FC | (216, 100) |
| Funkcja aktywacji | RELU |
| Współczynnik uczenia | 6,01e-4 |
| Parametry sieci krytyka | |
| Rozmiar warstw wejściowych FC | Brak |
| Rozmiar komórek LSTM | (124,) |
| Rozmiar warstw wyjściowych FC | (96,100) |
| Funkcja aktywacji | RELU |
| Współczynnik uczenia | 5,43-3 |
| Parametry agenta | |
| Tau | 0,001 |
| Okres aktualizacji sieci docelowej | 1 |
| Okres aktualizacji sieci aktora | 6 |
| Odchylenie standardowe szumu eksploracji | 0,12 |
| gamma | 0,96 |
| Współczynnik skalowania nagrody | 0,99 |
| Parametry danych uczących | |
| Rozmiar batchy | 64 |
| Liczba kroków czasowych analizowanych jako sekwencja | 4 |

****

****

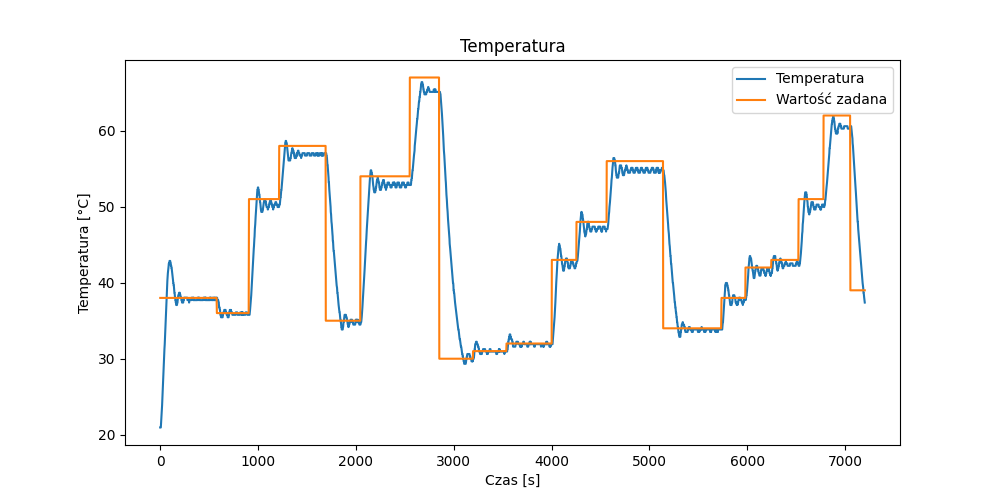
****

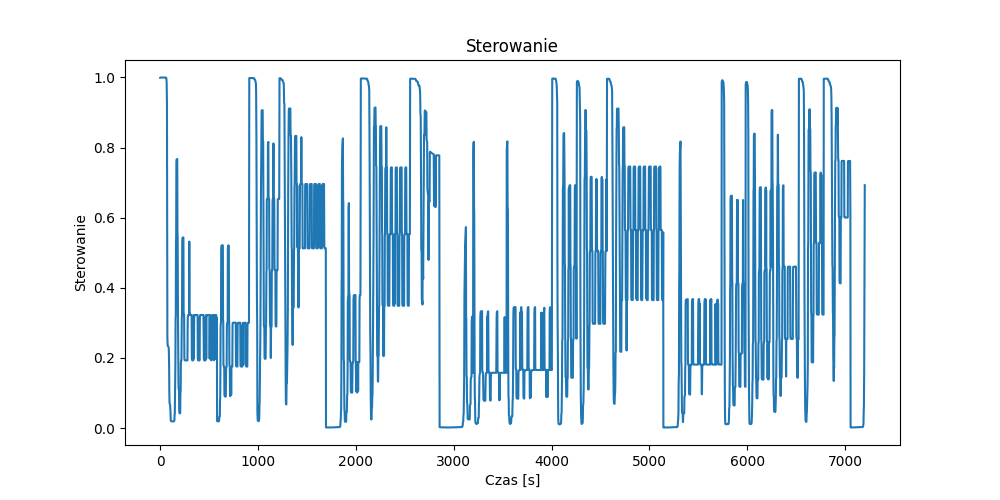
Rys. 13. Wykresy trajektorii dla TD3 uczonego offline.

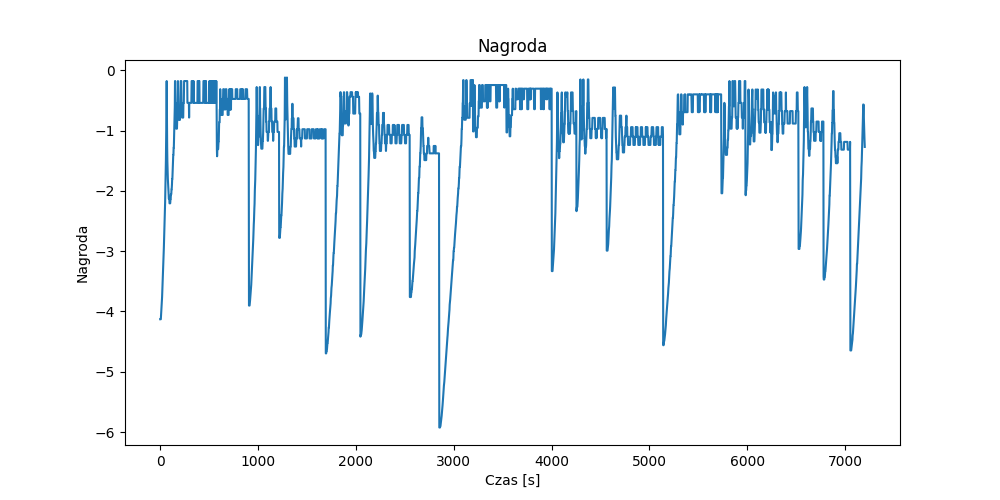
**SAC**

Tabela 6. Parametry agenta SAC otrzymane w wyniku optymalizacji.

|  |  |
| --- | --- |
| Parametry sieci aktora | |
| Rozmiar warstw wejściowych FC | (195,) |
| Rozmiar komórek LSTM | (101,) |
| Rozmiar warstw wyjściowych FC | (100,) |
| Funkcja aktywacji | RELU |
| Współczynnik uczenia | 1,88e-4 |
| Parametry sieci krytyka | |
| Rozmiar warstw wejściowych FC | Brak |
| Rozmiar komórek LSTM | (35,) |
| Rozmiar warstw wyjściowych FC | (105, 100) |
| Funkcja aktywacji | RELU |
| Współczynnik uczenia | 8,95e-3 |
| Parametry agenta | |
| Tau | 0.031 |
| Okres aktualizacji sieci docelowej | 1 |
| Okres aktualizacji sieci aktora | 3 |
| gamma | 0,97 |
| Współczynnik skalowania nagrody | 0.23 |
| Współczynnik uczenia parametru α | 8.85e-05 |
| Parametry danych uczących | |
| Rozmiar batchy | 64 |
| Liczba kroków czasowych analizowanych jako sekwencja | 12 |

****

****

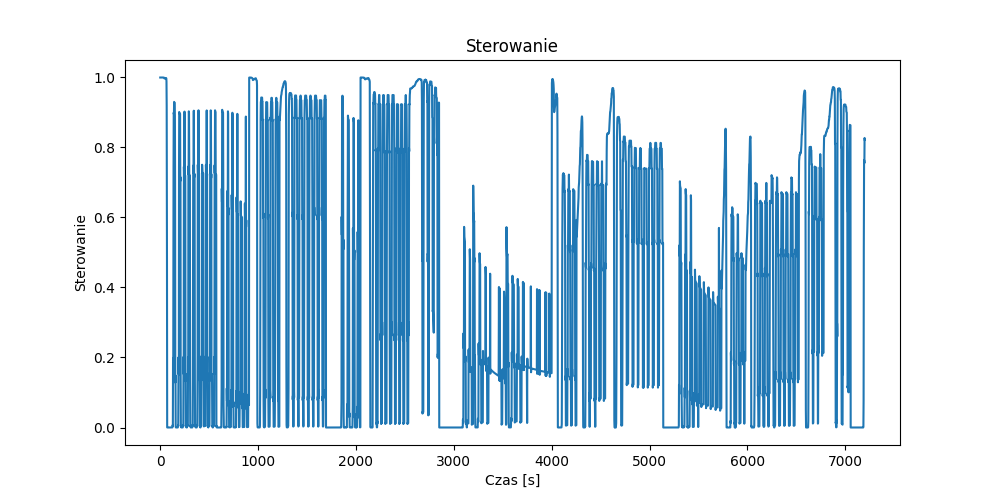
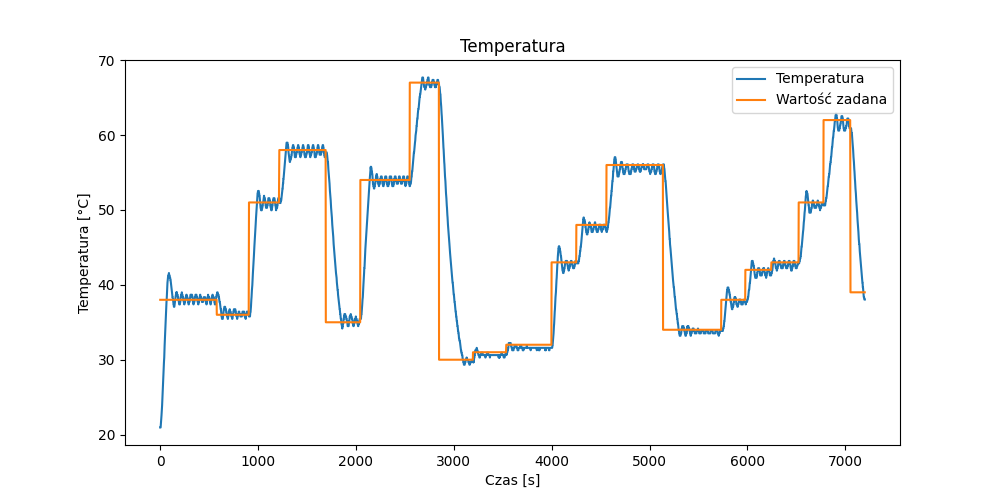
****

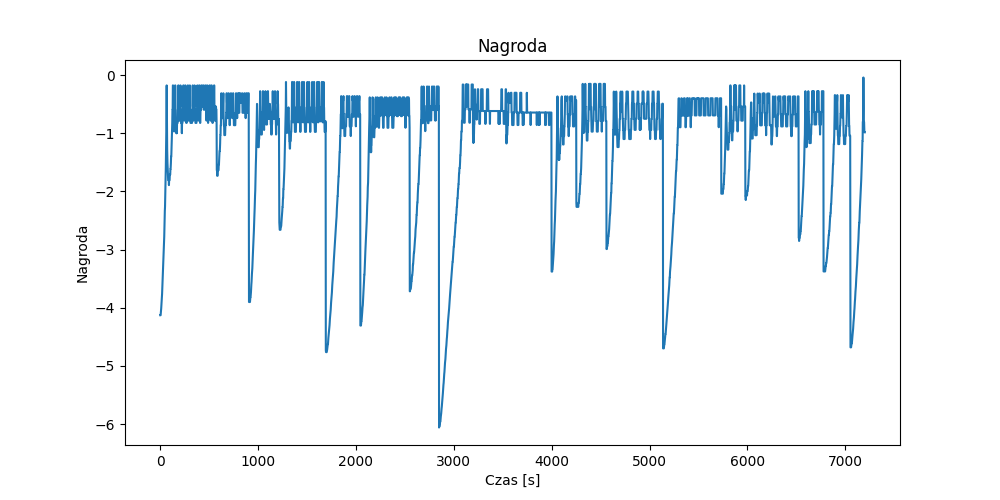
Rys. 14. Wykresy trajektorii dla SAC uczonego offline.

**CQL-SAC**

Tabela 7. Parametry agenta CQL-SAC otrzymane w wyniku optymalizacji.

|  |  |
| --- | --- |
| Parametry sieci aktora | |
| Rozmiar warstw wejściowych FC | (134,) |
| Rozmiar komórek LSTM | (103,) |
| Rozmiar warstw wyjściowych FC | (100,) |
| Funkcja aktywacji | SELU |
| Współczynnik uczenia | 4.61e-5 |
| Parametry sieci krytyka | |
| Rozmiar warstw wejściowych FC | Brak |
| Rozmiar komórek LSTM | (40,) |
| Rozmiar warstw wyjściowych FC | (100, 100) |
| Funkcja aktywacji | SELU |
| Współczynnik uczenia | 1.16e-3 |
| Parametry agenta | |
| Tau | 0.003 |
| Okres aktualizacji sieci docelowej | 1 |
| Okres aktualizacji sieci aktora | 3 |
| gamma | 0,86 |
| Cql-α | 0.88 |
| Współczynnik skalowania nagrody | 1.00 |
| Współczynnik uczenia parametru α | 8.85e-05 |
| Współczynnik uczenia parametru cql | 1.00e-5 |
| Parametry danych uczących | |
| Rozmiar batchy | 256 |
| Liczba kroków czasowych analizowanych jako sekwencja | 10 |





Rys. 15. Wykresy trajektorii dla CQL-SAC uczonego offline.

Wskazane na Rys. 12-15 trajektorie porównano między sobą oraz z regulatorem na podstawie którego były uczone. Wyznaczone zostały średnie wartości zużycia energii, średnia nagroda, średni uchyb bezwzględny oraz czas jaki wartość regulowana mieściła się w zakresie +- 2°C od wartości zadanej. Wyniki przedstawia Tabela 5. Najgorszym pod względem analizowanych wskaźników jest agent DQN. Prawdopodobnie wynika to z błędów kwantyzacji, które wymuszają ograniczony zakres dyskretnych akcji. Algorytm TD3 charakteryzuje się największą oscylacją wybieranych akcji, osiąga jednak przy tym porównywalne wyniki do pozostałych agentów. Pozwala to też na najszybsze śledzenie wartości zadanej na co wskazuje najdłuższy czas w komfortowym zakresie. Najkorzystniej energetycznie wypada algorytm SAC, jednak różnice pod tym względem są niewielkie i nie przekraczają 4%. Najmniejszy uchyb bezwzględny otrzymano przy pomocy agenta CQL-SAC. W Tabeli 8 przedstawiono również dla porównania osiągi regulatora PID zastosowanego podczas przygotowania danych historycznych. Energetycznie zajmuje on drugie miejsce spośród wskazanych metod sterowania, a pod względem uchybu i czasu w komfortowym zakresie znacząco odstaje (o kolejno 26% i 12% względem CQL-SAC). Warto zauważyć, że uczenie ze wzmocnieniem pozwoliło na osiągnięcie lepszych wyników niż wykorzystany PID, jednak należy pamiętać, że są różne metody wyznaczania nastaw PID, można było również zastosować mechanizm anti-windup, co mogłoby znacząco zniwelować przeregulowania, które są źródłem tych istotnych różnic. Można jednak uznać, że wyniki uczenia off-line były zadowalające.

Tabela 8. Porównanie jakości sterowania na etapie uczenia offline.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Agent | DQN | TD3 | SAC | CQL-SAC | PID |
| Średnie zużycie energii % | 41.09 | 40.98 | 39.74 | 40.24 | 39.97 |
| Średnia nagroda | -1.38 | -1.21 | -1.20 | -1.14 | -1.40 |
| Średni uchyb bezwzględny temperatury [°C] | 3.07 | 2.56 | 2.54 | 2.47 | 3.12 |
| Czas utrzymany w komfortowym zakresie [s] | 5279.99 | 5788.00 | 5765.02 | 5721.97 | 5008.08 |

## Efekty uczenia online

W kolejnym kroku podjęto próbę poprawienia wyników poprzez 120 minutową sesję dostrajania online (za wyjątkiem DQN, który był uczony przez 180 minut, ponieważ miał większe trudności z oceną wartości akcji). W przeciwieństwie do poprzedniego etapu, w którym cały zbiór danych był przechowywany w buforze powtórek, w uczeniu online bufor jest uzupełniany na bieżąco w trakcie interakcji z otoczeniem. Hiperparametry agentów, w większości zostały zachowane, jednak część z nich została dostosowana poprzez optymalizację analogiczną jak na etapie uczenia offline. Zmienione hiperparametry wymieniono w Tabeli 9. Tabela 10. przedstawia wyniki testów dostrojonych agentów. Poniżej, na Rys. 16-19. znajdują się również wykresy trajektorii dla których wyznaczono przedstawione wskaźniki.

Tabela 9. Zmienione parametry agentów przed uczeniem online.

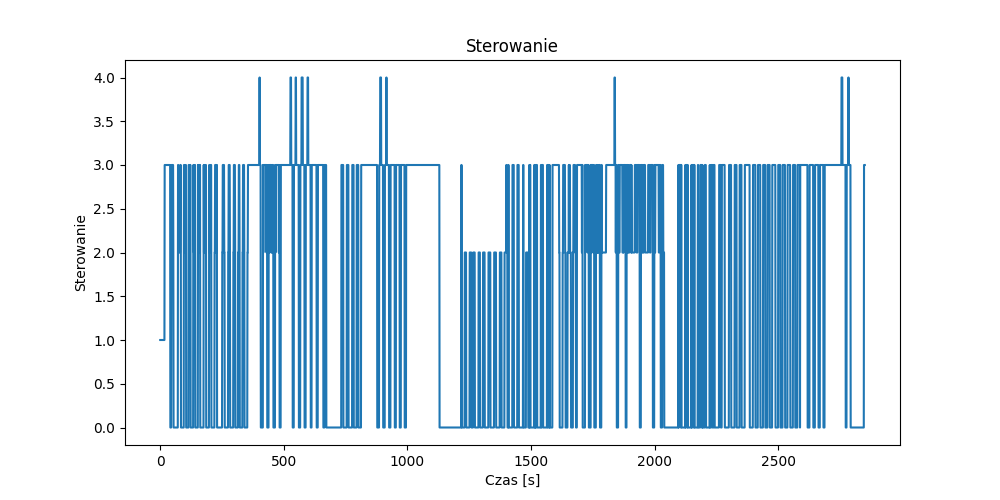
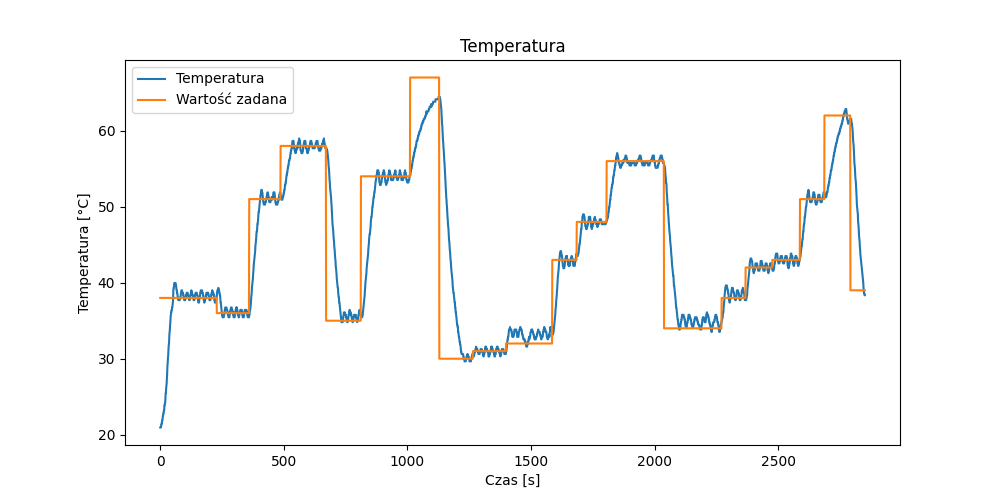
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parametr | DQN | TD3 | SAC | CQL-SQC |
| Współczynnik uczenia aktora | 6.01e-6 | 1.33e-05 | 3.76e-05 | 1.53e-06 |
| Współczynnik uczenia krytyka | - | 1.08e-04 | 1.79e-03 | 3.85e-05 |
| Współczynnik uczenia parametru α | - | - | 2.95e-06 | 2.47e-06 |
| Współczynnik uczenia parametru cql | - | - | - | 3.33e-07 |
| Odchylenie standardowe szumu eksploracji | - | 0.23 | - | - |
| epsilon | 0.06 | - | - | - |
| Rozmiar batchy | 256 | 256 | 256 | 256 |

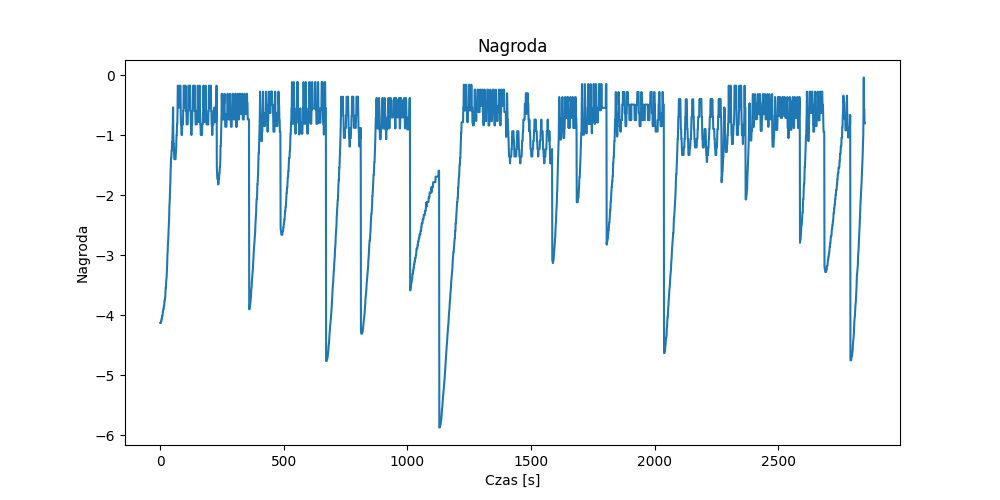
Tabela 10. Porównanie jakości sterowania na etapie uczenia online.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Agent | DQN | TD3 | SAC | CQL-SAC |
| Średnie zużycie energii % | 40.07 | 41.00 | 39.80 | 40.40 |
| Średnia nagroda | -1.26 | -1.18 | -1.19 | -1.15 |
| Średni uchyb bezwzględny temperatury [°C] | 2.78 | 2.44 | 2.53 | 2.46 |
| Czas utrzymany w komfortowym zakresie [s] | 5407.96 | 5881.98 | 5726.00 | 5762.02 |

Porównując wyniki z Tabeli 10 i Tabeli 8 można stwierdzić że po dostrojeniu algorytmy wykazują się, w większości nieco lepszą jakością sterowania. W przypadku DQN średnie zużycie energii zmniejsza się po dostrojeniu (2.5%), natomiast dla SAC, TD3 i CQL-SAC nieznacznie wzrasta (mniej niż 1%) . Podczas uczenia online wartości średniego uchybu bezwzględnego temperatury i czas komfortowej temperatury poprawiają się dla DQN (9% i 2%), oraz w mniejszym stopniu (<1%) dla TD3 i CQL-SAC, natomiast dla SAC wynik nie zmienia się lub nieznacznie pogarsza. Najlepszym agentem pod względem energetycznym jest nadal algorytm SAC, natomiast najmniejszy uchyb temperatury zapewnia TD3.

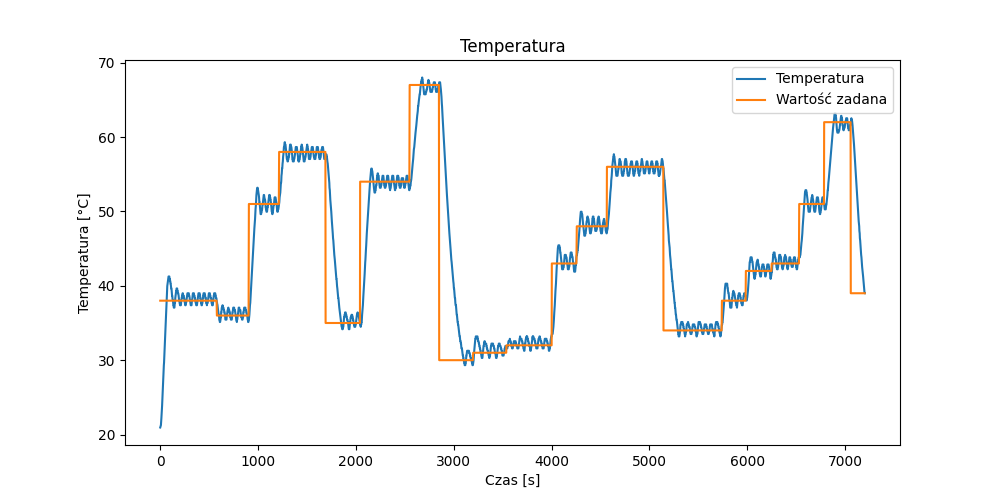
**DQN**

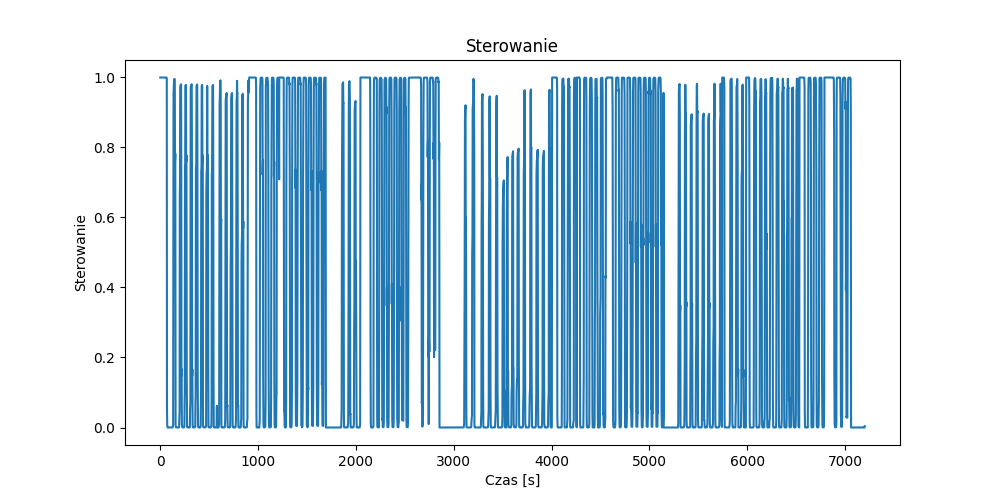
****

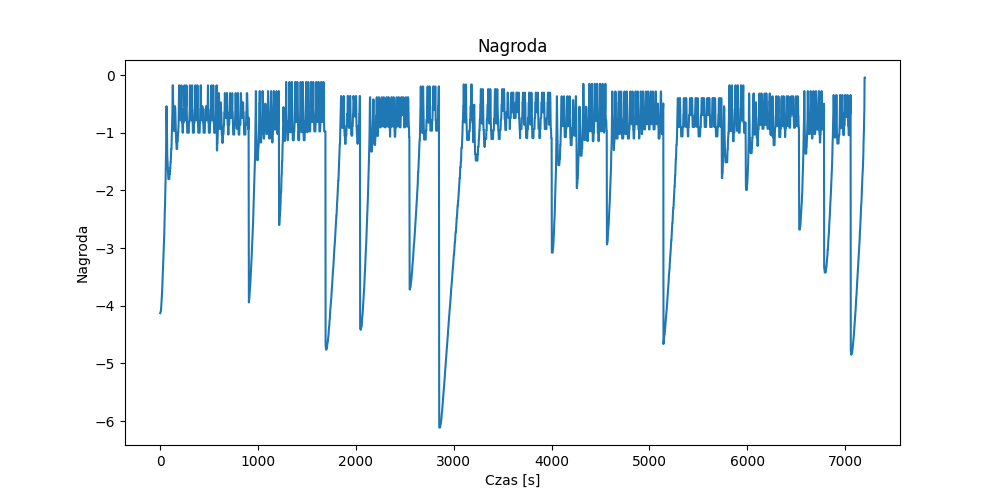
****

Rys. 16. Wykresy trajektorii dla DQN uczonego online.

**TD3**

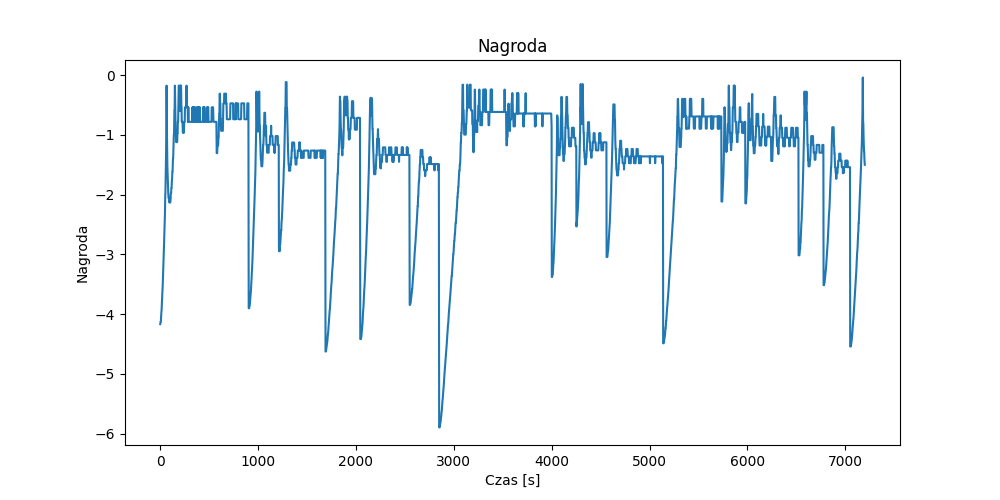
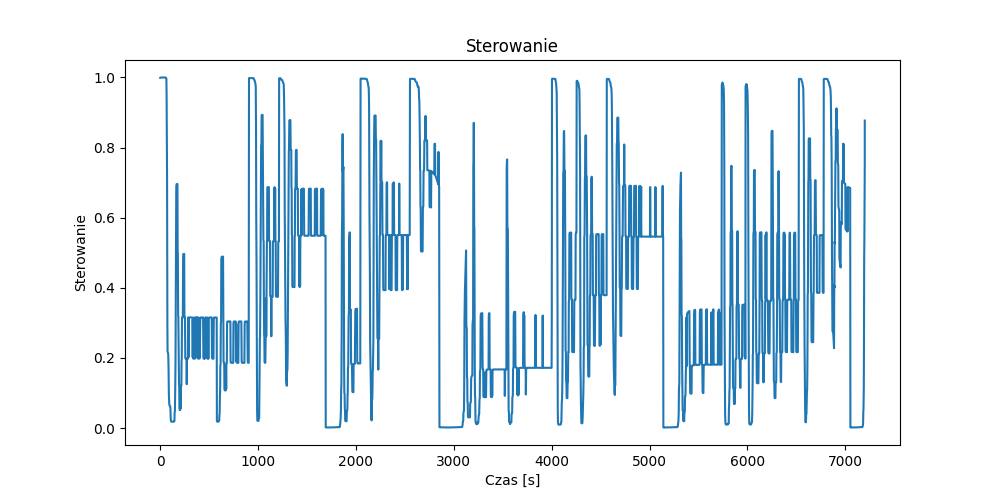
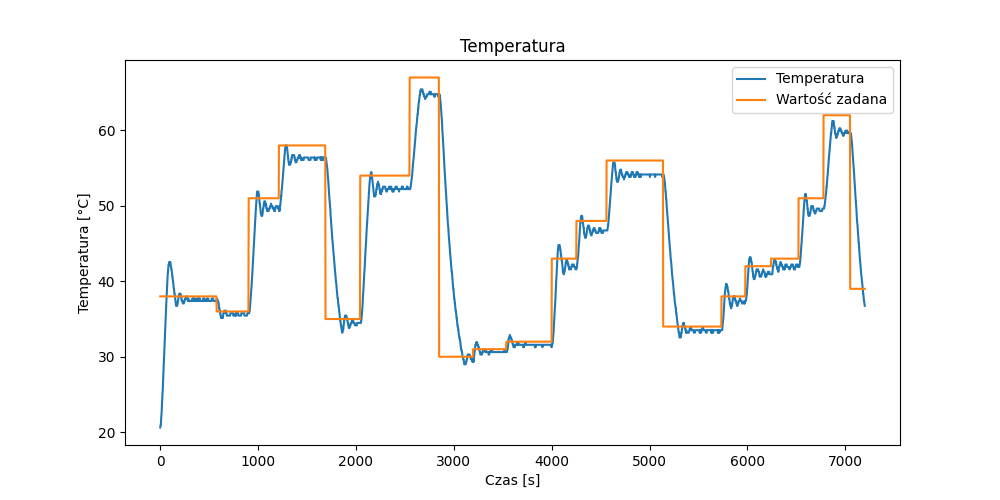
****

****

****

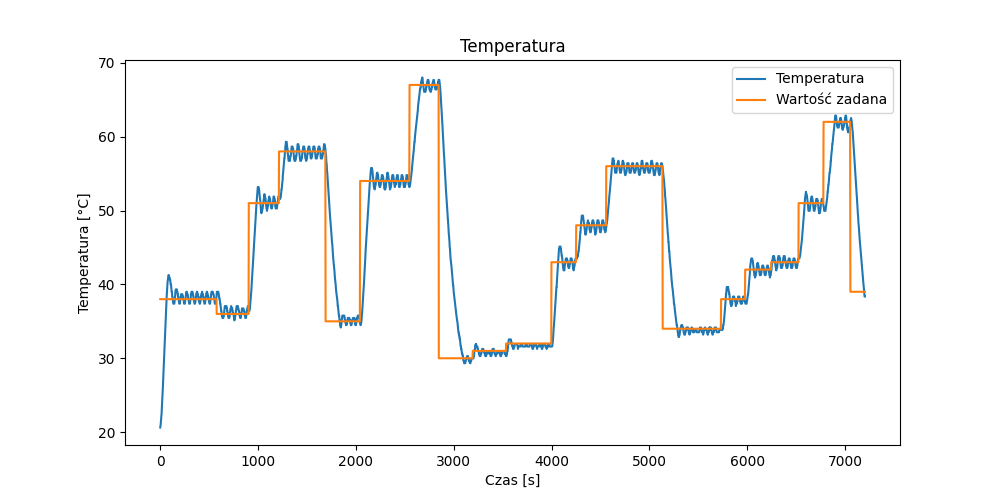
Rys. 17. Wykresy trajektorii dla TD3 uczonego online.

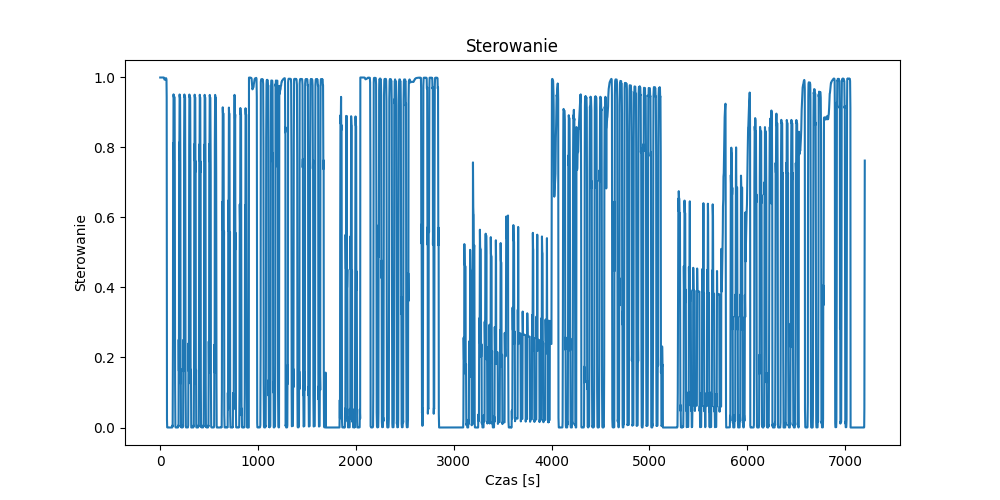
**SAC**

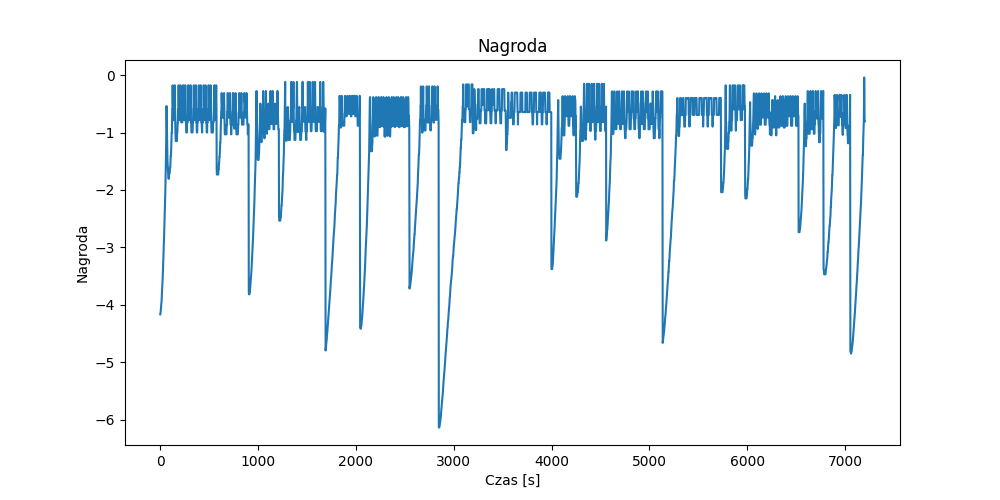


Rys. 18. Wykresy trajektorii dla SAC uczonego online.

**CQL-SAC**







Rys. 19. Wykresy trajektorii dla CQL-SAC uczonego online.

# 6. Testowanie i weryfikacja sprzętowa

W rozdziale 5. przedstawiono proces uczenia i testowania 4 algorytmów w oparciu o symulację systemu ogrzewania. Następnym krokiem jest przeprowadzenie weryfikacji i testów na rzeczywistym sprzęcie, co pozwoli pokazać praktyczną użyteczność przeprowadzania podobnych procesów.

# Bibliografia

TODO: Uporządkowanie bibliografii

TODO: Formatowanie kodu

1. Sutton, R. S., & Barto, A. G. (2018). "Reinforcement Learning: An Introduction." <http://incompleteideas.net/book/RLbook2020.pdf>
2. Berger-Tal O, Nathan J, Meron E, Saltz D (2014) The Exploration-Exploitation Dilemma: A Multidisciplinary Framework. PLoS ONE 9(4): e95693. <https://doi.org/10.1371/journal.pone.0095693>
3. Alexandre dos Santos Mignon, Ricardo Luis de Azevedo da Rocha, An Adaptive Implementation of ε-Greedy in Reinforcement Learning, Procedia Computer Science, Volume 109, 2017, Pages 1146-1151, ISSN 1877-0509, <https://doi.org/10.1016/j.procs.2017.05.431>. (<https://www.sciencedirect.com/science/article/pii/S1877050917311134>)
4. Kan WANG, Jinfeng WANG, Jing XIE, Xingxing MA. Improvement of Temperature Control Performance for Electric Heating Water Tank. Thermal Science [Internet]. 2024 Jan;28(1A):25–37.  
   <https://search-1ebscohost-1com-1000003nk00ab.wbg2.bg.agh.edu.pl/login.aspx?direct=true&db=asn&AN=176003236&lang=pl&site=eds-live>
5. Khodadadi H, Dehghani A. Fuzzy logic self-tuning PID controller design based on smith predictor for heating system. 2016 16th International Conference on Control, Automation and Systems (ICCAS), Control, Automation and Systems (ICCAS), 2016 16th International Conference on [Internet]. 2016 Oct 1;161–6.   
   <https://search-1ebscohost-1com-1000003nk00ab.wbg2.bg.agh.edu.pl/login.aspx?direct=true&db=edseee&AN=edseee.7832314&lang=pl&site=eds-live>
6. Chakrabarti A. , Sadhu P. , Chakraborty A. , & Pal P.. A fuzzy pid controller for induction heating systems with llc voltage source inverter. International Journal of Power Electronics and Drive Systems (IJPEDS) 2017;8(3):1168. <https://doi.org/10.11591/ijpeds.v8.i3.pp1168-1175>
7. Ahmed Sabah Al-Araji. Applying Cognitive Methodology in Designing On-Line Auto-Tuning Robust PID Controller for the Real Heating System. Journal of Engineering [Internet]. 2023 Jul 1;20(09). Available from: <https://search-1ebscohost-1com-1000003nk00ab.wbg2.bg.agh.edu.pl/login.aspx?direct=true&db=edsdoj&AN=edsdoj.917559b07303475a820781ef91148d4f&lang=pl&site=eds-live>
8. Chojecki A, Ambroziak A, Borkowski P. Fuzzy Controllers Instead of Classical PIDs in HVAC Equipment: Dusting off a Well-Known Technology and Today’s Implementation for Better Energy Efficiency and User Comfort. Energies (19961073) [Internet]. 2023 Apr;16(7):2967.   
   <https://search-1ebscohost-1com-1000003nk00e2.wbg2.bg.agh.edu.pl/login.aspx?direct=true&db=asn&AN=163043447&lang=pl&site=eds-live>
9. Ardabili S. , Beszedes B. , Nádai L. , Széll K. , Mosavi A. , & Felde I.. Comparative analysis of single and hybrid neuro-fuzzy-based models for an industrial heating ventilation and air conditioning control system. 2020 RIVF International Conference on Computing and Communication Technologies (RIVF) 2020. <https://doi.org/10.1109/rivf48685.2020.9140753>
10. Kleiminger W. , Santini S. , & Mattern F.. Smart heating control with occupancy prediction. Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication 2014. <https://doi.org/10.1145/2638728.2641555>
11. Thilker C. , Bergsteinsson H. , Bacher P. , Madsen H. , Calı̀ D. , & Junker R.. Non-linear model predictive control for smart heating of buildings. E3S Web of Conferences 2021;246:09005. <https://doi.org/10.1051/e3sconf/202124609005>
12. Makram Abdellatif, Julien Chamoin, Jean-Marie Nianga, Didier Defer, A thermal control methodology based on a machine learning forecasting model for indoor heating, Energy and Buildings, Volume 255, 2022, 111692, ISSN 0378-7788,  
    <https://doi.org/10.1016/j.enbuild.2021.111692>.
13. Minjae Shin, Sungsoo Kim, Youngjin Kim, Ahhyun Song, Yeeun Kim, Ha Young Kim, Development of an HVAC system control method using weather forecasting data with deep reinforcement learning algorithms, Building and Environment, Volume 248, 2024, 111069, ISSN 0360-1323,  
    <https://doi.org/10.1016/j.buildenv.2023.111069>.
14. Liu B. , Akçakaya M. , & McDermott T.. Automated control of transactive hvacs in energy distribution systems. IEEE Transactions on Smart Grid 2021;12(3):2462-2471. <https://doi.org/10.1109/tsg.2020.3042498>
15. Blad, C., Bøgh, S., & Kallesøe, C. S. (2021). A multi-agent reinforcement learning approach to price and comfort optimization in hvac-systems. Energies, 14(22), 7491. <https://doi.org/10.3390/en14227491>
16. Adjei P, Tasfi N, Gomez-Rosero S, Capretz MAM. Safe Reinforcement Learning for Arm Manipulation with Constrained Markov Decision Process. Robotics [Internet]. 2024 Apr 1;13(4).   
    <https://search-1ebscohost-1com-1000003w3012f.wbg2.bg.agh.edu.pl/login.aspx?direct=true&db=edselc&AN=edselc.2-52.0-85191607938&lang=pl&site=eds-live>
17. Bumsoo Park, Alexandra R. Rempel, Alan K.L. Lai, Julianna Chiaramonte, Sandipan Mishra, Reinforcement Learning for Control of Passive Heating and Cooling in Buildings⁎⁎This work was funded by National Science Foundation CBET-1804218., IFAC-PapersOnLine, Volume 54, Issue 20, 2021, Pages 907-912, ISSN2405-8963, <https://doi.org/10.1016/j.ifacol.2021.11.287>.
18. Watkins, C.J.C.H. (1989). Learning from Delayed Rewards (Ph.D. thesis). University of Cambridge. EThOS uk.bl.ethos.330022, <https://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf>
19. Watkins C. and Dayan P.. Q-learning. Machine Learning 1992;8(3-4):279-292. <https://doi.org/10.1007/bf00992698>
20. Wiering M, van Otterlo M. Reinforcement learning: State-of-the-art. Springer Science & Business Media; 2012.
21. Williams RJ. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning [Internet]. 1992 May 1;8(3–4):229–56. <https://search-1ebscohost-1com-1000003bj00bd.wbg2.bg.agh.edu.pl/login.aspx?direct=true&db=edssjs&AN=edssjs.BABE2B18&lang=pl&site=eds-live>
22. X. Wang et al., "Deep Reinforcement Learning: A Survey," in IEEE Transactions on Neural Networks and Learning Systems, vol. 35, no. 4, pp. 5064-5078, April 2024, doi: 10.1109/TNNLS.2022.3207346.
23. Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D. &amp; Riedmiller, M.. (2014). Deterministic Policy Gradient Algorithms. Proceedings of the 31st International Conference on Machine Learning, in Proceedings of Machine Learning Research 32(1):387-395   
    https://proceedings.mlr.press/v32/silver14.html.
24. Lillicrap, Timothy & Hunt, Jonathan & Pritzel, Alexander & Heess, Nicolas & Erez, Tom & Tassa, Yuval & Silver, David & Wierstra, Daan. (2015). Continuous control with deep reinforcement learning. CoRR. <https://doi.org/10.48550/arXiv.1509.02971>
25. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal Policy Optimization Algorithms. 2017;   
    <https://doi.org/10.48550/arXiv.1707.06347>
26. Scott Fujimoto, Herke van Hoof, and David Meger. (2018). Addressing Function Approximation Error in Actor-Critic Methods.  
    <https://doi.org/10.48550/arXiv.1802.09477>
27. Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In Proceedings of the 28th International Conference on machine learning (ICML-11), pages 465–472, 2011
28. Bansal S, Calandra R, Chua K, Levine S, Tomlin C. MBMF: Model-Based Priors for Model-Free Reinforcement Learning. 2017;   
    <https://doi.org/10.48550/arXiv.1709.03153>
29. Weiner A, Geise J. Model-based deep reinforcement learning for accelerated learning from flow simulations. Meccanica: An International Journal of Theoretical and Applied Mechanics AIMETA [Internet]. 2024 May 14;1–18. <https://doi.org/10.48550/arXiv.2402.16543>
30. Graciela Lopez-Muniz, Ana Cerezo-Hernández, Félix del Campo, Tomas Ruiz-Albi, Daniel Álvarez, Tania Alvaro-De Castro, et al. Usefulness of Artificial Neural Networks in the Diagnosis and Treatment of Sleep Apnea-Hypopnea Syndrome. 2017 Apr 5   
    <https://doi.org/10.5772/66570>
31. R. Tadeusiewicz i M. Szaleniec, Leksykon sieci neuronowych. Projekt Nauka.

Fundacja na rzecz promocji nauki polskiej.

1. Python Software Foundation. „General Python FAQ”. (29 maj 2024). <https://docs.python.org/3/faq/general.html#what-is-python>
2. Python Software Foundation and JetBrains „Python Developers Survey 2022 Results”, (29 maj 2024) <https://lp.jetbrains.com/python-developers-survey-2022/>
3. Google Brain Team „Introduction to TensorFlow” (29 maj 2024) <https://www.tensorflow.org/learn>
4. Optuna Contributors „Optuna: A hyperparameter optimization framework” (29 maj 2024) <https://optuna.readthedocs.io/en/stable/reference/samplers/index.html>
5. Jeffrey Kantor and Carl Sandrock „TCLab: Temperature Control Laboratory” (29 maj 2024) <https://tclab.readthedocs.io/en/latest/README.html>
6. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal Policy Optimization Algorithms. 2017; <https://doi.org/10.48550/arXiv.1502.05477>
7. Alsulaimani AA. The Efficiency of Artificial Recurrent Neural Network (RNN) in Predicting Academic Performance for Students. International Journal of Simulation -- Systems, Science & Technology. 2024 Mar; 25(1):2.1-2.6. <https://doi.org/10.5013/IJSSST.a.25.01.02>.
8. Colah’s Blog „Understanding LSTM Networks” (24.06.24) <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>
9. Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., ... & Levine, S. (2018). Soft actor-critic algorithms and applications. [*https://doi.org/10.48550/arXiv.1812.05905*](https://doi.org/10.48550/arXiv.1812.05905)
10. Kumar, A., Zhou, A., Tucker, G., & Levine, S. (2020). Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, *33*, 1179-1191. <https://doi.org/10.48550/arXiv.2006.04779>
11. K. J. Åström i T. Hägglund, „Revisiting the Ziegler–Nichols step response method for PID control”, J. Process Control, t. 14, nr 6, s. 635–650, wrz. 2004, doi:10.1016/j.jprocont.2004.01.002.
12. Google Brain Team “Tensorflow Agents tf-agents overiew” (10 lipiec 2024) https://www.tensorflow.org/agents/api\_docs/python/tf\_agents