

# Optymalizacja Agentów RL w Środowiskach Gier

## Teoria i Praktyka: DQN, A2C, PPO

Jan Zakroczymski, Jan Zadrażg, Sławek Brzózka

January 15, 2026

# Plan Prezentacji

- 1 Definicja Problemu
- 2 Dane i Przygotowanie
- 3 Analiza Danych (EDA)
- 4 Wybór Modelu
- 5 Podstawy Teoretyczne
- 6 Opisy Rozważanych Modeli
  - DQN (Deep Q-Network)
  - A2C (Advantage Actor-Critic)
  - PPO (Proximal Policy Optimization)
- 7 Kluczowe Hiperparametry
- 8 Trening i Ewaluacja
- 9 Analiza Błędów
- 10 Udoskonalenia
- 11 Porównanie i Wyniki
  - Optymalizacja vs bez
- 12 Podsumowanie

# 1. Definicja Problemu i Cel

## Cel Projektu:

Stworzenie i optymalizacja agentów uczenia ze wzmacnieniem (ang. RL) dla środowisk Gymnasium (LunarLander, CarRacing, CartPole).

## Wyzwanie:

Znalezienie balansu między stabilnością a szybkością uczenia poprzez:

- Dobór odpowiedniego algorytmu (Model Selection).
- Automatyczną optymalizację hiperparametrów (Optuna).

## 2. Dane w RL i Źródła

- **Źródło danych:** interakcja agenta ze środowiskiem Gymnasium (symulator).
- **Dane są generowane online:** kolejne stany powstają w czasie treningu.
- **Różne przestrzenie stanów:** wektor stanu (CartPole, LunarLander) vs obraz RGB 96x96 (CarRacing).
- **Powtarzalność:** ustalony seed dla treningu, ewaluacji oraz optymalizacji hiperparametrów.

### 3. Przygotowanie Danych

- **Preprocessing CarRacing:** grayscale + resize do 84x84 + frame stacking (4 klatki).
- **Normalizacja obrazu:** model normalizuje piksele na wejściu.
- **Dostosowanie akcji:** dyskretyzacja sterowania tylko dla DQN w CarRacing.
- **Wektoryzacja środowisk:** DQN(16 env), A2C (16 env), PPO (16 env).

## 4. EDA w RL

W RL nie ma klas, a dane powstają w trakcie interakcji. EDA oznacza analizę:

- **Trajektorii:** sekwencji stanów.
- **Rozkładu nagród w czasie (episodic return):** wykresach sumy nagród względem kroków symulacji.
- **Stabilność względem długości epizodów:** wykrywanie oscylacji nagród.
- **Efektywność względem czasu treningu.**
- **Epsilon względem czasu (w przypadku DQN).**

# Uzasadnienie Wyboru Modelu

- **DQN:** najlepszy dla dyskretnych akcji i prostszych przestrzeni stanów.
- **A2C:** prosty algorytm on-policy obsługujący akcje ciągłe.
- **PPO:** zawaansowana wersja A2C poprawiająca stabilność algorytmu.

# Podstawy Teoretyczne RL

**RL opiera się na założeniu że rozpatrywany proces jest procesem markowa. Przyjęty model środowiska:** Agent w stanie  $s_t$  podejmuje akcję  $a_t$ , otrzymuje nagrodę  $r_t$  i przechodzi do stanu  $s_{t+1}$ .

**Cel Agenta:** Maksymalizacja sumy zdyskontowanych nagród:

$$G_t = \sum_{k=1}^{\infty} \gamma^k r_{t+k}$$

gdzie  $\gamma \in [0, 1]$  to współczynnik dyskontujący (discount factor).

# Podstawy Teoretyczne RL - Pojęcia

- **Polityka  $\pi(s)$  lub  $\pi(a|s)$ :** wybrana akcja lub rozkład akcji w danym stanie.
- **Funkcja wartości  $V_\pi(s)$ :** oczekiwany zwrot z danego stanu przy polityce  $\pi$ .
- **Funkcja jakości  $Q_\pi(s, a)$ :** oczekiwany zwrot po wykonaniu akcji  $a$  w stanie  $s$  przy polityce  $\pi$ .
- **Eksploracja vs eksplotacja:** wybór między akcjami o najwyższej ocenie a nowymi.
- **On-policy vs off-policy:** czy dane do uczenia pochodzą z tylko aktualnej polityki.
- **Doświadczenie:** czwórka  $(s, a, r, s_{next})$  gdzie  $s$  - stan obecny,  $a$  - akcja,  $r$  - nagroda ze środowiska,  $s_{next}$  - następny stan.

# Deep Q-Network (DQN)

- Deep Q-Network (DQN), nazywany też Deep Q-Learning, to ulepszona wersja Q-Learning.
- Zamiast tabeli wartości używa sieci neuronowej, dzięki czemu działa w dużych przestrzeniach stanów.
- Nie rozwiązuje problemu ciągłych przestrzeni akcji, ponieważ sieć neuronowa ma skończoną liczbę wyjść.
- Mimo ograniczeń jest skuteczny i pozwala grać nawet w skomplikowane gry 3D poprzez dyskretyzacje akcji.

# DQN - Funkcja straty

Funkcja straty (loss):

$$loss = (Q_{target} - Q_{current})$$

gdzie:

- $Q_{current}$  to wartość  $Q$  zwracana przez model dla stanu  $s$ .
- $Q_{target} = nagroda + \gamma \max_a Q(s_{next}, a)$ .
- nagroda - zwrócona nagroda przez środowisko.

# DQN - Kluczowe mechanizmy

- **Target Network:** osobna sieć do obliczania  $Q_{target}$  co stabilizuje uczenie.
- **Replay Buffer:** przechowuje doświadczenia i umożliwia losowe próbkowanie, co zmniejsza korelacje akcji.
- **Eksploracja  $\epsilon$ -greedy:**  $\epsilon$  zmniejszający się w czasie współczynnik akcji losowych - kompromis pomiędzy eksploracją a eksploatacją.

## Advantage Actor-Critic (A2C)

- Składa się z dwóch sieci: **Aktor** steruje zachowaniem, **Krytyk** ocenia akcję.
- Podejście hybrydowe: łączy zalety on-policy (actor) i off-policy (critic).
- Rozwija algorytm strategii gradientowej.

## A2C - Aktualizacja krytyka

Krytyk uczy się funkcji wartości  $V(s)$  w której wagi są aktualizowane za pomocą wzoru:

$$\Delta w = \beta(r + \gamma V(s_{next}) - V(s))\nabla_w V_w(s)$$

## A2C - Aktualizacja aktora

Funkcja przewagi:

$$A(s, s_{next}) = r + \gamma V(s_{next}) - V(s)$$

Aktualizacja aktora:

$$\Delta\theta = \alpha[\nabla(\log \pi(s, a, \theta)) \cdot A(s, s_{next}) + c \nabla S_\pi(s)]$$

Jest to forma analogiczna do strategii gradientowej.

# Proximal Policy Optimization (PPO)

- PPO to state of the art w RL, użyty m.in. do przełomu w Dota 2 (2018).
- Architektura podobna do A2C: aktor i krytyk.
- Główna idea: unikanie zbyt dużych aktualizacji aktora.

## PPO - Funkcja zmiany

Stosunek prawdopodobieństw:

$$r_t(\theta) = \frac{\pi_\theta(a_t | s_t)}{\pi_{\theta_{old}}(a_t | s_t)}$$

Cel: utrzymać  $r_t(\theta)$  w przedziale  $[1 - \epsilon, 1 + \epsilon]$ .

## PPO - Funkcja clip

$$clip(r_t(\theta), \epsilon) = \begin{cases} 1 - \epsilon & \text{gdy } r_t(\theta) < 1 - \epsilon \\ r_t(\theta) & \text{gdy } r_t(\theta) \in (1 - \epsilon, 1 + \epsilon) \\ 1 + \epsilon & \text{gdy } r_t(\theta) > 1 + \epsilon \end{cases}$$

# PPO - Funkcje loss

**Loss polityki:**

$$L_{clip} = \mathbb{E}_t[\min(r_t(\theta) \cdot A_t, clip(r_t(\theta), \epsilon) \cdot A_t)]$$

Wartość powyższej funkcji jest bardzo mocno zależna od  $A_t(s_t, s_{t+1})$ .

**Loss krytyka:**

$$L_V = r_t + \gamma V(s_{t+1}) - V(s_t)$$

**Loss całkowity:**

$$Loss = \mathbb{E}_t[L_V - c_1 L_{clip} + c_2 S_\pi(s_t)]$$

## Kluczowe Hiperparametry - Wspólne

Wspólne parametry optymalizowane dla wszystkich modeli (Optuna):

- **Learning Rate (LR):** Szybkość uczenia ( $10^{-5}$  do  $10^{-2}$ ). Zbyt duży destabilizuje, zbyt mały spowalnia.
- **Gamma ( $\gamma$ ):** Czynnik dyskontujący (0.9 do 0.9999). Określa, jak ważne są przyszłe nagrody (krótkowzroczność vs dalekowzroczność).
- **Batch Size:** Ilość próbek w jednej aktualizacji gradientu (16 do 512).
- **Architektura Sieci:** Rozmiar warstw ukrytych (Tiny, Small, Medium).

# Kluczowe Hiperparametry - Specyficzne

## DQN (Deep Q-Network):

- **Target Update Interval:** Częstotliwość aktualizacji sieci docelowej (1000-20000 kroków). Kluczowe dla stabilności.
- **Exploration Fraction:** Jak długo zmniejszać epsilon (eksplorację).

## PPO / A2C (Actor-Critic):

- **Entropy Coefficient:** Premia za entropię. Zapobiega przedwczesnej zbieżności do suboptimalnej polityki.
- **GAE Lambda ( $\lambda$ ):** Balans między bias a wariancją w estymacji przewagi.
- **N Steps:** Długość trajektorii zbieranej przed aktualizacją.

# Optymalizacja w Praktyce - LunarLander (Case Study)

Obecnie Optuna optymalizuje na środowisku test (niezależne seedy), co redukuje przeuczenie do jednej ewaluacji.

- **Cel optymalizacji:** test/mean\_reward.
- **Budżet:** domyślnie 60 prób, 300k kroków na próbę.
- **Różne algorytmy:** osobne przestrzenie hiperparametrów dla każdego algorytmu.

# Trening Modeli

- **Budżet treningu:** konfigurowalny w config.json (aktualnie 1 000 000 kroków).
- **Podział środowisk:** trening seed=0, ewaluacja seed=42, test seed=1000.
- **Optymalizacja:** Optuna (domyślnie 40 prób, 150k kroków na próbę, inne dla każdej gry).

# Ewaluacja Modeli

- **Mean Reward:** główna metryka skuteczności (eval/mean\_reward, test/mean\_reward).
- **Episodic Length:** dodatkowa kontrola stabilności.
- **Ewaluacja deterministyczna:** stałe warunki porównań, 10 epizodów na ewaluację.

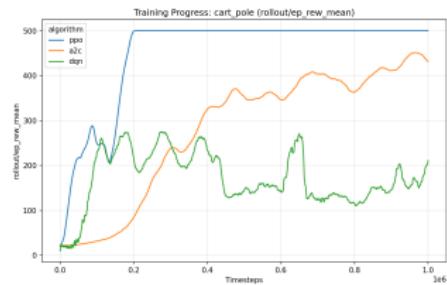
# Analiza Błędów i Ograniczeń

- **Plateau nagrody:** widoczne wypłaszczenia na wykresach reward.
- **Katastrofalne zapominanie:** spadki po długich treningach.
- **CarRacing + DQN:** utrata płynności sterowania przy dyskretnych akcjach.
- **Diagnostyka:** porównanie reward/loss/epsilon między algorytmami.

# Optymalizacja i Ulepszenia

- **Preprocessing CarRacing:** grayscale + resize + frame stacking (4).
- **Dyskretyzacja akcji:** tylko dla DQN w CarRacing.
- **DQN buffer size:** redukcja do 50k dla danych obrazowych.
- **Tuning Optuna:** LR  $10^{-5}$ – $10^{-2}$ ,  $\gamma$  0.9–0.9999, batch 16–512, itd.

# CartPole



(a) Bez optymalizacji



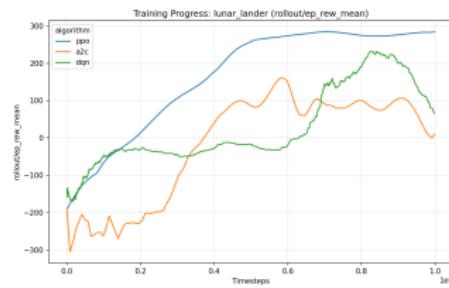
(b) Po optymalizacji

Figure: CartPole: porównanie średniej nagrody w epizodzie.

# LunarLander



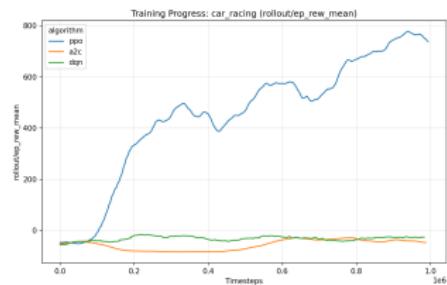
(a) Bez optymalizacji



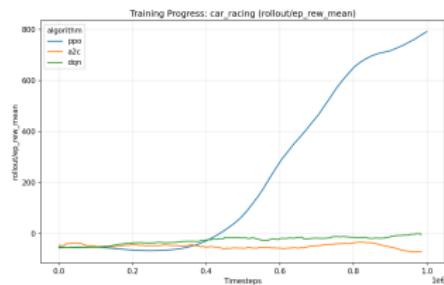
(b) Po optymalizacji

**Figure:** LunarLander: porównanie średniej nagrody w epizodzie.

# CarRacing



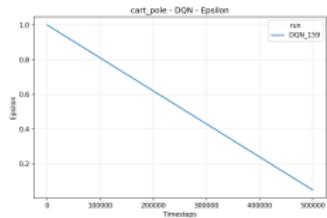
(a) Bez optymalizacji



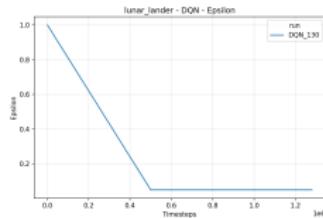
(b) Po optymalizacji

Figure: CarRacing: porównanie średniej nagrody w epizodzie.

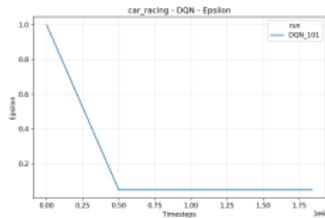
# DQN - Epsilon Decay (Porownanie)



(a) CartPole



(b) LunarLander



(c) CarRacing

# Optuna - DQN: Najlepsze parametry

Środowisko	Parametry
CartPole	learning_rate=0.0001793261, gamma=0.9534737990, net_arch=medium, batch_size=16, optimizer_class=RMSprop, activation_fn=Tanh, weight_decay=0.0000216215, target_update_interval=1000, train_freq=16, gradient_steps=2, exploration_fraction=0.4929325775, exploration_final_eps=0.0154726366
LunarLander	learning_rate=0.0030895318, gamma=0.9942452945, net_arch=tiny, batch_size=256, optimizer_class=Adam, activation_fn=ReLU, weight_decay=0.0000029136, target_update_interval=5000, train_freq=16, gradient_steps=8, exploration_fraction=0.2565756418, exploration_final_eps=0.0214548086
CarRacing	learning_rate=0.0000190746, gamma=0.9964980011, net_arch=small, batch_size=256, optimizer_class=AdamW, activation_fn=Tanh, weight_decay=0.0001018457, target_update_interval=5000, train_freq=16, gradient_steps=4, exploration_fraction=0.4662493899, exploration_final_eps=0.0880439823

# Optuna - A2C: Najlepsze parametry

Środowisko	Parametry
CartPole	learning_rate=0.0004946017, gamma=0.9554415436, net_arch=medium, optimizer_class=RMSprop, activation_fn=Tanh, weight_decay=0.0000534000, ent_coef=0.0000226172, vf_coef=0.6432393827, max_grad_norm=0.9263505727, gae_lambda=0.9680905634, n_steps=1024
LunarLander	learning_rate=0.0051225863, gamma=0.9997455014, net_arch=medium, optimizer_class=RMSprop, activation_fn=Tanh, weight_decay=0.0000000267, ent_coef=0.0000950109, vf_coef=0.6683089411, max_grad_norm=0.7851667075, gae_lambda=0.9361844125, n_steps=512
CarRacing	learning_rate=0.0001833084, gamma=0.9988692600, net_arch=small, optimizer_class=AdamW, activation_fn=Tanh, weight_decay=0.0007935114, ent_coef=0.0001100083, vf_coef=0.7279991641, max_grad_norm=0.3641113129, gae_lambda=0.9781574680, n_steps=50

# Optuna - PPO: Najlepsze parametry

Środowisko	Parametry
CartPole	learning_rate=0.0011687224, gamma=0.9640504137, net_arch=medium, batch_size=512, optimizer_class=RMSprop, activation_fn=ReLU, weight_decay=0.0000045619, ent_coef=0.0000000143, vf_coef=0.2781731853, max_grad_norm=2.6276624082, gae_lambda=0.8823810760, n_steps=256
LunarLander	learning_rate=0.0006263061, gamma=0.9897638689, net_arch=tiny, batch_size=64, optimizer_class=AdamW, activation_fn=ReLU, weight_decay=0.0000000287, ent_coef=0.0000514425, vf_coef=0.7284058948, max_grad_norm=0.4699947472, gae_lambda=0.9753815924, n_steps=1024, clip_range=0.1958369398
CarRacing	learning_rate=0.0002487322, gamma=0.9862992164, net_arch=tiny, batch_size=64, optimizer_class=RMSprop, activation_fn=ReLU, weight_decay=0.0000000505, ent_coef=0.0000595936, vf_coef=0.6120567114, max_grad_norm=0.8796216903, gae_lambda=0.9709323978, n_steps=1024, clip_range=0.1825195801

# Wyniki Optymalizacji z Optuna

Wyniki zależą od środowiska i długości triali. Aktualnie optymalizacja:

- wybiera parametry na podstawie **test mean reward**,
- używa 100k kroków na próbę w CLI (batch: env-specific),
- ogranicza ryzyko przeuczenia do jednego seeda.

# Porównanie Algorytmów w Projekcie

Cecha	DQN	A2C	PPO
Typ	Off-policy	On-policy	On-policy
Akcje	Dyskretne	Dyskretne/Ciągłe	Dyskretne/Ciągłe
Stabilność	Średnia	Średnia	<b>Wysoka</b>
Sample Eff.	<b>Wysoka</b> (Replay)	Średnia	Niska

## Wnioski Końcowe

- **Stabilność:** PPO jest zwykle stabilniejsze niż A2C, ale wymaga sensownego budżetu kroków.
- **Obrazy:** DQN w środowiskach obrazowych wymaga preprocessing (grayscale, resize, frame stack).
- **Automatyzacja:** Optuna przyspiesza tuning, ale kosztuje czas i wymaga kontroli hold-out.