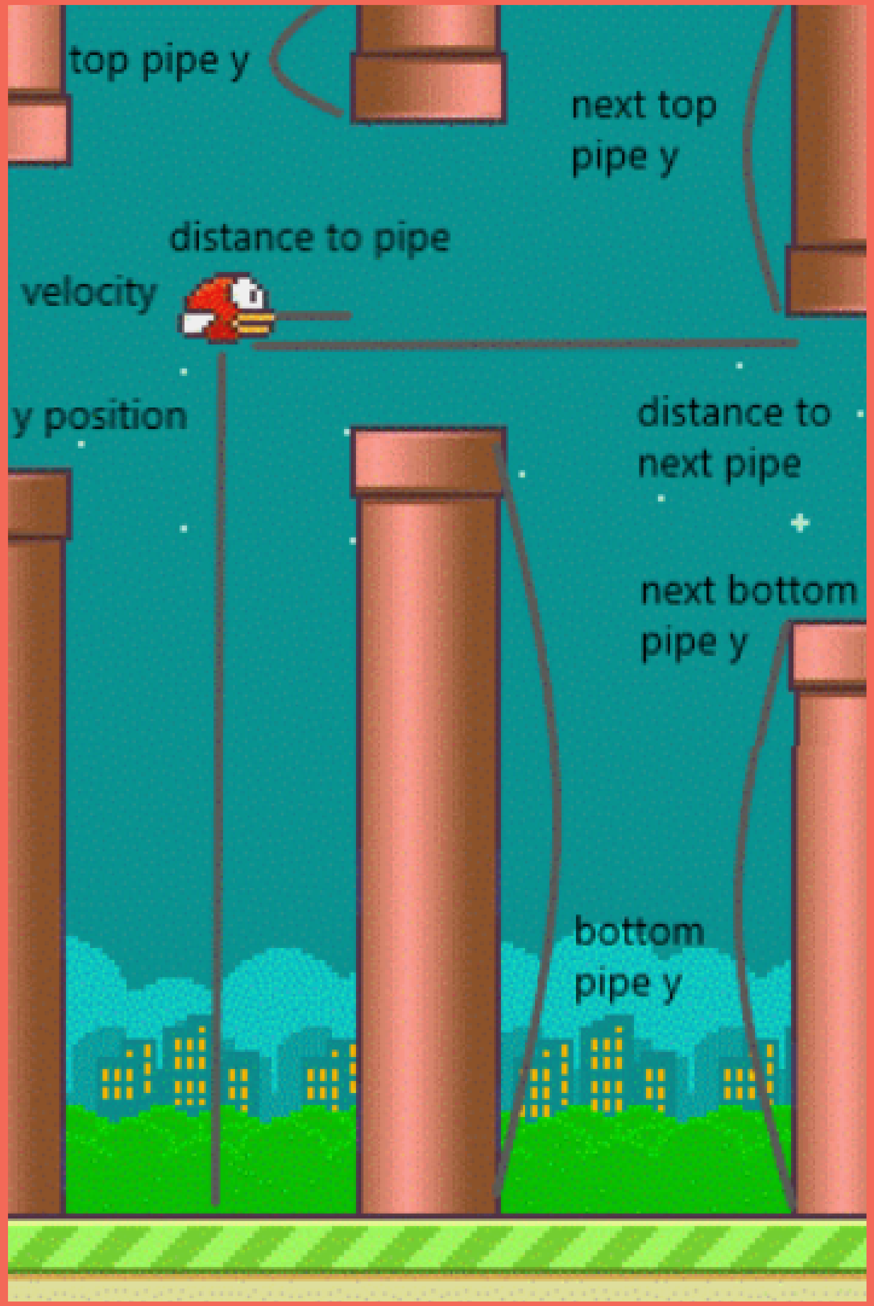
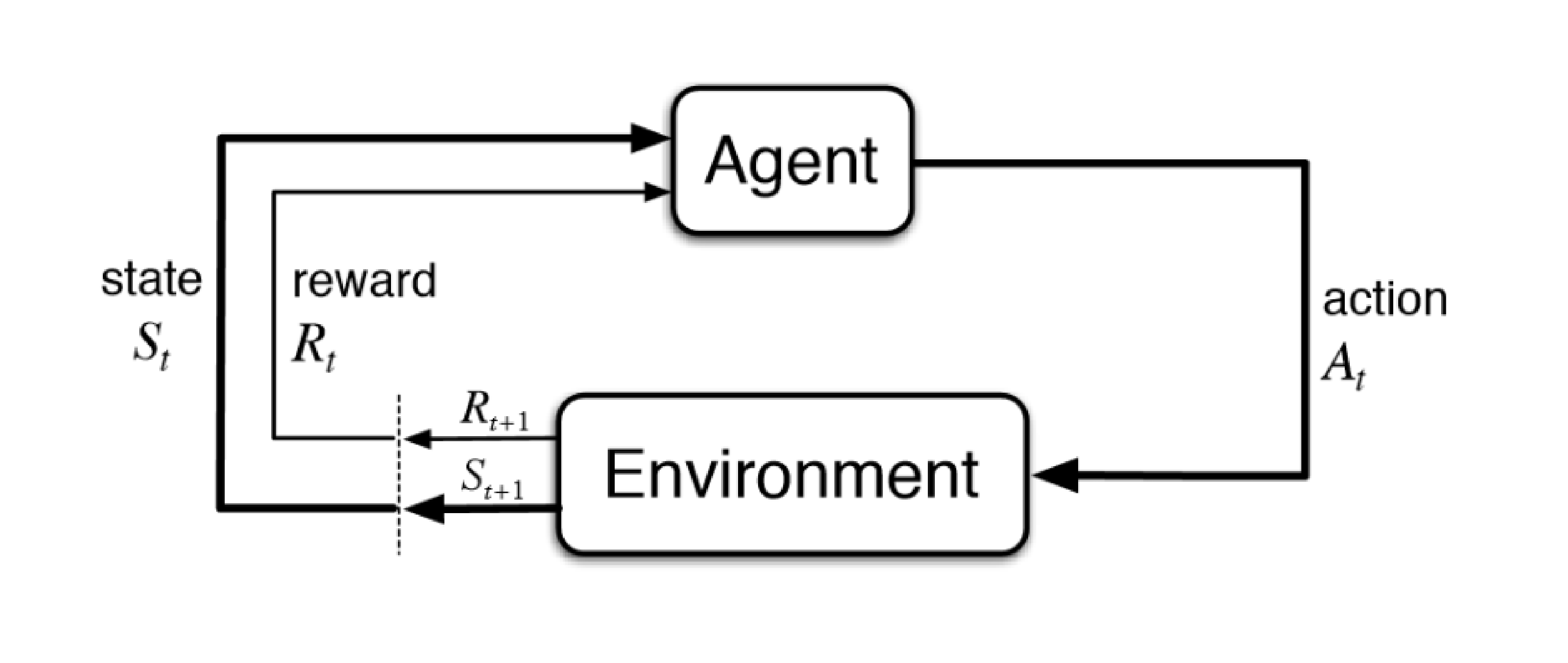
PIOTR PAWŁOWSKI

Deep Q-Learning in FlappyBird

Agenda

Omówienie środowiska FlappyBird
Deep Q-Learning - teoria oraz kod
Analiza hiperparametrów
Prezentacja najlepszego modelu





źródło: https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292

GRA

Długość rozgrywki determinują decyzje agenta - brak kryterium sukcesu

Kryteria porażki

- udzerzenie w dowolną z przeszkód
- upadek na ziemię
- wylot poza ekran

AKCJE

W każdym kroku agent ma do dyspozycji 2 akcje - **machnąć / nie machać** skrzydłami

NAGRODY

- +1 za każdą ominiętą przeszkodę
- -5 za przegraną rozgrywkę
- +x za każdy przeżyty krok*

STAN ŚRODOWISKA

Reprezentowany przez 8 wymiarowy wektor

Benchmark

Flappy Bird io Gameplay By The Numbers

So far, 47,127,433 games have been played on flappybird.io. That's not including some missed tracking days so I'm guessing it's around 50 million at this point. That blows my mind. I hope you have all enjoyed playing.

Here are some quick additional stats:

1. 95% of all games score 6 points or lower. 27% score 0, 62% score 0 or 1.

2. The highest score ever is 999,999,999,999,999, this is certainly a fraudulent score. I don't even think enough time has elapsed since the the game was posted for this to even be possible. It looks like a handful of people have legitimately scored from 100-200 points, but it's hard to tell if these don't involve some kind of code manipulation.

3. If you happen to score 10 points or higher you're doing better than 97.97% of all games. Keep clicking!

4. Here's a quick graph that show's the number of games played for scores 1-18: http://flappybird.io/graph.html

#analytics #information #scoring

Feb 22nd, 2014
9 notes

ŹRÓDŁO: HTTPS://BLOG.FLAPPYBIRD.IO/

Możesz sprawdzić swoje siły na http://flappybird.io/

Występujące trudności

STOCHASTYCZNOŚĆ*

Losowa generacja przeszkód (odległość stała, równa 100px)

SEKWENCYJNOŚĆ

Informacja o błędzie w *t* spropagowana dopiero w *t'*

PRZESTRZEŃ STANÓW

Nieskończona przestrzeń stanów uniemożliwia zastosowanie np. Q-Learning'u

FUNKCJA CELU/STRATY

Nieokreślona, nieróżniczkowalna funkcja straty; Wewnętrzny stan gry nie jest znany agentowi**

Human-level control through deep reinforcement learning

Volodymyr Mnih[1]*, Koray Kavukcuoglu[1]*, David Silver[1]*, Andrei A. Rusu[1], Joel Veness[1], Marc G. Bellemare[1], Alex Graves[1], Martin Riedmiller[1], Andreas K. Fidjeland[1], Georg Ostrovski[1], Stig Petersen[1], Charles Beattie[1], Amir Sadik[1], Ioannis Antonoglou[1], Helen King[1], Dharshan Kumaran[1], Daan Wierstra[1], Shane Legg[1] & Demis Hassabis[1]

The theory of reinforcement learning provides a normative account[1], deeply rooted in psychological[2] and neuroscientific[3] perspectives on animal behaviour, of how agents may optimize their control of an environment. To use reinforcement learning successfully in situations approaching real-world complexity, however, agents are confronted with a difficult task: they must derive efficient representations of the environment from high-dimensional sensory inputs, and use these to generalize past experience to new situations. Remarkably, humans and other animals seem to solve this problem through a harmonious combination of reinforcement learning and hierarchical sensory processing systems[4,5], the former evidenced by a wealth of neural data revealing notable parallels between the phasic signals emitted by dopaminergic neurons and temporal difference reinforcement learning algorithms[3]. While reinforcement learning agents have achieved some successes in a variety of domains[6–8], their applicability has previously been limited to domains in which useful features can be handcrafted, or to domains with fully observed, low-dimensional state spaces.

agent is to select actions in a fashion that maximizes cumulative future reward. More formally, we use a deep convolutional neural network to approximate the optimal action-value function

$$Q^*(s,a) = \max_\pi \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots | s_t = s, a_t = a, \pi\right],$$

which is the maximum sum of rewards r_t discounted by γ at each time-step t, achievable by a behaviour policy $\pi = P(a|s)$, after making an observation (s) and taking an action (a) (see Methods)[19].

Reinforcement learning is known to be unstable or even to diverge when a nonlinear function approximator such as a neural network is used to represent the action-value (also known as Q) function[20]. This instability has several causes: the correlations present in the sequence of observations, the fact that small updates to Q may significantly change the policy and therefore change the data distribution, and the correlations between the action-values (Q) and the target values $r + \gamma \max_{a'} Q(s', a')$. We address these instabilities with a novel variant of Q-learning, which uses two key ideas. First, we used a biologically inspired mechanism

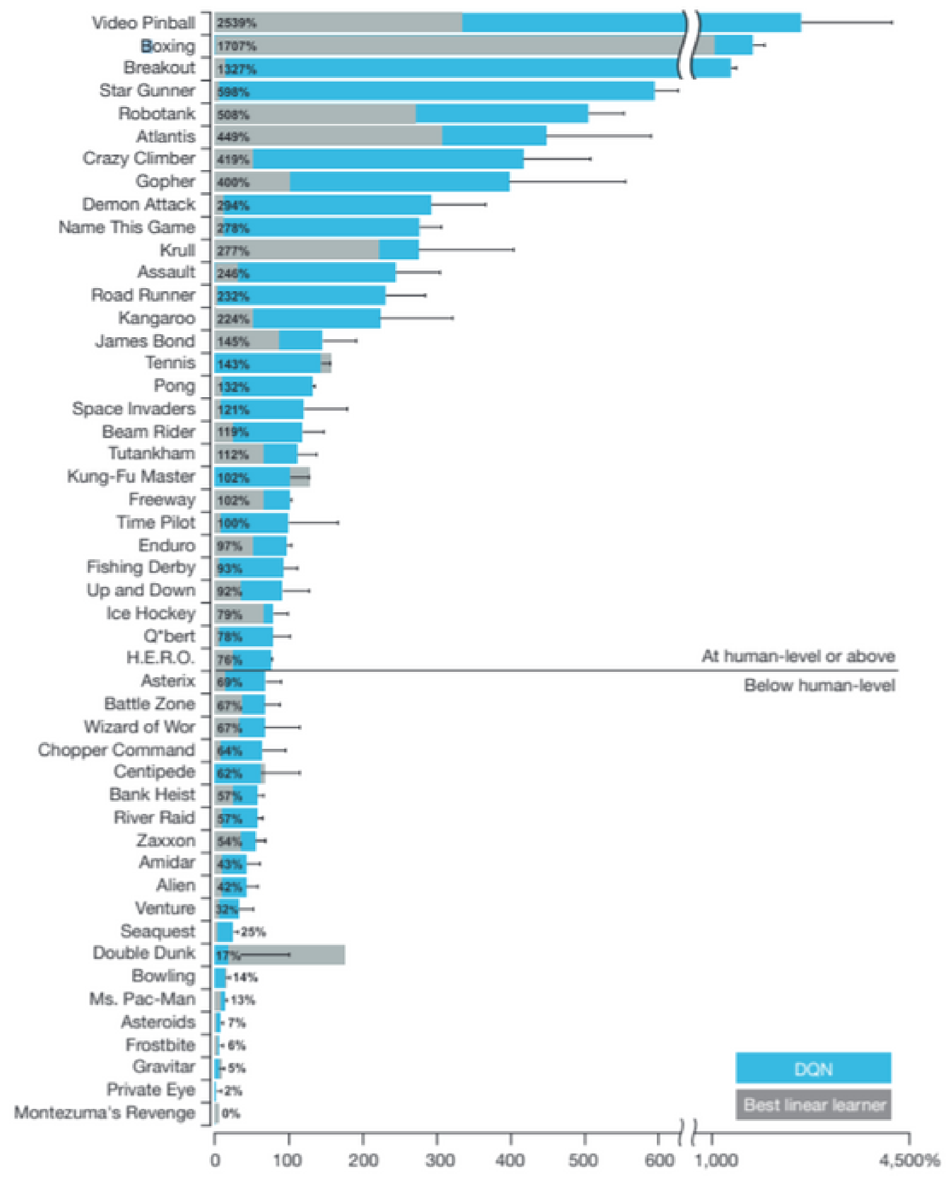


Figure 3 | Comparison of the DQN agent with the best reinforcement learning methods[15] in the literature. The performance of DQN is normalized

outperforms competing methods (also see Extended Data Table 2) in almost all the games, and performs at a level that is broadly comparable with or superior

Deep Q-Learning

WYKORZYSTANIE SIECI NEURONOWYCH W RL

DeepMind, 2015

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \mid s_t = s,\ a_t = a,\ \pi\right]$$

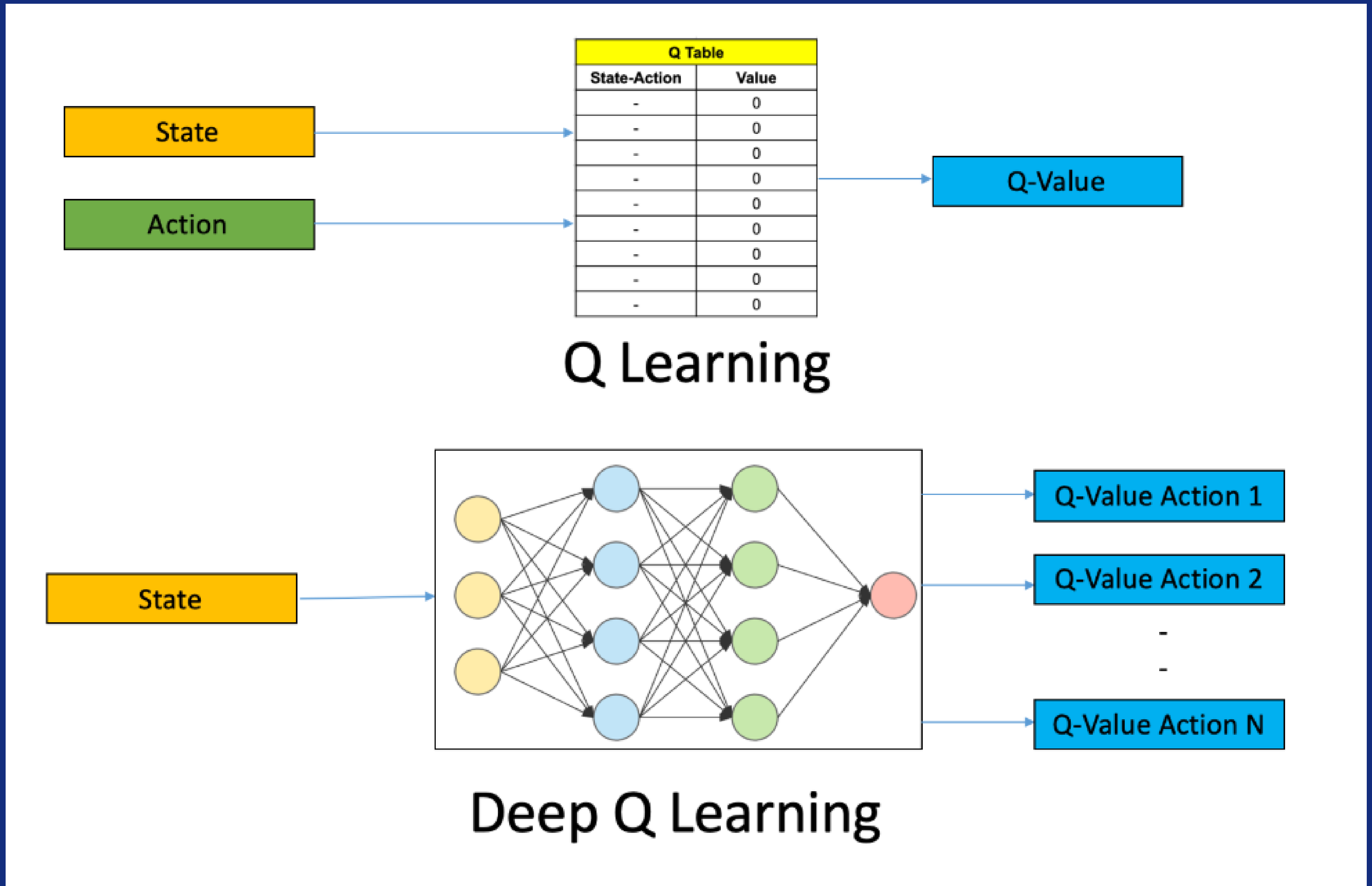
$$Q^*(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^*(s',a') \mid s,a\right]$$

źródło: https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf

$$Loss = (r + \gamma max_{a'} Q(s', a'; \theta') - Q(s, a; \theta))^2$$

źródło: https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/

DQL vs. QL

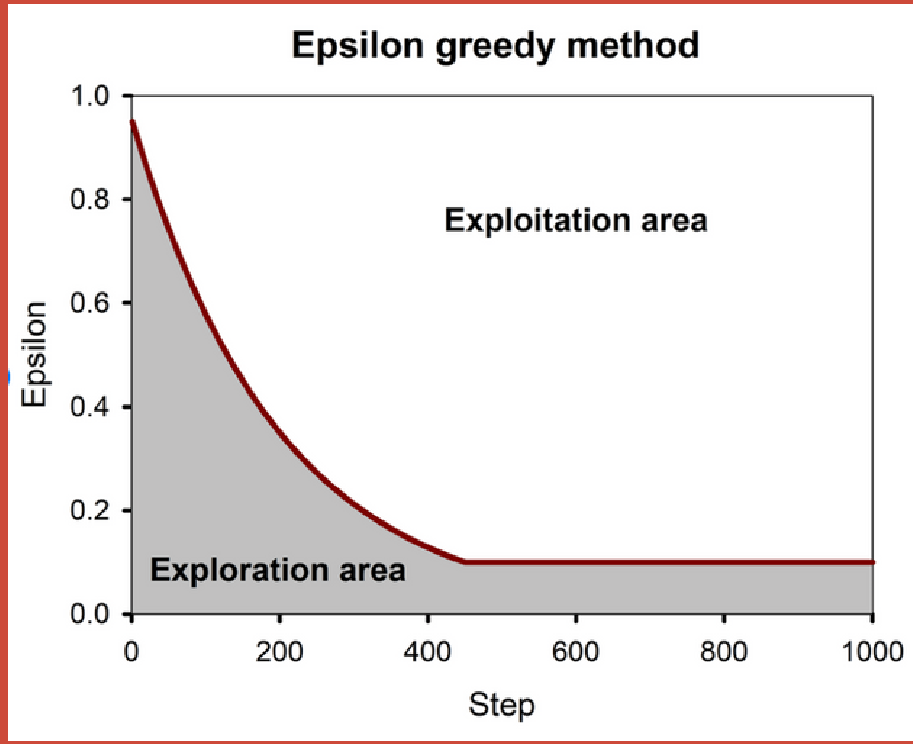


źródło: https://blogs.oracle.com/datascience/reinforcement-learning-deep-q-networks

Zastosowane techniki

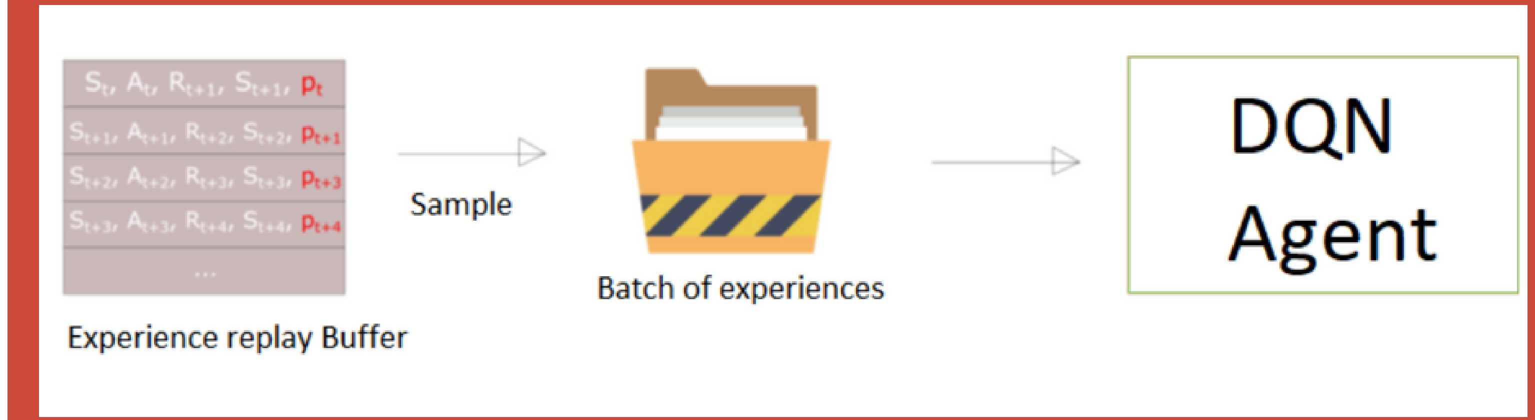
W PROCESIE TRENINGU

EPSILON GREEDY



źródło: https://www.researchgate.net/figure/Epsilon-greedy-method-At-each-step-a-random-number-is-generated-by-the-model-If-the_fig2_334741451

EXPERIENCE REPLAY



źródło: https://medium.com/analytics-vidhya/reinforcement-learning-d3qn-agent-with-prioritized-experience-replay-memory-6d79653e8561

TARGET NETWORK

Przebieg Algorytmu

```python
if random.random() < self._eps["epsilon"]:
    action = random.randrange(self._num_actions)
else:
    state = tf.expand_dims(state, axis=0)
    action = tf.argmax(self._net(state),
                       axis=-1)[0].numpy()

return action
```

```python
if max(self._min_memory, batch_size) > len(self._memory):
    return

batch = self._memory.sample(batch_size)

states = tf.stack(batch["state"], axis=0)
next_states = tf.stack(batch["next_state"], axis=0)
actions = tf.one_hot(indices=batch["action"],
                     depth=self._num_actions)

q_values_next = tf.reduce_max(self._net(next_states), axis=-1)
q_target = (tf.convert_to_tensor(batch["reward"]) +
            self._discount_factor *
            q_values_next *
            (1 - tf.convert_to_tensor(batch["done"])))

with tf.GradientTape() as tape:
    q_values = self._net(states)
    prediction = tf.reduce_sum(q_values * actions, axis=1)
    loss = mean_squared_error(q_target, prediction)
weights = self._net.trainable_variables
gradients = tape.gradient(loss, weights)
self._optimizer.apply_gradients(zip(gradients, weights))

if self._eps["eps_minimum"] < self._eps["epsilon"]:
    self._eps["epsilon"] *= self._eps["eps_decrement"]
```

```python
env.init()
for ep in range(1, FLAGS.episodes + 1):
    if ep % 20 == 0:
        best_score = np.max(scores)
        best_ep = np.argmax(scores) + 1
        mean_score = np.mean(scores)
        logging.info(f"Current episode: {ep}")
        logging.info(f"Best score: {best_score:.2f}, from episode no. {best_ep}")
        logging.info(f"Mean score: {mean_score:.2f}")

    episode_score = 0
    pipes_passed = 0
    done = False
    env.reset_game()
    state = list(env.getGameState().values())
    while not done:

        action = agent.predict(state)
        reward = env.act(ACTION_MAP[action])
        next_state = list(env.getGameState().values())
        done = env.game_over()

        if not done:
            if reward == 1:
                pipes_passed += 1
            reward += FLAGS.survived_step_reward

        episode_score += reward

        if FLAGS.train:
            agent.extend_memory(state, action, reward, next_state, done)
            agent.fit(FLAGS.batch_size)
        else:
            sleep(0.01)

        state = next_state

    scores.append(episode_score)
    pipes.append(pipes_passed)

logging.info("Done playing!")
```

Analiza hiperparametrów

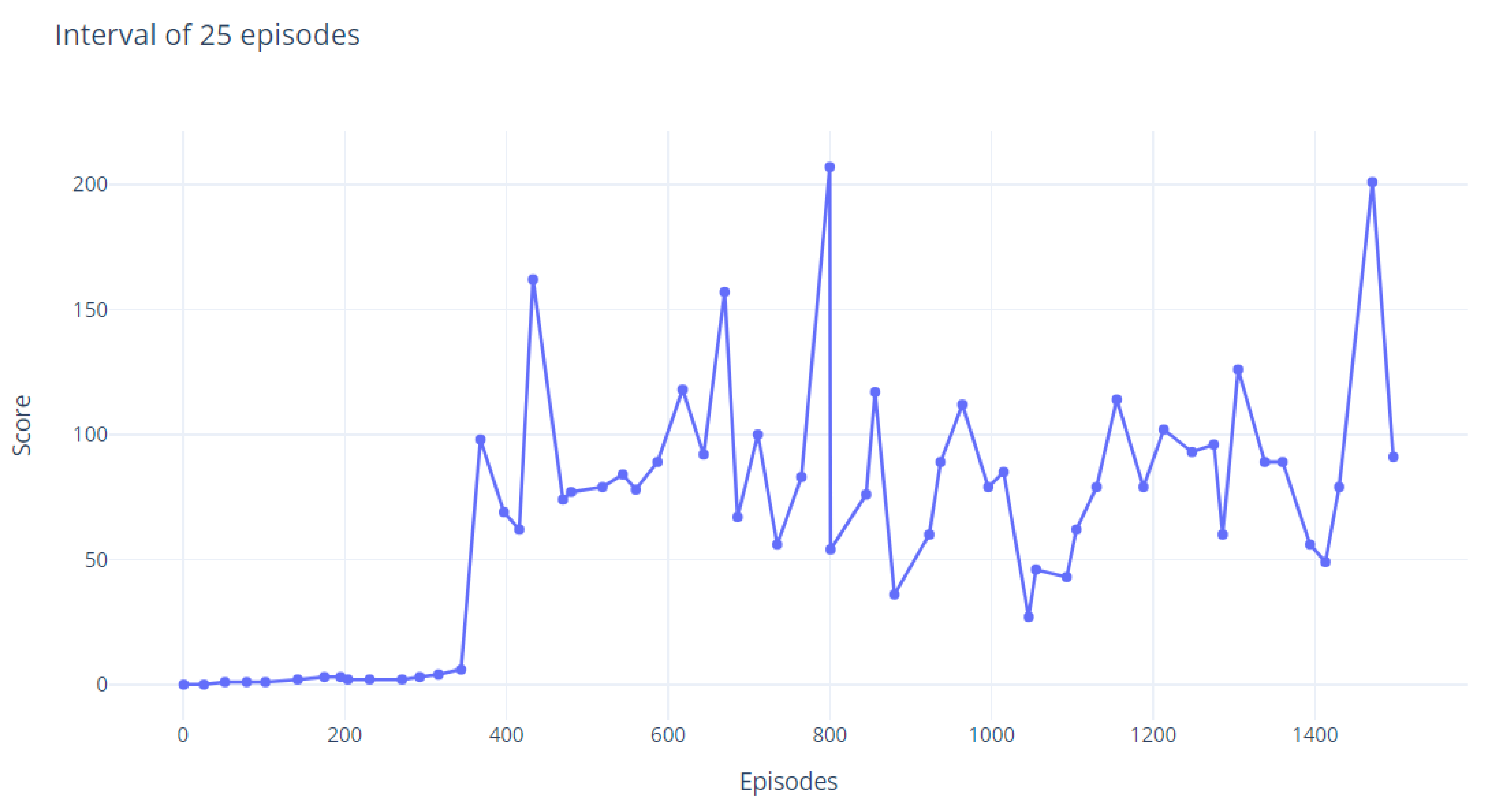
- mlp_hidden_units - liczba neuronów w każdej z warstw ukrytych wielowarstwowego perceptronu (256, 256)
- learning_rate - wielkość kroku w kierunku minimum (5e-05)
- discount_factor - czynnik determinujący w jakim stopniu agent bierze po uwagę nagrody z następnego kroku, cz
- memory_size - maksymalna liczba historycznych kroków zachowanych w pamięci agenta (20_000)
- min_memory - minimalna liczba zebranych w pamięci historycznych kroków, zanim agent zacznie się uczyć (3000)
- epsilon - ϵ, początkowe prawdopodobieństwo podjęcia losowej akcji (0.8)
- eps_minimum - minimalna wartość epsilona (0.01)
- batch_size - liczba historycznych kroków na których jednorazowo agent był uczony (32)
- survived_step_reward - dodatkowa nagroda za przeżycie każdego kroku (0.1)

Ponadto, zmiana następujących parametrów, z różnych przyczyn nie została poddana analizie:

- num_actions - liczba dostępnych akcji, które agent może podjąć (2 - stała dla środowiska FlappyBird)
- seed - ziarno dla generatorów liczb losowych (modyfikowane w celu uśrednienia wyników)
- episodes - liczba gier rozegranych przez agenta (500 - stała dla całej analizy)
- eps_decrement - ubytek epsilona w każdym kroku (0.99 - stały dla całej analizy)

- 27 różnych modeli
- każdy wariant trenowany 3-krotnie
- 500 gier
- modyfikowany 1 parametr, ceteris paribus
- łącznie około 1190 minut (ca. 20h)

Najlepszy agent




```
{'mlp_hidden_units': [256, 128, 64],
 'learning_rate': 5e-05,
 'discount_factor': 0.95,
 'memory_size': 15000,
 'min_memory': 3000,
 'epsilon': 1.0,
 'eps_decrement': 0.99,
 'eps_minimum': 0.01,
 'batch_size': 32,
 'survived_step_reward': 1.0}
```

HIPERPARAMETRY ORAZ PRZEBIEG TRENINGU

Wybrany na podstawie analizy agent rozegrał 1500 gier

Dziękuję za uwagę

PIOTR PAWŁOWSKI