Machine Learning 2

Lab 2

Oscar Teeninga

1. Kod

Uzupełniłem luki zgodnie z instrukcją. Zastosowałem funkcję Categorical().eperimental_sample_and_log_prob(), który zwraca parę wylosowanej akcji oraz logarytm z prawdopodobieństwa wybranej akcji (a więc dokładnie to o co nam chodzi)

```
self.optimizer: tf.keras.optimizers.Optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
# TODO: przygotuj odpowiedni optymizator, pamiętaj o learning rate!
```

```
probabilities = tfp.distributions.Categorical(probs=self.model(state)[0][0])
action, self.log_action_probability = probabilities.experimental_sample_and_log_prob()
# TODO: tu trzeba wybrać odpowiednią akcję korzystając z aktora
```

```
if not terminal:
    error = reward + self.discount_factor * float(self.model(new_state)[1][0].numpy()) - self.model(state)[1][0]
else:
    error = reward - self.model(state)[1][0]
# TODO: tu trzeba obliczyć błąd wartościowania aktualnego krytyka
self.last_error_squared = error ** 2
loss = self.last_error_squared - float(error.numpy()) * self.log_action_probability
# TODO: tu trzeba obliczyć sumę strat dla aktora i krytyka
```

Tutaj wkradł się duży błąd, ponieważ korzystając ze wcześniejszej instrukcji ustawiłem krok uczący na 0.0001, zamiast 0.00001, a więc wyniki będą nieco inne niż oczekiwane (do etapu zwiększenia kroku uczącego na patyku). Dla zasady przetestowałem dodatkowo dla standardowej sytuacji, ale już z poprawnym krokiem.
Kolejnym błędem wynikającym ze złej wersji jest BatchNormalization zamiast LayerNormalization - moment poprawki jest ten sam.

Wszystkie testy dla kijka przeprowadziłem w dwóch konfiguracjach:

1. BatchNormalization i learning_rate = 0.0001 (poprzednia wersja instrukcji)
2. LayerNormalization i learning_rate = 0.00001 (nowa wersja)

Omyłkowo wziąłem złą wersję i wyszło jak wyszło, że testowałem drugi raz, a skoro wyniki już mam to warto sobie porównać. W sprawozdaniu będę nazywał to odpowiednio konfiguracją (1) i konfiguracją (2)

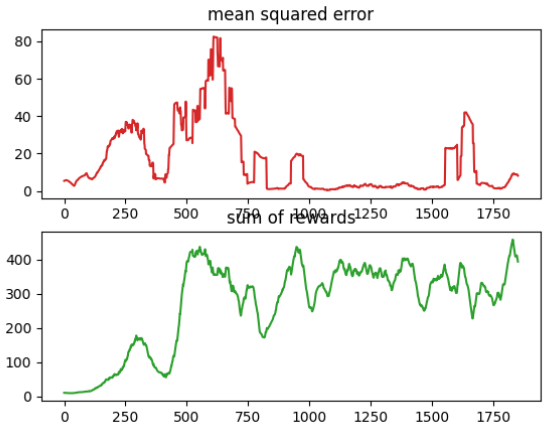
2. Patyk z wspólnymi warstwami

Zgodnie z poleceniem stworzyłem sieć neuronowym o jednym wejściu o rozmiarze 4 (ponieważ stan składa się z 4 wartości) oraz rozdzielny koniec sieci dedykowany aktorowi i krytykowi.

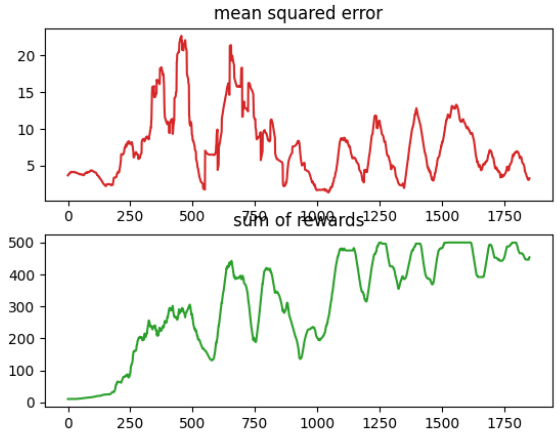
```python
@staticmethod
def create_actor_critic_model() -> tf.keras.Model:
    x_in = tf.keras.layers.Input(4)
    x = tf.keras.layers.Dense(1024, activation='relu')(x_in)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.Dense(256, activation='relu')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    actor_output = tf.keras.layers.Dense(2, activation="softmax")(x)
    critic_output = tf.keras.layers.Dense(1, activation="linear")(x)
    # TODO: przygotuj potrzebne warstwy sieci neuronowej o odpowiednich aktywacjach i rozmiarach
    return tf.keras.Model(inputs=x_in, outputs=[actor_output, critic_output])
```

```python
def create_actor_critic_model() -> tf.keras.Model:
    x_in = tf.keras.layers.Input(4)
    x = tf.keras.layers.Dense(1024, activation='relu')(x_in)
    x = tf.keras.layers.LayerNormalization()(x)
    x = tf.keras.layers.Dense(256, activation='relu')(x)
    x = tf.keras.layers.LayerNormalization()(x)
    actor_output = tf.keras.layers.Dense(2, activation="softmax")(x)
    critic_output = tf.keras.layers.Dense(1, activation="linear")(x)
    # TODO: przygotuj potrzebne warstwy sieci neuronowej o odpowiednich aktywacjach i rozmiarach
    return tf.keras.Model(inputs=x_in, outputs=[actor_output, critic_output])
```

Proces uczenia trwał ok. 2h. Widzimy, że uczenie się udało, około 1100 iteracji osiągamy 500 punktów nagrody, a ~1500 jest stały wynik maksymalny.



Patyk(1): wspólne warstwy



Patyk(2): wspólne warstwy

Zależność błędu od nagrody jest na
pierwszy rzut oka taka, że pewna zależność istnieje, natomiast nie jest jasno określona. Gdy error jest duży to można zauważyć, że zmiana nagrody jest większa. Gdy błąd jest stały i niski to nagroda nie rośnie ani nie maleje. Widać, że błąd jest impulsem do zmiany.

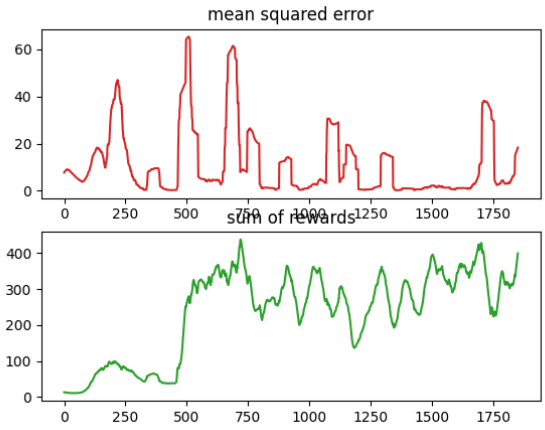
Porównując obie konfiguracje, widzimy, że lepiej poradziła sobie konfiguracja 2, ze względu na niestabilność rozwiązania w którym małe kroki będą lepsze. Natomiast konfiguracja 1 jest szybsza i już ok 500 iteracji otrzymujemy bardzo dobry wynik grubo ponad 400 punktów.

3. Patyk z rozdzielnymi warstwami

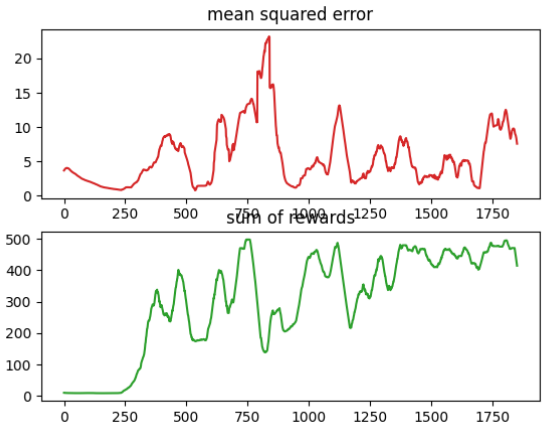
Zgodnie z poleceniem, stworzyłem dwie oddzielne sieci neuronowe (ze wspólnym wejściem) dla ułatwienia.

```python
@staticmethod
def create_actor_critic_model() -> tf.keras.Model:
    x_in = tf.keras.layers.Input(4)
    a_x = tf.keras.layers.Dense(1024, activation='relu')(x_in)
    a_x = tf.keras.layers.BatchNormalization()(a_x)
    a_x = tf.keras.layers.Dense(256, activation='relu')(a_x)
    a_x = tf.keras.layers.BatchNormalization()(a_x)
    actor_output = tf.keras.layers.Dense(2, activation="softmax")(a_x)
    c_x = tf.keras.layers.Dense(1024, activation='relu')(x_in)
    c_x = tf.keras.layers.BatchNormalization()(c_x)
    c_x = tf.keras.layers.Dense(256, activation='relu')(c_x)
    c_x = tf.keras.layers.BatchNormalization()(c_x)
    critic_output = tf.keras.layers.Dense(1, activation="linear")(c_x)
    # TODO: przygotuj potrzebne warstwy sieci neuronowej o odpowiednich aktywacjach i rozmiarach
    return tf.keras.Model(inputs=x_in, outputs=[actor_output, critic_output])
```

W tym przypadku nauka trwała nieco dłużej, bo ok. 4h. Ciekawą różnicą było to, że w wizualizacji było widać jak od samego początku (300 iteracji) widać było mocne kołysanie się na boki, jak huśtawka, z lewa na prawo. W przypadku poprzednim nie była widoczne żadne podobne zjawisko.
Wyniki dla konfiguracji (2) są znacznie lepsze, już przy ok. 380 iteracjach otrzymujemy bardzo fajne rezultaty. Jest to zatem najlepsze rozwiązanie do tej pory. Rozwiązanie jest znacznie stabilniejsze niż jedna sieć.



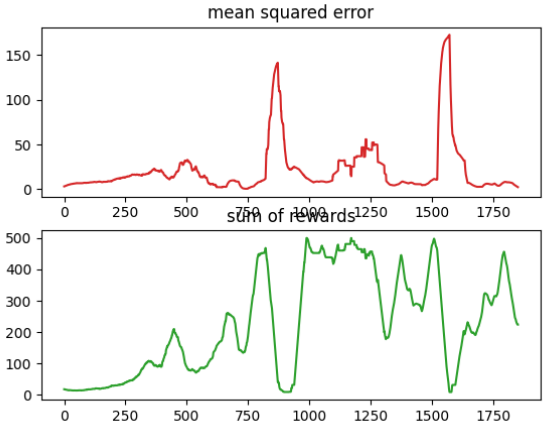
Patyk(1): rozdzielne warstwy



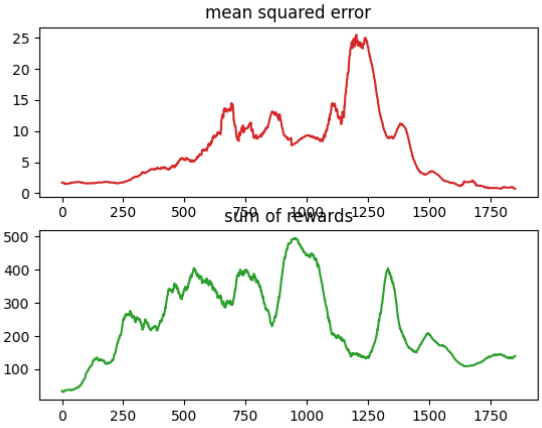
Patyk(2): rozdzielne warstwy

4. Patyk z siecią o mniejszej liczbie neuronów

Uczenie przebiegało szybciej niż w poprzednich przypadkach. Widać, że mniejsza sieć jest w stanie szybciej osiągnąć akceptowalne wyniki, natomiast jest dużo bardziej niestabilne, co sprawia, że po pewnym czasie nie poprawiamy nagrody - a wręcz przeciwnie - idzie coraz gorzej.



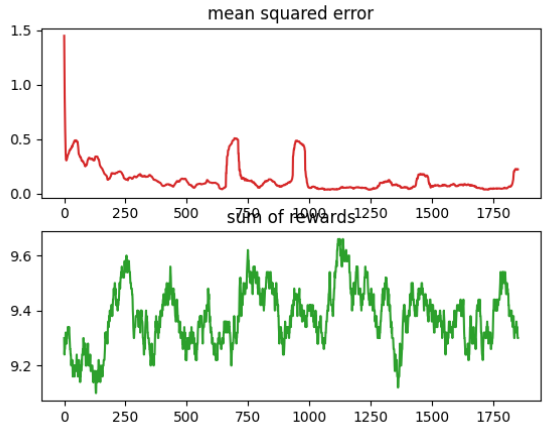
Patyk(1): mniejsza sieć



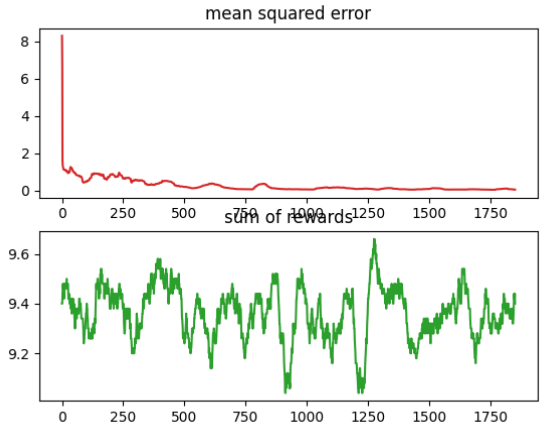
Patyk(2): mniejsza sieć

5. Patyk - zwiększony krok uczenia

Zwiększenie kroku uczącego do 0.01 sprawia, że uczenie zupełnie przestaje dawać rezultaty i wygląda to jak random. Ciekawe jest, że błąd jest zerowy, przez co algorytm nie wie, że “da się lepiej”.



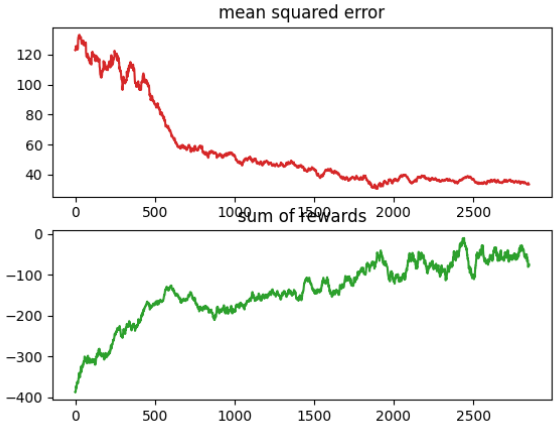
Patyk(1): zwiększony krok uczenia



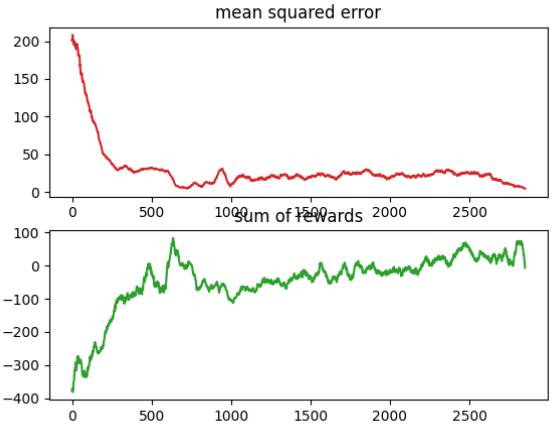
Patyk(2): zwiększony krok uczenia

6. Lądowanie

Widzimy, że mniejsza krok uczenia nie dał lepszych rezultatów. Udało się osiągnąć w okolicach 700 iteracja bardzo fajny wynik ok. 100 punktów nagrody. Obserwując zachowanie lądownika w ostatnich kilkuset iteracjach widać było, że starał się maksymalnie stabilizować lot, nieco zaniedbując lądowanie w środku - ważniejsze było, żeby wylądować bezpiecznie. Cała nauka trwała około 12 godzin.



α = 0.0000001



α = 0.000001

7. Ręcznie dobrane wartości dla patyka oraz lądownika

Napisałem specjalny skrypt, który ładuje model, a następnie wybiera akcję oraz wartościowanie dla danego stanu. Zaobserwowane wynik wpisałem do tabeli. Zapisałem sobie modele jako ‘cart.model’ oraz ‘lunar.model’.

Wszystkie wartości stanów były w postaci [0, …, 0, a, 0, …, 0], gdzie a = 1 (Max) lub a = -1 (Min). W większości przypadków zmiana min na max sprawia, że wartość się zmienia oraz akcja również.

Dla patyka mamy dwie akcje - prawo i lewo. Zgodnie z tym co widzimy, gdy kijek przechyla się i porusza się w drugą stronę - akcja się zmienia - w celu utrzymania kijka w pionie.

Patyk	**Max**		**Min**	
	Wartość	**Akcja**	**Wartość**	**Akcja**
Pozycja wagonika	80.6187	1	69.9600	1
Prędkość wagonika	81.8561	1	80.9398	0
Kąt wychylenia kijka	42.7593	1	65.3734	0
Prędkość kątowa	79.6691	1	83.7940	0

Dla lądownika analiza jest nieco trudniejsza. Tutaj akcji mamy cztery:

0) Nie rób niczego
1) Lewy silnik
2) Dolny silnik
3) Prawy silnik

Wydaje się, że żadne z zebranych informacji nie mówią niczego ciekawego. Widać, że najczęściej używany jest silnik prawy i lewy - ponieważ ich użycie nie jest w żaden sposób karany. Gdy położenie w poziomie jest maksymalne, używamy dolnego silnika, ponieważ chcemy utrzymać statek w sensownym położeniu. Poza tym nie widzę niczego sensownego.

Lądownik	**Max**		**Min**	
	Wartość	**Akcja**	**Wartość**	**Akcja**
Położenie w poziomie	-6.7347	2	-0.8915	3
Położenie w pionie	-11.924	3	9.5837	2
Prędkość w pionie	6.2111	1	8.8427	3
Prędkość w poziomie	5.6425	1	16.362	2
Kąt nachylenia	14.395	3	15.092	1
Prędkość kątowa	-20.003	0	-17.614	1
Lewa noga	15.625	3	0.6804	2
Prawa noga	5.8299	2	3.7039	1