Uniwersytet Jagielloński w Krakowie

Wydział Fizyki, Astronomii i Informatyki Stosowanej

Piotr Kucharski

Nr albumu: 1124564

Uczenie agentów sterowania pojazdami kosmicznymi

Praca licencjacka
na kierunku Informatyka Stosowana

Praca wykonana pod kierunkiem
prof. dr hab. Piotr Białas
Zakład Technologii Gier

Kraków 2020

Oświadczenie autora pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

.....	.....
Kraków, dnia	Podpis autora pracy

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

.....	.....
Kraków, dnia	Podpis kierującego pracą

Spis treści

1 Abstrakt

Niniejsza praca prezentuje proces optymalizacji parametrów sieci neuronowej za pomocą uczenia ze wzmocnieniem. Zadaniem sieci neuronowej jest sterowanie pojazdem w środowiskach symulowanej przestrzeni kosmicznej na podstawie obserwacji stanu pojazdu. W środowisku celem jest znalezienie optymalnej strategii na zatrzymanie poruszającego się pojazdu lub dotarcie pojazdem do wyznaczonego punktu.

2 Wstęp

Dziedziny głębokiego uczenia maszynowego takie jak detekcja obiektów na zdjęciach czy rozpoznawanie mowy są już z powodzeniem wykorzystywane w komercyjnych produktach. Uczenie ze wzmocnieniem które polega na optymalizacji procesu podejmowania decyzj pomimo wielu sukcesów w rozwiązywaniu skomplikowanych strategii w symulacjach i grach komputerowych ma bardzo wąskie zastosowania. Algorytm Atari DQN [7] z 2014 roku zaprezentował uniwersalny sposób szukania optymalnej strategii w kilku prostych grach Atari. Już w roku 2016 algorytm AlphaGo [10] zwyciężył z mistrzem świata w grę GO. Zaledwie rok później ten sam zespół opublikował algorytm będący w stanie nauczyć się optymalnej strategii podejmowania decyzji w każdej grze 2 osobowej bez wykorzystywania eksperckiej wiedzy podczas treningu, AlphaZero [11]. Tutaj pojęcie nauczenia się optymalnej strategii sprowadza się do znalezienia lepszej strategii, czyli takiej która pozwala pokonać dotychczasowych mistrzów świata we wszystkie z wymienionych gier. W 2019 zwycięstwa w dwuch następnych grach odniosły algorytmy AlphaStar [13] oraz OpenAI Five [8] pokonując mistrzów w grach, które są uważane za najtrudniejsze. Sukcesów w wirtualnych środowiskach nie da się przenieść na świat rzeczywisty, gdzie konsekwencje podjęcia złej decyzji mogą wpływać na życie ludzi. Jednym z dziewięciu wyzwań uczenia ze wzmoenieniem [3] jest możliwość przejrzystej interpretacji zachowań agenta, który chciałbym poruszyć w tej pracy.

3 Definicja uczenia ze wzmocnieniem

Proces uczenia agenta polega na interakcji ze środowiskiem ε w sekwencji akcji, obserwacji i nagród której celem jest uzyskiwania największych możliwych nagród. W każdym kroku agent na podstawie obserwacji przekazuje do środowiska wybraną akcję, środowisko aktualizuje stan symulacji uwzględniając odebraną akcję, oraz zwraca następne obserwacje, nagrodę za ten krok oraz informację czy aktualizacja doprowadziła do terminacji symulacji agenta. Proces Decyzyjny Markova jest matematycznym sformułowaniem problemu uczenia się interakcji ze środowiskiem, którego celem jest osiągnięcie najwyższej możliwej nagrody. Składa się z krotki *(S, A, P, R)* gdzie S to zbiór stanów środowiska, A to skończony zbiór akcji które agent może wykonać w środowisku, P to rozkład prawdopodobieństwa przejścia pomiędzy stanami, R to funkcja nagrody.

$$P(s'|s,a) = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a] \tag{1}$$

$$R(s,a) = \mathbb{E}[R_t|S_t = s, A_t = a] \tag{2}$$

Proces interakcji ze środowiskiem tworzy ciąg $S_1, A_1, R_1, S_2, A_2, R_2 ... S_T, A_T, R_T$ gdzie T oznacza numer kroku w którym agent osiągnął stan terminalny. Politykę π definiuje się jako funkcję rozkładu prawdopodobieństwa akcji na zadanym stanie

$$\pi(a|s) = \mathbb{P}[A_t = a|S_t = s] \tag{3}$$

Optymalna polityka π^* w każdym stanie ze zbioru S podejmuje akcje która maksymalizuje oczekiwaną nagrodę którą agent jest w stanie uzyskać zaczynając z danego stanu. Standardowo korzystam z parametru $0 < \gamma < 1$ zmniejszający nagrody z przyszłych kroków. Maksymalna oczekiwana nagroda możliwa do uzyskania z danego stanu definiowana jest jako optymalna funkcja wartości.

$$V^*(s) = \max_a [R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s')] \tag{4}$$

Maksymalna oczekiwana nagroda do uzyskania z danego stanu po podjęciu określonej akcji to optymalna funkcja wartości akcji Q^*.

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a') \tag{5}$$

$$V^*(s) = \max_a Q^*(s,a)$$

Jeśli optymalna wartość $Q^*(s',a')$ w następnym kroku jest znana dla wszystkich możliwych akcji a' to optymalną polityką podejmowania akcji jest wybieranie a' maksymalizując oczekiwaną nagrodę.

$$\pi^*(s) = arg \max_a Q^*(s,a) \tag{6}$$

$$V_\pi(s) = \sum_a \pi(a|s)[R(s,a) + \gamma \sum_{s'} P(s'|s,a) V_\pi(s')] \tag{7}$$

$$Q_\pi(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \sum_{a'} \pi(a'|s') Q_\pi(s',a') \tag{8}$$

$$V_\pi(s) = \sum_a \pi(a|s) Q_\pi(s,a)$$

Funkcję $Q(s,a)$ można optymalizować za pomocą iteracji równania Temporal Difference [12] gdzie α to wielkość kroku optymalizacyjnego.

$$Q_{i+1}(s,a) = Q_t(s,a) + \alpha[R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q_t(s',a') - Q_t(s,a)] \tag{9}$$

Iterowanie funkcji wartości akcji zbiega do optymalnej funkcji $Q_i \to Q^*$ dla $i \to \infty$ [12]. W praktyce takie podejście jest niepraktyczne, gdyż algorytm uczący nie ma dostępu do funkcji $P(s'|s,a)$ oraz jest za dużo stanów w których może się znaleźć agent. W omawianym środowisku przestrzeń możliwych stanów jest wielowymiarowa i ciągła. Bez dostępu do $P(s'|s,a)$ algorytm musi się uczyć na podstawie zaobserwowanych stanów, wykonanych akcji i nagród, co wymagałoby ponownej iteracji funkcji Q dla każdej nowej sekwencji nie oferując przy tym generalizacji.

4 Głębokie uczenie funkcji wartości akcji Q

Bez jawnie podanej funkcji przejścia $P(s'|s,a)$ optymalizowane funkcje muszą bazować na ciągu obserwacji S_t, akcji A_t i nagród R_t. Ciąg jest generowanym podczas interakcji agenta podążającego polityką π ze środowiskiem ε. Elementy ciągu zapisane są w pamięci M w której zawarte są również następne stany S_{t+1}.

$$V_\pi(s) = \mathbb{E}[R_t + \gamma V_\pi(S_{t+1})|S_t = s, A_t \sim \pi(S_t)] \tag{10}$$

$$Q_\pi(s,a) = \mathbb{E}[R_t + \gamma Q_\pi(S_{t+1}, A_{t+1})|S_t = s, A_t = a, \pi] \tag{11}$$

Celem znalezienia optymalnej strategii korzystam z nieliniowej funkcji z parametrami θ która aproksymuje optymalną funkcję wartości akcji $Q(s,a:\theta) \approx Q^*(s,a)$. W tej pracy nieliniową funkcją jest głęboka sieć neuronowa z nieliniowymi aktywacjami. Sieć neuronową można uczyć minimalizując sekwencje funkcji kosztu $L_i(\theta_i)$.

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s' \sim M}[(r + \gamma \max_{a'} Q(s', a' : \theta_{i-1}) - Q(s, a : \theta_i))^2] \tag{12}$$

Wagi sieci neuronowej optymalizuje się za pomocą stochastycznego spadku wzdłuż gradientu.

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a,r,s' \sim M}[(r + \gamma \max_{a'} Q(s', a' : \theta_{i-1}) - Q(s, a : \theta_i)) \nabla_{\theta_i} Q(s, a : \theta_i)] \tag{13}$$

Wagi θ_{t-1} są zamrożone w czasie kroku uczącego. Krok optymalizacyjny następuje po każdej aktualizacji stanu środowiska po podjęciu akcji.

4.1 Problemy głębokiego uczenia ze wzmocnieniem

Kolejne stany środowiska są bardzo do siebie podobne co w procesie uczenia prowadziłoby do nadmiernego dopasowania funkcji Q bądź zbiegania do lokalnego minimum. Celem uniknięcia tej sytuacji pamięć M ma dużą pojemność n, a partie do uczenia o wielkości $b < n$ są wybierane z pamięci z jednakowym prawdopodobieństwem $(S_b, A_b, R_b, S'_b) \sim M$. Prowadzi to do zwiększenia niezależności obserwacji na podstawie których obliczany jest krok uczący. Drugim sposobem na przeciwdziałanie podobieństwa stanów jest jednoczesne symulowanie wielu niezależnych pojazdów które inicjalizowane są w sposób losowy. Znacząco zwiększa to różnorodność obserwowanych stanów. Prowadzi to do lepszej eksploracji przestrzeni stanów oraz pozwala na lepszą ewaluację aktualnej polityki podczas treningu. Wyniki przeprowadzonych eksperymentów wskazują na znaczącą poprawę wyników już przy 5 symulowanych jednocześnie pojazdach.

Gdyby agent od początku chciwie podejmował akcje na podstawie losowo zainicjalizowanej funkcji wartości akcji $a_t = \max_a Q(s_t, a : \theta)$ mógłby, przez niedostateczną eksplorację, pozostać w lokalnym minimum. Eksplorację wymusza się na agencie poprzez wybieranie z prawdopodobieństwem ϵ losowej akcji $a_t \sim A$, a z prawdopodobieństwem $1 - \epsilon$ akcji chciwej $a_t = \max_a Q(s_t, a : \theta)$. Ten sposób podejmowania akcji nazywa się ϵ-greedy[12]. Wartość ϵ jest warunkowana początkową wartością ϵ_{start}, końcową wartością ϵ_{end} oraz ilością kroków po których ϵ ma osiągnąć wartość końcową ϵ_{decay}. W trakcie treningu ϵ zmniejsza się z każdym krokiem zgodnie z równaniem $\epsilon_t = \epsilon_{end} + (\epsilon_{start} - \epsilon_{end}) e^{(-t/\epsilon_{decay})}$.

Następnym problemem w głębokim uczeniu ze wzmocnieniem są eksplodujące gradienty. Główną przyczyną eksplozji jest duże odchylenie standardowe zaobserwowanych nagród, obserwacje zawierające zmienne o różnych skalach wielkości. Wagi sieci θ_{t-1} służą do obliczania funkcji kosztu jak i na podstawie wag θ są generowane obserwacje podczas interakcji ze środowiskiem. Z tych powodów problem eksplodujących gradientów występuje w uczeniu ze wzmocnieniem częściej niż w przypadku innych dziedzin uczenia maszynowego. Żeby przeciwdziałać eksplodującym gradientom korzystam z funkcji kosztu $smooth\_L1\_loss$ [4] celem zmniejszenia wpływu skrajnych pomiarów na całkowity gradient danego kroku uczącego.

$$smooth_{L_1}(x) = \begin{cases} 0.5x^2 & dla|x| < 1 \\ |x| - 0.5 & dla|x| \geq 1 \end{cases} \tag{14}$$

Podobnie jak w Atari DQN [7] korzystam z optymalizatora *RMSprop* zaproponowanego przez G. Hinton [14] i po raz pierwszy opublikowanego w pracy na temat optymalizacji rekurencyjnych sieci neuronowych służących do generowania tekstu[6]. Optymizator przechowuje średnią kroczącą kwadratów gradientów dla każdej wagi $w \in \theta$, a podczas aktualizacji wagi dzieli gradient przez pierwiastek kwadratowy tej średniej.

$$\mathbb{E}[g^2]_t = \beta\mathbb{E}[g^2]_{t-1} + (1-\beta)(\frac{\delta L(x)}{\delta w})^2 \tag{15}$$

$$w_t = w_{t-1} - \frac{\alpha}{\sqrt{\mathbb{E}[g^2]_t}}\frac{\delta L(x)}{\delta w} \tag{16}$$

4.2 Algorytm

Algorithm 1: Głębokie uczenie Q-network z wykorzystaniem pamięci

Inicjalizacja parametrów θ funkcji wartości akcji Q, $\theta_f = \theta$
Inizjalizacja środowiska oraz otrzymanie pierwszej obserwacji s_1
for *t=0; t<T; t++* **do**
  Z prawdopodobieństwem ϵ wybierz losową akcję $a_t \sim A$,
  w innym przypadku wybierz $a_t = \max_a Q(s_t, a : \theta)$
  Zaktualizuj stan środowiska wykonując akcję a_t otrzymując (r_t, s_{t+1})
  Zapisz w pamięci element (s_t, a_t, r_t, s_{t+1})
  if *t>b* **then**
    $(S_b, A_b, R_b, S'_b) \sim M$
    Zaktualizuj wagi θ zgodnie z $(R_b + \gamma \max_{a'} Q(S'_b, a' : \theta_f) - Q(S_b, A_b : \theta))^2$
  end
  if *(t % częstotliwość aktualizacji θ_f) == 0* **then**
    $\theta_f = \theta$
  end
end

4.3 Sztuczna sieć neuronowa

Funkcja aproksymująca Q^* jest złożona z wielu warstw funkcji liniowych zawierających nieliniowe aktywacje pomiędzy kolejnymi warstwami. Pierwsza funkcja liniowa przyjmuje na wejściu wektor obserwacji otrzymany ze środowiska, przekształca na wektor o wymiarze w a następnie oblicza nieliniową funkcję aktywacji $ReLu$[2] na wektorze wyjściowym. Kolejne d funkcji liniowych o tej samej szerokości w przekształca wektor w ten sam sposób do pierwsza funkcja. Ostatnia funkcja liniowa przekształca wektor o wymiarze w na wektor o wymiarze odpowiadającym rozmiarowi przestrzeni akcji A. Wektor wyjściowy ostatniej funkcji opisuje oczekiwaną wartość wszystkich akcji w zadanym stanie, dlatego omijany jest krok obliczania nieliniowej funkcji aktywacji. W tej pracy korzystam z algorytmów uczenia maszynowego zaimplementowanych w bibiliotece PyTorch[9].

5 Środowisko

Na potrzeby tej pracy zaimplementowałem w języku C++ ciągłą dwuwymiarową przestrzeni po której zgodnie z zasadami dynamiki Newtona poruszają się pojazdy bez oporów wpływających na aktualną prędkość z którą porusza się pojazd. Środowisko korzysta z biblioteki SFML[5] do renderowania stanu środowiska za pomocą prymitywnych kształtów oraz z biblioteki BOOST[1] która dostarcza możliwość wywoływania funkcji zaimplementowanych w C++ w wątku pythona oraz możliwość przekazywania adresów wektorów Numpy pomiędzy symulacją a algorytmem uczenia maszynowego. Środowisko jest obiektem alokowanym i całkowicie kontrolowanym przez wątek uczenia maszynowego w Pythonie. Interfejs komunikacji ze środowiskiem złożony jest z czterech elementów: konstruktor zwracający inteligentny współdzielony wskaźnik, funkcja kroku, funkcja pozwalająca zresetować środowisko do stanu początkowego, informacja o tym czy środowisko jest nadal aktywne oraz destruktor.

5.1 Obserwacje

Agent nie ma dostępu do rzeczywistego stanu środowiska, ale ma dostęp do obserwacji. Obserwacje są przeliczane dla każdego agenta w nieinercjalnym układzie odniesienia, gdzie środek układu oraz jego rotacja jest ustawiony na podstawie aktualnego położenia pojazdu. Wszystkie wektory które agent obserwuje są przeliczane ze współrzędnych kartezjańskich na współrzędne biegunowe. Współrzędne biegunowe uznałem za łatwiejsze do interpretacji przez sieć neuronową, jednak nie dowiodłem tego eksperymentalnie. Agent w każdym kroku środowiska otrzymuje wektor obserwacji złożony z długości prędkości, amplitudy prędkości w odniesieniu do aktualnej rotacji pojazdu oraz szybkości kątowej. W scenariuszu w którym trzeba dolecieć do wyznaczonego punktu agent dodatkowo obserwuje odległość od wyznaczonego punktu oraz odległość kątową pomiędzy kierunkiem pojazdu a prostą przechodzącą przez środek pojazdu i wyznaczony punkt.

5.2 Przestrzeń akcji

Pojazdy poruszają się za pomocą kontrolowania mocy silnika nadającego przyspieszenie do przodu oraz silnika nadającego przyspieszenie obrotowe. Dla uproszczenia treningu agent

może jedyne włączyć bądź wyłączyć silnik główny $[0, 1]$. Żeby wykluczyć z przestrzeni negujące się wzajemnie akcje kontrolowaniem rotacji pojazdu zajmuje się jeden silnik którym agent może sterować lewo, wyłączony, prawo $[-1, 0, 1]$. Przestrzeń akcji jest tworzona w wyniku przecięcia możliwych akcji silnika głównego oraz silnika rotacyjnego. Z przestrzeni usunięty jest element neutralny, oba silniki wyłączone, gdyż w początkowym etapie uczenia ta akcja była faworyzowana co prowadziło do stagnacji procesu uczenia.

5.3 Funkcja aktualizacji środowiska

Środowisko aktualizowane jest jednocześnie dla wszystkich pojazdów. Agent wywołuje funkcję aktualizacji środowiska przekazując wskaźnik do wektora w którym są zapisane podjęte w tym kroku akcje. Wektor akcji składający się z liczb zmiennoprzecinkowych ma wielkość $n * 2$ gdzie n to ilość symulowanych pojazdów oraz każdy pojazd ma dwa silniki do kontrolowania. Wartość maksymalnej mocy silnika jest mnożona przez wartość przekazaną w akcji i na tej podstawie jest aktualizowana jest prędkość pojazdu. Środowisko aktualizuje swój stan ze stałym krokiem czasowym wynoszącym dziesiątą część sekundy. Po zaktualizowaniu położenia sprawdzane są warunki kończące symulacje oraz obliczanie nagrody dla każdego pojazdu. Funkcja zwraca krotkę z adresami pamięci pod którymi znajdują się wektory zawierające obserwacje, nagrody oraz binarną informację czy dany pojazd osiągnął stan terminalny. Po osiągnięciu stanu terminalnego pojazd jest resetowany do stanu początkowego zgodnie z zasadami narzuconymi przez scenariusz.

5.4 Modelowanie funkcji nagrody oraz scenariusze

Na potrzeby tej pracy zaimplementowałem 3 scenariusze. We wszystkich scenariuszach pojazd posiada te same akcje, parametry oraz sposób aktualizacji. Scenariusz wybierany jest na podstawie parametru podanego podczas alokacji obiektu środowiska. Interfejs środowiska pozostaje niezmienny.

Pierwszy scenariusz polega na zatrzymaniu rozpędzonego pojazdu. Pojazd zaczyna symulację posiadając losową rotację, losową prędkość obrotową z zakresu $[-2, 2]/s$ oraz wektor prędkości którego elementy losowane są z zakresu $[5, 10]m/s$. Symulacja pojazdu jest resetowana z negatywną nagrodą w momencie gdy szybkość pojazdu przekroczy $20m/s$ albo prędkość obrotowa przekroczy $2/s$. Symulacja pojazdu jest resetowana z pozytywną nagrodą kiedy pojazd osiągnie szybkość poniżej $2m$. Otrzymywana przez agenta nagroda odpowiada odwrotnej zmianie szybkości pomiędzy kolejnymi krokami aktualizacji środowiska. Parametry pojazdu są dobrane w taki sposób, aby wartości nagrody były z zakresu $[-1, 1]$.

Drugi scenariusz polega na dotarciu statkiem w pobliże wyznaczonego punktu kontrolnego. Podobnie jak w pierwszym scenariuszu pojazd na początku posiada losową rotację, prędkość obrotową $[-1, 1]/s$, elementy składowe prędkość z zakresu $[0, 10]m/s$. Współrzędne punktu kontrolnego losowane są z zakresu $[-700, 700]m$. Pojazd jest resetowany z negatywną nagrodą kiedy jego prędkość obrotowa przekroczy $2/s$ albo oddali się od punktu kontrolnego na więcej niż $1400m$. Nagroda odpowiada odwrotnej zmianie odległości pojazdu od punktu kontrolnego.

Trzeci scenariusz od drugiego różni się sposobem obliczania nagrody. W trzecim scenariuszu nagroda odpowiada zmianie długości wektora prędkości po rzutowaniu na prostą łączącą pojazd z punktem kontrolnym.

5.5 Graficzna reprezentacja środowiska

6 System do przeprowadzania eksperymentów

7 Przeprowadzone eksperymenty

8 Wyniki i ich interpretacja

9 pomysły i propozycje następnych eksperymentów, badań

Literatura

[1] boost c++ libraries.

[2] Abien Fred Agarap. Deep learning using rectified linear units (relu). *CoRR*, abs/1803.08375, 2018.

[3] Gabriel Dulac-Arnold, Daniel J. Mankowitz, and Todd Hester. Challenges of real-world reinforcement learning. *CoRR*, abs/1904.12901, 2019.

[4] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[5] Laurent Gomila. Simple and fast multimedia library sfml.

[6] Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[8] OpenAI. Openai five defeats dota 2 world champions, 2019.

[9] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alche-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[10] David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

[11] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362:1140–1144, 12 2018.

[12] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018.

[13] Oriol Vinyals, Igor Babuschkin, Wojciech Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John Agapiou, Max Jaderberg, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575, 11 2019.

[14] Geoffrey Hinton with Nitish Srivastava Kevin Swersky. Neural networks for machine learning.