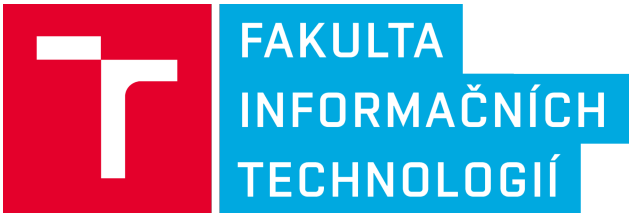


Vysoké učení technické v Brně

EVO

Aplikované evoluční algoritmy

Evoluční návrh neuronové sítě v modelové úloze

Autor:
Petr Buchal

Login:
xbucha02

10. května 2020

1 Úvod

Pomocí evolučního algoritmu jsem se pokusil navrhnout vhodnou architekturu neuronové sítě využívanou v algoritmu DQN. Evoluční algoritmus je schopný navrhnout ideální optimalizátor, počet vrstev a i jednotlivé vrstvy - počet neuronů v každé vrstvě a její aktivační funkci. Algoritmus DQN jsem aplikoval na prostředí CartPole-v0.

2 Hluboké Q-učení

Problém, který má vyvíjená neuronová síť řešit spadá do oblasti zpětnovazebního učení, jedná se o metodu Hlubokého Q-učení. Zpětnovazební učení je oblast strojového učení, zabývající se chováním agentů v různých prostředích. Cílem zpětnovazebního učení je, co nejlépe vytrénovat agenta k maximalizaci kumulativní odměny za akce, které v prostředí provádí. Prostředí v jakých se agenti pohybují bývají většinou formulovány jako Markovův rozhodovací proces. Zjednodušeně se jedná o diskrétní stochastický proces, ve kterém se agent nachází ve stavu s a následným provedením jedné z možných akcí a se dostane do stavu s' a obdrží odměnu r [1].

Algoritmus 1 DQN [3]

Inicializuj paměť D s kapacitou N
Inicializuj neuronovou síť Q s náhodnými váhami
for $epizoda = 1, M$ **do**
  Inicializuj prostředí a pozoruj počáteční stav s_t
  for $krok = 1, T$ **do**
    S pravděpodobností ϵ vyber náhodnou akci a_t
    jinak $a_t = argmax(Q(s_t))$
    Proveď akci a_t, pozoruj odměnu r_t a nový stav s_{t+1}
    Ulož vzpomínku (s_t, a_t, r_t, s_{t+1}) do paměti D
    Vezmi náhodný vzorek vzpomínek (s_i, a_i, r_i, s_{i+1}) z paměti D
    $l_i = \begin{cases} r_i & \text{pro stav } s_i, \text{ který je koncový} \\ r_i + \ * \max(Q(s_{i+1})) & \text{pro stav } s_i, \text{ který není koncový} \end{cases}$
    Za použití s_i jako trénovacích dat a l_i jako štítků trénuj Q
    $s_t = s_{t+1}$
  end for
end for

DQN vychází z algoritmu Q-učení. Při použití Q-učení se agent snaží pohybovat v prostředí na základě předpovědí tzv. Q-funkce. Ta předpovídá očekávaný užitek z

provedení konkrétní akce v konkrétním stavu tzv. Q-hodnotu. Akci s největší Q-hodnotou poté agent provede. Problémem Q-učení je, že pro každý stav a v něm možnou akci musí mít uložený záznam. Jejich počet činní jen pro piškvorky o velikosti 3x3 skoro 27 000 záznamů. Když se poté vezme v úvahu ještě to, že by agent měl při učení všech 27 000 záznamů aktualizovat, stane se z Q-učení nepříliš použitelná metoda pro složitější prostředí. DQN problém s ukládáním záznamů řeší tak, že je neukládá. Místo toho veškeré Q-hodnoty aproximuje pomocí neuronové sítě. Ta se učí z paměti se vzpomínkami, ve které jsou uloženy stavy, kterými agent prošel [2]. Pseudokód algoritmu DQN se nachází v algoritmu 1.

2.1 Parametry učení

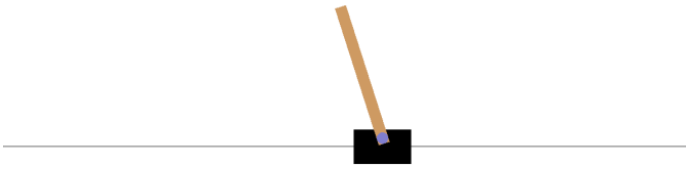
Váhy neuronové sítě jsou na začátku trénování inicializované náhodně. Učící konstanta je nastavena na výchozí hodnotu jednotlivých optimalizátorů. Proměnná ϵ má na začátku trénování hodnotu 1. Její rozklad probíhá lineárně odečítáním hodnoty 0.001 v každém kroku, dokud ϵ neklesne na hodnotu 0.1, od této hranice zůstává hodnota ϵ konstantní. Paměť vzpomínek D má velikost 1 000 záznamů.

3 Prostředí CartPole-v0

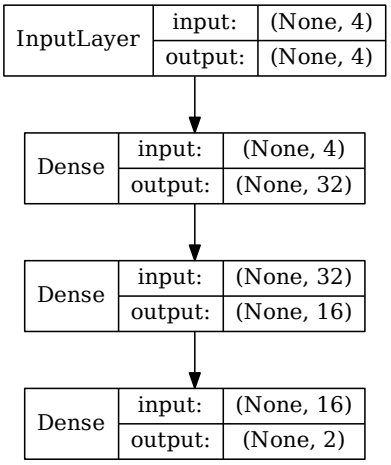
V prostředí CartPole-v0 řeší agent „problém inverzního kyvadla“, viz obrázek 1. Jeho úkolem je balancovat s vozíkem tak, aby se tyč, kterou v sobě vozík veze, nenaklonila o více než 15 stupňů, pokud při tom selže, hra končí. Od začátku pohybu v prostředí agent dostává kladnou odměnu (+1) za udržování rovnováhy, v okamžiku její ztráty dostane odměnu zápornou (-1). Díky těmto rozdílům v odměnách agent od začátku ví, které stavy jsou pro něj dobré a které ne, to mu umožňuje relativně rychlé učení. Toto prostředí je vhodné ke zkoumání rychlosti učení agenta. To je dáno tím, že agent v těchto prostředích není limitován nutností prohledávání stavového prostoru za vidinou lepšího stavu, jelikož se v nejlepším možném stavu nachází od začátku [1]. Agent vyřeší prostředí, pokud ve 100 po sobě jdoucích hrách bude mít průměrné skóre alespoň 195. Prostředí poskytuje agentovi 4 údaje - pozice a rychlost vozíku, úhel tyče a rychlost tyče na špičce. Agent může provést 2 akce - pohyb doleva a pohyb doprava.

4 Neuronová síť

V práci [1] se k řešení prostředí CartPole-v0 používá třívrstvá dopředná neuronová síť, viz obrázek 2. V ní se jako aktivační funkce používají ReLu s výjimkou poslední vrstvy, kde je aktivační funkce lineární. Jako optimalizátor byl využit adam.



Obrázek 1: Vizualizace prostředí CartPole-v0



Obrázek 2: Vizualizace neuronové sítě z [1] použité na prostředí CartPole-v0.

4.1 Zkoumané parametry

Implementovaný evoluční algoritmus neuronové sítě mění její počet vrstev, počet neuronů v jednotlivých vrstvách, aktivační funkce v jednotlivých vrstvách a optimalizátor. Jelikož z [1] víme, že úloha nepotřebuje k řešení velkou neuronovou síť, parametry byly nastaveny tak, aby trénování netrvalo nepřiměřeně dlouho. I s relativně střídmými parametry ale jeden běh algoritmu trval v průměru 22 hodin. Parametry můžete vidět v tabulce 4.1.

Počet neuronů	4	8	16	32	64		
Počet vrstev	1	2	3	4			
Aktivační funkce	relu	elu	tanh	sigmoid			
Optimalizátor	rmsprop	adam	sgd	adagrad	adadelta	adamax	nadam

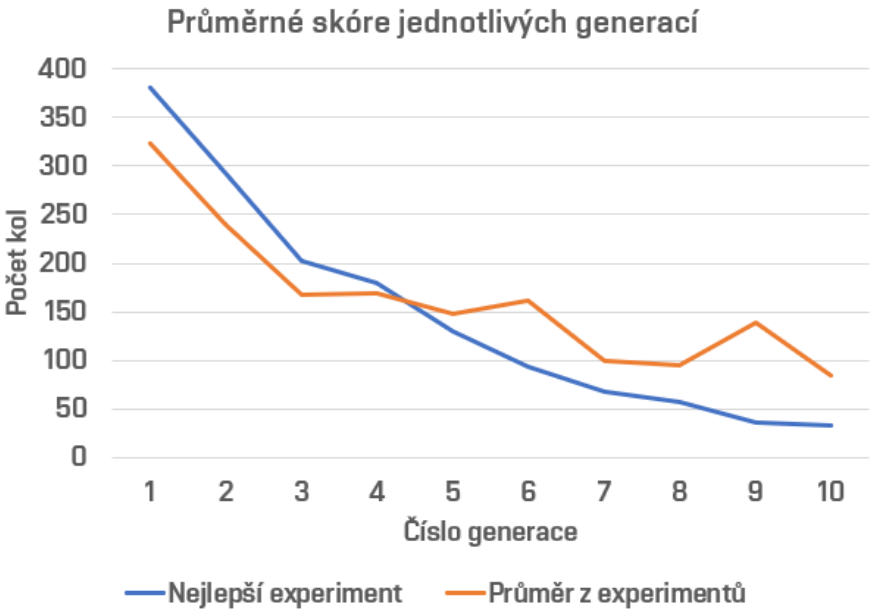
5 Experimenty

Experiment jsem spustil celkem třikrát se stejnými parametry. Velikost populace byla 20 a počet generací 10. Pro další populaci bylo zachováváno vždy 40 % nejlepších sítí, nezachované sítě v další generaci zůstaly s pravděpodobností 10 %. K mutaci docházelo s pravděpodobností 20 %. Pokud došlo k mutaci, tak se s pravděpodobností 30 % provedla změna v počtu vrstev (tzn. náhodná vrstva byla odstraněna nebo byla na náhodné místo přidána nová vrstva s náhodnými parametry). Následně se během mutace s pravděpodobností 70 % pro každou vrstvu stanovilo, zdali v ní dojde ke změně. Ve vrstvách ve kterých mělo dojít ke změně, se s pravděpodobností 50 % změnil počet neuronů a s pravděpodobností 50 % aktivační funkce.

Skóre podle kterého se hodnotila úspěšnost neuronové sítě byl počet her, který neuronová síť potřebovala ke zvládnutí prostředí. Tedy čím menší byl počet her potřebných pro zvládnutí prostředí, tím byl výsledek lepší.

Na obrázku 3 lze vidět průměrné skóre jednotlivých generací v experimentu s nejlepšími výsledky a průměr výsledků ze všech experimentů. Lze pozorovat, že evoluční algoritmus vytvořil architekturu, kterou lze natrénovat opravdu rychle, je tomu ale skutečně tak? Průměrný počet epizod potřebný k natrénování neuronové sítě v [1] byl 280. Při experimentech v [1] se počet epizod potřebný k natrénování neuronové sítě odvíjel od vhodnosti náhodné inicializace neuronové sítě a od počáteční náhodné inicializace paměti vzpomínek. V sekci 5.1 se podrobněji věnuji tomu, zdali je získaná architektura opravdu tak rychlá na natrénování.

Z experimentů vychází nejlépe architektura, která má 4 vrstvy, kde první 3 vrstvy mají 16, 64 a 16 neuronů. Tato neuronová síť obsadila první 4 místa v jednom z provedených experimentů. Architektura používá optimalizátor adam a vrstvy používají aktivační funkce tahn, elu, sigmoid. Obecné poznatky vycházející z experimentů čítají následující informace. Úspěšné architektury používají jen dva optimalizátory - sgd a adam. Všechny úspěšné neuronové sítě měly mimo výstupní vrstvy 2 až 3 další vrstvy.



Obrázek 3: Výsledky evolučního vývoje neuronové sítě.

Obrázek 4: Počty her potřebné k natrénování vítězné architektury neuronové sítě.

5.1 Ověření výsledků

Zdali se vítězná neuronová síť trénuje opravdu o tolik rychleji než neuronová síť použitá v experimentech [1], jsem ověřil pomocí skriptu *validator.py*. Síť jsem 40x natrénoval, viz obrázek 4, a průměrný počet her potřebných k natrénování je o dost větší než byl potřebný v genetickém algoritmu - konkrétně 225. To je ale pořád znatelně lepší výsledek než neuronové sítě v [1], která potřebovala v průměru 280 her.

6 Implementace

Projekt je implementovaný v jazyce Python 3. K implementaci projektu jsem využil kód, který jsem používal na evoluční vývoj neuronových sítí pro soutěž Numerai. Podle podobnosti kódu z 5. cvičení a mého kódu, který jsem znovu použil, je původní zdroj pravděpodobně stejný. Můj kód jsem upravil a přidal možnost evolučního vývoje každé vrstvy neuronové sítě. V originálním kódu měly všechny vrstvy stejný počet neuronů a stejnou aktivační funkci. Nyní může mít každá vrstva neuronové sítě odlišný počet neuronů a odlišnou aktivační funkci.

6.1 Spuštění

Pro spuštění je třeba nainstalovat Pythonovské knihovny Tensorflow, Logging, Argparse, Numpy, Gym.

6.1.1 Parametry spuštění evolučního algoritmu

Evoluční algoritmus se spouští souborem *main.py*, kterému je třeba předat parametry v tabulce 1. Soubor s parametry obsahuje ty, které jsou zmíněné v kapitole 4.1.

–nn-parameters FILE-PATH	cesta k souboru s parametry neuronové sítě ve formátu JSON
–generations COUNT	počet generací
–population SIZE	velikost populace
–environment NAME	jméno prostředí na kterém se bude provádět DQN

Tabulka 1: Parametry spuštění *main.py*.

7 Závěr

Experimenty ukázaly funkčnost genetického algoritmu při vývoji neuronové sítě využívané v algoritmu DQN. Vyvinutá síť dokonce předčila výsledky architektury v [1]. Projekt byl zajímavý a návrat k problematice mé bakalářské práce byl zpestřením semestru.

Reference

[1] Petr Buchal. Hraní her pomocí neuronových sítí. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2018.

[2] Petr Buchal. Hraní her pomocí neuronových sítí. Excel@Fit 2018, May 2018. https://goo.gl/FHbELw.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, December 2013. https://arxiv.org/pdf/1312.5602.pdf.