České vysoké učení technické v Praze

Fakulta elektrotechnická

Katedra počítačů

Diplomová práce

Učení hlubokých neuronových sítí v Mathematica

Martin Kerhart

Květen 2016

Vedoucí práce: Ing. Jan Drchal, Ph.D.

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačů

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Martin Kerhart**

Studijní program: Otevřená informatika
Obor: Umělá inteligence

Název tématu: **Učení hlubokých neuronových sítí v Mathematica**

Pokyny pro vypracování:

Nastudujte problematiku posilovaného učení (reinforcement learning) a učení hlubokých neuronových sítí (deep learning). Aplikujte tyto metody na tzv. obecné hraní her (General Game Playing): zaměřte se na přímé zpracování obrazové informace u 2D videoher. Pro experimenty použijte emulátor konzole Atari 2600. Vámi navržená metoda bude na základě co nejméně předzpracovaného obrazu hry vybírat nejvhodnější akci s cílem maximalizovat konečné skóre. Vytvořte knihovnu jazyka Wolfram Mathematica pro vyhodnocování a učení hlubokých neuronových sítí podporující akceleraci na GPU.

Seznam odborné literatury:

Mnih, V., Kavukcuoglu, K., Silver, D., Alex, G., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. NIPS Deep Learning Workshop 2013 (pp. 1–9).

Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2013). The Arcade Learning Environment : An Evaluation Platform for General Agents. Journal of Artificial Intelligence Research, 47, 253–279.

Sutton, Barto: Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning), 1998

Vedoucí: Ing. Jan Drchal, Ph.D.

Platnost zadání: do konce letního semestru 2015/2016

L.S.

doc. Ing. Filip Železný, Ph.D.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 26. 3. 2015

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 25. 5. 2016

Poděkování

Mé díky patří především vedoucímu práce Ing. Janu Drchalovi, Ph.D. za poskytnutý čas, rady a připomínky. Rodině děkuji za klid a prostor pro pohodlnou tvorbu.

Velice si vážím možnosti využít výpočetní a úložné zařízení sdružené pod Národní Gridovou Infrastrukturou MetaCentrum, poskytnuté programem „Projekty velkých infrastruktur pro výzkum, vývoj a inovace“ (CESNET LM2015042).

Abstrakt

Náplní této práce je obecné hraní her prostřednictvím umělých neuronových sítí. Experimenty jsou prováděné na ALE, simulátoru herní konzole Atari 2600. Problém je řešen kombinací autoenkodéru a rekurentní neuronové sítě. Konvoluční nebo plně propojený autoenkodér se na předem generovaných snímcích hry učí jejich reprezentaci v nižší dimenzi. Tento vektor příznaků je použit jako vjem herního agenta. Rekurentní neuronová síť – herní agent je učen evoluční strategií dle dosaženého skóre. Metoda je testována na vybraných hrách. Herní agent především díky mezerám v designu her překonává zkušené lidské hráče.

Klíčová slova: Obecné hraní her, Autoenkodér, Umělá neuronová síť, Konvoluční neuronová síť, Rekurentní neuronová síť, Evoluční strategie

Abstract

The content of this thesis is general game playing by means of artificial neural networks. Experiments are done on ALE, a simulator of the Atari 2600 game console. The problem is solved by a combination of autoencoder and recurrent neural network. A convolutional or fully connected autoencoder learns low dimensional representations of generated frames from the game. This feature vector is used as a perception of the game agent. The recurrent neural network – game agent is taught by the means of evolution strategy following gained score. The method is tested on selected games. The game agent is, thanks to design flaws of games, able to overcome experienced human players.

Keywords: General game playing, Autoencoder, Artificial neural network, Convolutional neural network, Recurrent neural network, Evolution strategies

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1
Úvod

Hraní her je jednou z mnoha úloh umělé inteligence. Od počátků informatiky existují snahy vyvinout počítače a programy schopné porazit člověka. Mnohé hry mají poměrně snadná pravidla, možná právě proto se zdá, že by v nich počítače, se svou enormní výpočetní silou, měly excelovat. Zdání klame, na pohled jednoduché hry skrývají často kombinatorickou explozi variant, tahů a rozhodnutí. Výpočet optimální strategie pak naráží na časovou nebo paměťovou bariéru. Kompletní řešení nebo optimalita herního postupu však není vždy nutná. Za pomoci specializovaných algoritmů a heuristik se daří porážet profesionální hráče šach, dámy a nedávno i go.

Náročnější varianta je obecné hraní her *General Game Playing* (GGP). Zde se předpokládá použití stejného, obecného systému pro hraní více her. Herní agent se musí přizpůsobit pravidlům každé hry, to zabraňuje použití vysoce specializovaných algoritmů. Všestranné algoritmy potřebují univerzální způsob definice herních pravidel. Za tímto účelem a pro pohodlné testovaní byl vyvinut např. jazyk *Game Definition Language* (GDL). Každoročně se v rámci AAAI konference pořádá GGP soutěž, kde spolu soupeří univerzální hráči v celé řadě her popsaných GDL.

(a) (b)

Obrázek 1: a) Konsole Atari 2600[1], b) Logo Atari Learning Environment - ALE

Zvláštní kapitolou v GGP jsou video hry hrané z „lidského“ pohledu – tedy jen na základě vizuální informace. Agent si musí z obrazu odvodit nejen pozici a význam objektů ale i pravidla a ovládání hry. Populární testovací platformou je herní konzole Atari 2600 (obr. 1) z 80. let. Hraje se ovšem na jejím emulátoru *The Arcade Learning Environment* (ALE)[1]. Atari nabízí pozoruhodné množství her s prostou grafikou v ro-

[1]Evan-Amos (Own work) [Public domain], via Wikimedia Commons

zlišení 210 × 160 pixelů. Notoricky známé hry jako Pong, Breakout či Space Invaders úspěšně hrají [2][3].

Tato práce je zaměřena na video GGP s využitím autoenkodéru co by extraktoru a rekurentní neuronové sítě jakožto kontroléru. Nejprve je předložen úvod do problematiky a teorie. Následně se krátce pojedná o dříve aplikovaných metodách hraní her na základě videa. Posléze je detailně rozebrána zvolená metoda této práce. Dále je uveden popis implementace nezbytných knihoven k zprovoznění vyvinutých skriptů a spouštění použitého softwaru. V neposlední řadě jsou předvedeny experimenty – výsledky naučených herních agentů jednotlivých her.

Kapitola 2
Teorie a terminologie

2.1 Neuronové sítě

Umělé neuronové sítě jsou inspirované biologickými neuronovými sítěmi. Jejich základní vlastností je schopnost učit se. Dnes jsou známé architektury schopné klasifikovat miliony obrázků do stovek tříd [4][5], segmentovat objekty v obrazu [6] nebo generovat uchu libé melodie [7].

Tradiční neuronová síť *Neural network* (NN) se skládá z neuronů řazených do vrstev. Neurony jsou propojeny jen mezi dvěma sousedními vrstvami, uvnitř vrstev spojeny nejsou. Analogie s biologickým neuronem je následující: Neuron na jedné straně přijímá signál z okolních neuronů vstupními hranami (dendrit), signál zpracuje (soma) a vyšle po výstupní hraně (axon) k dalšímu neuronu.

Matematický model neuronu je poměrně prostý. Každé vstupní hraně přísluší váha – parametr laděný učením. Výpočet aktivace (výstupu) neuronu k napojeného na neurony 0 až m je:

$$y_k = \varphi\left(\sum_{j=0}^{m} w_{kj} x_j\right), \tag{1}$$

kde w_{kj} značí váhu signálu x_j z neuronu j do k. Suma značí přenosovou a φ aktivační funkci. Typicky je φ sigmoida:

$$\varphi(x) = \frac{1}{1 + e^{-x}}, \tag{2}$$

mezi další aktivační funkce patří hyperbolický tangent, skoková či lineární funkce.

Obvykle se ke vstupním neuronům přidává jeden s konstantním výstupem $y_0 = 1$, v kombinaci s příslušnou vahou pak tvoří tzv. bias. Bias umožňuje posunutí oboru hodnot aktivační funkce nezávisle na vstupu sítě.

Pokud váhy celé jedné vrstvy seskupíme do matice $\mathbf{W}$, lze 1 zapsat jako:

$$\mathbf{y}_j = \varphi(\mathbf{W}_j \cdot \mathbf{x}_{j-1}^T). \tag{3}$$

Přičemž řádky $\mathbf{W}_j$ jsou váhy jednotlivých neuronů z vrstvy j a $\mathbf{x}_{j-1}$ je vektor signálů z vrstvy $j-1$. Výsledný vektor $\mathbf{y}_j$ obsahuje signály vrstvy j.

Vstupní vrstva je obvykle lineární, pouze distribuuje vstup do další (skryté) vrstvy. Poslední vrstva sítě se nazývá výstupní. Je prokázáno [8], že neuronovou síť s jednou a více skrytými nelineárními vrstvami je schopna aproximovat libovolnou funkci, za jistých podmínek. To z NN činí atraktivní nástroj k učení modelů tzv. černých skříněk.

2.1.1 Rekurentní neuronová síť

Obecně mohou být neurony propojeny v libovolné struktuře. Pokud se v grafu sítě nevyskytuje žádný cyklus, jedná se o tzv. dopřednou *feedforward* síť. Vyhodnocení každého vstupu je zcela nezávislé na ostatních. Opakem dopředné sítě je rekurentní neuronová síť *Recurrent neural network* (RNN). Ta využívá zacyklení signálu k udržení vnitřního stavu sítě, což umožňuje zpracovat závislou sekvenci vstupů. RNN tak může řídit auto [9] nebo skládat hudbu [7]. Výpočet aktivací se nijak neliší od 1 a lze snadno aplikovat 3. Vektor signálů $\mathbf{x}_{j-1}$ se vždy skládá z aktuálního vstupu a aktivací neuronů v předchozí iteraci.

2.1.2 Konvoluční neuronová síť

Klasické NN s plně propojenými vrstvami nejsou ideální pro zpracování obrazu. Kombinace obrázku s hranou o pár desítkách pixelů a první skrytou vrstvou byť jen s desítkami neuronů produkuje počet vah v řádech deseti tisíců. Takové množství parametrů vyžaduje velký objem trénovacích dat a tedy i zdlouhavé učení. Lineární vrstvy navíc zcela ignorují prostorovou korelaci pixelů v obrazu [10]. Každý skrytý neuron také hledí skrz své váhy na celý vstup. Globální přístup však není vhodný např. k detekci menších objektů na různých místech.

Inspirovány poznatky v biologii o oku, konvoluční sítě *Convolutional neural network* (CNN) zpracovávají vstup po malých lokálních oblastech [11]. Neurony nedostávají vstup z celé předchozí vrstvy, ale jen malou část danou velikostí konvolučního jádra. Těch bývá od jednotek po stovky [4] a jsou typicky výrazně menší než vstup. Navíc si data zachovávají 2D strukturu obrazu a to i ve skrytých vrstvách. Tento typ NN tak cílí na uvedené nedostatky plně propojených vrstev s obrazovým vstupem.

Výstupy neuronů z jedné vrstvy s více jádry (filtry) se dělí na kanály, každý kanál odpovídá konvoluci vstupu s příslušným filtrem. Filtry se sdílí v rámci celého kanálu, tím je dosaženo výrazné úspory parametrů k učení. Zároveň neurony získávají lokání pohled na vstup a podporuje se specializace filtrů na opakující se objekty (příznaky) napříč sadou dat. Konvoluční princip aplikace sdílených vah zvyšuje pravděpodobnost detekce objektů v ostrých datech i na jiných pozicích než v trénovací sadě.

Architektura CNN může nabývat rozličných tvarů a i na první pohled stejné struktury se mohou lišit počtem a velikostí jader. Typická klasifikační síť se skládá z řady konvolučních vrstev místy prostřídané poolingovou vrstvou (viz. níže). Ke konci je pár plně propojených vrstev zakončených ztrátovou vrstvou, podle které se počítá gradient. Velmi hluboké modely nejsou výjimkou [5], myšlenkou je hierarchické učení příznaků: Na vrcholu se učí komplexní struktury jako rysy obličeje, níže třeba jen pouhé hrany. Odtud pochází označení *Deep learning* a *Deep believe networks*.

K učení CNN se používá modifikovaná verze zpětné propagace *Backpropagation* na základě SGD viz. [12][10]. CNN jsou s úspěchem používány při klasifikaci obrazů [4][5], rozpoznávání tváří, dopravních značek nebo psaného textu [13]. Popularitu a schopnosti

CNN dosvědčuje jejich hojné nasazení v soutěži LSVRC[2].

2.1.2.1 Kalkulace CNN

Velikost výstupu konvoluční vrstvy je dána velikostí vstupu, filtru a kroku s jakým se ve vstupní matici posouvá. Vrstva se vstupem o dimenzi $\mathbf{d}$, s c jádry velikosti $\mathbf{k}$ produkuje výstup $c \times \mathbf{d}'$:

$$\mathbf{d}' = \frac{\mathbf{d} - \mathbf{k} + 2\mathbf{p}}{\mathbf{s}} + 1, \tag{4}$$

kde $\mathbf{p}$ značí okrajové obložení sloupci a řádky nul a $\mathbf{s}$ je posun konvolučního jádra. Výpočet aktivace jednoho kanálu g lze zapsat jako 2D konvoluci vstupu f a filtru h:

$$g(x,y) = \varphi\left(b_g + \sum_{j=1}^{k_1}\sum_{l=1}^{k_2} h(j,l) f\big(s_1(x-1)+j, s_2(y-1)+l\big)\right), \tag{5}$$

Souřadnice x, y postupně nabývají hodnot $1, \ldots, d'_1$ resp. $1, \ldots, d'_2$ přičemž $\mathbf{d}' = \{d'_1, d'_2\}$ a plní tak celý kanál. Obdobně j, l prochází masku přes $\mathbf{k} = \{k_1, k_2\}$. Kanálu g přísluší bias b_g a s_1, s_2 jsou složky $\mathbf{s}$.

2.1.2.2 Pooling

Vedle neuronových vrstev se v CNN používají i tzv. *Pooling* vrstvy. Jejich účelem je rapidně snížit dimenzi vstupu při zachování hlavní informace. Pooling operuje se vstupem skrze poolovací jádra stejným způsobem jako konvoluční vrstva postupně aplikuje masku na celou vstupní matici. Jádro vybere oblast, z které poolovací funkce vyrobí jednu hodnotu výstupu. Mezi běžné varianty patří výpočet maxima nebo průměru z vybrané oblasti. Vžilo se pro ně označení *max-pooling* a *avg-pooling*.

Pro pooling také platí 4, pokud se uvažuje jen jeden filtr. Krok posunu jádra se zpravidla volí roven jeho velikosti. Nedochází tak k opakování stejných hodnot z překryvu sousedních oblastí a dimenze vstupu klesá nepřímo úměrně velikosti jádra. Pooling s maskou 2×2 a krokem 2×2 sníží velikost vstupu na čtvrtinu.

2.1.2.3 ReLU

Při klasifikaci rozsáhlých obrazových dat se aktivační funkce *Rectified linear unit* (ReLU) ukázala být výhodnější než sigmoida [4][14]. ReLU je definován jako:

$$\varphi(x) = max(0, x). \tag{6}$$

2.1.3 Dekonvoluční neuronová síť

Dekonvoluční neuronová vrstva provádí převrácenou konvoluci [15]. Výstup má větší dimenzi než vstup. Při stejných parametrech na sebe napojených konvoluční a dekonvoluční vrstvy bude výsledek stejné velikosti jako vstup. To umožňuje vytvářet 2D

[2]Large Scale Visual Recognition Challenge, většina participantů v loňském roce (`http://image-net.org/challenges/LSVRC/2015/results`) CNN využila.

autoenkodéry 2.2.0.1 a promítat příznaky z nízké úrovně hierarchie do vyšší dimenze. Dekonvoluční vrstvy nachází uplatnění při zkoumání chování CNN [16], segmentaci obrazu [17] či trénování filtrů pro klasifikační síťe [18].

Konvoluce lineární kombinací masky a oblasti vstupu produkuje jednu hodnotu výstupu. Dekonvoluce naopak vynásobením jádra jednou hodnotou vstupu vytváří příspěvky do oblasti výstupu. Při zachování značení 5 (f je nyní výstup a g vstup) se dekonvoluce jednoho vstupního kanálu g a filtru h spočte:

$$f(x,y) = \varphi\left(b_g + \sum_{j=1}^{d_1'}\sum_{l=1}^{d_2'} g(j,l)v(i,j,x,y)\right). \tag{7}$$

Funkce $v(i,j,x,y)$ určuje, zda hodnota vstupu na pozici j,l přispívá do výstupu na souřadnicích x,y:

$$v(i,j,x,y) = \begin{cases} h\big(x \bmod s_1(j-1), y \bmod s_2(l-1)\big) & s_1(j-1) < x < s_1 j \wedge \\ & s_2(l-1) < y < s_2 l \\ 0 & jinak. \end{cases} \tag{8}$$

2.1.3.1 Unpooling

Pokud CNN obsahuje pooling a zrcadlově převrácená síť z dekonvolučních vrstev má používat stejné parametry kernelů, je nutno zvrátit efekt poolingu. K tomu slouží tzv. unpooling. Analogicky k dekonvoluci je to opak pooling vrstvy.

Během poolingu je možno poznamenat pozice, kde se vyskytly maxima. Tyto tzv. *switches* [16][17][18] lze následně použít k výrazně přesnějšímu unpoolingu. Ve výstupu jsou umístěny jen maxima na svých původních pozicích, zbytek je ponechán nulový.

Bez přepínačů se celý výstup plní stejně jako v 7, ale filtr je jen jeden a plný jedniček. Pokud je krok filtru roven jeho velikosti, max-unpooling celou oblast masky ve výstupu zaplní jednou hodnotou. Bez ohledu na použití switches, výstup průměrového unpoolingu je dělen velikostí kernelu.

2.1.4 Učení neuronových sítí

Neuronové sítě se nejčastěji učí zpětnou propagací *Backpropagation* (BP). Jedná se o techniku učení se supervizorem. Je tedy potřeba znát, jaký má být výstup sítě z trénovacích dat a síť se ho snaží aproximovat. Učení probíhá postupným upravováním parametrů (vah) tak, aby výstup co nejvíce odpovídal očekávání. Rozdíl mezi nimi měří ztrátová *loss* funkce. Počítá se po každém dopředném běhu – po propagaci vstupu celou sítí až k výstupní vrstvě.

Princip BP spočívá ve výpočtu gradientu chyby sítě vzhledem ke každé její váze. Nejprve se určí chyba každého neuronu propagací celkové chyby z výstupní vrstvy zpět na vstup. Z těchto chyb se následně určí derivace *loss* podle všech vah. Nyní se gradientovým sestupem aktualizují váhy. Jednoduše se odečte gradient vážený rychlostí učení *learning rate*.

Síť je možno učit po jednotlivých pozorováních nebo po skupinách. Obvykle se volí druhá varianta z důvodu paralelizace. Celou podsadu dat jde paralelně vyhodnotit na fixních vahách a chyby zkombinovat. Aktualizace vah je pak provedena jen jednou vzhledem k této celkové chybě. Existují různé varianty gradientového sestupu jako *Stochastic Gradient Descent* (SGD), *AdaGrad*, *AdaDelta* nebo *RMSProp*. Využívají různého vážení gradientu z předchozích iterací nebo adaptivní *learning rate*.

2.2 Autoenkodér

Tradiční kompresní algoritmy přímočaře vyhledávají v datech redundanci, korelaci blízko ležících hodnot či rovnou vynechají méně výrazné detaily [19]. Autoenkodér (AE) je neuronová síť s jednou skrytou vrstvou, hledající reprezentaci vstupních dat v nižší dimenzi[3]. První část (enkodér) vstupní data komprimuje. Druhá polovina (dekodér) navazuje na enkodér a komprimovaná data převádí zas zpět do původní dimenze [20]. Ztrátová funkce pak dekódovaný výstup porovná s originálem. Cílem je naučit NN takové váhy, aby byl výstup co nejvěrnější originálu.

Rozepsáním výpočtu aktivace 3 NN s explicitním bias vektorem $\mathbf{b}$ se dostane rovnice pro enkodér:

$$\mathbf{y} = \varphi(\mathbf{W} \cdot \mathbf{x}^T + \mathbf{b}^T), \tag{9}$$

dekodér je obdobně:

$$\mathbf{z} = \varphi(\mathbf{W}' \cdot \mathbf{y}^T + \mathbf{b}'^T). \tag{10}$$

Kde $\mathbf{x}$ je vstup AE, $\mathbf{y}$ je kód a $\mathbf{z}$ rekonstrukce originálu. Dále $\mathbf{W}$ je váhová matice enkodéru a $\mathbf{W}'$ dekodéru, stejně tak bias $\mathbf{b}'$. A φ značí aktivační funkci, např. sigmoidu. Váhy autoenkodéru lze „svázat“ jejich transponováním:

$$\mathbf{W}' = \mathbf{W}^T, \tag{11}$$

stupeň volnosti se tak sníží na polovinu. Obratem je učení efektivnější a vliv na chybu AE zanedbatelný [21]. Případně lze svázání využít jen k předučení a následně váhy samostatně doladit.

Autoenkodér se učí minimalizovat ztrátovou funkci E – rekonstrukční chybu sítě. Podle charakteru dat se nejčastěji volí průměr kvadrátů rozdílu vzorků (MSE) pro reálná data:

$$E_{\text{MSE}}(\mathcal{X}, \mathcal{Z}) = \frac{1}{2n} \sum_{i=1}^{n} (\mathbf{x}_i - \mathbf{y}_i)^2, \tag{12}$$

a *Sigmoid cross-entropy* (SCE) pro binární data:

$$E_{\text{SCE}}(\mathcal{X}, \mathcal{Z}) = -\sum_{i=1}^{n} \big(\mathbf{x}_i \log \mathbf{z}_i + (1 - \mathbf{x}_i) \log (1 - \mathbf{z}_i)\big). \tag{13}$$

Vektor $\mathbf{x}_i \in \mathcal{X}$ značí originální data a $\mathbf{z}_i \in \mathcal{Z}$ jejich rekonstrukci. Množina $\mathcal{Z}$ obsahuje n odpovídajících rekonstrukcí n pozorování v $\mathcal{X}$.

[3] Autoenkodér může kódovat vstup i do vyšší dimenze, tzv. *overcomplete* AE zde nemají význam, proto se nadále neuvažují.

2.2.0.1 CNN jako AE

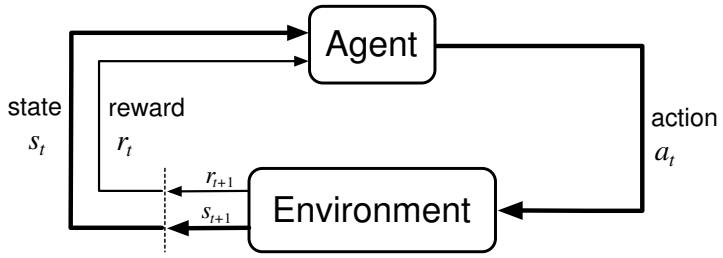
Podle stejného principu lze sestavit i konvoluční autoenkodér [22]. Enkodér tvoří CNN s případnou pooling vrstvou a na druhé straně je dekodér z dekonvoluční a možné unpooling vrstvy. Aby vstupní a výstupní rozměry seděly, musí být parametry odpovídajících si vrstev stejné.

2.2.0.2 Vrstvený autoenkodér

Zatím byl AE popisován jen s jednou skrytou vrstvou. Nahrazením této vrstvy dalším AE vznikne tzv. *stacked* autoenkodér. Prostým zřetězením AE lze postupně snižovat dimenzi dat stále více [23]. Každý stupeň lze přitom učit zvlášť: naučit první, váhy zafixovat, přidat další stupeň a zas učit ten. Posléze je vhodné naráz doladit váhy celé sítě. Postupné učení je efektivnější než hromadné všech stupňů naráz. Také skýtá větší variabilitu, je možno na nižších úrovních zkoušet různé velikosti oproti jedné fixní konfiguraci.

2.3 Posilované učení

Posilované (zpětnovazebné) učení *Reinforcement leaarning* (RL) je technika učení bez učitele. Používá se na trénování systémů interagující s prostředím, jehož model je neznámý nebo příliš komplexní na simulaci. Typicky se systém, agent, snaží naučit nějakou rozhodující strategii, sekvenci akcí, maximalizující odměnu distribuovanou samotným prostředím. Bez možnosti simulace na modelu, musí agent trénovat přímou konfrontací s okolím: provádět akce, akceptovat odměnu a učit se z dřívějších rozhodnutí. Interakci agenta s prostředím vystihuje obr. 2.3.



Obrázek 2: Vzájemná interakce agenta a prostředí, převzato z [24]

RL úlohu lze dle [24] popsat následujícím způsobem: Prostředí ovládané agentem se v čase $t \in \langle 1, T \rangle$ nachází ve stavu $s_t \in \mathcal{S}$, kde $\mathcal{S}$ je množina všech stavů a T finální časový okamžik. Agent na základě s_t vybere akci $a_t \in \mathcal{A}(s_t)$, zde $\mathcal{A}(s_t)$ značí množinu právě proveditelných akcí ve stavu s_t. Po aplikaci a_t se prostředí přesune do stavu s_{t+1} a obdaří agenta odměnou $r_{t+1} \in \mathbb{R}$. Agent se rozhoduje podle taktiky $\pi(a|s)$, což je pravděpodobnost výběru akce $a = a_t$ ve stavu $s = s_t$.

Úloha může být epizodická nebo nekonečná, v obou případech agent cílí za maximalizací celkové odměny. Pro epizodu to může být prostá suma:

$$R_T = \sum_{t=1}^{T} r_t, \tag{14}$$

jinak je třeba přistoupit ke slevování starších odměn koeficientem $0 < \gamma < 1$:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}, \tag{15}$$

Vzhledem k π lze definovat užitkové funkce $v_\pi(s)$ a $q_\pi(s, a)$, které udávají celkové očekávané odměny při sledování π od stavu s a dál, $q_\pi(s, a)$ má navíc danou akci a pro výchozí stav s. Označované jsou jako *state-value* a *action-value* pro taktiku π. Funkce $v_\pi(s)$ a $q_\pi(s, a)$ lze aproximovat ze zkušeností agenta nabytých konfrontací s prostředím, čehož využívá většina RL technik. Odhad $q_\pi(s, a)$ se značí $Q_\pi(s, a)$.

Důležitým aspektem RL je vyvážení průzkumu a využití již získaných znalostí, tedy otázka aplikace nejslibnější akce nebo nějaké jiné. Minimální nebo žádný průzkum by vedl strategii do lokálního maxima. Naopak příliš častá explorace by učení výrazně zpomalila. Toto dilema řeší ϵ-greedy strategie: S pravděpodobností $1 - \epsilon$ se volí akce maximalizující Q a s pravděpodobností ϵ se použije náhodná.

2.3.1 Q-learning

Q-learning je jedna z nejvýznamnějších RL technik. Iterativně aproximuje *action-value* funkci a postupně konverguje k optimální taktice. Totiž pokud je zaručená stálá dostupnost všech stavů. Jeden krok, aktualizace Q po přechodu ze stavu s_t do s_{t+1} skrz akci a_t je definován takto:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\big(r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\big). \tag{16}$$

Přičemž $\max_a$ vybírá odhad optimální užitné funkce z nového stavu a α je rychlost učení. Pro neznámé stavy vrací Q fixní hodnotu (např. 0). Jedině pro terminální stavy se Q neaktualizuje.

Podle 16 je možno Q-learning implementovat tabulkou mapující s a a na Q. Ale např. v [2] s úspěchem použili CNN učenou SGD, viz. 3.3.

2.3.2 Evoluční algoritmy

Genetické programování a evoluce je kategorie sama o sobě, nicméně lze jimi efektně řešit mnohé RL úlohy. Nepracují s jednotlivými přechody stavů ani s užitkovými funkcemi. Evoluci podobnými technikami hledají rovnou celé agenty či strategie. Hledání probíhá na bázi generací, populací a jedinců. Jedinec reprezentuje jedno řešení, jeho úspěšnost hodnotí fitness funkce. Evoluce probíhá iterativně po generacích, nová generace vzniká z předchozí populace selekcí, mutací a křížením vybraných jedinců podle fitness.

2.3.2.1 Exponential Natural Evolution Strategies

Exponential Natural Evolution Strategies (xNES) patří do rodiny evolučních strategií (ES). Ty se od ostatních evolučních metod odlišují uplatňováním především selekce a mutace. Ta je prováděna přičtením vektoru náhodných hodnot z normálních rozdělení, jejichž parametry jsou součástí kódu jedince. Celá evoluce je tak řízena vlastní evolucí. Typickým příkladem evoluční strategie je populace dvou jedinců – rodiče a potomka. Potomek vzniká mutací rodiče tak dlouho, dokud nemá lepší fitness. V ten moment nahradí rodiče a celý proces se opakuje.

Evoluční strategie xNES stejně jako CMA-ES[4] či NES používají k reprezentaci celé populace vícerozměrné normální rozdělení. Jedinci jsou generovaní vzorkováním, ohodnoceni a seřazeni podle fitness. Následně jsou upraveny parametry rozdělení tak, aby se zvýšila pravděpodobnost výběru právě úspěšných vzorků.

xNES je vylepšením *Natural Evolution Strategies* (NES), evoluční kroky počítají podle odhadu přirozeného gradientu fitness funkce vzorkované populací [25]. Fitness je maximalizována, takže se sleduje vzestup gradientu. Použití přirozeného *natural* gradientu zajišťuje sledování nejpříkřejšího směru bez ohledu na jeho parametrizaci. Dále zabraňuje oscilaci a předčasné konvergenci oproti klasickému gradientu. xNES zavádí exponenciální parametrizaci a zefektivňuje výpočet přirozeného gradientu [26].

[4] *Covariance Matrix Adaptation Evolution Strategy.*

Kapitola 3
Hraní video her

3.1 Extrakce příznaků obrazu

Ve strojovém učení platí, čím větší je dimensionalita pozorování, tím bude učení delší a náročnější. To je zřejmé, neboť je jednoduše potřeba více výpočetních prostředků. Zásadní roli zde hraje způsob reprezentace dat. Obraz je typicky tříkanálový v poslední době až v 4K rozlišení. Znát hodnotu každého pixelu je přitom většinou nepodstatné. Obvykle stačí rozpoznat základní aspekty objektů jako: tvar, pozici, barvu a případně rychlost, aby se hra dala hrát a vyhrát. Reprezentace snímku v této podobě má samozřejmě mnohonásobně menší dimenzi. A vyjma grafických (nepodstatných) detailů nepostrádá žádnou informaci originálního obrazu.

Například aktuální obraz hry Pong všeho všudy vystihuje šest souřadnic, nepočítáme-li velikost objektů a nynější skóre. Tomu odpovídá obraz 210×160 px (bez skóre pak výřez 160×160 px). Učit agenty přímo na vizuálních datech je určitě možné, ale výhodnější je obraz předzpracovat – extrahovat příznaky. Při pohledu na Pong se přímo nabízí ručně zkonstruovat speciální mechanismus detekce objektů a reprezentace celého stavu. Takovýto přístup však postrádá obecnost a u některých her by byl těžko aplikovatelný. Nehledě na časovou náročnost analýzy her, mnohdy není ani zcela jasné jak u konkrétní hry univerzálně popsat každý stav. Nezbývá než pro extrakci příznaků použít vhodný obecný algoritmus.

3.2 Herní agent

GGP vyžaduje univerzální herní agenty připravené takřka na cokoliv. Všestrannost má své meze, zde se jí myslí stejná struktura a vlastnosti agenta napříč širokým spektrem her, bez bližší specializace na konkrétní hru. Pokud je agent učen, musí nejprve na dané hře trénovat než dosáhne uspokojivých výsledků.

V následující sekci jsou zmíněny některé dřívější práce úspěšně hrající video hry. Povětšinou aplikují RL techniky v kombinaci s nějakou variantou předzpracování obrazu.

3.3 Související práce

ALE vzniklo za účelem testování různých herních agentů a algoritmů. Sami autoři v [1] uvádí hned několik možných řešení. Zaměřují se na metodu posilovaného učení SARSA(λ) a porovnávají různé varianty předzpracování obrazu a extrahování příznaků.

Také předvádí aplikaci klasického plánování – prohledávání stavového prostoru dle akcí a skóre. Vyhledávací strom je budován postupnou evaluací odlišných akcí ve stejném stavu. Stav ALE je dán obsahem emulované RAM Atari 2600 a je možno ho uložit a znovu načíst. Většinu testovaných her vyhrál prohledávací algoritmus UCT, který preferuje expanzi slibnějších uzlů stromu. Neporovnává se jen aktuální hodnota uzlů, ale i jeho budoucí přínos na základě aplikace krátké sekvence náhodných akcí.

Prohledávání zaostává za naučenými agenty ve hrách, kde je skóre udělováno jen zřídka a je nutno plánovat více do předu, než je algoritmu povoleno. Velikost stromu stavů je exponenciální vzhledem k jeho hloubce a při šedesáti snímcích za sekundu a až osmnácti akcemi jsou nároky na paměť enormní.

Naučit řídícího agenta pro 3D závodní hru TORCS[5] si vzali za cíl v [9]. Použili kombinaci konvoluční neuronové sítě (extraktor příznaků), rekurentní neuronové sítě (agent) a RL techniky genetické evoluce pro učení agenta.

Hra byla spouštěna v rozlišení 64×64px. Obraz byl převeden do stupňů šedi konverzí RGB barevného prostoru do HSV a ponecháním jen saturace (S komponenta). Na takto upravených snímcích učili CNN klasickým SGD se ztrátovou funkcí v podobě součtu minima a průměru vzájemných euklidovských vzdáleností všech příznaků z aktuální podsady dat. Síť se tedy snaží zakódovat každý snímek co možná nejodlišněji od všech ostatních. Použitá CNN síť se skládá ze tří skupin konvoluční a poolingové vrstvy. Na konci jsou pak dvě plně propojené vrstvy neuronů se sigmoidou. Počáteční obraz 64×64 resp. $64 \times 64 \times 3$ je zredukován na pouhé tři příznaky, podle nichž agent určil akci – brzda, plyn a zatočení vlevo či vpravo.

Agent, řidič nepřekonal dostupné UI v TORCS, ale testovanou trať zvládl projet bez karambolu. Nutno podotknout, že soupeři měli k dipozici komplexní vnitřní popis stavu auta jako je rychlost či vzdálenost od bariér vozovky.

Podobně jako [1], se i [2] zabývá RL hraní Atari 2600 her na ALE. Místo předzpracování obrazu závislého na malém počtu barev a jednoduché grafice Atari 2600, pracují jen s lehce oříznutými černobílými snímky. Herní agenty učí Q-learning algoritmem, Q-value je počítána CNN ze vstupního obrazu pro každou akci naráz. Tyto sítě označují jako *Deep Q-Networks* (DQN) a v uvedených hrách překonávají RL techniku [1].

Stav hry a vstup DQN jsou čtyři poslední snímky hry, síť tak může reagovat podle kontextu nejbližší historie. DQN každé hry sestávala ze tří konvolučních vrstev s ReLU neurony a dvěma plně propojenými vrstvami: jedné též s ReLU a druhé lineární. Velikost poslední vrstvy se lišila pro každou kru podle počtu platných akcí. Výstupem sítě je očekávané skóre při aplikaci dané akce v aktuálním stavu.

DQN se jednoduše řečeno učí ve dvou krocích: odehraje se několik her, postup je rozdělen a zaznamenán v podobě čtveřic (výchozí stav, provedená akce, následující stav a obdržené skóre). Pokračuje se trénováním vah CNN klasickým SGD, ztrátovou funkcí je rozdíl pozorovaného skóre vybrané čtveřice a jeho odhadu dle DQN. Akce jsou voleny ϵ-greedy strategií, zajišťující exploraci hry.

[5] The Open Racing Car Simulator, `http://torcs.sourceforge.net/`

Na [2] přímo navazuje [3] vylepšeným učícím algoritmem DQN. Nově CNN přidali větší počet filtrů a jednu konvoluční vrstvu navíc. Místo černobílých snímků použili luminiscenci (Y kanál z YUV barevného prostoru) z maxima odpovídajících si pixelů dvou následných snímků. Tak předchází problémům s blikajícími objekty v některých hrách. Ve většině z 49 testovaných her překonávají RL agenta z [1] a momentálně se jedná o nejúspěšnější GGP postup hraní Atari 2600 her v reálném čase.

V další práci [27] hrající na ALE si přímo kladli za cíl překonat právě zmíněné DQN. Zkombinovali UCT plánovač a Q-learning s CNN. Myšlenka je nechat UCT generovat efektní herní strategie a podle nich učit CNN jako v DQN. UCT stále překonává RL techniky hraní, ale není schopno hrát v reálném čase. Výběr jedné akce trval UCT v [1] přibližně 15 s, oproti tomu jeden průchod zde použitou CNN se pohyboval v řádu 10^{-4} s. Skutečně se podařilo skloubit výhody obou přístupů a CNN učená podle dat UCT překonala DQN ve všech sedmi hrách [2].

Zde použitá CNN byla stejné struktury jako v DQN, ale měla tanh neurony místo ReLU. Obraz byl též oříznut a převeden na stupně šedi. K tomu navíc po odečtení průměrných hodnot pixelů byly celé snímky přeškálovaly na interval $\langle -1, 1 \rangle$. Otestovali tři varianty CNN. Dvě se učily čistě na datech UCT, první byla identická s DQN (produkovala Q-value), výstupem druhé byla akce zvolená UCT plánovačem. Třetí byla předučena jen na čtvrtině čistě UCT dat. Zbylé tři čtvrtiny byly natřikrát generovány také UCT, ale ze stavů navštívených rozhodnutím CNN. Síť byla doučována vždy po čtvrtině dat. Nejlepších výsledků dosáhla varianta učená kombinovanými daty.

Kapitola 4
Software

Implementace algoritmů může být zábava i cvičení, optimalizace specializovaných výpočtů však není nic lehkého. Ve strojovém učení hraje doba exekuce zvlášť významnou roli, každá ušetřená instrukce ve výsledku znamená rychlejší a kvalitnější učení. Použití hotového a rychlého software šetří výpočetní čas a umožňuje zaměřit se na experimenty.

Jedním z úkolů práce bylo vybrat vhodnou knihovnu pro akcelerované výpočty NN na grafické kartě (GPU) a propojit ji s matematickým programem Mathematica. Po krátké analýze dostupných řešení padla volba na Caffe viz. 4.1. Dalším daným software je již zmíněný Atari 2600 simulátor ALE. Mathematica podporuje dynamické načítání C++ knihoven, bylo tedy zapotřebí naprogramovat knihovny zprostředkovávající komunikace mezi Caffe, ALE a Mathematica.

4.1 Caffe

Caffe je otevřené rozhraní implementující akcelerované výpočty umělých neuronových sítí na grafických kartách [28]. Caffe bylo vybráno pro svou velkou popularitu a rychlost s jakou je vyvíjeno a rozšiřováno o nové funkce. Důraz je kladen na modularitu a všestrannost. Základním stavebním prvkem jsou pojmenované vrstvy a datové bloby. Definicí vstupních a výstupním blobů každé vrstvě, lze sestavit takřka libovolný DAG. Dokonce i aktivace neuronů počítají samostatné vrstvy pro dosažení maximální variability sítí. Definovat a trénovat sítě lze buď přímo přes Protocol Buffers soubory nebo prostřednictvím Python a Matlab rozhraní.

Nevýhodou Caffe je implementace v CUDA a tedy nutnost grafických karet Nvidia. Výpočty lze ovšem provádět i čistě na CPU, ale přijde se tak o značnou část výkonu.

4.2 Implementace

Zde jsou popsány implementované knihovny pro dynamické načítání v Matematica. Je využita technologie *LibraryLink*[6], jejíž hlavní výhoda je sdílení paměti mezi knihovnou a Mathematica. Absence kopírování či konverze dat výrazně zvyšuje rychlost komunikace.

4.2.1 CaffeLink

Za účelem snadné komunikace s Caffe byla vyvinuta knihovna CaffeLink. Již tak výkonný a univerzální nástroj Mathematica je rázem rozšířen o skvělou implementaci neuronových

[6] `http://reference.wolfram.com/language/guide/LibraryLink.html`

sítí. CaffeLink vedle učení a testování modelů, umožňuje přímý přístup k datům jednotlivých vrstev sítě. Po evaluaci vstupu tak lze sledovat jak se ve vrstvách postupně transformuje až na výstup. Dále je možno měnit naučené váhy sítě a kombinovat tak různé modely. Specialitkou pak může být např. napojení web kamery na klasifikační síť a v reálném čase sledovat odhad sítě a obraz kamery.

Caffelink byl publikován [29] na mezinárodním sympoziu IMS2015[7] v Praze a kód je společně s ukázkami použití veřejně dostupný na `https://github.com/Seilim/CaffeLink`.

4.2.2 ALEPipeLink

Knihovna ALEPipeLink zajišťuje komunikaci mezi ALE a Mathematica. Místo klasického nalinkování ALE jako knihovny je použito pojmenovaných rour. Ty umožňují jedním procesem spravovat více instancí ALE naráz, což s linkovanou knihovnou dost dobře nelze. Paralelizace ALE je velmi důležitý aspekt pro evaluaci celých generací herních agentů při jejich vývoji.

Zde je také provedena úprava obrazových dat do požadovaného formátu dle 5.1. Matematica již dostane jen ukazatel do paměti a může ho buď předat autoenkodéru k zakódování pro agenta nebo data uloží do vytvářené sady snímků. Obratem ALEPipeLink předává akci, již se agent rozhodl vykonat. Dále jsou zde vyřízeny dotazy na stav hry jako: počet snímků, skóre, konec apod.

[7] The International Mathematica Symposium, `http://www.ims2015.net`

Kapitola 5
Experimenty

Inspirován především [30] a [9] rozhodl jsem se na GPP aplikovat autoenkodér v kombinaci s RNN. Autoenkodér má za úkol extrahovat nízko dimenzionální příznaky z obrazu 8-bitových her. Zkoumá se zde, zda takto automaticky naučená reprezentace obrazu může posloužit jako podnět agenta při hraní her. Agent je realizován malou RNN učenou evoluční strategií dle dosaženého skóre ve hře.

Většina potřebné teorie je vysvětlena ve druhé kapitole. Zde jsou uvedeny provedené experimenty. Zejména se jedná o návrh a testování architektur autoenkodérů. Zkoumány byly plně propojené a konvoluční autoenkodéry. Na základě experimentů je vybrán formát vstupních dat a metoda učení. Schopnosti naučených AE jsou předvedeny na vzorcích dat jakož i na grafech průběhu učení.

Adekvátnost autoenkodérů může potvrdit až úspěšně naučený a hrající agent. Takový agent musí dosáhnout lepšího skóre než náhodně generovaná sekvence akcí. Tím se dokáže nejen použitelnost autoenkodérem produkovaných příznaků, ale i schopnost RNN osvojit si je a řídit se jimi.

Rozsah provedených testů a zkoušené metody byly koncipovány s ohledem na použitou knihovnu akcelerovaných výpočtů neuronových sítí, Caffe. To se týká především první fáze. Ta spočívala v určení finální podoby dat, výběru řešitele učení a použité ztrátové funkce. Většina výpočtů proběhla na počítačových clustrech spravovných Virtuální organizací MetaCentrum.

5.1 Příprava dat

Základ architektur AE se odvíjí od velikosti vstupních dat. Všechny testy byly provedeny na snímcích hry Pong. Originální obraz 240×160 px byl oříznut na 160×160 px zachycujících vlastní herní plochu bez skóre a statického ohraničení. Dále byl obraz zmenšen na polovinu lineárním vzorkováním. Každá sada snímků byla rozdělena v poměru $7 : 3$ na trénovací a testovací množinu.

Trénovací data byla shromážděna z her náhodného hráče. Ten ovšem není moc úspěšný a jen málokdy míček vůbec trefí. Většina snímků se proto sestávala ze stále opakující se rozehrávky, kde míček letí po stejné dráze. Problém byl částečně vyřešen ukládáním pouze unikátních snímků a to jen z her, v nichž hráč zvládl dát alespoň jeden gól. Tak se vždy zaznamená i nějaká zajímavá situace z pozdější části hry a autoenkodér má šanci naučit se více obecný model.

V první fázi testování prošel obraz ještě speciální úpravou pro rychlejší trénink. Samotný míček byl dilatován a celý obraz opět zmenšen na čtvrtinu: 40×40 px. Dilatace míčku byla nutná, aby se místy zcela neztrácel vlivem podvzorkování. Obrazový výstup ALE je kódovaný v indexech RGB palety o 128 barvách. Je tedy nasnadě, že

vhodná monochromatická reprezentace může elegantně snížit dimenzi při nulové ztrátě informací. Nabízí se hned několik variant. Zde jsou uvažovány tři: binarizace, převedení do stupňů šedi průměrem z RGB [2] a vytažení S kanálu z HSB prostoru barev [9]. Binarizace je provedena nastavením pozadí na nulu (černá) a zbytek (objekty hry) přepsán na jedna (bílá). Binarizace je přirozeně ztrátová a v některých hrách přímo nepoužitelná, ale v jiných může být velice efektivní. Tato speciální datová sada čítala devět tisíc snímků a varianty jsou referovány jako gray (černobílá), bin. (binarizovaná) a HSB (pouze S kanál).

5.2 Varianty učení

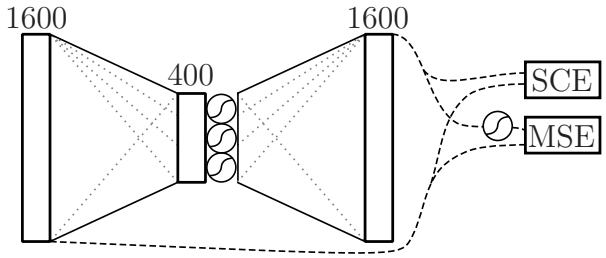
Aktualizaci vah, vlastní učení je v Caffe realizováno jedním z šesti řešitelů: *Stochastic Gradient Descent* (SGD), *AdaDelta*, *Adaptive Gradient* (AdaGrad), *Adam*, *Nesterov's Accelerated Gradient* (Nesterov) a *RMSprop*. Všechny metody jsou založené na sestupu gradientem, bližší informace v [31].

Řešitel aktualizaci provádí na základě ztrátové funkce. Caffe nabící dvě vhodné varianty pro autoenkodér: *Sigmoid Cross-Entropy* (SCE) a *Mean Squared Error* (MSE).

Na speciální zmenšené sadě dat byly otestovány všechny kombinace tří reprezentací dat, dvou rekonstrukčních chyb a šesti řešitelů. SCE byla pro svou charakteristiku použita jen u binarizovaných snímků.

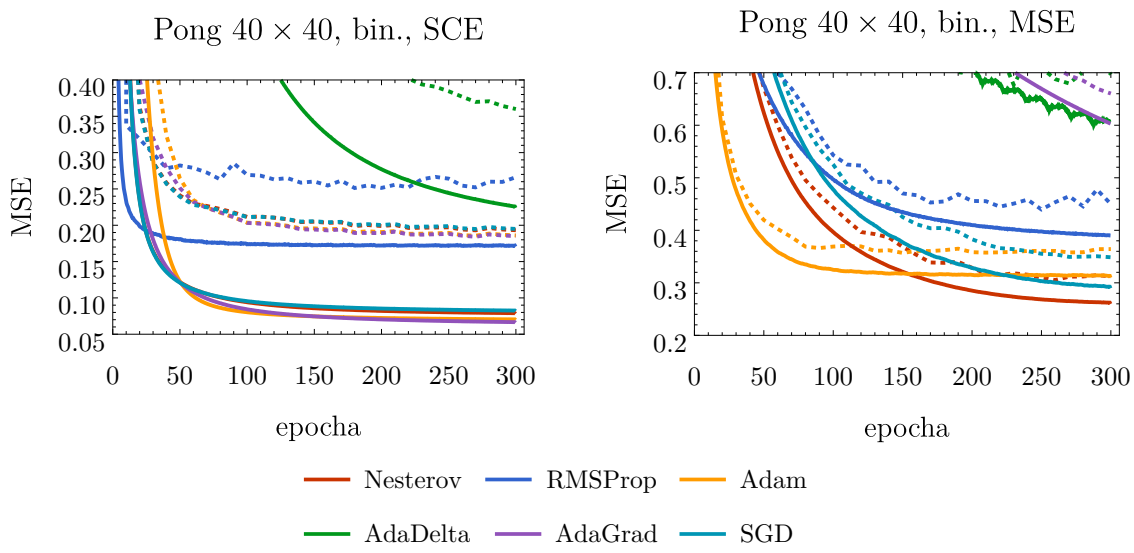
5.3 Plně propojený autoenkodér

V první fázi byl testován jeden stupeň AE s rozměry $1600 \rightarrow 400 \rightarrow 1600$ hodnot pro vstupní, skrytou a výstupní vrstvu. Vstupní data byly snímky Pongu ve speciální 40×40 px podobě. Neurony skryté vrstvy měly aktivační funkci sigmoidu a výstupní vrstva byla lineární. Výpočet SCE sigmoidu zahrnuje, proto pro porovnání výsledků s MSE verzí je přidána i podvrstva sigmoidy. Schéma na obr. 3 obě situace znázorňuje. Přidaná MSE vrstva se však na učení nepodílí, je jen pro pozorování. Modulární systém tvorby NN v Caffe si s touto situací lehce poradí.

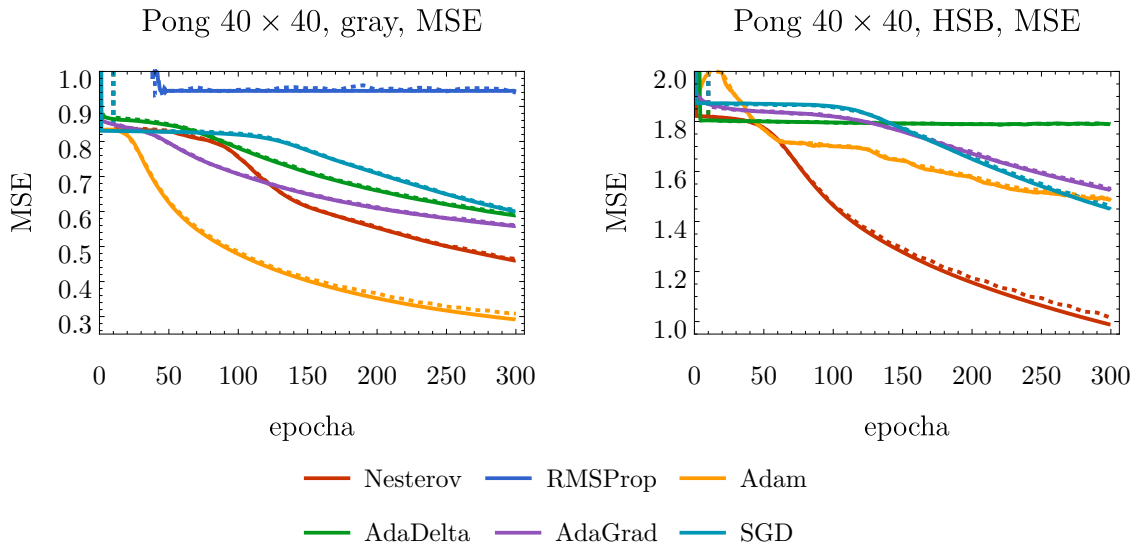


Obrázek 3: Schéma AE pro Pong 40×40, řešení dvou rekonstrukčních chyb.

Grafy na obrázcích 4 až 5 porovnávají průběh učení šesti řešitelů vždy na jednom formátu dat dle SCE nebo MSE. Každá konfigurace byla spuštěna pětkrát, grafy zobrazují průměr ze všech běhů. Jelikož učení probíhá na skupinkách třiceti dvou snímků,



Obrázek 4: Srovnání průběhu MSE při učení dle SCE a MSE různými řešiteli. Data: Pong 40 × 40, binarizována. Redukce AE: 1600 → 400 hodnot.



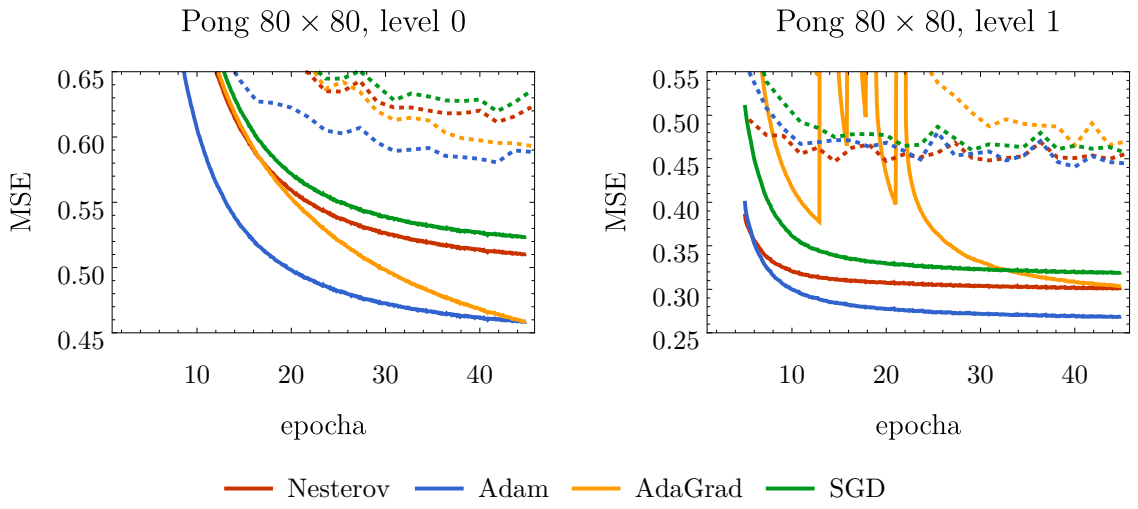
Obrázek 5: Srovnání průběhu MSE při učení různými řešiteli. Data: Pong 40 × 40, černobílá vlevo a S kanál z HSB vpravo. Redukce AE: 1600 → 400 hodnot.

grafy zobrazují klouzavý průměr přes jednu epochu. Jeden běh sestával z tří set epoch učení. Jedna epocha znamenala dvě stě iterací po třiceti dvou snímcích. Test sítě proběhl každých deset epoch, grafy chyb na testovacích datech jsou vykresleny přerušovaně.

Parametry řešitelů byly ponechány na výchozích hodnotách, rychlost učení však byla experimentálně měněna pro dosažení nejlepších výsledků v tomto testu. Na první pohled je zřejmé, že si bin. data vedou výrazně lépe než černobílá a S kanál z HSB. Přičemž průběh řízený SCE má o řád nižší chybu než MSE. Přes jisté nevýhody binární

reprezentace obrazu byla pro další testy zvolena tato varianta v kombinaci se SCE.

Další test byl zaměřen na výběr jednoho ze čtyř řešitelů, kteří dříve dosáhli přibližně 0.1 chyby. Zde již byla použitá ostrá sada dat sestávající z padesáti tisíc binarizovaných snímků 80 × 80 px. Tentokrát se pět náhodně iniciovaných vah učilo jen pět epoch a dál pokračoval pouze nejlepší model. Experiment proběhl na dvou postupně učených stupních AE: 6400 → 1600 → 800 → 1600 → 6400. Druhý stupeň (level 1) byl učen se zafixovanými vahami prvního (level 0).



Obrázek 6: Průběh MSE při učení různými řešiteli dle SCE. Data: Pong 80 × 80, binarizováno. Redukce AE: 6400 → 1600 → 800 hodnot. Zobrazen prodloužený nejlepší běh z pěti po pěti echochách.

Průběh učení obou úrovní AE zachycuje obr. 6, chyba testovací množiny dat je vykreslena přerušovaně po dvou epochách. V prvním stupni vedou Adam a AdaGrad, dle průběhu učení druhého stupně byl vybrán Adam. Je pozoruhodné, že přidáním druhé úrovně se vylepšila chyba samotné první, aniž by se její váhy upravily.

Nyní se ve stejném duchu pokračovalo prohlubováním započatého AE. S řešitelem Adam, ztrátovou funkcí SCE a binarizovanými daty byl postupně naučen autoenkodér o devíti stupních: 6400 → 1600 → 800 → 400 → 200 → 100 → 70 → 40 → 20 → 10. Učení probíhalo na stejném principu jako u prvních dvou stupňů. Od šesté úrovně byly navíc váhy celého AE dolaďovány.

Počátečních 6400 hodnot se podařilo zredukovat na 20 a 10 s průměrnou chybou necelého jednoho pixelu na testové sadě dříve neviděných snímků. Následujícím krokem bylo zkusit dle této reprezentace naučit herního agenta.

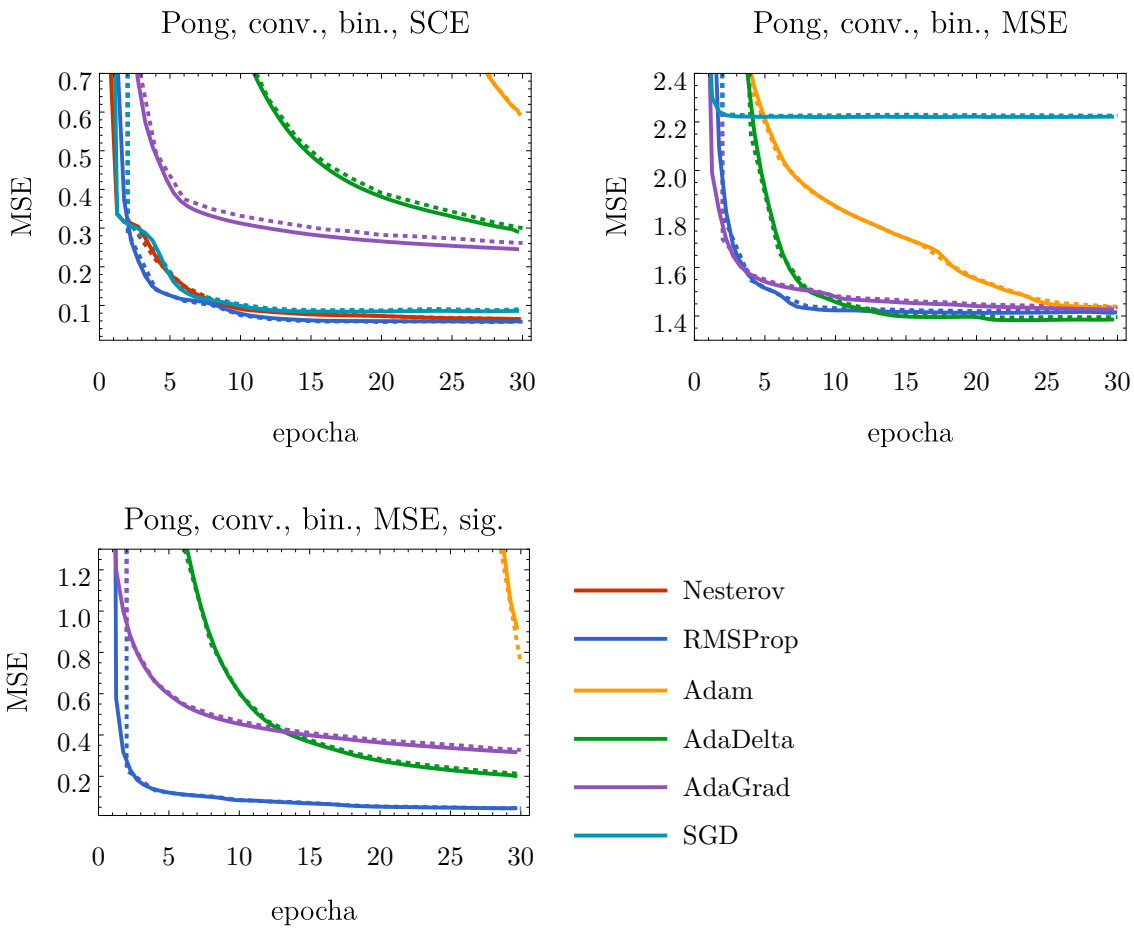
5.4 Konvoluční autoenkodér

Hlavním přínosem konvolučního autoenkodéru oproti plně propojenému, by měla být lepší generalizace, tedy nižší rozdíl rekonstrukční chyby na trénovacích a testovacích

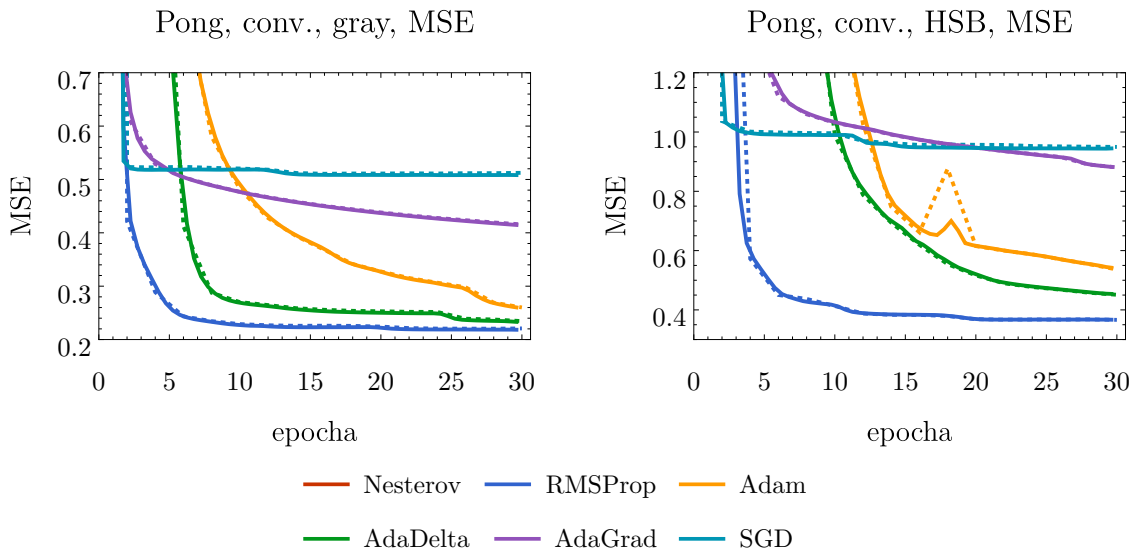
datech. Tyto AE se chovají poněkud odlišně a zopakovat s nimi první fázi testů, vybrat ideálního řešitele, formát dat a ztrátovou funkci, bylo nutností. Postup byl stejný, kombinace všech variant byly opakovaně spouštěny na 30 epoch a výsledné průběhy porovnány v grafech 7 a 8.

Pokud některý z řešitelů není zobrazen, je buď příliš mimo rozsah anebo byl jeho běh nestabilní a ztrátová funkce skončila na NaN. V dané kombinaci je tedy nepoužitelný. Týká se to SGD a Nesterova, ten se do výřezu grafu vešel jen na binarizovaných datech se SCE. Oproti plně propojenému AE je zde použita MSE napojená za sigmoidou i pro učení. Ukazuje se, že v případě konvolučního AE je to při použití RMSProp nejvýhodnější způsob učení. Další zajímavou variantou je Nesterov se SCE též na bin. datech. Toto jsou však údaje jen z první fáze. Na ostrých datech (padesát tisíc bin. snímků 80×8) se však obě varianty potýkají ze značnými potížemi. Neprojevuje se zde efekt vylepšování předchozí vrstvy jako v plně prpojeném AE, spíše naopak. Váhy jsou také silně závislé na počáteční inicializaci.

Konvoluční síť ve své podstatě poskytuje nepřeberné množství možností nastavení parametrů. Počtem a velikostí filtrů počínaje, různým řazením pooling a aktivačních



Obrázek 7: Srovnání průběhu MSE při učení dle SCE a MSE různými řešiteli. Data: Pong 40×40, binarizována. Konvoluční AE se 4 filtry 3×3 a max-poolingem 2×2.



Obrázek 8: Srovnání průběhu MSE při učení různými řešiteli. Data: Pong 40 × 40, černobílá vlevo a S kanál z HSB vpravo. Konvoluční AE se 4 filtry 3×3 a max-poolingem 2 × 2.

vrstev konče. Dle [32] není třeba velkých filtrů, bohatě má postačit více vrstev za sebou s menšími jádry. Velkými kernely je myšlena délka hran 5 px a více.

Testovaný AE měl jeden stupeň z konvoluční, pooling vrstvy a ReLU aktivace. Konvoluční a protilehlá dekonvoluční vrstva obsahovaly čtyři filtry velikost 3×3 produkující čtyři kanály dimenze 39 × 39. Dále byly testovány různé počty a řazení filtrů s hranou nejčastěji 3 a 5. Pooling se ukázal efektivní maximálně ve dvou vrstvách. Max-pooling sice zachovává nejvýraznější informaci, ale ne její polohu. Problém pooling je unpooling. To by téměř geniálně mohly řešit přepínače, jenže pak informace o poloze z příznaků téměř úplně vymizí. Autoenkodér s přepínači a poolingem za každou konvolucí dokázal rekonstruovat Pong z pouhých čtyř hodnot, ale při bližším zkoumání se ukázalo, že kód snímků se prakticky neliší od kódu stejného snímku ale bez míčku. RNN také podle takového vstupu nebyla schopná projevit jakékoliv známky učení.

Často se AE naučil velice dobře kódovat pálky, ale míček ne. To vedlo k návrhu autoenkodéru přijímajícího dvoukanálový obraz. Herní objekty v graficky prostých Atari 2600 hrách jsou většinou přímo odlišitelné prostřednictvím barev. Právě v Pongu mají pálky i míček různé barvy, rozdělit je do dvou kanálů je tedy prosté. Nyní bylo zapotřebí naučit dva AE. Jeden měl hledat příznaky míčku a druhý kódovat pálky. Výhodným vedlejším efektem bylo rapidní snížení velikosti datové sady nutné pro rovnoměrný pokryv stavů hry. Byly sestaveny dvě nové separátní sady dat, každá s přibližně šesti tisíci snímků.

Architektury AE se nijak zvlášť nelišily. Testovány byly verze s dvěma až třemi pooling vrstvami následované řadou šesti až sedmi konvolučních vrstev s ReLU. Snaha byla dosáhnout snížení dimenze vstupu na sedmdesát až sto hodnot. Konfigurace AE pro míček a pálky jsou uvedené v tabulce 1.

	Míček		Pálky	
	Parametry	Data	Parametry	Data
		1×80^2		1×80^2
lvl 0	$4, 3 \times 2$ pooling	4×78^2 4×39^2	$4, 5 \times 2$ pooling	4×76^2 4×38^2
lvl 1	$4, 3 \times 2$ pooling	4×37^2 4×19^2	$4, 3 \times 2$ pooling	4×36^2 4×18^2
lvl 2	$4, 3 \times 2$	4×17^2	$4, 3 \times 2$ pooling	4×16^2 4×8^2
lvl 3	$4, 3 \times 2$	4×15^2	$4, 3 \times 2$ pooling	4×6^2 4×3^2
lvl 4	$4, 3 \times 2$	4×13^2	$4, 3 \times 2$	4×2^2
lvl 5	$4, 3 \times 2$	4×11^2		
lvl 6	$4, 3 \times 2$	4×9^2		
lvl 7	$4, 3 \times 2$	4×7^2		
lvl 8	$4, 3 \times 2$	4×5^2		
lvl 9	$4, 3 \times 2$	4×3^2		

Tabulka 1: Sloupec parametrů obsahuje velikosti a počet filtrů, na stejném řádku je velikost a počet kanálů dat po průchodu touto vrstvou. Na označených řádcích probíhal max-pooling 2×2.

Modely Míček A a Pálky B byly zkombinovány a výsledná síť s dvoukanálovým vstupem byla použita pro herní test s RNN, při opakování stejné hry poměrně snadno dosáhl perfektní strategie – odpalu míčku na takové pozici, že protihráč nemá šanci.

5.4.1 Boxing

Další testovanou hrou byl Boxing. Obrazová data byla zpracovaná stejným způsobem jako u Pongu. Z původních snímků velikosti 210×160 px bylo vyříznuto okno 140×96 px obsahující veškerou dynamiku hry: dva různobarevné boxery stojící proti sobě. Tento výřez byl zmenšen na 70×48 px a binarizován. Záznamem náhodného hráče byly vytvořeny dva soubory dat. Jeden o padesáti tisících snímcích pro plně propojený autoenkodér a druhý pro konvoluční AE. Druhá množina dat byla vytvořena v duchu

dvoukanálového AE pro Pong. Dle barev boxerů byl obraz rozdělen do dvou kanálů a binarizován. Tím je ovšem ztracena informace, kdo je který boxer, nebýt separátních kanálů. Nyní oba kanály zachycují stejný objekt, není proto problém je sloučit do jedné sady. Ve výsledku dvoukanálový AE tudíž stačí trénovat jen s jedním kanálem. Tento trik nemusí být limitován jenom na boxing. Pokud hra obsahuje množství objektů stejných nebo podobných tvarů a plánuje se je reprezentovat v samostatných kanálech, pak lze využít stejný princip a ušetřit výpočetní zdroje. Ve struktuře AE pro Boxing jsou opět použity osvědčené malé filtry, počáteční pooling a řada samotných konvolučních vrstev.

5.5 Herní agent

Finálním testem autoenkodéru je jeho nasazení do vývoje herního agenta. RNN byla zvolena pro svou snadnou implementaci a všestrannost. Plně propojená RNN (použitá zde) má jen tři parametry: počty neuronů v každé z vrstev. Velikost vstupu je určena napojeným AE. Dimenze výstupu je dána komplexností ovládání hry, jeden výstup odpovídá jedné akci agenta. Atari 2600 bylo ovládáno osmi směrným joystickem s tlačítkem, umožňujícím provést až sedmnáct unikátních aktivních akcí a navíc prázdnou operaci. Většina her si vystačí s menším počtem. Pongu stačí $2+1$ akce, Boxing využije $8+1$ akcí.

Agenta reprezentují jeho váhy. Počty neuronů ve vstupní, skryté a výstupní vrstvě n_{in}, n_{hid}, n_{out} udávají celkový počet vah:

$$n_w = (n_{in} + n_{hid} + n_{out} + 1) * (n_{hid} + n_{out}), \tag{17}$$

kde 1 představuje bias.

Všech n_w je učeno evoluční strategií xNES. Na počátku je pravděpodobnostní model náhodně iniciován a začne první iterace. Jedna iterace se sestává ze vzorkování populace herních agentů (přibližně dvacíti, podle počtu vah), jejich ohodnocení hraním hry a následné aktualizace modelu. Před další iterací je celá populace vyhlazena.

Vyhodnocení populace probíhalo paralelně na centrálně spravovaných procesech (Mathematica podkernelech). Každý kernel dostane skupinku agentů a obratem je vyhodnotí na své instanci AE a ALE hry. Samotná evoluce probíhala v centrálním kernelu Mathematica. Ten skrz alePipeLink přijímal obraz a prostřednictvím CaffeLink ho posílal do Caffe na AE. Výsledné příznaky zpracovala RNN a určila další akci. Tento proces se opakuje dokud hra neskončí. Evoluční strategie nepracují podle absolutní fitness, jen podle pořadí. Aby se od sebe alespoň trochu odlišili prohrávající hráči, dostávají malý bonus na základě délky kola. Bonus je omezen jen na hráče se záporným skóre a nemůže být větší než jedna.

5.6 Zhodnocení

Samotné autoenkodéry se hodnotí lehce. Stačí se podívat na jejich rekonstrukční chybu případně porovnat rekonstrukce několika snímků s originálem. Důležitým aspektem je

chyba na testovacích datech. To je jediné vodítko naznačující, jak se asi AE bude chovat v reálném nasazení, protože na neznámé snímky bude nutně narážet neustále. To je výhoda konvolučních AE, které zde uplatní lokalitu filtrů a jsou tak imunní vůči posunu objektů v obraze. Na druhou stranu se konvoluční AE hůře učí, jsou více náchylné na počáteční iniciaci vah a to zvlášť ve spodních vrstvách. Nedaří se s nimi tak výrazně redukovat dimenzi jako s plně propojenými AE.

Porovnávat AE podle skóre na nich učených RNN není snadné, jelikož není zaručeno, že se RNN dokáže naučit hrát hru z použitelných příznaků za každých podmínek. Případně po kolika epochách. Typický si agent na začátku vede velmi špatně a až po čase se začnou objevovat známky učení. V Pongu to většinou znamená nalezení pozice, z které protihráče vždy „vyšplouchne“. Proti tomu v Boxing je skórování výrazně lehčí a tak se dá ve vývoji RNN sledovat jisté postupné zlepšení.

Při pohledu na výsledky v [1] či [2] je evidentní převaha RL metod přistupujících k učení na bázi stavů a odhadů budoucích zisků při aplikaci konkrétní akce. V Pongu není moc prostor pro soupeření, jak agent získá maximum 21 bodů je hotov. I prostá RNN tohoto byla schpná dosáhnout nalezením neporazitelné pozice. Tato strategie se jí dařila udržet přibližně v polovině případů, v průměrném skóre tedy zaostávala.

Nejlepší výsledek v Boxingu dosáhl konvoluční AE redukující obraz na třicet dvě hodnoty v kombinaci s RNN o dvou skrytých neuronech. Skóre činilo 37 bondů, průměrné jen kolem 10. Dokonalého výsledku sta bodů je v [1] dosaženo prohledáváním, jejich nejlepší učený kontrolér získal 44. DQN zvládá průměrně až 77 bodů [3].

Kapitola 6
Závěr

V této práci je prozkoumáno využití autoenkodéru a rekurentní neuronové sítě k obecnému hraní her. Speciální architektura umělé neuronové sítě, autoenkodér (AE), byla použita k naučení nízko dimenzionálních vektorů příznaků obrazu her. Tato redukovaná reprezentace posloužila jako vjem herního agenta, rekurentní neuronové sítě (RNN). Agent byl učen evoluční strategií snažíc se dosáhnout co nejvyšší skóre. Za testovací platformu agentů bylo zvoleno ALE - simulátor herní konzole Atari 2600.

Prozkoumány byly vlastnosti konvolučních i klasických, plně propojených autoenkodérů. Otestována byla jejich schopnost kódovat vstupní data – snímky hry v závislosti na jejich formátu, metodě aktualizace vah sítě a ztrátové funkci. Snímky byly převedeny do monochromní podoby jedním ze tří způsobů: černobílá (průměr RGB hodnot), binarizace (pozadí černé, objekty bílé) a vytažení S kanálu z HSB barevného prostoru. Nejlépe se osvědčily binarizované snímky. Pro plně propojené AE se jako ideální projevila kombinace ztrátové funkce *sigmoid cross-entropy* a řešitele *Adam* (varianta sestupu gradientem). Konvoluční AE dosahoval nejnižších rekonstrukčních chyb se ztrátovou funkcí *mean squared error* a řešitelem *RMSProp*.

Dle těchto poznatků byly naučeny AE redukující obraz her z původních 33600 px (předem vyříznutého a zmenšeného přibližně na čtvrtinu v závislosti na hře) na pouhých pár desítek hodnot. Načež byla spuštěna evoluční strategie *xNES* řídící vývoj RNN (napojené na výstup enkódovací části AE), jejímž výstupem byl vektor hodnot mapován přímo na herní akce ALE. Podařilo se naučit agenty perfektně hrající Pong a obstojně Boxing. Schopnost AE naučit se použitelnou reprezentaci obrazu ke hraní her tedy byla prokázána.

Tuto práci by nebylo možno uskutečnit bez patřičného softwarového vybavení. Především se jednalo o Caffe, ALE a vše zahrnující Mathematica. Jinak samostatné programy byly propojeny zde implementovanými knihovnami CaffeLink (publikované na IMS2015) a alePipeLink. Ve výsledku tak bylo možné přímo z Mathematica řídit učení AE v Caffe, spouštět hry na ALE a evolucí vyvíjet agenty. Výstupem práce je vedle rozhraní pro práci s neuronovými sítěmi i soustava skriptů, umožňující téměř automatické trénovaní autoenkodérů a evoluci agentů nejen pro ALE.

Literatura

[1] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[5] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[6] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. *CoRR*, abs/1505.04366, 2015.

[7] Douglas Eck and Juergen Schmidhuber. A first look at music composition using LSTM recurrent neural networks. Technical Report IDSIA-07-02, IDSIA, www.idsia.ch/techrep.html, March 2002.

[8] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Netw.*, 4(2):251–257, 1991.

[9] Jan Koutník, Juergen Schmidhuber, and Faustino Gomez. Evolving deep unsupervised convolutional networks for vision-based reinforcement learning. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO '14, pages 541–548, New York, NY, USA, 2014. ACM.

[10] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[11] Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time-series. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*. MIT Press, 1995.

[12] Jake Bouvrie. Notes on convolutional neural networks. 2006.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

[14] George E. Dahl, Tara N. Sainath, and Geoffrey E. Hinton. Improving deep neural networks for LVCSR using rectified linear units and dropout. In *ICASSP*, pages 8609–8613. IEEE, 2013.

[15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038, 2014.

[16] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.

[17] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. *CoRR*, abs/1505.04366, 2015.

[18] Matthew D. Zeiler, Graham W. Taylor, and Rob Fergus. Adaptive deconvolutional networks for mid and high level feature learning. In Dimitris N. Metaxas, Long Quan, Alberto Sanfeliu, and Luc J. Van Gool, editors, *IEEE International Conference on Computer Vision, ICCV 2011, Barcelona, Spain, November 6-13, 2011*, pages 2018–2025. IEEE Computer Society, 2011.

[19] A. M. Raid, W. M. Khedr, M. A. El-dosuky, and Wesam Ahmed. Jpeg image compression using discrete cosine transform - A survey. *CoRR*, abs/1405.6147, 2014.

[20] Yoshua Bengio. Learning deep architectures for ai. *Found. Trends Mach. Learn.*, 2(1):1–127, January 2009.

[21] Jing Wang, Haibo He, and Danil V. Prokhorov. A folded neural network autoencoder for dimensionality reduction. In Jonathan H. Chan and Ah-Hwee Tan, editors, *Proceedings of the 3rd International Neural Network Society Winter Conference, INNS-WC 2012, Bangkok, Thailand, October 3-5, 2012*, volume 13 of *Procedia Computer Science*, pages 120–127. Elsevier, 2012.

[22] Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *Proceedings of the 21th International Conference on Artificial Neural Networks - Volume Part I*, ICANN'11, pages 52–59, Berlin, Heidelberg, 2011. Springer-Verlag.

[23] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, December 2010.

[24] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[25] Daan Wierstra, Tom Schaul, Jan Peters, and Jürgen Schmidhuber. Natural evolution strategies. In *Proceedings of the Congress on Evolutionary Computation (CEC08), Hongkong*. IEEE Press, 2008.

[26] Tobias Glasmachers, Tom Schaul, Sun Yi, Daan Wierstra, and Jürgen Schmidhuber. Exponential natural evolution strategies. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, GECCO '10, pages 393–400, New York, NY, USA, 2010. ACM.

[27] Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard L Lewis, and Xiaoshi Wang. Deep learning for real-time atari game play using offline monte-carlo tree search planning. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3338–3346. Curran Associates, Inc., 2014.

[28] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[29] Martin Kerhart and Jan Drchal. Mathematica binding for caffe deep learning framework. 2015.

[30] Sascha Lange and Martin A. Riedmiller. Deep auto-encoder neural networks in reinforcement learning. In *International Joint Conference on Neural Networks, IJCNN 2010, Barcelona, Spain, 18-23 July, 2010*, pages 1–8. IEEE, 2010.

[31] Solver / model optimization. `https://github.com/BVLC/caffe/blob/master/docs/tutorial/layers.md`. Cit.: 2016-05-20.

[32] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

Obsah CD

Na přiloženém CD jsou k nalezení:

- Elektronická verze textu v pdf.
- Zdrojové soubory implementovaných knihoven.
- Použité sady snímků k učení neuronových sítí.
- Videa zachycující hraní RNN.