Adaptação de Domínio em Deep Q Learning

Sérgio José de Sousa
Universidade Federal de Minas Gerais
Belo Horizonte, Minas Gerais
sergiosjs@ufmg.br

ABSTRACT

Este artigo propõe-se verificar a técnica de adaptação de domínio em 2 jogos de Atari 2600, Space Invaders e Demon Attack, jogos com jogabilidade e estilo parecidos. O objetivo de criar um modelo generalista capaz de jogar mais de um jogo e tentar ajudar a convergir mais rapidamente. Notamos que a transferência de domínio desacelerou o aprendizado mas permitiu um score maior no início do treinamento. Ao compararmos os modelos treinados em dois jogos e testarmos no domínio de origem pudemos verificar que o modelo esquece o que aprendeu chegando a ter um score próximo do aleatório.

KEYWORDS

Aprendizado por reforço, Transferência de conhecimento, Deep Q Learning, Adaptação de domínio

1 INTRODUÇÃO

Em diversas aplicações de aprendizado de máquinas e mineração de dados é desejado que os dados tenham a mesma distribuição do espaço e mesmo espaço de atributos. Contudo, no mundo real essa suposição não é válida.

Se temos um conjunto de dados pequeno de algum domínio de maneira que ao se criar e treinar com apenas esses dados gera-se modelos fracos com alta variância e baixo viés, caso típico chamado de overfitting, quando decora-se o dado gerando grande erro de teste. Para tentar resolver esse problema podemos realizar a chamada transferência de aprendizado e adaptação de domínio onde a situação que foi aprendida em um ambiente é explorada para melhorar a generalização em outro ambiente [3]. Ou seja, treina-se um modelo em um domínio onde se tem uma grande quantidade de dados rotulados, a partir disso, reutilizamos em outro conjunto de dados. Temos várias técnicas com diferentes abordagens para a transferência de aprendizado, [9] descreve três principais, transferência de aprendizado indutivo, transdutivo e não supervisionado, todas apresentaram melhoria se comparadas a não utilizá-las

Aprendizagem por reforço é uma técnica que envolve um agente que consegue observar os estados, interagir com seu ambiente e receber um sinal de recompensa como resposta. Seu objetivo então é maximizar o valor recompensa ao longo do tempo.

Suponhamos que exista um oráculo capaz de identificar todas as possibilidades de estados e todas as alternativas de ações que pode-se escolher, com isso conseguimos maximizar facilmente a recompensa. Como isso não existe, podemos supor que oráculo é uma função e que possui métodos que aproximam os valores dessa função. Essa abordagem é chamada de Q-Learning [15]. Estudos recentes juntam a abordagem de aprendizado por reforço e aprendizado profundo onde a rede neural representa a função Q e a cada iteração os parâmetros são ajustados para maximizar o resultado final. Essa técnica foi nomeada como Deep Q-Network, DQN [7], ou Deep Reinforcement Network, DRN.

O presente trabalho propõe-se realizar a transferência de aprendizado através de uma aplicação da DQN entre dois jogos de atari, Space Invaders e Demon Attack. Dois jogos semelhantes classificados como fixed shooter onde o jogador controla uma espécie de nave que fica fixa verticalmente conseguindo mover para direita, esquerda, atirar e combinar as ações. Com isso, após treinar a DQN para cada jogo, verificou-se se a transferência de aprendizado ajuda a convergir mais rapidamente e se uma mesma rede é capaz de jogar os dois jogos efetivamente.

2 TRABALHOS RELACIONADOS

Com a implementação de Arcade Learning Environment (ALE) [1], uma interface para centenas de ambientes de jogos do Atari 2600, a segunda geração do console que foi lançado em 1977 e popular por décadas, contendo diversos gêneros de jogos foram pioneiros nesta plataforma, tivemos um grande desafio a disposição da comunidade de inteligência artificial. ALE é uma metodologia experimental, foi construída em cima de Stella[1], um emulador open-source. Por padrão possui 60 frames por segundo, cada frame com 160 pixels de largura e 210 pixels de altura com 3 faixa de cores RGB como pode ser visto na 1. São 18 ações discretas correspondentes às combinações de botões do controle.

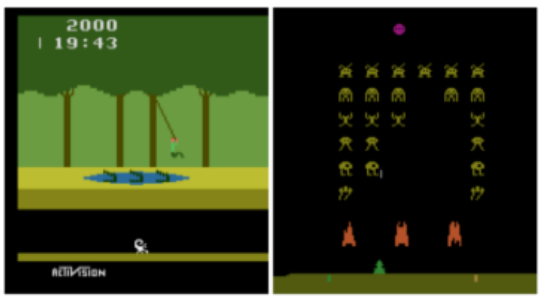

Figure 1: Screenshots de Pitfall! e Space Invaders com 160x210x3 atributos.

O controle e o aprendizado de agentes em ambientes com alta dimensionalidade sempre foram um dos maiores desafios do aprendizado por reforço. Muitas técnicas com aplicação de aprendizado por reforço foram propostas e grande parte utiliza métodos handcrafted para extrair as características combinadas com funções de políticas lineares, utilizando de técnicas como Q-Learning [15] que consiste num simples método de agentes aprenderem como otimizar suas recompensas utilizando o domínio Markoviano, um método

[1] https://stella-emu.github.io/

incremental de programação dinâmica onde é necessário apenas do presente que de alguma forma codifica estados e ações do passado para tentar prever a próxima ação. O Q-learning baseia-se na equação de Bellman como pode ser vista na Equação 1 onde o valor em colchete é o erro de predição o maxQ irá selecionar a melhor ação naquele instante, α define a taxa de aprendizado e γ o valor de decréscimo daquela previsão, reduzindo o valor de crença ao se tentar prever ações mais longes no futuro.

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a)] \quad (1)$$

Redes neurais profundas possibilitam a extração de alto nível de características em dados sensoriais como computação visual [4] e reconhecimento de fala [2]. Estes métodos utilizam arquiteturas de redes neurais que incluem camadas convolucionais e diversas camadas ocultas. Esses tipos de redes requerem uma grande quantidade de dados rotulados e não assumem a relação entre os dados o que tipicamente em aprendizado por reforço temos uma alta correlação entre os estados.

Utilizando um ambiente com alta dimensionalidade como [1] e rede neural convolucional [7] demonstrou-se uma arquitetura que é treinada utilizando uma variante do algoritmo Q-learning [15] com gradiente descendente estocástico atualizando os pesos da rede. Para resolver o problema da correlação dos dados utilizou-se a técnica de experiência de replay, memorizando e amostrando aleatoriamente as transações anteriores. Com o objetivo de criar uma rede neural capaz de aprender muitos jogos o artigo exibe testes em 7 jogos de Atari 2600, Pong, Breakout, Space Invaders, Seaquest, Beam Rider, Enduro e Q*bert. O algoritmo foi nomeado como deep Q-learning (DQN), nas próximas sessões será tratado com mais detalhes.

Aprimorando a DQN, [12] propoẽ uma nova arquitetura chamada Double Q-learning utilizando de duas redes neurais, uma atualizando constantemente e outra atualizando a cada intervalo configurado. Dessa maneira o modelo resultante não apenas reduz as superestimações observadas como também melhorou o desempenho em vários jogos.

Em [14] é apresentada uma nova arquitetura de rede neural, Dueling DQN, que utiliza dois estimadores separadamente,um para a função de valor de estado e outro para função de vantagem. O seu principal benefício é generalizar as ações superando vários algoritmos propostos.

Algumas vezes o valor de recompensa não é explícito para os agentes e por vezes a curiosidade serve como um valor de recompensa intrínseco incentivando nosso agente a explorar o ambiente e aprender as habilidades que podem ser úteis, o artigo [10] propõem uma formulação para a curiosidade como um erro na capacidade do agente prever as consequências de suas próprias ações.

Em 2016 foi apresentado o programa AlphaGO [13] que foi capaz de vencer um jogador profissional num placar de 4:1 no jogo Go, um jogo de tabuleiro chinês que possui um espaço de estados muito maior que do xadrez. AlphaGO aprendeu a jogar de maneira similar aos humanos, ele observou milhares jogos e jogou outros milhares aprimorando a cada partida utilizando de redes neurais profundas de maneira similar ao DQN. Conseguindo fazer jogadas que nunca foram vistas por humanos que a princípio ão pareciam fazer sentido e depois se mostraram ótimas estratégias.

Em ambientes mais complexos como jogos em 3D de tiro em primeira pessoa temos um desafio ainda maior se comparado com os ambientes dos jogos de Atari 2600. Em [5] propoẽ uma arquitetura modularizada para permitir que diferentes modelos sejam treinados independentemente para aprender informações extras sobre o jogo como presença de recursos e inimigos. Conseguindo superar diversas técnicas e até seres humanos no jogo Doom.

Seguindo metodologia multitarefas de transferência de aprendizado, [16] propõe uma arquitetura de rede neural com várias camadas convolucionais em paralelo, cada uma treinada em um diferente jogo de Atari, que se concatenam em camadas totalmente conectadas criando uma arquitetura multitarefas de destilação de políticas.

3 METODOLOGIA

Um problema comum em muitos jogos e especialmente nos de Atari 2600 é o screen flickering, devido ao limitado hardware o console não consegue desenhar todos os elementos na tela ao mesmo tempo. Para compensar isso alguns elementos são desenhados alternadamente na tela, piscando, de maneira rápida suficiente para ser imperceptível na maioria das vezes. Para resolver este problema é necessário uma amostragem maior de frames por vez, deste modo, usaremos 4 frames a cada passo do algoritmo.

Deste modo nosso espaço dimensional ficou ainda maior, sendo 210 x 160 pixels com 128 cores possíveis mas em escala RGB e 4 frames por amostra, o que nos daria uma entrada com 403.200 dimensões que é computacionalmente inviável. Seguindo a ideia proposta por [7], primeiramente reduzimos a dimensão da imagem para algo menor mas sem perder detalhes, ficando com 84 x 84 e depois convertemos para uma escala de tons de cinza como pode ser visto na Figura 2. Essa escala quadrada é requerida devido ao uso de GPU nas implementações de convoluções como pode ser visto em [4]. Após todas estas operações teremos amostras com dimensão 28.224.

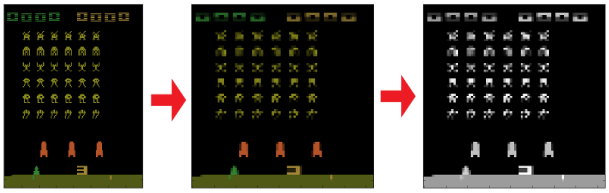

Figure 2: Pre-processamento dos estados, reduzindo a dimensão de 210x160 para 84x84 e depois em escala de cinza.

A partir deste conjunto de observações retornadas pelo ambiente E em tempo discreto definido como M imagens dos frames: $s_t = (x_{t-M+1}, ..., x_t) \in S$ a cada passo de tempo t. O agente então deve escolher entre as ações discretas possíveis descritas como $a_t \in A = \{0, ..., |A| - 1\}$, observamos a recompensa r_t retornada pelo emulador.

O agente tentará maximizar a recompensa descontada esperada onde $R_t = \sum_{t`=t}^{T} \gamma^{t`-1} r_{t`}$ onde t é o passo do algoritmo até o jogo terminar, γ variando de 0 a 1 é o fator de desconto servindo de um trades-off para recompensas futuras. Definimos a ação ótima como sendo a função $Q^*(s,a)$ que maximiza a recompensa esperada dado

um estado s e tomar uma ação a, $Q^*(s,a) = max_\pi E[R_t|s_t = s, a_t = a, \pi]$ onde π é a política que mapeia uma sequência de ações.

Com isso, seguimos a equação de Bellman e encontraremos o Q ótimo:

$$Q^*(s,a) = E_{s`}[r + \gamma max_{a`} Q^*(s`, a`)|s, a] \quad (2)$$

3.1 Experiência de Replay

Denominada como experience replay [6], uma técnica de aprendizado de máquina onde o agente pode salvar muitas etapas com estados anteriores, ações e próximos estados com a finalidade de um aprendizado online, reduzindo o número de ações requeridas para convergir.

Nós salvamos a cada passo a experiência adquirida pelo agente $e_t = (s_t, a_t, r_t, s_{t+1}, done_t)$ em uma fila de memória $D = \{e_1, \ldots, e_N\}$ onde $done_t$ é uma variável booleana indicando se aquele estado levou ao fim de um episódio. É necessário apagar as memórias mais antigas quando excede o tamanho máximo definido.

Na etapa de aprendizado do Q-learning nós utilizamos um mini-batch D_b extraído aleatoriamente de D. Essa abordagem possui muita vantagem em relação ao Q-learning padrão [11]. Primeiro porque um mesmo passo de experiência pode ser usado diversas vezes na etapa de aprendizado. Segundo, treinar amostras consecutivas é ineficiente devido a forte correlação entre as amostras. Terceiro ajuda a evitar mínimos locais.

Ao usar essa técnica o comportamento do agente é calculado pela média de muitos de seus estados anteriores, suavizando o aprendizado e evitando grandes oscilações [7].

3.2 DQN - Deep Q-networks

Basicamente a ideia o DQN é estimar a função de valor utilizando a equação de Bellman numa atualização iterativa através de uma rede neural θ. Para otimizar a rede utilizamos a função de perda de erro quadrático na iteração i:

$$L_i(\theta_i) = E_{s,a,r,s`}[(y_i^{DQN} - Q(s,a;\theta_i)^2), \quad (3)$$

onde

$$y_i^{DQN} = r + \gamma max_{a`} Q(s`, a`; \theta^-), \quad (4)$$

θ^- é uma segunda rede chamada target network. Possui a mesma estrutura de θ porém com os pesos atualizados em intervalos. Os pesos de θ são atualizados a cada passo utilizando gradiente descendente tendo como entrada o mini-batch D_b:

$$L_i(\theta_i) = E_{(s,a,r,s`)\sim U(D_b)}[(y_i^{DQN} - Q(s,a;\theta_i)^2), \quad (5)$$

A primeira versão da DQN [7] utiliza uma estrutura mais simples, o input da rede conta com a dimensões 84 x 84 x 4 imagens pré-processadas. A primeira camada escondida é uma convolução com 16 filtros de tamanho 8 x 8 e stride 4 usando ReLu [8] como função de ativação. A segunda camada convolucional com 32 filtros de tamanho 4 x 4 e stride 2, também com relu. A terceira camada totalmente conectada com 256 neurônios. Por fim a camada de saída totalmente conectada contendo uma quantidade de neurônios igual a quantidade de ações disponíveis em A.

3.3 Double DQN

No nosso trabalho utilizamos a técnica chamada Double DQN [12] que possui duas redes neurais utilizadas no momento de prever as ações do mini-batch D_b para o treino. Na versão anterior o operador max usa a mesma rede para ambas as variáveis de avaliação e seleção de ação.

A ideia é reduzir as superestimações na decomposição da operação max. Para tanto usaremos uma rede que é atualizada de maneira online θ e uma segunda rede θ^- atualizada em intervalos predefinidos.

É calculada os valores de predições das ações com θ e os estados anteriores s_t do mini-batch D_b. A rede target θ^- é usada para calcular o valor da melhor ação prevista utilizando os dados de estados futuros s_{t+1} do mini-batch D_b seguindo a equação:

$$Y_t^{DoubleDQN} = R_{t+1} + \gamma Q(S_{t+1}, argmax_a Q(S_{t+1}, a; \theta_t), \theta_t^-). \quad (6)$$

Os pesos da rede θ^- são copiados periodicamentes de θ. Dessa maneira os resultados obtiveram uma média de scores maiores e oscilações menores.

O Double DQN modificou a estrutura da rede em comparação com a versão anterior [7]. Nela temos o input com a dimensões 84 x 84 x 4 imagens pré-processadas. A primeira camada convolucional possui 32 filtros de tamanho 8x8 e stride 4. A segunda camada convolucional possui 64 filtros de tamanho 4x4 e stride 4. A última camada convolucional possui 64 filtros de tamanho 3x3 e stride 1. Seguidos de uma camada totalmente conectada com 512 neurônios. Todas com função de ativação ReLu [8]. Terminando com a camada de saída totalmente conectada contendo uma quantidade de neurônios igual a quantidade de ações disponíveis em A.

3.4 Dueling DQN

A intuição por trás dessa arquitetura é que para muitos estados, não é necessário estimar todas possibilidades de ações. Por exemplo, no jogo Space Invaders a ação atirar só faz sentido quando se tem um inimigo no alvo e se não tem barreira de proteção entre a nave do jogador e inimigo.

Como ilustrado na Figura 3 temos uma arquitetura que segue as camadas convolucionais idênticas ao proposto em [12]. Ao final das convoluções a arquitetura segue em dois fluxos com camadas totalmente conectadas, possuindo capacidade de fornecer estimativas separadas das funções de valor e vantagem. Esses dois fluxos se concatenam produzindo a camada de saída com os valores Q de cada ação.

Nesta última camada temos uma importante formulação para calcular a vantagem:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + (A(s,a;\theta,\alpha) - \frac{1}{|A|}\sum_{a'} A(s,a';\theta,\alpha)). \quad (7)$$

Aqui, θ representa os parâmetros das camadas convolucionais, enquanto α e β são os parâmetros dos dois fluxos de camadas totalmente conectadas. Basicamente subtraímos os valores do fluxo α pela sua média e somamos com o valor do fluxo β assim temos a camada de saída de nossa arquitetura.

Com essa formulação perdemos a semântica de V e A propostas em [14] porque agora eles funcionam como off-target por uma constante, mas ganhamos em estabilidade com essa otimização.

A rede usada nesse trabalho se basea-se a Dueling DQN [14] bem similar a Double DQN com o mesmo input de dimensões 84 x 84 x 4, 3 camadas convolucionais. A primeira possui 32 filtros de tamanho

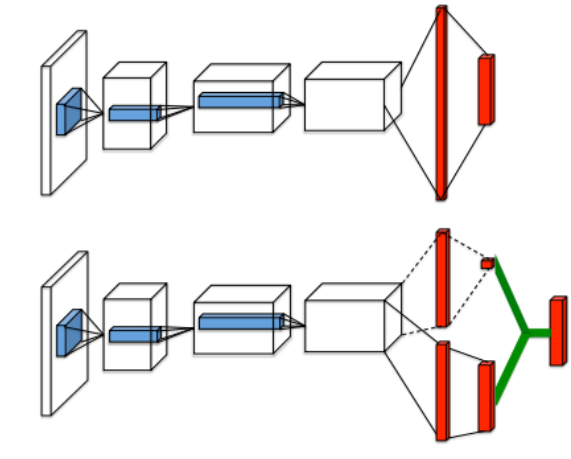

Figure 3: Acima temos a arquitetura Double DQN [12]. Abaixo temos a Dueling DQN [14] que, após as camadas convolucionais possui dois fluxos de dados, uma que estima o valor do estado e a outra a vantagem por cada ação. A linha em verde representa a equação que combina esses dois valores que nesse trabalho é dado pela equação 7

8x8 e stride 4. A segunda possui 64 filtros de tamanho 4x4 e stride 4. A terceira possui 64 filtros de tamanho 3x3 e stride 1. Como pode ser visto em Figure3 a rede se divide em dois fluxos em paralelo, o primeiro fluxo possui uma camada totalmente conectada com 512 neurônios seguido de mais uma camada com apenas 1 neurônio chamada de saída de vantagem. O segundo fluxo possui uma camada totalmente conectada com 512 neurônios seguido de mais uma camada com totalmente conectada com uma quantidade de neurônios igual a quantidade de ações disponíveis em A. Esses dois fluxos são concatenados em uma nova camada com $|A|$ neurônios respeitando a equação7. todas camadas possuem função de ativação ReLu.

4 EXPERIMENTO

A máquina usada no experimento possui um processador Intel i3-4150 CPU @ 3.50GHz 3.50GHz com 12GB de RAM e NVIDIA GeForce GTX 960 com 2GB de VRAM.

O pseudocódigo do algoritmo implementado pode ser verificado em Algoritmo 1. A execução do algoritmo levou cerca de 24 horas para cada 1 milhão de passos. Devido a esse alto custo computacional e de tempo, decidimos aplicar a técnica em apenas dois jogos e nos limitamos a 1 milhão de passos em cada experimento.

O experimento consiste em rodar a mesma arquitetura e hiperparâmetros nos jogos. Iniciamos com treinamento no Space Invaders como pode ser visto em Figura 5 (a) apesar da oscilação podemos identificar uma tendência crescente no gráfico. O mesmo pode ser visto com relação ao jogo Demon Attack, Figura 5 (d) onde temos uma tendência crescente visível.

O domínio dos jogos selecionados são próximos, ambos com o mesmo número de ações e estilo parecido. O segundo passo do experimento consiste em utilizar o modelo treinado em um jogo,

Table 1: Test with each model network with each game

Model Network	Space Invaders	Demon Attack
Normal Space Invaders	182.4	0.0
Transfer to Space Invaders	184.275	119.8
Normal Demon Attack	147.025	331.45
Transfer to Demon Attack	153.775	196.5

transferir o domínio para outro e treiná-lo novamente, com a finalidade de verificar se o conhecimento adquirido em outro domínio pode ajudar ou não no treinamento.

Ao utilizar o conhecimento adquirido em Figura 5 (d) no jogo Space Invaders obtivemos o resultado Figura 5 (b), também com uma tendência crescente. Ao compararmos as tendências de Figura 5 (a) e Figura 5 (b) podemos ver que o conhecimento adquirido em Demon Atack piorou um pouco o aprendizado se comparado com o treinamento puro em Space Invaders Figura 5 (c).

Aplicando o conhecimento adquirido em Figura 5 (a) e treinarmos no jogo Demon Attack obtivemos o resultado Figura 5 (e) com uma tendência com crescimento pequena. Ao compararmos os resultados das tendências Figura 5 (f) podemos ver que a transferência de conhecimento ajudou ao modelo a iniciar com um score maior, porém o seu crescimento é muito lento, levando o modelo treinado puramente no jogo obter um melhor resultado final.

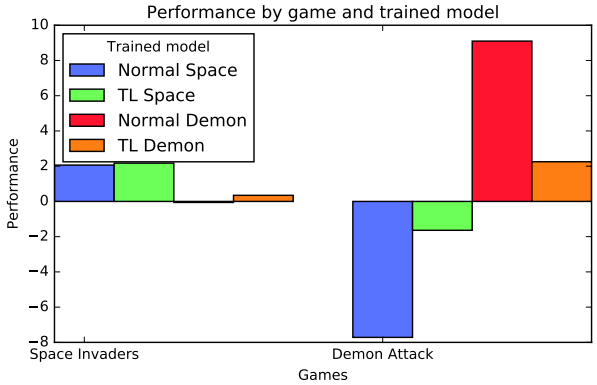

Figure 4: Performance dos 4 modelos treinados e testados nos jogos Space Invaders e Demon Attack. O resultado é uma normalização da média dos scores subtraindo pela média aleatória e dividindo pela média de score humano [14]. Valo-res acima de 100 representam habilidades semelhantes a humano, próximas ou abaixo de zero representam agentes aleatórios.

A próxima etapa do experimento consiste em testar os 4 modelos gerados com os dois jogos. Rodamos 200 episódios de cada jogo em cada modelo e calculamos a média dos scores, os resultados podem ser visto na Tabela 1.

O modelo treinado apenas no jogo Space Invaders aplicado no jogo Demon Attack resultou em um score zerado, mostrando que o conhecimento apenas de Space Invaders não ajudou ao testar em um jogo não treinado.

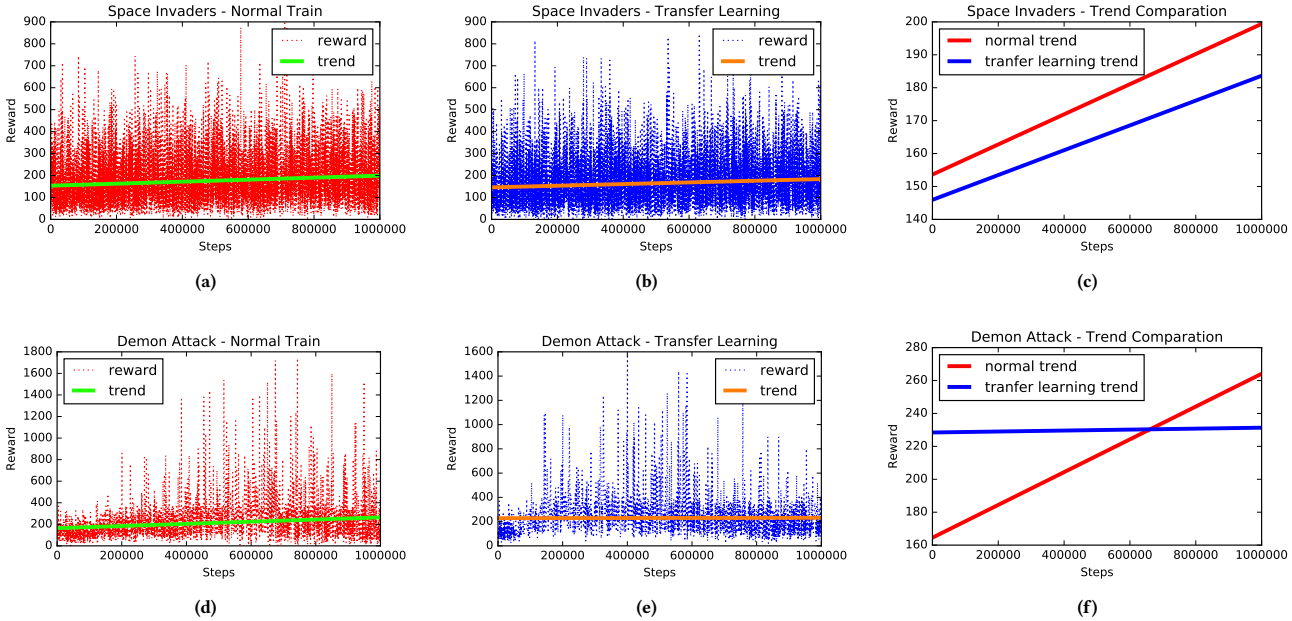

Figure 5: Resultados obtidos no treinamento das redes juntamente com a tendência de crescimento. (a) Rede treinada puramente no jogo Space Invaders. (b) Rede treinada no jogo Demon Attack e transferida para o jogo Space Invaders. (d) Rede treinada puramente no jogo Demon Attack. (e) Rede treinada no jogo Space Invaders e transferida para o jogo Demon Attack. (c) e (f) são comparações entre as tendências.

Apesar da tendência de crescimento do modelo transferido para o Space Invaders ter sido menor que o crescimento do modelo treinado puramente Figura 5 (c) ele acabou alcançando uma média de score maior.

Já os modelos que foram treinados com a adaptação de domínio se testado no domínio de origem, obtiveram um comportamento bem próximo ou igual ao aleatório. Demonstrando um certo esquecimento do conhecimento anterior.

A Figura 4 mostra a relação dos valores de score com scores aleatórios e humano. Sendo o valor zerado modelos aleatórios e modelos e valor 100 modelos com scores humano. Em pesquisas anteriores [7] [12] [14] vemos que os modelos superam humanos em treinamentos acima de 50M de passos, nossa pesquisa se restringiu a apenas 1M de passos.

4.1 Hiperparâmetros

Utilizamos learning rate igual a 0.000007, um ϵ inicial de 1.0, com 200000 de passos reduzindo até um valor final de 0.1. C passos para atualizar θ^{-} igual a 5000. γ igual a 0.99. Memória de replay com tamanho 300000, mini-batch de tamanho 32.

5 CONCLUSÃO

Esse artigo demonstrou uma das técnicas transferência de aprendizado chamada adaptação de domínio. Utilizamos a arquitetura Dueling DQN e aplicamos o experimento entre os jogos Demon Attack e Space Invaders. Verificamos que a adaptação de domínio pode proporcionar um score maior no início do treinamento mas pode afetar negativamente o crescimento da curva de aprendizagem. Outro fator importante, ao testarmos no domínio de origem do modelo verificou-se a grande perda de conhecimento do jogo original.

Para trabalhos futuros podemos testar em mais passos de maneira que o modelo se torne mais especialista no jogo. Outra abordagem é verificar se o conhecimento adquirido em jogos cujo o fator de exploração seja uma recompensa não explicita como Super Mario Bros pode ajudar em outros jogos do gênero.

REFERENCES

[1] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. 2013. The Arcade Learning Environment: An evaluation platform for general agents. *J. Artif. Intell. Res.(JAIR)* 47 (2013), 253–279.
[2] George E Dahl, Dong Yu, Li Deng, and Alex Acero. 2012. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on audio, speech, and language processing* 20, 1 (2012), 30–42.
[3] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep learning*. Vol. 1. MIT press Cambridge.
[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.
[5] Guillaume Lample and Devendra Singh Chaplot. 2017. Playing FPS Games with Deep Reinforcement Learning.. In *AAAI*. 2140–2146.
[6] Long-Ji Lin. 1993. *Reinforcement learning for robots using neural networks*. Technical Report. Carnegie-Mellon Univ Pittsburgh PA School of Computer Science.
[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).
[8] Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*. 807–814.
[9] Sinno Jialin Pan and Qiang Yang. 2010. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering* 22, 10 (2010), 1345–1359.

[10] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. 2017. Curiosity-driven Exploration by Self-supervised Prediction. In *ICML*.
[11] Richard S Sutton and Andrew G Barto. 1998. *Reinforcement learning: An introduction*. Vol. 1. MIT press Cambridge.
[12] Hado Van Hasselt, Arthur Guez, and David Silver. 2016. Deep Reinforcement Learning with Double Q-Learning.. In *AAAI*, Vol. 16. 2094–2100.
[13] Fei-Yue Wang, Jun Jason Zhang, Xinhu Zheng, Xiao Wang, Yong Yuan, Xiaoxiao Dai, Jie Zhang, and Liuqing Yang. 2016. Where does AlphaGo go: From church-turing thesis to AlphaGo thesis and beyond. *IEEE/CAA Journal of Automatica Sinica* 3, 2 (2016), 113–120.
[14] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. 2015. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581* (2015).
[15] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8, 3-4 (1992), 279–292.
[16] Haiyan Yin and Sinno Jialin Pan. 2017. Knowledge Transfer for Deep Reinforcement Learning with Hierarchical Experience Replay.. In *AAAI*. 1640–1646.

ALGORITMO

Input: D - initialize replay memory to capacity N
Input: θ - initialize random weights
Input: θ^{-} - initialize weights $\theta^{-} = \theta$
$t = 0$;
for *episode* $e \in \{1, 2, ..., M\}$ **do**
  Initialize initial state s_{old} with preprocessed frames;
  while ***not*** *done* **do**
    if *probability* ε **then**
      $a_t = randomAction$;
    else
      $a_t = argmax_a Q(s_t, a; \theta)$;
    end
    $r_t, s_t, done = 0, [], False$;
    for $frame\ f \in \{1, 2, 3, 4\}$ **do**
      Execute action a_t and observe the reward $tempR_t$, frame $tempS_t$ and if done $tempDone$;
      $s_t.add(tempS_t)$;
      $r_t = r_t + tempR_t$;
      $done = done|tempDone$;
    end
    Store transition $(s_{old}, a_t, r_t, s_t, done)$ in memory replay D ;
    if $|D| > N$ **then**
      delete oldest memories in D;
    end
    if $|D| > MemoryMinToLearn$ **then**
      Sample random mini-batch of transitions $(s_j, a_j, r_j, s_{j+1}, done_j)$ in D;
      if $done_j$ **then**
        $y_j = r_j$;
      else
        $y_j = r_j + \gamma max_{a'} Q(s_{j+1}, a'; \theta^{-})$;
      end
      Perform a gradient descent in network θ step on $(y_j - Q(s_j, a_j; \theta))^2$;
    end
    $s_{old} = s_t$;
    $t++$;
    Every C steps $\theta^{-} = \theta$
  end
end

Algorithm 1: Dueling DQN