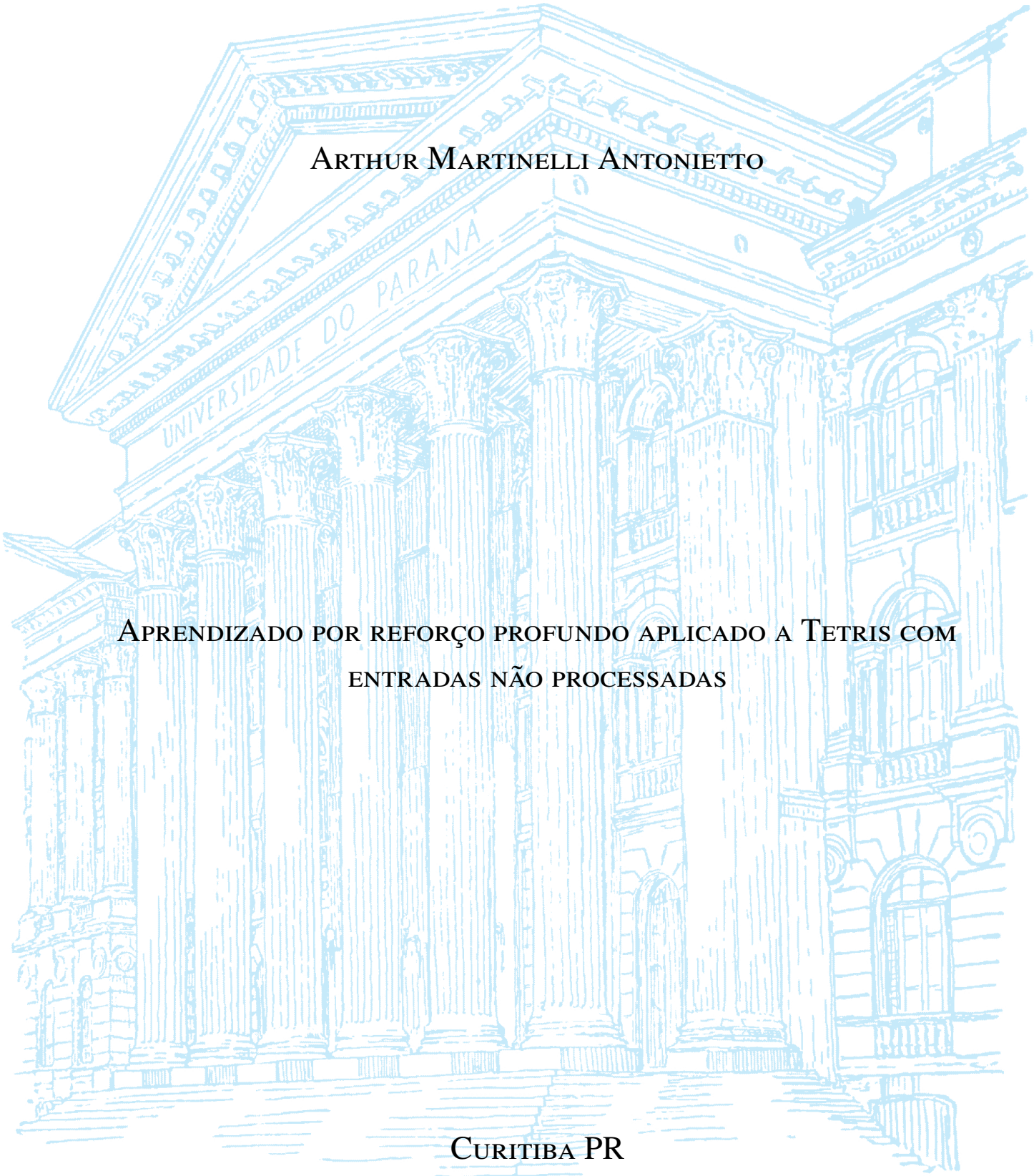
Universidade Federal do Paraná

Arthur Martinelli Antonietto



Aprendizado por reforço profundo aplicado a Tetris com entradas não processadas

Curitiba PR
2023

Arthur Martinelli Antonietto

Aprendizado por reforço profundo aplicado a Tetris com entradas não processadas

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Fabiano Silva.

Curitiba PR
2023

Universidade Federal do Paraná
Setor de Ciências Exatas
Curso de Ciência da Computação

Ata de Apresentação de Trabalho de Conclusão de Curso 2

Título do Trabalho:

Aprendizado por reforço profundo aplicado a Tetris com entradas cruas

Autor:

GRR20182559 **Nome: Arthur Martinelli Antonietto**

Apresentação: Data: 06/03/2023 Hora: 13:30 Local: Auditório Alexandre Direne

Orientador: Fabiano Silva

Membro 1: Guilherme A. Derenievicz

Membro 2: Mariane R. Sponchiado Cassenote

(nome) (assinatura)

AVALIAÇÃO – Produto escrito		ORIENTADOR	MEMBRO 1	MEMBRO 2	MÉDIA
Conteúdo	(00-40)				
Referência Bibliográfica	(00-10)				
Formato	(00-05)				
AVALIAÇÃO – Apresentação Oral					
Domínio do Assunto	(00-15)				
Desenvolvimento do Assunto	(00-05)				
Técnica de Apresentação	(00-03)				
Uso do Tempo	(00-02)				
AVALIAÇÃO – Desenvolvimento					
Nota do Orientador	(00-20)		**********	**********	
NOTA FINAL		**********	**********	**********	**90**

Os pesos indicados são sugestões.

Conforme decisão do colegiado do curso de Ciência da Computação, a entrega dos documentos comprobatório de trabalho de Conclusão de Curso 2 deve deve respeitar os seguintes procedimentos: o orientador deve abrir um processo no Sistema Eletrônico de Informações (SEI – UFPR); Selecionar o tipo: *Graduação: Trabalho Conclusão de Curso;* informar os interessados: nome do aluno e o nome do orientador; anexar esta ata escaneada e a versão final do PDF da monografia do aluno; Tramitar o processo para CCOMP (Coordenação de Ciência da Computação).

À minha família e amigos, pela alegria, pelo conforto e por todos os momentos que compartilhamos. É em vocês que encontro meu a razão de cada instante.

Agradecimentos

À minha família e amigos, por todo o apoio que me deram do início ao fim desse projeto e contribuiram, mesmo que apenas como ouvidos, para eu dar o melhor de mim aqui.

Ao meu orientador, pelas inúmeras ideias e discussões que trouxe, sempre enriquecendo meu trabalho com novos conceitos e me guiando para que fosse possível finalizá-lo.

Aos meus colegas de curso, por criar um ambiente não só acolhedor e descontraído, mas também curioso, que sempre me fez aprender a cada dia algo novo.

À sociedade brasileira como um todo, pela oportunidade de estudar em uma universidade pública e gratuita, que é capaz de colocar a ciência, a diversidade e o compromisso com o saber acima de quaisquer outros interesses.

Resumo

Esta monografia busca aplicar *Deep Q Learning* ao jogo Tetris, da maneira mais generalista possível. Evita-se ao máximo utilizar heurísticas que sejam produtos da forma de pensar humana nesse jogo, a fim de fazer com que o agente esteja livre para explorar outras soluções viáveis no espaço de busca, que costuma ser bastante restritivo visando uma convergência rápida. Para isso, utiliza-se uma rede neural convolucional, que recebe como entrada o tabuleiro inteiro e a peça atual e retorna uma ação composta, isto é, uma sequência de rotações seguida de uma sequência de translações horizontais seguidas de uma queda brusca. Essa rede é inicialmente alimentada com dados que indicam uma sequência curta de jogadas até fechar 1 linha em diferentes estados de jogo. Após isso, ela passa por um treinamento de aprendizado por reforço profundo, em que joga diversos jogos do início ao fim, divide as jogadas em conjuntos que finalizam com o fechamento de linhas e utiliza os conjuntos classificados como melhores para o retreinamento. Esse ciclo é repetido diversas vezes tentando melhorar o *score* médio obtido nos jogos.

Palavras-chave: Aprendizado por reforço profundo, Redes neurais convolucionais, Tetris.

Abstract

This monography applies Deep Q Learning to the game Tetris, taking the most generalistic approach possible. We avoid using heuristics that are a product of the human way of thinking this game, so that the agent is free to explore other viable solutions in the search space, which is usually very restrictive to guarantee fast convergence. For this, a convolutional neural network is used, that receives as an input the whole board and the current piece and outputs a composite action, that is, a sequence of rotation followed by a sequence of horizontal translations followed by a hard drop. This neural network is initially fed data that indicate a small sequence of actions until it finishes one line in different game states. After that, it goes through a deep reinforcement learning training, in which it plays a batch of games from start to finish, divides the plays in sets that end with finishing one or more simultaneous lines and then uses the sets ranked as best for retraining. This cycle is repeated multiple times trying to improve the mean score obtained in the games.

Keywords: Deep Q Learning, Deep reinforcement learning, Convolutional neural networks, Tetris.

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

DQN	*Deep Q Learning*
MLP	*Multi-Layer Perceptron*
CNN	*Convolutional Neural Network*

Lista de Símbolos

$f(T, p)$	função ideal que indica um *score* potencial para cada possível posição da peça p no tabuleiro T
$Q(T, p)$	função representada por uma rede neural que busca aproximar f
R_S	recompensa do segmento S
R_j	recompensa da jogada j
γ	fator de desconto de jogadas futuras
Métrica m	medida da capacidade da rede de fechar uma linha em um tabuleiro que permita isso

Sumário

Capítulo 1

Introdução

Tetris é um jogo extremamente popular e um dos clássicos dos jogos eletrônicos desde sua criação. Criado em 1984 pelo engenheiro de software soviético Alexey Pajitnov, originalmente para o computador, também soviético, Electronika 60, e posteriormente para dezenas de plataformas diferentes, desde o *Nintendo Entertainment System* até o *Play Station 5*. Tal fama ao decorrer de quase 40 anos revela sua atemporalidade e alto potencial de rejogabilidade, com cada jogo se tornando rapidamente diferente de todos os anteriores, e essas diferenças gerando uma necessidade de mudança de estratégia constante.

O jogo consiste em um tabuleiro com 20 linhas por 10 colunas, inicialmente vazio, e uma peça, escolhida aleatoriamente entre 7 possíveis, todas com 4 blocos arranjados de todas as formas contíguas não redundantes possíveis, no topo, caindo lentamente. O jogador tem a tarefa de colocar essa peça no tabuleiro, a rotacionando e transladando conforme julgar necessário, de modo a evitar que as peças se empilhem até o topo e, consequentemente, gere um *gameover*. Sempre que uma peça toca o fundo do tabuleiro ou a pilha de peças já colocadas, uma nova surge no topo.

O objetivo maior, além de apenas sobreviver, é colocar as peças de modo que uma ou mais linhas fiquem completas, isto é, preenchidas totalmente por peças, sem nenhum buraco. Ao realizar isso, a linha é removida e a pontuação do jogador aumentada de acordo com a quantidade de linhas completadas de cada vez, de tal forma que completar 2 simultaneamente gera uma pontuação maior que completar 1 linhas 2 vezes. O mesmo se aplica também a qualquer outra combinação, fazendo com que a melhor pontuação possível seja adquirida ao completar 4 linhas de uma só vez, a maior quantidade possível no jogo.

Com essas mecânicas simples, combinadas com elementos de aleatoriedade e sem fim definido, Tetris acaba por demandar estratégias moderadamente complexas sob uma pressão de tempo cada vez maior para conseguir uma pontuação elevada. Todos esses elementos tornam esse jogo bastante interessante para a área de inteligência artificial: como modelar um algoritmo que consiga uma pontuação boa? Especialmente considerando que o jogo não possui um fim definido, dependendo somente das decisões do jogador.

Vários algoritmos, com sucesso relativamente bom, na casa de centenas de milhares de linhas completas em um único jogo, já foram feitos com esse fim, porém a imensa maioria se baseia em heurísticas implementadas à mão, isto é, carregando as pré-noções humanas do jogo, sem deixar que estratégias possivelmente ainda melhores emerjam das suas mecânicas postas à prova. Este trabalho busca uma abordagem diferente, utilizando como entrada de uma rede neural o tabuleiro de tetris sem nenhum pré processamento, evitando utilizar heurísticas óbvias, que embora possam ser ótimas não necessariamente são.

A ideia principal por trás disso vem da inteligência artificial AlphaZero, que, no xadrez, após treinar exclusivamente jogando contra si mesma e aprendendo quais jogadas levariam a melhores resultados, foi capaz de ganhar dezenas de partidas sem perder nenhuma contra o até então melhor programa Stockfish, que utiliza heurísticas criadas por cientistas da computação e grãomestres de xadrez. Isso mostra como, mesmo em um jogo em que humanos possuem séculos de teorias e estratégias, mesmo treinando uma máquina para seguir, sem falhas, essas teorias, ainda assim há muito a ser melhorado e descoberto utilizando esses algoritmos de aprendizado por reforço.

É possível categorizar este trabalho como um estudo de um problema de brinquedo. Obviamente, não há nenhuma aplicação prática para isso, porém é a partir da análise de problemas como esse que, quando destrinchado, abrem-se portas para diversas formas diferentes de entender as aplicações utilizadas. Aqui, no caso, algumas das ideias a se ter em mente são: aplicação de métodos de aprendizado por reforço para problemas com componentes aleatórios, utilização de redes neurais não-recursivas para realização de planejamentos ao decorrer do tempo, utilização de redes neurais convolucionais para fins lógicos, resolução de problemas de classificação através de redes de regressão e modelagem de aprendizado por reforço para essas redes.

Capítulo 2

Fundamentação

2.1 Redes Neurais

Redes neurais são compostas por nodos conectados por ligações direcionadas. Uma ligação do nodo i para o nodo j serve para propagar a ativação a_i de i para j. Cada ligação também tem um peso numérico $w_{i,j}$ associado, que determina a força e o sinal da conexão (Russell et al., 2010). Essas estruturas têm se tornado a base do campo de aprendizagem profunda, se mostrando capazes de inferir conexões entre elementos que são muito difíceis de se colocar em código manualmente. Redes neurais, em sua forma mais tradicional, um *Multi-Layer Perceptron* ou *MLP* (Murtagh, 1991), consistem de estruturas matemáticas compostas por nodos agrupados em camadas em série. Cada nodo da primeira camada representa uma entrada da rede, e cada um de camadas subsequentes realizam operações com pesos e desvios sobre todos os nodos da camada anterior e passam para a próxima. Isso é realizado repetidamente até alcançar a camada de saída.

A princípio, como geralmente são inicializadas de modo aleatório, o resultado da saída ainda não representada nada. Porém, ao disponibilizar uma base de dados que indique entradas e as saídas respectivas, a rede, através de um processo chamado de retropropagação, reajusta seus pesos de modo que os valores que ela gera dada uma entrada sejam cada vez mais próximos dos esperados. É notório que essa estrutura possui grande capacidade de generalização, não necessitando já ter sido treinado com todos os valores possíveis para ser capaz de inferir saídas próximas das esperadas.

A seguir são descritos alguns conceitos de redes neurais mais específicos que serão utilizados nesta monografia.

2.1.1 Camada Densa

Uma camada densa é, simplesmente, uma camada de nodos dentro de uma rede neural em que cada nodo dela se conecta a todos os nodos da camada anterior. É a camada mais comum utilizada em redes neurais e é a única camada, além da de entrada, utilizada no *Multi-Layer Perceptron*, que foi discutido nesta seção.

2.1.2 *Overfitting*

Overfitting, em ciência de dados, se refere ao cenário em que um modelo estatístico se alinha exatamente com os dados da base de treinamento, fazendo com que seja incapaz de inferir o valor de dados fora dessa base, o tornando inútil. Em redes neurais, isso muitas vezes ocorre

pois ou teve dados de treinamento em demasia de poucos ou de um único caso da aplicação ou porque possui uma taxa de aprendizado alta demais.

2.1.3 Taxa de Aprendizado

Taxa de aprendizado é uma constante entre 0 e 1 que deve ser definida para cada rede. Ela define quanto os pesos de cada conexão serão reajustados durante a retropropagação, com valores altos levando a um cenário de *overfitting* e valores muito baixos levando a um treinamento extremamente lento e que demanda quantidades enormes de dados. Muitas vezes, qual valor é bom para determinada rede se mostra bastante dependente da aplicação em que será utilizada, e encontrá-lo é de suma importância para que a rede seja capaz de fornecer a generalização para o problema.

2.1.4 Regressão ou Classificação

Um algoritmo de regressão busca encontrar uma função que correlaciona as variáveis de entrada e as de saída. Assim, dada uma nova entrada, o algoritmo aplica a função estimada e infere o valor da saída.

Já um algoritmo de classificação busca dividir o espaço definido pelas variáveis de entrada em múltiplas classes. Assim, dada uma nova entrada, o algoritmo retorna a classe que melhor estima que o novo dado faz parte.

2.2 Redes Neurais de Convolução

Uma rede neural de convolução, ou *CNN*, é uma subcategoria de rede neural, especificamente feita para lidar com entradas matriciais cuja relação de vizinhança entre os valores é de bastante relevância, muitas vezes mais do que qualquer relação global entre elas. Um exemplo de aplicação tradicional são em análise de imagens (LeCun et al., 1989).

Como as entradas dessas redes costumam ser muito grandes, são removidas conexões vistas como desnecessárias entre os nodos da camada anterior, e, em vez disso, as operações de determinado nodo são executadas somente sobre seus vizinhos dentro de certa janela, seja 3x3, 5x5, ou maior dependendo da aplicação. Essa estrutura é comumente chamada de filtro. Esse tipo de rede se mostra muito boa em encontrar padrões locais nessas matrizes, que podem ser estendidos cada vez mais para que o padrão se torne cada vez maior e mais complexo, podendo, novamente ser utilizado para generalizações de maneira bastante eficiente.

2.3 Aprendizagem por Reforço

Aprendizagem por reforço é um método de treinamento de *machine learning*, que geralmente é utilizado para agentes que agem sobre ambientes, utilizando recompensas e punições para encorajá-lo a realizar somente ações que levem a resultados desejados. Esses agentes devem escolher ações que ou recebem recompensas imediatas ou criem condições para que essas sejam alcançadas no futuro.

O maior problema é, porém, que as ações intermediárias entre o estado inicial e os estados desejados são, em geral, desconhecidas. Para que o agente seja capaz de reconhecer um caminho de ações que o recompense o máximo possível, buscando evitar punições, será utilizada

a Equação de Bellman, descrita a seguir, de modo a definir o valor de um determinado estado baseado nas recompensas imediatas somado ao valor do estado gerado.

$$V(s) = r + \gamma * max_{s' \in S'}(V(s')) \tag{2.1}$$

Onde:

- r é a recompensa imediata recebida;
- $\gamma \in [0, 1)$ é o fator de desconto para estado futuro, que serve para diminuir o valor do estado futuro em relação ao atual, para garantir que o agente tenda a se aproximar cada vez mais da recompensa;
- S' é o conjunto de todos os estados possíveis que podem ser atingidos a partir de uma única ação sobre o estado s.

Tendo, agora, os valores de cada estado recursivamente definidos, basta utilizá-los para o treinamento do agente e fazê-lo sempre escolher de maneira gulosa a ação que gere o estado que possui maior valoração.

Capítulo 3

Revisão da Literatura

Vários artigos demonstram o desenvolvimento de algoritmos que resolvem problemas similares ao que será discutido neste trabalho. Adiante, serão discutidos alguns deles, no que se assemelham e no que diferem do trabalho realizado aqui.

Böhm et al. (2005) utilizam de algoritmos genéticos a fim de determinar bons pesos para composições algébricas de atributos que refletiriam a qualidade de um determinado estado do tabuleiro. Isso é feito utilizando um total de 12 atributos distintos e 3 funções que os compõem. Os resultados são bastante expressivos, figurando na casa de milhões de linhas em um único jogo, porém a falta de distinção entre a simultaneidade da conclusão dessas linhas (isto é, quantas linhas foram fechadas em uma única jogada) indica que, comparado a um jogador humano, esse algoritmo não é tão eficiente na geração de pontos por peça colocada. Esse é um exemplo de algoritmo eficiente e funcional, porém que diverge bastante da abordagem deste trabalho: note que esses atributos implicam em uma visão sobre o jogo similar à que um humano teria. Isso faz com que o espaço de busca das estratégias seja restrito às mesmas que um humano teria, trazendo um certo viés, que de um ponto de vista de eficiência é ótimo, mas pode impedir que estratégias ainda melhores sejam exploradas. Aqui, buscar-se-á algo diferente, tentando evitar ao máximo a utilização dessas heurísticas para que surjam estratégias a partir somente das mecânicas diretas do jogo.

Algorta e Şimşek (2019) apresentam uma longa revisão da literatura sobre aplicações de aprendizado de máquina sobre o jogo Tetris. É notória a dominância de estratégias que envolvem a utilização de algoritmos genéticos, alguns chegando até a centenas de milhões de linhas concluídas, porém algoritmos de aprendizagem por reforço ainda continuam melhorando e começaram a chegar próximo do mesmo nível de magnitude. Outra importante coisa a se perceber é como a imensa maioria dos artigos publicados não aplica diferenciação entre linhas fechadas sequencialmente ou simultaneamente, dando a elas o mesmo *score*. Como já foi dito, dado que nenhuma das versões jogadas atualmente usa esse sistema, além de ser significativamente menos complexo, neste trabalho será tomada uma abordagem distinta, buscando maximizar não só a quantidade de linhas como também a pontuação por peça.

Lin (1992) detalha a base do método de aprendizado por reforço profundo utilizado neste trabalho: *Deep Q Learning*. Esse algoritmo consiste de um problema com um agente e um ambiente, em que cada ação do primeiro cria uma modificação no segundo. Dado isso, é necessário que o algoritmo não só calcule a melhor ação imediata, mas que também gerará um ambiente potencialmente mais benéfico para o agente no futuro. Dessa forma, define-se a função que representa o valor de um estado recursivamente, em geral visando alguma grande recompensa futura, ou evitando alguma punição.

Arulkumaran et al. (2017) realizam uma revisão dos preceitos e das aplicações de aprendizado por reforço profundo, sendo uma das principais a de interações com ambientes que retornam imagens. Esse cenário é bastante parecido com o analisado nesta monografia, visto que também é matricial e muitas vezes também utiliza redes neurais convolucionais. Já Jang et al. (2019), trazem uma boa fundamentação conceitual e base matemática para *Deep Q Learning*, ou *DQN*, que é o algoritmo de aprendizado por reforço profundo utilizado aqui.

Ambos os artigos fornecem uma forte base teórica genérica sobre esse tipo de aprendizado, que tanto ajuda a formular as ideias utilizadas para a resolução do problema como ajuda a explicar alguns comportamentos imprevistos do agente.

Stevens e Pradhan (2016) propõem-se a criar um método para que redes neurais convolucionais consigam aprender a jogar Tetris. Também utiliza aprendizado por reforço, usando somente equação de Bellman para calcular a diferença entre a saída da rede e a saída esperada da função ideal de avaliação do estado do tabuleiro definida por eles, e priorizando o treinamento para os quais essa diferença é muito distinta. Isso incorre em um problema dada a forte recursividade na definição, fazendo com que a convergência seja bastante sensível aos valores iniciais.

Ainda, utilizam para a inicialização da rede uma heurística simples de avaliação do tabuleiro, usando atributos clássicos para calcular seu valor, já trazendo as ideias tradicionais de algoritmos determinísticos e esperando melhorá-los, porém sem muito sucesso. Neste trabalho, propõe-se um treinamento inicial menos enviesado com estratégias humanas, apenas focando em fechar linhas, e buscando melhorar a partir daí, a fim de potencialmente explorar novas.

Capítulo 4

Implementação

Foi criado um jogo de Tetris baseado na versão clássica de NES (*Nintendo Entertainment System*), com algumas pequenas modificações para facilitar o uso da rede neural[1]. A pontuação é gerada com a altura da queda de cada peça somado ao valor das linhas completadas, que crescem de forma cúbica de acordo com a quantidade feita simultaneamente: 400 para 1, 1000 para 2, 3000 para 3 e 12000 para 4. Há 7 peças ao todo, com 4 rotações cada, com a exceção das rotações que geram a mesma imagem, por exemplo a rotação de um quadrado. Por fim, foi implementado também um botão para queda brusca, que faz com que a peça seja imediatamente colocada na posição em que primeiro toca o fundo do tabuleiro ou outra peça a partir da coluna onde se encontrava e da sua rotação.

A forma com que é realizada cada jogada pelo agente é bastante simples: dá-se o tabuleiro e a peça atual para uma rede neural para que ela estime um *score* potencial de cada posição e rotação, e, a partir disso, apenas se escolhe aquela cujo *score* é maior. Efetivamente, a rede neural busca aproximar uma função ideal $f(T, p)$, onde T é o tabuleiro e p a peça atual, que indica um *score* potencial para cada uma das possíveis posições da peça nesse determinado tabuleiro. Faz-se isso definindo a rede como a função $Q(T, p)$ e a treinando a fim de que $Q(T, p) \approx f(T, p)$.

4.1 Regressão ou Classificação?

Embora o problema indique fortemente demandar uma rede de classificação, a final, busca-se qual a melhor jogada, o modelo utilizado é de regressão. Isso foi feito pois, devido à forma como o treinamento por reforço é realizado (utilizando a equação de Bellman), requer-se que haja um valor contínuo para cada jogada, para que os cálculos de valor de cada estado possam ser reajustados. Foi tentado implementar uma versão de classificação da rede, porém ela se mostrou incompatível com o treinamento utilizado.

4.2 Topologia

A topologia da rede consiste em 1 camada convolucional 2D, com 20 filtros de tamanho 3, cujo intuito é receber o tabuleiro e, através dos filtros, verificar a plauseabilidade de colocar cada peça em cada rotação naquela determinada coordenada. O número de filtros foi decidido justamente com esse critério: existem 19 formas possíveis de peça no tabuleiro, considerando a

[1]Código fonte disponível no link a seguir: `https://github.com/arthur-mant/tetris-ai`

peça e sua rotação. Em seguida, a saída dessa camada é transformada em uma camada densa e concatenada com um vetor que indica qual peça será jogada.

Essa camada, então, leva a mais uma, também densa porém significativamente menor, que, espera-se, servirá para conectar a noção de espaço disponível conseguida através da camada de convolução com a peça atual, gerando assim a melhor jogada. A saída, enfim, são 40 números em ponto flutuante, sendo cada uma das 40 referente a uma coluna do tabuleiro e uma rotação da peça, com os números sendo o *score* potencial gerado por aquela jogada.

Foram testadas topologias com mais camadas convolucionais e, também, com mais camadas densas, porém não apresentaram ganho de desempenho e decidiu-se pelo modelo mais simples por convergir mais facilmente.

4.3 Treinamento

A rede é treinada em 2 etapas: a primeira tem o objetivo de tornar ela minimamente familiar com o Tetris, fazendo com que ela consiga ao menos fechar linhas em alguns jogos a cada lote de treinamento. Após isso, é realizado um treinamento por reforço, que visa tomar como base esse conhecimento, aplicá-lo e, selecionando as melhores jogadas, aos poucos ir refinando a rede, de forma que cada vez mais se aproxime da função ideal.

4.3.1 Via base de dados

É gerada uma enorme quantidade de tabuleiros com ao menos 1 linha fechada, cerca de uma centena de milhar, e, então, são retiradas uma ou mais peças, de modo que ainda não tenha nenhuma linha fechada. Essas peças naquela coluna e rotação específicas serão indicadas para a rede como o movimento correto a ser feito. Assim, é gerado um pequeno subconjunto do universo de jogadas do jogo, em que a rede vê uma jogada razoavelmente boa ser feita, sempre finalizando ao menos uma linha. Por fim, alimenta-se a rede com esses dados. Essa etapa é executada algumas vezes até que se tenha uma rede viável para o próximo passo.

Tentou-se executar a próxima etapa sem antes passar por esse treinamento, porém o agente demonstrou um comportamento muito errático e linhas quase nunca eram fechadas, fazendo com que o treinamento por reforço não tivesse com o que trabalhar.

4.3.2 Treinamento por reforço

A rede, então, se põe a jogar diretamente do início ao fim do jogo, 100 vezes por lote. As partidas passam por um processo de segmentação: são separadas as jogadas entre o início da partida até se concluir uma ou mais linhas simultaneamente, dali até a próxima conclusão e assim por diante, até que o jogo seja perdido. Cada segmento tem uma recompensa calculada do seguinte modo:

$$R_s = \frac{\text{score gerado pela(s) linha(s)}}{\text{quantidade de jogadas}} \tag{4.1}$$

Favorecendo, assim, segmentos pequenos e recompensando aqueles que fecham uma grande quantidade de linhas de uma vez. O último segmento de cada jogo, isto é, o que não fecha nenhuma linha e leva a perder, possui uma enorme penalidade, de modo a desencorajar jogadas que levem a esse cenário.

Agora, utilizar-se-á a Equação de Bellman para determinar R_j, ou a recompensa de cada jogada em específico. Isso é feito utilizando a recompensa imediata da jogada r_j como o *score*

gerado por ela somado à recompensa do segmento a qual ela pertence. Ainda, é adicionado o termo $\gamma * Q(T', p')$, referente à estimativa do valor potencial do tabuleiro que ela cria.

$$R_j = score + R_s + \gamma * Q(T', p') \tag{4.2}$$

Onde γ é uma constante pré-definida menor que $1, 0, 99$ nesse caso, T' é o estado do tabuleiro após essa jogada, p' é a próxima peça e $Q(T', p')$ é a estimativa do *score* potencial do novo estado do tabuleiro.

Por fim, R_j é multiplicado por um fator global do jogo. Esse fator existe para recompensar mais jogadas de segmentos que pertencem a jogos cujo *score* foi maior, assim buscando penalizar sequencia que podem ter gerado bons resultados à primeira vista, porém comprometeram o tabuleiro a longo prazo, levando a uma derrota antecipada. O fator pertence ao intervalo entre 0 e 1, pois é calculado através da normalização pelo *score* máximo do lote. Ele é calculado da seguinte forma:

Se $R_j >= 0$:

$$Fator = \text{Pontuação do jogo atual/Maior pontuação do lote} \tag{4.3}$$

Se $R_j < 0$:

$$Fator = 1 - (\text{Pontuação do jogo atual/Maior pontuação do lote}) \tag{4.4}$$

Por fim:

$$R_{jfinal} = R_j * Fator \tag{4.5}$$

O fator é calculado dessa maneira para que, independente se a recompensa R_j é positiva ou negativa, jogadas pertencentes a jogos com maior pontuação sejam recompensadas, pois supõe-se que elas possuíram um impacto mais benéfico no jogo que as que pertencem a um jogo que perdeu mais cedo.

Por fim, é calculado $Q(T, p)$, sendo T o estado inicial do tabuleiro dessa jogada e p a peça dessa jogada, e substitui-se o valor calculado da ação tomada por R_j, gerando um Q'. Assim, tem-se uma nova tupla (T, p, Q') que será usada para o retreinamento da rede neural.

Após calculadas todas as novas tuplas, a rede neural é retreinada com esses novos dados. Essas operações são executadas uma enorme quantidade de vezes, determinada pela quantidade de episódios definida, que costuma figurar na casa das dezenas de milhares, buscando refinar a rede neural de modo que ela cada vez mais se aproxime da função f ideal.

Capítulo 5

Experimentos

Os experimentos buscam encontrar parâmetros para a rede neural que possibilitem um bom treinamento que, posteriormente, gere um bom *score*. O treinamento novamente será subdividido em 2: uma parte para o treinamento via base de dados e outra para o treinamento por reforço. As métricas usadas para avaliar a qualidade dos resultados serão o *score* médio de 500 jogos e o que se definirá como métrica m, tal que $m \in [0, 1]$, que representa a capacidade da rede reconhecer a jogada que fechará uma linha quando diante de um tabuleiro capaz disso. Para essa avaliação será utilizada uma base de dados gerada da mesma forma que os dados que servem para o primeiro treinamento da rede.

Note que não necessariamente um aumento nessa métrica indica uma melhora na performance, por exemplo imagine uma posição que possui toda a estrutura pronta para um tetris, isto é, fechar 4 linhas simultaneamente, porém a peça atual não é a peça necessária para executar isso. Logicamente, a melhor jogada seria *não* completar ainda, e esperar pela peça adequada, porém isso incorre numa diminuição da métrica m dessa rede. Ainda assim, ela é considerada adequada para a avaliação de redes ainda pouco treinadas e que, antes de tudo, precisam se mostrar capazes de fechar linhas uma a uma, e será utilizada nesse contexto.

5.1 Treinamento via base de dados

Serão testados diferentes valores para as seguintes variáveis, a fim de conseguir uma rede neural que maximize o *score* e a métrica m ao final dessa etapa tanto quanto possível:

- Quantidade de lotes iniciais, que define quantos lotes serão usados para esse treinamento;
- Quantidade de épocas iniciais, que define quantas épocas serão usadas por lote;
- Tamanho do lote inicial, que define quantas tuplas (tabuleiro, peça, jogada) serão utilizadas por lote;
- Taxa de aprendizado inicial, que define o *learning rate* da rede neural nessa etapa do treinamento.

A seguir, seguem as tabelas com o *score* médio de 500 jogos e a métrica m resultantes dos experimentos para cada alteração na variável. Cada valor foi escolhido com base em experimentação, no que parecia razoável para a rede mas que não tornasse o treinamento demasiadamente lento. Caso a variável não seja mencionada, ela possuirá seu valor padrão, que é:

- Lotes = 10
- Épocas = 500
- Tamanho = 100000
- Taxa de Aprendizado = 10^{-7}

5.1.1 Quantidade de lotes iniciais

Lotes	*Score*	Métrica *m*
5	359,464	0,2256
10	318,626	0,2512
20	331,206	0,2721

Tabela 5.1: Tabela de lotes iniciais

Nota-se, aqui, o que já era de alguma maneira esperado: a métrica *m* tende a aumentar, embora modestamente, com o aumento do número de lotes iniciais. O único valor que distoa das expectativas iniciais é o *score* de 5 lotes, que foi muito maior que o esperado, maior até do que o de 20, porém, nessa etapa, priorizar-se-á a métrica *m*, e, portanto, o valor de 20 será considerado o melhor.

5.1.2 Quantidade de épocas iniciais

Épocas	*Score*	Métrica *m*
100	341,222	0,1821
500	318,626	0,2512
1000	320,05	0,2846

Tabela 5.2: Tabela de épocas iniciais

Segue-se o mesmo padrão que na tabela anterior: o aumento da quantidade de épocas gera um aumento na métrica *m* e, como essa é a métrica mais relevante para essa etapa, 1000 épocas é escolhido como o melhor valor para essa variável. Porém, assim como na anterior, é interessante notar o *score* da menor quantidade de épocas ser o maior, talvez pelo componente aleatório do sistema ou talvez pelo fato de que essa faixa de score ainda é fortemente afetada pela quantidade de peças colocadas, e não tanto pela quantidade de linhas fechadas.

5.1.3 Tamanho do lote inicial

Tamanho	*Score*	Métrica *m*
50000	346,526	0,2219
100000	318,626	0,2512
200000	317,926	0,2812

Tabela 5.3: Tabela de tamanho do lote inicial

O padrão se repete aqui também, e o mesmo dito sobre as variáveis anteriores se aplica aqui. O tamanho de lote igual a 200000 será, portanto, tido como o melhor, justamente por ter-se como principal métrica a métrica *m*.

5.1.4 Taxa de aprendizado inicial

Taxa de aprendizado	*Score*	Métrica *m*
10^{-5}	426,986	0,5006
10^{-6}	402,586	0,3144
10^{-7}	318,626	0,2512
10^{-8}	224,726	0,1036

Tabela 5.4: Tabela de taxa de aprendizado inicial

Essa variável é, provavelmente, a mais importante entre todas as testadas, e seus resultados comprovam isso. Uma métrica *m* absurdamente grande de, aproximadamente, 0,5 foi atingida aumentando a taxa de aprendizado para 10^{-5}, que será o valor tido como melhor para as próximas etapas. Aqui, não há nenhuma restrição de tempo para a mudança do valor, mas ainda assim, embora ele possa ser alterado para qualquer um, e parecer haver uma tendência de crescimento tanto do *score* quanto da métrica *m* quando se aumenta a taxa de crescimento, decidiu-se parar nessa, pelo seguinte motivo: essa métrica não resulta diretamente na qualidade de um jogo, apenas na capacidade de fechar linhas, então é fácil imaginar que uma rede com ela muito alta estaria preocupada demais em fechar linhas e acabaria não realizando um bom trabalho em preparar as linhas a serem fechadas, ou evitar perder, entre outros aspectos importantes do jogo. Além disso, posteriormente ela será testada em conjuntos com ainda mais dados, devido às mudanças nas outras variáveis, o que poderia tornar esse problema ainda pior.

5.1.5 Melhor resultado obtido no treinamento via base de dados

Utilizando os experimentos acima como base, é fácil ver que a maior diferença, tanto no *score* quanto na métrica *m*, é causado pela variação na taxa de aprendizado, em particular quando é igual a 10^{-5}. Por isso, todos os próximos experimentos serão realizados com essa taxa de aprendizado.

Contudo, também há diferenças significativas causadas pelas outras variáveis: essa métrica tende a subir junto com a quantidade de lotes, de épocas ou do tamanho do lote inicial. Infelizmente, todas essas mudanças aumentam a demanda de tempo de processamento e de maneira exponencial, então serão testadas todas as mudanças novamente, agora com a taxa de aprendizado igual a 10^{-5}, e escolhido somente a que apresentar a melhor métrica *m*. Isso será feito pois foi julgado que o aumento de uma única varíavel é um compromisso adequado entre eficiência no treinamento e o tempo demandado.

A tabela a seguir indica os resultados obtidos ao utilizar a rede com a nova taxa de aprendizado e mudar uma ou nenhuma variável por vez, mantendo as outras como o valor padrão indicado no começo dessa subseção.

Mudança	*Score*	Métrica *m*
Nenhuma	426,986	0,5006
Lotes = 20	472,098	0,5402
Épocas = 1000	423,762	0,4939
Tamanho do lote = 200000	468,444	0,5541

Tabela 5.5: Tabela dos melhores resultados via base de dados

Daqui, conclue-se que o melhor parâmetro testado foi Tamanho do lote = 200000, usando a métrica *m* ainda como fator de decisão, e uma rede neural treinada com essa configuração será utilizada para todos os experimentos seguintes.

5.2 Treinamento por reforço

Nesta subseção serão testadas algumas variáveis referentes ao aprendizado da rede por reforço, de forma que se consiga um bom *score* ao fim do treinamento ou que a curva de treinamento por episódio implique em um ganho razoável de performance. Cada episódio considerado nesse experimento consiste em um jogo de tetris, jogado do início até perder. As variáveis serão as seguintes:

- Fração dos segmentos, que define a fração dos melhores segmentos a serem utilizados para o treinamento. Quanto menor, mais elitista será a política de seleção desses segmentos.
- Épocas por lote, que define quantas épocas serão aplicadas no treinamento da rede a cada lote de treinamento;
- Taxa de Aprendizado, que define o *learning rate* da rede durante a aprendizagem por reforço.

A seguir, dispõe-se gráficos referentes à evolução do treinamento, utilizando a métrica *m* e a quantidade média de linhas completas por jogo (separadas em uma curva para cada quantidade fechada simultaneamente), seguidos das tabelas com o score médio, a métrica *m* e a média de linhas completas final, resultantes das alterações em cada variável, utilizando 25000 episódios no total. Será utilizada uma única rede pré-treinada como base, no lugar do treinamento via base de dados, utilizando os melhores parâmetros conseguidos na etapa anterior. Todas as variáveis referentes ao treinamento por reforço que não forem mencionadas em determinada tabela estarão com seu valor padrão, que são:

- Fração dos segmentos = 0,2;
- Épocas por lote = 50;
- Taxa de aprendizado = 10^{-7}.

Note que os gráficos 5.2, 5.5 e 5.9 são idênticos. Isso é intencional, pois todos se referem às mesmas variáveis de treinamento, mas foi incluído uma vez em cada subseção para que possa ser comparado e analisado com as mudanças específicas de cada uma dessas subseções.

5.2.1 Fração dos segmentos

Gráficos

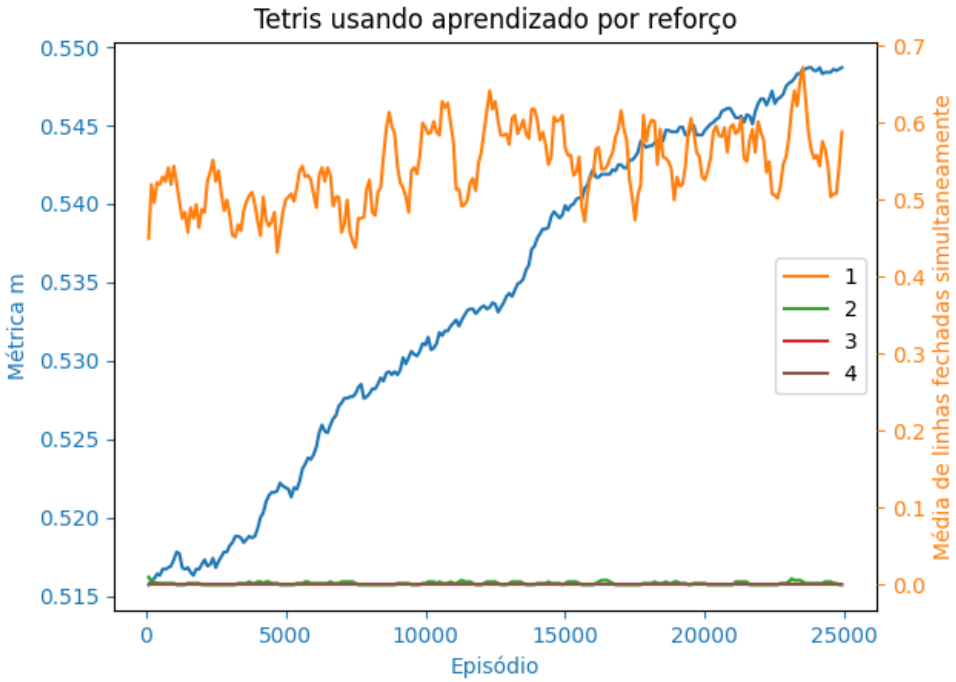


Figura 5.1: Curva de aprendizado com fração dos segmentos = 0,1

O gráfico 5.1 se mostra relativamente promissor: possuindo uma curva da métrica *m* sempre ascendente e aumentando em valores expressivos, e a média de linhas únicas fechadas parece aumentar levemente ao decorrer do treino. Infelizmente, uma quantidade muito pequena, quase desprezível, de linhas duplas simultâneas foram fechadas, e nenhuma tripla ou quádrupla.

Reitera-se que a métrica *m* utilizada não é necessariamente a melhorar forma de analisar a qualidade de um agente. Porém, dado o desempenho baixo deste, ela ainda se mostra bastante eficaz nisso, pois pelo menos se garante que o agente consegue fechar linhas únicas e não entrou em um cenário de *overfitting* que acabaria com essa capacidade.

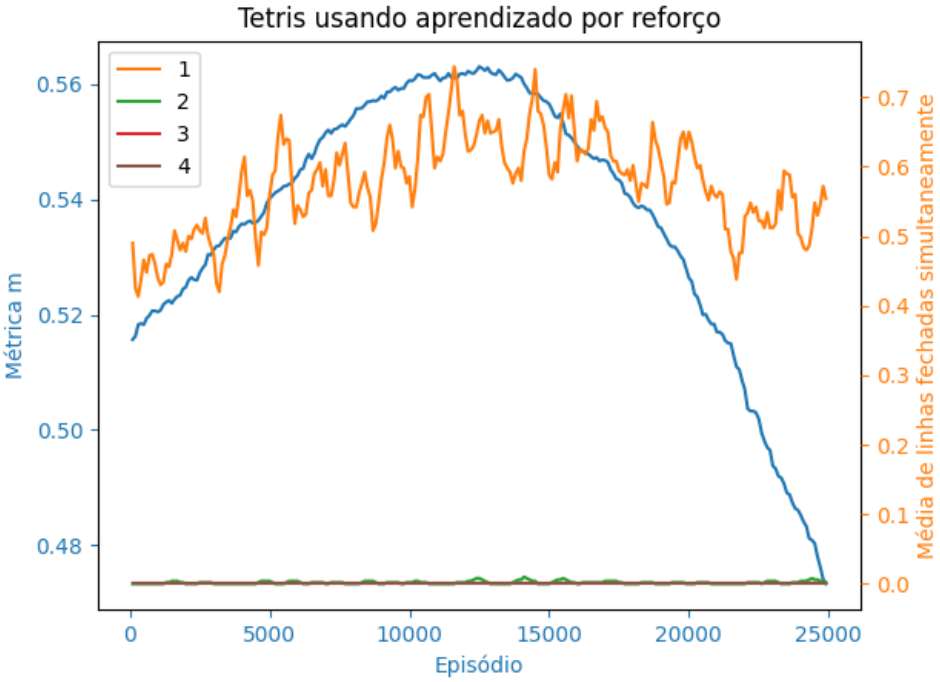


Figura 5.2: Curva de aprendizado com fração dos segmentos = 0,2

Já em 5.2 nota-se o começo de uma queda na métrica *m*, sendo acompanhada da média de linhas únicas. Isso ocorre pois, ao duplicar a variável de fração dos segmentos, duplica-se também os dados de treinamento que serão utilizados pelo agente. Apesar de esse relaxamento no elitismo da escolha dos segmentos aumentar a variedade de cenários, a baixa qualidade dos exemplos e sua grande quantidade geram o que aparentam ser sintomas de um cenário de *overfitting*.

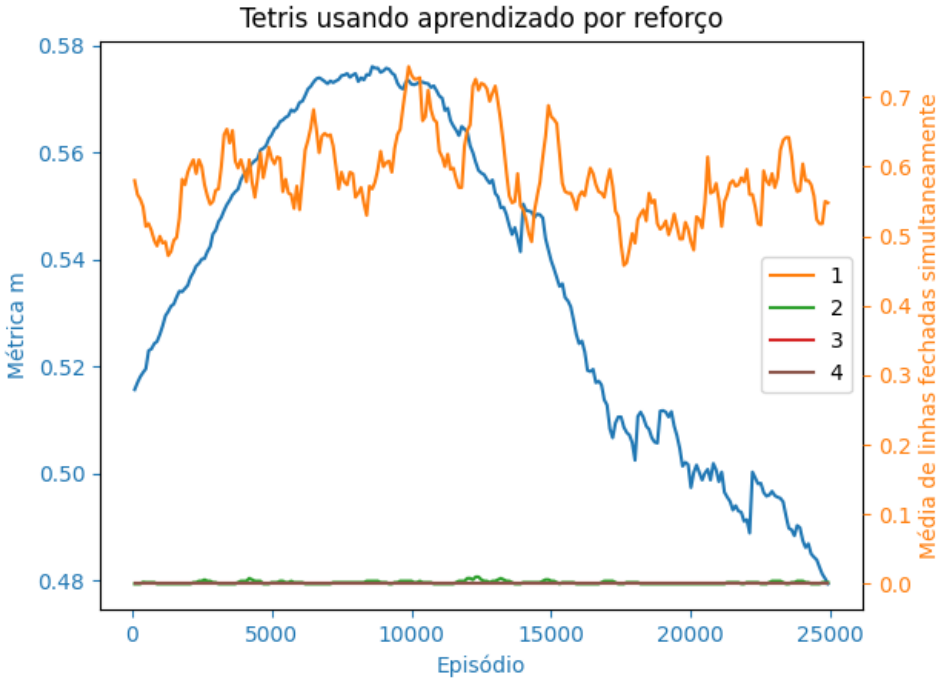


Figura 5.3: Curva de aprendizado com fração dos segmentos = 0,3

Esperaria-se que, em 5.3, o *overfitting* seria ainda mais drástico que em 5.2, mas surpreendentemente não. Quanto à métrica *m*, ela possui tanto o pico quanto o vale bastante próximos, e até mais altos que no gráfico anterior, e o seu formato é bastante parecido, apesar de mais íngreme e menos regular. Já para a média de linhas únicas, ela é um pouco mais alta que a de 5.2, embora também indique um sinal de queda junto com a métrica *m*.

Essa melhora bastante tímida talvez ocorra pela adição de segmentos que falham em completar uma linha, já que a política elitista nesse ponto já está muito relaxada. Esses segmentos possuem um score bastante negativo, e servem como reforço negativo para o agente, o que até então não se tinha.

Resumo dos resultados

Fração	*Score*	Métrica *m*	Linhas simultâneas			
			1	2	3	4
0,1	522,6	0,5487	0,5771	0,0020	0,0	0,0
0,2	499,3	0,4725	0,5270	0,0020	0,0	0,0
0,3	518,1	0,4822	0,5691	0,0040	0,0	0,0

Tabela 5.6: Tabela de fração dos segmentos

Podemos, enfim, analisar os números do final do treinamento, e fica fácil ver que o melhor valor para a variável fração dos segmentos é igual a 0,1, seguido de perto por 0,3 e só depois 0,2. Utilizar-se-á, portanto, SF = 0,1 durante a próxima subseção.

5.2.2 Épocas por lote

Gráficos

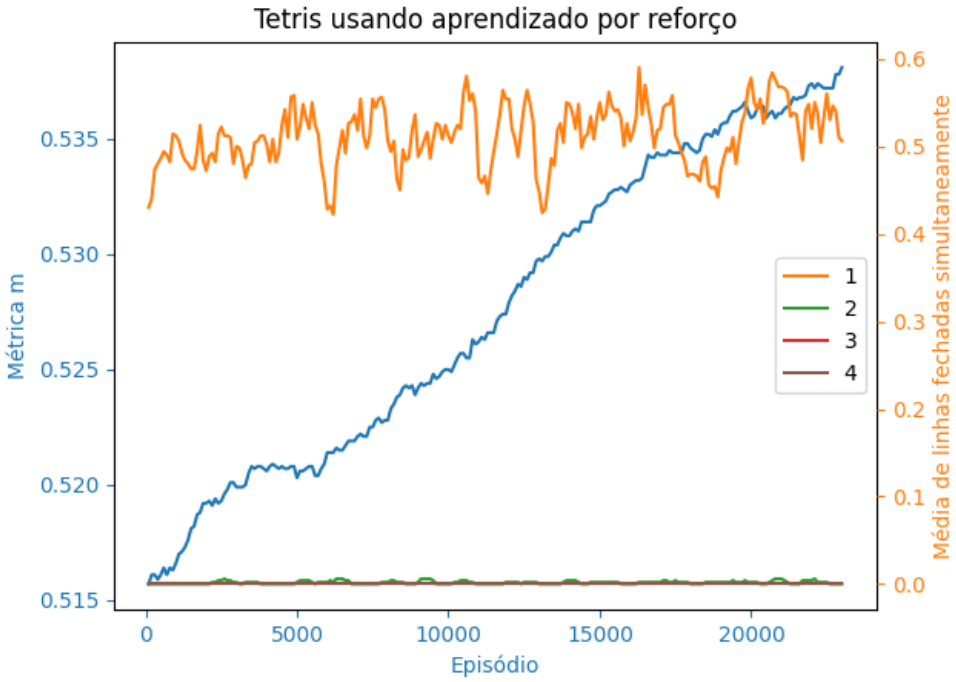


Figura 5.4: Curva de aprendizado com épocas por lote = 10

Têm-se, na figura 5.4, mais um gráfico promissor: sem sinal de *overfitting*, com a métrica *m* crescendo de forma quase linear e até a média das linhas únicas completas aparentam crescer aos poucos. Provavelmente isso ocorre devido à menor quantidade de vezes que os dados são utilizados para o treinamento, fazendo com que, embora mais lento, a rede neural aprenda com dados mais variados.

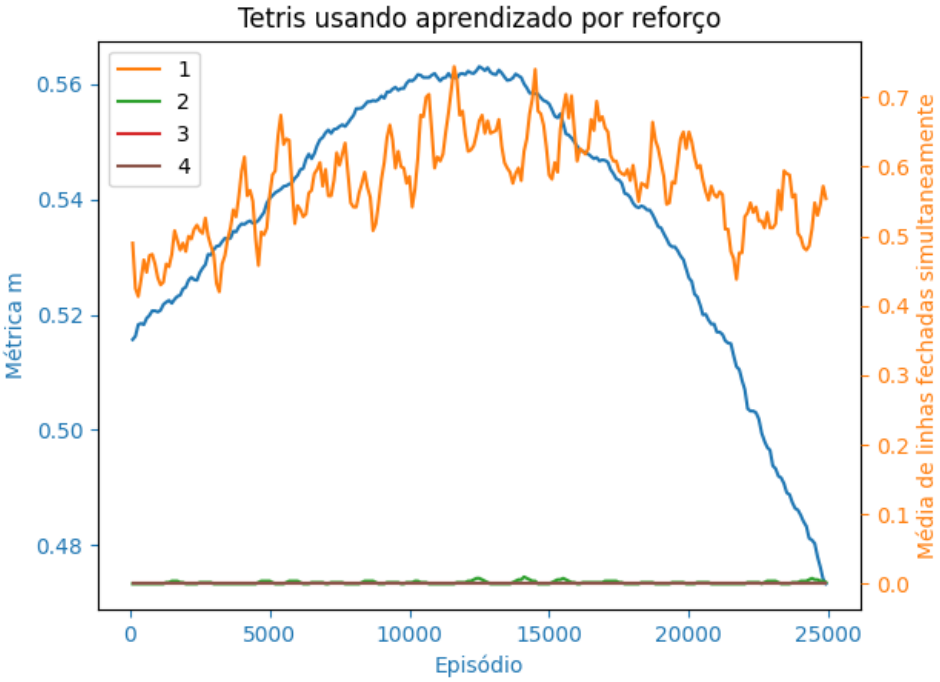


Figura 5.5: Curva de aprendizado com épocas por lote = 50

Já em 5.5 têm-se novamente o início de um caso claro de *overfitting*: tanto a métrica *m* quanto a média das linhas únicas completas atingem um pico e, após ele, iniciam uma queda cada vez mais brusca. Os dados de treinamento aqui são repetidas vezes demais, e a falta de variedade fere a capacidade de generalizar da rede, e aos poucos vai se tornando cada vez pior.

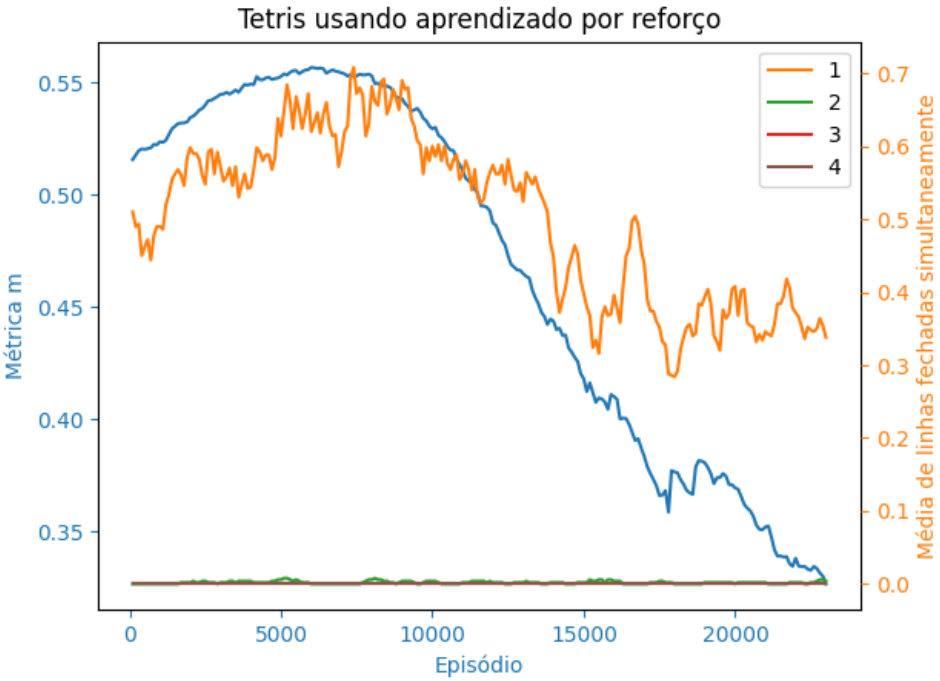


Figura 5.6: Curva de aprendizado com épocas por lote = 100

Em 5.6, mais uma vez temos um caso de *overfitting*, dessa vez ainda mais crítico, com a queda sendo ainda mais aguda nas 2 variáveis calculadas. As razões são as mesmas que as anteriores, somente mais intensas, visto que a variável que determina quantas vezes cada dado é utilizado está ainda maior.

Resumo dos resultados

Épocas	*Score*	Métrica *m*	Linhas simultâneas			
			1	2	3	4
10	501,0	0,5379	0,5410	0,0	0,0	0,0
50	499,3	0,4725	0,5270	0,0020	0,0	0,0
100	391,1	0,3241	0,3326	0,0020	0,0	0,0

Tabela 5.7: Tabela de épocas por lote

Essa tabela é bem direto ao ponto: 50 épocas por lote parece ser demais, 100 é pior ainda, 10 épocas por lote é o melhor valor dos três analisado, e, portanto, será utilizado na próxima seção.

5.2.3 Taxa de Aprendizado

Gráficos

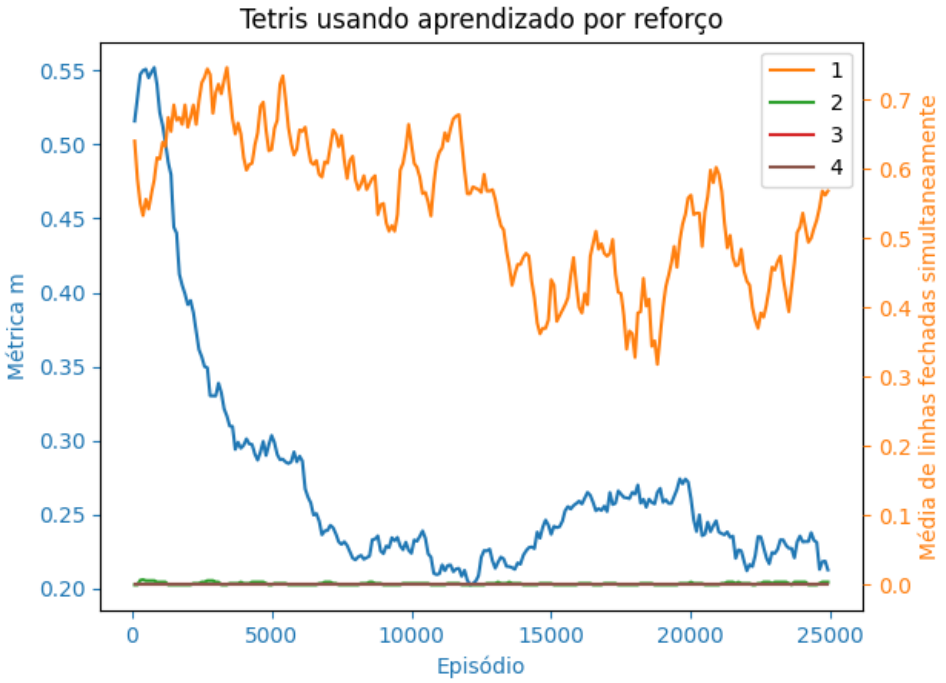


Figura 5.7: Curva de aprendizado com taxa de aprendizado = 10^{-5}

A figura 5.7 apresenta o *overfitting* mais intenso até agora, gerando uma queda gigantesca na métrica *m* e, embora menor, uma grande queda na média de linhas únicas completas. O curioso,

porém, é que após esses vales as métricas voltam a subir, gerando resultados até razoáveis, que podem ser vistos mais a frente. Isso pode indicar que, apesar de esquecer boa parte do seu pré-treinamento, essa rede começa a aprender coisas novas, e potencialmente se torne boa em resolver esse problema.

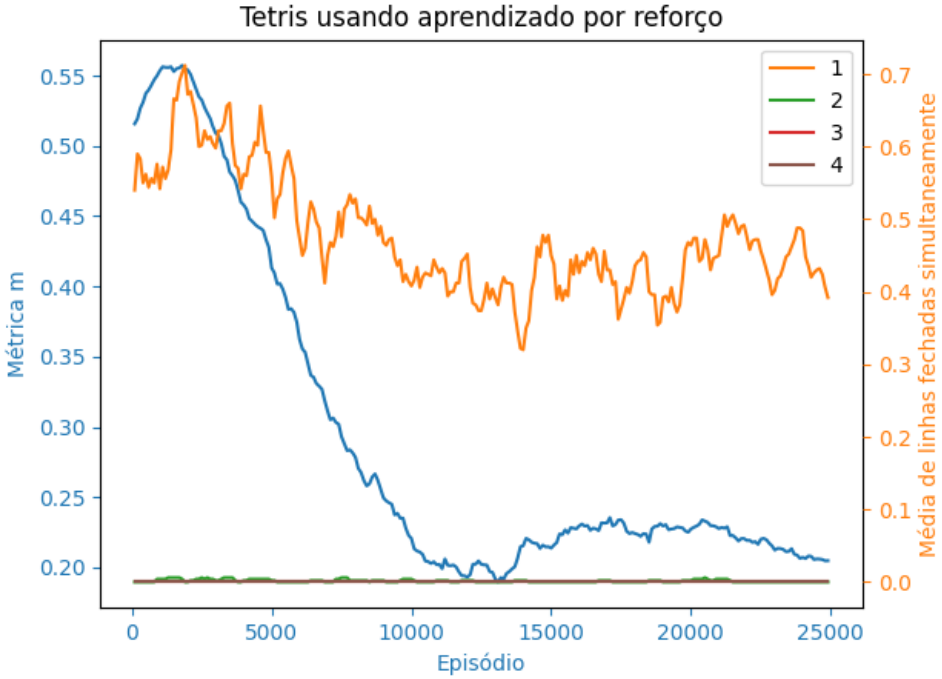


Figura 5.8: Curva de aprendizado com taxa de aprendizado = 10^{-6}

No gráfico 5.8 têm-se uma versão mais branda do gráfico 5.7: ainda assim um *overfitting* extremo, seguido de uma leve ascenção das métricas nos episódios mais próximos do fim. Porém, essa ascenção é bastante modesta, e não há nenhuma garantia de que aumente do mesmo modo que a anterior

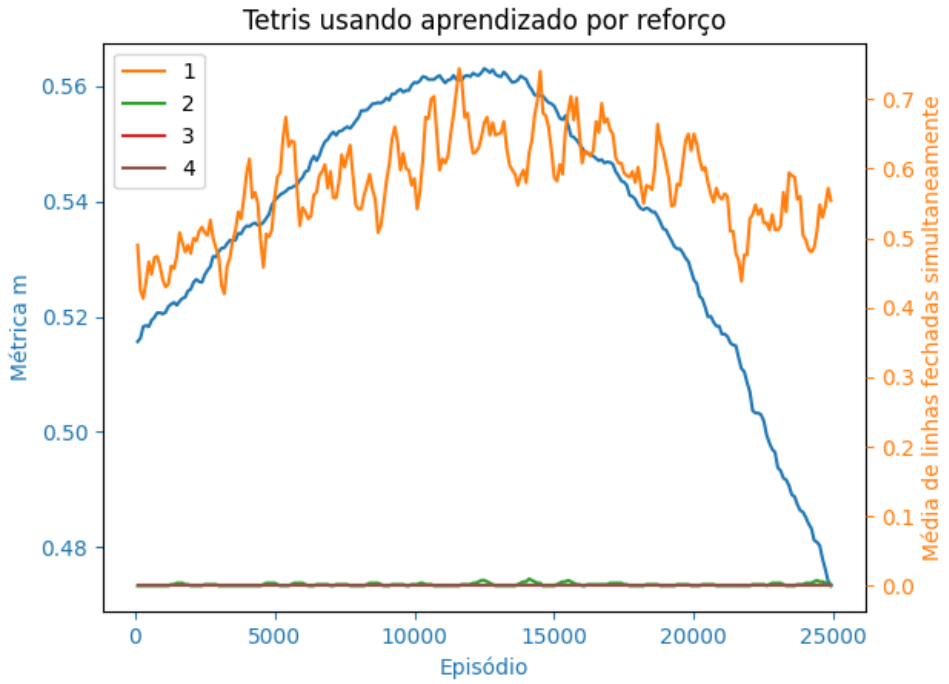


Figura 5.9: Curva de aprendizado com taxa de aprendizado = 10^{-7}

Já em 5.9 temos de novo um caso óbvio de *overfitting*, sem nenhum crescimento expressivo posterior. A queda não é tão expressiva, mas ainda indica que um treinamento mais longo incorreria no seu agravamento dada todas as outras variáveis iguais.

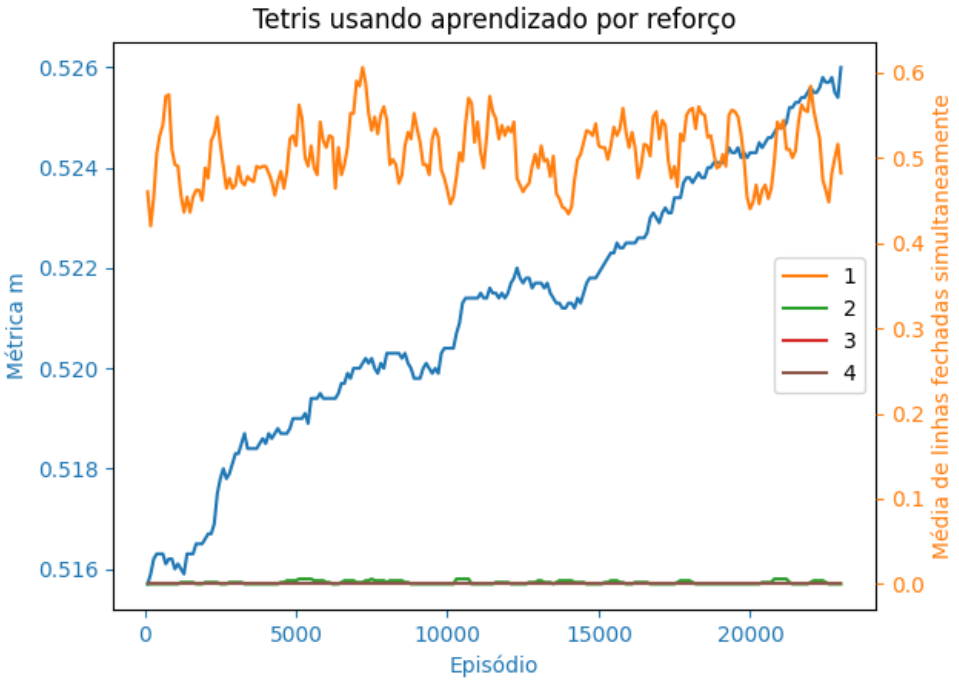


Figura 5.10: Curva de aprendizado com taxa de aprendizado = 10^{-8}

Em 5.10, a taxa de aprendizado já se encontra demasiadamente baixa, gerando, sim, um crescimento na métrica *m*, mas é bastante modesto, e a média de linhas únicas completas parece estar estagnada.

Resumo dos resultados

Taxa de aprendizado	*Score*	Métrica *m*	Linhas simultâneas			
			1	2	3	4
10^{-5}	498,3	0,2164	0,5230	0,0	0,0	0,0
10^{-6}	459,3	0,2027	0,4368	0,0	0,0	0,0
10^{-7}	499,3	0,4725	0,5270	0,0020	0,0	0,0
10^{-8}	505,4	0,5259	0,5531	0,0020	0,0	0,0

Tabela 5.8: Tabela de taxa de aprendizado

Os resultados dessa seção são, com certeza, os mais intrigantes. O melhor valor se for olhada somente a tabela é com LR = 10^{-8}. Porém, fica claro ao olhar seu gráfico que esse valor é extremamente lento para aumentar seu desempenho, então não é muito interessante.

Tanto LR = 10^{-5} quanto 10^{-6} possuem um comportamento similar e bastante curioso, contudo o fato da sua métrica *m* ter caído tanto e a média de mais de 1 linha simultânea permanecer 0 é preocupante. Além disso, esses valores mais altos já se mostraram bastante vulneráveis a gerar *overfitting*, o que poderia ocorrer novamente no futuro.

Por fim, LR = 10^{-7} tem seu próprio caso óbvio de *overfitting*, porém as outras variáveis escolhidas foram justamente por que se mostraram capazes de evitar esse exato caso. Além disso, também apresenta os segundos melhores resultados na tabela. Por ambos esses motivos, essa taxa de aprendizado será escolhida para a próxima subseção.

5.3 Resultados obtidos utilizando os valores previamente escolhidos

Uma nova rede neural será treinada utilizando as variáveis do treinamento por reforço que levaram aos melhores resultados, como discutido na subseção anterior, que são:

- Fração dos segmentos = 0,1;
- Épocas por lote = 10;
- Taxa de aprendizado = 10^{-7}.

A seguir segue o gráfico de treinamento realizado com essas variáveis, agora com o número de episódios aumentado para 50000.

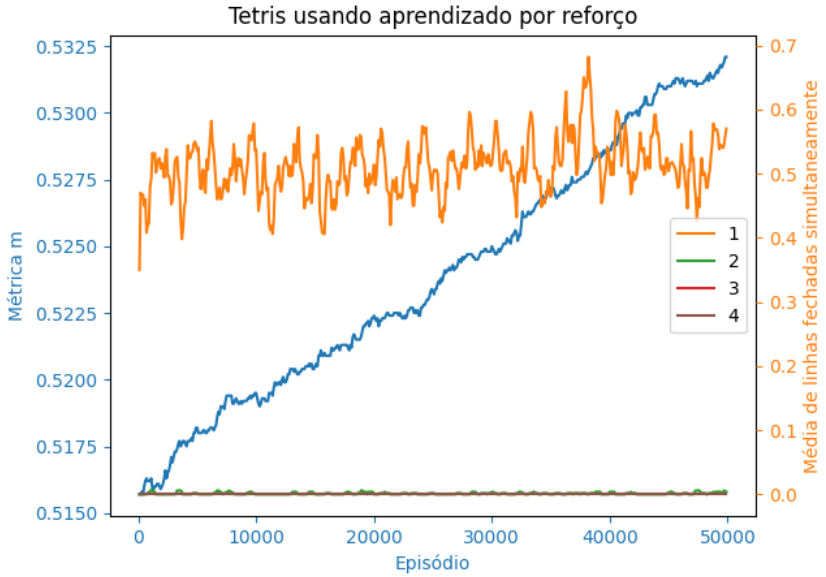


Figura 5.11: Curva de aprendizado com fração dos segmentos = 0.1, épocas por lote = 10 e taxa de aprendizado = 10^{-7}.

Ou, ainda, mais diretamente, conseguiu-se os seguintes valores médios de 500 jogos com a rede neural após esse treinamento:

Score	Métrica *m*	Linhas simultâneas			
		1	2	3	4
484,8	0,5321	0,4969	0,0	0,0	0,0

Tabela 5.9: Tabela de resultados finais

Apesar de utilizar o valor para as variáveis que foram considerados ótimos nas etapas anteriores, e ainda ter sido treinado pelo dobro de episódios, esses resultados não se destacam em nada em relação aos outros já documentados aqui. O *score* médio é, na verdade, um dos menores encontrados, assim como o número médio de linhas únicas completas. A métrica *m* ainda é sólida, embora não a melhor, mas isso de pouco importa considerando que essa rede havia de ser a final, e essa métrica não passa de uma auxiliar para tentar ter alguma segurança no crescimento futuro.

Provavelmente isso ocorreu porque ambas as variáveis alteradas do padrão em relação à etapa anterior, tanto a fração de segmentos quanto épocas por lote, tem relação direta com a velocidade do treinamento, e foram diminuídas sensivelmente. Isso acabou atrasando em demasia o aprendizado e, mesmo com o dobro dos episódios, não foi possível compensar isso. É possível notar esse comportamento analisando a curva de crescimento das métricas na figura 5.11, bastante parecido com a figura 5.10 que é também bastante notória pelo crescimento lento. Ainda assim, assim como essa, a curva aqui analisada possui suas virtudes: seu crescimento para ambas as métricas parece ser contínuo, embora vagaroso. Talvez, com um treinamento mais longo, seu pico de desempenho se mostre bom, e, possivelmente, dado que não demonstra qualquer sinal de queda ou mesmo de estabilização, ainda melhor que os demais.

Capítulo 6

Conclusão

Em comparação com os artigos relacionados ao Tetris em específico analisados na seção de revisão bibliográfica desta monografia, que chegaram a figurar no nível de centenas de milhões de linhas em um único jogo, os resultados obtidos foram extremamente aquém das expectativas, com uma média de menos de 1 linha por jogo.

O objetivo, porém, deste trabalho nunca foi tentar competir com heurísticas mais complexas, mas sim criar um agente que possuísse a capacidade de explorar o espaço de busca de políticas com o mínimo de auxílio possível, tentando encontrar soluções também viáveis mas sem utilizar o mesmo viés tradicional utilizado por humanos.

Visto desse modo, o agente desenvolvido se mostrou capaz de aprender o suficiente inicialmente para atingir resultados, ainda que baixos, de forma consistente, e, ainda, continuamente melhorar ao decorrer do treinamento (dado, claro, uma boa escolha de variáveis de controle).

6.1 Trabalhos futuros

Há, ainda, de ser feita uma análise mais exaustiva, tanto de combinações entre diferentes valores das variáveis exploradas aqui, em especial das taxas de aprendizado, quanto com o aumento de episódios para cada teste. Isso, muito provavelmente renderia novaas maneiras novas de melhorar esse algoritmo, visto que não foi possível atingir um ponto de estabilização em muitas das curvas.

Outra coisa a ser explorada é a topologia da rede neural utilizada, que é relativamente simples, mas poderia ter tanto os filtros mudados de tamanho ou número, quanto as próprias camadas, convolucionais ou não, aumentadas, especialmente em quantidade, a fim de criar uma rede mais profunda.

Por fim, outra abordagem interessante seria expandir os segmentos utilizados para também utilizar os piores, desse modo criando um reforço negativo no agente. Assim, aumentaria-se a diversidade de dados e a rede seria mais acurada em suas previsões, facilitando diferenciar as jogadas boas das ruins.

Referências Bibliográficas

Algorta, S. e Şimşek, Ö. (2019). The game of tetris in machine learning. *arXiv preprint arXiv:1905.01652*.

Arulkumaran, K., Deisenroth, M. P., Brundage, M. e Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38.

Böhm, N., Kókai, G. e Mandl, S. (2005). An evolutionary approach to tetris. Em *The Sixth Metaheuristics International Conference (MIC2005)*, página 5. Citeseer.

Jang, B., Kim, M., Harerimana, G. e Kim, J. W. (2019). Q-learning algorithms: A comprehensive classification and applications. *IEEE access*, 7:133653–133667.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. e Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8:293–321.

Murtagh, F. (1991). Multilayer perceptrons for classification and regression. *Neurocomputing*, 2(5-6):183–197.

Russell, S. J., Norvig, P. e Davis, E. (2010). 18 Learning from Example. Em *Artificial intelligence: a modern approach*, Prentice Hall series in artificial intelligence, páginas 693–757. Prentice Hall, Upper Saddle River, 3rd ed edition.

Stevens, M. e Pradhan, S. (2016). Playing tetris with deep reinforcement learning. *Convolutional Neural Networks for Visual Recognition CS23, Stanford Univ., Stanford, CA, USA, Tech. Rep*.