# Algoritmo genético para evolução de rede neural totalmente conectada

Este trabalho apresenta um estudo sobre a utilização de algoritmos genéticos na otimização da arquitetura de redes neurais para um problema específico. A experimentação foi realizada através de um conjunto fixo de parâmetros e de uma população limitada de indivíduos. Os resultados indicam que o algoritmo genético foi eficaz na busca por soluções eficientes, convergindo rapidamente para redes neurais de poucas camadas escondidas, apesar da definição de um tamanho máximo de indivíduos maior. Entretanto, a pressão seletiva, o tamanho da população e a limitação do espaço de solução se mostraram fatores determinantes para o desempenho do algoritmo, conduzindo a uma convergência precoce. Embora o estudo tenha demonstrado a viabilidade da aplicação de algoritmos genéticos na otimização de redes neurais, a complexidade do problema e o espaço de solução devem ser considerados com cuidado para evitar a convergência prematura e garantir uma exploração eficiente do espaço de busca.

**ACM Reference Format:**
. 2023. Algoritmo genético para evolução de rede neural totalmente conectada. 1, 1 (July 2023), 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUÇÃO

Nos últimos anos, o uso de técnicas de inteligência artificial tem crescido em diversas áreas, incluindo os jogos eletrônicos. Uma dessas técnicas é o aprendizado por reforço, que tem se mostrado particularmente eficaz para melhorar a jogabilidade e tornar os jogos mais desafiadores e interessantes. O aprendizado por reforço permite que um agente (como um personagem controlado pelo computador) aprenda a tomar decisões em um ambiente dinâmico, com base em recompensas e punições que são recebidas a cada ação tomada. Este é um problema desafiador devido à alta dimensionalidade do espaço de estados dos jogos.

Neste projeto, exploraremos como o aprendizado por reforço pode ser combinado com aprendizado profundo e algoritmos genéticos para gerar indivíduos generalistas e inteligentes dentro de jogos. O objetivo é aplicar técnicas de algoritmo genético para evoluir uma rede neural Deep Q-network totalmente conectada, que será aplicada ao jogo Snake.

O jogo Snake é um exemplo clássico de jogo eletrônico em que um personagem, representado por uma cobra, deve se mover em um espaço bidimensional, coletando alimentos e evitando colisões com o próprio corpo e as bordas do ambiente. O objetivo é fazer a cobra crescer ao comer os alimentos e obter a maior pontuação possível.

A implementação do algoritmo genético neste projeto foi baseada na técnica DENSER (Deep NeuroEvolution for Stock Trading) [Assunção et al. 2019]. Essa técnica utiliza operadores genéticos, como cruzamento, mutação e seleção, para evoluir arquiteturas de redes neurais totalmente conectadas em redes CNNs (Convolutional Neural Networks).

A combinação do aprendizado por reforço, aprendizado profundo e algoritmos genéticos oferece uma abordagem promissora para aprimorar a jogabilidade em jogos eletrônicos. A abordagem aqui implementada pode ser implementada nos mais diversos campos onde as redes neurais se encontram.

Author's address:


XXXX-XXXX/2023/7-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 2 TRABALHOS RELACIONADOS

O aprendizado profundo, utilizando redes neurais complexas e aprendizado por reforço, permite a obtenção de competências em jogos eletrônicos de maneira semelhante ao processo de aprendizagem humano. Isso é alcançado por meio de interações e experimentações, permitindo que a IA entenda e se adapte ao ambiente virtual de maneira autônoma, recebendo recompensas ou punições.

As arquiteturas mais avançadas de aprendizado profundo para jogos incluem o Deep Q-network (DQN) [Mnih et al. 2015], o mais renomado modelo de aprendizado profundo para jogos. Este modelo recebe entradas multidimensionais, sendo os pixels do estado atual do jogo e as possíveis ações, e aprende a estimar a recompensa que obterá se tomar uma determinada ação. Para ajudar a manter a estabilidade do treinamento, o DQN utiliza a técnica de "Experience Replay".

Com base no DQN, o modelo Double Deep Q-Network (DDQN) [van Hasselt et al. 2016] foi implementado. Este é uma extensão do DQN desenvolvido para resolver o problema de superestimação de valores Q que pode ocorrer no DQN. Para resolver este problema, o DDQN usa duas redes neurais em vez de uma.

Outro modelo relevante é o "Pop-Art", uma técnica popular no campo de aprendizado profundo por reforço (DRL) [van Hasselt et al. 2016]. Este modelo foi criado para resolver o problema de recompensas esparsas em jogos, utilizando uma média móvel das recompensas anteriores para normalizá-las.

O Trust Region Policy Optimization (TRPO) é um algoritmo de otimização de políticas de aprendizado por reforço que usa uma rede neural para representar a política do agente [Schulman et al. 2015]. O TRPO garante que as atualizações da política não sejam muito grandes, para que o desempenho do agente não se deteriore drasticamente durante o treinamento. Uma das principais vantagens do TRPO é que ele é capaz de lidar com políticas de alta dimensionalidade, adequado para jogos e ambientes complexos.

A Representação Estruturada de Rede Evolucionária Profunda (DENSER) [Assunção et al. 2019] é uma representação de modelo de IA que combina os princípios dos Algoritmos Genéticos (GAs) e da Evolução Gramatical Estruturada Dinâmica (DSGE), uma variante da Programação Genética (GP). Ela permite codificar a estrutura das Redes Neurais Artificiais (ANN) e os hiperparâmetros nas soluções candidatas. No nível GA, um exemplo de estrutura permitida para evoluir Redes Neurais Convolucionais (CNNs) pode ser: [(features, 1, 10), (classification, 1, 2), (softmax, 1, 1), (learning, 1, 1)], onde cada tupla indica os símbolos de início válidos e o número mínimo e máximo de vezes que eles podem ser usados. Já no nível DSGE, os parâmetros e seus valores ou intervalos permitidos são codificados

na gramática definida pelo usuário. Por exemplo, para as camadas de pooling, ajustamos o tamanho do kernel, o stride e o tipo de padding.

## 3 METODOLOGIA

Existem três principais componentes neste trabalho, o jogo, o modelo de rede neural baseado em Deep Q-net e o modelo de programação genética.

### 3.1 Snake Game

Para a implementação do projeto foi utilizado o jogo "Snake" devido a sua simplicidade e baixo espaço de ações possíveis. No jogo, o jogador controla uma criatura, a serpente, que rasteja pela tela, coletando comida. Cada vez que a serpente come a comida, ela cresce em comprimento. O jogo termina quando a serpente colide com a borda da tela ou com o próprio corpo. O objetivo do jogo é coletar a maior quantidade possível de comida, fazendo a serpente crescer o máximo possível, sem colidir.

O jogo foi implementado com 3 possíveis a cada iteração do jogo:

- **Continuar em Linha Reta:** A serpente mantém sua direção atual e avança uma unidade.
- **Virar à Direita: A serpente** muda sua direção 90 graus à direita e avança uma unidade.
- **Virar à Esquerda:** A serpente muda sua direção 90 graus à esquerda e avança uma unidade.

Estas ações são codificadas como um vetor de três elementos, onde cada elemento representa a escolha da ação correspondente.

Em aplicações de IA, como aprendizado por reforço, o estado do jogo é uma parte crítica da informação que a IA usa para decidir suas ações. O estado de jogo é o que representa o jogo num determinado momento. A IA aprende a associar estados de jogo com ações que maximizam algum tipo de recompensa ao longo do tempo.

Neste jogo o estado foi definido por onze elementos com significados distintos:

- **Posição da Cabeça da Serpente:** A posição atual da cabeça da serpente na tela.
- **Direção da Serpente:** A direção atual em que a serpente está se movendo.
- **Posição da Comida:** A posição atual da comida na tela.
- **Tamanho da Serpente:** O número atual de segmentos da serpente, ou seu "tamanho".
- **Distância da Cabeça da Serpente até a Comida:** A distância em linha reta da cabeça da serpente até a comida.
- **Direção da Comida em Relação à Cabeça da Serpente:** A direção da comida vista a partir da cabeça da serpente.
- **Existência de Obstáculo Diretamente à Frente:** Se a serpente colidirá com algo se continuar em linha reta.
- **Existência de Obstáculo à Direita Direta:** Se a serpente colidirá com algo se virar à direita.
- **Existência de Obstáculo à Esquerda Direta:** Se a serpente colidirá com algo se virar à esquerda.
- **Tempo Desde a Última Comida Consumida:** O número de passos desde a última vez que a serpente comeu.
- **Pontuação Atual:** Esta é a pontuação que o jogador ou a IA ganhou até o momento atual no jogo (quantidade de vezes que a serpente conseguiu comer).

### 3.2 Deep Q-Network

Em uma Q-Network, a rede neural é treinada para estimar a função Q, que mapeia pares de estados e ações para seus respectivos valores esperados de recompensa futura. A função Q representa a utilidade esperada de uma ação específica em um determinado estado. A rede neural recebe o estado atual como entrada e produz uma saída que representa os valores Q para todas as ações possíveis.

Durante o treinamento, a Q-Network é atualizada iterativamente utilizando a técnica de aprendizado por reforço conhecida como Q-learning. O objetivo do treinamento é fazer com que a rede aprenda a estimar os valores Q corretamente, de modo que a ação escolhida em um determinado estado maximize a recompensa total esperada ao longo do tempo.

Na implementação de uma DQN para o jogo Snake, a rede neural recebe o estado atual do jogo como entrada e retorna uma ação. Esta ação pode ser aleatória ou não, dependendo da etapa do treinamento da rede. Inicialmente, espera-se que a probabilidade de uma ação aleatória ser tomada seja mais alta, aumentando assim a exploração. Conforme o modelo acumula mais relações estado/ação/recompensa em sua memória, a ação a ser tomada tende a deixar de ser aleatória.

### 3.3 Programação Genética

Para a implementação do algoritmo foram definidos os possíveis tipos de camada e suas possíveis dimensões:

- Layers:
  - Linear/ReLU
  - Linear/Tahn
  - Linear/Sigmoid
- Dimensões: [32,64,128,256]

Os operadores genéticos são componentes fundamentais dos algoritmos genéticos, que são técnicas de otimização inspiradas no processo de evolução biológica. Esses operadores são responsáveis pela manipulação e combinação dos indivíduos na população, simulando processos de reprodução e recombinação genética que ocorrem na natureza.

Os operadores genéticos foram implementados para a evolução das redes neurais. O operador de seleção implementado foi a seleção por torneio, que seleciona aleatoriamente um número K de indivíduos, e, desses K indivíduos, o melhor é selecionado. Este método é uma forma eficaz de controlar a pressão seletiva do algoritmo genético e não necessita de estatísticas de todos os indivíduos existentes.

O operador de cruzamento foi implementado com base na estratégia de cruzamento por ponto. Uma camada da rede neural de cada um dos dois indivíduos é selecionada aleatoriamente. Se as funções de ativação das camadas escolhidas forem diferentes, as dimensões dessas camadas são alteradas para corresponderem à outra camada e, em seguida, as camadas são trocadas entre os indivíduos.

Como pode ser observado a operação de cruzamento altera apenas a função de ativação, isso evitar que grandes mudanças ocorram e

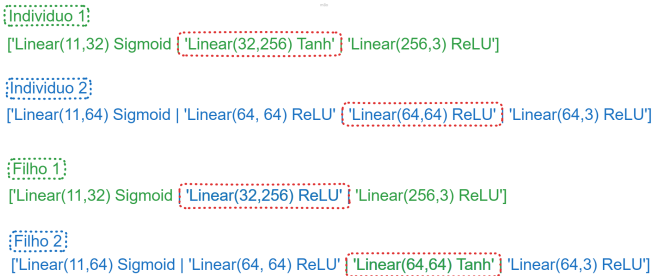


Fig. 1. Representação da etapa de cruzamento por ponto.

que uma camada incompatível seja adicionada em alguma das novas redes geradas.

O operador de mutação foi dividido em quatro técnicas com igual probabilidade de ocorrer:

**Deleção:** uma camada aleatória da rede que não seja a camada de entrada e a de saída, portanto essa mutação só pode ser utilizada se existir ao menos uma camada escondida, após a deleção também é necessário adaptar as dimensões das camadas que continuaram na rede.

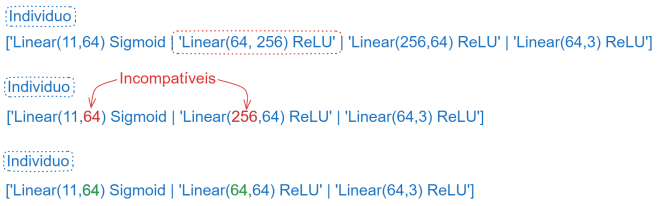


Fig. 2. Representação do "delete layer" existente no operador de cruzamento.

**Cópia de camada:** uma camada da rede, que não seja a de entrada e nem a de saída, é aleatoriamente selecionada e essa camada será copiada e adicionada a na próxima posição em relação a camada original. Portanto para essa operação ser feita é necessário pelo menos uma cama escondida para ser replicada. Por fim, essa nova camada terá suas dimensões alteradas para compatibilidade da rede.


Fig. 3. Representação da estratégia "clone layer" existente no operador de cruzamento.

**Adicionar camada aleatória:** caso o indivíduo apresente apenas camada de entrada e camada de saída, ou seja, não possua camadas escondidas, então é gerado uma camada de dimensões e função de ativação aleatórias. Essa camada possui suas dimensões adaptadas para ser encaixada no indivíduo onde ela é incluída como camada escondida.

**Substituir camada aleatória:** Caso o indivíduo possua pelo menos uma camada escondida, então esta estratégia pode ser acionada.


Fig. 4. Representação da estratégia "add random layer" existente no operador de cruzamento.

Ela consiste em selecionar aleatoriamente uma camada escondida do indivíduo e registrar suas dimensões de entrada e saída. Em seguida, calcula-se a diferença entre o tamanho do indivíduo e o tamanho máximo estipulado. A partir disso, um número aleatório entre 1 e essa diferença é sorteado, esse número será o tamanho da lista de camadas aleatórias a serem geradas. Finalmente, essa lista de camadas aleatórias terá, na primeira camada, a dimensão de entrada compatível com a camada original e, na última camada, a dimensão de saída também compatível com a camada original.

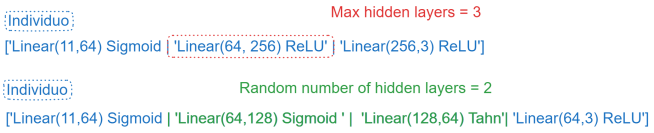


Fig. 5. Representação da estratégia "replace with n layers" existente no operador de cruzamento.

Como foi descrito, algumas das estratégias possuem restrição em relação ao tamanho do indivíduo para ocorrerem, porém caso todas as restrições sejam respeitadas elas possuem iguais chances de ocorrerem.

## 4 EXPERIMENTAÇÃO

Os parâmetros para a experimentação do modelo proposto foram feitos com os seguintes valores:

| Parâmetro | Valor |
|---|---|
| # Gerações | 50 |
| Tamanho da População | 10 |
| Elitismo | True |
| Tamanho máximo do indivíduo | 6 |
| Taxa de Cruzamento | 0.8 |
| Taxa de mutação | 0.2 |
| Tamanho do torneio | 2 |
| Etapas de treinamento | 150 |

Table 1. Tabela de parâmetros fixos para a experimentação.

Devido à quantidade limitada de possibilidades de arquitetura de redes, foi evitado usar uma população muito grande de gerações para não esgotar todas as opções e encontrar a melhor solução por um método praticamente aleatório. Este fato também torna possível a realização do experimento em tempo adequado para a disciplina, sem a necessidade de um grande poder computacional.

A escolha do tamanho máximo do indivíduo foi escolhida visando aumentar a quantidade de possíveis arquiteturas no espaço de soluções, porém limitando a quantidade máxima de camadas escondidas para evitar redes muito complexas. Foi perceptível que redes neurais com um ou até nenhuma camada escondida performam melhor devido a baixa dimensionalidade de entrada (o estado possui apenas 11 variáveis) e a baixa dimensionalidade de saída (existem apenas 3 ações possíveis). Portanto, o tamanho máximo do indivíduo não foi definido como 2 para simular um problema onde não sabemos qual a quantidade de camadas escondidas ideal e também não foi definido como maior que 2 para evitar que o espaço de soluções possíveis fique grande demais.

A escolha da quantidade de etapas de treinamento veio de conhecimento prévio do problema, onde foi perceptível que com mais de 150 etapas de treinamento as redes começavam a se sobre ajustar e o indivíduo apresentava comportamentos ruins, sem generalização.

A taxa de cruzamento foi escolhida se baseando em outras técnicas que utilizaram mecanismos parecidos. E por fim, a escolha da taxa de mutação foi mais alta que a geralmente utilizada em técnicas parecidas para aumentar a exploração já que temos um espaço de solução limitado e com poucos indivíduos/gerações. Isso é uma forma de tentar fazer a exploração em diversas partes do espaço de busca de forma mais rápida.

Embora a evolução dos parâmetros de uma rede neural possam ser também evoluídos através de estratégias evolutivas eles não serão abordadas neste projeto. Portanto os parâmetros como taxa de aprendizagem e decaimento do épsilon serão fixos.

As experimentações feitas foram coletadas a partir da execução de 5 runs com sementes aleatórias diferentes.

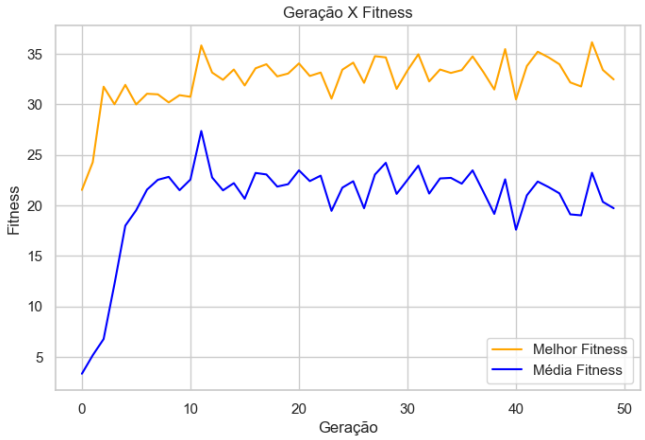


Fig. 6. Resultados obtidos a partir da média de cinco runs. Em amarelo é representada a fitness média dos melhores indivíduos de cada geração em 5 runs e em azul a fitness média de todos os indivíduos das 5 runs.

É possível notar que embora o uso do elitismo definido como "True", a fitness média dos melhores indivíduos oscila. Isso ocorre pois o elitismo foi implementado de forma a reavaliar o melhor indivíduo sempre para que o exploration seja favorecido. Além disso, devido a baixa complexidade do problema e a quantidade limitada de possíveis indivíduos diferentes, temos que o algoritmo se estabilizou por volta da geração 10. Esse comportamento não deve ocorrer em soluções que admitam camadas diferentes e com muitas dimensões.

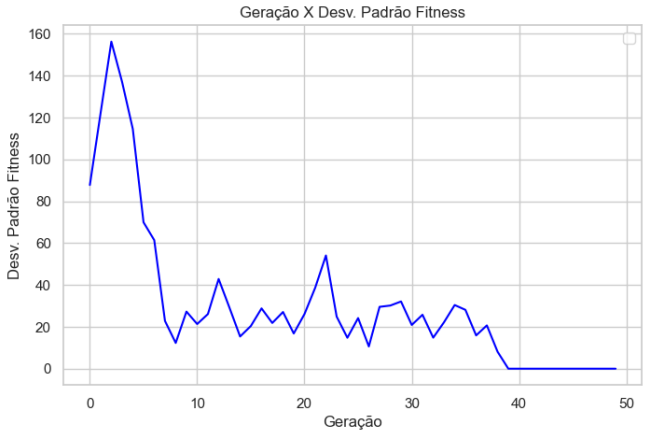


Fig. 7. Desvio padrão médio de 5 runs por geração.

Ao analisar o desvio padrão obtido na experimentação, é possível perceber que os indivíduos gerados inicialmente produzem resultados muito diferentes. Porém, pelo mesmo fato evidenciado anteriormente, a conversão do algoritmo para indivíduos parecidos ocorre de forma muito rápida, por volta da décima geração. A partir da geração 40 os indivíduos são praticamente os mesmos, ocasionando em resultados iguais e um desvio padrão próximo ao 0. Isso é um comportamento esperado para um algoritmo onde ocorreu convergência, porém essa convergência ocorreu de forma prematura devido a complexidade e quantidade total de possíveis indivíduos.

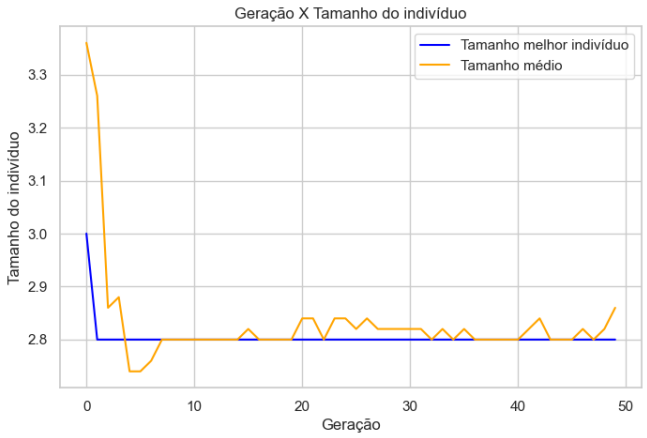


Fig. 8. Resultados obtidos a partir da média de cinco runs. Em amarelo temos o tamanho médio de todos os indivíduos em cada geração (quantidade total de camadas). Em azul temos o tamanho médio do melhor indivíduo de cada geração.

O principal objetivo deste trabalho é encontrar topologias de redes neurais que são eficientes para o problema proposto. Ao analisar

o tamanho médio dos indivíduos é possível perceber que, embora o tamanho máximo definido de 6, o algoritmo rapidamente converge para indivíduos com poucas camadas. Isso ocorre porque indivíduos com poucas camadas escondidas performaram melhor. O principal motivo para que o tamanho médio dos indivíduos reduza quase instantaneamente é porque a população de 10 indivíduos e uma seleção por torneio de k=2 gera muita pressão seletiva, desta forma os piores indivíduos (com muitas camadas escondidas) são removidos logo nas primeiras gerações.

## 5 CONCLUSÃO

Os resultados experimentais obtidos neste estudo indicam que, para o problema abordado, soluções com redes neurais de poucas camadas escondidas apresentaram melhor desempenho. Apesar da definição do tamanho máximo do indivíduo ser de 6, o algoritmo rapidamente convergiu para soluções de menor complexidade, demonstrando que, neste contexto, redes mais simples foram mais eficazes.

A convergência rápida observada pode ser atribuída à combinação de uma população relativamente pequena de indivíduos e a aplicação de um método de seleção de torneio, que tende a exercer alta pressão seletiva, especialmente quando k=2. Tal situação acelerou a exclusão dos indivíduos menos aptos, levando a uma rápida estabilização do algoritmo.

A oscilação na fitness média dos melhores indivíduos, apesar do uso de elitismo, pode ser explicada pela reavaliação constante do melhor indivíduo, favorecendo a exploração em detrimento da explotação.

Observou-se também que o desvio padrão dos indivíduos diminuiu significativamente ao longo das gerações, indicando uma convergência para um conjunto similar de soluções. Porém, essa convergência ocorreu de forma precoce, provavelmente devido à simplicidade do problema e à limitação no número de soluções possíveis.

Portanto, este estudo demonstrou que o uso de algoritmos genéticos é uma técnica viável para encontrar topologias eficientes de redes neurais. Porém, a complexidade do problema, a pressão seletiva, o tamanho da população e a limitação do espaço de solução são fatores que devem ser cuidadosamente considerados para garantir uma busca eficiente e evitar a convergência prematura.

## REFERENCES

Filipe Assunção, Nuno Lourenço, Penousal Machado, and Bernardete Ribeiro. 2019. DENSER: deep evolutionary network structured representation. *Genetic Programming and Evolvable Machines* 20 (2019), 5–35.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *International conference on machine learning*. PMLR, 1889–1897.

Hado P van Hasselt, Arthur Guez, Matteo Hessel, Volodymyr Mnih, and David Silver. 2016. Learning values across many orders of magnitude. *Advances in neural information processing systems* 29 (2016).