UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
PROJETO DE GRADUAÇÃO

VICTOR PEDROTE CESCONETTO

UMA ABORDAGEM AO PROBLEMA DE CARREGAMENTO EM CONTÊINER VIA MODELOS DE APRENDIZADO POR REFORÇO PROFUNDO

Vitória-ES

Outubro/2021

VICTOR PEDROTE CESCONETTO

UMA ABORDAGEM AO PROBLEMA DE CARREGAMENTO EM CONTÊINER VIA MODELOS DE APRENDIZADO POR REFORÇO PROFUNDO

Parte manuscrita do Projeto de Graduação do aluno VICTOR PEDROTE CESCONETTO, apresentado ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Engenheiro Eletricista.

Vitória-ES

Outubro/2021

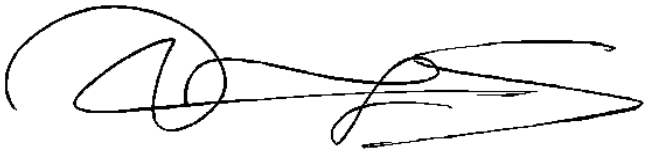
VICTOR PEDROTE CESCONETTO

UMA ABORDAGEM AO PROBLEMA DE CARREGAMENTO EM CONTÊINER VIA MODELOS DE APRENDIZADO POR REFORÇO PROFUNDO

Parte manuscrita do Projeto de Graduação do aluno VICTOR PEDROTE CESCONETTO, apresentado ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Engenheiro Eletricista.

Aprovado em 7 de Outubro de 2021.

COMISSÃO EXAMINADORA:



Prof. Dr. Jorge Leonid Aching Samatelo
Universidade Federal do Espírito Santo
Orientador

Prof. Dr. Patrick Marques Ciarelli
Universidade Federal do Espírito Santo
Examinador

Dr. Clebeson Canuto dos Santos
Examinador

Vitória-ES

Outubro/2021

A meus pais, familiares e grandes amigos, razões da minha dedicação.

AGRADECIMENTOS

Gostaria de agradecer a todas essas pessoas especiais que apareceram e estão na minha vida e que me permitem seguir sempre com entusiasmo no meu dia a dia. A todos vocês, meu eterno reconhecimento.

Aos meus pais, Edival e Geni, pelo apoio e compreensão constantes ao longo de todos esses anos.

Ao meu irmão Raphael por todos momentos em que me encorajou a seguir em frente.

A todos os meus grandes amigos que sempre acreditaram no meu potencial.

A meu orientador Jorge Aching pela grande paciência comigo durante todo o projeto. E principalmente por toda a dedicação e perseverança ao me orientar neste desafio não habitual.

À banca examinadora pela aceitação do convite e pelo tempo investido para leitura e avaliação desse trabalho.

Agradeço à Universidade Federal do Espirito Santo pela minha formação.

RESUMO

O problema de carregamento em contêiner é um clássico desafio da área de logística que estuda formas de organizar, com mais eficiência, cargas em um determinado espaço. Ele é encontrado em sistemas de transporte e até de estoque. A importância desse tipo de estudo é cada vez maior com a busca por processos de menores custos e tempo por empresas inseridas em um meio de alta concorrência.

Assim, alguma ferramenta para a otimização da alocação das cargas se torna indispensável. Porém, as soluções automatizadas existentes não são tão eficazes em relação a qualidade e/ou tempo. Essa situação, aliada ao grande avanço em metodologias voltadas para aprendizado de máquina, cria a oportunidade para uma nova concepção.

Este projeto faz uso da abordagem de aprendizado por reforço para criar um agente que atue sobre um conjunto de cargas a ser organizado. Para a estrutura de decisões desse agente faz se uso de uma rede neural profunda tanto para a escolha da carga a ser alocada como para a sua alocação no espaço destinado.

Para auxiliar no processo de treinamento desse modelo foi criado um ambiente virtual, tanto para a criação de conjuntos aleatórios de cargas ao longo do extenso processo de treinamento, como para a atuação sobre estes conjuntos, tendo sempre um *feedback* a cada ação do agente. Um destaque adicional foi dado à arquitetura da rede e a rotina de treinamento baseadas em melhorias atuais apresentadas pela comunidade de aprendizado por reforço profundo.

Foi utilizado o modelo heurístico *Down Bottom Left with Fill* (DBLF) como comparação para os resultados manifestados pelo presente projeto apresentado. Estes chegaram a ser até 28% melhores que o DBLF considerando como métrica a média das recompensas totais de 1.0000 episódios.

Palavras-chave: *Aprendizado de Máquina*; *Aprendizado Profundo*; *Aprendizado por reforço*; *Problema de carregamento em contêiner*.

ABSTRACT

The container loading problem is a classic challenge in the logistics area, which study ways to efficiently organize loads in a given space. This problem is found in transport and even on stock systems. The importance of this type of study is increasing with the search for processes with lower cost and time by companies inserted in a highly competitive environment.

Thus, some tool for optimizing the allocation of loads becomes indispensable. However, existing automated solutions are not as effective in terms of quality and/or time. This situation, together with the great advance in methodologies aimed at machine learning, creates the opportunity for a new conception.

This project uses a reinforcement learning approach to create an agent that acts on a set of loads to be organized. For the decision structure of this agent, a deep neural network is used both for choosing the load to be allocated and for its allocation in the destined space.

To assist in the training process of this model, a virtual environment was created, both for the creation of random sets of loads throughout the extensive training process, and for acting on these sets, always having a feedback for each action of the agent. A further highlight was given to the network architecture and the training routine based on improvements just introduced by the deep reinforcement learning community.

The Down Bottom Left with Fill (DBLF) heuristic model was used as a comparison for the results manifested by the present project presented. These were up to 28% better than DBLF considering as matric an avaerage of the total rewards of 10000 episodes.

Keywords: *Machine Learning*; *Deep-learning*; *Reinforcement Learning*; *Container Loading Problem.*

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

DBLF	*Deep Bottom Left with Fill*
DL	*Deep Learning*
GPU	*Graphics Processing Unit*
IA	Inteligência Artificial
ML	*Machine Learning*
UFES	Universidade Federal do Espírito Santo

SUMÁRIO

1 INTRODUÇÃO

1.1 Apresentação

O processo de planejamento e execução eficiente de transporte e armazenamentos de mercadorias é algo presente na história da humanidade. Com o avanço da globalização, a necessidade para a redução de custos e maior velocidade no processo fica cada vez maior, de forma a se ter uma vantagem em relação a concorrência. Assim, esse processo do campo da logística começou a se desenvolver até chegar ao ponto atual.

Nesse meio nasceu a necessidade de métodos que tenham como objetivo a otimização de diversos fatores, como a máxima ocupação de volume, máxima quantidade de carga, maior valor em carregamento, máxima quantidade de peso e até melhor distribuição, buscando posições de centro de gravidade total específicos, sendo estes fatores abordados individualmente ou coletivamente.

O problema de carregamento de contêiner único aparece neste contexto, no qual se aborda essa problemática ou desafio de forma a ser organizada uma quantidade específica de cargas, buscando a otimização de um ou mais fatores para um único contêiner por vez. Nessa abordagem, são consideradas cargas retangulares que devem ser alocadas dentro de uma caixa retangular maior.

Neste tipo de problema ainda podem existir algumas variações. Dependendo da variação das cargas a serem utilizadas, é possível ter: um caso completamente homogêneo, onde as caixas possuem tamanhos e orientações idênticas, até um caso fortemente heterogêneo, onde as caixas têm diferentes tamanhos, além de poder apresentar uma rotação completa de 90^o em qualquer um dos eixos (GONÇALVES; RESENDE, 2012).

Existem diversos tipos de algoritmos para se trabalhar esta problemática. Modelos completamente heurísticos, como o apresentado por George e Robinson (1980), possuem custo computacional baixo e resultados bons quando são comparados a apenas colocar as caixas aleatoriamente dentro do contêiner. Já modelos meta-heurísticos, como busca tabu, por Bortfeldt, Gehring e Mack (2003) ou algoritmo genético, por Zheng, Chien e Gen (2015), conseguem atingir resultados melhores, mantendo o custo computacional tolerável, desde que não sejam exigidos objetivos de otimização de alta complexidade.

A principal característica dos métodos meta-heurísticos de otimização é o uso de uma componente probabilística, a qual, utilizando uma aleatoriedade direcionada pela heurística codificada na função objetivo, torna possível fazer uma busca pelo ótimo em diversas

regiões do espaço de busca (VENDRAMINI, 2007).

Devido a essa varredura do espaço em busca do ótimo global, mesmo que o espaço total a ser percorrido acabe sendo reduzido devido ao direcionamento do modelo, um certo tempo será consumido no processo. Conforme aumenta a complexidade do modelo a ser otimizado, maior vai ser o espaço a ser percorrido.

Com a diminuição do tamanho das cargas a serem colocadas no contêiner e/ou o aumento de objetivos de otimização, chegará um ponto em que o espaço a ser percorrido, devido as grandes possibilidades, fará com que o custo computacional seja alto. Isso pode acabar consumindo um tempo maior do que o disponível para a organização das cargas ou gerando um resultado não tão bom quanto se esperaria de uma otimização.

Já para métodos que utilizam o aprendizado de máquina, a execução de um algoritmo para a organização das caixas seria quase que instantâneo, uma vez que, na etapa de inferência, não se teria uma variação dos parâmetros dos modelos em busca de um ótimo global e sim uma resposta direta ao problema proposto. Isso viabiliza a aplicação de IA para situações que necessitam de ações em tempo real, bem comuns na área da robótica. Entretanto, é importante lembrar que, existe uma razoável dificuldade relacionada ao projeto e treino desses modelos.

A presença de algoritmos desse tipo baseados em aprendizado de máquina também é encontrada para resoluções de problemas similares, como proposto por Tanaka et al. (2020). No entanto, mesmo com o uso e crescimento do aprendizado de máquina nos dias de hoje em diversas áreas, é notável a menor presença destes tipos de algoritmo para a resolução do problema de carregamento em contêiner.

Diante dessas considerações, torna-se de interesse o estudo sobre a viabilidade, vantagens e o desenvolvimento de uma solução em algoritmo que trabalhe o problema de carregamento em contêiner utilizando aprendizado de máquina.

Este projeto faz uso de uma abordagem de aprendizado por reforço chamada *Q-Learning* para criar um agente que atue sobre um conjunto de cargas a ser organizado. A estrutura de decisões desse agente, para permitir uma melhor atuação deste, faz uso de uma rede neural profunda, tanto para a escolha da carga a ser alocada como para a sua alocação no espaço destinado.

Devido a natureza do modelo, baseado em aprendizado por reforço, foi criado um ambiente virtual. Este é responsável tanto pela criação de conjuntos aleatórios de cargas ao longo

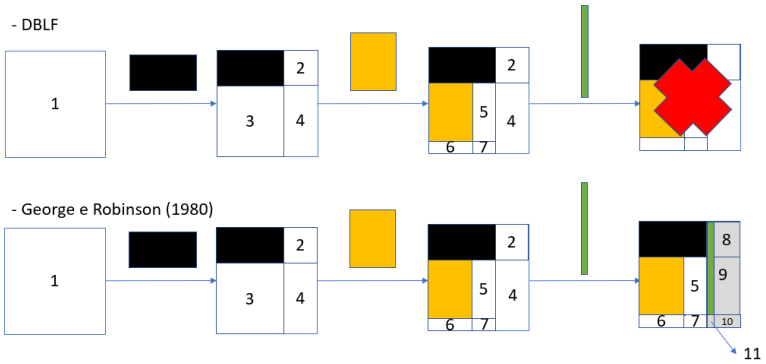
do extenso processo de treinamento, como para a atuação do agente sobre estes conjuntos, tendo sempre um *feedback* a cada ação. Estes *feedbacks* são compostos dos novos estados do ambiente que sofreu as ações do agente e as respectivas recompensas causadas por essas ações, baseados na lógica previamente definida. Um destaque adicional foi dado a arquitetura da rede e a rotina de treinamento baseado em melhorias recém apresentadas pela comunidade de aprendizado por reforço profundo, como: *Double Deep Q-Learning Network*, priorização de experiências, *Multi-step bootstrapping*, etc.

1.2 Trabalhos relacionados

Sobre a problemática de carregamento em contêiner, existem diversos trabalhos na literatura que abordam e propõem soluções. Os métodos propostos podem basicamente ser divididos em três grupos: heurísticos, meta-heurísticos e aprendizado de máquina. A seguir será mostrada uma proposta de cada uma desses grupos.

George e Robinson (1980) discutem o problema de achar maneiras de alocar remessas de caixas de diferentes tamanhos retangulares em um contêiner. Na solução apresentada as caixas são ordenadas pelas suas dimensões de maior para menor, e então são alocadas nessa ordem, onde inicialmente o único espaço definido é o total do contêiner e, então, a cada iteração, sempre que uma caixa é alocada é gerada uma disposição de espaços a serem preenchidos pelas próximas caixas na sequência. A ideia é parecida com o modelo heurístico DBLF, diferenciando-se em que, aqui, o modelo busca usar espaços próximos combinados para tentar alocar caixas maiores como visto na Figura 1.

Figura 1 – Representação 2D da diferença do método apresentado em George e Robinson (1980) com o método DBLF



Fonte: Produção do próprio autor.

Percebe-se que a cada caixa alocada são definidos novos espaços a serem preenchidos, em que no DBLF ao chegar na etapa da caixa verde o algoritmo é finalizado, pois esta

acomoda-se em nenhum dos espaços definidos. Já no modelo de George e Robinson (1980), na mesma etapa, o algoritmo usa os espaços 2 e 4 para alocar a caixa verde. Os espaços novos definidos a seguir são definidos como se a caixa alocada tivesse sido dividida em duas, e cada parte dela alocada no espaço livre separadamente.

Em Gonçalves e Resende (2012) é apresentada uma abordagem meta-heurística baseada na combinação de um algoritmo genético e o método DBLF levemente modificado. A forma de geração dos espaços e alocação das caixas é feita via o DBLF, porém, em vez de usar a prioridade de alocação decrescente em tamanho para caixas e crescente para os espaços, a ordenação é definida pela indexação das caixas representadas nos genes do algoritmo genético.

Em Tanaka et al. (2020) é proposta uma abordagem baseada em aprendizado por reforço. Aqui os autores buscam criar um sistema robótico que possa tanto pegar como colocar as caixas de forma organizada, de modo que, a penalização por colocar caixas maiores em cima de menores só ocorre quando essa situação causaria desequilíbrio, como visto na Figura 2, permitindo esse tipo de alocação desde que a base da caixa inferior tenha 50% do tamanho da base da caixa superior, possibilitando a geração de espaços vazios sem qualquer penalidade. O modelo neural contém camadas convolucionais e a entrada é formada por duas imagens que são vetorizadas e concatenadas, como mostrado na Figura 3. Cabe indicar que esta abordagem é a principal referência para a solução proposta neste trabalho de pesquisa.

Figura 2 – Representação de situação de desequilíbrio

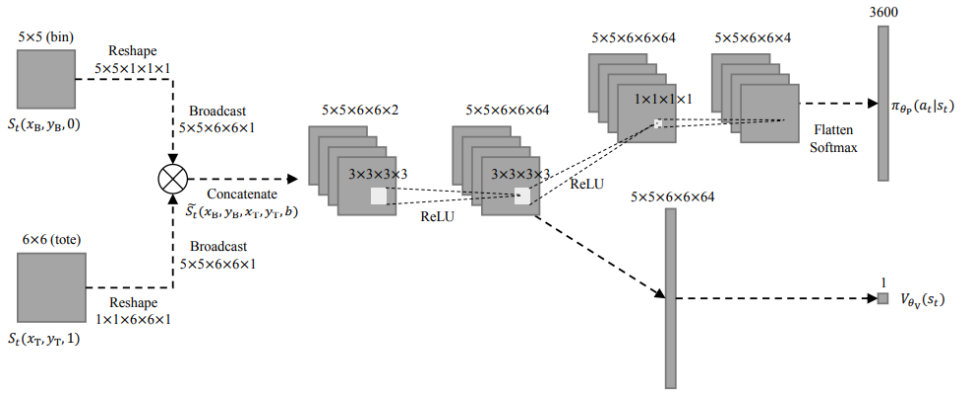
Fonte: Produção do próprio autor.

1.3 Objetivos

Objetivo Geral

Desenvolver um algoritmo utilizando como base o aprendizado por reforço profundo. Tal algoritmo deve ser capaz de organizar as cargas dentro de um contêiner de forma a cumprir com os objetivos de otimização determinados.

Figura 3 – Arquitetura da rede neural usada em (TANAKA et al., 2020).



Fonte: Tanaka et al. (2020).

Objetivos Específicos

- Desenvolver uma metodologia de manipulação de dados de forma a suprir a entrada do modelo neural com dados que representem as cargas a serem organizadas e o espaço disponível para cada instante de tempo;
- Implementar uma modelo neural treinado via uma metodologia de aprendizado por reforço que consiga um melhor desempenho que o algoritmo heurístico DBLF, utilizado como referência de comparação;
- Testar diferentes estruturas para o modelo neural do agente objetivando os melhores resultados.

1.4 Estrutura do Texto

O presente trabalho está estruturado da seguinte maneira:

- **Introdução**: este capítulo inicial teve como objetivo contextualizar o leitor acerca da problemática abordada do carregamento em contêiner, os tipos de soluções comumente usadas para a sua solução e uma breve explicação sobre como a proposta por aprendizado de máquina se encaixa no problema. Aqui também foi comentado sobre algumas propostas apresentadas por outros autores.
- **Referencial Teórico**: neste capítulo são apresentados e explicados conceitos chaves que serão abordados ao longo de todo o resto do manuscrito, como o funcionamento do método heurístico DBLF que será utilizado como comparativo nesse trabalho,

uma introdução básica sobre aprendizado por reforço e uma breve explicação do funcionamento de um modelo neural para tomada de decisões de um agente.

- **Proposta**: neste capítulo é apresentada a abordagem proposta para a solução do problema em estudo e detalhes acerca do trabalho feito. Aqui, mais especificamente, serão comentados sobre o ambiente virtual criado e como o agente se comunica com este, e particularidades da forma de treinamento da rede neural profunda.

- **Resultados**: neste capítulo são descritos a geração dos conjuntos dos dados de entrada e os resultados gerados pelos processos de treinamento, assim como os recursos de *Software* e *Hardware* utilizados para eles. Também são feitos alguns comentários comparativos com outros modelos a partir de algumas métricas de referência estabelecidas.

- **Conclusão**: no capítulo final deste trabalho, é apresentado um resumo sobre a problemática e solução proposta, salientado as contribuições do projeto. Ao fim, também são feitas algumas recomendações para trabalhos futuros.

2 REFERENCIAL TEÓRICO

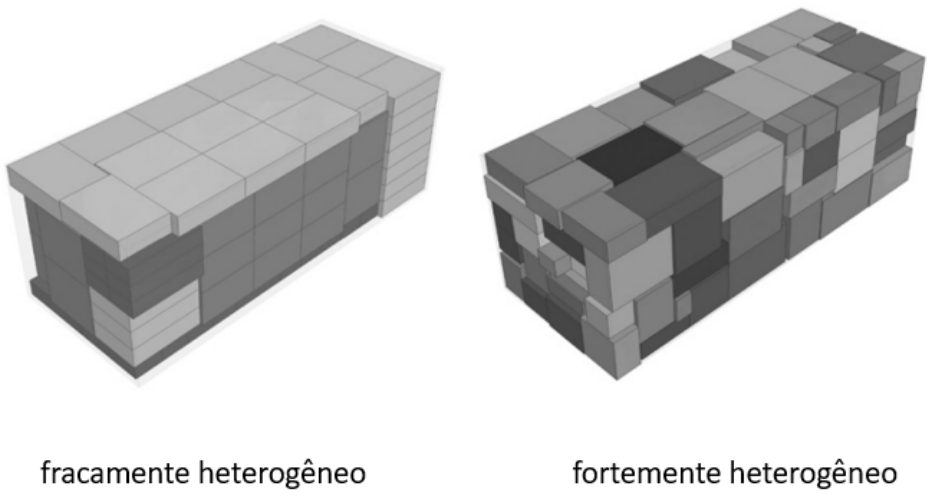
2.1 Problema de carregamento de contêiner

O problema de carregamento em contêiner tem diversas aplicações no meio comercial e industrial. São exemplos de aplicações a movimentação de objetos em *pallets*, a organização de materiais e/ou produtos em estoques e o carregamento de cargas para transporte.

O problema foi estudado inicialmente por Gilmore e Gomory (1961), e desde então foram elaborados artigos e algoritmos em abundância sobre a problemática. Ao longo do crescimento de materiais sobre o assunto foram sendo criadas diversas definições e variações do problema.

De acordo com Bortfeldt e Gehring (2001) o problema de carregamento em contêiner pode ser diferenciado por várias características. Primeiramente, este pode ser classificado como *Bin-Packing* quando a carga completa tem que ser armazenada, ou *Knapsack*, se for permitido que algumas cargas não sejam utilizadas no processo de organização. Outro tipo de diferenciação é referente ao tipo das cargas, em que estas podem ser homogêneas ou heterogêneas, sendo que no próprio enquadramento heterogêneo pode haver diversas variações referente ao nível de diferenciação das caixas, conforme mostrado na Figura 4.

Figura 4 – Exemplificação da variação de heterogeneidade



Fonte: Gonçalves e Resende (2012).
Nota: Adaptado pelo autor.

Dentro dessas variações já explicadas, ainda podem existir outras pequenas variações

como permitir ou não a rotação das cargas a serem alocadas, utilização de mais de um contêiner para a organização, exigências de equilíbrio para o peso, fragilidade das cargas necessitando uma posição superior destas e qualquer outro tipo de especificação que possa surgir na prática.

2.2 Algoritmo *Deepest Bottom Left with Fill*

O método heurístico utilizado como base para uma exploração mais eficiente do espaço, tanto pelo algoritmo genético de referência como para o modelo de aprendizagem de máquina em foco deste trabalho, foi o *Deepest Bottom Left with Fill* (DBLF).

De acordo com Karabulut e İnceoğlu (2004), *Bottom Left* (BL) e *Bottom Left with Fill* (BLF) são métodos utilizados para problemas de empacotamento 2D. No BL, introduzido por Jr (1966), cada item será movido o máximo que for possível para o fundo e então ao máximo para a esquerda. O problema é que acabam sendo gerados espaços vazios nesta esquemática. Para resolver este problema, Hopper (2000) criou o BLF. O diferencial deste algoritmo é dar prioridade aos menores espaços primeiro, preenchendo os espaços vazios na maioria das vezes.

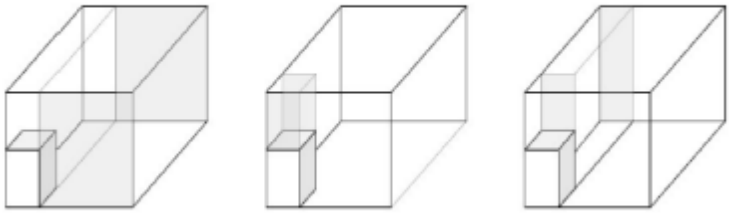
O DBLF que será utilizado neste trabalho é uma extensão do método BLF de forma a aplicar a mesma ideia só que para problemas em 3D. Agora, primeiramente a carga será movida para o ponto mais baixo, para então ser movida para o mais fundo e à esquerda possível, porém, é claro, dando prioridade para os espaços menores.

Devido à complexidade desse tipo de sistema, o algoritmo utilizado representará os espaços livres sempre em espaços retangulares. Nessa ideia, sempre que uma carga for colocada em um espaço livre, conforme visto na Figura 5, será gerado um espaço livre novo para cada dimensão da carga que for menor que o tamanho do espaço onde foi colocada. A prioridade de preenchimento será dado aos menores espaços livres até que não seja mais possível para qualquer carga disponível. Uma representação do algoritmo completo utilizado neste projeto pode ser visto no Algoritmo 1.

Importante indicar que a eficácia do DBLF é consequência da organização do sequenciamento do preenchimento a partir do tamanho das caixas. Assim, a eficácia desse método cai bastante para aplicações em tempo real, onde é necessário organizar partes do conjunto total por vez.

O DBLF por possuir uma lógica direta é de fácil implementação, e por não possuir

Figura 5 – Exemplo de novos espaços criados ao alocar uma caixa no espaço inicial



Fonte: Ramesh Umashankar (2021).

Algorithm 1 Pseudo código do método DBLF

```
while possui caixas a colocar do
    EspaçosOrganizados ← OrdemCrescente(EspaçosLivres)
    MaiorCaixa ← PegarMaior(Caixas)
    for EspaçosLivres do
        if PossívelEncaixar then
            NovosEspaços ← ColocarBaixoFundoEsquerda(Espaço)
            EspaçosAtualizados ← AdicionarEspaços(NovosEspaços)
            Break
        end if
        if UltimaIteração then
            NovoConjuntoCaixas ← TirarMaiorCaixa(ListaCaixas)
        end if
    end for
end while
```

hiperparâmetros não tem a necessidade de uma etapa de sintonia. Está ultima característica torna ao DBLF uma referencia válida para a comparação com outros métodos.

2.3 Aprendizado de máquina

De acordo com Morales (2020), inteligência artificial é uma ramificação dentro da ciência da computação que está diretamente ligada com o objetivo de criar programas computacionais capazes de demonstrar inteligência. Dentro desse grupo há a área de aprendizado de máquina, onde existem programas capazes de resolver problemas requerendo inteligência adquirida a partir de dados.

A área de aprendizado de máquina pode ser dividida em três principais ramificações:

- Aprendizado supervisionado: feito a partir de dados rotulados, onde um humano que decidirá quais os dados que serão coletados e como serão rotulados. É esperado que o modelo treinado consiga generalizar e gerar resultados corretamente a partir de dados de entrada;
- Aprendizado não supervisionado: feito a partir de dados não rotulados, no qual a partir de alguma estratégia de agrupamento definida por um ser humano terá como objetivo separar os dados em grupos. O significado destes grupos terá que ser clarificado posteriormente;
- Aprendizado por reforço: feito a partir da tentativa e erro, em que não usará rótulos nem conjuntos de dados pré-determinados como entrada, com o objetivo de agir corretamente. A tomada de decisões é feita por um agente que será treinado a partir de um contato direto com o ambiente, tomada de decisões anteriores e observações dos efeitos criados por estas.

Como essa proposta tem por objetivo fazer um sistema chegar a um determinado estado a partir de uma sequência de ações, a área de aprendizado de máquina que mais se encaixa é a de aprendizado por reforço.

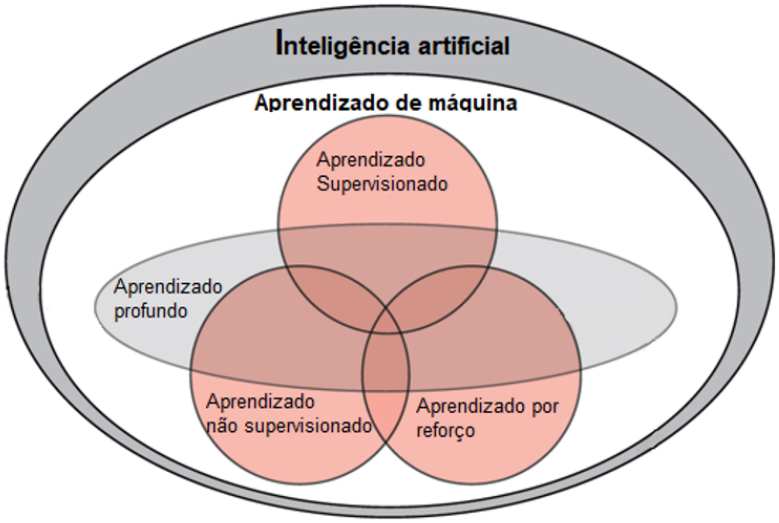
Além disso, para representar o agente a ser treinado para atuar sobre o ambiente, será utilizada uma rede neural artificial como estrutura para aproximação de função. E, assim, ao empregar o aprendizado de máquina junto com estruturas de redes neurais artificiais, fará com que o modelo esteja, não necessariamente num novo ramo do aprendizado de máquina, mas sim numa região que abrange uma parte das três áreas principais do aprendizado de máquina. Essa ideia pode ser representada pela Figura 6.

2.4 Redes neurais artificiais

É uma estrutura composta por diversas camadas de transformações matemáticas de forma a aproximar uma função não linear, em que será aplicada um ou mais valores de entrada de forma a se obter uma ou mais saídas.

De acordo com Morales (2020), redes neurais artificiais, apesar de consumirem uma quantidade grande de dados, geralmente são as estruturas que apresentam os melhores resultados. Porém, é importante lembrar que o ponto negativo, do alto consumo de dados, não necessariamente é um problema caso gerar dados em grande quantidade seja uma tarefa fácil para a problemática trabalhada.

Figura 6 – Representação do aprendizado de máquina dentro de inteligência artificial

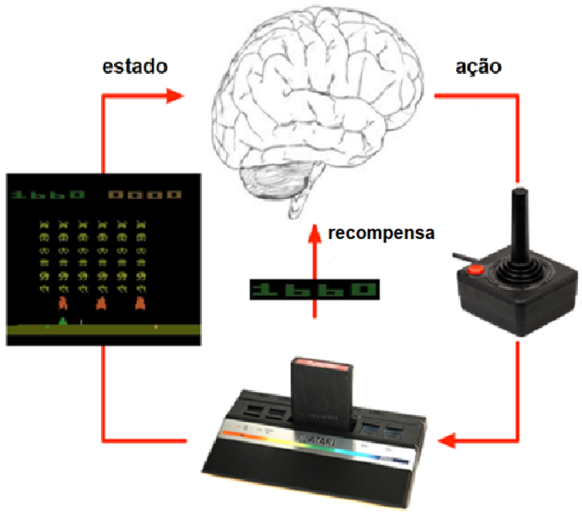


Fonte: Morales (2020).
Nota: Adaptado pelo autor.

2.5 Aprendizado por reforço

Neste tipo de aprendizado, o agente irá interagir diretamente com o ambiente e tomar ações de acordo com suas observações. Para essa aplicação é necessário algum *feedback* para que seja possível saber se o agente está agindo ou não de forma correta. Este raciocínio pode ser ilustrado pela Figura 7, em que iterativamente o agente, representado pelo cérebro, observa o estado do ambiente, retratado pelo console de videogame, e toma alguma ação sobre este, recebendo então uma recompensa relativa a qualidade da ação escolhida.

Figura 7 – Representação do aprendizado por reforço



Fonte: Keon (2017).
Nota: Adaptado pelo autor.

O sistema utilizado pelo agente para a tomada de decisões neste projeto é um modelo neural. Porém, a forma que esse modelo neural é usado, construído e treinado será baseada em um algoritmo de aprendizado por reforço chamado *Q-learning*.

2.5.1 *Q-learning*

De acordo com Watkins e Dayan (1992), *Q-learning* é uma forma de aprendizado por reforço. Nela, para cada estado e ação possível existirá um valor-Q, representando assim a qualidade da escolha daquela ação para determinado estado. Esse formato pode ser visto na Tabela 1. Esses valores-Q podem ser iniciados aleatoriamente ou baseados em alguma estimativa, para serem atualizados aos poucos via um processo iterativo.

Tabela 1 – Exemplo de tabela de decisão utilizada por *Q-learning*.

Estado	**Ação**	**valor-Q**
s_1	a_1	$Q(s_1, a_1)$
s_1	a_2	$Q(s_1, a_2)$
s_2	a_1	$Q(s_2, a_1)$
s_2	a_2	$Q(s_2, a_2)$

Fonte – Produção do próprio autor

A atualização do valor-Q é feita a partir da seguente expressão:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma max_{a'}(Q(s',a') - Q(s,a))), \tag{2.1}$$

onde s é o estado atual, s' é o estado seguinte, a é a ação atual, a' é a ação seguinte, r é a recompensa imediata, $Q(s,a)$ representa o valor-Q quando o agente está no estado atual s e efetua a ação a, $Q(s',a')$ representa o valor-Q quando o agente está no estado seguinte s' e efetua a ação a', α é a taxa de aprendizado, γ é a taxa de desconto, que permite ajustar a importância da recompensa ao longo do tempo, de modo que, quanto mais tarde for recebida a recompensa menos atrativa ela deva ser.

Importante notar que, devido à natureza de funcionamento do *Q-learning*, a quantidade de valores-Q cresce exponencialmente com a quantidade de estados e ações. Esta abordagem torna-se inviável para sistemas complexos, pelo grande espaço de memória ocupado e tempo necessário para explorar cada combinação de espaço e ação.

O funcionamento do *Q-learning* pode ser visto no Algoritmo 2.

Algorithm 2 Pseudo código do algoritmo *Q-learning*

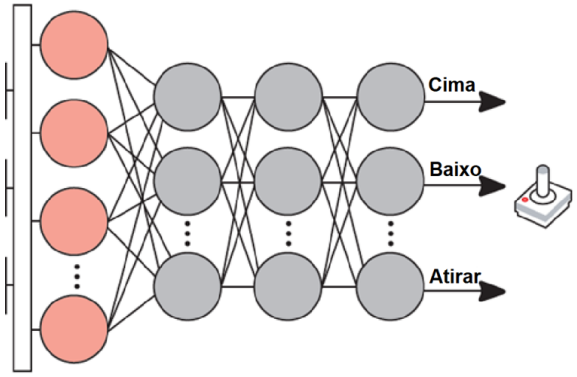
Inicializar: α
Inicializar a Tabela: Q
Inicializar: $\gamma, \epsilon, \epsilon_{min}$
Inicializar: $s_0 \leftarrow Ambiente()$
for $episodio = 1, \cdots, E$ **do**
  //Dado o estado inicial
  $s \leftarrow s_0$
  for $passo = 1, \cdots, T$ **do**
    //Escolher uma ação a partir do estado s usando *epsilon-greedy*:
    if $ValorAleatorio(0, 1) < \epsilon$ **then**
      $a \leftarrow AcaoAleatória(AcoesPossiveis)$
    else
      $a \leftarrow argmax_a\{Q(s, a)\})$
    end if
    //Atualize a taxa ϵ
    $\epsilon \leftarrow AtualizarTaxa(\epsilon, \epsilon_{min})$
    //Execute a ação a (no ambiente) e observe a recompensa r
    //e o próximo estado s'.
    $(r, s', done) \leftarrow Ambiente(a)$
    //Atualize a tabela Q
    if *done* **then**
      $Q(s, a) \leftarrow Q(s, a) + \alpha(r - Q(s, a))$
    else
      $Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma max_{a'}\{Q(s', a')\} - Q(s, a))$
    end if
    //Atualize o estado.
    $s \leftarrow s'$
  end for
end for

2.5.2 *Deep Q-Learning*

De forma a solucionar o problema citado anteriormente, é possível usar um modelo neural para a tomada de decisão do agente. Para que este possa ser treinado, é preciso que uma saída predita seja comparável a um resultado esperado, tendo assim um erro associado.

Seguindo a ideia do *Q-learning*, em vez de se ter uma rede com uma saída em forma de ação, na verdade, é utilizada a previsão das possíveis recompensas para cada possível ação a ser desempenhada. Esse formato pode ser visto na Figura 8, onde o agente só possui três ações que são “subir”, “descer” e “atirar”. A escolha de cada ação é feita selecionando a recompensa respectiva que tiver o maior valor.

Figura 8 – Rede neural adaptada ao aprendizado por reforço



Fonte: Morales (2020).
Nota: Adaptado pelo autor.

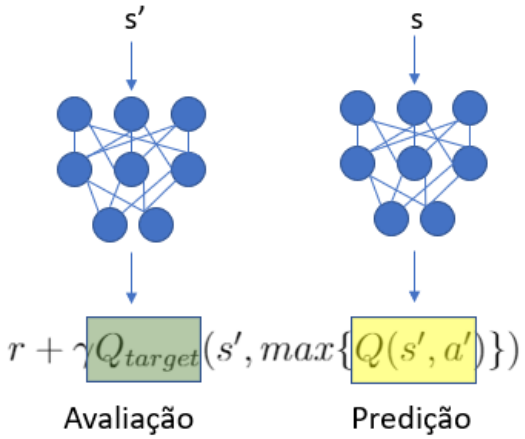
O treinamento então pode ser feito alterando os pesos da rede com o objetivo de prever adequadamente as recompensas para cada ação, de acordo com o estado do ambiente observado, a partir do erro. O cálculo do erro é feito baseado na equação de atualização dos valores-Q do modelo do *Q-learning*:

$$L = (r + \gamma max_{a'}\{Q(s', a') - Q(s, a))^2. \tag{2.2}$$

Diferente do efetuado em um treinamento supervisionado em que é utilizado um conjunto de dados, que pode ser subdividido em conjuntos de treinamento, validação e teste, no aprendizado por reforço a criação dos dados e o próprio treinamento são feitos simultaneamente. A medida que o agente efetua ações no ambiente, são registradas em uma memória as ações, os estados e as recompensas. Essa cadeia é considerada um passo e uma sequência de passos até o momento de reinicialização do ambiente é definido como um episódio, ou seja, uma sucessão de passos até o momento que não seja mais possível efetuar uma ação no ambiente. O treinamento pode ocorrer a cada passo ou episódio, dependendo da possibilidade de receber recompensas a cada ação ou uma sequência delas. A cada etapa de treinamento é utilizada uma amostra das tuplas armazenadas na memória, chamada *mini-batch*, para efetuar um rápido e pequeno treinamento.

Como explicado, o cálculo do erro para o treinamento do modelo neural é dado pela Equação (2.2). Percebe-se que a mesma rede é usada tanto para a predição como avaliação, porém, com o tempo, isso pode levar à superestimação das ações. Para resolver este problema, são criadas duas redes separadas para predição e avaliação, conforme vistas na Figura 9 onde Q é a rede de predição e Q_{target} é a rede de avaliação. Assim, ainda teremos a rede Q decidindo a provável ação futura para avaliação, porém com a rede Q_{target} qualificando essa escolha.

Figura 9 – Demonstração de cada rede atuando em parte do cálculo do erro



Fonte: Produção do próprio autor.

Diferente da rede Q, que terá seus parâmetros treináveis alterados normalmente ao longo das etapas ou episódios, a rede Q_{target} é atualizada para ser igual à rede Q a cada C passos de treinamento. Assim a rede Q_{target} estará, em relação ao treinamento, sempre defasada quando comparada à rede Q, evitando assim a superestimação das ações.

Um pseudo código representando o algoritmo DQN pode ser visto no Algoritmo 3.

2.6 Melhoras do algoritmo DQN

2.6.1 Priorização no replay

Já foi explicado anteriormente, na seção 2.5.2, que para o treinamento do modelo neural são utilizados dados de estado do ambiente, ação, recompensa e estado do ambiente após ação. Estes dados advêm de um conjunto de dados acumulados e a seleção do *mini-batch* para cada etapa de treinamento é feita, a priori, aleatoriamente. Seguir o esquema descrito faz com que dados que tenham informação com muito a ser aprendido sejam tratados igualmente aos que não, o que não é muito efetivo já que se busca treinar a rede com os dados de melhor qualidade possível.

Assim, para melhorar a eficácia do treinamento, é possível criar uma lógica de priorização dos dados. Quanto maior for o erro entre o resultado previsto e o verdadeiro, maior é a prioridade desse dado para o treinamento, fazendo com que o *mini-batch* de treinamento sempre tenha dados com maior importância para a geração de melhores resultados do

Algorithm 3 Pseudo código do algoritmo DQN

Inicializar o *memory replay*: $M \leftarrow MemoryReplay(Capacidade)$
Inicializar os Pesos: $\theta \leftarrow PesosAleatórios$
Inicializar a Rede Q: $Q(\bullet;\theta)$
Inicializar a Rede Q alvo: $Q_{target}(\bullet;\theta^{-})$; $\theta^{-} = \theta$
Inicializar: $\gamma, \epsilon, \epsilon_{min}$
Inicializar: $s_0 \leftarrow Ambiente()$
for $episodio = 1, \cdots, E$ **do**
  //Dado o estado inicial
  $s \leftarrow s_0$
  for $passo = 1, \cdots, T$ **do**
    //Escolher uma ação a partir do estado s usando *epsilon-greedy*:
    if $ValorAleatorio(0,1) < \epsilon$ **then**
      $a \leftarrow AcaoAleatória(AcoesPossiveis)$
    else
      $a \leftarrow argmax_a\{Q(s,a)\})$
    end if
    //Atualize a taxa ϵ-*greedy*
    $\epsilon \leftarrow AtualizarTaxa(\epsilon, \epsilon_{min})$
    //Execute a ação a (no ambiente) e observe a recompensa r
    //e o próximo estado s'.
    $(r, s', done) \leftarrow Ambiente(a)$
    //Guarde a tupla de experiência $(s, a, r, s, done)$ na memoria M.
    $M \leftarrow RegistrarTupla((s, a, r, s', done), M)$
    //Extraia aleatoriamente um *mini-batch* de tuplas da memoria M
    $\mathcal{B} \leftarrow SelecionarTuplasAleatoriamente(M)$
    //Determine o valor de Q_{max} para cada tupla do *mini-batch*
    for $(s_i, a_i, r_i, s'_i, done_i) \in \mathcal{B}$ **do**
      if $done_i$ **then**
        $Q_{max_i} \leftarrow r_i$
      else
        $Q_{max_i} \leftarrow r_i + \gamma max_{a'}\{(Q_{target}(s'_i, a')\}$
      end if
    end for
    //Calcule a perda considerando as tuplas do *mini-batch*
    $L \leftarrow \frac{1}{|\mathcal{B}|}\sum_{i=1}^{|\mathcal{B}|}(Q_{max_i} - Q(s_i, a_i))^2$
    //Atualizar os pesos da rede Q para minimizar a perda L
    $\theta \leftarrow SGD(\theta, L)$
    //Atualizar a rede Q_{target} a cada C passos
    if $passo\%C == 0$ **then**
      $Q_{target} = Q$
    end if
    //Atualize o estado.
    $s \leftarrow s'$
  end for
end for

sistema.

Infortunadamente, gerar *mini-batchs* apenas com dados de prioridade alta pode acabar por causar *overfitting* já que sempre os mesmos dados serão utilizados. Para contornar este problema, foi adicionado um fator aleatório durante a escolha das amostras para o *mini-batch*.

2.6.2 *Multi-step Bootstrapping*

Também criado para aumentar a eficiência do treinamento, a ideia é conseguir propagar para a rede mais rapidamente resultados novos observados em que na sequência tenha-se um estado já visitado anteriormente. Para se fazer isso é necessário alterar como é feito o cálculo do erro para treinamento. Em vez de se utilizar a recompensa do estado e a possível recompensa do estado seguinte, são empregadas as recompensas de vários estados para frente. Essa formulação pode ser vista para os erros abaixo para cada passo a frente:

$$\begin{aligned}
\text{Um passo} &: ((r + \gamma V(s')) - Q(s,a))^2 \\
\text{Dois passos} &: ((r + \gamma V(s')) + \gamma^2 V(s'')) - Q(s,a))^2 \\
\text{Três passos} &: ((r + \gamma V(s)) + \gamma^2 V(s'')) + \gamma^3 V(s''')) - Q(s,a))^2 \\
&\vdots \\
P \text{ passos} &: ((r + \sum \gamma^k V(s_k)) - Q(s,a))^2.
\end{aligned}$$

Importante notar que a variável γ anteriormente usada para priorizar a recompensa presente em relação ao que se pode ter, continua sendo utilizada de forma exponencial de acordo com quanto futura é a possível recompensa, priorizando a recompensa da ação quanto antes ela vir.

A seguir, em Algoritmo 4, é mostrado um pseudo código do DQN já efetuadas as modificações para melhoria da eficiência do algoritmo.

Algorithm 4 Pseudo código do algoritmo DQN com modificações

Inicializar o *memory replay*: $M \leftarrow MemoryReplay(Capacidade)$
Inicializar Pesos: $\theta \leftarrow PesosAleatórios$
Inicializar Rede Q: $Q(\bullet;\theta)$
Inicializar Rede Q alvo: $Q_{target}(\bullet;\theta^{-})$; $\theta^{-} = \theta$
Inicializar: $\gamma, \epsilon, \epsilon_{min}$
Inicializar: $s_0 \leftarrow Ambiente()$
for $episodio = 1, \cdots, E$ **do**
  //Dado o estado inicial
  $s \leftarrow s_0$
  for $passo = 1, \cdots, T$ **do**
    //Escolher uma ação a partir do estado s usando *epsilon-greedy*:
    if $ValorAleatorio(0,1) < \epsilon$ **then**
      $a \leftarrow AcaoAleatória(AcoesPossiveis)$
    else
      $a \leftarrow argmax_a\{Q(s,a)\}$
    end if
    //Atualize a taxa ϵ
    $\epsilon \leftarrow AtualizarTaxa(\epsilon, \epsilon_{min})$
    //Execute a ação a (no ambiente) e observe a recompensa r
    //e o próximo estado s'.
    $(r, s', done) \leftarrow Ambiente(a)$
    //Guarde a tupla de experiência $(s, a, r, s, done)$ na memoria M.
    $M \leftarrow RegistrarTupla((s, a, r, s', done), M)$
    //Extraia aleatoriamente um *mini-batch* de tuplas da memoria M
    $\mathcal{B} \leftarrow SelecionarTuplasPriorizandoMaiorErro(M)$
    //Determine o valor de Q_{max} para cada tupla do *mini-batch*
    for $(s_i, a_i, r_i, s'_i, done_i) \in \mathcal{B}$ **do**
      if *done* **then**
        $Q_{max_i} \leftarrow r_i$
      else
        $Q_{max_i} \leftarrow r_i + \sum_{k=1}^{P} \gamma^k Q_{target}(s_i^k, argmax_a\{Q(s_i^k, a)\})$
      end if
    end for
    //Calcule a perda para o *mini-batch*
    $L \leftarrow \frac{1}{|\mathcal{B}|} \sum_{i=1}^{|\mathcal{B}|} (Q_{max_i} - Q(s_i, a_i))^2$
    //Atualizar os pesos da rede Q usando SGD objetivando minimizar a perda L
    $\theta \leftarrow SGD(Q, L)$
    //Atualizar a rede Q_{target} a cada C passos
    if $resto(passo/C) == 0$ **then**
      $Q_{target} = Q$
    end if
    //Atualize o estado.
    $s \leftarrow s'$
  end for
end for

3 PROPOSTA

3.1 Componentes de Aprendizado por Reforço Desenvolvido

Considerando que a proposta está baseada em aprendizado por reforço profundo, ela apresentará a seguinte lógica: o ambiente é um braço robótico que coloca caixas de um contêiner de entrada em um contêiner de organização, enquanto que, o agente é um modelo neural que recebe o estado observado do ambiente e gera como saída as recompensas previstas para cada ação possível, sendo escolhida a ação com maior recompensa. Tal ação é aplicada no ambiente, o qual efetua a ação de pegar uma caixa no contêiner de entrada e colocar no contêiner de organização gerando assim um novo estado do ambiente. No fluxograma da Figura 10 é explicitada a interação entre esses componentes.

Figura 10 – Relação entre componentes do sistema de aprendizado por reforço

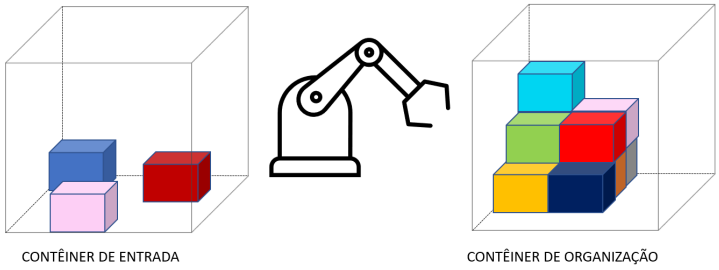
Fonte: Produção do próprio autor.

A seguir, são explicados os diferentes componentes de uma abordagem baseada em aprendizado por reforço para o problema em estudo. Iniciando com a explicação da construção do ambiente, seguido pela descrição dos estados que o ambiente pode assumir, como das ações a serem tomadas pelo agente e as recompensas que ele recebe, finalizando com a descrição da modelagem do agente via um modelo neural.

3.1.1 Ambiente

Devido a natureza da solução proposta, é necessária a existência de um ambiente onde o agente, com um modelo neural responsável pela tomada de decisões, possa ser treinado continuamente até atingir resultados satisfatórios. Uma representação visual desse sistema pode ser vista na Figura 11.

Figura 11 – Ambiente virtual com os contêineres de entrada e de organização



Fonte: Produção do próprio autor.

Percebe-se que há nesse ambiente dois contêineres, isso acontece porque existe um contêiner de onde vêm as caixas de entrada geradas aleatoriamente, chamado de contêiner de entrada, e o outro onde o braço robótico irá colocar as caixas uma por uma de forma organizada, denominado contêiner de organização que sempre será inicializado vazio. A cada movimentação de pegar e colocar uma caixa o agente receberá uma recompensa proporcional à qualidade da ação escolhida por ele, assim podendo ter sua rede treinada durante o processo ao receber constantemente dados de observação do ambiente, ações realizadas e recompensas.

Algumas regras ficam estabelecidas em relação a esse ambiente:

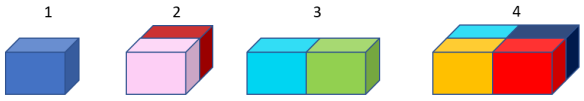
- Cada caixa do contêiner de entrada só pode ser retirada se não houver outra caixa por cima desta;
- O contêiner onde as caixas são organizadas só pode ter as caixas de entrada colocadas por cima. Ou seja, espaços não preenchidos que ficaram por baixo não poderão mais ser acessados;
- Um novo conjunto de caixas de entrada serão geradas no contêiner de entrada sempre que este ficar sem caixas;
- Caso o agente erre ao tentar pegar uma caixa do contêiner de entrada ou passe da altura limite ao colocar uma caixa no contêiner de organização, reiniciará o ambiente

esvaziando o contêiner de organização e gerando um novo conjunto de caixas para o contêiner de entrada.

As dimensões dos contêineres de entrada e de organização utilizadas no projeto foram iguais e de $4 \times 4 \times 4$, porém nada impede o uso de contêineres de tamanhos diferentes. Essa escolha foi baseada apenas na redução de custo computacional para acelerar o treinamento e consequentemente os testes dos modelos.

O treinamento dos modelos neurais necessita de conjuntos de caixas a partir do contêiner de entrada. Para isso, foi criado um procedimento simples para criar conjuntos aleatórios de caixas. Esse procedimento escolhe aleatoriamente quais pontos da superfície discretizada do contêiner de entrada receberão uma caixa, no qual seu tipo também será selecionado aleatoriamente entre os pré-determinados mostrados na Figura 12.

Figura 12 – Tipos de caixas pré-determinados para serem trabalhados no projeto



Fonte: Produção do próprio autor.

Uma regra foi determinada para não empilhar caixas uma em cima da outra, permitindo a visualização de todas as caixas no contêiner de entrada por parte do agente. A seguir é apresentado um pseudo código desse procedimento no Algoritmo 5.

Algorithm 5 Pseudo código da geração das caixas de entrada

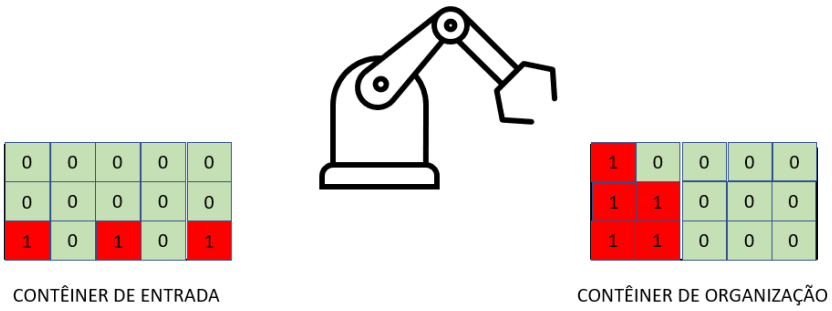
```
for EspaçosDiescretizados do
    if ValorAleatorio(0,1) < PorcentagemColocarCaixa then
        NovaCaixa ← GerarCaixaAleatória(ConjuntoReferência)
        if SemConflitoParaColocarCaixa(Posição,Caixa,Ambiente) then
            AmbienteAtualizado ← AlocarCaixa(Posição, NovaCaixa, Ambiente)
        end if
    end if
end for
```

3.1.2 Estado do Ambiente

O estado do ambiente precisa ser representado de alguma forma que possa ser utilizado como dado de entrada para o modelo neural. Assim, o ambiente foi discretizado, tendo os espaços ocupados representados por 1 e os espaços livres por 0, conforme a representação

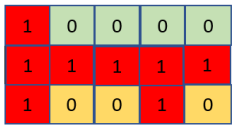
mostrada na Figura 13. Devido à não possibilidade de alocação de caixas abaixo de caixas já posicionadas, conforme mostrado na Figura 14, é viável a redução de uma dimensão na representação do estado de um contêiner utilizando o espaço disponível a cada ponto como mostrado na Figura 15. Assim, tem-se o espaço vertical disponível de cima para baixo em cada ponto da discretização do espaço, como se fosse a visão superior do contêiner. Na Figura 16 essa representação é mostrada para um contêiner junto com sua respectiva ocupação tridimensional.

Figura 13 – Representação bidimensional da discretização do espaço do ambiente virtual



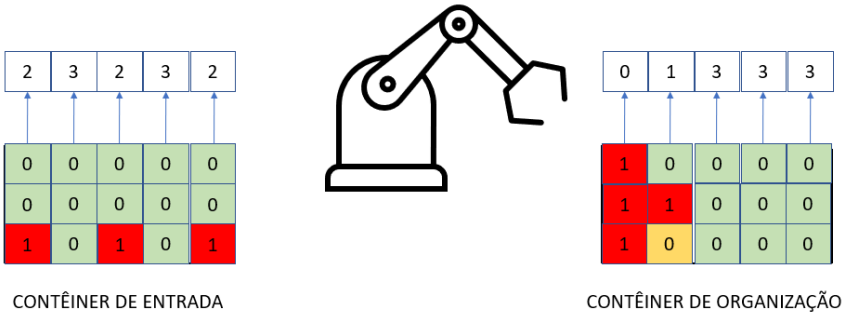
Fonte: Produção do próprio autor.

Figura 14 – Representação bidimensional de espaços desperdiçados, em amarelo, pois não poderão mais ser preenchidos



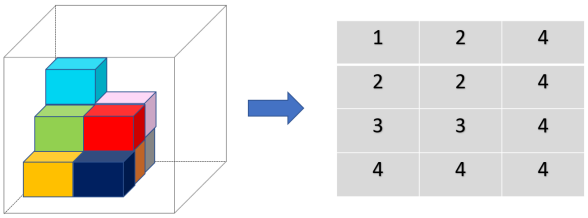
Fonte: Produção do próprio autor.

Figura 15 – Demonstração em bidimensional da redução de uma dimensão ao utilizar a altura disponível



Fonte: Produção do próprio autor.

Figura 16 – Exemplo de representação espacial da alocação das caixas no espaço tridimensional



Fonte: Produção do próprio autor.

3.1.3 Ações do Agente

A quantidade de saídas do modelo neural responsável pela tomada de decisão é definida pela quantidade de ações possíveis, e logo deve ser fixa. Para definir essa quantidade de forma fixa foi estipulado: primeiro, o agente escolhe um dos pontos discretizados do contêiner de entrada para pegar uma caixa; depois o agente deve escolher um dos pontos discretizados do contêiner de organização para colocar a caixa, considerando também a possibilidade de rotacionar a caixa que pegou antes de colocar no contêiner de organização. Seguindo esse procedimento, o número de ações será igual a:

$$A = M_b \times N_b \times N_t \times M_t \times R,$$

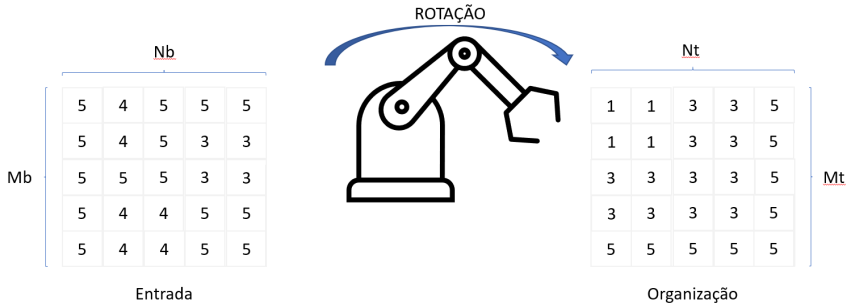
onde A é o número de ações possíveis, M_b é o número de linhas da matriz que representa o contêiner de entrada, N_b é o número de colunas da matriz que representa o contêiner de entrada, M_t é o número de linhas da matriz que representa o contêiner de organização, N_t é o número de colunas da matriz que representa o contêiner de organização, e R é o número de rotações disponibilizadas para o agente efetuar.

Na Figura 17, é observada esta situação, onde são considerados que: $M_b = 5$, $N_b = 5$, $M_t = 5$, $N_t = 5$ e $R = 4$, por tanto o número de ações possíveis será $A = 2.500$.

3.1.4 Recompensas

Em aprendizado por reforço, é necessário definir quantitativamente a qualidade de cada ação efetuada. Assim, decidiu-se usar o próprio valor do volume da caixa alocada no contêiner de organização como recompensa para a ação efetuada. Dessa forma, quando o agente não consegue pegar ou colocar uma caixa a recompensa será 0.

Figura 17 – Representação da discretização para construção das ações



Fonte: Produção do próprio autor.

Porém, essa metodologia resultará em recompensas iguais, para uma mesma caixa, para ações evidentemente com qualidades diferentes, como visto na Figura 18, em que a alocação à esquerda possui uma maior qualidade por não deixar um espaço vazio em baixo que nunca será preenchido ao longo das próximas ações.

Figura 18 – Dois tipos de alocações e suas qualidades

Fonte: Produção do próprio autor.

De forma a sanar este problema, foi estabelecido que a recompensa seria em função do espaço disponível abaixo dele. Assim, têm-se o sistema de recompensas para a alocação descrito matematicamente da seguinte forma:

$$r = \frac{b}{bi} \times v,$$

onde r é a recompensa adquirida, b é a base da caixa colocada, bi é a base preenchida inferior à base da caixa colocada, e v é o volume da caixa colocada.

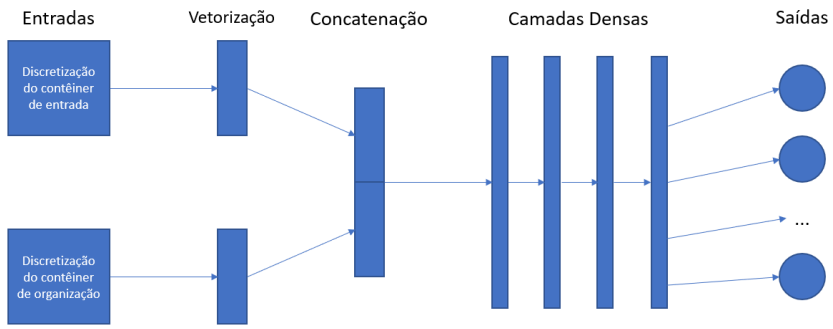
3.1.5 Agente

Foram criados dois modelos neurais que podem representar o agente. Ambos os modelos têm como entrada duas matrizes que correspondem aos estados dos contêineres de entrada e de organização, e como saída um vetor contendo a qualidade correspondente a cada

possível ação que pode ser tomada pelo agente. Devido a esta configuração de entradas e saídas, ambos os modelos têm uma estrutura de dois ramos de entrada unidos por uma operação de concatenação e um ramo de saída. Especificamente: (*i*) o primeiro modelo, representado na Figura 19, vetoriza as matrizes de entrada e as concatena, constituindo-se seu ramo de saída por uma sequência de camadas densas; (*ii*) o segundo modelo, mostrado na Figura 20, apresenta como ramos de entrada uma sequência de camadas convolucionais, cujos mapas de características de saída são vetorizados e concatenados, tal vetor resultante é propagado por uma sequência de camadas densas de forma similar ao primeiro modelo.

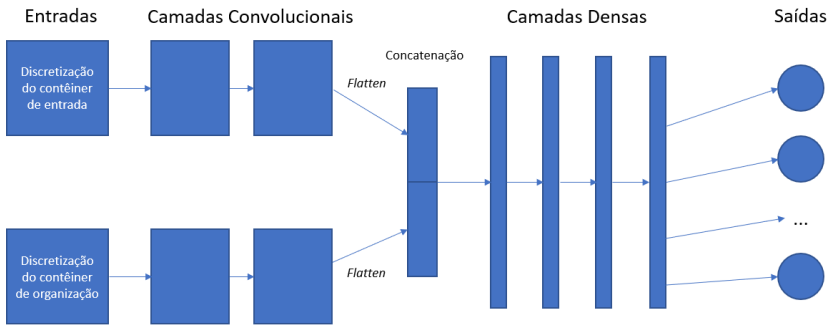
De forma resumida, o primeiro modelo ao usar camadas densas na entrada não considera nenhum tipo de correlação de vizinhança nos dados de entrada, enquanto que, o segundo modelo, ao serem usadas camadas convolucionais, considera.

Figura 19 – Representação de uma rede de camadas densas



Fonte: Produção do próprio autor.

Figura 20 – Representação de uma rede com camadas convolucionais e densas



Fonte: Produção do próprio autor.

4 RESULTADOS

4.1 Recursos Computacionais

Recursos de *Software*: As etapas de treinamento e teste dos modelos neurais foram realizadas utilizando *Tensorflow*, *software* de código aberto desenvolvido pela *Google brain Team*[1]. Todo o código nesse projeto utilizou linguagem de programação Python e foi desenvolvido e executado no *Google Colaboratory*, uma ferramenta em nuvem que permite criar e executar códigos na linguagem Python diretamente pelo navegador.

Recursos de *Hardware*: A máquina utilizada é a disponível pelo *Google Colab* na versão Pro. Esta possui a seguinte configuração: (*i*) processador Intel(R) Xeon(R), 2.20GHz com 1 núcleo físico disponível; (*ii*) memória RAM de 16 GB; (*iii*) unidade de armazenamento de 157GB disponível; (*iv*) placa de vídeo Tesla K80, com 12 GB de memória dedicada.

4.2 Estrutura das arquiteturas implementadas

Como já foi indicado na Seção 3.1.5, duas arquiteturas de modelos neurais foram implementadas e podem ser vistas nas Tabelas 2 e 3.

Tabela 2 – Arquitetura do modelo neural com camadas convolucionais

Camada (tipo)	**Formato da saída**	**Nº de parâmetros**	**Conectada a**
Entrada A	(4, 4, 1)	0	
Entrada B	(4, 4, 1)	0	
$\text{Convolucional}_{A1}(2,2)$	(4, 4, 64)	320	Entrada A
$\text{Convolucional}_{A2}(2,2)$	(4, 4, 64)	320	$\text{Convolucional}_{A1}$
$\text{Convolucional}_{B1}(2,2)$	(4, 4, 64)	16448	Entrada B
$\text{Convolucional}_{B2}(2,2)$	(4, 4, 64)	16448	$\text{Convolucional}_{B1}$
Flatten A	1024	0	$\text{Convolucional}_{A2}$
Flatten B	1024	0	$\text{Convolucional}_{B2}$
Concatenação	2048	0	Flatten A e Flatten B
Densa_1	1024	2098176	Concatenação
Densa_2	512	524800	Densa_1
Saída	256	131328	Densa_2

Fonte – Produção do próprio autor

[1] <https://research.google.com/teams/brain/>

Tabela 3 – Arquitetura do modelo neural de camadas densas

Camada (tipo)	**Formato da saída**	**Nº de parâmetros**	**Conectada a**
Entrada A	(4, 4, 1)	0	
Entrada B	(4, 4, 1)	0	
Flatten A	16	0	Entrada A
Flatten B	16	0	Entrada B
Concatenação	32	0	Flatten A e Flatten B
Densa_1	64	2112	Concatenação
Densa_2	64	4160	Densa_1
Densa_3	64	4160	Densa_2
Densa_4	128	8320	Densa_3
Saída	256	33024	Densa_4

Fonte – Produção do próprio autor

4.3 Descrição dos Hiperparâmetros

Os valores dos hiperparâmetros usados para o treinamento dos dois modelos neurais estão listados nas Tabelas 4 e 5. Esses hiperparâmetros foram escolhidos empiricamente, a partir de múltiplas execuções dos modelos avaliados.

Tabela 4 – Hiperparâmetros da rede com camadas convolucionais e do aprendizado por reforço utilizado

Hiperparâmetro	**valor**
Taxa de aprendizado de Adam	0,0025
Tamanho do *Minibatch*	64
Fator de desconto (γ)	0,95
Decaimento da aleatoriedade (ϵ_d)	0,999
Aleatoriedade mínima (ϵ_{min})	0,01
Nº de ações para atualizar rede de avaliação	400
Passos máximos do *multi-step*	4

Fonte – Produção do próprio autor

Tabela 5 – Hiperparâmetros da rede com apenas camadas densas e do aprendizado por reforço utilizado

Hiperparâmetro	**valor**
Taxa de aprendizado de Adam	0,005
Tamanho do *Minibatch*	64
Fator de desconto (γ)	0,95
Decaimento da aleatoriedade (ϵ_d)	0,999
Aleatoriedade mínima (ϵ_{min})	0,01
N^o de ações para atualizar rede de avaliação	400
Passos máximos do *multi-step*	4

Fonte – Produção do próprio autor

4.4 Experimentos

4.4.1 Métricas

Devido a natureza do aprendizado por reforço, não existe uma métrica padrão em si a ser utilizada para comparar resultados. Assim, para avaliar os experimentos foram usados os valores de máximo e média das recompensas dos episódios para 10.000 episódios.

4.4.2 Experimentos efetuados

Colocando os dois modelos para treinar iterativamente com o ambiente virtual com os hiperparâmetros mostrados foram obtidos os seguintes gráficos de média de recompensa dos episódios ao longo de sequências de passos de treinamento, mostrados na Figura 21. Observa-se que, na evolução da pontuação dos dois modelos, apesar de conseguirem resultados cada vez maiores, ainda existe uma lacuna entre os resultados adquiridos e o melhor resultado possível que é 1.

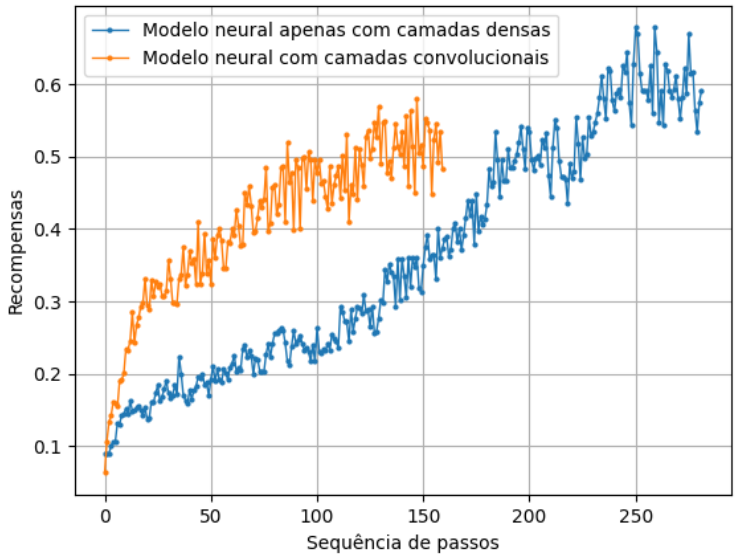
No modelo neural com camadas convolucionais percebe-se uma evolução mais rápida da pontuação no geral quando comparada com o modelo de camadas densas. Porém ainda mais rápido logo no início. Suspeitou-se que esse comportamento no início se deveu ao agente aprendendo primeiramente a sempre pegar uma caixa, já garantindo assim um rápido crescimento da pontuação no início. Para conferir essa suposição, o sistema de recompensa foi alterado para recompensar apenas pegando caixas, onde um resultado parecido pode ser mostrado na Figura 22.

Diferente dos resultados do modelo com camadas convolucionais, o modelo com camadas densas tem um começo lento e, inclusive, com um crescimento similar durante quase todo o treinamento. Apesar desse começo lento, este conseguiu melhores resultados.

4.5 Comparação com outros trabalhos

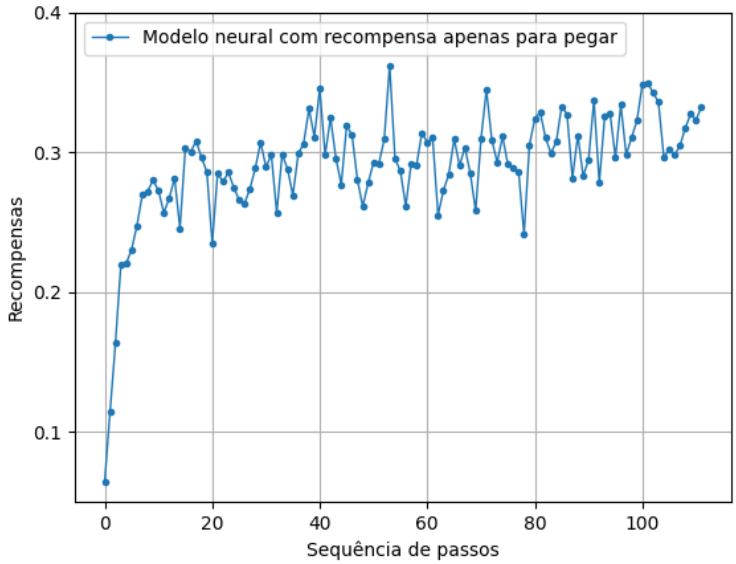
A comparação foi feita entre os dois modelos de rede implementadas no projeto e o modelo heurístico DBLF usando também o ambiente virtual criado. Nessa avaliação foram utilizados para cada modelo 10.000 episódios completamente aleatórios, onde os valores adquiridos são referentes à recompensa acumulada nos episódios. Os valores adquiridos

Figura 21 – Média das recompensas acumuladas dos episódios a cada 25.000 e 50.000 passos para o modelo com camada convolucional e com camada densa, respectivamente



Fonte: Produção do próprio autor.

Figura 22 – Média das recompensas acumuladas dos episódios a cada 25000 passos da rede com camadas convolucionais com sistema de recompensa apenas para pegar a caixa, independente do resultado na alocação



Fonte: Produção do próprio autor.

de cada um são apresentados na Tabela 6. Ambos modelos neurais apresentaram um desempenho marginalmente superior ao DBLF, mas entre os dois modelos neurais o que apresentou melhor desempenho foi aquele constituído unicamente por camadas densas, o

que pode ser um indicativo de que, apesar das entradas serem matrizes, a informação de correlação de vizinhança não é preponderante para o problema.

Tabela 6 – Comparação das recompensas dos episódios das propostas deste trabalho com o método heurístico DBLF.

Agente	**Média**	**Desvio Padrão**
DBLF	0,458	0,051
Redes com camadas convolucionais	0.524	0,042
Redes apenas de camadas densas	0,592	0,035

Fonte – Produção do próprio autor

Além da média das recompensas do modelo de camadas densas ser 29% superior e o desvio padrão ser 32% inferior que o método DBLF, na Tabela 7 pode ser visto que o tempo de execução do modelo é tão pequeno quanto o DBLF. Isso mostra a capacidade da utilização do modelo para execuções em tempo real.

Tabela 7 – Comparação dos tempos, em milissegundos, de execução das propostas deste trabalho com o método heurístico DBLF.

Agente	**Média**	**Desvio Padrão**
DBLF	5,28	0,50
Redes com camadas convolucionais	6,68	0,52
Redes apenas de camadas densas	5,91	0,47

Fonte – Produção do próprio autor

5 CONCLUSÕES E PROJETOS FUTUROS

5.1 Conclusões

O objetivo principal deste trabalho foi propor uma abordagem ao problema de carregamento em contêiner. A proposta é baseada em aprendizado de máquina, mais especificamente na sub-área de aprendizado por reforço. Tal abordagem foi motivada pelo aumento expressivo do uso e aplicação do aprendizado de máquina no meio comercial e industrial a fim de atingir resultados cada vez melhores e mais rápidos.

Como resultado, usando um ambiente virtual próprio para simular a ação de um agente comandado por uma codificação com sua ações baseadas em um modelo neural, foram treinados dois agentes representados por dois modelos neurais, o primeiro modelo baseado em camadas densas e o segundo modelo baseado em camadas convolucionais. Os agentes resultantes desse treinamento foram então colocados no ambiente virtual para organizar caixas em um contêiner até o limite que conseguissem alocar caixas 10.000 vezes cada.

A partir dos valores de volume máximo e médio adquiridos por esses agentes foram então comparados com um modelo heurístico, mostrando o potencial dos modelos propostos em conseguir bons resultados mantendo a velocidade de modelos heurísticos na tomada de decisão.

Ao fim deste trabalho, conclui-se que o uso de modelos neurais como atuador na decisão de ações de um agente atuando no problema de carregamento em contêiner é eficaz, principalmente, para aplicações onde é necessária a organização de cargas em tempo real. Em especial, o uso de modelos de apenas camadas densas possuem resultados melhores em comparação a modelos com camadas convolucionais, apesar de precisarem de um tempo consideravelmente maior para o treinamento. Em suma, o trabalho em questão mostra a viabilidade do uso de aprendizado por reforço para a problemática de carregamento de contêineres servindo como uma referência a futuros trabalhos sobre o tema.

5.2 Recomendações

Durante o planejamento inicial do projeto não se esperava períodos tão longos para adquirir bons resultados ao longo do treinamento, que chegavam a durar dias, o que acabou por

limitar o trabalho em relação a otimização dos hiperparâmetros das redes. Assim, torna-se necessário contar com poder computacional para trabalhar com esta problemática considerando o tipo de abordagem proposta aqui.

Seria interessante analisar metodologias que permitam usar os modelos neurais em contêineres maiores do que estes que foram utilizados durante o treinamento. Uma ideia, por exemplo, seria usar as dimensões das entradas do contêineres de entrada e de organização, no qual o modelo foi treinado, como uma máscara que se desloca sobre os contêineres atuais e maiores em que o agente está atuando. É válido também, avaliar o efeito de aplicar contêineres de organização já inicializados, com cargas aleatórias, durante o treinamento.

A visualização dos dados deste projeto foram utilizadas a própria recompensa adquirida a cada episódio, porém para se ter uma ideia mais humana acerca dos resultados adquiridos recomenda-se verificar informações como: volumes preenchido e vazio, espaço vazio abaixo das caixas alocadas e visualização simulada de como foram realmente alocadas as caixas no contêiner de organização.

Finalmente, é sugerida a adição de algumas extensões do algoritmo *Deep Q-learning* que não foram utilizadas aqui. Sendo algumas delas: *Dueling network architeture* por Wang et al. (2016) para acrescentar estabilidade, *Distributional Q-learning* por Bellemare, Dabney e Munos (2017) e *Noisy DQN* por Fortunato et al. (2017) para melhorar a exploração.

REFERÊNCIAS

BELLEMARE, M. G.; DABNEY, W.; MUNOS, R. A distributional perspective on reinforcement learning. In: PMLR. International Conference on Machine Learning. [S.l.], 2017. p. 449–458. Citado na página 44.

BORTFELDT, A.; GEHRING, H. A hybrid genetic algorithm for the container loading problem. European Journal of Operational Research, Elsevier, v. 131, n. 1, p. 143–161, 2001. Citado na página 19.

BORTFELDT, A.; GEHRING, H.; MACK, D. A parallel tabu search algorithm for solving the container loading problem. Parallel computing, Elsevier, v. 29, n. 5, p. 641–662, 2003. Citado na página 13.

FORTUNATO, M.; AZAR, M. G.; PIOT, B.; MENICK, J.; OSBAND, I.; GRAVES, A.; MNIH, V.; MUNOS, R.; HASSABIS, D.; PIETQUIN, O. et al. Noisy networks for exploration. arXiv preprint arXiv:1706.10295, 2017. Citado na página 44.

GEORGE, J. A.; ROBINSON, D. F. A heuristic for packing boxes into a container. Computers & Operations Research, Elsevier, v. 7, n. 3, p. 147–156, 1980. Citado 4 vezes nas páginas 7, 13, 15 e 16.

GILMORE, P. C.; GOMORY, R. E. A linear programming approach to the cutting-stock problem. Operations research, INFORMS, v. 9, n. 6, p. 849–859, 1961. Citado na página 19.

GONÇALVES, J. F.; RESENDE, M. G. A parallel multi-population biased random-key genetic algorithm for a container loading problem. Computers & Operations Research, Elsevier, v. 39, n. 2, p. 179–190, 2012. Citado 3 vezes nas páginas 13, 16 e 19.

HOPPER, E. Two-dimensional packing utilising evolutionary algorithms and other meta-heuristic methods. Tese (Doutorado) — University of Wales. Cardiff, 2000. Citado na página 20.

JR, R. C. A. An approach to the two dimensional irregular cutting stock problem. Tese (Doutorado) — Massachusetts Institute of Technology, 1966. Citado na página 20.

KARABULUT, K.; İNCEOĞLU, M. M. A hybrid genetic algorithm for packing in 3d with deepest bottom left with fill method. In: SPRINGER. International Conference on Advances in Information Systems. [S.l.], 2004. p. 441–450. Citado na página 20.

KEON. Deep-q-learning. 2017. Disponível em: <https://keon.github.io/deep-q-learning/>. Citado na página 23.

MORALES, M. Grokking deep reinforcement learning. [S.l.]: Simon and Schuster, 2020. Citado 4 vezes nas páginas 21, 22, 23 e 26.

RAMESH UMASHANKAR, J. R. A hybrid multi-objective generic algorithm for the container loading problem. 2021. Disponível em: <https://github.com/Nivedha-Ramesh/Container-Loading-Problem/blob/master/Report.pdf>. Citado na página 21.

TANAKA, T.; KANEKO, T.; SEKINE, M.; TANGKARATT, V.; SUGIYAMA, M. Simultaneous planning for item picking and placing by deep reinforcement learning. In: IEEE. 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). [S.l.], 2020. p. 9705–9711. Citado 4 vezes nas páginas 7, 14, 16 e 17.

VENDRAMINI, E. Otimização do problema de carregamento de container usando uma metaheurística eficiente. Universidade Estadual Paulista (UNESP), 2007. Citado na página 14.

WANG, Z.; SCHAUL, T.; HESSEL, M.; HASSELT, H.; LANCTOT, M.; FREITAS, N. Dueling network architectures for deep reinforcement learning. In: PMLR. International conference on machine learning. [S.l.], 2016. p. 1995–2003. Citado na página 44.

WATKINS, C. J.; DAYAN, P. Q-learning. Machine learning, Springer, v. 8, n. 3-4, p. 279–292, 1992. Citado na página 24.

ZHENG, J.-N.; CHIEN, C.-F.; GEN, M. Multi-objective multi-population biased random-key genetic algorithm for the 3-d container loading problem. Computers & Industrial Engineering, Elsevier, v. 89, p. 80–87, 2015. Citado na página 13.