UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE ENGENHARIA DE PRODUÇÃO

Cleverton Bueno dos Santos Júnior

PATH-PLANNING UTILIZANDO UMA REDE NEURAL ARTIFICIAL DE APRENDIZADO PROFUNDO

Santa Maria, RS
2021

Cleverton Bueno dos Santos Júnior

PATH-PLANNING UTILIZANDO UMA REDE NEURAL ARTIFICIAL DE APRENDIZADO PROFUNDO

Trabalho de Conclusão de Curso apresentado ao curso de Graduação em Engenharia de Produção da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do título de **Engenheiro(a) de Produção.**

Orientador: Prof. Dr. Felipe Martins Müller

Santa Maria, RS
2021

PATH-PLANNING UTILIZANDO UMA REDE NEURAL ARTIFICIAL DE APRENDIZADO PROFUNDO

PATH-PLANNING USING A DEEP LEARNING ARTIFICIAL NEURAL NETWORK

Cleverton Bueno dos Santos Júnior[1], Felipe Martins Müller[2]

RESUMO

Este estudo busca verificar a possibilidade da utilização de técnicas avançadas de inteligência artificial como opção para o planejamento de caminhos de unidades de transporte em um armazém autônomo, baseando-se em um clássico problema de logística, conhecido como "Problema do Caixeiro Viajante" (PCV) um problema que tenta determinar a menor rota para percorrer uma série de cidades (visitando uma única vez cada uma delas). Foram utilizadas 2 redes neurais artificiais com arquiteturas diferentes e os resultados foram comparados à técnicas já utilizadas bem como técnicas que foram apenas simuladas em ambiente virtual. A partir dos testes pudemos chegar a algumas conclusões que serão relatadas no trabalho, bem como todos os problemas encontrados no desenvolvimento da plataforma de testes.

Descritores: Aprendizado profundo. Redes neurais. Aprendizado de máquina. Planejamento de caminho.

ABSTRACT

This study aims to verify the possibility of utilizing advanced artificial intelligence techniques as an option to the already developed algorithms used for path-planning, based on the classic logistical problem known as the "Travelling Salesman Problem" (TSP) a problem that tries to determine the shortest route needed to travel through a series of cities (visiting each one only once). Two neural networks with different architectures were created, and their results were compared to techniques already in use, as well as, techniques which were only simulated in a virtual environment. From those tests, it was possible to reach some conclusions, which are described within this paper, as well as all the problems faced during the development of the testing environment.

Keywords: Deep learning. Neural Networks. Machine Learning. Path-planning.

---

[1]Aluno, autor; Graduando do curso de Engenharia de Produção – UFSM/CT

[2]Professor, orientador; Doutor em Engenharia Elétrica – UFSM/CT

DEDICATÓRIA

A minha família, amigos, colegas e todos aqueles que contribuíram de forma ativa ou passiva para a realização deste trabalho, o qual não teria sido possível a realização se o suporte incondicional que me foi dado durante o período de pesquisa.

AGRADECIMENTOS

A concretização deste trabalho ocorreu, principalmente, pelo auxílio, compreensão e dedicação de várias pessoas. Agradeço a todos que de alguma forma, contribuíram para a conclusão deste estudo e, de uma maneira especial, agradeço:

- a meu orientador Felipe Martins Müller pela oportunidade da realização deste trabalho, bem como seus conselhos e correções que foram de suma importância para a qualidade do projeto;
- ao prof. Dr. Rodrigo da Silva Guerra que foi uma grande inspiração em minha jornada de estudo e também por ter sido quem sugeriu originalmente o tema abordado;
- ao amigo Moisés que me ajudou e aconselhou durante todo o percurso da pesquisa;
- ao amigo Guilherme Henrique Galleli Christmann que me ajudou e ensinou diversas técnicas utilizadas neste trabalho;
- a minha família por todo o suporte que me foi dado;
- aos amigos cuja companhia permitiu a manutenção de minha saúde mental, e incríveis momentos de companheirismo que levarei para a vida toda;
- aos colegas da Equipe TauraBots pela ajuda durante o desenvolvimento deste trabalho;
- à Universidade Pública pela oferta de um ensino de qualidade e a infraestrutura que permitiu a agregação de conhecimentos necessários para a conclusão desta pesquisa;
- aos professores e funcionários do Curso de Engenharia de Produção pela atenção e qualidade dos serviços e aulas ministradas;
- ao meu gato Oliver cuja companhia e carinho ajudou a superar os momentos de crise enfrentados;

Enfim a todos aqueles que fazem parte da minha vida e que são essenciais para eu ser, a cada dia nessa longa jornada, um ser humano melhor.

“Tudo o que devemos decidir é o que
fazer com o tempo que nos é dado.”

(John R. R. Tolkien)

SUMÁRIO

1 - INTRODUÇÃO

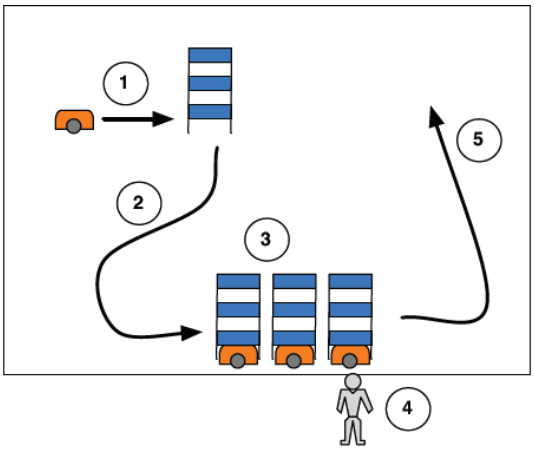
De tempos em tempos, surgem inovações tecnológicas tão impactantes que revolucionam certos aspectos da indústria. Nos últimos anos cada vez mais pesquisas comprovam que sistemas multiagentes e robótica têm se posicionado como uma dessas inovações na indústria de manejo de materiais. Enquanto AGVs (*Autonomous Guided Vehicles*) são utilizados para a locomoção interna de materiais desde a década de 50, eles são utilizados, principalmente, para a movimentação de cargas muito grandes e/ou muito pesadas, como, por exemplo, blocos de motores e rolos de papel não cortados. Com o advento de tecnologias baratas que oferecem, comunicação *Wireless*, poder computacional e componentes robóticos fazem com que veículos autônomos se tornem baratos, menores e cada vez mais capazes (WURMAN, D'ANDREA, MOUNTZ, 2008).

No ano de 2008 a empresa Kiva Systems anunciou a disponibilidade de um novo sistema para manejo de materiais voltados para o mercado onde o estoque é do estilo pegar-empacotar-enviar. A maior inovação deste sistema sendo a utilização de robôs pequenos, ágeis e baratos capazes de levantar unidades de armazenamento, de em torno de 1m², chamadas de *pods* de armazenamento. Os robôs, chamados de unidades de condução, transportam os *pods* de seus locais de armazenamento até estações, onde um funcionário pode pegar o item referente a um pedido e empacotá-lo para o transporte.

De acordo com Enright e Wurman (2011), os passos realizados por um dos robôs durante um ciclo de sua tarefa são (veja Figura 1):

1. Dirigir de seu local atual até os *pods* requisitados.
2. Carregar os *pods* até a fila da estação de coleta.
3. Esperar na fila até seu turno.
4. Deixar o operador pegar o(s) item(s) necessário(s) dos *pods.*
5. Retornar os *pods* ao espaço de armazenamento.

Figura 1 - Representação dos passos realizados pelo robô durante uma entrega



Fonte: ENRIGHT, WURMAN (2011)

Normalmente, uma instalação Kiva é arranjada com as zonas de estocagem no meio e as estações de inventário espalhadas ao redor do perímetro. Conforme comentado anteriormente, as unidades de condução são usadas para mover os *pods* de armazenagem com as cestas corretas de seu local de armazenagem até a estação de inventário onde um operador recolhe e empacota os produtos necessários. Vale ressaltar que cada *pod* possui 4 faces de armazenagem, então é possível que a unidade de condução deva rotacionar para permitir que o servidor acesse a face correta. Quando o operador acabar de recolher os itens de um *pod*, a unidade deve então retorná-lo até um espaço livre na zona de armazenamento. Na figura 2 é possível ver a relação entre um operador, *pod* e unidade de condução.

Figura 2 - Robô e prateleiras utilizadas pela Amazon em seus armazéns automatizados

Fonte: WURMAN, D'ANDREA, MOUNTZ (2008)

O problema representado nesta pesquisa é um problema clássico de logística conhecido como “Problema do Caixeiro Viajante” (PCV) um problema que tenta determinar a menor rota para percorrer uma série de cidades (visitando uma única vez cada uma delas), retornando à cidade de origem. O PCV é um problema de otimização NP-difícil inspirado na necessidade dos vendedores em realizar entregas em diversos locais (as cidades) percorrendo o menor caminho possível, reduzindo o tempo necessário para a viagem e os possíveis custos com transporte e combustível. No caso estudado, o PCV é representado pela melhor rota de coleta e armazenagem dos *pods* e o controle da bateria das unidades autônomas.

Conforme mostrado por Wurman, D'andrea e Mountz (2008), o sistema Kiva é altamente influenciado por técnicas de IA, apesar de que muitas das técnicas são implementações de algoritmos altamente conhecidos, oriundos de livros didáticos. A arquitetura do *Software* reflete o fato de que o sistema é, por sua natureza, um sistema multi-agente. Cada unidade de condução e estação são dispositivos computacionais que podem receber pedidos e agir a partir deles. Ao mesmo tempo o sistema, encarna um massivo, em tempo-real, problema de alocação de recursos. Os recursos em questão são o espaço de armazenagem na estação, as unidades de condução, os *pods*, o próprio inventário e o espaço físico em si.

Diversos estudos da área focam em possíveis aperfeiçoamentos de sistemas de estoque, de diversas maneiras, seja modificando o leiaute atual, implementando novos algoritmos, entre outros. Com cada unidade de condução e estação de trabalho, sendo um agente independente, com algum nível de autonomia. Existem também outros agentes que tomam as decisões de alocação que são críticos para a funcionalidade do todo. Estes agentes estão distribuídos através de diversos servidores e nos próprios computadores de cada estação de trabalho.

Como explicado anteriormente os atuais algoritmos aplicados nesse sistema são criados e estabelecidos como boas soluções por seres humanos, mas existem limites para o nível de complexidade de um algoritmo criado por um programador. Por isso propõem-se a aplicação de técnicas de aprendizado profundo para aperfeiçoar este sistema, em específico aprendizado-Q profundo. A técnica conhecida como aprendizado-Q permite que o agente aprenda de maneira similar ao ser humano, com recompensas quando realiza uma tarefa corretamente, e punições caso contrário. E quando se une esta técnica a uma rede neural de aprendizado profundo, o resultado é um sistema robusto capaz de agir de maneiras tão complexas que um programador jamais seria capaz de criar tal comportamento.

Como objetivos do trabalho, é tido como objetivo específico a verificação e análise da utilização de redes neurais para a solução do PCV apresentado, e como objetivos gerais a proposição de possíveis aprofundamentos e continuação dos estudos retratados.

2- REFERÊNCIAL TEÓRICO

A partir dos resultados desta aplicação será possível realizar uma análise comparando os resultados obtidos com aqueles colocados em prática no mercado. E a partir dessa análise, realizar uma proposta de estudos mais aprofundados na área, como a possibilidade de criação de um sistema de controle total de armazém onde uma RNA (Rede Neural Artificial), capaz de controlar as variáveis envolvidas neste setor, desde as ordens de compras, o local físico de armazenagem de cada produto, a ordem de cumprimento de pedidos e os diversos outros parâmetros naturais deste tipo de sistema, com tempo mínimo de locomoção de produtos, criando um sistema cada vez mais ágil, diminuindo, assim, o tempo de espera do consumidor.

2.1 – APRENDIZADO DE MÁQUINA

Em 1959, Arthur Samuel definiu aprendizado de máquina como o "campo de estudo que dá aos computadores a habilidade de aprender sem serem explicitamente programados". O aprendizado automático explora o estudo e construção de algoritmos que podem aprender com os seus erros e fazer previsões a partir de dados. Estes algoritmos funcionam através da construção de um modelo a partir de entradas amostrais a fim de fazer previsões ou decisões guiadas pelos dados e não simplesmente seguir instruções programadas. Enquanto que na inteligência artificial existem dois principais tipos de raciocínio (o indutivo, que extrai regras e padrões de conjuntos de dados, e o dedutivo), o aprendizado de máquina se preocupa somente com o raciocínio indutivo.

Mitchell (1997) forneceu uma definição formal amplamente citada: "Diz-se que um programa de computador aprende pela experiência E, com respeito a algum tipo de tarefa T e performance P, se sua performance P nas tarefas em T, na forma medida por P, melhoram com a experiência E." Esta definição das tarefas envolvidas no aprendizado de máquina é dada de uma maneira fundamentalmente operacional, ao invés de cognitiva, isto é, seguindo a proposta de Alan Turing em seu artigo "Computadores e inteligência", em que a pergunta "As máquinas são capazes de pensar?" é substituída pela pergunta "As máquinas são capazes de fazer o que (nós como entidades pensantes) podemos fazer?"

As tarefas de aprendizado de máquina normalmente são classificadas em três categorias de acordo com a natureza do "sinal" ou "*feedback*" de aprendizado disponível para um sistema de aprendizado. Essas categorias são:

- Aprendizado supervisionado: São apresentados ao computador exemplos de entradas e saídas desejadas, o que é fornecido por um "professor". O objetivo é o aprendizado de uma regra generalizada que mapeia das entradas para as saídas.
- Aprendizado não supervisionado: Não são dadas etiquetas ao algoritmo de aprendizado, deixando-o sozinho para encontrar a estrutura nas entradas fornecidas. O aprendizado não supervisionado pode ser visto como um objetivo em si mesmo (descoberta novos padrões nos dados) ou como um meio para atingir um fim.
- Aprendizado por reforço: Um programa de computador interage com um ambiente dinâmico, em que o programa deve cumprir com um determinado objetivo (por exemplo, dirigir um veículo). É fornecido, ao programa, reforços positivos ou negativos, na medida em que é navegado o espaço do problema.

Outra categorização de tarefas de aprendizado de máquina surge quando é considerada a saída desejada:

- Classificação: as entradas são divididas em duas ou mais classes, e o algoritmo deve produzir um modelo que vincula as entradas não vistas a uma ou mais destas classes (classificação multi-etiquetada). Isso é tipicamente abordado através de aprendizado supervisionado.
- Regressão, também é considerado como um problema supervisionado, mas as suas saídas são contínuas, e não discretas.
- Clustering, conjuntos de entradas são divididos em grupos. De maneira diferente da classificação, os grupos não são conhecidos previamente, tornando o clustering uma tarefa de aprendizado não supervisionado.
- Estimativa de densidades, encontra a distribuição de entradas em um espaço.
- Redução dimensional, simplifica as entradas ao traduzí-las para um espaço de menor dimensão.

2.2 – APRENDIZADO PROFUNDO

Aprendizado profundo refere-se a uma classe de técnicas de aprendizado de máquina. Essa área pode ser considerada como a intersecção entre diversas áreas da TI (Tecnologia da

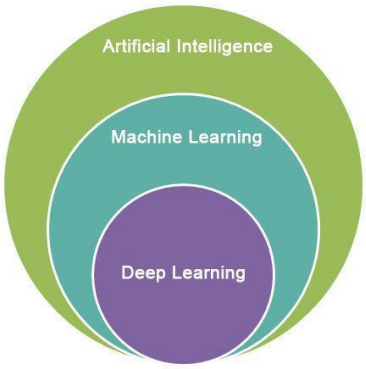
Informação), sendo elas as áreas de otimização, reconhecimento de padrões, modelagem gráfica, redes neurais e processamento de sinais. UM dos importantes motivos para a popularidade do aprendizado profundo atualmente é o aumento drástico de processamento nos computadores com a utilização de placas de vídeo (*Graphical Processing Units* - GPUs), um custo menor de *hardware* e os recentes avanços na pesquisa de aprendizado de máquina e processamento de informações (DENG, L. et al., 2013).

Conforme Pereira (2017) esse drástico aumento na velocidade do treinamento com o uso de GPUs se deve ao fato do treinamento de redes neurais profundas usarem diversas operações matemáticas em grandes matrizes para o treinamento da rede, o que leva bastante tempo para se computar em um processador comum, mas que são computadas rapidamente por GPUs.

Aprendizado profundo é um nome de fácil compreensão quando comparado com Rede Neural Artificial, que acarretou em sua recente popularidade. O termo “profundo” se refere à profundidade da rede, isto é, a sua quantidade de camadas. Uma rede neural artificial normalmente possui apenas uma ou duas camadas escondidas, enquanto uma rede profunda pode contar com um número muito maior de camadas, dependendo de sua arquitetura e aplicação, existindo um consenso que para que uma rede possa ser chamada de Profunda são necessárias, no mínimo, cinco camadas escondidas.

A Figura 3 apresenta o Aprendizado Profundo com relação às áreas de Inteligência Artificial e Aprendizado de Máquina. A área de Aprendizado Profundo está contida dentro da área de Aprendizado de Máquina, e esta por sua vez dentro da área de Inteligência Artificial.

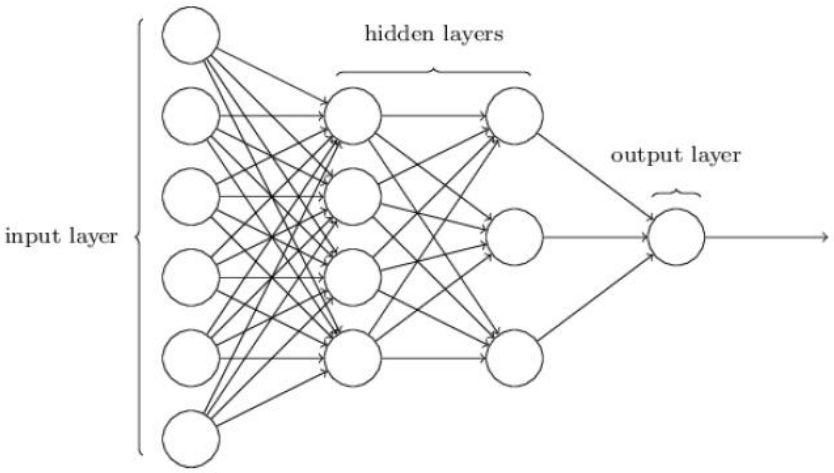
Figura 3: O aprendizado profundo (*deep learning)* inserido na área de aprendizado de máquina (*machine learning)* e na área geral de inteligência artificial (*artificial intelligence)*.



Fonte: https://www.codeproject.com.

A arquitetura de uma rede neural artificial (RNA) possui quatro componentes, as camadas, os nós, a função de ativação e a função de otimização. Conforme citado anteriormente, uma RNA pode ser criada com um número qualquer de camadas, a primeira camada de uma rede neural é conhecida como a camada de entrada. Cada uma das camadas da RNA é criada com um número de nós, este número é definido pelo programador, pois não existem regras para o número de nós e camadas que uma RNA deve possuir a fim de realizar uma dada tarefa. Na camada de entrada cada nó recebe um valor de entrada, e então passa sua saída como a entrada para cada nó na camada seguinte. Não existem conexões entre nós da mesma camada, e a última camada produz a saída. A Figura 4 apresenta a arquitetura de uma RNA com 2 camadas escondidas.

Figura 4: Arquitetura de uma rede neural com duas camadas escondidas (*hidden layers)*, camada de entrada (*input layer)* e camada de saída *(output layer)*.



Fonte: NIELSEN, 2015.

Por meio de um conjunto de técnicas de aprendizado profundo foi possível superar o limite no número de camadas de uma rede, possibilitando a resolução do *vanishing gradient problem*(Problema do Gradiente de Desaparecimento) e treinar redes com mais camadas escondidas. Algumas das técnicas que contribuíram para esse avanço são abordadas a seguir.

2.2.1 – RELU

A ReLU (*Rectified Linear Unit)* é uma função de ativação não linear criada por Hinton e Nair (2010) que segue a seguinte equação:

$$f(x) = Max\ (0, x)$$

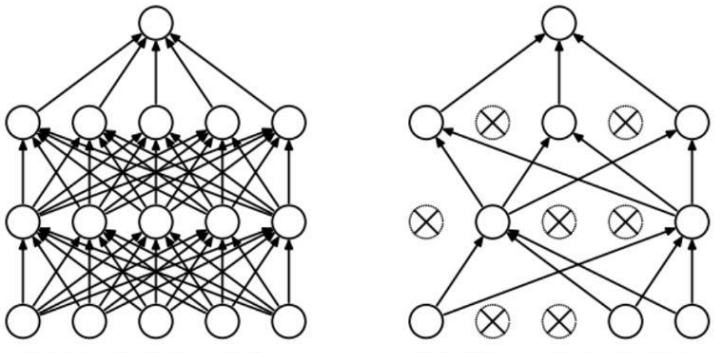
onde Max (0,x) é uma função que retorna 0 caso x < 0, e x caso x > 0.

Essa função é muito utilizada, pois é calculada rapidamente, quando comparada à outras funções de ativação não lineares mais convencionais como a função sigmóide, representada pela equação: $\frac{1}{1+e^{-x}}$ que retorna um valor entre 0 e 1, ou a função tangente hiperbólica, representada por $tanh\ (x) = 2\sigma(2x) - 1$ e que retorna um valor de -1 a 1. Pelo fato de ser tão simples de se computar uma ReLU, o tempo de treinamento é reduzido, o que facilita o treinamento de redes com várias camadas.

2.2.2 – DROPOUT

Conforme Srivastava et al. (2014) o *Dropout* é uma técnica usada para aumentar a velocidade de treinamento de redes com várias camadas utilizando um método para aleatoriamente excluir neurônios e suas conexões da rede neural durante o treinamento. Essa técnica previne um fenômeno conhecido como *overfitting*, onde a rede neural treinada aprende a resolver o problema apresentado no set de dados utilizado muito bem, mas não consegue generalizar esse aprendizado para outros problemas. A figura 5 contém um exemplo de como funciona o *Dropout*.

Figura 5: a) Uma rede neural comum com duas camadas escondidas. b) uma rede neural com duas camadas escondidas após o *dropout*.



Fonte: SRIVASTAVA et al, 2014.

Com uma rede mais profunda é possível aprender e representar modelos mais complexos. Neste trabalho será abordado um tipo de rede profunda: a rede *Long Short-Term Memory* (LSTM), que corresponde a um tipo de uma rede neural recorrente, e, como o nome

sugere, a rede possui uma “memória de curto prazo” dos dados que já passaram por ela, isto será explicado mais profundamente nas seções que seguem.

2.3 - APRENDIZADO–Q

O Aprendizado-Q (*Q-Learning*) é uma técnica de aprendizado de reforço usada em aprendizado de máquina. O objetivo do *Q-Learning* é aprender uma política, que diz a um agente qual ação tomar sob dadas circunstâncias. Não requer um modelo do ambiente e pode lidar com problemas com transições e recompensas estocásticas, sem exigir adaptações.

Para qualquer processo finito de decisão de Markov (FMDP), o *Q-learning* encontra uma política que é ótima, isto é, maximiza o valor esperado da recompensa total sobre todos os passos sucessivos, a partir do estado atual. O *Q-learning* possui a capacidade de identificar uma política ótima de seleção de ação para qualquer FMDP, dado o tempo infinito de exploração e uma política parcialmente aleatória. "Q" nomeia a função que retorna a recompensa usada para fornecer o reforço e pode ser considerada a "qualidade" de uma ação tomada em um determinado estado.

Queiróz (2018) classifica as seguintes variáveis importantes:

- **Exploração vs Aproveitamento:** A taxa de aprendizado ou o tamanho da etapa determina até que ponto as informações recém-adquiridas substituem as informações antigas. Um fator de 0 faz com que o agente não aprenda nada (explorando exclusivamente o conhecimento prévio), enquanto um fator de 1 faz com que o agente considere apenas as informações mais recentes (ignorando o conhecimento anterior para explorar as possibilidades). Em ambientes totalmente determinísticos, uma taxa de aprendizado 1 é ideal. Quando o problema é estocástico, o algoritmo converge sob condições técnicas na taxa de aprendizado que exigem que este diminua para zero. Na prática, na maioria das vezes é usada uma taxa de aprendizado constante.
- **Fator de Desconto:** O fator de desconto determina a importância de recompensas futuras. Um fator de 0 tornará o agente "míope" (ou míope) considerando apenas as recompensas atuais, enquanto um fator que se aproxima de 1 fará com que se esforce por uma recompensa alta de longo prazo. Se o fator de desconto atender ou exceder 1, os valores da ação podem divergir, sem um estado terminal, ou se o agente nunca atingir um, todos os históricos de ambiente tornam-se infinitamente longos e utilitários com recompensas aditivas não-recontadas geralmente se tornam infinitos. Mesmo com um fator de desconto apenas um pouco menor que 1, o aprendizado da função Q leva à

propagação de erros e instabilidades quando a função valor é aproximada com uma rede neural artificial. Nesse caso, começar com um menor fator de desconto e aumentá-lo em direção ao seu valor final acarreta na aceleração do aprendizado.

- **Condições iniciais:** Como o Q-learning é um algoritmo iterativo, ele assume implicitamente uma condição inicial antes que a primeira atualização ocorra. Altos valores iniciais, também conhecidos como "condições iniciais otimistas", podem estimular a exploração: não importa qual ação seja selecionada, a regra de atualização fará com que ela tenha valores menores que a outra alternativa, aumentando assim sua probabilidade de escolha. A primeira recompensa r pode ser usada para redefinir as condições iniciais. De acordo com essa ideia, a primeira vez que uma ação é tomada, a recompensa é usada para definir o valor de Q. Isso permite o aprendizado imediato em caso de recompensas determinísticas fixas. Espera-se que um modelo que incorpore o reset das condições iniciais (RIC) preveja melhor o comportamento dos participantes do que um modelo que assume qualquer condição inicial arbitrária (AIC).
- **Quantização:** Considere o exemplo de aprender a equilibrar um bastão em um dedo. Descrever um estado em um determinado momento envolve a posição do dedo no espaço, sua velocidade, o ângulo do bastão e a velocidade angular do bastão. Isso gera um vetor de quatro valores que descreve um estado, ou seja, um instante de um estado codificado em quatro elementos. O problema é que estão presentes infinitos estados possíveis. Para reduzir o espaço possível de ações válidas, vários valores podem ser atribuídos a um intervalo.

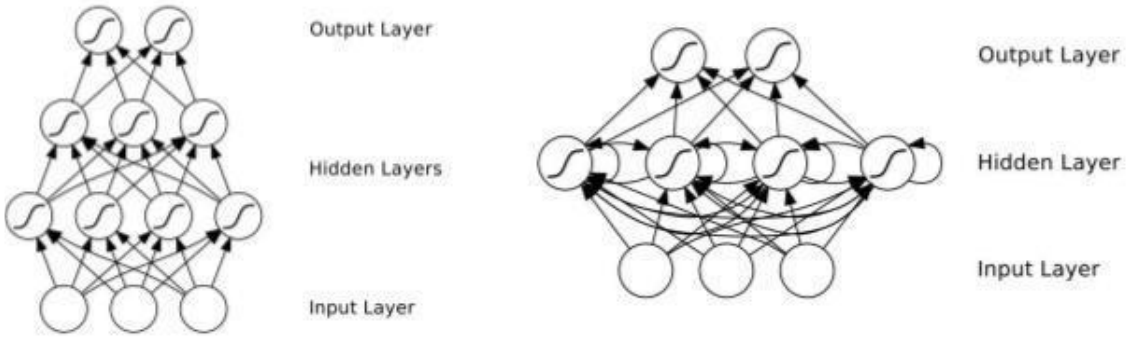
2.4 – REDES NEURAIS RECORRENTES

Redes Neurais Recorrentes (*Recurrent Neural Networks* - RNNs, em inglês) é uma classe de redes neurais artificiais onde as conexões entre as unidades formam um ciclo direcionado. São modelos de redes neurais bastante populares e que vem mostrando bastante potencial em diversas áreas de aprendizado de máquina como processamento de linguagem natural, legenda automática de imagens, tradução e reconhecimento de voz (BRITZ, 2015).

De acordo com Pereira (2017) a principal ideia por trás das RNNs utilizar informações sequenciais. Em uma rede neural tradicional é assumido que todas as entradas e saídas são independentes uma das outras, mas nem sempre esta é a melhor abordagem. Por exemplo, para fazer a predição da próxima palavra de uma frase é necessário saber as palavras que antecedem a palavra gerada. As RNNs são ditas recorrentes, pois realizam a mesma tarefa

para cada elemento da sentença, com a saída sendo dependente dos resultados anteriores, a Figura 6 ilustra uma comparação entre uma rede *feedforward* e uma rede recorrente.

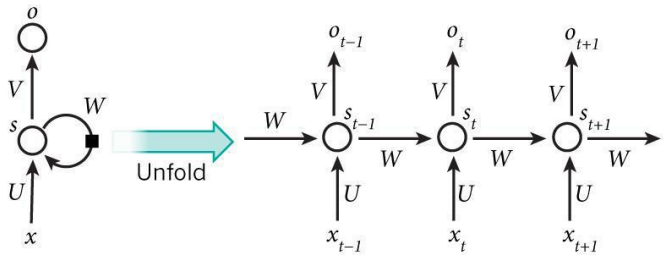
Figura 6: Comparação da arquitetura de uma rede *feedforward* (esquerda) e uma rede recorrente (direita). As camadas escondidas também trocam informações na mesma camada, diferentemente de uma rede *feedforward.*



Fonte: Sapunov, G. (2016) Disponível em: <https://www.slideshare.net/grigorysapunov/multidimensional-rnn>.

Uma maneira simplista de se pensar sobre as RNNs é que elas possuem uma “memória” que captura informações sobre o que foi calculado até o momento.

Figura 7: Uma rede neural recorrente e o processo do desdobramento (*unfold*) de seu tempo de computação.



Fonte: BRITZ, 2015

O diagrama na Figura 7 mostra uma RNN sendo desdobrada em uma rede completa. Esse processo de desdobramento representa somente a sequência de toda a rede. Por exemplo, se uma sequência é uma sentença de cinco palavras, a rede desdobraria em uma rede neural com cinco camadas, com uma camada para cada palavra. A descrição dos símbolos da figura 7 é como seguem:

- X_t é a entrada em um passo de tempo *t*;
- S_t é um estado escondido em um passo de tempo *t*, que é calculado baseado no estado escondido anterior e na entrada no passo atual: $s_t = f(U_x + W_{St-1})$;
- S_{t-1} é usado para calcular o primeiro estado escondido, e é geralmente iniciado com 0;
- O_t é a saída no passo t.

A parte mais importante do diagrama da Figura 7 é o estado escondido St, que corresponde à memória da rede, e guarda informações sobre o que aconteceu nos passos anteriores.

O treinamento de uma RNN é similar ao treinamento de uma rede neural comum, ainda utilizando o algoritmo de *backpropagation*, mas com uma alteração para possibilitar o cálculo do efeito de tempo/sequência na rede neural, fazendo com que esse novo algoritmo usado seja conhecido como *backpropagation through time* (BPTT).

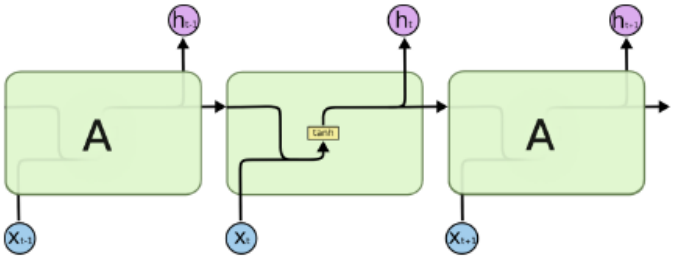
Um problema comum em RNNs é são as "dependências de longo prazo". Dependendo da tarefa que a RNN tenta executar, a RNN precisa buscar informações recentes na rede para a realização dessa tarefa. Porém, caso a informação que a rede necessita não seja tão recente, se torna difícil que a RNN aprenda a informação correta. Quanto mais "antiga" for essa dependência, mais difícil será a RNN aprender baseando-se nessa informação. Uma abordagem que tenta resolver esse problema das dependências de longo pra são as chamadas redes *Long Short Term Memory* (Memória de Curto Prazo Longa), popularmente conhecidas como LSTMs.

2.5 – LSTM

Redes LSTMs são um tipo especial de RNN, capazes de aprender as dependências de longo prazo. Introduzidas inicialmente por Hochreiter e Schmidhuber (1997), funcionam muito bem em uma grande variedade de problemas, sendo amplamente utilizadas atualmente.

Todas as RNNs têm a arquitetura das camadas na forma de uma cadeia de módulos que se repetem. Em uma RNN padrão, esse módulo possui uma estrutura simples, como por exemplo, uma camada com a função *tanh* conforme a Figura 8.

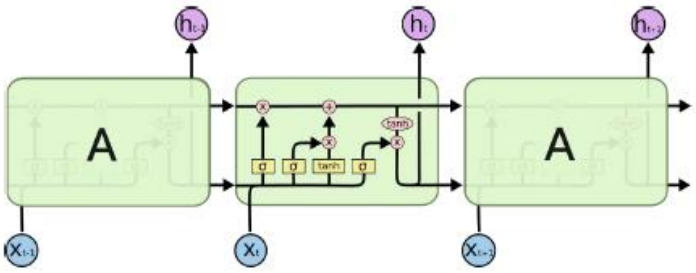
Figura 8 – O módulo que se repete em uma RNN contém apenas uma única camada



Fonte: OLAH (2015)

Redes LSTM também possuem esta estrutura de cadeia, porém o módulo que se repete é estruturado de uma forma diferente. Ao invés de ter apenas uma única camada, existem quatro, que interagem de maneira específica. A Figura 9 apresenta este conceito.

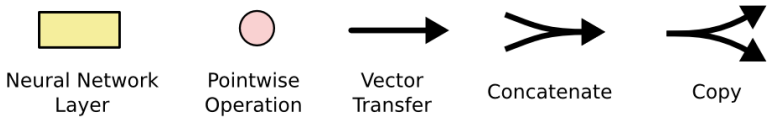
Figura 9 – O módulo que repete em uma LSTM contém quatro camadas que se interagem.



Fonte: OLAH (2015)

A Figura 10 apresenta a notação utilizada na imagem:

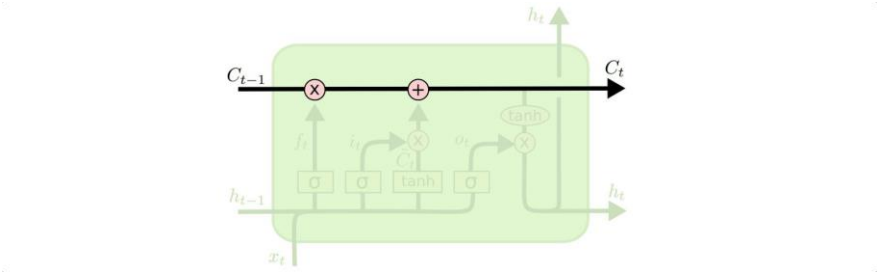
Figura 10 - Notação utilizada no exemplo. Lê-se da esquerda para direita: Camada da rede neural, operação ponto a ponto, transferência de vetor, concatenar, copiar.



Fonte: OLAH (2015)

A principal ideia das LSTMs é a criação de uma representação do estado da célula, que corresponde à linha horizontal na parte superior da representação demonstrada na Figura 11. Esse estado da corre por toda a cadeia, sofrendo interações lineares, o que possibilita que a informação possa fluir sem muitas alterações.

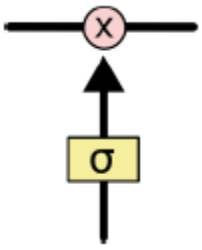
Figura 11 - O estado da célula, representado por uma linha horizontal no topo do diagrama.



Fonte: OLAH (2015)

As redes LSTM também possuem a capacidade de remover ou adicionar informação para o estado da célula, sendo regulada por Portões (*gates*). Os *gates* são uma maneira de pemitir que a informação flua na rede neural. Eles são compostos por uma camada sigmóide e uma multiplicação *pointwise*. A Figura 12 ilustra este conceito.

Figura 12 - Exemplo de um *gate* com uma camada sigmóide de uma rede neural e uma operação de multiplicação *pointwise*.



Fonte: OLAH (2015)

A camada sigmóide tem como saída números que variam entre 0 e 1, descrevendo o quanto de cada componente deve ser permitido passar pelo *gate*. Quanto maior o valor, maior é a informação passada por essa camada. Uma rede LSTM possui três *gates* a fim de proteger e controlar o estado da célula.

3 – METODOLOGIA

3.1 CENÁRIO

Durante o período de realização desta pesquisa, o mundo enfrenta uma situação extrema, em frente à pandemia que assola a população mundial toda cautela se faz necessária, por esse motivo a pesquisa em seu início será realizada no âmbito domiciliar do autor, utilizando como equipamento principal seu computador pessoal. Conforme a situação global melhorar e as condições permitirem, a pesquisa passará a ser realizada no laboratório multidisciplinar PRO+E, ainda utilizando o computador pessoal do autor como ferramenta principal, mas contando com o suporte da placa Nvidia Jetson TX2 para o treinamento da rede neural.

Para a execução dos testes, a unidade autônoma, deverá realizar as mesmas tarefas que são desempenhadas no sistema físico, isto é, ela deverá se deslocar de seu local atual, até o *pod*, a ser recuperado, levar este até o ponto de entrega, e então retorná-lo à um local de armazenagem livre.

3.2 MÉTODOS DE PESQUISA

A pesquisa é uma pesquisa aplicada de natureza exploratória com abordagem quantitativa utilizando procedimentos de modelagem e simulações. Esta pesquisa possui diversas similaridades com a pesquisa realizada por Jiang e Wang em 2014, a diferença sendo o sujeito de testes, que em sua pesquisa foi o algoritmo de busca da mosca da fruta e algumas de suas modificações.

3.3 ETAPAS DA PESQUISA

A pesquisa seguiu a seguinte estrutura, em sua fase inicial foi realizado um estudo de diferentes publicações nacionais e internacionais, para encontrar tais pesquisas foi utilizado a plataforma nacional Sucupira e outras bases de artigos científicos, assim como buscas pelo Google, as pesquisas são focadas nos principais temas desta pesquisa, que são o desenvolvimento de IAs para unidades autônomas, com foco principal nas aplicações em armazéns, e o desenvolvimento de RNAs como alternativas para algoritmos de menor caminho assim como plataformas de testes para estas. Nas etapas finais da realização deste

estudo, foi realizado o desenvolvimento das Redes Neurais, utilizando a técnica de aprendizado-Q profundo e realizando a programação na linguagem Python. A escolha de Python como linguagem de desenvolvimento dá-se pela ampla variedade em bibliotecas que oferecem suporte para a criação de redes robustas, como as bibliotecas PyTorch (PASZKE, 2019) e TensorFlow (ABADI, 2015), ambas sendo de suma importância para o desenvolvimento desta pesquisa.

Depois de desenvolvida as inteligências artificiais, iniciou-se o desenvolvimento da plataforma de teste, que foi desenvolvida com base na plataforma apresentada por Hazard, Wurman e D'Andrea (2006), esta plataforma também foi desenvolvida na linguagem Python para evitar problemas de compatibilidade entre linguagens. A plataforma desenvolvida seguiu a mesma lógica da apresentada pelos autores supracitados, diferenciando-se em alguns aspectos, onde os autores utilizaram um algoritmo para atribuir uma letra de uma palavra escolhida aleatoriamente, o programa desenvolvido neste trabalho simulou uma situação mais próxima da realidade, utilizando diferentes tipos de produtos que deverão ser entregues em diferentes locais, com as ordens e conteúdos de pedidos aleatorizados.

Concluído o desenvolvimento da plataforma de testes, iniciou-se o período de realização dos testes, durante este período as redes foram colocadas para realizar seus treinamentos e testes utilizando a plataforma construída, tomando como medidas de desempenho a distância percorrida por cada unidade autônoma e o tempo de entrega.

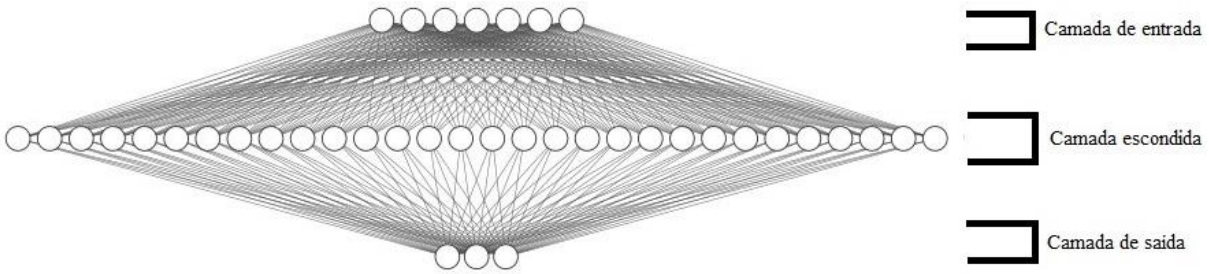
Depois de realizados os testes, os resultados a partir destes foram analisados e comparados com aqueles obtidos por outros autores, encontrados na etapa inicial desta pesquisa. E durante todo o período de desenvolvimento da pesquisa, foi desenvolvido paralelamente o texto da mesma. Nas próximas seções são apresentados os resultados obtidos após a realização dos testes, os problemas enfrentados, os resultados selecionados para fins comparativos, bem como as razões destas escolhas.

4 – REALIZAÇÃO DA PESQUISA

Durante a primeira etapa da realização desta pesquisa realizou-se uma pesquisa de diversos artigos sobre o assunto a fim de agregar conhecimento e possíveis resultados para fins comparativos. Foram então desenvolvidas duas LSTMs que utilizam técnicas de *Q-Learning*, na Figura 14 e 15 é apresentada a representação gráfica da arquitetura das redes. Após a criação das redes desenvolveu-se o ambiente de testes.

Para a criação das redes foram utilizadas duas bibliotecas Python, a biblioteca PyTorch, que foi utilizada para a construção da arquitetura da rede, sendo definidas em código, o número de camadas e neurônios de cada camada, o tipo de rede escolhida, e o tamanho da memória desta rede. Para o ambiente de testes foi, utilizada a biblioteca Kivy, sendo ela utilizada para a criação da tela de simulação, bem como os objetos nela presentes. Diferentes redes, com diferentes números de camadas foram testadas, dentre elas foram utilizadas para os testes aquelas que se mostraram promissoras e que atendem à definição de redes profundas anteriormente estabelecida neste trabalho.

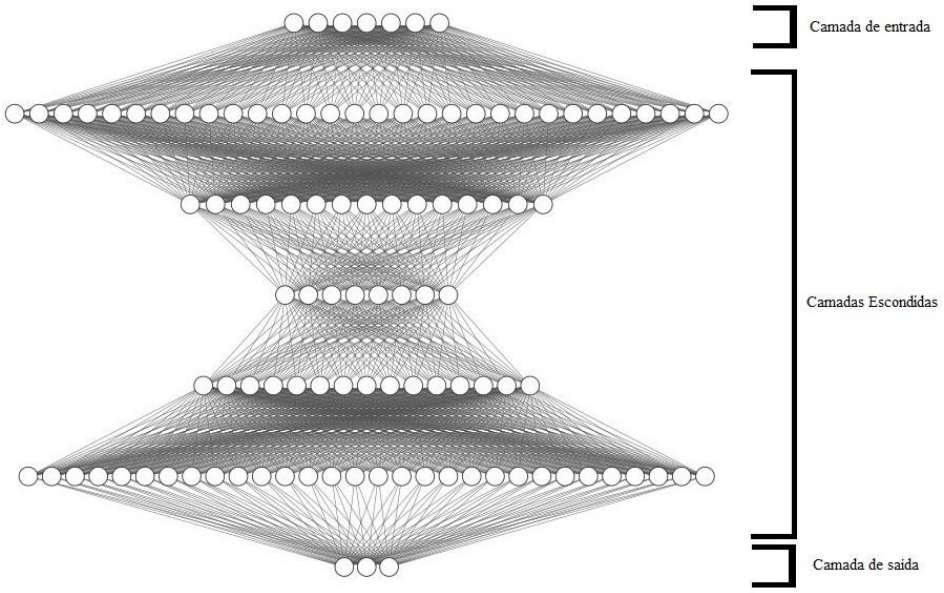
Figura 14 – Arquitetura da LSTM com apenas 1 camada escondida



Fonte: Autor

A rede que contém apenas uma camada escondida possui 7 neurônios de entrada, 30 neurônios na camada escondida e 3 neurônios de saída. A rede que possui 5 camadas escondidas possui os mesmos números de neurônios de entrada e saída que a anterior, mas possui 30 neurônios na primeira e última camada escondida, 15 na segunda e quarta camada e apenas 8 na terceira camada. A diferença do número de neurônios das camadas é uma técnica utilizada para evitar que a rede deixe de realizar os cálculos entre os neurônios, garantindo que a rede inteira será utilizada no treinamento.

Figura 15 – Arquitetura da LSTM com 5 camadas escondidas



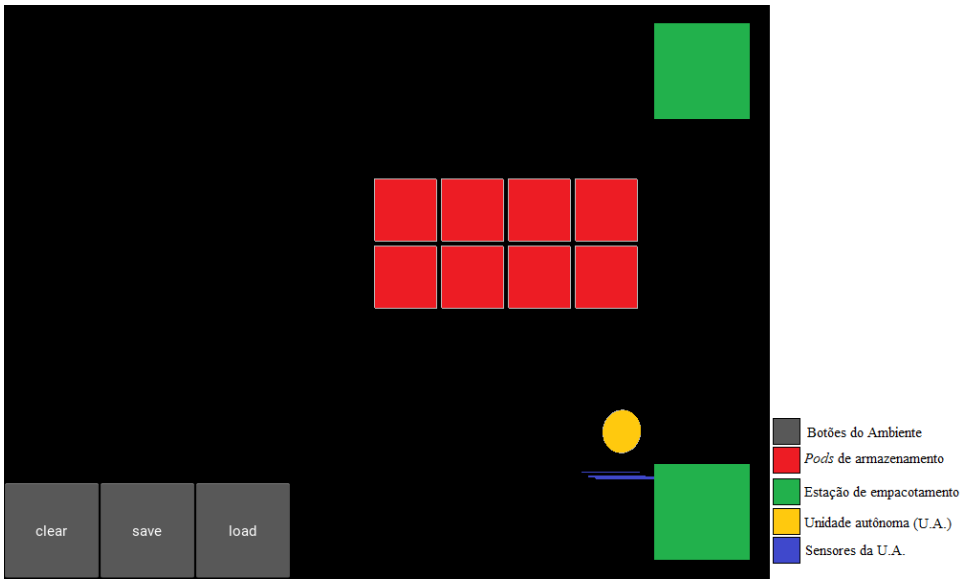
Fonte: Autor

Após o desenvolvimento das RNs foi desenvolvida a plataforma de testes que foi utilizada para o treinamento das redes, na Figura 16 é mostrada a tela de representação gráfica do treinamento. Durante o desenvolvimento desta plataforma de testes diversos problemas ocorreram e por causa disso a visão inicial do autor não pode ser alcançada, a versão pretendida possuía alguns aspectos que não foram possíveis de programar, devido a problemas de compatibilidade nas bibliotecas utilizadas no programa, no tempo disponível para a realização da pesquisa. Os aspectos faltantes no ambiente de testes final foram a colisão dos *pods* de armazenagem assim como a colisão dos suportes dos *pods* com a unidade autônoma.

Os problemas possuíram diversas causas, para a colisão dos pods entre si, bem como a colisão dos suportes dos *pods* com a unidade autônoma, a principal causa foi devido às incompatibilidades inesperadas entre as bibliotecas da linguagem utilizadas, diversas soluções foram implementadas para tentar solucionar os problemas, mas nenhuma surtiu efeito de maneira satisfatória, isto acabou por causar um grande atraso nos testes, e com o intuito de evitar uma extensão do tempo alocado, optou-se por não utilizar estes aspectos nos testes finais.

Também era pretendido utilizar uma única rede para o controle de múltiplas unidades, mas isto se mostrou, novamente, uma tarefa grande demais para o tempo disponível, uma possível solução seria a utilização de outras redes para a realização do controle das outras unidades, mas infelizmente não foi possível obter o acesso às ferramentas e equipamentos necessários para o treinamento de múltiplas RNAs de uma forma paralela, portanto o produto final conta com apenas uma unidade autônoma.

Figura 16 – Tela gráfica do programa de treinamento



Fonte: Autor

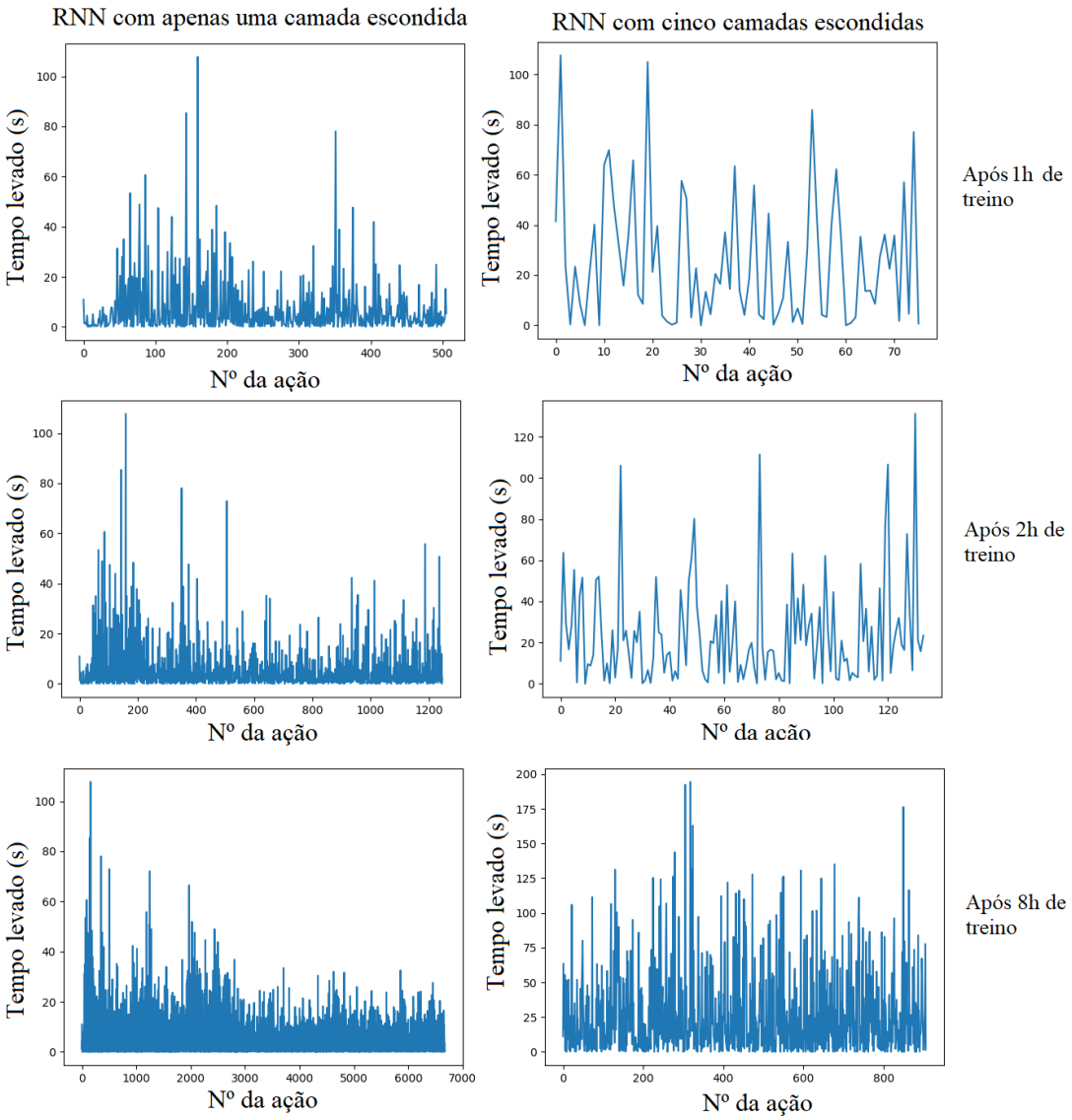
As informações retornadas para a rede pelo ambiente de teste são, a distância até o objetivo atual, a densidade de “areia” medida pelos 5 sensores, as possíveis ações da U.A. e as recompensas de cada ação tomada. Originalmente os sensores seriam utilizados para detectar colisões, mas como explicado anteriormente, não foi possível programar este aspecto, portanto os sensores detectam a densidade de “areia”, que é utilizado para retornar uma recompensa negativa à rede a fim de evitar que a rede deixe a U.A. parada.

4.1 – RESULTADOS OBTIDOS

Os resultados obtidos nesta aplicação de redes neurais em um PCV estático são apresentados a seguir, vale ressaltar novamente, que estes resultados foram comprometidos devido aos problemas enfrentados durante o decorrer do desenvolvimento da pesquisa.

Durante a fase de testes, ambas as redes foram treinadas por um total de 8 horas, a RNA com cinco camadas escondidas obteve um tempo médio para a execução das ações de 26.4 segundos após as 8 horas de treino e 23.5 segundos após apenas 2 horas de treinamento. A rede com apenas obteve um tempo médio para a execução das ações de 4.32 segundos após as 8 horas de treino e 5.72 segundos após apenas 2 horas de treinamento. Na figura 17, são apresentados os quadros de tempo x ação, para 1, 2 e 8 horas de treinamento para ambas as redes.

Figura 17 – Quadros de tempos necessários para completar cada ação, após 1, 2 e 8 horas de treino.



Fonte: Autor

A partir destes números, vemos que, a rede com cinco camadas escondidas atingiu após o treino uma média de tempo de 79.2 segundos para completar todo o percurso. Já a rede com apenas uma camada escondida atingiu, após o treino, uma média de tempo de 12.96 segundos.

A partir destes resultados, observa-se que, conforme citado em seções anteriores, uma RNA com mais camadas e complexa, não significa que ela cumprirá a tarefa melhor que uma mais simples. Estes resultados também demonstram, que apesar dos problemas enfrentados durante a realização da pesquisa, bem como a falta de aspectos no ambiente de testes, uma rede neural, quando desenvolvida para a tarefa específica de *path-planning*, consegue realizar as tarefas numa velocidade satisfatória.

4.2 – RESULTADOS OBTIDOS POR OUTROS AUTORES

Conforme citado anteriormente o objetivo desta pesquisa é de verificar a aplicação da técnica discutida e para este fim foi realizada uma comparação com os resultados previamente obtidos por outros pesquisadores, nesta seção são demonstrados e discutidos estes resultados. A fim de realizar uma comparação robusta foram selecionados diversos trabalhos, que utilizam variadas técnicas e algoritmos de programação para solucionar o PCV apresentado, dentre eles estão algoritmos clássicos como A*, e algoritmos mais novos como é o caso do algoritmo da mosca fruta e suas variações.

Foram pesquisados diversos trabalhos, e alguns deles utilizavam o algoritmo A* para a realização do planejamento de caminho, dentre estes trabalhos aquele que mais se destaca e que será utilizado para comparação é o artigo “Multi-agent Pathfinding Based on Improved Cooperative A* in Kiva System” de LIU e CHEN(2019), seu destaque é dado pela utilização do A* de maneira direta em uma simulação do sistema kiva e também com a proposta de uma versão otimizada do algoritmo CA*(*Cooperative* A*) nomeado como ICA*(*Improved Cooperative* A*). Os autores obtiveram diversos resultados, utilizando desde apenas 1 unidade autônoma até 20, obtendo um tempo médio de entrega, utilizando apenas uma unidade, de 165 segundos para o A* e 105 segundos para a versão própria dos autores, já com 20 unidades o tempo médio para o A* foi novamente de 165 segundos e de 120 segundos para o ICA*. Vale ressaltar que dentre os resultados obtidos pelos autores, os melhores para ambos os algoritmos foram de 120 segundos para o A* e 100 segundos para o ICA*, utilizando um sistema com apenas duas unidades autônomas.

Outro algoritmo que será utilizado para as comparações é o algoritmo da mosca fruta e suas variações, dentre os artigos estudados, o que recebe o destaque para este trabalho e será utilizado para fins comparativos é o "Modified fruit fly optimization algorithm of logistics storage selection" de PAN et. Al. (2017), seu destaque se dá ao fato de que foi o único artigo encontrago que relata sobre a aplicação não só do algoritmo da mosca da fruta (FOA), mas também das suas variações caótica (CFOA), adaptativa (AFOA) e melhorada (IFOA). Os resultados obtidos foram simulados utilizando ambiente virtual, e diversas unidades autônomas, ao total foram feitos 3 testes duas vezes para cada algoritmo, com 10, 20 e 30 unidades autônomas, obtendo os resultados que o algoritmo com o melhor desempenho foi o CFOA com 136 e 142.08 segundos para encontrar o caminho ótimo nos testes com apenas 10 unidades, 328.8 e 319.54 segundos para os testes com 20 unidades e 522.82 e 520.44 segundo nos testes com 30 unidades na Tabela 1 podem ser vistos as medias obtidas pelos autores na íntegra.

Tabela 1 – Menor tempo de seleção utilizado pelos quatro FOAs.

Algorithm		FOA	IFOA	AFOA	CFOA
10 freight sections in the first group	AVG	242.62 s	242.14 s	212.36 s	136 s
	STD	0.674 s	6.713 s	6.889 s	6.967 s
10 freight sections in the second group	AVG	252.6 s	252.34 s	212.24 s	142.08 s
	STD	0.56 s	4.942 s	5.044 s	4.441 s
20 freight sections in the first group	AVG	568.2 s	572.56 s	488.54 s	328.8 s
	STD	7.250 s	14.950 s	14.983 s	15.112 s
20 freight sections in the second group	AVG	560.54 s	542.4 s	479.92 s	319.54 s
	STD	6.968 s	18.030 s	15.399 s	16.032 s
30 freight sections in the first group	AVG	924.78 s	925.9 s	819.96 s	522.82 s
	STD	11.587 s	19.225 s	19.739 s	18.625 s
30 freight sections in the second group	AVG	913 s	915.92 s	812.44 s	520.44 s
	STD	12.849 s	26.125 s	21.592 s	24.440 s

Fonte: PAN et. Al. (2017)

Vale lembrar, que as aplicações destes algoritmos, tanto os apresentados nesta seção, quanto os desenvolvidos pelo autor, não são limitados, quanto às suas aplicações, apenas nesta versão do PCV, mas sim como possíveis soluções para o problema como um todo,

contanto que adaptados às peculiaridades de cada versão, demonstrando que são ferramentas de apoio fortes para Engenheiros de Produção na solução destes problemas.

5 - CONCLUSÃO

Após a conclusão dos testes, e utilizando os resultados obtidos, pode ser realizada uma verificação da efetividade de uma RNA para o planejamento do caminho, utilizando os tempos obtidos durante a pesquisa, percebe-se que o desempenho de ambas as redes, é comparável e até melhor que os dos algoritmos utilizados atualmente na indústria, mas deve-se lembrar de que existem pontos faltantes no ambiente de testes e uma grande diferença nos tamanhos dos armazéns simulados, portanto os resultados obtidos não são capazes de gerar um veredito final, mas servem para iluminar esta possibilidade, demonstrando a capacidade de obter um desempenho melhor do que se utiliza atualmente.

Ressalta-se uma última vez, que durante o desenvolvimento da plataforma de testes diversos problemas ocorreram e por causa disso a visão inicial do autor não pode ser alcançada, a versão pretendida possuía alguns aspectos que não foram possíveis de programar, devido a problemas de compatibilidade nas bibliotecas utilizadas no programa e do tempo disponível para a realização da pesquisa.

A partir deste estudo foi possível descobrir a possibilidade de aplicações de técnicas avançadas de inteligência artificial no cenário da Engenharia de Produção, portanto, propõe-se a continuação destes estudos para trabalhos futuros, implementando em uma plataforma com as mecânicas que faltaram neste estudo, e também a exploração de outros aspectos que podem ser utilizados em conjunto com as técnicas apresentadas, como o controle da distribuição de pedidos para as U.A., o controle dos níveis de estoque, e tantos outros aspectos que podem ser automatizados.

Portanto consideram-se como cumpridos parcialmente os objetivos orginalmente propostos neste trabalho, apesar de não ter sido possível o estabelecimento de um veredito final a respeito da eficácia de RNNs no problema apresentado, os resultados obtidos iluminam esta possibilidade em uma luz otimista, assim como a necessidade de estudos mais aprofundados em aplicações destas técnicas no cenário da Engenharia de Produção, o que permitiu a proposta de estudos futuros e mais aprofundados.

REFERÊNCIAS

ABADI, M. et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015. Disponível em: <https://www.tensorflow.org/>. Acesso em: 25 junho 2020

BRITZ, Denny. Recurrent neural networks tutorial, Part 1 – Introduction to RNNs. Disponível em: <http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/>. Acesso em: 20 junho 2020.

DENG, L.; YU, D. Deep Learning : Methods and Applications. Foundations and Trends® in Signal Processing v. 7, p. 197–387, 2013. Redmond: Microsoft. Disponível em: < https://www.microsoft.com/en-us/research/publication/deep-learning-methods-and-applications/>. Acesso em: 10 junho 2020

ENRIGHT, J. J.; WURMAN, P. R. Optimization and Coordinated Autonomy in Mobile Fulfillment Systems. In: AAAI CONFERENCE ON AUTOMATED ACTION PLANNING FOR AUTONOMOUS MOBILE ROBOTS, 9., 2011, San Francisco. **Anais …** San Francisco: Hyatt Regency San Francisco. Disponível em: <https://www.aaai.org/ocs/index.php/WS/AAAIW11/paper/view/3917>. Acesso em: 08 maio 2020.

HAZARD, C. J.; WURMAN, P. R.; D'ANDREA, R. Alphabet Soup: A Testbed for Studying Resource Allocation in Multi-vehicle Systems In: AAAI WORKSHOP ON AUCTION MECHANISMS FOR ROBOT COORDINATION, 2006, Massachusetts **Anais …** San Francisco: Hyatt Regency San Francisco. Disponível em: <https://www.researchgate.net/publication/228709440_Alphabet_Soup_A_Testbed_for_Studying_Resource_Allocation_in_Multi-vehicle_Systems>. Acesso em: 05 maio 2020.

HINTON, G. E.; NAIR, V. Rectified Linear Units Improve Restricted Boltzmann Machines. **Proceedings of the 27th International Conference on Machine Learning (ICML-10)**, p. 807–814, 2010. Madison: Omnipress. Disponível em: <https://dl.acm.org/doi/proceedings/10.5555/3104322>. Acesso em: 15 junho 2020

HOCHREITER, S.; SCHMIDHUBER, J. Long Short-Term Memory. **Neural Computation**, v. 9, n. 8, p. 1735–1780, 1997.

JIANG , T.; WANG, J. Z. Study on Path Planning Method for Mobile Robot Based on Fruit Fly Optimization Algorithm. **Applied Mechanics and Materials**, Zürich, v. 536-537, p. 970-973, 2014. Disponível em: < https://www.scientific.net/AMM.536-537.970> . Acesso em: 05 maio 2020.

LIU, Y.; CHEN, M.; Multi-agent Pathfinding Based on Improved Cooperative A* in Kiva System In: 5th International Conference on Control, Automation and Robotics, 2019, Beijing **Anais ...** Nova Jersey: IEEE. Disponível em: <https://ieeexplore.ieee.org/abstract/document/8813319>. Acesso em: 03 maio 2020.

MITCHELL, T. M. Machine Learning. Nova Iorque, v. 1, p. 414, 1997 Nova Iorque: McGraw-Hill

OLAH, C. Understanding LSTM Networks. Disponível em: < http://colah.github.io/posts/2015-08-Understanding-LSTMs/>. Acesso em: 10 maio 2020.

PAN, W. et. Al. "Modified fruit fly optimization algorithm of logistics storage selection" **The International Journal of Advanced Manufacturing Technology**, Londres, v. 93, p. 547-558, 2017. Disponível em: <https://link.springer.com/article/10.1007/s00170-017-0699-x>. Acesso em: 04 maio 2020.

PASZKE, A. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: 33rd Conference on Neural Information Processing Systems, 2019, British Columbia **Anais…** Vancouver, 2019. Disponível em: <https://papers.nips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library>. Acesso em: 25 junho 2020

PEREIRA, M. M. **APRENDIZADO PROFUNDO: REDES LSTM.** 2017. Trabalho de Counclusão de Curso (Graduação em Sistemas de Informação)-Universidade Federal da Grande Dourados, Dourados, MS, 2017

QUEIROZ, A. Q. R. S. CONTROLE DE UM SISTEMA VARIANTE NO TEMPO USANDO REDES NEURAIS ARTIFICIAIS E APRENDIZADO POR REFORÇO. 2018.

41p. Trabalho de Conclusão de Curso (Graduação em Engenharia de Controle e Automação)- Universidade de Brasília, Brasília, DF, 2018

SAMUEL, A. L. Some Studies in Machine Learning Using the Game of Checkers. **IBM Journal of Research and Development,** Armnok, 3v., n. 3., p. 210-229. 1959. Armonk: IBM Publishing. Disponível em: <https://ieeexplore.ieee.org/abstract/document/5392560/authors#authors>. Acesso em: 25 junho 2020

SRIVASTAVA, N. et al. Dropout : A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research, v. 15, p. 1929–1958, 2014. Disponível em: < https://www.jmlr.org/papers/v15/>. Acesso em: 15 junho 2020.

WURMAN, P. R.; D'ANDREA, R.; MOUNTZ, M. Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. **AI Magazine**, Palo Alto, v. 29., n. 1, p. 9-20, 2008. Palo Alto: Association for the Advancement of Artificial Intelligence (AAAI). Disponível em: <http://www.aaai.org/ojs/index.php/aimagazine/article/view/2082 >. Acesso em: 20 mar. 2020.