Desenvolvimento de uma aplicação baseada em *reinforcement learning* na resolução de problemas de tráfego

Alex Drewin Sievers[1], **Guilherme Chagas Kurtz**[1]

[1]Ciência da Computação – Universidade Franciscana (UFN)
Caixa Postal 97.010-032 – Santa Maria – RS – Brasil

{alexsievers, guilhermechagaskurtz}@gmail.com

Abstract. *Traffic problems are considered complex to fully solve with only urban mobility plans because there is a vast amount of variables involved. It is possible to model this data with help of reinforcement learning (RL) technique due to being able to solve - or ease - situations that can occur in traffic. The objective of the present work was to implement a Deep Q-learning agent to maximize vehicle flow efficiency. Lastly, comparing versions of the trained agent with different penalty and testing the best agent in simulations with different configurations.*

Resumo. *Os problemas que ocorrem no trânsito são considerados complexos de serem resolvidos apenas com planos de mobilidade urbana, pois apresentam inúmeras variáveis. É possível modelar essas informações com o auxílio da técnica de reinforcement learning (RL), pois com ela pode-se solucionar - ou amenizar - situações apresentadas no trânsito. O objetivo deste trabalho foi implementar um agente com o algoritmo Deep Q-learning para aumentar a eficiência de fluxo de veículos. Por fim, comparar versões de treinamento com diferentes taxas de penalidade e testar o melhor agente em simulações de diferentes configurações.*

1. Introdução

Segundo [Salatiel 2012], "São Paulo é a cidade [brasileira] que mais recebeu veículos nas ruas: 3,4 milhões. Enquanto a população da capital paulista cresceu 7,9% na primeira década deste século, o número de carros aumentou 68,2%". Muitas metrópoles sofrem com o rápido aumento de veículos no trânsito, porém, este problema depende de diversos fatores, por exemplo: fluxo diário de veículos e horários de movimentação elevados, e mesmo traçando planos de mobilidade urbana, alguns locais ainda sofrem com essa problemática.

A evolução exponencial da computação vem se demonstrando um aliado importante na resolução de problemas no mundo real. Desde os primórdios, o homem enfrentou (e ainda enfrenta) inúmeros contratempos, onde muitos destes não apresentam uma solução até os dias atuais. Uma das áreas promissoras para auxiliar na resolução é a de *machine learning*.

Machine learning consiste em algoritmos para encontrar padrões em quantidades extensas de dados e propor soluções para diversos problemas. Este estudo foi citado pela primeira vez por Arthur Lee Samuel em 1959, pioneiro na área de Inteligência Artificial, onde demonstrou a possibilidade de o computador realizar tarefas melhor do que um ser humano.

Há três categorias básicas que sustentam a área de *machine learning*, são elas: *supervised*, *unsupervised* e *reinforcement learning* (*RL*). A técnica de *RL* pode ser aplicada para solucionar problemas em diversas situações no mundo real, como por exemplo, analisar dados clínicos e encontrar a dose efetiva de um medicamento, jogar uma partida de xadrez ou controlar o ambiente de tráfego veicular. Os problemas enfrentados no trânsito podem ser avaliados e tratados com a utilização de algoritmos para tomada de decisão dentro de *reinforcement learning*.

O objetivo geral deste trabalho foi implementar um algoritmo de *reinforcement learning* para resolver o problema de tráfego veicular. Para alcançar isto, os objetivos específicos são:

- Adquirir conhecimento sobre a técnica de *RL* e seus principais conceitos;
- Estudar casos de uso e algoritmos relevantes ao problema em questão;
- Definir um ambiente da vida real;
- Desenvolver uma solução para o problema do tráfego baseada na técnica de *RL*;
- Comparar os resultados da aplicação com diferentes configurações;

2. *Machine learning*

Arthur Lee Samuel, na década de 50, definiu o conceito *machine learning* como a possibilidade de dar ao computador a habilidade de aprender sem a intervenção humana (*self-learning*), isto é, sem ser diretamente programado [Theobald 2018]. Segundo [Samuel 1959], um computador pode ser programado de modo que ele consiga, por exemplo, jogar uma partida de xadrez melhor do que o próprio desenvolvedor do jogo. *Machine learning* consiste em analisar uma grande quantidade de dados e ter a capacidade de elaborar análises preditivas de possíveis soluções de um problema muito mais rápido que um humano, porém os dois ainda têm de trabalhar juntos, pois a máquina, até então, pode ter resultados inconsistentes [Mueller and Massaron 2016].

Esta área subdivide-se em três categorias, são elas: *supervised*, *unsupervised* e *reinforcement learning* (*RL*). *Supervised* trabalha com hipóteses, consiste em detectar padrões, treinar-se à partir de informações e conclusões já existentes e então elaborar previsões de como resolver o problema em questão [Bishop 2006]. Segundo [Goodfellow et al. 2016], "em *machine learning unsupervised*, não há instrutor, o algoritmo precisa aprender a guiar-se pelos dados sem o seu guia". Isso demonstra a necessidade da máquina de prever os resultados corretos por si só.

Reinforcement learning baseia-se em duas entidades - agente e ambiente - e seus relacionamentos - ação, recompensa e observação. Trata-se de um agente (máquina) que necessita tomar decisões em um determinado ambiente, o qual ele observa, para resolver um problema, onde cada ação possui uma recompensa, ações essas respeitando uma política para determinar qual estratégia tomar com base no estado atual. Durante a observação, o agente pode se encontrar em dois modos: *exploration*, ato de tentar soluções novas experimentando ações diferentes; e *exploitation*, tomar ações que funcionaram no passado. [Sutton and Barto 2018]. Esta categoria em questão será abordada com mais profundidade na próxima seção.

2.1. *Reinforcement learning*

Esta técnica, como esboçado na Figura 1, fundamenta-se na automação do processo de aprendizado, onde o agente necessita tomar ações a fim de maximizar a sua pontuação

final, sem conhecimento de quais comportamentos lhe trarão uma maior recompensa. Ou seja, é preciso que ele descubra por si só, por meio de tentativa e erro.

No entanto, não é suficiente que ele encontre a melhor decisão para o problema em questão, todas as ações tomadas influenciam situações e recompensas futuras, o agente não só tem de achar o melhor comportamento, mas sim combinações dos mesmos para melhorar ainda mais sua recompensa. A principal razão desta técnica utilizar o ambiente é devido ao fato de que nada é conhecido, pois ao contrário das outras categorias, não há modelos para treinamento e nem conjunto de ações ideais para se chegar à melhor pontuação possível.

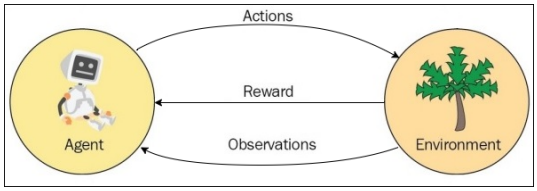

Figura 1. *RL entities and their communications*. Adaptado de [Lapan 2018].

Comparando às outras técnicas, *reinforcement learning* possui um sistema de *feedback* (um retorno do ambiente) que consiste em premiar o agente por ter tomado uma decisão e não outra, assim a máquina aprende a sempre tentar melhorar suas ações. [Lapan 2018] demonstra um modelo que utiliza *RL* em um labirinto, onde há três elementos: rato que é o agente, a comida, sua recompensa e eletricidade, sua penalidade. O agente precisa tomar as melhores decisões para chegar às recompensas da melhor forma, isto é, ter menos ações consideradas erradas, sem nenhum conhecimento prévio sobre o ambiente, portanto, aprenderá com base em tentativas e erros. O retorno do ambiente se mostra muito eficaz no processo de aprendizado, pois com ele o agente tentará tomar decisões melhores nas próximas vezes que for exposto ao mesmo.

Essa técnica é muito mais desafiante comparando às outras pelo fato de que a observação do agente e suas ações estão ligadas diretamente ao seu comportamento. O rato, por exemplo, pode aprender a pegar a recompensa de uma maneira menos eficiente, então não haverá um *feedback* negativo. Outro ponto a se destacar é o conhecimento do ambiente: a única forma do agente saber quais são as melhores ações a serem tomadas é explorando o local onde se encontra (*exploration*), porém, pode-se ter um decréscimo drástico de recompensa caso a exploração seja extensa [Lapan 2018]. A troca entre *exploration* e *exploitation* pode ser feita ao se adotar uma política de decisão.

Um exemplo de política que possibilita uma troca aleatória entre *exploration* e *exploitation* é a ϵ-*greedy*. Nela, o agente continua escolhendo as ações com as maiores recompensas, porém, esta escolha depende de uma variável ϵ, cujo valor varia entre 0 e 1. Assim, gera-se outro valor aleatório, e caso esse valor seja maior que ϵ, é escolhida a ação com a maior recompensa (*exploitation*); caso contrário, uma nova ação é selecionada (*exploration*). Desta forma, garante-se a variação das ações [Parkinson 2019].

Há dois tipos de *reinforcement learning*: *model-free* e *model-based*, conforme mostra a Figura 2. No primeiro, o agente depende somente de tentativas e erros para atualizar seu conhecimento de quais ações trazem uma melhor recompensa. Já no segundo, o agente cria um modelo de transições e recompensas de ações, e então procura

ações que julga serem mais apropriadas para realizar conforme o modelo do ambiente [Dayan and Niv 2008].

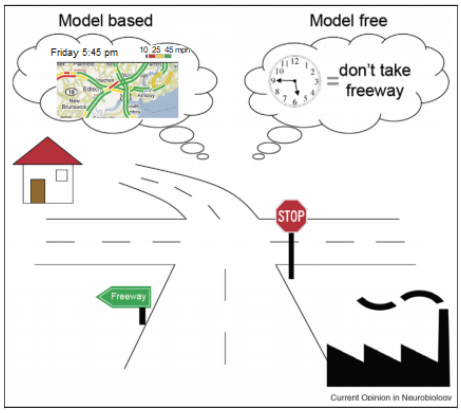

Figura 2. Duas possibilidades para chegar ao destino [Dayan and Niv 2008].

RL pode se demonstrar uma ferramenta muito útil para um programador, principalmente em problemas complexos, pois o humano é suscetível a erros, os quais uma máquina teria uma chance mínima de cometer, sendo somente necessário dar à ela um *feedback* de como se saiu na execução da tarefa. Com isso, evitará executar ações que não foram consideradas boas, podendo alcançar uma solução perfeita, dependendo do problema em questão e do tipo de resultado esperado [Russell and Norvig 1995] [Goodfellow et al. 2016].

As três técnicas de *machine learning* apresentadas podem ser utilizadas para resolução de problemas, porém, é necessária a análise do mesmo para que se escolha a melhor opção. *Supervised* e *unsupervised learning* têm como base informações já coletadas, onde esse aprende com os dados juntamente com as respostas - ou também denominado supervisionado por instrutor, e este aprende por si só. Já em *reinforcement learning*, como ambiente é desconhecido, é preciso aprender por experiência, com recompensa por suas ações, assim torna-se uma alternativa para resolver problemas onde as funções de recompensas são bem definidas, com disponibilidade de execução a longo prazo e com a possibilidade de cometer erros.

Conforme [Li 2019], *RL* pode ser empregado em diversos tipos de problemas no mundo real, estando presente em inúmeras áreas de conhecimento, desde jogos de computador, Química, Medicina, controle de tráfego e outras. Em 2017, a inteligência artificial AlphaGo, desenvolvida pela empresa Google, venceu o melhor jogador do mundo em uma partida do jogo de tabuleiro estratégico Go, um jogo tão complexo quanto xadrez. O AlphaGo utiliza a técnica Monte Carlo, algoritmo de busca heurística para tomada de decisões que, guiado por uma rede neural, calcula a probabilidade de realizar cada jogada. A rede neural simula uma jogada no nó mais superior, buscando maximizar padrões como: valor da ação (Q) e coeficiente de exploração (U); e então simula a jogada. Dentro desta, outras simulações serão geradas até não restar nenhum nó folha a ser explorado. Então, os nós mais abaixo retornam sua pontuação para o nó pai, e na sequência é realizada a jogada de fato no jogo Go (ver Anexo A, Figura 13) [Ye 2020].

2.2. *Reinforcement learning* aplicado ao problema de tráfego

Machine learning, mais precisamente a técnica *reinforcement learning*, pode auxiliar na resolução do problema de tráfego veicular por apresentar características que são adequadas a um modelo de *RL*, tais como:

- Agente: veículo automotor ou semáforo;
- Ambiente: vias de locomoção veicular e intersecções entre elas;
- Ação: comportamento do agente no ambiente, podendo ele ser troca de via ou ultrapassagem, caso o agente seja o veículo, ou troca de sinal (verde, vermelho) se semáforo;
- Recompensa: resultado da ação no ambiente;
- Estado: representação do estado atual do agente/ambiente;
- Observação: observação do estado atual do ambiente;

2.2.1. Algoritmos

Além dos agentes empregados, é necessário utilizar algoritmos para auxiliar na resolução do problema, tal como o *Longest Queue First* (*LQF*), que baseia-se em priorizar filas mais extensas, sendo uma tentativa para auxiliar veículos autorizados à quebrar regras de trânsito (ambulâncias, polícia, bombeiros) [Louati et al. 2018].

Ademais, há o *Q-learning* e sua variação *Deep Q-learning*, sendo que os dois se mostram muito eficiente neste tipo de problema em particular. Estes algoritmos consistem em predizer quais decisões tomar, em diferentes situações, para interagir com o ambiente, que é desconhecido, para maximizar a recompensa do agente, assim criando uma tabela de combinações de estados e ações no caso do *Q-learning*. Já a sua outra versão, utiliza uma rede neural para aproximar o *Q-value* dado um estado, assim eliminando a necessidade de utilizar uma tabela de combinações dos mesmos [Venkatachalam 2019].

A literatura indica empregar o algoritmo *Deep Q-learning* ao invés de *Q-learning*, pois em diversas situações a tabela de combinação pode se tornar mais extensa conforme a quantidade de estados e ações, assim perdendo eficiência.

2.2.1.1 *Deep Q-learning*

Este algoritmo de *RL* trabalha com uma rede neural para aproximar os *Q-values* (valor relativo de todas as possíveis ações) do agente, assim, aumenta-se a eficiência de tomada de decisões do mesmo durante os processos de treinamento e teste (prova de conceito), pois é capaz de generalizar experiências com estados não vistos [Choudhary 2019]. Em Inteligência Artificial, é possível criar uma analogia entre rede neural artificial e o cérebro humano: ela [rede neural] é composta por camadas de neurônios que transmitem dados entre si, onde a primeira é considerada a camada de entrada, a última como de saída, e uma ou muitas camadas ocultas ligadas às duas extremidades, como demonstrado na Figura 3.

A configuração da rede neural depende do problema a ser solucionado, segundo [Heaton 2018], uma camada oculta demonstra-se suficiente para resolução de diversos

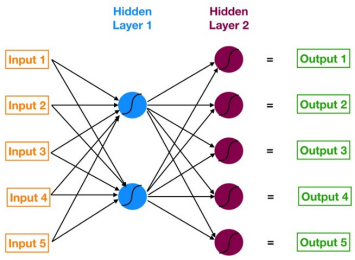

Figura 3. *Neural Network* [Yiu 2019].

problemas. Para determinar a quantidade de neurônios de uma camada oculta, o autor sugere seguir alguns critérios, como por exemplo:

- Ser um valor entre o tamanho da camada de entrada e o da camada de saída;
- Ser menor que o dobro do tamanho da camada de entrada;
- Ser 2/3 do tamanho da camada de entrada, somado ao da camada de saída;

No exemplo da Figura 3, é empregada uma rede neural artificial com arquitetura *feed forward* com a camada de entrada contendo cinco neurônios, duas camadas ocultas com dois e cinco neurônios, respectivamente, e a camada de saída com cinco neurônios, onde todos os neurônios de uma camada estão ligados a todos da camada seguinte. Cada conexão (também chamada de sinapse) possui um valor numérico denominado *weight* (peso), que é um parâmetro para transformar o dado do neurônio de saída para o de chegada. Além disso, é necessário determinar se esse dado deve ser ativado ou não por meio de uma função de ativação, causando, então, um efeito de não linearidade [Yiu 2019].

Ademais, para uma rede neural artificial estar pronta para obter resultados concisos de predição, é necessário ajustar os pesos de cada sinapse por meio de funções de erro e otimização, onde o objetivo é encontrar valores que minimizam erros (diferença entre a saída gerada e o valor real). Uma delas é o *gradient descent* que indica para que direção o peso deve seguir, isto é, aumentar ou diminuir, além da intensidade do ajuste. Assim, ajusta-se os pesos por meio da técnica *backpropagation*, realizando o percurso inverso, da camada de saída para a de entrada, ajustando o peso de cada conexão. Este processo ocorre até haver um erro ínfimo (o mais próximo de zero) [Nielsen 2019].

Na rede neural do *Deep Q-learning*, o fluxo de dados é dado pela Figura 4. Usa-se o conceito de experiência de *replay*, que é considerada uma memória para armazenar as ações tomadas pelo agente, e após cada episódio de treinamento - durante um *loop* - envia-se um conjunto (*batch*) de ações para otimizar os pesos das sinapses [Paszke 2017].

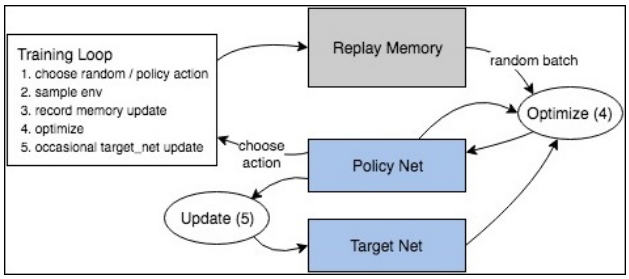

Figura 4. *DQN's data flow* [Paszke 2017].

2.2.2. Métodos de controle de tráfego

São mencionados diversos métodos de controle de tráfego veicular, os quais servem como base para a configuração do agente, são eles: *Split Cycle Offset Optimisation Technique* (*SCOOT*), *Fixed-time Control* (*FT*), *Self-Organizing Traffic Light Control* (SOTL) e *Deep Reinforcement Learning for Traffic Light Control* (*DRL*). O *SCOOT* mensura constantemente a quantidade de tráfego em uma região com o objetivo de ajustar o tempo de espera de veículos em uma área específica. Em *FT*, os semáforos possuem tempos fixos. O *SOTL* é um sistema onde todos os elementos interagem entre si para alcançar dinamicamente um objetivo [Gershenson 2004] e *DRL* é um método que aplica o algoritmo *Deep Q-learning* para encontrar a melhor configuração para os semáforos de intersecções [Wei et al. 2018]. Para o presente trabalho, será abordado o *FT*, pois a ideia principal é que o agente adeque o sinal sem mudar o tempo de duração.

2.2.3. Ferramentas

A linguagem de programação Python é uma opção para o desenvolvimento de modelos que realizam o tratamento de uma grande quantidade de dados, uma vez que apresenta bibliotecas e *frameworks* benéficas para a área de Inteligência Artificial e *machine learning*, listados abaixo:

- Pyqlearning [Pyqlearning 2019]: criada especificamente para *Q-learning* e *Deep Q-learning*, podendo ser utilizada para jogos e robótica;
- Keras [Keras 2015]: implementada principalmente para jogos, trabalha com algoritmos *deep reinforcement learning*;
- TensorForce [Kuhnle et al. 2017]: juntamente com a biblioteca TensorFlow, TensorForce realiza análise com quantidade massiva de dados envolvendo técnicas *machine learning*;
- Chainer [Fujita et al. 2019]: *framework* que implementa redes neurais para *deep learning*;

Além disso, é possível modelar e simular áreas de tráfego no software *Simulation of Urban Mobility* (*SUMO*), a fim de criar algo mais próximo da realidade. O *SUMO* pode ser considerado como uma conexão entre o código em Python e os objetos de simulação, como veículos, vias, pedestres, semáforos [Lopez et al. 2018].

3. Trabalhos Relacionados

Nesta seção serão apresentados trabalhos que se baseiam na técnica de *reinforcement learning* e que são considerados correlatos ao presente trabalho. O primeiro trata-se de um estudo com dados reais, onde uma parcela dos semáforos possuem câmeras equipadas com inteligência artificial. Já o segundo trabalho simula o controle de tráfego utilizando o algoritmo *Q-learning*. Por fim, o terceiro trabalho trata o uso de multiagentes com o *Deep Q-learning*.

3.1. *IntelliLight: A Reinforcement Learning Approach for Intelligent Traffic Light Control*

Em grande parte do mundo, o tráfego é controlado por planos já existentes, e não pela observação do fluxo de veículos nas vias em tempo real. No estudo de [Wei et al. 2018]

foi proposto um modelo inteligente para realizar o controle de tráfego, utilizando a técnica de *reinforcement learning* em dados de câmeras de vigilância na cidade de Jinan, China.

O trabalho foi dividido em duas partes: *offline*, onde o agente realiza o treinamento em um ambiente com tempo fixo para troca de sinais nos semáforos, armazenando todas informações em um banco de dados; e *online*, onde o agente é submetido a um ambiente mais complexo, onde ele irá observar o estado do ambiente em intervalos de tempos Δt, e assim tomará uma ação conforme uma combinação entre *exploration* e *exploitation*. À cada ação tomada o agente recebe uma recompensa, e as informações de estado, ação e recompensa são armazenadas em uma memória, a qual é utilizada, depois de um tempo t, para melhorias no modelo, como esboçado no Anexo E, Figura 16.

Além disso, foram definidos os conceitos de *RL* - estado, recompensa e ação como:

- Estado: representado por uma intersecção de vias, onde têm-se informações sobre o número de veículos, tamanho da fila, o tempo de espera de cada veículo, estado atual e o próximo sinal do semáforo (verde, vermelho ou amarelo);
- Recompensa: definida como um valor real onde se leva em consideração os valores de recompensa, penalidade, taxa de aprendizado e estimativas dos últimos *Q-values*. A equação está evidenciada no Anexo F, Eq. 2.
- Ação: Podendo ser 1 (trocar o sinal do semáforo) ou 0 (manter o sinal).

Por meio do modelo criado, foram realizados inúmeros testes com dados sintéticos e dados reais, e por fim, demonstrou-se que o modelo apresentou uma melhor performance em comparação a métodos convencionais (apresentados na Seção 2.2.2).

3.2. *Reinforcement Learning for True Adaptive Traffic Signal Control*

No trabalho de [Abdulhai et al. 2003] é abordada a técnica de *RL* em um ambiente de tráfego isolado e escalável, juntamente com o algoritmo *Q-learning* (explicado na seção 2.2.1).

É enfatizado, no artigo, que o *Q-learning* possui uma característica única, entradas sensoriais, que é considerado um *feedback*, ao agente, sobre o estado atual s do ambiente. No algoritmo, o agente é responsável por interpretar essas entradas para tomar ações a em um tempo t, e a ação a ser tomada depende diretamente do *Q-value* $\mathbf{Q}_{s,a}$, o qual reflete na decisão tomada, gerando uma transição do estado atual do ambiente s para s'. Como o ambiente é desconhecido, o agente deve aprender alternativas (ações) para cada estado do ambiente, por meio de combinações entre *exploitation* e *exploration*. O objetivo do agente, dentro do algoritmo, é de maximizar a recompensa ou minimizar a penalidade (dependendo do problema), assim recebendo, a cada ação tomada, uma recompensa $\mathbf{r_{s,a}}$. Por fim, as variáveis estado, ação e recompensa são utilizadas para atualizar o valor do *Q-value* $\mathbf{Q}_{t-1(s,a)}$ (Ver Anexo H, Eq. 3).

Conforme o autor, a interação dinâmica entre o agente e o ambiente demonstrou ser significativamente benéfica para o controle de tráfego, ao contrário de métodos convencionais. A utilização do algoritmo *Q-learning* tanto em um ambiente de tráfego isolado como em outro com uma intersecção de vias duplas trouxe resultados promissores, em que o agente demonstrou a capacidade de se adaptar conforme mudanças são efetuadas no ambiente.

3.3. *Reinforcement learning-based multi-agent system for network traffic signal control*

Segundo [Arel et al. 2010], utilizar *RL* para o controle de sinais de tráfego veicular é uma tarefa desafiadora. No artigo, a técnica emprega dois agentes para monitorarem intersecções de múltiplas vias duplas, esboçado no Anexo G, na Figura 17. O agente, responsável pelo controle das vias marginais, emprega o algoritmo *LQF*, explicado na seção 2.2.1, enquanto que o segundo agente, controlador das vias mais internas, trabalha com valores (*feedback*) das condições de tráfego na sua área, utilizando o algoritmo *Deep Q-learning*, além de receber as observações do agente externo.

No trabalho, o autor definiu os três principais conceitos como:

- Estado: representado por um vetor de oito posições, onde cada uma simboliza o fluxo de veículos de uma via dupla. Esse fluxo é definido como a razão entre o tempo total de espera na via e a média do tempo total de espera de todas as vias na intersecção;
- Recompensa: Um valor entre -1 e 1 (onde positivo se o tempo de espera foi menor que o da ação anterior). A equação é evidenciada no Anexo I, Eq. 4:
- Ação: Podendo ser qualquer valor dentro da combinação de cada intersecção;

O algoritmo *Deep Q-learning* é empregado, no agente interno, por ser efetivo em escolher ações conforme o estado atual das vias, e para cada ação selecionada, o *Q-value* é atualizado conforme a ação tomada e sua recompensa, taxa de aprendizado e de penalidade, além da estimativa da recompensa da próxima ação. A fórmula é mais detalhada no Anexo J, Eq. 5.

Acrescentando ainda, o autor afirma que há a possibilidade de empregar a metodologia em um ambiente com intersecções mais complexas sem ter muitas mudanças na sua estrutura. Por fim, ressalta que o protótipo realiza o controle de uma forma mais próxima do ideal, isto é, minimizando o tempo de espera de veículos na fila em diversas intersecções.

3.4. Considerações sobre os trabalhos relacionados

Os trabalhos acima apresentados foram de suma importância para o presente trabalho, onde todos definiram os conceitos e algoritmos de *reinforcement learning* com base na sua problematização.

O artigo de [Wei et al. 2018] demonstrou a capacidade de utilizar *RL* com dados reais de tráfego, além de apresentar um modelo bem definido. Contudo, foi ressaltado que o ambiente deveria ser alterado para que fosse simulado algo mais realístico. Já o trabalho de [Abdulhai et al. 2003] trouxe a utilização do algoritmo *Q-learning* em um ambiente que apresenta probabilidades de congestionamento. E, finalmente, o trabalho de [Arel et al. 2010] apresentou um cenário multiagente, fazendo uso do *Deep Q-learning*, onde se mostra necessária a troca de informações nas intersecções.

4. Metodologia

Para o desenvolvimento do presente trabalho, foi utilizada a linguagem de programação Python, juntamente com a biblioteca Keras. Para resolução do mesmo, os principais conceitos da técnica de *reinforcement learning* foram definidos e modelados conforme o resultado esperado, isto é, maior fluidez no trânsito e menos congestionamento de veículos.

O algoritmo desenvolvido foi dividido em duas etapas, são elas: *offline*, em que o agente treinou a si próprio, sem um conhecimento prévio, precisando explorar o ambiente; e *online*, em que de fato ele é colocado na situação que simula o ambiente. Além disso, fez-se necessário executar o algoritmo na plataforma Google Colab, o qual dispõe de servidores equipado com bibliotecas para processamento (com CPU ou GPU) de algoritmos de *machine learning*.

4.1. Agente

Conforme a Figura 5 (a), optou-se considerar o agente como um semáforo que realiza ações dentro de um ambiente, essas ações podendo ser: troca de sinal seguindo a regra: verde para sinal aberto e vermelho para fechado, ou prolongar o tempo na situação que se encontra. Todas as decisões tomadas pelo agente foram com base na quantidade de veículos esperando nas vias e também quantos realizaram o cruzamento de uma via para outra.

Um trecho do código do agente é exposto na Figura 5 (b), onde inicializa-se a rede neural com os parâmetros: número de estados (como camada de entrada); número de ações (camada de saída); tamanho da camada oculta e sua quantidade de neurônios; e, taxa de aprendizado (para ajuste dos pesos). A primeira versão do agente não foi codificada com base no semáforo da vida real, o qual contém sinal amarelo entre os sinais verde e vermelho, considerando somente estes como ações, assim, o agente proposto trata todos os sinais como possíveis ações.

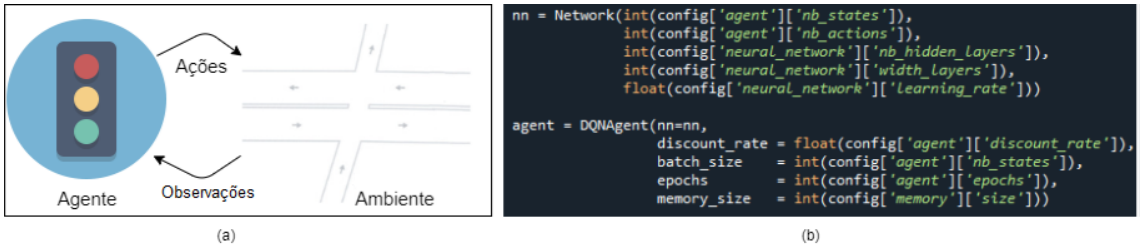

Figura 5. Modelo a ser utilizado.

4.2. Ambiente

Como ambiente, foi considerado um exemplo de trânsito real no município de Santa Maria, RS, o qual não possui um plano de mobilidade urbana eficiente capaz de coordenar a demanda de veículos, assim causando diversos congestionamentos e lentidão nas vias próximas. Neste caso, modelou-se a intersecção entre as ruas Presidente Vargas e Visconde de Pelotas. Cada ação, tomada pelo agente, implica a mudança de estado do ambiente, isto é, a quantidade de veículos presentes nas vias conforme demonstra a Figura 6, informações da estrutura do ambiente são mostrados no Anexo B, Figura 14.

4.2.1. Vias

Para que o agente aprendesse e posteriormente tomasse decisões, foi necessário que o mesmo observasse o estado do ambiente durante um episódio. Conforme [Vidali 2017], as vias do ambiente foram mapeadas e agrupadas conforme o seu sentido de condução,

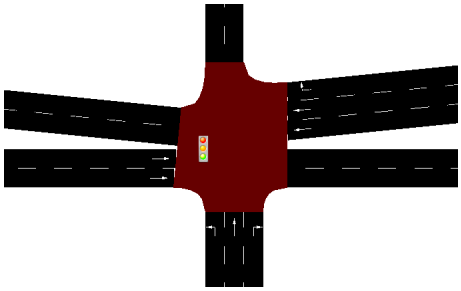

Figura 6. Ambiente modelado no SUMO.

como exemplificado na Figura 7, onde as vias em vermelho (sentido direta para esquerda) contam somente como uma via, já a via em laranja, que tem sentido direita para cima, é considerada outra via.

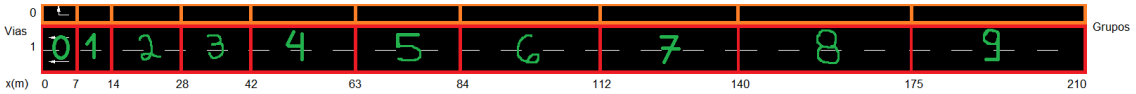

Figura 7. Exemplo de mapeamento do estado de uma via.

Foi preciso mapear as posições dos veículos dentro de uma via, isto é, identificar o quão longe os mesmos estão da intersecção (onde está o agente). Para isso, levou-se em consideração o tamanho dos veículos para a divisão de grupos, onde o primeiro grupo está mais próximo do agente e o último mais longe.

Assim, seguindo a ideia da Seção 3.3, as vias foram divididas em segmentos seguindo o critério de quantidade de veículos em cada grupo, por exemplo: 1 veículo no 1º e 2º grupo, 2 veículos no 3º e 4º grupo, e assim sucessivamente, até o 10º grupo.

4.3. Deep Q-Learning

Para aumentar a eficiência do agente, isto é, diminuir o tempo médio de espera e o tamanho de filas de espera, foi necessário que houvesse uma avaliação das ações tomadas pelo agente por meio de *feedbacks* (recompensas e observações). A recompensa geralmente é definida como sendo um valor real, positivo ou negativo, em intervalos que variam de acordo com o algoritmo utilizado para o seu cálculo. Para calcular a recompensa do agente, foi levada em consideração a diferença entre o tempo de espera de todos os veículos, de todas as vias, da ação anterior com a ação mais recente. Valores menores do que zero significam que o tempo de espera aumentou, consequentemente recompensas negativas, e vice versa.

Foi utilizado o algoritmo *Deep Q-learning*, juntamente com uma rede neural e experiência de *replay*, para calcular a recompensa, com base na derivação da equação matemática de Bellman (Ver Anexo C, Eq. 1), a qual tem o propósito de encontrar a melhor política a ser utilizada pelo agente [Tanwar 2019]. Para mais detalhes sobre o pseudocódigo do *Deep Q-learning*, ver Anexo D, Figura 15.

Seguindo a ideia de [Vidali 2017], as informações de entrada do algoritmo são: taxa de penalidade (γ) e de aprendizado (α), quantidade total de episódios que irão ser executados, quantidade de veículos gerados, quantidade e tamanho de camadas ocultas,

capacidade da memória do agente, tamanho do *batch* de ações para treinamento e o conjunto de estados e de ações.

No final da execução do algoritmo, informações relevantes foram comparadas com resultados de outras configurações para análise comportamental do agente e do ambiente. As informações coletadas e comparadas são: o tempo médio de espera de veículos; a média do tamanho das filas de espera; e a média de todas as recompensas.

Para fins de validação, a taxa de penalidade foi variada entre 0.05 e 0.70, pois esta é considerada um incentivo, mesmo que negativo, para que o agente procure terminar o treinamento em um menor tempo possível [Granatyr et al. 2019]. Quanto menor o valor da taxa, mais flexível é o ambiente com erros e vice-versa. Assim evidenciando a maleabilidade do ambiente e a liberdade do agente para escolher as ações a tomar.

Desta maneira, a simulação do algoritmo foi feita conforme demonstra a Figura 8, onde a mesma é inicializada juntamente com o agente, configurações do SUMO (visualização e tempo de duração de cada episódio), tempo de duração de um episódio e uma *flag* para verificar se a simulação deve ser no modo treinamento ou teste. Posteriormente roda-se os episódios gerando veículos aleatoriamente, e, por fim, salva-se o modelo da rede neural.

```
simulation = Simulation(agent, sumo,
                        int(config['simulation']['max_steps']), is_training=True)
total_epochs = int(config['simulation']['total_episodes'])
for epoch in range(total_epochs):
    print('Starting simulation - Episode: {}/{}'.format(epoch+1, total_epochs))
    traffic_generator.generate_routefile()
    start_time = timeit.default_timer()
    simulation.run(agent.get_epsilon(epoch)) #get next epsilon and send to simulation
    print('Execution time: {:.1f} s\n'.format(timeit.default_timer() - start_time))
    if (epoch+1) % 100 == 0 or (epoch+1) == total_epochs: #backup
        agent.nn._save_model(model_path)
        for key, value in simulation.get_stats().items():
            plot_data(value, key, model_path, 'training')
```

Figura 8. Trecho de código para simulação dos episódios.

4.4. Treinamento, simulação e validação

Para o treinamento do agente, foi escolhido executar 200 episódios, onde em média, no Google Colab, durou 120 segundos, cada um gerou 1000 veículos em um período de tempo de 6000 microssegundos, o agente foi criado contendo uma memória com armazenamento máximo de 50 mil tuplas de dados (último estado, novo estado, ação e recompensa). O treinamento da rede neural foi-se dado com uma amostra de 100 tuplas por vez, durante 500 épocas com uma taxa de aprendizado de 0.001.

Seguindo o pseudocódigo do *Deep Q-learning*, a execução da simulação é dada da seguinte maneira (como mostrado na Figura 9): para um episódio, cria-se uma conexão entre o ambiente e o agente, inicializa-se os valores que são usados para avaliar o desempenho do agente (soma das recompensas, tempo total de espera e tamanho das filas), é escolhida a ação a ser tomada com base no estado atual do ambiente e na política adotada, executa-se a ação no ambiente e o agente observa as consequências da mesma no ambiente. Por fim, o agente salva, em sua memória, a tupla de dados (mencionado no parágrafo anterior). Então, ele [o agente] realiza o ajuste dos pesos da rede neural no intervalo entre episódios até finalizar a simulação. Após o treinamento do agente, o mesmo foi colocado em uma nova simulação com 100 episódios para validar o seu aprendizado, isto é, sem ajustar os pesos da rede neural.

```
def run(self, epsilon):
    traci.start(self.sumo)

    total_reward = 0
    total_waiting_time = 0
    total_waiting_length = 0
    step = 0
    last_state = self.get_state()
    last_total_waiting = 0

    while step < self.max_steps:
        action = self.agent.select_action(last_state, epsilon)

        next_state, steps_done, total_waiting, observation = self.execute_step(action, step)

        reward = last_total_waiting - total_waiting

        #save data in agent's memory
        self.agent.memory.add_sample((last_state, next_state, action, reward))

        step += steps_done
        total_waiting_time += observation[0]
        total_waiting_length += observation[1]
        last_state = next_state
        last_total_waiting = total_waiting

        if reward < 0:
          total_reward += reward

    print('Total reward: {}'.format(total_reward))
    #save stats for later plotting
    self.save_stats(total_reward, total_waiting_time, total_waiting_length)
    traci.close()

    #train NN only in between epochs
    self.agent.experience_replay()
```

Figura 9. Trecho de código para simulação dos episódios.

Para a tomada de decisão do agente, foi-se escolhido uma variação do ϵ-*greedy*, a qual diminui-se exponencialmente a probabilidade de escolher novas ações, uma vez que, no decorrer dos episódios, o agente vai melhorando o seu conhecimento, com isso tendo mais chances de escolher as melhores ações [Balsys 2019].

5. Resultados

Como resultados, à partir da variação da taxa de penalidade descrita anteriormente, nas configurações da taxa 0.05 demonstrado na Figura 10, (a, b, c), 0.35 (d, e, f) e 0.70 (g, h, i), respectivamente.

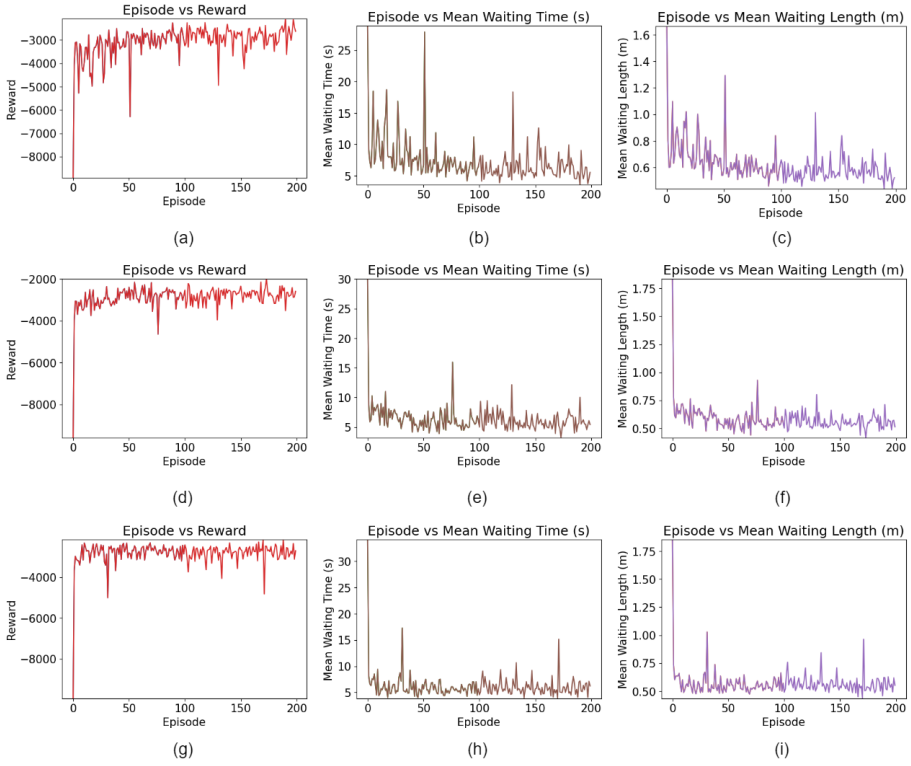

Figura 10. Três exemplares de treinamento do agente.

Foi possível notar que, quanto menor a taxa de penalidade, mais liberdade o agente tem para tomar ações, consequentemente, haverá uma grande variação nas suas recompensas negativas e isto influencia, diretamente, o tempo de espera nas vias e o tamanho das filas de espera. Ademais, foi possível evidenciar que há uma semelhança entre os resultados obtidos nos treinamentos com taxa de penalidade de 0.35 e 0.70. Com isso, foi-se

colocado em teste o agente com penalidade 0.70, pois deseja-se que ele tome ações corretas o mais rápido possível, como mostrado na Figura 11, onde gerou-se 2000 veículos, em 100 episódios de 5000 microssegundos cada. Já na Figura 12, com o mesmo agente, foi-se gerado 3000 veículos, pôde-se perceber que, quanto mais demanda o agente tiver, mais negativas são suas recompensas.

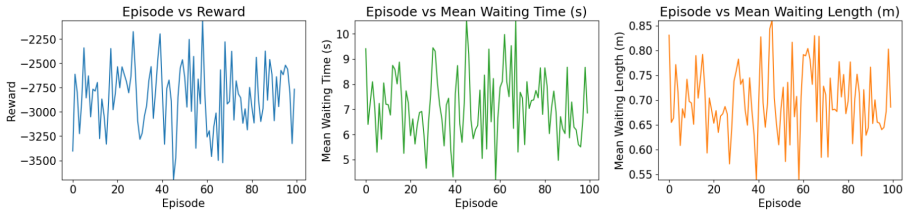

Figura 11. Fase de teste do agente com 2000 veículos.

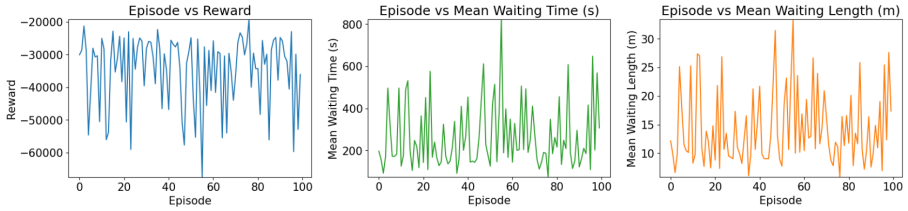

Figura 12. Fase de teste do agente com 3000 veículos.

6. Conclusão

Neste trabalho, foram abordados os principais conceitos da técnica de *reinforcement learning*, modelando suas principais características para resolver o problema de tráfego em locais que possuem dificuldade de evasão de veículos.

No TFG 2, foi implementada uma solução visual para o problema em um ambiente da vida real presente na cidade de Santa Maria, RS, com a utilização do algoritmo *Deep Q-learning* e da biblioteca Keras para analisar o comportamento do agente com diferentes taxas de penalização.

Para trabalhos futuros, poderá ser realizado um estudo maior em relação à troca de sinais do agente, assim aproximando-se mais da realidade, além da inclusão de novos elementos, como pedestres e faixas de segurança.

Referências

Abdulhai, B., Pringle, R., and Karakoulas, G. J. (2003). Reinforcement learning for true adaptive traffic signal control. *Journal of Transportation Engineering*, 129(3).

Arel, I., Liu, C., Urbanik, T., and Kohls, A. (2010). Reinforcement learning-based multi-agent system for network traffic signal control. *IET Intelligent Transport Systems*, 4(2):128–135.

Balsys, R. (2019). Epsilon-greedy in deep q learning. `https://pylessons.com/Epsilon-Greedy-DQN/`. [Online; acessado em 09-12-2020].

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer Science Business Media, LLC.

Choudhary, A. (2019). A hands-on introduction to deep q-learning using openai gym in python. https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/. [Online; acessado em 24-05-2020].

Dayan, P. and Niv, Y. (2008). Reinforcement learning: The good, the bad and the ugly. *Elsevier*, 18:1–12.

Fujita, Y., Kataoka, T., Nagarajan, P., and Ishikawa, T. (2019). Chainerrl: A deep reinforcement learning library. https://chainer.org/. [Online; acessado em 19-05-2020].

Gershenson, C. (2004). Self-organizing traffic lights. *Complex Systems*, 16:29–53.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Granatyr, J., de Ponteves, H., and Eremenko, K. (2019). Aprendizagem por reforço com deep learning, pytorch e python. https://www.udemy.com/course/aprendizagem-reforco-deep-learning-pytorch-python/. [Online; acessado em 21-08-2020].

Gupta, A. (2020). Deep q-learning with tensorflow 2. https://medium.com/@aniket.tcdav/deep-q-learning-with-tensorflow-2-686b700c868b. [Online; acessado em 23-10-2020].

Heaton, J. (2018). *Introduction to Neural Networks for Java*. Heaton Research, 2 edition.

Keras (2015). Keras. https://keras.io/. [Online; acessado em 19-05-2020].

Kuhnle, A., Schaarschmidt, M., and Fricke, K. (2017). Tensorforce: a tensorflow library for applied reinforcement learning. https://github.com/tensorforce/tensorforce. [Online; acessado em 19-05-2020].

Lapan, M. (2018). *Deep Reinforcement Learning Hands-On*. Packt Publishing.

Li, Y. (2019). Reinforcement learning applications. https://medium.com/@yuxili/rl-applications-73ef685c07eb. [Online; acessado em 31-03-2020].

Lopez, P. A., Behrisch, M., Bieker-Walz, L., Erdmann, J., Flötteröd, Y.-P., Hilbrich, R., Lücken, L., Rummel, J., Wagner, P., and Wießner, E. (2018). Microscopic traffic simulation using sumo. *IEEE Intelligent Transportation Systems Conference (ITSC)*.

Louati, A., Elkosantini, S., and Darmoul, S. (2018). Multi-agent preemptive longest queue first system to manage the crossing of emergency vehicles at interrupted intersections. *European Transport Research Review*, 10.

Mueller, J. P. and Massaron, L. (2016). *Machine Learning For Dummies*. John Wiley Sons, Inc.

Nielsen, M. (2019). How the backpropagation algorithm works. http://neuralnetworksanddeeplearning.com/chap2.html. [Online; acessado em 17-10-2020].

Parkinson, A. (2019). The epsilon-greedy algorithm for reinforcement learning. https://medium.com/analytics-vidhya/the-epsilon-greedy-

algorithm-for-reinforcement-learning-5fe6f96dc870. [Online; acessado em 02-06-2020].

Paszke, A. (2017). Reinforcement learning (dqn) tutorial. https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html. [Online; acessado em 18-10-2020].

Pyqlearning (2019). Pyqlearning. https://pypi.org/project/pyqlearning/. [Online; acessado em 19-05-2020].

Russell, S. J. and Norvig, P. (1995). *Artificial Intelligence A Modern Approach*. Alan Apt.

Salatiel, J. R. (2012). Mobilidade urbana - como solucionar o problema do trânsito nas metrópoles. https://vestibular.uol.com.br/resumo-das-disciplinas/atualidades/mobilidade-urbana-como-solucionar-o-problema-do-transito-nas-metropoles.htm. [Online; acessado em 12-05-2020].

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210—229.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D. (2017). Mastering the game of go without human knowledge. *Nature*, (550):354–359.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT Press.

Tanwar, S. (2019). Bellman equation and dynamic programming. https://medium.com/@sanchittanwar75/bellman-equation-and-dynamic-programming-773ce67fc6a7. [Online; acessado em 24-05-2020].

Theobald, O. (2018). *Machine Learning for Absolute Beginners: A Plain English Introduction*. Independently Published, 2 edition.

Venkatachalam, M. (2019). Q-learning. https://towardsdatascience.com/q-learning-54b841f3f9e4. [Online; acessado em 17-04-2020].

Vidali, A. (2017). Simulation of a traffic light scenario controlled by a deep reinforcement learning agent.

Wei, H., Zheng, G., Yao, H., and Li, Z. (2018). Intellilight: A reinforcement learning approach for intelligent traffic light control.

Ye, A. (2020). How deepmind's alphago became the world's top go player. https://medium.com/better-programming/how-deepminds-alphago-became-the-world-s-top-go-player-5b275e553d6a. [Online; acessado em 07-04-2020].

Yiu, T. (2019). Understanding neural networks. https://towardsdatascience.com/understanding-neural-networks-19020b758230. [Online; acessado em 17-10-2020].

A. Figura explicativa do modelo de Monte Carlo do AlphaGo na Seção 2.1

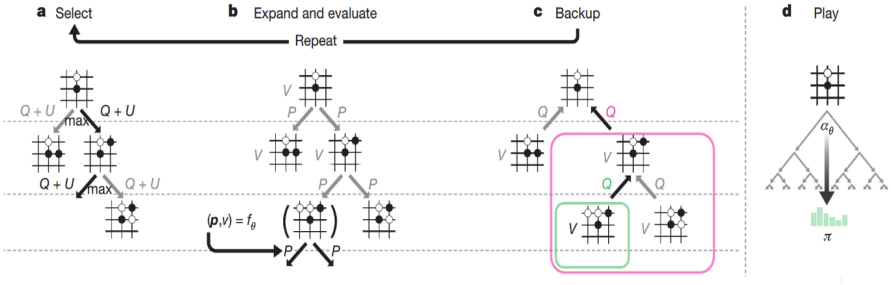

Figura 13. ***AlphaGo's Monte Carlo Tree Search*** **[Silver et al. 2017].**

B. Modelagem do ambiente

```
<?xml version="1.0" encoding="UTF-8"?>

<net version="1.6" junctionCornerDetail="5" limitTurnSpeed="5.50" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/ne

    <location netOffset="0.00,0.00" convBoundary="-100.00,-70.00,100.00,50.00" origBoundary="-10000000000.00,-10000000000.00,10000000000.00,10000000000.00" projParameter="!"/>

    <edge id=":center_0" function="internal">
        <lane id=":center_0_0" index="0" disallow="pedestrian" speed="7.82" length="14.53" shape="13.54,9.39 8.32,9.25 4.58,10.02 2.35,11.70 1.60,14.29"/>
    </edge>
    <edge id=":center_1" function="internal">
        <lane id=":center_1_0" index="0" disallow="pedestrian" speed="13.89" length="17.94" shape="13.55,6.18 8.31,5.54 4.71,5.05 1.11,4.89 -4.14,5.24"/>
        <lane id=":center_1_1" index="1" disallow="pedestrian" speed="13.89" length="17.94" shape="13.56,2.96 8.23,2.32 4.56,1.84 0.89,1.69 -4.45,2.05"/>
    </edge>
    <edge id=":center_3" function="internal">
        <lane id=":center_3_0" index="0" disallow="pedestrian" speed="7.32" length="11.71" shape="8.00,-10.40 8.35,-6.55 9.40,-3.80 11.14,-2.15 13.58,-1.60"/>
    </edge>
    <edge id=":center_4" function="internal">
        <lane id=":center_4_0" index="0" disallow="pedestrian" speed="13.89" length="24.82" shape="4.80,-10.40 4.80,14.29"/>
        <lane id=":center_4_1" index="1" disallow="pedestrian" speed="13.89" length="24.82" shape="4.80,-10.40 4.30,-2.70 3.20,2.86 2.10,7.96 1.60,14.29"/>
    </edge>
    <edge id=":center_6" function="internal">
        <lane id=":center_6_0" index="0" disallow="pedestrian" speed="9.25" length="17.85" shape="1.60,-10.40 1.24,-3.77 0.17,1.04 -1.63,4.05 -4.14,5.24"/>
    </edge>
    <edge id=":center_7" function="internal">
        <lane id=":center_7_0" index="0" disallow="pedestrian" speed="13.89" length="18.55" shape="-5.12,-4.80 13.59,-4.80"/>
        <lane id=":center_7_1" index="1" disallow="pedestrian" speed="13.89" length="18.55" shape="-4.81,-1.60 13.58,-1.60"/>
    </edge>

    <edge id="BC" from="bot_center" to="center" priority="-1" length="210.00">
        <lane id="BC_0" index="0" disallow="pedestrian" speed="13.89" length="210.00" shape="8.00,-70.00 8.00,-10.40"/>
        <lane id="BC_1" index="1" disallow="pedestrian" speed="13.89" length="210.00" shape="4.80,-70.00 4.80,-10.40"/>
        <lane id="BC_2" index="2" disallow="pedestrian" speed="13.89" length="210.00" shape="1.60,-70.00 1.60,-10.40"/>
    </edge>
    <edge id="BL" from="bot_left" to="center" priority="-1" length="210.00">
        <lane id="BL_0" index="0" disallow="pedestrian" speed="13.89" length="210.00" shape="-100.00,-4.80 -5.12,-4.80"/>
        <lane id="BL_1" index="1" disallow="pedestrian" speed="13.89" length="210.00" shape="-100.00,-1.60 -4.81,-1.60"/>
    </edge>
    <edge id="BR" from="center" to="bot_right" priority="-1" spreadType="roadCenter" length="210.00">
        <lane id="BR_0" index="0" speed="13.89" length="210.00" shape="13.59,-4.80 100.00,-4.80"/>
        <lane id="BR_1" index="1" speed="13.89" length="210.00" shape="13.58,-1.60 100.00,-1.60"/>
    </edge>
    <edge id="TC" from="center" to="top_center" priority="-1" length="210.00">
        <lane id="TC_0" index="0" speed="13.89" length="210.00" shape="4.80,14.29 4.80,50.00"/>
        <lane id="TC_1" index="1" speed="13.89" length="210.00" shape="1.60,14.29 1.60,50.00"/>
    </edge>
    <edge id="TL" from="center" to="top_left" priority="-1" length="210.00">
        <lane id="TL_0" index="0" speed="13.89" length="210.00" shape="-4.14,5.24 -99.52,14.78"/>
        <lane id="TL_1" index="1" speed="13.89" length="210.00" shape="-4.45,2.05 -99.84,11.59"/>
    </edge>
    <edge id="TR" from="top_right" to="center" priority="-1" length="210.00">
        <lane id="TR_0" index="0" disallow="pedestrian" speed="13.89" length="210.00" shape="99.20,17.96 13.54,9.39"/>
        <lane id="TR_1" index="1" disallow="pedestrian" speed="13.89" length="210.00" shape="99.52,14.78 13.55,6.18"/>
        <lane id="TR_2" index="2" disallow="pedestrian" speed="13.89" length="210.00" shape="99.84,11.59 13.56,2.96"/>
    </edge>

    <tlLogic id="center" type="static" programID="0" offset="0">
        <phase duration="10" state="GGGrrrrGG"/>
        <phase duration="3"  state="yyyrrrryy"/>
        <phase duration="10" state="rrrGGGGrr"/>
        <phase duration="3"  state="rrryyyyrr"/>
    </tlLogic>

    <junction id="bot_center" type="dead_end" x="0.00" y="-70.00" incLanes="" intLanes="" shape="0.00,-70.00 9.60,-70.00"/>
    <junction id="bot_left" type="dead_end" x="-100.00" y="0.00" incLanes="" intLanes="" shape="-100.00,0.00 -100.00,-6.40"/>
    <junction id="bot_right" type="dead_end" x="100.00" y="0.00" incLanes="BR_0 BR_1" intLanes="" shape="100.00,-6.40 100.00,0.00"/>
    <junction id="center" type="traffic_light" x="0.00" y="0.00" incLanes="TR_0 TR_1 TR_2 BC_0 BC_1 BC_2 BL_0 BL_1" intLanes=":center_0_0 :center_1_0 :center_1_1 :center_3_0
        <request index="0" response="000000000" foes="000110000" cont="0"/>
        <request index="1" response="000000000" foes="001110000" cont="0"/>
        <request index="2" response="000000000" foes="001110000" cont="0"/>
        <request index="3" response="000000000" foes="110000000" cont="0"/>
        <request index="4" response="000000111" foes="110000111" cont="0"/>
        <request index="5" response="000000111" foes="110000111" cont="0"/>
        <request index="6" response="000000110" foes="110000110" cont="0"/>
        <request index="7" response="001111000" foes="001111000" cont="0"/>
        <request index="8" response="001111000" foes="001111000" cont="0"/>
    </junction>
    <junction id="top_center" type="dead_end" x="0.00" y="50.00" incLanes="TC_0 TC_1" intLanes="" shape="6.40,50.00 0.00,50.00"/>
    <junction id="top_left" type="dead_end" x="-100.00" y="10.00" incLanes="TL_0 TL_1" intLanes="" shape="-99.36,16.37 -100.00,10.00"/>
    <junction id="top_right" type="dead_end" x="100.00" y="10.00" incLanes="" intLanes="" shape="100.00,10.00 99.04,19.55"/>

    <connection from="BC" to="BR" fromLane="0" toLane="1" via=":center_3_0" tl="center" linkIndex="3" dir="r" state="O"/>
    <connection from="BC" to="TC" fromLane="1" toLane="0" via=":center_4_0" tl="center" linkIndex="5" dir="s" state="o"/>
    <connection from="BC" to="TC" fromLane="1" toLane="1" via=":center_4_1" tl="center" linkIndex="4" dir="s" state="o"/>
    <connection from="BC" to="TL" fromLane="2" toLane="0" via=":center_6_0" tl="center" linkIndex="6" dir="l" state="o"/>
    <connection from="BL" to="BR" fromLane="0" toLane="0" via=":center_7_0" tl="center" linkIndex="7" dir="s" state="o"/>
    <connection from="BL" to="BR" fromLane="1" toLane="1" via=":center_7_1" tl="center" linkIndex="8" dir="s" state="o"/>
    <connection from="TR" to="TC" fromLane="0" toLane="1" via=":center_0_0" tl="center" linkIndex="0" dir="r" state="O"/>
    <connection from="TR" to="TL" fromLane="1" toLane="0" via=":center_1_0" tl="center" linkIndex="1" dir="s" state="O"/>
    <connection from="TR" to="TL" fromLane="2" toLane="1" via=":center_1_1" tl="center" linkIndex="2" dir="s" state="O"/>

    <connection from=":center_0" to="TC" fromLane="0" toLane="1" dir="r" state="M"/>
    <connection from=":center_1" to="TL" fromLane="0" toLane="0" dir="s" state="M"/>
    <connection from=":center_1" to="TL" fromLane="1" toLane="1" dir="s" state="M"/>
    <connection from=":center_3" to="BR" fromLane="0" toLane="1" dir="r" state="M"/>
    <connection from=":center_4" to="TC" fromLane="0" toLane="0" dir="s" state="M"/>
    <connection from=":center_4" to="TC" fromLane="1" toLane="1" dir="s" state="M"/>
    <connection from=":center_6" to="TL" fromLane="0" toLane="0" dir="l" state="M"/>
    <connection from=":center_7" to="BR" fromLane="0" toLane="0" dir="s" state="M"/>
    <connection from=":center_7" to="BR" fromLane="1" toLane="1" dir="s" state="M"/>

</net>
```

Figura 14. Estrutura do ambiente utilizada.

C. Equação de Bellman

$$V(s) = \max_{a} \left(R(s,a) + \gamma \, V(s') \right) \tag{1}$$

em que:

- $\max_{a}$ representa o maior valor do somatório entre a recompensa e o valor do próximo estado;
- $R(s,a)$ é a recompensa;
- γ indica as penalidades;
- $V(s')$ é a estimativa do valor do próximo estado;

D. Pseudocódigo do algoritmo *Deep Q-learning*

Algorithm 1 Deep Q-Learning

Initialize Replay memory D with capacity N
Initialize Policy network with random weights θ
Initialize Target network with weights $\theta_{target} = \theta$
for $episode = 1, 2, \ldots, M$ **do**
    Reset the environment and observe the starting state
    for $timestep = 1, 2, \ldots, T$ **do**
        Select action using ϵ-greedy policy.
        Observe the next_state, reward and done value.
        Store the data in the replay memory
        Sample Random batch from the the replay memory
        Pass batch of states through the policy network to calculate Q(s,a)
        Pass batch of next_states through the target network to calculate Q'(s',a)
        Calculate TD-target = $reward + \gamma * max_a(Q'(s', a))$
        Calculate Loss function = $TD - target - Q(s, a)$
        Compute policy update by using standard gradient descent or ADAM
        if timestep%target_update_time == 0 **then**
            Update weights of target network
            $\theta_{target} \leftarrow \theta$
        end if
    end for
end for

Figura 15. ***Deep Q-learning pseudocode*** **[Gupta 2020].**

E. Modelo que enfatiza o fluxo do agente utilizado na Seção 3.1

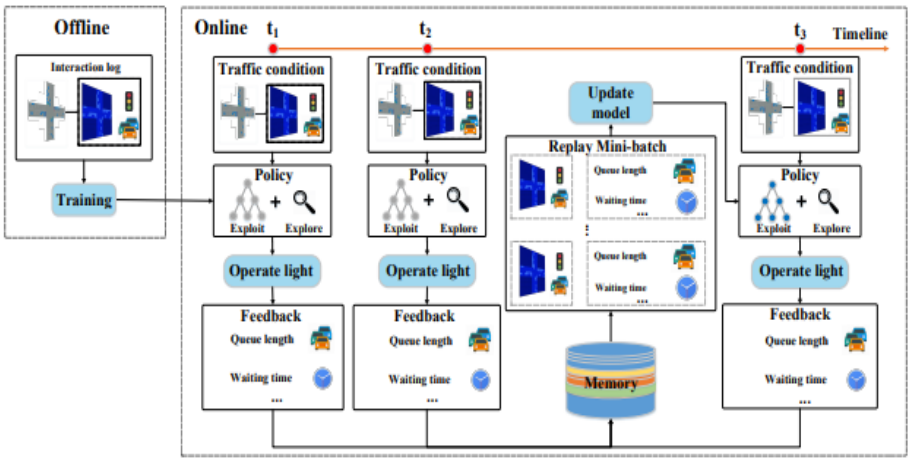

Figura 16. ***Model framework*** **[Wei et al. 2018].**

F. Equação de recompensa da Seção 3.1

$$w_1 * \sum_{i \in l} L_i + w_2 * \sum_{i \in l} D_i + w_3 * \sum_{i \in l} W_i + w_4 * C + w_5 * N + w_6 * T \qquad (2)$$

onde:

- L é a soma do tamanho da fila de uma via juntamente com a velocidade de cada veículo;
- D representa o tempo para a fila se movimentar;
- W é a soma atualizada do tempo de espera na via;
- C indica a troca ou não de sinal do semáforo;
- N é a quantidade de veículos que percorreram a intersecção em um tempo Δt depois da última ação;
- T o tempo total de veículos que percorreram a intersecção em um tempo Δt depois da última ação;

G. Ambiente utilizado na Seção 3.2

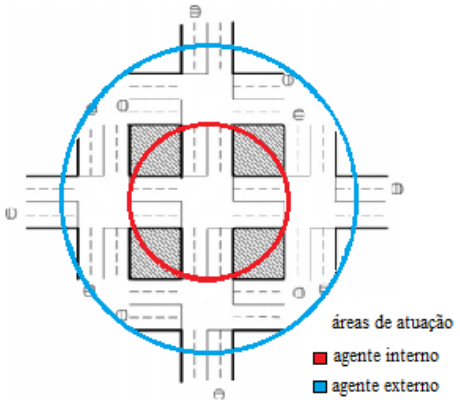

Figura 17. ***Five-intersection, centrally connected vehicular traffic network studied*****. Adaptado de [Arel et al. 2010].**

H. Equação de recompensa da Seção 3.2

$$\delta = \alpha_{s,a} \left\{ r_{s,a} + \gamma_t \cdot \mathbf{MAX} \left[Q_{t-1(s',a')} \right] - Q_{t-1(s,a)} \right\} \tag{3}$$

em que:

- δ é o incremento para ser adicionado ao valor de $\mathbf{Q}_{t-1(s,a)}$ para encontrar $\mathbf{Q}_{t(s,a)}$;
- $\alpha_{s,a}$ representa a taxa do treino no intervalo [0, 1];
- $\mathbf{r}_{s,a}$ é a recompensa do agente por tomar uma ação a no estado do ambiente s;
- $\mathbf{MAX} \left[\mathbf{Q}_{t-1(s',a')} \right]$ indica os valores estimados para recompensas ou $\mathbf{MIN}$ para penalidades;
- γ_t representa as penalidades no intervalo [0, 1];
- $\mathbf{Q}_{t-1(s,a)}$ indica as últimas estimativas de *Q-value*;

I. Equação de recompensa da Seção 3.3

$$r = \frac{D_{last} - D_{current}}{max[D_{last}, D_{current}]} \tag{4}$$

onde:

- D_{last} e $D_{current}$ representam os valores do último e atual tempo total de espera em uma intersecção, respectivamente;

J. Equação para encontrar *Q-values* da Seção 3.3

$$Q(s_t, a_t) \leftarrow Q(s_{t-1}, a_{t-1}) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_{t-1}, a_{t-1}) \right] \quad (5)$$

onde:

- $Q(s_t, a_t)$ é o *Q-value* a ser atualizado;
- a_t representa a ação tomada em um estado s_t, que é seguido pelo estado s_{t+1};
- r_{t+1} é a recompensa;
- α é a porcentagem de aprendizado (valor fixo);
- $\max_a Q(s_{t+1}, a)$ representa a estimativa da recompensa da próxima ação;
- γ indica as penalidades;