CONTROLE DINÂMICO PARA SEMÁFOROS: DESENVOLVIMENTO DE UM SEMÁFORO INTELIGENTE PARA CRUZAMENTOS DE ALTO FLUXO

Guilherme Pallares Bilton – guilhermebilton@hotmail.com
Prof. Dr. Sergio Luis Rabelo de Almeida (Orientador) – prof.sergio@mackenzie.br

RESUMO

Este artigo mostra o desenvolvimento de um sistema de semáforos com sensores, para a redução do trânsito em cruzamentos onde existe um alto fluxo de veículos, quando comparados a sistemas tradicionais, que não consideram a alta variabilidade de fluxo que pode existir a cada instante. Para o aproveitamento dos dados provindos de sensores eletromagnéticos, visando otimizar o instante de mudança de fase do semáforo, buscando o mínimo tempo médio de espera dos veículos, foi treinada uma rede neural utilizando técnicas de *reinforcement learning*, dentro de um simulador capaz de simular diferentes condições de trânsito em diversas localizações. O modelo se mostrou capaz, com resultados cada vez mais promissores durante a etapa de treinamento. Na etapa de validação, a rede neural chegou a superar métodos utilizados tradicionalmente em alguns cenários.

Palavras-chave: Artigo. *Deep Q-Learning*. Python. Engenharia. Semáforos. Trânsito.

DYNAMIC TRAFFIC LIGHT CONTROL: DEVELOPMENT OF AN INTELLIGENT TRAFFIC LIGHT SYSTEM FOR HIGH-FLOW INTERSECTIONS

ABSTRACT

This article shows the development of a traffic light system with sensors, to reduce traffic at intersections where there is a high flow of vehicles, when compared to traditional systems, which do not take into account the high variability of flow that can exist at any given moment. In order to take advantage of the data from electromagnetic sensors, with the goal of changing the traffic light's phase, seeking the minimum average waiting time of the vehicles, a neural network was trained using reinforcement learning techniques, within a simulator capable of simulating different traffic conditions in different locations. The model proved to be capable, with increasingly promising results during the training stage. In the validation stage, the neural network even surpassed methods traditionally used in some scenarios.

Keywords: Article. Deep Q-Learning. Python. Engineering. Traffic Lights. Traffic.

1 INTRODUÇÃO

Historicamente, o transporte sempre foi um dos principais pilares para o desenvolvimento econômico de um país. A construção de caminhos e rotas seguras e eficientes é fundamental para transitar pelos países e cidades. Com o surgimento dos veículos motorizados, houve a necessidade de criar cada vez mais vias pavimentadas e sinalizações que visassem a segurança dos condutores e dos pedestres.

A preocupação com a segurança trouxe um debate muito importante, como a mobilidade urbana, que cada vez mais tem evoluído com o desenvolvimento de novas tecnologias. Principalmente pois na década de 1950 o governo federal brasileiro decidiu incentivar a indústria automotiva com investimentos em rodovias como atalho ao crescimento econômico como cita Fernandes (2013). Como resultado dessa política, hoje cerca de 75% do transporte brasileiro é rodoviário, como cita Labre (2018).

Atualmente existem áreas e órgãos dedicados ao estudo do trânsito, por exemplo a Companhia de Engenharia de Tráfego, uma companhia vinculada ao governo da cidade de São Paulo, que controla e implementa sistemas de trânsito na cidade. Esse órgão é responsável por toda operação e fiscalização do sistema viário da cidade.

Os estudos sobre o setor rodoviário envolvem diversas subcategorias, desde o planejamento das rodovias, até a programação de semáforos em intersecções. Onde visam cada vez mais melhorar a mobilidade urbana, principalmente em grandes cidades como São Paulo. Além da mobilidade atualmente há um grande medo na população, que é o de parar em semáforos vermelhos a noite, com o sistema inteligente será possível reduzir essas paradas indesejadas.

Com o crescimento rápido das metrópoles, a programação otimizada de semáforos, fica cada vez mais importante, uma vez que só em São Paulo, de acordo com a CET existem mais de 6000 interseções com semáforos na cidade. A prefeitura tem um projeto de modernizar 185 cruzamentos da região central da cidade, com isso os equipamentos serão de Led e além de poderem ser controlados remotamente, também poderá fazer a transição de luzes baseado em cálculos momentâneos da quantidade de carros e velocidade dos mesmos na via. Sabendo o fluxo de carros de cada via pode-se determinar mais facilmente quanto tempo será necessário de cada luz sinalizadora do semáforo, melhorando o fluxo de carros, economia de combustível ao acelerar e frear, além de possibilitar uma melhor mobilidade urbana esse sistema faz com que haja menor emissão de poluentes no ar pois os automóveis não terão que frear com tanta frequência como antes, economizando a acelerada após parada no semáforo.

O método utilizado tradicionalmente em semáforos não utiliza sensores, portanto é incapaz de tomar decisões baseados em informações momentâneas, e tem que se basear em dados históricos.

Isto pode ser um problema em condições anormais, e não é o modo mais eficiente até em condições normais.

Este trabalho mostra a implementação de um sistema inteligente por meio de um modelo matemático e uma rede neural. Esse método possibilita saber o tempo e momento exato da execução de mudança de luz no semáforo. Com isso será possível otimizar os principais cruzamentos da cidade.

2 MATERIAIS E MÉTODOS

Para testar o desempenho de diferentes métodos de controle semafórico, foi construído um simulador. A linguagem de programação escolhida foi Python, uma linguagem com programação orientada a objetos, ajudando na etapa de desenvolvimento de simuladores ou jogos, segundo Kajarekar. Pontos negativos incluem a performance, que segundo Zehra, é inferior a outras linguagens como C++, mas, segundo Beklemysheva, a facilidade de implementação, e a disponibilidade de bibliotecas de *machine learning* fez Python ser a linguagem escolhida.

A base do simulador, tanto na parte visual como em interações entre objetos, a biblioteca escolhida foi Pygame, normalmente utilizada para o desenvolvimento de jogos. Como biblioteca para o desenvolvimento das redes neurais, foi utilizado Pytorch, que é de código aberto e sua implementação tem o código fácil de entender, segundo Pointer, o que facilita o processo de desenvolvimento.

Após o desenvolvimento completo do simulador, o mesmo foi utilizado como ambiente para um algoritmo de *Deep Q Learning*. Este algoritmo foi escolhido por ser capaz de alterar os pesos e vieses de uma rede neural de maneira a solucionar problemas complexos. Durante a etapa de treinamento da rede neural, foram coletados dados para analisar se o modelo estava de fato aprendendo com o tempo.

Para realizar a comparação entre os diferentes métodos de controle semafórico, foram primeiro coletados dados do desempenho de cada método, durante a simulação de episódios em dois níveis diferentes, com condições iniciais vereáveis. Os dados foram então plotados em gráficos, e analisados.

3 FUNDAMENTAÇÃO TEÓRICA

3.1 DEEP LEARNING

O *deep learning* é uma subcategoria do aprendizado de máquina, que usa algoritmos inspirados na estrutura do cérebro humano.

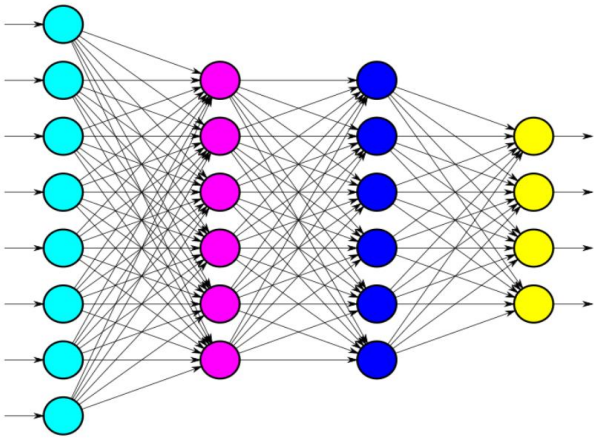
Assim como outros algoritmos de aprendizado de máquina, o modelo de *deep learning* aprende com os dados fornecidos, sem ser programado diretamente.

O componente base do *deep learning* é a rede neural. Uma rede neural artificial é um sistema de computação composto por uma coleção de unidades conectadas chamadas neurônios ou nós, organizadas em camadas, como cita Nielsen (2019).

Existem três categorias de camadas:

- As camadas de input, que recebem os dados;
- As camadas escondidas;
- Camadas de output, que contém um neurônio para cada possível output.

Figura 1 - Rede Neural de 4 camadas

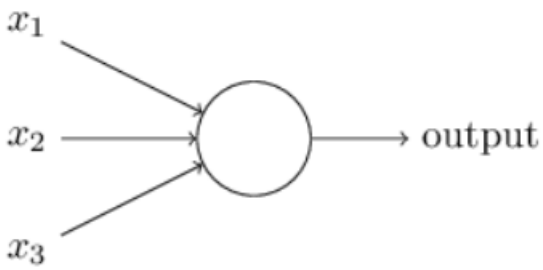


Fonte: DEEPLIZARD, 2018

A Figura 1 é um exemplo de uma rede neural com quatro camadas. A camada de input, em azul-claro, contém 8 neurônios, seguida por duas camadas escondidas (roxo e azul-escuro) com 6 neurônios cada, e uma camada de output em amarelo com 4 neurônios. Neste caso, todos os neurônios da rede neural estão conectados com todos os neurônios da camada vizinha.

Cada neurônio, nada mais é que uma função, que considera todos os inputs que chegam da camada anterior, e passa um output para a camada seguinte.

Figura 2 - Perceptron



Fonte: NIELSEN, 2019

Na figura 2, x1, x2, x3 são inputs que chegam no neurônio. Cada conexão de entre neurônios possui um peso w, e cada neurônio possui um viés b. A fórmula que o neurônio executa é dada pela equação (1):

$$output = wx + b \quad (1)$$

Sendo:

$$wx = \sum_j w_j \, x_j$$

Onde:

- x: valor dos inputs
- w: peso das conexões
- b: viés do neurônio

O output z do neurônio é passado para uma função de ativação para quebrar a linearidade. De acordo com Chen (2021), algumas das funções de ativação mais populares incluem sigmoid, tanh e ReLu.

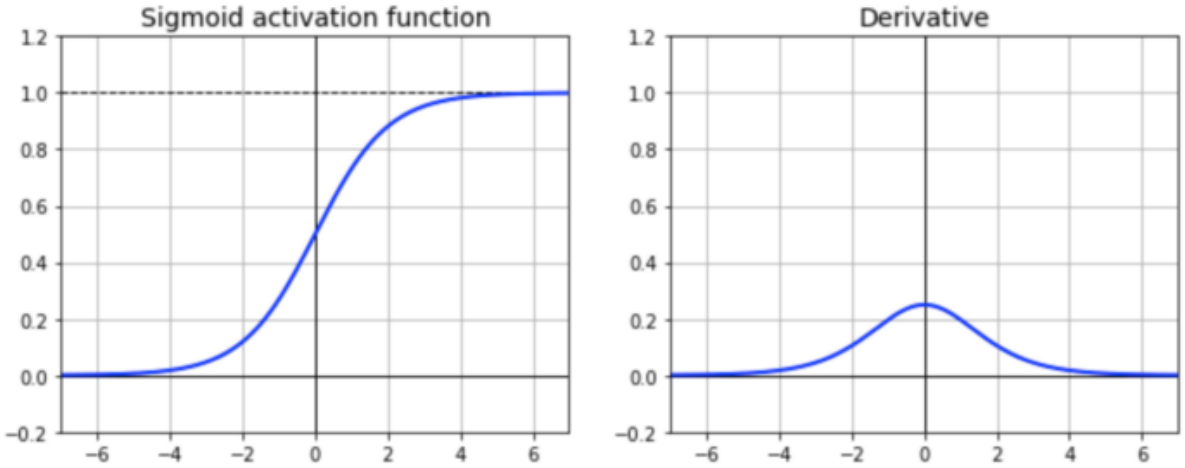
A função sigmoid é dada pela equação (2):

$$\sigma(z) = \frac{1}{(1+e^{-z})} \quad (2)$$

Onde:

- z: output do neurônio

Figura 3 - Comportamento da função Sigmoid.



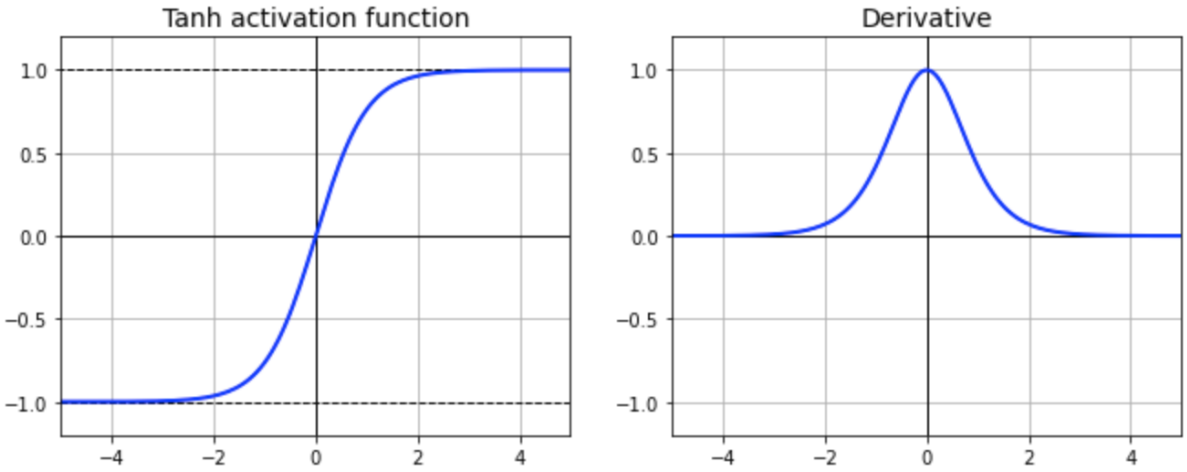
Fonte: CHEN, 2021

A função sigmoid sempre retornará um valor entre 0 e 1, e tem um formato de S, como pode ser visto na figura 3.

A função tanh é dada pela equação (3):

$$tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}} \quad (3)$$

Figura 4 - Gráfico da função tanh



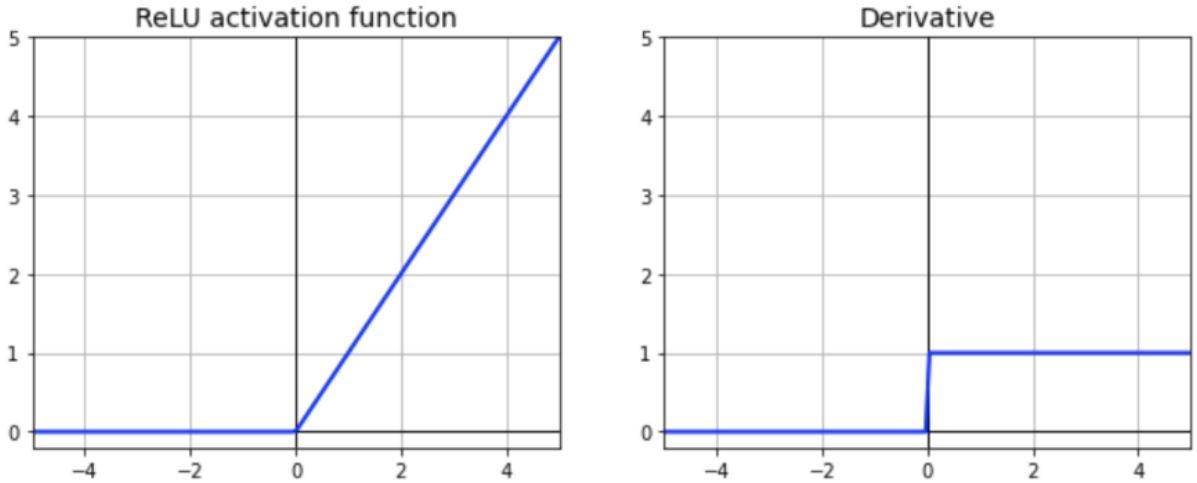
Fonte: CHEN, 2021

Pode ser observado na figura 4 que a função tanh tem um comportamento parecido com a sigmoid, mas ela retorna valores entre -1 e 1.

A função ReLu é dada pelas condições (4):

$$\begin{cases} 0 \; if \; x \leq 0 \\ x \; if \; x > 0 \end{cases} \quad (4)$$

Figura 5 - Função ReLu



Fonte: CHEN, 2021

A função Relu mostrada na figura 5, retorna 0 se o valor for negativo, ou o próprio valor se for positivo.

O resultado então é passado como entrada para os nós da próxima camada. Esse processo continua até que a camada de output seja atingida.

Na camada de output, cada neurônio representa uma categoria, por exemplo, no caso de uma rede neural artificial para identificação de animais, os quatro neurônios poderiam representar categorias como: cachorro, gato, tigre, leão.

Para que a rede neural consiga prever os resultados corretamente, os pesos w das conexões entre neurônios precisam ter valores corretos. Na etapa de treino da rede neural, temos os *labels*, dizendo qual a resposta. Com esta informação, podemos comparar o output da rede neural, com o label e calcular o erro. Normalmente esta comparação é feita utilizando o MSE (*mean squared error*), equação (5):

$$MSE(input) = (output - label)^2 \quad (5)$$

Utilizamos então o SGD (*stochastic gradient descent*), para calcular o gradiente de cada neurônio, visando minimizar o erro obtido pelo MSE.

Os pesos e vieses são atualizados, e o processo repete iterativamente, até que os outputs tenham uma acurácia satisfatória.

3.1.1 *REINFORCEMENT LEARNING*

O *reinforcement learning* (RL) é uma área de aprendizado de máquina que se concentra em treinar um agente, para que suas ações maximizem o retorno (soma de recompensas) ao final de um episódio.

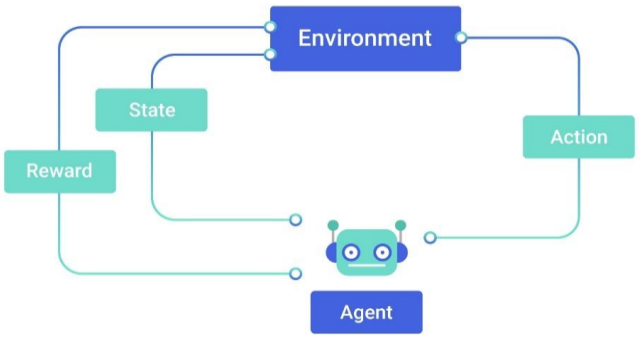
Usando um jogo de xadrez como exemplo de ambiente, o RL está preocupado com como o jogador pode tomar ações, como mover uma peça, a fim de maximizar sua recompensa, no caso ganhar a partida.

Os componentes em um ambiente e aprendizado são:

- Agente
- Ambiente
- Estado
- Ação
- Recompensa

Segundo Li (2018), a ideia básica de um algoritmo de RL é mostrada na figura 6:

Figura 6 - Diagrama do algoritmo de *Reinforcement Learning.*



Fonte: DEMUSH, 2020

Neste caso, um agente recebe uma representação do estado atual do ambiente. A partir desse estado ele toma uma ação. Esta ação tem influência no ambiente. O agente então recebe um novo estado, e uma recompensa pela sua ação. Este processo se repete até o fim do episódio.

3.1.1.1 DEEP Q LEARNING

Q-Learning é uma técnica de RL, que permitirá acharmos uma política que ao ser seguida pelo agente, terá o maior retorno no episódio.

A função Q, é uma função que descreve o "quão valioso" é um determinado par estado-ação. Ou seja, ela mapeia o valor de todas as ações possíveis para todos os estados e atribui um valor Q para cada um desses pares estados-ações. A função Q que representa os pares estado-ação com o maior retorno, é descrita na equação de Bellman (6) como:

$$q_*(s,a) = E[R_{t+1} + \gamma \max q_*(s',a') \quad (6)$$

Onde:

- R: recompensa de cada instante
- γ: desconto de valores futuros
- E: expectativa

O desconto de valores futuros γ serve para que estados mais próximos sejam mais influentes, e a expectativa E, de acordo com Torres, 2020, está na equação para caso o ambiente se comporte de maneira estocástica.

Ou seja, o valor do par estado-ação (s, a), é igual à recompensa do estado que ela gerará somado ao valor da função q* do próximo estado, se tomar uma ação ótima.

Para que a função q seja mapeada, devem ser explorados todos os possíveis estados. Em um ambiente muito complexo, o número de possíveis pares estado-ação pode ser tão grande que ficaria inviável encontrar uma função q*. O *Deep Q-Learning* utiliza uma rede neural, atualizada com a rede de política, para estimar esta função q*.

3.2 PROGRAMAÇÃO TRADICIONAL DE SEMÁFOROS

De acordo com Viana (2019), atualmente, calcula-se o tempo do que um semáforo fica na fase verde pela equação (7):

$$T_{v,real,i} = T_{v,ef,i} + T_{percepção} - T_a \quad (7)$$

Onde:

- $T_{v,real,i}$: é o tempo de verde real para cada estágio;
- $T_{v,ef,i}$: é o tempo de verde efetivo para cada estágio;
- $T_{percepção}$: é o tempo de percepção do condutor (s), que geralmente leva 2s;
- T_a: é o tempo de amarelo (s), é geralmente entre 2 e 3s;

Figura 7 - Tabela de tempo para a fase amarela.
Fonte: PERMIT, 2015

Velocidade da via (km/h)	Tempo amarelo (s)
<56	3
56-64	3.5
64-72	4
72-80	4.5
>80	5

Nota: Em intersecções largas ou onde a velocidade é muito alta o tempo amarelo pode ser aumentado.
Fonte: PERMIT, 2015

O tempo da fase amarela pode ser obtido na figura 7, que varia com a velocidade máxima permitida da via.

O tempo verde efetivo é dado pela equação (8):

$$T_{v,ef,i} = yi * \left(\frac{T_{co} - T_p}{Y}\right) \quad (8)$$

Onde:

- $T_{v,ef,i}$: é o tempo de verde efetivo para cada estágio;
- yi*: é a taxa máxima para cada estágio;
- T_{co}: é o tempo de ciclo ótimo (s);
- T_p: é o tempo perdido (s);
- Y: é a ocupação da interseção.

O tempo de ciclo ótimo é dado pela equação (9):

$$T_{co} = \frac{1{,}5 * T_p + 5}{1 - Y} \quad (9)$$

Onde:

- Tp: Tempo perdido

O tempo perdido é dado pela equação (10):

$$T_p = \sum T_{vermelho,total} + \sum T_{percepção} \quad (10)$$

A taxa de ocupação é dada pela equação (11):

$$Y = \sum yi^* \quad (11)$$

Sendo:

$$yi^* = yi, máx$$

Onde:

- yi,máx: é a taxa de ocupação máxima entres as aproximações de um estágio;

A taxa de ocupação pode ser dada pela equação (12):

$$y_i = \frac{q_i}{s_i} \quad (12)$$

Onde:

- qi: é o fluxo na aproximação (veículo/h);
- si: é o fluxo de saturação (veículo/h).

O fluxo de saturação pode ser dado pela equação (13):

$$s = 525L \quad (13)$$

Onde:

- L: largura da intersecção em metros.

3.3 SENSORES

Para captar o estado atual do ambiente (intersecção), será necessário o uso de sensores. Estes sensores podem trazer informações como a quantidade de carros esperando no sinal vermelho, o fluxo de carros em determinado tempo e a velocidade média entre os veículos ao passarem pela intersecção.

Para a escolha entre os tipos de sensores, serão considerados preço, facilidade de implementação, efetividade, resiliência.

O radar de velocidade do tipo fixo, também conhecido como Pardal, é composto por dois principais elementos: a câmera, e o sensor eletromagnético. A câmera é utilizada para a identificação da placa do veículo e a comprovação em caso de infração.

O sensor eletromagnético, segundo Mattede (2014), é composto normalmente por dois ou três laços indutivos, inseridos no asfalto, utilizados para calcular a velocidade do veículo. Quando um objeto metálico passa por cima dos laços indutivos, ocorre uma perturbação no campo magnético. Como a distância entre os laços é conhecida, é possível calcular a velocidade que os veículos trafegam no momento que passam sobre o sensor, baseado na diferença de tempo entre as perturbações dos diferentes campos magnéticos gerados pelos laços indutivos.

Figura 8 - Perturbações no campo magnético para diferentes tipos de veículos.

Fonte: NICHIMOTO, 2011

Com este tipo de sensor, é possível distinguir entre diferentes tipos de veículos, já que seus perfis eletromagnéticos diferem, como pode ser observado na figura 8.

Isto pode ser uma informação útil quando passada no estado para a rede neural: ela pode fazer adaptações como relacionar o tipo de veículo com sua velocidade e então ter mais informações para a tomada de decisões.

4 RESULTADOS E DISCUSSÃO

Foi construído um simulador em Python, para poder ser testada e comparada a performance de três métodos distintos de controle para semáforos. Os principais componentes do ambiente de simulação são um mapa, veículos e semáforos.

Figura 9 – Ambiente simulado com 1 cruzamento

Figura 10 – Ambiente simulado com 2 cruzamentos

O mapa do simulador consiste em imagens contendo cruzamentos localizados na região de Higienópolis, São Paulo.

As figuras 9 e 10 são imagens tiradas do simulador. Cada círculo azul representa um veículo, e em cada cruzamento existe um par de semáforos representados por pequenos círculos que podem estar nas fases vermelha, amarela ou verde. Cada nível possui algumas características diferentes, que podem influenciar no resultado quando medindo a performance dos algoritmos. A figura 1 representa um local no mapa onde há somente um cruzamento, e será chamado de nível 1. Existem duas

trajetórias neste nível, sendo uma horizontal, onde os veículos percorrem da esquerda para a direita, e uma trajetória na vertical onde os veículos percorrem de cima para baixo. A figura 2 mostra outro local onde existem dois cruzamentos, chamaremos de nível 2. Existem três trajetórias neste nível, sendo uma delas horizontal, onde os veículos percorrem seu trajeto da esquerda para a direita, passando por dois cruzamentos. Como os dois cruzamentos estão em série na via horizontal, o acúmulo de veículos no cruzamento mais à direita pode influenciar o comportamento do cruzamento da esquerda. Os outros dois trajetos na vertical possuem um cruzamento cada, sendo que o trajeto mais à esquerda tem o sentido de cima para baixo, e o trajeto mais à direita tem o sentido de baixo para cima.

As imagens-base foram tiradas do Google Maps, que possui uma ferramenta de medição de distância, possibilitando a conversão de metros para píxeis. No simulador, temos que 1m equivale a 4,31 píxeis. A taxa de atualização do simulador é de 60 quadros por segundo. Com estas informações, podemos simular constantes como velocidade máxima da via, acelerações, desacelerações e tamanho dos veículos, para ficar tudo em "escala", possibilitando uma representação que se aproxima da realidade.

Cada veículo foi programado para seguir um trajeto pré-definida, onde devem passar por ao menos um cruzamento com semáforo. Eles atuam independentemente e possuem sensores para identificar outros veículos ou semáforos a frente, simulando o comportamento humano. As suas acelerações, velocidade de rotação e velocidade máxima foram calibradas para uma aparência realística.

Os semáforos são objetos posicionados nos cruzamentos, para redirecionar o fluxo de veículos. Eles possuem duas fases, verde e vermelha, e uma fase transitória amarela. A mudança de fase sempre é efetuada na sequência: verde, amarelo, vermelha, repete. O tempo na fase de transição (amarela), foi retirada de uma tabela para vias com velocidade máxima de 50km/h e equivale a 3 segundos ou 180 quadros. Cada semáforo possui sensores que conseguem identificar as características de veículos que estão se aproximando.

Um episódio no simulador resume-se a algumas etapas. Para cada trajetória no mapa, são gerados veículos de maneira pseudoaleatória, variando a quantidade de veículos e distância inicial entre eles. Cada veículo parte de seu ponto de partida em direção ao destino de sua trajetória. Quando todos os veículos alcançarem o seu destino, o episódio termina sendo gerados os dados para análise.

O parâmetro para julgar a performance dos modelos consiste em comparar a média do tempo gasto pelos veículos desde a partida até o seu destino.

Visando comparar a performance entre diferentes métodos, foram programados três algoritmos para ditarem o momento de mudança de fase dos semáforos.

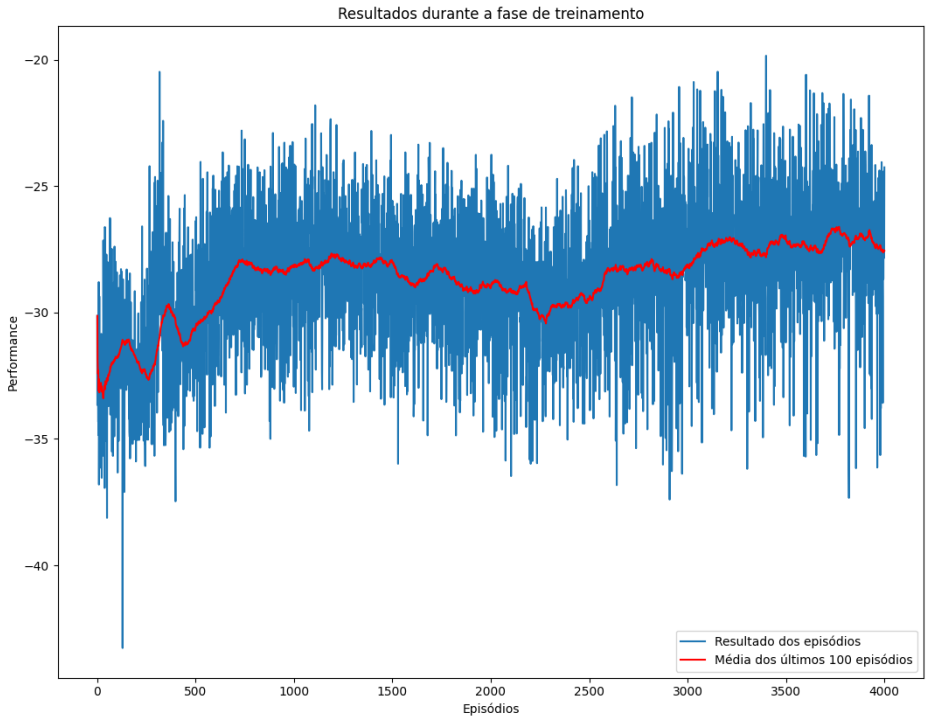
O primeiro método utiliza valores de uma curva normal com média 50 e desvio padrão 10. Se o valor selecionado for maior que 75 o semáforo inicia a mudança de fase. Chamaremos este método de método aleatório.

O segundo método, é o mais utilizado atualmente em semáforos e utiliza valores como o fluxo médio de veículos na via para determinar a duração de cada fase. O seu funcionamento será explicado em detalhes na secção de Fundamentação Teórica. Chamaremos este método de método tradicional.

O terceiro método utiliza uma rede neural para determinar o instante de mudança de fases. Para treinar esta rede neural, foi utilizado um algoritmo de *Deep Q-Learning*, implementado junto ao simulador. Os parâmetros da rede neural foram determinados experimentalmente. As redes neurais consistem em uma camada de input com 4 neurônios, três camadas escondidas com 100 neurônios cada, uma camada de saída com 2 neurônios. Todas as camadas são completamente conectadas e utilizam *tanh* como função de ativação em todos os níveis. A mudança de fase é determinada pelo maior valor entre os dois neurônios na camada de saída (um representa a ação de iniciar a mudança de fase, o outro representa manter o semáforo na fase atual). Chamaremos este método de método inteligente.

Para coletar os resultados do modelo inteligente, a rede neural precisa passar por uma fase de treinamento para atualizar os seus parâmetros. O modelo foi treinado no nível 1, onde existe apenas um semáforo, por aproximadamente 4000 episódios. O processo durou cerca de 24 horas e por motivos de desempenho, foram utilizados apenas 5 veículos por trajetória. Durante processo de treinamento, foram coletados os seguintes dados mostrados na figura 11 a seguir:

Figura 11 – Treinamento da rede neural



O gráfico da figura 11 acima mostra a performance alcançada pelo modelo durante a fase de treinamento. A performance do modelo é dada pela equação (14):

$$P = -1 * \frac{\sum_i^n T_i}{n} \quad (14)$$

Onde:

- P: Performance
- T: Tempo para um veículo completar a sua trajetória
- n: Número de veículos no episódio

Portanto um número maior indica previsões melhores pelo modelo.

Para facilitar a visualização da melhoria na performance, podemos olhar para a linha vermelha, que representa a performance média dos últimos 100 episódios. A performance começa próximo dos -32 e segue subindo com o passar dos episódios, tendo alguns momentos de instabilidade, sendo normal para um algoritmo de *deep reinforcement learning* e, eventualmente, após cerca de 4000 episódios, atinge uma faixa próxima dos -27.

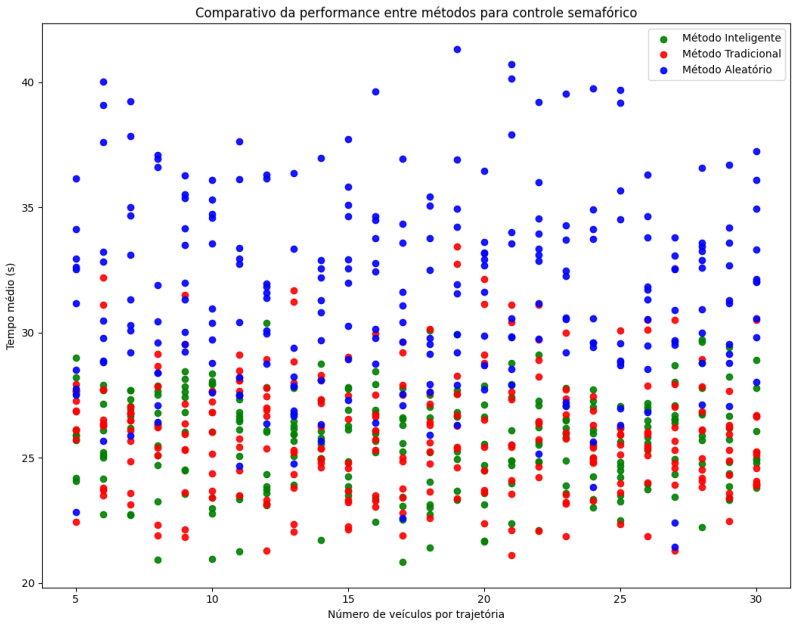
Com o passar dos episódios, o modelo convergiu e performou de maneira superior quando comparado com o início do treinamento.

Para a coleta de dados referentes aos resultados, foram simulados 780 episódios no simulador. Cada episódio novo teria valores diferentes para as seguintes variáveis:

- O nível pode variar entre nível 1 ou nível 2;
- O número de veículos por trajetória variou de 5 a 30, podendo assumir qualquer valor inteiro entre e incluindo os dois;
- A posição inicial de cada veículo varia de forma aleatória;
- O método para controle das fases dos semáforos, poderia ser um dos três: método aleatório, o método tradicional ou o método inteligente;

No final de cada episódio, foi coletado o tempo médio que os veículos do episódio levaram para completar suas respectivas trajetórias.

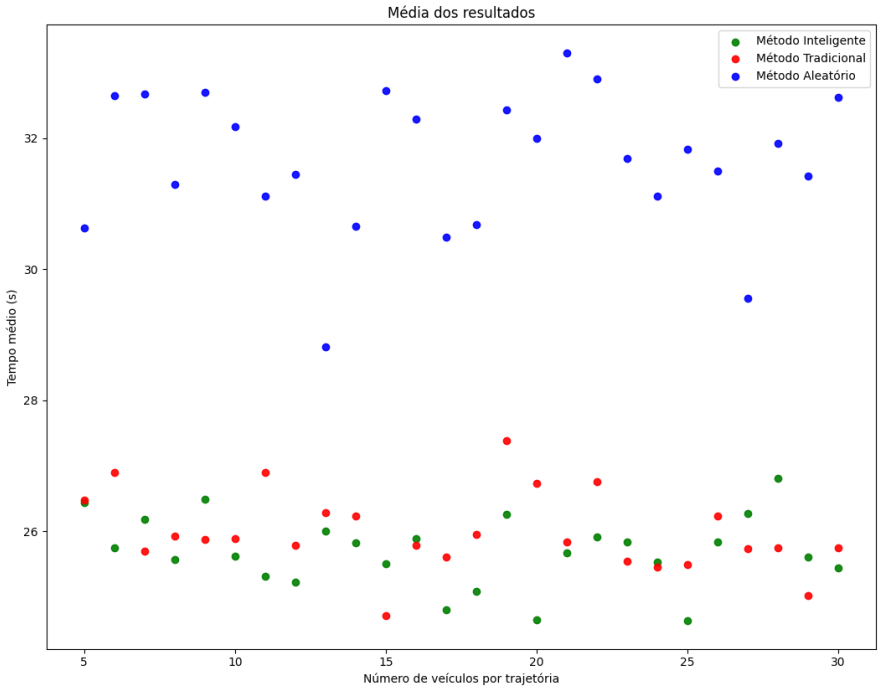
Figura 12 – Dados coletados



No gráfico da figura 12, podem ser observados todos os dados coletados, comparando-se o tempo médio de cada método, ao variar o número de veículos.

Um tempo médio menor corresponde a um modelo mais eficiente, pois mostra que a fase dos semáforos foi alterada de maneira que os veículos levaram em média menos tempo para completarem seus trajetos.

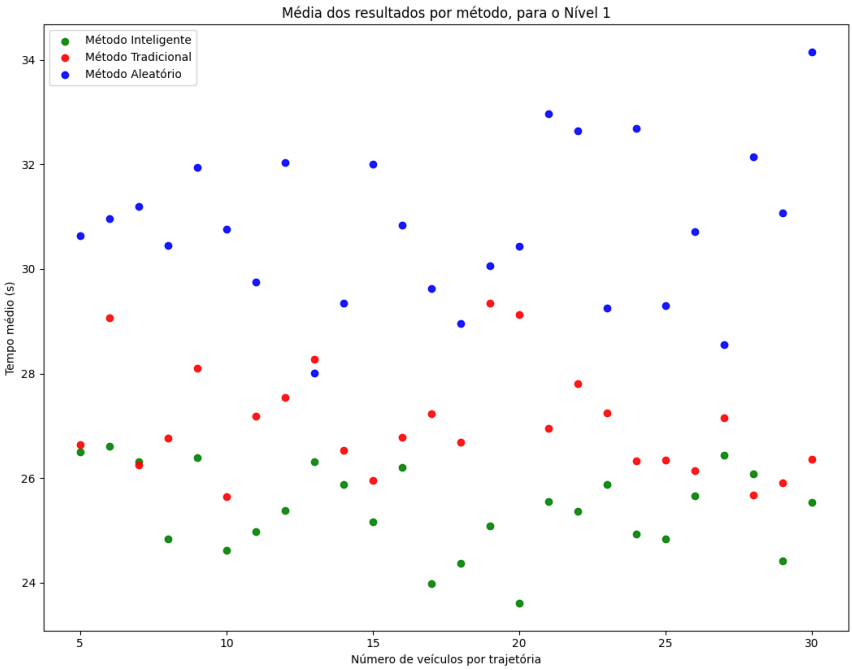
Figura 13 – Visualização simplificada dos dados



Cada ponto no gráfico da figura 13 acima, representa uma média dos tempos médios por número de veículos, separados por método, facilitando a visualização.

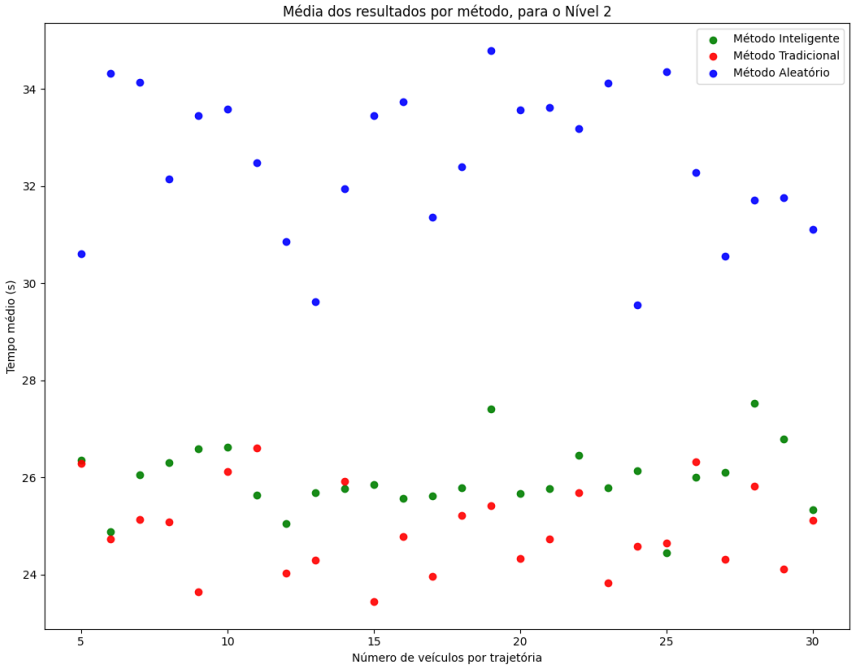
Os pontos em azul estão agrupados na parte superior, enquanto os pontos verdes e vermelhos estão agrupados na parte inferior, indicando que o método aleatório teve uma performance pior que os demais modelos.

Figura 14 – Visualização simplificada dos dados no nível 1



Na figura 14, os dados coletados estão separados apenas para o nível 1. O método inteligente, em verde, apesar de ter sido treinado apenas com 5 veículos por trajetória, teve a melhor performance mesmo em situações com mais veículos. O método tradicional teve o segundo melhor resultado e por último o método aleatório.

Figura 15 – Visualização simplificada dos dados no nível 2



Os resultados para o nível 2 na figura 15, mostra que o método inteligente teve uma performance melhor que o método aleatório, mas não conseguiu superar o método tradicional. Considerando que o modelo AI nunca havia visto o nível 2, onde existem diferenças nas dimensões e número das trajetórias, além de mais cruzamentos quando comparado com o nível 1, o fato de superar o método aleatório, mostra que durante a fase de treinamento, ocorreu algum tipo de generalização, ou seja, aprendeu um comportamento que pode ser benéfico para diferentes contextos. Isto é um ótimo sinal e indica que modelos deste tipo tem o potencial para serem aplicados em diversas situações. Considerando que a performance foi próxima a do método, até o superando para alguns números de veículos, se o modelo fosse treinado por mais tempo e/ou em mais condições, teria grandes chances de superar a performance do método tradicional em qualquer circunstância.

6 CONSIDERAÇÕES FINAIS

Este trabalho explorou a possibilidade de avanços na eficiência de semáforos com a utilização de sensores e técnicas de *deep reinforcement learning.*

A partir da análise dos resultados, foi possível concluir que a técnica de *deep Q-learning* tem potencial para ser aplicada em sistemas de semáforos com sensores. No nível 1, onde o ambiente foi o mesmo do treinamento, o método inteligente superou o método tradicional. Já no nível 2, onde o modelo nunca havia visto aquele cenário, o método inteligente superou o método aleatório, mas não chegou na performance do método tradicional.

Devido à abundância de parâmetros e decisões que devem ser tomadas para o aprendizado da rede neural, e considerando o longo período que leva para sinais de aprendizado e convergência do modelo, a performance final foi limitada pelo hardware em relação às condições simuladas na etapa de treinamento. Isto resultou em uma rede neural que, apesar de performar bem e superar expectativas, indica que existe um potencial a ser explorado, com sistemas mais complexos e com mais variedades.

A técnica de *deep Q-learning* não é a única que pode ser utilizada neste tipo de problema, e outras como *Proximal Policy Optimization, segundo* Schulman, podem ser exploradas visando reduzir a ineficiência dos semáforos em situações de alto fluxo.

Como continuação deste projeto, podem ser exploradas maneiras de integrar os métodos utilizados neste artigo em sistemas reais. Como ponto de partida e sugestão, podem ser utilizados alguns dispositivos citados na secção de fundamentação teórica, como sensores eletromagnéticos. Os dados provindos dos sensores eletromagnéticos terão que ser filtrados e manipulados para serem compatíveis com os dados simulados, utilizado pela rede neural.

Para mais detalhes sobre o projeto, o código pode ser encontrado em https://github.com/gbilton/TCC.

REFERÊNCIAS

FERNANDES, Fernanda Basilio. **O PLANO DE METAS E A QUESTÃO DO TRANSPORTE RODOVIÁRIO NO BRASIL**. 2013. 87 f. TCC (Graduação) - Curso de Curso de História, Faculdades Integradas de Taquara Curso de História, Taquara, 2013.

VIANA, Dandara. **Programação semafórica: entenda como funciona. entenda como funciona**. 2019. Disponível em: https://www.guiadaengenharia.com/programacao-semaforo/. Acesso em: 19 nov. 2021.

CHEN, Bindi. **7 popular activation functions you should know in Deep Learning and how to use them with Keras and TensorFlow 2**. 2021. Disponível em: https://towardsdatascience.com/7-popular-activation-functions-you-should-know-in-deep-learning-and-how-to-use-them-with-keras-and-27b4d838dfe6. Acesso em: 20 out. 2021.

LABRE, Eduardo. **Conheça os 5 principais modais de transporte no Brasil**. 2018. Disponível em: https://simplificafretes.com.br/conheca-os-5-principais-modais-de-transporte-no-brasil/. Acesso em: 15 nov. 2021.

NIELSEN, Michael. **Neural Networks and Deep Learning**. 2019. Disponível em: http://neuralnetworksanddeeplearning.com/. Acesso em: 10 nov. 2021.

MATTEDE, Henrique. **Como funciona um radar de velocidade?** 2014. Disponível em: https://www.mundodaeletrica.com.br/como-funciona-um-radar-de-velocidade/. Acesso em: 24 nov. 2021.

LI, Yuxi. **Deep Reinforcement Learning**. arXiv:1810.06339v1 [cs.LG]. 2018. Disponível em: https://arxiv.org/pdf/1810.06339.pdf. Acesso em: 27 mai. 2022.

SCHULMAN, John; WOLSKY, Filip, DHARIWAL, Prafulla; RADFORD, Alec; KLIMOV, Oleg. **Proximal Policy Optimization Algorithms**. arXiv:1707.06347v2 [cs.LG]. 2017. Disponível em: https://arxiv.org/pdf/1707.06347.pdf. Acesso em: 27 mai. 2022.

ZEHRA, Farzeen; DARKHSHAN; JAVED, Maha, PASHA, Maria. **Comparative Analysis of C++ and Python in Terms of Memory and Time.** Department of Software Engineering, NED University of Engineering and Technology, Karachi, Pakisan, 2020.

BEKLEMYSHEVA, Angel. **Why Use Python for AI and Machine Learning?.** 2021. Disponível em: https://steelkiwi.com/blog/python-for-ai-and-machine-learning/. Acesso em: 27 mai. 2022.

POINTER, IAN. **5 reasons to choose Pytorch for deep learning.** 2020. Disponível em: https://www.infoworld.com/article/3528780/5-reasons-to-choose-pytorch-for-deep-learning.html. Acesso em: 27 mai. 2022.

KAJAREKAR, Varad. **Object oriented programming in Game development.** 2021. Disponível em: https://varad-kajarekar19.medium.com/object-oriented-programming-in-game-development-1293e6ebed45. Acesso em: 27 mai. 2022.

AGRADECIMENTOS

Ao meu gestor, Caio Carvalho, que sugeriu possíveis soluções para o problema dos hardwares, e disponibilizou tempo para a conclusão deste trabalho.

Ao professor Alexandre Lasthaus, pelo esclarecimento de ideias para a implementação do simulador.