

Aplicando rede neural artificial para tomada de decisões

1st Douglas Prado dos Santos
Instituto Federal Catarinense
Campus Videira
Videira, SC
tridops@gmail.com

I. INTRODUÇÃO

Todos os animais, inclusive o humano, obtêm informações sobre o seu entorno através de receptores sensoriais. A informação conseguida pelos receptores se transforma no encéfalo em percepções ou ordens de movimento. [1]

A tecnologia das Redes Neurais Artificiais (RNA's), baseia-se no funcionamento do cérebro humano para solucionar problemas de reconhecimento de padrões que geralmente são baseados em um conjunto de informações previamente conhecido. [2]

Já o algoritmo genético tem o objetivo inicial de estudar os fenômenos relacionados à adaptação das espécies e da seleção natural que ocorre na natureza (Darwin 1859), bem como desenvolver uma maneira de incorporar estes conceitos aos computadores (Mitchell 1997). [3]

Portanto, será utilizado neste trabalho, uma rede neural artificial (RNA) e algoritmos genéticos (GE) que consiga aprender com seus erros e acertos ao longo de gerações. Com o intuito de tomar decisões em ações do teclado, de acordo com pixels na tela, tais pixels serão os inputs transmitidos a IA, e as ações serão os outputs, onde terão controle do teclado.

A ideia central, é criar um identificador de pixels da tela do monitor, e de acordo com a distância, ou altura de um referencial, fazer com que a IA escolha a melhor ação a ser tomada.

II. METODOLOGIA

A. Definição do escopo

Antes de entrar na criação e treinamento da IA, é preciso deixar claro qual será a sua área de atuação.

A IA está sendo desenvolvida para conseguir, através de pixels na tela tomar uma decisão. Porém, é um escopo muito abrangente, onde não seria possível resolver nenhum problema com precisão. Portanto, foi escolhido um problema específico: Fazer com que a IA consiga tomar decisões no jogo do dinossauro do Google Chrome.

O jogo é bem simples, um tradicional jogo de corrida infinita no qual seu objetivo é evitar obstáculos pelo caminho enquanto ganha pontos pelo tempo que sobrevive. A barra de espaço, ou tecla para cima, pula e a seta para baixo faz o dinossauro se abaixar.

Então, será observado os obstáculos na tela, pegando os pixels x e y de cada obstáculo. Com esses dados em mãos

treinaremos a IA para que seus outputs sejam as ações de pular, ou abaixar.

B. Detecção de pixels

Para dar início ao treinamento da IA, o primeiro passo foi definir bem quais serão as entradas utilizadas por ela.

Então, foi desenvolvido um detector de pixels básico, utilizando a biblioteca Pillow do python. Tal detector, foi desenvolvido para conseguir gerar as entradas da IA, com ele é possível ficar monitorando a tela, em um range definido, e identificar a posição dos obstáculos, pegando assim seu eixo X e Y. A imagem abaixo ilustrará melhor.

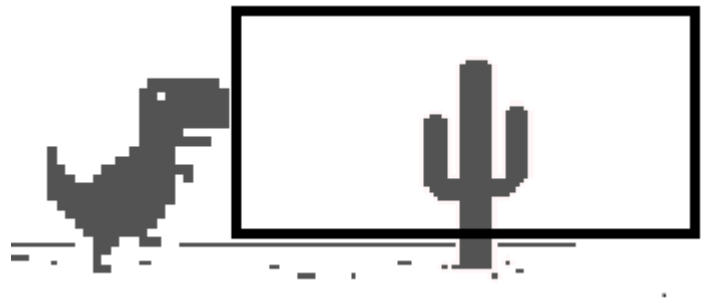


Fig. 1. Example of a figure caption.

Na ilustração acima, o quadrado preto é onde o detector atua, tornando-se a "visão" da rede neural. Qualquer troca de cor que ocorrer dentro dessa área, será enviada como input para a rede neural.

O funcionamento do detector foi definido em algumas regras básicas, para não torná-lo muito pesado, mas que funcione bem.

- Definir qual será o range de atuação, para não ficar analisando a tela inteira.
- Deixar o mais simples possível, a única funcionalidade dele é pegar a alteração da cor do pixel e retornar onde isso aconteceu.

O código ficou dividido em 3 principais funções, na primeira foi a captura da tela, utilizando a biblioteca do Pillow.

basicamente, foi definido a função de captura de tela, e gravado em uma variável que será utilizado posteriormente na função principal.

```
def capture_screen():
    screen = ImageGrab.grab()
    return screen
```

Fig. 2. Example of a figure caption.

A segunda parte do código, ficou a delimitação de observação do algoritmo.

```
X = 150
X2 = 450
Y1 = 390
Y2 = 470
```

Fig. 3. Example of a figure caption.

Na imagem, mostra as variáveis globais criadas, e definidas com os valores dos pixels x e y da área do monitor onde será analisado. Para saber exatamente os pixels da tela, é preciso de alguma ferramenta de imagem, para este exemplo, foi utilizado o Paint do windows.

A terceira e última parte do algoritmo, é onde a mágica acontece. Após pegar tela, e o range que será observado.

```
def detect_enemy(screen):
    aux_color = screen.getpixel((int(X), Y1))
    for x in range(int(X), int(X2)):
        for y in range(Y1, Y2):
            color = screen.getpixel((x, y))
            if color != aux_color:
                return (x - 150, 1)
            else:
                aux_color = color
    return (999, 0)
```

Fig. 4. Example of a figure caption.

Na primeira linha, está pegando a cor atual do pixel, para posteriormente seja possível ver se foi alterado ou não. Seguindo o código, foi aberto 2 laços de repetição (for), que é exatamente onde está sendo definido o range X e Y, em seguida pega-se novamente a cor do pixel e dentro do if analisa-se se essa cor obtida corresponde a cor obtida anteriormente. Caso não corresponda, significa que um obstáculo entrou no caminho e retorna a posição em que isso ocorreu para a IA. Caso contrário, não houve alteração, então é enviado 999, um valor genérico utilizado para que a IA entenda que quando isso ocorrer ela tome a decisão de não fazer ação.

C. Utilizando o NEAT

O NEAT é um método desenvolvido por Kenneth O. Stanley para a evolução de redes neurais arbitrárias e está disponível como uma biblioteca no python. [4]

O Neat é um algoritmo genético que visa tirar vantagem de uma rede por meio de sua estrutura, visando diminuir o

tamanho da pesquisa dos pesos das conexões de uma rede neural. Caso a estrutura evolua para que a topologia da rede seja minimizada, e cresça ao longo do tempo, ganhos significativos em velocidade são apresentados (STANLEY; MIIKKULAINEN, 2002, apud SODER, 2018).

NEAT é designado para que os genes possam ser colocados em ordem quando há cruzamento. Genomas são representações de conectividade da rede. Cada genoma possui uma lista de genes de conexão, cada um referenciando a dois genes de nó conectados. Genes de nó fornecem uma lista de entradas, contendo nós escondidos, e de saídas que podem ser conectadas. Cada gene de conexão possui um nó de entrada, um nó de saída, seus respectivos pesos, a habilidade de estar ou não ativado e um número de inovação, que permite encontrar os genes correspondentes (STANLEY; MIIKKULAINEN, 2002, apud SODER, 2018).

Em tarefas de controle simples, o algoritmo NEAT geralmente chega a redes eficazes mais rapidamente do que outras técnicas neuroevolucionárias contemporâneas e métodos de aprendizado por reforço [4]. E será através dele que será utilizado a rede neural. Antes de começar a utilizá-lo, primeiro foi criado um arquivos de configurações, para facilitar o configuração das gerações, populações, entradas, saídas, entre outros parâmetros.

Abaixo segue a lista de dados de configuração e os parâmetros configurados que foram utilizados para a realização da IA.

Arquivo de configuração:

- generation = 1000
- pop-size = 2
- fitness-criterion = max
- reset-on-extinction = False
- num-inputs = 2
- num-outputs = 2
- response-max-value = 30.0
- response-min-value = -30.0
- species-elitism = 1
- elitism = 1
- max-stagnation = 20
- enabled-mutate-rate = 0.01
- conn-add-prob = 0.5
- conn-delete-prob = 0.5
- survival-threshold = 0.2

Após toda a configuração necessária, o fitness foi definido para corresponder a quantidade de tempo que o personagem permanece vivo. A cada segundo vivo, recebe +0.1 no fitness.

```
ge[0].fitness += 0.1
screen = capture_screen()
output = nets[0].activate(detect_enemy(screen))
```

Fig. 5. Example of a figure caption.

Na imagem acima, mostra a adição do fitness a cada ciclo do while, e também passa os inputs gerados pelo detector, para

a IA. Que neste caso são 2 inputs, a existência de obstáculo e a distância dele.

Assim como a entrada, a IA retorna 2 saídas, que vai de 0 a 1, e dependendo da saída o personagem realiza uma ação (pular ou agachar).

```
if (output[0] > 0.5):  
    jump()  
if (output[1] > 0.5):  
    down()
```

Fig. 6. Example of a figure caption.

Este processo se repete até que todas as gerações sejam concluídas, porém para reiniciar o jogo após a morte do personagem e colocar a nova geração para rodar, foi preciso desenvolver uma outra função, que identifica o "Game Over" na tela e pressionar o "space" do teclado para reiniciar

```
def morte(screen):  
    pyautogui.press(["space"])  
    return screen.getpixel((890, 290)) == (83,83,83)
```

Fig. 7. Example of a figure caption.

III. RESULTADOS

OBS.: Todos os resultados foram tirados com 500 gerações, seguindo as configurações apresentadas na metodologia.

Os resultados obtidos após as 500 gerações não foram satisfatórios, e existem algumas hipóteses para o ocorrido. Primeiro vamos aos valores.

O melhor indivíduo das 500 gerações, alcançou um score de 138, porém foi na sorte.

00138

Fig. 8. Example of a figure caption.

As hipóteses levantadas, para o fracasso da IA são as seguintes:

- **Baixa população** - Pelo motivo de só ter um jogador, não é possível colocar milhares de indivíduos simultaneamente para "brigarem", e selecionar os melhores para a próxima geração.
- **Baixo números de entradas** - Ainda pelo motivo de ser um sistema aberto, sem o controle de todas variáveis, não foi possível pegar todas as entradas necessárias para o treinamento da IA. Entradas como velocidade, que é essencial para o treinamento da IA ficaram de fora por não conseguir tal valor.

A. Buscando solução

O solução encontrada para testar as hipóteses, e resolver os problemas. Foi de criar um ambiente fechado, onde tenho controle sobre todas as variáveis.

Portanto, foi retirada a coleta dos dados através do detector, e foi pego direto do jogo. Para isso ser possível, foi feito o download de uma réplica do jogo original feita pelo Ethan (Pois o jogo original não está disponível para download). Com isso, é possível pegar exatamente a distância, velocidade, largura e altura, tanto dos obstáculos, quanto do jogador.

Com o jogo em mãos, não só as entradas foram resolvidas, mas também a possibilidade de colocar vários indivíduos simultaneamente para "brigarem".



Fig. 9. Example of a figure caption.

Com todas essas mudanças, rapidamente foi obtido uma IA que consiga jogar com maestria o jogo. Foi definido um score a ser atingido de 1000, foram necessárias 14 gerações com 100 indivíduos para tal marca fosse alcançada.

Gen: 14

Fig. 10. Example of a figure caption.

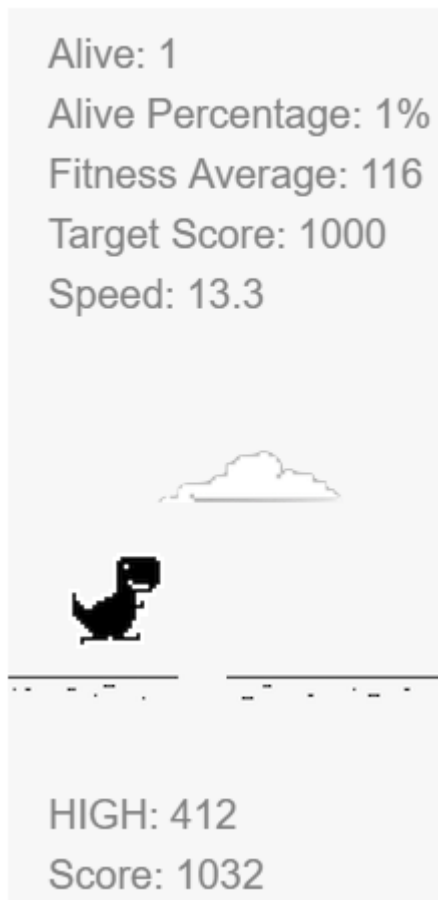


Fig. 11. Example of a figure caption.

IV. CONCLUSÃO

As dificuldades encontradas, não possibilitou a execução da ideia inicial proposta. As variáveis, e a falta de concorrência fez com que a IA não evoluísse, tornando-a em um aleatório entre pular e agachar.

Contudo, revelou-se a diferença entre as dificuldades encontradas em um sistema fechado e um sistema aberto, mostrando na prática como que a ausência de dados influencia no treinamento da IA.

REFERENCES

- [1] GSIGMA. Introdução às Redes Neurais Artificiais. Disponível em: <https://www.gsigma.ufsc.br/popov/aulas/ia/modulo9.pdf>. Acesso em: 14 ago. 2020.
- [2] FACURE, Matheus. Introdução às Redes Neurais Artificiais. 2017. Disponível em: <https://matheusfacure.github.io/2017/03/05/ann-intro/Redes-Neurais>. Acesso em: 14 ago. 2020.
- [3] POZO, Aurora. COMPUTAÇÃO EVOLUTIVA. Disponível em: <http://www.inf.ufpr.br/aurora/tutoriais/Ceapostila.pdf>. Acesso em: 14 ago. 2020.
- [4] WIKEPEDIA. Neuroevolution of augmenting topologies. Disponível em: <https://en.wikipedia.org/wiki/Neuroevolution-of-augmenting-topologies>. Acesso em: 14 ago. 2020.
- [5] SODER, Alan. AVALIAÇÃO DA ESTRATÉGIA DE APRENDIZADO NEAT APLICADA A UM JOGO ELETRÔNICO DE CONSOLE DE 8 BITS. 2018. 48 f. Tese (Doutorado) - Curso de Engenharia da Computação, Univates, Lajeado, 2018.