Memorial da disciplina de Redes Neurais e Algoritmos Genéticos

Gabriel Martins Sousa

Ilum Escola de Ciência

Prof. Responsável: Dr. Daniel Roberto Cassar

Agradecimentos

Agradeço à colaboração de minha guilda *Clareyamar*, formada por mim, a Maria Clara Lelis e a Yasmin Shimizu, a qual compartilhei o desenvolvimento do Projeto Final Tarrasque e as manhãs de terça-feira na mesa da porta, juntamente com meu amigo Vitor Dreveck, que trabalhou comigo e me ajudou até fora dos trabalhos, assim como minha amiga Letícia Nunes, com a qual cooperei no desenvolvimento de algumas Feras e os intervalos famintos no meio das aulas. Dividi essa dura jornada com muitos fiéis companheiros, os quais guardo no coração. *Obrigado, amigos! :)*

Contracapa

Don't try to look so wise

Don't cry cause you're so right

Don't dry with fakes or fears

'Cause you will hate yourself in the end

— Wind, Akeboshi

1 Introdução

Esse é o Memorial do Cavaleiro Mago Gabriel Martins e suas atividades realizadas na disciplina de Redes Neurais e Algoritmos Genéticos, ministrada pelo Prof. Dr. Daniel Cassar. Nesse documento, encontram-se todas as atividades e os projetos realizados durante o decorrer da matéria, desde os monstrinhos, as feras formidáveis, e o chefão final Tarrasque. Em cada atividade, será descrito seu título, quem participou da sua execução, o objetivo da tarefa, como ela foi desenvolvida, a conclusão do trabalho, o que foi aprendido durante o processo e onde é possível encontrá-la.

Este é um documento de recapitulação da jornada do Mago Haryell Marino (Gabriel Martins) no enfrentamento das intimidadoras Redes Neurais e dos numerosos Algoritmos Genéticos.

2 Monstrinhos

2.1 Monstrinho 3.1

Nome: "Eu podia jurar que não veria grafos nunca mais na minha vida" — aluno da turma 2024 que não quis se identificar

Equipe: Gabriel Martins Sousa

Objetivo: Fazer em uma folha sulfite o grafo computacional da expressão abaixo.

$$L = (y(ab + cd + ef + g))^2$$

Realizar o forward pass considerando que L é o vértice folha e os seguintes valores

Resolução: Realizei manualmente em uma folha sulfite a construção do grafo representando a equação, calculando as operações na etapa de forward pass, e, através do backpropagation, obtive as derivadas locais através de derivadas parciais. Ao final, obtive o resultado de L.

Conclusão e aprendizado: Através desta tarefa, relembrei os conceitos de derivada parcial, regra da cadeia e grafos. Entendi manualmente o mecanismo de backpropagation e, de maneira simplificada, a operação dos neurônios na rede. Além disso, conclui que conforme a expressão aumenta ou surgem muitas expressões, é inviável fazer manualmente, por isso, vamos aprender como realizar de maneira computacional e automatizada.

Onde está a tarefa: O arquivo com a folha sulfite escaneada está em um repositório do Github que pode ser acessado aqui. Ele está na pasta Monstrinhos com o nome de "Monstrinho 3.1-Backpropagation-Gabriel Martins.pdf"

2.2 Monstrinho 3.2

Nome: “Átomos não são bolinhas e ligações químicas não são pauzinhos” — Prof. Julio

Objetivo: Utilizar classes de Python para modelar elementos químicos e moléculas.

Equipe: Gabriel Martins Sousa

Resolução: Para modelar elementos químicos, foi criado uma classe que cria objetos que representam as informações do elemento desejado, como símbolo, número atômico e peso atômico.

Queríamos inicialmente modelar uma molécula de glicose, então instanciamos os elementos Carbono, Hidrogênio e Oxigênio, e exibimos suas informações. Em seguida, para modelarmos moléculas apartir dos elementos, criamos a classe Molécula, que gera os objetos que armazenarão as informações das moléculas como fórmula química e peso molecular. Para instanciar uma molécula, é necessário um dicionário que contenha a instância dos elementos como chave e sua quantidade na molécula como valor. A partir disso, a instância da classe exibe a fórmula química e o peso molecular.

Conclusão e aprendizado: Ao final, conseguimos modelar a molécula desejada e obter suas informações principais. No caminho para isso, praticamos os conceitos de classe que aprendemos e relembramos o uso de dicionários.

Onde está a tarefa: O Notebook de Python com a atividade está no github da matéria de Redes Neurais, que pode ser acessado aqui. O notebook se encontra na pasta Monstrinhos com o nome de ”Monstrinho 3.2-Moléculas com classe-Gabriel Martins.ipynb”.

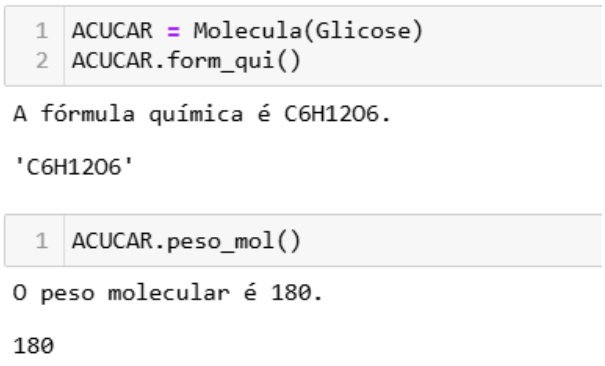

Figure 1: Retorno das funções *form_qui* e *peso_mol* da molécula de glicose.

2.3 Monstrinho 3.3

Nome: Classes em Python não pagam imposto sobre herança

Objetivo: Modelar algum conceito científico utilizando herança de classes.

Equipe: Gabriel Martins Sousa

Resolução: O conceito que iremos modelar é o de partículas subatômicas e algumas de suas interações, e ao final, modelar, de maneira bastante simplificada, a colisão de 2 partículas em velocidades relativísticas. Queremos estudá-las sob os ramos da Física de Partículas e de Altas Energias.

Diante disso, vamos iniciar criar uma classe que modele uma partícula genérica com massa e carga. Nela, criamos um método que calcula a interação coulombiana entre duas partículas carregadas em função de suas cargas e a distância entre elas. Em seguida, criamos a classe **Nucleo**, que modela um núcleo de átomo. Essa classe herda a classe **Particula**, e adiciona informações como **número atômico** e o **elemento** a qual pertence aquele núcleo. Nessa classe, criamos métodos para calcular a **energia cinética relativística** e simular a **colisão de partículas** para aferir se houve formação do Quark Gluon Plasma(QGP).

Conclusão e aprendizado: Ao final, conseguimos modelar de maneira simples as partículas e os fenômenos e obtivemos os valores esperados. Com o desenvolvimento da atividade, foi possível entender melhor o conceito de classes, herança de classes e suas aplicações. Além disso, pude revisar alguns conceitos de Física de Partículas e Altas Energias.

Onde está a tarefa: O Notebook de Python com a atividade está no github da matéria de Redes Neurais, que pode ser acessado aqui. O notebook se encontra na pasta Monstrinhos com o nome de "Monstrinho 3.3-Herança de Classes-Gabriel Martins.ipynb".

```
In [18]:  1 CHUMBO.colisao(290000000, CHUMBO2, 290000000)
Out[18]: 'A energia total da colisão foi de 1086141237179.5946 eV e houve formação de QGP'
```

Simulamos a colisão e ela resultou em um QGP. Porém, e se diminuirmos a velocidade das partículas na colisão.

```
In [20]:  1 CHUMBO.colisao(280000000, CHUMBO2, 280000000)
Out[20]: 'A energia total da colisão foi de 667395475171.5994 eV, porém, não houve formação de QGP'
```

Figure 2: Retorno da simulação da colisão com 2 velocidades diferentes e mesmo núcleo

2.4 Monstrinho 3.5

Nome: Forma, função e ativação

Objetivo: Implementar 3 novas funções de ativação na rede neural feita em Python puro nos vídeos da disciplina. Escrever brevemente sobre estas 3 funções de ativação, mostrando a equação delas e comentando a diferença com relação à função de ativação sigmoidal. Mostrar que seu código funciona rodando alguns testes simples.

Equipe: Gabriel Martins Sousa

Resolução: Utilizei a classe **Valor** criada pelo Professor Daniel Cassar e adaptei para adicionar as novas funções de ativação: a **Função Tangente Hiperbólica**, a **Função de Unidade Linear Retificada (ReLU)** e a **Função de ReLU paramétrica (PReLU)**. Para as funções de ReLU e PReLU, precisei definir um novo método de realizar a propagação, que está detalhado no *Notebook* da atividade. Realizamos o treino e testamos com as 3 funções novas e a função Sigmoid, e obtivemos o valor de *loss* de cada uma delas para compará-las em um gráfico.

Conclusão e aprendizado: Ao final da atividade, foi possível observar a diferença de performance das redes com diferentes funções de ativação, com o PReLU sendo o que apresentou a menor *loss*, e a função Sigmoid a que teve a maior *loss*. Ainda sim, estudando as funções, cada uma tem suas aplicações especifícas, nas quais desempenham um melhor resultado.

Onde está a tarefa: O Notebook de Python com a atividade está no github da matéria de Redes Neurais, que pode ser acessado aqui. O notebook se encontra na pasta Monstrinhos com o nome de "Monstrinho 3.5-Forma, função e ativação-Gabriel Martins.ipynb".

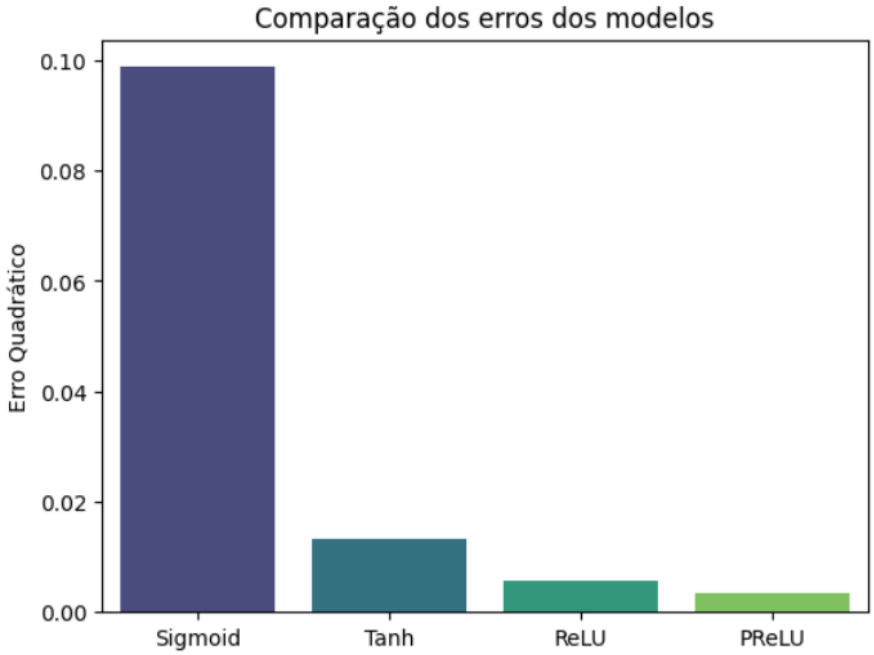

Figure 3: Comparação entre o erro quadrático dos modelos de MLP com as 4 funções de ativação

2.5 Monstrinho 3.5

Nome: Praticamente um X-man!

Objetivo: Criar e evoluir um algoritmo genético para resolver um problema de otimização que tenha mais do que um operador de mutação.

Equipe: Gabriel Martins Sousa

Resolução: Decidi aplicar 3 métodos de mutação no problema da senha variável, me baseando na atividade *Fera formidável 4.9 - Senha de tamanho variável.ipynb*, que desenvolvi com minha colega Letícia Nunes. No problema desenvolvido, tentamos encontrar uma senha desejada sem saber o tamanho da senha verdadeira, apenas a distância dos caracteres do candidato para os caracteres da senha correta, usando como referência a ordem da *tabela ascii*. Os três tipos de mutações que aplicamos foram: a **mutação simples**, que seleciona um gene para trocá-lo por um valor aleatório dentre os possíveis, a **mutação de salto**, que troca um gene pelo valor seguinte ou anterior a ele, e a nova mutação que criei, a **mutação de tamanho**, que adiciona ou remove um gene aleatório ao indivíduo.

Conclusão e aprendizado: Com o desenvolvimento do trabalho, pude entender mais na prática o funcionamento dos operadores de mutação, além de criar uma mutação específica para ajudar na convergência do problema de interesse. Consegui obter um bom resultado, e exercitar ainda mais a prática de algoritmos genéticos.

Onde está a tarefa: O Notebook de Python com a atividade está no github da matéria de Redes Neurais, que pode ser acessado aqui. O notebook se encontra na pasta Monstrinhos com o nome de "Monstrinho 3.10-Praticamente um X-man!.ipynb".

```
Geração 191 - AmqAgterENecanicbNuantica
Geração 194 - AmqAgterENecanicaNuantica
Geração 205 - AmpAgterENecanicaNuantica
Geração 208 - AmpAfterENecanicaNuantica
Geração 225 - AmpAfterENecanicaOuantica
Geração 237 - AmpAfterEMecanicaOuantica
Geração 243 - AmoAfterEMecanicaOuantica
Geração 246 - AmoAfterEMecanicaPuantica
Geração 256 - AmoAfterEMecanicaQuantica
```

Figure 4: Últimos candidatos encontrados pela evolução do AG e o indivíduo final correspondente a senha verdadeira.

3 Feras Formidáveis

3.1 Fera 4.3

Nome: Derrube pra fora

Objetivo: Implementar o regularizador dropout na rede neural feita em Python puro.

Equipe: Gabriel Martins Sousa e Letícia Almeida Nunes

Resolução: Para a atividade, utilizamos de base o código da rede neural feito em Python puro pelo Professor Daniel Cassar. A partir dele, alteramos a classe Neurônio para que, quando acessamos um Neurônio, sorteamos um valor aleatório entre 0 e 1, e caso esse valor seja menor que a taxa de *dropout* definida por nós, o Neurônio não processa a informação de entrada e apenas retorna valor 0. Com isso, ocorre o desligamento de uma porcentagem dos neurônios da rede. A taxa de *dropout* é fornecida ao chamar a classe MLP e é passada para as classes Camada e Neurônio. Além disso, adaptamos a classe Camada para que ela não realize o *dropout* na camada de entrada e de saída. Ao fim, calculamos a soma dos erros quadráticos das previsões para analisar a performance do modelo.

Conclusão e aprendizado: Ao final, estudamos e implementamos a estratégia de *dropout* dos neurônios da rede neural, a fim de evitar o sobreajuste e melhorar o desempenho da rede. Obtivemos um valor baixo de erro em comparação a magnitude dos nossos dados, o que indica que a rede foi construída e treinada com sucesso.

Onde está a tarefa: O Notebook de Python com a atividade está no github da matéria de Redes Neurais, que pode ser acessado aqui. O notebook se encontra na pasta Feras Formidáveis com o nome de "Fera formidável 4.3 - Dropout.ipynb".

```python
soma = 0

for y_t, y_p in zip(y_true, y_pred):
    erro_quad = (y_t - y_p.data)**2
    raiz_erro_quad = (erro_quad)**(1/2)
    soma += raiz_erro_quad

RMSE = soma/len(y_true)
print(f"A soma do erro quadrático médio foi {RMSE:.3f}")

print(f"O intervalo máximo de valores dos nossos dados são {max(y_true)-min(y_true)}")
```

```
A soma do erro quadrático médio foi 0.059
O intervalo máximo de valores dos nossos dados são 1
```

Figure 5: Soma dos erros quadráticos das previsões da MLP

3.2 Fera 4.9

Nome: A senha de tamanho variável

Objetivo: Resolver o problema da senha de forma que você não forneça a informação do tamanho da senha para a função que gera a população, considerando que a senha pode ser uma string de 1 até 30 caracteres.

Equipe: Gabriel Martins Sousa e Letícia Almeida Nunes

Resolução: Usamos de base o algoritmo do Professor Daniel Cassar para o Problema da Senha. Como não podemos fornecer o tamanho da senha para a criação dos candidatos e da população, vamos gerar indivíduos de tamanhos aleatórios entre 1 e 30 caracteres. Além disso, criamos uma mutação que adiciona ou retira um gene, para conseguirmos alterar também o tamanho dos indivíduos. Para calcular a função objetivo com indivíduos de tamanhos diferentes, comparamos um candidato com a senha original, caso eles não possuam o mesmo tamanho, o de menor tamanho é preenchido com zeros até atingir o tamanho do maior, e em seguida fazemos o cálculo da distância entre os caracteres do candidato e da senha.

Conclusão e aprendizado: Ao fim, o algoritmo conseguiu encontrar a senha que queríamos de maneira rápida. Através do trabalho, conseguimos explorar melhor os operadores genéticos, e também a contornar situações nas quais temos que resolver um problema, mas não temos todas as informações sobre ele.

Onde está a tarefa: O Notebook de Python com a atividade está no github da matéria de Algoritmos Genéticos, que pode ser acessado aqui. O notebook se encontra na pasta Feras Formidáveis com o nome de "Fera formidável 4.9 - Senha de tamanho variável.ipynb".

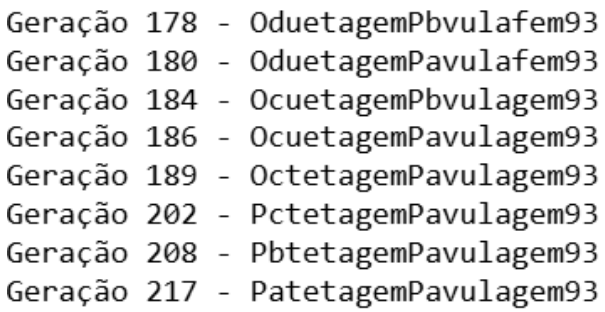

Figure 6: Melhores candidatos de cada geração e o candidato final correspondente a senha original

3.3 Fera 4.12

Nome: Novos palíndromos

Objetivo: Encontrar pelo menos 10 palíndromos diferentes de 5 letras. Estes palíndromos devem ter pelo menos uma vogal. Não é necessário que eles formem palavras válidas em português ou qualquer outro idioma.

Equipe: Gabriel Martins Sousa e Letícia Almeida Nunes

Resolução: Para solucionar essa tarefa, utilizamos como base a resolução do *problema da senha*, feita pelo Professor Daniel Cassar. No entanto, alteramos a função de criação de um indivíduo e a função objetivo, além disso, selecionamos as mutações que consideramos mais adequadas. Para ajudar na convergência do problema, criamos indivíduos que possuam pelo menos uma vogal. Já na função objetivo, nós analisamos a simetria da *string*, analisando se as letras em posições opostas são iguais, a função nos dá uma razão do quão palíndroma é aquela palavra. Além disso, devido às funções de mutação, que podem alterar a presença de vogais na palavra, a função objetivo retorna 0.1 se não houver vogais na função.

Conclusão e aprendizado: Concluído o trabalho, aprendemos como adaptar nosso problema a um outro problema parecido para que possamos reutilizar os métodos já conhecidos. Além disso, treinamos mais como utilizar AGs para resolver nossos desafios.

Onde está a tarefa: O Notebook de Python com a atividade está no github da matéria de Algoritmos Genéticos, que pode ser acessado aqui. O notebook se encontra na pasta Feras Formidáveis com o nome de "Fera formidável 4.12 - Gabriel Martins e Letícia Nunes.ipynb".

```
1 ['ipepi']
2 ['ipepi', 'lpepl']
4 ['ipepi', 'lpepl', 'ypipy']
7 ['ipepi', 'lpepl', 'ypipy', 'ypepy']
14 ['ipepi', 'lpepl', 'ypipy', 'ypepy', 'ypopy']
109 ['ipepi', 'lpepl', 'ypipy', 'ypepy', 'ypopy', 'ypupy']
515 ['ipepi', 'lpepl', 'ypipy', 'ypepy', 'ypopy', 'ypupy', 'ypapy']
6658 ['ipepi', 'lpepl', 'ypipy', 'ypepy', 'ypopy', 'ypupy', 'ypapy', 'zpupz']
11911 ['ipepi', 'lpepl', 'ypipy', 'ypepy', 'ypopy', 'ypupy', 'ypapy', 'zpupz', 'xpupx']
38390 ['ipepi', 'lpepl', 'ypipy', 'ypepy', 'ypopy', 'ypupy', 'ypapy', 'zpupz', 'xpupx', 'yqiqy']
```

Figure 7: 10 palíndromos gerados pela evolução do Algoritmo Genético, cada um com 5 letras e pelo menos 1 vogal.

3.4 Fera 4.13

Nome: A liga ternária mais cara do mundo

Objetivo: Encontrar uma liga de três elementos que tenha o maior custo possível. A liga ternária deve ser da forma xA.yB.zC sendo que $x + y + z = 100$ g, $x \geq 5$ g, $y \geq 5$ g, z ≥ 5 g e "A", "B" e "C" são elementos químicos diferentes. Utilizar o preço dado em um dicionário fornecido pelo Prof. Daniel. Considerar que qualquer composto com 3 elementos químicos é chamado de liga.

Equipe: Gabriel Martins Sousa e Vitor Gabriel Dreveck

Resolução: Inicialmente, percebemos que o nosso problema poderia ser adaptado para algo similar ao problema das caixas não-binárias, com isso poderíamos apenas alterar os operadores que haviámos aprendido em aula para resolver a nossa atividade. Para isso, atribuímos um valor numérico inteiro para cada um dos elementos no dicionário. Em seguida, criamos os nossos operadores genéticos para o problema da liga, desde a função de criação da população, que cria indivíduos com a soma dos elementos constituintes igual a 100g, passando pelos operadores de seleção, cruzamento e mutação. Por último, nossa função objetivo é o preço final da liga, calculada através da massa e do preço dos elementos. Nós evoluimos o algoritmo genético e obtivemos o melhor candidato dentre os testados.

Conclusão e aprendizado: Com a conclusão do trabalho, nós exercitamos mais nossas habilidades com Algoritmos Genéticos (AG), além de aprender a contornar restrições e adaptar métodos já existentes para resolver o nosso problema. Aprendemos um pouco sobre ligas e aplicamos os AGs em algo mais cotidiano.

Onde está a tarefa: O Notebook de Python com a atividade está no github da matéria de Algoritmos Genéticos, que pode ser acessado aqui. O notebook se encontra na pasta Feras Formidáveis com o nome de "Fera Formidavel 4.13 - Gabriel e Vitor.ipynb".

```
Melhor indivíduo: [89, 5, 6, 82, 83, 91]
Preço: $4524910000000.00

Composição:
89g de Po
5g de Ac
6g de Bk
```

Figure 8: O melhor candidato obtido pelo algoritmo após 10000 gerações. Não é o melhor indivíduo possível mas é bastante próximo.

3.5 Fera 4.14

Nome: A liga ternária leve mais cara do mundo

Objetivo: Encontrar uma liga de três elementos que tenha o maior custo e o menor peso atômico. A liga ternária deve ser da forma xA.yB.zC sendo que $x + y + z = 100$ g, $x \geq 5$ g, $y \geq 5$ g, $z \geq 5$ g e "A", "B" e "C" são elementos químicos diferentes. Utilizar o preço dado no exercício anterior e peso atômico dado em outro dicionário. Considerar que qualquer composto com 3 elementos químicos é chamado de liga.

Equipe: Gabriel Martins Sousa e Vitor Gabriel Dreveck

Resolução: Resolvemos o problema pelo mesmo método da questão anterior. Percebemos que nosso problema podia ser adaptado ao modelo das caixas não-binárias, bastando ajustar os operadores vistos em aula. Atribuímos valores inteiros aos elementos do dicionário e desenvolvemos operadores genéticos específicos: geração de indivíduos cuja soma totaliza 100g, além de seleção, cruzamento e mutação. A função objetivo foi definida como o preço final da liga dividida pela soma das massas atômicas dos elementos constituintes, com base nas massas e nos preços dos elementos. Após evoluir o algoritmo genético, encontramos o melhor candidato entre aqueles que testamos.

Conclusão e aprendizado: Ao final, treinamos ainda mais o desenvolvimento de Algoritmos Genéticos. Vimos como contornar restrições e aprendemos um pouco mais sobre ligas.

Onde está a tarefa: O Notebook de Python com a atividade está no github da matéria de Algoritmos Genéticos, que pode ser acessado aqui. O notebook se encontra na pasta Feras Formidáveis com o nome de "Fera Formidavel 4.14 - Gabriel e Vitor.ipynb".

```
Melhor indivíduo: [90, 5, 5, 81, 1, 0]
Preço: $4428000000000.13
Peso atômico: 188.35 g/mol

Composição:
90g de Po
5g de He
5g de H
```

Figure 9: Melhor indivíduo encontrado pela evolução do AG após 10000 gerações.

4 Projeto Final

Nome: **Tarrasque**

Objetivo: Identificar e otimizar os hiperparâmetros de uma rede neural do tipo MLP para resolver um problema de regressão ou de classificação de interesse científico. Testar ao menos 100 diferentes arquiteturas de rede durante a otimização de hiperparâmetros. Os dados para treinar o modelo são dados tabulados.

Equipe: Gabriel Martins Sousa, Maria Clara Macedo Lelis e Yasmin Shimizu

Resolução: O grupo desenvolveu uma Rede Neural do tipo MLP para regressão utilizando o módulo **Pytorch**, a fim de prever o pH de amostras de água a partir de quantidades de diversos componentes químicos presentes nas amostras. Além disso, otimizamos os hiperparâmetros da rede MLP através do módulo **Optuna**, que testa diferentes arquiteturas da rede, compara as métricas de performance dessas diferentes arquiteturas e retorna ao usuário os hiperparâmetros do modelo com a menor função de perda. O dataset usado no projeto foi obtido na plataforma *Kaggle*, e se refere a características de amostras de água obtidos de vários distritos.

Conclusão e aprendizado: Ao longo do projeto, pudemos rever e pôr em prática os conceitos aprendidos durante o semestre, desde a construção de uma Rede Neural com *Pytorch* até a otimização dos hiperparâmetros com o *Optuna*. Entendemos melhor quais são e como funcionam os hiperparâmetros de uma MLP. Além disso, pudemos trabalhar em grupo de maneira bastante colaborativa e construtiva.

Onde está a tarefa: O Notebook de Python com todos os processos está no github do projeto, que pode ser acessado aqui. O repositório contém o Notebook com os códigos, o dataset utilizado e um README detalhando o desenvolvimento do projeto.

```
parametros = {
    "neuronios":neuronios,

    "func_act":trial.suggest_categorical("func_act",['Sigmoid','ReLU','LeakyReLU','PReLU','Tanh']),

    "lr":trial.suggest_float("lr",1e-4,1e-1,log=True)

    "taxa_drop":trial.suggest_float("taxa_drop",0.05,0.90,log=True)
}
```

Figure 10: Espaço de busca dos parâmetros definidos utilizando o *Optuna*.

5 Reflexões sobre a jornada

Ao fim da matéria, foi possível aprender muito sobre diversas ferramentas de Redes Neurais e Algoritmos Genéticos, desde o seu funcionamento mais básico até aplicações mais complexas e úteis. Ao longo das atividades, também revi conceitos aprendidos nas matérias anteriores de Aprendizado de Máquina e Lógica Computacional, e como o ferramentário que desenvolvemos naquele período ainda nos ajuda na resolução dos problemas mais complexos.

Foi muito bom realizar as atividades em colaboração com meus colegas, nas quais, em todas elas, houveram frutíferas discussões sobre a melhor forma de resolver os problemas e como apresentá-las bem. Trabalhar novamente com o *Clareyamar* no Tarrasque foi um trabalho nostálgico e satisfatório. Foi uma atividade desenvolvida em intensa e igualitária cooperação.

Ainda sim, gostaria de ter praticado mais Redes Neurais (RNs), tanto durante as aulas quanto durante as atividades. As atividades de RNs eram consideravelmente mais complexas do que as de AGs, portanto, foi mais viável no tempo disponível para a matéria me dedicar as Feras de AGs.

Para mim, a matéria foi muito boa, os assuntos aprendidos eram divertidos e as atividades exploravam bem isso. Ainda sim, acho que há algumas coisas a melhorar, como a extensão do curso de Redes e a revisão da quantidade de nota para algumas tarefas. Termino o semestre satisfeito e animado para aplicar o aprendizado desenvolvido ao longo desses 4 meses.

6 Referências

[1] JAMES, Gareth; WITTEN, Daniela; HASTIE, Trevor; et al. An Introduction to Statistical Learning: with Applications in Python. [s.l.: s.n.], 2023. Disponível em: https://www.statlearning.com/.

[2] PYTORCH. PyTorch Documentation. Disponível em: https://docs.pytorch.org/docs/stable/index.html . Acesso em: 07 jun. 2025.

[3] OPTUNA. Optuna: A hyperparameter optimization framework. Disponível em: https://optuna.org/. Acesso em: 07 jun. 2025.

[4] GARLADINNE, Sivapriya. Telangana Post Monsoon Ground Water Quality Data. Kaggle, 2023. Disponível em: https://www.kaggle.com/datasets/sivapriyagarladinne/telangana-post-monsoon-ground-water-quality-data. Acesso em: 07 jun. 2025.

[5] Aulas e Notebooks fornecidos pelo Prof. Dr. Daniel Roberto Cassar na disciplina de Redes Neurais e Algoritmos Genéticos.