Redes Neurais — Backpropagation

Trabalho 2 — INF01017

Gabriel Stepien - 00265035
Leonardo Vianna - 00274721
Vinícius Scheffel - 00264311

1. Características gerais da implementação

O trabalho foi realizado utilizando ferramentas para facilitar o manuseio dos dados, como numpy e pandas. Além disso, procuramos utilizar bons princípios de orientação a objetos, dividindo os algoritmos em classes com nomes propícios. Um breve resumo da implementação será dado agora.

Network

A classe Network é a classe principal, que implementa a rede neural e os seus principais métodos relacionados. O seu construtor recebe como parâmetros de entrada:

1) Número de layers: int
2) Valores de X: matrix
3) Valores de Y: matrix
4) Fator de regularização: float
5) Pesos: lista
6) Topologia da rede: lista
7) Flag de debug: bool
8) Alpha: float
9) Critério de parada: float

Os principais métodos dessa classe são:

- **train**: utilizado para treinar a rede. Esse método é chamado uma vez e, a partir dos dados contidos na rede, irá realizar o *backpropagation* e a atualização dos pesos da rede, até que um erro mínimo seja alcançado, representado pelo critério de parada.
- **backpropagation**: implementado de forma vetorizada, o algoritmo de backpropagation está neste método. Para cada exemplo do training set, realizamos a chamada para o método *propagate* calculamos o delta da última layer, calculamos o delta para as hidden layers, e atualizamos os gradientes das layers baseado no exemplo atual. No final do epoch, atualizamos os pesos.
- **propagate**: neste método realizamos a propagação através de toda a rede.
- **cost_function**: implementação da função J de custo. Para tal, inicializamos um acumulador, com a finalidade de calcular o erro total da rede. Para cada exemplo do

set de treinamento, propagamos o *X* e coletamos o *f(X)* de saída predito pela rede; calculamos o vetor J com o custo associado para cada saída para a rede com o exemplo atual. No fim, dividimos o erro total pelo número de exemplos, calculamos o termo de regularização, e retornamos o custo regularizado.

- **calculate_gradient_numerical_verification**: neste método, calculamos a estimativa para os gradientes na rede e os guardamos em uma variável de classe.

KFolds

A lógica de K-Folds foi implementada na classe **KFolds**, que recebe o conjunto de dados, a coluna de predição, e o valor *k* (número de folds), filtro de dados de predição (previstos). O método *get_folds* retorna uma lista de folds, sendo que cada folds é um subconjunto dos dados de entrada, contidos dentro de um *dataframe* Pandas.

CrossValidation

A classe **CrossValidator** é responsável por realizar a lógica de cross-validation, utilizando os K-Folds gerados pela classe **KFolds**. Os parâmetros do construtor dessa classe são: k (número de folds), conjunto de dados, função de filtro para dados previstos, a coluna de predição, as colunas numéricas, os pesos, a topologia, o alpha, e o critério de parada. O processo de cross-validation é realizado no método *k_fold_cross_validation* e segue, em alto nível, a seguinte lógica:

1) Para cada um dos kfolds:
    a) Utilizar esse kfold como teste, e todos os outros como conjunto de dados de treinamento.
    b) Treinamento da rede utilizando os folds de treinamento.
    c) Coleta de resultados de predição utilizando o fold de teste.
    d) Adicionar resultados dessa iteração na lista de resultados.
2) Retornar lista de resultados.

2 Execução do backpropagation e treinamento

Dentro da pasta "neural-network/new_world" execute:

```
python3 main.py -n network_rede2.txt -w initial_weights_rede2.txt -f
dataset_rede2.txt -alpha 0.0 -backprogtest True
```

Nesse exemplo de execução estamos rodando uma vez o back propagation para coletar os valores de dos gradientes, deltas, fx e erro comparando os gradientes com a aproximação numéricas.

Para rodar o algoritmo, o algoritmo de cross validation para 10 folds e ter os dados das acurácias e erros salvos em arquivos de saída, deve-se executar o seguinte comando alterando os parâmetros:

```
python3 main.py -n network_rede2.txt -f dataset_rede2.txt -alpha 0.1
```

Os arquivos de saída serão "j.txt" "acc.txt" "result.txt" dentro da pasta "new_world" com os valores das medidas para cada iteração do cross-validation.

3 Classificação de novas instâncias

Para classificar instâncias, dentro da classe **Network**, que constitui internamente uma Rede Neural que será utilizada para classificação, implementamos o método *classify_dataset* que recebe a instância a ser classificada.

O método funciona iterativamente, utilizando outro método, chamado *classify,* para classificar instâncias singulares. O método *classify* realiza um propagate da instância, e retorna os resultados para serem coletados.

```
def classify_dataset(self, dataset):
    """
      Classifies an dataset
    :param dataset: pd.Dataframe
    :return: list of results
    """

    instances = np.matrix(dataset.to_numpy())
    results = []

    for i in range(instances.shape[0]):
        results.append(self.classify(instances[i]))

    return results

def classify(self, instance: np.matrix):
    """
      Classifies an instance
    :param instance: np.matrix
    :return: the result of the propagation
    """
    self.current_x = instance
    return self.propagate()
```

Código fonte da classificação de uma instância

4 Resultados

House Votes

Teste do algoritmo com o Cross Validation para o dataset House Votes com o seguinte conjunto de parâmetros:

- Taxa de atualização dos gradientes (alpha): 1.0
- Regularization factor: 2.0
- Topologia da rede: [16, 2, 3, 2] (primeiro elemento é o número de atributos, o último é o número de classes possível.
- Stop Criteria: 0.001

Iteration	Acurácia
1	0.886363636
2	0.977272727
3	0.954545455
4	0.909090909
5	0.613636364
6	0.886363636
7	0.954545455
8	0.976744186
9	0.619047619
10	0.880952381
Mean	0.865856
Desvio padrão	0.136589

Acurácia por iteração do cross validation

Caso 0:

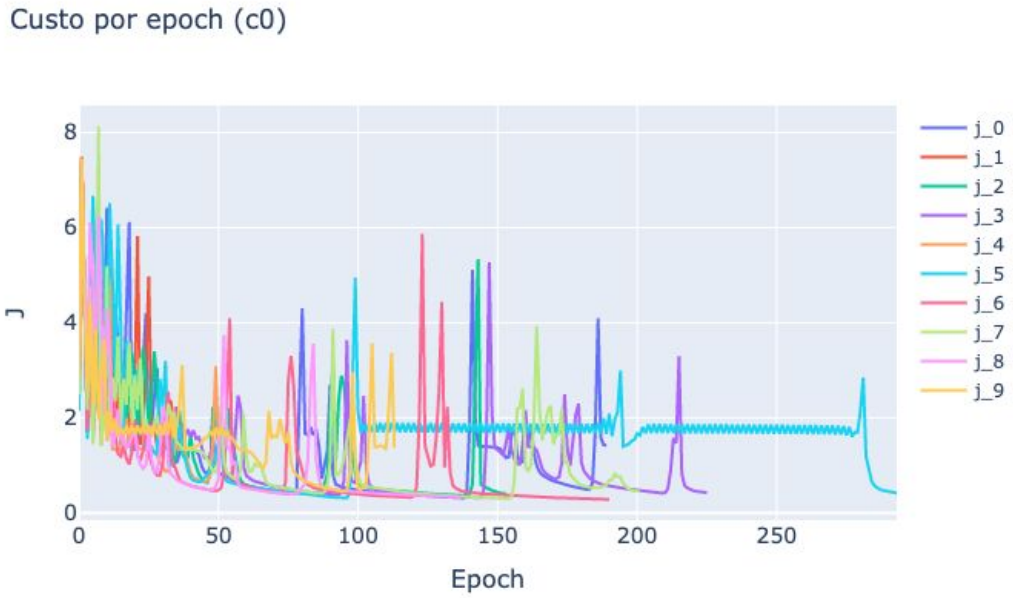
Rede: 16 4 2

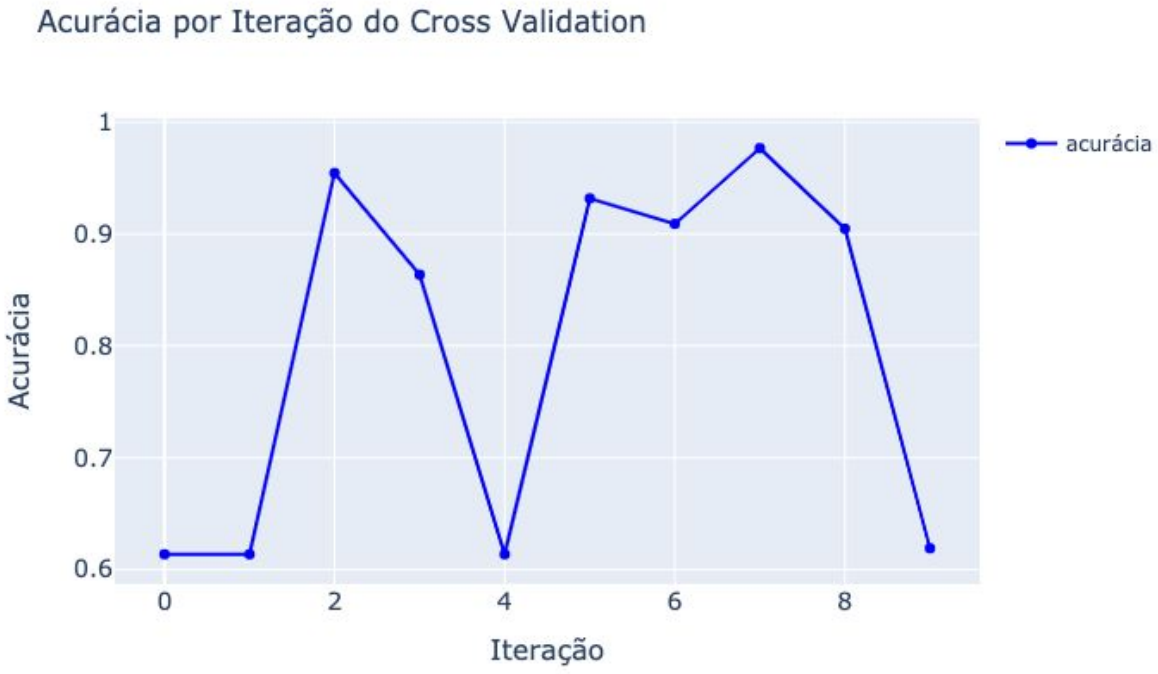
Regularização: 0.2

Alpha: 5

Stop Criteria: 0.001

Dados: house-votes-84.tsv





Caso 1:

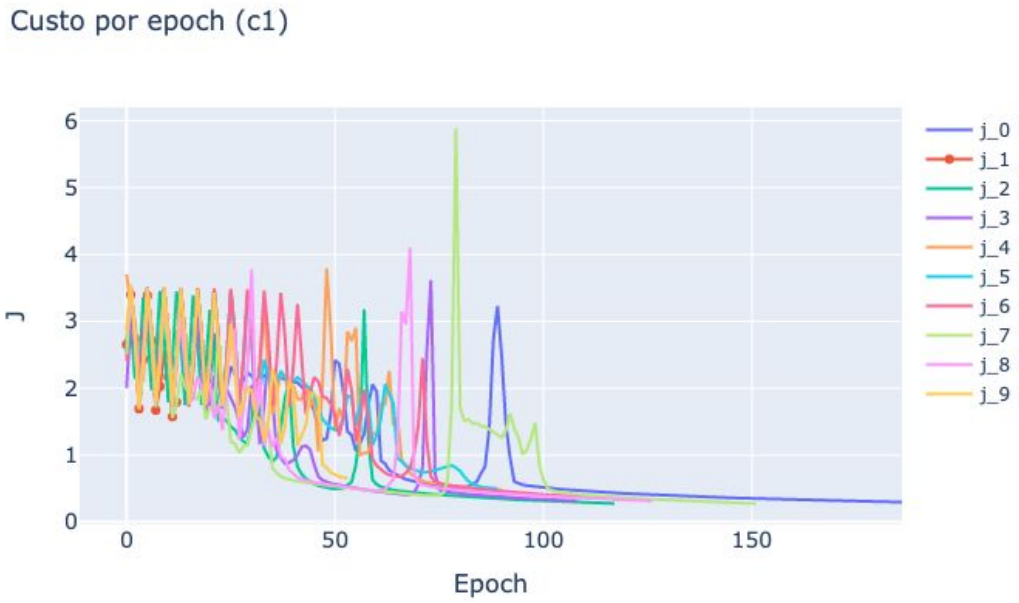
Rede: 16 4 2

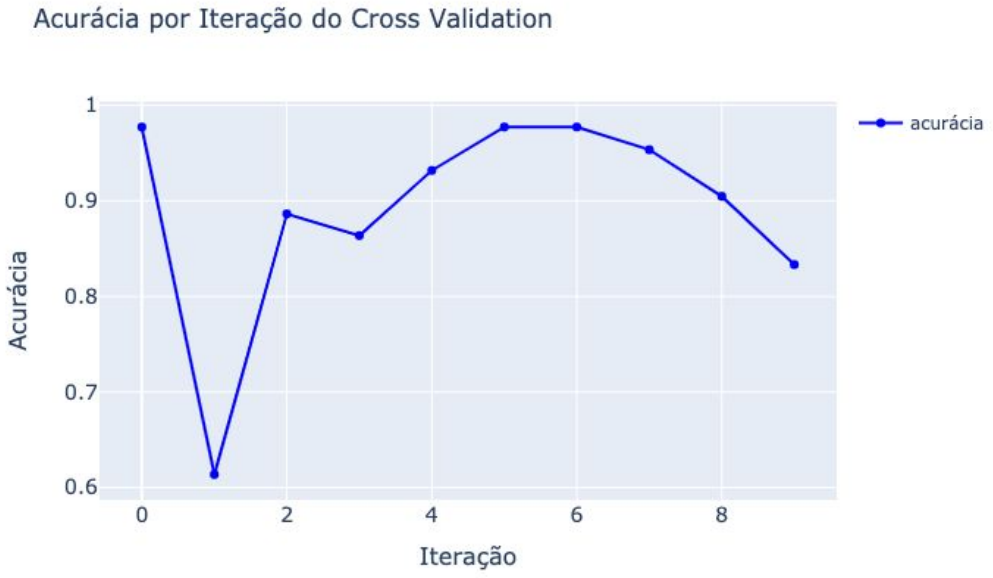
Regularização: 0.2

Alpha: 3

Stop Criteria: 0.001

Dados: house-votes-84.tsv





Caso 2:

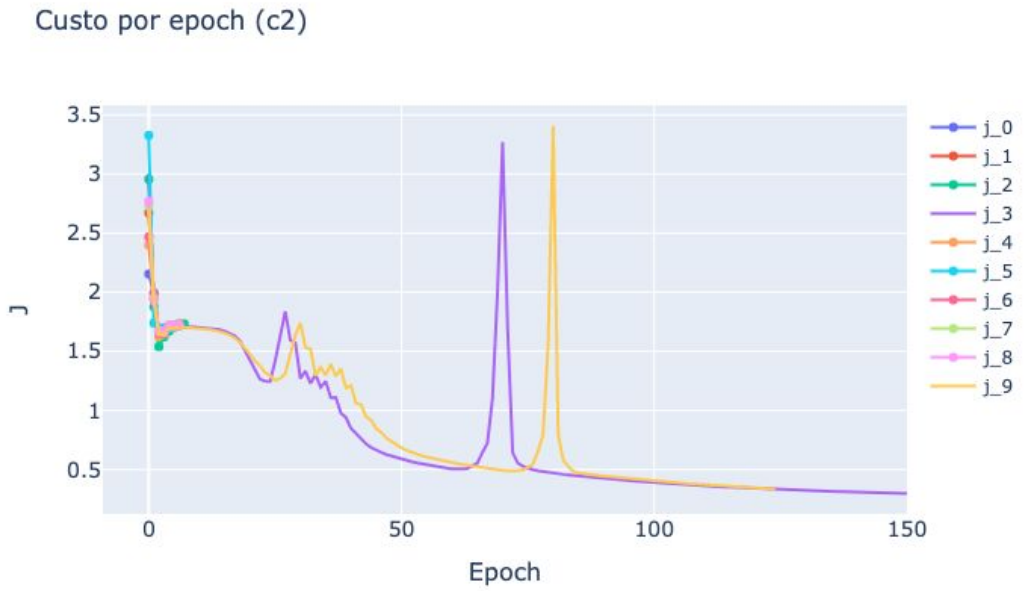
Rede: 16 4 2

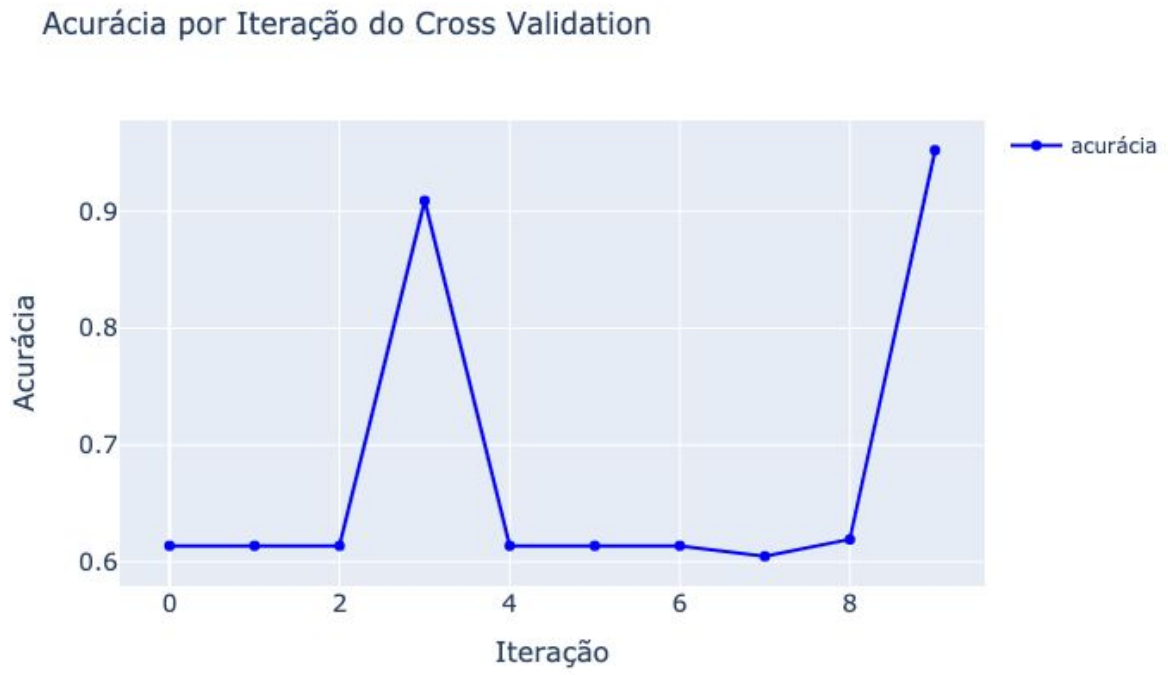
Regularização: 0.2

Alpha: 2

Stop Criteria: 0.001

Dados: house-votes-84.tsv





Wine Recognition

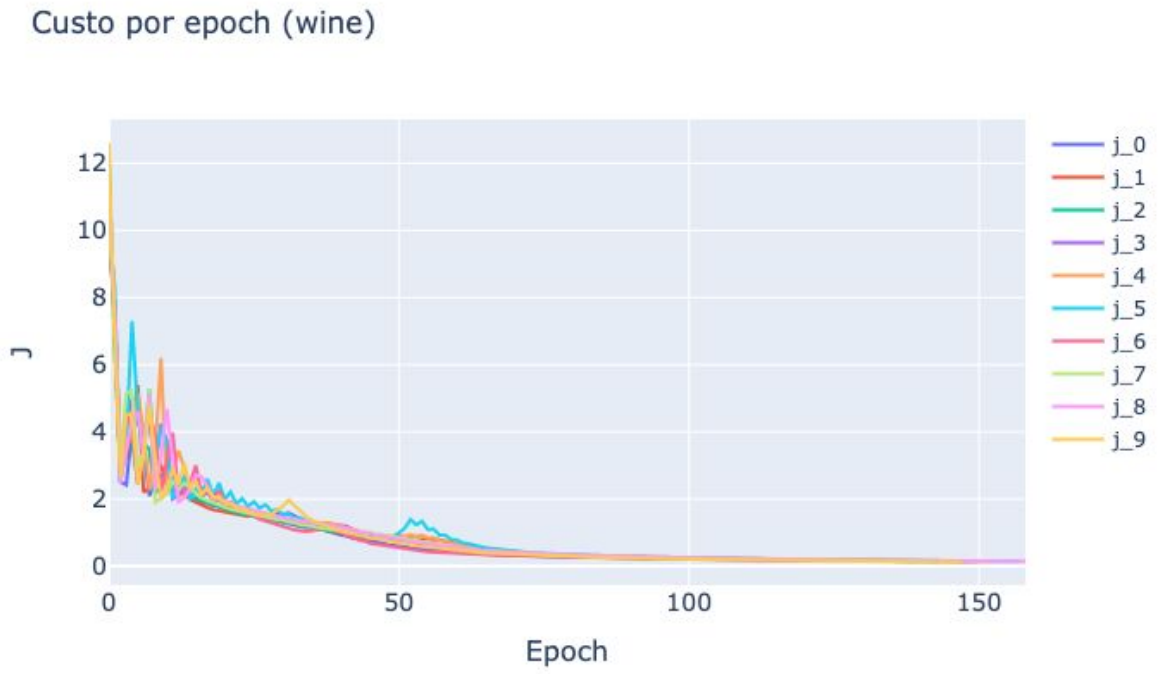
Não foi possível adquirir resultados satisfatórios da acurácia, pois acredita-se que há um bug ao fazer o cálculo da acurácia para o dataset wine-recognition.tsv. Entretanto, verificamos que o valor do J foi satisfatório, o que pode ser visto no gráfico a seguir:

Rede: 13 10 3

Regularização: 0.01

Alpha: 2

Stop Criteria: 0.001



Cada traço representa a execução de uma iteração do Cross Validation. As cores são especificadas na legenda.

5 Análise da corretude da implementação

Para a maioria dos parâmetros utilizados, o algoritmo parece ficar travado em um ótimo local muito ruim. No entanto, utilizando certos parâmetros específicos para o dataset *House Votes*, o grupo percebeu uma precisão e corretude muito grande. Dessa maneira, não ficou claro o porquê do resultado ruim nos outros casos.

Conclusão

Concluímos, com esse trabalho, que é importante pensar sobre a manipulação dos dados na hora de iniciar o desenvolvimento dos algoritmos, porque a biblioteca Pandas facilitou consideravelmente o trabalho em certos pontos da implementação. Começamos não utilizando uma boa manipulação e estruturação dos dados, e isso causou uma demora excessiva para realizar as implementações. Depois disso, percebemos que seria um melhor caminho voltar atrás, e reiniciar o trabalho utilizando um ponto de vista mais inteligente,

através do vetorização dos dados. Dessa maneira, a implementação foi facilitada, e conseguimos continuar o trabalho até a conclusão.

Algo interessante que percebemos foi que o algoritmo de Redes Neurais com *backpropagation,* quando recebe argumentos corretos e coerentes, é um algoritmo extremamente preciso. Tivemos esse insight ao comparar os resultados que obtivemos no caso do *House Votes*, com o resultado obtido no trabalho anterior de Random Forests.

É preciso salientar a gigantesca importância dos parâmetros para o bom resultado da rede, porque o seu impacto pode ser notado pelo grupo na hora de classificar novas entradas. Utilizando parâmetros com valores quaisquer, a função de custo J estava tendendo sempre para um valor ruim, acima de 1. Além disso, as predições resultantes estavam sempre prevendo os mesmos valores, mesmo utilizando dados de predição diferentes. Depois de um longo tempo de experimentação, descobrimos bons valores para o caso do *House Votes,* o que pode ser verificado nos resultados.

O grupo entendeu como insatisfatório o resultado das acurácias em geral, o que pode ser verificado nos gráficos analisados, porque não conseguimos ter um entendimento total do porquê do algoritmo várias vezes resultar no mesmo valor. Uma das hipóteses é a de que o algoritmo está atingindo um ótimo local que não é bom o suficiente. Ainda assim, não sabemos porque ele não é capaz de atingir o ótimo global mesmo alterando de diversas maneiras os parâmetros.

Vídeo da apresentação

- https://youtu.be/9hxRZycUKOM