Implementando sua primeira rede neural em TensorFlow

Introdução

Esse exercício é, em grande parte, uma tradução e adaptação para o português brasileiro do tutorial intitulado Build Your First Neural Network with Pytorch, entretanto algumas adaptações foram realizadas, tanto no texto, quanto no código, em relação à versão original para utilizar a biblioteca TensorFlow. Este é o enunciado para apenas a última parte da lista 2 (questões 3 e 4) do curso MAC5725, o aluno deve realizar o resto da lista antes de fazer este exercício.

Nesse exercício, você aprenderá como implementar, treinar e utilizar uma Rede Neural *Feed-Foward* simples para uma tarefa de classificação binária.

Para tal, utilizaremos o pacote TensorFlow 2.0 que é, atualmente, uma das principais ferramentas para a implementação de modelos neurais viáveis.

Implementaremos nossa rede neural em *Python 3* e, além do *TensorFlow*, você também precisará instalar e importar os pacotes NumPy, Pandas e Scikit-learn.

A tarefa que usaremos para fins de exemplo será a de prever se choverá ou não numa cidade australiana amanhã, utilizando dados meteorológicos mensurados na mesma cidade no dia de hoje. A redução dessa tarefa de previsão a uma classificação binária é, evidentemente, uma grande simplificação do problema real de previsão meteorológica, mas como veremos, ainda pode apresentar resultados interessantes, além do caráter didático.

Entendendo os dados

As informações que utilizaremos para treinar nosso modelo para a tarefa de previsão de chuvas estão contidas num conjunto que reúne dados meteorológicos de diversas cidades australianas. Esse conjunto de dados foi curado e disponibilizado através do Kaggle[1] por Joe Young[2].

Os dados estão no formato `.csv` e, com eles em mãos, o primeiro passo é carregá-los em um *data-frame*, usando a função `pd.read_csv()` do *pandas*.

Com os dados carregados, é possível averiguar que eles são constituídos por 142193 entradas, cada uma contando com 24 variáveis distintas. É possível notar, também, que existem entradas para as quais nem todas as variáveis estão instanciadas. Além disso, que nem todos os valores estão nos formatos que gostaríamos que estivessem para serem processados.

[1] https://www.kaggle.com/jsphyg/weather-dataset-rattle-package
[2] https://www.kaggle.com/jsphyg

Isso é normal. Dados reais são cheios de falhas e problemas, e exigem trabalho e entendimento para serem utilizados da maneira correta. Por isso, é necessário realizar um **pré-processamento** para adequar os dados, antes de os passarmos para o modelo.

O primeiro passo é escolher quais das variáveis meteorológicas nos interessam. No nosso caso, queremos prever se choverá ou não amanhã, então `RainTomorrow` será nossa variável alvo. Para prevê-la usaremos as variáveis `Rainfall`, `Humidity3pm`, `Pressure9am` e `RainToday`, que serão nossas *features*.

Em seguida, iniciamos o pré-processamento, propriamente dito.

As variáveis `RainToday` e `RainTomorrow` possuem dois valores possíveis, *"Yes"* e *"No"*. Adequamos esses valores, convertendo-os para 1 e 0, respectivamente.

A seguir, removemos todas as entradas que não tenham instanciados os valores de todas as variáveis de interesse com a função `dropna()`, pois essas entradas são inúteis para treinar nosso modelo.

Com os dados pré-processados, é possível, agora, plotar as distribuições das variáveis de interesse, para poder entender melhor como essas distribuições funcionam. Esse tipo de trabalho é muito importante na implementação real de redes neurais, conhecer os dados é fundamental para tirar o maior proveito do seu modelo e entender verdadeiramente seus resultados.

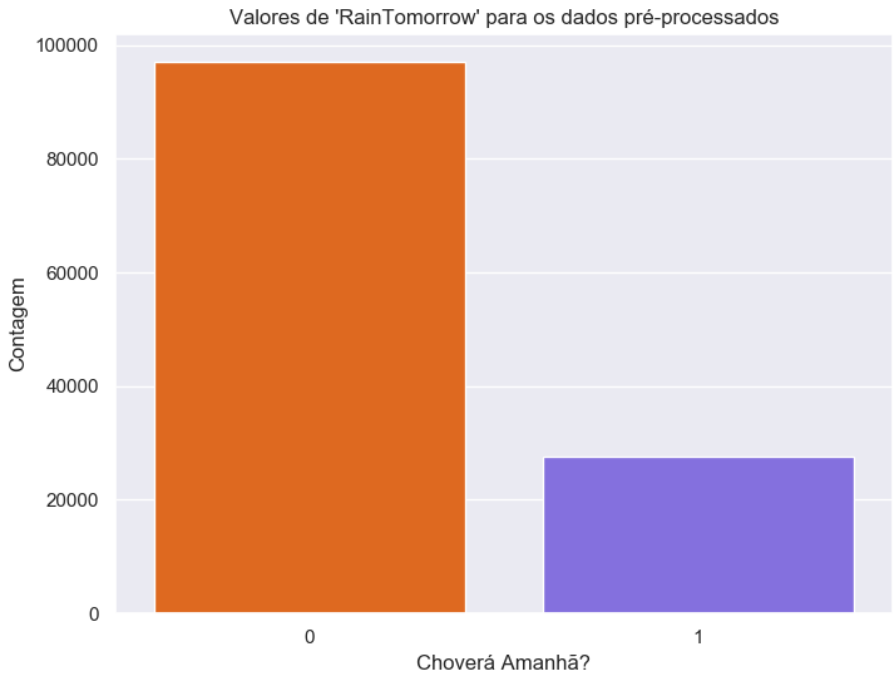

Figura 1: Distribuição de valores para a variável `RainTomorrow`, após o pré-processamento

Dentre todas as distribuições das variáveis de interesse, a que mais nos concerne é a da variável alvo, `RainTomorrow`, representada na *Figura 1*.

Essa distribuição nos revela um grande desbalanço entre os dois valores possíveis dessa variável, que constituirão as duas classes do nosso problema de classificação. Esse é um dado importante, pois pode influenciar significativamente a capacidade preditiva do modelo treinado.

Existem maneiras de se lidar com o desbalanceamento dos dados, mas nesse tutorial utilizaremos os dados dessa forma. Isso significa que o *baseline* para a performance do nosso modelo deve ser 78%, isso porque, se um modelo chutasse que amanhã não irá chover, todas as vezes, ele obteria uma performance dessa ordem e, como esperamos gerar um modelo mais inteligente que isso, esperamos também que a a nossa performance seja superior a essa.

Todo o código referente a esse pré-processamento deve ser escrito pelo próprio aluno seguindo o esqueleto das funções presentes no notebook. Para agilizar a exploração dos dados nós já fornecemos a implementação da função `visualize_data()` que plota visualizações para as distribuições das variáveis de interesse, como o gráfico da *Figura 1*.

```python
def load_data(data_path='data/weatherAUS.csv')-> pd.DataFrame:
    """Funcao que importa dados de um arquivo csv, usando pandas"""
    #Seu código aqui
    return raw_data

def pre_processing(raw_data:pd.DataFrame)-> pd.DataFrame:
    """Funcao que filtra e limpa os dados meteorologicos para o treinamento"""
    #Seu código aqui
    return processed_data

def split_data(data:pd.DataFrame, val_size= 0.2)-> np.array:
    """Funcao que separa seus dados em teste e treino conforme a proporcao val_size"""
    #Seu código aqui
    return x_train, x_val, y_train, y_val
```

Montando a Arquitetura

Agora com os dados já pré-processados, você deve definir uma arquitetura para sua rede, compilá-la, treiná-la e avaliar sua performance.