Redes Neurais Convolucionais para Forecasting

gabrielstella28@gmail.com
Gabriel Felipe Dalla Stella

1 Introdução ao Problema

O conjunto de dados das temperaturas mínimas de Melbourne (dataset em Machine Learning Mastery) foi extraído do site (Encontrado Aqui).

Neste conjunto há uma série temporal contendo as datas e as temperaturas mínimas, em graus Celsius, da cidade de Melbourne, Australia, entre 1981 e 1990. O intuito desse projeto é apresentar um modelo de Rede Neural para utilizar forecasting.

Para realizar essa análise, a linguagem de programação utilizada foi Python, utilizando a IDE Jupyter Notebook. Além disso, foi utilizada as bibliotecas numpy e pandas (extração e manipulação de dados), matplotlib, seaborn (visualização), scikit-learn (manipulação e pré-processamento) e pytorch (redes neurais).

2 Pré-processamento

O único pré-processamento utilizado é a mudança de escala, realizada através da classe `MinMaxScaler()`, do módulo `sklearn.preprocessing`, nos dados de treino e aplicado nos dados de treino e teste.

Outra transformação realizada é tomar os exemplos da forma `(num_exemplos)` em dados de entrada da forma `(num_exemplos-tamanho_janela,1,tamanho_janela)` onde a primeira dimensão correspoderá aos exemplos, a segunda ao número de canais e a terceira à janela de observações utilizadas para prever o próximo valor. Os dados de saída serão os valores consecutivos à cada janela de observações e da forma `(num_exemplos-tamanho_janela,1)`.

Mais precisamente tomaremos um vetor `z` e transformaremos em um tensor de entrada `X` e um de saída `y` tais que

```
X[i,0,:]  = z[i:i+tamanho_janela],
   y[i,0] = z[i+tamanho_janela].
```

Nesse caso utilizamos uma janela de 365 dias. A escolha é devido à sazonalidade anual das temperaturas, portanto seria uma escolha razoável para tamanho de janela.

3 Rede Neural Convolucional para Forecasting

A divisão de dados de treino e teste é de 90%-10%. No apêndice 8 temos o a arquitetura exata da rede neural utilizada. Porém, de forma resumida, utilizamos uma rede neural com 4 camadas da seguinte forma:

Além disso, a camada de saída é a combinação linear da última camada.

Os motivos das escolhas são as seguintes:

- Convolução: Em cada entrada escolhemos aplicar a convolução primeiramente, para mesclar os dados antes de passar por filtros, assim como nas redes feedforward.
- Batch Normalization: Esta técnica servirá para facilitar o treinamento por, em geral, permitir uma convergência mais rápida dos parâmetros, pois normaliza a escala de cada convolução, deixando-as mais estáveis e menos propensas à problemas de redes neurais profundas, como *vanishing* e *exploding gradient*. Com isso também permite uma *learning rate* maior e assim também contribuindo para um treinamento mais rápido.
- GELU: É comum utilizar a função de ativação `RELU` em redes neurais profundas, porém problemas de gradiente que levam à estagnação ou ao treinamento mais lento preferi utilizar a função de ativação `GELU`. Outras funções de ativação com o mesmo aspecto também são interessantes, como a `SiLU`.
- Dropout: O dropout é uma regularização, onde definimos uma probabilidade de cada uma das saídas da ativação ser anulada. Esta técnica permite que a rede treine de forma mais uniforme os parâmetros da camada, além de inserir uma aleatoriedade nas previsões.
- AvgPool: O average pooling é um filtro que realiza a média em pequenos blocos da saída da camada. Assim transformamos uma grande janela de saída anterior em janelas menores. Outros filtros como max pooling ou min pooling também poderiam ser utilizados.

O otimizador utilizado foi o Adam com `learning_rate=0.1` e `weight_decay=1e-5`, onde `weight_decay` é o peso da penalidade ℓ_2. O otimizador Adam é um dos modelos com convergência mais rápida. Porém um treinamento muito prolongado ou convergência podem levar à um *overffiting*. Por esse motivo seria interessante o teste com outros otimizadores, porém algumas vezes pode não ser viável dependendo do tamanho dos dados e da rede neural.

O treinamento foi realizado com 101 épocas e `batch_size = 497`.

4 Desempenho do Modelo

À partir do modelo verificamos os seguintes desempenhos nos dados de teste:

Métrica	Valor
MSE	$7,40$
RMSE	$2,72$
MAE	$2,12$

Nesse caso vemos um erro médio entre 2 e 3 °C (a unidade pode ser encontrada na descrição do banco de dados aqui). Vemos também que a diferença entre MAE e o RMSE é pequena, portanto não há previsões tão discrepantes nos dados de teste. Podemos verificar isso na Figura 1 e Figura 2, onde apresentamos as previsões e os dados correspondentes e a distribuição dos resíduos, respectivamente.

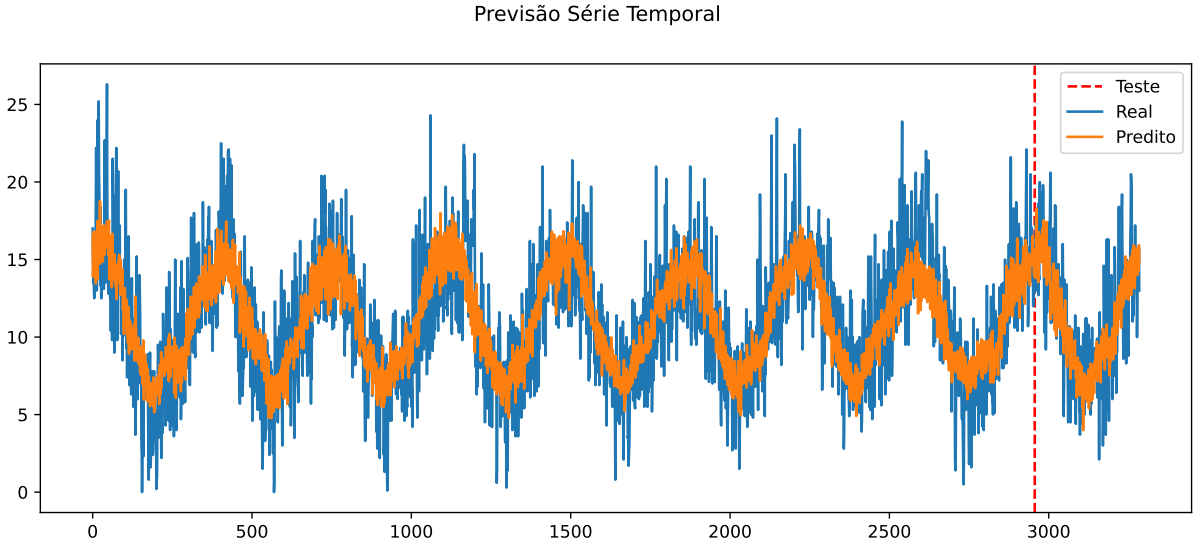

Figura 1: Previsão/Dados: A linha pontilhada delimita o início dos dados de teste

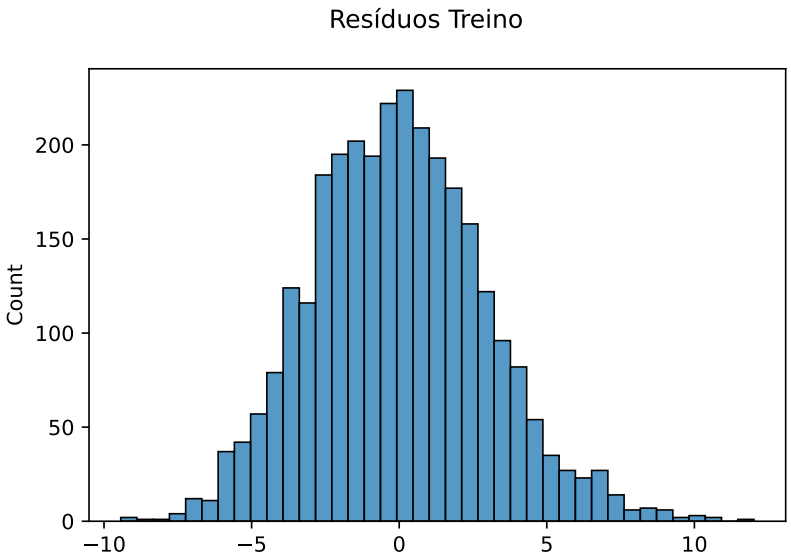

Figura 2: Distribuição de Resíduos

Nessas duas figuras vemos uma distribuição de resíduos quase normal, com leve tendência de resíduos mais altos.

5 Intervalo de Previsão - Bootstrap Resíduos

Muitos modelos de previsão de séries temporais utilizam métodos estatísticos para estimar os intervalos de previsão. No caso de redes neurais esta estimativa se torna inviável, devido à não-linearidade do modelo. Além disso, os métodos estatísticos mais básicos levam somente em conta os parâmetros do modelo e a variância da previsão, assumindo uma distribuição de resíduos normal, o que em muitos casos isso não é uma boa aproximação.

Tendo isso em vista, propomos a utilização do método bootstrap, que é uma forma de sintetizar novos dados à partir dos dados já existentes. Existem diversas técnicas de bootstrap para séries temporais, porém iremos simular diversas previsões para os dados de testes da forma `pred + res`, onde `pred` é a previsão do modelo e `res` é um resíduo escolhido aleatoriamente dos dados de treino.

Para a previsão do intervalo de confiança utilizamos os quartis 2.5% (limite inferior) e 97.5% (limite superior) dos dados simulados, em cada dia, e portanto esperamos uma cobertura de por volta de 95% dos valores da série. Além disso, devido à grande variabilidade do intervalo de confiança, realizamos uma convolução com um filtro de média de tamanho 20, semelhante aos filtros *blur* utilizado no processamento de imagens com ruído.

Na Figura 3 vemos os limites inferior e superior do intervalo de previsão, a mediana das simulações, todos suavizados, e os valores da série temporal.

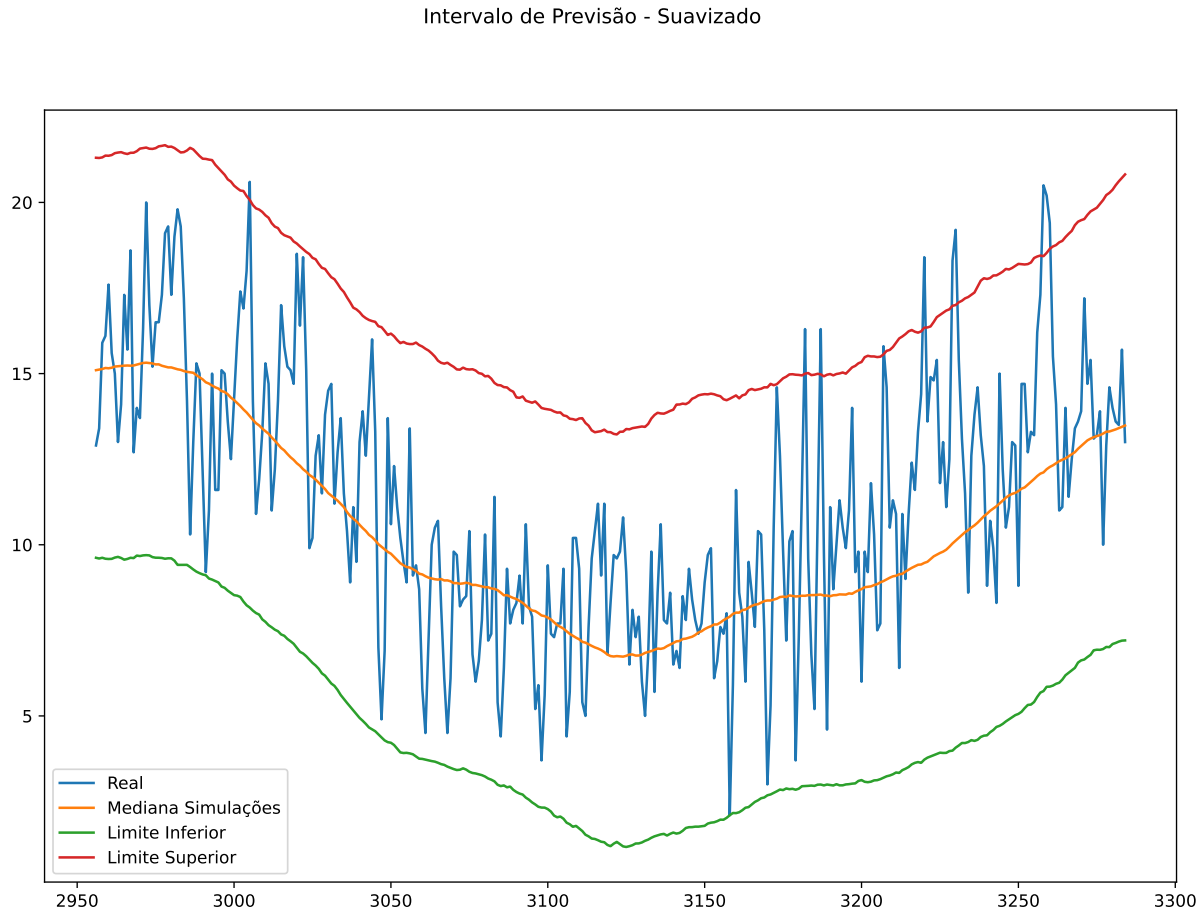

Figura 3: Intervalo de Previsão

5.1 Performance Intervalo de Previsão

Agora temos um intervalo de previsão e iremos avaliar seu desempenho na cobertura dos valores. Para isso iremos estimar a probabilidade p de uma previsão estar no intervalo de previsão.

Mais precisamente, supondo que Y_i é uma variável aleatória, onde $Y_i = 1$ se o valor de índice i da série está no intervalo de previsão e $Y_i = 0$ caso contrário, para cada i nos dados de teste. Então iremos supor que cada Y_i tem uma distribuição Bernoulli com probabilidade de sucesso p. Para estimar o valor de p utilizamos método do Wilson Score para estimar o intervalo de confiança de 95% de p.

À partir do teste, a chance p de um dado da série estar no intervalo de previsão está

no intervalo [0.9448,0.9835], com 95% de confiança. Ou seja, o intervalo de previsão tem uma cobertura de 94-98% nos dados, que é bastante satisfatório.

6 Previsões e Intervalo de Previsão

Realizando a previsão para os próximo 3 anos, a partir do dia do último dado, obtemos a Figura 4.

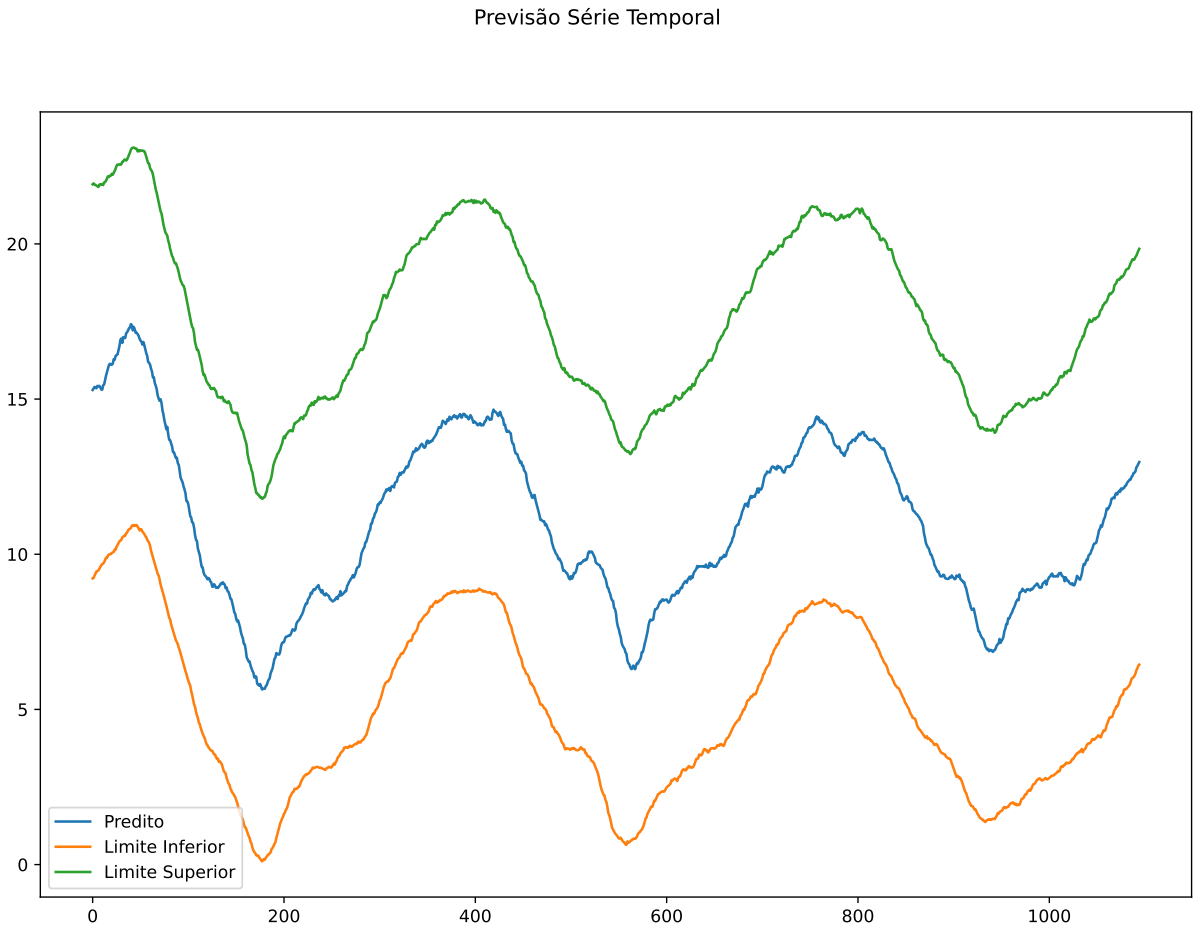

Figura 4: Intervalo de Previsão e Previsão para os próximo 3 anos

7 Conclusão

Neste projeto aprendemos a utilizar redes neurais convolucionais para previsão de séries temporais e também utilizar bootstrap nos resíduos para estimar o intervalo de previsão.

Com essa técnica conseguimos cobrir cerca de 95% dos valores com o intervalo de previsão, um resultado bastante interessante no forecasting.

8 Apêndice

A rede neural utilizada tem a seguinte arquitetura:

Camada 1:

```
(0): Conv1d(in_channels = 1, out_channels = 2, kernel_size=(5,), stride=(1,))

(1): BatchNorm1d(num_features = 2, eps=1e-05, momentum=0.1)

(2): GELU()

(3): Dropout(p=0.25, inplace=False)

(4): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
```

Camada 2:

```
(0): Conv1d(in_channels = 2, out_channels = 4, kernel_size=(5,), stride=(1,))

(1): BatchNorm1d(num_features = 4, eps=1e-05, momentum=0.1)

(2): GELU()

(3): Dropout(p=0.25, inplace=False)

(4): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
```

Camada 3:

```
(0): Conv1d(in_channels = 4, out_channels = 4, kernel_size=(5,), stride=(1,))

(1): BatchNorm1d(num_features = 4, eps=1e-05, momentum=0.1)

(2): GELU()

(3): Dropout(p=0.25, inplace=False)

(4): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
```

Camada 4:

```
(0): Conv1d(in_channels = 4, out_channels = 8, kernel_size=(3,), stride=(1,))

(1): BatchNorm1d(num_features = 8, eps=1e-05, momentum=0.1)

(2): GELU()

(3): Dropout(p=0.25, inplace=False)

(4): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
```

Camada Saída:

```
Linear(in_features=160, out_features=1, bias=True)</sub>
```