



Otávio Rodrigues de Oliveira

# Aprendizado de máquina utilizando análise multivariada de séries temporais

Projeto de Graduação em Computação

Santo André - SP

2019

Otávio Rodrigues de Oliveira

# Aprendizado de máquina utilizando análise multivariada de séries temporais

Projeto de Graduação apresentado ao Curso de Bacharelado em Ciências da Computação da Universidade Federal ABC, como requisito parcial para obtenção do título de Bacharel em Ciências da Computação.

Universidade Federal do ABC – UFABC

Centro de Matemática, Computação e Cognição

Curso de Bacharelado em Ciências da Computação

Orientador: Professor Doutor Denis Gustavo Fantinato

Santo André - SP

2019

# Sumário

# 1 Introdução

Uma série temporal pode ser descrita como um conjunto de observações feitas sequencialmente ao longo do tempo, cada uma sendo registrada em um momento específico do tempo $t$. Muitos eventos do dia-a-dia podem ser descritos como séries temporais, pois, na prática, podemos apenas fazer medições em intervalos discretos de tempo e que, muitas vezes, possuem uma relação com o estado anterior (logo, uma relação temporal). O modelamento dessas séries serve como uma forma de melhorar nosso entendimento dos mecanismos que a geram e pode nos prover desde um descrição compacta do dado, até uma forma de se controlar processos e prever eventos futuros (BROCKWELL; DAVIS, 2010).

Dentre as muitas técnicas utilizadas para realizar previsões baseadas em séries temporais, as chamadas Redes Neurais Artificiais demonstram ser promissoras dentro do campo de Aprendizado de Máquina na forma com que relações e padrões não-lineares que podem estar presentes nessas séries são representados em sua estrutura, levando a bons resultados na previsão de variáveis discretas e que não apresentam comportamento linear.

Uma parte muito importante do funcionamento de uma Rede Neural Artificial é o treinamento, na qual a partir de muitos exemplos já definidos é possível extrair as características de certa base de dados e prever futuros valores a partir de outros valores de entrada. Portanto, garantir que a rede seja bem treinada é um passo fundamental na modelagem de uma série temporal.

Dentro desse tema, é necessária uma forma de se medir a diferença entre o que está sendo previsto pela rede e os valores esperados para dada base de dados durante o treinamento, de forma a guiar os ajustes necessários em sua estrutura. No caso de séries temporais, o MSE (do inglês *Mean Square Error*) ou Erro Quadrático Médio é muito utilizado por ser simples, ter um significado físico claro, e já ser o método mais utilizado historicamente. Mas, apesar disso, quando se tratando de sinais "não clássicos"o MSE pode apresentar um desempenho pior em casos mais específicos, por não conseguir captar as características probabilísticas e incertezas do sinal (WANG; BOVIK, 2009).

Dada essa limitação, nesse projeto será explorada uma forma diferente da usual de realizar o treinamento de uma Rede Neural voltada ao processamento de séries temporais, buscando aproveitar técnicas utilizadas no campo do Processamento de Sinais, explorando as séries temporais como sinais aleatórios a fim de se melhorar a convergência do modelo com a distribuição real dos dados analisados. Para isso, será utilizado um critério de otimização chamado Casamento de Distribuições, que utiliza as informações estatísticas presentes na distribuição de uma série temporal a fim de gerar uma métrica de similiaridade entre dois sinais.

# 2 Justificativa

Modelos de séries temporais são muito utilizados hoje em dia e aplicáveis em diversas áreas e o uso de Redes Neurais Artificiais vêm se mostrando muito úteis nesse âmbito, tendo como exemplo os seguintes casos de uso:

- Previsão de tendências de valores no mercado financeiro;
- Monitoramento e previsão de condições climáticas;
- Controle de qualidade da produção de um produto na indústria.

Dentro desse tema, as *Recurrent Neural Networks* e subsequentemente a *Long Short-Term Memory*, ambas a serem explorados na Seção 3.1, estão chegando nas limitações do que pode ser modificado em sua estrutura para otimização do modelo. Logo, a busca por outros meios de melhorar a rede é uma proposta interessante para a ampliação do conhecimento existente na área de inteligência artificial.

Utilizar técnicas vindas da área de Processamento de Sinais e aplicá-las no âmbito do aprendizado de máquina é algo que proporciona uma aproximação das áreas e, a partir daí, aumenta a quantidade de técnicas disponíveis para resolução de problemas que, talvez, não pudessem ser totalmente explorados de maneira mais clássica.

Logo, esse projeto vem como uma forma de explorar o modelamento de séries temporais a partir de redes neurais por uma outra perspectiva, buscando maneiras mais efetivas de se realizar a otimização desses modelos e ampliar o conhecimento nessa área.

# 3 Objetivos

Esse projeto tem o objetivo de implementar o método de casamento de distribuições probabilísticas, para ser usado como critério de treinamento semi-supervisionado no modelamento de séries temporais. Com esse fim, será necessário implementar o modelo e comparar o método com o critério de treinamento mais utilizado atualmente, se preocupando com: fidelidade com o dado original e distribuição do dado de saída da rede.

## 3.1 Objetivos Específicos

Os objetivos podem ser separados em:

1. Elaborar um modelo base para processamento de séries temporais, utilizando o critério de avaliação padrão.
2. Implementar o casamento de distribuições univariado e integrar com o modelo base.
3. Criar um pipeline de testes para avaliar os critérios nas métricas mencionadas acima.
4. Adaptar o casamento de distribuições para o caso multivariado.
5. Testar o modelo com os diferentes critérios de avaliação utilizando diversas bases de dados.

# 4 Fundamentação Teórica

## 4.1 Long-Short Term Memory

Geralmente, tratando-se de séries temporais, um dos tipos de estruturas de processamento mais usados são as chamadas Redes Neurais Recorrentes (que será denotado por RNN, do inglês *Recurrent Neural Network*). Essa classe de rede neural é vantajosa para esse tipo de dados por ser uma estrutura que possibilita a reutilização do último resultado na computação do próximo valor, expressando uma dependência com o tempo, de forma a melhor modelar a entrada da rede (LIPTON; BERKOWITZ; ELKAN, 2015).

O problema das RNNs aparece quando se quer priorizar dados do passado baseado na distância ao tempo presente (longo, curto prazo), pois não se tem uma forma de quantizar a influência de eventos ocorridos em tempos diferentes. Torna-se necessário o uso de algum tipo de memória. Como uma forma de se resolver esse problema desenvolvou-se a chamada rede LSTM (do inglês *Long Short-Term Memory*)

A LSTM é um tipo específico de RNN, mantendo a estrutura de realimentação de resultados passados, mas com a adição de uma estrutura de memória denominada *memory cell*, que será denominada como célula de memória nesse projeto. O termo *Long Short-Term Memory* vem da intuição que RNNs simples possuem memória de longo prazo na forma de pesos, que vão mudando lentamente durante o treinamento, encapsulando conhecimentos gerais dos dados. Também possuem memória de curto prazo na forma de rápidas ativações dos nós da rede, passando de maneira sucessiva para os próximos nós. A LSTM introduz a essa estrutura um tipo intermediário de armazenamento de informações na forma de uma célula de memória, combinando essas duas propriedades, modificando a influência de cada uma na saída da rede(LIPTON; BERKOWITZ; ELKAN, 2015).

A estrutura de uma LSTM é descrita conforme a Figura 1:

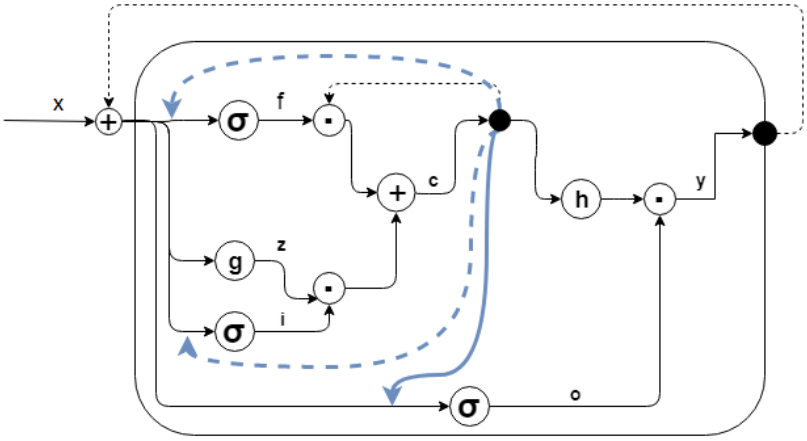


Figura 1 – Célula de LSTM

No diagrama é possível ver as conexões entre os elementos de uma célula de LSTM, onde as linhas tracejadas representam conexões com defasagem de tempo e as conexões azuis são as chamadas "*peephole conections*", que permitem que os elementos da LSTM consigam obter informações do estado da célula.

Considerando um vetor de entrada $x^t$ no instante de tempo $t$, $N$ o número de blocos LSTM e $M$ o número de entradas da rede, são descritos em uma célula de LSTM os seguintes pesos:

- Pesos de entrada: $W_z, W_i, W_f, W_o \in \mathbb{R}^{N \times M}$
- Pesos Recorrentes: $R_z, R_i, R_f, R_o \in \mathbb{R}^{N \times N}$
- Pesos "peephole": $p_i, p_f, p_o \in \mathbb{R}^N$
- Pesos dos bias: $b_z, b_i, b_f, b_o \in \mathbb{R}^N$

E a partir daí, se tem as seguintes equações para o denominado *foward pass* de uma LSTM, que representa o movimento da transformação do dado de entrada ao longo da célula até a saída da mesma:

- $\overline{z}^t = W_z x^t + R_z y^{t-1} + b_z$
- $z^t = g(\overline{z}^t)$
- $\overline{i}^t = W_i x^t + R_i y^{t-1} + p_i \odot c^{t-1} + b_i$
- $i^t = \sigma(\overline{i}^t)$
- $\overline{f}^t = W_f x^t + R_f yt - 1 + p_f \odot c^{t-1} + b_f$
- $f^t = \sigma(\overline{f}^t)$
- $c^t = z^t \odot i^t + c^{t-1} \odot f^t$

- $\overline{o}^t = W_o x^t + R_o y^{t-1} + p_o \odot c^t + b_o$
- $o^t = \sigma(\overline{o}^t)$
- $y^t = h(c^t) \odot o^t$

Onde $\sigma$, $g$ e $h$ são funções de ativação não-lineares, sendo $\sigma(x) = \frac{1}{1+e^{-x}}$, $g(x) = h(x) = tanh(x)$, $y^{t-1}$ a saída anterior da rede e $\odot$ o produto da matriz elemento a elemento. (GREFF et al., 2015).

### 4.1.1 Otimização da LSTM

O processo de treinamento de uma LSTM conta com uma série de iterações da base de dados sobre o modelo, atualizando os pesos das matrizes gradualmente, em função do resultado naquele momento comparado ao valor esperado dado um input conhecido. Para isso, existem diferentes tipos de critérios de otimização. O mais utilizado, como dito antes, é o MSE e é descrito pela fórmula abaixo:

$$MSE = \frac{1}{N}\sum_{t=1}^{N}(y^t - d^t)^2 \tag{4.1}$$

em que $y$ é o valor após a computação na LSTM e $d$ o valor esperado.

#### 4.1.1.1 Backpropagation Through Time

Após o *foward pass* e calculado o erro atual da rede, é possível fazer a atualização dos pesos da célula de LSTM — o "aprendizado"da rede — de forma a reduzir esse erro ao longo da rede a fim de convergí-la à representação real da série temporal. A essa etapa é dada o nome de *Backpropagation* e, para uma LSTM utilizando o erro quadrático médio como critério de avalição, tem-se as seguintes equações para os gradientes da propagação do erro, onde é utilizado não só o valor atual, mas também, valores precedentes da rede, demonstrando o *Backpropagation Through Time*:

- $\partial W_* = \sum_{t=0}^{T} \langle \partial *^t, x^t \rangle$
- $\partial R_* = \sum_{t=0}^{T} \langle \partial *^{t+1}, y^t \rangle$
- $\partial b_* = \sum_{t=0}^{T} \partial *^t$
- $\partial p_i = \sum_{t=0}^{T-1} c^t \odot \partial \overline{i}^{t+i}$
- $\partial p_f = \sum_{t=0}^{T-1} c^t \odot \partial \overline{f}^{t+i}$
- $\partial p_o = \sum_{t=0}^{T} c^t \odot \partial \overline{o}^t$

Nas quais, $*$ pode ser substituido por $\overline{z}, \overline{i}, \overline{f}, \overline{o}$ e $\langle \partial *_1, *_2 \rangle$ representa o produto vetorial de dois vetores (GREFF et al., 2015).

## 4.2 Estimação de distribuições multivariadas

Para se extrair as informações subjacentes aos dados de forma mais efetiva, será usado distribuições multivariadas. Entretanto, é necessário estimá-las. Um dos métodos de se estimar a distribuição de uma variável é a baseada em funções kernel, que pode ser definida como uma forma de se aproximar as observações de uma variável em relação a um ponto, ponderando-as e aplicando a função *kernel* escolhida (PRINCIPE, 2010). Ao se estender essa definição para o espaço infinito de valores da variável observada, tem-se a estimativa da função densidade de probabilidade (PDF, do inglês *Probability Density Function*) do sinal.

### 4.2.1 Método de Estimação por Kernel

A estimativa multivariada de uma PDF pode ser analisada de uma perspectiva temporal da seguinte maneira: assumindo que $x_t = [x_t \quad x_{t-1} \quad \dots \quad x_{t-M}]^T$ é o vetor composto do sinal $x_t$ no instante $t$ junto das suas $M$ versões atrasadas, associado ao vetor de coeficientes da variável aleatória $\underline{X} = \{X_n, X_{n-1}, \dots, X_{n-M}\}$. Logo, dada uma janela de $N$ vetores $\{x_1, \dots, x_N\}$, a estimativa da PDF multivariada $f_{\underline{X}}(v)$ é:

$$\hat{f}_{\underline{X}}(v) = \frac{1}{N} \sum_{i=1}^{N} \kappa_\sigma \left( \frac{v - x_i}{\sigma} \right), \tag{4.2}$$

onde $\kappa_\sigma(\cdot)$ é a função kernel multivariada e $\sigma$ é o tamanho do kernel (PRINCIPE, 2010).

Uma boa opção de função kernel a ser utilizada para esse projeto é a Gaussiana que pode ser definida matematicamente da seguinte maneira:

$$G_\Sigma (v - x_i) = \frac{1}{\sqrt{(2\pi)^{M+1} \det(\Sigma)}} exp \left[ \frac{-1}{2} (v - x_i)^T \Sigma^{-1} (v - x_i) \right], \tag{4.3}$$

onde $\Sigma = \sigma^2 I_{M+1}$ é a matriz de covariância e $\det(\cdot)$ é a operação determinante. Como é contínua e diferenciável, essa função kernel é muito útil para ser usada em técnicas de otimização em Redes Neurais Artificiais (FANTINATO, 2017).

## 4.3 Casamento de distribuições

Dado o caráter probabilístico de eventos e séries temporais, bem como sobre as incertezas devidas a erros de medida, imprecisões e/ou ruído, uma forma interessante de se modelar é tratá-las como sinais aleatórios, pela forma com que a distribuição da variável de estudo se comporta (variações e tendências). Dada essa modelagem, pode-se utilizar de técnicas herdadas da área de processamento de sinais para explorar os dados de entrada e saída de uma LSTM.

Uma característica importante de um sinal aleatório é sua função de densidade de probabilidade (ou como será usada a partir daqui PDF), pois, a partir dela pode-se extrair tendências e viéses que o dado pode apresentar ao longo do tempo e é possível obter uma aproximação dessa função utilizando métodos de estimativa de distruibuição.

E, a partir de sua distribuição, obtem-se a seguinte função custo para o treinamento:

$$\begin{aligned} J_{MQD} &= \int_D (f_{\underline{Y}}(v) - f_{\underline{S}}(v))^2 dv \\ &= \int_D f^2_{\underline{Y}}(v)dv + \int_D f^2_{\underline{S}}(v)dv - 2\int_D f_{\underline{Y}}(v) f_{\underline{S}}(v)dv \end{aligned} \tag{4.4}$$

sendo,

$$f_{\underline{S}}(v_i) = \frac{1}{T_s}\sum_{j=0}^{N_s-1} G_{\sum}(v_i - s_{t-j}) \tag{4.5}$$

em que $G_{\sum}$ é a função kernel (definida na equação (4.3)) e $N_s$ a quantidade de elementos da série.

A equação (4.6) é obtida substituindo a equação (4.5) na (4.4) descrevendo uma função custo para uma série temporal:

$$\begin{aligned} J_{MQD-C} =& \frac{1}{N_y^2}\sum_{i=0}^{N_y-1}\sum_{j=0}^{N_y-1} G_{2\sum}(y_{n-i} - y_{n-j}) + \frac{1}{N_s^2}\sum_{i=0}^{N_s-1}\sum_{j=0}^{N_s-1} G_{2\sum}(s_{t-i} - s_{t-j}) \\ &- \frac{2}{N_y N_s}\sum_{i=0}^{N_y-1}\sum_{j=0}^{N_s-1} G_{2\sum}(y_{n-i} - s_{t-j}) \end{aligned} \tag{4.6}$$

O objetivo é minimizar essa função custo, tendo seu valor mínimo quando a distribuição dos dois sinais é igual ($J = 0$). Diferentemente do MSE, esse valor não é calculado amostra a amostra, logo não é necessário saber os valores exatos de $s_t$, apenas saber a distribuição de $f_S$, utilizando o custo de forma semi-supervisionada.

# 5 Metodologia

A abordagem deste projeto é de natureza quantitativa. A LSTM e o algoritmo do critério de avaliação da rede serão desenvolvidos utilizando a linguagem de programação Python, além da biblioteca Pytorch.

Antes de começar a implementar a rede, será adotado o método *Backpropagation Through Time* para o caso do uso do casamento de distribuições como critério de otimização da rede, isso envolve calcular os gradientes da rede através da derivação da função custo sobre os seus coeficientes.

No método proposto haverá uma célula de uma LSTM que será modificada a cada iteração utilizando o critério de avaliação proposto, comparando a saída da célula com o valor esperado da base de dados e aplicando as modificações necessárias no modelo.

Para medir o impacto da mudança na avaliação do modelo durante o treinamento será feita a mesma análise em cima da base de dados utilizando o critério MSE e, a partir daí, será feita uma comparação dos resultados, com foco na acurácia dos dois métodos.

## 5.1 Implementação

### 5.1.1 Pytorch

Como descrito antes, a implementação do casamento de distribuições foi realizada utilizando a biblioteca PyTorch, usada em aplicações de *Deep Learning*. Ela consiste em um conjunto de operações envolvendo um conceito geométrico chamado Tensor.

O Tensor é um simples objeto matemático que pode ser descrito como uma matriz (podendo o mesmo ser um vetor/matriz de 0, 1, 2 ou $n$ dimensões). Sendo seu *rank* o número de dimensões que o compõe, *e.g.* um Tensor de *2* possui 2 dimensões. Esse tipo de representação é útil devido a representação de dados utilizados em *Deep Learning* (*e.g.* uma imagem pode ser representada por uma matriz de 2 dimensões, sendo cada "ponto"dessa matriz um vetor de tamanho 3, logo um Tensor de *rank 3*).

Junto com as operações tensoriais já implementadas, a biblioteca oferece algo muito importante para a execução do *backpropagation*, o denominado *Autograd*. Ele é um pacote integrado ao PyTorch para a diferenciação automática de todas operações envolvendo Tensores, não sendo necessária a diferenciação manual das operações realizadas no treinamento da rede, contanto que todo o processo seja feito com operações diferenciáveis.

### 5.1.2 Implementação da LSTM

A biblioteca PyTorch já possui uma implementação de uma célula de LSTM, sendo necessário apenas passar alguns parâmetros para a inicialização da rede. São eles: tamanho da entrada (*i.e.* a dimensão dos dados que vão entrar na rede, quantidade de variáveis), tamanho da *hidden layer* (o número de pesos utilizados dentro de uma camada), número de camadas e outros parâmetros adicionais de otimização não explorados nesse projeto (utilizando os valores padrão da biblioteca).

Inicializada uma célula de LSTM, a rede já é capaz de receber dados e obter uma saída.

### 5.1.3 Casamento de Distribuições

#### 5.1.3.1 Caso Univariado

Primeiramente, para se testar a implementação, foi-se estudado o caso do casamento de distribuições univariadas, onde a comparação é feita apenas utilizando o vetor original, sem a expansão de suas versões atrasadas. O código é descrito abaixo:

```
import torch

class CasamentoLoss(torch.nn.Module):
    def __init__(self, sig=1.):
        super(CasamentoLoss, self).__init__()
        self.sig = sig
        self.sqrt_pi = math.sqrt(2. * math.pi)

    def forward(self, d, y):
        d = d.unsqueeze(0)
        y = y.unsqueeze(0)

        # d - y
        combined = -2 * self.get_component(d, y)

        # y - y
        same_y = self.get_component(y, y)

        # d - d
        same_d = self.get_component(d, d)
```

```
        return same_y + same_d + combined

    def get_component(self, y1, y2):
        rows = y1.t().repeat(1, y2.shape[1])
        cols = y2.repeat(y1.shape[1], 1)
        res = (rows - cols) / self.sig

        part_exp = torch.exp(-1/2. * d_int * d_int)
        gaussian = part_exp * 1/(self.sig * self.sqrt_pi)

        return torch.sum(gaussian) / (y1.shape[1] * y2.shape[1])
```

Para aproveitar o uso do Autograd, o novo componente precisa ser herdado da classe *torch.nn.Module* e possuir um método *forward*, onde será descrito seu funcionamento. Com isso, o método *backward* herdado realiza o *backpropagation*.

A avaliação da saída da rede foi dividida em 3 componentes, seguindo a equação 4.4, um componente comparando o valor esperado ($d$) com o valor obtido ($y$), outro comparando o valor esperado com ele mesmo e, por último, uma comparação do valor obtido sobre ele mesmo.

Essa comparação é feita através do método *get_component*, que requer dois vetores e retorna um valor de acordo com a similaridade dos dois. O que está acontecendo nesse momento é o cálculo da diferença entre cada elemento de um vetor e os elementos do outro, um por um. De forma a otimizar esse processo, é utilizado a função *repeat* presente na definição de um Tensor de forma a replicar o mesmo ao longo de uma ou mais dimensões. Com isso, tem-se um vetor replicado ao longo de uma dimensão e o outro ao longo da contrária. A partir daí, é calculada a diferença entre as duas matrizes resultantes, resultando em uma matriz de diferenças. Em cima desse resultado é aplicado o kernel gaussiano, multiplicando *elementwise* (elemento por elemento) as diferenças, obtendo o expoente da função exponencial e dividindo por $\sigma\sqrt{\pi}$. Finalmente, se tem a soma da matriz inteira e a divisão desse valor pelas dimensões dos dois vetores a fim de se obter um resultado único e normalizado.

Após testes de treinamento da LSTM utilizando o caso univariado e, obtendo a convergência da rede nos mesmos, o próximo passo foi a implementação do casamento de distribuições multivariadas. Trata-se de uma abordagem particulamente interessante, pois uma análise dos vetores em instantes diferentes no tempo, revelaria diferentes relações temporais entre os vetores.

#### 5.1.3.2 Caso Multivariado

A diferença na implementação do caso univariado para o caso multivariado vem da representação do Tensor e das operações realizadas em cima do mesmo, acrescentando uma dimensão a mais, já que a comparação não seria realizada em cima de apenas um vetor, mas, sim, em um conjunto de vetores. O código para essa implementação é descrito abaixo:

```
class CasamentoMult(torch.nn.Module):
    def __init__(self, sig=.5, m=5):
        super(CasamentoMult, self).__init__()
        self.sig = sig
        self.m = m

    def forward(self, d, y):

        d = self.toeplitz_like(d, self.m)
        y = self.toeplitz_like(y, self.m)

        self.sqrt_pi = math.sqrt(((2 * math.pi) ** y.shape[1]) *
                        (self.sig ** (2 * y.shape[1])))

        d = d.unsqueeze(0)
        y = y.unsqueeze(0)

        # d - y
        combined = -2 * self.get_component(d, y)
        # y - y
        same_y = self.get_component(y, y)
        # d - d
        same_d = self.get_component(d, d)

        return same_y + same_d + combined


    def get_component(self, y1, y2):
        d_int = (torch.transpose(y1, 0,1).repeat(1, y2.shape[1], 1)
                 - y2.repeat(y1.shape[1], 1, 1))

        mid = (1 /self.sig ** 2) * d_int * d_int
```

```
        gaussian = torch.exp(-1/2. * mid) / self.sqrt_pi
        return torch.sum(gaussian) / (y1.shape[1] * y2.shape[1])

    def toeplitz(self, v):
        c = v.view(-1)
        vals = torch.cat((torch.flip(c, [0]), c[1:]))
        a = torch.arange(c.shape[0]).unsqueeze(0).t()
        b = torch.arange(c.shape[0] - 1, -1, step=-1).unsqueeze(0)
        indx = a + b

        return vals[indx]

    def toeplitz_like(self, x, n):
        r = x
        stop = x.shape[0] - 1

        if n < stop:
            stop = n
        else:
            stop = 2

        r = self.toeplitz(r)

        return r.narrow(1, 0, stop)
                .narrow(0, stop - 1, r.shape[0] - stop)
```

A estrutura segue os mesmos princípios do caso unviariado: herdeiro da classe *torch.nn.Module*, seguindo a equação 4.6 e dividindo a operação entre componentes dos valores. O que muda entre as duas implementações é a forma da entrada do método *get_component*, recebendo dois conjuntos de vetores ao invés de vetores individuais.

É esperado um conjunto de vetores, composto do vetor original e suas $M$ versões atrasadas. Para isso, é utilizada uma função chamada *toeplitz*, para construir uma matriz de diagonais constantes, obtendo o mesmo vetor defasado em uma unidade a mais a cada coluna da matriz. Como a mesma não está implementada na biblioteca PyTorch, foi necessário realizar a implementação da mesma utilizando manipulações em uma matriz a partir de seus índices. Com a matriz resultante, é realizado um corte nas $M$ primeiras colunas e excluindo as $M$ primeiras linhas da mesma, de forma a representar melhor os vetores atrasados no tempo.

Após isso, o método *get_component* segue o mesmo princípio de comparação.

Como as operações do PyTorch são aplicadas de uniformemente ao longo do Tensor, é possível utilizar as operações do caso univariado (Note que o caso multivariado com $M = 0$ é igual ao caso univariado).

## 5.2 Bases de Dados

A avaliação do modelo foi realizada usando 4 bases de dados reais: *Beijing PM2.5*[1], *Bike Sharing* [2], obtidas do repositório de aprendizado de máquina UCI. E *NSW2016* e *TAS2016*, disponíveis no site da agência de energia australiana AEMO. Para gerar o dado anual, foram concatenados os dados mensais de preço e demanda de eletricidade, para as regiões de New South Wales e Tasmania [3]. Apresentados na Tabela 1 estão o tamanho e número de variáveis de cada uma das bases de dados.

Tabela 1 – Bases de dados utilizadas

| Base de Dados | Tamanho | Qtd. Variáveis |
|---|---|---|
| Beijing PM2.5 | 43824 | 13 |
| Bike Sharing | 17379 | 17 |
| NSW 2016 | 17568 | 5 |
| TAS 2016 | 17568 | 5 |

Todas as bases são representações reais de séries temporais e para os modelos a serem treinados para cada uma delas terá como objetivo a previsão de valores relacionados a cada uma delas. O objetivo será o treinamento de modelos de LSTMs para a previsão do comportamento específico de cada dado. Para a base *Beijing PM 2.5* será a previsão do índice do poluente PM 2.5 na cidade de *Beijing*, para *Bike Sharing* a previsão do número de aluguéis de bicicletas em determinado período e para as bases *NSW2016* e *TAS2016* a previsão da demanda de energia elétrica para as duas regiões.

## 5.3 Modelo e Treinamento

### 5.3.1 Parâmetros da Rede

De maneira a simplificar a comparação e padronizar os experimentos, foi utilizado uma mesma inicialização para a LSTM a ser treinada utilizando cada método e base. É

[1] https://archive.ics.uci.edu/ml/datasets/Beijing+PM2.5+Data
[2] https://archive.ics.uci.edu/ml/datasets/bike+sharing+dataset
[3] https://www.aemo.com.au/Electricity/National-Electricity-Market-NEM/Data-dashboard#aggregated-data

descrito na tabela 2 os parâmetros de inicialização.

Tabela 2 – Parâmetros de inicialização da LSTM

| | |
|---|---|
| Nº de épocas | 20 |
| Batch Size | 32 |
| Nº de camadas | 1 |
| Tamanho das camadas | 64 |
| Learning Rate | $10^{-4}$ |
| Lag | 24 |

**Número de épocas**: É um parâmetro de treinamento que indica quantas vezes a base de dados inteira realizará o movimento de ida (*foward pass*) e depois o de volta (*backward pass*) dentro do modelo.

***Batch Size***: Como geralmente as bases de dados são muito grandes, o *foward pass* acaba sendo muito custoso no treinamento (tanto em tempo, quanto em memória). Logo, tem-se pequenos movimentos de ida e volta com uma fração da base de cada vez, sendo o tamanho dessa divisão o *Batch Size*. Após $\frac{N}{Batch\ Size}$ iterações se tem uma época, sendo $N$ o tamanho da base de dados.

**Número de camadas**: Representa a quantidade de camadas de presentes em uma célula de LSTM, geralmente cada camada representa um nível de não-linearidade a mais para o modelo.

**Tamanho de camadas**: Representa a quantidade de parâmetros dentro de cada camada da célula.

**Learning Rate**: Um parâmetro de treinamento que representa a taxa de mudança dos valores dos parâmetros dentro de uma camada.

**Lag**: Indica o intervalo de tempo entre o valor presente e o valor predito pela rede, *i.e.* quão futura será a previsão.

### 5.3.2 Pré-Processamento

Primeiramente foi realizada a normalização dos dados de cada base. Para isso, foi utilizada uma técnica de redimensionamento dos dados relativo a seu mínimo e máximo, restringindo-os a um intervalo de 0 a 1.

Devido a relação temporal dos dados, as bases foram apenas divididas em duas partes, um conjunto de treino e outro de teste, com o primeiro precedendo temporalmente o segundo. Foram separados 75% do total como dados de treino e 25% como dados de teste.

Logo após, foi gerado o conjunto de dados de entrada e de saída ($X$ e $d$). Isso foi feito copiando o sinal de entrada e defasando-o em $k$ unidades de tempo, sendo $k$ o parâmetro *lag* da rede, obtendo, assim, um sinal no "presente"que será o X e um sinal adiantado, o valor esperado na saída da rede, representado por $d$ (*i.e.* o vetor $X = [x(0), x(1), ..., x(n)]$ e $d = [x(k), x(k+1), ..., x(n+k))$.

### 5.3.3 Treinamento

Uma instância de treinamento consistiu na escolha de uma base e um critério de avaliação. Então, para cada um deles, os dados de treinamento foram divididos em frações menores, seguindo o parâmetro *Batch Size*.

Cada iteração consiste em um movimento de ida do dado dividido dentro do modelo, gerando uma saída $y$. É feita uma comparação com o valor esperado $d$, utilizando o critério de avaliação e salvando o valor obtido. Concluindo o ciclo, é realizado o movimento de volta dentro do modelo, o *Backpropagation*, atualizando os pesos da rede.

Após a esse ciclo ser realizado para todas as divisões da base, se tem uma época. E, depois de completadas $e$ épocas (sendo $l$ o número de épocas da inicialização), o treinamento acaba.

No final do treinamento são gerados gráficos da evolução do modelo ao longo das épocas, para comparar o desempenho em tempo de treinamento dos critérios.

### 5.3.4 Teste

Com o modelo treinado, é possível gerar algumas métricas de acurácia com o conjunto de testes. As métricas utilizadas foram as seguintes:

- **MAE**: *Mean Absolute Error*

$$MAE = \frac{\sum_{i=1}^{n} |d_i - y_i|}{n} \tag{5.1}$$

- **RMSE**: *Root Mean Square Error*

$$RMSE = \sqrt{MSE} \tag{5.2}$$

- **SMAPE**: *Symmetric Mean Absolute Percent Error*

$$SMAPE = \frac{100\%}{n} \sum_{i=1}^{n} \frac{|y_i - d_i|}{(|d_i| + |y_i|)/2} \tag{5.3}$$

Utilizando o conjunto de testes como entrada do modelo, é gerada uma saída, denominada $y$. E, comparando com o valor esperado $d$ do conjunto de testes, utilizando as métricas acima, foi gerado os resultados para uma instância de treinamento.

## 5.4 Resultados

*A priori*, foram geradas 8 instâncias de treinamento, uma para cada base e cada critério de avaliação da rede, gerando os resultados confome Tabelas 3 e 4.

Tabela 3 – Primeiros Resultados - *Beijing PM 2.5* e *Bike Sharing*

| | Beijing PM2.5 | | Bike Sharing | |
|---|---|---|---|---|
| | MSE | Casamento | MSE | Casamento |
| MAE | 14.20 | 13.95 | 37.87 | 53.35 |
| RMSE | 25.01 | 24.84 | 57.00 | 78.90 |
| SMAPE | 26.10 | 26.16 | 30.00 | 47.30 |

Tabela 4 – Primeiros Resultados - *NSW 2016* e *TAS 2016*

| | NSW 2016 | | TAS 2016 | |
|---|---|---|---|---|
| | MSE | Casamento | MSE | Casamento |
| MAE | 72 | 253.27 | 14.89 | 16.03 |
| RMSE | 92.00 | 318.86 | 21.01 | 22.36 |
| SMAPE | 0.96 | 3.48 | 1.42 | 1.53 |

Diante desses resultados, pode-se ver que dentre as bases, a *Beijing PM 2.5* foi a que o casamento de distribuições desmontrou um ganho em cima do MSE, logo após a TAS 2016. Com o resultado das outras bases pode-se ver que o casamento não é uma boa escolha para esses casos, oferecendo um resultado muito pior que o MSE. Um motivo para essa variação nos resultados é o Casamento de distribuições ser um método semi-supervisionado, não sendo necessário saber os valores ponto a ponto de um sinal, apenas sua distribuição. Logo em um caso supervisionado, como no treinamento da LSTM para essas bases de dados, o MSE obtêm resultados mais consistentes.

Então, para estudar melhor o desempenho da LSTM e do Casamento de distribuições, o foco dos próximos testes será na base *Beijing PM 2.5*.

Os próximos testes foram realizados com o intuito de ver o desempenho da rede e do critério de avaliação diante de diferentes valores dos parâmetros, começando com o *Lag* da rede, conforme mostra a Tabela 5.

Tabela 5 – Mudança do parâmetro *Lag* - *Beijing PM 2.5*

| Beijing PM2.5 | | | | |
|---|---|---|---|---|
| | Lag = 24 | | Lag = 5 | |
| | MSE | Casamento | MSE | Casamento |
| MAE | 14.20 | 13.95 | 13.12 | 12.81 |
| RMSE | 25.01 | 24.84 | 24.81 | 24.97 |
| SMAPE | 26.10 | 26.16 | 21.79 | 20.92 |

O valor inicial desse parâmetro, 24, foi escolhido devido a base, na qual os registros foram realizados a cada uma hora. Logo ao utilizar um intervalo de previsão de vinte e quatro horas, tem-se a previsão do próximo dia. Mas, como se tem um desempenho melhor num período menor de tempo, foi escolhido o período de cinco horas para minimizar grandes variações no treinamento da rede.

Como observado na Tabela 5, um *Lag* de 5 obteve um desempenho melhor nos dois critérios de treinamento, logo é o valor do parâmetro a ser utilizado.

Tabela 6 – Mudança do parâmetro Tamanho de camadas - *Beijing PM 2.5*

| Beijing PM2.5 | | | | | | |
|---|---|---|---|---|---|---|
| | Hidden = 8 | | Hidden = 32 | | Hidden = 64 | |
| | MSE | Casamento | MSE | Casamento | MSE | Casamento |
| MAE | 12.75 | 15.02 | 13.12 | 12.81 | 13.31 | 13.33 |
| RMSE | 24.54 | 26.21 | 24.81 | 24.97 | 25.10 | 24.88 |
| SMAPE | 20.54 | 23.86 | 21.79 | 20.92 | 22.56 | 21.22 |

Variou-se também o tamanho da camada, conforme Tabela 6. Para o MSE, teve-se um melhor resultado utilizando uma camada de tamanho 8. Enquanto isso, para o casamento uma de 32 acabou se saindo melhor. Logo, esses serão os valores utilizados para os próximos testes.

Tabela 7 – Mudança do parâmetro Tamanho de camadas - *Beijing PM 2.5*

| Beijing PM2.5 | | | | | | |
|---|---|---|---|---|---|---|
| | Learning Rate = $10^{-5}$ | | Learning Rate = $10^{-4}$ | | Learning Rate = $10^{-3}$ | |
| | MSE | Casamento | MSE | Casamento | MSE | Casamento |
| MAE | 12.75 | 12.81 | 14.70 | 13.04 | 13.37 | 15.04 |
| RMSE | 24.54 | 24.97 | 25.32 | 24.60 | 24.48 | 25.22 |
| SMAPE | 20.54 | 20.92 | 27.70 | 22.21 | 27.95 | 26.94 |

A tabela 7 mostra a variação da *Learning Rate*. Com essa mudança tem-se que o valor de $10^{-5}$ é o melhor valor para o treinamento da rede em ambos os casos. Como esse foi um caso de rápida convergência, o uso de uma *Learning Rate* menor ocasionou em um resultado melhor, já que são realizadas mudanças menos bruscas, mais refinadas ao longo do treinamento.

Tabela 8 – Mudança do $\sigma$ do Casamento de Distribuições - *Beijing PM 2.5*

| Beijing PM2.5, Lag = 5, hidden = 32 | | | | | |
|---|---|---|---|---|---|
| | $\sigma = 1$ | $\sigma = 0.75$ | $\sigma = 0.6$ | $\sigma = 0.5$ | $\sigma = 0.25$ |
| MAE | 12.81 | 12.80 | 12.89 | 13.33 | 13.39 |
| RMSE | 24.97 | 24.85 | 24.80 | 24.70 | 25.27 |
| SMAPE | 20.92 | 20.24 | 21.66 | 22.68 | 23.15 |

Logo após, foi-se estudado o efeito da mudança do valor $\sigma$ do critério de avaliação, conforme tabela 8. Pode-se enxergar que geralmente busca-se um valor médio para esse parâmetro, já que ao mesmo tempo que um valor de $\sigma$ alto é mais tolerante a uma maior variação na distribuição do dado, a rede passa a ser menos precisa nos valores mais frequentes. O contrário também vale, fazendo a estimativa ser mais precisa nos picos da distribuição, ignorando valores menos frequentes. Nos testes, o valor 0.75 foi escolhido porque obteve um resultado melhor e está perto da média dos valores. Logo, será o valor utilizado nos próximos testes.

Outro parâmetro do Casamento de distribuições estudado foi o valor de $M$, que representa a quantidade de vetores atrasados (de 0 a $M$ unidades de tempo) utilizada nesse critério de treinamento.

Tabela 9 – Mudança do $M$ do Casamento de Distribuições - *Beijing PM 2.5*

| Beijing PM2.5, Lag = 5, hidden = 32 | | | | |
|---|---|---|---|---|
| | M = 2 | M = 5 | M = 10 | M = 15 |
| MAE | 13.98 | 12.80 | 13.17 | 14.40 |
| RMSE | 24.35 | 24.85 | 24.62 | 24.61 |
| SMAPE | 26.69 | 20.24 | 21.71 | 26.02 |

Observando a tabela 9, pode-se percerber que um valor próximo do *Lag* da rede ocasiona em uma performance melhor, já que o vetor é comparado dentro o intervalo de previsão da rede. Também é possível observar que valores mais altos para esse parâmetro pode ocasionar em uma perda de precisão, provavelmente, devido a comparações com muitos vetores diferentes, ocasionando em um ruído no resultado. Enquanto isso, para valores mais baixos de $M$ é possível que seriam necessários mais amostras de tempo para

capturar melhor as relações temporais do dado. Logo, o valor de 5 foi escolhido para esse parâmetro.

Abaixo foi aplicada uma técnica conhecida como decomposição de uma série temporal. Ela consiste em decompor a série temporal em componentes: sistemáticos, *i.e.* que são consistentes ou possuem uma recorrência e conseguem ser descritos e modelados; e um não-sistemáticos, que não podem ser diretamente modelados. Para isso, a série foi decomposta em dois sistemáticos: Tendência, se a série está crescendo ou decrescendo em dado momento; e Sazonalidade, ciclos que ocorrem em um curto espaço de tempo. Também decomposta em um não-sistemático: Ruído, variação aleatória na série. Com essa decomposição e passando cada elemento separado na LSTM foram obtidos os resultados exibidos na Tabela 10.

Tabela 10 – Decomposição da série temporal - *Beijing PM 2.5*

| Beijing PM2.5, Lag = 5, hidden = 32 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Tendência | | Ruído | | Sazonalidade | | Total | |
| | MSE | Casamento | MSE | Casamento | MSE | Casamento | MSE | Casamento |
| MAE | 2.43 | 2.30 | 10.43 | 10.76 | 0.0003 | 0.0011 | 11.04 | 11.18 |
| RMSE | 3.15 | 3.13 | 18.01 | 18.24 | 0.0003 | 0.0013 | 18.07 | 18.46 |
| SMAPE | 6.15 | 5.82 | 130.90 | 132.58 | 0.0685 | 0.3807 | 21.73 | 22.70 |

Analisando os resultados da tabela 10, pode-se perceber que o Casamento de distribuições só obeteve um desempenho melhor no componente de Tendência. Esse teste foi interessante pois pode-se perceber que é possível modelar o dado dessa maneira e realizar o treinamento utilizando o Casamento de Distribuições e se obter a convergência da rede.

### 5.4.1 Análise Gráfica dos Resultados

Utilizando o dataset *Beijing PM 2.5* e os parâmetros ótimos achados previamente, foram gerados os seguintes histogramas, onde Casamento e MSE representa a saída da LSTM para cada um dos dois critérios e o valor esperado representa o a saída esperada da rede:

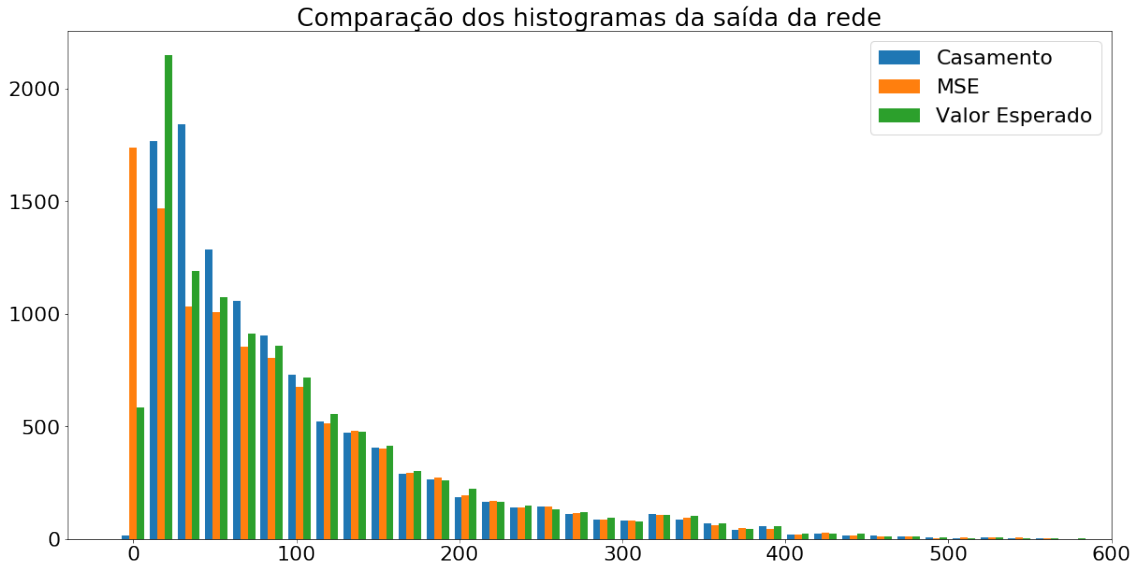


Figura 2 – Comparação dos histogramas resultantes do treinamento completo da rede

Para o treinamento completo, é possível observar que apesar dos dois métodos possuirem bastante divergencia do valor esperado no começo da distribuição, a MSE acaba errando mais para os valores mais frequentes.

Analisando esses resultados, é possível perceber que o Casamento de Distribuições age de acordo com o esperado, obtendo uma distribuição mais próxima à da série original. Enquanto isso, o MSE é asertivo quando se tratando apenas dos valores puros, mas podem ocorrer divergências na distribuição da saída da rede.

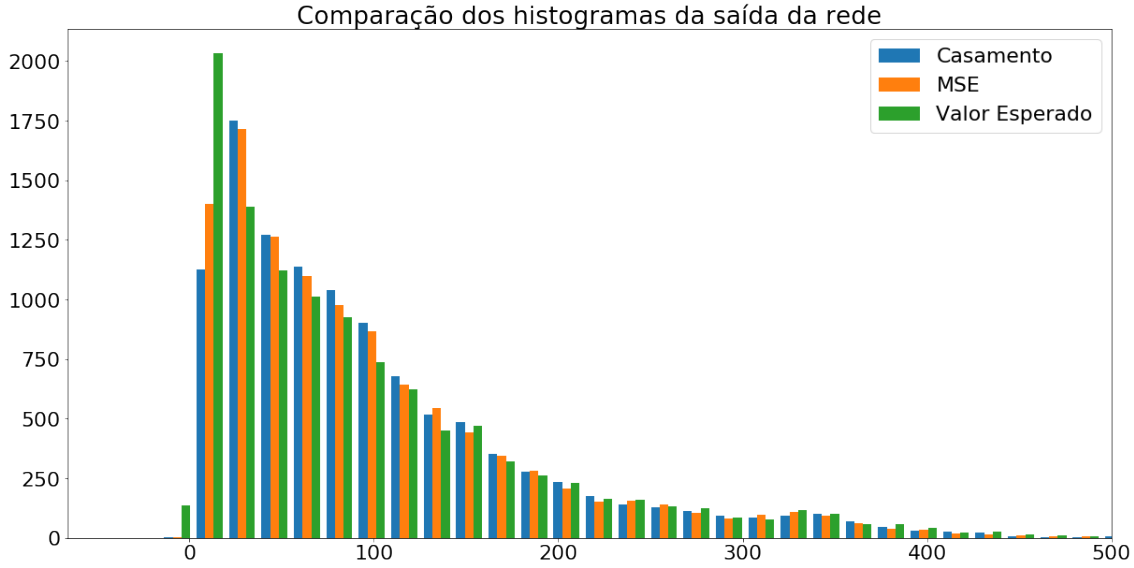


Figura 3 – Comparação dos histogramas resultantes do treinamento da rede utilizando a decomposição

Utilizando a decomposição o MSE apresentou um resultado melhor que o Casa-

mento, isso pode ter acontecido pela forma com que a decomposição foi realisada (sendo uma soma das três componentes), já que cada uma tem uma distribuição diferente, fazendo com que o Casamento de Distribuições fique em desvantagem em comapração a um método que avalia valores absolutos de cada uma.

# 6 Considerações Finais

Este projeto consistiu no estudo e implementação de um modelo de séries temporais utilizando uma LSTM com o casamento de distribuições como critério de treinamento.

O casamento de distribuições é uma métrica utilizada em Processamento de Sinais como uma forma de comparar dois sinais de forma a levar em conta sua natureza probabilística. Logo testar essa abordagem para dados temporais e em um campo diferente do conhecimento, Redes Neurais Artificiais, consistiu em desafio interessante.

Para as bases de dados analisadas, o casamento de distribuições teve um desempenho razoável comparado ao MSE. Com exceção da base *NSW2016*, ele teve um desempenho parecido e, no caso da *Beijing PM 2.5* um pouco superior ao MSE. Na análise gráfica dos dois casos destacados, o efeito da comparação feita em cima da distribuição (ao invés dos valores únicos de cada série) é perceptivel, pois tem-se que a saída da rede tende a seguir mais a frequência dos valores esperados.

Pelos resultados é realista dizer que o casamento de distribuições pode ser uma alternativa ao MSE no treinamento de uma LSTM para séries temporais. Um estudo mais aprimorado da rede e seus parâmetros pode levar a uma melhora do resultado de saída, mudando de caso para caso.

Quanto a escolha do critério de treinamento para a rede, o MSE ainda parece uma escolha mais interessante *a priori*, já que é simples, muito explorado e por ser um método padrão no treinamento de Redes Neurais Artificiais. É possível que existam casos nos quais o casamento de distribuições possua um desempenho muito melhor que justificaria seu uso, mas, para esse objetivo, é necessário um estudo mais aprofundado.

## 6.1 Trabalhos Futuros

O próximo passo desse projeto seria a exploração de outras bases de dados para tentar definir quais as propriedades que uma série temporal deve ter para casamento de distribuições ser um método mais eficaz, no treinamento de uma LSTM e estudar melhor como aproveitar a decomposição de uma série temporal, relacionando aos diferentes critérios para cada fator resultante. Adicionalmente, analisar mais cuidadosamente a composição do vetor de atributos e os efeitos da mesma sob o desempenho do critério de treinamento. Por fim, estudar a arquitetura da LSTM a fim de se encontrar otimizações que provém um melhor desempenho para o casamento de distribuições.

# Referências


BROCKWELL, P. J.; DAVIS, R. A. *Introduction to Time Series and Forecasting (Springer Texts in Statistics)*. [S.l.]: Springer, 2010. ISBN 0387953515. Citado na página 5.

FANTINATO, D. G. *Novas Metodologias de Aprendizado Baseado na Teoria da Informação para Equalização Adaptativa*. Campinas: [s.n.], 2017. Citado na página 11.

GREFF, K. et al. Lstm: A search space odyssey. 2015. Citado 2 vezes nas páginas 10 e 11.

LIPTON, Z. C.; BERKOWITZ, J.; ELKAN, C. *A Critical Review of Recurrent Neural Networks for Sequence Learning*. 2015. Citado na página 8.

PRINCIPE, J. C. *Information theoretic learning : Renyi's entropy and kernel perspectives*. New York London: Springer, 2010. ISBN 978-1-4419-1569-6. Citado na página 11.

WANG, Z.; BOVIK, A. C. Mean squared error: Love it or leave it? *IEEE SIGNAL PROCESSING MAGAZINE*, Jan 2009. Citado na página 5.