USO DE COMPUTAÇÃO DE RESERVATÓRIO NA PREVISÃO DE SÉRIES TEMPORAIS DA COVID-19 NO DISTRITO FEDERAL

ISRAEL GARCIA DE OLIVEIRA

DISSERTAÇÃO DE MESTRADO EM FÍSICA TEÓRICA

DEPARTAMENTO DE PÓS- GRADUAÇÃO

INSTITUTO DE FÍSICA

UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
INSTITUTO DE FÍSICA
DEPARTAMENTO DE PÓS- GRADUAÇÃO

USING RESERVOIR COMPUTING TO PREDICT COVID-19 TIME SERIES IN DISTRITO FEDERAL

USO DE COMPUTAÇÃO DE RESERVATÓRIO NA PREVISÃO DE SÉRIES TEMPORAIS DA COVID-19 NO DISTRITO FEDERAL

ISRAEL GARCIA DE OLIVEIRA

ORIENTADOR: DR. TARCÍSIO MARCIANO DA ROCHA FILHO

DISSERTAÇÃO DE MESTRADO EM FÍSICA
TEÓRICA

PUBLICAÇÃO: PPGEA.TD-001/11

BRASÍLIA/DF: OUTUBRO - 2021

UNIVERSIDADE DE BRASÍLIA
INSTITUTO DE FÍSICA
DEPARTAMENTO DE PÓS- GRADUAÇÃO

USO DE COMPUTAÇÃO DE RESERVATÓRIO NA PREVISÃO DE SÉRIES TEMPORAIS DA COVID-19 NO DISTRITO FEDERAL

ISRAEL GARCIA DE OLIVEIRA

DISSERTAÇÃO DE MESTRADO SUBMETIDA AO DEPARTAMENTO DE PÓS- GRADUAÇÃO DA INSTITUTO DE FÍSICA DA UNIVERSIDADE DE BRASÍLIA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE.

MEMBROS DA BANCA:

Tarcísio Marciano da Rocha Filho – IF/Universidade de Brasília
Orientador

Bernardo de Assunção Mello – IF/Universidade de Brasília
Membro Interno

Marcelo Albano Moret – SENAI-CINATEC Salvador e UNEB
Membro Externo

BRASÍLIA, 25 DE OUTUBRO DE 2021.

FICHA CATALOGRÁFICA

GARCIA; ISRAEL
Uso de computação de reservatório na previsão de séries temporais da COVID-19 no Distrito Federal [Distrito Federal] 2021.
xiii, 81p., 210 x 297 mm (FIS/IF/UnB, Mestre, Física teórica, 2021).
Dissertação de mestrado – Universidade de Brasília, Instituto de Física.
Departamento de pós- graduação

1. COVID	2. Aprendizado de máquina
3. Sistemas dinâmicos	4. Computação por reservatório
I. FIS/IF/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

GARCIA, I. (2021). Uso de computação de reservatório na previsão de séries temporais da COVID-19 no Distrito Federal . Dissertação de mestrado em Física teórica, Publicação PPGEA.TD-001/11, Departamento de pós- graduação, Universidade de Brasília, Brasília, DF, 81p.

CESSÃO DE DIREITOS

AUTOR: Israel Garcia de Oliveira
TÍTULO: Uso de computação de reservatório na previsão de séries temporais da COVID-19 no Distrito Federal .
GRAU: Mestre ANO: 2021

É concedida à Universidade de Brasília permissão para reproduzir cópias desta dissertação de mestrado e para emprestar ou vender tais cópias somente para propósitos acadêmicos e científicos. O autor reserva outros direitos de publicação e nenhuma parte dessa dissertação de mestrado pode ser reproduzida sem autorização por escrito do autor.

Israel Garcia de Oliveira

Departamento de pós- graduação (FIS) - IF
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

Acredito que a última grande invenção humana seja a inteligência artificial.

ACKNOWLEDGMENTS

Primeiramente agradeço aos meus pais, Rozanilde Garcia Prazeres e Gerson Camilo de Oliveira, por me proporcionarem uma vida estável e cheia de felicidades. Tenho certeza que sem eles a minha jornada de me tornar cientista seria impossível. Agradeço também ao meu Orientador, Tarcísio Marciano da Rocha Filho, que sempre me ajudou com muita paciência em momentos em que eu me sentia perdido. Por último, agradeço à CAPES pela bolsa de mestrado.

RESUMO

Título: Uso de computação de reservatório na previsão de séries temporais da COVID-19 no Distrito Federal
Autor: Israel Garcia de Oliveira
Orientador: Dr. Tarcísio Marciano da Rocha Filho
Programa de Pós-Graduação em Física teórica
Brasília, 25 de outubro de 2021

A epidemia do coronavírus, causada pelo vírus SARS-COV-2, teve seu início em dezembro de 2019 na china e rapidamente se espalhou para vários países ao redor do mundo, sendo então declarada uma pandemia pela Organização Mundia da Saúde (OMS) em questão de dois meses. Prever tendências de uma doença contagiosa em uma população é um assunto de grande relevância para a sociedade, pois causa impacto positivo nas tomadas de decisão por parte dos agente competentes. Diferentes metodologias existem, como por exemplo o modelo SIR ou o modelo *time-since-infection*. Uma alternativa que vem cada vez mais tendo atenção de pesquisadores são os métodos baseados em aprendizado de máquina. Uma possibilidade é alimentar o algoritmo de aprendizado de máquina com a série temporal (de casos, por exemplo) e obter estimativas da evolução futura da série temporal por uma janela de tempo. Neste trabalho usamos redes neurais baseadas em computação de reservatório para prever a evolução do coronavírus no Distrito Federal. Obtemos boas previsões para o número de reprodução instantâneo R_t em até cinco dias no futuro e para as médias móveis de casos e mortes em até dez dias no futuro com um erro relativo médio de 10%.

Palavras-chave: COVID, Aprendizado de máquina, Sistemas dinâmicos, Computação por reservatório.

ABSTRACT

Title: Using reservoir computing to predict COVID-19 time series in Distrito Federal
Author: Israel Garcia de Oliveira
Supervisor: Dr. Tarcísio Marciano da Rocha Filho
Graduate Program in Theoretical Physics
Brasília, October 25th, 2021

The curent COVID-19 pandemic, caused by the SARS-CoV-2 virus, begun in December 2019 in China, and rapidly spread to all countries in the World, being declared pandemic by the World Health Organization (WHO) in a matter of two months. Predicting trends of an infectious disease is a topic of high social relevance, being able to impact on better decision making. Different theoretical approaches exist, such as the SIR and time-since-infection models. An alternative gradually drawing more attention from researchers is machine-learning based approaches. One possibility is to fed an algorithm with the previous time series of daily cases and/or deaths, and obtain estimates for the future evolution fow a given time window. In the present work we use the reservoir computing approach to predict the evolution of coronavirus in the Federal Disctrict in Brazil. We obtained good quality predictions for instantaneous reproduction number R_t up to five days in the future, and the moving average of cases and deaths up to ten days ahead with a maximum relative error of 10%.

Keywords: COVID, Machine Learning, Dynamical Systems, Reservoir Computing.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1 MOTIVAÇÃO E OBJETIVOS

Doenças contagiosas sempre foi um assunto de grande importância para a sociedade, visto que o modo como lidamos com isso afeta diretamente a qualidade de vida das pessoas.

1.1 MOTIVAÇÃO

Em 2020 a humanidade presenciou o início de um dos maiores e mais importantes eventos da era moderna. A propagação de um vírus a nível global muda para sempre a forma como vivemos e nos relacionamos. No final de dezembro de 2019, o vírus SARS-CoV-2, conhecido também como coronavírus, foi identificado. O vírus teve seu epicentro na China, mais precisamente na província de Hubei [10]. O vírus rapidamente se espalhou para outros países, fazendo com que no dia 30 de janeiro de 2020 a organização mundial da saúde declarasse emergência a nível global. Pesquisadores do mundo todo começaram então o combate à doença, estudando-a de várias formas. Como consequência positiva, uma enorme quantidade de trabalhos acadêmicos são publicados afim de aumentar o conhecimento sobre o novo vírus. Atkeson em [1] discute sobre os impactos econômicos que o coronavírus poderá causar e estuda vários possíveis cenários para a propagação da doença nos Estados Unidos com projeções de até dezoito meses, admitindo vários valores diferentes para o número de reprodução efetivo no modelo teórico SIR (sigla que representa suscetíveis-infectados-removidos, os removidos sendo recuperados ou falecidos) [3], que será abordado com mais calma posteriormente neste trabalho. Seus resultados nas figuras 1.1 e 1.2 mostram e evolução do número total de casos e de infectados para vários valores de R_t, respectivamente. R_t é chamado de número de reprodução efetivo e varia no tempo, mas nesse caso R_t é mantido constante e igual a R_0, o número de reprodução básico, medido logo no início da epidemia. Atkeson mostra que apenas para valores de R_t próximos de 1.6 a epidemia pode ser controlada teoricamente. Dharmaratne et al. em [11] usam também o modelo teórico SIR para estimar o número de reprodução básico (R_0) do coronavírus na população do Sri Lanka. Eles estimam valores de R_0 dentro do intervalo $[0.93, 1.23]$ indicando que, no momento em que foi feito o estudo, a quantidade de casos diários do coronavírus estava crescendo no Sri Lanka. Em [12], Hsiang et al. estudam os impactos das políticas anti contágio em larga escala em vários países. Chegando em um pensamento parecido ao de Atkeson em [1], eles concluem que as políticas de isolamento, como fechar escolas, estabelecimentos comerciais e também a prática do *home office* são medidas valiosas contra a proliferação da doença mas que podem trazer sérios impactos sociais e econômicos para a vida das pessoas. Aloísio et al. em [2] estimam o número de reprodução efetivo em todos os estados do Brasil em janeiro de 2021. Usando o modelo de Kermack McKendrick, conhecido também como modelo *time-*

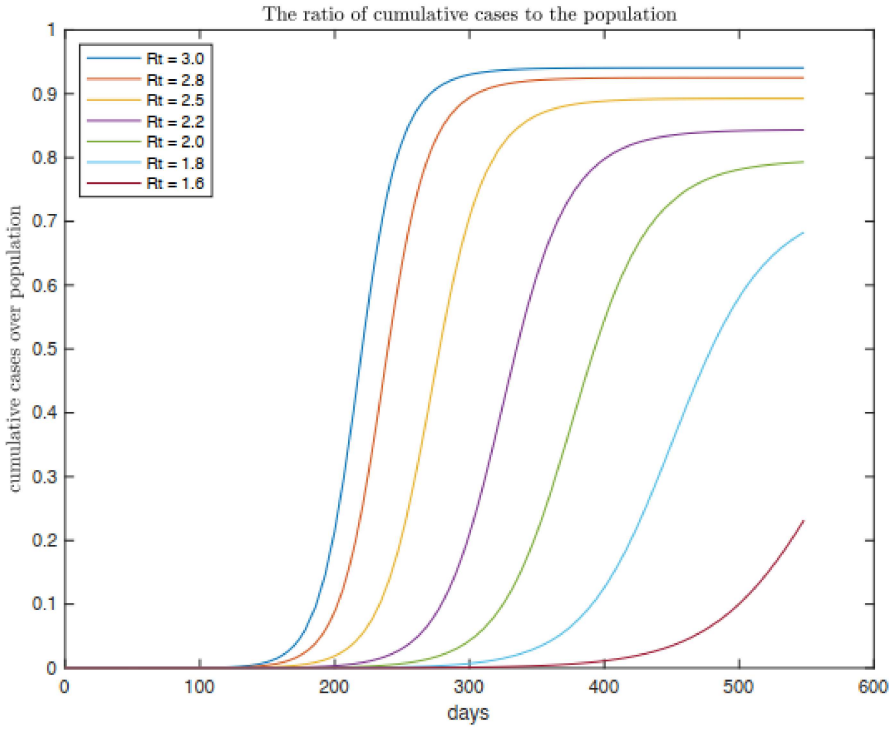

Figura 1.1 – Simulação da fração de casos cumulativos nos Estados unidos durante dezoito meses para vários valores do número de reprodução efetivo R_t. Neste caso $R_t = R_0 = cte$. Figura retirada de [1].

since-infection [13], os autores chegam nos resultados que podem ser conferidos na figura 1.3, onde vemos que em janeiro de 2021 a epidemia estava crescendo em todos os estados do Brasil. Tatiana et al. em [14] usam também o modelo SIR para estimar o número de reprodução efetivo R_t usando dados de vários países ao redor do mundo. Para isso os autores usam os dados sobre a quantidade de novos casos, recuperados e falecidos diários com o objetivo de estimar uma condição inicial para o sistema dinâmico, para poder integrá-lo e assim estimar o valor de R_t. Vale lembrar que em situações reais o valor de Rt varia com o tempo devido a diversos fatores como decisões de quarentena, uso de máscara e também a própria dinâmica de interação das pessoas de uma certa região. De fato, uma boa estimativa desse parâmetro em tempo real é muito importante e bastante desafiador.

Todos os trabalhos citados até agora têm bastante coisa em comum quanto aos seus objetivos. Todos eles procuram estimar ou estudar alguma quantidade dinâmica sobre a propagação do coronavírus, como a quantidade total de casos ou o valor do número de reprodução efetivo em um certo dia. A razão disso é que essas quantidade são de grande importância para tomadas de decisão por parte dos agentes públicos, pois elas carregam bastante informação sobre o andamento da epidemia e são simples de entender. Outra coisa que os trabalhos citados até agora têm em comum é que eles usam de forma direta ou indireta algum modelo teórico para poder tentar explicar o comportamento do coronavírus em uma população. Apesar de servirem muito bem para seus propósitos, esses modelos teóricos são bastante li-

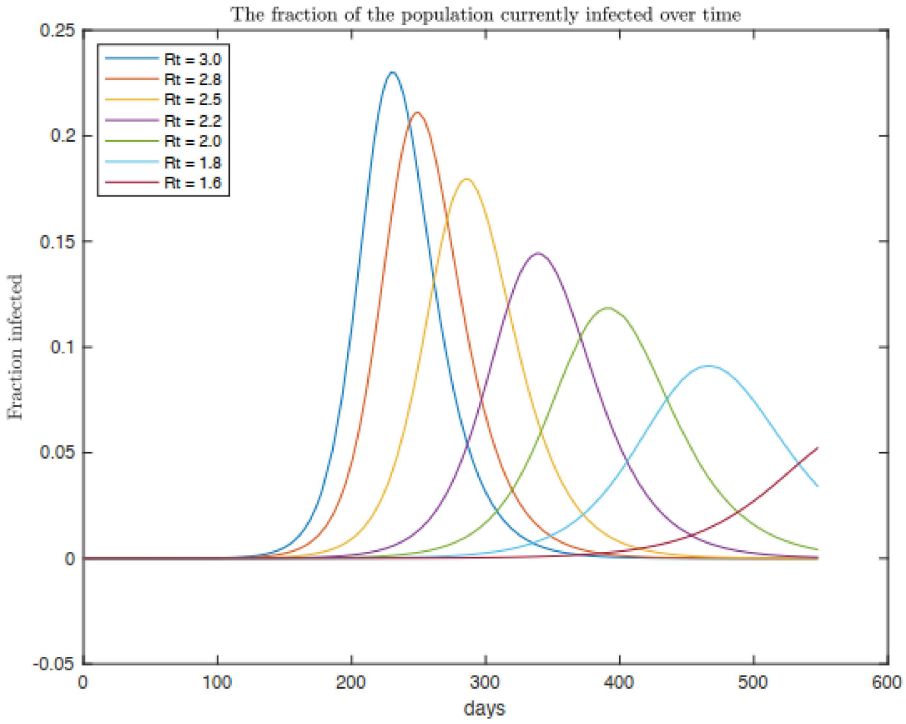

Figura 1.2 – Simulação da fração de infectados nos Estados unidos durante dezoito meses para vários valores do número de reprodução efetivo R_t. Neste caso $R_t = R_0 = cte$. Figura retirada de [1].

mitados quando consideramos uma dinâmica real de toda uma população, levando em conta suas interações e também eventuais medidas anti contágio para limitar de alguma forma essas interações. Para acompanhar tais mudanças, seriam necessárias alterações nos modelos que os deixariam mais complicados a ponto de serem muito difíceis de utilizar e os tornariam pouco práticos.

Uma abordagem que recentemente vem ganhando espaço entre os pesquisadores é o uso de redes neurais artificiais (RNA) no estudo do comportamento do coronavírus. Uma rede neural artificial é composta por neurônios computacionais que por sua vez são formados por pesos e uma função de ativação. A unidade mais básica de uma rede neural é o perceptron. Observando a figura 1.4, o perceptron é composto por duas camadas. A primeira é a camada de entrada de dados (*input layer*), onde o neurônio recebe os dados. A segunda camada é a saída de dados (*output layer*), onde o neurônio dá como resposta algum valor. As arestas que ligam os nós da primeira camada para a segunda é a característica que revolucionou a forma como lidamos com vários problemas teóricos e práticos. Essas arestas representam os pesos (*weights*) do perceptron e eles são treináveis, no sentido que (a princípio) podem se ajustar de acordo com a necessidade do problema. Essa característica de poder ser treinado de acordo com a necessidade do problema é uma propriedade atraente que as redes neurais possuem, sendo esse um ponto forte quando tratamos de previsões em dados do coronavírus.

Como exemplo de aplicação, ArunKumar et al. em [15] utilizam redes neurais recorren-

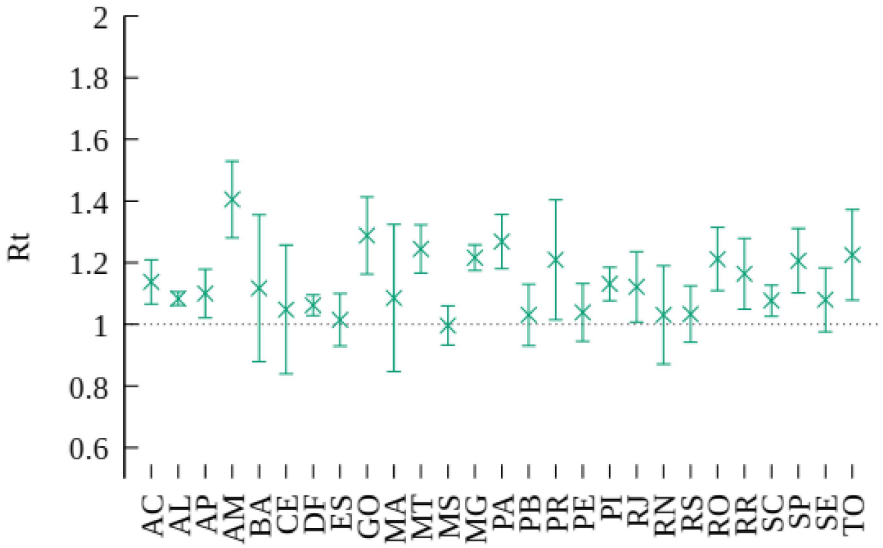

Figura 1.3 – Número de reprodução efetivo em cada estado brasileiro em janeiro de 2021. Figura retirada de [2].

tes para fazer previsões nas séries temporais de casos, recuperações e mortes acumulados. Para isso, os autores empregam dois tipos de redes neurais bastante famosos: GRU (*gated recurrent unit*) e LSTM (*long short-term memory*). Essas redes neurais são chamadas de recorrentes porque elas têm a capacidade de "lembrar" padrões de dados passados. Matematicamente, o estado atual de uma rede neural recorrente é função da entrada de dados atual e do estado anterior (referente à entrada de dados anterior) e pode ser representado pela seguinte relação

$$h_t = f(W\,[h_{t-1}, x_t] + b), \tag{1.1}$$

onde h_t é o estado da rede neural no tempo t, x_t são os dados de entrada no tempo t e b é chamado de viés (*bias*). f é chamada de função de ativação e geralmente tem como imagem o intervalo $[-1, 1]$. Shastri et al. em [16] Usam variantes da LSTM para fazer previsões na epidemia do coronavírus nos Estados Unidos e Índia. Nesse trabalho, os autores fazem previsões na série temporal de casos cumulativos dos dois países usando três versões modificadas da rede neural: LSTM empilhada, LSTM bidirecional e LSTM convolucional. As redes neurais recorrentes são bastante eficazes na previsão de dados sequenciais, que na maioria das vezes se apresentam em forma de séries temporais. Exemplos de séries temporais são a sequência de casos diários do coronavírus, o preço de uma ação ou qualquer evolução no tempo de um sistema dinâmico.

Observando o panorama atual e encarando o problema do coronavírus como motivação, o objetivo deste trabalho é contribuir na área propondo novas técnicas de aprendizado de máquina que, no nosso conhecimento atual, ainda não foram aplicadas. As redes neurais

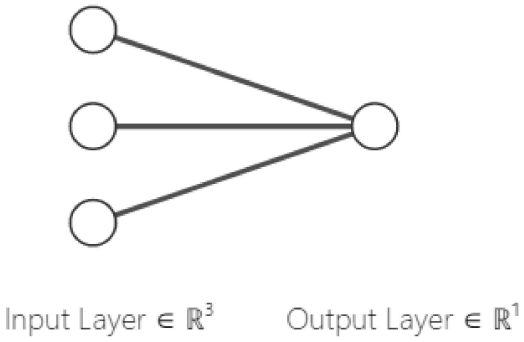

Figura 1.4 – Figura esquemática do perceptron. A primeira camada recebe os dados e a segunda o processa e dá como resposta outro dado. Os pesos (arestas ligando os nós) são treináveis com algoritmos específicos.

que vamos utilizar são também redes neurais recorrentes mas com uma diferença fundamental. Chamada de *Echo State Network*, ou simplesmente ESN, essa rede neural possui uma camada com neurônios não treináveis, chamada de reservatório de neurônios (ou simplesmente reservatório). Com essa característica é possível efetuar treinamentos muito mais rápidos e eficazes em comparação com as redes tradicionais [17].

1.2 OBJETIVOS

1.2.1 Objetivos gerais

Este trabalho se dedica a usar variações da ESN para fazer previsões na evolução do coronavírus no Distrito Federal. Mais precisamente vamos fazer algumas modificações na ESN e também usar a *deep* ESN, que é a echo state com camadas internas.

1.2.2 Objetivos específicos

- Utilizar duas variações da ESN, com duas topologias de reservatório diferentes.
- Utilizar duas variações da *deep* ESN, com duas topologias de reservatório diferentes.
- Fazer previsões em casos acumulados, mortes acumuladas, média móvel de casos e

média móvel no Distrito Federal.

- Utilizar a ESN em conjunto com o modelo *time-since-infection* para fazer previsões no número de reprodução efetivo no distrito federal.

1.3 ORGANIZAÇÃO DO TRABALHO

O trabalho se organiza da seguinte maneira: o capítulo 2 apresenta uma teoria sobre doenças contagiosas focando no ponto de vista matemático, mostrando alguns dos modelos teóricos mais utilizados e também uma explicação do número de reprodução efetivo. O capítulo 3 se trata da teoria por trás das redes neurais. Aqui vamos aprofundar sobre o funcionamento dessa ferramenta que vem revolucionando todos os campos da ciência, apresentando os fundamentos e depois dando foco em computação de reservatório, uma subárea dentro de redes neurais recorrentes. O capítulo 4 mostra como foi feita a pesquisa de forma detalhada, explicando desde a coleta de dados até a obtenção de resultados. O capítulo 5 mostra os resultados obtidos. Finalmente o capítulo 6 conclui o trabalho.

2 DOENÇAS CONTAGIOSAS

A dinâmica de propagação de uma doença contagiosa é há muito tempo uma área do conhecimento de grande interesse por parte dos matemáticos e físicos pela sua grande importância para a sociedade e os desafios que pode apresentar.

2.1 INTRODUÇÃO

A propagação de doenças contagiosas está entre as maiores ameaças que países ou até mesmo o próprio mundo podem enfrentar. Esse tipo de doença é bastante perigoso porque fere gravemente a saúde pública e a economia da região afetada. Seu crescimento dentro de uma população logo no início se dá de forma exponencial [3], o que leva a uma grande quantidade de novos pacientes para os hospitais. Como consequência, pode acontecer a superlotação de hospitais e assim a falta de leitos, o que afeta o tratamento de outras doenças já existentes. Dependendo da natureza da doença, como ela se propaga e o prognóstico das pessoas infectadas, medidas anti contágio devem ser tomadas na tentativa de mitigar o número de infectados. Exemplos dessas medidas são o distanciamento entre pessoas, uso de proteção (dependendo de como a doença é transmitida), políticas educativas para informar a população sobre a doença e, em casos mais sérios, o isolamento parcial ou total das pessoas em suas casas (chamado de *lockdown*).

2.2 MODELOS EPIDEMIOLÓGICOS

Uma das formas de modelar uma epidemia é utilizando modelos epidemiológicos. Geralmente esses modelos tratam a população dividida em compartimentos de pessoas, onde cada compartimento possui uma característica, podendo ela ser de pessoas infectadas ou pessoas recuperadas, por exemplo. Toda a dinâmica desses modelos vem da interação desses compartimentos, visto que uma pessoa que é infectada entra no compartimento de infecciosos e uma pessoa que é recuperada (ou falece) sai do compartimento de infecciosos e entra no compartimento de recuperados. Por causa disso esses modelos teóricos são representados por equações diferenciais ordinárias. Neste trabalho introduziremos apenas os modelos teóricos mais conhecidos e então focaremos no modelo *time-since-infection* que é utilizado em conjunto com a ESN para a previsão do número de reprodução.

2.2.1 Modelo SIR

Talvez o modelo mais famoso na literatura seja o SIR [3], que significa suscetível-infeccioso-recuperado (*susceptible-infective-recovered*). Esse modelo divide a população em três compartimentos, sendo eles:

- suscetíveis: pessoas que podem ser infectadas, ou seja, não estão imunes;
- infecciosos: pessoas que estão infectadas e podem infectar outras pessoas;
- recuperados: pessoas que foram infectadas mas já estão recuperadas.

De acordo com a figura 2.1, vemos que uma pessoa que é suscetível deve esperar passar por três estágios. Matematicamente o modelo SIR é representado pelo seguinte sistema de

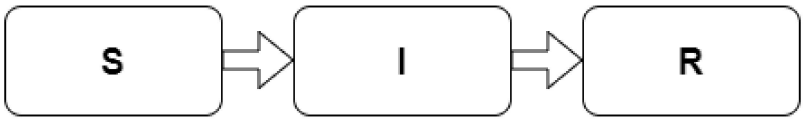

Figura 2.1 – Diagrama de interação entre compartimentos para o modelo SIR.

equações:

$$\begin{aligned} \frac{dS}{dt} &= -\frac{\beta SI}{N}, \\ \frac{dI}{dt} &= \frac{\beta SI}{N} - \gamma I, \\ \frac{dR}{dt} &= \gamma I, \end{aligned} \tag{2.1}$$

onde S, I e R são os compartimentos, N é a população total da região considerada, β é a razão de contato efetivo (pode ser considerada a probabilidade de infecção entre dois indivíduos) e γ é a taxa de remoção. A quantidade $1/\gamma$ define o número de dias que uma pessoa permanece infectada. Olhando com mais cuidado o sistema de equações 2.1, vemos que o compartimento de suscetíveis diminui a uma taxa dada pela quantidade

$$-\frac{\beta SI}{N}. \tag{2.2}$$

A taxa de variação na quantidade de infecciosos é dada pela entrada de suscetíveis (o termo $\beta SI/N$) e saída de recuperados (o termo $-\gamma I$). O último compartimento, dos recuperados, cresce a uma taxa dada por γI. Como a população aqui é considerada constante, temos:

$$N = S + I + R. \tag{2.3}$$

Como consequência:

$$\frac{dS}{dt} + \frac{dI}{dt} + \frac{dR}{dt} = 0. \tag{2.4}$$

Uma quantidade muito importante que podemos extrair do modelo SIR é o número de reprodução básico R_0. R_0 é definido como o número médio de pessoas que um infeccioso pode infectar se inserido em uma população totalmente suscetível ($S \approx N$) [3]. Essa quantidade é fundamental para determinar a situação de uma epidemia ou surto logo no seu início, pois com ela temos uma boa ideia da força que a doença tem na população. No caso do modelo SIR, R_0 é dado por:

$$R_0 = \frac{\beta}{\gamma}. \tag{2.5}$$

A equação 2.5 mostra que R_0 é constante, mas sabemos que em uma situação real, a quantidade de infecções que um infeccioso pode causar pode variar no tempo, ou seja,

$$R_t = f(t), \tag{2.6}$$

onde chamamos R_t de número de reprodução efetivo. Como devemos esperar, o cálculo de R_t é bastante complicado, pois depende da situação atual da epidemia ou surto, que por sua parte depende de vários outros fatores, como a dinâmica de interação das pessoas, as formas como a doença pode se propagar e também decisões por parte dos agentes competentes. Uma forma de obter R_t é através do relatório de casos diários. Petrova et al. em [14] utiliza o modelo SIR para estimar o R_t do coronavírus em vários países. Para isso os autores estimam em tempo real o valor de β, chamado de β^* e o usa para calcular R_t da seguinte maneira:

$$R_t = \frac{\beta^*}{\gamma}, \tag{2.7}$$

onde γ é mantido fixo durante todo o processo. Para estimar β^*, os autores utilizam o seguinte método de minimização:

$$\beta^* = \arg\min_{\beta} \sum_{t=t_0}^{t_0+n} (V'(t) - I'_\beta(t))^2, \tag{2.8}$$

onde t_0 é o primeiro dia da epidemia, $V'(t)$ é o número de casos no dia t e $I'_\beta(t)$ é o número de casos no dia t estimado pelo modelo.

Uma aplicação muito interessante de uma adaptação do modelo SIR e de uma rede neural do tipo *feedforward* é encontrada em [18]. Neste trabalho os autores usam uma espécie de modelo SIR mais robusto junto com uma *deep fedforward* para fazer vários tipos de previsões na propagação do coronavírus na Coreia do Sul. Eles consideram vários cenários, como por exemplo situações de isolamento, com e sem vacina, entre outros. Um ponto interessante do trabalho é que os autores modificam a função custo no algoritmo de treinamento

da rede neural de forma que os erros da estimativa do modelo SIR adaptado são levados em consideração.

2.2.2 Modelo MSEIR

O modelo MSEIR merece atenção por ser bastante conhecido e também por ser uma evolução natural do modelo SIR. Este modelo resolve três limitações do anterior: agora a população varia, temos o compartimento de pessoas imunes e o compartimento de pessoas expostas. Como a população agora pode variar, os compartimentos de imunes e suscetíveis vão depender também de mortes e nascimentos de pessoas. Observando a figura 2.2, ve-

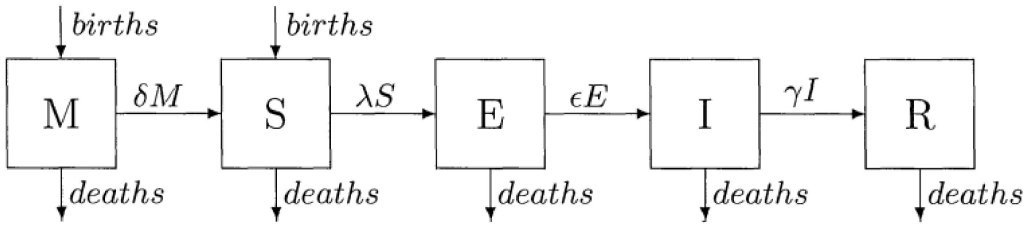

Figura 2.2 – Diagrama de interação entre compartimentos para o modelo MSEIR. Figura retirada de [3].

mos que o compartimento de imunes recebe pessoas por nascimento mas perde pessoas por falecimento e transferência para o compartimento de suscetíveis, que por suas vez também recebe pessoas por nascimento e perde pessoas por falecimento. Este compartimento perde pessoas também por transferência para o compartimento de expostos. Os expostos perdem pessoas por morte e por transferência para o compartimento de infecciosos. Este compartimento, por sua vez, perde pessoas por falecimento e transferência para o compartimento de recuperados. Finalmente, este compartimento perde pessoas por falecimento. Todo esse processo pode ser representado pelo seguinte sistema de equações diferenciais:

$$\begin{aligned}
\frac{dM}{dt} &= b(N-S) - (\delta + d)M, \\
\frac{dS}{dt} &= bS + \delta M - \frac{\beta SI}{N} - dS, \\
\frac{dE}{dt} &= \frac{\beta SI}{N} - (\epsilon + d)E, \\
\frac{dI}{dt} &= \epsilon E - (\gamma + d)I, \\
\frac{dR}{dt} &= \gamma I - dR, \\
\frac{dN}{dt} &= (b-d)N,
\end{aligned} \tag{2.9}$$

onde b é a taxa de nascimento, d é a taxa de morte, δ é a probabilidade de transferência dos imunes para os suscetíveis, λ é a probabilidade de transferência dos suscetíveis para os expostos, ϵ é a probabilidade de transferência dos expostos para os infecciosos e γ é a probabilidade de transferência dos infecciosos para os recuperados. Todos os compartimentos podem perder pessoas por falecimento. De acordo com o modelo temos:

$$M + S + E + I + R = N, \tag{2.10}$$

onde N é a população total. Como consequência:

$$\frac{dM}{dt} + \frac{dS}{dt} + \frac{dE}{dt} + \frac{dI}{dt} + \frac{dR}{dt} = (b - d)N. \tag{2.11}$$

Novamente podemos extrair do sistema de equações 2.9 o número básico de reprodução R_0. De acordo com [3] R_0 é dado por [13]:

$$R_0 = \frac{\beta\epsilon}{(\gamma + d + q)(\epsilon + d + q)}. \tag{2.12}$$

Novamente, para se obter o número de reprodução efetivo podemos a adaptar a técnica apresentada por [14].

2.2.3 Modelo *time-since-infection*

Apesar dos modelos SIR e MSEIR serem de fácil interpretação (de modo geral modelos de compartimento são fáceis de interpretar), a tarefa de obter o número de reprodução pode ser desafiadora. Para ser mais preciso, usar esse tipo de modelo em conjunto com uma rede neural pode ser desafiador por alguns fatores. O primeiro fator é o simples acompanhamento do número de reprodução que a todo instante depende da minimização de uma função custo. Já do ponto de vista de aprendizado de máquina, precisamos de duas séries temporais, uma de casos diários e outra de recuperados diários. E isso é um problema pois nem todo banco de dados possui o acompanhamento de recuperados diários. Para contornar este problema introduzimos o modelo *time-since-infection*.

A primeira vantagem do modelo *time-since-infection* é na dedução do número de reprodução. A equação para R_t que obtemos precisa apenas da série temporal de casos diários e de uma distribuição de probabilidade para fazer o acompanhamento diário da quantidade. Como mostrado em [13] essa distribuição deve ser de Poisson. A teoria aqui mostrada pode não ser tão intuitiva como os modelos já apresentados, mas torna muito mais fácil o acompanhamento e previsão do R_t em cooperação com uma rede neural.

O modelo tratado aqui tem a função de prever a incidência de uma doença, ou seja, quantas pessoas na média poderão ser infectadas em um período fixo de tempo. Matemati-

camente, a incidência pode ser representada pela seguinte equação recursiva:

$$I(t) = \int_0^\infty \beta(t,\tau) I(t-\tau) d\tau, \tag{2.13}$$

onde $I(t)$ é a incidência média no tempo t e $\beta(t,\tau)$ é uma função arbitrária que depende da doença e da dinâmica de interação das pessoas. Essa função pode ser considerada como um análogo ao parâmetro β dos modelos SIR e MSEIR. De acordo com [13], $\beta(t,\tau)$ pode ter o formato de uma distribuição de Poisson. Dependendo das circunstâncias é conveniente truncar a função β para valores de τ maiores que uma constante α tal que $\alpha > 0$. O número de reprodução R_t é o número médio de pessoas que um infeccioso pode infectar no instante t. Aqui ele é dado por:

$$R_t = R(t) = \int_0^\infty \beta(t,\tau) d\tau. \tag{2.14}$$

Novamente, R_t é uma propriedade de grande importância e bastante intuitiva. Caso $R_t < 1$, a epidemia está diminuindo. No caso onde $R_t > 1$, a epidemia está crescendo. Ainda de acordo com [13], podemos admitir que $\beta(t,\tau)$ é um produto de duas funções, ou seja:

$$\beta(t,\tau) = R(t) w(\tau), \tag{2.15}$$

onde $w(\tau)$ é uma distribuição tal que, se discretizada no tempo, admite a seguinte relação:

$$\sum_{i=0}^{n} w_i = 1. \tag{2.16}$$

Usando a equação 2.15 a fórmula recursiva para a incidência de infectados, temos:

$$I(t) = \int_0^\infty R(t) w(\tau) I(t-\tau) d\tau. \tag{2.17}$$

Resolvendo para $R(t)$ chegamos em:

$$R(t) = \frac{I(t)}{\int_0^\infty I(t-\tau) w(\tau) d\tau}. \tag{2.18}$$

Finalmente, na forma discreta temos:

$$R(t_i) = \frac{I_i}{\sum_{j=0}^{n} w_j I_{i-j}}, \tag{2.19}$$

onde t_i é o dia i, I_i é o número de infectados no dia i e w_j representa a probabilidade de aparição de sintomas no dia j.

3 REDES NEURAIS

O uso de aprendizado de máquina na ciência e na tecnologia revolucionou a forma como lidamos com problemas das mais variadas formas. Esse novo modo de abordagem que as redes neurais proporcionam permite soluções simples para desafios complexos.

3.1 INTRODUÇÃO

O cérebro é uma máquina extremamente poderosa, complexa e não linear capaz de executar tarefas que os melhores computadores atuais sequer sonham em tentar. Tarefas simples como reconhecer uma pessoa em uma multidão, reconhecer um objeto pela sua foto, assimilar uma situação e agir de acordo, classificar objetos ou qualquer outra coisa concreta ou abstrata são muitas vezes bastante complexas para um computador e requerem tempo e algoritmos complexos. Em termos práticos, considerando o cérebro como uma caixa-preta (*black box*), ele recebe dados em forma de estímulos e retorna uma resposta em forma de estímulo ou ação. Por exemplo, em uma conversa entre duas pessoas, onde as duas falam o mesmo idioma, quando uma pessoa pergunta para a outra "que horas vamos jantar?", a segunda pessoa recebe essa mensagem via audição, o cérebro processa e analisa as possibilidades, como o horário do jantar, se vai chover ou não e muitos outros fatores, e assim o cérebro retorna uma resposta, com a pessoa dizendo se vai querer jantar ou não (claro que essa situação pode ser muito mais complicada do que isso) com o horário do jantar no caso positivo. Outro exemplo seria o processo de aprender um novo idioma. Um dos métodos mais empregados em aprendizagem de línguas estrangeiras é o da repetição, onde o aluno fala várias vezes a mesma frase no idioma que está aprendendo até que tenha um bom entendimento do que está sendo falado. Outra forma de aprendizado também é assistir muitas vezes um mesmo episódio de uma série ou um mesmo filme, repetindo as falas dos personagens. Esses métodos de aprendizagem exploram a neuroplasticidade do cérebro, que é a habilidade em mudar sua estrutura neuronial de acordo com novas experiências [19]. Essa característica não é exclusiva dos seres humanos. O sonar do morcego por exemplo (figura 3.1), é um sistema ativo de ecolocalização que tem a capacidade de estimar a distância de objetos e outros animais. Para fazer isso, esse sistema precisa de vários dados sobre o alvo, como seu tamanho, velocidade relativa, forma geométrica, entre outros [20]. E esse tipo de computação complexa acontece em um cérebro minúsculo.

Por trás de toda a complexidade do cérebro, existem estruturas minúsculas que são as grandes responsáveis pela capacidade de resolução de tarefas. Os neurônios são os grandes protagonistas na formação do cérebro. O cérebro humano contém aproximadamente cem

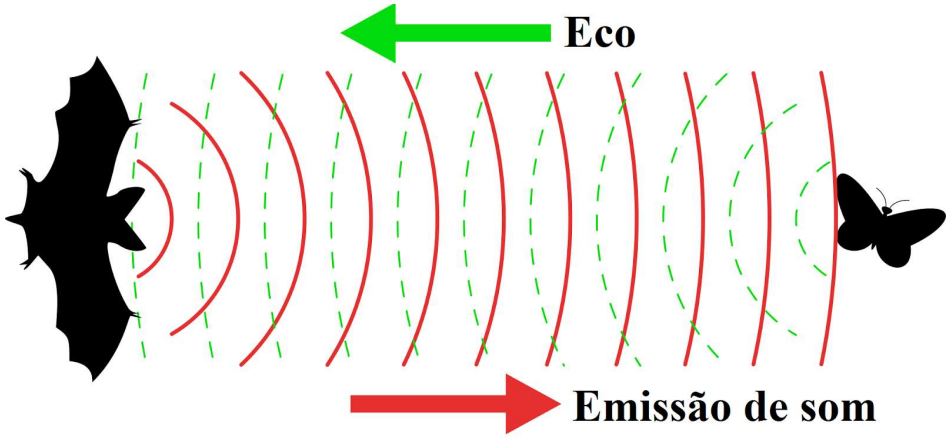

Figura 3.1 – Representação do sistema de ecolocalização do morcego. Figura retirada de [4].

bilhões dessas pequenas estruturas. O neurônio é formado pro três subestruturas: o corpo, os dendritos e o axônio. Observando a figura 3.2, os dendritos são responsáveis por receber estímulos externos em forma de corrente elétrica, o axônio é responsável por enviar estímulos para o exterior em forma de corrente elétrica e o corpo forma o restante da célula. Como os dendritos recebem estímulos, eles podem ser chamados de *input-gate* (porta de entrada). De forma análoga, o axônio pode ser chamado de *output-gate* (porta de saída). Como os neurônios estão contidos em água com íons de sódio (Na^+), potássio (K^+), cloreto (Cl^-) e cálcio (Ca^{2+}) dissolvidos, os neurônios vão receber e enviar estímulos baseados nesses íons [6]. Os neurônios são excitáveis, ou seja, eles são capazes de gerar breves picos de voltagem. Tal fenômeno é chamado de potencial de ação [21]. A figura 3.3 mostra uma simulação computacional do modelo de Hodgkin-Huxley com $I = 7\mu A/cm^2$ [6]. Observando a figura vemos que no momento que o neurônio atinge um potencial mínimo, seu potencial começa a crescer de forma cada vez mais acentuada até chegar o momento do potencial de ação. Logo em seguida o potencial volta para o seu valor mínimo e assim começa novamente seu ciclo.

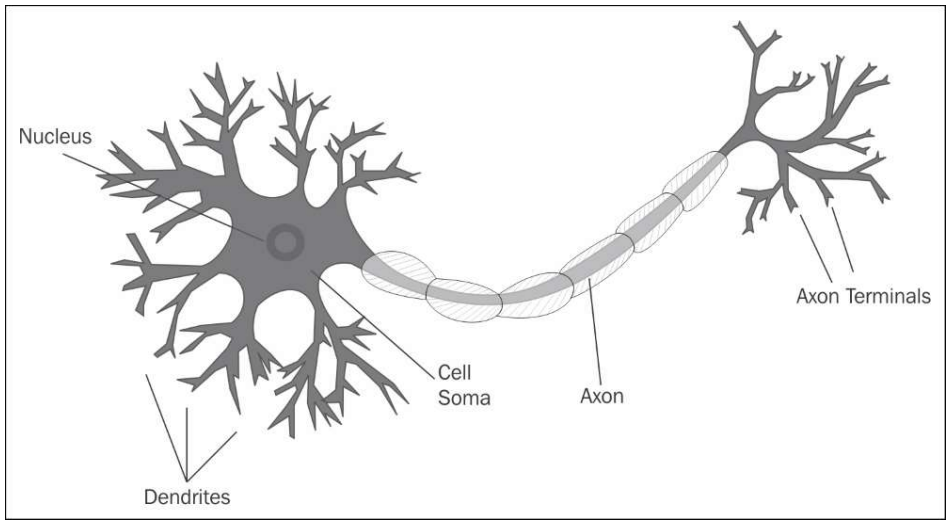

Figura 3.2 – Representação esquemática do neurônio biológico. Figura retirada de [5].

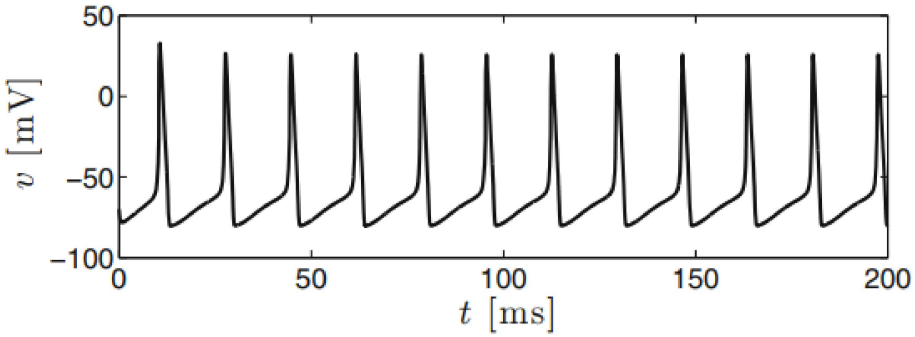

Figura 3.3 – Simulação computacional do modelo de Hodgkin Huxley com $I = 7\mu A/cm^2$. Figura retirada de [6].

3.2 MODELAGEM MATEMÁTICA DE UM NEURÔNIO

Alan Hodgkin e Andrew Huxley foram os primeiros a explicar com sucesso como as células nervosas geram impulsos elétricos. Em 1952 eles publicaram uma série de artigos sobre o fenômeno. O último dos artigos é talvez o mais importante artigo já escrito em neurociência, pois serviu como fundamento para as áreas de neurociência matemática e neurociência computacional [6] [22]. Esta seção é dedicada a introduzir o modelo de Hodgkin-Huxley, que é um sistema de equações diferenciais ordinárias que explica como acontecem os potenciais de ação, na tentativa de servir como motivação para a construção da teoria de redes neurais artificiais [22].

Para descrever como o potencial de uma célula nervosa evolui no tempo, primeiro devemos assumir que tal célula se comporta como um capacitor de duas camadas que possuem sinais de carga diferentes. Encarando esse sistema como um dipolo elétrico, ele pode gerar saltos no potencial elétrico da célula (potencial de ação). Esse capacitor tem a sua capacitância dada pela seguinte relação:

$$C = \frac{Q}{v}, \tag{3.1}$$

onde C é a capacitância, Q é a carga em cada placa (uma placa com carga $+Q$ e a outra com carga $-Q$) e v é o potencial da membrana celular. Nessa situação, a carga e o potencial dependem do tempo. Resolvendo para a carga e derivando dos dois lados em relação ao tempo, obtemos:

$$Q = Cv, \tag{3.2}$$

$$\frac{dQ}{dt} = \frac{d}{dt}(Cv). \tag{3.3}$$

Considerando a capacitância constante, obtemos

$$I = C\frac{dv}{dt}, \tag{3.4}$$

onde I é a corrente total e dv/dt é a variação do potencial com o tempo. Em seus experimentos, Hodgkin e Huxley admitiram que a corrente total seria composta pela soma das correntes de cada íon e a corrente criada por eles. Mais precisamente, a corrente total foi dividida em quatro componentes: íons de sódio, íons de potássio, uma corrente formada pelo restante dos íons e a corrente criada em laboratório [6]. Considerando essa divisão chegamos na seguinte relação:

$$C\frac{dv}{dt} = I_{Na} + I_K + I_L + I, \tag{3.5}$$

onde I_L representa a corrente envolvendo o restante dos íons e I é a corrente produzida em laboratório. As correntes I_{Na}, I_K e I_L podem ser escritas em termos de seus respectivos potenciais de Nernst da seguinte forma [23] [6]:

$$I_{Na} = g_{Na}(v_{Na} - v), \tag{3.6}$$

$$I_K = g_K(v_K - v), \tag{3.7}$$

$$I_L = g_L(v_L - v). \tag{3.8}$$

Nas equações 3.6, 3.7 e 3.8, v_{Na}, v_k e v_L são os potenciais de Nernst de sódio, potássio e o restante dos íons, respectivamente. Mais precisamente, como I_L é a soma das correntes de vários tipos de íons com pequenas contribuições para a corrente total, v_L é melhor interpretado como sendo a média dos potenciais de Nernst de cada íon que contribui para a corrente I_L. Para a dedução dos potenciais de Nernst o leitor pode consultar o capítulo 2 de [6]. As constantes g_{Na} e g_K foram também deduzidas experimentalmente e assumem a seguinte forma:

$$g_{Na} = \overline{g}_{Na} m^3 h, \tag{3.9}$$

$$g_K = \overline{g}_K n^4, \tag{3.10}$$

onde $\overline{g}_{Na}$ e $\overline{g}_K$ são condutâncias e m, h e n variáveis dependentes do tempo que admitem valores no intervalo $[0, 1]$. A explicação dada por Hodgkin e Huxley sobre as variáveis m, h e n é que elas são portas para a passagem de íons. Por exemplo, o canal de sódio será ativado apenas se as três portas tipo m e a porta tipo h forem ativadas (valores próximos de um) [22]. Para obter o valor dessas portas no tempo, devemos resolver as seguinte equações diferenciais:

$$\frac{dm}{dt} = \frac{m_\infty(v) - m}{\tau_m(v)}, \tag{3.11}$$

$$\frac{dh}{dt} = \frac{h_\infty(v) - h}{\tau_h(v)}, \tag{3.12}$$

$$\frac{dn}{dt} = \frac{n_\infty(v) - n}{\tau_n(v)}. \tag{3.13}$$

Juntando as equações das portas e a equação 3.5 obtemos finalmente o modelo de Hodgkin-Huxley:

$$C\frac{dv}{dt} = \overline{g}_{Na}m^3h(v_{Na} - v) + \overline{g}_K n^4(v_K - v) + \overline{g}_L(v_L - v) + I, \tag{3.14}$$

$$\frac{dx}{dt} = \frac{x_\infty(v) - x}{\tau_x(v)}, \tag{3.15}$$

onde x é as portas m, h e n.

Para modelar um único neurônio é necessário um sistema de 4 equações diferenciais. Isso mostra como a estrutura do cérebro é bastante complexa.

3.3 O PERCEPTRON

Apesar do modelo para o neurônio biológico chegar em 1952 com Hodgkin e Huxley [22], a introdução de um modelo para o neurônio computacional já vinha sendo discutido a aproximadamente uma década antes. Em 1943, McCulloch e Pitts introduziram a ideia de redes neurais como meios de processamento de dados e abriram portas para pesquisas nessa área por pesquisadores nos campos de matemática, física e engenharia [20]. Mas apenas em 1958 (seis anos depois dos artigos de Hodgkin e Huxley) que Rosenblatt propõe o perceptron. É interessante perceber que o desenvolvimento da teoria computacional do neurônio acompanha a teoria biológica. É válido apontar que o desenvolvimento da teoria de redes neurais artificiais depende do entendimento de como funciona um cérebro e, em

um nível mais fundamental, o neurônio biológico. Por causa disso os trabalhos de Hodgkin e Huxley merecem atenção pelo fato de terem obtido grandes avanços na compreensão do funcionamento físico de um neurônio biológico. O modelo de McCulloch e Pitts é mostrado na figura 3.4. Observando a figura, o neurônio recebe os dados x, processa esses dados de acordo com uma função g e o resultado passa por uma etapa final que é uma função de ativação. Geralmente x é um vetor de tamanho n e as flechas são chamadas de pesos. A função g é uma aplicação que recebe o vetor x e retorna um valor real. Ela pode ser representada na forma 3.16.

$$g : \mathbb{R}^n \to \mathbb{R} \tag{3.16}$$

Geralmente g assume a forma de um produto interno entre os valores de entrada e os pesos, ou seja:

$$g(x) = \sum_{i=0}^{n} w_i x_i. \tag{3.17}$$

Esse modelo aceita valores de entrada binários, ou seja, $x_i \in \{0, 1\}$, $i \in \{1, 2, \ldots, n\}$,

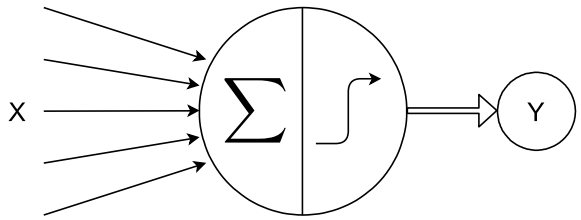

Figura 3.4 – Modelo de McCulloch e Pitts.

e retorna um valor também binário. Quando algum valor de entrada é igual a zero, ele é chamado de inibidor. No caso do valor ser um, ele é chamado de excitador. As entradas que possuem valor zero têm grande peso na decisão do neurônio enquanto que as entradas com valores iguais a um, quando combinadas podem afetar sua decisão. A decisão, por sua vez depende de um valor que vamos chamar de limiar (*threshold*). Se o valor de $g(x)$ for maior que esse limiar, a saída do neurônio será um, caso contrário será zero. Matematicamente essa decisão é representada na seguinte forma:

$$y = \begin{cases} 0 & \text{se } g(x) \geq \theta, \\ 1 & \text{se } g(x) < \theta. \end{cases} \tag{3.18}$$

O perceptron de Rosenblatt possui praticamente a mesma estrutura que o neurônio de McCulloch e Pitts, com a diferença de que ele é capaz de executar tarefas mais sofisticadas, como classificação. O perceptron aceita n valores de entrada, processa esses valores usando

uma função g e depois o resultado passa por uma função de ativação. Novamente a função de processamento g pode ser um produto interno entre os dados de entrada, mas aqui podemos adicionar uma constante b a esse valor, chamada de viés (*bias*). Chamando esse resultado de v temos:

$$v = g(x) = \sum_{i=1}^{n} x_i w_i + b. \tag{3.19}$$

Passando agora v por uma função de ativação f temos finalmente a saída do perceptron:

$$y = f(v). \tag{3.20}$$

A função de ativação f pode possuir diversas formas de acordo com a necessidade do problema. A tabela 3.1 mostra alguns exemplos. As funções mais comuns para ativação são a função logística, a tangente hiperbólica e a ReLU.

Funções de ativação	
Identidade	$f(x) = x$
Binary step	$f(x) = \begin{cases} 0 & \text{if } x < 0, \\ 1 & \text{if } x \geq 0 \end{cases}$
Logística	$f(x) = \frac{1}{1+e^{-x}}$
Hiperbólica	$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
ReLU	$f(x) = \begin{cases} 0 & \text{if } x \leq 0, \\ x & \text{if } x > 0 \end{cases}$
GELU	$f(x) = \frac{1}{2}x\left(1 + erf\left(\frac{x}{\sqrt{2}}\right)\right)$

Tabela 3.1 – Exemplos de funções de ativação.

Um dos trabalhos que o perceptron consegue desempenhar bem é a classificação de dados em categorias [24]. Como exemplo, vamos considerar um caso onde queremos classificar vetores de duas dimensões em duas classes, $\mathcal{C}_1$ e $\mathcal{C}_2$. Esses vetores podem representar dimensões de frutas, parâmetros de uma máquina, ou qualquer outra coisa que possa ser colocada em valores. Podemos então encarar o perceptron como um mapa que recebe vetores do $\mathbb{R}^2$ e retorna um valor inteiro. Se o vetor for da classe $\mathcal{C}_1$, o valor de saída do perceptron deverá ser 1. Caso o vetor seja da classe $\mathcal{C}_2$, o perceptron deverá retornar o valor 0. Para treinar o perceptron devemos ter dados que já estão classificados para assim testar em dados que ainda não foram classificados. O objetivo final é treinar o perceptron de tal forma que suas previsões nos dados de teste (ainda não classificados) sejam mais próximas possíveis dos dados reais (*targets*). Matematicamente significa minimizar uma função custo ou energia. Comumente as funções mais empregadas são o erro quadrático médio (equação 3.21) e o

erro absoluto médio (equação 3.22),

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{3.21}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|, \tag{3.22}$$

onde n é o número de amostras de dados, y_i é o valor observado da amostra i e $\hat{y}_i$ é o valor previsto pelo perceptron da amostra i. Mais precisamente, precisamos achar pesos w que minimizem a função custo. Os algoritmos que resolvem esse problema são chamados de algoritmos de aprendizado, pois ele são os responsáveis por ajustar os pesos (conexões) do perceptron. Existem diversos algoritmos de aprendizado na literatura, sendo alguns deles o *gradient descent* [25] [26], o método de Newton e o *conjugate gradient* [27]. Um dos métodos mais comuns é o *back-propagation*, que é baseado no algoritmo *gradient descent*. Esse método é tão importante que merece uma seção neste trabalho.

3.4 APRENDIZAGEM PROFUNDA

Aprendizagem profunda (*deep learning*) é uma subárea de aprendizado de máquina onde são criadas redes neurais mais sofisticadas com melhor poder de processamento de informação. Essa subárea se interessa por tarefas mais complexas e desafiadoras como processamento de linguagem natural, reconhecimento de imagem e reconhecimento de fala [28]. Atualmente aprendizagem profunda é a área mais dominante dentro de aprendizado de máquina devido aos seus poderosos algoritmos e também às infinitas possibilidades de aplicação [29].

Essencialmente um algoritmo de aprendizagem profunda consiste em processar os dados de entrada por mais de uma camada de neurônios, tornando a rede neural mais complexa e capaz de realizar tarefas mais complicadas. Observando novamente a figura 3.4, os dados de entrada são agregados por uma função g e depois passam por uma função de ativação. No caso de uma rede neural com mais camadas, os dados passam por funções de ativação sequenciais permitindo assim um processamento de informação mais profundo. De forma geral, o perceptron pode ser representado pela seguinte função:

$$y_i = h(x_i, \theta), \tag{3.23}$$

onde y_i é a previsão, x_i são os dados de entrada e θ são os parâmetros da função, podendo ser os pesos w e possivelmente algum viés. Podemos processar o resultado y_i novamente

aplicando a função h, ou seja:

$$y_i' = h'(y_i, \theta'). \tag{3.24}$$

Repetindo esse processo N vezes obtemos a seguinte aplicação H:

$$H(x_i, \Theta) = h_N(\dots h_2(h_1(x_i, \theta_1), \theta_2), \theta_N), \tag{3.25}$$

onde Θ representa todos os parâmetros θ_i. A função h_i é a combinação das equações 3.19 e 3.20 podendo cada uma usar uma função de ativação arbitrária (ver tabela 3.1). Existem vários algoritmos que seguem o modelo da função H [30], mas um que merece atenção pelo seu caráter didático é o *multilayer-perceptron* [31], que possui camadas sequenciais de perceptrons, uns conectados aos outros. Observando a figura 3.5, temos uma rede neural com quatro camadas, sendo a primeira de entrada de dados ($x \in \mathbb{R}^2$), a segunda e terceira de processamento de dados, chamadas de camadas escondidas (*hidden layers*) e a camada de saída responsável pela saída final da rede. Cada neurônio da segunda camada vai receber os

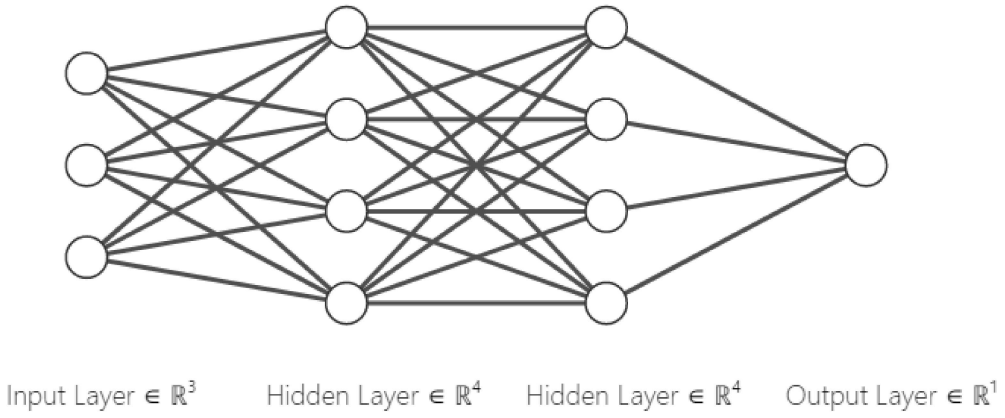

Figura 3.5 – Esquema de uma rede neural com duas camadas escondidas.

dados e processá-los de acordo com a equação 3.23:

$$v_i = \sum_{j=1}^{3} w_{ij} x_j = w_{i1}x_1 + w_{i2}x_2 + w_{i3}x_3, \tag{3.26}$$

$$y_i = f(v_i), \tag{3.27}$$

onde o subíndice i indica o neurônio i. Podemos ver que cada neurônio da segunda camada pode ser representado pelo seguinte mapa:

$$g : \mathbb{R}^3 \to \mathbb{R}. \tag{3.28}$$

Com os dados da segunda camada processados, passamos para a terceira camada de neurônios. Cada neurônio irá processar os dados da seguinte maneira:

$$v_i = \sum_{j=1}^{4} w_{ij} x_j = w_{i1}x_1 + w_{i2}x_2 + w_{i3}x_3 + w_{i4}x_4, \tag{3.29}$$

$$y_i = f(v_i), \tag{3.30}$$

onde x_i é a saída do neurônio i da segunda camada. Cada neurônio da terceira camada pode ser representado pelo seguinte mapa:

$$g : \mathbb{R}^4 \to \mathbb{R}. \tag{3.31}$$

O que vale ressaltar aqui é que os neurônios da terceira camada não recebem os dados de entrada puros, mas sim eles processados pela segunda camada. Em outras palavras, esses neurônios já recebem dados filtrados. A última camada é formada por apenas um único neurônio, sendo ele representado na seguinte maneira:

$$v = \sum_{j=1}^{4} w_j x_j = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4, \tag{3.32}$$

$$y = f(v), \tag{3.33}$$

onde cada x_i é a saída do neurônio i da terceira camada. Esse último neurônio pode ser representado pelo seguinte mapa:

$$g : \mathbb{R}^4 \to \mathbb{R}. \tag{3.34}$$

3.5 ALGORITMO BACK-PROPAGATION

Até agora foi discutido como as redes neurais processam os dados usando funções de ativação e agregadores (como por exemplo o produto interno entre os pesos e os dados de entrada). Nesta seção vamos mostrar como uma rede neural aprende com os dados e ver em detalhes o algoritmo *back-propagation*.

Seja $x = \{x_i\}_{i=1}^{n}$ uma sequência de dados de entrada (podendo cada x_i ser um vetor

de dimensão arbitrária ou um escalar real) e $y = \{\overline{y}_i\}_{i=1}^n$ seus respectivos rótulos (*labels*). Suponha que x e y tenham alguma relação de acordo com alguma função h, podendo ela ser não-linear. Um exemplo é $x \in [-\pi, \pi]$ e $y = h(x) = \sin(x)$. Seja $g(x, \theta)$ uma rede neural que depende de x e dos parâmetros θ. Seja também $y_i = g(x_i, \theta)$ uma previsão da rede neural. O problema de aprendizado de máquina consiste em encontrar parâmetros θ tais que o valor das previsões y_i sejam o mais próximo possível dos valores reais $\overline{y}_i$. De forma equivalente, precisamos encontrar os parâmetros θ que minimizam uma função custo C, ou seja, devemos resolver:

$$\theta = \arg\min_{\theta} \mathrm{C}(\overline{y}, y), \tag{3.35}$$

onde $\overline{y}$ são os dados reais, y são as previsões e θ são os parâmetros da rede neural. Para ilustrar, vamos considerar um único perceptron e uma função custo $\mathcal{E}$. Sabemos que o campo de indução local do perceptron é dado por

$$v_j = \sum_{j=1}^{n} w_{ji} x_i, \tag{3.36}$$

onde estamos desconsiderando o viés para tornar os cálculos mais simples. A saída do perceptron é dado pela aplicação da função de ativação f:

$$y_j = f(v_j). \tag{3.37}$$

O algoritmo *back-propagation* consiste em aplicar uma correção δw_{ji} nos pesos w_{ji}. Essa correção é proporcional a derivada parcial da função custo em relação ao peso w_{ji}, ou seja:

$$\delta w_{ji} \propto \frac{\partial \mathcal{E}}{\partial w_{ji}}. \tag{3.38}$$

Sabemos que $\mathcal{E} = \mathcal{E}(\overline{y}, y)$, onde y depende dos pesos w_{ji}. Utilizando a regra da cadeia temos:

$$\frac{\partial \mathcal{E}}{\partial w_{ji}} = \frac{\partial \mathcal{E}}{\partial y_j} \frac{\partial y_j}{\partial v_j} \frac{\partial v_j}{\partial w_{ji}}. \tag{3.39}$$

Com os resultados da equação anterior, chegamos então na seguinte relação:

$$\delta w_{ji} = \eta \frac{\partial \mathcal{E}}{\partial y_j} \frac{\partial y_j}{\partial v_j} \frac{\partial v_j}{\partial w_{ji}}, \tag{3.40}$$

onde η é a razão de aprendizado (*learning rate*). Essa relação é também chamada de *delta rule* [20]. A cada vez que o algoritmo é executado, os pesos w_{ji} são atualizados na seguinte forma:

$$w_{ji}(n+1) = w_{ji}(n) + \eta \frac{\partial \mathcal{E}(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)}, \tag{3.41}$$

ou seja:

$$w_{ji}(n+1) = w_{ji}(n) + \delta w_{ji}(n), \tag{3.42}$$

onde $w_{ji}(n+1)$ são os novos pesos da rede neural.

3.6 REDES NEURAIS RECORRENTES

Um tipo e tarefa que as redes neurais apresentadas até agora não conseguem desempenhar muito bem é a previsão de dados sequenciais. Dados sequenciais são informações que estão endereçadas pelo tempo. Exemplos desse tipo de dados é o preço de uma ação, as legendas de um filme e também o número de casos por dia de uma doença em um surto ou epidemia. O que faz o perceptron ou o *multilayer-perceptron* não serem bons nesse tipo e tarefa é por eles não levarem em conta o fato dos dados serem sequenciais. Em outras palavras, essas redes neurais não conseguem ver autocorrelações nos dados. De forma geral, uma rede neural recorrente recebe um dado de entrada, processa ele e também guarda parâmetros que aqui chamaremos de estado da rede. Esse estado pode ser interpretado como a memória da rede neural com relação aos dados passados, pois ele muda de acordo com os novos dados. Observando a figura 3.6, vemos que a rede recebe o dado x_1, processa ele de acordo com o estado H e retorna y_1. No próximo passo recebe x_2 e retorna y_2. O mesmo procedimento é feito até a rede percorrer por todos os dados. A diferença aqui em relação a uma simples rede neural é que o estado atual H_i depende do estado anterior H_{i-1}. Matematicamente, uma

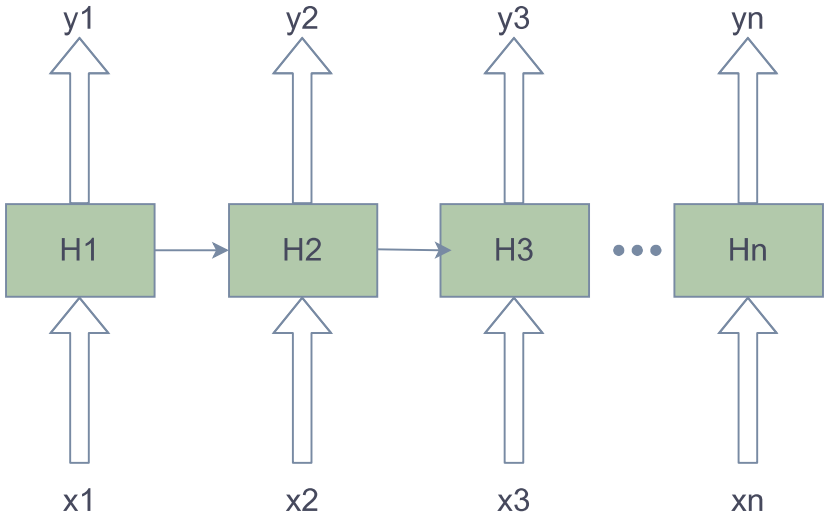

Figura 3.6 – Esquema do funcionamento de uma rede neural recorrente.

rede neural recorrente pode ser representada por uma transformação T da seguinte forma:

$$H_i = T(H_{i-1}, x_{i-1}). \tag{3.43}$$

A saída da rede é então obtida por:

$$y_i = f(\theta, H_i, x_i), \tag{3.44}$$

onde f é uma função de ativação e θ é um conjunto de parâmetros treináveis. As redes neurais recorrentes mais conhecidas são a *long short-term memory* (LSTM) e a *gated recurrent unit* (GRU)[15]. Um dos algoritmos mais usados para treinar esse tipo de rede é o *back-propagation through time* (BPTT) [32].

3.7 COMPUTAÇÃO DE RESERVATÓRIO

Redes neurais recorrentes representam uma grande variedade de modelos que são capazes de realizar vários tipos de tarefas. Com a propriedade da recorrência de um estado interno em cada passo do treinamento, essas redes se tornaram bem mais capazes que as redes neurais tradicionais (que não possuem recorrência). De acordo com Mantas et al. em [7], essa existência de recorrência tem profundo impacto, como por exemplo:

- com essa característica de recorrência, uma rede neural recorrente pode ser vista como um sistema dinâmico, enquanto que uma rede neural do tipo *feedforward* é vista com uma função;
- quando treinada com dados sequenciais, uma rede neural recorrente é capaz de preservar em seu estado uma transformação (não linear) dos dados passados (que pode ser interpretada como a história dos dados). Em outras palavras, esse tipo de rede neural possui uma memória dinâmica, o que a torna capaz de perceber padrões e comportamentos em dados sequenciais.

Apesar de serem ótimos processadoras de informação, essas redes perdem em desempenho pelo fato dos algoritmos de treinamento utilizados serem bastante custosos computacionalmente. Um exemplo é o *back-propagation through time*, já citado neste trabalho, que é um método baseado em *gradient-descent*. Esse método procura ajustar os parâmetros da rede a cada iteração, minimizando uma função custo $\mathcal{E}$. Como uma rede neural recorrente geralmente possui muitos parâmetros, o seu treinamento pode ser tornar bastante longo. Ainda de acordo com Mantas et al. [7], as redes neurais recorrentes possuem as seguintes desvantagens:

- a constante mudança nos parâmetros da rede neural durante treinamento pode levar a bifurcações e como consequência a convergência não pode ser garantida;

- muitas iterações podem ser necessárias para o treinamento de uma rede neural recorrente e como esse tipo de rede pode ter muitos parâmetros, o treinamento pode levar um tempo considerável;
- dependendo da complexidade do modelo, o ajuste de muitos parâmetros será necessário e para fazer isso geralmente são necessários habilidade e bastante experiência.

Tomando essas desvantagens como motivação, uma abordagem para redes neurais recorrentes totalmente nova foi proposta por duas pessoas independentemente. Maass et al. em [33] propõe tal modelo usando o nome de *Liquid state machines* enquanto que Herbert Jaeger em [34] o propõe sob o nome *Echo state network*, ou ESN. Esse tipo de abordagem, atualmente mais comumente chamado de computação de reservatório (*reservoir computing*), tenta evitar as desvantagens do método *gradient-descent*.

A grande novidade que computação por reservatório traz é a forma em que os parâmetros são treinados (figura 3.7). Enquanto que o método *gradient-descent* leva em conta todos os parâmetros de uma rede neural recorrente durante o treinamento, a nova abordagem considera apenas os parâmetros de saída da rede neural, ou seja, a camada que gera o resultado final.

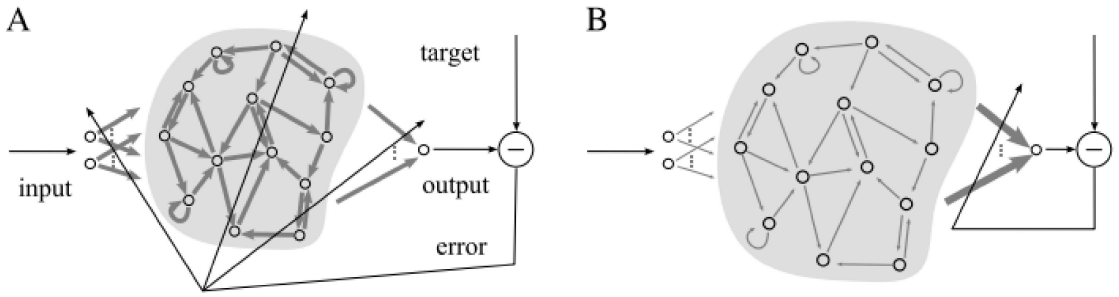

Figura 3.7 – Fluxo dos dados em dois tipos de redes neurais recorrentes: convencional (A) e computação de reservatório (B). Vemos que em uma rede convencional a correção do erro percorre por todos os parâmetros da rede, enquanto que em computação de reservatório apenas os pesos de saída são ajustados Figura retirada de [7].

Para criar uma rede neural recorrente usando essa abordagem, seguimos os seguintes passos [7]:

- uma rede neural recorrente é criada de forma aleatória. Essa rede é chamada de reservatório (*reservoir*) e permanece inalterada durante o treinamento. O reservatório serve para guardar o histórico dos dados e é o responsável, juntamente com os dados, de transformar o estado da ESN;
- a saída da rede neural é gerada como uma combinação linear da forma $A \cdot X + B$, onde X pode ser encarado como o estado da ESN e A e B parâmetros treináveis por regressão linear.

Para ilustrar como funciona de modo geral a abordagem de computação de reservatório, seja $u \in \mathbb{R}^{N_u}$ o dado de entrada e $y \in \mathbb{R}^{N_y}$ seu respectivo rótulo. Aqui estamos interessados em uma mapa $T : \mathbb{R}^{N_u} \to \mathbb{R}^{N_y}$ que faça a previsão dos valores $\hat{y}$ com o menor erro possível em relação aos valores reais y. Na abordagem de computação de reservatório a forma geral desse mapa é dada por [7]:

$$y(n) = f_{out}(W_{out} \cdot x\,[u(n)]), \tag{3.45}$$

onde f_{out} é uma função qualquer, W_{out} é a matriz dos parâmetros ajustáveis, e x é o estado da rede neural que tem a mesma dimensão que o reservatório da rede neural e depende da entrada $u(n)$. n representa o n-ésimo dado. Após a rede neural percorrer todos os dados de treinamento, a matriz W_{out} é corrigida usando métodos que serão discutidos nas próximas seções.

3.8 ECHO STATE

Nesta seção vamos ver em detalhes o funcionamento da ESN. Observando a figura 3.8 vemos que na camada de entrada os dados $u(n)$ sofrem a seguinte transformação:

$$W_{in} \cdot u(n), \tag{3.46}$$

onde $W_{in} \in \mathbb{R}^{N_x \times N_u}$. Aqui estamos ignorando o possível viés para simplificar os cálculos.

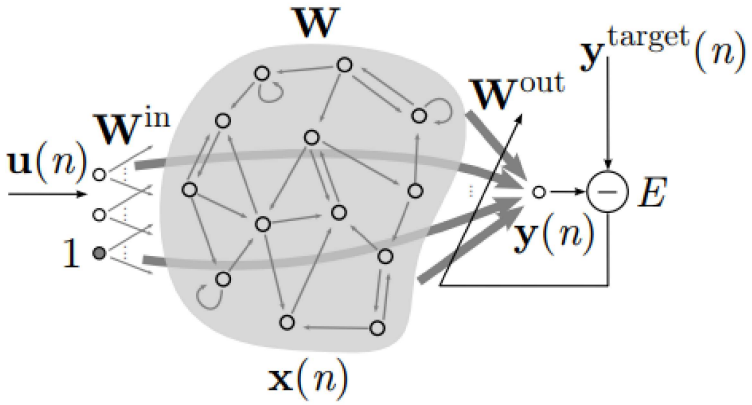

Figura 3.8 – Desenho esquemático da *echo state network*. Figura retirada de [8].

Para obter o estado temporário $\tilde{x}$ da ESN utilizamos a seguinte equação:

$$\tilde{x} = \tanh\left(W_{in} \cdot u(n) + W \cdot x(n-1)\right), \tag{3.47}$$

onde $W \in \mathbb{R}^{N_x \times N_x}$ é o reservatório da ESN e $x(n-1) \in \mathbb{R}^{N_x}$ é o seu estado na iteração

anterior. Finalmente o estado da ESN é dado por:

$$x(n) = (1 - \alpha)x(n - 1) + \alpha\tilde{x(n)}, \tag{3.48}$$

onde $\alpha \in (0, 1]$. Para obter a previsão $\hat{y}$ utilizamos a seguinte transformação:

$$\hat{y}(n) = W_{out} \cdot [u(n); x(n)], \tag{3.49}$$

onde $W_{out} \in \mathbb{R}^{N_y \times (N_u + N_x)}$ é a matriz de saída e $[\cdot;\cdot]$ representa a concatenação de vetores. De acordo com Mantas em [8], o método de computação de reservatório para a ESN é composto pelos seguintes passos:

1. gerar os parâmetros W_{in} e α e o reservatório W de forma aleatória;
2. processar os dados de entrada $u(n)$ de acordo com as equações apresentadas nesta seção e coletar os estados obtidos $x(n)$;
3. computar a matriz W_{out} usando regressão linear, minimizando uma função custo $\mathcal{C}$ entre os valores reais $y(n)$ e os valores previstos $\hat{y}(n)$;
4. com os pesos W_{out} treinados, fazer previsões em novos dados.

Geralmente W_{in} é gerado de acordo com uma distribuição uniforme no intervalo $[-a, a]$, onde a é chamado de escala de entrada (*input scaling*), e W_{out} é inicializado com todas as suas entradas iguais a zero. O maior desafio para fazer uma boa ESN é a construção do reservatório W. Mantas em [8] gera W de acordo com uma distribuição uniforme no intervalo $[0, 1]$, zera todos os elementos que forem menores que s ($s \in (0, 1)$) e divide cada elemento resultante de W pelo seu raio espectral $\rho(W)$, ou seja, seu maior autovalor absoluto.

Usando teoria dos grafos podemos explorar outras topologias para o reservatório da ESN. Liu et al. em [35] utiliza a teoria do grafo mundo pequeno (*small world*) para criar um novo reservatório para a ESN, mostrando melhor desempenho comparado ao reservatório aleatório. Liebald em [36] faz um estudo comparativo entre várias topologias para o reservatório da ESN e (infelizmente) chega à conclusão de que nenhuma topologia tem melhora significativa em relação ao reservatório aleatório.

Como já foi dito, o treinamento da ESN é bem diferente de uma rede neural recorrente convencional. Aqui estamos preocupados apenas com os parâmetros de saída, ou seja, a matriz W_{out}. Vamos supor que na n-ésima iteração coletamos o estado $x(n)$ gerado pelo dado de entrada $u(n)$. Vamos supor também que em cada época, ou seja, quando a rede neural percorre todos os dados de treinamento, utilizamos T amostras de dados. Guardando

todos os estados coletados em uma matriz X, chamada de matriz design (*design matrix*), ao final de T iterações, X toma a seguinte forma:

$$X = \begin{pmatrix} x'(1) & x'(2) & \dots & x'(T) \end{pmatrix}, \tag{3.50}$$

onde $x'(j) = [u(j), x(j)]^T$ são os vetores colunas da matriz X, lembrando novamente que estamos omitindo o viés para simplificar os cálculos. É fácil ver que X tem dimensão $N_u + N_x$ por T. Encontrar a matriz W_{out} que minimiza uma função custo entre os valores previstos e os valores reais se resume a seguinte equação:

$$Y_{target} = W_{out} \cdot X, \tag{3.51}$$

onde Y_{target} são os rótulos de todos os valores de entrada $u(n)$. Existem métodos conhecidos para resolver a equação 3.51. Mantas em [8] recomenda o uso do método *Ridge regression*, conhecido também como *Tikhonov regularization*:

$$W_{out} = Y_{target} \cdot X^T \left(X \cdot X^T + \beta I \right)^{-1}, \tag{3.52}$$

onde β é chamado de coeficiente de regularização e I é a matriz identidade.

Como é de se imaginar, o treinamento por computação de reservatório é extremamente rápido por se tratar apenas de uma regressão linear [17]. O único desafio (computacional) é a inversão do termo entre parênteses na equação 3.52.

3.9 DEEP ECHO STATE

Para aumentar ainda mais a capacidade de processamento da ESN, podemos criar um modelo de aprendizagem profunda onde temos camadas de reservatórios sucessivas. Chamamos essa rede neural de *deep* ESN. A *deep* ESN é bastante recente mas já existem trabalhos que mostram seu bom desempenho em várias tarefas. Gallicchio et al. em [37] fazem um estudo comparativo entre a *deep* ESN e a ESN na detecção da doença de Parkinson e verificam que o modelo de aprendizagem profunda mostra um desempenho melhor comparado ao modelo tradicional da ESN. Kim et al. em [17] também fazem um estudo comparativo entra vários tipos de redes neurais recorrentes, incluindo a *deep* ESN, na previsão de séries temporais dos mais variados tipos. Eles mostram que a *deep* ESN não só tem um desempenho melhor que as outras redes neurais, mas também um tempo de treinamento muito menor que a LSTM e GRU, por exemplo.

Observando a figura 3.9, os dados de entrada $u(t)$ sofrem a seguinte transformação na

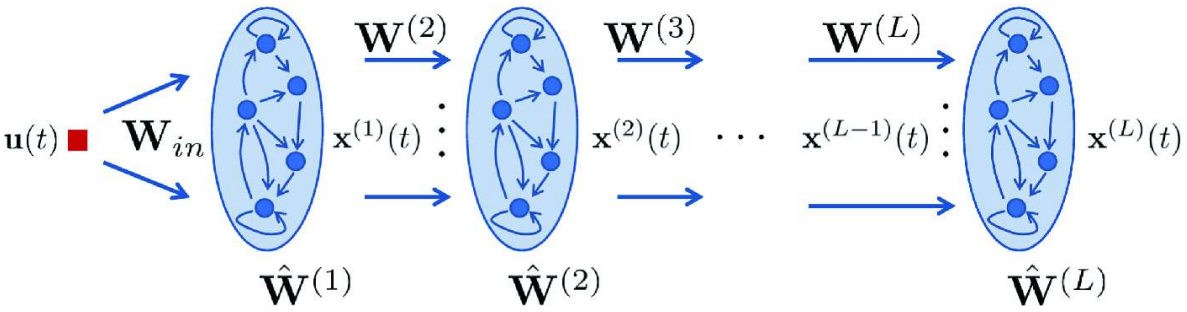

Figura 3.9 – Modelo esquemático de uma *deep* ESN. Figura retirada de [9].

camada de entrada:

$$u(t) \cdot W_{in}, \tag{3.53}$$

onde W_{in} são os pesos de entrada. Para obter o estado $x^1(t)$ vindo do primeiro reservatório, fazemos:

$$x^{(1)}(t) = (1 - \alpha^{(1)})x^{(1)}(t-1) + \alpha^{(1)} \tanh\left(W_{in} \cdot u(t) + \hat{W}^{(1)} \cdot x^{(1)}(t-1)\right), \tag{3.54}$$

onde $W_{in} \in \mathbb{R}^{N_r \times N_u}$, N_r sendo a dimensão do estado e N_u a dimensão dos dados de entrada. O sobrescrito 1 no estado e no α indica que eles estão na primeira camada. A segunda camada será um pouco diferente:

$$x^{(2)}(t) = (1 - \alpha^{(2)})x^{(2)}(t-1) + \alpha^{(2)} \tanh\left(W^{(2)} \cdot x^{(1)}(t) + \hat{W}^{(2)} \cdot x^{(2)}(t-1)\right), \tag{3.55}$$

onde temos uma nova matriz $W \in \mathbb{R}^{N_r \times N_r}$, chamada de matriz de interconexão. Essa matriz é responsável por passar informações dos dados entre camadas. De forma geral, a l-ésima camada da *deep* ESN pode ser escrita da seguinte forma:

$$x^{(l)}(t) = (1 - \alpha^{(l)})x^{(l)}(t-1) + \alpha^{(l)} \tanh\left(W^{(l)} \cdot x^{(l-1)}(t) + \hat{W}^{(l)} \cdot x^{(l)}(t-1)\right). \tag{3.56}$$

Para coletar toda a informação que os reservatórios produzem e transmitem uns para os outros, definimos um vetor H que é a concatenação dos estados dos reservatórios. Matematicamente, H é dado por:

$$H = \left[x^{(1)}(t) \quad x^{(2)}(t) \quad x^{(3)}(t) \quad \ldots \quad x^{(L)}(t)\right], \tag{3.57}$$

onde L é a quantidade de reservatórios. Aqui podemos chamar H de estado total da *deep* ESN. Comparando H com o estado convencional da equação 3.48, podemos ver que H carrega mais informação e também essa informação é mais filtrada por causa das sucessivas camadas. Por exemplo, a primeira entrada de H, ou seja, $x^1(t)$ passa pelas mesmas transformações que o estado da ESN. Mas a partir da segunda camada, os novos estados $x^k(t)$ sofreram mais processamento e por causa disso podem conter informações mais relevantes

sobre os dados estudados. Para se ter uma ideia de como a *deep* ESN tem um poder de abstração maior que uma ESN convencional, o mapa que leva dos dados de entrada para o estado total é dado por:

$$g : \mathbb{R}^{N_u} \to \mathbb{R}^{N_r \cdot L}, \tag{3.58}$$

onde N_u é o tamanho dos dados de entrada, N_r é o tamanho dos reservatórios e L é a quantidade de reservatórios (camadas). Já no caso da ESN a transformação que faz o mesmo procedimento é dada por

$$h : \mathbb{R}^{N_u} \to \mathbb{R}^{N_r}, \tag{3.59}$$

ou seja, a *deep* ESN leva os dados de entrada para um espaço L vezes maior que no caso da ESN (considerando as duas redes com o reservatórios do mesmo tamanho). Ou seja, qualquer padrão que é capturado pelo estado da ESN pode eventualmente ser capturado pela *deep* ESN, pois sempre podemos fazer uma combinação linear em $\mathbb{R}^{N_r \cdot L}$ que gere um vetor em $\mathbb{R}^{N_r}$.

Para obtermos a saída da rede neural, primeiramente devemos calcular o estado total médio que é dado por

$$S(H) = \frac{1}{T} \sum_{n=1}^{T} H(n), \tag{3.60}$$

onde $H(n)$ é o estado total da *deep* ESN referente aos dados $u(n)$. Aqui estamos supondo que temos T amostras de dados, ou seja, T pares $(u(n), y(n))$, onde $y(n)$ é o rótulo. Finalmente a saída da rede é dada por:

$$\hat{y}(n) = W_{out} \cdot S(H), \tag{3.61}$$

onde $W_{out} \in \mathbb{R}^{N_y \times N_r \cdot L}$. Novamente, somente W_{out} é ajustado utilizando também a equação 3.52.

Como no caso da ESN, os reservatórios da *deep* ESN também devem satisfazer algumas condições para que a rede neural funcione corretamente. Primeiramente, para construir as matrizes W_{in} e $W^{(k)}$, devemos gerar seus valores de acordo com uma distribuição uniforme (comumente entre zero e um) e reescalar a matriz resultante de acordo com a sua norma-2. Ou seja, devemos ter:

$$||W_{in}||_2 = \sigma_{in}, \tag{3.62}$$

$$\left|\left|W^k\right|\right|_2 = \sigma. \tag{3.63}$$

onde σ_{in} e σ são as novas normas escolhidas. A norma-2 é dada pela seguinte equação:

$$||A||_2 = \sqrt{\lambda_{max}}, \tag{3.64}$$

onde λ_{max} é o maior autovalor da matriz $(A^\dagger A)$, onde $A^\dagger$ é o conjugado transposto da matriz

A. No caso dos reservatórios $\hat{W}^{(k)}$ o procedimento é diferente. Primeiramente geramos as matrizes $\hat{W}^{(k)}$ de acordo com uma distribuição uniforme entre zero e um (escolhemos esse método de geração para simplificar os cálculos, mas na próxima seção vamos mostrar como construir os reservatórios usando teoria dos grafos), depois reescalamos as matrizes para que a seguinte relação seja verdadeira [37]:

$$\max_{1\leq l\leq L} \rho\left((1-\alpha^{(l)})I + \alpha^{l}\hat{W}^{(l)}\right) < 1, \tag{3.65}$$

onde ρ representa o raio espectral da matriz, ou seja, o seu maior autovalor.

3.10 TOPOLOGIAS DE RESERVATÓRIO

Toda a teoria de computação de reservatório gira em torno do procedimento de gerar um bom reservatório. Um bom reservatório não apenas consegue capturar bem a dinâmica dos dados estudados mas também possui boa memória. Para possuir boa memória, o reservatório deve conseguir imitar comportamentos caóticos. Sabemos que sistemas dinâmicos caóticos são aqueles que são bastante sensíveis a pequenas variações nas condições iniciais. Dito isso, é ideal que a ESN (ou *deep* ESN) consiga se adaptar bem à mudanças repentinas nos dados. Neste trabalho vamos utilizar dois tipos de grafos para gerar os reservatórios das redes neurais. O primeiro será o grafo aleatório, criado por Paul Erdős e Alfréd Rényi e o segundo será o grafo pequeno mundo, criado por Duncan J. Watts e Steven Strogatz.

3.10.1 Modelo de Erdős–Rényi

Para construir o grafo aleatório precisamos de dois parâmetros: o número de nós N e a probabilidade de conexão entre nós p. Selecionando um nó da coleção de N nós, esse nó tem uma probabilidade p de se conectar com cada um dos $N-1$ nós restantes. Selecionando um segundo nó, ele terá uma probabilidade p de se conectar com os $N-2$ nós restantes, e assim por diante. Ao final do procedimento teremos um grafo aleatório. A figura 3.10 mostra um exemplo de um grafo com 50 nós e probabilidade de conexão de 4%. Existem várias formas de representação diferentes para o grafo da figura 3.10, mas a conveniente para nós será a matriz de adjacência. Essa matriz representa todas as conexões entre nós do grafo. Para ilustrar, considere o grafo da figura 3.11. O grafo é composto por quatro nós e algumas conexões. A matriz de adjacência desse grafo é dada por:

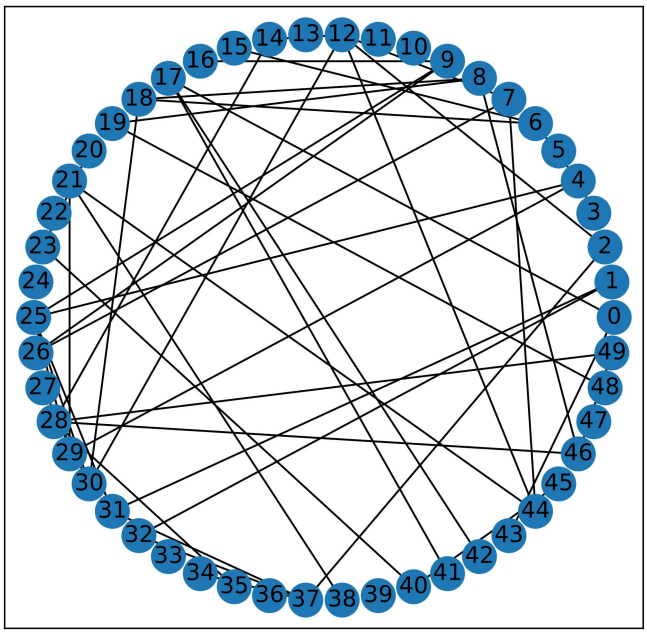

Figura 3.10 – Grafo aleatório com $N = 50$ nós e probabilidade de conexão $p = 4\%$.

$$A = \begin{bmatrix} c_{00} & c_{01} & c_{02} & c_{03} \\ c_{10} & c_{11} & c_{12} & c_{13} \\ c_{20} & c_{21} & c_{22} & c_{23} \\ c_{30} & c_{31} & c_{32} & c_{33} \end{bmatrix}, \tag{3.66}$$

onde $c_{ij} = 1$ se existir conexão entre os nós i e j e $c_{ij} = 0$ caso contrário. Cada linha ou coluna representa um nó. Começando pelo nó 0, vemos que ele tem conexão apenas com o nó 1, então na primeira linha da matriz A apenas o elemento c_{01} será igual a 1 e o resto será zero. Temos então:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ c_{10} & c_{11} & c_{12} & c_{13} \\ c_{20} & c_{21} & c_{22} & c_{23} \\ c_{30} & c_{31} & c_{32} & c_{33} \end{bmatrix}. \tag{3.67}$$

Partindo para o nó 1 vemos que ele tem conexão com todos os outros nós menos com ele mesmo. A matriz A fica então da seguinte forma:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ c_{20} & c_{21} & c_{22} & c_{23} \\ c_{30} & c_{31} & c_{32} & c_{33} \end{bmatrix}. \tag{3.68}$$

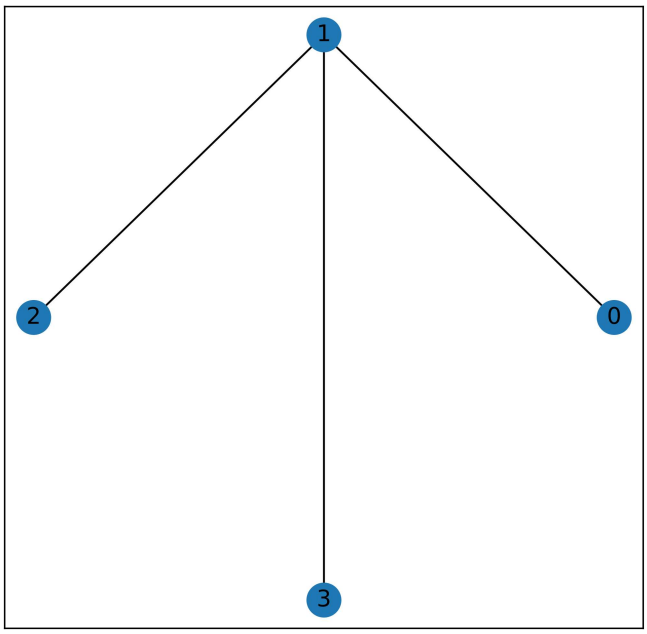

Figura 3.11 – Grafo aleatório com $N = 4$ nós e probabilidade de conexão $p = 40\%$.

Considerando agora o nó 2, vemos que ele se conecta apenas com o nó 1. Então:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ c_{30} & c_{31} & c_{32} & c_{33} \end{bmatrix}. \tag{3.69}$$

Finalmente o nó 3 se conecta apenas com o nó 1. A matriz de adjacência toma então a seguinte forma:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}. \tag{3.70}$$

A matriz A da equação 3.70 é a matriz de adjacência do grafo da figura 3.11. Essa matriz contém todas as informações que precisamos para a construção do reservatório. Agora vamos listar o procedimento para a produção do reservatório baseado no grafo aleatório:

1. definir o número de nós;
2. definir a probabilidade de conexão p;

3. para cada nó decidir se faz ou não a conexão com os nós restantes de acordo com a probabilidade p (geralmente geramos um número aleatório no intervalo $[0, 1]$ e se ele for menor que p fazemos a conexão);

4. construir a matriz de adjacência A de acordo com o procedimento mostrado;

5. gerar uma matriz B (com a mesma dimensão da matriz de adjacência) com números aleatórios no intervalo $[-\gamma, \gamma]$ de acordo com uma distribuição uniforme (geralmente $\gamma = 0.5$ [36]);

6. fazer o produto de Hadamard entre as matrizes A e B.

3.10.2 Modelo de Watts-Strogatz

Muitos sistemas reais, como redes sociais e redes de coautoria de artigos ou patentes possuem a propriedade pequeno mundo. Essa propriedade diz que dois nós quaisquer dentro da rede possuem uma distância pequena (quantidade mínima de arestas que conectam os dois nós) [38]. No caso de doenças contagiosas, a ideia de usar esse tipo de topologia é pelo fato de que as interações entre pessoas também funcionam como uma rede pequeno mundo. Como o reservatório é o responsável por possuir as propriedades de um sistema dinâmico, acreditamos que esse modelo pode se assemelhar mais à uma dinâmica que dependa de interações entre pessoas, que é o caso da propagação de uma doença contagiosa.

Para construir uma grafo pequeno mundo precisamos de três parâmetros: o número N de nós, a quantidade inicial k de vizinhos iniciais que cada nó irá possuir e a probabilidade p de rearranjo das arestas. Primeiramente geramos um grafo com N nós e cada nó tera k vizinhos próximos. Em seguida, para cada aresta, o rearranjo acontecerá de acordo com a probabilidade p. A figura 3.12 mostra um grafo pequeno mundo com $N = 50$ nós, $k = 2$ vizinhos iniciais e probabilidade de conexão $p = 4\%$. O procedimento para a criação do reservatório pequeno mundo é listado a seguir:

1. definir número de nós;

2. definir a quantidade de vizinhos iniciais para cada nó;

3. definir a probabilidade de rearranjo das arestas;

4. após a criação do grafo de acordo com os parâmetros escolhidos obter a matriz de adjacência A;

5. gerar uma matriz B (com a mesma dimensão da matriz de adjacência) com números aleatórios no intervalo $[-\gamma, \gamma]$ de acordo com uma distribuição uniforme (geralmente $\gamma = 0.5$);

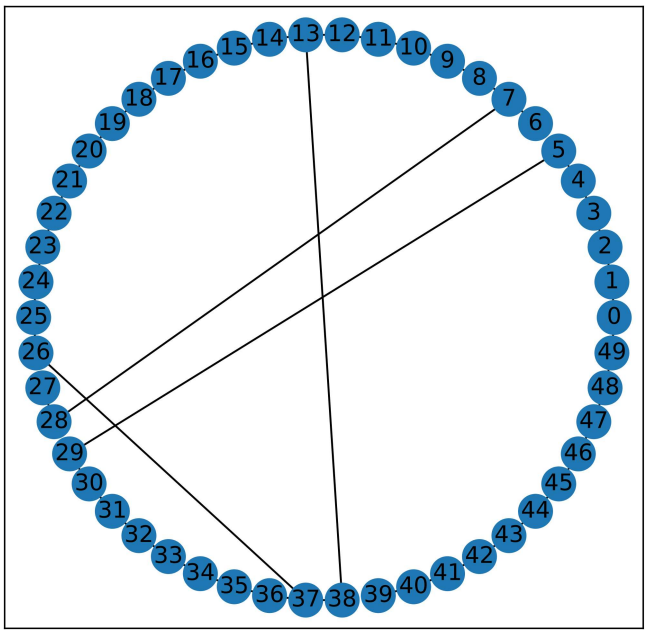

Figura 3.12 – Grafo pequeno mundo com $N = 50$ nós, $k = 2$ vizinhos iniciais e probabilidade de conexão $p = 4\%$.

6. fazer o produto de Hadamard entra as matrizes A e B.

4 METODOLOGIA

4.1 ESTRUTURA GERAL DO TRABALHO

O trabalho é dividido em três partes: coleta e pré- processamento de dados, realização dos experimentos e por último coleta e análise de resultados. Aqui realizamos três experimentos. No primeiro experimento fazemos previsões mensais nas médias móveis de casos e mortes, ambas de 14 dias, e nos casos e mortes cumulativos. No segundo experimento usamos o resultado da etapa anterior para prever o número de reprodução efetivo mensalmente. Na última etapa fazemos previsões na série temporal de casos diários (suavizada com transformada de Fourier) em cada ponto dez dias na frente com o objetivo de analisar a performance das redes neurais em cada estágio da epidemia. Todo o trabalho foi desenvolvido usando a linguagem de programação Python. A tabela 4.1 mostra a relação dos pacotes Python utilizados para a realização do estudo. O computador utilizado possui as seguintes especificações: processador AMD Ryzen 7 2700 de 8 núcleos (físicos) e 16 *threads* com frequência base de 3.2 GHz (turbo de 4.1 GHz), 32 Gigabyte de memória RAM com frequência de 2666 MHz e uma placa de vídeo AMD Radeon RX 5600 XT com 6 Gigabyte de memória de vídeo dedicada.

Pacotes Python	
Pacote	Versão
Numpy	1.19.3
Matplotlib	3.3.3
Scikit learn	0.23.2
Pandas	1.2.0
PyTorch	1.8.1+cpu
Seaborn	0.11.1
Streamlit	0.80.0

Tabela 4.1 – Pacotes Python utilizados para a realização do trabalho.

4.2 EXPERIMENTOS

Para fazer os três experimentos utilizamos quatro redes neurais: ESN com reservatório aleatório, ESN com reservatório pequeno mundo, *deep* ESN com reservatórios aleatórios e *deep* ESN com reservatórios pequeno mundo. Os dados coletados estão disponíveis no sítio [39]. Este trabalho foca apenas nos dados sobre o Distrito Federal, mas as mesmas técnicas aqui apresentadas podem ser usadas para fazer análise e previsões da situação do coronavirus

em outros estados. O arquivo baixado no endereço [39] contém dados da pandemia em todos os estados do Brasil. O arquivo está no formato CSV (*comma-separated values*), onde a separação das colunas se dá pela vírgula. O arquivo CSV bruto contém as seguintes colunas: **regiao**, **estado**, **municipio**, **coduf**, **codmun**, **codRegiaoSaude**, **nomeRegiaoSaude**, **data**, **semanaEpi**, **populacaoTCU2019**, **casosAcumulado**, **casosNovos**, **obitosAcumulado**, **obitosNovos**, **RecuperadosNovos**, **emAcompanhamentosNovos**, **interior/metropolitana**. As colunas de interesse são: **data**, **casosAcumulado**, **casosNovos**, **obitosAcumulado**, **obitosNovos**. A coluna **data** representa a data da coleta do dado, **casosAcumulado** e **obitosAcumulado** representam o número total de casos e mortes, respectivamente e **casosNovos** e **obitosNovos** representam o número de casos e mortes diários, respectivamente.

4.2.1 Previsões mensais

Neste experimento fizemos previsões mensais em quatro séries temporais: média móvel de casos (14 dias), média móvel de mortes (14 dias), casos e mortes cumulativos. As figuras 4.1 e 4.2 mostram o gráfico de casos e mortes diários (com suas médias móveis). As figuras 4.3 e 4.4 mostram a curva de casos e mortes acumulados no Distrito Federal.

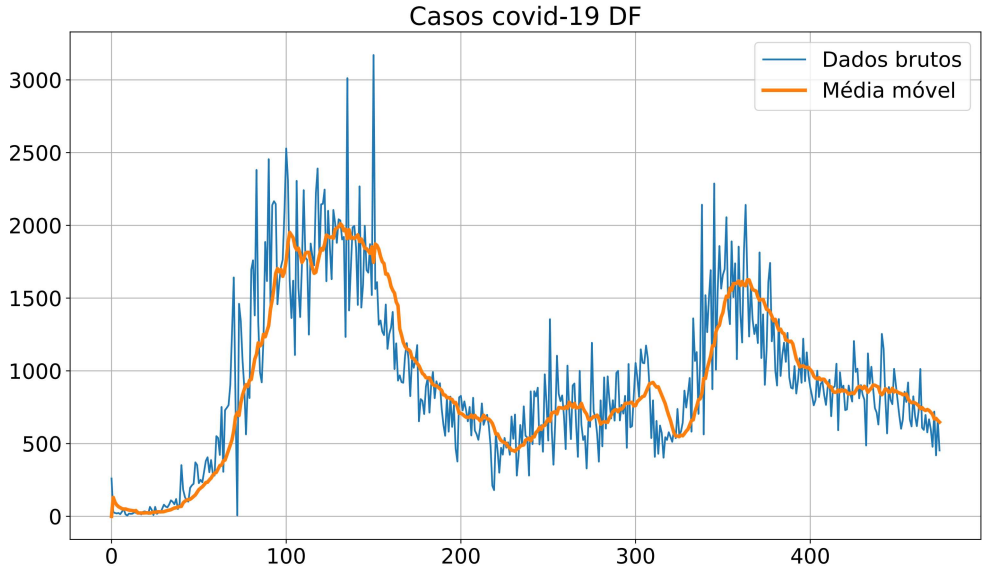

Figura 4.1 – Casos diários de coronavirus no Distrito Federal do dia 28 de março de 2020 até o dia 15 de julho de 2021. Em laranja sua média móvel de 14 dias.

Como as séries temporais de casos e mortes diários são bastante ruidosos, vamos optar por suas respectivas médias móveis de 14 dias. O processo para construir uma média móvel de uma série temporal é o seguinte: suponha uma série temporal $X = (x_1, x_2, x_3, \ldots, x_T)$, onde T é o total de passos no tempo. Ou seja, X é um vetor de dimensão T. Para extrair a

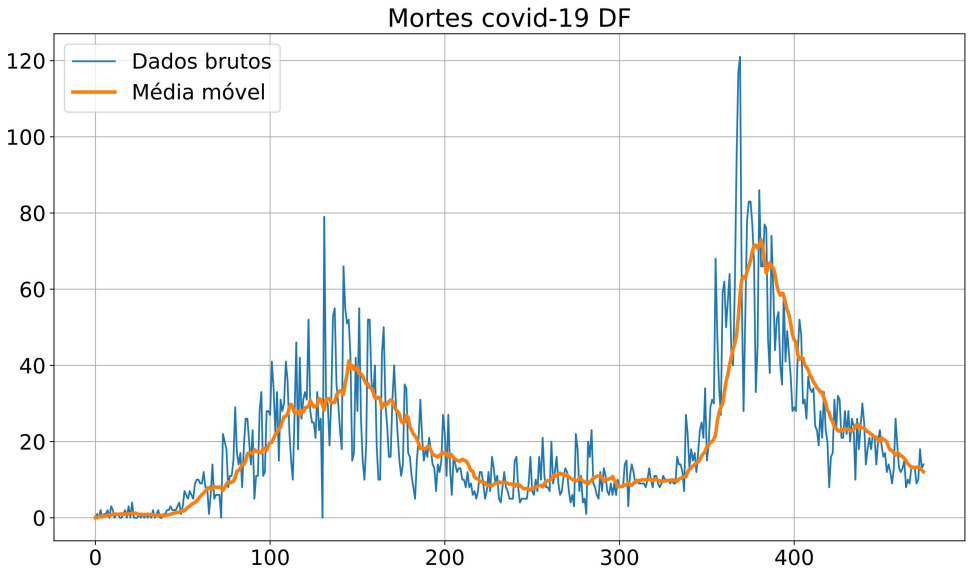

Figura 4.2 – Mortes diárias de coronavirus no Distrito Federal do dia 28 de março de 2020 até o dia 15 de julho de 2021. Em laranja sua média móvel de 14 dias.

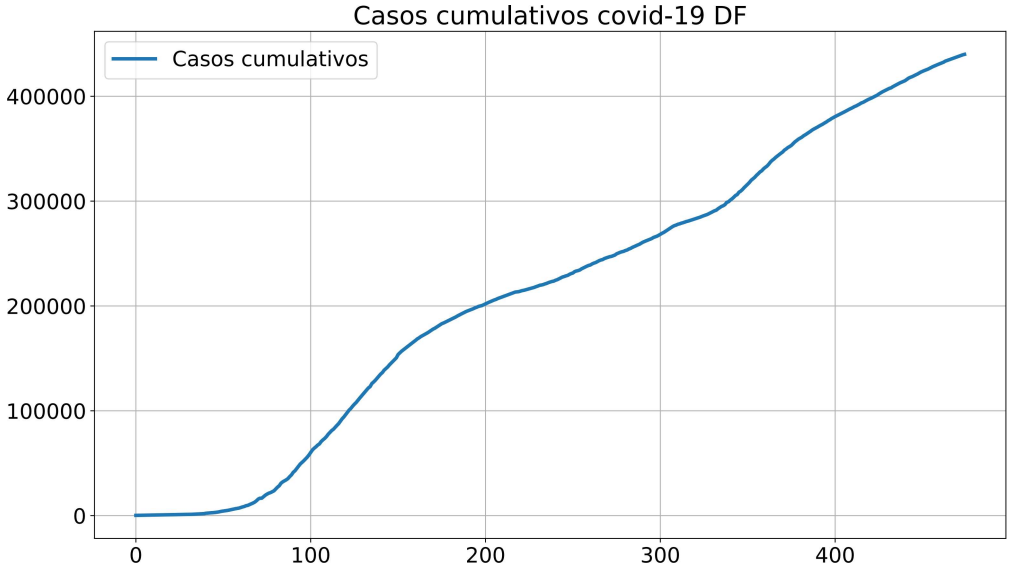

Figura 4.3 – Casos acumulados de coronavirus no Distrito Federal do dia 28 de março de 2020 até o dia 15 de julho de 2021.

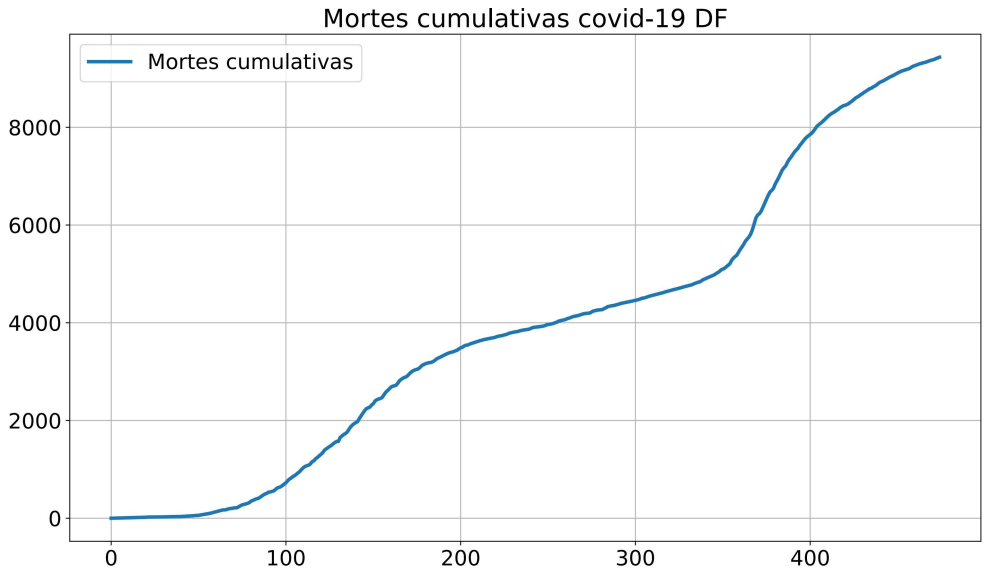

Figura 4.4 – Mortes acumuladas de coronavirus no Distrito Federal do dia 28 de março de 2020 até o dia 15 de julho de 2021.

média móvel de X usamos a seguinte fórmula:

$$y_t = \frac{1}{N} \sum_{i=0}^{N-1} x_{t-i}, \tag{4.1}$$

onde N é a quantidade de pontos necessários para a média móvel e y_t é a média aritmética dos pontos $x_t, x_{t-1}, x_{t-2}, \ldots, x_{t-(N-1)}$. No caso da média móvel de $N = 14$ dias temos:

$$y_t = \frac{1}{14} \sum_{i=0}^{13} x_{t-i}, \tag{4.2}$$

lembrando que aqui estamos contando a partir do zero.

As séries temporais de casos e mortes cumulativos (figuras 4.3 e 4.4) são obtidas somando todas as ocorrências até o dia da leitura. De forma mais precisa, suponha uma série temporal $X = (x_1, x_2, x_3, \ldots, x_T)$, podendo representar casos ou mortes diários. Para construir a série temporal de casos cumulativos, por exemplo, usamos a seguinte fórmula:

$$y_t = \sum_{i=0}^{t} x_i, \tag{4.3}$$

onde x_t é a quantidade de novos casos no dia t e y_t é a soma de todos os casos até o dia t. Obtidas as quatro séries temporais, devemos padronizar os seus dados. A padronização é feita com o objetivo de mitigar um possível viés nos pesos das redes neurais causado por

dados com valores muitos distantes (valores que podem estar muito longe da média). Em todos os experimentos deste trabalho, vamos padronizar os dados na seguinte forma:

1. Escolha um intervalo $[a, b]$ no qual os dados da série temporal deverão estar.

2. Dada a série temporal X, calcule a quantidade:

$$X_{std} = \frac{X - X_{min}}{X_{max} - X_{min}}, \tag{4.4}$$

   onde X_{min} e X_{max} são o menor e maior valores da série temporal X, respectivamente. Lembrando que essa operação retorna uma nova série temporal com a mesma dimensão do vetor original X.

3. Utilizando o intervalo escolhido no item um, compute a nova série temporal padronizada de acordo com a seguinte operação:

$$X_{scaled} = X_{std}(b - a) + a, \tag{4.5}$$

   onde X_{scaled} é a série temporal padronizada.

O intervalo escolhido para fazer a padronização foi $[0, 1]$. Com os dados padronizados, a próxima etapa é transformar eles para tratar o problema de forma supervisionada, ou seja, durante o treinamento as redes neurais vão ter acesso ao valor real do *target*. Antes de tratar o problema de forma supervisionada, devemos dividir os dados em dois conjuntos: treinamento e teste. O conjunto de treinamento serve para treinar a rede neural, ajustando seus pesos da melhor forma possível. O conjunto de teste tem como função verificar se a rede neural aprendeu bem sobre os dados. É importante ressaltar que durante o treinamento a rede neural não tem acesso aos dados de teste. No caso deste experimento, vamos treinar a rede usando todos os dados anteriores ao mês de início. Como exemplo, para fazer a previsão do mês de julho de 2020, usamos todos os dias até 30 de junho como treinamento e o próprio mês de julho como teste. A previsão dos outros meses é feita do mesmo modo. Aqui vamos testar quatro arquiteturas diferentes: ESN e *deep* ESN com reservatório aleatório (figura 3.10) e ESN e *deep* ESN com reservatório pequeno mundo (figura 3.12). Até aonde sabemos ainda não foi usada uma *deep* ESN com reservatórios pequeno mundo para previsões em séries temporais de modo geral. Para buscar os melhores parâmetros durante o treinamento usamos o *particle swarm optimization* (PSO), um algoritmo feito para procurar mínimos em funções não lineares [40]. As tabelas 4.2 e 4.3 mostram o o intervalo escolhido para cada parâmetro das redes neurais.

Para avaliar os modelos vamos utilizar duas métricas, sendo elas o *root mean squared*

Hiperparâmetros ESN	
Hiperparâmetro	Valores
input scaling	$[0.1, 5]$
state size	$[50, 1000]$
connection probability	$[0.01, 0.1]$
k neighbors	$[2, 10]$
spectral radius	$[0.1, 1]$
noise	$[0, 0.1]$
alpha	$[0.1, 1]$
beta	$[0, 4]$
epochs	$[2, 10]$

Tabela 4.2 – Intervalo de pesquisa de cada hiperparâmetro para a ESN usando o PSO.

Hiperparâmetros *deep* ESN	
Hiperparâmetro	Valores
input scaling	$[0.1, 2]$
inter layer scaling	$[0.1, 2]$
spectral radius	$[0.96, 1]$
epochs	$[2, 10]$
connection probability	$[0.01, 0.1]$
k neighbors	$[2, 6]$
state size	$[50, 100]$
number of layers	$[3, 10]$
beta	$[10^{-10}, 10^{-4}]$

Tabela 4.3 – Intervalo de pesquisa de cada hiperparâmetro para a *deep* ESN usando o PSO.

error (a raiz quadrada de 3.21) e o erro relativo médio (MRE) que é dado por:

$$MRE = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{y_i}, \tag{4.6}$$

onde y_i é o valor observado (*label*) e $\hat{y}_i$ é o valor previsto pelo modelo. É importante observar que essa métrica gera problemas quando o valor observado é igual a zero.

4.2.2 Previsões do $R(t)$

O segundo experimento consiste em fazer previsões no número de reprodução efetivo mensalmente, como descrito no experimento anterior. Aqui vamos utilizar a melhor rede neural encontrada no experimento um e usar suas previsões para calcular R_t no futuro, mais precisamente os primeiros 27 dias de cada mês, indo de julho de 2020 até junho de 2021. Observando a figura 4.5, juntamos a série temporal de dados reais com a série temporal da previsão da ESN. Por exemplo, casos queremos prever o comportamento do R_t no mês de julho de 2020, o bloco azul seria tudo anterior ao mês de julho e o bloco vermelho (ESN)

seria a previsão do mês de julho feita pela melhor rede neural do experimento um. A série temporal então entra no modelo *time-since-infection* e finalmente obtemos o número de reprodução. Para a distribuição w da equação 2.19, usamos uma distribuição de Poisson com média de quatro dias, como mostrado em [41].

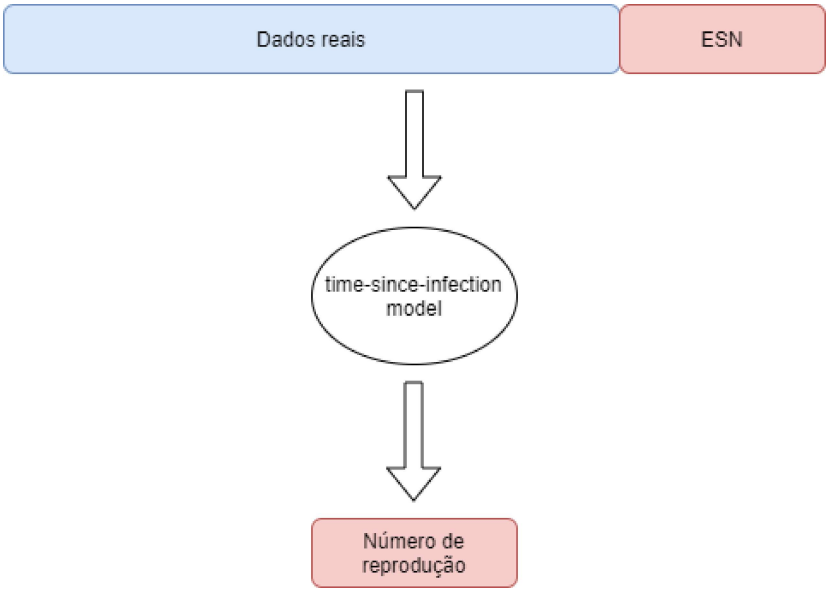

Figura 4.5 – Processo para previsão do número de reprodução.

4.2.3 Previsões dez dias no futuro em cada ponto

O terceiro experimento é focado em verificar a capacidade de cada uma das quatro arquiteturas na previsão da série temporal de casos. Neste experimento vamos filtrar a série temporal de casos usando transformada rápida de Fourier, selecionando os modos mais importantes e em seguida padronizar os dados usando o mesmo procedimento apresentado no experimento um. A série temporal filtrada (ainda sem padronização) é mostrada na figura 4.6. Para fazer as previsões, vamos começar com 90 dias de treinamento e os próximos 10 de teste. No próximo passo vamos usar 91 dias de treinamento e os próximos 10 de teste. O método iterativo é executado até percorrer toda a série temporal. Nesse experimento não ocorre pesquisa de hiperparâmetros pois queremos apenas saber aonde os modelos estão fazendo boas previsões e aonde não. Para avaliar a performance vamos usar o RMSE, já discutido no experimento um.

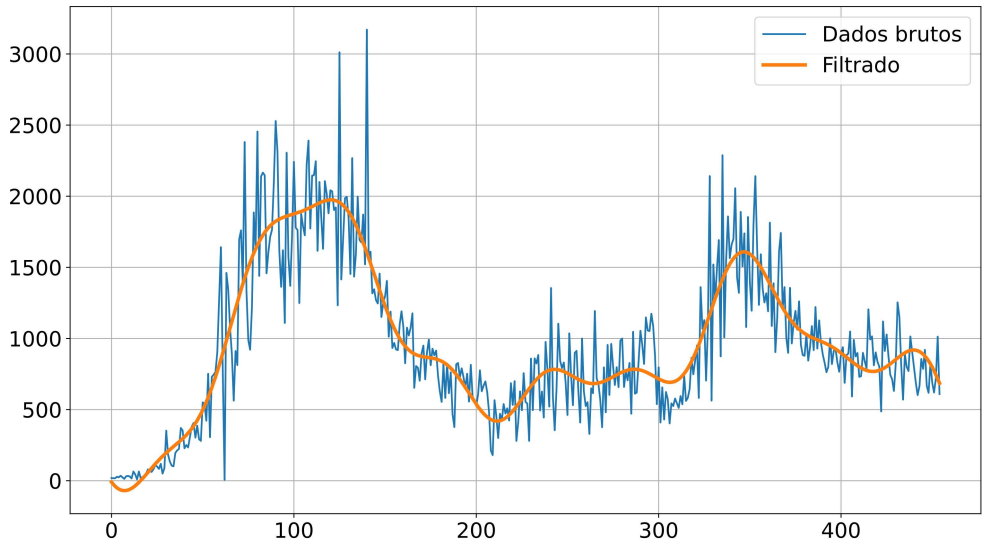

Figura 4.6 – Casos diários coronavirus no Distrito Federal do dia 28 de março de 2020 até o dia 15 de julho de 2021. Em laranja os dados filtrados usando transformada de Fourier.

5 RESULTADOS

No total foram realizados quatro experimentos usando os dados do coronavirus no Distrito Federal: previsões mensais nas quatro séries temporais, previsão da segunda onda, previsões de dez dias a partir de cada dia e previsões do número de reprodução efetivo.

5.1 PREVISÕES MENSAIS USANDO MÉDIA MÓVEL

No primeiro experimento, fizemos previsões nas quatro séries temporais, sendo elas: a média móvel de casos em 14 dias, a média móvel de mortes em 14 dias, casos cumulativos e mortes cumulativas. Fizemos as previsões de mês a mês, começando em julho de 2020 e terminando em junho de 2021. Usamos sempre os primeiros 27 dias de cada mês para teste e as duas semanas anteriores para validação. O algoritmo usado para realizar a pesquisa de hiper parâmetros das redes neurais foi o *particle swarm optimization*. Aqui usamos quatro redes neurais diferentes: a ESN com o reservatório aleatório, a ESN com o reservatório pequeno mundo, a *deep* ESN com reservatórios aleatórios e a *deep* ESN com reservatórios pequeno mundo.

5.1.1 Média móvel de casos (14 dias)

A figura B.1 mostra as previsões dos doze meses usando as quatro redes neurais. Podemos perceber que ao longo que os meses passam as previsões de cada rede neural vão melhorando. A explicação lógica para isso é a maior quantidade de dados disponíveis a cada mês. As previsões dos meses de setembro de 2020 e setembro de 2021 não foram tão boas pois apresentaram padrões novos para as redes neurais. Em março de 2021 a única rede neural que teve um bom desempenho foi a ESN com reservatório pequeno mundo (azul). Observando as figuras 5.1 e 5.2 vemos que as métricas RMSE (*root mean square error*) e MRE (*mean relative error*) possuem uma leve tendência de queda com o passar dos meses, o que fortalece as melhores previsões nos últimos meses. É válido observar também que na média a melhor rede neural foi a *deep* ESN com os reservatórios pequeno mundo, mas sua performance ficou bem próxima da ESN com reservatório aleatório e da *deep* ESN com reservatórios aleatórios. A figura 5.3 mostra o MRE cumulativo de cada rede neural ao longo dos 27 dias. A média é feita sobre todos os meses. Como era de se esperar a melhor rede neural é a *deep* ESN com reservatório pequeno mundo. Apesar da ESN com reservatório pequeno mundo apresentar valores pequenos para o MRE nos primeiros dez dias, seu erro cresce mais que todas as outras redes neurais com o passar do tempo. Tomando como aceitável um erro cumulativo por volta de 10%, todas as quatro redes neurais conseguem fazer

boas previsões de até dez dias na média móvel de casos. Uma última observação é que o comportamento do erro cumulativo tem caráter linear.

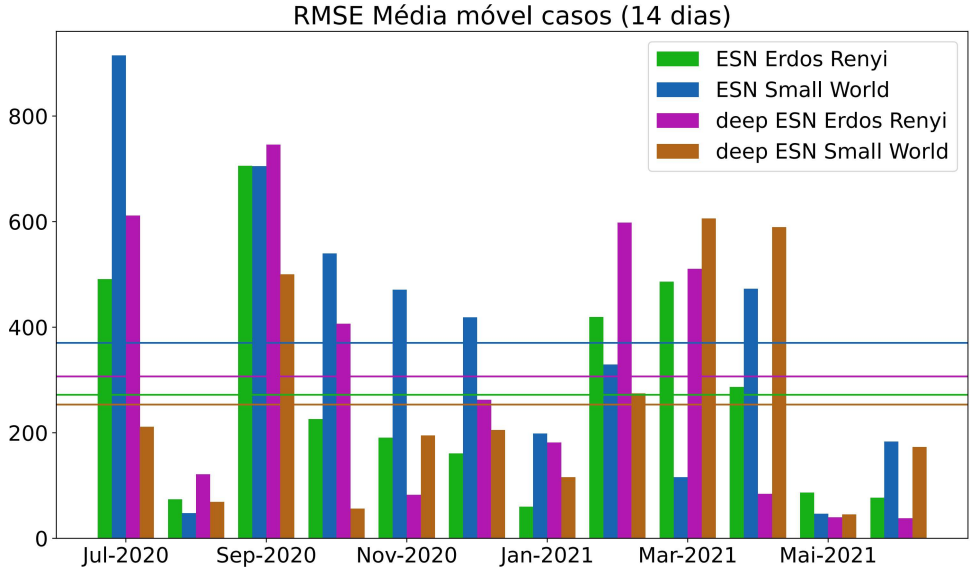

Figura 5.1 – Root mean squared error (RMSE) para a média móvel de casos em cada mês. As linhas horizontais são as médias do erro de cada rede neural sobre todos os meses considerados.

5.1.2 Média móvel de mortes (14 dias)

A figura B.2 mostra as previsões mensais da média móvel de casos no Distrito Federal. Podemos observar que as melhores previsões são da *deep* ESN com reservatórios pequeno mundo. Os erros das previsões das quatro redes neurais foram menores em relação à média móvel de casos. A explicação mais lógica para isso é a maior qualidade de dados relacionado a óbitos, ou seja, existe um controle maior com as informações sobre fatalidades em relação aos casos e isso faz com que as redes neurais possam trabalhar melhor. Novamente as redes tiveram dificuldade de prever a segunda onda (março de 2021). Observando a figura 5.4, apesar de algumas vezes não pegarem a tendência da série temporal, as redes neurais conseguiram manter um valor baixo para o RMSE em quase todos os meses. O único caso onde houve grande discrepância entre os dados e as previsões foi no mês de abril. Nesse mês o Distrito Federal estava na segunda onda do coronavirus. Vale ressaltar que as redes também não tiveram bom desempenho no mesmo período para a média móvel de casos. Ainda na mesma figura concluímos que a melhor rede neural foi a *deep* ESN com reservatórios pequeno mundo. A pior rede neural foi a ESN com reservatório aleatório. Observando a figura 5.5 podemos ver o desempenho de cada rede neural de forma mais intuitiva com o erro relativo médio. Na maioria dos meses as redes neurais tiveram erro por volta dos 10%. Todas

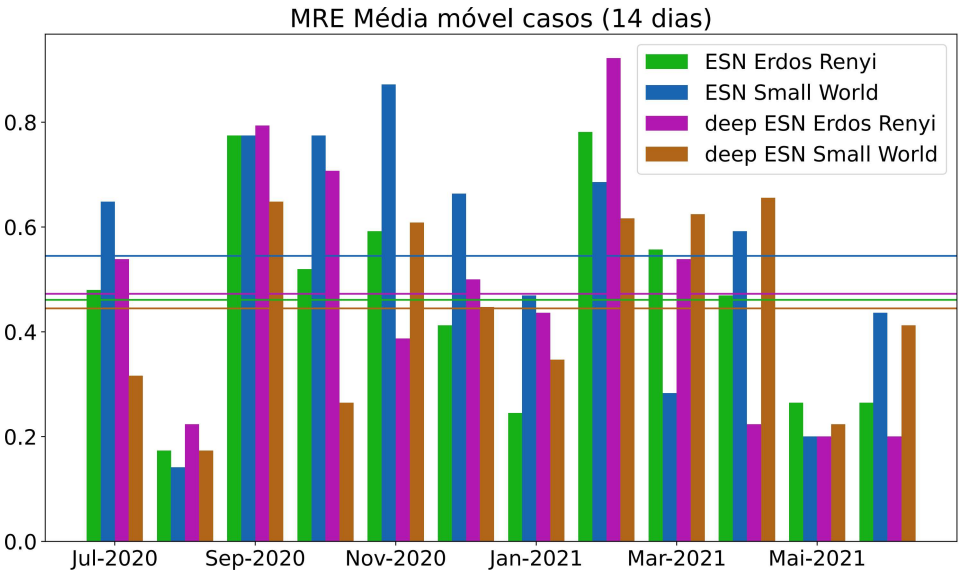

Figura 5.2 – Mean relative error (MRE) para a média móvel de casos em cada mês. As linhas horizontais são as médias do erro de cada rede neural sobre todos os meses considerados.

as redes neurais, com exceção da ESN com reservatório aleatório, tiveram o RMSE abaixo de 10.O resultado mais interessante vem da figura 5.6 que mostra o erro relativo acumulado médio sobre os meses de cada rede. O ponto mais importante a se observar é que as duas redes neurais com reservatório pequeno mundo tiveram melhor desempenho. Mesmo que as redes com reservatórios aleatórios apresentem um erro relativo pequeno para poucos dias, esse erro aumenta com uma taxa maior em relação às outras redes. Isso é interessante pois serve como um indício de que o reservatório pequeno mundo pode possuir mais poder de memória.

5.1.3 Casos cumulativos

A figura B.3 mostra as previsões feitas nos casos cumulativos do coronavirus no Distrito Federal. Podemos ver que, de modo geral, todas as redes neurais foram bem nas previsões. O único mês em que nenhuma rede conseguiu capturar o comportamento da doença foi em março de 2021, período em que o Distrito Federal se encontrava na segunda onda. Nos primeiros dois meses a *deep* ESN com reservatórios pequeno mundo superestimou o número total de casos, enquanto que nos meses de novembro de 2020 e fevereiro de 2021, as redes ESN com reservatório pequeno mundo e ESN com reservatório aleatório subestimaram o número total de casos, respectivamente. Observando a figura 5.7, a rede neural que apresentou o menor RMSE médio foi a ESN com reservatório aleatório, enquanto que a rede que apresentou o maior RMSE foi a *deep* ESN com reservatórios pequeno mundo. Novamente no período da segunda onda tivemos uma leve alta nos erros devido ao novo padrão apresen-

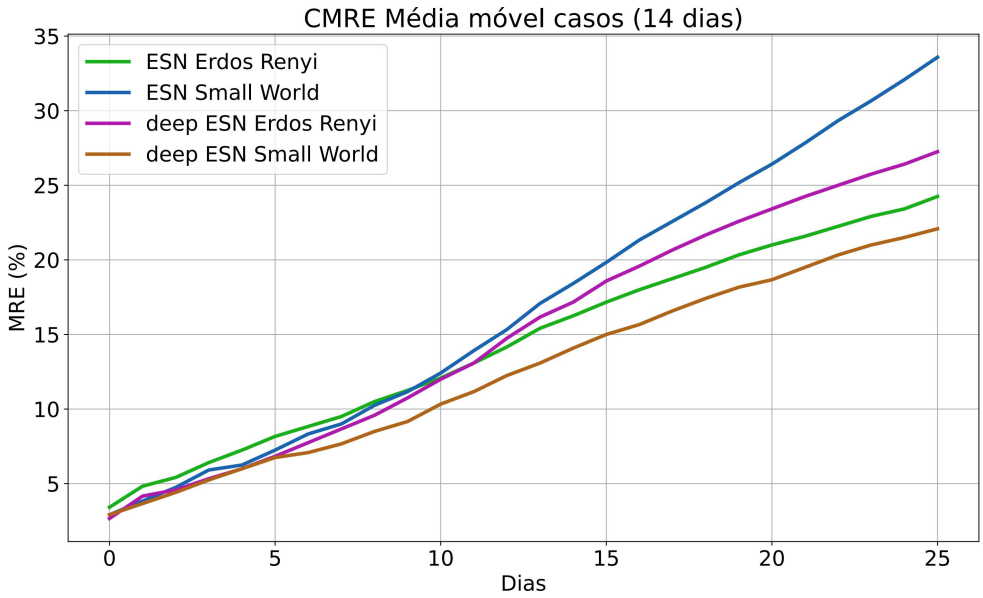

Figura 5.3 – MRE cumulativo médio para a média móvel de casos.

tado pela série temporal. A figura 5.8 mostra que em termos relativos todas as redes neurais fizeram um bom trabalho pois mantiveram o erro relativo médio sobre os meses abaixo dos 20%, com destaque para a ESN com reservatórios aleatórios que teve seu erro próximo dos 10%. A figura 5.9 mostra o erro médio cumulativo de cada rede. Podemos ver que a *deep* ESN com reservatórios pequeno mundo apresenta um aumento no erro considerável a partir do décimo dia, enquanto que todas as outras conseguem mantar a progressão do erro em uma baixa taxa de variação. Nos primeiros dez dias a melhor rede é a *deep* ESN com reservatórios aleatórios, mas logo após a ESN com reservatório aleatório toma o seu lugar. Considerando que 10% de erro relativo é razoável, todas as redes neurais aqui testadas fizeram boas previsões. As previsões feitas nesta seção são muito importantes porque podemos saber de forma precisa quais serão as proporções da doença nos próximos trinta dias.

5.1.4 Mortes cumulativas

A figura B.4 mostra as previsões mensais das quatro redes neurais sobre a série temporal de mortes cumulativas. Podemos ver que no mês de março de 2021 nenhuma das redes neurais conseguiu capturar a tendência, mas em todos os outros meses pelo menos um conseguiu acompanhar a progressão de mortes cumulativas. Percebemos também que as redes neurais de várias camadas subestimaram várias vezes o número cumulativo de mortes (maio de 2021, por exemplo). A figura 5.10 mostra o RMSE de cada rede neural em cada mês. Novamente todas as redes apresentaram dificuldade em prever a segunda onda. A melhor rede nesse caso foi a ESN com reservatório pequeno mundo e a pior foi a *deep* ESN com

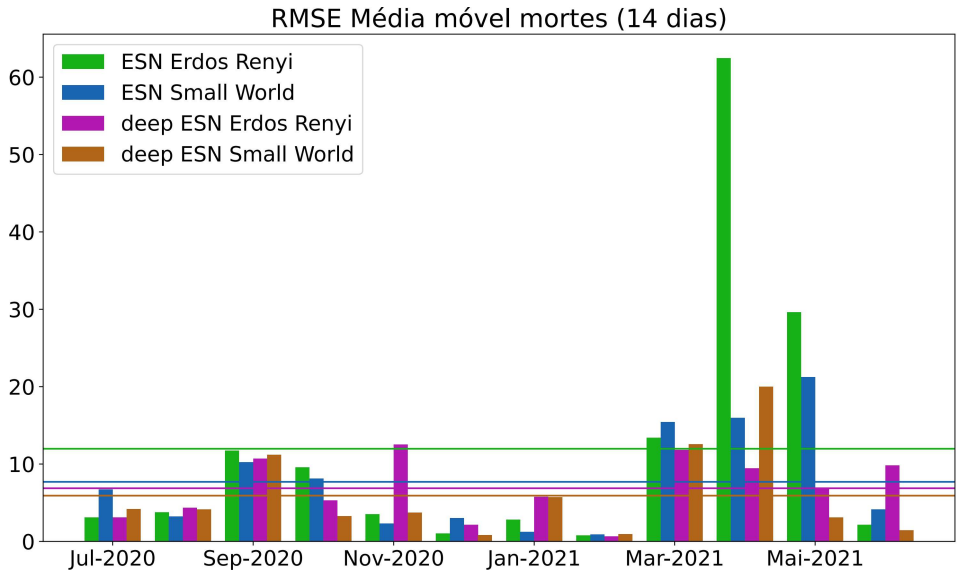

Figura 5.4 – RMSE para a média móvel de mortes em cada mês.

reservatórios aleatórios. Observando agora a figura 5.11 temos duas situação que devem ser comentadas. Nos primeiros meses o MRE começa em alta com uma leve tendência de queda até o mês de fevereiro. Isso pode ser explicado pela quantidade crescente de dados que vai ficando disponível para treinamento. A partir do mês de março de 2021 observamos uma nova subida no erro por conta da segunda onda. Sabemos que as redes demoram um pouco para perceber novos padrões, ainda mais nessa situação onde temos uma quantidade bem limitada de dados para treinamento. Comparando com os resultados das previsões sobre casos cumulativos o erro relativo na média foi maior. A explicação para isso é talvez a forma mais acentuada da segunda onda na série temporal de mortes cumulativas.

Como último resultado deste experimento, a figura 5.12 mostra o erro relativo médio cumulativo sobre os meses para cada rede neural. Nos primeiros cinco dias todas as redes apresentam uma progressão de erro semelhante mas logo após esse ponto os erros das redes que possuem camadas internas começam a aumentar a uma taxa maior que os erros das redes de apenas uma camada. Observando a curva da ESN pequeno mundo, mesmo em previsões de 25 dias seu MRE permanece abaixo dos 3%. Considerando um erro relativo de 10% como razoável para esse tipo de previsão, todas as quatro redes neurais têm condições de fazer boas previsões.

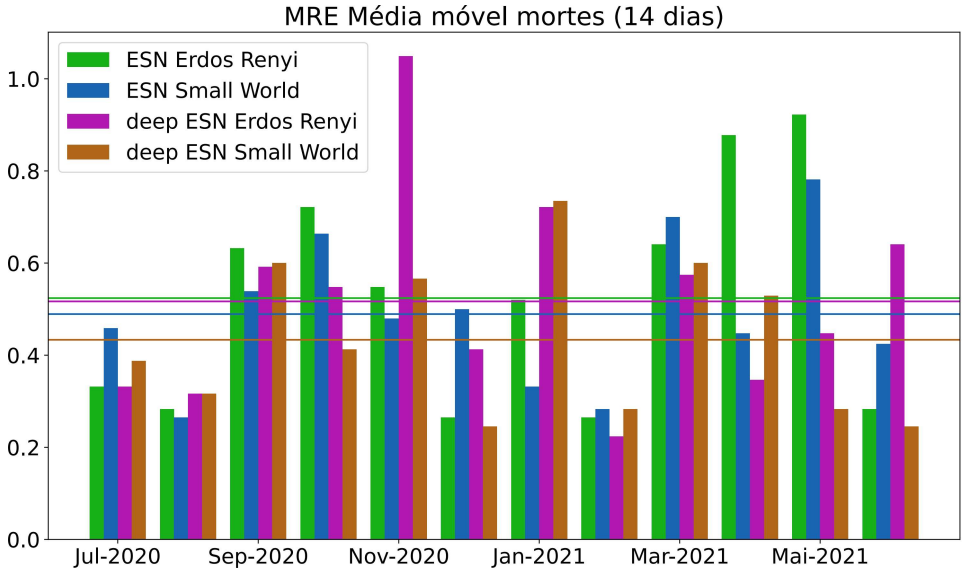

Figura 5.5 – MRE para a média móvel de mortes em cada mês.

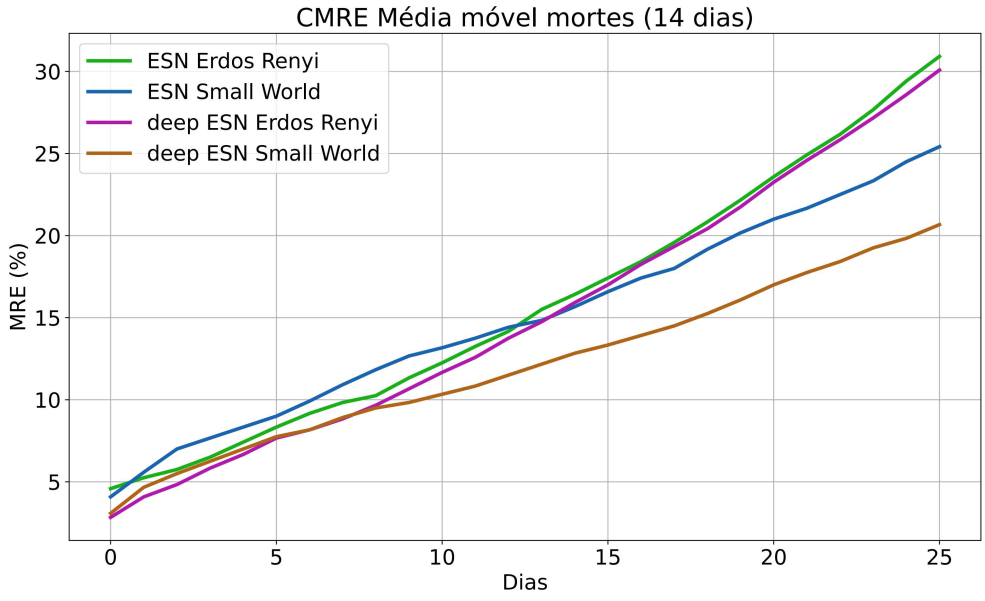

Figura 5.6 – MRE cumulativo médio para a média móvel de mortes.

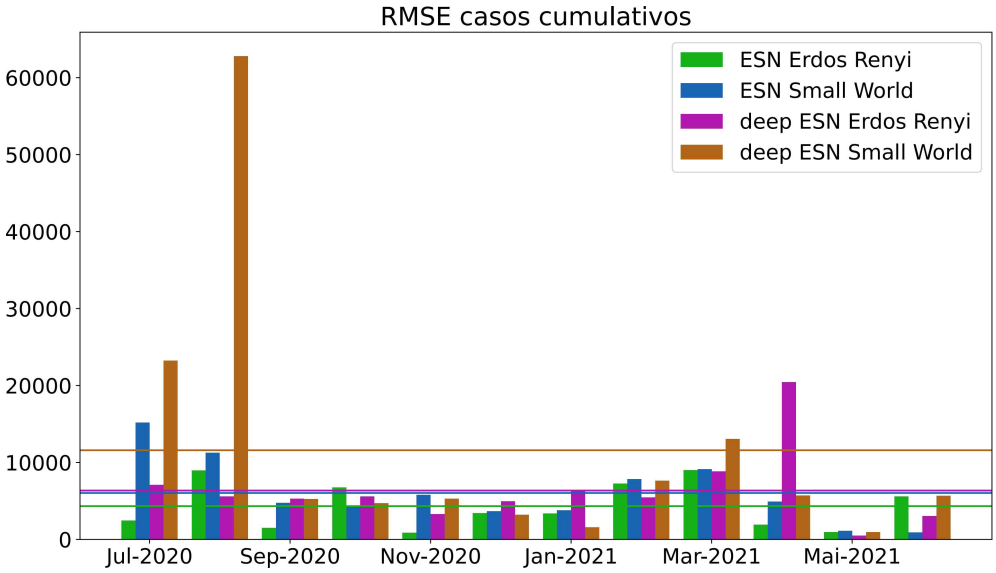

Figura 5.7 – RMSE para os casos cumulativos.

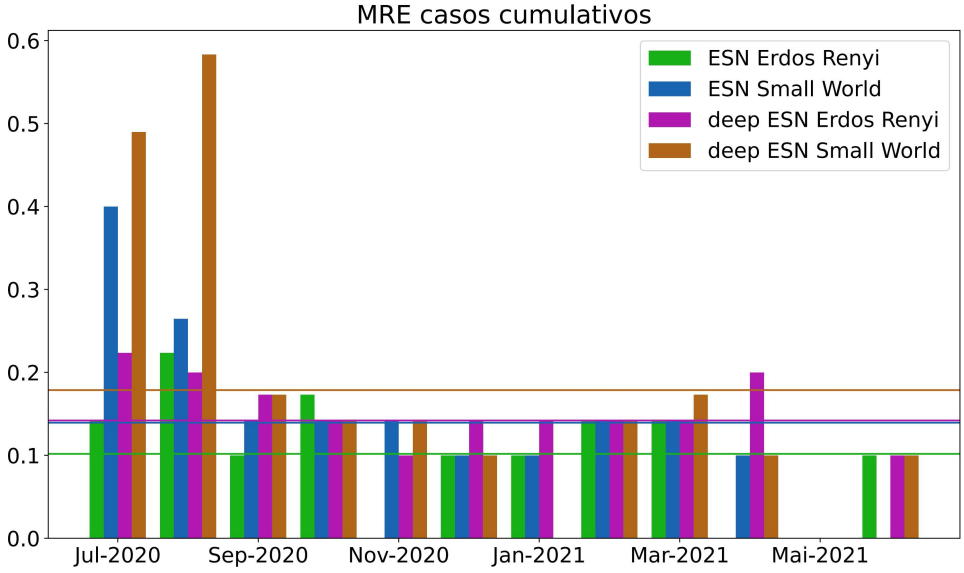

Figura 5.8 – MRE para os casos cumulativos.

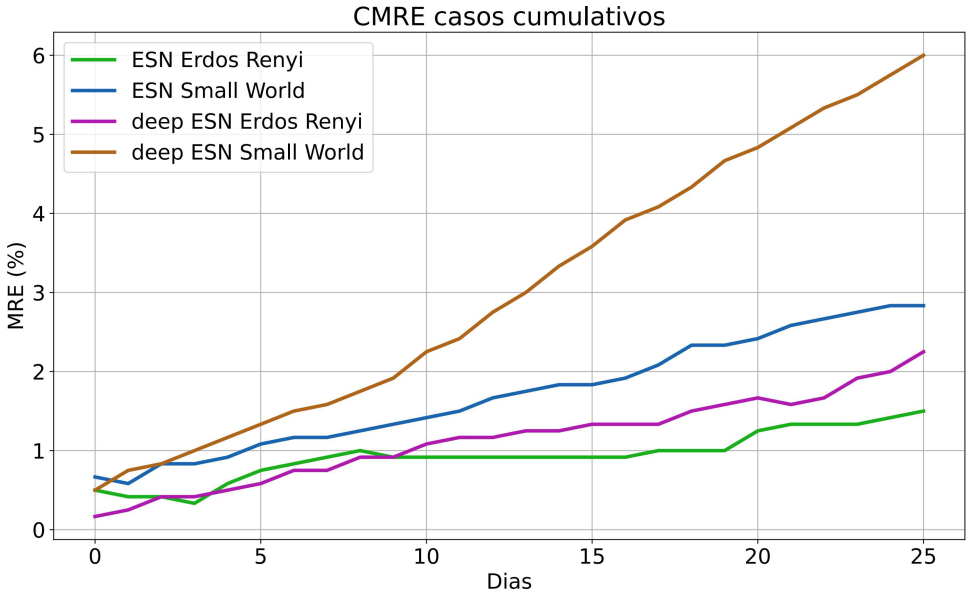

Figura 5.9 – MRE cumulativo para casos cumulativos.

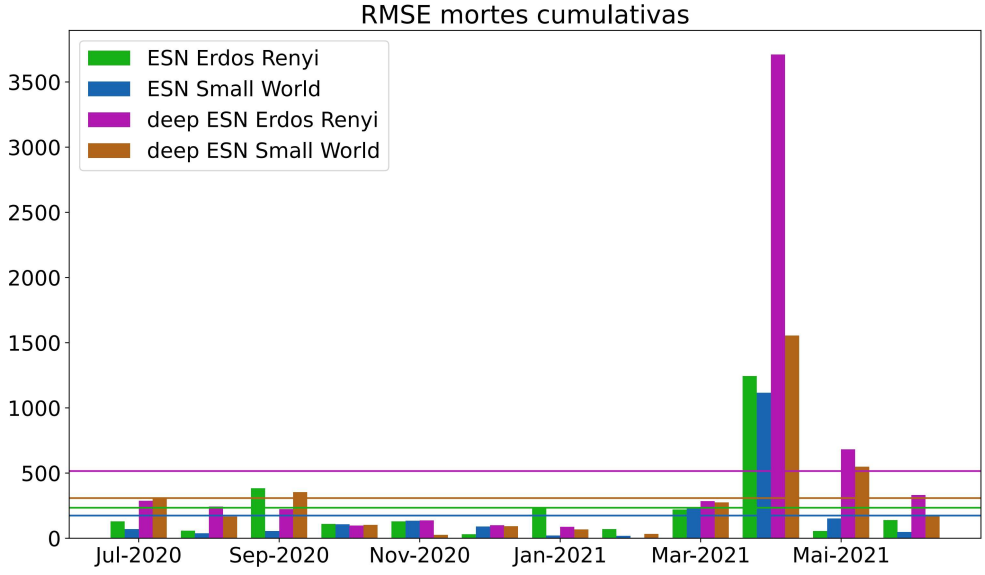

Figura 5.10 – RMSE para mortes cumulativas.

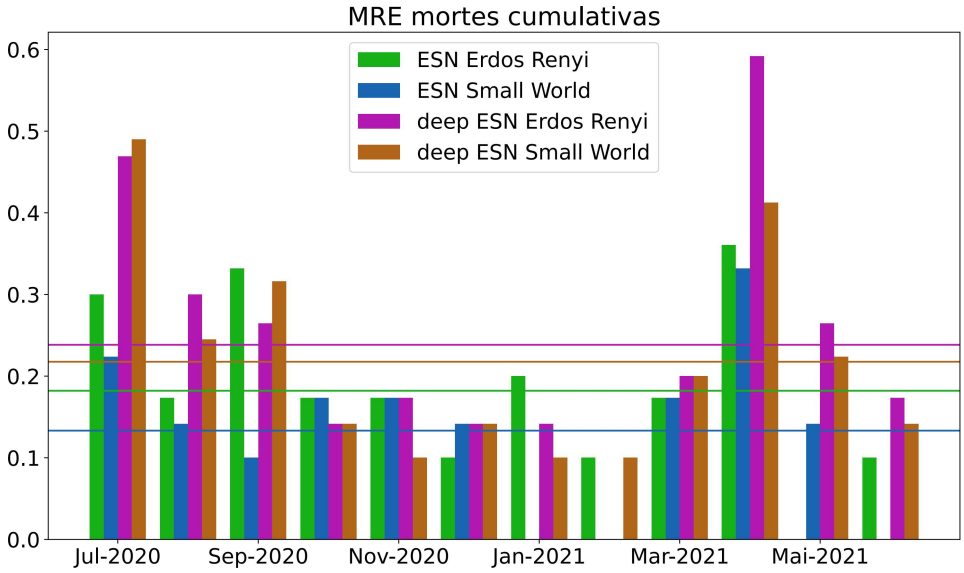

Figura 5.11 – MRE para mortes cumulativas.

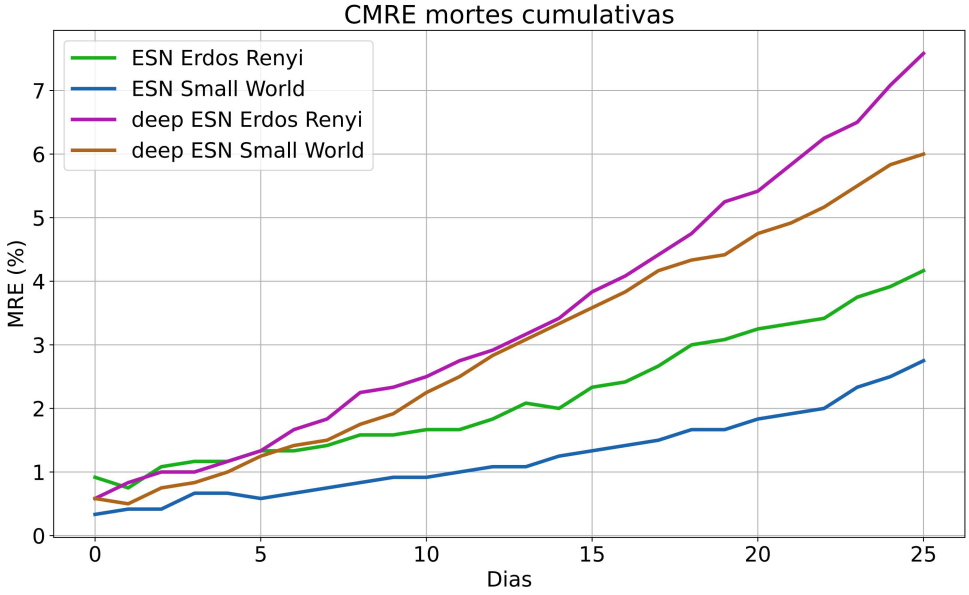

Figura 5.12 – MRE cumulativo para mortes cumulativas.

5.2 PREVISÃO DO NÚMERO DE REPRODUÇÃO EFETIVO

A figura B.5 mostra as previsões mensais do número de reprodução instantâneo usando a *deep* ESN com reservatórios pequeno mundo, a melhor arquitetura do primeiro experimento. O primeiro ponto a observar são as previsões dos meses de março e abril de 2021, onde o desempenho da rede neural não foi muito boa por causa da mudança de comportamento devido a segunda onda. Com exceção do mês de março de 2021, todas as previsões ficaram razoáveis por pelo menos cinco dias. As melhores previsões são dos meses de agosto de 2020, outubro de 2020 e maio de 2021. Considerando agora o limiar do número de reprodução instantâneo (linha horizontal preta em cada gráfico), os valores observados cruzaram esse limiar oito vezes, enquanto que os valores previstos pelo modelo cruzaram o limiar cinco vezes nesses mesmos meses. Isso representa um *recall* de 0.625 se o evento relevante fosse prever se o número de reprodução instantâneo cruza o limiar. Isso significa que aproximadamente 62% das vezes que o número de reprodução instantâneo de fato cruzar o limiar, a rede neural conseguirá prever, independentemente da previsão ser boa ou não. É claro que esse não é o foco deste trabalho, mas ter uma boa estimativa se essa quantidade irá ficar maior ou menor que 1 é de extrema importância para tomadas de decisão.

5.3 PREVISÕES DE DEZ DIAS NO FUTURO EM CADA PONTO

As figuras 5.13, 5.14, 5.15 e 5.16 mostram os resultados das previsões de dez dias no futuro. O eixo vertical representa a transformada de Fourier da série temporal original e o eixo horizontal representa o tempo em dias. O primeiro ponto a observar é o comportamento das previsões na segunda onda (segundo máximo de cada figura). Percebemos que as redes de uma camada fazem previsões piores do que as redes de camadas internas. Isso pode ser explicado pela maior capacidade de memória da *deep* ESN, pois suas camadas internas contribuem para aumentar o poder de processamento da rede neural. Comparando as ESN com uma camada, podemos ver que na fase decrescente da segunda onda, a rede com reservatório aleatório, tem mais dificuldade para aprender a tendência de descida.

Todos esses resultados podem ser conferidos na figura 5.17, onde são plotadas as médias móveis do RMSE para as quatro redes neurais. Cada ponto no gráfico é o erro da previsão de dez dias usando N dias anteriores para treinamento (eixo x). Aqui utilizamos a média móvel para diminuir o ruído da série temporal original. No primeiro pico podemos ver que a ESN conseguiu melhor desempenho que a *deep* ESN. Uma boa explicação para isso é o fato da *deep* ESN precisar de mais tempo para corrigir seus parâmetros, que geralmente são mais numerosos que os da ESN. Entre o primeiro pico e o início da segunda onda os erros aparentam possuir um comportamento estacionário, mas no início da segunda onda, os erros

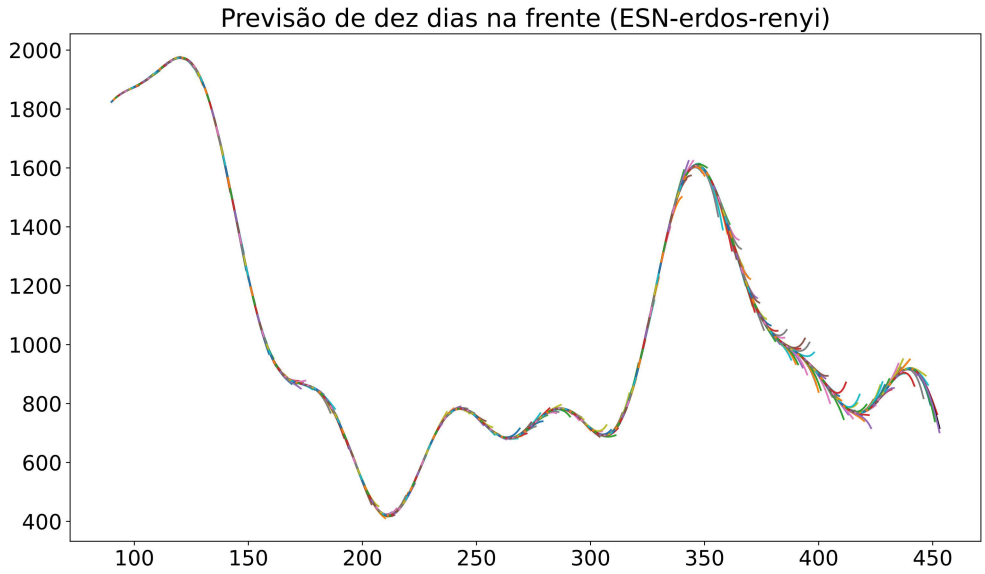

Figura 5.13 – Previsões de dez dias no futuro usando a ESN com reservatório aleatório. Cada linha colorida no gráfico é uma previsão de dez dias.

da ENS crescem enquanto que os erros da *deep* ESN continuam diminuindo. A segunda onda representa uma grande mudança de padrões dentro da série temporal de casos diários, fazendo com que as ESN de uma camada tenham um fraco desempenho nessa região.

Voltando novamente com a discussão sobre o poder de abstração de uma *deep* ESN, ela consegue levar os dados de entrada para um espaço l vezes maior que uma ESN, considerando os reservatórios com a mesma quantidade de neurônios. Talvez essa vantagem explique a diferença entre os erros na segunda onda.

Considerando agora as topologias, tanto no caso da ESN quanto da *deep* ESN, o reservatório pequeno mundo teve uma pequena vantagem. Após o início da segunda onda podemos observar que o erro da ESN com reservatório pequeno mundo começa a diminuir antes da ESN com reservatório aleatório. Isso é um indício que um reservatório mais trabalhado como o de Watts- Strogatz pode acelerar o aprendizado da ESN. Esse fenômeno é também observado no primeiro pico para a *deep* ESN, onde a que possui os reservatórios pequeno mundo apresenta erros menores. Um último ponto bastante interessante a ressaltar é as médias no gráfico. O distanciamento entre as médias da ESN e as médias da *deep* ESN mostra a vantagem do aprendizado profundo.

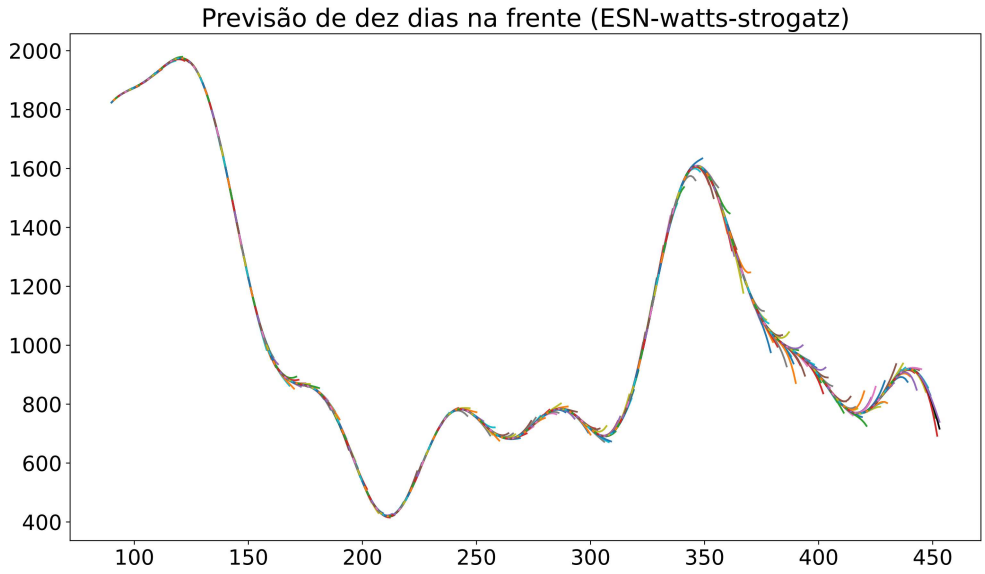

Figura 5.14 – Previsões de dez dias no futuro usando a ESN com reservatório pequeno mundo. Cada linha colorida no gráfico é uma previsão de dez dias.

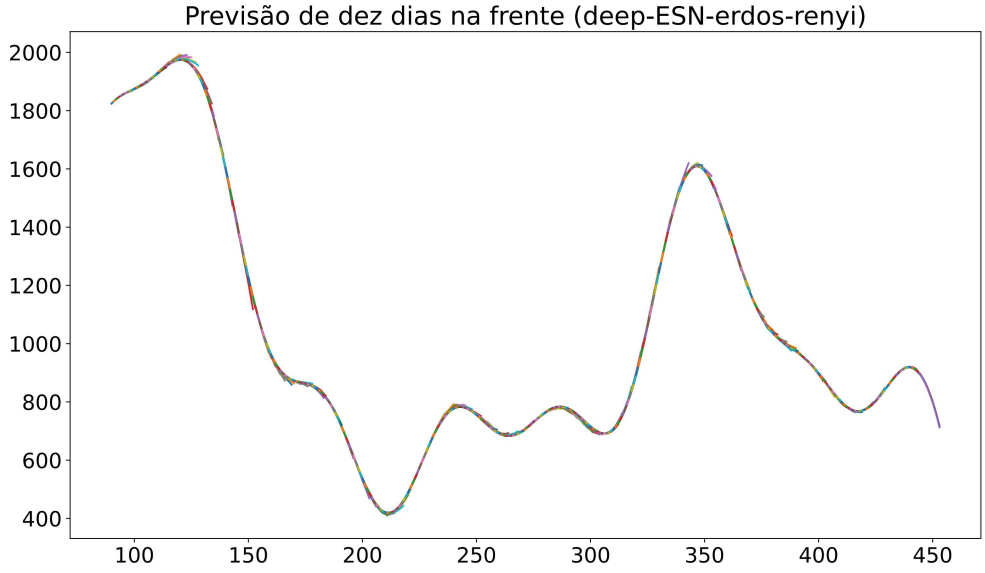

Figura 5.15 – Previsões de dez dias no futuro usando a *deep* ESN com reservatório aleatório. Cada linha colorida no gráfico é uma previsão de dez dias.

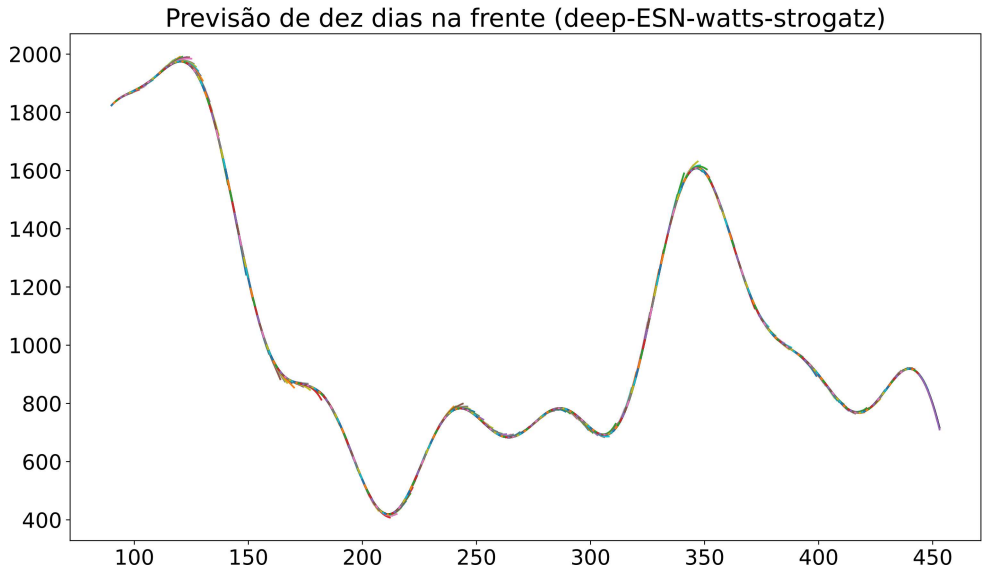

Figura 5.16 – Previsões de dez dias no futuro usando a *deep* ESN com reservatório pequeno mundo. Cada linha colorida no gráfico é uma previsão de dez dias.

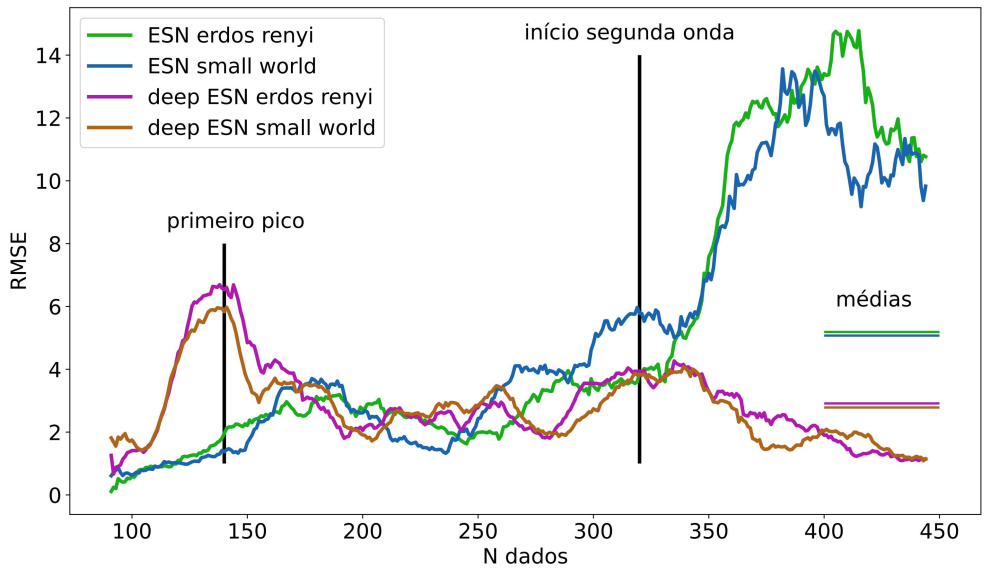

Figura 5.17 – Média móvel de sete pontos do RMSE para os quatro casos na série temporal de casos diários. O eixo x representa a quantidade de pontos usados para treinamento.

6 CONCLUSÃO E CONSIDERAÇÕES FINAIS

6.1 CONCLUSÃO

Neste trabalho usamos quatro variações da *Echo State Network* para fazer experimentos nas séries temporais do coronavirus no Distrito Federal. O trabalho foi dividido em três experimentos sendo o primeiro as previsões mensais nas quatro séries temporais, o segundo a previsão do número de reprodução instantâneo usando os resultados do primeiro experimento e o terceiro as previsões de dez dias no futuro em cada ponto da série temporal de casos.

No primeiro experimento, o resultado mais intuitivo e também mais importante é a curva de erros relativos cumulativos médios para cada série temporal. No caso da média móvel de casos todas as redes neurais conseguiram fazer previsões de dez dias no futuro com erro relativo abaixo dos 15%, com destaque para a *deep* ESN com reservatório pequeno mundo. No caso da média móvel de mortes, novamente todas as redes neurais conseguiram realizar previsões de dez dias no futuro com erros relativos abaixo dos 15%. Novamente o destaque vai para a *deep* ESN com reservatório pequeno mundo, que no nosso conhecimento, este é o primeiro trabalho de sua aplicação em previsões de séries temporais e também no contexto de doenças contagiosas. Nas previsões de casos e mortes cumulativos a ESN teve melhor desempenho em relação a *deep* ESN. Em casos cumulativos o grafo aleatório foi o melhor e em mortes cumulativas o grafo pequeno mundo foi o melhor. Ignorando por um momento a quantidade de camadas internas da rede, dos quatro casos analisados, em três deles o melhor grafo foi o pequeno mundo. Isso é mais um indício que essa topologia pode sim beneficiar a ESN, como mostrado em [35].

No segundo experimento selecionamos a melhor rede neural do experimento anterior, a *deep* ESN com reservatório pequeno mundo. Esse é talvez o experimento mais importante pois nele fazemos as previsões do número de reprodução instantâneo. Na média a rede neural conseguiu fazer boas previsões de pelo menos cinco dias no futuro. Nos melhores casos essa marca passa dos quinze dias, margem muito boa para tomadas de decisão por parte dos agentes públicos.

O terceiro experimento foi focado em ver a performance das redes neurais em cada ponto da epidemia. Aqui vemos novamente a superioridade da aprendizagem profunda. A *deep* ESN teve um desempenho muito melhor que a ESN na segunda onda, como mostra a figura 5.17. Essa vantagem da *deep* ESN sobre a ESN na previsão de séries temporais também foi observada em [37] onde a *deep* ESN obtém 3.07% a mais de acurácia em dados de teste

sobre a ESN na classificação de séries temporais com o objetivo de identificar pessoas com doença de Parkinson.

Computação de reservatório vem conquistando bastante espaço no meio acadêmico e industrial por vários fatores, sendo algum deles sua facilidade de entendimento e implementação (comparado a uma LSTM, por exemplo) e tempo de treinamento [17]. Em aprendizagem profunda não é diferente. A adição de reservatórios internos faz com que a *deep* ESN seja mais poderosa em processar dados continuando mais leve e melhor que outras redes neurais [17].

Como mostrado em [7] toda a teoria de computação de reservatório gira em torno da construção de um bom reservatório, que é a matriz de conexão aleatória dos neurônios internos. Isso é muito importante pois essas conexões permanecem constantes durante todo o treinamento e teste. Este trabalho mostrou que o uso do grafo pequeno mundo melhora tanto a ESN quanto a *deep* ESN na previsão de séries temporais. Trabalhos como [35] já mostraram a superioridade da ESN com o reservatório pequeno mundo, mas até onde sabemos este é o primeiro trabalho que faz um estudo comparativo entre as topologias dos grafos aleatório e pequeno mundo na aplicação da *deep* ESN.

O uso de redes neurais na previsão de doenças contagiosas, mais especificamente o coronavirus, mostra que essa ferramenta pode ser tão valiosa quanto os modelos epidemiológicos já bem conhecidos. Artigos como [15] [35] [42] mostram que as redes neurais podem também desempenhar um ótimo trabalho no campo de doenças contagiosas. Este trabalho não tem a intenção de encontrar uma metodologia que exclua o uso de modelos epidemiológicos. Muito pelo contrário, ele apoia a junção dessas técnicas, como faz na previsão do número de reprodução instantâneo, usando a teoria de redes neurais junto com o modelo *time-since-infection* [13].

6.2 TRABALHOS FUTUROS

Este trabalho abre espaço para algumas discussões. Talvez a mais importante aqui é sobre o uso de redes neurais como ferramenta na ajuda de tomada de decisão em situações de epidemia. Decisões sobre medidas restritivas seriam tomadas com muito mais inteligência se baseadas em previsões e não somente em informações acerca da situação atual. É claro que os modelos epidemiológicos desempenham um ótimo trabalho nesse tipo de situação, mas o aprendizado de máquina é uma forma que se apresenta mais prática, rápida e fácil (modelos podem ficar bastante complexos dependendo do número de parâmetros utilizados) para tentar saber o que vai acontecer no futuro de uma epidemia.

Outra discussão importante é sobre as topologias da ESN. Mais experimentos deve ser

feitos utilizando os grafos aleatório e pequeno mundo com o objetivo de quantificar a diferença de desempenho entre os dois grafos. Na parte de aprendizagem profunda é importante também aplicar as diferente topologias e observar a performance. Liebald em [36] mostra o desempenho da ESN usando três tipos de grafo diferentes e compara seus resultados. Seu trabalho é um ótimo ponto de partida para aplicação no caso de aprendizagem profunda.

O aprendizado de máquina quântico (*quantum machine learning*) é uma área que vem recebendo grande atenção de pesquisadores, particularmente dos físicos. A subárea que trata de reservatórios quânticos é chamada de *quantum reservoir computing*, onde são usados sistemas quânticos como reservatórios [43] [44] [45] [46]. Talvez uma continuação natural deste trabalho do ponto de vista de redes neurais seja explorar a *deep* ESN com sistemas quânticos como camadas internas na previsões de séries temporais.

REFERÊNCIAS BIBLIOGRÁFICAS

1 ATKESON, A. G. What will be the economic impact of covid-19 in the us? rough estimates of disease scenarios. *Staff Report*, v. 595, mar. 2020.

2 FILHO, A. S. N. et al. Nota técnica 19 de janeiro de 2021, situação da pandemia de covid-19 no brasil e impactos da campanha de vacinação. Jan 2021.

3 HETHCOTE, H. W. The methematics of infectious diseases. *SIAM review*, v. 42, p. 599–653, dez. 2000.

4 APONTAMENTOS sobre ecolocalização. Disponível em: <http://umdiadecampo.blogspot.com/2015/01/>.

5 THE biological neuron. Disponível em: <https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781787121393/5/ch05lvl1sec38/the-biological-neuron>.

6 BöRGERS, C. *An introduction to neuronal dynamics*. [S.l.]: Springer, 2007. ISBN 978-3-319-51170-2.

7 LUKOSEVICIUS, M.; JAEGER, H. Reservoir computing approaches to recurrent neural network training. *Computer science review*, 2010.

8 LUKOSEVICIUS, M. A practical guide to applying echo state networks. 2012.

9 GALLICCHIO, C.; MICHELI, A. *Deep ESN*. Disponível em: <https://media.springernature.com/original/springer-static/image/chp3A10.10072F978-3-030-20521-8_40/MediaObjects/484723_1_En_40_Fig1_HTML.png>.

10 VELAVAN, T. P.; MEYER, C. G. The covid-19 epidemic. *Tropical Medicine and International Health*, v. 25, p. 278–280, mar. 2020.

11 DHARMARATNE, S. et al. Estimation of the basic reproduction number (r0) for the novel coronavirus disease in sri lanka. *Virology Journal*, v. 17, 2020.

12 HSIANG, S. et al. The effect of large-scale anti-contagion policies on the covid-19 pandemic. *Nature*, May 2020.

13 FRASER, C. Estimating individual and household reproduction numbers in an emerging epidemic. *Plos One*, Aug 2007.

14 PETROVA, T.; SOSHNIKOV, D.; GRUNIN, A. Estimation of time-dependent reproduction number for global covid-19 outbreak. Jun 2020.

15 ARUNKUMAR, K. et al. Forecasting of covid-19 using deep layer recurrent neural networks (rnns) with gated recurrent units (grus) and long short-term memory (lstm) cells. *Chaos, solitons and fractals*, v. 146, 2021.

16 SHASTRI, S. et al. Time series forecasting of covid-19 using deep learning models: India-usa comparative case study. *Chaos, solitons and fractals*, v. 140, 2020.

17 KIM, T.; KING, B. R. Time series prediction using deep echo state networks. *Neural Computing and Applications*, Apr 2020.

18 OLUMOYIN, K.; KHALIQ, A.; FURATI, K. Data-driven deep learning algorithms for time-varying infection rates of covid-19 and mitigation measures. 2021.

19 CRAMER, S. C. et al. Harnessing neuroplasticity for clinical applications. *Brain a Journal of Neurology*, v. 134, p. 1591–1609, Oct 2010.

20 HAYKIN, S. *Neural networks and learning machines*. [S.l.]: Pearson, 2008. ISBN 978-0-13-147139-9.

21 COHEN, L. B. Changes in neuron structure during action potential propagation and synaptic transmission. *Physiological reviews*, v. 53, p. 373–409, Apr 1973.

22 HODGKIN, A. L.; HUXLEY, A. F. A quantitative description of membrane current and its application to conduction and excitation in nerve. *Bulletin of mathematical biology*, v. 52, p. 25–71, 1990.

23 NOORI, H. R. The impact of the glial spatial buffering on the k+ nernst potential. *Springer Science+Business Media*, Jul 2011.

24 SUYKENS, J.; VANDEWALLE, J. Training multilayer perceptron classifiers based on a modified support vector method. *IEEE Transactions on Neural Networks*, v. 10, n. 4, p. 907–911, 1999.

25 RUDER, S. *An overview of gradient descent optimization algorithms*. 2017.

26 MANDIC, D. A generalized normalized gradient descent algorithm. *IEEE Signal Processing Letters*, v. 11, n. 2, p. 115–118, 2004.

27 MøLLER, M. F. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, v. 6, n. 4, p. 525–533, 1993.

28 DENG, L.; HINTON, G.; KINGSBURY, B. New types of deep neural network learning for speech recognition and related applications: an overview. p. 8599–8603, 2013.

29 POUYANFAR, S. et al. A survey on deep learning: Algorithms, techniques, and applications. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 51, n. 5, set. 2018. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3234150>.

30 WANG, H. et al. Application of deep-learning algorithms to mstar data. p. 3743–3745, 2015.

31 THIMM, G.; FIESLER, E. High-order and multilayer perceptron initialization. *IEEE Transactions on Neural Networks*, v. 8, n. 2, p. 349–359, 1997.

32 BELLEC, G. et al. Biologically inspired alternatives to backpropagation through time for learning in recurrent neural nets. *CoRR*, abs/1901.09049, 2019. Disponível em: <http://arxiv.org/abs/1901.09049>.

33 MAASS, W.; NATSCHLäGER, T.; MARKRAM, H. Real-time computing without stable states: a new framework for neural computation based on perturbations. *Neural computation*, v. 14, n. 11, p. 2531–2560, 2002.

34 JAEGER, H. The echo state approach to analysing and training recurrent neural networks. *Technical report GMD*, v. 148, 2001.

35 LIU, B. et al. A prediction method based on improved echo state network for covid-19 nonlinear time series. *Journal of Computer and Communications*, v. 8, p. 113–122, 2020.

36 LIEBALD, B. Exploration of effects of different network topologies on the esn signal crosscorrelation matrix spectrum. 2004.

37 GALLICCHIO, C.; MICHELI, A.; PEDRELLI, L. Deep echo state networks for diagnosis of parkinson's disease. *ESANN*, 2018.

38 STROGATZ, D. J. W. ans S. H. Collective dynamics of small-world networks. *Nature*, v. 393, p. 440–442, 1998.

39 FEDERAL, G. *Coronavirus Brasil*. Disponível em: <https://covid.saude.gov.br>.

40 KENNEDY, J.; EBERHART, R. Particle swarm optimization. v. 4, p. 1942–1948 vol.4, 1995.

41 MEYEROWITZ, E. A. et al. Transmission of sars-cov-2: a review of viral, host, and environmental factors. 2021.

42 CAR, Z. et al. Modeling the spread of covid-19 infection using a multilayer perceptron. *Computational and mathematical methods in medicine*, v. 2020, 2020.

43 MARTíNEZ-PEñA, R. et al. Information processing capacity of spin-based quantum reservoir computing systems. *Cognitive computation*, 2020.

44 FUJII, K.; NAKAJIMA, K. Harnessing disordered-ensemble quantum dynamics for machine learning. *Physical Review Applied*, 2017.

45 LIU, J. et al. An echo state network architecture based on quantum logic gate and its optimization. *USIR*, 2020.

46 FUJII, K.; NAKAJIMA, K. Quantum reservoir computing: a reservoir approach toward quantum machine learning on non near-term quantum devices. 2020.

Apêndice

A CÓDIGOS EM PYTHON

A.1 ECHO STATE

Abaixo segue o código da Echo state em Python.

```
import numpy as np
import numpy.linalg as LA
import time
import helpers as hp
import networkx as nx



class echoState:
    """
    Echo state network using Ridge Regression

    Parameters:

    input_size: length of vector input
    output_size: length of the output
    state_size: state matrix (W) dimension
    alpha: leaking rate
    beta: regularization
    epochs: number of times you train te whole train data

    """

    def __init__(
        self,
        input_scaling,
        input_size,
        output_size,
        state_size,
        connection_probability,
        k_neighbors,
        spectral_radius,
        noise,
        alpha,
        beta,
        epochs,
        graph_model = 'erdos-renyi',
        verbose=False):
```

```
39
40          # PARAMETERS
41
42          self.input_scaling = input_scaling
43          self.input_size = input_size
44          self.output_size = output_size
45          self.state_size = state_size
46          self.connection_probability = connection_probability
47          self.k_neighbors = k_neighbors  # number og initial neighbors for
     the watts strogatz model
48          self.spectral_radius = spectral_radius
49          self.graph_model = graph_model
50          self.noise = noise
51          self.alpha = alpha
52          self.beta = beta
53          self.epochs = epochs
54          self.verbose = verbose
55          self.vector_state = np.zeros(self.state_size, dtype=float)
56
57          # MATRICES
58
59          self.W_in = np.zeros(
60              (self.state_size, 1 + self.input_size), dtype=float)
61          self.W = np.zeros((state_size, state_size), dtype=float)
62          self.W_out = np.zeros(
63              (output_size, 1 + input_size + state_size), dtype=float)
64
65
66      # INITIATE MATRICES
67
68      def generatewin(self):
69          self.W_in = np.random.uniform(- self.input_scaling, self.
    input_scaling, (self.state_size, 1 + self.input_size))
70          if self.verbose:
71              print('Matrix W_in generated. Shape: ', self.W_in.shape, '.')
72          pass
73
74      def generatew(self):
75          if self.graph_model == 'erdos-renyi':
76              G = nx.fast_gnp_random_graph(n = self.state_size, p = self.
    connection_probability)
77              A = nx.adjacency_matrix(G).todense()
78              self.W = np.squeeze(np.asarray(A))
79              self.W = np.multiply(self.W, np.random.uniform(-0.5, 0.5, (
    self.state_size, self.state_size)))
80              w, _ = LA.eig(self.W)
81              w = np.absolute(w)
```

```
            self.W = self.W * (self.spectral_radius / np.amax(w))
            if self.verbose:
                print('Reservoir (W) created. Shape: ', self.W.shape, '.')
            pass

        elif self.graph_model == 'watts-strogatz':
            G = nx.watts_strogatz_graph(n = self.state_size, k = self.k_neighbors, p = self.connection_probability)
            A = nx.adjacency_matrix(G).todense()
            self.W = np.squeeze(np.asarray(A))
            self.W = np.multiply(self.W, np.random.uniform(-0.5, 0.5, (self.state_size, self.state_size)))
            w, _ = LA.eig(self.W)
            w = np.absolute(w)
            self.W = self.W * (self.spectral_radius / np.amax(w))
            if self.verbose:
                print('Reservoir (W) created. Shape: ', self.W.shape, '.')
            pass

        else:
            raise NameError('Perhaps you have not written the graph model correctly. Try erdos-renyi or watts-strogatz.')


    def generatewout(self):
        self.W_out = np.zeros((self.output_size, 1 + self.input_size + self.state_size))
        if self.verbose:
            print('Matrix W_out created. Shape: ', self.W_out.shape, '.')
        pass

    def generateMatrices(self):
        #np.random.seed(2)
        self.generatewin()
        self.generatewout()
        self.generatew()

    # FORWARD PASS

    def forward_pass(self, last_state, input_signal):
        vector_state_prime = np.tanh(np.dot(self.W_in, np.concatenate((np.ones(1), input_signal))) + np.dot(self.W, last_state))
        new_state = (1 - self.alpha) * last_state + self.alpha * vector_state_prime + self.noise * np.random.rand(last_state.shape[0])
        return new_state
```

```
121
122
123     # TRAIN ECHO STATE
124
125     def train(self, train_input, train_label):
126
127         # generate matrices and state size
128
129         design_matrix = np.zeros((1 + self.input_size + self.state_size,
    train_input.shape[0]))
130
131         # train
132
133         for epoch in range(self.epochs):
134             for i in range(train_input.shape[0]):
135                 self.vector_state = np.random.uniform(-1,1,self.
    state_size)
136                 self.vector_state = self.forward_pass(self.vector_state,
    train_input[i])
137                 design_matrix[:, i] = np.concatenate((np.ones(1),
    train_input[i], self.vector_state))
138
139             # Ridge regression
140             a = np.dot(train_label.T, design_matrix.T)
141             b = np.dot(design_matrix, design_matrix.T) + self.beta * np.
    identity(1 + self.input_size + self.state_size, dtype=float)
142             b_inverse = np.linalg.inv(b)
143             self.W_out = np.dot(a, b_inverse)
144
145     # PREDICT
146
147     def predict(self, test_data):
148         self.vector_state = np.random.uniform(-1,1,self.state_size)
149         self.vector_state = self.forward_pass(self.vector_state,
    test_data)
150         return np.real(np.dot(self.W_out, np.concatenate((np.ones(1),
    test_data, self.vector_state))))
```

A Echo state foi criada utilizando uma classe Python. Nas linhas 1 à 5 são importadas bibliotecas necessárias para o funcionamento da ESN. Nas linhas 24 à 38 são listados os parâmetros que a classe **echoState** aceita, sendo eles:

- **input scaling**: ordem das entradas da matriz da primeira camada da ESN.
- **input size**: tamanho dos dados de entrada da ESN, geralmente sendo um vetor de dimensão $N \times 1$.

- **output size**: tamanho dos dados de saída da ESN.
- **state size**: dimensão do reservatório de neurônios da ESN, sendo este uma matriz quadrada.
- **connection probability**: probabilidade de conexão entre dois nós quaisquer nos grafos de Erdős–Rényi e Watts e Strogatz.
- **k neighbors**: quantidade inicial de vizinhos conectados a um nó qualquer no grafo de Watts e Strogatz.
- **spectral radius**: raio espectral do reservatório da ESN.
- **noise**: ruído aplicado ao estado da ESN.
- **alpha**: taxa de "vazamento" na atualização do estado da ESN.
- **beta**: coeficiente de regularização no método *Ridge regression*.
- **epochs**: número de vezes em que o treinamento da ESN vai percorrer sobre todos os dados de treinamento.
- **graph model**: modelo do grafo que será utilizado pela ESN.

Nas linhas 68, 74 e 103 são feitas as funções que criam as matrizes de entrada, reservatório e saída, respectivamente. A função da linha 109 chama as funções que acabamos de mencionar. Na linha 117 temos a função **forward pass** que serve para atualizar o estado da ESN. A função de treinamento é criada na linha 125 com o método *Ridge regression* aplicado na linha 139. Na linha 147 é criada a função **predict** que serve para fazer previsões em novos dados utilizando a ESN com os pesos treinados.

A.2 DEEP ECHO STATE

Abaixo segue o código da Deep echo state em Python.

```
1 import numpy as np
2 import numpy.linalg as LA
3 import networkx as nx
4
5 def get_p2_norm(x):
6     x = x.conj().T @ x
7     w, v = LA.eig(x)
8     return w, v, x
9
```

```
def get_matrix_normalized(x, sigma):
    w, _, _ = get_p2_norm(x)
    x = x * sigma/ np.sqrt(np.amax(np.real(w)))
    return x

class deep_echo_state_np:
    """
    comments here!
    """

    def __init__(
        self,
        input_scaling,
        inter_layer_scaling,
        spectral_radius,
        epochs,
        connection_probability,
        k_neighbors,
        N_u,
        N_y,
        N_r,
        N_l,
        beta = 0,
        verbose = False,
        method = 'ridge_regression',
        graph_model = 'erdos-renyi'):

        # parameters

        self.sigma_in = input_scaling
        self.sigma_inter_layer = inter_layer_scaling
        self.spectral_radius = spectral_radius
        self.epochs = epochs # epochs
        self.connection_probability = connection_probability # probability of connection between nodes
        self.k_neighbors = k_neighbors # number of nodes each will be connected with
        self.N_u = N_u # input length
        self.N_y = N_y # output length
        self.N_r = N_r # reservoir size
        self.N_l = N_l # number of layers in the deep echo state
        self.beta = beta # coefficient for the regularization in the ridge regression
        self.x = np.zeros((self.N_l, self.N_r))   # the state of the deep echo state
        #self.x = np.random.uniform(size = (self.N_l, self.N_r))   # the state of the deep echo state
```

```
52          #self.alpha = np.random.uniform(low = 0.1, high = 1, size=(self.
    N_l,1)) # tensor containing the leaky parameter for each iteration in
    the deep echo state
53          self.alpha = np.ones((self.N_l,1)) * 0.95
54          self.verbose = verbose  # show logs during the running of the
    deep echo
55          self.method = method # method used for training the readout
56          self.graph_model = graph_model
57
58
59      # create a erdos renyi graph
60      def _erdos_renyi_graph(self):
61          G = nx.fast_gnp_random_graph(n = self.N_r, p = self.
    connection_probability)
62          A = nx.adjacency_matrix(G).todense()
63          W = np.squeeze(np.asarray(A))
64          W = np.multiply(W, np.random.uniform(-0.5, 0.5, (self.N_r, self.
    N_r)))
65          return W
66
67      # create a watts strogatz graph
68      def _watts_strogatz_graph(self):
69          G = nx.watts_strogatz_graph(n = self.N_r, k = self.k_neighbors, p
     = self.connection_probability)
70          A = nx.adjacency_matrix(G).todense()
71          W = np.squeeze(np.asarray(A))
72          W = np.multiply(W, np.random.uniform(-0.5, .05, (self.N_r, self.
    N_r)))
73          return W
74
75
76      # initiate tensors
77      def _initiate_tensors(self):
78          # initiate the reservoir tensor
79          self.W_reservoir = np.zeros((self.N_l,self.N_r,self.N_r))
80          if self.graph_model == 'erdos-renyi':
81              for layer in range(self.N_l):
82                  self.W_reservoir[layer] = self._erdos_renyi_graph()
83          elif self.graph_model == 'watts-strogatz':
84              for layer in range(self.N_l):
85                  self.W_reservoir[layer] = self._watts_strogatz_graph()
86          else:
87              raise NameError('Perhaps you have not written the graph model
     correctly. Try erdos-renyi or watts-strogatz.')
88
89          # calculate the spectral radius of the reservoir tensor
90          spectral_r = []
```

```
        for layer in range(self.N_l):
            temp_reservoir = (1 - self.alpha[layer]) * np.identity(self.
    N_r) + self.alpha[layer] * self.W_reservoir[layer]
            w, _ = LA.eig(temp_reservoir)
            w = np.real(w)
            spectral_r.append(np.amax(w))

        self.W_reservoir = self.W_reservoir * self.spectral_radius / np.
    amax(spectral_r)


        # initiate the W_in tensor
        self.W_in = np.random.uniform(low=-1, high=1, size=(self.N_r,self
    .N_u))
        self.W_in = get_matrix_normalized(self.W_in, self.sigma_in)

        # initiate the W_connection tensor
        self.W_connection = np.random.uniform(low=-1, high=1, size=(self.
    N_l - 1,self.N_r,self.N_r))
        for layer in range(self.N_l - 1):
            self.W_connection[layer] = get_matrix_normalized(self.
    W_connection[layer], self.sigma_inter_layer)

        # initiate the W_out tensor
        self.W_out = np.random.uniform(size=(self.N_y, self.N_l * self.
    N_r))

        if self.verbose:
            print('W_reservoir.shape: ', self.W_reservoir.shape)
            print('W_in.shape: ', self.W_in.shape)
            print('W_connection.shape: ', self.W_connection.shape)
            print('W_out.shape: ', self.W_out.shape)

        pass


    # forward pass
    def _forward(self, last_state, input_signal):
        # initiate state
        #self.x = np.zeros((self.N_l, self.N_r))

        # comput next state
        for layer in range(self.N_l):
            if layer == 0:
                self.x[layer] = (1 - self.alpha[layer]) * last_state[
    layer] + self.alpha[layer] * np.tanh(np.dot(self.W_in, input_signal) +
     np.dot(self.W_reservoir[layer], last_state[layer]))
```

```
            else:
                self.x[layer] = (1 - self.alpha[layer]) * last_state[layer] + self.alpha[layer] * np.tanh(np.dot(self.W_connection[layer - 1], self.x[layer - 1]) + np.dot(self.W_reservoir[layer], last_state[layer]))

        return self.x


    def _train(self, train_input, train_label):

        """
        The train_input tensor must have the shape (N_samples, N_u), where N_u is the input dimension and N_samples is the number of samples
        The train_label tensor must have the shape (N_samples, N_y), where N_y is the output dimension and N_samples is the number of samples
        """

        # initiate the design matrix
        design_matrix = np.zeros((self.x.shape[0] * self.x.shape[1], train_input.shape[0]))

        # iteration over epochs
        for epoch in range(self.epochs):
            # iteration over the train data
            for i in range(train_input.shape[0]):
                # get new state
                self.x = self._forward(self.x, train_input[i])

                # concatenate states
                X = np.reshape(self.x,[self.x.shape[0] * self.x.shape[1]])

                if self.verbose:
                    print(f"Full deep echo state shape: {X.shape}")

                # update the design matrix
                design_matrix[:, i] = X

                if self.verbose:
                    print(f"Design matrix shape: {design_matrix.shape}.")

            # compute the output
            output = self.W_out @ X

```

```
168             if self.method == 'ridge_regression':
169                 # update W_out
170                 self.W_out = np.transpose(train_label) @ np.transpose(
    design_matrix) @ np.linalg.inv(design_matrix @ np.transpose(
    design_matrix) + self.beta * np.identity(self.x.shape[0] * self.x.
    shape[1]))
171
172             if self.verbose:
173                 print(f"Epoch {epoch + 1} of {self.epochs}", end = '\r')
174
175     # function to predict
176     def _predict(self, input_signal):
177         # get new state
178         self.x = self._forward(self.x, input_signal)
179
180         # compute the output signal
181         X = np.reshape(self.x,[self.x.shape[0] * self.x.shape[1]])
182         output_signal = self.W_out @ X
183         return output_signal
```

A deep echo state foi criada utilizando uma classe Python. Nas linhas 1 à 3 são importadas as bibliotecas necessárias para o funcionamento da rede neural. Na linha 5 é criada uma função que calcula o quadrado da norma p_2 de uma matriz. Na linha 10 é definida uma função que normaliza uma matriz de acordo com a norma p_2 escolhida. Nas linhas 22 à 35 são listados os parâmetros que a deep ESN aceita, sendo eles:

- **input scaling**: norma p_2 da matriz da camada de entrada.
- **inter layer scaling**: norma p_2 do tensor que conecta as camadas escondidas da deep ESN.
- **spectral radius**: raio espectral do tensor do reservatório.
- **epochs**: número de vezes em que o treinamento vai percorrer sobre todos os dados de trinamento.
- **connection probability**: probabilidade de criação de uma aresta entre dois nós quaisquer nos modelos de Erdős–Rényi e Watts e Strogatz.
- **k neighbors**: quantidade inicial de vizinhos conectados a um nó qualquer no grafo de Watts e Strogatz.
- N_u: tamanho do vetor de entrada.
- N_y: tamanho do vetor de saída.
- N_r: tamanho do reservatório de cada camada.

- N_l: número de camadas.
- **beta**: coeficiente de regularização no método *Ridge regression*.
- **method**: método de treinamento para ajustar os pesos da matriz de saída.
- **graph model**: modelo do grafo que será utilizado pela ESN.

Na linha 60 é definido o método que cria os tensores necessários para a deep ESN. Na linha 122 é definido o método que atualiza o estado da deep ESN. Na linha 136 é criado o método de treinamento e na linha 176 é definido o método para fazer previsões em novos dados.

B RESULTADOS

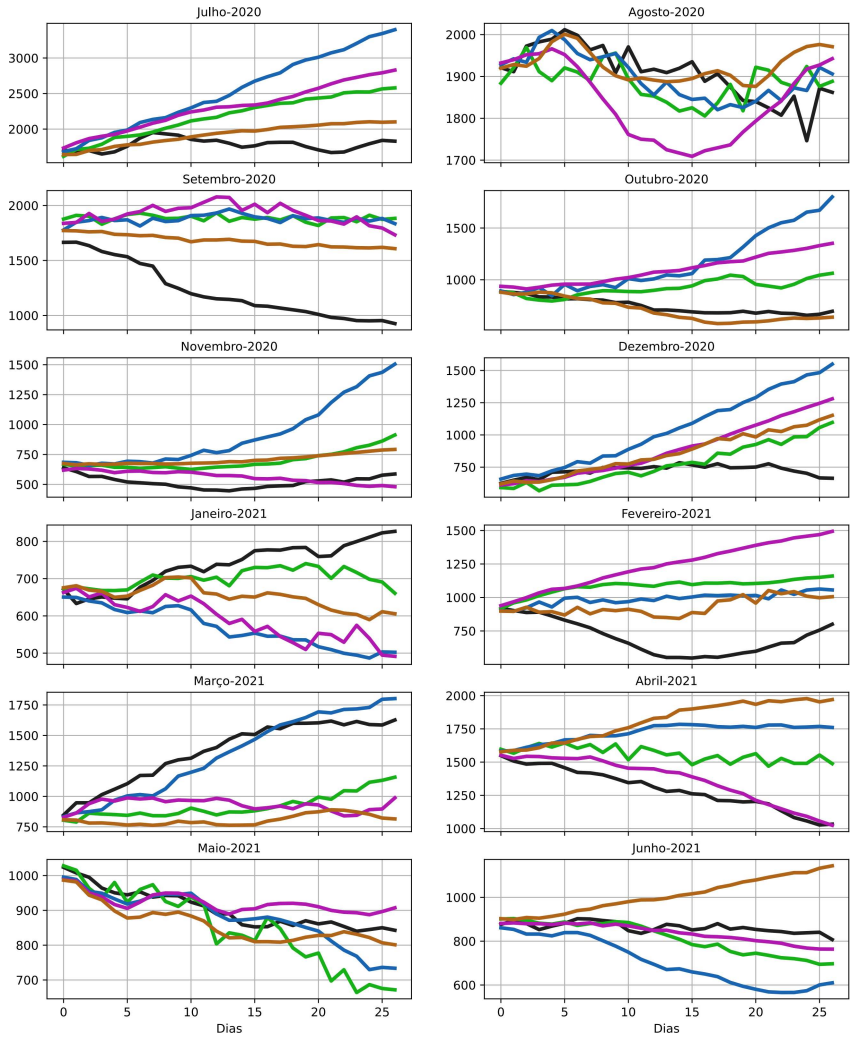

Figura B.1 – Previsões na média móvel de casos. Em preto são os dados reais, verde a ESN com reservatório aleatório, azul a ESN com reservatório pequeno mundo, roxo a *deep* ESN com reservatórios aleatórios e marrom a *deep* ESN com reservatórios mundo pequeno.

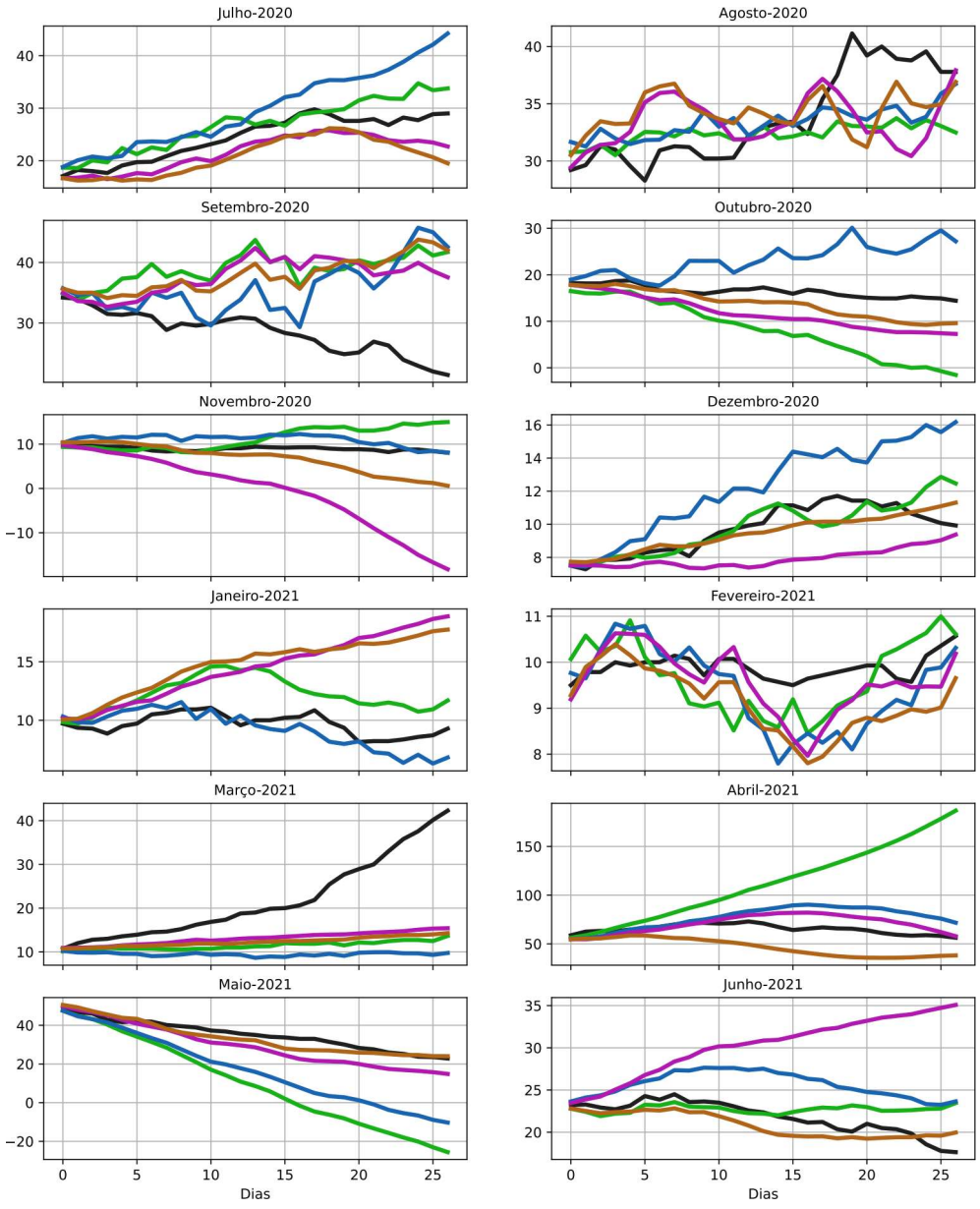

Figura B.2 – Previsões na média móvel de mortes. Em preto são os dados reais, verde a ESN com reservatório aleatório, azul a ESN com reservatório pequeno mundo, roxo a *deep* ESN com reservatórios aleatórios e marrom a *deep* ESN com reservatórios mundo pequeno.

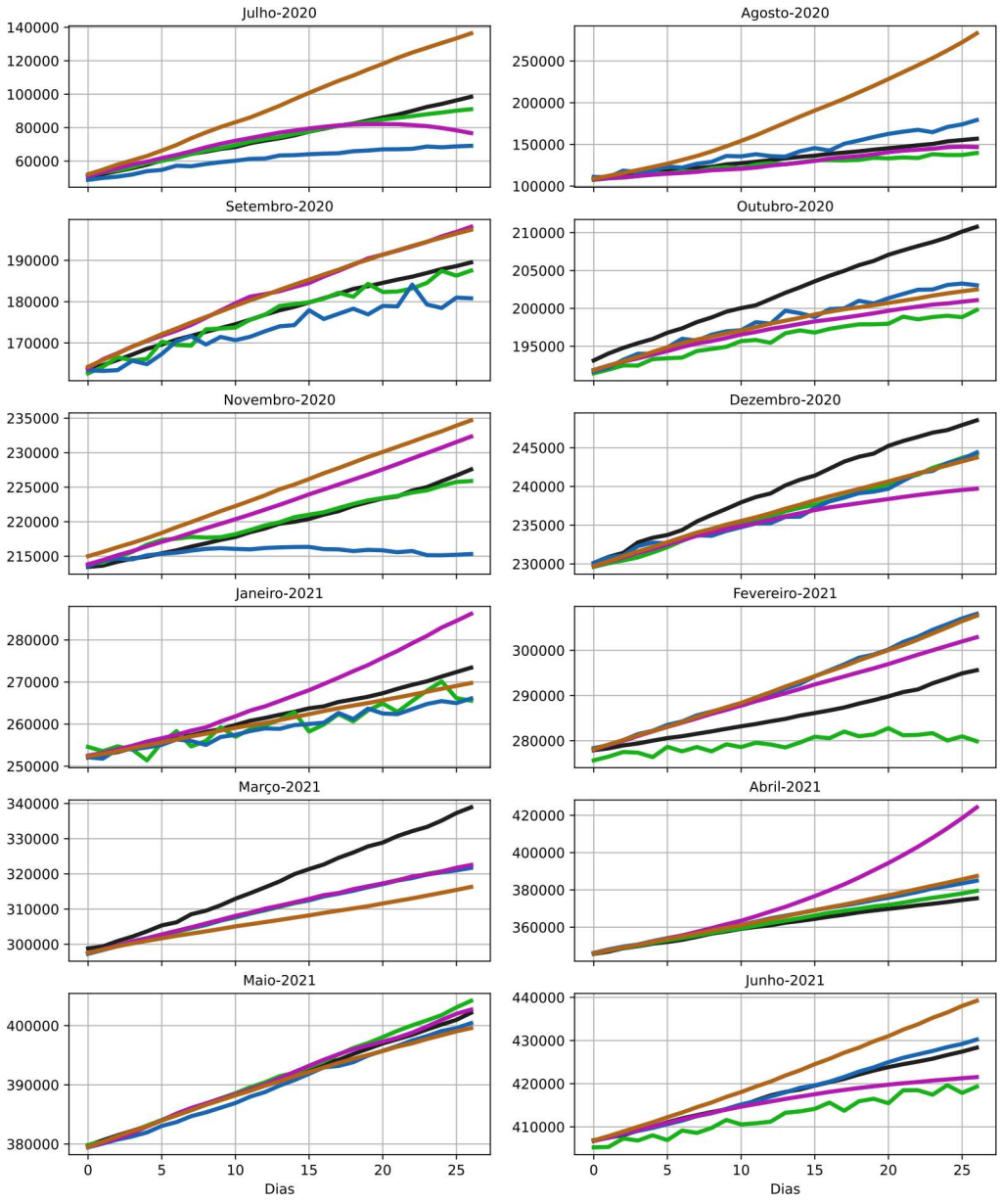

Figura B.3 – Previsões nos casos cumulativos. Em preto são os dados reais, verde a ESN com reservatório aleatório, azul a ESN com reservatório pequeno mundo, roxo a *deep* ESN com reservatórios aleatórios e marrom a *deep* ESN com reservatórios mundo pequeno.

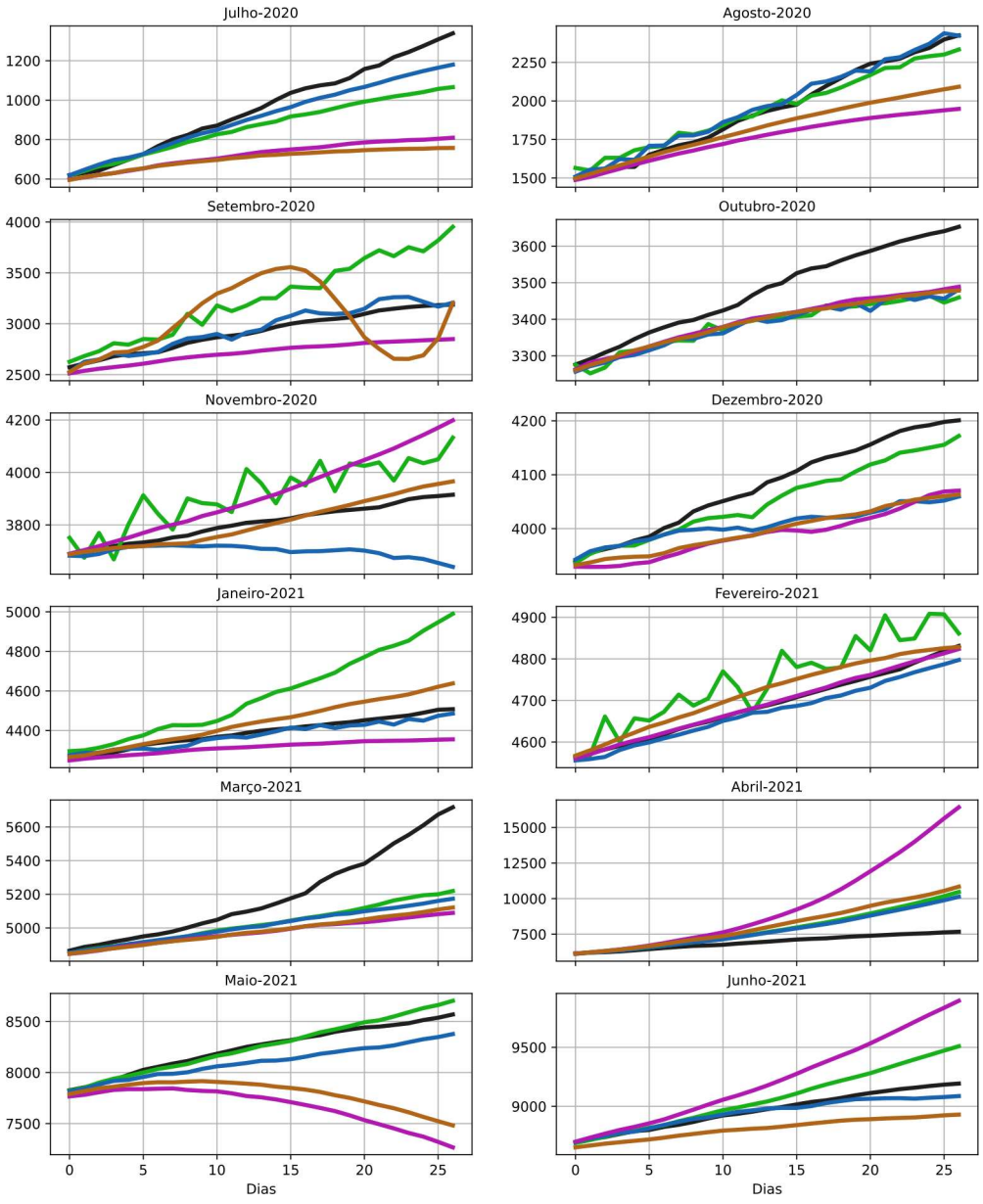

Figura B.4 – Previsões nas mortes cumulativas. Em preto são os dados reais, verde a ESN com reservatório aleatório, azul a ESN com reservatório pequeno mundo, roxo a *deep* ESN com reservatórios aleatórios e marrom a *deep* ESN com reservatórios mundo pequeno.

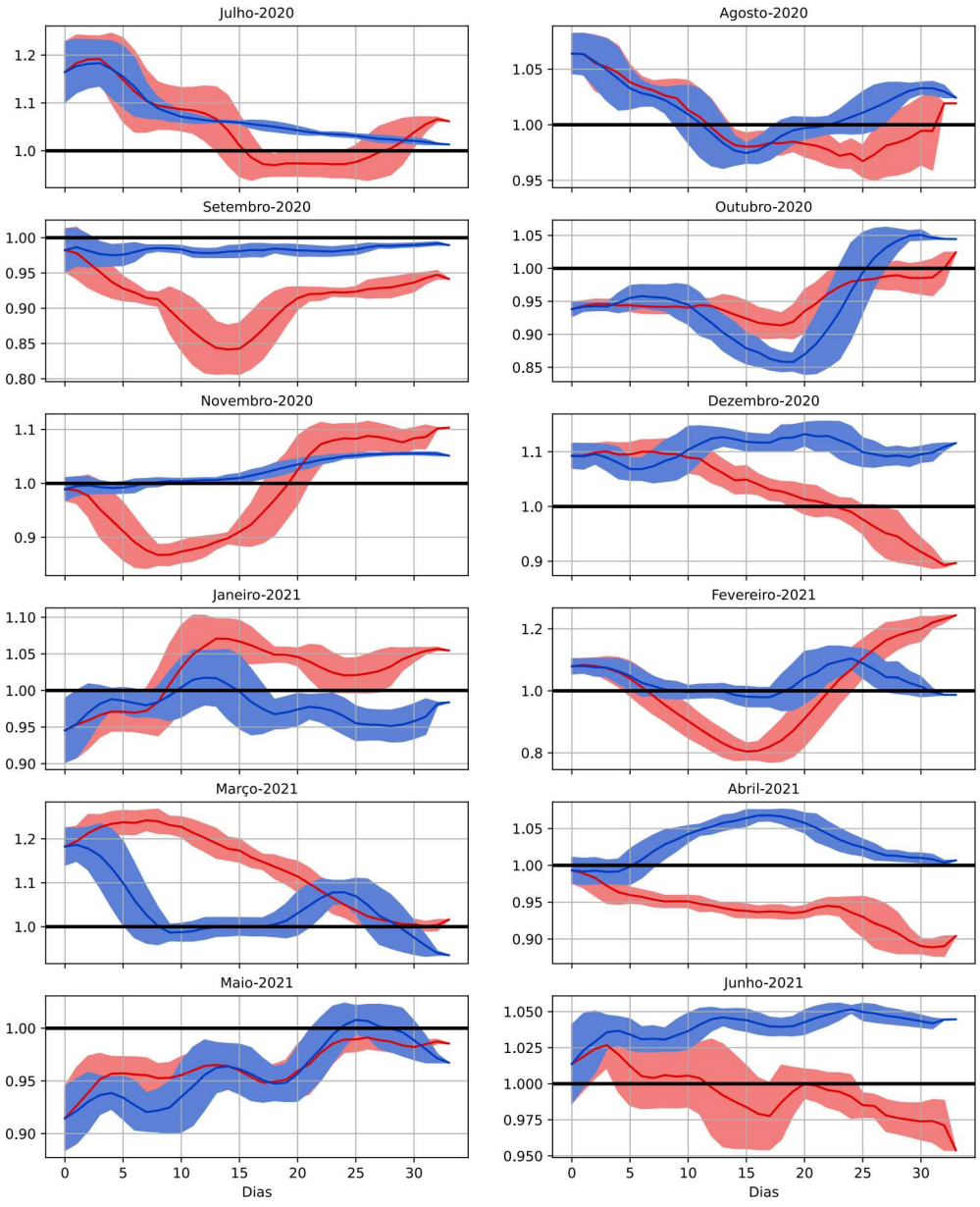

Figura B.5 – Previsões mensais do número de reprodução instantâneo. Em vermelho os dados reais e em azul as previsões da *deep* ESN com reservatórios pequeno mundo. A parte sombreada em vermelho ou azul representa o intervalo de confiança de 95%.