



UNIVERSIDADE FEDERAL DA BAHIA  
INSTITUTO DE GEOCIÊNCIAS  
CURSO DE GRADUAÇÃO EM GEOFÍSICA

GEO213 – TRABALHO DE GRADUAÇÃO

REDES NEURAIS RECORRENTES DE  
ELMAN APLICADAS À INVERSÃO  
ACÚSTICA DA FORMA DE ONDA  
COMPLETA UTILIZANDO  
DIFERENCIADA AUTOMÁTICA

MARCOS CONCEIÇÃO

SALVADOR – BAHIA  
DEZEMBRO – 2021

**Redes neurais recorrentes de Elman aplicadas à inversão acústica da forma de  
onda completa utilizando diferenciação automática**  
por  
**MARCOS CONCEIÇÃO**

Orientador: Prof. Dr. Reynam Pestana

**GEO213 – TRABALHO DE GRADUAÇÃO**

DEPARTAMENTO DE GEOFÍSICA  
DO  
INSTITUTO DE GEOCIÊNCIAS  
DA  
UNIVERSIDADE FEDERAL DA BAHIA

**COMISSÃO EXAMINADORA**

- 
- \_\_\_\_\_  
Dr. Reynam C. Pestana
- 
- \_\_\_\_\_  
Dr. Edvaldo Suzarthe de Araújo
- 
- \_\_\_\_\_  
MC. Victor Koehne Ramalho

**DATA DA APROVAÇÃO: 06/12/2021**

NÓS TEMOS QUE SABER.  
NÓS SABEREMOS.

*David Hilbert*

# Resumo

Neste trabalho é implementada uma rede neural recorrente, como estipulada por Elman, em cujo interior são utilizadas operações de diferenças finitas para a modelagem de dados sísmicos pela solução da equação da onda acústica. A implementação foi realizada num ambiente de diferenciação automática. O treinamento de tal rede é o processo de inversão dos parâmetros físicos do meio propagado. Na Geofísica, este procedimento é chamado de inversão da forma de onda completa, sendo tradicionalmente implementado por meio do método adjunto. A diferenciação automática, por outro lado, tem por vantagem ser prontamente utilizável em inversões quaisquer. A reformulação desta inversão em termos do treinamento de uma rede neural possibilita novas interações do método com o universo do aprendizado de máquina. A inversão implementada via rede neural recorrente é testada através do modelo Marmousi e os métodos SGD, Momento e Adam, comuns ao treinamento de redes neurais, são utilizados possibilitando a obtenção de resultados bastante satisfatórios, quando comparadas ao modelo verdadeiro. Como ainda é difícil encontrar materiais em português que introduzam as redes neurais e as amarrem aos processos de modelagem acústica, um objetivo primordial deste trabalho é servir como material didático, que resume estes tópicos para estudos futuros.

# Abstract

In this work a recurring neural network is implemented, as stipulated by Elman, in which finite difference operations are used for the modeling of seismic data by solving the acoustic wave equation. Implementation was carried out in an automatic differentiation environment. The training of such a network is the process of inversion of the physical parameters of the propagated medium. In geophysics, this procedure is called full-waveform inversion and is traditionally implemented through the adjoint method. Automatic differentiation, on the other hand, has the advantage of being readily usable at general inversion process. The reformulation of the inversion problem as the training of a neural network enables new interactions of the method with the universe of machine learning. The inversion implemented via recurrent neural network was tested through the Marmousi model and the SGD, Moment and Adam, common to neural network training methods, were used enabling quite satisfactory results when compared to the true model. As only a handful of materials were found to introduce neural networks and tie them to acoustic modeling processes in Portuguese, a primary objective of this work is to serve as didactic material that sums up these topics for future studies.

# Sumário

<b>Resumo</b>	<b>3</b>
<b>Abstract</b>	<b>4</b>
<b>Introdução</b>	<b>10</b>
<b>1 Aprendizado estatístico</b>	<b>12</b>
1.1 Dados como matrizes . . . . .	13
1.2 Função objetivo . . . . .	14
1.3 Atualização de parâmetros . . . . .	17
1.3.1 Máximo declive . . . . .	17
1.3.2 Otimização estocástica e o máximo declive estocástico (SGD) . . . . .	18
1.3.3 Momento . . . . .	19
1.3.4 Estimativa adaptável de momento (Adam) . . . . .	20
1.4 Validação estatística, viés e variância de modelos . . . . .	21
1.5 Interrupção precoce . . . . .	22
<b>2 Cálculo do gradiente</b>	<b>25</b>
2.1 Diferenciação manual . . . . .	26
2.2 Diferenciação simbólica . . . . .	26
2.3 Diferenciação numérica . . . . .	26
2.4 Diferenciação automática (autodiff) . . . . .	27
2.4.1 Diferenciação automática direta . . . . .	28
2.4.2 Diferenciação automática reversa . . . . .	29
<b>3 De regressões lineares a redes neurais</b>	<b>32</b>
3.1 Regressão linear . . . . .	32
3.2 Regressão logística . . . . .	34
3.3 Redes neurais artificiais . . . . .	36
3.3.1 Redes neurais pré-alimentadas (FNN) . . . . .	37
3.3.2 Redes neurais recorrentes (RNN) . . . . .	40
<b>4 Modelagem da onda acústica como uma rede neural recorrente</b>	<b>44</b>
4.1 Método das diferenças finitas (FDM) . . . . .	44
4.2 Rede de Elman para propagação acústica . . . . .	47
4.3 Inversão da forma de onda completa (FWI) . . . . .	47
4.4 Inversão como treinamento de uma rede recorrente de Elman . . . . .	48
4.5 Inversão multiescala . . . . .	49

<b>5 Metodologia e Resultados</b>	<b>50</b>
5.1 Metodologia . . . . .	50
5.2 Resultados da inversão . . . . .	53
<b>6 Conclusões</b>	<b>61</b>
<b>Agradecimentos</b>	<b>63</b>
<b>A Derivação do cálculo do gradiente da rede de Elman para modelagem acústica por diferenciação automática</b>	<b>66</b>
<b>B Receita para o treinamento estocástico de um modelo supervisionado</b>	<b>70</b>
<b>C Transformação <i>one-hot</i></b>	<b>73</b>
<b>D Código para diferenciação automática direta</b>	<b>74</b>
<b>E Código para diferenciação automática reversa</b>	<b>78</b>
<b>F Código para treinamento de rede neural MLP usando diferenciação automática</b>	<b>82</b>
<b>Referências</b>	<b>86</b>

# Listas de Tabelas

2.1	Processo de diferenciação automática direta da função $\vec{f}$ em relação ao seu primeiro parâmetro, $p_1$ no ponto $\vec{p} = [2 \ 2]^\top$ . . . . .	29
2.2	Processo de diferenciação automática reversa da primeira saída da função $\vec{f}$ , $f_1$ , em relação aos seus parâmetros no ponto $\vec{p} = [2 \ 2]^\top$ . . . . .	31
2.3	Características dos métodos de diferenciação. (*): Caso a função estudada seja dada em forma fechada e um tratamento simbólico for dado ao grafo. . .	31

# Listas de Figuras

1.1	Curvas representam métricas de erro avaliadas sobre os dados de treinamento (em azul) e de validação (em vermelho) ao longo das épocas de otimização. O melhor modelo se localiza onde a curva de validação atinge seu menor valor, ponto dado pela linha tracejada. . . . .	23
2.1	Grafo da função $\vec{f}(\vec{p}) = [(p_2 + p_1 p_2) \ (\sin p_2)]^\top = [f_1 \ f_2]^\top$ . . . . .	28
3.1	Grafo de um modelo de regressão linear com três entradas e duas saídas. . . . .	32
3.2	Grafo de um modelo de regressão logística multinomial com três entradas e duas saídas probabilísticas. . . . .	34
3.3	Gráfico da função sigmoide ( $\sigma(z) = \frac{1}{1+e^{-z}}$ ) . . . . .	36
3.4	Estrutura básica de um neurônio. . . . .	37
3.5	Ilustração de uma rede neural tipo MLP com número de camadas $L = 5$ e sequência de neurônios por camada $\{n_l\} = \{3, 4, 5, 3, 2\}$ . Esta rede possui entradas em $\mathcal{R}^3$ e saídas em $\mathcal{R}^2$ . . . . .	39
3.6	Redes recorrente de Jordan em suas representações compacta (esquerda) e desenrolada (direita). . . . .	42
3.7	Redes recorrente de Elman em suas representações compacta (esquerda) e desenrolada (direita). . . . .	42
4.1	Rede recorrente de Elman para modelagem. $\mathbf{P}_{t-2}$ e $\mathbf{P}_{t-1}$ e $\mathbf{S}_t$ são utilizados na composição de um novo par $(\mathbf{P}_{t-1}, \mathbf{P}_t)$ por meio de modelagem por diferenças finitas (FDM). $\vec{y}_t$ recebe as amplitudes de $\mathbf{P}_t$ na posição dos receptores por meio de uma operação $\delta_{\vec{r}, \vec{r}_r}$ . . . . .	47
5.1	Modelo Marmousi real utilizado na produção dos dados sintéticos (A) e modelo borrado utilizado como ponto inicial da para o treinamento da rede (B). . . . .	51
5.2	Modelagem de sismograma num modelo de duas interfaces em (a), modelagem de onda direta do mesmo tiro em (b) e subtração de (a) por (b) em (c), na qual se apresentam apenas os eventos de reflexão e refração. . . . .	52
5.3	Da esquerda para a direita se apresentam os sismogramas: do décimo primeiro tiro, do décimo sexto tiro, e do vigésimo segundo tiro. . . . .	52
5.4	Mesmo sismograma modelado com fontes de frequência: 2,7 Hz, 5,3 Hz, e 8 Hz, da esquerda para a direita. Observe como algumas feições do sismograma das frequências mais altas não são visíveis em baixas frequências. . . . .	53
5.5	Função custo (esquerda) e erro médio (direita) do modelo em função das épocas de otimização para cada modelo. . . . .	54

---

5.6	Modelo obtido na última época de otimização durante a etapa de modelagem sob a banda de frequência de 0 a 2.7 $Hz$ usando SGD (a), Momento (b) e Adam (c). . . . .	56
5.7	Modelo obtido na última época de otimização durante a etapa de modelagem sob a banda de frequência de 0 a 5.3 $Hz$ usando SGD (a), Momento (b) e Adam (c). . . . .	57
5.8	Melhor modelo obtido durante a otimização na última banda de frequência (0 a 8 $Hz$ ) usando SGD (a), Momento (b) e Adam (c). . . . .	58
5.9	Diferença entre o melhor modelo obtido na FWI multiescala e o modelo inicial para os métodos: SGD (a), Momento (b) e Adam (c). . . . .	59
5.10	Perfil vertical de velocidades em $x = 2.500$ m para o modelo verdadeiro (preto) e os modelos resultantes da FWI por meio do método SGD (tracejada em verde), Momento (tracejada em lilás) e Adam (tracejada em vermelho). . . . .	60
A.1	Grafo da dependência (setas vermelhas) do custo $c$ em relação ao campo de pressão $P_t$ . Num esquema de diferenças finitas de segunda ordem no tempo, três caminhos vão do custo ao campo de pressão em um tempo $t$ : o caminho $A$ indica o erro gerado diretamente pela falta de fase entre o dado predito e o observado nos receptores no instante $t$ ; o caminho $B$ se refere ao erro propagado à pressão do instante seguinte ( $t + 1$ ) devido ao erro presente no instante $t$ ; e o caminho $C$ trata do erro causado à pressão em $t + 2$ devido ao erro em $P_t$ . . . . .	67

# Introdução

Na última década, o ritmo de evolução dos computadores tornou possível a popularização das técnicas de redes neurais artificiais, anteriormente muito custosas. Atualmente estas redes são aplicadas em virtualmente todas áreas da Ciência, afetando ao menos indiretamente a vida de todos. As redes são muito valorosas por sua versatilidade, nos fornecendo capacidades preditoras dignas de ficção<sup>1</sup>. Porém esta habilidade de resolver problemas altamente não-lineares frequentemente vem ao preço de modelos praticamente inexplicáveis. O conjunto dos incontáveis parâmetros que se emaranham para formar estas redes é treinado para a obtenção dos melhores e mais robustos resultados, mas a conexão entre cada um de seus elementos e as previsões realizadas é indecifrável em termos práticos. Como dito por Castelvecchi (2016), o conhecimento fica retido nas redes ao invés de conosco.

Confiar em modelos que mal compreendidos é inquietantemente inseguro. Por isto, muitas formas de retornar o controle destes algoritmos ao ser humano são discutidas na atualidade, sobretudo no contexto das aplicações relacionadas com a Física. A primeira estratégia que vale ser citada aqui são as redes neurais informadas de Física (do inglês, *Physics informed neural network* — PINN). Uma PINN tem em seu treinamento uma penalização sobre erros de previsão, como qualquer outro modelo de aprendizado de máquina, mas soma-se a ela uma penalização sobre a insatisfação da Física do problema, a qual é calculada a partir de suas equações físicas e as previsões do modelo. Uma vez que seus parâmetros são treinados para satisfazer a leis da realidade, é mais fácil crer que uma PINN evitará previsões absurdas. Aplicações interessantes das PINNs podem ser vistas nos trabalhos recentes, como os de Karimpouli e Tahmasebi (2020), Moseley et al. (2020) e Raissi et al. (2019). Este esquema vem sendo utilizado nos últimos anos para realização de modelagens e inversões menos custosas.

Outra estratégia também estudada nos últimos anos é a construção de redes cujo interior é parcialmente dado por um processo de modelagem física. Estes casos são interessantes, pois os ambientes de implementação de redes neurais são baseados em técnicas de diferenciação automática, que tornam o cálculo do gradiente da função objetivo a ser otimizada muito prático. Graças a este tipo de ambiente, é possível inverter virtualmente qualquer equação física uma vez que sua modelagem direta já tenha sido implementada. Alguns trabalhos interessantes utilizando esta metodologia são apresentados por: Sun et al. (2019) para o caso das ondas acústicas; Wang et al. (2021) para o caso das ondas elásticas; e por Hu et al. (2020) para o caso de ondas eletromagnéticas. Esses trabalhos se utilizam de redes neurais recorrentes, às quais tratam o tempo de forma iterativa e por isto são similares a alguns métodos de integração numérica de equações diferenciais.

Este trabalho tem por primeiro objetivo servir como material didático de revisão para

---

<sup>1</sup>O resultado de Devlin et al. (2018) é o modelo BERT, que não apenas é capaz de interpretar textos, como supera os resultados da média dos seres humanos ao responder perguntas sobre o que leu.

trabalhos futuros na área de redes neurais e inversão utilizando diferenciação automática. Seu segundo objetivo é apresentar uma rede recorrente de Elman na qual são internalizadas operações de modelagem sísmica para realizar a recuperação de um modelo de velocidades. No primeiro capítulo é feita a introdução do leitor aos conceitos básicos de aprendizado estatístico que levam aos modelos de aprendizado de máquina atuais, entre os quais as redes neurais são incluídas, bem como aos algoritmos de otimização estocástica mais utilizados dentro desta área atualmente. Como o treinamento dos modelos tipicamente depende de métodos baseados no gradiente de uma função objetivo, o segundo capítulo visa demonstrar as possibilidades existentes para o seu cálculo, bem como compará-los. O terceiro capítulo introduz as redes neurais artificiais como uma sucessão natural das regressões lineares, sendo finalizado pela definição das redes recorrentes de Elman. Em sequência, o quarto capítulo lida com a modelagem sísmica tradicional por meio da equação da onda acústica, cujos parâmetros são a distribuição de velocidades de propagação do meio estudado. Discute-se então sobre como incluir a Física da simulação acústica tradicional numa rede de Elman que opera modelagem, a partir da qual a inversão dos parâmetros — completamente compreendidos — é dada pelo seu treinamento. Demonstra-se que tal procedimento é matematicamente equivalente à inversão da forma completa de onda (do inglês *full-waveform inversion*, FWI) tradicional, mesmo sem o uso explícito do método adjunto para o cálculo do gradiente. O quinto capítulo apresenta os resultados da experimentação com a rede desenhada através da aplicação da mesma à inversão do modelo Marmousi (Brougois et al., 1990) utilizando os métodos de otimização típicos do treinamento de redes neurais SGD, Momento e Adam.

# Capítulo 1

## Aprendizado estatístico

A primeira motivação por trás de um modelo estatístico é estimar, a partir de um conjunto de características de determinado fenômeno (*i.e.*, altitude, volume) um outro conjunto de atributos associados (*i.e.*, temperatura, pressão) (James et al., 2013). Para ser genérico, um modelo possui uma série de parâmetros adaptáveis, que são iterativamente otimizados para um problema específico a partir da minimização ou maximização de uma função objetivo definida, processo conhecido como *treinamento* ou *ajuste* do modelo. Existem diferentes métodos de otimização, e os mais utilizados no treinamento de modelos estatísticos são baseados no cálculo do gradiente da função objetivo. Alguns destes serão alvos de discussão na Seção 1.3.

Além de um bom preditor, um modelo estatístico possui outro objetivo: possibilitar uma melhor interpretação aos dados adquiridos. Cada parâmetro de um modelo treinado guarda um condensado de informações acerca dos dados aplicados em seu treinamento. Assim, a análise destes parâmetros e de como se relacionam com as previsões do modelo gera conhecimento sistemático sobre o fenômeno estudado.

A maior parte dos métodos de aprendizado de máquina recaem em duas grandes áreas: métodos supervisionados e não-supervisionados. O aprendizado *supervisionado* envolve algoritmos treinados a partir de pares de dados de entrada e saída já conhecidos. A otimização dos parâmetros é feita de forma que o algoritmo escolhido realize um bom mapeamento entre o espaço das entradas e o das saídas dentro dos dados consumidos. Este grupo de algoritmos ainda se divide em problemas de *regressão*, quando as saídas são dados quantitativos (*i.e.*, regressão linear,  $k$ -vizinhos mais próximos), e problemas de *classificação*, quando as saídas são dados categóricos, ou como também são chamados, qualitativos (*i.e.*, regressão logística, árvore de decisão).

Algoritmos cujo treinamento requer apenas dados de entrada, e não suas saídas associadas, são ditos *não-supervisionados*. Eles buscam estrutura nos dados estudados. Estes métodos se dividem entre métodos de *agrupamento* (do inglês *clustering*), nos quais amostras similares são associadas a um mesmo grupo, resultando em saídas categóricas (*i.e.*,  $k$ -médias e modelo de misturas gaussianas) e os modelos de *variáveis latentes* (*i.e.*, análise de componentes principais e dicionários). Os algoritmos não-supervisionados fogem ao escopo deste trabalho e não serão aqui tratados. Dentre os algoritmos supervisionados, os algoritmos de regressão terão o foco principal neste material.

A notação da álgebra linear é frequentemente utilizada na descrição dos métodos estatísticos, uma vez que tensores, como matrizes e vetores, possuem grande capacidade de simplificação dos processos empregados. Este tópico é introduzido a seguir, na Seção 1.1.

Dois pontos cruciais na avaliação e seleção de bons modelos é a escolha de métricas de desempenho e de estratégias de validação, pontos elucidados na Seção 1.4.

De um ponto de vista mais prático, o Apêndice B mostra um passo-a-passo básico para a implementação do treinamento de modelos estatísticos supervisionados e não-supervisionados.

## 1.1 Dados como matrizes

O conjunto de observações utilizadas como entradas será aqui notado pela matriz  $\mathbf{X}_{N \times M}$ .

$$\mathbf{X} = \begin{bmatrix} x_{11} & \cdots & x_{1m} & \cdots & x_{1M} \\ & & \vdots & & \\ x_{n1} & \cdots & x_{nm} & \cdots & x_{nM} \\ & & \vdots & & \\ x_{N1} & \cdots & x_{Nm} & \cdots & x_{NM} \end{bmatrix}. \quad (1.1)$$

Cada linha de  $\mathbf{X}$  corresponde a uma amostra com diversos atributos em seus elementos. Por exemplo, cada linha  $n$  pode representar um aluno (amostra), enquanto que cada valor nesta linha representa média em cada disciplina cursada por este aluno (atributos da amostra). Para se referir aos atributos do aluno  $n$  utilizaremos o vetor  $\vec{x}_n$ :

$$\vec{x}_n = [x_{n1} \ \cdots \ x_{nm} \ \cdots \ x_{nM}]^\top, \quad (1.2)$$

de forma que o conjunto de dados  $\mathbf{X}$  pode ser escrito na forma compacta:

$$\mathbf{X} = \begin{bmatrix} \vec{x}_1^\top \\ \vdots \\ \vec{x}_n^\top \\ \vdots \\ \vec{x}_N^\top \end{bmatrix}. \quad (1.3)$$

No caso dos algoritmos supervisionados, cada observação  $\vec{x}_n$  utilizada no treinamento ou validação dos dados se associa a outros  $Q$  atributos que formam um vetor  $\vec{y}_n$ , o qual se relaciona à matriz  $\mathbf{Y}_{N \times Q}$ , tal como  $\vec{x}_n$  se relaciona a  $\mathbf{X}_{N \times M}$ :

$$\mathbf{Y} = \begin{bmatrix} y_{11} & \cdots & y_{1q} & \cdots & y_{1Q} \\ & & \vdots & & \\ y_{n1} & \cdots & y_{nq} & \cdots & y_{nQ} \\ & & \vdots & & \\ y_{N1} & \cdots & y_{Nq} & \cdots & y_{NQ} \end{bmatrix} = \begin{bmatrix} \vec{y}_1^\top \\ \vdots \\ \vec{y}_n^\top \\ \vdots \\ \vec{y}_N^\top \end{bmatrix}, \quad (1.4)$$

onde

$$\vec{y}_n = [y_{n1} \ \cdots \ y_{nq} \ \cdots \ y_{nQ}]^\top. \quad (1.5)$$

De volta ao exemplo,  $\vec{y}_n$  representa o desejo do aluno  $n$  de trabalhar em cada uma das  $Q$  áreas do conhecimento humano. Neste exemplo, o modelo  $\vec{f}(\vec{x})$  é uma função tal que

$$\vec{f}: \mathcal{R}^M \longrightarrow \mathcal{R}^Q$$

$$\vec{x} \longmapsto \vec{f}(\vec{x}) = \vec{y}$$

tem por objetivo estimar  $\vec{y}_n$  a partir de  $\vec{x}_n$ . A saída estimada é notada por  $\vec{\hat{y}}_n$ . A matriz resultante da aplicação de  $\vec{f}$  em cada linha de  $\mathbf{X}$  é chamada  $\hat{\mathbf{Y}}_{N \times Q}$ :

$$\hat{\mathbf{Y}} = \begin{bmatrix} \hat{y}_{11} & \cdots & \hat{y}_{1q} & \cdots & \hat{y}_{1Q} \\ \vdots & & & & \\ \hat{y}_{n1} & \cdots & \hat{y}_{nq} & \cdots & \hat{y}_{nQ} \\ \vdots & & & & \\ \hat{y}_{N1} & \cdots & \hat{y}_{Nq} & \cdots & \hat{y}_{NQ} \end{bmatrix} = \begin{bmatrix} \vec{\hat{y}}_1^\top \\ \vdots \\ \vec{\hat{y}}_n^\top \\ \vdots \\ \vec{\hat{y}}_N^\top \end{bmatrix}, \quad (1.6)$$

Caso  $\vec{f}$  seja um bom preditor, a matriz  $\hat{\mathbf{Y}}$  deve ser uma aproximação de  $\mathbf{Y}$ . Uma forma de mensurar a qualidade de um modelo é através da definição de uma função objetivo, a qual pode ser utilizada mais à frente para guiar suas melhorias. Trataremos deste ponto na seção 1.2.

## 1.2 Função objetivo

O termo função objetivo e se refere a uma função cuja minimização ou maximização implica na obtenção de um modelo mais bem ajustado. Especialmente, as terminologias função *custo* ou *perda* (respectivamente, em inglês, *cost* ou *loss*) são utilizadas para se referir a funções às quais se deseja minimizar, ao passo que funções *aptidão* ou *utilidade* (respectivamente, em inglês, *fitness* ou *utility*) apontam para funções cuja maximização é interessante. Os métodos de otimização mais utilizados são baseados no cálculo do gradiente da função objetivo escolhida, os quais serão descritos na seção 1.3.

Nos problemas supervisionados, a avaliação de erros de predição é definida de forma simples, e como desejamos minimizar erros, estes se tornam a própria função objetivo. Este também é o caso nos métodos não-supervisionados, mas não o é, por exemplo no caso dos métodos de aprendizado por reforço, onde é mais simples avaliar um modelo por sua aptidão a determinado ambiente. Como o enfoque deste trabalho são os métodos supervisionados, aqui será utilizado apenas o termo *função custo* para se referir à função alvo de otimização dos parâmetros do modelo. Entretanto isto não representa perda de generalidade, uma vez que o inverso de uma qualquer função aptidão pode ser tratado como uma função custo.

De longe, a função objetivo mais utilizada em problemas de regressão é o erro quadrático, seja por sua simplicidade, diferenciabilidade, velocidade de processamento ou eficácia. Para uma dada amostra  $\vec{x}_n$ , sua resposta associada,  $\vec{y}_n$ , e estimação de  $\vec{y}$  pelo modelo  $\vec{f}$ ,  $\vec{\hat{y}}_n$ , tal que  $\vec{\hat{y}} = \vec{f}(\vec{x})$ , definimos  $c_n$  como o custo de erro quadrático da seguinte maneira:

$$c_n = \left\| \vec{\hat{y}}_n - \vec{y}_n \right\|_2^2 = \left\| \vec{f}(\vec{x}_n) - \vec{y}_n \right\|_2^2, \quad (1.7)$$

outra função bastante utilizada é o erro absoluto:

$$c_n = \left\| \vec{\hat{y}}_n - \vec{y}_n \right\|_1 = \left\| \vec{f}(\vec{x}_n) - \vec{y}_n \right\|_1. \quad (1.8)$$

É bastante comum que se avalie o erro médio quadrático (*mean squared error* — MSE) em relação a um lote de dados composto por  $\mathbf{X}_{N \times M}$  e  $\mathbf{Y}_{N \times Q}$ :

$$C_{MSE} = \text{MSE}(\hat{\mathbf{Y}}, \mathbf{Y}) = \frac{1}{N} \sum_{n=1}^N \left\| \vec{\hat{y}}_n - \vec{y}_n \right\|_2^2. \quad (1.9)$$

O erro médio quadrático penaliza muito mais fortemente os grandes erros em relação aos erros mais brandos, o que via de regra é desejável. Quando este não é o caso — quando o dado possui muitos valores espúrios, por exemplo —, a função erro absoluto médio (*mean absolute error*, MAE) sobre o lote é bem frutífera:

$$C_{MAE} = \text{MAE}(\hat{\mathbf{Y}}, \mathbf{Y}) = \frac{1}{N} \sum_{n=1}^N \left\| \vec{\hat{y}}_n - \vec{y}_n \right\|_1. \quad (1.10)$$

Em problemas de classificação, o resultado do modelo é tipicamente uma distribuição de probabilidades de pertencimento a cada classe,  $\vec{\hat{y}}_n$ , sendo um vetor de dimensão igual ao número de classes,  $Q$ . Esta deve ser comparada à distribuição real do caso  $n$  do treinamento ou validação,  $\vec{y}_n$  (como a obtida após uma transformação *one-hot*<sup>1</sup> de seus rótulos, que são números inteiros representando as classes às quais a amostra pertence — normalmente uma). O custo  $c_n$  mais utilizado neste caso é a entropia cruzada, que se define como:

$$c_n = - \sum_{q=1}^Q y_{n,q} \log(\hat{y}_{n,q}) = - \sum_{q=1}^Q y_{n,q} \log(\vec{f}(\vec{x}_n)_q). \quad (1.11)$$

Existem muitas outras métricas utilizadas como função custo, cada qual servindo a um objetivo. A citar, pode-se construir funções que descrevem quanto uma previsão do modelo é fisicamente incoerente, como usualmente realizado nos modelos de aprendizado de máquina informados de Física (tradução livre do inglês *Physics-informed machine learning* — PIML). Estes modelos possuem crescente importância nos dias de hoje, uma vez que eles naturalmente possuem maior confiabilidade e capacidade de extração — modelos treinados usando Física na função custo possuem a Física incluída em seus parâmetros, portanto, possuem menos chances de sobreajuste. Um trabalho bastante interessante sobre este tópico aplicado à equação da onda foi escrito por Karimpouli e Tahmasebi (2020).

Além das várias funções objetivo definíveis, é comum hibridizá-las para que a minimização seja multi-objetivo. Assim pode-se compor  $c_n$  como:

$$c_n = \sum_{r=1}^R \lambda_r c_{r,n}, \quad (1.12)$$

onde  $c_{r,n}$  e  $\lambda_r$  ( $r = 1, \dots, R$ ) representam, respectivamente, a função custo  $r$  sobre a observação  $n$  e o peso deste custo na hibridização. Por exemplo, considere um custo dado pela seguinte equação:

$$c_n = \left\| \vec{\hat{y}} - \vec{y} \right\|_2^2 + \lambda \left\| \vec{p} \right\|_1. \quad (1.13)$$

aqui o primeiro termo representa um erro de predição do modelo, enquanto que segundo termo é adicionado para penalizar um vetor de parâmetros muito diferente do vetor de zeros.

---

<sup>1</sup>Ver Apêndice C.

Em outras palavras, o segundo adiciona preferência a modelos com representações esparsas, e é dito um termo de regularização de parâmetros. Observe que o melhor modelo para este segundo custo possui parâmetros  $\vec{p} = \vec{0}$ , e não possui qualquer relação com os dados de treinamento. O custo associado à esparsidade da solução deve ser adicionado, porém, como um termo extra de menor importância. E este é o papel do coeficiente  $\lambda$ : balancear o quanto queremos uma solução que resulte em baixo erro de predição e o quanto queremos que ela seja esparsa. A função custo híbrida descrita em (1.13) é bastante utilizada, visto que modelos esparsos condensam mais informação em menos parâmetros, sendo mais facilmente interpretáveis.

Em geral, no treinamento de um modelo interessa escrever  $c_n$  como função dos parâmetros de  $\vec{f}$ ,  $\vec{p}$ , de forma que se possa mensurar o incremento do erro de predição de um modelo em função da variação de seus parâmetros. Desta forma, fixa-se constante o conjunto de dados  $\mathbf{X}$  e  $\mathbf{Y}$  utilizado, e considera-se que a única variável envolvida no cálculo da função custo escolhida é o vetor de parâmetros. Nestes termos se faz útil definir o custo médio,  $C$ :

$$C(\vec{p}) = \frac{1}{N} \sum_{n=1}^N c_n(\vec{p}). \quad (1.14)$$

Um grande desafio ao buscar mínimos em funções objetivo é que elas podem ser extremamente não-lineares e requerer métodos robustos de otimização para uma travessia eficaz.

Os desenvolvimentos deste capítulo partem do princípio de que os parâmetros dos modelos utilizados se apresentam como um vetor. Entretanto, o caso mais geral é que estes se apresentem como uma sequência de tensores de ordens variadas. Afortunadamente, por meio da operação de vetorialização, pode-se transformar uma sequência de tensores em um único vetor  $\vec{p}$ . Para tal, a operação de vetorialização,  $\text{vec}$ , sobre uma sequência de tensores  $\mathbf{W}_1, \dots, \mathbf{W}_j, \dots, \mathbf{W}_J$  é aqui definida como um vetor cujos elementos são obtidos da concatenação de cada elemento em cada tensor da sequência de parâmetros, de forma ordenada. Assim, se os parâmetros de um modelo são os tensores  $\mathbf{W}_1, \dots, \mathbf{W}_j, \dots, \mathbf{W}_J$ , o vetor de parâmetros  $\vec{p}$  utilizado na otimização pode ser construído como:

$$\vec{p} \equiv \text{vec}(\mathbf{W}_1, \dots, \mathbf{W}_j, \dots, \mathbf{W}_J). \quad (1.15)$$

E como  $\vec{p}$  é construído de forma ordenada, sempre pode-se retornar à descrição original dos parâmetros  $\mathbf{W}_1, \dots, \mathbf{W}_j, \dots, \mathbf{W}_J$  a partir de  $\vec{p}$ , processo aqui notado pela inversa da função  $\text{vec}$ ,  $\text{vec}^{-1}$ . Para evitar excessos de notação, este material notará a função custo sobre um conjunto de parâmetros como a função custo sobre a vetorialização destes parâmetros:

$$C(\mathbf{W}_1, \dots, \mathbf{W}_j, \dots, \mathbf{W}_j) = C(\vec{p}), \quad \vec{p} \equiv \text{vec}(\mathbf{W}_1, \dots, \mathbf{W}_j, \dots, \mathbf{W}_j). \quad (1.16)$$

De forma similar, o gradiente de uma função custo como na equação (1.16) será uma sequencia de tensores de parâmetros, e será notado como:

$$\nabla C(\mathbf{W}_1, \dots, \mathbf{W}_j, \dots, \mathbf{W}_j) \equiv \text{vec}^{-1}(\vec{\nabla} C(\vec{p})), \quad \vec{p} = \text{vec}(\mathbf{W}_1, \dots, \mathbf{W}_j, \dots, \mathbf{W}_j). \quad (1.17)$$

Por fim, a adição ou multiplicação de uma sequência de tensores de parâmetros a outra sequência de tensores com as mesmas dimensões é definida pela adição ou multiplicação ponto-a-ponto de seus elementos. Esta definição generaliza o procedimento de atualização de parâmetros descrita na seção a seguir para o caso dos modelos com múltiplos tensores como parâmetros.

## 1.3 Atualização de parâmetros

Esta seção soluciona a seguinte questão: *como melhorar um modelo a partir da atualização de seus parâmetros?*, partindo da hipótese de que o modelo atual está próximo do melhor modelo possível — ou mais rigorosamente, que o modelo atual se aproxima do mínimo global da função custo escolhida. Supondo que a mesma é diferenciável em relação aos seus  $S$  parâmetros, que constituem o vetor de parâmetros  $\vec{p}$ , podemos expandir polinomialmente o custo calculado após uma breve variação  $\vec{\Delta}p$  nos parâmetros atuais da seguinte maneira:

$$C(\vec{p} + \vec{\Delta}p) = \frac{1}{0!}C(\vec{p}) + \frac{1}{1!}\vec{\Delta}p^\top \vec{\nabla}C(\vec{p}) + \frac{1}{2!}\vec{\Delta}p^\top \nabla^2C(\vec{p})\vec{\Delta}p + \dots \quad (1.18)$$

Esta expansão pode ser truncada nos dois primeiros termos para que se chegue na aproximação a seguir:

$$C(\vec{p} + \vec{\Delta}p) \approx C(\vec{p}) + \vec{\nabla}C(\vec{p}) \cdot \vec{\Delta}p \quad (1.19)$$

Seria interessante para um melhor modelo que o incremento  $\vec{\Delta}p$  aponte para uma direção tal, que  $C(\vec{p} + \vec{\Delta}p)$  seja menor que  $C(\vec{p})$ . Neste caso, temos que o termo  $\vec{\nabla}C(\vec{p}) \cdot \vec{\Delta}p$  deve ser negativo em (1.19). Tem-se pela definição do produto interno:

$$\vec{\nabla}C(\vec{p}) \cdot \vec{\Delta}p = \|\vec{\nabla}C(\vec{p})\|_2 \|\vec{\Delta}p\|_2 \cos(\vec{\nabla}C(\vec{p}), \vec{\Delta}p). \quad (1.20)$$

Considerando os limites inferior e superior da função cosseno, pode-se ver uma desigualdade surgir:

$$-\|\vec{\nabla}C(\vec{p})\|_2 \|\vec{\Delta}p\|_2 \leq \vec{\nabla}C(\vec{p}) \cdot \vec{\Delta}p \leq \|\vec{\nabla}C(\vec{p})\|_2 \|\vec{\Delta}p\|_2. \quad (1.21)$$

Com isto, concluímos que o menor valor possível para o termo  $\vec{\nabla}C(\vec{p}) \cdot \vec{\Delta}p$  em (1.19) ocorre quando  $\vec{\nabla}C(\vec{p})$  e  $\vec{\Delta}p$  possuem direções opostas — caso em que o cosseno entre estes vetores vale  $-1$ .

Este resultado é a essência dos métodos mais famosos de otimização baseados no gradiente da função custo, alguns dos quais serão detalhados nas subseções a seguir. Vale aqui salientar que a utilização destes métodos em geral não leva ao mínimo global da função custo num único passo, mas iterativamente.

### 1.3.1 Máximo declive

O *método do máximo declive*, ou ainda *método gradiente*, foi descrito detalhadamente pela primeira vez na academia por Curry (1944). Este método se caracteriza por atualizar os parâmetros do modelo atual na direção oposta do vetor gradiente da função custo — portanto na direção inversa daquela em que o custo mais aumenta. O enunciado das direções opostas se traduz na desigualdade entre os respectivos versores dos vetores envolvidos:

$$\frac{\vec{\Delta}p}{\|\vec{\Delta}p\|_2} = -\frac{\vec{\nabla}C(\vec{p})}{\|\vec{\nabla}C(\vec{p})\|_2} \quad (1.22)$$

onde se segue que:

$$\vec{\Delta}p = -\frac{\|\vec{\nabla}C(\vec{p})\|_2}{\|\vec{\nabla}C(\vec{p})\|_2} \vec{\nabla}C(\vec{p}) = -\eta \vec{\nabla}C(\vec{p}), \quad (1.23)$$

onde  $\eta = \frac{\|\vec{\Delta}p\|_2}{\|\vec{\nabla}C(\vec{p})\|_2}$  é um escalar que dita a distância caminhada pelos parâmetros na direção oposta do gradiente. Como  $\vec{\Delta}p$  apenas obedece a restrição de ser pequeno o suficiente para que a expansão polinomial truncada em (1.19) continue válida,  $\eta$  é um parâmetro de otimização arbitrário, que usualmente é escolhido por tentativa e erro.

A atualização dos parâmetros pelo método do máximo declive se dá então, em cada iteração  $i$ , por:

$$\vec{p}_{i+1} = \vec{p}_i + \vec{\Delta}p_i. \quad (1.24)$$

### 1.3.2 Otimização estocástica e o máximo declive estocástico (SGD)

Em diversas situações a matriz de observações possui um grande volume de amostras. Por consequência, o cálculo da função custo é demorado. Assim a taxa de atualização do modelo estatístico em treinamento é prejudicada, e em última instância, sua convergência pode ser retardada. Nestas situações, pode ser mais vantajoso calcular a função custo em lotes (Wardi, 1988).

A otimização estocástica se baseia na repartição das  $N$  amostras de treinamento em  $\mathbf{X}$  e  $\mathbf{Y}$ , anteriormente associadas a uma única função custo  $C(\vec{p}, \mathbf{X}, \mathbf{Y})$ , entre pequenos lotes,  $\mathbf{X}_b$  e  $\mathbf{Y}_b$ , com  $b = 1, \dots, B$ , os quais são associados a funções custo individuais,  $C_b(\vec{p}, \mathbf{X}_b, \mathbf{Y}_b)$ . Desta forma, pode-se compor  $C$  como:

$$C(\vec{p}) = \frac{1}{B} \sum_{b=1}^B C_b(\vec{p}) \quad (1.25)$$

e seu gradiente, pela regra da cadeia, como:

$$\vec{\nabla}C(\vec{p}) = \frac{1}{B} \sum_{b=1}^B \vec{\nabla}C_b(\vec{p}), \quad (1.26)$$

onde cada gradiente  $\vec{\nabla}C_b(\vec{p})$  pode ser calculado de forma rápida e independente.

Resta então discutir como se dá o treinamento do modelo. Em cada iteração de otimização realiza-se para cada lote, de forma aleatória, o cálculo do gradiente de sua função custo em relação aos parâmetros e com base nele o modelo é atualizado. Assim, o método estocástico multiplica por  $B$  o número de atualizações do modelo por iteração de otimização.

Em certos casos, utilizar a estratégia de lotes pode levar a um processo instável, principalmente quando estes possuem poucas amostras. Nestes casos, pode-se utilizar o método estocástico com acumulação de gradientes, outro modelo de otimização estocástica. Esta se beneficia apenas da menor exigência computacional do método estocástico para calcular os gradientes de cada lote, os quais são combinados ao final de cada iteração (1.26) para obter o gradiente da função custo total (Robbins e Monro, 1951; Bottou et al., 2018).

O método do máximo declive estocástico (do inglês *stochastic gradient descent*, SGD) é nada mais que a extensão estocástica do método do máximo declive. É fácil perceber que o SGD é uma generalização do método do máximo declive, pois aquele recai neste quando  $B = 1$ .

Apesar de o algoritmo considerar apenas rodadas incontáveis de lotes aleatórios, é comum impedir a reutilização de lotes antes até que os outros já terem sido utilizados. Neste caso, uma época é definida como um conjunto de operações de otimização estocástica na

---

**Algoritmo 1:** SGD - Máximo declive estocástico
 

---

**Entrada:**  $\vec{p}_0$ : parâmetros iniciais do modelo

**Entrada:**  $\eta > 0$ : passo (taxa de aprendizado)

**Saída:**  $\vec{p}_I$ : parâmetros do modelo treinado

```

 $i \leftarrow 0$  enquanto  $\vec{p}_i$  não convergiu faça
     $i \leftarrow i + 1$ 
     $b \leftarrow$  inteiro aleatório  $\in [1, B]$                                  $\triangleright B$ : número de lotes
     $\vec{g}_i \leftarrow \vec{\nabla}C_b(\vec{p}_{i-1})$ 
     $\vec{\Delta p}_i \leftarrow -\eta \vec{g}_i$ 
     $\vec{p}_i \leftarrow \vec{p}_{i-1} \vec{\Delta p}_i$ 
fim
 $\vec{p}_I = \vec{p}_i$ 
```

---

qual não há repetição de lotes. Este conceito é utilizado generalizadamente entre os métodos estocásticos, incluindo o SGD, o Momento e Adam — estes dois apresentados nas próximas subseções.

### 1.3.3 Momento

Os psicólogos David Rumelhart e Geoffrey Hinton, juntamente a Ronald Williams publicaram em 1986 um artigo chamado *Learning representations by back-propagating errors* — em português, “aprendendo representações por retropropagação dos erros” —, no qual o método de otimização Momento é apresentado. Sua ideia é aproveitar parte das informações da iteração de otimização anterior para produzir a nova atualização de parâmetros. O método se inicia por uma primeira iteração do método de máximo declive, na qual obtém a variação de parâmetros  $\vec{\Delta p} = -\eta \vec{\nabla}C(\vec{p})$  e o modelo é atualizado como usual. Da segunda iteração em diante, a atualização de parâmetros é escrita como uma combinação linear entre a atualização passada e o gradiente da função custo, como avaliada no conjunto de parâmetros da iteração atual:

$$\vec{\Delta p}_{i+1} = \alpha \vec{\Delta p}_i - \eta \vec{\nabla}C_i(\vec{p}). \quad (1.27)$$

É importante observar que o novo parâmetro do processo de otimização,  $\alpha$ , compõe um termo exponencial. Isto ocorre, pois a variável  $\vec{\Delta p}$  é definida com base em seu próprio valor na iteração anterior, o qual é composto pelo coeficiente  $\alpha$ . Neste esquema,  $\alpha$  é escolhido no intervalo  $[0, 1]$ . Os autores nomeiam tal parâmetro “fator de decaimento exponencial”, uma vez que ele atenua  $\vec{p}_i$  continuamente em seu produto. Desta forma, quanto maior seu valor, mais os resultados passados pesam sobre a escolha da trajetória percorrida pelos parâmetros na otimização do modelo em questão, ou em outros termos, maior a inércia do movimento dos parâmetros. No caso limite em que  $\alpha = 0$ , retornamos ao método de máximo declive. Apesar de não ser inicialmente intuitivo, a prática mostra que a utilização de informações passadas na atualização de parâmetros é bastante eficaz para uma convergência mais rápida e estável. O Algoritmo 2 descreve o procedimento para otimização estocástica utilizando Momento.

---

**Algoritmo 2:** Momento
 

---

**Entrada:**  $\vec{p}_0$ : parâmetros iniciais do modelo

**Entrada:**  $\eta > 0$ : passo (taxa de aprendizado)

**Entrada:**  $\alpha \in [0, 1]$ : taxa de decaimento exponencial

**Saída:**  $\vec{p}_I$ : parâmetros do modelo treinado

$$\vec{\Delta p}_0 \leftarrow \vec{0}$$

$$i \leftarrow 0$$

**enquanto**  $\vec{p}_i$  não convergiu **faça**

$$i \leftarrow i + 1$$

$$b \leftarrow \text{inteiro aleatório } \in [1, B]$$

▷ B: número de lotes

$$\vec{g}_i \leftarrow \vec{\nabla} C_b(\vec{p}_{i-1})$$

$$\vec{\Delta p}_i \leftarrow \alpha \vec{\Delta p}_{i-1} - \eta \vec{g}_i$$

$$\vec{p}_i \leftarrow \vec{p}_{i-1} \vec{\Delta p}_i$$

**fim**

$$\vec{p}_I = \vec{p}_i$$


---

### 1.3.4 Estimativa adaptável de momento (Adam)

Apesar de bastante recente, o método de estimativa adaptável de momento — *Adaptive moment estimation*, Adam — é o mais utilizado no treinamento de redes neurais atualmente. O método foi introduzido no artigo de Kingma e Ba (2014), e tenta unir ideias de outros dois algoritmos de otimização, a citar: AdaGrad (Duchi et al., 2011) e RMSProp (Tieleman e Hinton, 2012). Seu algoritmo é desenvolvido sobre a hipótese de que a função objetivo estudada é ruidosa, seja por problemas intrínsecos à mesma, seja por a otimização se dar em lotes, e envolve guardar médias móveis do gradiente e do quadrado do gradiente da função custo ao longo das últimas iterações para estimar o melhor percurso a seguir no espaço dos modelos, iterativamente. Estas duas medidas guardadas representam os primeiro e segundo momentos centrais do gradiente, ou seja, suas média e variância ponto-a-ponto ao longo das últimas iterações. Tecnicamente, ao invés da variância, estima-se o segundo momento do gradiente, que é sua variância em torno de zero.

O método possui dois parâmetros de decaimento no intervalo  $[0, 1]$  além da taxa de aprendizado,  $\eta$ . Eles são  $\beta_1$  e  $\beta_2$ , os quais definem as taxas de decaimento exponencial dos primeiro e segundo momentos do gradiente, respectivamente. De forma simplificada,  $\beta_1$  dita a importância da média dos gradientes passados na predição do próximo passo de otimização,  $\vec{\Delta p}$ , enquanto  $\beta_2$ , a de suas variâncias.

Como as médias móveis são inicializadas nulas, as estimativas dos momentos é inicialmente enviesada. Para reduzir o viés nos valores estimados, devemos dividi-las por  $(1 - \beta^i)$  (Kingma e Ba, 2014), onde  $\beta$  representa a taxa de decaimento associada ao momento em questão e  $i$  é o número da iteração de otimização.

O Algoritmo 3 descreve o funcionamento do Adam. Os parâmetros sugeridos por Kingma e Ba (2014) são:  $\eta = 0,001$ ,  $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$  e  $\epsilon = 10^{-8}$ . Caso  $\beta_1 = \beta_2 = 0$ , retorna-se ao método gradiente estocástico, entretanto o passo tomado tem intensidade igual a  $\eta$  em cada dimensão do espaço de modelos.

---

**Algoritmo 3:** Adam - Estimativa adaptável de momento
 

---

**Entrada:**  $\vec{p}_0$ : parâmetros iniciais do modelo  
**Entrada:**  $\eta > 0$ : passo (taxa de aprendizado)  
**Entrada:**  $\beta_1 \in [0, 1[$ : taxa de decaimento exponencial do primeiro momento  
**Entrada:**  $\beta_2 \in [0, 1[$ : taxa de decaimento exponencial do segundo momento  
**Entrada:**  $\epsilon \ll 1$ : valor de segurança para prevenir divisão por 0  
**Saída:**  $\vec{p}_I$ : parâmetros do modelo treinado

```

 $\vec{m}_0 \leftarrow \vec{0}$ 
 $\vec{v}_0 \leftarrow \vec{0}$ 
 $i \leftarrow 0$ 
enquanto  $\vec{p}_i$  não convergiu faça
     $i \leftarrow i + 1$ 
     $b \leftarrow$  inteiro aleatório  $\in [1, B]$  ▷ B: número de lotes
     $\vec{g}_i \leftarrow \vec{\nabla} C_b(\vec{p}_{i-1})$ 
     $\vec{m}_i \leftarrow \beta_1 m_{i-1} + (1 - \beta_1) \vec{g}_i$ 
     $\vec{v}_i \leftarrow \beta_2 v_{i-1} + (1 - \beta_2) \vec{g}_i^2$ 
     $\hat{m}_i \leftarrow m_i / (1 - \beta_1^i)$ 
     $\hat{v}_i \leftarrow v_i / (1 - \beta_2^i)$ 
     $\vec{p}_i \leftarrow \vec{p}_{i-1} + \alpha \frac{\hat{m}_i}{\sqrt{\hat{v}_i} + \epsilon}$ 
fim
 $\vec{p}_I = \vec{p}_i$ 

```

---

## 1.4 Validação estatística, viés e variância de modelos

Uma anedota: você projetou um modelo com o objetivo de estimar uma matriz de alvos  $\mathbf{Y}$  a partir de uma matriz de observações  $\mathbf{X}$ . Você construiu estas matrizes a partir de dados previamente adquiridos, e utilizou um método baseado em gradiente para treinar seu modelo nelas. Após certa quantidade de iterações, seus parâmetros convergem para uma solução, e então você sente que é capaz de utilizá-lo para prever matrizes similares a  $\mathbf{Y}$  para qualquer matriz construída de forma similar a  $\mathbf{X}$  de modo confiável. Você faz um esforço para validar seu modelo por meio de uma métrica, como a raíz do erro médio quadrático (RMSE) ou a média dos desvios absolutos (MAE), e como resultado obtém valores impressionantes. Já seguro sobre o método que projetou, aplica-o assim que surge uma oportunidade visando obter a desejada matriz de quantidades  $\mathbf{Y}$  para o caso em questão. Infelizmente veio o resultado: seu modelo é completamente insatisfatório na predição de novos dados.

O que ocorreu é um conflito de interesses entre o que se objetiva e como se definiu matematicamente a função objetivo do problema. Em geral, nosso objetivo é obter um modelo que consiga generalizar as relações estudadas mesmo para observações nas quais ele não foi treinado com a maior exatidão possível. Entretanto, o que escrevemos na função custo é, por exemplo, que o erro de predição do modelo diminua sobre os dados, e assim iniciamos o processo de otimização. Desta forma, seu modelo inicia o aprendizado geral do fenômeno, mas em determinado momento passa a tentar memorizar as saídas dos dados de treinamento sempre que necessário para diminuir a função custo. Logo em seguida ele aprende a reproduzir inclusive os ruídos de aquisição presentes nos dados.

A saída para este problema é escolher uma estratégia de validação cruzada. O esquema mais simples é o método *holdout*. Nele as observações e quantidades adquiridas são utilizadas

na composição de dois lotes: um lote  $a$  composto por  $\mathbf{X}_a$  e  $\mathbf{Y}_a$  utilizado no treinamento dos dados, e outro,  $b$ , composto por  $\mathbf{X}_b$  e  $\mathbf{Y}_b$ , que serão utilizados na validação do modelo. Deste modo, ao fim do treinamento usando o lote  $a$ , pode-se verificar as métricas desejadas sobre previsões em dados novos controlados, e ter noção do grau de confiança que devemos assumir em aplicações reais.

Os conceitos de viés e variância de um modelo são essenciais para o estudo do aprendizado de modelos. O *viés* de um modelo se refere à sua incapacidade de capturar o comportamento do fenômeno estudado. Em outras palavras, o quanto pouco ele consegue ser "dobrado" ou ajustado para se adequar ao conjunto de dados<sup>2</sup>. Ele está relacionado de forma inversa ao grau de liberdade e ao número de parâmetros deste modelo. O viés pode ser avaliado já sobre os dados de treinamento. Desta forma, se após o treinamento, um modelo responde com um baixo valor de erro (ou alta exatidão) sobre os dados em que foi treinado (lote  $a$ ), ele possui um baixo viés. Se o erro é alto, o viés também o é, e uma adição ao seu número de parâmetros tenderá a melhorá-lo. A interpretação do viés como alto ou baixo depende das necessidades de exatidão nas aplicações do modelo. Já a *variância* de um modelo representa a sensibilidade da qualidade de sua resposta quando exposto a diferentes conjuntos de dados. Na prática, a variância pode ser mensurada como a diferença de métricas obtidas sobre os dados de treinamento e os de validação. Se o viés representa a dificuldade de um modelo se ajustar ao conjunto de dados de treinamento, a variância é uma medida de quanto realísticos foram os ajustes do treinamento, considerando a aplicação em dados futuros. Enquanto um maior número de parâmetros tende a gerar modelos pouco enviesados, um excesso de parâmetros tende a aumentar a variância, tornando o modelo inválido a novas aplicações. O estudioso de modelos deve dosar estas métricas de acordo com as necessidades do projeto em que trabalha.

O fenômeno descrito na anedota contada é caracterizado por um modelo com baixo viés e alta variância, situação chamada de sobreajuste (do inglês, *overfit*). Este caso tem tanto mais possibilidade de acontecer quanto maior for o número de parâmetros do modelo utilizado, e quanto menor for o número de dados representativos disponíveis. Por outro lado, seu inverso é o fenômeno do subajuste (do inglês, *underfit*). Neste segundo caso, o modelo é muito pouco maleável aos dados do problema, não permitindo alta extatidão mesmo nos dados em que foi treinado. Assim os resultados geralmente são ruins tanto nos dados de treinamento, quanto nos de validação, o que caracteriza um alto viés e uma baixa variância.

Assim, possuímos uma técnica para avaliar se um modelo foi sobreajustado no treinamento ou não. Mas como evitar que isso venha a ocorrer, desde o princípio? Este é o tópico da próxima seção.

## 1.5 Interrupção precoce

Como explicado na Seção 1.4, na validação *holdout* realiza-se uma separação dos dados entre os lotes de treinamento,  $a$ , e validação,  $b$ . A resposta da validação cruzada ao final do treinamento nos diz se houve sobreajuste e, portanto, se possuímos o suficiente para descartá-lo — o que acontece frequentemente. Ao invés disso, caso validássemos o modelo a cada etapa de otimização e fizéssemos um gráfico das métricas de desempenho ao longo

---

<sup>2</sup>A etimologia de viés pode gerar confusão inicial, uma vez que enviesamento vem de inclinação. Desta forma, o leitor pode acabar acreditando que um modelo é enviesado se ele se inclinou bem aos dados de treinamento. *Isto não procede*: o termo se refere a uma inclinação intrínseca do modelo, que o impede de ser remoldado durante treinamento.

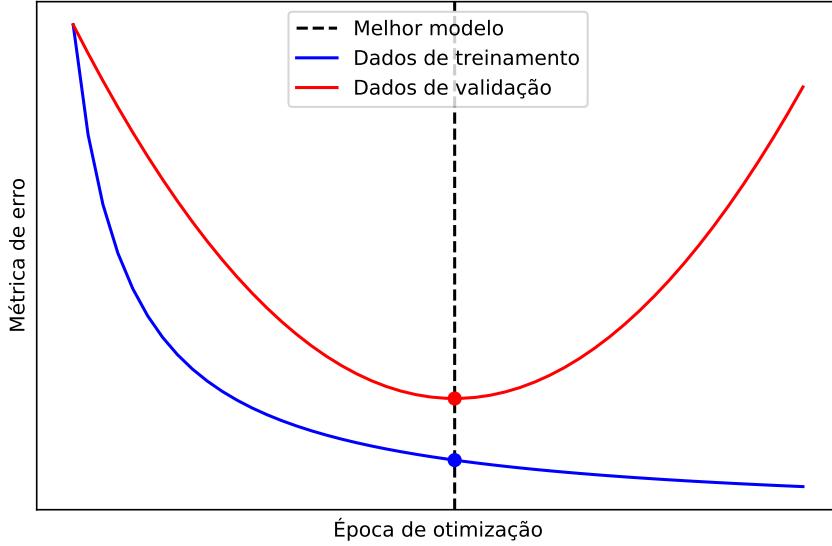


Figura 1.1: Curvas representam métricas de erro avaliadas sobre os dados de treinamento (em azul) e de validação (em vermelho) ao longo das épocas de otimização. O melhor modelo se localiza onde a curva de validação atinge seu menor valor, ponto dado pela linha tracejada.

destas iterações, veríamos algo como o presente na Figura 1.1<sup>3</sup>. Pode-se perceber que as métricas de otimização nos dados de treinamento melhoraram (o erro de predição cai) com as iterações, tendendo a um platô de forma assintótica. É neste ponto que podemos considerar a convergência do modelo com relação à função custo utilizada. As métricas nos dados de validação, por outro lado, são inicialmente próximas às obtidas nos dados de treinamento, mas logo começam a se distanciar. Em uma dada iteração de treinamento, as métricas obtidas nos dados de validação atingem um mínimo, e posteriormente começam a degradar (subir, no caso de uma métrica de erro). O modelo mais interessante para aplicações futuras é aquele que desempenha melhor nos dados de validação, mesmo não sendo treinado neles. As curvas da Figura 1.1 são esquemáticas: na prática elas costumam ser mais caóticas, então é difícil perceber o momento exato em que a métrica dos dados de validação começa a divergir. Muitas vezes, principalmente na etapa inicial do treinamento, estas métricas podem aumentar por algumas épocas antes de voltarem a diminuir ainda mais intensamente. O método da interrupção precoce é uma tentativa de cessar o treinamento antes que a degradação das métricas de validação ocorra.

A forma mais simples de interrupção precoce envolve guardar o vetor de parâmetros  $\vec{p}$  que resulta na melhor métrica das últimas iterações de otimização, o qual deve ser atualizado sempre que um passo de otimização resultar na melhora histórica da métrica por uma variação mínima necessária notada por  $\Delta m_0 \geq 0$ . Caso contrário, inicia-se um contador para quantificar o número de iterações desde que o melhor modelo foi conseguido. Caso este número se torne superior a um valor  $I_p \geq 0$  chamado de paciência, deve-se parar a otimização

<sup>3</sup>A rigor, o conjunto de dados é dividido em três partes: um conjunto de treinamento, um de teste, que é ser avaliado a cada época para validação e interrupção precoce durante o treinamento, e um de validação, que de fato se refere ao teste final de validação do modelo. Neste trabalho, o conjunto de teste será o próprio conjunto de validação final.

e retornar os melhores parâmetros guardados como resultado do treinamento. Quanto maior o valor da paciência, maior a tolerância do treinamento ao caos da convergência estocástica. Com este processo, reduzimos a possibilidade de descarte de um modelo quando isto não é necessário.

# Capítulo 2

## Cálculo do gradiente

Até o momento, sabemos construir uma função custo  $C(\vec{p})$  para treinar um modelo (Seção 1.2), bem como melhorá-lo a partir dela e de métodos de otimização local baseados em gradiente (Seção 1.3). Neste capítulo serão discutidas as maneiras pelas quais estes gradientes podem ser calculados, incluindo considerações teóricas e práticas.

De forma introdutória, o gradiente de uma função real  $f$  em  $\vec{p} \in \mathcal{R}^S$ , notado por  $\vec{\nabla}f(\vec{p})$ , é definido matematicamente pelo vetor de derivadas parciais desta função em relação a  $\vec{p}$ :

$$\nabla f(\vec{p}) = \frac{\partial f}{\partial \vec{p}} = \begin{bmatrix} \frac{\partial f}{\partial p_1}(\vec{p}) \\ \vdots \\ \frac{\partial f}{\partial p_s}(\vec{p}) \\ \vdots \\ \frac{\partial f}{\partial p_S}(\vec{p}) \end{bmatrix}. \quad (2.1)$$

Por sua vez, suas derivadas parciais são calculadas a partir do limite:

$$\frac{\partial f}{\partial p_s}(\vec{p}) = \lim_{h \rightarrow 0} \frac{f(\vec{p} + h\vec{e}_s) - f(\vec{p})}{h}, \quad (2.2)$$

onde  $h \in \mathcal{R}$  e  $\vec{e}_s$  é o vetor unitário positivo que aponta na direção da dimensão  $s$  de  $\mathcal{R}^S$ .

Quando a função diferenciada passa a ser vetorial,  $\vec{f} \in \mathcal{R}^Q$ , faz mais sentido falar no termo matriz jacobiana (ou simplesmente jacobiana), que agrupa em linhas as transpostas dos gradientes de cada elemento de  $\vec{f}$ , notada por  $J_{\vec{f}}(\vec{p})$ :

$$J_{\vec{f}}(\vec{p}) = \frac{\partial \vec{f}}{\partial \vec{p}} = \left[ \frac{\partial \vec{f}}{\partial p_1} \quad \dots \quad \frac{\partial \vec{f}}{\partial p_s} \quad \dots \quad \frac{\partial \vec{f}}{\partial p_S} \right] = \begin{bmatrix} \vec{\nabla}^\top f_1(\vec{p}) \\ \vdots \\ \vec{\nabla}^\top f_q(\vec{p}) \\ \vdots \\ \vec{\nabla}^\top f_Q(\vec{p}) \end{bmatrix}, \quad (2.3)$$

ou de forma explícita:

$$J_f(\vec{p}) = \begin{bmatrix} \frac{\partial f_1}{\partial p_1}(\vec{p}) & \cdots & \frac{\partial f_1}{\partial p_s}(\vec{p}) & \cdots & \frac{\partial f_1}{\partial p_S}(\vec{p}) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \frac{\partial f_q}{\partial p_1}(\vec{p}) & \cdots & \frac{\partial f_q}{\partial p_s}(\vec{p}) & \cdots & \frac{\partial f_q}{\partial p_S}(\vec{p}) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \frac{\partial f_Q}{\partial p_1}(\vec{p}) & \cdots & \frac{\partial f_Q}{\partial p_s}(\vec{p}) & \cdots & \frac{\partial f_Q}{\partial p_S}(\vec{p}) \end{bmatrix} \quad (2.4)$$

Desta forma, a jacobiana de uma função real é a transposta de seu gradiente. A montagem computacional da jacobiana, de modo geral, exige estratégias como as que serão alvo das seções a seguir.

## 2.1 Diferenciação manual

O método de diferenciação manual é o método básico de diferenciação. Assumindo que  $\vec{f}$  é diferenciável em um domínio  $\mathcal{D} \subset \mathbb{R}^S$ , utiliza-se a definição de limite e suas propriedades para compor a função jacobiana, que leva cada elemento  $\vec{p}$  em  $\mathcal{D}$  a  $J_{\vec{f}}(\vec{p})$ . A principal vantagem deste método é que uma vez que se possua a função jacobiana, pode-se estimar matrizes jacobianas em qualquer ponto em  $\mathcal{D}$  de forma exata e eficiente. Por outro lado, a diferenciação manual exige que se possua uma equação em fórmula fechada, isto é, que as operações sejam as funções básicas da matemática e que estas sejam finitas em número. Para além destas limitações, existem funções muito difíceis ou até mesmo impossíveis de ser diferenciadas dadas as tecnologias matemáticas atuais. Outro ponto bastante negativo do método é a inescalabilidade computacional: a cada novo problema a ser diferenciado, alguém deve dedicar um intervalo de tempo potencialmente elevado na formulação da função jacobiana, para só então computar seus resultados.

## 2.2 Diferenciação simbólica

A diferenciação simbólica consiste em representar, em linguagem computacional, conceitos abstratos como o de símbolos e expressões matemáticas, de forma que a manipulação dos mesmos possa resultar na solução de problemas diferenciais de forma exata (Pavelle et al., 1981). Por meio da computação simbólica, pode-se representar uma função de forma fechada  $\vec{f}(\vec{p})$  como uma expressão matemática e utilizar da manipulação simbólica automatizada para calcular a expressão que representa a função jacobiana dentro do domínio desejado. Este método supera a diferenciação manual, uma vez que pode produzir soluções complicadas em pouco tempo, além de ser facilmente aplicado a outros problemas realizando o mesmo processo. Entretanto, como dito, o método é limitado a expressões fechadas, o que muitas vezes não é o caso.

## 2.3 Diferenciação numérica

A diferenciação numérica diverge das técnicas manual e simbólica por não buscar respostas exatas, mas aproximações. Para tal, cada derivada parcial da matriz jacobiana é aproximada no ponto de aplicação  $\vec{p}$  de interesse. No caso do método das diferenças finitas, as derivadas parciais da forma explicitada na equação (2.2) são aproximadas por:

$$\frac{\partial \vec{f}}{\partial p_s}(\vec{p}) = \lim_{h \rightarrow 0} \frac{\vec{f}(\vec{p} + h\vec{e}_s) - \vec{f}(\vec{p})}{h} \approx \frac{\vec{f}(\vec{p} + h\vec{e}_s) - \vec{f}(\vec{p})}{h} \quad (2.5)$$

onde  $h$  na aproximação deixa de ser uma quantidade infinitesimal para se tornar um número muito pequeno (*i.e.*,  $h = 0,01$ ).

A vantagem da diferenciação numérica é que ela é generalizável a qualquer expressão, mesmo que ela não seja de forma fechada. A principal desvantagem é que para calcular cada derivada, deve-se aplicar a função  $\vec{f}$  uma vez durante a variação de cada um dos  $S$  parâmetros de  $\vec{p}$ . Desta maneira, se  $\vec{p}$  represente uma malha de propriedades físicas de dimensão  $n_x \times n_y = 1000 \times 1000 = 1.000.000$ , serão necessários um milhão de aplicações diretas de  $\vec{f}$  além de sua aplicação em  $\vec{p}$ , para o cálculo da jacobiana, o que a depender da operação realizada (*i.e.*, simulação física) pode se tornar inviável. Assim, a diferenciação numérica é de aplicação geral, mas ineficiente.

## 2.4 Diferenciação automática (autodiff)

A diferenciação automática (AD), também referida como autodiferenciação ou simplesmente autodiff, consegue um grande número de troféus dentre os métodos de diferenciação descritos neste capítulo. Seus resultados são exatos, eficientes, não exigem forma fechada da função a ser diferenciada, e uma vez implementada, é de aplicação a qualquer outro problema (Baydin et al., 2018). Sua desvantagem é não necessariamente produzir uma função fechada como resultado, o que é um problema apenas se a manipulação simbólica da matriz jacobiana é desejada. Neste caso, se a função utilizada tiver uma forma fechada, o tratamento simbólico sobre seu gráfico é capaz de gerar as derivadas parciais em forma fechada (*i.e.*, é o caso realizado no Apêndice A). Este esquema de diferenciação é generalizadamente utilizado no treinamento de modelos de aprendizado de máquina, principalmente no caso das redes neurais artificiais.

O coração da autodiff é a ideia de que todas as computações feitas são compostas de operações mais básicas, as quais conhecemos as regras de derivação (Deisenroth et al., 2020). Assim, conhecendo o grafo que leva as entradas às saídas de uma função genérica  $\vec{f}(\vec{p})$  pode-se calcular suas derivadas por simples aplicação da regra da cadeia. Esta pode ser realizada em concordância com a ordem de execução do grafo que liga  $\vec{p}$  a  $\vec{f}(\vec{p})$ , o que define a diferenciação automática direta, ou no sentido oposto, partindo das saídas de  $\vec{f}(\vec{p})$  às suas entradas,  $\vec{p}$ . Este último método é denominado diferenciação automática reversa (Baydin et al., 2018).

Algumas ideias comuns aos dois métodos de diferenciação automática são as seguintes:

1. A maior parte das computações pode ser escrita como um grafo que liga variáveis por meio de composições de operações mais básicas e com derivadas bem conhecidas.
2. Ao invés de números, o fluxo pelo grafo de qualquer computação é realizado por meio de pares de valores primários e diferenciais — sejam tangentes ou adjuntos.
3. Variáveis e constantes só diferem por as últimas serem inicializadas com diferenciais nulas no antes da diferenciação (a segunda etapa, no caso da diferenciação reversa).
4. A diferenciação parcial com relação a uma variável envolve tornar as outras variáveis do problema constantes (tornar suas diferenciais nulas).

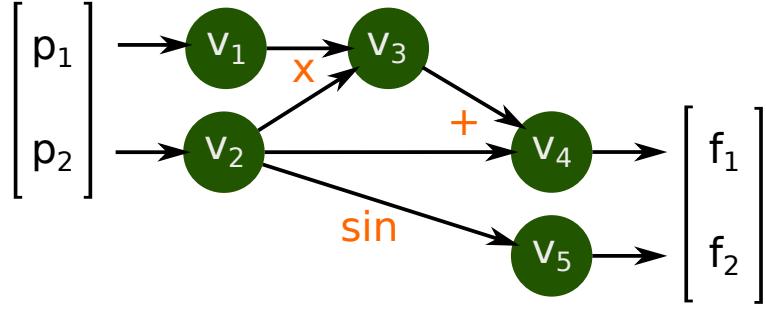


Figura 2.1: Grafo da função  $\vec{f}(\vec{p}) = [(p_2 + p_1 p_2) \ (\sin p_2)]^\top = [f_1 \ f_2]^\top$ .

### 2.4.1 Diferenciação automática direta

Considere a seguinte função:

$$\vec{f}(\vec{p}) = \vec{f}\left(\begin{bmatrix} p_1 \\ p_2 \end{bmatrix}\right) = \begin{bmatrix} p_2 + p_1 p_2 \\ \sin(p_2) \end{bmatrix} = \begin{bmatrix} f_1(\vec{p}) \\ f_2(\vec{p}) \end{bmatrix}. \quad (2.6)$$

O grafo de  $\vec{f}(\vec{p})$  é desenhado na Figura 2.1. Nela, cada círculo representa um nó, com uma variável composta por uma dupla de valores primário e tangente — por simplicidade, apenas o valor primário está sendo representado. As setas indicam que  $v_3$  pode ser calculado apenas com o conhecimento de  $v_1$  e  $v_2$  por meio da operação produto. Estes, por sua vez, dependem ou não das entradas  $\vec{p}$  da função  $\vec{f}$ .

As tangentes são as diferenciais parciais de cada nó do grafo de  $\vec{f}$  em relação à entrada à qual se deseja diferenciar, obtidas pela aplicação da regra da cadeia. A regra da cadeia é sempre calculada em partes na diferenciação automática, e é aí que mora sua capacidade de diferenciar com exatidão algoritmos complicados. Assim, caso desejemos diferenciar  $f(\vec{p})$  em relação a  $p_1$  no ponto  $\vec{p} = [2 \ 2]^\top$  por autodiff direta, devemos percorrer o grafo na Figura 2.1 em seu sentido natural, enquanto calculamos os valores primários e tangentes associados a cada nó até as saídas da função  $\vec{f}$ . As tangentes de cada nó são calculadas por aplicação da regra da cadeia em relação aos seus nós de dependência direta, e estes resultados são acumulados para o cálculo dos nós seguintes. Vale definir a tangente do nó  $v_i$ ,  $\dot{v}_i$ , como a diferencial de  $v_i$  em relação à entrada  $p_s$ :

$$\dot{v}_i = \frac{\partial v_i}{\partial p_s}, \quad (2.7)$$

onde  $\dot{v}_i$  é calculada sobre as diferenciais acumuladas nos nós passados. Por exemplo, para calcularmos  $\dot{v}_3$  quando diferenciamos em relação à entrada  $p_1$ , não precisaremos explicitar toda a expressão  $v_3$ , mas escrever sua derivada em função de suas dependências bem como suas respectivas tangentes. Por regra da cadeia, fazemos:

$$\dot{v}_3 \equiv \frac{\partial v_3}{\partial p_1} = \frac{\partial v_3}{\partial p_1} v_2 + v_1 \frac{\partial v_2}{\partial p_1} = \dot{v}_1 v_2 + v_1 \dot{v}_2, \quad (2.8)$$

onde  $\dot{v}_1$  é  $\frac{\partial v_1}{\partial p_1} = 1$ , pois  $v_1 = p_1$ , e  $\dot{v}_2 = \frac{\partial v_2}{\partial p_1} = 0$ , pois  $v_2$  não depende de  $p_1$ . Todos os passos da autodiferenciação direta do grafo em discussão são descritos na Tabela 2.1, onde os nós são calculados por ordem de necessidade. Como realizado aqui, a ideia da diferenciação automática é calcular a derivada parcial passo a passo, por aplicação recorrente da regra

Passo	Primários		Tangentes	
	Nós de entrada		Nós de entrada	
1	$v_1 = p_1$	= 2	$\dot{v}_1 = \frac{\partial v_1}{\partial p_1}$	= 1
2	$v_2 = p_2$	= 2	$\dot{v}_2 = \frac{\partial v_2}{\partial p_1}$	= 0
	Nós intermediários		Nós intermediários	
3	$v_3 = v_1 v_2$	= 4	$\dot{v}_3 = \dot{v}_1 v_2 + \dot{v}_2 v_1$	= $2 + 0 = 2$
4	$v_4 = v_2 + v_3$	= 6	$\dot{v}_4 = \dot{v}_2 + \dot{v}_3$	= 2
5	$v_5 = \sin(v_2)$	= 0.91 ...	$\dot{v}_5 = 0$	= 0
	Nós de saída		Nós de saída	
6	$f_1 = v_4$	= 6	$\frac{\partial f_1}{\partial p_1} = \dot{f}_1 = \dot{v}_4$	= 2
7	$f_2 = v_5$	= 0.91 ...	$\frac{\partial f_2}{\partial p_1} = \dot{f}_2 = \dot{v}_5$	= 0

Tabela 2.1: Processo de diferenciação automática direta da função  $\vec{f}$  em relação ao seu primeiro parâmetro,  $p_1$  no ponto  $\vec{p} = [2 \ 2]^\top$ .

da cadeia, enquanto se resolve o cálculo direto da função. Neste método, as tangentes dos últimos nós compõem exatamente as colunas da matriz jacobiana de  $\vec{f}(\vec{p})$  de número igual ao número do parâmetro ao qual  $\vec{f}$  foi diferenciada.

A implementação da autodiff direta geralmente inclui uma estrutura de dados capaz de armazenar os dois valores, o primário e a tangente, e a definição de todas as operações básicas em conjunto com suas regras de derivação. Assim, a função de multiplicação entre duas variáveis deve ser implementada de forma a não só multiplicar os primários das duas entradas, como somar as multiplicações entre o primário de uma entrada e a tangente da outra (regra da derivada do produto). Daí em frente, sempre que a multiplicação entre dois valores for realizada, a propagação das tangentes também o será.

Um exemplo de implementação da diferenciação automática direta escrito em Python está presente no Apêndice D deste documento.

Por completude, a multiplicação da matriz jacobiana por um vetor à direita é muito comum em problemas de otimização (*i.e.*, pelo vetor de resíduos:  $J_{\vec{f}} \vec{r}$ ) e é geralmente custosa. Este procedimento pode ser realizado sem necessidade de construir a matriz jacobiana usando tecnologia de autodiferenciação. Para isso, os nós de entrada recebem como primários os valores do ponto em que se deseja calcular o jacobiano, enquanto suas tangentes são tomadas como o vetor ao qual se deseja multiplicar,  $\vec{r}$ . As tangentes finais deste processo são  $J_{\vec{f}} \vec{r}$  (Baydin et al., 2018).

A autodiferenciação direta é muito eficiente quando se possui poucas entradas e muitas saídas. De outro modo, a autodiferenciação reversa é vantajosa. Este método será descrito na seção a seguir.

## 2.4.2 Diferenciação automática reversa

Na diferenciação automática reversa, tenta-se calcular a regra da cadeia no sentido oposto do grafo. O modo reverso é um pouco mais capcioso que o direto, uma vez que nele se deseja derivar o resultado que ainda não foi calculado com relação aos valores intermediários, e finalmente, os de entrada da função. Por isso, ao invés de uma única passada pelo grafo da função, a autodiff reversa exige duas. A primeira calcula os primários e mapeia as operações e dependências ocorridas no grafo, enquanto a segunda ocorre no sentido oposto, uma vez

que o grafo já é conhecido a esse ponto, calculando as diferenciais, por propagação reversa, das saídas às entradas.

A autodiff reversa possui dois conceitos similares aos da direta. Ao invés de primários e tangentes, temos primários e adjuntos. Neste modelo, escolhemos uma das saídas da função  $\vec{f}(\vec{p})$ ,  $f_q$ , e a diferenciamos com relação às entradas, ou seja, calcular  $\vec{\nabla}f_q(\vec{p})$  — note a semelhança com o gradiente de uma função custo,  $\vec{\nabla}C(\vec{p})$ . Nestes termos, o novo conceito de adjunto é definido para  $v_i$  da seguinte maneira:

$$\bar{v}_i = \frac{\partial f_q}{\partial v_i}, \quad (2.9)$$

e representa a sensibilidade da saída  $f_q$  em relação a  $v_i$ , e deve ser calculado pela aplicação contínua da regra da cadeia em sentido reverso. Assim, para calcularmos a derivada parcial de  $f_1$  em relação a  $v_2$  a partir do grafo na Figura 2.1, temos:

$$\bar{v}_2 = \frac{\partial f_1}{\partial v_4} \left( \frac{\partial v_4}{\partial v_3} \frac{\partial v_3}{\partial v_2} + \frac{\partial v_4}{\partial v_2} \right) = \frac{\partial f_1}{\partial v_3} \frac{\partial v_3}{\partial v_2} + \frac{\partial f_1}{\partial v_4} \frac{\partial v_4}{\partial v_2} = \bar{v}_3 \frac{\partial v_3}{\partial v_2} + \bar{v}_4 \frac{\partial v_4}{\partial v_2}, \quad (2.10)$$

O resultado da equação (2.10) é composto naturalmente durante diferenciação reversa, onde múltiplas contribuições a um mesmo nó são somadas, como observado na Tabela 2.2 durante a diferenciação de  $f_1$  em relação a seus parâmetros no ponto  $\vec{p} = [2 \ 2]^\top$ .

Por comparação, na seção 2.3 vimos que se uma função custo  $C(\vec{p})$  depende de  $S = 1.000.000$  parâmetros, utilizar diferenciação numérica para calcular o gradiente de  $C$  em  $\vec{p}$  implicaria no cálculo direto de  $C$  um milhão e uma de vezes; por autodiferenciação direta, o número de operações necessárias seria de um milhão; já por autodiferenciação reversa conseguimos resultados expressivamente superiores em casos como este: apenas duas operações são necessárias para calcular todo o gradiente de  $C$  em relação a  $\vec{p}$ . De forma mais genérica, temos por garantia que, se  $\vec{f}$  é um mapeamento  $\mathcal{R}^S \rightarrow \mathcal{R}^Q$ , utilizar diferenciação reversa é mais vantajoso se  $Q \ll S$ . Um detalhe importante, é que a complexidade extra causada pelo cálculo de uma função dentro de um esquema de diferenciação automática é garantida de ser de um fator menor que 6, tipicamente se enquadrando entre 2 e 3.

A implementação deste método exige mais cuidado, pois é necessária a construção de alguma estrutura para guardar todas as informações sobre estrutura do grafo, o que torna este tipo de diferenciação custosa em uso de memória. Assim como na implementação do modo direto, também aqui é necessário definir todas as operações básicas juntamente às suas regras de diferenciação, além da estrutura de grafo para guardar estas informações das operações, de maneira que seja possível realizar o segundo passe diferenciando pelo grafo em sentido oposto. Um exemplo de esquema de diferenciação automática reversa foi implementado na linguagem Python se encontra no Apêndice D deste material.

De forma similar à autodiferenciação direta, onde as tangentes no fim do processo representam uma coluna da matriz jacobiana, os adjuntos no final da diferenciação reversa representam uma de suas linhas (a linha referente à dimensão parcialmente diferenciada). Este método também é útil no cálculo da multiplicação pela transposta da jacobiana,  $\vec{J}_{\vec{f}}^\top \vec{r}$ . Para tal, de modo semelhante à autodiff direta, os valores adjuntos que inicializam a segunda etapa da diferenciação reversa devem ser inicializados com os valores de  $\vec{r}$ . Desta forma, as tangentes obtidas ao fim do processo são o resultado da multiplicação, sem a necessidade de construir toda a matriz jacobiana antes do resultado.

As características dos métodos de diferenciação discutidas nesta seção são summarizadas na Tabela 2.3.

<b>Passo</b>	<b>Primários</b>		<b>Passo</b>	<b>Adjuntos</b>	
	Nós de entrada			Nós de entrada	
1	$v_1 = p_1$	= 2	15	$\frac{\partial f_1}{\partial p_1} = \bar{v}_1$	= 2
2	$v_2 = p_2$	= 2	14	$\frac{\partial f_1}{\partial p_2} = \bar{v}_2$	= 3
	Nós intermediários			Nós intermediários	
3	$v_3 = v_1 v_2$	= 4	13	$\bar{v}_1 = v_2 \bar{v}_3$	= $2 \times 1 = 2$
4	$v_4 = v_2 + v_3$	= 6	12	$\bar{v}_2 = \bar{v}_2 + v_1 \bar{v}_3$	= $1 + 2 \times 1 = 3$
5	$v_5 = \sin(v_2)$	= 0.91 ...	11	$\bar{v}_3 = \bar{v}_4$	= 1
	Nós de saída		10	$\bar{v}_2 = \bar{v}_4$	= 1
6	$f_1 = v_4$	= 6	9	$\bar{v}_4 = \frac{\partial f_1}{\partial v_4}$	= 1
7	$f_2 = v_5$	= 0.91 ...	8	$\bar{v}_5 = \frac{\partial f_1}{\partial v_5}$	= 0
	Nós de saída			Nós de saída	

Tabela 2.2: Processo de diferenciação automática reversa da primeira saída da função  $\vec{f}, f_1$ , em relação aos seus parâmetros no ponto  $\vec{p} = [2 \ 2]^\top$ .

Método de diferenciação				
Manual	Simbólica	Numérica	Automática	Característica
✓	✓	-	✓	Exata
✓	✓	-	✓	Computacionalmente eficiente
-	✓	✓	✓	Generalizável
-	-	✓	✓	Não exige forma fechada de $\vec{f}$
✓	✓	-	✓*	Retorna forma fechada de $J_{\vec{f}}$

Tabela 2.3: Características dos métodos de diferenciação. (\*): Caso a função estudada seja dada em forma fechada e um tratamento simbólico for dado ao grafo.

# Capítulo 3

## De regressões lineares a redes neurais

As redes neurais artificiais (ANNs) formam uma imensa contribuição tecnológica ao mundo moderno. Esta classe de modelos com estrutura arbitrária é altamente versátil, sendo capaz de produzir resultados no estado-da-arte nas áreas de previsão temporal, filtragem de sinais, segmentação de imagens, aproximação de funções, análise de sentimento, produção automática de música, reconhecimento de voz, tradução e muito mais. É possível enxergar no percurso histórico na criação destes modelos um caminho que segue naturalmente dos problemas de regressão linear ao seu estado atual. Este capítulo visa evidenciar tal percurso a partir da matemática.

### 3.1 Regressão linear

O método da regressão linear é um modelo estatístico bastante simples e com propriedades profundamente conhecidas. A técnica é considerada um dos primeiros métodos de aprendizado de máquina existentes. Considerando uma amostra  $\vec{x}$ , o modelo,  $\vec{f}$ , consiste em sua combinação linear com uma matriz de pesos,  $\mathbf{W}$ , de forma a gerar um vetor  $\vec{y}$  de previsões:

$$\vec{y} = \vec{f}(\vec{x}) = \mathbf{W}\vec{x}. \quad (3.1)$$

Um esquema deste modelo pode ser visto na Figura 3.1.

Para generalizar o modelo a um conjunto de observações,  $\mathbf{X}_{N \times M}$ , tendo-se em vista um conjunto de efeitos alvos a elas associados,  $\mathbf{Y}_{N \times Q}$ , pode-se desenvolver um modelo linear  $f$

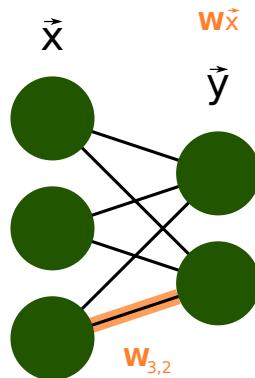


Figura 3.1: Grafo de um modelo de regressão linear com três entradas e duas saídas.

tal que  $f(\mathbf{X}) = \hat{\mathbf{Y}}_{N \times Q} \approx \mathbf{Y}_{N \times Q}$  para regredir  $\mathbf{Y}_{N \times Q}$  a partir de  $\mathbf{X}_{N \times M}$  da seguinte maneira:

$$f(\mathbf{X}) = \mathbf{X}\mathbf{W}, \quad (3.2)$$

onde  $\mathbf{W}_{M \times Q}$  é a matriz de parâmetros que possui em cada coluna  $m$  as relações lineares das colunas de  $\mathbf{X}$  e a coluna  $m$  de  $\mathbf{Y}$ . Quando  $M = Q = 1$ , diz-se que a regressão é simples, isto é, uma variável de entrada está sendo relacionada a uma única variável de saída; quando  $Q = 1$  e  $M > 1$ , trata-se de uma regressão multi-variável, uma vez que cada observação consta de  $M$  variáveis, as quais se relacionam com uma única variável de saída; e quando  $M, Q > 1$ , trata-se de um modelo linear geral, que associa  $M$  variáveis a  $Q$  outras variáveis. Este último, porém, pode ser escrito como uma composição de  $Q$  modelos lineares multivariáveis que são independentes entre si (Kim e Timm, 2006).

Quando treinado diretamente sobre os dados, um modelo com a forma da equação (3.2) possui viés, uma vez que este não considera efeitos em  $\mathbf{Y}$  que são independentes das variáveis utilizadas na regressão. Em outras palavras, ele parte do princípio de que se as variáveis de entrada são nulas, as saídas também serão nulas. Para atenuar este viés, deve-se adicionar uma variável auxiliar de valor fixo e independente das observações reais. A adição desta variável independente toma a forma de  $\vec{b}$  na equação a seguir:

$$\vec{y} = \vec{f}(\vec{x}) = \mathbf{W}\vec{x} + \vec{b}. \quad (3.3)$$

Já para várias amostras, uma variável vetorial cujos elementos possuem um mesmo valor — tipicamente unitário ( $\vec{1}$ ) — é adicionada à esquerda de  $\mathbf{X}$ , formando a matrix de observações estendida,  $\mathbf{X}'$ . Nestes termos, a matriz de parâmetros  $\mathbf{W}$ , que multiplica  $\mathbf{X}'$  à direita, deve pertencer a  $\mathcal{R}^{(M+1) \times Q}$ :

$$\mathbf{W} = \begin{bmatrix} \vec{w}_0^\top \\ \vec{w}_1^\top \\ \vdots \\ \vec{w}_m^\top \\ \vdots \\ \vec{w}_M^\top \end{bmatrix} = \begin{bmatrix} w_{01} & \cdots & w_{0q} & \cdots & w_{0Q} \\ w_{11} & \cdots & w_{1q} & \cdots & w_{1Q} \\ \vdots & & \vdots & & \vdots \\ w_{m1} & \cdots & w_{mq} & \cdots & w_{mQ} \\ \vdots & & \vdots & & \vdots \\ w_{M1} & \cdots & w_{Mq} & \cdots & w_{MQ} \end{bmatrix}, \quad (3.4)$$

e o modelo em (3.5) se torna desenviesado, uma vez que a primeira linha de  $\mathbf{W}$ ,  $\vec{w}_0^\top$ , que não multiplica as reais variáveis observadas,  $\mathbf{X}$ , é treinada para obter valores que balanceiem o viés do modelo linear.  $\vec{w}_0^\top$  recebe os nomes de *termo independente*, *valor de interceptação* e, no contexto das redes neurais, de *viés*.

$$f(\mathbf{X}) = [\vec{1} \quad \mathbf{X}] \mathbf{W}, \quad (3.5)$$

Existem diversas técnicas para inverter este modelo e descobrir os valores dos parâmetros em  $\mathbf{W}$  (i.e.,  $\mathbf{W} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{Y}$  para um problema sobredeterminado (Deisenroth et al., 2020)). Utilizando os conceitos deste material, podemos criar uma função custo  $C(\mathbf{W})$  adequada (i.e.,  $C(\mathbf{W}) = MSE(f(\mathbf{X}), \mathbf{Y})$ ), e treiná-lo com base em um dos métodos gradiente citados (i.e., SGD).

A interpretação de  $\mathbf{W}$  é profundamente compreendida (James et al., 2013). A partir de seus valores pode-se retirar informações quantitativas e qualitativas sobre as relações entre as variáveis observadas e as alvejadas. Primeiramente cada um dos  $Q$  elementos em  $\vec{w}_0$  afirma sobre o que ocorre com as respectivas  $Q$  saídas do modelo quando todas as variáveis

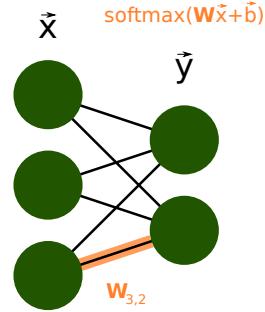


Figura 3.2: Grafo de um modelo de regressão logística multinomial com três entradas e duas saídas probabilísticas.

são nulas, uma espécie de ponto inicial do modelo  $f$ . Para  $m > 0$ , os  $Q$  elementos de  $\vec{w}_m$  são relacionados à importância das variáveis observadas para a predição das  $Q$  saídas. Valores pequenos implicam em baixa correlação entre as variáveis relacionadas, enquanto valores muito positivos implicam em forte correlação linear entre as variáveis. Por outro lado, valores muito negativos implicam em uma relação inversa entre as variáveis observada e alvo analisadas. Diretamente, o valor  $w_{m,q}$  explica o incremento sobre a saída  $q$  do modelo quando a variável observada  $m$  aumenta de uma unidade. Uso destes modelos pode, por exemplo, justificar, para uma empresa, o aumento de verba destinada a publicidade, caso o modelo linear treinado explique o número de vendas realizadas a partir desta variável por meio de um parâmetro  $w$  positivo.

## 3.2 Regressão logística

Apesar do nome de regressão, a regressão logística é um modelo de classificação. A história deste método é bem desenvolvida por Cramer (2002). Tal modelo se estende da regressão linear para calcular probabilidades de uma determinada amostra com  $M$  dimensões (ou  $M$  variáveis) pertencer a cada uma das  $Q$  classes consideradas. Um exemplo simples é o cálculo da probabilidade de um cliente com determinada quantidade de recursos ser confiável a empréstimos bancários — em outras palavras, a probabilidade de o cliente fazer parte da categoria de clientes confiáveis em detrimento da classe de clientes não-confiáveis.

O modelo de regressão logística é definido de forma muito similar ao da regressão linear. Considera-se que as probabilidades reais de pertencimento de cada amostra de treinamento nas linhas da matriz de observações  $\mathbf{X}$  a cada uma das  $Q > 1$  classes é dada nas linhas da matriz  $\mathbf{Y}$ <sup>1</sup>. Nestes termos, um modelo de regressão logística é descrito por  $f(\mathbf{X}) = \hat{\mathbf{Y}} \approx \mathbf{Y}$ , como na equação (3.6). O grafo deste modelo para uma amostra pode ser visto na Figura 3.2 e deve ser comparado ao da regressão linear na Figura 3.1.

$$f(\mathbf{X}) = \text{softmax}(\mathbf{X}\mathbf{W}) \quad (3.6)$$

Aqui, a função de normalização exponencial é notada por softmax, e funciona como um normalizador de probabilidades, sendo aplicada a cada linha da matriz resultante de  $\mathbf{X}\mathbf{W}$ . Assim, as saídas desta combinação linear, cujos valores pertencem ao intervalo  $[-\infty, +\infty]$ ,

<sup>1</sup>No caso da classificação,  $\mathbf{Y}$  é tipicamente construída a partir da transformação *one-hot* dos rótulos das amostras às quais se sabe a classificação. Ver Apêndice C.

são comprimidas ao intervalo de probabilidades válidas:  $[0, 1]$ . Além disso, esta função garante que o somatório das probabilidades de pertencimento de uma mesma amostra às classes possíveis seja sempre igual a 1 (100%). Matematicamente, a função softmax é definida para um vetor  $\vec{z}$  da seguinte maneira:

$$\text{softmax}(\vec{z}) = \frac{e^{\vec{z}}}{\vec{1}^\top e^{\vec{z}}} = \frac{e^{\vec{z}}}{\sum_{q=1}^Q e^{z_q}}, \quad \vec{z} \in \mathcal{R}^Q. \quad (3.7)$$

A regressão logística é dita binomial quando o número de classes  $Q$  é igual a 2, e multinomial quando  $Q > 2$ . Mostra-se que uma regressão logística multinomial com  $Q$  classes pode ser escrita a partir de  $Q - 1$  regressões binomiais, uma vez que sabendo-se de  $Q - 1$  probabilidades de classificação, a desconhecida deve somar 100% com as primeiras. Isto significa que para o caso binomial basta que uma das probabilidades seja conhecida para se determinar a outra. Este artifício pode ser implementada por meio da investigação da função softmax quando  $Q = 2$ :

$$\text{softmax}(\vec{z}) = \begin{bmatrix} \frac{e^{z_1}}{e^{z_1} + e^{z_2}} \\ \frac{e^{z_2}}{e^{z_1} + e^{z_2}} \end{bmatrix} = \begin{bmatrix} \frac{1}{1+e^{-(z_1-z_2)}} \\ \frac{1}{1+e^{z_1-z_2}} \end{bmatrix} = \begin{bmatrix} \frac{1}{1+e^{-z_*}} \\ \frac{1}{1+e^{z_*}} \end{bmatrix} = \begin{bmatrix} \sigma(z_*) \\ \sigma(-z_*) \end{bmatrix}, \quad z_* = z_1 - z_2, \quad (3.8)$$

onde  $\sigma$  é a função sigmoide descrita por:

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \quad (3.9)$$

Em nosso contexto,  $\vec{z}^\top$  deve ser lido como uma das linhas da combinação linear  $\mathbf{X}\mathbf{W}$  na equação (3.6). Por fim, a probabilidade das amostras em  $\mathbf{X}$  pertencerem à primeira classe de uma regressão logística binomial é descrita por:

$$f(\mathbf{X}) = \sigma(\mathbf{X}\vec{w}). \quad (3.10)$$

onde  $\vec{w} \in \mathcal{R}^M$  é a matriz com um único vetor de parâmetros. Este tratamento implica num treinamento eficiente, uma vez que o modelo de uma única probabilidade só depende de  $M$  parâmetros. Em comparação, o tratamento multinomial padrão exigiria  $2M$  parâmetros em  $\mathbf{W}$  ( $M$  deles para cada uma das  $Q = 2$  classes).

O gráfico da função sigmoide (Figura 3.3) demonstra de forma visual sua não-linearidade, e portanto a não-linearidade inerente à função softmax.

Assim como no caso da regressão linear, por padrão o modelo logístico é enviesado por assumir que uma amostra nula possui probabilidade  $1/Q$  de pertencer a cada uma de suas  $Q$  classes. A remoção deste viés ocorre da mesma forma que a demonstrada na Seção 3.1: adicionamos uma coluna extra com valor unitário ao início da matriz de observações  $\mathbf{X}$ , de forma que a primeira linha de  $\mathbf{W}$  contenha os parâmetros que balancearão o viés do modelo para cada classe. Portanto, o modelo de regressão logística multinomial desenviesado é descrito na equação (3.11):

$$f(\mathbf{X}) = \text{softmax}([\vec{1} \ \mathbf{X}] \mathbf{W}), \quad (3.11)$$

enquanto o modelo de regressão logística binomial desenviesado é dado na equação (3.12)

$$f(\mathbf{X}) = \sigma([\vec{1} \ \mathbf{X}] \vec{w}) \quad (3.12)$$

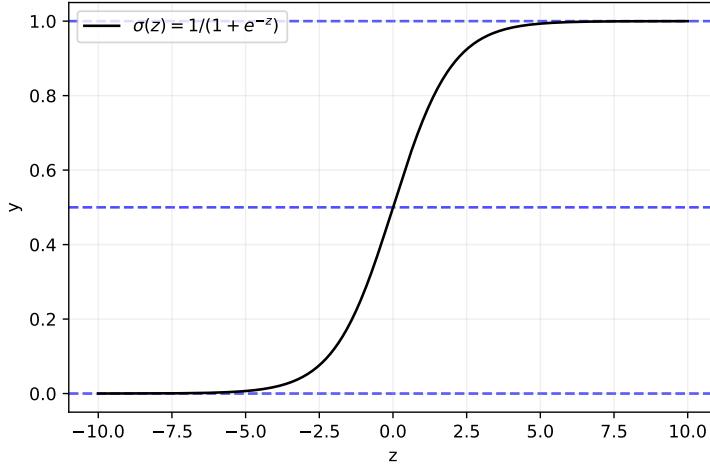


Figura 3.3: Gráfico da função sigmoide ( $\sigma(z) = \frac{1}{1+e^{-z}}$ )

A rotulação de cada amostra é dada então pela classe de maior probabilidade calculada. A interpretação destes modelos também foi fortemente estudada, e segue similar ao caso da regressão linear, exceto que o elemento  $w_{m,q}$  da matriz  $\mathbf{W}$  afirma sobre o incremento de probabilidade de uma amostra à classe  $q$  dado um determinado incremento sobre a variável  $m$  monotonicamente, entretanto não-linear. Menard (2001) descreve o processo de interpretação de modelos logísticos de forma detalhada, bem como suas propriedades essenciais.

Uma forma para inverter modelos de classificação para descobrir os valores dos parâmetros em  $\mathbf{W}$  é definir uma função custo  $C(\mathbf{W})$  adequada, por exemplo a entropia cruzada média, e buscar seu mínimo por meio de um dos métodos gradiente aqui descritos (*i.e.*, Adam).

### 3.3 Redes neurais artificiais

Em 1943, o neurofisiologista Warren McCulloch e o lógico Walter Pitts publicam um artigo explicando o funcionamento do sistema nervoso. Sua descrição é especial na história da ciência por seu empenho em formalizar o comportamento biológico dos neurônios. É sabido da biologia que um neurônio pode ser dividido em duas partes: soma e axônio. A soma se refere ao corpo celular de um neurônio, enquanto o axônio é sua parte mais alongada, funcionando como um transmissor de sinais gerados na soma a outros neurônios ou tecidos.

Quando é estimulado com intensidade suficiente para ser excitado, isto é, com uma diferença de potencial superior a um valor limiar (seu potencial de ação), ele é completamente ativado, propagando um pulso elétrico à frente. O intervalo até a inicialização da resposta elétrica é determinado pelo próprio neurônio, entretanto a amplitude do sinal é sempre fixa. Este princípio é chamado “princípio do tudo ou nada da atividade nervosa”. Após transitar pelo axônio, o sinal emitido pela soma é capturado por outro neurônio em regiões conhecidas como sinapses, e um conjunto de neurônios conectados entre si, por vezes chamado de rede neural. Rede, não linha, pois não é comum que um neurônio seja ativado pelo impulso elétrico gerado por um único vizinho. Em geral a excitação de um neurônio ocorre quando uma célula neural é estimulada por múltiplos vizinhos dentro de frações de milisegundo. Por fim, a velocidade de propagação elétrica nas porções mais longas dos axônios é alta, da ordem

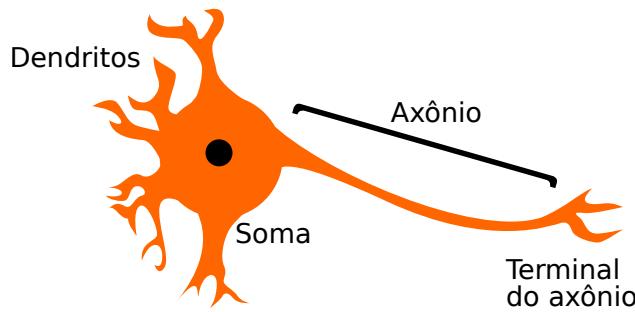


Figura 3.4: Estrutura básica de um neurônio.

de 150 m/s, de forma que é razoável considerar que pulsos que saem de uma mesma fonte chegam instantaneamente em seus destinos, independentemente da distância percorrida.

McCulloch e Pitts nos propõem que tal comportamento definido no princípio do tudo ou nada, onde a resposta de um neurônio é binária (sem sinal ou pulso definido) é tratável em termos de lógica proposicional. Assim, a atividade de cada neurônio pode ser representada como uma proposição que é calculada sobre os pulsos previamente recebidos. A representação matemática inicialmente proposta em seu trabalho, bem como suas incontáveis variações recentes, é atualmente conhecida pelo termo geral “rede neural artificial” (*artificial neural network* – ANN).

Os primeiros trabalhos envolvendo ANNs foram justificados como uma tentativa de melhor entender o funcionamento do cérebro, desde a biofísica matemática de Rashevsky (1938), por meio de seu modelamento matemático. Rummelhart et al. (1986) mostraram então que utilizar propagação reversa — um caso especial da diferenciação automática reversa para quando se possui uma única variável de saída (*i.e.*, função objetivo) —, torna possível o treinar modelos de rede neural parametrizados matematicamente, e que estes poderiam resultar em aplicações interessantes. Somente em 1989, LeCun et al. demonstraram um uso prático de treinamento de rede para a comunidade científica. Seu resultado foi um programa de computador capaz de reconhecer dígitos em códigos de barras por meio de uma rede neural convolucional. A introdução de placas gráficas para o treinamento de redes profundas foi fortemente evidenciada como um avanço tecnológico útil ao assunto por meio do trabalho de Ciresan et al. (2011). Dez anos depois, as redes neurais são estudadas como modelos preditores versáteis e eficientes, sendo aplicadas em virtualmente todas as áreas da ciência, direta ou indiretamente afetando a vida de todos os seres humanos.

Em especial, no que tange a este trabalho, o artigo de McCulloch e Pitts (1943) divide-as em dois tipos: livres de círculos e com presença de círculos. Estes dois casos são respectivamente tratados hoje pelos termos "rede neural pré-alimentada" (*feed-forward neural network* – FNN) e "rede neural retro-alimentada" (*recurrent neural networks* – RNN), as quais serão tratadas nas subseções a seguir.

### 3.3.1 Redes neurais pré-alimentadas (FNN)

O perceptron (Rosenblatt, 1958) é considerado o primeiro modelo de rede neural artifical inventado. Com as conexões corretas, este algoritmo pode desempenhar uma classificação binária. Sua formulação matemática para classificar uma matriz de amostras  $\mathbf{X}$  é dada por:

$$f(\mathbf{X}) = H([\vec{1} \quad \mathbf{X}] \vec{w}), \quad (3.13)$$

onde  $H$  é a função degrau de Heaviside, utilizada como função de ativação para o neurônio de saída, e  $\vec{w}$  são os pesos do modelo:

$$H(z) = \begin{cases} 0, & z \leq 0 \\ 1, & z > 0 \end{cases}. \quad (3.14)$$

É perceptível a semelhança entre o perceptron e a regressão logística binomial desenviesada — equação (3.12), onde ao invés da função degrau, utiliza-se a função sigmoide ( $\sigma$ ) como função de ativação. A generalização do perceptron é a sua versão multicamadas (*multilayer perceptron*, MLP). Estes modelos podem ser usados como solução para um número infinito de problemas, e podem ser pensados como o empilhamento de múltiplas regressões multinomiais com funções de ativação arbitrárias. O fato de que uma operação linear como a função identidade pode ser utilizada como função de ativação final de um MLP implica que estes modelos são úteis tanto para classificação, quanto para regressão (Murtagh, 2003) — de fato, este modelo generaliza os modelos de regressão linear multivariada e logística multinomial.

Nos termos do MLP, a regressão linear, a regressão logística e até mesmo um perceptron possuem  $L = 2$  camadas de neurônios: uma cujos neurônios comportam os valores de entrada, e outra cujos neurônios comportarão as saídas do modelo. Um modelo MLP possui ao menos  $L = 3$  camadas, sendo que cada camada  $\vec{a}^{[l]} \in \mathcal{R}^{n_l}$  possui  $n_l$  neurônios e é calculada com base na aplicação de uma função de ativação,  $T^{[l-1]}$ , sobre a combinação linear da camada anterior  $\vec{a}^{[l-1]} \in \mathcal{R}^{n_{l-1}}$  com uma matriz de pesos  $\mathbf{W}^{[l-1]} \in \mathcal{R}^{n_{l-1} \times n_l}$ . Como usualmente deseja-se um modelo desenviesado, deve-se adicionar um termo que balanceie o viés em cada camada antes da ativação, o qual será notado por  $\vec{b}^{[l-1]}$ . Esta recursão se inicia a partir da segunda camada, uma vez que a primeira camada,  $\vec{a}^{[1]}$ , recebe apenas os valores de entrada,  $\vec{x} \in \mathcal{R}^M$ . Deste modo, a camada  $\vec{a}^{[L]} \in \mathcal{R}^Q$  dá as saídas da rede (Chen et al., 2018; Nguyen e Malinsky, 2020). Todas as camadas entre a de entrada e a de saída são chamadas de intermediárias. A Figura 3.5 ilustra uma rede com 5 camadas, onde  $\{n_l\} = \{3, 4, 5, 3, 2\}$  e  $T^{[i]} = \sigma$ ,  $i = 1, \dots, 4$ . Esta rede é representada pela seguinte função:

$$f(\vec{x}) = \sigma(\mathbf{W}^{[4]}\sigma(\mathbf{W}^{[3]}\sigma(\mathbf{W}^{[2]}\sigma(\mathbf{W}^{[1]}\vec{x} + \vec{b}^{[1]}) + \vec{b}^{[2]}) + \vec{b}^{[3]}) + \vec{b}^{[4]}), \quad (3.15)$$

ou para um conjunto de observações  $\mathbf{X}_{N \times M}$ :

$$f(\mathbf{X}) = \sigma \left( [\vec{1} \quad \sigma \left( [\vec{1} \quad \sigma \left( [\vec{1} \quad \sigma \left( [\vec{1} \quad \mathbf{X}] \mathbf{W}^{[1]} \right) \right] \mathbf{W}^{[2]} \right) \right] \mathbf{W}^{[3]} \right] \mathbf{W}^{[4]}, \quad (3.16)$$

onde o viés é solucionado como nas seções anteriores, por adição de uma coluna unitária a  $\mathbf{X}$ , de forma que a primeira linha de cada matriz de pesos  $\mathbf{W}^{[l]}$  assume um valor para equilibrar o viés do modelo linear e  $\sigma$  é aplicada elemento-a-elemento.

A grande vantagem dos MLPs com relação aos modelos de regressão linear é sua capacidade de prever eventos não-lineares, o que decorre diretamente do uso de funções de ativação não-lineares. Quanto mais profunda uma rede é, isto é, quanto maior seu número de camadas intermediárias, mais livre para aproximar funções não-lineares ela se torna, desde que suas camadas intermediárias sejam ativadas por funções não-lineares. Quando apenas ativações

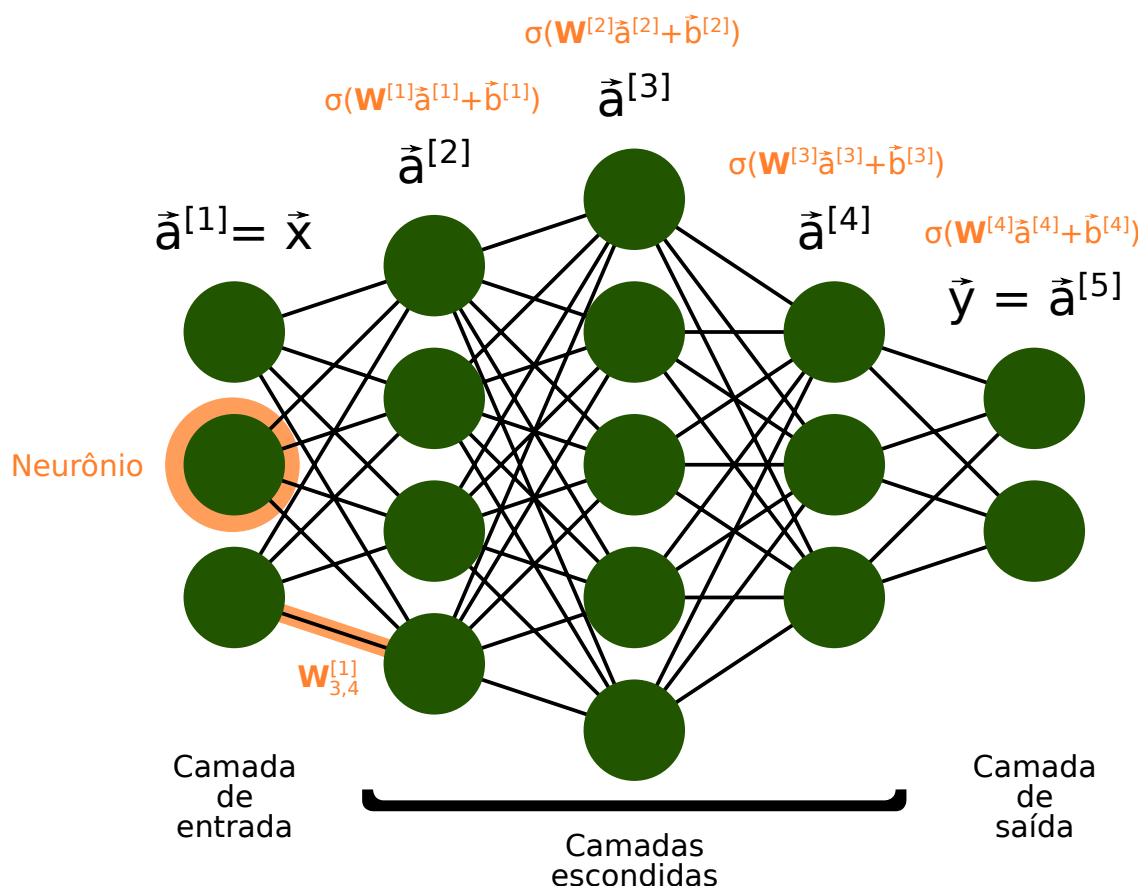


Figura 3.5: Ilustração de uma rede neural tipo MLP com número de camadas  $L = 5$  e sequência de neurônios por camada  $\{n_l\} = \{3, 4, 5, 3, 2\}$ . Esta rede possui entradas em  $\mathcal{R}^3$  e saídas em  $\mathcal{R}^2$ .

lineares são utilizadas, porém, mesmo um modelo de um milhão de camadas será sempre equivalente a um outro modelo de duas, por conta da dependência linear entre as mesmas.

Um código para treinamento de MLPs utilizando o esquema de diferenciação automática do Apêndice E é provido no Apêndice F. O que é comum não só às redes tipo MLP como a todas as redes FNN é a ausência de conexões cíclicas para o cálculo de qualquer variável. Desta forma, elementos de uma mesma camada de neurônios não possuem dependência. Isto não necessariamente é o caso para as redes RNN, as quais serão discutidas na subseção a seguir.

### 3.3.2 Redes neurais recorrentes (RNN)

O modelo de rede neural pré-alimentada considera, enquanto tentativa de modelagem do sistema nervoso, que um processo de pensamento no cérebro é dependente apenas de estímulos externos. Mesmo McCulloch e Pitts (1943) tinham noção de que isto é uma aproximação, uma vez que este sistema é dotado de ligações cíclicas<sup>2</sup>, as quais permitem que um mesmo estímulo elétrico continue reverberando dentro da rede por um longo período de tempo, interagindo multiplas vezes com um mesmo neurônio.

De um ponto de vista mais pragmático, as redes pré-alimentadas incorrem em problemas ao representar entradas que envolvem a dimensão do tempo. A representação de séries temporais nestes modelos, apesar de tratável, é estruturalmente inadequada. Por exemplo, considere as três sequências temporais a seguir:

1.  $[0 \ 0 \ 1]$ ;
2.  $[0 \ 1 \ 0]$ ;
3.  $[1 \ 0 \ 0]$ .

Tais sequências são facilmente identificadas como similares, podendo indicar o caminhar de um pulso unitário ao longo de três intervalos de tempo. Entretanto, este significado seria corrompido numa rede pré-alimentada como o perceptron, onde cada uma das sequências representariam vetores que apontam em direções completamente diferentes — observe-as como pontos num gráfico sobre eixos cartesianos em  $\mathcal{R}^3$ , e enxergará estes dados da forma que uma rede neural pré-alimentada os enxerga.

Uma solução ao problema das séries temporais é a implementação de algum sistema de memória. Tal solução já se encontrava teoricamente no artigo de McCulloch e Pitts (1943), mas sua implementação e tratamento eram inviáveis. Somente em 1986 nasce o primeiro método para se lidar com entradas sequenciais numa rede neural utilizando variáveis de estado. Ambos os modelos de Jordan (1986) e seu sucessor, o de Elman (1990), se baseiam na geração de uma variável de memória calculada com base em variáveis do passo temporal anterior e corrente, a qual é utilizada no cálculo das novas saídas.

O modelo de Jordan utiliza como memória as saídas do último passo temporal processado ( $\vec{y}_{t-1}$ ), que são utilizadas junto às entradas do instante atual,  $\vec{x}_t$ , na geração de uma variável de estado com dimensão fixa,  $\vec{h}_t$ , que por sua vez é utilizada no cálculo da saída atual ( $\vec{y}_t$ ). As equações que descrevem a célula recorrente deste modelo para cada passo  $t$  são dadas em (3.17) e (3.18), onde  $f_h$  e  $f_y$  são funções genéricas. Duas representações para a rede de Jordan podem ser vistas na Figura 3.6.

---

<sup>2</sup>Dentre outros fenômenos aqui desconsiderados, como a facilitação, a extinção e o aprendizado.

$$\vec{h}_t = f_h(\vec{x}_t, \vec{y}_{t-1}) \quad (3.17)$$

$$\vec{y}_t = f_y(\vec{h}_t) \quad (3.18)$$

No caso da rede de Elman, por outro lado, utiliza-se a própria memória do instante anterior,  $\vec{h}_{t-1}$ , para compor com  $\vec{x}_t$  uma nova memória,  $\vec{h}_t$ , que como na rede Jordan, é utilizada na geração da saída atual ( $\vec{y}_t$ ). As seguintes equações mostram a célula de recorrência de uma rede de Elman:

$$\vec{h}_t = f_h(\vec{x}_t, \vec{h}_{t-1}) \quad (3.19)$$

$$\vec{y}_t = f_y(\vec{h}_t), \quad (3.20)$$

onde  $f_h$  e  $f_y$  são funções genéricas. Duas representações para a rede de Elman podem ser vistas na Figura 3.7.

Para que as redes descritas acima sejam treináveis, é necessário que as funções  $f_h$  e  $f_y$  possuam parâmetros de aplicação, os quais podem ser aprendidos num processo futuro de otimização. Desta forma, tais funções são geralmente definidas como simples combinações:

$$\vec{h}_t = \mathbf{W}_{hx}\vec{x}_t + \mathbf{W}_{hy}\vec{y}_{t-1} \quad (3.21)$$

$$\vec{y}_t = \mathbf{W}_{yh}\vec{h}_t, \quad (3.22)$$

sendo cada um dos elementos dos operadores  $\mathbf{W}_{hx}$ ,  $\mathbf{W}_{hh}$  e  $\mathbf{W}_{yh}$  os parâmetros do modelo.

Observe que em (3.21) e (3.22),  $\vec{y}_t$  pode ser escrita como combinação linear de  $\vec{x}_t$  e  $\vec{y}_{t-1}$  diretamente, sem necessidade da utilização da variável de estado  $\vec{h}_t$ . Durante o treinamento do modelo, tamanha linearidade logo se torna uma limitação ao aprendizado de comportamentos, pois muitos dos problemas estudados envolvem soluções que não são lineares, de forma similar ao já discutido na Seção 3.3.1. Portanto, nas redes de Jordan e Elman derivadas de operadores lineares adiciona-se um nível controlado de não-linearidade por meio da aplicação final de uma função de ativação bem conhecida, como a função sigmoide, que é definida na equação (3.9) e pode ser vista na Figura 3.3, que comprime os resultados das combinações lineares gerando maior complexidade ao modelo. Observa-se facilmente pelo gráfico da função sigmoide, que quanto mais à esquerda de zero (abscissa da inflexão) for a entrada, mais sua imagem se aproxima de 0. Nos termos da rede neural, estímulos negativos serão suprimidos pela rede, e como a escolha do valor de corte suave igual a zero utilizado na supressão é arbitrária, a própria decisão da função de ativação introduziu um viés ao modelo.

Para resolver o viés causado na escolha de funções de ativação, geralmente se adiciona um termo de translação nas abscissas, na forma de um vetor de parâmetros extra a ser treinado, propriamente nomeado *termo de viés*. As redes de Jordan e de Elman são finalmente escritas em suas formas desenviesadas mais simples nas duplas de equações (3.23) e (3.24), e (3.25) e (3.26), nas quais as funções de ativação escolhidas são  $T_h$  e  $T_y$ , e os termos de viés são  $\vec{b}_h$  e  $\vec{b}_y$ .

$$\vec{h}_t = T_h(\mathbf{W}_{hx}\vec{x}_t + \mathbf{W}_{hy}\vec{y}_{t-1} + \vec{b}_h) \quad (3.23)$$

$$\vec{y}_t = T_y(\mathbf{W}_{yh}\vec{h}_t + \vec{b}_y) \quad (3.24)$$

$$\vec{h}_t = T_h(\mathbf{W}_{hx}\vec{x}_t + \mathbf{W}_{hh}\vec{h}_{t-1} + \vec{b}_h) \quad (3.25)$$

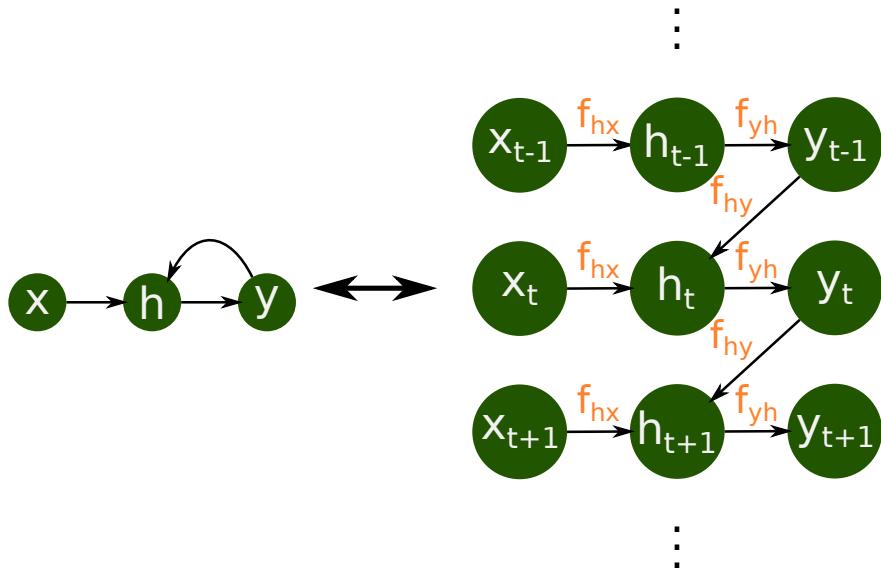


Figura 3.6: Redes recorrente de Jordan em suas representações compacta (esquerda) e desenrolada (direita).

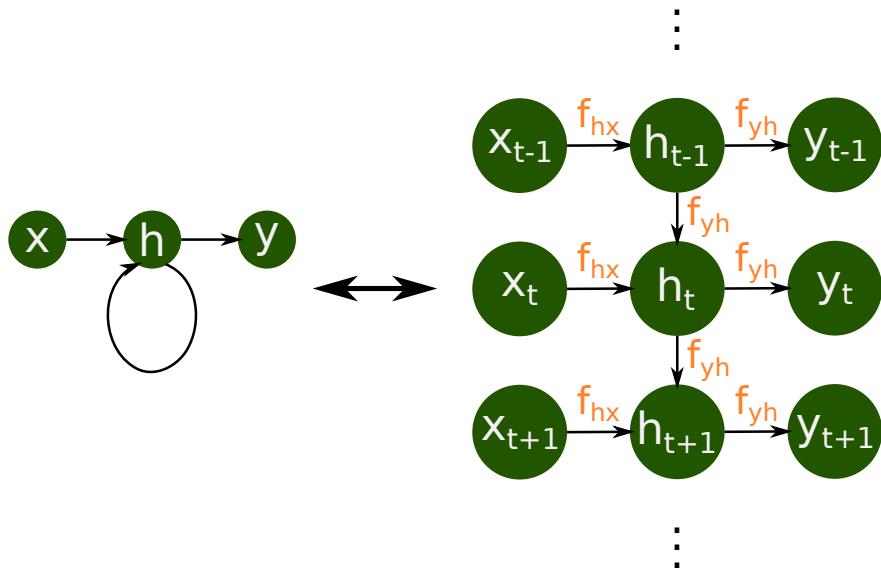


Figura 3.7: Redes recorrente de Elman em suas representações compacta (esquerda) e desenrolada (direita).

$$\vec{y}_t = T_y(\mathbf{W}_{yh}\vec{h}_t + \vec{b}_y) \quad (3.26)$$

As redes neurais recorrentes são por definição redes que permitem grafos com laços de retroalimentação. Por isso, as redes neurais pré-alimentadas são um caso particular das redes retroalimentadas em que nenhum dos nós é retroalimentado na rede. Em detrimento da incrível complexidade que tanto as redes pré-alimentadas como os MLPs, quanto as redes recorrentes, como as de Elman, podem adquirir, estes modelos facilmente se tornam inexplícaveis. O grafo frequentemente apresenta bons resultados em suas aplicações, mas são tão entrelaçados que acaba sendo muito difícil para um ser humano destrinchar-lhos. Uma vez que os modelos não são compreendidos, o controle que podemos ter sobre eles é pequeno. Muitas

alternativas são propostas para aumentar o domínio humano sobre as entranhas das redes neurais. Uma delas é incluir comportamentos conhecidos dentro de suas estruturas. Este é por exemplo o caso das redes neurais informadas de Física (do inglês, *Physics informed neural network* — PINN). No treinamento de uma PINN insatisfações das respostas geradas a equações físicas são postas na função custo de otimização, de forma que os modelos são treinados para não retornarem resultados intangíveis à realidade. Outra abordagem, a utilizada aqui, é a construção de modelos de redes neurais cujo interior contém equações físicas diretamente. No Capítulo 4 é mostrado como construir uma rede de Elman na qual se incorpora um esquema de modelagem a partir da equação física da onda acústica.

# Capítulo 4

## Modelagem da onda acústica como uma rede neural recorrente

Um simulador da equação da onda é imensa serventia à prospecção de hidrocarbonetos em profundidade, uma vez que esta é fortemente baseada na obtenção de dados sísmicos do alvo. Por meio de um simulador, a inversão destes dados sísmicos em propriedades da subsuperfície se torna possível. Neste capítulo aplicaremos o método das diferenças finitas para resolver a equação da onda acústica bidimensional, e assim fazer simulações numéricas de ondas sísmicas. Mostraremos então como podemos implementar uma rede de Elman, como introduzida em 3.3.2, que incorpora uma simulação acústica, e cujos parâmetros são as velocidades de propagação acústica do meio. Neste ponto, verifica-se que o procedimento de treinamento de tal rede é equivalente à inversão da forma de onda completa tradicional. Por fim, revisaremos a versão multiescala da inversão acústica da forma de onda completa, a qual será aplicada também ao treinamento da citada rede neural.

### 4.1 Método das diferenças finitas (FDM)

Considerando a modelagem acústica, isto é, que se pode aproximar o meio modelado a partir de um modelo composto por fluidos, tem-se que a única onda capaz de existir é a compressional, a qual pode ser estudada a partir da equação escalar da onda num campo de pressão:

$$\frac{\partial^2 P(t, \vec{r})}{\partial t^2} = v^2 \nabla^2 P(t, \vec{r}), \quad (4.1)$$

onde  $P(t, \vec{r})$  é o campo de pressão,  $t$  é o tempo e  $\vec{r}$  é uma posição espacial. Com condições iniciais e de contorno dadas, a solução para a equação (4.1) é geralmente muito difícil de ser integrada manual ou mesmo simbolicamente. Por isto, aplicações práticas envolvendo sua resolução via de regra utilizam métodos de integração numérica. Um deles é o método das diferenças finitas (*finite difference method*, FDM), que consiste em aproximar as derivadas por meio do truncamento de séries polinomiais. Começando pela dimensão do tempo, pode-se calcular o valor do campo  $P$  após um pequeno intervalo de tempo  $\Delta t$  a partir da série de Taylor:

$$P(t + \Delta t, \vec{r}) = \frac{1}{0!} P(t, \vec{r}) + \frac{1}{1!} \frac{\partial P(t, \vec{r})}{\partial t}(t) \Delta t + \frac{1}{2!} \frac{\partial^2 P(t, \vec{r})}{\partial t^2}(t) \Delta t^2 + \frac{1}{3!} \frac{\partial^3 P(t, \vec{r})}{\partial t^3}(t) \Delta t^3 + \dots, \quad (4.2)$$

por outro lado, pode-se calcular o valor de  $P$  antes de um mesmo intervalo  $\Delta t$  por:

$$P(t - \Delta t, \vec{r}) = \frac{1}{0!} P(t, \vec{r}) - \frac{1}{1!} \frac{\partial P(t, \vec{r})}{\partial t}(t) \Delta t + \frac{1}{2!} \frac{\partial^2 P(t, \vec{r})}{\partial t^2}(t) \Delta t^2 - \frac{1}{3!} \frac{\partial^3 P(t, \vec{r})}{\partial t^3}(t) \Delta t^3 + \dots, \quad (4.3)$$

A soma das séries das equações (4.2) e (4.3) pode ser truncado ao termo de segunda ordem, donde obtemos:

$$\frac{\partial^2 P(t, \vec{r})}{\partial t^2} \approx \frac{P(t - \Delta t, \vec{r}) - 2P(t, \vec{r}) + P(t + \Delta t, \vec{r})}{\Delta t^2}. \quad (4.4)$$

Este resultado implica que um campo de pressão futuro pode ser estimado a partir de suas condições atual e passada:

$$P(t + \Delta t, \vec{r}) = 2P(t, \vec{r}) - P(t - \Delta t, \vec{r}) + v^2 \Delta t^2 \nabla^2 P(t, \vec{r}) \quad (4.5)$$

Agora resta calcular o Laplaciano do campo de pressão. No caso bidimensional, esta equação se torna:

$$P(t + \Delta t, \vec{r}) = 2P(t, \vec{r}) - P(t - \Delta t, \vec{r}) + v(\vec{r})^2 \Delta t^2 \left( \frac{\partial^2 P(t, \vec{r})}{\partial x^2} + \frac{\partial^2 P(t, \vec{r})}{\partial z^2} \right) \quad (4.6)$$

E tal qual foi realizado para o tempo, podemos aproximar uma derivada espacial de  $P$  na direção  $\vec{r}_s$  por:

$$\frac{\partial^2 P}{\partial \vec{r}_s^2}(t, \vec{r}) \approx \frac{P(t, \vec{r} - \Delta r \vec{r}_s) - 2P(t, \vec{r}) + P(t, \vec{r} + \Delta r \vec{r}_s)}{\Delta r^2}, \quad (4.7)$$

onde  $\vec{r}_s$  é uma direção genérica do espaço e  $\Delta r$  é uma distância pequena. Assim, considerando uma malha de valores de pressão, o valor de  $\frac{\partial^2 P}{\partial \vec{r}_s^2}(t, \vec{r})$  em um ponto  $\vec{r}_0$  do espaço pode ser aproximado a partir das pressões no ponto  $\vec{r}_0$ , no ponto à sua frente e no ponto às suas costas, considerando o sentido de  $\vec{r}_s$ . Tal tratamento discreto naturalmente resulta no conceito de estêncil de diferenças finitas, ou seja, a geometria de nós de uma malha utilizados em uma operação sobre um de seus pontos. Desta forma, o estêncil de diferenças finitas de segunda ordem para  $\frac{\partial^2 P(t, \vec{r})}{\partial r^2}$  na direção  $\vec{r}_s$  da malha num ponto  $\vec{r}_0$  é  $\{\vec{r}_0 - \Delta r \vec{r}_s, \vec{r}_0, \vec{r}_0 + \Delta r \vec{r}_s\}$  e seus coeficientes são  $\{1, -2, 1\}$ . O resultado da combinação dos coeficientes com suas pressões associadas pode ser dividido por  $\Delta r^2$  para que se obtenha o resultado desejado. Caso a ordem de truncamento da série de Taylor fosse diferente de 2, outro estêncil e coeficientes associados seriam necessários para realizar o cálculo das derivadas. Uma vez que se possui a aproximação adequada de diferenças finitas, pode-se aproximar a equação da onda na equação (4.6). Por exemplo, utilizando a aproximação de segunda ordem, tanto para o tempo, quanto para o espaço, obtém-se:

$$\begin{aligned}
P(t + \Delta t, \vec{r}) &= 2P(t, \vec{r}) - P(t - \Delta t, \vec{r}) \\
&\quad + v(\vec{r})^2 \Delta t^2 \left( \frac{P(t, \vec{r} - \Delta x \vec{e}_x) - 2P(t, \vec{r}) + P(t, \vec{r} + \Delta x \vec{e}_x)}{\Delta x^2}, \right. \\
&\quad \left. + \frac{P(t, \vec{r} - \Delta z \vec{e}_z) - 2P(t, \vec{r}) + P(t, \vec{r} + \Delta z \vec{e}_z)}{\Delta z^2} \right)
\end{aligned} \tag{4.8}$$

sendo  $\vec{e}_x$  e  $\vec{e}_z$  os versores na direção dos eixos  $x$  e  $z$ , respectivamente.

Como a modelagem por diferenças finitas é geralmente realizada utilizando aproximação de segunda ordem no tempo, vale de forma abrangente que se  $P$  é dado na forma de uma malha regular de valores discretos, representada matricialmente por  $\mathbf{P}_n$  para o instante  $t_n = t_0 + n\Delta t$ , onde  $t_0$  é o tempo inicial da modelagem, temos que  $\mathbf{P}_{n+1}$  pode ser calculado por:

$$\mathbf{P}_{n+1} = \mathbf{V}^2 \Delta t^2 \nabla^2 \mathbf{P}_n + 2\mathbf{P}_n - \mathbf{P}_{n-1} \tag{4.9}$$

onde  $\mathbf{V}$  é o modelo de velocidades parametrizado como uma matriz que representa valores na mesma malha espacial de  $\mathbf{P}_t$ . A partir da equação (4.9) pode-se iterativamente prever passos futuros da onda no tempo. Com frequência, porém, deseja-se modelar o comportamento do campo de pressão resultante de alguma excitação sobre o mesmo. Para tal, deve-se adicionar o termo fonte  $\mathbf{S}_n$  à equação (4.9) na fase anterior à sua resolução. O termo fonte é função do tempo, e seu valor compõe  $\mathbf{P}_n$ , em que  $n = \frac{t-t_0}{\Delta t}$ :

$$\mathbf{P}_{n+1} = \mathbf{V}^2 \Delta t^2 (\nabla^2 \mathbf{P}_n + \mathbf{S}_{n+1}) + 2\mathbf{P}_n - \mathbf{P}_{n-1} \equiv f(\mathbf{V}, \mathbf{P}_{n-1}, \mathbf{P}_n, \mathbf{S}_{n+1}) \tag{4.10}$$

Nesta equação,  $\mathbf{S}_{n+1}$  pode ser uma matriz nula em todo ponto diferente do ponto de aplicação da fonte. Matematicamente este caso é equivalente a multiplicar a amplitude  $s_n$  da fonte por uma matriz  $\delta_{\vec{r}, \vec{r}_s} \equiv \delta_{\vec{r}, \vec{r}_s} \mathbf{1}$ , na qual  $\delta_{\vec{r}, \vec{r}_s}$  é o delta de Kronecker, uma função que zera todas as posições de seu argumento, que não a posição  $\vec{r}_s$  da fonte, e  $\mathbf{1}$  é uma matriz com todos os elementos iguais a 1. Nesta equação, a função de modelagem recursiva foi chamada de  $f$ .

A simulação do pulso que chega num receptor em  $\vec{r}_r$  dentro do campo de pressão pode ser obtida em cada tempo por  $\sum_{\vec{r} \in \mathcal{D}} \delta_{\vec{r}, \vec{r}_r} \mathbf{P}_n$ , onde  $\mathcal{D}$  é o domínio discreto do modelo, dado pelos índices de  $\mathbf{P}_n$ . Na sísmica, o vetor destas amplitudes para cada tempo é chamado de traço sísmico, e o conjunto de traços sísmicos de cada receptor utilizado é chamado de sismograma.

Detalhes na implementação do método das diferenças finitas são: a questão da estabilidade numérica, que depende da escolha de um  $\Delta t$  que torne a modelagem possível dadas a velocidade máxima do modelo e as distâncias  $\Delta x$  e  $\Delta z$  utilizadas na malha de discretização; a questão da dispersão, que pode ser causada pela injeção de fontes com frequência muito alta, considerando que um valor limiar é dado em função da velocidade mínima do modelo e dos espaçamentos da malha; e a questão das condições de contorno, uma vez que o modelo discretizado possui limites espaciais. Estes pontos são bem sumarizados nas dissertações de mestrado de dos Santos (2013) e Koehne (2018), bem como na tese de doutorado de Andrade (2017).

Neste trabalho, os métodos das diferenças finitas, pseudo-espectral e de expansão rápida foram implementados com a condição de contorno atenuadora dada por Cerjan et al.

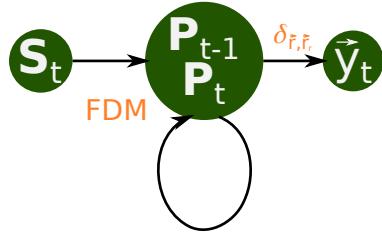


Figura 4.1: Rede recorrente de Elman para modelagem.  $\mathbf{P}_{t-2}$  e  $\mathbf{P}_{t-1}$  e  $\mathbf{S}_t$  são utilizados na composição de um novo par  $(\mathbf{P}_{t-1}, \mathbf{P}_t)$  por meio de modelagem por diferenças finitas (FDM).  $\vec{y}_t$  recebe as amplitudes de  $\mathbf{P}_t$  na posição dos receptores por meio de uma operação  $\delta_{\vec{r},\vec{r}_r}$ .

(1985), apesar de só o primeiro deles ser relatado nos resultados. O controle de estabilidade implementado é dado no artigo de Lines et al. (1999), enquanto que o de dispersão considera o critério de Wu et al. (1996).

A título de curiosidade, vale citar que existem muitos outros métodos para a resolução da equação da onda, entre eles: o método pseudo-espectral (Kosloff e Baysal, 1982; Carcione, 2010), que envolve calcular o Laplaciano na equação (4.10) dentro do domínio da frequência, gerando maior exatidão espacial; o método de expansão rápida (*Rapid expansion method*, REM), o qual permite marcha no tempo com valores de  $\Delta t$  maiores de forma estável, diferentemente do FDM (Pestana e Stoffa, 2009); e o método dos elementos finitos (*finite element method*, FEM), que remove a necessidade do uso de uma malha regular (Carcione et al., 2002). Uma revisão sobre alguns destes métodos está presente em Virieux et al. (2011).

## 4.2 Rede de Elman para propagação acústica

Uma rede recorrente pode ser elaborada para realizar a modelagem por diferenças finitas de segunda ordem no tempo. Para tal, monta-se uma rede de Elman de forma que sua célula recorrente, dada pelas equações (3.19) e (3.20), possua:

- $x_t = \mathbf{S}_t$ , onde  $\mathbf{S}_t$  é o termo fonte no tempo discreto  $t$ ;
- $h_t = (\mathbf{P}_{t-1}, \mathbf{P}_t)$ , onde  $\mathbf{P}_t = f(\mathbf{V}, h_{t-1}, x_t)$  é o campo de pressão no tempo discreto  $t$ , sendo  $f$  a função de modelagem dada na equação (4.10) e  $\mathbf{V}$  o campo discreto de velocidades, parâmetro da rede;
- $y_t = \vec{y}_t$ , onde  $\vec{y}_t$  são as amplitudes de  $\mathbf{P}_t$  nas posições dos receptores.

Esta configuração da rede de Elman pode ser vista na Figura 4.1. Um sismograma é conseguido acumulando os vetores  $\vec{y}_t$  como linhas de uma matriz.

## 4.3 Inversão da forma de onda completa (FWI)

A inversão da forma de onda completa consiste na recuperação dos parâmetros sísmicos em subsuperfície de forma iterativa, a partir da modelagem linearizada de uma equação de onda. Ao contrário de métodos anteriores, em que os tempos de chegada de determinados eventos

são invertidos, na FWI constrói-se uma função custo entre as próprias amplitudes do dado adquirido e do dado modelado, a qual deve ser minimizada a partir dos parâmetros sísmicos do modelo atual (Tarantola, 1984). Os parâmetros da propagação acústica são tipicamente as velocidades de propagação sísmica no meio estudado, o qual é representado como uma malha de pontos. Caso esta malha seja retangular, podemos aplicar o método das diferenças finitas, como explicado na Seção 4.1, para modelar as amplitudes que chegam nos receptores. Desta forma podemos comparar dados modelados com a função custo escolhida (ver Seção 1.2) e atualizar o modelo por métodos de inversão linearizada baseados em seu gradiente.

O desafio do cálculo do gradiente de uma função custo sobre uma modelagem física altamente não-linear é tradicionalmente resolvido a partir do método do estado adjunto (Fichtner et al., 2006; Chen et al., 2018). Este método consiste na definição de um problema com variáveis adjuntas, o qual é então utilizado no cálculo do gradiente do problema original com complexidade computacional comparável à resolução do problema direto e independente do número de parâmetros envolvidos (Johnson, 2012). Caso o custo de erro quadrático  $c(v) = \frac{1}{2} \|\vec{e}\|_2^2 = \|\vec{y} - \hat{y}\|_2^2$  seja considerado, onde  $\hat{y}$  é o dado modelado, e  $y$  o dado observado, o gradiente da função custo obtido pelo método adjunto para a FWI acústica (AFWI) se dá por:

$$\nabla c = \frac{2}{v(\vec{r})^3} \int_0^T \lambda(t, \vec{r}) \ddot{\hat{P}}(t, \vec{r}) dt \quad (4.11)$$

onde  $v$  é o campo de velocidade do modelo e  $\hat{P}$  é o campo de pressão modelado e  $\lambda$  é o campo adjunto que satisfaz à equação (dos Santos, 2013):

$$\frac{1}{v(\vec{r})^2} \frac{\partial^2 \lambda(t, \vec{r})}{\partial t^2} = \nabla^2 \lambda(t, \vec{r}) + (\hat{y} - y) \quad (4.12)$$

onde  $\hat{y}$  é o dado calculado e  $y$  é o dado observado. A solução de  $\lambda$  nesta equação é obtida pela propagação reversa do resíduo do dado, por conta das condições de contorno a seguir:

$$\begin{cases} \lambda(t = T, \vec{r}) = 0 \\ \frac{\partial \lambda}{\partial t}(t = T, \vec{r}) = 0 \end{cases} \quad (4.13)$$

Assim, o método adjunto envolve uma modelagem que parte de condições iniciais para calcular  $\hat{P}$  e outra, de condições finais, para a obtenção de  $\lambda$ . O somatório do produto destes campos na equação 4.11 pode ser enxergado como uma migração dos resíduos utilizando uma condição de imagem de correlação cruzada (Virieux e Operto, 2009; Biondi e Sava, 1999).

## 4.4 Inversão como treinamento de uma rede recorrente de Elman

O método adjunto é extremamente eficiente no cálculo do gradiente da função custo na FWI, entretanto, sua utilização em um problema de inversão sobre outro fenômeno físico exigiria uma nova formulação para a variável adjunta, à qual a implementação do gradiente se tornaria secundária. A escrita de processos de simulação física como redes neurais, ou mais precisamente, como grafos num esquema de diferenciação automática, tem como vantagem

que o cálculo do gradiente pode ser realizado imediatamente após a implementação do operador de simulação. Aliado a isto, prova-se (ver Apêndice A) que a diferenciação automática calcula o gradiente da função custo de erro quadrático de forma equivalente ao procedimento realizado no método adjunto.

Outro ponto positivo do tratamento de processos de modelagem como redes neurais é a inclusão do problema no contexto do aprendizado de máquina. Atualmente existem diversos ambientes com todo o ferramental necessário para treinar modelos de aprendizado de máquina com eficiência e controle. Entre eles o TensorFlow (Abadi et al., 2015), o JAX (Bradbury et al., 2018). Tais ambientes provêm ao usuário formas rápidas e reutilizáveis de realizar operações em diferentes tipos de unidades de processamento, de implementar otimizadores estocásticos, de desenvolver novos tipos de interrupção precoce e muito mais.

Ainda assim, o método adjunto está longe de ser descartado. O conhecimento da solução analítica que o método adjunto proporciona por natureza pode abrir novas janelas para otimização. No caso da FWI, por exemplo, a implementação do método adjunto se inicializa pela multiplicação do campo primário no tempo inicial  $P(t_0)$  pelo campo adjunto  $\lambda$  no mesmo instante. O campo adjunto inicial, por sua vez, depende das condições finais da modelagem, as quais são por sua vez dependentes do último campo primário modelado. Isto implica que na solução do método adjunto, e por consequência naquela realizada na diferenciação automática, por padrão é necessário guardar todos os campos da modelagem na memória para então proceder com a resolução do campo adjunto e do gradiente da função custo. Desta forma, seu cálculo depende de grandes volumes de memória. Ter acesso à equação do gradiente do método adjunto nos permite utilizar técnicas para minimizar o uso de memória, de forma a salvar apenas o mínimo de passos temporais da modelagem para estimar a integral da equação (4.11).

## 4.5 Inversão multiescala

Um tópico importante sobre a inversão sísmica é que os dados modelados possuem uma banda de frequência ampla, o que faz com que durante a inversão os picos de um determinado evento sísmico modelado possam estar mais próximos de um outro evento sísmico no dado observado ao invés de seus verdadeiros pares. Assim, no cálculo da função custo, a comparação será feita entre eventos que não são verdadeiramente correlacionados resultando num mínimo local que pode prejudicar a busca por um modelo mais verdadeiro. Este fenômeno é conhecido pelo termo *salto de ciclo* (Virieux e Operto, 2009), e pode ser evitado por meio de uma abordagem multiescala da FWI (Bunks et al., 1995).

A FWI multiescala consiste de modelagens em frequências crescentes — e portanto com escalas espaciais decrescentes — as quais são comparadas com os dados filtrados com passa-baixa de frequência máxima equivalente. Desta forma evita-se que a inversão comece a otimizar determinado evento antes de localizar sua origem no campo de velocidades corretamente. Para cada banda de frequência, uma série de iterações do método gradiente escolhido são realizadas considerando dados filtrados. A fonte utilizada na modelagen, por sua vez, deve sempre possuir conteúdo de frequência compatível com os dados em uso.

O esquema multiescala também é aplicável à FWI formulada como treinamento de uma rede recorrente. Para tal, treina-se os parâmetros da rede uma vez para cada escala mantendo sempre os parâmetros treinados sob a banda de frequência anterior como parâmetros iniciais da próxima.

# Capítulo 5

## Metodologia e Resultados

### 5.1 Metodologia

A metodologia deste trabalho envolve a implementação de uma rede recorrente de Elman que incorpora a equação da onda acústica em sua formulação por meio da técnica das diferenças finitas e é capaz de realizar modelagem. Para isto o modelo descrito na seção 4.2 foi implementado na linguagem Python. Num primeiro momento, foi utilizada a biblioteca de diferenciação automática e aceleração em diferentes unidades de processamento *JAX* (Bradbury et al., 2018), entretanto após a implementação da modelagem direta foi percebido um *bug* da biblioteca na autodiferenciação reversa da operação de convolução, a qual é utilizada no cálculo do Laplaciano durante a propagação da onda<sup>1</sup>, o que zerava as saídas do gradiente fora da região dos receptores. Uma nova implementação foi então realizada utilizando as ferramentas da biblioteca *TensorFlow* (Abadi et al., 2015), a qual obteve êxito na autodiferenciação reversa.

Tal rede implementada possui como parâmetro um modelo de velocidades de propagação da onda acústica em forma matricial. Em posse da rede modeladora, escolheu-se o modelo Marmousi (Figura 5.1) com 187 amostras de profundidade e 184 na horizontal para gerar os dados sintéticos. O modelo utilizado possui escala duas vezes menor que a original (Brougois et al., 1990), pois as operações de modelagem e diferenciação reversa exigem mais poder computacional e memória do que as disponíveis durante a realização de testes em tempo hábil para a escrita deste material. Para a realização da modelagem de 31 sismogramas cujos tiros foram equiespaçados pela superfície do modelo. Os espaçamentos vertical e horizontal entre nós foram respectivamente iguais a 16 e 50 m. A assinatura da fonte escolhida para os tiros foi uma onda Ricker de frequência de pico igual a 8 Hz, a maior frequência modelável para os espaçamentos da malha de diferenças finitas pelo critério dado em Wu et al. (1996), enquanto que o intervalo  $\Delta t$  de amostragem dos receptores foi de 4 ms, com número de amostras suficientes para a modelagem de 4 s de dados. Foi considerada a geometria de aquisição sísmica *split-spread* com afastamento mínimo igual a 500 m, afastamento máximo dado pelos limites do modelo, e afastamento entre receptores igual a 50 m.

A onda direta não é útil à inversão em profundidade, uma vez que a mesma não interage com os volumes mais profundos do modelo. Infelizmente este evento carrega o maior conteúdo de energia do dado sísmico. Inverter um dado com esta característica pode levar o modelo a mínimos locais da função custo, como já relatado por Koehne (2018). Isto também foi experienciado durante a elaboração dos experimentos deste trabalho. Uma solução

---

<sup>1</sup>Ver discussão em <https://github.com/google/jax/discussions/8411>.

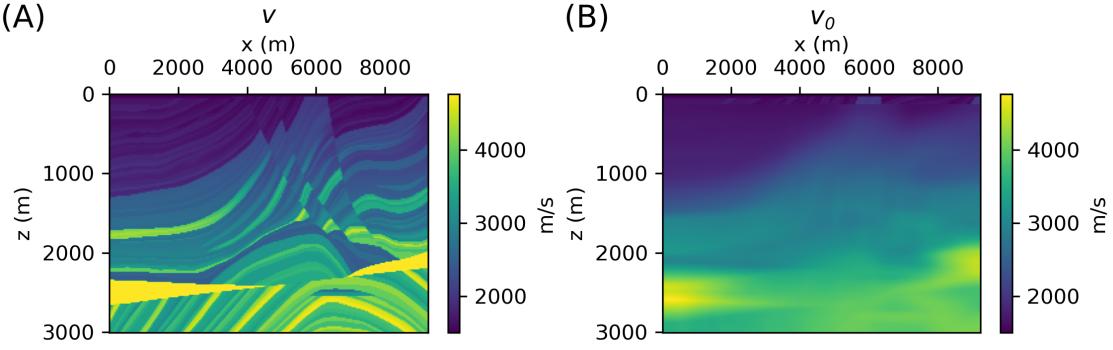


Figura 5.1: Modelo Marmousi real utilizado na produção dos dados sintéticos (A) e modelo borrado utilizado como ponto inicial da para o treinamento da rede (B).

comum para este problema é a retirada da onda direta dos sismograma gerado por meio do silenciamento de suas amplitudes. Em dados reais, isto pode exigir uma etapa manual de seleção destes eventos, mas como dados sintéticos foram aqui utilizados, a forma mais simples de produzir tal silenciamento é subtraindo dos sismogramas modelados, outros sismogramas, estes simulados usando um modelo de velocidade constante cujo valor seja igual à velocidade na primeira camada, a qual possui apenas informações superficiais — e que portanto só gerará as ondas diretas. Alguns sismogramas das modelagens sem onda direta realizados podem ser vistos na Figura 5.3, enquanto que um exemplo de silenciamento de dados sintéticos como o relatado é mostrado na Figura 5.2 para um modelo de duas interfaces.

Os dados calculados com o modelo real foram então passados à categoria de dados reais observados em campo, para que se pudesse testar a capacidade de inversão por treinamento da rede recorrente. Nesta nova etapa, considerou-se um modelo inicial filtrado por meio de uma média móvel com janela de tamanho igual a  $30 \times 30$  sobre o modelo verdadeiro, cujo resultado é uma versão de baixa resolução do modelo real e pode ser aludido a um modelo obtido por uma prévia análise de velocidades do processamento sísmico tradicional. O modelo Marmousi real e o inicial para a inversão são vistos na Figura 5.1. Mesmo tomando as precauções com a onda direta descrita há pouco, valores muito intensos foram encontrados na parte superficial dos gradientes calculados, os quais instabilizavam a otimização do modelo. Por isso, os gradientes calculados nas primeiras 8 amostras do modelo foram zerados ao longo da inversão. Esta região não foi alterada durante a geração do modelo inicial, e considerada conhecimento *a priori* durante a inversão.

Para a fase de inversão foram testados os algoritmos SGD, Momento e Adam para a atualização dos parâmetros, utilizando um tiro por lote. Para eles, o MSE foi utilizado como função custo, e seu gradiente foi calculado por diferenciação automática reversa. Além disto, três quartos dos dados (24 tiros) foram utilizados para o treinamento da rede, enquanto que o restante deles foi aleatoriamente escolhido para validação dos resultados. A partir desta separação, utilizou-se uma estratégia de interrupção precoce como a descrita na Seção 1.5 para controle, de forma a evitar o sobreajuste do modelo. Nela o critério de paciência utilizado foi de 6 épocas, de forma que, caso não seja vista melhora na função custo sobre os tiros de validação após tal período, a otimização é cessada. A modelagem direta utilizada na inversão já incorpora o silenciamento da onda direta da mesma maneira utilizada na geração dos dados sintéticos.

A abordagem multiescala de inversão foi adotada. Para tal, foram consideradas três

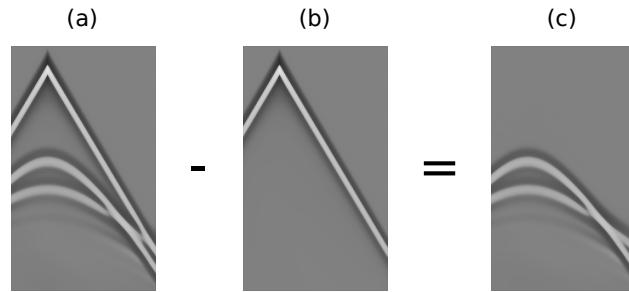


Figura 5.2: Modelagem de sismograma num modelo de duas interfaces em (a), modelagem de onda direta do mesmo tiro em (b) e subtração de (a) por (b) em (c), na qual se apresentam apenas os eventos de reflexão e refração.

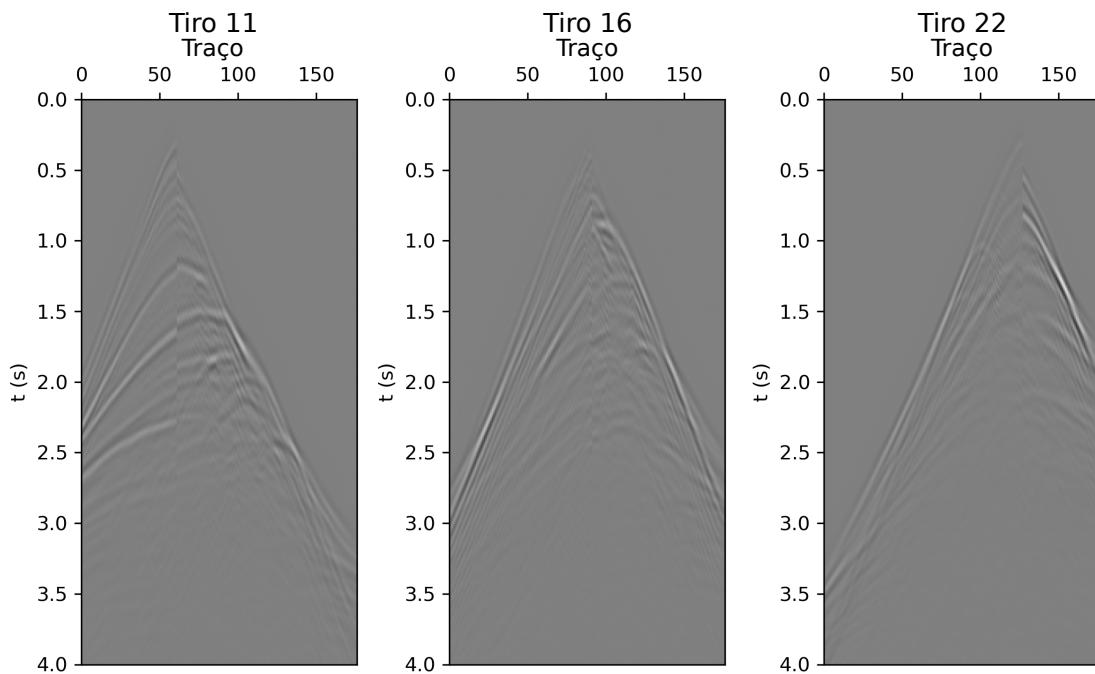


Figura 5.3: Da esquerda para a direita se apresentam os sismogramas: do décimo primeiro tiro, do décimo sexto tiro, e do vigésimo segundo tiro.

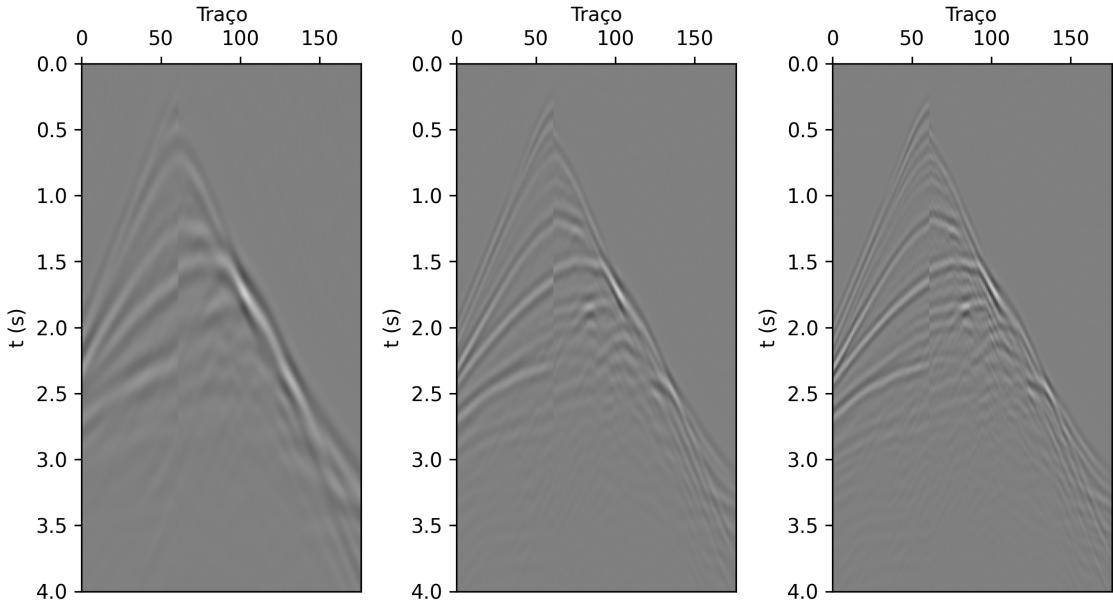


Figura 5.4: Mesmo sismograma modelado com fontes de frequência: 2,7 Hz, 5,3 Hz, e 8 Hz, da esquerda para a direita. Observe como algumas feições do sismograma das frequências mais altas não são visíveis em baixas frequências.

bandas de frequência compreendidas de 0 até respectivamente: 2.7, 5.3 e 8 Hz. O sismograma do tiro 11 pode ser visto em suas três bandas de frequências na Figura 5.4. Para cada banda de frequência a estratégia de interrupção precoce foi reinicializada e os parâmetros inicializados a partir do melhor modelo obtido durante a banda de frequência anterior, e apenas um valor máximo de 30 épocas de otimização eram permitidas.

## 5.2 Resultados da inversão

Um fator muito importante a ser selecionado durante a inversão por métodos baseados em gradiente é o seu passo ou taxa de aprendizado,  $\eta$ , o qual varia a depender do problema e do método de otimização. Nos métodos com mais parâmetros, como o Momento e o Adam, o melhor passo varia também com seus outros parâmetros de otimização. Uma taxa de aprendizado nula não atualiza o modelo, uma pequena pode levar a mínimos locais da função custo, enquanto que valores muito grandes podem fazer os parâmetros divergirem. Neste trabalho, bons valores para  $\eta$  foram obtidos por tentativa e erro em cada caso. Os outros parâmetros de otimização, tanto do algoritmo Momento ( $\alpha$ ), quanto do Adam ( $\beta_1$  e  $\beta_2$ ), foram mantidos com seus valores tradicionais, uma vez que a otimização não melhorou com suas modificações. Assim, os seguintes parâmetros foram utilizados nos métodos testados:

- SGD:  $\eta = 10^6$ ;
- Momento:  $\eta = 10^6$ ,  $\alpha = 0,9$ ; e
- Adam:  $\eta = 10$ ,  $\beta_1 = 0,9$ ,  $\beta_2 = 0,999$ .

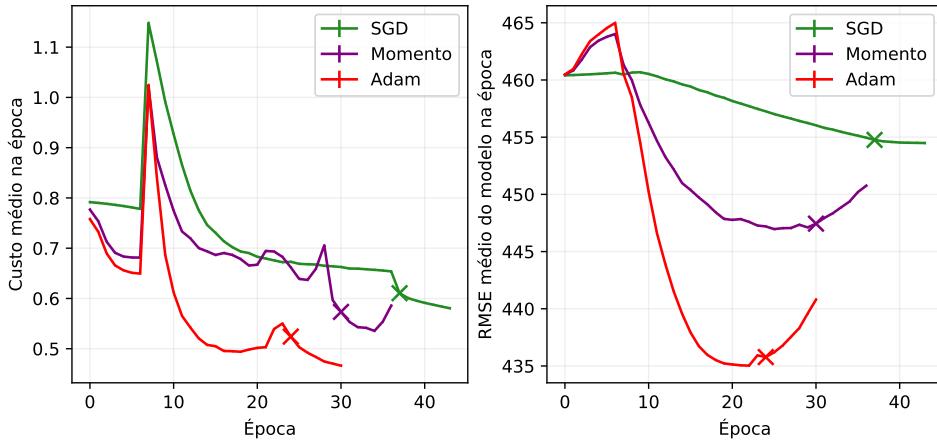


Figura 5.5: Função custo (esquerda) e erro médio (direita) do modelo em função das épocas de otimização para cada modelo.

As curvas de aprendizado utilizando cada método podem ser vistas em função das épocas de iteração na Figura 5.5. Um ponto importante no gráfico do custo médio por época são os saltos de comportamento nas curvas, os quais valem uma importante discussão. Estes pontos coincidem com o aumento da frequência de modelagem, e, portanto, com o aumento da frequência dos dados envolvidos na função custo, algo que os algoritmos de otimização de uso comum não levam em consideração em sua teoria. Felizmente, existem abordagens automatizadas hoje para lidar com este problema: os autores Bernal-Romero e Iturrarán-Viveros (2021) propõem uma forma de lidar com este problema por meio da variação da taxa de aprendizado em função da época de otimização e, principalmente, das bandas de frequência utilizadas na FWI. Esta faz parte de uma classe de soluções chamada de agendamento (do inglês, *scheduling*), entretanto este tema não foi tratado no presente material.

Outro ponto importante na Figura 5.5 é o papel fundamental da interrupção precoce no controle da FWI. Nas curvas da função custo média, podemos perceber duas discontinuidades, as quais estão ligadas à variação de escala da FWI. Podemos apontar no gráfico, que para cada nova frequência, a função custo decai com as épocas até determinado valor, mas mesmo quando ainda decaindo, a otimização é parada, o que especialmente visível durante o primeira e última bandas de frequência utilizadas. O critério de parada da otimização explica: para cada parada, a função custo sobre os tiros de controle não decai ou até mesmo aumenta pelas últimas 6 iterações. A função custo representa de forma preliminar o desejo de obter um modelo cujas saídas sejam similares às da realidade, mas nosso verdadeira vontade é minimizar o erro entre o modelo que possuímos e o modelo real. Deste modo, uma vantagem em utilizarmos dados sintéticos para testar algoritmos de inversão é que sabemos a resposta e podemos calcular o erro do modelo diretamente. Com base nisto, vemos no segundo gráfico da mesma figura, que para cada vez que uma frequência foi interrompida, o erro entre modelos estava aumentando — mesmo que a função custo de treinamento estivesse decaindo. Tiros de controle e a utilização da interrupção precoce evitam o sobreajuste na FWI, assim como nos métodos de aprendizado de máquina supervisionados.

Nas Figuras 5.6 e 5.7 podemos ver, respectivamente, o resultado da última iteração dos

métodos de otimização testados nas duas primeiras bandas de frequências da FWI multiescala realizada por meio do treinamento da rede recorrente, enquanto que na Figura 5.8, podemos ver o melhor modelo obtido na última banda de frequência da inversão. Na Figura 5.9 são mostradas as diferenças entre o modelo final obtido e o modelo inicial. De forma geral, o SGD consegue recuperar razoavelmente as velocidades mais próximas à superfície, convergindo em 43 épocas. O método do Momento atravessa espaço de modelos com inércia das otimizações passadas e consegue inverter melhor nas profundidades médias a altas após 34 épocas. O método Adam supera os resultados dos outros dois métodos com o menor número de épocas necessárias para convergência. Em 30 épocas, o método consegue o menor valor da função custo e menor diferença com relação aos parâmetros reais. No método Momento e no Adam, é possível imagear sensivelmente as camadas mais profundas do modelo, aos 2,700 metros sob a superfície.

A análise do perfil vertical de velocidades em  $x = 2500$  m mostrado na Figura 5.10 demonstra que as informações de mais alta frequência foram melhor identificadas utilizando os métodos Momento e Adam, sendo que o último obteve os valores de velocidade mais próximas do modelo verdadeiro.

Em todas as inversões, mas em especial no caso dos otimizadores Adam e Momento, são visíveis artefatos de maior velocidade e até mesmo a presença de bandamentos com orientação contrária à direção de acamamento das estruturas geológicas reais. Este problema pode ser devido à baixa quantidade de tiros utilizada na inversão, mas uma investigação mais aprofundada deve ser feita para uma maior compreensão do problema.

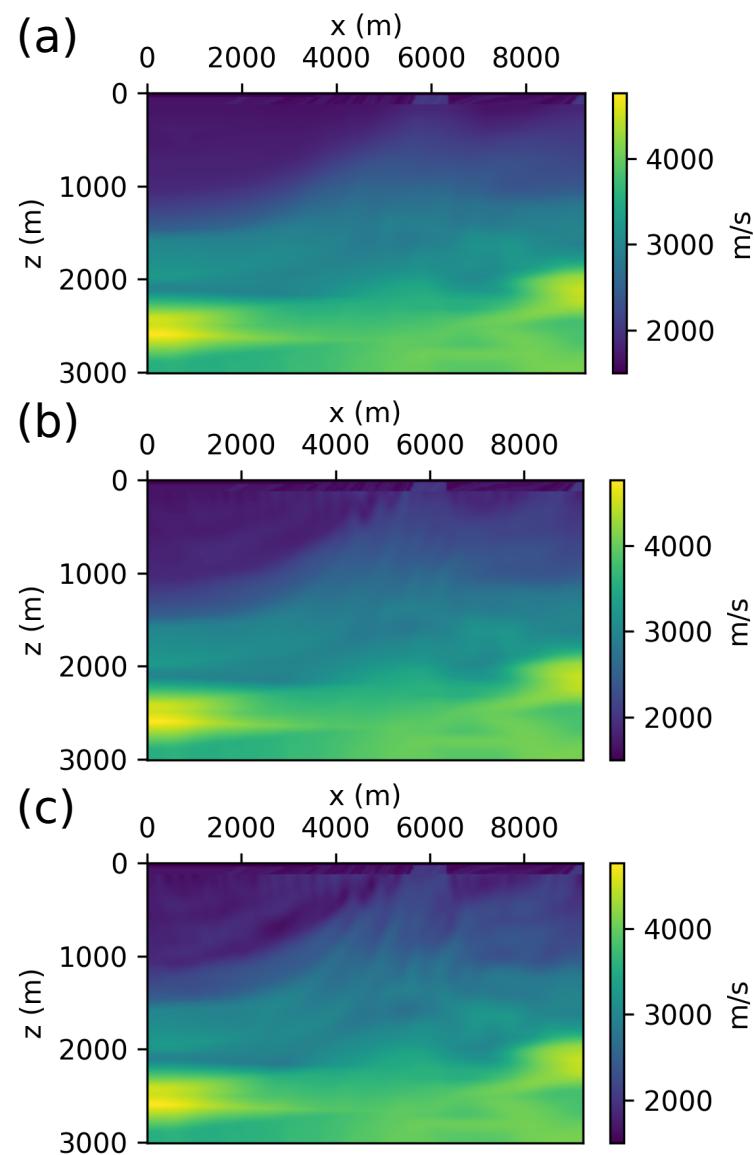


Figura 5.6: Modelo obtido na última época de otimização durante a etapa de modelagem sob a banda de frequência de 0 a 2.7 Hz usando SGD (a), Momento (b) e Adam (c).

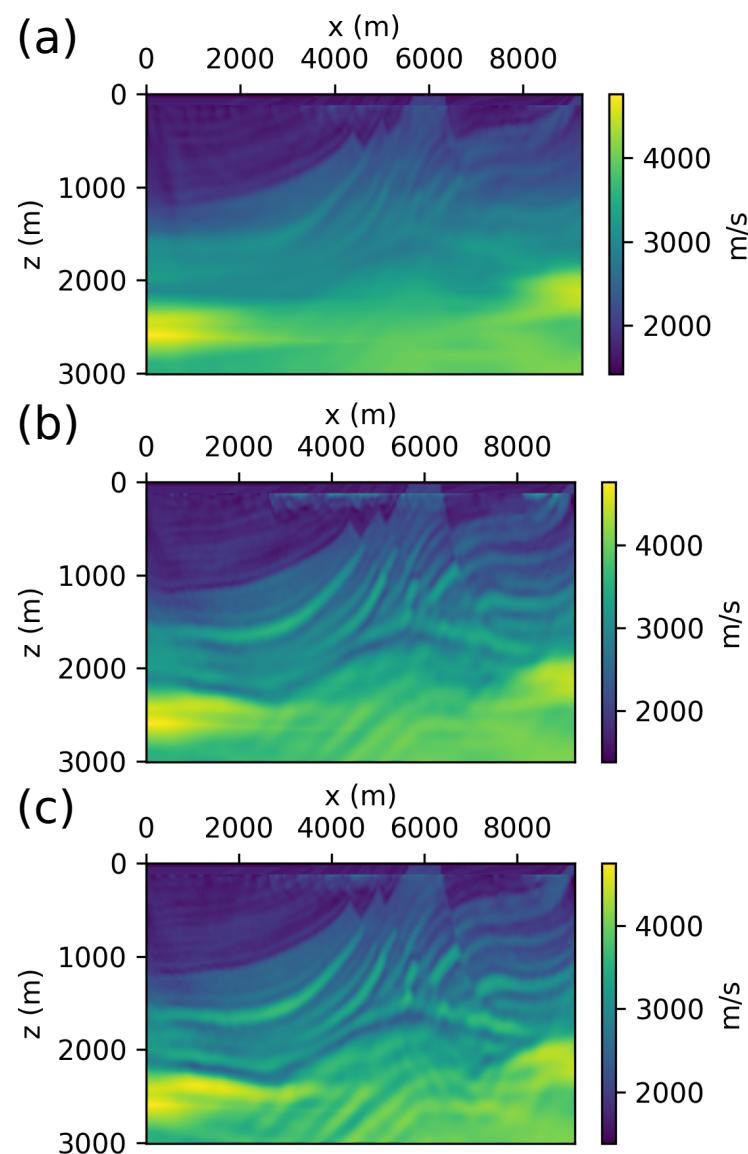


Figura 5.7: Modelo obtido na última época de otimização durante a etapa de modelagem sob a banda de frequência de 0 a 5.3 Hz usando SGD (a), Momento (b) e Adam (c).

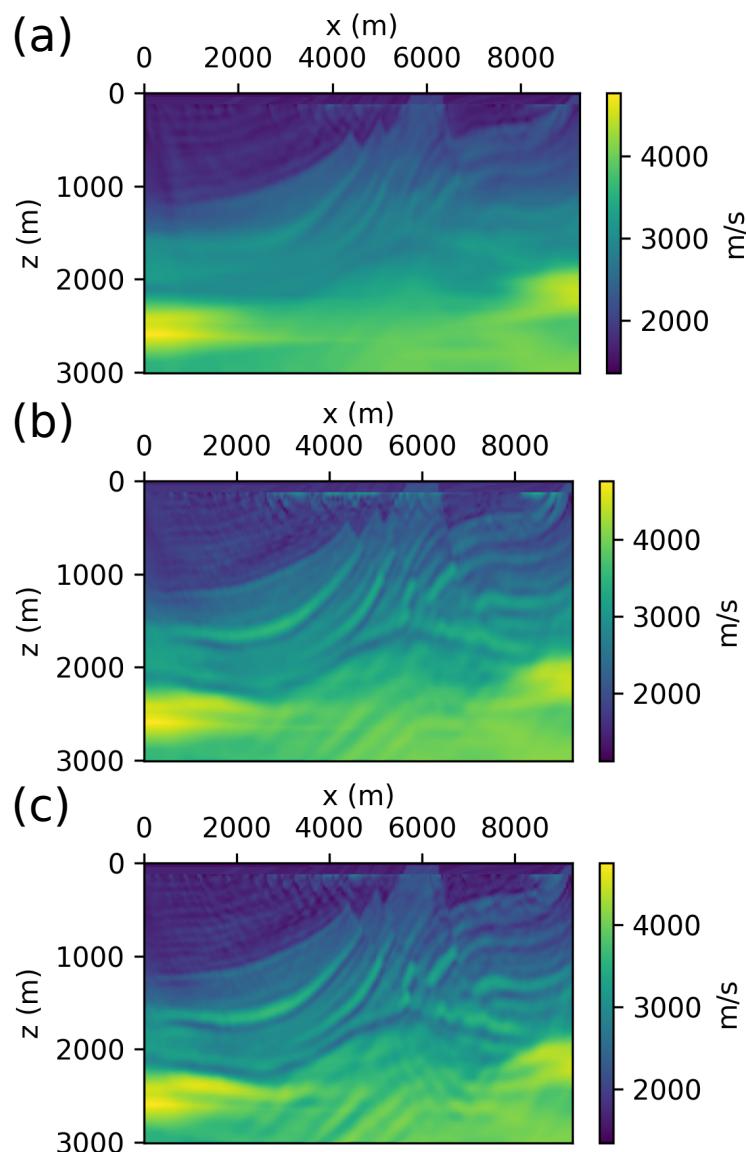


Figura 5.8: Melhor modelo obtido durante a otimização na última banda de frequência (0 a 8 Hz) usando SGD (a), Momento (b) e Adam (c).

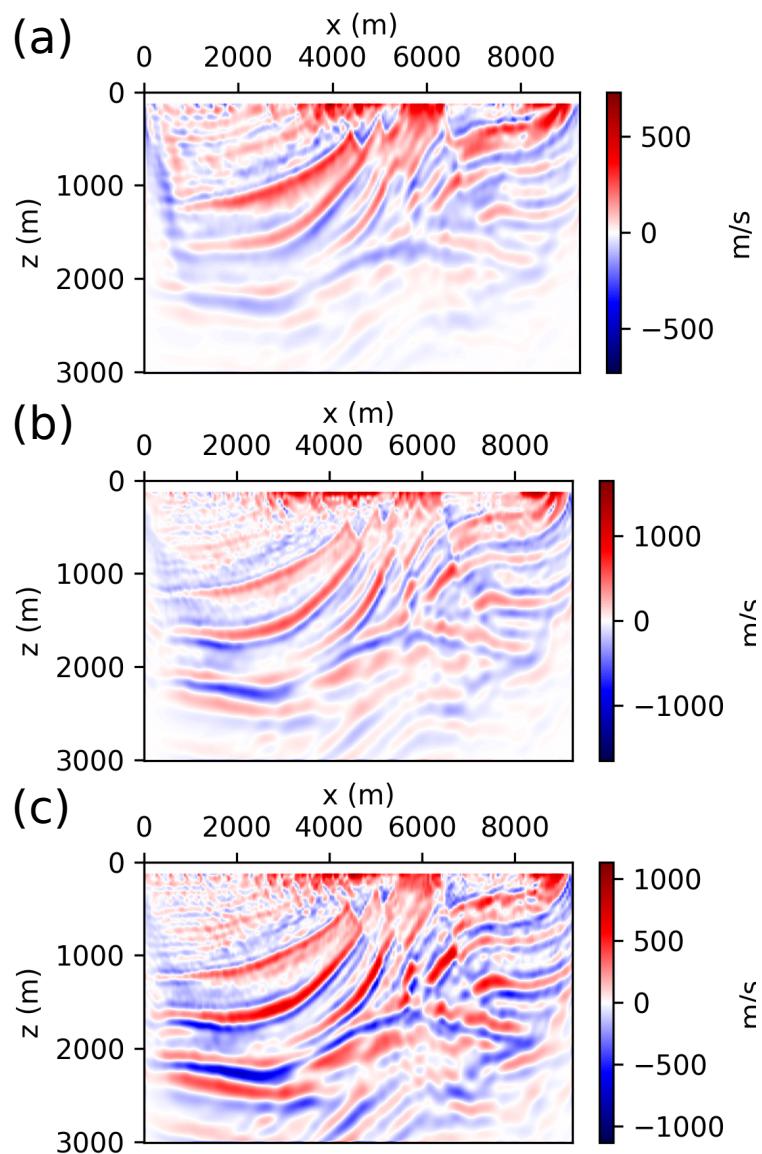


Figura 5.9: Diferença entre o melhor modelo obtido na FWI multiescala e o modelo inicial para os métodos: SGD (a), Momento (b) e Adam (c).

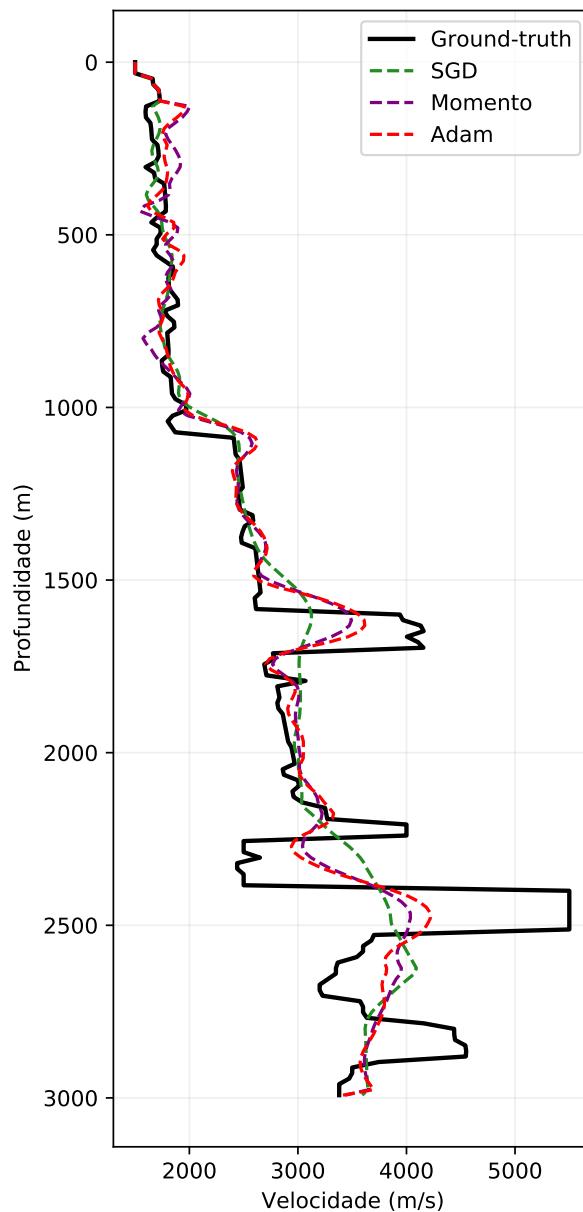


Figura 5.10: Perfil vertical de velocidades em  $x = 2.500$  m para o modelo verdadeiro (preto) e os modelos resultantes da FWI por meio do método SGD (tracejada em verde), Momento (tracejada em lilás) e Adam (tracejada em vermelho).

# Capítulo 6

## Conclusões

Neste material foram introduzidos os conceitos básicos de aprendizado estatístico, seus métodos de otimização e a extensão de seus modelos mais simples às redes neurais. A extensão da rede neural recorrente de Elman para um modelo capaz de realizar simulações acústicas foi teorizada e implementada, gerando resultados satisfatórios de modelagem e inversão utilizando três otimizadores típicos do universo das redes neurais: SGD, Momento e Adam. Problemas de otimização multi-frequência devem ser corrigidos em trabalhos futuros por meio da implementação de técnicas de agendamento no processo de otimização, as quais são tidas como tecnologias relativamente avançadas de treinamento de redes. Uma pequena inadequação próxima à superfície dos modelos invertidos foi encontrada, a qual é uma suposta consequência do uso de poucos tiros para a inversão, mas que deve ser alvo de futuras investigações. É necessário salientar que, apesar da possibilidade de realização da FWI pelo método aqui apresentado, a demanda por memória no cálculo dos gradientes se torna superior à comumente utilizada em casos de aplicação real. Em casos similares, a solução do método adjunto pode permitir a implementação de otimizações valorosas.

Do ponto de vista da modelagem tradicional, a generalidade do método empregado abre janelas à pesquisa neste e em outros problemas físicos, como a dispersão de poluentes em fluidos, a dissipação de calor ou a modelagem de deformações numa estrutura mecânica, uma vez que a inversão aqui realizada acontece por diferenciação automática, sem mesmo o uso explícito do método adjunto. Este ponto é muito valoroso, pois o método adjunto exige uma etapa de pré-formulação de sua teoria com relação ao problema estudado da qual sua implementação depende, enquanto que a inversão por diferenciação automática é de implementação imediata para qualquer problema que pode ser expresso num grafo de computações comuns com diferenciais conhecidas. Uma contribuição interessante vinda do contexto das redes neurais é a de interrupção precoce, técnica que permitiu uma inversão controlada na FWI aqui implementada. Desta forma, ao invés de um número fixo de iterações de otimização, um critério de parada baseado na modelagem sobre tiros de validação do campo de velocidades otimizado foi implementado.

Do ponto de vista das redes neurais, este trabalho constrói uma rede com bons resultados e parâmetros facilmente interpretáveis — as velocidades em subsuperfície. A junção da área das redes com a da inversão acústica traz uma ideia não muito comum ao treinamento das redes: é possível que um treinamento multiescala seja útil também para o treinamento de outras redes neurais. Esta ideia é diretamente aplicável a redes de segmentação semântica, por exemplo. Vale citarmos que a metodologia das redes neurais informadas de física — isto é, a penalização de insatisfações de equações físicas dentro da função custo utilizada

na inversão — é um próximo passo a ser introduzido ao modelo físico desenvolvido neste material e tratado em trabalhos futuros.

Creio que este material é adequado à leitura por estudantes de Geofísica que desejem ser introduzidos ao universo do aprendizado de máquina, das redes neurais ou à diferenciação automática, ou ao mesmo, aos que desejem revisar seus tópicos básicos, os quais foram essenciais à implementação alvo deste trabalho.

# Agradecimentos

Este trabalho é resultado de um esforço muito superior ao que eu poderia compor sozinho. Neste espaço tentarei elencar aqueles que mais diretamente estão ligados ao meu percurso universitário. Sei que faltarei em lembrar de todos e peço perdão de antemão por isto.

Em primeiro lugar, eu agradeço aos meus pais, por se esforçarem para construir uma família estruturada e por nos serem excelentes exemplos. Agradeço à minha mãe, Antonia Conceição, por me incentivar na caminhada acadêmica e por vibrar com cada conquista minha. Agradeço ao meu pai, Everaldo da Conceição, por seu jeito de educar agindo, sobretudo a ser mais paciente com as fases das pessoas e com minhas próprias falhas. Não tenho dúvida que vocês fizerem todo sacrifício possível por mim e pelos meus irmãos.

Agradeço a Jamile Conceição, minha irmã, por ser meu espelho desde pequeno e a Bruno Conceição, meu irmão caçula, por se empolgar junto comigo e ser boa companhia. Estive triste em cada momento que concluí o dever de abdicar da presença de vocês e de meus pais ao longo deste trajeto, bem como em cada um dos momentos em que não pude contribuir de forma adequada com a nossa instituição.

Agradeço imensamente ao meu orientador do trabalho de graduação, Reynam Pestana. Meu ritmo de aprendizado nunca foi tão intenso quanto como nestes últimos anos em que tentei te acompanhar. Obrigado pela paciência, direcionamentos e sermões. Mais de uma vez, seus “vocês têm que estudar isto aqui...” mudaram minha forma de pensar sobre o mundo. Agradeço também a Átila Saraiva, meu grande amigo e segundo orientador neste trabalho, quem me introduziu à inversão, à computação de alto desempenho e à migração reversa no tempo, bem como me apresentou Reynam. Reynam dizia “vocês”, pois eu e meus dois orientadores nos reuníamos semanalmente numa espécie de grupo de estudos, o que foi muito inspirador e tecnicamente amadurecedor para mim. Átila foi muito paciente e preocupado com minha evolução e responsabilização por mim mesmo, desde o IFBA, além de um dos melhores professores que eu já tive!

Victor Koehne me auxiliou bastante a entender a aplicação das diferenças finitas, e sou muito grato por ele se mostrar tão disposto. Ver sua apresentação num evento do capítulo de Geofísica alguns anos atrás foi bem motivador para mim.

Agradeço de maneira especial ao professor Luis Felipe de Mendonça, um orientador incrível e um dos melhores seres humanos que eu já conheci. Felipe nunca subestima seus alunos, e trabalhar com ele é energizante como um bom chimarrão! Muito obrigado pela liberdade, pela empolgação pela vida e pelo apoio! Aproveito para agradecer ao meu co-orientador de pesquisa, o professor Carlos Lentini, que me abriu os olhos sobre o funcionamento da academia e sempre se mostrou muito disposto a ajudar aos meus colegas e mim. Agradeço também aos meus companheiros do Laboratório de Oceanografia por Satélites (LOS) por estes dois anos de conquistas incríveis.

Wilson Figueiró é o professor mais didático que eu já tive e sou muito grato por ter sido seu aluno dentro desta casa. Figueiró mostra que é possível ser extremamente embasado

e aplicado ao mesmo tempo, e é sem sombra de dúvida uma referência para o Marcos de hoje. Além disso, a escrita dos meus primeiros artigos foram sob sua orientação. Agradeço também ao professor Hédison Sato, tanto pela qualidade das aulas, quanto pela preocupação e puxões de orelha que sempre calharam de vir em momentos necessários.

Eu não sei se viria a trabalhar com sísmica se eu não me apaixonasse pela matéria de ondas durante o curso de Física 2, que foi ministrada pelo professor Thiago Albuquerque. Obrigado por não subestimar seus alunos, Thiago! Agradeço também ao professor Tiago Paes, o qual foi meu primeiro orientador numa bolsa de extensão, e me deu liberdade para investir um tempo estudando campos de pressão, além de sempre ter sido bem solícito. Muito do que aprendi nesse período foi utilizado nos meus estudos de sísmica mais para frente.

Tenho muito a agradecer ao professor Alexsandro Cerqueira. Alex é muito responsável, um professor exímio e um exemplo de ser humano. Aprendi muito do que já tinha desistido sozinho sob a tutela dele. Fico feliz de considerá-lo hoje um grande amigo também! Todo o resto da turma de 2017 — Matheus Radamés e Lorena da Silva — são seus orientandos jutamente a Jonh Brian, que também foi meu colega. Estas são pessoas às quais eu admiro em demasia, e cuja companhia foi sempre muito agradável.

Agradeço enormemente a Gabriela Brito, minha companheira, pela paciência e apoio nos momentos difíceis e pela vibração cada vitória. Não consigo imaginar minha caminhada até aqui sem você por perto. Agradeço também aos seus pais, Daniela Brito e Elviton dos Santos, pela compreensão e pelas conversas!

Gostaria de agradecer enormemente a Lucidalva de Assis, minha tia e primeira professora. Seus desafios foram essenciais para que eu me afeiçoaasse à Matemática. Tia Luci sempre me incentivou de todas as maneiras possíveis a evoluir. Espero poder retribuir tanto amor! Aproveito para agradecer aos meus tios Ronaldo, Ricardo, Cristina e Maria do Carmo de Assis por me darem exemplo no estudo.

Gostaria de agradecer à minha prima, Manuela Barreto, bem como a Bruno Bulhões, pelas conversas importantes, pelo exemplo nas ações e pela leveza. Apesar da pequena frequência, conversar com Manu sempre virou chaves na minha cabeça, desde minha primeira infância.

Agradeço a todo o Capítulo Estudantil de Geofísica da UFBA. Com certeza não teria entrado na modelagem, não fosse a presença do Capítulo na minha vida. Agradeço particularmente pelas conversas com Mariana Sampaio e Juliana Diniz, as quais são ídolos para mim e me estabilizaram ao longo do curso. Agradeço enormemente a Letícia Pires por sua responsabilidade e humanidade, bem como por me ajudar tanto no período em que me sobrecreguei. Devo muito a você, Leti!

Sou muito grato ao meu amigo Felipe Duarte. Felipe me aguentou nos meus momentos mais caóticos e sempre me trouxe sentido e calmaria. Não tenho como retribuir o que ele fez por mim. Uma pequena versão sua existe para sempre entre os personagens da minha consciência.

Aproveito para agradecer pela presença de Felipe Heleno Borges pela amizade desde a adolescência e por Zanoni Neto, meu amigo de infância. Obrigado por estarem sempre prontos para ajudar minhas versões passadas. Eu amo vocês. Agradeço às incríveis outras pessoas de Ardur também.

Mário Oliveira Luciem foi quem plantou em mim a vontade de ser pesquisador e quem me convenceu das possibilidades. Quem me incentivou a gostar de Física e a me tranquilizou desde a adolescência. Mário foi um dos meus maiores educadores. Muito obrigado, Mário!

Gostaria de agradecer a Victor Said e Walter G. Neves por serem tão universais, por

sempre me ofertarem paz e estarem abertos para ouvir e criticar minhas ideias. Meu percurso teria sido muito mais tortuoso sem os seus conselhos.

Agradeço a Amanda Oliveira e a Ingrid Queiroz pela amizade e pela paciência nos meus períodos mais difíceis. Vocês me ensinaram a dar o devido valor à vida.

Sou bastante grato aos meus amigos Milena Leite e Igor Moraes, pessoas que me estabilizaram em diversos momentos e que sempre me incentivaram a ser melhor.

Por fim, agradeço à minha avó Gildete Assis e, de maneira especial, à minha avó Maria Emilia Santos, que faleceu neste ano. A valorização do conhecimento na minha família, e portanto a possibilidade da vida que posso, provêm diretamente da visão e esforço destas mulheres.

## Apêndice A

# Derivação do cálculo do gradiente da rede de Elman para modelagem acústica por diferenciação automática

Considere a inversão sísmica de dados obtidos por um receptor, e cujas amplitudes são geradas por uma fonte pontual. Neste contexto, deduziremos o cálculo do gradiente de um modelo de velocidades parametrizado na matriz  $\mathbf{V}$  utilizando da equação acústica da onda e da função custo de meio erro quadrático  $c$  entre as amplitudes modeladas  $\hat{y}_t$  e medidas  $y_t$  para cada tempo  $t$ . Desenvolvimentos semelhantes são encontrados nos trabalhos de Richardson (2018), Sun et al. (2019) e Ren et al. (2020). O caso elástico é discutido por Wang et al. (2021). Nos termos descritos, temos:

$$c(\mathbf{V}) = \frac{1}{2} \sum_{t=1}^T e_t^2 = \frac{1}{2} \sum_{t=1}^T [\hat{y}_t - y_t]^2 = \frac{1}{2} \sum_{t=1}^T \left[ \sum_{\vec{r} \in \mathcal{D}} \delta_{\vec{r}, \vec{r}_r} (\hat{\mathbf{P}}_t - y_t) \right]^2, \quad (\text{A.1})$$

onde  $\hat{\mathbf{P}}_t$  é o campo modelado no tempo  $t$  e  $\delta_{\vec{r}, \vec{r}_r}$  é o delta de Kronecker que zera os valores de todos os índices, que não os presentes na posição do receptor  $\vec{r}_r$  da matriz à qual é aplicado. Portanto, o somatório da matriz esparsa  $\delta_{\vec{r}, \vec{r}_r} (\hat{\mathbf{P}}_t - y_t)$  é exatamente o valor do seu único elemento não nulo,  $\hat{y}_t - y_t$ . Esta escrita facilitará a diferenciação numa etapa posterior. Podemos então diferenciar parcialmente a função custo com relação ao campo de velocidades para calcularmos seu gradiente:

$$\frac{\partial c}{\partial \mathbf{V}} = \sum_{t=1}^T \frac{\partial c}{\partial \hat{\mathbf{P}}_t} \Big|_{\mathbf{V}} \frac{\partial \hat{\mathbf{P}}_t}{\partial \mathbf{V}} \Big|_{\mathbf{P}_{t' \neq t}}. \quad (\text{A.2})$$

Por sua vez, num esquema de diferenças finitas de segunda ordem no tempo,  $\hat{\mathbf{P}}_t$  é dado por:

$$\hat{\mathbf{P}}_t = \mathbf{V}^2 \Delta t^2 (\nabla^2 \hat{\mathbf{P}}_{t-1} + s_t \boldsymbol{\delta}_{\vec{r}, \vec{r}_s}) + 2\hat{\mathbf{P}}_{t-1} - \hat{\mathbf{P}}_{t-2}, \quad (\text{A.3})$$

onde  $\Delta t$  é o menor intervalo de tempo utilizado na modelagem,  $s_t$  e  $\vec{r}_s$  são respectivamente a amplitude atual e a posição da fonte, e  $\boldsymbol{\delta}_{\vec{r}, \vec{r}_s} = \delta_{\vec{r}, \vec{r}_s} \mathbf{1}$  é a matriz esparsa cujo único valor não-nulo vale um e se localiza na posição da fonte. Deste modo, função de propagação no tempo pode ser dada por  $f$  na equação a seguir:

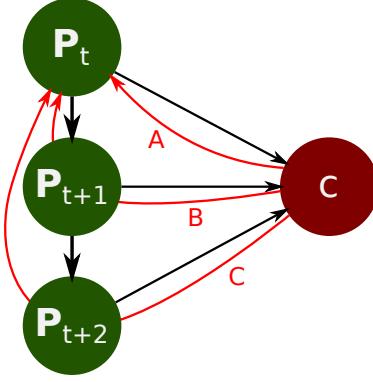


Figura A.1: Grafo da dependência (setas vermelhas) do custo  $c$  em relação ao campo de pressão  $P_t$ . Num esquema de diferenças finitas de segunda ordem no tempo, três caminhos vão do custo ao campo de pressão em um tempo  $t$ : o caminho  $A$  indica o erro gerado diretamente pela falta de fase entre o dado predito e o observado nos receptores no instante  $t$ ; o caminho  $B$  se refere ao erro propagado à pressão do instante seguinte ( $t + 1$ ) devido ao erro presente no instante  $t$ ; e o caminho  $C$  trata do erro causado à pressão em  $t + 2$  devido ao erro em  $P_t$ .

$$f(\mathbf{V}, \hat{\mathbf{P}}_{t-2}, \hat{\mathbf{P}}_{t-1}, \mathbf{S}) = \mathbf{V}^2 \Delta t^2 (\nabla^2 \hat{\mathbf{P}}_{t-1} + \mathbf{S}) + 2\hat{\mathbf{P}}_{t-1} - \hat{\mathbf{P}}_{t-2}, \quad (\text{A.4})$$

onde  $\mathbf{S}$  é a matriz do termo fonte.

A diferenciação parcial do custo em relação ao campo de pressão do tempo discreto  $t$  na equação A.2 pode ser destrinchada em três fatores pela regra da cadeia reversa (grafo na Figura A.1). Isto porque a modelagem por diferenças finitas de segunda ordem no tempo implica a propagação de um novo instante de pressão depender de outros dois instantes passados. Desta forma, o custo devido à pressão em um instante  $t$  depende não só dele, mas dos outros dois instantes futuros que dependem dele, o que num gráfico de diferenciação automática significa que um mesmo nó de pressão é encontrado por três setas.

$$\frac{\partial c}{\partial \hat{\mathbf{P}}_t} \Big|_{\mathbf{V}} = \frac{\partial c}{\partial \hat{\mathbf{P}}_t} \Big|_{\mathbf{V}, \mathbf{P}_{t' \neq t}} + \frac{\partial c}{\partial \hat{\mathbf{P}}_{t+1}} \Big|_{\mathbf{V}} \frac{\partial \hat{\mathbf{P}}_{t+1}}{\partial \hat{\mathbf{P}}_t} \Big|_{\mathbf{V}, \mathbf{P}_{t' \neq t, t+1}} + \frac{\partial c}{\partial \hat{\mathbf{P}}_{t+2}} \Big|_{\mathbf{V}} \frac{\partial \hat{\mathbf{P}}_{t+2}}{\partial \hat{\mathbf{P}}_t} \Big|_{\mathbf{V}, \mathbf{P}_{t' \neq t, t+2}}. \quad (\text{A.5})$$

O primeiro termo da equação (A.5) pode ser calculado com facilidade a partir da função custo:

$$\frac{\partial c}{\partial \hat{\mathbf{P}}_t} \Big|_{\mathbf{V}, \mathbf{P}_{t' \neq t}} = \frac{\partial}{\partial \hat{\mathbf{P}}_t} \left[ \frac{1}{2} \sum_{\vec{r} \in \mathcal{D}} \delta_{\vec{r}, \vec{r}_r} (\hat{\mathbf{P}}_t - y_t) \right]^2 \Big|_{\mathbf{V}, \mathbf{P}_{t' \neq t}} = \delta_{\vec{r}, \vec{r}_r} (\hat{\mathbf{P}}_t - y_t) = e_t \boldsymbol{\delta}_{\vec{r}, \vec{r}_r}, \quad (\text{A.6})$$

ou seja, seu valor é a amplitude do resíduo entre o dado modelado e o observado na posição do receptor,  $e_t$ . As equações (A.7), (A.8) e (A.9) podem ser calculadas a partir de (A.3).

$$\frac{\partial \hat{\mathbf{P}}_{t+1}}{\partial \hat{\mathbf{P}}_t} \Big|_{\mathbf{V}, \mathbf{P}_{t' \neq t, t+1}} = \mathbf{V}^2 \Delta t^2 \nabla^2 + 2 \quad (\text{A.7})$$

$$\frac{\partial \hat{\mathbf{P}}_{t+2}}{\partial \hat{\mathbf{P}}_t} \Big|_{\mathbf{V}, \mathbf{P}_{t' \neq t, t+2}} = -1 \quad (\text{A.8})$$

$$\frac{\partial \hat{\mathbf{P}}_t}{\partial \mathbf{V}} \Big|_{\mathbf{P}_{t' \neq t}} = 2\mathbf{V}\Delta t^2(\nabla^2 \hat{\mathbf{P}}_{t-1} + 2s_{t-1}\delta_{\vec{r}, \vec{r}_s}) \quad (\text{A.9})$$

Substituindo (A.5) em (A.2), temos:

$$\begin{aligned} \frac{\partial c}{\partial \mathbf{V}} &= \sum_{t=1}^T \left( \frac{\partial c}{\partial \hat{\mathbf{P}}_t} \Big|_{\mathbf{V}, \mathbf{P}_{t' \neq t}} \right. \\ &\quad + \frac{\partial c}{\partial \hat{\mathbf{P}}_{t+1}} \Big|_{\mathbf{V}} \frac{\partial \hat{\mathbf{P}}_{t+1}}{\partial \hat{\mathbf{P}}_t} \Big|_{\mathbf{V}, \mathbf{P}_{t' \neq t, t+1}} \\ &\quad \left. + \frac{\partial c}{\partial \hat{\mathbf{P}}_{t+2}} \Big|_{\mathbf{V}} \frac{\partial \hat{\mathbf{P}}_{t+2}}{\partial \hat{\mathbf{P}}_t} \Big|_{\mathbf{V}, \mathbf{P}_{t' \neq t, t+2}} \right) 2\mathbf{V}\Delta t^2(\nabla^2 \hat{\mathbf{P}}_{t-1} + 2s_{t-1}\delta_{\vec{r}, \vec{r}_s}) \end{aligned} . \quad (\text{A.10})$$

Sobrepondo então (A.6), (A.7) e (A.8) a (A.10), conseguimos:

$$\begin{aligned} \frac{\partial c}{\partial \mathbf{V}} &= \sum_{t=1}^T \left( e_t \delta_{\vec{r}, \vec{r}_r} + (\mathbf{V}^2 \Delta t^2 \nabla^2 + 2) \frac{\partial c}{\partial \hat{\mathbf{P}}_{t+1}} \Big|_{\mathbf{V}} - \frac{\partial \hat{\mathbf{P}}_{t+2}}{\partial \hat{\mathbf{P}}_t} \Big|_{\mathbf{V}, \mathbf{P}_{t' \neq t, t+2}} \right), \\ &\quad 2\mathbf{V}\Delta t^2(\nabla^2 \hat{\mathbf{P}}_{t-1} + 2s_{t-1}\delta_{\vec{r}, \vec{r}_s}) \end{aligned} \quad (\text{A.11})$$

Podemos rearrumar os termos de (A.11) da seguinte maneira:

$$\begin{aligned} \frac{\partial c}{\partial \mathbf{V}} &= \sum_{t=1}^T \left( \mathbf{V}^2 \Delta t^2 \left( \nabla^2 \frac{\partial c}{\partial \hat{\mathbf{P}}_{t+1}} \Big|_{\mathbf{V}} + \frac{e_t \delta_{\vec{r}, \vec{r}_r}}{\mathbf{V}^2 \Delta t^2} \right) + 2 \frac{\partial c}{\partial \hat{\mathbf{P}}_{t+1}} \Big|_{\mathbf{V}} - \frac{\partial c}{\partial \hat{\mathbf{P}}_{t+2}} \Big|_{\mathbf{V}} \right) \\ &\quad 2\mathbf{V}\Delta t^2(\nabla^2 \hat{\mathbf{P}}_{t-1} + 2s_{t-1}\delta_{\vec{r}, \vec{r}_s}) \end{aligned} \quad (\text{A.12})$$

Enxergamos em (A.12) o padrão da função de modelagem  $f$  no primeiro fator do somatório:

$$\frac{\partial c}{\partial \mathbf{V}} = \sum_{t=1}^T f \left( \mathbf{V}, \frac{\partial c}{\partial \hat{\mathbf{P}}_{t+2}} \Big|_{\mathbf{V}}, \frac{\partial c}{\partial \hat{\mathbf{P}}_{t+1}} \Big|_{\mathbf{V}}, \frac{e_t \delta_{\vec{r}, \vec{r}_r}}{\mathbf{V} \Delta t^2} \right) 2\mathbf{V}\Delta t^2(\nabla^2 \hat{\mathbf{P}}_{t-1} + 2s_{t-1}\delta_{\vec{r}, \vec{r}_s}). \quad (\text{A.13})$$

Já no segundo fator de (A.12) podemos ver a segunda derivada do campo primário, uma vez que:

$$\frac{\hat{\mathbf{P}}_t - 2\hat{\mathbf{P}}_{t-1} + \hat{\mathbf{P}}_{t-2}}{\Delta t^2} \approx \ddot{\hat{\mathbf{P}}}_{t-1} = \mathbf{V}^2(\nabla^2 \hat{\mathbf{P}}_{t-1} + s_{t-1}\delta_{\vec{r}, \vec{r}_s}), \quad (\text{A.14})$$

e portanto,

$$\nabla^2 \hat{\mathbf{P}}_{t-1} + s_{t-1}\delta_{\vec{r}, \vec{r}_s} = \frac{\ddot{\hat{\mathbf{P}}}_{t-1}}{\mathbf{V}^2}. \quad (\text{A.15})$$

Ficamos então com (A.16).

$$\frac{\partial c}{\partial \mathbf{V}} = \sum_{t=1}^T f \left( \mathbf{V}, \frac{\partial c}{\partial \hat{\mathbf{P}}_{t+2}} \Big|_{\mathbf{V}}, \frac{\partial c}{\partial \hat{\mathbf{P}}_{t+1}} \Big|_{\mathbf{V}}, \frac{e_t \delta_{\vec{r}, \vec{r}_r}}{\mathbf{V} \Delta t^2} \right) \frac{2}{\mathbf{V}} \Delta t^2 \ddot{\hat{\mathbf{P}}}_{t-1} \quad (\text{A.16})$$

Dada a linearidade da equação da onda a tempos pequenos, podemos por para fora os fatores do terceiro argumento de  $f$  e mover as variáveis independentes do tempo para fora do somatório para obter (A.17).

$$\nabla c \equiv \frac{\partial c}{\partial \mathbf{V}} = \frac{2}{\mathbf{V}^3} \sum_{t=1}^T f \left( \mathbf{V}, \frac{\partial c}{\partial \hat{\mathbf{P}}_{t+2}} \Big|_{\mathbf{V}}, \frac{\partial c}{\partial \hat{\mathbf{P}}_{t+1}} \Big|_{\mathbf{V}}, e_t \delta_{\vec{r}, \vec{r}_r} \right) \ddot{\hat{\mathbf{P}}}_{t-1} \quad (\text{A.17})$$

A equação (A.17) é exatamente a discretização da solução para o gradiente da função custo do erro quadrático dada pelo método adjunto: a função  $f$  está propagando o resíduo  $e_t$  reversamente no tempo — a extrapolação vai do tempo  $t + 2$  para  $t + 1$  para em seguida produzir  $t$  — a partir da posição do receptor,  $\vec{r}_r$ . O resultado desta operação é o campo adjunto,  $\lambda$ , que é multiplicado pela segunda derivada no tempo do campo modelado. Este resultado para uma única fonte é estendido a mais receptores somando suas contribuições na função custo, e a mais fontes, ao realizar a modelagem direta com mais fontes (Sun et al., 2019).

## Apêndice B

# Receita para o treinamento estocástico de um modelo supervisionado

Durante os estudos práticos de treinamento de modelos supervisionados, tais como o do modelo de regressão logística, a inversão de forma completa da onda implementada, o autor verificou um padrão estrutural para a implementação de códigos de otimização em um paradigma fortemente funcional, de maneira que os códigos se tornam mais facilmente reutilizáveis. Tal padrão pode ser utilizado para o desenvolvimento de qualquer modelo estatístico supervisionado ou de qualquer inversão geofísica, e é sumarizado em três fases:

1. Fase de *criação do modelo*, a qual envolve:
  - (a) Definição do(s) modelo(s) direto(s) — durante esta etapa deve-se implementar os problemas diretos necessários que levarão uma sequência de parâmetros  $\{W_s\}$  e uma matriz de amostras observadas  $\mathbf{X}$  a uma matriz de previsões  $\hat{\mathbf{Y}}$ . Este processo muitas vezes inclui o primeiro desenvolvimento de modelos que levam uma única amostra  $\vec{x}$  à sua previsão  $\vec{y}$ .
  - (b) Em caso de otimização conjunta de dois ou mais modelos diretos deve-se implementar um modelo amarrador que retorna as saídas finais que podem ser utilizadas no treinamento.
  - (c) Em caso de regularização dos parâmetros durante treinamento, pode ser desejável adicionar os parâmetros como saídas do modelo final utilizado no treinamento.
2. Fase de *preparação do treinamento*, que nos termos aqui descritos só faz sentido para numa abordagem funcional. Caso se utilize uma abordagem imperativa, pular à próxima fase. Suas etapas são:
  - (a) Preparação dos dados a serem utilizados no treinamento, incluindo a estrutura de observações  $\mathbf{X}$  e estrutura de seus alvos  $\mathbf{Y}$  caso aplicável.
  - (b) Preparação do método de validação (*i.e., holdout, k-fold*).
  - (c) Definição da função custo de erro médio  $C(\{\mathbf{W}_s\}, \mathbf{X}_b, \mathbf{Y}_b)$ , que toma os parâmetros, o lote de dados e seus alvos, e retorna um valor em  $\mathcal{R}$ . Frequentemente inclui-se aqui a definição anterior do custo calculado sobre uma única amostra,  $c_n(\{\mathbf{W}_s\}, \vec{x}, \vec{y})$ .

- (d) Opcional: pode-se definir uma função para calcular outras métricas úteis (*i.e.*, precisão, acurácia, além do custo) a partir de um lote de previsões,  $\hat{\mathbf{Y}}$ , e seus valores verdadeiros,  $\mathbf{Y}$ .
- (e) Implementação do funcional que toma um modelo, uma função custo médio e, se aplicável, parâmetros da regularização de parâmetros utilizada (*i.e.*, coeficiente do termo de esparsidade dos parâmetros na função custo), e define uma função para calcular o custo  $C$  e o gradiente do custo  $\nabla C$  deste modelo com relação aos parâmetros dados, por diferenciação automática, partindo deles e de um lote de dados composto por  $\mathbf{X}_b$  e  $\mathbf{Y}_b$ , caso aplicável. Eventualmente pode ser interessante retornar também nesta função as métricas avaliadas em treinamento por meio de uma função previamente definida, desde que isto não adicione complexidade significativa ao algoritmo de treinamento e haja tempo o suficiente para tal.
- (f) Criação de um funcional que toma o modelo e uma função custo para produzir uma função que toma parâmetros e um lote de dados, e retorna suas previsões, custo e outras métricas, caso definidas.
- (g) Criação de um funcional que toma uma função custo e retorna uma função que treina o modelo por uma época. Para tal, esta função deve receber um otimizador (*i.e.*, SGD) com suas variáveis de estado, o lote total de dados a ser utilizado no treinamento, o número de amostras a ser utilizada no treinamento estocástico, e se desejado, uma chave para realizar acumulação de gradientes ou não. A função gerada retornará em cada aplicação pelo menos os parâmetros atualizados após uma época, as novas variáveis de estado do otimizador e as métricas médias obtidas nesta mesma época.

3. Fase de *treinamento*, a qual inclui:

- (a) A geração das funções — aplicar os funcionais implementados no passo anterior:
  - i. de treinamento de uma época em função de um otimizador, dados de treinamento, número de amostras por lote e, se desejado, uma opção para acumular gradientes ou não;
  - ii. de avaliação das métricas do modelo.
- (b) Inicializar o otimizador escolhido (*i.e.*, SGD).
- (c) Definir uma condição de parada, como a descrita na Seção 1.5 ou número máximo de épocas — não recomendado, pois este parâmetro é difícil de ser escolhido.
- (d) Definir uma estrutura de dados para o guardar um histórico de métricas de treinamento.
- (e) Criar o laço de treinamento, que tipicamente envolve:
  - i. treinar uma época;
  - ii. imprimir o estado do treinamento (*i.e.*, época e valor do custo atual);
  - iii. atualizar histórico com as métricas atuais;
  - iv. criar *checkpoint* do modelo — isto é, salvar seus parâmetros em uma estrutura (tipicamente em disco);
  - v. atualizar da condição de parada a partir da métrica avaliada e dos parâmetros atuais;

- vi. verificar se a condição de parada retorna verdadeiro. Caso positivo, recuperar o *checkpoint* dos melhores parâmetros do modelo e finalizar treinamento.
- (f) Visualizar saídas do modelo.
- (g) Visualizar histórico de métricas salvo, verificar condições de viés e variância para considerar situações de subajuste ou sobreajuste.
- (h) Salvar parâmetros do modelo treinado.

Além dos passos apontados acima, vale ressaltar que os parâmetros do otimizador, como a taxa de aprendizado, ou os parâmetros da função custo, como os relacionados à regularização do modelo, devem ser testados para que possam ser definidos.

# Apêndice C

## Transformação *one-hot*

A transformação ou codificação *one-hot* consiste numa simples transformação rótulos inteiros em probabilidades de pertencimento a cada classe. Assim, se um problema considera que amostras podem pertencer a três classes, enumeradas de 1 a 3, o rótulo 1 se torna  $[1.0, 0.0, 0.0]^\top$ , o rótulo 2 se torna  $[0.0, 1.0, 0.0]^\top$ , e o rótulo 3 se torna  $[0.0, 0.0, 1.0]^\top$ , onde cada linha  $i$  destes vetores diz respeito à probabilidade de pertencimento da amostra à classe  $i$ .

Geralmente esta aplicação é realizada em massa para um vetor de  $N$  rótulos,  $\vec{r}$ , o qual é associado a uma matriz de observações das  $N$  amostras  $\mathbf{X}_{N \times M}$ . O resultado deste processo é uma matriz  $\mathbf{Y}$  cujas linhas são dadas pela transposta da transformação *one-hot* das linhas em  $\vec{r}$ . Isto pode ser verificado no caso abaixo:

$$\text{one-hot} \begin{pmatrix} 1 \\ 2 \\ 2 \\ 3 \end{pmatrix} = \begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}. \quad (\text{C.1})$$

A transformação one-hot pode ser invertida aplicando a função de argumento do máximo (argmax) em cada linha da matriz transformada. Esta função retorna o índice do elemento com maior valor do vetor aplicado:

$$\text{argmax} \begin{pmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{pmatrix} = \begin{bmatrix} 1 \\ 2 \\ 2 \\ 3 \end{bmatrix}. \quad (\text{C.2})$$

# Apêndice D

## Código para diferenciação automática direta

Este código implementa um ambiente de diferenciação automática direta na linguagem Python 3.9 de forma simples, sem o uso de bibliotecas externas. Ele não deve ser considerado em aplicações com alta demanda computacional, uma vez que foi escrito com fins didáticos.

```
1 # forward_mode.py
2 #
3 # Author: Marcos Conceicao (marcosrdac@gmail.com)
4 # Date: 2021-07-26
5
6 from dataclasses import dataclass
7 import math
8
9
10 @dataclass(frozen=True)
11 class Var:
12     """
13         Structure for naive implementation of forward mode autodiff.
14
15     Attributes:
16         primal    node's primal value
17         tangent   node's tangent value
18     """
19     primal: float
20     tangent: float = 0
21
22     def __repr__(self):
23         '''How to represent this structure.'''
24         class_name = self.__class__.__name__
25         primal, tangent = self.primal, self.tangent
26         return f"{class_name}({{primal=:4g}}, {{tangent=:4g}})"
27
28     def __add__(a, b):
29         '''Addition operation.'''
30         return Var(
31             a.primal + b.primal,
32             a.tangent + b.tangent,
33         )
34
```

```

35     def __sub__(a, b):
36         '''Subtraction operation.'''
37         return Var(
38             a.primal - b.primal,
39             a.tangent - b.tangent,
40         )
41
42     def __mul__(a, b):
43         '''Product operation.'''
44         return Var(
45             a.primal * b.primal,
46             a.tangent * b.primal + a.primal * b.tangent,
47         )
48
49     def __truediv__(a, b):
50         '''Division operation.'''
51         return Var(a.primal / b.primal,
52                    (a.tangent * b.primal - a.primal * b.tangent) / b.
53                    primal**2)
54
55     def __pow__(a, b):
56         '''Power operation.'''
57         return Var(
58             a.primal**b.primal,
59             b.primal * a.primal**(b.primal - 1) * a.tangent +
60             a.primal**b.primal * b.tangent,
61         )
62
63     def __abs__(a):
64         '''Absolute value.'''
65         return Var(
66             abs(a.primal),
67             a.tangent * math.copysign(1, a.primal),
68         )
69
70 def sin(var):
71     '''Sine operation.'''
72     return Var(math.sin(var.primal), var.tangent * math.cos(var.primal))
73
74 def cos(var):
75     '''Cosine operation.'''
76     return Var(math.cos(var.primal), -var.tangent * math.sin(var.primal))
77
78
79 def exp(var):
80     '''Exponentiation operation.'''
81     return Var(math.exp(var.primal), var.tangent * math.exp(var.primal))
82
83
84 def ln(var):
85     '''Natural logarithm operation.'''
86     return Var(math.log(var.primal), var.tangent * 1 / var.primal)
87
88
89 if __name__ == '__main__':

```

```

91 print('== Definitions')
92 # constants
93 ONE = Var(1, 0) # init deriv = d(1)/da = 0
94 # variables
95 a = Var(2, 1) # init deriv = da/dx = 1
96 b = Var(-1, 0) # init deriv = db/dx = 0
97 print('-- Constants (tangent = dC/dx = 0)')
98 print(f'{ONE = }')
99 print('-- Variables (tangent = dx/dx)')
100 print(f'{a = }')
101 print(f'{b = }')
102 print()
103
104 print(
105     '== Basic operation overloading')
106 print('(result tangents are derivatives to a)')
107 print(f'{a + b = }')
108 print(f'{a - b = }')
109 print(f'{a * b = }')
110 print(f'{a / b = }')
111 print(f'{a ** b = }')
112 print(f'{abs(a) = }')
113 print()
114
115 print('== Some transcendental operations')
116 print(f'{sin(a) = }')
117 print(f'{cos(a) = }')
118 print(f'{exp(a) = }')
119 print(f'{ln(a) = }')
120 print()
121
122 print('== Composite operations')
123
124 def f(x, y):
125     return ONE / exp(cos(sin(x) + y))
126
127 print('-- f(x,y) := 1/(exp(cos(sin(x) + y)))')
128 print(f'{f(a, b) = }')
129 print()
130
131 print('== Partial derivatives')
132
133 print('-- f(x) := [x[1] + x[0]*x[1], sin(x[1])]')
134
135 def f(x):
136     v_1 = x[0]
137     v_2 = x[1]
138     v_3 = v_1 * v_2
139     v_4 = v_2 + v_3
140     v_5 = sin(v_2)
141     f = [v_4, v_5]
142     return f
143
144 print('-- Partial derivative relative to x_0 at x=[2,2]')
145 x = [Var(2, 1), Var(2, 0)]
146 print(f'{f(x) = }')
147 print('--> Tangents:', [x_i.tangent for x_i in f(x)])

```

```
148 print(4 * ' , + 19 * '*')
149 print()
150
151 print('-- Jacobian vector product (J_f@r)')
152 print(f'{x = }')
153 r = Var(.1, 0), Var(-1, 0), Var(5, 0)
154 print(f'{r = }')
155
156 # make x's tangent = r
157 x_with_r_as_tangent = tuple(
158     Var(xi.primal, ri.primal) for xi, ri in zip(x, r))
159
160 # run f(x)
161 print(f'{f(x_with_r_as_tangent) = }')
162 print('... pairs are (f(x)[i], J_f@r)[i])')
```

# Apêndice E

## Código para diferenciação automática reversa

Este código implementa um ambiente de diferenciação automática reversa na linguagem Python 3.9 de forma simples, sem o uso de bibliotecas externas. Ele não deve ser considerado em aplicações com alta demanda computacional, uma vez que foi escrito com fins didáticos. Uma aplicação desta implementação pode ser vista no Apêndice F, onde uma rede neural tipo perceptron multicamadas é treinada.

```
1 # reverse_mode.py
2 #
3 # Author: Marcos Conceicao (marcosrdac@gmail.com)
4 # Date: 2021-07-28
5
6 from dataclasses import dataclass
7 import math
8
9
10 @dataclass(frozen=True, eq=False)
11 class Var:
12     """
13         Structure for naive implementation of reverse mode autodiff.
14
15     Attributes:
16         primal node's primal value
17         parents_and_op_adoints tuple of tuples of a node's parents and
18             associated adjoints for reverse chain rule application. The
19             recurrent
20                 tuple structure is able to keep track of the paths drawn in the
21             graph.
22     """
23     primal: float
24     parents_and_op_adoints: tuple = ()
25
26     def __add__(a, b):
27         '''Addition operation.'''
28         return Var(
29             a.primal + b.primal,
30             ((a, 1), (b, 1)),
31         )
32
33     def __mul__(a, b):
```

```

32     '''Product operation.'''  

33     return Var(  

34         a.primal * b.primal,  

35         ((a, b.primal), (b, a.primal)),  

36     )  

37  

38     def __neg__(a):  

39         '''Negation operation.'''  

40         return Var(  

41             -a.primal,  

42             ((a, -1), ),  

43         )  

44  

45     def __inv__(a):  

46         '''Inversion operation.'''  

47         return Var(  

48             1 / a.primal,  

49             ((a, -1 / a.primal**2), ),  

50         )  

51  

52     def __sub__(self, other):  

53         '''Subtraction operation.'''  

54         return self + other.__neg__()  

55  

56     def __truediv__(self, other):  

57         '''Division operation.'''  

58         return self * other.__inv__()  

59  

60     def __pow__(self, other):  

61         '''Power operation.'''  

62         primal = self.primal**other.primal  

63         return Var(primal, (  

64             (self, other.primal * self.primal**(other.primal - 1)),  

65             (other, self.primal**other.primal),  

66         ))  

67  

68     def partials(self, last_adjoint=1):  

69         '''Function to perform second step of reverse autodiff and  

70         actually  

71         calculate derivatives to base inputs.'''  

72         partials = {}  

73  

74         def get_adoints(var, adjoint):  

75             for child, derivative in var.parents_and_op_adoints:  

76                 accumulated = adjoint * derivative  

77                 partials[child] = partials.get(child, 0) + accumulated  

78  

79                 # graph propagation  

80                 get_adoints(child, accumulated)  

81  

82         get_adoints(self, last_adjoint)
83         return partials
84  

85     def sin(a):
86         '''Sine operation.'''
87         return Var(

```

```

88         math.sin(a.primal),
89         ((a, math.cos(a.primal)), ),
90     )
91
92
93 def cos(a):
94     '''Cosine operation.'''
95     return Var(
96         math.cos(a.primal),
97         ((a, -math.sin(a.primal)), ),
98     )
99
100
101 def exp(a):
102     '''Exponentiation operation.'''
103     result = math.exp(a.primal)
104     return Var(
105         result,
106         ((a, result), ),
107     )
108
109
110 def ln(a):
111     '''Natural logarithm operation.'''
112     return Var(
113         math.log(a.primal),
114         ((a, 1 / a.primal), ),
115     )
116
117
118 if __name__ == '__main__':
119
120     def f(x):
121         v_1 = x[0]
122         v_2 = x[1]
123         v_3 = v_1 * v_2
124         v_4 = v_2 + v_3
125         v_5 = sin(v_2)
126         f = [v_4, v_5]
127         return f
128
129     x = [Var(2), Var(2)]
130
131     grads = tuple(fi.partials() for fi in f(x))
132
133     print('-- Gradient of first output to each input')
134     print(f'{grads[0].get(x[0]) = }')
135     print(f'{grads[0].get(x[1]) = }')
136     print()
137     print('-- Gradient of second output to each input')
138     print(f'{grads[1].get(x[0]) = }') # None, as f_2 is not linked to x_0
139     print(f'{grads[1].get(x[1]) = }')
140
141     print('== Transpose jacobian product')
142     print('-- First element')
143
144     r = (3, .2)

```

```
145 print(f'{r = }')
146 outs = f(x)
147 edited_outs = [
148     out.partials(last_adjoint=r[i]) for i, out in enumerate(outs)
149 ]
150 print('--- First output')
151 print(f'{edited_outs[0].get(x[0]) = }')
152 print(f'{edited_outs[0].get(x[1]) = }')
153 print('--- Second output')
154 print(f'{edited_outs[1].get(x[0]) = }')
155 print(f'{edited_outs[1].get(x[1]) = }')
```

## Apêndice F

# Código para treinamento de rede neural MLP usando diferenciação automática

Este código escrito em Python 3.9 importa o código presente no Apêndice E para realizar o cálculo de gradientes por meio da diferenciação automática reversa, qual é utilizado no treinamento de uma rede perceptron multicamadas sem o uso de bibliotecas externas. O algoritmo de otimização utilizado é o SGD.

```
1 # mlp.py
2 #
3 # Author: Marcos Conceicao (marcosrdac@gmail.com)
4 # Date: 2021-07-29
5
6 import math
7 from random import random
8 from reverse_mode import Var, exp
9
10
11 def rVar():
12     '''Random variable generator.'''
13     return Var(2 * (random() - 0.5))
14
15
16 def mat_vec_mul(W, x):
17     '''Multiplies a matrix by a vector.'''
18     res = [Var(0) for j in W]
19     for i in range(len(W)):
20         for j in range(len(W[0])):
21             res[i] += W[i][j] * x[j]
22     return res
23
24
25 def vec_vec_add(u, v):
26     '''Adds two vectors.'''
27     return [ui + vi for ui, vi in zip(u, v)]
28
29
30 def sigmoid(x):
31     '''Sigmoid function.'''
32     return Var(1) / (Var(1) + exp(-x))
33
34
```

```

35 def vec_sigmoid(x):
36     '''Vectorized version of the sigmoid function.'''
37     return [sigmoid(xi) for xi in x]
38
39
40 def mlp(params, x):
41     '''Multilayer perceptron.'''
42     for layer, (W, b) in enumerate(params):
43         x = mat_vec_mul(W, x)
44         x = vec_vec_add(x, b)
45         not_last = layer < len(params) - 1
46         if not_last:
47             x = vec_sigmoid(x)
48     return x
49
50
51 def batch_mlp(params, X):
52     '''Batch version of the multilayer perceptron.'''
53     return [mlp(params, xi) for xi in X]
54
55
56 def vec_mean(x):
57     '''Takes the mean of a vector.'''
58     return sum(x, start=Var(0)) / Var(len(x))
59
60
61 def mse(u, v):
62     '''Mean squared error (MSE) function.'''
63     return vec_mean([(ui - vi)**Var(2) for ui, vi in zip(u, v)])
64
65
66 def batch_loss(params, X, Y):
67     '''Definition of the loss function from MSE.'''
68     Y_pred = batch_mlp(params, X)
69     losses = [mse(yi_pred, yi) for yi_pred, yi in zip(Y_pred, Y)]
70     return vec_mean(losses)
71
72
73 def steepest_descent(params, loss_grad, step=.02):
74     '''
75     Defines a way to perform steepest descent on parameters given loss
76     gradients and a step.
77     '''
78     new_params = []
79     for W, b in params:
80         # weights
81         W_new = W.copy()
82         for i in range(len(W)):
83             for j in range(len(W[0])):
84                 # gradient descent method
85                 Wij_new = W[i][j].primal - step * loss_grad[W[i][j]]
86                 W_new[i][j] = Var(Wij_new)
87
88         # biases
89         b_new = b.copy()
90         for i in range(len(b)):
91             # gradient descent method

```

```

92         bi_new = b[i].primal - step * loss_grad[b[i]]
93         b_new[i] = Var(bi_new)
94
95     new_params.append([W_new, b_new])
96     return new_params
97
98
99 def mat_print(X, fmt='%.2f'):
100     '''Prints a matrix.'''
101     for row in range(len(X)):
102         start = '[' if row == 0 else '['
103         print(start, end='')
104         for col in range(len(X[0])):
105             if col < len(X[0]) - 1:
106                 end = ', '
107             else:
108                 if row < len(X) - 1:
109                     end = ']'
110                 else:
111                     end = ']]'
112             print(f'{X[row][col].primal:{fmt}}', end=end)
113     print()
114
115
116 if __name__ == '__main__':
117     # observations matrix
118     X = [
119         [Var(0), Var(0)], # 1
120         [Var(1), Var(0)], # 2
121         [Var(0), Var(1)], # 3
122         [Var(0), Var(0)], # 4
123         [Var(1), Var(1)], # 5
124         [Var(0), Var(1)], # 6
125         [Var(1), Var(0)], # 7
126         [Var(1), Var(1)], # 8
127     ]
128
129     # targets matrix
130     Y = [
131         # (2x[0]+x[1], 2x[1]+x[0])
132         [Var(0), Var(0)], # 1
133         [Var(2), Var(1)], # 2
134         [Var(1), Var(2)], # 3
135         [Var(0), Var(0)], # 4
136         [Var(3), Var(3)], # 5
137         [Var(1), Var(2)], # 6
138         [Var(2), Var(1)], # 7
139         [Var(3), Var(3)], # 8
140     ]
141
142     # initial parameters for MLP architecture: 2->3->2
143     # layer 1
144     W1 = [ # 3x2
145         [rVar(), rVar()],
146         [rVar(), rVar()],
147         [rVar(), rVar()],
148     ]

```

```

149     b1 = [ # 3
150         rVar(),
151         rVar(),
152         rVar(),
153     ]
154     # layer 2
155     W2 = [ # 2x3
156         [rVar(), rVar(), rVar()],
157         [rVar(), rVar(), rVar()],
158     ]
159     b2 = [ # 2
160         rVar(),
161         rVar(),
162     ]
163
164     # a list of parameters per layer
165     params = [[W1, b1], [W2, b2]]
166
167     print('Y (true outputs):')
168     mat_print(Y)
169     print()
170
171     print('Y_pred (predictions before training):')
172     Y_pred = batch_mlp(params, X)
173     mat_print(Y_pred)
174     print()
175
176     # training
177     epochs = 500
178     print('Training with SGD... ')
179     for epoch in range(epochs):
180         loss = batch_loss(params, X, Y)
181         if epoch % 100 == 0:
182             print(f'- epoch={epoch} -> loss={loss.primal:.4g}')
183
184         loss_grad = loss.partials()
185         params = steepest_descent(params, loss_grad, step=.2)
186     print()
187     # params is now a list of trained parameters per layer
188
189     Y_pred = batch_mlp(params, X)
190     print('Y_pred (predictions after training):')
191     mat_print(Y_pred)

```

# Referências

- Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G. S.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Goodfellow, I.; Harp, A.; Irving, G.; Isard, M.; Jozefowicz, R.; Jia, Y.; Kaiser, L.; Kudlur, M.; Levenberg, J.; Mané, D.; Schuster, M.; Monga, R.; Moore, S.; Murray, D.; Olah, C.; Shlens, J.; Steiner, B.; Sutskever, I.; Talwar, K.; Tucker, P.; Vanhoucke, V.; Vasudevan, V.; Viégas, F.; Vinyals, O.; Warden, P.; Wattenberg, M.; Wicke, M.; Yu, Y. e Zheng, X. (2015) TensorFlow, Large-scale machine learning on heterogeneous systems.
- Andrade, P. N. (2017) Imageamento sísmico através do método de expansão rápida nos domínios do tempo e da frequência, Tese de Doutorado em Geofísica, Universidade Federal da Bahia, Salvador, Brasil.
- Baydin, A. G.; Pearlmutter, B. A.; Radul, A. A. e Siskind, J. M. (2018) Automatic differentiation in machine learning: a survey, *Journal of machine learning research*, **18**.
- Bernal-Romero, M. e Iturrarán-Viveros, U. (2021) Accelerating full-waveform inversion through adaptive gradient optimization methods and dynamic simultaneous sources, *Geophysical Journal International*, **225**(1):97–126.
- Biondi, B. e Sava, P. (1999) Wave-equation migration velocity analysis, In: *SEG Technical Program Expanded Abstracts 1999*, pp. 1723–1726, Society of Exploration Geophysicists.
- Bottou, L.; Curtis, F. E. e Nocedal, J. (2018) Optimization methods for large-scale machine learning, *Siam Review*, **60**(2):223–311.
- Bradbury, J.; Frostig, R.; Hawkins, P.; Johnson, M. J.; Leary, C.; Maclaurin, D.; Necula, G.; Paszke, A.; VanderPlas, J.; Wanderman-Milne, S. e Zhang, Q. (2018) JAX: composable transformations of Python+NumPy programs.
- Brougois, A.; Bourget, M.; Lailly, P.; Poulet, M.; Ricarte, P. e Versteeg, R. (1990) Marmousi, model and data, In: *EAEG workshop-practical aspects of seismic data inversion*, pp. cp–108, European Association of Geoscientists & Engineers.
- Bunks, C.; Saleck, F. M.; Zaleski, S. e Chavent, G. (1995) Multiscale seismic waveform inversion, *Geophysics*, **60**(5):1457–1473.
- Carcione, J. M. (2010) A generalization of the fourier pseudospectral method, *Geophysics*, **75**(6):A53–A56.
- Carcione, J. M.; Herman, G. C. e Ten Kroode, A. (2002) Seismic modeling, *Geophysics*, **67**(4):1304–1325.
- Castelvecchi, D. (2016) Can we open the black box of AI?, *Nature News*, **538**(7623):20.
- Cerjan, C.; Kosloff, D.; Kosloff, R. e Reshef, M. (1985) A nonreflecting boundary condition for discrete acoustic and elastic wave equations, *Geophysics*, **50**(4):705–708.

- Chen, R. T.; Rubanova, Y.; Bettencourt, J. e Duvenaud, D. (2018) Neural ordinary differential equations, arXiv preprint arXiv:1806.07366.
- Ciresan, D. C.; Meier, U.; Masci, J.; Gambardella, L. M. e Schmidhuber, J. (2011) Flexible, high performance convolutional neural networks for image classification, In: *Twenty-second international joint conference on artificial intelligence*.
- Cramer, J. S. (2002) The origins of logistic regression.
- Curry, H. B. (1944) The method of steepest descent for non-linear minimization problems, *Quarterly of Applied Mathematics*, **2**(3):258–261.
- Deisenroth, M. P.; Faisal, A. A. e Ong, C. S. (2020) Mathematics for machine learning, Cambridge University Press.
- Devlin, J.; Chang, M.-W.; Lee, K. e Toutanova, K. (2018) Bert: Pre-training of deep bidirectional transformers for language understanding, arXiv preprint arXiv:1810.04805.
- Duchi, J.; Hazan, E. e Singer, Y. (2011) Adaptive subgradient methods for online learning and stochastic optimization., *Journal of machine learning research*, **12**(7).
- Elman, J. L. (1990) Finding structure in time, *Cognitive science*, **14**(2):179–211.
- Fichtner, A.; Bunge, H.-P. e Igel, H. (2006) The adjoint method in seismology: I. theory, *Physics of the Earth and Planetary Interiors*, **157**(1-2):86–104.
- Hu, Y.; Jin, Y.; Wu, X. e Chen, J. (2020) Solving time domain electromagnetic problems using a differentiable programming platform, In: *2020 IEEE USNC-CNC-URSI North American Radio Science Meeting (Joint with AP-S Symposium)*, pp. 181–182, IEEE.
- James, G.; Witten, D.; Hastie, T. e Tibshirani, R. (2013) An introduction to statistical learning, vol. 112, Springer.
- Johnson, S. G. (2012) Notes on adjoint methods for 18.335, *Introduction to Numerical Methods*.
- Jordan, M. (1986) Serial order: a parallel distributed processing approach. technical report, june 1985-march 1986, Rel. Téc., California Univ., San Diego, La Jolla (USA). Inst. for Cognitive Science.
- Karimpouli, S. e Tahmasebi, P. (2020) Physics informed machine learning: Seismic wave equation, *Geoscience Frontiers*, **11**(6):1993–2001.
- Kim, K. e Timm, N. (2006) Univariate and multivariate general linear models: theory and applications with SAS, CRC Press.
- Kingma, D. P. e Ba, J. (2014) Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980.
- Koehne, V. (2018) FWI multiescala: uma implementação em GPU, Dissertação de Mestrado em Geofísica, Universidade Federal da Bahia, Salvador, Brasil.
- Kosloff, D. D. e Baysal, E. (1982) Forward modeling by a fourier method, *Geophysics*, **47**(10):1402–1412.
- LeCun, Y.; Boser, B.; Denker, J. S.; Henderson, D.; Howard, R. E.; Hubbard, W. e Jackel, L. D. (1989) Backpropagation applied to handwritten zip code recognition, *Neural computation*, **1**(4):541–551.

- Lines, L. R.; Slawinski, R. e Bording, R. P. (1999) A recipe for stability of finite-difference wave-equation computations, *Geophysics*, **64**(3):967–969.
- McCulloch, W. S. e Pitts, W. (1943) A logical calculus of the ideas immanent in nervous activity, *The bulletin of mathematical biophysics*, **5**(4):115–133.
- Menard, S. (2001) *Applied Logistic Regression Analysis*, SAGE Publications.
- Moseley, B.; Markham, A. e Nissen-Meyer, T. (2020) Solving the wave equation with physics-informed deep learning, arXiv preprint arXiv:2006.11894.
- Murtagh, F. (2003) Multilayer perceptrons and regression for classification.
- Nguyen, L. H. e Malinsky, A. (2020) Exploration and implementation of neural ordinary differential equations.
- Pavelle, R.; Rothstein, M. e Fitch, J. (1981) Computer algebra, *Scientific American*, **245**(6):136–153.
- Pestana, R. C. e Stoffa, P. L. (2009) Rapid expansion method (REM) for time-stepping in reverse time migration (RTM), In: *SEG Technical Program Expanded Abstracts 2009*, pp. 2819–2823, Society of Exploration Geophysicists.
- Raissi, M.; Perdikaris, P. e Karniadakis, G. E. (2019) Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, *Journal of Computational Physics*, **378**:686–707.
- Rashevsky, N. (1938) Contribution to the mathematical biophysics of visual perception with special reference to the theory of aesthetic values of geometrical patterns, *Psychometrika*, **3**(4):253–271.
- Ren, Y.; Xu, X.; Yang, S.; Nie, L. e Chen, Y. (2020) A physics-based neural-network way to perform seismic full waveform inversion, *IEEE Access*, **8**:112266–112277.
- Richardson, A. (2018) Seismic full-waveform inversion using deep learning tools and techniques, arXiv preprint arXiv:1801.07232.
- Robbins, H. e Monro, S. (1951) A stochastic approximation method, *The annals of mathematical statistics*, pp. 400–407.
- Rosenblatt, F. (1958) The perceptron: a probabilistic model for information storage and organization in the brain., *Psychological review*, **65**(6):386.
- Rumelhart, D. E.; Hinton, G. E. e Williams, R. J. (1986) Learning representations by back-propagating errors, *nature*, **323**(6088):533–536.
- Rummelhart, D. E.; McClelland, J. L.; Group, P. R. et al. (1986) Parallel distributed processing.
- dos Santos, A. W. G. (2013) Inversão de forma de onda aplicada à análise de velocidades sísmicas utilizando uma abordagem multiescala, Dissertação de Mestrado em Geofísica, Universidade Federal da Bahia, Salvador, Brasil.
- Sun, J.; Niu, Z.; Innanen, K.; Li, J. e Trad, D. (2019) A deep learning perspective of the forward and inverse problems in exploration geophysics, CSEG GeoConvention.
- Tarantola, A. (1984) Linearized inversion of seismic reflection data, *Geophysical prospecting*, **32**(6):998–1015.

- Tieleman, T. e Hinton, G. (2012) Lecture 6.5-rmsprop, coursera: Neural networks for machine learning, University of Toronto, Technical Report.
- Virieux, J. e Operto, S. (2009) An overview of full-waveform inversion in exploration geophysics, *Geophysics*, **74**(6):WCC1–WCC26.
- Virieux, J.; Calandra, H. e Plessix, R.-É. (2011) A review of the spectral, pseudo-spectral, finite-difference and finite-element modelling techniques for geophysical imaging, *Geophysical Prospecting*, **59**(Modelling Methods for Geophysical Imaging: Trends and Perspectives):794–813.
- Wang, W.; McMechan, G. A. e Ma, J. (2021) Elastic isotropic and anisotropic full-waveform inversions using automatic differentiation for gradient calculations in a framework of recurrent neural networks, *Geophysics*, **86**(6):R795–R810.
- Wardi, Y. (1988) A stochastic steepest-descent algorithm, *Journal of optimization theory and applications*, **59**(2):307–323.
- Wu, W.-J.; Lines, L. R. e Lu, H.-X. (1996) Analysis of higher-order, finite-difference schemes in 3-d reverse-time migration, *Geophysics*, **61**(3):845–856.