Relatório Parcial de Iniciação Científica

Métodos de Otimização aplicados a redes neurais para detecção de anomalias em transações com cartão de crédito

Bolsista: Frederico José Ribeiro Pelogia

Orientador: Prof. Dr. Luís Felipe Bueno

Número do Projeto FAPESP: 2019/13420-4

Período de vigência: 01/09/2019 a 31/08/2020

Período abordado pelo relatório: 01/09/2019 a 10/02/2020

Departamento de Ciência e Tecnologia
Universidade Federal de São Paulo
Janeiro de 2020

Resumo do projeto

Neste projeto pretende-se estudar métodos de otimização, em especial algoritmos estocásticos, aplicados ao treinamento de redes neurais. Em um trabalho recente proposto por L. F. Bueno e J. M. Martínez, On the complexity of solving feasibility problems, 2018, foi apresentado um algoritmo de primeira ordem com bons resultados de complexidade para problemas de quadrados mínimos. Um dos principais pontos da pesquisa deste projeto será desenvolver uma versão estocástica desse algoritmo. Serão analisados os desempenhos dos algoritmos estudados quando aplicados à detecção de fraudes em operações de cartão de crédito utilizando a base de dados Credit Card Fraud Detection do Kaggle.

Resumo das realizações do período

Este relatório aborda as realizações do período de 01/09/2019 a 10/02/2020. Neste intervalo de tempo, as seguintes atividades foram concluídas:

- Revisão bibliográfica sobre detecção de anomalias, com base no artigo [6].
- Estudo de redes neurais profundas e implementação de alguns códigos em *Python* para familiarização com o assunto.
- Estudo e implementação, em *Python*, de alguns métodos de otimização amplamente utilizados na área de *Deep Learning*. São eles:
  - Método do Gradiente (*Gradient Descent*).
  - Método do Gradiente Estocástico (*SGD*).
  - Método do Gradiente Estocástico com Redução de Variância (*SVRG*).
  - Averaged Stochastic Gradient Descent (*SAG* e *SAGA*).
  - Método do Gradiente com Passo Adaptativo (*Adagrad*).
  - Método *RMSProp*.
  - Métodos com momento de Polyak: foram feitas implementações com o *SGD* e *SVRG*.
  - Métodos com momento de Nesterov: foi feita a implementação com o *SGD*.
  - Método *Adam*.
- Estudo de métodos estocásticos combinados aplicados a redes neurais, com base no artigo [8].
- Familiarização com a base de dados *Credit Card Fraud Detection* da plataforma *Kaggle*.

1 Revisão bibliográfica sobre detecção de anomalias

O primeiro passo no desenvolvimento do projeto foi a realização de um breve estudo sobre detecção de anomalias com base no artigo [6].

A detecção de anomalias consiste na análise de dados em busca de padrões que não estejam de acordo com o comportamento esperado. A presença de dados anômalos pode ter diferentes significados em diferentes domínios de dados. Um padrão anômalo em uma imagem por ressonância magnética, por exemplo, pode indicar a presença de um tumor maligno [19]. Da mesma forma, um comportamento anômalo em dados de sensores de uma nave espacial pode indicar falhas em componentes cruciais da aeronave [10]. Tratando-se de transações com cartão de crédito, anomalias em dados dessa natureza podem sugerir a ocorrência de fraudes [1].

Tendo em vista a importância da detecção de anomalias, que pode lidar com questões sensíveis de diferentes áreas, é importante conhecer a natureza da anomalia com que se trabalha, para ter certeza do que está sendo detectado. De acordo com o artigo estudado, podem haver três principais tipos de anomalia: as pontuais, as contextuais e as coletivas. Uma anomalia pontual é qualquer porção de dados que não tenha um comportamento considerado normal em relação aos demais. Anomalias contextuais são dados que possuem um comportamento anômalo apenas em um contexto específico, sendo necessário analisar a vizinhança que os contém. As anomalias coletivas só podem ser encontradas em conjuntos de dados que possuem instâncias **relacionadas** [6]. Esse tipo de anomalia ocorre quando uma seleção dessas é anômala em relação a todo o conjunto de dados.

2 Estudo de redes neurais profundas

O aprendizado profundo, também conhecido por *Deep Learning*, é um tipo específico de aprendizado de máquina. Isso significa que seu estudo é facilitado quando se constrói previamente uma base teórica nessa área.

Basicamente, o aprendizado de máquina (*machine learning*) aborda algoritmos que conseguem medir seu desempenho em realizar certa atividade e, assim, reajustar seus parâmetros de forma a minimizar numericamente uma função que representa seu erro. Uma característica imprescindível para bons modelos de *machine learning* é a capacidade de generalização, isto é, conseguir aprender com um conjunto de dados de modo a fazer predições aceitáveis para qualquer novo dado, evitando o que é conhecido por *overfitting* ao conjunto de treinamento.

2.1 Redes Neurais

As redes neurais artificiais são estudadas desde a década de 1940, com Warren McCulloch e Walter Pitts. Porém, só nas ultimas décadas houve um expressivo aumento em seu estudo e aplicação a problemas reais, principalmente pelo alto poder de processamento dos computadores atuais.

A teoria das redes neurais artificiais, segundo [5], é inspirada na estrutura neural de organismos inteligentes, que adquirem conhecimento através da experiência. Desse modo, assim como os neurônios biológicos se comunicam através de sinapses e disparam ao atingirem um limiar de ação, os neurônios artificias possuem um comportamento semelhante.

2.1.1 Modelo de um neurônio artificial

O funcionamento de um neurônio artificial foi proposto por McCullock e Pitts em 1943 e hoje é amplamente estudado, pois é a unidade básica de uma rede neural. Como pode-se encontrar em [5], um neurônio artificial funciona da seguinte forma:

- O neurônio recebe como entrada diversos sinais X_i, que são multiplicados por valores W_i denominados pesos, que simbolizam a influência de cada sinal na saída.
- É feita, então, uma soma ponderada desses sinais, resultando em um valor que é passado a uma **função de ativação**. Essa função verifica se o valor excedeu um certo limite e, dessa forma, decide se passará um sinal à saída Y ou não, decisão essa similar ao disparo dos neurônios biológicos.

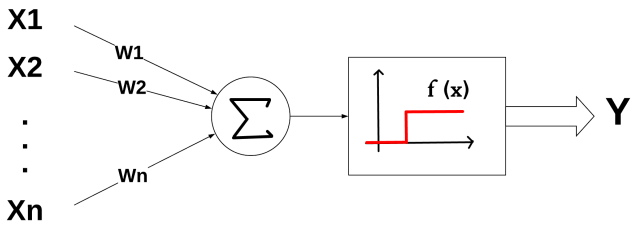

Figura 1: Esquema simplificado de um neurônio de McCullock e Pitts. Fonte: Autor

Assim, um neurônio artificial pode ser traduzido na seguinte função:

$$Y = f(X^T W).$$

2.2 Estrutura de uma rede neural

Uma rede neural é composta por camadas com diversos neurônios. A primeira camada é denominada **camada de entrada**, seguida então por um número arbitrário de camadas intermediárias, chamadas **camadas ocultas** ou **escondidas** e a última camada é chamada **camada de saída**. Em redes totalmente conexas, cada neurônio de uma certa camada se comunica com todos os da próxima, possuindo um certo peso. A representação da rede como um grafo facilita a vizualização das camadas, isso pode ser observado na Figura 2.

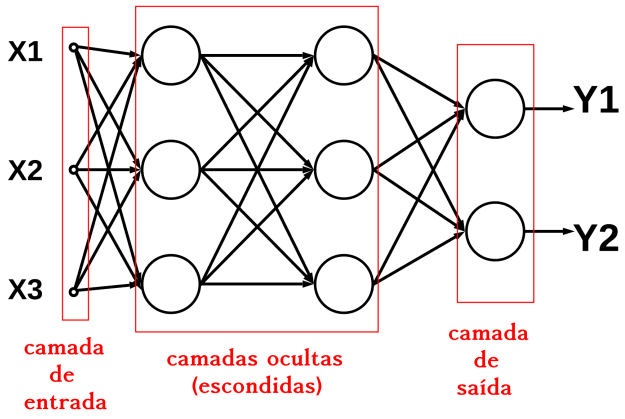

Figura 2: Estrutura de uma rede neural. Fonte: Autor

2.3 Alimentação da rede e função de perda

Para entender o funcionamento de uma rede neural, é necessário entender como os dados se propagam por sua estrutura até gerar uma resposta. Esse mecanismo de alimentação da rede é conhecido como *Feedforward*.

Seja uma rede neural com 2 entradas, 1 camada oculta com 2 neurônios e 1 saída, como na Figura 3.

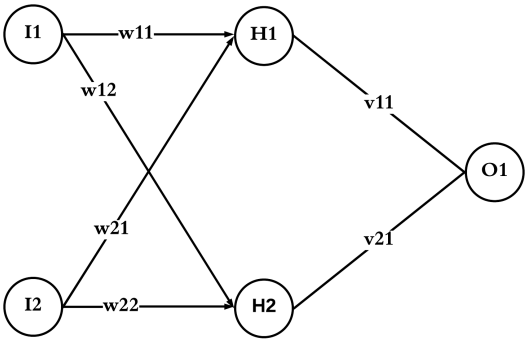

Figura 3: Rede neural com 1 camada oculta, 2 entradas e uma saída. Fonte: Autor

A matriz I será definida como a camada de entrada, H como a camada oculta, O como a camada de saída, W como a matriz de pesos entre I e H, e V como a matriz de pesos entre H e O. Seja f_h a função de ativação da camada H e f_o a função de ativação da camada O.

Definindo

$$I = \begin{pmatrix} I_1 \\ I_2 \end{pmatrix}, H = \begin{pmatrix} H_1 \\ H_2 \end{pmatrix}, O = \begin{pmatrix} O_1 \end{pmatrix},$$

$$W = \begin{pmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{pmatrix}, V = \begin{pmatrix} v_{11} & v_{21} \end{pmatrix},$$

Pode-se analisar o mecanismo de alimentação da rede. Para gerar uma resposta a partir de um dado de treinamento, as seguintes operações são feitas:

Sendo H_{in} o conteúdo da camada escondida antes de passar pela função de ativação,

$$H_{in} = W \cdot I = \begin{pmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{pmatrix} \cdot \begin{pmatrix} I_1 \\ I_2 \end{pmatrix} = \begin{pmatrix} w_{11}I_1 + w_{21}I_2 \\ w_{12}I_1 + w_{22}I_2 \end{pmatrix},$$

o que implica que

$$H = f_h(H_{in}) = \begin{pmatrix} f_h(w_{11}I_1 + w_{21}I_2) \\ f_h(w_{12}I_1 + w_{22}I_2) \end{pmatrix}.$$

Da mesma forma, sendo O_{in} o conteúdo da camada de saída antes de passar pela função de ativação,

$$O_{in} = V \cdot H = \begin{pmatrix} v_{11} & v_{21} \end{pmatrix} \cdot \begin{pmatrix} H_1 \\ H_2 \end{pmatrix} = v_{11}f_h(w_{11}I_1 + w_{21}I_2) + v_{21}f_h(w_{12}I_1 + w_{22}I_2).$$

Assim, a saída O (*output*) da rede pode ser escrita como

$$O = f_o(O_{in}) = f_o(v_{11}f_h(w_{11}I_1 + w_{21}I_2) + v_{21}f_h(w_{12}I_1 + w_{22}I_2)).$$

Agora, tendo o *output* da rede para o exemplo de treinamento e a resposta correta (*label*) y, é possível montar uma função EQ que represente o erro quadrático (perda) da predição, da seguinte maneira

$$EQ\begin{pmatrix} w_{11} \\ \vdots \\ v_{21} \end{pmatrix} = \frac{1}{2}\left(O\begin{pmatrix} I_1 \\ \vdots \\ v_{21} \end{pmatrix} - y \right)^2.$$

Assim, somando o erro quadrático obtido para cada exemplo de treinamento, obtém-se a função que deve ser minimizada para que a rede aprenda com os dados.

3 Backpropagation e o cálculo do gradiente

Backpropagation é um algoritmo que calcula de forma eficiente o gradiente da função do erro quadrático de uma rede neural em relação a cada um dos pesos. Isso é feito através de uma regra da cadeia, iterando de trás para frente pelas camadas da rede. Esse algoritmo é essencial para treinar redes neurais, pois o cálculo do gradiente é feito para que algum método de otimização possa ser aplicado para atualizar os parâmetros do modelo.

Quando uma rede neural é treinada por aprendizado supervisionado, é preciso, para cada exemplo de treinamento, descobrir o quão sensível é a função EQ em relação a cada peso da rede. Com isso, é possível saber qual ajuste nestes pesos vai ocasionar o maior decrescimento dessa função.

A sensibilidade da função do erro quadrático (EQ) em relação a um peso w específico é justamente sua taxa de variação em relação ao mesmo. A derivada parcial $\frac{\partial EQ}{\partial w}$ é calculada por meio de uma regra da cadeia.

Seja

$$E\begin{pmatrix} I_1 \\ \vdots \\ v_{21} \end{pmatrix} = (O\begin{pmatrix} I_1 \\ \vdots \\ v_{21} \end{pmatrix} - y).$$

As derivadas parciais de EQ em relação a cada peso da rede podem ser calculadas da seguinte forma

$$\frac{\partial EQ}{\partial w_{11}}\begin{pmatrix} w_{11} \\ \vdots \\ v_{21} \end{pmatrix} = E\begin{pmatrix} I_1 \\ \vdots \\ v_{21} \end{pmatrix} \cdot f_o'(O_{in}) \cdot v_{11} \cdot f_h'(H_1) \cdot I_1,$$

$$\frac{\partial EQ}{\partial w_{12}}\begin{pmatrix} w_{11} \\ \vdots \\ v_{21} \end{pmatrix} = E\begin{pmatrix} I_1 \\ \vdots \\ v_{21} \end{pmatrix} \cdot f_o'(O_{in}) \cdot v_{21} \cdot f_h'(H_2) \cdot I_1,$$

$$\frac{\partial EQ}{\partial w_{21}}\begin{pmatrix} w_{11} \\ \vdots \\ v_{21} \end{pmatrix} = E\begin{pmatrix} I_1 \\ \vdots \\ v_{21} \end{pmatrix} \cdot f_o'(O_{in}) \cdot v_{11} \cdot f_h'(H_1) \cdot I_2,$$

$$\frac{\partial EQ}{\partial w_{22}}\begin{pmatrix} w_{11} \\ \vdots \\ v_{21} \end{pmatrix} = E\begin{pmatrix} I_1 \\ \vdots \\ v_{21} \end{pmatrix} \cdot f_o'(O_{in}) \cdot v_{21} \cdot f_h'(H_2) \cdot I_2,$$

$$\frac{\partial EQ}{\partial v_{11}}\begin{pmatrix} w_{11} \\ \vdots \\ v_{21} \end{pmatrix} = E\begin{pmatrix} I_1 \\ \vdots \\ v_{21} \end{pmatrix} \cdot f_o'(O_{in}) \cdot f_h(H_{in1}),$$

$$\frac{\partial EQ}{\partial v_{21}}\begin{pmatrix} w_{11} \\ \vdots \\ v_{21} \end{pmatrix} = E\begin{pmatrix} I_1 \\ \vdots \\ v_{21} \end{pmatrix} \cdot f_o'(O_{in}) \cdot f_h(H_{in2}).$$

Agora, montando as matrizes para atualização dos pesos, será possível também montar o gradiente da função EQ.

A matriz de atualização dos pesos de W tem a seguinte forma

$$\begin{pmatrix} \frac{\partial EQ}{\partial v_{11}}\begin{pmatrix} w_{11} \\ \vdots \\ v_{21} \end{pmatrix} \\ \frac{\partial EQ}{\partial v_{21}}\begin{pmatrix} w_{11} \\ \vdots \\ v_{21} \end{pmatrix} \end{pmatrix} = E\begin{pmatrix} I_1 \\ \vdots \\ v_{21} \end{pmatrix} \cdot f_o'(O_{in}) \cdot H.$$

A matriz de atualização dos pesos de V tem a seguinte forma

$$\begin{pmatrix} \frac{\partial EQ}{\partial w_{11}} \begin{pmatrix} w_{11} \\ \vdots \\ v_{21} \end{pmatrix} \\ \frac{\partial EQ}{\partial w_{12}} \begin{pmatrix} w_{11} \\ \vdots \\ v_{21} \end{pmatrix} \\ \frac{\partial EQ}{\partial w_{21}} \begin{pmatrix} w_{11} \\ \vdots \\ v_{21} \end{pmatrix} \\ \frac{\partial EQ}{\partial w_{22}} \begin{pmatrix} w_{11} \\ \vdots \\ v_{21} \end{pmatrix} = \end{pmatrix} = E \begin{pmatrix} I_1 \\ \vdots \\ v_{21} \end{pmatrix} \cdot f'_o(O_{in}) \cdot \begin{pmatrix} v_{11} f'_h(H_1) & 0 \\ v_{21} f'_h(H_2) & 0 \\ 0 & v_{11} f'_h(H_1) \\ 0 & v_{21} f'_h(H_2) \end{pmatrix} \cdot \begin{pmatrix} I_1 \\ I_2 \end{pmatrix}.$$

Logo, o gradiente da função EQ é disposto a seguir

$$\nabla EQ = \begin{pmatrix} E \begin{pmatrix} I_1 \\ \vdots \\ v_{21} \end{pmatrix} \cdot f'_o(O_{in}) \cdot \begin{pmatrix} v_{11} f'_h(H_1) & 0 \\ v_{21} f'_h(H_2) & 0 \\ 0 & v_{11} f'_h(H_1) \\ 0 & v_{21} f'_h(H_2) \end{pmatrix} \cdot \begin{pmatrix} I_1 \\ I_2 \end{pmatrix} \\ E \begin{pmatrix} I_1 \\ \vdots \\ v_{21} \end{pmatrix} \cdot f'_o(O_{in}) \cdot H \end{pmatrix}.$$

Como a função que representa o erro da rede é uma **soma** de funções de erro quadrático, o gradiente que será utilizado para dar o passo do método será a soma dos gradientes dessas funções EQ.

4 Estudo e implementação de alguns métodos de otimização

Sabendo calcular o gradiente da função EQ, pode-se utilizar métodos de otimização baseados em gradiente para efetivamente treinar a rede neural e fazer com que ela aprenda com os dados de treinamento. Todas as 12 implementações feitas estão disponiveis no link [16].

Exemplo de teste

O problema que foi utilizado de exemplo para testar os algoritmos implementados nessa seção é uma regressão simples da função $y = x^2$. O conjunto de treinamento é composto pelos números de 1 a 5 e o resultado esperado é composto pelo quadrado de cada um desses números. O conjunto de treinamento X

e o conjunto com as respostas esperadas (*labels*) y, podem ser observados a seguir

$$X = \{1, 2, 3, 4, 5\},$$

$$y = \{1, 4, 9, 16, 25\}.$$

Todos os testes realizados utilizam uma rede neural com 1 entrada, 2 nós na camada escondida e 1 saída, representada na Figura 4. A função de ativação das camadas oculta e de saída é $f(x) = max(x, 0)$.

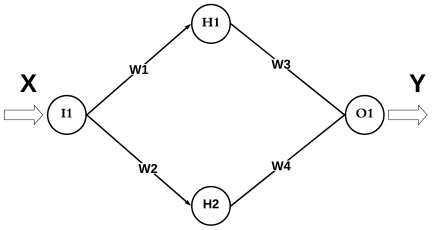

Figura 4: Rede neural utilizada para testes. Fonte: Autor

Note que, como $X > 0$,

$$Y = (W_1W_3 + W_2W_4)X,$$

o que implica que

$$Y = mX, \text{ para } m = W_1W_3 + W_2W_4.$$

Isso revela que, com essa arquitetura de rede de apenas 1 camada escondida, está sendo feita uma regressão **linear**. Assim, um valor ideal de m pode ser calculado para fins de comparação com os resultados dos algoritmos. O primeiro passo para encontrar m é montar o sistema

$$\begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix} m = \begin{pmatrix} 1 \\ 4 \\ 9 \\ 16 \\ 25 \end{pmatrix}.$$

Pelo método dos quadrados mínimos linear, a melhor aproximação para um sistema $Ax = b$ ocorre quando $A^TAx = A^Tb$, logo

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix} m = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \end{pmatrix} \begin{pmatrix} 1 \\ 4 \\ 9 \\ 16 \\ 25 \end{pmatrix}.$$

Isso significa que

$$(1 + 4 + 9 + 16 + 25)m = (1 + 8 + 27 + 64 + 125),$$

o que implica que

$$55m = 225.$$

Logo,

$$m \approx 4.09091.$$

4.1 Método do Gradiente

O primeiro método estudado foi o Método do Gradiente. Este é um dos métodos mais clássicos no estudo de otimização. O método é iterativo e consiste em avançar, a partir de um ponto inicial, na direção oposta à do gradiente da função neste ponto, isto é, na direção de maior decrescimento. O tamanho do passo (t), também conhecido como *learning rate*, geralmente é definido de modo que o ponto x^{k+1} esteja mais próximo da solução (do mínimo da função) do que x^k.

Basicamente:

$$x^{k+1} = x^k - t^k \nabla f(x^k).$$

Assim, para treinar uma rede neural utilizando o Método do Gradiente, seguimos os seguintes passos:

- Inicializar a matriz de pesos de forma aleatória;
- Calcular o gradiente da função erro ao fazer predições com os dados de treinamento, passando cada um deles, juntamente com a resposta esperada, para a função *backpropagation* e somando os gradientes individuais;
- Atualizar a matriz de pesos de forma que cada peso w receba o valor de $w - t\frac{\partial EQ}{\partial w}$, sendo EQ a função de erro quadrático e t o passo;

Um algoritmo referente ao Método do Gradiente foi implementado e submetido a testes com diferentes tamanho de passo (t). As Figuras a seguir mostram os resultados do teste com $t = 0.001$.

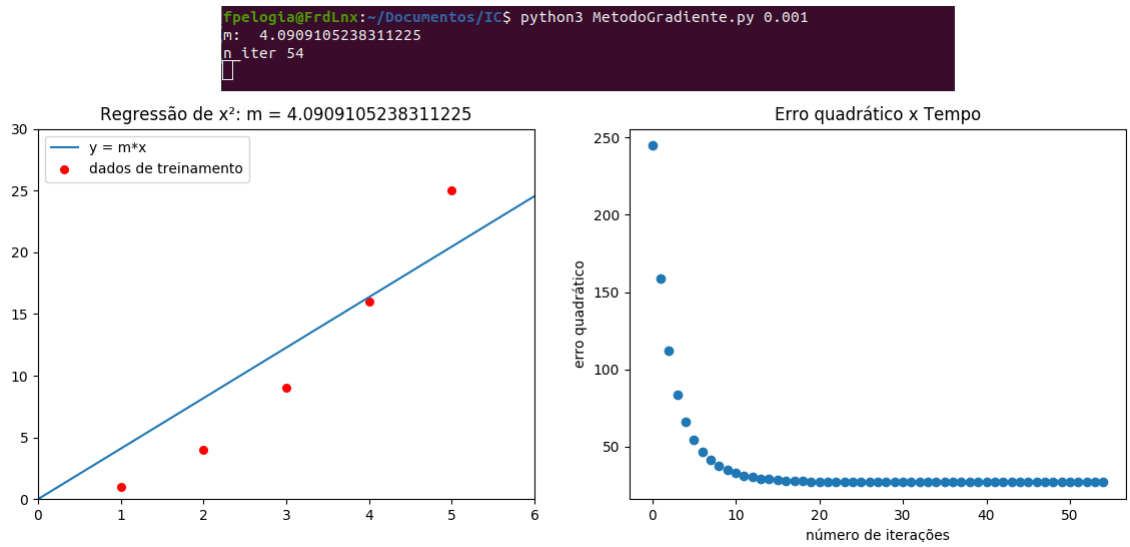

Figura 5: Resultados do teste com o Método do Gradiente para $t = 0.001$. Fonte: Autor

Como esperado, o algoritmo convergiu para a resposta correta com facilidade, levando poucas iterações para isso, mesmo com o passo relativamente pequeno. Como o método calcula o gradiente considerando todos os exemplos de treinamento, ele minimiza com exatidão a função de erro quadrático, dependendo apenas do tamanho do passo para definir a velocidade de convergência.

4.2 Método do Gradiente Estocástico (SGD)

O Método do Gradiente Estocástico é uma extensão do Método do Gradiente que o torna mais viável para trabalhar com quantidades massivas de dados, sendo um dos métodos mais utilizados na área de *Deep Learning* atualmente, segundo [11].

A ideia desse método se encaixa bem a um problema comum nos algoritmos de aprendizado de máquina, que são beneficiados por grandes conjuntos de treinamentos mas a complexidade computacional escala de forma inviável com o tamanho desses conjuntos. Por exemplo, utilizando o Método do Gradiente para minimizar uma função EQ que é soma de n termos, referente a um conjunto de treinamento com n dados, para calcular o gradiente completo, é preciso tirar a média de n gradientes, o que tem custo $O(n)$. Esse custo para dar um passo do método não escala nada bem quando se há bilhões de exemplos de treinamento, por exemplo.

O Método do Gradiente Estocástico resolve esse problema ao considerar o gradiente como uma esperança matemática, que pode ser estimada ao dar passos calculando o gradiente para apenas um exemplo de treinamento ou para pequenas amostras (lotes) selecionadas aleatoriamente do conjunto de treinamento, o que é conhecido como *mini-batch SGD*. Esse cálculo de poucos gradientes por passo reduz drásticamente o custo computacional enquanto mantém uma boa aproximação das direções de descida da função erro.

O algoritmo referente a esse método foi implementado e testado com diversos tamanhos de passo e de número de amostras (*batch size*). As Figuras 6 e 7 mostram, respectivamente, os resultados para os testes

com passo $t = 0.0001$ e lotes de tamanho 4 e 2. A condição de parada foi definida para 10000 iterações ou quando a norma do gradiente se tornasse menor do que uma tolerância $tol = 0.0001$.

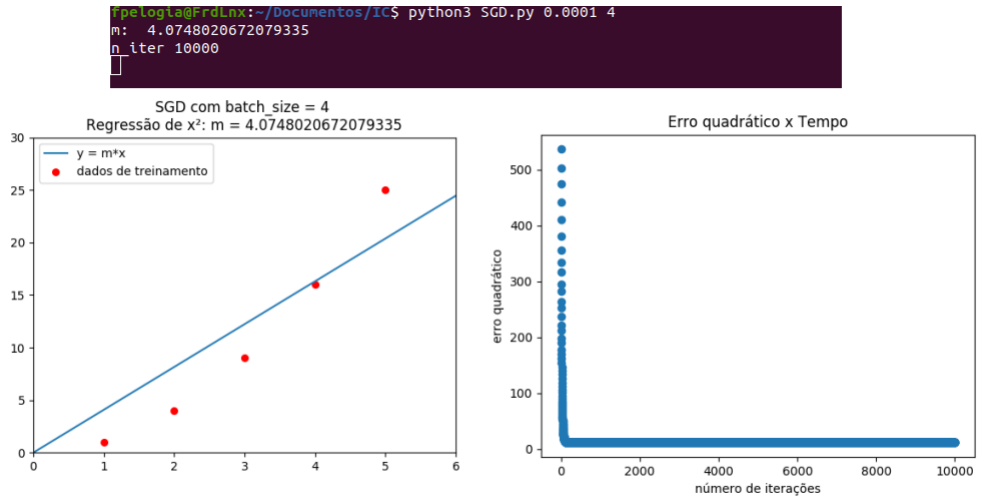

Figura 6: Resultados do teste do *SGD* com passo $t = 0.0001$ e lote de tamanho 4. Fonte: Autor

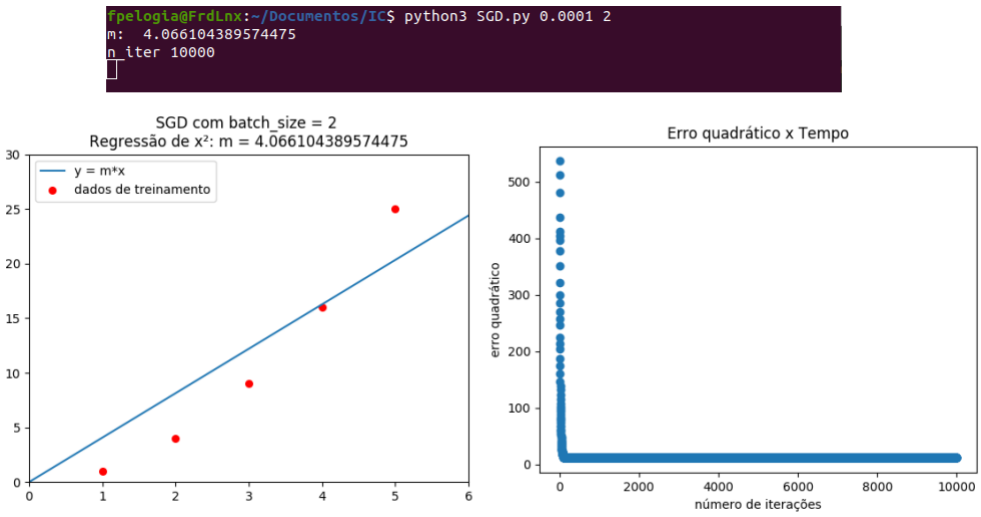

Figura 7: Resultados do teste do *SGD* com passo $t = 0.0001$ e lote de tamanho 2. Fonte: Autor

Os resultados foram coerentes com o que se esperava da ideia do algoritmo. Quanto menor o tamanho do lote, maior a dificuldade de convergência (revelada pelos valores de m), pois cada vez mais o gradiente a ser estimado é aproximado com menos informação e a variância aumenta.

A fim de demonstrar a superioridade do *SGD* sobre o Método do Gradiente para bases de dados grandes, foi realizado um teste (Figura 8) de tempo de execução com 10000 exemplos de treinamento, contendo valores de 1 a 5 (na mesma quantidade) embaralhados. Foi definido, exclusivamente para esse teste, que a condição de parada seria atingida quando $4.03 < m < 4.15$. A comparação em questão foi do Método do Gradiente com o SGD de tamanho de lote 10. O tamanho do passo utilizado foi de $t = 0.001$.

```
fpelogia@FrdLnx:~/Documentos/IC/TESTE$ python3 GD_versus_SGD.py 0.001 10

 Treinando com 10000 dados

===== Método do gradiente =====
m:  4.1379731306170955
n_iter 16
Tempo de execução: 20.33174729347229s
====================================


===== SGD com batch_size 10 =====
m:  4.13193844402326
n_iter 20
Tempo de execução: 0.03824949264526367s
====================================
```

Figura 8: Resultados do teste de comparação do Método do Gradiente com o *SGD* de tamanho de lote 10 para 10000 dados de treinamento. Fonte: Autor

A Figura 8 deixa claro que o *SGD* é muito mais viável para grandes bases de dados.

Algo interessante de se apontar é que, como o gradiente é estimado de forma inexata, o método pode, em alguns pontos, seguir uma direção em que a função cresce. Assim, podem ocorrer pequenos aumentos do erro quadrático durante o treinamento, fenômeno esse conhecido por **bola de ruído**.

A Figura 9 apresenta os resultados do teste com tamanho do lote 3 e passo $t = 0.01$, em que pode ser observado o fenômeno da bola de ruído.

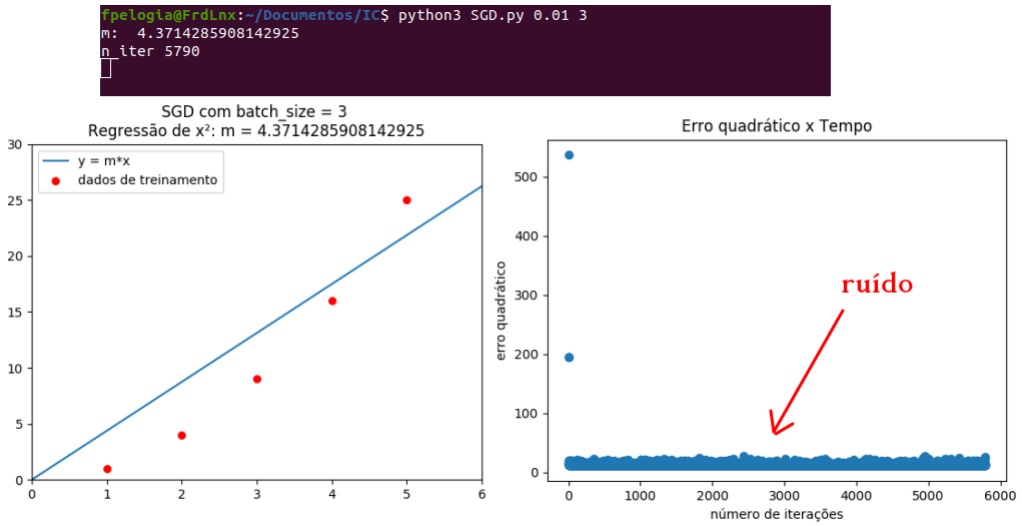

Figura 9: Resultados do teste do *SGD* com tamanho do lote 3 e passo $t = 0.001$. Fonte:Autor

4.2.1 Teorema de convergência e bola de ruído

A abordagem do Método do Gradiente Estocástico, para minimizar uma função $F(x)$ da forma

$$F(x) = \frac{1}{N}\sum_{i=1}^{N} f_i(x),$$

pode ser resumida como:

$$x^{k+1} = x^k - t\nabla f_{i_k}(x^k).$$

A esperança matemática fica:

$$E[x^{k+1} \mid x^k] = x^k - t\frac{1}{N}\sum_{i=1}^{N} \nabla f_i(x^k).$$

Serão consideradas funções duas vezes diferenciáveis, com gradiente Lipschitz e fortemente convexas, o que indica a presença de um único minimizador x^* e que existem $0 < \mu \leq L$ tais que

$$\mu I \preceq \nabla^2 f(x) \preceq LI, \ \ \forall x.$$

Assim, como $\nabla F(x^*) = 0$, é possível obter:

$$\begin{aligned} x^{k+1} - x^* &= x^k - x^* - t(\nabla F(x^k) - \nabla F(x^*)) \\ &\quad -t(\nabla f_i(x^k) - \nabla F(x^k)) \\ &= \left(I - t\int_0^1 \nabla^2 F(x^* + \xi(x^k - x^*))\, d\xi\right)(x^k - x^*) \\ &\quad -t(\nabla f_i(x^k) - \nabla F(x^k)). \end{aligned}$$

Se a norma for tomada,

$$\begin{aligned} \|x^{k+1} - x^*\| &\leq \max\{|1 - t\mu|, |1 - tL|\} \left\|x^k - x^*\right\| \\ &\quad +t\|\nabla f_i(x^k) - \nabla F(x^k)\|. \end{aligned}$$

Para $t < \frac{2}{L}$, é possível demonstrar [3] que há uma contração em relação ao primeiro termo.

Dado x^k, têm-se que

$$\left(I - t\int_0^1 \nabla^2 F(x^* + \xi(x^k - x^*))\, d\xi\right)(x^k - x^*)$$

é constante.

Analisando a variância $\mathrm{Var}(x) = E((x - E(x))^2)$ do processo:

$$\begin{aligned} \mathrm{Var}(\|x^{k+1} - x^*\| \mid x^k) &= \mathrm{Var}(\|t(\nabla f_i(x^k) - \nabla F(x^k))\| \mid x^k) \\ &= t^2\ \mathrm{Var}(\|\nabla f_i(x^k) - \nabla F(x^k)\| \mid x^k). \end{aligned}$$

Tomando a hipótese de que a variância do gradiente estocástico seja limitada por σ, obtém-se

$$\mathrm{Var}(\|x^{k+1} - x^*\| \,|\, x^k) \leq t^2\sigma.$$

Sabe-se que, para $t < \frac{2}{L}$,

$$\|E(x^{k+1} - x^* \,|\, x^k)\| \leq \gamma \left\|x^k - x^*\right\|,$$

onde $\gamma \equiv \max\{|1 - t\mu|, |1 - tL|\} \in [0, 1)$.

Logo,

$$\begin{aligned} E\left[\left\|x^{k+1} - x^*\right\|^2 |x^k\right] &= \left\|E[x^{k+1} - x^*|x^k]\right\|^2 + Var(x^{k+1} - x^*|x^k) \\ &\leq \gamma^2 \left\|x^k - x^*\right\|^2 + t^2\sigma. \end{aligned}$$

Pela esperança total,

$$E\left[\left\|x^{k+1} - x^*\right\|^2\right] \leq \gamma^2 E\left[\left\|x^k - x^*\right\|^2\right] + t^2\sigma.$$

Então,

$$E\left[\left\|x^{k+1} - x^*\right\|^2\right] \leq \gamma^2(\gamma^2 E\left[\left\|x^{k-1} - x^*\right\|^2\right] + t^2\sigma) + t^2\sigma.$$

Portanto ,

$$E\left[\left\|x^{k+1} - x^*\right\|^2\right] \leq \gamma^{2k} E\left[\left\|x^0 - x^*\right\|^2\right] + t^2\sigma\left(1 + \gamma^2 + \gamma^4 + \cdots + \gamma^{2k-2}\right).$$

Assim,

$$E\left[\left\|x^k - x^*\right\|^2\right] \leq \frac{t^2\sigma}{1 - \gamma^2}.$$

Esse resultado indica uma convergência para uma região ao redor da solução ótima, que representa justamente o efeito da **bola de ruído**, citado anteriormente. É importante perceber que o **raio** da bola de ruído depende do tamanho do passo e da variância. Assim, reduzindo o tamanho do passo ou diminuindo a variância, o raio da bola de ruído será menor e a solução será mais próxima da exata, o que nem sempre é vantajoso, pois pode causar um *Overfitting*.

4.3 Método do Gradiente Estocástico com Redução de Variância (SVRG)

O método *SVRG*, proposto em [12], é uma adaptação do *SGD* que procura reduzir a variância da estimação do gradiente da função erro. O método recebe um parâmetro especial r e calcula o gradiente completo nas iterações múltiplas de r, chamados pontos base. Cada vez que o gradiente completo é calculado, essa informação é armazenada e, nas r próximas iterações, um exemplo de treinamento é sorteado para ter o gradiente computado. Para os demais exemplos de treinamento, utilizamos as informações que já tinham sido calculadas no ultimo ponto base.

Para fazer essa recuperação de informações, é necessário sempre armazenar a **soma dos gradientes** e os pesos utilizados no último ponto base. Por exemplo, em um problema com 5 dados de treinamento, seja G o gradiente calculado no ponto base e g_i o gradiente em relação a uma amostra i. Cada g_i é obtido pela função *backpropagation* utilizando os pesos da iteração atual. Então, o gradiente no ponto base é:

$$G = \frac{g_1 + g_2 + g_3 + g_4 + g_5}{5} = \frac{\text{SOMA\_BASE}}{5}.$$

Em uma iteração não múltipla de r, um dado de treinamento é sorteado. Supondo que o dado 2 seja sorteado, g_{2novo} é calculado, assim como g_2, que deve ser computado com os pesos da iteração base que estavama armazenados. Assim, $G_{2sorteado}$ pode ser estimado da seguinte forma:

$$G_{2sorteado} = \frac{\text{SOMA\_BASE} - g_2 + g_{2novo}}{5} = \frac{g_1 + g_{2novo} + g_3 + g_4 + g_5}{5}.$$

A forma de estimar o gradiente nas iterações não múltiplas de r que acaba de ser apresentada é enviesada. Uma forma de resolver isso, seria fazendo a divisão por 5 apenas no termo referente à soma do último ponto base:

$$G_{2sorteado} = \frac{\text{SOMA\_BASE}}{5} - g_2 + g_{2novo} = \frac{g_1 + g_2 + g_3 + g_4 + g_5}{5} - g_2 + g_{2novo}.$$

O método foi implementado e testes foram realizados para diversos tamanhos de passo (t) e para diferentes valores de r. Um dos resultados está apresentado na Figura 10.

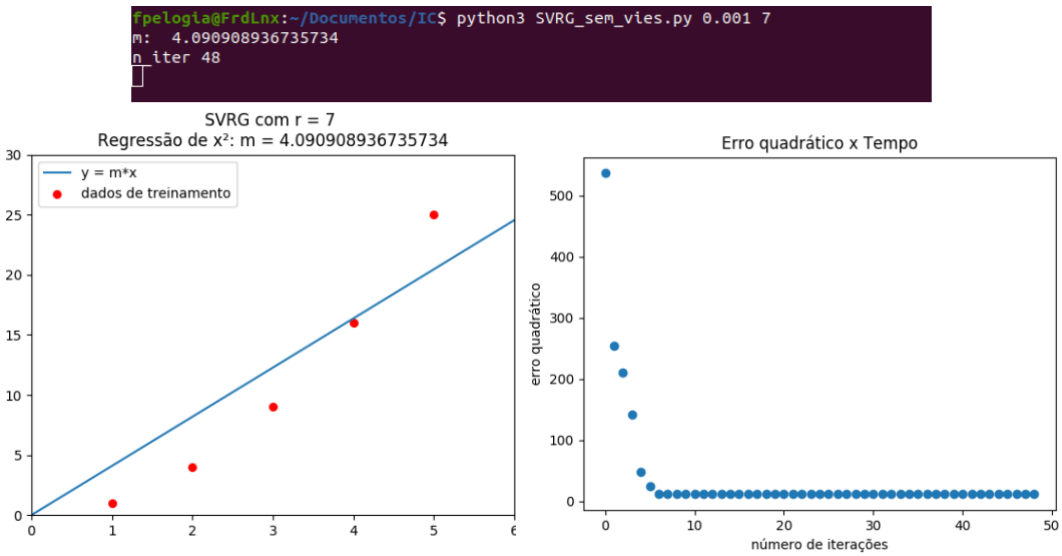

Figura 10: Resultados do teste do SVRG (sem viés) com passo 0.001 e r = 7. Fonte: Autor

4.4 Averaged Stochastic Gradient Descent (SAG e SAGA)

O método *SAG* tem uma ideia parecida com a do *SVRG*, pois calcula, na primeira iteração, o gradiente completo e, nas demais iterações, sorteia um exemplo de treinamento para calcular seu gradiente. A diferença é que não há um ponto base, isto é, sempre que um novo gradiente em relacao a um dado de treinamento for calculado, ele terá seu valor atualizado para quaisquer futuros cálculos.

Dessa forma, a maneira de recuperar as informações de iterações anteriores para estimar o gradiente é similar à apresentada para o método *SVRG*, mas agora cada dado de teinamento tem seu ultimo valor de gradiente salvo na memória. Do mesmo modo que o método *SVRG* tem uma versão com viés e uma sem, o *SAG* é enviesado e tem sua versão não enviesada, que é conhecida como **SAGA**.

Ambos os métodos foram implementados em *Python* e submetidos a diversos testes. Um teste do *SAG* está apresentado na Figura 11 e um teste do *SAGA*, na Figura 12.

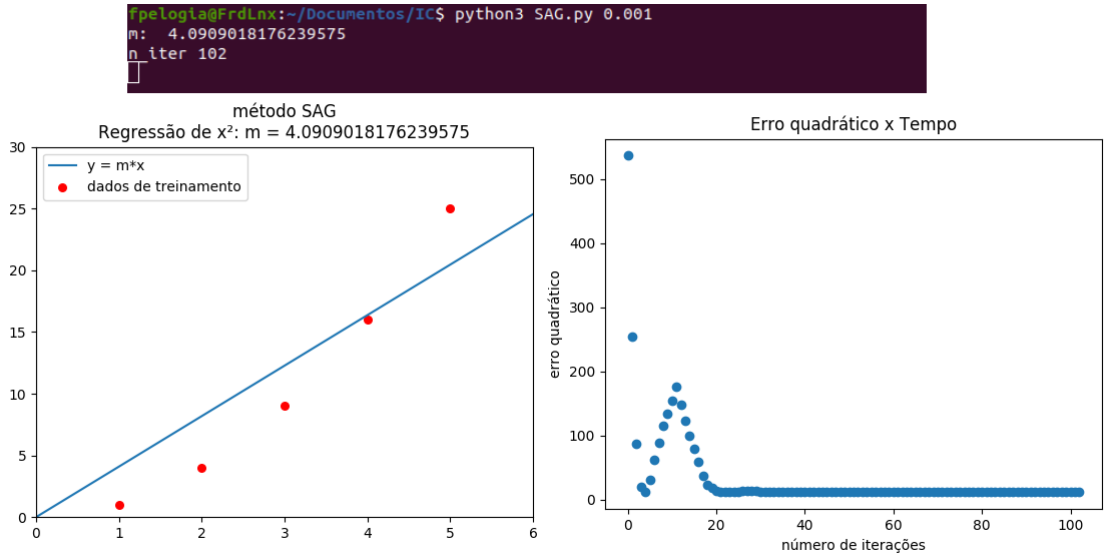

Figura 11: Resultados do teste do SAG com passo 0.001. Fonte: Autor

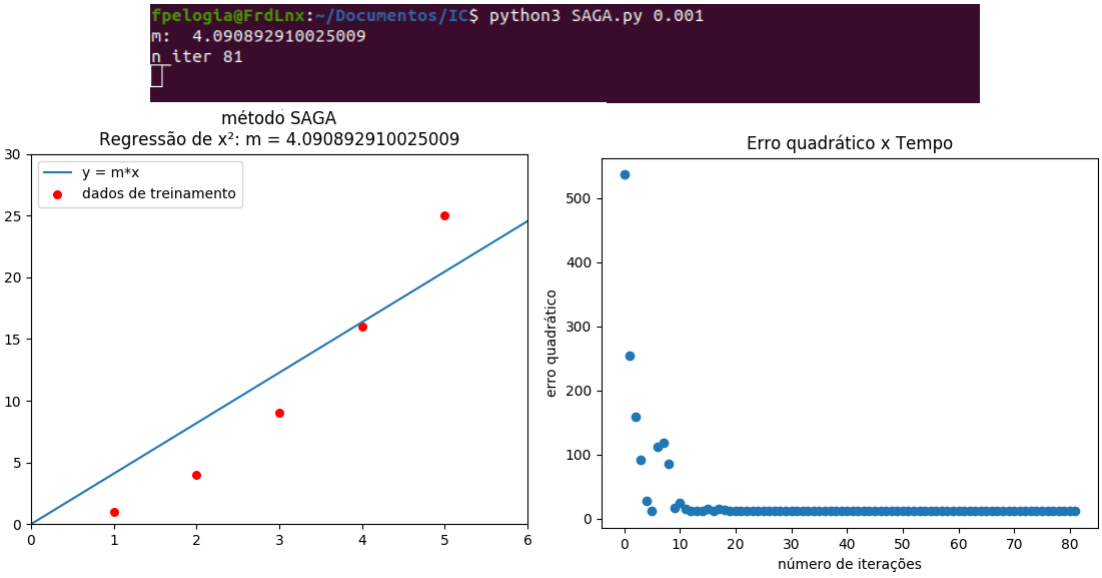

Figura 12: Resultados do teste do SAGA com passo 0.001. Fonte: Autor

4.5 Método do Gradiente com Passo Adaptativo (Adagrad)

O método *Adagrad* [9] propõe um tamanho de passo que decai a cada iteração. O método define

$$G_i^k = \sum_{j=1}^{k} [g_j(x^j)]_i^2 = G_i^{k-1} + [g_k(x^k)]_i^2,$$

sendo G_i a soma dos quadrados da coordenada i dos gradientes até a iteração k. Isso gera um acúmulo de informação. Assim, a cada iteração existirá uma matriz G, que será utilizada para o cálculo da diminuição do passo.

Basicamente, o método propõe:

$$x^{k+1} = x^k - \frac{\eta}{\sqrt{G^k + \epsilon}} \odot g_k(x^k),$$

sendo $\odot$ definido como uma multiplicação elemento a elemento.

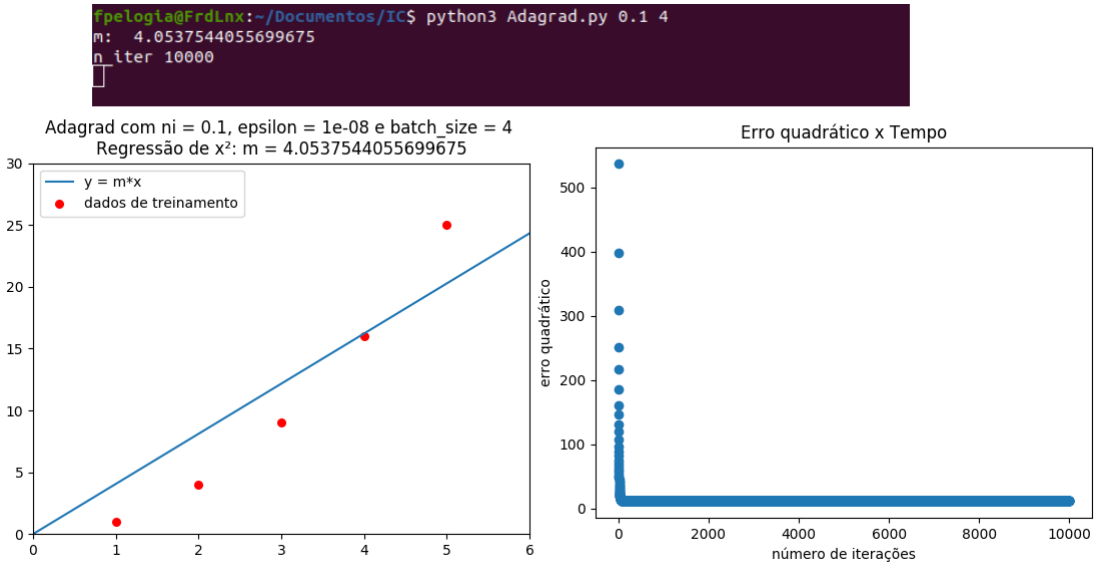

Figura 13: Resultados do teste do Adagrad com $\eta = 0.1$, $\epsilon = 1 \cdot 10^{-8}$ e tamanho do lote 4. Fonte: Autor

4.6 Método RMSProp

O método *RMSProp* é uma adaptação do *Adagrad*, mas evita o acúmulo de informação de iterações muito distantes. Isso é feito com uma média ponderada, que define uma taxa de decaimento para os termos de iterações anteriores. O termo G é calculado da seguinte forma:

$$G_i^k = \gamma G_i^{k-1} + (1 - \gamma)[g_k(x^k)]_i^2.$$

Assim, o passo adaptativo é dado de modo idêntico ao do *Adagrad*:

$$x^{k+1} = x^k - \frac{\eta}{\sqrt{G^k + \epsilon}} \odot g_k(x^k).$$

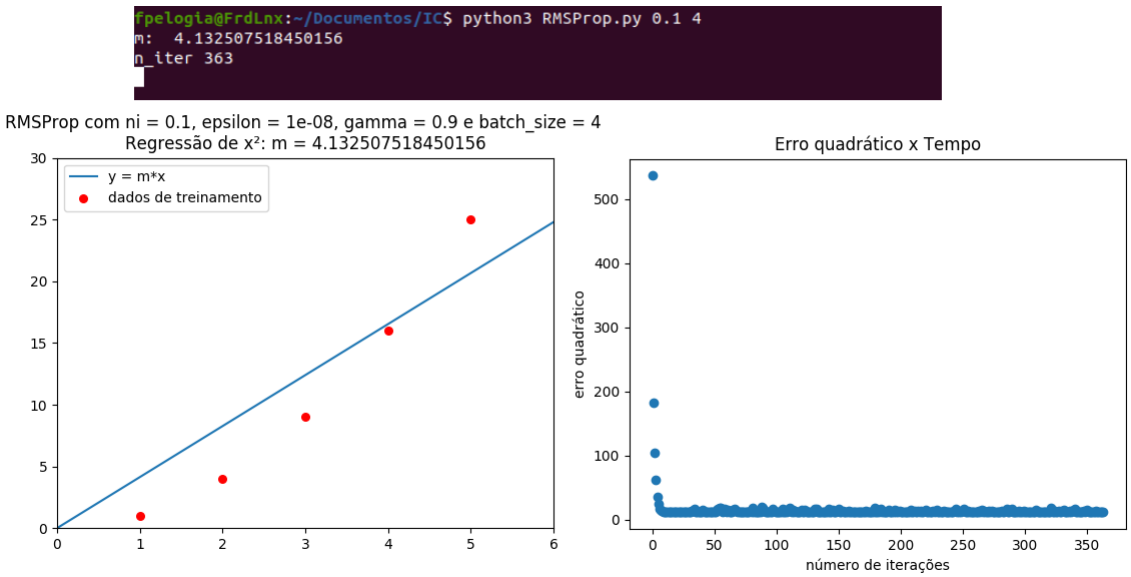

Figura 14: Resultados do teste do RMSProp com $\eta = 0.1$, $\epsilon = 1 \cdot 10^{-8}$, $\gamma = 0.9$ e tamanho do lote 4. Fonte: Autor

Note, na Figura 14, que o método converge mais rapidamente que o *Adagrad*, porém apresenta mais ruído.

4.7 Métodos com Momento (Polyak e Nesterov)

Os métodos com momento visam acelerar a convergência, evitando dar passos longos ou curtos demais. Isso é feito adicionando uma parte da direção anterior à atual. Sabe-se que a direção anterior é

$$m^k = x^k - x^{k-1}.$$

Assim, nos métodos com momento de *Polyak*, a direção atual é

$$x^{k+1} = x^k - t_k g_k(x^k) + \gamma_k m^k,$$

onde γ_k é um valor de 0 a 1 que representa o fator de decaimento da influência das direções anteriores. Geralmente t_k e γ_k são definidas constantes e $g_k(x^k) = \nabla F(x^k)$.

É importante saber que o momento de Polyak pode falhar em alguns momentos, pois para certas funções ele pode dar passos cíclicos e ficar estagnado.

Um outro método com momento amplamente conhecido e utilizado é o de *Nesterov*. A diferença é que, neste método, o gradiente é calculado após se dar o passo com o termo de momento. Sendo m^k a direção anterior, como previamente definida, o passo do método é dado da seguinte forma

$$x^{k+1} = x^k - t_k g_k(x^k + \gamma_k m^k) + \gamma_k m^k.$$

O que ocorre é um passo intermediário $z^k = x^k + \gamma_k m^k$, seguido do passo $x^{k+1} = z^k - t_k g_k(z^k)$. Assim, é garantido que o método irá tomar uma direção de descida em z^k. Da mesma forma, $g_k(x^k) = \nabla F(x^k)$, t_k e γ_k são definidas constantes.

Foram feitas implementações dos algoritmos em Python e vários testes foram realizados. As Figuras a seguir apresentam o resultados para o *SVRG* com momento de Polyak e para o *SGD* com momento de Nesterov.

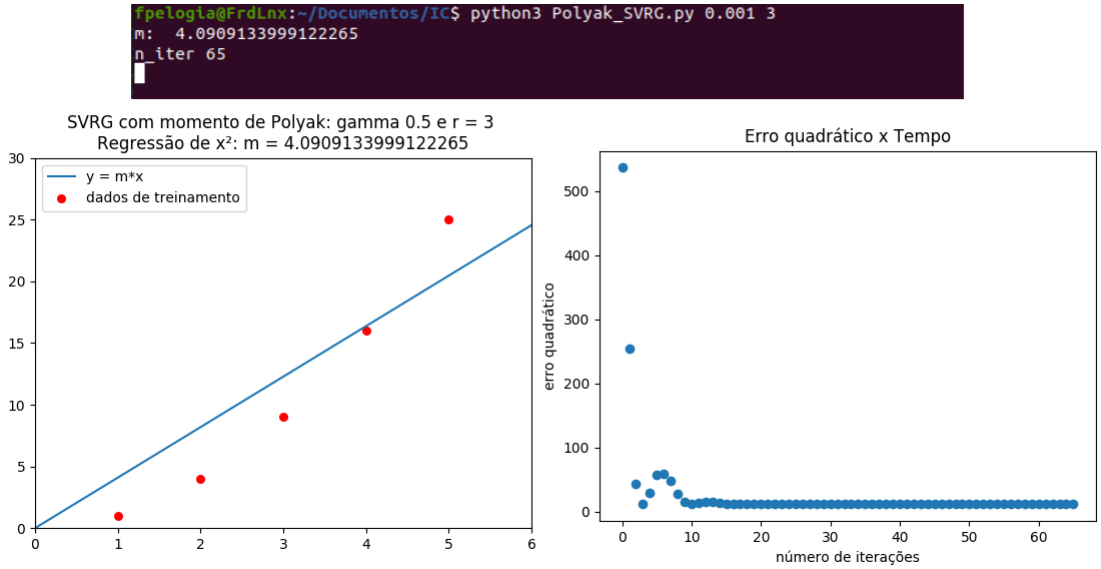

Figura 15: Resultados do teste do SVRG com momento de Polyak com passo 0.001 , r = 3 e $\gamma = 0.5$. Fonte: Autor

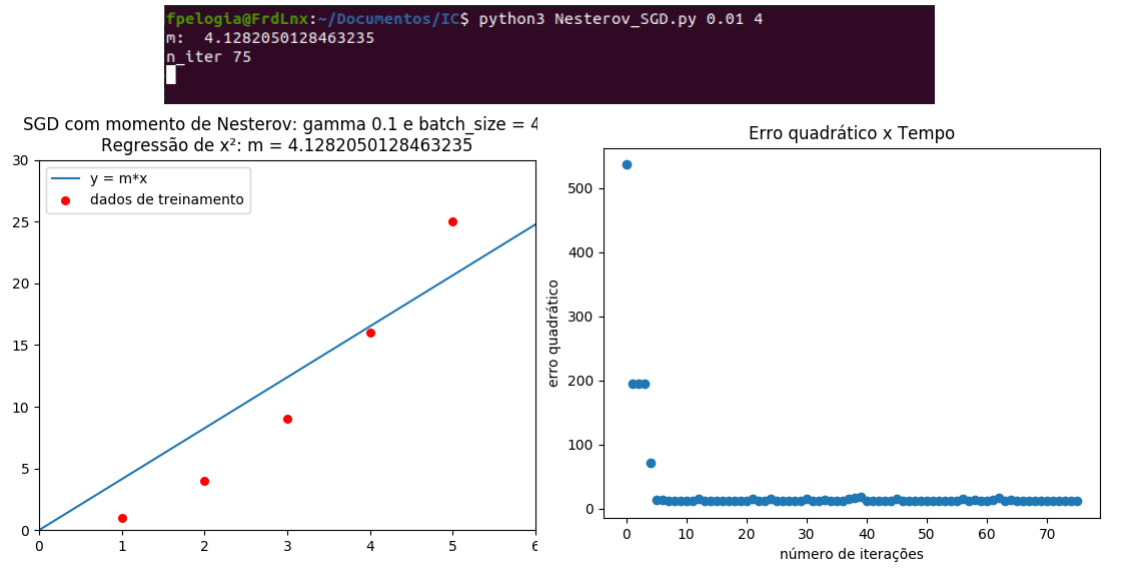

Figura 16: Resultados do teste do SGD com momento de Nesterov com passo 0.01 , $\gamma = 0.1$ e tamanho do lote 4. Fonte: Autor

4.8 Método adaptativo com momento (Adam)

O método *Adam*, proposto em [13], combina as estratégias de passo adaptativo com a adição de um termo de momento. É, atualmente, um dos métodos mais conhecidos e eficientes na área de *Deep Learning*.

Basicamente, um resumo de como os passos do método são dados é:

$$x_i^{k+1} = x_i^k - \frac{\eta}{\sqrt{\hat{G}_i^k + \epsilon}} \hat{m}_i^k.$$

Os termos $\hat{m}^k$ e $\hat{G}_i^k$ são estimadores que, para não terem viés, são definidos da seguinte maneira:

$$\hat{m}^k = \frac{m^k}{1 - (\gamma_1)^k},$$

$$\hat{G}_i^k = \frac{G_i^k}{1 - (\gamma_2)^k}.$$

O algoritmo foi implementado em Python e vários testes foram realizados. O resultado de um dos testes está na Figura 17.

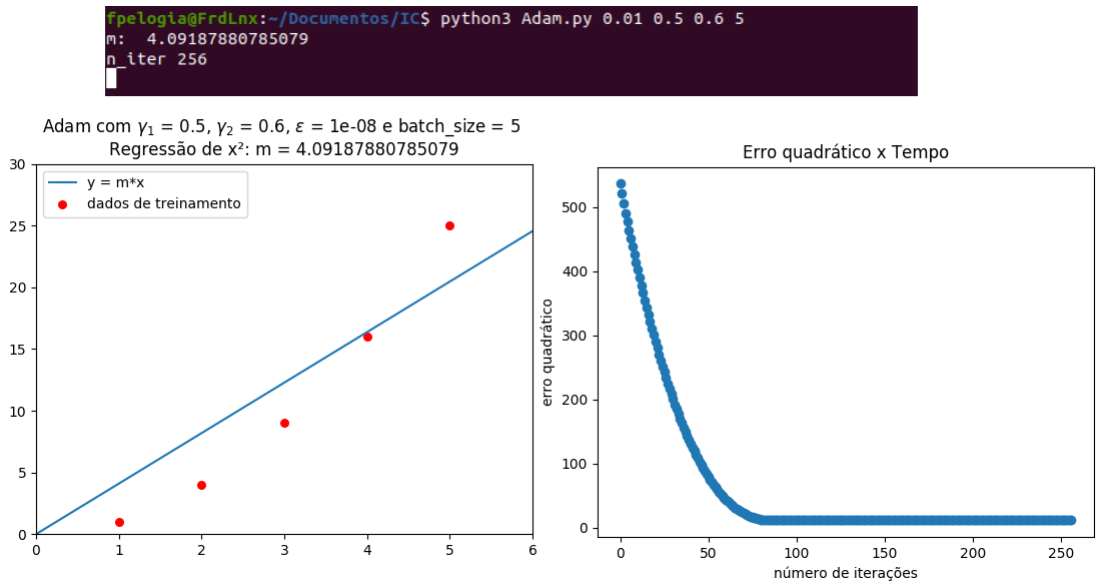

Figura 17: Resultados do teste do Adam com $\gamma_1 = 0.5$, $\gamma_2 = 0.6$, $\eta = 0.001$, $\epsilon = 10^{-8}$ e tamanho do lote 5. Fonte: Autor

5 Estudo de métodos estocásticos combinados aplicados a redes neurais

Foi realizada uma revisão bibliográfica do artigo [8], que analisa a combinação das estratégias de alguns dos métodos da seção anterior. No trabalho abordado pelo artigo, foram feitos testes aplicados a regressão logística e a redes neurais.

Nos testes para regressão logística, observou-se que a combinação do SVRG com os métodos *SGD*, *Adam* e *RMSProp* trouxe resultados consideravelmente melhores do que os métodos isolados. Em particular, a combinação do *SVRG* com o método *Adam* se destacou nos experimentos, apresentando um grande ganho de desempenho em relação aos testes com *SGD*.

Os testes para redes neurais utilizaram uma rede com 2 camadas escondidas de 1000 neurônios cada, funcão de ativação $f(x) = max(x, 0)$ (ReLU), e 25% de abandono (*dropout*). Diferentemente dos testes para regressão logística, os testes para redes neurais mostraram um desempenho pior dos métodos quando combinados com *SVRG*. Assim, é possível notar que estratégias de redução de variância não são efetivos para aplicações com redes neurais.

6 Familiarização com a base de dados Credit Card Fraud Detection

A base de dados *Credit Card Fraud Detection* contém transações feitas com cartão de crétido realizadas em Setembro de 2013, por consumidores europeus. O número de fraudes na base de dados representa 0.172% do total, isto é, 492 dentre as 284,807 transações. Todos os dados são válidos, não havendo valores nulos.

O conjunto de dados é composto por 31 colunas ou variáveis: 30 de entrada e 1 de reposta. Das 30 variáveis de entrada, 2 têm significado conhecido: "Time", que contém o valor em segundos decorrido desde a primeira transação da base de dados, e "Amount", que representa o valor movimentado na transação. As 28 demais variáveis de entrada, denominadas $V1, V2, \cdots, V28$, são os **componentes principais** resultantes da aplicação da técnica de redução de dimensionalidade *Principal Component Analysis (**PCA**)*, devido a problemas de confidencialidade. A técnica *PCA* consiste em aplicar uma transformação ortogonal a variáveis que podem estar correlacionadas de modo a gerar novas variáveis não correlacionadas e de menor dimensão [11]. A variável de resposta, denominada "Class", possui valor 0 ou 1 para cada transação, indicando se essa é ou não fraudulenta.

Primeiramente, foi feito o *download* do arquivo "creditcard.csv", com os dados, e de um arquivo ".ipynb", retirado de [18]. No segundo arquivo, que é um *Jupyter notebook*, havia uma análise dos dados e a aplicação de algumas técnicas de *Deep learning*, utilizando ferramentas de bibliotecas prontas, como **Keras** e **Scikit-learn**. Foi feita, então, a leitura e execução do *notebook* baixado para entender como devia ser feito o uso da base de dados e também como os dados estavam distribuidos e relacionados.

Após essa etapa, foram feitos vários experimentos através da modificação do código baixado e testanto seu funcionamento em diferentes cenários. Os principais alvos da experimentação foram a arquitetura da rede neural, o método de otimização utilizado e a formulação da função de erro.

Os melhores resultados foram obtidos utilizando a arquitetura de rede proposta em [18], juntamente com a função de erro *Binary crossentropy*, famosa e largamente utilizada para problemas de decisão binária. Na descrição da base de dados, na plataforma *Kaggle*, estava sendo dito que matrizes de confusão não são suficientemente significativas para analisar base de dados não equilibradas. Também na descrição do *dataset*, estava sendo sugerida a utilização da técnica *Area Under the Precision-Recall Curve (**AUPRC**)*. Após uma breve pesquisa, com base em [17] e [7], sobre o funcinamento e o significado dessa técnica, ela foi devidamente implementada e adicionada ao código como meio de comparação entre os métodos.

Modelos de classificação binária definem 2 classes para os dados: positivos ou negativos. Como, ao fazer uma predição, o modelo pode acertar ou errar, existem 4 possíveis eventos, segundo [7]:

- Verdadeiro Positivo (**TP**) - Dados positivos classificados como positivos.
- Verdadeiro Negativo (**TN**) - Dados negativos classificados como negativos.
- Falso Positivo (**TP**) - Dados negativos classificados como positivos.
- Falso Negativo (**TP**) - Dados positivos classificados como negativos.

A medida *Precision* representa a porção dos exemplos positivos corretemante classificados e é definida como $Precision = \frac{TP}{TP+FP}$. Já a medida *Recall*, também conhecida como Sensibilidade, é definida como $Recall = \frac{TP}{TP+FN}$.

A construção de uma curva de *Precision-Recall* é feita registrando, no plano, os pontos (*Recall, Precision*) avaliados com diferentes critérios de arredondamento, variando o valor limiar para essa decisão. Segundo [17], a área sob a curva *Precision-Recall* (AUPRC) é efetiva para a comparação de classificadores agindo em bases de dados não balanceadas.

Assim, estão apresentadas as curvas de *Precision-Recall* dos testes feitos utilizando a rede proposta em [18], com 4 diferentes métodos de otimização, na Figura 19. Note os valores de área sob a curva de *Precision-Recall* (**AUC**), localizados na legenda da imagem, pois esses definem a métrica de comparação utilizada. O método que mais se destacou foi o *Adam*, embora todos tenham ficado com uma acurácia parecida, como pode ser visto na Figura 18. A curva de *Precision-Recall* foi construída utilizando uma função da biblioteca **Scikit-learn**.

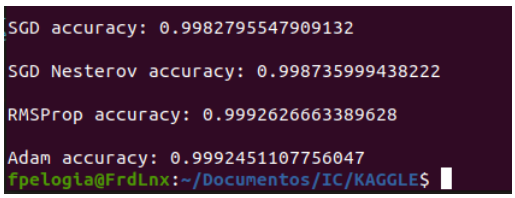

Figura 18: Acurácia dos métodos *SGD*, *SGD* com momento de Nesterov ($\gamma = 0.9$), *RMSProp* e *Adam*. Fonte: Autor

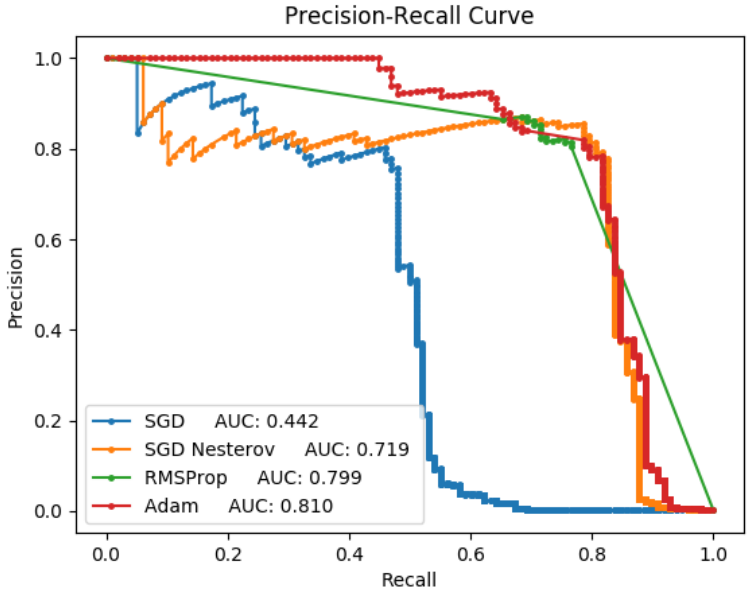

Figura 19: Curvas de *Precision-Recall* dos métodos *SGD*, *SGD* com momento de Nesterov, *RMSProp* e *Adam*. Fonte: Autor

Descrição e avaliação do apoio institucional recebido no período

A instituição forneceu todo o apoio necessário para realização do projeto. As reuniões aconteceram em salas da universidade e grande parte do desenvolvimento do projeto foi feito utilizando a internet do campus. Sempre que recursos computacionais foram necessários, eles estiveram disponíveis.

Plano de atividades para o próximo período

A primeira tarefa a ser realiazada no próximo período é o estudo dos artigos [14] e [15], que estava previsto para acontecer no primeiro semestre de atividades, porém foi atrasado pelo deslocamento da data de inicio do projeto em 1 mês.

Após isso, seguimos com o planejamento inicial, que consiste em:

- Experimentar a aplicação dos algoritmos implementados em Python, sem utilização de bibliotecas prontas de *machine learning*, à base de dados Credit Card Fraud Detection.
- Estudo do método de Levenberg–Marquardt, com base em [2].
- Estudo teórico de [4].
- Testes numéricos associados a versões estocásticas presentes em [4].
- Aplicações de técnicas de [4] no problema de interesse.

Além disso, será planejada uma apresentação do projeto no VI Congresso Acadêmico Unifesp. Há, também, a intenção de solicitar uma apresentação do projeto na terceira edição do Simpósio de Engenharia, Gestão e Inovação (SENGI), que será realizado durante os dias 18 e 19 de junho de 2020 no CDI da Universidade de São Paulo - USP, utilizando os recursos da reserva técnica da bolsa.
Por fim, por volta do mês de Agosto de 2020 será iniciada a redação do Relatório Científico Final a ser enviado à FAPESP até o dia 10/09/2020.

Utilização de recursos da reseva técnica da bolsa

Durante o período abordado por este relatório, não houve utilização dos recursos financeiros da reserva técnica da bolsa. Há, entretanto, a intenção de utilizar esses recursos no próximo período, a fim de participar do Simpósio de Engenharia, Gestão e Inovação (SENGI), que será realizado durante os dias 18 e 19 de junho de 2020.

Referências

[1] Emin Aleskerov, Bernd Freisleben e R. Bharat Rao. "CARDWATCH: a neural network based database mining system for credit card fraud detection". Em: *Proceedings of the IEEE/IAFE 1997 Computational Intelligence for Financial Engineering (CIFEr)* (1997), pp. 220–226.

[2] K. A. Benatti, T. S. Nazaré e L. F. Bueno. *O método de Levenberg-Marquardt estocástico aplicado à redes neurais artificiais*. Artigo submetido. 2019.

[3] Léon Bottou, Frank E. Curtis e Jorge Nocedal. "Optimization Methods for Large-Scale Machine Learning". Em: (2016). arXiv: 1606.04838 [stat.ML].

[4] L. F. Bueno e J. M. Martínez. *On the complexity of solving feasibility problems*. Rel. técn. 2018. URL: http://www.optimization-online.org/DB_HTML/2018/11/6929.html.

[5] André C.P.L.F. de Carvalho. *Redes Neurais Artificiais*. URL: http://conteudo.icmc.usp.br/pessoas/andre/research/neural/.

[6] Varun Chandola, Arindam Banerjee e Vipin Kumar. "Anomaly Detection: A Survey". Em: *ACM Comput. Surv.* 41.3 (2009), 15:1–15:58. DOI: 10.1145/1541880.1541882. URL: http://doi.acm.org/10.1145/1541880.1541882.

[7] Jesse Davis e Mark Goadrich. "The Relationship between Precision-Recall and ROC Curves". Em: ICML '06 (2006), 233–240. DOI: 10.1145/1143844.1143874. URL: https://doi.org/10.1145/1143844.1143874.

[8] L. F. Dos Santos e L. F. Bueno. *Stochastic Variance Reduction with Adaptive Optimization Methods*. LI Simpósio Brasileiro de Pesquisa Operacional, 2019, Limeira. Anais Eletrônicos, 2019. v. 2. 2019.

[9] John C. Duchi, Elad Hazan e Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization." Em: *J. Mach. Learn. Res.* 12 (2011), pp. 2121–2159. URL: http://dblp.uni-trier.de/db/journals/jmlr/jmlr12.html#DuchiHS11.

[10] Ryohei Fujimaki, Takehisa Yairi e Kazuo Machida. "An approach to spacecraft anomaly detection problem using kernel feature space". Em: *Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining - KDD 05* (2005). DOI: 10.1145/1081870.1081917.

[11] I. Goodfellow, Y. Bengio e A. Courville. *Deep Learning*. Massachussets: MIT press, 2016.

[12] Rie Johnson e Tong Zhang. "Accelerating Stochastic Gradient Descent using Predictive Variance Reduction". Em: *NIPS*. 2013.

[13] Diederik P Kingma e Jimmy Ba. "Adam: A method for stochastic optimization". Em: *arXiv preprint arXiv:1412.6980* (2014).

[14] F. S. Mennini, L. Gittoa, Russoa S., A. Cicchetti, M. Ruggeri, S. Coretti, G. Maurelli e P. M. Buscema. "Artificial neural networks and their potentialities in analyzing budget health data: an application for Italy of what-if theory". Em: *Quality & Quantity: International Journal of Methodology* 51.3 (2016), pp. 1261–1276.

[15] F. S. Mennini, L. Gittoa, Russoa S., A. Cicchetti, M. Ruggeri, S. Coretti, G. Maurelli e P. M. Buscema. "Does regional belonging explain the similarities in the expenditure determinants of Italian healthcare deliveries? An approach based on Artificial Neural Networks". Em: *Economic Analysis and Policy* (2017).

[16] F. J. R. Pelogia. *Implementações IC*. 2020. URL: https://repl.it/@Pelogia/Implementacoes-IC.

[17] Takaya Saito e Marc Rehmsmeier. "The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets". Em: *PLOS ONE* 10.3 (2015), pp. 1–21. DOI: 10.1371/journal.pone.0118432. URL: https://doi.org/10.1371/journal.pone.0118432.

[18] Fares Sayah. *Credit Card Fraud Detection — Keras*. URL: https://www.kaggle.com/faressayah/credit-card-fraud-detection-keras.

[19] Clay Spence, Lucas Parra e Paul Sajda. "Detection, Synthesis and Compression in Mammographic Image Analysis with a Hierarchical Image Probability Model". Em: MMBIA '01 (2001), p. 3.