Relatório Final 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: 10/02/2020 a 31/08/2020

Departamento de Ciência e Tecnologia
Universidade Federal de São Paulo
Julho 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[5] recente proposto por E. G. Birgin, L. F. Bueno e J. M. Martínez, On the complexity of solving feasibility problems with regularized models , 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 10/02/2020 a 31/08/2020. Neste intervalo de tempo, as seguintes atividades foram concluídas:

- Estudo dos artigos [11] e [10].
- 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 [5].
- Testes associados a [5].

Como planejávamos, foi submetido um resumo e um **vídeo-poster** para apresentação no Congresso Acadêmico UNIFESP 2020, que ocorreu virtualmente, dos dias 13 a 17 de julho.

Além disso, participamos do desenvolvimento da *Ferramenta de apoio ao planejamento de salas de aula pós pandemia de Covid-19*, que é um site com objetivo de otimizar as posições das carteiras nas salas de aulas de acordo com as medidas de distanciamento propostas pelas autoridades sanitárias. Esse projeto foi desenvolvido com pesquisadores da Instituições Unifesp, UEM e IFSP. Assim, houve contato com técnicas de empacotamento, otimização com restrições e a utilização do software ALGENCAN, do projeto TANGO[4].

1 Estudo dos artigos [11] e [10]

O artigo [11] aplica redes neurais artificiais ao problema de identificar os determinantes das despesas das unidades locais de saúde da Itália, chamadas de **LHUs** (Local Health Units). Segundo o mesmo, a utilização das redes neurais é vantajosa pois detecta o que são chamadas de dependências difusas, considerando variáveis que não são completamente captadas por dependências funcionais. A ideia é compreender as similaridades nos dados orçamentarios de saúde em diferentes localidades na Itália.

O artigo [10] tem intuito de identificar, dentre os dados de LHUs italianas, sinais de ineficiência e mal gerenciamento dos gastos. Ele investiga a aplicação de um tipo específico de redes neurais, chamado de **AWIT**[6](Artificial Neural Networks What-If Theory) e testa a utilização de uma região italiana como referência para tirar conclusões sobre dados de outras regiões, obtendo resultados favoráveis. O artigo também afirma que , como os dados orçamentários do sistema de saúde italiano são padronizados e bem detalhados, a aplicação de redes neurais é promissora para esse tipo de identificação.

Em uma apresentação do grupo **Semeion**[12], são comparadas diversas técnicas para detectar fraudes em dados de seguros de saúde na Itália. Os testes com as redes **AWIT** foram realizados utilizando o **Semeion Software #68** e notou-se que essa técnica se destacou em detectar seguros de saúde anômalos baseando-se no número de apólices de seguro direcionadas a certas empresas.

2 Aplicação dos algoritmos implementados ao problema de deteção de fraudes

A forma como os métodos estavam implementados não era adequada para processar os dados do dataset do Kaggle, além de não possibilitar a alteração da estrutura da rede de forma dinâmica. Procurávamos, então, algum framework que nos permitisse montar uma rede de forma fácil e receber o gradiente em relação aos parâmetros para os refinarmos manualmente de diferentes maneiras. Um exemplo de interface que permite criação dinâmica de rede é a da bilioteca Keras[8] e está apresentada a seguir.

Listing 1: Interface do Keras

```
model = keras.models.Sequential([
    keras.layers.Dense(units=16, input_dim=30, activation="relu"),
    keras.layers.Dense(units=24, activation="relu"),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(20, activation="relu"),
    keras.layers.Dense(24, activation="relu"),
    keras.layers.Dense(1, activation="sigmoid"),
])
```

A solução foi alterar as implementações para o paradigma de orientação a objeto, construindo uma pequena biblioteca de *Deep Learning* com base na *joelnet*[7], de Joel Grus. Assim, podemos organizar tudo de forma modularizada, sendo possível criar uma interface inspirada na mostrada acima.

Módulos do repositório joelnet

A biblioteca de Joel era composta, originalmente, por 7 módulos.

- O arquivo **data.py** controla a separação dos dados em "mini lotes"(*batches*).
- O arquivo **layers.py** define o comportamento genérico de uma camada e também as diferentes funções de ativação possíveis.
- O arquivo **loss.py** define as possíveis formulações da função Erro da rede. Apenas a *Mean Squared Error* está implementada.
- O arquivo **nn.py** define a Classe *NeuralNetwork*, que é a mais importante da biblioteca. Essa Classe é a portadora dos métodos responsáveis pelo Feed-Forward e pelo Backpropagation.
- O arquivo **optim.py** define os métodos de otimização que poderão ser utilizados para o treinamento das redes. Originalmente possuía apenas o SGD (Método do Gradiente Estocástico).
- O arquivo **train.py** define a função *train*, que é responsável pelo treinamento da rede neural.
- O arquivo **tensor.py** define o tipo abstrato *Tensor*, que, para nossa implementação simples, nada mais é do que o *ndarray* da biblioteca Numpy.

Alterações e adições no módulo joelnet

Para utilizarmos o repositório apresentado, foram necessárias algumas modificações e a adição de algumas funcionalidades.

- Em **layers.py** foram adicionadas as funções de ativação Sigmoid e reLu, além de suas primeiras derivadas.

$$Sig(x) = \frac{1}{1+e^{-x}}, \qquad Sig'(x) = Sig(x) \cdot (1 - Sig(x))$$

$$reLu(x) = \frac{x + \sqrt{x^2+\epsilon}}{2}, \qquad reLu'(x) = \frac{1}{2}\left(\frac{x}{\sqrt{x^2+\epsilon}} + 1.0\right)$$

- Em **data.py** foi alterada a forma de sorteio de batches. Agora, para cada *epoch*, embaralhamos os dados e retiramos amostras adjuntas do tamanho do *batch_size*.
- Em **nn.py**, o método chamado *params_and_grads* ganhou 3 novas versões, para conseguir retornar a matriz Jacobiana e também valores antigos de parâmetro e de gradiente, pois são necessários para alguns dos métodos de otimização implementados.

- Em **train.py**, a função *train* agora retorna uma lista chamada *loss_list*, que contém os valores do erro de cada *epoch*, definido como a soma dos erros de cada batch avaliado. Essa lista pode ser utilizada para construir um gráfico do erro ao longo das epochs.

- Criado um módulo **jacobian.py** com 2 funções auxiliares para cálculo da matriz jacobiana.

- Em **optim.py**, temos agora os seguintes métodos de otimização:

  - Método do Gradiente Estocástico - SGD
  - RMSProp
  - SGD com momento de Nesterov
  - Adam
  - Método de Barzilai–Borwein
  - Levenberg Marquardt
  - Levenberg Marquardt com condição de busca linear

  Nas implementações atuais, foram adicionados novos tipos de parâmetros. Agora, cada camada, após receber o conteúdo da anterior multiplicado pelos pesos, soma um novo parâmetro, conhecido por *bias* (viés), antes de aplicar a função de ativação. Sendo assim, cada camada L_i tem valor $f_i(L_{i-1} \cdot W_i + b_i)$. Por consequência, o método *backpropagation* teve que ser atualizado, agora fazendo a regra da cadeia também para calcular as derivadas parciais em relação a esses novos termos.

Testando as novas implementações

Para testar as novas implementações dos métodos, cada um deles foi avaliado em 2 cenários.

Problema 1: Regressão de $y = x^2$

O problema é uma regressão simples da função $y = x^2$. O conjunto de treinamento é composto pelos números de 1 a 20 e o resultado esperado é composto pelo quadrado de cada um desses números. Por ser um problema mais rápido de se resolver, foi importante para pequenos testes e ajustes durante a implementação.

Problema 2: Deteção de Fraudes na base de dados do Kaggle

A base de dados *Credit Card Fraud Detection* possui 284,807 transações e 492 fraudes. Foi definida uma divisão de 70% dos dados para treinamento e 30% para teste. Para as comparações entre os métodos, foi fixada uma arquitetura de rede e a mesma inicialização de seus parâmetros.

3 Estudo do Método de Levenberg–Marquardt

Primeiramente foi feito um estudo sobre alguns métodos clássicos de segunda ordem.

Método de Newton

É um dos mais clássicos métodos de segunda ordem. Ele calcula a matriz Hessiana da função erro, resolve o sistema

$$\nabla^2 f(x^k)d = -\nabla f(x^k),$$

e utiliza a direção encontrada para dar o passo $x^{k+1} = x^k + td$.
O método com $t = 1$ apresenta convergência local quadrática[9], porém só consegue convergência global procurando um t adequado com uma busca linear, como a de Armijo[1]

$$f(x^k + td^k) \leq f(x^k) + \alpha t \nabla f(x^k)^T d^k.$$

Como a matriz Hessiana possui formato $n \times n$, onde n é o número de variáveis, seu cálculo demonstra-se custoso. O método é mais caro ainda, pois além de calcular essa matriz, resolve um sistema com ela e com o gradiente, resultando em algo da ordem de $O(n^3)$.

Quasi-Newton: Método da Secante, Barzilai-Borwein e Gauss-Newton

A ideia principal dos métodos Quasi-Newton é herdar a eficácia do Método de Newton utilizando alternativas ou aproximações mais baratas da Hessiana.

Método da Secante

Propõe a escolha de uma matriz M_k tal que

$$M_k s^{k-1} = y^{k-1},$$

onde

$$s^{k-1} = x^k - x^{k-1} \text{ e } y^{k-1} = \nabla f(x^k) - \nabla f(x^{k-1}).$$

Assim, pode-se encontrar d_k tal que

$$M_k d^k = -F(x^k),$$

para que seja possível fazer

$$x^{k+1} = x^k + d^k.$$

Método de Barzilai-Borwein

Sua proposta é ser um intermediário entre o Método do Gradiente, que possui complexidade $O(n)$, e o Método da Secante, que tem complexidade $O(n^2)$. Sendo

$$s^{k-1} = x^k - x^{k-1} \text{ e } y^{k-1} = \nabla f(x^k) - \nabla f(x^{k-1}),$$

A ideia é definir $M_k = \lambda_k I$, implicando que $\lambda_k s^k = y^k$. A melhor aproximação para essa equação é

$$\lambda_k = \frac{(s^k)^T y^k}{\|s^k\|^2}.$$

Então, a direção tomada pelo método será

$$d = -\frac{1}{\lambda_k}\nabla f(x^k),$$

sendo semelhante a uma iteração do Método do Gradiente com passo $\frac{1}{\lambda_k}$. O método também funciona bem em um formato estocástico, se assemelhando, nesse caso, ao SGD com passo $\frac{1}{\lambda_k}$.

Método de Gauss-Newton

Problemas não lineares de quadrados mínimos podem ser formulados da seguinte forma

$$f(x) = \|F(x)\|^2 = \sum_{i=1}^{N} f_i(x)^2.$$

Neste caso,

$$\nabla^2 f(x) = J_F(x)^T J_F(x) + \sum_{i=1}^{N} f_i(x)\nabla^2 f_i(x).$$

Uma possível aproximação para a Hessiana é $J_F(x)^T J_F(x)$, tendo em vista que o segundo termo possui f_i, que queremos que seja pequeno e, próximo da solução, fica menor ainda.

Assim, a direção de Gauss-Newton é a direção d_k que satisfaz

$$(J_F(x)^T J_F(x))d_k = -\nabla f(x^k).$$

Segundo [3], o método é bem definido para quando $J_F(x)^T J_F(x)$ é invertível e, portanto, definida positiva.

Método de Levenberg-Marquardt

Por fim, o Método de Levenberg-Marquardt é uma mistura do Método de Gauss-Newton com um passo de Barzilai-Borwein. A dreção é calculada da seguinte forma

$$(J_F(x)^T J_F(x) + \lambda I)d_k = -\nabla f(x^k),$$

onde λ pode ser definido de diversas maneiras. Para a implementação realizada, foi considerado

$$\lambda = ||\nabla f(x^k)||.$$

Note que pequenos valores de λ deixam o método mais parecido com o Método de Gauss-Newton, enquanto valores altos de λ aproximam a um passo de Barzilai-Borwein.

4 Estudo teórico de [5]

O artigo, definindo restrições baratas $\underline{g}(x) = 0$ e $\underline{h}(x) <= 0$ e restrições custosas $g(x) = 0$ e $h(x) <= 0$, propõe a minimização de uma função

$$\phi(x) = \frac{1}{2}(||h(x)||^2 + ||g(x)_+||^2),$$

através da minimização de um modelo de ordem **p** regularizado, como segue:

$$min \quad M_{x^k}(x) + \sigma_k ||x - x^k||^{p+1} \quad sujeito \quad a \quad \underline{h}(x) = 0 \quad e \quad \underline{g}(x) <= 0.$$

Entretanto, desconsiderando as restrições, podemos escrever:

$$\phi(x) = \frac{1}{2}(||F(x)||^2),$$

que é uma soma de quadrados, como em nosso problema de interesse. Além disso, se considerarmos o caso $\mathbf{p = 1}$, podemos pensar em

$$M_{x^k} = \nabla\phi(x^k)^T(x - x^k) + \frac{1}{2}(x - x^k)^T B_k(x - x^k),$$

o que pode levar a

$$min \quad \nabla\phi(x^k)^T(x - x^k) + \frac{1}{2}(x - x^k)^T B_k(x - x^k) + \frac{\sigma}{2}||x - x^k||^2.$$

A solução dessa formulação resumida pode ser feita igualando o gradiente de toda a expressão a zero, o que resulta em

$$\nabla\phi(x^k) + B_k(x - x^k) + \sigma(x - x^k) = 0,$$

o que implica em

$$x = x_k - (B_k + \sigma I)^{-1}\nabla\phi(x^k),$$

que é o próprio método de Levenberg Marquardt, se definirmos a matriz B_k como $J_F(x)^T J_F(x)$.

Assim, o artigo propõe uma versão do método de Levenberg Marquardt em que é imposta uma condição de descida

$$\phi(x) <= \phi(x^k) - \alpha||x - x^k||^2$$

e realizada uma busca por um valor de λ que a satisfaça. Segue um pseudo-código do processo:

Listing 2: Levenberg Marquardt com busca linear

```
x, f = LevenbergMarquardt(x[k], lambda);
while(f > f[k] - alpha*(norm(x - x[k]))^2 ){
        lambda = 2*lambda;
}
x[k + 1] = x;
```

Além disso, o artigo demonstra[5] que em alguns casos, onde as linhas de $J_F(x)$ são uniformemente linearmente independentes, a solução com precisão ε é atingida com $log(\varepsilon)$ avaliações de função. Na seção de resultados do presente relatório estão apresentados alguns testes experimentando esse resultado dentro e fora do cenário proposto.

5 Resultados e Discussões

Comparação dos métodos no Problema 1

Para as comparações referentes ao Problema 1, foi fixada uma arquitetura de rede **1-2-1**, com ativação **reLu**, **batch_size** $= 2$ e inicialização fixa dos parâmetros como

$$W_{L1\to L2} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \qquad b_{L1\to L2} = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix} \qquad W_{L2\to L3} = \begin{pmatrix} 3 & 4 \end{pmatrix}, \qquad b_{L2\to L3} = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}.$$

A Figura 1 apresenta o resultado das predições de cada método, juntamente com o valor da função erro na ultima *epoch*.

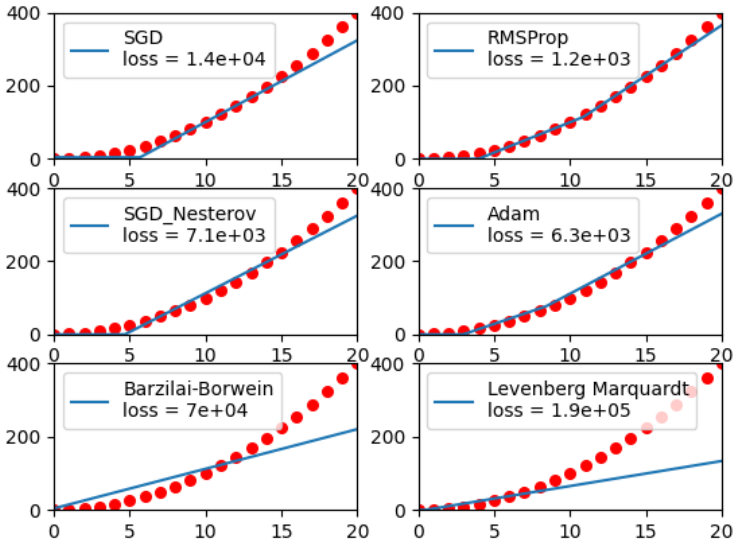


Figura 1: Resultados da aplicação dos métodos ao Problema 1 (Regressão de $y = x^2$). Fonte: Autor

Note que o melhor desempenho foi do **RMSProp**, método com passo adaptativo. Pode-se perceber que os 2 métodos quasi-Newton obtiveram os piores resultados no teste.

Comparação dos métodos no Problema 2

Para o problema 2, foi fixada uma arquitetura de rede **30-24-30-35-1**, com 3 camadas ocultas. A inicialização dos pesos foi aleatória, porém a mesma para todos os métodos. O tamanho do lote (**batch_size**) escolhido foi de 150 dados. A Figura 2 apresenta as curvas de Precision-Recall de cada um dos métodos, além do valor da área debaixo delas, indicado por AUC na legenda.

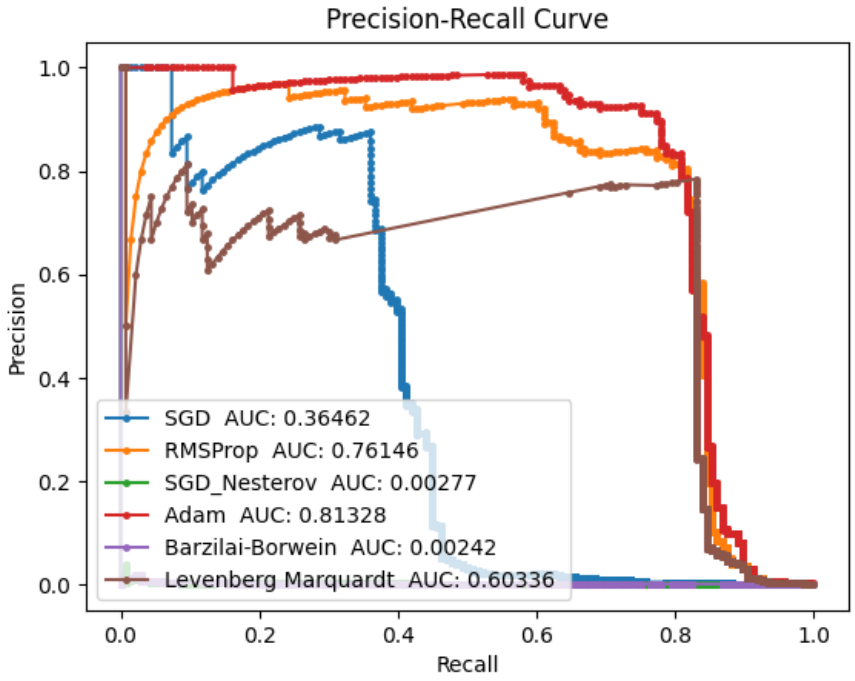


Figura 2: Comparação das curvas de Precision-Recall dos métodos aplicados ao Problema 2. Fonte: Autor

O método **Adam** foi o que mais se destacou, obtendo o maior valor de AUPRC, e os métodos **SGD_Nesterov** e **Barzilai-Borwein** tiveram um desempenho muito abaixo dos outros. Talvez com outras inicializações de parâmetros esses métodos poderiam apresentar melhorias. As curvas de Precision-Recall são montadas variando os *tresholds* de arredondamento e registrando os valores de Precision e de Recall. Foram registradas, também, as **matrizes de confusão** dos métodos para o *treshold* de 0.5. Essas matrizes estão apresentadas na Figura 3.

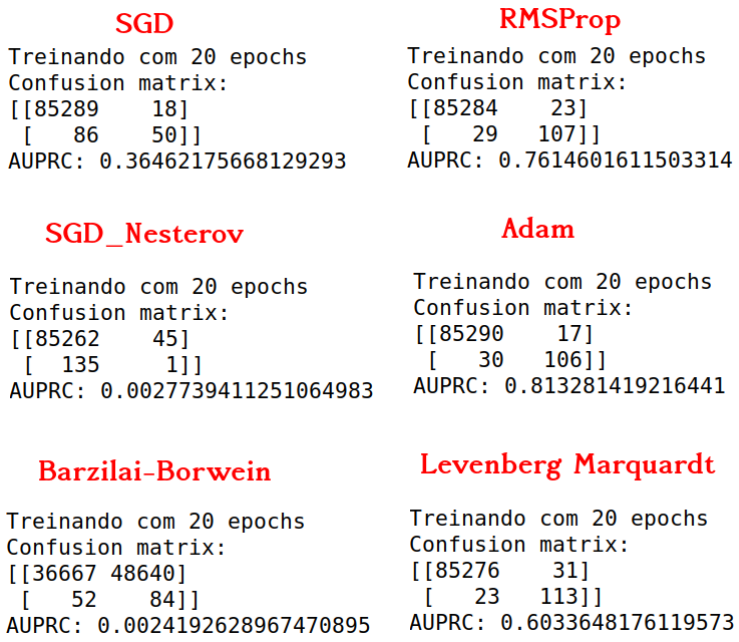


Figura 3: Comparação das matrizes de confusão dos métodos aplicados ao Problema 2. Fonte: Autor

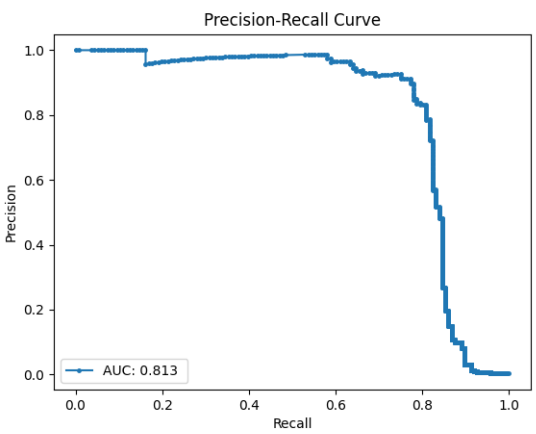
Pode-se perceber que o método com mais fraudes detectadas foi o Levenberg Marquardt, com 113 acertos, mas deixou 23 passarem e detectou indevidamente 31 transações limpas. O método RMSProp foi o segundo com mais detecções, com 107 fraudes identificadas e um desempenho mais equilibrado que o anterior. Já o método Adam foi o terceiro com mais fraudes detectadas, com 106, porém foi o mais balanceado de todos os métodos no teste. É interessante notar que a utilização da AUPRC (Área sob a curva de Precision-Recall) revela diferenças mais sutis entre os resultados de cada método do que as matrizes de confusão. Como a curva considera diversos *tresholds* de arredondamento, ela realça a forma como cada método lida com o desbalanceamento do *dataset*.

Comparação com o Keras

Foi feita uma comparação entre o método Adam implementado no projeto e o método Adam presente na biblioteca Keras, ambos aplicados ao Problema 2. Para isso, os dados foram embaralhados da mesma maneira, a arquitetura de rede **30-24-30-35-1** aplicada ao Keras e os dados divididos como nos demais testes de deteção de fraudes. Fazendo os testes com 20 *epochs* e *batch_size* 32, foram obtidos os resultados apresentados na Figura 4.

Adam implementado

```
Treinando com 20 epochs
Confusion matrix:
[[85290    17]
 [   30   106]]
AUPRC: 0.813281419216441
```



Adam do Keras

```
Treinando com 20 epochs
Confusion matrix:
[[85284    23]
 [   27   109]]
AUPRC: 0.8386217105335106
```

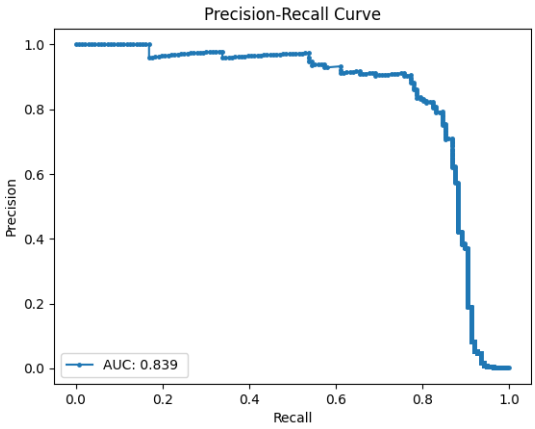


Figura 4: Comparação da implementação do Adam com o Adam da biblioteca Keras. Fonte: Autor

Observando os valores de **AUPRC**, é possivel notar que o desempenho do algoritmo implementado no presente projeto foi levemente inferior ao do método da biblioteca Keras. As matrizes de confusão apresentadas, que são para um *treshold* de 0.5, demonstram que o método implementado no projeto detectou 3 fraudes a menos, porém obteve mais acertos no geral. Claramente não há garantia de que os resultados possam ser extendidos para outros pontos iniciais ou até mesmo arquiteturas de rede, mas a proximidade dos resultados aos de uma biblioteca bem estabelecida tem caráter positivo para o projeto.

Testes com o método de [5]

Aplicado ao Problema 1, variando α

O primeiro teste realizado foi a aplicação do método ao Problema 1, com 3 diferentes valores de α, o que implica em 3 níveis de exigência durante a busca linear pelo λ. Os resultados do teste estão apresentados na Figura 5

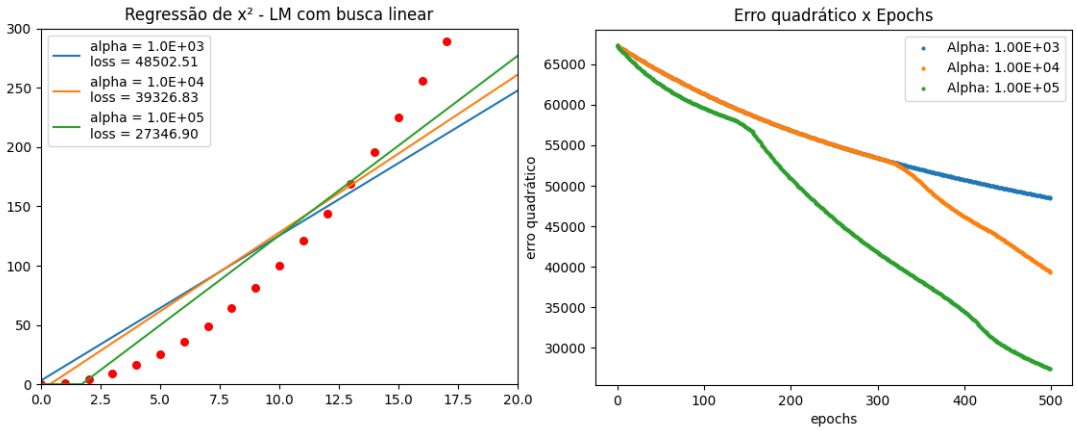


Figura 5: Comparação de diferentes valores de α para o método de [5]. Fonte: Autor

Note que maiores valores de α implicam em um decrescimento mais rápido da função, afinal serão feitas mais buscas lineares. Entretanto, o tempo de execução não escala bem com o crescimento de α, afinal cada vez mais iterações terão o loop de busca, como pode ser observado na Figura 6.

```
========= Método LM com busca linear =========
treinando com 500 epochs
alpha: 1.0E+03
Tempo gasto no treinamento: 2.8311407566070557s
========= Método LM com busca linear =========
treinando com 500 epochs
alpha: 1.0E+04
Tempo gasto no treinamento: 5.777575731277466s
========= Método LM com busca linear =========
treinando com 500 epochs
alpha: 1.0E+05
Tempo gasto no treinamento: 25.40390372276306s
fpelogia@FrdLnx:~/Documentos/IC/RESULTADOS/LM_BUSC_LIN$
```

Figura 6: Comparação do tempo de execução com diferentes valores de α para o método de [5]. Fonte: Autor

Explorando os resultados de complexidade

O artigo [5] afirma que o método atinge uma precisão ε após $|log\varepsilon|$ avaliações de função quando os vetores gradiente são uniformemente linearmente independentes. Para experimentar esse resultado, foram feitos 3 testes: um dentro do caso proposto e outros 2 fora. Para esses testes, foi utilizada uma arquitetura de rede **1-2-1**, como apresentada na Figura 7, com funções de ativação $f(x) = x$.

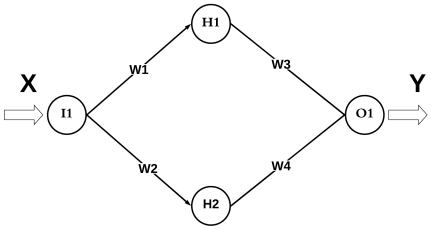


Figura 7: Rede neural 1-2-1. Fonte: Autor

Para esses testes, foram removidos os parâmetros **bias**, para que a predição resultasse em uma reta

$$y = (w_1 w_3 + w_2 w_4) \cdot x = m \cdot x,$$

que estará sempre fixada na origem e apenas a sua inclinação é variável. A inicialização escolhida foi

$$\begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \quad \text{e} \quad \begin{pmatrix} w_3 & w_4 \end{pmatrix} = \begin{pmatrix} 3 & 4 \end{pmatrix}.$$

Para o primeiro teste, apenas 1 dado de treinamento foi enviado: $x_1 = 1 \quad y_1 = 1^2 = 1$. Note que, havendo apenas um dado de treinamento, existirá apenas 1 gradiente:

$$\nabla f_1(w)^T = \begin{pmatrix} -x_1 w_3 & -x_1 w_4 & -x_1 w_1 & -x_1 w_2 \end{pmatrix},$$

o que implica que não há dependências lineares, correspondendo ao caso de interesse para a afirmação do artigo. Assim, a Figura 8 apresenta o resultado da aplicação do método a esse caso, utilizando uma precisão de 1×10^{-4} e $\alpha = 1 \times 10^2$.

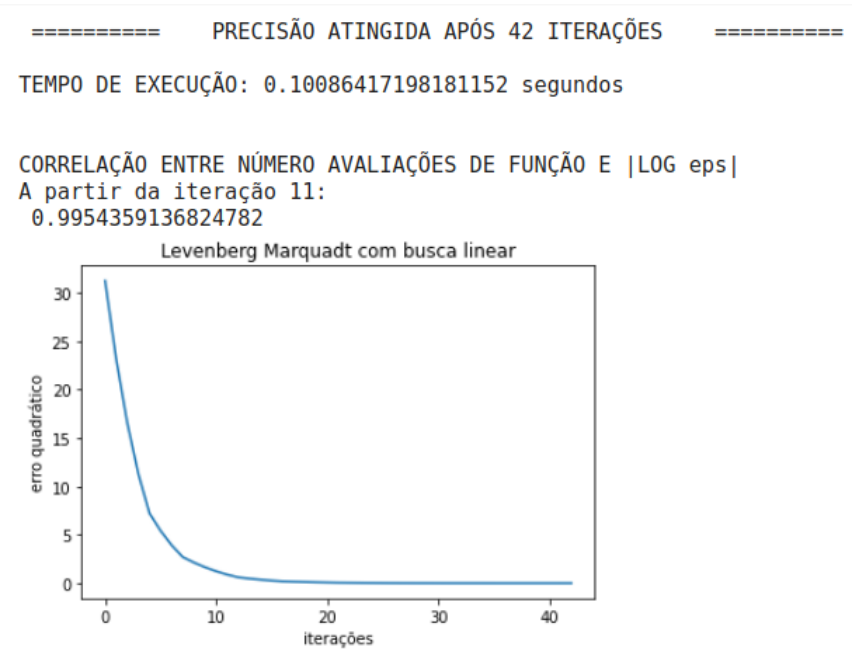


Figura 8: Teste do método de [5] com 1 dado (gradiente LI). Fonte: Autor

É possível notar que o método convergiu com facilidade e a correlação entre o número de avaliações de função e $|log\varepsilon|$ foi de aproximadamente 0.995, sendo coerente com os resultados do artigo. A correlação foi medida a partir das iterações com erro abaixo de 1.

Caso sejam enviados 2 dados de treinamento, note que os gradientes serão:

$$\begin{aligned}\nabla f_1(w)^T &= \begin{pmatrix}-x_1w_3 & -x_1w_4 & -x_1w_1 & -x_1w_2\end{pmatrix} \\ &= x1 \cdot \begin{pmatrix}-w_3 & -w_4 & -w_1 & -w_2\end{pmatrix}, \\ \nabla f_2(w)^T &= \begin{pmatrix}-x_2w_3 & -x_2w_4 & -x_2w_1 & -x_2w_2\end{pmatrix}, \\ &= x2 \cdot \begin{pmatrix}-w_3 & -w_4 & -w_1 & -w_2\end{pmatrix},\end{aligned}$$

Devido ao termo $\begin{pmatrix}-w_3 & -w_4 & -w_1 & -w_2\end{pmatrix}$, os gradientes mostram-se linearmente **dependentes**.

Os 2 outros testes, seguem a ideia apresentada na Figura 9.

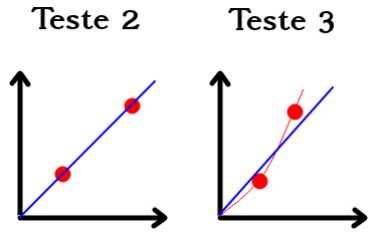


Figura 9: Ideia dos testes 2 e 3 para o método de [5]. Fonte: Autor

O segundo teste envia $x = [1, 1.5]$ e $y = [1, 1.5]$, que é um caso em que a aproximação exata por uma reta é possível, embora os gradientes inda sejam linearmente dependentes. A Figura 10 apresenta o resultado da aplicação do método a esse caso, utilizando uma precisão de 1×10^{-4} e $\alpha = 1 \times 10^2$.

```
==========    PRECISÃO ATINGIDA APÓS 14 ITERAÇÕES    ==========

TEMPO DE EXECUÇÃO: 0.052579641342163086 segundos


CORRELAÇÃO ENTRE NÚMERO AVALIAÇÕES DE FUNÇÃO E |LOG eps|
A partir da iteração 4:
 0.9906722245035238
```

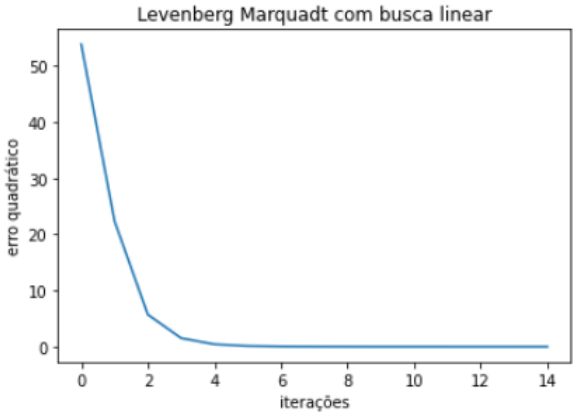


Figura 10: Teste do método de [5] com 2 dados y = x (gradientes LD, mas aproximação possível). Fonte: Autor

Embora o caso tenha gradientes linearmente dependentes, fugindo do cenário proposto pelo artigo, o caráter linear do problema permitiu uma convergência fácil, com a correlação entre o número de avaliações de função e $|log\varepsilon|$ sendo de aproximadamente 0.991. A correlação foi medida a partir das iterações com erro abaixo de 1.

O terceiro teste envia $x = [1, 1.5]$ e $y = [1, 2.25]$, que é um caso em que a aproximação do modelo não conseguirá exatidão e os gradientes seguem sendo linearmente dependentes. A Figura 11 apresenta o resultado da aplicação do método a esse caso, utilizando uma precisão de 1×10^{-4} e $\alpha = 1 \times 10^{2}$.

```
lambda muito grande
FINALIZANDO COM  44  ITERAÇÕES POR LAMBDA MUITO GRANDE
TEMPO DE EXECUÇÃO: 0.30687808990478516 segundos


CORRELAÇÃO ENTRE NÚMERO AVALIAÇÕES DE FUNÇÃO E |LOG eps|
A partir da iteração 5:
 0.3175413748501029
```

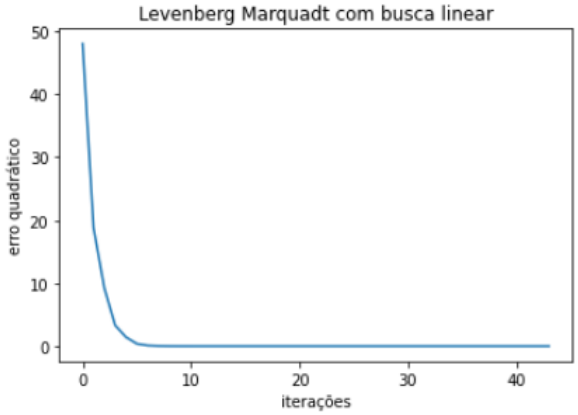


Figura 11: Teste do método de [5] com 2 dados $y = x^2$ (gradientes LD). Fonte: Autor

Note que o método teve mais dificuldade e a correlação entre o número de avaliações de função e $|log\varepsilon|$ foi de aproximadamente 0.318. Entretanto, isso ocorre pois a função não chega a zero, mas a um valor mínimo que deveria ser levado em conta para calcular essa correlação.

Assim, para encontrar esse valor, sabemos que

$$\begin{pmatrix} 1 \\ 1.5 \end{pmatrix} m = \begin{pmatrix} 1 \\ 2.25 \end{pmatrix},$$

e, por quadrados mínimos, a melhor aproximação vem de

$$\begin{pmatrix} 1 & 1.5 \end{pmatrix} \begin{pmatrix} 1 \\ 1.5 \end{pmatrix} m = \begin{pmatrix} 1 & 1.5 \end{pmatrix} \begin{pmatrix} 1 \\ 2.25 \end{pmatrix}.$$

Assim,

$$3.25m = 4.375,$$

o que implica que

$$m = \frac{35}{26} = 1.346153846.$$

Logo, desejamos ver o valor mínimo que f assume:

$$f_{min} = \frac{1}{2}((1m - 1)^2 + (1.5m - 2.25)^2)$$

$$= \frac{9}{104} = 0.08653846154 \approx 0.0865$$

Assim, podemos utilizar uma precisão de parada um pouco maior que esse valor e considerar $|log(\varepsilon - 0.0865)|$ para fazer a correlação. Dessa forma, o efeito gerado seria o mesmo de somar esse valor à função objetivo da minimização. Assim, a Figura 12 apresenta os resultados desse teste para $\alpha = 1 \times 10^2$.

```
==========    PRECISÃO ATINGIDA APÓS 13 ITERAÇÕES    ==========

TEMPO DE EXECUÇÃO: 0.029129981994628906 segundos


CORRELAÇÃO ENTRE NÚMERO AVALIAÇÕES DE FUNÇÃO E |LOG eps|
A partir da iteração 5:

 0.6938041469419514

CORRELAÇÃO ENTRE NÚMERO AVALIAÇÕES DE FUNÇÃO E |LOG (eps - 0.0865)|
A partir da iteração 5:

 0.972412123512165
```

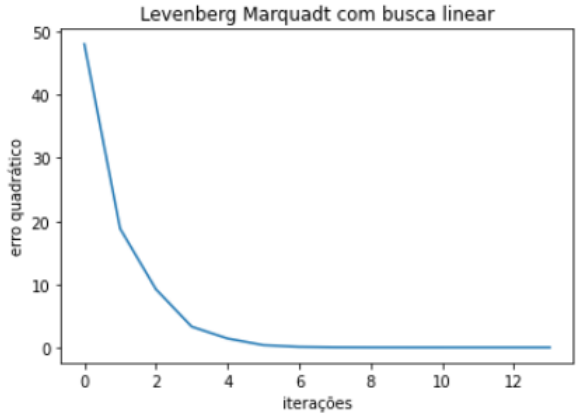


Figura 12: Teste com 2 dados $y = x^2$ (gradientes LD), utilizando o valor 0.0866 como condição de parada e $\alpha = 1 \times 10^2$. Fonte: Autor

Note que agora o método consegue atingir a precisão e que a correlação do número de avaliações de função com $|log(\varepsilon - 0.0865)|$ vale aproximadamente 0.972, seguindo o resultado de complexidade do artigo, embora os gradientes sejam linearmente dependentes.

6 Conclusão

Portanto, as realizações do período de vigência da bolsa foram importantes para o aprendizado do aluno, possibilitando o contato com assuntos que não seriam abordados durante sua graduação e podem ser relevantes para a vida profissional. Além disso, o aluno teve seu primeiro contato com o ambiente acadêmico e de pesquisa. O projeto proporcionou, também, uma maior familiaridade com a linguagem de programação Python e algumas de suas bibliotecas de matemática e ciência de dados.

Nos períodos referentes a ambos os relatórios, foram feitas revisões bibliográficas de artigos e a verificação prática de seus resultados em cenários levemente divergentes. A aplicação

dos métodos estudados à detecção de fraudes foi importante para comparar a eficácia de cada estratégia a esse tipo de problema. Isso é relevante, pois a detecção de anomalias é largamente utilizada para diferentes domínios de aplicação na atualidade.

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.

Referências

[1] Larry Armijo. "Minimization of functions having Lipschitz continuous first partial derivatives." Em: *Pacific J. Math.* 16.1 (1966), pp. 1–3. URL: https://projecteuclid.org:443/euclid.pjm/1102995080.

[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] K. A. Benatti e A. A. Ribeiro. "O Método de Levenberg-Marquardt para o Problema de Quadrados Mínimos não Linear". Em: *II Simpósio de Métodos Numéricos em Engenharia - UFPR* (2017).

[4] E. G. Birgin. *TANGO Project*. URL: https://www.ime.usp.br/~egbirgin/tango/.

[5] Martínez J. M. Bueno L. F. e E. G. Birgin. "On the complexity of solving feasibility problems with regularized models". Em: *Optimization Methods and Software (GOMS)* (2020). DOI: 10.1080/10556788.2020.1786564.

[6] Massimo Buscema e William Tastle. "Artificial Neural Network What-If Theory". Em: *International Journal of Information Systems and Social Change* 6 (2015), pp. 52–81. DOI: 10.4018/IJISSC.2015100104.

[7] Joel Grus. *joelnet - live coding deep learning library*. URL: https://github.com/joelgrus/joelnet.

[8] *Keras: the Python deep learning API*. URL: https://keras.io/.

[9] J. M. Martínez e S. A. Santos. *Métodos Computacionais de Otimização*. Campinas: Departamento de Matemática Aplicada, IMECC - UNICAMP, 1995.

[10] 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.

[11] 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).

[12] *SEMEION - Centro Ricerche di Scienze della Comunicazione*. URL: http://www.semeion.it/wordpress/en/.