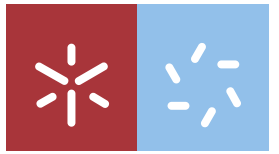
Rui Pedro Silva Teixeira *Autoencoders* lineares e *autoencoders* não lineares (ReLU)

UMinho | 2020

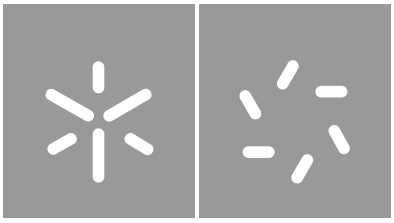


Universidade do Minho
Escola de Ciências

Rui Pedro Silva Teixeira

Autoencoders lineares e *autoencoders* não lineares (ReLU)

dezembro de 2020



Universidade do Minho
Escola de Ciências

Rui Pedro Filipe Teixeira

Autoencoders lineares e *autoencoders* não lineares (ReLU)

Dissertação de Mestrado
em Matemática e Computação

Trabalho efetuado sob a orientação do
Professor Doutor Stéphane Louis Clain
e do
Professor Doutor Luís Filipe Ribeiro Pinto

dezembro de 2020

Agradecimentos

Ao longo da realização desta dissertação, contei com o apoio de inúmeras pessoas às quais não posso deixar de agradecer.

Em primeiro lugar, dirijo o meu sentido agradecimento aos meus orientadores, Doutor Stéphane Clain e Doutor Luís Pinto, pela paciência, conhecimento transmitido, enorme dedicação e toda a disponibilidade demonstrada ao longo deste tempo todo.

Agradeço aos meus colegas de Mestrado porque também contribuíram para que isto fosse possível.

Aos meus amigos mais próximos agradeço todos os bons momentos, todas as corridas e guitarradas, que me permitiram manter o foco e não perder a cabeça.

À Juliana agradeço toda a paciência, apoio e ânimo que me deu.

Por fim, agradeço à minha família toda a ajuda e motivação. Obrigado por acreditarem em mim e por toda a força que me transmitem.

DECLARAÇÃO DE INTEGRIDADE

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração.

Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

Autoencoders lineares e autoencoders não lineares (ReLU)

Resumo

Este trabalho é dedicado ao estudo de *autoencoders* lineares, onde se destacam as suas ligações com a técnica *PCA* e com *autoencoders* não lineares, nomeadamente, usando a função de ativação ReLU. Ao longo desta dissertação, são demonstrados diversos resultados sobre esta temática, através de diversas simplificações e hipóteses adicionais. É ainda elaborada uma análise numérica que visa corroborar os resultados abordados ao longo do documento.

Como principais destaques deste trabalho, pode-se enunciar o facto de que, para diversas bases de dados, é possível calcular uma solução ótima, ou seja, uma solução que atinge o valor mínimo que a *loss function* associada ao *autoencoder* consegue apresentar. Em particular, consideramos cenários com bases de dados de atributos não negativos, bem como a situação em que se assume que a base de dados é regular. É ainda de salientar a criação de novas propostas de algoritmos, em particular no contexto de *autoencoders* ReLU, que proporcionam muito boas aproximações das soluções ótimas com um baixo custo computacional em comparação aos tradicionais métodos de treino dos *autoencoders* com recurso às principais bibliotecas de *Python*.

Palavras-Chave: *Machine Learning*, *PCA*, *Autoencoder* linear, *Autoencoder* ReLU

Linear autoencoders and nonlinear autoencoders (ReLU)

Abstract

This work is dedicated to the study of linear autoencoders, standing out its connections both with the PCA technique and the nonlinear autoenconders, namely, using the ReLU function. Throughout this dissertation, various results on this topic are demonstrated through various simplifications and additional hypotheses. A numerical analysis is also elaborated to support the results stated throughout the document.

As main highlights of this work, it can be stated that, for several databases, we manage to calculate an optimal solution that reaches the minimum value which can be presented by the loss function associated with the autoenconder. In particular, we considered scenarios based on nonnegative data, or situations in which it is assumed that the database is regular. We also stress that the study proposes new algorithms, in particular in the context of autoencoders ReLU, that provide very good approximations of the optimal solution with a low computational cost regarded to the traditional autoencoder training method proposed in the standard Python libraries.

Keywords: *Machine Learning*, *PCA*, *Linear autoencoder*, ReLU *autoencoder*

Conteúdo

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

1.1 *Machine Learning*

Machine Learning é a ciência (e arte) de programar computadores de tal modo que estes possam aprender através de dados [1]. Os dados podem vir da natureza (recolhidos, por exemplo, através de sensores), podem ser gerados manualmente por humanos ou podem até ser gerados por outro algoritmo [2].

Os algoritmos em *Machine Learning* podem ser classificados em diversos tipos. Destacamos três dos principais tipos:

- Algoritmos supervisionados: Neste tipo de algoritmos existe uma *label* que classifica o *input*, por exemplo, num sistema para detetar se um doente tem uma doença, além de existir o *input* com todos os atributos correspondentes a um doente, há ainda uma *label* que menciona se o doente tem ou não a doença em causa.

- Algoritmos não supervisionados: Neste tipo de algoritmos, ao contrário do tipo anterior, não está presente a *label* que classifica se o doente tem ou não a doença.

- Algoritmos semi-supervisionados: Em certos problemas, a atribuição de *labels* é um processo bastante custoso e, por isso, existem, muitas vezes, problemas nos quais apenas alguns *inputs* se apresentam classificados com *labels*.

Os algoritmos estudados nesta dissertação inserem-se na classe dos algoritmos não supervisionados. Nesta classe de algoritmos não supervisionados, há, ainda assim, uma grande variedade de algoritmos, e o nosso estudo focar-se-á numa família mais específica de algoritmos, designados *autoencoders*. Os *autoencoders* são um tipo de rede neuronal

artificial que funciona de uma maneira não supervisionada [3]. Os *autoencoders* têm várias aplicações, como por exemplo, deteção de anomalias [4] [5], onde a ideia passa por, durante o processo de treino do *autoencoder*, este aprender a reproduzir com precisão as características mais frequentes nas observações. Assim, quando surgir uma anomalia este deve piorar o seu desempenho na reconstrução de *inputs* com anomalias. Resumindo, o *autoencoder* irá reconstruir os dados normais de uma maneira bastante satisfatória e quando surgir uma anomalia irá reconstruir de uma maneira má. [5]. Outra vertente onde os *autoencoders* aparecem é em problemas de redução de dimensão que foi uma das primeiras motivações para estudar *autoencoders*. Em suma, o objetivo é encontrar um método de projeção adequado, que mapeia dados de um espaço de alta dimensão para um espaço de baixa dimensão [6].

1.2 Objetivos do estudo e principais contribuições

Os *autoencoders* podem ser vistos como uma extensão não-linear do método *PCA*. A finalidade do *PCA*, é encontrar um meio de reduzir a informação contida em várias variáveis num conjunto menor de variáveis, onde o objetivo é a perda mínima de informação [7]. Com efeito, na situação particular de uma função de ativação linear, podemos demonstrar que as duas arquiteturas vão produzir uma decomposição em espaços próprios. No entanto, quando introduzimos uma função de ativação não linear, por exemplo, a função ReLU, que é linear na parte positiva e nula na parte negativa, o *autoencoder* diferencia-se completamente do caso linear *PCA*.

O objetivo deste trabalho é, por um lado, estudar relações entre os autoencoders lineares e o método PCA e, por outro, investigar até que ponto estas relações são preservadas quando são considerados autoencoders não lineares, em particular, com função de ativação ReLU. Com este trabalho, pretende-se identificar e compreender os fundamentos matemáticos que estão na base destas relações e, ao mesmo tempo, verificar numericamente estas relações, através de exemplos sintéticos, perfeitamente controlados.

Este documento apresenta várias contribuições no que diz respeito ao estudo de diversas propriedades matemáticas dos *autoencoders*. Ao longo do documento são apresentadas várias proposições e teoremas com o objetivo de evidenciar os principais resultados encontrados. Este documento apresenta ainda diversas contribuições na vertente da verificação numérica, sendo de destacar a este respeito um método novo para minimização de uma *loss function* no contexto de *autoencoders* com função de ativação ReLU, capaz de boas

aproximações da tradicional *loss function* para este tipo de *autoencoders*, mas com um custo computacional bastante inferior.

1.3 Estrutura da dissertação

Além deste capítulo, o documento contém mais seis capítulos.

O segundo capítulo é dedicado a vários conceitos e resultados de álgebra linear, considerados essenciais para a realização desta dissertação, entre eles várias propriedades sobre decomposição espetral e o enunciado e explicação do método de Análise de componentes principais (PCA).

O terceiro capítulo inicia o estudo dos *autoencoders*, fazendo uma introdução aos mesmos, e começando o estudo do caso linear. São apresentadas várias propriedades ao longo das suas secções e ainda vários exemplos que satisfazem condições necessárias à minimização da usual *loss function* para este tipo de *autoencoders*.

No quarto capítulo é realizada uma verificação numérica das principais propriedades encontradas ao longo do terceiro capítulo, com o objetivo de as corroborar. Ainda neste capítulo é demonstrado como se geraram as bases de dados para testes, que são posteriormente usadas em outros capítulos, com o objetivo de realizar também uma validação prática das propriedades.

No quinto capítulo é iniciado o estudo de *autoencoders* não lineares, começando-se com apresentação dos princípios gerais na base destes *autoencoders*. De seguida é realizado um estudo de várias funções de ativação passíveis de serem usadas com este tipo de *autoencoders*. O capítulo termina com uma introdução aos regularizadores, sendo explicados os conceitos gerais, a par da sua utilidade e dos problemas que pretendem resolver.

O sexto capítulo é dedicado ao estudo aprofundado de *autoencoders* ReLU, sendo este estudo semelhante ao efetuado no terceiro capítulo para *autoencoders* lineares. Em particular, é estudada a função de minimização, assim como o valor mínimo que esta poderá obter. Ao longo deste capítulo é ainda realizada uma análise prática dos resultados obtidos e são resolvidos alguns problemas encontrados durante o processo de treino. Este capítulo apresenta o novo método desenvolvido nesta dissertação para minimização de uma *loss function* para *autoencoders* ReLU, que se caracteriza por ter um custo computacional reduzido.

Por fim, o último capítulo é dedicado à conclusão do documento e nele são ainda apresentadas algumas sugestões de trabalho futuro.

Toda a parte prática desta dissertação foi desenvolvida com recurso à linguagem de programação *Python*. O código desenvolvido é apresentado no anexo desta dissertação.

Capítulo 2

Preliminares de Álgebra Linear

Nesta dissertação assumiremos por norma uma *base de dados* X com N *leituras* $x^1, ..., x^N$, cada qual com I *atributos*, ou seja, para cada leitura x^n tem-se

$$x^n = \begin{pmatrix} x_1^n \\ \vdots \\ x_I^n \end{pmatrix},$$

onde x_i^n indica o que se obtém no atributo i desta leitura x^n. Assim, é imediato observar que a base de dados X corresponde a uma matriz, designada *matriz dos dados*, dada do seguinte modo:

$$X = \begin{bmatrix} x^{1^T} \\ \vdots \\ x^{N^T} \end{bmatrix} = \begin{bmatrix} x_1^1 & x_2^1 & \dots & x_I^1 \\ \vdots & \vdots & & \vdots \\ x_1^N & x_2^N & \dots & x_I^N \end{bmatrix}$$

Ao longo das duas secções que constituem este capítulo iremos recordar alguns conceitos e propriedades fundamentais sobre decomposição espectral de matrizes e sobre o método de análise de componentes principais, que serão necessários ao estudo sobre *autoencoders* desenvolvido ao longo desta dissertação.

2.1 Decomposição Espectral de Matrizes

A generalidade dos conceitos e resultados mencionados nesta secção podem ser consultados em [8] e [9]. É de salientar, que ao longo desta dissertação, iremos considerar apenas matrizes sobre o conjunto $\mathbb{R}$.

Definição 2.1.1. *Seja $A_{n\times n}$ uma matriz quadrada, define-se o polinómio característico de A como $p_A(\lambda) = \left|\lambda Id - A\right|$, onde Id denota a matriz identidade de ordem n e $\left|\lambda Id - A\right|$ denota o determinante da matriz $\lambda Id - A$. As raízes de $p_A(\lambda)$ chamam-se os valores próprios de A. O conjunto dos valores próprios de A é também designado por espectro de A e é denotado por $\sigma(A)$.*

Definição 2.1.2. *Seja λ um valor próprio de uma matriz $A_{n\times n}$. Um vetor v n-dimensional (i.e. uma matriz coluna $n \times 1$) diz-se um vetor próprio associado a λ quando existe $v \neq 0$ tal que $Av = \lambda v$. v diz-se vetor próprio de A associado a λ.*

Definição 2.1.3. *Uma matriz U quadrada diz-se ortogonal se $U^{-1} = U^T$, onde, como habitualmente, U^{-1} e U^T indicam, respetivamente, a matriz inversa e a matriz transposta de T.*

Proposição 2.1.1. *Se uma matriz U é ortogonal, então $U^TU = UU^T = Id$.*

Definição 2.1.4. *Uma matriz P quadrada diz-se idempotente se $P^2 = P$.*

Definição 2.1.5. *Uma matriz quadrada S diz-se uma matriz simétrica se $S^T = S$.*

Definição 2.1.6. *Denota-se de produto interno usual a aplicação de $\mathbb{R}^n \to \mathbb{N}$ definida por:*

$$(x^1, x^2, \ldots, x^n) \cdot (y^1, y^2, \ldots, y^n) = x^1y^1 + x^2y^2 + \ldots + x^ny^n$$

com $(x^1, x^2, \ldots, x^n), (y^1, y^2, \ldots, y^n) \in \mathbb{R}^n$.

Definição 2.1.7. *Dois vetores $x, y \in \mathbb{R}^n$, dizem-se ortogonais se o produto interno entre eles for zero, ou seja, $x \cdot y = 0$.*

Definição 2.1.8. *Uma matriz quadrada D diz-se diagonalizável se é semelhante a uma matriz diagonal, isto é, D é diagonalizável se existe uma matriz invertível P tal que $P^{-1}DP$ é uma matriz diagonal.*

Proposição 2.1.2. *Uma matriz D de ordem $d \times d$ é diagonalizável se tiver d valores próprios distintos, ou seja, se o seu polinómio característico tiver d raízes distintas.*

Proposição 2.1.3. *Seja S simétrica:*

(1) $\sigma(S) \subseteq \mathbb{R}$*;*

(2) Os vetores próprios associados a valores próprios distintos são ortogonais dois a dois;

(3) S *é ortogonalmente diagonalizável, ou seja, existe uma matriz* U*, tal que,* $U^T = U^{-1}$ *e uma matriz* D *que é diagonal, tal que,* $S = UDU^T$*.*

Definição 2.1.9. *Uma matriz S simétrica diz-se semi-definida positiva (SDP) se $v \cdot Sv \geqslant 0$, para qualquer vetor v.*

Proposição 2.1.4. *Se S é SDP então $\sigma(S) \subseteq \mathbb{R}_0^+$.*

Proposição 2.1.5. *Seja A uma matriz $m \times n$:*

(1) $(AA^T)^T = AA^T$ *e* $(A^TA)^T = A^TA$*, logo* AA^T *e* A^TA *são simétricas;*

(2) AA^T *e* A^TA *têm os mesmos valores próprios não nulos;*

(3) AA^T *é SDP, logo* $\sigma(AA^T) \subseteq \mathbb{R}_0^+$ *e* $\sigma(A^TA) \subseteq \mathbb{R}_0^+$*.*

Exemplo 2.1.1. *Ilustramos, agora, uma aplicação importante da proposição anterior. Consideremos que A é uma matriz 1000×5. Então, AA^T é uma matriz 1000×1000. Assim, para calcular os valores próprios de AA^T, basta calcular os valores próprios de A^TA, que é uma matriz 5×5. Os restantes 995 valores próprios de AA^T serão nulos.*

Definição 2.1.10. *Seja A uma matriz quadrada o traço de A é a soma dos elementos da sua diagonal. Usamos a notação $tr(A)$ para denotar o traço da matriz.*

Proposição 2.1.6. *Dadas matrizes A, B e C, o traço apresenta as seguintes propriedades:*

(1) $tr(A + B) = tr(A) + tr(B)$*;*

(2) $tr(A) = tr(A^T)$*;*

(3) $tr(AB) = tr(BA)$*;*

(4) $tr(ABC) = tr(CAB) = tr(BCA)$*;*

(5) Se U é uma matriz ortogonal, $tr(UAU^T) = tr(A)$*;*

(6) Se P é uma matriz idempotente, $tr(PQP) = tr(P^2Q) = tr(PQ)$*.*

Definição 2.1.11. *A vetorização de uma matriz é uma transformação linear que converte uma matriz num vetor coluna. Mais concretamente a vetorização de uma matriz A $m \times n$, denotada por $vec(A)$, é o vetor coluna de dimensão $mn \times 1$ que resulta da escrita sucessiva das n colunas de A.*

Por exemplo, se $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ *então* $vec(A) = \begin{pmatrix} a \\ c \\ b \\ d \end{pmatrix}$.

O resultado seguinte pode ser consultado, por exemplo, em [10].

Proposição 2.1.7. *Dadas duas matrizes A e B, $tr(AB^T) = vec(A)^T vec(B)$.*

Definição 2.1.12. *Seja V um espaço vetorial sobre $\mathbb{K}$ com produto interno $v \cdot w$. Diz-se que dois vetores v e w de V são ortogonais, e escreve-se $v \perp w$, se $v \cdot w = 0$. Um sistema $(v_1, v_2, \cdots, v_r)$ de vectores de V diz-se um sistema ortogonal se, para todos $1 \leq i, j \leq r$*

$$i \neq j \implies v_i \cdot v_j = 0$$

Definição 2.1.13. *Sejam V um espaço vetorial sobre $\mathbb{K}$ com produto interno $(v, w) \to v \cdot w$ e $(v_1, v_2, \cdots, v_r)$ uma base de V . Então, diz-se que $(v_1, v_2, \cdots, v_r)$ é uma base ortogonal se o sistema $(v_1, v_2, \cdots, v_r)$ é um sistema ortogonal.*

Definição 2.1.14. *Seja V um espaço euclideano. Uma base B de V diz-se ortonormada se é ortogonal e se todos os vetores que a constituem têm norma 1.*

De seguida, consideraremos o *processo de ortogonalização de Gram-Schmidt*, que nos permitirá determinar uma base ortogonal de um espaço vetorial munido com produto interno, partindo de uma qualquer base conhecida.

Teorema 2.1.1. *Sejam V um espaço euclidiano ou unitário e $(u_1, u_2, ..., u_n)$ uma base de V. Então, o sistema $(v_1, v_2, ..., v_n)$, onde :*

$$
\begin{aligned}
v_1 &= u_1;\\
v_2 &= u_2 - \frac{u_2 \cdot v}{v_1 \cdot v_1} v_1;\\
v_3 &= u_3 - \frac{u_3 \cdot v_2}{v_2 \cdot v_2} v_2 - \frac{u_3 \cdot v_1}{v_1 \cdot v_1} v_1;\\
&\vdots\\
v_n &= u_n - \frac{u_n \cdot v_{n-1}}{v_{n-1} \cdot v_{n-1}} v_{n-1} - \ldots - \frac{u_n \cdot v_2}{v_2 \cdot v_2} v_2 - \frac{u_n \cdot v_1}{v_1 \cdot v_1} v_1.
\end{aligned}
$$

é uma base ortogonal de V.

Exemplo 2.1.2. *Seja $V = \mathbb{R}^3$ o espaço euclideano munido do produto interno*

$$(x, y, z) \cdot (a, b, c) = 2xa + xb + ya + 2yb - yc - zb + 4zc.$$

Então, a base canónica $B = ((1,0,0),(0,1,0),(0,0,1))$ não é uma base ortogonal de V, pois, por exemplo, $(1,0,0) \cdot (0,1,0) = 1$.

No entanto, o processo de ortogonalização de Gram-Schmidt permite-nos determinar uma base ortogonal em função da base canónica. De facto, os vetores $(v1, v2, v3)$, definidos de seguida, constituem uma base ortogonal de V:

$$v_1 = (1,0,0)$$

$$v_2 = (0,1,0) - \frac{(0,1,0)\cdot(1,0,0)}{(1,0,0)\cdot(1,0,0)}(1,0,0) = (0,1,0) - \frac{1}{2}(1,0,0) = -(\frac{1}{2}, 1, 0)$$

$$
\begin{aligned}
v_3 &= (0,0,1) - \frac{(0,0,1)\cdot(\frac{-1}{2},1,0)}{(\frac{-1}{2},1,0)\cdot(\frac{-1}{2},1,0)}\left(\frac{-1}{2},1,0\right) - \frac{(0,0,1)\cdot(1,0,0)}{(1,0,0)\cdot(1,0,0)}(1,0,0)\\
&= (0,0,1) - \frac{-1}{\frac{1}{2} - \frac{1}{2} - \frac{1}{2} + 2}\left(\frac{-1}{2},1,0\right) - 0(1,0,0)\\
&= (0,0,1) + \frac{2}{3}\left(\frac{-1}{2},1,0\right) = \left(\frac{-1}{3},\frac{2}{3},1\right)
\end{aligned}
$$

Definição 2.1.15. *Sejam $q_1, q_2, \ldots, q_j$ um conjunto de vetores ortogonais. Uma base de dados é chamada de base de dados regular se todas as suas componentes são geradas por*

esses vetores, ou seja, cada elemento da base de dados é da forma

$$x = \alpha_1 q_1 + \alpha_2 q_2 + \ldots + \alpha_j q_j$$

onde, $\alpha_1, \alpha_2, \ldots, \alpha_j \in \mathbb{R}$.

2.2 Análise de componentes principais (PCA)

Como referido anteriormente, um dos objetivos desta dissertação é relacionar *autoencoders* com o método de análise de componentes principais. Nesta secção fazemos uma rápida revisão deste método. Mais detalhes acerca deste esquema podem ser encontrados, por exemplo, em [11].

A análise de componentes principais é um método usado habitualmente na análise de dados multivariados, onde cada um dos atributos dos dados corresponde a uma variável aleatória. O PCA é uma transformação linear que converte a matriz dos dados para um novo sistema de coordenadas em que a primeira coordenada tem a maior variância possível (chamada primeira componente), a segunda maior variância fica ao longo da segunda coordenada, e assim por diante. Como consequência de tal transformação, muitas vezes, um subconjunto pequeno de componentes concentra a quase totalidade da variância dos dados (designadas as componentes principais) e pode ser utilizado para análises subsequentes dos dados, funcionando como um método de redução de dimensão. Normalmente, as componentes principais de um conjunto de dados são encontradas através da decomposição em valores singulares da matriz dos dados.

Tal como no início deste capítulo, consideremos que temos uma base de dados X com N leituras $x^1, ..., x^N$, cada qual com I atributos, ou seja,

$$X = \begin{bmatrix} x^{1^T} \\ \vdots \\ x^{N^T} \end{bmatrix} = \begin{bmatrix} x_1^1 & x_2^1 & \ldots & x_I^1 \\ \vdots & \vdots & & \vdots \\ x_1^N & x_2^N & \ldots & x_I^N \end{bmatrix}$$

É habitual centrar os dados para a *média* $\mu = \frac{1}{N}(x^1 + ... + x^N)$ [12]. Denotemos por B

a *matriz dos dados centrados* que é dada por:

$$B = \begin{bmatrix} \vdots & \vdots & \cdots & \vdots \\ x^1 - \mu & x^2 - \mu & \cdots & x^N - \mu \\ \vdots & \vdots & \cdots & \vdots \end{bmatrix}$$

A *matriz de covariância dos dados* é então dada por $Cov_X = \frac{1}{N} B^T B$, onde as entradas de Cov_X correspondem à covariância entre as variáveis aleatórias associadas a cada um dos atributos dos dados.

2.2.1 Minimização dos resíduos

O espaço vetorial gerado por w vai ser denotado por $< w >$.

Supondo que os dados estão centrados na média. Para cada x^n vai ser calculada a $proj_{<w>} x^n$, tal que a soma das distâncias de x^n a $< w >$ seja mínima. É de notar que $proj_{<w>} x^n = (x^n \cdot w)w$, com $\|w\| = 1$.

Será, então, necessário saber quanto vale w por forma a minimizar as distâncias.

Ora, para cada x^n, tem-se a distância ao quadrado,

$$\|x^n - proj_{<w>} x^n\|^2 = \|x^n - (x^n \cdot w)w\|^2 =$$
$$\|x^n\|^2 - 2(w \cdot x^n)(w \cdot x^n) + \|w\|^2 = \|x^n\|^2 - 2(w \cdot x^n)^2 + 1$$

A soma dos resíduos será,

$$Res(w) = \sum_{n=1}^{N} (\|x^n\|^2 - 2(w \cdot x^n)^2 + 1) = (N + \sum_{n=1}^{N} \|x^n\|^2) - 2 \sum_{n=1}^{N} (w \cdot x^n)^2$$

Ora, como $N + \sum_{n=1}^{N} \|x^n\|^2$ não depende de w, para minimizar $Res(w)$ tem-se de maximizar $\sum_{n=1}^{N} (w \cdot x^n)^2$, que é o mesmo que maximizar $\frac{1}{N} \sum_{n=1}^{N} (w \cdot x^n)^2$, i.e. , a média de $(w \cdot x^n)^2$

Seja,

$$X = \begin{bmatrix} x^{1^T} \\ \vdots \\ x^{N^T} \end{bmatrix} = \begin{bmatrix} x_1^1 & x_2^1 & \dots & x_I^1 \\ \vdots & \vdots & & \vdots \\ x_1^N & x_2^N & \dots & x_I^N \end{bmatrix}$$

Então, $Xw = \begin{bmatrix} x^1 \cdot w \\ x^2 \cdot w \\ \vdots \\ x^N \cdot w \end{bmatrix}$ e $(Xw)^T = \begin{bmatrix} w \cdot x^1 & w \cdot x^2 & \cdots & w \cdot x^N \end{bmatrix}$

$$\frac{1}{N}\sum_{n=1}^{N}(w \cdot x^n)^2 = \frac{1}{N}\begin{bmatrix} w \cdot x^1 & w \cdot x^2 & \cdots & w \cdot x^N \end{bmatrix}\begin{bmatrix} x^1 \cdot w \\ x^2 \cdot w \\ \vdots \\ x^N \cdot w \end{bmatrix} =$$

$$= \frac{1}{N}(Xw)^T(Xw) = \frac{1}{N}w^T(X^TX)w = w^TVw, \text{ com } V = \frac{1}{N}X^TX$$

Será, então, necessário maximizar w^TVw, considerando que $\|w\|^2 = 1$.

Sendo $u = w^TVw - \lambda(\|w\|^2 - 1)$ e aplicando multiplicadores de Lagrange, omitindo as contas, é fácil concluir que

$$\frac{\partial u}{\partial w} = 2Vw - 2\lambda w = 0 \implies Vw = \lambda w$$

.

Então, pode-se concluir que: w é vetor próprio de V associado ao valor próprio λ.

Consegue-se então inferir que maximizar a função pretendida é o mesmo que escolher os maiores valores próprios. Logo, a primeira componente principal (vetor próprio associado ao maior valor próprio) indica a direção da maior variância e assim por diante.

Para realizar o estudo da grandeza dos valores próprios em relação aos restantes, por outras palavras, para determinar o número de vetores próprios a utilizar, vamos analisar a razão entre os maiores valores próprios em relação à soma de todos os valores próprios (que iguala o traço).

Supondo que a razão é r, diz-se que cerca de $(r \times 100)\%$ da informação está "contida"nas direções dos vetores próprios associados aos maiores valores próprios.

Na maior parte dos casos, o número de valores próprios a usar vai ser determinado por duas abordagens. Uma delas será através da regra do cotovelo, que, de uma maneira geral, o que este método vai fazer é perceber quando é que a função está a caminhar para a solução mínima antes de a atingir, e, assim, nos dizer o número ideal de vetores próprios a utilizar. Outra abordagem será procurar a quantidade de informação que se quer manter, ou seja, encontrar o valor r mencionado acima consoante as necessidades do problema.

De seguida, vamos ilustrar, com um exemplo, ambas as técnicas de maneira a ficar mais claro este conceito.

Analisando a figura 2.1, pode-se concluir que, ao aplicar a regra do cotovelo, se vai utilizar o ponto verde como referência para o número de vetores próprios a utilizar, que corresponde a 21 vetores próprios num total de 400. Por outro lado, se a abordagem for, por exemplo, conter 90% da informação, vai ser usado o ponto amarelo como referência, que vai fazer uso de 109 vetores próprios.

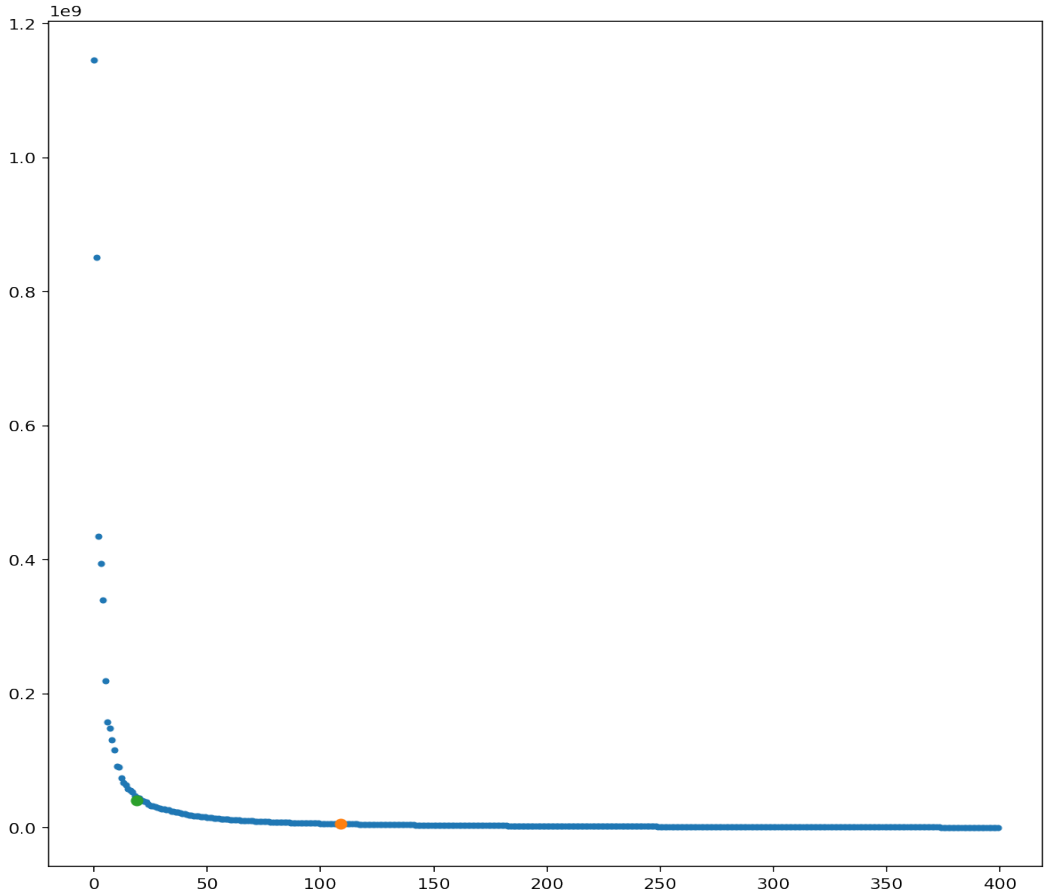


Figura 2.1: Exemplo da técnica para escolher o número de vetores próprios a usar

Capítulo 3

Autoencoders Lineares

Iniciaremos neste capítulo o nosso estudo sobre *autoencoders*. Estes algoritmos de machine learning correspondem a certas redes neuronais, treinadas de maneira não supervisionada, cujo objetivo central é reconstruir o *input* fornecido [13].

3.1 Definição do *autoencoder* linear

Os *autoencoders* podem ser divididos em duas peças fulcrais, um *Encoder* e um *Decoder*, como se pode observar na figura 3.1. Vamos, inicialmente, considerar *autoencoders* lineares apenas com uma camada oculta, pois um *autoencoder* linear com várias camadas ocultas pode ser visto como uma combinação de um *autoeconder* com apenas uma camada oculta, devido ao produto de matrizes.

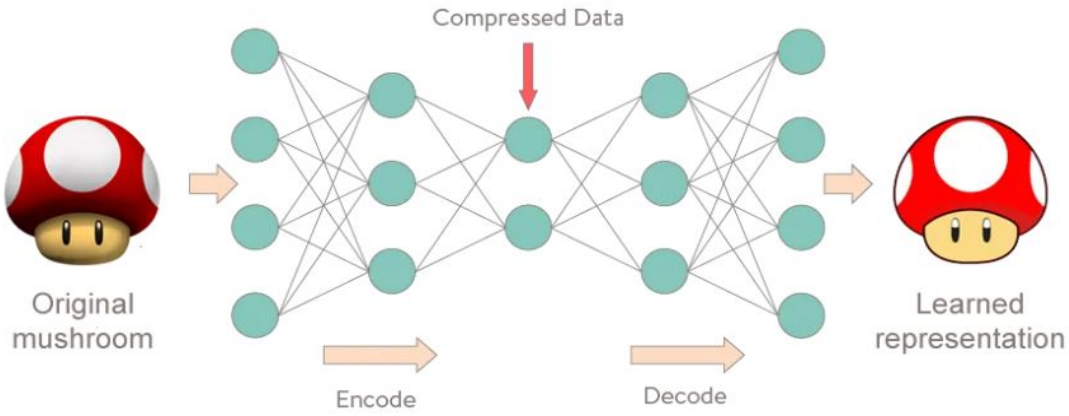


Figura 3.1: Estrutura de um *autoencoder* [14]

Seja X uma base de dados que será representada numa matriz e, sem perda de generalidade, considere-se essa matriz com dimensões $N \times I$, ou seja, a matriz tem N leituras (*inputs*) e cada um desses *inputs* tem I atributos.

Seja x um *input* arbitrário do *dataset*, $x = \begin{pmatrix} x_1 \\ \vdots \\ x_I \end{pmatrix}$, a matriz X, tal como no capítulo 2.2, será representada da seguinte forma :

$$X = \begin{bmatrix} x^{1^T} \\ \vdots \\ x^{N^T} \end{bmatrix} = \begin{bmatrix} x_1^1 & x_2^1 & \dots & x_I^1 \\ \vdots & \vdots & & \vdots \\ x_1^N & x_2^N & \dots & x_I^N \end{bmatrix}$$

Sejam $V \in \mathbb{R}^{I \times J}$, $U \in \mathbb{R}^{J \times I}$ matrizes de pesos e $a \in \mathbb{R}^{J \times 1}$, $b \in \mathbb{R}^{I \times 1}$ vetores *bias*.

O *Encoder* é uma função que vai mapear o *input* recebido $x \in \mathbb{R}^{I \times 1}$, em $y \in \mathbb{R}^{J \times 1}$, da seguinte forma $y = Ux + a$.

O *Decoder*, por outro lado, vai ser uma função que vai mapear o vetor codificado no formato original, ou seja, vai transformar um *input* $y \in \mathbb{R}^{J \times 1}$ em $z \in \mathbb{R}^{I \times 1}$, da seguinte maneira, $z = Vy + b$.

Recapitulando, o *autoencoder* será, então, constituído por a composta das duas funções [14]:

(1) ***Encoder:*** $y = Ux + a$;

(2) ***Decoder:*** $z = Vy + b$, que resulta em $z = V(Ux + a) + b$.

Treinar um *autoencoder* linear consiste em encontrar os parâmetros V, U, a, b que vão minimizar uma *loss function* que, de certa maneira, vai procurar minimizar a diferença entre o *input* x e a reconstrução feita do mesmo, que foi denotada por z. Normalmente, opta-se por usar a função *mean squared error* [15], ou seja:

$$F(x; U, V, a, b) = \|x - z\|_2^2 = \|x - (Vy + b)\|_2^2 = \|x - (V(Ux + a) + b)\|_2^2 = \\ = \|x - VUx - (b + Va))\|_2^2$$

Note-se que se a', b' são tais que $Va' = -b'$, então, tem-se que :

$$F(x; U, V, a, b) = F(x; U, V, a + a', b + b')$$

E portanto, tomando $a' = -a$, então, $b' = Va$. Logo,

$$F(x; U, V, a - a, b + Va) = F(x; U, V, c) = \|x - VUx + c)\|^2$$

$$\text{Sendo, } c = -b - Va$$

É fácil observar que a função que se pretende minimizar depende de V, U, c e de x.

Treinar um *autoencoder* vai passar por minimizar esta função para qualquer valor de x presente no *dataset* X.

Vai-se denotar essa função por E, ou seja :

$$E(X; U, V, c) = \sum_{n=1}^{N} F(x^n; U, V, c) = \sum_{n=1}^{N} \|x^n - VUx^n + c\|^2$$

3.2 Derivada da função erro

Nesta secção começaremos por introduzir alguns resultados sobre cálculo matricial. Estes resultados podem ser vistos em [16] e [17].

Lema 3.2.1. *Sejam X e Z matrizes, y e w dois vetores. (Sem perda de generalidade, vamos assumir que as várias matrizes referidas de seguida estão bem definidas.) Então:*

(1) $\frac{\partial y}{\partial w^T} = \frac{\partial y^T}{\partial w}$

(2) $\frac{\partial y}{\partial X^T} = \frac{\partial y}{\partial X}$

(3) $\frac{\partial y^T}{\partial w} = \frac{\partial w^T y}{\partial w} = y^T$

(4) $\frac{\partial y^T y}{\partial y} = 2y^T$

(5) $\frac{\partial y^T X}{\partial y} = X^T$

(6) $\frac{\partial y^T X y}{\partial y} = y^T(X + X^T)$

(7) $\frac{\partial y^T X^T w}{\partial X} = wy^T$

(8) $\frac{\partial y^T X^T X w}{\partial X} = X(yw^T + wy^T)$

(9) $\frac{\partial y^T X^T Z X w}{\partial X} = Z^T X y w^T + Z X w y^T$

De seguida, vamos considerar de novo a função $E(X; U, V, c)$, tendo em vista a análise da sua derivada.

Como se pode observar, o único parâmetro que está fixo na função E é X, que corresponde ao *dataset* obtido; tudo o resto, U, V, c, são variáveis a encontrar de maneira a minimizar a função E.

Note-se que a função $F(x; U, V, c)$ usada em E pode ser escrita da seguinte forma :

$$\begin{gathered} F(x;U,V,c) = \|x - VUx + c\|^2 = (x - VUx + c)^T (x - VUx + c) = \\ = (x^T - (VUx)^T + c^T)(x - VUx + c) = \\ = (x^T - x^T U^T V^T + c^T)(x - VUx + c) = \\ x^T x - x^T VUx + x^T c - x^T U^T V^T x + \\ x^T U^T V^T VUx - x^T U^T V^T c + c^T x - c^T VUx + c^T c \end{gathered}$$

Ao longo das seguintes proposições vamos derivar $E(X; U, V, c)$ em ordem a c, U e V.

Proposição 3.2.1. *Calculando as derivadas de E em ordem a c, obtém-se:*

$$\begin{gathered} \nabla_c E(X;U,V,c) = \sum_{n=1}^{N} \nabla_c F(x^n;U,V,c) = \\ = \sum_{n=1}^{N} \nabla_c (x^{n^T} c - x^{n^T} U^T V^T c + c^T x^n - c^T VUx^n + c^T c) = \\ = \sum_{n=1}^{N} x^{n^T} - x^{n^T} U^T V^T + x^{n^T} - x^{n^T} U^T V^T + 2c^T = \\ = \sum_{n=1}^{N} 2x^{n^T} - 2x^{n^T} U^T V^T + 2c^T \end{gathered}$$

Igualando a expressão a zero,

$$\nabla_c E(X;U,V,c) = 0 \iff \sum_{n=1}^{N} 2x^{n^T} - 2x^{n^T}U^TV^T + 2c^T = 0 \iff$$

$$\iff \sum_{n=1}^{N} x^{n^T}U^TV^T - x^{n^T} = c^T \iff \sum_{n=1}^{N} VUx^n - x^n = c \iff$$

$$\iff (VU - I)\sum_{n=1}^{N} x^n = c$$

Fazendo uso da proposição 3.2.1 podem-se estabelecer os seguintes corolários :

Corolário 3.2.1.

$$(VU - I)\sum_{n=1}^{N} x^n = c$$

Corolário 3.2.2. *Se a base de dados X é centrada, ou seja, $\sum_{n=1}^{N} x^n = 0$, então, $c = 0$.*

Nota 3.2.1. $\sum_{n=1}^{N} x^n x^{n^T} = X^TX$

Fazendo uso do corolário 3.2.2, pode-se fazer um estudo de como se comporta a função $F(x;U,V)$, caso a base de dados seja centrada.

Proposição 3.2.2. *Se a base de dados for centrada, ou seja, $c = 0$.*

$$F(x;U,V) = \|(I - VU)x\|^2 =$$
$$= ((I - VU)x)^T((I - VU)x) =$$
$$= (x - VUx)^T(x - VUx) = (x^T - x^TU^TV^T)(x - VUx) =$$
$$= x^Tx - x^TVUx - x^TU^TV^Tx + x^TU^TV^TVUx$$

Então:

$$E(X;U,V) = \sum_{n=1}^{N} F(x;U,V) =$$
$$= \sum_{n=1}^{N} x^{n^T}x^n - x^{n^T}VUx^n - x^{n^T}U^TV^Tx^n + x^{n^T}U^TV^TVUx^n$$

De seguida, vamos calcular a derivada de $E(X;U,V,c)$ para o caso de X ser uma base de dados centrada e para o caso de X não ser centrada.

Derivando $E(X;U,V,c)$ em ordem a U, obtém-se a seguinte proposição.

Proposição 3.2.3.

(1) ***Caso geral:***

$$\nabla_U E(X;U,V,c) = \sum_{n=1}^{N} \nabla_U F(x^n;U,V,c) =$$

$$\sum_{n=1}^{N}(-x^TVUx - x^TU^TV^Tx + x^TU^TV^TVUx - x^TU^TV^Tc - c^TVUx) =$$

$$= \sum_{n=1}^{N} -2V^Txx^T + 2V^TVUxx^T - 2V^Tcx^T$$

(2) ***Caso c=0:***

$$\nabla_U E(X;U,V) = -V^TX^TX - V^TX^TX + V^TVUX^TX + V^TVUX^TX =$$
$$= -2V^TX^TX + 2V^TVUX^TX$$

Igualando a expressão a zero,

$$\nabla_U E(X;U,V) = 0 \iff -2V^TX^TX + 2V^TVUX^TX = 0 \iff$$
$$\iff (V^T - V^TVU)(X^TX) = 0$$

No seguimento do que foi obtido na proposição 3.2.3, é, então, possível obter os seguintes resultados:

Proposição 3.2.4. *Se X^TX é invertível, então:*

(1) $V^T = V^TVU$.

(2) Além disso, se a característica de V for máxima, V^TV é invertível, e $U = (V^TV)^{-1}V^T$.

Dada a proposição acima, é possível obter o seguinte corolário.

Corolário 3.2.3. *Seja X um dataset. Se X^TX é invertível e a característica de V for máxima, então, consegue-se escrever U à custa de V, ou seja, $U = (V^TV)^{-1}V^T$.*

Derivando agora $E(X;U,V,c)$ em ordem a V, obtém-se:

Proposição 3.2.5.

(1) ***Caso geral:***

$$\nabla_V E(X;U,V,c) = \sum_{n=1}^{N} \nabla_V F(x^n;U,V,c) =$$

$$\sum_{n=1}^{N}(-x^TVUx - x^TU^TV^Tx + x^TU^TV^TVUx - x^TU^TV^Tc - c^TVUx) =$$

$$= \sum_{n=1}^{N} -2xx^TU^T + 2VUxx^TU - 2x^TU^TV^Tc$$

(2) ***Caso c=0:***

$$\nabla_V E(X;U,V) = -2X^TXU^T + 2VUX^TXU^T$$

Igualando a expressão a zero,

$$\nabla_V E(X;U,V) = 0 \iff -2X^TXU^T + 2VUX^TXU^T = 0 \iff$$
$$\iff (VU - Id)X^TXU^T = 0$$

Utilizando os resultados obtidos na proposição 3.2.5, pode-se dizer o seguinte:

Proposição 3.2.6. *Se $E(X;U,V)$ admite um mínimo, então:*

$$(VU - Id)X^TXU^T = 0$$
$$\iff$$
$$VUX^TXU^T = X^TXU^T$$

Dada a proposição acima, é possível obter o seguinte corolário.

Corolário 3.2.4. *Seja X um dataset. Se X^TX é invertível e U tiver característica máxima, então, consegue-se escrever V à custa de U, ou seja, $V = X^TXU^T(UX^TXU^T)^{-1}$*

3.2.1 Síntese dos resultados principais

Até aqui, o trabalho principal deste capítulo foi identificar condições necessárias para obter o mínimo para a função de erro do *autoencoder* linear. De seguida fazemos um resumo mais compacto das propriedades principais obtidas, até ao momento, neste trabalho, que vão ser utilizadas nos capítulos seguintes.

Os principais resultados encontrados foram:

(1) Se a base de dados for centrada:

   (a) $c = 0$.

   (b) $(V^T - V^T VU)(X^T X) = 0$

   (c) $(VU - Id_I)X^T XU^T = 0$

(2) Se a base de dados for centrada e a característica for máxima:

   (a) $V^T = V^T VU$

   (b) $VUX^T XU^T = X^T XU^T$

3.3 Exemplos que satisfazem condições necessárias à minimização do erro

No caso em que se está a considerar *autoencoders* lineares e centrados na média, já foi visto em cima que $c = 0$, portanto, a função de minimização pode escrever-se da seguinte forma simplificada:

$$E(X; U, V) = \sum_{n=1}^{N} F(x; U, V) = \sum_{n=1}^{N} \|(I - VU)(x^n)\|^2$$

É de notar que qualquer base de dados pode ser sempre centrada na média, tal como foi visto no capítulo 2.2. Por esta razão, no que se segue, assumiremos que X é uma base de dados centrada.

Encontrar uma solução que satisfaça as condições necessárias à minimização do erro anteriormente identificadas consiste em encontrar matrizes U e V que satisfaçam as seguintes

equações:

$$(V^T - V^T VU)(X^T X) = 0 \tag{3.1}$$

$$(VU - Id_I)X^T XU^T = 0 \tag{3.2}$$

Assim, o objetivo desta secção será determinar exemplos de matrizes U e V que satisfaçam as condições necessárias 3.1 e 3.2.

As seguintes sub-secções ilustram vários tipos de soluções distintas que foram encontradas ao longo deste estudo.

3.3.1 Solução baseada na decomposição espectral

Esta sub-secção terá como objetivo inicial encontrar matrizes U e V que satisfaçam as condições de minimização. Para tal, começaremos por considerar a sequinte proposição:

Proposição 3.3.1. *Seja U uma matriz constituída por J vetores próprios de $X^T X$, ou seja,*

$$U = \begin{bmatrix} \leftarrow & q_1 & \rightarrow \\ \leftarrow & q_2 & \rightarrow \\ & \vdots & \\ \leftarrow & q_J & \rightarrow \end{bmatrix} \text{ e } V = \begin{bmatrix} \uparrow & \uparrow & \dots & \uparrow \\ q_1 & q_2 & \dots & q_J \\ \downarrow & \downarrow & \dots & \downarrow \end{bmatrix}$$

isto é, $V = U^T$. Então, estas matrizes vão satisfazer as condições necessárias 3.1 e 3.2.

A seguinte nota será útil na demonstração da proposição 3.3.1.

Nota 3.3.1. *Uma matriz Q pode ser vista como sendo a soma de duas matrizes, isto é, se*

$$U = \begin{bmatrix} \leftarrow & q_1 & \rightarrow \\ \leftarrow & q_2 & \rightarrow \\ & \vdots & \\ \leftarrow & q_J & \rightarrow \end{bmatrix}, \tilde{U} = \begin{bmatrix} \leftarrow & q_{J+1} & \rightarrow \\ \leftarrow & q_{J+2} & \rightarrow \\ & \vdots & \\ \leftarrow & q_I & \rightarrow \end{bmatrix}$$

então,

$$Q = \begin{bmatrix} \leftarrow & q_1 & \rightarrow \\ \leftarrow & q_2 & \rightarrow \\ & \vdots & \\ \leftarrow & q_J & \rightarrow \\ & \vdots & \\ \leftarrow & q_I & \rightarrow \end{bmatrix} = \begin{bmatrix} U \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \tilde{U} \end{bmatrix} \text{ , } onde \begin{bmatrix} 0 \\ \tilde{U} \end{bmatrix} = \begin{bmatrix} 0 & \dots & 0 \\ \vdots & \vdots & \vdots \\ 0 & \dots & 0 \\ \leftarrow & q_{J+1} & \rightarrow \\ & \vdots & \\ \leftarrow & q_I & \rightarrow \end{bmatrix}$$

Demonstremos agora a proposição 3.3.1.

Demonstração. Como se pode observar, U é de dimensão $J \times I$, e, devido ao facto de U ser constituída por J vetores próprios, então, é fácil notar que $UU^T = Id_J$.

No entanto, U^TU não tem necessariamente de ser Id_I.

Por substituição direta, a condição 3.1 é satisfeita.

$$(V^T - V^TVU)(X^TX) = 0 \iff (U - UU^TU)(X^TX) = 0$$
$$\iff (U - Id_JU)(X^TX) \iff (U - U)(X^TX) = 0 \iff 0 = 0$$

Enquanto para satisfazer a primeira equação apenas foi necessário usar a propriedade $UU^T = Id_J$, para satisfazer a segunda equação já vai ser imposto o uso do facto de U ser composto por vetores próprios associados a valores próprios de X^TX.

Para tal, vai-se fazer uso da diagonalização de matrizes descrita em 2.1.8.

Seja, então, $X^TX = Q\Lambda Q^T$, onde $Q = \begin{bmatrix} \leftarrow & q_1 & \rightarrow \\ \leftarrow & q_2 & \rightarrow \\ & \vdots & \\ \leftarrow & q_J & \rightarrow \\ & \vdots & \\ \leftarrow & q_I & \rightarrow \end{bmatrix}$, $\Lambda = \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & \lambda_I \end{bmatrix}$

e onde q_i representa o vetor próprio associado ao valor próprio λ_i.

É de notar que $Q\Lambda Q^{-1} = Q\Lambda Q^T$, pois $Q^T = Q^{-1}$, pelo facto de Q ser uma matriz ortogonal.

Ou seja, Q é uma matriz $I \times I$, constituída pelos vetores próprios usados em U e ainda por mais $I - J$ vetores próprios associados a valores próprios de X^TX.

Fazendo uso da definição 3.3.1, vai-se, então, provar a segunda condição (3.2).

$$(VU - Id_I)X^T XU^T = (U^T U - Id_I)X^T XU^T =$$
$$(U^T U - Id_I)Q^T \Lambda Q U^T =$$
$$(U^T U - Id_I)Q^T \Lambda \left(\begin{bmatrix} U \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \tilde{U} \end{bmatrix} \right) U^T =$$
$$(U^T U - Id_I)Q^T \Lambda \left(\begin{bmatrix} Id_J \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right) =$$
$$(U^T U - Id_I)Q^T \left(\begin{bmatrix} \Lambda_J \\ 0 \end{bmatrix} \right) = (U^T U - Id_I) \left(\begin{bmatrix} U \\ \tilde{U} \end{bmatrix}^T \begin{bmatrix} \Lambda_J \\ 0 \end{bmatrix} \right) =$$
$$(U^T U - Id_I) \left(\begin{bmatrix} \Lambda_J & 0 \end{bmatrix} \begin{bmatrix} U \\ \tilde{U} \end{bmatrix} \right)^T = (U^T U - Id_I) \begin{bmatrix} \Lambda_J U & 0 \end{bmatrix}^T =$$
$$(U^T U - Id_I) \begin{bmatrix} (\Lambda_J U)^T \\ 0 \end{bmatrix} = \begin{bmatrix} U^T U U^T \Lambda_J^T \\ 0 \end{bmatrix} - \begin{bmatrix} U^T \Lambda_J^T \\ 0 \end{bmatrix} =$$
$$\begin{bmatrix} U^T Id_J \Lambda_J^T \\ 0 \end{bmatrix} - \begin{bmatrix} U^T \Lambda_J^T \\ 0 \end{bmatrix} = \begin{bmatrix} U^T \Lambda_J^T \\ 0 \end{bmatrix} - \begin{bmatrix} U^T \Lambda_J^T \\ 0 \end{bmatrix} = 0$$

□

Em conclusão, através da diagonalização de matrizes e centrando a base de dados X na média, é possível obter matrizes U e V, que satisfazem as condições necessárias à minimização do erro.

3.3.2 Soluções no mesmo espaço próprio

Nesta sub-secção, a ideia passa por provar que também é possível encontrar uma solução que não vai utilizar os vetores próprios associados a valores próprios de $X^T X$, mas sim combinações lineares destes vetores.

Proposição 3.3.2. *Sejam*

$$Z = \begin{bmatrix} \leftarrow & z_1 & \rightarrow \\ \leftarrow & z_2 & \rightarrow \\ & \vdots & \\ \leftarrow & z_J & \rightarrow \end{bmatrix} \tilde{Z} = \begin{bmatrix} \leftarrow & z_{J+1} & \rightarrow \\ \leftarrow & z_{J+2} & \rightarrow \\ & \vdots & \\ \leftarrow & z_I & \rightarrow \end{bmatrix}$$

tal que

$$Vect\{z_1, z_2, ..., z_J\} = Vect\{q_1, q_2, ..., q_J\}$$

e

$$Vect\{z_{J+1}, p_{J+2}, ..., p_I\} = Vect\{q_{J+1}, q_{J+2}, ..., q_I\}$$

ou seja, z_k pode ser escrito à custa de uma combinação linear de vetores do conjunto $Vect\{q_1, q_2, ..., q_J\}$, caso $k \in [1, J]$. E caso $k \in [J+1, I]$, z_k pode ser escrito à custa de uma combinação linear de vetores do conjunto $Vect\{q_{J+1}, q_{J+2}, ..., q_I\}$. Assim, estas matrizes também satisfazem as condições necessárias à minimização do erro 3.1 e 3.2.

Demonstração. É fácil notar que $\tilde{Z}Z^T = 0$, pelo simples facto de que qualquer produto interno $z_r.z_{r'} = 0$, pois $q_r.q_{r'} = 0$, onde $1 \leqslant r \leqslant J$ e $J+1 \leqslant r' \leqslant I$.

Como Z é gerado pela mesma base de vetores de U, então, $Z = PU$, onde P é chamada uma matriz de rotação, além disso P é necessariamente invertível. No entanto, vai-se apenas estudar o caso em que P é uma matriz ortogonal, ou seja, $P^{-1} = P^T$.

Dado isto, vamos demonstrar algumas propriedades que serão essenciais para provar que a nossa hipótese proposta é uma solução para o problema apresentado.

$$ZZ^T = PUU^TP^T = PId_JP^T = Id_J$$
$$\iff$$
$$ZZ^T = Id_J$$

Além disso,

$$UZ^T = U(PU)^T = UU^TP^T = Id_JP^T = P^T$$

e $\tilde{U}Z^T = 0$, pelo facto de $\tilde{U}$ e Z^T serem matrizes constituídas por vetores linearmente independentes e não terem nenhum desses vetores em comum.

Vai-se, assim, provar que continuamos a ter uma solução para a nossa função de mini-

mização, tomando, então, $U = Z$, e supondo que $V = Z^T$.

No que diz respeito à primeira condição, vamos ter que :

$$(V^T - V^T VU)(X^T X) = 0 \iff (Z - ZZ^T Z)(X^T X) = 0$$
$$\iff (Z - Id_J Z)(X^T X) \iff (Z - Z)(X^T X) = 0 \iff 0 = 0$$

Relativamente à segunda condição:

$$(VU - Id_I)X^T XU^T = (Z^T Z - Id_I)X^T XZ^T =$$
$$(Z^T Z - Id_I)Q^T \Lambda QZ^T =$$
$$(Z^T Z - Id_I)Q^T \Lambda \left(\begin{bmatrix} U \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \tilde{U} \end{bmatrix} \right) Z^T =$$
$$(Z^T Z - Id_I)Q^T \Lambda \left(\begin{bmatrix} P^T \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right) =$$
$$(Z^T Z - Id_I)Q^T \left(\begin{bmatrix} \Lambda_J P^T \\ 0 \end{bmatrix} \right) = (Z^T Z - Id_I) \left(\begin{bmatrix} U \\ \tilde{U} \end{bmatrix}^T \begin{bmatrix} \Lambda_J P^T \\ 0 \end{bmatrix} \right) =$$
$$(Z^T Z - Id_I) \left(\begin{bmatrix} \Lambda_J P^T & 0 \end{bmatrix} \begin{bmatrix} U \\ \tilde{U} \end{bmatrix} \right)^T = (Z^T Z - Id_I) \begin{bmatrix} \Lambda_J P^T U & 0 \end{bmatrix}^T =$$
$$(Z^T Z - Id_I) \begin{bmatrix} (\Lambda_J P^T U)^T \\ 0 \end{bmatrix} = \begin{bmatrix} Z^T ZU^T P\Lambda_J^T \\ 0 \end{bmatrix} - \begin{bmatrix} U^T P\Lambda_J^T \\ 0 \end{bmatrix} =$$
$$\begin{bmatrix} Z^T Z(P^T Z)^T P\Lambda_J^T \\ 0 \end{bmatrix} - \begin{bmatrix} U^T P\Lambda_J^T \\ 0 \end{bmatrix} =$$
$$\begin{bmatrix} Z^T ZZ^T PP\Lambda_J^T \\ 0 \end{bmatrix} - \begin{bmatrix} U^T P\Lambda_J^T \\ 0 \end{bmatrix} =$$
$$\begin{bmatrix} Z^T PP\Lambda_J^T \\ 0 \end{bmatrix} - \begin{bmatrix} U^T P\Lambda_J^T \\ 0 \end{bmatrix} =$$
$$\begin{bmatrix} (P^T Z)^T P\Lambda_J^T \\ 0 \end{bmatrix} - \begin{bmatrix} U^T P\Lambda_J^T \\ 0 \end{bmatrix} =$$
$$\begin{bmatrix} U^T P\Lambda_J^T \\ 0 \end{bmatrix} - \begin{bmatrix} U^T P\Lambda_J^T \\ 0 \end{bmatrix} = 0$$

□

Logo, pode-se concluir que também se consegue satisfazer as condições necessárias à minimização do erro em *autoencoders* lineares através de combinações lineares de vetores próprios.

3.4 Solução mínima do *autoencoder* linear

Ao longo das secções anteriores, foram estudadas condições necessárias à minimização do erro em *autoencoders* lineares e verificou-se que existe um grande número de pares U e V que satisfazem estas condições.

No entanto, vamos agora tentar identificar qual é a solução do problema de minimização. Previamente, as condições necessárias foram obtidas na base dos espaços próprios, ou seja, não havia nenhuma consideração pelos valores próprios que eram utilizados. Nesta secção exploraremos esta última vertente.

Relembrando que a nossa base de dados X tem dimensão $N \times I$, e fazendo uso da *Definição 2.1.11* e da *Proposição 2.1.7*, e, mais uma vez, tratando apenas o caso da base de dados $X^T X = Q \Lambda Q^T$ ser centrada, ou seja, $c = 0$ pode-se escrever a função de minimização de maneira diferente.

Vai-se assumir que os I valores próprios de Λ estão ordenados, ou seja,
$\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_I$ e seja $\mathcal{J} = \{r_1, r_2, \ldots, r_j\}$, onde $0 \leq r_j \leq I$.

Vamos definir a matriz $Q_{\mathcal{J}}$ como sendo constituída pelos vetores próprios associados aos valores próprios λ_{r_j}, correspondentes aos coeficientes de $\mathcal{J}$.

Estamos agora em condições de estabelecer o seguinte teorema:

Teorema 3.4.1. *Consideremos que U e V são matrizes tais que V tem característica máxima e $U = (V^T V)^{-1} V^T$. Consideremos ainda que existe uma matriz invertível C com dimensões $J \times J$ com característica J, tal que $V = Q_{\mathcal{J}} C$ e que as condições mencionadas nos dois parágrafos anteriores a este teorema são satisfeitas. Então:*

$$E(X; U, V) = tr(X^T X) - \sum_{r_i \in \mathcal{J}} \lambda_{r_i}$$

.

Demonstração. A demonstração deste teorema está organizada em três partes.

Primeira parte

Supondo que V tem característica máxima e que $U = (V^T V)^{-1} V^T$, e além disso se existir uma matriz invertível C com dimensões $J \times J$ com característica igual a J tal que $V = Q_{\mathcal{J}} C$, então, é fácil de notar que:

$$U = (V^T V)^{-1} V^T = (C^T Q_{\mathcal{J}}^T Q_{\mathcal{J}} C)^{-1} C^T Q_{\mathcal{J}}^T = C^{-1} C^{T^{-1}} C^T Q_{\mathcal{J}}^T = C^{-1} Q_{\mathcal{J}}^T$$

E, além disso, ambas as condições necessárias são satisfeitas, logo, U e V definem um ponto crítico de E.

Segunda parte

Pode-se escrever a função de minimização $E(X; U, V)$ da seguinte maneira :

$$E(X; U, V) = \sum \|x - VUx\|^2 = vec(X^T - VUX^T)^T vec(X^T - VUX^T) =$$
$$vec(X^T)^T vec(X^T) - 2vec(VUX^T)^T vec(X^T) + vec(VUX^T)^T vec(VUX^T) =$$
$$tr(X^T X) - 2tr(VUX^T X) + tr(VUX^T XU^T V^T)$$

Também é de notar que:

$$tr(VUX^T XU^T V^T) = tr(V(V^T V)^{-1} V^T X^T X((V^T V)^{-1} V^T)^T V^T)$$

Além disso, seja $W = V(V^T V)^{-1} V^T$, é fácil de notar que W é uma matriz de projeção, então, é uma matriz idempotente. Logo:

$$tr(VUX^T XU^T V^T) = tr(WX^T XW^T)$$

Recordando a proposição 2.1.6 6, sabe-se que:

$$tr(WX^T XW^T) = tr(WX^T X)$$

Logo, fazendo uso da proposição 2.1.6 5, conclui-se que:

$$tr(WX^TX) = tr(QQ^TW(QQ^T)^TX^TX) =$$
$$tr(QQ^TWQQ^TQ\Lambda Q^T) = tr(QQ^TWQ\Lambda Q^T) =$$
$$tr(Q^TWQ\Lambda)$$

Recapitulando,

$$E(X;U,V) = tr(X^TX) - 2tr(VUX^TX) + tr(VUX^TXU^TV^T) =$$
$$tr(X^TX) - 2tr(WX^TX) + tr(WX^TXW^T) =$$
$$tr(X^TX) - 2tr(WX^TX) + tr(WX^TX)$$
$$= tr(X^TX) - tr(WX^TX) = tr(X^TX) - tr(Q^TWQ\Lambda)$$

Terceira parte

No entanto, se $V = Q_{\mathcal{J}}C$, então:

$$Q^TWQ = Q^TV(V^TV)^{-1}V^TQ =$$
$$Q^T(Q_{\mathcal{J}}C)((Q_{\mathcal{J}}C)^T)Q_{\mathcal{J}}C)^{-1}(Q_{\mathcal{J}}C)^TQ =$$
$$Q^T(Q_{\mathcal{J}}C)(C^TQ_{\mathcal{J}}^TQ_{\mathcal{J}}C)^{-1}(Q_{\mathcal{J}}C)^TQ =$$
$$Q^T(Q_{\mathcal{J}}C)(C^TC)^{-1}C^TQ_{\mathcal{J}}^TQ =$$
$$Q^TQ_{\mathcal{J}}CC^{-1}C^{T^{-1}}C^TQ_{\mathcal{J}}^TQ =$$
$$Q^TQ_{\mathcal{J}}Q_{\mathcal{J}}^TQ =$$
$$Q^TQ_{\mathcal{J}}(Q^TQ_{\mathcal{J}})^T$$

No entanto, $Q^TQ_{\mathcal{J}}$ é uma matriz $I \times J$ que pode ser escrita como sendo

$$Q^TQ_{\mathcal{J}} = \begin{bmatrix} \leftarrow & q_1 & \rightarrow \\ \leftarrow & q_2 & \rightarrow \\ & \vdots & \\ \leftarrow & q_I & \rightarrow \end{bmatrix} \times \begin{bmatrix} \uparrow & \uparrow & \cdots & \uparrow \\ q_{r_1} & q_{r_2} & \cdots & q_{r_J} \\ \downarrow & \downarrow & \cdots & \downarrow \end{bmatrix} = \begin{bmatrix} z_{11} & z_{12} & \cdots & z_{1J} \\ z_{21} & z_{22} & \cdots & z_{2J} \\ & \vdots & & \\ z_{I1} & z_{I2} & \cdots & z_{IJ} \end{bmatrix}$$

$$z_{ij} = \begin{cases} 1, & q_i = q_{r_j} \\ 0, & q_i \neq q_{r_j} \end{cases}$$

Onde $i \in [0, I]$ e $r_j \in \mathcal{J}$. É ainda de notar que os vetores qr_j são vetores próprios associados a valores próprios de $X^T X$ assim como q_i.

No entanto $(Q^T Q_{\mathcal{J}})^T = \begin{bmatrix} z_{11} & z_{21} & \dots & z_{I1} \\ z_{12} & z_{22} & \dots & z_{I2} \\ & \vdots & & \\ z_{1J} & z_{2J} & \dots & z_{IJ} \end{bmatrix}$

Podemos então notar que: $\begin{bmatrix} z_{11} & z_{12} & \dots & z_{1J} \\ z_{21} & z_{22} & \dots & z_{2J} \\ & \vdots & & \\ z_{I1} & z_{I2} & \dots & z_{IJ} \end{bmatrix} \times \begin{bmatrix} z_{11} & z_{21} & \dots & z_{I1} \\ z_{12} & z_{22} & \dots & z_{I2} \\ & \vdots & & \\ z_{1J} & z_{2J} & \dots & z_{IJ} \end{bmatrix} = \begin{bmatrix} k_1 & 0 & \dots & 0 \\ 0 & k_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & k_I \end{bmatrix}$

$$k_i = \begin{cases} 1, & se \ \ r_i \in \mathcal{J} \\ 0, & se \ \ r_i \notin \mathcal{J} \end{cases}$$

Assim, dado que $Q^T W Q = \begin{bmatrix} k_1 & 0 & \dots & 0 \\ 0 & k_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & k_I \end{bmatrix}$, pode-se escrever a expressão $E(X; U, V)$ como :

$$E(X; U, V) = tr(X^T X) - tr(Q^T W Q \Lambda) =$$

$$= tr(X^T X) - tr(\begin{bmatrix} k_1 & 0 & \dots & 0 \\ 0 & k_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & k_I \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_I \end{bmatrix}) =$$

$$= tr(X^T X) - \sum_{r_i \in \mathcal{J}} \lambda_{r_i}$$

□

Do seguimento do teorema acabado de demonstrar, é, então, possível enunciar o seguinte

corolário:

Corolário 3.4.1. *O mínimo de* $E(X;U,V)$ *é atingido quando* $\mathcal{J} = \{1, 2, \ldots, j\}$.

Demonstração. É fácil de notar que minimizar $E(X;U,V)$ consiste em maximizar $\sum_{r_i \in \mathcal{J}} \lambda_{r_i}$. Como $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_I$, então, $\sum_{r_i \in \mathcal{J}} \lambda_{r_i}$ será maximizado quando considerarmos os J maiores valores próprios, ou seja, tomando $\mathcal{J} = \{1, 2, \ldots, j\}$.

$$\begin{aligned} E(X;U,V) &= tr(X^T X) - tr(Id_{\mathcal{J}}\Lambda) = \\ &= tr(X^T X) - \sum_{r_i \in \mathcal{J}} \lambda_{r_i} = \\ &= \sum_{i=1}^{I} \lambda_i - \sum_{r_i \in \mathcal{J}} \lambda_{r_i} = \\ &= \sum_{r_i \notin \mathcal{J}} \lambda_{r_i} \end{aligned}$$

□

Para concluir, de maneira a minimizar a função $E(X;U,V)$ o conjunto $\mathcal{J}$ tem de conter os maiores valores próprios de Λ.

Capítulo 4

Verificação numérica

Além do estudo dos fundamentos matemáticos dos *autoencoders*, o trabalho desenvolvido nesta dissertação compreendeu uma implementação de *autoencoders* em *python*, que, em particular, teve como objetivos melhorar a compreensão destes algoritmos na prática e a verificação numérica de diversos resultados estudados ao longo das secções do capítulo 3. Neste capítulo serão apresentados os aspetos principais desta vertente de implementação da dissertação, que inclui a análise e algumas conclusões acerca dos resultados obtidos.

4.1 Criação da Base de Dados

De maneira a testar se o nosso *autoencoder* se comporta como o esperado, inicialmente, será necessário criar bases de dados distintas, de forma a experimentar diversos cenários possíveis. Vão ser utilizadas bases de dados aleatoriamente geradas, mas também outras que serão criadas de modo específico para se poder verificar certas propriedades.

Antes de se passar a este processo, é preciso definir e recordar algumas noções.

As seguintes variáveis neste capítulo terão o seguinte significado :

- N : Será o tamanho da base de dados, ou seja, o número de linhas da mesma;
- I : Será o número de atributos que cada linha da base de dados tem, ou seja, o número de colunas;
- J : Será o número de atributos dos dados comprimidos, melhor dizendo, o tamanho dos dados depois de passarem pelo processo de *encode*.

De modo a criar as bases de dados de tamanho N, será necessário criar um gerador para as mesmas. Criaram-se 3 tipos de geradores, em que cada gerador é composto por K vetores cada um de dimensão I, são eles:

- Tipo 1 : Que corresponde à base canónica, ou seja,

$$e^1 = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{bmatrix}, e^2 = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \ldots, e^L = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \ldots, e^K = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

- Tipo 2 :

$$e^1 = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 1 \\ \vdots \\ 1 \end{bmatrix} \frac{1}{\sqrt{I-1}}, e^2 = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 1 \end{bmatrix} \frac{1}{\sqrt{I-1}}, \ldots, e^L = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 0 \\ \vdots \\ 1 \end{bmatrix} \frac{1}{\sqrt{I-1}}, \ldots, e^K = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \frac{1}{\sqrt{I-1}}$$

- Tipo 3: e^1 vai ser um vetor de dimensão I em que todas as suas componentes são aleatoriamente geradas e os outros $K-1$ vetores do gerador serão construídos a partir de e^1

$$e^1 = \begin{bmatrix} e_1^1 \\ e_2^1 \\ e_3^1 \\ \vdots \\ e_L^1 \\ \vdots \\ e_I^1 \end{bmatrix}, e^2 = \begin{bmatrix} e_2^1 \\ -e_1^1 \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{bmatrix}, e^3 = \begin{bmatrix} e_3^1 \\ 0 \\ -e_1^1 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \ldots, e^L = \begin{bmatrix} e_L^1 \\ 0 \\ 0 \\ \vdots \\ -e_1^1 \\ \vdots \\ 0 \end{bmatrix}, \ldots, e^K = \begin{bmatrix} e_I^1 \\ 0 \\ 0 \\ \vdots \\ 0 \\ \vdots \\ -e_1^1 \end{bmatrix}$$

Estes geradores são construídos de maneira a que os vetores $e^2, \ldots, e^K$ sejam ortogonais

a e^1.

Para criar a base de dados, será apenas necessário produzir combinações lineares destes vetores previamente gerados, ou seja, cada x^n é uma combinação linear de $e^1, e^2, ..., e^K$, melhor dizendo,

$$x^n = \rho_1^n e^1 + \rho_2^n e^2 + ... + \rho_K^n e^K, \rho_k \in \mathbb{R}$$

É de notar que o número de geradores para criar a base de dados não tem de ser máximo, ou seja, $K = I$; podemos gerar uma base de dados com K vetores tal que $K < I$, para tal selecionamos apenas os K primeiros vetores criados. O que vai suceder, ao escolher este último caso, é que a base de dados criada não será de característica máxima.

Vai-se testar um *autoencoder* linear para 3 bases de dados distintas que serão descritas de seguida:

- Base de Dados 1: Será um base de dados com 15 linhas, ou seja, $N = 15$, e vai usar um gerador de tipo 3. Além disso, vai-se considerar $I = 5$ e $K = 3$, e os ρ_k são todos gerados aleatoriamente.

- Base de Dados 2: Será um base de dados com $N = 15$, todos eles gerados aleatoriamente, sendo $I = 5$ e $K = 5$, mas, neste caso, os ρ_k vão ser criados de maneira a que $\rho_1 \geq \rho_2 \geq \ldots \geq \rho_K$, mais ainda, vamos forçar que a proporção entre eles seja muito grande.

- Base de Dados 3: Será a base de dados igual à anterior, mas vamos centrar a base de dados para posteriormente poder comparar resultados com a anterior.

4.2 Análise da precisão dos resultados

Nesta secção, o objetivo passa por quantificar o erro entre o que prevê a teoria matemática e o resultado efetivamente obtido do lado computacional.

Durante o processo de treino de um *autoencoder* o objetivo é que a função

$$E(X; U, V, c) = \sum_{n=1}^{N} \|x^n - VUx^n + c\|^2$$

seja zero. Na verdade, por questões de erro computacional, pretende-se que seja um valor muito perto de zero.

Para se poder, mais tarde, quantificar este erro faremos uso da seguinte definição.

Definição 4.2.1. *Sejam A e $\hat{A}$ duas matrizes $I \times J$, considerem-se, então, duas métricas.*

Média da diferença : $D(A, \hat{A}) = \dfrac{\sum_{i=1}^{I} \sum_{j=1}^{J} |A_{i,j} - \hat{A}_{i,j}|}{I \times J}$, *onde* $i \in [1, I], j \in [1, J]$.

Diferença máxima : $M(A, \hat{A}) = max(|A_{i,j} - \hat{A}_{i,j}|)$, *onde* $i \in [1, I], j \in [1, J]$.

De maneira a analisar os erros obtidos, usaremos as duas métricas da definição 4.2.1.

Expostas as métricas para analisar os erros e as características das base de dados a utilizar, façamos uma revisão prévias das propriedades a analisar ao longo desta secção:

Dada uma base de dados X :

(1) Se $X^T X$ tiver característica máxima, ou seja, a característica da base de dados X é I:

   (a) $U = (V^T V)^{-1} V^T$

   (b) $V = X^T X U^T (U X^T X U^T)^{-1}$

(2) Se X for uma base de dados centrada, vamos querer verificar que:

   (a) a e b são vetores nulos.

   (b) $-Va - b = 0$

(3) Calcular o desvio à ortogonalidade, em que o desvio à ortogonalidade consiste no módulo do valor máximo dos produtos internos entre os vetores da matriz U que sai do *autoencoder* e os $I - J$ vetores próprios associados aos menores $I - J$ valores próprios de $X^T X$.

Para melhor se avaliar os erros e comparar resultados, vai-se definir o erro de cada propriedade mencionada acima como:

- 1.(a) será avaliado através das quantidades :
  - E1Dif= $D(U, (V^T V)^{-1} V^T)$
  - E1Max= $M(U, (V^T V)^{-1} V^T)$
- 1.(b) será avaliado através das quantidades :
  - E2Dif= $D(V, X^T X U^T (U X^T X U^T)^{-1})$

- E2Max = $M(V, X^T X U^T (U X^T X U^T)^{-1})$

- 2.(b) será avaliado através da quantidade :

  - E3=$max(|-Va-b|)$

- 3. será avaliado através da quantidade :

  - E4 que corresponde ao desvio à ortogonalidade.

O *autoencoder* criado para realizar os testes será constituído apenas por uma camada, na qual o *optimizer* usado será o *adam*, e como *loss function* vai-se utilizar a função *mean squared error*. É de notar que para implementar este *autoencoder* fez-se uso das bibliotecas *tensorflow* e *keras* desenvolvidas em *Python*. O modelo vai ser treinado para 500/1000/5000/10000 épocas, de maneira a ver como se comporta com o aumento destas. Como foi visto acima, todas as bases de dados vão ter dimensão $N \times I$, onde $I = 5$ e serão reduzidas na função de *enconder* para um tamanho $J = 3$.

Observação 4.2.1. *O número de épocas é um parâmetro que controla o número de passagens completas pela base de dados usada para realizar o treino.*

Nas secções seguintes, vão ser realizados os testes para cada base de dados em separado.

4.2.1 Base de dados 1

Esta base de dados apresenta os seguintes parâmetros, $N = 15, I = 5, J = 3$

$$X = \begin{bmatrix} 59.53546264 & -27.83594759 & 39.83075352 & -35.91366852 & 78.81460737 \\ 20.97614854 & -3.22286397 & -50.3546249 & 19.60797172 & -43.03081963 \\ -28.01474761 & 40.98643668 & -30.36044624 & 28.20799273 & -61.90405946 \\ 85.84098469 & 2.01178263 & 19.02370353 & -24.68369725 & 54.16979071 \\ 71.61191555 & 21.20528513 & -36.03677716 & 8.33804851 & -18.29832615 \\ -48.46111044 & 51.06868484 & -43.40856561 & 40.34636273 & -88.5424093 \\ 3.88831727 & 8.95116718 & -10.165277 & 5.99613819 & -13.1588695 \\ 91.12823326 & -2.59565793 & 51.58588619 & -42.24476183 & 92.70855512 \\ -45.01797896 & 16.47904845 & -46.83532317 & 34.24186924 & -75.14574788 \\ -63.53297162 & -22.35019833 & 28.22233429 & -6.32477983 & 13.88009244 \\ -49.19759089 & -34.41862669 & 42.11947759 & -18.14235631 & 39.81444247 \\ 25.61666366 & -12.07546761 & 14.80828945 & -14.35456456 & 31.50191602 \\ 67.60698631 & -19.70896649 & 33.70085212 & -32.80203482 & 71.98594858 \\ -39.9150502 & 7.49649812 & -24.39382548 & 20.67410098 & -45.3705015 \\ -12.5070048 & -6.78150166 & 30.022111 & -13.47909446 & 29.58064664 \end{bmatrix}$$

Processo de treino:

Vamos correr o modelo numa primeira fase para 500 épocas.

Realizando-se o respetivo treino, pode ver-se, fazendo uso da figura 4.1, que, ao longo das 500 épocas, o modelo está sempre a melhorar. No final, este consegue atingir um valor de *loss* de 371.0181.

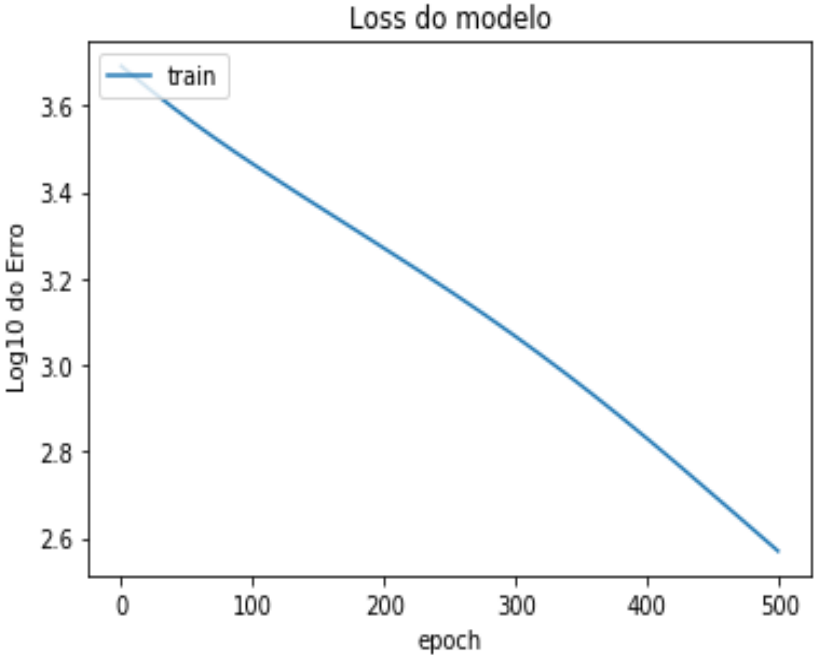


Figura 4.1: Treino da Base de dados 1 com 500 épocas

Além disso, vai-se também realizar os treino com 1000, 5000 e 10000 épocas. Apenas se vai exibir o gráfico correspondente às 10000 épocas.

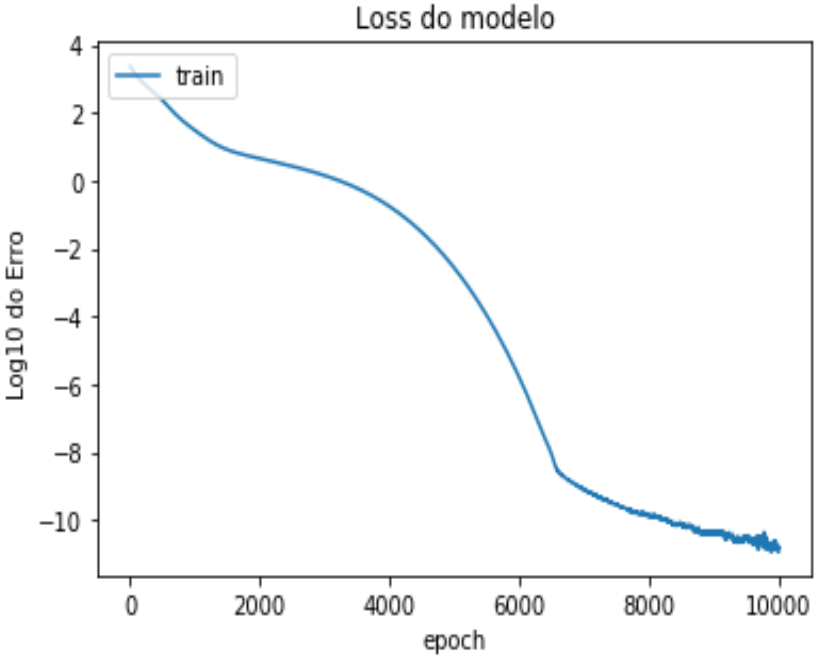


Figura 4.2: Treino da Base de dados 1 com 10000 épocas

Concluindo, com o caso das 10000 épocas, o valor de *loss* obtido foi 1.1974×10^{-11}. Note-se que é possível observar, através da figura 4.2, que, no final, o valor de *loss* aumenta e baixa, para tal ao correr o modelo para 10000 épocas vai-se sempre guardar a época que apresenta menor valor de *loss*, de forma a que no final de percorrer todas as épocas possamos usar os parâmetros U, V, a e b que correspondem ao treino com o valor de *loss* mais baixo, pois a época que apresenta o valor de *loss* mais baixo pode não ser a última a ser executada, como acontece neste caso. Este raciocínio será usado em todos os exemplos durante esta secção.

As matrizes de pesos e os vetores bias retornados pelo *autoencoder* ao percorrer as 10000 épocas foram os seguintes :

$$U = \begin{bmatrix} 1.0241157 & -0.04849293 & -0.16178618 & -0.12277623 & 0.2694395 \\ 0.04428794 & 0.47062027 & -0.30171528 & 0.23331937 & -0.5120327 \\ -0.44426918 & 0.8640096 & 0.5123271 & 0.01427269 & -0.03132215 \end{bmatrix}$$

$$V = \begin{bmatrix} 0.8722241 & 0.3441329 & -0.01226593 \\ 0.45747536 & 0.31513175 & 0.87193227 \\ 0.01464303 & -0.32133025 & 0.5016381 \\ -0.3113015 & 0.95508164 & -0.22692715 \\ 0.3454379 & -1.0090418 & 0.40135604 \end{bmatrix}$$

$$a = \begin{bmatrix} -0.12296712 & 0.10696594 & -0.15383813 \end{bmatrix}$$

$$b = \begin{bmatrix} 0.05285055 & 0.07661539 & 0.09119512 & -0.03785891 & 0.08308224 \end{bmatrix}$$

Quantificação dos erros:

Vamos, agora, construir a tabela que corresponde ao resultado dos erros para cada época, de maneira a poder analisar como se comportou o modelo para esta base de dados específica.

	500	1000	5000	10000
Loss	371.018	28.830	1.7136×10^{-4}	1.1974×10^{-11}
E1Dif	0.465	0.253	0.196	0.094
E1Max	1.582	0.838	0.554	0.252
E2Dif	0.958	0.675	0.256	0.145
E2Max	4.370	1.903	0.680	0.550
E3	0.555	0.411	0.533	0.137
E4	0.593	0.486	0.002	4.591×10^{-7}

Tabela 4.1: Valores do erro para a base de dados 1

De maneira a melhor visualizar e auxiliar estes resultados, vai-se exibir um gráfico de forma a comparar os erros obtidos acima consoante o número de épocas treinadas.

Como a base de dados não é centrada, o gráfico relativo a $E3$ não será exposto, pois este erro não apresenta qualquer interesse neste caso.

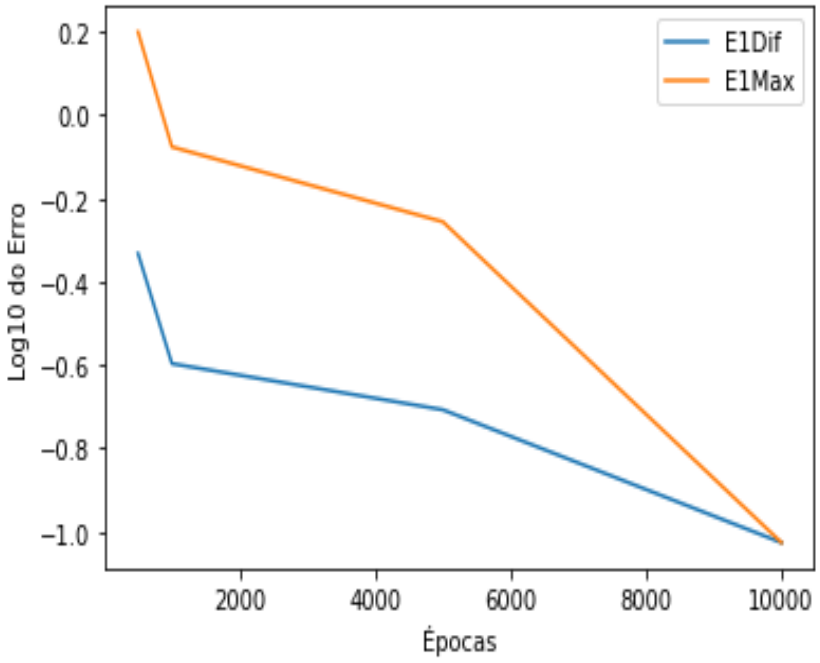


Figura 4.3: Diferença entre E1Dif e E1Max para a base de dados 1

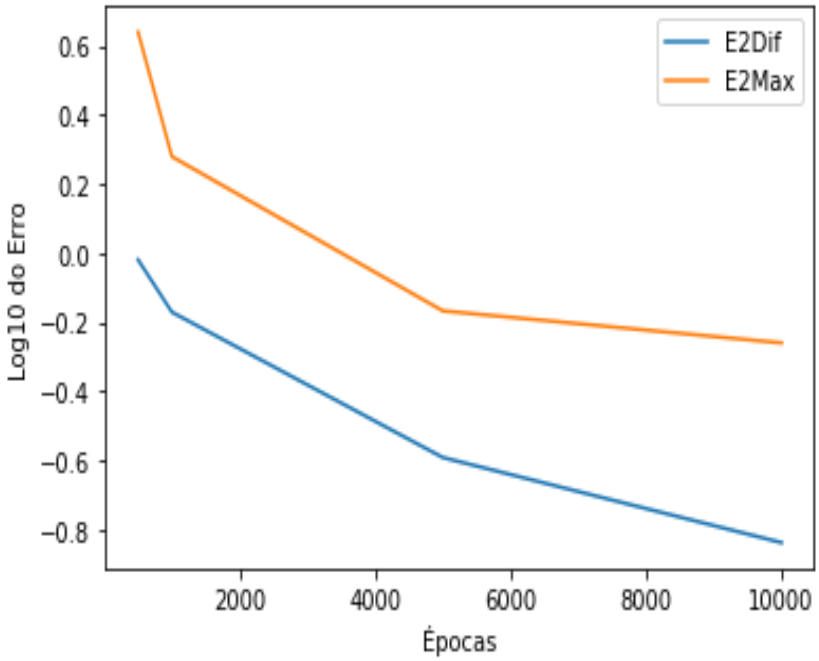


Figura 4.4: Diferença entre E2Dif e E2Max para a base de dados 1

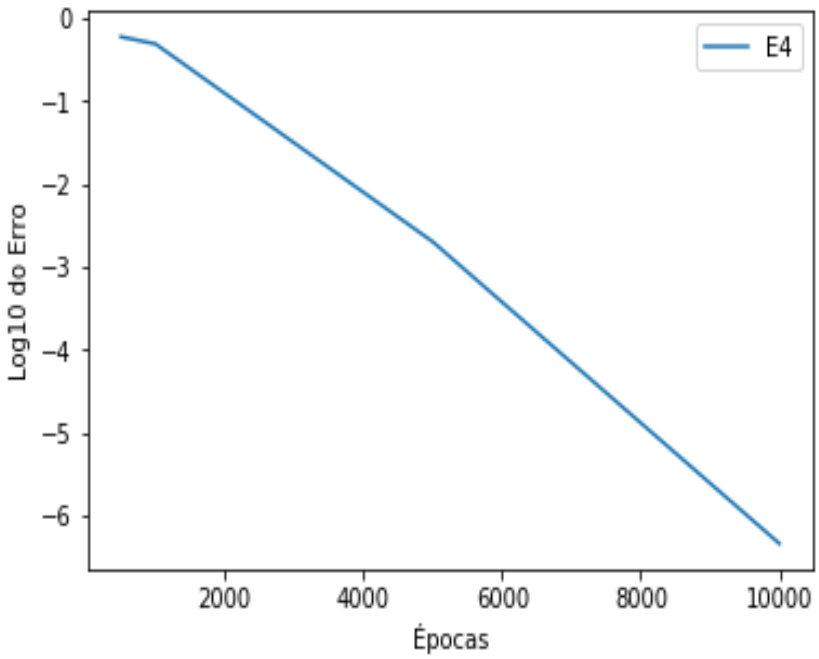


Figura 4.5: Representação de E4 para a base de dados 1

Discussão dos resultados:

Como esta base de dados tem característica 3, pois foi gerada por uma base de 3 vetores, por consequência, sabe-se que X^TX não tem característica máxima, ou seja, a característica não é 5. Logo, é de esperar que as propriedades 1. (a) e (b) não sejam satisfeitas. Aliás, através do erro de E1 e de E2 conseguimos ver que o valor obtido entre a diferença de ambas as matrizes, embora seja baixo, não é nulo, nem perto de nulo.

De maneira a visualizar melhor os resultados, vamos exprimir os valores dessas matrizes.

$$(V^TV)^{-1}V^T = \begin{bmatrix} 0.9816951 & 0.01266774 & -0.25999463 & -0.24695536 & 0.18780904 \\ 0.14387728 & 0.33350068 & -0.04874595 & 0.4279352 & -0.41724122 \\ -0.43465745 & 0.84526265 & 0.5176321 & 0.11570148 & 0.0604164 \end{bmatrix}$$

Neste caso, consegue-se facilmente ver que $(V^TV)^{-1}V^T \neq U$.

$$X^TXU^T(UX^TXU^T)^{-1} = \begin{bmatrix} 0.91438919 & 0.19278851 & 0.03863413 \\ 0.41088057 & 0.48237589 & 0.81568494 \\ 0.12347068 & -0.71194779 & 0.63301052 \\ -0.14488688 & 0.40503947 & -0.14340405 \\ 0.31796258 & -0.88888238 & 0.31470843 \end{bmatrix}$$

Trivialmente, nesta situação, também se observa que $X^TXU^T(UX^TXU^T)^{-1} \neq V$.

Como esta base de dados não é centrada, a e b, como era de esperar, são diferentes de zero.

E o valor do nosso desvio à ortogonalidade é 4.591×10^{-7}, ou seja, muito perto de zero como era necessário garantir, de modo a que as propriedades necessárias de verificar fossem satisfeitas.

Através da tabela 4.1 e da figura 4.3, conseguimos ver que $E1$ e $E2$ se vão tornando mais baixos consoante o número de épocas, e ainda que $E1Dif$ e $E1Max$ tendem a convergir para o mesmo valor. Fazendo uso da figura 4.4, podemos concluir que $E2Dif$ e $E2Max$ já não tendem a convergir para o mesmo valor.

Consegue-se também observar que, quanto a $E3$, existe um aumento na época 5000, mas tal pode ser explicado pelo facto de a base de dados não ser centrada e, então, $-b-Va$ não ter de ser zero, logo, é normal que esse erro possa variar.

Falando de $E4$, esse sim desce de uma maneira acentuada consoante o número de épocas,

como se pode observar na figura 4.5.

4.2.2 Base de dados 2

Esta base de dados apresenta os seguintes parâmetros, $N = 15, I = 5, J = 3$

$$X = \begin{bmatrix} -6.33098082 \times 10^3 & -1.78468734 \times 10^3 & -6.96223552 \times 10^2 & 33.6262197 & -14.7298599 \\ 1.05346697 \times 10^3 & -1.02351460 \times 10^2 & -5.30077199 \times 10^2 & -52.7763864 & 8.24380265 \\ -8.44546863 \times 10^3 & -1.95095904 \times 10^3 & -4.23769199 \times 10^2 & 266496714 & 163821789 \\ -10.3682958 & 4.06818762 \times 10^2 & -5.64708474 \times 10^2 & -48.8180410 & 13.5105884 \\ 6.45065244 \times 10^3 & 5.94789655 \times 10^2 & -7.53792149 \times 10^2 & -72.9289131 & -13.7516700 \\ -4.35005481 \times 10^3 & -1.00969134 \times 10^2 & 6.35630312 \times 10^2 & 14.8688037 & 17.3478484 \\ -3.18338334 \times 10^3 & 1.04887088 \times 10^3 & -4.81749433 \times 10^2 & 63.1202259 & -8.46589992 \\ 2.01225540 \times 10^3 & -1.62164740 \times 10^3 & 9.53252095 \times 10^2 & 92.3060536 & -17.3872626 \\ 7.60545269 \times 10^3 & -1.89263765 \times 10^3 & -1.83100524 \times 10^2 & -36.5245267 & 4.65863945 \\ 9.76998064 \times 10^3 & -1.55407400 \times 10^3 & 8.80085241 \times 10^2 & -8.86264805 & -12.3235132 \\ 9.45509565 \times 10^3 & 1.53437441 \times 10^3 & -24.4085441 & 68.5897989 & -6.51442104 \\ 1.81274974 \times 10^3 & 1.07705480 \times 10^3 & -5.31158667 \times 10^2 & -20.0157916 & 16.9726139\times \\ -4.90435172 \times 10^3 & -1.35594847 \times 10^3 & 6.86676570 \times 10^2 & -31.2644606 & 3.94770327 \\ 9.57843417 \times 10^3 & 2.30545244 \times 10^2 & -6.55865256 \times 10^2 & -89.3019082 & -18.2522830 \\ 6.67203276 \times 10^3 & -4.12631620 \times 10^2 & 5.56825092 \times 10^2 & 40.0385643 & -5.64605125 \end{bmatrix}$$

Processo de treino:

Treinado o modelo, ao fim das 500 épocas, atinge-se um valor de *loss* de 237689.031. Pode-se observar através da figura 4.6, que, ao fim de 500 épocas, o valor de *loss* continua a descer, o que nos indica que o modelo pode ainda ser melhorado com o aumento das mesmas. Outro aspeto a notar é que o valor de *loss* apresentado nesta base de dados é bem mais alto do que no exemplo anterior, e tal era de esperar dado que a característica desta base de dados é maior do que o tamanho dos dados codificados, J, pois o que aconteceu no caso anterior é que a característica da base de dados era igual ao tamanho dos dados codificados, logo, não havia perdas de informação ao passar pela função de *enconde*, por conseguinte, o valor de *loss* foi muito mais baixo.

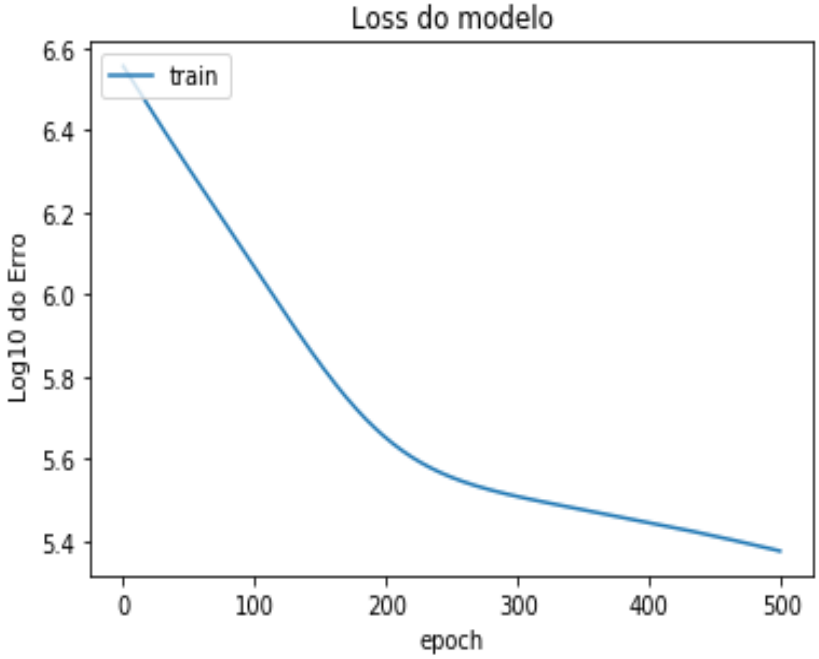


Figura 4.6: Treino da Base de dados 2 com 500 épocas

Vai-se apresentar, de seguida, o gráfico do comportamento do modelo ao correr 10000 épocas.

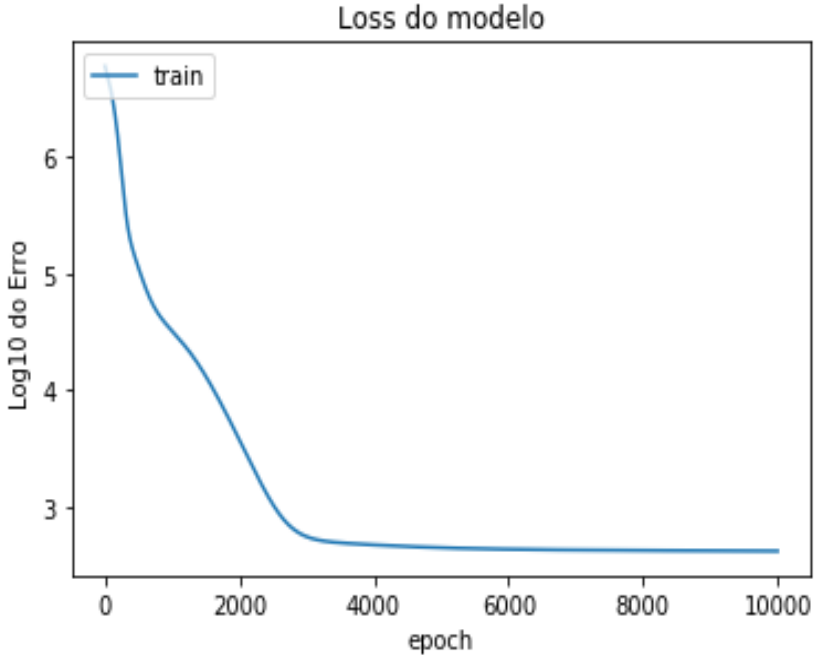


Figura 4.7: Treino da Base de dados 2 com 10000 épocas

Através da figura 4.7, consegue-se ver que, a partir da época 4000, o modelo deixa de aprender.

Neste exemplo, vamos ocultar o valor das matrizes U, V e dos vetores a, b porque o formato destas matrizes já foi mostrado acima e vamos apenas basear-nos no resultado dos erros para tirar as nossas conclusões.

Quantificação dos erros:

De seguida, vão-se ilustrar os erros obtidos para cada teste e realizar a análise dos mesmos.

	500	1000	5000	10000
Loss	237689.031	54239.104	562.231	425.466
E1Dif	0.310	0.382	0.125	0.002
E1Max	1.186	1.091	0.310	0.023
E2Dif	0.564	0.385	0.274	0.004
E2Max	1.385	1.244	0.834	0.028
E3	0.831	1.420	3.521	3.232
E4	0.694	0.637	0.363	0.0014

Tabela 4.2: Valores do erro para a base de dados 2

À semelhança do caso anterior, serão exibidos os gráficos dos erros para melhor entender como estes se comportam ao longo das várias épocas.

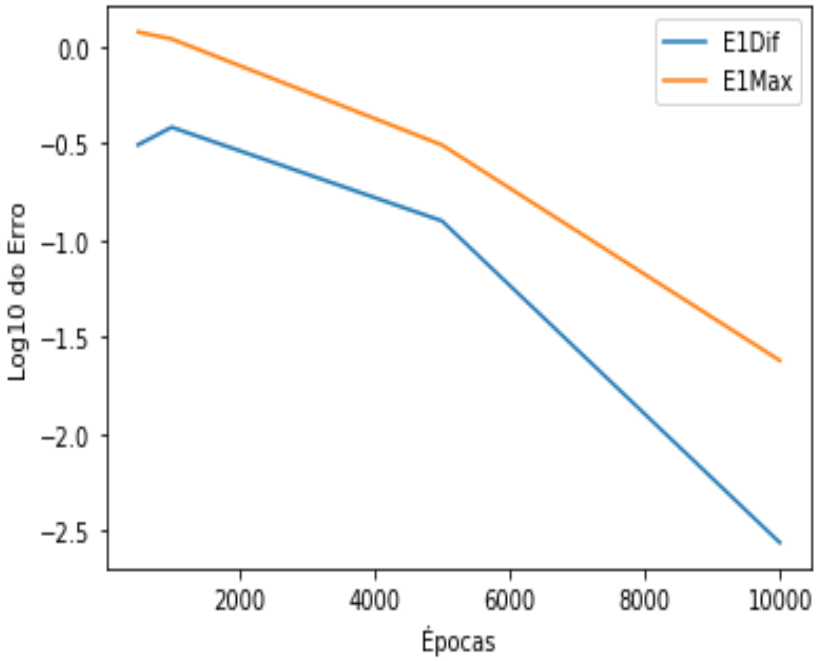


Figura 4.8: Diferença entre E1Dif e E1Max para a base de dados 2

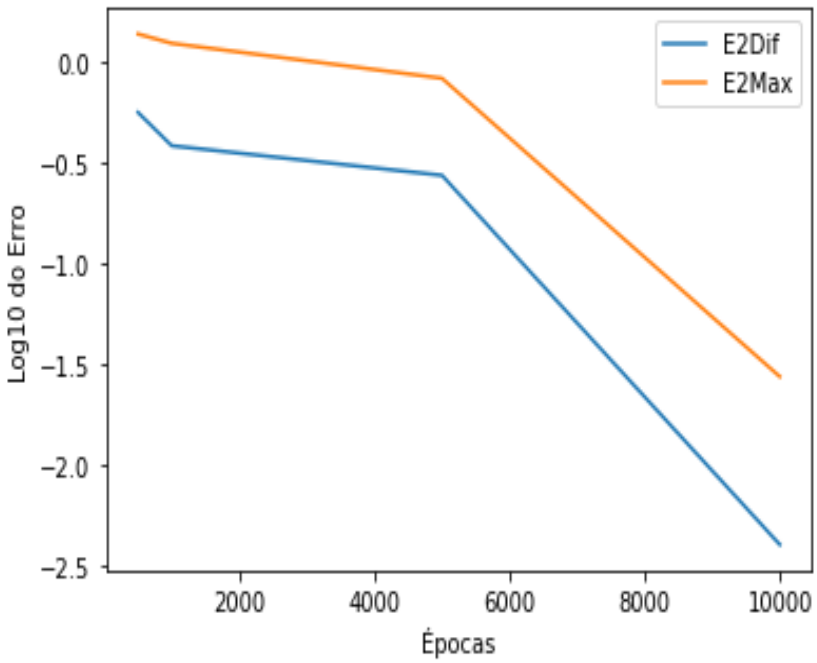


Figura 4.9: Diferença entre E2Dif e E2Max para a base de dados 2

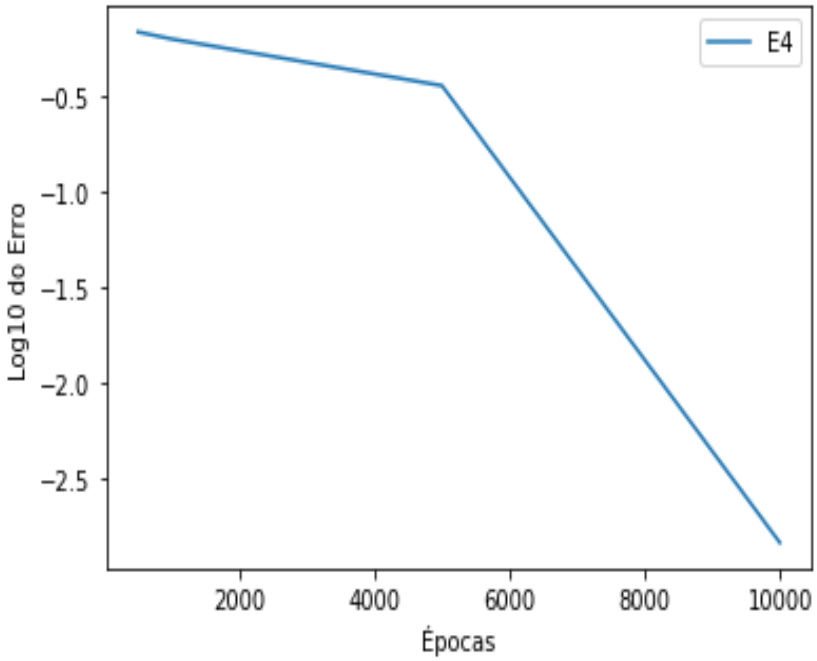


Figura 4.10: Representação de E4 para a base de dados 1

Discussão dos resultados:

Como esta base de dados tem característica 5, sabemos que X^TX tem característica máxima. Logo, é de esperar que as propriedades 1. (a) e (b) sejam satisfeitas.

Através do valor do erro $E1Max$, representado na tabela 4.2, conseguimos ver que a diferença máxima nas entradas das matrizes é de 0.023, posto isto, é possível concluir que, $U \simeq (V^TV)^{-1}V^T$.

E de maneira análoga podemos observar que $V \simeq X^TXU^T(UX^TXU^T)^{-1}$, pois $E2Max = 0.0278$.

Como esta base de dados não é centrada, a e b, como era de esperar, são diferentes de zero e, por conseguinte, o erro de $E3$ não vai ser nulo. Quanto ao valor do desvio à ortogonalidade, ele é 0.0014, ou seja, muito perto de zero como se queria garantir.

Por intermédio da figura 4.8 e da figura 4.9, deduzimos que os erros correspondentes a $E1$ e $E2$ estão a tender para valores muito baixos e, comparando ao caso anterior, podemos ver que eles apresentam valores mais pequenos. Tal era de esperar uma vez que, neste caso, X^TX tem característica máxima e, então, o valor de erro para $E1$ e $E2$ tinha, obrigatoriamente, de ser baixo de maneira a ir ao encontro do que foi provado na parte matemática deste documento.

Quanto ao valor de $E4$, segundo a figura 4.10, vemos que quantas mais épocas são executadas, mais o valor tende a descer. No entanto, este apresenta valores mais elevados do que no caso anterior e tal é justificado pelo facto, já mencionado acima, de que a característica da base de dados é superior ao tamanho dos dados codificados. Logo, é mais complicado treinar tal modelo uma vez que existe perda de informação.

4.2.3 Base de dados 3

Neste exemplo, consideremos de novo a base de dados do caso anterior, mas, em vez de usar a base de dados original, vamos centrar esta base de dados na média, e verificar qual a diferença nos resultados obtidos.

Dado que a base de dados é a mesma que a anterior, esta vai apresentar os mesmos parâmetros, ou seja, $N = 15, I = 5, J = 3$.

O vetor da média é dado por :

$$\begin{bmatrix} 1812.3676 & -392.23016 & -75.49224 & -1.4195561 & -1.0671719 \end{bmatrix}$$

e

$$X = \begin{bmatrix} -8.14334863 \times 10^3 & -1.39245728 \times 10^3 & -6.20731323 \times 10^2 & 35.0457764 & -13.6626883 \\ -7.58900635 \times 10^2 & 2.89878693 \times 10^2 & -4.54584961 \times 10^2 & -51.3568306 & 9.31097507 \\ -1.02578359 \times 10^4 & -1.55872876 \times 10 & -3.48276947 \times 10^2 & 28.0692272 & 17.4493504 \\ -1.82273584 \times 10^3 & 7.99048950 \times 10^2 & -4.89216248 \times 10^2 & -47.3984871 & 14.5777607 \\ 4.63828467 \times 10^3 & 9.87019836 \times 10^2 & -6.78299927 \times 10^2 & -71.5093613 & -12.6844978 \\ -6.16242236 \times 10^3 & 2.91261047 \times 10^2 & 7.11122559 \times 10^2 & 16.2883606 & 18.4150200 \\ -4.99575098 \times 10^3 & 1.44110107 \times 10^3 & -4.06257172 \times 10^2 2 & 64.5397873 & -7.39872742 \\ 1.99887817 \times 10^2 & -1.22941724 \times 10^3 & 1.02874426 \times 10^3 & 93.7256088 & -16.3200912 \\ 5.79308496 \times 10^3 & -1.50040747 \times 10^3 & -1.07608284 \times 10^2 & -35.1049728 & 5.72581148 \\ 7.95761279 \times 10^3 & -1.16184375 \times 10^3 & 9.55577515 \times 10^2 & -7.44309187 & -11.2563410 \\ 7.64272803 \times 10^3 & 1.92660449 \times 10^3 & 51.0836945 & 70.0093536 & -5.44724894 \\ 0.382202148 & 1.46928491 \times 10^3 & -4.55666443 \times 10^2 & -18.5962353 & 18.0397854 \\ -6.71671924 \times 10^3 & -9.63718323 \times 10^2 & 7.62168823 \times 10^2 & -29.8449059 & 5.01487541 \\ 7.76606689 \times 10^3 & 6.22775391 \times 10^2 & -5.80372986 \times 10^2 & -87.8823547 & -17.1851120 \\ 4.85966504 \times 10^3 & -20.4014587 & 6.32317322 \times 10^2 & 41.4581184 & -4.57887936 \end{bmatrix}$$

Processo de treino:

Na figura 4.11, é demonstrado o gráfico correspondente ao treino do modelo para 500 épocas, onde é atingido um valor de *loss* de 330229.0625.

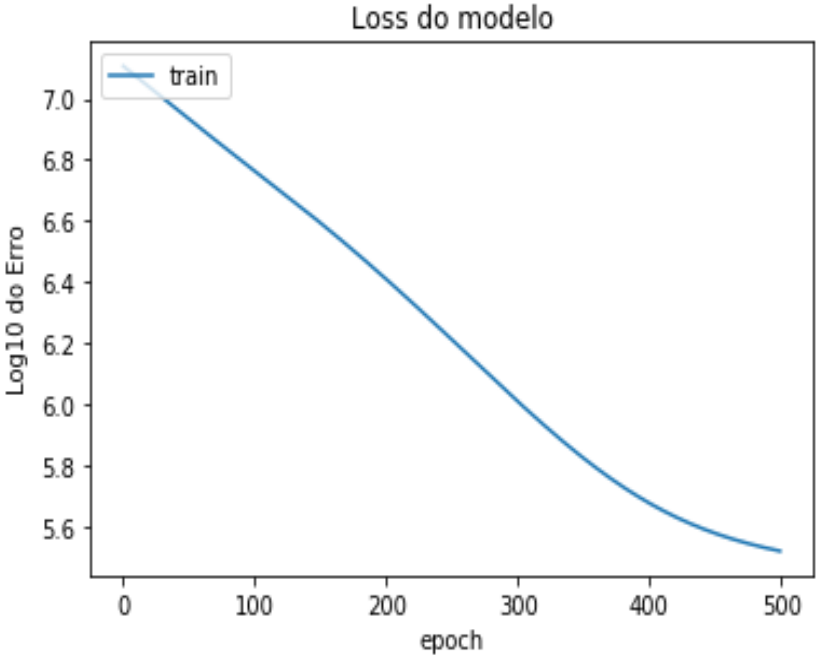


Figura 4.11: Treino da Base de dados 3 com 500 épocas

Treinado agora o modelo para 10000 épocas, é atingido um valor de *loss* de 415.70. Através da figura 4.12, pode-se concluir que o modelo estava a deixar de aprender, ou seja, não seria necessário aumentar o número de épocas.

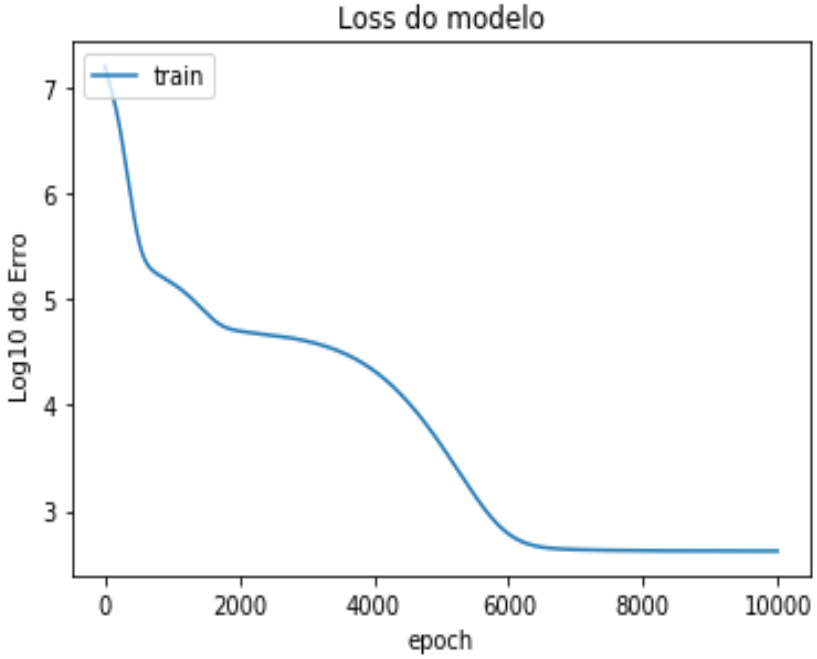


Figura 4.12: Treino da Base de dados 3 com 10000 épocas

As matrizes de pesos e os vetores bias retornados pelo *autoencoder* foram os seguintes:

$$U = \begin{bmatrix} -0.70670825 & -1.2454990 & -0.66011506 & -0.040725820 & -2.9013003 \times 10^{-3} \\ -0.15557285 & 0.23497075 & -0.73771417 & -0.032245975 & -8.5060465 \times 10^{-6} \\ 0.88561958 & -0.62008178 & -0.14992110 & -0.015719123 & -2.633959 \times 10^{-3} \end{bmatrix}$$

$$V = \begin{bmatrix} -0.38958818 & 0.17577943 & 0.8491213 \\ -0.53501344 & 0.5467223 & -0.33082247 \\ -0.08794504 & -1.2159069 & -0.28354242 \\ -0.00698135 & -0.05857696 & -0.02046256 \\ 0.00736276 & -0.04802873 & -0.01269148 \end{bmatrix}$$

$$a = \begin{bmatrix} 0.00022397 & -0.00817454 & -0.00106394 \end{bmatrix}$$

$$b = \begin{bmatrix} -1.8886343 \times 10^{-4} \\ 1.8111370 \times 10^{-3} \\ -5.7914569 \times 10^{-3} \\ -2.6269472 \times 10^{-4} \\ 5.8696487 \times 10^{-6} \end{bmatrix}^T$$

Quantificação dos erros:

A seguinte tabela exibe os resultados dos erros para cada teste realizado.

	500	1000	5000	10000
Loss	330229.062	101638.0078	596.419	415.70
E1Dif	0.560	0.366	0.021	0.0024
E1Max	1.263	0.989	0.096	0.0255
E2Dif	0.643	0.299	0.0493	0.005
E2Max	2.662	0.898	0.233	0.048
E3	0.00079	0.001	0.00056	0.004
E4	1.030	0.591	0.0265	0.008

Tabela 4.3: Valores do erro para a base de dados 3

De maneira a ter uma melhor percepção do erro, vão-se avaliar os gráficos dos mesmos e, de seguida, tirar as devidas conclusões.

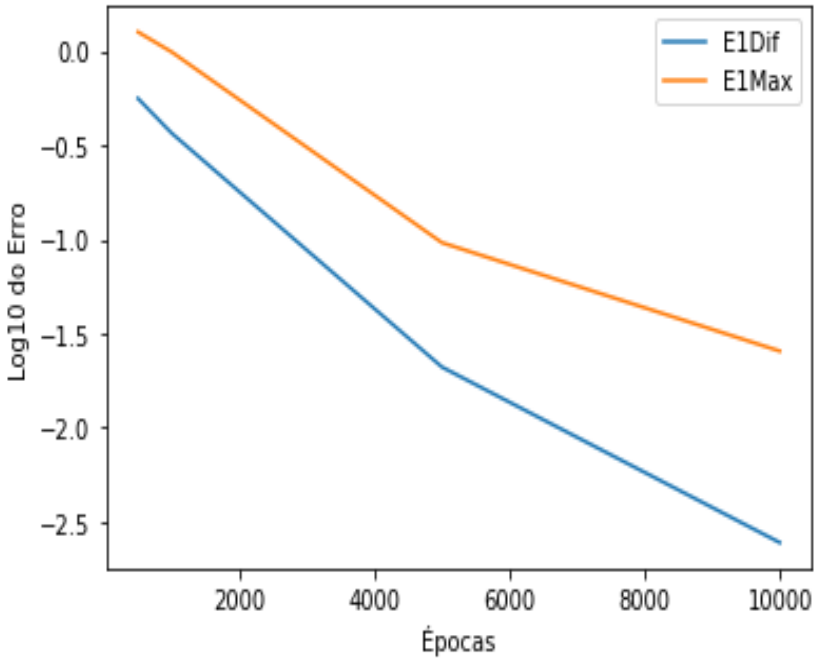


Figura 4.13: Diferença entre E1Dif e E1Max para a base de dados 3

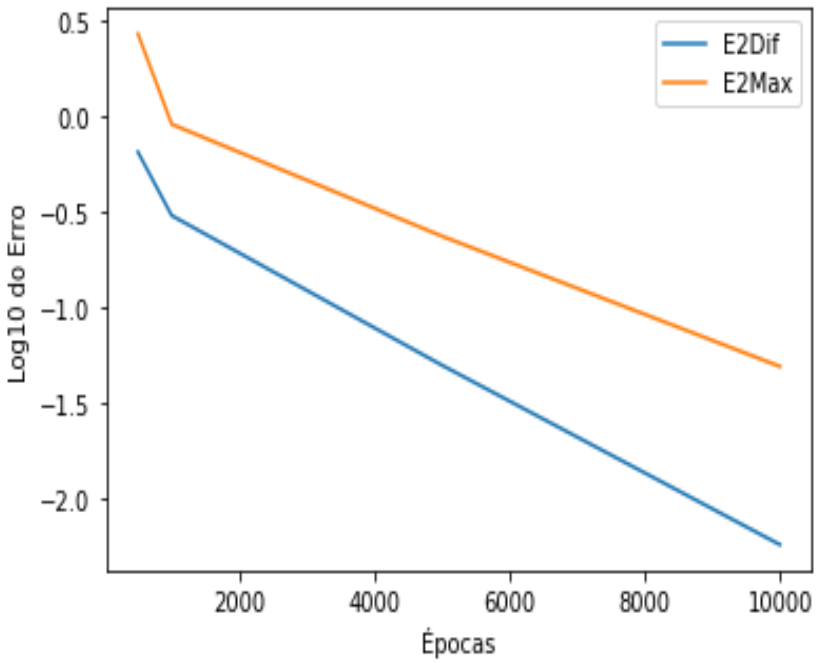


Figura 4.14: Diferença entre E2Dif e E2Max para a base de dados 3

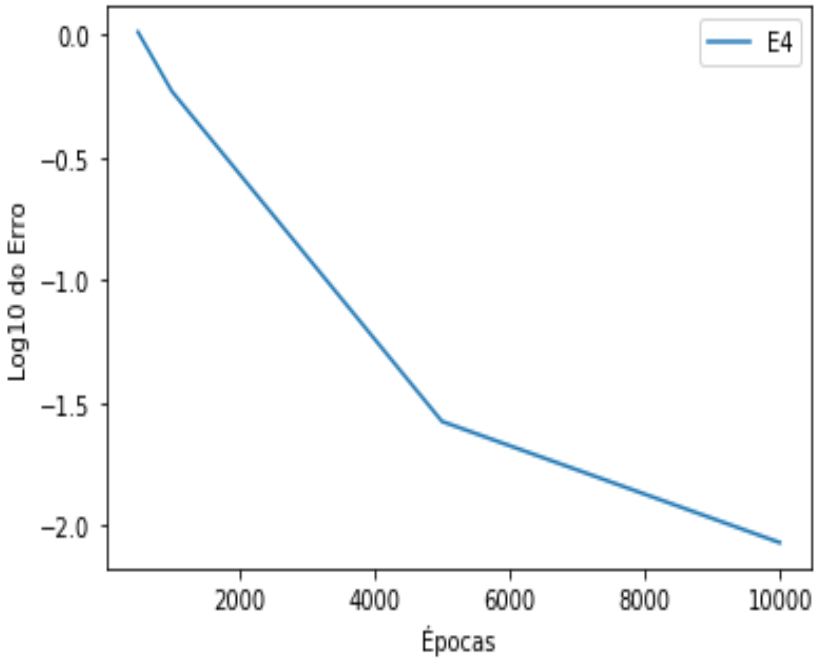


Figura 4.15: Representação de E4 para a base de dados 3

Discussão dos resultados:

Dado que esta base de dados corresponde a centrar a anterior, as propriedades estruturais são mantidas, ou seja, esta base de dados tem característica 5. Então, X^TX tem característica máxima. Assim, é de esperar que as propriedades 1. (a) e (b) sejam satisfeitas e, pelo facto de ser centrada, é, agora, de esperar que 2. (b) também o seja.

$$(V^TV)^{-1}V^T$$
$$=$$
$$\begin{bmatrix} -0.70667499 & -1.2454582 & -0.66024983 & -0.040310495 & 5.1562546 \times 10^{-4} \\ -0.15581350 & 0.23467232 & -0.73673618 & -0.035158552 & -0.025531733 \\ 0.88559747 & -0.62010932 & -0.14982978 & -0.015999259 & -5.0290320 \times 10^{-3} \end{bmatrix}$$

Através desta matriz, é possível ver que $(V^TV)^{-1}V^T \simeq U$, com exceção dos valores mais pequenos, e tal vai-se dever ao facto de estes estarem muito perto de zero, ou seja, a

diferença entre eles continua a ser muito pequena.

$$X^T X U^T (U X^T X U^T)^{-1}$$
$$=$$
$$\begin{bmatrix} -0.389591916 & 0.175864976 & 0.849149174 \\ -0.535008963 & 0.5465097391 & -0.330895706 \\ -0.0880121364 & -1.21643040 & -0.283784440 \\ -5.38759682 \times 10^{-3} & -0.0486284682 & -0.0156159260 \\ -5.13793876 \times 10^{-4} & 6.52848466 \times 10^{-4} & -1.41243366 \times 10^{-3} \end{bmatrix}$$

Facilmente se observa também que $X^T X U^T (U X^T X U^T)^{-1} \simeq V$, apenas com a exceção dos valores mais pequenos, que apresentam uma diferença entre eles também pequena.

$$-b - Va = \begin{bmatrix} 0.00261645 & 0.00242592 & -0.00443 & -0.00023635 & -0.00041363 \end{bmatrix}$$

Os vetores bias a e b são quase nulos, como era de esperar, dado que a base de dados em causa é centrada. Além disso, a propriedade 2.(b) também é verificada. Claramente, trabalhando a nível computacional, temos de verificar que será um vetor muito perto de zero e, através do valor do erro, conseguimos verificar tal facto. O erro de $E3$ é mais baixo do que nas duas base de dados anteriores, o que se deve ao facto de esta ser centrada.

Através da tabela 4.3, observa-se que o valor do desvio à ortogonalidade é 0.008, ou seja, muito perto de zero, como se pretende garantir.

Podemos verificar que, $E1$ e $E2$ têm um comportamento semelhante aos outros casos todos, ou seja, uma tendência dos erros serem menores consoante o número de épocas apresentado. Estas conclusões derivam do exibido na figura 4.13 e na figura 4.14.

Fazendo uso da figura 4.15, pode-se verificar que o erro de $E4$ desce acentuadamente com o aumento do número de épocas.

4.2.4 Comparação entre bases de dados centradas e não centradas

De modo a tentar perceber se existe uma melhoria ao escolher uma base de dados centrada ou não, e dado que qualquer base de dados pode ser centrada, vamos comparar o erro de ambos os casos em simultâneo, para tentar concluir algum facto em relação a este aspeto.

Vai-se, então, proceder à analise dos seguintes gráficos :

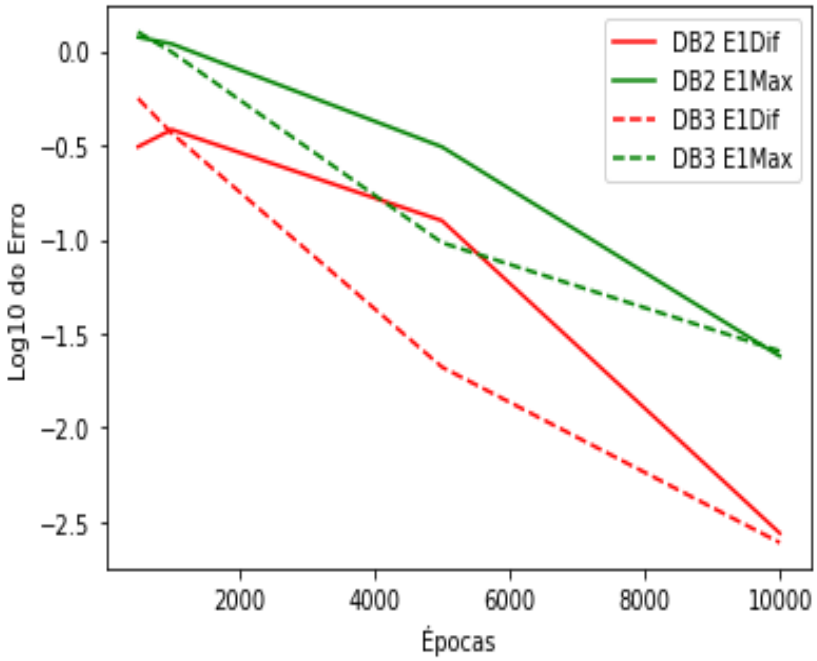


Figura 4.16: Diferença do erro E1 numa base de dados não centrada e centrada

Explorando a figura 4.16, através dos dois erros $E1Dif$ e $E1Max$, conseguimos analisar que, neste caso, a base de dados centrada(DB3) apresenta erros mais baixos do que a base de dados não centrada(DB2), no entanto, vemos que quando o modelo treina 10000 épocas, os erros são muito próximos um do outro.

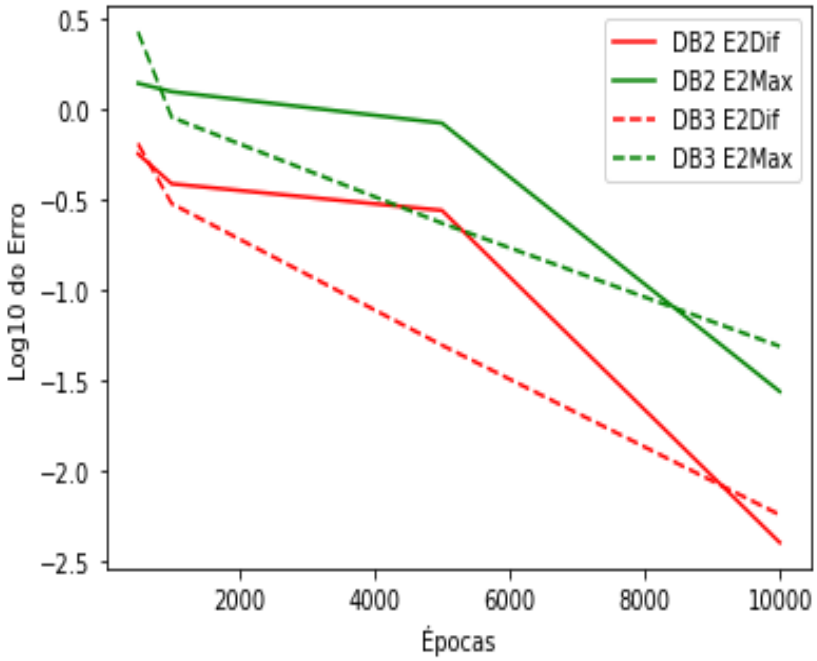


Figura 4.17: Diferença do erro E2 numa base de dados não centrada e centrada

Contudo, fazendo uma análise da figura 4.17, já se conclui o contrário, ou seja, no final das 10000 épocas, os resultados da base de dados não centrada(DB2), apresentam valores inferiores aos da base de dados centrada(DB3).

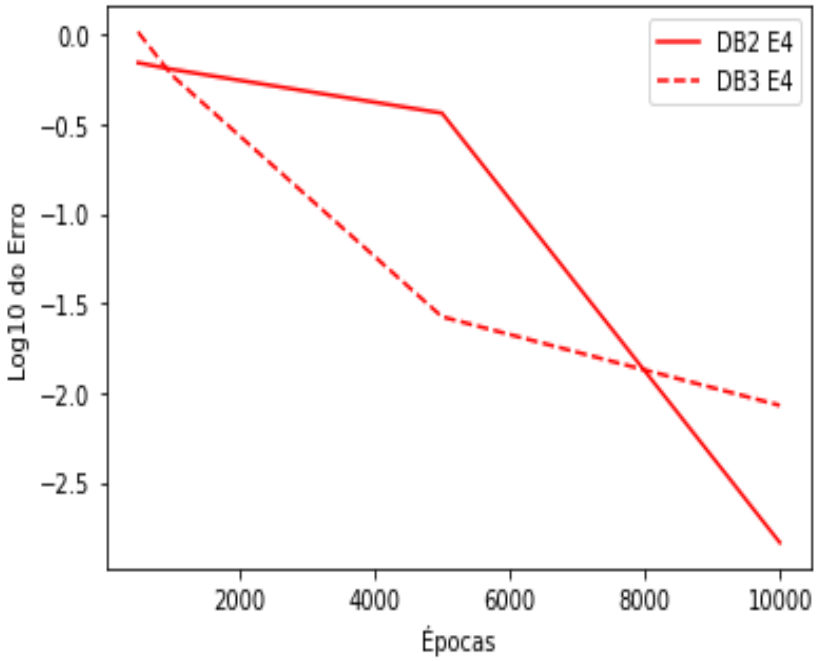


Figura 4.18: Representação de E4 numa base de dados não centrada e centrada

Falando de $E4$, o desvio à ortogonalidade é mais baixo para a base de dados não centrada(DB2), aliás a figura 4.18 ilustra isso mesmo.

Concluindo, embora se tenha conseguido verificar que os valores do erro obtidos são inferiores para dados não centrados, dado que a diferença dos erros não é muito acentuada e que foi analisado apenas um caso, não se pode concluir, para já, que seja efetivamente melhor usar uma base de dados não centrada.

Capítulo 5

Autoencoders não lineares

Nos dois capítulos anteriores o nosso estudou abordou apenas os *autoencoders* lineares. Neste capítulo iremos alargar o nosso estudo aos *autoencoders* não lineares, que acrescentam um ingrediente adicional aos *autoencoders* lineares, as denominadas funções de ativação. Estas funções, que dotarão os *autoencoders* da capacidade de ativar ou inibir os seus neurónios, serão apresentadas em detalhe na Secção 5.2. Dedicamos também uma secção deste capítulo aos regularizadores, que constituem um ferramenta auxiliar importante na obtenção experimental de valores de erro próximos dos valores mínimos teóricos.

5.1 Definição do *autoencoder* não linear

Tudo será semelhante aos *autoencoders* lineares no que toca aos dados de *input* e ao formatos das matrizes e vetores de pesos, ou seja, assumiremos que $V \in \mathbb{R}^{I\times J}$, $U \in \mathbb{R}^{J\times I}$ serão as matrizes de pesos e $a \in \mathbb{R}^{J\times 1}$, $b \in \mathbb{R}^{I\times 1}$ os vetores *bias*. Mas, além destes ingredientes, agora será ainda utilizada uma função de ativação, que irá ser denotada por α e que detalharemos mais tarde.

O *Encoder* será, mais uma vez, uma função que vai mapear o *input* recebido $x \in \mathbb{R}^{I\times 1}$, em $y \in \mathbb{R}^{J\times 1}$. No entanto, agora, será aplicada uma transformação no mesmo, mais especificamente, teremos $y = \alpha(Ux + a)$.

Quanto ao *Decoder*, será uma função dada pela seguinte expressão, $z = \alpha(Vy + b)$.

O *autoencoder* não linear será, então, constituído pela composta das duas funções:

(1) ***Encoder:*** $y = \alpha(Ux + a)$

(2) ***Decoder:*** $z = \alpha(Vy + b)$, que resulta em $z = \alpha(V\alpha(Ux + a) + b)$

A função que se vai procurar minimizar, para cada *input* x da base de dados, será dada por :

$$G(x;U,V,a,b) = \|x - z\|_2^2 = \|x - \alpha(Vy + b)\|_2^2 =$$
$$= \|x - \alpha(V\alpha(Ux + a) + b)\|_2^2 =$$

Nota 5.1.1. *É de salientar que* $\alpha(Ux + a) \neq \alpha(Ux) + a$

5.2 Funções de Ativação

No capítulo 3, foi possível verificar que por mais complexo que fosse o *autoencoder*, este só conseguia captar relações lineares entre o *input* e o *output*. Para este ser capaz de detetar também relações não lineares, os resultados do *output* de cada camada vão passar a ser processados por uma função de ativação.

As funções de ativação serão, então, cruciais para possibilitar que uma rede neuronal aprenda, pois elas vão permitir que, ao aplicar mudanças nos pesos e nos *bias*, apenas seja aplicada uma pequena modificação no *output*.

De certa maneira, estas vão poder determinar se um neurónio vai ser ativado ou não, ou seja, se a informação recebida deve ou não ser ignorada.

De seguida, descreveremos alguns tipos de funções de ativação, que são muito utilizados em redes neuronais, em particular em *autoencoders*. A maior parte da informação apresentada nesta secção pode ser encontrada em [1] e [6].

5.2.1 Binary Step Function

Uma das funções de ativação mais simples tem o nome de *binary step function* e pode ser vista como um classificador binário, que vai retornar 1 ou 0. No caso de x ser maior ou igual a 0, a função irá retornar 1, e irá retornar 0 caso x seja menor do que 0. Por outras palavras, se o valor x for positivo o neurónio é ativado, caso contrário não será.

A expressão da função é dada por :

$$f(x) = \begin{cases} 0 & se \quad x < 0 \\ 1 & se \quad x \geqslant 0 \end{cases}$$

E o gráfico da função pode ser consultado na figura 5.1.

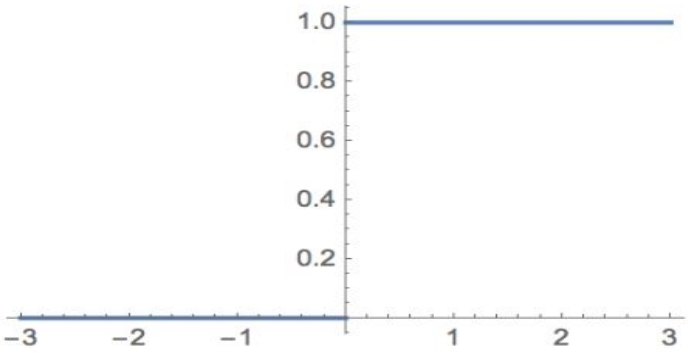


Figura 5.1: Função Binary Step

5.2.2 Sigmóide

A função *sigmóide* é uma função de ativação amplamente utilizada. Esta é representada pela seguinte expressão:

$$Sigmoide(x) = \frac{1}{1 + \mathrm{e}^{-x}}$$

O gráfico da função pode ser consultado na figura 5.2.

A função varia entre 0 e 1. Esta função vai, fundamentalmente, impulsionar os valores do *input* para os extremos. É útil quando tentamos classificar os valores para uma classe particular.

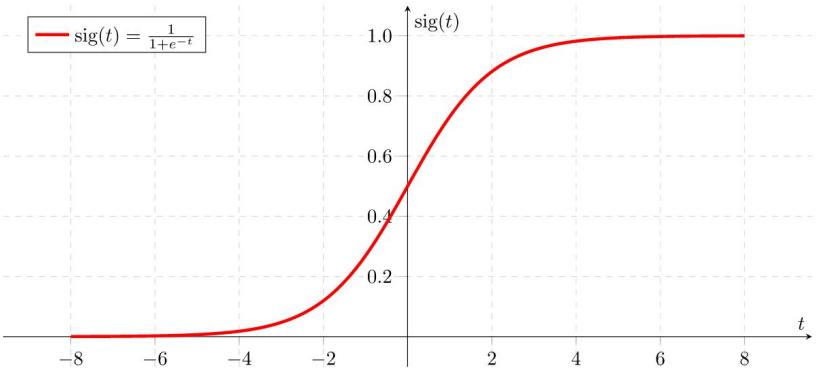


Figura 5.2: Função Sigmóide

5.2.3 Tangente Hiperbólica (Tanh)

A função *tanh* é muito semelhante à função *sigmóide*. É dada pela expressão:

$$\tanh(x) = \frac{2}{1 + \mathrm{e}^{-2x}} - 1$$
$$\Longleftrightarrow$$
$$\tanh(x) = 2Sigmoide(2x) - 1$$

E o gráfico da função pode ser consultado na figura 5.3.

A função tanh funciona de forma semelhante à função *sigmóide*, mas com a particularidade de ser simétrica em relação à origem, e esta varia de -1 a 1.

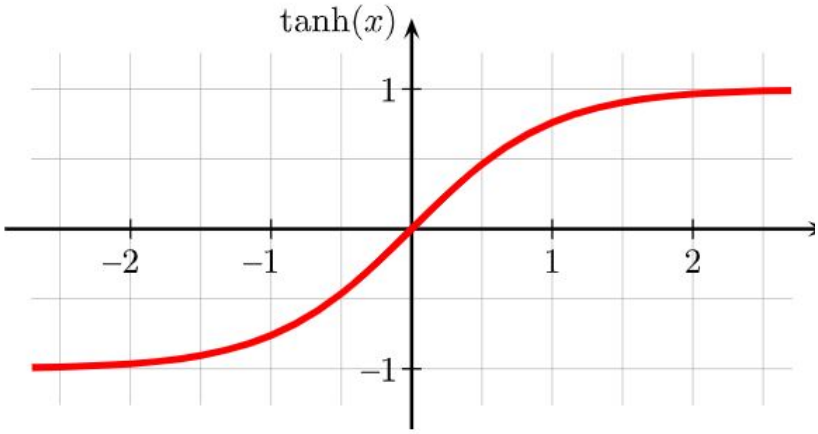


Figura 5.3: Função Tangente Hiperbólica

5.2.4 ReLU

A função Rectified Linear Unit (ReLU) é definida como:

$$\mathrm{ReLU}(x) = \max(0, x)$$

O gráfico da função pode ser consultado na figura 5.4.

A função ReLU é a função de ativação normalmente mais utilizada, e será aquela que aplicaremos em geral nas nossas implementações computacionais.

Dado um *input*, a função ReLU vai converter *inputs* negativos em zero e o neurónio não será ativado. Por outro lado, se o *input* for positivo este valor vai-se manter igual.

A principal desvantagem da função de ativação ReLU é que as unidades podem ter uma tendência, durante o processo de treino, de produzir apenas zeros. Isto acontece quando a soma ponderada antes de aplicar a ReLU se torna negativa, fazendo com que o vetor *output* da camada retorne o vetor nulo.

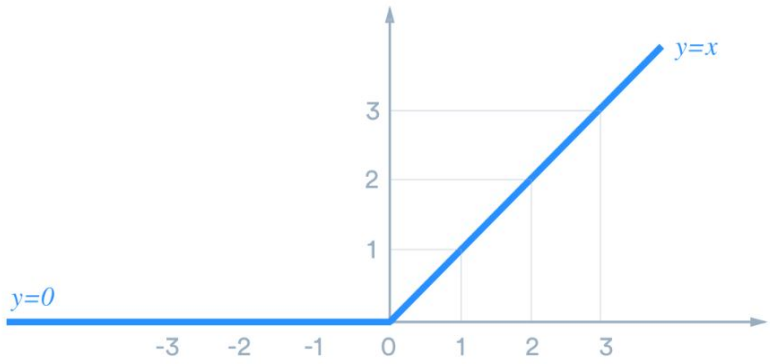


Figura 5.4: Função ReLU

5.2.5 Leaky ReLU e Parametric ReLU

Um grande problema da ReLU, acontece quando alguns neurónios não são ativados, permanecendo inativos, independentemente do *input* fornecido. Se um grande número de neurónios inativos estiver na rede neuronal, então, o desempenho da mesma está a ser afetado. Para tal, pode-se corrigir este problema fazendo uso da função *leaky ReLU* (L-ReLU), na qual o resultado da parte negativa é alterado de 0 para o valor $0.01x$, isto é, permite a existência de um valor não nulo quando aplicado o método do gradiente. Esta função pode ser generalizada para um valor qualquer a, e, assim, obtém-se a função Parametric ReLU. É trivial notar que a função L-ReLU é um caso particular da Parametric ReLU.

$$\text{L-ReLU}(x) = \begin{cases} x & : \quad \text{se } x > 0 \\ 0.01x & : \quad \text{caso contrário} \end{cases}$$

Na figura 5.5, pode-se analisar o gráfico de ambas as funções mencionadas acima.

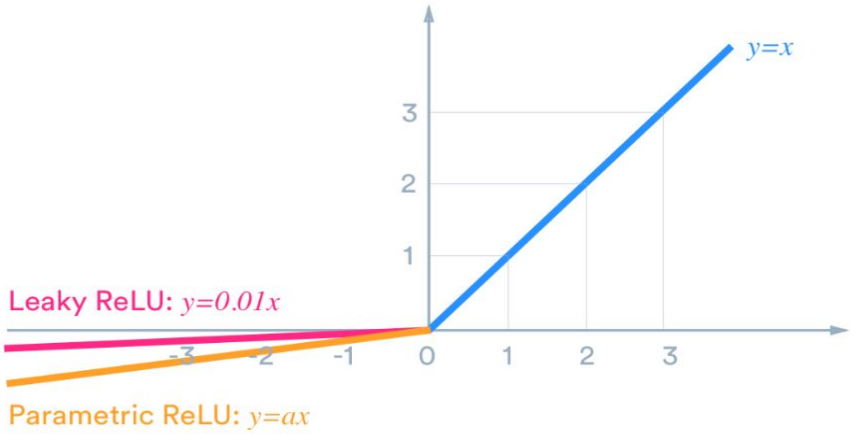


Figura 5.5: Função Leaky ReLU e Parametric ReLU

5.2.6 Exponential Linear Unit (ELU)

A função *Exponential Linear Unit* (ELU) é uma alternativa ao uso da função L-ReLU, pois tem como objetivo resolver o mesmo problema, descrito em 5.2.5, que surge na função ReLU.

O problema da função ELU surge do facto de utilizar a função exponencial, que é uma operação ineficiente, podendo tornar o processo de treino de um *autoencoder* mais lento. Contudo, por vezes, *autoencoders* com uma função de ativação ELU superam os resultados obtidos com as funções ReLU e L-ReLU.

A definição da função ELU é a seguinte:

$$\text{ELU}(x) = \begin{cases} x & : \quad \text{se } x > 0 \\ (e^x - 1) & : \quad \text{caso contrário} \end{cases}$$

A derivada desta função é representada pela seguinte expressão:

$$\frac{d}{dx}\text{ELU}(x) = \begin{cases} 1 & : \quad \text{se } x > 0 \\ e^x & : \quad \text{caso contrário} \end{cases}$$

Na figura 5.6, pode ser visto o gráfico da função.

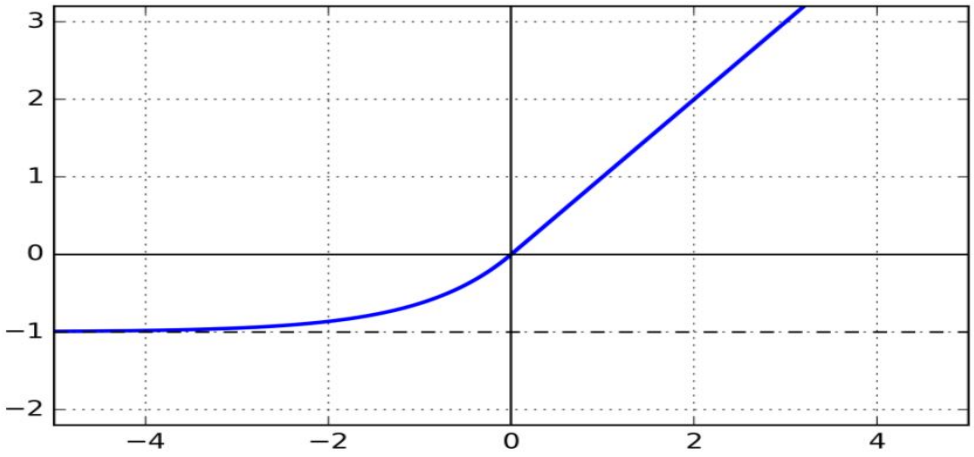


Figura 5.6: Função ELU

5.3 Regularizadores

Treinar um *autoencoder*, como já se viu acima, corresponde a encontrar parâmetros, tais que, dada uma função de *loss*, o *input* x, seja igual ao *output* $\hat{x}$.

Por vezes, os parâmetros estabelecidos durante o treino tendem a ser muito complexos, logo, não vão generalizar bem os dados e, porventura, será mais complicado determinar um mínimo para uma função que se queira minimizar. É nesta alturas que a regularização tem um papel importante, uma vez que esta técnica, que vai ter em conta a complexidade dos pesos durante a optimização dos mesmos, pode conduzir o *autoencoder* a suavizar tal função de minimização.

De seguida, será dado um exemplo ilustrativo desta questão.

Exemplo 5.3.1. *Considere-se a função:*

$$\Pi(x; a_0, a_1) = ReLU(a_0 + a_1 x)$$

e sejam $M1 = (-1, \frac{1}{2}), M2 = (0, 0)$ *e* $M3 = (1, 1)$ *três pontos.*

Procuram-se os a_0 *e* a_1*, tais que,* Π *seja o mais próximo dos pontos, ou seja, quer-se minimizar a função:*

$$E(a_0, a_1) = (\Pi(-1; a_0, a_1) - \frac{1}{2})^2 + (\Pi(0; a_0, a_1) - 0)^2 + (\Pi(1; a_0, a_1) - 1)^2$$

Seja ainda a função R correspondente à regularização, dada por:

$$R(a_0, a_1) = a_0^2 + a_1^2$$

Aplicando regularizadores, a função $E(a_0, a_1)$ vai-se definir como sendo:

$$E(a_0, a_1) = (\Pi(-1; a0, a1) - \frac{1}{2})^2 + (\Pi(0; a_0, a_1) - 0)^2 + (\Pi(1; a_0, a_1) - 1)^2 + \lambda R(a_0, a_1)$$

Analisemos agora como se comportam os gráficos para diferentes valores λ de regularização.

A figura 5.7 ilustra o gráfico da função quando não é aplicada regularização. É de notar que as zonas mais escuras indicam os pontos mínimos e as mais claras os máximos.

Neste caso, é fácil notar que existem vários pontos que podem ser mínimos locais, por exemplo, na zona laranja.

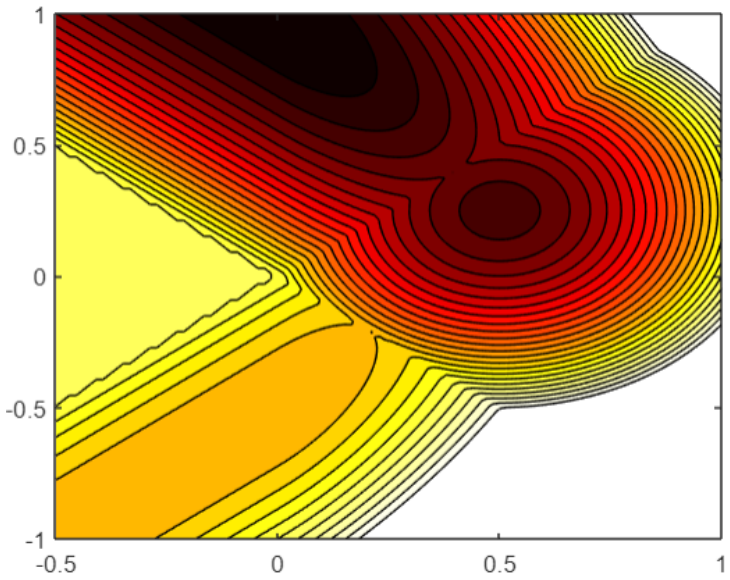


Figura 5.7: $E(a_0, a_1)$ com $\lambda = 0$

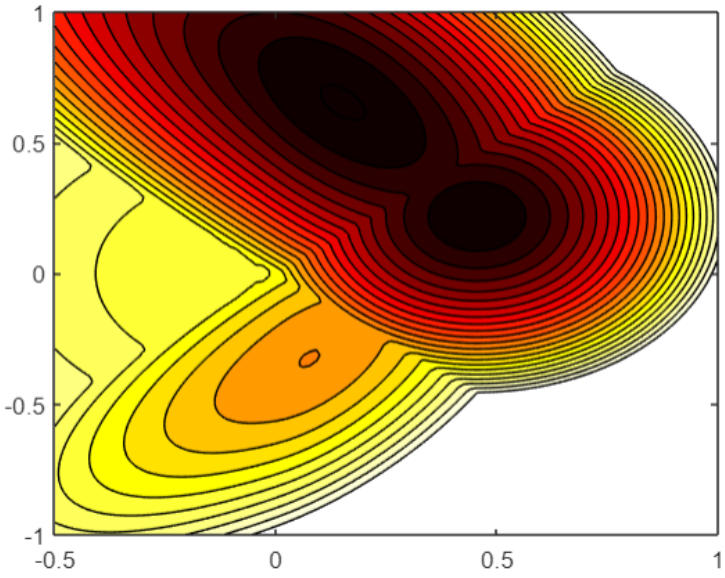


Figura 5.8: $E(a_0, a_1)$ com $\lambda = 0.3$

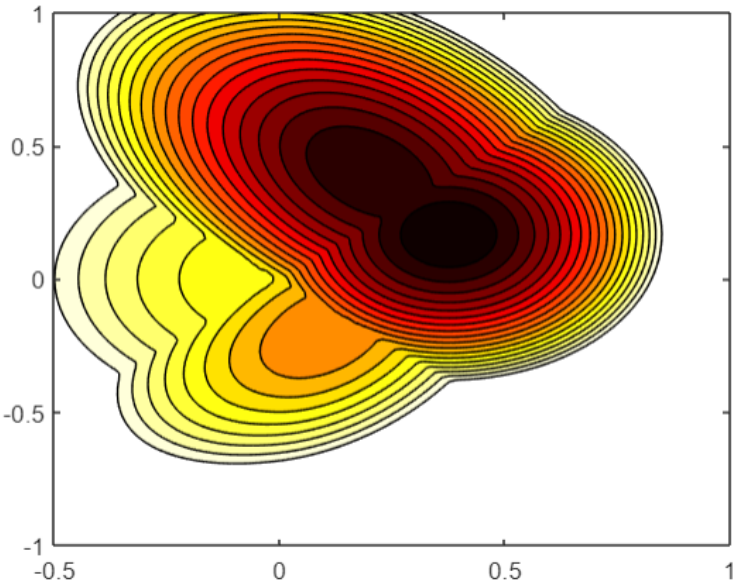


Figura 5.9: $E(a_0, a_1)$ com $\lambda = 1$

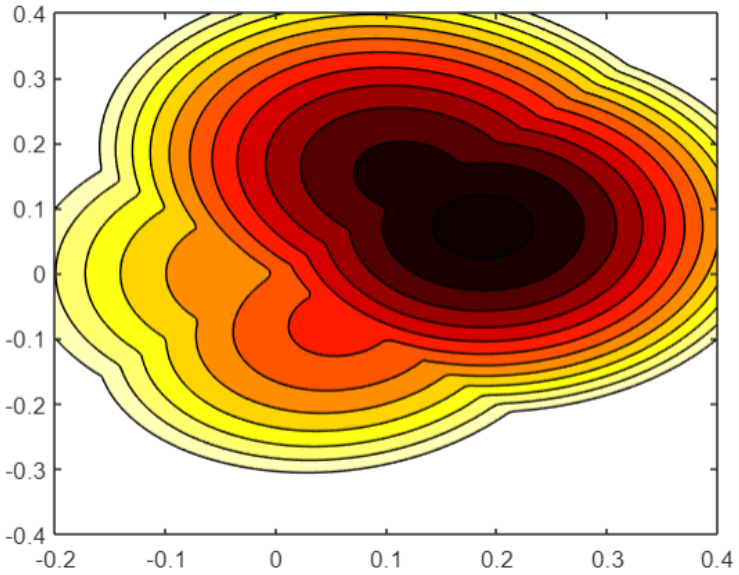


Figura 5.10: $E(a_0, a_1)$ com $\lambda = 5$

Caso seja aplicado um valor de λ de 0.3, é possível analisar que continuam a existir vários mínimos locais, ou seja, ao treinar um modelo para encontrar o mínimo, este pode não conseguir encontrar o mínimo global e ficar preso num mínimo local.

No entanto, considerando-se o caso que utiliza $\lambda = 1$, já se começa a notar a remoção dos mínimos locais e pode ver-se que todos os pontos vão começar a tender para o mínimo global.

Caso λ seja igual a 5, aí sim, já se consegue observar que todos os mínimos locais foram removidos e um algoritmo de otimização com um número de épocas suficiente, à partida, será capaz de encontrar o mínimo global.

Resumindo, pode-se ver que se $\lambda < 1$, existem 2 mínimos, enquanto que se $\lambda > 1$, apenas existe um. Logo, neste último caso, a convergência é para o mínimo global.

Os regularizadores vão, então, permitir aplicar penalizações aos parâmetros da camada ou às funções de ativação durante a otimização. Estas penalizações são somadas à *loss function* que a rede neuronal está a otimizar, no nosso caso concreto, aos *autoencoders*.

Existem vários tipos de regularizadores, contudo, ao longo deste trabalho, apenas utilizaremos regularização *L1* (*Lasso*) e *L2* (*Ridge*) [18].

Explicada a importância do processo de regularização, de seguida, mostraremos como é que estes funcionam concretamente.

Consideremos a função de *loss* mencionada no capítulo 3, ou seja,

$$E(X;U,V,a,b) = \sum_{n=1}^{N} F(x^n;U,V,a,b) = \|x - VUx - Va - b\|^2$$

O objetivo durante o processo de treino é encontrar o $\min(E(X;U,V,c))$.

Então, o que vai acontecer durante o processo de treino de um *autoencoder* usando regularizadores é que a função de *loss* será agora representada por

$$E(X;U,V,a,b) = E(X;U,V,a,b) + \lambda R(U,V,a,b)$$

onde λ é um hiperparâmetro, não negativo, que determina a importância que se quer dar à componente relativa à regularização.

Vamos, assim, estudar duas possibilidades para $R(U,V,a,b)$: regularização *L1* e regularização *L2*.

5.3.1 Regularização L1

Quando é aplicada regularização *L1*, está a ser dito que:

$$R(U,V,a,b) = \sum|U_{i,j}| \ + \sum|V_{i,j}| \ + \sum|a_j| \ + \sum|b_i| \text{ , onde } i \in [1,I] \text{ e } j \in [1,J]$$

Existe também uma regularização em que os vetores *bias* a e b não são utilizados.

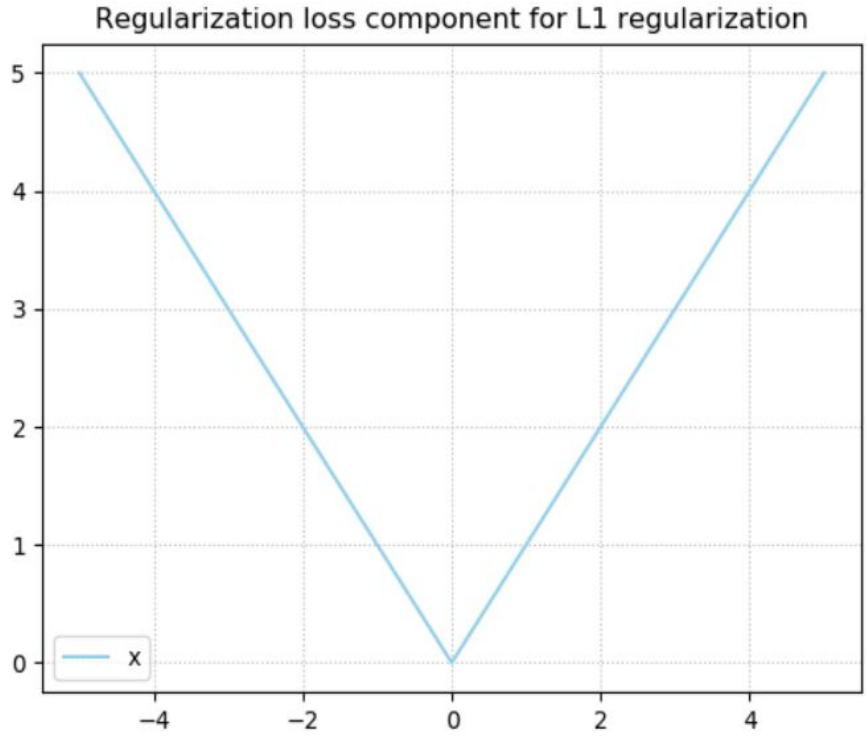


Figura 5.11: Regularizador L1 [18]

Ao aplicar regularização *L1*, vai-se garantir que dado uma *loss function* relativamente constante, os pesos vão assumir valores muito pequenos, já que o valor L1 para $x = 0$ é o mais baixo, como pode ser observado na figura 5.11. Aliás, o mais provável é que os valores se tornem mesmo zero, pelo facto de a derivada de *L1* ser constante, é fácil de notar que a derivada de *L1* é sign(w). Aplicar a regularização *L1* às camadas de uma rede neuronal faz com que os pesos que não contribuem para o seu poder preditivo de forma significativa sejam eliminados.

5.3.2 Regularização L2

No caso de ser aplicada a regularização *L2*, está a ser dito que:

$$R(U, V, a, b) = \sum U_{i,j}^2 + \sum V_{i,j}^2 + \sum a_j^2 + \sum b_i^2, \text{ onde } i \in [1, I] \text{ e } j \in [1, J]$$

Neste caso, à semelhança da regularização *L1*, também existe uma regularização em que os vetores *bias* a e b não são utilizados.

L2 é uma expressão quadrática, como pode ser visto na figura 5.12.

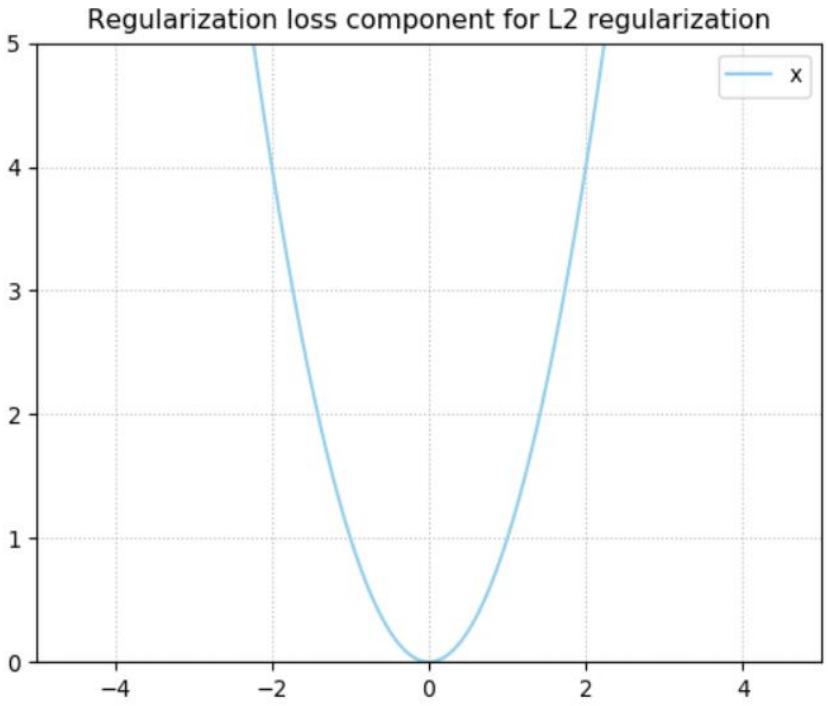


Figura 5.12: Regularizador L2 [18]

Ao aplicar regularização *L2*, faz-se com que os pesos obtenham valores relativamente pequenos, ou seja, é semelhante a aplicar regularização *L1*. No entanto, ao contrário de *L1*, a regularização *L2* não empurra os pesos para serem exatamente zero. Isto é devido ao valor da derivada, onde, como mencionado previamente, a derivada de *L1* é uma constante, é 1 ou -1, a derivada *L2* é $2x$. Isto significa que quanto mais perto estiver o valor de zero, menor será a derivada e, portanto, menor será a atualização; no entanto, este valor nunca vai chegar a zero, embora fiquem realmente pequenos. É útil utilizar *L2* em situações em que não pode ser aplicada a regularização *L1*.

5.3.3 Regularizadores em *Python*

Concluiremos esta secção com uma breve incursão sobre o funcionamento dos regularizadores em *Python*. Consideremos que se tem a seguinte equação $y = Wx + b$. No *tensorflow*, que vai ser a biblioteca a utilizar quando se passar para uma parte prática, os regularizadores podem ser aplicados da seguinte forma:

(1) *Kernel Regularizer*: Tenta reduzir os pesos W, excluindo o vetor *bias*;

(2) *Bias Regularizer*: Tenta reduzir o vetor *bias* b;

(3) *Activity Regularizer*: Tenta reduzir a camada de *output* y, consequentemente, tenta reduzir a matriz de pesos W e ajustar o *bias* tal que $Wx + b$ seja o mais pequeno.

Capítulo 6

Autoencoder ReLU

Este capítulo dedica-se ao caso particular dos *autoencoders* ReLU, ou seja, aos *autoencoders* não lineares cuja função de activação α é a função ReLU descrita na secção 5.2.4. Devido à estrutura particular da função ReLU, com uma parte linear, o objetivo deste capítulo passará por investigar em que casos se pode aproveitar a decomposição U e V do *autoencoder* linear para construir aproximações do *autoencoder* não linear.

6.1 Definição do *autoencoder* ReLU

O *autoencoder* ReLU será, então, constituído pela composta das duas funções:

(1) ***Encoder:*** $y = \mathrm{ReLU}(Ux + a)$

(2) ***Decoder:*** $z = \mathrm{ReLU}(Vy + b)$, que resulta em $z = \mathrm{ReLU}(V\mathrm{ReLU}(Ux + a) + b)$

Relembremos que estamos a utilizar uma base de dados X com dimensão N.
Então, a função de minimização poderá ser escrita como,

$$\mathcal{L}(X; U, V, a, b) = \frac{1}{N \times I} \sum_{n=1}^{N} \|x^n - \mathrm{ReLU}(V\mathrm{ReLU}(Ux^n + a) + b)\|^2$$

Definamos $\max Ux$ e $\min Ux$ como sendo dados pela seguinte expressão:

$$\max Ux = \begin{bmatrix} \max Ux_1 \\ \max Ux_2 \\ \vdots \\ \max Ux_J \end{bmatrix}, \min Ux = \begin{bmatrix} \min Ux_1 \\ \min Ux_2 \\ \vdots \\ \min Ux_J \end{bmatrix}$$

onde, $\max Ux_i$ representa o valor máximo apresentado nas i-ésimas componentes de todos os vetores $Ux^1, Ux^2, \ldots, Ux^n$ (analogamente, para $\min Ux_i$).

Exemplo 6.1.1. *Supondo que se tem uma matriz U e três vetores x^1, x^2, x^3 quaisquer, tais que:*

$$Ux^1 = \begin{bmatrix} 4 \\ 3 \\ 1 \end{bmatrix}, Ux^2 = \begin{bmatrix} 7 \\ 2 \\ -2 \end{bmatrix} \text{ e } Ux^3 = \begin{bmatrix} 3 \\ 9 \\ -1 \end{bmatrix} \text{ então,}$$

$\max Ux_1 = \max(4, 7, 3) = 7$, $\max Ux_2 = 9$ *e* $\max Ux_3 = 1$.

Logo, $\max Ux = \begin{bmatrix} 7 \\ 9 \\ 1 \end{bmatrix}$ *e, analogamente,* $\min Ux = \begin{bmatrix} 3 \\ 2 \\ -2 \end{bmatrix}$

De maneira a simplificar a notação, vai-se denotar $\min Ux$ por $\underline{a}$ e $\max Ux$ por $\overline{a}$.

Definição 6.1.1. *Dados dois vetores v e w, diz-se que $v \geq w$ se $\forall j$, $v_j \geq w_j$.*

Como consequência desta última definição e do que foi enunciado acima, pode-se concluir que :

$$\underline{a} \leq Ux^n \leq \overline{a} \text{ , para qualquer } n \text{ entre 1 e } N.$$

Lema 6.1.1. *Dado um vetor $a_0 \in \mathbb{R}^{J \times 1}$*

(1) $\forall a_0 : a_0 \leq -\overline{a}$, $ReLU(Ux + a_0) = 0$, ou seja, todos os nodos vão ser ativados.

(2) $\forall a_0 : a_0 \geq -\underline{a}$, $ReLU(Ux + a_0) = Ux + a_0$, nenhum nodo será ativado.

6.2 Estudo com parâmetros fixos

Nesta fase inicial, faremos um estudo dos *autoencoders* ReLU fixando os parâmetros das matrizes de pesos. Considere-se U como sendo a matriz dada pelos J vetores próprios associados aos maiores valores próprios de $X^T X$ (para uma dada base de dados X), e V

como sendo a transposta de U, ou seja, $V = U^T$. Pode-se, pois, ver U e V como sendo :

$$U = \begin{bmatrix} \leftarrow & q_1 & \rightarrow \\ \leftarrow & q_2 & \rightarrow \\ & \vdots & \\ \leftarrow & q_J & \rightarrow \end{bmatrix} \quad V = \begin{bmatrix} \leftarrow & q_1 & \rightarrow \\ \leftarrow & q_2 & \rightarrow \\ & \vdots & \\ \leftarrow & q_J & \rightarrow \end{bmatrix}^T$$

Dado que se está a considerar U e V como fixos, então, a função que se quer minimizar apenas vai depender de a e b. Melhor dizendo, a *loss function* será dada pela seguinte expressão.

$$\mathcal{K}(X; a, b) = \frac{1}{N \times I} \sum_{n=1}^{N} \|x^n - \hat{z}^n\|^2$$

$$\text{sendo, } \hat{z}^n = \text{ReLU}(V\text{ReLU}(Ux^n + a) + b)$$

O objetivo mais à frente será encontrar a^* e b^* tal que a função $\mathcal{K}(X; a, b)$ seja minimizada. Isto é:

$$a^*, b^* = \arg\min_{a,b} \mathcal{K}(X; a, b).$$

Nota 6.2.1. *É trivial notar que :*

(1) se $Ux + a \leq 0$ *então* $ReLU(Ux + a) = 0$ *e* $ReLU(VReLU(Ux + a) + b) = ReLU(b)$.

(2) se $Ux + a > 0$ *então* $ReLU(Ux + a) = Ux + a$ *e* $ReLU(VReLU(Ux + a) + b) = ReLU(VUx + Va + b)$.

6.3 Minimização da função $\mathcal{K}(X; a, b)$

Nesta secção, vai-se fazer um estudo acerca da minimização da função $\mathcal{K}(X; a, b)$, assim como o cálculo das derivadas, à semelhança do que foi feito com *autoencoders* lineares na secção 3.2.

Seja s^n um vetor coluna de dimensão J, tal que :

$$s^n = \begin{cases} se \quad [\text{ReLU}(Ux^n + a)]_j > 0 \quad então \quad s_j^n = 1 \\ se \quad [\text{ReLU}(Ux^n + a)]_j \leq 0 \quad então \quad s_j^n = 0 \end{cases}$$

onde , $[\text{ReLU}(Ux^n + a)]_j$ denotada a posição j do vetor $\text{ReLU}(Ux^n + a)$.

Vai-se ainda considerar t^n um vetor coluna de dimensão I, tal que :

$$t^n = \begin{cases} se & [\text{ReLU}(V\text{ReLU}(Ux^n + a) + b)]_i > 0 \quad então \quad t_i^n = 1 \\ se & [\text{ReLU}(V\text{ReLU}(Ux^n + a) + b)]_i \leq 0 \quad então \quad t_i^n = 0 \end{cases}$$

Dados estes dois vetores, vai-se denotar I_{s^n} como sendo a matriz com todas as entradas a 0, excepto na diagonal que tem os elementos de s^n. Analogamente, para I_{t^n}. Resumindo,

$$I_{s^n}(a) = \begin{bmatrix} s_1^n & 0 & \dots & 0 \\ 0 & s_2^n & \ddots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & s_j^n \end{bmatrix}, I_{t^n}(a,b) = \begin{bmatrix} t_1^n & 0 & \dots & 0 \\ 0 & t_2^n & \ddots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & t_i^n \end{bmatrix}$$

É de notar que I_{s^n} depende implicitamente de a e I_{t^n} depende implicitamente de a e b.

Posto isto, é fácil observar os seguintes resultados:

Proposição 6.3.1. *$ReLU(Ux^n + a) = I_{s^n}(Ux^n + a)$*

Proposição 6.3.2. *$ReLU(VReLU(Ux^n + a) + b) = I_{t^n}[VI_{s^n}(Ux^n + a) + b]$*

Teorema 6.3.1. *Seja x^n tal que $Ux^n + a$ não tem nenhuma componente nula. Então, existe uma vizinhança v de a, tal que, se a' está dentro desta vizinhança, $Ux^n + a'$ também não tem nenhuma componente nula, logo $I_{s^n}(a') = I_{s^n}(a)$.*

Relembrando que,

$$\mathcal{K}(X; a, b) = \frac{1}{N \times I} \sum_{n=1}^{N} \|x^n - \hat{z}^n\|^2$$

Seja, então,

$$F(x; a, b) = \|x - \hat{z}\|^2$$

Proposição 6.3.3. *A função $F(x; a, b)$ pode ser escrita da seguinte forma:*

$$F(x; a, b) = x^T x - x^T I_t V I_s U x - x^T I_t V I_s a - x^T I_t b - x^T U^T I_s^T V^T I_t^T x + x^T U^T U x + x^T U^T a + x^T U^T I_s^T V^T b - a^T I_s^T V^T I_t^T x + a^T U x + a^T a + a^T I_s^T V^T b - b^T I_t^T x + b^T V I_s U x + b^T V I_s a + b^T b$$

Demonstração.

$$F(x;a,b) = \|x - \hat{z}\|^2 = \|x - I_t[VI_s(Ux+a)+b]\|^2 =$$
$$(x - I_t[VI_s(Ux+a)+b])^T(x - I_t[VI_s(Ux+a)+b]) =$$
$$(x^T - I_tVI_s(Ux+a)^T - (I_tb)^T)(x - I_t[VI_s(Ux+a)] - I_tb) =$$
$$(x^T - (I_tVI_sUx)^T - (I_tVI_sa)^T - b^TI_t^T)(x - I_tVI_sUx - I_tVI_sa - I_tb) =$$
$$(x^T - x^TU^TI_s^TV^TI_t^T - a^TI_s^TV^TI_t^T - b^TI_t^T)(x - I_tVI_sUx - I_tVI_sa - I_tb)$$

Realizando-se este produto, obtêm-se:

$$F(x;a,b) = x^Tx - x^TI_tVI_sUx - x^TI_tVI_sa - x^TI_tb -$$
$$x^TU^TI_s^TV^TI_t^Tx + x^TU^TI_s^TV^TI_t^TI_tVI_sUx + x^TU^TI_s^TV^TI_t^TI_tVI_sa +$$
$$x^TU^TI_s^TV^TI_t^TI_tb - a^TI_s^TV^TI_t^Tx + a^TI_s^TV^TI_t^TI_tVI_sUx +$$
$$a^TI_s^TV^TI_t^TI_tVI_sa + a^TI_s^TV^TI_t^TI_tb - b^TI_t^Tx + b^TI_t^TI_tVI_sUx +$$
$$b^TI_t^TI_tVI_sa + b^TI_t^TI_tb$$

E notando-se que :

- $U = V^T$
- $UU^T = Id_J = V^TV$
- $I_s = I_s^T$
- $I_sI_s = I_s = I_sI_s^T$
- $I_t = I_t^T$
- $I_tI_t = I_t = I_tI_t^T$

Pode-se simplificar a expressão, e reescrevê-la da seguinte maneira:

$$F(x;a,b) = x^Tx - x^TI_tVI_sUx - x^TI_tVI_sa - x^TI_tb - x^TU^TI_s^TV^TI_t^Tx +$$
$$x^TU^TUx + x^TU^Ta + x^TU^TI_s^TV^Tb - a^TI_s^TV^TI_t^Tx + a^TUx +$$
$$a^Ta + a^TI_s^TV^Tb - b^TI_t^Tx + b^TVI_sUx + b^TVI_sa + b^Tb$$

□

À semelhança do que foi feito em capítulos anteriores, por exemplo, na secção 3.2, vai-se também, neste caso, calcular as derivadas de F em ordem a a e b.

Proposição 6.3.4.

$$\nabla_a F(x;a,b) = 2x^T U^T - 2x^T I_t V I_s + 2b^T V I_s + 2a^T$$

$$\nabla_b F(x;a,b) = -2x^T I_t + 2x^T U^T I_s^T V^T + 2a^T I_s^T V^T + 2b^T$$

Demonstração.

$$\begin{gathered}\nabla_a F(x;a,b) = \\ \nabla_a(-x^T I_t V I_s a + x^T U^T a - a^T I_s^T V^T I_t^T x + a^T U x + a^T a + a^T I_s^T V^T b + b^T V I_s a) = \\ -x^T I_t V I_s + x^T U^T - x^T I_t V I_s + x^T U^T + 2a^T + b^T V I_s + b^T V I_s = \\ 2x^T U^T - 2x^T I_t V I_s + 2b^T V I_s + 2a^T\end{gathered}$$

$$\begin{gathered}\nabla_b F(x;a,b) = \\ \nabla_b(-x^T I_t b + x^T U^T I_s^T V^T b + a^T I_s^T V^T b - b^T I_t^T x + b^T V I_s U x + b^T V I_s a + b^T b) = \\ -x^T I_t + x^T U^T I_s^T V^T + a^T I_s^T V^T - x^T I_t + x^T U^T I_s^T V^T + a^T I_s^T V^T + 2b^T = \\ -2x^T I_t + 2x^T U^T I_s^T V^T + 2a^T I_s^T V^T + 2b^T\end{gathered}$$

□

Nota 6.3.1.

$$\nabla_a \mathcal{K}(X;a^k,b^k)^T = \frac{1}{N \times I} \sum_{n=1}^{N} \nabla_a F(X;a,b)$$

.

Analogamente, para $\nabla_b \mathcal{K}(X;a^k,b^k)^T$.

6.4 Método de *batch*

Com o objetivo de encontrar os parâmetros a^* e b^*, tais que

$$a^*, b^* = \arg\min_{a,b} \mathcal{K}(X; a, b)$$

foi criado um novo método de *batch* que consistirá em ir atualizando os parâmetros a e b durante um certo número de iterações, como se descreve de seguida:

Dados (a^k, b^k), atualizaremos o seu valor para (a^{k+1}, b^{k+1}) da seguinte forma :

$$a^{k+1} = a^k - \eta \nabla_a \mathcal{K}(X; a^k, b^k)^T$$

$$b^{k+1} = b^k - \eta \nabla_b \mathcal{K}(X; a^k, b^k)^T$$

As iterações são executadas até que o valor de $\mathcal{K}(X; a^k, b^k)$ seja tão pequeno quanto se queira.

Nota 6.4.1. *η é um parâmetro chamado learning rate que, em Machine Learning, é um parâmetro de afinação num algoritmo de otimização que determina o tamanho do passo em cada iteração enquanto procura o mínimo da loss function.*

Deve salientar-se que no contexto desta dissertação foi desenvolvida uma implementação deste novo método de batch que não faz uso da biblioteca *tensorflow*. Na secção seguinte analisaremos o comportamento deste novo método de batch em contexto de treino de *autoencoders* ReLU.

6.5 Análise de resultados

Com o objetivo de tentar avaliar o método desenvolvido na secção anterior, apresentaremos um estudo comparativo dos resultados obtidos com este novo método e com a minimização da função apresentada inicialmente na secção 6.1, ou seja, iremos analisar a diferença entre os erros apresentados na minimização de $\mathcal{L}(X; U, V, a, b)$ e $\mathcal{K}(X; a, b)$ para o mesmo número de épocas. É de salientar que a minimização de $\mathcal{L}(X; U, V, a, b)$ é implementada com recurso à biblioteca *tensorflow*, mas a implementação que desenvolvemos no novo método de batch não recorre a esta biblioteca.

De maneira a simplificar a notação, irá ser utilizada a notação U_L para referenciar a matriz U que advém de treinar o *autoencoder* com base na função $\mathcal{L}(X; U, V, a, b)$, e a

notação U_K para o treino do *autoencoder* com base na função $\mathcal{K}(X; a, b)$. Analogamente, para os restantes parâmetros V, a e b.

6.5.1 Base de dados 1

Para iniciar a avaliação dos modelos criados, começar-se-á por realizar os testes com a base de dados usada na subsecção 4.2.1. Relembrando, esta base de dados foi construída por um gerador de tipo 3, o que significa que é uma base de dados com característica igual a 3, por consequência, não tem característica máxima.

Realizando o treino dos dois métodos, obtêm-se os seguintes resultados:

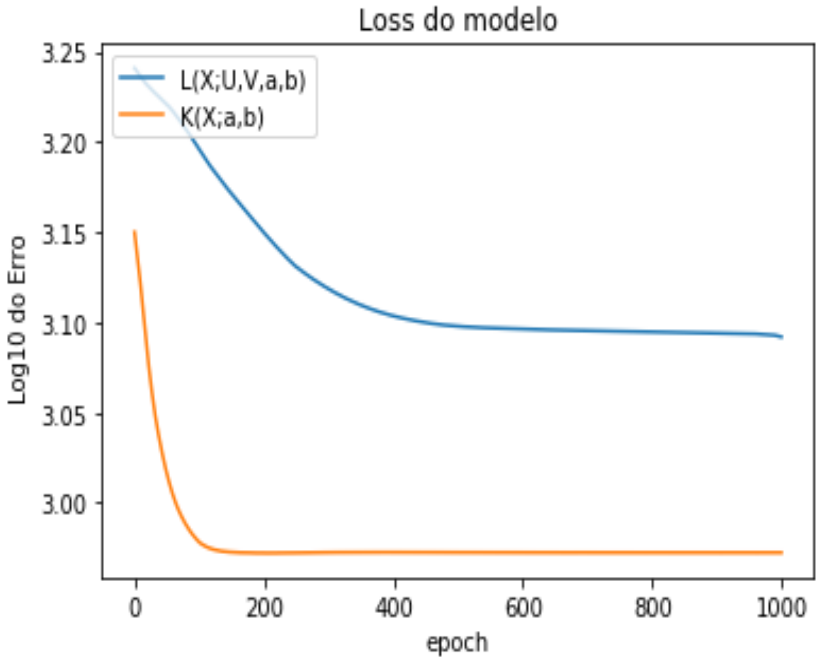


Figura 6.1: Diferença entre a função $\mathcal{K}$ e $\mathcal{L}$ para a base de dados 1 com 1000 épocas

O valor de *loss* no caso $\mathcal{L}(X; U, V, a, b)$ foi de 1452.5328, no entanto, o valor de *loss* no caso $\mathcal{K}(X; a, b)$ já foi de 938.6169. Ou seja, o valor de *loss* $\mathcal{K}(X; a, b)$ foi mais baixo do que $\mathcal{L}(X; U, V, a, b)$, como pode ser visto na figura 6.1.

Os parâmetros do treino obtidos da minimização de $\mathcal{L}(X; U, V, a, b)$ foram os seguintes:

$$U_L = \begin{bmatrix} -0.6570959 & -0.5552441 & 0.6969805 & -0.53212345 & -0.6195877 \\ 0.93912965 & -0.21971065 & 0.33403978 & -0.007044 & -0.5463419 \\ 0.6183142 & -0.76403135 & -0.14339818 & -0.80865777 & 0.5341533 \end{bmatrix}$$

$$V_L = \begin{bmatrix} -0.6593771 & 0.66795546 & -0.20979258 \\ -0.16941303 & 0.58467025 & -0.05695897 \\ -0.09023509 & 0.41569796 & -0.00228941 \\ 0.26956075 & -0.5803906 & -0.33830804 \\ 0.8853938 & 0.0859249 & 0.05096223 \end{bmatrix}$$

$$a_L = \begin{bmatrix} 0.24410146 & 0.6151206 & -0.13057883 \end{bmatrix}$$

$$b_L = \begin{bmatrix} -5.8776804 \times 10^{-4} & 0 & -0.66992599 & -0.32796109 & -0.031564783 \end{bmatrix}$$

E os parâmetros do treino que resultaram da minimização de $\mathcal{K}(X; a, b)$ foram:

$$U_K = \begin{bmatrix} -0.50647306 & 0.19338666 & -0.37187017 & 0.31245115 & -0.68569198 \\ 0.81585813 & 0.29961839 & -0.41722422 & 0.11012004 & -0.24166476 \\ -0.07533331 & 0.88582225 & 0.45472487 & -0.02221052 & 0.04874227 \end{bmatrix}$$

$$V_K = \begin{bmatrix} -0.50647306 & 0.81585813 & -0.07533331 \\ 0.19338666 & 0.29961839 & 0.88582225 \\ -0.37187017 & -0.41722422 & 0.45472487 \\ 0.31245115 & 0.11012004 & -0.02221052 \\ -0.68569198 & -0.24166476 & 0.04874227 \end{bmatrix}$$

$$a_k = \begin{bmatrix} 28.45276908 & 28.77374178 & 4.11855453 \end{bmatrix}$$

$$b_k = \begin{bmatrix} 13.47738735 & -18.31448925 & 45.55492403 & -8.30466096 & 65.72968447 \end{bmatrix}$$

Realizando os testes com 10000 épocas, obtêm-se os seguintes resultados:

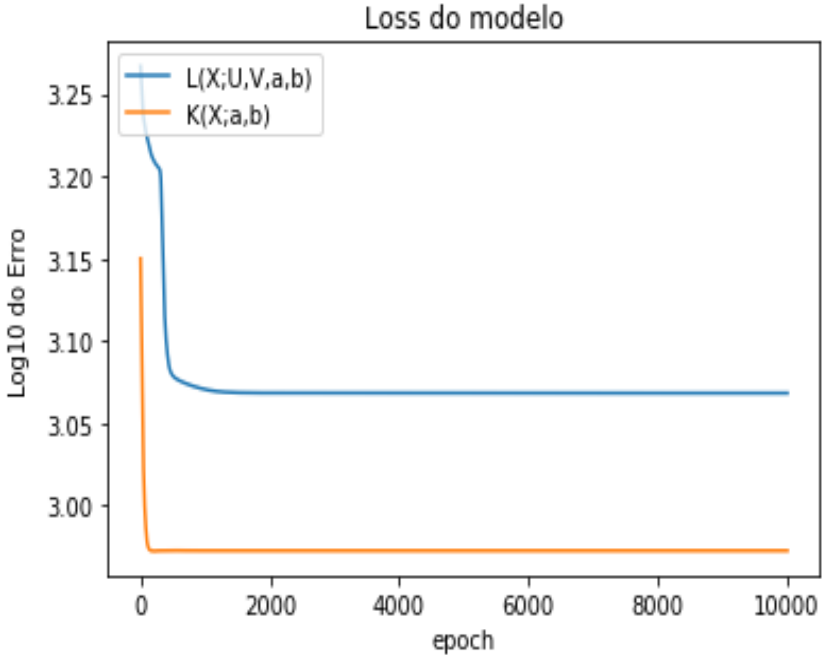


Figura 6.2: Diferença entre a função $\mathcal{K}$ e $\mathcal{L}$ para a base de dados 1 com 10000 épocas

O valor de *loss* no caso $\mathcal{L}(X; U, V, a, b)$ foi de 1448.1264. No entanto, o valor de de *loss* no caso $\mathcal{K}(X; a, b)$ já foi de 938.6169.

Ou seja, o valor de $\mathcal{K}(X; a, b)$ manteve-se exatamente igual e o valor de *loss* de $\mathcal{L}(X; U, V, a, b)$ baixou ligeiramente. Por conseguinte, é possível concluir que ao usar mais épocas não se está a melhorar os resultado. Através da análise da figura 6.2, também facilmente se observa tal facto.

6.5.2 Base de dados 2

O segundo teste será realizado com a base de dados já usada na subsecção 4.2.2. Relembre-se que esta base de dados tem característica 5, ou seja, tem característica máxima.

Realizando o treino dos dois métodos obtêm-se os seguintes resultados:

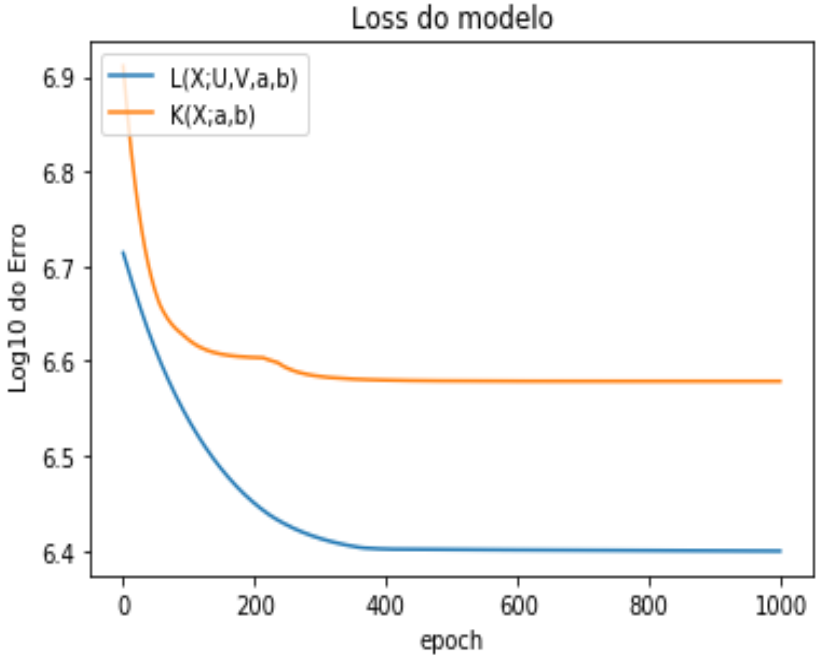


Figura 6.3: Diferença entre a função $\mathcal{K}$ e $\mathcal{L}$ para a base de dados 2 com 1000 épocas

Fazendo uso da figura 6.3, pode-se observar que o valor de *loss* no caso $\mathcal{L}(X; U, V, a, b)$ foi mais baixo do que o valor de *loss* no caso $\mathcal{K}(X; a, b)$. Mais concretamente, o valor de *loss* de $\mathcal{L}(X; U, V, a, b)$ foi de 2608279.21257 e a *loss* no caso $\mathcal{K}(X; a, b)$ já foi de 3789160.8234.

Os parâmetros do treino no caso de $\mathcal{L}(X; U, V, a, b)$ foram:

$$U_L = \begin{bmatrix} 0.7674575 & -0.09670337 & 0.5321901 & -0.34807503 & -0.5536555 \\ 0.6371132 & -0.09147482 & -0.7344761 & 0.15283301 & -0.32846928 \\ -0.08492498 & 0.5717168 & 0.14574333 & -0.7275872 & 0.17387635 \end{bmatrix}$$

$$V_L = \begin{bmatrix} 0.7955367 & 0.6534469 & 0.31988853 \\ -0.15936309 & 0.28150356 & 0.62413 \\ 0.46764672 & -0.48973003 & -0.24865521 \\ 0.5438202 & -1.220811 & 0.49238858 \\ -0.5068068 & 0.14074424 & -0.55475557 \end{bmatrix}$$

$$a_L = \begin{bmatrix} -0.05499759 & 0.02808622 & -0.18498442 \end{bmatrix}$$

$$b_L = \begin{bmatrix} -0.04715861 & -0.20735447 & -0.19545439 & -0.11393943 & 0 \end{bmatrix}$$

E os parâmetros do treino de $\mathcal{K}(X; a, b)$ foram:

$$U_K = \begin{bmatrix} -0.999469534 & -0.0320246049 & -0.00546862208 & 0.00207247522 & 0.000935036931 \\ -0.0301790376 & 0.977396341 & -0.209208747 & -0.00382180281 & 0.00164703606 \\ 0.0119519557 & -0.208910078 & -0.976962679 & -0.0419221561 & -0.0004.32247895 \end{bmatrix}$$

$$V_K = \begin{bmatrix} -0.999469534 & -0.0301790376 & 0.0119519557 \\ -0.0320246049 & 0.977396341 & -0.208910078 \\ -0.00546862208 & -0.209208747 & -0.976962679 \\ 0.00207247522 & -0.00382180281 & -0.0419221561 \\ 0.000935036931 & 0.00164703606 & -0.000432247895 \end{bmatrix}$$

$$a_K = \begin{bmatrix} 2485.94644702 & 307.37365236 & 344.87794478 \end{bmatrix}$$

$$b_K = \begin{bmatrix} 6410.33942 & -0.217550238 & 893.668072 & 32.6670213 & -0.957897204 \end{bmatrix}$$

Realizando os testes com 10000 épocas, obtêm-se os seguintes resultados:

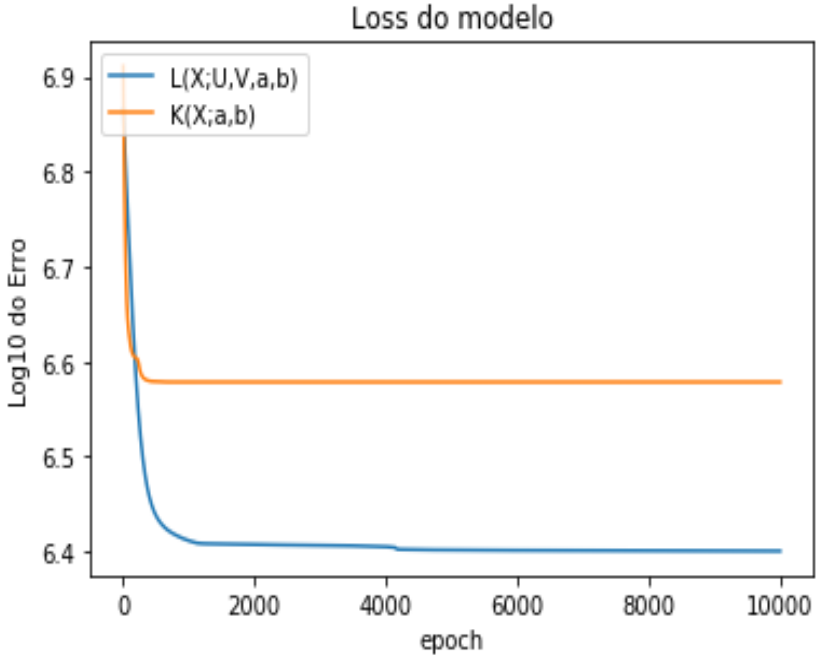


Figura 6.4: Diferença entre a função $\mathcal{K}$ e $\mathcal{L}$ para a base de dados 2 com 10000 épocas

O valor de *loss* no caso $\mathcal{L}(X;U,V,a,b)$ foi de 2515640.5, porém o valor de de *loss* no caso $\mathcal{K}(X;a,b)$ já foi de 3789148.780126454.

Ou seja, o valor de $\mathcal{K}(X;a,b)$ manteve-se bastante semelhante e o valor de *loss* de $\mathcal{L}(X;U,V,a,b)$ baixou ligeiramente. Assim, mais uma vez, é possível concluir que treinar apenas 1000 épocas é suficiente para tirar as conclusões necessárias, tal como a figura 6.4 ilustra.

6.5.3 Base de dados 3

O último teste será realizado com uma base de dados regular. (Recorde-se que a definição de base de dados regular foi dada em 2.1.15.)

A base de dados X que consideraremos é constituída por 15 linhas, ou seja, $N = 15$, e é a seguinte:

$$X = \begin{bmatrix} 6.97748325 & -9.15479954 & 3.80844915 & -0.27328328 & -1.40149557 \\ -3.98592679 & -5.20502529 & -1.33214202 & -0.91570085 & -4.6960453 \\ -8.63231429 & -1.9581832 & 5.18222074 & -0.26760086 & -1.37235406 \\ 11.05137686 & -2.8772163 & 2.7281249 & 0.5575807 & 2.85947557 \\ -2.34053282 & -6.73024395 & 5.79087608 & -0.37512187 & -1.92376069 \\ -8.17935306 & -2.12445693 & 3.23143023 & -0.42749681 & -2.19235831 \\ 5.31730211 & 3.67397834 & 5.18538079 & 1.15619919 & 5.92940778 \\ -10.27028801 & 3.37891818 & -0.79023415 & -0.29071883 & -1.49091134 \\ 1.06594102 & -0.89488144 & -11.67444526 & -1.04092302 & -5.3382299 \\ 5.78974715 & -7.38569808 & 3.76588502 & -0.1515418 & -0.77716116 \\ 1.21278012 & -4.87153967 & -0.27138671 & -0.48754814 & -2.50032328 \\ 3.33272486 & 1.05339904 & -11.15168796 & -0.6513333 & -3.34027285 \\ -8.71646574 & 6.654388 & -2.76736884 & -0.01183037 & -0.06067043 \\ 4.85283557 & -0.42982791 & -0.24521026 & 0.21210672 & 1.08776 \\ -0.48795224 & 1.05719903 & -2.51717976 & -0.12879988 & -0.66053237 \end{bmatrix}$$

Realizando o treino dos dois métodos, obtêm-se os seguintes resultados:

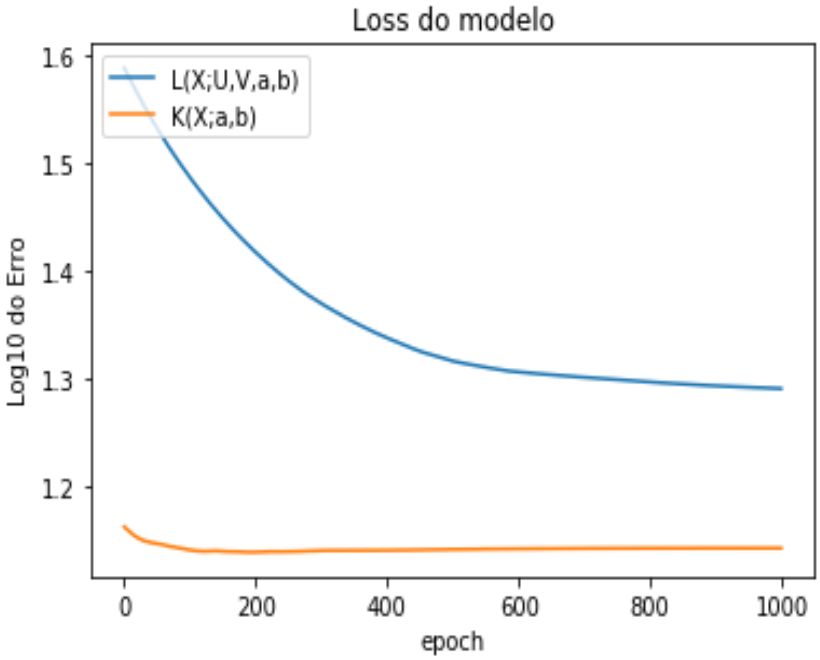


Figura 6.5: Diferença entre a função $\mathcal{K}$ e $\mathcal{L}$ para a base de dados 3 com 1000 épocas

O valor de *loss* no caso $\mathcal{L}(X; U, V, a, b)$ foi de 23.2298. Contudo, o valor de *loss* no caso $\mathcal{K}(X; a, b)$ já foi de 13.8920, como pode ser observado na figura 6.5.

Os parâmetros do treino de $\mathcal{L}(X; U, V, a, b)$ foram :

$$U_L = \begin{bmatrix} 0.5790286 & -0.08122768 & -0.55172837 & 0.01311023 & 0.03716916 \\ -0.28968653 & -0.4797359 & 0.27514476 & -0.7332698 & 0.01031636 \\ -0.293517 & 0.11625956 & -0.44947687 & 0.203451 & 0.3994666 \end{bmatrix}$$

$$V_L = \begin{bmatrix} -0.13455214 & -0.56608754 & 0.344792 \\ 0.49305156 & -0.07427498 & 0.01979682 \\ 0.3352104 & -0.34669334 & -0.16959263 \\ -0.19709598 & -0.48187453 & 0.14850554 \\ 0.48264298 & 0.42033428 & -0.10404934 \end{bmatrix}$$

$$a_L = \begin{bmatrix} -0.31268337 & -0.23728213 & -0.41552913 \end{bmatrix}$$

$$b_L = \begin{bmatrix} -0.35857457 & -0.30928373 & -0.21050759 & -0.17350762 & -0.50681025 \end{bmatrix}$$

E os parâmetros do treino de $\mathcal{K}(X; a, b)$ foram :

$$U_K = \begin{bmatrix} 0.87181902 & -0.38771775 & 0.255568 & 0.02982946 & 0.15297628 \\ -0.39968066 & -0.28763387 & 0.86423881 & 0.01971721 & 0.10111699 \\ -0.15492277 & -0.75129536 & -0.25133093 & -0.11296629 & -0.57933202 \end{bmatrix}$$

$$V_K = \begin{bmatrix} 0.87181902 & -0.39968066 & -0.15492277 \\ -0.38771775 & -0.28763387 & -0.75129536 \\ 0.255568 & 0.86423881 & -0.25133093 \\ 0.02982946 & 0.01971721 & -0.11296629 \\ 0.15297628 & 0.10111699 & -0.57933202 \end{bmatrix}$$

$$a_K = \begin{bmatrix} 2.86847226 & 2.19299594 & 4.49713384 \end{bmatrix}$$

$$b_K = \begin{bmatrix} 0.79171175 & 8.49652229 & -0.88918568 & 0.61945086 & 3.17676816 \end{bmatrix}$$

6.5.4 Conclusões

Apresentados os diversos testes realizados com as implementações que desenvolvemos dos dois métodos de treino para *autoencoders* ReLU, baseados nas funções de *loss* $\mathcal{K}(X; a, b)$ e $\mathcal{L}(X; U, V, a, b)$, podemos, agora, proceder a uma comparação dos dois métodos.

Ora, é fácil notar que $\mathcal{L}(X; U, V, a, b)$ tem mais parâmetros para minimizar do que $\mathcal{K}(X; a, b)$, pois tem como objetivo, encontrar matrizes U e V, e ainda vetores a e b que minimizem esta função. Porém, $\mathcal{K}(X; a, b)$ apenas irá ter que minimizar dois parâmetros que também se encontram em $\mathcal{L}(X; U, V, a, b)$, sendo estes, os vetores a e b. Aqui, trivialmente, consegue-se observar que minimizar $\mathcal{L}(X; U, V, a, b)$ será um processo mais custoso, tanto a nível de tempo como de esforço computacional, do que minimizar $\mathcal{K}(X; a, b)$.

Recorde-se, contudo, que o objetivo inicial é, de facto, minimizar $\mathcal{L}(X; U, V, a, b)$, ou seja, encontrar U, V, a, b que minimizem esta função. A ideia na base do novo método que desenvolvemos passa por substituir a minimização de $\mathcal{L}(X; U, V, a, b)$ pela minimização da função $\mathcal{K}(X; a, b)$, porque parece ser razoável utilizar as matrizes U e V mencionadas em 6.2 como aproximação das matrizes que vão ser retornadas ao minimizar $\mathcal{L}(X; U, V, a, b)$, e, assim, transformar o problema inicial num problema que apenas depende de a e b. Assim, importa perceber se a função $\mathcal{K}(X; a, b)$ pode ser uma alternativa efetiva ao uso de $\mathcal{L}(X; U, V, a, b)$, por exemplo, em situações em que a utilização desta última não seja viável. Por outras palavras, importa perceber se a minimização de $\mathcal{K}(X; a, b)$ é suficiente em termos de qualidade de resultados em relação à minimização de $\mathcal{L}(X; U, V, a, b)$.

Sem perder de vista as considerações anteriores, passemos à comparação dos resultados obtidos e algumas conclusões.

Pode-se observar que em ambas as bases de dados o método baseado em $\mathcal{K}(X; a, b)$ convergiu sempre mais rápido que o método baseado em $\mathcal{L}(X; U, V, a, b)$. No entanto, falando da qualidade da solução, torna-se mais complicado tirar conclusões pelo facto de que na Base de dados 2, o método baseado em $\mathcal{L}(X; U, V, a, b)$ apresenta melhores resultados, mas já na Base de dados 1 e na Base de dados 3, o método baseado em $\mathcal{K}(X; a, b)$ apresentou a melhor solução. Estas observações levam-nos a concluir que a eficácia de cada um dos métodos poderá depender imenso da base de dados que estamos a utilizar. Contudo,

é de salientar que devido ao facto do método baseado em $\mathcal{K}(X;a,b)$ ser mais eficiente, se utilizarmos bases de dados muito grandes será, provavelmente, muito mais proveitoso usar este método. Por outro lado, em bases de dados pequenas, como as utilizadas nos exemplos que considerámos, podemos utilizar o método baseado em $\mathcal{L}(X;U,V,a,b)$ como *loss function*, visto que a qualidade da solução poderá ser melhor, pois a Base de dados 2 é uma base mais genérica, pois tem característica máxima e não é regular.

É de referir que por vezes é possível que não se consiga averiguar a qualidade da solução com grande exatidão, na medida em que não conseguimos obter um mínimo global, mas sim um mínimo local nas funções de *loss* utilizadas.

Terminado este estudo comparativo dos métodos baseados em $\mathcal{L}(X;U,V,a,b)$ e $\mathcal{K}(X;a,b)$ como funções de *loss*, regressaremos ao problema inicial que utiliza a função $\mathcal{L}(X;U,V,a,b)$ como função de *loss* para continuar o seu estudo.

6.6 Valor mínimo da função de *loss* $\mathcal{L}(X;U,V,a,b)$

Nesta secção, continuaremos o estudo da função de *loss* $\mathcal{L}(X;U,V,a,b)$, embora, fixando os parâmetros das matrizes e dos vetores *bias* de maneira a procurar minimizá-la, assim como, averiguar qual o valor mínimo que esta vai apresentar em certos casos particulares.

Comecemos por relembrar que esta função de *loss* é dada pela seguinte expressão:

$$\mathcal{L}(X;U,V,a,b) = \frac{1}{N \times I} \sum_{n=1}^{N} \|x^n - \mathrm{ReLU}(V\mathrm{ReLU}(Ux^n + a) + b)\|^2$$

O primeiro caso particular que abordaremos para a minimização desta função é o caso das bases de dados regulares (conforme a definição 2.1.15).

Consideremos uma base de dados X regular. Sabemos que será sempre possível encontrar matrizes U e V, tais que $VUx = x$, $\forall x \in X$

Consideremos adicionalmente que $a = -\underline{a}$ e $b = -V\underline{a} + c$, onde, $c \in \mathbb{R}^{I\times 1}$ e, como atrás, $\underline{a}$ denota $\min Ux$.

Assim, tendo tomado $a = -\underline{a}$, é trivial notar que $Ux - \underline{a} \geq 0$. Logo:

$$\mathcal{L}(X; U, V, a, b) = \frac{1}{N \times I} \sum_{n=1}^{N} \|x^n - \text{ReLU}(V\text{ReLU}(Ux^n + a) + b)\|^2 =$$

$$= \frac{1}{N \times I} \sum_{n=1}^{N} \|x^n - \text{ReLU}(VUx^n - V\underline{a} + b)\|^2 =$$

$$= \frac{1}{N \times I} \sum_{n=1}^{N} \|x^n - ReLU(x^n + c)\|^2$$

Podemos, agora, estabelecer a seguinte proposição:

Proposição 6.6.1. *Supondo as condições acima para as matrizes U e V e para os vetores bias a e b, tem-se:*

$$\mathcal{L}(X; U, V, c) = \sum_{i=1}^{I} (\frac{1}{N \times I} \sum_{n=1}^{N} \min(-c_i, x_i^n)^2)$$

Demonstração. A demonstração faz uso das seguintes propriedades elementares:

(1) $ReLU(s) = \max(s, 0)$

(2) $-\max(a, b) = \min(-a, -b)$

(3) $\max(a, b) + c = \max(a + c, b + c)$

A demonstração segue, então, do seguinte modo:

$$\mathcal{L}(X;U,V,c) = \frac{1}{N \times I}\sum_{n=1}^{N} \|x^n - ReLU(x^n + c)\|^2 =$$

$$= \frac{1}{N \times I}\sum_{n=1}^{N}\sum_{i=1}^{I}(x_i^n - ReLU(x_i^n + c_i))^2 =$$

$$= \sum_{i=1}^{I}(\frac{1}{N \times I}\sum_{n=1}^{N}(x_i^n - ReLU(x_i^n + c_i))^2) =$$

$$= \sum_{i=1}^{I}(\frac{1}{N \times I}\sum_{n=1}^{N}(x_i^n - \max(x_i^n + c_i, 0))^2) =$$

$$= \sum_{i=1}^{I}(\frac{1}{N \times I}\sum_{n=1}^{N}(x_i^n + \min(-(x_i^n + c_i), 0))^2) =$$

$$= \sum_{i=1}^{I}(\frac{1}{N \times I}\sum_{n=1}^{N}\min(-c_i, x_i^n)^2)$$

□

Uma das consequências importantes deste resultado é a de que, para $\mathcal{L}(X;U,V,c)$ ser 0, tem que se ter:

$$\sum_{i=1}^{I}\sum_{n=1}^{N}\min(-c_i, x_i^n)^2 = 0$$

o que, por sua vez, que seja necessário ter-se, para todo o i, n, $\min(-c_i, x_i^n) = 0$.

No entanto, note-se que, se todos os elementos da base de dados forem positivos, ou seja, $x_i^n \geq 0$, basta tomar $c_i = 0$. Logo, o problema apenas reside em descobrir qual o melhor c_i a tomar, quando existem valores negativos, melhor dizendo, quando existem elementos tais que, $x_i^n < 0$.

Obtém-se, então, o seguinte corolário.

Corolário 6.6.1. *Se todos os elementos da base de dados forem positivos, ou seja, $x_i^n \geq 0$, então, $c = 0$ faz com que $\mathcal{L}(X;U,V,c) = 0$.*

Fazendo uso deste corolário pode-se dizer que, neste caso específico, $c = 0$ é um minimizante, o que implica que o problema do *autoencoder* não linear com a função de ativação ReLU, torna a ser o problema de minimização linear, pois a função ReLU nunca é ativada.

6.7 Validação numérica

Nesta secção, desenvolveremos um exemplo prático, de maneira a realizar uma verificação numérica do resultado enunciado no corolário 6.6.1.

Supondo que temos uma base de dados em que todos os seus elementos são positivos, ou seja, $x_i^n \geq 0$, $\forall n \in N$ e $\forall i \in I$, à luz do referido corolário, é, então, possível encontrar parâmetros que produzam uma *loss* de 0 para a função $\mathcal{L}(X; U, V, c)$, bastando para tal tomar o vetor c como sendo o vetor nulo.

Na nossa exemplificação, serão considerados três vetores linearmente independentes, por exemplo, $v_1 = (1, 2, 3, 4, 5), v_2 = (2, 3, 4, 5, 1)$e $v_3 = (3, 4, 5, 1, 2)$ e cada um dos elementos, x^n, da base de dados X, será construído como uma combinação linear destes três vetores, isto é,

$$x^n = \lambda_1 v_1 + \lambda_2 v_2 + \lambda_3 v_3$$

com $\lambda_i \in \mathbb{R}^+$.

Para tal, pode-se considerar a seguinte base de dados que foi construída da maneira descrita acima:

$$X = \begin{bmatrix} 247.03579458 & 371.10420807 & 495.17262157 & 380.0790164 & 367.63456181 \\ 63.75037416 & 117.99068984 & 172.23100551 & 209.10623407 & 250.52643157 \\ 222.21747418 & 329.55006227 & 436.88265037 & 328.44693098 & 292.89170365 \\ 133.89087244 & 211.04217609 & 288.19347974 & 283.14345349 & 240.99957306 \\ 139.53358086 & 214.90336426 & 290.27314766 & 253.97834396 & 231.85831424 \\ 140.90681577 & 231.75153847 & 322.59626116 & 403.31323868 & 264.10298636 \\ 27.39009921 & 43.73564868 & 60.08119815 & 72.94671333 & 41.02958263 \\ 142.01857256 & 211.25224186 & 280.48591116 & 206.21392845 & 198.53438551 \\ 119.492675 & 188.24237512 & 256.99207524 & 294.00813208 & 172.51024429 \\ 175.40980402 & 243.85546877 & 312.30113352 & 171.21782995 & 123.90073499 \\ 227.43313292 & 334.08250313 & 440.73187333 & 317.69057396 & 279.80246981 \\ 233.20712529 & 350.69665538 & 468.18618547 & 395.68944201 & 314.5635432 \\ 128.46980517 & 199.94258934 & 271.41537351 & 270.56718414 & 201.69681041 \\ 95.56408813 & 159.012593 & 222.46109787 & 262.66424941 & 212.02554461 \\ 220.09858323 & 334.75770573 & 449.41682823 & 368.49824579 & 347.11547449 \end{bmatrix}$$

Tomemos, então, c como sendo o vetor nulo, ou seja, $c_i = 0$, $\forall i \in I$.

Considerem-se de novo os vetores $v_1 = (1, 2, 3, 4, 5), v_2 = (2, 3, 4, 5, 1)$ e $v_3 = (3, 4, 5, 1, 2)$

e considere-se que o resultado da aplicação da transformação de *Gram-Schmidt*, vista no exemplo 2.1.2 produz os seguintes vetores ortogonais:

$$q_1 = (0.13483997, 0.26967994, 0.40451992, 0.53935989, 0.67419986)$$

$$q_2 = (0.27716093, 0.31980107, 0.36244122, 0.40508136, -0.72488244)$$

$$q_3 = 0.35133989, 0.38706937, 0.42279885, -0.73245434, 0.10718844)$$

.

Considere-se agora que a matriz U é construída com estes vetores:

$$U = \begin{bmatrix} 0.13483997 & 0.26967994 & 0.40451992 & 0.53935989 & 0.67419986 \\ 0.27716093 & 0.31980107 & 0.36244122 & 0.40508136 & -0.72488244 \\ 0.35133989 & 0.38706937 & 0.42279885 & -0.73245434 & 0.10718844 \end{bmatrix}$$

e que a matriz V será U^T, ou seja :

$$V = \begin{bmatrix} 0.13483997 & 0.27716093 & 0.35133989 \\ 0.26967994 & 0.31980107 & 0.38706937 \\ 0.40451992 & 0.36244122 & 0.42279885 \\ 0.53935989 & 0.40508136 & -0.73245434 \\ 0.67419986 & -0.72488244 & 0.10718844 \end{bmatrix}$$

Além disso, considere-se ainda $a = -\underline{a}$ e $b = V\underline{a} + c = V\overline{a}$, ou seja:

$$a = \begin{bmatrix} -106.79861911 & -20.92910585 & -2.92198567 \end{bmatrix}$$

e

$$b = \begin{bmatrix} 21.22806345 & 36.62560741 & 52.02315137 & 63.94066105 & 57.14567618 \end{bmatrix}$$

Dados estes parâmetros, o valor de *loss* é 2.113×10^{-27}.

Pode-se, assim, concluir que a função de minimização, para esta base de dados em específico, dá um erro praticamente nulo.

No entanto, se passarmos esta base de dados pelo *tensorflow*, sem lhe fornecer os parâmetros mencionados anteriormente, isto é, se executarmos o treino de raiz, o resultado da *loss* já será muito distinto.

Treinando-se o modelo com esta base de dados para 10000 épocas, obtêm-se os seguintes resultados:

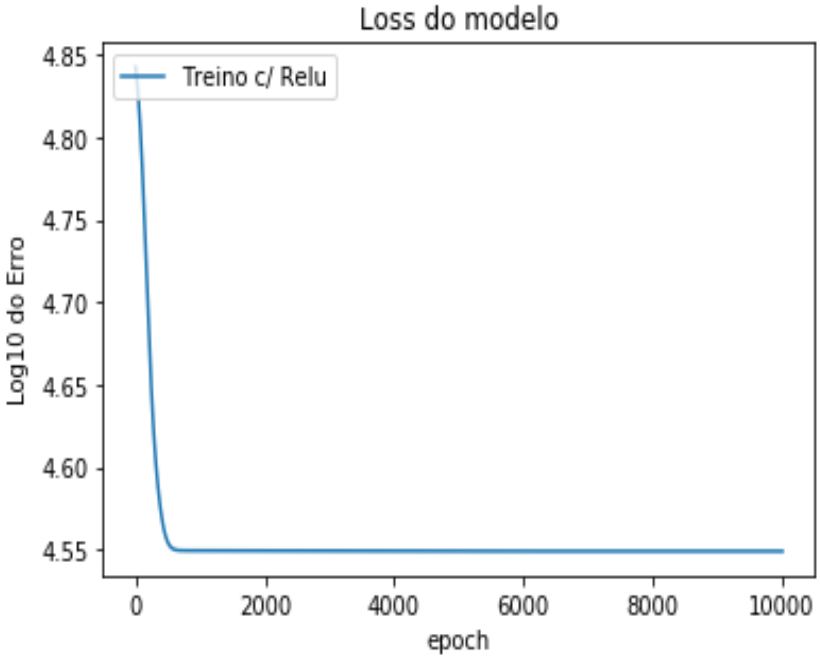


Figura 6.6: Treino de $\mathcal{L}(X; U, V, a, b)$ com ReLU

Neste caso, obtém-se um valor de *loss* de 35406.5, além disso, é possível verificar, através da figura 6.6, que o modelo parou de aprender por volta da época 500.

Esta constatação faz-nos pensar que muitas vezes o *tensorflow* poderá ficar bloqueado num mínimo local e, de facto, poderá não conseguir encontrar sempre os melhores parâmetros, como sucedeu neste caso, em que sabemos teoricamente que é possível atingir o valor mínimo 0 para esta função.

De maneira a tentar resolver este problema, vão-se empregar duas abordagens. A primeira vai passar por testar como se comporta a aprendizagem com outra função de ativação e, para tal, recorreremos à função de ativação ELU, que foi introduzida na secção 5.2.6. A outra abordagem vai passar por introduzir ao método de *enconde* e *decode* do *autoencoder* um *kernel regularizer*, cujos conceitos podem ser revistos no capítulo 5.3. É de salientar que nesta última abordagem a função de ativação volta a ser a função ReLU.

6.7.1 Valor mínimo com a função de ativação Elu

Como mencionado na última secção, sabe-se que é possível obter uma solução ótima "manualmente", para uma base de dados cujos valores são todos positivos, no entanto, se o *tensorflow* tentar encontrar por ele mesmo uma solução, o valor da mesma não era nula, mas sim um valor de *loss* bastante alto. Para tentar resolver este problema, faremos uso da função de ativação ELU, de maneira a tentar observar se o modelo com tal mudança vai conseguir obter resultados mais satisfatórios.

Correndo o modelo para a base de dados X usada acima, ou seja,

$$X = \begin{bmatrix} 247.03579458 & 371.10420807 & 495.17262157 & 380.0790164 & 367.63456181 \\ 63.75037416 & 117.99068984 & 172.23100551 & 209.10623407 & 250.52643157 \\ 222.21747418 & 329.55006227 & 436.88265037 & 328.44693098 & 292.89170365 \\ 133.89087244 & 211.04217609 & 288.19347974 & 283.14345349 & 240.99957306 \\ 139.53358086 & 214.90336426 & 290.27314766 & 253.97834396 & 231.85831424 \\ 140.90681577 & 231.75153847 & 322.59626116 & 403.31323868 & 264.10298636 \\ 27.39009921 & 43.73564868 & 60.08119815 & 72.94671333 & 41.02958263 \\ 142.01857256 & 211.25224186 & 280.48591116 & 206.21392845 & 198.53438551 \\ 119.492675 & 188.24237512 & 256.99207524 & 294.00813208 & 172.51024429 \\ 175.40980402 & 243.85546877 & 312.30113352 & 171.21782995 & 123.90073499 \\ 227.43313292 & 334.08250313 & 440.73187333 & 317.69057396 & 279.80246981 \\ 233.20712529 & 350.69665538 & 468.18618547 & 395.68944201 & 314.5635432 \\ 128.46980517 & 199.94258934 & 271.41537351 & 270.56718414 & 201.69681041 \\ 95.56408813 & 159.012593 & 222.46109787 & 262.66424941 & 212.02554461 \\ 220.09858323 & 334.75770573 & 449.41682823 & 368.49824579 & 347.11547449 \end{bmatrix}$$

e usando, então, como função de ativação a função ELU, obtêm-se os seguintes resultados:

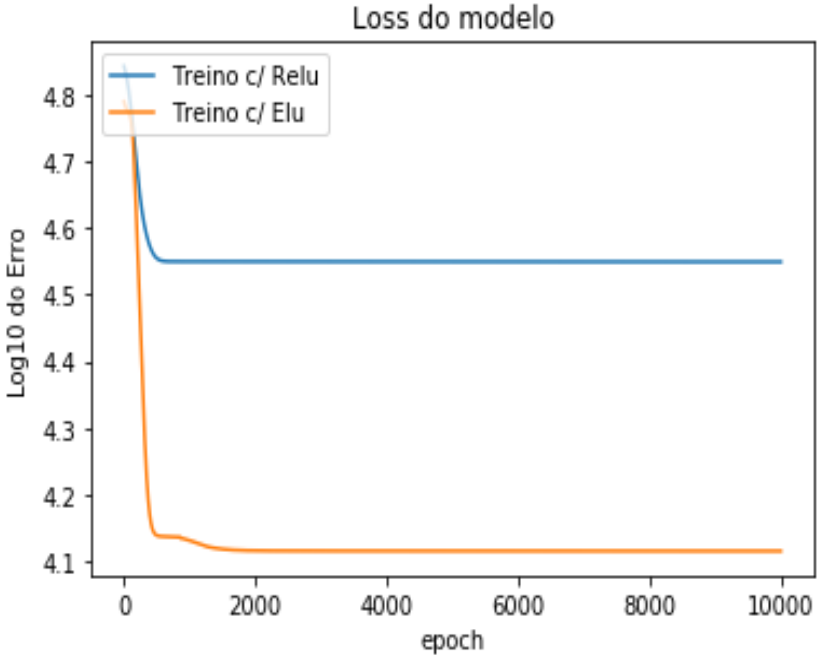


Figura 6.7: Comparação do treino usando ReLU e *Elu*

O valor de *loss* com a função ReLU foi de 35406.5, como visto na secção anterior. Contudo, usando a função ELU, o valor de *loss* diminuiu para 13047.4238. Em suma, observando-se a figura 6.7, pode-se concluir que se registou uma melhoria bastante significativa. Todavia, ainda se está longe de atingir o valor de *loss* 0, que é possível obter, através da solução encontrada "manualmente".

Olhando-se para o gráfico, consegue-se ainda observar que não será relevante aumentar o número de épocas pois o modelo parece ter parado de aprender.

De seguida, vai-se experimentar a segunda abordagem mencionada, que passa por tentar utilizar regularizadores de maneira a solucionar este problema.

6.7.2 Valor mínimo usando regularizadores

Dado que na última subsecção ainda não se conseguiu solucionar o problema da obtenção de um valor de *loss* nulo, para a base de dados cujos valores eram todos positivos, então, com o intuito de tentar resolver este problema, vai-se adicionar às camadas do modelo regularização. Para tal, tanto na camada de *enconde* como de *decode*, vai ser adicionado *kernel regularizer L1* e *L2*, de forma a observar se o modelo com tal adição vai conseguir obter resultados mais satisfatórios. Como mencionado acima, neste caso, a função de ativação

voltou a ser a função ReLU.

Usando regularizadores *L1* e *L2* com valor de penalização 10^{-4}, obtém-se um valor de *loss* de 180.3340 ao fim de 10000 épocas.

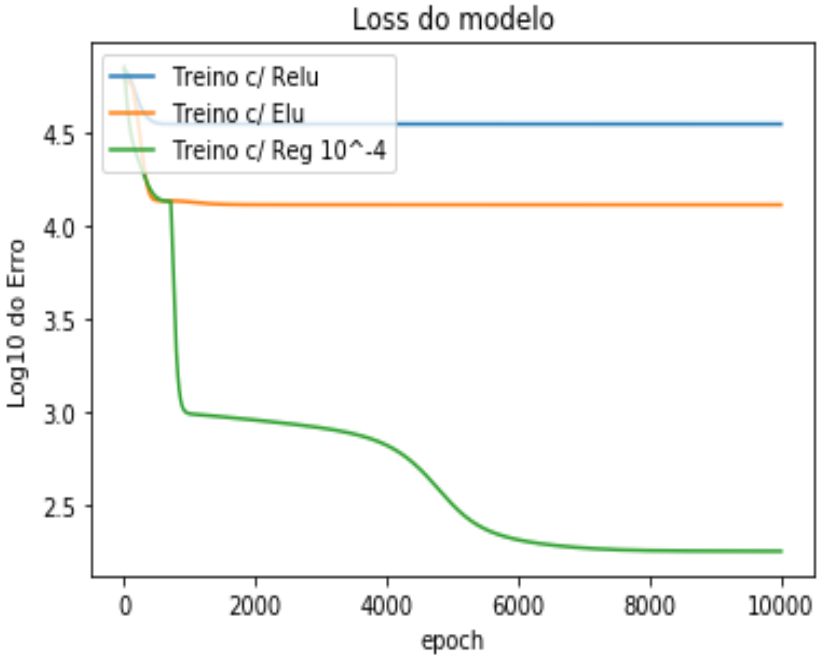


Figura 6.8: Comparação do treino usando ReLU, ELU e regularizadores

No entanto, ainda não se obtém o valor zero, o que se pode ficar a dever ao facto de o valor da penalização não ser suficientemente alto. Para tal, vai-se realizar outro teste com um valor de penalização de 0.01.

Com este valor, já se consegue obter um valor de *loss* de 0.2322.

A figura 6.9 apresenta o gráfico usando regularizadores com penalizações distintas.

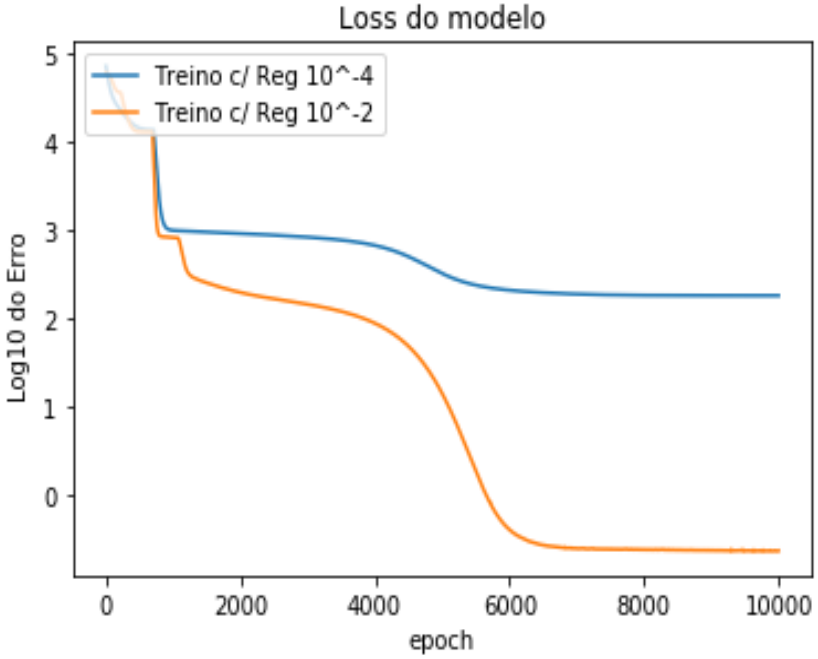


Figura 6.9: Comparação do treino usando valores diferentes de regularizadores

Realizando-se agora a comparação com os resultados anteriores, obtém-se o seguinte gráfico ilustrado na figura 6.10:

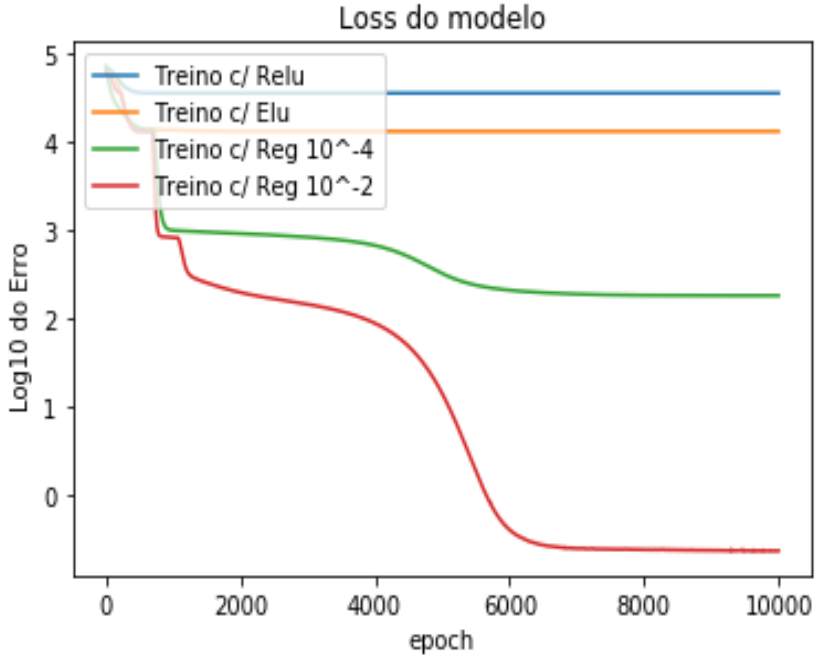


Figura 6.10: Comparação do treino usando todas as abordagens testadas

Esta última solução apresentada já pode ser considerada uma solução aceitável, dado que regista um valor muito próximo de zero.

Através da figura 6.10, pode-se ainda observar que a reta está a estabilizar e, por conseguinte, não será necessário adicionar mais épocas ao modelo, porque, à partida, já se conseguiu obter a solução mais baixa que este modelo pode alcançar.

É possível concluir que adicionar regularizadores, neste caso concreto, foi bastante eficaz, pois, passou-se, por exemplo, de um valor de *loss* 35406.5, para um valor de *loss*, no nosso último exemplo, de 0.2322, apenas pelo facto de adicionar regularizadores às nossas camadas de *enconde* e *decode*.

6.8 Função $\mathcal{L}(X;U,V,c)$ no caso de dados negativos

Nesta secção, continuaremos o estudo da função $\mathcal{L}(X;U,V,c)$. Na secção 6.6, concluiu-se que:

$$\mathcal{L}(X;U,V,c) = \sum_{i=1}^{I}(\frac{1}{N\times I}\sum_{n=1}^{N}\min(-c_i, x_i^n)^2)$$

Facilmente se conseguiu deduzir que se todos os elementos da base de dados fossem positivos, isto é, $x_i^n > 0$, então, bastava considerar c como sendo o vetor nulo, de modo a encontrar o mínimo da função. Porém, sabemos que em muitas situações os elementos de uma base de dados não são todos nulos. Nesta secção investigaremos qual deverá ser o valor de c a utilizar quando a base de dados apresentar tanto valores positivos como negativos.

Teorema 6.8.1. *Dada uma base de dados regular X, matrizes U e V, tal que, $\forall x \in X$, $VUx = x$,e considerando-se um a, tal que, $a \geq -\underline{a}$ e por fim $b = -V\underline{a}+c$, onde, $c \in \mathbb{R}^{I\times 1}$, tem-se que*

$$\mathcal{L}(X;U,V,c) = \sum_{i=1}^{I}(\frac{1}{N\times I}\sum_{n=1}^{N}\min(-c_i, x_i^n)^2)$$

atinge o mínimo quando $c = 0$.

Demonstração. Supondo que X é uma base de dados regular, considere-se U como sendo a matriz formada pelos vetores próprios associados aos maiores valores próprios de X^TX e $V = U^T$.

É de notar que VU pode ser diferente da identidade, no entanto, fazendo-se uso deste tipo de matrizes mencionadas acima, $VUx = x$, $\forall x \in X$.

Dado isto, vai-se ainda considerar um a, tal que $a \geq -\underline{a} = -\min Ux$.

Por consequência desta última hipótese, $\text{ReLU}(Ux^n + a) = Ux^n + a$, onde $1 \leq n \leq N$.

Assim, tal como na proposição 6.6.1, temos que:

$$\mathcal{L}(X; U, V, c) = \sum_{i=1}^{I} (\frac{1}{N \times I} \sum_{n=1}^{N} \min(-c_i, x_i^n)^2)$$

.

De maneira a simplificar a notação, vai-se denotar $P_i(c_i)$ como sendo:

$$P_i(c_i) = \frac{1}{N \times I} \sum_{n=1}^{N} \min(-c_i, x_i^n)^2 =$$

$$= \frac{1}{N \times I} \sum_{x_i^n \leq -c_i} \min(-c_i, x_i^n)^2 + \sum_{x_i^n > -c_i} \min(-c_i, x_i^n)^2 =$$

$$= \frac{1}{N \times I} \left[\sum_{x_i^n \leq -c_i} (x_i^n)^2 + \sum_{x_i^n > -c_i} (-c_i)^2 \right]$$

Primeira parte:

Vai-se começar por demonstrar que $c_i \geq 0$, para tal, vai-se realizar uma demonstração por redução ao absurdo.

Suponhamos que $c_i < 0$.

É de notar que :

$$P_i(0) = \frac{1}{N \times I} \left[\sum_{x_i^n \leq 0} (x_i^n)^2 + \sum_{x_i^n > 0} (0)^2 \right]$$

Dado que $c_i < 0 \implies -c_i > 0$. Logo, é possível concluir que

$$\sum_{x_i^n \leq -c_i} (x_i^n)^2 \geq \sum_{x_i^n \leq 0} (x_i^n)^2$$

Pois como $-c_i > 0$, existe um número maior de elementos x_i menores que $-c_i$ do que 0.

Além disso, dado que $-c_i > 0$, então :

$$\sum_{x_i^n > -c_i} (-c_i)^2 > \sum_{x_i^n > 0} (0)^2$$

Retomando-se as contas:

$$P_i(c_i) = \frac{1}{N \times I} \left[\sum_{x_i^n \leq -c_i} (x_i^n)^2 + \sum_{x_i^n > -c_i} (-c_i)^2 \right] >$$
$$\frac{1}{N \times I} \left[\sum_{x_i^n \leq -c_i} (x_i^n)^2 + \sum_{x_i^n > 0} (0)^2 \right] \geq$$
$$\frac{1}{N \times I} \left[\sum_{x_i^n \leq 0} (x_i^n)^2 + \sum_{x_i^n > 0} (0)^2 \right] = P_i(0)$$

Logo, pode-se concluir que :

$$P_i(c_i) > P_i(0) \text{ se } c_i < 0 \tag{6.1}$$

Através da equação 6.1, consegue-se concluir que para qualquer $c_i < 0$, $P_i(c_i) > P_i(0)$, no entanto, isto é um absurdo, porque, c_i é o ponto onde o P_i é mínimo.

Posto isto, concluímos que $c_i \geq 0$.

Segunda parte:

Supondo que $c_i > 0$, então $-c_i < 0$.

Além disso, note-se que:

$$\sum_{-c_i < x_i^n \leq 0} (-c_i)^2 \geq \sum_{-c_i < x_i \leq 0} (x_i^n)^2$$

Esta última propriedade é válida porque se $-c_i < x_i \leq 0$ então $x_i^2 \leq c_i^2$

Aplicando-se esta propriedade, concluímos que :

$$P_i(c_i) = \frac{1}{N \times I}\left[\sum_{x_i^n \leq -c_i}(x_i^n)^2 + \sum_{x_i^n > -c_i}(-c_i)^2\right] =$$

$$\frac{1}{N \times I}\left[\sum_{x_i^n \leq -c_i}(x_i^n)^2 + \sum_{-c_i < x_i^n \leq 0}(-c_i)^2 + \sum_{x_i^n > 0}(-c_i)^2\right] \geq$$

$$\frac{1}{N \times I}\left[\sum_{x_i^n \leq -c_i}(x_i^n)^2 + \sum_{-c_i < x_i^n \leq 0}(x_i^n)^2 + \sum_{x_i^n > 0}(-c_i)^2\right] =$$

$$\frac{1}{N \times I}\left[\sum_{x_i^n \leq 0}(x_i^n)^2 + \sum_{x_i^n > 0}(-c_i)^2\right] \geq$$

$$\frac{1}{N \times I}\left[\sum_{x_i^n \leq 0}(x_i^n)^2 + \sum_{x_i^n > 0}(0)^2\right] = P_i(0)$$

Logo, $P_i(c_i) \geq P_i(0)$, então concluímos que $P_i(0)$ é um mínimo e $c_i = 0$ é um minimizante.

□

Dado que se conseguiu provar que $c = 0$ é a melhor solução, pode-se enunciar o seguinte corolário.

Corolário 6.8.1. *A função de loss $\mathcal{L}(X; U, V, c)$ pode ser escrita da seguinte maneira:*

$$\mathcal{L}(X; U, V, c) = \frac{1}{N \times I}\sum_{i=1}^{I}\sum_{x_i^n \leq 0}(x_i^n)^2$$

Demonstração.

$$\mathcal{L}(X; U, V, c) = \sum_{i=1}^{I}(\frac{1}{N \times I}\sum_{n=1}^{N}\min(-c_i, x_i^n)^2) =$$

$$= \sum_{i=1}^{I}(\frac{1}{N \times I}\sum_{n=1}^{N}\min(0, x_i^n)^2) =$$

$$\sum_{i=1}^{I}\frac{1}{N \times I}\left[\sum_{x_i^n \leq 0}(x_i^n)^2 + \sum_{x_i^n > 0}(0)^2\right]$$

□

6.8.1 Validação numérica

Nesta subsecção vai-se realizar um teste de maneira a comprovar computacionalmente o que foi demonstrado no secção anterior.

Nota 6.8.1. *Ao longo desta secção vai-se denotar o valor visto no corolário 6.8.1 por E, ou seja,*

$$E = \frac{1}{N \times I} \sum_{i=1}^{I} \sum_{x_i^n \leq 0} (x_i^n)^2$$

Vai-se, então, considerar uma base de dados regular, por exemplo:

$$X = \begin{bmatrix} 152.68832091 & 195.18113442 & 237.67394792 & 72.9653076 & -21.11650828 \\ 12.11276288 & 9.9517971 & 7.79083132 & -142.33127283 & 80.06139482 \\ -51.4854771 & -65.31923111 & -79.15298513 & 46.97893777 & -58.52755459 \\ 33.21603283 & 54.76599507 & 76.3159573 & 112.07249474 & 46.87895357 \\ 46.60620021 & 46.40322883 & 46.20025745 & -80.23577542 & -62.01848177 \\ -178.4100194 & -235.17169139 & -291.93336338 & -150.61820656 & 4.70820089 \\ -27.2391474 & -58.33437058 & -89.42959375 & -99.48917978 & -191.93605617 \\ 213.88920068 & 313.58061614 & 413.2720316 & 296.03557126 & 258.59381221 \\ -168.26012372 & -219.52480195 & -270.78948018 & -133.53547346 & 23.13970588 \\ 37.70442683 & 63.29493017 & 88.88543352 & 34.7720536 & 159.20070606 \\ 39.99768379 & 64.7806367 & 89.56358961 & 60.58887488 & 116.81350866 \\ 140.40184189 & 185.74716339 & 231.09248488 & 126.7564125 & -3.81808025 \\ -72.78727629 & -133.17169213 & -193.55610797 & -265.97151224 & -240.27964895 \\ -34.2688556 & -23.9456907 & -13.62252581 & 165.80722924 & 60.87731632 \\ -118.97697455 & -181.78645238 & -244.59593022 & -211.79176397 & -184.99104644 \end{bmatrix}$$

Segundo o corolário 6.8.1 :

$$E = \frac{1}{N \times I} \sum_{i=1}^{I} \sum_{x_i^n < 0} (x_i^n)^2 =$$

$$= \frac{1}{75}[(-21.116)^2 + (-142.331)^2 + (-51.485)^2 + (-65.319)^2 + (-79.152)^2 + (-58.527)^2 +$$
$$(-80.235)^2 + (-62.018)^2 + (-178.41)^2 + (-235.171)^2 + (-291.933)^2 + (-150.618)^2 +$$
$$(-27.2391)^2 + (-58.3343)^2 + (-89.429)^2 + (-99.489)^2 + (-191.936)^2 + (-168.260)^2 +$$
$$(-219.524)^2 + (-270.789)^2 + (-133.535)^2 + (-3.818)^2 + (-72.787)^2 + (-133.171)^2 +$$
$$(-193.556)^2 + (-265.971)^2 + (-240.279)^2 + (-34.268)^2 + (-23.945)^2 + (-13.622)^2 +$$
$$(-118.976)^2 + (-181.786)^2 + (-244.595)^2 + (-211.791)^2 + (-184.991)^2] =$$
$$= 11282.806$$

De maneira a obter as matrizes de pesos e vetores bias que vão gerar este mínimo, basta considerar:

$$U = \begin{bmatrix} -0.33309933 & -0.47708251 & -0.6210657 & -0.43621622 & -0.29228398 \\ -0.30453238 & -0.28308749 & -0.26164259 & 0.39426228 & 0.77667362 \\ 0.12142726 & 0.13067688 & 0.13992649 & -0.8035932 & 0.55030676 \end{bmatrix}$$

$$V = \begin{bmatrix} -0.33309933 & -0.47708251 & -0.6210657 & -0.43621622 & -0.29228398 \\ -0.30453238 & -0.28308749 & -0.26164259 & 0.39426228 & 0.77667362 \\ 0.12142726 & 0.13067688 & 0.13992649 & -0.8035932 & 0.55030676 \end{bmatrix}$$

$$a = \begin{bmatrix} 682.23760561 & 180.97355656 & 108.93683629 \end{bmatrix}$$

$$b = \begin{bmatrix} 269.13729608 & 362.47945508 & 455.82161407 & 313.79296035 & -1.09894071 \end{bmatrix}$$

É de salientar que $\forall x^n \in X$, $VUx^n = x^n$.

$$VU = \begin{bmatrix} 0.21843972 & 0.26099291 & 0.3035461 & -0.07234043 & -0.07234043 \\ 0.26099291 & 0.3248227 & 0.38865248 & -0.00851064 & -0.00851064 \\ 0.3035461 & 0.38865248 & 0.47375887 & 0.05531915 & 0.05531915 \\ -0.07234043 & -0.00851064 & 0.05531915 & 0.99148936 & -0.00851064 \\ -0.07234043 & -0.00851064 & 0.05531915 & -0.00851064 & 0.99148936 \end{bmatrix}$$

Por exemplo, para o primeiro elemento

$$x^0 = [152.68832091, 195.18113442, 237.67394792, 72.9653076, -21.11650828]$$

$$VUx^0$$

$$=$$

$$\begin{bmatrix} 0.21843972 & 0.26099291 & 0.3035461 & -0.07234043 & -0.07234043 \\ 0.26099291 & 0.3248227 & 0.38865248 & -0.00851064 & -0.00851064 \\ 0.3035461 & 0.38865248 & 0.47375887 & 0.05531915 & 0.05531915 \\ -0.07234043 & -0.00851064 & 0.05531915 & 0.99148936 & -0.00851064 \\ -0.07234043 & -0.00851064 & 0.05531915 & -0.00851064 & 0.99148936 \end{bmatrix} \times \begin{bmatrix} 152.68832091 \\ 195.18113442 \\ 237.67394792 \\ 72.9653076 \\ -21.11650828 \end{bmatrix} =$$

$$= \begin{bmatrix} 152.68832091 \\ 195.18113442 \\ 237.67394792 \\ 72.9653076 \\ -21.11650828 \end{bmatrix}$$

Analogamente, podia-se demonstrar o mesmo para os restantes vetores.

Definindo-se os pesos do *autoencoder*, com as matrizes e com os vetores bias mencionados acima, usando *Python*, este diz que o valor de *loss* é de 11282.806, o que vai ao encontro do que se demonstrou.

Vai-se agora testar se o *tensorflow* consegue encontrar a solução que foi descrita acima como sendo a mínima. Para tal, vai-se treinar 3 tipos de *autoencoder*: inicialmente, um *autoencoder* normal, ou seja, o *autoencoder* que faz apenas uso da função de ativação ReLU, outro que irá fazer uso da função de ativação Elu e, por fim, um que vai utilizar regularizadores.

Os resultados do treino podem ser observados na figura 6.11

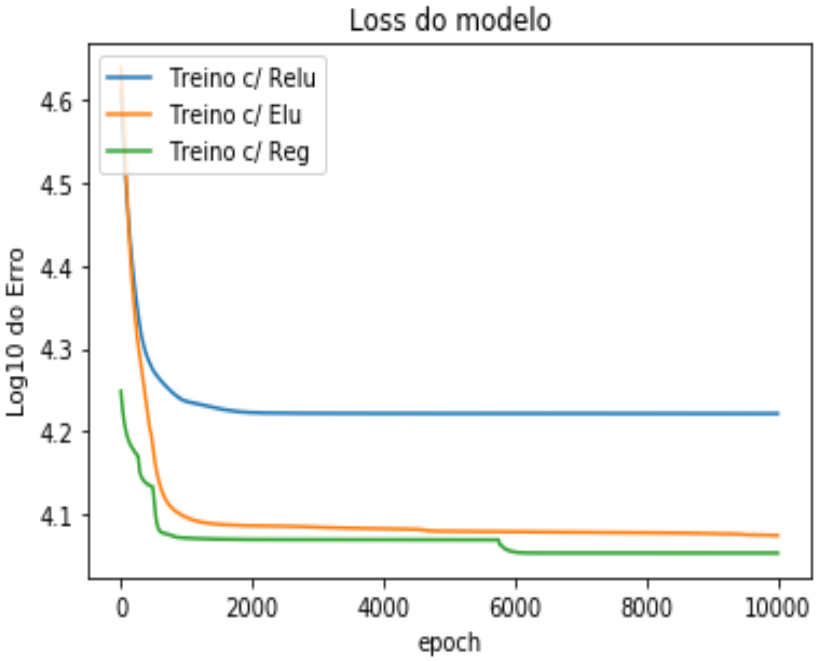


Figura 6.11: Comparação do valor de *loss*

Sendo os valores de *loss*:

- Treino c/ ReLU : 16656.3613

- Treino c/ ELU : 11879.6572

- Treino c/ Regularizadores : 11314.6181

Através da informação mencionada acima e na figura 6.11, pode-se concluir que nenhuma abordagem conseguiu chegar ao real valor do mínimo. Porém, o treino com ELU e com regularizadores está bastante próximo de atingir o valor mínimo.

Capítulo 7

Conclusão

Esta dissertação teve como objetivo principal estudar os fundamentos matemáticos dos *autoencoders*, uma família de algoritmos de machine learning não supervisionados.

Os *autoencoders* foram abordados em duas etapas. Inicialmente, o estudo focou os *autoencoders* lineares, tendo sido desenvolvido um trabalho aprofundado a nível matemático e estabelecidos diversos resultados importantes acerca desta classe de *autoencoders*. Outra vertente do estudo foram os *autoencoders* que utilizam como função de ativação a função ReLU, cuja análise, à semelhança do caso anterior, permitiu alcançar diversos resultados, apresentados ao longo do trabalho.

Além da vertente matemática, esta dissertação compreendeu também um parte prática, na qual a generalidade das propriedades demonstradas matematicamente puderam ser corroboradas através da sua validação numérica. A este respeito, é ainda de salientar o desenvolvimento na secção 6.4 de um algoritmo novo de treino para *autoencoders* ReLU, capaz de aproximar bem os algoritmos mais usuais, mas com um custo computacional inferior, que pode ser visto como uma contribuição no contexto do desenvolvimento e optimização de métodos de *machine learning*.

Em muitos casos, as validações numéricas permitiram encontrar boas aproximações às soluções ideais e, em certas situações, foi mesmo possível atingir as soluções ideais. O conhecimento das soluções ideais, por exemplo, no caso de um *autoencoder* ReLU, constitui uma ferramenta útil para, através de diversas experiências de treino, se perceber se o treino está a ser realizado de forma adequada.

E ainda importante salientar que, ao longo deste trabalho, por diversas vezes, foram utilizados regularizadores nos processos de minimização de *loss functions*, que, de facto, permitiram atingir menores valores de erro. ´

É de notar que, em muitas simulações, se optou pelo uso de bases de dados regulares para provar certas propriedades. A este respeito há dois aspetos importantes a destacar: por um lado, quando temos uma base de dados apenas com valores positivos, a função ReLU não é ativada, retornando, então, o problema ao caso linear (na investigação bibliográfica que efetuamos, não conseguimos encontrar esta observação na literatura); por outro lado, ainda no contexto de uma base de dados regular, mas agora com a possibilidade de existirem valores negativos, é possível identificar que, ao nível do cálculo do erro, a função de ativação irá cortar todos os elementos de valor negativo Uma extensão futura ao trabalho desenvolvido nesta dissertação poderia passar por analisar como se comportam as propriedades estudadas caso se trabalhe com bases de dados regulares com algumas perturbações, ou mesmo, caso se considerem bases de dados que não sejam de todo regulares.

Bibliografia

[1] Géron, Aurélien. Hands-On Machine Learning with Scikit-Learn and Keras and TensorFlow, 2nd Edition

[2] Burkov, Andriy. The hundred-page machine learning book , 2019

[3] D. Tomar, Y. Prasad, M. K. Thakur and K. K. Biswas, "Feature Selection Using Autoencoders,"2017 International Conference on Machine Learning and Data Science (MLDS), Noida, 2017, pp. 56-60, doi: 10.1109/MLDS.2017.20

[4] Mayu Sakurada, Takehisa Yairi. "Anomaly Detection Using Autoencoders with Nonlinear Dimensionality Reduction." , dl.acm.org/doi/10.1145/2689746.2689747.

[5] An, Jinwon, and Sungzoon Cho. "Variational Autoencoder based Anomaly Detection using Reconstruction Probability", dm.snu.ac.kr/static/docs/TR/SNUDM-TR-2015-03.pdf.

[6] Ian Goodfellow and Yoshua Bengio and Aaron Courville, Deep Learning, MIT Press, 2016

[7] Abdi, Hervé, and Lynne J. Williams. "Principal Component Analysis."

[8] Mendes Martins, Paula. Um Curso De Álgebra Linear I. Second ed., 2006, ISBN: 972-8810-09-1.

[9] Mendes Martins, Paula. Um Curso De Álgebra Linear II. 2005, ISBN: 972-8810-10-5

[10] Baldi, Pierre and Hornik, Kurt. Neural Networks and Principal Component Analysis: Learning from Examples Without Local Minima

[11] Zhu, Xiaojin. Principal Component Analysis 1 Basic Linear Algebra Review. pages.cs.wisc.edu/ jerryzhu/cs540/handouts/PCA.pdf.

[12] Zemel, Richard, et al. CSC 411: Lecture 14: Principal Components Analysis ... www.cs.toronto.edu/ urtasun/courses/CSC411_Fall16/14_pca.pdf.

[13] Manning-Dahan, Tyler. PCA and Autoencoders. 2017, tylermd.com/pdf/pca_ae.pdf

[14] https://deeplearningbook.com.br/introducao-aos-autoencoders/

[15] Plaut, Elad. "From Principal Subspaces to Principal Components with Linear Autoencoders."

[16] Randal J. Barnes, Matrix Differentiation

[17] Leow Wee Kheng, Matrix Differentiation, Theoretical Foundations in Multimedia , https://www.comp.nus.edu.sg/~cs5240/lecture/matrix-differentiation.pdf

[18] Chris, and David Cato. "How to Use L1, L2 and Elastic Net Regularization with Keras?" MachineCurve, 28 Apr. 2020, www.machinecurve.com/index.php/2020/01/23/how-to-use-l1-l2-and-elastic-net-regularization-with-keras/.

Anexo A

Funções implementadas

A.1 Ficheiro `Autoencoders.py`

O código pode ser visto em https://github.com/RuiTeixeira1997/Autoencoders

```
# -*- coding: utf-8 -*-
"""

@author: Rui Teixeira

"""
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
import os
import keras
import tensorflow as tf
from keras.layers import Input, Dense
from keras import regularizers, models, optimizers
import numpy as np
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
from sklearn import decomposition
import random
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import math
import abc
import six
import math
from tensorflow.python.distribute import distribution_strategy_context
from tensorflow.python.framework import ops
from tensorflow.python.framework import smart_cond
from tensorflow.python.framework import tensor_util
from tensorflow.python.keras import backend as K
```

```
30 from tensorflow.python.keras.utils import losses_utils
31 from tensorflow.python.keras.utils import tf_utils
32 from tensorflow.python.keras.utils.generic_utils import deserialize_keras_object
33 from tensorflow.python.keras.utils.generic_utils import serialize_keras_object
34 from tensorflow.python.ops import array_ops
35 from tensorflow.python.ops import math_ops
36 from tensorflow.python.ops import nn
37 from tensorflow.python.ops.losses import losses_impl
38 from tensorflow.python.ops.losses import util as tf_losses_util
39 from tensorflow.python.util.tf_export import keras_export
40 from tensorflow.tools.docs import doc_controls
41 from tensorflow.keras.callbacks import ModelCheckpoint
42 from keras.models import Sequential, Model
43 from keras.optimizers import Adam
44 from numpy import linalg as LA
45 import pickle
46 from numpy.linalg import matrix_rank
47
48 K=np.array([[1,2,1,0.5,2.5],[1.5,0.5,1,0,2],[1,0.5,2,2,5]])
49
50 def AnalyticalPCA(y, dimension):
51     pca = PCA(n_components=dimension)
52     pca.fit(y)
53     loadings = pca.components_
54
55     return loadings
56
57 def generate_sequence(length, n_features):
58     '''
59     Parameters
60     ----------
61     length : Tamanho da atributos, corresponde ao i
62     n_features : Número máximo e mínimo dos valores de base de dados, por exemplo
63     se n_features = 5 vai gerar valores entre [-5,5]
64     Returns
65     -------
66     list : Vetor de dimensão lenght com valores entre [-n_features,n_features]
67     '''
68     return [random.uniform(-n_features, n_features) for _ in range(length)]
69
70 def criar_DB(Tamanho_da_DB,Tamanho_das_amostras,Intervalo,ficheiro):
71     '''
72     Gerar base de dados para teste nos autoenconders e guarda-la num ficheiro.
73
74     Parameters
75     ----------
76     Tamanho_da_DB : Corresponde ao n
77     Tamanho_das_amostras : Corresponde ao I
78     Intervalo : Intervalo para gerar os valores dos vetores
79     ficheiro : Nome do ficheiro
80
```

```
    '''
    X=[]
    for i in range(Tamanho_da_DB):
        a=generate_sequence(Tamanho_das_amostras,Intervalo)
        X.append(a)
    pickle.dump( np.asarray(X), open( ficheiro, "wb" ) )
    return np.array(X, dtype='f')

def criar_DB1(Tamanho_da_DB,Tamanho_das_amostras,Intervalo,ficheiro):
    '''
    Caso especial para os valores proprios serem muito distintos em tamanho de grandeza

    '''
    X=[]
    for i in range(Tamanho_da_DB):
        a=generate_sequence(Tamanho_das_amostras,Intervalo)
        a[0]=a[0]*100
        a[1]=a[1]*20
        a[2]=a[2]*10
        a[3]=a[3]
        a[4]=a[4]*(1/5)
        X.append(a)
    pickle.dump( np.asarray(X), open( ficheiro, "wb" ) )
    return np.array(X, dtype='f')

def load_DB(ficheiro):
    '''
    Carregar base de dados atrvés de um ficheiro

    '''
    return pickle.load( open(ficheiro, "rb" ) )

def centrar_DB(X,A='F'):
    '''
    Função que recebe uma base de dados e vai centrar a mesma.
    Caso A=='T' devolve o vetor da média.
    '''
    P=np.array(X, dtype='f')
    media=[]
    for j in range(len(P[0])):
        media.append(np.mean(P[:,j]))
    for i in range(X.shape[0]):
        P[i]=P[i] - media
    if A=='T':
        print('vetor_media= ', media)

    return P


def loss_f(U,V,a,b,x):
    '''
```

```
132     Função de loss contruída manualmente
133     '''
134     k=x
135     z=np.matmul(U,x)+a
136     xn=np.matmul(V,z)+b
137     for i in range(len(k)):
138         k[i]=k[i]*k[i]
139         xn[i]=xn[i]*xn[i]
140     erro1=k-xn
141     return np.sum(erro1)/len(k)
142 
143 def loss_TF(U,V,a,b,x):
144     '''
145     Função de loss baseada no tensorflow
146     '''
147     z=np.matmul(U,x)+a
148     xx=np.matmul(V,z)+b
149     y_pred = ops.convert_to_tensor(xx)
150     y_true = math_ops.cast(x, y_pred.dtype)
151     return K.mean(math_ops.squared_difference(y_pred, y_true), axis=-1)
152 
153 def loss_database(U,V,a,b,database):
154     '''
155     Dados os pesos do autoencoder, calcula o valor de loss para todos os elementos da
    base
156     de dados fazendo uso da função de loss construída manualmente
157     '''
158 
159     err2=0
160     for x in database:
161         r=loss_f(U,V,a,b,x)
162         err2=err2+r
163 
164     j=err2/len(database)
165     return j
166 
167 def loss_databaseTF(U,V,a,b,database):
168     '''
169     Dados os pesos do autoencoder, calcula o valor de loss para todos os elementos da
    base
170     de dados fazendo uso da função de loss baseada no tensorflow
171     '''
172 
173     err3=0
174     for x in database:
175         r=loss_TF(U,V,a,b,x)
176         err3=err3+r
177     return err3/len(database)
178 
179 def Teste(V):
180     '''
```

```
    Função que vai testar se (V^TV)^-1V^T=U
    '''
    z=np.matmul(np.transpose(V),V)
    k=LA.inv(z)
    j=np.matmul(k,np.transpose(V))
    return j

def Teste1(V,a,b):
    '''
    Verificar se -b -Va dá zero

    Returns
    -------
    z : O valor de Va
    b : O valor de b
    -b -z : O valor de -b -Va que queremos verificar se está perto de zero
    k : o valor máximo em modulo de -b-Va
    '''
    z=np.matmul(V,a)
    k=-b-z
    return z , b , k, np.max(np.abs(k))


def XTX(x):
    '''
    Recebendo uma matriz A,retorna o valor de A^TA.
    '''
    return np.matmul(np.transpose(x),x)

def Teste2(X,U):
    '''
    Verificar se V=X^{T}XU^{T}(UX^{T}XU^{T})^{-1}

    '''
    O=XTX(X)
    O1=np.matmul(O,np.transpose(U))
    O2=np.matmul(U,O)
    O3=np.matmul(O2,np.transpose(U))
    k=LA.inv(O3)
    FINAL=np.matmul(O1,k)
    return FINAL

def produtoInterno(x,y,axes):
    '''
    Calcula o produto interno entre dois vetores
    '''
    return tf.tensordot(x,y,axes)

def Operations(x):
    '''
    Dado como input uma matriz A, devolve os valores e vetores próprios da matriz A^TA.
```

```
232     '''
233     xTx=np.matmul(np.transpose(x),x)
234     w, v = LA.eig(xTx)
235     return w,v
236 
237 
238 def OrdenaVal(C):
239     '''
240     Função que recebendo uma matriz A, devolve os valores e vetores próprios ordenados
    de A^TA
241     '''
242     val,vec=Operations(C)
243     idx = val.argsort()[::-1]
244     val = val[idx]
245     vec = vec[:,idx]
246     return val,vec
247 
248 def VerifyOrt(U,C,mostrar='F'):
249     '''
250     Função que vai verificar a ortogonalidade entre os vetores de U e os últimos
251     vetores próprios da matriz C^TC
252 
253     Retorna o desvio à ortogonalidade
254     '''
255     C=np.array(C, dtype='f')
256     val,vec=OrdenaVal(C)
257     if mostrar=='T':
258         print('Valores próprios:')
259         print(val)
260         print('Vetores próprios:')
261         print(vec)
262     J=len(C[0])-len(U)
263     A=[]
264     for i in U:
265         B=[]
266         for p in range(J):
267             q=produtoInterno(i,np.transpose(vec[:,J+p+1]),1)
268             B.append(q/(np.linalg.norm(i,2)*np.linalg.norm(vec[:,J+p+1],2)))
269         A.append(B)
270     return A , np.max(np.abs(A))
271 
272 def Relu(vec):
273     '''
274     recebe um vetor como input e deolve o vetor correspondente ao calculo da ReLU
275     para esse mesmo vetor
276     '''
277     return tf.nn.relu(vec).numpy()
278 
279 def Find_pattern_Enc(X,U,a):
280     '''
281     Função que procura padrões ao fazer encode dos dados
```

```
    '''
    contador=np.zeros(len(U))
    for i in range(len(X)):
        k=Relu(np.matmul(U,X[i])+a)
        for j in range(len(U)):
            if k[j]==0:
                contador[j]+=1
    perc=np.zeros(len(U))
    for i in range(len(contador)):
        perc[i]=(contador[i]/len(X))*100
    return contador,perc


def Find_pattern_Dec(X,U,V,a,b):
    '''
    Função que procura padrões ao fazer decode dos dados
    '''
    contador=np.zeros(len(V))
    for i in range(len(X)):
        k=Relu(np.matmul(U,X[i])+a)
        z=Relu(np.matmul(V,k)+b)
        for j in range(len(V)):
            if z[j]==0:
                contador[j]+=1
    perc=np.zeros(len(V))
    for i in range(len(contador)):
        perc[i]=(contador[i]/len(X))*100
    return contador,perc


def get_par(X,I,J):
    '''
    Função que retorna parâmetros específios de U e V muito usados ao longo
    do documento
    '''
    val,vec=OrdenaVal(X)
    V=vec[:,:J]
    U=np.transpose(V)
    return U ,V

def DB_Generator(tipo,I,J):
    '''
    Parameters
    ----------
    tipo : É o tipo do gerador:
        tipo 1 = Base canónica
        tipo 2 = Base em que o vetor i tem componentes todas 1's excepto na posição i
que é zero a multiplicar por
                 1*sqrt(I-1)
        tipo 3 = Gerar um valor aleatorio para o primeiro vetor e de seguidas os outros
vão ser todos zeros excepto na
```

```
331                     primeira posição que vai ser o elemento j do primeiro vetor e na
    posição i vamos ter - a primeiro
332                     elemento do vetor aleatorio
333         I : Número de atributos
334         J : Número a reduzir a dimensão
335         Returns
336         -------
337         A : Devolve o gerador
338 
339         '''
340         if tipo==1:
341             A=[]
342             for i in range(J):
343                 p=np.zeros(I)
344                 p[i]=1
345                 A.append(p)
346             return A
347 
348         if tipo==2:
349             A=[]
350             for i in range(J):
351                 p=np.ones(I)
352                 p[i]=0
353                 k=1/math.sqrt(I-1)
354                 A.append(k* p)
355             return A
356         if tipo==3:
357             e=generate_sequence(I,10)
358             A=[]
359             A.append(np.array(e))
360             for i in range(J-1):
361                 p=np.zeros(I)
362                 p[0]=e[i+1]
363                 p[i+1]=-e[0]
364                 A.append(p)
365             return A
366 
367     def DB_type(tipo,ficheiro,N,I,J):
368         '''
369 
370         Parameters
371         ----------
372         tipo : Tipo de gerador, opções : 1 ,2 e 3
373         N : Tamanho do Dataset
374         I : Número de Atributos
375         J : Valor a reduzir no autoenconder
376 
377         Returns
378         -------
379 
380         Um dataset com N amostras.
```

```
381
382     '''
383     gerador= DB_Generator(tipo,I,J)
384     L=[]
385     for i in range(N):
386         j=0
387         Val=np.zeros(I)
388         Lambdas=generate_sequence(J, 10)
389         for ele in gerador:
390 
391             Val+= (ele*Lambdas[j])
392             j=j+1
393         L.append(Val)
394     pickle.dump( np.array(L), open(ficheiro, "wb" ) )
395     return np.array(L)
396 
397 def Base_Regular(N,I,J,ficheiro):
398     '''
399     Gerar uma base de dados regular
400 
401     '''
402     X= DB_type(3,'Db_teste',10,I,J)
403     vec,_=get_par(X,I,J)
404     L=[]
405     for i in range(N):
406         j=0
407         Val=np.zeros(I)
408         Lambdas=generate_sequence(J, 10)
409         print(Lambdas)
410         for ele in vec:
411 
412             Val+= (ele*Lambdas[j])
413             j=j+1
414         L.append(Val)
415     pickle.dump( np.array(L), open(ficheiro, "wb" ) )
416     return np.array(L)
417 
418 def Base_Regular_positiva(N,I,J,ficheiro,ficheiro1):
419     '''
420     Gerar base de dados positiva (Todos os elementos são maiores que zero)
421     e guardar num ficheiro a base de dados e os vetores que a geram
422     '''
423     vec=[[1,2,3,4,5],[2,3,4,5,1],[3,4,5,1,2]]
424     vec=np.array(vec)
425     vec.astype(float)
426     L=[]
427     for i in range(N):
428         j=0
429         Val=np.zeros(I)
430         Lambdas=[random.uniform(0,50) for _ in range(J)]
431         for ele in vec:
```

```
                Val+= (ele*Lambdas[j])
                j=j+1
            L.append(Val)
        pickle.dump( np.array(L), open(ficheiro, "wb" ) )
        vec_norm=gs(vec)
        pickle.dump( vec_norm, open(ficheiro1, "wb" ) )

        return vec_norm,np.array(L)


def D(U,Util):
    '''
    Métrica : Diferença máxima
    '''
    return np.mean(np.abs(U-Util))

def M(U,Util):
    '''
    Métrica : Média da diferença
    '''
    return np.max(np.abs(U-Util))

def bmatrix(a):
    """Returns a LaTeX bmatrix

    :a: numpy array
    :returns: LaTeX bmatrix as a string
    """
    if len(a.shape) > 2:
        raise ValueError('bmatrix can at most display two dimensions')
    lines = np.array2string(a, max_line_width=np.infty).replace('[', '').replace(']',
    '').splitlines()
    rv = [r'\begin{bmatrix}']
    rv += ['  ' + ' & '.join(l.split()) + r'\\' for l in lines]
    rv +=  [r'\end{bmatrix}']
    return rv

def sn(x,U,a):
    'Calculo de sn'
    s=np.zeros(len(U))
    k=Relu(np.matmul(U,x)+a)
    for i in range(len(s)):
        if k[i]>0:
            s[i]=1
        else:
            pass
    return s


def tn(x,U,V,a,b):
```

```
    'calculo de tn'
    t=np.zeros(len(V))
    z=Relu(np.matmul(U,x)+a)
    xx=Relu(np.matmul(V,z)+b)
    for i in range(len(t)):
        if xx[i]>0:
            t[i]=1
        else:
            pass
    return t

def Isn(x,U,a):
    'calculo da matriz Isn'
    s=sn(x,U,a)
    Is=np.zeros([len(U),len(U)])
    for i in range(len(U)):
        Is[i][i]=s[i]
    return Is

def Itn(x,U,V,a,b):
    'calculo da matriz Itn'
    t=tn(x,U,V,a,b)
    It=np.zeros([len(V),len(V)])
    for i in range(len(V)):
        It[i][i]=t[i]
    return It

def print_history_loss(history):
    '''
    Devolve o gráfico do valor de loss para o nº de épocas
    '''
    # print(history.history.keys())
    plt.plot(np.log10(history.history['loss']))
    plt.title('Loss do modelo')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.show()

def print_erro_loss(history,erro):
    '''
    Gráfico que compara os erros de L(X;U,V,a,b) e K(X;a,b)
    '''
    # print(history.history.keys())
    plt.plot(np.log10(history.history['loss']))
    plt.plot(np.log10(erro))
    plt.title('Loss do modelo')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['L(X;U,V,a,b)', 'K(X;a,b)'], loc='upper left')
    plt.show()
```

```
533
534 def create_Autoencoder(I,J,activ):
535     '''
536     Parameters
537     ----------
538     I : Valor de I
539     J : Tamanho para o qual se vai reduzir os dados
540
541     Returns
542     -------
543     autoencoder : Retorna o modelo do autoenconder
544     '''
545     input_i=Input(shape=(I,))
546     encoded = Dense(units=J, activation=activ)(input_i)
547     decoded = Dense(units=I, activation=activ)(encoded)
548     encoder = Model(inputs=input_i, outputs=encoded)
549     autoencoder = Model(inputs=input_i, outputs=decoded)
550     return autoencoder
551
552 def create_AutoencoderELU(I,J):
553     '''
554     Autoencoder com a função de ativação ELU
555     '''
556     input_i=Input(shape=(I,))
557     encoded = Dense(units=J, activation=tf.nn.elu)(input_i)
558     decoded = Dense(units=I, activation=tf.nn.elu)(encoded)
559     encoder = Model(inputs=input_i, outputs=encoded)
560     autoencoder = Model(inputs=input_i, outputs=decoded)
561     return autoencoder, encoder
562
563
564
565 def create_Autoencoder_regularizador(I,J,activ):
566     '''
567     Autoencoder que utiliza regularizadores
568     '''
569     input_i=Input(shape=(I,))
570     encoded = Dense(units=J,
        activation=activ,kernel_regularizer=regularizers.l1_l2(l1=0.01, l2=0.01))(input_i)
571     decoded = Dense(units=I,
        activation=activ,kernel_regularizer=regularizers.l1_l2(l1=0.01, l2=0.01))(encoded)
572     encoder = Model(inputs=input_i, outputs=encoded)
573     autoencoder = Model(inputs=input_i, outputs=decoded)
574     return autoencoder, encoder
575
576
577
578 def compiletrain(autoencoder, X_train,epocas):
579     '''
580     Função que realiza o treino do autoencoder
581     '''
```

```
582     checkpointer = ModelCheckpoint(filepath="best_weights_modelo.hdf5", monitor =
        'loss', verbose=1, save_best_only=True)
583     autoencoder.compile(optimizer='adam', loss='mean_squared_error', metrics=['accuracy'])
584     history = autoencoder.fit(X_train,
        X_train,epochs=epocas,shuffle=True,callbacks=[checkpointer])
585     #history = autoencoder.fit(X_train, X_train,epochs=epocas,shuffle=True)
586     return autoencoder,history
587
588
589
590 def compiletrain_bias_set(autoencoder, X_train,epocas):
591     '''
592     Função que permite inicializar o treino escolhe os parâmetros iniciais
593     '''
594     #checkpointer = ModelCheckpoint(filepath="best_weights_modelo.hdf5", monitor =
        'loss', verbose=1, save_best_only=True)
595     autoencoder.compile(optimizer='adam', loss='mean_squared_error', metrics=['accuracy'])
596     #history = autoencoder.fit(X_train,
        X_train,epochs=10000,shuffle=True,callbacks=[checkpointer])
597     for i in range(epocas):
598         history = autoencoder.fit(X_train, X_train,epochs=1,shuffle=True)
599         V,a,U,b=autoencoder.get_weights()
600         autoencoder.set_weights((V,[100,100,100],U,[100,100,100,100,100]))
601     return autoencoder,history
602
603
604 def compiletrain_SET_ALL(autoencoder, X_train,U,V,a,b):
605     '''
606     Função que permite avaliar o erro do autoencoder com a função de loss
607     do tensorflow com os parâmetros escolhidos pelo utlizador
608     '''
609     autoencoder.compile(optimizer='adam', loss='mean_squared_error', metrics=['accuracy'])
610     autoencoder.set_weights((V,a,U,b))
611     for i in range(2):
612         history = autoencoder.fit(X_train, X_train,epochs=1,shuffle=True)
613         autoencoder.set_weights((V,a,U,b))
614     return autoencoder,history
615
616 def gradiente_a(x,U,V,a,b):
617
618     '''
619     Calculo do gradiente de a
620     '''
621
622     Is=Isn(x,U,a)
623     It=Itn(x,U,V,a,b)
624     p=2*(np.matmul(np.transpose(x),np.transpose(U)))
625     s=2*(np.matmul(np.transpose(x),It))
626     ss=np.matmul(s,V)
627     sss=np.matmul(ss,Is)
628     t=2*(np.matmul(np.transpose(b),V))
```

```
629     tt=np.matmul(t,Is)
630     q=2*np.transpose(a)
631     return a-sss+tt+q
632 
633 
634 def gradiente_b(x,U,V,a,b):
635 
636     '''
637     Calculo do gradiente de b
638     '''
639 
640     Is=Isn(x,U,a)
641     It=Itn(x,U,V,a,b)
642     p=2*(np.matmul(np.transpose(x),It))
643     s=2*(np.matmul(np.transpose(x),np.transpose(U)))
644     ss=np.matmul(s,np.transpose(Is))
645     sss=np.matmul(ss,np.transpose(V))
646     t=2*(np.matmul(np.transpose(a),np.transpose(Is)))
647     tt=np.matmul(t,np.transpose(V))
648     q=2*np.transpose(b)
649     return -p +sss+tt+q
650 
651 
652 
653 def loss_gradiente(X,a,b):
654 
655     '''
656     Calculo do valor de loss da base de dados X com os parâmetros calculados
657     a e b
658     '''
659 
660     J=len(a)
661     I=len(b)
662     U,V=get_par(X,I,J)
663     v_lossA=np.zeros(J)
664     for x in X:
665         v_lossA=v_lossA+gradiente_a(x,U,V,a,b)
666     v_lossA=v_lossA*(1/len(X))
667     v_lossB=np.zeros(I)
668     for x in X:
669         v_lossB=v_lossB+gradiente_b(x,U,V,a,b)
670     v_lossB=v_lossB*(1/len(X))
671     return v_lossA,v_lossB
672 
673 
674 
675 def Metodo_gradiente(n_iteras,X,a,b,lr):
676     '''
677     Parameters
678     ----------
679     n_iteras : Nº de itreções do método do gradiente desenvolvido
```

```
680         X : Base de dados
681         a : valor de a inicial
682         b : valor de b inicial
683         lr : learning rate
684 
685         Returns
686         -------
687         a : o valor a depois de aplicado o método do gradiente
688         b : o valor b depois de aplicado o método do gradiente
689         erros : vetor dos erros
690 
691         '''
692         U,V=get_par(X,5,3)
693         erros=[]
694         for i in range(n_iteras):
695             v_lossA,v_lossB=loss_gradiente(X,a,b)
696             a=a-lr*v_lossA
697             b=b-lr*v_lossB
698             erro=loss_databasenova(U,V,a,b,X)
699             erros.append(erro)
700             print(i,erro)
701         return a,b,erros
702 
703     def gs(X, row_vecs=True, norm = True):
704 
705         '''
706         Função que calcula O Processo de ortogonalização de Gram-Schmidt
707         '''
708         if not row_vecs:
709             X = X.T
710         Y = X[0:1,:].copy()
711         for i in range(1, X.shape[0]):
712             proj = np.diag((X[i,:].dot(Y.T)/np.linalg.norm(Y,axis=1)**2).flat).dot(Y)
713             Y = np.vstack((Y, X[i,:] - proj.sum(0)))
714         if norm:
715             Y = np.diag(1/np.linalg.norm(Y,axis=1)).dot(Y)
716         if row_vecs:
717             return Y
718         else:
719             return Y.T
720 
721 
722     def overbar_a(X,U):
723 
724         '''
725         Calculo de a_overbar
726         '''
727 
728         a=np.zeros(len(U))
729         val_max=-10000000000000000000
730         for j in range(len(U)):
```

```
731         val_max=-10000000000000000000
732         for i in range(len(X)):
733             z=np.matmul(U,X[i])
734             if val_max < z[j]:
735                 a[j]=z[j]
736                 val_max=z[j]
737     return a
738 
739 def underbar_a(X,U):
740 
741     '''
742     Calculo de a_underbar
743     '''
744     a=np.zeros(len(U))
745     val_min=10000000000000
746     for j in range(len(U)):
747         val_min=10000000000000
748         for i in range(len(X)):
749             z=np.matmul(U,X[i])
750             if val_min > z[j]:
751                 a[j]=z[j]
752                 val_min=z[j]
753     return -a
754 
755 
756 def erro(X):
757     '''
758     Calculo do erro da secção 6.7
759     '''
760     val=0
761     for i in range(15):
762         for j in range(5):
763             if X[i][j]<0:
764                 val=val+(X[i][j]*X[i][j])
765     return val/(15*5)
```