Julio César Barbieri Gonzalez de Almeida

Aprendizado Automático de Características Utilizando Autoencoders Esparsos

Brasil

2014

Julio César Barbieri Gonzalez de Almeida

Aprendizado Automático de Características Utilizando Autoencoders Esparsos

Universidade Federal Rural do Rio de Janeiro – UFRRJ

Instituto Multidisciplinar

Departamento de Ciência da Computação

Orientador: Prof. Leandro Guimarães Marques Alvim, D.Sc.

Brasil

2014

Julio César Barbieri Gonzalez de Almeida
Aprendizado Automático de Características Utilizando Autoencoders Esparsos/ Julio César Barbieri Gonzalez de Almeida. – Brasil, 2014-
56 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Leandro Guimarães Marques Alvim, D.Sc.

Trabalho de Conclusão de Curso (TCC) – Universidade Federal Rural do Rio de Janeiro – UFRRJ
Instituto Multidisciplinar
Departamento de Ciência da Computação, 2014.

1. Autoencoder Esparsos. 2. Aprendizado Não-Supervisionado. I. Orientador: D.Sc. Leandro Guimarães Marques Alvim. II. Universidade Federal Rural do Rio de Janeiro. III. Departamento de Ciência da Computação/Instituto Multidisciplinar. IV. Bacharel em Ciência da Computação

CDU 02:141:005.7

Julio César Barbieri Gonzalez de Almeida

Aprendizado Automático de Características Utilizando Autoencoders Esparsos

---

Prof. Leandro Guimarães Marques Alvim, D.Sc.
Orientador
DCC/IM/UFRRJ

Prof. Carlos Eduardo Ribeiro de Mello, Ph.D.
DCC/IM/UFRRJ

Prof. Fellipe Ribeiro Duarte, M.Sc.
DCC/IM/UFRRJ

Prof. Filipe Braida do Carmo, M.Sc.
DCC/IM/UFRRJ

Brasil
2014

Este trabalho é dedicado ao mundo,
afinal, é ele que irá usufruir deste.

Agradecimentos

Os agradecimentos principais são direcionados à minha namorada Mariana Kosiba Furtado, pelo apoio demonstrado durante essa longa caminhada, à minha mãe Rosilene Barbieri Gonzalez de Almeida e à toda minha família por todo o apoio e investimento realizado em mim ao longo de todos esses anos.

Agradecimentos especiais são direcionados ao meu orientador Leandro Guimarães Marques Alvim, pelo apoio na condução do presente trabalho, assim como pelo conhecimento passado a mim durante esta orientação, ao professor Ronaldo Ribeiro Goldschmidt, por me dar a primeira oportunidade para trabalhar em um projeto de iniciação científica e à todos os demais professores do curso de Ciência da Computação da Universidade Federal Rural do Rio de Janeiro.

“Stay hungry, stay foolish“
(Steve Jobs, 1955-2011)

Resumo

Em aprendizado de máquina, as variáveis de um conjunto de dados que representam um problema são denominadas *atributos*. Denomina-se *característica*, uma representação de mais alto nível gerada em função destes atributos. Uma dificuldade comum é a elaboração de características do problema que sejam relevantes à resolução de uma tarefa específica. Para isto, é necessário, muitas vezes, um especialista do domínio do problema para elaboração destas, que dependendo da natureza do problema pode ser muito caro. Neste trabalho, investigamos uma abordagem de aprendizado automático de características, cuja vantagem é a não utilização de um especialista do domínio. Em específico, investigamos a técnica de autoencoders esparsos e suas variações. Autoencoders esparsos, são redes neurais de três camadas cuja camada de entrada de dados é igual à camada de saída de dados. Seu principal objetivo é encontrar uma representação compacta do domínio que explique os dados. Estes são esparsos quando restringe-se o número de neurônios na camada intermediária ou limita-se o número de ativações nesta. Para a indicar a relevância da técnica abordada, realizamos experimentos com autoencoders e suas variações, com sucessivos empilhamentos e adição de ruído nos exemplos, como preprocessamento para uma rede neural de três camadas. Para estes experimentos, abordamos um problema de classificação de dígitos manuscritos em imagens digitais. O conjunto de dados utilizado foi o MNIST, com 70000 imagens. Nosso melhor modelo, utilizando autoencoders empilhados com adição de ruído nos dados, apresentou uma taxa de acerto de $98,59\%$ e uma redução de erro de 20% com relação ao *baseline* sem este preprocessamento. Indicamos também que, o não empilhamento de autoencoders não gera redução de erro significativa. Adicionalmente, a adição de ruído no conjunto de dados de treino acarreta em redução de erro considerável nos modelos. Nossa principal contribuição acerca deste trabalho se dá em elucidar ao leitor que pode-se melhorar de forma significativa a qualidade de um modelo sem a necessidade de um especialista no domínio do problema.

Palavras-chaves: Autoencoders Esparsos; Aprendizado de Máquina Não-Supervisionado.

Abstract

In machine learning, dataset variables representing a problem are named *attributes*. We denominate *feature*, a higher level variable representation generated according to these attributes. A common issue is feature development of a problem that are relevant for solving an specific task. Thus, it is often necessary a domain specialist to design these features, that depending on the nature of the problem can be very expensive. In this paper, we investigate an automated learning approach with no need of domain experts. In particular, we investigate sparse autoencoders and their variations. Sparse Autoencoders are three-layer neural network whose input layer is the same as the data output layer. It's main objective is to find a compact representation of the domain in order to explain the data. We call it sparse when the number of neurons is restricted in the hidden layer or the number of activations is limited. To indicate the technique relevance, we conducted experiments with autoencoders and it's variations, with successive stacks and adding noise in the data samples, as a preprocessing for a three layer neural network. For the experiements, we approach a a handwritten digit classification problem in digital images. The data set used was MNIST with 70000 images. Our best model, using stacked autoencoders with the addition of noise in the data, showed an acuracy rate of 98.59% and an error reduction of 20% regarding *baseline* without this preprocessing. We also note that non-stacking autoencoders doesn't generate significant error reduction. Furthermore, the addition of noise in the training data set causes a considerable reduction in error models. For this work, our main contribution is about elucidating the reader that is possible to significantly improve the quality of a model without the need of a domain specialist.

Key-words: Sparse Autoencoders; Unsupervised Learning.

Lista de ilustrações

Figura 1 – Demonstração gráfica de como ocorre o aprendizado supervisionado. Baseada em (NG, 2000) 24
Figura 2 – Imagem ilustrativa ao exemplo de classificação de dígitos. As setas vermelhas indicam as distâncias e curvaturas características da letra A . . 25
Figura 3 – Um exemplo de como uma imagem pode ser gradualmente transformada em representações mais abstratas à partir de dados brutos. Adaptada de (BENGIO, 2009) 26

Figura 4 – Perceptron (NG, 2011) 31
Figura 5 – Exemplo de Rede Neural de Múltiplas Camadas com 3 neurônios na camada de entrada, 3 na camada oculta e 1 na camada de saída(NG, 2011). 33
Figura 6 – Exemplo de Autoencoder, com o número de entradas igual ao número de saídas. (NG, 2011) 37
Figura 7 – Um exemplo x é corrompido para $\tilde{x}$. O autoencoder então, o mapeia para y e tenta reconstruir x (VINCENT et al., 2008). 39

Figura 8 – Exemplo de dígitos contidos no conjunto de dados MNIST. 45
Figura 9 – Gráfico de decaimento do erro de validação da rede neural (azul) e do autoencoder (vermelho). 48
Figura 10 – Gráfico de decaimento do erro de validação do autoencoder esparso e rede neural (azul) e do autoencoder (vermelho). 49
Figura 11 – Gráfico de erro por número de neurônios na camada oculta para os experimentos com o autoencoder. 50
Figura 12 – Gráfico de erro por número de neurônios na camada oculta para os experimentos com a rede neural. 51
Figura 13 – Gráfico de erro por iterações para os experimentos com a rede neural e o autoencoder empilhado. 52

Lista de tabelas

Tabela 1 – Resumo dos resultados empíricos para o conjunto de dados MNIST. . . 28

Tabela 2 – Resultados para a Rede Neural 46
Tabela 3 – Resultados para o Autoencoder + Rede Neural 47
Tabela 4 – Tabela com os resultados para Autoencoders Empilhados 50
Tabela 5 – Tabela com os resultados para o Autoencoder sem a utilização do parâmetro de ruído 51

Lista de abreviaturas e siglas

KL	*Kullback-Leibler*
MNIST	*Mixed National Institute of Standards and Technology database*
PCA	Análise por Componentes Principais
TTS	*Text to Speech*

Sumário

1 Introdução 23
1.1 Aprendizado de Máquina 23
1.1.1 Definição Formal 23
1.1.2 Aprendizado de Máquina Supervisionado 23
1.1.3 Desafios Quanto à Construção de Modelos 25
1.2 Aprendizado Automático de Características 26
1.3 Objetivos e Metodologia 28
1.4 Resumo dos Resultados 28
1.5 Principais Contribuições 29
1.6 Organização da Dissertação 29

2 Introdução aos Autoencoders Esparsos 31
2.1 Perceptron 31
2.2 Redes Neurais de Múltiplas Camadas 32
2.3 Backpropagation 34
2.4 Autoencoders Esparsos 37
2.5 Denoising Autoencoders 39

3 Metodologia 41
3.1 Parâmetros do Autoencoder 41
3.2 Parâmetros da Rede Neural 41
3.3 Geração de Características do Autoencoder para a Rede Neural 42
3.4 Validação dos modelos e Métricas 42

4 Experimentos 45
4.1 Classificação de Dígitos Manuscritos 45
4.2 O Conjunto de Dados MNIST 45
4.3 Resultados 46

5 Conclusões 53
5.1 Resumo dos Resultados 53
5.2 Principais Contribuições 53
5.3 Trabalhos Futuros 54

Referências 55

1 Introdução

Neste capítulo, introduzimos o surgimento e definição de Aprendizado de Máquina; as formas de aprendizado para construção de modelos; os desafios relativos à construção destes; técnicas para aprendizado automático de características; um resumo dos resultados obtidos com este trabalho; as principais contribuições deste trabalho e por fim, a organização desta monografia.

1.1 Aprendizado de Máquina

1.1.1 Definição Formal

Aprendizado de Máquina, é uma sub-área da inteligência artificial dedicada ao desenvolvimento de técnicas e algoritmos que permitam que que um computador aprenda. Esta área foi inicialmente abordada por Arthur Samuel, um pioneiro no campo de jogos de computador e inteligência artificial. Samuel desenvolveu um programa de Xadrez nos anos 50, que é considerado o primeiro programa de auto-aprendizado do mundo. Samuel também implementou uma variação da heurística alfa-beta (SAMUEL, 1959). Uma das primeiras definições de aprendizado de máquina, também veio de Samuel, como descrito a abaixo:

Definição: O campo de estudo que dá aos computadores a capacidade de aprender sem serem explicitamente programados.

Mais tarde, Tom M. Mitchell, cientista da computação e professor da Universidade Carnegie Mellon, cunhou uma definição (MITCHELL, 1997) mais completa e formal sobre o tema, como descrita a seguir:

Definição: Um programa de computador é dito para aprender com a experiência E com relação a alguma classe de tarefas T e medida de desempenho P, se o seu desempenho em tarefas em T, medido por P, melhora com a experiência E.

Em outras palavras, podemos dizer que um programa de computador aprende a resolução de uma tarefa quando este, a partir de experiências passadas, consegue melhorar seu desempenho.

1.1.2 Aprendizado de Máquina Supervisionado

Existem diversos tipos de abordagens associadas ao Aprendizado de Máquina, dentre elas, podemos destacar o aprendizado supervisionado, de onde tiraremos base para fundamentar os aspectos teóricos da proposta presente neste trabalho.

O aprendizado supervisionado é um tipo de tarefa de aprendizado de máquina, onde o conjunto de exemplos de treinamento possui identificação e é utilizado para a predição de uma função com base nesses dados. Essa função será utilizada para mapear novos exemplos, indicando o que os mesmos representam no conjunto de dados (MITCHELL, 1997).

Utilizaremos $x^{(i)}$ para representar as variáveis, ou características, de entrada e $y^{(i)}$ para representar as variáveis alvo que tentaremos predizer. O par $(x^{(i)}, y^{(i)})$ é chamado de exemplo de treinamento, e uma lista com m exemplos de treinamento $(x^{(i)}, y^{(i)}); i = 1, ..., m$ é chamada de conjunto de treinamento. Por fim, X representará o espaço de valores de entrada e Y o espaço de valores de saída.

Formalmente podemos dizer que em aprendizado supervisionado nosso objetivo é, dado um conjunto de dados de treinamento, estimar uma função $h : X \mapsto Y$ de modo que a função $h(x)$, chamada de hipótese, seja um bom preditor para o valor correspondente de y (NG, 2000).

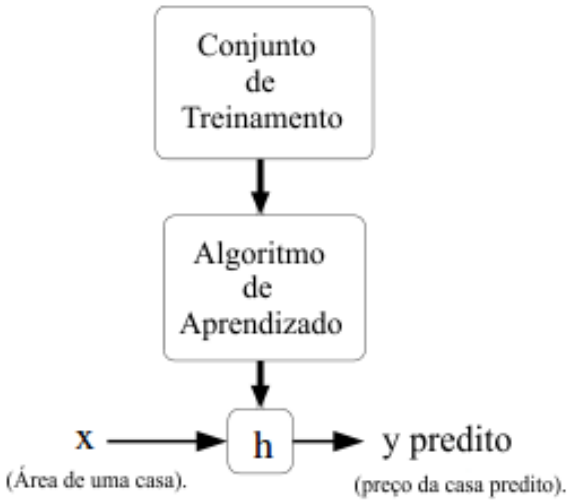

Figura 1: Demonstração gráfica de como ocorre o aprendizado supervisionado. Baseada em (NG, 2000)

Tarefas como classificação e regressão são tidas como problemas de aprendizado supervisionado (MOHRI; ROSTAMIZADEH; TALWALKAR, 2012). Quando a variável alvo que estamos tentando predizer é contínua, chamamos nosso problema de problema de regressão, enquanto quando nossa variável pode receber apenas uma pequena quantidade de valores discretos, o chamamos de problema de classificação (NG, 2000).

Costuma ser conveniente representar a função h utilizando uma função de pontuação $f : X \times Y \mapsto R$ tal que h é definido como o retorno do valor y que dá a maior pontuação: $g(x) = argmax_y f(x, y)$, sendo assim f denota o espaço de funções de pontuação.

Algoritmos de aprendizagem são divididos entre discriminativos e generativos, onde classificadores generativos aprendem um modelo de probabilidade conjunta $p(x, y)$, onde x representa a entrada e y sua identificação, usando a regra de Bayens para realizar suas predicções calculando $p(x|y)$ que o leva a escolher a identificação mais próxima y. Já classificadores discriminativos modelam o posterior $p(y|x)$ diretamente, ou aprendem um mapa direto das entradas x para os rótulos de identificação (JORDAN, 2002).

1.1.3 Desafios Quanto à Construção de Modelos

A construção de um bom modelo nem sempre é uma tarefa simples, sobretudo quando levamos em consideração certos fatores como a quantidade e a qualidade das variáveis que representam o problema.

Quando tratamos de atributos *versus* características, podemos ilustrar bem a situação com o caso da classificação de dígitos. Suponha duas imagens mostrando a letra A onde a única diferença é que na segunda imagem a mesma se encontra deslocada à direita. Se treinarmos um classificador que utiliza os dados brutos da imagem (*pixels*) e a primeira imagem estiver em um conjunto de treino e a segunda em um conjunto de teste, há grandes chances do classificador não reconhecer a letra A na segunda imagem. Isto porque, um simples deslocamento na imagem faz com que os valores dos pixels mudem completamente. Porém, se utilizarmos características que auxiliem a diferenciar uma letra A das demais letras também auxiliem a identificar as mesmas letras, o deslocamento em *pixels* não afetará tanto no desempenho do classificador. Um exemplo de característica é a distância entre as bordas da letra A; informações da curvatura e demais que são, por exemplo, características bem distintas da letra I.

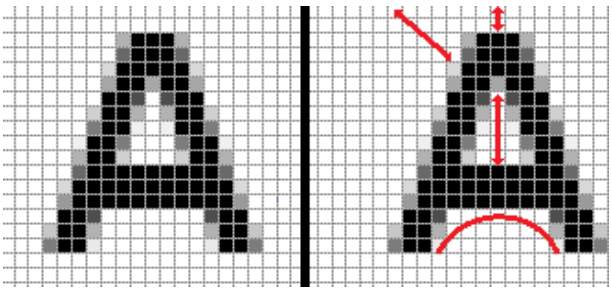

Figura 2: Imagem ilustrativa ao exemplo de classificação de dígitos. As setas vermelhas indicam as distâncias e curvaturas características da letra A

Vejamos um exemplo com imagens. Na prática, como ilustrado na figura 3, desejamos, a partir de atributos (pixels), transformar gradativamente estes em representações de mais alto nível, denominadas características. No exemplo da figura, se tivéssemos várias camadas de representação, a generalização do modelo seria maior para o aprendizado não-supervisionado. Isto porque, há maior chance de alguns destes serem informativos para uma determinada função objetivo desejada.

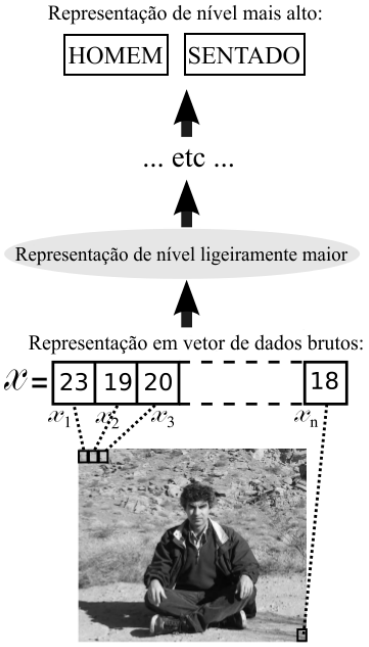

Figura 3: Um exemplo de como uma imagem pode ser gradualmente transformada em representações mais abstratas à partir de dados brutos. Adaptada de (BENGIO, 2009)

Desta forma, é importante ressaltar aqui a vantagem da utilização de características de qualidade, uma vez que aumentam o poder de generalização do modelo e contribuem para a redução da quantidade de atributos, acarretando em um melhor desempenho computacional. Entretanto, se estas forem desenvolvidas por um especialista no domínio, acaba por ser demasiadamente oneroso.

1.2 Aprendizado Automático de Características

Uma alternativa para a falta de conhecimento sobre o domínio é o Aprendizado Automático de Características, apresentado no presente trabalho. Dado um conjunto de atributos brutos para o problema, é possível produzir características utilizando Aprendizado Não-Supervisionado.

Além das vantagens dessa forma de aprendizado, devemos levar em consideração que a qualidade do modelo pode ser inferior à qualidade de modelos que utilizam características produzidas por especialistas do domínio. Porém, é uma alternativa a um custo menor, sem especialista. Entretanto, há casos em que podemos obter melhora dos resultados (BETECHUOH; MARWALA; TETTEY, 2006), como explicaremos adiante.

Existem diversas técnicas para tratar o problema de Aprendizado de Atributos Não-Supervisionado. Um exemplo é o método de Análise por Componentes Principais (PCA) (PEARSON, 1901). Este é um método muito comum na literatura e eficiente. PCA gera, a partir de um conjunto de variáveis possivelmente correlacionadas, um conjunto de

variáveis não-correlacionadas. Estas, são denominadas as componentes principais. que apontam para as direções de maior variância dos dados. Porém, a desvantagem é que este método é linear e para muitos problemas reais, é comum a não-linearidade entre as variáveis que representam o problema.

Entretanto, o PCA ainda é muito útil quando o número de atributos é consideravelmente elevado. Isto porque sintetiza o conjunto de atributos em poucas variáveis que podem ser utilizadas como características para uma rede neural. A redução do número de atributos impacta consideravelmente no desempenho do algoritmo de treinamento de uma Rede Neural.

Uma alternativa com relação ao PCA, é o Autoencoder. A grande vantagem deste com relação ao PCA, é que este não é linear. Adiante, damos um passeio pela literatura a respeito da aplicação de Autoencoders para várias tarefas distintas e indicamos sua relevância no contexto.

Em (HINTON; SALAKHUTDINOV, 2006), os autores apresentam um autoencoder como um método para, dado um conjunto de dados de grande dimensionalidade, reduzir a dimensionalidade destes dados e reconstruí-los com base neles mesmos. A técnica faz o uso de uma rede neural de múltiplas camadas, onde a camada oculta da rede possui poucos neurônios existindo uma etapa de pré-treino para detecção de features e outras para refinamento de dados obtidos. Com base nos resultados, foi constatado que a reconstrução de dados foi muito melhor do que a vista em aplicações de PCA.

Em (BETECHUOH; MARWALA; TETTEY, 2006), foi proposto um método para identificar HIV utilizando autoencoders e algoritmos genéticos. O conjunto de dados testado possuía propriedades demográficas dos indivíduos obtidos em uma pesquisa de pré-natal na África do Sul. A utilização do Autoencoder, que apresentou taxa de acerto de 92% para identificação de HIV, acarretou em uma melhora de $9,5\%$ se comparado à redes neurais convencionais (84 %), bem como a área sob a curva COR do modelo proposto, que foi de $0,86$, enquanto em redes neurais convencionais foi de $0,8$.

Em (VISHNUBHOTLA; FERNANDEZ; RAMABHADRAN, 2010), os autores abordam o problema de geração sons de fala a partir de texto, *Text to Speech* (TTS). Neste trabalho, os autores propuserem melhorar uma tradicional abordagem que utiliza um Modelo Oculto de Markov, a partir da redução de dimensionalidade do espaço de atributos e consequente geração de características por Autoencoders. Foi feito um teste de audição real, sendo mostrado que o modelo produz fala de maior qualidade perceptiva scom nível de significância de $p < 0,01$.

Em (LE et al., 2012), os autores treinaram um Autoencoder Esparso de 9 camadas para detecção de rostos com um conjunto de dados composto por 10 milhões de imagens de 200x200 *pixels*. Os resultados revelaram que esse modelo de detector de características

é robusto e é possível treinar um reconhecedor de rostos sem a necessidade de rotular as imagens como contendo um rosto ou não. Outro ponto importante, foi que o experimento também se mostrou eficiente em reconhecimento de outros conceitos de alto nível, como gatos e partes do corpo humano.

Desta forma, consideramos que Autoencoder Esparso é um tema recente e promissor.

1.3 Objetivos e Metodologia

Neste trabalho propomos evidenciar a importância de Autoencoders Esparsos para construção automática de características de problema de classificação sem a necessidade de um especialista para o domínio do problema. Em específico, nosso problema de classificação corresponde ao reconhecimento de dígitos manuscritos em imagens digitais. Para isto, aplicamos Autoencoders Esparsos na etapa de preprocessamento do dados para gerar de forma automática características para uma Rede Neural. Como comparativo (*baseline*), utilizamos uma Rede Neural de três camadas. O algoritmo de treinamento utilizado para ambas as estratégias é o *Backpropagation* via Gradiente Descendente.

1.4 Resumo dos Resultados

Na tabela 1, apresentamos um resumo dos resultados para a tarefa de classificação de dez dígitos para o conjunto de dados MNIST que possui 70000 imagens de 28x28 pixels. A primeira coluna da tabela representa o método utilizado. No caso, a rede neural corresponde ao *baseline* adotado e o autoencoder ao nosso melhor modelo autoencoder obtido a partir da análise dos resultados empíricos.

Método	Erro (%)
Autoencoder Esparso e Rede Neural	1,44%
Rede Neural	1,77%

Tabela 1: Resumo dos resultados empíricos para o conjunto de dados MNIST.

Aqui, evidenciamos que, a aplicação do Autoencoder no preprocessamento da rede neural apresenta uma taxa de erro na classificação de dígitos manuscritos de $1,44\%$, o que equivale a $98,56\%$ de taxa de acerto. Uma redução do erro de classificação de $20,33\%$ com relação à utilização de uma rede neural de três camadas, cuja taxa de erro corresponde a $1,77\%$. Desta forma, já evidenciamos aqui, a relevância da técnica.

1.5 Principais Contribuições

Nesta seção, buscamos elucidar sucintamente, a partir de itens, as principais contribuições do presente trabalho.

- *Um estudo sobre Autoencoders Esparsos*

  O presente trabalho contribui como material didático acerca de Autoencoders Esparsos, contextualizando com trabalhos relevantes da literatura. O estudo torna-se ainda mais importante se considerada a escassez de material didático na língua portuguesa para este assunto.

- *Análise empírica dos resultados*

  Uma segunda contribuição deste trabalho se dá pelo indicativo de uma proposta de utilização da técnica de autoencoders com intuito de melhorar os resultados de uma rede neural padrão com três camadas. Adicionalmente, a partir de resultados empíricos, pretendemos convencer o leitor das vantagens do aprendizado automático de características do problema, pois este não requerem da utilização de um especialista do domínio.

- *Metodologia*

  Por fim, apresentamos uma metodologia empírica para validação dos modelos de autoencoders e rede neural de forma a auxiliar o leitor ao aprimoramento da utilização destes e obter resultados expressivos com a aplicação de autoencoders esparsos.

1.6 Organização da Dissertação

Este trabalho é organizado da seguinte forma: No capítulo 1, apresentamos uma breve introdução sobre Aprendizado de Máquina; aprendizado supervisionado; desafios quanto a construção de modelos de qualidade; revisamos os principais trabalhos relacionados à Autoencoders Esparsos; apresentamos nossos objetivos, resumo dos resultados e contribuições. No capítulo 2, apresentamos a fundamentação teórica para entendimento de Autoencoders Esparsos. Neste apresentamos perceptrons, redes neurais, algoritmo backpropagation e por fim, os autoencoders esparsos e variações. No capítulo 3, apresentamos a metodologia adotada no presente trabalho. Nesta, indicamos ao leitor como ajustamos os parâmetros dos nossos modelos; validação dos modelos e métricas de erro. No capítulo 4, apresentamos o conunto de dados MNIST; os experimentos realizados e uma an;alise dos resultados obtidos. Por fim, no capítulo 5, apresentamos nossas conclusões e trabalhos futuros.

2 Introdução aos Autoencoders Esparsos

Neste capítulo apresentaremos a fundamentação teórica sobre redes neurais e autoencoders esparsos, que darão base para o entendimento da implementação utilizada para os experimentos.

2.1 Perceptron

Um tipo simples de Rede Neural é o Perceptron que é baseado em apenas um neurônio (MITCHELL, 1997). Um Perceptron recebe um vetor de valores reais, calcula a combinação real dessas entradas e retorna 1 se o resultado é maior do que um *threshold* e -1 caso contrário.

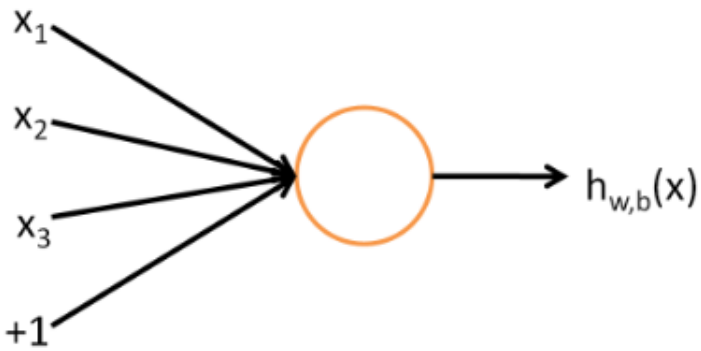

Figura 4: Perceptron (NG, 2011)

Dado entradas x_1 até x_n, a saída $o(x_1, ..., x_n)$ computada pelo Perceptron é

$$o(x_1, ..., x_n) = \begin{cases} 1 & \text{se } w_0 + w_1x_1 + w_2x_2 + ... + w_nx_n > 0 \\ -1 & \text{caso contrário} \end{cases}$$

onde cada w_i é uma constante real, ou *peso*, que determina a contribuição da entrada x_i para o valor de saída (MITCHELL, 1997).

Para simplificar a notação, podemos imaginar uma entrada constante adicional $x_0 = 1$, permitindo escrever esta inequação como $\sum_{i=0}^{n} w_i x_i > 0$, ou um vetor no formado $\vec{w}.\vec{x} > 0$. Podemos então, escrever a função do Perceptron como

$$o(\vec{x}) = sgn(\vec{w}.\vec{x})$$

onde

$$sgn(y) = \begin{cases} 1 & se\ y > 0 \\ -1 & caso\ contrário \end{cases}$$

A aprendizagem do Perceptron envolve escolher valores para os pesos $w_0, ..., w_n$. Dessa forma, o espaço de hipóteses candidatas H consideradas durante a aprendizagem do perceptron é um conjunto de todos os valores reais possíveis para o vetor de pesos.

$$H = \{\vec{w} | \vec{w} \in R^{(n+1)}\}$$

Para realizar a aprendizagem do perceptron é necessário iniciar o vetor de pesos com valores aleatórios, posteriormente aplicando a função de ativação para cada exemplo de treinamento, modificando os pesos do Perceptron toda vez que a saída do mesmo não estiver de acordo com os alvos.

$$w_i \leftarrow w_i + \Delta\ w_i \tag{2.1}$$

onde

$$\Delta\ w_i = \eta(t - o)x_i \tag{2.2}$$

Nas fórmulas 2.1 e 2.2, t é o conjunto de alvos para os exemplos de teste utilizados, enquanto o é a saída gerada pelo perceptron e η é uma constante positiva chamada taxa de aprendizado, responsável pela regulação da taxa na qual os pesos são alterados a cada passo do algoritmo (NG, 2011).

2.2 Redes Neurais de Múltiplas Camadas

Um perceptron é capaz apenas de expressar classes linearmente separáveis. Para que seja possível expressar classes que não sejam linearmente separáveis, precisaremos de um tipo diferente de determinação do *thresold* (MITCHELL, 1997).

Uma possível solução para esse problema é a utilização de uma diferente função de ativação, a função *sigmoid*.

$$f(z) = \frac{1}{1 + exp(-z)}$$

Uma rede neural de múltiplas camadas costuma fazer uso da função *sigmoid*, além de múltiplos neurônios e uma, ou mais, camadas ocultas. Podemos enxergar uma rede

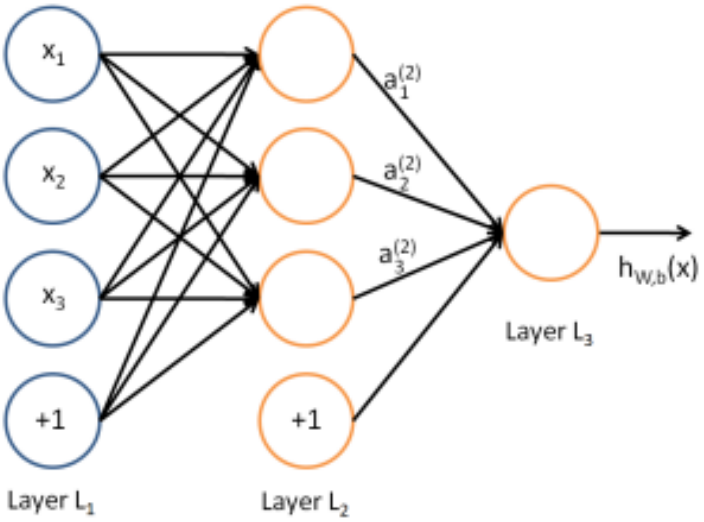

Figura 5: Exemplo de Rede Neural de Múltiplas Camadas com 3 neurônios na camada de entrada, 3 na camada oculta e 1 na camada de saída(NG, 2011).

neural de múltiplas camadas como diversos dos nossos simples neurônios conectados de modo que a saída de um neurônio pode ser a entrada para outro (Figura 5) (NG, 2011).

Podemos notar que existem entradas que recebem o valor constante "+1", que são chamadas de *bias*. Mais à esquerda estão as unidades que correspondem à camada de entrada da rede, enquanto, mais à esquerda está sua camada de saída, que, nesse caso, possui apenas um neurônio.

Seja n_l o número de camadas em nossa rede neural e L_l as camadas, temos que $n_l = 3$, L_1 é a camada referente às entradas e L_{nl} é a camada referente às saídas. A rede possui com parâmetros $(W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)})$, onde $W_{ij}^{(l)}$ denota os pesos associados às conexões entre o neurônio j na camada l e o neurônio i na camada $l+1$, $b_i^{(l)}$ é o *bias* associado ao neurônio i na camada $l+1$ e s_l denota o número de neurônios na camada l. Neste exemplo temos $W^{(1)} \in R^{3\times3}$ e $W^{(2)} \in R^{1\times3}$.

Utiliza-se $a_i^{(l)}$ para denotar a ativação do neurônio i na camada l. Para $l = 1$ teremos que utilizar também $a_i^{(1)} = x_i$ para denotar a i-ézima entrada. Dado um número fixo de parâmetros W,b, nossa rede neural define uma hipótese $h_{W,b}(x)$ que gera um número real. A computação que essa rede neural representa é dada por:

$$a_1^{(2)} = f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}) \tag{2.3}$$

$$a_2^{(2)} = f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)}) \tag{2.4}$$

$$a_3^{(2)} = f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)}) \tag{2.5}$$

$$h_{W,b}(x) = a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + f(W_{12}^{(2)}a_2^{(2)} + f(W_{13}^{(2)}a_3^{(2)} + b_1^{(2)}) \tag{2.6}$$

Seja $z_i^{(l)}$ a soma de todos os pesos das entradas com o neurônio i na camada l, incluindo o bias, então $a_i^{(l)} = f(z_i^{(l)})$.

Se estendermos a função de ativação $f(.)$ para aplicar em vetores elemento a elemento, podemos reescrever as equações 2.3-2.6 de forma mais compacta:

$$z^{(2)} = W^{(1)}x + b^{(1)}$$

$$a^{(2)} = f(z^{(2)})$$

$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$

$$h_{W,b}(x) = a^{(3)} + f(z^{(3)})$$

Generalizando:

$$z^{(l+1)} = W^{(1)}a^{(l)} + b^{(l)}$$

$$a^{(l+1)} = f(z^{(l+1)})$$

Para treinar essa rede precisaremos de exemplos de treinamento $(x^{(i)}, y^{(i)})$ onde $y^{(i)} \in R^2$ (NG, 2011).

2.3 Backpropagation

O algoritmo Backpropagation (abreviação de *backward propagation of errors*) é um método comum para treinamento de redes neurais artificiais. Este algoritmo aprende os pesos para uma rede neural de múltiplas camadas, dada uma rede com um conjunto fixo de unidades e interligações. Também emprega a regra de atualização de pesos Gradiente Descendente para minimização do erro quadrático entre os valores da saída da rede e os valores alvo desejados. (MITCHELL, 1997)

Suponha que tenhamos exemplos de treinamento de tamanho fixo m, $(x^{(1)}, y^{(1)}, ..., (x^m, y^{(m)})$. Podemos treinar uma rede neural utilizando o algoritmo do Gradiente Descendente. Para um único exemplo de treinamento (x, y), definimos a função de custo:

$$J(W, b; x, y) = \frac{1}{2}||h_{W,b}(x) - y||^2$$

Esta é a função do erro quadrático. Dado um conjunto de exemplos de treinamento de m exemplos, podemos definir a função de custo total.

$$J(W, b) = \left[\frac{1}{m}\sum_{i=1}^{m} J(W, b; x^{(i)}, y^{(i)})\right] + \frac{\lambda}{2}\sum_{l=1}^{n_{l-1}}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2 =$$

$$= \left[\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}||h_{W,b}(x^{(i)}) - y^{(i)}||^2\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{n_{l-1}}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2 \tag{2.7}$$

O primeiro termo é a média da soma do erro quadrático, enquanto o segundo termo é o termo de regularização que tende a reduzir a magnitude dos pesos e ajuda a prever o *overfitting*. A variável λ é chamada de *weight decay parameter* e controlará a importância relativa entre os dois termos.

Nosso objetivo é minimizar $J(W, b)$ como uma função de W e b. Para treinar a rede neural, cada parâmetro $W_{ij}^{(l)}$ e $b_i^{(l)}$ será inicializado com um valor randômico próximo de zero e então aplicando um algoritmo de otimização como o do Gradiente Descendente.

Uma iteração do Gradiente Descendente atualiza os parâmetros W, b da seguinte maneira:

$$W_{ij}^{(l)} := W_{ij}^{(l)} - \alpha\frac{\partial}{\partial W_{ij}^{(l)}}J(W, b)$$

$$b_i^{(l)} := b_i^{(l)} - \alpha\frac{\partial}{\partial b_i^{(l)}}J(W, b)$$

onde α é a taxa de aprendizagem.

O desafio é computar as variáveis parciais da equação acima, o que pode nos ser proporcionado de forma eficiente pelo uso do algoritmo *Backpropagation*.

Dado um exemplo de treinamento (x, y), deve-se computar todas as ativações da rede neural incluindo a saída da hipótese $h_{W,b}(x)$. Então, para cada nó i na camada l devemos computar o termo de erro $\delta_i^{(l)}$ que mede o erro que o nó causa na saída. Para cada nó de saída, $\delta_i^{(n_l)}$, onde n_l é a camada de saída, é possível definir a diferença entre a saída computada da rede e o valor esperado. Para os neurônios da camada intermediária computaremos $\delta_i^{(l)}$ com base na média ponderada dos termos de erro dos nós que utilizam $a_i^{(l)}$ como entrada (NG, 2011). A seguir, é apresentado o algoritmo *Backpropagation*.

1. Computar todas as ativações para cada camada da rede neural, inclusive a camada de saída L_n^1.

2. Para cada saída i na camada n_l, atribua

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}}\frac{1}{2}||y - h_{W,b}(x)||^2 = -(y_i - a_i^{(n_l)}).f'(z_i^{(n_l)})$$

3. Para $l = n_l - 1, n_l - 2, n_1 - 3, ..., n$

Para cada nó i na camada l, atribua

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)}\delta_j^{(l+1)}\right)f'(z_i^{(l)})$$

4. Compute as derivadas parciais desejadas, dadas por:

$$\frac{\partial}{\partial b_{ij}^{(l)}} J(W,b;x,y) = a_j^{(l)} \delta_i^{(l+1)}$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W,b;x,y) = \delta_i^{(l+1)}.$$

Podemos reescrever o algoritmo utilizando uma notação matriz-vetorial.

1. Computar todas as ativações para cada camada da rede neural utilizando as Equações (2.16-17).

2. Para a camada de saída (n_l), atribua

$$\delta^{(n_l)} = -(y - a^{(n_1)}) . * f'(z^{(n)})$$

3. Para $l = n_l - 1, n_l - 2, n_1 - 3, ..., n$ atribua

$$\delta^{(l)} = ((W^{(l)})^{T} \delta^{(l+1)}) . * f'(z^{(l)})$$

4. Compute as derivadas parciais desejadas:

$$\nabla_W^{(l)} J(W,b;x,y) = \delta^{(l+1)} (a^{(l)})^T,$$

$$\nabla_b^{(l)} J(W,b;x,y) = \delta^{(l+1)}.$$

Por fim, escreveremos o algoritmo do Gradiente Descendente.

1. Atribua $\Delta W^{(l)} := 0, \Delta b^{(l)} := 0$ para todo l.

2. Para $i = 1$ até m,

2a. Utilize o *Backpropagation* para computar $\nabla_W^{(l)} J(W,b;x,y)$ e $\nabla_b^{(l)} J(W,b;x,y)$

2b. Atribua

$$\Delta W^{(l)} := \Delta W^{(l)} + \nabla_W^{(l)} J(W,b;x,y)$$

.

2c. Atribua

$$\Delta b^{(l)} := \Delta b^{(l)} + \nabla_b^{(l)} J(W,b;x,y)$$

.

3. Atualize os parâmetros:

$$W^{(l)} := W^{(l)} - \alpha \left[\left(\frac{1}{m} \Delta W^{(l)} \right) + \lambda W^{(l)} \right]$$

$$b^{(l)} := b^{(l)} - \alpha \left[\left(\frac{1}{m} \Delta b^{(l)} \right) \right]$$

Para treinar uma rede neural, podemos repetir os passos do algoritmo do Gradiente Descendente a fim de minimizar a função de custo $J(W;b)$ (NG, 2011).

2.4 Autoencoders Esparsos

Suponha que temos um conjunto de exemplos de treinamento sem identificação $x^{(1)}, x^{(2)}, x^{(3)}, ...$, onde $x^{(i)} \in R^n$. um autoencoder é um algoritmo de aprendizagem não supervisionada que aplica o *Backpropagation* atribuindo os valores iguais nas entradas e nos conjunto das saídas esperadas (NG, 2011).

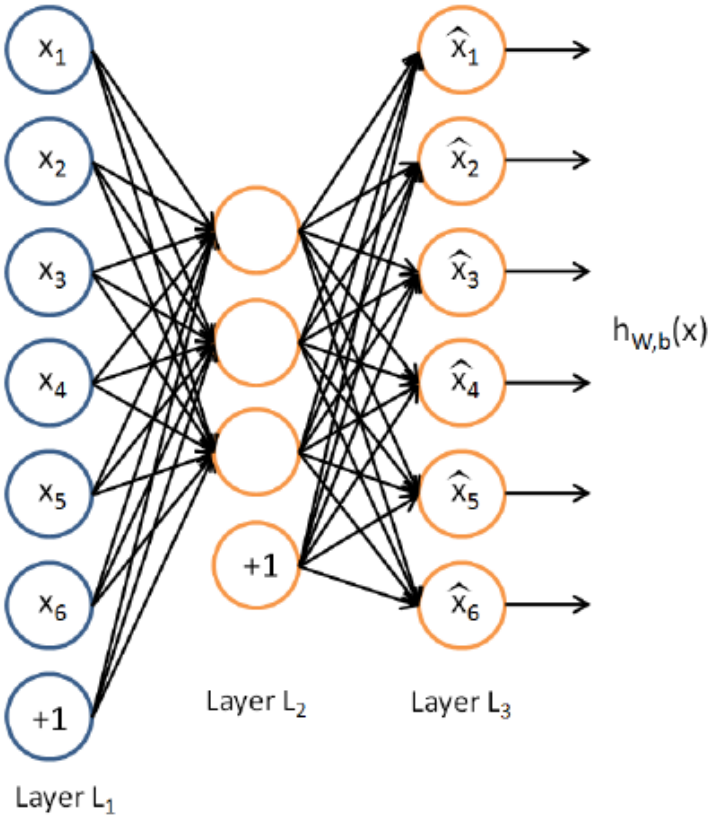

Figura 6: Exemplo de Autoencoder, com o número de entradas igual ao número de saídas. (NG, 2011)

[1]

O autoencoder tenta aprender uma função $h_{W,b}(x) \approx x$, ou seja, tenta aprender uma aproximação da função identidade, sendo que as saídas serão similares às entradas ao fim do processo.

Um autoencoder acaba servindo, de alguma forma, para compactar os dados existentes no dataset utilizado para a alimentação da rede. Limitando o número de neurônios da camada oculta podemos reduzir o número de features do dataset original.

Como exemplo, suponha que as entradas x são os valores de intensidade de pixel de uma imagem 10 x 10 (100 pixels), dessa forma $n = 100$, e teremos $s_2 = 50$ neurônios na camada oculta L_2. Tendo apenas 50 neurônios na camada oculta, a rede é forçada a aprender uma representação compactada da entrada. Caso as entradas sejam completamente aleatórias, ou seja, cada uma das características sejam independentes umas das outras, a compactação de dados será bem complicada. Entratando, se houver alguma estrutura

nos dados, como por exemplo, características correlacionadas, o autoencoder será capaz de descobrir essas correlações.

Lembrando que $a_j^{(2)}$ representa a ativação do neurônio j da camada oculta no autoencoder, podemos notar que essa notação não torna explícito que o input x que levou à esta ativação. Escreveremos $a_j^{(2)}(x)$ para denotar a ativação desse neurônio da camada oculta quando a rede receber uma entrada específica x. Sendo assim

$$\hat{\rho}_j = \frac{1}{m}\sum_{i=1}^{m}\left[a_j^{(2)}(x^{(i)})\right]$$

será a ativação média para cada neurônio j da camada oculta.

Colocaremos uma restrição

$$\hat{\rho}_j = \rho,$$

onde ρ é um **parâmetro de esparsidade**, um valor tipicamente próximo de zero. Esse parâmetro fará com que a ativação média de cada neurônio j da camada oculta seja próximo de ρ.

Entretando nada impediria que o valor se distanciasse de 0, por isso, adicionaremos um termo de penalidade extra, que penaliza $\hat{\rho}_j$ de desviar significantemente de ρ.

$$\sum_{j=1}^{s_2}\rho\ log\frac{\rho}{\hat{\rho}_j} + (1-\rho)\ log\frac{1-\rho}{1-\hat{\rho}_j}.$$

Sendo que s_2 é o número de neurônios na camada oculta e o índice j representa cada neurônio nessa camada. Esse termo de penalidade é baseado no conceito de divergência KL e também pode ser escrita da forma a seguir

$$\sum_{j=1}^{s_2} KL(\rho||\hat{\rho}_j),$$

onde $KL(\rho||\hat{\rho}_j) = \rho\ log\frac{1-\rho}{1-\hat{\rho}_j}$ é a divergência *Kullback-Leibler*.

Essa função de penalidade possui uma propriedade que $KL(\rho||\hat{\rho}_j) = 0$ se $\rho = \hat{\rho}_j$ e caso contrário, aumenta monotonicamente conforme $\hat{\rho}_j$ diverge de ρ.

Nossa função de custo total é agora

$$J_{sparse}(W,b) = J(W,b) + \beta\sum_{j=1}^{s_2} KL(\rho||\hat{\rho}_j),$$

onde $J(W,b)$ é como definido anteriormente e β controla o peso do termo penalidade de esparsidade. O termo $\hat{\rho}_j$ depende também de W,b, pois este termo é a ativação média do neurônio j da camada oculta, e sua ativação depende destes parâmetros W,b (NG, 2011).

2.5 Denoising Autoencoders

Denoising autoencoders são autoencoders que utilizam-se da técnica de corrompimento dos dados (*denoising*) para aumentar seu poder de generalização. Este corrompimento é feito aleatoriamente seguindo uma determinada distribuição de probabilidade. Uma das vantagens da corrupção dos dados é que esta técnica força com que a camada oculta do autoencoder não aprenda a função identidade dos dados.

Formalmente, o corrompimento dos dados pode ser feito na entrada inicial x para conseguir uma versão parcialmente destruída $\widetilde{x}$ por meio de um mapeamento estocástico $\widetilde{x} \sim q_D(\widetilde{x}|x)$. O interessante é que este processo de corrupção pode ser parametrizado (v) de forma a regular o ruído inserido. Desta forma, para todo x, um número fixo vd de componentes é escolhido aleatoriamente e seu valor forçado a 0, enquanto os demais se mantém intocados. A entrada corrompida x é então mapeada, como no autoencoder padrão, para a representação oculta $y = f_\theta(\widetilde{x}) = s(W\widetilde{x}+b)$ a partir da qual reconstruímos um $z = g_{\theta'}(y) = s(W'y + b')$.

Então os parâmetros são treinados para minimizar o erro de reconstrução médio $L_H(x, z) = H(B_x|B_z)$ sobre um conjunto de treinamento, isto é, ter z o mais próximo possível do conjunto de treinamento x, sendo que z, será uma função determinística de $\widetilde{x}$ ao invés de x, ou seja, será o resultado do mapeamento estocástico de x (VINCENT et al., 2008). Na figura 7, ilustramos este processo.

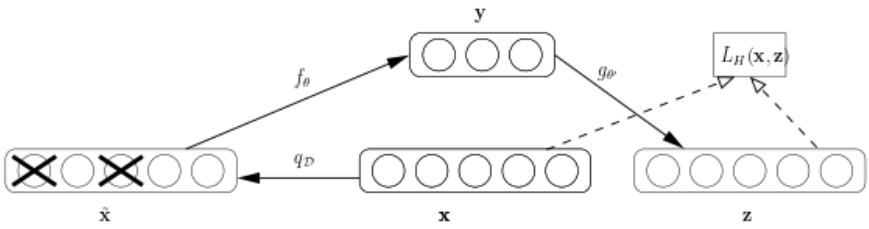

Figura 7: Um exemplo x é corrompido para $\widetilde{x}$. O autoencoder então, o mapeia para y e tenta reconstruir x (VINCENT et al., 2008).

Neste capítulo apresentamos uma breve introdução sobre redes neurais e autoencoders esparsos, que deram base para o entendimento da implementação utilizada para os experimentos. A seguir, apresentamos a nossa metodologia empírica para a condução dos experimentos com relação aos autoencoders esperos e redes neurais.

3 Metodologia

Neste capítulo descrevemos a metodologia utilizada para a condução dos experimentos, assim como os parâmetros de configuração do Autoencoder e da Rede Neural.

3.1 Parâmetros do Autoencoder

Para os experimentos com o autoencoder foi utilizada uma camada oculta com o número de neurônios variando de 25 a 600, com o valor do parâmetro de ruído de $0,3$. A taxa de aprendizado utilizada foi de $0,001$ para o pré-treinamento e $0,1$ para a etapa de refinamento, sendo que estes valores foram considerados ideais no balanceamento entre *overfitting* e *underfitting*. A função de ativação utilizada foi a função *sigmoid*, comum na literatura e considerada a que se obtém os melhores resultados.

Para os testes adicionais com duas e três camadas ocultas, foram utilizados os valores $0,1$, $0,2$ e $0,3$ para os parâmetros de ruído de cada respectiva camada. A quantidade de cada neurônio nas camadas foi de 500 ou 1000 para a primeira, 250 a 1000 para a segunda e terceira. Os demais parâmetros permaneceram inalterados.

Seu conjunto de dados de entrada e alvo são iguais conforme já descrito anteriormente e o número máximo de épocas de treinamento é 1000 pra possibilitar um treinamento adequado.

3.2 Parâmetros da Rede Neural

Para a rede padrão, que consiste em uma rede neural artificial de três camadas, utilizamos parâmetros semelhantes aos utilizados no autoencoder para que ocorrese uma comparação justa.

Para os experimentos com o rede neural padrão foram utilizadas uma camada oculta com o número de neurônios variando de 25 a 600. A taxa de aprendizado utilizada para o treinamento também foi de $0,1$, e conforme já explicado, este valor foi considerado ideal no balanceamento entre *overfitting* e *underfitting*. A função de ativação utilizada para as primeiras camadas foi a função *sigmoid*, comum na literatura e considerada a que se obtém os melhores resultados. Já a função utilizada para a última camada foi a função *softmax*, onde cada saída representa uma das classes. A classe escolhida será a que terá o maior valor como saída de seu respectivo neurônio.

O número máximo de épocas de treinamento é 1000 pra possibilitar um treinamento adequado.

3.3 Geração de Características do Autoencoder para a Rede Neural

A fase inicial do autoencoder é exatamente a predição de características com base nos dados brutos de um determinado dataset. Onde, por fim, geramos um novo dataset a ser utilizado na rede final.

Conforme já ilustrado, o conjunto de entrada é igual ao conjunto de resultado esperado, onde serão armazenados os valores de saída da camada oculta. Estes resultados constituirão o novo dataset, e também uma representação mais enxuta do dataset original, utilizado na próxima fase, fase de refinamento, do experimento.

3.4 Validação dos modelos e Métricas

Para validação dos modelos, todo o conjunto foi dividido em partições (*minibatches*). Assim, geramos *minibatches* para treino, validação e teste. O motivo para tal é que o treinamento da rede é demasiadamente custoso, e o treinamento em pequenas *minibatches* auxilia à resolução deste problema. Adicionalmente, optamos em minimizar o erro médio para todos os *minibatches* ao invés do erro global.

Desta forma, seja um dado número constante de épocas, para cada época, treinamos a rede ou autoencoder em um *minibatch* e decidimos, de acordo com uma frequência especificada à priori, se vamos avaliar ou não o erro geral para todos os *minibatches* de validação. Quando desejamos avaliar, comparamos com o menor erro de validação geral encontrado e guardamos as informações do modelo. Ao término de todas as épocas, avaliamos o desempenho do melhor modelo para todos os *minibatches* do conjunto de teste.

Com relação à métrica de erro final, adotamos o erro médio entre *minibatches*. No caso, um dado erro e_j para um exemplo j corresponde ao oposto da métrica de acurácia a_j para um dado exemplo j. Como descrito a seguir:

$$a_j = \begin{cases} 1 \text{ se } y_j^{'} = y_j, \\ 0 \text{ caso contrário.} \end{cases}$$

em que $y_j^{'}$ é a j-ésima estimativa do modelo e y_j o gabarito do exemplo j.

Na equação 3.1, descrevemos o erro médio e para todos os *minibatches*.

$$e = \frac{1}{k} \sum_{i=1}^{k} \sum_{j=1}^{m_i} (1 - a_j) \tag{3.1}$$

tal que k corresponde ao número de *minibatches* e m_i o número de exemplos do i-ésimo *minibatch*.

O decaimento do erro também foi salvo para cada experimento, onde o melhor modelo encontrado teve seu decaimento plotado e utilizado para demonstração de resultados no presente trabalho. No caso, o melhor modelo é o que obteve menor erro médio de classificação no conjunto de validação.

Cabe ressaltar que utilizamos a mesma métrica de erro adotada na literatura (CIRESAN et al., 2010), (DENG; YU, 2011), (SALAKHUTDINOV; HINTON, 2007). Desta forma, não utilizamos outras métricas de erro.

Neste capítulo descrevemos nossa metodologia para condução dos experimentos. Neste vimos os parâmetros a serem ajustados para o autoencoder e rede neural, assim como deve ser feita a validação dos modelos e métriac de erro utilizada. No capítulo seguinte, apresentamos nossos experimentos alinhados a esta metodologia.

4 Experimentos

Neste capítulo apresentamos a tarefa de classificação de dígitos manuscritos em imagens digitais. Logo em seguida apresentamos o conjunto de dados utilizado e por fim, nossos experimentos e análise dos resultados.

4.1 Classificação de Dígitos Manuscritos

Seja um conjunto de imagens de dígitos manuscritos anotados por humanos, tal que uma imagem qualquer pode pertencer a uma das dez classes de um a dez. Nosso objetivo consiste de, dado uma imagem sem sua devida classe associada, identificá-la corretamente apenas com informações dos *pixels* das imagens. A seguir descrevemos o conjunto de dados utilizado para os experimentos.

4.2 O Conjunto de Dados MNIST

Conhecido como MNIST (*Mixed National Institute of Standards and Technology Database*) (LECUN; CORTES, 1998), o conjunto de dados utilizado consiste em 70000 imagens de tamanho fixo (60000 para treinamento e 10000 para teste e validação) de $28x28$ pixels que representam dígitos escritos à mão, normalizados e centralizados.

Este conjunto de dados é parte de um conjunto ainda maior de nome NIST (*National Institute of Standards and Technology*), que é dividido entre SD-3, o conjunto de treino e SD-1, o conjunto de teste. O conjunto de treino do MNIST foi composto de 30000 padrões do SD-3 e 30000 padrões do SD-1, enquanto o conjunto de teste foi compostos por 5000 padrões do SD-3 e 5000 padrões do SD-1. O conjunto de treinamento possui exemplos de escrita de aproximadamente 250 pessoas.

Figura 8: Exemplo de dígitos contidos no conjunto de dados MNIST.

Neurônios HL	Erro Validação (%)	Erro Teste (%)	Tempo (min)
600	1,78	1,82	126,17
500	1,84	1,77	73,07
400	**1,76**	**1,77**	**149,53**
300	1,92	1,97	11,86
200	1,83	1,91	18,46
100	1,84	1,93	27,75
50	2,16	2,39	25,41
25	4,04	4,13	0,53

Tabela 2: Resultados para a Rede Neural

Na figura 8, podemos observar que o problema, em alguns casos, é complexo até mesmo para humanos. Como por exemplo na quarta linha e primeira coluna, é complicado de identificar se o dígito é um número dois ou um.

4.3 Resultados

Aqui, apresentamos os resultados dos vários experimentos com redes neurais com e sem processamento de autoencoders esparsos. Todos alinhados com a metodologia descrita no capítulo anterior. Primeiramente, como descrito na tabela 2 apresentamos os resultados de uma rede neural de três camadas, tanto para validação quanto para teste. Adicionalmente, informamos o tempo computacional. Como podemos observar na tabela 2, o melhor modelo da validação apresenta taxa de erro de $1,77\%$ para o conjunto de teste. Uma pequena diferença do resultado da validação, significando boa generalização do modelo. Adiconalmente, cabe ressaltar que um erro de $1,77\%$ no conjunto de teste, equivale a uma elevada acurácia de $98,23\%$. Com relação ao tempo computacional, o melhor modelo apresenta tempo bem mais elevado dos demais modelos.

Em seguida, na tabela 3, apresentamos os resultados obtidos por um autoencoder esparso simples. Aqui, o melhor modelo da validação apresenta taxa de erro de $1,76\%$ para o conjunto de teste, porém com uma grande diferença para o resultado da validação. Um possível indicativo de problema de generalização. Adicionalmente, verificamos o aumento do tempo computacional, próximo do tempo da rede neural simples, o que neste caso é bom, pois o preprocessamento está fazendo com que o algoritmo de treinamento da rede convirja mais rapidamente. Em termos de redução de erro $(0,56\%)$, a abordagem por autoencoders só causa resultados significativos para conjuntos de dados com muitos exemplos. Como problemas com dados reais apresentam muitos exemplos, este ligeiro ganho, em números absolutos pode ser significativo.

Neurônios HL	Erro Validação (%)	Erro Teste (%)	Tempo (min)
600	1,96	1,95	55,57
500	**1,70**	**1,76**	**159,53**
400	2,08	2,15	28,83
300	2,11	2,26	23,15
200	2,29	2,25	17,38
100	2,91	3,28	8,11
50	3,58	4,05	5,2
25	4,50	5,01	6,01

Tabela 3: Resultados para o Autoencoder + Rede Neural

Para melhor entendimento do leitor, extraímos os dados de números de neurônios e erro percentual no conjunto de validação, das tabelas 3 e 2. Podemos notar que a arquitetura da rede neural com três camadas mantém uma regularidade nos resultados a partir de 100 neurônios. Porém, os modelos de rede neural gerados com preprocessamento dos autoencoders esparsos, para que sejam efetivos, necessitam de um número elevado de neurônios, apresentando melhor resultado apenas em uma bateria de testes. Adicionalmente, para tirarmos a dúvida, ilustramos na figura 9, os resultados dos melhores modelos para o conjunto de validação. Como podemos notar, há um comportamento muito parecido entre as duas abordagens e apenas uma ligeira melhora dos resultados. As figuras 11 e 12 complementam o indicativo da figura 9, porém, individualmente para validação (azul) e teste (vermelho) em cada modelo. Aqui, podemos notar novamente que com poucos neurônios, a rede neural sem autoencoder apresenta bons resultados em comparação a utilização de autoencoders no preprocessamento.

Desta forma, concluímos que a utilização de uma passada de autoencoder como preprocessamento não nos indica uma melhora consistente dos resultados, o que nos leva à condução de experimentos com autoencoders esparsos empilhados. A seguir, conduzimos experimentos com dois empilhamentos e três empilhamentos de autoencoders esparsos. Para selecionar o melhor modelo, também variamos o número de neurônios em cada autoencoder que participa da pilha.

Na tabela 4, verificamos que a abordagem de empilhamento de autoencoders como preprocessamento da rede neural acarreta em melhora significativa de resultados. Quantitativamente, quase que todos os resultados de validação apresentaram erros mais baixos com relação ao *baseline*. Resultado que persiste no conjunto de teste. Se selecionarmos o melhor modelo, em negrito na tabela, da validação e comparamos com o nosso melhor modelo do *baseline*, conseguimos uma redução considerável do erro de $20,33\%$ e uma taxa de acerto de $98,56\%$. É claro que, devemos dar atenção ao tempo computacional, pois este foi de $617,99$ minutos, aproximadamente quatro vezes mais que o do nosso *baseline*.

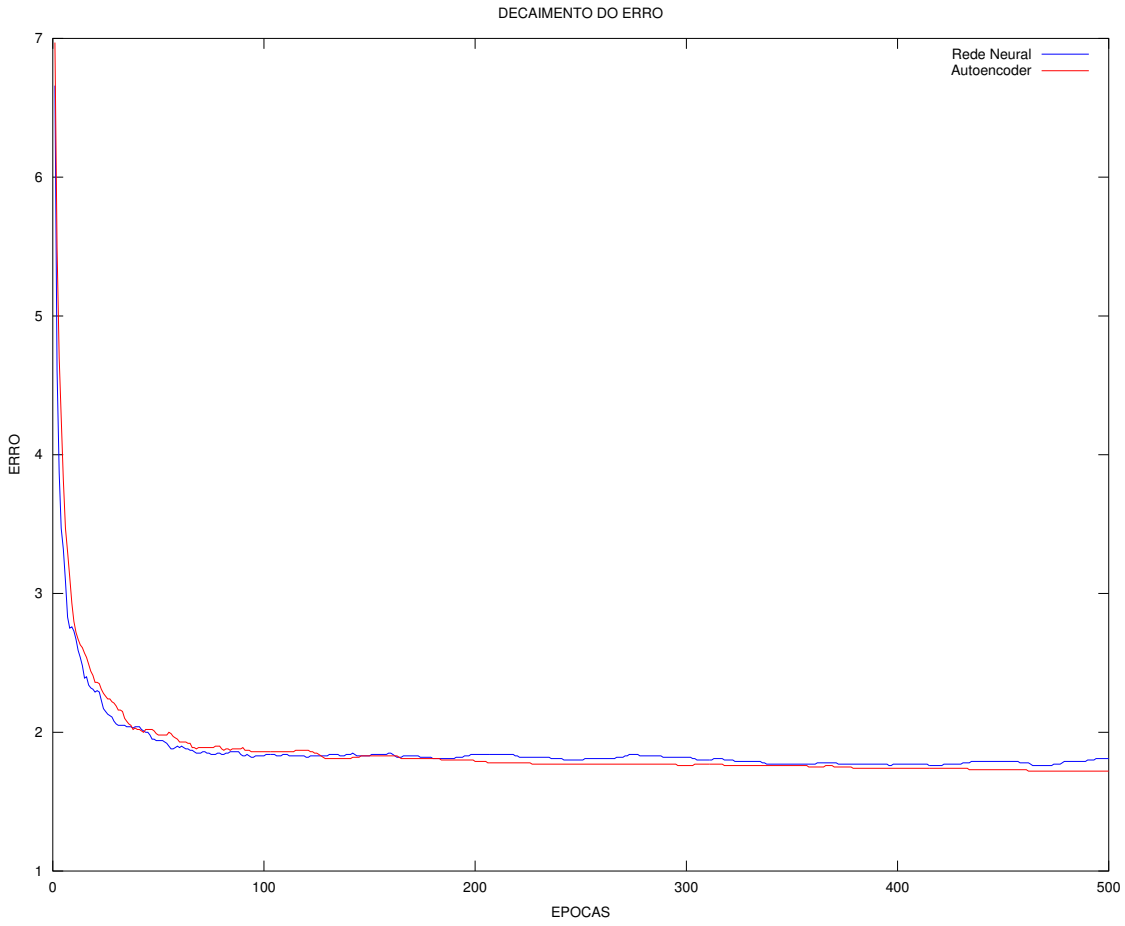

Figura 9: Gráfico de decaimento do erro de validação da rede neural (azul) e do autoencoder (vermelho).

Este fator, deve ser levado em consideração para problema de tempo real.

Na figura 13, podemos ilustramos os decaimentos dos erros de validação para o melhor modelo autoencoder empilhado e o *baseline*. É possível notar o rápido decaimento do erro do autoencoder empilhado se comparado à rede neural padrão, com uma área sob a curva significativamente inferior. Ressaltamos aqui que, as iterações posteriores foram retiradas para que fosse possível a observação mais detalhada da área indicada.

Por fim, ressaltamos que todos os modelos anteriores utilizaram a técnica de adição de ruído nos exemplos. No entanto, não verificamos seu impacto à qualidade dos modelos. Desta forma, fizemos três experimentos com preprocessmento dos autoencoders sem a adição de ruídos nos exemplos, como descritos na tabela 5. Como podemos observar, a não utilização dos parâmetros de ruído acarretam em um aumento relevante da porcentagem de erro do autoencoder. Uma piora em torno de 10%, o que reforça a aplicação da técnica combinada aos autoencoders.

Neste capítulo vimos que autoencoders esparsos, em especial os empilhados acar-

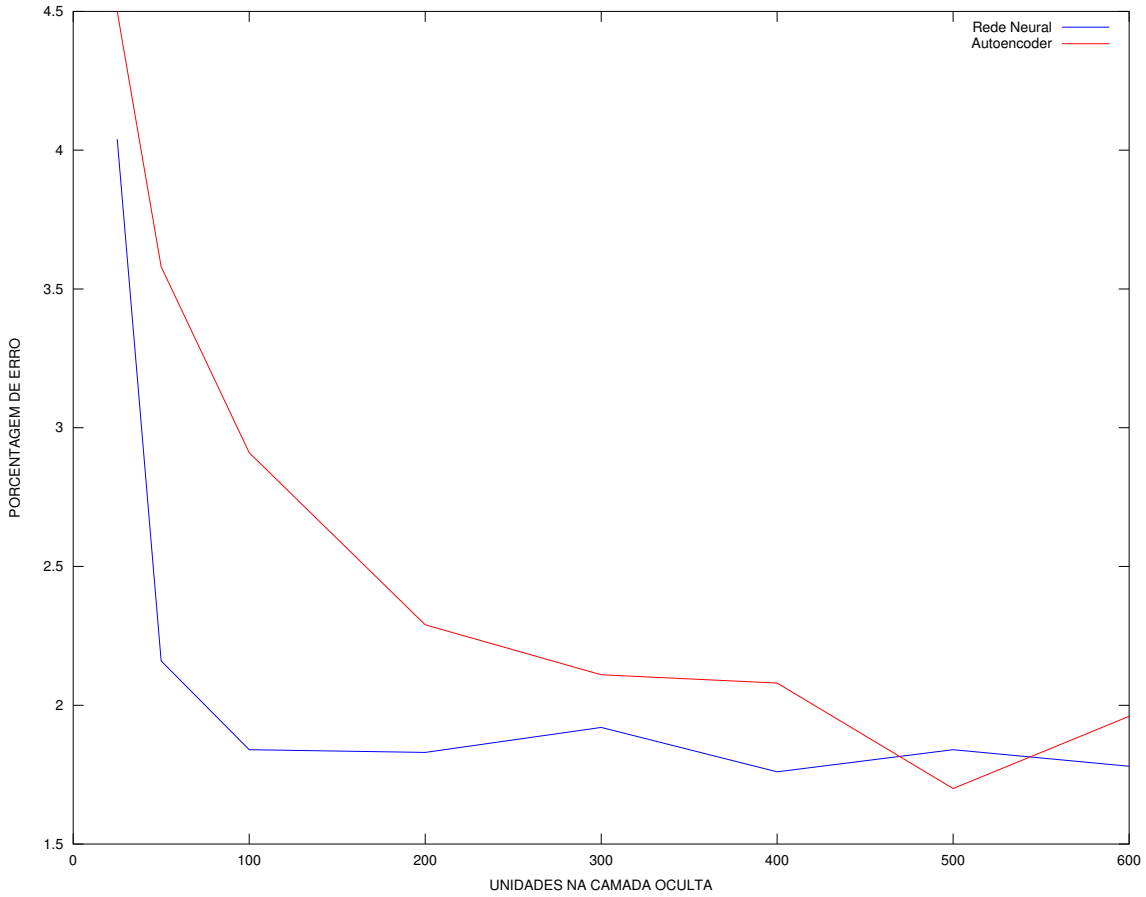

Figura 10: Gráfico de decaimento do erro de validação do autoencoder esparso e rede neural (azul) e do autoencoder (vermelho).

retam em uma melhora significativa no desempenho de uma rede neural de três camadas. Adicionalmente, estes, quando utilizados em conjunto com a técnica de adição de ruídos, apresentam melhoras nos resultados. No capítulo seguinte, finalizamos este trabalho e apresentamos nossas considerações finais acerca deste trabalho.

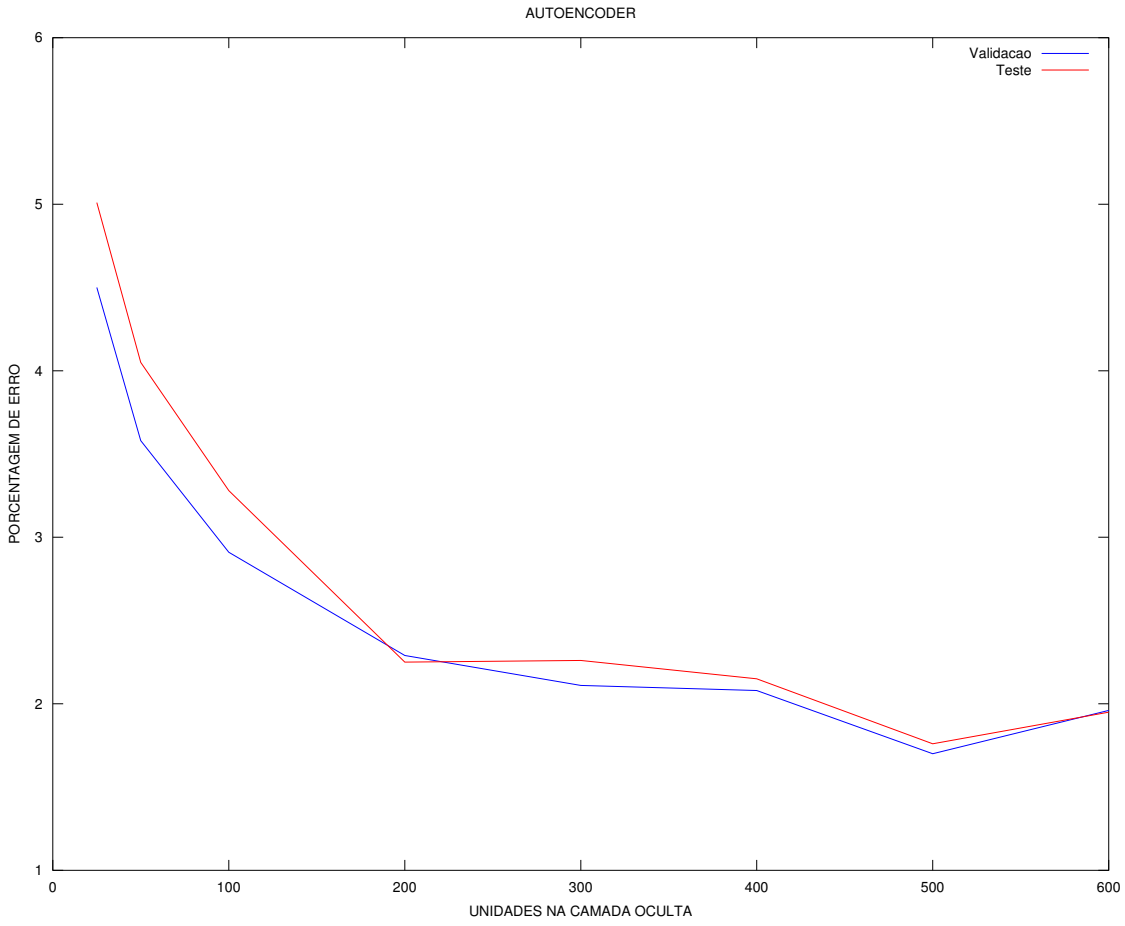

Figura 11: Gráfico de erro por número de neurônios na camada oculta para os experimentos com o autoencoder.

HL1	HL2	HL3	Erro Validação (%)	Erro Teste (%)	Tempo (min)
1000	1000	1000	1,46	1,32	1505,68
1000	500	500	1,50	1,45	840,95
1000	500	250	1,53	1,49	1310,76
500	**500**	**500**	**1,44**	**1,41**	**617,99**
500	500	250	2,21	2,53	234,17
500	500	-	1,66	1,87	162,48
500	250	-	1,83	1,89	449,73

Tabela 4: Tabela com os resultados para Autoencoders Empilhados

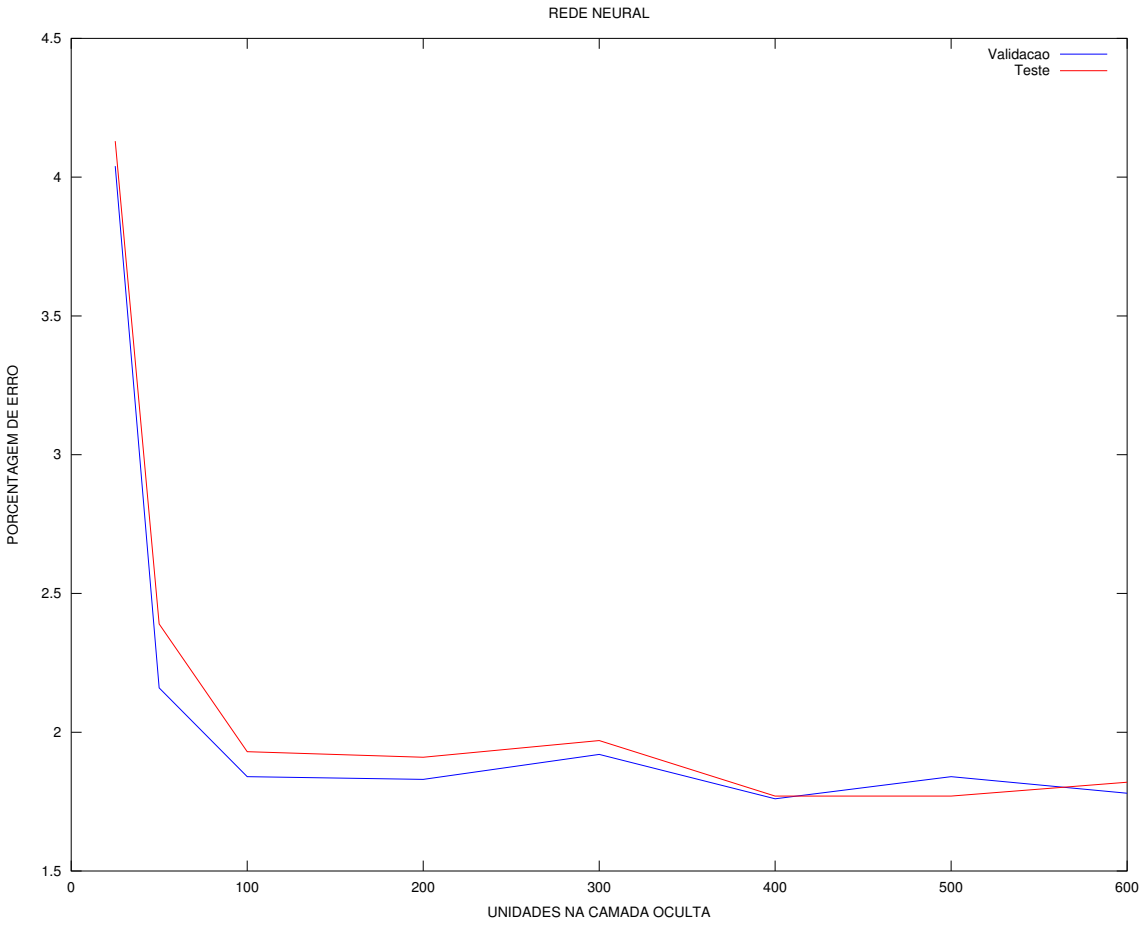

Figura 12: Gráfico de erro por número de neurônios na camada oculta para os experimentos com a rede neural.

HL1	HL2	HL3	Erro Validação (%)	Erro Teste (%)	Tempo (min)
500	500	500	1,59	1,75	689,24
500	500	-	1,72	1,74	226,87
500	-	-	1,75	1,84	1242,73

Tabela 5: Tabela com os resultados para o Autoencoder sem a utilização do parâmetro de ruído

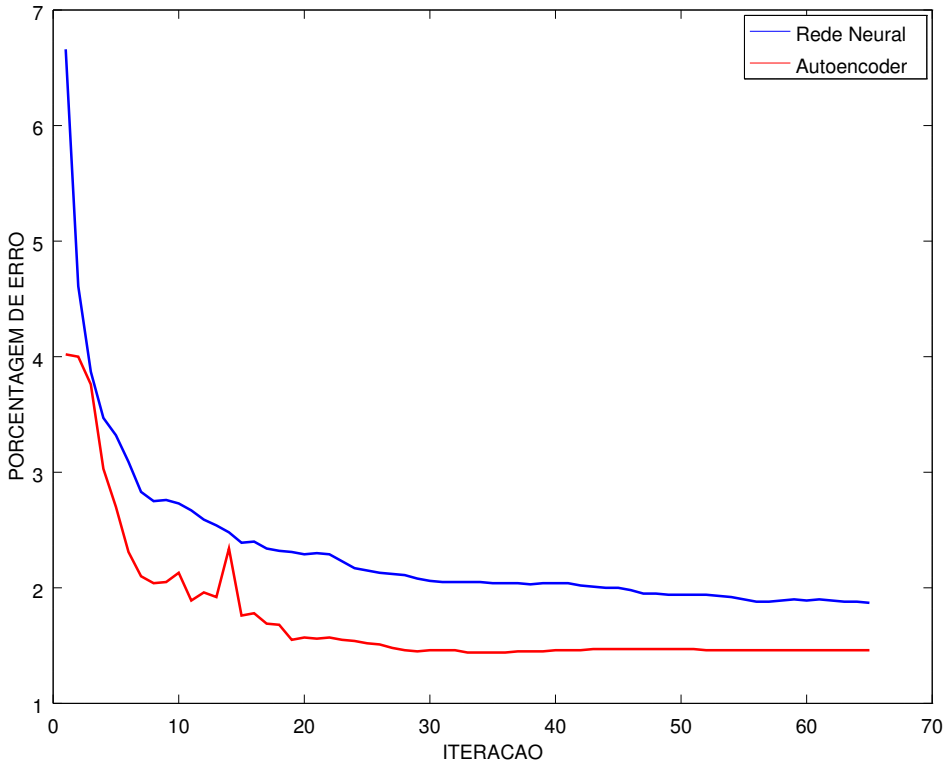

Figura 13: Gráfico de erro por iterações para os experimentos com a rede neural e o autoencoder empilhado.

5 Conclusões

Neste trabalho abordamos a técnica de aprendizado automático de características para o problema de classificação de dígitos manuscritos para o conjuntos de dados de imagens MNIST. Para o aprendizado automático, utilizamos um autoencoder esparso, uma rede neural utilizada como preprocesamento para aprender representações compactas do domínio. Para mostrar sua eficácia e variabilidade de resultados realizamos diversos testes empíricos variando seus parâmetros, como o seu empilhamento e verificação da eficácia do ruído adicionado nos dados.

5.1 Resumo dos Resultados

De acordo com os resultados, a abordagem de aprendizado automático de características contribui para a a melhoria da taxa de acerto de uma rede neural. A abordagem por autoencoders esparsos não empilhados com adição de ruído nos exemplos apresentou um erro de $1,76\%$ para o conjunto de teste em comparação à rede neural sem a extração de características que apresentou erro de $1,77\%$, ou seja, uma redução de erro modesta de $0,56\%$ que em análise dos gráficos constatamos que não é significativa em termos práticos. Entretanto, a abordagem por autoencoders esparsos empilhados apresentou erro de $1,41\%$ no conjunto de teste, ou seja, uma redução de erro considerável de $20,33\%$ em comparação ao *baseline*. No entanto, deve-se ponderar à utilização deste tipo de técnica quando se trata de aplicações em tempo real, pois houve um aumento de quatro vezes no tempo computacional com relação ao *baseline*. Por fim, constatamos que a adição de ruído nos exemplos de treino acarreta em redução significativa de erro no conjunto de teste. Nosso melhor modelo sem a utilização do ruído, apresentou um aumento da taxa de erro de 24%. Em vista dos resultados obtidos, concluímos que a abordagem por autoencoders esparsos deve ser utilizada em conjunto com a técnica de empilhamento e adição de ruídos nos exemplos para que se obtenha resultados expressivos.

5.2 Principais Contribuições

A seguir, elencamos nossas principais contribuições, dentre outras, contidas no presente trabalho.

- *Um estudo sobre Autoencoders Esparsos*

  O presente trabalho contribuiu como material didático acerca de Autoencoders Esparsos, contextualizando com trabalhos relevantes da literatura. O estudo torna-se

ainda mais importante se considerada a escassez de material didático na língua portuguesa para este assunto.

- *Análise empírica dos resultados*

  Com base nos experimentos realizados e nos resultados obtidos, o presente trabalho indica, a partir de uma análise de resultados empíricos, as vantagens do aprendizado automático de características do problema sem a utilização de especialista do domínio. Desta forma, elucidamos que preprocessar os dados brutos com um autoencoder esparso com sucessivos empilhamentos e adição de ruídos nos exemplos contribui consideravelmente para um aumento da taxa de acerto de uma rede neural.

- *Metodologia*

  Por fim, apresentamos uma metodologia empírica para validação dos modelos de autoencoders e rede neural de forma a auxiliar o leitor ao aprimoramento da utilização destes e obter resultados expressivos com a aplicação de autoencoders esparsos.

5.3 Trabalhos Futuros

Nesta seção, apresentamos algumas direções de pesquisa que consideramos relevantes. A seguir, elucidamos os seguintes pontos: função objetivo, conjuntos de dados, comparativos e paralelização do algoritmo.

Para a função objetivo, é pretendido projetar heurísticas de esparsidade da função objetivo de modo a evitar a escolha do número de neurônios da camada escondida. Desta forma, com uma boa heurística, bastaria fixar um número de neurônios que a heurística restringiria o número de neurônios que fariam parte das ativações a cada iteração.

Já se tratando de conjunto de dados, pretendemos realizar testes com conjuntos de dados relativos ao mercado financeiro; dados não estruturados como texto; e relativos à biomédica. Seria interessante, a partir da utilização da proposta, encontrar variáveis latentes das séries de preço de forma a prever a direção do mercado financeiro. Adicionalmente, como autoencoders tiveram sua principal motivação para o domínio de imagens, seria interessante investigar problemas da área da biomédica para a classificação de imagens de pacientes com determinada doença. Um exemplo seria a classificação de tumores em malignos ou benignos.

Com relação aos comparativos feitos neste trabalho, este poderiam ser refinados utilizando como baseline uma rede neural com mais de três camadas. Ou seja, um comparativo mais fiel com os autoencoders empilhados.

Por fim, consideramos que há campo para a paralelização do algoritmo de treinamento do autoencoder. Poderiam ser estudadas formas de paralelização utilizando GPUs.

Referências

BENGIO, Y. Learning deep architectures for ai. *Found. Trends Mach. Learn.*, Now Publishers Inc., Hanover, MA, USA, v. 2, n. 1, p. 1–127, jan. 2009. ISSN 1935-8237. Disponível em: <http://dx.doi.org/10.1561/2200000006>. Citado 2 vezes nas páginas 15 e 26.

BETECHUOH, B. L.; MARWALA, T.; TETTEY, T. Autoencoder networks for hiv classification. *CURRENT SCIENCE-BANGALORE-*, CURRENT SCIENCE ASSOC/INDIAN ACADEMY OF SCIENCES, v. 91, n. 11, p. 1467, 2006. Citado 2 vezes nas páginas 26 e 27.

CIRESAN, D. C. et al. Deep big simple neural nets excel on handwritten digit recognition. *arXiv preprint arXiv:1003.0358*, 2010. Citado na página 43.

DENG, L.; YU, D. Deep convex net: A scalable architecture for speech pattern classification. In: *Proceedings of the Interspeech*. [S.l.: s.n.], 2011. Citado na página 43.

HINTON, G. E.; SALAKHUTDINOV, R. R. Reducing the dimensionality of data with neural networks. *Science*, American Association for the Advancement of Science, v. 313, n. 5786, p. 504–507, 2006. Citado na página 27.

JORDAN, A. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. *Advances in neural information processing systems*, v. 14, p. 841, 2002. Citado na página 25.

LE, Q. V. et al. Building high-level features using large scale unsupervised learning. In: *ICML*. icml.cc / Omnipress, 2012. Disponível em: <http://dblp.uni-trier.de/db/conf-/icml/icml2012.htmlLeRMDCCDN12>. Citado na página 27.

LECUN, Y.; CORTES, C. *The MNIST database of handwritten digits*. 1998. Citado na página 45.

MITCHELL, T. M. *Machine Learning*. 1. ed. New York, NY, USA: McGraw-Hill, Inc., 1997. ISBN 0070428077, 9780070428072. Citado 5 vezes nas páginas 23, 24, 31, 32 e 34.

MOHRI, M.; ROSTAMIZADEH, A.; TALWALKAR, A. *Foundations of machine learning*. [S.l.]: MIT press, 2012. Citado na página 24.

NG, A. Cs229 lecture notes. 2000. Citado 2 vezes nas páginas 15 e 24.

NG, A. Sparse autoencoder. 2011. Citado 9 vezes nas páginas 15, 31, 32, 33, 34, 35, 36, 37 e 38.

PEARSON, K. *On Lines and Planes of Closest Fit to Systems of Points in Space*. University College, 1901. Disponível em: <http://books.google.com.br-/books?id=uGt_YgEACAAJ>. Citado na página 26.

SALAKHUTDINOV, R.; HINTON, G. E. Learning a nonlinear embedding by preserving class neighbourhood structure. In: *International Conference on Artificial Intelligence and Statistics*. [S.l.: s.n.], 2007. p. 412–419. Citado na página 43.

SAMUEL, A. L. Some studies in machine learning using the game of checkers. *IBM J. Res. Dev.*, IBM Corp., Riverton, NJ, USA, v. 3, n. 3, p. 210–229, jul. 1959. ISSN 0018-8646. Disponível em: <http://dx.doi.org/10.1147/rd.33.0210>. Citado na página 23.

VINCENT, P. et al. Extracting and composing robust features with denoising autoencoders. In: . [S.l.: s.n.], 2008. p. 1096–1103. Citado 2 vezes nas páginas 15 e 39.

VISHNUBHOTLA, S.; FERNANDEZ, R.; RAMABHADRAN, B. An autoencoder neural-network based low-dimensionality approach to excitation modeling for hmm-based text-to-speech. In: *ICASSP*. IEEE, 2010. p. 4614–4617. ISBN 978-1-4244-4296-6. Disponível em: <http://dblp.uni-trier.de/db/conf/icassp/icassp2010.htmlVishnubhotlaFR10>. Citado na página 27.