UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO

AGRUPAMENTO DE PRODUTOS INDUSTRIAIS UTILIZANDO *DENOISING AUTOENCODERS:* UM ESTUDO DE CASO

MONOGRAFIA

MARCELO DOS SANTOS CANAPARRO

Santa Maria, RS, Brasil

2016

AGRUPAMENTO DE PRODUTOS INDUSTRIAIS UTILIZANDO *DENOISING AUTOENCODERS:* UM ESTUDO DE CASO

Autor: Marcelo dos Santos Canaparro

Monografia apresentada ao Curso de Engenharia de Controle e Automação, Área de Concentração em Redes Neurais Artificiais, da Universidade Federal de Santa Maria (UFSM,RS), como requisito parcial para obtenção do grau de **Bacharel em Engenharia de Controle e Automação**.

Orientador: Prof. Rodrigo da Silva Guerra, Ph.D.

Santa Maria, RS, Brasil

2016

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Engenharia de Controle e Automação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Conclusão de Curso

AGRUPAMENTO DE PRODUTOS INDUSTRIAIS UTILIZANDO *DENOISING AUTOENCODERS:* UM ESTUDO DE CASO

elaborado por
Marcelo Canaparro

como requisito parcial para obtenção do grau de
Bacharel em engenharia de controle e Automação

COMISSÃO EXAMINADORA

Prof. Rodrigo da Silva Guerra, Ph.D.
(Presidente/Orientador)

Prof. Dr. Robinson Figueiredo de Camargo (UFSM)

Prof.ª Dr.ª Ana Trindade Winck (UFSM)

Santa Maria, 15 de Fevereiro de 2016.

RESUMO

Monografia para Trabalho de Conclusão de Curso
Centro de Tecnologia
Curso de Engenharia de Controle e Automação
Universidade Federal de Santa Maria

AGRUPAMENTO DE PRODUTOS INDUSTRIAIS UTILIZANDO *DENOISING AUTOENCODERS:* UM ESTUDO DE CASO

AUTOR: MARCELO DOS SANTOS CANAPARRO
ORIENTADOR: PROF. RODRIGO DA SILVA GUERRA, Ph.D.
Data e Local da apresentação: Santa Maria, 15 de Fevereiro de 2016.

O presente trabalho objetiva o desenvolvimento de um sistema eficaz para agrupamento de produtos por meio de uma arquitetura especifica de redes neurais artificiais, chamadas *denoising autoencoders*. Este problema é dividido em duas etapas: o pré-processamento dos dados de produtos e o treinamento da rede. O pré-processamento dos dados é realizado com o apoio de um software de mineração de dados, transformando-os de forma a permitir o acoplamento dos dados à rede. Esta, por sua vez, é arquitetada com base em estudos recentes na área e treinada camada a camada utilizando-se uma aprendizagem não-supervisionada. Desta forma, tendo seu erro minimizado com o uso de um algoritmo de *backpropagation* sem necessitar de dados previamente classificados. Objetiva-se que, após treinada, a rede encontre associações entre as características dos produtos e, baseadas nestas associações, conceba classificações abstratas destes produtos.

Palavras-chave: Mineração de dados. Redes neurais artificiais. *Backpropagation*. Aprendizagem não-supervisionada. *Denoising Autoencoder*.

ABSTRACT

Final Paper for Course Conclusion
Technology Center
Control and Automation Engineering Course
Federal University of Santa Maria

INDUSTRIAL PRODUCTS CLUSTERING USING DENOISING AUTOENCODERS: A CASE STUDY

AUTHOR: MARCELO DOS SANTOS CANAPARRO
ADVISOR: PROF. RODRIGO DA SILVA GUERRA, Ph.D.
Submission date and place: Santa Maria, February 15th, 2016.

The present paper has as objective the development of an effective system for product clustering using a specific artificial neural network's architecture, called denoising autoencoders. This problem is divided into two steps: the pre-processing of the product's data and the network's training process. The data pre-processing is accomplished with the support of a data mining software, transforming the data so that it allows the coupling between the data and the network. This network is constructed based on recent studies in the field and trained layer by layer using an unsupervised learning. Thus, having its error minimized with a backpropagation algorithm without the need of previously labeled data. It is the main purpose that, once trained, the network find associations between the product's characteristics and, based on this associations, conceive abstract classifications of those products.

Keywords: Data Mining. Artificial neural networks. Backpropagation. Unsupervised learning. Denoising Autoencoder.

Sumário

1 INTRODUÇÃO

A cada dia o mercado torna-se mais competitivo, fazendo com que fabricantes tenham que buscar menores preços em seus produtos e possuir um portfólio adaptável às mudanças de opinião de seu publico alvo. Segundo Shao (2015), os consumidores em geral são conscientes de suas necessidades, porem, não possuem conhecimento das configurações de produto disponíveis a um determinado fabricante. Do ponto de vista do fabricante, entretanto, seu distanciamento do cliente é grande, o que não permite que a percepção das necessidades dos consumidores sejam transparentes, acarretando na criação ou sugestão de produtos com baixa aceitação no mercado.

Estas mesmas empresas muitas vezes possuem bancos de dados com registros de suas atividades, sejam elas produtos fabricados, vendas realizadas, fichas de dados de compradores, projetos descartados, entre muitas outras informações. Estes conjuntos de dados são em sua maioria utilizados para auditoria e controle de produção, possuindo um volume muito grande para ser analisado por uma pessoa ou grupo. Contudo, a mineração de dados nos permite analisar exatamente este tipo de situação, onde a quantidade de informação é tão grande que padrões normalmente imperceptíveis emergem. Podem-se utilizar algoritmos de mineração de dados para encontrar grupos de compradores com interesses comuns e, a partir destes grupos, indicarem produtos ou variações de produtos aos mesmos. É possível distinguir características que tornam um produto líder de mercado e usa-las para definir a aceitação de um novo produto. A competitividade do comércio colabora para o surgimento de novas técnicas, as quais são cada vez mais utilizadas, principalmente em ambientes de vendas online, os chamados *E-commerces*. Nestes, por exemplo, segundo Linden (2003), a *Amazon.com* utiliza filtros colaborativos e modelos de grupos para realizar, item a item, associações de acordo com suas características. Sendo assim capaz de oferecer a cada usuário, baseado em seu histórico, produtos similares aos adquiridos anteriormente pelo consumidor e indicá-los a este cliente antes de uma nova compra. Estudos mais atuais aplicam filtros colaborativos combinados às informações contidas nos produtos para aprimorar o processo original. Desta forma, utilizando não somente as avaliações dos consumidores como é o caso de Wang (2015). O projeto proposto visa seguir estes conceitos para desenvolver um sistema capaz de agrupar produtos de acordo com os padrões encontrados em suas configurações, baseando-se em características comuns no portfólio disponível.

1.1 REVISÃO BIBLIOGRÁFICA

Nesta seção são apresentados alguns estudos que mantém proximidade ao projeto descrito neste documento.

1.1.1 *Integrating Data Mining and Rough Set for Customer Group-based Discovery of Product Configuration Rules*

Na china Shao (2005) utiliza *fuzzy clustering* e mineração de regras associativas para relacionar grupos de consumidores a grupos de configurações de bicicletas, para melhor determinar as possíveis configurações destas bicicletas em um ambiente de produção em massa. Em um segundo momento utiliza estas técnicas para gerar regras de configuração de produtos baseados em produtos vendidos anteriormente a estes grupos de clientes.

1.1.2 *Image Denoising and Inpainting with Deep Neural Networks*

Neste artigo Chen (2012) propõem um novo método de treino para *denoising autoencoders*, permitindo a este tipo de rede reconstituir imagens deteriorada. Este tipo de reconstituição não se limita apenas a recuperar pixels perdidos, mas podendo até mesmo remover textos sobrepostos à imagem. Ainda, devido à técnica utilizada, não existe a necessidade de indicar-se a região onde realizar os reparos.

1.1.3 *Reducing the Dimensionality of Data with Neural Networks*

De acordo com Salakhutdinov (2006), um conjunto de dados de grande dimensão pode ser reduzido e representado por uma versão codificada de menor dimensão. Um método utilizado amplamente é *principal componente analysis* (PCA), entretanto os autores fazem uso de uma rede neural multicamada com uma camada central de dimensão reduzida.

A rede tem seus pesos ajustados pelo gradiente do erro, mas apenas se os pesos iniciais já encontram-se perto de uma solução satisfatória. É então descrito uma técnica para inicializar tais pesos de forma a capacitar a rede ao aprendizado de códigos com dimensões menores e com melhores resultados quando comparados à PCA.

1.1.4 *Collaborative Topic Modeling for Recommending Scientific Articles*

O trabalho publicado por Blei (2011) aborda o problema encontrado por pesquisadores que buscam artigos científicos relevantes à suas pesquisas, em função da elevada quantidade de fontes. Neste estudo, faz-se uso de sites voltados a comunidade cientifica para relacionar pesquisadores e citações. Os autores desenvolveram duas técnicas: um algoritmo que combina filtros colaborativos, relacionando as preferências dos pesquisadores; e, modelagem de tópicos probabilística, levando em consideração o conteúdo dos artigos. Desta maneira são determinados tópicos para os artigos e estes são correlacionados as preferências dos pesquisadores.

1.2 OBJETIVOS

O projeto visa desenvolver um sistema capaz de investigar a relação entre produtos altamente configuráveis. Este, possibilita analisar as combinações das características que tornam cada produto único e, baseando-se nestas combinações, buscar uma codificação com maior abstração destes itens. Desta forma pode-se criar um mecanismo capaz de classificar produtos de forma mais abstrata, tornando possível agrupar produtos baseados nas similaridades encontradas.

Para desenvolver este sistema utiliza-se técnicas de mineração de dados para o condicionamento dos dados, de forma a reduzir-se o espaço de busca, solidificar as informações e adaptá-las a estrutura da rede escolhida. Ainda, para a construção do sistema de relacionamento e abstração será implementada uma rede neural artificial com capacidade de comparação de produtos, baseando-se em dados de venda e portfólio disponíveis.

Para alcançar o objetivo geral do trabalho propõem-se:

- Determinar um produto ou portfólio de produtos alvo dos estudos;
- Processar os dados dos produtos de forma a melhor adaptá-los ao projeto;
- Treinar uma rede neural capaz de instituir categorias para as quais produtos são atribuídos;
- Realizar testes e análise dos dados para garantir a qualidade do sistema.

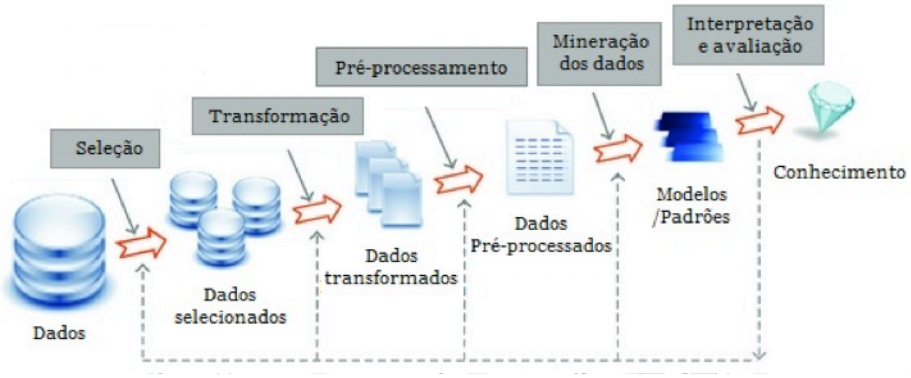
2 REFERENCIAL TEÓRICO

Nas subseções a seguir são revisados os conhecimentos necessários para a realização do trabalho. Os assuntos abordados são mineração de dados, redes neurais artificiais, *backpropagation*, métodos de treinamento de redes, *autoencoders* e *collaborative deep learning*.

2.1 MINERAÇÃO DE DADOS

A mineração de dados, segundo Witten (2011), é definida como o processo de descoberta de padrões em um conjunto de dados. Este processo deve ser automatizado ou semiautomático, e os padrões descobertos devem ser significativos o suficiente para trazer algum tipo de vantagem, geralmente econômica. Além disto os dados em questão se apresentam invariavelmente em quantidades substanciais. Na Figura 1, pode-se observar as etapas necessárias para a obtenção de conhecimento a partir de um conjunto de dados.

Figura 1: Etapas da mineração de dados



Na etapa de seleção de dados é feita a classificação dos dados a serem utilizados para estudo, realizando uma possível eliminação preliminar de dados irrelevantes ao objetivo almejado. Posteriormente, durante a transformação, estes são analisados buscando remover erros, lacunas, ruídos e incoerências. Uma vez feito isto, os mesmos passam por uma etapa de pré-processamento com o fim de adaptá-los ao algoritmo de mineração de dados selecionado para o estudo. As técnicas utilizadas nesta etapa são do tipo normalização, agregação, discretização, balanceamento, entre outras. Com os dados adaptados são então aplicados os algoritmos de mineração de dados, buscando a descoberta de regras, padrões, associações, entre outros tipos de

modelos. Estes modelos devem então, passar por uma etapa de validação na qual são testados avaliando sua eficácia e viabilidade no ambiente ao qual se destinam.

Para a analise inicial dos dados e o condicionamento necessário para este trabalho são utilizadas as três primeiras etapas do processo de mineração de dados. Nestas etapas encontramos conceitos importantes que serão esclarecidos a seguir.

Transformação

Quando lidamos com dados de operações reais nos deparamos com vários aspectos incomuns ao ambiente controlado de simulações. É muito normal que dados como, por exemplo, inseridos por uma pessoa, contenham erros de digitação, incoerências devido a distrações ou mesmo informações incompletas. Para contornar-se estes problemas pode-se examinar as informações e descartar entradas que contenham estes lapsos.

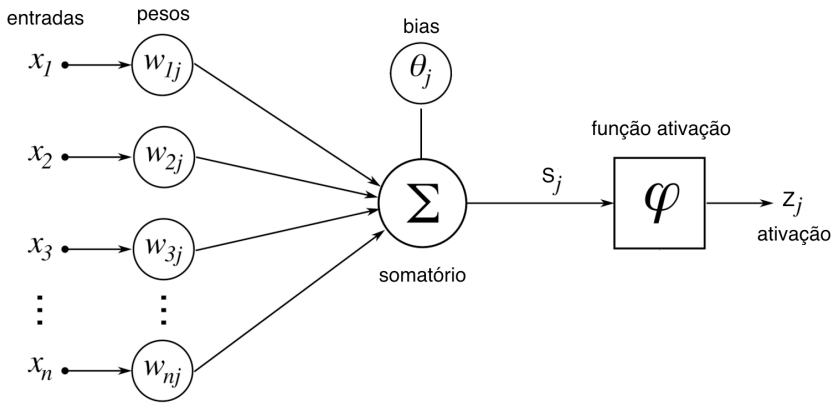
Pré-processamento

Na grande maioria dos casos os dados armazenados encontram-se em seu estado bruto e, portanto, não estão adequados aos possíveis métodos utilizados para sua análise. Pode-se encontrar incompatibilidades entre os tipos de dados aceitados pelo algoritmo de estudo e a base. Ainda, o algoritmo de análise pode não ser compatível com dados contínuos, requerendo a discretização dos mesmos. Outra situação é encontrar-se valores em um intervalo muito grande necessitando normalizar as informações. Por fim, pode-se gerar dados novos a partir das informações contidas com o intuito de compactar os dados, neste caso é permitido, por exemplo, utilizar a média ou desvio padrão de alguma característica, isto recebe o nome de agregação.

2.2 REDE NEURAL ARTIFICIAL

Redes neurais artificiais são modelos computacionais que imitam o cérebro humano, capazes de aprendizagem e reconhecimento empírico de padrões por meio de unidades de processamento simples chamados nodos ou neurônios. Estes, são tipicamente distribuídos em camadas e conectados entre si através de sinapses, como demonstra a Figura 3. Tais neurônios são modelos matemáticos com n entradas e apenas uma saída z, cada entrada possui um peso w associado que determina o quanto a mesma deve ser considerada. A estrutura de um neurônio e suas características, acima citadas, podem ser observadas na Figura 2.

Figura 2: Estrutura interna de um neurônio artificial



É importante mencionar um fator adicional ao somatório de cada nodo, o *bias* denotado por θ_j. Trata-se de um neurônio com saída constante um e que se conecta a todos os outros neurônios da rede por um peso θ. Este tem o papel de ajuste para que o somatório do neurônio não permaneça em valores que saturarem a função de ativação, agindo assim como um regulador de nível cc.

Similar a um neurônio biológico, o modelo matemático emite um sinal de ativação determinado, primeiramente, pela resultante do somatório de suas entradas, após aplicarem-se seus respectivos pesos, como mostra a equação (1).

$$s_j = \sum_{i=0}^{n} z_i w_i \qquad \textbf{(1)}$$

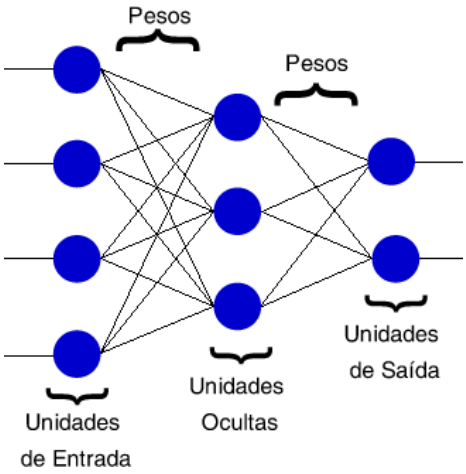
Denota-se os índices ***j*** e ***i*** como um neurônio qualquer e seus predecessores respectivamente.

A ativação neural total é dada por uma "função ativação" que permite a emissão de uma saída limitada por assíntotas horizontais. Entre algumas funções ativação podem ser citadas a linear saturada, a rampa, a tangente hiperbólica e a sigmoide também conhecida como S-Shape. Uma das funções sigmoides mais conhecidas é a logística, representada matematicamente pela equação (2).

$$\sigma(s_j) = \frac{1}{1 + e^{\frac{-s_j}{T}}} \qquad \textbf{(2)}$$

Onde ***T*** determina a suavidade da curva.

Figura 3: Rede neural perceptron



2.3 *BACKPROPAGATION*

A fim de treinar uma rede neural para desempenhar uma tarefa é necessário um conjunto de pares de entradas e saídas para que a rede possa aprender seu padrão e, em seguida, determinarmos a metodologia de treinamento. Um método amplamente aplicado é o algoritmo de *backpropagation*. Este faz uso de computações de gradientes do erro entre as saídas em função dos pesos $\boldsymbol{w}$. Para tal deve-se primeiramente definir uma equação para o erro.

Ao treinar-se uma rede neural por aprendizagem supervisionada espera-se que para um dado vetor de entradas $\boldsymbol{x}$ se obtenha o vetor de saída $\boldsymbol{y}$, contudo a saída que se obtém é $\hat{\boldsymbol{y}}$. O erro desta saída é dado pela equação (3).

$$E = \frac{\sum_n (y_n - \hat{y}_n)^2}{2} \qquad (3)$$

Com esta formulação garante-se que os erros de cada nodo de saída sejam positivos, evitando assim que dois erros distintos subtraiam-se.

A próxima definição que se deve fazer é determinar a derivada da função ativação da rede. Neste caso, será utilizado a função sigmoide descrita pela equação (4).

$$\sigma(s_j) = \frac{1}{1 + e^{-s_j}} \qquad (4)$$

Sua derivada segue como,

$$\sigma'^{(s_j)} = -(1 + e^{-s_j})^{-2}(1 + e^{-s_j})' \qquad (5)$$

$$\sigma'^{(s_j)} = \frac{e^{-s_j}}{(1 + e^{-s_j})^2} = \frac{1}{1 + e^{-s_j}} \frac{1 + e^{-s_j} - 1}{1 + e^{-s_j}} \qquad (6)$$

$$\sigma'(s_j) = \frac{1}{1+e^{-s_j}}\left(\frac{1+e^{-s_j}}{1+e^{-s_j}} - \frac{1}{1+e^{-s_j}}\right) \quad (7)$$

$$\sigma'(s_j) = \sigma(s_j)\left(1-\sigma(s_j)\right) \quad (8)$$

O próximo passo consiste em se determinar o gradiente do erro em função de cada peso da rede. Para tal será considerado as seguintes definições;

- O peso que conecta a saída do neurônio n_i para o neurônio n_j é w_{ij};
- A soma ponderada das entradas no nodo n_j é $s_j = \sum_q w_{qj} z_q$, sendo q o índice de cada nodo ligado as entradas de n_j;
- A saída do nodo n_j, após a função de ativação é dada por $z_j = \sigma(s_j)$.

Assim o gradiente do erro em função de um determinado peso pode ser definido como;

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial z_j}\frac{\partial z_j}{\partial s_j}\frac{\partial s_j}{\partial w_{kj}} \quad (9)$$

Considerando a definição de s_j, o ultimo fator resume-se a,

$$\frac{\partial s_j}{\partial w_{kj}} = z_k \quad (10)$$

Para os fatores restantes é possível definir que,

$$\frac{\partial E}{\partial z_j}\frac{\partial z_j}{\partial s_j} = -\delta_j \quad (11)$$

Assim,

$$\frac{\partial E}{\partial w_{kj}} = -\delta_j z_i \quad (12)$$

Para o caso da computação do gradiente na ultima camada tem-se que,

$$\frac{\partial z_j}{\partial s_j} = \frac{\partial \sigma(s_j)}{\partial s_j} = \sigma(s_j)\left(1-\sigma(s_j)\right) = z_j(1-z_j) \quad (13)$$

$$\frac{\partial E}{\partial z_j} = \frac{\sum_n (y_n - \hat{y}_n)^2}{2} = -(y_j - z_j) \quad (14)$$

Combinando em (12) as equações (13) e (14) obtem-se,

$$\frac{\partial E}{\partial w_{kj}} = -\delta_j z_k = -z_j(1-z_j)(y_j - z_j)z_k \quad (15)$$

Para as camadas anteriores à ultima não é possível acessar o erro diretamente e, portanto, precisa-se obter $\frac{\partial E}{\partial z_j}$ de outra maneira. Aplicando-se a regra da cadeia tem-se que,

$$\frac{\partial E}{\partial z_j} = \sum_i \frac{\partial E}{\partial z_i}\frac{\partial z_i}{\partial s_i}\frac{\partial s_i}{\partial z_j} \quad (16)$$

Caso j for a penúltima camada, então i é a ultima camada, e os primeiros dois fatores são dados por (11). Restando apenas o ultimo fator,

$$\frac{\partial s_i}{\partial z_j} = \frac{\partial \sum_q w_{qi} z_q}{\partial z_j} = w_{ji} \quad (17)$$

Logo,

$$\frac{\partial E}{\partial z_j} = \sum_i -\delta_i w_{ji} \quad (18)$$

Combinando (16) com os resultados dos fatores em (10),(13) e (18) obtendo-se a seguinte expressão,

$$\frac{\partial E}{\partial w_{kj}} = -\left(\sum_i -\delta_i w_{ji}\right) z_j(1 - z_j) z_k \quad (19)$$

Este processo é repetido para todas as camadas seguintes de forma recursiva. E, por fim, realiza-se o ajuste dos pesos

$$\boldsymbol{w}^{[k+1]} = \boldsymbol{w}^{[k]} - \eta \frac{\partial E}{\partial \boldsymbol{w}^{[k]}} \quad (20)$$

Onde η determina o passo de aprendizagem da rede e deve ser escolhido cuidadosamente. Um valor muito pequeno de η tornará o aprendizado muito lento e poderá guiar a rede até um mínimo local, enquanto que um valor muito alto fará com que a rede tenha dificuldade de aprendizagem por não conseguir acompanhar a curva do erro.

2.4 REDES PROFUNDAS

Redes neurais artificiais podem variar sua arquitetura, desde modelos simples compostos por um único neurônio ou combinando-se neurônios e ligando-os de forma a criarmos uma rede multicamadas. Quanto maior a complexidade da tarefa a qual a rede deve executar, maior deve ser sua capacidade de aprender e armazenar experiências, conceitos e padrões.

A rede neural hipotética representada na Figura 3 contém apenas duas camadas, associadas pelos seus respectivos pesos e seu treinamento seria relativamente simples utilizando

o algoritmo de *backpropagation*. É possível afirmar que este treinamento é simples, pois as conexões entre os neurônios são poucas e tratam-se apenas de duas camadas, logo, o erro apresentado em suas saídas não tem grande propagação. Entretanto, ao enfrentar um problema complexo, muitas vezes o número de variáveis de entrada é grande, implicando em um conjunto maior de nodos de entrada e, consequentemente, aumentando a quantidade de nós em cada camada da rede. Ainda, a quantidade de camadas da rede é afetada pela complexidade da tarefa, demandando a criação de uma rede neural profunda.

2.4.1 Treinamento de redes profundas

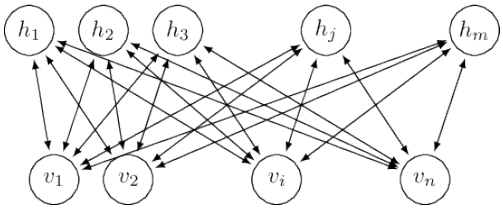
Segundo Bengio (2009), redes neurais profundas não entravam em foco de grandes discussões na literatura até o ano de 2006. A causa disto seriam erros de generalização e treinamento inferior devido aos parâmetros inicias da rede serem gerados aleatoriamente. Durante o treinamento deste tipo de arquitetura com propagação de gradientes, é comum que a rede acabe presa à mínimos locais ou grandes platôs. Entretanto, em um estudo publicado em 2006, Hinton (2006), demonstra uma estratégia rápida e efetiva para treinar este tipo de arquitetura.

2.4.2 Estratégia de treinamento

Como mencionado na seção anterior, o treinamento de redes neurais profundas através de *backpropagation* é susceptível a erros de generalização, mínimos locais e platôs. Contudo, não se limita a isto. O tempo de treinamento é exponencialmente mais longo para cada camada adicional que a rede possua, e o treinamento depende de dados previamente classificados.

Para solucionar estes problemas considere a seguinte rede neural da Figura 4.

Figura 4: Rede neural representativa de uma Restricted Boltzman Machine

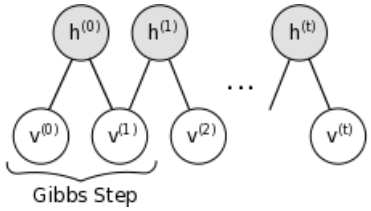


Nesta rede, chamada *Restricted Boltzman Machine* (RMB), as unidades denotadas pelo índice *v* são as unidades visíveis da rede, ou seja, são suas entradas, e aquelas denotadas pelo índice *h* são suas unidades ocultas, as quais, quando ativadas pelas entradas, devem gerar as

saídas correspondentes. É importante perceber que as unidades visíveis não se conectam entre si e que o mesmo é valido para as unidades ocultas. Observe ainda que, neste caso, para um dado conjunto de entrada na camada *v*, os cálculos de ativação de cada neurônio na camada *h* são independentes entre si e, portanto, podem ser calculados paralelamente.

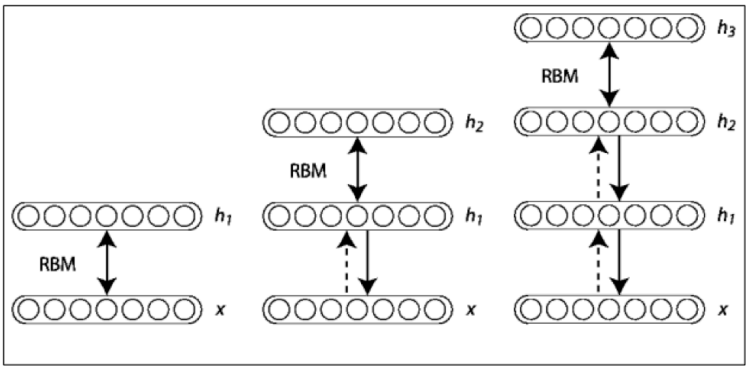
Para melhor entendimento como o treinamento desta rede é realizado, assume-se a rede apresentada na Figura 4 com pesos de ligações w. Tal rede, ao receber um dado vetor de entradas x, tem suas unidades ocultas ativadas produzindo um conjunto de saídas y de acordo com seus pesos. É possível realizar o processo inverso, chamado de modo gerador onde, a partir desta saída y, a rede reconstrua o vetor de entradas x. Contudo, devido aos pesos de suas conexões, a geração obtida é apenas uma versão corrompida de x, denotada por $\tilde{x}$. Baseando-se no quão próximo $\tilde{x}$ está de x se realiza a alteração dos pesos das conexões. Este processo pode ser repetido em cadeia como mostra a Figura 5, chamado de cadeia de Markov, e cada passo deste treinamento é chamado *Gibbs Step*.

Figura 5: Visualização do processo de treinamento



É importante salientar que este tipo de treinamento não necessita de dados previamente classificados. Esta rede, entretanto, é considerada profunda e, por tal motivo, não tem capacidade de aprender um grande número de características. Porém, ao se concluir o treinamento da mesma, pode-se utilizar suas saídas y para treinar uma nova rede a qual será utilizada como próxima camada da mesma, conforme ilustrado na Figura 6 abaixo.

Figura 6: Exemplo de rede profunda utilizando RMBs



Desta forma, é possível, ao término do treinamento de cada camada, utilizar suas saídas para treinar uma nova camada, formando assim uma rede neural profunda capaz de aprender uma grande quantidade de características.

2.5 AUTOENCODERS

Autoencoders são redes neurais que recebem um conjunto de entrada x e fazem seu mapeamento para uma representação y em suas camadas ocultas, chamadas *encoders*. Assumindo que W seja a matriz peso deste tipo de rede e b seja seu *bias*,

$$y = s(Wx + b) \quad (21)$$

Onde s representa a função sigmoide. Esta representação é, em seguida, mapeada novamente em uma reconstrução z de mesma dimensão que x, através de sua parcela decodificadora (*decoder*), da forma:

$$z = s(W'y + b') \quad (22)$$

Onde W' pode ser uma nova matriz ou ser restrita à $W' = W^T$, chama-se essa situação de pesos amarrados. Os parâmetros W, W', b, b' são otimizados de forma a minimizar o erro de reconstrução do autoencoder, o qual pode ser mensurado de várias maneiras. A equação mais comum é o erro quadrático dado por:

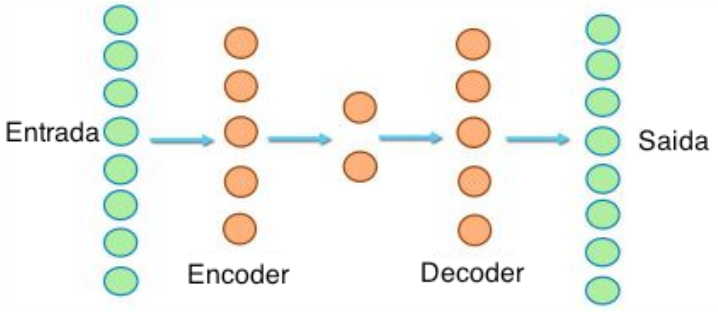
$$L(x, z) = \|x - z\|^2 \quad (23)$$

No caso de entradas com valores binários ou probabilidades binárias, a equação de entropia cruzada pode ser utilizada. Neste caso, sendo n o número de entradas tem-se,

$$L(x,z) = -\sum_{k=1}^{n}[x_k \log z_k + (1-x_k)\log(1-z_k)] \quad (24)$$

A Figura 7 ilustra a arquitetura de um *autoencoder*. Percebe-se que na parte central da rede deve existir um maior grau de abstração por parte da mesma, para que uma quantidade menor de nodos seja capaz de representar dados de maior complexidade considerando as características mais relevantes dos dados de treinamento.

Figura 7: Exemplo de *Autoencoder*



O intuito principal é que esta rede seja capaz de capturar uma representação dos dados de entrada, baseando-se nos fatores principais de variação destes dados. Entretanto, devido a representação y ser treinada para encontrar as características mais relevantes dos exemplos de treino, ela não apresenta bons resultados ao ser aplicada a um exemplo aleatório do espaço de entrada $[0,1]^n$. Porém, encontra baixos erros de reconstrução quando testada com exemplos com distribuição similar as do espaço de treino.

Pode-se ainda, aumentar a quantidade de camadas desta arquitetura alocando camadas do tipo *encoders* em seu lado esquerdo e de mesma forma inserindo camadas de decodificação em seu lado direito.

2.6 *DENOISING AUTOENCODERS*

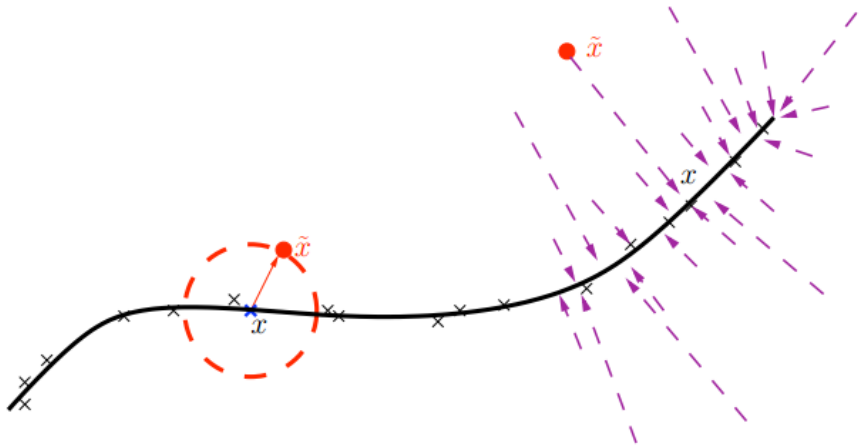
Semelhante aos *autoencoders*, este tipo de rede almeja encontrar uma representação mais consistente do que a anterior. Para tal, faz-se com que a rede tente reconstruir a entrada a partir de uma versão corrompida da mesma.

Logo, além de absorver uma entrada, comprimi-la de forma a manter apenas suas características mais importantes e reconstruí-la baseando-se nesta forma comprimida, a rede deve ser capaz de capturar as dependências entre cada entrada de maneira que possa reverter o efeito

de corrupção dos dados. Para isto, a mesma é treinada utilizando um conjunto de dados corrompidos de maneira aleatória, onde cada dado de uma entrada recebe uma probabilidade de tornar-se zero, e após a reconstrução o erro é calculado em relação à entrada não corrompida.

Considerando-se um conjunto de dados de apenas duas dimensões, durante o treinamento percebe-se que os dados seguem um padrão formando uma curva no espaço, representada na Figura 8. Nesta, as cruzes (×) representam os vários dados do conjunto de treinamento, $\tilde{x}$ que denota a versão corrompida do resultado encontra-se em uma região próxima à curva formada pelos dados de treinamento e as flechas descontínuas simbolizam uma projeção a qual a rede utiliza para reconstruir $\tilde{x}$ em x.

Figura 8: Exemplo de aproximação exercida pelo *denoising autoencoder*



2.7 *COLLABORATIVE DEEP LEARNING*

Esta técnica foi desenvolvida por Wang (2015) e serve de alicerce para o trabalho desenvolvido neste documento. Este método é baseado na união de *Collaborative Topic Regression* (CTR) com uma rede neural profunda. CTR utiliza filtros colaborativos, onde é possível gerar recomendações baseadas nos padrões de compra dos consumidores, não havendo necessidade de muitas informações sobre os clientes ou sobre os produtos.

Além disto, CTR utiliza uma abordagem de modelagem por tópicos tanto para usuários quanto para produtos. Analisando-se o portfólio, pode-se alocar cada produto como pertencente a uma ou mais categorias, o mesmo vale para os consumidores, baseando-se nos produtos adquiridos pelos mesmos. Como exemplo, pode-se citar o caso de um acervo de livros aonde criam-se categorias tais como; romances, fantasias, contos, sátiras, crônicas entre outros. Para os

compradores, as mesmas categorias são aplicadas, designando o quanto cada consumidor se identifica com determinado grupo. Uma vez obtidas ambas classificações, a satisfação de um determinado cliente com um produto nunca antes comprado por este, é dada pelo grau de relação entre os tópicos do cliente e produto.

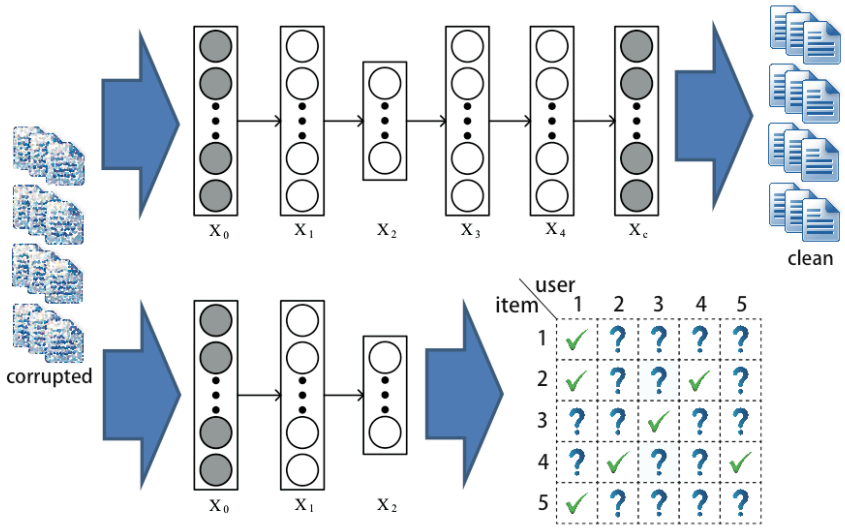
Collaborative Topic Regression é amplamente utilizado atualmente e alcança resultados razoáveis. Contudo, seu desempenho é insatisfatório quando a informação na qual se baseia é muito esparsa. No exemplo dos livros, citado acima, não é possível gerar uma recomendação para uma categoria de livros a qual um consumidor nunca comprou. *Collaborative Deep Learning* é uma possível solução para este tipo de situação.

O trabalho desenvolvido por Wang (2015) utiliza *denoising autoencoders* encadeados para formar uma rede profunda chamada SDAE (*stacked denoising autoencoder*), esta é responsável por gerar representações para os produtos baseando-se nos conteúdos dos mesmos. CDL por sua vez combina este reflexo das informações contidas nos produtos e obtido pelo SDAE com um conjunto de avaliações chamada matriz de *feedback* gerada por cada consumidor.

Como descrito no item 2.5, a interpretação y, obtida após a codificação dos dados em um *autoencoder*, é utilizada para obter-se um conceito com maior abstração de cada produto, baseando-se nas informações contidas em cada um deles. No trabalho citado, tais informações são *bag-of-words*, utilizadas em processamento de linguagem, ou seja, palavras contidas por exemplo, em sinopses de livros ou filmes, sem consideração por sua ordem ou gramatica, porém considerando a contagem de suas ocorrências. Desta maneira, a ocorrência de cada palavra é uma entrada da rede, e seu conjunto de palavras de um dado produto, um vetor de entrada.

O procedimento referenciado acima é feito para todos os produtos em um dado acervo, e suas representações codificadas pela rede são armazenadas. Pelo lado do usuário, cada produto adquirido pelo mesmo gera uma marcação na matriz de *feedback*, ou seja, produtos adquiridos recebem o valor um, caso contrário, recebem zero. A sistemática deste processo encontra-se ilustrado na Figura 9, onde pode-se entender a relação entre a representação abstrata gerada pela rede com a matriz de *feedback* construída com base nos produtos previamente adquiridos pelos consumidores.

Figura 9: Representação de *Collaborative Deep Learning*



Fonte: Wang (2015, p. 4)

Logo, para um novo produto ser recomendado a um cliente, basta comparar-se o valor obtido pela multiplicação do vetor de saída da rede (sua representação abstrata), com vetores de produtos já adquiridos pelo usuário. O resultado desta multiplicação aponta o grau de similaridade entre produtos, e em consequência, reflete o nível de aceitação que este produto terá pelo consumidor.

3 MATERIAIS E MÉTODOS

Nesta seção serão apresentadas as ferramentas empregadas para desenvolver este projeto e suas peculiaridades, assim como os métodos utilizados. Serão tópicos, portanto, o pré-processamento dos dados seguido pelo desenvolvimento da rede neural utilizada no projeto.

3.1 OBTENÇÃO E PRÉ-PROCESSAMENTO DOS DADOS

Como o intuito do projeto é criar um sistema de relacionamento de produtos, o ideal seria fazer uso de produtos reais já inseridos no mercado. Desta maneira, a empresa WEG S.A. concordou em ceder alguns dados para o estudo, com o ressalvo de que informações de seus clientes ou de seu processo de negócio não fossem abertos ao público.

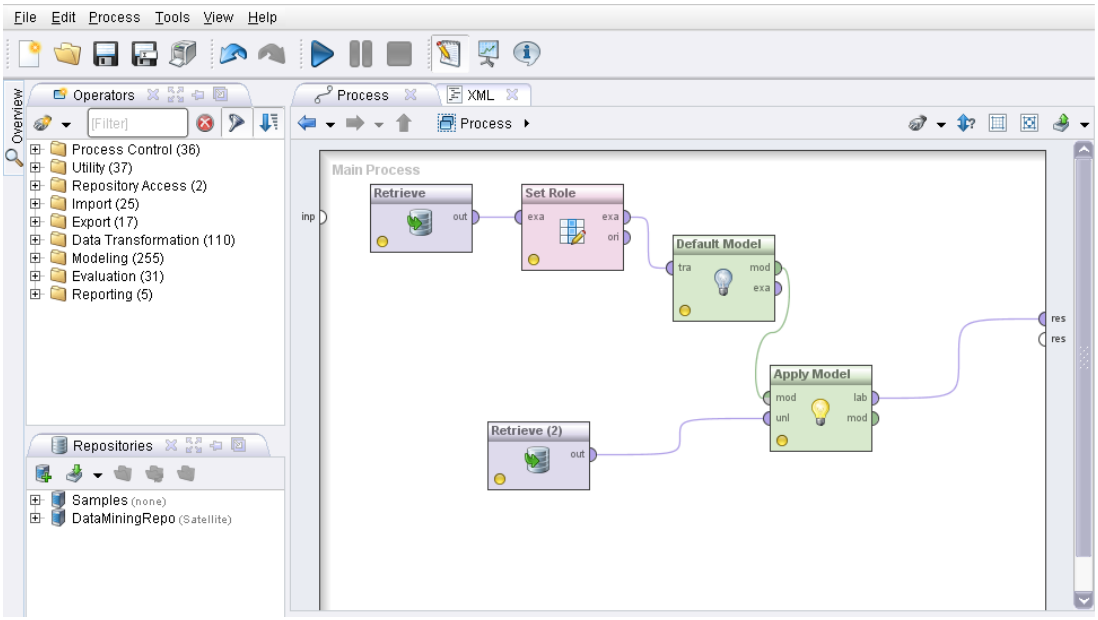
Os dados em questão foram obtidos a partir de um banco de dados Oracle, onde tabelas verticalizadas guardam a relação entre “agrupadores” - tratam-se de números identificadores de produtos - e características com seus respectivos valores, um conjunto destas caracterizam um produto. É importante salientar que os valores destas características foram mantidos com valores em forma de código, ou seja, foi utilizado apenas o número identificador dos valores e não sua descrição, com o intuito de proteger a confidencialidade dos dados. De mesma forma, obtivemos um conjunto de dados que relaciona clientes a produtos adquiridos, ou seja, agrupadores. Para manter o sigilo das informações do cliente, fez-se uso de uma chave relacional que associa as informações do cliente, como nome, endereço, CNPJ, entre outras.

Foi acordado também utilizar dados de apenas um dos segmentos da empresa. Neste caso, motores de alta tensão. Isto se deve a dois motivos, o primeiro é facilitar o desenvolvimento do projeto, já que desta forma espera-se encontrar uma maior homogeneidade dos dados, e por segundo, devido às informações neste setor serem mais recentes e, consequentemente, melhor condicionadas em virtude da experiência obtida em seus setores mais antigos. A linha de motores selecionada para o estudo foi a de motores trifásicos de indução HGF com potência nominal variando de 125 cv a 4000 cv.

3.1.1 RapidMiner

A ferramenta, conhecida como RapidMiner, foi desenvolvida pela *Artificial Inteligence Division* da *TU Dortmund University* na Alemanha. Esta divisão atualmente é uma empresa independente com fins lucrativos, possuindo clientes como eBay, Thyssenkrupp e Volkswagen. O RapidMiner baseia-se unicamente em um esquema de diagrama de blocos para construir as análises. Possui uma grande gama de operadores, principalmente para métodos de pré-processamento, validação e visualização. A Figura 10 apresenta um pequeno diagrama construído nesta ferramenta.

Figura 10: Exemplo de processo no RapidMiner

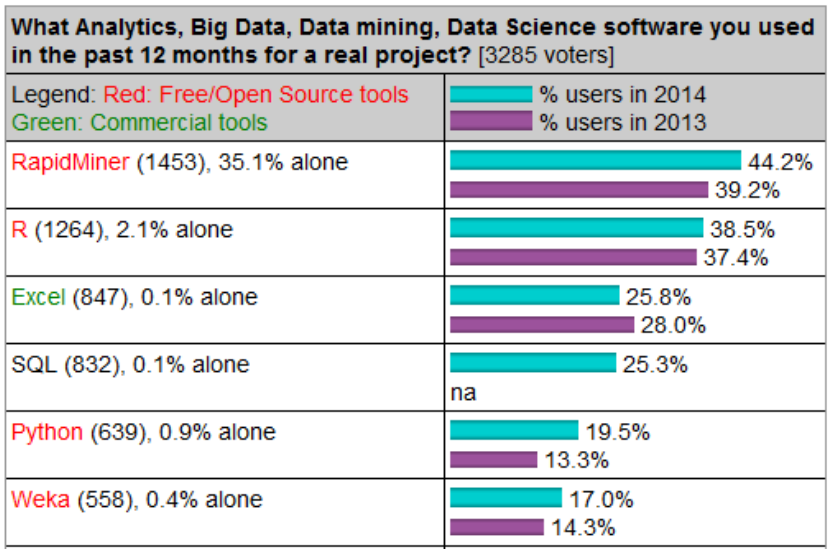


Ainda no tópico de operadores, o RapidMiner possui aproximadamente 500 permitindo grande flexibilidade de pré-processamento de dados, validação e visualização de resultados.

Usuários também reportam maior eficiência por parte do RapidMiner ao lidar com grandes quantidades de dados quando comparado aos seus concorrentes. Incluindo um menor consumo de memória devido ao fato de varias transformações nos dados ocorrerem durante a execução e não transformando e armazenando em memória. Ainda, o software é desenvolvido em linguagem Java e permite conexão direta a bancos de dados.

Na Figura 11 abaixo, segue uma seção do resultado de uma pesquisa realizada anualmente pelo site KDnuggets, o qual dedica-se unicamente a estudos de mineração de dados, oferecendo uma variedade de serviços e possuindo uma grande comunidade.

Figura 11: Resultado parcial da 15ª pesquisa anual KDnuggets



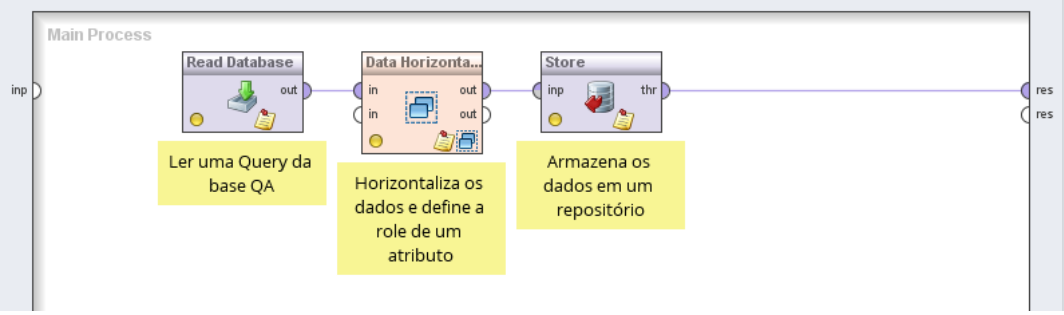
Fonte: http://www.kdnuggets.com

Em conclusão, a ferramenta escolhida para o projeto foi o RapidMiner devido a sua flexibilidade e capacidade de integração de extensões. Logo, uma licença de testes foi obtida para sua utilização.

3.1.2 Aquisição dos dados

Uma vez recebido acesso direto ao banco de dados, a aquisição dos dados foi efetuada diretamente pelo RapidMiner. Os operadores observados na Figura 12 foram utilizados para aquisição dos dados da base de QA (*Quality Assurance*) para os casos estudados neste projeto, esta base tem este nome por ser uma representação da base utilizada para produção utilizada para testes, apenas com uma defasagem de poucos meses na maioria dos casos. O operador "Read Database" realiza a conexão com o banco de dados. Ainda, para este mesmo operador, deve-se inserir uma query em linguagem SQL a qual delimita quais dados estão sendo solicitados. Esta linguagem foi estudada pelo usuário e utilizada com frequência em varias outras etapas para verificação dos dados no banco.

Figura 12: Operadores para acesso ao banco de dados



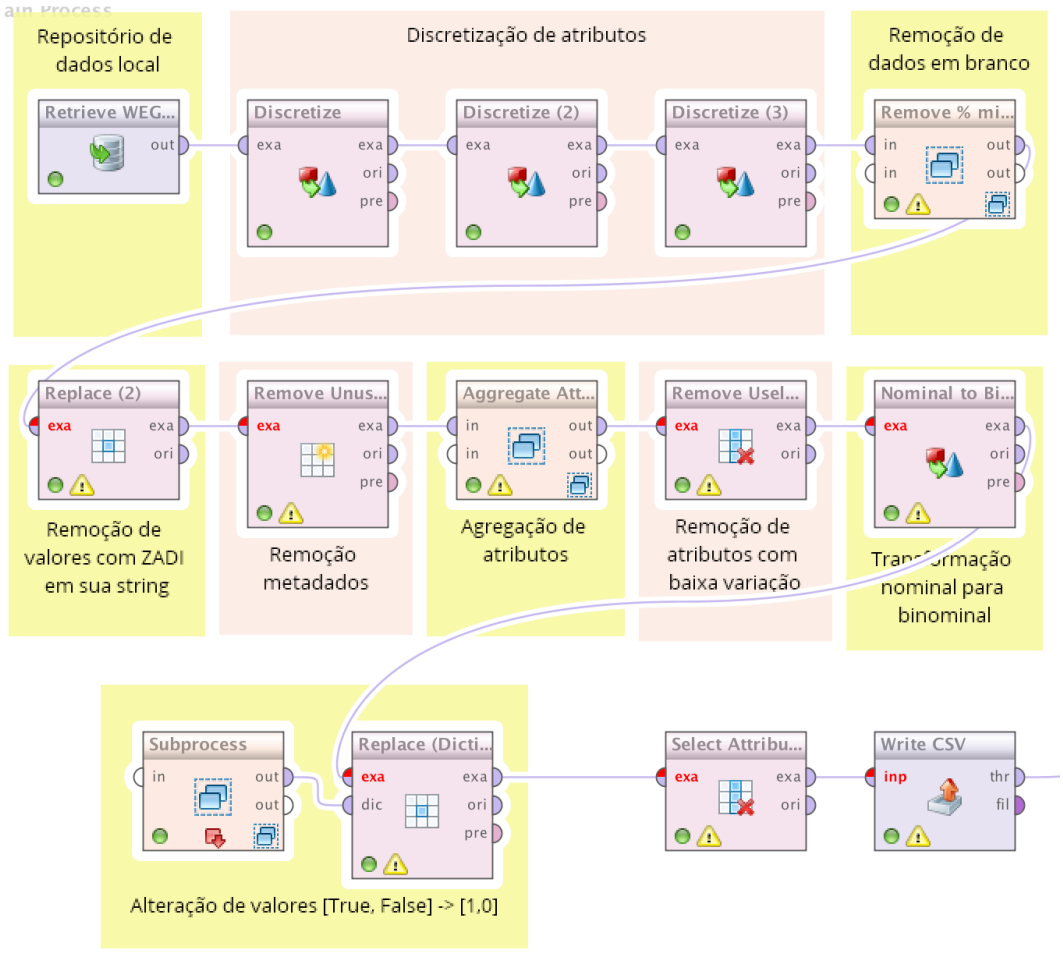
O operador central, chamado "*Data Horizontalization*" foi construído com o intuito de transformar uma tabela que contem apenas três colunas, relacionando itens baseados em um código "agrupador", em uma tabela onde cada item, onde todas as características e valores que o definem, encontram-se em uma única linha desta tabela.

3.1.3 Pré-processamento dos dados

Após esta transformação percebeu-se que os dados contém 6287 exemplos de motores, cada um destes sendo caracterizados por 1479 características. Ainda, cada característica pode assumir um conjunto variável de valores possíveis. Este elevado número de combinações causa implicações à arquitetura da rede. Exemplificando, caso utilizássemos os dados neste formato, assumindo que cada característica de um motor fosse considerada como um nó de entrada na rede, tornar-se-ia imperativa a discretização e a distribuição de cada valor uniformemente entre o intervalo [0,1]. Todavia, um maior número de valores torna a diferenciação entre estes incerta. A rede, por sua vez, poderia associar valores próximos ou mesmo confundi-los. Considere, por exemplo, a característica "Grau de proteção". Esta possui 21 valores possíveis entre os motores analisados. Desta forma, teríamos que distribuir estes valores no intervalo [0,1] para acomodar esta característica em um único nodo de entrada. Da mesma forma, a quantidade de características que configuram um motor é muito grande em relação a quantidade de configurações utilizadas em treino. Dificultando, assim, o aprendizado da rede.

Para solucionar o problema em questão, optou-se por reduzir tanto a quantidade de valores possíveis para cada característica, quanto o número de características em si. Utilizando-se, efetivamente, os conceitos discutidos na seção 2.1. Fez-se uso do processo ilustrado na Figura 13, para condicionar os dados à rede neural aplicada neste projeto.

Figura 13: Processo de condicionamento dos dados



Dentre as características, poucas apresentavam valores numéricos que não correspondessem a códigos de descrições e sim ao próprio valor. Um exemplo prático é a "altitude", característica nominal de motores elétricos que exprime a altura máxima em que a máquina pode ser submetida sem que o seu funcionamento seja comprometido. Além desta, ainda tem-se potência, tensão e frequência, discretizadas em intervalos possibilitando a redução de seus respectivos conjuntos de valores. Em todos os casos, adotou-se uma discretização baseada na frequência de cada valor, ou seja, cada intervalo final deve enquadrar uma quantidade similar de valores. Para a característica altitude, por exemplo, obteve-se apenas dois intervalos de valores: [0,1050] englobando 74% dos casos e (1050,∞) acomodando os 26% restantes. Desta forma, todos os motores da amostra se encaixariam e assumiriam um destes dois valores.

Ainda, observou-se que uma certa porção dos 1479 atributos caracterizadores não eram utilizados em todos os motores. Em alguns casos, apenas um motor entre os 6287 utilizava

determinado atributo e para todos os outros seu valor era nulo. A partir deste ponto, assumiu-se a equação abaixo para determinar a remoção de dada característica do conjunto final de características.

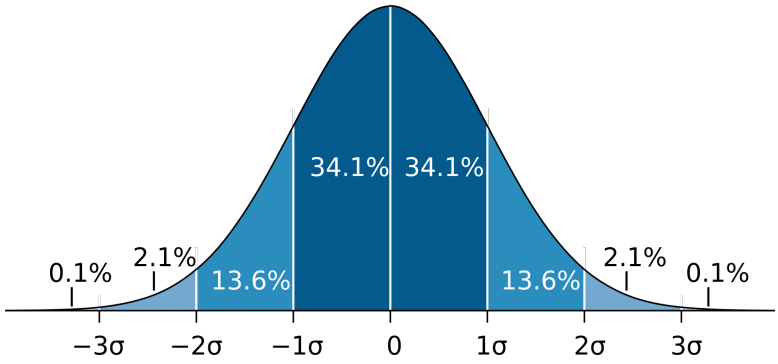
$$\frac{número\ de\ valores\ em\ branco}{número\ total\ de\ exemplos} > 0{,}1 \qquad (25)$$

A análise e remoção destas características é realizada no bloco superior direito da Figura 13.

Em outro ponto, devido às informações contidas nos dados serem provenientes de um configurador de produto utilizado pelos representantes da empresa, muitas vezes precisa-se fazer a inclusão de valores em algumas características para satisfazer o pedido de um cliente. Estes valores são registrados no produto e armazenados no banco de dados da empresa com o código ZADI, seguido por uma sequência de algarismos que os tornam únicos no sistema. Por estes valores não corresponderem a valores padrões de cada especificação eletromecânica, decidiu-se por torna-los um único valor, o qual chamamos de "none".

De mesma forma decidiu-se que características com baixa variação carregam potencialmente menos informação para distinguir cada produto. Para determinarmos o corte destas características assumimos a curva de distribuição Gaussiana apresentada na Figura 14. Nesta distribuição observa-se que com até dois desvios padrões representamos aproximadamente 95% dos eventos contidos na amostra. Logo, todas as características que assumissem o mesmo valor em 95% dos casos foram eliminadas.

Figura 14: Curva de distribuição Gaussiana



Após estas transformações foi possível reduzir drasticamente a quantidade de atributos de 1479 para 63. Entretanto, a questão de como discriminar uma quantidade excessiva de valores em

um intervalo entre [0,1] seguiu pertinente. Para sanar esta dificuldade, foi proposta uma transformação binominal, fazendo com que características deixassem de existir e apenas seus valores caracterizem um produto. Assim, a cada produto são atribuídos uma série distinta de valores, e, a cada valor qualificamos como "zero" caso o produto não faça uso deste valor e "um" caso o faça. Para manter a informação contida no nome de característica, cada valor é renomeado da forma "nome da característica = valor".

Desta maneira, soluciona-se a discriminação dos valores. Ainda assim, a quantidade total de atributos aumentou de 63 para 641. Assim, uma nova redução fez-se necessária. Para tal, definiu-se que o valor de maior ocorrência em uma característica seria considerado como principal, e os restantes seriam agregados, de modo que fossem considerados como um valor único para esta característica. Logo, houve uma redução da quantidade de atributos de 641 para 162. Esta quantidade foi considerada razoável em relação ao número de exemplos à disposição.

3.2 A REDE NEURAL

A rede desenvolvida para o projeto utiliza como base os conceitos expostos por Wang (2015), além do alicerce teórico discutido na seção 2.6. A estrutura da rede é dada por um conjunto de *denoising autoencoders*, conectados em sequência para formar uma rede neural profunda.

3.2.1 Linguagem utilizada

A rede foi desenvolvida inteiramente em linguagem Python fazendo uso de uma biblioteca popular no ambiente acadêmico chamada Theano. Os desenvolvedores desta biblioteca a definem como capaz de alcançar e mesmo ultrapassar a velocidade de códigos desenvolvidos em linguagem C, quando envolvendo grandes quantidades de dados. Theano em si não é uma linguagem de programação, entretanto possui aspectos de uma linguagem, diferente de Python que possui tipos de dados dinâmicos, o Theano exige uma forte declaração dos tipos de dados utilizados, e realiza uma pré-compilação das funções construídas pelo usuário.

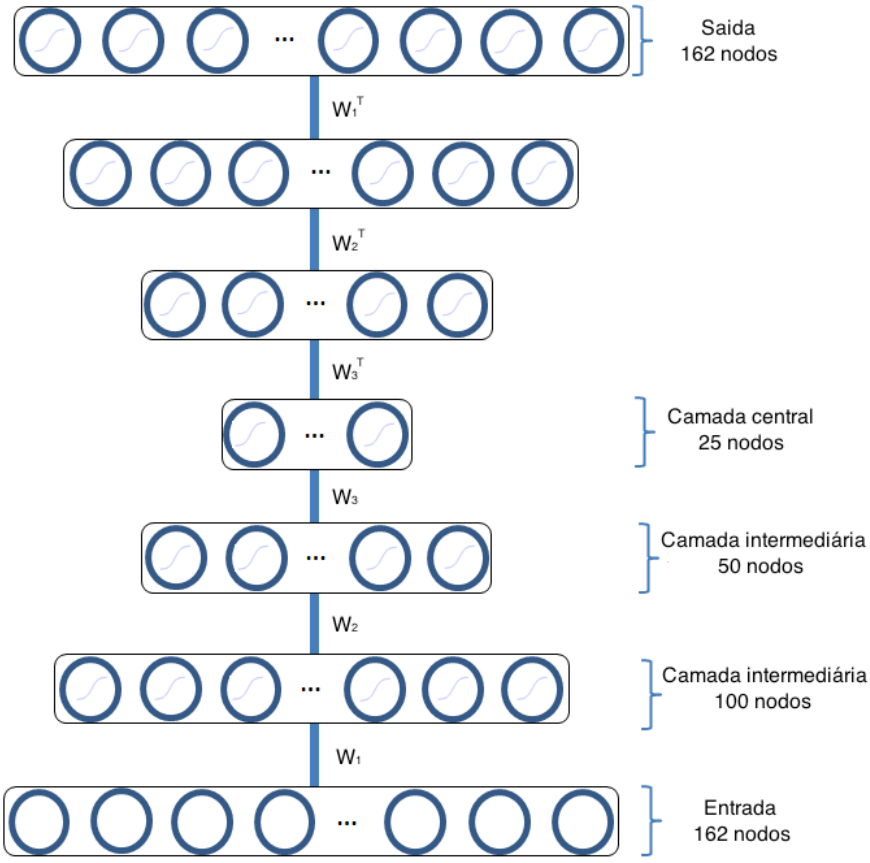
3.2.2 Estruturação da rede

Levando em consideração o conjunto de dados obtidos após a etapa de condicionamento, construímos a rede com 162 nodos de entrada de forma a acomodar as características finais em

cada nodo e, como precisamos reconstruir a mesma informação na saída para o treinamento, a mesma quantidade de neurônios são alocados na saída da rede.

Nas camadas subsequentes reduzimos gradativamente a quantidade de neurônios a fim de exigir que a rede encontre relações entre as características, e obtenha uma visão cada vez mais abstrata do produto. A terceira camada da rede, chamada de camada central é a interface entre a metade codificadora da rede e o seu lado decodificador, é esta mesma camada que após o treinamento será utilizada como saída final da rede neural. As camadas seguintes, que formam o lado decodificador, são uma reflexão das camadas anteriores. É importante salientar, que optamos por pesos amarrados para construir a rede, logo, as matrizes de pesos do codificador, denotados por W_1, W_2 e W_3 são as mesmas utilizadas no decodificador, porém transpostas, e denotadas por W_1^T, W_2^T, e W_3^T. A estrutura aqui descrita encontra-se ilustrada na Figura 15.

Figura 15: Arquitetura da rede neural utilizada



Para a inicialização das matrizes de peso foram considerados os estudos de Bengio (2010), os quais encontraram melhores resultados em redes neurais treinadas com base em

backpropagation, utilizando o que os autores chamam "*normalized initialization*". Este tipo de inicialização segue a equação abaixo.

$$W_{j,j+1} = \left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}} \right] \quad (26)$$

Onde j é a camada em questão e n representa a quantidade de nós na camada.

Na seção 2.6 mencionamos que *denoising autoencoders* recebem em suas entradas versões corrompidas das entradas originais. Portanto, usamos uma função que gera um vetor de mesma dimensão que a cada camada da rede, onde a probabilidade de corrupção de cada elemento é dada por,

$$P(1|k_i) = 1 - c \quad (27)$$

Sendo k_i, o i-ésimo elemento do vetor k, e c o nível de corrupção imposto para a camada, tal vetor é então multiplicado à entrada para corrompê-la.

Uma vez definidos valores iniciais para os pesos da rede e o nível de corrupção de cada camada, podemos efetuar o treinamento da rede. Devido aos dados pós condicionamento possuírem valores binários, adotamos a equação de entropia cruzada definida na seção 2.5 para definir o erro ou custo durante a etapa de aprendizagem da rede. A média destes erros é então efetuada por uma função da biblioteca Theano que computa os gradientes dos parâmetros da rede em relação a este erro. Os gradientes resultantes são multiplicados pelo passo de aprendizado de cada camada, inserido pelo usuário, e subtraídos dos parâmetros relevantes. O erro mencionado serve de indicador da rede, é com base nele que decidimos o sucesso ou falha do treinamento. Ainda, como o treinamento é realizado uma camada por vez, é possível observar o erro individual de cada camada.

4 RESULTADOS

Em um primeiro momento, a fim de determinarmos os parâmetros ideias da rede, aplicamos os dados obtidos após a fase de condicionamento para o treinamento. Os parâmetros neste caso são: passo de aprendizagem (η), corrupção dos dados (c) e a quantidade de neurônios em cada camada da rede neural. Alguns dos valores avaliados para estes parâmetros estão dispostos na Tabela 1.

Tabela 1: Parâmetros de treinamento

Disposição da rede	Passo de aprendizagem (η)	Corrupção (c)
200 - 100 - 200	0.1	0.2 - 0.2 - 0.2
150 - 75 - 150	0.01	0.1 - 0.2 - 0.3
75 - 35 - 75	0.001	0.3 - 0.2 - 0.1
50 - 25 -50		0.3 - 0.0 - 0.0

Com base nestes testes iniciais, observamos o erro médio para cada produto e encontramos variações entre erros de 40 a 60 por produto. Considerando que temos 162 neurônios e que cada um destes é representado por "zeros" e "uns", a rede poderia ter aprendido a representar uma quantidade muito grande de exemplos de forma equivocada. Além disto, não foi possível relacionar a combinação dos parâmetros com a variação do erro. Nesta situação, a biblioteca na qual o código foi desenvolvido não colabora para a descoberta de uma solução, isto se deve a grande parte das funções utilizadas serem previamente compiladas, reduzindo a capacidade de depuração do código.

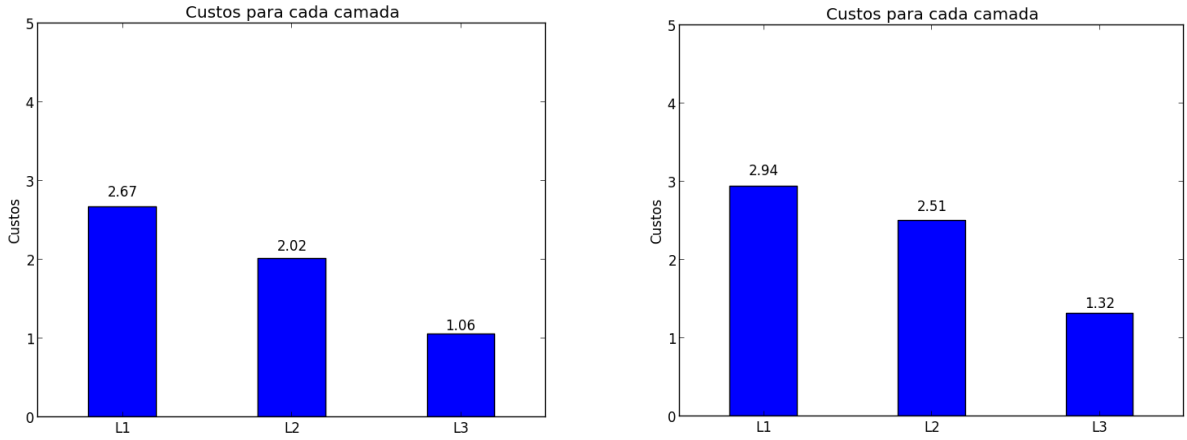
Para melhor compreender o ocorrido, treinamos novamente a rede, porém, desta vez utilizamos dados com padrões bem definidos gerados para este propósito. Estes dados continham padrões simples como "zeros" e "uns" alternados ou sequências de zeros fixas seguidas por sequências de mesmo tamanho com o dígito "um". A rede também foi reduzida tanto em número de camadas quanto de neurônios por camada, o objetivo foi perceber se a rede era capaz de absorver algum padrão e, em caso positivo, até que ponto. Durante estes testes, a rede foi gradativamente expandida conforme conseguia distinguir alguns padrões e repeti-los. Entretanto, nunca chegando ao ponto de ter consistência suficiente para lidar com os dados reais. Ainda assim, em grande parte dos treinamentos percebeu-se uma característica importante, a rede muitas vezes encontrava baixos erros para sua primeira camada devido ao uso da função de entropia cruzada como base para cálculo do erro médio de treinamento. Este tipo de função é ideal para dados binários. Mas, devido a função não linear sigmoide adotada para a ativação dos neurônios, não temos mais dados binários após a primeira camada oculta da rede, isto cria a impossibilidade de treino das camadas subsequentes.

O problema comentado acima foi corrigido substituindo-se a função de erro de entropia cruzada pelo erro médio quadrático, reduzindo a média dos erros com dados reais para um

intervalo entre 5 e 20. Além disto, tornou-se possível classificar os parâmetros adequadamente de acordo com o erro médio apresentado durante o treinamento.

Quanto ao passo de aprendizagem (η), a Figura 16 demonstra que um passo de aprendizagem menor como na figura 16(a), é efetivamente mais eficaz na redução do erro em cada camada L e, consequentemente, reduz o erro de reconstrução médio.

Figura 16: Exemplos de treinamento com passos de aprendizagem diferentes

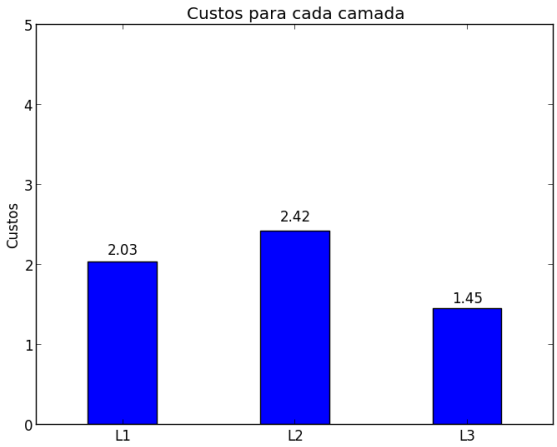


(a) **(b)**

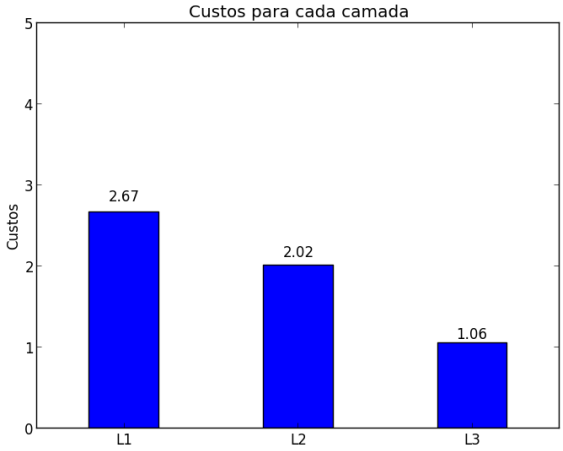
(a) η = 0.01 e erro médio = 12.88. (b) η = 0.1 e erro médio = 21.11

Comparando-se a quantidade de neurônios em cada camada, observamos que uma redução da quantidade de nós pode prejudicar a capacidade de abstração da rede, tornando-a incapaz de reconstruir amostras para as quais foi treinada, devido às limitações impostas pela etapa de codificação. Na Figura 17, mantivemos tanto o passo de aprendizagem quanto a corrupção da rede. Porém, variamos a quantidade de nós em cada camada L a estrutura de afunilamento necessária ao *autoencoder*.

Figura 17: Exemplos de treinamento com diferentes quantidade de neurônios por camada



(a)



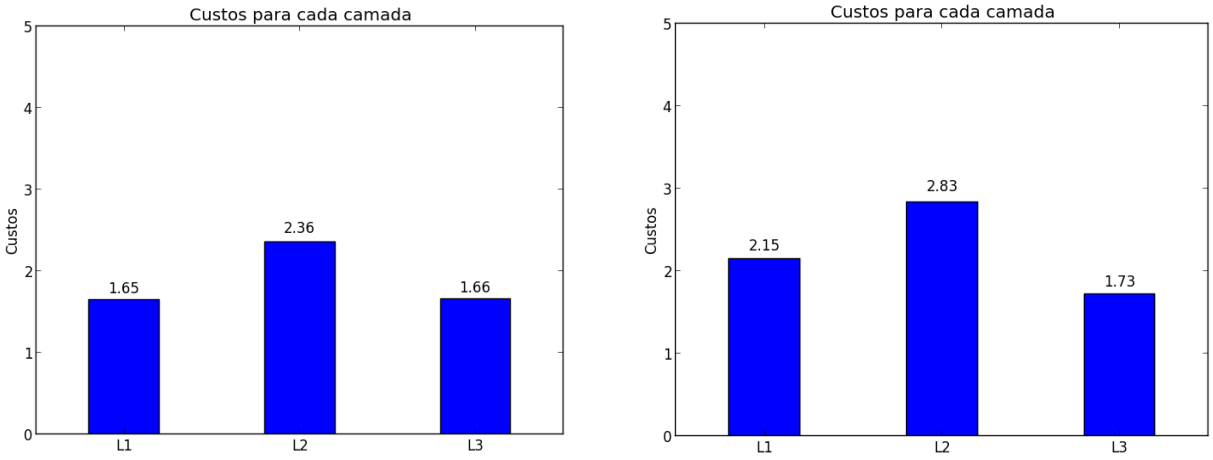
(b)

(a) $L1 - L2 - L3 = 100 - 50 - 25$, e erro médio = 15.16.
(b) $L1 - L2 - L3 = 50 - 25 - 10$, e erro médio = 21.11

Percebe-se que a figura 17(b) é a mesma apresentada na figura 16(a). Porém, quando aumentamos a quantidade de neurônios em cada camada encontramos um erro de reconstrução médio menor devido em grande parte ao a primeira camada $\boldsymbol{L1}$. O erro nesta é reduzido em 24% e, mesmo com um maior erro nas camadas subsequentes, reduz o erro médio já que esta atenuação inicial é propagada a cada nível.

O último parâmetro a ser considerado é a corrupção (c) dos dados em cada camada. Foi perceptível que quanto maior a corrupção dos dados maior seria a dificuldade da rede em reconstruir as informações não corrompidas, a Figura 18 ilustra esta constatação.

Figura 18: Exemplos de treinamento com diferentes corrupções

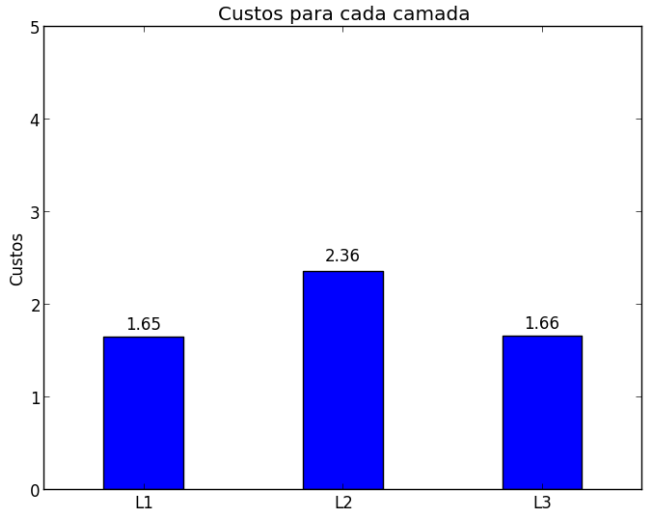


(a) **(b)**

(a) $c1 - c2 - c3 = 0.2 - 0.2 - 0.2$, e erro médio = 15.16.
(b) $c1 - c2 - c3 = 0.3 - 0.3 - 0.3$, e erro médio = 21.11

Por fim, apresentamos o resultado encontrado utilizando-se os parâmetros mais adequados conforme apresentado. Neste caso, trata-se de uma rede composta por $L1 - L2 - L3 = 100 - 50 - 25$, reduzindo o erro no primeiro nível da rede e alocando mais nodos na camada central para realizar a reconstrução da informação. Um passo de aprendizagem de 0.01 que contribui com o ajuste das conexões de forma suave e a corrupção por camada de $c1 - c2 - c3 = 0.2 - 0.2 - 0.2$. Este último parâmetro foi selecionado não apenas para reduzir o erro, mas também manter certo nível de similaridade entre um determinado item e seus semelhantes. O intuito é treinar esta rede para encontrar padrões entre as características dos produtos e, a partir destes, relacionar quais produtos são mais próximos aos já adquiridos para indicar ao cliente. Com base nestes parâmetros, obtemos o resultado da Figura 19.

Figura 19: Nível de erro por camada obtido a partir dos parâmetros finais



Com esta configuração, encontramos um erro médio de reconstrução dos exemplos de treinamento de 6.42, representando um erro de aproximadamente 4% em relação aos dados originais. Desta forma, obtivemos uma rede robusta o suficiente para encontrar padrões nos dados utilizados e reconstruí-los a partir de dados corrompidos.

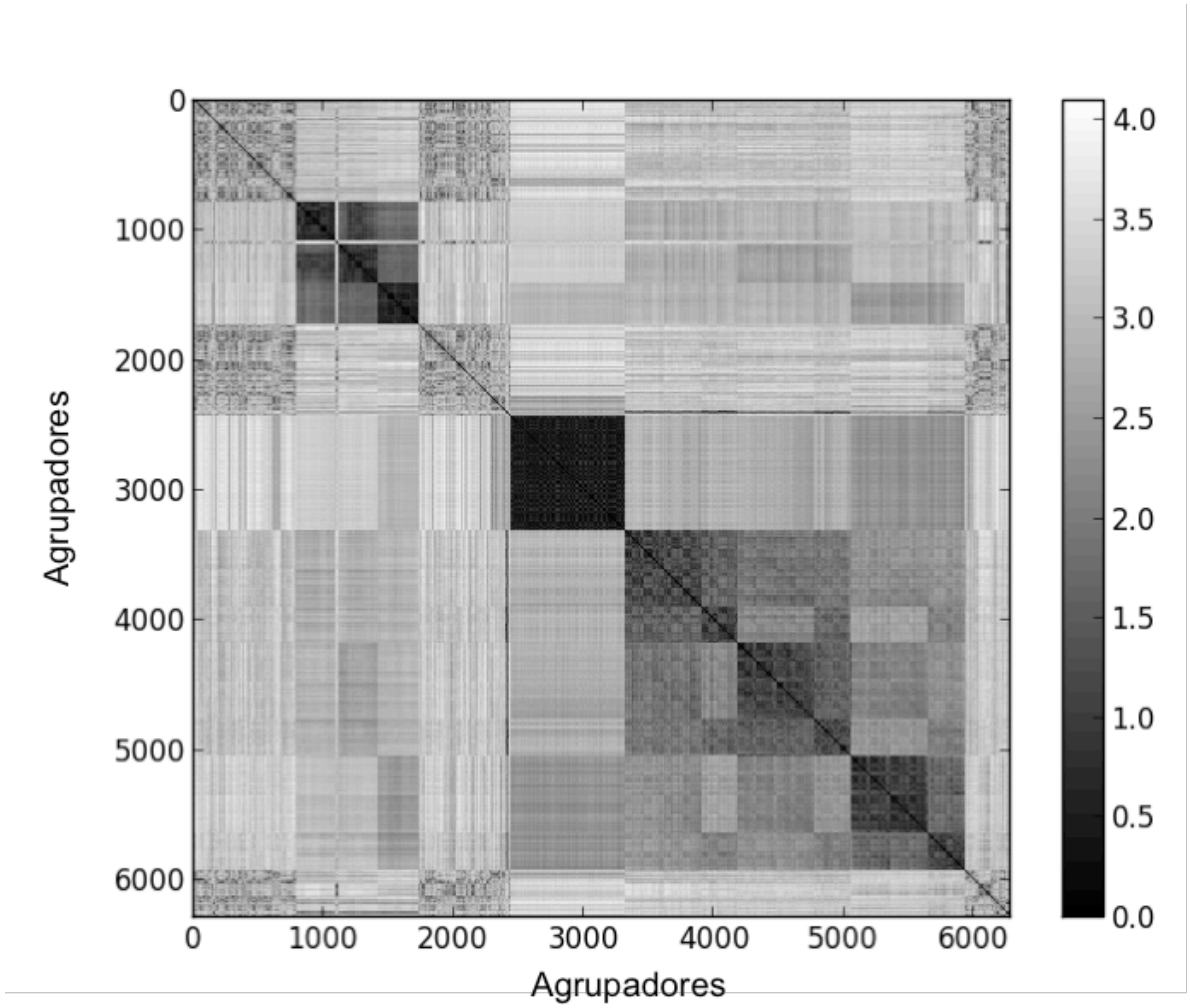
Para comprovar este fato, utilizamos a codificação realizada pela rede para comparar os agrupadores. Como forma de comparação faz-se uso da distância euclidiana dada pela equação 28.

$$D(p,q) = \sqrt{\sum_{k=1}^{n} [p_k - q_k]^2} \qquad \textbf{(28)}$$

Onde p e q são vetores com n elementos.

Desta forma obtêm-se uma matriz que correlaciona as distâncias entre cada agrupador disponível na amostra utilizada neste projeto. Tal matriz pode ser melhor visualizada por meio de uma escala em tons de cinza, como pode-se observar na Figura 20.

Figura 20: Medição da distância euclidiana entre agrupadores



Tomando-se, por exemplo, o produto de posição 3000 na figura e buscando-se o produto com maior distância euclidiana para este, encontra-se o produto de número 6286 com distância 3.95 para com o primeiro.

Neste caso, dentre as 1479 características que distinguem estes produtos, apenas 35 possuem o mesmo valor em ambos os produtos. Não obstante, dentre estas muitas se referem à detalhes que são comuns aos produtos de uma mesma família, como é o caso da amostra analisada.

O oposto também pode ser verificado ao buscarmos a menor distância encontrada com relação ao mesmo produto. Para este cenário encontra-se o produto de posição 2956 com distância igual a zero. Ambos produtos são bastante similares, contendo apenas 52 características

divergentes, sendo grande maioria delas variantes opcionais como cor de acabamento, opções de aterramento, especialidades da caixa de ligação do motor, entre outros.

5 CONSIDERAÇÕES FINAIS

O trabalho em questão teve como meta principal desenvolver um sistema capaz de analisar os conjuntos de características que compõem um portfólio de produtos e obter uma codificação abstrata diretamente relacionada a este portfólio. Este sistema é a base necessária para desenvolver-se um *software* com intuito de recomendar produtos ou agrupá-los de acordo com uma classificação abstrata. Com o fim de alcançar este objetivo, desenvolveu-se uma rede neural artificial baseada em *denoising autoencoders*, a qual obteve sucesso codificando as informações contidas em cada produto.

Como fonte de dados, foi cedido pela empresa WEG S.A. um conjunto de informações de uma família de motores elétricos pertencente a seu portfólio. Estes dados foram então condicionados através de técnicas de mineração de dados aplicadas no *software* RapidMiner, para melhor adaptá-los à rede. Posteriormente desenvolveu-se uma rede neural profunda baseada em *denoising autoencoders,* tendo seu treinamento apoiado por conceitos como *Gibbs Step* e cadeias de Markov.

Com base nos resultados durante o treinamento da rede, foi possível determinar os parâmetros que propiciam seu melhor desempenho. Estes parâmetros são, portanto, a quantidade de neurônios por camada da rede, onde 100, 50 e 25 neurônios são alocados nas camadas L1, L2 e L3 respectivamente, o passo de aprendizagem η de 0.01 e a corrupção de 20% dos dados em cada camada. A partir da combinação destas configurações obteve-se um erro médio por amostra de 6.42, que representa 4% de erro de reconstrução.

Além disto, foi possível verificar a capacidade da rede em gerar uma classificação eficaz para relacionar produtos semelhantes. Utilizou-se a distância euclidiana entre estas classificações para determinar quais produtos são afins e comparou-se tais produtos para constatar a semelhança.

Logo os resultados obtidos demonstram que a rede é eficiente em reconstruir os produtos a partir de classificações subjetivas, sendo estas criadas por ela mesma durante seu treinamento.

De mesmo modo, a rede se provou competente em criar associações entre as características dos motores elétricos, tendo em vista a corrupção aplicada aos exemplos durante o treinamento.

Propõe-se, como continuação deste trabalho, utilizar o seu resultado e, a partir das ferramentas construídas, comparar produtos por meio de sua interface codificadora. A camada central é capaz de condensar as informações de cada produto, obtendo um conceito abstrato de cada um destes, podemos então analisar cada uma destas abstrações e encontrar aquela que mais se assemelha a um dado conjunto de itens. Este conjunto base seria produtos previamente adquiridos por um consumidor que, por consequência, retrata seus desejos, necessidades e padrões de compra.

6 REFERÊNCIAS

Cohen W. W. **Fast Effective Rule Induction**. In: Twelfth International Conference on Machine Learning, 115-123, 1995.

Witten I. H., Eibe F. **Generating Accurate Rule Sets Without Global Optimization**. In: Fifteenth International Conference on Machine Learning, 144-151, 1998.

Witten I. H., Eibe F., Hall M. A. **Data mining: practical machine learning tools and techniques**. 3rd ed. Elsevier, 2011.

Shao X-Y, et al. **Integrating data mining and rough set for customer group-based discovery of product configuration rules**. International Journal of Production Research, 2005.

Linden G., Smith B., York J. **Amazon.com Recommendations: Item-to-Item Collaborative Filtering.** IEEE Internet Computing, 2003.

Wang H., Wang N., Yeung D. **Collaborative Deep Learning for Recommender Systems.** Conference on Knowledge discovery and Data Mining, 2015.

Blei D. Wang C. **Collaborative Topic Modeling for Recommending Scientific Articles**. Conference on Knowledge discovery and Data Mining, 2011.

Bengio Y. **Learning Deep Architectures for AI**. Foundations and Trends in Machine Learning, 2009.

Hinton G. E., Osindero S., Teh Y. **A fast learning algorithm for deep belief nets**. Neural Computation, vol. 18, 1527–1554, 2006.

Bengio Y., Glorot X. **Understanding the difficulty of training deep feedforward neural networks**. 13rd. International Conference on Artificial Intelligence and Statistics (AISTATS), 2010.

Hinton G. E., Salakhutdinov R. R. **Reducing the Dimensionality of Data with Neural Networks.** Science Magazine, vol. 313, 504–507, 2006.

Chen E., Xie j., Xu L. **Image Denoising and Inpainting with Deep Neural Networks.** Advances in Neural Information Processing Systems 25, 350–358, 2012.

APÊNDICE A – RELAÇÃO DE CARACTERÍSTICAS

Nesta seção é relacionado algumas características que constituem os produtos estudados neste documento. Não encontram-se aqui detalhados todos os atributos com o fim de proteger a propriedade intelectual da empresa WEG S.A. que cedeu seus dados para o estudo.

Tabela 2: Referência de características

Nome	Tipo de valor	Média	Desvio Padrão
ZALTITUDE_WM_08	Número inteiro	1626.3	1402.5
ZFREQUENCIA_1_WM_08	Número inteiro	54.9	5.7
ZPOTENCIA_POLO_1_SEM_UND_WM_08	Número inteiro	2222.5	2037.7
ZTENSAO_PARTE_1_WM_08	Número inteiro	5328.7	3375.3
Nome	**Tipo de valor**	**Quantidade de valores**	
ZCLASSE_ISOLAMENTO_01	Nominal	2	
ZFORMA_CONSTRUTIVA_WM_04	Nominal	26	
ZGRAU_PROTECAO_04	Nominal	21	
ZMANCAL_DIANTEIRO_WM_04	Nominal	66	
ZREFRIGERACAO_WM_04	Nominal	15	
ZTIPO_CX_ESTAT_WM_30	Nominal	15	
ZFATOR_SERVICO_1_WM_08	Nominal	7	
ZPOLARIDADE_COMPLETA_WM_08	Nominal	13	
ZCLASS_TERMOSENS_A_MANCA_WM_04	Nominal	11	
ZTIPO_PONTA_EIXO_WM_04	Nominal	5	
ZENTIDADE_NORMA_COMPLEM_WM_04	Nominal	13	
ZIDIOMA_MANUAL_WM_04	Nominal	18	
ZLINHA_PRODUTO_ENERGIA_01	Nominal	7	
ZMAT_CHAVETA_1A_PONTA_WM_04	Nominal	10	

Nome	Tipo de valor	Quantidade de valores
ZTENSAO_RESIST_AQUEC_WM_04	Nominal	17
ZTRAMENTO_PINTURA_WM_04	Nominal	6
ZBITOLA_FR1_CX_ACESS_A_WM_04	Nominal	75
ZACESS_LIGAD_CX_ACESS_A_WM_04	Nominal	23
ZBIT_CONEC_CX_LIG_ACES_A_WM_04	Nominal	19
ZBIT_FURO_ROS_CX_ESTAT_1_WM_04	Nominal	115
ZMATERIAL_EIXO_WM_04	Nominal	7
ZCARCACA_SEM_COMPLEMENTO_WM_08	Nominal	28
ZENTIDADE_NORMA_01	Nominal	5
ZTIPO_ACIONAMENTO_01	Nominal	4
ZCLASS_TERMOSENSO_A_FASE_WM_04	Nominal	8
ZCOR_ACABAMENTO_WM_04	Nominal	118
ZREF_POSICAO_CX_ACESS_A_WM_04	Nominal	2
ZTAMPAO_ROSC_CX_ACESS_A_WM_04	Nominal	9
ZCARCACA_EQUIVALENTE_WM_08	Nominal	15
ZTIPO_EMBALAGEM_WM_04	Nominal	6
ZMERCADO_01	Nominal	8
ZSEGMENTO_MERCADO_WM_04	Nominal	10
ZCOR_CAB_TERMS_A_MANCAL_WM_04	Nominal	14
ZMATERIAL_CX_ESTATOR_WM_04	Nominal	5
ZDETETOR_VAZAM_AGUA_WM_04	Binária	2
ZMONTAGEM_MAQUINA_WM_04	Binária	2
ZCONDICAO_TRANSPORTE_SH_04	Binária	2
ZISOLAMENTO_MANCAL_TRAS_WM_04	Binária	2
ZREVERSAO_FORMA_CONSTRUT_WM_04	Binária	2
ZPRODUTO_DESMONTADO_WM_04	Binária	2