



UNIVERSIDADE ESTADUAL PAULISTA

"JÚLIO DE MESQUITA FILHO"

Câmpus São José do Rio Preto

Gustavo Henrique de Rosa

Otimização Meta-Heurística para Regularização de Modelos de Aprendizado em Profundidade

São José do Rio Preto

2018

Gustavo Henrique de Rosa

Otimização Meta-Heurística para Regularização de Modelos de Aprendizado em Profundidade

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Câmpus de São José do Rio Preto.

Financiadora: FAPESP - Proc. 2015/25739-4

Orientador: Prof. Dr. João Paulo Papa

São José do Rio Preto

2018

de Rosa, Gustavo Henrique.

Otimização Meta-Heurística para Regularização de Modelos de Aprendizado em Profundidade / Gustavo Henrique de Rosa. – São José do Rio Preto, 2018
90 f. : il., tabs.

Orientador: Prof. Dr. João Paulo Papa

Dissertação (mestrado) - Universidade Estadual Paulista "Júlio de Mesquita Filho", Instituto de Biociências, Letras e Ciências Exatas

1. Aprendizado de Máquina. 2. Aprendizado em Profundidade. 3. Técnicas de Regularização. 4. *Dropout* & *Dropconnect*. 5. Otimização. 6. Otimização Meta-Heurística. I. Papa, João Paulo. II. Universidade Estadual Paulista "Júlio de Mesquita Filho", Instituto de Biociências, Letras e Ciências Exatas. III. Título.

CDU – 518.72:76

Gustavo Henrique de Rosa

Otimização Meta-Heurística para Regularização de Modelos de Aprendizado em Profundidade

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Câmpus de São José do Rio Preto.

Financiadora: FAPESP - Proc. 2015/25739-4

Comissão Examinadora

Prof. Dr. João Paulo Papa

UNESP - Câmpus de Bauru

Orientador

Prof. Dr. André Carlos Ponce de Leon Ferreira de Carvalho

ICMC/USP - Universidade de São Paulo

Prof. Dr. Aparecido Nilceu Marana

UNESP - Câmpus de Bauru

Bauru

28 de agosto de 2018

*Em tempos de caos, que a educação seja reflexo de uma nova união,
humanitária e científica.*

Agradecimentos

Em primeiro lugar, gostaria de agradecer à fé, que independente da religião, doutrina, ensinamentos, deve sempre existir e prevalecer, principalmente a fé no amor e na vida, os quais são sem dúvidas os grandes motivos de estarmos presente nessa passagem.

Agradeço aos meus pais, Ana Valéria e Fernando, por sempre estarem presentes em minha vida, me auxiliando e fortalecendo nos momentos que mais precisei, por sempre apoiarem as minhas decisões e me incentivarem em tudo que faço.

Agradeço à minha família, à minha prima Bárbara, aos meus tios Fabrício e Helena, Luciene e Marcos, Helen, Daiane e João, por todas as nossas conversas, ensinamentos, tudo que eles puderam proporcionar e contribuir à minha formação. Agradeço em especial aos meus avós, José e Thereza, Lurdinha e principalmente ao meu falecido vô Mauro, pessoa mais nobre e batalhadora que já conheci e que infelizmente só pude entender tardiamente o quão grande e evoluído era.

Agradeço à minha pequena, Bruna, a qual eu sempre amarei mais que tudo e que toda noite agradeço por ter aparecido em minha vida. Sem dúvidas todo o tempo que passamos juntos e que iremos passar me faz ser uma pessoa melhor e me motivar mais para, juntos, melhorarmos o mundo e crescermos. Toda a sua paciência e amor que tem comigo é algo inacreditável, o qual espero poder retribuir todos os dias da mesma forma.

Agradeço ao meu orientador, João Paulo Papa, o qual sempre foi uma fonte de inspiração para mim, seguir os seus passos é somente umas das coisas que mais almejo, podendo ter certeza de que ele é e sempre será um dos meus maiores orgulhos.

Agradeço à Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) pelo concebimento de uma bolsa de Mestrado (Processo número 2015/25739-4), a qual possibilitou o desenvolvimento deste trabalho.

Agradeço à todos os meus amigos de Brasília, principalmente os de infância e que estiverem sempre comigo, independente da distância. Ao grupo Brothers e Lindões, que hoje em dia, nos reúne e acaba por fortalecer os nossos laços. Sempre que nos reencontramos acaba sendo o mesmo amor e amizade de sempre.

Agradeço à UNESP Bauru, a única universidade a qual gostaria de ter participado e a qual é o meu maior orgulho de poder fazer parte e ser considerado um unespiano. Agradeço também à todos os professores do DCo e da FC, em especial ao Humberto, Nilceu e Kelton, os quais contribuíram enormemente para a minha formação acadêmica e social.

*"You can fool some people sometimes,
But you can't fool all the people all the time."
(Get Up, Stand Up, Bob Marley)*

Resumo

Arquiteturas de aprendizado em profundidade têm sido amplamente estudadas nos últimos anos, principalmente pelo seu alto poder discriminativo em muitos problemas considerados essenciais na área de visão computacional. Entretanto, um problema destes modelos diz respeito ao grande número de parâmetros a serem ajustados, que podem chegar a milhares. Um outro ponto crítico está relacionado à necessidade de grandes bases de dados para treinar essas técnicas de aprendizado em profundidade, bem como a sua alta propensão ao chamado super-ajuste dos dados. Recentemente, a simplista ideia de desconectar neurônios ou conexões de uma rede, técnicas denominadas de *Dropout* e *Dropconnect*, respectivamente, tem se demonstrado muito eficazes e primordiais ao processo de aprendizado, embora ainda necessitem de uma escolha adequada de parâmetros. O presente projeto pretende identificar possíveis soluções para o problema mencionado por meio de técnicas de otimização meta-heurística, objetivando encontrar o número adequado do limiar de desligamento dos neurônios e conexões. Diferentes abordagens de aprendizado em profundidade, tais como, Máquinas de Boltzmann Restritas, Máquinas de Boltzmann em Profundidade, Redes de Crença em Profundidade, Redes Neurais Convolucionais; e diferentes meta-heurísticas, tais como, Algoritmo do Morcego, Algoritmo do Vagalume, Busca do Cuckoo, Otimização por Enxame de Partículas, foram utilizadas a fim de tentar solucionar este problema. Os resultados apresentados indicam uma possível tendência em utilizar a otimização meta-heurística para encontrar parâmetros mais adequados, os quais, conseqüentemente, auxiliam no processo de aprendizado e melhoria da arquitetura neural.

Palavras-chave: Aprendizado de máquina. Aprendizado em profundidade. Técnicas de regularização. Otimização meta-heurística.

Abstract

Deep learning architectures have been extensively studied in the last years, mainly due to their discriminative power in many crucial problems in computer vision. However, one problem related to these models concerns with their number of parameters, which can easily reach thousands of hundreds. Another drawback is related to the need for large datasets for training purposes, as well as their high probability of overfitting, mainly because of their complex architecture. Recently, a naïve idea of disconnecting neurones or connections from a network, known as Dropout or Dropconnect, respectively, has shown to be a promising solution to this problem. Nevertheless, it still requires an adequate parameter setting. This project aims to identify possible solutions to the depicted problem by means of meta-heuristic optimization, trying to find the most suitable drop rate. Several machine learning approaches, such as, Restricted Boltzmann Machines, Deep Boltzmann Machines, Deep Belief Networks, Convolutional Neural Networks and several meta-heuristic techniques, such as, Particle Swarm Optimization, Bat Algorithm, Firefly Algorithm, Cuckoo Search, were employed in the context. The presented results show a possible trend in using meta-heuristic optimization to find suitable parameters in a wide range of applications, helping the learning process and improving the network's architecture.

Keywords: Machine learning. Deep learning. Regularization methods. Meta-heuristic optimization.

Lista de ilustrações

Figura 1 – Exemplo do procedimento de aprendizado do ser humano através de sua visão.	16
Figura 2 – Exemplo de (a) treinamento correto e (b) super-treinamento dos dados. . .	17
Figura 3 – Exemplo de (a) arquitetura normal de rede, (b) arquitetura com <i>Dropout</i> e (c) arquitetura com <i>Dropconnect</i>	19
Figura 4 – Diagrama de Venn ilustrando as sub-áreas referentes às técnicas de inteligência artificial.	22
Figura 5 – Arquitetura de uma RBM.	24
Figura 6 – Amostragem de Gibbs. As setas “vermelhas” denotam uma única iteração. . .	26
Figura 7 – Ilustração do método baseado em divergência contrastiva.	27
Figura 8 – Arquitetura padrão de uma DBM.	30
Figura 9 – Arquitetura padrão de uma DBN.	32
Figura 10 – Arquitetura típica de uma Rede Neural Convolucional.	34
Figura 11 – Camadas de uma pseudo-arquitetura de uma CNN.	35
Figura 12 – Exemplo de (a) sub-treinamento, (b) treinamento adequado e (c) super-treinamento.	37
Figura 13 – Exemplo da função $f(x, y) = \text{sen}(x) + \text{cos}(y)$ com diversos mínimos e máximos locais.	41
Figura 14 – Fluxograma da metodologia utilizada.	49
Figura 15 – Exemplos de imagens de treinamento da base de dados CIFAR-10.	50
Figura 16 – Exemplos de imagens de treinamento da base de dados MNIST.	51
Figura 17 – Exemplos de imagens de treinamento da base de dados Semeion.	52
Figura 18 – Exemplos de imagens de treinamento da base de dados USPS.	52
Figura 19 – Exemplos de imagens de treinamento da base de dados Brain-PET.	53
Figura 20 – Exemplos de imagens de treinamento da base de dados Brain-MRI.	54
Figura 21 – Arquiteturas de CNNs para as bases de dados: (a) MNIST, Semeion e USPS, e (b) CIFAR-10, Brain-PET e Brain-MRI.	58
Figura 22 – Gráficos: <i>número de iterações x erro de treinamento</i> para a Tabela 10. . .	61
Figura 23 – Gráficos: <i>número de iterações x erro de treinamento</i> para a Tabela 11. . .	62
Figura 24 – Gráficos: <i>número de iterações x erro de treinamento</i> para a Tabela 12. . .	63
Figura 25 – Gráficos: <i>número de iterações x erro de treinamento</i> para a Tabela 13. . .	65
Figura 26 – Gráficos: <i>número de iterações x erro de treinamento</i> para a Tabela 14. . .	66
Figura 27 – Gráficos: <i>número de iterações x erro de treinamento</i> para a Tabela 15. . .	68
Figura 28 – Gráficos: <i>número de iterações x erro de treinamento</i> para a Tabela 16. . .	68
Figura 29 – Gráficos: <i>número de iterações x erro de treinamento</i> para a Tabela 17. . .	70
Figura 30 – Gráficos: <i>número de iterações x erro de treinamento</i> para a Tabela 18. . .	70

Figura 31 – Gráficos: <i>número de iterações x erro de treinamento</i> para a Tabela 19. . .	72
Figura 32 – Gráficos: <i>número de iterações x erro de treinamento</i> para a Tabela 20. . .	73
Figura 33 – Gráficos: <i>número de iterações x erro de treinamento</i> para a Tabela 21. . .	75
Figura 34 – Gráficos: <i>número de iterações x erro de treinamento</i> para a Tabela 22. . .	75
Figura 35 – Gráficos: <i>número de iterações x erro de treinamento</i> para a Tabela 23. . .	77
Figura 36 – Gráficos: <i>número de iterações x erro de treinamento</i> para a Tabela 24. . .	77

Lista de tabelas

Tabela 1 – Utilização de meta-heurísticas e suas siglas.	42
Tabela 2 – Resumo de informações sobre as bases de dados.	50
Tabela 3 – Configuração dos parâmetros das técnicas meta-heurísticas.	54
Tabela 4 – Configuração das bases de dados para o Experimento A.	56
Tabela 5 – Configuração das arquiteturas para o Experimento B.	56
Tabela 6 – Configuração das bases de dados para o Experimento B.	56
Tabela 7 – Configuração das arquiteturas para o Experimento C.	57
Tabela 8 – Configuração das bases de dados para o Experimento C.	57
Tabela 9 – Configuração dos parâmetros da CNN.	59
Tabela 10 – Erro médio de teste (erro médio de treinamento) $\pm$ desvio padrão do erro de teste [parâmetro médio p], considerando RBMs sem regularização e otimizadas com <i>Dropout</i> e <i>Dropconnect</i> na base de dados MNIST.	61
Tabela 11 – Erro médio de teste (erro médio de treinamento) $\pm$ desvio padrão do erro de teste [parâmetro médio p], considerando RBMs sem regularização e otimizadas com <i>Dropout</i> e <i>Dropconnect</i> na base de dados Semeion.	62
Tabela 12 – Erro médio de teste (erro médio de treinamento) $\pm$ desvio padrão do erro de teste [parâmetro médio p], considerando RBMs sem regularização e otimizadas com <i>Dropout</i> e <i>Dropconnect</i> na base de dados USPS.	63
Tabela 13 – Erro médio de teste (erro médio de treinamento) $\pm$ desvio padrão do erro de teste [parâmetro p], considerando DBMs sem regularização e otimizadas com <i>Dropout</i> na base de dados MNIST.	64
Tabela 14 – Erro médio de teste (erro médio de treinamento) $\pm$ desvio padrão do erro de teste [parâmetro p], considerando DBMs sem regularização e otimizadas com <i>Dropconnect</i> na base de dados MNIST.	65
Tabela 15 – Erro médio de teste (erro médio de treinamento) $\pm$ desvio padrão do erro de teste [parâmetro p], considerando DBMs sem regularização e otimizadas com <i>Dropout</i> na base de dados Semeion.	67
Tabela 16 – Erro médio de teste (erro médio de treinamento) $\pm$ desvio padrão do erro de teste [parâmetro p], considerando DBMs sem regularização e otimizadas com <i>Dropconnect</i> na base de dados Semeion.	67
Tabela 17 – Erro médio de teste (erro médio de treinamento) $\pm$ desvio padrão do erro de teste [parâmetro p], considerando DBMs sem regularização e otimizadas com <i>Dropout</i> na base de dados USPS.	69
Tabela 18 – Erro médio de teste (erro médio de treinamento) $\pm$ desvio padrão do erro de teste [parâmetro p], considerando DBMs sem regularização e otimizadas com <i>Dropconnect</i> na base de dados USPS.	69

Tabela 19 – Erro médio de teste (erro médio de treinamento) $\pm$ desvio padrão do erro de teste [parâmetro p], considerando DBNs sem regularização e otimizadas com <i>Dropout</i> na base de dados MNIST.	71
Tabela 20 – Erro médio de teste (erro médio de treinamento) $\pm$ desvio padrão do erro de teste [parâmetro p], considerando DBNs sem regularização e otimizadas com <i>Dropconnect</i> na base de dados MNIST.	72
Tabela 21 – Erro médio de teste (erro médio de treinamento) $\pm$ desvio padrão do erro de teste [parâmetro p], considerando DBNs sem regularização e otimizadas com <i>Dropout</i> na base de dados Semeion.	74
Tabela 22 – Erro médio de teste (erro médio de treinamento) $\pm$ desvio padrão do erro de teste [parâmetro p], considerando DBNs sem regularização e otimizadas com <i>Dropconnect</i> na base de dados Semeion.	74
Tabela 23 – Erro médio de teste (erro médio de treinamento) $\pm$ desvio padrão do erro de teste [parâmetro p], considerando DBNs sem regularização e otimizadas com <i>Dropout</i> na base de dados USPS.	76
Tabela 24 – Erro médio de teste (erro médio de treinamento) $\pm$ desvio padrão do erro de teste [parâmetro p], considerando DBNs sem regularização e otimizadas com <i>Dropconnect</i> na base de dados USPS.	76
Tabela 25 – Acurácia média de teste $\pm$ desvio padrão da acurácia de teste [parâmetro médio p], considerando CNNs sem regularização e otimizadas com <i>Dropout</i> na base de dados MNIST.	78
Tabela 26 – Acurácia média de teste $\pm$ desvio padrão da acurácia de teste [parâmetro médio p], considerando CNNs sem regularização e otimizadas com <i>Dropout</i> na base de dados Semeion.	79
Tabela 27 – Acurácia média de teste $\pm$ desvio padrão da acurácia de teste [parâmetro médio p], considerando CNNs sem regularização e otimizadas com <i>Dropout</i> na base de dados USPS.	79
Tabela 28 – Acurácia média de teste $\pm$ desvio padrão da acurácia de teste [parâmetro médio p], considerando CNNs sem regularização e otimizadas com <i>Dropout</i> na base de dados CIFAR-10.	80
Tabela 29 – Acurácia média de teste $\pm$ desvio padrão da acurácia de teste [parâmetro médio p], considerando CNNs sem regularização e otimizadas com <i>Dropout</i> na base de dados Brain-PET.	80
Tabela 30 – Acurácia média de teste $\pm$ desvio padrão da acurácia de teste [parâmetro médio p], considerando CNNs sem regularização e otimizadas com <i>Dropout</i> na base de dados Brain-MRI.	81

Lista de abreviaturas e siglas

BA	<i>Bat Algorithm</i>
BRBM	<i>Bernoulli Restricted Boltzmann Machine</i>
CD	<i>Contrastive Divergence</i>
CNN	<i>Convolutional Neural Network</i>
CS	<i>Cuckoo Search</i>
DBM	<i>Deep Boltzmann Machine</i>
DBN	<i>Deep Belief Network</i>
DL	<i>Deep Learning</i>
FA	<i>Firefly Algorithm</i>
GPU	<i>Graphics Processor Unit</i>
LSTM	<i>Long Short-Term Memory</i>
MCI	<i>Mild Cognitive Impairment</i>
MRI	<i>Magnetic Resonance Imaging</i>
MSE	<i>Mean Squared Error</i>
PET	<i>Positron Emission Tomography</i>
PSO	<i>Particle Swarm Optimization</i>
RBM	<i>Restricted Boltzmann Machine</i>
USPS	<i>United States Postal Service</i>

Sumário

1	INTRODUÇÃO	16
1.1	Problema	18
1.2	Proposta	19
1.3	Objetivos	19
1.3.1	Objetivo Geral	19
1.3.2	Objetivos Específicos	19
1.4	Estrutura da Dissertação	20
2	APRENDIZADO EM PROFUNDIDADE	21
2.1	Máquinas de Boltzmann Restritas	23
2.1.1	Etapa de Treinamento	25
2.1.2	Divergência Contrastiva	27
2.2	Máquinas de Boltzmann em Profundidade	30
2.3	Redes de Crença em Profundidade	32
2.4	Redes Neurais Convolucionais	33
3	TÉCNICAS DE REGULARIZAÇÃO	36
3.1	<i>Dropout</i>	37
3.2	<i>Dropconnect</i>	38
4	OTIMIZAÇÃO META-HEURÍSTICA	40
4.1	Algoritmo do Morcego	42
4.2	Algoritmo do Vagalume	43
4.3	Busca do Cuckoo	46
4.4	Otimização por Enxame de Partículas	47
5	METODOLOGIA	49
5.1	Bases de Dados	50
5.1.1	CIFAR-10	50
5.1.2	MNIST	51
5.1.3	Semeion Handwritten Digits	51
5.1.4	USPS	51
5.1.5	Brain-PET	53
5.1.6	Brain-MRI	53
5.2	Configuração Experimental	54
5.2.1	Experimento A - RBMs	55

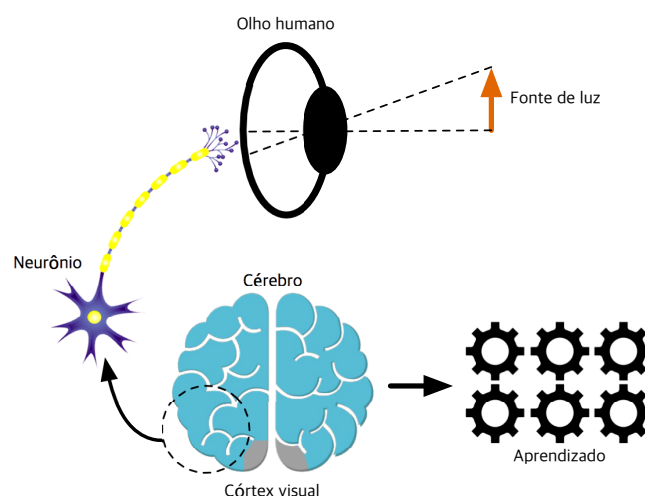
5.2.2	Experimento B - DBMs	56
5.2.3	Experimento C - DBNs	57
5.2.4	Experimento D - CNNs	57
6	EXPERIMENTOS E RESULTADOS	60
6.1	Experimento A - RBMs	60
6.2	Experimento B - DBMs	64
6.3	Experimento C - DBNs	71
6.4	Experimento D - CNNs	78
7	CONCLUSÃO	82
7.1	Direcionamentos para Trabalhos Futuros	83
7.2	Publicações Realizadas	84
	REFERÊNCIAS	86

1 Introdução

Uma das principais questões relacionadas à visão computacional consiste em produzir boas representações do mundo real, de modo a criar descrições que possam permitir a detecção e classificação de objetos em rótulos por meio de um sistema de aprendizado de máquina (BISHOP, 1995). O problema ainda persiste quando confrontado com situações em que existam variações de luminosidade no ambiente, bem como diferentes perspectivas no processo de aquisição de imagem e problemas relacionados à rotação, translação e escala. Portanto, a grande questão a ser respondida por pesquisadores desta área corresponde à simplicidade com que seres humanos e animais aprendem apenas olhando ao seu redor (LECUN; KAVUKCUOGLU; FARABET, 2010).

Tradicionais abordagens para o aprendizado de máquina procuram resolver a situação apresentada via extração de vetores de características, com o propósito de alimentar um classificador por meio de um conjunto de treinamento e, posteriormente, classificar as imagens restantes. Embora o problema de aprendizado de características tenha recebido grande atenção nas últimas décadas, um esforço considerável vem sendo dedicado ao estudo de técnicas de aprendizado em profundidade (*Deep Learning* - DL) (LECUN et al., 1998; AREL; ROSE; KARNOWSKI, 2010; SOCHER et al., 2011; FARABET et al., 2013; KAVUKCUOGLU et al., 2010). Estas técnicas são baseados no aprendizado hierárquico de características, dado que o processamento visual do ser humano também é efetuado da mesma forma, onde a informação é comprimida pela visão (olhos) e processada pelo procedimento de aprendizado (córtex visual), ilustrado pela Figura 1.

Figura 1 – Exemplo do procedimento de aprendizado do ser humano através de sua visão.

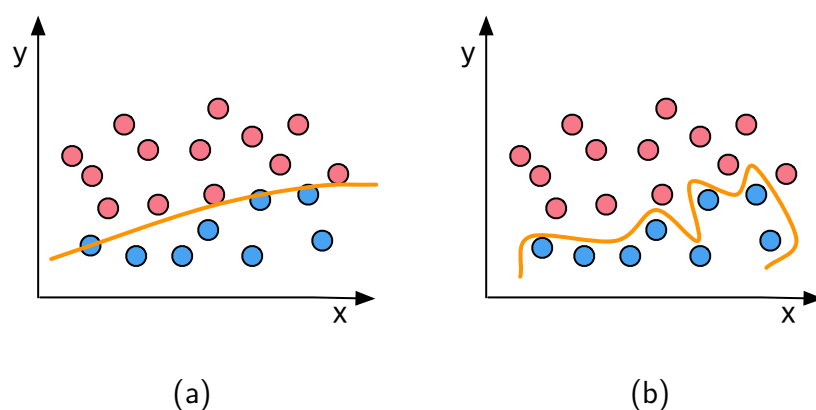


Fonte: Elaborado pelo autor.

Redes Neurais Convolucionais, do inglês *Convolutional Neural Networks* (CNNs) (LECUN et al., 1998), e Máquinas de Boltzmann Restritas, do inglês *Restricted Boltzmann Machines* (RBMs) (SMOLENSKY, 1986), estão entre as técnicas de aprendizado de máquina mais utilizadas atualmente. As CNNs modelam o processamento de informações hierárquico do ser humano de uma maneira mais direta, composto por três etapas: convoluções com diferentes filtros no sinal de entrada, seguido por uma amostragem do sinal e, posteriormente, uma normalização. Já as RBMs podem ser classificadas como redes neurais estocásticas, onde um conjunto de unidades invisíveis, também chamadas de unidades ocultas, tentam modelar os dados de entrada. Contudo, uma RBM propriamente dita não é classificada como um método de DL, mas o seu “empilhamento” sim, onde a saída de uma RBM pode ser utilizada como entrada para outra, formando uma arquitetura composta por várias camadas, denominada de Redes de Crença em Profundidade, do inglês *Deep Belief Nets* (DBNs) (HINTON; OSINDERO; TEH, 2006). Uma variante desta arquitetura consiste nas chamadas Máquinas de Boltzmann em Profundidade, do inglês *Deep Boltzmann Machines* (DBMs) (SALAKHUTDINOV; HINTON, 2012).

Entretanto, o treinamento de técnicas de aprendizado em profundidade em grandes bases de dados ou até mesmo em bases que possuam um alto número de parâmetros acarretam dois principais problemas. O primeiro problema refere-se ao super-treinamento (*overfitting*) (HAWKINS, 2004) dos dados e o segundo à uma apropriada seleção dos parâmetros utilizados (PAPA; SCHEIRER; COX, 2016). A principal ideia por trás do primeiro problema consiste em que, durante o treinamento, os neurônios podem adaptar-se à matriz de pesos obtida através da quantidade limitada de dados de treino. Em outras palavras, quando trabalhamos com informações complexas, os neurônios podem acomodar-se entre si durante o treinamento e “memorizar” os dados ao invés de aprendê-los, resultando em péssimos modelos de predição. A Figura 2 ilustra exemplos de um treinamento adequado e um super-treinamento dos dados.

Figura 2 – Exemplo de (a) treinamento correto e (b) super-treinamento dos dados.



Fonte: Elaborado pelo autor.

Com relação ao segundo problema, pode-se solucioná-lo a partir da aplicação de procedimentos de otimização nas técnicas de aprendizado de máquina, os quais almejam encontrar um conjunto plausível de parâmetros que minimizam ou maximizam alguma função de aptidão, tal como a qualidade do agrupamento (*clustering*) em abordagens não-supervisionadas, ou até mesmo a acurácia do reconhecimento de um classificador em problemas supervisionados. Técnicas meta-heurísticas são vastamente empregadas em problemas de otimização, dado que elas provêm soluções simples e de fácil compreensão para uma extensa gama de aplicações. Estas técnicas podem contemplar algoritmos com heurísticas baseadas em políticas populacionais e de enxame, bem como soluções estocásticas ou inspiradas pela própria natureza.

1.1 Problema

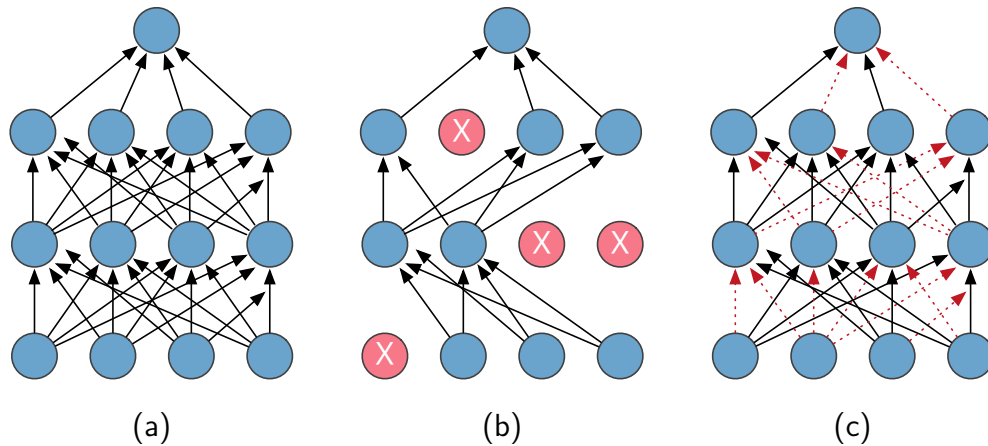
Um dos principais problemas relativos ao aprendizado em profundidade consiste no super-treinamento dos dados, visto que técnicas baseadas em DL possuem, geralmente, uma arquitetura complexa. O ponto negativo de uma arquitetura deste porte é que a mesma pode “aprender” como o treinamento se comporta, adaptar-se e, conseqüentemente, não obter resultados desejáveis em futuras predições.

Diversas abordagens foram desenvolvidas a fim de contornar o problema de *overfitting*, tais como, parar a etapa de treinamento assim que sua performance no conjunto de validação começar a decair, ou introduzir métodos de regularização, como o *soft-weight sharing*¹ (NOWLAN; HINTON, 1992). Por outro lado, a melhor alternativa para abordar uma regularização seria empregar uma média entre as predições de todas as possíveis configurações de parâmetros, ponderando todas as possibilidades e checando a melhor alternativa. Contudo, tal procedimento exige um alto custo computacional, sendo apenas viável para pequenos ou simples modelos (XIONG; BARASH; FREY, 2011).

Uma interessante tentativa de solucionar este problema é considerar diferentes modelos juntos, onde a combinação de diferentes arquiteturas auxilia na redução do ruído obtido no treinamento. Todavia, treinar diferentes arquiteturas é um processo moroso, o qual exige uma considerável quantia de dados de treinamento. Recentemente, Srivastava et al. (SRIVASTAVA et al., 2014) propuseram uma técnica denominada *Dropout*, a qual visa “desligar” neurônios aleatoriamente durante o processo de convergência. Esta técnica demonstrou-se bastante eficaz no combate ao super-treinamento dos dados. Outra iniciativa diz respeito à técnica *Dropconnect* (WAN et al., 2013), a qual desliga conexões ao invés de neurônios. Muito embora estas propostas obtiveram resultados superiores ao estado da arte, elas possuem parâmetros como, o limiar para desligamento de neurônios ou conexões. A Figura 3 ilustra exemplos de redes com as técnicas de regularização mencionadas acima.

¹ *Weight sharing* é o uso compartilhado de pesos pelos neurônios de uma rede, enquanto que sua versão “soft” permite com que a rede seja capaz de decidir quais pesos reutilizará ao longo do treinamento.

Figura 3 – Exemplo de (a) arquitetura normal de rede, (b) arquitetura com *Dropout* e (c) arquitetura com *Dropconnect*.



Fonte: Elaborado pelo autor.

1.2 Proposta

A presente dissertação pretende identificar possíveis soluções para o problema mencionado por meio de técnicas de otimização meta-heurística, objetivando encontrar o número adequado para o limiar de desligamento dos neurônios e conexões. Diferentes abordagens de aprendizado em profundidade, tais como, Máquinas de Boltzmann Restritas, Máquinas de Boltzmann em Profundidade, Redes de Crença em Profundidade, Redes Neurais Convolucionais; e diferentes meta-heurísticas, tais como, Algoritmo do Morcego, Algoritmo do Vagalume, Busca do Cuckoo, Otimização por Enxame de Partículas, serão utilizadas a fim de tentar estabelecer um conhecimento mais robusto sobre a semântica de distintas arquiteturas e aplicações das mesmas. Até onde os autores têm conhecimento, ainda não existem trabalhos com estas características.

1.3 Objetivos

1.3.1 Objetivo Geral

Aliar o uso da otimização meta-heurística ao problema de seleção de parâmetros de técnicas de regularização de aprendizado em profundidade, especificamente a seleção da taxa de desligamento de neurônios e conexões. Esta análise possibilitará adquirir um maior conhecimento sobre as melhores arquiteturas neurais para diversas aplicações.

1.3.2 Objetivos Específicos

A validação em diferentes áreas de aplicação para o problema proposto por este trabalho é de grande valia, sendo os passos necessários à sua obtenção descritos a seguir:

- a) estudar sobre as principais técnicas de aprendizado em profundidade, compreendendo seus principais problemas;
- b) analisar formas de regularização para solucionar o problema de super-treinamento do aprendizado em profundidade;
- c) estudar técnicas meta-heurísticas de otimização, dentre elas Otimização por Enxame de Partículas, Busca do Cuckoo, Algoritmo do Morcego e Algoritmo do Vagalume;
- d) modelar as abordagens de regularização como problemas de otimização, encontrando a mais adequada seleção de parâmetros;
- e) testar e validar a metodologia proposta em diferentes bases de dados; e
- f) verificar os resultados obtidos entre as técnicas utilizadas através de métodos estatísticos, provendo uma maior confiabilidade à parte experimental.

1.4 Estrutura da Dissertação

Esta dissertação de mestrado está organizada da seguinte forma: o Capítulo 2 apresenta conceitos e definições sobre o aprendizado em profundidade e algumas de suas técnicas. O Capítulo 3 apresenta duas técnicas de regularização empregadas para prevenir o problema de super-treinamento. O Capítulo 4 apresenta conceitos e definições sobre otimização e técnicas meta-heurísticas. O Capítulo 5 apresenta o método proposto para a otimização dos parâmetros das técnicas de regularização. O Capítulo 6 apresenta os experimentos e os resultados obtidos nesta dissertação. Finalmente, o Capítulo 7 apresenta a conclusão desta dissertação, bem como futuros trabalhos e as publicações realizadas durante o período da mesma.

2 Aprendizado em Profundidade

A necessidade de um método automático de análise de dados, fomentado pelo avanço tecnológico, contribuiu para o desenvolvimento da área de aprendizado de máquina. A partir do uso de algoritmos que aprendem iterativamente com os dados, o aprendizado de máquina permite que computadores encontrem padrões escondidos sem terem uma programação específica para tal propósito. O aspecto iterativo do aprendizado de máquina é um dos principais conceitos desta área, dado que suas técnicas são capazes de adaptarem-se independentemente, ou seja, baseiam-se em cálculos anteriores para produzirem suas decisões e resultados.

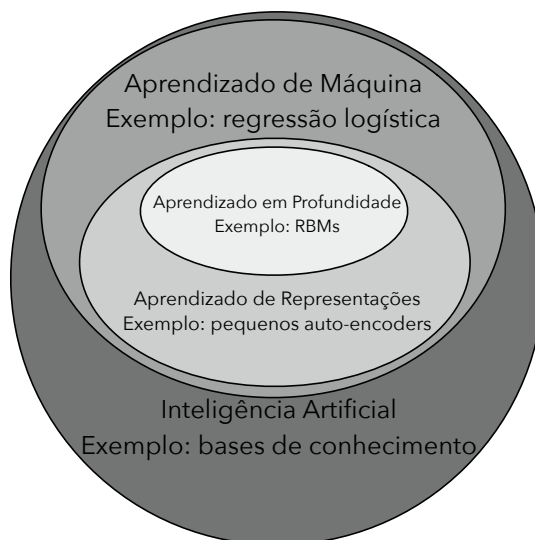
Entretanto, tal tarefa torna-se demasiadamente complexa quando empregada em um domínio de aplicações reais. A diversidade no processo de aquisição de dados contribui para este problema, dado que existem muitos fatores de variação que influenciam cada informação analisada. Por exemplo, quando analisamos uma gravação de voz, podemos citar como fatores de variação o timbre da voz de uma pessoa, bem como seu sotaque e até mesmo a sua língua materna. Por outro lado, quando analisamos imagens, os *pixels* coloridos de uma imagem pode sofrer variações de acordo com a luminosidade do ambiente ou até mesmo com fatores relacionados à rotação e escala da própria imagem.

A questão central que podemos levantar destas análises é como extrair tais informações de alto-nível a partir de dados brutos. Considerando este fato, uma subárea do aprendizado de máquina vem sendo extensamente pesquisada, denominada de aprendizado em profundidade. A combinação do avanço computacional com tipos específicos de redes neurais permite a aprendizagem de padrões complicados em grandes volumes de dados. A principal ideia por trás desta abordagem é introduzir um método de aprendizagem de representações, ou seja, representar informações sofisticadas a partir de representações mais simples. Este processo é similar ao aprendizado biológico dos seres humanos, onde nosso cérebro é responsável por hierarquizar informações e, a partir de entradas mais simples, construir conhecimentos mais robustos e complexos.

Embora o aprendizado em profundidade pareça ser uma nova tecnologia, esta área começou a ser desenvolvida por volta do início da década de 40. Com uma aparência relativamente nova, o aprendizado em profundidade passou por décadas de impopularidade e diversos nomes até chegar nos moldes atuais. Inicialmente, por volta dos anos 1940 - 1960, o aprendizado em profundidade era conhecido por cibernética. Mais tarde, por volta dos anos 1980 - 1990, passou a ser nomeado de connexionismo. E finalmente, a partir do início de 2006, recebeu seu nome atual. A Figura 4 é responsável por ilustrar o diagrama de Venn da área da inteligência artificial.

Os mais antigos algoritmos tinham por objetivo modelar computacionalmente o apren-

Figura 4 – Diagrama de Venn ilustrando as sub-áreas referentes às técnicas de inteligência artificial.



Fonte: Elaborado pelo autor.

dizado biológico, ou seja, modelos de como o aprendizado ocorre ou até mesmo como o cérebro funciona. Um dos principais algoritmos que podem ser citados são as Redes Neurais Artificiais, do inglês *Artificial Neural Networks* (ANN), inspiradas pelo funcionamento cerebral de seres vivos. Contudo, o aprendizado em profundidade não se restringe a tal ideia, indo além da perspectiva neurocientífica, ou seja, sua principal motivação é aprender diferentes níveis de composição de informações, podendo ser aplicado em infinitos domínios, não se restringindo aos inspirados neurologicamente.

O neurônio de McCulloch-Pitts (MCCULLOCH; PITTS, 1943) foi uma das primeiras modelagens da função cerebral. Este modelo linear reconhecia duas categorias de entrada, se $f(x, w)$ era positivo ou negativo, porém, necessitava da correta seleção de pesos. Já por volta do início de 1950, o Perceptron (ROSENBLATT, 1958) tornou-se o primeiro modelo a aprender pesos automaticamente, dada as informações de entrada. Entretanto, tais modelos, por serem lineares, possuíam diversas limitações, sendo a mais conhecida a incapacidade de aprender a função lógica XOR, onde $f(0, 1) = 1$ e $f(1, 0) = 1$ mas $f(1, 1) = 0$ e $f(0, 0) = 0$. Tal problema foi extensamente observado por críticos, causando uma perda na popularidade de técnicas biologicamente inspiradas (MINSKY; PAPERT, 1969). Apenas na década de 70, os originais “Cognitron” (FUKUSHIMA, 1975) e “Neocognitron” (FUKUSHIMA, 1980) introduziram modelos de processamento de imagens inspirados pelo sistema visual dos mamíferos.

Posteriormente, na década de 80, a segunda onda de pesquisa em redes neurais emergiu, fazendo parte do movimento conexionista (RUMELHART; MCCLELLAND; GROUP, 1988; MCCLELLAND; RUMELHART; HINTON, 1986). Basicamente, a principal ideia por trás deste movimento é mostrar que um grande número de unidades computacionais podem adquirir um comportamento inteligente quando utilizadas conjuntamente. Diversos conhecimentos deste

movimento ainda permanecem como pontos centrais do atual aprendizado em profundidade. A representação distribuída (HINTON, 1986) descreve que cada entrada do sistema deve ser representada por diversas características, e cada característica deve estar relacionada à representações de diversas possíveis entradas. Por exemplo, num sistema de reconhecimento de carros, caminhões e pássaros, e cores vermelhas, azuis e verdes, podemos ter um neurônio que reconhece o objeto e outro neurônio que reconhece sua cor ao invés de possuir um neurônio para cada combinação do problema. Outro importante conceito advindo do movimento conexionista é o uso da retropropagação, do inglês *backpropagation* (RUMELHART; HINTON; WILLIAMS, 1986; LECUN, 1987), para treinamento de redes neurais em profundidade.

Durante a década de 90, pesquisadores realizaram importantes avanços na modelagem de sequências a partir de redes neurais. Hochreiter (HOCHREITER, 1991) e Bengio et. al (BENGIO; SIMARD; FRASCONI, 1994) identificaram algumas das dificuldades matemáticas em modelar extensas sequências, sendo tal problema resolvido pela rede de longa memória de curto prazo, do inglês *Long Short-Term Memory* (LSTM) (HOCHREITER; SCHMIDHUBER, 1997). Atualmente, tal técnica de aprendizado em profundidade é vastamente utilizada em tarefas de modelagens de sequências, incluindo diversas tarefas de processamento de linguagem natural.

Finalmente, no ano de 2006, Geoffrey Hinton contribuiu para o início da terceira onda do aprendizado em profundidade, introduzindo a Rede de Crença em Profundidade (HINTON; OSINDERO; TEH, 2006). Esta última onda de redes neurais acabou por popularizar o termo “aprendizado em profundidade”, enfatizando a possibilidade de treinar redes ainda mais profundas, atingindo um novo patamar dentro dos sistemas de inteligência artificial. Recentemente, técnicas de aprendizado em profundidade obtiveram os melhores resultados para identificação de objetos em imagens e palavras em áudios (GRAHAM, 2014; CLEVERT; UNTERTHINER; HOCHREITER, 2015; HINTON et al., 2012).

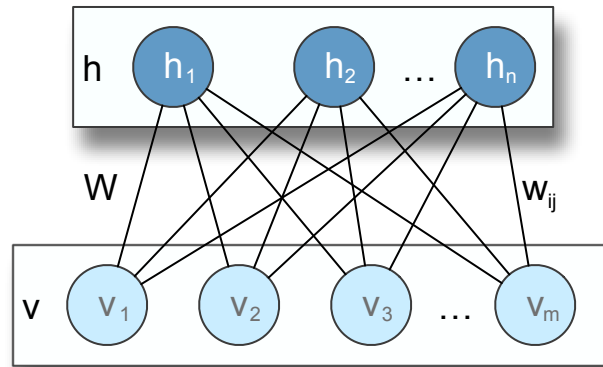
2.1 Máquinas de Boltzmann Restritas

As Máquinas de Boltzmann Restritas são redes neurais estocásticas baseadas em energia e compostas por duas camadas de neurônios (visíveis e escondidos), onde sua fase de treinamento é conduzida por meios de um aprendizado não supervisionado. A RBM é similar à clássica Máquina de Boltzmann (ACKLEY; HINTON; SEJNOWSKI, 1988), exceto que não existem conexões entre os neurônios da mesma camada². A Figura 5 descreve a arquitetura de uma RBM, a qual compreende uma camada visível $\mathbf{v}$ com m unidades e uma camada oculta $\mathbf{h}$ com n unidades. O valor real $m \times n$ da matriz $\mathbf{W}$ modela os pesos entre os neurônios visíveis e ocultos, onde w_{ij} é o peso entre a unidade visível v_i e a unidade oculta h_j .

Inicialmente, as RBMs foram desenvolvidas usando somente unidades binárias visíveis

² Essencialmente, uma RBM é modelada por um grafo bipartido.

Figura 5 – Arquitetura de uma RBM.



Fonte: Elaborado pelo autor.

e ocultas, as chamadas Máquinas de Boltzmann Restritas de Bernoulli, do inglês *Bernoulli RBMs* (BRBMs). Posteriormente, Welling et al. (WELLING; ROSEN-ZVI; HINTON, 2005) apresentaram outros tipos de unidades que podem ser usadas em uma RBM, como unidades binomiais e gaussianas³. Como neste estudo estamos interessados nas BRBMs, introduziremos seus principais conceitos, que servem como base para outras generalizações das RBMs.

Assumiremos $\mathbf{v}$ e $\mathbf{h}$ como as unidades binárias visíveis e ocultas, respectivamente. Em outras palavras, $\mathbf{v} \in \{0, 1\}^m$ e $\mathbf{h} \in \{0, 1\}^n$. A função de ativação de uma Máquina de Boltzmann Restrita de Bernoulli é dada por:

$$E(\mathbf{v}, \mathbf{h}) = - \sum_{i=1}^m a_i v_i - \sum_{j=1}^n b_j h_j - \sum_{i=1}^m \sum_{j=1}^n v_i h_j w_{ij}, \quad (1)$$

onde $\mathbf{a}$ e $\mathbf{b}$ são os valores dos vieses (*biases*) das unidades visíveis e ocultas, respectivamente. A probabilidade de uma configuração $(\mathbf{v}, \mathbf{h})$ é calculada como segue:

$$P(\mathbf{v}, \mathbf{h}) = \frac{e^{-E(\mathbf{v}, \mathbf{h})}}{\sum_{\mathbf{v}, \mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})}}, \quad (2)$$

onde o denominador da equação acima é um fator de normalização que serve para todas as possíveis configurações envolvendo unidades visíveis e ocultas⁴. Em resumo, o algoritmo de treinamento de uma BRBM tem por objetivo estimar $\mathbf{W}$, $\mathbf{a}$ e $\mathbf{b}$. A próxima seção descreve com mais detalhes esse procedimento.

³ Essas abordagens também são conhecidas como Máquinas de Boltzmann Restritas Gaussianas quando os neurônios da camada de entrada são modelados por unidades gaussianas.

⁴ Note que este fator de normalização é extremamente difícil de ser computado quando o número de unidades é muito grande.

2.1.1 Etapa de Treinamento

Os parâmetros de uma BRBM podem ser otimizados executando a técnica de gradiente ascendente estocástico no logaritmo da verossimilhança dos dados de treinamento. Assim, dada uma amostra de treinamento (unidade visível), a sua probabilidade é calculada sobre todas as possíveis unidades escondidas, conforme segue:

$$P(\mathbf{v}) = \frac{\sum_{\mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})}}{\sum_{\mathbf{v}, \mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})}}. \quad (3)$$

Para atualizar os pesos e os *biases*, é necessário computar as seguintes derivadas:

$$\frac{\partial \log P(\mathbf{v})}{\partial w_{ij}} = E[h_j v_i]^{dado} - E[h_j v_i]^{modelo}, \quad (4)$$

$$\frac{\partial \log P(\mathbf{v})}{\partial a_i} = E[v_i]^{dado} - E[v_i]^{modelo}, \quad (5)$$

$$\frac{\partial \log P(\mathbf{v})}{\partial b_j} = E[h_j]^{dado} - E[h_j]^{modelo}, \quad (6)$$

onde $E[\cdot]$ denota a operação de esperança estatística, e $E[\cdot]^{dado}$ e $E[\cdot]^{modelo}$ correspondem à probabilidade dos dados originais e da sua versão reconstruída, respectivamente.

Em termos práticos, podemos computar $E[h_j v_i]^{dado}$ considerando $\mathbf{h}$ e $\mathbf{v}$ conforme segue:

$$E[\mathbf{h}\mathbf{v}]^{dado} = P(\mathbf{h}|\mathbf{v})\mathbf{v}^T, \quad (7)$$

onde $P(\mathbf{h}|\mathbf{v})$ é a probabilidade de obter $\mathbf{h}$ dado um vetor visível (dado de treinamento) $\mathbf{v}$:

$$P(h_j = 1|\mathbf{v}) = \sigma \left(\sum_{i=1}^m w_{ij} v_i + b_j \right), \quad (8)$$

onde $\sigma(\cdot)$ é a função sigmóide-logística⁵. Por conseguinte, o cálculo para $E[\mathbf{h}\mathbf{v}]^{dado}$ é realizado da seguinte maneira: dada uma amostra de treinamento $\mathbf{x} \in X$, onde X é um conjunto de treinamento, precisamos ajustar $\mathbf{v} \leftarrow \mathbf{x}$ e então utilizar a Equação 8 para obter $P(\mathbf{h}|\mathbf{v})$. Além disso, usamos a Equação 7 para finalmente obter $E[\mathbf{h}\mathbf{v}]^{dado}$.

A grande questão agora é como obter $E[\mathbf{h}\mathbf{v}]^{modelo}$, o qual é o modelo aprendido pelo sistema⁶. Uma possível estratégia seria obtê-lo por meio da técnica de amostragem de Gibbs,

⁵ A função sigmóide-logística pode ser computada pela seguinte equação: $\sigma(x) = 1/(1 + \exp(-x))$.

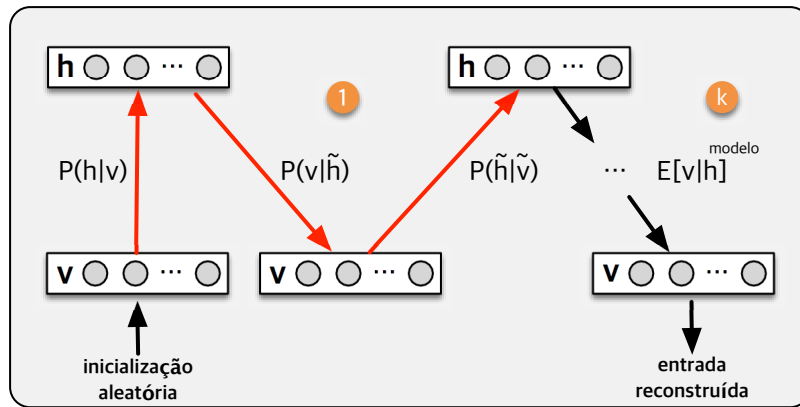
⁶ Estamos escrevendo agora $E[h_j v_i]^{modelo}$ em termos de $\mathbf{h}$ e $\mathbf{v}$.

iniciando-se as unidades visíveis com valores aleatórios até um certo critério de convergência ser atingido, como k iterações, por exemplo. A amostragem de Gibbs consiste em atualizar as unidades ocultas utilizando a Equação 8 seguida pela atualização das unidades visíveis usando $P(\mathbf{v}|\mathbf{h})$, dada por:

$$P(v_i = 1|\mathbf{h}) = \sigma \left(\sum_{j=1}^n w_{ij}h_j + a_i \right), \quad (9)$$

e então atualizando as unidades ocultas utilizando novamente a Equação 8. Em resumo, é possível obter uma estimativa de $E[\mathbf{h}\mathbf{v}]^{\text{modelo}}$ inicializando a unidade visível com valores aleatórios e então efetuando a amostragem de Gibbs. A Figura 6 ilustra esse processo, em que $E[\mathbf{h}\mathbf{v}]^{\text{modelo}}$ pode ser aproximado após k iterações. Note que uma simples iteração é definida computando $P(\mathbf{h}|\mathbf{v})$, seguida pelo cálculo de $P(\mathbf{v}|\mathbf{h})$ e então calculando $P(\mathbf{h}|\mathbf{v})$ novamente.

Figura 6 – Amostragem de Gibbs. As setas “vermelhas” denotam uma única iteração.



Fonte: Elaborado pelo autor.

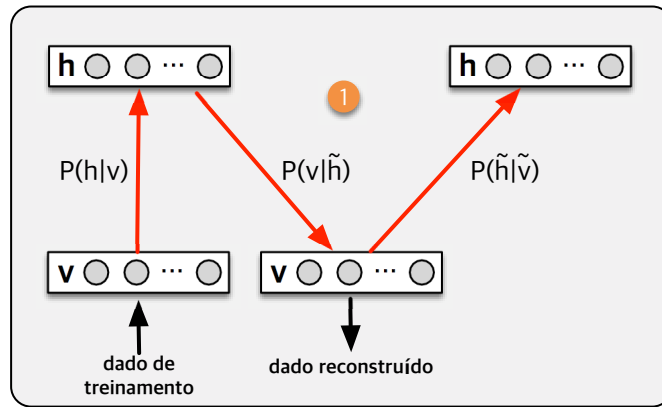
Por motivos de clareza, a Figura 6 emprega $P(\mathbf{v}|\tilde{\mathbf{h}})$ ao invés de $P(\mathbf{v}|\mathbf{h})$, e $P(\tilde{\mathbf{h}}|\tilde{\mathbf{v}})$ ao invés de $P(\mathbf{h}|\mathbf{v})$. Essencialmente, eles possuem o mesmo significado, mas $P(\mathbf{v}|\tilde{\mathbf{h}})$ é usada aqui para denotar a unidade visível $\mathbf{v}$ que irá ser reconstruída utilizando $\tilde{\mathbf{h}}$, a qual foi obtida através $P(\mathbf{h}|\mathbf{v})$. O mesmo acontece com $P(\tilde{\mathbf{h}}|\tilde{\mathbf{v}})$, que reconstrói $\tilde{\mathbf{h}}$ usando $\tilde{\mathbf{v}}$, o qual foi obtido através de $P(\mathbf{v}|\tilde{\mathbf{h}})$.

Entretanto, o procedimento apresentado na Figura 6 é bastante caro computacionalmente, dado que é esperada uma boa reconstrução do modelo quando $k \rightarrow +\infty$. Desta forma, alguns trabalhos apresentaram alternativas à amostragem de Gibbs, sendo alguns deles o método da Divergência Contrastiva (*Contrastive Divergence* - CD) (HINTON, 2002), Divergência Contrastiva Persistente (*Persistent Contrastive Divergence* - PCD) (TIELEMAN, 2008) e a abordagem baseada em Têmperas Paralelas (*Parallel Tempering* - PT) (DESJARDINS et al., 2010). Todas estas abordagens são baseadas em cadeias de Markov, sendo basicamente utilizadas para amostrar dados destas cadeias.

2.1.2 Divergência Contrastiva

Hinton (HINTON, 2002) introduziu uma metodologia mais rápida para o cálculo de $E[\mathbf{h}\mathbf{v}]^{modelo}$ baseado na ideia da divergência contrastiva (*Contrastive Divergence* - CD). Basicamente, a ideia é inicializar as unidades visíveis com uma amostra de treinamento e estimar os estados das unidades ocultas utilizando a Equação 8, e então, calcular os estados das unidades visíveis (passo de reconstrução) por meio da Equação 9. Em suma, isso seria equivalente a executar a amostragem de Gibbs usando $k = 1$ com a cadeia inicializada com uma amostra de treinamento. A Figura 7 ilustra esta abordagem e o Algoritmo 1 é responsável por descrever o pseudo-código do treinamento de uma RBM.

Figura 7 – Ilustração do método baseado em divergência contrastiva.



Fonte: Elaborado pelo autor.

A noção de “cadeia de Markov” em uma BRBM, basicamente, diz respeito ao conjunto de dados reconstruídos durante o processo de amostragem, os quais são calculados pela Equação 9. Seja $P^k(\mathbf{v}|\mathbf{h})$ a estimativa do dado de entrada $\mathbf{v}$ na k -ésima iteração de algum processo de amostragem. Assim sendo, para um processo com $k = 3$ iterações, por exemplo, temos a seguinte cadeia: $P^1(\mathbf{v}|\mathbf{h}) \rightarrow P^2(\mathbf{v}|\mathbf{h}) \rightarrow P^3(\mathbf{v}|\mathbf{h})$, onde $P^1(\mathbf{v}|\mathbf{h})$ denota a primeira etapa de reconstrução (seria o modelo estimado pelo método CD), e $P^3(\mathbf{v}|\mathbf{h})$ denota o último elemento dessa cadeia.

Baseados na conclusão acima, podemos calcular $E[\mathbf{h}\mathbf{v}]^{modelo}$ como segue:

$$E[\mathbf{h}\mathbf{v}]^{modelo} = P(\tilde{\mathbf{h}}|\tilde{\mathbf{v}})\tilde{\mathbf{v}}^T. \quad (10)$$

Portanto, a equação abaixo leva à uma simples regra de aprendizado para atualizar a matriz de peso $\mathbf{W}$, como segue:

$$\begin{aligned}
\mathbf{W}^{t+1} &= \mathbf{W}^t + \eta(E[\mathbf{h}\mathbf{v}]^{dado} - E[\mathbf{h}\mathbf{v}]^{modelo}) \\
&= \mathbf{W}^t + \eta(P(\mathbf{h}|\mathbf{v})\mathbf{v}^T - P(\tilde{\mathbf{h}}|\tilde{\mathbf{v}})\tilde{\mathbf{v}}^T),
\end{aligned} \tag{11}$$

onde $\mathbf{W}^t$ é a matriz de pesos no momento t , e η corresponde à taxa de aprendizado. Adicionalmente, temos as seguintes formulações para atualizar os *biases* das unidades visíveis e ocultas:

$$\begin{aligned}
\mathbf{a}^{t+1} &= \mathbf{a}^t + \eta(\mathbf{v} - E[\mathbf{v}]^{modelo}) \\
&= \mathbf{a}^t + \eta(\mathbf{v} - \tilde{\mathbf{v}}),
\end{aligned} \tag{12}$$

e

$$\begin{aligned}
\mathbf{b}^{t+1} &= \mathbf{b}^t + \eta(E[\mathbf{h}]^{dado} - E[\mathbf{h}]^{modelo}) \\
&= \mathbf{b}^t + \eta(P(\mathbf{h}|\mathbf{v}) - P(\tilde{\mathbf{h}}|\tilde{\mathbf{v}})),
\end{aligned} \tag{13}$$

onde $\mathbf{a}^t$ and $\mathbf{b}^t$ são os valores dos *biases* das unidades visíveis e ocultas no momento t , respectivamente. Em resumo, as Equações 11, 12 e 13 são as formulações para atualizar os parâmetros da RBM.

Mais tarde, Hinton (HINTON, 2012) introduziu um parâmetro de decaimento de peso λ , que penaliza os pesos em larga magnitude⁷, bem como um parâmetro de momento α para controlar as possíveis oscilações durante o processo de aprendizado. Portanto, podemos reescrever as Equações 11, 12 e 13 como segue⁸:

$$\mathbf{W}^{t+1} = \mathbf{W}^t + \underbrace{\eta(P(\mathbf{h}|\mathbf{v})\mathbf{v}^T - P(\tilde{\mathbf{h}}|\tilde{\mathbf{v}})\tilde{\mathbf{v}}^T)}_{=\Delta\mathbf{W}^t} - \lambda\mathbf{W}^t + \alpha\Delta\mathbf{W}^{t-1}, \tag{14}$$

$$\mathbf{a}^{t+1} = \mathbf{a}^t + \underbrace{\eta(\mathbf{v} - \tilde{\mathbf{v}})}_{=\Delta\mathbf{a}^t} + \alpha\Delta\mathbf{a}^{t-1} \tag{15}$$

e

$$\mathbf{b}^{t+1} = \mathbf{b}^t + \underbrace{\eta(P(\mathbf{h}|\mathbf{v}) - P(\tilde{\mathbf{h}}|\tilde{\mathbf{v}}))}_{=\Delta\mathbf{b}^t} + \alpha\Delta\mathbf{b}^{t-1}. \tag{16}$$

O algoritmo 1 ilustra um pseudo-código do treinamento de uma BRBM. Note que $\mathcal{U}$ é a distribuição uniforme e MSE o erro médio quadrático.

⁷ Os pesos podem aumentar durante o processo de convergência.

⁸ Note que quando $\lambda = 0$ e $\alpha = 0$, temos o gradiente ascendente.

ALGORITMO 1: Pseudo-código da RBM de Bernoulli.

Entrada: Um conjunto de treinamento $\mathcal{X} = \{x^{(1)}, x^{(2)}, \dots, x^{(l)}\}$, taxa de aprendizado η , número de unidades ocultas n , número de épocas T , fator de penalidade (decaimento de peso) λ , momento α e tamanho do lote (*batch size*) l .

Saída: Matriz de pesos $\mathbf{W}$ e os *biases* $\mathbf{a}$ e $\mathbf{b}$.

```

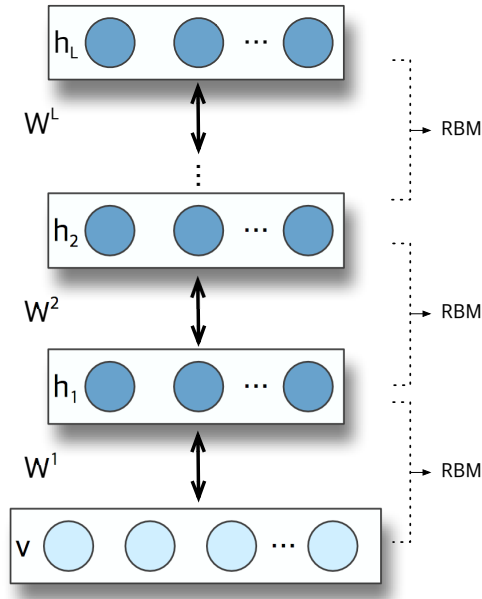
1  $\mathbf{W} \leftarrow \mathcal{N}(0, 0.01)$ ;  $\mathbf{a} \leftarrow 0$ ;  $\mathbf{b} \leftarrow 0$ ;  $t = 1$ ;  $\Delta\mathbf{W} \leftarrow 0$ ;  $\Delta\mathbf{a} \leftarrow 0$ ;  $\Delta\mathbf{b} \leftarrow 0$ ;
2  $erro = +\infty$ ;  $\epsilon = 0.0001$ ;
3 enquanto ( $t \leq T$ ) e ( $erro \geq \epsilon$ ) faça
4    $erro = 0$ ;  $\underline{\mathbf{v}} \leftarrow 0$ ;  $\tilde{\mathbf{v}} \leftarrow 0$ ;  $\underline{\mathbf{h}} \leftarrow 0$ ;  $\tilde{\mathbf{h}} \leftarrow 0$ ;
5   para  $z$  de 1 até  $l$  faça
6      $\mathbf{v} \leftarrow \mathbf{x}^{(z)}$ ;  $\underline{\mathbf{v}} \leftarrow \underline{\mathbf{v}} + \mathbf{v}$ ;
7     para  $j$  de 1 até  $n$  faça
8       Computar  $P(h_j|\mathbf{v})$  de acordo com a Equação 8;
9       se ( $P(h_j|\mathbf{v}) \geq \mathcal{U}(0, 1)$ ) então
10        |  $\tilde{h}_j = 1$ ;
11       senão
12        |  $\tilde{h}_j = 0$ ;
13      $\underline{\mathbf{h}} \leftarrow \underline{\mathbf{h}} + P(\mathbf{h}|\mathbf{v})$ ;
14     para  $i$  de 1 até  $m$  faça
15       Computar  $P(v_i|\tilde{\mathbf{h}})$  de acordo com a Equação 9;
16       se ( $P(v_i|\tilde{\mathbf{h}}) \geq \mathcal{U}(0, 1)$ ) então
17        |  $\tilde{v}_i = 1$ ;
18       senão
19        |  $\tilde{v}_i = 0$ ;
20      $\tilde{\mathbf{v}} \leftarrow \tilde{\mathbf{v}} + \tilde{\mathbf{v}}$ ;
21     para  $j$  de 1 até  $n$  faça
22       Computar  $P(h_j|\tilde{\mathbf{v}})$  de acordo com a Equação 8;
23       se ( $P(h_j|\tilde{\mathbf{v}}) \geq \mathcal{U}(0, 1)$ ) então
24        |  $\tilde{h}_j = 1$ ;
25       senão
26        |  $\tilde{h}_j = 0$ ;
27      $\tilde{\mathbf{h}} \leftarrow \tilde{\mathbf{h}} + P(\tilde{\mathbf{h}}|\mathbf{v})$ ;  $erro = erro + MSE(x^{(z)}, \tilde{\mathbf{v}})$ ;
28    $erro = erro/l$ ;  $\underline{\mathbf{v}} \leftarrow \underline{\mathbf{v}}/l$ ;  $\tilde{\mathbf{v}} \leftarrow \tilde{\mathbf{v}}/l$ ;  $\underline{\mathbf{h}} \leftarrow \underline{\mathbf{h}}/l$ ;  $\tilde{\mathbf{h}} \leftarrow \tilde{\mathbf{h}}/l$ ;
29    $\Delta\mathbf{W} \leftarrow \Delta\mathbf{W} - \lambda\mathbf{W} + \alpha\Delta\mathbf{W}$ ;  $\mathbf{W} \leftarrow \mathbf{W} + \eta(\underline{\mathbf{h}}\underline{\mathbf{v}}^T - \tilde{\mathbf{h}}\tilde{\mathbf{v}}^T) + \Delta\mathbf{W}$ ;
30    $\Delta\mathbf{a} \leftarrow \Delta\mathbf{a} + \alpha\mathbf{a}$ ;  $\mathbf{a} \leftarrow \mathbf{a} + \eta(\underline{\mathbf{v}} - \tilde{\mathbf{v}}) + \Delta\mathbf{a}$ ;
31    $\Delta\mathbf{b} \leftarrow \Delta\mathbf{b} + \alpha\mathbf{b}$ ;  $\mathbf{b} \leftarrow \mathbf{b} + \eta(\underline{\mathbf{h}} - \tilde{\mathbf{h}}) + \Delta\mathbf{b}$ ;
32    $t = t + 1$ ;

```

2.2 Máquinas de Boltzmann em Profundidade

A necessidade de aprender informações mais complexas e intrínsecas fomentou o desenvolvimento de RBMs empilhadas, tais como DBMs e DBNs. A Figura 8 apresenta a arquitetura de uma DBM, sendo cada camada representada por uma RBM. Note que a Figura 8 é uma generalização para uma DBM com L camadas.

Figura 8 – Arquitetura padrão de uma DBM.



Fonte: Elaborado pelo autor.

Suponha que tenhamos uma DBM com duas camadas, onde $\mathbf{h}^1$ e $\mathbf{h}^2$ são as unidades ocultas da primeira e da segunda camada, respectivamente. A energia de uma DBM pode ser calculada pela Equação 17:

$$E(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2) = - \sum_{i=1}^{m^1} \sum_{j=1}^{n^1} v_i h_j^1 w_{ij}^1 - \sum_{i=1}^{m^2} \sum_{j=1}^{n^2} h_i^1 h_j^2 w_{ij}^2, \quad (17)$$

onde m^1 e m^2 são a quantidade de unidades visíveis na primeira e segunda camadas, respectivamente, e n^1 e n^2 a quantidade de unidades ocultas na primeira e segunda camadas respectivamente. Adicionalmente, as matrizes de peso $\mathbf{W}_{m^1 \times n^1}^1$ e $\mathbf{W}_{m^2 \times n^2}^2$ são responsáveis por armazenar os pesos das conexões entre os vetores $\mathbf{v}$ e $\mathbf{h}^1$, e os vetores $\mathbf{h}^1$ e $\mathbf{h}^2$, respectivamente. Para uma explanação mais simplificada, os termos referentes aos *biases* foram omitidos.

A probabilidade mínima que o modelo atribui à um vetor de entrada $\mathbf{v}$ é dada pela Equação 18:

$$P(\mathbf{v}) = \frac{1}{Z} \sum_{\mathbf{h}^1, \mathbf{h}^2} e^{-E(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2)}. \quad (18)$$

Finalmente, as probabilidades condicionais sobre a unidade visível e as duas unidades ocultas são formuladas pelas Equações 19, 20 e 21:

$$P(v_i = 1|\mathbf{h}^1) = \phi \left(\sum_{j=1}^{n^1} w_{ij}^1 h_j^1 \right), \quad (19)$$

$$P(h_z^2 = 1|\mathbf{h}^1) = \phi \left(\sum_{i=1}^{m^2} w_{iz}^2 h_i^1 \right), \quad (20)$$

e

$$P(h_j^1 = 1|\mathbf{v}, \mathbf{h}^2) = \phi \left(\sum_{i=1}^{m^1} w_{ij}^1 v_i + \sum_{z=1}^{n^2} w_{jz}^2 h_z^2 \right). \quad (21)$$

Após o aprendizado da primeira RBM pelo método da Divergência Contrastiva, o modelo generativo pode ser descrito pela Equação 22:

$$P(\mathbf{v}) = \sum_{\mathbf{h}^1} P(\mathbf{h}^1) P(\mathbf{v}|\mathbf{h}^1), \quad (22)$$

onde $P(\mathbf{h}^1) = \sum_{\mathbf{v}} P(\mathbf{h}^1, \mathbf{v})$. Ademais, o processo de aprendizado deve continuar para a segunda RBM, substituindo $P(\mathbf{h}^1)$ por $P(\mathbf{h}^2) = \sum_{\mathbf{h}^1} P(\mathbf{h}^1, \mathbf{h}^2)$. Em outras palavras, ao utilizar tal procedimento, os parâmetros de uma DBM são aprendidos a cada camada (SALAKHUT-DINOV; HINTON, 2012). O Algoritmo 2 é responsável por descrever o pseudo-código do treinamento de uma DBM.

ALGORITMO 2: Pseudo-código da DBM.

Entrada: Um conjunto de treinamento $\mathcal{X} = \{x^{(1)}, x^{(2)}, \dots, x^{(l)}\}$, número de camadas L , vetor de taxas de aprendizado $\vec{\eta}$, vetor de unidades ocultas $\vec{n}$, número de épocas T , vetor de fatores de penalidade (decaimento de peso) $\vec{\lambda}$ e vetor de momentos $\vec{\alpha}$.

Saída: Vetor de matrizes de pesos $\vec{W}$ e vetor de *biases* $\vec{a}$ e $\vec{b}$. Note que os vetores são todos de tamanho L .

```

1 para  $i$  de 1 até  $L$  faça
2   se  $(i = 1)$  então
3      $erro =$  erro do treinamento da primeira RBM;
4   se  $(i = L)$  então
5      $erro +=$  erro do treinamento da última RBM;
6   senão
7      $erro +=$  erro do treinamento das RBMs intermediárias;
8   Atualizar a camada oculta da RBM  $i$  como entrada da próxima RBM, ou seja, da
   camada  $i + 1$ ;

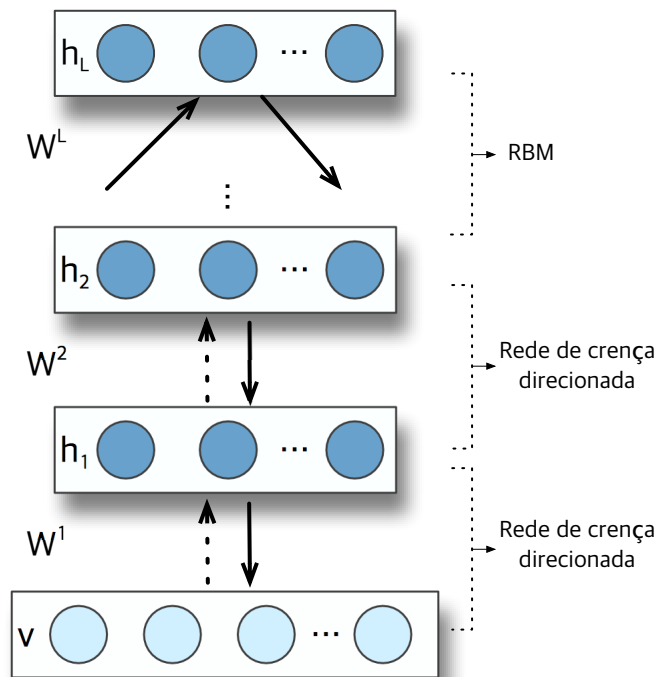
```

2.3 Redes de Crença em Profundidade

As Redes de Crença em Profundidade são, na verdade, compostas por uma série de RBMs empilhadas, sendo cada uma delas treinada de uma maneira “gulosa”, conforme o algoritmo apresentado na Seção 2.1.1. Isto significa que uma RBM de certa camada não considera outras durante o seu treinamento, contrário às DBMs, onde a formulação para se obter um neurônio da terceira camada escondida, por exemplo, considera todas as camadas inferiores.

A Figura 9 apresenta a arquitetura de uma DBN. Neste caso, temos uma DBN composta por L camadas, sendo $\mathbf{W}^i$ a matriz de pesos de uma RBM na camada i . Adicionalmente, podemos observar que as unidades ocultas da camada i tornam-se as unidades de entrada da camada $i + 1$. Embora não tenhamos ilustrado as unidades de *biases* para as camadas visíveis (entrada) e ocultas, também há unidades para cada camada. O Algoritmo 3 é responsável por descrever o pseudo-código do treinamento de uma DBN.

Figura 9 – Arquitetura padrão de uma DBN.



Fonte: Elaborado pelo autor.

A abordagem proposta por Hinton et al. (HINTON; OSINDERO; TEH, 2006) para a etapa de treinamento de uma DBN também considera uma etapa final de afinamento de parâmetros, a qual pode ser realizada utilizando o algoritmo de retropropagação ou gradiente descendente, visando ajustar as matrizes de peso $\mathbf{W}^i$, $i = 1, 2, \dots, L$. O algoritmo de otimização objetiva minimizar alguma medida de erro baseado na saída de uma *camada adicional* posicionada no topo da DBN. Tal camada é geralmente composta por unidades logísticas ou

do tipo *softmax*, ou até mesmo alguma técnica de classificação supervisionada de padrões. Zhou et al. (ZHOU; CHEN; WANG, 2010), por exemplo, apresentaram uma DBN Discriminativa para a etapa de classificação de imagens, sendo a camada mais superior empregada para codificar todos os rótulos possíveis (geralmente um vetor binário é utilizado para tal propósito). Naquele trabalho, uma função que calcula o erro de classificação é utilizada como critério de minimização.

ALGORITMO 3: Pseudo-código da DBN.

Entrada: Um conjunto de treinamento $\mathcal{X} = \{x^{(1)}, x^{(2)}, \dots, x^{(l)}\}$, número de camadas L , vetor de taxas de aprendizado $\vec{\eta}$, vetor de unidades ocultas $\vec{n}$, número de épocas T , vetor de fatores de penalidade (decaimento de peso) $\vec{\lambda}$ e vetor de momentos $\vec{\alpha}$.

Saída: Vetor de matrizes de pesos $\vec{W}$ e vetor de *biases* $\vec{a}$ e $\vec{b}$. Note que os vetores são todos de tamanho L .

```

1 para  $i$  de 1 até  $L$  faça
2    $erro =$  erro do treinamento da RBM (Algoritmo 1) da camada  $i$ ;
3   Atualizar a camada  $i$  como entrada da próxima RBM, ou seja, da camada  $i + 1$ ;

```

2.4 Redes Neurais Convolucionais

As Redes Neurais Convolucionais podem ser vistas como representação de uma classe maior de modelos baseados na arquitetura de Hubel e Wiesel, os quais realizaram um estudo seminal sobre o córtex primário de gatos em 1962 (HUBEL; WIESEL, 1962). Este trabalho identificou, basicamente, dois tipos de células: (i) células simples, as quais possuem uma tarefa análoga à etapa de filtragem por banco de máscaras, e (ii) células complexas, as quais realizam uma tarefa semelhante à etapa de amostragem das CNNs.

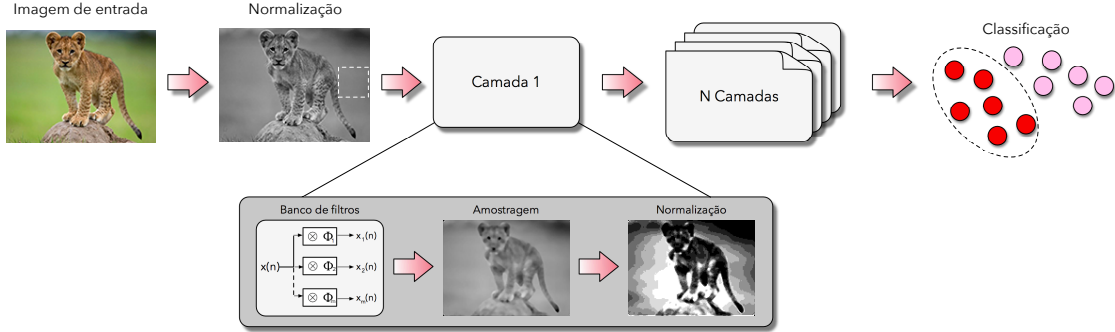
O primeiro modelo que simulou uma Rede Neural por Convolução foi o amplamente conhecido “Neocognitron” (FUKUSHIMA; MIYAKE, 1982), o qual aplicava um algoritmo de treinamento não supervisionado na etapa de filtragem por banco de máscaras, seguido por um algoritmo de treinamento supervisionado na última camada da rede. Posteriormente, LeCun et al. (LECUN et al., 1989; LECUN et al., 1990) simplificaram esta arquitetura propondo a utilização do algoritmo de retropropagação para o treinamento da rede de maneira supervisionada.

Basicamente, uma CNN pode ser entendida como uma sequência de N camadas de processamento de imagem. Dada uma imagem de entrada⁹, uma CNN essencialmente extrai uma representação de alto nível da mesma, chamada de *imagem multibandas*, cujos atributos dos *pixels* são concatenados em um vetor de características para posterior aplicação de técnicas

⁹ A mesma abordagem pode ser estendida para aplicações baseadas em processamento de sinais.

de reconhecimento de padrões. A Figura 10 apresenta a arquitetura típica de uma Rede Neural Convolutional.

Figura 10 – Arquitetura típica de uma Rede Neural Convolutional.



Fonte: Elaborado pelo autor.

Geralmente, cada camada da CNN é composta por três operações, sendo a primeira delas uma convolução com um banco de filtros, seguida por uma fase de amostragem e finalmente, uma etapa de normalização. Conforme pode ser observado na Figura 10, existe ainda a possibilidade de ser realizada uma operação de normalização no início de todo o processo. Segue abaixo uma breve descrição de cada uma destas etapas:

- a) **Banco de Filtros:** Seja uma imagem multibandas $\hat{I} = (D_I, \vec{I})$ tal que $D_I \in n \times n$ seja o domínio da imagem, e $\vec{I} = \{I_1(p), I_2(p), \dots, I_m(p)\}$ corresponda ao vetor de atributos de um *pixel* $p = (x_p, y_p) \in D_I$, sendo m o número de bandas. Quando $\hat{I}$ é uma imagem tons de cinza, por exemplo, temos então que $m = 1$ e $\hat{I} = (D_I, I)$. Seja $\phi = (A, W)$ um filtro com pesos $W(q)$ associado a cada *pixel* $q \in A(p)$, onde $A(p)$ denota uma máscara de tamanho $L_A \times L_A$ centrada em p , e $q \in A(p)$ se, e somente se, $\max\{|x_q - x_p|, |y_q - y_p|\} \leq (L_A - 1)/2$. No caso de filtros multibandas, seus pesos podem ser representados como vetores $\vec{W}_i(q) = \{w_{i,1}(q), w_{i,2}(q), \dots, w_{i,m}(q)\}$ para cada filtro i e um banco multibanda de filtros pode ser então definido como $\phi = \{\phi_1, \phi_2, \dots, \phi_n\}$, onde $\phi_i = (A, \vec{W}_i)$, $i = \{1, 2, \dots, n\}$.

Assim, a convolução entre uma imagem de entrada $\hat{I}$ e um filtro ϕ_i produz a banda i da imagem filtrada $\hat{J} = (D_J, \vec{J})$, onde $D_J \in D_I$ e $\vec{J} = \{J_1(p), J_2(p), \dots, J_n(p)\}$, $\forall p \in D_J$:

$$J_i(p) = \sum_{\forall q \in A(p)} \vec{I}(q) \otimes \vec{W}_i(q), \quad (23)$$

onde $\otimes$ denota o operador de convolução. Os pesos de ϕ_i são usualmente gerados a partir de uma distribuição uniforme, i.e., $\mathcal{U}(0, 1)$, e posteriormente normalizados com média zero e norma unitária para garantir o seu espalhamento dentro da esfera unitária.

- b) **Amostragem:** Esta operação é de fundamental importância para uma CNN, a qual objetiva proporcionar invariância translacional para as características extraídas. Seja $B(p)$ as regiões de amostragem de tamanho $L_B \times L_B$ centrada em p . Adicionalmente, seja $D_K = D_I/s$ uma operação de amostragem regular a cada s *pixels*. Assim, a operação de amostragem resultante na imagem $\hat{K} = (D_K, \vec{K})$ é definida como segue:

$$K_i(p) = \sqrt[\alpha]{\sum_{\forall q \in B(p)} J_i(q)^\alpha}, \quad (24)$$

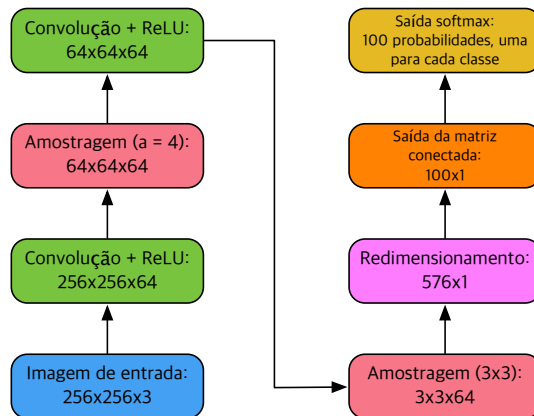
onde $p \in D_K$ denota cada *pixel* da nova imagem, $i = \{1, 2, \dots, n^2\}$ corresponde às bandas da imagem, e α corresponde ao parâmetro de passo (*stride*), controlando o fator de amostragem da operação.

- c) **Normalização:** A última operação de uma CNN é a sua normalização, a qual é amplamente utilizada para melhorar a performance desse tipo de arquitetura de aprendizado em profundidade (COX; PINTO, 2011). Esta operação é baseada nos mecanismos encontrados em neurônios corticais (GEISLER; ALBRECHT, 1992), sendo também definida sob uma região quadrada $C(p)$ de tamanho $L_C \times L_C$ centralizada no *pixel* p , tal que:

$$O_i(p) = \frac{K_i(p)}{\sum_{j=1}^n \sum_{\forall q \in C(p)} K_j(q) K_i(q)}. \quad (25)$$

Assim, a operação acima é realizada para cada *pixel* $p \in D_O \subset D_k$ da imagem resultante $\hat{O} = (D_O, \vec{O})$. A Figura 11 é responsável por ilustrar uma pseudo-arquitetura de uma CNN.

Figura 11 – Camadas de uma pseudo-arquitetura de uma CNN.



Fonte: Elaborado pelo autor.

3 Técnicas de Regularização

Um dos principais problemas do aprendizado de máquina consiste em modelar um algoritmo que não seja somente efetivo nos dados de treinamento, mas também em novos dados, não vistos anteriormente. Esta capacidade de desempenhar bem em dados não vistos é chamada de generalização. Basicamente, quando treinamos um modelo de aprendizado de máquina, podemos otimizar o treinamento minimizando uma determinada métrica de erro, chamada de erro de treinamento. Embora isto pareça um simples problema de otimização, a diferença consiste na necessidade de também obter um baixo erro de generalização (erro de teste), ou seja, o erro esperado quando confrontado com novos dados. Em outras palavras, o erro de generalização é estimado a partir da avaliação da performance de um conjunto de teste, o qual possui dados distintos do conjunto de treinamento.

Geralmente, o erro de teste é maior do que o erro de treinamento, dado que amostramos o conjunto de treinamento e escolhemos parâmetros para otimizá-lo e então, amostramos o conjunto de teste. Portanto, dois principais fatores que determinam a qualidade de um algoritmo de aprendizado de máquina são:

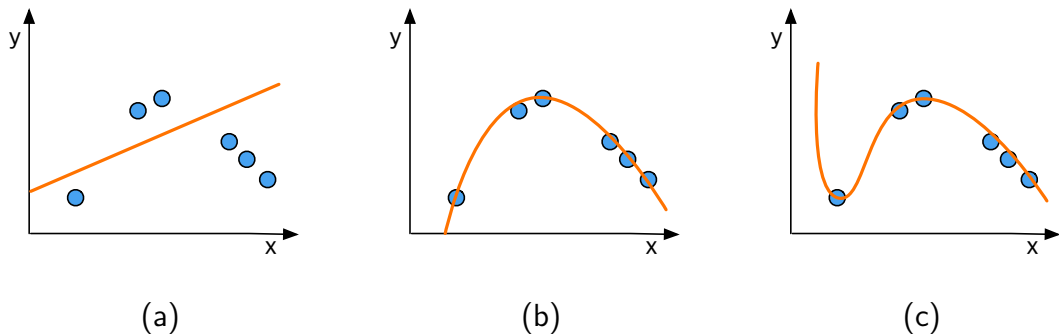
- a) Gerar um baixo erro de treinamento;
- b) Gerar a diferença mínima possível entre o erro de treino e teste.

Estes dois fatores correspondem aos dois principais desafios do aprendizado de máquina, os chamados sub-treinamento (*underfitting*) e super-treinamento (*overfitting*). O sub-treinamento ocorre quando o modelo não obtém um valor de erro suficientemente baixo no conjunto de treinamento, ou seja, o modelo não tem capacidade para distinguir os dados de treino. Já o super-treinamento ocorre quando o modelo possui uma alta diferença entre o erro de treino e teste, ou seja, o modelo acaba por “memorizar” os dados de treinamento, perdendo a sua capacidade de generalização e não desempenhando bem quando confrontado com novos dados. A Figura 12 é responsável por ilustrar exemplos destes problemas.

Diversas estratégias são investigadas para tentar reduzir estes problemas, comumente conhecidas por regularização. Regularização é qualquer modificação feita no algoritmo de treinamento com o objetivo de diminuir o erro de generalização, mas não o erro de treinamento. Atualmente, ela é tida como uma das principais questões estudadas pela área de aprendizado de máquina, junto com a otimização.

Por exemplo, Nowlan et. al introduziram um método de regularização denominado *soft-weight sharing* (NOWLAN; HINTON, 1992), o qual consiste em um termo de penalização ao longo do processo de treinamento. Outra técnica consiste na regularização L_1 ou *Lasso* (TIBSHIRANI, 1996), mais conhecida por menor erro absoluto. Também há a regula-

Figura 12 – Exemplo de (a) sub-treinamento, (b) treinamento adequado e (c) super-treinamento.



Fonte: Elaborado pelo autor.

rização L_2 ou *Ridge* (HOERL; KENNARD, 1970), conhecida como menor erro quadrático. Basicamente, a L_1 e L_2 podem ser definidas como normas, as quais são aplicadas como fator de penalidade em cima da matriz de pesos ao longo do processo de treinamento.

Como citado na Seção 1.1, a melhor alternativa para um técnica de regularização seria utilizar uma média das predições de todas as possíveis configurações de parâmetros, escolhendo a melhor alternativa. Entretanto, este processo é computacionalmente inviável para modelos complexos, não sendo tão efetivo nos problemas atuais. Por outro lado, uma possível abordagem seria a combinação de diferentes arquiteturas, auxiliando na redução do ruído obtido dos dados de treinamento. Todavia, treinar diferentes arquiteturas é um processo custoso, exigindo consideráveis quantias de dados de treinamento. Recentemente, Srivastava et al. (SRIVASTAVA et al., 2014) propuseram uma técnica denominada *Dropout*, a qual visa “desligar” neurônios aleatoriamente de acordo com um limiar durante o processo de convergência. Outra iniciativa diz respeito à técnica *Dropconnect* (WAN et al., 2013), a qual desliga conexões ao invés de neurônios. Nas próximas seções estas técnicas serão descritas em maiores detalhes.

3.1 *Dropout*

A técnica de *Dropout* é um forte método de regularização, proporcionando um baixo custo computacional. Inicialmente, podemos considerar o *Dropout* como uma forma de avaliar inúmeras sub-arquiteturas de redes a partir do “desligamento” aleatório de neurônios ao longo do processo de treinamento, prevenindo com que as unidades acomodem-se entre si e “memorizem” os dados ao invés de aprendê-los.

Considerando os conceitos apresentados acima, é direta a formulação matemática que embasa a teoria do *Dropout*. Inicialmente, seja $\vec{r}$ o vetor de ativação ou desligamento de M neurônios em uma camada específica da rede, onde cada variável r_j , $j = \{1, 2, \dots, M\}$, contém o valor 1 (um) com probabilidade $1 - p$, independente de outras variáveis r_i , $i =$

$\{1, 2, \dots, M\}$ e $i \neq j$. Caso r_j seja igual à 1 (um), a unidade h_j é retida, caso contrário, ela é desligada temporariamente da rede junto de suas conexões.

Note que a probabilidade p é um valor entre 0 e 1 e independente de outras unidades, onde $\vec{r}$ é amostrado diretamente de uma distribuição de Bernoulli (SRIVASTAVA et al., 2014), sendo re-amostrado para cada lote de dados durante o processo de treinamento. A Equação 26 descreve esta distribuição:

$$r_j \sim \text{Bernoulli}(p), \forall j = \{1, 2, \dots, M\}. \quad (26)$$

Portanto, durante a etapa de treinamento, seja a a função de ativação neural e $\vec{W}^{(L)}$ a matriz de pesos de uma determinada camada L . O vetor de ativações $\tilde{y}^{(L)}$, considerando o *Dropout*, é formulado pela Equação 27:

$$\tilde{y}^{(L)} = \vec{r} * a(\vec{W}^{(L)} \vec{v}^{(L)}). \quad (27)$$

onde $*$ é o operador de produto interno e $\vec{v}^{(L)}$ os dados de entrada da camada L .

É possível observar que a técnica de *Dropout* fornece um treinamento baseado em todas as possíveis 2^M sub-redes, dado que o processo de desligamento de neurônios é aleatório e apenas baseado na probabilidade p . Entretanto, ao final do treinamento, é necessário redimensionar a matriz de pesos $\vec{W}$ com probabilidade p a fim de ponderar todas estas possíveis sub-redes. A Equação 28 é responsável por descrever este processo.

$$\vec{W}^{(L)} = p \vec{W}^{(L)}. \quad (28)$$

Esta é a maior contribuição da regularização por *Dropout*, dado que podemos utilizar técnicas não-supervisionadas, por exemplo RBMs, com o intuito de efetuar um pré-treinamento dos dados. A partir disso, pode-se fornecer para outra técnica de aprendizado de máquina uma matriz de pesos mais robusta ao invés de uma inicialização aleatória dos pesos.

3.2 *Dropconnect*

Recentemente, uma generalização do *Dropout* foi proposta por Wan et. al, denominada *Dropconnect* (WAN et al., 2013). A ideia por trás da técnica do *Dropconnect* é retirar aleatoriamente conexões (pesos) ao invés de neurônios, obrigando as unidades à receberem um sub-conjunto de entradas das unidades da camada anterior.

Assim como o *Dropout*, o método *Dropconnect* introduz uma esparsidade dinâmica ao modelo, porém, esta esparsidade é introduzida diretamente na matriz de pesos $\vec{W}$ ao invés do vetor de saída da camada em questão. Em outras palavras, a camada da rede contendo

a regularização *Dropconnect* torna-se uma camada esparsamente conectada, com conexões aleatórias sendo escolhidas à cada iteração do treinamento.

Considerando os conceitos apresentados acima, é direta a formulação matemática que embasa a teoria do *Dropconnect*. Inicialmente, seja $\vec{R}$ a matriz de ativação ou desligamento de $M \times N$ conexões em uma camada específica da rede, onde cada variável R_{ij} , $i = \{1, 2, \dots, M\}$ e $j = \{1, 2, \dots, N\}$, contém o valor 1 (um) com probabilidade $1 - p$, independente de outras variáveis R_{kl} , $k = \{1, 2, \dots, M\}$, $l = \{1, 2, \dots, N\}$ e $ij \neq kl$. Caso R_{ij} seja igual à 1 (um), a conexão w_{ij} é retida, caso contrário, ela é desligada temporariamente da rede.

Note que a probabilidade p é um valor entre 0 e 1 e independente de outras unidades, onde $\vec{R}$ é amostrado diretamente de uma distribuição de Bernoulli (SRIVASTAVA et al., 2014), sendo re-amostrado para cada lote de dados durante o processo de treinamento. A Equação 29 descreve esta distribuição:

$$R_{ij} \sim \text{Bernoulli}(p), \forall i = \{1, 2, \dots, M\} \text{ e } \forall j = \{1, 2, \dots, N\}. \quad (29)$$

Portanto, durante a etapa de treinamento, seja a a função de ativação neural e $\vec{W}^{(L)}$ a matriz de pesos de uma determinada camada L . O vetor de ativações $\tilde{y}^{(L)}$ da camada L , considerando o *Dropconnect*, é formulado pela Equação 30:

$$\tilde{y}^{(L)} = a((\vec{R} * \vec{W}^{(L)})v^{(L)}), \quad (30)$$

onde $*$ é o operador de produto interno e $v^{(L)}$ os dados de entrada da camada L .

A técnica de *Dropconnect* utiliza, possivelmente, uma versão única da rede para cada lote de dados do treinamento, dado que as conexões da matriz de pesos foram desligadas aleatoriamente. Logo, para a etapa de teste é necessário levar em consideração este fato. Seja a solução correta representada por $\vec{o}^{(L)}$, o qual é o vetor de saídas de uma rede neural na camada L . A expectativa da resposta correta $E_{M \times N}$ requer uma soma sobre todas as possíveis $2^{M \times N}$ máscaras $\vec{R}$, podendo ser representada pela Equação 31:

$$\vec{o} = E_{M \times N}[\tilde{y}^{(L)}] = \sum_{M \times N} p(\vec{R})a((\vec{R} * \vec{W}^{(L)})v^{(L)}). \quad (31)$$

Pode-se observar que computar a expectativa real é inviável devido à quantidade de possíveis máscaras. Entretanto, como cada ativação de um neurônio é uma combinação linear da distribuição de Bernoulli, podemos aproximar esta função binomial para uma função normal (Gaussiana) através do Teorema do Limite Central. Portanto, a resposta da saída da camada L pode ser calculada através de integração numérica ou um simples procedimento de amostragem.

4 Otimização Meta-Heurística

Atualmente, há uma grande interconectividade entre diversas áreas do conhecimento, onde a matemática é tida como uma base formal para a representação e solução dos mais variados problemas. Economia (DIXIT, 1990), Geofísica (WILKINSON et al., 2012), Engenharia Elétrica (CHENG; BANDLER; KOZIEL, 2010), Engenharia Mecânica (RAO; SAVSANI; VAKHARIA, 2011), Engenharia Civil (GOLDBERG; KUO, 1987), dentre outras, são alguns exemplos de áreas que utilizam artifícios matemáticos com o intuito de solucionar seus desafios.

A necessidade de encontrar um conjunto melhor de informações (parâmetros) para solucionar determinada aplicação acabou por fomentar o estudo da programação matemática, mais conhecida por otimização. Um clássico exemplo de otimização é o problema do Caixeiro Viajante (PAPADIMITRIOU, 1977), o qual consiste em um objeto sair de uma localidade m e visitar outras $m - 1$ localidades, passando uma única vez por cada uma delas e por fim, retornando à localidade inicial. O objetivo final deste problema é encontrar uma rota de custo mínimo.

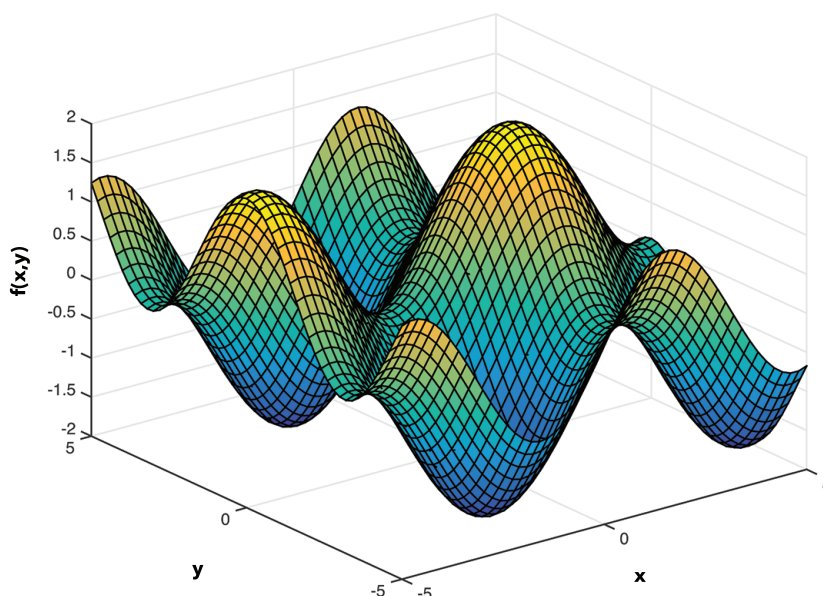
A partir do problema mencionado acima, podemos criar uma analogia com situações encontradas em nosso cotidiano, por exemplo, uma empresa de logística necessita encontrar rotas de custo mínimo para transportar sua carga, minimizando as suas despesas. Ademais, existem diversos problemas de otimização encontrados diariamente, tais como, modelagem de componentes industriais (RAO; RAO, 2009), pesquisa operacional (RARDIN, 1998), modelos econômicos de mercado (KONNO; YAMAZAKI, 1991), modelagem molecular (BARONE; COSSI; TOMASI, 1998), dentre outros.

Um problema de otimização consiste em maximizar ou minimizar uma função a partir da escolha sistemática de um conjunto de valores permitidos para o problema. Em outras palavras, o processo de otimização encontra os “melhores valores” de uma função objetivo, dado um domínio pré-definido. Seja $f(x)$ uma função de aptidão (objetivo) pertencente ao domínio $\mathbb{R}^n$, ou seja, composta por um vetor de variáveis reais $\vec{x}$, onde $\vec{x} = \{x_0, x_1, \dots, x_n\}$. A otimização desta função visa encontrar o conjunto de valores mais adequado para o vetor $\vec{x}$, dado por $\vec{x}^*$, onde $f(\vec{x}^*) \leq f(\vec{x})$ para problemas de minimização ou $f(\vec{x}^*) \geq f(\vec{x})$ para problemas de maximização.

Entretanto, à medida que são utilizadas funções de aptidão mais complexas, podemos encontrar diversos pontos de mínimo ou máximo, chamados de mínimos ou máximos locais. A solução ótima do problema deve ser dada pelo melhor ponto deste conjunto, ou seja, por um ponto de mínimo ou máximo global. A Figura 13 mostra a superfície de uma função objetivo que possui diversos ótimos locais.

Métodos tradicionais para a resolução de problemas de otimização, os chamados méto-

Figura 13 – Exemplo da função $f(x, y) = \sin(x) + \cos(y)$ com diversos mínimos e máximos locais.



Fonte: Elaborado pelo autor.

dos iterativos, utilizam a avaliação de hessianos, gradientes ou o próprio valor da função para encontrar a solução ótima. Embora esta metodologia auxilie no processo de convergência, tais avaliações também aumentam a complexidade computacional de cada iteração, sendo inviável para determinadas aplicações. Dentre os algoritmos mais utilizados nesta abordagem estão: método de Newton, método Quasi-Newton, gradiente descendente, métodos de interpolação, dentre outros (BERTSEKAS, 1999).

Recentemente, uma abordagem interessante denominada meta-heurística (YANG, 2011) vem sendo utilizada para auxiliar na solução de problemas de otimização. Uma meta-heurística consiste em um procedimento de alto nível projetado para buscar, gerar ou selecionar uma heurística, a qual provém uma solução suficientemente boa para o problema de otimização. Em outras palavras, uma meta-heurística é um processo generativo que combina conceitos de diversificação (*exploration*), utilizado para realizar buscas em grandes porções do espaço de busca, e intensificação (*exploitation*), utilizado para refinar uma solução promissora a partir de uma busca em sua vizinhança.

Basicamente, meta-heurísticas são estratégias que guiam o processo de busca à procura de soluções quase-ótimas. As técnicas baseadas em meta-heurísticas constituem-se em simples buscas locais e procedimentos complexos de aprendizado, geralmente inspirados por alguma estratégia ou comportamento biológico da natureza. Adicionalmente, elas não possuem domínios específicos, podendo ser aplicadas à qualquer problema, e também possuem mecanismos

que evitam o aprisionamento em regiões de mínimos ou máximos locais. Devido à inúmera quantia de algoritmos de otimização meta-heurística, foi necessária uma escolha seletiva dos mesmos para efetuar o projeto. Optamos por escolher, dentre eles, um dos algoritmos mais tradicionais e utilizados no contexto de otimização, a Otimização por Enxame de Partículas, bem como três técnicas desenvolvidas pelo professor Xin-She Yang, da Middlesex University, dado que o mesmo possui extenso currículo na área de computação bioinspirada e é atual colaborador de um projeto do nosso grupo de pesquisa.

Sigla	Meta-Heurística (Inglês)	Meta-Heurística (Português)
BA	<i>Bat Algorithm</i>	Algoritmo do Morcego
CS	<i>Cuckoo Search</i>	Busca do Cuckoo
FA	<i>Firefly Algorithm</i>	Algoritmo do Vagalume
PSO	<i>Particle Swarm Optimization</i>	Otimização por Enxame de Partículas

Tabela 1 – Utilização de meta-heurísticas e suas siglas.

4.1 Algoritmo do Morcego

Um tipo de sonar, chamado ecolocalização, é uma das características mais perceptíveis dos morcegos, sendo utilizado extensivamente por micro-morcegos a fim de detectar presas, evitar obstáculos e localizar pontos de descanso no escuro (YANG, 2010b). Estes morcegos emitem um pulso sonoro de alta frequência e escutam o eco que ressalta dos objetos ao seu redor. Embora cada pulso dure apenas milissegundos (8 à 10 ms), sua frequência é constante e varia no intervalo de 25kHz à 150 kHz, detectando a distância e orientação do alvo, bem como o tipo e a velocidade do mesmo.

Tal mecanismo de ecolocalização pode ser associado à uma função objetivo a ser otimizada, possibilitando a formulação de um novo algoritmo de otimização. O Algoritmo do Morcego, do inglês *Bat Algorithm* (BA), utiliza três princípios básicos idealizados:

- Todos os morcegos utilizam a ecolocalização para ter percepção de distância, sabendo diferenciar entre comida ou presas e obstáculos;
- Morcegos voam randomicamente com velocidade $\vec{v}$ na posição $\vec{x}$ com uma frequência fixa f_{min} , variando seu ruído A em busca de presas. Sua frequência de pulsos emitidos pode ser automaticamente ajustada, bem como a taxa de emissão de pulsos $r \in [0, 1]$, dependendo da proximidade do alvo;
- Embora o ruído possa variar de diversas maneiras, podemos assumi-lo como um valor fixo ou variando de um valor alto (positivo) A para uma constante mínima A_{min} .

Seja $\mathbf{X}_{M \times N}$ um bando de morcegos, onde N é o número de variáveis de decisão e M é o número de morcegos. Inicialmente, para cada morcego i , onde $i = 1, 2, \dots, M$, é assinalada

uma frequência f_i distribuída uniformemente entre $[f_{min}, f_{max}]$. À cada iteração, ajusta-se o novo valor de frequência para cada morcego, bem como sua velocidade e posição, sendo dadas pelas Equações 32, 33 e 34, respectivamente:

$$f_i = f_{min} + (f_{max} - f_{min})\beta, \quad (32)$$

$$\vec{v}_i^{(t)} = \vec{v}_i^{(t-1)} + (\vec{x}_i^{(t)} - \vec{x}^*)f_i, \quad (33)$$

$$\vec{x}_i^{(t)} = \vec{x}_i^{(t-1)} + \vec{v}_i^{(t)}, \quad (34)$$

onde β é um vetor randômico obtido através de uma distribuição uniforme¹⁰ e $\vec{x}^*$ é a melhor solução global, localizada após a comparação entre todos os M morcegos.

Finalmente, é efetuada uma busca local utilizando a melhor solução encontrada, gerando uma nova solução para cada morcego a partir de uma busca local randômica, dada pela Equação 35:

$$\vec{x}_{novo} = \vec{x}_{antigo} + \epsilon A, \quad (35)$$

onde $\epsilon \in [-1, 1]$ é um número randômico. Note que o Algoritmo 4 é responsável por descrever o pseudo-código do funcionamento do Algoritmo do Morcego.

4.2 Algoritmo do Vagalume

A luz cintilante dos vagalumes é produzida por um processo bioluminescente, sendo frequentemente única para cada espécie em particular. Duas funções de tais sinais luminosos dizem respeito à atração de parceiros (comunicação) e atração de possíveis presas (YANG, 2010a). Adicionalmente, também é utilizado como um mecanismo de alerta pelos vagalumes. A principal ideia por trás deste algoritmo é que a intensidade de luz I decresce à medida que uma distância r aumenta, sendo $I \propto 1/r^2$. Aliado à absorção de luz pelo ar, os vagalumes podem ser vistos em distâncias limitadas, permitindo a comunicação entre si.

Os sinais luminosos podem ser associados à uma função de aptidão a ser otimizada, de tal forma que seja possível formular um novo algoritmo de otimização. Basicamente, o Algoritmo do Vagalume, do inglês *Firefly Algorithm* (FA) segue três princípios idealizados:

- a) Todos os vagalumes são unissexos, sendo atraídos a qualquer outro vagalume independente do sexo;

¹⁰ Neste trabalho, $\mathcal{U}$ foi adotado como o símbolo da distribuição uniforme.

ALGORITMO 4: Pseudo-código do Algoritmo do Morcego.

Entrada: Número de morcegos M , quantidade de variáveis de decisão N , número máximo de iterações T , frequência mínima f_{min} e máxima f_{max} , ruído A e taxa de pulso r .

Saída: Vetor $\vec{g}$ de variáveis de decisão contendo os melhores valores para a função de aptidão do problema.

```

1 Função objetivo  $f(\vec{x})$ ,  $\vec{x} = (x_1, \dots, x_N)^T$ ;
2 Inicializar a população de morcegos  $\vec{x}_i, i = (1, 2, \dots, M)$ , e suas velocidades  $v_i$ ;
3 Definir a frequência de pulso  $f_i$  para cada  $x_i$ ;
4  $t \leftarrow 1$ ;
5 enquanto ( $t < T$ ) faça
6   para  $i$  de 1 até  $M$  faça
7     Gerar nova solução  $\vec{x}_i$  a partir do ajuste da frequência e atualização de
       velocidade e posição, de acordo com as Equações 32, 33 e 34;
8      $rand \leftarrow \mathcal{U}(0, 1)$ ;
9     se ( $rand > r$ ) então
10      Selecionar a melhor solução  $\vec{x}^*$ ;
11      Gerar uma solução local  $\vec{x}_i$  em torno de melhor solução selecionada;
12     se ( $rand < A$  &  $f(\vec{x}_i) < f(\vec{x}^*)$ ) então
13      Aceitar nova solução  $\vec{x}_i$ ;
14     Ordenar os morcegos e encontrar o atual melhor  $\vec{x}^*$ ;
15    $t \leftarrow t + 1$ ;
16  $\vec{g} \leftarrow \vec{x}^*$ ;
```

- b) Atratividade é proporcional à luminosidade, decrescendo à medida que a distância aumenta. Se não existe nenhum vagalume mais brilhante que outro, ele irá se mover aleatoriamente;
- c) A luminosidade de cada vagalume é afetada ou determinada pela superfície da função de aptidão. Para um problema de maximização, a luminosidade é diretamente proporcional ao valor da função de aptidão.

Seja $\mathbf{X}_{M \times N}$ um enxame de vagalumes, onde N é o número de variáveis de decisão e M é o número de vagalumes. Adicionalmente, cada vagalume i é associado à uma posição $\vec{x}_i \in \mathbb{R}^N$, $i = 1, 2, \dots, M$, sendo sua luminosidade determinada pelo valor da função de aptidão $f(\vec{x}_i)$ nesta posição.

A atratividade de um vagalume $\vec{x}_i$ em relação à outro vagalume $\vec{x}_j$, $\beta(\vec{x}_i, \vec{x}_j)$, varia conforme uma distância r_{ij} (distância Euclidiana) entre o vagalume i e o vagalume j , sendo formulada pela Equação 36:

$$\beta(\vec{x}_i, \vec{x}_j) = \beta_0 e^{-\gamma r_{ij}^2}, \quad (36)$$

onde β_0 é a atratividade inicial e γ o coeficiente de absorção de luz.

Cada vagalume na posição $\vec{x}_i$ e iteração t é atraído em direção à outro vagalume mais brilhante na posição $\vec{x}_j$ ($f(\vec{x}_j) > f(\vec{x}_i)$) pela Equação 37:

$$\vec{x}_i^{(t+1)} = \vec{x}_i^{(t)} + \beta(\vec{x}_i^{(t)}, \vec{x}_j^{(t)})(\vec{x}_j^{(t)} - \vec{x}_i^{(t)}) + \alpha \left(\mu \frac{1}{2} \right), \quad (37)$$

onde α é um fator de aleatoriedade que controla a magnitude da perturbação estocástica de $\vec{x}_i^{(t)}$ e $\mu \in \mathcal{U}(0, 1)$. Após todos os vagalumes na iteração t terem sido atualizados, o melhor vagalume $\vec{x}^*$ realizará um caminho aleatório controlado sobre o espaço de busca, descrito pela Equação 38:

$$\vec{x}^* = \vec{x}^* + \alpha \left(\psi \frac{1}{2} \right), \quad (38)$$

onde $\psi \in \mathcal{U}(0, 1)$. O Algoritmo 5 é responsável por descrever o pseudo-código do funcionamento do Algoritmo do Vagalume.

ALGORITMO 5: Pseudo-código do Algoritmo do Vagalume.

Entrada: Número de vagalumes M , quantidade de variáveis de decisão N , número máximo de iterações T , coeficiente de absorção de luz γ , atratividade inicial β_0 e fator de aleatoriedade α .

Saída: Vetor $\vec{g}$ de variáveis de decisão contendo os melhores valores para a função de aptidão do problema.

```

1 Função objetivo  $f(\vec{x})$ ,  $\vec{x} = (x_1, \dots, x_N)^T$ ;
2 Inicializar a população de vagalumes  $x_i, i = (1, 2, \dots, M)$ ;
3 enquanto ( $t < T$ ) faça
4   para  $i$  de 1 até  $M$  faça
5     para  $j$  de 1 até  $M$  faça
6        $\text{dist} \leftarrow$  distância euclidiana entre vagalumes  $i$  e  $j$ ;
7       se ( $f(\vec{x}_i) > f(\vec{x}_j)$ ) então
8         Gerar atratividade  $\beta$  de acordo com a Equação 36;
9         Mover o vagalume  $i$  em direção ao vagalume  $j$  de acordo com a
           Equação 37;
10    $t \leftarrow t + 1$ ;
11   Melhor vagalume  $\vec{x}^*$  anda aleatoriamente sobre o espaço a partir da Equação 38;
12  $\vec{g} \leftarrow \vec{x}^*$ ;

```

4.3 Busca do Cuckoo

Os cuckoos são pássaros deslumbrantes devido aos bonitos sons que conseguem produzir e, principalmente, devido à sua agressiva estratégia de reprodução (YANG; DEB, 2010). Basicamente, os cuckoos possuem uma reprodução parasita, depositando seus ovos em ninhos de outras aves (PAYNE, 2005). A partir disso, é possível gerar um extenso número de filhotes sem se preocupar com os devidos cuidados, dado que, geralmente, os filhotes de cuckoo nascem antes que os de outras espécies, eliminando seus pseudo-irmãos e tomando conta do ninho hospedeiro. Por outro lado, estudos recentes tem demonstrado que a busca de comida por animais ocorre de uma maneira quase randômica, assemelhando-se muito às características dos vãos de Lévy, do inglês *Lévy Flights* (BROWN; LIEBOVITCH; GLENDON, 2007; PAVLYUKEVICH, 2007).

A Busca do Cuckoo, do inglês *Cuckoo Search* (CS), obedece à três regras idealizadas:

- a) Cada cuckoo deposita um ovo por vez, escolhendo um ninho randômico qualquer;
- b) Os melhores ninhos com os melhores ovos (soluções) são carregados para as próximas gerações;
- c) O número de ninhos hospedeiros é fixo, sendo que o hospedeiro pode descobrir o ovo intruso com probabilidade $p_a \in [0, 1]$. Neste caso, o pássaro hospedeiro pode jogar fora o ovo ou abandonar o ninho, construindo um novo ninho em uma nova localização.

Seja $\mathbf{X}_{M \times N}$ um conjunto de ninhos hospedeiros, onde N é o número de variáveis de decisão e M é o número de ninhos hospedeiros. À cada iteração, uma nova solução $\vec{x}^{(t+1)}$ é gerada para um cuckoo i por meio de um vão de Lévy, dada pela Equação 39:

$$\vec{x}_i^{(t+1)} = \vec{x}_i^{(t)} + \alpha * Levy(\lambda), \quad (39)$$

onde $\alpha > 0$ é o tamanho do passo e $*$ é o operador de produto interno. Os vãos de Lévy efetuam uma busca randômica com valores obtidos através de uma distribuição de Lévy, dada pela Equação 40:

$$Levy \sim u = t^{-\lambda}, (1 < \lambda \leq 3), \quad (40)$$

a qual possui média infinita com uma variância infinita.

Caso esta nova solução seja melhor que a anterior, ela é substituída. Note que podemos trocar a probabilidade p_a por uma fração dos M ninhos a serem substituídos, abandonando-os e gerando novos ninhos em novas localizações por meio dos vãos de Lévy. O Algoritmo 6 é responsável por descrever o pseudo-código do funcionamento da Busca do Cuckoo.

ALGORITMO 6: Pseudo-código da Busca do Cuckoo.

Entrada: Número de ninhos hospedeiros M , quantidade de variáveis de decisão N , número máximo de iterações T , probabilidade de substituição dos piores ninhos p_a , tamanho do passo α e variável β para computar a distribuição de Lévy.

Saída: Vetor $\vec{g}$ de variáveis de decisão contendo os melhores valores para a função de aptidão do problema.

```

1 Função objetivo  $f(\vec{x})$ ,  $\vec{x} = (x_1, \dots, x_N)^T$ ;
2 Inicializar a população de ninhos hospedeiros  $x_i, i = (1, 2, \dots, M)$ ;
3  $t \leftarrow 1$ ;
4 enquanto ( $t < T$ ) faça
5     Escolher um ninho  $i$  aleatório;
6     Gerar uma nova solução  $\vec{x}_i$  a partir de um vôo randômico de Lévy, dada pela
        Equação 39;
7     Escolher um ninho  $j$  aleatório;
8     se ( $f(\vec{x}_i) < f(\vec{x}_j)$ ) então
9         Substituir o ninho  $j$  pela nova solução obtida anteriormente;
10    Ordenar os ninhos e encontrar o atual melhor  $\vec{x}^*$ ;
11    Abandonar uma fração  $p_a$  dos piores ninhos e substituir suas soluções por novas
        posições via vôos de Lévy, utilizando a Equação 40;
12    Ordenar novamente os ninhos e encontrar o atual melhor  $\vec{x}^*$ ;
13     $t \leftarrow t + 1$ ;
14  $\vec{g} \leftarrow \vec{x}^*$ ;

```

4.4 Otimização por Enxame de Partículas

A Otimização por Enxame de Partículas (*Particle Swarm Optimization* - PSO) é um algoritmo modelado em inteligência coletiva que encontra uma solução em um espaço de busca com base na dinâmica do comportamento social (KENNEDY et al., 2001). Cada possível solução do problema é modelada como uma partícula do enxame, a qual imita seu vizinho baseando-se em uma função de aptidão. Outras definições consideram o PSO como um algoritmo de pesquisa baseado em processos estocásticos e populacionais, no qual a aprendizagem do comportamento social permite a cada solução possível (partícula) voar dentro deste espaço (enxame) à procura de outras partículas que possuam melhores características e, assim, maximizar ou minimizar a função de aptidão.

Cada partícula tem uma memória para armazenar sua melhor solução local (mínimos ou máximos locais) e a melhor solução global (mínimos ou máximos globais). Levando-se em conta estas informações, cada partícula tem a capacidade de imitar as outras que proporcionam à ela melhores posições no enxame. Este processo simula a interação social entre um bando de aves à procura de comida, por exemplo. Este mecanismo sócio-recognitivo pode ser resumido em três grandes princípios (KENNEDY et al., 2001): (i) avaliação, (ii) comparação e (iii) imitação. Cada partícula pode avaliar outras dentro de sua vizinhança através de alguma função objetivo,

pode compará-la com seu próprio valor e, finalmente, pode decidir se é uma boa escolha imitá-la ou não. Seja $\vec{p} = (p_1, p_2, \dots, p_M)$ um conjunto de partículas que compõem o enxame, tal que $p_i = (\vec{x}_i, \vec{v}_i)$, $i \in \{1, 2, \dots, M\}$, onde $\vec{x}_i \in \mathbb{R}^N$ e $\vec{v}_i \in \mathbb{R}^N$ corresponda à posição e velocidade da partícula p_i , respectivamente. Além disso, para cada partícula, também é conhecida sua melhor solução (posição no enxame) local $\hat{\vec{x}}_i$, bem como a melhor solução (global) $\vec{g}$ de todo o enxame.

Após a definição do tamanho do enxame, ou seja, o número de partículas, cada uma delas é inicializada com valores aleatórios de velocidade e posição. Cada indivíduo é então avaliado com relação à alguma função de aptidão e seu máximo ou mínimo local atualizado. No final, o valor máximo ou mínimo global é atualizado com a partícula que alcançou a melhor posição no enxame. Este processo é repetido até que algum critério de convergência seja satisfeito. As Equações 41 e 42 são responsáveis por modelar a atualização da velocidade e posição da partícula p_i na iteração t :

$$\vec{v}_i^{(t+1)} = w\vec{v}_i^{(t)} + c_1r_1(\hat{\vec{x}}_i^{(t)} - \vec{x}_i^{(t)}) + c_2r_2(\vec{g} - \vec{x}_i^{(t)}) \quad (41)$$

e

$$\vec{x}_i^{(t+1)} = \vec{x}_i^{(t)} + \vec{v}_i^{(t+1)}. \quad (42)$$

onde w é a força de inércia que controla o poder de interação entre as partículas, e $r_1, r_2 \in [0, 1]$ são variáveis aleatórias que dão a ideia da estocasticidade ao método PSO. As constantes c_1 e c_2 são também utilizadas para guiar as partículas e são definidas como parâmetros de entrada para o algoritmo, o qual é descrito em pseudo-código pelo Algoritmo 7.

ALGORITMO 7: Pseudo-código da Otimização por Enxame de Partículas.

Entrada: Número de partículas M , quantidade de variáveis de decisão N , número máximo de iterações T , constantes c_1 e c_2 e peso de inércia w .

Saída: Vetor $\vec{g}$ de variáveis de decisão contendo os melhores valores para a função de aptidão do problema.

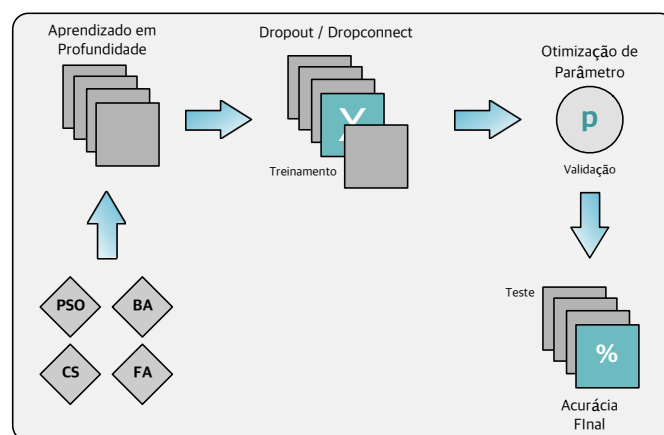
- 1 Função objetivo $f(\vec{x})$, $\vec{x} = (x_1, \dots, x_N)^T$;
 - 2 Inicializar a população de partículas $x_i, i = (1, 2, \dots, M)$;
 - 3 $t \leftarrow 1$;
 - 4 **enquanto** ($t < T$) **faça**
 - 5 **para** i **de** 1 **até** M **faça**
 - 6 Atualizar velocidade de cada partícula i de acordo com a Equação 41;
 - 7 Atualizar posição de cada partícula i de acordo com a Equação 42;
 - 8 $t \leftarrow t + 1$;
 - 9 $\vec{x}^* \leftarrow$ melhor partícula do enxame;
 - 10 $\vec{g} \leftarrow \vec{x}^*$;
-

5 Metodologia

Este capítulo é responsável por descrever a metodologia empregada na presente dissertação. Conforme descrito no Capítulo 2, existem diferentes arquiteturas de aprendizado em profundidade, cada uma possuindo características específicas. Entretanto, é possível afirmar que todas sofrem do mesmo problema relatado por esta dissertação, o sub-treinamento e super-treinamento dos dados. A proposta para solucionar tal problema é a utilização de métodos de regularização, que visam adicionar penalidades ou modificações ao longo do processo de treinamento destas arquiteturas, possibilitando a prevenção da co-adaptação de informações dos neurônios da rede, isto é, prevenindo com que um neurônio aprenda o erro de seu vizinho e, conseqüentemente, não consiga aprender qualquer outro tipo de informação.

Considerando estas informações, a presente dissertação procura solucionar tal problema por meio de duas técnicas recentes de regularização, *Dropout* e *Dropconnect*, descritas no Capítulo 3. Tais técnicas possuem uma formulação simples e podem ser adaptadas para todas as arquiteturas de aprendizado em profundidade. Não obstante, estas técnicas necessitam da escolha de um parâmetro, o qual indica a taxa de desligamento de neurônios/conexões da rede. Srivastava et. al (SRIVASTAVA et al., 2014) propuseram um parâmetro comum para todos os problemas ($p = 0,5$), porém, a literatura ainda necessita de um estudo maior do porquê da escolha deste parâmetro, dado que o mesmo foi escolhido de forma empírica em sua proposição original. Portanto, este trabalho procura solucionar tal problema a partir da escolha do parâmetro p via otimização meta-heurística, onde agentes (partículas) de cada técnica de otimização serão responsáveis por explorar e intensificar o espaço de busca à procura do melhor valor de p que minimiza a função de aptidão sobre um conjunto de validação. A Figura 14 ilustra um fluxograma da metodologia em questão.

Figura 14 – Fluxograma da metodologia utilizada.



Fonte: Elaborado pelo autor.

5.1 Bases de Dados

Esta seção descreve as bases de dados utilizadas para a validação da proposta desta dissertação, das quais, quatro de domínio público e duas obtidas por meio da colaboração do aluno em seu período de intercâmbio na *University of Virginia* (UVa).

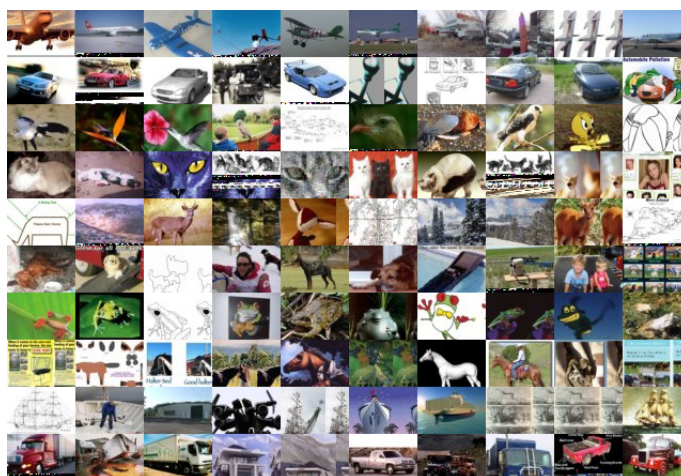
Base de dados	# Treinamento	# Validação	# Teste	# Resolução	# Classes
CIFAR-10	20.000 (100)	30.000 (100)	10.000 (100)	32×32	10
MNIST	20.000 (64)	40.000 (100)	10.000 (100)	28×28	10
Semeion	200 (2)	400 (4)	993 (993)	16×16	10
USPS	2.406 (32)	4.855 (977)	2.007 (2.007)	16×16	10
Brain-PET	53 (2)	53 (53)	158 (158)	182×218	2
Brain-MRI	53 (2)	53 (53)	158 (158)	182×218	3

Tabela 2 – Resumo de informações sobre as bases de dados.

5.1.1 CIFAR-10

A base de dados CIFAR-10¹¹ é um subconjunto de imagens da base de dados “80 million tiny images”, coletada por Alex Krizhevsky, Vinod Nair e Geoffrey Hinton. É composta por 60.000 imagens coloridas de tamanho 32×32 divididas em 10 classes, com 6.000 imagens por classe. Também é dividida em cinco blocos de treinamento e um bloco de teste, cada um contendo 10.000 imagens. Portanto, temos 50.000 imagens para a etapa de treinamento e 10.000 para a etapa de teste. A Figura 15 ilustra alguns exemplos desta base de dados.

Figura 15 – Exemplos de imagens de treinamento da base de dados CIFAR-10.



Fonte: Elaborado pelo autor.

¹¹ <http://www.cs.toronto.edu/~kriz/cifar.html>

5.1.2 MNIST

A base de dados MNIST¹² é composta por imagens de dígitos manuscritos. A versão original contém um conjunto de treinamento com 60.000 imagens de dígitos de '0' - '9', bem como um conjunto de teste com 10.000 imagens. As imagens contidas nesta base tem tamanho 28×28 *pixels* e estão em tons de cinza. A Figura 16 ilustra alguns exemplos desta base de dados.

Figura 16 – Exemplos de imagens de treinamento da base de dados MNIST.



Fonte: Elaborado pelo autor.

5.1.3 Semeion Handwritten Digits

A base de dados Semeion¹³ é um conjunto de 1.593 imagens de dígitos manuscritos de '0' - '9' escritos de duas formas: a primeira em um modo normal (com acurácia) e a segunda em um modo rápido (sem acurácia). No final do processo, elas foram redimensionadas para uma resolução de 16×16 *pixels* em tons de cinza, e então, cada *pixel* foi binarizado. A Figura 17 ilustra alguns exemplos desta base de dados.

5.1.4 USPS

A base de dados USPS¹⁴ possui dados numéricos obtidos a partir do escaneamento de dígitos manuscritos de envelopes do serviço postal dos Estados Unidos (*U.S. Postal Service*). É composta por 7.291 imagens de treinamento e um conjunto de teste de 2.007 imagens. Todos os dígitos originais foram redimensionados e normalizados, resultando em imagens de tons de cinza com tamanho de 16×16 *pixels*. A Figura 18 ilustra alguns exemplos desta base de dados.

¹² <http://yann.lecun.com/exdb/mnist/>

¹³ <https://archive.ics.uci.edu/ml/datasets/Semeion+Handwritten+Digit>

¹⁴ <http://statweb.stanford.edu/~tibs/ElemStatLearn/datasets/zip.info.txt>

Figura 17 – Exemplos de imagens de treinamento da base de dados Semeion.



Fonte: Elaborado pelo autor.

Figura 18 – Exemplos de imagens de treinamento da base de dados USPS.

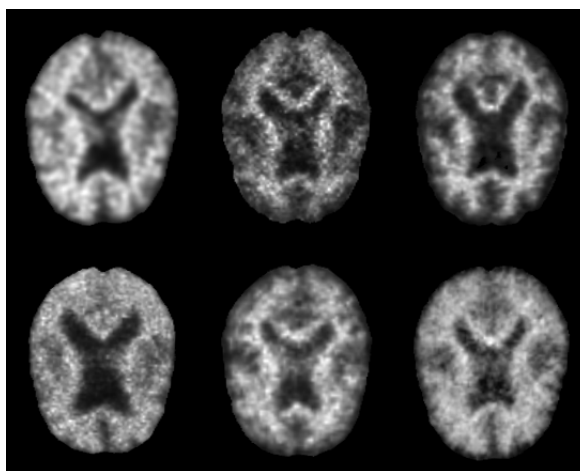


Fonte: Elaborado pelo autor.

5.1.5 Brain-PET

A base de dados Brain-PET constitui cortes 2-D de cérebros obtidos através de um exame de tomografia (*Positron Emission Tomography* - PET), sendo composta por 264 imagens tons de cinza de 50 indivíduos, em uma resolução de 182×218 *pixels*. Esta base de dados é resultado de um programa conjunto entre os departamentos de medicina e engenharia biomédica da UVa. Cada retalho do cérebro humano contém informações significantes da proteína amiloidose, sendo utilizada para discernir e identificar pacientes sadios de pacientes com o mal de Alzheimer (GHISO; FRANGIONE, 2002). Portanto, pode-se observar que esta base trata de uma classificação binária. A Figura 19 ilustra alguns exemplos desta base de dados.

Figura 19 – Exemplos de imagens de treinamento da base de dados Brain-PET.

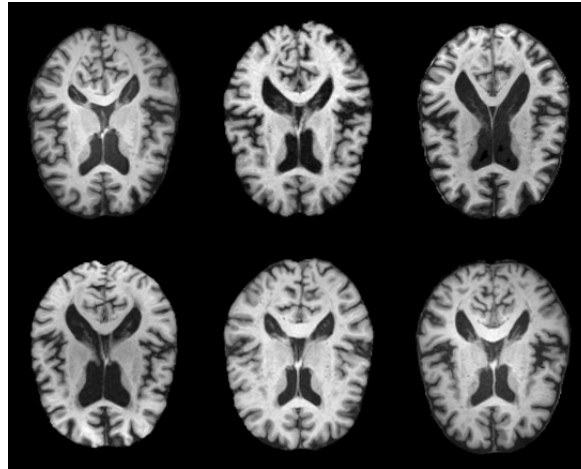


Fonte: Elaborado pelo autor.

5.1.6 Brain-MRI

A base de dados Brain-MRI constitui cortes 2-D de cérebros obtidos através de um exame de ressonância magnética (*Magnetic Resonance Imaging* - MRI), sendo composta por 264 imagens tons de cinza de 50 indivíduos (mesmos indivíduos da Brain-PET), em uma resolução de 182×218 *pixels*. Como mencionado anteriormente, esta base de dados é resultado de um programa conjunto entre os departamentos de medicina e engenharia biomédica da UVa. Entretanto, como foi obtida através de um exame de MRI, ela possibilita a identificação de pacientes que possuem algum tipo de deficiência cognitiva (*Mild Cognitive Impairment* - MCI), além daqueles que são sadios ou possuem mal de Alzheimer. Desta forma, é uma tarefa mais difícil que os exames PET, resultando em um problema de 3 classes. A Figura 20 ilustra alguns exemplos desta base de dados.

Figura 20 – Exemplos de imagens de treinamento da base de dados Brain-MRI.



Fonte: Elaborado pelo autor.

5.2 Configuração Experimental

Nesta seção descreve-se a configuração experimental da metodologia empregada para a realização dos experimentos desta dissertação. Note que a fim de organizar o texto de forma mais compreensiva, os experimentos foram separados em diferentes subseções, cada qual correspondendo à uma determinada categoria. Ademais, como a principal proposta deste trabalho é validar diferentes algoritmos de otimização meta-heurística no contexto de seleção de parâmetros de técnicas de regularização em diferentes arquiteturas de aprendizado em profundidade, foi estabelecido um padrão de parâmetros para cada um dos algoritmos, sendo o mesmo aplicado em todos os experimentos descritos a seguir. A Tabela 3 descreve a parametrização utilizada por cada meta-heurística.

Técnica	Parâmetros	# Agentes	# Iterações
BA	$f_{min} = 0, f_{max} = 2, A = 0, 5, r = 0, 5$	7	10
CS	$\beta = 1, 5, p = 0, 25, \alpha = 0, 8$	7	10
FA	$\gamma = 1, 0, \beta_0 = 1, 0, \alpha = 0, 2$	7	10
PSO	$c_1 = 1, 7, c_2 = 1, 7, w = 0, 7$	7	10

Tabela 3 – Configuração dos parâmetros das técnicas meta-heurísticas.

Com relação aos códigos-fontes utilizados nesta dissertação, utilizamos nossas próprias implementações dos algoritmos em questão, os quais estão disponíveis para livre acesso e uso na plataforma GitHub. A biblioteca LibOPF¹⁵ possibilita a utilização das bases de dados no formato .opf (padrão para os nossos experimentos), a biblioteca LibDEEP¹⁶ contém a implementação dos algoritmos de aprendizado em profundidade, a biblioteca LibOPT¹⁷ possui

¹⁵ <https://github.com/jppbsi/LibOPF>

¹⁶ <https://github.com/jppbsi/LibDEEP>

¹⁷ <https://github.com/jppbsi/LibOPT>

a implementação dos algoritmos de otimização meta-heurística e, finalmente, a biblioteca LibDEV¹⁸ faz a integração entre todas as bibliotecas, possibilitando a utilização da otimização em conjunto com o aprendizado em profundidade, dentre outras implementações.

Entretanto, dado que as CNNs possuem implementações mais eficientes quando utilizadas com placas gráficas (*Graphics Processor Unit* - GPU), utilizamos a conhecida biblioteca Caffe¹⁹ (JIA et al., 2014) para sua implementação, a qual é desenvolvida especificamente para o uso em GPUs. Ademais, para integrar a LibOPT com a Caffe, foi criada uma nova biblioteca chamada LibOPT4Caffe²⁰, também disponível para livre uso no GitHub.

5.2.1 Experimento A - RBMs

A proposta é modelar o problema da seleção de parâmetros de *Dropout* e *Dropconnect* em RBMs no contexto de reconstrução de imagens binárias. Como mencionado nas Seções 2.1 e 3.1, a etapa de treinamento possui cinco parâmetros: taxa de aprendizado (η), decaimento de peso (λ), parâmetro de penalização (α), número de unidades escondidas (n) e taxa de desligamento (p). Como estamos interessados em avaliar a otimização dos parâmetros das técnicas de regularização mencionadas nesta dissertação, a 4-tupla $(\eta, \lambda, \alpha, n)$ foi fixada, originando um problema de otimização de uma dimensão, o qual tem por objetivo minimizar o erro médio quadrático (*Mean Squared Error* - MSE) das imagens reconstruídas no conjunto de validação. Posteriormente, o conjunto de parâmetros encontrado é aplicado na reconstrução de novas imagens, contidas no conjunto de teste.

Cada parâmetro da RBM foi fixado como segue: $\eta = 0,1$, $\lambda = 0,0002$ e $\alpha = 0,5$. Com o intuito de avaliar a eficácia da regularização em diferentes tamanhos de arquitetura de RBMs, note que existem duas configurações para o seu tamanho, uma contendo 500 unidades escondidas (n) e outra contendo 2.000. Com relação aos parâmetros de *Dropout* e *Dropconnect*, os mesmos foram mantidos no intervalo $p \in [0, 1; 1, 0]$. Também empregamos $T = 25$ como o número de épocas para o procedimento de aprendizado de pesos das RBMs, com lotes (*batches*) de tamanho 20, e o algoritmo de CD-1 para o seu treinamento. A fim de obter uma análise estatística, as bases de dados sofreram um procedimento de validação cruzada *hold-out* de 10 iterações, dividindo-as em 10 conjuntos de treinamento, validação e teste, e então aplicado o teste de Wilcoxon (WILCOXON, 1945) no vetor resultante obtido dos conjuntos de teste, com significância igual a 0,05. A Tabela 4 descreve as informações relativas às bases de dados utilizadas no Experimento A.

¹⁸ <https://github.com/jppbsi/LibDEV>

¹⁹ <https://caffe.berkeleyvision.org>

²⁰ <https://github.com/gugarosa/LibOPT4Caffe>

Base de dados	# Treinamento	# Validação	# Teste	# Características	# Classes
MNIST	3.216	2.728	1.078	196	10
Semeion	473	320	800	256	10
USPS	4.371	2.920	2.007	256	10

Tabela 4 – Configuração das bases de dados para o Experimento A.

5.2.2 Experimento B - DBMs

A proposta é modelar o problema da seleção de parâmetros de *Dropout* e *Dropconnect* em DBMs no contexto de reconstrução de imagens binárias. Como mencionado na Seção 5.2.1, cada parâmetro da DBM foi fixado como segue: $\eta = 0,1$, $\lambda = 0,0002$ e $\alpha = 0,5$, independente da quantidade de camadas, ou seja, para uma DBM com duas camadas existirão dois $\eta = 0,1$, dois $\lambda = 0,0002$ e assim por diante. Contudo, com o intuito de avaliar a eficácia da regularização em diferentes tamanhos de arquitetura de DBMs, note que existem diferentes configurações para o seu número de unidades escondidas, descritas pela Tabela 5. Com relação aos parâmetros de *Dropout* e *Dropconnect*, os mesmos foram mantidos no intervalo $p \in [0,1;1,0]$, independente do número de camadas. Também empregamos $T = 25$ como o número de épocas para o procedimento de aprendizado de pesos das DBMs, com *batches* de tamanho 20, e o algoritmo de CD-1 para o seu treinamento. A fim de obter uma análise estatística, as bases de dados sofreram um procedimento de validação cruzada *hold-out* de 10 iterações, dividindo-as em 10 conjuntos de treinamento, validação e teste, e então aplicado o teste de Wilcoxon no vetor resultante obtido dos conjuntos de teste, com significância igual a 0,05. A Tabela 6 descreve as informações relativas às bases de dados utilizadas no Experimento B.

# Camadas	# Parâmetros	CONF #1	CONF #2
1	5	100	500
2	10	100 – 100	500 – 500
3	15	100 – 100 – 400	500 – 500 – 2.000

Tabela 5 – Configuração das arquiteturas para o Experimento B.

Base de dados	# Treinamento	# Validação	# Teste	# Características	# Classes
MNIST	3.216	2.728	1.078	196	10
Semeion	473	320	800	256	10
USPS	4.371	2.920	2.007	256	10

Tabela 6 – Configuração das bases de dados para o Experimento B.

5.2.3 Experimento C - DBNs

A proposta é modelar o problema da seleção de parâmetros de *Dropout* e *Dropconnect* em DBNs no contexto de reconstrução de imagens binárias. Como mencionado nas Seção 5.2.1, cada parâmetro da DBN foi fixado como segue: $\eta = 0,1$, $\lambda = 0,0002$ e $\alpha = 0,5$, independente da quantidade de camadas, ou seja, para uma DBM com duas camadas existirão dois $\eta = 0,1$, dois $\lambda = 0,0002$, e assim por diante. Contudo, com o intuito de avaliar a eficácia da regularização em diferentes tamanhos de arquitetura de DBNs, note que existem diferentes configurações para o seu número de unidades escondidas, descritas pela Tabela 7. Com relação aos parâmetros de *Dropout* e *Dropconnect*, os mesmos foram mantidos no intervalo $p \in [0,1;1,0]$, independente do número de camadas. Também empregamos $T = 25$ como o número de épocas para o procedimento de aprendizado de pesos das DBNs, com *batches* de tamanho 20, e o algoritmo de CD-1 para o seu treinamento. A fim de obter uma análise estatística, as bases de dados sofreram um procedimento de validação cruzada *hold-out* de 10 iterações, dividindo-as em 10 conjuntos de treinamento, validação e teste, e então aplicado o teste de Wilcoxon no vetor resultante obtido dos conjuntos de teste, com significância igual a 0,05. A Tabela 8 descreve as informações relativas às bases de dados utilizadas no Experimento C.

# Camadas	# Parâmetros	CONF #1	CONF #2
1	5	100	500
2	10	100 – 100	500 – 500
3	15	100 – 100 – 400	500 – 500 – 2.000

Tabela 7 – Configuração das arquiteturas para o Experimento C.

Base de dados	# Treinamento	# Validação	# Teste	# Características	# Classes
MNIST	3.216	2.728	1.078	196	10
Semeion	473	320	800	256	10
USPS	4.371	2.920	2.007	256	10

Tabela 8 – Configuração das bases de dados para o Experimento C.

5.2.4 Experimento D - CNNs

A proposta é modelar o problema da seleção de parâmetros de *Dropout* e *Dropconnect* em CNNs no contexto de classificação de imagens. Como mencionado nas Seções 2.4 e 3.1, a etapa de treinamento possui quatro parâmetros: taxa de aprendizado (η), decaimento de peso (λ), parâmetro de penalização (α) e taxa de desligamento (p). Como estamos interessados em avaliar a otimização dos parâmetros das técnicas de regularização mencionadas nesta dissertação, a 3-tupla (η, λ, α) foi fixada, originando um problema de otimização de uma dimensão, o qual tem por objetivo minimizar a função de perda das imagens classificadas no

conjunto de validação. Posteriormente, o conjunto de parâmetros encontrado é aplicado na classificação de novas imagens, contidas no conjunto de teste.

Adicionalmente, foram consideradas duas diferentes arquiteturas de CNN para uma melhor análise experimental: uma para as bases de dados CIFAR-10, Brain-PET e Brain-MRI, e outra para as bases MNIST, Semeion e USPS. Estas arquiteturas são as originais propostas pelos exemplos da Caffe, apenas com uma camada extra de *Dropout* ou *Dropconnect*. Entretanto, considerando as bases Brain-PET e Brain-MRI, note que foi necessário ajustar a última camada “ip” para o número de classes, sendo 2 para PET e 3 para MRI. Além disso, considerando as bases Semeion e USPS, utilizamos um tamanho de núcleo para as camadas de convolução igual a 3 ao invés de 5 devido à menor resolução das imagens destas bases. A Figura 21 ilustra as arquiteturas usadas nesta dissertação. Note que “conv” é a camada de convolução, “pool” é a de amostragem, “relu” são as unidades retificadoras lineares, “drop” é a camada de *Dropout* ou *Dropconnect* e, finalmente, “ip” é a camada de produto interno, ou seja, uma camada de multiplicação no espaço de vetores.

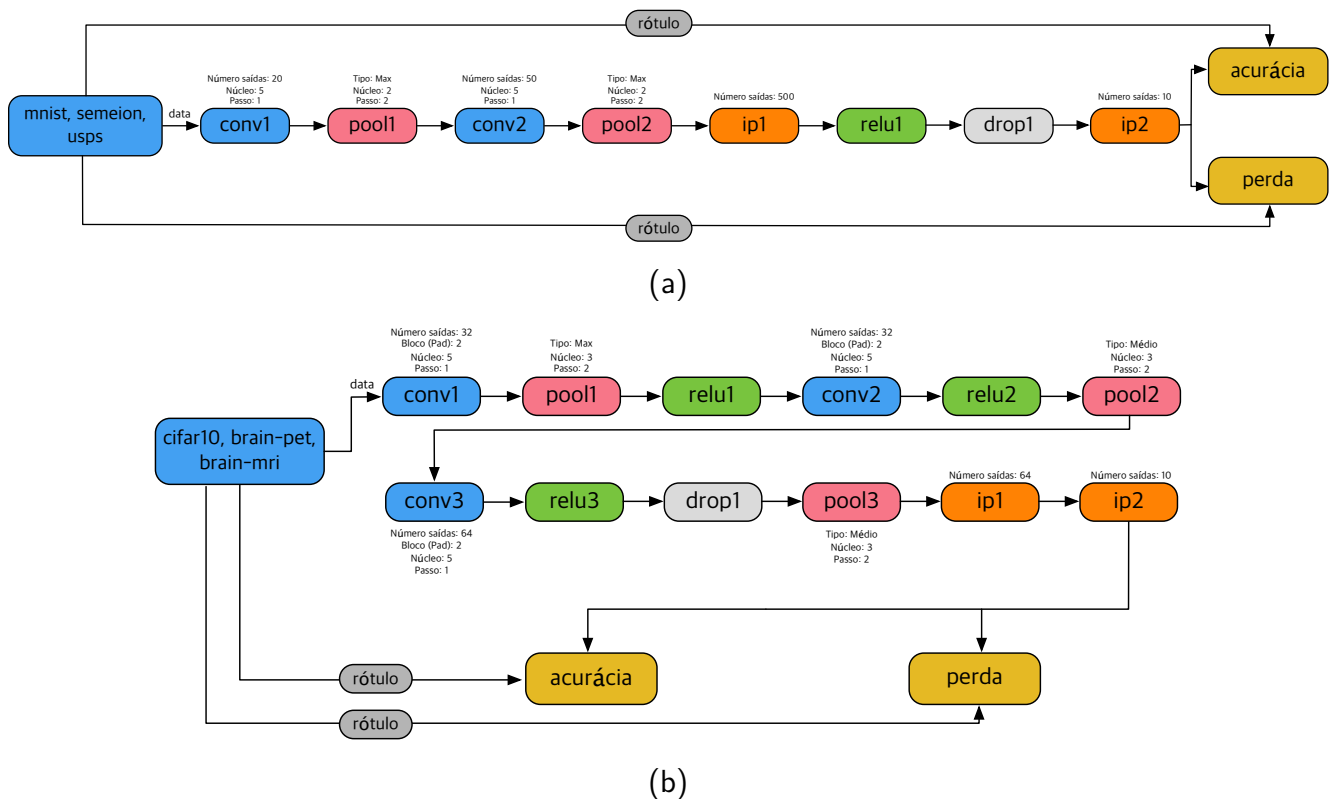


Figura 21 – Arquiteturas de CNNs para as bases de dados: (a) MNIST, Semeion e USPS, e (b) CIFAR-10, Brain-PET e Brain-MRI.

Fonte: Elaborado pelo autor.

Todas as bases de dados foram divididas em conjuntos de treinamento, validação e teste. A Tabela 2 descreve a quantidade de imagens para cada conjunto, bem como seu número de características e classes. Note que o tamanho do lote de imagens empregado está entre parênteses para cada conjunto. Finalmente, cada parâmetro da CNN (pré-definido pela Caffe) foi fixado de acordo com a Tabela 9. A fim de obter uma análise estatística, as bases de dados sofreram um procedimento de validação cruzada *hold-out* de 20 iterações, dividindo-as em 20 conjuntos de treinamento, validação e teste, e então aplicado o teste de Wilcoxon no vetor resultante obtido dos conjuntos de teste, com significância igual a 0,05.

Base de Dados	η	α	λ	p	# Iterações
CIFAR-10	0,001	0,9	0,004	[0; 1]	4.000
MNIST	0,01	0,9	0,0005	[0; 1]	10.000
Semeion	0,001	0,9	0,0005	[0; 1]	10.000
USPS	0,01	0,9	0,0005	[0; 1]	10.000
Brain-PET	0,0001	0,9	0,004	[0; 1]	500
Brain-MRI	0,0001	0,9	0,004	[0; 1]	500

Tabela 9 – Configuração dos parâmetros da CNN.

6 Experimentos e Resultados

Neste capítulo descrevem-se os resultados obtidos a partir dos experimentos efetuados. Note que os experimentos estão divididos em subseções correspondentes às apresentadas no Capítulo 5. Outra importante informação acerca das tabelas de resultados é o caractere * (asterisco), que indica a arquitetura da rede utilizada como resultado de *baseline*. Esta arquitetura não emprega nenhum processo de otimização ou camadas de *Dropout* e *Dropeconnect*.

6.1 Experimento A - RBMs

A base de dados MNIST é conhecida por sua facilidade em tarefas de aprendizado de máquina, dado que o reconhecimento de dígitos manuscritos é um processo relativamente fácil. Entretanto, nenhuma técnica ainda foi capaz de maximizar alguma métrica de avaliação nesta base.

A introdução de técnicas de regularização tem por objetivo auxiliar o processo de aprendizado e impedir que alguns vícios sejam carregados ao longo do mesmo. Pode parecer muito fácil introduzir novos processos com o intuito de melhorar o aprendizado, mas é preciso levar em conta o poder computacional para atingir tal objetivo. A ideia por trás de utilizar técnicas simplistas é não alterar um processo já complexo e sim, fornecer uma ferramenta extra à ele. Um fator a ser lembrado é que nem sempre as técnicas comportam-se como esperamos.

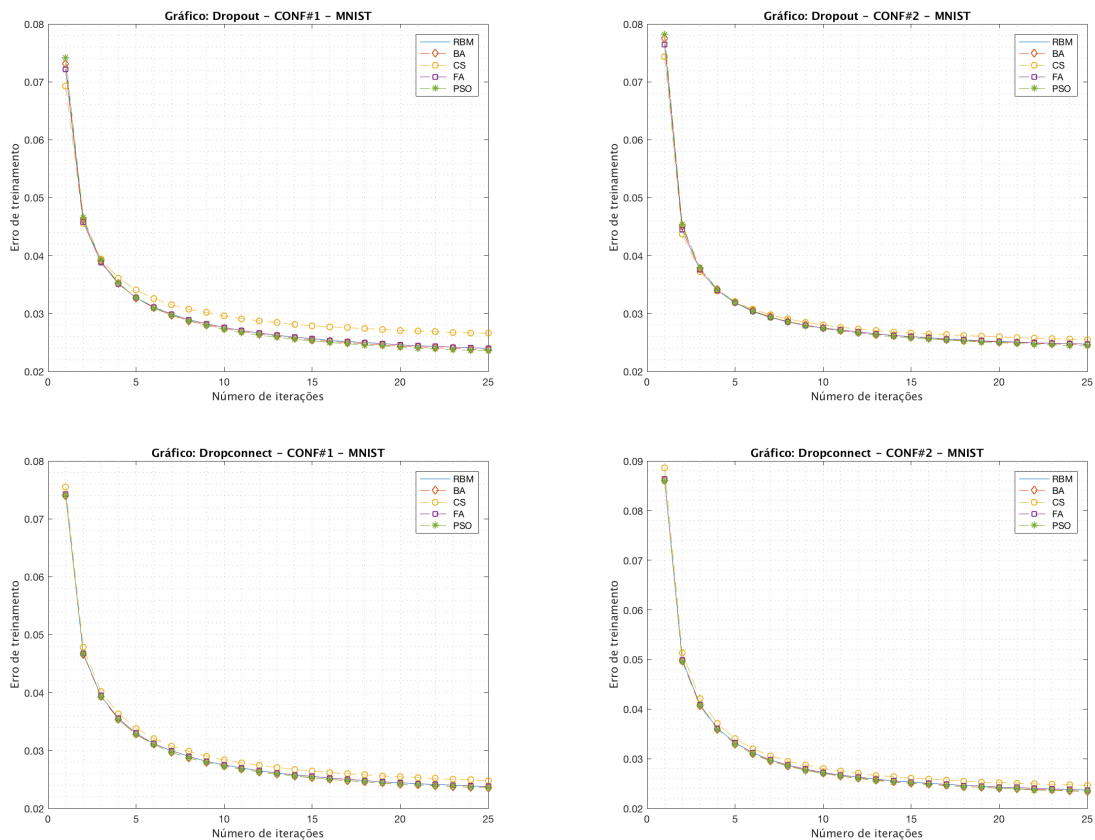
A Tabela 10 descreve os resultados obtidos pela RBM e técnicas meta-heurísticas na base de dados MNIST. A priori, é possível observar que todas as meta-heurísticas, exceto CS, obtiveram resultados estatisticamente iguais de acordo com o teste de Wilcoxon (células da tabela em negrito). Considerando a primeira configuração, a quantia de *Dropout* introduzida foi mínima, levando a crer que talvez o problema estivesse saturado para estas configurações. Considerando a segunda configuração, podemos observar que a quantia de *Dropout* introduzida foi maior devido à existência de um maior número de neurônios disponíveis. Outro fato interessante é o aumento do erro de treinamento devido à natureza das técnicas de regularização, onde uma sub-arquitetura da rede é utilizada a cada lote de treinamento, introduzindo uma quantia de ruído ao longo do processo. Em relação ao método *Dropeconnect*, não houve nenhuma melhora em sua utilização, precisando de uma investigação maior ou uma refatoração do algoritmo para o caso apresentado. A Figura 22 ilustra os gráficos desta tabela.

A Tabela 11 descreve os resultados obtidos na base de dados Semeion. Novamente, é possível observar que a utilização do *Dropeconnect* não trouxe nenhuma melhora ao processo. Em relação ao *Dropout*, pode-se analisar que o mesmo trouxe uma pequena melhora no aprendizado, principalmente na segunda configuração, onde a existência de um maior número

MNIST				
	CONF #1		CONF #2	
	Dropout	Dropconnect	Dropout	Dropconnect
RBM*	0,0145(0,0235) $\pm$ 0,0002[—]	0,0145(0,0235) $\pm$ 0,0002[—]	0,0146(0,0234) $\pm$ 0,0002[—]	0,0146(0,0234) $\pm$ 0,0002[—]
BA	0,0145(0,0237) $\pm$ 0,0002[0,0098]	0,0145(0,0236) $\pm$ 0,0003[0,0000]	0,0148(0,0246) $\pm$ 0,0004[0,0887]	0,0144(0,0234) $\pm$ 0,0003[0,0000]
CS	0,0161(0,0266) $\pm$ 0,0027[0,1204]	0,0154(0,0248) $\pm$ 0,0008[0,0001]	0,0159(0,0255) $\pm$ 0,0016[0,1559]	0,0153(0,0246) $\pm$ 0,0009[0,0001]
FA	0,0146(0,0240) $\pm$ 0,0003[0,0184]	0,0147(0,0238) $\pm$ 0,0004[0,0000]	0,0145(0,0247) $\pm$ 0,0004[0,0956]	0,0150(0,0237) $\pm$ 0,0003[0,0000]
PSO	0,0145(0,0235) $\pm$ 0,0002[0,0000]	0,0145(0,0235) $\pm$ 0,0002[0,0000]	0,0145(0,0244) $\pm$ 0,0003[0,0749]	0,0148(0,0234) $\pm$ 0,0004[0,0000]

Tabela 10 – Erro médio de teste (erro médio de treinamento) $\pm$ desvio padrão do erro de teste [parâmetro médio p], considerando RBMs sem regularização e otimizadas com *Dropout* e *Dropconnect* na base de dados MNIST.

Figura 22 – Gráficos: número de iterações $\times$ erro de treinamento para a Tabela 10.



Fonte: Elaborado pelo autor.

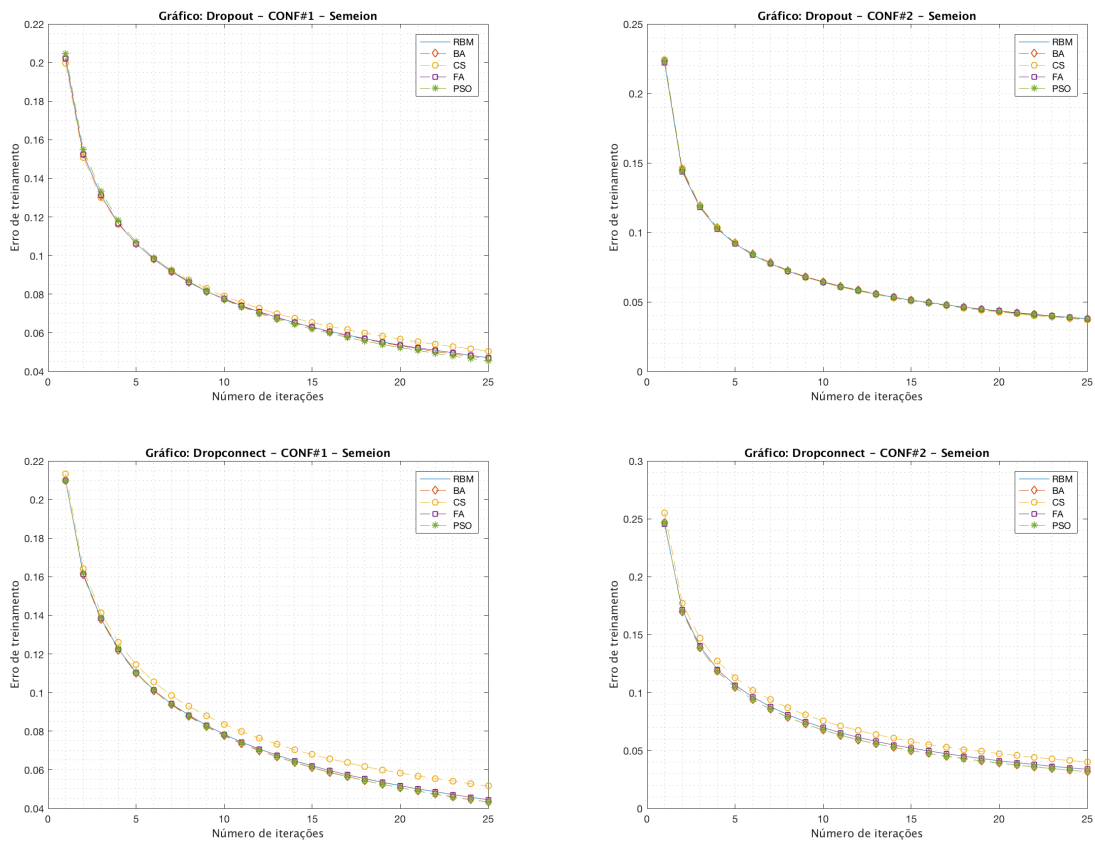
de neurônios pode acarretar problemas de super-treinamento dos dados. Nesta configuração, de acordo com Wilcoxon, os resultados obtidos pelas meta-heurísticas BA, FA e PSO conseguiram um resultado superior à rede sem *Dropout*. Também podemos observar que novamente o erro de treinamento foi superior ao da rede sem regularização e que a quantia de *Dropout* introduzida foi maior do que a primeira configuração, reforçando a discussão apresentada no experimento anterior. A Figura 23 ilustra os gráficos desta tabela.

Finalmente, a última base de dados analisada é a USPS, descrita pela Tabela 12. Assim como a base de dados MNIST, a USPS também é um problema mais simples e fácil de ser executado. Entretanto, as técnicas de *Dropout* e *Dropconnect* não surtiram tanto efeito neste

Semeion				
	CONF #1		CONF #2	
	Dropout	Dropconnect	Dropout	Dropconnect
RBM*	0,0548(0,0431) $\pm$ 0,0015[–]	0,0548(0,0431) $\pm$ 0,0015[–]	0,0460 (0,0316) $\pm$ 0,0008 [–]	0,0460(0,0316) $\pm$ 0,0008[–]
BA	0,0548(0,0466) $\pm$ 0,0007[0,0807]	0,0550(0,0433) $\pm$ 0,0014[0,0000]	0,0442(0,0377) $\pm$ 0,0009[0,2320]	0,0458(0,0317) $\pm$ 0,0008[0,0000]
CS	0,0562 (0,0505) $\pm$ 0,0019 [0,1498]	0,0599(0,0516) $\pm$ 0,0068[0,0001]	0,0443 (0,0369) $\pm$ 0,0009 [0,2174]	0,0542 (0,0400) $\pm$ 0,0097 [0,0002]
FA	0,0549(0,0471) $\pm$ 0,0008[0,0846]	0,0557(0,0444) $\pm$ 0,0020[0,0000]	0,0440(0,0379) $\pm$ 0,0009[0,2447]	0,0475 (0,0337) $\pm$ 0,0048 [0,0000]
PSO	0,0550(0,0454) $\pm$ 0,0008[0,0545]	0,0550(0,0433) $\pm$ 0,0013[0,0000]	0,0440(0,0375) $\pm$ 0,0010[0,2341]	0,0456(0,0318) $\pm$ 0,0008[0,0000]

Tabela 11 – Erro médio de teste (erro médio de treinamento) $\pm$ desvio padrão do erro de teste [parâmetro médio p], considerando RBMs sem regularização e otimizadas com *Dropout* e *Dropconnect* na base de dados Semeion.

Figura 23 – Gráficos: número de iterações $\times$ erro de treinamento para a Tabela 11.



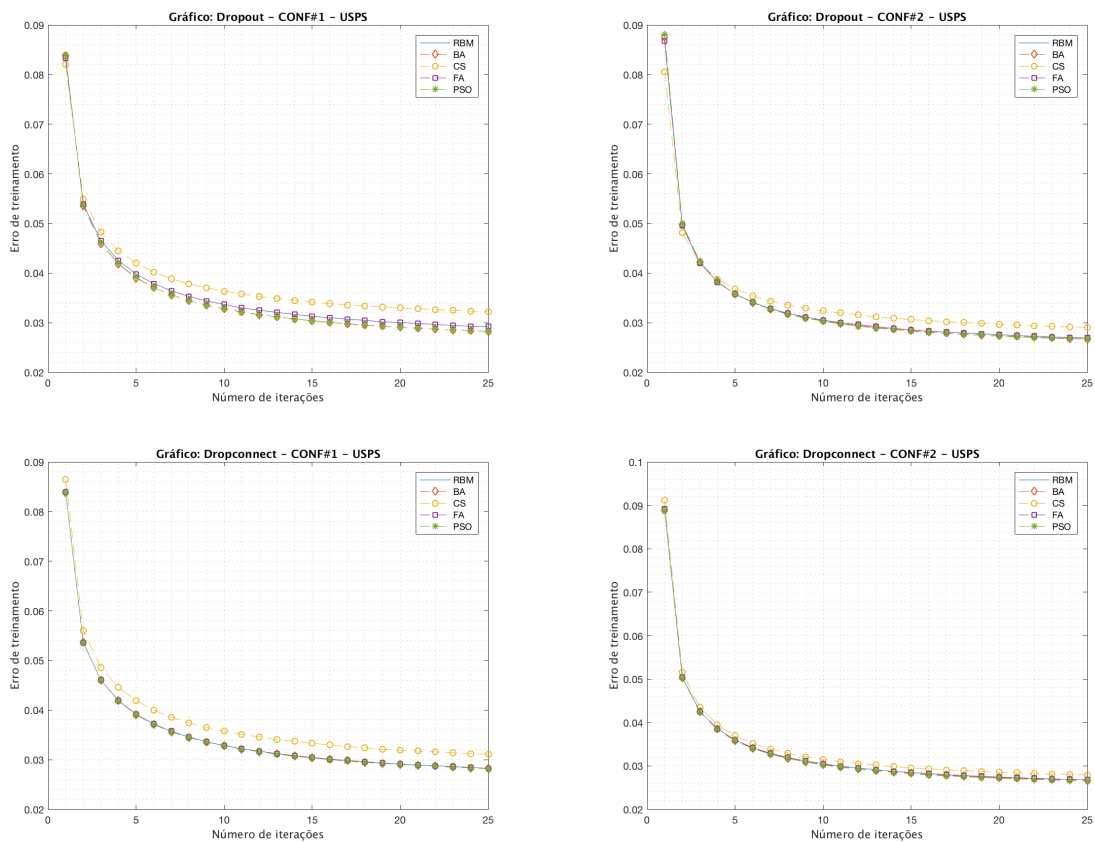
Fonte: Elaborado pelo autor.

experimento. Pode-se observar que na maioria dos casos o erro de treinamento fica maior com a utilização do *Dropout* e *Dropconnect*, enquanto o erro de teste tende a ser menor ou estatisticamente igual à rede sem regularização. A Figura 24 é responsável por ilustrar os gráficos desta tabela.

USPS				
	CONF #1		CONF #2	
	<i>Dropout</i>	<i>Dropconnect</i>	<i>Dropout</i>	<i>Dropconnect</i>
RBM*	0,0233(0,0282) $\pm$ 0,0004[—]	0,0233(0,0282) $\pm$ 0,0004[—]	0,0214(0,0265) $\pm$ 0,0003[—]	0,0214(0,0265) $\pm$ 0,0003[—]
BA	0,0232(0,0282) $\pm$ 0,0003[0,0000]	0,0232(0,0282) $\pm$ 0,0003[0,0000]	0,0215(0,0267) $\pm$ 0,0003[0,0134]	0,0217(0,0267) $\pm$ 0,0003[0,0000]
CS	0,0256 (0,0322) $\pm$ 0,0032 [0,1096]	0,0258 (0,0311) $\pm$ 0,0028 [0,0001]	0,0227(0,0291) $\pm$ 0,0020[0,1563]	0,0227 (0,0279) $\pm$ 0,0012 [0,0001]
FA	0,0237(0,0292) $\pm$ 0,0013[0,0266]	0,0231(0,0283) $\pm$ 0,0003[0,0000]	0,0215(0,0270) $\pm$ 0,0003[0,0226]	0,0217(0,0268) $\pm$ 0,0007[0,0000]
PSO	0,0233(0,0282) $\pm$ 0,0003[0,0000]	0,0233(0,0282) $\pm$ 0,0002[0,0000]	0,0217(0,0267) $\pm$ 0,0003[0,0054]	0,0216 (0,0266) $\pm$ 0,0002 [0,0000]

Tabela 12 – Erro médio de teste (erro médio de treinamento) $\pm$ desvio padrão do erro de teste [parâmetro médio p], considerando RBMs sem regularização e otimizadas com *Dropout* e *Dropconnect* na base de dados USPS.

Figura 24 – Gráficos: *número de iterações x erro de treinamento* para a Tabela 12.



Fonte: Elaborado pelo autor.

6.2 Experimento B - DBMs

Esta seção é responsável por descrever os experimentos realizados com as DBMs. Nos experimentos realizados é possível observar que existem três diferentes tipos de DBMs: 1, 2 e 3 camadas. À medida que incrementamos camadas nas DBMs, é possível observar que o *Dropout* e *Dropconnect* tornam-se mais presentes, dado que uma maior quantidade de camadas implica em número maior de neurônios e consequentemente, em um número maior de possíveis sub-redes a serem utilizadas ao longo do treinamento.

As Tabelas 13 e 14 descrevem os resultados obtidos a partir da aplicação do *Dropout* e *Dropconnect*, respectivamente, na base de dados MNIST. Como estamos considerando arquiteturas mais profundas do que as RBMs, é plausível que as DBMs necessitem de uma aplicação maior destas técnicas de regularização e consigam obter resultados mais desejáveis com seu uso. A utilização do *Dropout* e *Dropconnect* mostra a necessidade ou não de neurônios em determinada camada quando utilizados. Por exemplo, podemos observar que na terceira camada dos experimentos a utilização do *Dropout* e *Dropconnect* tende a ser maior do que os modelos de 1 e 2 camadas. Portanto, podemos levantar a possibilidade desta rede não necessitar de tantos neurônios para aprender o conjunto de dados requisitado, fato que pode ser comprovado ao analisarmos os erros finais de teste entre, por exemplo, uma DBM de 1 camada e uma DBM de 3 camadas. A Figuras 25 e 26 são responsáveis por ilustrar os gráficos destas tabelas, respectivamente.

MNIST (CONF #1)			
Dropout			
	1L	2L	3L
DBM*	0,0359(0,0248) ± 0,0011[—]	0,0327(0,0732) ± 0,0013[—; —]	0,0350(0,0900) ± 0,0015[—; —; —]
BA	0,0355(0,0249) ± 0,0010[0,0013]	0,0335(0,0843) ± 0,0018[0,0156; 0,1534]	0,0367(0,1132) ± 0,0019[0,0276; 0,3059; 0,3340]
CS	0,0363(0,0300) ± 0,0017[0,0877]	0,0357(0,0996) ± 0,0020[0,1168; 0,3766]	0,0361(0,1103) ± 0,0014[0,1283; 0,2114; 0,4918]
FA	0,0358(0,0258) ± 0,0007[0,0144]	0,0342(0,0846) ± 0,0016[0,0556; 0,1841]	0,0368(0,1129) ± 0,0019[0,0389; 0,3130; 0,3407]
PSO	0,0364(0,0248) ± 0,0008[0,0000]	0,0331(0,0807) ± 0,0020[0,0002; 0,0874]	0,0365(0,1139) ± 0,0015[0,0104; 0,1882; 0,4446]

MNIST (CONF #2)			
Dropout			
	1L	2L	3L
DBM*	0,0383(0,0163) ± 0,0012[—]	0,0322(0,0303) ± 0,0008[—; —]	0,0375(0,0350) ± 0,0021[—; —; —]
BA	0,0381(0,0163) ± 0,0013[0,0010]	0,0302(0,0359) ± 0,0030[0,1273; 0,1900]	0,0314(0,0532) ± 0,0028[0,3549; 0,3815; 0,2563]
CS	0,0402(0,0199) ± 0,0028[0,1445]	0,0282(0,0423) ± 0,0014[0,3088; 0,3172]	0,0322(0,0541) ± 0,0022[0,3633; 0,3723; 0,3433]
FA	0,0383(0,0166) ± 0,0013[0,0121]	0,0278(0,0420) ± 0,0008[0,2999; 0,3437]	0,0304(0,0517) ± 0,0012[0,3503; 0,3618; 0,2266]
PSO	0,0383(0,0163) ± 0,0010[0,0000]	0,0292(0,0374) ± 0,0024[0,1777; 0,2026]	0,0301(0,0500) ± 0,0011[0,3355; 0,2886; 0,2123]

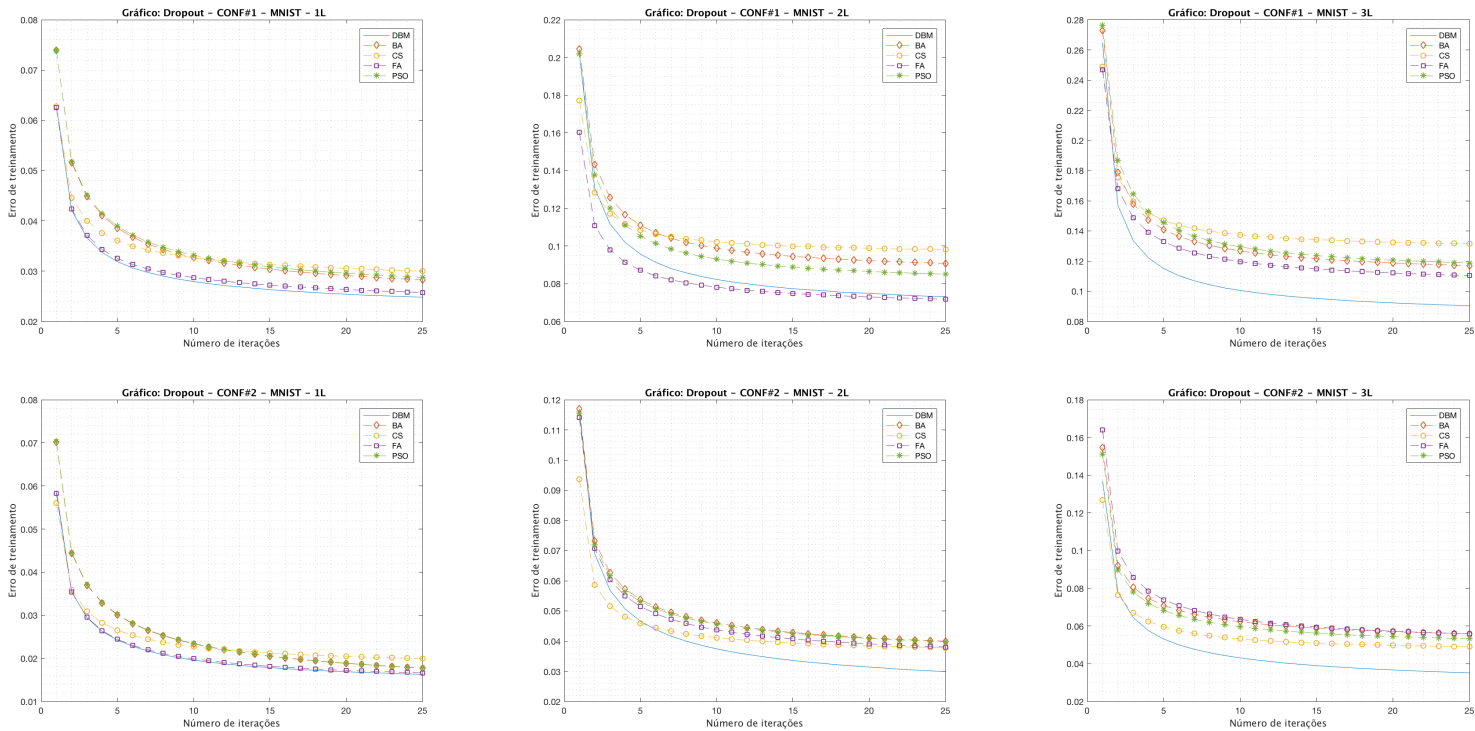
Tabela 13 – Erro médio de teste (erro médio de treinamento) ± desvio padrão do erro de teste [parâmetro p], considerando DBMs sem regularização e otimizadas com *Dropout* na base de dados MNIST.

MNIST (CONF #1)			
Dropconnect			
1L	2L	3L	
DBM*	0,0359(0,0248) $\pm$ 0,0011[—]	0,0327(0,0732) $\pm$ 0,0013[—; —]	0,0350(0,0900) $\pm$ 0,0015[—; —; —]
BA	0,0291(0,0299) $\pm$ 0,0033[0,0982]	0,0338(0,0774) $\pm$ 0,0026[0,0000; 0,0430]	0,0384 (0,1028) $\pm$ 0,0036 [0,0542; 0,1239; 0,1712]
CS	0,0287(0,0302) $\pm$ 0,0018[0,1018]	0,0375 (0,0817) $\pm$ 0,0025 [0,1612; 0,2395]	0,0389 (0,1065) $\pm$ 0,0017 [0,1474; 0,2160; 0,4541]
FA	0,0279(0,0307) $\pm$ 0,0005[0,1153]	0,0355 (0,0778) $\pm$ 0,0021 [0,1277; 0,1448]	0,0374 (0,1012) $\pm$ 0,0021 [0,1461; 0,1722; 0,4223]
PSO	0,0283(0,0298) $\pm$ 0,0011[0,0975]	0,0325(0,0741) $\pm$ 0,0016[0,0062; 0,0000]	0,0365(0,1088) $\pm$ 0,0020[0,0000; 0,0000; 0,3617]

MNIST (CONF #2)			
Dropconnect			
1L	2L	3L	
DBM*	0,0383 (0,0163) $\pm$ 0,0012 [—]	0,0322(0,0303) $\pm$ 0,0008[—; —]	0,0375 (0,0350) $\pm$ 0,0021 [—; —; —]
BA	0,0217(0,0215) $\pm$ 0,0010[0,2948]	0,0308(0,0324) $\pm$ 0,0017[0,1154; 0,0900]	0,0348(0,0385) $\pm$ 0,0018[0,1132; 0,0210; 0,2602]
CS	0,0216(0,0213) $\pm$ 0,0007[0,2867]	0,0321(0,0355) $\pm$ 0,0014[0,3330; 0,1886]	0,0357 (0,0438) $\pm$ 0,0015 [0,2868; 0,1825; 0,4035]
FA	0,0213(0,0213) $\pm$ 0,0004[0,2855]	0,0312(0,0338) $\pm$ 0,0016[0,2196; 0,1157]	0,0353 (0,0413) $\pm$ 0,0011 [0,2480; 0,1274; 0,3214]
PSO	0,0219(0,0214) $\pm$ 0,0007[0,2888]	0,0318(0,0310) $\pm$ 0,0027[0,0208; 0,0336]	0,0331(0,0368) $\pm$ 0,0017[0,0237; 0,0251; 0,1389]

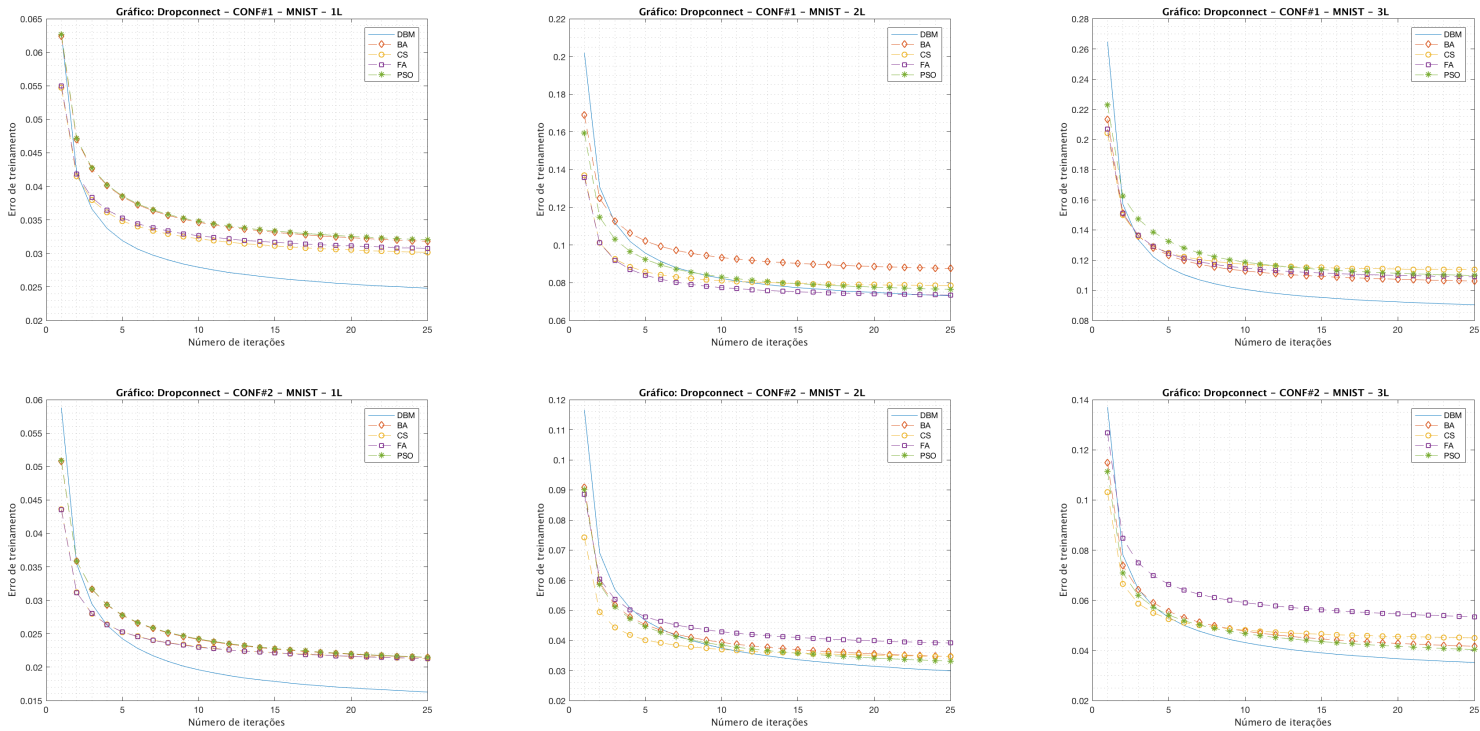
Tabela 14 – Erro médio de teste (erro médio de treinamento) $\pm$ desvio padrão do erro de teste [parâmetro p], considerando DBMs sem regularização e otimizadas com *Dropconnect* na base de dados MNIST.

Figura 25 – Gráficos: número de iterações x erro de treinamento para a Tabela 13.



Fonte: Elaborado pelo autor.

Figura 26 – Gráficos: *número de iterações x erro de treinamento* para a Tabela 14.



Fonte: Elaborado pelo autor.

Tratando agora de uma problema um pouco mais complicado, as Tabelas 15 e 16 descrevem os resultados obtidos a partir da utilização da base de dados Semeion. A única vantagem da utilização do *Dropout* pode ser observada na DBM de 3 camadas em sua segunda configuração, isto é, na arquitetura que emprega a maior quantidade de neurônios estabelecidos. Neste experimento, é possível observar que as técnicas BA, FA e PSO obtiveram um resultado superior à arquitetura sem regularização de acordo com o teste de Wilcoxon. Também é possível notar que à medida que incrementamos uma camada no modelo, a quantidade de *Dropout* introduzida também fica maior. Entretanto, não necessariamente redes mais profundas e complexas irão ter uma performance melhor do que redes mais simplistas. Este fator de crescimento é limitante até determinado ponto, onde acima deste ponto a quantidade de possíveis caminhos e nós receptores de informações tornam-se demasiadamente grandes, causando sub- ou super-treinamento dos dados, paradas em ótimos locais, estagnação do aprendizado, dentre outros problemas.

Ao analisarmos a utilização do *Dropconnect* na base de dados Semeion, podemos observar que o mesmo teve um comportamento superior ao *Dropout* e à rede sem regularização. As DBMs com regularização obtiveram resultados mais adequados, exceto na DBM de 2 camadas na primeira configuração. As técnicas que acabaram por encontrar ótimos locais e que introduziam um maior probabilidade de desligamento de conexões não obtiveram resultados tão satisfatórios, fato que reforça a teoria de que para determinada base de dados, é esperado

Semeion (CONF #1)			
Dropout			
	1L	2L	3L
DBM*	0,0753(0,0634) $\pm$ 0,0009[–]	0,0815(0,1251) $\pm$ 0,0010[–; –]	0,0826(0,1454) $\pm$ 0,0008[–; –; –]
BA	0,0754(0,0633) $\pm$ 0,0007[0,0000]	0,0822(0,1336) $\pm$ 0,0011[0,0000; 0,0998]	0,0828(0,1839) $\pm$ 0,0016[0,0001; 0,1470; 0,5495]
CS	0,0798 (0,0758) $\pm$ 0,0045 [0,0924]	0,0880 (0,1741) $\pm$ 0,0059 [0,0957; 0,4521]	0,0922 (0,2108) $\pm$ 0,0085 [0,1556; 0,4983; 0,4685]
FA	0,0755(0,0640) $\pm$ 0,0013[0,0035]	0,0828 (0,1638) $\pm$ 0,0012 [0,0018; 0,4446]	0,0833(0,1811) $\pm$ 0,0016[0,0081; 0,3093; 0,4008]
PSO	0,0752(0,0634) $\pm$ 0,0008[0,0000]	0,0820(0,1318) $\pm$ 0,0006[0,0000; 0,0831]	0,0840(0,2021) $\pm$ 0,0042[0,0000; 0,1875; 0,7861]

Semeion (CONF #2)			
Dropout			
	1L	2L	3L
DBM*	0,0548(0,0285) $\pm$ 0,0011[–]	0,0592(0,0475) $\pm$ 0,0008[–; –]	0,0655 (0,0581) $\pm$ 0,0008 [–; –; –]
BA	0,0537(0,0364) $\pm$ 0,0016[0,1813]	0,0600 (0,0502) $\pm$ 0,0018 [0,0126; 0,0629]	0,0642(0,0710) $\pm$ 0,0021[0,0054; 0,2291; 0,3440]
CS	0,0543(0,0392) $\pm$ 0,0019[0,1839]	0,0640 (0,0646) $\pm$ 0,0032 [0,1197; 0,3052]	0,0685 (0,0844) $\pm$ 0,0037 [0,2102; 0,2650; 0,4263]
FA	0,0544(0,0400) $\pm$ 0,0009[0,1990]	0,0600 (0,0526) $\pm$ 0,0013 [0,0183; 0,1596]	0,0639(0,0693) $\pm$ 0,0018[0,0197; 0,2167; 0,3269]
PSO	0,0539(0,0385) $\pm$ 0,0010[0,1753]	0,0590(0,0478) $\pm$ 0,0007[0,0000; 0,0175]	0,0637(0,0720) $\pm$ 0,0018[0,0005; 0,0978; 0,6073]

Tabela 15 – Erro médio de teste (erro médio de treinamento) $\pm$ desvio padrão do erro de teste [parâmetro p], considerando DBMs sem regularização e otimizadas com *Dropout* na base de dados Semeion.

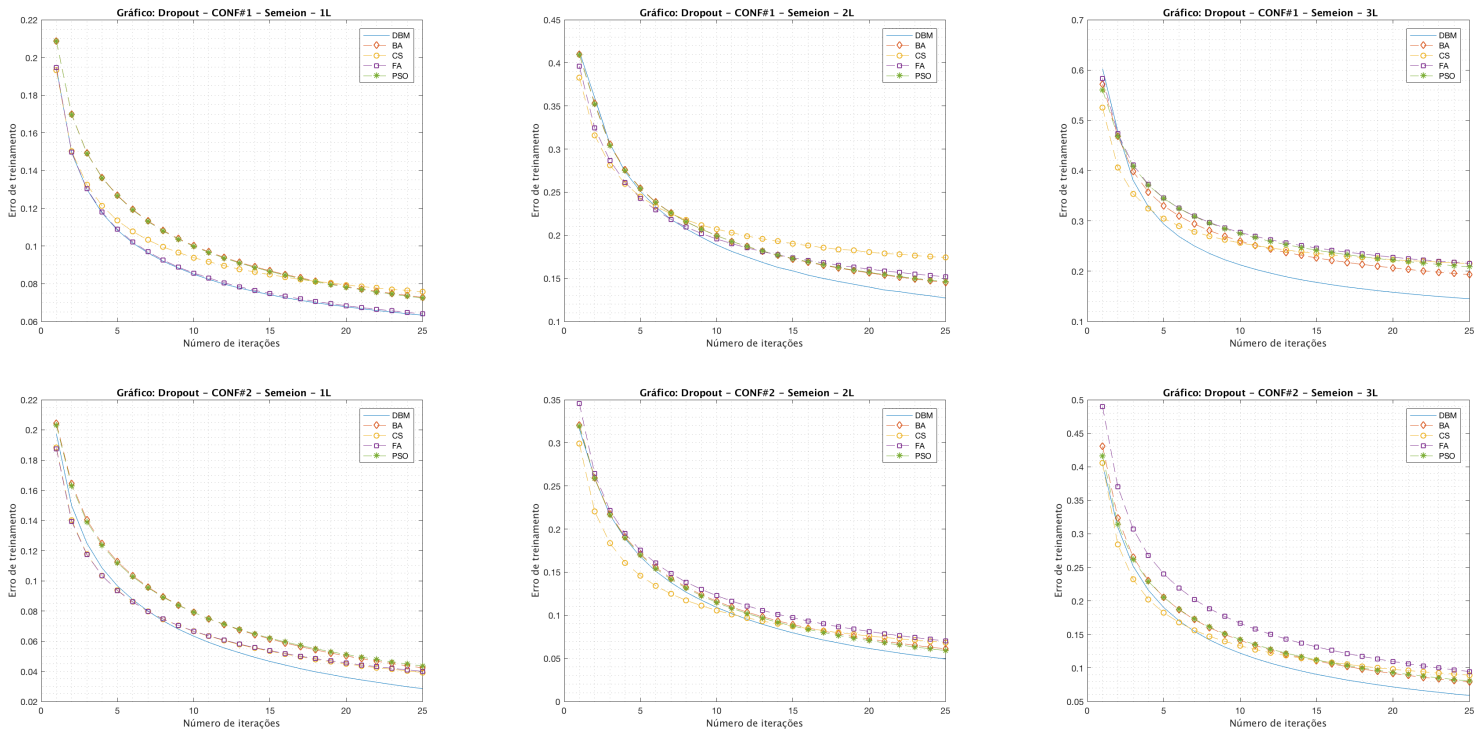
um número mínimo de neurônios para que o aprendizado seja adequado. A taxa de 0,5 para o desligamento de neurônios ou conexões é apenas uma expectativa média da usabilidade de alguma ferramenta matemática, isto é, 1/2, não sendo tão efetiva quando confrontado com aplicações de diferentes domínios de dados. As Figuras 27 e 28 ilustram o comportamento destas respectivas tabelas.

Semeion (CONF #1)			
Dropconnect			
	1L	2L	3L
DBM*	0,0753 (0,0634) $\pm$ 0,0009 [–]	0,0815(0,1251) $\pm$ 0,0010[–; –]	0,0826 (0,1454) $\pm$ 0,0008 [–; –; –]
BA	0,0728(0,0605) $\pm$ 0,0010[0,0022]	0,0825(0,1566) $\pm$ 0,0050[0,0000; 0,2710]	0,0815(0,1790) $\pm$ 0,0014[0,0000; 0,0120; 0,4447]
CS	0,0778 (0,0750) $\pm$ 0,0048 [0,1154]	0,0878 (0,1656) $\pm$ 0,0039 [0,1243; 0,3901]	0,0887 (0,1951) $\pm$ 0,0060 [0,0983; 0,3748; 0,5027]
FA	0,0734(0,0621) $\pm$ 0,0016[0,0143]	0,0842(0,1619) $\pm$ 0,0067[0,0368; 0,3663]	0,0827(0,1819) $\pm$ 0,0014[0,0051; 0,2981; 0,4034]
PSO	0,0728(0,0602) $\pm$ 0,0009[0,0000]	0,0830(0,1687) $\pm$ 0,0045[0,0000; 0,3600]	0,0825(0,2000) $\pm$ 0,0031[0,0000; 0,3514; 0,4510]

Semeion (CONF #2)			
Dropconnect			
	1L	2L	3L
DBM*	0,0548 (0,0285) $\pm$ 0,0011 [–]	0,0592 (0,0475) $\pm$ 0,0008 [–; –]	0,0655 (0,0581) $\pm$ 0,0008 [–; –; –]
BA	0,0467(0,0229) $\pm$ 0,0012[0,0198]	0,0587(0,0519) $\pm$ 0,0096[0,0011; 0,1167]	0,0589(0,0566) $\pm$ 0,0057[0,0000; 0,0461; 0,0899]
CS	0,0473(0,0280) $\pm$ 0,0020[0,0890]	0,0634 (0,0614) $\pm$ 0,0080 [0,1617; 0,1958]	0,0701 (0,0766) $\pm$ 0,0061 [0,1289; 0,2390; 0,4595]
FA	0,0465(0,0241) $\pm$ 0,0011[0,0346]	0,0568(0,0500) $\pm$ 0,0029[0,0082; 0,1386]	0,0605 (0,0641) $\pm$ 0,0017 [0,0085; 0,2380; 0,1337]
PSO	0,0471(0,0225) $\pm$ 0,0013[0,0098]	0,0554(0,0443) $\pm$ 0,0007[0,0000; 0,0000]	0,0573(0,0534) $\pm$ 0,0008[0,0000; 0,0139; 0,0000]

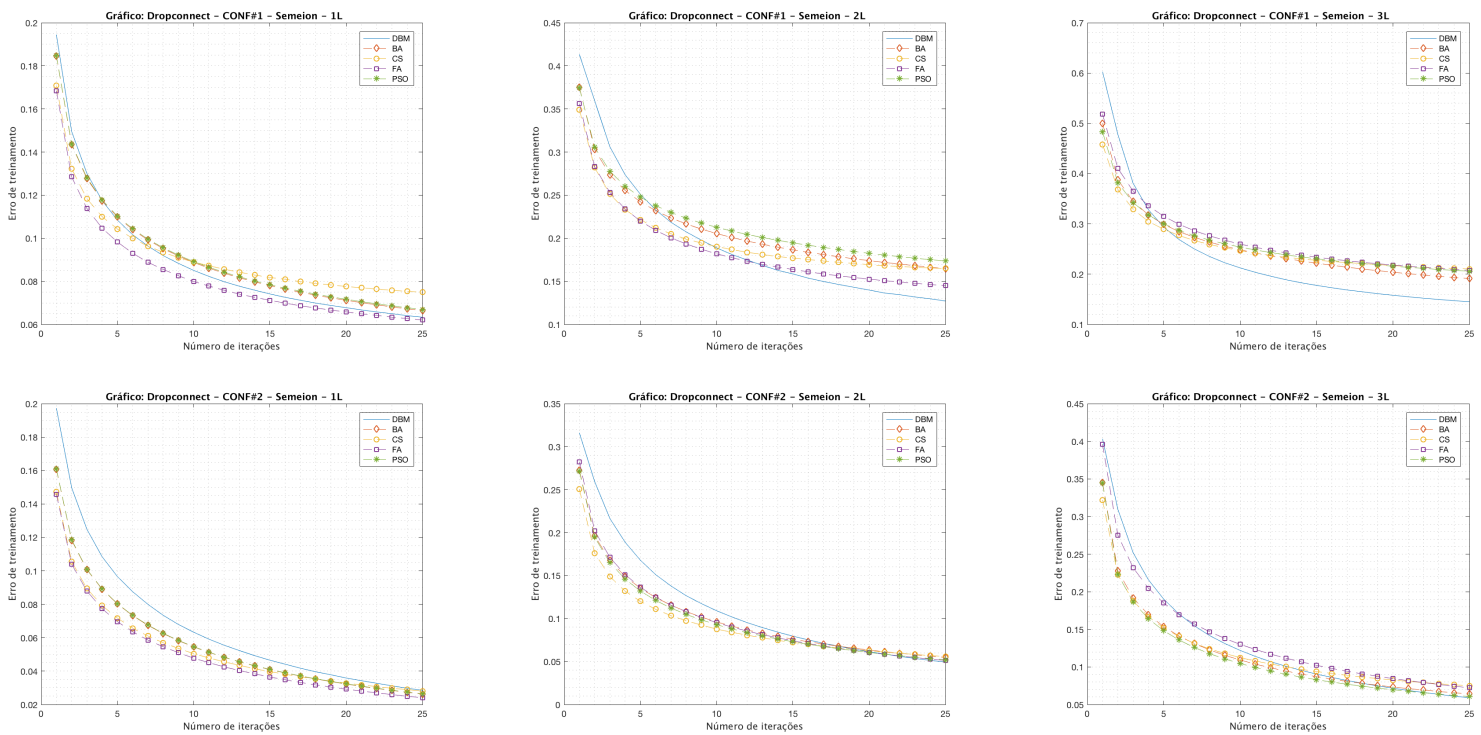
Tabela 16 – Erro médio de teste (erro médio de treinamento) $\pm$ desvio padrão do erro de teste [parâmetro p], considerando DBMs sem regularização e otimizadas com *Dropconnect* na base de dados Semeion.

Figura 27 – Gráficos: *número de iterações x erro de treinamento* para a Tabela 15.



Fonte: Elaborado pelo autor.

Figura 28 – Gráficos: *número de iterações x erro de treinamento* para a Tabela 16.



Fonte: Elaborado pelo autor.

Finalmente, a última base de dados a ser analisada é a USPS, que em nosso objetivo, obteve os maiores erros de reconstrução, sendo a tarefa mais “difícil” dentre todas. As Tabelas 17 e 18 descrevem os resultados obtidos a partir do uso desta base de dados, e as Figuras 29 e 30 ilustram o comportamento das mesmas. Em sua primeira configuração, nenhuma das meta-heurísticas foi capaz de encontrar um resultado satisfatório, tanto para o *Dropout* quanto para o *Dropconnect*. Entretanto, à medida que estendemos a quantidade de neurônios das arquiteturas (segunda configuração), é possível observar que a introdução da regularização obteve um resultado superior, encontrando erros de teste menores. Já com a utilização do *Dropconnect*, a técnica meta-heurística PSO conseguiu obter os melhores resultados em todas as três camadas, reforçando comparações entre outras técnicas meta-heurísticas (PAPA; ROSA; YANG, 2018), onde o PSO sempre despontava como uma das melhores alternativas.

USPS (CONF #1)			
Dropout			
	1L	2L	3L
DBM*	0,0916(0,0389) ± 0,0017 [—]	0,0505(0,0911) ± 0,0008 [—; —]	0,0495(0,1112) ± 0,0025 [—; —; —]
BA	0,0929(0,0421) ± 0,0023 [0,0370]	0,0520(0,0976) ± 0,0031 [0,0375; 0,0737]	0,0494(0,1203) ± 0,0015 [0,0004; 0,0000; 0,2304]
CS	0,0942 (0,0493) ± 0,0037 [0,1241]	0,0599 (0,1242) ± 0,0055 [0,2262; 0,3887]	0,0584 (0,1412) ± 0,0031 [0,2106; 0,2597; 0,5573]
FA	0,0924(0,0455) ± 0,0015 [0,0727]	0,0521(0,0979) ± 0,0029 [0,0371; 0,0756]	0,0518 (0,1299) ± 0,0034 [0,0438; 0,1357; 0,4803]
PSO	0,0920(0,0396) ± 0,0014 [0,0061]	0,0507(0,0914) ± 0,0012 [0,0009; 0,0041]	0,0498(0,1304) ± 0,0028 [0,0017; 0,0014; 0,4838]

USPS (CONF #2)			
Dropout			
	1L	2L	3L
DBM*	0,0678 (0,0215) ± 0,0011 [—]	0,0353 (0,0417) ± 0,0011 [—; —]	0,0349 (0,0493) ± 0,0011 [—; —; —]
BA	0,0665(0,0215) ± 0,0010 [0,0000]	0,0329 (0,0524) ± 0,0019 [0,0422; 0,3197]	0,0348 (0,0659) ± 0,0020 [0,0397; 0,3209; 0,4325]
CS	0,0799 (0,0272) ± 0,0105 [0,1319]	0,0350 (0,0592) ± 0,0040 [0,1627; 0,4206]	0,0358 (0,0704) ± 0,0042 [0,1340; 0,3990; 0,4338]
FA	0,0696 (0,0224) ± 0,0050 [0,0213]	0,0312(0,0521) ± 0,0010 [0,0081; 0,3732]	0,0334(0,0605) ± 0,0009 [0,0179; 0,2620; 0,3463]
PSO	0,0676(0,0215) ± 0,0016 [0,0001]	0,0318(0,0471) ± 0,0023 [0,0000; 0,2015]	0,0338(0,0598) ± 0,0007 [0,0025; 0,1733; 0,4344]

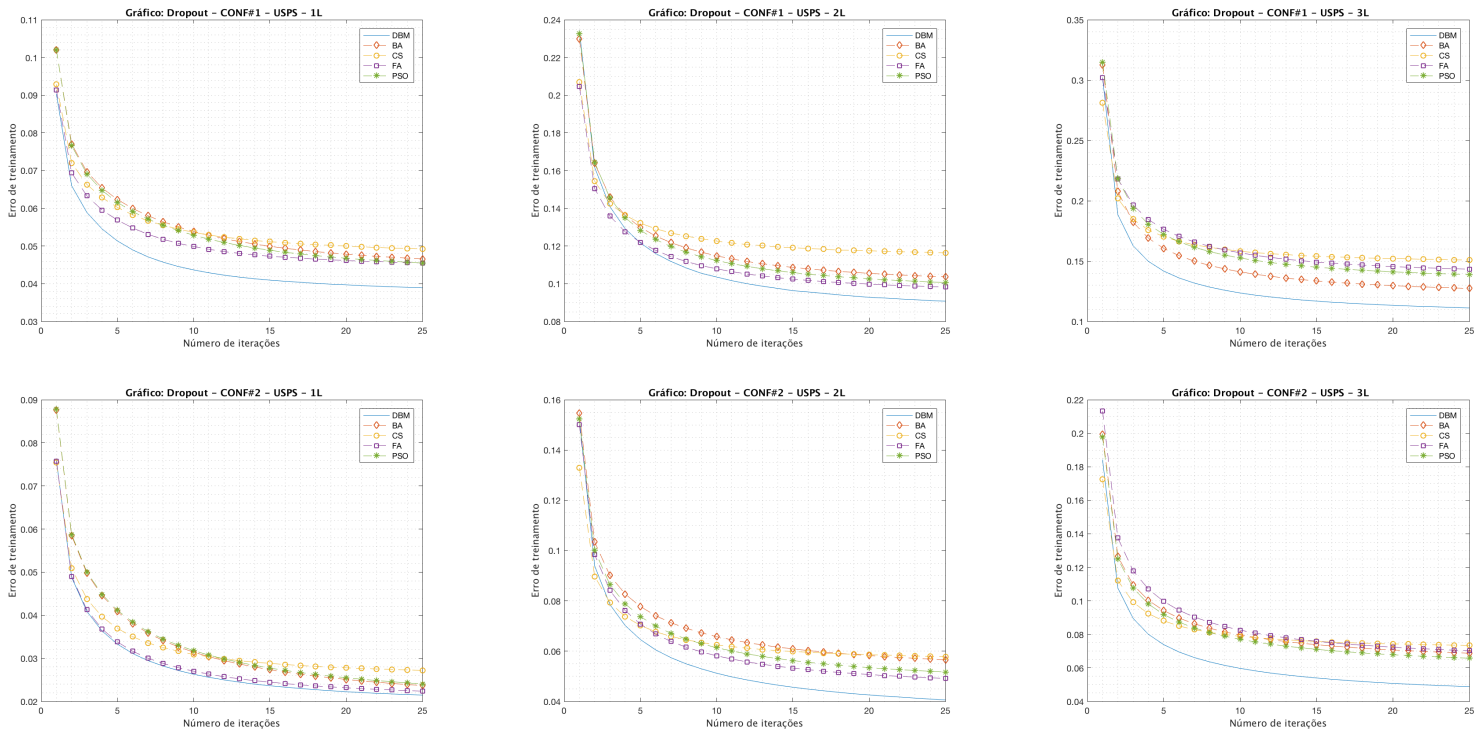
Tabela 17 – Erro médio de teste (erro médio de treinamento) ± desvio padrão do erro de teste [parâmetro p], considerando DBMs sem regularização e otimizadas com *Dropout* na base de dados USPS.

USPS (CONF #1)			
Dropconnect			
	1L	2L	3L
DBM*	0,0916 (0,0389) ± 0,0017 [—]	0,0505(0,0911) ± 0,0008 [—; —]	0,0495(0,1112) ± 0,0025 [—; —; —]
BA	0,0572(0,0572) ± 0,0006 [0,2349]	0,0546 (0,1055) ± 0,0018 [0,0092; 0,1511]	0,0535 (0,1263) ± 0,0030 [0,0287; 0,1239; 0,3005]
CS	0,0599(0,0571) ± 0,0076 [0,2341]	0,0621 (0,1061) ± 0,0095 [0,2017; 0,2026]	0,0653 (0,1373) ± 0,0096 [0,1933; 0,3454; 0,4254]
FA	0,0567(0,0575) ± 0,0006 [0,2400]	0,0534 (0,1049) ± 0,0011 [0,0184; 0,1505]	0,0549 (0,1291) ± 0,0036 [0,0325; 0,1706; 0,3550]
PSO	0,0571(0,0573) ± 0,0008 [0,2379]	0,0530 (0,0958) ± 0,0017 [0,0000; 0,0559]	0,0524 (0,1159) ± 0,0009 [0,0000; 0,0180; 0,1529]

USPS (CONF #2)			
Dropconnect			
	1L	2L	3L
DBM*	0,0678 (0,0215) ± 0,0011 [—]	0,0353 (0,0417) ± 0,0011 [—; —]	0,0349 (0,0493) ± 0,0011 [—; —; —]
BA	0,0373(0,0323) ± 0,0011 [0,3297]	0,0365 (0,0527) ± 0,0068 [0,0712; 0,2741]	0,0346(0,0582) ± 0,0042 [0,0509; 0,1843; 0,2855]
CS	0,0376 (0,0318) ± 0,0013 [0,3160]	0,0358 (0,0526) ± 0,0042 [0,1608; 0,3091]	0,0381 (0,0675) ± 0,0036 [0,1039; 0,3993; 0,3068]
FA	0,0366(0,0331) ± 0,0004 [0,3520]	0,0324(0,0475) ± 0,0017 [0,0683; 0,1621]	0,0344 (0,0574) ± 0,0012 [0,0370; 0,1605; 0,2879]
PSO	0,0370(0,0327) ± 0,0005 [0,3401]	0,0319(0,0453) ± 0,0011 [0,0394; 0,0830]	0,0327(0,0498) ± 0,0007 [0,0012; 0,0000; 0,0892]

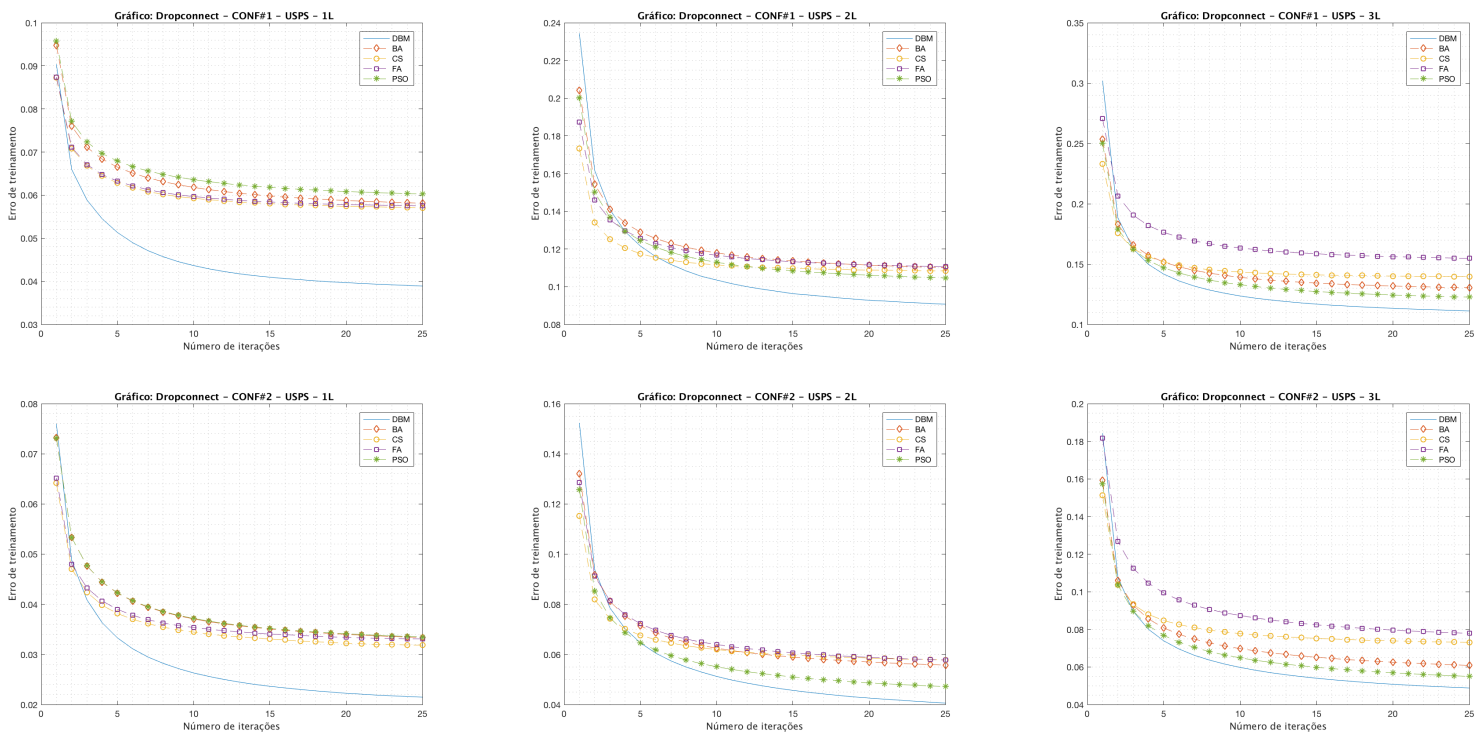
Tabela 18 – Erro médio de teste (erro médio de treinamento) ± desvio padrão do erro de teste [parâmetro p], considerando DBMs sem regularização e otimizadas com *Dropconnect* na base de dados USPS.

Figura 29 – Gráficos: *número de iterações x erro de treinamento* para a Tabela 17.



Fonte: Elaborado pelo autor.

Figura 30 – Gráficos: *número de iterações x erro de treinamento* para a Tabela 18.



Fonte: Elaborado pelo autor.

6.3 Experimento C - DBNs

Esta seção descreve os resultados obtidos através da utilização de DBNs. Note que dentre todas as arquiteturas utilizadas até agora, as DBNs obtiveram resultados tão bons quanto as RBMs, inclusive em arquiteturas que tiveram o parâmetro p encontrado por técnicas meta-heurísticas. Inicialmente, as Tabelas 19 e 20 são responsáveis por detalhar os experimentos acerca da base de dados MNIST, enquanto que as Figuras 31 e 32 ilustram os mesmos. Nesta base, é possível observar a manutenção do comportamento de uma maior probabilidade de regularização à medida que introduzimos mais camadas na arquitetura da rede. Também é possível observar que, de acordo com o teste de Wilcoxon, apenas dois casos de arquiteturas com regularização obtiveram resultados melhores que suas arquiteturas iniciais, isto é, nas DBNs de 3 camadas com *Dropconnect*. Uma interessante reflexão que também podemos observar nesta tabela os resultados estatisticamente melhores por Wilcoxon (células em negrito) correspondem às meta-heurísticas BA, FA e PSO, fato que também pode ser analisado se considerarmos os experimentos anteriores. O CS não se sai muitos bem nestas tarefas devido à sua metodologia de reposição de ninhos, onde possivelmente pode jogar boas soluções fora e iniciar uma nova busca no espaço de soluções.

MNIST (CONF #1)			
Dropout			
	1L	2L	3L
DBN*	0,0206(0,0197) $\pm$ 0,0003[—]	0,0374(0,0366) $\pm$ 0,0005[—; —]	0,0484(0,0479) $\pm$ 0,0005[—; —; —]
BA	0,0214(0,0206) $\pm$ 0,0020[0,0188]	0,0376(0,0368) $\pm$ 0,0004[0,0000; 0,0013]	0,0488(0,0484) $\pm$ 0,0012[0,0219; 0,0335; 0,0244]
CS	0,0270 (0,0264) $\pm$ 0,0061 [0,1831]	0,0456 (0,0450) $\pm$ 0,0058 [0,1780; 0,3087]	0,0540 (0,0536) $\pm$ 0,0024 [0,1643; 0,3232; 0,3091]
FA	0,0206(0,0197) $\pm$ 0,0003[0,0000]	0,0409 (0,0403) $\pm$ 0,0033 [0,0744; 0,1402]	0,0503 (0,0498) $\pm$ 0,0017 [0,0749; 0,1218; 0,2124]
PSO	0,0209(0,0199) $\pm$ 0,0003[0,0000]	0,0375(0,0367) $\pm$ 0,0005[0,0000; 0,0004]	0,0483(0,0479) $\pm$ 0,0004[0,0001; 0,0000; 0,0044]

MNIST (CONF #2)			
Dropout			
	1L	2L	3L
DBN*	0,0147(0,0134) $\pm$ 0,0004[—]	0,0312(0,0300) $\pm$ 0,0003[—; —]	0,0420(0,0411) $\pm$ 0,0004[—; —; —]
BA	0,0144(0,0131) $\pm$ 0,0002[0,0022]	0,0321(0,0309) $\pm$ 0,0023[0,0604; 0,0979]	0,0440(0,0428) $\pm$ 0,0046[0,0883; 0,2546; 0,1147]
CS	0,0178 (0,0167) $\pm$ 0,0041 [0,1984]	0,0369 (0,0356) $\pm$ 0,0049 [0,2444; 0,3480]	0,0449 (0,0435) $\pm$ 0,0042 [0,1452; 0,3542; 0,1983]
FA	0,0147(0,0135) $\pm$ 0,0005[0,0334]	0,0322 (0,0309) $\pm$ 0,0012 [0,1036; 0,1538]	0,0413(0,0400) $\pm$ 0,0006[0,1258; 0,2903; 0,0641]
PSO	0,0144(0,0132) $\pm$ 0,0002[0,0198]	0,0313(0,0300) $\pm$ 0,0003[0,0057; 0,0035]	0,0417 (0,0405) $\pm$ 0,0007 [0,0357; 0,2173; 0,0267]

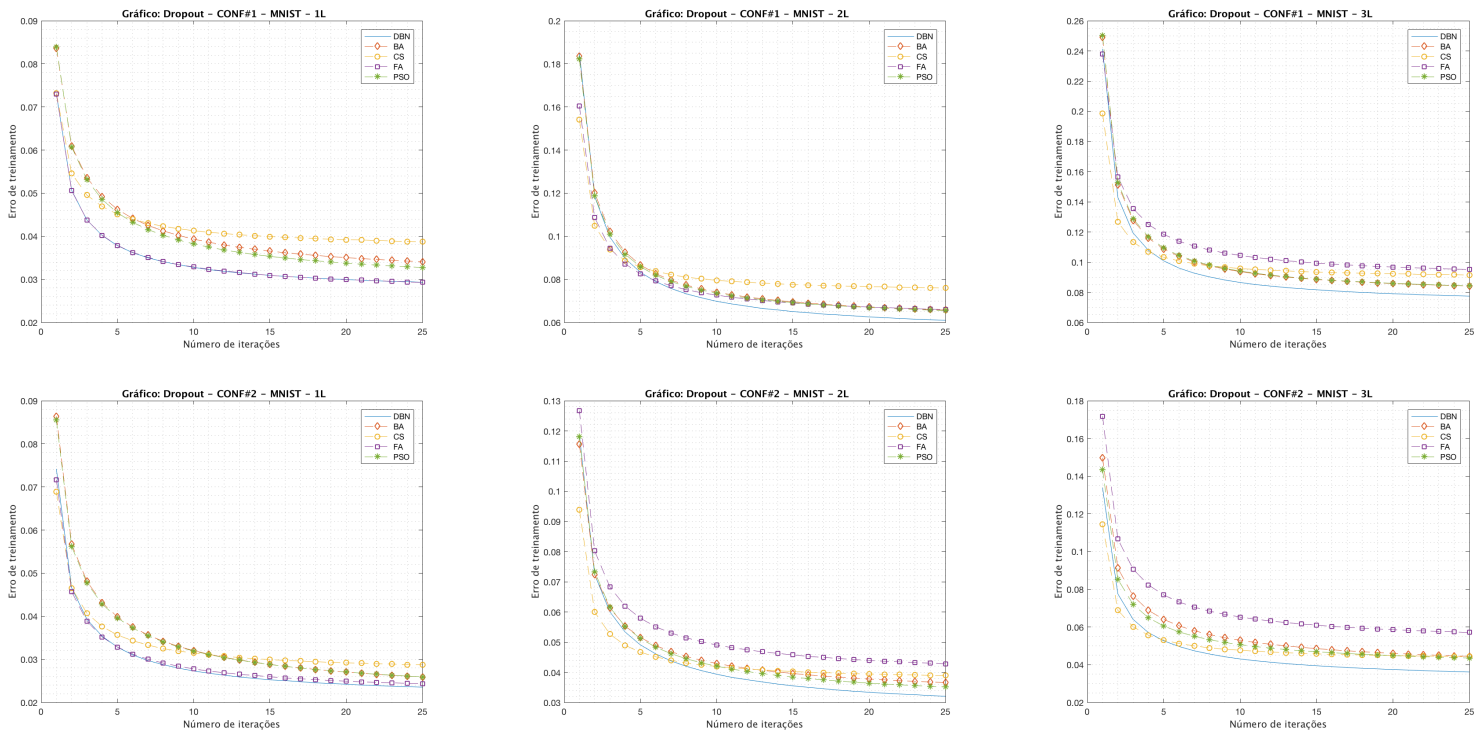
Tabela 19 – Erro médio de teste (erro médio de treinamento) $\pm$ desvio padrão do erro de teste [parâmetro p], considerando DBNs sem regularização e otimizadas com *Dropout* na base de dados MNIST.

MNIST (CONF #1)			
Dropconnect			
	1L	2L	3L
DBN*	0,0206(0,0197) $\pm$ 0,0003 [—]	0,0374(0,0366) $\pm$ 0,0005 [—; —]	0,0484 (0,0479) $\pm$ 0,0005 [—; —; —]
BA	0,0207(0,0198) $\pm$ 0,0003 [0,0000]	0,0384(0,0376) $\pm$ 0,0016 [0,0120; 0,0236]	0,0529 (0,0523) $\pm$ 0,0079 [0,0966; 0,0580; 0,1895]
CS	0,0242 (0,0233) $\pm$ 0,0041 [0,1038]	0,0472 (0,0463) $\pm$ 0,0079 [0,2037; 0,2352]	0,0609 (0,0601) $\pm$ 0,0087 [0,2160; 0,2969; 0,3430]
FA	0,0208 (0,0199) $\pm$ 0,0002 [0,0000]	0,0404 (0,0395) $\pm$ 0,0028 [0,1348; 0,0673]	0,0506 (0,0498) $\pm$ 0,0044 [0,1033; 0,1009; 0,1744]
PSO	0,0206(0,0197) $\pm$ 0,0001 [0,0000]	0,0375(0,0368) $\pm$ 0,0004 [0,0065; 0,0000]	0,0480(0,0476) $\pm$ 0,0002 [0,0115; 0,0114; 0,0123]

MNIST (CONF #2)			
Dropconnect			
	1L	2L	3L
DBN*	0,0147(0,0134) $\pm$ 0,0004 [—]	0,0312(0,0300) $\pm$ 0,0003 [—; —]	0,0420 (0,0411) $\pm$ 0,0004 [—; —; —]
BA	0,0145(0,0132) $\pm$ 0,0002 [0,0000]	0,0326(0,0314) $\pm$ 0,0040 [0,0411; 0,0442]	0,0425(0,0413) $\pm$ 0,0026 [0,1368; 0,0164; 0,0600]
CS	0,0155 (0,0142) $\pm$ 0,0007 [0,0932]	0,0357 (0,0343) $\pm$ 0,0085 [0,2031; 0,1302]	0,0501 (0,0485) $\pm$ 0,0060 [0,2246; 0,2686; 0,3178]
FA	0,0144(0,0132) $\pm$ 0,0002 [0,0052]	0,0322(0,0308) $\pm$ 0,0031 [0,1509; 0,0426]	0,0409(0,0395) $\pm$ 0,0010 [0,1435; 0,0388; 0,1110]
PSO	0,0145(0,0132) $\pm$ 0,0002 [0,0000]	0,0313(0,0300) $\pm$ 0,0004 [0,0210; 0,0000]	0,0410(0,0397) $\pm$ 0,0011 [0,1114; 0,0000; 0,0057]

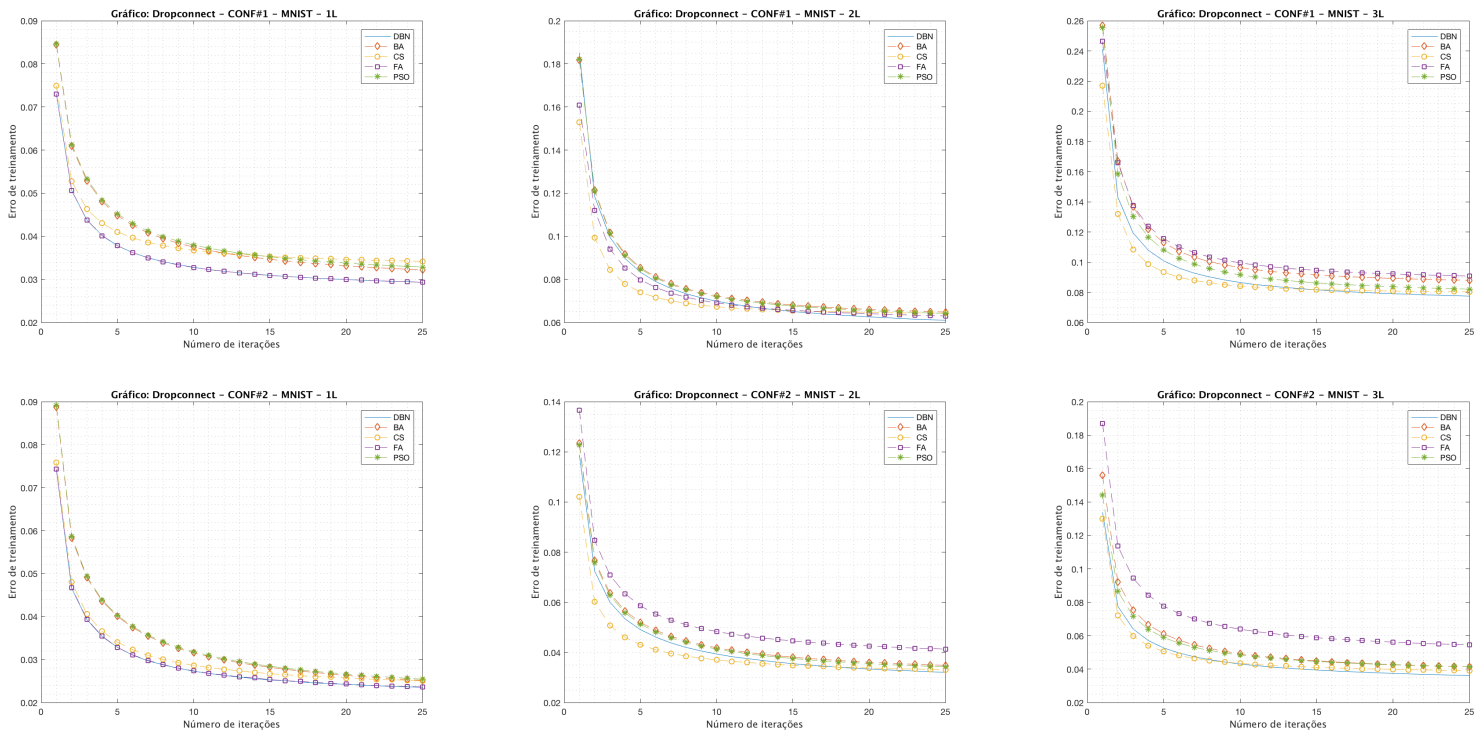
Tabela 20 – Erro médio de teste (erro médio de treinamento) $\pm$ desvio padrão do erro de teste [parâmetro p], considerando DBNs sem regularização e otimizadas com *Dropconnect* na base de dados MNIST.

Figura 31 – Gráficos: *número de iterações x erro de treinamento* para a Tabela 19.



Fonte: Elaborado pelo autor.

Figura 32 – Gráficos: *número de iterações x erro de treinamento* para a Tabela 20.



Fonte: Elaborado pelo autor.

Posteriormente, as Tabelas 21 e 22 relatam os experimentos realizados na base de dados Semeion, e as Figuras 33 e 34 ilustram as mesmas, respectivamente. Como mencionado anteriormente, esta base é uma tarefa um pouco mais complicada do que sua antecessora, elevando as taxas de erro de reconstrução, tanto para treino quanto para teste. Contudo, pode-se observar que para as DBNs, a introdução de mais camadas resultou no contrário do esperado, isto é, os erros de treino e teste cresceram consideravelmente e as probabilidades de regularização não foram tão eficientes quanto desejáveis.

Semeion (CONF #1)			
Dropout			
	1L	2L	3L
DBN*	0,0769(0,0586) $\pm$ 0,0007[—]	0,1044(0,0879) $\pm$ 0,0010[—; —]	0,1200(0,1057) $\pm$ 0,0017[—; —; —]
BA	0,0767(0,0583) $\pm$ 0,0010[0,0000]	0,1053 (0,0888) $\pm$ 0,0015 [0,0117; 0,0106]	0,1232(0,1092) $\pm$ 0,0092[0,0597; 0,0325; 0,0649]
CS	0,0788 (0,0611) $\pm$ 0,0014 [0,0471]	0,1151 (0,1011) $\pm$ 0,0062 [0,1438; 0,2402]	0,1346 (0,1223) $\pm$ 0,0060 [0,2503; 0,2311; 0,3239]
FA	0,0769(0,0587) $\pm$ 0,0008[0,0000]	0,1091 (0,0936) $\pm$ 0,0049 [0,0457; 0,1246]	0,1249 (0,1112) $\pm$ 0,0036 [0,0870; 0,1478; 0,1200]
PSO	0,0769(0,0587) $\pm$ 0,0007[0,0000]	0,1048 (0,0882) $\pm$ 0,0009 [0,0000; 0,0024]	0,1200(0,1056) $\pm$ 0,0014[0,0011; 0,0009; 0,0042]

Semeion (CONF #2)			
Dropout			
	1L	2L	3L
DBN*	0,0549(0,0310) $\pm$ 0,0012[—]	0,0828(0,0533) $\pm$ 0,0013[—; —]	0,0984(0,0688) $\pm$ 0,0015[—; —; —]
BA	0,0550(0,0310) $\pm$ 0,0009[0,0750]	0,0828(0,0526) $\pm$ 0,0016[0,0511; 0,0110]	0,0997(0,0693) $\pm$ 0,0036[0,0258; 0,0907; 0,0863]
CS	0,0568(0,0330) $\pm$ 0,0055[0,1352]	0,0893 (0,0594) $\pm$ 0,0051 [0,1738; 0,3174]	0,1059 (0,0745) $\pm$ 0,0080 [0,2026; 0,2574; 0,1979]
FA	0,0554(0,0312) $\pm$ 0,0008[0,0923]	0,0837 (0,0529) $\pm$ 0,0011 [0,0927; 0,1088]	0,1015 (0,0692) $\pm$ 0,0046 [0,1199; 0,2128; 0,2122]
PSO	0,0549(0,0309) $\pm$ 0,0010[0,0494]	0,0827(0,0532) $\pm$ 0,0012[0,0109; 0,0088]	0,0992(0,0692) $\pm$ 0,0031[0,0068; 0,0903; 0,0482]

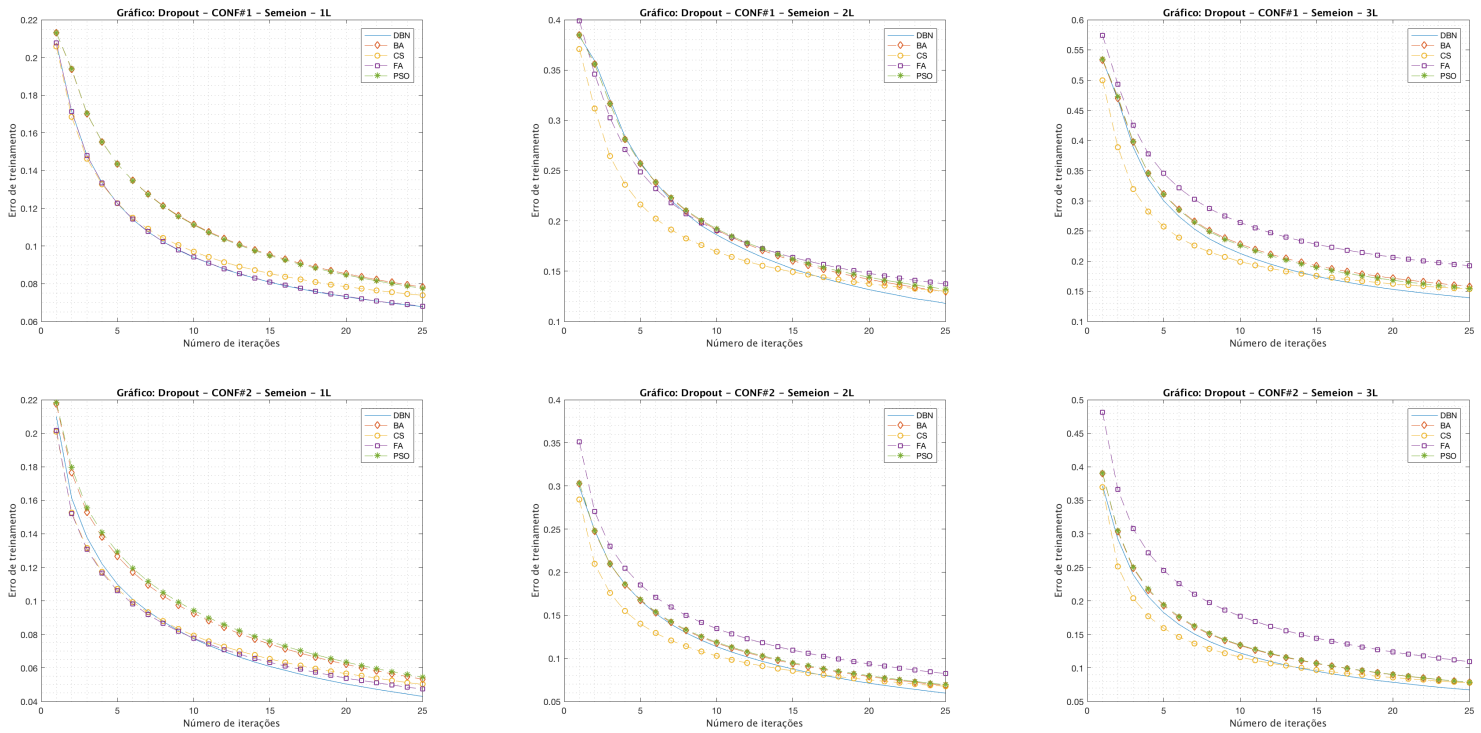
Tabela 21 – Erro médio de teste (erro médio de treinamento) $\pm$ desvio padrão do erro de teste [parâmetro p], considerando DBNs sem regularização e otimizadas com *Dropout* na base de dados Semeion.

Semeion (CONF #1)			
Dropconnect			
	1L	2L	3L
DBN*	0,0769(0,0586) $\pm$ 0,0007[—]	0,1044(0,0879) $\pm$ 0,0010[—; —]	0,1200(0,1057) $\pm$ 0,0017[—; —; —]
BA	0,0771(0,0588) $\pm$ 0,0009[0,0031]	0,1044(0,0880) $\pm$ 0,0009[0,0004; 0,0011]	0,1278(0,1141) $\pm$ 0,0166[0,0366; 0,0810; 0,1416]
CS	0,0801 (0,0619) $\pm$ 0,0023 [0,0712]	0,1162 (0,1003) $\pm$ 0,0072 [0,1432; 0,1629]	0,1374 (0,1226) $\pm$ 0,0102 [0,2713; 0,1492; 0,2141]
FA	0,0780(0,0598) $\pm$ 0,0030[0,0233]	0,1076 (0,0912) $\pm$ 0,0024 [0,0585; 0,0542]	0,1327 (0,1179) $\pm$ 0,0106 [0,1085; 0,1413; 0,2694]
PSO	0,0772(0,0589) $\pm$ 0,0009[0,0000]	0,1042(0,0877) $\pm$ 0,0010[0,0000; 0,0000]	0,1196(0,1052) $\pm$ 0,0013[0,0019; 0,0000; 0,0000]

Semeion (CONF #2)			
Dropconnect			
	1L	2L	3L
DBN*	0,0549(0,0310) $\pm$ 0,0012[—]	0,0828(0,0533) $\pm$ 0,0013[—; —]	0,0984(0,0688) $\pm$ 0,0015[—; —; —]
BA	0,0550(0,0310) $\pm$ 0,0011[0,0011]	0,0841(0,0541) $\pm$ 0,0039[0,0355; 0,0089]	0,1025(0,0723) $\pm$ 0,0094[0,0348; 0,0589; 0,1395]
CS	0,0590 (0,0344) $\pm$ 0,0059 [0,1193]	0,0929 (0,0620) $\pm$ 0,0070 [0,1468; 0,2269]	0,1155 (0,0831) $\pm$ 0,0094 [0,1966; 0,3487; 0,3256]
FA	0,0547(0,0308) $\pm$ 0,0014[0,0002]	0,0867 (0,0558) $\pm$ 0,0054 [0,0682; 0,1194]	0,1031 (0,0701) $\pm$ 0,0037 [0,0782; 0,1530; 0,1740]
PSO	0,0551(0,0311) $\pm$ 0,0013[0,0000]	0,0826(0,0530) $\pm$ 0,0013[0,0000; 0,0000]	0,0982(0,0682) $\pm$ 0,0015[0,0150; 0,0009; 0,0126]

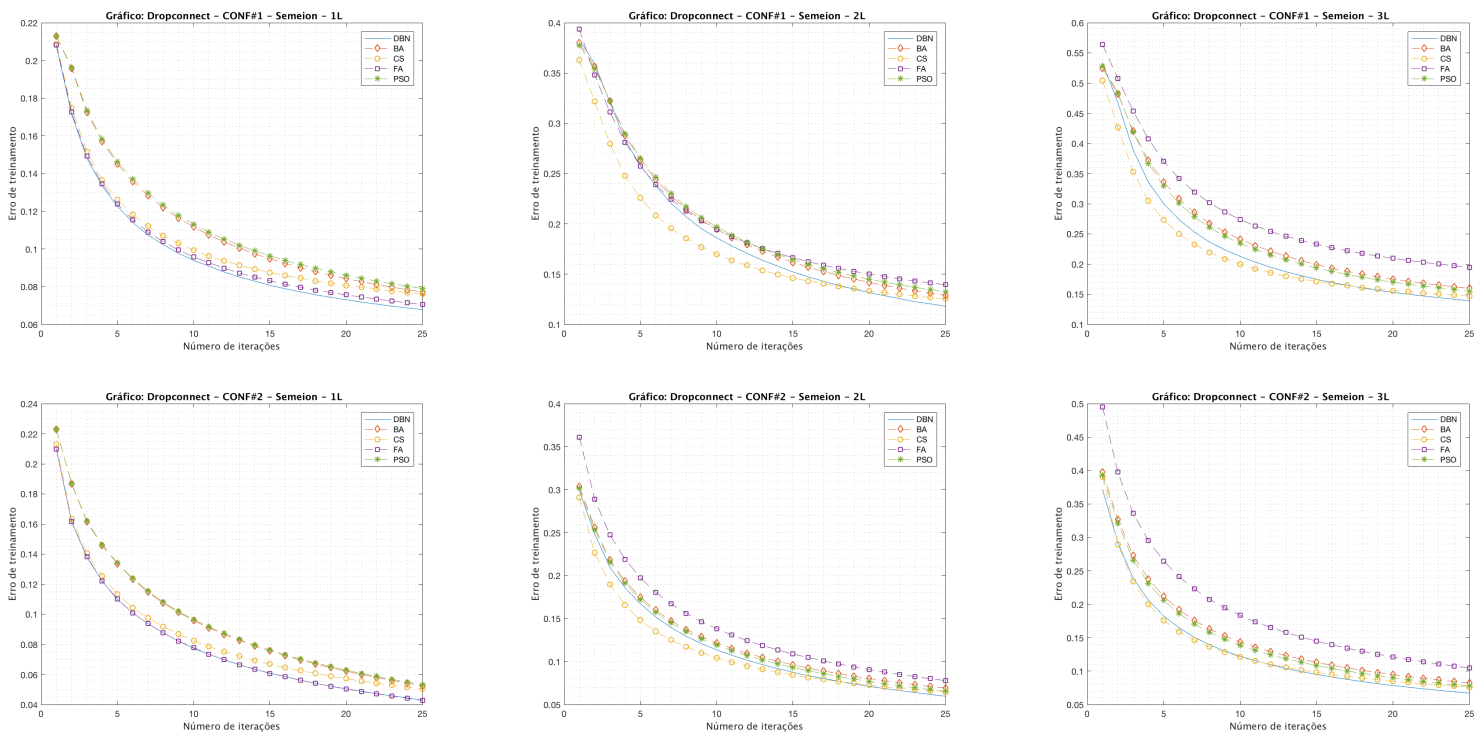
Tabela 22 – Erro médio de teste (erro médio de treinamento) $\pm$ desvio padrão do erro de teste [parâmetro p], considerando DBNs sem regularização e otimizadas com *Dropconnect* na base de dados Semeion.

Figura 33 – Gráficos: *número de iterações x erro de treinamento* para a Tabela 21.



Fonte: Elaborado pelo autor.

Figura 34 – Gráficos: *número de iterações x erro de treinamento* para a Tabela 22.



Fonte: Elaborado pelo autor.

Finalmente, as Tabelas 23 e 24 descrevem os resultados obtidos na base de dados USPS, enquanto que as Figuras 35 e 36 ilustram os mesmos. Novamente, é possível observar que a introdução de camadas à arquitetura também atenuou o processo de aprendizado, elevando as taxas de erro nos conjuntos de treinamento e teste. Particularmente, as DBNs obtiveram um comportamento homogêneo ao longo dos experimentos realizados nas três bases de dados, algo já esperado.

USPS (CONF #1)			
Dropout			
	1L	2L	3L
DBN*	0,0373(0,0331) ± 0,0003[—]	0,0549 (0,0504) ± 0,0004 [—; —]	0,0688(0,0643) ± 0,0005[—; —; —]
BA	0,0373(0,0331) ± 0,0003[0,0000]	0,0568(0,0523) ± 0,0067[0,0054; 0,0517]	0,0751(0,0704) ± 0,0096[0,0995; 0,1580; 0,0940]
CS	0,0436 (0,0394) ± 0,0047 [0,1252]	0,0686 (0,0639) ± 0,0064 [0,2857; 0,1804]	0,0829 (0,0781) ± 0,0086 [0,2602; 0,2968; 0,1635]
FA	0,0374(0,0332) ± 0,0007[0,0035]	0,0580 (0,0535) ± 0,0029 [0,0339; 0,0776]	0,0733 (0,0686) ± 0,0036 [0,0844; 0,1003; 0,1550]
PSO	0,0372(0,0330) ± 0,0003[0,0000]	0,0545(0,0500) ± 0,0004[0,0000; 0,0000]	0,0691(0,0644) ± 0,0005[0,0105; 0,0000; 0,0000]

USPS (CONF #2)			
Dropout			
	1L	2L	3L
DBN*	0,0233(0,0186) ± 0,0003[—]	0,0414(0,0355) ± 0,0003[—; —]	0,0535(0,0472) ± 0,0004[—; —; —]
BA	0,0232(0,0186) ± 0,0002[0,0004]	0,0412(0,0353) ± 0,0004[0,0000; 0,0007]	0,0578 (0,0514) ± 0,0056 [0,1776; 0,1571; 0,1252]
CS	0,0254 (0,0209) ± 0,0016 [0,1248]	0,0470 (0,0412) ± 0,0039 [0,1410; 0,3177]	0,0629 (0,0562) ± 0,0098 [0,2111; 0,2633; 0,2903]
FA	0,0234(0,0187) ± 0,0002[0,0030]	0,0419(0,0362) ± 0,0018[0,0439; 0,0444]	0,0556 (0,0492) ± 0,0016 [0,1374; 0,2254; 0,1508]
PSO	0,0232(0,0186) ± 0,0003[0,0000]	0,0412(0,0352) ± 0,0005[0,0000; 0,0000]	0,0534(0,0471) ± 0,0004[0,0243; 0,0371; 0,0000]

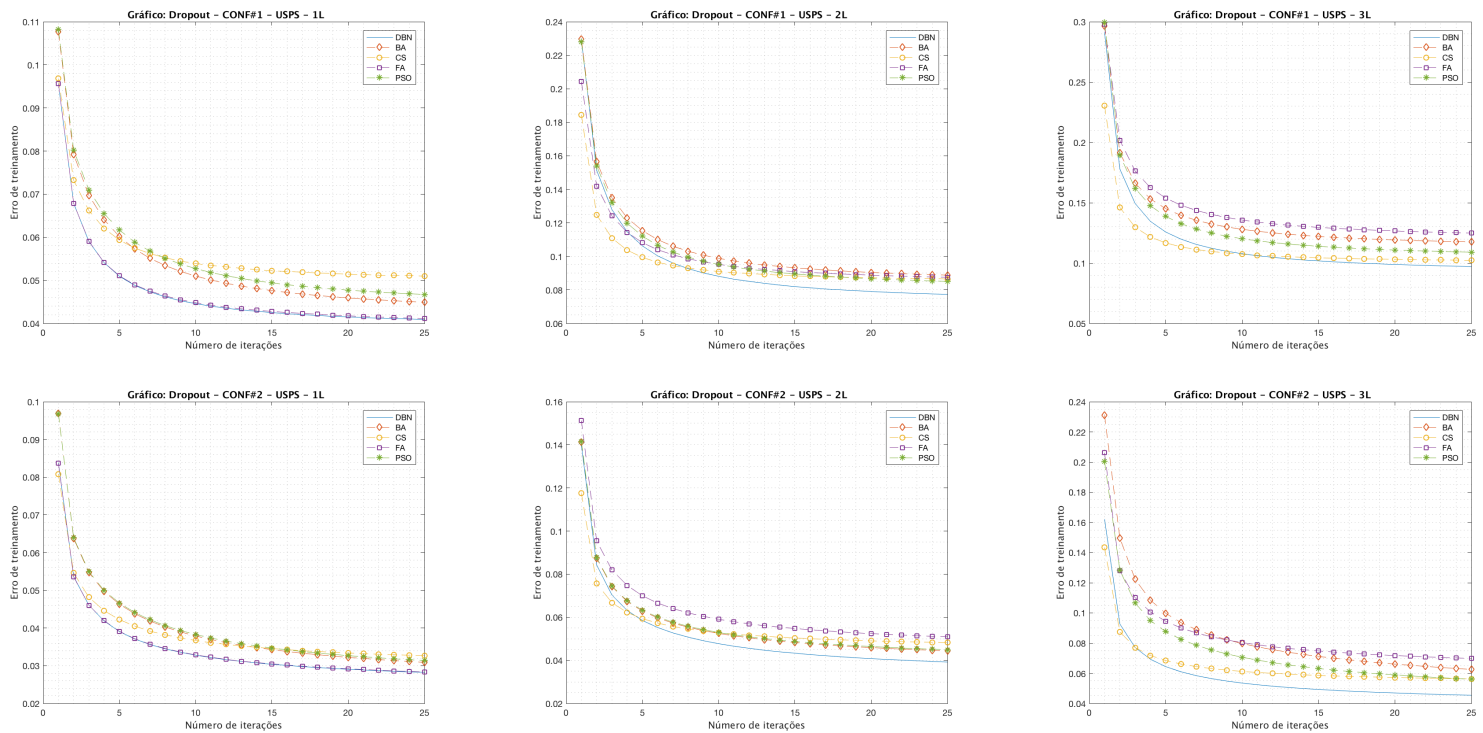
Tabela 23 – Erro médio de teste (erro médio de treinamento) ± desvio padrão do erro de teste [parâmetro p], considerando DBNs sem regularização e otimizadas com *Dropout* na base de dados USPS.

USPS (CONF #1)			
Dropconnect			
	1L	2L	3L
DBN*	0,0373(0,0331) ± 0,0003[—]	0,0549 (0,0504) ± 0,0004 [—; —]	0,0688(0,0643) ± 0,0005[—; —; —]
BA	0,0373(0,0331) ± 0,0003[0,0000]	0,0548(0,0503) ± 0,0005[0,0000; 0,0000]	0,0738(0,0691) ± 0,0139[0,0297; 0,0471; 0,1040]
CS	0,0428 (0,0377) ± 0,0045 [0,1097]	0,0659 (0,0606) ± 0,0092 [0,1197; 0,1694]	0,0940 (0,0884) ± 0,0142 [0,2312; 0,2610; 0,2711]
FA	0,0372(0,0329) ± 0,0003[0,0005]	0,0560(0,0513) ± 0,0020[0,0295; 0,0407]	0,0754 (0,0704) ± 0,0065 [0,0789; 0,1456; 0,1676]
PSO	0,0371(0,0329) ± 0,0003[0,0000]	0,0545(0,0500) ± 0,0003[0,0000; 0,0000]	0,0684(0,0639) ± 0,0005[0,0008; 0,0000; 0,0000]

USPS (CONF #2)			
Dropconnect			
	1L	2L	3L
DBN*	0,0233(0,0186) ± 0,0003[—]	0,0414(0,0355) ± 0,0003[—; —]	0,0535(0,0472) ± 0,0004[—; —; —]
BA	0,0233(0,0186) ± 0,0003[0,0000]	0,0420 (0,0360) ± 0,0016 [0,0006; 0,0318]	0,0574(0,0507) ± 0,0058[0,0422; 0,1424; 0,1080]
CS	0,0255 (0,0205) ± 0,0023 [0,1051]	0,0509 (0,0435) ± 0,0044 [0,2223; 0,2998]	0,0624 (0,0546) ± 0,0030 [0,1610; 0,3017; 0,3238]
FA	0,0233(0,0186) ± 0,0002[0,0019]	0,0421 (0,0359) ± 0,0014 [0,0534; 0,0490]	0,0543(0,0471) ± 0,0013[0,0963; 0,1136; 0,1493]
PSO	0,0233(0,0187) ± 0,0002[0,0000]	0,0411(0,0352) ± 0,0004[0,0007; 0,0000]	0,0534(0,0473) ± 0,0004[0,0000; 0,0000; 0,0000]

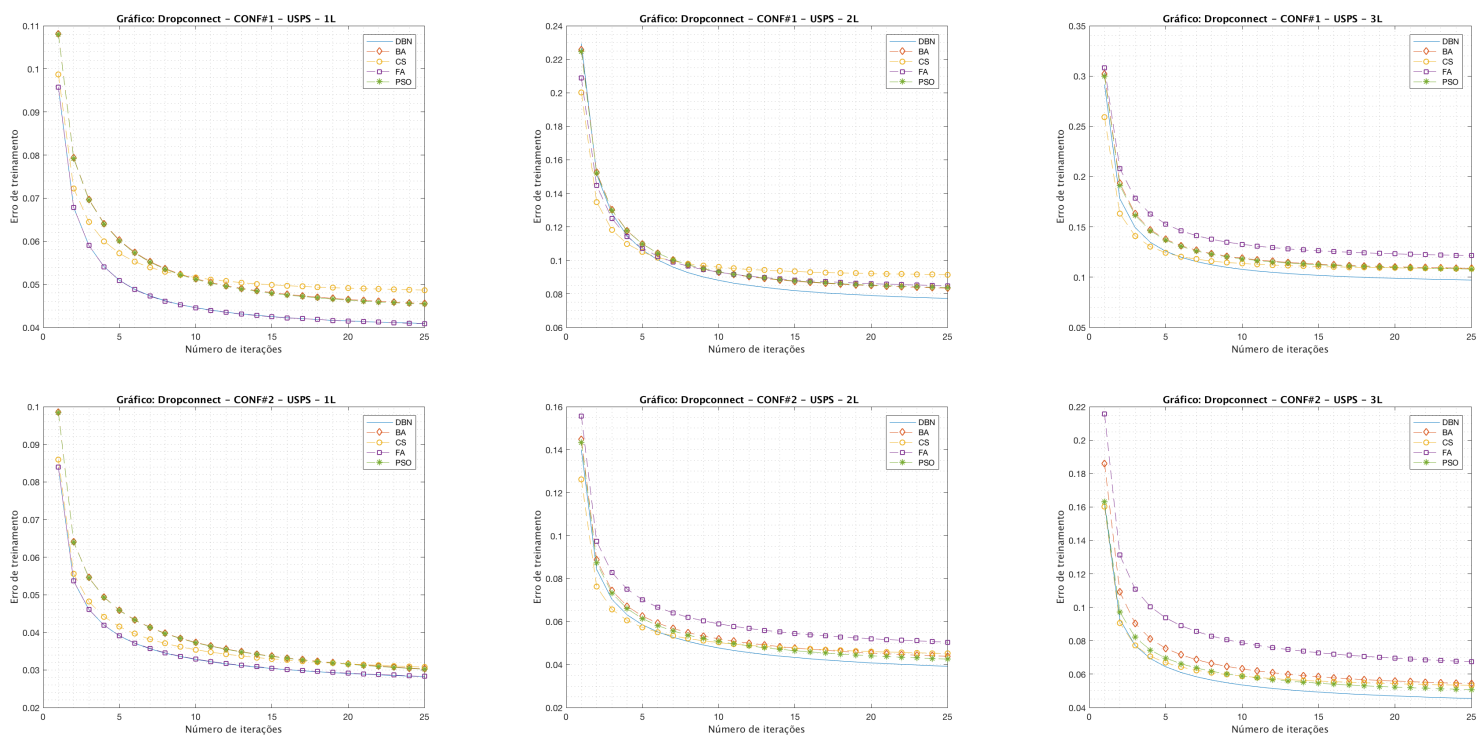
Tabela 24 – Erro médio de teste (erro médio de treinamento) ± desvio padrão do erro de teste [parâmetro p], considerando DBNs sem regularização e otimizadas com *Dropconnect* na base de dados USPS.

Figura 35 – Gráficos: *número de iterações x erro de treinamento* para a Tabela 23.



Fonte: Elaborado pelo autor.

Figura 36 – Gráficos: *número de iterações x erro de treinamento* para a Tabela 24.



Fonte: Elaborado pelo autor.

6.4 Experimento D - CNNs

Esta seção descreve os experimentos realizados com as CNNs. É possível notar que nestes experimentos existem três bases de dados não vistas anteriormente (CIFAR-10, Brain-PET e Brain-MRI) e uma técnica extra em cada experimento realizado (DropCaffe). Como a técnica de *Dropout* já é implementada no *framework* da Caffe, a mesma fornece um parâmetro inicial p de 0,5 (mesmo proposto pela literatura). Portanto, a ideia por trás de utilizar esta configuração é realmente verificar se o parâmetro proposto pela literatura é adequado para todos os casos utilizados.

Antes de iniciar uma discussão mais detalhada, é importante salientar que, em 5 de 6 bases de dados, as técnicas que utilizaram o *Dropout* tiveram um desempenho superior à rede sem o mesmo. Este fato já era previsto dado que esta é a principal ideia por trás de um processo de regularização, isto é, aumentar a performance de uma determinada técnica sem torná-la demasiada complexa.

A Tabela 25 descreve com maiores detalhes os experimentos realizados na base de dados MNIST. Novamente, é possível observar que todas as técnicas que utilizaram o *Dropout*, exceto o BA, são estatisticamente melhores de acordo com o teste de Wilcoxon. Outro ponto importante que deve ser levantado é a capacidade dos algoritmos meta-heurísticos em encontrar valores de p diferentes do proposto pela literatura, confirmando a hipótese de que é útil a realização de uma busca pelo parâmetro mais adequado.

MNIST	
<i>Dropout</i> (%)	
Caffe*	99,07 $\pm$ 0,04 [—]
DropCaffe*	99,18 $\pm$ 0,07 [0,5000]
BA	99,13 $\pm$ 0,07 [0,4988]
CS	99,14 $\pm$ 0,09 [0,4883]
FA	99,16 $\pm$ 0,06 [0,4630]
PSO	99,17 $\pm$ 0,07 [0,4559]

Tabela 25 – Acurácia média de teste $\pm$ desvio padrão da acurácia de teste [parâmetro médio p], considerando CNNs sem regularização e otimizadas com *Dropout* na base de dados MNIST.

Ao analisarmos a Tabela 26, podemos observar ainda com maior clareza a diferença que o *Dropout* proporciona ao processo de aprendizado. Ademais, nesta base de dados, também é possível observar que bons valores de acurácia também foram obtidos a partir da utilização de um parâmetro p totalmente distinto do proposto pela literatura. Este fato reforça ainda mais a possibilidade de existirem diversas configurações adequadas de parâmetros para cada problema em específico.

Semeion	
<i>Dropout (%)</i>	
Caffe*	97,62 $\pm$ 0,27 [–]
DropCaffe*	98,14 $\pm$ 0,34 [0,5000]
BA	98,30 $\pm$ 0,44 [0,7532]
CS	98,19 $\pm$ 0,63 [0,7546]
FA	98,23 $\pm$ 0,45 [0,7786]
PSO	97,66 $\pm$ 1,36 [0,8263]

Tabela 26 – Acurácia média de teste $\pm$ desvio padrão da acurácia de teste [parâmetro médio p], considerando CNNs sem regularização e otimizadas com *Dropout* na base de dados Semeion.

Ao analisarmos a Tabela 27, também é possível observar que as técnicas meta-heurísticas obtiveram um resultado aceitável, encontrando melhores taxas de acurácia e parâmetros p distintos. Neste caso, de acordo com Wilcoxon, as técnicas meta-heurísticas BA, FA e PSO obtiveram um resultado superior ao da Caffe com o parâmetro proposto pela literatura.

USPS	
<i>Dropout (%)</i>	
Caffe*	95,80 $\pm$ 0,19 [–]
DropCaffe*	96,21 $\pm$ 0,23 [0,5000]
BA	96,43 $\pm$ 0,19 [0,7688]
CS	96,28 $\pm$ 0,21 [0,6870]
FA	96,38 $\pm$ 0,24 [0,7319]
PSO	96,37 $\pm$ 0,18 [0,8031]

Tabela 27 – Acurácia média de teste $\pm$ desvio padrão da acurácia de teste [parâmetro médio p], considerando CNNs sem regularização e otimizadas com *Dropout* na base de dados USPS.

Fora do domínio dos dígitos manuscritos, a ideia por trás da utilização da base de dados CIFAR-10 é verificar como a abordagem proposta por este projeto se comporta em um nicho diferente de dados, agora representados por pequenas imagens de objetos coloridos. Novamente, a rede com a utilização do *Dropout* foi superior à todas as outras técnicas. Todavia neste experimento, o parâmetro p proposto pela literatura foi responsável por encontrar o melhor resultado. Tal fator deve-se à possibilidade de existência de vários ótimos locais, os quais podem aprisionar um processo de busca, metodologia principal dos algoritmos meta-heurísticos.

CIFAR-10	
<i>Dropout (%)</i>	
Caffe*	71,47 $\pm$ 0,64 [–]
DropCaffe*	72,08 $\pm$ 0,72 [0,5000]
BA	71,43 $\pm$ 0,66 [0,6430]
CS	71,16 $\pm$ 1,30 [0,6270]
FA	71,52 $\pm$ 0,73 [0,6629]
PSO	71,55 $\pm$ 0,60 [0,6655]

Tabela 28 – Acurácia média de teste $\pm$ desvio padrão da acurácia de teste [parâmetro médio p], considerando CNNs sem regularização e otimizadas com *Dropout* na base de dados CIFAR-10.

O primeiro problema relacionado às imagens de cérebro, Brain-PET, pode ser considerado mais simples que o segundo, Brain-MRI. Na Tabela 29 é possível verificar que quase todas as técnicas, exceto o CS, obtiveram resultados estatisticamente iguais. A falta de amostras na base de dados é um fator limitante do processo de aprendizado e, conseqüentemente, de uma possível regularização em cima do mesmo. Dado ao pequeno conjunto de dados, a rede foi treinada com uma quantidade mínima de iterações, inibindo a possibilidade de adaptação de seus neurônios e até mesmo o vício em seu processo de aprendizado.

Brain-PET	
<i>Dropout (%)</i>	
Caffe*	91,55 $\pm$ 3,01[–]
DropCaffe*	91,58 $\pm$ 3,41 [0,5000]
BA	91,14 $\pm$ 4,23 [0,5260]
CS	87,88 $\pm$ 4,09 [0,6023]
FA	89,62 $\pm$ 5,27 [0,5603]
PSO	89,59 $\pm$ 4,16 [0,4966]

Tabela 29 – Acurácia média de teste $\pm$ desvio padrão da acurácia de teste [parâmetro médio p], considerando CNNs sem regularização e otimizadas com *Dropout* na base de dados Brain-PET.

Finalmente, ao considerarmos o problema mais complexo desta seção de experimentos, a Tabela 30 é responsável por ilustrar que nem sempre uma busca retornará o resultado mais adequado. Pode-se observar nesta tabela que a rede com o parâmetro proposto pela literatura obteve os melhores resultados. Entretanto, as abordagens meta-heurísticas obtiveram resultados mais concisos e menos ruidosos, fato que pode ser analisado através do menor desvio padrão obtido por estas técnicas.

Brain-MRI	
<i>Dropout</i> (%)	
Caffe*	$56,68 \pm 3,81$ [–]
DropCaffe*	$60,38 \pm 4,11$ [0,5000]
BA	$58,23 \pm 2,51$ [0,8617]
CS	$57,69 \pm 2,07$ [0,7203]
FA	$57,00 \pm 1,57$ [0,8190]
PSO	$57,22 \pm 2,61$ [0,8371]

Tabela 30 – Acurácia média de teste $\pm$ desvio padrão da acurácia de teste [parâmetro médio p], considerando CNNs sem regularização e otimizadas com *Dropout* na base de dados Brain-MRI.

7 Conclusão

A inteligência artificial vem se tornando uma importante ferramenta para todos os seus usuários. É possível observar um grande crescimento em seu uso nas mais diversas aplicações, incluindo distintas áreas da computação. A possibilidade de "ensinar" uma máquina e fazê-la compreender um vasto rol de informações é de grande valia para os mais variados setores. Entretanto, tal tarefa não é tão trivial quanto soa, precisando de um conhecimento robusto sobre o assunto a fim de tentar utilizá-lo em algum problema. Diversas construções computacionais, isto é, bibliotecas e *frameworks* tentam abstrair conceitos matemáticos em funções mais simplistas, possibilitando sua utilização por um maior número de pessoas. Esta crescente onda de aplicar inteligência artificial à qualquer processo do cotidiano é um dos fatores mais motivadores destas construções, onde empresas públicas e privadas fomentam a automatização de seus processos. Em um mundo utópico, a inteligência artificial seria capaz de solucionar todos os problemas, porém não é o que vem acontecendo. Vários problemas oriundos de dados mal formatados, arquiteturas pessimamente modeladas, pesquisas falhas e sem objetividade, são fatores que contribuem para a dificuldade do avanço desta área. Não obstante, ainda há problemas relacionados à própria natureza desta área, dados que seus modelos são baseados em teorias matemáticas e probabilísticas.

A especialização dentro da área de inteligência artificial é outro importante fator que fomenta o seu desenvolvimento. Pesquisadores e aficcionados podem trabalhar em suas zonas de conforto e desenvolver técnicas mais complexas a partir deste pressuposto. Aprendizado de máquina, aprendizado em profundidade, processamento de linguagem natural, dentre outras, podem ser enumeradas como especificações da inteligência artificial. Dentro do campo do aprendizado em profundidade, a maioria dos modelos sofrem com o chamado super-treinamento dos dados, memorizando informações e tornando-se inadequados de serem utilizados em ambientes reais. Este problema, oriundo da natureza complexa destas arquiteturas neurais, acaba por desestimular e coibir o avanço que todos esperam. Contudo, a introdução dos métodos de regularização, técnicas capazes de amenizar o ruído obtido através do treinamento destas redes, vem se estabelecendo como uma importante fonte de soluções. Novamente, é importante salientar que a cada nova ferramenta descoberta, prós e contras também surgirão.

À medida que o aprendizado em profundidade em conjunto com as técnicas de regularização desenvolvem-se, outro problema começa a surgir e atrapalhar a sua evolução. Agora, a grande quantidade de parâmetros sensíveis a serem selecionados são capazes de transformar um ótimo aprendizado em um péssimo modelo. A utilização de processos de otimização, a fim de encontrar parâmetros mais adequados de forma automática, vem se demonstrando bastante promissora.

Este projeto propôs a utilização de otimizações inspiradas na natureza, denominadas meta-heurísticas, a fim de tentar solucionar o problema da seleção de parâmetros de técnicas de regularização em modelos de aprendizado em profundidade. Os resultados obtidos a partir dos experimentos realizados demonstram a possibilidade destas técnicas em selecionar conjuntos mais adequados de parâmetros, contribuindo para uma menor interação do ser humano com o processo e deixando com que a própria máquina o realize. Todavia, tal processo possui um custo computacional relativamente alto, onerando recursos computacionais e tempo, não possibilitando sua aplicação em problemas que necessitam de respostas *on-the-fly*, isto é, em tempo real. Portanto, a seção a seguir é responsável por explicar de forma sucinta um próximo passo a ser estudado dentro desta área, tentando solucionar o problema apresentado acima.

7.1 Direcionamentos para Trabalhos Futuros

A utilização de técnicas meta-heurísticas tem se demonstrado bastante úteis em uma gama de aplicações de aprendizado de máquina. O surgimento da computação paralela e seu constante desenvolvimento tem-se tornado uma importante fonte de conhecimento para a modelagem de algoritmos iterativos que gastam quantias demasiadas de tempo de execução. A ideia é realizar operações similares que não dependam de outros resultados ao mesmo tempo, ou seja, processar cada operação em um diferente núcleo de processamento. A introdução deste conceito facilita a modelagem de algoritmos meta-heurísticos, dado que várias operações, tais como, atualização de posições de agentes ou até mesmo cálculo de funções de aptidão, possam ser executadas ao mesmo tempo, encurtando o tempo de execução e introduzindo a possibilidade de expansão destes mesmos algoritmos. Uma pequena descrição de como isso poderia ser aproveitado é a seguinte: imagine que um certo algoritmo meta-heurístico paralelizável possa ser executado em 1 segundo. Sua função é encontrar um conjunto de parâmetros que auxiliem um correto treinamento de, por exemplo, íris humanas. A combinação desta otimização com uma técnica de aprendizado de máquina ao usuário comum pode facilitar a modelagem de uma estrutura de aprendizado.

A inteligência artificial tem se tornado cada vez mais presente em nosso cotidiano, até mesmo em tarefas básicas, como simples buscas na internet. A tentativa de aproximar um usuário comum à utilização destas ferramentas traz à comunidade uma possibilidade de também utilizá-las. A tendência é que no futuro cada pessoa possa utilizar técnicas inteligentes a fim de resolver tarefas de seu cotidiano. Ao facilitar uma adequada seleção de parâmetros de uma técnica de aprendizado de máquina, é possível introduzir uma facilidade no manuseio de um usuário comum, possibilitando, por exemplo, a criação de um sistema de detecção de pessoas com a câmera do seu próprio celular. Imagine você, criar em seu próprio espaço, um sistema inteligente que possa te auxiliar em determinadas tarefas?

Portanto, ao aproximar operações de otimização de tempos plausíveis de execução, podemos atingir tais objetivos. A principal ideia a ser desenvolvida em trabalhos futuros investigar formas de introduzir paralelismo nestas técnicas de otimização e criar bibliotecas que possam facilitar o uso das mesmas. O paralelismo tem se demonstrado como a melhor ferramenta para solucionar tal problema.

7.2 Publicações Realizadas

PEREIRA, D. R.; Filho, P.P.R.; ROSA, G.; PAPA, J. P.; ALBURQUEQUE, V. H. C. Stroke Lesion Detection Using Convolutional Neural Networks. In: *2018 International Joint Conference on Neural Networks*, Rio de Janeiro, 2018.

THOMAZELLA, R.; CASTANHO, J. E. C.; DOTTO, F.; RODRIGUES JUNIOR, O.; ROSA, G. H.; NILCEU, A.; PAPA, J. P. Environmental Monitoring using Drone Images and Convolutional Neural Networks. In: *IEEE International Geoscience and Remote Sensing Symposium, Valencia*, 2018.

ROSA, G.; PAPA, J. P.; SCHEIRER, W. Person Identification Using Handwriting Dynamics and Convolutional Neural Networks. *Deep Learning in Biometrics*, v. 1, p. 1-, 2018.

PAPA, J. P.; ROSA, G. H.; YANG, X. On the Hypercomplex-Based Search Spaces for Optimization Purposes. *Studies in Computational Intelligence*. 1ed.: Springer International Publishing, p. 119-147, 2018.

ROSA, G. H.; PAPA, J. P.; YANG, X. Handling dropout probability estimation in convolution neural networks using meta-heuristics. *Soft Computing*, p. 1-10, 2017.

PAPA, J. P.; ROSA, G. H.; PEREIRA, D. R.; YANG, X. Quaternion-based Deep Belief Networks Fine-Tuning. *Applied Soft Computing*, v. 60, p. 328-335, 2017.

ROSA, G. H.; AFONSO, L. C. S.; BALDASSIN, A.; PAPA, J. P.; YANG, X. Quaternionic Flower Pollination Algorithm. In: *17th international Conference on Computer Analysis of Images and Patterns*, Ystad, p. 47-58, 2018.

ROSA, G. H.; PAPA, J. P. Feature Selection Using Geometric Semantic Genetic Programming. In: *The Genetic and Evolutionary Computation Conference Companion*, 2017, Berlin. Proceedings of the Genetic and Evolutionary Computation Conference Companion on - GECCO '17. New York: ACM Press, 2017. p. 253-254.

COSTA, K. A. P.; SILVA, L. A.; MARTINS, G. B.; ROSA, G. H.; PIRES, R. G.; PAPA, J. P. On the Evaluation of Restricted Boltzmann Machines for Malware Identification. *International Journal of Computer Science and Information Security*, v. 5, p. 69-81, 2016.

PEREIRA, C. R.; WEBER, S. A. T.; HOOK, C.; ROSA, G. H.; PAPA, J. P. Deep Learning-Aided Parkinson's Disease Diagnosis from Handwritten Dynamics. In: *29th SIBGRAPI Conference on Graphics, Patterns and Images*, p. 340, 2016.

PEREIRA, C. R.; PEREIRA, D. R.; PAPA, J. P.; ROSA, G. H.; YANG, X. Convolutional Neural Networks Applied for Parkinson's Disease Identification. *Lecture Notes in Computer Science*. 1ed.: Springer International Publishing, p. 377-390, 2016.

PISANI, R.; COSTA, K. A. P.; ROSA, G. H.; PEREIRA, D. R.; PAPA, J. P. River Sediment Yield Classification Using Remote Sensing Imagery. In: *9th Workshop on Pattern Recognition in Remote Sensing*, p.1, 2016.

ROSA, G. H.; PAPA, J. P.; COSTA, K. A. P.; PASSOS JUNIOR, L. A.; PEREIRA, C. R.; YANG, X. Learning Parameters in Deep Belief Networks Through Firefly Algorithm. In: *7th International IAPR TC3 Workshop on Artificial Neural Networks in Pattern Recognition*, v. 9896. p. 138-149, 2016.

SILVA, L. A.; COSTA, K. A. P.; RIBEIRO, P. B.; ROSA, G. H.; PAPA, J. P. Learning Spam Features using Restricted Boltzmann Machines. *IADIS International Journal on Computer Science and Information Systems*, v. 11, p. 99-114, 2016.

Referências

- ACKLEY, D. H.; HINTON, G. E.; SEJNOWSKI, T. J. A learning algorithm for boltzmann machines. In: *Connectionist Models and their Implications: Readings from Cognitive Science*. [S.l.]: Ablex Publishing Corp., 1988. p. 285–307.
- AREL, I.; ROSE, D. C.; KARNOWSKI, T. Deep machine learning - a new frontier in artificial intelligence research. *IEEE Computational Intelligence Magazine*, IEEE, v. 5, n. 4, p. 13–18, 2010.
- BARONE, V.; COSSI, M.; TOMASI, J. Geometry optimization of molecular structures in solution by the polarizable continuum model. *Journal of Computational Chemistry*, Wiley Online Library, v. 19, n. 4, p. 404–417, 1998.
- BENGIO, Y.; SIMARD, P.; FRASCONI, P. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, IEEE, v. 5, n. 2, p. 157–166, 1994.
- BERTSEKAS, D. P. *Nonlinear programming*. [S.l.]: Athena Scientific, 1999.
- BISHOP, C. *Neural networks for pattern recognition*. [S.l.]: Oxford University Press, 1995.
- BROWN, C. T.; LIEBOVITCH, L. S.; GLENDON, R. Lévy flights in dobe ju/'hoansi foraging patterns. *Human Ecology*, Springer, v. 35, n. 1, p. 129–138, 2007.
- CHENG, Q. S.; BANDLER, J. W.; KOZIEL, S. Space mapping design framework exploiting tuning elements. *IEEE Transactions on Microwave Theory and Techniques*, IEEE, v. 58, n. 1, p. 136–144, 2010.
- CLEVERT, D.; UNTERTHINER, T.; HOCHREITER, S. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.
- COX, D.; PINTO, N. Beyond simple features: A large-scale feature search approach to unconstrained face recognition. In: *Automatic Face & Gesture Recognition and Workshops (FG 2011)*. [S.l.]: IEEE, 2011. p. 8–15.
- DESJARDINS, G.; COURVILLE, A.; BENGIO, Y.; VINCENT, P.; DELALLEAU, O. Parallel tempering for training of restricted boltzmann machines. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. [S.l.]: MIT Press, 2010. v. 9, p. 145–152.
- DIXIT, A. K. *Optimization in economic theory*. [S.l.]: Oxford University Press, 1990.
- FARABET, C.; COUPRIE, C.; NAJMAN, L.; LECUN, Y. Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, IEEE, v. 35, n. 8, p. 1915–1929, 2013.
- FUKUSHIMA, K. Cognitron: A self-organizing multilayered neural network. *Biological Cybernetics*, Springer, v. 20, n. 3, p. 121–136, 1975.

- FUKUSHIMA, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, Springer, v. 36, n. 4, p. 193–202, 1980.
- FUKUSHIMA, K.; MIYAKE, S. Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern Recognition*, Elsevier, v. 15, n. 6, p. 455–469, 1982.
- GEISLER, W. S.; ALBRECHT, D. G. Cortical neurons: Isolation of contrast gain control. *Vision Research*, Elsevier, v. 32, n. 8, p. 1409–1410, 1992.
- GHISO, J.; FRANGIONE, B. Amyloidosis and alzheimers disease. *Advanced Drug Delivery Reviews*, Elsevier, v. 54, n. 12, p. 1539–1551, 2002.
- GOLDBERG, D. E.; KUO, C. H. Genetic algorithms in pipeline optimization. *Journal of Computing in Civil Engineering*, American Society of Civil Engineers, v. 1, n. 2, p. 128–141, 1987.
- GRAHAM, B. Fractional max-pooling. *arXiv preprint arXiv:1412.6071*, 2014.
- HAWKINS, D. M. The problem of overfitting. *Journal of Chemical Information and Computer Sciences*, ACS Publications, v. 44, n. 1, p. 1–12, 2004.
- HINTON, G. E. Learning distributed representations of concepts. In: *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*. [S.l.]: Cognitive Science Society, 1986. v. 1, p. 47–60.
- HINTON, G. E. Training products of experts by minimizing contrastive divergence. *Neural Computation*, MIT Press, v. 14, n. 8, p. 1771–1800, 2002.
- HINTON, G. E. A practical guide to training restricted boltzmann machines. In: *Neural Networks: Tricks of the Trade*. [S.l.]: Springer, 2012. p. 599–619.
- HINTON, G. E.; DENG, L.; YU, D.; DAHL, G. E.; MOHAMED, A.; JAITLY, N.; SENIOR, A.; VANHOUCKE, V.; NGUYEN, P.; SAINATH, T. N. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, IEEE, v. 29, n. 6, p. 82–97, 2012.
- HINTON, G. E.; OSINDERO, S.; TEH, Y.-W. A fast learning algorithm for deep belief nets. *Neural Computation*, MIT Press, v. 18, n. 7, p. 1527–1554, 2006.
- HOCHREITER, S. *Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Technische Universität München*. 1991.
- HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. *Neural Computation*, MIT Press, v. 9, n. 8, p. 1735–1780, 1997.
- HOERL, A. E.; KENNARD, R. W. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, Taylor & Francis Group, v. 12, n. 1, p. 55–67, 1970.
- HUBEL, D.; WIESEL, T. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, Wiley Online Library, v. 160, n. 1, p. 106–154, 1962.

- JIA, Y.; SHELHAMER, E.; DONAHUE, J.; KARAYEV, S.; LONG, J.; GIRSHICK, R.; GUARDARRAMA, S.; DARRELL, T. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.
- KAVUKCUOGLU, K.; SERMANET, P.; BOUREAU, Y.; GREGOR, K.; MATHIEU, M.; CUN, Y. L. Learning convolutional feature hierarchies for visual recognition. In: *Advances in Neural Information Processing Systems 23*. [S.l.]: Neural Information Processing Systems, 2010. p. 1090–1098.
- KENNEDY, J.; EBERHART, R. C.; RUSSEL, C.; KENNEDY, J. F.; SHI, Y. *Swarm Intelligence*. [S.l.]: Morgan Kaufmann, 2001.
- KONNO, H.; YAMAZAKI, H. Mean-absolute deviation portfolio optimization model and its applications to tokyo stock market. *Management Science, INFORMS*, v. 37, n. 5, p. 519–531, 1991.
- LECUN, Y. *Modèles connexionnistes de l'apprentissage*. Tese (Doutorado) — Paris 6, 1987.
- LECUN, Y.; BOSER, B.; DENKER, J. S.; HENDERSON, D.; HOWARD, R. E.; HUBBARD, W.; JACKEL, L. D. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, MIT Press, v. 1, n. 4, p. 541–551, 1989.
- LECUN, Y.; BOSER, B.; DENKER, J. S.; HOWARD, R. E.; HABBARD, W.; JACKEL, L. D.; HENDERSON, D. Handwritten digit recognition with a back-propagation network. In: *Advances in Neural Information Processing Systems*. [S.l.]: Morgan Kaufmann, 1990. p. 396–404.
- LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE, IEEE*, v. 86, n. 11, p. 2278–2324, 1998.
- LECUN, Y.; KAVUKCUOGLU, K.; FARABET, C. Convolutional networks and applications in vision. In: *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. [S.l.]: IEEE, 2010. p. 253–256.
- MCCLELLAND, J. L.; RUMELHART, D. E.; HINTON, G. E. The appeal of parallel distributed processing. *MIT Press*, p. 3–44, 1986.
- MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, Springer, v. 5, n. 4, p. 115–133, 1943.
- MINSKY, M.; PAPERT, S. *Perceptrons: an introduction to computational geometry*. [S.l.]: MIT Press, 1969.
- NOWLAN, S. J.; HINTON, G. E. Simplifying neural networks by soft weight-sharing. *Neural Computation*, MIT Press, v. 4, n. 4, p. 473–493, 1992.
- PAPA, J. P.; ROSA, G. H.; YANG, X.-S. On the hypercomplex-based search spaces for optimization purposes. In: *Nature-Inspired Algorithms and Applied Optimization*. [S.l.]: Springer, 2018. p. 119–147.
- PAPA, J. P.; SCHEIRER, W.; COX, D. D. Fine-tuning deep belief networks using harmony search. *Applied Soft Computing*, Elsevier, v. 46, n. C, p. 875–885, 2016.
- PAPADIMITRIOU, C. H. The euclidean travelling salesman problem is np-complete. *Theoretical Computer Science*, Elsevier, v. 4, n. 3, p. 237–244, 1977.

- PAVLYUKEVICH, I. Lévy flights, non-local search and simulated annealing. *Journal of Computational Physics*, Elsevier, v. 226, n. 2, p. 1830–1844, 2007.
- PAYNE, R. B. *The Cuckoos*. [S.l.]: Oxford University Press, 2005.
- RAO, R. V.; SAVSANI, V. J.; VAKHARIA, D. P. Teaching–learning–based optimization: a novel method for constrained mechanical design optimization problems. *Computer-Aided Design*, Elsevier, v. 43, n. 3, p. 303–315, 2011.
- RAO, S. S.; RAO, S. S. *Engineering optimization: theory and practice*. [S.l.]: John Wiley & Sons, 2009.
- RARDIN, R. L. *Optimization in operations research*. [S.l.]: Prentice Hall, 1998. v. 166.
- ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, American Psychological Association, v. 65, n. 6, p. 386, 1958.
- RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. *Nature*, Nature Publishing Group, v. 323, p. 533–536, 1986.
- RUMELHART, D. E.; MCCLELLAND, J. L.; GROUP, P. R. *Parallel distributed processing*. [S.l.]: IEEE, 1988. v. 1.
- SALAKHUTDINOV, R.; HINTON, G. E. An efficient learning procedure for deep boltzmann machines. *Neural Computation*, MIT Press, v. 24, n. 8, p. 1967–2006, 2012.
- SMOLENSKY, P. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. [S.l.]: MIT Press, 1986. v. 1.
- SOCHER, R.; HUANG, E. H.; PENNIN, J.; MANNING, C. D.; NG, A. Y. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In: *Advances in Neural Information Processing Systems 24*. [S.l.]: Curran Associates, 2011. p. 801–809.
- SRIVASTAVA, N.; HINTON, G. E.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, JMLR, v. 15, n. 1, p. 1929–1958, 2014.
- TIBSHIRANI, R. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, JSTOR, v. 58, p. 267–288, 1996.
- TIELEMAN, T. Training restricted boltzmann machines using approximations to the likelihood gradient. In: *Proceedings of the 25th International Conference on Machine Learning*. [S.l.]: ACM, 2008. p. 1064–1071.
- WAN, L.; ZEILER, M.; ZHANG, S.; LECUN, Y.; FERGUS, R. Regularization of neural networks using dropconnect. In: *International Conference on Machine learning*. [S.l.]: JMLR, 2013. v. 3, p. 1058–1066.
- WELLING, M.; ROSEN-ZVI, M.; HINTON, G. E. Exponential family harmoniums with an application to information retrieval. In: *Advances in Neural Information Processing Systems 17*. [S.l.]: MIT Press, 2005. p. 1481–1488.
- WILCOXON, F. Individual comparisons by ranking methods. *Biometrics Bulletin*, JSTOR, v. 1, n. 6, p. 80–83, 1945.

WILKINSON, P. B.; LOKE, M. H.; MELDRUM, P. I.; CHAMBERS, J. E.; KURAS, O.; GUNN, D. A.; OGILVY, R. D. Practical aspects of applied optimized survey design for electrical resistivity tomography. *Geophysical Journal International*, Oxford University Press, v. 189, n. 1, p. 428–440, 2012.

XIONG, H. Y.; BARASH, Y.; FREY, B. J. Bayesian prediction of tissue-regulated splicing using rna sequence and cellular context. *Bioinformatics*, Oxford University Press, v. 27, n. 18, p. 2554–2562, 2011.

YANG, X.-S. Firefly algorithm, stochastic test functions and design optimisation. *International Journal Bio-Inspired Computing*, Inderscience Publishers, v. 2, n. 2, p. 78–84, 2010.

YANG, X.-S. A new metaheuristic bat-inspired algorithm. In: *Nature inspired cooperative strategies for optimization (NICSO 2010)*. [S.l.]: Springer, 2010. p. 65–74.

YANG, X.-S. Review of meta-heuristics and generalised evolutionary walk algorithm. *International Journal of Bio-Inspired Computation*, Inderscience Publishers, v. 3, n. 2, p. 77–84, 2011.

YANG, X.-S.; DEB, S. Engineering optimisation by cuckoo search. *International Journal of Mathematical Modelling and Numerical Optimisation*, Inderscience Publishers, v. 1, n. 4, p. 330–343, 2010.

ZHOU, S.; CHEN, Q.; WANG, X. Discriminative deep belief networks for image classification. In: *17th IEEE International Conference on Image Processing*. [S.l.]: IEEE, 2010. p. 1561–1564.