UNIVERSIDADE FEEVALE

ADONIS ANDRADES DIAS

ESTUDO TEÓRICO E PRÁTICO SOBRE DEEP LEARNING PARA A PROPOSIÇÃO DE UM MATERIAL INSTRUCIONAL SOBRE A TECNOLOGIA

Novo Hamburgo

2016

ADONIS ANDRADES DIAS

ESTUDO TEÓRICO E PRÁTICO SOBRE DEEP LEARNING PARA A PROPOSIÇÃO DE UM MATERIAL INSTRUCIONAL SOBRE A TECNOLOGIA

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do grau de Bacharel em Ciência da Computação pela Universidade Feevale

Orientador: Roberto Scheid

Novo Hamburgo

2016

AGRADECIMENTOS

Quero de agradecer a todos os que, de alguma maneira, contribuíram para a realização desse trabalho de conclusão, em especial:

Aos meus pais, João Batista Dias e Ledi Nair Andrades Dias pelo grande apoio que me deram ao longo dos nove anos dedicados a esta graduação. Meus pais sempre se esforçaram muito para garantir que eu concluísse meu curso superior. Sem eles eu não chegaria até aqui.

Também quero agradecer ao meu orientador Roberto Scheid pela ajuda e contribuições a este trabalho.

Muito obrigado a todos.

RESUMO

Nos últimos anos as maiores empresas de tecnologia do mundo, como Microsoft, Facebook e Google, estão investindo em pesquisas sobre novas tecnologias na área de inteligência artificial, como a tecnologia denominada *deep learning*. Deep learning pode ser definido como o conjunto de algoritmos e técnicas de aprendizado de máquina baseados na ideia de treinamento de redes camada por camada, possibilitando o aprendizado em diferentes níveis de representações. Esta tecnologia tem sido empregada em diversos sistemas de inteligência artificial. Em alguns casos os sistemas que se utilizam dessa técnica estão atingindo resultados melhores que os seres humanos, como no reconhecimento de imagens ou em partidas de jogos complexos como Go, que ocorreram em 2016. Desta forma, o presente trabalho propõe-se a investigar as principais técnicas de deep learning, suas aplicações e ferramentas disponíveis para empregá-la. Além disso, conceber a implementação de um software destinado ao ensino-aprendizado de deep learning, para demonstrar na prática o seu potencial.

Palavras-chave: Aprendizado de máquina. Deep learning. Inteligência artificial. Redes neurais.

ABSTRACT

In recent years the world's largest technology companies such as Microsoft, Facebook and Google are investing on researches of new technologies in the field of artificial intelligence, such as the so-called deep learning technology. Deep learning can be defined as the set of algorithms and machine learning techniques based on the network training idea layer by layer, enabling learning at different levels of representation. This technology has been used in many artificial intelligence systems. In some cases the systems that use these techniques are achieving better results than human beings, as the recognition of images or playing complex games like Go, which the matches occurred in 2016. Thus, the present study aims to investigate the main techniques of deep learning, and their applications tools available to use it. In addition, designing an implementation of these techniques for helping the teaching and learning of deep learning and to demonstrate in practice its potential.

Key words: Machine learning. Deep learning. Artificial intelligence. Neural networks.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE GRÁFICOS

LISTA DE ABREVIATURAS E SIGLAS

BP	*Backpropagation*
BPTT	*Backpropagation through time*
CAE	*Contractive autoencoder*
CNN	*Convolutional neural network*
DAE	*Denoising autoencoder*
DBN	*Deep belief network*
DL	*Deep Learning*
DNN	*Deep neural network*
EQM	Erro quadrático médio
GPU	*Graphics processing unit*
IA	Inteligência artificial
LSTM	*Long short term memory*
MIT	*Massachusetts Institute of Technology*
NLP	*Natural language processing*
RBM	*Restricted boltzmann machine*
ReLU	*Rectified Linear Unit*
RNN	*Recurrent neural network*
RNR	Rede neural recursiva
SGD	*Stochastic gradient descent*
SVM	*Support vector machine*
VG	*Vanishing gradiente*

SUMÁRIO

1 INTRODUÇÃO

Estudos sobre as redes neurais artificiais não são recentes, as primeiras referências surgiram por volta de 1943 com o trabalho de Warren McCulloch e Walter Pitts (MCCULLOCH; PITTS apud RUSSEL; NORVIG, 2013), no qual propuseram um modelo de neurônios artificiais, que se definido adequadamente, seria capaz de aprender. Posteriormente, em 1949, Donald Hebb retratou um estudo sobre uma regra de atualização para modificar a intensidade das conexões entre esses neurônios (HEBB apud RUSSEL; NORVIG, 2013). Marvin Minsky e Dean Edmonds (MINSKY; EDMONDS apud RUSSEL; NORVIG, 2013), criaram o primeiro computador de rede neural em 1950 em Harvard, o SNARC.

Na visão de Haykin (2001), uma rede neural é um sistema composto por unidades de processamento chamados neurônios interligados através de conexões sinápticas. Esse tipo de sistema é projetado para modelar o funcionamento do cérebro humano, visto que o conhecimento é adquirido a partir do ambiente e utiliza-se de pesos sinápticos para armazená-lo. Utilizando-se desse modelo, uma rede neural desenvolve algumas características muito importantes para sua empregabilidade. Como, por exemplo, adaptação da rede de acordo com as mudanças do ambiente e na classificação de padrões, sendo que é capaz não só fornecer informações sobre qual padrão selecionar, mas também sobre a confiança na decisão tomada (HAYKIN, 2001).

Com relação à arquitetura de camadas das redes neurais Haykin (2001) frisa dois modelos. Primeiramente as redes de ***camada única***, que possuem uma camada de entrada de neurônios de fonte (camada de entrada) e uma camada de neurônios de saída (nós computacionais). A camada de entrada de neurônios de fonte não é considerada, pois nenhuma computação é efetuada na mesma. Outros modelos de rede existente são as com ***múltiplas camadas***. Essa rede caracteriza-se por possuir uma ou mais camadas ocultas de neurônios e, quando adicionado uma ou mais camadas ocultas na rede, essa passa a possuir o poder de extrair estatísticas de ordem elevada (HAYKIN, 2001). Redes neurais com duas ou mais camadas ocultas são chamadas de ***deep neural networks*** **(DNN)** ou redes neurais profundas (NIELSEN, 2015).

Sob a análise de Bengio (2009), por décadas pesquisadores tentaram treinar redes neurais profundas, porém antes de 2006 não houve relatos de sucesso nessa tarefa. Até então pesquisadores reportaram a utilização com sucesso de redes neurais com duas ou três camadas ocultas. No entanto, o treinamento de redes neurais com mais camadas ocultas forneciam piores resultados. O que pode ser considerado um marco nessa área aconteceu em 2006,

quando Geoffrey Hinton e seus colaboradores apresentaram as *Deep Belief Networks*, redes neurais profundas compostas por uma camada visível e várias ocultas (BENGIO, 2009).

Posteriormente, algoritmos baseados em *autoencoders* foram anunciados (LAMBLIN et al., 2007 apud BENGIO, 2009); (RANZATO et al., 2007 apud BENGIO, 2009), os quais exploram as mesmas ideias divulgadas por Hinton, utilizando-se de aprendizado não supervisionado no treinamento das camadas intermediárias na rede. Outros algoritmos foram criados em 2009 com abordagens diferentes dos algoritmos anteriores (WESTON; RATLE; COLLOBERT, 2008 apud BENGIO, 2009); (MOBAHI; COLLOBERT; WESTON, 2009 apud BENGIO, 2009). Conforme Bengio (2009), este conjunto de algoritmos e técnicas de aprendizado de máquina são conhecidos como *Deep Learning* (DL) e baseiam-se na ideia de treinamento camada por camada da rede, possibilitando o aprendizado em diferentes níveis de representações (BENGIO, 2009).

Em outubro de 2006, Geoffrey Hinton, publicou um artigo para demonstrar algoritmos para a solução de problemas enfrentados por pesquisadores no treinamento de DNNs. Como exemplo desses problemas podem ser citados a inequação das *support vector machines* (SVM) no reconhecimento de objetos 3D (LECUN et al., 2004, apud HINTON, 2006) e na utilização do algoritmo *backpropagation* (BP) (HINTON, 2006).

De acordo com Geoffrey Hinton (2006), no algoritmo BP, o primeiro algoritmo a ser eficiente no ensinamento de redes neurais com mais de uma camada, existe o problema da necessidade de estabelecer valores aleatórios para os pesos sinápticos. Uma tarefa complexa, pois os valores dos pesos não podem ser baixos, pelo fato dos gradientes diminuírem na retropropagação pelas camadas ocultas e nem altos demais, ocasionando a seleção aleatória de uma região particular da rede. Outro problema desse algoritmo é que para grandes redes neurais é necessária uma quantidade significativa de dados classificados para o treinamento (HINTON, 2006).

Portanto, Geoffrey Hinton (2006) descreve uma nova estratégia para ser integrada nos treinamentos de DNNs. A estratégia baseia-se na realização de um pré-treinamento não supervisionado camada por camada da rede, fazendo com que as unidades ocultas de cada camada transmitem regras para a camada inferior a atual (HINTON, 2006).

Após as descobertas do grupo de pesquisadores do *Canadian Institute for Advanced Research*, o interesse nas pesquisas sobre deep neural networks foram renovados, possibilitando vários avanços no campo da inteligência artificial (LECUN; BENGIO; HINTON, 2015). A seguir serão relatados 4 projetos que empregam DL para melhorar o desempenho de sistemas de inteligência artificial.

1. Pesquisadores da empresa Deep Mind, adquirida pelo Google, em artigo publicado na revista de ciência Nature em 28 de janeiro de 2016, anunciaram o primeiro sistema computacional GO, chamado AlphaGo, a vencer um jogador profissional. O jogo Go é considerado o mais desafiador para programas de inteligência artificial, contendo 250^{150} possibilidades de movimentos em comparação ao xadrez que possui cerca de 35^{80}. O programa AlphaGo utiliza redes neurais profundas combinadas para aprimorar a política de seleção de ações no jogo, bem como para prever o resultado de cada jogada. Também foi introduzida no sistema uma *Monte Carlo Tree Search* para ler sequências de jogadas futuras, guiada pelas redes neurais para reduzir seu espaço de busca (SILVER et al., 2016).
2. Com o propósito de utilizar processamento de linguagem natural para criar uma máquina capaz de ler, entender o contexto do que foi lido e responder a perguntas sobre o texto processado, pesquisadores do Google empregaram deep learning em uma base de dados supervisionada gerada especificamente para o treinamento desta rede. O sistema é composto por três redes neurais do tipo *long short term memory*, a primeira faz leitura dos documentos e das perguntas, a segunda rede realiza buscas de dependências para assimilar as respostas às perguntas. A terceira rede tem o objetivo de reler os documentos a cada *token* identificado na pergunta, para regularmente acumular informações sobre a mesma (HERMANN et al., 2015).
3. Outro projeto desenvolvido pelos pesquisadores do Google visa o processamento de imagens. Uma das vantagens de modelagem generativa (processo de reconstrução da melhor maneira possível de uma imagem) é que existe uma vasta quantidade de dados disponíveis dos quais se pode aprender. Contudo no caso das imagens, por serem altamente dimensionáveis e estruturadas, estimar a sua distribuição é extremamente desafiador. Neste projeto os pesquisadores elaboraram um sistema com duas *deep recurrent neural networks* para aplicá-lo na modelagem de larga escala de imagens naturais. O objetivo principal do projeto é estimar uma distribuição sobre imagens naturais ou a geração de novas (OORD; KALCHBRENNER; KAVUKCUOGLU, 2016).
4. Além do Google, a Microsoft também está investindo em DL. Recentemente, a Microsoft venceu seu rival Google na disputa pelo melhor resultado na competição ImageNet de reconhecimento de imagens. Na competição, o software desenvolvido pela Microsoft atingiu a taxa de erro de 3,57%, abaixo da estimada para humanos de 5,1% (RUSSAKOVSKY et al, 2015). Para atingir essa marca no teste, uma nova

abordagem de DL foi empregada, baseada em *residual networks*. Nessa arquitetura de rede, existem conexões de atalhos entre as camadas, paralelas com as conexões padrões, possibilitando o treinamento de redes com uma profundidade maior de camadas (HE et al., 2015).

Atualmente, as pesquisas sobre DL e suas possíveis aplicações são amplas, igualmente no que diz respeito às tecnologias desenvolvidas para a sua implementação. Isso se deve ao fato de que, além de abranger técnicas de computação relativamente recentes, estão sendo realizadas diversas pesquisas e novas descobertas nesta área. Essas pesquisas são realizadas por equipes das maiores empresas de tecnologia do mundo, como Microsoft, Facebook e Google.

Com base nas informações que foram apresentadas, surge a problemática da pesquisa: **com relação ao conceito de Deep Learning, como implementá-lo para o desenvolvimento de uma ferramenta voltada ao ensino-aprendizado desta tecnologia?**

Inerente à questão introduzida acima, o presente trabalho propõe-se a investigar as principais técnicas de DL e as suas atuais e possíveis aplicações. Não obstante, explorar as ferramentas disponíveis para sua implementação a fim de gerar conhecimento sobre o assunto e tê-lo como base para futuros estudos. Ademais, neste estudo será sugerido uma modelagem de implementação de um software destinado ao ensino-aprendizado, para demonstrar na prática o potencial do Deep Learning.

1.1 OBJETIVOS

O principal objetivo do trabalho é i**nvestigar como o conceito de aprendizado de máquina, deep learning, pode ser empregado como ferramenta para auxílio no ensino-aprendizado da tecnologia.**

Objetivos específicos

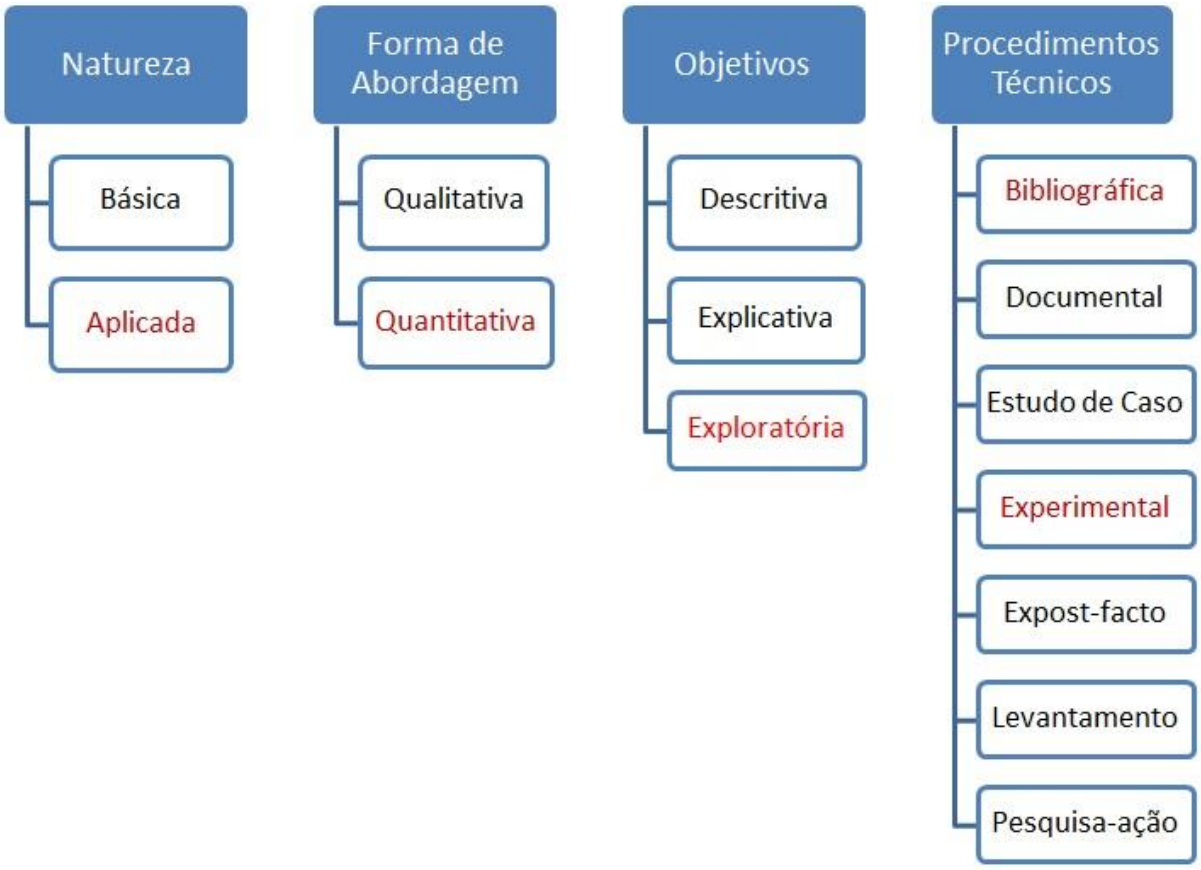
- Realizar um estudo teórico sobre os principais métodos de aprendizado de máquina com base no conceito de deep learning;
- Pesquisar sobre as atuais e possíveis aplicações de deep learning;
- Pesquisar as ferramentas disponíveis para a implementação de deep learning;

- Apresentar uma implementação do conceito de deep learning para auxiliar o ensino-aprendizado da tecnologia;
- Realizar simulações com a implementação do conceito de deep learning com o modelo proposto.

1.2 METODOLOGIA

A figura abaixo ilustra a metodologia utilizada e o tipo de pesquisa que embasa este trabalho, onde os quadros destacados em vermelho indicam os procedimentos utilizados.

Figura 1 – Classificação da pesquisa



Fonte: Adaptado de Bez (2011).

A partir dos conceitos metodológicos expostos por Prodanov e Freitas (2013). A natureza deste trabalho caracteriza-se como uma pesquisa aplicada, onde será empregado o conhecimento adquirido para a implementação de um novo conceito de inteligência artificial, utilizado para solucionar problemas específicos.

O objetivo do estudo é exploratório, para que se possa gerar conhecimento sobre um novo conceito de inteligência artificial e com isso auxiliar o desenvolvimento de futuros

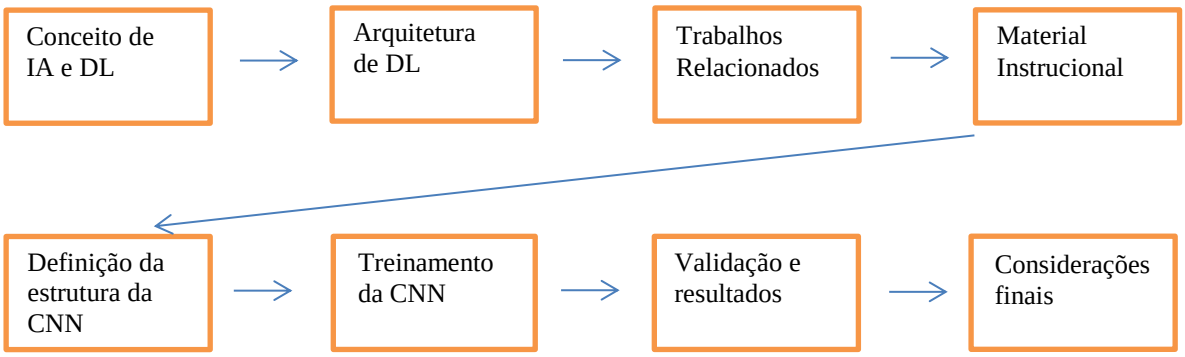
trabalhos na área. Segundo Prodanov e Freitas (2013) “tem como finalidade proporcionar mais informações sobre o assunto que vamos investigar, possibilitando sua definição e seu delineamento, isto é, facilitar a delimitação do tema da pesquisa; orientar a fixação dos objetivos e a formulação das hipóteses ou descobrir um novo tipo de enfoque para o assunto”.

Por fim, seguindo os conceitos de Prodanov e Freitas (2013), quanto aos procedimentos técnicos, o presente trabalho compreenderá uma pesquisa bibliográfica, que significa a busca do referencial teórico necessário para o estudo, utilizando-se para isso livros, artigos científicos, teses e outros trabalhos de conclusão de graduação. Além disso, este trabalho trata-se de uma pesquisa experimental, visto que será realizada uma modelagem de implementação do conceito de deep learning.

1.3 ESTRUTURA DO TRABALHO

O trabalho está organizado em quatro capítulos. O capítulo 2 faz uma breve introdução ao campo de estudo de inteligência artificial. O capítulo 3 é dedicado à área da inteligência artificial chamada DL, onde é apresentada sua definição bem como os conceitos que a envolvem. O capítulo 4 abrange três trabalhos relacionados ao ensino de deep learning. Nos capítulos 5 e 6 será retratado o material instrucional proposto e como foi executada sua validação. Por fim, o capítulo 7 apresentará as considerações finais. A figura 2 exibe um fluxograma com estas etapas a serem seguidas.

Figura 2 – Fluxograma da pesquisa



Fonte: Elaborado pelo autor (2016).

2 INTELIGÊNCIA ARTIFICIAL

Russel e Norvig (2013) abrangem a inteligência artificial (IA) como um dos campos mais novos dentro da ciência e engenharia. As pesquisas nessa área tiveram início após a segunda guerra mundial, sendo ainda tomada pelos pesquisadores com uma das áreas da ciência mais interessantes para se trabalhar. Incorporando diversas subáreas a inteligência artificial possui aplicações (RUSSEL; NORVIG, 2013):

- Representação do conhecimento: como armazenar e o que armazenar no sistema sobre aspectos do mundo real;
- Raciocínio automático: como utilizar o conhecimento armazenado para obter conclusões ou respostas;
- Visão computacional para perceber e compreender objetos. Robótica para manipular objetos;
- Processamento de linguagem natural: para um sistema poder adquirir conhecimento sobre algo escrito ou falado em linguagem natural, por exemplo, inglês ou português, é preciso que o sistema consiga interpretar a ambiguidade e línguas confusas que os humanos utilizam.

De acordo com Battiti e Brunato (2013) a área de IA denominada aprendizado de máquina, tem o objetivo de construir um sistema com a capacidade de generalizar dados novos e inéditos. Do contrário o sistema não está aprendendo, ele está simplesmente memorizando um conjunto de padrões conhecidos.

Barnes (2015) destaca exemplos de tendências das indústrias que estão impulsionando a evolução das técnicas de aprendizado de máquina, como, por exemplo:

- Crescimento de dados exponencial: disponibilidade grandes quantidades de dados sobre transações históricas altamente valiosas, na maioria delas digitalizadas e acessíveis para leitura. Além disso, surge uma quantidade crescente de dados em tempo real gerados através de sistemas embarcados e da internet das coisas;
- Armazenamento digital barato: possibilidade de acesso a vários mecanismos de armazenamento, de dispositivos pessoais a privados ou em nuvens públicas;
- Poder de computação ubíqua: servidores de computação em nuvem estão em toda a parte e facilmente disponíveis. O seu acesso é simples, basta um cartão de crédito e um browser para conseguir iniciar, pagando por hora ou minuto pelo poder de processamento necessário.

Um exemplo de técnica de aprendizado de máquina difundida no mundo acadêmico e comercial é a rede neural artificial. Uma rede neural pode ser definida de uma forma geral como uma máquina criada para modelar a maneira com que o cérebro executa suas tarefas (HAYKIN, 2008).

David e Schwartz (2014) referem-se ao aprendizado com uma interação entre o aprendiz e o ambiente e destacam três diferentes tipos de aprendizado: aprendizado supervisionado; aprendizado não supervisionado e aprendizado por reforço. Para exemplificar o aprendizado supervisionado e o não supervisionado os autores apresentam o processo de detecção de [1]***spams*** em e-mails em comparação com a tarefa de detecção de anomalias nas mensagens. Na tarefa de detecção de spams, é considerado um cenário no qual o aprendiz (sistema) recebe e-mails para o seu treinamento, para o qual o [2]label ou classificação "spam" e "não spam" é disponibilizado. Com os labels apresentados ao aprendiz, o mesmo deve procurar descobrir regras para classificar corretamente novas mensagens de e-mails recebidas. Já na tarefa para detectar anomalias nas mensagens de e-mail, o aprendiz recebe apenas uma grande quantia de e-mails para treinamento, todavia sem nenhum label associado a elas. Neste caso seu objetivo é encontrar "anomalias" nas mensagens (DAVID; SCHWARTZ, 2014).

Observando o aprendizado, David e Schwartz apontam o seu processo como "utilização de experiência para adquirir experiência". No caso do aprendizado supervisionado, é definido um cenário em que o exemplo de treinamento contém uma quantia significativa de dados (como os *labels* spam/não spam), que não possuem os dados referentes aos exemplos de teste, esses dados separados são utilizados para o teste do aprendizado gerado. Sendo assim é possível pensar que o ambiente emprega o papel de professor que "supervisiona" o aprendiz ao fornecer os labels como uma informação extra, além dos dados a serem classificados. Por outro lado no aprendizado não supervisionado, não existe diferença entre as informações para treinamento ou teste. O aprendiz precisa processar os dados fornecidos e sozinho gerar um resumo das informações ou se for possível já realizar uma pré-classificação (DAVID; SCHWARTZ, 2014).

Com relação ao aprendizado por reforço, David e Schwartz (2014) tratam-no como um método intermediário, com base na aprendizagem supervisionada e não supervisionada. Nesse modelo de aprendizagem são concedidas mais informações para treinamento do que para teste

[1] Spam – é o termo usado para referir-se aos e-mails não solicitados, que geralmente são enviados para um grande número de pessoas. Quando o conteúdo é exclusivamente comercial, esse tipo de mensagem é chamada de UCE (do inglês Unsolicited Commercial E-mail).

[2] Label – Legenda associada à uma informação com o objetivo de classifica-la.

do modelo. Com isso surge a necessidade do aprendiz prever ainda mais informações para os exemplos disponibilizados para teste.

Como exemplo do processo de aprendizado por reforço de um sistema de IA, pode ser utilizado o jogo de xadrez. Para aprender e gerar uma função que indique para cada configuração do tabuleiro de xadrez o quanto as posições brancas são melhores que as posições pretas, a única informação disponibilizada para o aprendiz são as posições das peças no tabuleiro. Estas posições são de jogos passados, classificados pelos ganhadores das partidas (DAVID; SCHWARTZ, 2014).

O objetivo do aprendizado por reforço segundo Haykin (2008) é minimizar o custo da função, não simplesmente considerando o seu custo imediato (uma jogada na partida de xadrez), mas determinada pela expectativa da soma dos custos das ações que serão tomadas sobre uma sequência de passos. Ou seja, ações que foram tomadas anteriormente naquela sequência de passos, compõem o conjunto determinante do comportamento do sistema no geral (HAYKIN, 2008).

Seguindo o tema da IA, o próximo capitulo irá trabalhar o a área de DL, sua definição e seus diferentes tipos de arquiteturas de redes.

3 DEEP LEARNING

Na literatura existem várias definições para DL, que também é chamado de ***deep structured learning*** ou ***hierarchical learning*** (muitos termos não serão traduzidos para o português, pois são raros os trabalhos publicados na língua portuguesa, dificultando futuras pesquisas baseadas nesse tema), portanto, antes de descrever as tecnologias em maior detalhe, serão apresentadas algumas definições de DL que foram compiladas por Deng e Yu (2014):

> Definition 1: A class of machine learning techniques that exploit many layers of non-linear information processing for 200 Introduction supervised or unsupervised feature extraction and transformation, and for pattern analysis and classification.
>
> Definition 2: "A sub-field within machine learning that is based on algorithms for learning multiple levels of representation in order to model complex relationships among data. Higher-level features and concepts are thus defined in terms of lower-level ones, and such a hierarchy of features is called a deep architecture. Most of these models are based on unsupervised learning of representations." (Wikipedia on "Deep Learning" around March 2012.)
>
> Definition 3: "A sub-field of machine learning that is based on learning several levels of representations, corresponding to a hierarchy of features or factors or concepts, where higher-level concepts are defined from lower-level ones, and the same lower-level concepts can help to define many higher-level concepts. Deep learning is part of a broader family of machine learning methods based on learning representations. An observation (e.g., an image) can be represented in many ways (e.g., a vector of pixels), but some representations make it easier to learn tasks of interest (e.g., is this the image of a human face?) from examples, and research in this area attempts to define what makes better representations and how to learn them." (Wikipedia on "Deep Learning" around February 2013.)
>
> Definition 4: "Deep learning is a set of algorithms in machine learning that attempt to learn in multiple levels, corresponding to different levels of abstraction. It typically uses artificial neural networks. The levels in these learned statistical models correspond to distinct levels of concepts, where higher-level concepts are defined from lower-level ones, and the same lower-level concepts can help to define many higher-level concepts." See Wikipedia http://en.wikipedia.org/wiki/Deep_learning on "Deep Learning" as of this most recent update in October 2013.
>
> Definition 5: "Deep Learning is a new area of Machine Learning research, which has been introduced with the objective of moving Machine Learning closer to one of its original goals: Artificial Intelligence. Deep Learning is about learning multiple levels of representation and abstraction that help to make sense of data such as images, sound, and text. See https://github.com/lisa-lab/DeepLearningTutorials. (DENG; YU, 2014, p.199)

Bengio também define deep learning como "... methods aim at learning feature hierarchies with features from higher levels of the hierarchy formed by the composition of lower level features…" (BENGIO, 2009. p.5).

Deng e Yu (2014) ressaltam, entre as definições apresentadas, os aspectos comuns encontrados nas suas descrições: 1) modelos consistentes de múltiplas camadas ou estágios de processamento de informação não linear; 2) métodos para aprendizado supervisionado e não supervisionado de representações de características em camadas mais altas e mais abstratas,

respectivamente. DL envolve muitas áreas de estudos diferentes, tais como: redes neurais, inteligência artificial, modelagem gráfica, otimização, reconhecimento de padrões e processamento de sinais (DENG; YU, 2014).

Uma questão que pode vir em mente é: quão profunda essas redes neurais devem ser? Para responder isso, Nielsen (2015) destaca que o verdadeiro avanço na aprendizagem em redes neurais profundas, foi ir além das redes com apenas uma ou duas camadas ocultas, arquitetura que dominou até a metade dos anos 2000. Entretanto o número de camadas ocultas não é o principal objetivo e sim a utilização das DNNs como uma ferramenta para atingir outros objetivos (NIELSEN, 2015).

Após esta breve definição do conceito de DL, os capítulos seguintes visam uma abordagem mais detalhada, iniciando por uma contextualização histórica, posteriormente explicando o que se refere DL, os modelos existentes para sua construção e suas possíveis aplicações.

3.1 CONTEXTO HISTÓRICO SOBRE DEEP LEARNING

Sob a análise de Deng e Yu (2014), as arquiteturas com até duas camadas de transformação não lineares, como por exemplo, modelos gaussianos, sistemas dinâmicos lineares/não lineares, máquinas de vetor de suporte (SVM) e *perceptrons* multicamadas com uma camada oculta, foram utilizadas como técnicas de aprendizado de máquina. Não obstante, essas técnicas convencionais eram limitadas em sua habilidade para processar dados naturais em sua forma bruta (DENG; YU, 2014). Por esse motivo, a construção de sistemas para reconhecimento de padrões ou aprendizado de máquina precisava de uma engenharia cuidadosa. Também existia a necessidade de um conhecimento avançado para modelar o sistema que transformaria os dados brutos em uma representação de onde outro subsistema, ou classificador, pudesse detectar e classificar os padrões (LECUN; BENGIO; HINTON, 2015).

Historicamente, o conceito de deep learning originou-se da pesquisa sobre redes neurais artificiais (perceptrons de múltiplas camadas com várias camadas ocultas). O algoritmo de aprendizado utilizado nessas redes mais conhecido é o BP, popularizado nos anos 80. Infelizmente BP sozinho não funcionava adequadamente na prática para redes com um número maior de camadas ocultas (DENG; YU, 2014).

Conforme matéria publicada na revista *Nature*, a dificuldade no treinamento de redes profundas foi mitigada quando um eficiente algoritmo de aprendizagem não supervisionada

foi apresentado em dois artigos ("*A fast learning algorithm for deep belief nets*", G. Hinton, S. Osindero, e Y. Teh; "*Reducing the dimensionality of data with neural networks*". G. Hinton and R. Salakhutdinov) em 2006 por um grupo de pesquisadores do *Canadian Institute for Advanced Research* (LECUN; BENGIO; HINTON, 2015). Esses pesquisadores apresentaram algoritmos capazes de detectar características sem precisar de dados classificados com *labels*.

No entendimento de Deng e Yu (2014), o objetivo em fazer com que o algoritmo treine cada camada como um detector de características era poder torná-lo hábil em reconstruir suas atividades em camadas inferiores. Desse modo, através de um pré-treinamento de várias camadas de detectores de características, progressivamente mais complexas, os pesos de uma DNN poderiam ser inicializados com dados sensíveis. Uma camada final de saída poderia então ser incluída na rede e o modelo conseguiria utilizar o algoritmo de BP para o seu treinamento. Essa técnica funcionou perfeitamente para o reconhecimento de imagens de dígitos manuscritos e detecção de pedestres, mesmo quando uma quantidade limitada de dados estava disponível. Nos artigos citados foi introduzida uma classe de modelos generativos profundos chamadas de deep belief network (DBN), compostas por *restricted boltzmann machines* (RBM) empilhadas (DENG; YU, 2014).

Deng e Yu (2014) também revelam que as primeiras aplicações dessa técnica de pré-treinamento foi no reconhecimento de rostos humanos, o qual só foi possível pelo avanço das unidades gráficas de processamento (GPUs), que permitiram aos pesquisadores treinar redes vinte vezes mais rápido. Em 2009 esta abordagem também foi utilizada para mapear janelas temporais curtas, extraídas a partir de ondas de som, atingindo resultados recordes em um *benchmark* de reconhecimento de fala que possui um vocabulário vasto de palavras. Em 2012, versões desse modelo de rede profunda criado em 2009 foram desenvolvidas pelos principais grupos de pesquisas no reconhecimento de falas e incluídas em smartphones Android (DENG; YU, 2014).

Algumas razões pelas quais DL popularizou-se nos últimos anos são frisadas por Deng e Yu (2014): aumento das habilidades de chips de processamento (exemplo, [3]GPUs), o crescimento de dados utilizados para treinamento e os recentes avanços nas pesquisas em aprendizado de máquina. Viabilizando aplicar DL em complexas funções capazes de aprender representações distribuídas e hierárquicas de dados com ou sem labels. Instituições de pesquisas ativas nessa área incluem: Universidade de Toronto, Universidade de Montreal, Universidade de Stanford, Microsoft Research (desde 2009), Google (desde 2011), IBM

[3] GPU – Unidade gráfica de processamento

Research (desde 2011), Baidu (desde 2012), Facebook (desde 2013) e Massachusetts Institute of Technology (MIT) (DENG; YU, 2014).

Pesquisadores destas instituições têm apresentado aplicações de DL em muitas áreas como: visão computacional, reconhecimento de fala, classificação, compreensão da linguagem natural, reconhecimento de escrita, processamento de áudio, robótica e até mesmo na análise de moléculas que podem levar à descoberta de novas drogas. Além Disso, pesquisas para filtragem de conteúdo em redes sociais, recomendações em sites de e-commerce, seleção de resultados relevantes para pesquisas (buscadores como Google e Bing) e previsão dos efeitos das mutações no DNA não codificado (LECUN; BENGIO; HINTON, 2015)

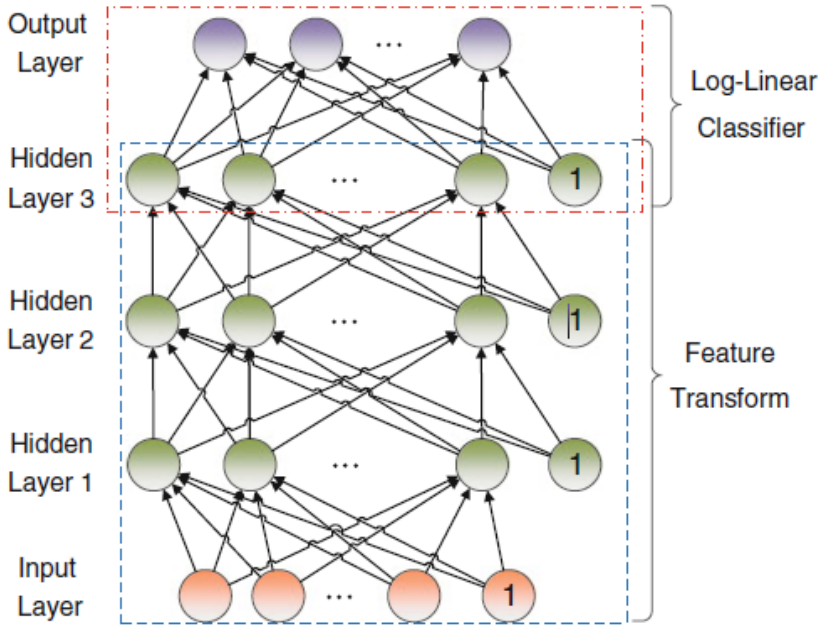
3.2 APRENDIZADO EM VÁRIOS NÍVEIS DE ABSTRAÇÃO

Como definido no início desse capítulo, as técnicas baseadas em DL visam o treinamento de redes camada por camada para geração de diversos níveis de abstração na rede. Essa seção visa explicar o que isso significa.

Arquiteturas de redes rasas (com poucas camadas), tal como revelam Deng e Yu (2014) em sua pesquisa, tem se mostrado eficazes na solução de muitos problemas simples, todavia, são modelos com poder de representação limitados. Por isso, geram dificuldades quando são empregados em tarefas mais complicadas como processamento da fala humana ou visão de cenas reais. Estas tarefas necessitam de arquiteturas profundas para extrair sua complexa estrutura e construir representações de sua rica quantidade de características. Por exemplo, os sistemas de fala e percepção humana estão ambos equipados com estruturas hierárquicas em camadas para transformar a informação a partir da forma de onda para o nível da linguística (DENG; YU, 2014).

Sob a ótica de Deng e Yu (2015), as DNNs automaticamente aprendem e criam as hierarquias de representações e classificadores de maneira conjunta, sendo assim não necessitam de alterações manuais para desempenhar essas tarefas. A combinação das camadas de neurônios ocultos pode ser considerada como um módulo de aprendizagem de características dentro da arquitetura da rede. Apesar das camadas serem compostas por estruturas que desempenham simples tarefas não lineares, a combinação de todas essas estruturas gera uma capacidade de transformações complexas. A última camada de uma DNN é geralmente utilizada como um classificador (figura 3) (DENG; YU, 2015).

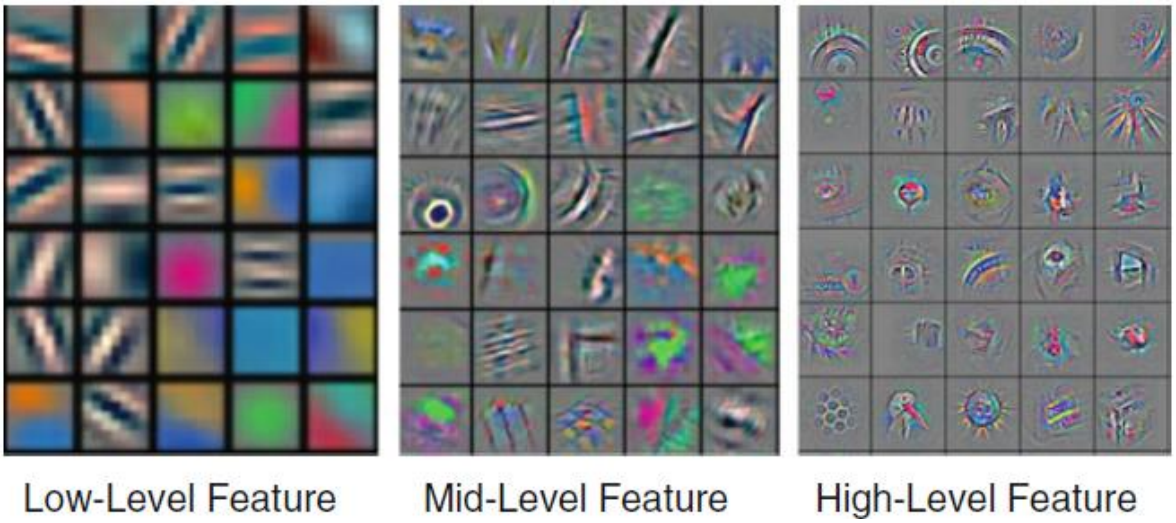
Figura 3 – Representação das camadas de uma DNN para a transformação e classificação de informações



Fonte: Deng; Yu (2015).

Como já citado anteriormente, as DNNs podem aprender hierarquias de características. Nesse sentido, Deng e Yu (2015) explicam que as camadas ocultas que estão posicionadas mais próximas à camada de entrada aprendem características de níveis mais básicos. Fazem isto identificando características locais dos dados e são mais sensíveis a mudanças nos parâmetros de entrada, como por exemplo, em uma imagem, as bordas e arestas. As camadas que estão mais próximas à camada de saída da rede neural representam características de mais alto nível, ou seja, elas são construídas e baseadas nas camadas mais baixas, com isso são mais abstratas e constantes com relação a alterações nos parâmetros de entrada, ilustradas na figura 4 (DENG, YU, 2015).

Figura 4 – Representação de múltiplos níveis de abstração de uma DNN



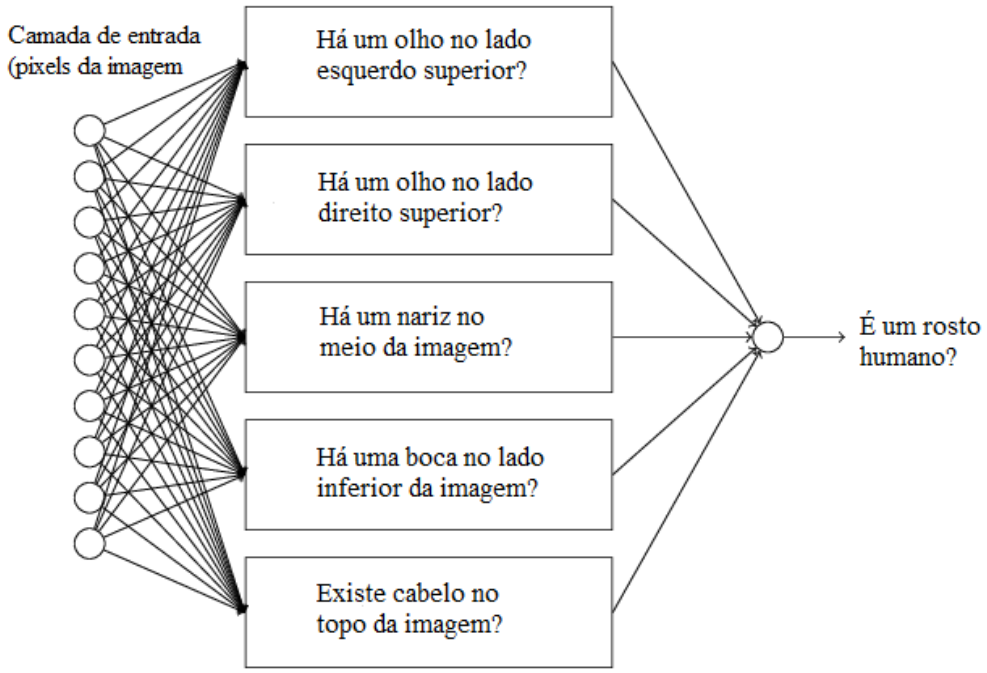
Fonte: Deng; Yu (2015).

Para facilitar o entendimento do aprendizado em diversos níveis de uma DNN, Nielsen (2015) exemplifica o problema através de um caso do mundo real de uma forma fácil de compreender a ideia. Será utilizada como exemplo a missão de identificar, dentre imagens apresentadas, qual mostra o rosto de um ser humano, através da aplicação de uma rede neural. Os pixels das imagens serão usados como entrada para a rede neural, a saída da rede possuirá um único neurônio para indicar "Sim, é um rosto" ou "Não, não é um rosto".

Neste exemplo não será utilizado um algoritmo de aprendizagem, como o BP, o modelo da rede será desenvolvido manualmente. Isso pode ser feito decompondo o problema em subproblemas: a imagem tem um olho no canto superior esquerdo? Será que ela tem um olho no canto superior direito? Será que ela tem um nariz no meio? Será que ela tem uma boca no meio? Existe cabelo no topo? E assim por diante (NIELSEN, 2015).

Se as respostas dessas perguntas forem “sim”, então pode concluir-se que a imagem provavelmente possui um rosto humano. Do contrário, se as respostas forem “não”, então a imagem não possui um rosto. Essa abordagem contêm muitas deficiências, pelo fato de ser somente exemplificação. Talvez a pessoa seja careca, talvez somente parte do rosto possa ser visto na imagem, ou o rosto está em um ângulo diferente. A figura a seguir ilustra como seria essa arquitetura de rede neural para identificação de rostos humanos, os retângulos são as sub-redes (NIELSEN, 2015):

Figura 5 – DNN fictícia para identificar imagens de rostos



Fonte: Adaptado de Nielsen (2015).

As sub-redes (na figura 5 representadas por retângulos) ainda podem ser quebradas em outras sub-redes para responder perguntas ainda mais específicas sobre a imagem. A pergunta “Há um olho no lado esquerdo superior” pode ser quebrada em outras como: "Existe uma sobrancelha?"; "Existem cílios?"; "Existe uma íris?"; "É a sobrancelha no canto superior esquerdo, e acima da íris?" (NIELSEN, 2015).

Nielsen (2015) complementa explicando que, se decomposição desse modelo continuasse as sub-redes iriam responder a perguntas tão simples que poderiam ser respondidas no nível de apenas um pixel da imagem, sobre a presença ou não de formas muito simples. O resultado é uma rede neural que consegue desmembrar uma complexa tarefa, capaz de identificar, através de várias camadas de neurônios, se a imagem apresenta ou não um rosto de uma pessoa. No qual as camadas iniciais respondem perguntas muito simples sobre a imagem, já as camadas finais responsabilizam-se por conceitos mais complexos e abstratos (NIELSEN, 2015).

Obviamente, na prática não é realizada a decomposição manual das camadas responsáveis por determinadas características em redes neurais profundas. São utilizados algoritmos de aprendizagem para gerar automaticamente os pesos e biases na rede e, por conseguinte, a criação da hierarquia de conceitos. Esses algoritmos de aprendizagem

baseados, por exemplo, no BP e no *stochastic gradient descent* (SGD) serão detalhados nos próximos subcapítulos (NIELSEN, 2015).

3.3 DEEP LEARNING - ARQUITETURA

Nessa seção serão apresentados os principais modelos de DNNs existentes: DBNs, redes neurais convolucionais, autoencoders, redes neurais recorrentes (RNN) e redes neurais recursivas (RNR). Bem como os componentes e métodos que envolvem o treinamento dessas redes. Os tipos de DNNs estudados serão divididos em três categorias: 1) para aprendizado não supervisionado; 2) para aprendizado supervisionado; 3) um modelo híbrido.

Para encontrar padrões em dados não classificados e extração de características, pode ser utilizada uma RBM ou um autoencoder. Se os dados são classificados e se quiser utilizar aprendizado supervisionado, dependendo da aplicação, podem ser utilizadas RNNs (para processamento de texto), DBN ou CNN (para reconhecimento de imagens), redes neurais recursivas (RNR) ou CNN (reconhecimento de objetos). Para análise de séries temporais é mais recomendado a utilização de RNNs.

3.3.1 CONCEITOS GERAIS

No decorrer da pesquisa para formulação deste trabalho, muitos conceitos diferentes sobre DL e redes neurais foram encontrados nas referências bibliográficas. Com o objetivo de centralizar e facilitar a procura pela definição desses conceitos foi elaborada a tabela 1 com a descrição do conceito, uma explicação do mesmo e sua fonte. Mesmo que o conceito seja básico, muitas vezes surgem dúvidas quanto a sua definição.

Tabela 1 – Conceitos gerais sobre redes neurais e deep learning

Conceito	Descrição	Autores
Perceptron	Tipo de neurônio artificial que recebe várias entradas binárias (x1, x2...) e produz uma única saída binária (0 ou 1). Para calcular a saída do neurônio perceptron são introduzidos pesos, valores reais expressando a importância das entradas para o cálculo do valor de saída. As saídas possíveis (0 ou 1) são determinadas se a soma dos pesos é menor ou maior do que o valor do *threshold* (bias) especificado. Atualmente é mais comum a utilização de outros tipos de neurônios.	Haykin (2008)
Sigmoid Function	É um tipo função de ativação de neurônio. A função sigmoid tem o objetivo de garantir que os valores de saída do neurônio permaneçam em uma pequena variação (HEATON, 2015). No caso do neurônio perceptron, quando realizada alguma alteração nos seus pesos ou em seu bias, pode ocorrer uma mudança brusca no resultado de sua saída e, consequentemente, alterar o comportamento de todo o restante da rede neural. Esse comportamento torna mais difícil o processo de alteração dos pesos e bias para que a rede consiga atingir o valor esperado para a sua	Nielsen (2015)

	saída. A função sigmoid possui um gráfico mais suave, gerando uma saída com qualquer valor entre 0 ou 1. Isso quer dizer que pequenas alterações nos pesos ou bias do neurônio, geram uma pequena alteração no resultado da função, facilitando o treinamento e otimizando a rede neural.	
Função tangente hiperbólica	Função de ativação de neurônio muito comum em redes neurais. A saída gerada encontra-se entre o intervalo de -1 a 1 e possui um gráfico com formato semelhante ao gráfico da função sigmoid. Possui algumas vantagens sobre a função sigmoid o que faz dela mais recomenda.	Heaton (2015)
***Rectified Linear Unit* (ReLU):**	Função de ativação de neurônio. Antes do surgimento da função ReLU, a tangente hiperbólica era a opção mais comum. Contudo atualmente, os pesquisadores tem recomendado a utilização da função de ativação ReLU por ter demonstrado resultados superiores no treinamento de redes neurais. O desempenho superior da função ReLU está relacionado ao fato da função ser linear e não saturar. Ao contrário das funções comentadas anteriormente, a ReLU não possui saturação nos valores próximos a -1, 0 ou 1. Já a função tangente hiperbólica, por exemplo, estabiliza seu resultado próximo a 1 quando x aumenta e estabiliza em -1 quando x diminui.	Heaton (2015)
Algoritmo de treinamento	A finalidade do algoritmo é encontrar pesos e biases para que a saída gerada pela rede neural (resultado) seja o mais próxima possível do valor esperado para todos os dados utilizados com treinamento, desse modo gerando aprendizado sobre os dados.	Nielsen (2015)
Função de custo	Quantifica o desempenho do algoritmo de treinamento em relação ao seu objetivo, ou seja, o algoritmo de treinamento está atingindo um bom desempenho se os pesos e biases que ele encontra fazem com que a função de custo tenda a zero. A função de custo quadrática, também chamada de erro quadrático médio (EQM), é um exemplo de função de custo e representa a média entre a resposta desejada da rede e resposta atual levando em consideração todos os exemplos utilizados no treinamento. Entretanto, nessa função existe um problema quando se fazem uso de pesos e biases com valores elevados no treinamento. Nesse caso, a função demora a convergir para o valor esperado na saída do neurônio [1]. Outra função de custo que soluciona esse problema é a *cross-entropy***,** indicada como melhor opção [2].	[1] Haykin, (2008) [2] Heaton, (2015)
Gradiente	Uma função matemática que mede a taxa de variação da função de custo em relação a uma mudança nos pesos e bias do neurônio. Essa taxa de variação, calculada pela derivada da curva da função de erro, fornece a informação necessária para ajustar os pesos e biases e diminuir o custo.	Heaton (2015)
Loss function	DNNs do tipo *autoencoder* possuem uma métrica diferente com relação ao erro no treinamento, chamada função de perda. Do contrário da métrica de custo, a função de perda mede a quantidade de informação que foi perdida na tentativa de reconstrução do dado de entrada da rede.	Goodfellow, Bengio e Courville, (2016)
Algoritmo de treinamento *Backpropagation*	É um dos métodos mais comuns para o treinamento de redes neurais. O seu algoritmo aprende processando os dados de treinamento em dois sentidos na rede. Na propagação o valor gerado pela rede é comparado com o valor desejado. No caminho inverso, ou na retropagação, os pesos são ajustados para minimizar a função de custo, desde a camada de saída até a primeira camada oculta da rede [1]. O algoritmo de retropropagação faz uso do cálculo do gradiente em relação à função de custo. Pelo fato da rede neural não gerar o valor esperado para o valor de treinamento, o gradiente fornecerá uma indicação de como alterar cada peso para atingir o valor desejado pela rede [2].	[1] Han, Kamper e Pei (2012) [2] Heaton, (2015)
***Stochastic gradient descent* (SGD)**	É uma variação do algoritmo de retropropagação e é utilizado para acelerar o aprendizado da rede neural. O funcionamento do SGD decorre através da estimativa do gradiente pelo processamento de uma pequena amostra, selecionada aleatoriamente, dos dados de treinamento. Pelo intermédio dessa pequena amostra é possível conseguir rapidamente uma boa estimativa do gradiente, fazendo com que isso acelere o treinamento. Essa pequena amostra de dados de treinamento é chamada *mini-batch*. Existem outras abordagens para o SGD, como por exemplo, a técnica	Deng e Yu (2015)

	Hessian e o *Momentum-based gradient descent*.	
Softmax	Função de ativação utilizada como uma alternativa para o problema na lentidão no processo de treinamento ocasionado pela função de custo. Softmax é geralmente empregada na camada de saída da rede neural destinada à classificação. A função de ativação softmax força a saída da rede neural para indicar a probabilidade da entrada enquadrar-se em uma das classes especificadas. Ou seja, a saída de uma camada softmax pode ser considerada uma distribuição probabilística. Na camada de saída deve existir um neurônio para cada classe possível a ser classificada, desse modo o neurônio com o maior valor é considerado como resultado final.	Heaton (2015)
Taxa de aprendizagem	Tem o propósito de dimensionar o gradiente e também pode diminuir ou acelerar a velocidade do treinamento da rede através das alterações dos pesos sinápticos, por exemplo, uma taxa de 0,5 irá diminuir todo o gradiente em 50%.	Haykin (2008)

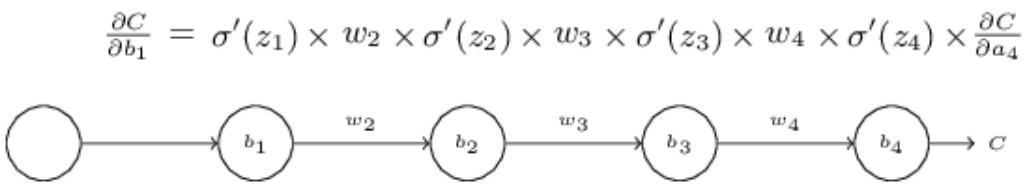
Fonte: Elaborado pelo autor (2016)

3.3.2 DESAPARACIMENTO (VANISHING) E EXPLOSÃO DE GRADIENTES

De acordo com Nielsen (2015), o fenômeno de desaparecimento de gradientes ou vanishing gradient (VG) decorre da retropopagação do erro das camadas finais até as camadas iniciais das DNNs. Este fenômeno é considerado um antigo problema no treinamento das redes neurais. Conforme o gradiente, calculado na camada final da rede, é movido até as camadas iniciais o seu valor tende a ficar menor, como a multiplicação entre dois números decimais (0,1 multiplicado por 0,1 resulta em 0,01). Isto significa que os neurônios nas camadas iniciais aprendem mais lentamente que os neurônios das camadas posteriores. O problema de VG não é inevitável, no entanto, sua solução tende às vezes tornar o gradiente maior nas camadas iniciais (NIELSEN, 2015).

Analisando o processo de cálculo do gradiente da primeira camada de uma rede neural através da retropropagação, é possível entender o problema de VG. A expressão que representa o cálculo deste gradiente é ilustrada na figura 6. Cada círculo na imagem simboliza uma camada da rede composta por um único neurônio e um bias "b", interligado com um peso "w". Ao final existe o cálculo do custo (letra C na figura). O termo σ' simboliza a derivada da função de ativação sigmoid de cada neurônio, $\partial C/\partial a4$ o custo relativo à saída do quarto neurônio e o termo $\partial C/\partial b1$ o gradiente do primeiro neurônio (NIELSEN, 2015).

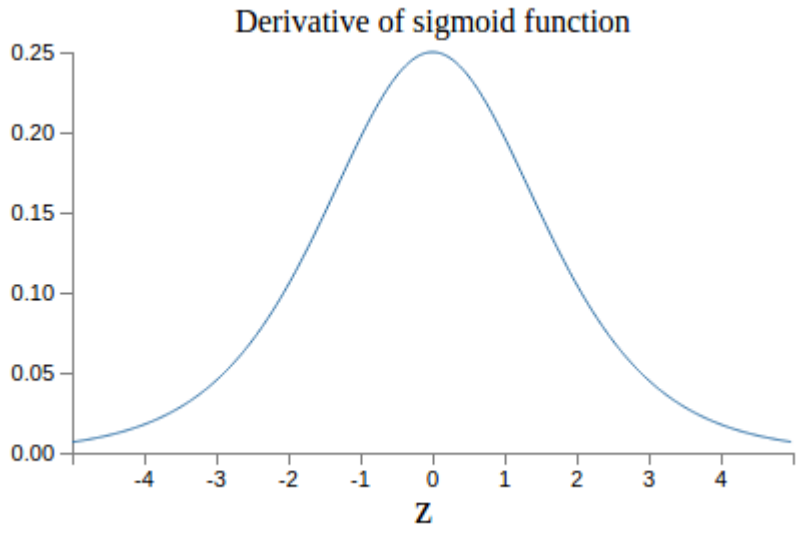
Figura 6 – Representação do cálculo do gradiente



Fonte: Nielsen (2015).

A derivada da função sigmoid possui o formato exposto na figura 7, na qual o valor máximo da função atinge 0,25 no gráfico.

Figura 7 – Derivada da função de ativação sigmoid

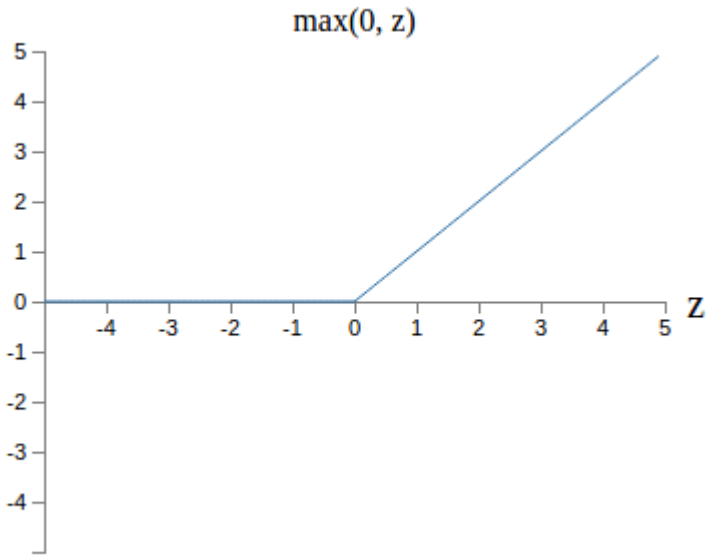


Fonte: Nielsen (2015).

Nota-se pelo gráfico da figura 7 que os termos $w_j\sigma'(z_j)$ satisfazem a condição $|w_j\sigma'(z_j)|<1/4$. Neste sentido, quando calculado o produto de muitos termos com essa condição, o produto final tende a diminuir exponencialmente, ou seja, ocasionando o fenômeno de vanishing (NIELSEN, 2015).

Uma maneira para contornar este problema é através do uso de outra variação de função de ativação chamada ReLU (figura 8). A saída de uma função ReLU com "x" de entrada, vetor de pesos "w" e bias "b" é dada pela fórmula: Max (0, w*x + b) (DENG, YU, 2015). Esta função max(0, z) possui um formato semelhante ao apresentado na figura 8.

Figura 8 – Comportamento da função de ativação RELU



Fonte: Nielsen (2015).

Observando o gráfico da função ReLU percebe-se que seu comportamento é diferente da função sigmoid.

Ainda que alguns trabalhos recentes sobre reconhecimento de imagens encontraram benefício considerável no uso da ReLU, não existe uma compreensão bem definida de quando exatamente ReLUs são preferíveis em relação às funções sigmoid e tangente hiperbólica. A função tangente hiperbólica sofre de um problema de saturação semelhante ao da função sigmoid. No caso da ReLU, quando o valor da entrada é aumentado nunca ocorre a saturação do gradiente, sendo assim, não gerando a lentidão no aprendizado. No entanto, quando a entrada para esta função é negativa o gradiente desaparece (zera) e perde o aprendizado inteiramente (NIELSEN, 2015).

A explosão dos gradientes é ocasionada pelo crescimento exponencial dos seus valores até as camadas iniciais da rede neural, contrário ao problema de desaparecimento dos gradientes. Porém uma solução para este problema é baseada em uma proposta simples de restrição de valores obedecendo a um threshold, caso o valor do gradiente exceda esse threshold este é diminuído a um valor mínimo (DENG, YU, 2015).

3.3.3 OVERFITTING E UNDERFITTING

Na visão de Goodfellow, Bengio e Courville (2016), o algoritmo de treinamento de uma rede neural está atingindo bons resultados caso consiga diminuir a taxa de custo do treinamento e tornando o tamanho do gap entre este custo e o custo em relação ao teste pequeno. Essas duas razões são consideradas os desafios centrais no aprendizado de máquina e são conhecidas como underfitting e overfitting. O underfitting acontece quando o modelo não possui a capacidade suficiente para convergir ao resultado esperado, não atingindo uma taxa de erro baixa no processo de treinamento. Já o overfitting é caracterizado se o gap entre o erro gerado no conjunto de treinamento e o erro gerado no conjunto de teste for grande (Goodfellow, Bengio e Courville, 2016).

Para amenizar os efeitos causados pelo overfitting e underfitting, pode ser necessário diminuir a capacidade de operação do modelo. Isto é, reduzir a capacidade de identificação de tipos diferentes de características do conjunto de treinamento. Uma rede com alta capacidade de adaptação pode acabar memorizando as características do conjunto de treinamento, obtendo resultados ruins quando aplicado em outras amostras. Por outro lado, uma rede com uma capacidade de adaptação muito baixa não conseguirá gerar o conhecimento suficiente para o conjunto de treinamento (Goodfellow, Bengio e Courville, 2016).

Muitas estratégias utilizadas no aprendizado de máquina são explicitamente projetadas para reduzir o erro de teste, estas estratégias são conhecidas como regularização. Qualquer modificação que for feita ao algoritmo de aprendizagem com o objetivo de reduzir o erro na generalização, mas não o erro durante o treinamento. Regularização L2 e dropout são exemplos dessas técnicas (Goodfellow, Bengio e Courville, 2016).

A estratégia de regularização L2 foca na alteração da função de custo, acrescentando um termo de regularização. Este termo consiste na soma dos quadrados de todos os pesos da rede, escalado por um parâmetro de regularização “λ”, em que $\lambda > 0$ (Goodfellow, Bengio e Courville, 2016).

O principal objetivo da regularização L2 é fazer com que a rede prefira gerar valores pequenos para os pesos, sendo que pesos com valores mais altos só serão permitidos caso melhorarem substancialmente a função de custo. Sendo assim, a regularização L2 pode ser vista como uma forma de compromisso entre gerar pesos com valores baixos e minimizar a função de custo original (LISA LAB, 2015). No ponto de vista de Nielsen (2015), a importância relativa aos dois elementos de compromisso depende do valor estipulado para o termo de regularização: quando o valor deste termo é pequeno é preferível minimizar a função

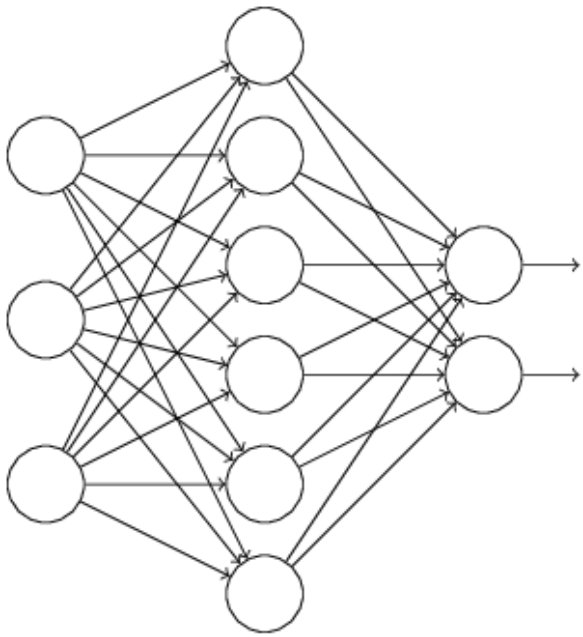
de custo original, todavia quando o seu valor é mais elevado, concentra-se na geração de valores baixos para os pesos sinápticos.

A questão que permeia a geração de valores baixos para os pesos é a possibilidade de reduzir a complexidade do modelo, fornecendo uma explicação mais simples e poderosa para os dados. Com valores baixos para os pesos o comportamento geral da rede não vai ser alterado caso sejam realizadas pequenas alterações aleatórias nos dados de entrada, portanto, reprimindo a rede de aprender ruídos nos dados (LISA LAB, 2015).

Em resumo uma rede regularizada não visa aprender características específicas dos dados, ao invés disso, procura generalizar o modelo aprendendo características que são vistas frequentemente nos dados. Ao contrário de uma rede neural não regularizada com pesos sinápticos elevados, em que se gera um modelo complexo baseado no ruído apresentado pelos dados (NIESLSEN, 2015).

Goodfellow, Bengio e Courville (2016) apresentam o dropout como uma técnica poderosa de regularização de baixo custo computacional. Sua estratégia não se baseia na alteração da função de custo. De maneira oposta, altera diretamente a estrutura da rede neural propriamente utilizando-se de uma prática de bagging. Bagging consiste no treinamento e avaliação de várias redes para o mesmo dado de exemplo. Desta maneira, dropout treina o conjunto de todos os modelos diferentes de sub-redes que podem ser formados removendo unidades das camadas ocultas da rede aleatoriamente. Esse processo é repetido inúmeras vezes, restaurando os neurônios removidos e escolhendo um novo conjunto de neurônios para serem deletados, estimando o gradiente e atualizando os valores dos pesos sinápticos e biases da rede. Desse modo, a rede poderá aprender pesos e biases sobre diferentes modelos (Goodfellow, Bengio e Courville, 2016).

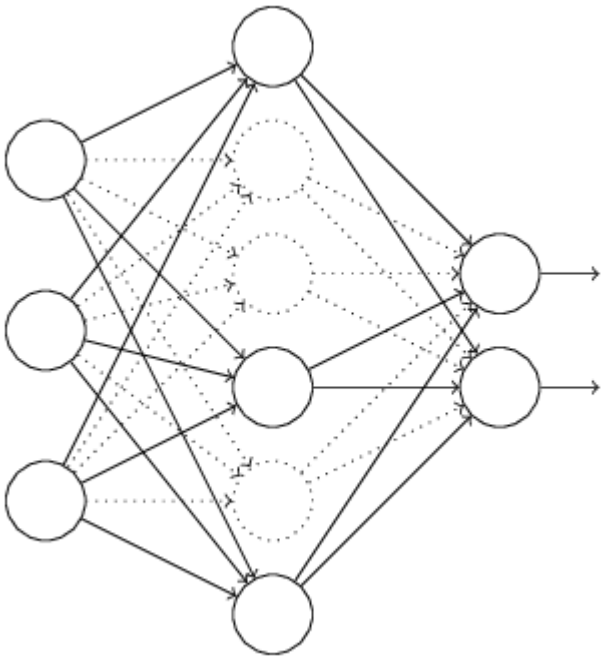
Figura 9 – Arquitetura de uma rede neural antes da aplicação de dropout



Fonte: Nielsen (2015).

Por exemplo, a figura 9 ilustra uma rede neural com três camadas, uma camada de entrada, uma camada oculta e por último a camada de saída. Antes de iniciar o treinamento desta rede neural são selecionados, aleatoriamente, metade dos neurônios para serem temporariamente deletados, como mostra a figura 10.

Figura 10 – Arquitetura de uma rede neural após a aplicação de dropout



Fonte: Nielsen (2015).

Em seguida os dados de entrada são propagados e retro propagados através da rede modificada. Após a realização deste passo para um mini-batch de exemplos, os pesos e biases são ajustados, repetindo o processo, primeiro restaurando os neurônios deletados e depois escolhendo um novo subconjunto aleatório de neurônios ocultos para serem excluídos (NIELSEN, 2015).

De uma forma mais simplificada, Nielsen (2015) resume a técnica de dropout como uma maneira de certificar que o modelo é robusto à perda de qualquer pedaço de evidência. Este método reduz complexas coadaptações de neurônios, uma vez que um neurônio não pode contar com a presença de determinados outros neurônios.

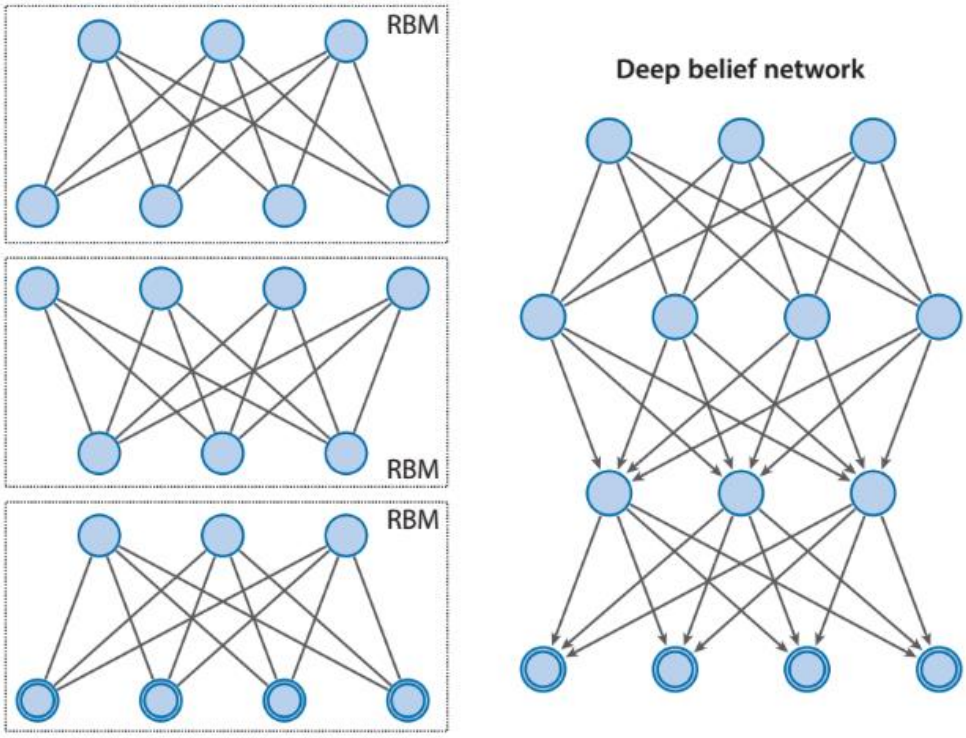
A seguir serão explicadas algumas arquiteturas de DNNs pesquisadas neste trabalho.

3.3.4 DEEP BELIEF NETWORK

Salakhutdinov (2015) define as DBNs como modelos probabilísticos que possuem muitas camadas ocultas de neurônios, em que as camadas mais altas aprendem correlações entre as camadas de nível mais baixo. Uma DBN é composta por uma pilha de RBMs (figura 11), nos quais a camada oculta de cada RBM é a camada visível da RBM acima. Com relação às suas possíveis aplicações, as DBNs são utilizadas em tarefas de classificação, regressão ou redução de dimensionalidade.

Seguindo sob a análise de Salakhutdinov (2015), uma RBM é uma rede neural generativa e sua estrutura é dividida em duas partes, uma camada com neurônios visíveis e outra camada com neurônios ocultos. Como modelo generativo, seu objetivo é encontrar padrões automaticamente por reconstrução da entrada utilizada na rede. Cada neurônio na primeira camada é conectado a todos os neurônios da segunda camada, mas nenhum neurônio compartilha conexões com os outros neurônios da mesma camada. RBMs são treinadas individualmente através de um algoritmo chamado c*ontrastive divergence.* Deve ser destacado que uma RBM está decidindo quais características da entrada são as mais importantes e como elas devem ser combinadas para a formação de padrões. (SALAKHUTDINOV, 2015).

Figura 11 – Ilustração de uma deep belief network



Fonte: Salakhutdinov (2015).

Os métodos de aprendizado dessa rede são ambos, o não supervisionado e o supervisionado. O não supervisionado pelo fato das RBMs serem modelos generativos e serem treinadas com dados inicialmente sem labels. Com isso o processo final de treinamento da DBN consiste no método supervisionado, a fim de realizar a classificação dos dados para convergir ao resultado esperado (BENGIO, 2009). Para o treinamento inicial de uma DBN é preciso de um conjunto significativo de dados não classificados, porém são necessários poucos dados classificados para o ajuste final do modelo, facilitando a sua aplicação em tarefas do mundo real, onde existem poucas informações classificadas (SALAKHUTDINOV, 2015).

Para Bengio (et al., 2009) o treinamento de uma DBN é dividido em duas fases o pré-treinamento e um *fine-tuning* ou ajuste geral da rede. No pré-treinamento a primeira RBM é treinada para reconstruir sua entrada da melhor maneira possível. A camada oculta (segunda camada) da primeira RBM é tratada como a camada visível da segunda RBM, que é treinada usando as saídas da RBM anterior. Esse processo é repetido até que cada camada da rede seja treinada (BENGIO et al., 2009). No estágio de fine-tuning: depois do treinamento inicial da rede, as RBMs criam um modelo capaz de detectar padrões particulares nos dados, todavia este modelo ainda não sabe exatamente o que esses padrões são. Para finalizar o treinamento é necessário introduzir labels para os padrões encontrados e afinar (fine-tuning) a rede com

aprendizado supervisionado. Isto pode ser realizado através do algoritmo BP. Os pesos e os biases são alterados levemente, resultando em uma pequena alteração da percepção de padrões da rede e geralmente uma pequena melhora na precisão final (BENGIO et al., 2009).

As conexões entre as camadas e neurônios de uma DBN dão-se da seguinte forma: as duas camadas superiores da rede possuem conexões simétricas e sem direção entre elas. As camadas mais baixas recebem as conexões no sentido de cima para baixo (DENG; YU, 2014). Como descrito por Deng e Yu (2015), quando o método de pré-treinamento é aplicado em modelos para classificação, por exemplo, deve ser adicionada uma camada final ao modelo com o objetivo de representar a saída desejada.

3.3.5 REDE NEURAL CONVOLUCIONAL

Para explicar como a rede neural convolucional (CNN) funciona, será tomado como base o exemplo apresentado por Nielsen (2015), na classificação de imagens de números escritos à mão (ilustração na figura 12). A explicação contida na obra desse autor é didática e de fácil entendimento, sem definições matemáticas complexas e por esses motivos guiará esta seção.

Figura 12 – Ilustração do *dataset* MINST conjunto de imagens de números manuscritos

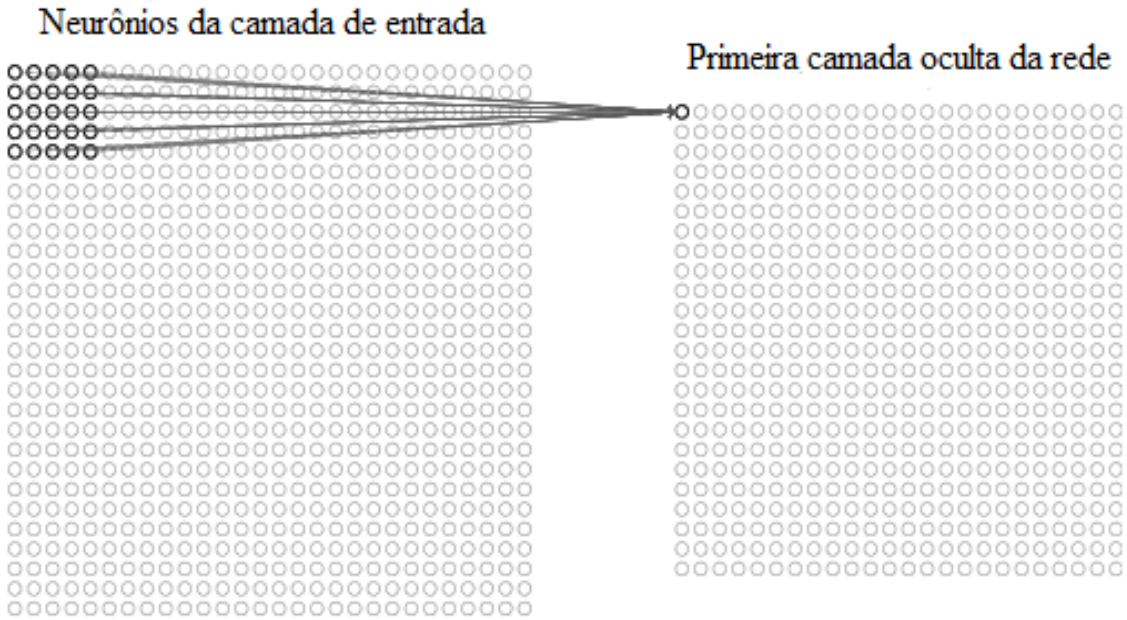
Fonte: Nielsen (2015).

Para cada pixel na imagem utilizada como entrada da rede é codificada a intensidade do pixel como o valor para o neurônio correspondente. Cada imagem do dataset MINST é composta por 28×28 pixels de resolução, desse modo a camada de entrada da rede irá possuir 784 neurônios. CNNs utilizam-se de uma arquitetura especial que é adaptada para a classificação de imagens, permitindo um treinamento mais rápido da rede. Atualmente, CNNs e algumas de suas variantes são empregadas na maioria das redes neurais para

reconhecimento de imagens e sua arquitetura é composta por 3 ideias básicas: 1) campos receptivos locais; 2) pesos compartilhados; 3) camadas de agrupamento (NIELSEN, 2015).

Os pixels de entrada serão conectados a uma camada oculta de neurônios. Contudo não serão conectados todos os pixels em todos os neurônios da camada oculta, ao invés disso, serão realizadas conexões em pequenas regiões da imagem de entrada (figura 13). Ou seja, cada neurônio na primeira camada oculta será conectado a uma pequena região da camada de entrada (kernel), por exemplo, uma região com a dimensão de pixels de 5x5. Essa região na imagem de entrada é chamada de campo receptivo local, no qual, cada conexão terá um peso associado a um neurônio oculto com um bias. Assim, o neurônio oculto irá aprender a analisar seu campo receptivo local particular (NIELSEN, 2015).

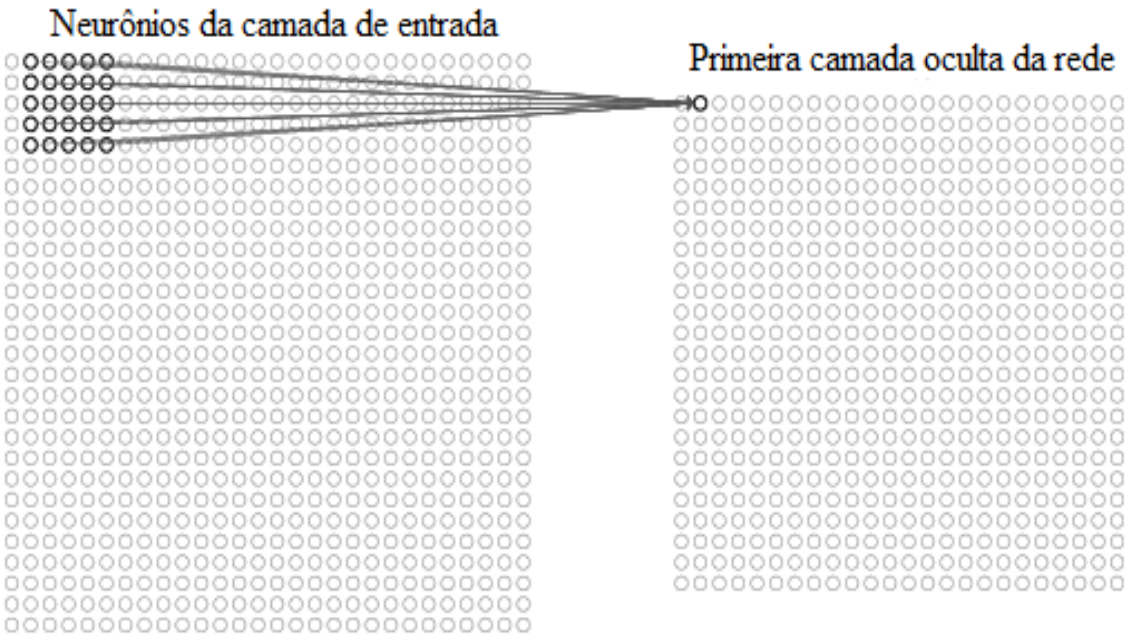
Figura 13 – Campo receptivo local de uma CNN



Fonte: Adaptado de Nielsen (2015).

O campo receptivo local é deslocado então por toda a imagem e para cada campo receptivo local existe um neurônio na camada oculta, gerando uma quantidade de 576 (24x24) neurônios. A figura 14 exibe está operação.

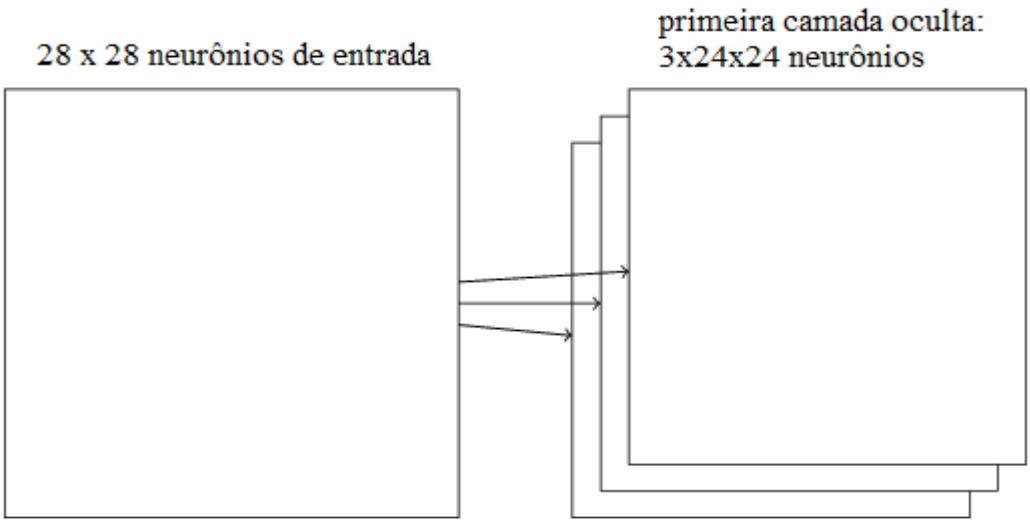
Figura 14 – Deslocamento do campo receptivo local de uma CNN



Fonte: Adaptado de Nielsen (2015).

Como comentado anteriormente, CNNs compartilham pesos e biases. Isso porque cada neurônio na camada oculta gerado pelos campos receptivos locais possuem os mesmos pesos e biases. Para que o conceito seja melhor compreendido, suponha-se que os pesos e os biases são tais que o neurônio na camada oculta possa selecionar uma aresta vertical em um campo receptivo local em particular. Essa habilidade também pode ser útil para outros locais na imagem. Uma grande vantagem no compartilhamento de pesos e biases é a redução no número de parâmetros da rede. Nessa arquitetura de rede de exemplo, podem ser detectados três diferentes tipos de características na imagem, definidas por um conjunto de 25 pesos e um bias compartilhados, formando três mapas de características diferentes. Este conjunto de mapas compõe uma camada convolucional. A figura 15 ilustra a arquitetura de rede com uma camada convolucional contendo os três mapas (NIELSEN, 2015).

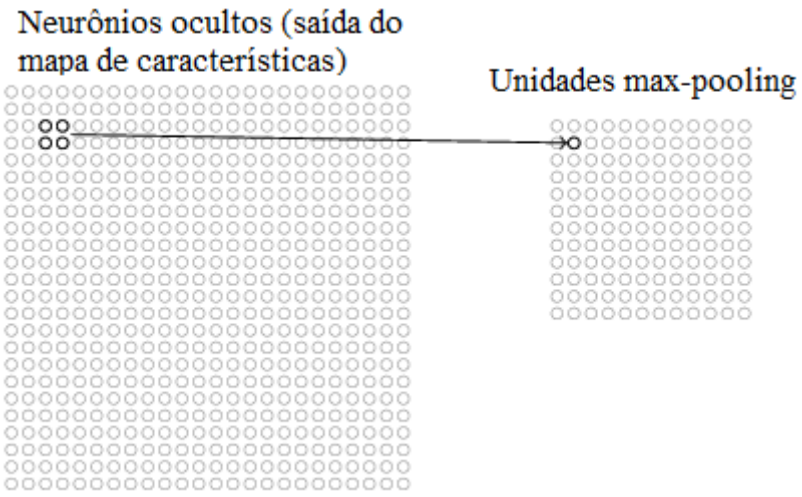
Figura 15 – Mapas de características em uma CNN



Fonte: Adaptado de Nielsen (2015)

Além das camadas convolucionais, CNNs também possuem camadas de agrupamento, com o objetivo de simplificar a informação na saída das camadas convolucionais. Uma camada de agrupamento pega cada saída da camada convolucional e prepara uma camada condensada de características, por exemplo, irá somar uma região de neurônios com dimensão de 2x2 da camada anterior, mostrado na figura 16. Um procedimento comum utilizado para a realização do agrupamento é o max-pooling, em que simplesmente retorna a máxima ativação da região aplicada (GOODFELLOW, BENGIO, COURVILLE, 2016).

Figura 16 – Camada de max-pooling em uma CNN

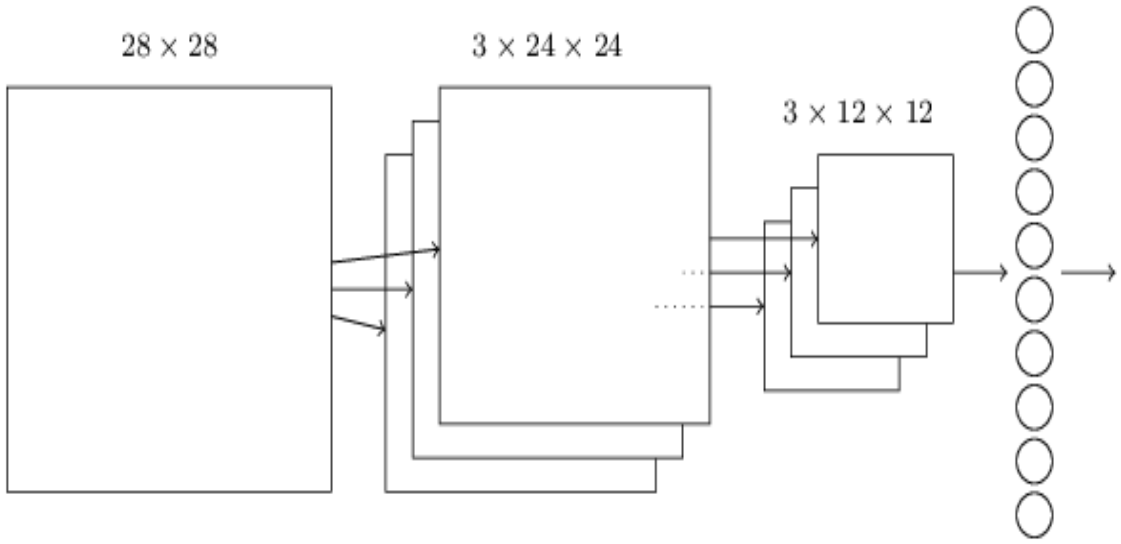


Fonte: Adaptado de Nielsen (2015)

Max-pooling pode ser considerado como uma maneira pela qual a rede verifica se determinada característica pode ser encontrada em toda uma região específica da imagem. Isso significa que não importa a localização exata dessa característica na imagem e sim se a mesma existe ou não. Tendo como grande vantagem a redução do número de parâmetros necessários para as camadas posteriores, pois existem poucas características diferentes agrupadas. Max-pooling não é a única técnica utilizada para agrupamento, outra abordagem comum é conhecida como L2 pooling (GOODFELLOW, BENGIO, COURVILLE, 2016).

A arquitetura final da CNN é representada na figura 17:

Figura 17 – Ilustração da arquitetura final de uma CNN para classificação de imagens



Fonte: Nielsen (2015)

A primeira camada contém 784 neurônios de entrada, destinados a codificar a intensidade de cada pixel da imagem. Seguida por uma camada convolucional composta por campos receptivos locais com dimensão de 5x5 e três mapas de características, resultando em uma camada de 3x24x24 neurônios ocultos. A próxima camada é formada pelo max-pooling ao longo dos três mapas, resultando em outra camada com 3x12x12 neurônios ocultos. A camada final possui 10 neurônios para gerar a saída da rede, cada neurônio responsável por um número dentro do conjunto de 0 a 9 e são conectados com todos os neurônios nas camadas formadas pelo max-pooling (NIELSEN, 2015).

3.3.6 REDE NEURAL RECORRENTE (RNN)

Uma RNN pertence a uma classe de redes neurais destinada ao processamento de informações sequenciais. As conexões entre seus neurônios formam um ciclo direcionado, gerando uma estrutura de estados internos da rede ou memória e, com isso, apresentando um comportamento dinâmico ao longo do tempo não apresentado pelas DNNs tradicionais (GOODFELLOW, BENGIO, COURVILLE, 2016).

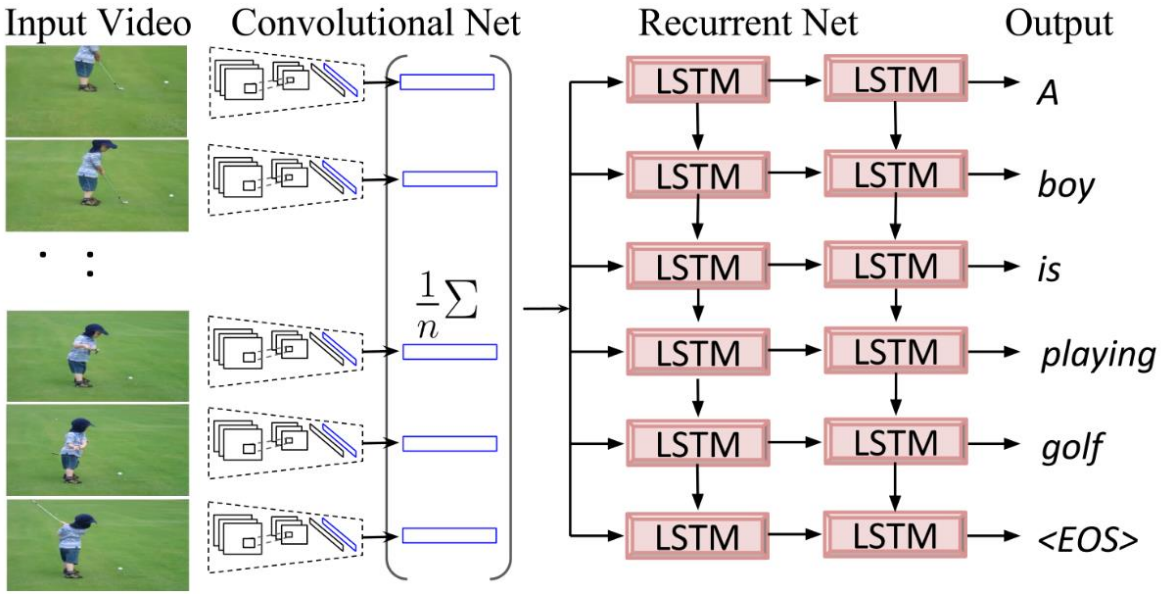
Deng e Yu (2015) frisam que uma das principais diferenças entre as DNNs tradicionais e RNNs é com relação aos seus parâmetros de entrada. As RNNs não operam somente com os parâmetros de entrada da rede, como acontece nas DNNs, mas também com estados internos da rede que codificam informações passadas (que já foram processadas anteriormente pela rede) em uma sequência temporal. As funções de ativação de neurônios comumente utilizadas nas camadas ocultas das RNNs são as sigmoid e ReLU. Já para as camadas de saída das RNNs são empregadas as funções lineares e softmax (DENG, YU, 2015).

Goodfellow, Bengio e Courville (2016) destacam alguns exemplos importantes de padrões de arquiteturas para as RNNs:

- RNNs que produzem uma saída em cada intervalo de tempo e possuem conexões recorrentes entre os neurônios ocultos da série temporal;
- RNNs que produzem uma saída para cada intervalo de tempo, porém possuem conexões recorrentes somente entre a saída de cada intervalo de tempo (t) para os neurônios ocultos do próximo intervalo de tempo (t+1);
- RNNs com conexões recorrentes entre os neurônios ocultos, que leem uma sequência completa de dados para depois gerar apenas uma saída.

Do contrário das feed forward networks, RNNs podem receber uma sequência de valores como entrada e podem também produzir uma sequência de valores como saída. A habilidade de operar com sequências abre um leque de opções para sua aplicação. Como por exemplo, quando a entrada é única e a saída é uma sequência, uma aplicação seria a captura de imagens. Já quando há uma sequência de entradas para uma única saída ela pode ser utilizada para a classificação de documentos. Quando ambas, entrada e saída, são sequências de valores, essas redes podem classificar vídeos frame por frame, como mostra a figura 18 (VENUGOPALAN et al., 2015).

Figura 18 – Ilustração de uma RNN para classificação de vídeos



Fonte: Venugopalan et.al., (2015)

Empilhando RNNs é possível ter uma rede com capacidades ainda mais complexas. RNNs são extremamente difíceis de treinar, desde que usam BP para o treinamento, voltando ao problema de vanishing gradient (VG), no qual é exponencialmente pior em uma RNN. A razão disto, é que para cada passo de tempo é equivalente a uma camada de rede inteira em uma rede feedforward (GOODFELLOW, BENGIO, COURVILLE, 2016).

O algoritmo backpropagation through time (BPTT) é o algoritmo padrão para o treinamento de uma RNN. No processo de atualização dos pesos sinápticos de uma RNN o BPTT desdobra a rede no tempo e retro propaga os sinais de erro. Pode-se perceber que o BPTT é uma variação do algoritmo BP utilizado em DNNs, a diferença é que as camadas ocultas para o mesmo frame de treinamento (t) são substituídas pelas mesmas camadas ocultas, entretanto divididas através do tempo (t = 1, 2,...). Outra diferença apresentada pelo algoritmo BPTT é com relação ao seu tempo de conversão mais lento. Isso se deve ao fato de existir dependências entre os frames das RNNs, propiciando a ocorrência dos problemas de exploding e VG (DENG, YU, 2015).

Para solucionar os problemas de exploding e VG é utilizada uma técnica de *gating* que ajuda a rede a decidir quando esquecer uma entrada atual ou quando lembrá-la em futuros intervalos de tempo. Os gatings mais populares atualmente são GRU e LSTM. Além da técnica de gating existem outras opções como *gradient clipping*, *steeper gates* e melhores otimizadores (GOODFELLOW, BENGIO, COURVILLE, 2016). As RNNs baseadas em células LSTM são adaptadas para aprenderem, de uma sequência de entradas, a classificar,

processar e prever séries temporais quando existem atrasos extensos de tamanhos desconhecidos entre eventos importantes da série analisada (DENG, YU, 2015).

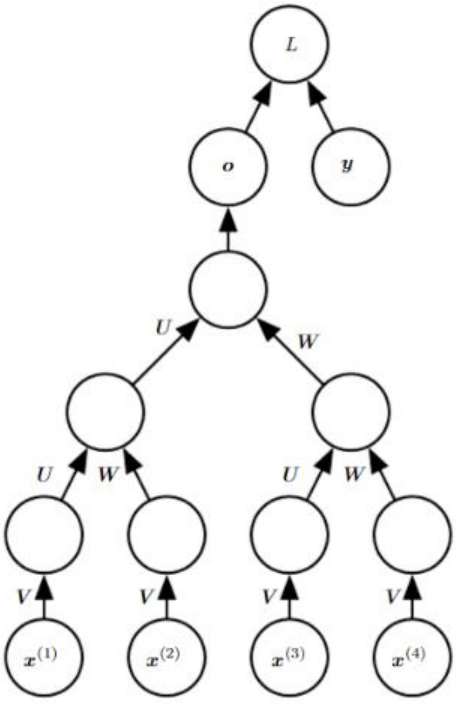
Deng e Yu (2015) caracterizam uma célula LSTM como uma unidade complexa e inteligente capaz de recordar informações de longo tempo, determinando quando uma entrada é relevante para ser lembrada, quando ela deve continuar a lembrar ou esquecer a informação, bem como quando deve ser a sua saída.

3.3.7 REDES NEURAIS RECURSIVAS

Goodfellow, Bengio e Courville (2016) apresentam as redes neurais recursivas como uma generalização das RNNs, pois, diferentemente da estrutura baseada em cadeia, RNRs são estruturadas como uma árvore. Uma clara vantagem desse tipo de rede neural sobre das redes recorrentes decorre da possibilidade de diminuição da profundidade da rede, auxiliando no processamento de dependências longas entre as informações.

Em uma RNR rasa, uma única camada é responsável pelo aprendizado de representações que são úteis e suficientes para a saída correta da rede, porém em uma deep RNR, uma camada pode aprender algumas características e transferir esse aprendizado para a próxima camada, a fim de processar o restante das informações (ÍRSOY; CARDIE, 2014). Na figura 19 é exibido um exemplo de RNR.

Figura 19 – Representação de uma rede neural recursiva



Fonte: Goodfellow; Bengio; Courville, (2016).

RNRs obtiveram o estado da arte em performance de tarefas relacionadas ao processamento de linguagem natural (NLP). A ideia por traz do uso de RNRs para NLP é treinar uma DNN com capacidade de processar entradas com qualquer tamanho. Ao contrário da visão computacional, na qual facilmente uma imagem pode ser fixada em um número de pixels, sentenças de linguagem não possuem um tamanho fixo e podem ser estruturadas em árvores. Analisando a estrutura da rede, pode ser adicionado para cada folha um tamanho fixo de vetor para uma palavra e combiná-las para a criação nodos intermediários do mesmo tamanho (SOCHER, PAULUS, MANNING, 2014).

3.3.8 AUTOENCODERS

Deep autoencoder é um exemplo de DNN que não possui a necessidade de labels para seu treinamento, o seu objetivo é realizar uma cópia da informação passada como parâmetro da rede e seu treinamento é realizado geralmente por uma variação do algoritmo BP como o SGD. Em sua estrutura existem camadas ocultas de neurônios que são responsáveis por codificarem a entrada para então representá-la, sendo assim, o tamanho da saída gerada é o mesmo tamanho do dado utilizado como entrada. Um autoencoder possui duas funções principais, um *encoder* (codifica a representação da informação) e um *decoder* (que produz a reconstrução da informação). Geralmente existem restrições nessas funções permitindo apenas uma cópia aproximada da entrada, desse modo o autoencoder é forçado a priorizar determinadas características que devem ser copiadas e por consequência aprende informações úteis dos dados (GOODFELLOW, BENGIO, COURVILLE, 2016).

Deng e Yu (2014) frisam que ao contrário de uma rede para classificação, um autoencoder é um método de extração de características que representam melhor os dados, embora esses dois modelos possam estar correlacionados, como em redes neurais híbridas.

Com relação à divisão de camadas de um autoencoder, a mesma se faz por uma camada de dados de entrada, que representam os dados originais (por exemplo, os pixels de uma imagem ou o espectro de uma fala). Uma ou mais camadas ocultas que são responsáveis pela transformação dos dados e uma camada de saída que compreende a reconstrução do dado de entrada (RBM é um exemplo de autoencoder que possui somente duas camadas). Dependendo da aplicação do autocoder a dimensão das camadas ocultas pode sofrer alterações, ela pode ser menor do que a dimensão da informação de entrada quando o objetivo é a compressão de dados, ou maior quando o objetivo é mapear características de dados de treinamento e separá-las (DENG; YU, 2014).

No livro em preparação por Goodfellow, Bengio e Courville (2016) são apresentadas algumas variações dos autoencoders, como por exemplo, *denoising autoencoders* (DAE), *contractive autoencoders* (CAE) e *sparsity autoencoders*.

Em um DAE a entrada é aleatoriamente corrompida, no entanto os dados não corrompidos são ainda utilizados com o objetivo da reconstrução. Basicamente um DAE realiza duas tarefas, ele tenta codificar a entrada, preservando as informações sobre ela, e tenta desfazer o efeito causado pelo processo de corrupção. Sendo que essa última tarefa só é viável caso sejam aprendidas as dependências entre as entradas do modelo. O processo de corrupção dos dados é realizado setando aleatoriamente alguns dados de entrada para zero e, portanto, o autoencoder irá tentar prever as informações que faltarem com os dados restantes (BENGIO, 2009).

A corrupção dos dados tem o objetivo de forçar a geração da saída para se igualar aos dados originais de entrada sem as distorções e com isso, evitar que o modelo aprenda uma solução comum para o problema. Outro objetivo é tornar o modelo robusto para os mesmos tipos de corrupções nos dados existentes no conjunto de teste. Por fim, como cada dado com corrupção torna-se diferente do outro, o tamanho do conjunto dos dados de treinamento aumenta auxiliando na diminuição do problema de overfitting (DENG; YU, 2014). A motivação para a utilização de DAEs é permitir o aprendizado de alta capacidade para o modelo, ao mesmo tempo em que prevenindo o encoder e o decoder de aprender informações desnecessárias (GOODFELLOW, BENGIO, COURVILLE, 2016).

Contractive autoencoder (CAE) é um algoritmo para aprendizado não supervisionado e outra variação de autoencoders. CAE adiciona um termo de regularização ao objetivo do autoencoder de minimizar o erro de reconstrução dos dados, esse termo de regularização penaliza a sensibilidade dos recursos de codificação. Selecionando corretamente o termo de penalidade, um CAE pode obter resultados tão bons ou melhores que um DAE (RIFAI et.al., 2011). O nome “contractive” deve-se à maneira com que o CAE envolve o espaço da informação, ou seja, esse autoencoder é treinado para resistir a distorções nos dados e por isso é estimulado a mapear um conjunto de entradas em um conjunto menor (GOODFELLOW, BENGIO, COURVILLE, 2016).

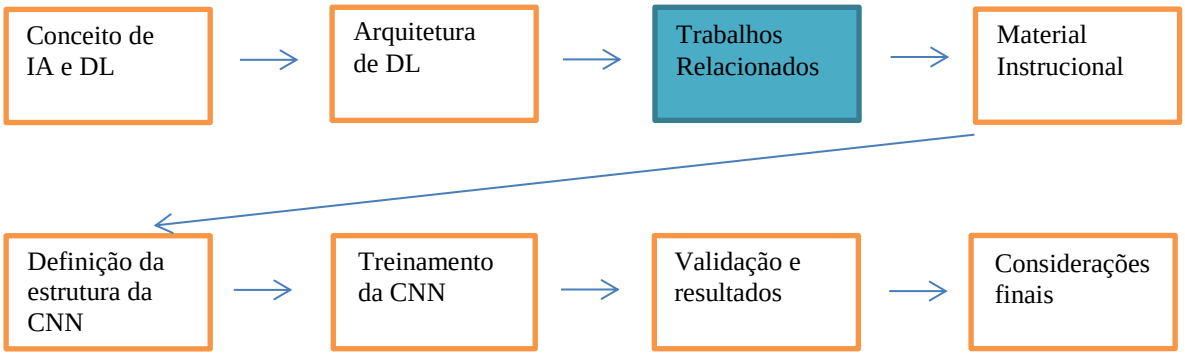
CAEs geram o aprendizado realizando o balanceamento de duas forças, a taxa de erro da reconstrução e o termo de penalidade. A taxa de erro na reconstrução, sozinha, estimula o CAE a aprender uma codificação que iguala o modelo à entrada. Já o termo de penalidade sozinho, o estimula a aprender características que são constantes com relação ao parâmetro selecionado (GOODFELLOW, BENGIO, COURVILLE, 2016).

Por fim, são descritos os sparse autoencoders que utilizam um termo de regularização diferente que envolve dispersão. Este autoencoder deve responder a características exclusivas do conjunto de dados, ao invés de simplesmente realizar uma cópia das mesmas. Esse tipo de autoencoder é melhor para tarefas de classificação em comparação aos DAEs, redes treinadas com dropout e RBMs (GOODFELLOW, BENGIO, COURVILLE, 2016).

Autoencoders são aplicados para redução de dimensionalidade e recuperação de informações, tarefa de encontrar informações em um banco de dados que respondem a uma determinada solicitação feita como entrada. Representações com uma dimensão reduzida podem melhorar a performance em processos de classificação, também consomem menos memória e possuem um menor tempo de processamento (GOODFELLOW, BENGIO, COURVILLE, 2016).

Seguindo os passos do fluxograma da pesquisa, o próximo capítulo aborda os trabalhos relacionados, etapa esta, marcada em azul na figura 20.

Figura 20 – Fluxograma da pesquisa



Fonte: Elaborado pelo autor (2016).

4 TRABALHOS RELACIONADOS

Neste capítulo, serão apresentados trabalhos relacionados à questão central desta pesquisa: o auxílio na aprendizagem dos conceitos chave de DL. Para tanto, 3 (três) materiais foram selecionados: 1) e-book online do autor Nielsen (2015); 2) um tutorial atualizado em 2015 e disponibilizado pelo instituto Lisa Lab da universidade de Montreal (dirigido por um dos principais autores na área de DL: Yoshua Bengio) (2015)); 3) Curso online realizado através de vídeo aulas interativas concedido pelo Google (2016).

4.1 TUTORIAL DISPONIBILIZADO PELO LISA LAB

O tutorial sobre DL disponibilizado pelo Lisa Lab da universidade de Montreal foi desenvolvido pela equipe que criou a biblioteca Theano. Esta biblioteca é escrita em python[4] e facilita o desenvolvimento de sistemas baseados em DL, com a possibilidade de realizar o treinamento em unidades de processamento gráfico (GPUs) (LISA LAB, 2015). Este tutorial foi selecionado por ter sido desenvolvido em um instituto dirigido por um dos principais autores de DL, Yoshua Bengio.

Todas as práticas feitas no tutorial utilizam-se da biblioteca Theano e do dataset MNIST, um conjunto de imagens de números escritos à mão, divididos em 60.000 exemplos para treinamento e 10.000 exemplos para teste. Todas as imagens possuem um tamanho fixo de 28 x 28 pixels (LISA LAB, 2015). Os algoritmos implementados são diversos, sendo essa uma das vantagens deste tutorial que abrange: *logistic regression, multilayer perceptrons, redes neurais convolucionais, autoencoders, denoising/stacked autoencores, restricted boltzmann machines, deep belief networks,* redes neurais recorrentes e redes LSTM (variação da rede recorrente) (LISA LAB, 2015).

O tutorial segue um mesmo padrão para todos exemplos, começando pelo modelo mais básico e avançando na complexidade dos algoritmos a cada seção prática. Para cada modelo de algoritmo proposto são feitos, primeiramente, descrições de seus conceitos chave. Em seguida, a implementação de cada conceito apresentado e, por fim, a realização da codificação completa do algoritmo, efetuando testes de desempenho e mostrando o resultado obtido com cada modelo de rede (LISA LAB, 2015).

[4] Página oficial https://www.python.org/

4.2 CURSO ONLINE NA PLATAFORMA UDACITY

Outro material instrucional relacionado às técnicas e teorias do DL são as vídeoaulas disponibilizadas pelo Google na plataforma da Udacity (GOOGLE, 2016), as quais possuem um método interativo e inovador para o estudo da matéria em questão. Com exercícios interativos no próprio vídeo, o estudante pode rever quantas vezes ele achar necessário a questão apresentada e inserir sua resposta no exercício. Em seguida a sua resposta é validada e se estiver errada, o estudante pode fazer novas tentativas ou ver a resposta elaborada pelo instrutor do curso. Possuem ainda a facilidade de realizar a programação dos exercícios em notebooks apresentados durante as vídeoaulas. Os notebooks são aplicativos web que fornecem uma plataforma de desenvolvimento completa com opção de incluir gráficos, textos e imagens entre os comandos programados (GOOGLE, 2016).

O instrutor do curso é o cientista líder da área de IA da Google e a biblioteca utilizada para a criação dos modelos de DL chama-se Tensorflow. Esta biblioteca foi liberada em novembro de 2015 pelo Google como projeto open source e é escrita na linguagem python. Todas as aulas são focadas no problema de classificação de imagens e textos, por meio da utilização de DNNs, CNNs e RNNs com a mesma abordagem dos tipos de redes feita no tutorial criado pelo Lisa Lab. Inicialmente com o modelo mais simples de classificação e avançando gradualmente para algoritmos mais complexos. A duração das vídeoaulas varia de 17 segundos até no máximo 4 minutos. Este foi um ponto considerado negativo para este curso. Conceitos explicados em segundos, ou em 1 ou 2 minutos, dificultam o seu entendimento. Às vezes, é necessário rever o vídeo várias vezes pelo fato de ter um tempo curto e o instrutor passar o conteúdo rapidamente (GOOGLE, 2016).

Em resumo, este curso em específico foi selecionado por estar disponível gratuitamente na Web a qualquer horário, basta realizar um cadastro no site da Udacity. Além disso, o curso é ministrado por um profissional experiente na área de IA dentro da Google. Não obstante, o curso possui uma interface interativa que facilita o aprendizado e incentiva o estudante. Com relação à tecnologia utilizada no curso, a biblioteca Tensorflow promete facilitar o desenvolvimento de sistemas de DL e ter a garantia de suporte e melhorias continuamente pelo Google.

4.3 EBOOK ONLINE DE NIELSEN

A última fonte de estudo analisada é o e-book online Neural Networks and Deep Learning, escrito pelo autor Michael Nielsen, disponível no endereço <http://neuralnetworksanddeeplearning.com/> (NIELSEN, 2015).

Nielsen faz uma abordagem diferente com relação aos outros materiais analisados. Este foca na explicação detalhada dos conceitos fundamentais sobre redes neurais e DL, já os outros materiais, fazem uma abordagem mais superficial dos conceitos e se concentram mais na questão prática. Um ponto em comum entre eles refere-se à utilização inicial de modelos simples de sistemas de IA, para então, posteriormente, introduzir modelos mais complexos (NIELSEN, 2015).

O e-book de Nielsen possui uma explicação muito simples e clara sobre como funciona e qual objetivo das redes neurais em geral, utilizando-se de exemplos do dia-a-dia, como decidir em ir ou não a uma festa. A mesma ideia é empregada para a explicação do funcionamento de um sistema baseado em DL, onde é usado o exemplo do reconhecimento de um rosto humano em uma imagem. O livro também possui explicações matemáticas dos conceitos, no entanto, essas seções podem ser desconsideradas pelos leitores mais leigos no assunto (NIELSEN, 2015).

Os exemplos práticos de redes neurais são programados com a biblioteca Theano, a mesma utilizada no tutorial disponibilizado pelo Lisa Lab. No livro o autor faz uma análise minuciosa do desempenho desses algoritmos, comparando os resultados obtidos a cada alteração de parâmetro de rede efetuado, bem como, com as implementações anteriores. O autor também propõe desafios e atividades, todavia não oferece respostas para os mesmos (NIELSEN, 2015).

Como foi explicado na introdução do presente trabalho, existem poucos livros publicados sobre DL. O livro de Nielsen, além de ser gratuito e de fácil acesso, é um dos poucos a explicar os conceitos fundamentais de DL e redes neurais de forma simplificada. Por esses motivos foi selecionado como fonte para este estudo.

4.4 COMPARAÇÃO DOS MATERIAIS INSTRUCIONAIS

Esta seção é dedicada à comparação dos três materiais instrucionais apresentados nas etapas anteriores. Para demonstrar a comparação foi criada uma tabela (tabela 2) contendo a descrição do conceito ou a característica referente à pesquisa sobre redes neurais e DL

extraídos dos materiais. Na primeira coluna da tabela foi incluído um código sequencial para identificar a linha da tabela, esta informação será utilizada posteriormente na tabela 5 para fazer um relacionamento entre as duas tabelas. A segunda coluna possui a descrição do conceito que será comparado. As outras três colunas da tabela referem-se a cada material analisado, nas quais será assinalado com um "Sim/Não" se o material atende ou não ao conceito descrito na segunda coluna.

Para a seleção das informações contidas na segunda coluna da tabela, foram considerados os conceitos sobre DL e redes neurais existentes nos três materiais examinados. Ademais, foram listadas características importantes dos objetos de estudo, como tecnologia utilizada (Theano, Tensorflow, etc.) e tipo de dataset.

A tabela comparativa em questão permitirá identificar quais características e conceitos devem ser cogitados na elaboração de um material instrucional sobre DL que contemple as bases para sua compreensão.

Tabela 2 - Comparação entre os materiais instrucionais

Cód.	**Conceito ou Característica**	**Nielsen**	**Lisa Lab**	**Vídeo-Aulas**
1	Biblioteca para implementação de *deep learning*	Theano	Theano	Tensorflow
2	Rede Convolucional	Sim	Sim	Sim
3	Rede Neural Recorrente	Não	Sim	Sim
4	DNN com ReLU	Não	Não	Sim
5	*Autoencoder*	Não	Sim	Não
6	*Deep Belief Network*	Não	Sim	Não
7	Rede LSTM	Não	Sim	Sim
8	*Backpropagation*	Sim	Não	Sim
9	*Stochastic Gradient Descent*	Sim	Sim	Sim
10	Regra geral de funcionamento das redes neurais	Sim	Não	Não
11	*Dropout*	Sim	Não	Sim
12	*Regularization*	Sim	Sim	Sim
13	*Vanishing Gradient*	Sim	Sim	Sim
14	*Exploding Gradient*	Sim	Sim	Não
15	Utilização de notebooks online para as tarefas práticas	Não	Não	Sim
16	*Momentum*	Sim	Não	Sim
17	ReLU	Não	Não	Sim
18	Possui exercícios para o aluno responder	Sim	Não	Sim
19	LSTM	Não	Sim	Sim
20	Hierarquia de características em DNNs	Sim	Não	Sim
21	Respostas para os exercícios	Não	Não	Sim

Cód.	Conceito ou Característica	Nielsen	Lisa Lab	Vídeo-Aulas
22	Exemplos práticos	Sim	Sim	Sim
23	*Softmax*	Sim	Não	Sim
24	O que é um classificador	Sim	Não	Sim
25	Função de custo	Sim	Não	Não
26	*Overfitting*	Sim	Não	Não
27	*Underfitting*	Sim	Não	Não
28	Função Sigmoid	Sim	Sim	Sim
29	*Cross-entropy*	Sim	Sim	Sim
30	*Mini-batch*	Sim	Sim	Não
31	Modelagem de um neurônio artificial	Sim	Não	Não
32	Definição de deep learning	Sim	Sim	Sim
33	CNNs, campos receptivos locais	Sim	Sim	Sim
34	CNNs, compartilhamento de pesos e biases	Sim	Sim	Sim
35	CNNs, camadas de grupamento ou pooling	Sim	Sim	Sim
36	Motivo pela utilização de camada FC ao final da CNN	Sim	Não	Não
37	Regularização L2	Sim	Sim	Sim
38	*DataSet* utilizado nas práticas	MNIST	MNIST	NotMNIST

Fonte: Elaborado pelo autor (2016)

Na tabela 2 pode ser constatado que existem conceitos sobre DL que são abordados em mais de um material. A partir dos estudos desses materiais, pode-se apurar que esses conceitos que se repetem fazem parte dos pilares das DNNs e de seu processo de treinamento. Dentre esses conceitos destacam-se: backpropagation, stochastic gradient descent, dropout, regularization, vanishing gradient, exploding gradient, hierarquia de características em DNNs, softmax, função de custo, cross-entropy e mini-batch.

Diante do que foi apresentado como referencial teórico, os próximos capítulos retratam a abordagem utilizada neste trabalho no intuito de investigar como criar um material instrucional sobre DL.

5 MATERIAL INSTRUCIONAL SOBRE DEEP LEARNING

Na investigação realizada para criar um material instrucional que potencializasse o entendimento dos conceitos de DL e suas técnicas de implementação, deparou-se com o ambiente de desenvolvimento chamado Jupyter Notebook: utilizado no curso de DL disponibilizado pelo Google na plataforma Udacity (https://br.udacity.com/). Esta ferramenta também é encontrada no site Kaggle (https://www.kaggle.com/), na qual é empregada como meio dos usuários apresentarem suas soluções para diversos desafios propostos.

No curso, o emprego dos notebooks[5] é intercalado com os vídeos do professor discursando sobre os conceitos de aprendizado de máquina e DL. Neste caso, os notebooks constam com definições iniciais de exercícios específicos sobre a matéria em questão, em que o aluno pode complementá-los com sua resposta, testar o programa e submeter a resposta ao curso.

Notebooks jupyter, detalhados a seguir neste capítulo, possuem um console web que pode ser acessado localmente, células de programação que podem ser executadas separadamente. Além disso, dispõem de um modo de exibição dos resultados computacionais através de representações em mídia avançadas (*rich media*) como rich text, LaTex, bibliotecas para equações matemáticas, gráficos etc. A possibilidade de utilização de rich media em conjunto com código de programação executável e textos para esclarecimentos, fazem com que os jupyter notebooks forneçam uma experiência interativa para ajudar na compreensão de conteúdos.

Para atender o objetivo central da pesquisa, que consiste em investigar como o conceito DL pode ser empregado como ferramenta para auxílio no ensino-aprendizado da tecnologia, foi proposto um material instrucional sobre DL escrito totalmente em um jupyter notebook. Este consta com a implementação de um modelo de DNN para reconhecimento de imagens de números manuscritos, baseado na abordagem dos conceitos extraídos da comparação realizada na tabela 2. Desta maneira, para atingir este objetivo foram selecionadas as tecnologias para o desenvolvimento do material instrucional. As subseções deste capítulo destinam-se à especificação das tecnologias utilizadas, apresentação dos ambientes de desenvolvimento e estruturação do material instrucional.

[5] Documentos web que contém *live code* in-line intercalados com outras formas de representação de informação (JUPYTER, 2016).

5.1 TECNOLOGIAS

Para a elaboração do material instrucional sobre DL, fez-se necessário a utilização de tecnologias específicas. Como biblioteca responsável pela aplicação de DL foi definido o Tensorflow. Com isso, a linguagem de programação Python também foi escolhida, pelo fato de ser uma linguagem de fácil compreensão e ser suportada pela biblioteca Tensorflow. O ambiente selecionado para o desenvolvimento do material instrucional foi o Jupyter Notebook. Todas estas tecnologias são detalhadas a seguir.

5.1.1 TENSORFLOW

A API Tensorflow (o projeto Google Brain iniciou em 2011 para explorar o uso de DNNs) é uma interface para o desenvolvimento e representação de diversos algoritmos de aprendizado de máquina criado pelo Google, com lançamento para a comunidade em novembro de 2015. Nele foi desenvolvida a DistBelief, a primeira geração de um sistema distribuído para treinamento e inferência. Posteriormente, com base na experiência adquirida nas pesquisas com a DistBelief, foi criado o Tensorflow como a segunda geração do sistema. Em comparação com o DistBelief, Tensorflow é mais flexível, sua performance é melhor e ainda suporta diversos modelos em vários tipos de plataformas de hardware (MARTÍN et al., 2015).

Tensorflow utiliza grafos direcionados compostos por nodos para demonstrar os cálculos realizados. Possui suporte para execução de processos localmente ou distribuído em diversas máquinas diferentes. Ademais, Tensorflow foi desenvolvido para rodar em CPUs ou GPUs (nesta última, cálculos paralelos são executados mais rapidamente) e os sistemas operacionais suportados são MAC OS e LINUX (MARTÍN et al., 2015).

As linguagens de programação utilizadas no Tensorflow são Python, C e C++. Ser escrito em Python foi a primeira característica considerada na escolha da biblioteca para modelagem de DL deste trabalho. Além disso, como já citado anteriormente, Tensorflow é uma biblioteca lançada pelo Google e amplamente utilizada em seus projetos. Sendo assim, seu suporte no longo prazo é mais garantido, assim como a qualidade da documentação concedida (MARTÍN et al., 2015). Outro diferencial da biblioteca Tensorflow com relação às outras, é um programa separado chamado TensorBoard que possibilita a visualização de inúmeras informações sobre o modelo em execução através de gráficos e diagramas (TENSORFLOW, 2016).

5.1.2 LINGUAGEM DE PROGRAMAÇÃO PYTHON

A linguagem de programação Python foi criada por Guido van Rossum. Orientada a objetos é comparável à Perl, Ruby e Java. A utilização dessa linguagem deu-se pelo fato de ser a linguagem em que foi escrita a biblioteca Tensorflow, não obstante é importante ressaltar os seus benefícios: linguagem amplamente utilizada nas áreas acadêmica e comercial, possui uma sintaxe simples e de fácil compreensão e dispõe de uma ampla variedade de bibliotecas padrões disponíveis (PYTHON, 2016).

5.1.3 JUPYTER NOTEBOOK

O ambiente Jupyter Notebook é uma aplicação web que permite a criação de documentos contendo trechos de programação executáveis, equações, imagens, textos explicativos e gráficos chamados “notebooks”. Possui suporte a diversas linguagens de programação, como Python, R, Java, C, Julia, Lua, Ruby, dentre outras (JUPYTER, 2016).

O ambiente é composto por um dashboard, que permite visualizar todos os notebooks disponíveis, criar novos e abrir outros. Dentro dos notebooks existem células para programação que podem ser criadas e executadas separadamente através do menu do dashboard. Outro componente do Jupyter Notebook é o kernel da linguagem de programação: como padrão o kernel IPython está habilitado, mas há a opção de instalar outros kernels de outras linguagens de programação (JUPYTER, 2016).

Jupyter Notebooks são derivados do IPython, aplicação que Fernando Perez, na universidade do Colorado, iniciou seu desenvolvimento em 2001. Esta ferramenta foi melhorada e aperfeiçoada com novas funcionalidades e assim, originando seu sucessor. A figura 21 representa o ambiente de desenvolvimento e o dashboard do Jupyter (JUPYTER, 2016).

Figura 21 – Ambiente de desenvolvimento de um notebook Jupyter

Fonte: Elaborado pelo autor (2016)

5.1.4 DOCKER

Como a biblioteca Tensorflow suporta apenas os sistemas operacionais Linux e Mac OS, para a sua utilização no ambiente Windows faz-se necessário o uso de uma ferramenta para virtualização. Na página oficial do Tensorflow é recomendada a utilização da ferramenta Docker, uma plataforma de virtualização através de containers. Cada container dentro do Docker compartilha o mesmo sistema operacional virtualizado, porém cada container roda no sistema operacional como um processo separado (DOCKER, 2016).

5.2 PREPARAÇÃO DO AMBIENTE PARA DESENVOLVIMENTO

Para trabalhar com a biblioteca Tensorflow no Windows foi instalada uma ferramenta para virtualização. Na página oficial do Tensorflow é recomendada a utilização da ferramenta Docker, uma plataforma de virtualização através de containers. Primeiro faça o download da ferramenta no link: https://www.docker.com/products/docker#/windows.

Depois da instalação do Docker, selecione uma imagem do Tensorflow para a criação de um container. As imagens disponíveis do Tensorflow são:

- gcr.io/tensorflow/tensorflow: TensorFlow CPU binary image;
- gcr.io/tensorflow/tensorflow:latest-devel: CPU Binary image plus source code;
- gcr.io/tensorflow/tensorflow:latest-gpu: TensorFlow GPU binary image;
- gcr.io/tensorflow/tensorflow:latest-devel-gpu: GPU Binary image plus source code.

Com a imagem selecionada execute o comando para criação de um container com a imagem: *docker run -it -p 8888:8888 gcr.io/tensorflow/tensorflow*.

Também pode ser usado o comando run com o parâmetro "-d", ele fará com que o container permaneça rodando: *docker run -d -p 8888:8888 gcr.io/tensorflow/tensorflow*.

Abaixo seguem outros comandos auxiliares para a manipulação dos containers e máquinas virtuais.

- *Docker-machine start "nome da máquina"*: comando para iniciar a máquina virtual;
- *Docker-machine ls*: lista as máquinas virtuais. Comando necessário para visualizar as URLs associadas às máquinas, as URLs são utilizadas para acessar o notebook jupyter;
- *Docker ps -a*: comando para listar os containers criados;
- *Docker start "id_container"*: inicia a execução de um container parado.

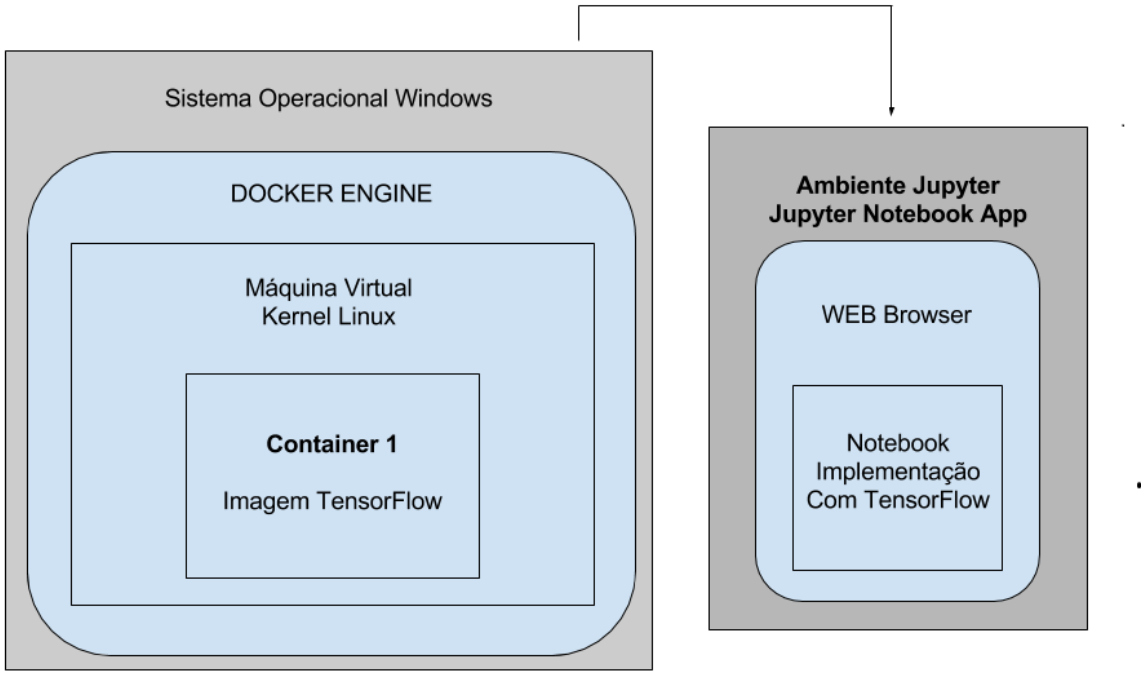
O ambiente Jupyter está inserido dentro da imagem do Tensorflow. Para acessá-lo basta colocar a URL da máquina virtual no browser, com a porta indicada na execução do comando "run", exemplo: http://192.168.99.100:8888/tree.

Para utilizar o powershell como terminal de comandos, ao invés do terminal padrão do docker basta rodar o comando abaixo no terminal do docker e depois rodar no powershell o comando que será gerado.

docker-machine env default --shell powershell

Esta estrutura de ambiente é ilustrada na imagem 22.

Figura 22 – Estrutura do ambiente no Windows



Fonte: Elaborado pelo autor (2016)

A instalação do ambiente (figura 23) no Linux é mais fácil, pois o Tensorflow já é suportado por ele, bastam poucos comandos no terminal do Linux. Para a instalação do Tensorflow sem suporte a GPU pode ser utilizado o Conda, um gerenciador de pacotes:

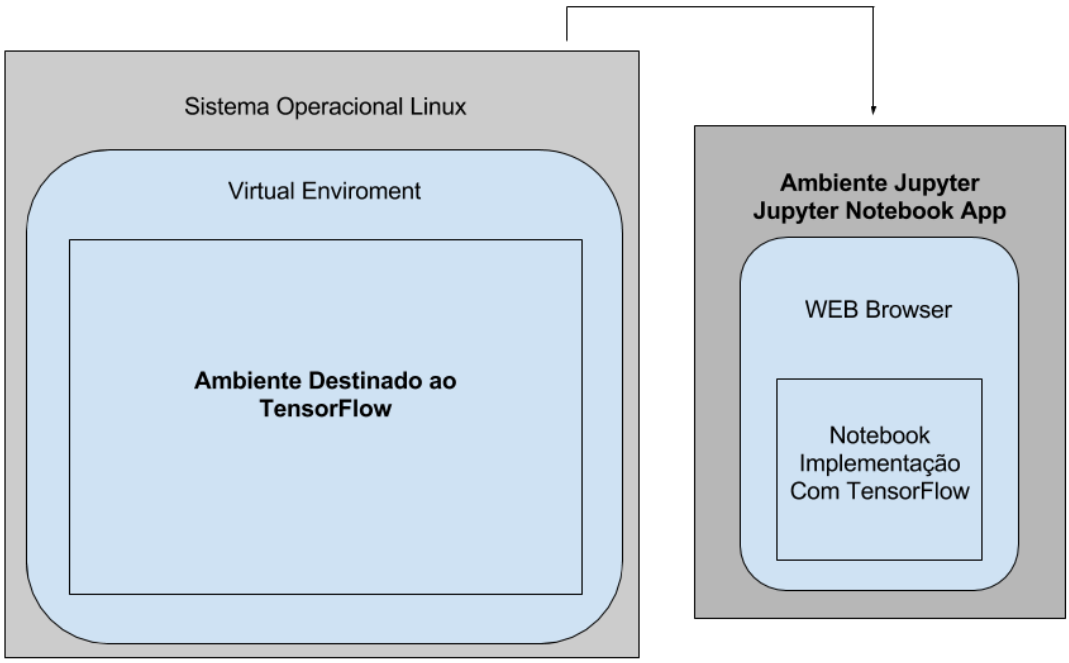
Ubuntu:~$ source activate tensorflow

(tensorflow) Ubuntu:~$ conda install jupyter

(tensorflow) Ubuntu:~$ jupyter notebook

A mensagem indicará o endereço que o notebook estará rodando: The Jupyter Notebook is running at: *http://localhost:8888/*

Figura 23 – Estrutura do ambiente no Linux



Fonte: Elaborado pelo autor (2016)

5.3 PROCESSO DE CRIAÇÃO DO MATERIAL INSTRUCIONAL

Para a criação do notebook como material instrucional sobre DL, inicialmente foi necessário definir qual modelo de rede neural que seria analisado. Decidiu-se pela implementação de uma CNN, o mesmo modelo empregado nos outros materiais estudados. A escolha do dataset também levou em consideração os trabalhos relacionados, ambos fazem uso do dataset MNIST[6]. Com isso, poderá ser realizada uma comparação entre todos eles e o mais importante, será possível enriquecer o conteúdo sobre esse tipo de rede neural e dataset.

Para que o material seja objetivo no que se refere às tecnologias envolvidas, as seguintes perguntas foram apontadas e servem como um guia para a progressão dos conteúdos:

- Qual a definição de deep learning?
- O que significa aprendizado em níveis de abstração?
- O que são as redes convolucionais e como funcionam?
- Como preparar o ambiente de desenvolvimento utilizando a biblioteca Tensorflow?
- Como realizar a implementação de um modelo de rede convolucional?

[6] http://yann.lecun.com/exdb/mnist/

- Quais os principais conceitos que envolvem a implementação de uma rede neural profunda?

No intuito de responder as perguntas acima, o notebook foi subdividido em algumas seções que abordam os seguintes conceitos:

1. Inicialmente é realizada uma introdução sobre **redes neurais**.
2. Os passos realizados **pelo algoritmo de treinamento gradiente descendente**.
3. Problema de **underfitting e overfitting**.
4. Em seguida uma definição de **deep learning** e o **aprendizado em diversos níveis de abstração**.
5. Dificuldades no treinamento de DNNs antes de 2006 (apresentação das técnicas de deep learning).
6. Após a introdução desses conteúdos é explicado o funcionamento de uma **rede convolucional**.
7. Descrição sobre a biblioteca **Tensorflow**.
8. Como preparar o **ambiente de desenvolvimento** para os sistemas operacionais Windows e Linux.
9. **Aplicações atuais** de deep learning.
10. A **implementação do modelo** de rede convolucional para reconhecimento de dígitos com o dataset MNIST.

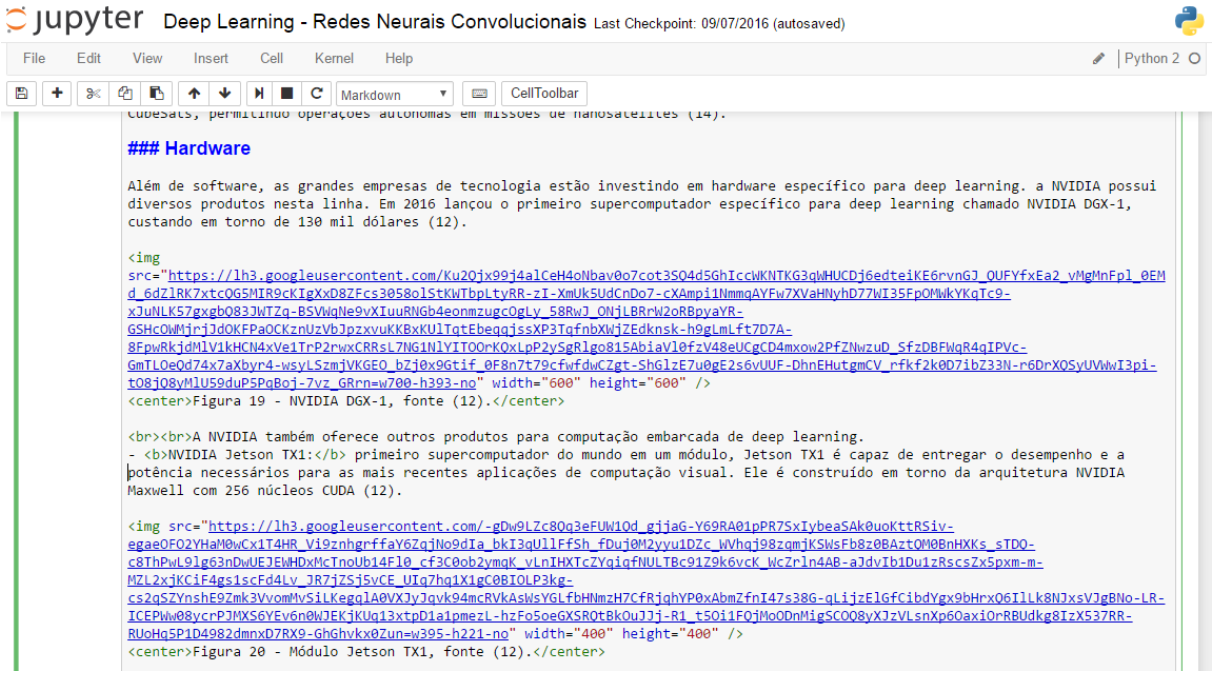
As formatações nos textos dentro das células do notebook são feitas através de códigos HTML. Já as imagens utilizadas como ilustração precisam ser hospedadas em algum serviço na nuvem, para que fiquem disponíveis a quem for abrir o notebook em seu computador. Assim, utilizou-se o serviço Google Fotos. As figuras 24, 25 e 26 demonstram como é realizada a formatação das células de texto do notebook:

Figura 24 – Células de texto formatadas dentro do notebook

Fonte: Elaborado pelo autor (2016)

A figura 25 ilustra a inclusão de uma imagem em uma célula de texto de um notebook jupyter.

Figura 25 – Células de texto formatadas dentro do notebook com link de imagem



Fonte: Elaborado pelo autor (2016)

A figura 26 ilustra o resultado da execução das células de texto.

Figura 26 – Resultado gerado pelas células de texto após sua execução

Fonte: Elaborado pelo autor (2016)

Desse modo, a parte inicial do notebook contempla a apresentação teórica dos conceitos a fim de familiarizar o leitor no tema abordado, mas principalmente para fazê-lo compreender como uma rede convolucional é construída. Após esta seção inicial, avançou-se para o desenvolvimento do modelo.

Na seção prática do notebook, consta a codificação da rede convolucional para o reconhecimento de imagens, com base no dataset MNIST, utilizando a biblioteca Tensorflow e a linguagem de programação Python. A intenção é detalhar os passos para criação da estrutura da rede neural (camadas, pesos sinápticos e biases, por exemplo), detalhar os elementos utilizados e suas finalidades, como por exemplo, funções de custo, funções de ativação e regularizadores. Além disso, explicar o propósito de cada classe do Tensorflow utilizada, pois somente lendo a referência da API no site oficial pode não ser suficiente para entender sua funcionalidade na prática.

Para atingir esses objetivos, dividiu-se o algoritmo em células, de modo que cada célula do notebook possa realizar uma pequena tarefa dentro do processo de criação e treinamento do modelo. Gerando sempre que possível uma saída com o resultado do processamento. Entre as células do notebook foram incluídas descrições explicando o que foi programado dentro das mesmas e a tarefa que cada classe do Tensorflow exerce no algoritmo.

Na seção a seguir será relatado o processo de criação e treinamento da CNN que resultou no modelo final utilizado no material instrucional.

5.3.1 MODELAGEM E TREINAMENTO DA CNN

Definido o tipo de rede neural o próximo passo é estabelecer o número de camadas iniciais da rede, que podem ser camadas convolucionais, camadas de pooling ou camadas totalmente conectadas. Sendo assim, a estrutura inicial escolhida para a CNN possuía duas camadas convolucionais, duas camadas de pooling e, por último, uma camada totalmente conectada (FC) seguida por mais uma camada FC com dez neurônios para gerar a classificação final dos números. Abaixo é retratada a arquitetura resultante da CNN:

1- **CAMADA DE ENTRADA.**
2- **CAMADA CONVOLUCIONAL1.**
3- **CAMADA DE POOLING1.**
4- **CAMADA CONVOLUCIONAL2.**
5- **CAMADA DE POOLING2.**
6- **CAMADA TOTALMENTE CONECTADA.**
7- **CAMADA DE SAÍDA FC.**

Além das camadas que irão compor a rede neural devem ser definidos valores para alguns parâmetros, relacionados a seguir:

- Algoritmo de treinamento;
- Função de ativação de neurônio;
- Função de custo;
- Taxa_aprendizagem;
- Beta_regularizacao
- Ciclos_treinamento;
- Tamanho_stride_convolucao;
- Tamanho_kernel_camada_convolucional;
- Tamanho_stride_pool;
- Tamanho_kernel_camada_pool;
- Tamanho_batch_validacao;

- Tamanho_mini_batch;
- Padding;
- Quantidade_filtros_imagem;
- Valor_taxa_dropout;
- Numero_camadas_pooling;
- Numero_neurônios_camada_fc;
- Tamanho_imagem_em_pixels;
- Quantidade_labels_de_imagem;
- Dimensao_imagem;
- Total_imagens_validacao;
- Total_imagens_treinamento;

Nas primeiras simulações com a CNN os parâmetros que envolvem a regularização não foram utilizados (como o regularizador L2 e o dropout). Estes métodos de regularização foram empregados posteriormente para poder avaliar o seu impacto na melhora da acurácia do modelo. O algoritmo foi subdividido em funções com o intuito de estruturar o código e favorecer a compreensão das operações. As funções podem ser separadas em três grupos principais: 1) para a definição da estrutura da rede; 2) especificação do método de treinamento 3) execução e validação do treinamento. As outras funções compreendem plotagem de gráficos, imagens e saídas das camadas da rede. Os próximos parágrafos descrevem com mais detalhes cada uma destas funções divididas subcapítulos.

5.3.1.1 VARIABLE_PESOS E VARIABLE_BIASES

As funções “variable_pesos” e “variable_biases” (figura 27) criam variáveis de sistema para os pesos e biases da rede dentro do diagrama computacional do TF. O formato dos tensors é passado como parâmetro e é especificado o modo de inicialização das variáveis, podendo ser com zeros, constantes ou valores randômicos.

Figura 27 – Funções para criação de variáveis para os pesos e biases

```
def variable_pesos(shape):
  inicializacao = tf.random_normal(shape, stddev=0.1)
  return tf.Variable(inicializacao)

def variable_biases(shape):
  inicializacao = tf.random_normal(shape, stddev=0.1)
  return tf.Variable(inicializacao)
```

Fonte: Elaborado pelo autor (2016)

5.3.1.2 CRIACAMADACONV E CRIACAMADAPOOLING

As funções “criaCamadaConvolucional” e “criaCamadaPooling” criam as camadas convolucionais e de pooling da rede. A função “criaCamadaConv” utiliza-se de uma função do TF para criar a camada convolucional chamada “tf.nn.conv2d”, no qual recebe dois parâmetros que indicam os valores de entrada (x) e os valores dos pesos (w). Nesta função pode ser especificado o tamanho do stride da convolução e o modo de padding.

A função “criaCamadaPooling” utiliza-se da função do TF para criar a camada de pooling chamada “tf.nn.max_pool”, que emprega a técnica do retorno do maior valor do kernel analisado. Esta função recebe como parâmetro os valores de entrada da camada e nela podem ser especificados o tamanho do kernel a ser analisado, o tamanho do stride e o modo de padding.

A API do TF contêm outras opções para a realização da convolução ou do pooling como, por exemplo, as funções “tf.nn.depthwise_conv2d” e “tf.nn.separable_conv2d” para convoluções, “tf.nn.avg_pool” e “tf.nn.agv_pool3d” para pooling.

A codificação destas funções é expressa na figura 28.

Figura 28 – Funções para criação da camada convolucional e de pooling

```
def criaCamadaConvolucional(x, W):
  return tf.nn.conv2d(x, W,strides=[1, tamanho_stride_convolucao,
         tamanho_stride_convolucao, 1], padding='SAME')

def criaCamadaPooling(x):
  return tf.nn.max_pool(x,ksize=[1, tamanho_kernel_camada_pool,
                          tamanho_kernel_camada_pool, 1],
                          strides=[1, tamanho_stride_pool,
                          tamanho_stride_pool, 1], padding='SAME')
```

Fonte: Elaborado pelo autor (2016)

5.3.1.3 FORMATO_CAMADA_CONVOLUCIONAL

A função "formato_camada_convolucional" (figura 29) é utilizada para definir o formato da camada convolucional, recebe como parâmetros o formato do kernel (*width* e *height*), o número dos canais de entrada e a quantidade de filtros.

Figura 29 – Função que define o formato da camada convolucional

```
def formato_camada_convolucional(kernel_width, kernel_height, canais,
                                 quantidade_filtros):
  return variable_pesos([kernel_width, kernel_height, canais,
                         quantidade_filtros])
```

Fonte: Elaborado pelo autor (2016)

5.3.1.4 RECUPERAR OS FILTROS DAS IMAGENS

As funções getFiltros e plotarFiltros (figura 30) foram criadas para a plotagem das imagens geradas como saídas pelas camadas convolucionais e de pooling. A função getFiltros recebe como parâmetro a camada da CNN e a imagem que deve ser filtrada pela camada.

Figura 30 – Função para recuperação dos filtros de imagem

```
def getFiltros(layer, imagem):
  unidades = layer.eval(session=sessao, feed_dict={
              x_pixels_imagem_entrada: np.reshape(imagem, [1,
  tamanho_imagem_em_pixels], order='F'), taxa_dropout: 1.0})
  plotarFiltros(unidades)

def plotarFiltros(unidades):
    filtros = unidades.shape[3]
    if filtros > 18:
        filtros = 18
    print('Numero de filtros ' + str(filtros))
    fig2 = plt.figure(1, figsize=(20, 20))
    for i in xrange(0, filtros):
        plt.subplot(5, 4, i + 1)
        plt.title('Filtro ' + str(i))
        plt.imshow(unidades[0, :, :, i], interpolation="nearest",
                   cmap="gray", aspect='auto')
    plt.tight_layout(pad=0.1, w_pad=2, h_pad=0.5)
```

Fonte: Elaborado pelo autor (2016)

5.3.1.5 FUNÇÕES DE CUSTO

As funções de custo utilizadas baseiam-se na cross-entropy. Foram criadas três funções para o cálculo do custo, a cross_entropy, coss_entropy_clip e softmax_cross_entropy, conforme a figura 31. A função de custo softmax_cross_entropy possui o diferencial de ter embutida na classe da biblioteca a implementação da operação de softmax da saída da rede neural, bem como o cálculo do custo. Para as outras funções é necessário especificar o cálculo do custo, onde primeiramente é computado o log da do tensor de saída (tf.log(p_y_conv)) multiplicado pelo tensor com as predições corretas (y_labels_imagem_entrada). O resultado é um tensor do mesmo formato. Por fim, realiza-se a soma dos elementos resultantes da multiplicação no índice 1 do tensor e aplica-se a média. O sinal negativo no cálculo é para garantir Custo > 0.

Figura 31 – Funções de custo utilizadas

```
def cross_entropy(p_y_conv):
    f_cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_labels_imagem_entrada
                        * tf.log(p_y_conv)
                        , reduction_indices=[1]))
    return f_cross_entropy

def cross_entropy_clip(p_y_conv):
    f_cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_labels_imagem_entrada
                         * tf.log(tf.clip_by_value(p_y_conv,1e-30,100))
                         , reduction_indices=[1]))
    return f_cross_entropy

def softmax_cross_entropy(p_y_conv):
    f_cross_entropy =
      tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
                       p_y_conv,y_labels_imagem_entrada))
    return f_cross_entropy
```

Fonte: Elaborado pelo autor (2016)

5.3.1.6 ALGORITMOS DE TREINAMENTO

O Tensorflow reúne diversos algoritmos de treinamento, nesta rede neural foram testados previamente 5 algoritmos (figura 32), no qual cada um recebe como parâmetro a taxa de aprendizado e a função de custo que precisa ser minimizada. Nas simulações apresentadas adiante, inicialmente será testado o algoritmo tf.train.GradientDescentOptimizer.

Figura 32 – Algoritmos de treinamento implementados

```
def AdamOptimizer(custo, p_taxa_aprendizagem):
    return tf.train.AdamOptimizer(p_taxa_aprendizagem).minimize(custo)

def AdadeltaOptimizer(custo, p_taxa_aprendizagem):
    return tf.train.AdadeltaOptimizer(p_taxa_aprendizagem).minimize(custo)

def MomentumOptimizer(custo, p_taxa_aprendizagem):
    return tf.train.MomentumOptimizer(p_taxa_aprendizagem,
0.9).minimize(custo)

def RMSPropOptimizer(custo, p_taxa_aprendizagem):
    return tf.train.RMSPropOptimizer(p_taxa_aprendizagem).minimize(custo)

def SGDOptimizer(custo, p_taxa_aprendizagem):
   return
     tf.train.GradientDescentOptimizer(p_taxa_aprendizagem).minimize(custo)
```

Fonte: Elaborado pelo autor (2016)

5.3.1.7 VALIDAÇÃO DA REDE

A função “validaCNN” (figura 33) é utilizada para validação e geração da acurácia da rede através das imagens de teste. Ela recebe como parâmetro o tamanho do lote de imagens que serão carregadas para o Tensorflow.

Figura 33 – Função para a validação da CNN

```
def validaCNN(ptamanho_batch_validacao):
    if ptamanho_batch_validacao < total_imagens_validacao:
        teste_batch = mnist.test.next_batch(tamanho_batch_validacao)
        feed_dict= {x_pixels_imagem_entrada: teste_batch[0],
                    y_labels_imagem_entrada: teste_batch[1],
                    taxa_dropout: 1.0}
    else:
        feed_dict= {x_pixels_imagem_entrada: mnist.test.images,
                    y_labels_imagem_entrada: mnist.test.labels,
                    taxa_dropout: 1.0}
    print("Acuracia nas imagens de teste
          %g"%calcula_acuracia.eval(session=sessao, feed_dict=feed_dict))
```

Fonte: Elaborado pelo autor (2016)

5.3.1.8 ESTRUTURA DA CNN

Nesta seção será descrito o processo de criação da estrutura da rede neural convolucional, ou seja, a determinação das camadas convolucionais, de pooling e totalmente conectadas, a maneira com que as camadas são interpoladas, a função de ativação dos neurônios, o dropout da rede e como a camada de saída irá gerar o resultado da predição. Este processo será descrito passo a passo, pois envolvem muitos detalhes importantes com relação ao modelo.

Primeiramente são criadas duas variáveis do sistema TF, uma para os pesos das conexões e outra para os biases dos neurônios. Deve ficar claro que essas variáveis são específicas do sistema TF (tf.variable) e diferenciam-se das variáveis convencionais criadas para utilização dentro do algoritmo em Python. São objetos que serão manipulados dentro do diagrama computacional do TF, as demais variáveis não.

A variável responsável pelos pesos da primeira camada tem o formato: [tamanho_kernel_camada_conv, tamanho_kernel_camada_conv, canais = 1, quantidade_filtros_imagem]. O número de canais é igual a um, porque na primeira camada só existe um conjunto de valores de entrada, os pixels da imagem. Tendo em vista que a camada convolucional terá um kernel de 5x5 e 32 filtros, serão um total de 5x5x32 valores de pesos iniciais, ou 800.

A variável que manterá os biases dos neurônios terá o formato de acordo com a quantidade de filtros de imagem da camada convolucional. Lembrando que é necessário apenas um bias por neurônio e os neurônios que compõem um filtro compartilham os mesmos pesos e biases.

Antes de criar a primeira camada convolucional, é preciso definir um tensor para receber os valores de entrada da camada baseado no placeholder já criado com o tamanho total de pixels da imagem. Esse tensor terá o formato das imagens do dataset, 28X28 com a última dimensão representando os canais de cores, neste caso somente uma por não se tratar de imagens coloridas. A função responsável por redimensionar o tensor é a “tf.reshape”. Na figura 34 é ilustrada a criação destes objetos.

Figura 34 – Definição de variáveis e placeholders iniciais

```
W_convolucional_1 =
  formato_camada_convolucional(tamanho_kernel_camada_convolucional,
  tamanho_kernel_camada_convolucional, 1, quantidade_filtros_imagem)

b_convolucional_1 = variable_biases([quantidade_filtros_imagem])

x_imagem_entrada = tf.reshape(x_pixels_imagem_entrada, [-1,
                              dimensao_imagem,dimensao_imagem,1])
```

Fonte: Elaborado pelo autor (2016)

Com os placeholders e variáveis devidamente criadas, o próximo passo é a definição das camadas da CNN. A primeira camada é uma convolucional, nela é utilizada a função "criaCamadaConv", passada como parâmetro para o objeto que aplica a função de ativação dos neurônios. Na figura 35 a função sigmoid (tf.nn.sigmoid) foi escolhida para a ativação, sendo aplicada a cada elemento existente no tensor. Em seguida é somada a matriz de biases. Para a criação da camada de pooling, o processo é mais simples, basta passar para a função "criaCamadaPooling" a camada convolucional anterior.

Figura 35 – Definição de variáveis e placeholders iniciais

```
h_camada_conv_oculta_1 =
  tf.nn.sigmoid(criaCamadaConvolucional(x_imagem_entrada,
    W_convolucional_1) + b_convolucional_1)
h_camada_pooling_1 = criaCamadaPooling(h_camada_conv_oculta_1)
```

Fonte: Elaborado pelo autor (2016)

Para adicionar outra camada convolucional devem ser criadas novas variáveis para os pesos e biases, já para os valores de entrada da camada não é preciso criar um novo placeholder. A saída da camada anterior será a entrada da próxima camada. Assim, utiliza-se novamente a função "criaCamadaConv" e a função "tf.nn.sigmoid" para a ativação dos neurônios. Nota-se que é passado o tensor "h_camada_pooling_1" como valor do primeiro parâmetro da função "criaCamadaConv". Desta maneira é feita a ligação entre as camadas da rede. A figura 36 apresenta a codificação da segunda camada convolucional.

Figura 36 – Criação da segunda camada convolucional e pooling

```
W_convolucional_2 =
  formato_camada_convolucional(tamanho_kernel_camada_convolucional,
                               tamanho_kernel_camada_convolucional,
                               quantidade_filtros_imagem,
                               quantidade_filtros_imagem * 2)
b_convolucional_2 = variable_biases([quantidade_filtros_imagem * 2])
h_camada_conv_oculta_2 =
  tf.nn.sigmoid(criaCamadaConvolucional(h_camada_pooling_1,
                                        W_convolucional_2) +
                                        b_convolucional_2)
h_camada_pooling_2 = criaCamadaPooling(h_camada_conv_oculta_2)
```

Fonte: Elaborado pelo autor (2016)

Tendo em vista que o objetivo da CNN é a classificação de imagens de dígitos manuscritos, ao final da rede deve existir uma camada capaz de gerar o resultado da predição, para esta tarefa é empregada uma camada softmax. Desse modo, após a última camada de pooling são adicionadas camadas FC (totalmente conectadas ou do inglês *full connected*), para fazer o processamento de toda a imagem e por fim, como saída da rede, uma camada softmax.

Para a criação destas camadas finais, no primeiro passo foi selecionada a dimensão da última camada da rede até então (neste caso h_camada_pooling_2) que retorna um tensor, essa informação será útil para gerar as próximas camadas. A dimensão foi armazenada na variável pool_shape (figura 37). Com o valor do pool_shape foram criadas as variáveis para os pesos e bisases dos neurônios da primeira camada FC. A variável para os pesos possui o formato da dimensão da saída da camada anterior. Por exemplo, se a última camada for um pooling com saída de dimensão 7x7 através de 64 filtros, a matriz de pesos terá o formato [7x7x64, numero_neuronios_camada_fc]. A variável para os biases é criada com o mesmo formato do número total de neurônios da camada FC.

Figura 37 – Criação das variáveis para a primeira camada FC

```
pool_shape = h_camada_pooling_2.get_shape()

W_camada_fc1 = variable_pesos([int(pool_shape[1] * pool_shape[1] *
pool_shape[3]), numero_neuronios_camada_fc])
b_camada_fc1 = variable_biases([numero_neuronios_camada_fc])
```

Fonte: Elaborado pelo autor (2016)

Para realizar a ligação entre a camada FC e a última camada de pooling, é preciso redimensionar o tensor de retorno do pooling. Desta forma, o tensor redimensionado torna-se um array sequenciado com o total de valores 7x7x64 (64 filtros de dimensão 7x7). Tendo as variáveis para os pesos e biases definidas pode-se criar a camada FC. No entanto, há um detalhe a mais na criação desse tipo de camada, antes de aplicar a função sigmoid aos neurônios, efetua-se a multiplicação entre as matrizes dos pesos e valores de entrada. A função "tf.nn.conv2d" usada para as camadas convolucionais já faz este trabalho, todavia, para as camadas FC este cálculo deve ser feito manualmente. Através da utilização da função "tf.matmul" a API torna a tarefa de multiplicação entre os tensors contendo os pesos e valores de entrada mais prática. Aplicada a multiplicação, basta passar o resultado para a função de ativação e somar com os biases para ter a primeira camada totalmente conectada pronta. A codificação da primeira camada FC é mostrada na figura 38.

Figura 38 – Criação da primeira camada FC

```
h_camada_pooling_2_flat = tf.reshape(h_camada_pooling_2, [-1,
                            int(pool_shape[1] * pool_shape[1] *
                            pool_shape[3])])
h_camada_fc1 = tf.nn.sigmoid(tf.matmul(h_camada_pooling_2_flat,
                  W_camada_fc1) + b_camada_fc1)
taxa_dropout = tf.placeholder(tf.float32)
h_camada_fc1_dropout = tf.nn.dropout(h_camada_fc1, taxa_dropout)
```

Fonte: Elaborado pelo autor (2016)

A segunda camada FC recebe como entrada a saída da camada FC anterior (figura 39), assim não é necessário redimensionar o tensor como foi feito na criação da primeira camada FC.

Figura 39 – Criação da segunda camada FC

```
W_camada_fc2 = variable_pesos([numero_neuronios_camada_fc,
                  numero_neuronios_camada_fc])
b_camada_fc2 = variable_biases([numero_neuronios_camada_fc])
h_camada_fc2 = tf.nn.sigmoid(tf.matmul(h_camada_fc1_dropout, W_camada_fc2)
  + b_camada_fc2)
h_camada_fc2_dropout = tf.nn.dropout(h_camada_fc2, taxa_dropout)
```

Fonte: Elaborado pelo autor (2016)

O processo de criação da camada de saída da rede (exibido na figura 40) é semelhante ao processo das camadas FC, a variável para a matriz de pesos é gerada com o formato

(número de neurônios da camada anterior ; quantidade de labels da imagem). A quantidade de labels também define o número de biases dos neurônios. O detalhe que diferencia esta etapa é a troca da função de ativação. Ao invés de utilizar a função sigmoid é utilizada a função softmax, gerando as probabilidades proporcionais aos labels conforme a predição realizada pela rede.

Figura 40 – Criação da terceira e última camada de classificação

```
W_softmax_fc_3 = variable_pesos([numero_neuronios_camada_fc,
                    quantidade_labes_de_imagem])
b_softmax_fc3 = variable_biases([quantidade_labes_de_imagem])
y_camada_saida_final = tf.nn.softmax(tf.matmul(h_camada_fc2_dropout,
                           W_softmax_fc_3) + b_softmax_fc3)
```

Fonte: Elaborado pelo autor (2016)

Com as funções acima implementadas juntamente com a definição da estrutura da CNN, a conclusão do digrama computacional do TF depende da formalização da função de custo que será computada, o algoritmo de treinamento e do método para calcular a acurácia do modelo.

Figura 41 – Finalização do diagrama computacional para a CNN

```
funcao_custo = cross_entropy(y_camada_saida_final)
otimizador = SGDOptimizer(funcao_custo, taxa_aprendizagem)
predicoes_finais = tf.equal(tf.argmax(y_camada_saida_final,1),
tf.argmax(y_labels_imagem_entrada,1))
calcula_acuracia = tf.reduce_mean(tf.cast(predicoes_finais, tf.float32))
```

Fonte: Elaborado pelo autor (2016)

Como pode ser visto na figura 41 na simulação de exemplo foi definida a função de custo cross_entropy passando como parâmetro a saída da rede neural (*y_camada_saida_final*). O otimizador escolhido foi o “tf.train.GradientDescentOptimizer” (função SGDOptimizer), onde se passa a função de custo que deve ser minimizada e a taxa de aprendizagem. Para avaliar a acurácia da rede utiliza-se a função “tf.equal”, que compara a igualdade entre dois valores. Neste caso, cada valor correspondendo ao índice de probabilidade mais alta em uma dimensão de um tensor, selecionado com a função “tf.argmax”. A função “tf.argmax” é aplicada no tensor que contêm os labels das imagens e no tensor com as probabilidades de saída da rede. “tf.equal” gera como retorno uma lista de valores booleanos. Com a lista de valores booleanos gerados, aplica-se um cast na lista, para gerar uma relação de binários. Este processo de cast é necessário para o emprego da função

“tf.reduce_mean” que computa a média entre uma lista de valores para a geração da porcentagem de acurácia.

5.3.1.8 REGULARIZAÇÃO L2

Para aplicar a regularização L2 são seguidos três passos (exibidos na figura 42): 1) calcular soma dos quadrados de todos os pesos da rede através da função "tf.nn.l2_loss"; 2) multiplicar o resultado do cálculo anterior pelo fator de regularização (beta) dividido pelo número das imagens de treinamento sobre o tamanho do mini-batch; 3) adicionar à função de custo o termo de regularização gerado no passo 2.

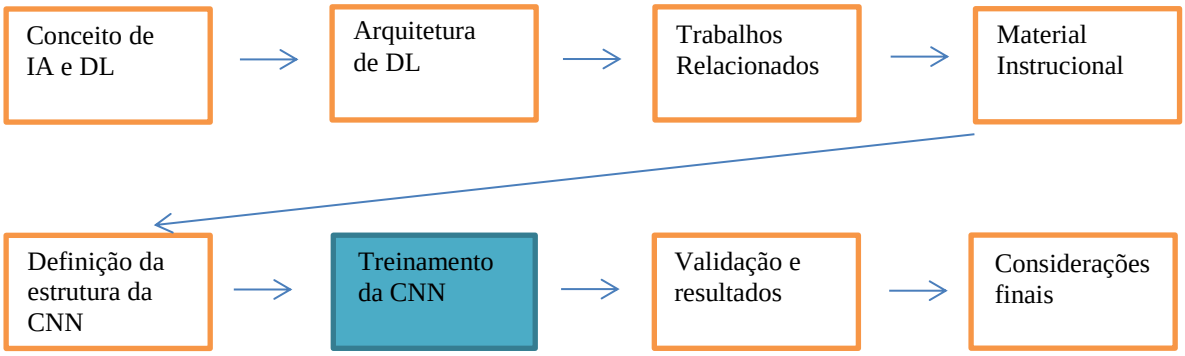
Figura 42 – Aplicação da regularização L2

```
regularizadores = (tf.nn.l2_loss(W_camada_fc1) +
                   tf.nn.l2_loss(W_camada_fc2) +
                   tf.nn.l2_loss(W_softmax_fc_3))
funcao_custo += regularizadores * (beta_regularizacao /
                  (total_imagens_treinamento / tamanho_mini_batch))
```

Fonte: Elaborado pelo autor (2016)

Após a finalização da estruturação da CNN, o próximo estágio procura realizar simulações com a estrutura criada. Etapa assinalada em azul na figura 43.

Figura 43 – Fluxograma da pesquisa



Fonte: Elaborado pelo autor (2016).

5.3.2 SIMULAÇÕES

Este subcapítulo apresenta as simulações realizadas com a CNN, no qual se buscou demonstrar os problemas recorrentes no treinamento de DNNs e alguns recursos disponíveis na biblioteca TF para a otimização dos treinamentos. Através das simulações executadas também foram analisados quais componentes e recursos contribuem na obtenção de uma melhor acurácia.

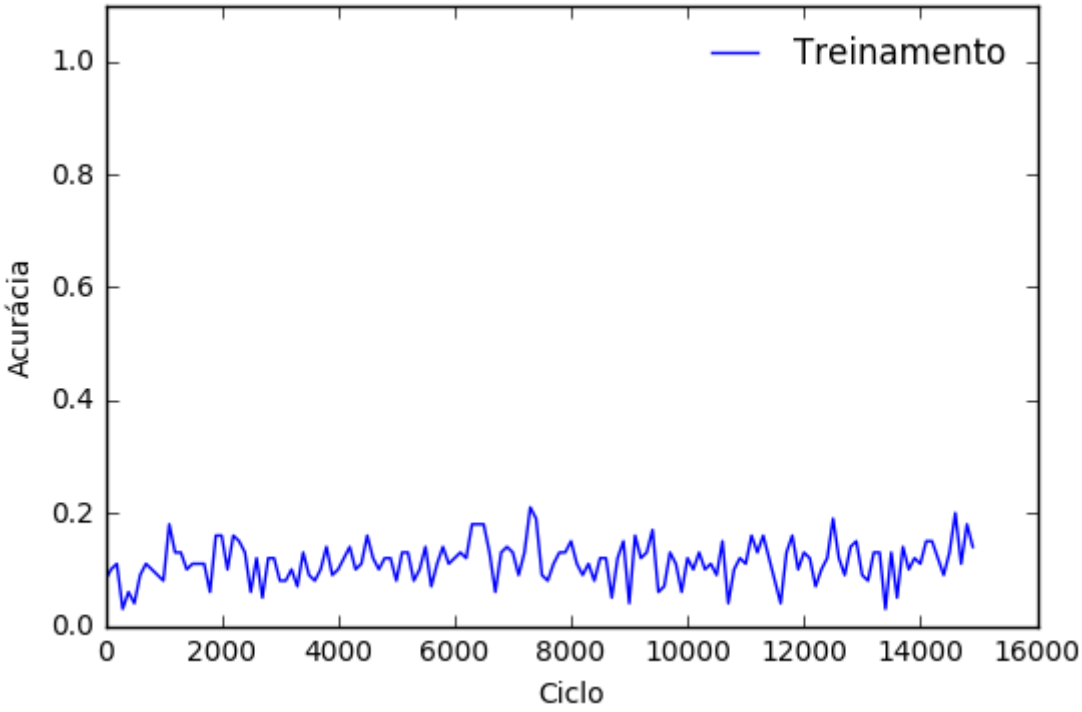
Para cada simulação descrita, serão relacionados os parâmetros escolhidos e o seu resultado final, que consiste no percentual de acerto da rede sobre as imagens de teste contidas no dataset MNIST. Além disso, para cada simulação serão exibidos dois gráficos que computam a acurácia e o custo para facilitar a análise.

As simulações partiram dos seguintes valores de parâmetros:

- Algoritmo de treinamento: SGD;
- função de ativação de neurônio: sigmoid;
- função de custo: cross-entropy;
- taxa_aprendizagem: 0.0001;
- beta_regularização: não utilizado;
- ciclos_treinamento: 16000;
- tamanho_stride_convolução: 1;
- tamanho_kernel_camada_conv: 5;
- tamanho_kernel_camada_pool: 2;
- tamanho_batch_validacao: 5000;
- tamanho_mini_batch: 100;
- padding: SAME;
- quantidade_filtros_imagem: 32;
- valor_taxa_dropout: não utilizado;
- numero_camadas_pooling: 2;
- numero_neuronios_camada_fc: 1024;
- tamanho_imagem_em_pixels: int((mnist.test.images[0].shape)[0]) #28 pixels;
- quantidade_labels_de_imagem: int(mnist.test.labels[0].size) #10 labels;
- dimensao_imagem: 28;
- total_imagens_validacao: mnist.test.images.shape[0] #55000 imagens;
- total_imagens_treinamento: mnist.train.images.shape[0] #10000 imagens;

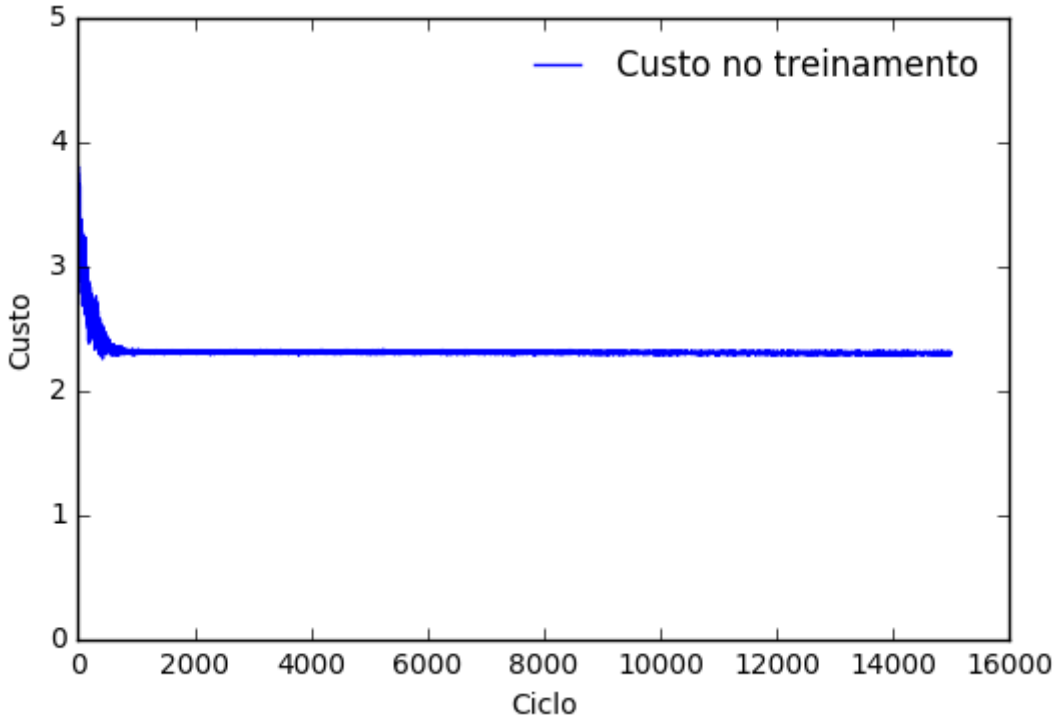
Na primeira simulação buscou-se parametrizar a rede para demonstrar o problema de vanishing nos gradientes pela utilização da função de ativação sigmoid. Ambos os gráficos demonstram a ocorrência do problema, não ocorrendo uma evolução da acurácia durante o treinamento. A acurácia da rede permaneceu na faixa de < 20% durante todos os ciclos com acurácia final da rede em 9,8% (gráficos 1 e 2). O número de ciclos também precisou ser diminuído pela metade do número padrão utilizado nas outras simulações porque o tempo de processamento aumenta conforme os gradientes diminuem. Alterando a função de ativação para ReLU e aumentando o número de ciclos para 30000, obteve-se uma acurácia final de 94,56% em 2 horas e 42 minutos de processamento (gráficos 3 e 4). Os gráficos demonstram a diferença.

Gráfico 1 – Acurácia no treinamento 1



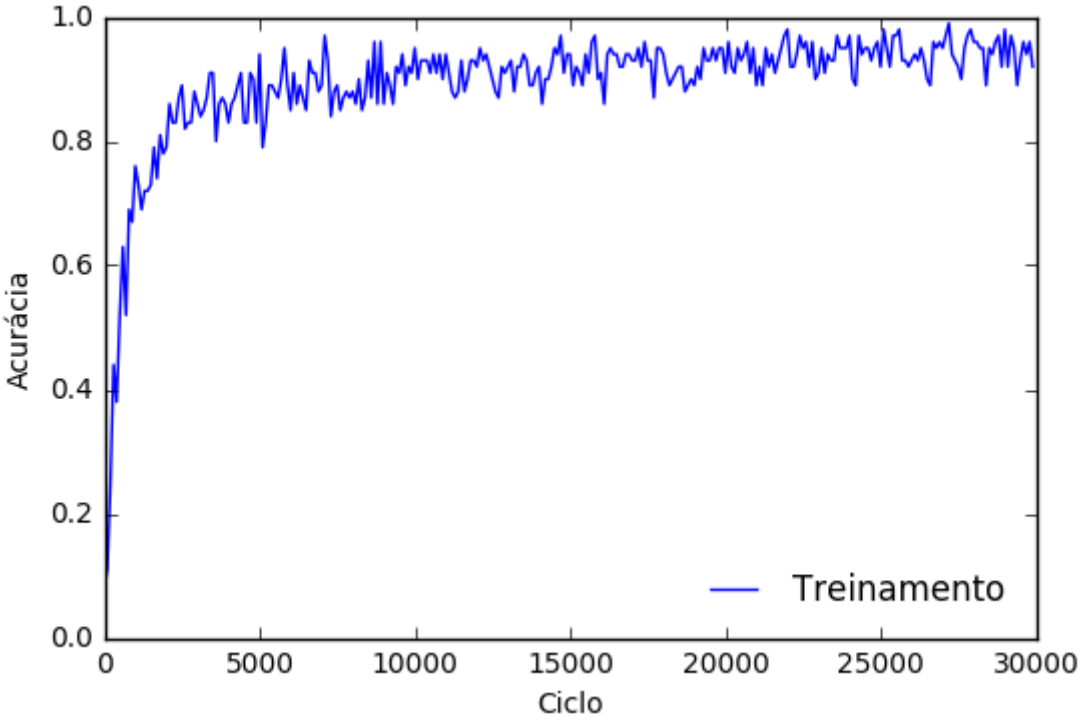
Fonte: Elaborado pelo autor (2016)

Gráfico 2 – Custo no treinamento 1



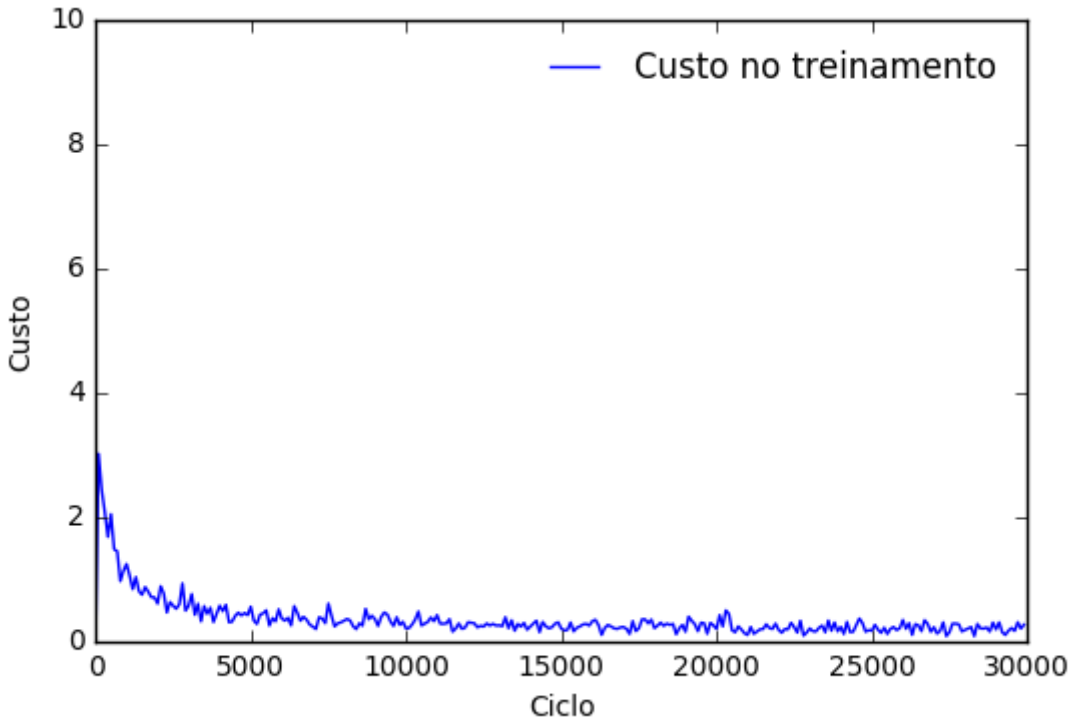
Fonte: Elaborado pelo autor (2016)

Gráfico 3 – Acurácia no treinamento 2



Fonte: Elaborado pelo autor (2016)

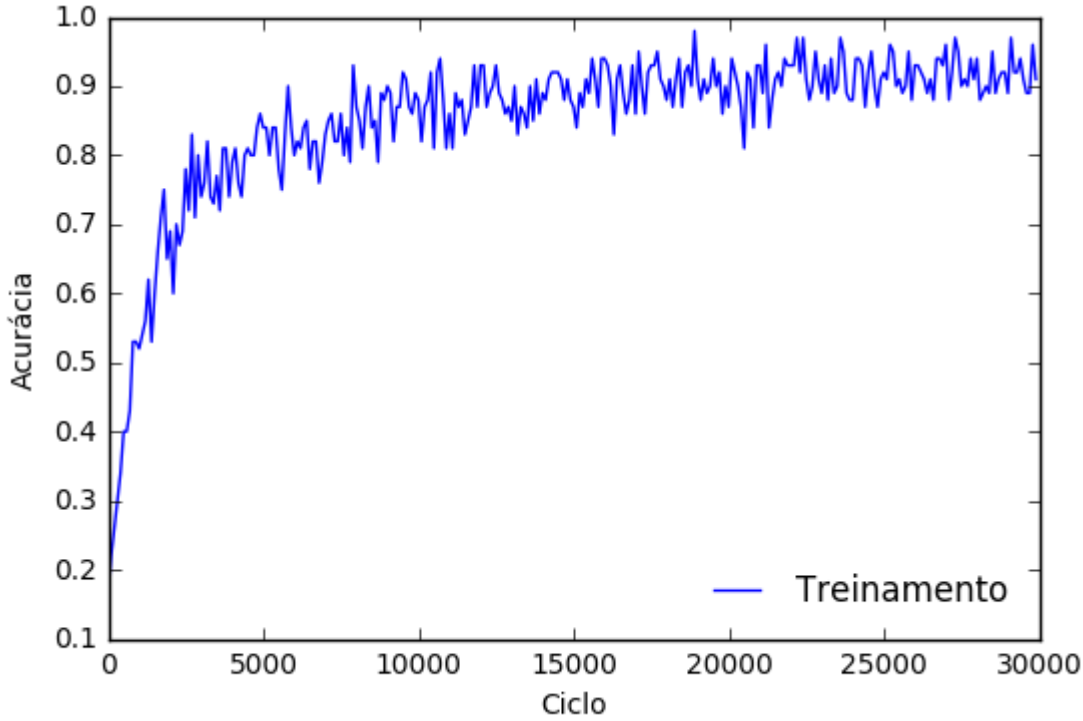
Gráfico 4 – Custo no treinamento 2



Fonte: Elaborado pelo autor (2016)

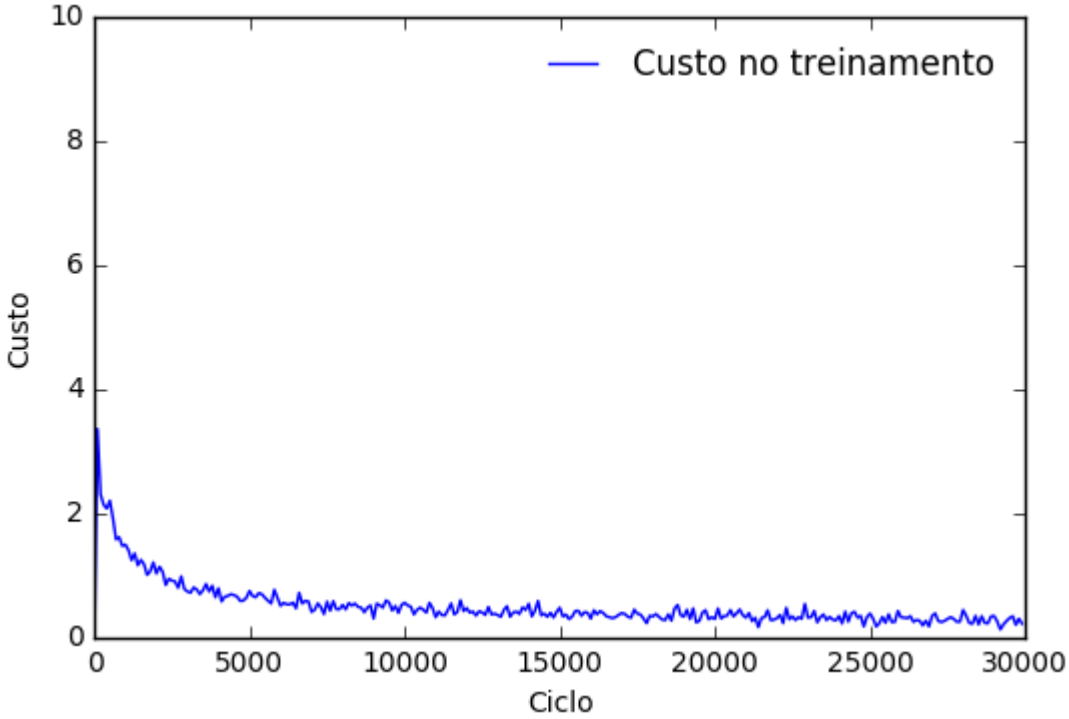
Outra função de ativação testada foi a tangente hiperbólica, essa apresentou um resultado um pouco inferior em comparação à ReLU. Com resultado de 92,15% de acerto durante 2 horas e 44 minutos. Nos gráficos 5 e 6 pode-se perceber uma demora um pouco maior no aumento da acurácia nos primeiros ciclos de treinamento para atingir o nível de 60%, bem como uma curva mais suave na diminuição do custo.

Gráfico 5 – Acurácia no treinamento 3



Fonte: Elaborado pelo autor (2016)

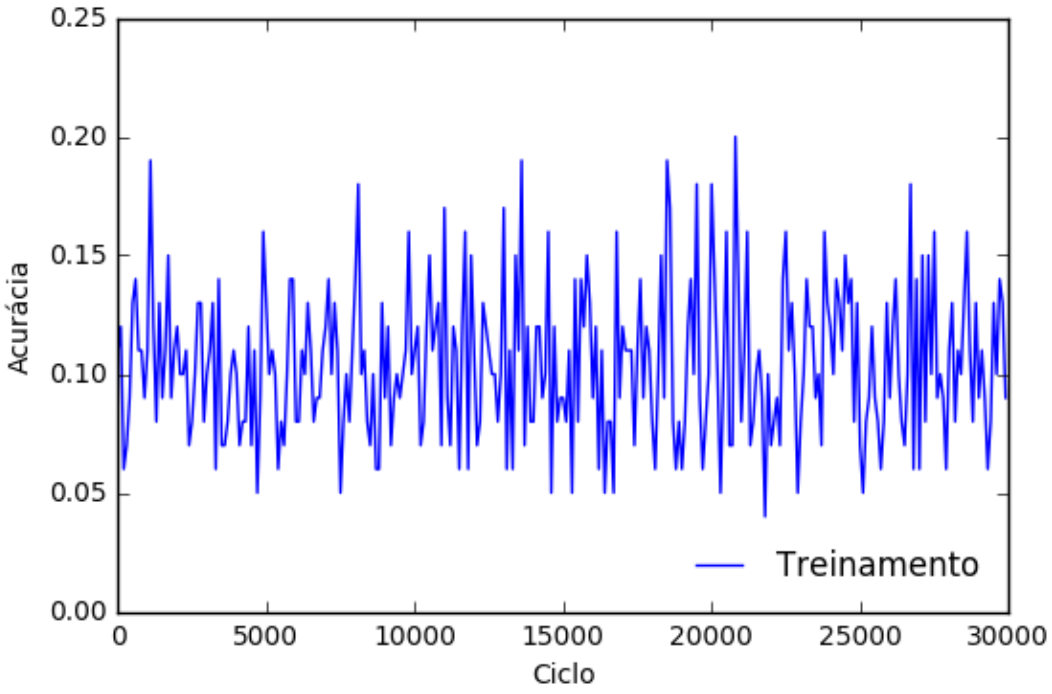
Gráfico 6 – Custo no treinamento 3



Fonte: Elaborado pelo autor (2016)

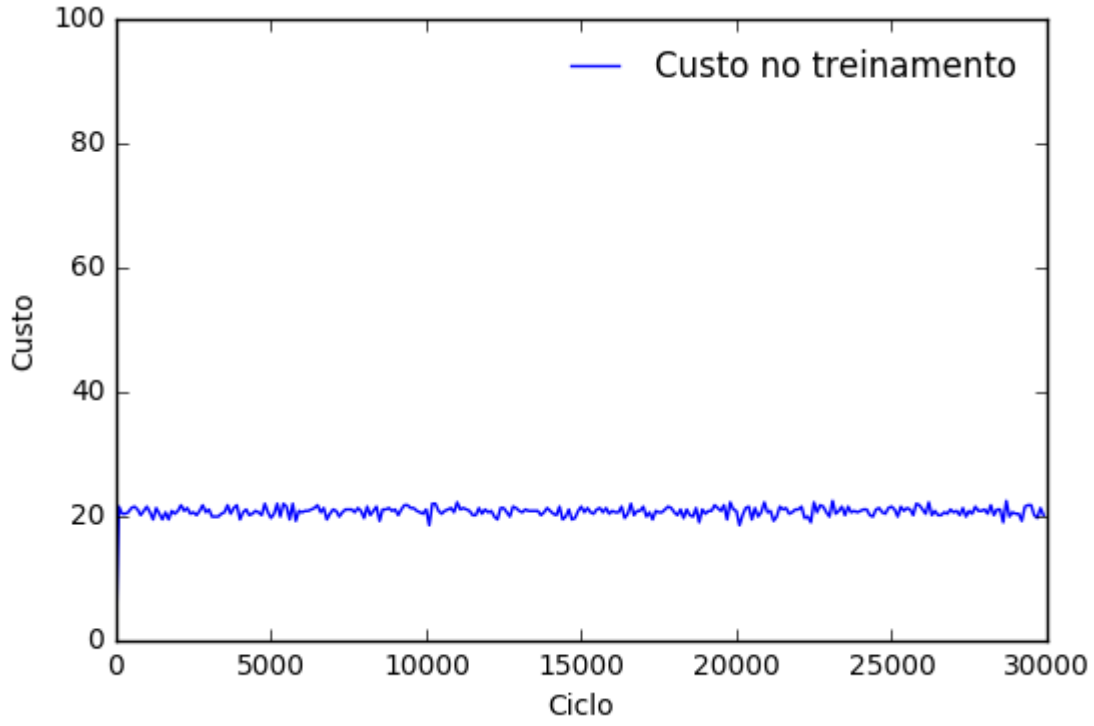
Até então, a taxa de aprendizado utilizado foi 0.0001 e quanto menor a taxa de aprendizado mais demorado torna-se o treinamento da CNN. No entanto, é atingido um melhor resultado com um tempo maior de processamento. Um fator ligado diretamente à taxa de aprendizado é o processo do algoritmo do gradiente descendente para a tentativa de obtenção da melhor direção ao encontro do resultado ótimo. Quando a taxa de aprendizado é muito alta, o algoritmo encontra dificuldades de encontrar a direção correta pelo fato do salto (taxa de aprendizado) ser grande. Para demonstrar esse problema, a taxa de aprendizado foi elevada para 0.1 e os gráficos abaixo foram gerados apresentando novamente uma acurácia em torno de 20%, custo sem diminuição e uma acurácia nas imagens de teste de 10,03% (gráficos 7 e 8).

Gráfico 7 – Acurácia no treinamento 4



Fonte: Elaborado pelo autor (2016)

Gráfico 8 – Custo no treinamento 4

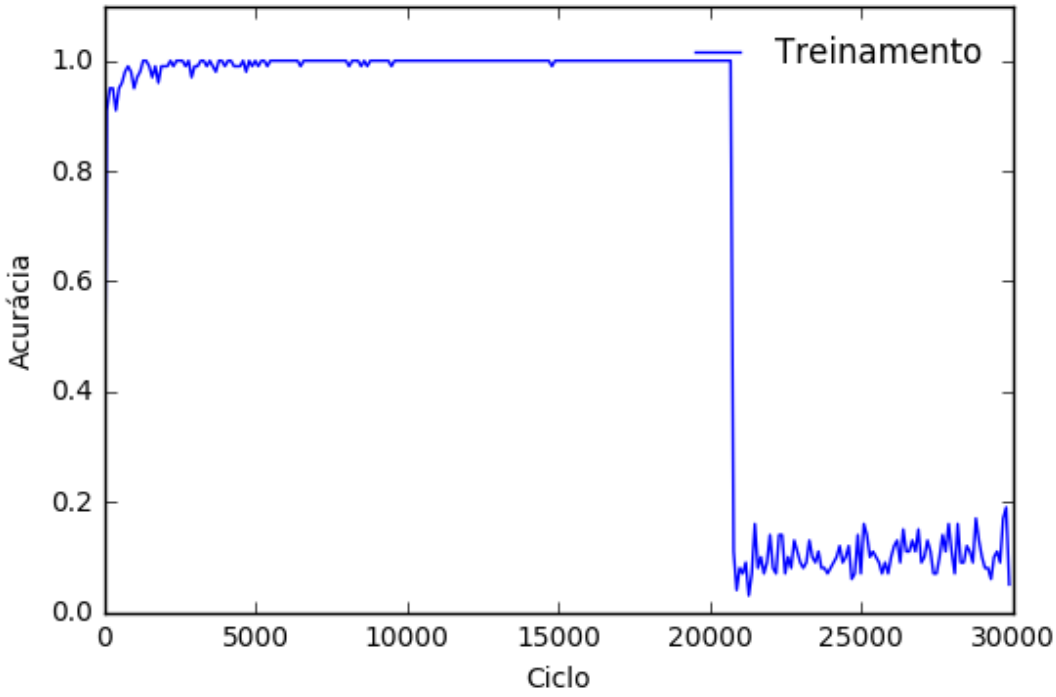


Fonte: Elaborado pelo autor (2016)

Nestas primeiras simulações o algoritmo gradiente descendente estocástico (tf.train.GradientDescentOptimizer) foi o responsável pelo treinamento da rede. Contudo a biblioteca TF oferece outras opções de algoritmos como: Adadelta, Adam, Adagrad, Momentum dentre outros. Em artigo publicado (KINGMA, BA, 2015), o algoritmo Adam obteve melhores resultados nos testes apresentados, portanto serão realizadas simulações com esse algoritmo mantendo os demais parâmetros com os mesmos valores da última simulação.

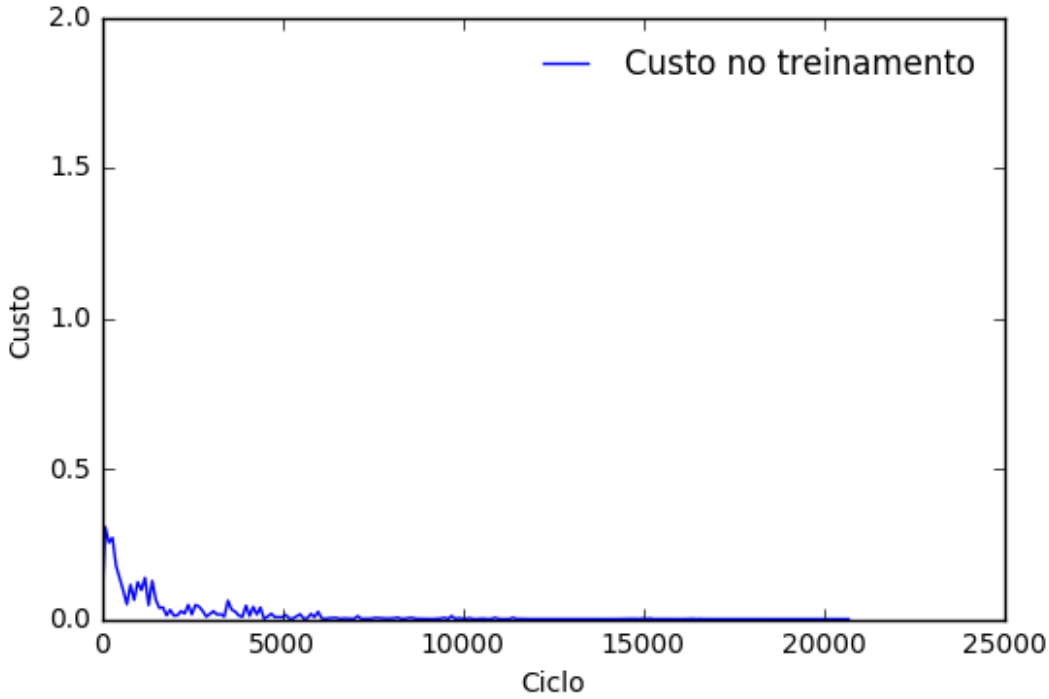
Na primeira execução os seguintes resultados foram obtidos (gráficos 9 e 10):

Gráfico 9 – Acurácia no treinamento 5



Fonte: Elaborado pelo autor (2016)

Gráfico 10 – Custo no treinamento 5

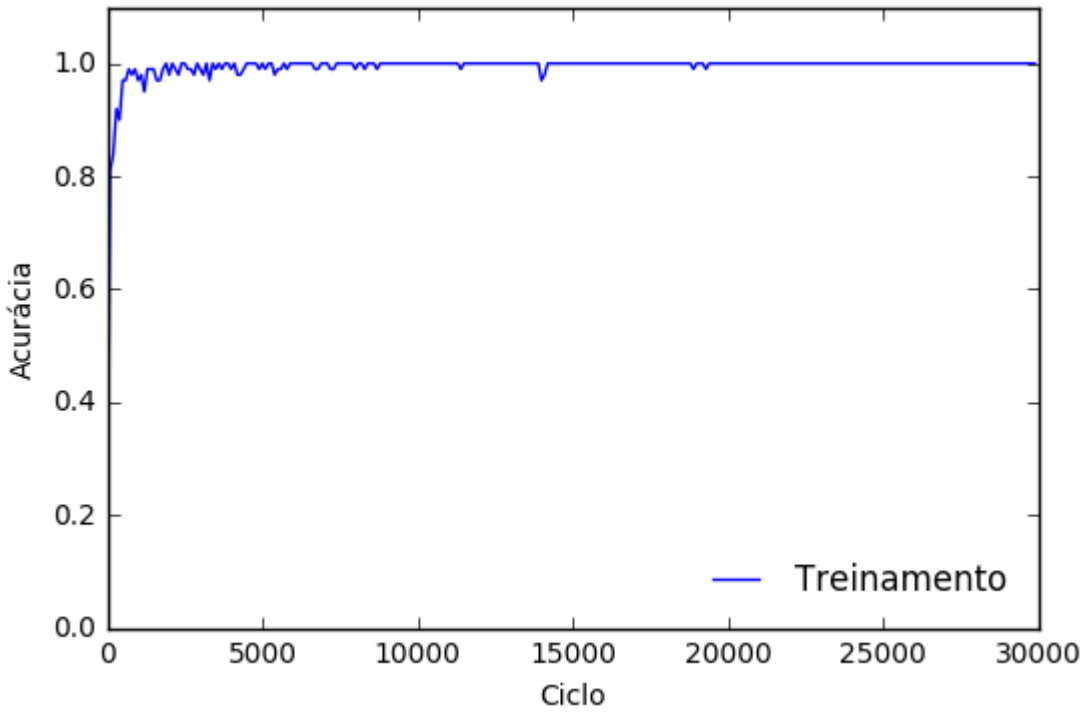


Fonte: Elaborado pelo autor (2016)

Pode-se observar no gráfico da acurácia do treinamento a existência de uma queda brusca no seu valor após vinte mil ciclos de treinamento. Isso ocorre durante o cálculo da

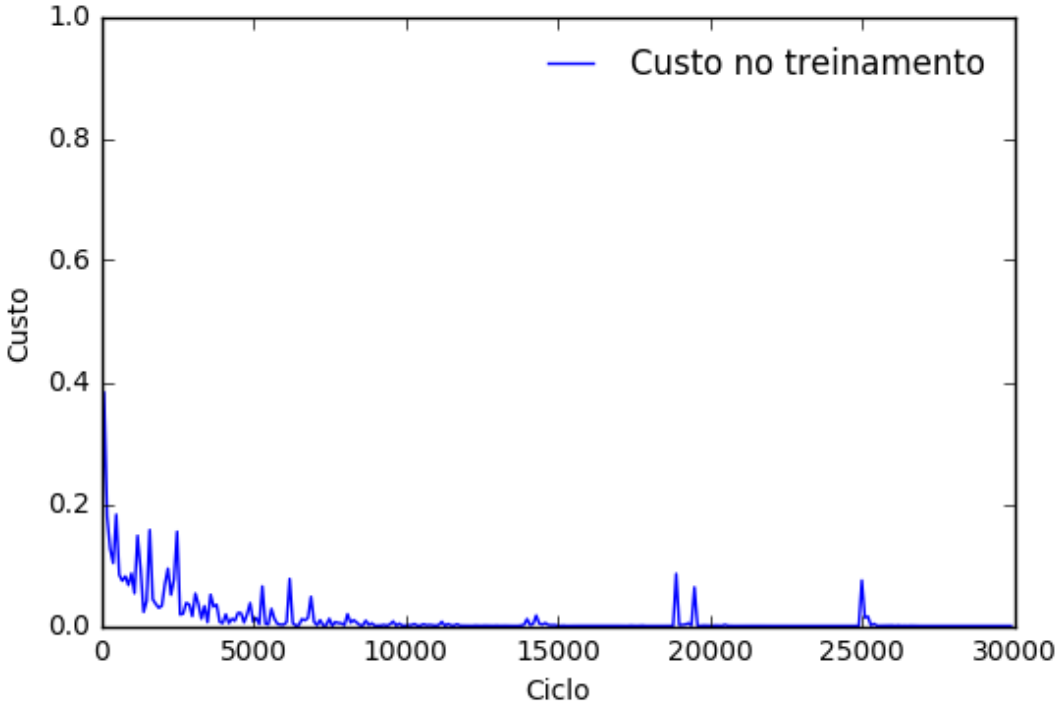
função de custo onde, para o algoritmo Adam, são geradas probabilidades zeradas como saída e com isso, ao aplicar o log, é gerado NAN como resultado, pois não existe log de zero. Para contornar este problema pode ser aplicada uma função chamada tf.clip_by_value. Esta função analisa o valor: se este for maior ou menor que os valores máximo e mínimo estipulados, os extremos são retornados. Aplicando esta técnica de clip os seguintes resultados foram obtidos (gráficos 11 e 12).

Gráfico 11 – Acurácia no treinamento 6



Fonte: Elaborado pelo autor (2016)

Gráfico 12 – Custo no treinamento 6

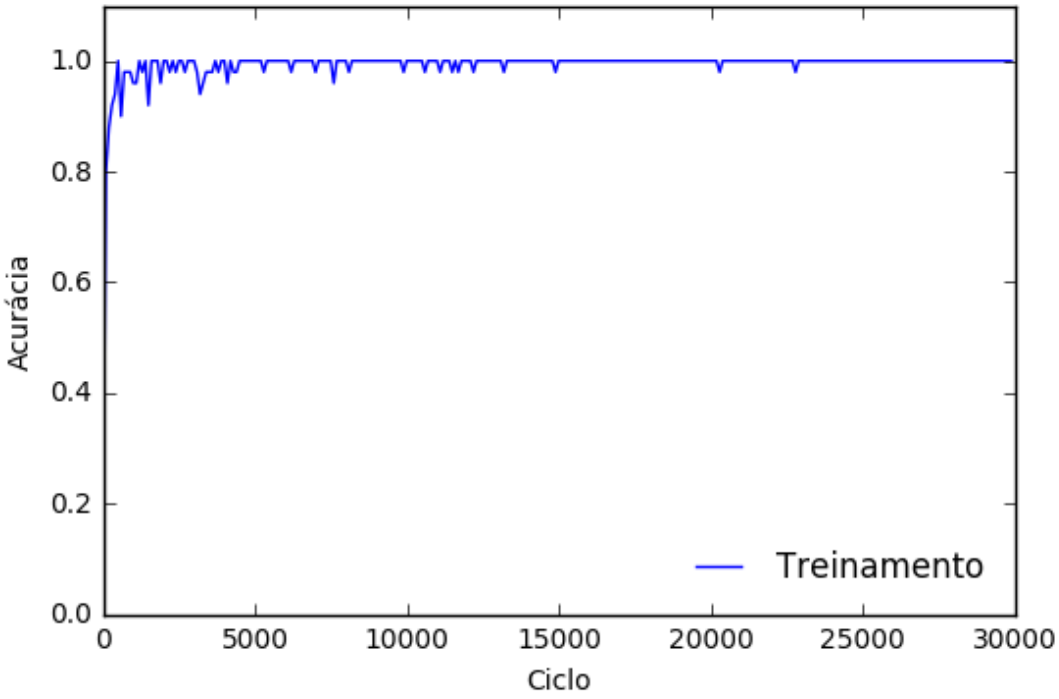


Fonte: Elaborado pelo autor (2016)

O treinamento percorreu todos os ciclos sem a ocorrência do erro no cálculo da função de custo, garantindo uma acurácia final de 99,1% durante 2 horas e 46 minutos de processamento. Em comparação com o resultado obtido utilizando o algoritmo de treinamento SGD (em torno de 94%), o algoritmo Adam demonstra maior eficiência.

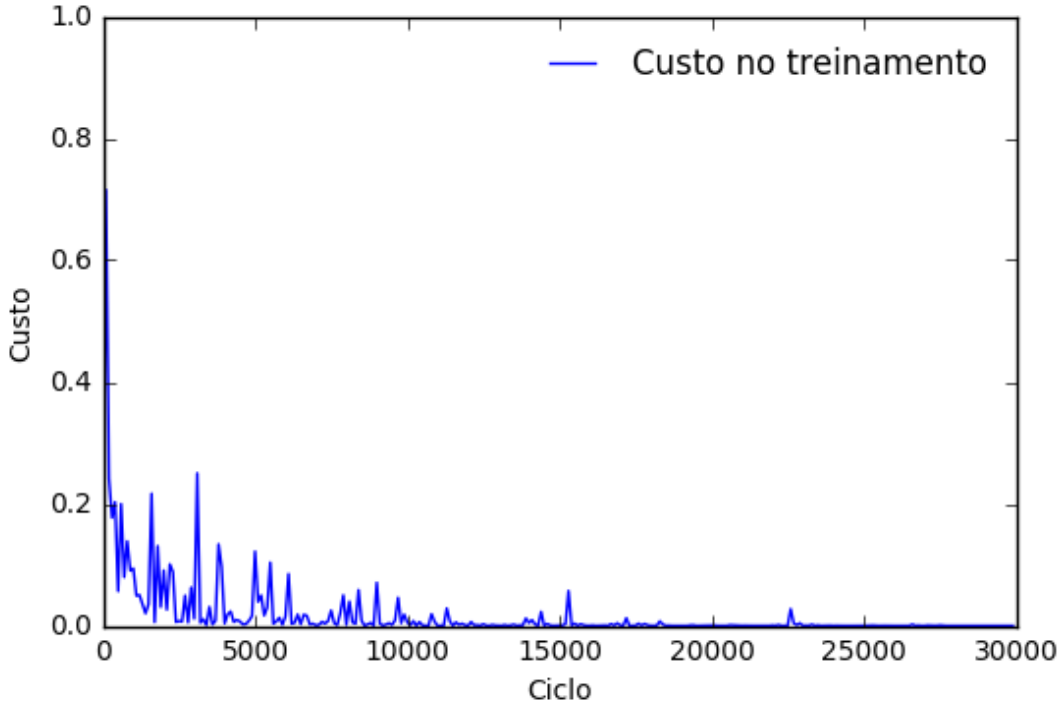
Nota-se que o tamanho do mini-batch de treinamento foi iniciado com 100 imagens; ou seja, cada ciclo de treinamento contêm 100 imagens e ainda foi atingido mais de 95% de acerto na CNN em ambos os algoritmos de treinamento testados. Todavia, se o tamanho do mini-batch for diminuído, significa que os pesos da rede são ajustados com maior frequência e os recursos computacionais exigidos também baixam, portanto, optou-se por reduzir pela metade o tamanho do mini-batch e analisar o resultado da alteração (gráficos 13 e 14).

Gráfico 13 – Acurácia no treinamento 8



Fonte: Elaborado pelo autor (2016)

Gráfico 14 – Custo no treinamento 8



Fonte: Elaborado pelo autor (2016)

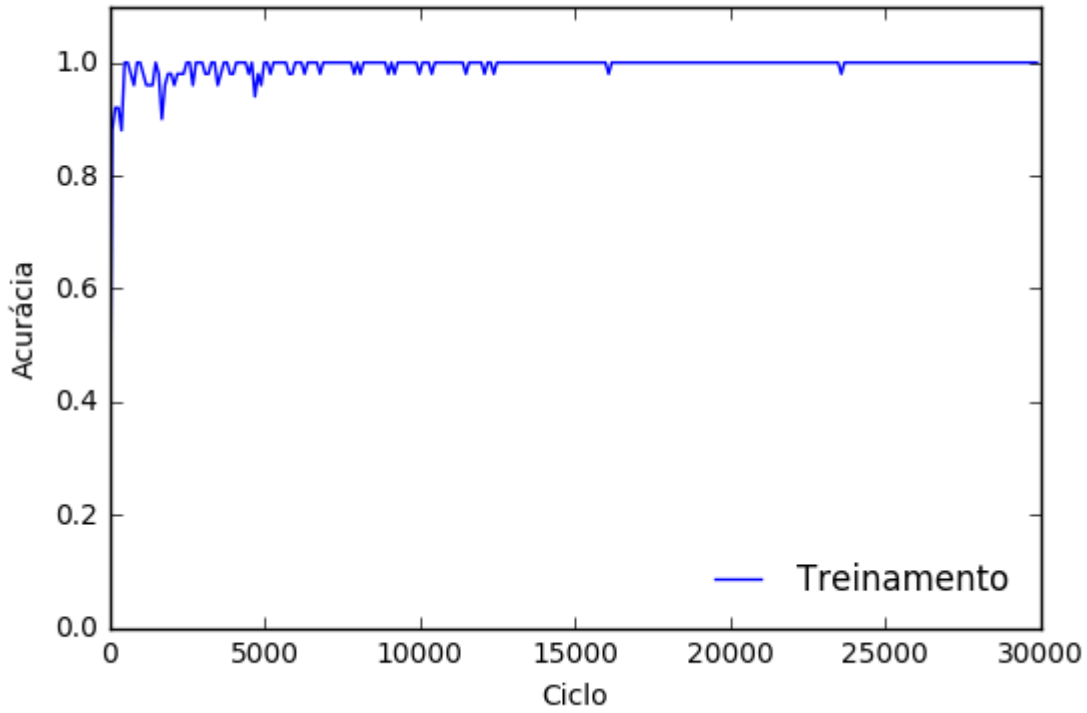
Nos gráficos 13 e 14 pode-se visualizar o andamento do treinamento com um mini-batch reduzido. O gráfico de custo apresenta uma maior oscilação nos primeiros ciclos, ou seja, os pesos entre as camadas foram atualizados mais vezes.

Até este momento, a taxa de aprendizagem parametrizada possui o valor de 0,0001 e conforme já discutido ela influencia diretamente, obviamente, o aprendizado da rede neural. Quando a taxa for muito alta a rede neural irá mapear mais rápido os dados de treinamento fornecidos, porém a rede pode não assimilar todas as características necessárias para atingir uma generalização satisfatória. É preciso então ponderar o valor da taxa de aprendizagem de acordo com o objetivo desejado. Na simulação com a CNN, modificando a taxa de aprendizagem para 0,00001 obteve-se uma acurácia nas imagens de teste de 98,16% em 1 hora e 26 minutos, podendo constatar que com a redução não houve melhora no resultado final e somente tornou o aprendizado mais lento.

Uma das opções existentes para reduzir o overffiting da rede é uma melhor escolha de função de custo. Por exemplo, nas simulações realizadas foi utilizada função cross-entropy ao invés da MSE, ou erro médio quadrático. Porém, existem outras opções para redução do overfitting como o dropout e a regularização L2, ambas as técnicas foram testadas na CNN para serem analisados os seus efeitos nos gráficos.

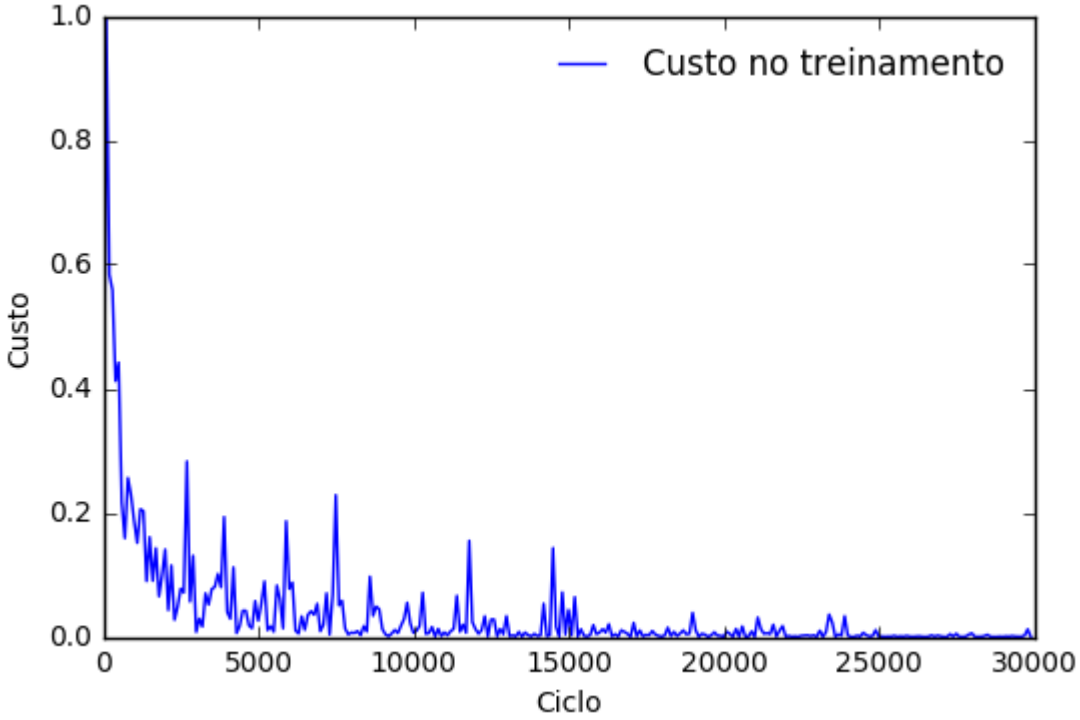
A primeira técnica aplicada foi o dropout, esta foi introduzida somente nas camadas finais da rede que são totalmente conectadas, isto é, possuem um número elevado de parâmetros em comparação com as camadas convolucionais. A taxa de dropout utilizada foi de 50%, segundo demonstram os gráficos 15 e 16.

Gráfico 15 – Acurácia no treinamento 9



Fonte: Elaborado pelo autor (2016)

Gráfico 16 – Custo no treinamento 9

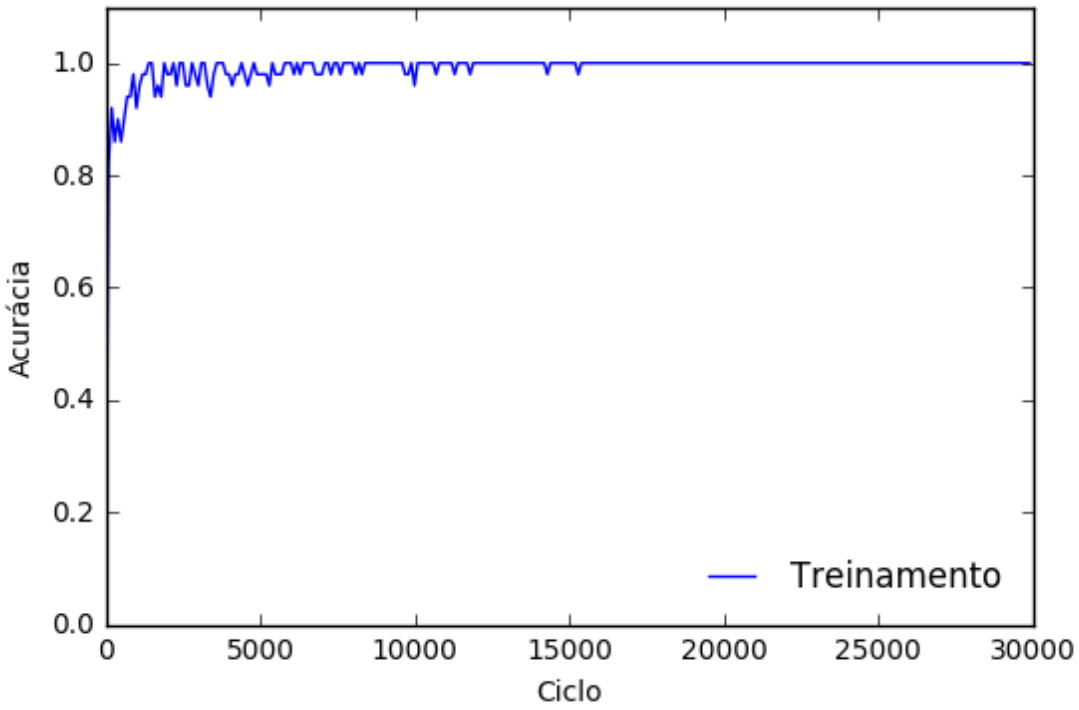


Fonte: Elaborado pelo autor (2016)

Claramente pode ser observada no gráfico de custo uma mudança no comportamento da rede durante o treinamento com o dropout, com o aumento do custo até o ciclo de número 25000. Isto faz com que a rede neural seja atualizada ao longo de quase todos os ciclos de treinamento refletindo em uma maior generalização. Fornecendo uma acurácia final de 99,27% nas imagens de teste, um aumento de 0,17% na taxa de acerto final durante 1 hora e 28 minutos de processamento.

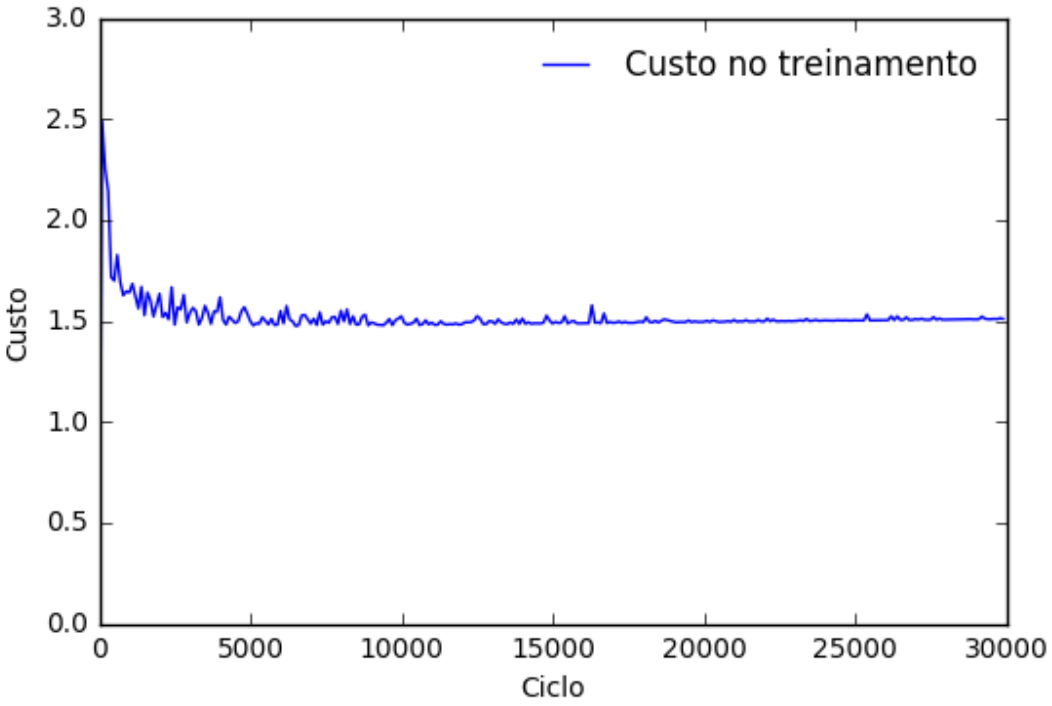
A outra técnica empregada para a redução do overfitting foi a regularização L2. Foram realizadas duas simulações alterando o valor do parâmetro de regularização em 0,1 e 0,001. Na primeira simulação com o parâmetro de regularização em 0,1 pode-se perceber o aumento na escala de valores da função de custo, gerando ao final do treinamento uma acurácia de 99,35% em 1 hora e 28 minutos de processamento representados nos gráficos 17 e 18.

Gráfico 17 – Acurácia no treinamento 11



Fonte: Elaborado pelo autor (2016)

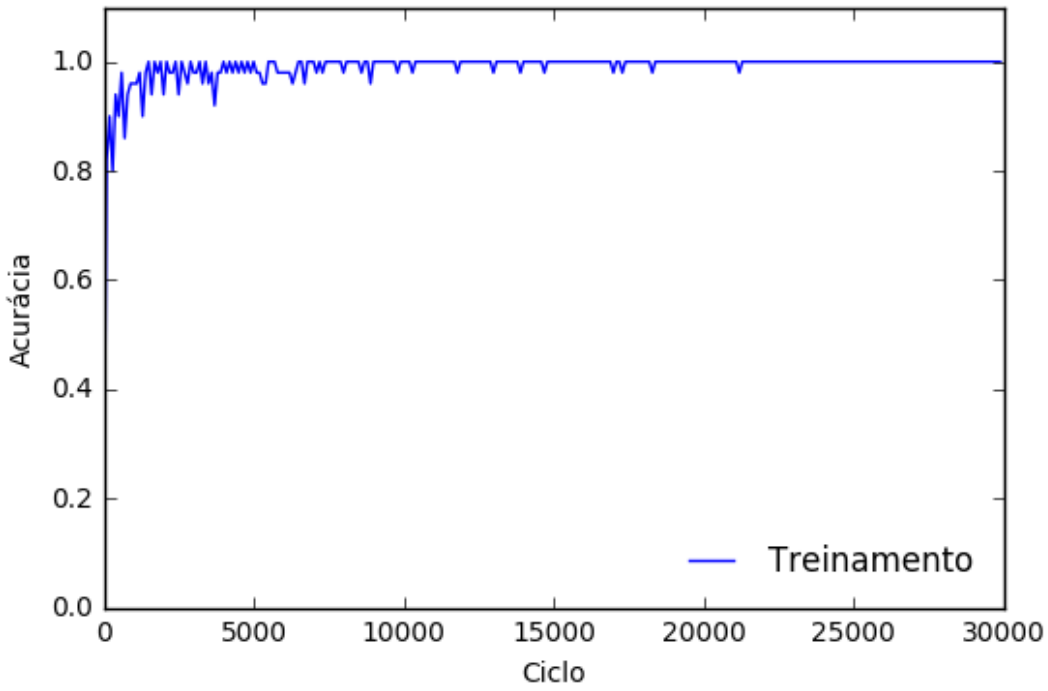
Gráfico 18 – Custo no treinamento 11



Fonte: Elaborado pelo autor (2016)

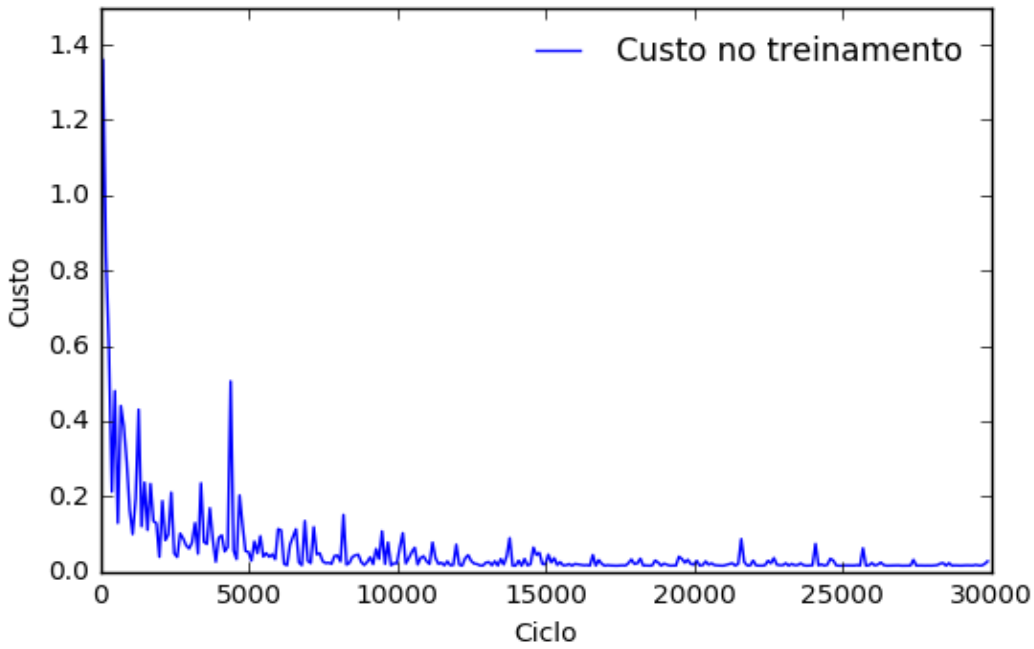
Mudando o valor do parâmetro de regularização de 0,1 para 0,001 pode-se perceber o a maior proximidade dos valores da função de custo na base em zero, exibido nos gráficos 19 e 20, gerando ao final do treinamento uma acurácia de 99,40% em 1 hora e 28 minutos de processamento.

Gráfico 19 – Acurácia no treinamento 12



Fonte: Elaborado pelo autor (2016)

Gráfico 20 – Custo no treinamento 12



Fonte: Elaborado pelo autor (2016)

Como o dataset utilizado não possui imagens com uma alta complexidade, por exemplo, se for comparado com o dataset ImageNet com milhões de imagens coloridas em alta resolução, a rede neural converge rapidamente para uma acurácia acima de 80%. Em poucos ciclos de treinamento a acurácia fica acima de 98%. Isso faz com que o impacto das

alterações implementadas tenha pouco efeito no resultado final, aumentando em décimos percentuais a acurácia da rede nas imagens de teste. Por esse motivo, foram gerados os gráficos para auxiliar na visualização das mudanças no comportamento da rede durante os treinamentos.

A acurácia máxima atingida ficou em 99,43%. Os parâmetros envolvidos nesta simulação são descritos na tabela 4, simulação 15. As tabelas 3 e 4 apresentam um resumo das simulações executadas com a relação dos parâmetros envolvidos para cada simulação. Estes parâmetros empregados envolvem:

- As funções de ativação sigmoid, tangente hiperbólica e RELU.
- Os algoritmos de treinamento gradiente descendente estocástico, Adam, RMSProp e Momentum.
- As funções de custo cross-entropy, cross-entropy + cllip e a softmax_cross_entropy.
- As técnicas de regularização dropout e L2.
- As demais alterações de parâmetros focaram na taxa de aprendizado, número de camadas totalmente conectadas, tamanho do mini-batch e na taxa de regularização beta.

Tabela 3 – Simulações com a CNN de 1 a 8

Simulação	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Acurácia	9,8%	94,56%	92,15%	10,03%	9,6%	99,10%	98,16%	99,05%
Tempo Treinamento	1h 27m	2h 42m	2h 44m	2h 41m	2h 46m	2h 46m	1h 26m	1h 28m
Algoritmo	SGD	SGD	SGD	SGD	Adam	Adam	Adam	Adam
Convoluções	2	2	2	2	2	2	2	2
Poolings	2	2	2	2	2	2	2	2
Camadas FC	2	2	2	2	2	2	2	2
Função Ativação	Sigmoid	RELU	Tanh	RELU	RELU	RELU	RELU	RELU
Função Custo	Cross Entropy	Cross Entropy	Cross Entropy	Cross Entropy	Cross Entropy	Cross Entropy + Clip	Cross Entropy + Clip	Cross Entropy + Clip
Taxa Aprendizado	0,0001	0,0001	0,0001	0,1	0,0001	0,0001	0,00001	0,0001
Ciclos	16000	30000	30000	30000	30000	30000	30000	30000
Stride Convolução	1	1	1	1	1	1	1	1
Stride Pooling	2	2	2	2	2	2	2	2
Kernel Convolução.	5x5	5x5	5x5	5x5	5x5	5x5	5x5	5x5
Kernel Pooling	2x2	2x2	2x2	2x2	2x2	2x2	2x2	2x2
Algorit. Pooling	Max	Max	Max	Max	Max	Max	Max	Max
Padding	SAME	SAME	SAME	SAME	SAME	SAME	SAME	SAME
Filtros Camada 1	32	32	32	32	32	32	32	32
Filtros Camada 2	64	64	64	64	64	64	64	64

Neurônios FC	1024	1024	1024	1024	1024	1024	1024	1024
Batch Validação	8000	8000	8000	8000	8000	8000	8000	8000
Mini Batch	100	100	100	100	100	100	50	50
Taxa Dropout	Não	Não	Não	Não	Não	Não	0,5	0,5
Regularizador L2	Não	Não	Não	Não	Não	Não	Não	Não
Beta Regulariz. L2	Não	Não	Não	Não	Não	Não	Não	Não

Fonte: Elaborado pelo autor (2016)

Continuação da tabela:

Tabela 4 – Simulações com a CNN de 9 a 16

Simulação	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Acurácia	99,27%	98,97%	99,35%	99,40%	99,03%	99,03%	99,43%	97,16%
Tempo Treinamento	1h 28m	1h 26m	1h 28m	1h 28m	2h 17m	4h 03m	1h 36m	1h 32m
Algoritmo	Adam	Adam	Adam	Adam	RMSProp	Adam	Adam	Momentum
Convoluções	2	2	2	2	2	2	2	2
Poolings	2	2	2	2	2	2	2	2
Camadas FC	2	2	2	2	3	2	3	3
Função Ativação	RELU	RELU	RELU	RELU	RELU	RELU	RELU	RELU
Função Custo	Cross Entropy + Clip	Softmax – Cross - Entropy	Cross Entropy + Clip	Cross Entropy + Clip	Cross Entropy + Clip	Cross Entropy + Clip	Cross Entropy + Clip	Cross Entropy + Clip
Taxa Aprendizado	0,0001	0,0001	0,0001	0,0001	0,0001	0,0001	0,0001	0,0001
Ciclos	30000	30000	30000	30000	30000	30000	30000	30000
Stride Convolução	1	1	1	1	1	1	1	1
Stride Pooling	2	2	2	2	2	2	2	2
Kernel Convolução.	5x5	5x5	5x5	5x5	5x5	5x5	5x5	5x5
Kernel Pooling	2x2	2x2	2x2	2x2	2x2	2x2	2x2	2x2
Algorit. Pooling	Max	Max	Max	Max	Max	Max	Max	Max
Padding	SAME	SAME	SAME	SAME	SAME	SAME	SAME	SAME
Filtros Camada 1	32	32	32	32	32	32	32	32
Filtros Camada 2	64	64	64	64	64	64	64	64
Neurônios FC	1024	1024	1024	1024	1024	1024	1024	1024
Batch Validação	8000	8000	8000	8000	8000	8000	8000	8000
Mini Batch	50	50	50	50	50	100	50	50
Taxa Dropout	0,5	0,5	0,5	0,5	0,5	Não	0,5	0,5
Regularizador L2	Não	Não	Sim	Sim	Sim	Não	Sim	Sim
Beta Regulariz. L2	Não	Não	0,1	0,001	0,004	Não	0,001	0,001

Fonte: Elaborado pelo autor (2016)

6 VALIDAÇÃO DO MATERIAL INSTRUCIONAL

A validação realizada tem como objetivo averiguar se o material instrucional aplicado contribuiu no aumento do interesse dos voluntários no estudo ou aplicação de DL. Também verificar se foi possível a compreensão dos conceitos-chave que envolvem as redes neurais e DL e caso não foram, poder identificá-los para ter a possibilidade de melhorar o conteúdo criado. Para atingir esta finalidade foi elaborado um questionário a partir do referencial teórico pesquisado, abordando conceitos baseados na comparação realizada entre os três materiais instrucionais sobre DL analisados na tabela 2. Para examinar as respostas obtidas no questionário foi utilizada a técnica de análise de conteúdo (FREITAS; JANISSEK, 2000, p.37 apud ELEUTER, 2011). Esta análise abrange a estipulação do universo a ser estudado (questionário), categorização das questões propostas e, por fim, quantificar e interpretar os dados coletados.

As questões foram distribuídas em 5 categorias diferentes (chamadas de categorias de perguntas), e são do tipo abertas (qualitativas) e fechadas (quantitativas).

Categorias das perguntas:

- Motivação;
- Redes Neurais;
- Conceito de Deep Learning;
- Redes Neurais Convolucionais;
- Underfitting, Overfitting e Regularização.

A criação das categorias visa agrupar as questões de modo que auxilie na análise dos conteúdos centrais abordados no referencial teórico. A categoria motivação busca identificar a familiaridade do voluntário com a área de inteligência artificial e as tecnologias de DL. Outra finalidade é verificar se o material instrucional contribuiu no interesse dos indivíduos na utilização de DL em algum projeto ou na procura de mais conhecimento sobre o tema. Lourenço e Paiva (2010) destacam a importância da motivação para as pessoas no processo de busca pelo conhecimento, relacionando fatores que influenciam no aumento do interesse do indivíduo para alcançar um objetivo. As demais categorias originam-se da tabela 2 e pretendem avaliar o conhecimento dos voluntários nestes conteúdos, após o término do estudo.

O material instrucional foi aplicado em uma turma da Universidade Feevale da cadeira de inteligência artificial no dia 19/10/2016. Estavam presentes 7 alunos da turma no qual cada

um utilizou o material instrucional em um computador da universidade. Neste dia, ocorreu um problema na importação do dataset MNIST e também problemas na execução das máquinas virtuais. Por exemplo, o sistema operacional apresentava muita lentidão utilizando 100% do disco rígido. Com estes problemas, os alunos apenas conseguiram fazer a leitura do material, não podendo interagir com o código implementado. Dentre os 7 alunos presentes uma dupla foi formada e o restante trabalhou individualmente. Ao final foram obtidas 5 respostas do questionário, uma a menos do que deveria ter sido.

O outro grupo (Outros) de voluntários constituiu-se de profissionais da área de desenvolvimento de software que trabalham em uma software house, bem como um ex-aluno do curso de Ciência da Computação da Universidade Feevale, totalizando 6 pessoas. Os voluntários neste grupo utilizaram o notebook nos seus computadores pessoais ou nos do seu trabalho. Este grupo foi mantido com um integrante a mais, em comparação com as repostas obtidas pelos alunos da Feevale, para que a avaliação geral possuísse mais informações para análise.

Nas tabelas 5 e 6 estão as compilações das respostas dos voluntários. Na tabela 5 ainda foi incluída uma divisão pelo grupo dos alunos e pelo grupo dos demais voluntários para comparar as respostas em ambientes diferentes (os alunos não conseguirem executar o algoritmo).

Tabela 5 – Respostas objetivas do questionário

Origem da questão Conforme item 1 da tabela 2 (pg 53)	**Questão**	**Sim**	**Não**
Categoria Motivação			
	Possui algum conhecimento sobre Inteligência Artificial?	8	3
	Grupo alunos	4	1
	Grupo outros	4	2
	Você já utilizou a linguagem Python?	5	6
	Grupo alunos	3	2
	Grupo outros	2	4
	Possui algum conhecimento sobre deep learning ou sobre a biblioteca TensorFlow?	1	10
	Grupo alunos	0	5
	Grupo outros	1	5
	O notebook utilizado possui uma linguagem objetiva que visa facilitar a sua compreensão?	8	3
	Grupo alunos	4	1
	Grupo outros	4	2

	Após a utilização do material sobre deep learning houve um aumento de interesse no estudo da tecnologia?	8	3
	Grupo alunos	3	2
	Grupo outros	5	1
Categoria redes neurais			
31	Foi possível a compreensão de como um neurônio artificial é modelado?	10	1
	Grupo alunos	4	1
	Grupo outros	6	0
10	Sobre a interligação dos neurônios, o papel dos pesos sinápticos foi compreendido?	9	2
	Grupo alunos	4	1
	Grupo outros	5	1
10	Ficou clara a maneira com que a saída de um neurônio influencia na ativação dos neurônios da camada seguinte?	11	0
	Grupo alunos	5	0
	Grupo outros	6	0
9	Com a leitura do material, o modo com que o algoritmo gradiente descendente busca os pesos sinápticos mais adequados ao modelo neural foi assimilado?	6	5
	Grupo alunos	2	3
	Grupo outros	4	2
Categoria conceito deep learning			
32	Sobre a descrição do conceito de deep learning, este foi apresentado de forma clara e objetiva?	11	0
	Grupo alunos	5	0
	Grupo outros	6	0
13, 14	O motivo que gerava a dificuldade no treinamento de redes neurais profundas, antes das técnicas de deep learning, pôde ser entendido?	9	2
	Grupo alunos	4	1
	Grupo outros	5	1
6	A solução que levou na possibilidade de realizar o treinamento de redes neurais profundas foi compreendido?	11	0
	Grupo alunos	5	0
	Grupo outros	6	0
20	A ideia de aprendizado em vários níveis pelas redes neurais profundas foi apresentado de forma intuitiva?	9	2
	Grupo alunos	5	0
	Grupo outros	4	2
20	O motivo pelo qual as camadas iniciais da rede neural possui uma representação menos abstrata em comparação com as camadas finais, pôde ser entendido?	9	2
	Grupo alunos	4	1
	Grupo outros	5	1
Categoria rede neural convolucional			
2	Através do modo com que foi abordada a descrição da arquitetura de uma rede convolucional foi possível sua compreensão?	11	0
	Grupo alunos	5	0
	Grupo outros	6	0
33, 34	Com relação às três ideias que permeiam uma camada convolucional, foi	8	3

	possível entender a relação entre os campos receptivos locais e o compartilhamento de pesos e biases?		
	Grupo alunos	3	2
	Grupo outros	5	1
35	Com base no que foi explicado sobre as camadas de pooling ou agrupamento, o objetivo de se utilizar tais camadas na arquitetura da rede convolucional pôde ser compreendido?	9	2
	Grupo alunos	5	0
	Grupo outros	4	2
36	Com relação ao emprego de uma camada totalmente conectada ao final da arquitetura da rede convolucional, o motivo da inclusão desta camada ficou claro?	11	0
	Grupo alunos	5	0
	Grupo outros	6	0
Categoria Underfitting, Overfitting e Regularização			
11	A técnica de regularização dropout altera a estrutura da rede neural visando a redução do overfitting durante o treinamento. Após a leitura, execução dos scripts e visualização dos gráficos de acurácia e custo este processo de regularização pôde ser entendido?	8	3
	Grupo alunos	3	2
	Grupo outros	5	1
26, 27	A consequência da ocorrência do overfitting e underfitting no desempenho da rede neural ficou claro?	11	0
	Grupo alunos	5	0
	Grupo outros	6	0
37	Houve dificuldade na compreensão da maneira com que a técnica regularização L2 altera os pesos para a redução do overfitting?	7	4
	Grupo alunos	3	2
	Grupo outros	4	2

Fonte: Elaborado pelo autor (2016)

Tabela com os comentários.

Tabela 6 – Respostas qualitativas do questionário

Item	Categoria	Perguntas e Respostas	Palavras Chave
1	**Motivação**	**No caso de ter se interessado na tecnologia de deep learning, o que mais gostaria de aprender sobre a mesma? Você pensou em algum projeto utilizando deep learning? Quais aplicações você imaginou para a tecnologia?**	Veículos autônomos, big data, processamento de imagens.
		Processamento de imagens médicas e semelhantes.	
		Assuntos de interesse seriam sobre veículos autônomos; Análise, tendências e sugestões de compra e venda de ações.	
		Utilização da tecnologia para identificação de padrões e formações de padrões para bolsa de valores.	
		Não pensei em nenhum projeto especifico, mas vou passar a cuidar mais sobre as tecnologias que utilizam deep learning no dia a dia.	
		Gostaria de aprender sobre deep learning em big data, para aprendizado de grandes dados. Pesquisei nos últimos dias sobre uma operação chamada serenata do amor, onde se busca dados de políticos, cruzam-se os dados e se verifica se ouve alguma	

		corrupção.	
		Tecnologia autônoma no campo com uso de drones e veículos auto dirigíveis.	
2	**Redes Neurais**	**Houve dificuldade na compreensão dos conceitos relacionados a redes neurais? Se sim, descreva quais.**	Linguagem técnica, complexidade, SGD.
		A linguagem utilizada é demasiadamente técnica.	
		O conceito em si é muito complexo na minha opinião.	
		Encontrei dificuldades na compreensão das ativações dos neurônios de saída; Compreensão do algoritmo gradiente.	
		Sim, para um leigo no assunto ou pessoa sem interesse no assunto, fica maçante a leitura, tornado difícil a compreensão.	
		Não, ficou claro e objetivo.	
		Algoritmo gradiente.	
3	**Deep Learning**	**Caso tenha ficado com dúvidas sobre o conceito de deep learning descreva quais.**	
		Sem comentários para esta categoria.	
4	**Redes neurais convolucionais**	**Caso tenha encontrado dificuldade na compreensão do funcionamento das redes convolucionais, descreva quais.**	
		Sem comentários para esta categoria.	
5	**Underfitting, Overfitting e Regularização**	**Sobre os conceitos de regularização, overfitting e underfitting, caso tenha surgido dificuldades na sua compreensão descreva-as.**	Regularização L2, complexidade.
		Como é efetuado a regularização do L2, diminuição dos pesos.	
		O conceito total desta tecnologia é complexo para entender rapidamente, mas de forma geral pode ser entendido.	

Fonte: Elaborado pelo autor (2016)

Com as respostas do questionário compiladas pôde ser efetuada uma análise quantitativa e, para as questões abertas, procurou-se verificação qualitativa.

Quando os voluntários foram questionados se possuíam algum conhecimento sobre DL ou sobre a biblioteca Tensorflow apenas 1 afirmou que "Sim", mesmo que 72,7% (8 pessoas) dos voluntários afirmarem ter algum conhecimento sobre IA. Um dos principais dados extraídos desta pesquisa foi o aumento do interesse dos voluntários no estudo sobre DL e suas possíveis aplicações, 72,7% (8 pessoas). Além disso, surgiram muitas ideias para aplicações de DL (item 1 da tabela 6). Como por exemplo, processamento de imagens médicas, veículos autônomos e análise de cotações de ações na bolsa de valores.

Ao questionar se o notebook sobre DL foi apresentado de maneira que facilitasse a compreensão dos conceitos, o mesmo percentual de participantes afirmou que "Sim" com 72,7%. Este dado pode ter uma relação com o conhecimento prévio do voluntário com a área

de IA, tornando mais fácil o entendimento dos demais conceitos. Os três percentuais com o mesmo valor são relacionados abaixo:

- Com conhecimento sobre IA = 72,7% sim;
- Com aumento de interesse sobre deep learning = 72,7% sim;
- Notebook possui linguagem objetiva visando facilitar a sua compreensão = 72,7% sim.

Na categoria **redes neurais** os conceitos sobre como um neurônio é modelado, o papel dos pesos sinápticos e a influência da saída de um neurônio para a ativação dos neurônios das camadas posteriores foi assimilado por mais de 80% dos participantes. A questão envolvendo o algoritmo gradiente descendente foi a que mais gerou dificuldades nesta categoria, com 45,5% das pessoas indicando não ter conseguido compreender o algoritmo. Essa dificuldade já era prevista com base nos estudos prévios, onde também surgiram muitas dúvidas até poder entendê-lo.

O item 2 da tabela 6 possui os comentários feitos pelos voluntários na categoria de redes neurais, com críticas sobre a linguagem puramente técnica utilizada e indicando que o conceito sobre redes neurais é muito complexo. Dois participantes enfatizaram a dificuldade para compreender o algoritmo gradiente descendente.

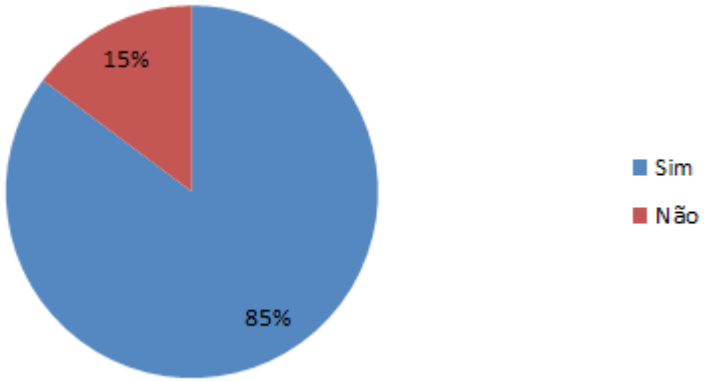
As categorias **conceito deep learning** e **rede neural convolucional** apresentaram os melhores resultados analisando a opinião dos voluntários sobre a facilidade no entendimento destes conceitos. Na maioria das questões, mais de 80% dos voluntários indicaram que foi possível a compreensão dos conceitos questionados. Em destaque a opinião de 100% dos participantes sobre a apresentação clara e objetiva do conceito de DL, tal como a solução que levou a possibilidade de treinar DNNs. No caso da rede neural convolucional, o entendimento sobre a arquitetura de rede e a utilização de uma camada totalmente conectada ao final da arquitetura apresentaram a taxa de 100% de afirmação. Apenas a relação entre as três ideias que permeiam a camada convolucional apresentou uma taxa inferior, 72,7%.

Na última categoria avaliada, **underfitting, overfitting e regularização**, os conceitos também obtiveram bons resultados. Sobre o dropout, 72,7% opinaram que foi viável a assimilação da maneira com que esta técnica altera a estrutura da rede para a redução do overfitting. Com relação a consequência da ocorrência do underfitting e overfitting no desempenho da rede neural, 100% das pessoas afirmou tê-lo absorvido. O conceito que gerou mais dúvidas foi a regularização L2 com 63,6%. Os comentários gerados nesta categoria destacam a dificuldade na compreensão deste conceito, eles foram descritos por dois voluntários da seguinte forma: “Como é efetuado a regularização do L2, diminuição dos

pesos” e “o conceito total desta tecnologia é complexo para entender rapidamente, mas de forma geral pode ser entendido”.

O gráfico 21 apresenta a divisão do total das respostas, desconsiderando a categoria motivação, 15% das mesmas foram negativas e 85% positivas num total de 176. Estes números indicam em suma, a contribuição satisfatória do notebook para a compreensão dos conceitos-chave de DL pelos participantes.

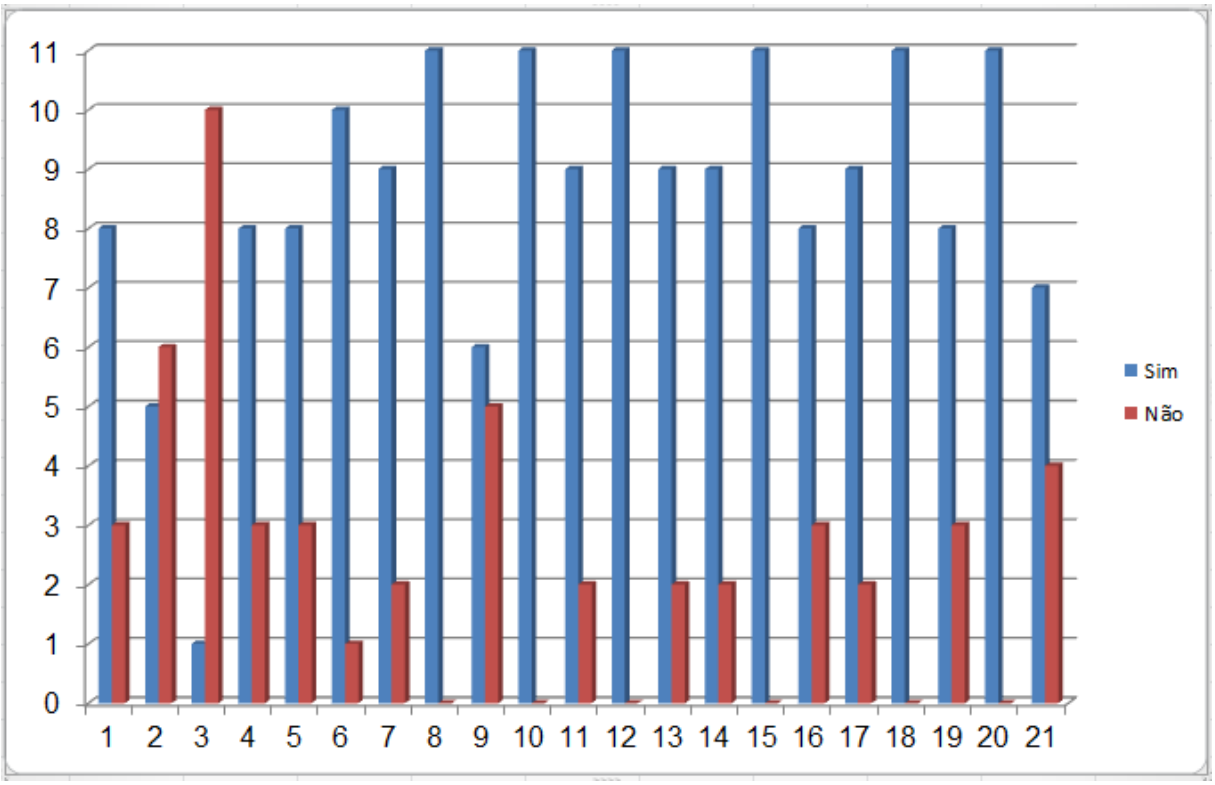
Gráfico 21 – Resumo das respostas objetivas



Fonte: Elaborado pelo autor (2016)

Outra observação que pode ser feita através da tabela 5 refere-se à diferenciação nas respostas dos dois grupos de participantes, com a maioria das questões contendo 1 participante com opinião diferente. As respostas adquiridas pelos dois grupos foram semelhantes, pois tiveram dificuldades no entendimento dos mesmos conceitos. Nos conteúdos abordados sobre o algoritmo gradiente descendente e regularização L2 foram onde apareceram as maiores dúvidas. No entanto, para o gradiente descendente a taxa de afirmação no grupo “Outros” foi duas vezes maior. Isto pode ser devido ao maior tempo disponibilizado para estudo ou aos auxílios prestados nas dúvidas. Quanto à regularização L2 os resultados foram mais próximos entre os dois grupos.

Gráfico 22 – Total das respostas divididas por cada questão objetiva



Fonte: Elaborado pelo autor (2016)

O gráfico 22 apresenta o resultado das respostas divididas por cada questão aplicada na validação. A coluna da esquerda indica o número total de voluntários (11) e abaixo é assinalado o número de sequência de cada questão, conforme a ordem apresentada na tabela 5.

7 CONSIDERAÇÕES FINAIS

A tecnologia de deep learning permitiu que sistemas de inteligência artificial realizassem tarefas que anteriormente esperava-se que levaria mais tempo. Com diferentes arquiteturas disponíveis, esta tecnologia é empregada em inúmeros segmentos, com seus métodos sendo evoluídos constantemente por pesquisadores de todo o mundo. Entretanto existem poucas pessoas especializadas nesta área no mercado, tão pouco que conhecem a tecnologia.

A partir dos trabalhos relacionados, pode-se constatar a tentativa de empresas como o Google de buscar mais profissionais interessados em trabalhar com deep learning, através do curso grátis na plataforma Udacity disponibilizado gratuitamente. Todavia, estes materiais geralmente focam na formalização matemática das funções que envolvem deep learning ou na teoria dos conceitos. Como esta tecnologia possui muitas características e detalhes para serem pesquisados, quem pouco conhece sobre inteligência artificial pode ter dificuldade no seu entendimento, mesmo que sua intensão seja uma aplicação "simples" com DL. Neste sentido, o presente trabalho teve por objetivo realizar um estudo teórico e prático sobre DL, com o propósito de criar um material instrucional sobre esta tecnologia que possa ser utilizado posteriormente por outros pesquisadores ou estudantes. O material instrucional desenvolvido, acompanhado com esta pesquisa, pode ser utilizado como um ponto de partida para entender os conceitos-chave sobre DL e CNNs. Além disto, apresentar uma implementação prática de deep learning com a biblioteca de machine learning Tensorflow criada pelo Google, assim como, fornecer as principais referências disponíveis nesta área de estudo. A seção prática visa demonstrar como realizar o treinamento de uma CNN, além do mais, alguns problemas que podem ocorrer durante o treinamento e como contorná-los.

Para atingir o objetivo citado primeiramente foi elaborada a problemática da pesquisa: **Com relação ao conceito de Deep Learning, como implementá-lo para o desenvolvimento de uma ferramenta voltada ao ensino-aprendizado desta tecnologia?** Relativo a questão introduzida acima, após o estudo sobre a temática proposta e as tecnologias disponíveis elaborou-se o objetivo geral deste trabalho: **investigar como o conceito de aprendizado de máquina, deep learning, pode ser empregado como ferramenta para auxílio no ensino-aprendizado da tecnologia.**

O objetivo geral então foi dividido em tarefas menores ou objetivos específicos criados de forma sequencial. Desta forma, foi realizada uma **pesquisa bibliográfica** que partiu dos conceitos básicos de redes neurais, às arquiteturas de DNNs e objetos de estudo

relacionados ao propósito desta pesquisa, assim atingindo os dois objetivos específicos iniciais: **1) realizar um estudo teórico sobre os principais métodos de aprendizado de máquina com base no conceito de deep learning;** e, **2) pesquisar sobre as atuais e possíveis aplicações de deep learning.**

Com a fundamentação teórica buscou-se definir as tecnologias que seriam empregadas na criação de uma DNN, de maneira que possa ser facilmente utilizada por outras pessoas como material de estudo sobre deep learning. Neste caso, foi escolhida a biblioteca **Tensorflow** juntamente com o ambiente de desenvolvimento **Jupyter**. Através do uso destas tecnologias foi possível o desenvolvimento de um material instrucional interativo sobre DL, que explica os conceitos e ao mesmo tempo codifica e treina o modelo de uma CNN. Com isto obteve-se o terceiro e quarto objetivo específico: **3) pesquisar as ferramentas disponíveis para a implementação de deep learning;** e, **4) apresentar uma implementação do conceito de deep learning para auxiliar voltada no ensino-aprendizado da tecnologia.**

Através da CNN desenvolvida foram realizados treinamentos para demonstração da influência na alteração dos parâmetros da rede permitindo analisar o impacto da alteração das: funções de ativação de neurônios, taxa de aprendizado, algoritmo de treinamento, camadas adicionais de rede e métodos de regularização. Outro experimento efetuado foi a aplicação do material desenvolvido entre 12 voluntários, onde por meio de um questionário pode-se averiguar o conhecimento dos voluntários sobre a tecnologia de deep learning e o impacto do estudo sobre seus interesses. Por fim, foi completado o último objetivo específico: **5) realizar simulações com a implementação do conceito de deep learning conforme o modelo proposto.**

Portanto, o objetivo geral deste trabalho foi concluído com a pesquisa realizada e a criação do material instrucional que pode ser acessado e copiado no Github (https://github.com/). A partir destes dois conteúdos gerou-se um ponto de partida para auxiliar pessoas interessadas no estudo sobre DL. Não obstante, as pesquisas com DL serão continuadas a fim de tentar aplicá-la em dados de séries temporais. No próximo capítulo serão destacadas as dificuldades encontradas no decorrer da pesquisa e também os trabalhos futuros propostos.

7.1 DIFICULDADES E TRABALHOS FUTUROS

No processo de pesquisa bibliográfica deparou-se com a falta de bons livros específicos sobre deep learning por ser uma área recente da inteligência artificial. Muitos artigos são encontrados; porém, não são suficientes para entender os principais conceitos. O único livro escrito por alguns dos principais pesquisadores de deep learning ainda está em preparação, mas pode ser acessada no site <http://www.deeplearningbook.org/>. Outros três livros foram estudados dos autores Nielsen (2015), Deng e Yu (2014 e 2015), e Heaton (2015).

Quanto à implementação e treinamento da CNN, a principal limitação está na memória disponível para validação. Nos testes foi utilizado um notebook DELL Inspiron 14r 5437 com 8GB de RAM mais 2GB adicionais com SWAP e processador Intel i7-4500U (1.8GHz), contudo, não foi possível a utilização das 10000 imagens disponíveis para teste no dataset MNIST. Mesmo o dataset sendo pequeno, menos de 10MB no total de teste, ao carregar todas as imagens para o Tensorflow ocorria a ocupação total da memória disponível e eventualmente o travamento da máquina.

O mesmo problema de falta de memória ocorreu ao tentar realizar o treinamento da CNN diretamente na GPU do notebook. A GPU de modelo Geforce GT 740M possui suporte a tecnologia CUDA exigida pelo Tensorflow e consta com 2GB de memória. Mas, também, não foi suficiente para o treinamento. Talvez tenha alguma configuração diferente a ser feita para o funcionamento correto.

Como trabalhos futuros pode-se incluir melhorias no notebook desenvolvido tal qual segue:

- Extrair o cálculo dos gradientes;
- Tentar melhorar a explicação do gradiente descendente e da regularização L2, dois conceitos que geraram as maiores dúvidas na aplicação do notebook;
- Incluir outros modos de visualização do resultado do treinamento e validação da CNN;
- Realizar estudos para a implementação de uma CNN em datasets maiores como o CIFAR; e
- Realizar estudos para a implementação de uma rede neural recorrente em dados que não sejam de imagens.

REFERÊNCIAS BIBLIOGRÁFICAS

BENGIO, Yoshua. **Learning deep architectures for AI**, 2009. Disponível em: <http://sanghv.com/download/soft/machine%20learning,%20artificial%20intelligence,%20mathematics%20ebooks/ML/learning%20deep%20architectures%20for%20AI%20(2009).pdf>. Acessado em 05/03/2016.

BENGIO, Yoshua, LAROCHELLE, Hugo, LOURADOUR, Jérôme, LAMBLIN, Pascal. **Exploring strategies for training deep neural networks**, 2009. Disponível em: <http://deeplearning.cs.cmu.edu/pdfs/1111/jmlr10_larochelle.pdf>. Acessado em 19/05/2016.

BEZ, M. R. **Uso de tecnologia para apoiar a implantação de métodos ativos nos currículos de medicina.** Proposta de Tese. Centro Interdisciplinar de Novas Tecnologias na Educação, Universidade Federal do Rio Grande do Sul, Porto Alegre. 2011.

DAVID, Shai Bem, SHWARTZ, Shai Shalev. **Understanding machine learning from theory to algorithms.** Cambridge Unisersity Press, 2014.

DENG, Li, YU, Dong**. Automatic speech recognition. A deep learning approach.** Springer. 2015.

DENG, Li, YU, Dong. **Deep learning methods and applications**. Foundations and Trends in Signal Processing, 2014.

Deep learning tutorial. LISA lab, Universidade de Montreal, 2015. Disponível em: <http://deeplearning.net/tutorial/deeplearning.pdf>. Acessado em 05/03/2016.

DOCKER**.** Docker inc., 2016. Disponível em: <https://www.docker.com/>. Acessado em 30/09/2016.

GOODFELLOW, Ian BENGIO, Yoshua, COURVILLE, Aaron. **Deep learning book**, 2016. Disponível em: <http://www.deeplearningbook.org>. Acessado em 10/03/2016.

HAYKIN, Simon. **Redes neurais**: princípios e prática. Porto Alegre: Bookman, 2001.

HAYKIN, Simon. **Neural networks and learning machines**. Pearson, 2008.

HE, Kaiming, ZHANG, Xiangyu, REN Shaoqing, SUN, Jian. **Deep residual learning for image recognition**, 2015. Disponível em: <http://arxiv.org/pdf/1512.03385v1.pdf>. Acessado em 05/03/2016.

HEATON, Jeff. **Artificial intelligence for humans**, Volume 3: Deep Learning and Neural Networks. Heaton Research, Inc., 2015.

HERMANN, Karl Moritz, KOCISKY, Tomas, GREFENSTETTE, Edward, ESPEHOLT, Lasse, KAY, Will, SULEYMAN, Mustafa, BLUNSOM, Phil. **Teaching machines to read and comprehend**, 2015. Disponível em: <http://papers.nips.cc/paper/5945-teaching-machines-to-read-and-comprehend>. Acessado em 05/03/2016.

HINTON, Geoffrey. **To recognize shapes, first learn to generate images**, 2006. Department of Computer Science, University of Toronto. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.443.8824&rep=rep1&type=pdf>. Acessado em 05/03/2016.

ÍRSOY, Ozan, CARDIE, Claire. **Deep recursive neural networks for compositionality in language,** 2014**.** Disponível em: <http://papers.nips.cc/paper/5551-deep-recursive-neural-networks-for-compositionality-in-language.pdf>. Acessado em 07/06/2016.

JUPYTER. Jupyter project, 2016. Disponível em: <http://jupyter.org/>. Acessado em 30/09/2016.

KINGMA, Diederik P, BA, Jimmy Lei. **Adam: a method for stochastic optimization**, 2015. Disponível em: <https://arxiv.org/pdf/1412.6980.pdf>. Acessado em 30/06/2016.

LECUN, Yann, BENGIO, Yoshua, HINTON, Geoffrey. **Deep learning. Nature**, Volume 521, maio de 2015. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.436.894&rep=rep1&type=pdf>. Acessado em 05/03/2016.

LOURENÇO, Abílio Afonso, PAIVA, Maria Olímpia Almeida. **A motivação escolar e o processo de aprendizagem**, 2010. Disponível em: <http://www.cienciasecognicao.org/pdf/v15_2/12_132-141_m313.pdf>. Acessado em 05/10/2016.

MARTÍN Abadi, ASHISH Agarwal, PAUL Barham, EUGENE Brevdo, ZHIFENG Chen, CRAIG Citro, GREG S. Corrado, ANDY Davis, JEFFREY Dean, MATTHIEU Devin, SANJAY Ghemawat, IAN Goodfellow, ANDREW Harp, GEOFFREY Irving, MICHAEL Isard, RAFAL Jozefowicz, YANGQING Jia, LUKASZ Kaiser, MANJUNATH Kudlur, JOSH Levenberg, DAN Mané, MIKE Schuster, RAJAT Monga, SHERRY Moore, DEREK Murray, CHRIS Olah, JONATHON Shlens, BENOIT Steiner, ILYA Sutskever, KUNAL Talwar, PAUL Tucker, VINCENT Vanhoucke, VIJAY Vasudevan, FERNANDA Viégas, ORIOL Vinyals, PETE Warden, MARTIN Wattenberg, MARTIN Wicke, YUAN Yu, XIAOQIANG Zheng. **TensorFlow: Large-scale machine learning on heterogeneous systems**, 2015. Disponível em: <http://download.tensorflow.org/paper/whitepaper2015.pdf>. Acessado em 20/05/2016.

NIELSEN, Michael A. **Neural networks and deep learning**, Determination Press, 2015. Disponível em: <http://neuralnetworksanddeeplearning.com/index.html>. Acessado em 15/01/2016.

OORD, Aaron van den, KALCHBRENNER, Nal, KAVUKCUOGLU, Koray. **Pixel recurrent neural networks**, 2016. Disponível em: <http://arxiv.org/pdf/1601.06759>. Acessado em 05/03/2016.

PRODANOV, Cleber Cristiano, FREITAS, Ernani Cesar de. **Metodologia do trabalho científico:** métodos e técnicas da pesquisa e do trabalho acadêmico, 2013. Disponível em: <http://www.feevale.br/cultura/editora-feevale/metodologia-do-trabalho-cientifico---2-edicao>. Acessado em 05/03/2016.

PYTHON. Python Software Foundation, 2016. Disponível em <https://www.python.org/ >. Acessado em 30/08/2016.

RUSSAKOVSKY, Olga, DENG, Jia, SU Hao, KRAUSE Jonathan, SATHEESH, Sanjeev, MA Sean, HUANG, Zhiheng, KARPATHY, Andrej, KHOSLA, Aditya, BERNSTEIN, BERG, Michael Alexander C., FEI-FEI Li. **ImageNet large scale visual recognition challenge**, 2015. Disponível em: <http://arxiv.org/pdf/1409.0575.pdf>. Acessado em 05/03/2016.

RUSSELL Stuart, NORVIG, Peter. **Inteligência artificial**. Rio de Janeiro: Elsevier, 2013.

SALAKHUTDINOV, Ruslan. **Learning deep generative models,** 2015. Disponível em: <http://www.cs.cmu.edu/~rsalakhu/papers/annrev.pdf>. Acessado em 15/03/2016.

SILVER, David, SCHRITTWIESER, Julian, KALCHBRENNER, Nal, NHAM, John, GRAEPEL, Thore, HUANG, Aja, ANTONOGLOU, Ioannis, HASSABIS, Demis, MADDISON, Chris J., SUTSKEVER, Ilya, GUEZ, Arthur, PANNEERSHELVAM, Veda, LILLICRAP, Timothy, SIFRE, Laurent, LANCTOT, Marc, LEACH, Madeleine, DRIESSCHE, George van den, DIELEMAN, Sander, KAVUKCUOGLU, Koray, GREWE, Dominik. **Mastering the game of Go with deep neural networks and tree search**, 2016. Disponível em: <http://willamette.edu/~levenick/cs448/goNature.pdf>. Acessado em 05/03/2016.

SOCHER, Richard, PAULUS, Romain, MANNING, Christopher D. **Glogal belief recursive neural networks,** 2014. Disponível em: <http://papers.nips.cc/paper/5275-global-belief-recursive-neural-networks.pdf>. Acessado em 07/06/2016.

TENSORFLOW, 2016. Disponível em: <https://www.tensorflow.org/>. Acessado em 30/06/2016.

VENUGOPALAN, Subhashini, XU, Huijuan, DONAHUE, Jeff, ROHRBACH, Marcus, MOONEY, Raymond, SAENKO, Kate. **Translating videos to natural language using deep recurrent neural networks.** 2015. Disponível em: <https://arxiv.org/pdf/1412.4729v3.pdf>. Acessado em 10/04/2016.