UNIVERSIDADE DE ÉVORA

ESCOLA DE CIÊNCIAS E TECNOLOGIA

DEPARTAMENTO DE INFORMÁTICA

Aprendizagem Profunda: Estudo e Aplicações

João Miguel Neves Gusmão Pires

Orientação: Teresa Cristina de Freitas Gonçalves

Mestrado em Engenharia Informática

Dissertação

Évora, 2017

UNIVERSIDADE DE ÉVORA

ESCOLA DE CIÊNCIAS E TECNOLOGIA

DEPARTAMENTO DE INFORMÁTICA

Aprendizagem Profunda: Estudo e Aplicações

João Miguel Neves Gusmão Pires

Orientação: Teresa Cristina de Freitas Gonçalves

Mestrado em Engenharia Informática

Dissertação

Évora, 2017

Aos que potenciam o saber.

Agradecimentos

"Ninguém escapa ao sonho de voar, de ultrapassar os limites do espaço onde nasceu, de ver novos lugares e novas gentes. Mas saber ver em cada coisa, em cada pessoa, aquele algo que a define como especial, um objecto singular, um amigo - é fundamental. Navegar é preciso, reconhecer o valor das coisas e das pessoas, é mais preciso ainda!"

Antoine de Saint-Exupéry

A realização desta dissertação de mestrado contou com importantes apoios sem os quais não se teria tornado uma realidade e aos quais estarei eternamente grato.

À minha orientadora Teresa Gonçalves, pela dedicação, paciência e aconselhamento.

Ao meu colega Marco Nogueira pela troca de ideias durante a componente curricular do mestrado.

Aos meus ex-colegas e amigos do LIP, em particular ao Mário David pela dedicação e profissionalismo demonstrado no acesso à infraestrutura disponibilizada pelo LIP.

Aos meus professores Vasco Pedro e Pedro Salgueiro pela disponibilização dos recursos computacionais da Universidade de Évora.

A todos os outros intervenientes que de forma indireta e provavelmente sem se aperceberem me apoiaram durante este percurso.

Por último, tendo consciência que sozinho este trabalho teria sido bem mais difícil, dirijo um agradecimento especial à minha esposa e filhas, pelo seu apoio incondicional, incentivo, amizade e paciência demonstrados e total ajuda na superação dos obstáculos que ao longo desta caminhada foram surgindo. A elas dedico este trabalho!

"O conhecimento torna a alma jovem e diminui a amargura da velhice. Colhe, pois, a sabedoria. Armazena suavidade para o amanhã."

Leonardo da Vinci

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

AA Aprendizagem Automática, em inglês *Machine Learning* de acronimo ***ML***

ALD Alocação Latente de Dirichlet, em inglês *Latent Dirichlet Allocation* de acronimo ***LDA***

AP Aprendizagem Profunda, em inglês *Deep Learning* de acronimo ***DL***

ARP Assunção de Rede Profunda, em inglês *Deep Belief Network* de acronimo ***DBN***

CA Codificador Automático, em inglês *Auto Encoder* de acronimo ***AE***

DeepLDA LDA Profundo, em inglês *Deep LDA* de acronimo ***DeepLDA***

ECA Empilhamento de Codificadores Automáticos, em inglês *Stacked Auto Encoder* de acronimo ***SAE***

IA Inteligência Artificial, em inglês *Artificial Intelligence* de acronimo ***AI***

MBR Máquinas de Boltzman Restritas, em inglês *Restricted Boltzmann Machine* de acronimo ***RBM***

RAG Redes Adversas Geradas, em inglês *Generative Advesarial Networks* de acronimo ***GAN***

RN Rede Neuronal, em inglês *Neural Network* de acronimo ***NN***

RNA Rede Neuronal Artificial, em inglês *Artificial Neural Network* de acronimo ***ANN***

RNC Rede Neuronal Convoluída, em inglês *Convolutional Neural Network* de acronimo ***CNN***

SALB Subprogramas de Algebra Linear Básica, em inglês *Basic Linear Algebra Subprograms* de acronimo ***BLAS***

UE Universidade de Évora

Sumário

Esta tese aborda o tema da Aprendizagem Profunda, estudando-o através da comparação de várias frameworks e utilizando um conjunto de dados composto por imagens de algarismos manuscritos.

As *frameworks* utilizadas para o estudo foram algumas das mais conhecidas, como o Caffe, Theano e TensorFlow, entre outras; realizou-se também um estudo mais aprofundado da Theano com o Keras, TensorFlow com o Keras e Microsoft CNTK.

Foi analisado o desempenho de algoritmos pertencentes a três paradigmas da aprendizagem automática (supervisionada, semi-supervisionada e não supervisionada) através do conjunto de imagens de algarismos manuscritos.

À data em que foi realizado este trabalho de investigação, constata-se que os métodos de aprendizagem profunda são significativamente melhores do que os existentes na aprendizagem automática, quando os dados são supervisionados ou semi-supervisionados. No que diz ao trabalho com dados não supervisionados, conclui-se que o desenvolvimento ainda está numa fase embrionária. Com este tipo de dados, criam-se modelos que representem uma boa aproximação da realidade.

Palavras chave: aprendizagem profunda, aprendizagem automática, inteligência artificial, redes neuronais

Abstract

Deep Learning

Study and Aplications

This thesis is a study about Deep Learning, comparing various frameworks and using images and manuscript numeric characters datasets.

The used frameworks for this study where the most well known, like Caffe, Theano and TensorFlow, among outhers; within this study another one where made focused in Theano with Keras, TensorFlow with Keras and Microsoft CNTK.

Taking in account three types of Machine Learning (supervised, semi-supervised and unsupervised) the performance of some algorithms where analyzed using images and manuscripts numbers datasets.

At the date when this research study was made, it is verified that the deep learning methods were significatively better than the one's used in machine learning, when the data are supervised or semi-supervised. When the data is unsuperived, concludes that the development is in the begining. Trying to get a better aproach to build models that could be used with this kind of data.

Keywords: deep learning, machine learning, artificial intelligence, neural networks

1

Introdução

A Inteligência Artificial[1] é uma das áreas da informática que nos últimos anos tem mostrado as suas potenciais aplicações nos mais diversos domínios.

A Aprendizagem Automática[2] é uma área de investigação que utiliza conceitos de inteligência artificial e estatística. É uma disciplina extensa usando diversos métodos de aprendizagem, como por exemplo as redes neuronais, e, tendo como pontos de aplicação a robótica, entre outras. A AA a um nível mais complexo, utiliza métodos baseados em redes neuronais com diversas camadas. Neste caso, designa-se por Aprendizagem Profunda[3] é uma sub-área da AA. É sobre a AP que esta tese se debruça.

A AP é uma Rede Neuronal[4] composta por mais de duas camadas ocultas especializadas que utiliza métodos analíticos que permitem obter resultados positivos. Este tipo de redes complexas e elaboradas estão em fase de crescimento e permitem criar sistemas de previsão muito evoluídos.

[1] de acronimo IA, em inglês Artificial Intelligence.
[2] de acronimo AA, em inglês Machine Learning.
[3] de acronimo AP, em inglês Deep Learning
[4] de acronimo RN, em inglês Neural Network.

As RN tradicionais tinham dificuldade em lidar com quantidades de dados muito grandes ou com sistemas muito complexos. Com o aparecimento da AP conseguiram-se obter resultados bastante positivos no processamento da linguagem natural, na robótica através do reconhecimento visual e, dado que obtém melhores resultados com grandes quantidades de dados, permitiu integrar o Big Data[5].

1.1 Motivação

A AP é um tema muito atual e que perspetiva um grande crescimento, por isso o seu estudo, apesar de complexo torna-se interessante. Esta sub-área da AA permite a criação de modelos artificiais do cérebro humano com um rigor e uma complexidade elevados.

Assim, cria-se a possibilidade de comparar *frameworks*[6], e métodos e estudar alguns dos diversos modelos de aprendizagem artificial atuais.

Como o cérebro humano aprende com recurso a informação não supervisionada (i.e., não classificada), ou com uma pequena supervisão, seria interessante estudar os modelos que permitem a construção de sistemas de AP, tendo como entradas dados não supervisionados.

Conhecer, entender e aplicar as mais recentes descobertas neste capítulo é de grande utilidade para que novos horizontes do conhecimento sejam alargados.

1.2 Objetivos

Com este trabalho pretende-se tocar nos pontos chave da aplicação da AP num contexto de conjunto de dados de imagens. A comparação entre as *frameworks* é fundamental para percecionar qual a melhor abordagem a seguir num tipo de problema específico como, por exemplo, estudar as redes de AP com dados não supervisionados.

O objetivo principal será distinguir de entre as *frameworks* estudadas qual a melhor escolha para abordar conjuntos de dados compostos por imagens. O objetivo secundário será aplicar esta escolha a diferentes modelos de aprendizagem e em simultâneo comparar os resultados com outros modelos.

1.3 Abordagem

Na base desta dissertação está o estudo das *frameworks* que, numa fase inicial, vai permitir conhecer algumas delas e posteriormente escolher algumas (três) para comparação.

Na fase seguinte, a *framework* que apresentar melhores resultados vai ser usada para fazer a implementação de modelos adequados à construção de sistemas de AP e que têm como entradas conjuntos de dados não supervisionados.

Escolheram-se dois conjuntos de dados para este trabalho: o MNIST[7][LC] e o STL-10[8][AC11], ambos compostos por imagens e seus respetivos rótulos, no caso dos dados serem classificados total ou parcialmente. O primeiro foi usado por ser um conjunto clássico e que serve sempre de referência aos estudos realizados

[5] expressão inglesa que se atribui a conjuntos de dados muito grandes ou muito complexos.
[6] expressão inglesa que se refere às ferramentas disponíveis para uma determinada área de atividade.
[7] base de dados de algarismos escritos à mão disponível online.
[8] base de dados de imagens utilizadas para reconhecimento com uma resolução de 96x96.

neste domínio; o segundo foi usado, por ser um melhoramento do CIFAR-10[9].

1.4 Contribuições

Pretende-se que este trabalho sirva de referência para a investigação e desenvolvimento nesta área, constituindo um ponto de partida para estudos mais aprofundados e diversificados.

A comparação entre *frameworks* e métodos é fundamental para decidir qual o melhor caminho a seguir. A configuração experimental, permite perceber qual o melhor cenário.

As próximas linhas sistematizam a forma como se procedeu na preparação e organização do trabalho realizado.

A informação utilizada na tese foi recolhida de todas as fontes disponíveis: artigos científicos, cursos, livros e fontes disponíveis na web (forum, github, dockerhub, etc). A organização foi dividida em duas áreas distintas:

- informação útil para elaborar estudo comparativo entre algumas frameworks utilizadas em AP;
- código útil para a implementação de modelos de AP e que teria duas valências: ser usado para a comparação das *frameworks* e para implementação de modelos AP.

Embora o acesso à informação por vezes esteja bastante facilitado, a grande dificuldade é selecioná-la e adaptá-la aos objetivos estabelecidos. Neste caso, é importante validar a informação, por forma a que seja considerada fidedigna e, no caso do código disponibilizado online, testá-lo e executá-lo. Por vezes estas tarefas não são nada triviais se tivermos em conta que um programador coloca código operacional no *github* mas, por exemplo seis meses depois, o código para ser reutilizado vai ter de ser trabalhado, adaptado, pois as bibliotecas entretanto sofreram alterações que por vezes dificultam a sua reprodução. A reprodutibilidade por vezes pode falhar.

Todo este trabalho de integração de bibliotecas, de reutilização de código, de criação da infraestrutura em termos de software e de hardware foi realizado ao longo desta dissertação.

Pode dizer-se que para além das dificuldades encontradas e superadas, é um trabalho moroso e que requer bastante paciência e capacidade de encontrar soluções para todos os problemas encontrados durante este percurso.

Como contribuições adicionais temos:

- Estudo comparativo das diversas *framewoks* disponíveis, mas que se distingue de outros estudos por contemplar mais variáveis, maior número de *frameworks* e ter sido realizado na perspetiva de ajudar o leitor a fazer a escolha mais acertada. Repare-se que foram incluídos no estudo todo o tipo de publicações e de problemas, extraídos dados de fontes conceituadas e sendo realizada a análise estatística adequada;
- Adicionalmente ao ponto anterior, foi realizado um processo de seleção, seguido da implementação de modelos de AP, também eles escrupulosamente selecionados de entre artigos científicos, e, tendo em vista os resultados efetivos;

[9] outra base de dados composta por imagens usadas para classificação e com uma resolução inferior à STL-10.

- Outra contribuição deve-se ao facto deste estudo não se ficar pela comparação, mas servir de referência de apoio à decisão, até porque nele estão contemplados três dos paradigmas da aprendizagem, com consequente aplicação;
- No que diz respeito às aplicações, todo um processo de integração de sistemas e de escalonamento dos recursos computacionais foi realizado.

Com o trabalho realizado e apresentado, nesta tese, poderá um leigo nestas matérias compreender e decidir qual a melhor abordagem a seguir.

1.5 Organização da Dissertação

Esta tese encontra-se dividida em duas partes: a primeira que corresponde à comparação entre *frameworks*, e a segunda à aplicação de métodos de AP, que se encontram na vanguarda do desenvolvimento e investigação nesta área.

A dissertação está organizada da seguinte forma: o capítulo 1 apresenta uma introdução ao tema; o capítulo 2 introduz a AP e o contexto ou estado da arte em que esta se insere, o capitulo 3 descreve as *frameworks* estudadas e o capítulo 4 introduz as experiências realizadas e apresenta e discute os resultados experimentais; finalmente o capítulo 5 apresenta as conclusões do trabalho realizado e aponta possível trabalho futuro.

2

Estado da Arte

2.1 Breve História

Tudo começa nos anos 40, quando McCulloch e Pitts, em 1943 [MP43] apresentam as redes neuronais para serem usadas como base de um modelo que procura simular o funcionamento do cérebro humano. Contudo, quer as restrições computacionais, quer a parca compreensão pelos seus pares obstaculizaram a sua aplicabilidade.

Em 1965, vislumbrou-se a primeira teoria sobre *feed-forward*[1], quando Ivakhnenko e Lapa apresentam a primeira publicação sobre *"supervised deep feed-forward multilayer perceptrons"* [Sch14].

Nos anos 90, as redes neuronais convoluídas[2] foram introduzidas por Yann LeCun [NNT+98], mas só em 2006, e anos seguintes, surge Hinton [HOT06] com um novo estilo de redes neuronais[3], conhecidas

[1] onde a primeira camada representa a entrada e a última camada a saída. Quando se tem mais de uma camada escondida estamos a falar de aprendizagem profunda.

[2] em inglês, Convolutional Neural Network, de acrónimo CNN.

[3] de acronimo RN, en inglês Neural Network de acronimo NN

por redes neuronais profundas. Esta inovação originou o renascimento dos modelos teóricos que, apoiados pelo desenvolvimento tecnológico ao nível computacional, permitiram obter resultados práticos muito interessantes e consequentemente a promoção da inteligência artificial. Assim, com esta nova metodologia, novos métodos e novos algoritmos foram desenvolvidos, e, por intermédio da aprendizagem profunda, começaram-se a solucionar problemas de forma rápida e eficiente.

Desde o início de 2016, que a AP tornou-se numa área de investigação científica muito ativa, levando ao incremento de artigos publicados (em formato *preprint* ou em jornais da especialidade), assim como conferências e *workshops* sobre esta temática. Para além dos artigos, existem livros, blogs, tutoriais, vídeos e cursos.

Alguns dos nomes sonantes na área da AP são:

- LeCun que impulsionou grandes desenvolvimentos nesta área, em parte devido ao seu empenho;
- Geoffrey Hinton (designado pelo padrinho da AP, sendo o criador das RNA[4] e divulgador das máquinas restritas de Boltzmann[5]);
- Ian Goodfellow e Yoshua Bengio, pelas suas excelentes contribuições neste domínio.

Por outro lado, a existência de competições como a promovida pela ImageNet (ILSVRC[6]) fizeram emergir novas estruturas de redes neuronais e testar os métodos existentes utilizando dados devidamente classificados.

Subjacente a todo este desenvolvimento está a necessidade do ser humano adquiri mais saber. Um fator que potenciou este desenvolvimento foi a evolução conseguida ao nível computacional, e, que permitiu a AP ganhar novo fôlego. Consequentemente, os seus modelos podem agora ser utilizados por exemplo, para construir ferramentas de apoio à decisão, descoberta de drogas, processamento da linguagem natural, aplicações na área da toxicologia e da genética, reconhecimento automático do discurso, gestão da relação com os consumidores, entre outros. Agora, sistemas mais complexos podem ser analisados, estudados e previsões podem ser realizadas com um grau de erro muito reduzido. Mas tal como em outros ramos da ciência a preparação, os preliminares, ditam o sucesso ou o insucesso.

A AP é o estado da arte na área da perceção e é uma técnica central utilizada para dotar os computadores, robôs, as máquinas em geral, de capacidade de perceção (visual e/ou auditiva) e entendimento. O Facebook, a Microsoft, a Google e outras empresas usam a AP nos seus produtos, impulsionando a investigação e o desenvolvimento nesta área.

2.2 Aprendizagem Automática

A aprendizagem automática pode dividir-se em: supervisionada, semi-supervisionada, não supervisionada e por reforço[Nil98].

Na AA, os métodos usados com dados supervisionados (i.e., dados em que é conhecida a classe a que pertencem) são diferentes daqueles que não são supervisionados. Por esta razão, os algoritmos utilizados dependem do conjunto de dados e do propósito a atingir, delimitando o caminho a seguir.

Nas subseções seguintes serão descritos alguns conceitos gerais e os tipos de aprendizagem automática.

[4] em inglês, Artificial Neural Networks, de acrónimo ANN
[5] em inglês, Restricted Boltzmann Machine, de acrónimo RBM
[6] mais informação disponível em www.image-net.org/challenges/LSVRC/

2.2.1 Conceitos Gerais

Existem alguns conceitos que devem ser claros para que melhor se compreendam os tipos de aprendizagem, tais como:

- exemplo - é um objeto, elemento em estudo;
- atributo - é uma das características do exemplo; cada atributo tem associado um tipo de dados (binário, inteiro, real, enumerado, estruturado, etc);
- classe - identifica a categoria/tipo a que o exemplo pertence;
- parâmetros - são variáveis utilizadas pelos algoritmos;
- divisão dos dados - os dados podem ser divididos em conjuntos de: treino (serve para treinar o sistema de aprendizagem, comparando as entradas com as saídas), validação (serve para estudar a eficiência do modelo e redefinir os parâmetros) e teste (serve para avaliar o modelo, devendo este conjunto ser usado no final e permanecer independente dos outros dois).

2.2.2 Aprendizagem Supervisionada

Na aprendizagem supervisionada, os dados de entrada são usados para treino, recorrendo a um modelo que visa o menor erro e que se corrige automaticamente, alterando os pesos das entradas no sistema. Existem duas abordagems, dependendo dos autores, ou dividem-se os dados em dados de treino e de teste, ou dividem-se em dados de treino, validação e teste. A segunda hipótese é a mais fidedigna pois permite manter isolados os dados de teste do modelo de aprendizagem. Desta forma, conseguem afinar parâmetros do algoritmo supervisionado usado para criação do modelo. Os dados de treino encontram-se distribuídos por classes. Sendo esta um atributo que serve como orientador na criação do modelo, é uma referência. Estes dados são usados na resolução de problemas de classificação e de regressão, recorrendo aos algoritmos adequados para o efeito.

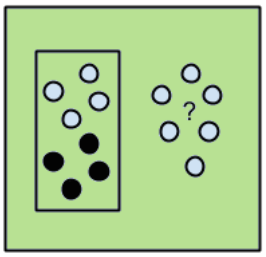

Figura 2.1: Esquema de classificação de um sistema supervisionado com o conjunto de dados divididos em duas partes.

A classificação corresponde a atribuir a uma dada entrada uma designação. Por exemplo, se virmos "A" numa folha de papel, e atribuímos-lhe um dado significado, representado por A. Para a classificação ser eficiente, é necessário ter um conjunto de dados para treino, dados esses que conterão vários exemplos da representação de "A". Se durante a fase de treino existir o rótulo "A"associado a estes dados, a associação e o agrupamento dos dados de acordo com a classe, fica facilitada.

A classificação, no contexto da aprendizagem automática, consegue-se através da utilização de funções e de métodos que promovem a identificação. Alguns classificadores, muito utilizados, em mineração de dados

são também aplicados na aprendizagem automática, como por exemplo as árvores de decisão. Por sua vez, a regressão, permite realizar previsões, dado que o resultado de uma regressão é uma função representativa do sistema e essa função procura descrever a realidade da forma mais exata possível. A figura 2.2 coloca em evidência a diferença entre classificação e regressão.

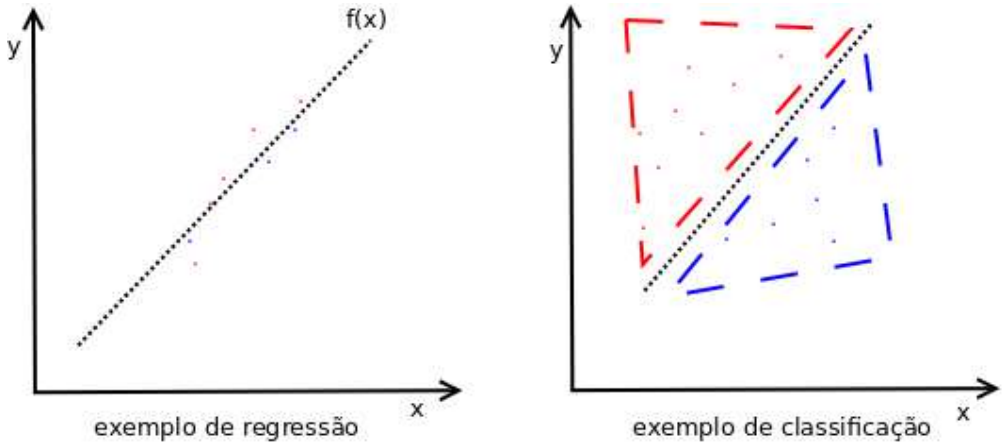

Figura 2.2: Esquema diferenciador dos métodos de regressão e de classificação.

2.2.3 Aprendizagem Semi-Supervisionada

Na aprendizagem semi-supervisionada, nem todos os dados de entrada têm uma classe associada, contrariamente ao que ocorre na aprendizagem supervisionada. Os dados classificados são substancialmente inferiores aos não classificados e o modelo escolhido terá de organizar os dados e posteriormente fazer previsões. A ideia deste tipo de aprendizagem é usar numa fase inicial os dados classificados para o treino inicial do modelo, seguindo-se a aprendizagem com recurso a dados não classificados. Este método permite que a exatidão do modelo seja consideravelmente melhorada. Numa perspetiva economicista, a utilização de dados classificados implica um custo acrescido. Tal como o modelo de aprendizagem anterior, este também é usado na resolução de problemas de classificação e de regressão.

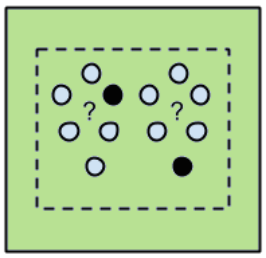

Figura 2.3: Esquema de classificação de um sistema semi-supervisionado com o conjunto de dados.

2.2.4 Aprendizagem não Supervisionada

Na aprendizagem não supervisionada, os dados de entrada não estão classificados, pois não se conhece a classe a que pertencem. Neste caso, a tarefa será promover uma inferência, procurando uma função que descreva a estrutura das camadas escondidas ou desconhecidas Como não existem dados classificados, não é possível calcular a exatidão da estrutura encontrada. Este tipo de aprendizagem assemelha-se ao modo

como o cérebro humano funciona realmente.

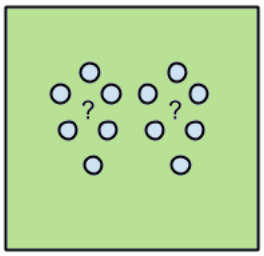

Figura 2.4: Esquema de classificação de um sistema não supervisionado com o conjunto de dados.

2.2.5 Aprendizagem por Reforço

Na aprendizagem por reforço, os dados de entrada são fornecidos com o intuito de obter uma resposta do sistema, avaliar e decidir qual o passo seguinte. Aqui é utilizado o princípio de recompensas e penalizações. Assemelha-se a uma metodologia educacional de uso comum no nosso ensino. São usados, por exemplo na robótica para permitir o controlo dos robôs.

2.3 Redes Neuronais

Desde os anos 40, que investigadores de diversas áreas e engenheiros tentam construir um sistema que reproduza o modo de funcionamento do cérebro humano. Com o aparecimento das RN, por intermédio de Warren McCulloch e Walter Pitts [MP43], esta área de investigação teve um avanço significativo.

As RN têm as seguintes características:

- são constituídas por várias unidades de processamento
- existem "pesos", coeficientes de ponderação associados à ligação entre as unidades de processamento
- o processamento é distribuído e pode ser paralelizado
- a aprendizagem é conseguida por intermédio do ajuste dos "pesos"ou coeficientes de ponderação

Tal como no cérebro humano, numa RN a unidade de base de processamento é o neurónio. No entanto, existem diferenças significativas entre ambos os sistemas; a Tabela 2.1 apresenta uma comparação.

Cérebro	**Rede Neuronal**
1011 neurónios com 1014 sinapses	Processador único com circuitos complexos
Velocidade: 10-3	Velocidade: 10-9
Processamento Não Linear	Processamento Linear
Processamento Distribuído	Processamento Central
Processamento Paralelo	Processamento Sequencial

Tabela 2.1: Comparação entre o cérebro humano e uma rede neuronal [Fre].

O cérebro controla toda a informação por intermédio dos neurónios, pelo que essa funcionalidade foi incluída também nas redes neuronais, ou seja, numa RN pretende-se obter um valor para a saída quando

é fornecida uma entrada à rede. Tal como acontece com o neurónio biológico, que recebe estímulos de diferentes fontes; o neurónio artificial também funciona da mesma maneira.

No neurónio humano, os dentritos são os sensores que recebem a informação e depois a distribuem para o local adequado. A agregação de toda essa informação é feita por intermédio do corpo celular e a saída vai ser realizada pelo axónio que está ligado a outro neurónio pelos terminais do axónio.

No neurónio artificial, que pretende reproduzir o sistema biológico, os sinais de entrada são recebidos e a cada um deles é atribuído um coeficiente, sendo toda esta informação agregada por intermédio de uma função soma. Por fim, a função de ativação vai atribuir uma função aos dados obtidos pela função soma.

A figura 2.5 é representativa das semelhanças entre um neurónio biológico e um artificial.

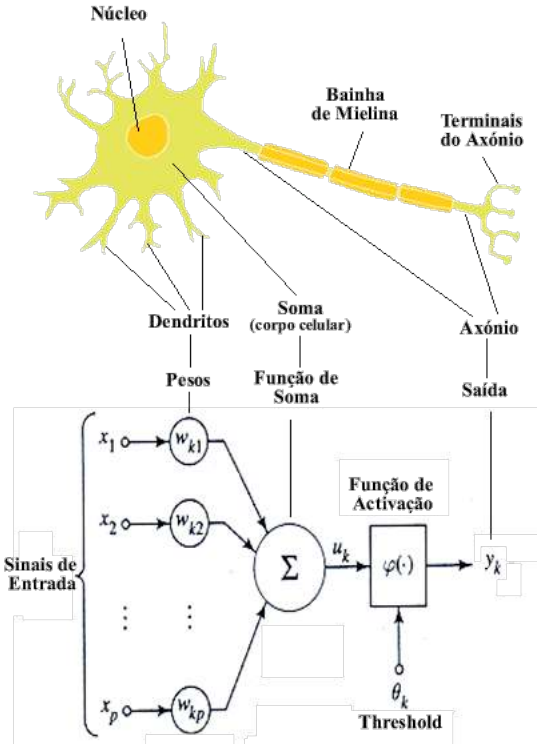

Figura 2.5: Neurónio Biológico (cima) vs Neurónio Artificial (baixo) [Fre].

Nas subsecções seguintes serão descritos os componentes neuronais, devidamente enquadrados.

2.3.1 Visão Histórica

O processo evolutivo das RN torna-se mais evidente se olharmos para uma reta temporal, numa perspetiva cronológica. A figura 2.6 esquematiza essa evolução.

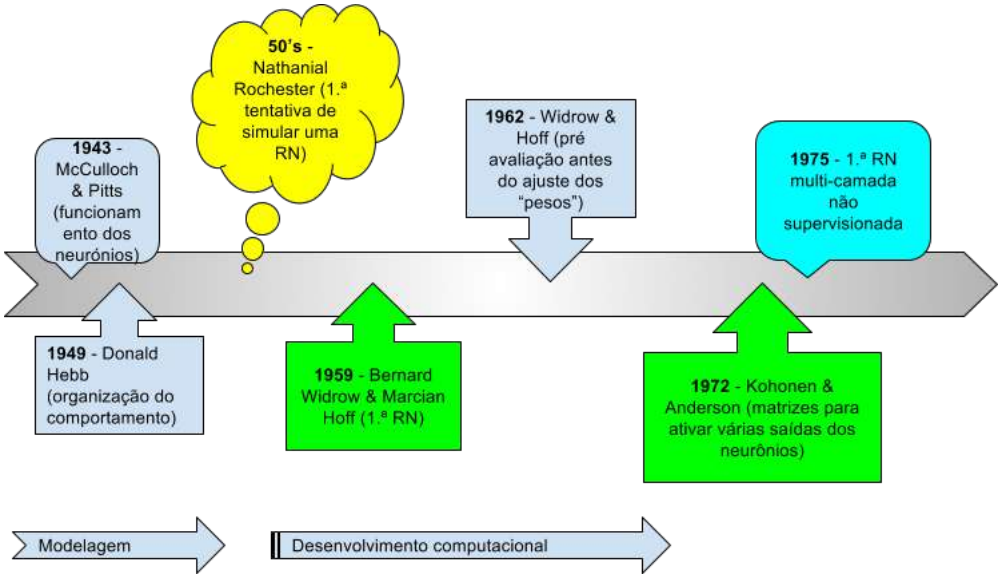

Figura 2.6: Cronologia da evolução das RN (entre os anos 40 e os 90)[Roba].

A figura 2.6. representa de forma cronológica o surgimento do primeiro modelo, desenvolvido por McCulloch Pitts, que serviu de base às redes neuronais. Pois descrevia o funcionamento dos neurónios. Uma década depois surge a primeira rede neuronal, por intermédio de Bernard Widrow Marcian Hoff. Antes dos anos 50 os recursos computacionais não permitiam o desenvolvimento das redes neuronais. Por isso, todo o trabalho consistia em criação de modelos que permitissem replicar o sistema neurológico humano.

Os constituintes de rede neuronal, ou como designado anteriormente, os componentes neuronais são simplesmente unidades que a rede neuronal utiliza para se assemelhar ao sistema biológico equivalente. O Perceptrão e as redes multi-camada surgem como tipos de redes neuronais.

2.3.2 Perceptrão

Uma rede neuronal artificial é um sistema que liga várias unidades e é constituído por várias camadas. Os seus principais componentes são os neurónios artificiais e os pesos atribuídos a cada uma das entradas na rede.

O perceptrão é um tipo de neurónio artificial utilizado em RN, e, é caracterizado por (ver figura 2.7):

- uma camada de entrada X;
- uma função soma;
- uma função de ativação;
- uma saída O.

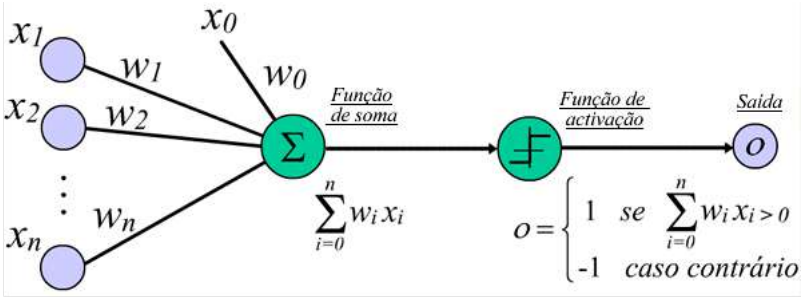

Figura 2.7: Perceptrão[Fre].

O seu funcionamento processa-se da seguinte forma:

1. a informação surge à entrada sob a forma de um vetor, X, ao qual é associado um coeficiente W_i a cada componente do vetor;

2. que depois são adicionados através da função soma ($\sum W_i.X_i$, para $i = 1, ..., n$);

3. a função de ativação funciona como uma função sinal que cuja saída é 1 quando o resultado da função soma é superior a 0, ou -1 no caso contrário.

2.3.3 Redes Multi-Camada

Uma rede multi-camada é constituída por uma camada de entrada, uma de saída e várias camadas escondidas. Por isso mesmo, permite expressar uma grande variedade de superfícies não lineares. A figura 2.8 permite visualizar a arquitetura de uma rede deste tipo.

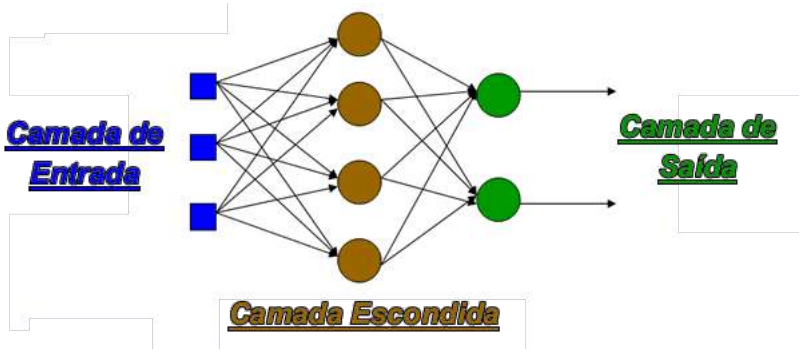

Figura 2.8: Rede Neuronal Multi-Camadas[Fre].

O número de camadas escondidas deve ser ajustado à complexidade do problema.

Por ser um sistema tão versátil, não parece adequado a utilização de métodos lineares para diferenciar as entradas. Recorre-se a uma unidade não linear, o *sigmoid* (ver figura 2.9).

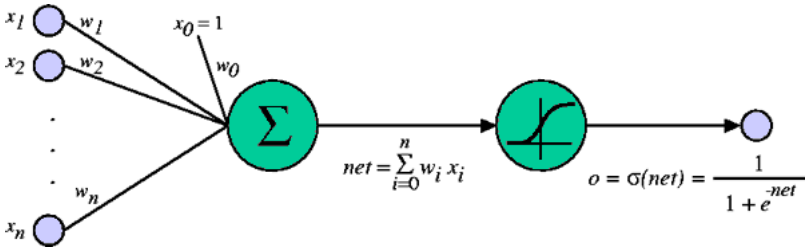

Figura 2.9: Unidade *sigmoid* com *threshold*[Fre].

A função *sigmoid* é um caso particular de uma função logística e a sua representação gráfica e respetiva fórmula são apresentadas na figura 2.10.

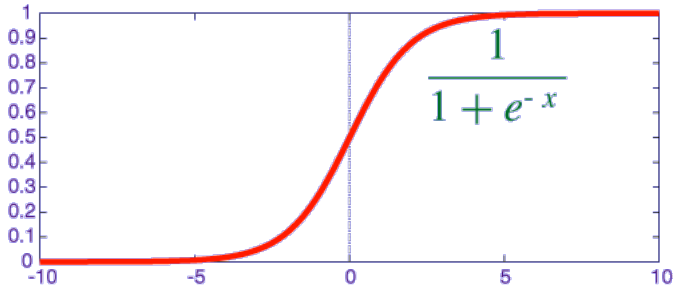

Figura 2.10: Representação gráfica da função sigmoid e respetiva fórmula matemática.

Esta função é utilizada como função de ativação devido às suas caraterísticas:

- os valores de x podem ser reais e é uma função diferenciável
- a sua forma assemelha-se bastante ao modo de funcionamento dos neurónios biológicos, pois permite saber quando é que um dado neurónio está ativo ou não

No entanto, outras funções matemáticas podem ser utilizadas como funções de ativação, como por exemplo, Tanh[7] e ReLU[8].

2.3.4 Áreas de Aplicação

As RN são bastante úteis em diversas áreas, como por exemplo na resolução de problemas da medicina, da bioinformática, entre outros. Dada a sua versatilidade podem lidar com dados em modo de texto, de imagem, de vídeo e de som. Por isso mesmo são tão apetecíveis no campo da robótica e da automação. Bons exemplos práticos da sua aplicação são: o reconhecimento de caracteres, a compressão de imagens, a previsão do valor das ações no mercado de capitais, entre outras [Robb].

[7] tangente hiperbólica.
[8] em inglês, Rectified Linear Unit.

2.4 Aprendizagem Profunda

A AP surge como uma nova camada de abstração da AA que permite resolver problemas mais complexos e obter graus de exatidão superiores aos obtidos com métodos de AA. As suas áreas de aplicação também são bastante diversificadas, intervindo no campo da imagem, video, som e robótica. Consiste numa ferramenta fundamental para obter resultados fidedignos de forma rápida [Broa, GBC16].

2.4.1 Definição

Bengio [Benc, Benb], define AP como "algoritmos de aprendizagem automática baseados na aprendizagem de múltiplos níveis de representação/abstração" [9]. Contudo existem outras definições, como:

- "Um sub-conjunto de aprendizagem automática que utiliza algoritmos capazes de aprenderem a representar as relações de elevada complexidade entre os dados. Utiliza-se uma arquitetura profunda que consiste em construir modelos mais complexos a partir dos mais simples. A maioria destes modelos baseiam-se em representações não supervisionadas."[10] [Wikipédia Março de 2012]

- "Aprendizagem Profunda (aprendizagem profunda estruturada, aprendizagem organizada hierarquicamente ou aprendizagem profunda automática) é um ramo da aprendizagem automática que se baseia numa série de algoritmos que tentam modelar elevados niveis de abstração dos dados através do uso de multiplas camadas de processamento, com estruturas complexas ou então, compostas por multiplas transformações não lineares."[11] [Wikipédia Abril de 2016]

Compreende-se esta constante alteração da definição da AP, porque esta é uma área recente e muito dinâmica, o que promove uma alteração do respetivo conceito.

Perspetivando a forma como a AP se insere dentro da área da IA e da AA, temos o esquema da figura 2.11 que é bem representativo da forma como esta está integrada na IA e na AA.

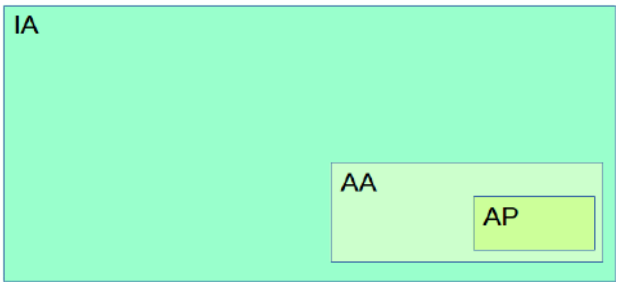

Figura 2.11: Representação esquemática da IA, da AA e AP.

[9] "Deep Learning: machine learning algorithms based on learning multiple level of representation/abstraction"

[10] (versão original) "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. "

[11] (versão original) "Deep Learning (deep structured learning, hierarchical learning or deep machine learning) is a branch of machine learning based on a set of algorithms that attempt to model high-level abstractions in data by using multiple processing layers, with complex structures or otherwise, composed of multiple nonlinear transformations."

2.4.2 Visão Histórica

Em termos efetivos, a AP é uma área relativamente recente. O conceito surgiu por volta de 2006, por intermédio de Hinton. Historicamente, Hinton construiu a primeira rede que conseguia corrigir erros em 1986 [HOT06] e em 1989, LeCunn apresenta o seu primeiro trabalho nessa área[LCJ], que veio a ser muito útil na área da visão computacional.

Por outro lado, Sepp Horchreiter e Jurgen Schmid constroem, em 1991, uma rede com memória.Esta vai tornar-se muito útil no processamento da linguagem natural [12].

De entre os vários sucessos obtidos com os métodos da IA, foi em 2007 que na área da imagem, Fei-Fei Li funda o ImageNet que vem a ser utilizado por investigadores do mundo inteiro na área da AA.

Desde esta data tudo parece ser possível e se repararmos nos dados apresentados na tabela 2.2, percebe-se o acréscimo da velocidade de desenvolvimento neste ramo da IA, comparativamente com o passado.

Ano	Autor	Desenvolvimento
2011	Microsoft	RN no reconhecimento de voz
2011	IBM Watson	vence dois campeões do concurso Jeopardy usando técnicas tradicionais de IA
2012	Google Brain	reconhecimento de gatos (dados não supervisionados)
2012	Google	RN no reconhecimento de voz
2012	dois alunos de Hinton	ganham a competição do ImageNet com uma RN
2013	Google	melhoramento da pesquisa de imagens com recurso a RN
2014	Google + DeepMind	técnicas de aprendizagem por reforço e de AP
2015	Microsoft	vence a competição da ImageNet com recurso a RN
2016	DeepMind AlphaGo	vence o campeão mundial do Go[13] na razão de 4:1, com técnicas de AP

Tabela 2.2: Desenvolvimentos na última década na área de RN, AA e AP [Par].

2.4.3 Áreas de Aplicação

A AP surge como uma nova área de AA, tendo a capacidade de obter melhores resultados do que os métodos de AA. Talvez por isso a sua área de ação é tão abrangente, cobrindo as seguintes áreas: linguagem musical simbólica, processamento natural da linguagem, texto, visão, discurso, voz e som.

A AP permite colorir imagens a preto e branco, adicionar sons a filmes mudos, tradução automática, processos de classificação, criação automática de caracteres escritos à mão, geração de imagens e automatismo de jogos[Broa, GBC16].

Como é a abordagem que mais se assemelha ao modo de funcionamento do cérebro humano, as suas áreas de aplicação só estão limitadas por dois fatores: a investigação desenvolvida nesta área e a imaginação aliada à exigência de novos desafios.

2.4.4 Algoritmos Principais

Segundo Michael Copeland [Cop], a IA consiste numa forma de dotar as máquinas de inteligência, tornando-as semelhantes aos humanos. Por sua vez, a AA é uma abordagem para atingir a IA. No entanto, é com a AP

[12] em inglês, Natural Language Processing, de acrónimo NLP

que surgem as técnicas que são verdadeiramente eficientes na implementação da AA, ou em última análise de dotar as máquinas de uma inteligência artificial. Esta aprendizagem pode ser conseguida recorrendo a árvores de decisão, a regressões lineares ou a redes neuronais com coeficientes de ponderação atribuídos às diferentes entradas[AG]. Concretamente, no domínio da Ciência dos Dados, a aprendizagem consiste num dos itens que a compõem. Assim, dentro dos algoritmos de aprendizagem existem aqueles que dizem respeito à AP. É por isso importante identificar quais os algoritmos mais conhecidos em AP, pois são estes que permitem criar diferentes arquiteturas de redes neuronais profundas, e, que são[oD]:

- RBM (*Deep Boltzmann Machine*) [Wikg]- consiste numa rede neuronal artificial que permite a aprendizagem mediante um conjunto de entradas e com recurso a métodos estocásticos. É uma variante das máquinas de Boltzmann [Wika] desenvolvida por Hinton. A sua aplicação é bastante diversificada, sendo usada na construção de modelos preditivos úteis para previsão de preços, de sotcks, ferramentas de apoio à decisão, entre outras.

- DBN (*Deep Belief Networks*) [Wikc] - é um tipo de rede neuronal profunda, constituída por múltiplas camadas escondidas. Onde existe ligação entre elas mas não entre os seus constituintes. Um conjunto de RBM pode ser usado para constituir uma DBN. [DY] A sua área de aplicação mais marcante é no processamento da linguagem natural[Hea15a].

- CNN (*Convolutional Neural Network*)[Wikb] - também pode ser designada por ConvNet e é um tipo de rede neuronal artificial do tipo "Feed-Forward" em que a ligação entre os neurónios da rede é inspirada no córtex visual dos animais. A sua maior área de aplicação é na visão computacional[Hea15a].

- SAE (*Stacked Auto-Encoders*)[Ng] - até aqui, na AP, utilizam-se várias camadas para construção da rede. Neste caso, utilizam-se várias camadas de codificadores/descodificadores na própria rede. Começaram por ser utilizados em sistemas de aprendizagem sem supervisão. Estes usam as matrizes das DBN para codificação e descodificação dos dados de entrada. Neste modelo não é utilizado o *dropout* utilizado no CNN, mas como alternativa adiciona-se à entrada ruído aleatório. Obriga-se o sistema a obter saídas que correspondam às entradas não distorcidas, como o ruído é adicionado de forma aleatória o modelo de aprendizagem fica mais robusto, e, como cada entrada ruidosa é diferente, a dimensão dos dados de treino aumenta, reduzindo o *overfitting*. [DY] As aplicações mais correntes deste método são: processamento de liguagem natural, tradução automática, análise de sentimentos, entre outras.

Desde as RBM aos SAE, ocorreu uma evolução com o intuito de dar uma resposta mais eficiente e rápida. Notando-se que existem áreas de aplicação comuns a todos estes modelos.

Para tornar claro o processo evolutivo nesta área apresenta-se um diagrama que descreve os desenvolvimentos principais na última década para o modelo de rede CNN (ver figura 2.12).

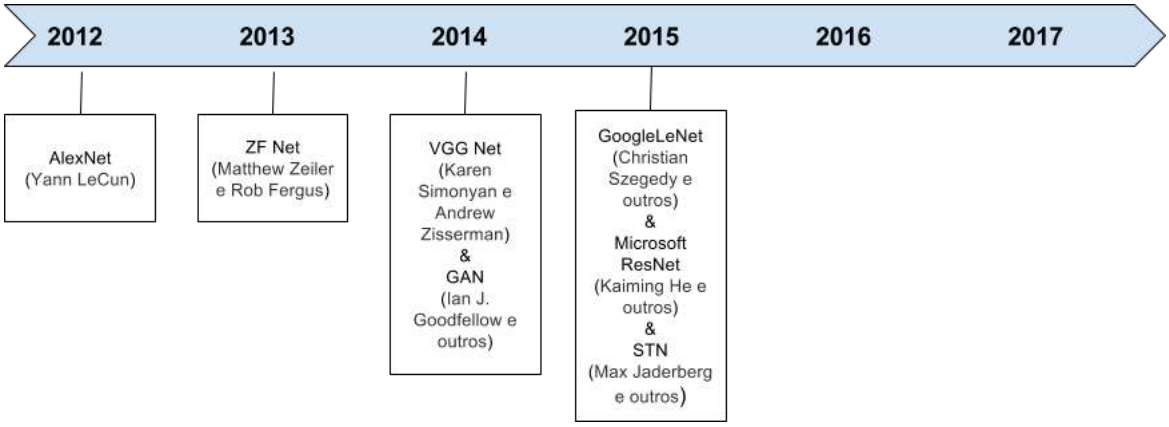

Figura 2.12: Desenvolvimentos no modelo CNN [Par].

No próximo capítulo serão abordadas as *frameworks* estudadas.

3

Estudo de Frameworks

3.1 Introdução

De acordo com Gokula Krishnan Santhanam, de RTH Zurich [San], existem quatro requisitos fundamentais para as ferramentas de AP:

- possibilidade de trabalhar com tensores [1] (o que implica a existência de um objeto tensorial sobre o qual é possível realizar operações, isto do ponto de vista de uma linguagem orientada por objetos, como é o caso do Python amplamente utilizado neste domínio)
- capacidade para computação gráfica e de otimização
- ferramentas de auto-diferenciação [AGB15]
- existência de extensões que permitam trabalhar com BLAS/cuBLAS[2] (implementação subrotinas de álgebra linear elementar que é executada nos processadores gráficos - GPUs) e cuDNN[3] (bibliotecas

[1] os tensores são objetos matemáticos associados à geometria e que são compostos por matrizes cujos constituintes são funções matemáticas.

[2] mais informação disponível em https://developer.nvidia.com/cublas

[3] mais informação disponível em https://developer.nvidia.com/cudnn

que permitem a implementação de redes neuronais profundas que são executadas nos processadores gráficos)

Estes permitirão tornar a Framework numa ferramenta útil e completa. Outros fatores não referidos pela autora, mas que são essenciais para a sua utilização têm a ver com a facilidade de instalação, configuração, implementação e a usabilidade, bem como a velocidade de execução do código. Estes parâmetros vão ser estudados para as três frameworks consideradas, de todo o espectro disponível. A próxima secção introduz frameworks para AP, ferramentas importantes utilizadas neste trabalho e plataformas para grandes quantidades de informação.

3.2 Frameworks, ferramentas e plataformas

A motivação que esteve na origem da escolha das frameworks deve-se à representatividade que elas possuem, com é o caso de Theano [Thea] e do TensorFlow [Tenc]. A terceira framework foi considerada principalmente por ser extremamente recente e ter potencialidades para ser uma das mais representativas num futuro próximo, a Microsoft CNTK [Mice].

Existem, no entanto, outras frameworks tais como [May, Tena]:

- OpenCV [Ope] - embora já exista desde 1999, começa a mostrar capacidade de entrar nesta área, a partir de 2010, principalmente com as suas valências ao nível de aplicações móveis e multimédia, por exemplo associar a realidade aumentada, ao IoT e ao AP;
- Torch [Tor] - utiliza como base a linguagem Lua [LUA] que está pouco divulgada;
- RStudio [RSt] - é uma interface gráfica para uso da linguagem R [Fou], permitindo a integração de diversos componentes;
- Matlab [Mat] - possui o Deep Learning Toolbox e é software proprietário;
- Caffe [Jia] - a sua implementação é difícil, bem como a instalação que parece estar repleta de "bugs", é um sistema pouco versátil;
- Lasagne [Las] - consiste numa biblioteca de python implementada numa nova camada do Theano, mas com a qual não é muito fácil a implementação. Depois do estudo realizado, nesta tese, torna-se claro que quando comparada com o Keras parece que o código deste é mais simples e claro;
- Keras citeKeras - consiste numa biblioteca de python implementada como uma nova camada do Theano e do TensorFlow. Possui um código claro e transparente. Por defeito esta usa o TensorFlow em segundo plano;
- MxNet [MxN] - é uma framework de AP que permite definir, treinar e implementar redes neuronais profundas para diversos dispositivos, desde os móveis até à Cloud;
- Deep Learning 4j [DL4] - utiliza a linguagem Java [SO], podendo ser útil, por exemplo, na área das aplicações móveis desenvolvidas para Android. Nomeadamente, naquelas que exigem analise de dados em tempo real, como por exemplo aplicações de realidade aumentada. No entanto, são precisas muitas linhas de código para a implementação, devido à estrutura da própria linguagem. Utiliza o Maven [Apab], também conhecida por Apache Maven, e, é uma ferramenta de automação de compilação utilizada primariamente em projetos Java;

- SAS [SAS] - é uma plataforma vastíssima na qual começa a emergir a AP dentro deste software proprietário;
- Intel Deep Learning SDK [Int] - utiliza o Caffe como base. Facilita a implementação dado que esta é feita através de uma interface web. Mas devido a usar o Caffe tem muitas limitações;
- Chainer [Cha] - apesar de não ser objeto de estudo, é importante falar dele porque é a única framework que permite ajustar o código enquanto ele está a ser executado [Dava], o que permite poupar tempo na execução de projetos nesta área, pois não é necessário que um processo termine para o avaliar;
- DLib-ML [Davb] - é uma biblioteca de C++ que tem um Toolkit para ML e Python inclusive. Mas é a componente de C++ que está amplamente desenvolvida, pois a de Python fica-se pela deteção de objectos em imagens e por SVM [4].

Em 2015, também a Apple resolveu apostar no AP para todos os seus dispositivos e criou o DeepLearningKit que pode ser instalado em qualquer um dos dispositivos que comercializa [App]. O DeepLearningKit vem possibilitar a integração de algoritmos de AP nos sistemas operativos da Apple. Consegue importar modelos de outras frameworks, é uma biblioteca que permite utilizar a capacidade de processamento das placas gráficas e implementar modelos de AP para as aplicações da Application Store.

Figura 3.1: Representação do ano em que surgiram pela primeira vez as frameworks apresentadas.

O gráfico apresentado na figura 3.1 evidência que nos últimos 2 anos ocorreu um *"boom"* produtivo na área da AP com o surgimento de novas frameworks.

Constata-se que a linguagem de programação mais utilizada e que por isso mesmo é comum a praticamente todas as frameworks é o Python, seguida do C++ e depois algumas mais específicas como a Lua, BrainScript, Go, etc.

[4] em inglês, Support Vector Machines.

Para além das ferramentas referidas existem outras que têm a sua aplicação na área da aprendizagem automática. Aqui fica um breve sumário de algumas delas [Yeg]:

- Scikit-learn [sl] - consiste numa nova camada de abstração de Python que trabalha em cima do Numpy, Scipy e Matplotlib.
- Shogun [ST] - é uma biblioteca de AA com capacidade para trabalhar com várias linguagens: C++, Java, Python, C#, Ruby, R, Lua, Octave e Matlab.
- Accord Framework/AForge.net [Acc] - permite a implementação de algoritmos de AA em .net.
- Mahout [Apaa] - é uma framework de Hadoop[5] focada em aplicações para AA e Mineração de Dados;
- MLlib [Apac] - é uma biblioteca do Apache para ser usada no Spark e no Hadoop. Tal como a Mahout também está direccionada para aplicações de AA, embora seja muito mais eficiente pois demora cerca de 10x a 100x menos a processar a informação;
- H2O [H2O] - fortemente orientada para o Hadoop trabalha com Python, Scala e R e é mais usada no domínio financeiro (detecção de fraudes), fazendo inclusive previsões.
- Cloudera Oryx [Orx] - desenhado a pensar no Hadoop, permite trabalhar com dados em tempo real, o que pode ser bastante útil para detecção e combate ao SPAM, entre outras aplicações.
- GoLearn [Whi] - A linguagem usada é o Go (linguagem da Google), uma linguagem com apenas 5 anos e que se pauta pela simplicidade e usabilidade/versatilidade.
- Weka [EF] - muito útil no que diz respeito ao Data Mining (Mineração de Dados), pois inclui diversos algoritmos e bibliotecas que permitem analisar conjuntos de dados sem ter que escrever código. A linguagem de base é o java e existem muitas contribuições e imensas "packages"(tal como acontece no R/RStudio).
- CUDA-Convnet2 [Goob] - A linguagem de base é o C++ mas as saídas da rede implementadoa pode ser lido no formato Pickle [Pyt] usado pelo Python.
- ConvNetJS [Conb] - versão das CNN para usar no *browser*. A linguagem de base é o javascript.

Para além do já referido existem alguns utilitários de extrema importância, tais como:

- Docker [Doc] - é uma plataforma aberta usada por Administradores de Sistemas e Developers para criar, distribuir e executar aplicações em qualquer lado; funciona de como os ambientes Python[6], mas sendo mais completo, pois permite configurar todo um sistema independentemente da linguagem, da versão e da aplicação. Permite criar sistemas com configurações muito variadas, diferentes versões de compiladores, de sistemas operativos, etc. É melhor que o Git pois possui um ficheiro de configuração *"DockerFile"* que contém toda a informação necessária para a criação das imagens e dos componentes designados por "Containers"quer reproduzem com exatidão as condições em que foram criados. Pode ser considerado um produto "chave na mão", que basta ser executado para reproduzir tudo o que foi configurado;

[5]Neste momento um pouco ultrapassada, mas ainda existente em muitos sistemas que utilizam dados de grande complexidade e dimensão, e, que precisam de ser escalonáveis, fiáveis e usados em computação distribuída (mais informação disponível em http://hadoop.apache.org/).

[6]designados por defeito como "env".

- Git [Git] - é uma ferramenta útil para divulgação de código na rede, embora por vezes contenha informação pouco clara, mas que se deve essencialmente à falta de cuidado dos autores;
- Jupyther [Jup] - é uma ferramenta muito útil para o ensino, para o estudo e para a exploração de novos métodos, novas técnicas. Pois permite a execução de pequenas partes de código de forma iterativa.
- Anaconda [Cona] - é uma ferramenta que surge para facilitar a instalação de bibliotecas que de outra forma seria mais complexo, ou seja, da forma tradicional, compilando o código a partir da fonte.

Quando se quer passar para um domínio de grande dimensão de informação a ser analisada já se recorrem a outras plataformas, como por exemplo:

- Sistemas Cloud (AWS, Azure, Google) [Ama, Micb, Good]
- Sistemas de Alta Disponibilidade, que funcionam 24/7 sem falhas. Podem ser sistemas de clusters, podem ser sistemas com arquiteturas que permitem redundâncias. Estes sistemas são implementados quer ao nível de hardware, quer ao nível do software. Por exemplo, os clusters usados pela Alfresco[7].
- Big Data (Hadoop, Hive, MapReduce, HDFS, Spark, entre outras). Nesta área o Apache tem um leque diversificado de ferramentas que podem ser usadas com os mais diversos fins. Por exemplo, o Spark é usado para dividir os dados de grande dimensão em pequenas parcelas, e, depois processá-los em paralelo. Em certa medida, já o MapReduce permite reduzir a dimensão dos dados original. O Apache Hadoop, permite a escalabilidade e o HDF5, que permite criar uma organização por contentores, permitindo também o acesso aos dados. Por seu lado, o Apache Hive é um Data Warehouse. Percebe-se que o Apache tem muitas ferramentas que podem ser utilizadas por quem trabalha com dados de grandes dimensões[8].

3.3 *Frameworks* Estudadas

Dada a grande quantidade de frameworks disponíveis decidiu-se estudar e fazer uma análise detalhada de um subconjunto daquelas apresentadas. As frameworks escolhidas foram Theano, TensorFlow e CNTK. Nas sub-secções seguintes introduzem-se estas frameworks (e ainda a biblioteca Keras), faz-se uma análise comparativa da sua utilização segundo o número de publicações e problemas encontrados pelos utilizadores na utilização destas e na implementação do código desenvolvido por eles e apresenta-se uma avaliação pessoal sobre a qualidade da documentação, facilidade de instalação e implementação das frameworks escolhidas.

3.3.1 Theano

O Theano [Thea] é uma biblioteca de Python que permite definir, otimizar e avaliar expressões matemáticas, ou funções, com recurso a vetores, matrizes e sistemas multi-dimensionais de forma eficiente (Tensores).

As suas características permitem a integração com outras bibliotecas como por exemplo a Numpy [dev], a configuração para que o código seja processado, quer nos CPU's [9], quer nos GPU's [10]. Os cálculos podem

[7] mais informação em https://www.alfresco.com/

[8] mais informação disponível sobre estas e outras ferramentas em https://www.apache.org/.

[9] Central Processing Unit, que significa unidade de processamento central existente em todos os dispositivos computacionais

[10] Graphical Processing Unit, que significa unidade de processamento gráfico que existem nas placas gráficas existentes por exemplo nos computadores.

ser cerca de 140x mais rápidos quando processados nos GPU's, considerando que se usa float32 [Thea].

A framework consegue ainda diferenciar código simbólico, permitindo derivar funções com várias variáveis, ou seja, com várias entradas, o que se torna verdadeiramente útil para a sua aplicação nas redes neuronais. Esta é talvez a razão que a faz ser uma das frameworks mais utilizadas.

As otimizações são rápidas e estáveis. Por exemplo consegue resolver a função $log(1 + x)$ para valores de x muito pequenos. Por outro lado, consegue obter melhores desempenhos quando comparado com o processamento realizado em Matlab ou em C++ [eab].

Como o Theano usa como linguagem o Python, consegue gerar código C de forma dinâmica, o que permite a avaliação das expressões mais rapidamente. O Theano tem sido largamente utilizado na investigação científica desde 2007, e, como utiliza a linguagem Python é suficientemente simples para ser usado ao nível do ensino universitário. Um bom exemplo disso são as aulas de *"deep learning/machine learning"* da Universidade de Montreal [Basc, Theb].

Alguns dos modelos disponíveis nesta ferramenta são [Basd]:

- *Dropout*
- Rede convoluída do Alex
- *RNN CTC*
- *TreeLSTM*
- *Variational Recurrent Neural Networks*

3.3.2 TensorFlow

O TensorFlow [Tenc] é uma biblioteca de Python desenvolvida pelos investigadores da Google que trabalhavam no projeto Google Brain [Gooc] e tem como homónimo o Theano. Apesar disso existe uma diferença significativa entre este e o Theano, que consiste na possibilidade do TensorFlow usar uma biblioteca de *"Data Flow Graph"* [Tenb] para computação numérica. Esta ferramenta foi criada com o intuito de facilitar a vida a que trabalha com a AA, embora possa ser utilizada noutras áreas, dada a sua versatilidade. Devido à sua arquitetura permite com uma simples API utilizar os mais variados recursos de hardware, sejam eles CPU's ou GPU's de computadores, de servidores ou de dispositivos móveis [Tend]. O TensorFlow só suporta processamento no GPU a 32 bits, ao contrário do Theano que permite o uso quer dos 32 bits, quer dos 64 bits. Quem usa o TensorFlow pode usar uma funcionalidade designada por TensorBoard [Tenb], na qual é possível através de um endereço web testar o código e monitorizar a sua execução. O TensorFlow usa o GPU por defeito. Mas existe uma vantagem enorme nesta framework que é a capacidade de utilizar recursos disponíveis via rede computacional [eaa].

Os modelos disponíveis nesta ferramenta são [Wu]:

- *autoencoder* - diversos tipos de autoenconders.
- *differential* - modelos para respeitar a privacidade dos estudantes relativamente a professores diferentes.
- *im2txt* - rede neuronal para captar imagens e convertê-las para texto.

- *inception* - CNN profunda usada em visão computacional.
- *namignizer* - reconhecer e gera nomes.
- *neural* - permite o processamento das redes nas gráficas com possibilidade de paralelismo.
- *neural* - rede neuronal aumentada através de recurso à lógica e a operações matemáticas.
- *resnet* - composto por redes neuronais residuais profundas e alargadas.
- *slim* - classifica imagens usando modelos TF-Slim.
- *swivel* - aplicação do algoritmo Swivel [eo16] para gerar palavras embebidas.
- *syntaxnet* - contêm modelos neuronais para sintaxe em linguagem natural.
- *textsum* - usa uma abordagem sequencial de sequência a sequência, permitindo um resumo textual.
- *transformer* - rede para transformação espacial, permitindo a manipulação espacial dos dados dentro da rede.

Estes modelos foram desenvolvidos por diversos autores e são mantidos pelos mesmos [Wu].

3.3.3 Keras

A Keras [Ker] é uma biblioteca de python que pode ser usada numa segunda camada de abstração do Theano ou do TensorFlow. A sua grande vantagem é permitir escolher uma destas frameworks para ser executada no *"backend"* e facilitar o processo de implementação do código. Existem dois tipos de modelos no KERAS:

- sequencial - é um conjunto de camadas compactadas e organizadas de forma linear, ou seja, uma seguidas às outras.
- API funcional - permite com os dados de entrada e de saída da rede construir um modelo que seja representativo desses mesmos dados.

Sendo esta biblioteca facilitadora da implementação do código, optou-se por usá-la em conjunção com o Theano e com o TensorFlow, no processo de análise e comparação.

3.3.4 Microsoft CNTK

A Microsoft CNTK é um componente de software que torna fácil o desenho e o teste de redes computacionais, como as redes neuronais profundas [Mica]. Tal como os concorrentes, é utilizado para treinar e testar diversos tipos de redes neuronais. A configuração da rede é centralizada num ficheiro de texto. Tal como o Theano e o TensorFlow, também permite processamento paralelo e direcionamento do mesmo para os CPU's e/ou GPU's, suportando o sistema CUDA [NVi]. Pode ser utilizado com uma linguagem própria, a BrainScript [Basa], ou em alternativa, com Python ou C++. Os modelos disponíveis nesta ferramenta são [Micd]:

- AlexNet [KSH12] - CNN desenvolvida por Alex Krizhevsky e que venceu o ILSVRC2012 [Ima] num processo de classificação.

- AN Speech (DNN e LSTM) - modelos usados na área do discurso e em que a sua estrutura de base consiste numa rede neuronal artificial.
- ConvNet [oSa] - rede neuronal convolucional desenvolvida na Universidade de Standford.
- VGG[11]
- AE - codificação automática, em inglês designa-se por Auto-Encoder.
- Aprendizagem por reforço.

O dinamismo na implementação de modelos nesta nova framework é bem evidente. Das diversas vezes que o site da Microsoft foi consultado, o incremento de novos modelos foi significativo.

3.4 Trabalho Desenvolvido

Para a comparação entre*frameworks*, seguiu-se o seguinte procedimento:

1. procedeu-se à pesquisa das *frameworks* existentes.
2. entre elas selecionaram-se três: Theano + Keras, TensorFlow + Keras e Microsoft CNTK.
3. definiram-se os critérios de avaliação e as escalas de classificação para: instalação, implementação e documentação.
4. escolheram-se três modelos do tipo CNN para análise do comportamento das três *frameworks* seleccionadas, para um estudo mais profundo, com dados supervisionados.
5. avaliou-se o género de recursos utilizados pelas três *frameworks* em estudo; neste caso a RAM do GPU usada, visto os cálculos terem sido processados no GPU.
6. quantificou-se o número de linhas de código necessárias para a implementação de cada um dos modelos utilizados.
7. registou-se o tempo de treino para cada modelo e para cada *framework*.
8. calculou-se a exatidão, quando se altera a dimensão de dados de treino para o modelo mais elaborado, usando a framework que demonstrou ter melhor pontuação nos itens anteriormente classificados.

3.4.1 Resultados e Discussão

Colocando num gráfico as *frameworks* em função do número de publicações (sejam elas, artigos, livros, vídeos, cursos, código disponibilizado no GitHub, ou mesmo imagens de sistemas disponibilizado no DockerHub) verifica-se que as que aparentemente têm mais representatividade a nível global são o RStudio e o DL4J, talvez por o primeiro ser frequentemente utilizado por Cientistas de Dados [12] e Analistas de Dados [13] no domínio do *Big Data* e do *IoT*[14] A utilização do DL4J reflete-se pelo facto de a linguagem JAVA ser uma das mais divulgadas e utilizadas (ver gráfico da figura 3.2).

[11] CNN profunda da Universidade de Oxford. Foi o modelo vencedor na competição ILSVRC2014 na tarefa de localização.
[12] em inglês, Data Scientits.
[13] em inglês, Data Analytics.
[14] Internet das Coisas, em inglês é o acrónimo de Internet of Things, e que está relacionada com a informação disponibilizada através de diversos dispositivos e usualmente associada ao REST, que consiste em armazenar e filtrar essa informação nos browsers.

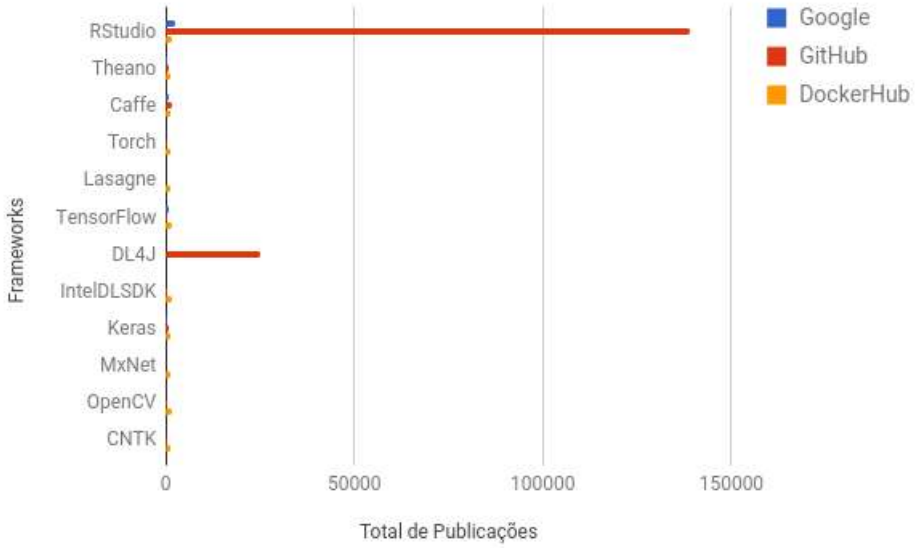

Figura 3.2: Relação entre as *frameworks* e o número total de publicações na Google, GitHub e DockerHub.

Dada a grande disparidade, para o RStudio e para o DL4J em relação ao GitHub, principalmente ao nível dos valores, resolveu-se retirar estas duas *frameworks* e analisar a relação entre as restantes. Obteve-se o gráfico apresentado na figura 3.3.

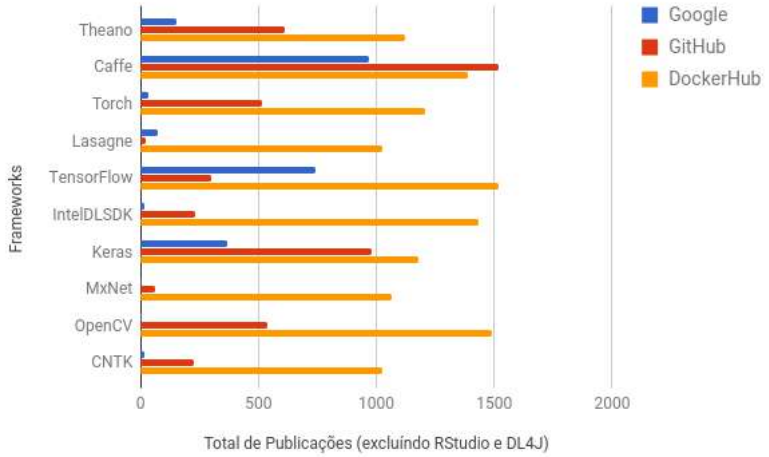

Figura 3.3: Relação entre as *frameworks* e o número total de publicações após exclusão do RStudio e do DL4J.

Da análise do gráfico da figura 3.3, infere-se que existe uma clara tendência para o DockerHub passar a ser uma ferramenta utilizada para o desenvolvimento e teste de novos sistemas. Este resultado compreende-se pelo facto de o DockerHub ser utilizável em qualquer Sistema Operativo, em qualquer rede, inclusive na Cloud e permitir construir diversas configurações que ficam automaticamente disponíveis para quem as quiser utilizar. Mas como neste trabalho o objetivo é estudar apenas três *frameworks*, vamos ver o que acontece quando se retiram todas as outras (ver gráfico da figura 3.4).

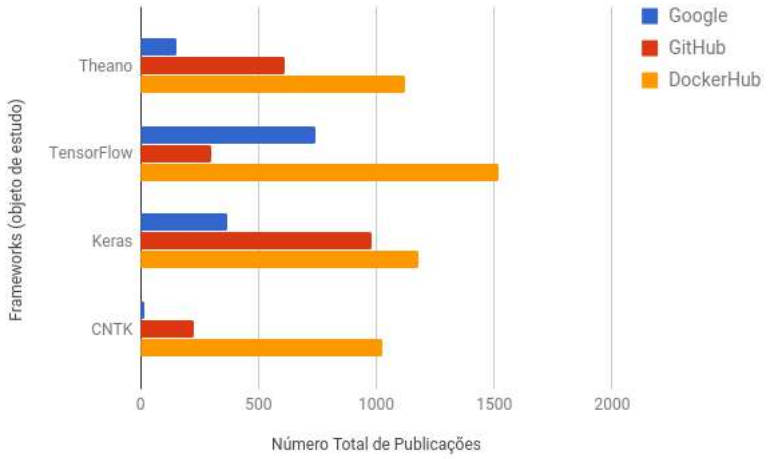

Figura 3.4: Comparação entre as três *frameworks* em análise e o número total de publicações na Google, no GitHub e no DockerHub.

Também aqui se evidencia a preponderância do Docker face aos outros sistemas. Estes dados estatísticos foram obtidos através da pesquisa "[frameworkname] deep learning" em cada um dos motores de pesquisa disponibilizados por cada um dos sistemas.

Vejamos agora outro tipo de análise que também pode ser interessante. Vamos comparar as *frameworks* com os problemas encontrados (abertos no GitHub, fechados no GitHub e no Stackoverlfow).

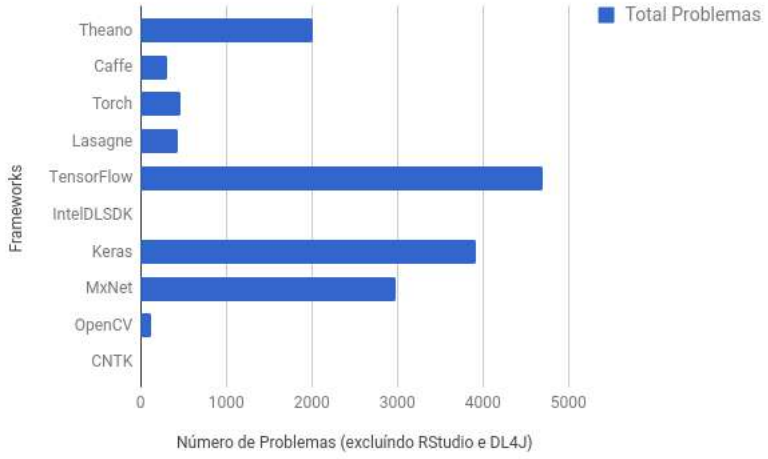

Figura 3.5: Comparação entre as *frameworks* e o número total de problemas encontrados.

Ao observarmos o gráfico da figura 3.5 temos uma noção de que o RStudio não apresenta dados nestes sistemas talvez por ser um produto "chave-na-mão", ou, por usar outro lugar para discutir e apresentar os problemas encontrados, como por exemplo um fórum específico para R. Quanto ao IntelDLSDK e ao CNTK, como são muito recentes também ainda não são amplamente usados e testados como as outras.

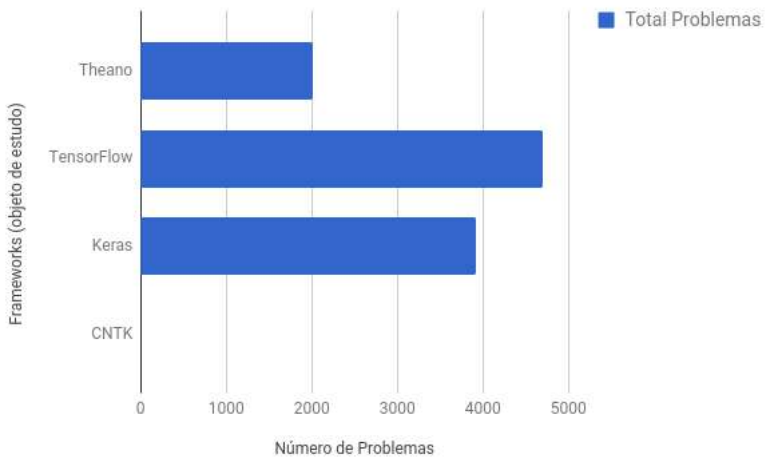

Figura 3.6: Relaçao entre as *frameworks* e o numero total de problemas encontrados.

Excluindo as outras *frameworks*, centrando a atenção exclusivamente nas selecionadas, obtêm-se o gráfico da figura 3.6.

Curiosamente constata-se que o Theano parece ter menos dissabores no que diz respeito a bug's e outro tipo de problemas que podem surgir quando se tenta instalar e quando se tenta implementar o código. Parece por isso mais maduro. O CNTK não permite grandes conclusões, pois é demasiado recente, não se tendo dados que possam servir para a análise. Os dados estatísticos deste estudo foram obtidos a 11-02-2017.

A fase a seguir da comparação passa por avaliar percentualmente as diversas *frameworks* no que diz respeito aos seguintes parâmetros: Documentação, Instalação e Implementação. Tendo sido definida uma escala para cada um deles apresentada no ponto 4.1.

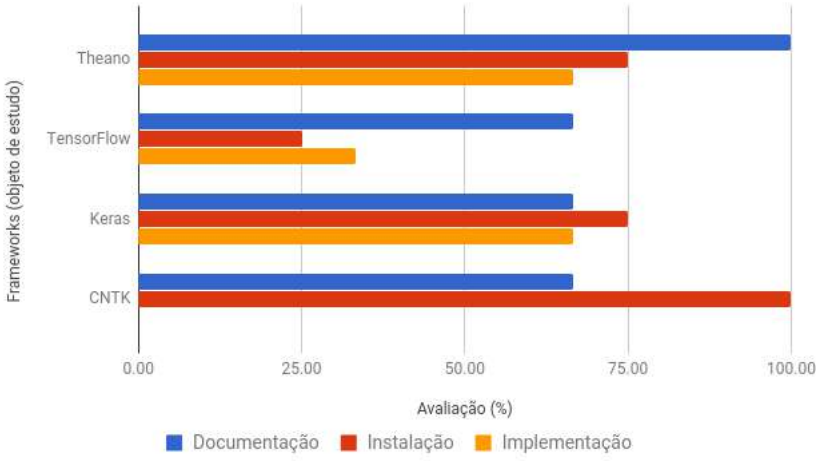

Figura 3.7: Avaliação das *frameworks* em estudo no que diz respeito à documentação disponibilizada, à instalação e à implementação.

A figura 3.7 retrata a cotação percentual obtida para cada um dos itens avaliados, evidenciando-se a facilidade de instalação da CNTK e a facilidade de implementação do Keras em conjunto com o Theano.

No capítulo seguinte, abordar-se-á um outro tipo de comparação entre elas, com três modelos CNN diferentes.

4

Experiências e Avaliação

Este capítulo contém a componente experimental deste trabalho, na qual se estudaram *frameworks* considerando diversos cenários representativos do processo de aprendizagem. Serve por isso para explicar metodologias e apresentar resultados. Descrevendo o *modus operandi* de forma sintética e clara.

4.1 Configuração Experimental

Como já referido o estudo compara as três *frameworks*: Theano, TensorFlow e CNTK, utilizando o dataset MNIST (dados supervisionados para testar métodos de AA para reconhecimento de algarismos constituídos em 2008 [LeC]. Numa perspetiva mais ambiciosa, utiliza-se o dataset STL-10 (versão melhorada do CIFAR-10), como uma das abordagens de topo para dados supervisionados/semi-supervisionados. O MNIST será utilizado para implementação de um sistema não supervisionado, que servirá de referência. O ajuste dos parâmetros dos vários modelos, não será realizado tendo em vista um processo de otimização, mas será ajustado por forma a que eles sejam constantes (na medida do possível) entre eles.

Uma vez que o estudo é meramente comparativo e exploratório das potencialidades, quer das framewoks, quer dos modelos em si, o conjunto de dados vai ser dividido em três partes: dados de treino, de validação

e de teste. Foi escolhido o modelo CNN para a fase de comparação entre as *frameworks*. Para a fase de estudo de sistemas não supervisionados utilizaram-se os dois conjuntos de dados referidos anteriormente (MNIST e STL-10), tendo sido escolhido para este efeito os modelos DeepLDA, AE e GAN.

A configuração experimental poderia passar única e exclusivamente por usar o processamento no CPU, mas esta era sem dúvida uma oportunidade para estudar algo mais evoluído, por isso para além do processamento no CPU também se usou o GPU para processamento. Numa fase inicial, contemplou-se a possibilidade de usar o OpenCL, associado a placas gráficas de arquitetura AMD para processamento no GPU, ou, em alternativa o CUDA, associado a placas gráficas de arquitetura NVIDIA. Optou-se pelo sistema cuda (em comparação com o OpenCL), visto ser este o mais amplamente desenvolvido e também por isso mesmo ser muito mais fácil a sua instalação, configuração e utilização.

O procedimento utilizado para construção da arquitetura de base para a configuração experimental que permitirá executar a componente prática desta tese foi o seguinte:

1. Primeira Fase - Os sistemas da Apple seriam o segundo cenário, tendo sido numa fase inicial utilizado um Macbook Pro de 2010 com uma placa gráfica com 320 MB de ram da NVIDIA, 4GB de RAM a 1066 Mhz, e um CPU C2D a 2.66 Ghz. Sendo neste último, que foi realizado o primeiro estudo comparativo entre o OpenCL e o CUDA para processamento nos GPU. Onde se tornou claro a grande disparidade entre estes sistemas.

2. Segunda Fase - As instalações começaram por ser realizadas de acordo com as instruções disponibilizadas pelos autores destas *frameworks*. Nesta fase foi usado um pc desktop com a seguinte configuração: sistema operativo (Ubuntu 16.04), CPU (AMD A10-5800K), GPU (NVIDIA GeForce GT 710 - 2GB, driver: 375.39), RAM (Kingston DDR3 1866Mhz - 2x4GB), SSD (Corsair - 256GB). O sistema operativo Windows foi descartado dos testes, uma vez que na área de desenvolvimento os sistemas operativos com arquitetura linux serem os mais comummente utilizados.

3. Foram analisadas as diversas etapas necessárias para a implementação, bem como se estabeleceu uma escala quantitativa para classificação das mesmas. As classificações foram atribuídas respeitando as seguintes escalas:

   - Instalação - foi usada uma escala de 1 a 4, em que:
     - **4** corresponde a uma instalação disponibilizada com recurso a um script ou a um ficheiro executável;
     - **3** corresponde à necessidade de recorrer entre 1 a 3 passos (significando com isto que pode ser necessário instalar bibliotecas que fazem parte dos requisitos);
     - **2** corresponde a uma instalação em que é necessário recorrer ao código fonte e compilá-lo;
     - **1** corresponde a uma instalação que não foi possível terminar porque a *framework* apresenta "bugs" ou que mesmo quando instalada não funciona corretamente.
   - Implementação - foi usada uma escala de 1 a 3, em que:
     - **3** corresponde a uma implementação rápida, fácil, com recurso a poucas linhas de código;
     - **2** é atribuído a um cenário intermédio;
     - **1** diz respeito a uma implementação morosa, que recorre a muitas linhas de código e de grande complexidade.

   Como foram definidas escalas distintas para cada um destes parâmetros, tornou-se necessário converter os resultados em valores percentuais para uma melhor perceção.

4. Como os recursos computacionais necessários são enormes para este tipo de implementações e a rapidez de execução é essencial para realizar um trabalho em tempo útil, recorreu-se numa segunda

fase à utilização do sistema Docker (Configuração: Ubuntu 16.04, a disponibilizar no dockerhub). Este sistema foi testado num pc desktop e posteriormente replicado numa parte da infraestrutura do LIP - Laboratório de Instrumentação e Física Experimental das Partículas[1], cujo hardware é composto por 15 CPU's Intel Haswell 2 GHz, GPU(Tesla K40m - 12GB, driver: 367.48), RAM (2x16 GB), sistema operativo Fedora 13.1.2-1.el7, Docker version 1.12.5, build 7392c3b (sistema virtualizado usando OpenStack) numa infraestrutura de Cloud [dCD]. Estes recursos foram fundamentais para realizar o estudo do modelo DeepLDA [MD].

5. Escolheu-se um conjunto de dados de base, que foi usado em todas as comparações realizadas, estudando a abordagem AP através da implementação de uma rede CNN no Theano, no TensorFlow e no Microsoft CNTK.

6. Obtiveram-se dados estatísticos, fizeram-se testes e análises.

7. O código utilizado para testes encontra-se disponível na internet (via GIT e não só), em livros e na documentação específica de cada *framework*, tendo posteriormente sido adaptado de maneira a que os testes realizados fossem iguais, apenas divergindo na *framework* utilizada, pois seria esse o objetivo, a comparação entre elas.

Posteriormente foi escolhida a *framework* que apresentava melhores resultados para as experiências seguintes. Estas consistiam em:

1. Aplicação do método DeepLDA para dois conjuntos de dados diferentes (MNIST e STL-10) para dados semi-supervisionados.

2. Utilização do conjunto de dados MNIST com dados não supervisionados e comparação entre dois dos métodos utilizados nesta área: AE e GAN.

4.2 Experiências e Resultados

Embora o estudo tenha sido centralizado em três *frameworks*, nesta etapa foram analisadas mais *frameworks*, como se pode visualizar no gráfico da figura 3.2.

4.2.1 Comparação entre *frameworks*

No processo de comparação entre *frameworks* procedeu-se numa primeira fase a uma comparação mais genérica que contemplava a instalação e o estudo destas. Neste caso, utilizaram-se as versões indicadas na tabela 4.2.1.

[1] mais informação disponível em www.lip.pt

Framework/Biblioteca	Versão
Theano	0.9.0dev3
TensorFlow	0.12.1
Keras	1.0.8
CNTK	2.0 beta12
RStudio	1.0.136
Torch	7
Caffe	-
Lasagne	0.2.dev1
DL4J	-
MxNet	0.9.3
OpenCV	3.2
Python	2.7.12
IntelDLSDK	-

Tabela 4.1: Versões utilizadas na fase geral de comparação entre *frameworks*.

Numa segunda fase, foram escolhidas três *frameworks* para uma análise mais detalhada, análise esta que passaria pela implementação de uma rede CNN simples, uma mais elaborada e uma igual à anterior mas com aplicação da técnica de Dropout[2]. Nesta experiência os dados utilizados fazem parte do conjunto de dados MNIST e são dados supervisionados.

Construindo o modelo no Keras e usando o Theano como "backend" a velocidade de processamento é superior, e a ocupação da memória é gerida de forma automática necessitando de menos recursos do que quando comparado com o processamento utilizando o modelo criado pura e simplesmente no TensorFlow. Um dos grande problemas do TensorFlow tem a ver com os recursos que exige, em termos de memória, e com as alternativas que propõe para que se possa gerir a memória. A sua implementação é sem sombra de dúvidas mais complicada (mais ao modo C++, do que ao modo Python), ao contrário do Keras e do Theano que são extremamente práticas e claras, o que simplifica em larga escala o desenvolvimento, a implementação e inclusive a leitura do código.

A selecção entre a utilização do Theano ou do TensorFlow em *"backend"* é realizada em /.keras/keras.json, cujo conteúdo deve ser:

```
"image_dim_ordering": "tf",
"epsilon": 1e-07,
"floatx": "float32",
"backend": "tensorflow"
```

quando se pretende usar o TensorFlow e deve ser

```
"image_dim_ordering": "th",
"epsilon": 1e-07,
"floatx": "float32",
"backend": "theano"
```

quando se pretende usar o Theano em *"backend"*.

Aplicando o modelo CNN, que consiste num modelo de uma rede neuronal convoluída, ou seja, é uma rede neuronal artificial inspirada em processos biológicos, o qual necessita de menos recursos computacionais.

[2]designação utilizada para uma técnica que permite reduzir o overfit.

Menos recursos porque a análise pode ser providenciada sem recurso ao pré-processamento, ou mesmo utilizá-lo de forma diminuta, e, mesmo assim os resultados obtidos serem excelentes.

A convolução matemática, que consiste simplesmente numa transformação, é aqui utilizada; como à entrada da rede temos dados que depois vão ser transformados, originando dados à saída, estamos num processo convolucional [Tri]. Este, claro, é o conceito físico, pois o conceito matemático transporta-nos para um processo correlacional. Este conceito não é novo, pois em teoria dos sinais é amplamente utilizado. Uma outra característica, é a possibilidade de dados convoluídos serem deconvoluídos, e assim obter os dados da entrada no sistema. Pode ser um método de verificação ou validação do próprio sistema.

Para além deste modelo existem conceitos que por estarem presentes na construção deste tipo de redes, vão ser descritos já de seguida de forma resumida. Estes conceitos são:

- PCA - consiste num processo estatístico que transforma dados de variáveis correlacionadas em dados de variáveis linearmente não correlacionadas e designadas por componentes principais. Este é usado por exemplo para uma primeira análise de dados ou para a construção de modelos preditivos, baseando-se na análise da variância máxima de cada variável num dado sistema [Wikf]. A análise é feita através da identificação do componente principal dos dados.
- Flatten - esta técnica permite a redução dimensional dos dados, convertendo por exemplo tensores 4D em matrizes 2D.
- Dense - representa uma matriz constituída por uma multiplicação de vetores. Esta técnica é usada para alterar a dimensão dos vetores. Do ponto de vista matemático, corresponde à promoção de rotações, translações e alterações das escalas dos vetores.
- Dropout - consiste na remoção aleatória de vários neurónios da rede, baseando-se na distribuição de Bernoulli. A sua aplicação é feita durante o processo de treino e obriga a um redimensionamento dos neurónios restantes. Com esta técnica consegue-se resolver o problema do "Overfitting". Esta é usada como um processo de regularização em que se escolhem aleatoriamente algums membros da rede para serem excluídos, obrigando o sistema a que ele aprenda por si próprio com os recursos disponíveis [Sch].
- MaxPooling - divisão de um conjunto de dados em subconjuntos e cálculo do seu máximo [dee]. Esta técnica é usada pois permite trabalhar com invariâncias e reduz o tempo de computação.

Em AP, o cálculo tensorial está subjacente a todo o processo pois ele envolve, essencialmente matrizes.

Na base das análises seguintes esteve sempre presente a linguagem Python, pois esta é a mais utilizada neste domínio. Foi determinado o rigor médio de cada modelo e comparado com os modelos utilizados no estudo. Estes modelos têm as seguintes características:

- singleCNN - este modelo é composto pelas seguintes camadas:
  1. Convolução 2D com entrada de (1,28,28) com função de activação "ReLu"
  2. MaxPooling 2D de (2,2)
  3. Flatten
  4. Dense no número total de classes com função de activação "SoftMax"
- 2D CNN - este modelo é composto pelas seguintes camadas:
  1. Convolução 2D com entrada de (1,28,28) com função de activação "ReLu"

2. Convolução 2D com entrada de (32,3,3)
3. MaxPooling 2D de (2,2)
4. Flatten
5. Dense de 128 com função de activação "ReLu"
6. Dense no número total de classes com função de activação "SoftMax"

- 2D CNN + Dropout - este modelo é composto pelas seguintes camadas:

1. Convolução 2D com entrada de (1,28,28) com função de activação "ReLu"
2. Convolução 2D com entrada de (32,3,3)
3. MaxPooling 2D de (2,2)
4. Dropout de 0.25
5. Flatten
6. Dense de 128 com função de activação "ReLu"
7. Dropout de 0.50
8. Dense no número total de classes com função de activação "SoftMax"

Todo este código é igual em termos de estrutura para as três *framework* em análise, ou seja, Theano+Keras, TensorFlow+Keras e CNTK. Para além dos modelos tudo o restante é comum. O mesmo conjunto de dados do MNIST, que foram posteriormente convertidos para binário utilizando a mesma técnica. Também foi usada a mesma fórmula de cálculo da exatidão do modelo. Os dados originais do MNIST foram divididos em dados de treino (60000) e dados de teste (10000), e os dados de treino foram divididos em dados de treino e dados de validação. Para os resultados de comparação entre as *frameworks* foi configurada uma divisão num subconjunto dividindo dados de treino em 48000 e de validação em 12000, ou seja, utilizando uma divisão de vinte porcento. O número de classes considerado foi sempre 10, o valor para o "batch"foi 100 e para a escolha de dados aleatória foi usado um "seed"de 2017. Resumindo, a única variação no estudo foi a *framework*, e o modelo utilizado. Na tabela 4.2 estão descritas as versões utilizadas das bibliotecas.

Framework/Biblioteca	**Versão**
Theano	0.9.0dev3
TensorFlow	0.12.1
Keras	1.0.8
CNTK	2.0 beta12
MatPlotLib	1.3.1

Tabela 4.2: Versões utilizadas para comparação entre as três *frameworks* escolhidas.

O gráfico da figura 4.1 apresenta os resultados obtidos. Pode constatar-se que o modelo 2D CNN está em "overfitting" (claramente representado pelo máximo da curva a azul, mas também evidente na curva a vermelho). A exatidão do sistema é reduzida após aplicação de uma técnica usada para suavizar ou mesmo eliminar esse efeito, e, conhecida por *"Dropout"*.

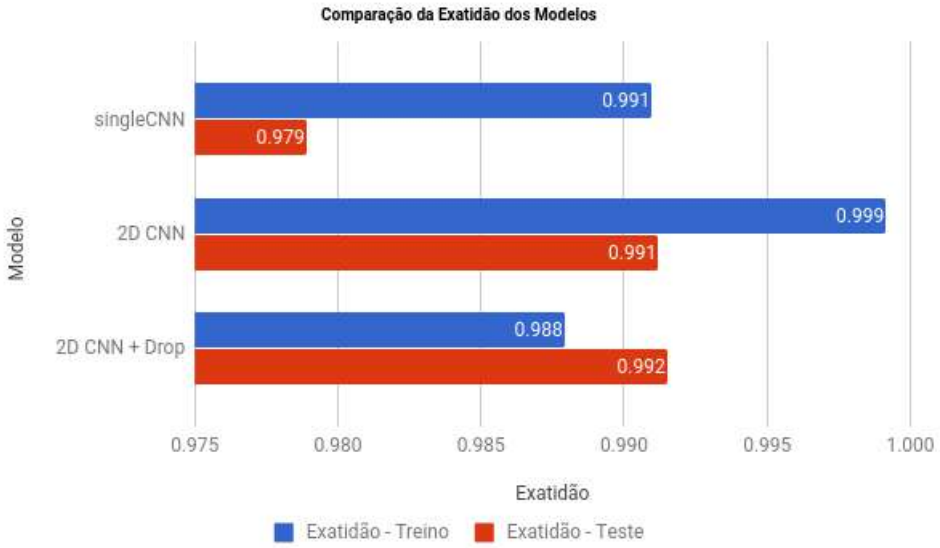

Figura 4.1: Representação da precisão de treino, e de teste, para cada modelo.

O gráfico da figura 4.2 torna claro que ao nível da exatidão quer o Keras+TF, quer o Keras+Th encontram-se ao mesmo nível. Porém, o CNTK não apresenta o mesmo tipo de desempenho, eventualmente por estar numa fase prematura do seu desenvolvimento. Note-se que ainda está na versão beta 12.

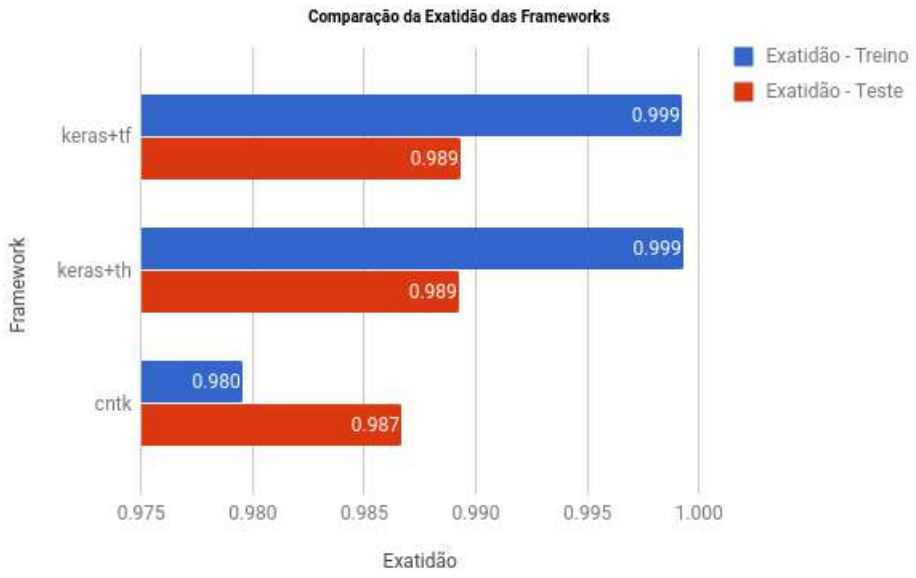

Figura 4.2: Representação da exatidão de treino, e de teste, de cada *framework* para o dataset MNIST.

O gráfico da figura 4.3 sumariza o erro cometido quando se compara a exatidão da fase de treino com a obtida durante a fase de teste, para as *frameworks* em estudo.

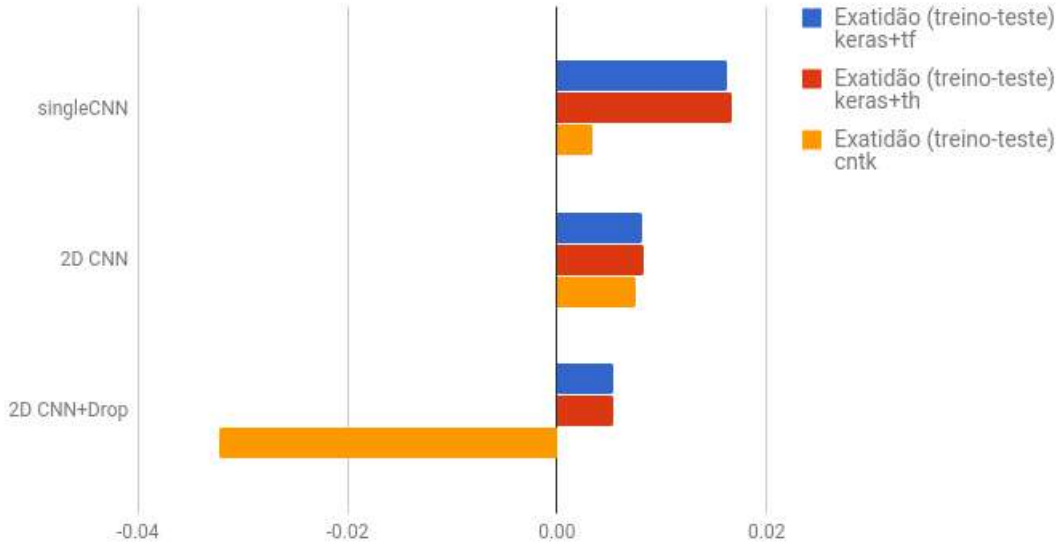

Figura 4.3: Diferença entre os valores de exatidão calculados durante a fase de treino e durante a fase de teste.

Torna-se claro que o erro é diminuto e ronda os 1.7% a 3.4% nos piores cenários. Como é o caso do CNTK com o modelo 2D CNN com Dropout.

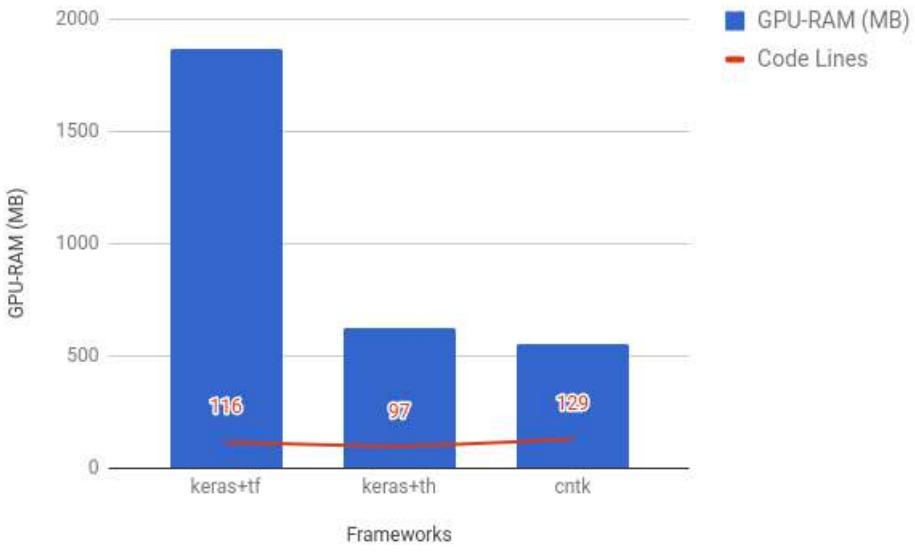

Figura 4.4: Número de linhas de código e RAM utilizada para cada *framework*.

Outro parâmetro considerado para análise foi a forma como cada uma das *frameworks* gere a RAM da placa gráfica e o número de linhas de código necessárias para implementar os modelos (ver gráfico da figura 4.4). Conclui-se que o número de linhas de código aproxima-se da centena para todas as *frameworks*. No entanto, é evidente que o tensorflow necessita de grandes recursos de hardware, nomeadamente a RAM da GPU. Para tornar o modelo exequível, foi essencial introduzir código que limitasse a quantidade de memória que podia ser utilizada, pois caso contrário não seria possível executar o código respetivo. De

facto, constatou-se que o tensorflow, face ao theano para além de ser mais lento não está tão otimizado em termos de processamento. Digamos que precisa de ajustes na sua configuração através da implementação.

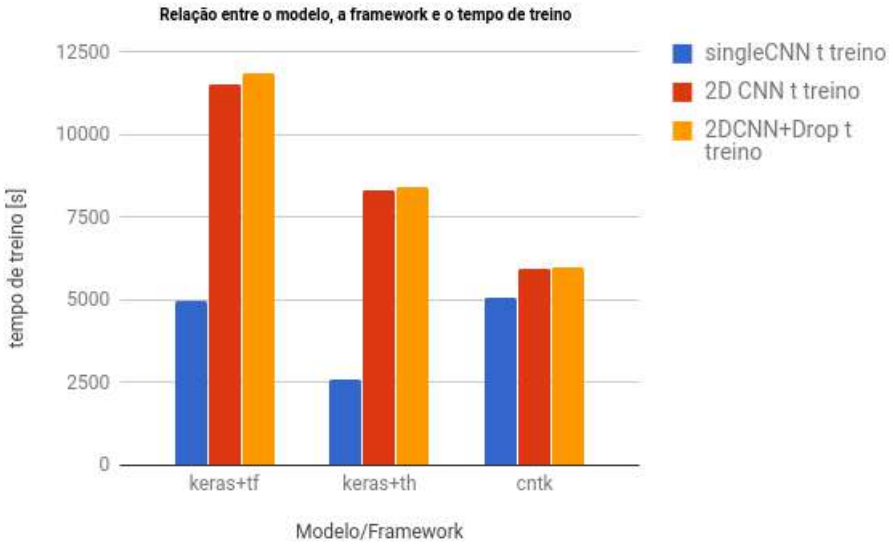

Figura 4.5: Tempo necessário para treinar o sistema de acordo com a *framework* e o modelo.

A relação entre a velocidade de processamento do código para cada um dos modelos está representada na figura 4.5. Constata-se que o Keras+TF demora mais tempo na fase de treino do que os outros dois. Estimando-se, no pior cenário, 50% de tempo a mais do que o CNTK e 25% a mais do que o Keras+Th.

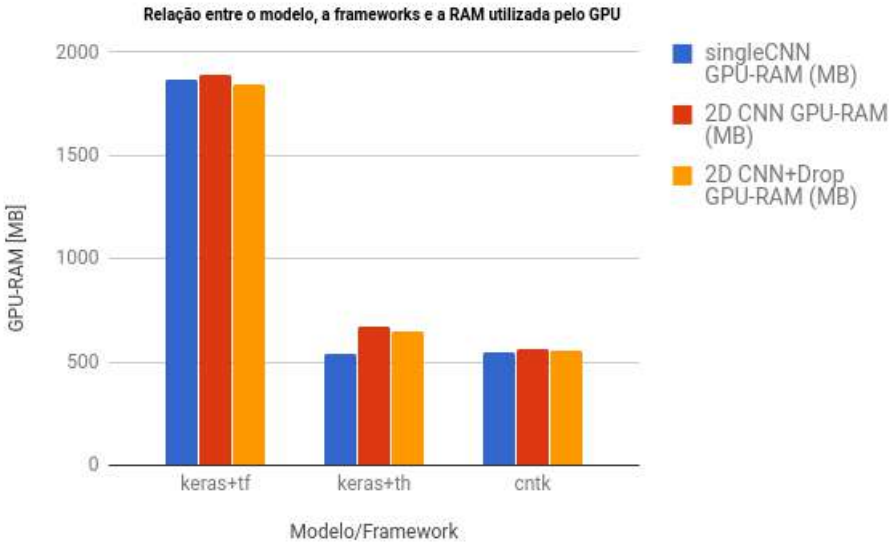

Figura 4.6: Quantidade de RAM da GPU dispendida por modelo e *framework*.

O gráfico da figura 4.6 torna claro os recursos que o tensorflow utiliza face aos do theano e do CNTK, mesmo com a limitação estabelecida por código do valor máximo a ser usado pelo sistema. Chega a consumir cerca de três vezes mais RAM do que os concorrentes em estudo.

Um outro estudo que ainda foi realizado com estes dados foi a forma como a dimensão dos dados de treino poderia ou não influenciar os resultados. Assim, para o modelo mais eficiente, i.e., 2D + CNN + Drop, e usando o Keras + Theano, procedeu-se da seguinte forma: executou-se o método para três cenários distintos, em que a única diferença entre estes cenários seria a dimensão dos dados de treino, e, consequentemente dos de validação. Foram consideradas três dimensões para os dados de treino, 20%, 50% e 80% das 60000 entradas.

Depois de realizados estes ensaios, obtiveram-se os seguintes resultados (ver gráfico da figura 4.7):

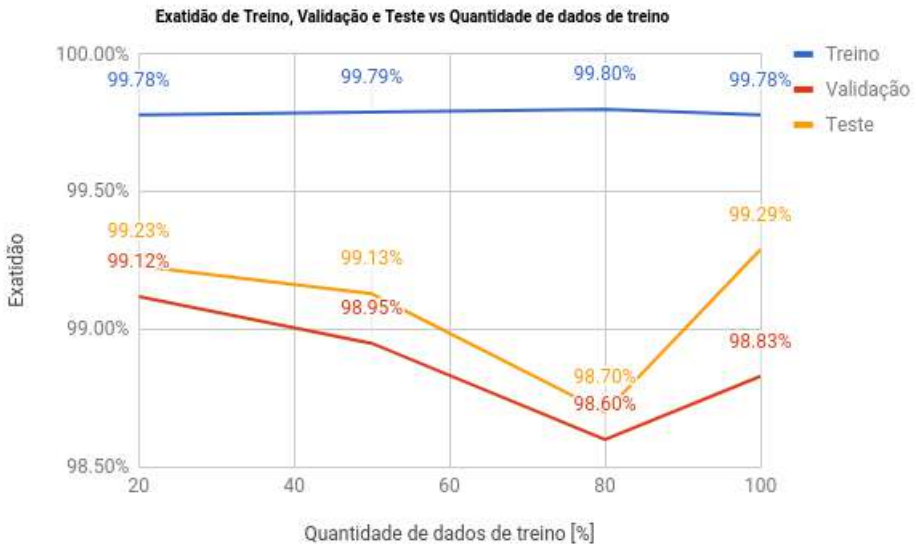

Figura 4.7: Representação gráfica dos resultados do estudo da variação da dimensão da percentagem de dados de treino e de validação usados para as entradas do modelo 2D+CNN+Drop.

Os erros obtidos nesta experiência encontram-se representados da tabela 4.3.

	Exatidão [3] [%]			Erro [%]		
%	**Treino**	**Validação**	**Teste**	**Treino**	**Validação**	**Teste**
20	99.78	99.12	99.23	0.63	5.74	5.33
50	99.79	98.95	99.13	0.52	6.85	5.63
80	99.80	98.60	98.70	0.61	8.47	7.66
100	99.78	98.83	98.83	0.61	9.37	3.86

Tabela 4.3: Resultados obtidos no estudo da variação da dimensão da percentagem de dados de treino e de validação usados para as entradas do modelo 2D+CNN+Drop.

De acordo com os dados apresentados, o modelo tem um bom comportamento. Tal facto é corroborado no grau de exatidão obtido, quer durante a fase de validação, quer durante a fase de teste, apresentando erros compreendidos entre os 4% e os 8%. É claro também que a percentagem de dados de treino utilizados, mesmo que diminuta, não altera a boa prestação do modelo utilizado.

4.2.2 DeepLDA

Para o conjunto de dados semi-supervisionados utilizou-se um outro dataset, o STL-10 e comparou-se com os dados obtidos com o MNIST, utilizando o modelo DeepLDA. Este método apresentado por Matthias Dorfer e outros, pretende mostrar a importância do modelo ao ser aplicado em redes neuronais e com maior sucesso do que o clássico LDA. O LDA consiste num modelo Bayesiano de três níveis de inferência com grande sucesso. A ideia dos autores foi usar como ponto de partida o LDA e as redes neuronais profundas, integrando-os por forma a conseguir extrair as características[4] dos dados.

Os recursos computacionais utilizados foram os referidos no ponto 4.1 do capítulo 4, e, os requisitos foram os seguintes (ver tabela 4.4):

Framework/Biblioteca	**Versão**
Theano	0.9.0.dev0
Lasagne	0.1
Numpy	1.11.0
SciPy	0.17.0
MatPlotLib	1.3.1

Tabela 4.4: Versões das bibliotecas usadas nos ensaios de dados semi-supervisionados com a aplicação do método DeepLDA.

Matthias Dorfer, Rainer Kelz e Gerhard Widmer do Departamento de Percepção Computacional da JKU Linz, publicaram um artigo em 2016, aplicando um método designado por análise linear discriminante profunda[5] ao conjunto de dados semi-supervisionados STL-10.

Mas em que consiste este modelo? Como funciona?

O LDA é um método que provêm da análise estatística multi-variada, que procura encontrar uma forma de reduzir a dimensão do sistema.

Em termos, organizacionais, os dados de entrada são introduzidos numa rede neuronal profunda e depois aplica-se o método LDA à saída desses dados da rede.

Como na AP o cálculo é vetorial, é perfeitamente natural que o conceito de base e de valor próprio esteja sempre presente. E este método analisa os valores próprios, com o objetivo de os maximizar.

Aplicando o método DeepLDA foram usadas 1000 imagens (3,96,96) para treino, 4000 para validação e 8000 para teste. O valor de algumas variáveis foram fixados em valores otimizados e determinados de acordo com as experiências realizadas para este método, consistindo em: batch=150, lr=0.1, momentum=0.9, epochs=1000. Utilizou-se a versão mais simples de decréscimo da lr para metade a cada 100 epochs. Depois executou-se um código [MD] com vista ao treino do sistema e posteriormente executou-se outro código para avaliar os resultados do treino e obtiveram-se os seguintes resultados (ver tabela 4.5):

[4] em inglês, features.
[5] em inglês, Deep Linear Discriminant Analysis, de acrónimo DeepLDA.

	MNIST	**STL-10**
Tempo de Execução [dias]	0.19	2.93
Exatidão (Treino) [%]	99.98	100.00
Exatidão (Teste) [%]	99.65	65.50
Erro (Teste) [%]	62.00	4.22

Tabela 4.5: Resultados obtidos da aplicação do método DeepLDA.

4.2.3 AE e GAN

Nos estudos relacionados com a utilização de dados não supervisionados recorre-se ao Keras e ao Theano em backend porque pelos resultados obtidos para as outras *frameworks*, não parecem ser a melhor escolha. O TensorFlow pelos recursos que necessita e o CNTK pelo estado embrionário em que se encontra.

Das várias abordagens (dentro da AP) existentes vão ser escolhidas apenas duas. Os métodos designados por "AutoEncoder" e "GAN"; o primeiro por ser uma abordagem considerada mais clássica e o segundo por ser mais recente. Apesar de tudo o que diz respeito aos desenvolvimentos na área da AP serem extremamente recentes.

Na tabela 4.6 encontram-se as versões das bibliotecas utilizadas para a implementação dos dois métodos.

GAN

Biblioteca	**Versão**
Theano	0.9.0dev3.dev-RELEASE
Keras	1.2.2
Python	2.7.12
Seaborn	0.7.1
Numpy	1.12.0
cPickle	1.71
MatPlotLib	1.5.3
tqdm	4.11.2

AE

Biblioteca	**Versão**
Theano	0.9.0dev3.dev-RELEASE
Keras	1.2.2
Python	2.7.12
Numpy	1.12.0
MatPlotLib	1.5.3
Scipy	0.18.1

Tabela 4.6: Versões das bibliotecas utilizadas para aplicação dos métodos GAN e AutoEncoder.

Dos vários métodos existentes para trabalhar com este tipo de dados, foram estes dois os selecionados porque o AutoEncoder permite pegar nos dados codificá-los e descodificá-los (ver figura 4.8).

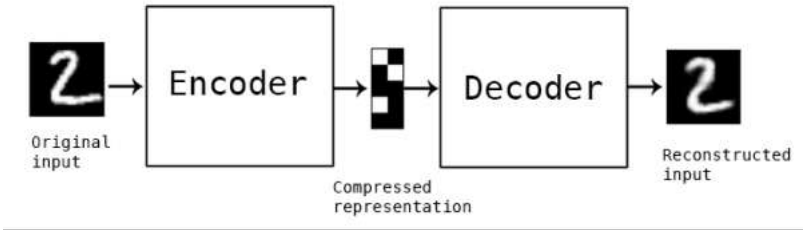

Figura 4.8: Representação esquemática do algoritmo AutoEncoder [Cho].

O método GAN é considerado um método importantíssimo nesta área pelas potencialidades que representa (ver figura 4.9).

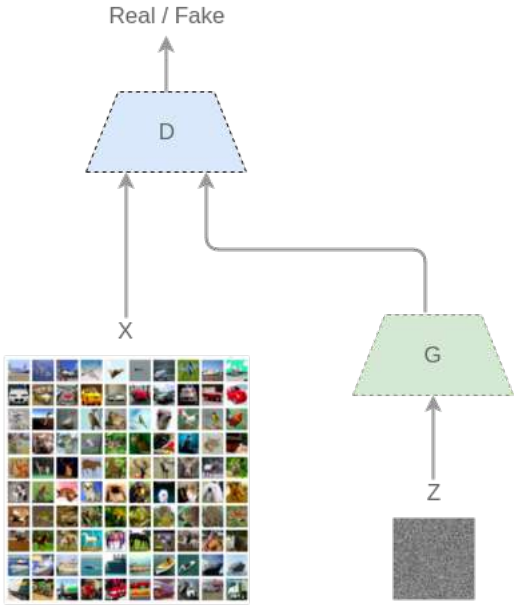

Figura 4.9: Representação esquemática do método GAN [Sot].

Este método utiliza duas peças na sua arquitetura, um gerador (G) e um discriminador (D). O sistema recebe como entradas um conjunto de dados, o gerador gera saídas e por sua vez o discriminador decide se estas provêm da mesma distribuição do conjunto de dados de treino. Temos aqui duas linhas paralelas de aprendizagem, uma baseada em imagens reais e outra em imagens fictícias geradas pelo gerador a partir de algum ruído. O discriminador compara a informação de ambas as linhas de aprendizagem e toma uma decisão. O resultado, embora curioso, podem ser objetos irreais, no caso de lidarmos com conjuntos de dados de imagens. A filosofia subjacente a este tipo de abordagem reside no princípio de que aquilo que não conseguimos criar, não conseguimos entender, parafraseando Richard Feynman ("What I cannot create, I do not understand") [Gooa]. Por isso, mesmo os métodos usados atualmente para dados não supervisionados propõem-se primeiro em criar conjuntos de dados próximos daqueles que se pretendem entender. O sistema está a ser ensinado a aprender, e quando se chega a um bom modelo, conseguem-se fazer previsões exatas. As duas áreas principais de desenvolvimento, focam-se em sistemas de Auto Codificação (já utilizados por alguns autores em dados semi-supervisionados) com várias variantes (como por exemplo, Deep AutoEncoders, VAE, entre outros) e sistemas GAN também com diversas técnicas (como por exemplo, InfoGAN, DCGAN, entre outros).

Nesta componente experimental usaram-se sempre as mesmas configurações variando unicamente o modelo de aplicação. Neste âmbito, consideraram-se os seguintes modelos para análise:

- GAN - é um modelo que gera dados e os compara com os dados reais. A avaliação da eficiência do mesmo é através do cálculo do erro, que consiste na diferença entre o gerado e o real.

- DCGAN - é equivalente ao anterior mas utiliza convoluções e deconvoluções não numa perspetiva de rede neuronal como o anterior mas numa de rede neuronal profunda. Assemelha-se muito à integração de um CNN com o GAN.

- AE - é um modelo codificador / descodificador de dados aplicado âs RN.

- DAE - é equivalente ao anterior mas numa perspetiva de rede neuronal profunda.

- CNN - já conhecido pelos excelentes resultados quando trabalha com dados supervisionados.

Os resultados experimentais foram obtidos mediante as seguintes configurações, sistematizadas na tabela 4.7.

Parâmetro	**Valor**
número de épocas	100
número de iterações / época	600
aleatóriedade	2017
lote (em inglês, batch)	100
forma	(1,28,28)
normalização dos dados	sim
dimensão dos dados de treino	60000
conjunto de dados utilizado	MNIST
taxa de aprendizagem	variável
momento	0.975
número de classes	10

Tabela 4.7: Configurações experimentais comuns aos modelos analisados.

Foram analisados os tempos de execução (que não vão ser apresentados pois não foram feitos os testes de forma normalizada, devido ao processamento simultâneo executado), os erros e os modelos (que podem ter um número diferente de parâmetros de aprendizagem, dependendo da sua estrutura).

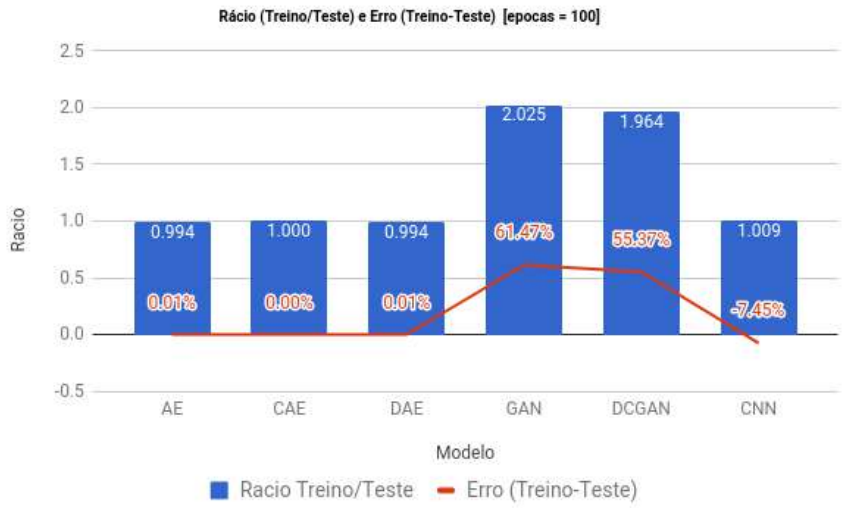

Figura 4.10: Comparação entre os diversos métodos utilizados para o conjunto MNIST não supervisionado.

Da comparação entre os diversos métodos, perceciona-se que as metodologias AE e CNN apresentam valores muito semelhantes e que a metodologia GAN obtém valores piores. Mas, os dados do método CNN são resultantes de dados supervisionados. O que significa que o modelo AE está ao mesmo nível do CNN, mas para dados não supervisionados. Isto implica claramente um grande progresso neste domínio.

O método GAN utiliza uma abordagem distinta dos anteriores, pois procura criar algarismos semelhantes aos que existem no MNIST. As figuras 4.11, 4.12, 4.13, 4.14 e 4.15, são elucidativas do tipo de evolução que os modelos GAN conseguiram atingir durante o processo de geração de algarismos.

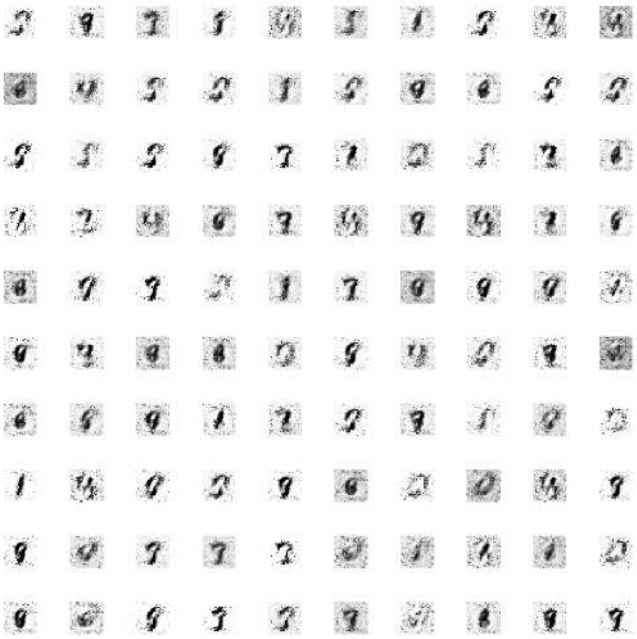

Figura 4.11: Sistema de autogeração de dígitos MNIST não supervisionado: modelo GAN, 1ª época.

Figura 4.12: Sistema de autogeração de dígitos MNIST não supervisionado: modelo GAN, 100ª época.

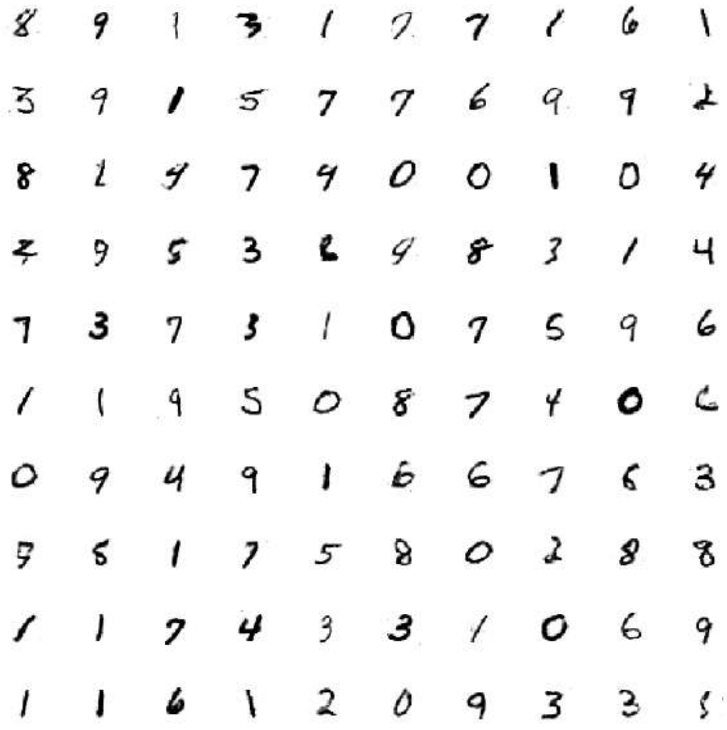

Figura 4.13: Sistema de autogeração de dígitos MNIST não supervisionado: modelo GAN, 1000ª época.

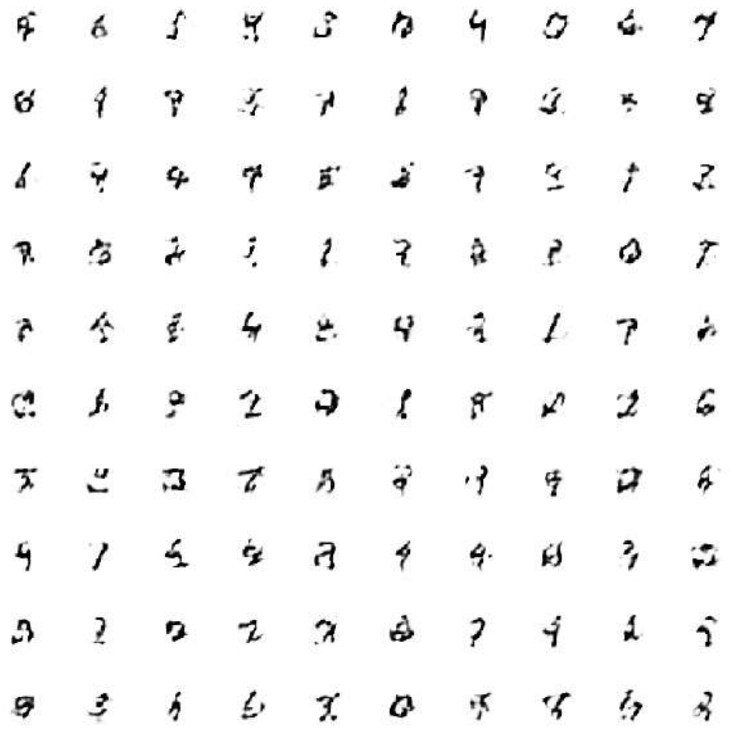

Figura 4.14: Sistema de autogeração de dígitos MNIST não supervisionado: modelo DGAN, 1ª época.

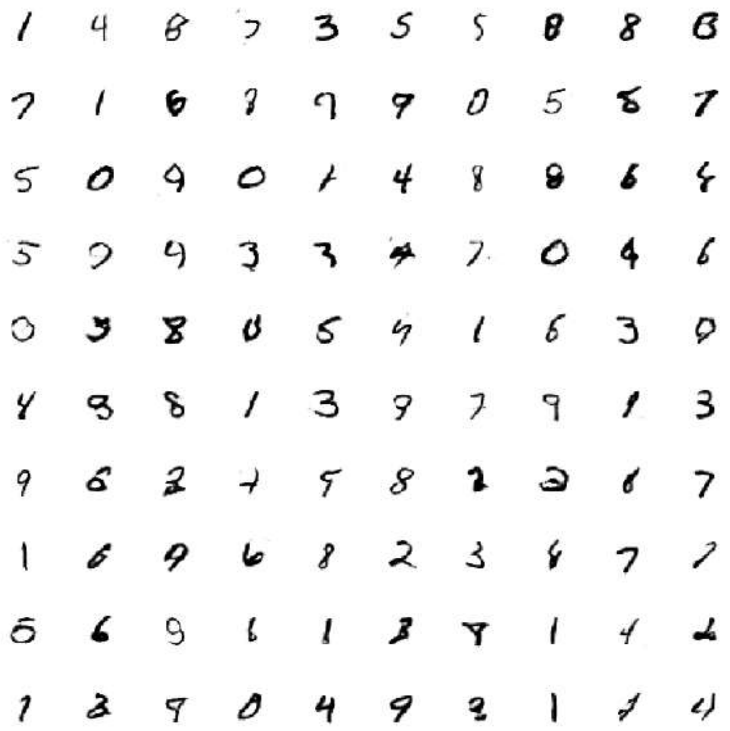

Figura 4.15: Sistema de autogeração de dígitos MNIST não supervisionado: modelo DGAN, 100ª época.

Teoricamente o comportamento das curvas de aprendizagem e de exatidão devem aproximar-se das funções representadas nos gráficos das figuras 4.16 e 4.17.

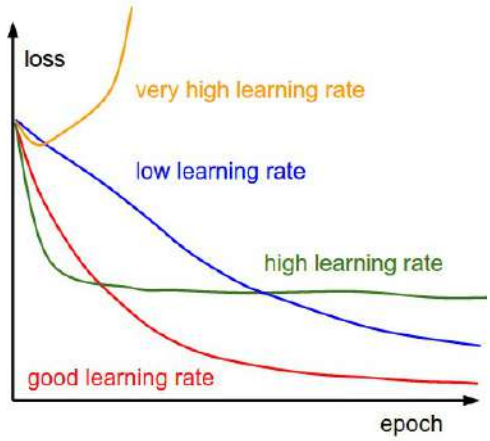

Figura 4.16: Curvas de aprendizagem [Anda].

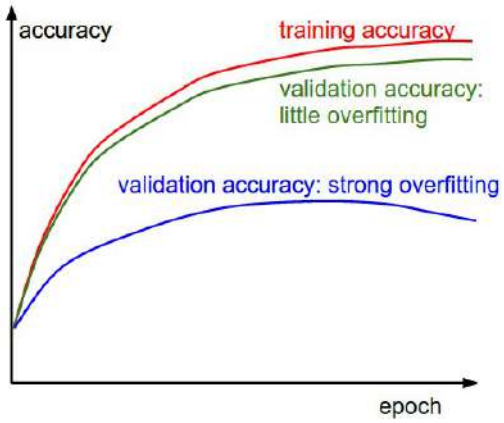

Figura 4.17: Curvas de exatidão [Anda].

As diferença entre a exatidão da fase de treino e da fase de validação permite estimar o grau de "overfitting".

Se juntarmos os dois gráficos, e, a diferença entre ambos os tipos de curvas for nula, estaremos na presença de um sistema de aprendizagem automática perfeita.

As curvas de aprendizagem destes modelos apresentam também bons resultados, vejamos as figuras 4.18 e 4.19.

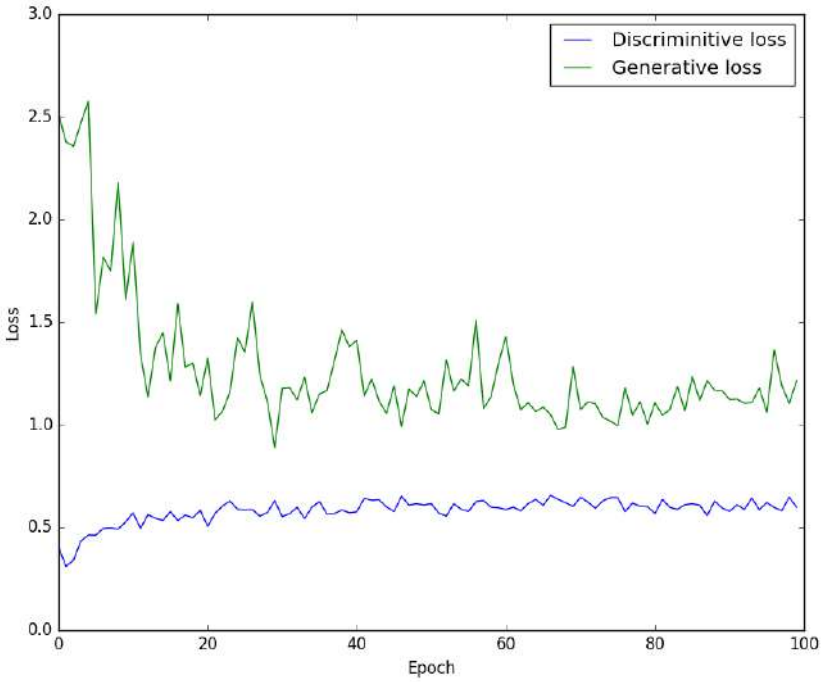

Figura 4.18: Sistema de autogeração de dígitos MNIST não supervisionado para as primeiras 100 épocas: modelo GAN, função perda.

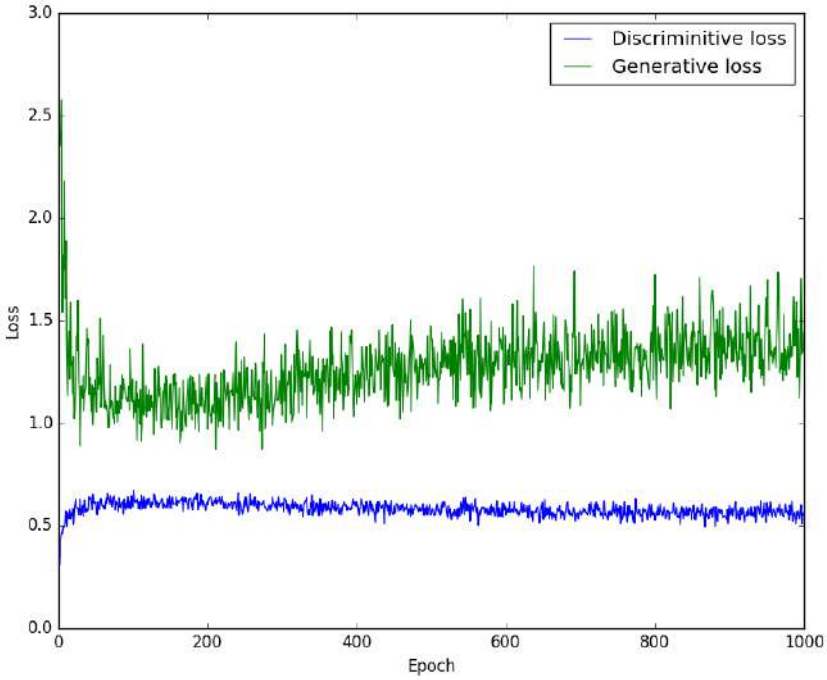

Figura 4.19: Sistema de autogeração de dígitos MNIST não supervisionado para as primeiras 1000 épocas: modelo GAN, função perda.

Como a interpretação e comparação de imagens torna-se difícil sem recorrer a uma quantificação, optou-se por incluir três parâmetros que permitem quantificar essa comparação (para o caso dos modelos GAN). A diferença percentual entre elas, o MSE[6] e o SSIM[7].

Foram implementados três algoritmos para quantificar esta comparação de acordo com o seguinte:

- Diferença - Converteram-se as imagens para uma escala de cinzentos e aplicou-se a fórmula

$$dif = \frac{|p1 - p2|}{255}.100$$

  onde $p1$ e $p2$ são pontos da matriz que compõe cada uma das imagens. Para controlo da qualidade do algoritmo, utilizou-se uma imagem totalmente branca e uma totalmente preta e obteve-se uma diferença de 100

- MSE - Converteram-se as imagens para uma escala de cinzentos e aplicou-se a fórmula

$$MSE = \frac{1}{m.n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2$$

  onde quando $MSE = 0$ a semelhança entre as imagens é perfeita.

- SSIM - Converteram-se as imagens para uma escala de cinzentos e aplicou-se a fórmula

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

  onde quando $SSIM = 1$ a semelhança entre as imagens é perfeita. Este método desenvolvido por [eo04] permite colmatar os erros provenientes da análise realizada com o MSE e os seus valores situam-se no intervalo $[-1; 1]$.

Com os algoritmos desenvolvidos, em Python, obtiveram-se os resultados apresentados na tabela 4.8.

epocas	MSE	SSIM	Diferença (%)
200	2390.68	0.86	4.63
400	2492.20	0.87	4.75
600	2302.61	0.87	4.43
800	2436.82	0.87	4.67
1000	2414.47	0.87	4.61

Tabela 4.8: Resultados da comparação entre as imagens geradas pelos métodos DCGAN e GAN em função do número de epocas.

Dos três métodos utilizados para comparar as imagens, o $SSIM$ e o da dif são os que melhor permitem percecionar as diferenças entre elas. Os cerca de 5 % de erro obtidos pelo algoritmo dif ou os 7 % de erro obtidos pelo algoritmo $SSIM$ permitem concluir que as imagens geradas por ambos os métodos estão muito próximas.

[6] em inglês, Mean Squared Error.
[7] em inglês, Structural Similarity Index.

5

Conclusões e Trabalho Futuro

5.1 Conclusões

As competições existentes sobre o tema da AA e da AP constituem uma ignição para o desenvolvimento nesta área. No github, Rodrigo Benenson, compilou a informação a esse respeito [Bena].

Comparando os resultados obtidos nesta tese com os disponíveis publicamente, constata-se que os resultados obtidos com os métodos utilizados estão dentro dos valores reportados por outros autores.

A abordagem exploratória deste tema permitiu perceber que o TensorFlow é mais adequado para sistemas com uma grande infraestrutura computacional que permita processamento paralelo em CPU's e GPU's. No entanto, o Theano torna-se mais versátil e menos exigente no que diz respeito a recursos, obtendo os mesmos resultados. Relativamente ao Microsoft CNTK, que ainda se encontra na versão beta, já apresenta um bom desempenho, mas é preciso a versão final para tirar mais conclusões. Neste momento, é uma *framework* promissora, pois tendo sido criada por volta de 2014, (e, não tendo grande evolução em 2014 e 2015), em 2016 mostrou-se muito dinâmica do ponto de vista de desenvolvimento e de integração de novos modelos de AP. Principalmente pelo *boom* da AP em 2016.

Das várias formas de aprendizagem, aquela que trabalha com dados não supervisionados, será a que terá que crescer mais nos próximos tempos. O seu desenvolvimento ainda se encontra no início, mas os métodos GAN já começam por apresentar resultados razoáveis, como se provou nesta tese.

Também é notório que a linguagem comum na Investigação e Desenvolvimento, nesta área, é sem dúvida o Python, que se tornou nos últimos dois anos a linguagem mais utilizada a nível mundial, cobrindo várias áreas, e, superando a linguagem JAVA.

Conclui-se ainda que como os recursos computacionais existentes são altamente escaláveis, o processo de cálculo vê-se extremamente facilitado, significando com isso uma redução de custos e otimizando a forma de trabalhar, seja pela utilização de *containers* no Docker, no Linux ou mesmo através dos ambientes que podem ser gerados no Python. Claro que existe todo um trabalho de integração e de configuração do aparato experimental que pode ser moroso, quer pelas dificuldades encontradas, quer pela complexidade da implementação e ajuste dos modelos utilizados em AP, ou mesmo, em AA.

Finalmente, levando em conta os resultados apresentados na tabela 4.3., parece conclusivo que a dimensão dos dados de treino não altera a boa prestação do modelo 2D+CNN+Drop.

5.2 Trabalho Futuro

A AP permite incrementar quer o nível de abstração, quer o nível de complexidade da aprendizagem. Os limites existentes no passado, com a falta de capacidade computacional, parecem ultrapassados. Com o emergir da computação quântica e com os desenvolvimentos na área da opto-electrónica pode vislumbrar-se o nascer de comunicações ópticas entre diversos dispositivos eletrónicos, que darão um novo fôlego à AP e às tecnologias que podem nascer a partir da sua existência [RLZG16]. O modelo tradicional em que a AP se baseia e que consiste em treinar um sistema pode ser alterado num futuro próximo pois, de acordo com Michael Andersch (Senior GPU architect at NVIDIA), o processo de inferência pode acelerar todo o processamento. A figura 5.1, que esquematiza e compara ambos os modelos.

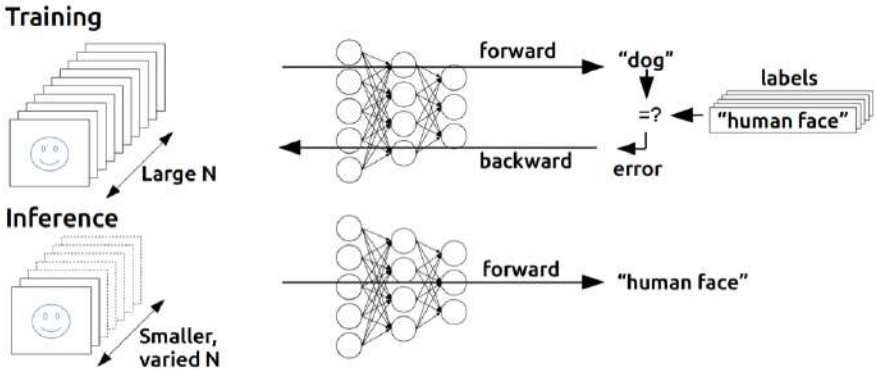

Figura 5.1: Comparação entre o processo de treino usada em AP e o processo de inferência. No treino, muitas entradas divididas em "batches" (grandes grupos), são usadas para treinar a rede neuronal profunda. Na inferência, a rede de treino é usada para descobrir a informação de novas entradas na rede em pequenos grupos [Andb].

Mas os novos desenvolvimentos, não ficaram simplesmente por aqui. Em 2016, na utilização de dados não supervisionados, foi-se além das abordagens clássicas. Começou-se por utilizar AE, recorreu-se ao MLP

[1], onde a entrada e a saída têm a mesma dimensão e a camada escondida é treinada para recuperar a entrada. No entanto, novos métodos foram desenvolvidos, como é o caso do GAN. Yann LeCun, o pai da AP, chegou a afirmar que o GAN é a ideia mais importante na AA nos últimos 20 anos [Sot].

No futuro, a inteligência artificial vai acabar por ser integrada na computação quântica. O trabalho de Alvarez-Rodriguez [UAR16] pode ser precisamente a ignição que despoletará a interligação das ciências físicas e biológicas com a engenharia informática.

Também o facto do físico Hoi-Kwan Lau, e outros, estarem a trabalhar numa máquina de aprendizagem automática quântica com dimensões infinitas pode levar a um alargamento dos horizontes [HKLW, LPSW17].

Num contexto diferente, ao nível de uma tese de doutoramento, teria todo o interesse em explorar mais profundamente esta matéria, com vista à construção de um sistema eficiente e transversal a outras áreas do conhecimento. A implementação de métodos de AP em reconhecimento da linguagem natural, deteção de objetos em vídeo, reconhecimento facial ou outros, seriam um grande passo evolutivo para a Inteligência Artificial.

Do ponto de vista, mais pessoal, projetos futuros decorrentes desta tese, seriam:

- Implementação de um dos modelos mais simples de AP, com possível escalonamento para outros, num computador quântico e/ou ótico;
- Aprofundar o estudo da AP, com vista ao desenvolvimento de novos modelos a serem aplicados num contexto de aprendizagem não supervisionada e eventualmente por reforço;
- Integrar modelos de AP em outros sistemas, como por exemplo: Outsystems, Salesforce, sistemas cloud, etc;
- Integrar dados da IoT [2] nos modelos de AP, podendo ter como aplicações práticas mais imediatas, sistemas de Realidade Aumentada;
- Explorar outras áreas aonde a AP pode ser aplicada, como por exemplo o processamento da linguagem natural.

Claro que estes projetos futuros teriam bastante interesse se fossem implementados num contexto de investigação e desenvolvimento ligado ao privado, e, visando uma grau académico superior, ou seja, um doutoramento bi-partido entre o público e o privado.

A um nível, menos académico, seria interessante:

- Utilizar o conhecimento adquirido para participar em competições existentes na área com vista a testar e otimizar os modelos apresentados e eventualmente explorar outro tipo de modelos;
- Ingressar num projeto que englobam três áreas que se ligam e complementam: *Data Science*, *Big Data* e Sistemas de Alta Disponibilidade.

[1] em inglês, multilayer perceptron
[2] em inglês, Internet of Things.

A

Código

Aqui encontra-se parte do código em python utilizado nesta tese.

A.1 singleCNN

As frameworks usadas para a implementação foram Theano + Keras.

```
# src: http://parneetk.github.io/blog/cnn-mnist/
# If using tensorflow, set image dimensions order
from keras import backend as K
if K.backend()=='tensorflow':
    K.set_image_dim_ordering("tf")
import time
import matplotlib.pyplot as plt
import numpy as np
from keras.utils import np_utils
```

```
from keras.models import Sequential
from keras.layers.convolutional import Convolution2D, MaxPooling2D
from keras.layers import Activation, Flatten, Dense, Dropout
from keras.optimizers import SGD
from keras.layers.normalization import BatchNormalization
#% matplotlib inline
np.random.seed(2017)
# Load MNIST Dataset
from keras.datasets import mnist
(train_features, train_labels), (test_features, test_labels) = mnist.load_data()
_, img_rows, img_cols =  train_features.shape
num_classes = len(np.unique(train_labels))
num_input_nodes = img_rows*img_cols
print "Number of training samples: %d"%train_features.shape[0]
print "Number of test samples: %d"%test_features.shape[0]
print "Image rows: %d"%train_features.shape[1]
print "Image columns: %d"%train_features.shape[2]
print "Number of classes: %d"%num_classes
# Pre-processing
train_features = train_features.reshape(train_features.shape[0], 1, img_rows, img_
test_features = test_features.reshape(test_features.shape[0], 1, img_rows, img_col
train_features /= 255
test_features /= 255
print('train_features shape:', train_features.shape)
print(train_features.shape[0], 'train samples')
print(test_features.shape[0], 'test samples')

# convert class labels to binary class labels
train_labels = np_utils.to_categorical(train_labels, num_classes)
test_labels = np_utils.to_categorical(test_labels, num_classes)
# Function to plot model accuracy and loss
def plot_model_history(model_history):
    fig, axs = plt.subplots(1,2,figsize=(15,5))
    # summarize history for accuracy
    axs[0].plot(range(1,len(model_history.history['acc'])+1),model_history.history
    axs[0].plot(range(1,len(model_history.history['val_acc'])+1),model_history.hist
    axs[0].set_title('Model Accuracy')
    axs[0].set_ylabel('Accuracy')
    axs[0].set_xlabel('Epoch')
    axs[0].set_xticks(np.arange(1,len(model_history.history['acc'])+1),len(model_h
    axs[0].legend(['train', 'val'], loc='best')
    # summarize history for loss
    axs[1].plot(range(1,len(model_history.history['loss'])+1),model_history.history
    axs[1].plot(range(1,len(model_history.history['val_loss'])+1),model_history.his
    axs[1].set_title('Model Loss')
    axs[1].set_ylabel('Loss')
    axs[1].set_xlabel('Epoch')
    axs[1].set_xticks(np.arange(1,len(model_history.history['loss'])+1),len(model_
    axs[1].legend(['train', 'val'], loc='best')
```

```
    plt.show()
# Funtion to compute test accuracy
def accuracy(test_x, test_y, model):
    result = model.predict(test_x)
    predicted_class = np.argmax(result, axis=1)
    true_class = np.argmax(test_y, axis=1)
    num_correct = np.sum(predicted_class == true_class)
    accuracy = float(num_correct)/result.shape[0]
    return (accuracy * 100)

# A Simple CNN
# Define the model
model1 = Sequential()
model1.add(Convolution2D(32, 3, 3, border_mode='valid', input_shape=(1, 28, 28)))
model1.add(Activation("relu"))
model1.add(MaxPooling2D(pool_size=(2, 2)))
model1.add(Flatten())
model1.add(Dense(num_classes))
model1.add(Activation("softmax"))
# Compile the model
model1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accurac
# Train the model
start = time.time()
model1_info = model1.fit(train_features, train_labels, batch_size=100, \
                         nb_epoch=100, verbose=1, validation_split=0.2)
end = time.time()
# plot model history
plot_model_history(model1_info)

print "Model took %0.2f seconds to train"%(end - start)
# compute test accuracy
print "Accuracy on test data is: %0.2f"%accuracy(test_features, test_labels, model1
score = model1.evaluate(test_features, test_labels, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

A.2 2D CNN

As frameworks usadas para a implementação foram Theano + Keras.

```
# src: http://parneetk.github.io/blog/cnn-mnist/
# If using tensorflow, set image dimensions order
from keras import backend as K
if K.backend()=='tensorflow':
    K.set_image_dim_ordering("tf")
import time
import matplotlib.pyplot as plt
import numpy as np
from keras.utils import np_utils
```

```
from keras.models import Sequential
from keras.layers.convolutional import Convolution2D, MaxPooling2D
from keras.layers import Activation, Flatten, Dense, Dropout
from keras.optimizers import SGD
from keras.layers.normalization import BatchNormalization
#% matplotlib inline
np.random.seed(2017)
# Load MNIST Dataset
from keras.datasets import mnist
(train_features, train_labels), (test_features, test_labels) = mnist.load_data()
_, img_rows, img_cols =  train_features.shape
num_classes = len(np.unique(train_labels))
num_input_nodes = img_rows*img_cols
print "Number of training samples: %d"%train_features.shape[0]
print "Number of test samples: %d"%test_features.shape[0]
print "Image rows: %d"%train_features.shape[1]
print "Image columns: %d"%train_features.shape[2]
print "Number of classes: %d"%num_classes
# Pre-processing
train_features = train_features.reshape(train_features.shape[0], 1, img_rows, img_
test_features = test_features.reshape(test_features.shape[0], 1, img_rows, img_col
train_features /= 255
test_features /= 255
print('train_features shape:', train_features.shape)
print(train_features.shape[0], 'train samples')
print(test_features.shape[0], 'test samples')

# convert class labels to binary class labels
train_labels = np_utils.to_categorical(train_labels, num_classes)
test_labels = np_utils.to_categorical(test_labels, num_classes)
# Function to plot model accuracy and loss
def plot_model_history(model_history):
    fig, axs = plt.subplots(1,2,figsize=(15,5))
    # summarize history for accuracy
    axs[0].plot(range(1,len(model_history.history['acc'])+1),model_history.history
    axs[0].plot(range(1,len(model_history.history['val_acc'])+1),model_history.hist
    axs[0].set_title('Model Accuracy')
    axs[0].set_ylabel('Accuracy')
    axs[0].set_xlabel('Epoch')
    axs[0].set_xticks(np.arange(1,len(model_history.history['acc'])+1),len(model_h
    axs[0].legend(['train', 'val'], loc='best')
    # summarize history for loss
    axs[1].plot(range(1,len(model_history.history['loss'])+1),model_history.history
    axs[1].plot(range(1,len(model_history.history['val_loss'])+1),model_history.his
    axs[1].set_title('Model Loss')
    axs[1].set_ylabel('Loss')
    axs[1].set_xlabel('Epoch')
    axs[1].set_xticks(np.arange(1,len(model_history.history['loss'])+1),len(model_
    axs[1].legend(['train', 'val'], loc='best')
```

```
    plt.show()
# Funtion to compute test accuracy
def accuracy(test_x, test_y, model):
    result = model.predict(test_x)
    predicted_class = np.argmax(result, axis=1)
    true_class = np.argmax(test_y, axis=1)
    num_correct = np.sum(predicted_class == true_class)
    accuracy = float(num_correct)/result.shape[0]
    return (accuracy * 100)

# A 2D CNN
# Define the model
model3 = Sequential()
model3.add(Convolution2D(32, 3, 3, border_mode='valid', input_shape=(1, 28, 28)))
model3.add(Activation("relu"))
model3.add(Convolution2D(32, 3, 3,  border_mode='valid'))
model3.add(Activation("relu"))
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(Flatten())
model3.add(Dense(128))
model3.add(Activation("relu"))
model3.add(Dense(num_classes))
model3.add(Activation("softmax"))
# Compile the model
model3.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accurac
# Train the model
start = time.time()
model3_info = model3.fit(train_features, train_labels, batch_size=100, \
                         nb_epoch=100, verbose=1, validation_split=0.2)
end = time.time()
# plot model history
plot_model_history(model3_info)


print "Model took %0.2f seconds to train"%(end - start)
# compute test accuracy
print "Accuracy on test data is: %0.2f"%accuracy(test_features, test_labels, model3
score = model3.evaluate(test_features, test_labels, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

A.3 2D CNN + Dropout

As frameworks usadas para a implementação foram Theano + Keras.

```
# src: http://parneetk.github.io/blog/cnn-mnist/
# If using tensorflow, set image dimensions order
from keras import backend as K
if K.backend()=='tensorflow':
```

```
    K.set_image_dim_ordering("tf")
import time
import matplotlib.pyplot as plt
import numpy as np
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers.convolutional import Convolution2D, MaxPooling2D
from keras.layers import Activation, Flatten, Dense, Dropout
from keras.optimizers import SGD
from keras.layers.normalization import BatchNormalization
#% matplotlib inline
np.random.seed(2017)
# Load MNIST Dataset
from keras.datasets import mnist
(train_features, train_labels), (test_features, test_labels) = mnist.load_data()
_, img_rows, img_cols =  train_features.shape
num_classes = len(np.unique(train_labels))
num_input_nodes = img_rows*img_cols
print "Number of training samples: %d"%train_features.shape[0]
print "Number of test samples: %d"%test_features.shape[0]
print "Image rows: %d"%train_features.shape[1]
print "Image columns: %d"%train_features.shape[2]
print "Number of classes: %d"%num_classes
# Pre-processing
train_features = train_features.reshape(train_features.shape[0], 1, img_rows, img_
test_features = test_features.reshape(test_features.shape[0], 1, img_rows, img_col
train_features /= 255
test_features /= 255
print('train_features shape:', train_features.shape)
print(train_features.shape[0], 'train samples')
print(test_features.shape[0], 'test samples')

# convert class labels to binary class labels
train_labels = np_utils.to_categorical(train_labels, num_classes)
test_labels = np_utils.to_categorical(test_labels, num_classes)
# Function to plot model accuracy and loss
def plot_model_history(model_history):
    fig, axs = plt.subplots(1,2,figsize=(15,5))
    # summarize history for accuracy
    axs[0].plot(range(1,len(model_history.history['acc'])+1),model_history.history
    axs[0].plot(range(1,len(model_history.history['val_acc'])+1),model_history.hist
    axs[0].set_title('Model Accuracy')
    axs[0].set_ylabel('Accuracy')
    axs[0].set_xlabel('Epoch')
    axs[0].set_xticks(np.arange(1,len(model_history.history['acc'])+1),len(model_h
    axs[0].legend(['train', 'val'], loc='best')
    # summarize history for loss
    axs[1].plot(range(1,len(model_history.history['loss'])+1),model_history.history
    axs[1].plot(range(1,len(model_history.history['val_loss'])+1),model_history.his
```

```
    axs[1].set_title('Model Loss')
    axs[1].set_ylabel('Loss')
    axs[1].set_xlabel('Epoch')
    axs[1].set_xticks(np.arange(1,len(model_history.history['loss'])+1),len(model_
    axs[1].legend(['train', 'val'], loc='best')
    plt.show()
# Funtion to compute test accuracy
def accuracy(test_x, test_y, model):
    result = model.predict(test_x)
    predicted_class = np.argmax(result, axis=1)
    true_class = np.argmax(test_y, axis=1)
    num_correct = np.sum(predicted_class == true_class)
    accuracy = float(num_correct)/result.shape[0]
    return (accuracy * 100)

# A 2D CNN + Dropout: 0.25; 0.50
# Define the model
model4 = Sequential()
model4.add(Convolution2D(32, 3, 3, border_mode='valid', input_shape=(1, 28, 28)))
model4.add(Activation("relu"))
model4.add(Convolution2D(32, 3, 3,  border_mode='valid'))
model4.add(Activation("relu"))
model4.add(MaxPooling2D(pool_size=(2, 2)))
model4.add(Dropout(0.25))
model4.add(Flatten())
model4.add(Dense(128))
model4.add(Activation("relu"))
model4.add(Dropout(0.5))
model4.add(Dense(num_classes))
model4.add(Activation("softmax"))
# Compile the model
model4.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accurac
# Train the model
start = time.time()
model4_info = model4.fit(train_features, train_labels, batch_size=100, \
                         nb_epoch=100, verbose=1, validation_split=0.2)
end = time.time()
# plot model history
plot_model_history(model4_info)


print "Model took %0.2f seconds to train"%(end - start)
# compute test accuracy
print "Accuracy on test data is: %0.2f"%accuracy(test_features, test_labels, model4
score = model4.evaluate(test_features, test_labels, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

A.4 DeepLDA

O código utilizado encontra-se em: https://github.com/CPJKU/deep$_lda$

A.5 AE's

O código usado foi baseado em fontes existentes na web sobre esta temática.

A.6 GAN's

O código usado foi baseado nesta fonte: https://oshearesearch.com/index.php/2016/07/01/mnist-generative-adversarial-model-in-keras/

A.7 Comparação das imagens

```
from itertools import izip
import Image

#i1 = Image.open("white.png")
#i2 = Image.open("black.png")
i1 = Image.open("./images2compare/dcgan_generated_image_epoch_200_1000.png")
i2 = Image.open("./images2compare/gan_generated_image_epoch_1000_1000.png")

assert i1.mode == i2.mode, "Different kinds of images."
assert i1.size == i2.size, "Different sizes."

pairs = izip(i1.getdata(), i2.getdata())
if len(i1.getbands()) == 1:
    # for gray-scale jpegs
    dif = sum(abs(p1-p2) for p1,p2 in pairs)
else:
    dif = sum(abs(c1-c2) for p1,p2 in pairs for c1,c2 in zip(p1,p2))

ncomponents = i1.size[0] * i1.size[1] * 3
print "Difference (percentage):", (dif / 255.0 * 100) / ncomponents

# src: http://www.pyimagesearch.com/2014/09/15/python-compare-two-images/

from skimage.measure import compare_ssim as ssim #structural_similarity as ssim
import matplotlib.pyplot as plt
import numpy as np
import cv2

def mse(imageA, imageB):
        # the 'Mean Squared Error' between the two images is the
        # sum of the squared difference between the two images;
        # NOTE: the two images must have the same dimension
```

```
        err = np.sum((imageA.astype("float") - imageB.astype("float")) ** 2)
        err /= float(imageA.shape[0] * imageA.shape[1])

        # return the MSE, the lower the error, the more "similar"
        # the two images are
        return err

def compare_images(imageA, imageB, title):
        # compute the mean squared error and structural similarity
        # index for the images
        m = mse(imageA, imageB)
        s = ssim(imageA, imageB)

        # setup the figure
        fig = plt.figure(title)
        plt.suptitle("MSE: %.2f, SSIM: %.2f" % (m, s))

        # show first image
        ax = fig.add_subplot(1, 2, 1)
        plt.imshow(imageA, cmap = plt.cm.gray)
        plt.axis("off")

        # show the second image
        ax = fig.add_subplot(1, 2, 2)
        plt.imshow(imageB, cmap = plt.cm.gray)
        plt.axis("off")

        # show the images
        plt.show()

# load the images -- the original, the original + contrast,
# and the original + photoshop
original1 = cv2.imread("images2compare/dcgan_generated_image_epoch_1000_1000.png")
original2 = cv2.imread("images2compare/gan_generated_image_epoch_1000_1000.png")
#contrast = cv2.imread("images2compare/jp_gates_contrast.png")
#shopped = cv2.imread("images2compare/jp_gates_photoshopped.png")

# convert the images to grayscale
original1 = cv2.cvtColor(original1, cv2.COLOR_BGR2GRAY)
original2 = cv2.cvtColor(original2, cv2.COLOR_BGR2GRAY)
#contrast = cv2.cvtColor(contrast, cv2.COLOR_BGR2GRAY)
#shopped = cv2.cvtColor(shopped, cv2.COLOR_BGR2GRAY)

# loop over the images
for (i, (name, image)) in enumerate(images):
        # show the image
        ax = fig.add_subplot(1, 3, i + 1)
        ax.set_title(name)
        plt.imshow(image, cmap = plt.cm.gray)
```

```
        plt.axis("off")

# show the figure
plt.show()

# compare the images
compare_images(original1, original2, "Original1 vs. Original2")
#compare_images(original, contrast, "Original vs. Contrast")
#compare_images(original, shopped, "Original vs. Photoshopped")
```

O Código relativo à implementação do SSIM é consideravelmente mais extenso por isso não vai ser apresentado aqui.

Bibliografia - Artigos

[03] "All-optical pattern recognition for digital real-time information processing." In: (2003). URL: http://www.ncbi.nlm.nih.gov/pubmed/13678353.

[15] "Automatic differentiation in machine learning: a survey". In: *arxiv.org* (2015). DOI: 1502.05767. URL: https://arxiv.org/pdf/1502.05767.pdf.

[16] "Quantum Machine Learning without Measurements". In: (2016). DOI: 1612.05535. URL: https://arxiv.org/abs/1612.05535.

[99] "A geometrical representation of McCulloch-Pitts neural model and its applications". In: *IEEE Transactions on Neural Networks* 10 (1999). ISSN: 10459227. DOI: 10.1109/72.774263.

[Ada11] Andrew Y. Ng Adam Coates Honglak Lee. "An Analysis of Single Layer Networks in Unsupervised Feature Learning". In: *AISTATS* (2011).

[AH15] Steven H. Adachi and Maxwell P. Henderson. "Application of quantum annealing to training of deep neural networks". In: *arXiv:1510.00635* (2015). eprint: 1510.00635. URL: http://arxiv.org/abs/1510.06356.

[DY14a] Li Deng and Dong Yu. "Deep Learning: Methods and Applications". In: (2014). DOI: 10.1136/bmj.319.7209.0a. URL: http://books.google.com/books?id=46qNoAEACAAJ%7B%5C%7B%7D%7B%5C&%7D%7B%5C%7D%7Dpgis=1.

[Fie98] Emile Fiesler. "Discrete all-positive multilayer perceptrons for optical implementation". In: *Optical Engineering* (1998). DOI: 10.1117/1.601963. URL: http://opticalengineering.spiedigitallibrary.org/article.aspx?doi=10.1117/1.601963.

[Hol+06] a J Holden et al. "Reducing the Dimensionality of". In: *Science (New York, N.Y.)* (2006). ISSN: 0036-8075.

[HOT06] Geoffrey E. Hinton, Simon Osindero, and Yee Whye Teh. "A fast learning algorithm for deep belief nets." In: *Neural computation* (2006). DOI: 10.1162/neco.2006.18.7.1527. URL: http://www.ncbi.nlm.nih.gov/pubmed/16764513.

[Kok+07] Pieter Kok et al. "Linear optical quantum computing with photonic qubits". In: *Reviews of Modern Physics* (2007). DOI: 10.1103/RevModPhys.79.135.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: (2012). Ed. by F Pereira et al., pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[Lar+07] Hugo Larochelle et al. "An empirical evaluation of deep architectures on problems with many factors of variation". In: *Proceedings of the 24th International Conference on Machine Learning (2007)* (2007). DOI: 10.1145/1273496.1273556. URL: http://portal.acm.org/citation.cfm?doid=1273496.1273556.

[Lau+17] Hoi-Kwan Lau et al. "Quantum Machine Learning over Infinite Dimensions". In: *Phys. Rev. Lett.* (2017). DOI: 10.1103/PhysRevLett.118.080501. URL: https://link.aps.org/doi/10.1103/PhysRevLett.118.080501.

[LC] Yann Lecun and Corinna Cortes. "The MNIST database of handwritten digits". In: (). URL: http://yann.lecun.com/exdb/mnist/.

[LMR13] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. "Quantum algorithms for supervised and unsupervised machine learning". In: *arXiv* 1307.0411 (2013). URL: http://arxiv.org/abs/1307.0411.

[MDH12] Abdel-rahman Mohamed, George E. Dahl, and Geoffrey Hinton. "Acoustic Modeling Using Deep Belief Networks". In: *IEEE Transactions on Audio, Speech, and Language Processing* (2012). DOI: 10.1109/TASL.2011.2109382.

[MP43] Warren S. McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The Bulletin of Mathematical Biophysics* (1943). DOI: 10.1007/BF02478259.

[Nam+98] Last Name et al. "Convolution Networks for Images, Speech, and Time-Series". In: *Igarss 2014* (1998). ISSN: 1726670X. DOI: 10.1007/s13398-014-0173-7.2.

[out04] Zhou Wang e outros. "Image Quality Assessment: From Error Visibility to Structural Similarity". In: *IEEE Transactions on Image Processing* (2004). URL: http://www.cns.nyu.edu/pub/eero/wang03-reprint.pdf.

[out16] Noam Shazeer e outros. "Swivel: Improving Embeddings by Noticing What is Missing". In: (2016). URL: https://arxiv.org/pdf/1602.02215.pdf.

[PPQ91] Invited Paper, Demetri Psaltis, and Yong Qiao. "Optical multilayer neural networks". In: (1991).

[PV14] Arnab Paul and Suresh Venkatasubramanian. "Why does Deep Learning work? - A perspective from Group Theory". In: *Arxiv* (2014). URL: http://arxiv.org/abs/1412.6621.

[Ren+16] Haoran Ren et al. "On-chip noninterference angular momentum multiplexing of broadband light". In: *Science* (2016). DOI: 10.1126/science.aaf1112. URL: http://science.sciencemag.org/content/early/2016/04/06/science.aaf1112.

[Sch14] Jfffdfffdrgen Schmidhuber. "Deep Learning in Neural Networks: An Overview". In: (2014). URL: http://www.idsia.ch.

[WKS14] Nathan Wiebe, Ashish Kapoor, and Krysta M. Svore. "Quantum Deep Learning". In: 2 (2014). URL: http://arxiv.org/abs/1412.3489.

[WP87] Kelvin Wagner and Demetri Psaltis. "Multilayer optical learning networks." In: *Applied optics* 26.23 (1987). DOI: 10.1364/AO.26.005061. URL: http://www.ncbi.nlm.nih.gov/pubmed/20523485.

[Yin10] Mingsheng Ying. "Quantum computation, quantum theory and AI". In: *Artificial Intelligence* (2010). DOI: 10.1016/j.artint.2009.11.009. URL: http://dx.doi.org/10.1016/j.artint.2009.11.009.

Bibliografia - Livros

[Apo67] Tom M Apostol. *One-Variable Calculus with Introduction to Linear Algebra*. Vol. 1. 1967. ISBN: 0471000051.

[Basb] Simone Bassis. *Recent Advances of Neural Network Models*. ISBN: 9783319041285.

[Ben09] Yoshua Bengio. *Learning Deep Architectures for AI*. Vol. 2. 2009. ISBN: 2200000006. DOI: 10.1561/2200000006.

[BN07] Cm Bishop and Nm Nasrabadi. *Pattern Recognition and Machine Learning*. 2007. ISBN: 9780387310732. DOI: 10.1117/1.2819119. URL: http://medcontent.metapress.com/index/A65RM03P4874243N.pdf$%5Cbackslash$nhttp://www.library.wisc.edu/selectedtocs/bg0137.pdf$%5Cbackslash$nhttp://electronicimaging.spiedigitallibrary.org/article.aspx?doi=10.1117/1.2819119.

[DY14b] Li Deng and Dong Yu. *Deep Learning: Methods and Applications*. DOI:10.1561/2000000039. Now the essence of knowledge, 2014.

[ESV] Series Editors, Ramesh Sharda, and Stefan Voß. *Machine Learning Models and Algorithms for Big Data Classifi cation Thinking with Examples for Eff ective*. ISBN: 9781489976406.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[Hea15a] Jeff Heaton. *AIFH, Volume 3: Deep Learning and Neural Networks*. Heaton Research, Inc., 2015.

[Hea15b] Jeff Heaton. *Artificial Intelligence for Humans Volume 3: Deep Learning and Neural Networks*. 2015. ISBN: 9788578110796. DOI: 10.1017/CBO9781107415324.004.

[Mica] Microsoft. *An Introduction to Computational Networks and the Computational Network Toolkit*. (consultado em 01-03-2017). URL: https://www.microsoft.com/en-us/research/publication/an-introduction-to-computational-networks-and-the-computational-network-toolkit/.

[Nil98] Nils J. Nilsson. *Introduction to Machine Learning*. 1998.

[Oxf96] B. D. Ripley (University of Oxford). *Pattern Recognition and Neural Networks*. 1996. ISBN: 9788578110796. DOI: 10.1017/CBO9781107415324.004.

[RN13] Stuart Russell and Peter Norvig. *Artificial Intelligence A Modern Approach*. 2013. ISBN: 9780136042594. DOI: 10.1017/S0269888900007724. URL: http://scholar.google.com/scholar?hl=en%7B%5C&%7DbtnG=Search%7B%5C&%7Dq=intitle:No+Title%7B%5C#%7D0.

Bibliografia - Weblinks

[Acc] Accord. *Accord*. (consultado em 01-03-2017). URL: http://accord-framework.net/.

[Ada] Josh Patterson Adam Gibson. *Deep Learning*. (consultado em 01-07-2017). URL: https://www.safaribooksonline.com/library/view/deep-learning/9781491924570/ch04.html.

[ala] Abadi e al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. (consultado em 01-03-2017). URL: http://download.tensorflow.org/paper/whitepaper2015.pdf.

[alb] James Bergstra et al. *Theano: A CPU and GPU Math Compiler in Python*. (consultado em 01-03-2017). URL: http://conference.scipy.org/proceedings/scipy2010/pdfs/bergstra.pdf.

[al12] Bastien et al. *Theano: new features and speed improvements*. 2012.

[Ama] Amazon. *AWS: Amazon Web Services*. (consultado em 01-03-2017). URL: https://aws.amazon.com/.

[And] Michael Andersch. *Inference: The Next Step in GPU-Accelerated Deep Learning*. (consultado em 28-02-2017). URL: https://devblogs.nvidia.com/parallelforall/inference-next-step-gpu-accelerated-deep-learning/.

[Apaa] Apache. *Mahout*. (consultado em 01-03-2017). URL: http://mahout.apache.org/.

[Apab] Apache. *MAVEN*. (consultado em 01-03-2017). URL: https://maven.apache.org/.

[Apac] Apache. *MLlib*. (consultado em 01-03-2017). URL: http://spark.apache.org/mllib/.

[App] Apple. *Deep Learning Kit for Apple devices*. (consultado em 01-03-2017). URL: http://deeplearningkit.org/.

[Basa] Chris Basoglu. *Using CNTK with BrainScript*. (consultado em 01-03-2017). URL: https://github.com/Microsoft/CNTK/wiki/Using-CNTK-with-BrainScript.

[Basc] Bastien. *Theano: new features and speed improvements*. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop.

[Basd] Bastien. *Theano Related Projects*. (consultado em 02-03-2017). URL: https://github.com/Theano/Theano/wiki/Related-projects.

[Bena] Rodrigo Benenson. *Results of Classification*. (consultado em 25-04-2016). URL: http://rodrigob.github.io/are%7B%5C_%7Dwe%7B%5C_%7Dthere%7B%5C_%7Dyet/build/classification%7B%5C_%7Ddatasets%7B%5C_%7Dresults.html.

[Benb] Yoshua Bengio. *Deep Learning: Theoretical Motivations*. (consultado em 24-04-2016). URL: http://videolectures.net/deeplearning2015_bengio_theoretical_motivations/?q=bengio.

[Benc] Yoshua Bengio. *Deep Learning:Theoretical Motivations*. (consultado em 24-04-2016). URL: http://videolectures.net/deeplearning2015%7B%5C_%7Dbengio%7B%5C_%7Dtheoretical%7B%5C_%7Dmotivations/.

[Broa] Jason Brownlee. *8 Inspirational Applications of Deep Learning*. (consultado em 01-03-2017). URL: http://machinelearningmastery.com/inspirational-applications-deep-learning/.

[Brob] Jason Brownlee. *Machine Learning Algorithms*. (consultado em 17-04-2016). URL: http://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/.

[Cal] University of Southern California. *ANT*. (consultado em 25-04-2016). URL: https://ant.isi.edu/datasets/index.html.

[Cha] Chainer. *Chainer*. (consultado em 01-03-2017). URL: http://chainer.org/.

[Cho] Francois Chollet. *Building Autoencoders in Keras*. (consultado em 01-03-2017). URL: https://blog.keras.io/building-autoencoders-in-keras.html.

[Com] Infraestrutura Nacional de Computacao Distribuida. *Piloto de Cloud Computing*. (consultado em 01-03-2017). URL: `http://www.incd.pt/cloud-pilot.php`.

[Cona] Continuum. *Anaconda: Enterprise Ready Python for Big Data*. (consultado em 01-03-2017). URL: `https://www.continuum.io/why-anaconda`.

[Conb] ConvNetjs. *ConvNetjs Documentation*. (consultado em 17-04-2016). URL: `http://cs.stanford.edu/people/karpathy/convnetjs/`.

[Cop] Michael Copeland. *What is the Difference Between Artificial Intelligence, Machine Learning, and Deep Learning?* (consultado em 01-07-2017). URL: `https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/`.

[Dat] World of DataScience. *Data Science Ontology*. (consultado em 01-07-2017). URL: `http://worldofdatascience.com/`.

[Dava] Alfred David. *Tools for Deep Learning Neural Networks*. (consultado em 03-03-2017). URL: `https://www.linkedin.com/pulse/tools-deep-learning-neural-networks-alfred-david`.

[Davb] E. King Davis. *DLib-ML*. (consultado em 01-03-2017). URL: `http://www.jmlr.org/papers/v10/king09a.html`.

[dee] deeplearning.net. *Convolutional Neural Networks (LeNet)*. (consultado em 12-03-2017). URL: `http://deeplearning.net/tutorial/lenet.html`.

[dev] NumPy developers. *Numpy*. (consultado em 01-03-2017). URL: `http://www.numpy.org/`.

[Dic] Dictionary. *Definição de Percetrão*. (consultado em 02-04-2016). URL: `http://www.dictionary.com/browse/perceptron`.

[DL4] DL4J. *DeepLearning 4J*. (consultado em 01-03-2017). URL: `https://deeplearning4j.org/`.

[Doc] Docker. *Docker: A Better Way to Build Apps*. (consultado em 01-03-2017). URL: `https://www.docker.com/`.

[Efs] (University of Amsterdam) Efstratios Gavves. *UVA Deep Learning Course*. (consultado em 02-04-2016). URL: `http://uvadlc.github.io/`.

[Fou] R Foundation. *R*. (consultado em 01-03-2017). URL: https://www.r-project.org/.

[Fre] Ana Freitas. *GFBioinfo*. (consultado em 21-03-2017). URL: http://web.tecnico.ulisboa.pt/ana.freitas/bioinformatics.ath.cx/bioinformatics.ath.cx/index0717.html?id=109.

[Git] Git. *Git*. (consultado em 01-03-2017). URL: https://github.com/.

[Gooa] Ian e outros Goodfellow. *Generative Models*. (consultado em 01-03-2017). URL: https://openai.com/blog/generative-models/.

[Goob] Google. *Cuda-ConvNet2*. (consultado em 01-03-2017). URL: https://code.google.com/archive/p/cuda-convnet2/.

[Gooc] Google. *Google Brain*. (consultado em 01-03-2017). URL: https://research.google.com/.

[Good] Google. *Google Cloud Platform*. (consultado em 01-03-2017). URL: https://cloud.google.com/.

[H2O] H2O. *H2O*. (consultado em 01-03-2017). URL: https://www.h2o.ai/.

[H2O16] H2O.ai. *H2O Deep Learning Package*. 2016. URL: http://www.h2o.ai/verticals/algos/deep-learning/ (visited on 04/17/2016).

[Ima] ImageNet. *Large Scale Visual Recognition Challenge 2012 (ILSVRC2012)*. (consultado em 01-03-2017). URL: http://image-net.org/challenges/LSVRC/2012/.

[Int] Intel. *Intel Deep Learning SDK*. (consultado em 01-03-2017). URL: https://software.intel.com/en-us/deep-learning-sdk.

[Jia] Yangqing Jia. *Caffe*. (consultado em 01-03-2017). URL: http://caffe.berkeleyvision.org/.

[Jup] Jupyther. *Jupyther*. (consultado em 01-03-2017). URL: http://jupyter.org/.

[Kag] Kaggle. *kaggle*. (consultado em 25-04-2016). URL: https://www.kaggle.com/.

[Kar] Karpathy. *ConvNetJS: Deep Learning in your browser*. (consultado em 01-03-2017). URL: http://cs.stanford.edu/people/karpathy/convnetjs/.

[Ker] Keras. *Keras*. (consultado em 01-03-2017). URL: `https://keras.io/`.

[Klo] Andreas Klockner. *CUDA vs OpenCL: Which should I use?* (consultado em 28-02-2017). URL: `https://wiki.tiker.net/CudaVsOpenCL`.

[Kno] Knoema. *knoema*. (consultado em 25-04-2016). URL: `http://knoema.com/atlas/topics/Cyber-Security/datasets`.

[Las] Lasagne. *Lasagne Library Documentation*. (consultado em 17-04-2016). URL: `http://lasagne.readthedocs.org/en/latest/index.html`.

[LCJ] NYU LeCun, Yann / Courant Institute, New York Cortes, Corinna /Google Labs, and Redmond J.C. Burges, Christopher / Microsoft Research. *MNIST*. (consultado em 25-04-2016). URL: `http://yann.lecun.com/exdb/mnist/`.

[LeC] Yann LeCun. *THE MNIST DATABASE of handwritten digits*. (consultado em 25-04-2016). URL: `http://yann.lecun.com/exdb/mnist/`.

[LUA] LUA. *LUA*. (consultado em 01-03-2017). URL: `https://www.lua.org/`.

[Mal] Tomasz Malisiewicz. *Deep Learning vs Machine Learning vs Pattern Recognition*. (consultado em 18-03-2016). URL: `http://www.computervisionblog.com/2015/03/deep-learning-vs-machine-learning-vs.html`.

[Mata] MathWorks. *Matlab*. (consultado em 01-03-2017). URL: `https://www.mathworks.com/products/matlab.html`.

[Matb] Rainer Kelz e Gerhard Widmer Matthias Dorfer. *Deep Linear Discriminant Analysis (DeepLDA)*. (consultado em 01-03-2017). URL: `https://github.com/CPJKU/deep_lda`.

[May] Madison May. *Python Deep Learning Frameworks Reviewed*. (consultado em 27-02-2017). URL: `https://indico.io/blog/python-deep-learning-frameworks-reviewed/`.

[Mei] Karlheinz Meier. *BrainScales*. (consultado em 28-03-2016). URL: `https://brainscales.kip.uni-heidelberg.de/public/index.html`.

[Roba] Eric Roberts. *Neural Networks*. (consultado em 21-03-2017). URL: https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history1.html.

[Robb] Eric Roberts. *Neural Networks*. (consultado em 01-03-2017). URL: https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/Applications/index.html.

[Robc] Eric (Sophomore College) Roberts. *The Intellectual Excitement of Computer Science*. (consultado em 02-04-2016). URL: https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/Neuron/index.html.

[RSt] RStudio. *RStudio*. (consultado em 01-03-2017). URL: https://www.rstudio.com/.

[San] Gokula Krishnan Santhanam. *Anatomy of Deep Learning Frameworks*. (consultado em 01-03-2017). URL: http://www.kdnuggets.com/2017/02/anatomy-deep-learning-frameworks.html.

[SAS] SAS. *SAS*. (consultado em 01-03-2017). URL: https://www.sas.com/pt_pt/home.html.

[Sch] Cristina Scheau. *Regularization in deep learning*. (consultado em 01-03-2017). URL: https://chatbotslife.com/regularization-in-deep-learning-f649a45d6e0#.6mitjbbya.

[sci] scikit-learn. *scikit-learn*. (consultado em 01-03-2017). URL: http://scikit-learn.org/stable/.

[Sho] Shogun-ToolBox. *Shogun-ToolBox*. (consultado em 01-03-2017). URL: http://www.shogun-toolbox.org/.

[Sot] Pablo Soto. *The major advancements in Deep Learning in 2016*. (consultado em 01-03-2017). URL: https://tryolabs.com/blog/2016/12/06/major-advancements-deep-learning-2016/.

[Staa] University of Stanford. *Convolutional Neural Networks (CNNs / ConvNets)*. (consultado em 01-03-2017). URL: http://cs231n.github.io/convolutional-networks/.

[Stab] University of Stanford. *deepdive*. (consultado em 25-04-2016). URL: http://deepdive.stanford.edu/opendata/.

[Sun] Sun-Oracle. *JAVA*. (consultado em 01-03-2017). URL: https://www.java.com/en/.

[Met] Cade Metz. *For Google, Quantum Computing Is Like Learning to Fly*. (consultado em 27-03-2016). URL: http://www.wired.com/2015/12/for-google-quantum-computing-is-like-learning-to-fly/.

[Micb] Microsoft. *Azure*. (consultado em 01-03-2017). URL: https://azure.microsoft.com/en-us/.

[Micc] Microsoft. *CNTK Microsoft Research*. (consultado em 02-03-2017). URL: https://habrahabr.ru/company/microsoft/blog/275959/.

[Micd] Microsoft. *Galeria de Modelos*. (consultado em 01-03-2017). URL: https://www.microsoft.com/en-us/research/product/cognitive-toolkit/model-gallery/.

[Mice] Microsoft. *Microsoft Cognitive ToolKit*. (consultado em 01-03-2017). URL: https://www.microsoft.com/en-us/research/product/cognitive-toolkit/.

[Moz] Michael (University of Colorado - Department of Computer Science) Mozer. *Neural Networks and Deep Learning CSCI 7222*. (consultado em 02-04-2016). URL: https://www.cs.colorado.edu/%7B~%7Dmozer/Teaching/syllabi/DeepLearning2015/.

[MxN] MxNet. *MxNet*. (consultado em 01-03-2017). URL: http://mxnet.io/.

[Ng] Andrew Ng. *Stacked Autoencoders*. (consultado em 01-03-2017). URL: %7Bhttp://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders%7D.

[NVi] NVidia. *CUDA*. (consultado em 01-03-2017). URL: http://www.nvidia.com/object/cuda_home_new.html.

[Ope] OpenCV. *OpenCV*. (consultado em 01-03-2017). URL: http://opencv.org/.

[Orx] Orxy. *Oryx2*. (consultado em 01-03-2017). URL: http://oryx.io/.

[Par] Roger Parloff. *Why Deep Learning is suddenly changing your life*. (consuldado em 21-03-2017). URL: http://fortune.com/ai-artificial-intelligence-deep-machine-learning/.

[Poi] West Point. *usma*. (consultado em 25-04-2016). URL: http://www.usma.edu/crc/sitepages/datasets.aspx.

[Pyt] Python. *Pickle*. (consultado em 01-03-2017). URL: https://docs.python.org/3.1/library/pickle.html.

[Tena] Fabien Tence. *Choosing a Deep Learning Software*. (consultado em 27-02-2017). URL: http://ankivil.com/choosing-a-deep-learning-software/.

[Tenb] TensorFlow. *TensorBoard: Graph Visualization*. (consultado em 01-03-2017). URL: https://www.tensorflow.org/get_started/graph_viz.

[Tenc] TensorFlow. *TensorFlow Documentation*. (consultado em 17-04-2016). URL: http://www.tensorflow.org/tutorials/mnist/beginners/index.md.

[Tend] TensorFlow. *TensorFlow Documentation*. (consultado em 17-04-2016). URL: http://www.tensorflow.org/tutorials/mnist/beginners/index.md.

[The] Theano. *Theano Documentation*. (consultado em 17-04-2016). URL: http://deeplearning.net/software/theano/.

[Tora] Torch. *Torch Documentation*. (consultado em 17-04-2016). URL: http://torch.ch/.

[Torb] Univeristy of Toronto. *Cifar*. (consultado em 25-04-2016). URL: http://www.cs.toronto.edu/%7B~%7Dkriz/cifar.html.

[Tri] Alexandre Trindade. *Convolufffdfffdfffdfffdo*. (consultado em 01-03-2017). URL: http://coimbra.lip.pt/~rui/Convolucao.PDF.

[Uci] Uci. *UCI*. (consultado em 29-04-2016). URL: http://archive.ics.uci.edu/ml/index.html.

[Uni] Andrej Karpathy - Stanford University. *CS231n CNN for Visual Recognition*. (consultado em 01-07-2017). URL: http://cs231n.github.io/neural-networks-3/.

[Van] Vicent Vanhoucke. *Deep Learning Take machine learning to the next level*. (consultado em 20-03-2016). URL: https://www.udacity.com/course/deep-learning--ud730.

[Wai] Waikato. *Weka: Data Mining Software in JAVA*. (consultado em 01-03-2017). URL: http://www.cs.waikato.ac.nz/ml/weka/.

[Wee] Christian Weedbrook. *Physicists extend quantum machine learning to infinite dimensions*. (consultado em 01-03-2017). URL: https://phys.org/news/2017-03-physicists-quantum-machine-infinite-dimensions.html?utm_source=nwletter&utm_medium=email&utm_campaign=weekly-nwletter.

[Whi] Stephen James Whitworth. *GoLearn*. (consultado em 01-03-2017). URL: `https://github.com/sjwhitworth/golearn`.

[Wika] Wikipedia. *Boltzmann Machine*. (consultado em 01-03-2017). URL: `https://en.wikipedia.org/wiki/Boltzmann_machine`.

[Wikb] Wikipedia. *Convolutional Neural Network*. (consultado em 01-03-2017). URL: `https://en.wikipedia.org/wiki/Convolutional_neural_network`.

[Wikc] Wikipedia. *Deep Belief Networks*. (consultado em 01-03-2017). URL: `https://en.wikipedia.org/wiki/Deep_belief_network`.

[Wikd] Wikipedia. *Deep Learning Definition*. (consultado em 21-04-2016). URL: `https://en.wikipedia.org/wiki/Deep%7B%5C_%7Dlearning`.

[Wike] Wikipedia. *List of datasets for Machine Learning*. (consultado em 25-04-2016). URL: `https://en.wikipedia.org/wiki/List%7B%5C_%7Dof%7B%5C_%7Ddatasets%7B%5C_%7Dfor%7B%5C_%7Dmachine%7B%5C_%7Dlearning%7B%5C_%7Dresearch`.

[Wikf] Wikipedia. *Principal Component Analysis*. (consultado em 01-03-2017). URL: `https://en.wikipedia.org/wiki/Principal_component_analysis`.

[Wikg] Wikipedia. *Restricted Boltzmann Machine*. (consultado em 01-03-2017). URL: `https://en.wikipedia.org/wiki/Restricted_Boltzmann_machine`.

[Wu] Neal Wu. *TensorFlow Models*. (consultado em 02-03-2017). URL: `https://github.com/tensorflow/models`.

[Yeg] Serdar Yegulalp. *11 open source tools to make the most of machine learning*. (consultado em 01-03-2017). URL: `http://www.infoworld.com/article/2853707/robotics/11-open-source-tools-machine-learning.html#slide10`.

UNIVERSIDADE DE ÉVORA

Contactos:
Universidade de Évora
Escola de Ciências e Tecnologia
Departamento de Informática
Rua Romão Ramalho 59
7000 - 671 Évora | Portugal