UNIVERSIDADE
FEDERAL RURAL
DE PERNAMBUCO

Diogo Felipe Félix de Melo

Aprendizado profundo com capacidade computacional reduzida: uma aplicação à quebra de CAPTCHAs.

Recife

Julho de 2018

Diogo Felipe Félix de Melo

Aprendizado profundo com capacidade computacional reduzida: uma aplicação à quebra de CAPTCHAs.

Monografia apresentada ao Curso de Bacharelado em Ciências da Camputação da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Ciências da Camputação.

Universidade Federal Rural de Pernambuco – UFRPE

Departamento de Computação

Bacharelado em Ciências da Camputação

Orientador: Pablo de Azevedo Sampaio

Recife

Julho de 2018

Agradecimentos

Meus pais, familiares e amigos.

Ao meu orientador, por toda a paciência e dedicação.

Resumo

Palavras-chave: Aprendizado de Maquina, Aprendizado Profundo, CAPTCHA.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

CAPTCHA	Completely Automated Public Turing tests to tell Computers and Humans Apart. Testes automatizados de Turing para diferenciar humanos e computadores.
OCR	Optical Character Recognition.
RGB	Red-Green-Blue. Canais de codificação de cores.
RAM	Random Access Memory. Memória de Acesso Aleatório.
HIP	Human Interaction Proofs. Provas de interação humana.

Lista de símbolos

$\{\ldots\}$	Conjunto de todas as possibilidades da variável .... Usualmente um espaço vetorial.
$\langle\ldots\rangle_D$	Valor esperado de ... no conjunto D.
$\lvert\ldots\rvert$	Dimensão de ....
∇_x	Operador gradiente com respeito ao vetor x.
$\Re$	Conjunto dos números reais.
$\Re^\kappa$	Espaço dos tensores de ranque $\lvert\kappa\rvert$ sob o conjunto dos números reais. κ define uma família de índices. $\lvert\kappa\rvert = 1$ para vetores, $\lvert\kappa\rvert = 2$ para matrizes e assim por diante.
$\Re^\kappa[0,1]$	Compacto dos tensores de ranque κ sob o conjunto $[a,b]$, com $a,b \in \Re$.
2^D	Conjunto de todos os subconjuntos de D.

Sumário

1 Introdução

Algoritmos de aprendizado baseados em neurologia são conhecidos desde meados do século passado (1). Das proposições iniciais até os dias de hoje, essa classe de modelos tem evoluído em complexidade e técnicas de forma contínua, culminando em um alto poder de expressividade e níveis cada vez mais abstratos de representação (ver (2) ou (3) para uma breve revisão histórica). Os poucos resultados teóricos disponíveis demonstram que redes neurais possuem um alto poder de generalização, sendo capaz de, sob certas circunstâncias, codificar diversas classes de funções (4, 5). Apesar dos avanços na área, foi apenas recentemente que modelos neurais começaram a redefinir o estado da arte, superando outras classes de algoritmos de aprendizado de máquina (6) e até mesmo alcançando performances sobre humanas (7). Tais avanços foram possíveis devido a três fatores chaves: a viabilização de bases de dados cada vez maiores, o aumento do poder computacional e o desenvolvimento de novas arquiteturas e técnicas de treino.

A crescente melhoria de performance dos modelos neurais de aprendizado profundo tem motivado estudos em áreas onde se é preciso distinguir computadores e humanos. Dentre essas áreas temos os CAPTCHAs (8) (do inglês Completely Automated Public Turing tests to tell Computers and Humans Apart), que definem uma coleção de técnicas que tem como objetivo bloquear a ação de agentes autônomos na rede mundial de computadores. Um dos subconjuntos mais conhecidos dessas técnicas talvez seja o de CAPTCHAs baseados em texto (9). Nesse tipo de desafio, uma imagem contendo uma sequência de caracteres é exibida e a validação é feita pela comparação entre o texto informado pelo usuário e a resposta correta. Formulado como um problema de aprendizado de máquina, desejamos descobrir de forma automatizada um mapa entre a imagem e o texto codificado. Na versão informada do problema, um ser humano escolhe previamente técnicas de preprocessamento (filtros, segmentação de caracteres, etc.) antes que o aprendizado propriamente dito ocorra. Ajudados por humanos, redes neurais simples e com poucos exemplos conseguem resultados satisfatórios nesse tipo de desafio (10). De fato, mesmo técnicas ingênuas como contagem de pixels podem obter bons resultados quando o preprocessamento correto é fornecido (11). Na versão não informada, entretanto, o problema encontrar mapas imagem-texto de forma automatizada é usualmente muito mais desafiador. Em trabalhos recentes, foram relatados modelos baseados em redes neurais capazes de burlar esse tipo de desafio com acurácias de acerto próximos à humana em sequências sorteadas a partir de um repositório com milhões de exemplos (12) e em modelos com alta eficiência de dados (13), mas computacionalmente intensivos. Para o problema geral de quebrar CAPTCHAs baseados em texto, entretanto, modelos de aprendizado profundo ainda mostram desempenho inferior ao humano. Contudo, pesquisas recentes apontam para avanços claros nos próximos anos (14). Em comum, esses

modelos possuem a necessidade de clusters e/ou sistemas de computação sob demanda para treinamento, com hardware de alto poder de processamento e/ou paralelização, como GPUs e TPUs. Adicionalmente, as bases de treinamento necessárias comumente alcançam alguns terabytes e envolvem grandes operações de aquisição e/ou geração.

Neste trabalho propomos uma abordagem comparativa entre diferentes arquiteturas de redes neurais para a solução de CAPTCHAs baseados em texto sem informação humana, nos restringindo, entretanto, à um ambiente com poder computacional reduzido. Pretendemos mostrar que é possível fazer uso dessas técnicas em um mero computador pessoal (na contra-mão dos trabalhos usualmente encontrados na literatura) e ainda obter resultados próximos ao estado da arte. Este trabalho se encontra organizado como segue. No capítulo 2 apresentamos uma breve introdução à diferentes tipos de CAPTCHAs, com ênfase em desafios baseados em texto. Sequencialmente, no capítulo 3, arquiteturas e técnicas de projeto e treino de redes neurais comuns na literatura são abordados. Os principais resultados do uso dessas técnicas em CAPTCHAs de texto explorados e nossas considerações iniciais sobre essa aplicação apresentadas. No capítulo 4 uma descrição das arquiteturas dos modelos usados neste estudo é feita em conjunto com uma breve fundamentação para as escolhas. No capítulo 5, detalhes dos experimentos realizados são formalizados. Por fim, no capítulo 6, os resultados dos experimentos são comparados e nossas conclusões e considerações finais apresentadas.

2 CAPTCHAs

Neste capítulo abordaremos como funcionam os principais tipos de CAPTCHA conhecidos. Daremos um enfoque especial aos CAPTCHAs baseados em texto, objeto de estudo do presente trabalho e uma formulação mais precisa para o esse tipo de desafio é apresentada.

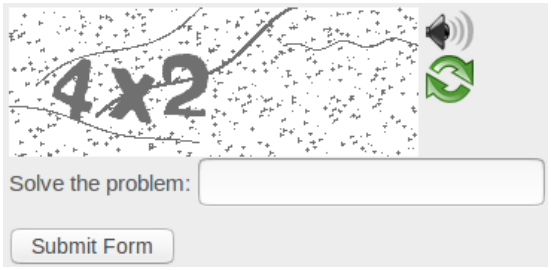
2.1 Introdução

CAPTCHAs (8) ou HIP (10) (do inglês Human Interaction Proofs), são um conjunto de técnicas que tem como objetivo discernir a ação automatizada de robôs da ação de seres humanos na Rede Mundial de Computadores. Esses filtros tem sido usados de forma efetiva na defesa de diversos tipos de ataque: proteger informações sensíveis, como *e-mail* e dados pessoais; impedir tentativas de *login* automatizados; coibir acesso massivo à sistemas de bases de dados; entre outros. Entretanto, desde as primeiras aplicações até os dias de hoje, existe uma corrida co-evolucionária entre atacantes e defensores. Por um lado, algoritmos de '*quebra*' de CAPTCHA se tornam cada vez mais sofisticados e precisos. Por outro lado, filtros mais complexos são desenvolvidos. Contudo, como explicado por Chellapilla *Et al.* (10), existe um balanço entre complexidade e factibilidade que os defensores devem buscar, explorando habilidades em que humanos ainda não foram ultrapassados por máquinas. O autor também estima que os requisitos mínimos de um HIP para ser considerado efetivo é ser solúvel por humanos 90% das vezes e robôs em apenas 1% das tentativas, sendo esses valores dependentes do custo de repetição do ataque.

De forma geral, esses filtros podem ser formulados como um desafio sobre um conjunto de domínio cuja a resposta é um token. O domínio pode ser um trecho de áudio, uma sequencia de imagens ou até mesmo o histórico de navegação do desafiado. O token pode ser constituído de um conjunto de ações, o texto extraído de um áudio ou imagem, ou possuir um histórico de navegação confiável. Podem ainda ser constituídos de uma única etapa ou de varias. Uma categorização dos tipos de CAPTCHA pode ser encontrado em (15). Na figura 1 podemos ver diferentes tipos de CAPTCHAs gerados com a biblioteca de código aberto *Securimage* (16).

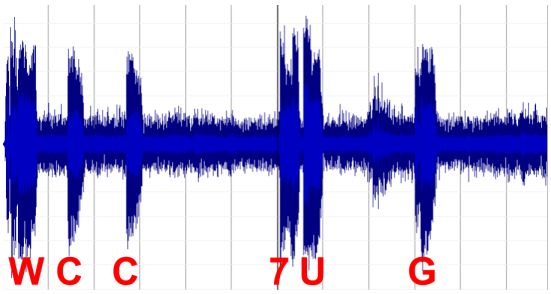
O processo de '*quebra*' envolve duas ideias principais: explorar vulnerabilidades e/ou uso de algoritmos inteligentes. Por serem testes automatizados, CAPTCHAs geralmente apresentam algum padrão de comportamento ou falha de projeto. A padronização na geração de desafios pode permitir que um atacante desenvolva heurísticas de ataque. Imagens com mesmo espaçamento de caracteres ou padrões repetitivos podem ser explorados e facilitar a segmentação da imagem, como foi explorado por (11). Falhas ou vieses

(a) Letras



(b) Desafio matemático

(c) Duas palavras



(d) Áudio

Figura 1 – Diferentes tipos de CAPTCHAs
(a) O desafiado deve reconhecer corretamente um texto formado por caracteres não correlacionados. (b) O desafio consiste em extrair e resolver corretamente um problema simples de álgebra. (c) Identificar palavras sorteadas de um repositório. (d) O desafio consiste com extrair corretamente um conjunto de símbolos codificados em áudio. Todos os exemplos foram gerados com a biblioteca Securimage. O espectro de intensidades em (d) foi gerado pelo autor a partir do áudio gerado pela biblioteca.

no projeto desses métodos também podem expor brechas. Quanto ao uso de algoritmos inteligentes, redes neurais recebem um lugar de destaque devido a flexibilidade e capacidade de generalização que esses modelos conseguem alcançar. O uso dessa classe de algoritmos foi a abordagem escolhida por (12) e (10), por exemplo. Redes neurais são exploradas no capítulo 3.

Ao longo dos anos os domínio e os desafios tem evoluído em complexidade, na medida que os atacantes evoluem em técnicas de ataque. Um exemplo da co-evolução desses sistemas é a forma como o sistema reCaptcha (17) tem mudado ao longo dos anos. Quando proposto, o sistema era baseado em trechos de livros e jornais antigos em que os melhores algoritmos conhecidos à época falharam em uma tarefa de OCR (do inglês optical character recognition). Trechos que não haviam sido corretamente identificados (teste) eram separados e exibidos para humanos juntamente com imagens cuja a extração era reconhecida (controle). Teste e controle eram comparados para certificar a atuação humana. Quando proposto, os autores alegaram que humanos seriam capazes de resolver o desafio em quase todos os casos e que comutadores teriam chance quase nula de passarem desapercebidos, já que o repositório de exemplos era composta por imagens em que as

técnicas vigentes à época haviam falhado. Porém, poucos anos depois, avanços em técnicas de OCR baseadas em redes neurais obtiveram 99% de precisão na base de palavras utilizada por esse sistema (12), inviabilizando o uso dessa técnica e forçando o reCaptcha a evoluir para uma segunda versão. Nessa nova abordagem, os dados de navegação dos usuários são analisados por um algoritmo de risco. Caso uma pontuação mínima seja atingida, o usuário é considerado humano, caso contrário, é exposto a problemas conhecidamente difíceis para computadores, como reconhecimento de objetos, contextualização de imagens e busca de similaridades, combinados com diferentes ações que devem ser performadas pelo usuário. Entretanto, Sivakorn *Et al.* (18) mostraram que explorando-se os critérios de avaliação de risco, seria possível enganar o sistema em 85% das vezes, forçando a constante atualização dos desafios.

2.2 Captchas de texto

Nos CAPTCHAs baseados em texto, uma imagem contendo uma sequência de caracteres é exibida ao desafiado. O teste consiste em conseguir recuperar corretamente o texto codificado na imagem. Matematicamente, uma imagem com altura A, largura L e C canais pode ser representada como um tensor $x \in \Re^{A,L,C}[0,1]$, de modo que x_{ijk} representa a intensidade do pixel na posição (i,j) canal k. Um token é uma sequencia u sob um alfabeto de símbolos Σ, podendo o comprimento da sequência ser limitado ou não. Assim, '*quebrar*' um CAPTCHA de texo é encontrar um mapa $f(x) \rightarrow u$ que inviabilize os critérios propostos por (10).

Quanto à imagem, usualmente se utiliza adição de ruído, linhas e/ou grades para dificultar o processo de segmentação de caracteres (10), bem como efeitos de distorção, corrosão e/ou dilatação, e transformações geométricas como rotação e translação, entre outros. Em desafios mais simplórios o número de canais pode ser reduzido à um, podendo ser representado como uma imagem em tons de cinza. Em desafios mais complexos o espaços de canais é explorado, aumentando algebricamente as possibilidades representação, mantendo, todavia, a factibilidade do desafio para seres humanos, dada nossa facilidade em distinguir cores[1]. Pode se explorar imagens sintéticas, geradas de forma automatizada por computadores, ou imagens naturais, como paisagens ou textos em fotos reais.

Quanto ao texto, diferentes complexidades podem ser adicionadas. Fixadas as demais variáveis, a dificuldade do desafio é usualmente proporcional ao tamanho de alfabeto escolhido. Dentre os mais simples, o conjunto dos números $\Sigma = 0123456789$ possui apenas dez símbolos. No outro extremo, os logogramas chineses constituem alfabetos com

[1] Nem sempre esta afirmação é verdadeira. De fato, pessoas que sofrem da Síndrome de Dalton, o Daltoninsmo, podem ter dificuldades em resolver desafios que explorem diferentes cores. O balanço entre complexidade e inclusão de pessoas com necessidades especiais ainda é uma questão em aberto no projeto de CAPTCHAs.

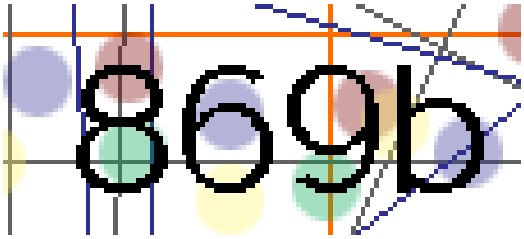
um número indeterminado de símbolos, com as representações mais usuais se limitando à algo em torno de 3000 caracteres. A tipografia também interfere na dificuldade de resolver um CAPTCHA. Para o mesmo alfabeto Σ existem diferentes possibilidades de representação gráfica dos seus caracteres. Fontes regulares tendem a fornecer representações mais simples, com a desvantagem de serem facilmente reconhecíveis por OCR. No outro extremo, fontes simbólicas e caracteres escritos à mão podem representar um grande desafio para máquinas. Quando os símbolos são sorteados de forma aleatória, humanos e máquinas, tipicamente, presenciam maiores dificuldades, dado que cada entrada deve ser reconhecida separadamente (errar o reconhecimento um símbolo significa uma falha completa). Quando sorteamos palavras a partir de um repositório, o desafio se torna mais amigável para humanos, primeiro por termos uma facilidade maior em reconhecer padrões do que especifidades e segundo nos permitir o uso indireto de outras fontes de conhecimento para a solução do problema. Se soubermos que os textos estão em Português, por exemplo, podemos esperar que nenhuma palavra conterá uma subsequência *tt* (gramaticalmente incorreto) ou que uma vogal será, provavelmente, seguida por uma consoante ou por uma vogal diferente (padrão mais frequente na língua). Entretanto, quando o repositório de palavras é exposto, atacantes podem desenvolver heurísticas para explorar o problema, como um dicionário de palavras ou correção gramatical para aprimorar a eficiência das técnicas. Além disso, repositórios de palavras induzem correlações entre os caracteres, o que limita espaço de possíveis tokens.

Ver (19) para um apanhado de CAPTCHAs de texto utilizados pelo mundo e (9) para uma a evolução desses sistemas ao longo dos anos. Na figura 2 temos alguns exemplos utilizados por instituições brasileiras.

(a) Banco Central do Brasil

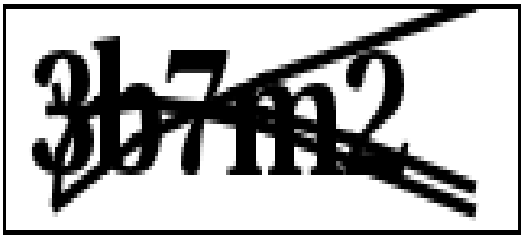
(b) Instituto Nacional do Seguro Social (INSS)

(c) Banco Itaú



(d) Ministério da Educação (MEC)

(e) Ministério Público do Estado do Rio de Janeiro (MPRJ)



(f) Tribunal de Justiça do Estado de Pernambuco (TJPE)

Figura 2 – Diferentes CPATCHAs de texto em uso por instituições brasileiras. (a) Disponível em: <https://www3.bcb.gov.br/faleconosco/#/login/consultar>. Acesso em: 23/06/201. (b) Disponível em: <https://cadsenha.inss.gov.br/cadsenha-internet-web/faces/pages/segurado/entradaNitNb.xhtml>. Acesso em: 23/06/201. (c) Disponível em: <https://bankline.itau.com.br/boletos-novosite/simulador_etapas_boletos_02.htm>. Acesso em: 23/06/201. (d) Disponível em: <http://painel.mec.gov.br/painel/captcha>. Acesso em: 23/06/201. (e) Disponível em: <http://www.mprj.mp.br/comunicacao/ouvidoria/consulte-a-sua-comunicacao>. Acesso em: 23/06/201. (f) Disponível em: <https://srv01.tjpe.jus.br/consultaprocessualunificada/>. Acesso em: 23/06/201.

3 Redes Neurais

Neste capítulo introduziremos redes neurais como uma forma genérica para escrever famílias de funções. Os conceitos de composição, tipos de camadas e arquiteturas são introduzidos. Daremos um enfoque aos dois tipos comuns de camadas que foram usados no presente trabalho. Posteriormente, detalhes técnicos do treino de redes neurais são explorados.

3.1 Introdução

Dentro do campo de Inteligência Artificial, aprendizado de máquina é uma abordagem algorítmica para inferir regras em um problema a partir de exemplos. Uma categoria especial é a de aprendizado supervisionado, onde os exemplos constituem-se de um *elemento* em um domínio conhecido e um *rótulo* associado. O objetivo é deduzir abstrações que permitam relacionar elementos e rótulos. Nesse contexto, uma rede neural[1] é, em última análise, uma forma genérica de escrever funções[2] sobre relações elemento-rótulo. Dada uma métrica para a estimativa dos erros cometidos por uma relação inferida, o desafio de inferir regras de associação ótimas se torna um problema de encontrar uma função que minimiza nossa estimativa de erro. O adjetivo '*neural*' advém da inspiração em funções biológicas que historicamente influenciaram e ainda influenciam essa forma de exprimir relações. Em resumo, redes neurais são um conjunto de técnicas inspiradas em processos cognitivos desempenhados pelo sistema nervoso que fornecem uma maneira genérica de descrever famílias de funções.

De forma mais específica, dado um conjunto de exemplos $D = \{(x, y)\}$, onde x pertence algum domínio conhecido e y um rótulo associado, desejamos encontrar a função $\hat{y} = f(x)$, de tal modo que $\hat{y}$ seja o mais similar possível à y. Por '*mais similar o possível*' entende-se que conhecemos uma estimativa para os erros, também referida como função custo, que é tão menor quanto melhor for a aproximação dada por $f(x)$, e é normalmente representada como $J(y, \hat{y})$. Formalmente, desejamos encontrar f^* tal que

$$f^* = \min_f J^{(D)} = \min_f \langle J(y, f(x)) \rangle_D, \tag{3.1}$$

onde $\langle \ldots \rangle_D$ representa o valor esperado no conjunto D.

Dada uma família de funções $f^\Theta : x \to y$ definida por uma rede neural e parametrizadas por Θ, podemos vasculhar o espaço de busca induzido, $\{\Theta\}$, para encontrar

[1] Apesar de introduzidas aqui como um algoritmo supervisionado, redes neurais podem ser aplicadas de forma efetiva à problemas não supervisionados.

[2] Funções estão sendo usadas aqui com um sentido mais relaxado do que o usualmente utilizado na matemática.

um função que satisfaça alguma propriedade de interesse. Em particular, no caso de aprendizado de máquina, estamos interessados em encontrar o parâmetro Θ^* tal que:

$$\Theta^* = \min_{\Theta} \langle J(y, f^{\Theta}(x)) \rangle_D. \tag{3.2}$$

A versatilidade desse formalismo nos permite explorar diferentes tipos de estruturas relacionais. Em particular, relações hierárquicas podem ser emuladas utilizando-se composição de funções. Essa abordagem nos permite construir redes neurais mais complexas e expressivas a partir de unidades básicas mais simples. Neste caso, podemos reescrever a função f^{Θ} como:

$$f^{\Theta}(x) = f_1^{\Theta_1}(f_2^{\Theta_2}(f_3^{\Theta_3}(\dots(x)))), \tag{3.3}$$

sendo $\Theta = (\Theta_1, \Theta_2, \Theta_3, \dots)$ o parâmetro composto por cada um dos parâmetros individuais. Quando compomos funções desta forma, é comum nos referirmos à cada função $f_i^{\Theta_i}$ como sendo a *i-ésima camada* da rede neural, sendo as camadas para além da mais externa também conhecidas como *camadas escondias* (em inglês *hidden layers*). A *profundidade* da rede é uma referência à quantidade de funções internas usadas na composição. Diferentes tipos de funções definem diferentes tipos de transformações, as quais nos referimos como *tipo da camada*. A especificação de todas as camadas em uma rede neural é o que chamamos de *arquitetura* da rede. A seguir vamos explorar dois tipos de camadas que foram utilizados no presente estudo.

3.2 Camadas Densas

Camadas *densas* ou totalmente conectadas definem uma transformação afim entre o conjunto de entradas e saídas. Tipicamente, após a transformação afim, segue-se a aplicação de uma função não linear elemento-à-elemento, conhecida como *função de ativação*, permitindo a expressão de relações mais complexas. As camadas densas são biologicamente inspiradas no mecanismo de comunicação dos neurônios, onde a diferença de potencial elétrico exprimida nas sinapses do axônio é proporcional, em alguma medida, à soma das diferenças de potenciais experimentadas nos dendritos (20). As conexões de entrada em um neurônio definem sua regra de ativação, ou seja, quais outros neurônios devem estar ativos (e em que intensidade) para que haja uma transição de estado. Como veremos mais adiante, camadas totalmente conectadas tentam emular o comportamento de vários neurônios simultaneamente.

De maneira mais formal, seja x um vetor no conjunto de entrada, a relação expressa por uma camada densa é dada por:

$$f^{W,b}(x) = act(W \odot x + b) \tag{3.4}$$

onde b é um vetor, referido como *viés* (ou, no inglês, *bias*), W uma matriz de transformação, act uma função de ativação e $\odot$ é a operação usual de multiplicação de matrizes, definida

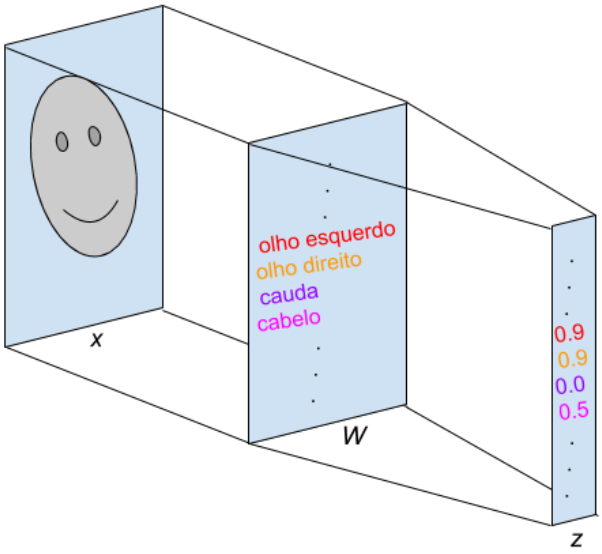
elemento-à-elemento como $[W \odot x]_i = \sum_k W_{ik} * x_k$. Diferentes funções de ativação expressam relações distintas sob um vetor de entrada $z = \{z_i\}$. Dentre os exemplos mais conhecidos na literatura, ressaltamos a função sigmoide, $\sigma(z)_i = \frac{1}{1+\exp(-z_i)}$, que mapeia os elementos de saída no intervalo $\Re[0, 1]$, a função de retificação linear (relu), $relu(z)_i = max(0, z_i)$, e a função máximo atenuado (softmax), $\sigma(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$, que possui a interessante propriedade $\sum_i \sigma(z)_i = 1$, sendo usualmente utilizada para expressar distribuições de probabilidade.

Podemos interpretar a transformação expressa na equação 3.4 como uma projeção do exemplo x sob um espaço de características que descrevem uma relação que desejamos expressar. De fato, se reescrevermos a *i-ésima* linha de W como $\alpha_i \hat{w}_i$, onde $\hat{w}_i$ é um vetor normalizado e α_i a constante de normalização, podemos interpretar cada uma das saídas de uma camada densa (antes da função de ativação) como uma série de projeções balanceadas $\alpha_i \hat{w}_i \odot x + b_i$ ($\odot$ aqui é o produto interno usual). Em outras palavras, a contribuição da *i-ésima* saída da camada é dada pela importância α_i dessa característica multiplicada por o quanto do exemplo é composto por essa característica, $\hat{w}_i \odot x$, adicionado de um viés b_i que independe do exemplo em questão. Assim, para redes densas, determinar os parâmetros ótimos ser entendido como encontrar projeções do espaço de entrada em um conjunto de características que sejam pertinentes ao problema. Neste sentido, cada saída de uma camada densa pode ser interpretada como um neurônio, que exprime em sua saída uma soma balanceada dos estímulos de entrada. Quando compomos camadas densas expressamos características que dependem de outras características. Durante o aprendizado, esperamos que essas projeções sejam descobertas de forma automática pelo modelo. Um desenho esquemático de como essas projeções funcionam pode ser visto na figura 3. O número de parâmetros em uma camada densa é dado pelo produto entre tamanho do exemplo de entrada e a quantidade de características.

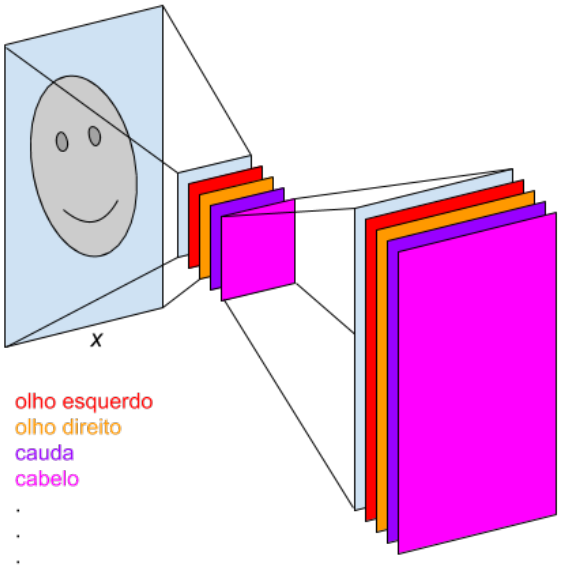
3.3 Camadas Convolucionais

Camadas *convolucionais* (21, 22) são similares às densas: definem uma transformação afim representada por uma matriz, seguida, possivelmente, de uma função de ativação. Diferem, entretanto, no tipo de operação matricial utilizada, trocando-se o produto usual de matrizes por uma operação convolucional. Como vimos na seção anterior, cada saída de uma camada densa pode ser interpretada como uma coleção de projeções independentes sob um espaço de características. Em camadas convolucionais, o mesmo operador projeção é reutilizado em diferentes subespaços do conjunto de domínio, varrendo exemplos em busca de padrões específicos, característica usualmente referida como *compartilhamento de parâmetros*. O reuso promove a vantagem de reduzir consideravelmente o espaço de busca $\{\Theta\}$ (Uma imagem ilustrativa do funcionamento dessas camadas pode ser visto na figura 3). O número de parâmetros nesse caso é dado pelo produto entre o tamanho da projeção

Figura 3 – Imagem ilustrativa das possíveis projeções aprendidas em uma camada (a) densa e (b) convolucional em um problema de reconhecimento de faces.



(a)



(b)

Em (a) as projeções atuam sobre todo o exemplo de entrada, mapeando-os em um espaço de importâncias de cada característica. Note que as projeções densas são fixas, no sentido que expressamos '*olho direito*' em uma posição específica. Em (b) expressamos projeções menores que '*varrem*' o exemplo de entrada buscando por uma característica em particular, projetando o exemplo em canais de características. Note que, neste caso, procuramos por uma característica que esteja presente em qualquer lugar da imagem, com os sinais de resposta registrados no canal respectivo. (Fonte: elaborado pelo autor.)

e o número características. Camadas convolucionais são especialmente efetivas quando os elementos do domínio possuem alta localidade, como é o caso de imagens (cada pixel é altamente correlacionado com seus vizinhos em imagens naturais) e séries temporais periódicas (o valor da série do tempo t é correlacionado com o valor da série em $t + \tau$, onde τ é o período da série). Tipicamente, mais de uma projeção é encapsulada em um mesmo tensor de transformação, sendo a matriz de cada projeção referida como núcleo. O resultado da convolução do núcleo com o exemplo de entrada é tipicamente referida como *canal*.

Matematicamente, dado um tensor x de ranque 3[3], a atuação da camada de convolução pode ser escrita como

$$f^{W,b}(x) = act(W \otimes x + b) \tag{3.5}$$

onde W é um tensor de transformação que encapsula a informação de diferentes projeções, b é o vetor de viés, de dimensão igual ao número de canais de saída, act uma função de ativação e $\otimes$ é a operação de convolução, definida elemento-à-elemento por

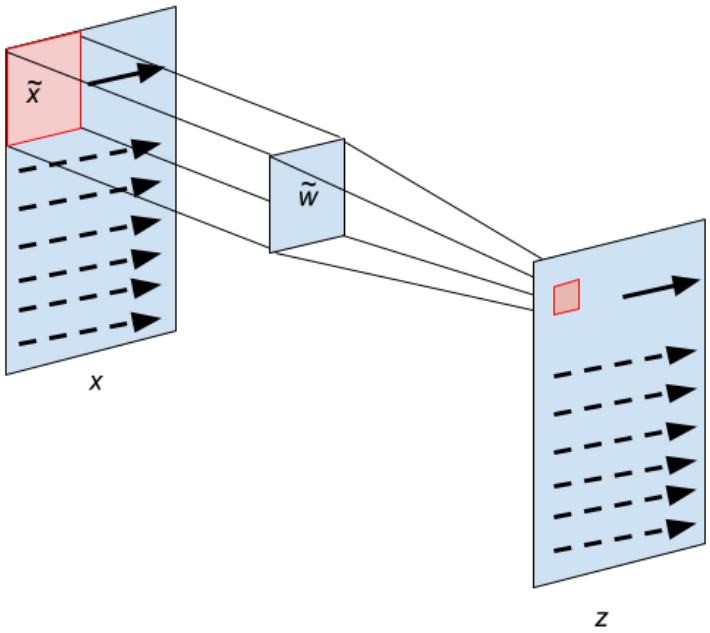
$$[W \otimes x]_{ijd} = \sum_{c} \sum_{m=i'-k_i}^{i'+k_i} \sum_{n=j'-k_j}^{j'+k_j} W_{m,n,c,d} * x_{m,n,c} \tag{3.6}$$

onde c (d) se estende por todos os canais de entrada (saída), $2k_i + 1$ ($2k_j + 1$) é o tamanho do núcleo da projeção na direção i (j) e a relação $s_i = i'/i$ ($s_j = j'/j$) define o passo da transformação (em geral, $k_i = k_j = k$ e $s_i = s_j = s$). Uma forma intuitiva de visualizar uma operação convolucional é imaginar que a cada etapa (i, j), cada operador de projeção $\tilde{w}$ no tensor W define, por canal, uma região $\tilde{x}$ no tensor de entrada. O resultado da operação de convolução é dado pelo produto interno usual entre $\tilde{w}$ e $\tilde{x}$. No passo seguinte a operação é repetida até que toda a imagem de entrada seja coberta. A contribuição total para cada canal de saída é dado pela soma das convoluções em cada canal de entrada. Um esquema ilustrativo pode ser visto na figura 4.

Como visto, a operação de convolução atua mapeando uma região proporcional ao tamanho do núcleo no canal de entrada em um único pixel no canal de saída. Dada a propriedade de localidade, essas representações podem adicionar muita redundância de informação no canal de saída. Uma forma de contornar o problema é adicionando uma operação de agrupamento (em inglês *pooling*) após as ativações da camada. Uma forma simples de realizar um agrupamento é aplicar um mapa entre um conjunto de pixels de entrada em apenas um pixel de saída, reduzindo o tamanho da representação e forçando a rede a encontrar representações mais eficientes entre os níveis de abstração. Escolhas usuais são a média, o valor máximo ('*maxpooling*') ou um valor fixo das ativações de entrada.

[3] Por simplicidade, estamos definindo aqui a operação de convolução para imagens coloridas, representadas por tensores de ranque 3. Entretanto, operações de convolução podem ser definidas para outros domínios, como séries temporais (ranque 1), imagens em tons de cinza (ranque 2) e vídeo (ranque 4).

Figura 4 – Exemplo esquemático da execução da operação de convolução em um canal do tensor de entrada x.



No passo i, j, cada operador de projeção $\tilde{w}$ define um campo de atuação $\tilde{x}$. Para o canal z, $z_{i,j} = \tilde{w} \odot \tilde{x}$. O produto interno é realizado transformando $\tilde{w}$ em um vetor linha e $\tilde{x}$ em um vetor coluna. (Fonte: elaborado pelo autor.)

As operações de agrupamento podem ser vistas como um tipo especial de convolução (possivelmente não linear) com parâmetros são fixos.

Camadas de convolução são inspiradas na interpretação vigente do funcionamento do córtex visual de mamíferos. Diferentes camadas hierárquicas são sensíveis a níveis distintos de abstração na representação de imagens: as iniciais apreendem informações de traços simples, posteriormente composições de traços, formas geométricas, objetos complexos e assim por diante. De fato, estudos mais detalhados das projeções hierárquicas aprendidas por redes convolucionais profundas após o treino mostraram similaridades com o seu análogo biológico (23) e crescentes níveis de abstração na representação das imagens (24, 25), iniciando com traços simples e texturas nas camadas iniciais e culminando em representações abstratas para objetos nas finais.

Frisamos que esta não é uma lista exaustiva dos diferentes tipos de camada que podem ser encontrados na literatura. Não listadas aqui, mas que merecem destaque, podemos citar redes de capsula (26), que desempenham operações de convolução e posterior alinhamento das ativações dos filtros, e redes recorrentes (e suas variações), que adicionam correlação temporal entre os exemplos através de mecanismos de memória. Uma revisão histórica da evolução dessas camadas pode ser encontrado em (3). Para uma descrição mais detalhada das arquiteturas descritas nesse capítulo, incluindo limitações, aplicabilidade e

detalhes técnicos, ver (2).

3.4 Aprendizado

Definida a arquitetura da rede neural, isto é, a sequencia das camadas e respectivas funções de ativação, procedemos para o *treino* da rede, que consiste em encontrar os parâmetros ótimos Θ^* como definido na equação 3.2. A forma mais ingenua para procurar o valor ótimo seria utilizar o método do gradiente (ou método do máximo declive), que consiste em atualizar iterativamente Θ na direção contrária à do gradiente da função custo segundo a regra

$$\Theta \leftarrow \Theta - l_r \nabla_\Theta J^{(D)}, \tag{3.7}$$

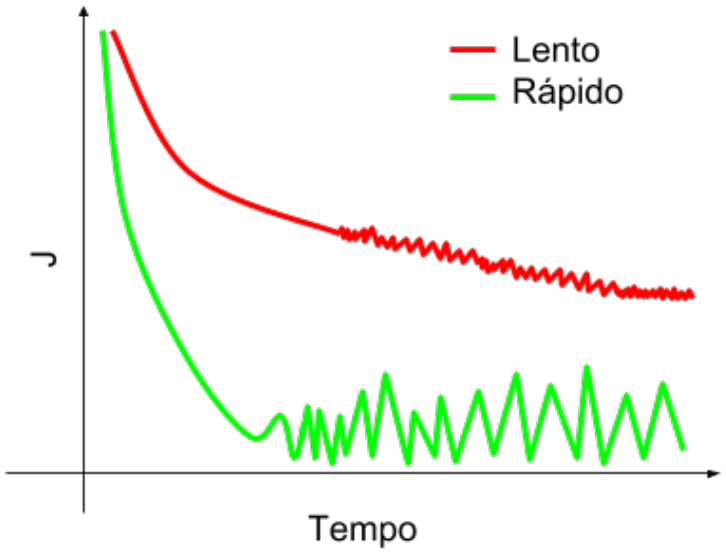
onde l_r é o *hiper-parâmetro de aprendizado* ou taxa de aprendizado, D o conjunto onde a atualização é realizada e ∇_Θ o operador gradiente com respeito aos parâmetros Θ. A evolução de Θ ao longo do treino é referida como *dinâmica do aprendizado*.

O método do gradiente nos garante que, uma vez atingido o mínimo global, o gradiente da função custo é nulo ($\nabla_\Theta J^{(D)} = 0$), e nenhuma atualização posterior se fará necessária. Entretanto, não nos é garantido o recíproco: uma vez que $\nabla_\Theta J^{(D)} = 0$, não podemos afirmar se encontramos um mínimo local, um ponto de sela ou até mesmo um ponto de máximo. Também não é garantido se eventualmente um ponto de gradiente nulo será encontrado. Adicionalmente, a escolha do hiper-parâmetro l_r e do conjunto de atualização D não são especificados pelo método.

Quanto à escolha do conjunto de atualização, podemos escolher dentre atualizar os parâmetros para cada exemplo, para um subconjunto ou para todos os exemplos disponíveis por vez. No primeiro caso, conhecido como gradiente estocástico, temos a vantagem de atualizar Θ sempre que a rede é exposta à um exemplo, o que poderia levar á uma convergência mais rápida. Essa escolha é comumente referida como aprendizado *online*. Entretanto, as flutuações estatísticas de cada exemplo podem gerar uma condição de difícil convergência, com o gradiente da função custo mudando drasticamente de direção a cada atualização. No outro extremo, conhecido como gradiente em lote, D constitui todo o conjunto disponível de treino. Nessa escolha, o gradiente em cada exemplo é acumulado e a atualização é feita pela média dos gradientes em todos os exemplos de treino. Apesar da vantagem da estabilidade estatística, corremos dois riscos: demorar demais para atualizar Θ, o que torna o aprendizado mais lento, e a média sobre o conjunto de treino anular ou diminuir particularidades de subconjuntos específicos. Uma terceira opção consiste em juntar o melhor das duas escolhas anteriores: promover a atualização dos parâmetros à cada N_B exemplos. Tal técnica é denominado atualização em mini-lote ou gradiente em mini-lote. E foi a técnica escolhida no presente trabalho.

Quanto à escolha de l_r temos que buscar um compromisso entre dois extremos. Valores pequenos produzem dinâmicas lentas, onde as atualizações do parâmetros são pequenas e o tempo de treino se estende por diversas iterações. Em alguns casos, a dinâmica pode ficar presa em regiões onde $\nabla_\Theta J^{(D)}$ é pequeno mas o valor do custo ainda é alto. No outro extremo, temos uma dinâmica mais rápida, porem mais instável. Pontos de mínimo podem ser completamente ignorados (fenômeno referido como *overshoot*) e a função de erro tende a exibir flutuações que dificultam a convergência. Uma maneira de contornar o problema é utilizar uma abordagem adaptativa. Uma solução simples é diminuir o valor da taxa de aprendizado ao longo do treino, sendo decaimento linear ou exponencial duas escolhas bastante comuns. Dessa forma, temos um balanço entre rápido aprendizado no início da dinâmica e maior estabilidade próximo ao fim. Outras técnicas incluem estabelecer regras de atualização baseadas no gradiente da função de custo e no momento (taxa de atualização do gradiente). A descrição detalhada dessas técnicas fogem ao escopo deste trabalho. Nos experimentos utilizamos a técnica conhecida como Estimativa Adaptativa co Momento (em inglês, *Adaptive Moment Estimation*), também referida como Adam (27) que foi especialmente desenvolvida para atualizações em mini-lote e utiliza o momento dos gradientes do custo para estabilizar o aprendizado e decaimento da taxa de treino. Um exemplo esquemático de uma dinâmica lenta (l_r pequeno) e rápida (l_r grande) pode ser vista na figura 5.

Figura 5 – Exemplo de comportamento característico da função de custo.



Em vermelho o aprendizado é lento e apresenta flutuações menores. Em verde o custo decai rapidamente, mas apresenta flutuações mais drásticas. (Fonte: elaborado pelo autor.)

Para redes compostas por várias camadas, como descrito na equação 3.3, a atualização dos parâmetros torna-se complicada e potencialmente suscetível a erros numéricos.

Uma forma eficiente de contornar o problema é utilizar a técnica de propagação reversa (em inglês *back-propagation*). Nesta técnica, a computação realizada pela rede é mapeada em um grafo, sendo cada operação realizada na computação expresso como um nó e o fluxo de dados entre duas operações representado por uma aresta direcionada. Durante a fase direta (*feed-foward*) a computação é executada nó-a-nó e os resultados parciais de cada operação armazenados. Durante a fase reversa, as arestas são invertidas e o erro propagado entre os vértices do grafo reverso utilizando-se os valores previamente armazenados e a regra da cadeia para expressar a dependência entre os erros. Durante essa etapa, cada nó que possui um parâmetro a ser aprendido é atualizado. Ver (3) para uma revisão do método.

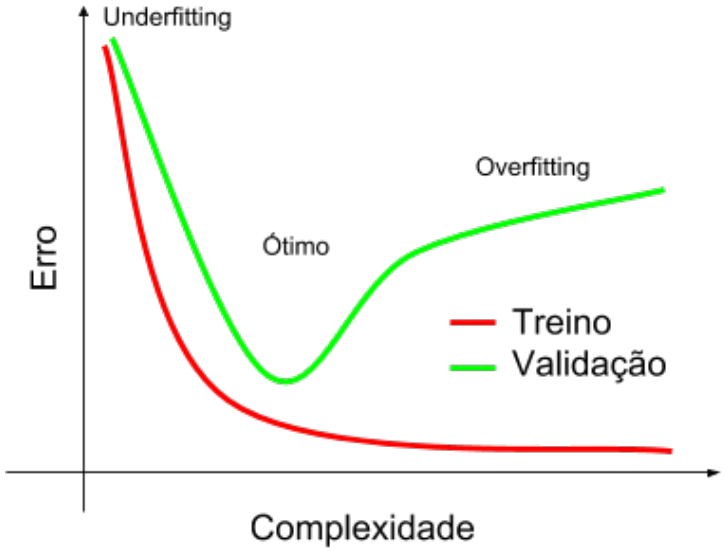
3.5 Regularização

Como visto da seção 3.4, durante a evolução da dinâmica de aprendizado, os parâmetros da rede se adaptam de forma a reduzir o valor esperado da função de custo no conjunto de treino. Entretanto, diminuir o erro nos exemplos vistos não garante que o modelo apresentará a mesma qualidade em novos elementos ausentes durante essa fase. De forma mais precisa, seja U o conjunto de todos elemento-rótulo possíveis, tipicamente temos a disposição um subconjunto $D \subset U$ para otimizar a rede, e utilizamos $J^{(D)}$ como um estimador estatístico para $J^{(U)}$, isto é, supondo que D tenha sido formado por instâncias aleatórias de U, teremos $J^{(D)} \to J^{(U)}$ quando $|D| \to |U|$. Tipicamente, entretanto, temos a situação em que $|D| \lll |U|$. Como o modelo foi otimizado neste subconjunto, o valor esperado da função custo em D se torna um estimador enviesado para o erro esperado em exemplos não vistos. Uma maneira simples e eficiente de estimar a capacidade de generalização modelo é criar a cobertura exata (D_{tr}, D_{val}) para D, treinar o modelo em D_{tr} e acessar sua qualidade em D_{val}. Esta técnica é conhecida como *hold-out*, e $J^{(D_{val})}$ uma estimativa para o poder de generalização do modelo (28).

Um outro problema recorrente no treino de modelos de aprendizado de máquina é a adequação do modelo ao conjunto de treino. Em um extremo, modelos de baixa expressividade podem não conseguir capturar a complexidade das relações exemplo-rótulo. Esse fenômeno é conhecido como *underfitting*. No outro limite, a medida que aumentamos a expressividade (mais camadas, mais projeções, etc.), aumenta também capacidade de apreender relações mais complexas entre exemplos e rótulos. Entretanto, modelos com maior expressividade tendem a ser mais suscetíveis à '*decorar*' exceções e superestimar particularidades do conjunto de treino. Tal fenômeno é conhecido como coadaptação ou *overfitting*. Outra situação em que podemos incorrer em *overfitting* é quando o modelo é excessivamente exposto aos mesmos exemplos durante uma seção de treino. O *underfitting* pode ser detectado diretamente da qualidade do modelo em D_{tr}, quando seu desempenho se mostra aquém do esperado. O segundo pode ser estimado utilizando a decomposição

viés-variância da estimativa do erro, onde entendemos que parte do erro cometido é devido a baixa expressividade e parte por superestimar particularidades do conjunto de treino. Para tal, a qualidade do modelo em D_{tr} e sua capacidade de generalização (estimada em D_{val}) são comparadas e a diferença entre as duas servem para estimar o compromisso entre viés e variância. Um exemplo ilustrativo do comportamento do erro nesses dois conjuntos pode ser visto na figura 6.

Figura 6 – Exemplo ilustrativo de *overfitting* e *underfitting*.



Em verde vemos que o erro estimado no conjunto de treino se comporta como uma função decrescente da complexidade do modelo. Em vermelho, o erro estimado o conjunto de validação mostra que existe um valor ótimo para o poder de generalização. Fixada a complexidade do modelo, experimenta-se um comportamento similar quando aumentamos a exposição aos exemplos no conjunto de treino. (Fonte: elaborado pelo autor.)

Para minimizar o efeito de coadaptação dos parâmetros, podemos impor condições extras durante o processo de aprendizado. Uma das condições mais usuais é aplicar ao vetor de parâmetros Θ alguma restrição. Podemos, por exemplo, adicionar à função custo um termo proporcional à norma do vetor Θ, na forma

$$\tilde{J} = J + \lambda L_n(\Theta), \tag{3.8}$$

onde $\tilde{J}$ é o custo utilizado durante a minimização, $L_n(\Theta) = \sqrt[n]{\Theta^n}$ é a norma n do vetor e λ o hiper-parâmetro de regularização ou constante de normalização. Desta forma penalizamos parâmetros que '*aprendam*' demais uma determinada característica. Apesar de efetivo, o uso de regularização baseado em normas adiciona dois novos hiper-parâmetros: a constante de normalização e o tipo de norma que devemos usar. Outra forma de regularização comum na literatura é o *dropout* (29). Nesta técnica, a saída i de uma camada é removida da computação (atribuída o valor zero) com uma probabilidade p, hiper-parâmetro da técnica.

Intuitivamente, o uso do *dropout* durante o treino força a rede a aprender representações redundantes e mais robustas à detalhes, dado que a cada nova iteração uma característica aprendida anteriormente pode estar ausente.

3.6 Redes Neurais e CAPTCHAs

Como visto na secção 2.2, CAPTCHAs de texto podem ser vistos como um problema de extração de texto em imagens, sendo assim uma generalização para o problema de OCR e um subproblema de localização de texto em imagens (identificar, se existir, a localização de todos os textos em uma imagem). É preciso ressaltar, entretanto, que nesses HIPs as imagens são especialmente desenvolvidas para serem de difícil solução para computadores e preferencialmente fáceis para seres humanos. Assim, algoritmos usuais de OCR tendem a demonstrar baixo desempenho na solução desses desafios. Vamos separar as abordagens para o problema em dois grandes grupos: abordagens *informadas* e não *informadas*.

Em abordagens informadas o autor do ataque estuda os efeitos e o alfabeto utilizado na composição do desafio e confecciona um pipeline para preprocessamento e segmentação dos caracteres componentes do token. Esta etapa envolve o uso de heurísticas e é extremamente dependente da regularidade das imagens[4] e da habilidade do atacante para que bons resultados sejam alcançados. Após a segmentação, o desafio se resume à reconhecer corretamente cada símbolo. Um dos trabalhos pioneiros na quebra de CAPTCHAs de texto foi conduzido por (10) utilizando-se essa abordagem. Inicialmente as imagens foram preprocessadas e segmentadas. Para o reconhecimento foram usadas redes convolucionais de duas camadas, alcançando acurácias entre 10% e 50% (fortemente dependente da qualidade do preprocessamento). Bursztein *et al.* formalizaram ainda mais o pipeline de processamento lançando a ferramenta *Decaptcha* (19), que permite explorar e testar etapas do processo de quebra especificando-se técnicas de preprocessamento, segmentação, pos-segmentação, reconhecimento e aprimoramento pós reconhecimento. Com o auxilio dessa ferramenta, os autores relataram modelos com acurácias mais baixas em trono de 20%, chegando à totalidade de acertos em desafios mais simples. Entretanto, como sugerido pelos próprios autores, a ferramente utiliza na fase de reconhecimento algoritmos com baixo poder de expressividade (k-vizinhos mais próximos e máquinas de vetores de suporte), se comparados com as técnicas modernas. Se olharmos apenas para o desafio pós segmentação, a extração de texto de CAPTCHAs de resume à um problema de reconhecimento de caracteres, e nesse campo o estado da arte é dominado por redes neurais. De fato, no repositório do MNIST[5] (30), por exemplo, redes convolucionais pontuam entre as melhores taxas de

[4] Isso é tipicamente verdade quando a imagem é gerada automaticamente por um computador. Neste caso, técnicas de engenharia reversa podem guiar o desenvolvimento de heurísticas.

[5] O MNIST (Modified National Institute of Standards and Technology database) é um repositório aberto contendo 70000 imagens de dígitos escritos à mão e usualmente usado como benchmark para técnicas de OCR. A tarefa consiste em reconhecer corretamente o número codificado na imagem. No sítio do

acerto, sendo o melhor modelo registrado o desenvolvido por Cireşan *et al.*, que alcançou 99.77% de acerto e é baseado na média do voto de 35 redes convolucionais independentes (31). Para CAPTCHAs de texto formados por imagens reais, as diversas condições de aquisição (iluminação, ângulo, escala, etc.) degradam as vantagens de um pipeline único de preprocessamento. Mesmo neste caso, redes neurais ainda são capazes de demonstrar bom desempenho. Netzer *et al.* propuseram o repositório SVHN[6] como benchmark para esse tipo de desafio (32). Partindo das imagens já segmentadas, os autores foram capazes de obter precisões em torno de 90% utilizando-se abordagens não supervisionadas de redes neurais. O resultado foi posteriormente refinado utilizando-se redes convolucionais profundas e treino supervisionado, alcançando 94.5% de precisão (33).

Quanto as abordagens não informadas, o modelo deve aprender de forma autônoma como localizar, segmentar e reconhecer os caracteres em uma imagem, minimizando a interferência direta humana. Entretanto, inferir esse nível de abstração apenas baseado em exemplos pode requerer quantidades massivas de dados anotados. Utilizando uma abordagem não informada, Goodfellow *et al.* foram capazes de obter 99.8% de acerto nos textos da primeira versão do reCAPTCHA (12). Os autores ainda conduziram um estudo no repositório SVHN, obtendo acertos de 97.84% por caractere e 96% para o token completo. Entretanto, a base de treino utilizada era formada por 600 mil imagens para o SVHN (toda a base de teino) e mais de um milhão de imagens para o reCaptcha. Em um estudo similar ao nosso, Pinto alcançou $76,6\%$ de precisão na quebra CAPTCHAs utilizando redes neurais profundas. Contudo, foi necessária uma base de treino com 180 mil imagens e placas de processamento gráfico de última geração, sendo o treino realizado em sistemas de computação sob demanda (34). Mais recentemente, foi investigada uma alternativa para diminuir volume de dados necessário para esse tipo de abordagem através de uma nova arquitetura denominada Redes Corticais Recorrentes (do inglês, Recurrent Cortical Network) (13). Os atores do estudo relatam resultados próximos ao estado da arte utilizando-se apenas algumas milhares ou até mesmo centenas de imagens de treino. O diferencial dessa arquitetura são conexões extras adicionadas entre as camadas que permite um fluxo mais complexo da informação. Aplicada à quebra de CAPTCHA, os autores treinaram esse tipo de rede em imagens contendo diferentes fontes tipográficas para o mesmo alfabeto. Durante a validação a arquitetura de mostrou capaz de generalizar e abstrair as transformações aplicadas em CAPTCHAs sintéticos, sendo capaz de reconhecer os símbolos mesmo após as deformações. Um ponto negativo, entretanto, é o tempo de treino dessa nova arquitetura. As mensagens trocadas entres as camadas e o algoritmo

repositório existe uma tabela comparativa da precisão de diversos estudos publicados utilizando-se diferentes técnicas ao longo dos anos.

[6] SVHN (Street View House Numbers), composto por mais de 600000 fotos de fachadas de construções contendo números. O desafio consistem em reconhecer corretamente as algarismos codificados na imagem. O repositório possuem dois formatos para os mesmas imagens: a) Caracteres segmentados; e b) Apenas a região contendo os números.

proposto para supressão de característica limitam a capacidade de paralelização e tornam a execução mais lenta. De fato, há uma nota no repositório dos autores informando que o treino em 1000 imagens do desafio MNIST (onde o algoritmo se aproxima do estado da arte nesse desafio) pode levar horas em múltiplas CPUs[7]. Adicionalmente, os autores alegam que fizeram ajustes nos filtros convolucionais e nas fontes de treino para cada aplicação, o que pode ser interpretado com uma forma de adicionar viés ao modelo (algo como uma abordagem semi-informada).

Apesar da qualidade dos resultados encontrados na literatura, os requisitos desses sistemas tornam-os proibitivos para o uso em computadores comuns. Na tabela 1 apresentamos um breve comparativo dos requisitos de alguns dos estudos selecionados. Essas restrições limitam o acesso dessa tecnologia à ambientes com menor poder de processamento ou restrições orçamentárias. É particularmente proibitivo para o estudante médio brasileiro, o que pode gerar uma defasagem de aprendizado tecnológico, e para pequenas empresas experimentarem soluções inovadoras baseadas nessas descobertas. Assim, o objetivo deste trabalho é propor um abordagem construtiva para a experimentação de redes neurais profundas focada no problema de **quebra não informada de CAPTCHAs de texto**. Pretendemos mostrar que é possível alcançar resultados próximos ao estado da arte a partir da investigação cautelosa do comportamento dos modelos durante a dinâmica de treino, com bases de treinos menores e arquiteturas mais simples.

Referência	Limitação
Netzer *et al.* (32)	600 mil imagens, 500 filtros convolucionais 8x8.
Sermanet *et al.* (33)	16 filtros 5x5, 512 filtros 7x7, 2 camadas densas de 4000 entradas.
Goodfellow *et al.* (12)	De 9 a 11 camadas convolucionais (8-192 filtros), camada densa com mais de 3 mil entradas, mais de 500 mil exemplos.
Pinto (34)	180 mil exemplos, quatro camadas convolucionais com 64, 128, 256 e 512 canais, duas camadas densas com mais de 4000 entradas.
Dileep *et al.* (13)	Treinameto de difícil paralelização. Muitas horas de treino para alcançar resultados satisfatórios.

Tabela 1 – Comparação entre os requisitos dos modelos encontrados na literatura.

[7] Fonte: <https://github.com/vicariousinc/science_rcn>. Acesso em: 23/06/2018.

4 Modelagem

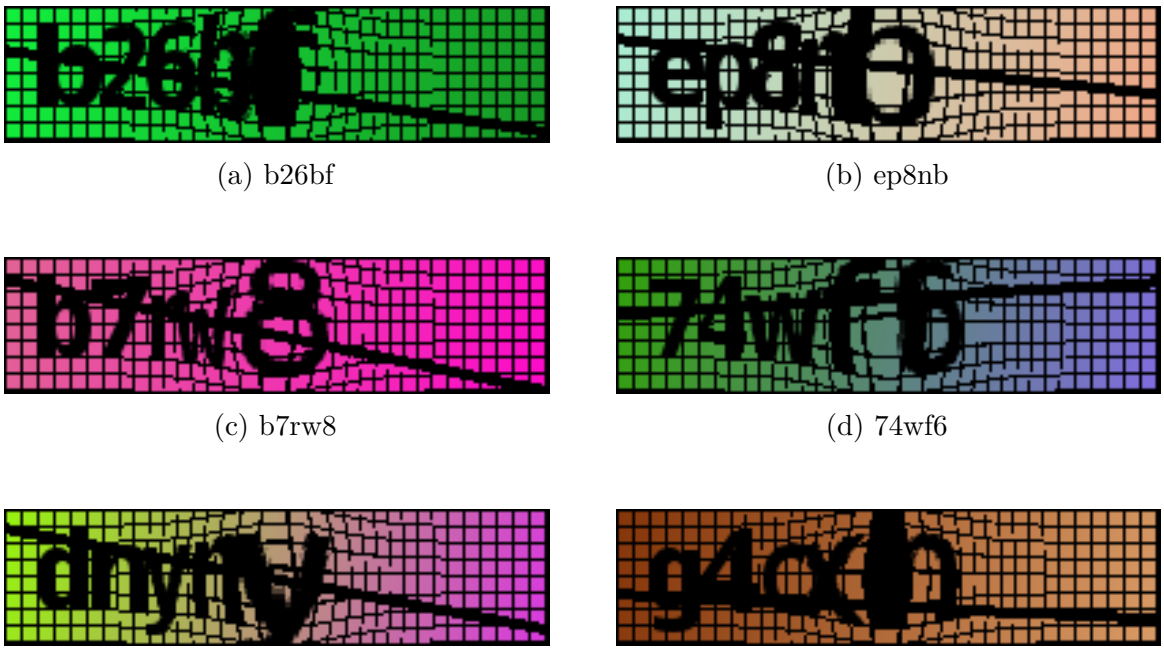
definição das redes definição da nomenclatura

5 Metodologia

Neste capítulo os detalhes envolvidos na geração das imagens de CAPTCHAs são expostos. Em seguida, definimos as grandezas de interesse que nos permitem acessar a qualidade dos modelos treinados. Por fim, as etapas de treino e validação são formalizadas.

5.1 Geração dos CAPTCHAs

Todos os exemplos foram gerados utilizando a biblioteca SimpleCaptcha(35). Ao total, foram gerados 30000 pares imagem-token. As sequências de texto possuem comprimento fixo em 5 e os caracteres foram sorteados de forma independente a partir do alfabeto ordenado $\Sigma = \{0123456789abcdefghijklmnopqrstuvwxyz\}$ de 36 símbolos. Dentre os efeitos escolhidos para as imagens, enfatizamos as variações nas cores de fundo, desenho de grades, adição de linhas aleatórias e deformação em explosão, que são técnicas efetivas para construir desafios fáceis para humanos e difíceis para computadores, de acordo com estudo conduzido por (10). Uma pequena amostra das imagen-token geradas pode ser vista na Fig.7.



(a) b26bf (b) ep8nb

(c) b7rw8 (d) 74wf6

(e) dnyny (f) g4cxh

Figura 7 – Exemplos de CAPTCHAs gerados e seus respectivos tokens.

Considere $D = (x, y)$ o conjunto formado por todos os pares de imagem-token gerados. Cada exemplo é formado por um tensor imagem x e uma matriz y representando

o token, de dimensões $(200, 50, 3)$ e $(5, 36)$, respectivamente. Cada entrada $x_{ijk} \in \Re[0, 1]$ representa a intensidade do pixel localizado na posição (i, j) e canal k. A entrada $y_{ij} \in \Re[0, 1]$ foi codificada utilizando-se a técnica *one-hot encoding*, onde i representa a posição na sequência u e j o índice no vocabulário do caractere nessa posição, de modo que

$$y_{ij} = \begin{cases} 1, & \text{se } u_i = \Sigma_j \\ 0, & \text{caso contrário,} \end{cases} \tag{5.1}$$

ou, de forma mais compacta, $y_{ij} = \delta_{u_i,\Sigma_j}$, onde $\delta_{m,n}$ é o delta de Kronecker. Essa codificação nos permite interpretar y como sendo uma distribuição de probabilidade. Se imaginarmos z como uma variável aleatória descrevendo a i-*ésima* entrada na sequência, e $p(z|x)$ como a probabilidade de na posição i da sequência termos o caractere z dada a imagem x, para os exemplos gerados, o conhecimento da imagem define automaticamente qual o caractere em cada posição com 100% de certeza, ou seja, se c é o caractere de fato na sequência $(c = u_i)$, teremos $p(z = c|x) = p(z = u_i|x) = 1$ e, caso contrário, $p(z \neq c|x) = p(z \neq u_i|x) = 0$. Ou, utilizando o delta de Kronecker, $p(z|x) = \delta_{z,u_i}$. Definindo $ord(c)$ como o índice do caractere c no alfabeto Σ (isto é, $c = \Sigma_{ord(c)}$), da equação 5.1 vem que, $p(z|x) = \delta_{z,u_i} = \delta_{u_i,z} = \delta_{u_i,\Sigma_{ord(z)}} = \delta_{u_i,\Sigma_j} = y_{ij}$.

D foi reordenando de forma aleatória e separado em dois subconjuntos: o conjunto de treino, D_{tr}, com $\frac{2}{3}$ do total de pares, e o conjunto de validação, D_{val}, com os demais exemplos. Devido a natureza combinatória do espaço de imagens possíveis (36^5 tokens $\times$ 255^3 cores de fundo $\times$ espaço de todas as pertubações possíveis), D é muito menor do que o conjunto de todas as possíveis composições imagem-token e, supondo que seus elementos tenham sido construídos ao acaso, é virtualmente improvável que existam exemplos repetidos nesse conjunto.

5.2 Treino e Validação

Uma época de aprendizado consiste em duas etapas: treino e validação. Durante o treino, um subconjunto $D_{batch} \subset D_{tr}$ é sorteado ao acaso. Os parâmetros da rede são atualizados utilizando o algoritmo adaptativo Adam com os parâmetros sugeridos em(27) e taxa de aprendizado l_r de forma a minimizar o erro nesse subconjunto. A etapa de treino se encerra após $|D_{tr}|/|D_{batch}|$ atualizações. Na etapa de validação, as grandezas de interesse são calculadas para D_{tr} e D_{val} e salvas para posterior análise.

As redes foram inicializadas segundo a heurística proposta em (36) e as épocas de aprendizado se sucedem até que o critério de parada se alcançado. Escolhemos como critério de parada uma heurística semelhante às definidas por (37). A dinâmica de aprendizado leva no mínimo T^{min} e no máximo T^{max} épocas. Após a etapa de validação o aprendizado é interrompido prematuramente se um dos dois critérios forem verificados: o custo calculado

em D_{val} na época atual ultrapasse em mais de 10% o menor valor de $J^{(D_{val})}$ nas épocas anteriores; o valor de $J^{(D_{tr})}$ atual seja maior do que 97% da média dos cinco últimos custos nesse conjunto. Ou seja, o treinamento é parado prematuramente se for detectado *overfitting* ou se não houver melhora significativa em relação aos últimos valores.

Para selecionar o valor do hiper-parâmetro l_r, foram realizados experimentos com diferentes valores fixos de l_r e $T^{max} = 10$ épocas para cada arquitetura. A partir dos experimentos, selecionamos manualmente os limites inferior e superior, $(l_r^-,\ l_r^+)$, que apresentam o melhor compromisso entre velocidade de aprendizado e estabilidade (vide seção 3.4). Os experimentos posteriores foram realizados com uma taxa de aprendizado segundo a equação:

$$l_r(t) = l_r^+ + (l_r^- - l_r^+) * \frac{t}{T_{max} - 1}, \tag{5.2}$$

onde $t = 0, 1, 2, \ldots, T_{max} - 1$, onde t é a época atual.

Todos os experimentos realizados nesse trabalho foram executados ema máquina com processador Intel® Core™i5-6200U, 8gb de RAM e placa de aceleração gráfica NVIDIA® 920M, utilizando a biblioteca de código aberto Tensorflow (38).

5.3 Métricas

No capítulo de fundamentação teórica de redes neurais (sec. 3), vimos que cada arquitetura é parametrizada Θ. Mais especificamente, cada uma das arquiteturas utilizadas neste trabalho possui como parâmetros um conjuntos de números reais. Assim, definimos a **complexidade do modelo**, para fins de comparação, como a soma da quantidade de parâmetros de cada camada da arquitetura. Para treinar e acessar a qualidade dos modelos, consideramos as grandezas definidas à seguir. Para todas as definições, considere D um conjunto de exemplos, $(x, y) \in D$ e $\hat{y} = f^{\Theta}(x)$ a distribuição de probabilidade inferida, como descrito anteriormente.

Para acessar o erro cometido pelos classificadores, podemos utilizar a **entropia cruzada** (no inglês *cross entropy*), que pode ser interpretada como uma medida de divergência entre duas distribuições de probabilidade. Assim, o custo associado ao inferir $\hat{y}$ quando a verdadeira distribuição deveria ser y, por caractere, é dado por

$$H_i(y, \hat{y}) = -\sum_j y_{ij} \log_2 \hat{y}_{ij} \tag{5.3}$$

$$= -\sum_j \delta_{u_i,\Sigma_j} \log_2 \hat{y}_{ij} \tag{5.4}$$

$$= -\log_2 \hat{y}_{i\ ord(u_i)} \tag{5.5}$$

onde utilizamos o fato de $\delta_{u_i,\Sigma_j} = 0$ exceto em $j = ord(u_i)$. Em outras palavras, a entropia associada ao classificador da posição i é o logaritmo da probabilidade predita

para o caractere correto nessa posição. Definimos o **custo esperado por caractere** do classificador i no subconjunto D como

$$J_i^{(D)} = \frac{1}{|D|} \sum_{(x,y)\in D} H_i(y, \hat{y}) \tag{5.6}$$

e **custo esperado por token** como a média dos erros em cada posição, ou seja:

$$J^{(D)} = \langle J_i^{(D)} \rangle_{\{i\}}. \tag{5.7}$$

Durante o treino tentaremos minimizar a 5.6 para cada classificador, e o custo total associado aos erros cometidos pelo modelo é calculado pela equação 5.7.

Uma estimativa da probabilidade de **acerto por caractere** é dada pela acurácia de cada classificador, isto é, o número de acertos do caractere i no conjunto D, N_i, normalizado pelo tamanho do conjunto D:

$$\hat{p}_i^{(D)} = acc_i^{(D)} = \frac{N_i}{|D|} \tag{5.8}$$

Supondo que os $\hat{p}_i^{(D)}$ sejam independentes entre si, podemos definir uma estimativa para a **probabilidade de acerto do token** como o produto das probabilidades individuais:

$$\hat{p_u}^{(D)} = \prod_i \hat{p}_i^{(D)}. \tag{5.9}$$

Adicionalmente, definimos a **acurácia do modelo por token** como sendo o número de acertos na predição do token, N_u, normalizado pelo tamanho do conjunto:

$$acc_u^{(D)} = \frac{N_u}{|D|}. \tag{5.10}$$

Chamamos a atenção de que as equações 5.9 e 5.10 não necessariamente representam a mesma grandeza, fornecendo duas estimativas diferentes para a qualidade do modelo.

Quanto ao tempo da dinâmica de aprendizado, definimos, para cada época t, o *tempo de treino* por época, $\tilde{\tau}$, como sendo o tempo gasto durante a fase de treino nessa época e o **tempo total** de uma época, τ com a soma dos tempos gastos com treino e validação. Adicionalmente, definimos o **tempo de convergência**, T, como o tempo decorrido até que o critério de parada tenha sido alcançado.

6 Resultados

Na tabela 2 vemos as especificações dos modelos treinados durante a experimentação com seus respectivos tamanhos e duração média do treino em uma época e a duração total de uma época. A partir da tabela podemos ver que o tempo consumido pelas redes é proporcional ao tipo de operações envolvidas e ao tamanho das redes. Sendo mais sensível ao primeiro. As arquiteturas (c) e (g), por exemplo, possuem aproximadamente o mesmo número de parâmetros, diferenciando-se nos tipos de operações e tipos de camadas. Em particular, percebemos que o uso do agrupamento maxpooling triplicou o tempo médio de execução de uma época nas arquiteturas (c) e (d). Por outro lado, a regularização com dropout teve pouco impacto nas estimativas de tempo das arquiteturas estudadas. Note ainda que (dentro da margem de erro), as redes (h) e (i) tem a mesma requisição de tempo, apesar de a última possuir menos parâmetros. Chamamos a atenção para a significativa variância entre as estimativas de duração para a mesma rede em épocas diferentes ($\approx 33\%$), independentemente da arquitetura. Essas flutuações podem estar relacionadas ao fluxo de dados intenso entre as unidades de processamento (CPU e GPU) durante os experimentos, acentuados pela restrição de memória que obriga que apenas parte dos exemplos de treino estejam prontamente disponíveis para o processamento, restando aos demais serem armazenados no disco rígido do computador. Assim, a hierarquia entre as memórias apresenta-se como um elemento importante à ser considerado em ambientes mais restritos. Adicionalmente, a escolha das operações e da arquitetura utilizada deve levar em consideração também os impactos na duração do treino caso o tempo seja um fator limitante.

Na figura 8 podemos ver a evolução da função custo durante as 10 primeiras épocas para diferentes arquiteturas de rede e valores 10^{-1}, 10^{-3}, 10^{-4} e 10^{-5} para l_r. Para $l_r = 10^{-1}$, vemos que o custo atinge um platô durante as primeiras épocas. Essa estagnação na dinâmica de J é um indicativo de que as correções para os parâmetros da rede cresceram de forma descontrolada. De fato, o algoritmo adaptativo Adam penaliza atualizações drásticas normalizando o valor de atualização de cada parâmetro pela norma do gradiente da função de custo. Assim, se $|\nabla_\Theta J| \to \infty$, as atualizações são nulas ($\Delta\Theta \to 0$). Esse comportamento torna-se mais explicito em (a) e (b), onde podemos ver um aumento no valor de J antes das atualizações cessarem. No outro extremo, $l_r = 10^{-5}$, vemos um comportamento similar durante as épocas iniciais para algumas arquiteturas (c-i). Entretanto, a aparente estagnação nesses casos é devida à lenta convergência induzida por esse valor do hiper-parâmetro, observando-se uma melhoria no valor de J na sequência da dinâmica, exceto em (h) e (i), onde o número de épocas no recorte da dinâmica foi pequeno demais para que fosse possível observar melhoras significativas. O valor $l_r = 10^{-4}$

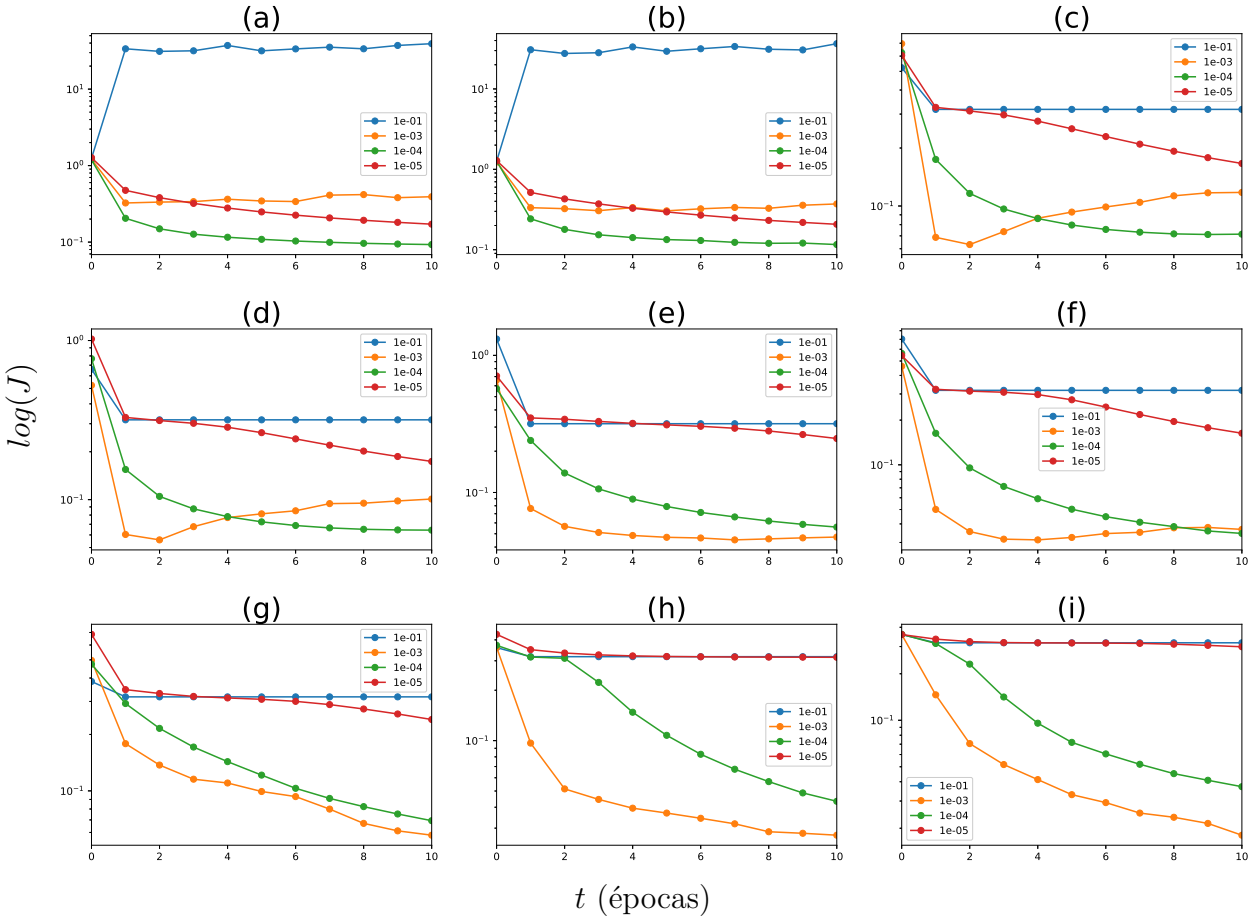
Tabela 2 – Comparação do tempo de treino entre as arquiteturas.

modelo	especificação	tamanho (parâmetros)	$\tilde{\tau}$ (segundos)	τ (segundos)
(a)	RMch	5.4 10^6	51 ($\pm$16)	68 ($\pm$22)
(b)	RMchD	5.4 10^6	59 ($\pm$19)	82 ($\pm$27)
(c)	C5o6RMch	2.4 10^6	95 ($\pm$31)	124 ($\pm$41)
(d)	C5o6MaxRMch	2.4 10^6	309 ($\pm$102)	391 ($\pm$129)
(e)	C5o6MaxRMchD	2.4 10^6	319 ($\pm$105)	405 ($\pm$134)
(f)	C5o6C5o12 MaxRMchD	4.4 10^6	369 ($\pm$122)	429 ($\pm$142)
(g)	C5o6C5o12 MaxRfl100MchD	2.4 10^6	438 ($\pm$145)	492 ($\pm$163)
(h)	C5o6C5o12C5o36 C5o12MaxRMchD	1.1 10^5	474 ($\pm$157)	540 ($\pm$179)
(i)	C5o6C5o12C5o36 C5o12MaxRfl100MchD	9.2 10^4	476 ($\pm$158)	542 ($\pm$179)

exibe as dinâmicas mais estáveis, onde a função de custo apresenta um comportamento monotonamente decrescente durante toda dinâmica e eventualmente estagnando, como visto em (a-d). Para as demais redes (e-i) nota-se que o valor da função de custo ainda apresentaria melhoras caso continuássemos a evolução por mais épocas. Para $l_r = 10^{-3}$, vemos que a evolução é, de fato, mais rápida nas arquiteturas (c-i). Contudo, esse valor leva a instabilidades (melhor visto em (g)) e eventualmente pioras no valor da função de custo (visto em (c), (d) e (f)). Esse fenômeno de overfitting foi discutido previamente no capítulo 3. Nestes casos, o comportamento pode ser explicado por uma possível memorização dos exemplos de treino, ou seja, esse valor do hiper-parâmetro superestima os erros cometidos. Buscando o compromisso entre velocidade de aprendizado e estabilidade, os limites para valor de l_r foram fixados em 10^{-3} e 10^{-4}.

Na tabela 3 podem ser vistos os valores da função de custo e da acurácia por palavra para os conjuntos de treino e validação na décima época para os dois valores escolhidos da taxa de aprendizado. A partir da tabela podemos notar que o overfitting, estimado como a diferença relativa entre treino e aprendizado ($J^{(D_{val})}/J^{(D_{tr})} - 1$), é menor para o valor 10^{-4} em todas as arquiteturas (última coluna na tabela). Observamos também que o uso do dropout contribuiu de forma significativa para o controle do overfitting. Esse efeito pode ser apreciado comparando-se os pares de rede (a)-(b) e (d)-(e), que possuem mesma arquitetura, diferindo apenas no uso da técnica. No par (c)-(d) podemos constatar que o uso da ativação máxima como critério de pooling contribuiu de forma positiva, porém tímida, no desempenho da arquitetura. Nota-se também que, no geral, a adição de camadas

Figura 8 – Dinâmica inicial da função de custo para diferentes taxas de aprendizado e diferentes arquiteturas.



Nos gráficos apresentamos a função custo na escala logarítmica de modo a facilitar a visualização. Os traços são guias para os olhos. As arquiteturas estão indicadas nos títulos e os valores do hiper-parâmetro de aprendizado na legenda.

convolucionais melhoraram o desempenho das redes ao mesmo tempo em que limitaram a quantidade de parâmetros a serem optimizados, com exceção de (f), onde a camada convolucional com 12 canais aumentou significativamente o espaço de busca. A arquitetura em (g) é similar à em (f), adicionando-se uma camada densa com 100 sinais de saída. Esta adição diminuiu o número de parâmetros, degradando, entretanto, a performance da rede, um indicativo de que a simples adição de camadas não necessariamente se traduz em arquiteturas mais expressivas. A mesma adição for repetida no par (h)-(i). Neste caso, entretanto, não foi observada uma degradação no desempenho da rede (de fato, houve uma pequena melhora). Na última linha da tabela temos a precisão reportada por Pinto (34). Através da comparação com os parâmetros de rede utilizados em seu trabalho podemos demonstrar a importância do estudo comparativo do desempenho das arquiteturas de rede no projeto de redes neurais. No referido trabalho, foram usadas redes com pelo menos 10 vezes mais parâmetros e uma base de treino com 180 mil exemplos (9 vezes maior que a nossa), com o equivalente à uma exposição pelo menos duas vezes maior aos exemplos

de treino (duas vezes o número de épocas, se convertida nossa definição de uma época para a utilizada no estudo). Estão presentes também fortes indicativos de coadaptação nos parâmetros. A acurácia reportada pelo autor é 23% maior no conjunto de treino do que no de validação, a despeito da aplicação de regularização com norma L_2 e dropout de 50%, sendo um indicativo de memorização da base de treino. Fomos capazes de projetar uma rede com mil vezes menos parâmetros e treinada em um conjunto de dados 9 vezes menor e ainda obter resultados similares no conjunto de validação.

Tabela 3 – Comparação entre as arquiteturas na décima época.

modelo	l_r	$J^{(D_{tr})}$	$J^{(D_{val})}$	$acc_u^{(D_{tr})}$	$acc_u^{(D_{tr})}$	$\frac{J^{(D_{val})}}{J^{(D_{tr})}} - 1$
(a)	10^{-3}	$1.72\ 10^{-1}$	$3.92\ 10^{-1}$	0.37	0.20	1.28
	10^{-4}	$4.66\ 10^{-2}$	$9.29\ 10^{-2}$	0.54	0.30	0.99
(b)	10^{-3}	$1.78\ 10^{-1}$	$3.71\ 10^{-1}$	0.39	0.23	1.08
	10^{-4}	$7.51\ 10^{-2}$	$1.16\ 10^{-1}$	0.49	0.35	0.54
(c)	10^{-3}	$1.20\ 10^{-3}$	$1.18\ 10^{-1}$	0.98	0.48	96.64
	10^{-4}	$2.20\ 10^{-2}$	$7.14\ 10^{-2}$	0.76	0.43	2.24
(d)	10^{-3}	$2.13\ 10^{-3}$	$1.01\ 10^{-1}$	0.97	0.53	46.35
	10^{-4}	$2.35\ 10^{-2}$	$6.43\ 10^{-2}$	0.74	0.47	1.73
(e)	10^{-3}	$4.75\ 10^{-3}$	$4.73\ 10^{-2}$	0.94	0.62	8.96
	10^{-4}	$3.50\ 10^{-2}$	$5.59\ 10^{-2}$	0.71	0.55	0.60
(f)	10^{-3}	$2.91\ 10^{-3}$	$3.67\ 10^{-2}$	0.96	0.71	11.61
	10^{-4}	$1.79\ 10^{-2}$	$3.45\ 10^{-2}$	0.84	0.71	0.93
(g)	10^{-3}	$4.20\ 10^{-2}$	$5.80\ 10^{-2}$	0.54	0.45	0.38
	10^{-4}	$5.81\ 10^{-2}$	$6.93\ 10^{-2}$	0.44	0.36	0.19
(h)	10^{-3}	$2.33\ 10^{-2}$	$2.72\ 10^{-2}$	0.80	0.77	0.17
	10^{-4}	$3.98\ 10^{-2}$	$4.34\ 10^{-2}$	0.71	0.68	0.09
(i)	10^{-3}	$1.48\ 10^{-2}$	$1.80\ 10^{-2}$	0.81	0.78	0.22
	10^{-4}	$3.33\ 10^{-2}$	$3.72\ 10^{-2}$	0.73	0.71	0.12
Pinto(34) $6.9\ 10^7$ params.	-	-	-	0.95	0.77	-

A última coluna foi reproduzida do trabalho em (34).

69

texto

(34) 5x5 k64, 5x5 e 128, 5x5 e 256, 3x3k512, 16896x4096 dense, 5 camadas densas 4096x36, totalizando 69 959 104 parâmetros 13 vezes maior que a maior arquitetura no presente estudo, 180 mil para treino exemplos 9 vezes mais,

alcança 76,6 no teste, 95,3 no treino exibido à um total de 500 mil imagens (200 mil no nosso) durando 1 hora 18 minutos e 23 segundos

alcança 76,6 no teste, 84,38 no treino exibido à um total de 200 mil imagens 1 hora 23 minutos e 54 segundos

Referências

1 ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, p. 65–386, 1958. Citado na página 9.

2 GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. Citado 2 vezes nas páginas 9 e 22.

3 SCHMIDHUBER, J. Deep learning in neural networks: An overview. *Neural Networks*, v. 61, p. 85–117, 2015. Published online 2014; based on TR arXiv:1404.7828 [cs.NE]. Citado 3 vezes nas páginas 9, 21 e 24.

4 BARRON, A. R. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Trans. Information Theory*, v. 39, p. 930–945, 1993. Citado na página 9.

5 ANDONI, A. et al. Learning polynomials with neural networks. v. 5, p. 3955–3963, 01 2014. Citado na página 9.

6 KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: PEREIRA, F. et al. (Ed.). *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012. p. 1097–1105. Disponível em: <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>. Citado na página 9.

7 MNIH, V. et al. Human-level control through deep reinforcement learning. *Nature*, Nature Publishing Group, a division of Macmillan Publishers Limited. All Rights Reserved., v. 518, n. 7540, p. 529–533, fev. 2015. ISSN 00280836. Disponível em: <http://dx.doi.org/10.1038/nature14236>. Citado na página 9.

8 AHN, L. von et al. Captcha: using hard ai problems for security. In: *Advances in Cryptology, Eurocrypt*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. v. 2656, p. 294–311. Citado 2 vezes nas páginas 9 e 11.

9 CHOW, Y. W.; SUSILO, W. Text-based captchas over the years. *IOP Conference Series: Materials Science and Engineering*, v. 273, n. 1, p. 012001, 2017. Disponível em: <http://stacks.iop.org/1757-899X/273/i=1/a=012001>. Citado 2 vezes nas páginas 9 e 14.

10 CHELLAPILLA, K. et al. *Building Segmentation Based Human-Friendly Human Interaction Proofs (HIPs)*. [S.l.]: Springer, Berlin, Heidelberg, 2005. v. 3517. 1-26 p. Citado 6 vezes nas páginas 9, 11, 12, 13, 26 e 30.

11 YAN, J.; AHMAD, A. S. E. Breaking visual captchas with naive pattern recognition algorithms. p. 279–291, 01 2008. Citado 2 vezes nas páginas 9 e 11.

12 GOODFELLOW, I. J. et al. Multi-digit number recognition from street view imagery using deep convolutional neural networks. *CoRR*, abs/1312.6082, 2013. Disponível em: <http://arxiv.org/abs/1312.6082>. Citado 5 vezes nas páginas 9, 12, 13, 27 e 28.

13 GEORGE, D. et al. A generative vision model that trains with high data efficiency and breaks text-based captchas. v. 358, p. eaag2612, 10 2017. Citado 3 vezes nas páginas 9, 27 e 28.

14 BURSZTEIN, E. et al. The end is nigh: Generic solving of text-based captchas. In: *WOOT*. [S.l.: s.n.], 2014. Citado na página 9.

15 SINGH, V. P.; PAL, P. Survey of different types of captcha. *International Journal of Computer Science and Information Technologies*, v. 5, n. 2, p. 2242–2245, 2014. Citado na página 11.

16 SECURIMAGE. *An open-source free PHP CAPTCHA script f.* 2018. Disponível em: <https://www.phpcaptcha.org/>. Acesso em: 23/06/2018. Citado na página 11.

17 AHN, L. von et al. recaptcha: Human-based character recognition via web security measures. v. 321, p. 1465–8, 09 2008. Citado na página 12.

18 SIVAKORN, S.; POLAKIS, I.; KEROMYTIS, A. D. I am robot: (deep) learning to break semantic image captchas. In: . [S.l.]: IEEE, 2016. p. 388–403. Citado na página 13.

19 BURSZTEIN, E.; MARTIN, M.; MITCHELL, J. Text-based captcha strengths and weaknesses. In: *Proceedings of the 18th ACM Conference on Computer and Communications Security*. New York, NY, USA: ACM, 2011. (CCS '11), p. 125–138. ISBN 978-1-4503-0948-6. Disponível em: <http://doi.acm.org/10.1145/2046707.2046724>. Citado 2 vezes nas páginas 14 e 26.

20 GERSTNER, W. et al. *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014. Disponível em: <http://neuronaldynamics.epfl.ch/>. Acesso em: 23/06/2018. Citado na página 17.

21 LECUN, Y. et al. Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, MIT Press, Cambridge, MA, USA, v. 1, n. 4, p. 541–551, dez. 1989. ISSN 0899-7667. Disponível em: <http://dx.doi.org/10.1162/neco.1989.1.4.541>. Citado na página 18.

22 LECUN, Y. et al. Gradient-based learning applied to document recognition. In: *Proceedings of the IEEE*. [S.l.: s.n.], 1998. v. 86, p. 2278 – 2324. Citado na página 18.

23 LEE, H.; EKANADHAM, C.; NG, A. Y. Sparse deep belief net model for visual area v2. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2008. p. 873–880. Citado na página 21.

24 GATYS, L. A.; ECKER, A. S.; BETHGE, M. Texture synthesis using convolutional neural networks. In: *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*. Cambridge, MA, USA: MIT Press, 2015. (NIPS'15), p. 262–270. Disponível em: <http://dl.acm.org/citation.cfm?id=2969239.2969269>. Acesso em: 23/06/2018. Citado na página 21.

25 GATYS, L. A.; ECKER, A. S.; BETHGE, M. Image style transfer using convolutional neural networks. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2016. p. 2414–2423. Citado na página 21.

26 SABOUR, S.; FROSST, N.; HINTON, G. E. Dynamic routing between capsules. In: *Advances in Neural Information Processing Systems*. [S.l.: s.n.], 2017. p. 3856–3866. Citado na página 21.

27 KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. Disponível em: <http://arxiv.org/abs/1412.6980>. Citado 2 vezes nas páginas 23 e 31.

28 FRIEDMAN, J.; HASTIE, T.; TIBSHIRANI, R. *The elements of statistical learning*. [S.l.]: Springer series in statistics New York, NY, USA:, 2001. v. 1. Citado na página 24.

29 HINTON, G. E. et al. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012. Disponível em: <http://arxiv.org/abs/1207.0580>. Citado na página 25.

30 YANN, L.; CORINNA, C.; CHRISTOPHER, J. *The MNIST database of handwritten digits*. 1998. Disponível em: <http://yann.lecun.com/exdb/mnist>. Acesso em: 23/06/2018. Citado na página 26.

31 Cireşan, D.; Meier, U.; Schmidhuber, J. Multi-column Deep Neural Networks for Image Classification. *CoRR*, fev. 2012. Citado na página 27.

32 NETZER, Y. et al. Reading digits in natural images with unsupervised feature learning. In: *NIPS workshop on deep learning and unsupervised feature learning*. [S.l.: s.n.], 2011. v. 2011, n. 2, p. 5. Citado 2 vezes nas páginas 27 e 28.

33 SERMANET, P.; CHINTALA, S.; LECUN, Y. Convolutional neural networks applied to house numbers digit classification. In: IEEE. *Pattern Recognition (ICPR), 2012 21st International Conference on*. [S.l.], 2012. p. 3288–3291. Citado 2 vezes nas páginas 27 e 28.

34 PINTO, V. A. *Redes Neurais Convolucionais de Profundidade Para Reconhecimento de Texto em Imagens de CAPTCHA*. Florianópolis, Santa Catarina.: UNIVERSIDADE FEDERAL DE SANTA CATARINA - DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA, 2016. Disponível em: <https://repositorio.ufsc.br/xmlui/handle/123456789/171436>. Acesso em: 23/06/2018. Citado 4 vezes nas páginas 27, 28, 36 e 37.

35 SIMPLECAPTCHA. *A CAPTCHA Framework for Java*. 2018. Disponível em: <http://simplecaptcha.sourceforge.net/>. Acesso em: 23/06/2018. Citado na página 30.

36 HE, K. et al. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015. Disponível em: <http://arxiv.org/abs/1502.01852>. Citado na página 31.

37 PRECHELT, L. Automatic early stopping using cross validation: Quantifying the criteria. v. 11, p. 761–767, 06 1998. Citado na página 31.

38 ABADI, M. et al. Tensorflow: a system for large-scale machine learning. In: *OSDI*. Savannah, GA, USA: [s.n.], 2016. v. 16, p. 265–283. ISBN 978-1-931971-33-1. Disponível em: <http://download.tensorflow.org/paper/whitepaper2015.pdf>. Acesso em: 23/06/2018. Citado na página 32.