UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ALAN FICAGNA

Classificação de imagens utilizando deep supervised learning

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Prof. Dr. Paulo Martins Engel

Porto Alegre
2015

"Virtue is more to be feared than vice,
because its excesses are not subject to the regulation of consciousness"
— Adam Smith

RESUMO

Este documento é uma revisão do estado da arte em relação as técnicas de Deep Learning utilizadas no reconhecimento de caracteres escritos à mão, tendo o banco de dados MNIST como meio de comparação. Além disso são apresentadas algumas variações nos meta parâmetros do algoritmo tais como <TODO> para avaliar o impacto no tempo de execução e na qualidade do resultado.

Palavras-chave: IA. inteligência artificial. deeplearning.

Recognizing Handwritten Characters Using Deep Learning Algorithms

ABSTRACT

This document is a review of the state of the art in regards to Deep Learning techniques used in handwritten characters recognition having the MNIST database for comparison. Also, the impact in the change of parameters such as <TODO> was measured to provide and indication of its effects on run speed and quality of result.

Keywords: IA, deeplearning.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

MNIST	Mixed National Institute of Standards and Technology
CNN	Convolutional Neural Network
GPU	Graphical Processing Unit
ReLU	Rectified Linear Unit
CONV	Convolutional
FC	Fully Connected
ILSVRC	Large Scale Visual Recognition Challenge
RBM	Restricted Boltzman Machines
EBM	Energy Based Model

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos houve um grande avanço na Computação Visual graças ao ressurgimento de um campo da Inteligência Artificial conhecido como Redes Neurais. Este progresso foi responsável pelo despertar do interesse e investimentos de grandes empresas como Google, Facebook e outros que apostam no desenvolvimento de aplicações que há poucas décadas atrás eram apenas tema de ficção científica tais como scanners faciais de alta precisão, reconhecimento de objetos em cenas, carros que não precisam de motorista e vários tipos de classificadores em geral. Essas novas técnicas ficaram conhecidas como Deep Learning e prometem fazer uso da enorme quantidade de dados e informações disponíveis na internet e nas bases de dados das empresas.

Este progresso foi causado por fatores como o incremento da capacidade de processamento dos computadores e a ubiquidade de tecnologias como GPUs que facilitaram o processamento vetorial e paralelo de imagens, sons e outros domínios de alta dimensionalidade que até então possuíam um custo computacional proibitivo.

1.1 Objetivos

<REVISAR> O objetivo deste trabalho consiste em efetuar uma revisão do histórico do desenvolvimento das técnicas de Deep Learning e um resumo do seu estado da arte com a finalidade de adquirir proficiência no uso das mesmas. Além disso, serão apresentados resultados práticos obtidos com a execução de uma modelo que utiliza Deep Learning chamado de Rede Neural Convolucional (CNN) no reconhecimento de caracteres escritos à mão utilizando o algoritmo da rede LeNet de LeCun et al sobre a base MNIST.

2 HISTÓRICO

2.1 Inteligência Artificial

Definir o que é inteligência não é uma tarefa fácil, e inteligência artificial é mais difícil ainda. Historicamente, existiram duas abordagens para a definição de inteligência: O comparativo com humanos e o comparativo com seres racionais.

2.1.1 Inteligência Humana

No primeiro modelo, o seres da espécie humana são tidos como parâmetros e portanto, qualquer avaliação de inteligência tem que se refletir em comportamentos ou pensamentos similares aos humanos. O Teste de Turing proposto pelo matemático Alan Turing (1950) foi uma maneira operacional de estabelecer inteligência pelo comparativo comportamental. Em sua versão mais simplificada, um computador poderia ser considerado inteligente se, ao ser interrogado por um humano com algumas questões, este não pudesse determinar se as respostas eram oriundas de um computador ou de uma pessoa. A Ciência Cognitiva, por outro lado, busca compreender o mecanismo pelos quais os humanos são capazes de pensar e à partir disso estabelecer comparações com algum algoritmo que eventualmente pudesse replicar estas capacidades.

2.1.2 Inteligência Racional

Além do paradigma humano, existe um outro que é a comparação com seres racionais. Um ser é dito racional se ele age da "maneira correta", dado o que ele conhece. Isso nos leva a seguinte definição: um ser é inteligente se ele age ou pensa como um ser racional.

Aristóteles foi o primeiro a sistematizar uma maneira de "pensar corretamente". Ele foi o criador da disciplina que hoje conhecemos por Lógica que estabelecia estruturas argumentativas tais como: "Sócrates é um homem; todos os homens são mortais; portanto, Sócrates é mortal" as quais sempre produzem conclusões corretas. Porém, estes silogismas nem sempre são capazes de traduzir conhecimento informal, principalmente em face à incertezas e por isso tendem a não ser reconhecidos como tudo que existe em termos de

inteligência.

Além disso existem diferenças entre resolver um problema em princípio e resolvê-lo na prática. Embora a abordagem pelas "leis do pensamento" do Aristóteles foquem em realizar inferências corretas, esta não é capaz de exaurir o que significa agir como um ser racional pois existem situações em que não há uma ação comprovadamente correta, mas em que uma ação deve ser tomada mesmo assim. Além disso, existem comportamentos racionais que não podem ser deduzidos por mera inferência como, por exemplo, o reflexo de remover rapidamente a mão do fogo sem uma cuidadosa pré deliberação sobre o assunto.

O propósito deste trabalho não é exaurir todos os ângulos pelos quais esta questão já foi abordada, tais como psicologia, lógica e filosofia mas sim apresentar apenas uma introdução que permita entender o surgimento do o campo da Inteligência Artificial dentro da ciência da computação.

2.2 Histórico da Inteligência Artificial

2.2.1 Primeiros anos

O primeiro trabalho a ser considerado efetivamente "Inteligência Artificial" foi feito em 1943 por Warren McCulloch e Walter Pits, no qual foi proposto o modelo de funcionamento de um neurônio artificial. Este neurônio podia assumir os estados "ligado" e "desligado", determinado em resposta aos estímulos proporcionados pelos outros neurônios vizinhos. Foi demonstrado também que uma rede de neurônios artificial conseguia computar qualquer função computável e sugerido que a mesma poderia aprender.

Em 1949 Donald Hebb demonstrou uma simples regra pela qual a força ou os pesos numéricos de conexão entre os neurônios poderia ser modificada, a qual ficou conhecida como aprendizado hebbiano. Logo em seguida, em 1950, foi construído o primeiro computador de rede neural chamado de SNARC que fora feito com 3000 tubos de vácuo e conseguia simular uma rede de 40 neurônios.

Embora o neurônio artificial seja considerado o primeiro trabalho, o termo Inteligência Artificial só viria a ser cunhado mais de 10 anos depois em Darthmouth nos Estados Unidos em um workshop organizado por John McCarthy que duraria dois meses e cujo propósito era estudar teoria dos autômatos, redes neurais e inteligência. Embora não tenha produzido nenhum progresso, o workshop serviu para introduzir as principais figuras da área entre si.

Nos seus primeiros anos a IA obteve bastante sucesso. Um a um, os items da lista "uma máquina nunca vai poder fazer X" compilada por Turing iam sendo desbancados. Newel and Simon criaram o primeiro GPS (General Problem Solver), um programa desenhado para imitar a maneira como humanos pensam ao invés de simplesmente seguir regras lógicas. Na classe limitada de desafios que conseguia resolver, a ordem dos subobjetivos que o programa tentava seguir era similar à humana. Na IBM, Nathaniel Rochester criaria o primeiro Provador de Teoremas Geométricos, o qual conseguia provar teoremas que vários estudantes consideravam complexos. Arthur Samuel refutou a ideia de que programas somente conseguiriam fazer o que lhes eram dito criando um programa de computador que conseguia jogar um jogo melhor que o seu criador.

Foi nessa época também que McCarthy criaria sua linguagem de programação *Lisp* que viraria a linguagem dominante em IA e com a qual ele faria o primeiro algoritmo que aceitava novos axiomas, sendo então o primeiro autômato que conseguia adquirir novas capacidades sem necessitar ser reprogramado. Outras contribuições significativas foram o programa SAINT (1963) que conseguia resolver problemas de Cálculo típicos do primeiro ano de curso, o programa ANALOGY (1968) respondia a perguntas de analogia geométrica como as de testes de QI, o programa STUDENT (1967) resolvia problemas algébricos e o programa SHRDLU (1972) era capaz de executar instruções do tipo "Encontre um bloco maior do que você está segurando agora e o coloque na caixa". Foi nessa época também que foram feitas melhorias no processo de aprendizado Hebbiano que ficaram conhecidas como redes *adalines*.

2.2.2 Maturação e Estado da Arte

O sucesso inicial foi seguido de um período de relativa estagnação. O fato de que estes algoritmos só conseguiam fazer manipulação simbólica e não conheciam nada sobre o domínio que trabalhavam provou ser um grande empecilho. Além disso a maioria dos primeiros programas de IA resolviam o problema tentando várias combinações até que o resultado fosse atingido, o que não era factível para instâncias de problemas marginalmente maiores. Foi nesta época que Minsky e Papert provaram que um neurônio artificial agora conhecido como *perceptron* era restrito no tipo de funções que conseguia representar e o financiamento de pesquisas na area se tornou escasso. Já na década de 1970, houve o surgimento dos sistemas baseados em conhecimento prévio como o programa DENDRAL que viu uma melhora significativa de desempenho na tarefa de inferir a estrutura mo-

lecular à partir da massa dos seus componentes. O programa que inicialmente tentava todas as alternativas possíveis passou a utilizar de conhecimento de especialistas em química para identificar subestruturas conhecidas o que reduzia significadamente o número de tentativas.

O primeiro caso de sucesso comercial reportado destes sistemas especialistas era um programa que ajudava na compra de novos computadores e que teria economizado à empresa US$40 milhões por ano, o que explica o fato destes sistemas continuarem sendo usados até hoje.

Paralelamente, houve o ressurgimento das redes neurais no final da década de 1980, proporcionado pela reinvenção do algoritmo de *back-propagation*, uma maneira eficiente de atualizar os pesos das conexões nos neurônios, e também uma melhoria nas práticas de pesquisa na área que passou a exigir um maior rigor científico acompanhado do uso de bancos de dados padrões que permitiam o teste e comparação dos resultados entre diferentes algoritmos. Em uma outra linha recente, a ênfase no algoritmo da lugar ao uso de grandes quantidades de dados inspirado no trabalho de Yarowsky (1995) que demostrou ser possível reconhecer se o uso da palavra *plant* significava uma flor ou uma fábrica com precisão acima de 96% sem utilizar de rótulos fornecidos por humanos, apenas com as definições do dicionário desta palavra. Trabalhos como o dele sugerem um "gargalo de conhecimento" na IA, ou seja, o conhecimento que o um sistema precisa pra resolver um problema pode vir de grandes massas de dados ao invés de intervenções de especialistas na área. Hoje em dia, a IA está em várias aplicações de campos diferentes, como por exemplo:

- Na área de robótica veicular com carros auto guiados.
- Na área de reconhecimento de fala.
- Planejamento automático de tarefas.
- Jogos, com destaque para o computador DEEP BLUE da IBM que derrotou o campeão mundial de xadrez, Garry Kasparov.
- Controle de SPAN em mensagens de e-mail.
- Militar e Logística.

3 REVISÃO SOBRE REDES NEURAIS

As primeiras definições de uma rede neural artificial foram feitas em 1943 por McCulloch e Pits inspiradas na hipótese de que as atividades mentais do cérebro humano consistem em reações eletroquímicas em redes de células cerebrais chamadas de neurônios. O neurônio artificial consiste em uma unidade computacional abstrata capaz de receber estímulos de entrada e produzir uma ou mais saídas e uma rede neural artificial — de agora em diante referenciada apenas por *rede neural* — consiste em um grafo direcionado cujos nodos são neurônios artificiais.

3.1 O neurônio artificial

Cada neurônio recebe m entradas ponderadas por um peso w, correspondente a entrada do seu i-ésimo vizinho. Esta configuração foi inspirada nos neurônios naturais cujas interconexões são reguladas por um potencial de ativação responsável por diferenciar a intensidade entre as mesmas. A saída é produzida aplicando-se a função limite σ na soma de todos as entradas ponderadas do neurônio. Esta função simula a *lógica de limite* existente nos neurônios naturais, os quais somente irão propagar o seu sinal caso haja um acúmulo suficiente de potencial elétrico no mesmo. Em sua forma matemática, a saída y do neurônio artificial pode ser expressa da seguinte forma:

$$y = \sigma(\sum_{i=1}^{m} x_i w_i + b)$$

σ representa uma classe de funções que podem ser utilizadas dependendo do objetivo da construção da rede neural. Para problemas de classificação *linearmente separáveis*, pode ser usada a função limite exemplificada na figura 3.2, enquanto para problemas não linearmente separáveis é necessário utilizar algum tipo de função *sigmoide*. A função logística é um exemplo de função sigmoide frequentemente utilizada por ser diferenciável em todo o seu domínio, o que permite produzir garantias de convergência nos algoritmos de aprendizagem.

O valor b corresponde ao *bias* aplicado a função e pode ser omitido simplesmente inserindo-se artificialmente uma nova entrada $x_0 = 1$.

Figura 3.1 – Abstração de um neurônio artificial

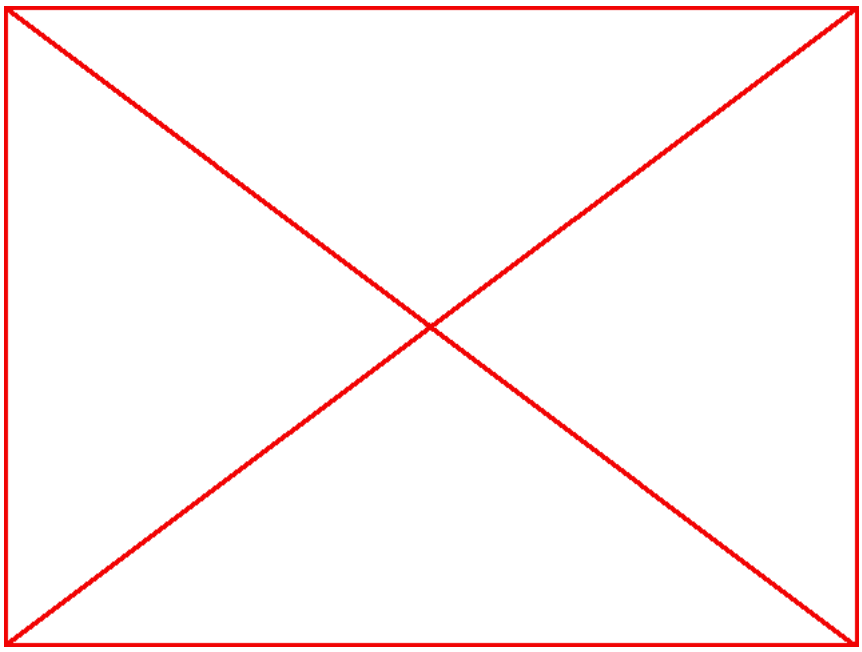


Figura 3.2 – Exemplo de funções de limite σ usadas em neurônios artificiais

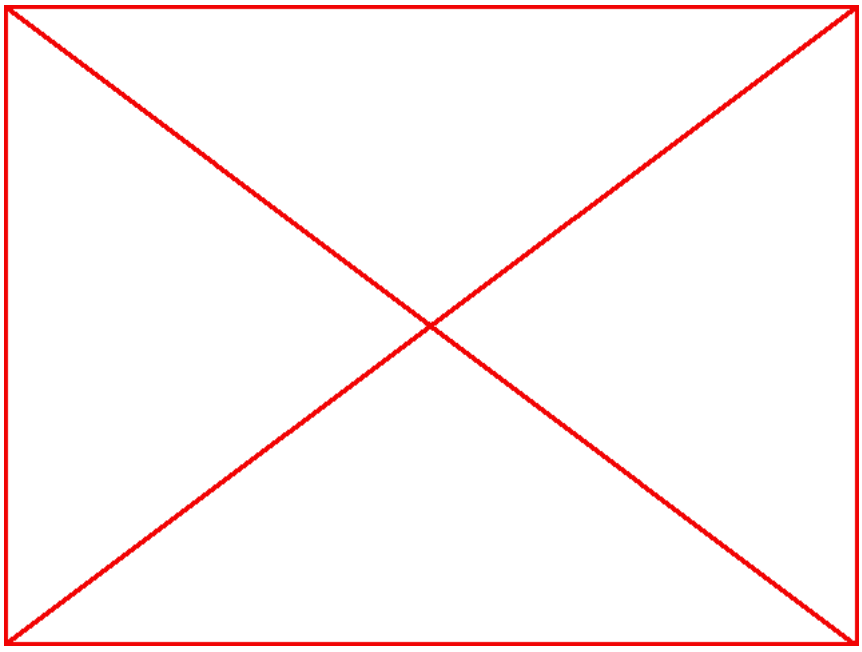
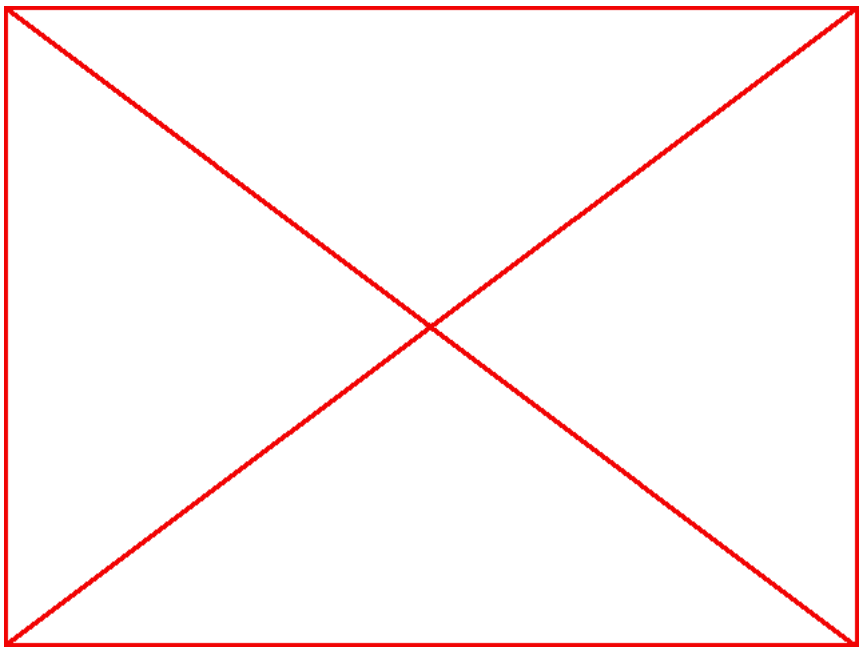


Figura 3.3 – Exemplo de função aproximada com e sem o vetor de bias



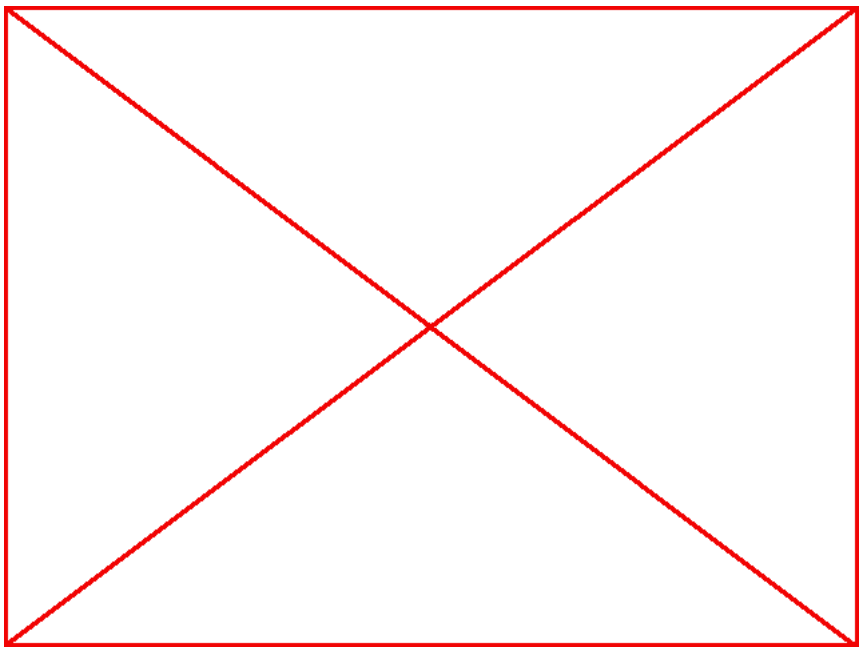
3.1.1 Bias

Todo neurônio artificial é dito ter pelo menos uma entrada extra $x_0 = 1$ e consequentemente, o peso extra associado w_0 chamado de bias b. O bias permite que a função aproximada pelo neurônio não necessite obrigatoriamente passar pela origem $(0, 0)$, o que reduz o erro médio da aproximação, caso os dados de treinamento não tenham sofrido nenhum tipo de ajuste numérico, como por exemplo, a remoção do valor médio.

3.1.2 Redes neurais

Em uma rede neural, os neurônios são organizados em *camadas*. Alguns autores não consideram a camada de entrada como uma camada distinta, mas esta abstração permite simplificar a notação e torna o processo de conectar redes neurais entre si mais intuitivo. Outra camada especial é a de saída, pois é ela que determina se a rede exercerá a função de um *classificador* ou de um *regressor*. Todas as demais camadas intermediárias são chamadas de *camadas ocultas*; o nome não possui nenhuma conotação especial e serve apenas para diferenciá-las das camadas de entrada e saída. Em uma rede *feed forward* as saídas de cada camada somente são computadas na direção e sentido da camada de saída,

Figura 3.4 – Exemplo de uma rede neural de 3 camadas



isto é, não há uma retroalimentação das conexões de saída para as de entrada. Quando isto acontece a rede é chamada de *recorrente* porém, o estudo deste tipo de rede neural não esta nos objetivos deste trabalho.

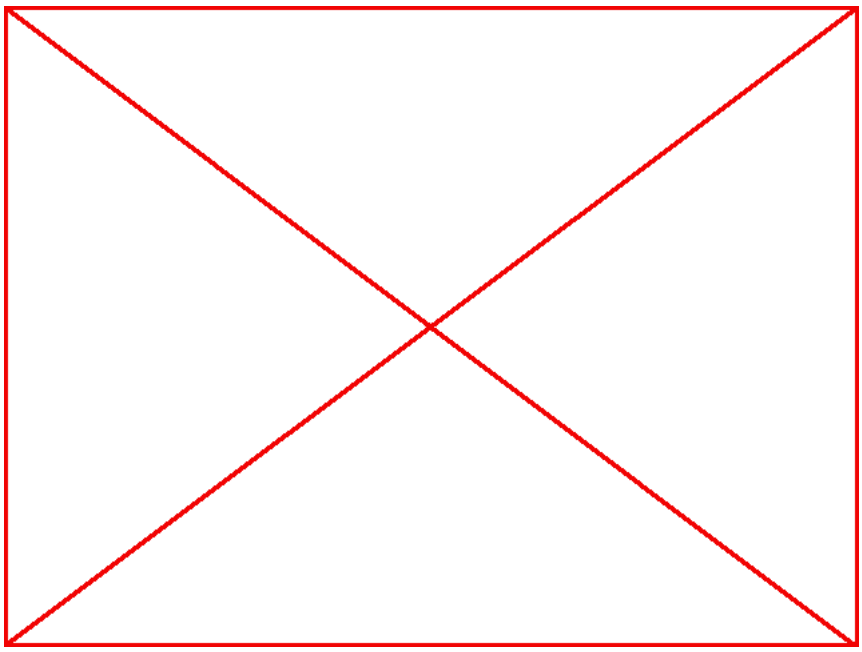
Sendo assim, uma rede neural necessita ter pelo menos duas camadas — a de entrada e a de saída. A adição de camadas ou neurônios não possui um efeito óbvio e generalizável para qualquer rede, e portanto é alvo de constante experimentação e observação empírica sujeito a comportamento variável dependendo do problema.

Cada camada S é formada por R neurônios, cujos pesos das conexões podem ser representados através de uma matriz $\mathbf{W}$. A notação em forma vetorial e matricial dos pesos das camadas permite que operações mais eficientes sejam utilizadas para realizar a computação da saída da rede neural e do ajuste dos mesmos.

$$W_{S,R} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ \vdots & \vdots & \ddots & \vdots \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix}$$

A saída ou *ativação* de uma camada $\tilde{\mathbf{a}}'$ pode ser expressa em função da ativação da camada anterior $\tilde{\mathbf{a}}$ da seguinte maneira:

Figura 3.5 – Exemplo de funções de ativação



Na esquerda uma simples função de limite; à direita a função $\frac{1}{1+e^{-x}}$

$$\tilde{\mathbf{a}}' = \sigma(\tilde{\mathbf{W}}\tilde{\mathbf{a}} + \tilde{\mathbf{b}})$$

3.1.3 Aprendizagem

A aprendizagem em uma rede neural se da pelo ajuste dos pesos w das conexões dos neurônios. Ao longo dos anos foram propostas várias maneiras de atualizar os pesos de uma rede neural automaticamente através de uma *regra de aprendizagem*. Estas regras foram categorizadas em três grandes grupos:

Aprendizagem Supervisionada As entradas são acompanhadas por um conjunto de valores corretamente associados a mesma.

Aprendizagem Por Reforço Semelhante ao aprendizado supervisionado, porém ao invés da resposta correta, é fornecido algo como uma nota ou uma indicação da performance da predição.

Aprendizagem Não Supervisionada Nenhuma informação adicional a respeito das entradas é fornecida ao algoritmo.

3.1.3.1 Backpropagation

Obviamente que a aplicabilidade de redes neurais seria extremamente limitada se não houvesse uma maneira eficiente de aprender os pesos da rede de forma automática. Atualmente, a melhor maneira conhecida de fazer o mesmo é através do algoritmo de *back-propagation*. Este algoritmo embora já sabido desde a década de 60, ganhou visibilidade na área de IA na década de 80, quando aplicado em redes neurais.(RUSSELL; NORVIG, 1995)

O processo de aprendizagem de uma rede neural segue a noção intuitiva que temos do assunto: Um individuo faz uma predição acerca de uma proposição ou pergunta, avalia o *feedback* recebido do ambiente e corrige a sua previsão baseado na distância entre o que foi previsto e o que de fato aconteceu.

A determinação da direção no espaço de pesos da correção é dado pelo *gradiente* função de erro $\mathbf{L}$, ou como é mais comumente conhecida, *loss function*. A eficiência do algoritmo de backpropagation se baseia no fato de que a computação de cada camada depende somente da computação das camadas anteriores.

Embora o algoritmo use a derivada da *loss function* pré calculada de maneira analítica, é comum ver nas implementações o uso da computação do gradiente de maneira iterativa como forma de validação da implementação.

A computação completa da rede neural então, se da em dois passos: O primeiro chamado de *forward pass*, a ativação de cada camada é computada até que se chegue numa resposta na camada de saída, a qual é usada para calcular o erro em cada neurônio e o ajuste dos pesos $\Delta W_{i,j}$ do mesmo de acordo com a seguinte equação:

$$\mathbf{\Delta W_{i,j}} = -\alpha \frac{\partial \mathbf{L}}{\partial \mathbf{W_{i,j}}}$$

Onde α representa a taxa de aprendizagem.

3.2 Deep Learning

3.2.1 Por que usar deep learning

Hoje sabemosHubel e Wiesel (1959) que o processamento visual no cérebro humano ocorre de maneira gradual e que diferentes regiões são responsáveis pelo processamento de diferentes *features* extraídas do estímulo sensorial. Primeiro são detectadas abstrações

simples como bordas, que por sua vez são combinadas em funções mais complexas como detecção de movimento e reconhecimento facial. Embora esta seja uma boa fonte de inspiração para o uso de *deep neural networks*, pesquisadores da área hesitam em comparar o que acontece em uma rede neural artificial com o que acontece dentro de um cérebro natural, visto que o completo funcionamento do mesmo é um problema científico em aberto.

A motivação para o uso de uma rede neural *deep* ao invés das arquiteturas tradicionais *shallow*, tais como *support vector machines* ou redes neurais com poucas camadas, visto que estas conseguem simular qualquer outra, é que existe um *tradeoff* entre espaço de memória e tempo de computação que pode ser explorado para acelerar o processamento das redes. Uma maneira de entender este *tradeoff* em redes neurais, é imaginar a saída das camadas escondidas como resultados intermediários da computação. *shallow*(BENGIO; LECUN et al., 2007)

Além disso, até bem pouco tempo atras, os melhores resultados nas tarefas de classificação e detecção de imagens, videos, sons e linguagem natural necessitavam de um amplo conhecimento *a priori* por parte dos cientistas cujo trabalho era elaborar *filtros* capazes de sintetizar toda a complexidade destas tarefas.

Devido ao alto custo do desenvolvimento e pesquisa destes filtros, surge a necessidade de um método que consiga aprendê-los de maneira automática, mas que ainda assim mantenha a precisão dos filtros artesanais. Estes métodos hoje são conhecidos como *deep learning* e envolvem um gama variada de técnicas para contornar os problemas de arquiteturas com múltiplas camadas, a fim de extrair o benefício do tradeoff entre memória e computação mencionado.

3.2.2 Problemas com deep learning

Já que redes neurais já são conhecidas há algum tempo, é de se estranhar que ninguém tenha tido a ideia de acrescentar mais camadas nestas redes anteriormente. O fato é que simplesmente adicionar mais camadas incorre em problemas que até alguns anos atrás eram difíceis de serem contornados.

overfitting: Uma rede com múltiplas camadas tende a apresentar um número maior de parâmetros que podem ser treinados, o que leva a um problema de *overfitting*, ou seja, a rede simplesmente aprende o conjunto de dados de treinamento, mas

não é muito capaz de realizar boas inferências sobre novos conjuntos de dados. Este problema foi combatido com a combinação de algoritmos de aprendizagem não supervisionada que atuam como regularizadores durante a fase de pré treinamento.

saturação do gradiente: Para poder aprender funções convexas complexas, é necessário que os neurônios da rede neural apliquem algum tipo de não linearidade em seu processamento. Porém, o algoritmo de *backpropagation*, utilizado na aprendizagem, pode sofrer do fenômeno de saturação do gradiente. Isto ocorre em neurônios cujo valor de ativação estejam próximos dos extremos da função de ativação fazendo com que o gradiente fique próximo de zero. Esta saturação também propagada para as camadas mais inferiores da rede.

processamento intensivo: Mesmo com estas técnicas, hoje em dia o processamento de redes profundas é predominantemente feito em GPUs, visto que estes hardwares são altamente especializados em processamento vetorial paralelo. O crescente suporte dos fabricantes destes hardwares e de frameworks e bibliotecas que facilitam a utilização dos mesmos também foi importante fator para tornar o uso de redes com múltiplas camadas viável.

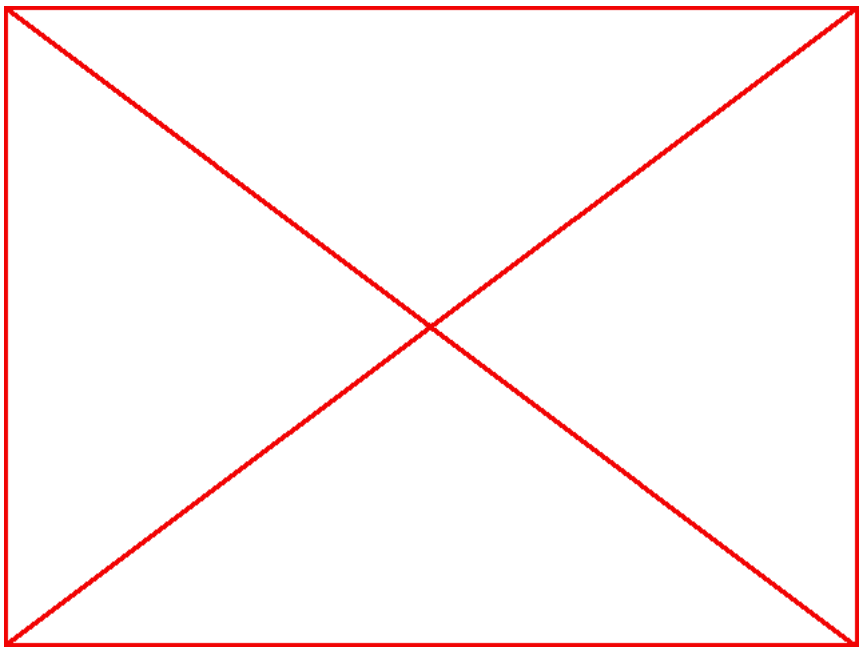
3.2.3 Aprendizado não supervisionado

Aprendizado não supervisionado é usado para regularizar uma rede neural a fim de impedir que a mesma apresente um alto grau de over fitting. Os algoritmos mais empregados para esta tarefa são *autoencoders* e *Restricted Boltzman Machines*, que atuam no pré treinamento de redes neurais profundas como regularizadores, fazendo uma pré calibragem dos parâmetros W antes da rede ser completamente conectada e *fine tuned*. Este processo limita os valores dos pesos das conexões da rede para um intervalo relativamente pequeno e tem gerado bons resultados. Esta combinação de métodos supervisionados com não supervisionados é conhecido como aprendizado semi supervisionado.

3.2.3.1 Autoencoders

Autoencoders são redes neurais de representação binária com uma única camada, cuja *loss function* usa a própria entrada como alvo de aprendizagem. Caso o número de neurônios na camada intermediária seja menor que o número na camada de entrada, então o autoencoder aprende uma representação mais compacta para os dados de entrada,

Figura 3.6 – Exemplo de Autoencoder



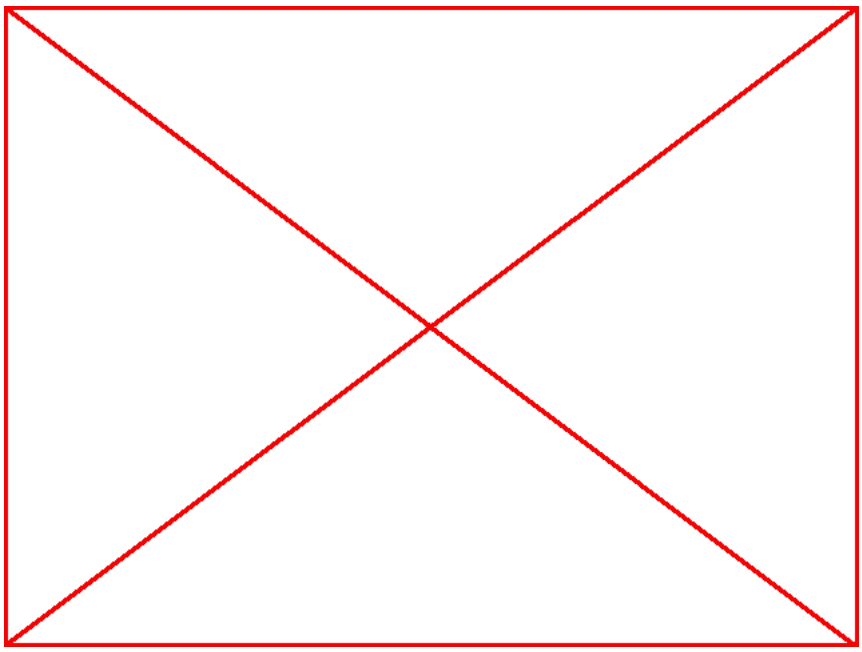
de fato, dependendo da função de erro usada, um autoencoder realiza fatoração de componentes principais na entrada. Se o número de neurônios na camada intermediária for maior ou igual ao número de neurônios na entrada então os autoencoder não tem grande utilidade, exceto nos *denoising autoencoders* onde cada neurônio da entrada original é corrompido (setado para 0) com uma probabilidade p. Isto confere ao autoencoder um pouco de invariabilidade a ruído na rede, o que melhora a generalização.

3.2.3.2 Restricted Boltzman Machines

Restricted Boltzman Machine é um exemplo de um formalismo conhecido como Modelo Baseado em Energia (EBM). O objetivo de um EBM é encontrar uma função de energia $E(X, Y)$ que associa um valor de entrada X e um valor de saída Y a um escalar chamado de energia. Uma boa função de energia é aquela que associa um valor baixo para uma configuração (X, Y) desejada, e um valor de energia alto caso contrário.(LECUN et al., 2006)

Porém, uma Restricted Boltzman Machine é não supervisionado, portanto, a variável Y dos rótulos conhecidos é substituída por uma variável escondida h. A representação mais comum de uma RBM é um grafo bipartido não direcionado completamente conexo, onde uma das partições do grafo representa a entrada que é visível $\vec{v}$, e a outra representa

Figura 3.7 – Exemplo de RBM



a variáveis escondidas do modelo h. Depois de um aprendizado realizado com sucesso, uma RBM contém a distribuição probabilísticas das observações de treinamento. Essa distribuição é expressa pela probabilidade conjunta destas duas variáveis:

$$p(v,h) = \frac{\exp(-E(v,h))}{Z}$$

Onde, $E(v,h)$ é a *função de energia* que é dada pela fórmula:

$$E(v,h) = -\mathbf{bv^T} - \mathbf{ch^T} - \mathbf{h^T Wv}$$

Onde $\mathbf{b}$ e $\mathbf{c}$ são *bias vectors* e $\mathbf{W}$ é a matriz de pesos entre a camada visível e a camada escondida. Intuitivamente, o objetivo destes parâmetros é armazenar uma preferência por uma configuração de variáveis em particular. Além, disso a função de normalização Z é dada por:

$$Z = \sum_v \sum_h \exp(-E(v,h))$$

3.2.4 Convolutional Neural Networks

3.2.4.1 Um breve histórico das CNNs

Embora as CNNs já fossem conhecidas há bastante tempo, o modelo tradicional para abordagem de problemas de classificação com *Convolutional Neural Networks* envolvia o desenvolvimento de filtros criados de maneira "artesanal" por humanos. Na prática, isto limitava muito a aplicação de CNNs e logo ficou clara a necessidade de um processo que pudesse realizar o aprendizado destes filtros de maneira automática. A seguir, a listagem adaptada de Kaparthy (2015a), mostra a evolução destas técnicas principalmente na última década.

- *Neocognitron (1988)*: Criado por Fukushima (1988), foi o precursor das redes neurais convolucionais modernas. Consiste em múltiplas camadas de dois tipos de neurônios artificiais que tinham por objetivo simular a configuração observada por Hubel e Wiesel (1959) em cérebros de gatos, os quais possuíam células responsáveis por tarefas distintas no reconhecimento visual.
- *LeNet (1990)*: Proposta por Cun et al. (1990) foi usada com sucesso para reconhecimento de caracteres escritos a mão usando o dataset MNIST.
- *AlexNet (2012)*: Krizhevsky, Sutskever e Hinton (2012) Foi a ganhadora da competição ILSVRC de 2012 e foi a responsável por grande parte do entusiasmo atual com redes convolucionais. Utiliza técnicas como *droppout* e *ReLUs*.
- *ZF Net (2013)*: Se destacou na competição ILSVRC de 2013, por apresentar melhorias em relação à AlexNet, com a otimização de hyperparametros e a expansão do número de layers intermediários. Zeiler e Fergus (2014)
- *GoogLeNet (2014)*: A principal contribuição de Szegedy et al. (2014) foi o desenvolvimento de um *Inception Module* que conseguiu reduzir o número de parâmetros (pesos) em 15 vezes.
- *VGGNet (2014)*: Zeiler e Fergus (2014) trouxe sua contribuição demonstrando que a profundidade da arquitetura é um componente crucial do desempenho da rede. Mais tarde foi descoberto que a sua performance supera a da GoogLeNet, embora necessite de mais memória e parâmetros. Hoje, é a rede preferida para extrair *features* de imagens.

3.2.4.2 VGGNet

Até o presente momento, o modelo de (ZEILER; FERGUS, 2014) — conhecido como VGGNet — é tido como um dos melhores, obtendo precisão semelhante à humanos (KAPARTHY, 2015b), com uma taxa de erro na tarefa de classificação da ILSVRC de 6,8% quando a classe correta da imagem não está entre as cinco primeiras sugeridas pelo modelo. A competição deste ano (2015) ainda não teve seu resultado anunciado, mas é provável que hajam novas melhorias nos modelos atuais. Mesmo assim, a VGGNet ainda é um caso relevante para estudo.

3.2.4.3 Arquitetura e Modelo

A arquitetura usada pode ser representada da seguinte maneira: (KAPARTHY, 2015a)

```
INPUT:     [224x224x3]   memory: 224*224*3=150K   weights: 0
CONV3-64:  [224x224x64]  memory: 224*224*64=3.2M  weights: (3*3*3)*64 = 1,728
CONV3-64:  [224x224x64]  memory: 224*224*64=3.2M  weights: (3*3*64)*64 = 36,864
POOL2:     [112x112x64]  memory: 112*112*64=800K  weights: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M weights: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M weights: (3*3*128)*128 = 147,456
POOL2:     [56x56x128]   memory: 56*56*128=400K   weights: 0
CONV3-256: [56x56x256]   memory: 56*56*256=800K   weights: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256]   memory: 56*56*256=800K   weights: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256]   memory: 56*56*256=800K   weights: (3*3*256)*256 = 589,824
POOL2:     [28x28x256]   memory: 28*28*256=200K   weights: 0
CONV3-512: [28x28x512]   memory: 28*28*512=400K   weights: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512]   memory: 28*28*512=400K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512]   memory: 28*28*512=400K   weights: (3*3*512)*512 = 2,359,296
POOL2:     [14x14x512]   memory: 14*14*512=100K   weights: 0
CONV3-512: [14x14x512]   memory: 14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]   memory: 14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]   memory: 14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
POOL2:     [7x7x512]     memory: 7*7*512=25K      weights: 0
FC:        [1x1x4096]    memory: 4096             weights: 7*7*512*4096 = 102,760,448
FC:        [1x1x4096]    memory: 4096             weights: 4096*4096 = 16,777,216
FC:        [1x1x1000]    memory: 1000             weights: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters
```

INPUT, *CONV*, *POOL* e *FC* representam camadas na rede com funções específicas. INPUT, CONV e POOL são auto explicativos e FC significa *Fully Connected*. Os números ao lado do nome da camada se referem a características dos filtros que devem ser aprendidos. CONV364, por exemplo, indica a presença de 64 filtros cuja distância espacial (altura e largura) é igual a 3. Na camada de pooling, somente é representado a distância espacial, ou seja 2.

As considerações de memória são cruciais visto que redes convolucionais modernas dependem de placas gráficas para acelerar o seu treinamento e a maioria destas possuem limites de memória que vão de 3 até 6GB.

O grande diferencial da VGGNet para a sua maior competidora, a GoogLeNet, é o uso de filtros pequenos $F = 3$, os quais aceleram o treinamento da rede. Além disso a rede apresenta uma profundidade (número de camadas) bastante elevado (até 19), corroborando a ideia de que a profundidade da rede é fundamental para um bom desempenho.

3.2.5 Funcionamento

Redes convolucionais são um tipo especial de *deep networks* projetadas especificamente para lidar com dados vetoriais tais como imagens e sons. Este tipo de rede neural ganhou fama na competição ILSVRC realizada em 2012 na qual conseguiu-se diminuir pela metade o erro do melhor competidor. Este feito foi atingido com o uso de ReLUs (*rectified linear units*), GPUs e uma técnica conhecida como *dropout*. Desde então, Redes Convolucionais ganharam força e hoje são as melhores técnicas para reconhecimento de imagens.(LECUN; BENGIO; HINTON, 2015)

Uma rede convolucional é baseada em 4 ideias principais: conexões locais, pesos compartilhados, *pooling* e múltiplas camadas. Conexões locais exploram o fato de que em uma imagem existe uma correlação entre os valores dos pixels vizinhos. Já pesos compartilhados, geram o conceito de *feature maps* e reduzem o número de pesos que precisam ser aprendidos e a operação de *pooling* (geralmente o máximo local) é usada pois é invariante a posição, o que da um maior poder de abstração para a rede.(LECUN; BENGIO; HINTON, 2015)

A operação de filtro é realizada pela função matemática de convolução, cujo papel

é detectar conjunções locais nas *features* da camada anterior. Logo em seguida, vem a camada de *pooling* cujo papel é combinar *features* semanticamente semelhantes.(LECUN; BENGIO; HINTON, 2015)

A rede é tipicamente composta de dois a três estágios de convolução seguidos da aplicação de não linearidade e de *pooling*. Por fim, são aplicadas mais camadas convolucionais, desta vez, completamente conectadas. O aprendizado é feito através de *backpropagation* como em redes neurais tradicionais.(LECUN; BENGIO; HINTON, 2015)

3.2.5.1 Hyper parâmetros

O termo "hiper parâmetros"se refere as "alavancas"que um *designer* pode puxar enquanto decide a arquitetura da rede convolucional. Além dos hyper parâmetros tradicionalmente conhecidos das redes neurais convencionais, tais como, quantidade de neurônios por camada, função de não linearidade, taxa de aprendizado e função de regularização, as redes convolucionais contém alguns parâmetros que são exclusivos para o seu funcionamento:

- *Número de filtros (K):* Este parâmetro regula quantos filtros serão treinados na rede.
- *Stride (S):* Se refere ao passo que é dado pela janela de convolução na imagem original.
- *Distância espacial (F):* Se refere a largura e altura dos filtros.
- *Padding ou Preenchimento (P):* Se refere a largura e altura dos filtros.

Porém nem toda combinação destes parâmetros é válida devido a propriedades geométricas. Considerando um volume de entrada de dimensões: $W_1 \times H_1 \times D_1$, produzirá um volume de tamanho: $W_2 \times H_2 \times D_2$ sendo que:

$$W_2 = \frac{(W_1 - F + 2P)}{S + 1}$$

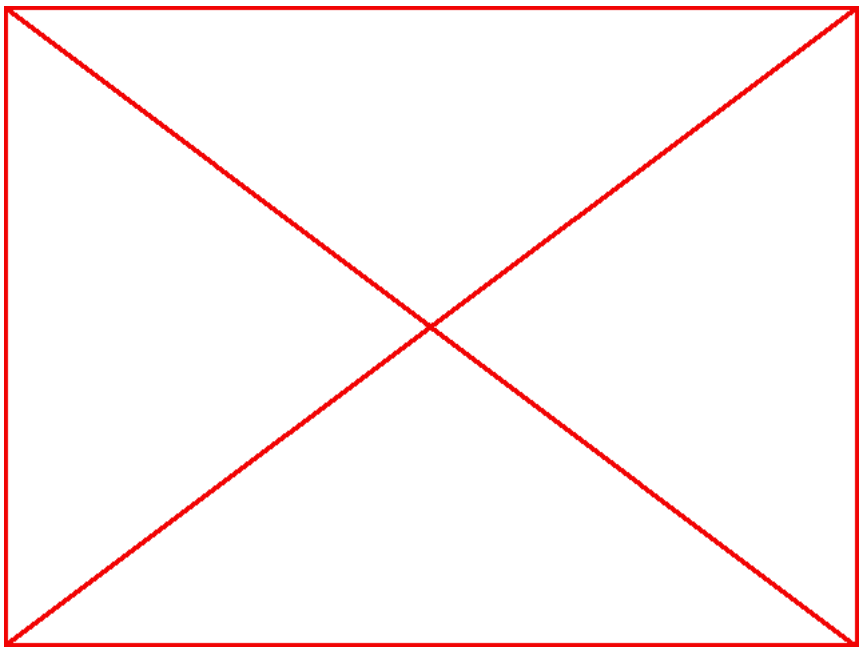
$$H_2 = \frac{(H_1 - F + 2P)}{S + 1}$$

$$D_2 = F$$

Uma combinação válida destes parâmetros produz um valor de W_2 (ou H_2) inteiro.

Assumindo o uso de *compartilhamento de parâmetros*, cada filtro adiciona um total de $(F \cdot F \cdot D_1) \cdot K$ pesos e K *biases*.

Figura 3.8 – Exemplo de configuração espacial



A camada de pooling também possui seu próprio parâmetro F e S e seu respectivo volume de saída terá as dimensões:

$$W_2 = (W_1 - F)/S + 1$$

$$H_2 = (H_1 - F)/S + 1$$

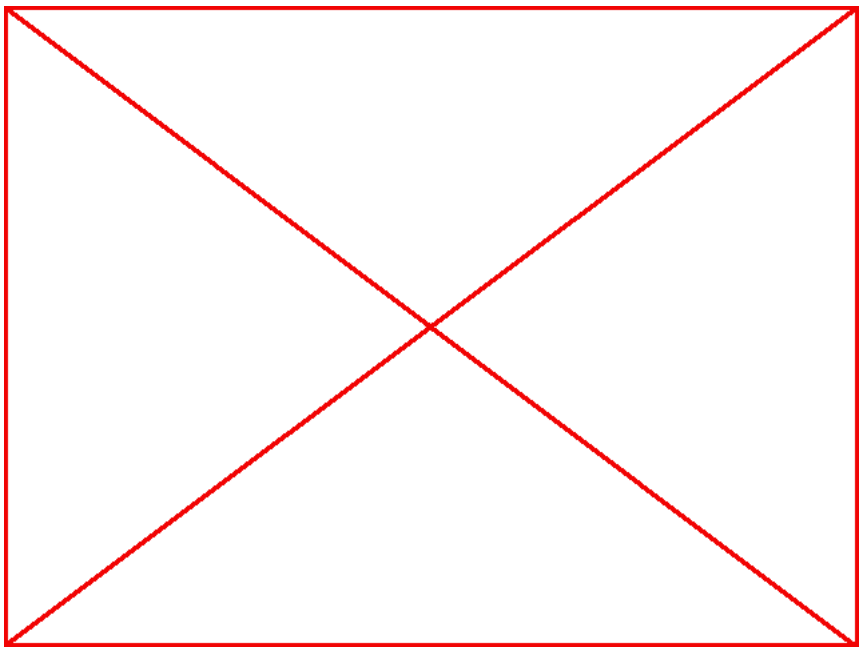
$$D_2 = D_1$$

3.2.5.2 Dropout

O objetivo da técnica de *dropout* é aproximar a solução ótima de calcular a média dos resultados de treinamento de todas as combinações de parâmetros possíveis. Isto é feito pela remoção aleatória e temporária de unidades da rede bem como suas conexões. Cada unidade é mantida na rede com uma probabilidade p, independente das outras.(SRIVASTAVA et al., 2014)

Com a aplicação de *dropout* há uma redução no *overfitting* e uma melhoria na regularização, porém, o treinamento da rede tende a demorar mais gerando um *tradeoff* entre *overfitting* e tempo de treinamento.(SRIVASTAVA et al., 2014)

Figura 3.9 – Uma rede neural com a aplicação de dropout



Está técnica não está restrita somente a redes *feed forward* e pode ser aplicadas a *RBM* e *autoencoders*, proporcionando uma maneira de combinar exponencialmente diferentes arquiteturas eficientemente.(SRIVASTAVA et al., 2014)

3.2.5.3 ReLU

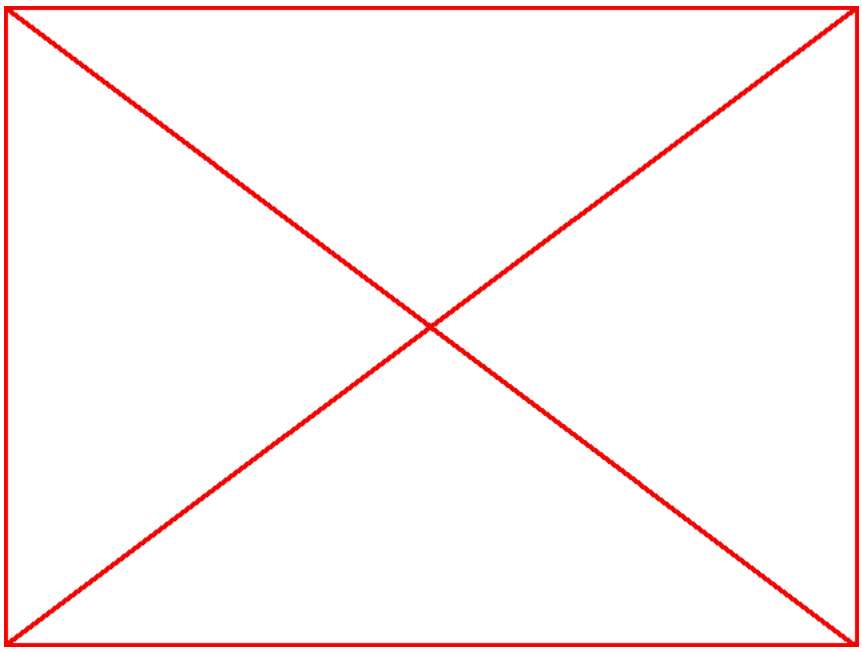
Em uma rede neural tradicional, as funções $\tanh(x)$ e $\frac{1}{1+e^{-x}}$ são usadas para introduzir não linearidade na computação do produto dos pesos pelas entradas dos neurônios. Porém, estas funções possuem a desvantagem de apresentarem o fenômeno de saturação. Isto ocorre quando a entrada da função está distante da ponto central ($x = 0$) e provoca uma perda de desempenho no aprendizado.(KRIZHEVSKY; SUTSKEVER; HINTON, 2012)

Por causa disto, o uso de *rectifiers* tem se tornado cada vez mais comum no treinamento de grandes redes. Este *rectifier* é definido como:

$$f(x) = \max(0, x)$$

Quando uma unidade ou neurônio usa este *rectifier* ele é chamado de Rectified Linear Unit ou ReLU. De acordo com (KRIZHEVSKY; SUTSKEVER; HINTON, 2012), ReLUs são muito mais rápidos do que *saturating non linearities* — como são conhecidas

Figura 3.10 – Tipos de não linearidades aplicadas



as funções mencionadas anteriormente — e não requerem regularização, embora isto ainda seja usado para aumentar o poder de generalização da rede.

3.2.5.4 Regularização

Em uma típica rede *deep*, existem milhares de pesos que sofrem adaptação em seus valores. Normalmente um aproximador de função com este grande número de parâmetros ajustáveis está sujeito a *over fitting*. Isto reduz a capacidade de generalização do modelo gerado para entradas que não fizeram parte do treinamento. Redes neurais contornam este problema com o uso de técnicas de regularização tais como:

- *Regularização L2*: Para cada peso w da rede, é adicionado uma penalidade $\frac{1}{2}\lambda w^2$, onde λ é a força de regularização. O termo $\frac{1}{2}$ é adicionado para facilitar o cálculo da derivada na fase de *backpropagation* e o termo todo tem a interpretação de penalizar vetores de pesos com alguns poucos pesos muito altos e incentivar o uso de vetores difusos, ou seja, eles incentivam a rede a utilizar todos as suas entradas ao invés de usar demasiadamente uma entrada ou outra.

- *Regularização L1*: Para cada peso w da rede, é adicionado uma penalidade $\lambda|w|$ com a finalidade de tornar os pesos na rede esparsos (próximos de zero). Isto confere a rede um certo grau de invariabilidade à entradas com ruído. A regularização L1 pode ser usada juntamente com a L2, sendo chamadas neste caso de *Elastic net regularization*.
- *Max norm constraints*: Consiste em estipular um valor máximo para os pesos w de maneira que a rede não se torne instável mesmo na presença de uma taxa de aprendizado muito grande.
- *Dropout*: A técnica de *dropout* também pode ser vista como uma regularização e pode ser usada juntamente com outros tipos.

3.3 Conclusão

<TODO>

REFERÊNCIAS

BENGIO, Y.; LECUN, Y. et al. Scaling learning algorithms towards ai. **Large-scale kernel machines**, v. 34, n. 5, 2007.

CUN, B. B. L. et al. Handwritten digit recognition with a back-propagation network. In: CITESEER. **Advances in neural information processing systems**. [S.l.], 1990.

FUKUSHIMA, K. Neocognitron: A hierarchical neural network capable of visual pattern recognition. **Neural networks**, Elsevier, v. 1, n. 2, p. 119–130, 1988.

HUBEL, D. H.; WIESEL, T. N. Receptive fields of single neurones in the cat's striate cortex. **The Journal of physiology**, Wiley Online Library, v. 148, n. 3, p. 574–591, 1959.

KAPARTHY, A. **CS231n Convolutional Neural Networks for Visual Recognition**. 2015. <http://cs231n.github.io/convolutional-networks/>. Accessed: 2015-11-27.

KAPARTHY, A. **What I learned from competing against a ConvNet on ImageNet**. 2015. <https://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/>. Accessed: 2015-11-5.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: **Advances in neural information processing systems**. [S.l.: s.n.], 2012. p. 1097–1105.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **Nature**, Nature Publishing Group, v. 521, n. 7553, p. 436–444, 2015.

LECUN, Y. et al. A tutorial on energy-based learning. **Predicting structured data**, v. 1, p. 0, 2006.

RUSSELL, S.; NORVIG, P. A modern approach. **Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs**, Citeseer, v. 25, 1995.

SRIVASTAVA, N. et al. Dropout: A simple way to prevent neural networks from overfitting. **The Journal of Machine Learning Research**, JMLR. org, v. 15, n. 1, p. 1929–1958, 2014.

SZEGEDY, C. et al. Going deeper with convolutions. **arXiv preprint arXiv:1409.4842**, 2014.

ZEILER, M. D.; FERGUS, R. Visualizing and understanding convolutional networks. In: **Computer Vision–ECCV 2014**. [S.l.]: Springer, 2014. p. 818–833.