# Deep Learning Book Data Science Academy

# Capítulo 1 – Deep Learning e a Tempestade Perfeita

O interesse pela Aprendizagem de Máquina (Machine Learning) explodiu na última década. O mundo a nossa volta está passando por uma transformação e vemos uma interação cada vez maior das aplicações de computador com os seres humanos. Softwares de detecção de spam, sistemas de recomendação, marcação em fotos de redes sociais, assistentes pessoais ativados por voz, carros autônomos, smartphones com reconhecimento facial e muito mais.

E o interesse por Machine Learning se mostra ainda mais evidente pelo número cada vez maior de conferências, meetups, artigos, livros, cursos, buscas no Google e profissionais e empresas procurando compreender o que é e como usar aprendizagem de máquina, embora muitos ainda confundem o que podem fazer com o que desejam fazer. Não há como ficar indiferente a esta revolução trazida pela aprendizagem de máquina e, segundo o Gartner, até 2020 todos os softwares corporativos terão alguma funcionalidade ligada a Machine Learning.

Fundamentalmente, Machine Learning é a utilização de algoritmos para extrair informações de dados brutos e representá-los através de algum tipo de modelo matemático. Usamos então este modelo para fazer inferências a partir de outros conjuntos de dados. Existem muitos algoritmos que permitem fazer isso, mas um tipo em especial vem se destacando, as redes neurais artificiais.

As redes neurais artificiais não são necessariamente novas, existem pelo menos desde a década de 1950. Mas durante várias décadas, embora a arquitetura desses modelos tivesse evoluído, ainda faltavam ingredientes que fizessem os modelos realmente funcionar. E esses ingredientes surgiram quase ao mesmo tempo. Um deles você já deve ter ouvido: Big Data. O volume de dados, gerado em variedade e velocidade cada vez maiores, permite criar modelos e atingir altos níveis de precisão. Mas ainda falta um ingrediente. Faltava! Como processar grandes modelos de Machine Learning com grandes quantidades de dados? As CPUs não conseguiam dar conta do recado.

Foi quando os gamers e sua avidez por poder computacional e gráficos perfeitos, nos ajudaram a encontrar o segundo ingrediente: Programação Paralela em GPUs. As unidades de processamento gráfico, que permitem realizar operações matemáticas de forma paralela, principalmente operações com matrizes e vetores, elementos presentes em modelos de redes neurais artificias, formaram a tempestade perfeita, que permitiu a evolução na qual nos encontramos hoje: Big Data + Processamento Paralelo + Modelos de Aprendizagem de Máquina = Inteligência Artificial.

A unidade fundamental de uma rede neural artificial é um nó (ou neurônio matemático), que por sua vez é baseado no neurônio biológico. As conexões entre esses neurônios matemáticos também foram inspiradas em cérebros biológicos, especialmente na forma como essas conexões se desenvolvem ao longo do tempo com “treinamento”. Em meados da década de 1980 e início da década de 1990, muitos avanços importantes na arquitetura das redes neurais artificias ocorreram. No entanto, a quantidade de tempo e dados necessários para obter bons resultados retardou a adoção e, portanto, o interesse foi arrefecido, com o que ficou conhecimento como AI Winter (Inverno da IA).

No início dos anos 2000, o poder computacional expandiu exponencialmente e o mercado viu uma “explosão” de técnicas computacionais que não eram possíveis antes disso. Foi quando o aprendizado profundo (Deep Learning) emergiu do crescimento computacional explosivo dessa década como o principal mecanismo de construção de sistemas de Inteligência Artificial, ganhando muitas competições importantes de aprendizagem de máquina. O interesse por Deep Learning não para de crescer e hoje vemos o termo aprendizado profundo sendo mencionado com frequência cada vez maior e soluções comerciais surgindo a todo momento.

# Capítulo 2 – Uma Breve História das Redes Neurais Artificiais

O desenvolvimento do cérebro humano ocorre principalmente nos dois primeiros anos de vida, mas se arrasta por toda a vida. Inspirando-se neste modelo, diversos pesquisadores tentaram simular o funcionamento do cérebro, principalmente o processo de aprendizagem por experiência, a fim de criar sistemas inteligentes capazes de realizar tarefas como classificação, reconhecimento de padrões, processamento de imagens, entre outras atividades. Como resultado destas pesquisas surgiu o modelo do neurônio artificial e posteriormente um sistema com vários neurônios interconectados, a chamada Rede Neural.

Em 1943, o neurofisiologista Warren McCulloch e o matemático Walter Pitts escreveram um artigo sobre como os neurônios poderiam funcionar e para isso, eles modelaram uma rede neural simples usando circuitos elétricos.

Warren McCulloch e Walter Pitts criaram um modelo computacional para redes neurais baseadas em matemática e algoritmos denominados lógica de limiar (threshold logic). Este modelo abriu o caminho para a pesquisa da rede neural dividida em duas abordagens: uma abordagem focada em processos biológicos no cérebro, enquanto a outra focada na aplicação de redes neurais à inteligência artificial.

Em 1949, Donald Hebb escreveu The Organization of Behavior, uma obra que apontou o fato de que os caminhos neurais são fortalecidos cada vez que são usados, um conceito fundamentalmente essencial para a maneira como os humanos aprendem. Se dois nervos dispararem ao mesmo tempo, argumentou, a conexão entre eles é melhorada.

À medida que os computadores se tornaram mais avançados na década de 1950, finalmente foi possível simular uma hipotética rede neural. O primeiro passo para isso foi feito por Nathanial Rochester dos laboratórios de pesquisa da IBM. Infelizmente para ele, a primeira tentativa de fazê-lo falhou.

No entanto, ao longo deste tempo, os defensores das “máquinas pensantes” continuaram a argumentar suas pesquisas. Em 1956, o Projeto de Pesquisa de Verão de Dartmouth sobre Inteligência Artificial proporcionou um impulso tanto à Inteligência Artificial como às Redes Neurais. Um dos resultados deste processo foi estimular a pesquisa em IA na parte de processamento neural.

Nos anos seguintes ao Projeto Dartmouth, John von Neumann sugeriu imitar funções simples de neurônios usando relés telegráficos ou tubos de vácuo. Além disso, Frank Rosenblatt, um neurobiologista, começou a trabalhar no Perceptron. Ele estava intrigado com o funcionamento do olho de uma mosca. Grande parte do processamento feito por uma mosca ao decidir fugir, é feito em seus olhos. O Perceptron, que resultou dessa pesquisa, foi construído em hardware e é a mais antiga rede neural ainda em uso hoje. Um Percetron de camada única foi útil para classificar um conjunto de entradas de valor contínuo em uma de duas classes. O Perceptron calcula uma soma ponderada das entradas, subtrai um limite e passa um dos dois valores possíveis como resultado. Infelizmente, o Perceptron é limitado e foi comprovado como tal durante os “anos desiludidos” por Marvin Minsky e o livro de Seymour Papert de 1969, Perceptrons.

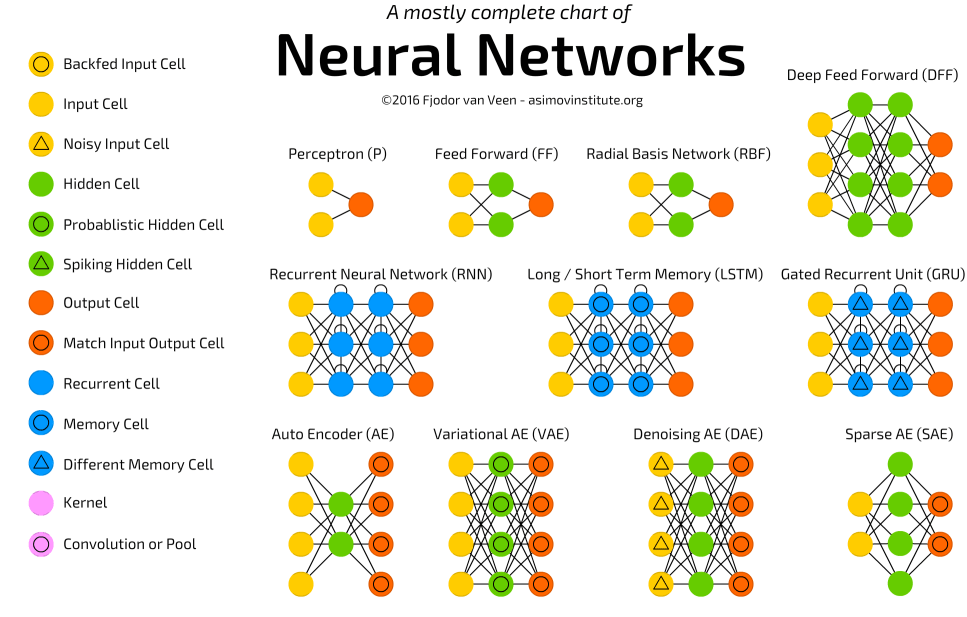


Fig2 – Algumas Arquiteturas de Redes Neurais

Em 1959, Bernard Widrow e Marcian Hoff, de Stanford, desenvolveram modelos denominados “ADALINE” e “MADALINE”. Em uma exibição típica do amor de Stanford por siglas, os nomes provêm do uso de múltiplos elementos ADAptive LINear. ADALINE foi desenvolvido para reconhecer padrões binários de modo que, se ele estivesse lendo bits de transmissão de uma linha telefônica, poderia prever o próximo bit. MADALINE foi a primeira rede neural aplicada a um problema do mundo real, usando um filtro adaptativo que elimina ecos nas linhas telefônicas. Embora o sistema seja tão antigo como os sistemas de controle de tráfego aéreo, ele ainda está em uso comercial.

Infelizmente, esses sucessos anteriores levaram as pessoas a exagerar o potencial das redes neurais, particularmente à luz da limitação na eletrônica, então disponível na época. Este exagero excessivo, que decorreu do mundo acadêmico e técnico, infectou a literatura geral da época. Muitas promessas foram feitas, mas o resultado foi o desapontamento. Além disso, muitos escritores começaram a refletir sobre o efeito que teria “máquinas pensantes” no homem. A série de Asimov em robôs revelou os efeitos sobre a moral e os valores do homem quando máquinas fossem capazes de fazer todo o trabalho da humanidade. Outros escritores criaram computadores mais sinistros, como HAL do filme 2001.

Toda essa discussão sobre o efeito da Inteligência Artificial sobre a vida humana, aliada aos poucos progressos, fizeram vozes respeitadas criticar a pesquisa em redes neurais. O resultado foi a redução drástica de grande parte do financiamento em pesquisas. Esse período de crescimento atrofiado durou até 1981, sendo conhecido como o Inverno da IA (AI Winter).

Em 1982, vários eventos provocaram um renovado interesse. John Hopfield da Caltech apresentou um documento à Academia Nacional de Ciências. A abordagem de Hopfield não era simplesmente modelar cérebros, mas criar dispositivos úteis. Com clareza e análise matemática, ele mostrou como essas redes poderiam funcionar e o que poderiam fazer. No entanto, o maior recurso de Hopfield foi seu carisma. Ele era articulado e simpático e isso colaborou bastante para que ele fosse ouvido.

Em 1985, o Instituto Americano de Física começou o que se tornou uma reunião anual – Redes Neurais para Computação. Em 1987, a primeira Conferência Internacional sobre Redes Neurais do Institute of Electrical and Electronic Engineer’s (IEEE) atraiu mais de 1.800 participantes.

Em 1986, com redes neurais de várias camadas nas notícias, o problema era como estender a regra Widrow-Hoff para várias camadas. Três grupos independentes de pesquisadores, dentre os quais David Rumelhart, ex-membro do departamento de psicologia de Stanford, apresentaram ideias semelhantes que agora são chamadas de redes Backpropagation porque distribuem erros de reconhecimento de padrões em toda a rede. As redes híbridas utilizavam apenas duas camadas, essas redes de Backpropagation utilizam muitas. O resultado é que as redes de Backpropagation “aprendem” de forma mais lenta, pois necessitam, possivelmente, de milhares de iterações para aprender, mas geram um resultado muito preciso.

Agora, as redes neurais são usadas em várias aplicações. A ideia fundamental por trás da natureza das redes neurais é que, se ela funcionar na natureza, deve ser capaz de funcionar em computadores. O futuro das redes neurais, no entanto, reside no desenvolvimento de hardware. As redes neurais rápidas e eficientes dependem do hardware especificado para seu eventual uso.

O diagrama abaixo mostra alguns marcos importantes na evolução e pesquisa das redes neurais artificiais. O fato, é que ainda estamos escrevendo esta história e muita evolução está ocorrendo neste momento, através do trabalho de milhares de pesquisadores e profissionais de Inteligência Artificial em todo mundo. E você, não quer ajudar a escrever esta história?

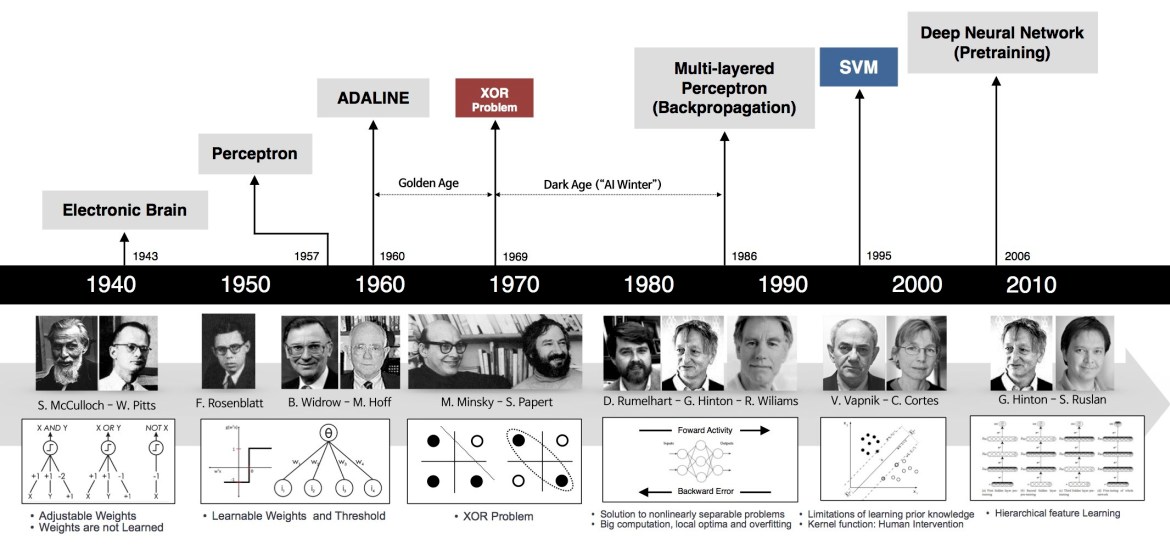


Fig3 – Marcos no desenvolvimento das redes neurais.

Podemos resumir assim os principais marcos na pesquisa e evolução das redes neurais artificiais até chegarmos ao Deep Learning:

**1943**: Warren McCulloch e Walter Pitts criam um modelo computacional para redes neurais baseadas em matemática e algoritmos denominados lógica de limiar.

**1958**: Frank Rosenblatt cria o Perceptron, um algoritmo para o reconhecimento de padrões baseado em uma rede neural computacional de duas camadas usando simples adição e subtração. Ele também propôs camadas adicionais com notações matemáticas, mas isso não seria realizado até 1975.

**1980**: Kunihiko Fukushima propõe a Neoconitron, uma rede neural de hierarquia, multicamada, que foi utilizada para o reconhecimento de caligrafia e outros problemas de reconhecimento de padrões.

**1989**: os cientistas conseguiram criar algoritmos que usavam redes neurais profundas, mas os tempos de treinamento para os sistemas foram medidos em dias, tornando-os impraticáveis ​​para o uso no mundo real.

**1992**: Juyang Weng publica o Cresceptron, um método para realizar o reconhecimento de objetos 3-D automaticamente a partir de cenas desordenadas.

**Meados dos anos 2000**: o termo “aprendizagem profunda” começa a ganhar popularidade após um artigo de Geoffrey Hinton e Ruslan Salakhutdinov mostrar como uma rede neural de várias camadas poderia ser pré-treinada uma camada por vez.

**2009**: acontece o NIPS Workshop sobre Aprendizagem Profunda para Reconhecimento de Voz e descobre-se que com um conjunto de dados suficientemente grande, as redes neurais não precisam de pré-treinamento e as taxas de erro caem significativamente.

**2012**: algoritmos de reconhecimento de padrões artificiais alcançam desempenho em nível humano em determinadas tarefas. E o algoritmo de aprendizagem profunda do Google é capaz de identificar gatos.

**2014**: o Google compra a Startup de Inteligência Artificial chamada DeepMind, do Reino Unido, por £ 400m

**2015**: Facebook coloca a tecnologia de aprendizado profundo – chamada DeepFace – em operação para marcar e identificar automaticamente usuários do Facebook em fotografias. Algoritmos executam tarefas superiores de reconhecimento facial usando redes profundas que levam em conta 120 milhões de parâmetros.

**2016**: o algoritmo do Google DeepMind, AlphaGo, mapeia a arte do complexo jogo de tabuleiro Go e vence o campeão mundial de Go, Lee Sedol, em um torneio altamente divulgado em Seul.

**2017**: adoção em massa do Deep Learning em diversas aplicações corporativas e mobile, além do avanço em pesquisas. Todos os eventos de tecnologia ligados a Data Science, IA e Big Data, apontam Deep Learning como a principal tecnologia para criação de sistemas inteligentes.

A promessa do aprendizado profundo não é que os computadores comecem a pensar como seres humanos. Isso é como pedir uma maçã para se tornar uma laranja. Em vez disso, demonstra que, dado um conjunto de dados suficientemente grande, processadores rápidos e um algoritmo suficientemente sofisticado, os computadores podem começar a realizar tarefas que até então só podiam ser realizadas apenas por seres humanos, como reconhecer imagens e voz, criar obras de arte ou tomar decisões por si mesmo.

Os estudos sobre as redes neurais sofreram uma grande revolução a partir dos anos 80 e esta área de estudos tem se destacado, seja pelas promissoras características apresentadas pelos modelos de redes neurais propostos, seja pelas condições tecnológicas atuais de implementação que permitem desenvolver arrojadas implementações de arquiteturas neurais paralelas em hardwares dedicado, obtendo assim ótimas performances destes sistemas (bastante superiores aos sistemas convencionais). A evolução natural das redes neurais, são as redes neurais profundas (ou Deep Learning). Mas isso é o que vamos discutir no próximo capítulo! Até lá.

Referências:

Christopher D. Manning. (2015). Computational Linguistics and Deep Learning Computational Linguistics, 41(4), 701–707.

F. Rosenblatt. The perceptron, a perceiving and recognizing automaton Project Para. Cornell Aeronautical Laboratory, 1957.

W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. The

# Capítulo 3 – O Que São Redes Neurais Artificiais Profundas ou Deep Learning?

veremos neste capítulo. Não se preocupe se alguns termos mais técnicos não fizerem sentido agora. Todos eles serão estudados ao longo deste livro online.

Deep Learning  usa camadas de neurônios matemáticos para processar dados, compreender a fala humana e reconhecer objetos visualmente. A informação é passada através de cada camada, com a saída da camada anterior fornecendo entrada para a próxima camada. A primeira camada em uma rede é chamada de camada de entrada, enquanto a última é chamada de camada de saída. Todas as camadas entre as duas são referidas como camadas ocultas. Cada camada é tipicamente um algoritmo simples e uniforme contendo um tipo de função de ativação.

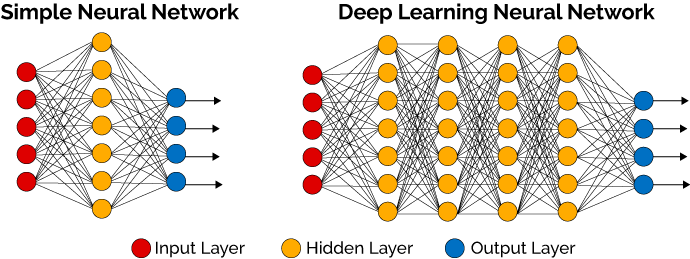


Fig4 – Rede Neural Simples e Rede Neural Profunda (Deep Learning)

A aprendizagem profunda é responsável por avanços recentes em visão computacional, reconhecimento de fala, processamento de linguagem natural e reconhecimento de áudio. O aprendizado profundo é baseado no conceito de redes neurais artificiais, ou sistemas computacionais que imitam a maneira como o cérebro humano funciona.

A extração de recursos é outro aspecto da Aprendizagem Profunda. A extração de recursos usa um algoritmo para construir automaticamente “recursos” significativos dos dados para fins de treinamento, aprendizado e compreensão. Normalmente, o Cientista de Dados, ou Engenheiro de IA, é responsável pela extração de recursos.

O aumento rápido e o aparente domínio do aprendizado profundo sobre os métodos tradicionais de aprendizagem de máquina em uma variedade de tarefas tem sido surpreendente de testemunhar e, às vezes, difícil de explicar. Deep Learning é uma evolução das Redes Neurais, que por sua vez possuem uma história fascinante que remonta à década de 1940, cheia de altos e baixos, voltas e reviravoltas, amigos e rivais, sucessos e fracassos. Em uma história digna de um filme dos anos 90, uma ideia que já foi uma espécie de patinho feio floresceu para se tornar a bola da vez.

Consequentemente, o interesse em aprendizagem profunda tem disparado, com cobertura constante na mídia popular. A pesquisa de aprendizagem profunda agora aparece rotineiramente em revistas como Science, Nature, Nature Methods e Forbes apenas para citar alguns. O aprendizado profundo conquistou Go, aprendeu a dirigir um carro, diagnosticou câncer de pele e autismo, tornou-se um falsificador de arte mestre e pode até alucinar imagens fotorrealistas.

Os primeiros algoritmos de aprendizagem profunda que possuíam múltiplas camadas de características não-lineares podem ser rastreados até Alexey Grigoryevich Ivakhnenko (desenvolveu o Método do Grupo de Manipulação de Dados) e Valentin Grigor’evich Lapa (autor de Cybernetics and Forecasting Techniques) em 1965 (Figura 5), que usaram modelos finos mas profundos com funções de ativação polinomial os quais eles analisaram com métodos estatísticos. Em cada camada, eles selecionavam os melhores recursos através de métodos estatísticos e encaminhavam para a próxima camada. Eles não usaram Backpropagation para treinar a rede de ponta a ponta, mas utilizaram mínimos quadrados camada-por-camada, onde as camadas anteriores foram independentemente instaladas em camadas posteriores (um processo lento e manual).

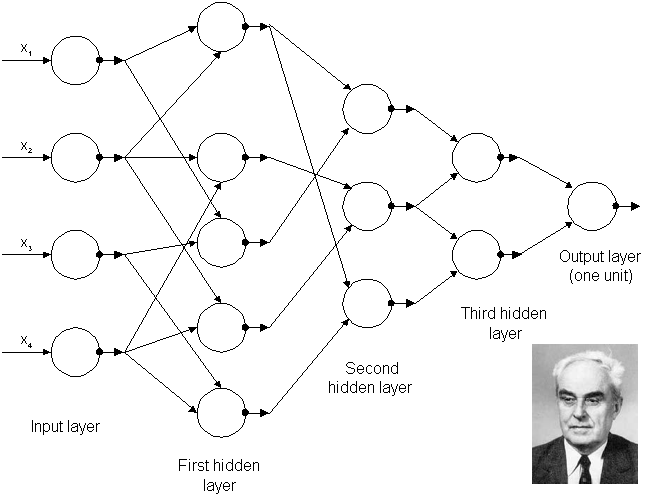


Fig5 – Arquitetura da primeira rede profunda conhecida treinada por Alexey Grigorevich Ivakhnenko em 1965.

No final da década de 1970, o primeiro inverno de AI começou, resultado de promessas que não poderiam ser mantidas. O impacto desta falta de financiamento limitou a pesquisa em Redes Neurais Profundas e Inteligência Artificial. Felizmente, houve indivíduos que realizaram a pesquisa sem financiamento.

As primeiras “redes neurais convolutivas” foram usadas por Kunihiko Fukushima. Fukushima concebeu redes neurais com múltiplas camadas de agrupamento e convoluções. Em 1979, ele desenvolveu uma rede neural artificial, chamada Neocognitron, que usava um design hierárquico e multicamadas. Este design permitiu ao computador “aprender” a reconhecer padrões visuais. As redes se assemelhavam a versões modernas, mas foram treinadas com uma estratégia de reforço de ativação recorrente em múltiplas camadas, que ganhou força ao longo do tempo. Além disso, o design de Fukushima permitiu que os recursos importantes fossem ajustados manualmente aumentando o “peso” de certas conexões.

Muitos dos conceitos de Neocognitron continuam a ser utilizados. O uso de conexões de cima para baixo e novos métodos de aprendizagem permitiram a realização de uma variedade de redes neurais. Quando mais de um padrão é apresentado ao mesmo tempo, o Modelo de Atenção Seletiva pode separar e reconhecer padrões individuais deslocando sua atenção de um para o outro (o mesmo processo que usamos em multitarefa). Um Neocognitron moderno não só pode identificar padrões com informações faltantes (por exemplo, um número 5 desenhado de maneira incompleta), mas também pode completar a imagem adicionando as informações que faltam. Isso pode ser descrito como “inferência”.

O Backpropagation, o uso de erros no treinamento de modelos de Deep Learning, evoluiu significativamente em 1970. Foi quando Seppo Linnainmaa escreveu sua tese de mestrado, incluindo um código FORTRAN para Backpropagation. Infelizmente, o conceito não foi aplicado às redes neurais até 1985. Foi quando Rumelhart, Williams e Hinton demonstraram o Backpropagation em uma rede neural que poderia fornecer representações de distribuição “interessantes”. Filosoficamente, essa descoberta trouxe à luz a questão dentro da psicologia cognitiva de saber se a compreensão humana depende da lógica simbólica (computacionalismo) ou de representações distribuídas (conexão). Em 1989, Yann LeCun forneceu a primeira demonstração prática de Backpropagation no Bell Labs. Ele combinou redes neurais convolutivas com Backpropagation para ler os dígitos “manuscritos” (assunto do próximo capítulo). Este sistema foi usado para ler o número de cheques manuscritos.



Fig6 – Os pioneiros da Inteligência Artificial. Da esquerda para a direita: Yann LeCun, Geoffrey Hinton, Yoshua Bengio e Andrew Ng

Porém, tivemos neste período o que ficou conhecido como segundo Inverno da IA, que ocorreu entre 1985-1990, que também afetou pesquisas em Redes Neurais e Aprendizagem Profunda. Vários indivíduos excessivamente otimistas haviam exagerado o potencial “imediato” da Inteligência Artificial, quebrando as expectativas e irritando os investidores. A raiva era tão intensa, que a frase Inteligência Artificial atingiu o status de pseudociência. Felizmente, algumas pessoas continuaram trabalhando em IA e Deep Learning, e alguns avanços significativos foram feitos. Em 1995, Dana Cortes e Vladimir Vapnik desenvolveram a máquina de vetor de suporte ou Support Vector Machine (um sistema para mapear e reconhecer dados semelhantes). O LSTM (Long-Short Term Memory) para redes neurais recorrentes foi desenvolvido em 1997, por Sepp Hochreiter e Juergen Schmidhuber.

O próximo passo evolutivo significativo para Deep Learning ocorreu em 1999, quando os computadores começaram a se tornar mais rápidos no processamento de dados e GPUs (unidades de processamento de gráfico) foram desenvolvidas. O uso de GPUs significou um salto no tempo de processamento, resultando em um aumento das velocidades computacionais em 1000 vezes ao longo de um período de 10 anos. Durante esse período, as redes neurais começaram a competir com máquinas de vetor de suporte. Enquanto uma rede neural poderia ser lenta em comparação com uma máquina de vetor de suporte, as redes neurais ofereciam melhores resultados usando os mesmos dados. As redes neurais também têm a vantagem de continuar a melhorar à medida que mais dados de treinamento são adicionados.

Em torno do ano 2000, apareceu o problema conhecido como Vanishing Gradient. Foi descoberto que as “características” aprendidas em camadas mais baixas não eram aprendidas pelas camadas superiores, pois nenhum sinal de aprendizado alcançou essas camadas. Este não era um problema fundamental para todas as redes neurais, apenas aquelas com métodos de aprendizagem baseados em gradientes. A origem do problema acabou por ser certas funções de ativação. Uma série de funções de ativação condensavam sua entrada, reduzindo, por sua vez, a faixa de saída de forma um tanto caótica. Isso produziu grandes áreas de entrada mapeadas em uma faixa extremamente pequena. Nessas áreas de entrada, uma grande mudança será reduzida a uma pequena mudança na saída, resultando em um gradiente em queda. Duas soluções utilizadas para resolver este problema foram o pré-treino camada-a-camada e o desenvolvimento de uma memória longa e de curto prazo.

Em 2001, um relatório de pesquisa do Grupo META (agora chamado Gartner) descreveu os desafios e oportunidades no crescimento do volume de dados. O relatório descreveu o aumento do volume de dados e a crescente velocidade de dados como o aumento da gama de fontes e tipos de dados. Este foi um apelo para se preparar para a investida do Big Data, que estava apenas começando.

Em 2009, Fei-Fei Li, professora de IA em Stanford na Califórnia, lançou o ImageNet e montou uma base de dados gratuita de mais de 14 milhões de imagens etiquetadas. Eram necessárias imagens marcadas para “treinar” as redes neurais. A professora Li disse: “Nossa visão é que o Big Data mudará a maneira como a aprendizagem de máquina funciona. Data drives learning.”. Ela acertou em cheio!

Até 2011, a velocidade das GPUs aumentou significativamente, possibilitando a formação de redes neurais convolutivas “sem” o pré-treino camada por camada. Com o aumento da velocidade de computação, tornou-se óbvio que Deep Learning tinha vantagens significativas em termos de eficiência e velocidade. Um exemplo é a AlexNet, uma rede neural convolutiva, cuja arquitetura ganhou várias competições internacionais durante 2011 e 2012. As unidades lineares retificadas foram usadas para melhorar a velocidade.

Também em 2012, o Google Brain lançou os resultados de um projeto incomum conhecido como The Cat Experiment. O projeto de espírito livre explorou as dificuldades de “aprendizagem sem supervisão”. A Aprendizagem profunda usa “aprendizagem supervisionada”, o que significa que a rede neural convolutiva é treinada usando dados rotulados. Usando a aprendizagem sem supervisão, uma rede neural convolucional é alimentada com dados não marcados, e é então solicitada a busca de padrões recorrentes.

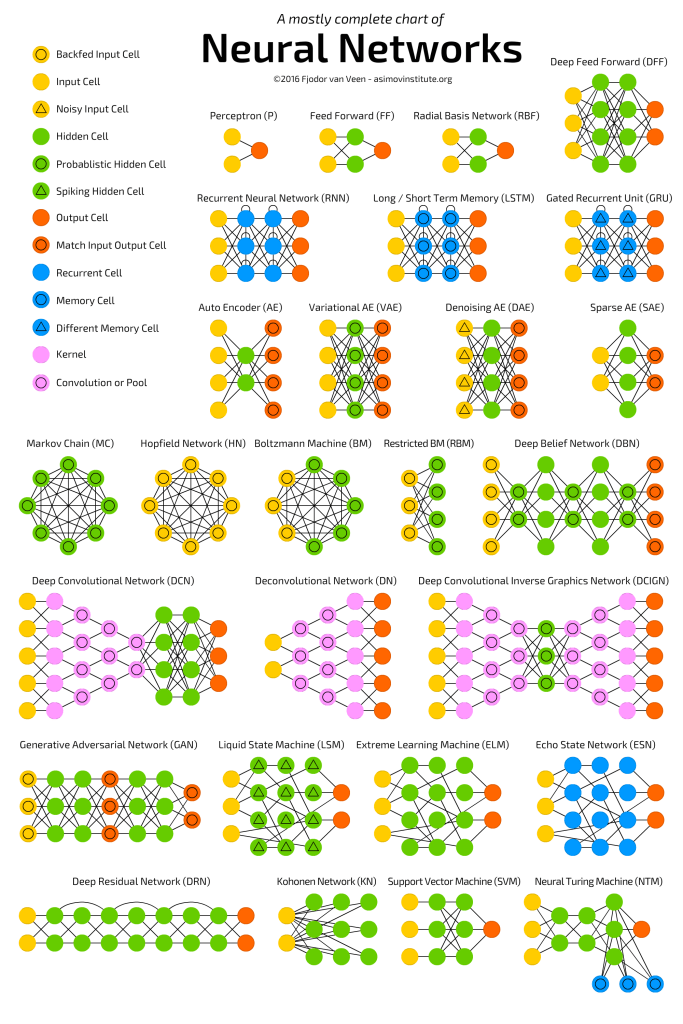
O Cat Experiment usou uma rede neural distribuída por mais de 1.000 computadores. Dez milhões de imagens “sem etiqueta” foram tiradas aleatoriamente do YouTube, mostradas ao sistema e, em seguida, o software de treinamento foi autorizado a ser executado. No final do treinamento, um neurônio na camada mais alta foi encontrado para responder fortemente às imagens de gatos. Andrew Ng, o fundador do projeto, disse: “Nós também encontramos um neurônio que respondeu fortemente aos rostos humanos”. A aprendizagem não supervisionada continua a ser um um campo ativo de pesquisa em Aprendizagem Profunda.

Atualmente, o processamento de Big Data e a evolução da Inteligência Artificial são ambos dependentes da Aprendizagem Profunda. Com Deep Learning podemos construir sistemas inteligentes e estamos nos aproximando da criação de uma IA totalmente autônoma. Isso vai gerar impacto em todas os segmentos da sociedade e aqueles que souberem trabalhar com a tecnologia, serão os líderes desse novo mundo que se apresenta diante de nós.

No próximo capítulo você vai começar a compreender tecnicamente como funciona a Aprendizagem Profunda. Até o capítulo 4.

# Capítulo 6 – O Perceptron – Parte 1

Você sabe quais são as principais arquiteturas de redes neurais artificias? Não. Então analise cuidadosamente a imagem abaixo (excelente trabalho criado pela equipe do Asimov Institute, cujo link você encontra na seção de referências ao final deste capítulo):

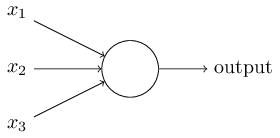


Incrível, não? São diversas arquiteturas, usadas para resolver diferentes tipos de problemas, como por exemplo as arquiteturas de redes neurais convolucionais usadas em problemas de Visão Computacional e as redes neurais recorrentes usadas em problemas de Processamento de Linguagem Natural. Estudaremos quase todas essas arquiteturas aqui neste livro. Sim, isso mesmo que você leu. Estamos apenas começando!! Caso queira aprender a construir modelos e projetos usando essas arquiteturas e trabalhando com linguagem Python e Google TensorFlow, clique [aqui](http://deeplearningbook.com.br/cursos-online/).

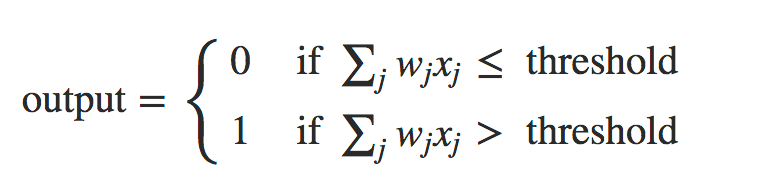
Embora todas essas arquiteturas sejam de redes neurais artificias, nem todas são de Deep Learning. O que caracteriza modelos de aprendizagem profunda, como o nome sugere, são redes neurais artificias com muitas camadas ocultas (ou intermediárias). Mas antes de chegarmos lá, precisamos passar pela arquitetura mais simples de uma rede neural artificial, o Perceptron. Como diz o ditado: “Toda grande caminhada começa pelo primeiro passo”.

O Modelo Perceptron foi desenvolvido nas décadas de 1950 e 1960 pelo cientista Frank Rosenblatt, inspirado em trabalhos anteriores de Warren McCulloch e Walter Pitts. Hoje, é mais comum usar outros modelos de neurônios artificiais, mas o Perceptron permite uma compreensão clara de como funciona uma rede neural em termos matemáticos, sendo uma excelente introdução.

Então, como funcionam os Perceptrons? Um Perceptron é um modelo matemático que recebe várias entradas, x1, x2, … e produz uma única saída binária:



No exemplo mostrado, o Perceptron possui três entradas: x1, x2, x3. Rosenblatt propôs uma regra simples para calcular a saída. Ele introduziu pesos, w1, w2, …, números reais expressando a importância das respectivas entradas para a saída. A saída do neurônio, 0 ou 1, é determinada pela soma ponderada, **Σjwjxj**, menor ou maior do que algum valor limiar (threshold). Assim como os pesos, o threshold é um número real que é um parâmetro do neurônio. Para colocá-lo em termos algébricos mais precisos:



Esse é o modelo matemático básico. Uma maneira de pensar sobre o Perceptron é que é um dispositivo que toma decisões ao comprovar evidências. Deixe-me dar um exemplo. Não é um exemplo muito realista, mas é fácil de entender, e logo chegaremos a exemplos mais realistas. Suponha que o fim de semana esteja chegando e você ouviu falar que haverá um festival de queijo em sua cidade. Você gosta de queijo e está tentando decidir se deve ou não ir ao festival. Você pode tomar sua decisão pesando três fatores:

* O tempo está bom?
* Seu namorado ou namorada quer acompanhá-lo(a)?
* O festival está perto de transporte público? (Você não possui um carro)

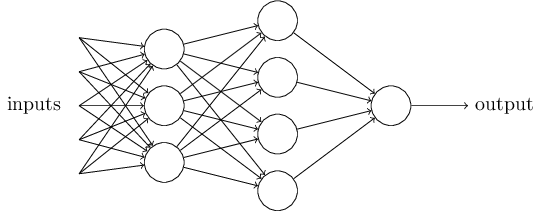
Podemos representar estes três fatores pelas variáveis binárias correspondentes x1, x2 e x3. Por exemplo, teríamos x1 = 1 se o tempo estiver bom e x1 = 0 se o tempo estiver ruim. Da mesma forma, x2 = 1 se seu namorado ou namorada quiser ir ao festival com você, e x2 = 0, se não. E similarmente para x3 e transporte público.

Agora, suponha que você adore queijo e está disposto a ir ao festival, mesmo que seu namorado ou namorada não esteja interessado e o festival fica em um lugar de difícil acesso e sem transporte público amplamente disponível. Além disso, você realmente detesta mau tempo, e não há como ir ao festival se o tempo estiver ruim. Você pode usar Perceptrons para modelar esse tipo de tomada de decisão.

Uma maneira de fazer isso é escolher um peso w1 = 6 para o tempo e w2 = 2 e w3 = 2 para as outras condições. O valor maior de w1 indica que o tempo é muito importante para você, muito mais do que se seu namorado ou namorada vai acompanhá-lo(a) ou se o festival é próximo do transporte público. Finalmente, suponha que você escolha um threshold de 5 para o Perceptron. Com essas escolhas, o Perceptron implementa o modelo de tomada de decisão desejado, produzindo 1 sempre que o tempo estiver bom e 0 sempre que o tempo estiver ruim. Não faz diferença para o resultado se seu namorado ou namorada quer ir, ou se o transporte público está acessível.

Variando os pesos e o limiar, podemos obter diferentes modelos de tomada de decisão. Por exemplo, suponha que escolhemos um threshold de 3. Então, o Perceptron decidirá que você deveria ir ao festival sempre que o tempo estiver bom ou quando o festival estiver perto do transporte público e seu namorado ou namorada estiver disposto a se juntar a você. Em outras palavras, seria um modelo diferente de tomada de decisão. Reduzir o threshold significa que você está mais propenso a ir ao festival.

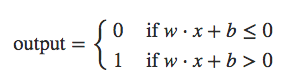
Obviamente, o Perceptron não é um modelo completo de tomada de decisão humana! Mas o que o exemplo ilustra é como um Perceptron pode pesar diferentes tipos de evidências para tomar decisões. E deve parecer plausível que uma rede complexa de Perceptrons possa tomar decisões bastante sutis.



Nesta rede, a primeira coluna de Perceptrons – o que chamaremos de primeira camada de Perceptrons – está tomando três decisões muito simples, pesando a evidência de entrada. E quanto aos Perceptrons na segunda camada? Cada um desses Perceptrons está tomando uma decisão ponderando os resultados da primeira camada de tomada de decisão. Desta forma, um Perceptron na segunda camada pode tomar uma decisão em um nível mais complexo e mais abstrato do que os Perceptrons na primeira camada. E as decisões ainda mais complexas podem ser feitas pelos Perceptrons na terceira camada. Desta forma, uma rede de Perceptrons de várias camadas pode envolver-se em uma tomada de decisão sofisticada.

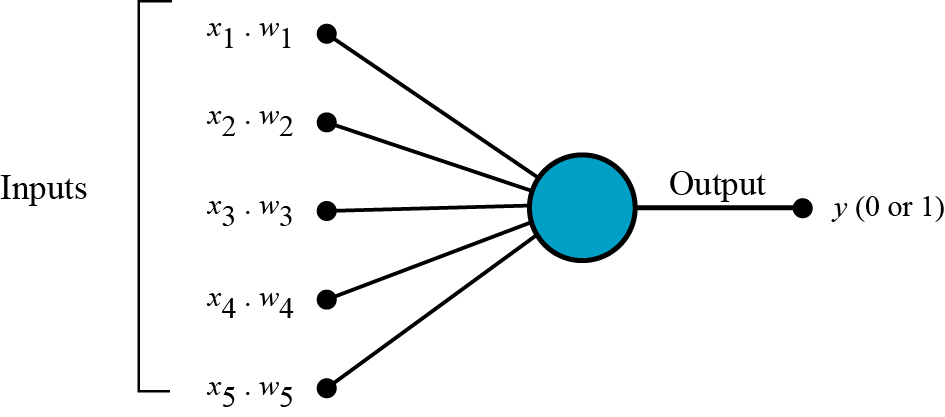
Aliás, quando definimos os Perceptrons, dissemos que um Perceptron possui apenas uma saída. Na rede acima, os Perceptrons parecem ter múltiplos resultados. Na verdade, eles ainda são de saída única. As setas de saída múltiplas são meramente uma maneira útil de indicar que a saída de um Perceptron está sendo usada como entrada para vários outros Perceptrons.

Vamos simplificar a maneira como descrevemos os Perceptrons. No limite de condição **Σjwjxj > threshold** podemos fazer duas mudanças de notação para simplificá-lo. A primeira mudança é escrever Σjwjxj como um produto (dot product), w⋅x≡Σjwjxj, onde w e x são vetores cujos componentes são os pesos e entradas, respectivamente. A segunda mudança é mover o threshold para o outro lado da equação e substituí-lo pelo que é conhecido como o viés (bias) do Perceptron, ou b ≡ -threshold. Usando o viés em vez do threshold, a regra Perceptron pode ser reescrita:



Você pode pensar no viés como uma medida de quão fácil é obter o Perceptron para produzir um 1. Ou para colocá-lo em termos mais biológicos, o viés é uma medida de quão fácil é fazer com que o Perceptron dispare. Para um Perceptron com um viés realmente grande, é extremamente fácil para o Perceptron emitir um 1. Mas se o viés é muito negativo, então é difícil para o Perceptron emitir um 1. Obviamente, a introdução do viés é apenas uma pequena mudança em como descrevemos Perceptrons, mas veremos mais adiante que isso leva a outras simplificações de notação. Por isso, no restante do livro, não usaremos o threshold, usaremos sempre o viés.

Agora começa a ficar mais fácil compreender o conceito por trás das redes neurais artificiais e isso será muito útil quando estudarmos arquiteturas mais avançadas! Um Perceptron segue o modelo “feed-forward”, o que significa que as entradas são enviadas para o neurônio, processadas e resultam em uma saída. No diagrama abaixo, isso significa que a rede (um neurônio) lê da esquerda para a direita.



O processo de treinamento de um modelo Perceptron consiste em fazer com que o modelo aprenda os valores ideais de pesos e bias. Apresentamos ao modelo os dados de entrada e as possíveis saídas, treinamos o modelo e pesos e bias são aprendidos. Com o modelo treinado, podemos apresentar novos dados de entrada e o modelo será capaz de prever a saída. Veremos isso em breve quando criarmos nosso primeiro modelo usando linguagem Python.

Perceptron é uma rede neural de camada única e um Perceptron de várias camadas é chamado de Rede Neural Artificial. O Perceptron é um classificador linear (binário). Além disso, é usado na aprendizagem supervisionada e pode ser usado para classificar os dados de entrada fornecidos.

Mas o Perceptron tem ainda outras características importantes, como a representação de condicionais lógicos (and, or, xor), problemas com dados não linearmente separáveis e as funções de ativação. Mas esses são temas para o próximo capítulo. Até lá!

# Capítulo 7 – O Perceptron – Parte 2

O Perceptron é um modelo matemático de um neurônio biológico. Enquanto nos neurônios reais o dendrito recebe sinais elétricos dos axônios de outros neurônios, no Perceptron estes sinais elétricos são representados como valores numéricos. Nas sinapses entre dendritos e axônio, os sinais elétricos são modulados em várias quantidades. Isso também é modelado no Perceptron multiplicando cada valor de entrada por um valor chamado peso. Um neurônio real dispara um sinal de saída somente quando a força total dos sinais de entrada excede um certo limiar. Nós modelamos esse fenômeno em um Perceptron calculando a soma ponderada das entradas para representar a força total dos sinais de entrada e aplicando uma função de ativação na soma para determinar sua saída. Tal como nas redes neurais biológicas, esta saída é alimentada em outros Perceptrons. Estudamos tudo isso no capítulo anterior. Agora vamos continuar nossa discussão sobre o Perceptron compreendendo mais alguns conceitos, que serão fundamentais mais a frente quando estudarmos as arquiteturas avançadas de Deep Learning.

Antes de iniciar, vamos definir dois conceitos que você vai ver com frequência daqui em diante, vetor de entrada e vetor de pesos:

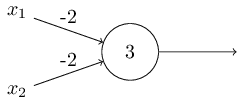
**Vetor de entrada** –  todos os valores de entrada de cada Perceptron são coletivamente chamados de vetor de entrada desse Perceptron. Esses são seus dados de entrada.

**Vetor de pesos** – de forma semelhante, todos os valores de peso de cada Perceptron são coletivamente chamados de vetor de peso desse Perceptron. Iniciamos nossa rede neural artificial com valores aleatórios de pesos e durante o treinamento a rede neural aprende os valores de peso ideais. Como veremos, existem muitas formas de realizar esse processo.

Boa parte do trabalho de uma rede neural vai girar em torno das operações algébricas entre o vetor de entrada e o vetor de pesos. Em seguida, vamos adicionando outras camadas matemáticas ou estatísticas para realizar diferentes operações, de acordo com o problema que estamos tentando resolver com o modelo de rede neural. Você vai perceber que tudo não passa de Matemática, que pode ser implementada com linguagens de programação, grandes conjuntos de dados e processamento paralelo, para formar sistemas de Inteligência Artificial.

### Mas o que um Perceptron pode fazer afinal?

[No capítulo anterior](http://deeplearningbook.com.br/capitulo-6-o-perceptron-parte-1/) descrevemos os Perceptrons como um método para pesar evidências a fim de tomar decisões. Outra forma em que os Perceptrons podem ser usados é para calcular as funções lógicas elementares tais como AND, OR e NAND (caso tenha dúvidas sobre as operações lógicas, consulte as referências ao final deste capítulo). Por exemplo, suponha que tenhamos um Perceptron com duas entradas, cada uma com peso -2 e um viés de 3. Aqui está o nosso Perceptron:



Então vemos que a entrada 00 produziria a saída 1, uma vez que (-2) \* 0 + (- 2) \* 0 + 3 = 3, é positivo (resultado positivo, gera saída 1 do Perceptron, lembra do capítulo anterior?). Aqui, incluímos o símbolo \* para tornar as multiplicações explícitas. Cálculos similares mostram que as entradas 01 e 10 produzem a saída 1. Mas a entrada 11 produz a saída 0, uma vez que (-2) \* 1 + (- 2) \* 1 + 3 = -1, é negativo. E assim nosso Perceptron implementa um “portão” NAND, ou uma operação lógica binária NAND.

O exemplo NAND mostra que podemos usar Perceptrons para calcular funções lógicas simples. Na verdade, podemos usar redes de Perceptrons para calcular qualquer função lógica. A razão é que o portão NAND é universal para computação, ou seja, podemos construir qualquer computação com portões NAND.

Uma rede de Perceptrons pode ser usada para simular um circuito contendo muitos portões NAND. E como os portões NAND são universais para a computação, segue-se que os Perceptrons também são universais para a computação. Considerando que o Perceptron é o modelo mais simples de rede neural, imagine o que pode ser feito com modelos bem mais avançados! Acertou se você pensou em Inteligência Artificial.

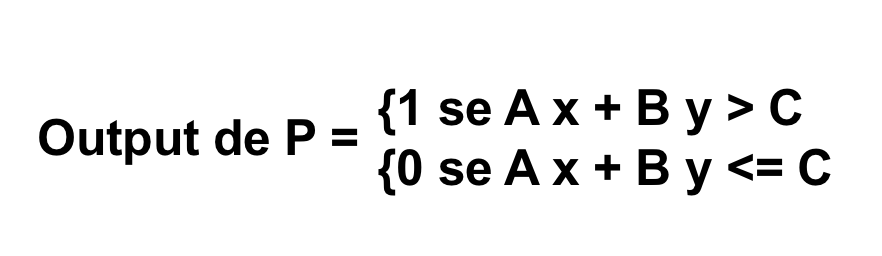
A universalidade computacional dos Perceptrons é simultaneamente reconfortante e decepcionante. É reconfortante porque nos diz que redes de Perceptrons podem ser tão poderosas como qualquer outro dispositivo de computação. Mas também é decepcionante, porque parece que os Perceptrons são meramente um novo tipo de portão NAND. Isso não é uma grande noticia!

No entanto, a situação é melhor do que esta visão sugere. Acontece que podemos conceber algoritmos de aprendizado que podem ajustar automaticamente os pesos e os vieses de uma rede de neurônios artificiais. Este ajuste ocorre em resposta a estímulos externos, sem intervenção direta de um programador. Esses algoritmos de aprendizagem nos permitem usar neurônios artificiais de uma maneira que é radicalmente diferente dos portões lógicos convencionais. Em vez de colocar explicitamente um circuito de NAND e outros portões, nossas redes neurais podem simplesmente aprender a resolver problemas, às vezes problemas em que seriam extremamente difíceis de projetar diretamente usando um circuito convencional de lógica.

### Operações Lógicas e Regiões Linearmente Separáveis

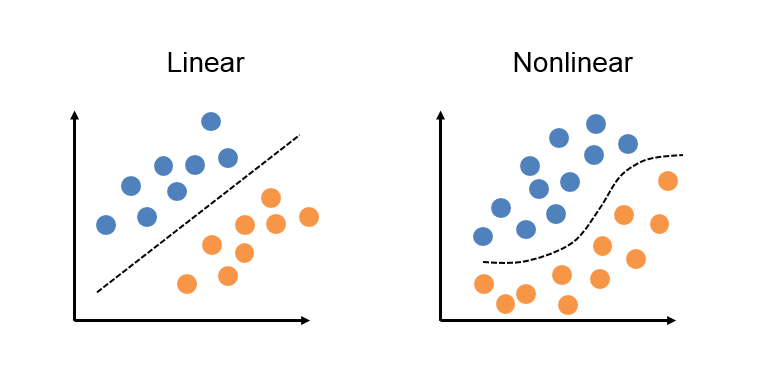
Conforme mencionado acima, um Perceptron calcula a soma ponderada dos valores de entrada. Por simplicidade, suponhamos que existem dois valores de entrada, x e y para um certo Perceptron P. Vamos definir os pesos de x e y, como sendo A e B, respectivamente. A soma ponderada pode ser representada como: A x + B y.

Uma vez que o Perceptron produz um valor não-zero somente quando a soma ponderada excede um certo limite C, pode-se escrever a saída deste Perceptron da seguinte maneira:



Considerando que A x + B y > C e A x + B y < C são as duas regiões no plano xy separadas pela linha A x + B y + C = 0, e se considerarmos ainda a entrada (x, y) como um ponto em um plano, então o Perceptron realmente nos diz qual região no plano a que esse ponto pertence. Tais regiões, uma vez que são separadas por uma única linha, são chamadas de regiões linearmente separáveis.

Um único Perceptron consegue resolver somente funções linearmente separáveis. Em funções não linearmente separáveis, o Perceptron não consegue gerar um hiperplano, esta linha nos gráficos abaixo, para separar os dados. A questão é que no mundo real raramente os dados são linearmente separáveis, fazendo com o que o Perceptron não seja muito útil para atividades práticas (mas sendo ideal para iniciar o estudo em redes neurais artificiais). E como separamos os dados não linearmente separáveis? Continue acompanhando este livro e você irá descobrir.

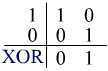


Mas ainda assim o Perceptron tem sua utilidade, porque resulta em algumas funções lógicas, como os operadores booleanos AND, OR e NOT, que são linearmente separáveis, isto é, eles podem ser realizadas usando um único Perceptron. Podemos ilustrar porque eles são linearmente separáveis ao traçar cada um deles em um gráfico:

Funções Lógicas

Nos gráficos acima, os dois eixos são as entradas que podem levar o valor de 0 ou 1 e os números no gráfico são a saída esperada para uma entrada específica. Usando um vetor de peso apropriado para cada caso, um único Perceptron pode executar todas essas funções.

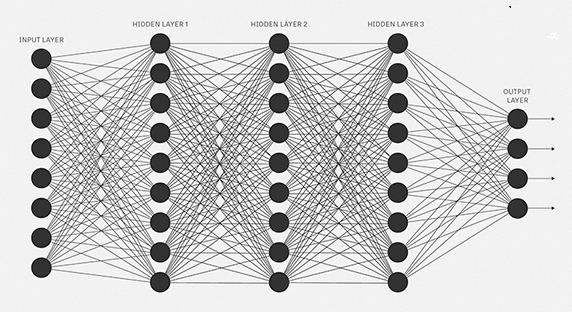
No entanto, nem todos os operadores de lógica são linearmente separáveis. Por exemplo, o operador XOR não é linearmente separável e não pode ser alcançado por um único Perceptron. No entanto, esse problema poderia ser superado usando mais de um Perceptron organizado em redes neurais feed-forward, que veremos mais a frente nos próximos capítulos.



Uma vez que é impossível desenhar uma linha para dividir as regiões contendo 1 ou 0, a função XOR não é linearmente separável, conforme pode ser visto no gráfico acima.

Agora fica mais fácil compreender porque precisamos de arquiteturas mais avançadas de redes neurais artificiais, uma vez que temos problemas complexos no mundo real, como Visão Computacional, Processamento de Linguagem Natural, Tradução, Detecção de Fraudes, Classificação e muitos outros. E veremos essas arquiteturas em detalhes. Mas antes, precisamos falar sobre um componente fundamental das redes neurais, a Função de Ativação. Não perca o próximo capítulo. Até lá.

# Capítulo 8 – Função de Ativação

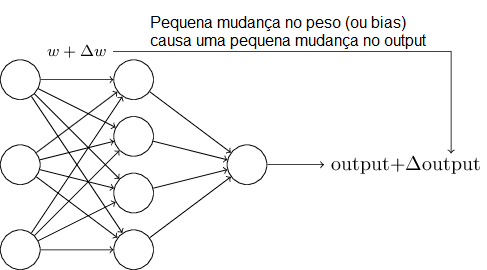


Os círculos negros na imagem acima são neurônios. Cada neurônio é caracterizado pelo peso, bias e a função de ativação. Os dados de entrada são alimentados na camada de entrada. Os neurônios fazem uma transformação linear na entrada pelos pesos e bias. A transformação não linear é feita pela função de ativação. A informação se move da camada de entrada para as camadas ocultas. As camadas ocultas fazem o processamento e enviam a saída final para a camada de saída. Este é o movimento direto da informação conhecido como propagação direta. Mas e se o resultado gerado estiver longe do valor esperado? Em uma rede neural, atualizaríamos os pesos e bias dos neurônios com base no erro. Este processo é conhecido como backpropagation. Uma vez que todos os dados passaram por este processo, os pesos e bias finais são usados para previsões.

Calma, calma, calma. Muita informação em um único parágrafo, eu sei! Vamos por partes. As entradas, os pesos e bias nós já discutimos nos capítulos anteriores. A função de ativação vamos discutir agora e a propagação direta e o backpropagation discutimos nos próximos capítulos!

### Função de Ativação

Os algoritmos de aprendizagem são fantásticos. Mas como podemos elaborar esses algoritmos para uma rede neural artificial? Suponhamos que tenhamos uma rede de Perceptrons que gostaríamos de usar para aprender a resolver algum problema. Por exemplo, as entradas para a rede poderiam ser os dados de pixel de uma imagem digitalizada, escrita à mão, de um dígito. Gostaríamos que a rede aprendesse pesos e bias para que a saída da rede classifique corretamente o dígito. Para ver como a aprendizagem pode funcionar, suponha que façamos uma pequena alteração em algum peso (ou bias) na rede. O que queremos é que esta pequena mudança de peso cause apenas uma pequena alteração correspondente na saída da rede. Como veremos em um momento, esta propriedade tornará possível a aprendizagem. Esquematicamente, aqui está o que queremos (obviamente, esta rede é muito simples para fazer reconhecimento de escrita, mas fique tranquilo que veremos redes bem mais complexas).

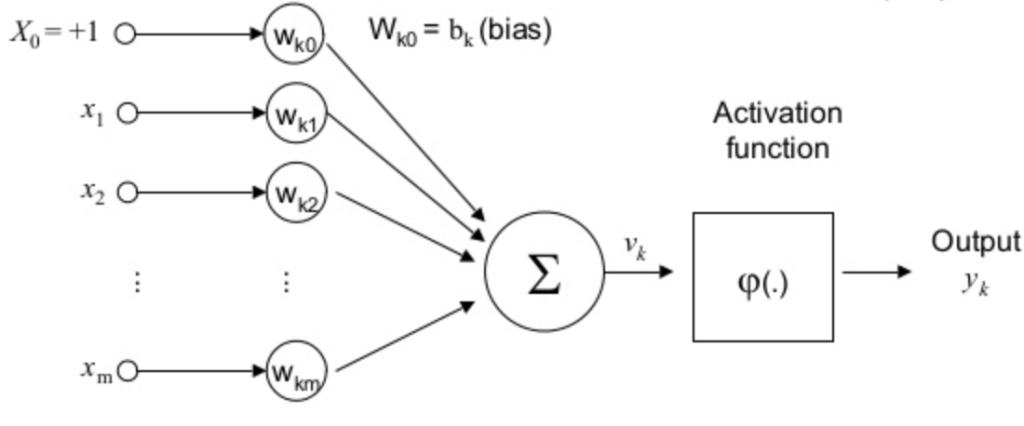


Se fosse verdade que uma pequena alteração em um peso (ou bias) fizesse com que tivéssemos apenas uma pequena alteração no resultado, então poderíamos usar esse fato para modificar os pesos e os valores de bias para que a nossa rede se comporte mais da maneira que queremos. Por exemplo, suponha que a rede classificasse equivocadamente uma imagem como “8” quando deveria ser um “9”. Podemos descobrir como fazer uma pequena mudança nos pesos e bias para que a rede fique um pouco mais próxima da classificação da imagem como “9”. E então, repetiríamos isso, mudando os pesos e os valores de bias repetidamente para produzir melhor e melhor resultado. A rede estaria aprendendo.

O problema é que isso não é o que acontece quando nossa rede contém apenas Perceptrons, conforme estudamos nos capítulos anteriores. De fato, uma pequena alteração nos pesos de um único Perceptron na rede pode, por vezes, fazer com que a saída desse Perceptron mude completamente, digamos de 0 a 1. Essa mudança pode então causar o comportamento do resto da rede mudar completamente de uma maneira muito complicada. Então, enquanto o seu “9” pode agora ser classificado corretamente, o comportamento da rede em todas as outras imagens provavelmente mudará completamente de maneira difícil de controlar. Talvez haja uma maneira inteligente de resolver esse problema. Sim, há. E é conhecida como função de ativação.

Podemos superar esse problema através da introdução de um componente matemático em nosso neurônio artificial, chamado função de ativação. As funções de ativação permitem que pequenas mudanças nos pesos e bias causem apenas uma pequena alteração no output. Esse é o fato crucial que permitirá que uma rede de neurônios artificiais aprenda.

Vejamos como isso funciona:



As funções de ativação são um elemento extremamente importante das redes neurais artificiais. Elas basicamente decidem se um neurônio deve ser ativado ou não. Ou seja, se a informação que o neurônio está recebendo é relevante para a informação fornecida ou deve ser ignorada. Veja na fórmula abaixo como a função de ativação é mais uma camada matemática no processamento.

Função de Ativação

A função de ativação é a transformação não linear que fazemos ao longo do sinal de entrada. Esta saída transformada é então enviada para a próxima camada de neurônios como entrada. Quando não temos a função de ativação, os pesos e bias simplesmente fazem uma transformação linear. Uma equação linear é simples de resolver, mas é limitada na sua capacidade de resolver problemas complexos. Uma rede neural sem função de ativação é essencialmente apenas um modelo de regressão linear. A função de ativação faz a transformação não-linear nos dados de entrada, tornando-o capaz de aprender e executar tarefas mais complexas. Queremos que nossas redes neurais funcionem em tarefas complicadas, como traduções de idiomas (Processamento de Linguagem Natural) e classificações de imagens (Visão Computacional). As transformações lineares nunca seriam capazes de executar tais tarefas.

As funções de ativação tornam possível a propagação posterior desde que os gradientes sejam fornecidos juntamente com o erro para atualizar os pesos e bias. Sem a função não linear diferenciável, isso não seria possível. Caso o termo gradiente não seja familiar, aguarde os próximos capítulos, quando vamos explicar este conceito em detalhes, visto que ele é a essência do processo de aprendizagem em redes neurais artificiais.

Mas não existe apenas um tipo de função de ativação. Na verdade existem vários, cada qual a ser usado em diferentes situações. Vamos a uma breve descrição dos tipos mais populares.

### Tipos Populares de Funções de Ativação

A função de ativação é um componente matemático incluído na estrutura de redes neurais artificiais a fim de permitir a solução de problemas complexos. Existem diversos tipos de funções de ativação e esta é uma área de pesquisa ativa, à medida que a Inteligência Artificial evolui (não é maravilhoso estar participando desta evolução, que vai transformar completamente o mundo?). Vejamos quais são os tipos mais populares.

#### Função de Etapa Binária (Binary Step Function)

A primeira coisa que vem à nossa mente quando temos uma função de ativação seria um classificador baseado em limiar (threshold), ou seja, se o neurônio deve ou não ser ativado. Se o valor Y estiver acima de um valor de limite determinado, ative o neurônio senão deixa desativado. Simples! Essa seria a regra:

**f(x) = 1, x>=0**

**f(x) = 0, x<0**

A função de etapa binária é isso mesmo, extremamente simples. Ela pode ser usada ao criar um classificador binário. Quando simplesmente precisamos dizer sim ou não para uma única classe, a função de etapa seria a melhor escolha, pois ativaria o neurônio ou deixaria zero.

A função é mais teórica do que prática, pois, na maioria dos casos, classificamos os dados em várias classes do que apenas uma única classe. A função de etapa não seria capaz de fazer isso.

Além disso, o gradiente da função de etapa é zero. Isso faz com que a função de etapa não seja tão útil durante o backpropagation quando os gradientes das funções de ativação são enviados para cálculos de erro para melhorar e otimizar os resultados. O gradiente da função de etapa reduz tudo para zero e a melhoria dos modelos realmente não acontece. Lembrando, mais uma vez, que veremos em detalhes os conceitos de gradiente e backpropagation mais adiante, nos próximos capítulos!

#### Função Linear

Nós vimos o problema com a função step, o gradiente sendo zero, é impossível atualizar o gradiente durante a backpropagation. Em vez de uma função de passo simples, podemos tentar usar uma função linear. Podemos definir a função como:

**f(x) = ax**

A derivada de uma função linear é constante, isto é, não depende do valor de entrada x. Isso significa que toda vez que fazemos backpropagation, o gradiente seria o mesmo. E este é um grande problema, não estamos realmente melhorando o erro, já que o gradiente é praticamente o mesmo. E não apenas suponha que estamos tentando realizar uma tarefa complicada para a qual precisamos de múltiplas camadas em nossa rede. Agora, se cada camada tiver uma transformação linear, não importa quantas camadas nós tenhamos, a saída final não é senão uma transformação linear da entrada. Portanto, a função linear pode ser ideal para tarefas simples, onde a interpretabilidade é altamente desejada.

#### Sigmóide

Sigmóide é uma função de ativação amplamente utilizada. É da forma:

**f (x) = 1 / (1 + e ^ -x)**

Esta é uma função suave e é continuamente diferenciável. A maior vantagem sobre a função de etapa e a função linear é que não é linear. Esta é uma característica incrivelmente interessante da função sigmóide. Isto significa essencialmente que quando eu tenho vários neurônios com função sigmóide como função de ativação – a saída também não é linear. A função varia de 0 a 1 tendo um formato S.

A função essencialmente tenta empurrar os valores de Y para os extremos. Esta é uma qualidade muito desejável quando tentamos classificar os valores para uma classe específica.

A função sigmóide ainda é amplamente utilizada até hoje, mas ainda temos problemas que precisamos abordar. Com a sigmóide temos problemas quando os gradientes se tornam muito pequenos. Isso significa que o gradiente está se aproximando de zero e a rede não está realmente aprendendo.

Outro problema que a função sigmóide sofre é que os valores variam apenas de 0 a 1. Esta medida que a função sigmóide não é simétrica em torno da origem e os valores recebidos são todos positivos. Nem sempre desejamos que os valores enviados ao próximo neurônio sejam todos do mesmo sinal. Isso pode ser abordado pela ampliação da função sigmóide. Isso é exatamente o que acontece na função tanh.

#### Tanh

A função tanh é muito semelhante à função sigmóide. Na verdade, é apenas uma versão escalonada da função sigmóide.

**Tanh (x) = 2sigmoides (2x) -1**

Pode ser escrito diretamente como:

**tanh (x) = 2 / (1 + e ^ (- 2x)) -1**

Tanh funciona de forma semelhante à função sigmóide, mas sim simétrico em relação à origem. varia de -1 a 1.

Basicamente, soluciona o nosso problema dos valores, sendo todos do mesmo sinal. Todas as outras propriedades são as mesmas da função sigmoide. É contínuo e diferenciável em todos os pontos. A função não é linear, então podemos fazer o backpropagation facilmente nos erros.

#### ReLU

A função ReLU é a unidade linear rectificada. É definida como:

**f(x) = max (0, x)**

ReLU é a função de ativação mais amplamente utilizada ao projetar redes neurais atualmente. Primeiramente, a função ReLU é não linear, o que significa que podemos facilmente copiar os erros para trás e ter várias camadas de neurônios ativados pela função ReLU.

A principal vantagem de usar a função ReLU sobre outras funções de ativação é que ela não ativa todos os neurônios ao mesmo tempo. O que isto significa ? Se você olhar para a função ReLU e a entrada for negativa, ela será convertida em zero e o neurônio não será ativado. Isso significa que, ao mesmo tempo, apenas alguns neurônios são ativados, tornando a rede esparsa e eficiente e fácil para a computação.

Mas ReLU também pode ter problemas com os gradientes que se deslocam em direção a zero. Mas quando temos um problema, sempre podemos pensar em uma solução. Aliás, isso é o que as empresas mais procuram nos dias de hoje: “resolvedores de problemas”. Seja um e sua empregabilidade estará garantida!

#### Leaky ReLU

A função Leaky ReLU não passa de uma versão melhorada da função ReLU. Na função ReLU, o gradiente é 0 para x < 0, o que fez os neurônios morrerem por ativações nessa região. Leaky ReLU ajuda a resolver este problema. Em vez de definir a função Relu como 0 para x inferior a 0, definimos como um pequeno componente linear de x. Pode ser definido como:

**f(x) = ax, x < 0**  
**f(x) = x, x > = 0**

O que fizemos aqui é que simplesmente substituímos a linha horizontal por uma linha não-zero, não horizontal. Aqui um é um valor pequeno como 0,01 ou algo parecido. A principal vantagem de substituir a linha horizontal é remover o gradiente zero.

#### Softmax

A função softmax também é um tipo de função sigmóide, mas é útil quando tentamos lidar com problemas de classificação. A função sigmóide como vimos anteriormente é capaz de lidar com apenas duas classes. O que devemos fazer quando estamos tentando lidar com várias classes? Apenas classificar sim ou não para uma única classe não ajudaria. A função softmax transforma as saídas para cada classe para valores entre 0 e 1 e também divide pela soma das saídas. Isso essencialmente dá a probabilidade de a entrada estar em uma determinada classe. Pode ser definido como:

Softmax

Digamos, por exemplo, que temos as saídas como [1.2, 0.9, 0.75], quando aplicamos a função softmax, obteríamos [0.42, 0.31, 0.27]. Então, agora podemos usá-los como probabilidades de que o valor seja de cada classe.

A função softmax é idealmente usada na camada de saída do classificador, onde realmente estamos tentando gerar as probabilidades para definir a classe de cada entrada.

### Escolhendo a Função de Ativação Correta

Ufa! Muita coisa, não? E ainda não vimos as questões matemáticas envolvidas nessas funções. Mas não tenhamos pressa, não existe atalho para o aprendizado e estudaremos tudo passo a passo, item a item, no padrão dos cursos na [Data Science Academy](https://www.datascienceacademy.com.br/).

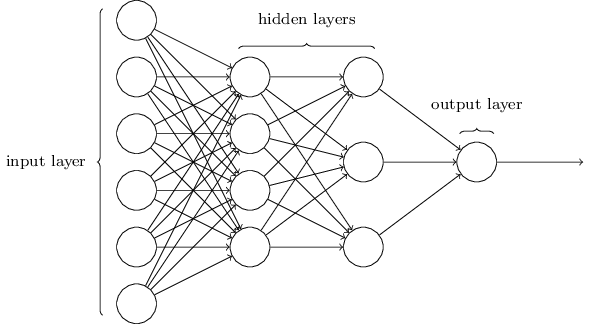
Agora que já vimos tantas funções de ativação, precisamos de alguma lógica/heurística para saber qual função de ativação deve ser usada em qual situação. Não há uma regra de ouro e a escolha depende do problema no qual você estiver trabalhando.

No entanto, dependendo das propriedades do problema, poderemos fazer uma melhor escolha para uma convergência fácil e rápida da rede neural.

* Funções Sigmóide e suas combinações geralmente funcionam melhor no caso de classificadores.
* Funções Sigmóide e Tanh às vezes são evitadas devido ao problema de Vanishing Gradient (que estudaremos no capítulo sobre redes neurais recorrentes).
* A função ReLU é uma função de ativação geral e é usada na maioria dos casos atualmente.
* Se encontrarmos um caso de neurônios deficientes em nossas redes, a função Leaky ReLU é a melhor escolha.
* Tenha sempre em mente que a função ReLU deve ser usada apenas nas camadas ocultas.
* Como regra geral, você pode começar usando a função ReLU e depois passar para outras funções de ativação no caso da ReLU não fornecer resultados ótimos.

Está começando a sentir a vibração em trabalhar com Inteligência Artificial? Então continue acompanhando, pois estamos apenas no começo! Até o próximo capítulo!

# Capítulo 9 – A Arquitetura das Redes Neurais



Tais redes de camadas múltiplas são chamados de Perceptrons Multicamadas ou MLPs (Multilayer Perceptrons), ou seja, uma rede neural formada por Perceptrons (embora na verdade seja uma rede de neurônios sigmóides, como veremos mais adiante).

O design das camadas de entrada e saída em uma rede geralmente é direto. Por exemplo, suponha que estamos tentando determinar se uma imagem manuscrita representa um “9” ou não. Uma maneira natural de projetar a rede é codificar as intensidades dos pixels da imagem nos neurônios de entrada. Se a imagem for uma imagem em escala de cinza 64 x 64, teríamos 64 × 64 = 4.096  neurônios de entrada, com as intensidades dimensionadas adequadamente entre 0 e 1. A camada de saída conterá apenas um único neurônio com valores inferiores a 0,5 indicando que “a imagem de entrada não é um 9” e valores maiores que 0,5 indicando que “a imagem de entrada é um 9”.

Embora o design das camadas de entrada e saída de uma rede neural seja frequentemente direto, pode haver bastante variação para o design das camadas ocultas. Em particular, não é possível resumir o processo de design das camadas ocultas com poucas regras simples. Em vez disso, pesquisadores de redes neurais desenvolveram muitas heurísticas de design para as camadas ocultas, que ajudam as pessoas a obter o comportamento que querem de suas redes. Conheceremos várias heurísticas de design desse tipo mais adiante ao longo dos próximos capítulos. O design das camadas ocultas é um dos pontos cruciais em modelos de Deep Learning.

Até agora, estamos discutindo redes neurais onde a saída de uma camada é usada como entrada para a próxima camada. Essas redes são chamadas de redes neurais feedforward. Isso significa que não há loops na rede – as informações sempre são alimentadas para a frente, nunca são enviadas de volta. Se tivéssemos loops, acabaríamos com situações em que a entrada para a função σ dependeria da saída. Isso seria difícil de entender e, portanto, não permitimos tais loops.

No entanto, existem outros modelos de redes neurais artificiais em que os circuitos de feedback são possíveis. Esses modelos são chamados de redes neurais recorrentes. A ideia nestes modelos é ter neurônios que disparem por algum período de tempo limitado. Disparar pode estimular outros neurônios, que podem disparar um pouco mais tarde, também por uma duração limitada. Isso faz com que ainda mais neurônios disparem e, ao longo do tempo, conseguimos uma cascata de disparos de neurônios. Loops não causam problemas em tal modelo, uma vez que a saída de um neurônio afeta apenas sua entrada em algum momento posterior, não instantaneamente.

Geralmente, as arquiteturas de redes neurais podem ser colocadas em 3 categorias específicas:

### 1 – Redes Neurais Feed-Forward

Estes são o tipo mais comum de rede neural em aplicações práticas. A primeira camada é a entrada e a última camada é a saída. Se houver mais de uma camada oculta, nós as chamamos de redes neurais “profundas” (ou Deep Learning). Esses tipos de redes neurais calculam uma série de transformações que alteram as semelhanças entre os casos. As atividades dos neurônios em cada camada são uma função não-linear das atividades na camada anterior.

### 2 – Redes Recorrentes

Estes tipos de redes neurais têm ciclos direcionados em seu grafo de conexão. Isso significa que às vezes você pode voltar para onde você começou seguindo as setas. Eles podem ter uma dinâmica complicada e isso pode torná-los muito difíceis de treinar. Entretanto, estes tipos são mais biologicamente realistas.

Atualmente, há muito interesse em encontrar formas eficientes de treinamento de redes recorrentes. As redes neurais recorrentes são uma maneira muito natural de modelar dados sequenciais. Eles são equivalentes a redes muito profundas com uma camada oculta por fatia de tempo; exceto que eles usam os mesmos pesos em cada fatia de tempo e recebem entrada em cada fatia. Eles têm a capacidade de lembrar informações em seu estado oculto por um longo período de tempo, mas é muito difícil treiná-las para usar esse potencial.

### 3 – Redes Conectadas Simetricamente

Estas são como redes recorrentes, mas as conexões entre as unidades são simétricas (elas têm o mesmo peso em ambas as direções). As redes simétricas são muito mais fáceis de analisar do que as redes recorrentes. Elas também são mais restritas no que podem fazer porque obedecem a uma função de energia. As redes conectadas simetricamente sem unidades ocultas são chamadas de “Redes Hopfield”. As redes conectadas simetricamente com unidades ocultas são chamadas de “Máquinas de Boltzmann”.

Dentre estas 3 categorias, podemos listar 10 arquiteturas principais de redes neurais:

* Redes Multilayer Perceptron
* Redes Neurais Convolucionais
* Redes Neurais Recorrentes
* Long Short-Term Memory (LSTM)
* Redes de Hopfield
* Máquinas de Boltzmann
* Deep Belief Network
* Deep Auto-Encoders
* Generative Adversarial Network
* Deep Neural Network Capsules (este é um tipo completamente novo de rede neural, lançado no final de 2017)

Quer aprender a construir essas arquiteturas de redes neurais de forma eficiente, profissional e totalmente prática, com mini-projetos para solução de problemas do mundo real, em visão computacional, processamento de linguagem natural, detecção de fraudes, previsão de séries temporais e muito mais? Então confira os únicos cursos online do Brasil, 100% em português, onde você aprende tudo sobre essas arquiteturas. Clique nos links abaixo para acessar os programas completos:

No próximo capítulo, daremos a você uma visão geral sobre cada uma dessas 10 arquiteturas e ao longo dos capítulos seguintes, estudaremos todas elas. Cada umas dessas arquiteturas tem sido usada para resolver diferentes problemas e criar sistemas de Inteligência Artificial. Saber trabalhar com IA de forma eficiente, será determinante para seu futuro profissional.

# Capítulo 10 – As 10 Principais Arquiteturas de Redes Neurais

Learning. O maior problema, por incrível que pareça, será a falta de profissionais qualificados em número suficiente para atender as demandas do mercado.

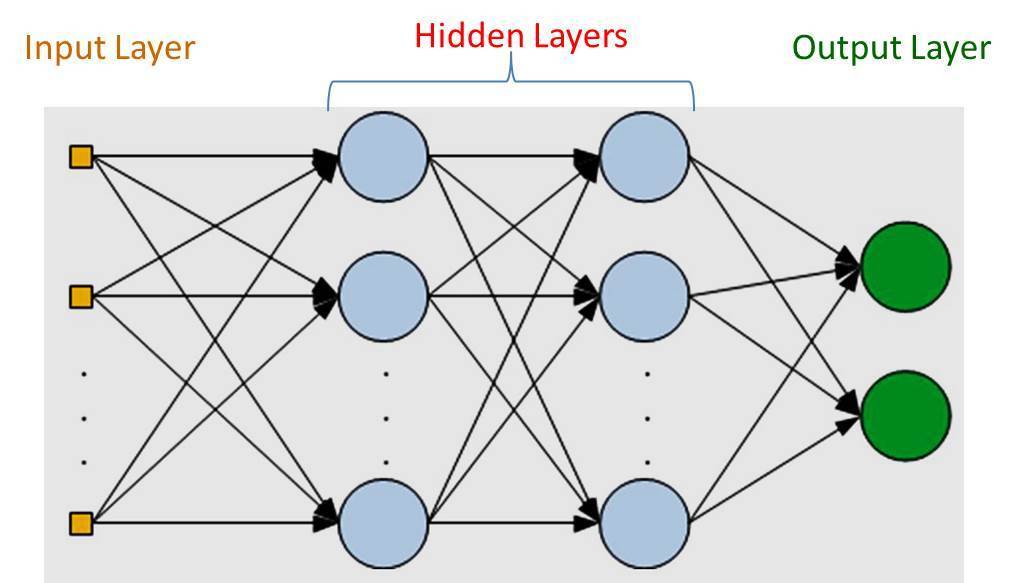
Alguns exemplos de tarefas melhor resolvidas pela aprendizagem de máquina incluem:

* Reconhecimento de padrões: objetos em cenas reais, identidades faciais ou expressões faciais, palavras escritas ou faladas.
* Detecção de anomalias: sequências incomuns de transações de cartão de crédito, padrões incomuns de leituras de sensores em máquinas de uma indústria têxtil.
* Previsão: preços de ações futuros ou taxas de câmbio, quais filmes uma pessoa gostaria de assistir, previsão de vendas.

Machine Learning é um campo abrangente dentro da Inteligência Artificial. Mas uma sub-área de Machine Learning, o [Deep Learning](http://deeplearningbook.com.br/capitulo-3-o-que-sao-redes-neurais-artificiais-profundas/" \t "_blank) (ou Redes Neurais Profundas), vem conseguindo resultados no estado da arte para as tarefas acima mencionadas. Neste capítulo você encontra As 10 Principais Arquiteturas de Redes Neurais, dentre elas as principais arquiteturas de Deep Learning.

### 1- Redes Multilayer Perceptrons

O Perceptron, conforme estudamos nos capítulos anteriores, é um algoritmo simples destinado a realizar a classificação binária; isto é, prevê se a entrada pertence a uma determinada categoria de interesse ou não: fraude ou não\_fraude, gato ou não\_gato.



Um Perceptron é um classificador linear; ou seja, é um algoritmo que classifica a entrada separando duas categorias com uma linha reta. A entrada geralmente é um vetor de recursos **x** multiplicado por pesos **w** e adicionado a um viés (ou bias) **b**. Aqui um exemplo do Perceptron: y = w \* x + b. Um Perceptron produz uma única saída com base em várias entradas de valor real, formando uma combinação linear usando os pesos (e às vezes passando a saída através de uma função de ativação não linear).

[Rosenblatt](https://en.wikipedia.org/wiki/Frank_Rosenblatt) construiu um Perceptron de uma camada. Ou seja, seu algoritmo não inclui múltiplas camadas, o que permite que as redes neurais modelem uma hierarquia de recursos. Isso impede que o Perceptron consiga realizar classificação não linear, como a função XOR (um disparador do operador XOR quando a entrada exibe uma característica ou outra, mas não ambas, significa “OR exclusivo” “), como [Minsky e Papert](https://mitpress.mit.edu/books/perceptrons" \t "_blank) mostraram em seu livro.

Um Multilayer Perceptron (MLP) é uma rede neural artificial composta por mais de um Perceptron. Eles são compostos por uma camada de entrada para receber o sinal, uma camada de saída que toma uma decisão ou previsão sobre a entrada, e entre esses dois, um número arbitrário de camadas ocultas que são o verdadeiro mecanismo computacional do MLP. MLPs com uma camada oculta são capazes de aproximar qualquer função contínua.

O Multilayer Perceptron é uma espécie de “Hello World” da aprendizagem profunda: uma boa forma de começar quando você está aprendendo sobre Deep Learning.

Os MLPs são frequentemente aplicados a problemas de aprendizagem supervisionados: treinam em um conjunto de pares entrada-saída e aprendem a modelar a correlação (ou dependências) entre essas entradas e saídas. O treinamento envolve o ajuste dos parâmetros, ou os pesos e bias, do modelo para minimizar o erro. O backpropagation é usado para fazer os ajustes dos pesos e de bias em relação ao erro, e o próprio erro pode ser medido de várias maneiras, inclusive pelo erro quadrático médio (MSE – Mean Squared Error).

As redes feed forward, como MLPs, são como ping-pong. Elas são principalmente envolvidas em dois movimentos, uma constante de ida e volta.  Na passagem para a frente, o fluxo de sinal se move da camada de entrada através das camadas ocultas para a camada de saída e a decisão da camada de saída é medida em relação às saídas esperadas.

Na passagem para trás, usando o backpropagation e a regra da cadeia (Chain Rule), derivadas parciais da função de erro dos vários pesos e bias são reproduzidos através do MLP. Esse ato de diferenciação nos dá um gradiente, ao longo do qual os parâmetros podem ser ajustados à medida que movem o MLP um passo mais perto do erro mínimo. Isso pode ser feito com qualquer algoritmo de otimização baseado em gradiente, como descida estocástica do gradiente. A rede continua jogando aquele jogo de ping-pong até que o erro não possa mais ser reduzido (chegou ao mínimo possível). Este estado é conhecido como convergência.

Parece muita coisa? Sim, é. Veremos esse processo em mais detalhes aqui mesmo neste livro e caso queira aprender a construir modelos MLP para aplicações práticas, através de vídeos em português, clique [aqui](https://www.datascienceacademy.com.br/pages/curso-deep-learning-i).

### 2- Redes Neurais Convolucionais

Em 1998, [Yann LeCun](http://yann.lecun.com/" \t "_blank) e seus colaboradores desenvolveram um reconhecedor, realmente bom, para dígitos manuscritos chamado [LeNet](http://yann.lecun.com/exdb/lenet/" \t "_blank). Ele usou o backpropagation em uma rede feed forward com muitas camadas ocultas, muitos mapas de unidades replicadas em cada camada, agrupando as saídas de unidades próximas, formando uma rede ampla que pode lidar com vários caracteres ao mesmo tempo, mesmo se eles se sobrepõem e uma inteligente maneira de treinar um sistema completo, não apenas um reconhecedor. Mais tarde, esta arquitetura foi formalizada sob o nome de redes neurais convolucionais.

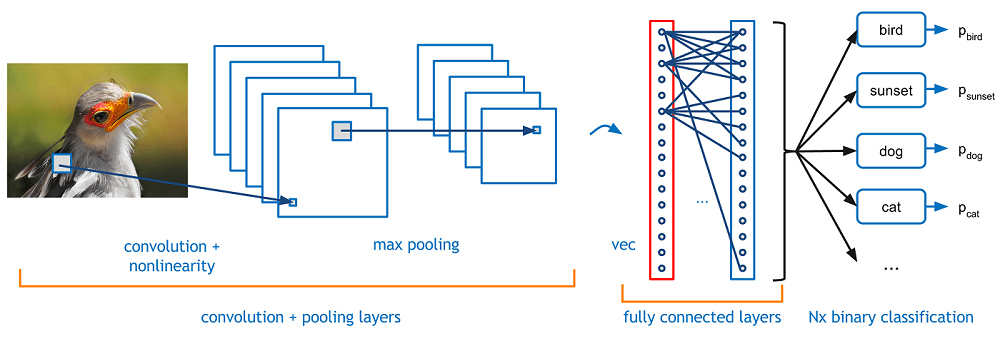
As Redes Neurais Convolucionais (ConvNets ou CNNs) são redes neurais artificiais profundas que podem ser usadas para classificar imagens, agrupá-las por similaridade (busca de fotos) e realizar reconhecimento de objetos dentro de cenas. São algoritmos que podem identificar rostos, indivíduos, sinais de rua, cenouras, ornitorrincos e muitos outros aspectos dos dados visuais.

As redes convolucionais realizam o reconhecimento óptico de caracteres (OCR) para digitalizar texto e tornar possível o processamento de linguagem natural em documentos analógicos e manuscritos, onde as imagens são símbolos a serem transcritos. CNNs também podem ser aplicadas a arquivos de áudio quando estes são representados visualmente como um espectrograma. Mais recentemente, as redes convolucionais foram aplicadas diretamente à análise de texto, bem como dados gráficos.

A eficácia das redes convolucionais no reconhecimento de imagem é uma das principais razões pelas quais o mundo testemunhou a eficácia do aprendizado profundo. Este tipo de rede está impulsionando grandes avanços em [Visão Computacional](https://www.datascienceacademy.com.br/pages/curso-visao-computacional-e-reconhecimento-de-imagens), que tem aplicações óbvias em carros autônomos, robótica, drones, segurança, diagnósticos médicos e tratamentos para deficientes visuais.

As redes convolucionais ingerem e processam imagens como tensores e tensores são matrizes de números com várias dimensões. Eles podem ser difíceis de visualizar, então vamos abordá-los por analogia. Um escalar é apenas um número, como 7; um vetor é uma lista de números (por exemplo, [7,8,9]); e uma matriz é uma grade retangular de números que ocupam várias linhas e colunas como uma planilha. Geometricamente, se um escalar é um ponto de dimensão zero, então um vetor é uma linha unidimensional, uma matriz é um plano bidimensional, uma pilha de matrizes é um cubo tridimensional e quando cada elemento dessas matrizes tem uma pilha de mapas de recursos ligados a ele, você entra na quarta dimensão. Calma, não se desespere (ainda). Veremos isso mais a frente com calma, quando estudarmos exclusivamente esta arquitetura. Em nossos cursos na Data Science Academy incluímos aulas completas sobre Álgebra Linear, onde escalares, vetores, matrizes e tensores são estudados na teoria e prática, pois este conhecimento é fundamental na construção de redes neurais profundas.

A primeira coisa a saber sobre redes convolucionais é que elas não percebem imagens como os humanos. Portanto, você terá que pensar de uma maneira diferente sobre o que uma imagem significa quando é alimentada e processada por uma rede convolucional.



As redes convolucionais percebem imagens como volumes; isto é, objetos tridimensionais, em vez de estruturas planas a serem medidas apenas por largura e altura. Isso porque as imagens de cores digitais têm uma codificação vermelho-verde-azul (RGB – Red-Green-Blue), misturando essas três cores para produzir o espectro de cores que os seres humanos percebem. Uma rede convolucional recebe imagens como três estratos separados de cores empilhados um em cima do outro.

Assim, uma rede convolucional recebe uma imagem como uma caixa retangular cuja largura e altura são medidas pelo número de pixels ao longo dessas dimensões e cuja profundidade é de três camadas profundas, uma para cada letra em RGB. Essas camadas de profundidade são referidas como canais.

À medida que as imagens se movem através de uma rede convolucional, descrevemos em termos de volumes de entrada e saída, expressando-as matematicamente como matrizes de múltiplas dimensões dessa forma: 30x30x3. De camada em camada, suas dimensões mudam à medida que atravessam a rede neural convolucional até gerar uma série de probabilidades na camada de saída, sendo uma probabilidade para cada possível classe de saída. Aquela com maior probabilidade, será a classe definida para a imagem de entrada, um pássaro por exemplo.

Você precisará prestar muita atenção às medidas de cada dimensão do volume da imagem, porque elas são a base das operações de álgebra linear usadas para processar imagens. Poderíamos dedicar dois capítulos inteiros somente a esta arquitetura. Aliás, é o que faremos mais à frente aqui no livro e o que já fazemos na prática [aqui](https://www.datascienceacademy.com.br/pages/curso-deep-learning-i).

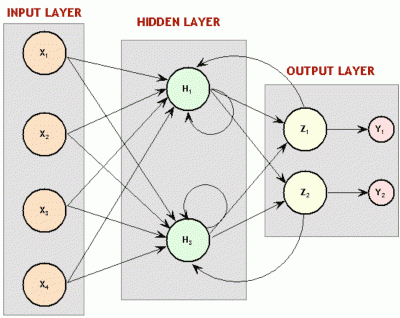
### 3- Redes Neurais Recorrentes

As redes recorrentes são um poderoso conjunto de algoritmos de redes neurais artificiais especialmente úteis para o processamento de dados sequenciais, como som, dados de séries temporais ou linguagem natural. Uma versão de redes recorrentes foi usada pelo [DeepMind](https://deepmind.com/" \t "_blank) no projeto de videogames com agentes autônomos.

As redes recorrentes diferem das redes feed forward porque incluem um loop de feedback, pelo qual a saída do passo n-1 é alimentada de volta à rede para afetar o resultado do passo n, e assim por diante para cada etapa subsequente. Por exemplo, se uma rede é exposta a uma palavra letra por letra, e é solicitado a adivinhar cada letra a seguir, a primeira letra de uma palavra ajudará a determinar o que uma rede recorrente pensa que a segunda letra pode ser.

Isso difere de uma rede feed forward, que aprende a classificar cada número manuscrito por exemplo, independentemente, e de acordo com os pixels de que é exposto a partir de um único exemplo, sem se referir ao exemplo anterior para ajustar suas previsões. As redes feed forward aceitam uma entrada por vez e produzem uma saída. As redes recorrentes não enfrentam a mesma restrição um-para-um.

Embora algumas formas de dados, como imagens, não pareçam ser sequenciais, elas podem ser entendidas como sequências quando alimentadas em uma rede recorrente. Considere uma imagem de uma palavra manuscrita. Assim como as redes recorrentes processam a escrita manual, convertendo cada imagem em uma letra e usando o início de uma palavra para adivinhar como essa palavra terminará, então as redes podem tratar parte de qualquer imagem como letras em uma sequência. Uma rede neural que percorre uma imagem grande pode aprender a partir de cada região, o que as regiões vizinhas, são mais prováveis ​​de ser.



As redes recorrentes e as redes feed forward “lembram” algo sobre o mundo, modelando os dados que estão expostos. Mas elas se lembram de maneiras muito diferentes. Após o treinamento, a rede feed forward produz um modelo estático dos dados e esse modelo pode então aceitar novos exemplos e classificá-los ou agrupá-los com precisão.

Em contraste, as redes recorrentes produzem modelos dinâmicos – ou seja, modelos que mudam ao longo do tempo – de formas que produzem classificações precisas dependentes do contexto dos exemplos que estão expostos.

Para ser preciso, um modelo recorrente inclui o estado oculto que determinou a classificação anterior em uma série. Em cada etapa subsequente, esse estado oculto é combinado com os dados de entrada do novo passo para produzir a) um novo estado oculto e, em seguida, b) uma nova classificação. Cada estado oculto é reciclado para produzir seu sucessor modificado.

As memórias humanas também são conscientes do contexto, reciclando a consciência de estados anteriores para interpretar corretamente novos dados. Por exemplo, vamos considerar dois indivíduos. Um está ciente de que ele está perto da casa de Bob. O outro está ciente de que entrou em um avião. Eles interpretarão os sons “Oi Bob!” de duas formas muito diferentes, precisamente porque retém um estado oculto afetado por suas memórias de curto prazo e sensações precedentes.

Diferentes lembranças de curto prazo devem ser recontadas em momentos diferentes, a fim de atribuir o significado certo à entrada atual. Algumas dessas memórias terão sido forjadas recentemente e outras memórias terão forjado muitos passos antes de serem necessários. A rede recorrente que efetivamente associa memórias e entrada remota no tempo é chamada de Memória de Longo Prazo (LSTM), a qual veremos em seguida.

### 4- Long Short-Term Memory (LSTM)

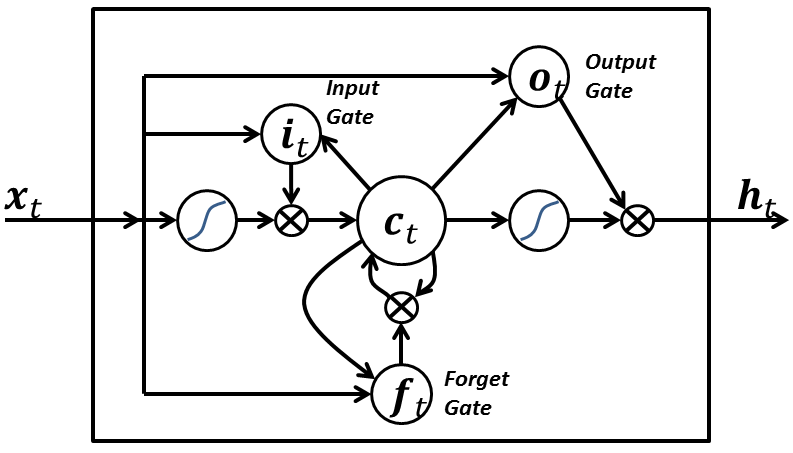
Em meados dos anos 90, a proposta dos pesquisadores alemães [Sepp Hochreiter e Juergen Schmidhuber](http://www.bioinf.jku.at/publications/older/2604.pdf) apresentou uma variação da rede recorrente com as chamadas unidades de Long Short-Term Memory, como uma solução para o problema do vanishing gradient, problema comum em redes neurais recorrentes.

Os LSTMs ajudam a preservar o erro que pode ser copiado por tempo e camadas. Ao manter um erro mais constante, eles permitem que as redes recorrentes continuem aprendendo durante vários passos de tempo (mais de 1000), abrindo assim um canal para vincular causas e efeitos remotamente. Este é um dos desafios centrais para a aprendizagem de máquina e a IA, uma vez que os algoritmos são frequentemente confrontados por ambientes onde os sinais de recompensa são escassos e atrasados, como a própria vida. (Os pensadores religiosos abordaram este mesmo problema com ideias de karma ou recompensas divinas, teorizando consequências invisíveis e distantes para nossas ações).

Os LSTMs contêm informações fora do fluxo normal da rede recorrente em uma célula fechada. As informações podem ser armazenadas, escritas ou lidas a partir de uma célula, como dados na memória de um computador. A célula toma decisões sobre o que armazenar, e quando permitir leituras, gravações e exclusões, através de portões abertos e fechados. Ao contrário do armazenamento digital em computadores, no entanto, esses portões são analógicos, implementados com a multiplicação de elementos por sigmóides, que estão todos na faixa de 0-1. Analógico tem a vantagem sobre o digital de ser diferenciável e, portanto, adequado para backpropagation.

Esses portões atuam sobre os sinais que recebem e, de forma semelhante aos nós da rede neural, eles bloqueiam ou transmitem informações com base em sua força e importação, que eles filtram com seus próprios conjuntos de pesos. Esses pesos, como os pesos que modulam a entrada e estados ocultos, são ajustados através do processo de aprendizagem das redes recorrentes. Ou seja, as células aprendem quando permitir que os dados entrem, saiam ou sejam excluídos através do processo iterativo de fazer suposições, calculando o erro durante o backpropagation e ajustando pesos através da descida do gradiente.

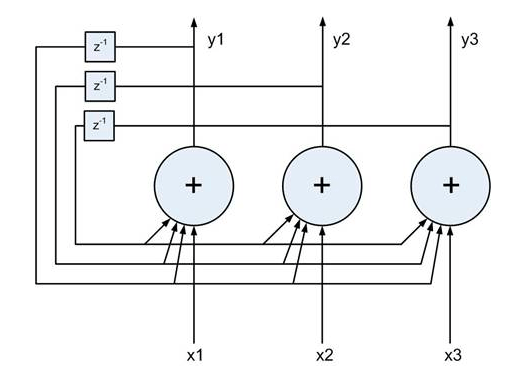
O diagrama abaixo ilustra como os dados fluem através de uma célula de memória e são controlados por seus portões.



Os LSTM’s possuem muitas aplicações práticas, incluindo processamento de linguagem natural, geração automática de texto e análise de séries temporais. Caso queira ver esses exemplos na prática, clique [aqui](https://www.datascienceacademy.com.br/pages/curso-deep-learning-ii). Teremos um capítulo inteiro dedicado aos LSTM’s aqui no livro.

### 5- Redes de Hopfield

Redes recorrentes de unidades não lineares geralmente são muito difíceis de analisar. Elas podem se comportar de muitas maneiras diferentes: se estabelecer em um estado estável, oscilar ou seguir trajetórias caóticas que não podem ser preditas no futuro. Uma Rede Hopfield é composta por unidades de limite binário com conexões recorrentes entre elas. Em 1982, John Hopfield percebeu que, se as conexões são simétricas, existe uma função de energia global. Cada “configuração” binária de toda a rede possui energia, enquanto a regra de decisão do limite binário faz com que a rede se conforme com um mínimo desta função de energia. Uma excelente maneira de usar esse tipo de computação é usar memórias como energia mínima para a rede neural. Usar mínimos de energia para representar memórias resulta em uma memória endereçável ao conteúdo. Um item pode ser acessado por apenas conhecer parte do seu conteúdo. É robusto contra danos no hardware.



Cada vez que memorizamos uma configuração, esperamos criar um novo mínimo de energia. Mas e se dois mínimos próximos estão em um local intermediário? Isso limita a capacidade de uma Rede Hopfield. Então, como aumentamos a capacidade de uma Rede Hopfield? Os físicos adoram a ideia de que a matemática que eles já conhecem pode explicar como o cérebro funciona. Muitos artigos foram publicados em revistas de física sobre Redes Hopfield e sua capacidade de armazenamento. Eventualmente, [Elizabeth Gardner](http://www.baginsky.de/eli/eg_portr.html) descobriu que havia uma regra de armazenamento muito melhor que usa a capacidade total dos pesos. Em vez de tentar armazenar vetores de uma só vez, ela percorreu o conjunto de treinamento muitas vezes e usou o procedimento de convergência Perceptron para treinar cada unidade para ter o estado correto, dado os estados de todas as outras unidades nesse vetor. **Os estatísticos chamam essa técnica de “pseudo-probabilidade”**.

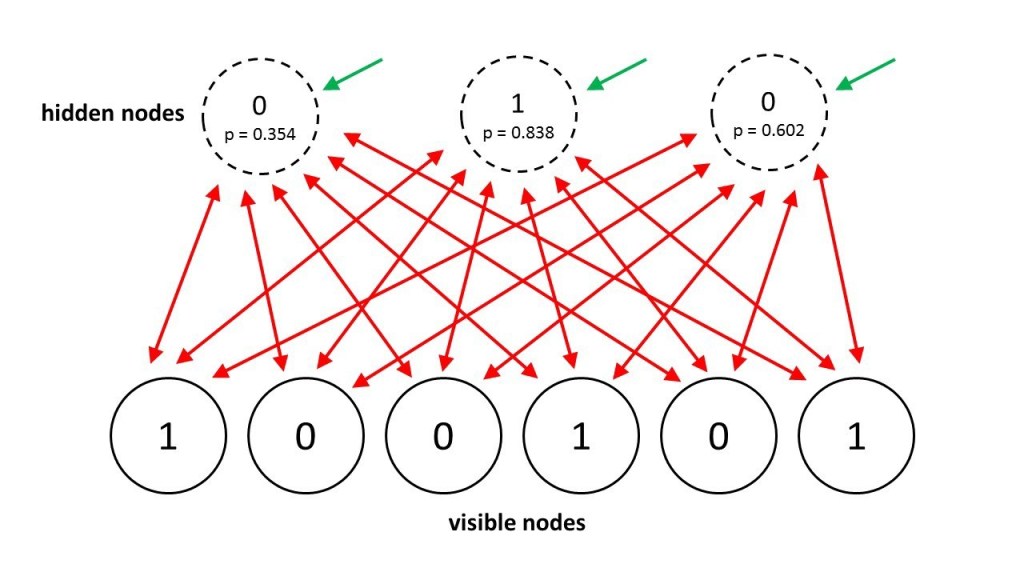
Existe outro papel computacional para as Redes Hopfield. Em vez de usar a rede para armazenar memórias, usamos para construir interpretações de entrada sensorial. A entrada é representada pelas unidades visíveis, a interpretação é representada pelos estados das unidades ocultas e o erro da interpretação é representado pela energia.

### 6- Máquinas de Boltzmann

Uma Máquina de Boltzmann é um tipo de rede neural recorrente estocástica. Pode ser visto como a contrapartida estocástica e generativa das Redes Hopfield. Foi uma das primeiras redes neurais capazes de aprender representações internas e é capaz de representar e resolver problemas combinatórios difíceis.

O objetivo do aprendizado do algoritmo da Máquina de Boltzmann é maximizar o produto das probabilidades que a Máquina de Boltzmann atribui aos vetores binários no conjunto de treinamento. Isso equivale a maximizar a soma das probabilidades de log que a Máquina de Boltzmann atribui aos vetores de treinamento. Também é equivalente a maximizar a probabilidade de obtermos exatamente os N casos de treinamento se fizéssemos o seguinte: 1) Deixar a rede se estabelecer em sua distribuição estacionária no tempo N diferente, sem entrada externa e 2) Mudar o vetor visível uma vez em cada passada.

Um procedimento eficiente de aprendizado de mini-lote foi proposto para as Máquinas de Boltzmann por [Salakhutdinov e Hinton em 2012](http://proceedings.mlr.press/v5/salakhutdinov09a/salakhutdinov09a.pdf" \t "_blank).



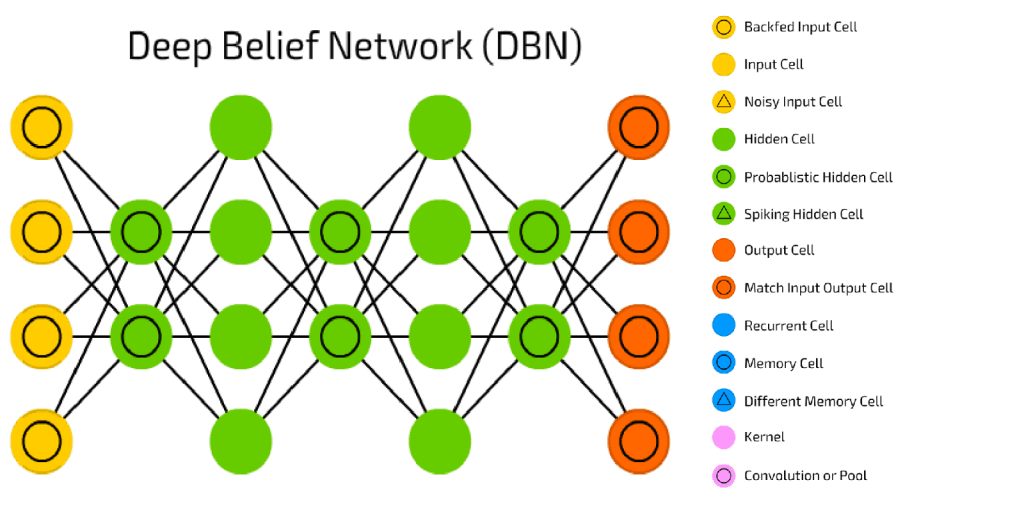
Em uma Máquina de Boltzmann geral, as atualizações estocásticas de unidades precisam ser sequenciais. Existe uma arquitetura especial que permite alternar atualizações paralelas que são muito mais eficientes (sem conexões dentro de uma camada, sem conexões de camada ignorada). Este procedimento de mini-lote torna as atualizações da Máquina de Boltzmann mais paralelas. Isso é chamado de Deep Boltzmann Machines (DBM), uma Máquina de Boltzmann geral, mas com muitas conexões ausentes.

Em 2014, Salakhutdinov e Hinton apresentaram outra atualização para seu modelo, chamando-o de Máquinas Boltzmann Restritas. Elas restringem a conectividade para facilitar a inferência e a aprendizagem (apenas uma camada de unidades escondidas e sem conexões entre unidades ocultas). Em um RBM, é preciso apenas um passo para alcançar o equilíbrio.

### 7- Deep Belief Network

O backpropagation é considerado o método padrão em redes neurais artificiais para calcular a contribuição de erro de cada neurônio após processar um lote de dados (teremos um capítulo inteiro sobre isso). No entanto, existem alguns problemas importantes no backpropagation. Em primeiro lugar, requer dados de treinamento rotulados; enquanto quase todos os dados estão sem rótulos. Em segundo lugar, o tempo de aprendizagem não escala bem, o que significa que é muito lento em redes com múltiplas camadas ocultas. Em terceiro lugar, pode ficar preso em um “local optima”. Portanto, para redes profundas, o backpropagation está longe de ser ótimo.

Para superar as limitações do backpropagation, os pesquisadores consideraram o uso de abordagens de aprendizado sem supervisão. Isso ajuda a manter a eficiência e a simplicidade de usar um método de gradiente para ajustar os pesos, mas também usá-lo para modelar a estrutura da entrada sensorial. Em particular, eles ajustam os pesos para maximizar a probabilidade de um modelo gerador ter gerado a entrada sensorial. A questão é que tipo de modelo generativo devemos aprender? Pode ser um modelo baseado em energia como uma Máquina de Boltzmann? Ou um modelo causal feito de neurônios? Ou um híbrido dos dois?



Uma Deep Belief Network pode ser definida como uma pilha de Máquinas de Boltzmann Restritas (RBM – Restricted Boltzmann Machines), em que cada camada RBM se comunica com as camadas anterior e posterior. Os nós de qualquer camada única não se comunicam lateralmente.

Esta pilha de RBMs pode terminar com uma camada Softmax para criar um classificador, ou simplesmente pode ajudar a agrupar dados não gravados em um cenário de aprendizado sem supervisão.

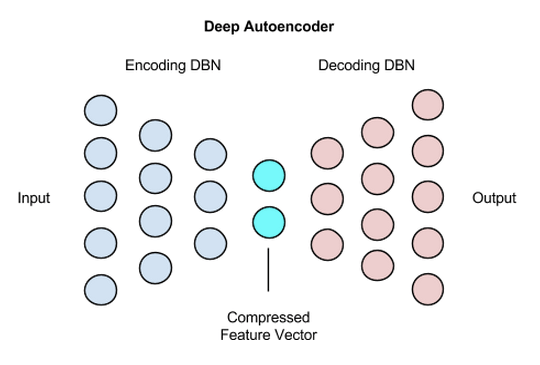
Com a exceção das camadas inicial e final, cada camada em uma Deep Belief Network tem uma função dupla: ela serve como a camada oculta para os nós que vem antes, e como a camada de entrada (ou “visível”) para a nós que vem depois. É uma rede construída de redes de camada única.

As Deep Belief Networks são usadas para reconhecer, agrupar e gerar imagens, sequências de vídeos e dados de captura de movimento. Outra aplicação das Deep Belief Networks é no [Processamento de Linguagem Natural](https://www.datascienceacademy.com.br/pages/curso-processamento-de-linguagem-natural-e-reconhecimento-de-voz). Esse tipo de rede foi apresentado por Geoff Hinton e seus alunos em 2006.

### 8- Deep Auto-Encoders

Um Deep Auto-Encoder é composto por duas redes simétricas Deep Belief que tipicamente têm quatro ou cinco camadas rasas que representam a metade da codificação (encoder) da rede e o segundo conjunto de quatro ou cinco camadas que compõem a metade da decodificação (decoder).

As camadas são Máquinas de Boltzmann Restritas, os blocos de construção das Deep Belief Networks, com várias peculiaridades que discutiremos abaixo. Aqui está um esquema simplificado da estrutura de um Deep Auto-Encoder:



Os Deep Auto-Encoders são uma maneira muito agradável de reduzir a dimensionalidade não linear devido a alguns motivos: eles fornecem mapeamentos flexíveis em ambos os sentidos. O tempo de aprendizagem é linear (ou melhor) no número de casos de treinamento. E o modelo de codificação final é bastante compacto e rápido. No entanto, pode ser muito difícil otimizar Deep Auto-Encoders usando backpropagation. Com pequenos pesos iniciais, o gradiente do backpropagation morre. Mas temos maneiras de otimizá-las, usando o pré-treinamento camada-por-camada sem supervisão ou apenas inicializando os pesos com cuidado.

Os Deep Auto-Encoders são úteis na modelagem de tópicos ou modelagem estatística de tópicos abstratos que são distribuídos em uma coleção de documentos. Isso, por sua vez, é um passo importante em sistemas de perguntas e respostas como o IBM Watson.

Em resumo, cada documento em uma coleção é convertido em um Bag-of-Words (ou seja, um conjunto de contagens de palavras) e essas contagens de palavras são dimensionadas para decimais entre 0 e 1, o que pode ser pensado como a probabilidade de uma palavra ocorrer no documento.

As contagens de palavras em escala são então alimentadas em uma Deep Belief Network, uma pilha de Máquinas de Boltzmann Restritas, que elas mesmas são apenas um subconjunto de Autoencoders. Essas Deep Belief Networks, ou DBNs, comprimem cada documento para um conjunto de 10 números através de uma série de transformações sigmóides que o mapeiam no espaço de recursos.

O conjunto de números de cada documento, ou vetor, é então introduzido no mesmo espaço vetorial, e sua distância de qualquer outro vetor de documento medido. Em termos aproximados, os vetores de documentos próximos se enquadram no mesmo tópico. Por exemplo, um documento poderia ser a “pergunta” e outros poderiam ser as “respostas”, uma combinação que o software faria usando medidas de espaço vetorial.

Em resumo, existem agora muitas maneiras diferentes de fazer pré-treinamento camada-por-camada de recursos. Para conjuntos de dados que não possuem um grande número de casos rotulados, o pré-treinamento ajuda a aprendizagem discriminativa subsequente. Para conjuntos de dados muito grandes e rotulados, não é necessário inicializar os pesos utilizados na aprendizagem supervisionada usando pré-treinamento não supervisionado, mesmo para redes profundas. O pré-treinamento foi o primeiro bom caminho para inicializar os pesos para redes profundas, mas agora existem outras formas. Mas se construímos redes muito maiores, precisaremos de pré-treinamento novamente! Se quiser aprender a construir Deep Auto-Encoders em Python, clique [aqui](https://www.datascienceacademy.com.br/pages/curso-deep-learning-ii).

### 9- Generative Adversarial Network

As Generative Adversarial Networks (GANs) são arquiteturas de redes neurais profundas compostas por duas redes, colocando uma contra a outra (daí o nome, “adversária”).

Os GANs foram introduzidos em um artigo de Ian Goodfellow e outros pesquisadores da Universidade de Montreal no Canadá, incluindo Yoshua Bengio, em 2014. Referindo-se aos GANs, o diretor de pesquisa de IA do Facebook, Yann LeCun, chamou de treinamento adversário “a ideia mais interessante nos últimos 10 anos em Machine Learning”.

O potencial de GANs é enorme, porque eles podem aprender a imitar qualquer distribuição de dados. Ou seja, os GANs podem ser ensinados a criar mundos estranhamente semelhantes aos nossos em qualquer domínio: imagens, música, fala, prosa. Eles são artistas robôs em um sentido, e sua produção é impressionante – até mesmo pungente.

Para entender os GANs, você deve saber como os algoritmos geradores funcionam, e para isso, contrastá-los com algoritmos discriminatórios é útil. Os algoritmos discriminatórios tentam classificar dados de entrada; isto é, dados os recursos de uma instância de dados, eles predizem um rótulo ou categoria a que esses dados pertencem.

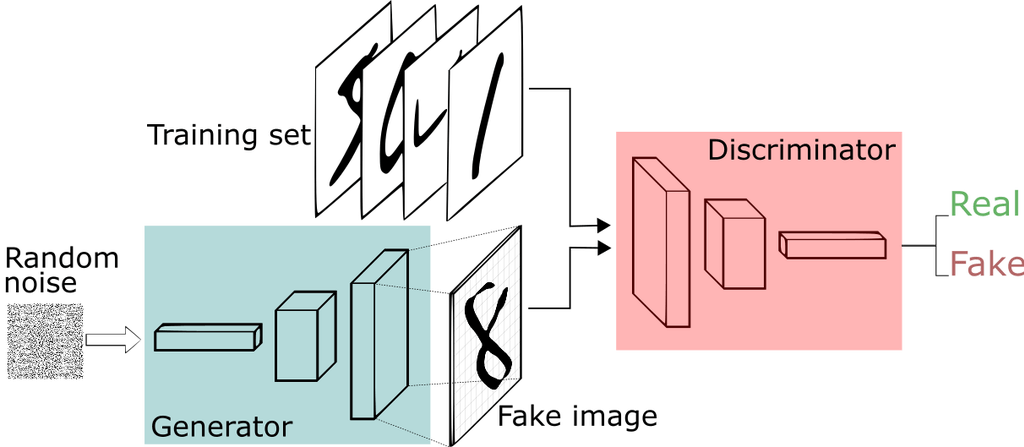
Por exemplo, tendo em conta todas as palavras em um e-mail, um algoritmo discriminatório pode prever se a mensagem é spam ou not\_spam. O spam é um dos rótulos, e o saco de palavras (Bag of Words) coletadas do e-mail são os recursos que constituem os dados de entrada. Quando este problema é expresso matematicamente, o rótulo é chamado y e os recursos são chamados de x. A formulação p (y | x) é usada para significar “a probabilidade de y dado x”, que neste caso seria traduzido para “a probabilidade de um email ser spam com as palavras que contém”.

Portanto, algoritmos discriminatórios mapeiam recursos para rótulos. Eles estão preocupados apenas com essa correlação. Uma maneira de pensar sobre algoritmos generativos é que eles fazem o contrário. Em vez de prever um rótulo com determinados recursos, eles tentam prever os recursos com um determinado rótulo.

A questão que um algoritmo gerador tenta responder é: assumir que este e-mail é spam, qual a probabilidade dos recursos? Enquanto os modelos discriminativos se preocupam com a relação entre y e x, os modelos generativos se preocupam com “como você obtém x”. Eles permitem que você capture p (x | y), a probabilidade de x dado y, ou a probabilidade de características oferecidas em uma classe . (Dito isto, os algoritmos geradores também podem ser usados ​​como classificadores, embora eles podem fazer mais do que categorizar dados de entrada.)

Outra maneira de pensar sobre isso é distinguir discriminativo de gerador assim:

* Modelos discriminativos aprendem o limite entre as classes
* Modelos generativos modelam a distribuição de classes individuais



Uma rede neural, chamada de gerador, gera novas instâncias de dados, enquanto a outra, o discriminador, as avalia por autenticidade; ou seja, o discriminador decide se cada instância de dados que revisa pertence ao conjunto de dados de treinamento real ou não.

Digamos que estamos tentando fazer algo mais banal do que imitar a Mona Lisa. Vamos gerar números escritos à mão como os encontrados no conjunto de dados [MNIST](http://yann.lecun.com/exdb/mnist/), que é retirado do mundo real. O objetivo do discriminador, quando mostrado uma instância do verdadeiro conjunto de dados MNIST, é reconhecê-los como autênticos.

Enquanto isso, o gerador está criando novas imagens que passa para o discriminador. Isso acontece com a esperança de que eles, também, sejam considerados autênticos, embora sejam falsos. O objetivo do gerador é gerar dígitos ​​escritos à mão por si mesmo. O objetivo do discriminador é identificar as imagens provenientes do gerador como falsas.

Aqui estão os passos que um GAN realiza:

* O gerador recebe números aleatórios e retorna uma imagem.
* Essa imagem gerada é alimentada no discriminador ao lado de um fluxo de imagens tiradas do conjunto de dados real.
* O discriminador assume imagens reais e falsas e retorna probabilidades, um número entre 0 e 1, com 1 representando uma previsão de autenticidade e 0 representando falsas.

Então você tem um loop de feedback duplo:

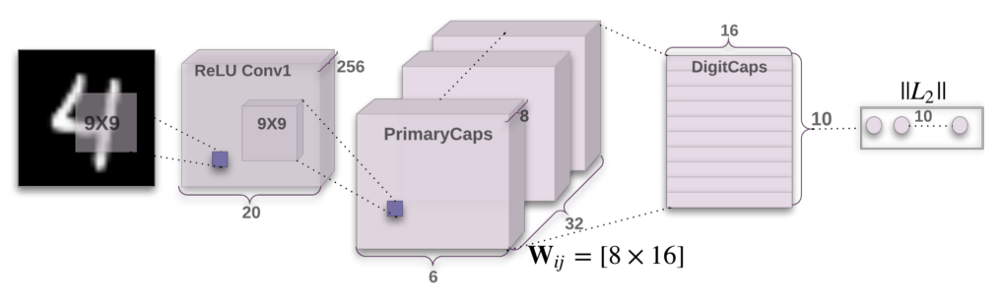
* O discriminador está em um loop de feedback com as imagens verdadeiras, que conhecemos.
* O gerador está em um loop de feedback com o discriminador.

Quer aprender como construir GANs, uma das arquiteturas mais incríveis de Deep Learning, 100% em português e 100% online, para gerar imagens de forma automática? Clique [aqui](https://www.datascienceacademy.com.br/pages/curso-deep-learning-ii).

## **10- Deep Neural Network Capsules**

No final de 2017, Geoffrey Hinton e sua equipe publicaram dois artigos que introduziram um novo tipo de rede neural chamada ***Capsules***. Além disso, a equipe publicou um algoritmo, denominado roteamento dinâmico entre cápsulas, que permite treinar essa rede.

Para todos na comunidade de Deep Learning, esta é uma grande notícia, e por várias razões. Em primeiro lugar, Hinton é um dos fundadores do Deep Learning e um inventor de inúmeros modelos e algoritmos que hoje são amplamente utilizados. Em segundo lugar, esses artigos apresentam algo completamente novo, e isso é muito emocionante porque provavelmente estimulará a onda adicional de pesquisas e aplicativos muito inovadores.



As ***Capsules*** introduzem um novo bloco de construção que pode ser usado na aprendizagem profunda para modelar melhor as relações hierárquicas dentro da representação do conhecimento interno de uma rede neural. A intuição por trás deles é muito simples e elegante.

Hinton e sua equipe propuseram uma maneira de treinar essa rede composta de cápsulas e treinou-a com êxito em um conjunto de dados simples, alcançando desempenho de ponta. Isso é muito encorajador. No entanto, há desafios. As implementações atuais são muito mais lentas do que outros modelos modernos de aprendizado profundo. O tempo mostrará se as redes ***Capsules*** podem ser treinadas de forma rápida e eficiente. Além disso, precisamos ver se elas funcionam bem em conjuntos de dados mais difíceis e em diferentes domínios.

Em qualquer caso, a rede ***Capsule*** é um modelo muito interessante e já funcionando, que definitivamente se desenvolverá ao longo do tempo e contribuirá para uma maior expansão de aplicações de aprendizagem profunda.

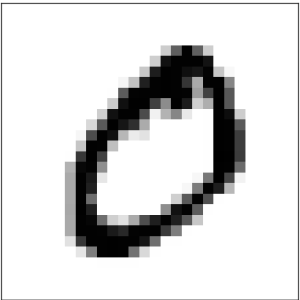
Incluímos as Capsules entre as 10 principais arquiteturas de redes neurais, pois elas representam a inovação e o avanço na incrível e vibrante área de Deep Learning e sistemas de Inteligência Artificial. Profissionais que realmente desejem abraçar IA como carreira, devem estar atentos aos movimentos e inovações na área.

Esta não é uma lista definitiva de arquiteturas e existem outras, tais como Word2Vec, Doc2vec, Neural Embeddings e variações das arquiteturas aqui apresentadas, como Denoising Autoencoders, Variational Autoencoders, além de outras categorias como Deep Reinforcement Learning. Exatamente para auxiliar aqueles que buscam conhecimento de ponta 100% em português e 100% online, que nós criamos a [Formação Inteligência Artificial](https://www.datascienceacademy.com.br/pages/formacao-inteligencia-artificial), o único programa do Brasil completo, com todas as ferramentas que o aluno precisa para aprender a trabalhar com IA de forma eficiente. O aluno aprende programação paralela em GPU, Deep Learning e seus frameworks, estuda as principais arquiteturas com aplicações práticas e desenvolve aplicações de Visão Computacional e Processamento de Linguagem Natural. Clique [aqui](https://www.datascienceacademy.com.br/pages/formacao-inteligencia-artificial) e veja mais detalhes sobre o programa.

Isso conclui a primeira parte deste livro, com uma introdução ao universo do Deep Learning. No próximo capítulo começaremos a ver as redes neurais em ação. Até lá.

# Capítulo 11 – Design De Uma Rede Neural Para Reconhecimento de Dígitos

Como você pode ter adivinhado, essas quatro imagens juntas compõem a imagem 0 que vimos na linha de dígitos mostrada anteriormente:



Então, se todos os quatro neurônios ocultos estão disparando, podemos concluir que o dígito é um 0. Claro, esse não é o único tipo de evidência que podemos usar para concluir que a imagem era um 0 – podemos legitimamente obter um 0 em muitas outras maneiras (por exemplo, através de traduções das imagens acima, ou pequenas distorções). Mas parece seguro dizer que, pelo menos neste caso, concluiríamos que a entrada era um 0.

Supondo que a rede neural funciona assim, podemos dar uma explicação plausível sobre porque é melhor ter 10 saídas da rede, em vez de 4. Se tivéssemos 4 saídas, o primeiro neurônio de saída tentaria decidir o que mais um bit significativo do dígito representa. E não existe uma maneira fácil de relacionar esse bit mais significativo com formas simples, como as mostradas acima. As formas componentes do dígito estarão intimamente relacionadas com (digamos) o bit mais significativo na saída.

Isso tudo é apenas uma heurística. Nada diz que a rede neural de três camadas tem que operar da maneira que descrevemos, com os neurônios ocultos detectando formas de componentes simples. Talvez um algoritmo de aprendizado inteligente encontre alguma atribuição de pesos que nos permita usar apenas 4 neurônios de saída. Mas, usar uma boa heurística pode economizar muito tempo na concepção de boas arquiteturas de redes neurais.

Já temos então um design para a nossa rede neural. Agora precisamos definir como será o processo de aprendizagem do algoritmo, antes de começar a codificar nossa rede em linguagem Python. Usaremos o treinamento com Gradiente Descendente, assunto do próximo capítulo, que aliás eu não perderia por nada, se fosse você, pois aí está a “magia” por trás das redes neurais. Até lá!

Para acompanhar os próximos capítulos e reproduzir os exemplos, você deve ter o Anaconda Python instalado no seu computador com Python versão 3.6.x. Acesse o capítulo 1 do curso gratuito [Python Fundamentos Para Análise de Dados](https://www.datascienceacademy.com.br/course?courseid=python-fundamentos), para aprender como instalar o Anaconda.

# Capítulo 12 – Aprendizado Com a Descida do Gradiente

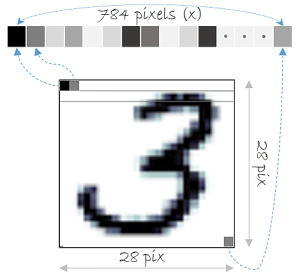
No capítulo anterior definimos o design para a nossa rede neural e agora podemos começar o processo de aprendizado de máquina. Neste capítulo você vai compreender o que é o Aprendizado Com a Descida do Gradiente.

A primeira coisa que precisamos é um conjunto de dados para o treinamento da rede. Usaremos o conjunto de dados [MNIST](http://yann.lecun.com/exdb/mnist/), que contém dezenas de milhares de imagens digitalizadas de dígitos manuscritos, juntamente com suas classificações corretas. O nome MNIST vem do fato de que é um subconjunto modificado de dois conjuntos de dados coletados pelo NIST, o Instituto Nacional de Padrões e Tecnologia dos Estados Unidos. Aqui estão algumas imagens do MNIST:

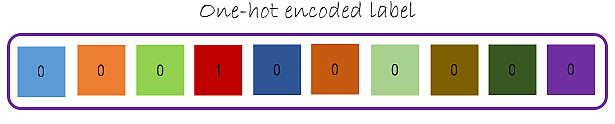


O MNIST tem duas partes. A primeira parte contém 60.000 imagens para serem usadas como dados de treinamento. Essas imagens são amostras de manuscritos escaneados de 250 pessoas, metade dos quais funcionários do Bureau do Censo dos EUA e metade dos estudantes do ensino médio. As imagens estão em escala de cinza e 28 por 28 pixels de tamanho. A segunda parte do conjunto de dados MNIST tem 10.000 imagens a serem usadas como dados de teste, também 28 por 28 pixels em escala de cinza. Usaremos os dados do teste para avaliar o quão bem a nossa rede neural aprendeu a reconhecer os dígitos. Para fazer deste um bom teste de desempenho, os dados de teste foram retirados de um conjunto diferente de 250 pessoas em relação aos dados de treinamento originais (embora ainda seja um grupo dividido entre funcionários do Census Bureau e alunos do ensino médio). Isso nos ajuda a confiar que nosso sistema pode reconhecer dígitos de pessoas cuja escrita não viu durante o treinamento.

Usaremos a notação x para indicar uma entrada (input) de treinamento. Será conveniente considerar cada entrada de treinamento x (cada imagem) como um vetor de 784 posições (28 x 28 pixels). A imagem abaixo representa como este vetor é construído:

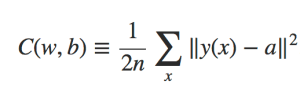


Cada entrada no vetor representa o valor de cinza para um único pixel na imagem. Vamos indicar a saída correspondente desejada por y = y(x), onde y é um vetor com dimensão 10. Por exemplo, se uma imagem de treinamento particular, x, representa um 3, então y(x) = (0,0,0,1,0,0,0,0,0,0)T é a saída desejada da rede . Observe que T aqui é a operação de transposição, transformando um vetor de linha em um vetor comum (coluna). Vamos deixar isso mais claro. Observe a figura abaixo:



Vamos usar os pixels de imagem correspondentes ao fluxo inteiro chamado “features”. Os rótulos são One-Hot Encoded 1-hot. O rótulo que representa a classe de saída da imagem com dígito 3 torna-se “0001000000” uma vez que temos 10 classes para os 10 dígitos possíveis, onde o primeiro índice corresponde ao dígito “0” e o último corresponde ao dígito “9”.

O que queremos é um algoritmo que nos permita encontrar pesos e bias para que a saída da rede se aproxime de y(x) para todas as entradas de treinamento x. Para quantificar o quão bem estamos alcançando esse objetivo, definimos uma função de custo:



Função de Custo Quadrático

Na fórmula acima, w indica a coleta de todos os pesos na rede, b todos os bias (viés), n é o número total de entradas de treinamento, a é o vetor de saídas da rede (quando x é entrada) e a soma é sobre todas as entradas de treinamento x. Claro, a saída a depende de x, w e b, mas para manter a notação simples, eu não indiquei explicitamente essa dependência. A notação ‖v‖ apenas indica a função de comprimento usual para um vetor v. Chamaremos C a função de custo quadrático, que também é conhecido como o erro quadrático médio ou apenas o MSE (Mean Squared Error). Inspecionando a forma da função de custo quadrático, vemos que C (w, b) não é negativo, pois cada termo na soma não é negativo. Além disso, o custo C (w, b) torna-se pequeno, isto é, C (w, b) ≈ 0, precisamente quando y(x) é aproximadamente igual à saída, a, para todas as entradas de treinamento x.

Portanto, nosso algoritmo de treinamento faz um bom trabalho se ele pode encontrar pesos e bias para que C (w, b) ≈ 0. Isso significa basicamente que nosso modelo fez as previsões corretas, ou seja, cada vez que apresentamos ao modelo uma imagem com dígito 3, ele é capaz de reconhecer que se trata do número 3.

Em contraste, o algoritmo não terá boa performance, quando C (w, b) for um valor maior que 0 – isso significaria que nosso algoritmo não está conseguindo fazer as previsões, ou seja, quando apresentado a imagem com o dígito 3, ele não é capaz de prever que se trata de um número 3. Isso ocorre, porque a diferença entre o valor real da saída e o valor previsto pelo modelo, é muito alta. Assim, o objetivo do nosso algoritmo de treinamento será minimizar o custo C(w, b) em função dos pesos e dos bias. Em outras palavras, queremos encontrar um conjunto de pesos e bias que tornem o custo o menor possível. Vamos fazer isso usando um algoritmo conhecido como Descida do Gradiente (Gradient Descent).

Mas antes, uma pergunta. Por que introduzir o custo quadrático? Afinal, não nos interessamos principalmente pelo número de imagens corretamente classificadas pela rede? Por que não tentar maximizar esse número diretamente, em vez de minimizar uma medida, como o custo quadrático? O problema com isso é que o número de imagens corretamente classificadas não é uma “smooth function” dos pesos e bias na rede. Geralmente, fazer pequenas mudanças nos pesos e bias não causará nenhuma alteração no número de imagens de treinamento classificadas corretamente. Isso torna difícil descobrir como mudar os pesos e os bias para melhorar o desempenho. Se, em vez disso, usamos uma “smooth cost function”, como o custo quadrático, revela-se fácil descobrir como fazer pequenas mudanças nos pesos e nos bias para obter uma melhoria no custo. É por isso que nos concentramos primeiro na minimização do custo quadrático e somente depois examinaremos a precisão da classificação.

Mesmo considerando que queremos usar uma “smooth cost function”, você ainda pode se perguntar por que escolhemos a função quadrática. Talvez se escolhêssemos uma função de custo diferente, obteríamos um conjunto totalmente diferente de pesos e bias? Esta é uma preocupação válida e, mais tarde, revisitaremos a função de custo e faremos algumas modificações. No entanto, a função de custo quadrático mostrada anteriormente funciona perfeitamente para entender os conceitos básicos de aprendizagem em redes neurais, então ficaremos com isso por enquanto.

Recapitulando, nosso objetivo na construção de uma rede neural é encontrar pesos e bias que minimizem a função de custo quadrático C (w, b).

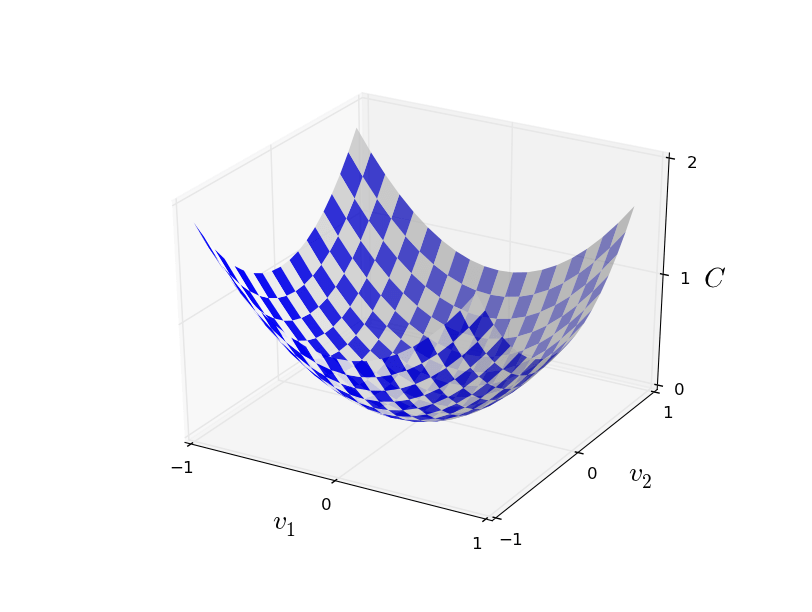
### Descida do Gradiente

A maioria das tarefas em Machine Learning são na verdade problemas de otimização e um dos algoritmos mais usados para isso é o Algoritmo de Descida do Gradiente. Para um iniciante, o nome Algoritmo de Descida do Gradiente pode parecer intimidante, mas espero que depois de ler o que está logo abaixo, isso deixe de ser um mistério para você.

A Descida do Gradiente é uma ferramenta padrão para otimizar funções complexas iterativamente dentro de um programa de computador. Seu objetivo é: dada alguma função arbitrária, encontrar um mínimo. Para alguns pequenos subconjuntos de funções – aqueles que são convexos – há apenas um único minumum que também acontece de ser global. Para as funções mais realistas, pode haver muitos mínimos, então a maioria dos mínimos são locais. Certifique-se de que a otimização encontre o “melhor” minimum e não fique preso em mínimos sub-otimistas (um problema comum durante o treinamento do algoritmo).

Para compreender a intuição da Descida do Gradiente, vamos simplificar um pouco as coisas. Vamos imaginar que simplesmente recebemos uma função de muitas variáveis e queremos minimizar essa função. Vamos desenvolver a técnica chamada Descida do Gradiente que pode ser usada para resolver tais problemas de minimização. Então, voltaremos para a função específica que queremos minimizar para as redes neurais.

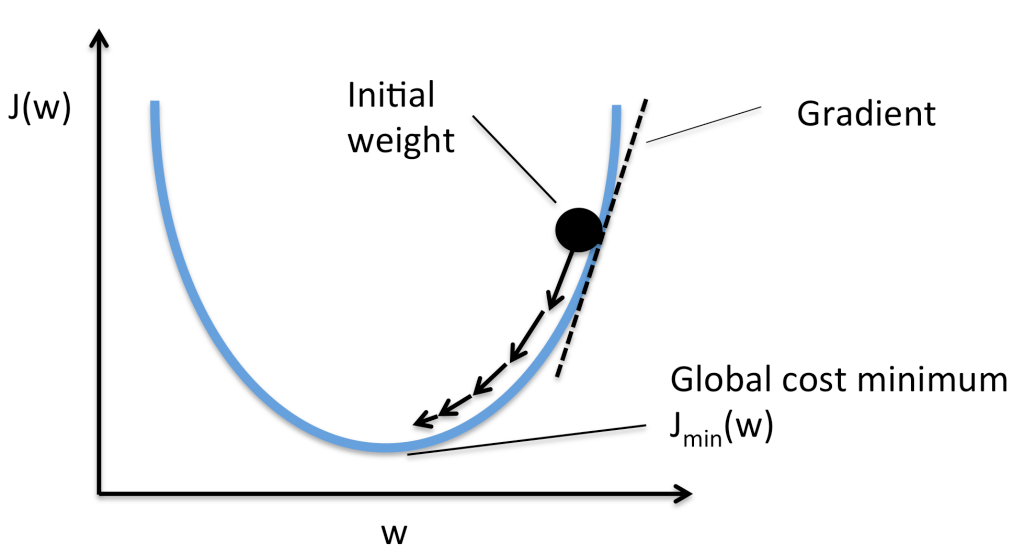
Ok, suponhamos que estamos tentando minimizar alguma função, C(v). Esta poderia ser qualquer função de valor real de muitas variáveis, onde v = v1, v2, …. Observe que eu substitui a notação w e b por v para enfatizar que esta poderia ser qualquer função – não estamos mais pensando especificamente no contexto das redes neurais apenas. Para minimizar C (v), vamos imaginar C como uma função de apenas duas variáveis, que chamaremos v1 e v2, conforme pode ser visto na figura abaixo:



O que queremos é encontrar onde C atinge seu mínimo global. Fica claro, que para a função traçada no gráfico acima, podemos observar facilmente o gráfico e encontrar o mínimo. Mas uma função geral, C, pode ser uma função complicada de muitas variáveis, e geralmente não será possível apenas observar o gráfico para encontrar o mínimo.

Uma maneira de atacar o problema é usar Cálculo (especificamente Álgebra Linear) para tentar encontrar o mínimo de forma analítica. Podemos calcular [derivadas](https://pt.wikipedia.org/wiki/Derivada) e depois tentar usá-las para encontrar lugares onde C é um extremum. Isso pode funcionar quando C é uma função de apenas uma ou algumas variáveis. Mas vai se transformar em um pesadelo quando tivermos muitas outras variáveis. E para as redes neurais, muitas vezes queremos muito mais variáveis – as maiores redes neurais têm funções de custo que dependem de bilhões de pesos e bias de uma maneira extremamente complicada. Usando “apenas” Cálculo para minimizar isso, não funcionará e precisamos de algo mais! Precisamos de um algoritmo de otimização capaz de minimizar C (v).

Felizmente, há uma analogia que nos ajuda a compreender como encontrar a solução. Começamos por pensar em nossa função como uma espécie de vale e imaginamos uma bola rolando pela encosta do vale, conforme pode ser visto na figura abaixo. Nossa experiência diária nos diz que a bola acabará rolando para o fundo do vale. Talvez possamos usar essa ideia como forma de encontrar um mínimo para a função? Escolheríamos aleatoriamente um ponto de partida para uma bola (imaginária), e então simularíamos o movimento da bola enquanto ela rola até o fundo do vale. Poderíamos fazer essa simulação simplesmente por derivadas de computação da função C – essas derivadas nos diriam tudo o que precisamos saber sobre a “forma” local do vale, e, portanto, como nossa bola deve rolar.



Representação da Descida do Gradiente (com o objetivo de minimizar a função de custo)

Ou seja, a Descida do Gradiente é um algoritmo de otimização usado para encontrar os valores de parâmetros (coeficientes ou se preferir w e b – weight e bias) de uma função que minimizam uma função de custo. A Descida do Gradiente é melhor usada quando os parâmetros não podem ser calculados analiticamente (por exemplo, usando álgebra linear) e devem ser pesquisados por um algoritmo de otimização.

O procedimento começa com valores iniciais para o coeficiente ou coeficientes da função. Estes poderiam ser 0.0 ou um pequeno valor aleatório (a inicialização dos coeficiente é parte crítica do processo e diversas técnicas podem ser usadas, ficando a escolha a cargo do [Cientista de Dados](https://www.datascienceacademy.com.br/pages/formacao-cientista-de-dados) e do problema a ser resolvido com o modelo). Poderíamos iniciar assim nossos coeficientes (valores de w e b):

**coeficiente = 0,0**

O custo dos coeficientes é avaliado ligando-os à função e calculando o custo.

**custo = f (coeficiente)**

ou

**custo = avaliar (f (coeficiente))**

A derivada do custo é calculada. A derivada é um conceito de Cálculo e refere-se à inclinação da função em um determinado ponto. Precisamos conhecer a inclinação para que possamos conhecer a direção (sinal) para mover os valores dos coeficientes para obter um custo menor na próxima iteração.

**delta = derivado (custo)**

Agora que sabemos da derivada em que direção está em declive, podemos atualizar os valores dos coeficientes. Um parâmetro de taxa de aprendizagem (alfa) deve ser especificado e controla o quanto os coeficientes podem mudar em cada atualização.

**coeficiente = coeficiente – (alfa \* delta)**

Este processo é repetido até que o **custo dos coeficientes** (**função de custo**) seja 0,0 ou próximo o suficiente de zero, indicando que as saídas da rede estão cada vez mais próximas dos valores reais (saídas desejadas).

A Descida do Gradiente é simples, mas exige que seja calculado o gradiente da função de custo ou a função que você está otimizando, mas além disso, é muito direto. Em resumo:

Você divide seus dados em amostras e a cada amostra (sample), você passa as entradas pela rede, multiplica pelos pesos, soma, e no final você vai ter sua saida (a previsão da rede). Você então compara a saída da sua rede com o a resposta certa, calcula o erro, e então retroage esse erro (backpropagation), ajustando os pesos de cada neurônio de cada camada. Quando você acabar de fazer a atualização dos pesos, uma nova amostra é introduzida e ela será multiplicada pelos pesos já atualizados. Esse processo de atualizar os pesos é que é chamado de “aprendizado”.

Se você observar os algoritmos mais atuais, todos trabalham dentro de um conceito relativamente novo chamado de mini-lotes (mini-batches). Para otimizar a performance, o que se faz é passar pela rede múltiplas amostras (por exemplo 128 amostras), calcular o erro médio delas e então realizar o backpropagation e a atualização dos pesos. Do ponto de vista da atualização dos pesos, 1 amostra = 128 amostras. Esse é um conceito mais novo, necessário principalmente no treinamento de grandes modelos de Deep Learning.

Em seguida, veremos como podemos usar isso em algoritmos de aprendizado de máquina.

### Batch Gradient Descent em Aprendizado de Máquina

O objetivo de todos os algoritmos supervisionados de aprendizagem de máquina é estimar uma função de destino (f) que mapeia dados de entrada (X) para as variáveis ​​de saída (Y). Isso descreve todos os problemas de classificação e regressão (aprendizagem supervisionada).

Alguns algoritmos de aprendizagem de máquina têm coeficientes que caracterizam a estimativa de algoritmos para a função alvo (f). Diferentes algoritmos têm diferentes representações e diferentes coeficientes, mas muitos deles requerem um processo de otimização para encontrar o conjunto de coeficientes que resultam na melhor estimativa da função alvo. Os exemplos comuns de algoritmos com coeficientes que podem ser otimizados usando descida do gradiente são Regressão linear e Regressão logística.

A avaliação de quão próximo um modelo de aprendizagem de máquina estima a função de destino pode ser calculada de várias maneiras, muitas vezes específicas para o algoritmo de aprendizagem de máquina. A função de custo envolve a avaliação dos coeficientes no modelo de aprendizagem de máquina calculando uma previsão para o modelo para cada instância de treinamento no conjunto de dados e comparando as previsões com os valores de saída reais e calculando uma soma ou erro médio (como a Soma de Residuais Quadrados ou SSR no caso de regressão linear).

A partir da função de custo, uma derivada pode ser calculada para cada coeficiente para que ele possa ser atualizado usando exatamente a equação de atualização descrita acima.

O custo é calculado para um algoritmo de aprendizado de máquina em todo o conjunto de dados de treinamento para cada iteração do algoritmo de descida de gradiente. Uma iteração do algoritmo é chamada de um lote e esta forma de descida do gradiente é referida como descida do gradiente em lote (Batch Gradient Descent).

A descida do gradiente em lote é a forma mais comum de descida do gradiente em Machine Learning.

### Stochastic Gradient Descent em Aprendizado de Máquina

A Descida do Gradiente pode ser lenta para executar em conjuntos de dados muito grandes. Como uma iteração do algoritmo de descida do gradiente requer uma previsão para cada instância no conjunto de dados de treinamento, pode demorar muito quando você tem muitos milhões de instâncias.

Em situações em que você possui grandes quantidades de dados, você pode usar uma variação da descida do gradiente chamada Stochastic Gradient Descent.

Nesta variação, o procedimento de descida do gradiente descrito acima é executado, mas a atualização para os coeficientes é realizada para cada instância de treinamento, em vez do final do lote de instâncias.

O primeiro passo do procedimento exige que a ordem do conjunto de dados de treinamento seja randomizada. Isto é, misturar a ordem que as atualizações são feitas para os coeficientes. Como os coeficientes são atualizados após cada instância de treinamento, as atualizações serão barulhentas saltando por todo o lado, e assim o custo correspondente funcionará. Ao misturar a ordem para as atualizações dos coeficientes, ela aproveita essa caminhada aleatória e evita que ela fique “distraída” ou presa.

O procedimento de atualização para os coeficientes é o mesmo que o anterior, exceto que o custo não é somado em todos os padrões de treinamento, mas sim calculado para um padrão de treinamento.

A aprendizagem pode ser muito mais rápida com descida de gradiente estocástica para conjuntos de dados de treinamento muito grandes e muitas vezes você só precisa de um pequeno número de passagens através do conjunto de dados para alcançar um conjunto de coeficientes bom o suficiente.

Ufa, você ainda está aí? Entende agora porque Cientistas de Dados e Engenheiros de IA devem ser muito bem remunerados? Eles são os “magos” que estão ajudando a transformar o mundo com Machine Learning. E este capítulo foi apenas uma breve introdução! Voltaremos a este assunto mais a frente no livro, quando estudarmos outros algoritmos. Mas caso você queira aprender em detalhes como tudo isso funciona e criar seus modelos usando linguagens R, Python, Scala ou Java, para aplicações comerciais, confira: [Machine Learning](https://www.datascienceacademy.com.br/pages/curso-machine-learning" \t "_blank), [Machine Learning com Scala e Spark](https://www.datascienceacademy.com.br/course?courseid=machine-learning-com-linguagem-scala" \t "_blank), [Deep Learning](https://www.datascienceacademy.com.br/course?courseid=deep-learning-i" \t "_blank) e [Análise Preditiva com Machine Learning em Java](https://www.datascienceacademy.com.br/course?courseid=machine-learning-em-java).