Redes neurais artificiais

Faz tempo que o homem quer saber como funciona e onde reside o motor que nos permite pensar, sentir, executar, amar e odiar, decidir, aprender e construir coisas. Uma boa maneira de saber mais sobre isto é construir artefatos que imitem este comportamento. Pode ser uma rede neural.

Neste texto, para chegar nas RNAs (redes neurais artificiais), precisa-se manobrar em 2 vertentes: o computador – já que uma RNA sempre precisa de um para se manifestar, e o cérebro humano, que é o modelo funcional de uma RNA. As RNAs nasceram para imitar o (pouco que sabemos) de um cérebro humano.

Em 350 aC, Hipócrates propõe que o cérebro está relacionado às sensações e é a sede da inteligência. Aristóteles, achava o contrário: para ele o centro do intelecto estava no coração. Já por volta de 180 dC, Galeno, um médico romano, após dissecações em animais sugeriu algum entendimento sobre o cérebro e suas funções.

É de se ressaltar que em paralelo se desenvolvia a matemática, a verdadeira[1] ferramenta de que dispõe a humanidade para pensar. Desde os filósofos gregos até os matemáticos contemporâneos, às vezes fica difícil apontar para alguém e cravar: é matemático ou é filósofo. [2]

Um sujeito que merece uma boa olhada é Gottfried Leibniz, um alemão que nasceu em 1646. Sua pegada matemática é inquestionável: autor (talvez co-autor?) do cálculo diferencial e integral. Já a mão filosófica está em duas propostas: *Characteristica Universalis* (Característica Universal) – uma linguagem universal e formal que pudesse expressar todos os conceitos matemáticos, científicos e metafísicos de forma unívoca. Também propôs o *Calculus Ratiocinator* (Cálculo Raciocinador) que junto com a proposta anterior seria um motor de inferência lógico.

Nada disso teve implementação, mas não se iludam: o trabalho de um computador e sobretudo a moderna IA bebeu muito lá no velho Leibniz.

Avançamos no tempo e chega-se a 1815, quando nasce um inglês de nome George Boole, matemático e filósofo. Sua contribuição é basilar para a computação. Por meio de sua Álgebra Booleana, pode-se converter um conjunto de afirmativas lógicas em operações matemáticas e nelas aplicar toda a matemática conhecida (e a ainda por inventar). Quando publicada em 1854 foi entendida como uma curiosidade, mas ela é a a linguagem fundamental para o design de circuitos digitais e para a programação de computadores. Pelo seu trabalho, a lógica deixa de ser apenas filosofia e passa a ser matemática.

Voltando ao estudo do cérebro, precisamos ir para a Espanha, onde em 1852 nasce Santiago Ramon y Cajal (médico, prêmio nobel de medicina de 1906). Em 1887, teve acesso ao método de coloração de Camilo Golgi (ambos ganharam o nobel juntos), que permitia individualizar e colorir um neurônio. Daqui, Cajal propôs a Doutrina do Neurônio, pela qual o sistema nervoso é formado por unidades individuais e discretas, os neurônios que se comunicam através de junções chamadas sinapses. Golgi, ao contrário, defendia que o sistema nervoso era uma rede contínua. [3] Uma curiosidade é que o neurônio é talvez a única célula do corpo que não se reproduz, não se renova.

Em resumo, Cajal disse:

- Temos muitos (bilhões) de neurônios, a célula fundamental do sistema nervoso.
- Cada um tem muitas ramificações: dendritos
- Cada um tem uma única terminação mais robusta: o axônio
- Dendritos e axônios se conectam nas sinapses, formando uma rede
- O processamento do neurônio ocorre no núcleo da célula.

No século XX, as coisas se aceleram. Primeiro vamos à Inglaterra de 1912, em Londres, onde nasce Alan Mathison Turing, lógico e matemático. Em 1936 publica *"On Computable Numbers, with an Application to the Entscheidungsproblem"* uma descrição matemática brilhante de como deveria ser um computador quando e se este viesse a ser um dia inventado. Também provou que há certos temas que nunca poderão ser tratados por um computador. Esta segunda afirmação se apoia no Teorema da Incompletude de Kurt Göedel de 1931.

Só este trabalho de 1936 já garantiria Turing no top 5 dos matemáticos do século 20, mas há mais. Durante a II guerra mundial, foi o autor da quebra da Enigma (máquina criptográfica nazista, esta história está no filme de 2014, *The imitation game*, indicado a 8 oscars e vencedor na categoria melhor roteiro adaptado). Depois da guerra escreveu *"Computing Machinery and Intelligence"* onde além de deitar e rolar sobre Inteligência Artificial, descreveu o teste de Turing, ainda um modelo para detectar inteligência plenamente funcional nos dias de hoje. Finalmente, no final da vida trabalhou com Morfogênese: padrões e formas em organismos vivos. *Esse sujeito, na minha opinião, era um ET: em 1951, fez uma previsão: Em 50 anos um computador vai ganhar uma partida de xadrez do campeão mundial humano.*

[1] Escreví verdadeira e única. Mas depois, pensando mais achei um pouco presunçoso e corrigi. Mas, lá no fundo continuo achando isso

[2] Exemplos: René Descartes, filósofo (*cogito, ergo sum*) e matemático: geometria cartesiana; Russell, filósofo (*vencedor do Nobel em literatura de 1950*) e matemático: *Principia Mathemática*; Pascal com seu *Pensamentos* e também a teoria das probabilidades.

[3] Cajal tem histórias ótimas: antes de médico foi palhaço de circo. Sua residência foi na guerra de Cuba, como médico militar. Já famoso e respeitado, nunca deixou de acudir a uma reunião (alcoólica) diária com os amigos entre 4 da tarde e 8 da noite. Além de médico foi um humanista, autor de ensaios e memórias, como "Recuerdos de mi vida". Tem uma história deliciosa, tirada de sua biografia: Já considerado o médico mais famoso da Espanha, um dia ao chegar para a tertúlia alcoólica diária, se defrontou com um acidente: o taberneiro havia caido e quebrado a perna. A decisão dos amigos foi unânime: nada de hospital, aqui está o melhor médico possível. O Cajal acudiu meio incomodado, afinal ele estudava o cérebro. Mas, fazer o quê: examinou o homem, coçou a barba e finalmente declarou *acho melhor levar ele a um médico.*

1951+50=2001. Errou por 4 anos, em 1997 Deep Blue ganhou uma série de Gary Kasparov, campeão mundial. Que outra previsão na área da TI teve tal grau de acerto?

Pula-se o Atlântico e vamos aos Laboratórios Bell, onde em 1948, Claude Shannon, um engenheiro elétrico publica *"A Mathematical Theory of Communication"* nada mais do que a tradução da Álgebra Booleana em circuitos elétricos. Para se ter ideia da importância deste trabalho, ele é a origem de um computador eletrônico funcional, com seus bits e bytes, conceitos que – a propósito, foram popularizados por Shannon neste trabalho. Em resumo, Boole pega a lógica e a transforma em matemática. Shannon pega a matemática e a transforma em engenharia: estava viabilizada a conversão da lógica em engenharia elétrica. Nascia o computador para automatizar a matemática.

Na outra vertente dessa história (a do cérebro), vamos para 1943, quando dois americanos, Walter Pitts[4] e Warren McCulloch publicam um artigo onde modelam matematicamente um neurônio natural. Em resumo, e simplificadamente, tal modelo diz que as entradas (dendritos) são multiplicadas por pesos associados a cada dendrito. O resultado é somado e se ultrapassar um valor (conhecido como limiar), o axônio dispara e o axônio lança o sinal. O aprendizado da rede reside nos pesos associados à soma, que são modificados durante o treinamento da rede. Lembrar que se fala em **rede** neural, logo este processamento se repete para todos os neurônios ligados em rede.

Aliás, é hora de fazer uma distinção importante entre os 2 modelos de computação. O modelo determinístico ou algorítmico, conta com algoritmos específicos para a solução dos problemas. Por exemplo, para escrever uma folha de pagamento (determinístico por natureza), o programador precisa escrever um código específico que implemente as regras que determinam o cálculo do salário. Em compensação, o software, uma vez bem feito, sai produzindo os resultados esperados de prima.

No modelo de redes neurais que estamos vendo também chamado de conexionista, há diferenças importantes: o algoritmo em termos básicos é sempre o mesmo, acima descrito (pondera as entradas, soma e dispara ou não). O que muda entre um reconhecedor de placas de veículos e um gerador de música, por exemplo, é o processo de treinamento. Qualquer rede neural precisa – depois de pronta – ser submetida a um treinamento. Nele, casos positivos e negativos são submetidos à rede, e após cada ciclo, os pesos são reconfigurados a depender do desempenho anterior da rede.

O artigo de McCulloch e Pitts é a referência zero de tudo o que se publicou e produziu depois em redes neurais (e na moderna IA, a propósito).

No final da década de 1950, Rosenblatt (universidade de Cornell) deu o próximo passo: conectando diversos neurônios e criando uma rede. Chamou esta rede de *perceptron*. Nela, os neurônios se dispõem em camadas. Funcionava, mas como a rede poderia aprender ?

Em 1949, Hebb, um biólogo que estudava o comportamento animal, propôs o princípio segundo o qual um aprendizado em sistemas nervosos complexos poderia ser reduzido a um processo local, simplesmente (?!) modificando as sinapses de acordo com exemplos. Como se viu, as sinapses, no modelo matemático são representadas pelos pesos numéricos.

Neste ponto da história, a teoria das redes neurais sofre com 2 problemas. Em 1969, um sujeito respeitadíssimo na área de IA, Marvin Minsky[5], escreveu um livro desancando a teoria acima e informando que as redes neurais só eram capazes de resolver problemas simples (em matemática *linearmente separáveis*) e portanto eram mera curiosidade meio inútil. [6]

Outro problema das redes era a falta de *hardware*. De fato, um computador da década de 70 era muito pequeno, além de absurdamente caro. Neste computador, uma rede poderia ter talvez 100 ou 200 neurônios no máximo, reforçando o caráter de curiosidade meio inútil, acima citado.

Mas o principal problema aqui era outro: como distribuir o erro cometido entre os diversos neurônios usados para calcular uma decisão errada da rede ? (Lembremos que este é o procedimento básico do aprendizado: recompensar o acerto e punir o erro. É só pensar em como ensinar um cachorrinho a não fazer xixi dentro de casa). Em redes de 1 camada (aquelas que só resolvem problemas simples) é fácil detectar o neurônio errado, mas numa rede de várias camadas (a que resolve problemas mais complexos) parecia impossível fazer uma distribuição das correções necessárias. Este fato (mais do que a falta de computador ou a *boutade* do Minsky) é que determinou a dormência das redes neurais entre os anos 70/90.

Em 1986, Rumelhart, Hinton e Williams, publicam um artigo sugerindo um método por eles chamado de *backpropagation* na qual propõem uma maneira de implementar o treinamento (na base da recompensa/punição) em diversas camadas, usando derivadas parciais do erro quadrático cometido. Foi o que bastou para o tema voltar a ganhar relevância. Não podemos nos esquecer do apelo das redes neurais: o algoritmo é sempre o mesmo, pelo quê 80% do trabalho para resolver QUALQUER problema já está feito.

[4] A história de vida de Walter Pitts é notável. O pai o tirou da escola para trabalhar. Aos 12 anos, um dia, ele entrou na biblioteca de Chicago e apanhou o livro *Principia Mathematica* ficando 3 dias lendo-o antes de voltar para casa. Escreveu ao autor (Bertrand Russel), apontando os erros que ele achava que havia no texto. Russel, depois de ler a carta, tomou um susto e imediatamente convidou o remetente para estudar com ele em Cambridge na Inglaterra. Pitts recusou informando que não podia ir: só tinha 12 anos. Em 1943 recebeu um PhD do MIT com a idade de 20 anos. Morreu aos 46, de cirrose hepática.

[5] Este cara andou por aqui no Brasil, trazido pela IBM. Estive com ele pessoalmente em São Paulo. A história que segue é verdadeira, meus olhos viram: Um homem grandalhão com alguns (muitos) quilos adicionais. Vestia terno e gravata, elegante. Começou a palestra e vivenciava algum incômodo. Logo pediu desculpas e afirmou ser incapaz de pensar portando um paletó. Ato contínuo arrancou o dito cujo e em mangas de camisa deu uma bela palestra.

[6] A motivação de Minsky era reforçar a outra vertente da IA à época: o processamento simbólico. 90% dos pesquisadores da IA nesse momento defendiam o simbolismo: a elaboração de um alfabeto de ideias que em conjunto resolveriam problemas. Parecia bom, mas apresentava dois problemas: No primeiro, cada solução de um problema exige um alfabeto próprio. Logo são/serão necessários infinitos alfabetos. O segundo: onde está e como é um alfabeto desses ? Ninguém sabia e ninguém sabe até hoje.

Em 1992, recebemos a visita no CEFET (eu era aluno do mestrado) de um aluno do doutorado não me lembro de qual universidade, em que ele contou essa história aqui citada, sendo que na época era pura novidade para todo mundo. Eu lembro que fiquei bem impressionado.

Os anos 2000 viram as redes neurais explodindo, passamos a ter máquina abundante. Todos aqueles simbólicos do século XX, se renderam à evidência e migraram para as redes, alguns fazendo barulho, a maioria silenciosamente, meio encabulados.

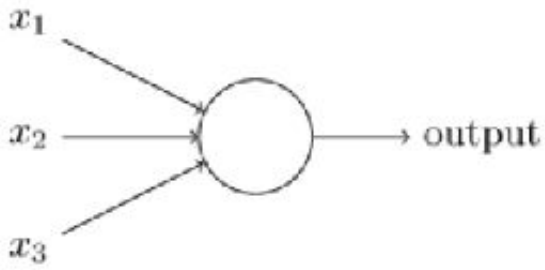
Em 2011, a equipe de pesquisa Google Brain, começou a desenvolver uma rede neural genérica que se aproveitaria da infraestrutura gigantesca do Google. Foi batizada de DistBelief e teve um grande sucesso ao impulsionar aplicações de IA da própria Google.

Mas, era proprietária e por ser a primeira versão de alguma coisa, com diversas imperfeições[7]. Com o que se aprendeu, o Google soltou em 2015 o TensorFlow. O nome sugere que uma rede neural é um conjunto de tensores (matrizes numéricas multidimensionais representando neurônios) e fluxos entre neurônios. Rapidamente TensorFlow se tornou uma das bibliotecas mais populares do mundo em aprendizado por máquina. De novo é software livre.

Há diversos implementações famosas como um sistema que detecta se uma foto contém um gato ou um cachorro. A rede foi treinada[8] com 10.000 fotos de cada um, e depois do treinamento, a rede tem um desempenho próximo a 99% de acerto ao ser alimentada com uma foto inédita de um dos dois bichos. A foto pode ser escura, estar borrada, ou cheia de ruido. Para um humano é quase trivial, mas pense na dificuldade do ponto de vista de um computador: uma imagem de 200×400 pixels (por exemplo) é apenas uma matriz numérica de 80.000 números.

Como uma rede neural é

Usando um perceptron, que é o primeiro modelo de neurônio artificial. Um perceptron tem várias entradas binárias $x_1, x_2, ...x_n$ e produz uma única saída binária.



Além das entradas, há pesos $w_1, w_2, ...w_n$ que são números reais e que descrevem a importância de cada entrada para calcular a saída. A saída é assim calculada:

$$\text{saida} \quad = \quad \begin{cases} 0 & \text{se} \sum_j w_j x_j \leq \text{limite} \\ 1 & \text{se} \sum_j w_j x_j > \text{limite} \end{cases}$$

Vamos ver um exemplo bem simples: Suponha que você precisa decidir se vai ou não ao próximo encontro de amigos dos cachorros de Curitiba. (sim ou não, decisão binária). Antes você decide considerar 3 coisas:

1. Fará sol ?
2. Sua namorada da hora gosta de eventos com animais ?
3. Haverá cobrança de ingressos ?

Note que — por simplicidade — as 3 questões são também binárias. Entretanto, certamente elas não têm o mesmo peso (olha o peso, aí). Para simplificar vamos adotar os seguintes critérios:

- limite = 4.5
- peso do sol 2
- peso da namorada gostar: 3
- peso de ser gratuíto: 4

Aplicando estes valores às condições de entrada e usando a fórmula acima é fácil decidir se você vai ou não ao evento. Antes de continuar, vale guardar o fato de que os pesos acima são os parâmetros da rede. São eles que ascendem aos bilhões nos LLMs modernos.

Note, que se os valores se modificarem (pesos e limite), as mesmas condições de entrada poderão dar origem a outras saídas. Por exemplo, para um limite de 1.5 basta uma das condições (qualquer uma) ser verdadeira, para a resposta ser sim.

[7] Nunca compre/use a versão 1 de qualquer software.

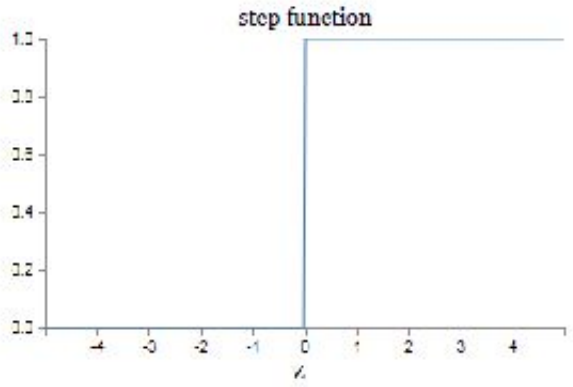
[8] no treinamento, uma foto é mostrada à rede e junto a informação se a foto é de um gato ou de um cão. Este é o treinamento

Para simplificar o cálculo computacional do neurônio vamos fazer uma mudança mecânica no processo, a saber:

$$\text{saida} \quad = \quad \begin{cases} 0 & \text{se} \sum_j w_j x_j - \text{limite} \leq 0 \\ 1 & \text{se} \sum_j w_j x_j - \text{limite} > 0 \end{cases}$$

A vantagem é que os computadores fazem de maneira mais fácil e rápida a comparação para zero. Na prática o limite é considerado como uma entrada adicional ao neurônio valendo sempre 1 e com peso igual ao valor original do limite multiplicado por -1. Esta entrada recebe o nome de *bias*.

Neurônios sigmoidais A equação acima tem um comportamento bizarro que pode ser analisado no seu gráfico



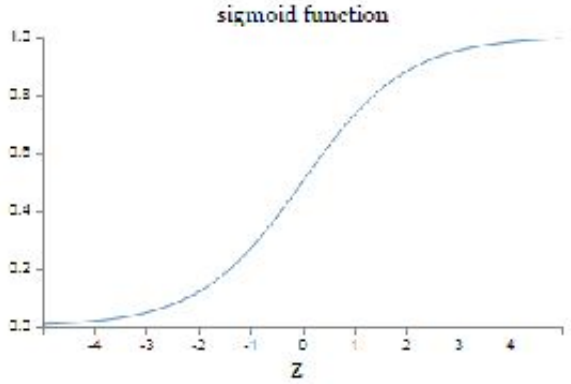
O problema desta função é a sua radicalidade. A resposta é 0 ou 1. Outro problema é que quando estudarmos o algoritmo *backpropagation* vamos precisar calcular a derivada desta função, e por ser descontínua no ponto 0, ela não admite a derivada. Vamos trocar esta função pela função sigmoidal, A função sigmoide de x é

$$y = \frac{1}{1.0 + e^{-x}}$$

ou em Python

```
def sigmoid(z):
    return 1.0/(1.0+np.exp(-z))
```

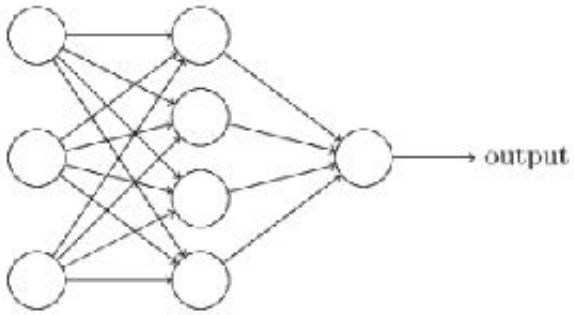
Agora a função pode ser examinada



Aprendizagem Se os pesos fossem atribuídos corretamente antes de começar a usar a rede, estaríamos diante de um engenho simples de criar e usar. A questão é que para a imensa maioria dos problemas que se quer resolver, não se tem ideia de quais são os pesos que devem ser usados. Entra em ação aqui talvez a principal características das redes neurais: a sua capacidade de aprender. Funciona assim: os pesos são inicializados com valores aleatórios, próximos a zero. A seguir a rede é posta a funcionar com um bom volume de dados com o detalhe que – neste momento – sabe-se qual a resposta que a rede deveria dar para cada dado a ela fornecida. Quando a rede acerta, seus pesos são reforçados e quando ela erra, os pesos são corrigidos. Após milhares de uso desta forma, os pesos estão corretamente ajustados.

Camadas intermediárias Em uma rede neural, há as entradas, que por facilidade de notação são representadas por neurônios que *disparam sozinhos* sem ter entradas, como em

e por enquanto as nossas redes neurais não têm realimentação, operando sempre da esquerda para a direita. São conhecidas por isso como *feedforward*. Por razões que são examinadas em detalhe em qualquer livro de redes (inclusive o aqui proposto), estas redes com uma única camada de neurônios de trabalho (além da entrada), não são capazes de mapear regiões côncavas de sim/não na saída. Resta portanto que só são capazes de responder perguntas muito simples. A solução para dar às redes esta capacidade é introduzir uma (ou mais) camadas escondidas entre o cálculo da entrada e a saída, formando esta arquitetura



Note-se que podem haver mais camadas intermediárias.

A(s) camada(s) intermediária(s) tornam a rede capaz de respostas mais complexas e ricas, mas introduzem um problema e tanto: quando um neurônio erra no treinamento, quais pesos devem ser ajustados ? Os da entrada ou os das camada(s) intermediária(s)? E como fazer isso ?

Backpropagation O objetivo do algoritmo é minimizar o erro quadrático

$$\epsilon_k^2 = (y_k - y'_k)^2,$$

ou mais propriamente o erro médio quadrático

$$E(\epsilon^2) = \frac{1}{p}\sum_{k=1}^{p} \epsilon_k^2.$$

O método é o do gradiente descendente ou

$$\nabla U = \frac{\partial U}{\partial x} i + \frac{\partial U}{\partial y} j + \frac{\partial U}{\partial z} k$$

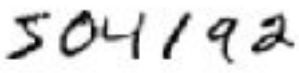
a fórmula pode assustar mas o conceito é simples: dado um ponto na superfície da função, o gradiente é de todas as semi-retas a partir desse ponto, a que mais aponta para cima. Então:

1. O gradiente no ponto w é a derivada parcial do erro médio quadrático.
2. δ é o produto do fator de aprendizagem pelo gradiente em w .
3. a nova direção do aprendizado é a direção antiga mais δ.

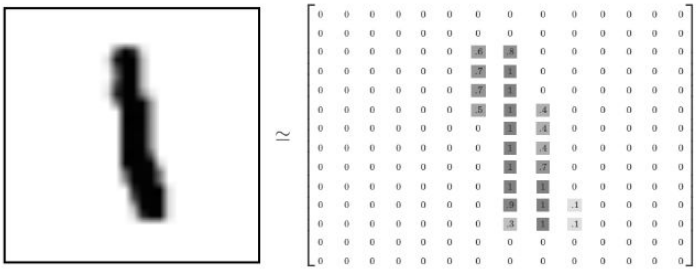
Um exemplo

Vamos imaginar uma rede para reconhecer números desenhados à mão.

Dados do MNIST Existe uma coleção de dados famosa usada para ensinar uma rede neural a reconhecer caracteres numéricos manuscritos, como por exemplo



São 70.000 imagens de dígitos manuscritos capturadas como imagens de 28 × 28 pixels desenhadas por 250 pessoas distintas, a metade de funcionários do Birô do Censo Americano e a outra metade de alunos de segundo grau. A cada imagem, está associado um vetor binário que informa – para o treinamento – qual é esta imagem, através de um vetor binário. (Por exemplo, se a imagem é de um seis, o vetor associado é `[0,0,0,0,0,0,1,0,0,0]`). Tradicionalmente, este conjunto é dividido em subconjuntos de treinamento, validação e teste, com os tamanhos definidos caso a caso. Veja um exemplo de um número 1



Neste caso, as 70.000 imagens de 28 × 28 pixels em níveis de cinza (0=preto, 127=cinza e 255=branco), com qualquer valor intermediário entre 0 e 255 indicando a quantidade de tinta branca no pixel, são transformadas em um vetor numérico de 784 valores entre 0 e 255. Associado a cada um desses 70.000 vetores existe a resposta na forma de um vetor binário de comprimento 10. Por exemplo, {1,0,0,0,0,0,0,0,0,0} indica que a imagem contém um zero, {0,1,0,0,0,0,0,0,0,0} indica um, ... {0,0,0,0,0,0,0,0,0,1} indica 9 e assim por diante.

Cabe ao designer (você!) dividir estas imagens em bloco de treinamento e bloco de controle. A ideia é usar a maior parte dos dados (60.000 ?) para treinamento e usar o resto para avaliar o acerto. A rede nunca deve ser treinada com os dados que a avaliarão, já que se espera que a rede seja competente em avaliar uma imagem NUNCA antes vista. Em resumo, treina-se a rede com 60.000 imagens e depois usam-se as 10.000 restantes para avaliar o desempenho da rede, comparando o que ela achou *vis-a-vis* o que deveria ter achado.

Em particular neste aplicação aqui sugerida (e conhecida como sendo um be-a-bá de redes), usando redes neurais simples, alcançou-se precisão entre 92% e 98% a depender do número de camadas, número de neurônios por camada e parâmetros de treinamento. Usando redes neurais convolucionais mais robustas e otimizadas alcançou-se neste conjunto de dados acertos entre 99% e 99.5%.

O livro acima citado é (distribuído de maneira livre em PDF) é **Neural Networks and Deep Learning** de Michael Nielsen, disponível em http://neuralnetworksanddeeplearning.com.

A explosão com as GPUs

Voltemos 20 anos no tempo. Um dos grandes mercados da computação passaram a ser os videogames. O cenário típico de um aplicativo de jogo é um ambiente completo e complexo, com centenas de objetos/personagens e com o ponto de vista de modificando e exigindo a recriação de cenários em tempo real. Para gerar a imagem final, milhares de pixels tem que ser calculados em paralelo (ao mesmo tempo). Em 1999, a empresa Nvidia lançou uma placa chamada GeForce 256, formalmente a primeira GPU (Graphics Processing Unit). O crédito de sua "invenção" vai para Jensen Huang co-fundador e CEO da Nvidia.

Para comparar com a CPU (Central Processing Unit), pense nesta como sendo a organizadora de todo o ciclo de processamento, a entrada e a saída de dados, a gestão dos comandos e dos programas em execução. Ela é serial (uma tarefa por vez), ou mais modernamente, multi-tarefa, mas com poucas tarefas em paralelo. Já a GPU não tem essa autonomia toda. Ela só efetua cálculos simples envolvendo matrizes. Ela também pode paralelizar centenas ou milhares de processos idênticos, juntando o resultado ao final.

Estamos diante de mais uma manifestação do *atirou no que viu e acertou no que não viu*, já que projetada para construir gráficos realistas em tempo real, a GPU estava prontíssima para calcular fluxos de tensores (ou o comportamento de neurônios, é a mesma coisa).

As GPUs mais modernas chegam a ter 15.000 CUDAs (Computer Unified Device Architecture), que é a unidade de processamento fundamental. Agora, treinar uma rede neural passa pelas seguintes etapas:

- a GPU divide o problema em milhares de tarefas
- cada tarefa vai para uma CUDA. Todas executam a mesma instrução, mas cada uma com seus dados particulares
- a GPU interpreta/agrupa os resultados

LLM: large languagem model

O ano é 2017, 8 funcionários do Google escrevem o artigo "Attention is all you need"[9] propondo uma arquitetura de aprendizado profundo, que eles chamaram de *transformer*. A ideia deles era propor um mecanismo para tradução automática, mas eles perceberam que o transformer era um modelo de linguagem de propósito geral. Este artigo foi citado 173.000 vezes até 2025, colocando-o entre os 10 artigos mais citados no século XXI. Vale citar que seis dos 8 autores nasceram fora dos EUA, os outros 2 são americanos filhos de imigrantes. Nenhum deles ainda está no Google. O transformer é o núcleo da maioria dos LLMs modernos. Sua grande contribuição é a paralelização do processamento, o que acelera o desempenho e permite usar a disponibilidade de GPUs.

[9] uma brincadeira com a canção dos Beatles *all you need is love*

Processar toda a internet

Existe uma organização sem fins lucrativos chamada Common Crawl (https://commoncrawl.org/). Eles varrem a Internet usando pequenos programas (*os crawls*) e recuperando o que é publicado no mundo. São hoje 250 bilhões de páginas e o repositório cresce cerca de 4 bilhões de páginas por mês. Agora em julho de 25 são 419 TB de dados.

Depois vamos para o Fine web, que é um "resumo inteligente" dos dados do common crawl. Ele filtra os dados, eliminando sites pornográficos, agressivos, de apostas, de religião, de astrologia, sites que injetam malware, sites de compras, etc etc.

Depois ele remove (ou tenta remover) conteúdos duplicados. Também é importante lembrar que numa página HTML o que de fato vemos na sua apresentação é cerca de 1/3 de tudo que compõe a página. Os outros 2/3 são tags HTML que formatam a apresentação. Estas, precisam ser retiradas, já que o objetivo é ter apenas um texto corrido representando o conteúdo do que é publicado e não a sua forma. Então o fine web devolve um textão com o que está publicado na Internet. Um ponto importante aqui é remover qualquer dado personalizado. Números de conta, CPFs, e assim por diante precisam ser obliterados.

Tokenização

A próxima etapa é converter um texto imenso em uma lista de tokens. Imagine um dicionário com cem mil palavras. Com ele em mãos, a tokenização converte o texto (lista de palavras) em uma lista de tokens, entendido o token como a posição daquela palavra no dicionário. Um detalhe importante é que palavras compostas podem dar origem a vários tokens. Acompanhe um exemplo, diretamente digitado do tiktokenizer.vercel.app. Aqui, você escolhe uma tokenização contra algum repositório (eu escolhi uma base de 200.000 tokens, a o200k_base) e digitei

```
To be or not to be, that is the question.
sunflower
```

e ele respondeu

```
To be or not to be, that is the question.
sunflower
```

associando cada palavra ou parte de palavra (de acordo com a cor) a seu token

```
1385, 413, 503, 625, 316, 413, 11, 484, 382, 290, 4928, 55
8, 41133, 54730
```

Veja que uma palavra composta (como sunflower=girassol) ganhou 2 tokens=41133 e 54730.

Treinamento inicial

Agora, a rede neural profunda deve ter os seus pesos (lembra deles, lá encima ?) calculados. Isto é que é o treinamento inicial. Haja máquina, os LLMs mais conhecidas são bem grandes [10].

Reforço

Após um primeiro treinamento, a rede é retreinada a partir dos resultados que ela mesma gera. O processo é sofisticado e complexo, só cito ele aqui para reforçar a necessidade de muita (muita) máquina.

Treinamento supervisionado

Agora, entra em ação um treinamento mais especializado. Diversos humanos[11] ensinando diretamente a rede. Essas pessoas criam roteiros de consultas e respostas esperadas por um humano. Este segundo treinamento acaba arredondando o comportamento do LLM. É daqui que vem a habilidade do engenho em responder a perguntas. Numa simplificação e tanto das complexidades associadas poder-se-ia dizer que a forma das respostas vem deste treinamento supervisionado no qual humanos ensinaram a responder questões. Quanto ao conteúdo do que é respondido vem da etapa anterior no qual o conhecimento da internet foi internalizado.

Um bom lugar para estudar esta história (pelo menos até 2021, portanto as LLMs ainda estavam bem no início), está no livro **Criadores de Gênios**, de Cade Metz, um jornalista do New York Times. A linguagem é jornalística.

[10] O Gemini não divulga a dimensão de sua rede já que o Google considera isto um segredo comercial. Entretanto o Gemini 1.0 Ultra, a versão mais capaz da primeira geração, supostamente tem cerca de 550 bilhões de parâmetros. A versão Gemini 1.0 Nano, projetada para dispositivos móveis, tem versões com 1,8 bilhão e 3,25 bilhões de parâmetros. Outro valor importante é a janela de contexto: indica qual o comprimento de frases usadas no treinamento: As versões mais recentes, como o Gemini 1.5 Pro, têm uma janela de contexto massiva de 1 milhão de tokens

[11] Há notícia de 40 contratados pelo GPT fazendo isto.