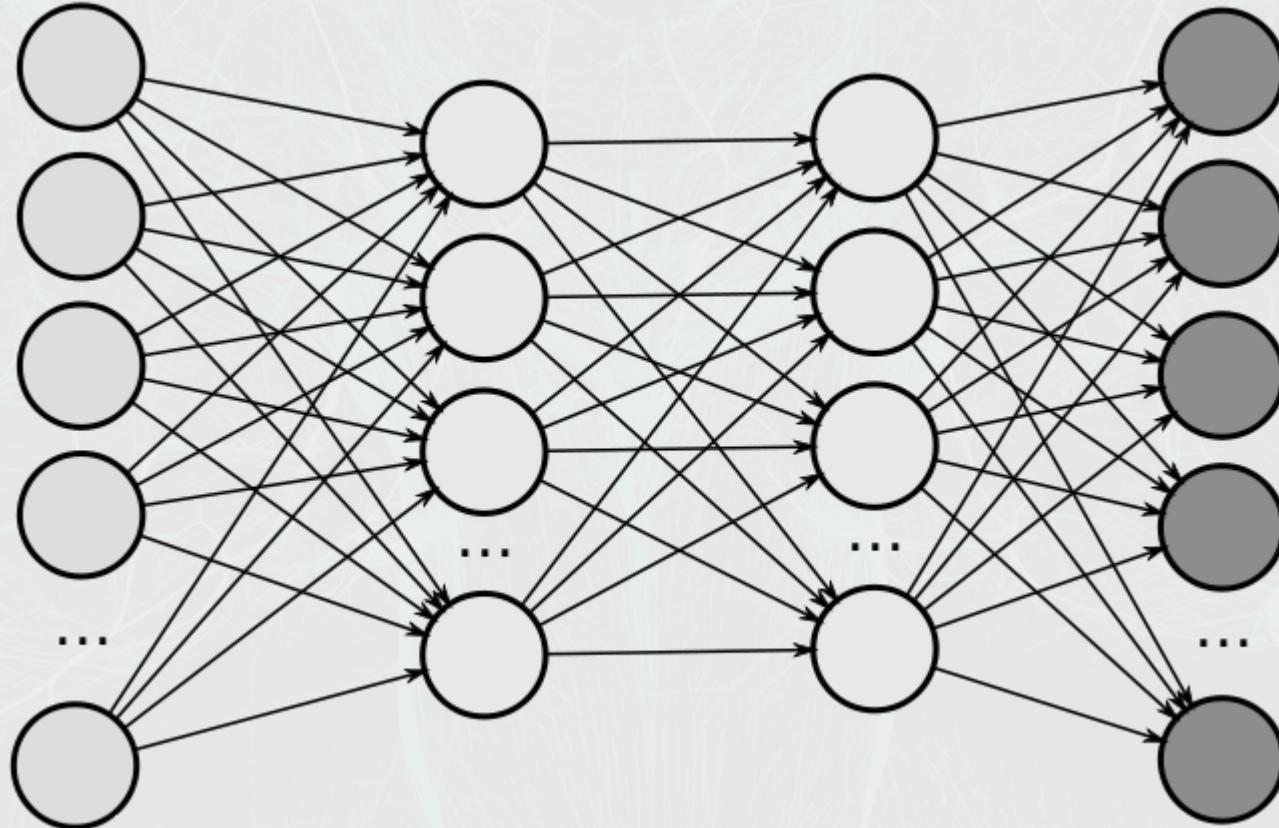


Redes Neurais Artificiais



Redes Neurais Artificiais

Gabriel Felippe

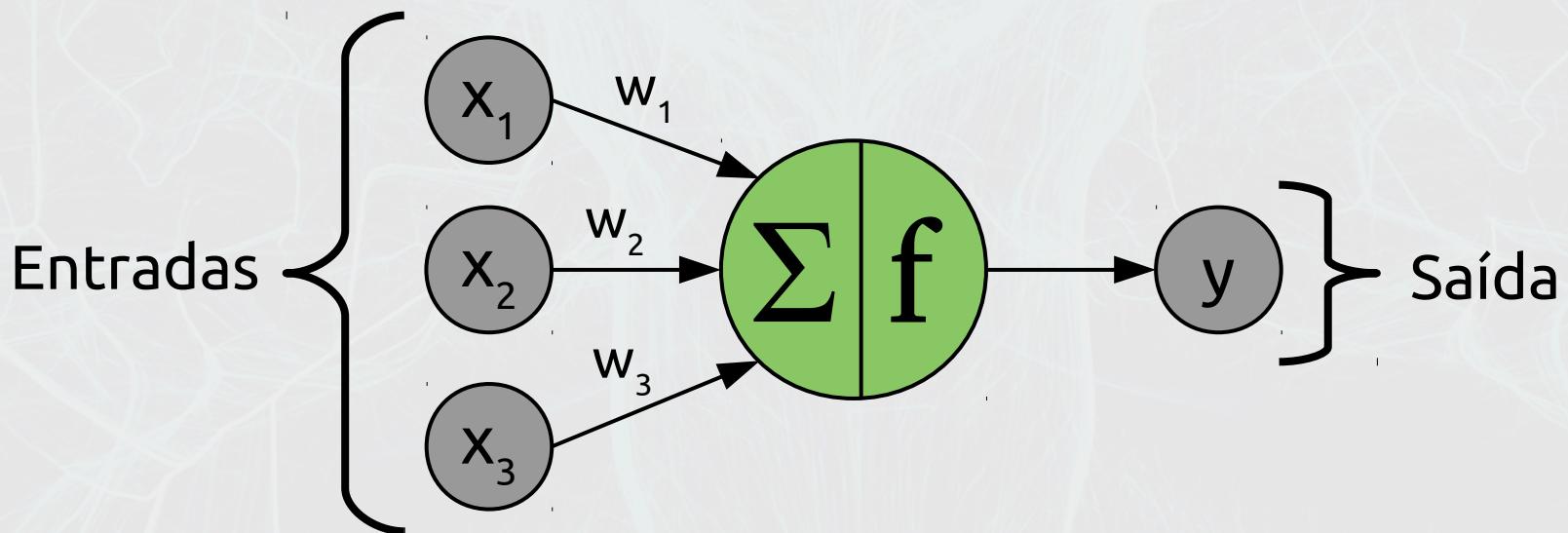
akiradev.netlify.app

Introdução

- As **Redes Neurais Artificiais** são sistemas de computação vagamente inspirados nas redes neurais biológicas que constituem os cérebros dos animais e humanos.
- Uma **RNA** é baseada em uma coleção de unidades conectadas ou nós chamados **neurônios articiais**, que modelam vagamente os neurônios em um cérebro biológico. Cada conexão, como as sinapses em um cérebro biológico, pode transmitir um **sinal** a outros neurônios.
- Um neurônio artificial que recebe um sinal, o processa e pode sinalizar os neurônios conectados a ele.
- O "sinal" em uma conexão é um número real, e a **saída** de cada neurônio é calculada por alguma função não-linear da soma de suas **entradas**.

Introdução

- As conexões são chamadas de arestas. Neurônios e arestas normalmente têm um **weight** (peso) que se ajusta à medida que o aprendizado avança.



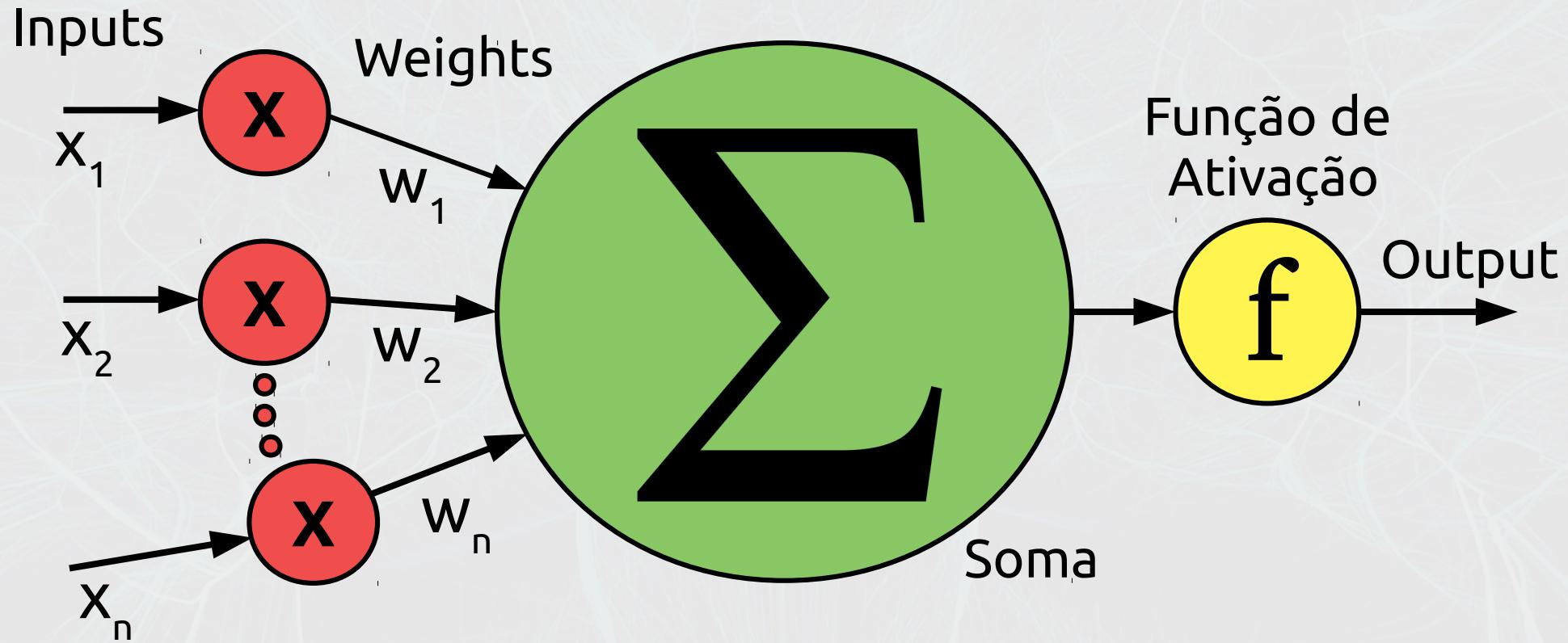
Introdução

- O **weight** aumenta ou diminui a força do sinal em uma conexão.
- Os neurônios podem ter um **threshold**, de modo que um sinal seja enviado apenas se o sinal agregado cruzar esse **threshold**.
- Normalmente, os neurônios são agregados em camadas (**layers**).
- Camadas diferentes podem realizar transformações diferentes em suas entradas.
- Os sinais viajam da primeira camada (a **camada de entrada**) até a última camada (a **camada de saída**), possivelmente depois de atravessar as camadas várias vezes.

Neurônio Artificial

- Um neurônio artificial é uma função matemática concebida como um modelo de neurônios biológicos.
- Neurônios artificiais são **unidades elementares** em uma rede neural artificial.
- O neurônio artificial recebe uma ou mais **entradas** (representando potenciais pós-sinápticos excitatórios e potenciais pós-sinápticos inibitórios em dendritos neurais) e os soma para produzir uma **saída** (ou ativação, representando o potencial de ação de um neurônio que é transmitido ao longo de seu axônio).
- Normalmente, cada entrada é ponderada (**weighted**) separadamente e a soma é passada por uma função não-linear conhecida como **função de ativação**.

Neurônio Artificial



Neurônio Artificial

- Três etapas estão ocorrendo na ilustração anterior:
 - Primeiro, cada **input** (entrada) é multiplicado por um **weight**.
$$X_1 = X_1 * W_1 \dots X_2 = X_2 * W_2 \dots X_n = X_n * W_n$$
 - Em seguida, todos os **inputs weighted**s são adicionados, junto com um termo **bias** (b):
$$(X_1 * W_1) + (X_2 * W_2) + (X_n * W_n) + b$$
 - Por fim, a soma é passada por uma **função de ativação**:
$$f(X_1 * W_1 + X_2 * W_2 + X_n * W_n + b)$$

Neurônio Artificial

- Pense em cada neurônio individual como seu próprio modelo de regressão linear, composto de dados de **entrada**, **weights**, um **bias** e uma **saída**. A fórmula seria mais ou menos assim:

$$\sum_{i=1}^m w_i x_i + bias = w_1 x_1 + w_2 x_2 + w_3 x_3 + bias$$

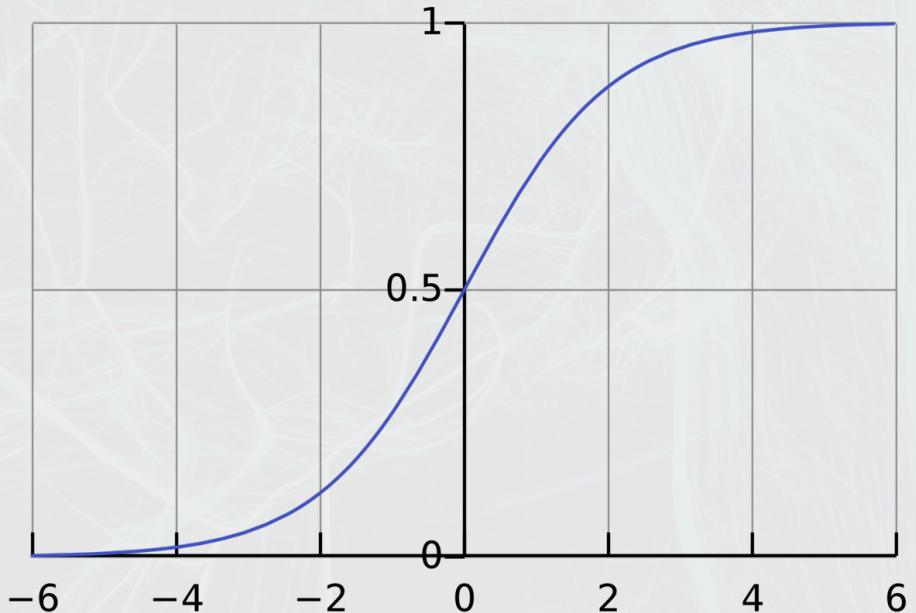
$$\text{output} = f(x) = \begin{cases} 1 & \text{if } \sum w_i x_i + b \geq 0 \\ 0 & \text{if } \sum w_i x_i + b < 0 \end{cases}$$

Neurônio Artificial

- Depois que uma **camada de entrada** é determinada, os **weights** são atribuídos.
- Esse **weights** ajudam a determinar a importância de qualquer variável específica, com as maiores contribuindo de forma mais significativa para a **saída** em comparação com outras entradas. Todas as entradas são então **multiplicadas** por seus respectivos weights e **somadas**.
- Posteriormente, a saída é passada por uma função de ativação, que determina a saída. Se essa saída exceder um determinado limite, ele “dispara” (ou ativa) o neurônio, passando os dados para a próxima camada da rede. Isso resulta na saída de um neurônio se tornando a entrada do próximo neurônio. Este processo de passagem de dados de uma camada para a próxima define esta rede neural como uma **feedforward network**.

Função de Ativação

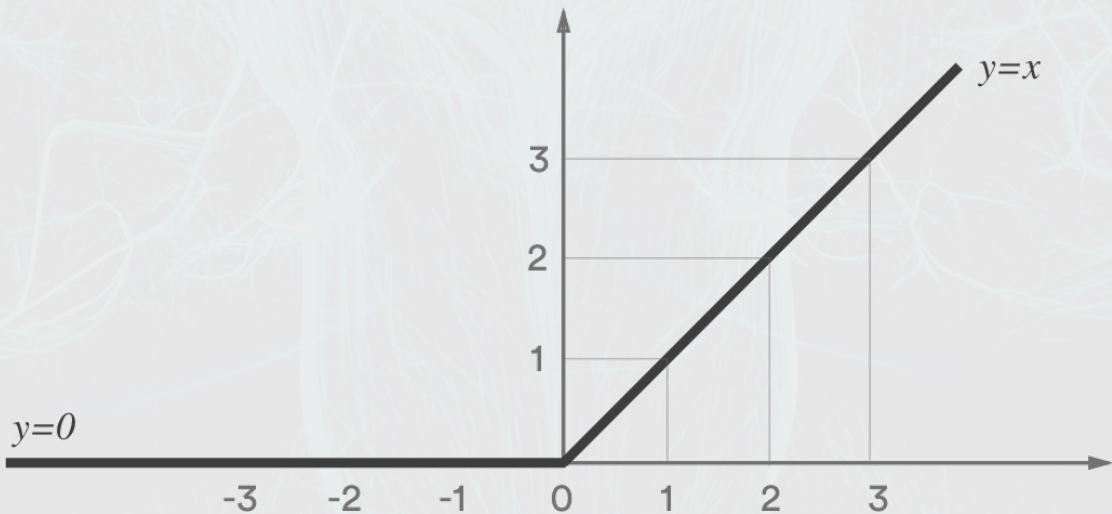
- A função de ativação é usada para transformar uma entrada ilimitada em uma saída que tem uma forma agradável e previsível. Uma função de ativação comumente usada é a **função sigmoid**:



A função sigmoid emite apenas números no intervalo $(0,1)$. Podemos pensar nisso como compactar $(-\infty,+\infty)$ para $(0,1)$. Grandes números negativos tornam-se ~ 0 e grandes números positivos tornam-se ~ 1 .

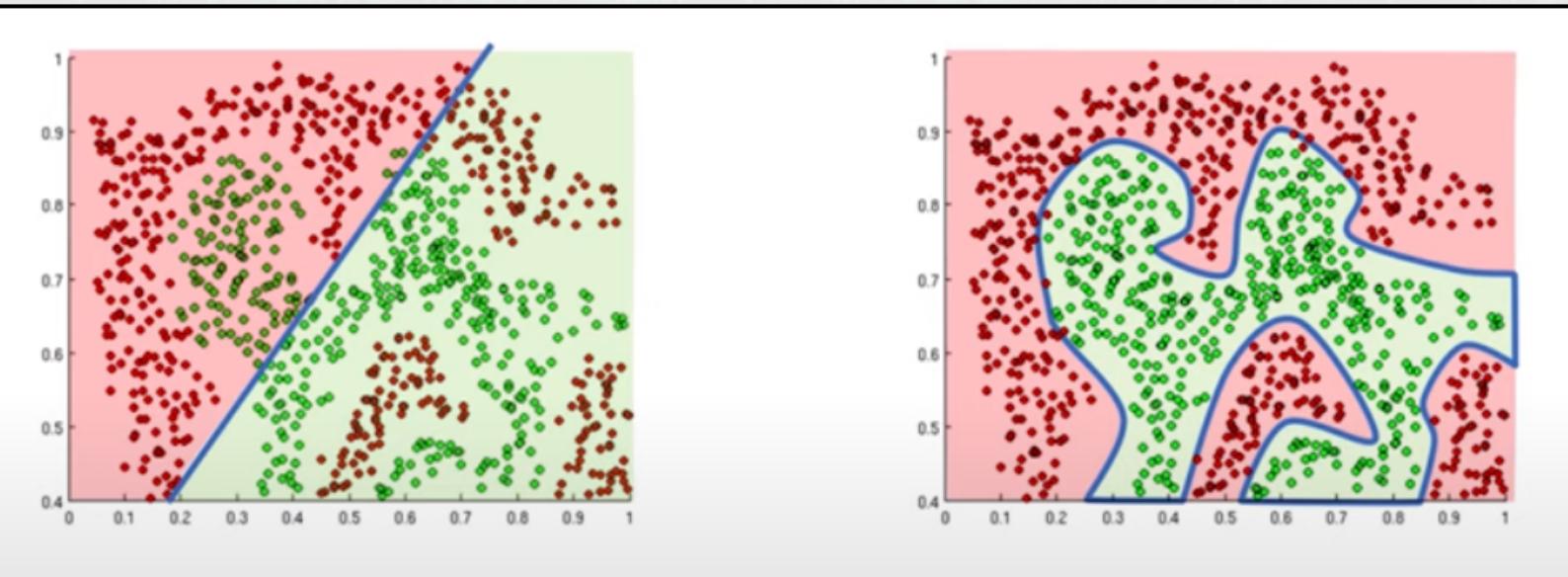
Função de Ativação

- A função de ativação linear retificada ou **ReLU** é uma função linear piecewise que produzirá a própria entrada diretamente se ela for positiva, caso contrário, ela produzirá zero. Ela se tornou a função de ativação padrão para muitos tipos de redes neurais porque um modelo que a usa é mais fácil de treinar e geralmente atinge melhor desempenho.



Importância das Funções de Ativação

- O propósito das funções de ativação é **introduzir não-linearidade** às redes neurais.



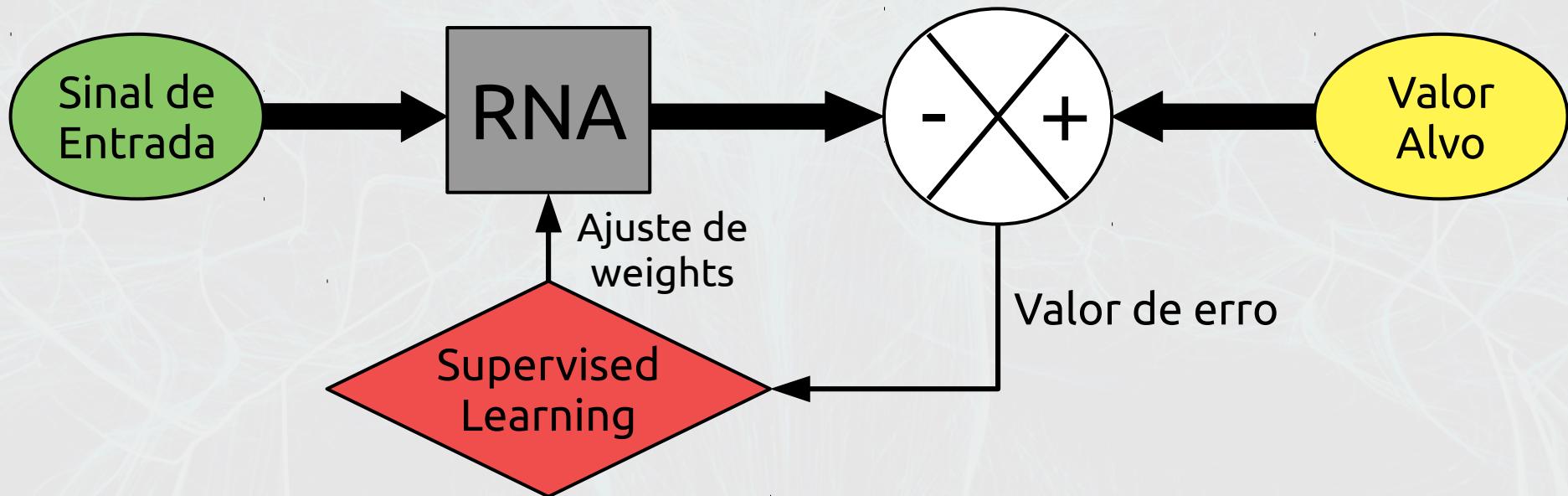
Funções de ativação lineares produzem decisões lineares, funções não-lineares nos permitem aproximar funções arbitrariamente complexas.

Treinando uma Rede Neural

- As redes neurais aprendem (ou são treinadas) processando exemplos, cada um dos quais contém uma "**entrada**" e um "**resultado**" conhecidos, formando associações ponderadas de probabilidade entre os dois, que são armazenadas na estrutura de dados da própria rede.
- O treinamento de uma rede neural a partir de um determinado exemplo é geralmente conduzido determinando a diferença entre a **saída processada** da rede (geralmente uma **previsão**) e uma **saída alvo**.
- Essa diferença é o **erro**.
- A rede então ajusta suas associações ponderadas (**weights**) de acordo com uma regra de aprendizado e usando esse valor de erro.
- Ajustes sucessivos farão com que a rede neural produza uma saída que é cada vez mais semelhante à saída alvo.

Treinando uma Rede Neural

- Depois de um número suficiente desses ajustes, o treinamento pode ser encerrado com base em certos critérios. Isso é conhecido como **supervised learning**.



Treinando uma Rede Neural

- À medida que começamos a pensar em casos de uso mais práticos para redes neurais, como reconhecimento ou classificação de imagens, vamos aproveitar o **supervised learning**, ou conjuntos de dados rotulados, para treinar o algoritmo.
- À medida que treinamos o modelo, queremos avaliar sua precisão (**accuracy**) usando uma função de custo (ou **loss**).
- Isso também é comumente referido como **mean squared error** (MSE).

Treinando uma Rede Neural

- Na equação (**MSE**) a seguir:
 - ◆ **i** representa o índice da amostra;
 - ◆ **y-hat** é o resultado previsto;
 - ◆ **y** é o valor real;
 - ◆ **m** é o número de amostras.

$$\text{Cost Function} = \text{MSE} = \frac{1}{2m} \sum_{i=1}^m (\hat{y} - y)^2$$

Treinando uma Rede Neural

- Em última análise, o objetivo é minimizar nossa função de custo para garantir a exatidão do ajuste para qualquer observação.
- À medida que o modelo ajusta seus **weights** e **bias**, ele usa a função de custo e o aprendizado de reforço para atingir o ponto de convergência, ou o mínimo local.
- O processo no qual o algoritmo ajusta seus weights é por meio de **gradient descent**, permitindo ao modelo determinar a direção a tomar para reduzir os erros (ou minimizar a função de custo).
- Com cada exemplo de treinamento, os parâmetros do modelo se ajustam para convergir gradualmente no mínimo.

Gradient Descent

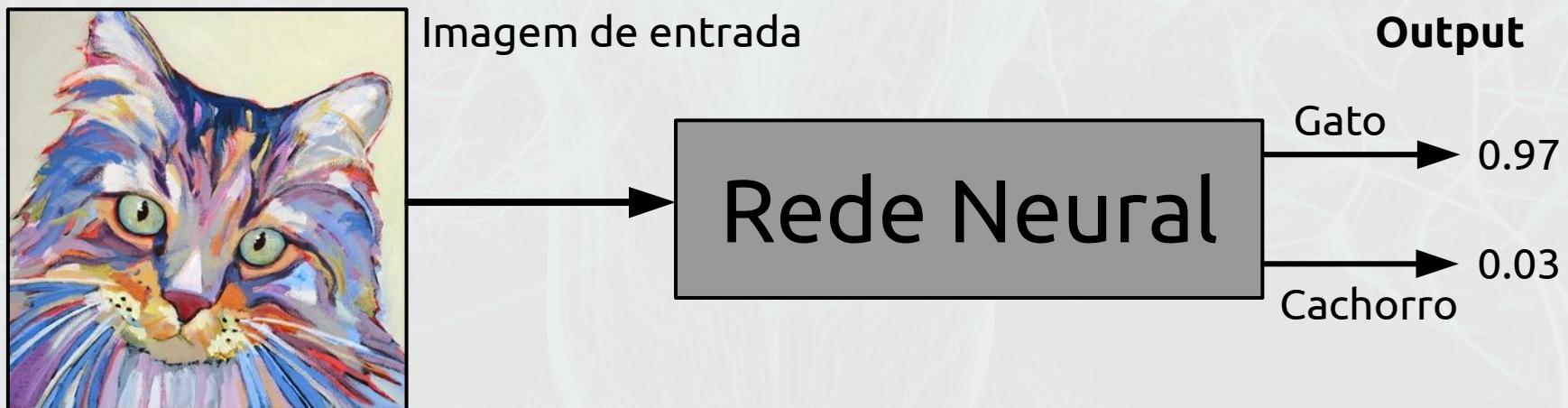


Treinando uma Rede Neural

- As **redes neurais artificiais** "aprendem" a executar tarefas considerando exemplos, geralmente sem serem programadas com regras específicas.
- Por exemplo, na classificação de imagens, eles podem aprender a **classificar imagens** que contêm **gatos** ou **cachorros**, analisando imagens de exemplo que foram marcadas manualmente como "gato" ou "cachorro" e usando os resultados para identificar gatos e cachorros em outras imagens.

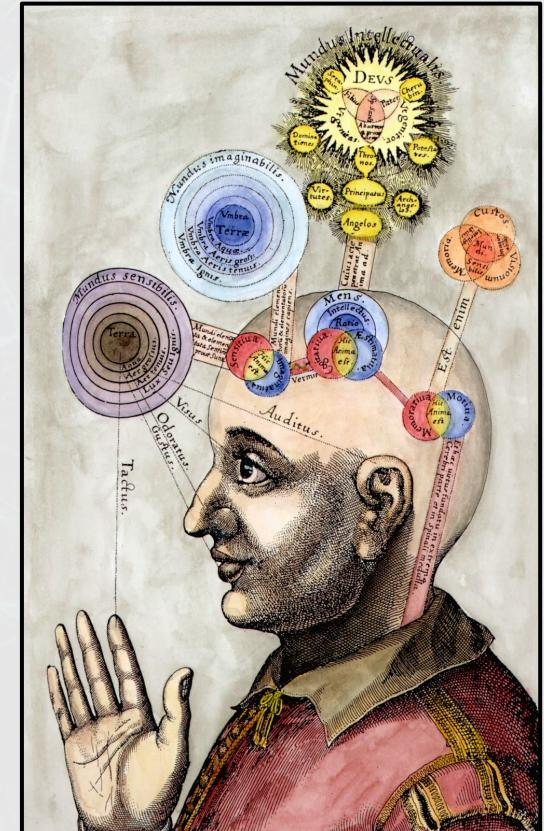
Treinando uma Rede Neural

- Elas fazem isso sem nenhum conhecimento prévio de gatos e cachorros, por exemplo, que eles têm pelos, rabos e bigodes. Em vez disso, elas geram automaticamente características de identificação a partir dos exemplos que processam.



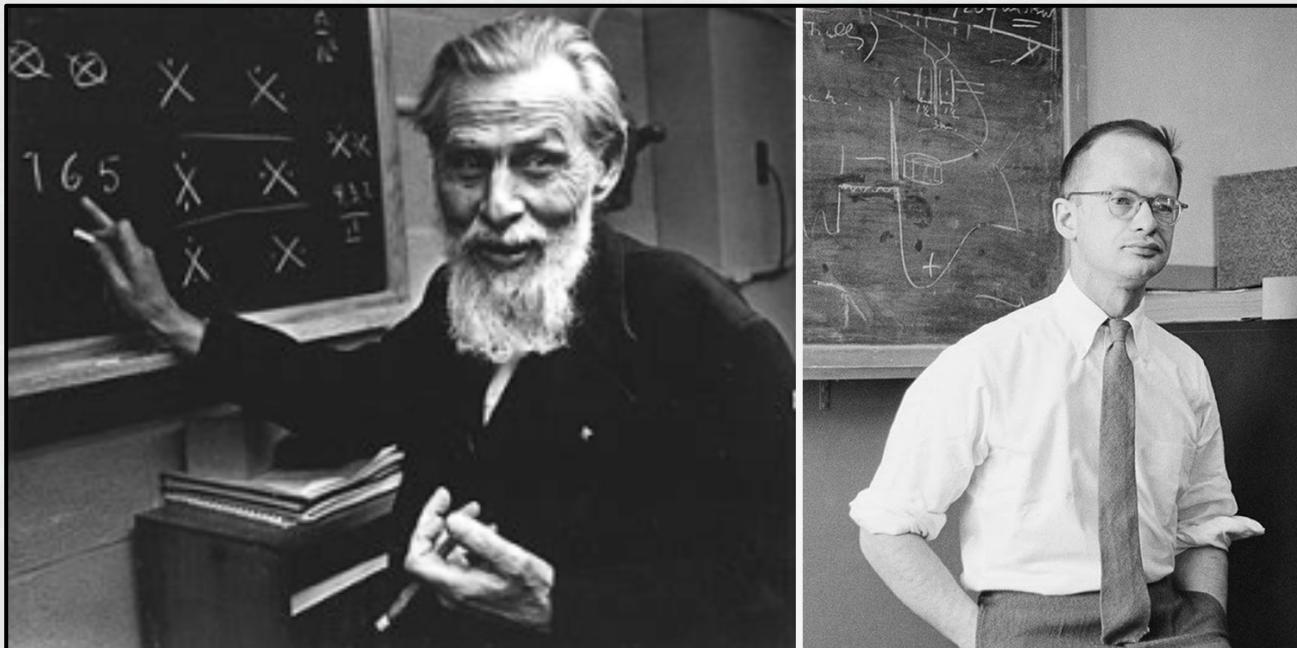
História das Redes Neurais

- A história das redes neurais é mais longa do que a maioria das pessoas pensa.
- Embora a ideia de "uma máquina que pense" possa ser rastreada até os gregos antigos, concentraremos nos principais eventos modernos que levaram à evolução do pensamento em torno das redes neurais.



História das Redes Neurais

- **Warren McCulloch e Walter Pitts** (1943) abriram o assunto criando um modelo computacional para redes neurais.



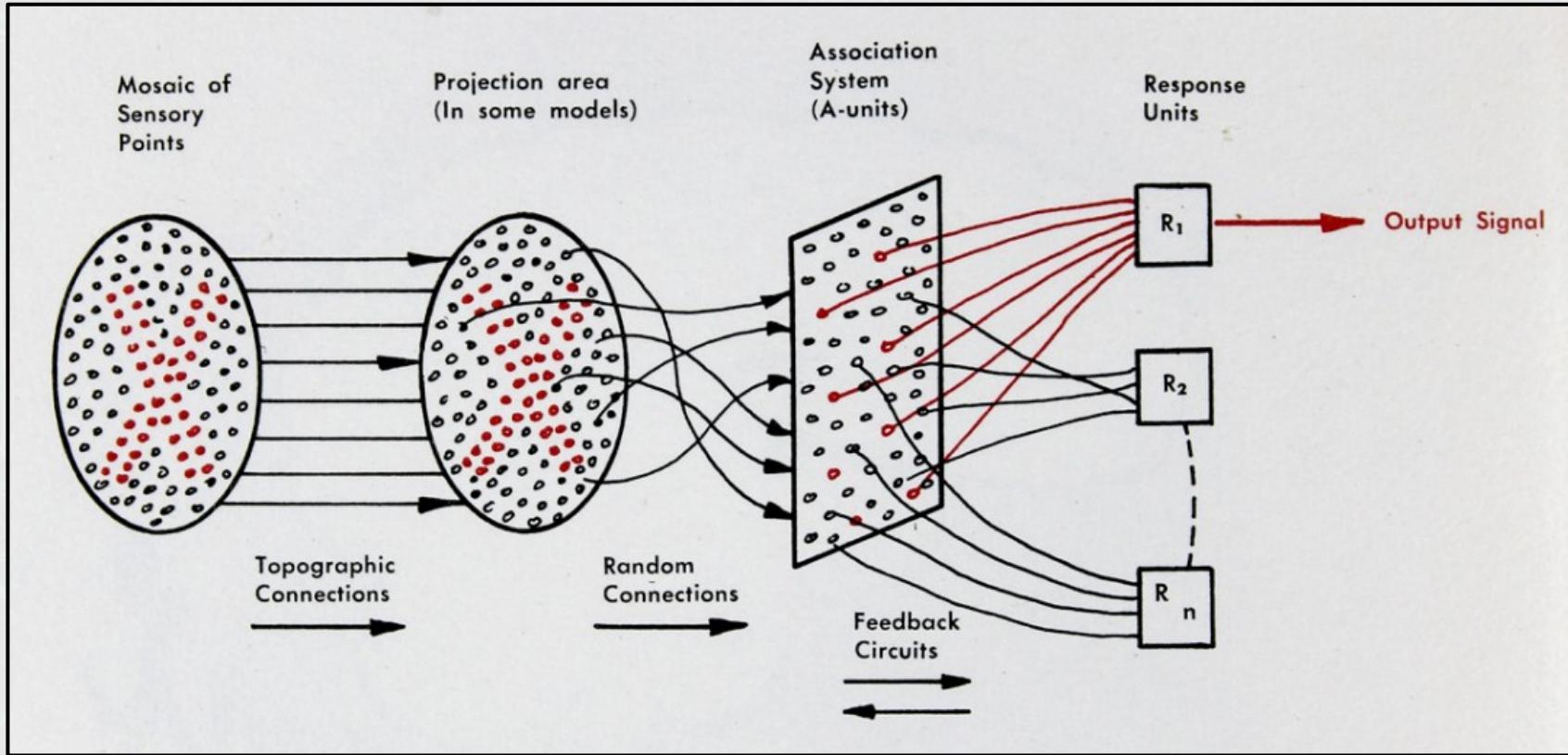
História das Redes Neurais

- **Warren McCulloch** e **Walter Pitts** publicaram o artigo “A logical calculus of the ideas immanent in nervous activity”, <https://web.csulb.edu/~cwallis/382/readings/482/mcculloch.logical.calculus.ideas.1943.pdf>
- Esta pesquisa buscou entender como o cérebro humano poderia produzir padrões complexos por meio de células cerebrais conectadas, ou neurônios.
- Uma das principais ideias que surgiram deste trabalho foi a comparação de neurônios com um limite binário para a lógica booleana (ou seja, 0/1 ou declarações verdadeiras/falsas).

História das Redes Neurais

- No final dos anos 1940, **D. O. Hebb** criou uma hipótese de aprendizado baseada no mecanismo de plasticidade neural que ficou conhecido como **Hebbian learning**.
- **Farley e Wesley A. Clark** (1954) usaram máquinas computacionais, então chamadas de "calculadoras", para simular uma rede Hebbian.
- Em 1958 Frank Rosenblatt é creditado com o desenvolvimento do **perceptron**, documentado em sua pesquisa "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain".
- Ele leva o trabalho de McCulloch e Pitt um passo adiante, introduzindo **weights** na equação. Utilizando um IBM 704, Rosenblatt conseguiu fazer com que um computador aprendesse a distinguir os cartões marcados à esquerda e os cartões marcados à direita.

Organização do Perceptron



História das Redes Neurais

- As primeiras redes funcionais com muitas camadas foram publicadas por **Ivakhnenco** e **Lapa** em 1965, como o **Group Method of Data Handling**.
- Os fundamentos da **continuous backpropagation** foram derivados no contexto da teoria de controle por **Kelley** em 1960 e por **Bryson** em 1961, usando princípios de **programação dinâmica**.
- Posteriormente, a pesquisa estagnou após **Minsky** e **Papert** (1969), que descobriram que os perceptrons básicos eram incapazes de processar o circuito **exclusive-or** e que os computadores não tinham energia suficiente para processar redes neurais úteis.
- Em 1970, **Seppo Linnainmaa** publicou o método geral para **automatic differentiation** (AD) de redes discretas conectadas de funções diferenciáveis aninhadas.

História das Redes Neurais

- Em 1973, **Dreyfus** usou **backpropagation** para adaptar os parâmetros dos controladores em proporção aos gradientes de erro.
- 1974: Enquanto vários pesquisadores contribuíram para a ideia de **backpropagation**, **Paul Werbos** foi a primeira pessoa nos Estados Unidos a observar sua aplicação em redes neurais em sua tese de doutorado.
- O algoritmo **backpropagation** de Werbos permitiu o treinamento prático de redes multicamadas. Em 1982, ele aplicou o método **AD** de **Linnainmaa** às redes neurais de uma maneira que se tornou amplamente usada.
- O desenvolvimento da **very-large-scale integration** (VLSI) de **metal-oxide-semiconductor** (MOS), na forma de tecnologia complementar MOS (**CMOS**), possibilitou o aumento da contagem de transistores MOS na eletrônica digital. Isso forneceu mais poder de processamento para o desenvolvimento de redes neurais artificiais práticas na década de 1980.

História das Redes Neurais

- Em 1986, **Rumelhart, Hinton e Williams** mostraram que o backpropagation aprendeu representações internas interessantes de palavras como **feature vectors** quando treinada para prever a próxima palavra em uma sequência.
- 1989: **Yann LeCun** publicou um artigo ilustrando como o uso de restrições no backpropagation e sua integração na arquitetura da rede neural podem ser usados para treinar algoritmos. Esta pesquisa disponibilizou com sucesso uma rede neural para reconhecer dígitos de código postal escritos à mão fornecidos pelo Serviço Postal dos EUA.

História das Redes Neurais

- Em 1992, o **max-pooling** foi introduzido para ajudar com a *least-shift invariance* e a tolerância à deformação para auxiliar no reconhecimento de objetos 3D. **Schmidhuber** adotou uma hierarquia multinível de redes (1992) pré-treinada um nível de cada vez por **unsupervised learning** e ajustado por backpropagation.
- **Geoffrey Hinton** (2006) propôs aprender uma representação de alto nível usando camadas sucessivas de variáveis latentes binárias ou de valor real com uma **máquina de Boltzmann restrita** para modelar cada camada.
- Em 2012, **Ng** e **Dean** criaram uma rede que aprendeu a reconhecer conceitos de nível superior, como gatos, apenas assistindo a imagens sem rótulos.

História das Redes Neurais

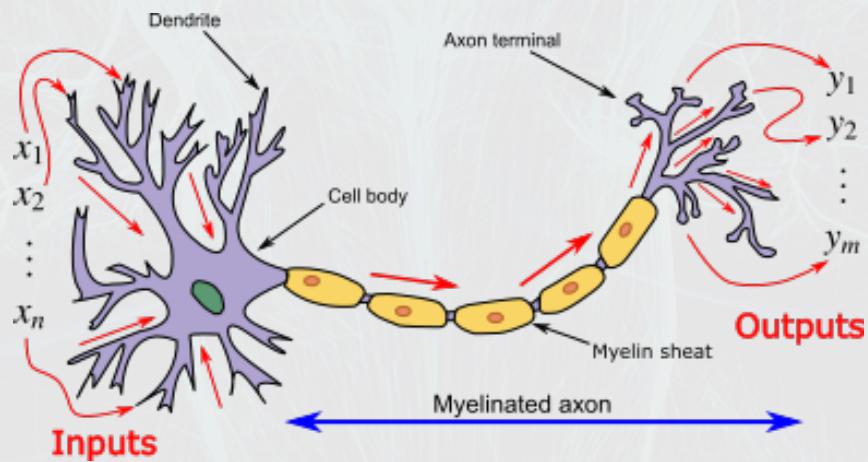
- O pré-treinamento *unsupervised* e o aumento do poder de computação das GPUs e da computação distribuída permitiram o uso de redes maiores, principalmente em problemas de imagem e reconhecimento visual, que ficaram conhecidos como "**deep learning**".
- **Ciresan** e colegas (2010) mostraram que, apesar do problema do *vanishing gradient*, as GPUs tornam backpropagation viável para redes neurais feedforward de muitas camadas.
- Entre 2009 e 2012, as RNAs começaram a ganhar prêmios em concursos de RNA, alcançando o desempenho de nível humano em várias tarefas, inicialmente em reconhecimento de padrões e machine learning.
- Por exemplo, a bi-direcional e multi-dimensional **long short-term memory** (LSTM) de **Graves** venceu três concursos de reconhecimento de caligrafia conectada em 2009 sem nenhum conhecimento prévio sobre os três idiomas a serem aprendidos.

Modelos

- As RNAs começaram como uma tentativa de explorar a arquitetura do cérebro humano para realizar tarefas nas quais os algoritmos convencionais tiveram pouco sucesso.
- Elas logo se reorientaram para melhorar os resultados empíricos, abandonando principalmente as tentativas de permanecer fiéis aos seus precursores biológicos.
- Os neurônios são conectados uns aos outros em vários padrões, para permitir que a saída de alguns neurônios se torne a entrada de outros. A rede neural forma um **grafo direcionado e weighted**.

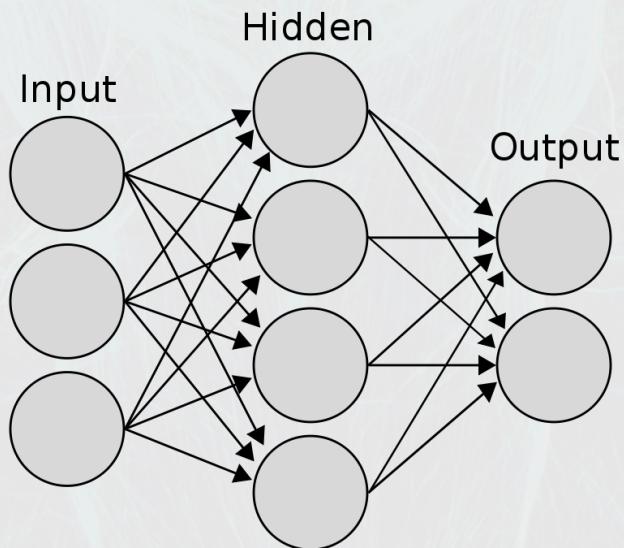
Modelos

- Uma rede neural artificial consiste em uma coleção de neurônios simulados. Cada neurônio é um nó que está conectado a outros nós por meio de links que correspondem às conexões biológicas axônio-sinapse-dendrito. Cada link tem um **weight**, que determina a força da influência de um nó sobre outro.



Conexões e Weights

- A rede consiste em conexões, cada conexão fornecendo a saída de um neurônio como uma entrada para outro neurônio. Cada conexão recebe um **weight** que representa sua importância relativa. Um determinado neurônio pode ter várias conexões de entrada e saída.



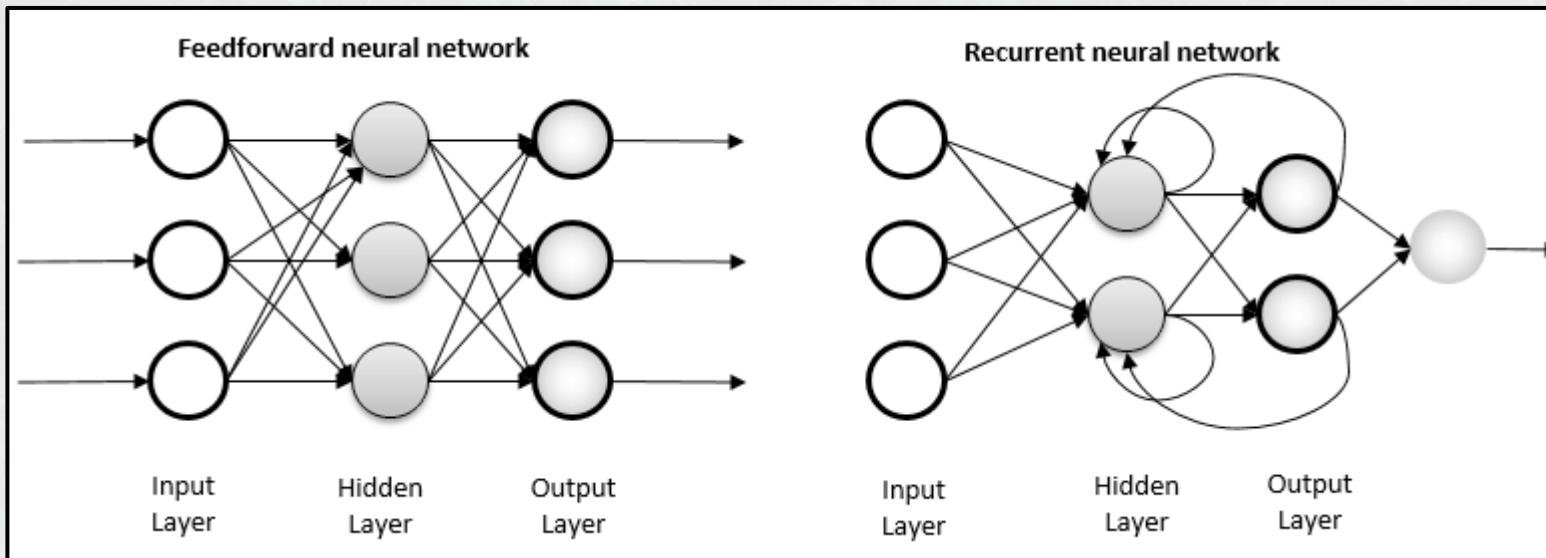
Organização das Redes Neurais

- Os neurônios são normalmente organizados em várias camadas, especialmente no **deep learning**. Os neurônios de uma camada se conectam apenas aos neurônios das camadas imediatamente anteriores e posteriores.
- A camada que recebe dados externos é a **camada de entrada**.
- A camada que produz o resultado final é a **camada de saída**.
- Entre elas estão zero ou mais **camadas ocultas**.
- Redes de camada única e sem camada também são usadas.

Organização das Redes Neurais

- Entre duas camadas, vários padrões de conexão são possíveis.
- Eles podem ser totalmente conectados, com cada neurônio em uma camada conectando-se a todos os neurônios na próxima camada.
- Eles podem ser **pooling**, onde um grupo de neurônios em uma camada se conecta a um único neurônio na próxima camada, reduzindo assim o número de neurônios nessa camada.
- Neurônios com apenas essas conexões formam um grafo acíclico direcionado e são conhecidos como **feedforward networks**.
- Alternativamente, as redes que permitem conexões entre neurônios na mesma camada ou em camadas anteriores são conhecidas como **recurrent networks**.

Organização das Redes Neurais

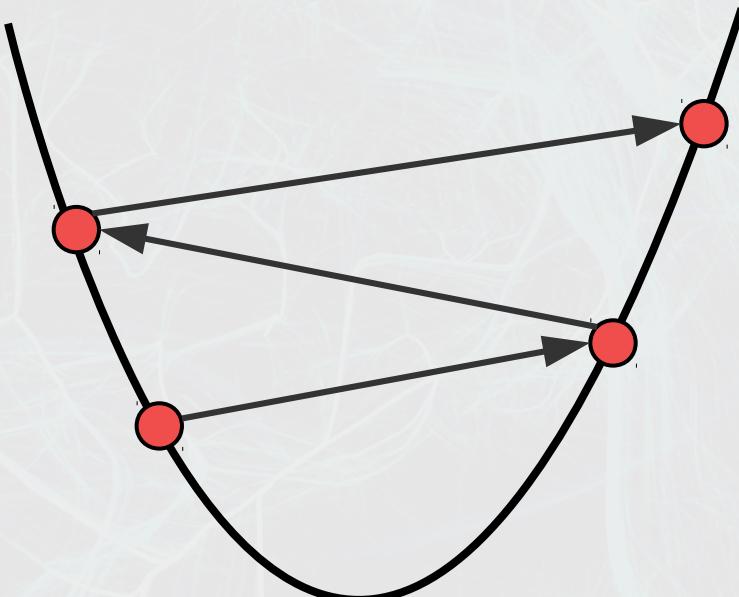


Hiperparâmetros

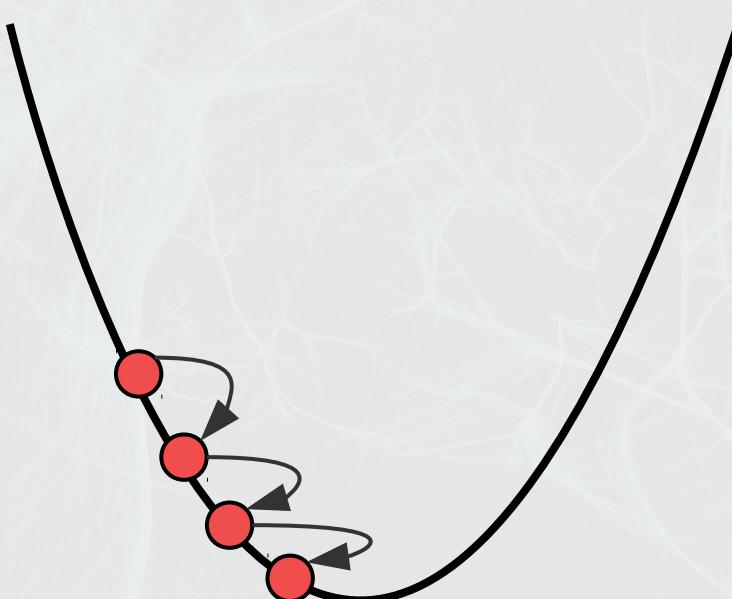
- Um hiperparâmetro é um parâmetro constante cujo valor é definido antes do início do processo de aprendizagem.
- Os valores dos parâmetros são derivados por meio do aprendizado.
- Exemplos de hiperparâmetros incluem **taxa de aprendizagem**, **o número de camadas ocultas** e **tamanho do batch**.
- Os valores de alguns hiperparâmetros podem ser dependentes dos valores de outros hiperparâmetros. Por exemplo, o tamanho de algumas camadas pode depender do número total de camadas.

Taxa de Aprendizado

Taxa de aprendizado alta



Taxa de aprendizado baixa



Aprendizado

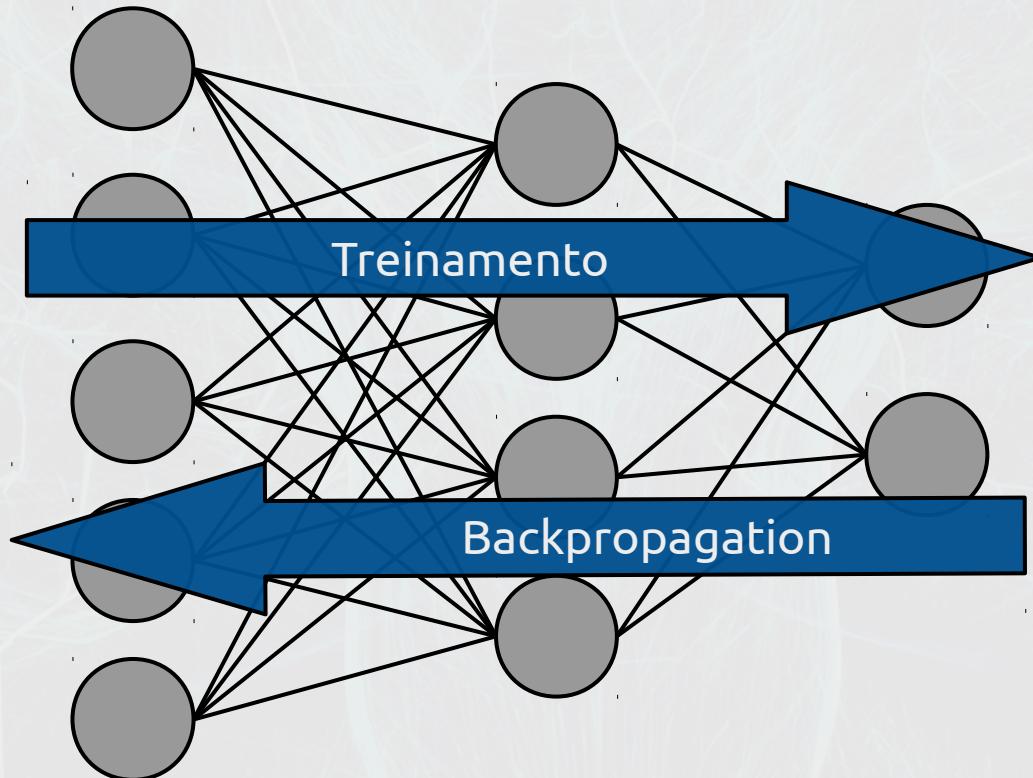
- A aprendizagem é a adaptação da rede para lidar melhor com uma tarefa, considerando **observações de amostra**.
- O aprendizado envolve o ajuste dos **weights** (e thresholds opcionais) da rede para melhorar a **accuracy** do resultado.
- Isso é feito minimizando os erros observados.
- O aprendizado é completo quando o exame de observações adicionais não reduz de forma útil a **taxa de erro**.
- Mesmo após o aprendizado, a taxa de erro normalmente não chega a 0. Se, após o aprendizado, a taxa de erro for muito alta, a rede normalmente deve ser reprojetada.

Aprendizado

- Na prática, isso é feito definindo uma **função de custo** ou **loss** que é avaliada periodicamente durante o aprendizado. Enquanto sua **saída** continua diminuindo, o aprendizado continua.
- O custo é frequentemente definido como uma estatística cujo valor só pode ser aproximado.
- As **saídas** são, na verdade, números, portanto, quando o erro é baixo, a diferença entre a **saída** (quase certamente um gato) e a **resposta correta** (gato) é pequena. A aprendizagem tenta reduzir o total das diferenças entre as observações.
- A maioria dos modelos de aprendizagem pode ser vista como uma aplicação direta da **teoria de otimização** e **estimativa estatística**.

Aprendizado

Conjunto
de
Treinamento



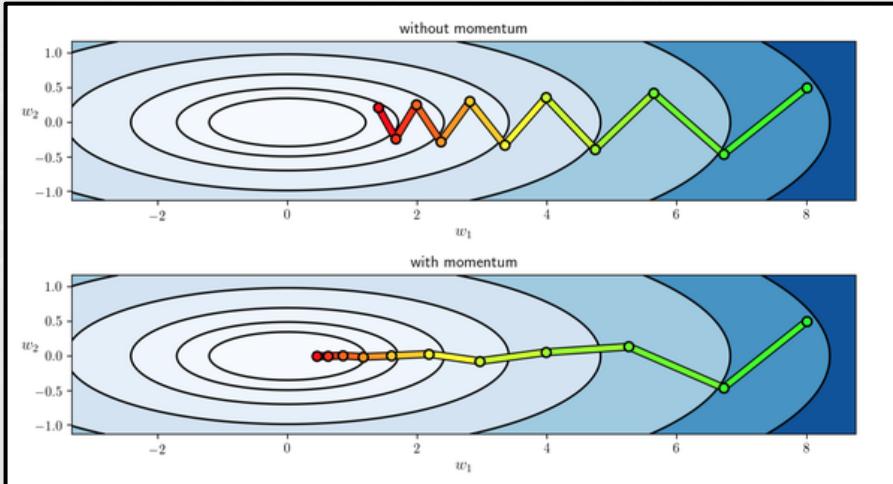
Comparação
com o conjunto
de rótulos (labels)

Taxa de Aprendizado

- A taxa de aprendizagem define o tamanho das etapas corretivas que o modelo executa para ajustar os erros em cada observação.
- Uma alta taxa de aprendizado encurta o tempo de treinamento, mas com menor accuracy final, enquanto uma taxa de aprendizado mais baixa leva mais tempo, mas com potencial para maior accuracy.
- Otimizações como **Quickprop** visam principalmente acelerar a minimização de erros, enquanto outras melhorias tentam principalmente aumentar a confiabilidade.
- Para evitar oscilações dentro da rede, como weights de conexão alternados, e para melhorar a taxa de convergência, os refinamentos usam uma **taxa de aprendizagem adaptativa** que aumenta ou diminui conforme apropriado.

Taxa de Aprendizado

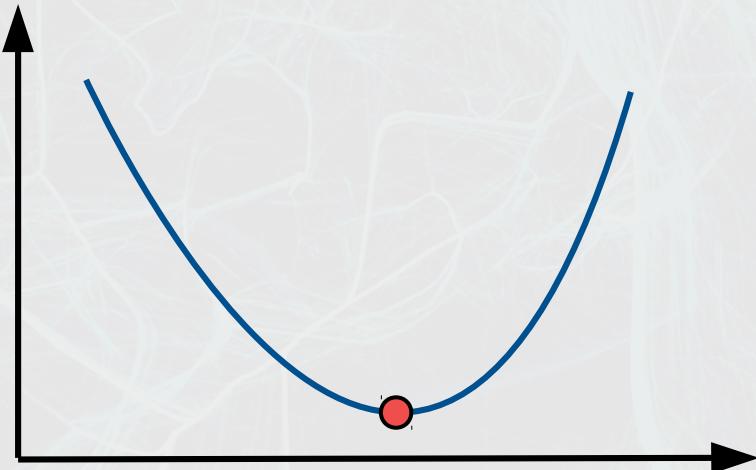
- O conceito de momentum permite que o equilíbrio entre o gradiente e a alteração anterior seja ponderado de forma que o ajuste do **weight** dependa em algum grau da alteração anterior.
- Um momentum próximo a 0 enfatiza o gradiente, enquanto um valor próximo a 1 enfatiza a última alteração.



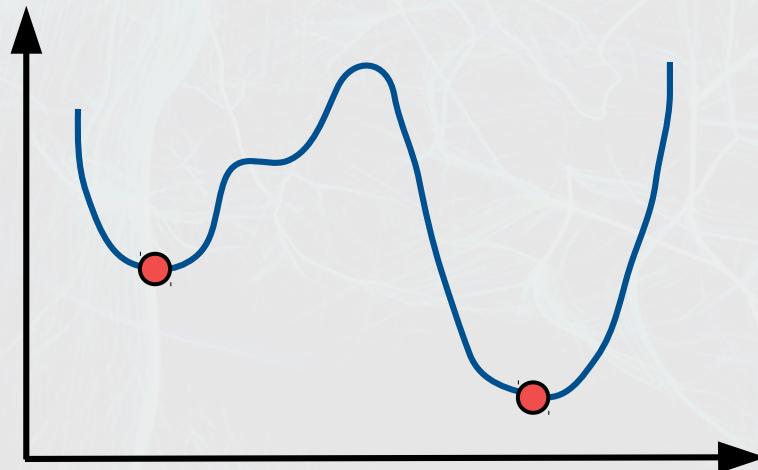
Função de Custo

- Embora seja possível definir uma função de custo **ad hoc**, frequentemente a escolha é determinada pelas propriedades desejáveis da função (como convexidade) ou porque surge do modelo.

Convexa



Não-Convexa

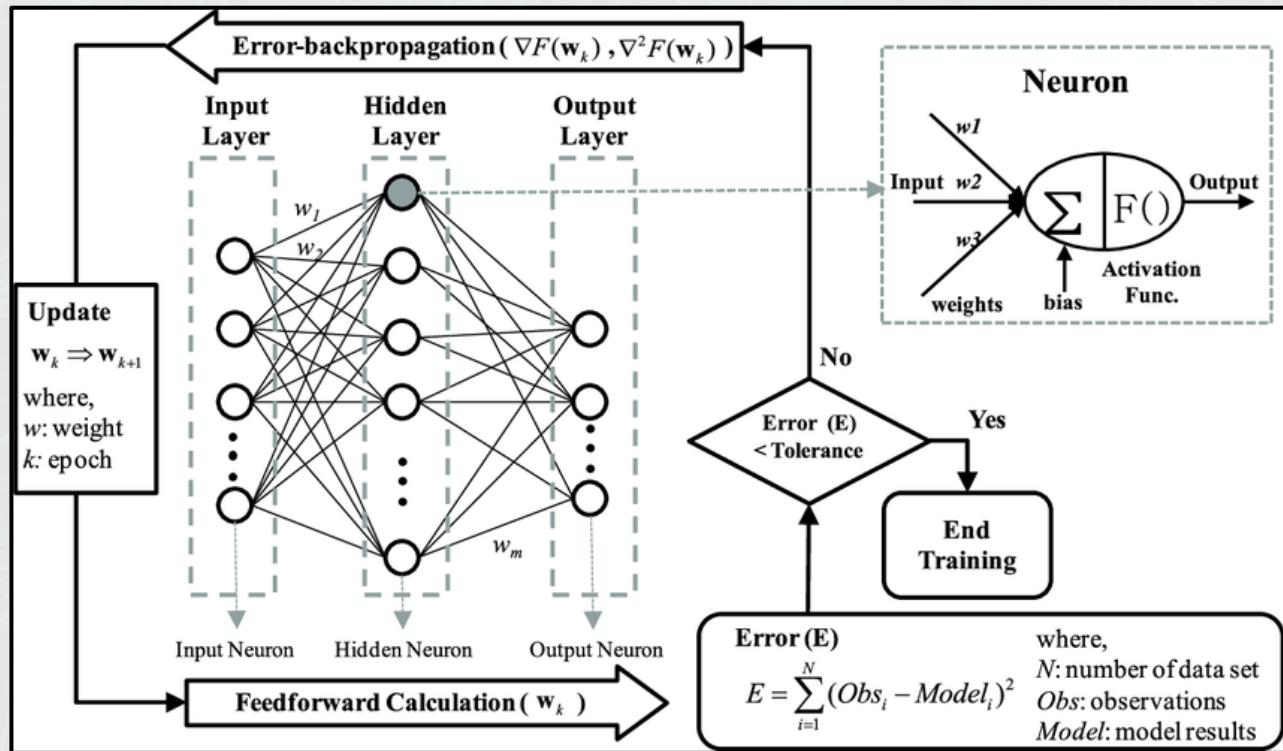


Backpropagation

- **Backpropagation** é um método usado para ajustar os weights da conexão para compensar cada erro encontrado durante o aprendizado.
- A quantidade de erros é efetivamente dividida entre as conexões.
- Tecnicamente, backpropagation calcula o gradiente (a derivada) da função de custo associada a um determinado estado em relação aos weights.
- As atualizações de weight podem ser feitas por meio de **stochastic gradient descent** ou outros métodos, como **Extreme Learning Machines**.

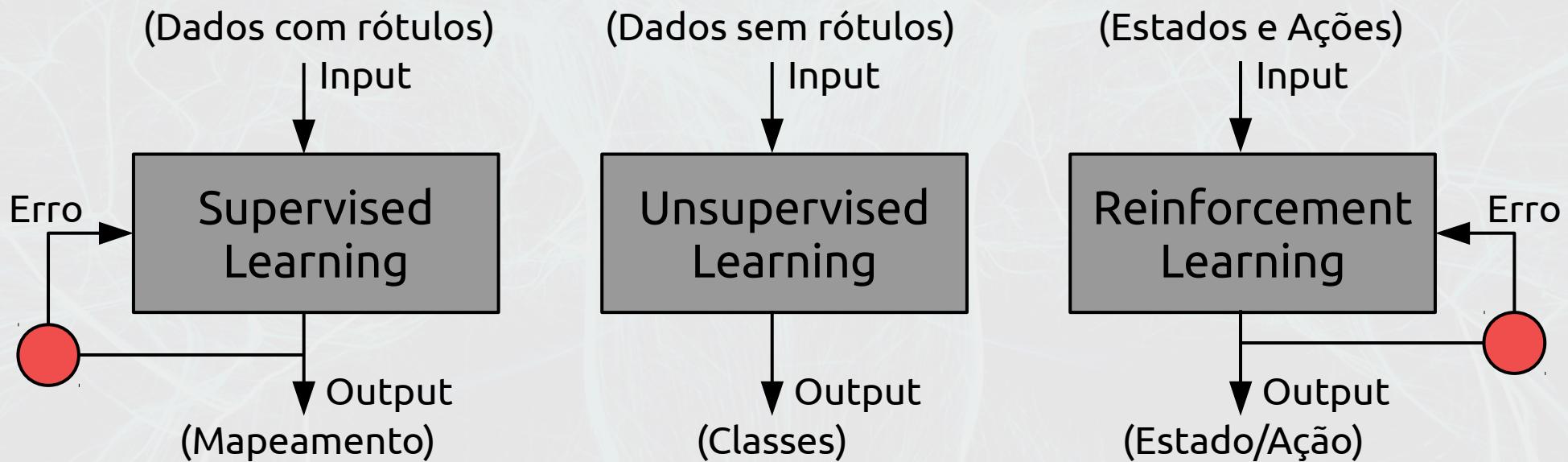
Backpropagation

- Diagrama esquemático do algoritmo de treinamento backpropagation.



Paradigmas de Aprendizado

- Os três principais paradigmas de aprendizagem são **supervised learning**, **unsupervised learning** e **reinforcement learning**. Cada um deles corresponde a uma tarefa de aprendizagem particular.



Supervised Learning

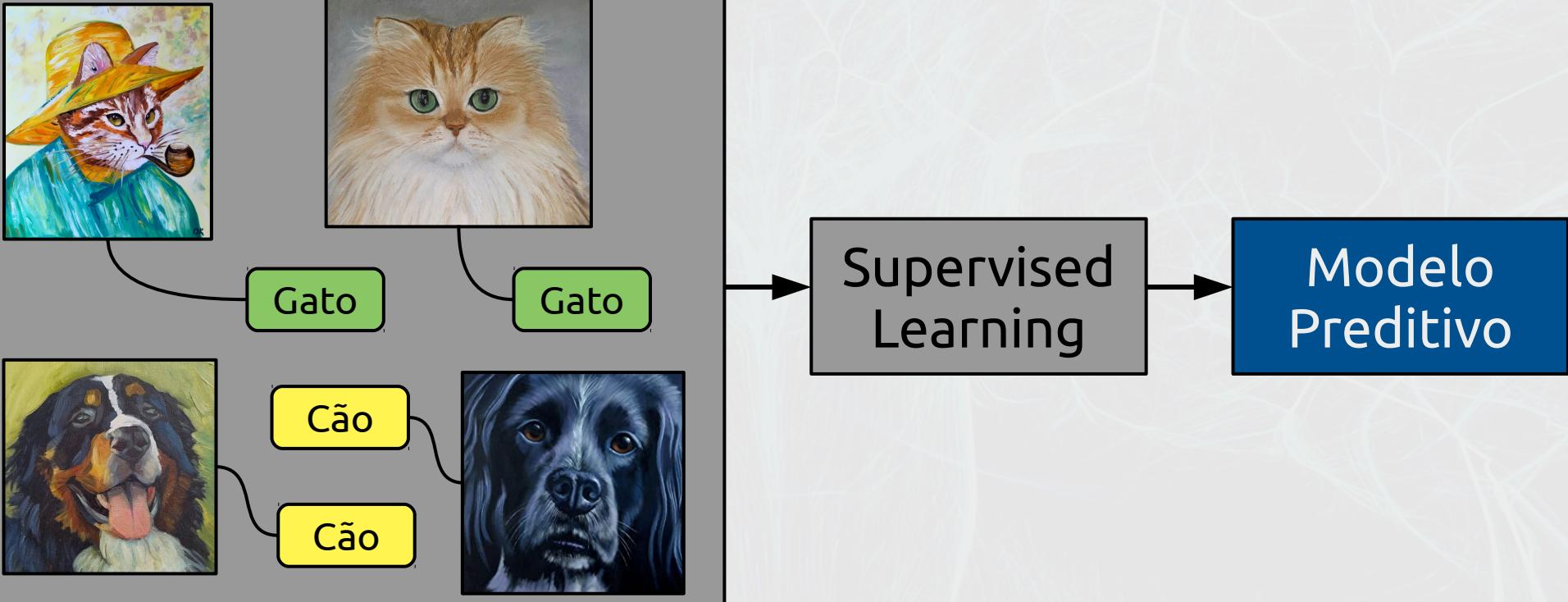
- Supervised Learning usa um **conjunto de entradas** emparelhadas e **saídas desejadas**.
- A tarefa de aprendizagem é produzir a saída desejada para cada entrada.
- Nesse caso, a **função de custo** está relacionada à eliminação de deduções incorretas.
- Uma função de custo comumente usada é a **mean-squared error**, que tenta minimizar o erro quadrático médio entre a saída da rede e a saída desejada.

Supervised Learning

- As tarefas adequadas para o supervised learning são o reconhecimento de padrões (também conhecido como **classificação**) e **regressão** (também conhecido como aproximação de função).
- Supervised Learning também é aplicável a dados sequenciais (por exemplo, para escrita à mão, reconhecimento de fala e gestos).
- Ele pode ser pensado como aprender com um "professor", na forma de uma função que fornece feedback contínuo sobre a qualidade das soluções obtidas até o momento.

Supervised Learning

Dados Rotulados



Supervised Learning



Modelo
Preditivo

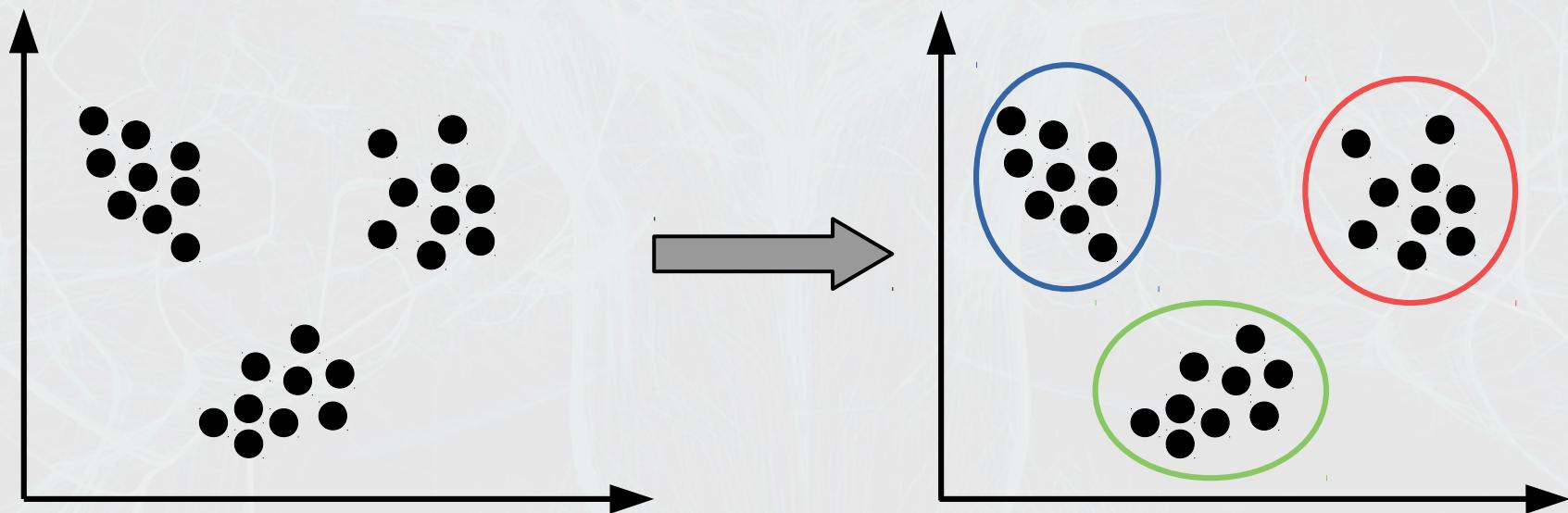
Cão

Unsupervised Learning

- Em unsupervised learning, os dados de entrada são fornecidos junto com a função de custo, alguma função dos dados x e a saída da rede.
- A função de custo depende da tarefa (o domínio do modelo) e de quaisquer suposições **a priori** (as propriedades implícitas do modelo, seus parâmetros e as variáveis observadas).
- Como um exemplo trivial, considere o modelo $f(x) = a$ onde a é uma constante e o custo $C = E[(x - f(x))^2]$.
- Minimizar este custo produz um valor de a que é igual à média dos dados. A função de custo pode ser muito mais complicada. Sua forma depende da aplicação: por exemplo, na compressão pode estar relacionada à informação mútua entre x e $f(x)$, enquanto na modelagem estatística, pode estar relacionado à probabilidade posterior do modelo dados os dados (observe que em ambos os exemplos essas quantidades seriam maximizadas em vez de minimizadas).

Unsupervised Learning

- As tarefas que se enquadram no paradigma unsupervised learning são, em geral, problemas de estimativa; as aplicações incluem clustering, estimativa de distribuições estatísticas, compressão e filtragem.



Reinforcement Learning

- Em aplicações como jogos de videogame, um **ator** realiza uma série de **ações**, recebendo uma resposta geralmente imprevisível do ambiente após cada uma.
- O objetivo é ganhar o jogo, ou seja, gerar as respostas mais positivas (menor custo).
- No reinforcement learning, o objetivo é ponderar a rede (elaborar uma **política**) para realizar ações que minimizem o custo de longo prazo (cumulativo esperado).
- A cada momento o agente executa uma ação e o ambiente gera uma observação e um custo instantâneo, de acordo com algumas regras (geralmente desconhecidas).
- As regras e o custo a longo prazo geralmente só podem ser estimados.

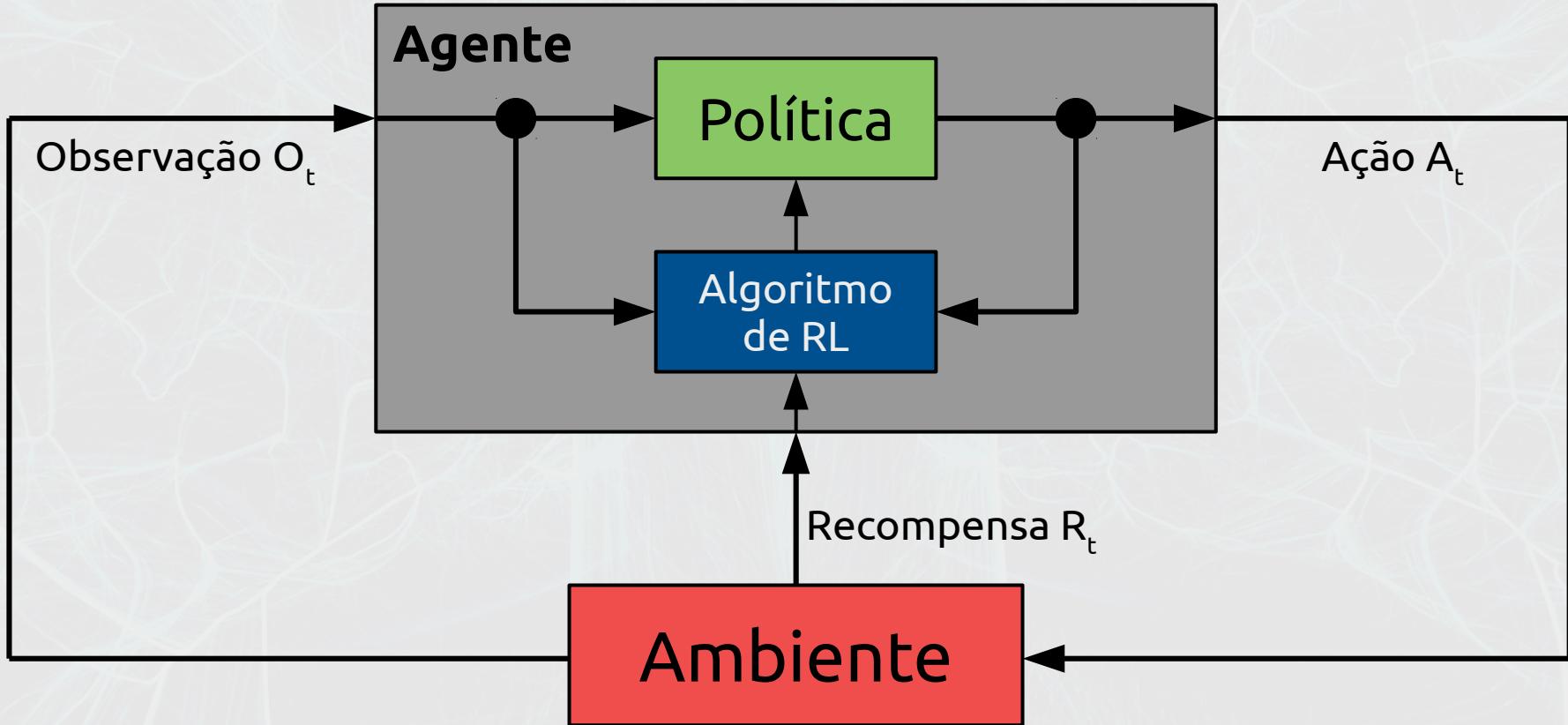
Reinforcement Learning

- Em qualquer conjuntura, o agente decide se vai explorar novas ações para descobrir seus custos ou explorar o aprendizado anterior para prosseguir mais rapidamente.
- Formalmente, o ambiente é modelado como um **Markov decision process** (MDP) com estados $s_1, \dots, s_n \in S$ e ações $a_1, \dots, a_m \in A$.
- Como as transições de estado não são conhecidas, as distribuições de probabilidade são usadas em seu lugar: a distribuição de custo instantânea $P(c_t | s_t)$, a distribuição de observação $P(x_t | s_t)$ e a distribuição de transição $P(s_{t+1} | s_t, a_t)$ enquanto uma política é definida como a distribuição condicional sobre as ações dadas as observações.
- Juntos, os dois definem uma **Markov chain** (MC). O objetivo é descobrir o MC de menor custo.

Reinforcement Learning

- RNAs servem como o componente de aprendizagem em tais aplicações.
- A programação dinâmica juntamente com RNAs (dando programação neurodinâmica) tem sido aplicada a problemas como aqueles envolvidos em roteamento de veículos, videogames, gerenciamento de recursos naturais e medicina, por causa da capacidade das RNAs de mitigar perdas de *accuracy*, mesmo ao reduzir a *discretization grid density* numericamente aproximando a solução de problemas de controle.
- As tarefas que se enquadram no paradigma de reinforcement learning são problemas de controle, jogos e outras tarefas sequenciais de tomada de decisão.

Reinforcement Learning



Uso das Redes Neurais

- O uso de redes neurais artificiais requer uma compreensão de suas características:
 - ◆ **Escolha do modelo:** Depende da representação dos dados e da aplicação. Modelos excessivamente complexos retardam o aprendizado.
 - ◆ **Algoritmo de aprendizagem:** Existem inúmeras vantagens e desvantagens entre os algoritmos de aprendizagem. Quase qualquer algoritmo funcionará bem com os hiperparâmetros corretos para treinamento em um determinado conjunto de dados. No entanto, selecionar e ajustar um algoritmo para treinar em dados não vistos requer experimentação significativa.
 - ◆ **Robustez:** Se o modelo, a função de custo e o algoritmo de aprendizagem forem selecionados de forma adequada, a RNA resultante pode se tornar robusta.

Uso das Redes Neurais

- As capacidades da RNA se enquadram nas seguintes categorias amplas:
 - ◆ Aproximação de função ou análise de regressão, incluindo previsão de séries temporais, aproximação de aptidão e modelagem.
 - ◆ Classificação, incluindo reconhecimento de padrão e sequência, detecção de novidades e tomada de decisão sequencial.
 - ◆ Processamento de dados, incluindo filtragem, clustering e compactação.
 - ◆ Robótica, incluindo manipuladores diretores e próteses.

Aplicações das Redes Neurais

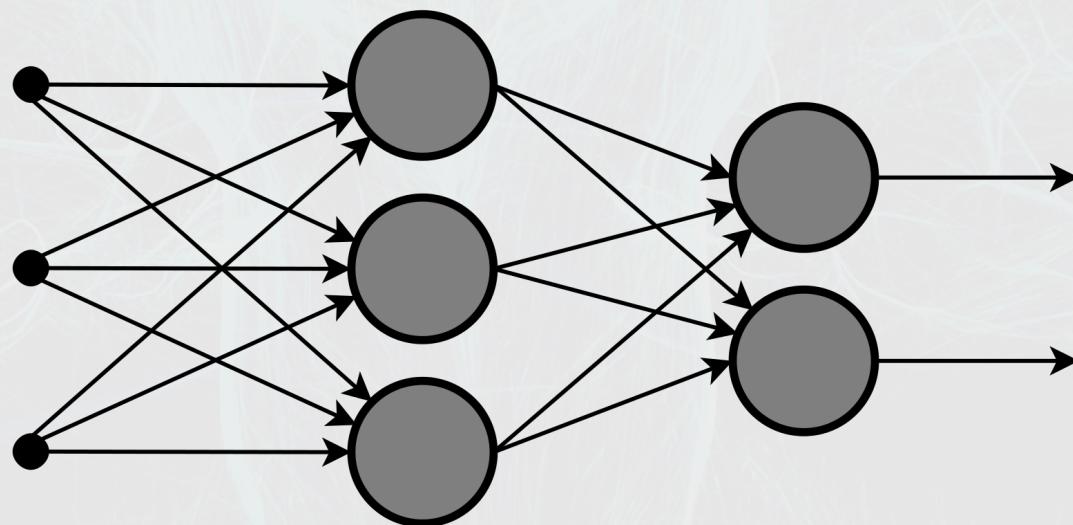
- Por causa de sua capacidade de reproduzir e modelar processos não lineares, as redes neurais artificiais encontraram aplicações em muitas disciplinas.
- As áreas de aplicação incluem identificação e controle de sistema (controle de veículo, previsão de trajetória, controle de processo, gerenciamento de recursos naturais), química quântica, jogos em geral, reconhecimento de padrões (sistemas de radar, identificação de rosto, classificação de sinal, reconstrução 3D, reconhecimento de objeto e muito mais), reconhecimento de sequência (gesto, fala, reconhecimento de texto manuscrito e impresso), diagnóstico médico, finanças (por exemplo, sistemas de negociação automatizados), mineração de dados, visualização, tradução automática, filtragem de rede social e filtragem de spam de e-mail. As RNAs têm sido usadas para diagnosticar vários tipos de câncer e para distinguir linhas de células cancerígenas altamente invasivas de linhas menos invasivas usando apenas informações sobre a forma da célula.

Aplicações das Redes Neurais

- ANNs têm sido propostas como uma ferramenta para resolver equações diferenciais parciais em física e simular as propriedades de sistemas quânticos abertos de muitos corpos.
- Em pesquisas sobre o cérebro, as RNAs estudaram o comportamento de curto prazo de neurônios individuais, a dinâmica dos circuitos neurais surgem de interações entre neurônios individuais e como o comportamento pode surgir de módulos neurais abstratos que representam subsistemas completos.
- Os estudos consideraram a plasticidade de longo e curto prazo dos sistemas neurais e sua relação com o aprendizado e a memória do neurônio individual para o nível do sistema.

Poder Computacional

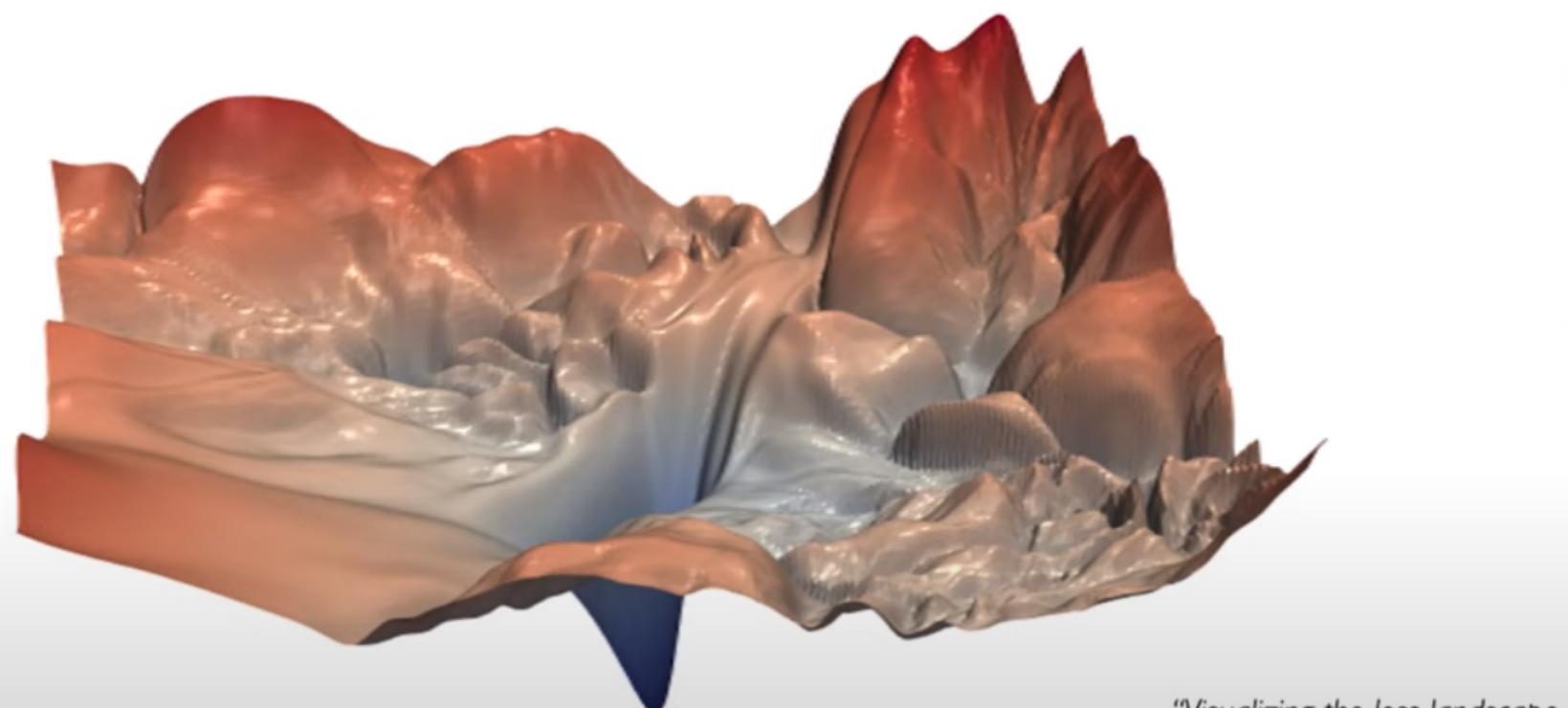
- O **perceptron multicamadas** é um aproximador de função universal, conforme comprovado pelo **teorema da aproximação universal**.
- Entretanto, a prova não é construtiva quanto ao número de neurônios necessários, a topologia da rede, os weights e os parâmetros de aprendizagem.



Convergência

- Os modelos podem não convergir de forma consistente para uma única solução, em primeiro lugar porque podem existir mínimos locais, dependendo da função de custo e do modelo. Em segundo lugar, o método de otimização usado pode não garantir a convergência quando começa longe de qualquer mínimo local. Em terceiro lugar, para dados ou parâmetros suficientemente grandes, alguns métodos tornam-se impraticáveis.

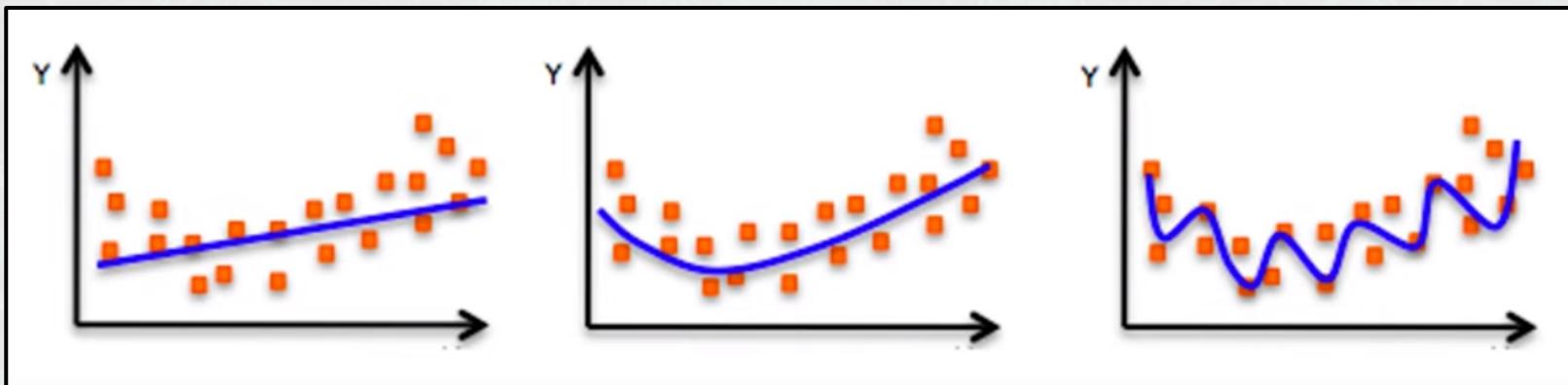
Convergência



"Visualizing the loss landscape
of neural nets". Dec 2017.

Generalização e Estatísticas

- Aplicações cujo objetivo é criar um sistema que generaliza bem para exemplos nunca vistos, enfrentam a possibilidade de over-training. Isso surge em sistemas complicados ou superespecificados quando a capacidade da rede excede significativamente os parâmetros livres necessários.



Underfitting

Fit Ideal

Overfitting

Generalização e Estatísticas

- Duas abordagens tratam do over-training. A primeira é usar **cross-validation** e técnicas semelhantes para verificar a presença de over-training e selecionar hiperparâmetros para minimizar o erro de generalização.



Generalização e Estatísticas

- A segunda é usar alguma forma de **regularização**, como por exemplo **early stopping**.



Generalização e Estatísticas

- Redes neurais supervisionadas que usam uma função de custo **mean squared error** (MSE) podem usar métodos estatísticos formais para determinar a confiança do modelo treinado.
- O MSE em um conjunto de validação pode ser usado como uma estimativa para a variação. Este valor pode então ser usado para calcular o intervalo de confiança da saída da rede, assumindo uma distribuição normal.
- Uma análise de confiança feita dessa forma é estatisticamente válida, desde que a distribuição de probabilidade de saída permaneça a mesma e a rede não seja modificada.

Generalização e Estatísticas

- Ao atribuir uma função de ativação **softmax**, uma generalização da função logística, na camada de saída da rede neural para variáveis de destino categóricas, as saídas podem ser interpretadas como probabilidades posteriores. Isso é útil na classificação, pois dá uma medida de certeza nas classificações.

Input pixels, x



Feedforward output, y_i

	cat	dog	horse
cat	5	4	2
dog	4	2	8
horse	4	4	1

Forward propagation

Softmax output, $S(y_i)$

	cat	dog	horse
cat	0.71	0.26	0.04
dog	0.02	0.00	0.98
horse	0.49	0.49	0.02

Shape: (3, 32, 32)

Shape: (3,)

Shape: (3,)

Deep Learning

- O **deep learning** faz parte de uma família mais ampla de métodos de **machine learning** baseados em redes neurais artificiais com **representation learning**.

ARTIFICIAL INTELLIGENCE

Qualquer técnica que possibilite um computador imitar o comportamento humano



MACHINE LEARNING

Habilidade de aprender sem ser explicitamente programado



DEEP LEARNING

Extrair padrões de dados usando redes neurais artificiais

3 1 3 4 7 2
1 7 4 2 3 5

Deep Learning

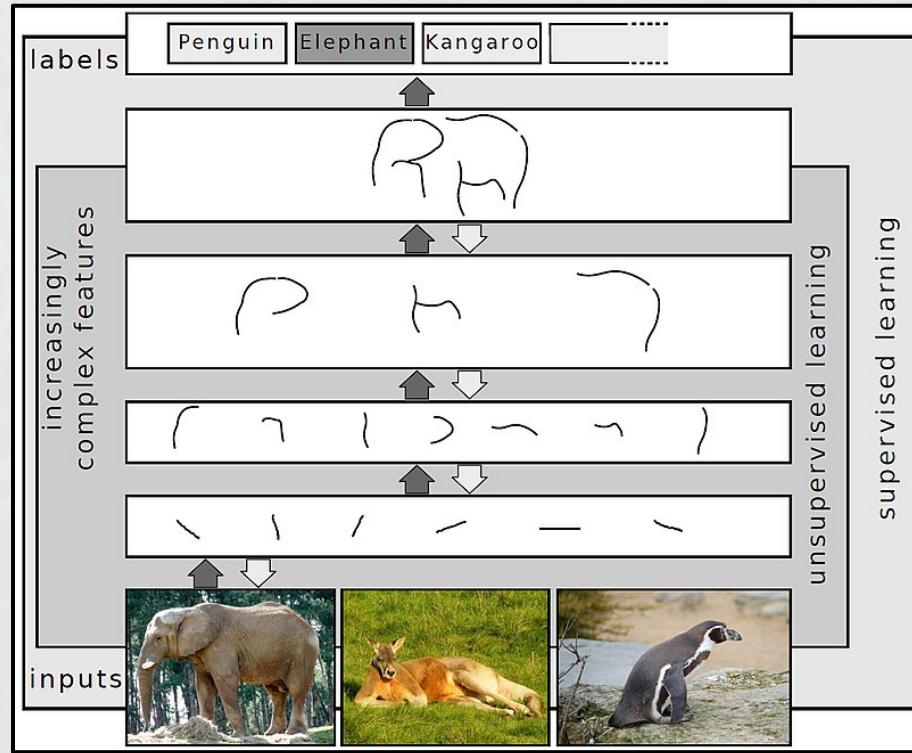
- O adjetivo "deep" em deep learning se refere ao uso de várias camadas (**layers**) na rede.
- Trabalhos anteriores mostraram que um perceptron linear não pode ser um classificador universal, mas que uma rede com uma função de ativação não-polinomial com uma camada oculta de largura ilimitada pode.
- No deep learning, as camadas também podem ser heterogêneas e se desviarem amplamente dos modelos conexionistas biologicamente inspirados, por uma questão de eficiência, treinabilidade e compreensibilidade.

Deep Learning

- Então, deep learning é uma classe de algoritmos de machine learning que usa várias camadas para extrair progressivamente recursos de nível superior da entrada bruta.
- Por exemplo, no **processamento de imagem**, as camadas inferiores podem identificar as bordas, enquanto as camadas superiores podem identificar os conceitos relevantes para um ser humano, como dígitos, letras ou rostos.

Deep Learning

- Representando Imagens em Múltiplas Camadas de Abstração no Deep Learning:

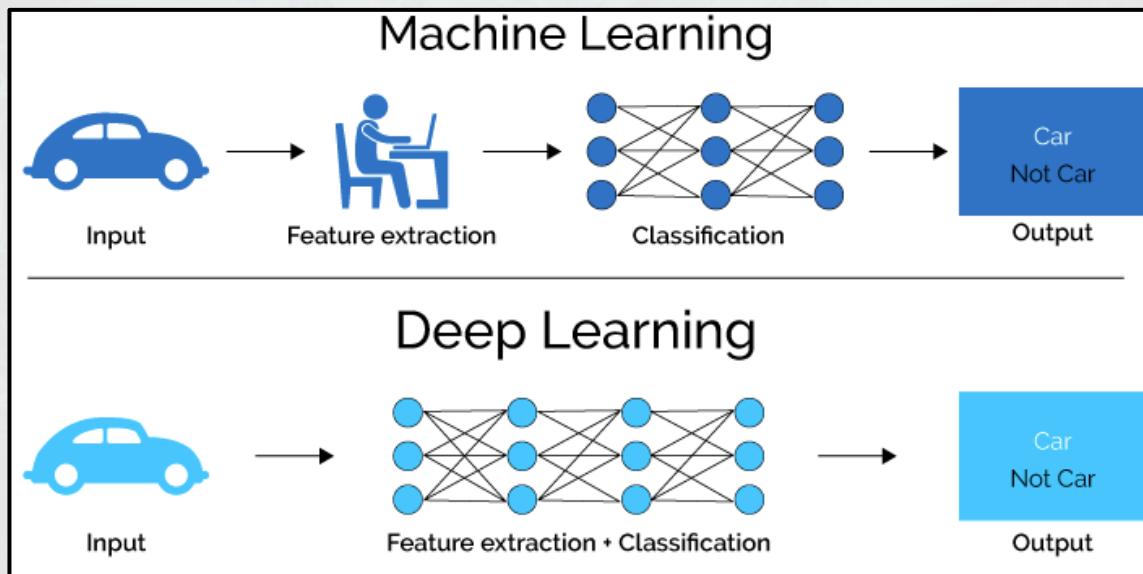


Deep Learning

- No deep learning, cada nível aprende a transformar seus dados de entrada em uma representação um pouco mais abstrata e composta.
- Em uma aplicação de reconhecimento de imagem, a entrada bruta pode ser uma matriz de pixels; a primeira camada representacional pode abstrair os pixels e codificar as bordas; a segunda camada pode compor e codificar arranjos de bordas; a terceira camada pode codificar um nariz e olhos; e a quarta camada pode reconhecer que a imagem contém um rosto.
- É importante ressaltar que um processo de deep learning pode aprender quais **features** devem ser colocados de maneira ideal em cada nível. Isso não elimina completamente a necessidade de ajuste manual; por exemplo, diferentes números de camadas e tamanhos de camadas podem fornecer diferentes graus de abstração.

Deep Learning

- Para tarefas de **supervised learning**, os métodos de deep learning eliminam a **feature engineering**, traduzindo os dados em representações intermediárias compactas semelhantes aos **principal components** e derivam estruturas em camadas que removem a redundância na representação.



Deep Neural Networks

- Uma **deep neural network** (DNN) é uma **rede neural artificial** (RNA) com várias camadas entre as camadas de entrada e saída.
- Existem diferentes tipos de redes neurais, mas sempre consistem nos mesmos componentes: **neurônios**, **sinapses**, **weights**, **biases** e **funções**.
- As DNNs podem modelar relacionamentos não lineares complexos. As arquiteturas DNN geram modelos compostionais em que o objeto é expresso como uma composição em camadas de primitivos.
- As camadas extras permitem a composição de **features** de camadas inferiores, potencialmente modelando dados complexos com menos unidades do que uma **shallow network** de desempenho semelhante.

Deep Neural Networks

- **Deep architectures** incluem muitas variantes de algumas abordagens básicas.
- DNNs são normalmente **feedforward networks** nas quais os dados fluem da camada de entrada para a camada de saída **sem loopback**.
- **Recurrent neural networks** (RNNs), nas quais os dados podem fluir em qualquer direção, são usadas para aplicações como modelagem de linguagem. **Long short-term memory** é particularmente eficaz para esse uso.
- **Convolutional deep neural networks** (CNNs) são usadas na visão computacional. As CNNs também foram aplicadas à modelagem acústica para **automatic speech recognition** (ASR).

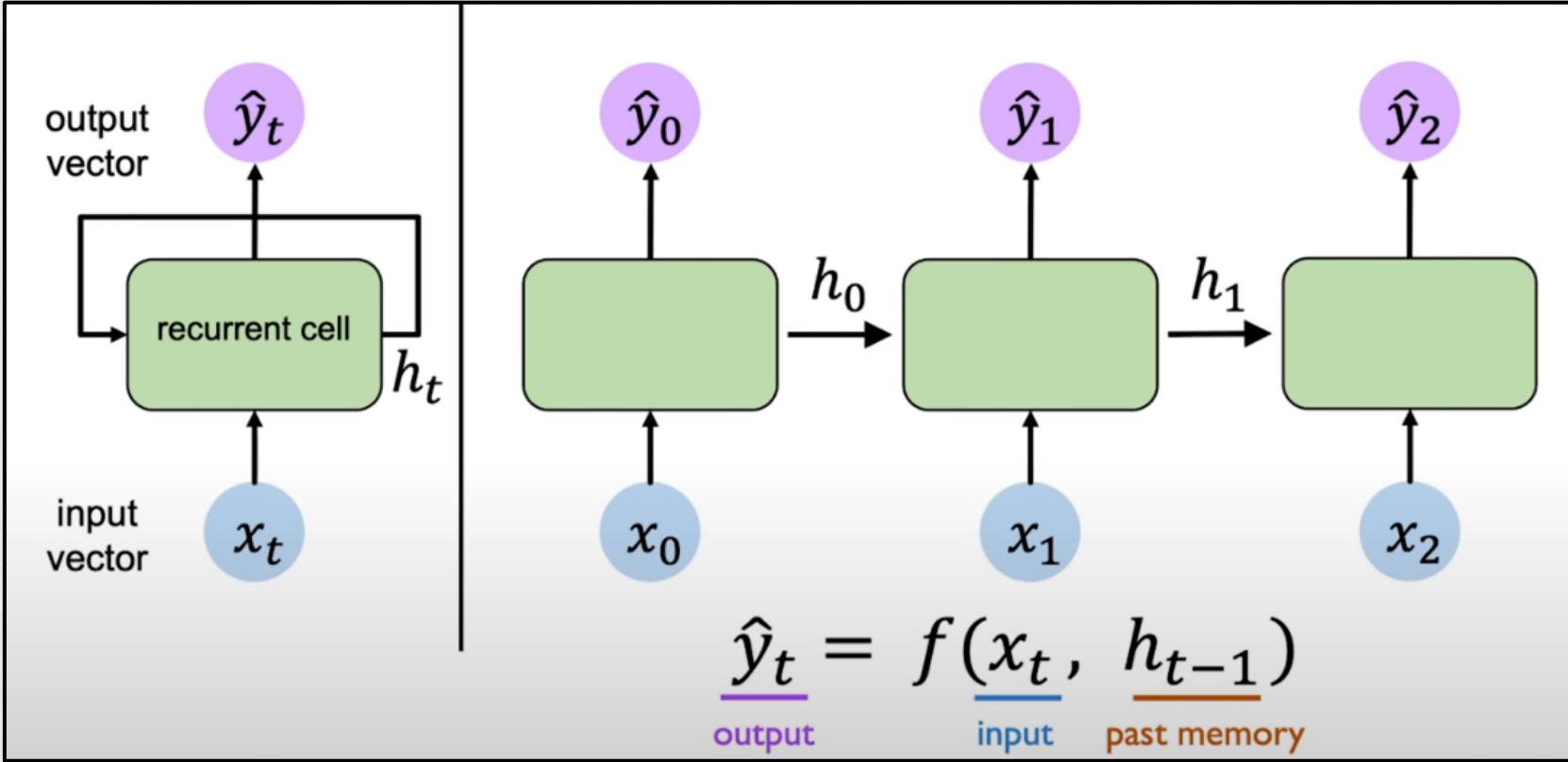
Recurrent Neural Network

- Uma **recurrent neural network** (RNN) é uma classe de redes neurais artificiais onde as conexões entre os nós formam um grafo direcionado ao longo de uma **sequência temporal**.
- Isso permite que ela exiba um comportamento dinâmico temporal. Derivado de redes neurais feedforward, as RNNs podem usar seu estado interno (memória) para processar sequências de entradas de comprimento variável.
- Suas aplicações incluem: previsão de séries temporais, síntese de fala, reconhecimento de fala, composição de música, reconhecimento de ação humana, aprendizagem de gramática, reconhecimento de caligrafia, previsão da localização subcelular de proteínas, etc.

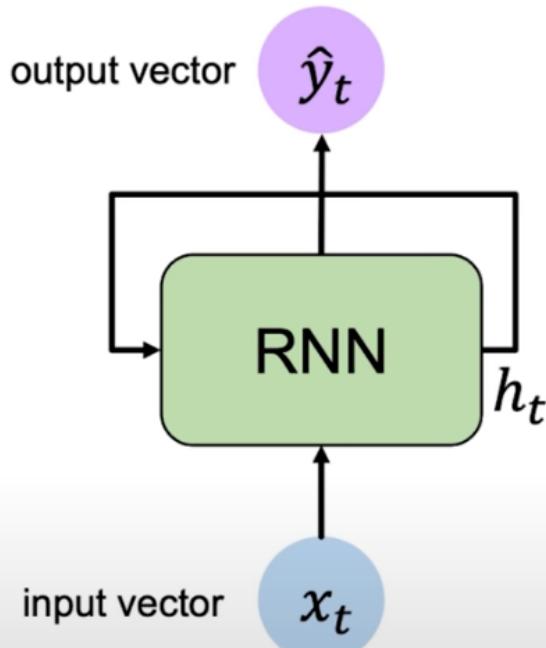
Recurrent Neural Network

- RNNs vêm em muitas variantes, **fully recurrent neural networks** (FRNN) conectam as saídas de todos os neurônios às entradas de todos os neurônios.
- A ilustração à seguir pode ser enganosa para muitos porque as topologias práticas de rede neural são frequentemente organizadas em "camadas" e o desenho dá essa aparência. No entanto, o que parecem ser camadas são, na verdade, etapas diferentes no tempo da mesma fully recurrent neural network. O item mais à esquerda na ilustração mostra as conexões recorrentes como um arco (recorrente). Ele é "desdobrado" no tempo para produzir a aparência de camadas.

Recurrent Neural Network



Recurrent Neural Network



Apply a **recurrence relation** at every time step to process a sequence:

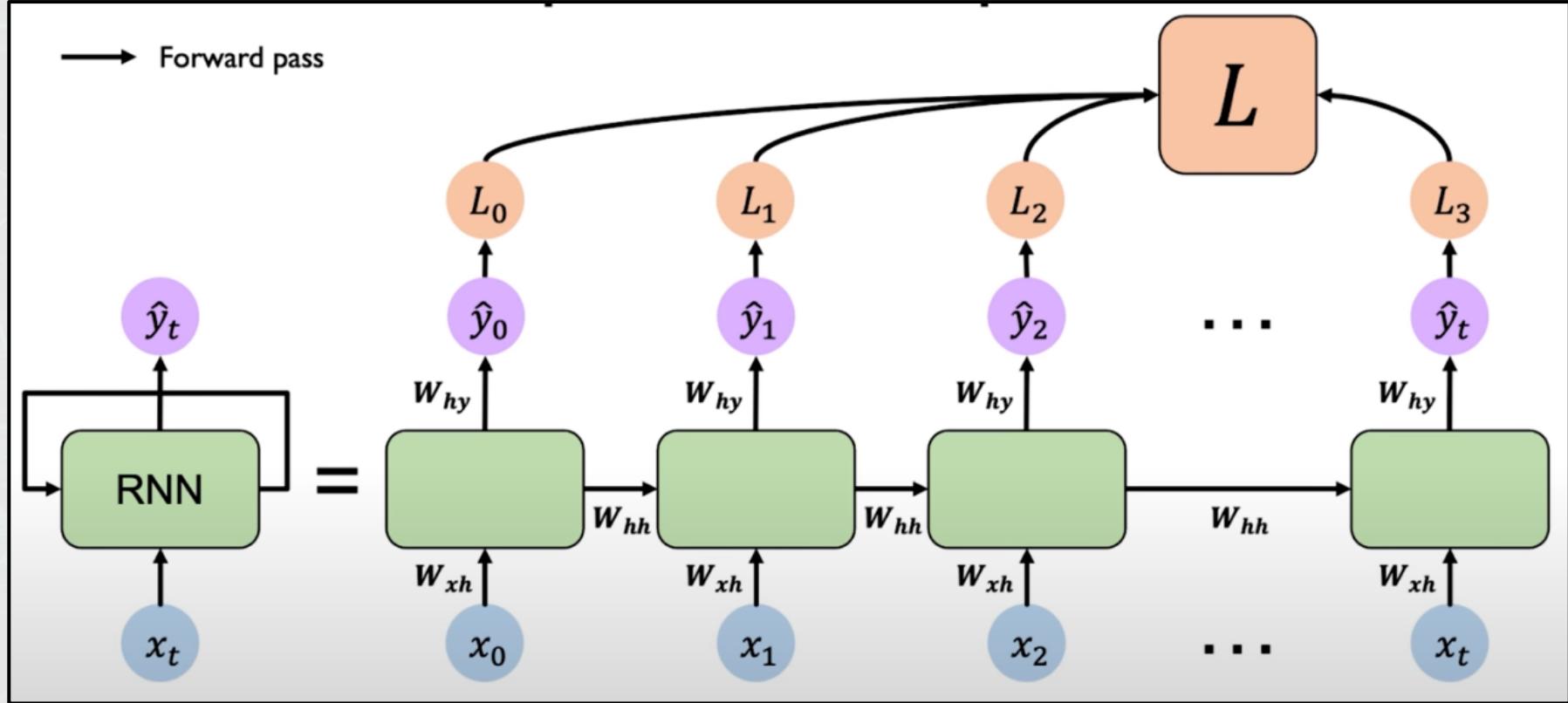
$$h_t = f_W(x_t, h_{t-1})$$

cell state function
 with weights
 input old state
 W

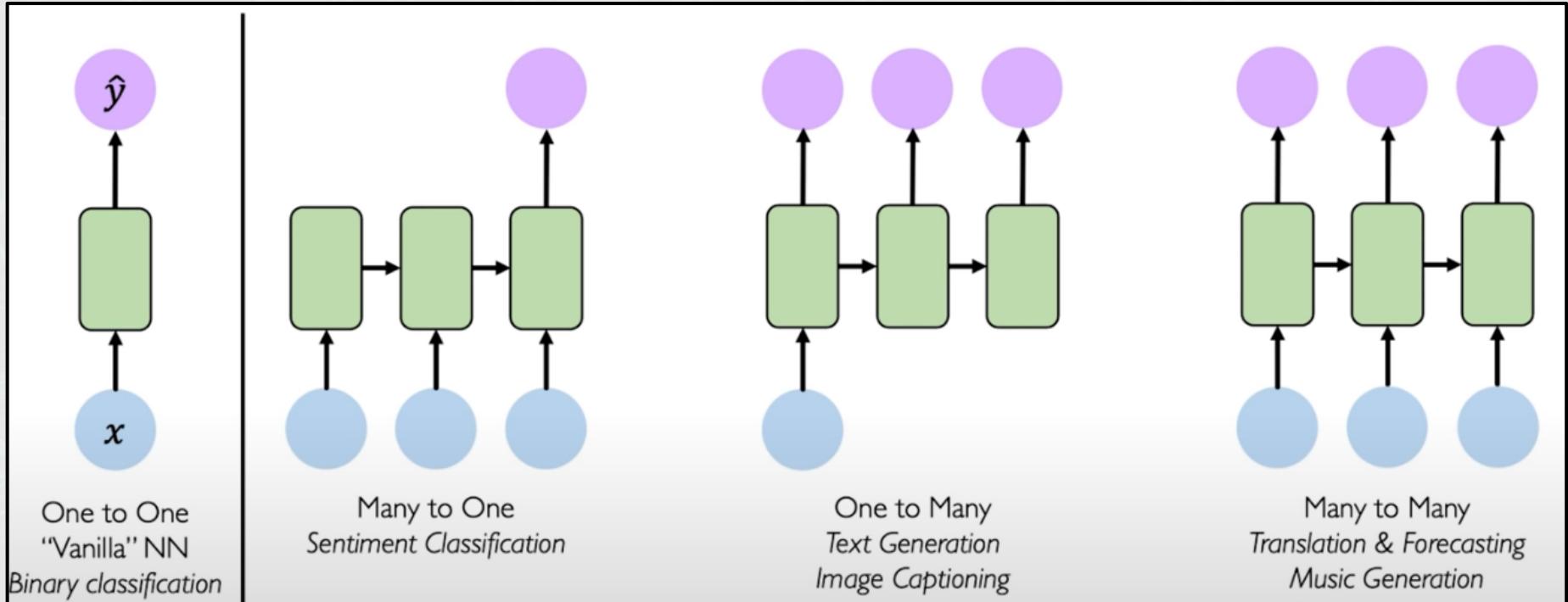
Note: the same function and set of parameters are used at every time step

RNNs have a **state**, h_t , that is updated **at each time step** as a sequence is processed

Recurrent Neural Network



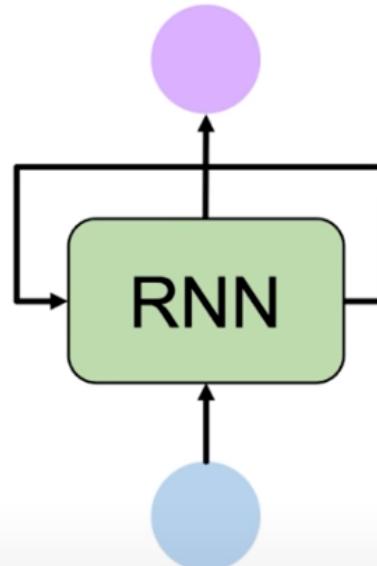
Recurrent Neural Network



Recurrent Neural Network

To model sequences, we need to:

1. Handle **variable-length** sequences
2. Track **long-term** dependencies
3. Maintain information about **order**
4. **Share parameters** across the sequence



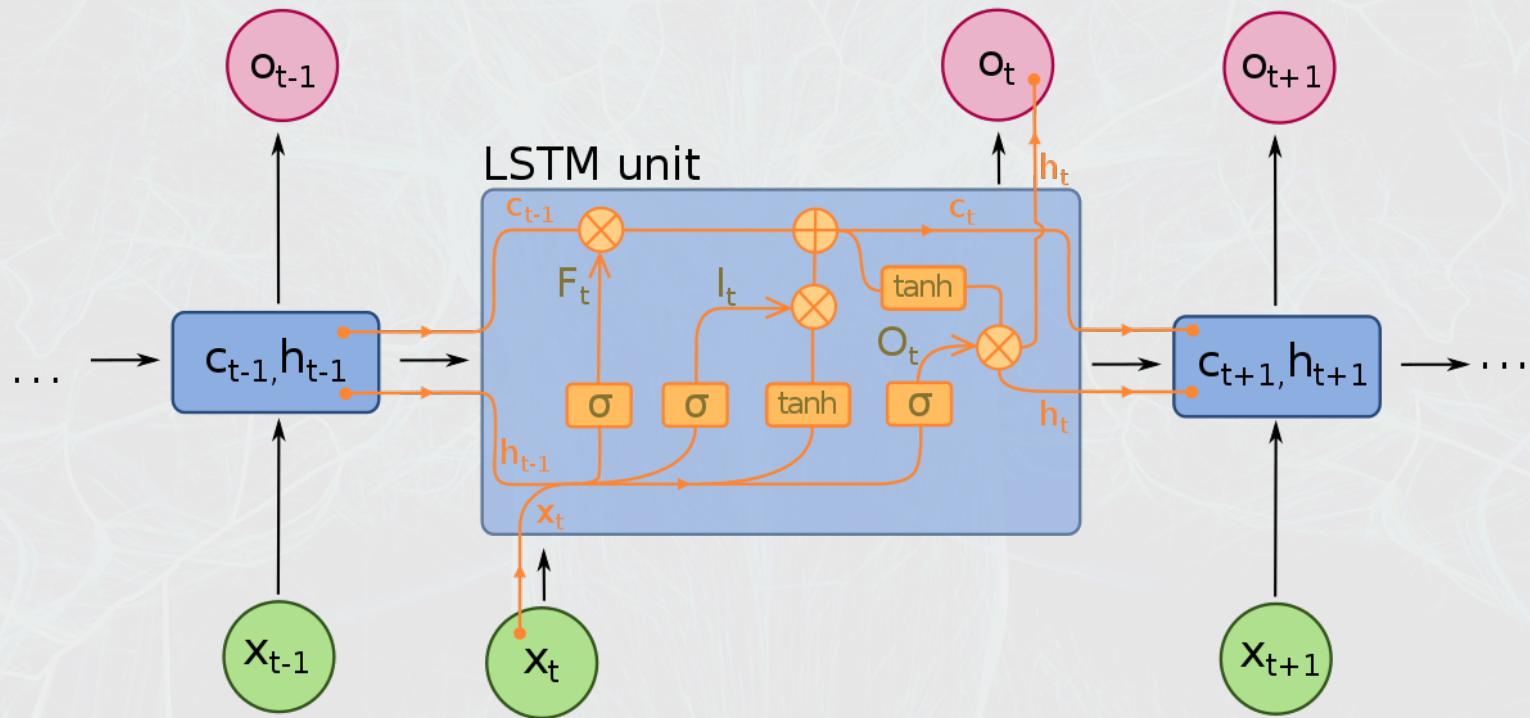
Recurrent Neural Networks (RNNs) meet
these sequence modeling design criteria

Long Short-Term Memory (LSTM)

- **Long short-term memory** (LSTM) é um sistema de deep learning que evita o **vanishing gradient problem**.
- O LSTM é normalmente aumentado por portas recorrentes chamadas “forget gates”.
- LSTM evita que erros retropropagados (backpropagated) desapareçam ou explodam.
- Em vez disso, os erros podem fluir para trás através de um número ilimitado de camadas virtuais desdobradas no espaço.
- Ou seja, o LSTM pode aprender tarefas que requerem memórias de eventos que aconteceram milhares ou até milhões de etapas discretas de tempo antes.

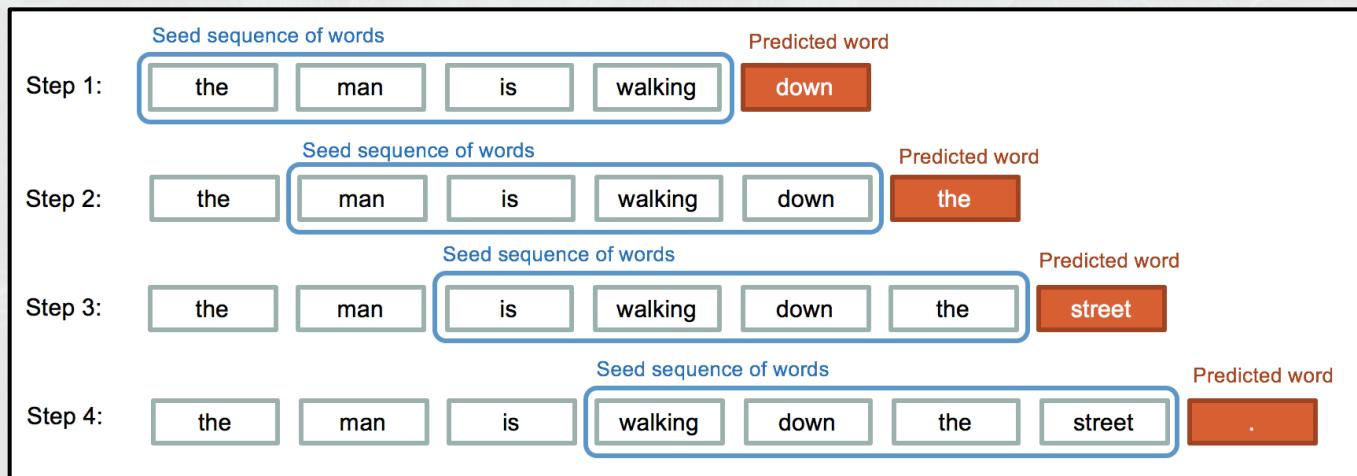
Long Short-Term Memory (LSTM)

- A seguir temos o exemplo de uma unidade de **long short-term memory**:



Recurrent Neural Network

- RNNs são adequados para tarefas de **modelagem de sequência**.
- Eles modelam sequências via **relação de recorrência**.
- O treinamento de RNNs é através de backpropagation pelo tempo.
- **Gated cells** como LSTMs nos permitem modelar dependências de longo prazo.



Convolutional Neural Network

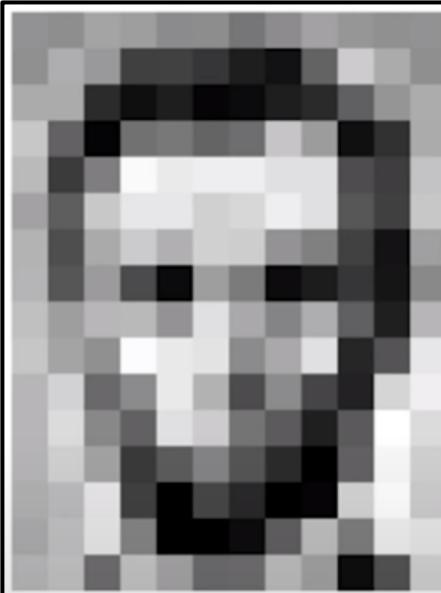
- Em deep learning, uma **convolutional neural network** (CNN ou ConvNet) é uma classe de **deep neural network**, mais comumente aplicada para analisar imagens visuais.
- Eles têm aplicações em reconhecimento de imagem e vídeo, sistemas de recomendação, classificação de imagens, segmentação de imagens, análise de imagens médicas, processamento de linguagem natural, interfaces cérebro-computador e séries temporais financeiras.
- O nome "convolutional neural network" indica que a rede emprega uma operação matemática chamada **convolution**.
- As convolutional networks são um tipo especializado de redes neurais que usam **convolution** no lugar da multiplicação geral da matriz em pelo menos uma de suas camadas.

Convolutional Neural Network

- CNNs são versões regularizadas de perceptrons multicamadas.
- Perceptrons multicamadas geralmente significam fully connected networks, ou seja, cada neurônio em uma camada é conectado a todos os neurônios na próxima camada.
- A "conectividade total" dessas redes as torna propensas a **overfitting** de dados.
- Formas típicas de regularização, ou prevenção de overfitting, incluem: penalizar parâmetros durante o treinamento (como **weight decay**) ou cortar conectividade (conexões ignoradas, **dropout**, etc.), CNNs têm uma abordagem diferente para a regularização: eles obtém vantagem do padrão hierárquico nos dados e montam padrões de complexidade crescente usando padrões menores e mais simples gravados em seus filtros. Portanto, em uma escala de conectividade e complexidade, as CNNs estão no extremo inferior.

Convolutional Neural Network

- O que um computador vê? Para um computador, imagens são números.



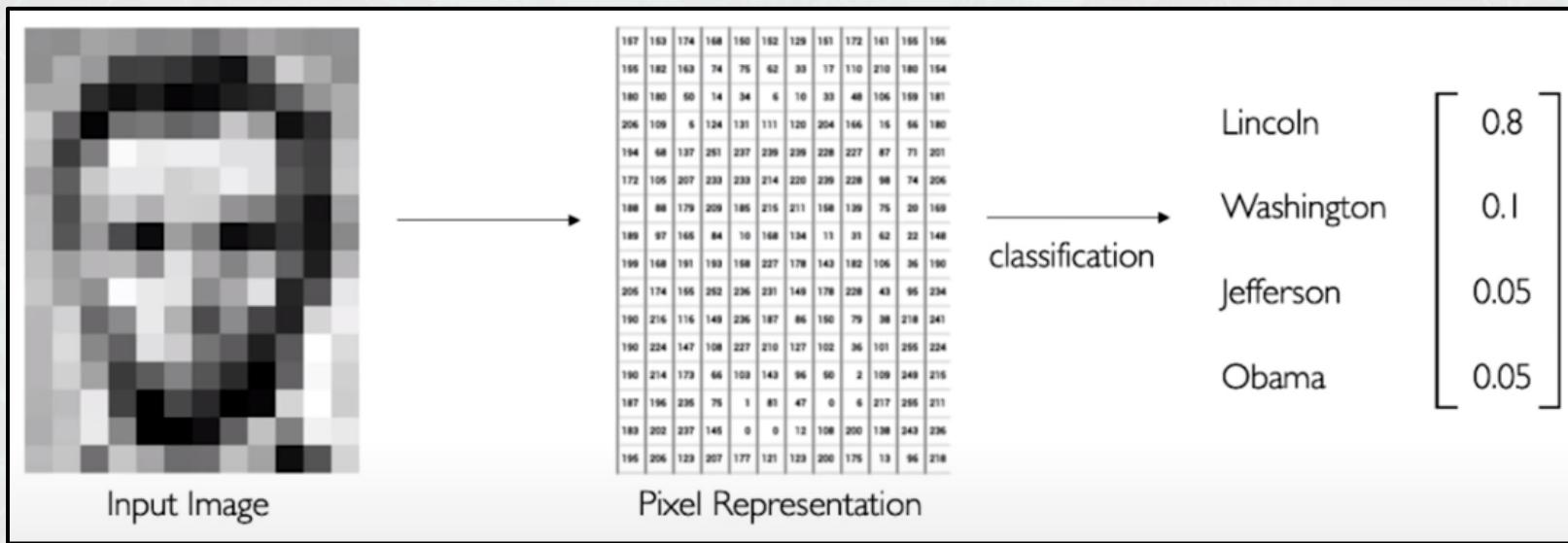
157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	93	17	110	210	180	154
180	180	50	14	34	6	10	93	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	197	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	156	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

157	153	174	168	160	152	129	151	172	161	155	156
195	182	163	74	75	62	93	17	110	210	180	154
180	180	50	14	34	6	10	93	48	106	159	181
206	109	5	124	131	111	120	204	166	16	56	180
194	68	197	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Uma imagem é apenas uma matriz de números [0, 255]
Por exemplo: 500x500x3 para uma imagem RGB

Convolutional Neural Network

- Como vimos, existem dois tipos de tarefas comuns de aprendizado:
 - ◆ **Régressão:** variável de saída é um valor contínuo
 - ◆ **Classificação:** variável de saída é um rótulo de classe. Também pode produzir uma probabilidade de pertencer a uma determinada classe.



Convolutional Neural Network

- **High Level Feature Detection:** para reconhecermos e classificarmos imagens, é importante identificarmos as características chaves em um determinada categoria de imagem.



Nariz, Olhos, Boca



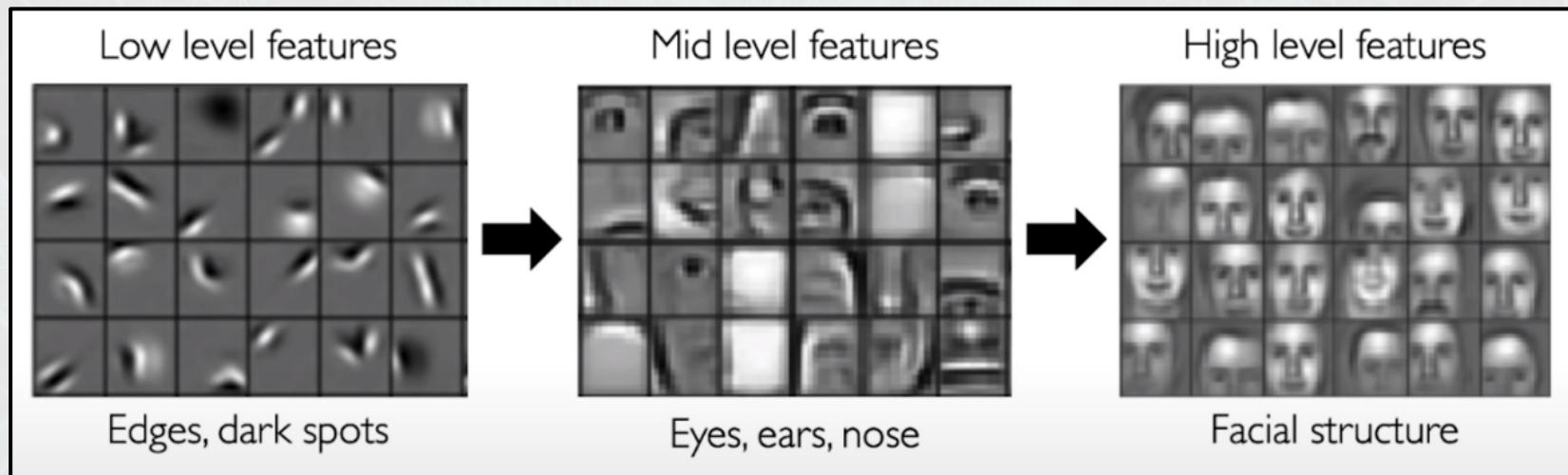
Rodas, faróis, placa de licença



Porta, janelas, escadaria, telhado

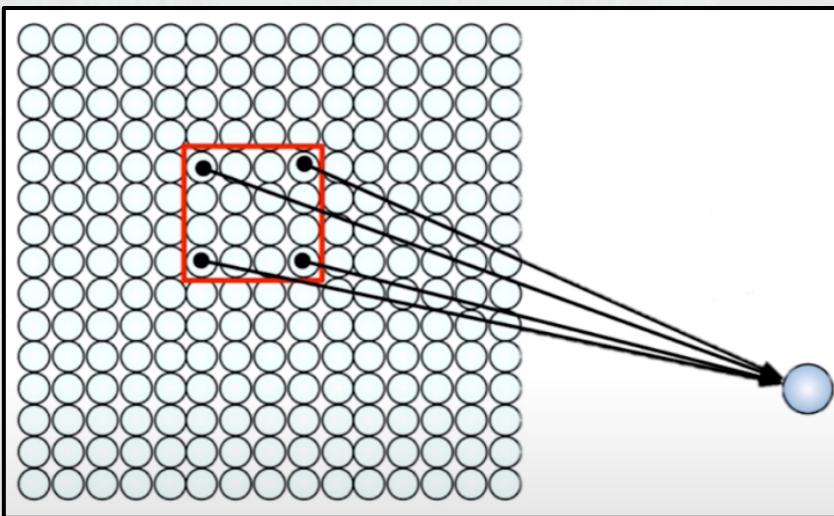
Convolutional Neural Network

- Detectar características em uma imagem de forma manual é um problema muito desafiador.
- Podemos usar redes neurais (especialmente CNNs), para detectar essas características de forma automática.



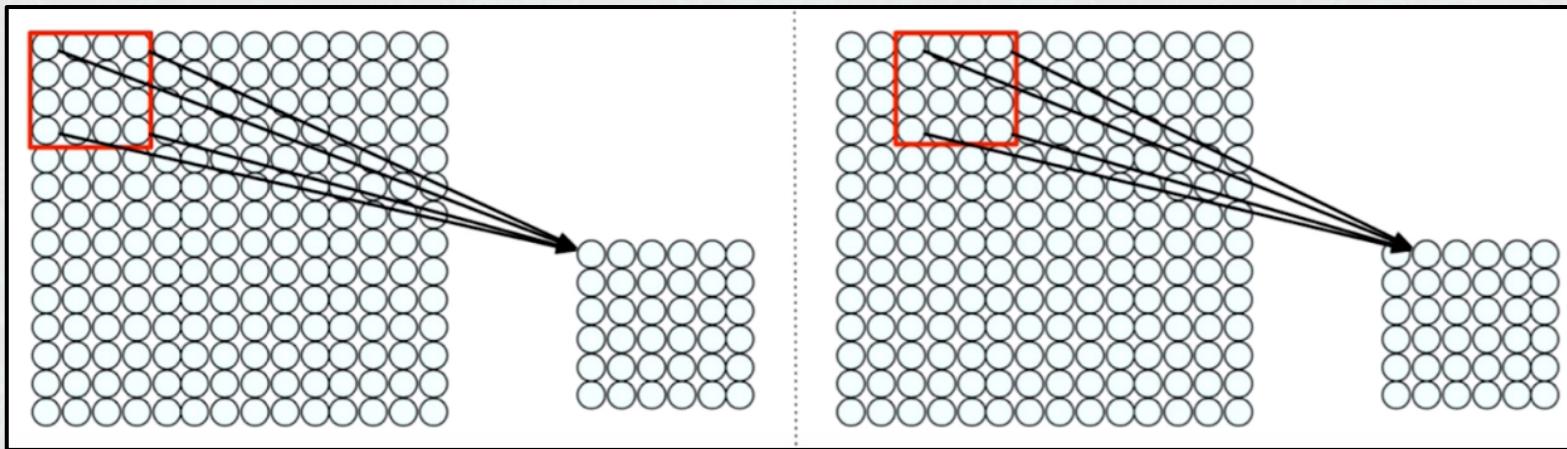
Convolutional Neural Network

- Para detectar as características automaticamente, podemos usar uma **Spatial Structure**.
- **Input:** Imagem 2D, array de valores de pixel
- **Ideia:** Conectar "áreas" do input aos neurônios das camadas ocultas.



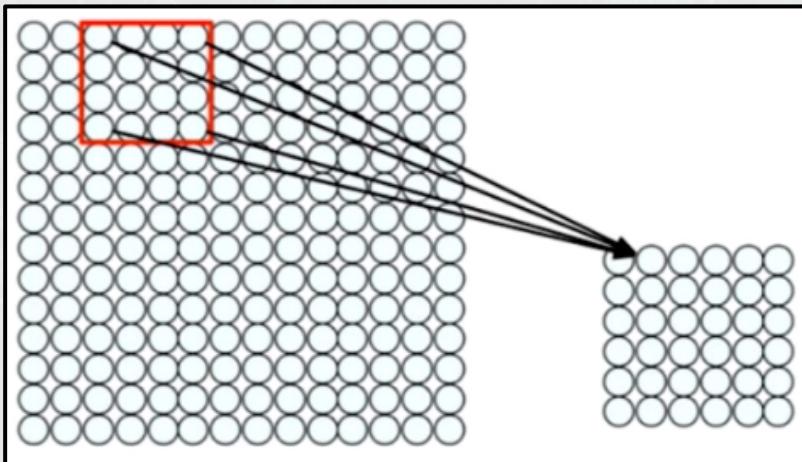
Convolutional Neural Network

- Conectar uma "área" na camada de entrada a um único neurônio na camada subsequente.
- Use uma "janela" deslizante para definir as conexões.
- Como nós podemos estabelecer um "**weight**" à "área" para detectar características particulares?



Convolutional Neural Network

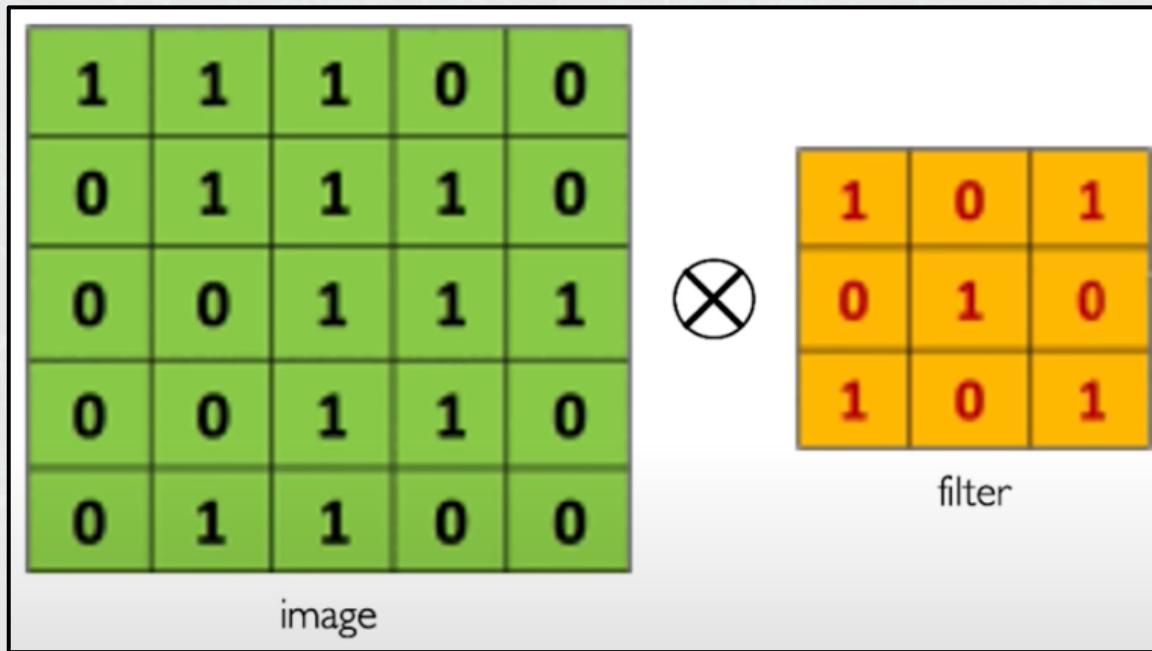
- Extração de características com Convolution:
 - ◆ Filtro de tamanho 4x4: 16 diferentes weights;
 - ◆ Aplicar este mesmo filtro à 4x4 áreas no input;
 - ◆ Deslocar por 2 pixels para a próxima área.



1. Aplicar um conjunto de weights – um filtro – para extrair **características locais**;
2. Usar **múltiplos filtros** para extrair diferentes características;
3. **Compartilhar espacialmente** os parâmetros de cada filtro.

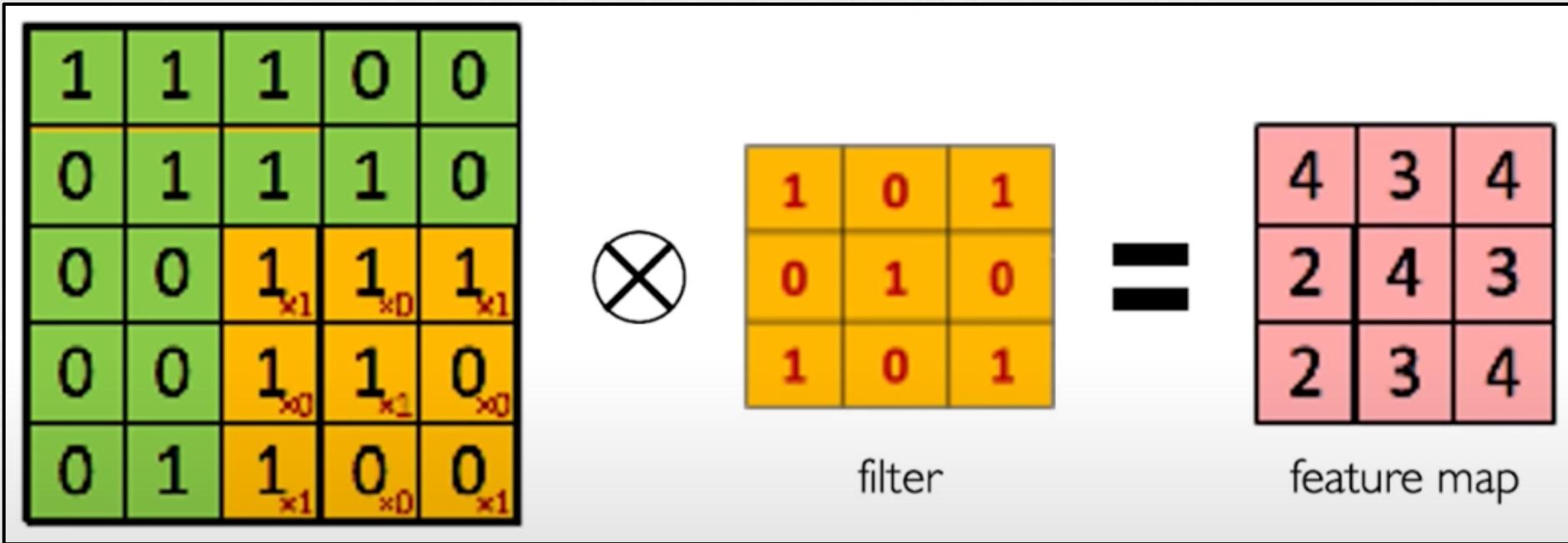
Convolutional Neural Network

- Imagine que desejamos computar a convolution de uma imagem 5x5 e um filtro 3x3:



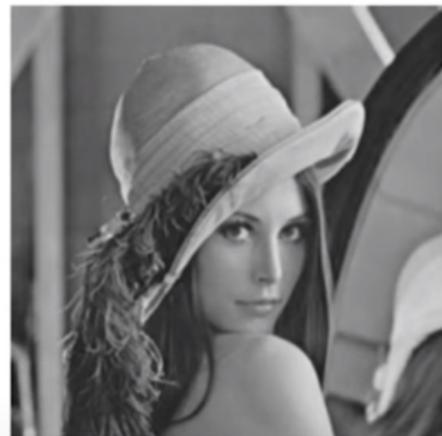
Convolutional Neural Network

- Deslocamos o filtro 3x3 por cima da imagem de input e executamos uma multiplicação *element-wise* e adicionamos os outputs:



Convolutional Neural Network

- A seguir podemos ver diferentes mapas de características, produzidos usando diferentes filtros:



Original



Sharpen



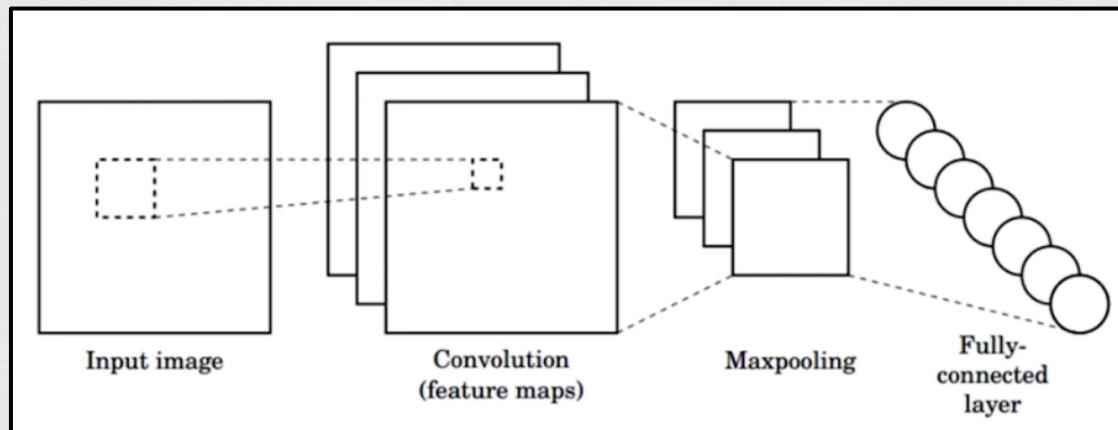
Edge Detect



"Strong" Edge
Detect

Convolutional Neural Network

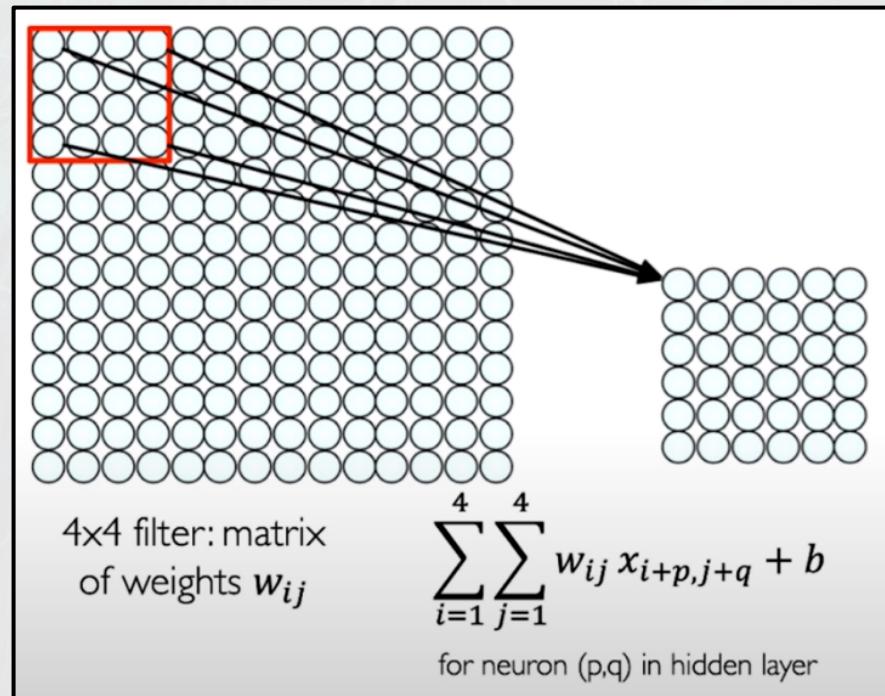
- CNNs são muito usadas na tarefa de classificação:
 1. **Convolution**: Aplicar filtros para gerar os mapas de características;
 2. **Não-linearidade**: Normalmente usa-se **ReLU**;
 3. **Pooling**: Operação downsampling em cada mapa de características.



Treinar o modelo com dados de imagem. Aprender os **weights** de filtros em camadas convolutional.

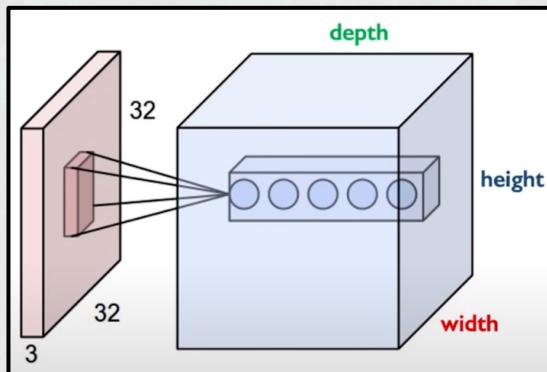
Convolutional Neural Network

- A operação **convolution** funciona da seguinte forma:
 - Para um neurônio em uma camada oculta:
 - ◆ Recebe inputs de uma "área";
 - ◆ Calcula a soma "weighted";
 - ◆ Aplica o bias.
1. Aplicar uma janela de weights;
 2. Computar as combinações lineares;
 3. Ativar com uma função não-linear.



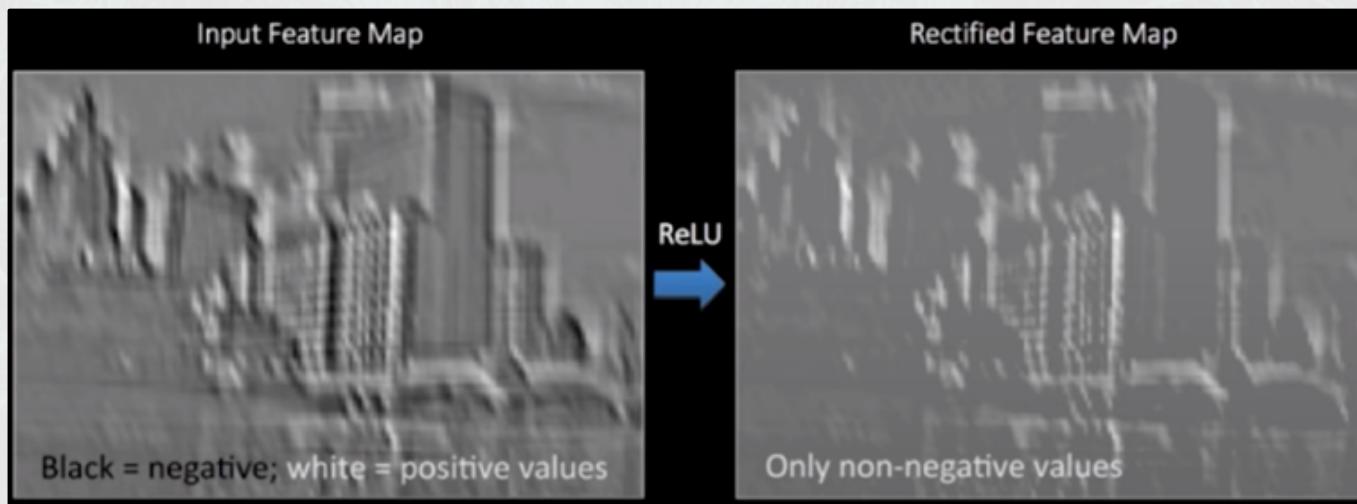
Convolutional Neural Network

- Arranjo espacial do volume de output:
- **Dimensões das camadas:**
 - ◆ $h \times w \times d$
 - ◆ Onde **h** e **w** são dimensões espaciais e **d** (depth) = número de filtros;
- **Stride:** tamanho de passos do filtro;
- **Receptive Field:** Localizações na imagem de input que um nó está path connected.



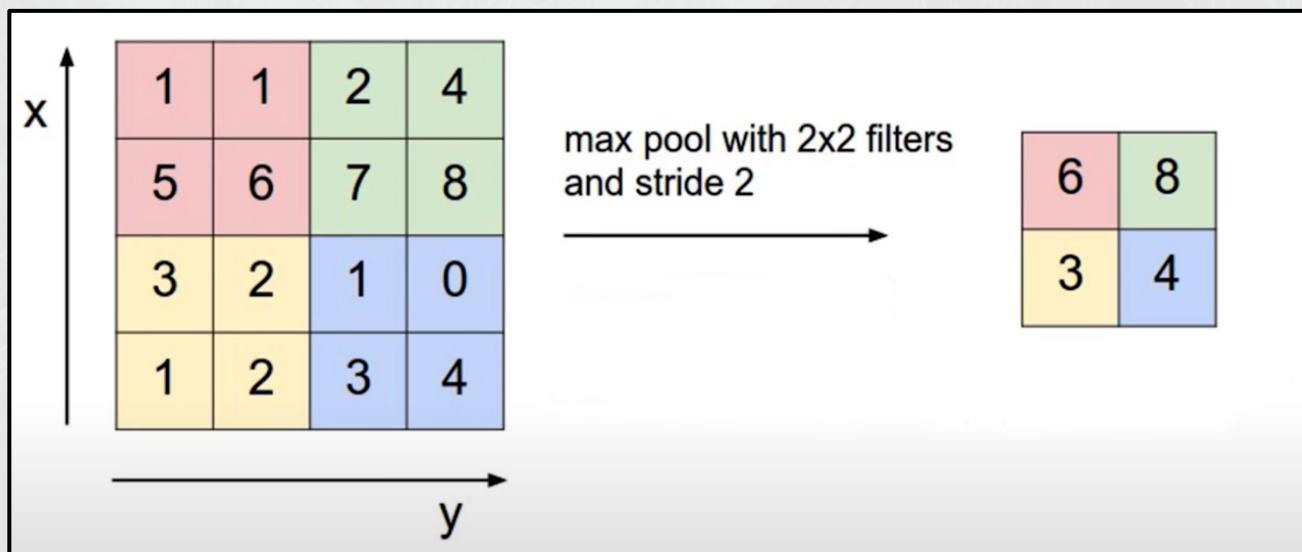
Convolutional Neural Network

- A próxima etapa, após todas as operações convolution (após as camadas convolutional) é aplicar a função não-linear.
- **ReLU:** Operação pixel-por-pixel que substitui todos os valores negativos por zero. (Operação não-linear)



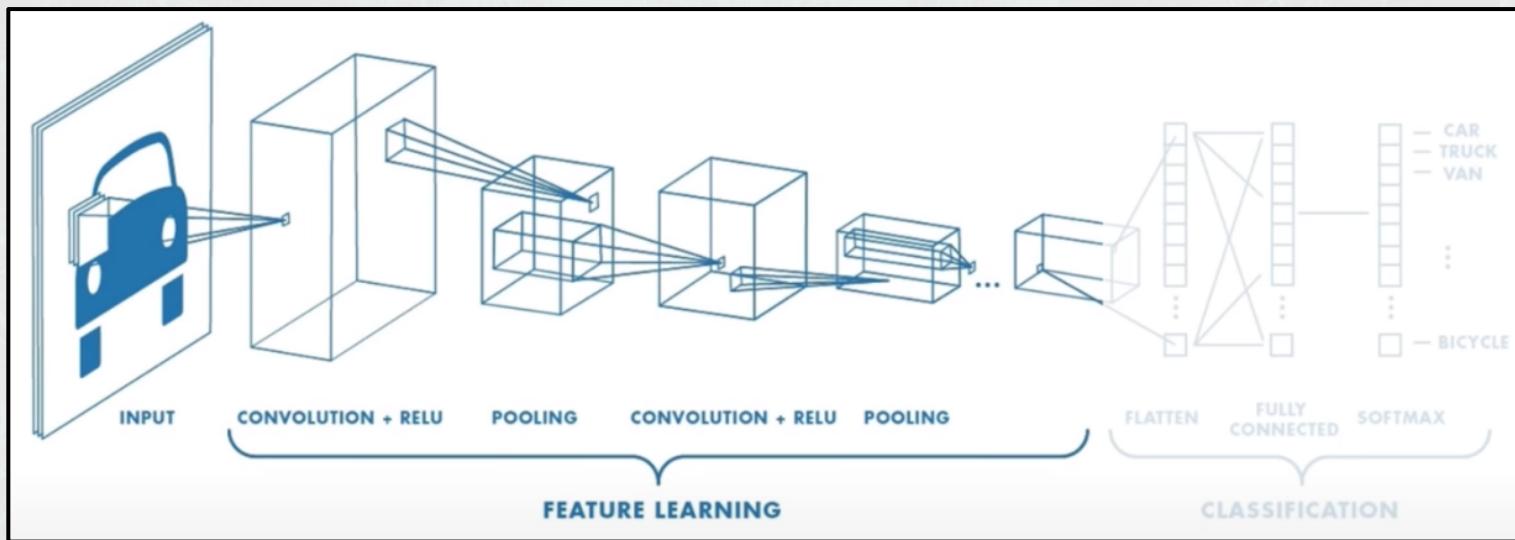
Convolutional Neural Network

- A próxima operação se chama **pooling**.
- Ela é uma operação para reduzir a dimensionalidade de nossos inputs e mapas de características, enquanto preserva **spatial invariance**.



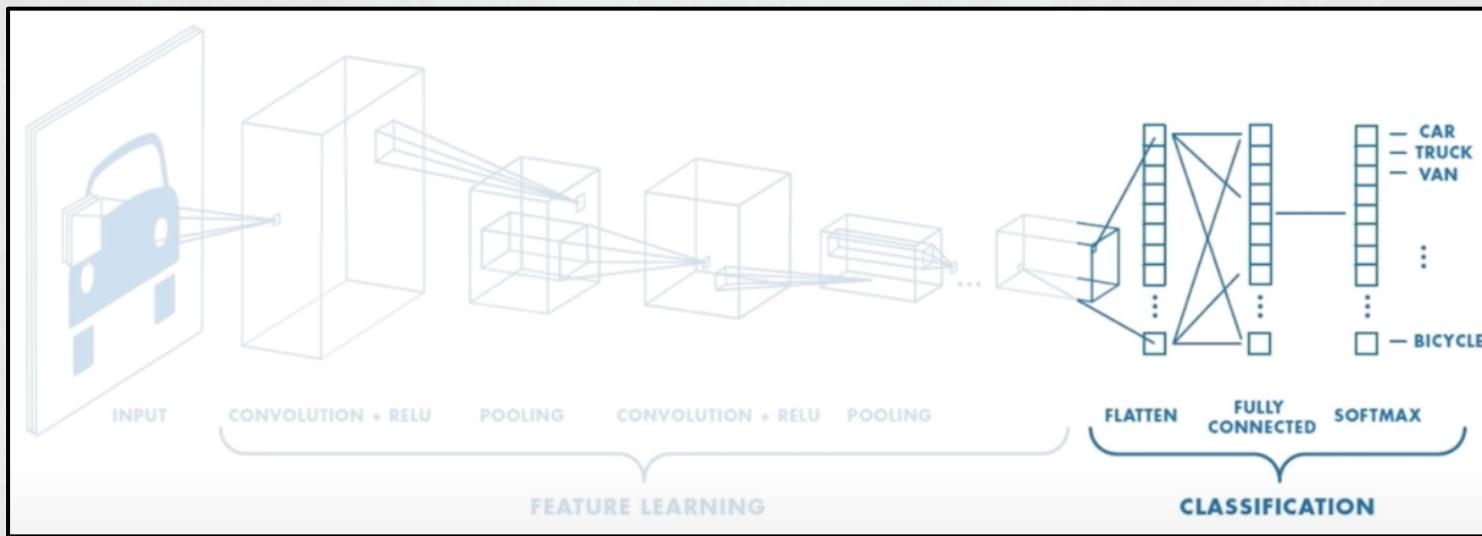
Convolutional Neural Network

- Uma CNN pode ser dividida em duas etapas fundamentais: **feature learning** e **classification**.
 1. Aprender as características de uma imagem de input através de **convolution**.
 2. Introduzir **não-linearidade** através de uma função de ativação. (dados do mundo real são não-lineares).
 3. Reduzir a dimensionalidade e preservar **spatial invariance** com **pooling**.



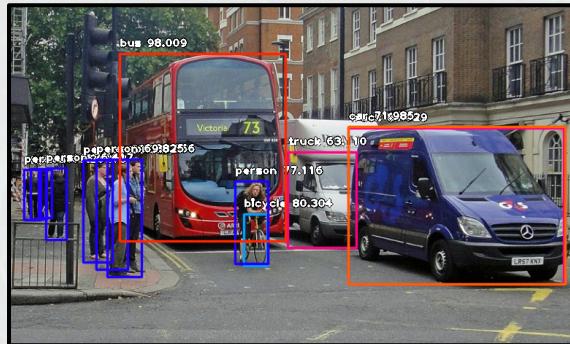
Convolutional Neural Network

- Uma CNN pode ser dividida em duas etapas fundamentais: **feature learning** e **classification**.
 - ◆ Camadas CONV e POOL emitem características de alto nível do input;
 - ◆ Camadas **fully connected** usam essas características para classificar uma imagem de input;
 - ◆ Podem expressar o output como uma **probabilidade** da imagem pertencer a uma classe particular com a função **softmax**.



Convolutional Neural Network

- É importante lembrarmos que as CNNs podem ser aplicadas em uma variedade de problemas:
 - ◆ Reconhecimento de imagens;
 - ◆ Análise de vídeos;
 - ◆ Processamento de linguagem natural;
 - ◆ Detecção de anomalias;
 - ◆ Descoberta de drogas;
 - ◆ Avaliação de risco à saúde;
 - ◆ Previsão de séries temporais;
 - ◆ Patrimônio cultural e conjuntos de dados 3D;



Generative Adversarial Network

- Uma **generative adversarial network** (GAN) é uma classe de frameworks de machine learning desenvolvida por **Ian Goodfellow** e seus colegas em 2014.
- Duas redes neurais competem entre si em um jogo (na forma de um jogo de soma zero, em que o ganho de um agente é a perda de outro).
- Dado um conjunto de treinamento, essa técnica aprende a gerar novos dados com as mesmas estatísticas do conjunto de treinamento.
- Por exemplo, uma GAN treinada em fotografias pode gerar novas fotografias que parecem pelo menos superficialmente autênticas para observadores humanos, com muitas características realistas.

Generative Adversarial Network

- Uma **generative adversarial network** (GAN) possui duas partes:
 - ◆ O **gerador** aprende a gerar dados plausíveis. As instâncias geradas tornam-se exemplos de treinamento negativo para o discriminador.
 - ◆ O **discriminador** aprende a distinguir os dados falsos do gerador dos dados reais. O discriminador penaliza o gerador por produzir resultados implausíveis.
- Quando o treinamento começa, o gerador produz dados obviamente falsos, e o discriminador aprende rapidamente a dizer que são falsos.

Generative Adversarial Network

- O treinamento começa com o **gerador** gerando dados obviamente falsos:



- Conforme o treinamento avança, o **gerador** fica mais perto de produzir uma saída que pode enganar o discriminador:



Generative Adversarial Network

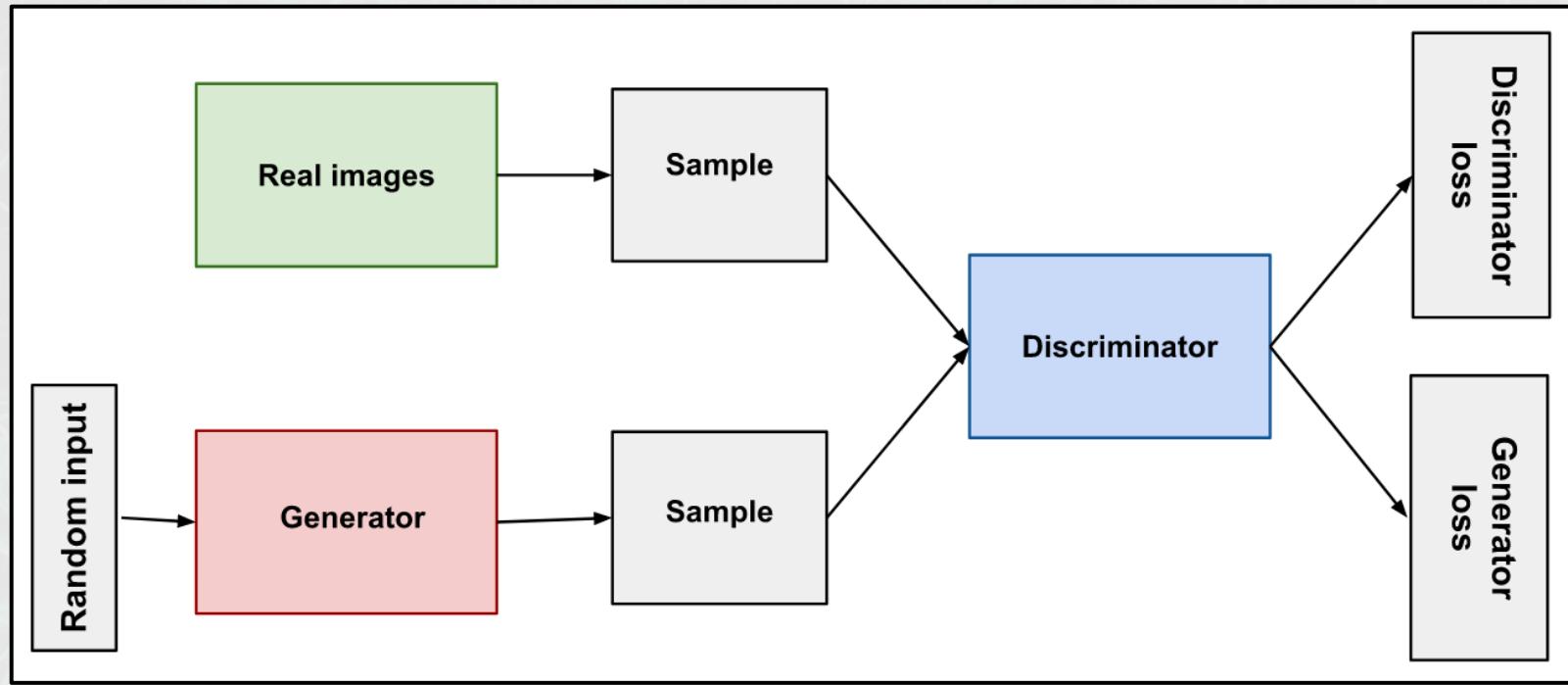
- Finalmente, se o treinamento do **gerador** for bem, o **discriminador** piorará em dizer a diferença entre o real e o falso. Ele começa a classificar os dados falsos como reais e sua precisão diminui.



- Tanto o gerador quanto o discriminador são redes neurais. A saída do gerador é conectada diretamente à entrada do discriminador.
- Por meio de **backpropagation**, a classificação do discriminador fornece um sinal que o gerador usa para atualizar seus **weights**.

Generative Adversarial Network

- Aqui está uma ilustração do sistema completo:



Generative Adversarial Network

- Exemplos de pessoas geradas por **Modelos Gerativos**:

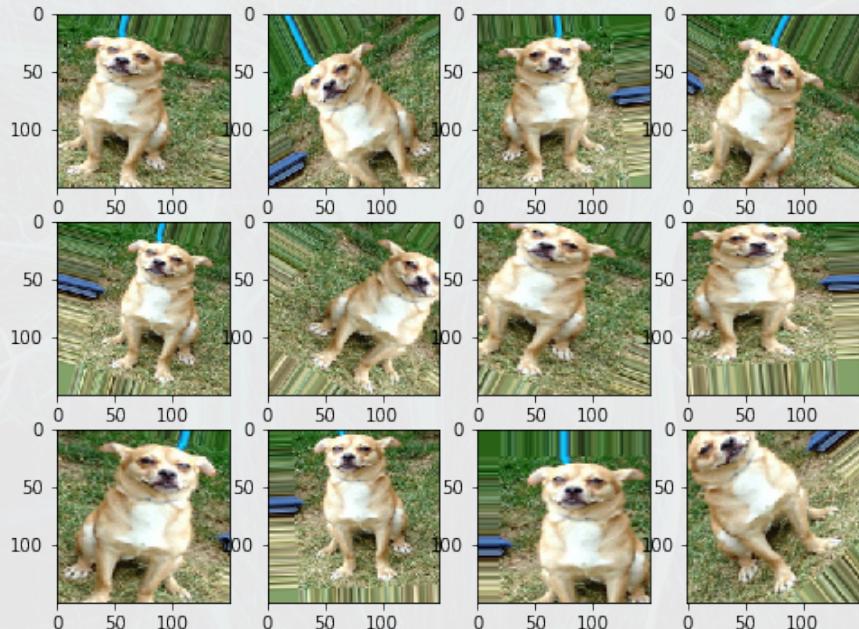


Desafios

- Tal como acontece com RNAs, muitos problemas podem surgir com DNNs treinadas ingenuamente.
- Dois problemas comuns são **overfitting** e **tempo de computação**.
- DNNs são propensas a overfitting devido às camadas adicionadas de abstração, que permitem modelar dependências raras nos dados de treinamento.
- Métodos de regularização, como poda de unidade de Ivakhnenko, **weight decay** (l_2 -regularization) ou **sparsity** (l_1 -regularization), podem ser aplicados durante o treinamento para combater o overfitting.
- Alternativamente, a regularização **dropout** omite unidades aleatoriamente das camadas ocultas durante o treinamento. Isso ajuda a excluir dependências raras.

Desafios

- Finalmente, os dados podem ser **augmented** por meio de métodos como recorte e rotação, de modo que conjuntos de treinamento menores possam ser aumentados em tamanho para reduzir as chances de overfitting.



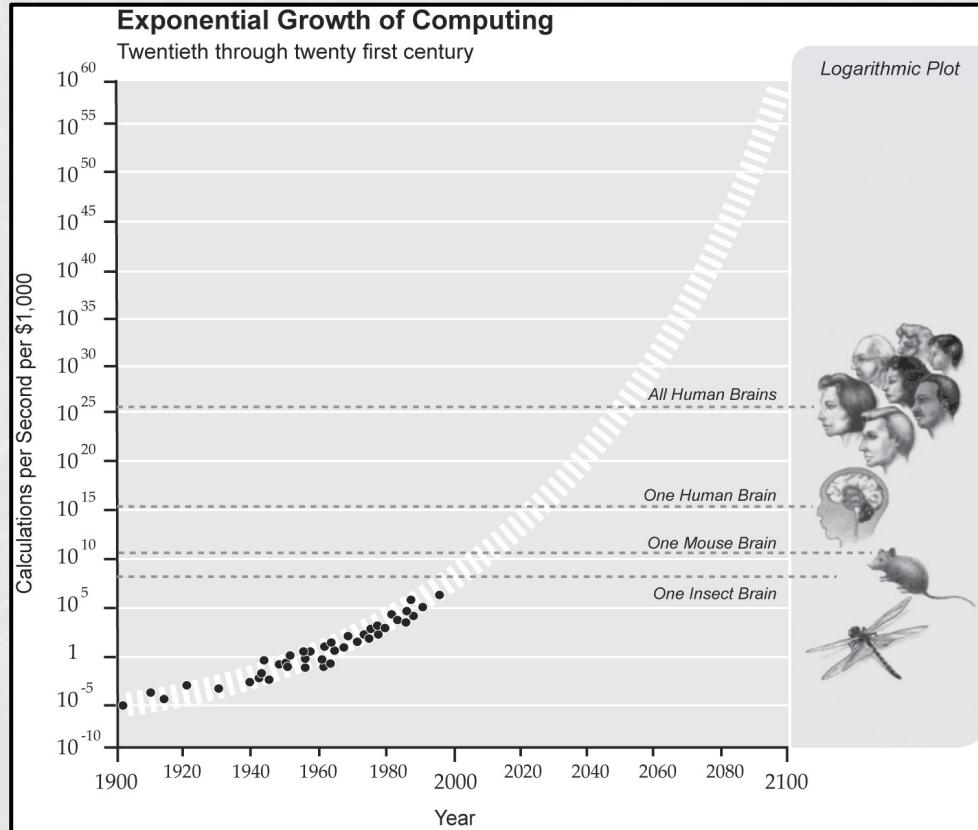
Desafios

- As DNNs devem considerar muitos parâmetros de treinamento, como o tamanho (número de camadas e número de unidades por camada), a taxa de aprendizado e os weights iniciais.
- Varrer o espaço de parâmetros em busca de parâmetros otimizados pode não ser viável devido ao custo em tempo e recursos computacionais.
- Vários truques, como **batching** (computar o gradiente em vários exemplos de treinamento de uma vez em vez de exemplos individuais) aceleram o cálculo.
- Grandes recursos de processamento de arquiteturas de muitos núcleos (como GPUs ou Intel Xeon Phi) produziram acelerações significativas no treinamento, devido à adequação de tais arquiteturas de processamento para cálculos matriciais e vetoriais.

Hardware

- Desde a década de 2010, os avanços em algoritmos de machine learning e hardware de computador levaram a métodos mais eficientes para treinar deep neural networks que contêm muitas camadas de unidades ocultas não lineares e uma camada de saída muito grande.
- Em 2019, as unidades de processamento gráfico (GPUs), muitas vezes com aprimoramentos específicos de IA, substituíram as CPUs como o método dominante de treinamento de IA em nuvem comercial em grande escala.
- A OpenAI estimou a computação de hardware usada nos maiores projetos de deep learning de AlexNet (2012) a AlphaZero (2017) e encontrou um aumento de 300.000 vezes na quantidade de computação necessária, com uma linha de tendência de tempo de duplicação de 3.4 meses.

Hardware



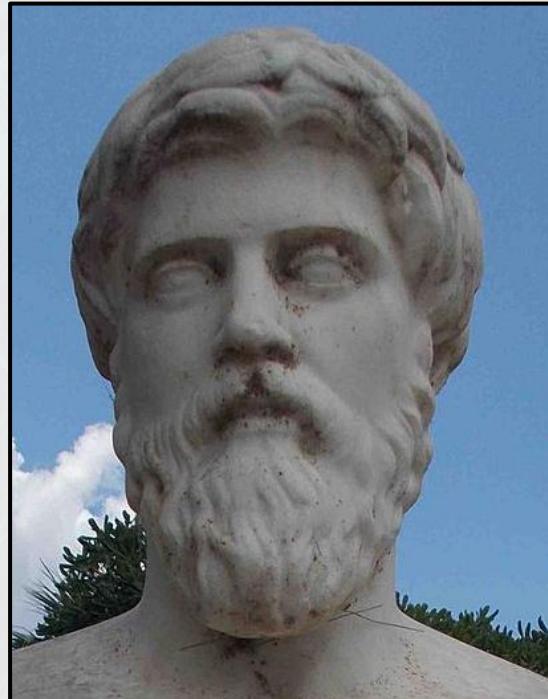
Material Complementar



<https://akiradev.netlify.app/posts/deep-learning/>

Bons Estudos!

“The mind is not a vessel to be filled, but a fire to be kindled.” **Plutarch**



Referências

- https://en.wikipedia.org/wiki/Deep_learning
- <http://introtodeeplearning.com/>
- https://developers.google.com/machine-learning/gan/gan_structure
- https://en.wikipedia.org/wiki/Generative_adversarial_network
- https://en.wikipedia.org/wiki/Artificial_neural_network
- https://en.wikipedia.org/wiki/Convolutional_neural_network
- https://en.wikipedia.org/wiki/Recurrent_neural_network
- <http://www.singularity.com/charts/page70.html>