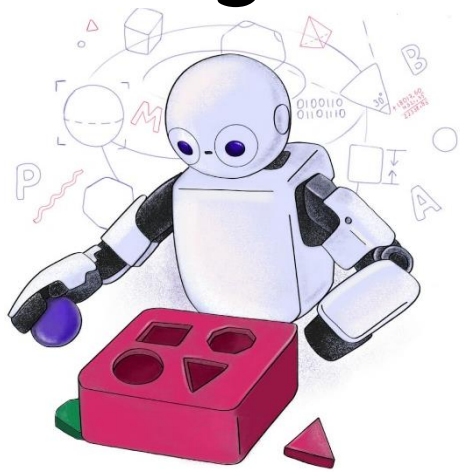
TP557 - Tópicos avançados em IoT e Machine Learning:

Desafios do TinyML:

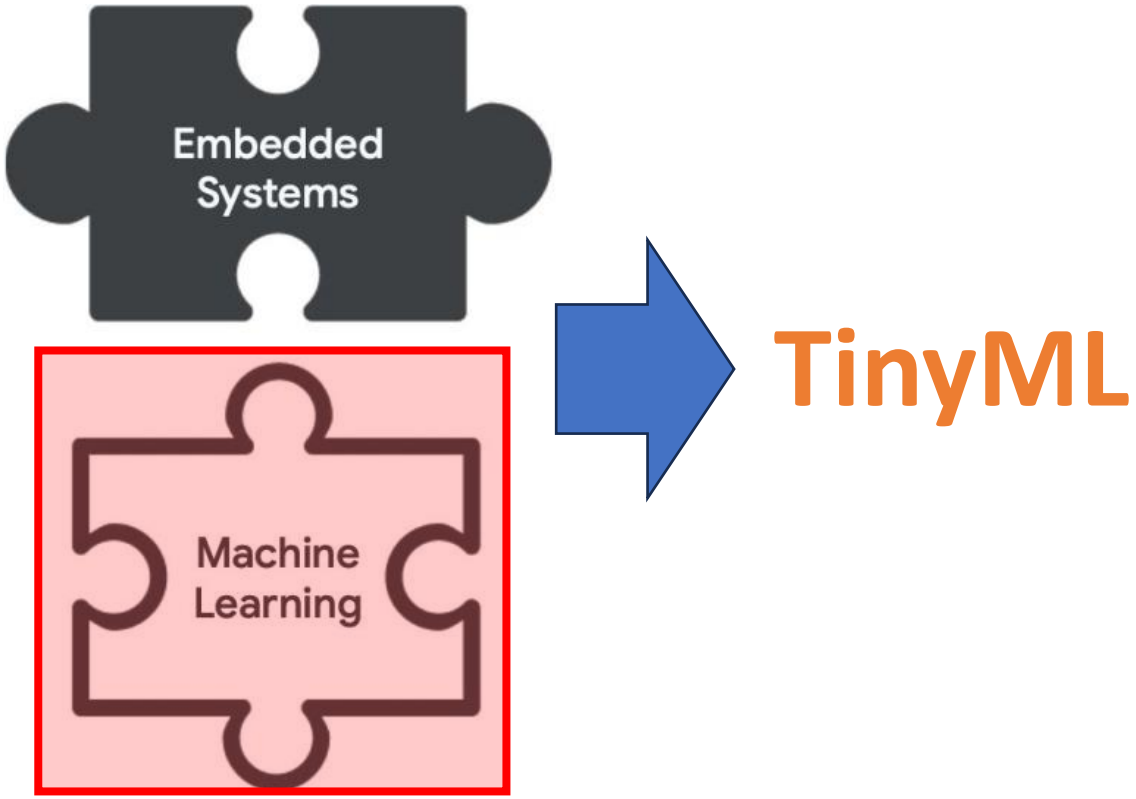
Machine Learning



Inatel

Felipe Augusto Pereira de Figueiredo
felipe.figueiredo@inatel.br

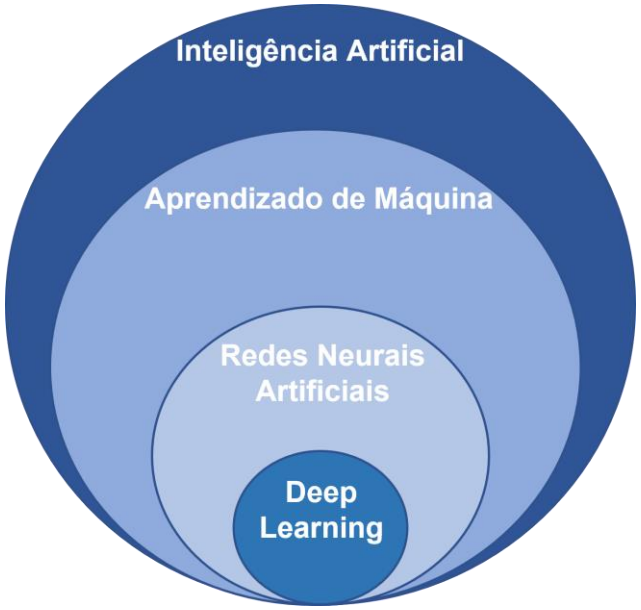
Desafios da execução de ML em sistemas embarcados

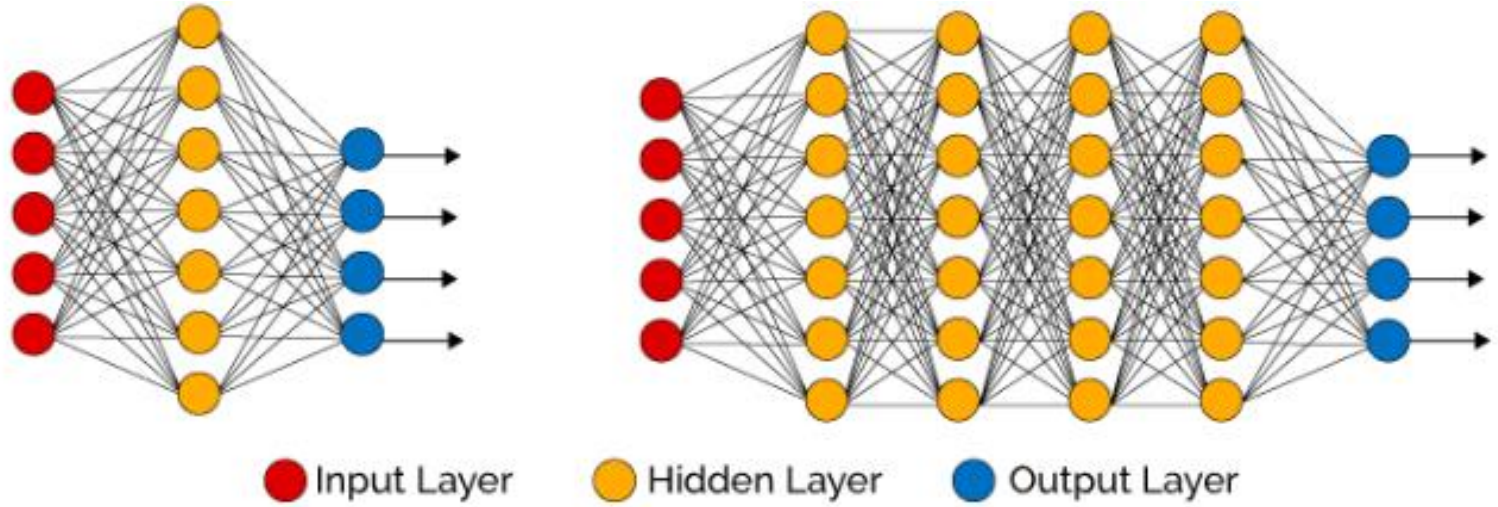


- Anteriormente, falamos dos desafios encontrados quando usamos sistemas embarcados, i.e., limitações de HW e SW.
- Agora, veremos os desafios para execução algoritmos de ML, mais especificamente de *Deep Learning*, nestes dispositivos pequenos, com restrições de custo, recursos computacionais e consumo.

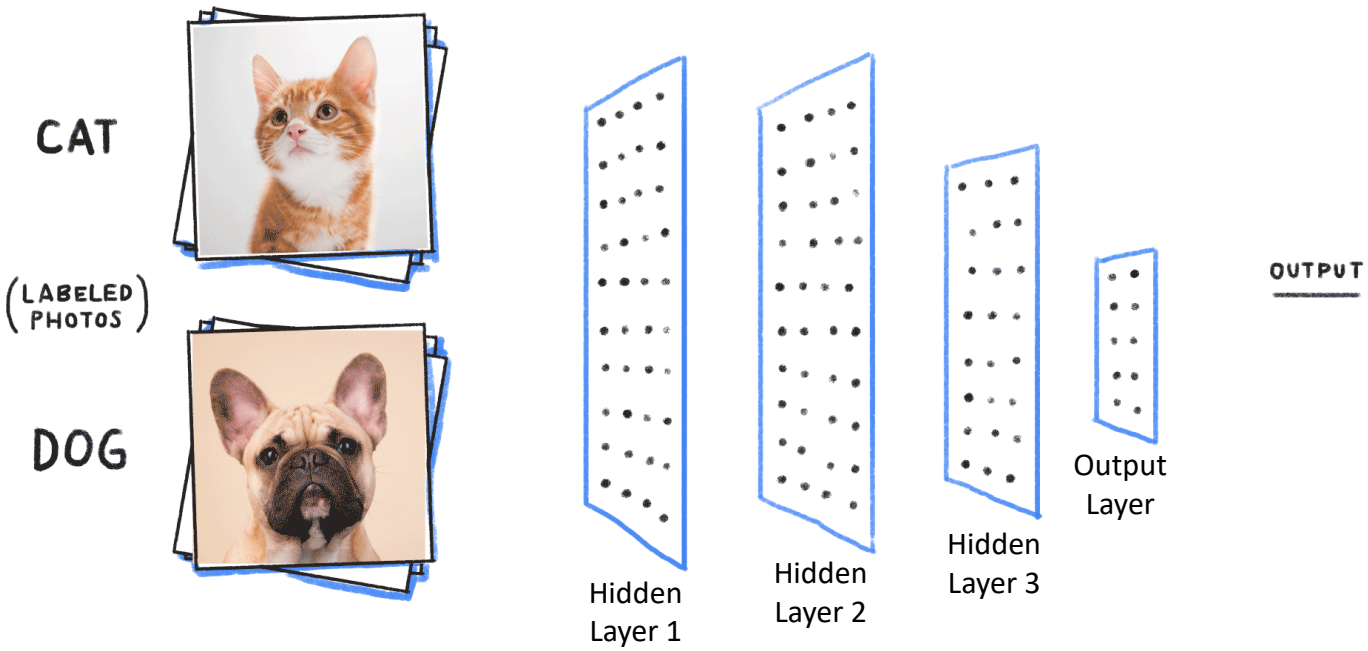
Deep Learning ou Aprendizado Profundo

- Subárea do aprendizado de máquina que usa redes neurais artificiais com muitas camadas ocultas para aprender com dados.
- Por terem uma grande capacidade, em geral, precisam de uma grande quantidade de dados para aprenderem (i.e., encontrar uma solução geral).



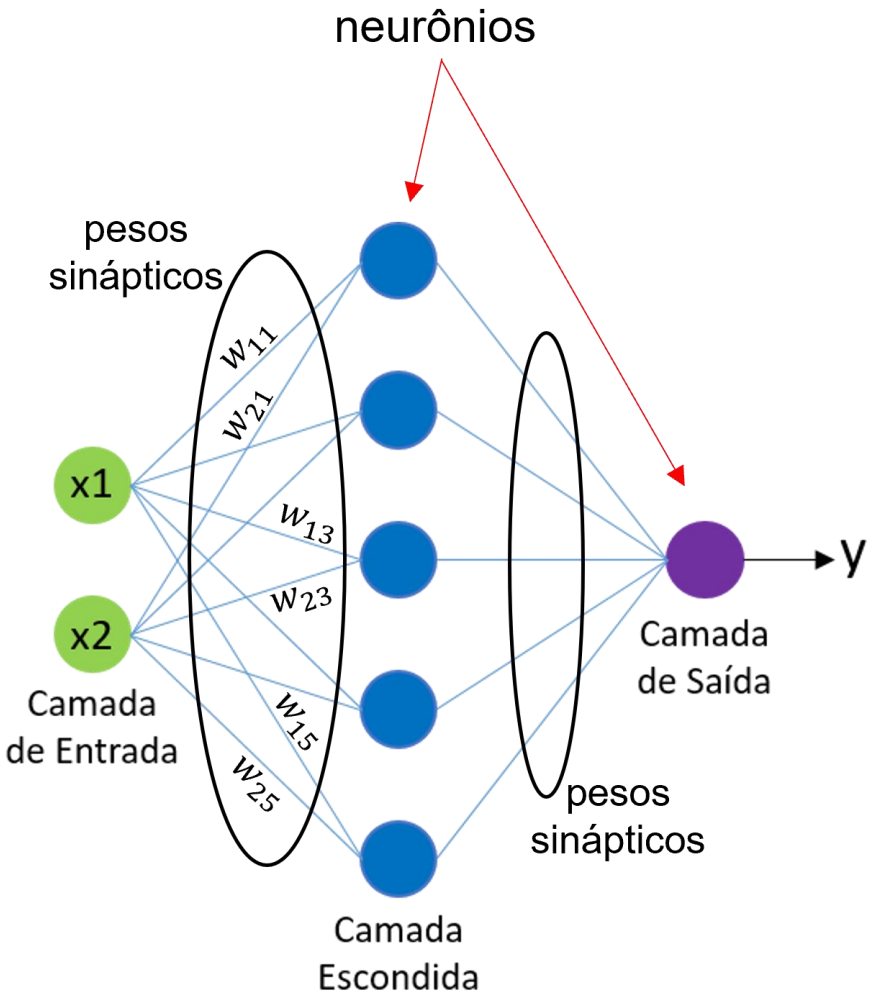


Deep Learning



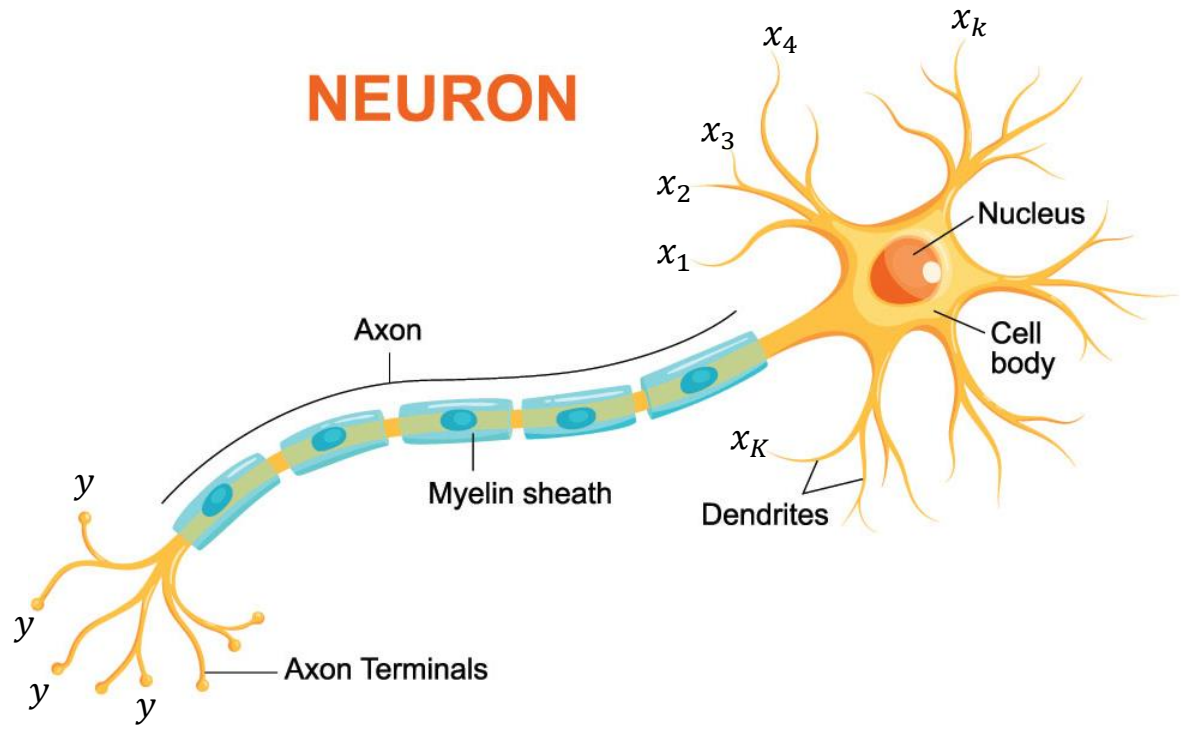
- Um exemplo muito comum de uso do *deep learning* é a classificação de imagens.
- Se tivermos uma base de dados com imagens de gatos e cachorros, podemos treinar uma rede neural profunda para identificá-los em imagens.
- O objetivo é obter um modelo que generalize, ou seja, que identifique gatos ou cachorros não vistos durante o treinamento.

O que é uma rede neural artificial?



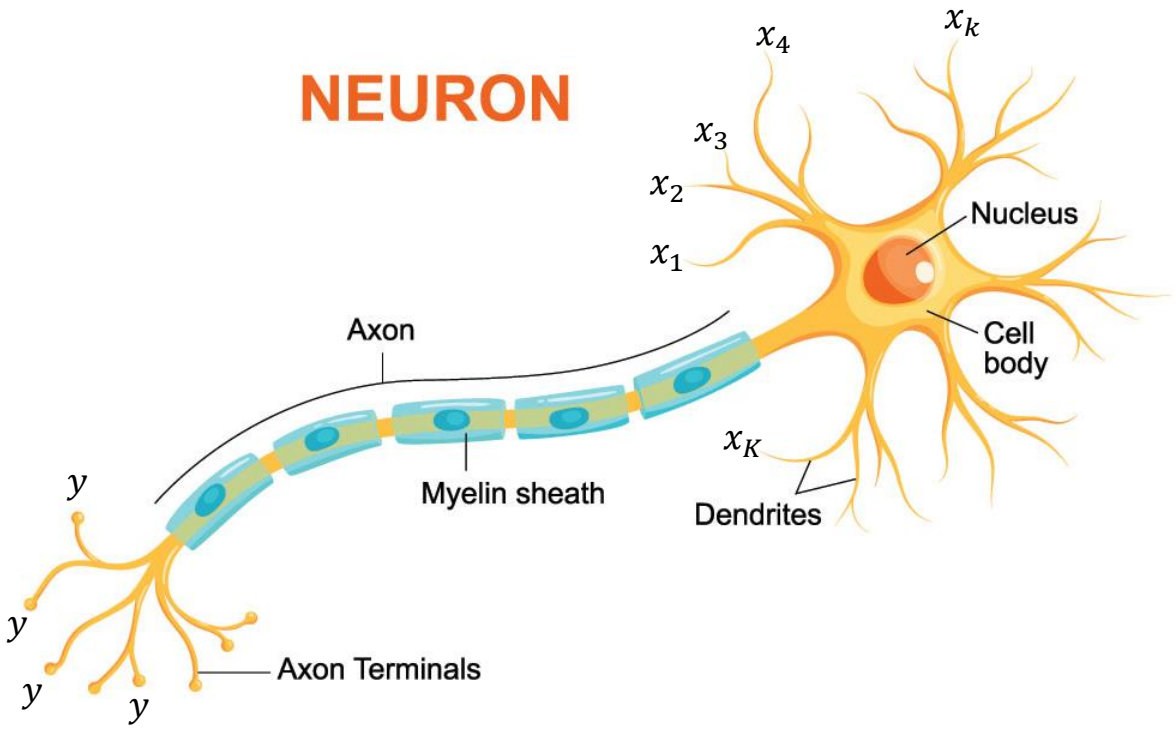
- É uma ***conexão de camadas de neurônios artificiais***, ou também chamados de nós.
- Os ***neurônios artificiais*** são ***modelos matemáticos inspirados no funcionamento de um neurônio biológico***.
- Os neurônios das diferentes camadas são conectados através dos ***pesos sinápticos***, que determinam a força daquela conexão.

Neurônio biológico



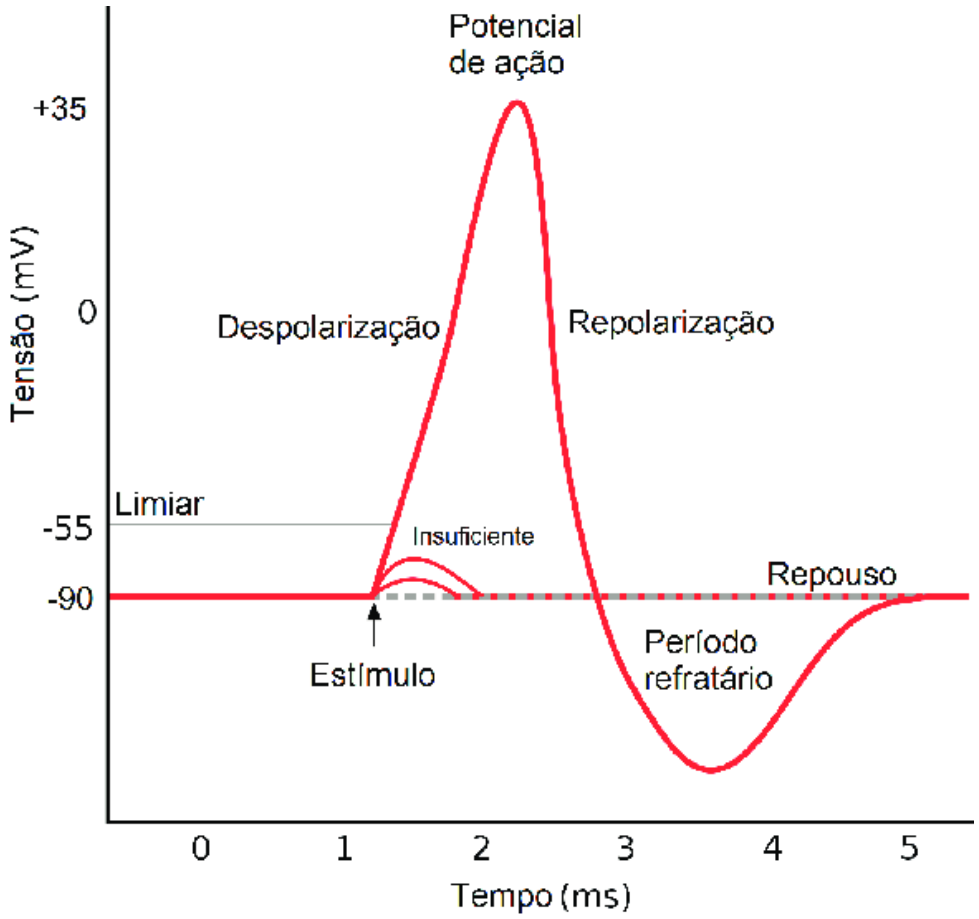
- São células que possuem ***mecanismos eletroquímicos*** para realizar a ***transmissão de sinais elétricos*** (i.e., informações) ***ao longo do sistema nervoso***.
- Têm três partes fundamentais: os ***dendritos***, o ***axônio*** e o ***corpo celular***, também chamado de ***soma***.
- Os ***dendritos*** recebem ***estímulos vindos*** de outros neurônios e os levam em direção ao ***corpo celular***.

Neurônio biológico



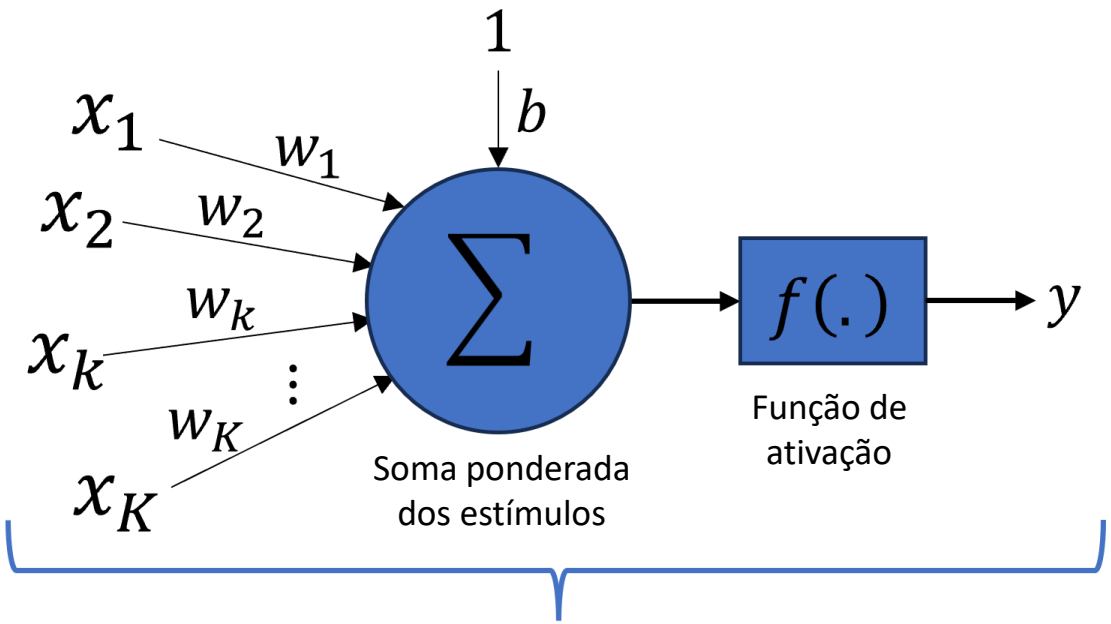
- O ***corpo celular*** realiza a ***integração*** dos ***estímulos***.
- O ***axônio*** envia ***impulsos*** a outros neurônios através de seus ***terminais***.
- Os neurônios se comunicam uns com os outros através das ***sinapses***, que são os pontos de contato entre os dentritos de um neurônio e os terminais do axônio de outros neurônios.

Neurônio biológico



- Em termos bem simples, o funcionamento do ***neurônio*** pode ser explicado da seguinte forma:
  - Ele recebe estímulos elétricos a partir dos dendritos.
  - Esses estímulos são somados no corpo celular (*soma*).
  - Se a soma dos estímulos exceder um certo ***limiar de ativação***, o ***neurônio*** gera um ***impulso*** (ou ***potencial de ação***) que é enviado pelos terminais do axônio a outros neurônios através das ***sinapses***.

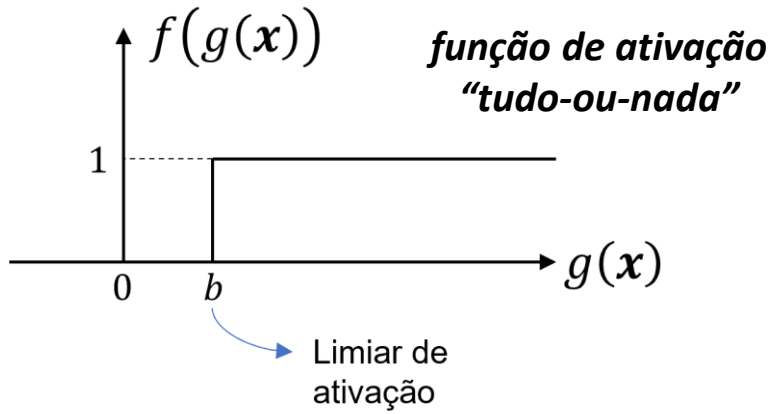
Neurônio artificial

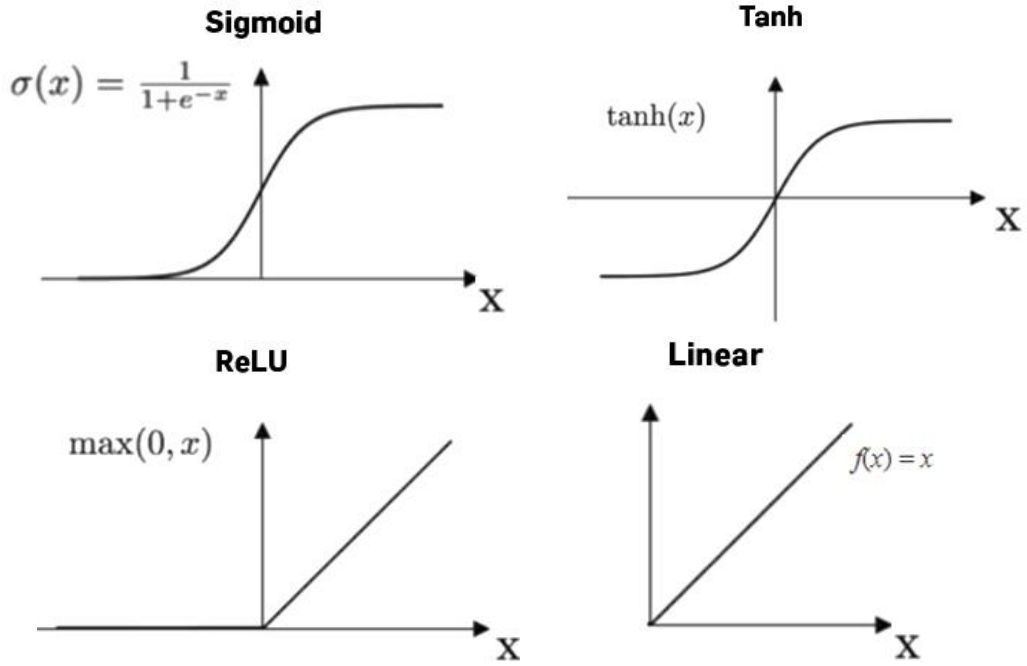


$$y = f\left(\sum_{i=1}^{K} w_i\, x_i + b\right)$$

- Baseado no entendimento do funcionamento do neurônio biológico, a partir de meados da década de 40, pesquisadores propuseram o modelo matemático ao lado.
  - É uma simplificação e abstração do funcionamento do neurônio biológico.
- Os ***estímulos***, $x_k, k = 1, \ldots, K$, são multiplicados pelos ***pesos sinápticos***, $w_k, k = 1, \ldots, K$, somados com o ***peso de bias***, b, e o resultado é passado pela ***função de ativação***, $f(.)$.

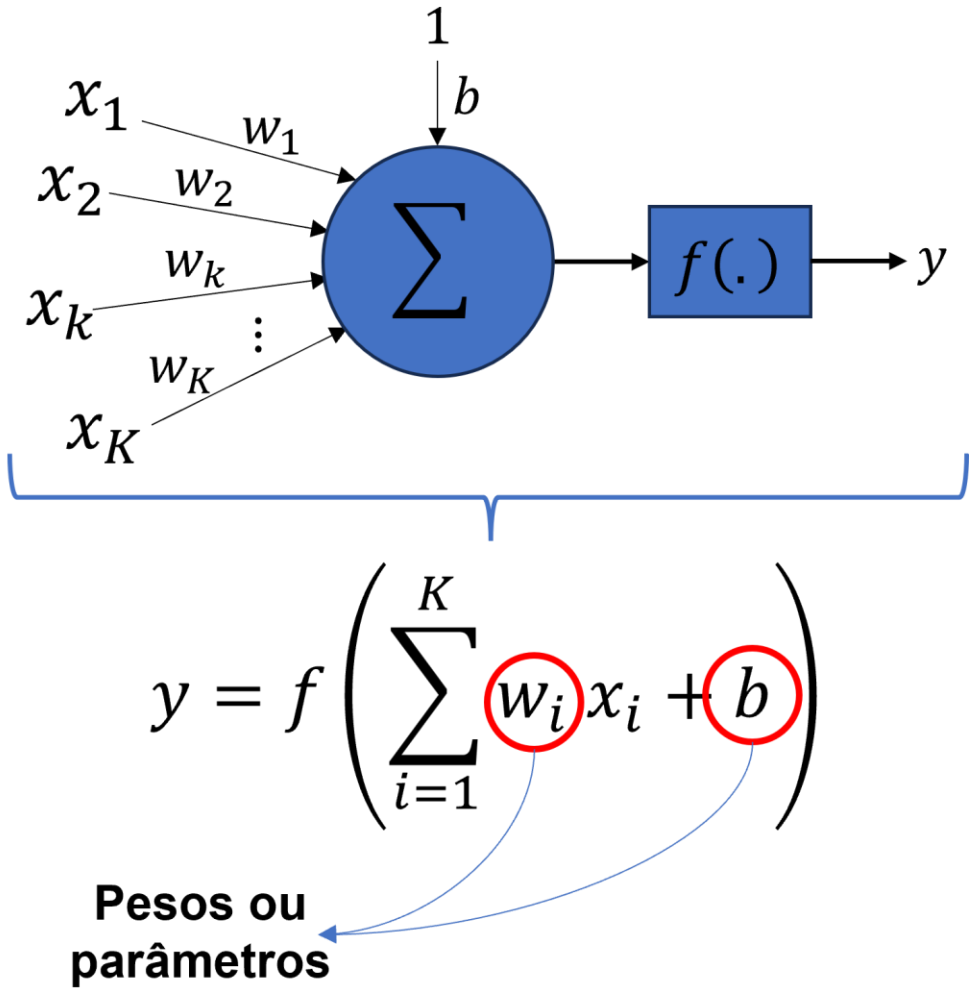
Neurônio artificial





- O ***peso de bias,*** b, permite ajustar o limiar de ativação (ou ponto de disparo) da função de ativação.
- No início dos estudos dos neurônios, a ***função de ativação*** era do tipo ***tudo-ou-nada***, ou seja, ativava ou não (degrau unitário).
- Com o decorrer do tempo, outras funções com características diferentes foram propostas, tais como as funções sigmóide, tangente hiperbólica, relu, linear, etc.

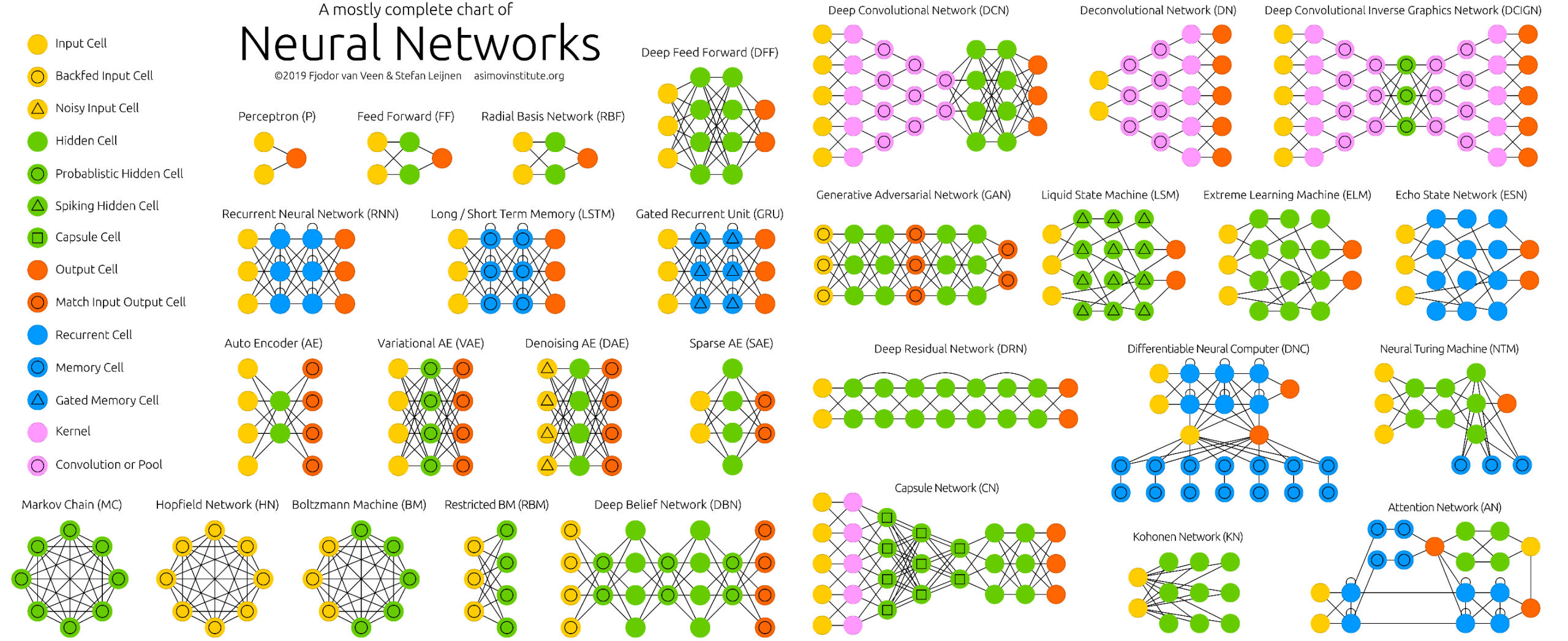
Neurônio artificial



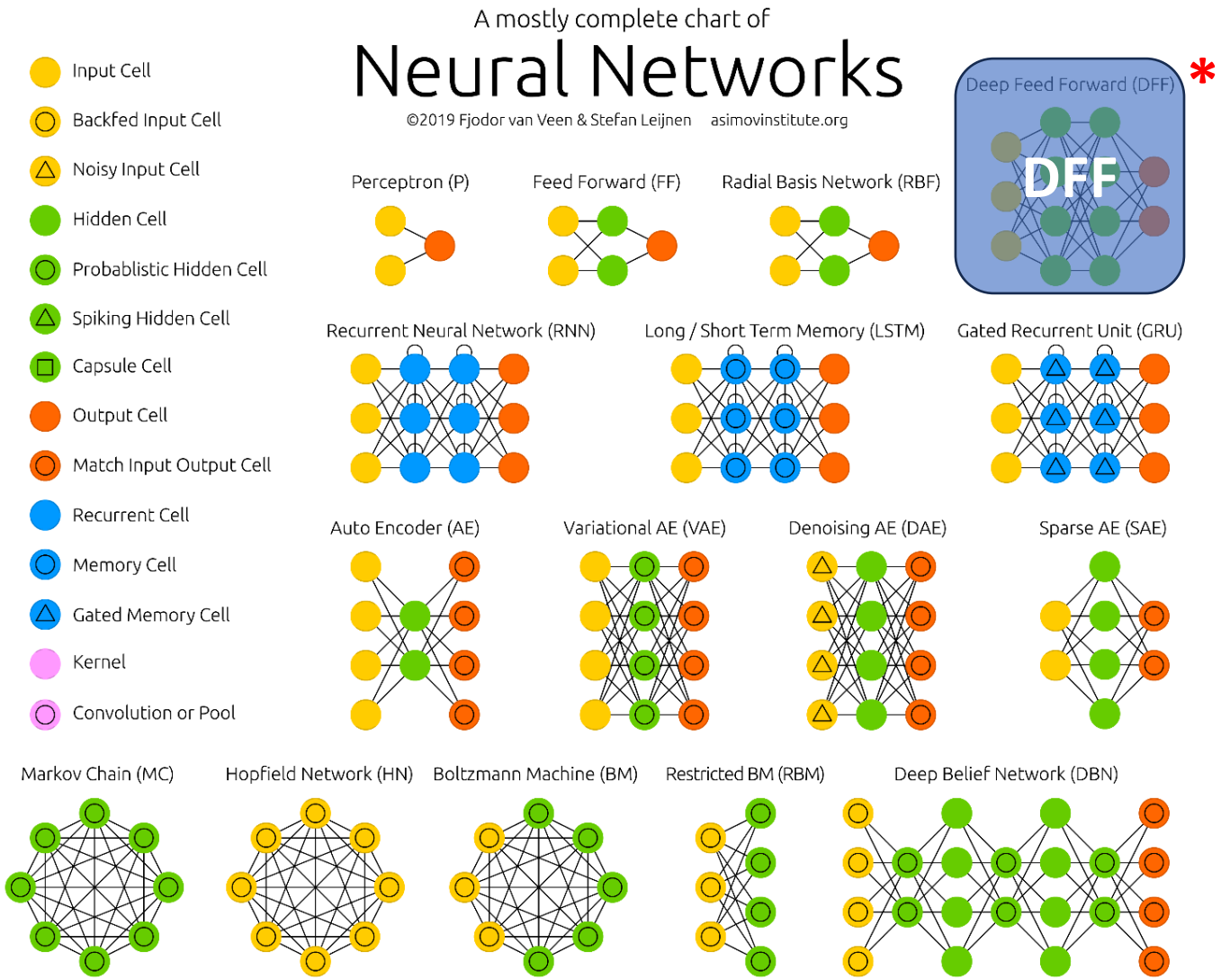
- O ***objetivo*** do treinamento de uma rede neural artificial é ***encontrar os valores ideais dos pesos*** (bias e sinápticos) de todos os neurônios de forma que a rede ***resolva uma tarefa específica***, como, por exemplo, a classificação de imagens.
- Os pesos são ajustados através de um processo iterativo chamado de ***retropropagação do erro***, onde apresenta-se ao modelo as ***entradas*** e ***saídas esperadas*** e o processo minimiza o erro entre a ***saída da rede*** e os ***valores de saída esperados***.

Topologias de redes neurais artificiais

- Diferentes tipos de neurônios, a quantidade de camadas e de neurônios em cada uma delas e a forma como eles estão conectados, resultam em topologias diferentes



Topologias de redes neurais artificiais

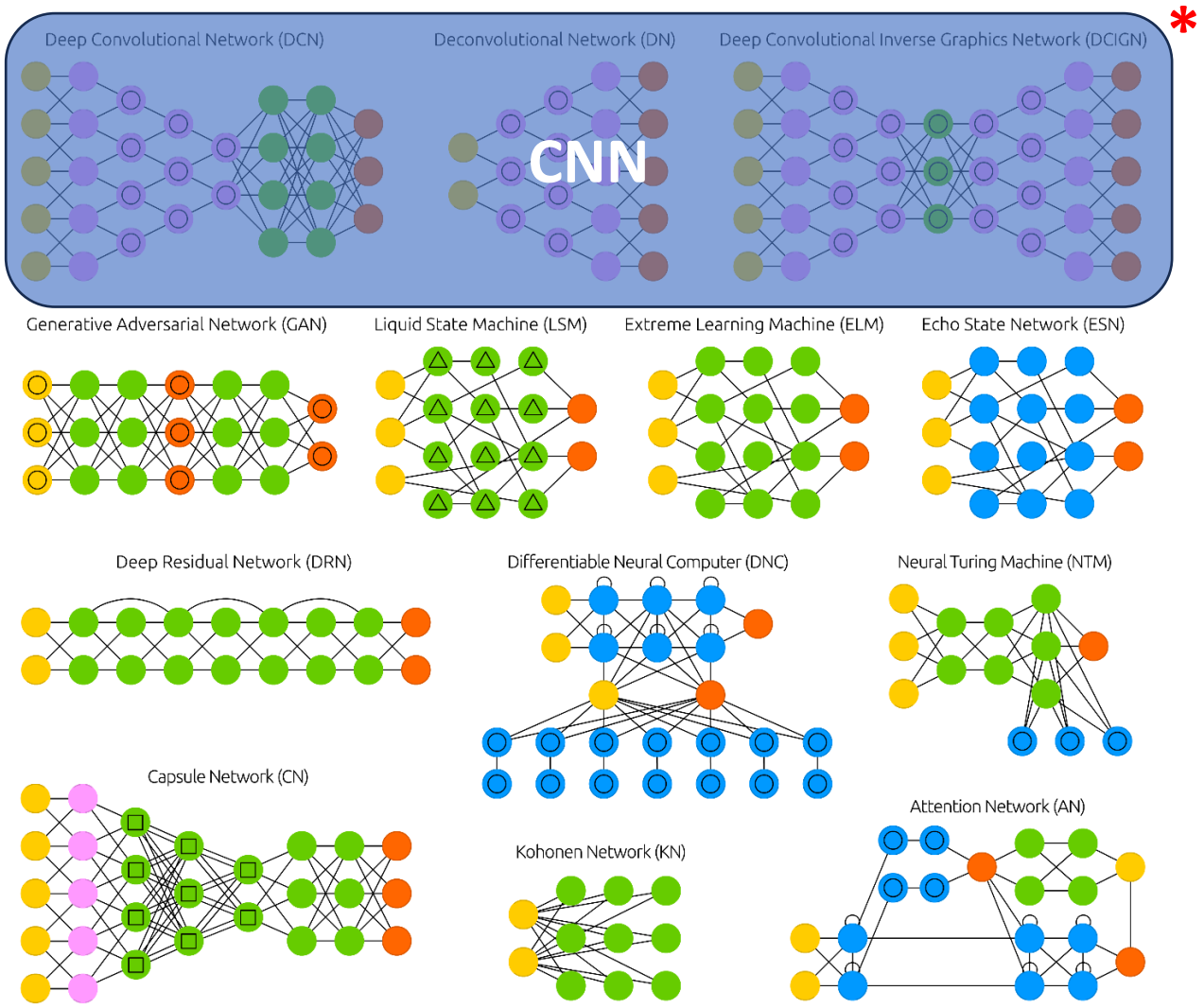


https://www.asimovinstitute.org/neural-network-zoo/

- **DFFs**: também chamadas de *dense neural networks* (DNN), são redes ***densamente*** conectadas (todas as saídas de uma camada se conectam a todos os nós da próxima) com 2 ou mais camadas ocultas.
- O termo ***feed forward*** vem do fato de que as conexões são sempre no sentido da ***entrada para a saída***.
- São usadas em tarefas de aproximação de funções (i.e., regressão e classificação.

* Modelo que será muito usado no curso.

Topologias de redes neurais artificiais

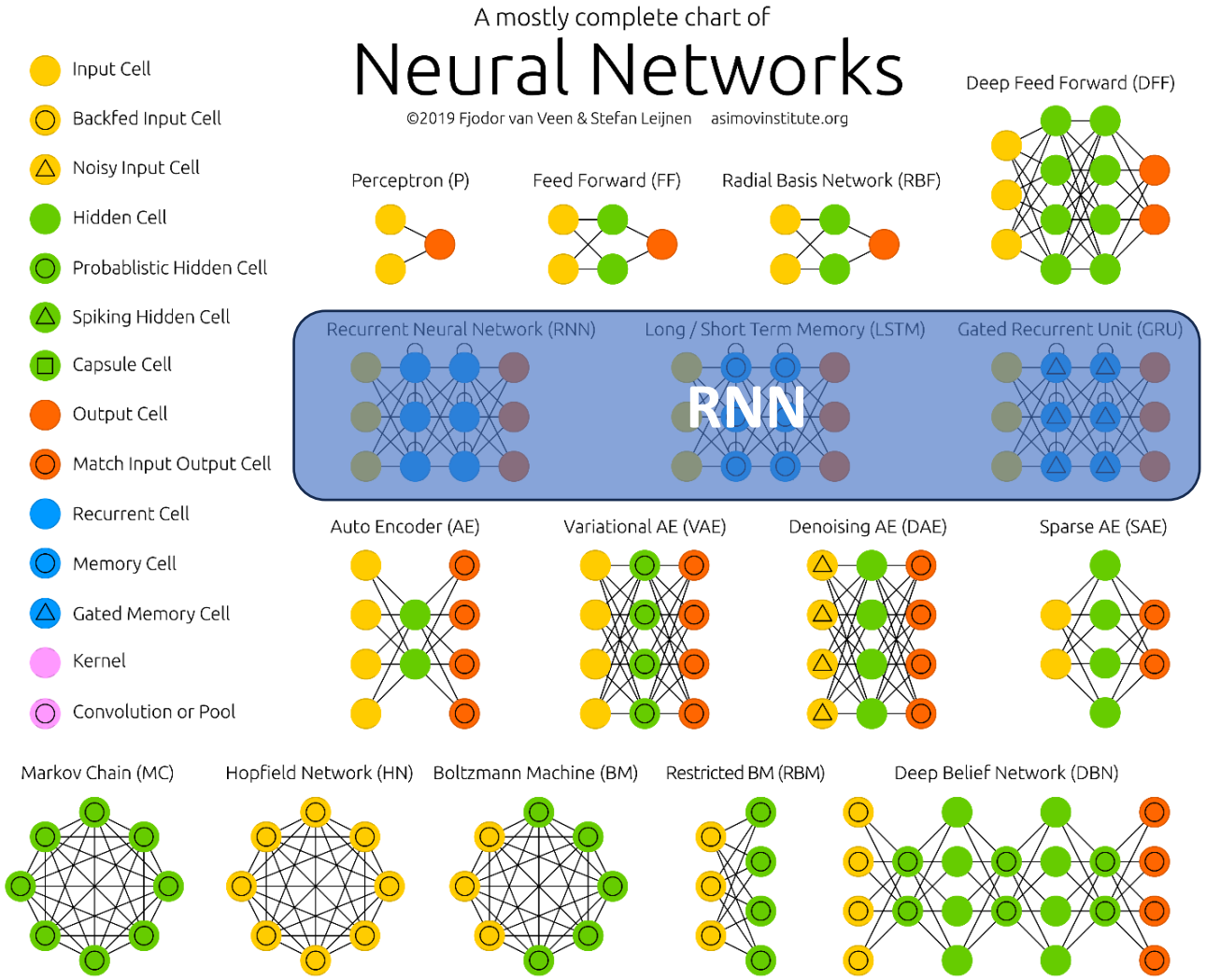


- **CNNs**: ou redes neurais convolucionais, são redes que utilizam ***camadas de convolução***, que aplicam ***filtros*** para ***extrair características relevantes*** dos dados de entrada, em geral imagens.
- São usadas em aplicações de ***visão computacional***, como classificação de imagens, processamento de vídeos, detecção de objetos em imagens ou vídeos.

https://www.asimovinstitute.org/neural-network-zoo/

* Modelo que será muito usado no curso.

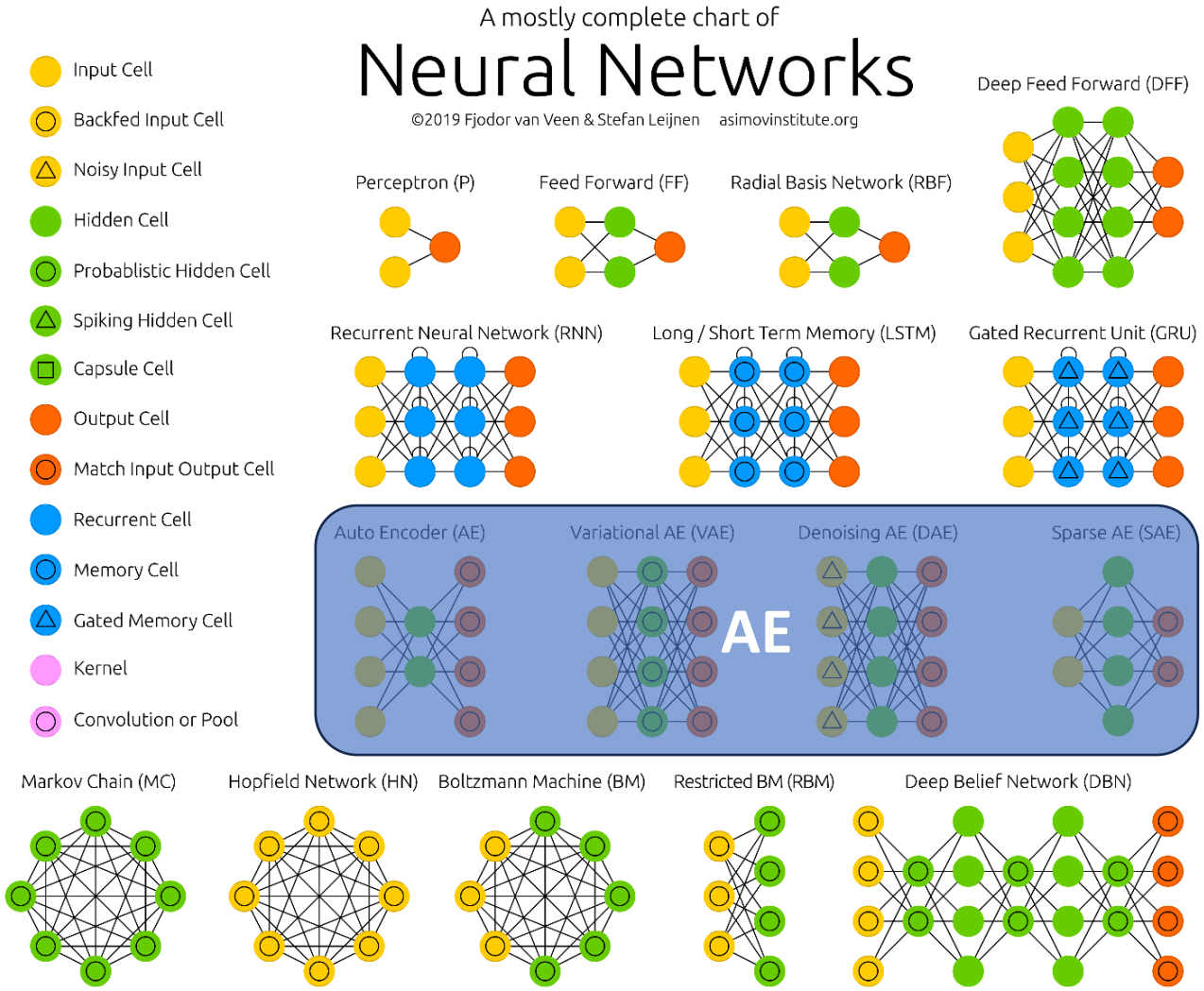
Topologias de redes neurais artificiais



- **RNNs**: ou redes neurais ***recorrentes***, possuem ***conexões que formam loops***, permitindo que ***informações anteriores sejam armazenadas e influenciem as entradas futuras***, ou seja, elas possuem memória.
- Essa capacidade as torna especialmente úteis em ***tarefas*** que envolvem ***dados sequenciais***, como ***análise de texto***, ***previsão de séries temporais*** e ***processamento de fala e áudio***.

https://www.asimovinstitute.org/neural-network-zoo/

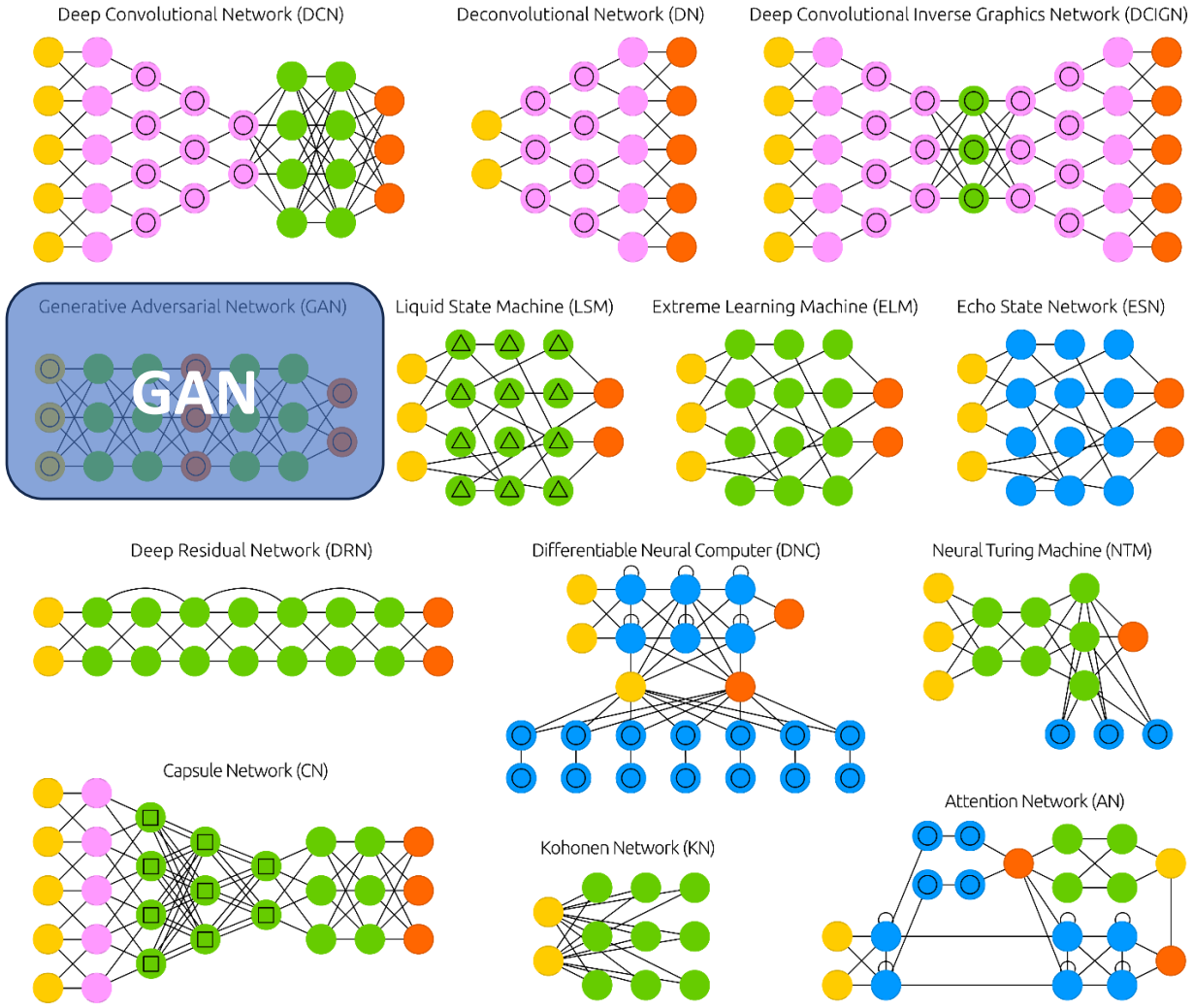
Topologias de redes neurais artificiais



https://www.asimovinstitute.org/neural-network-zoo/

- **AEs**: os *autoencoders* têm como objetivo aprender uma ***representação latente*** (i.e., características importantes, mas ocultas) dos dados de entrada.
- Consistem em duas partes: o ***codificador***, que mapeia os dados de entrada para uma representação latente (de maior ou menor dimensão), e o ***decodificador***, que reconstrói os dados originais a partir da representação latente.
- São usados em tarefas de redução de dimensionalidade, remoção de ruído, compressão de dados, geração de dados sintéticos e redundância.

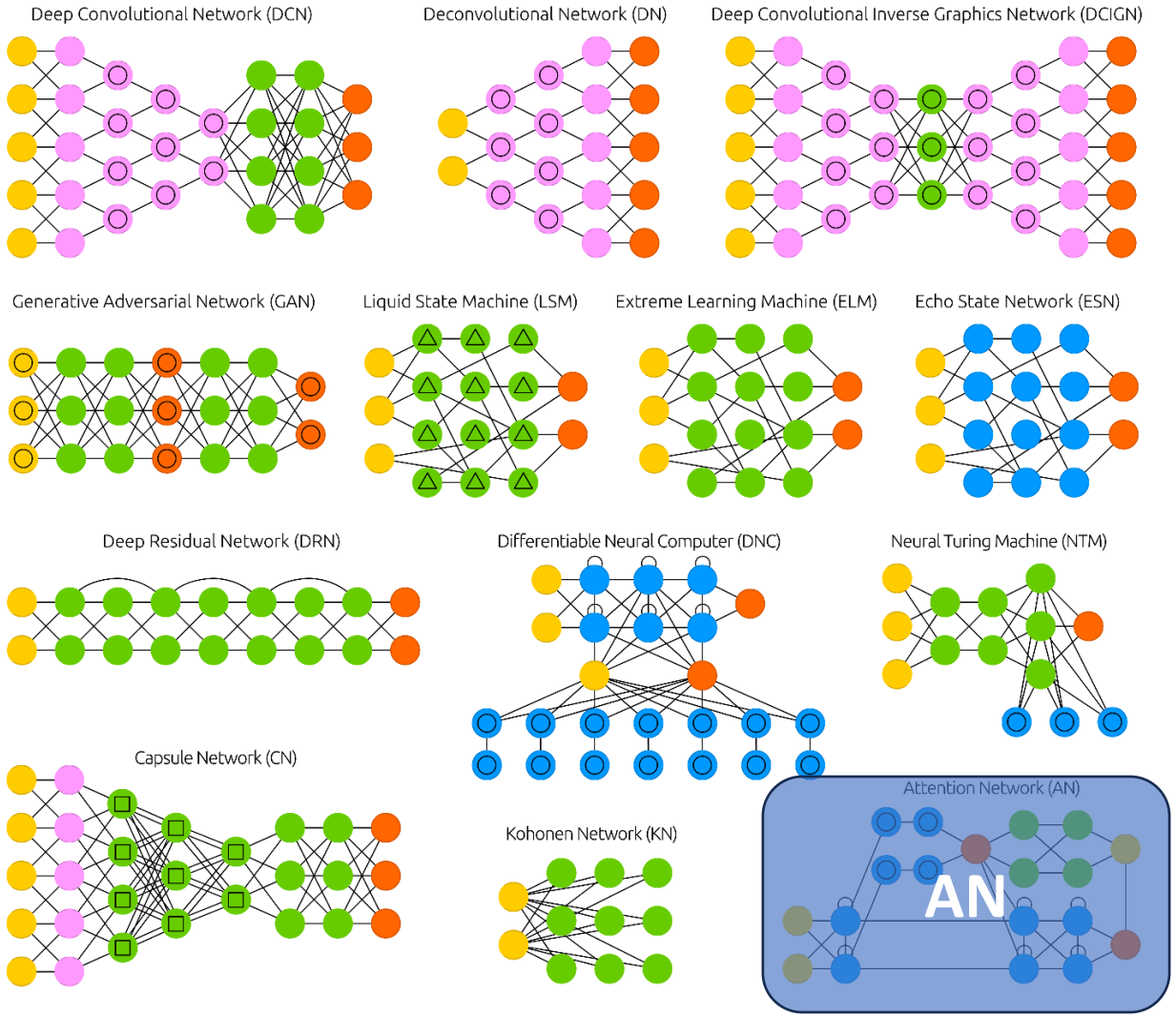
Topologias de redes neurais artificiais



- **GANs**: ou redes adversárias generativas, são um tipo especial de rede neural que consiste em ***duas redes em competição***: o ***gerador*** e o ***discriminador***.
- O gerador cria dados sintéticos que se assemelham aos dados reais, enquanto o discriminador tenta distinguir entre os dados reais e os dados gerados pelo gerador.
- São usadas para geração de imagens, vídeos e sons realistas, e em tarefas de geração de dados sintéticos.

https://www.asimovinstitute.org/neural-network-zoo/

Topologias de redes neurais artificiais



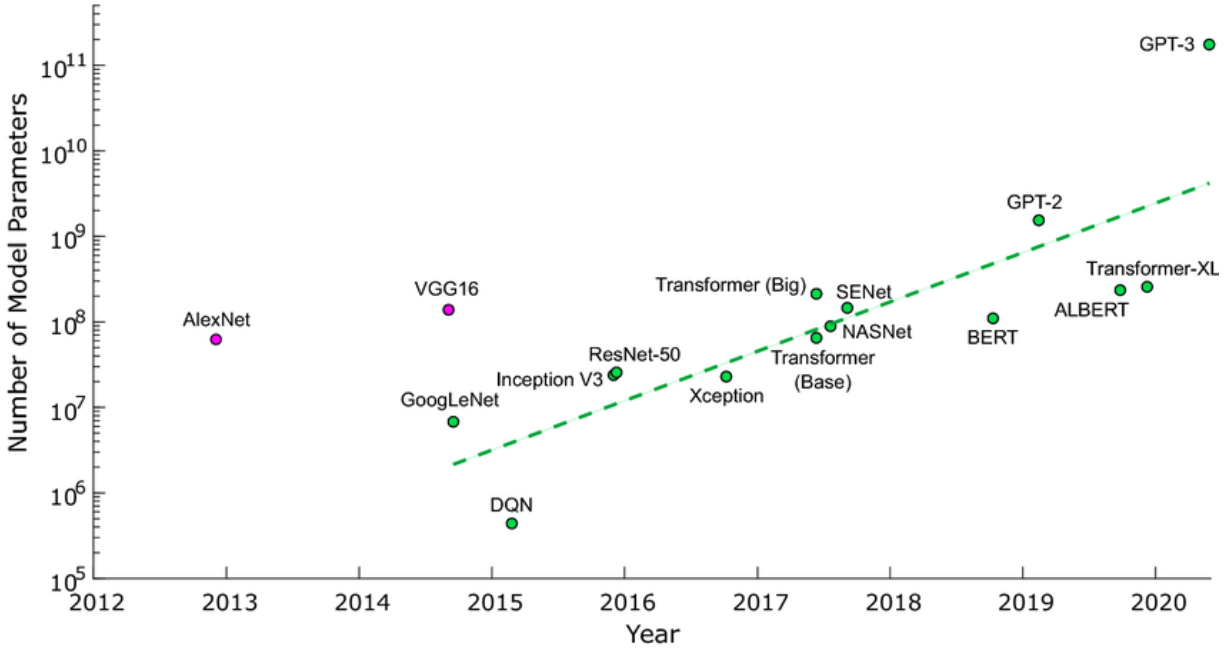
- **ANs**: ou redes de atenção, são um tipo de rede neural que se concentra em destacar partes importantes dos dados de entrada, dando-lhes maior peso durante o processamento.
- Elas atribuem pesos diferentes às partes da entrada e as combinam de forma ponderada, focando nas partes mais importantes e ignorando as menos importantes.
- São usadas em tarefas que envolvem sequências de dados, como tradução automática, processamento de linguagem natural e reconhecimento de fala.

https://www.asimovinstitute.org/neural-network-zoo/

Crescimento do tamanho dos modelos

- Ao longo dos anos, para resolver problemas mais e mais complexos com ótimos resultados, as topologias de redes neurais têm se tornado maiores e mais complexas.
  - Complexidade ou capacidade da rede neural está relacionada com sua quantidade de camadas e nós.
- A ***complexidade computacional*** de uma rede neural é ***diretamente proporcional ao número de conexões***, portanto, quanto mais camadas e nós, mais memória e cálculos matemáticos são necessários e, consequentemente, maior será o ***consumo de memória e energia***.
- Até recentemente, não havia uma preocupação com as eficiências computacional e energética de soluções envolvendo IA.

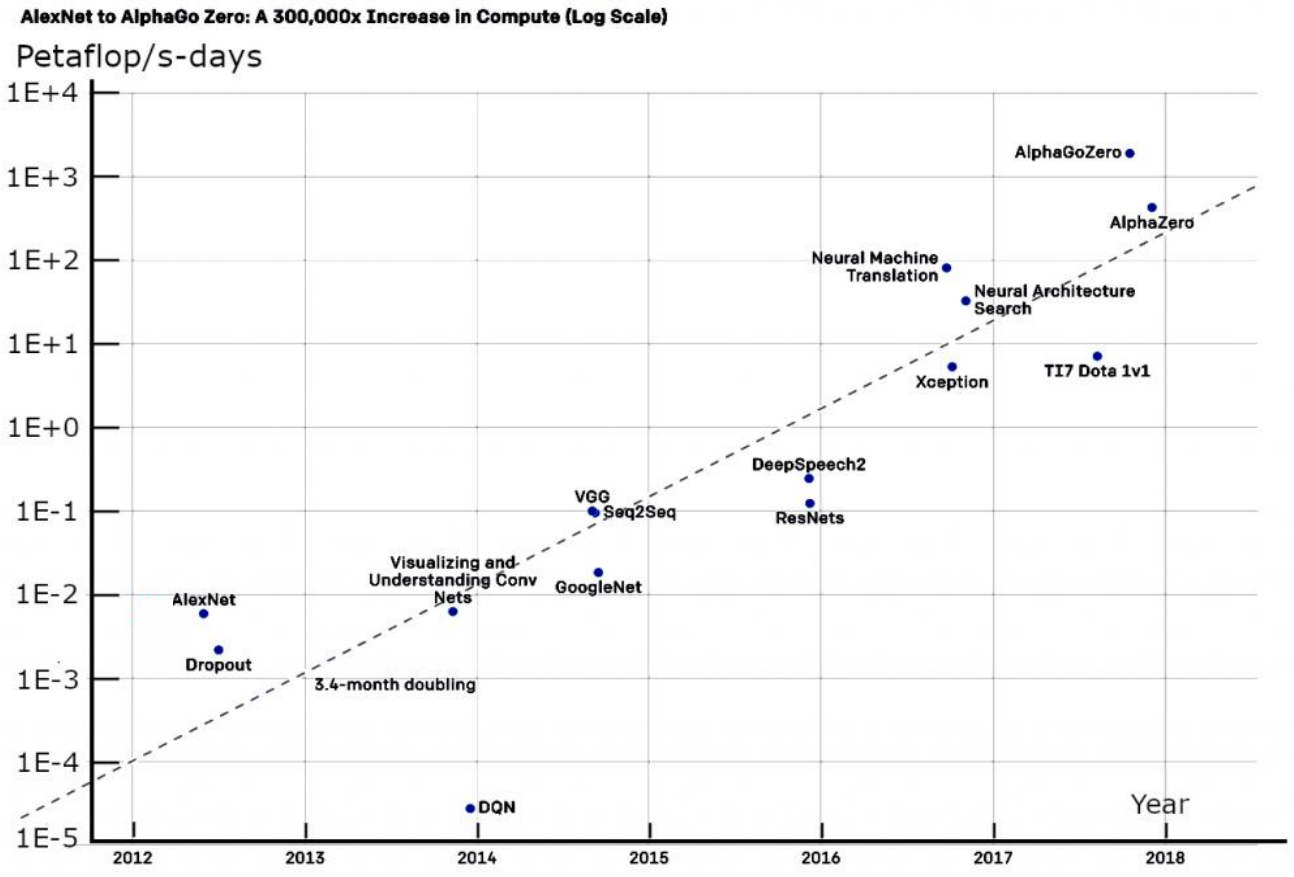
Crescimento do tamanho dos modelos



Se cada parâmetro corresponde a uma variável do tipo *float* (4 bytes), precisamos de $1.76 \times 10^{12} \times 4 \sim 7 \times 10^{12}$ bytes, ou seja, **7 Terabytes** para armazenar o modelo GPT-4!

- Os modelos não param de crescer!
- Isso se deve ao aumento da disponibilidade de dados, ao aumento do poder de computação e ao desenvolvimento de novas técnicas de aprendizado profundo.
- Vejamos o modelo de linguagem GPT:
  - O GPT-2 tinha aproximadamente 1.5 bilhão de parâmetros (i.e., pesos sinápticos).
  - Já os GPT-3/3.5 têm aproximadamente 175 bilhões de parâmetros.
  - E estima-se que o GPT-4 tenha 1.76 trilhão de parâmetros.

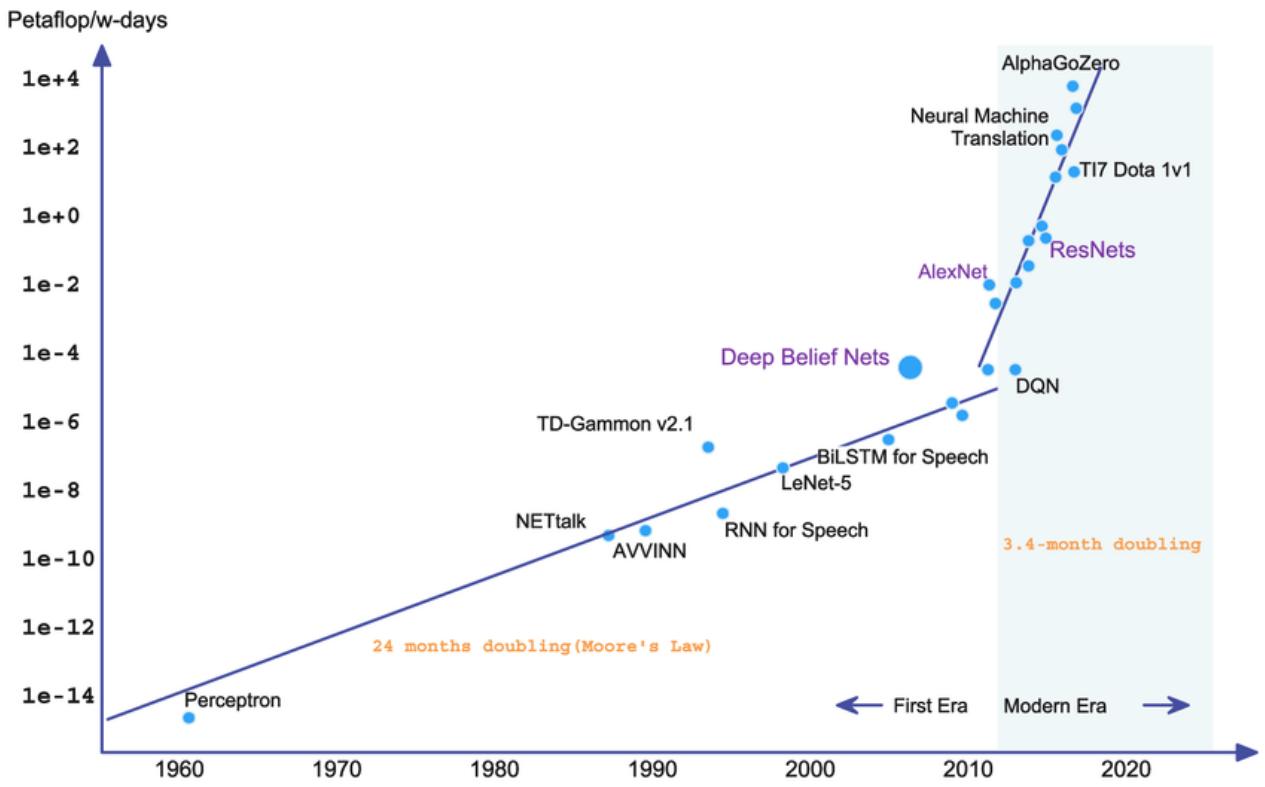
Necessidades computacionais (2012 - Atual)



flops: quantidade de operações em ponto flutuante executadas por um processador em um segundo.

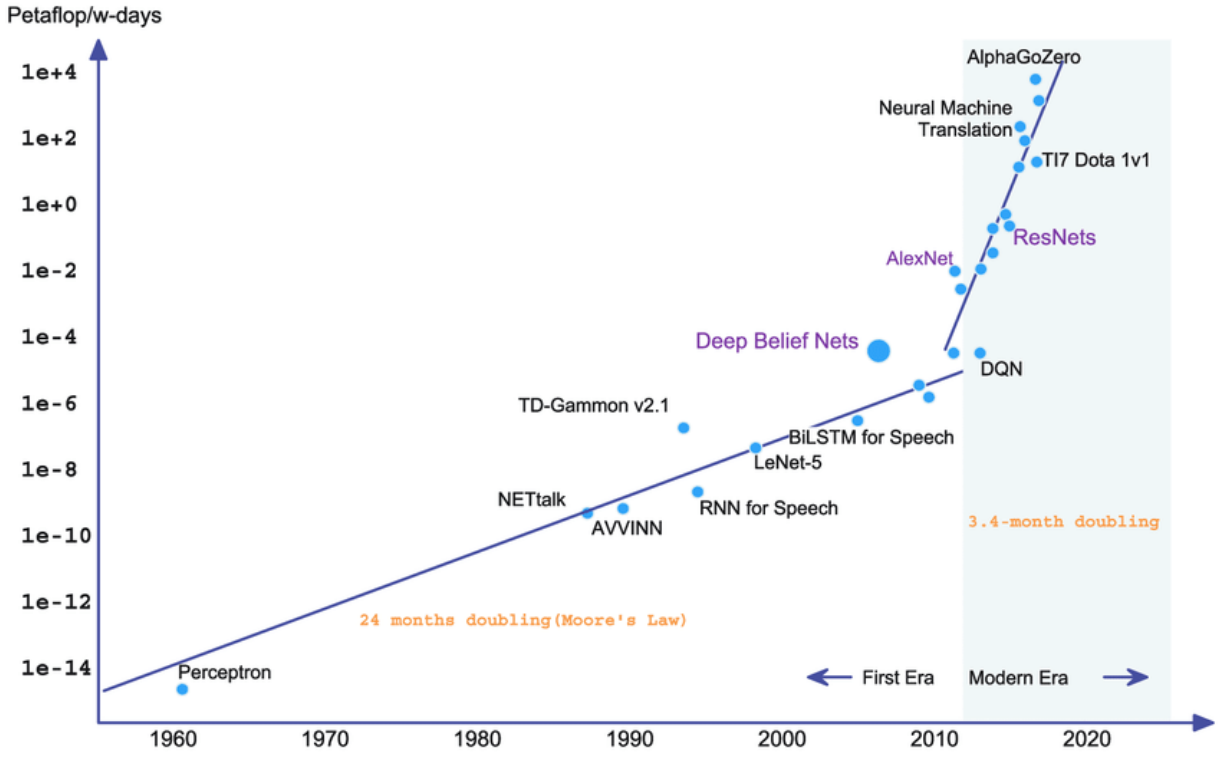
- Nos últimos anos, o poder computacional necessário para treinar os modelos de ML amplamente usados hoje teve que crescer 300.000 vezes.
- Essa tendência de redes imensas começou em 2012 com a AlexNet:
  - CNN que bateu com folga todos os recordes anteriores do desafio *ImageNet Large Scale Visual Recognition* de 2012.
  - Têm 60 milhões de parâmetros, o que na época a tornou uma das maiores e mais complexas redes.
  - Popularizou o uso de camadas convolucionais, impulsionando a revolução do aprendizado profundo.
  - Foi uma das primeiras CNNs a usar GPUs para reduzir o tempo de treinamento.

Necessidades computacionais (desde 1958)



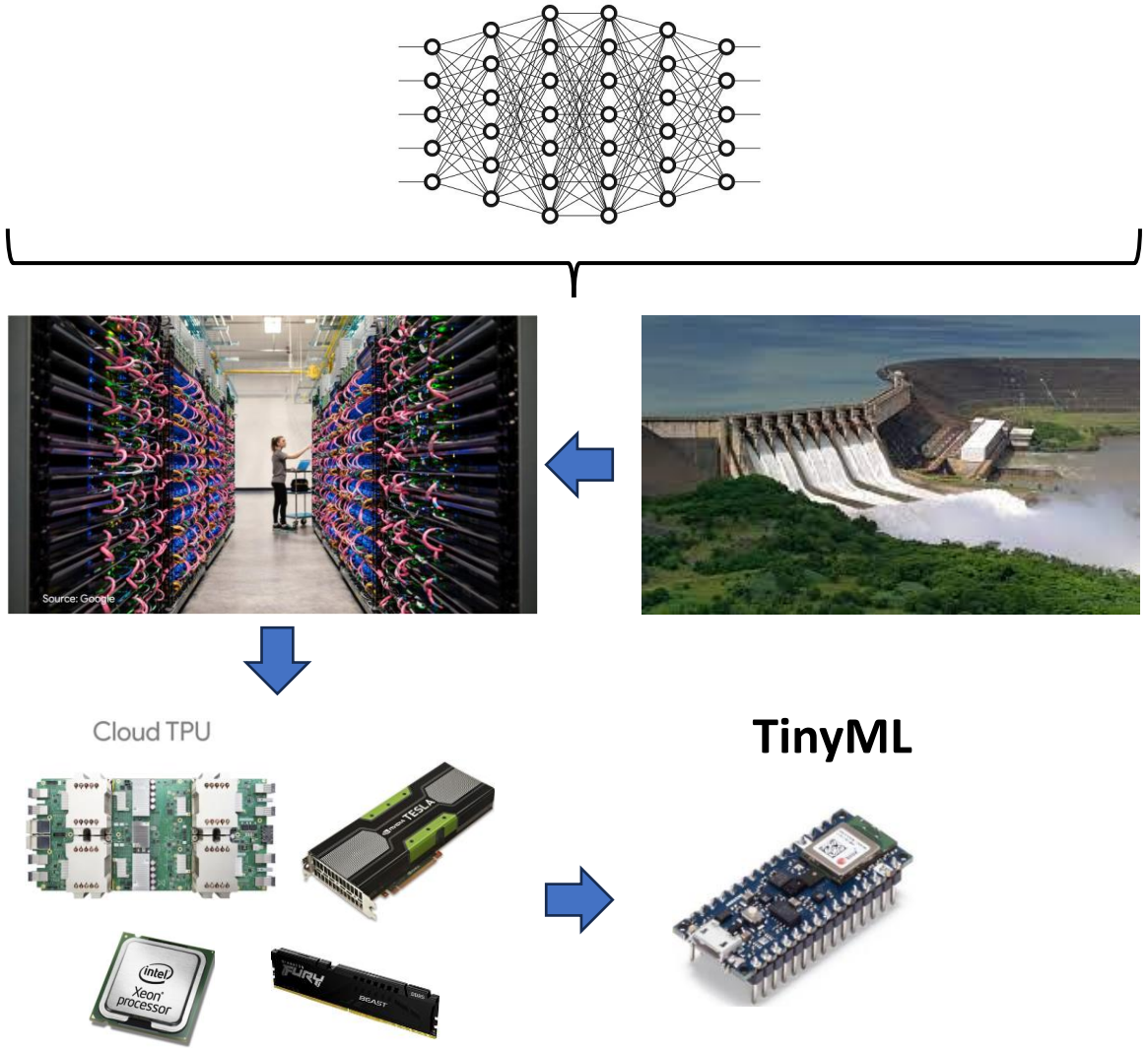
- Nos últimos 10 anos, a quantidade de cálculos necessários cresceu extraordinariamente rápido.
- Na “primeira era” do ML, a quantidade de cálculos e, consequentemente, o tamanho dos modelos, dobrava a cada dois anos aproximadamente.
- Na “era moderna”, os requisitos de computação praticamente dobram a cada 3/4 ***meses***.

Necessidades computacionais (desde 1958)



- O que aconteceu nesses últimos 10 anos que explica esse boom?
  - Disponibilidade de grandes volumes de dados (de tera a petabytes) devido à internet.
  - Aumento do poder de computacional através de GPUs, FPGAs e CPUs com múltiplos cores.
  - Surgimento de novos algoritmos e modelos de aprendizado de máquina, como redes neurais profundas, redes adversárias generativas (GANs), redes de atenção, *transformers*, *deep reinforment-learning*, etc.
  - Disponibilidade de *frameworks* e bibliotecas amigáveis, como TensorFlow e PyTorch, que facilitam o desenvolvimento de soluções com ML.
  - O surgimento de serviços de computação em nuvem ofereceu acesso econômico e escalável a recursos de computação.

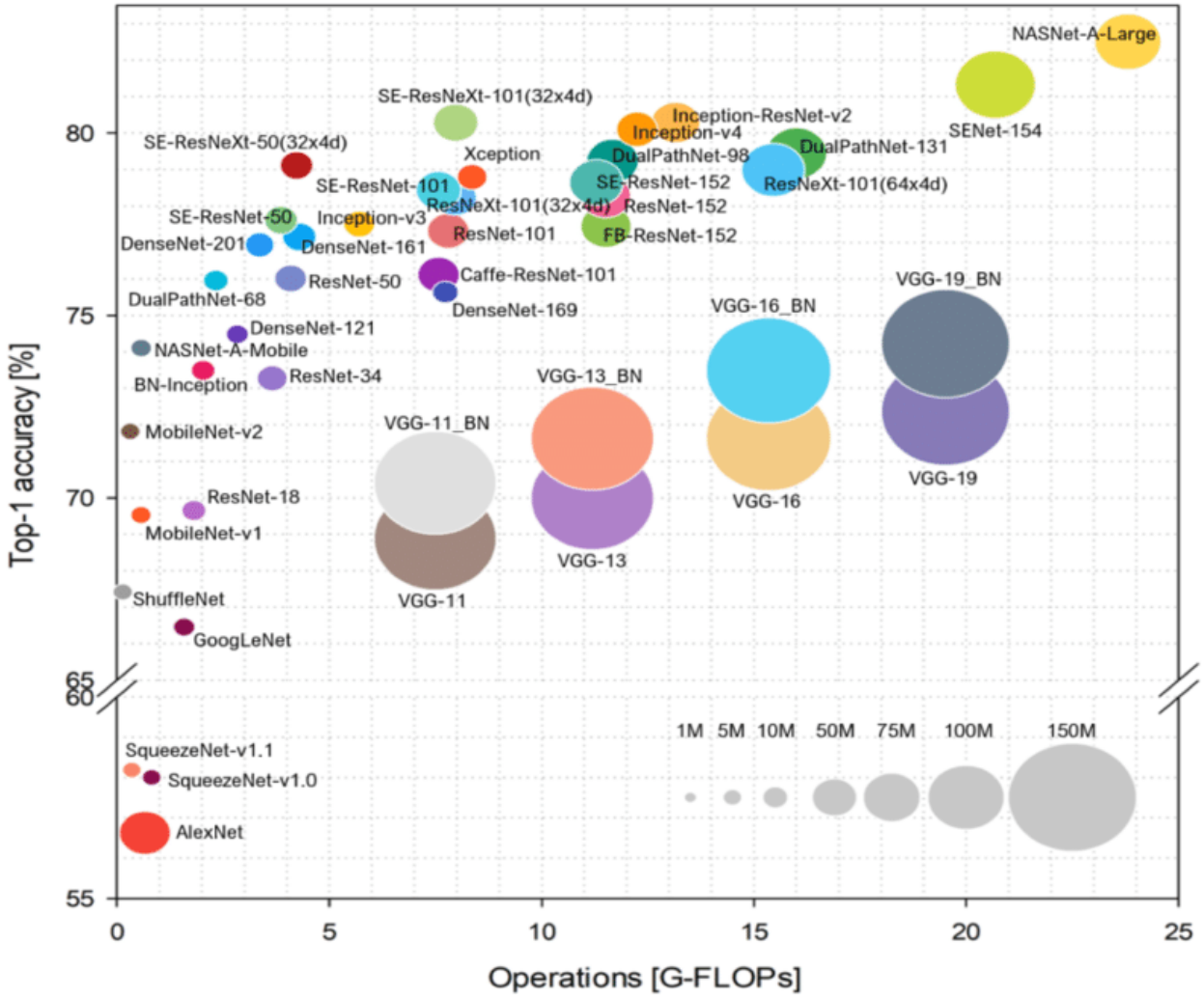
Consequências do aumento da capacidade



- Entretanto, esses modelos mais e mais complexos, necessitam, consequentemente, de muito mais capacidade de armazenamento, energia, dispositivos de processamento mais poderosos e muito mais caros e que ocupam grandes espaços.
- Como podemos colocar tudo isso em um dispositivo tinyML?

TPU: *Tensor Processing Unit*, são GPUs altamente especializadas para o treinamento e inferência de modelos de aprendizado de máquina, especialmente aqueles que usam o TensorFlow.

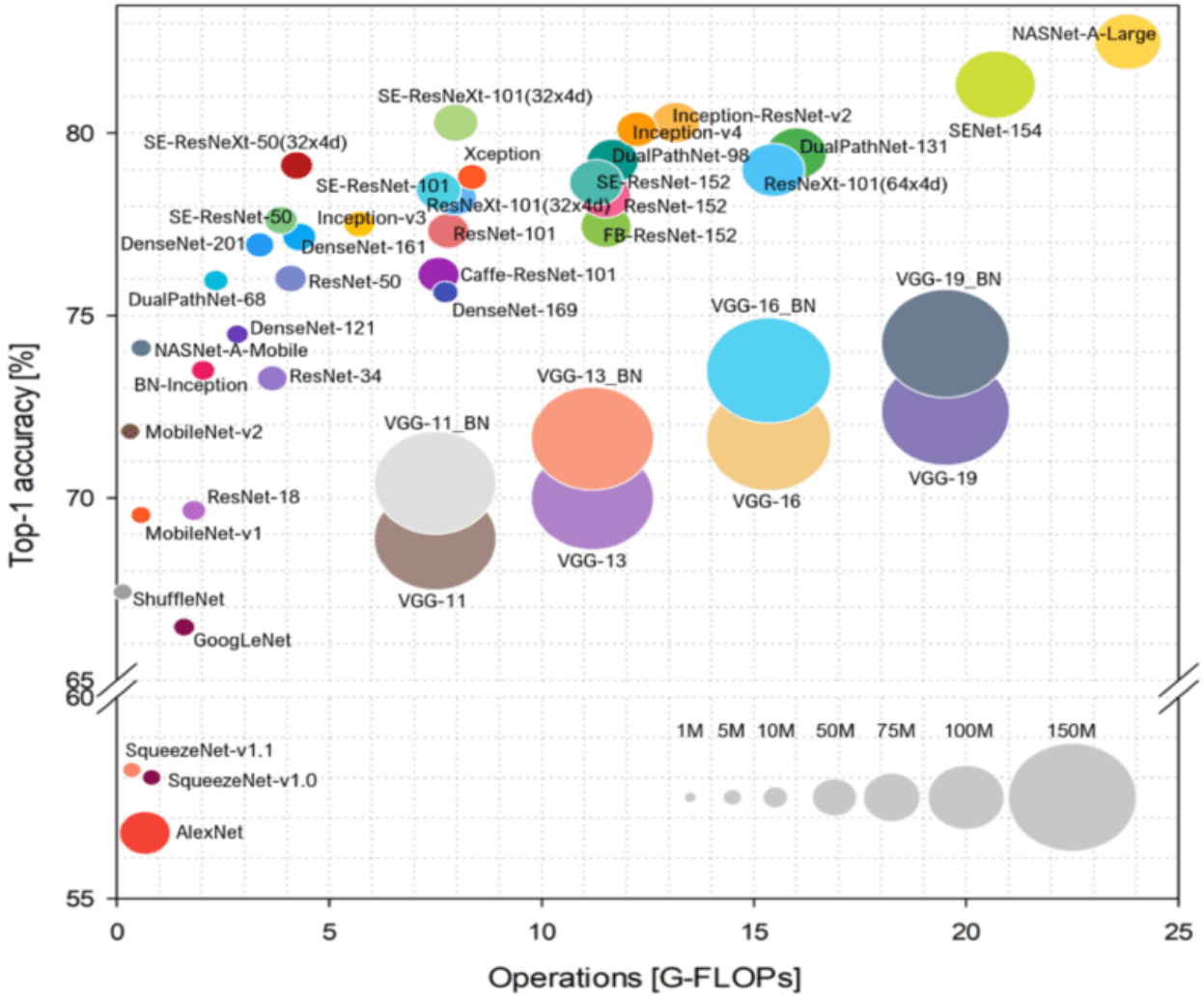
Evolução dos modelos de ML



- A figura ao lado mostra a evolução dos modelos de ML na tarefa de classificação de imagem usando o conjunto de dados ImageNet-1K (1000 classes) como *benchmark*.
- O eixo x mostra o ***custo computacional*** do modelo (i.e., quantidade de *flops* necessárias para uma classificação).
- O eixo y mostra a ***acurácia*** (taxa de acertos).
- O tamanho de cada círculo corresponde à ***complexidade*** (i.e., quantidade de parâmetros) do modelo.

Fonte: Bianco, Simone, et al. "*Benchmark analysis of representative deep neural network architectures*." *IEEE access* 6 (2018): 64270-64277.

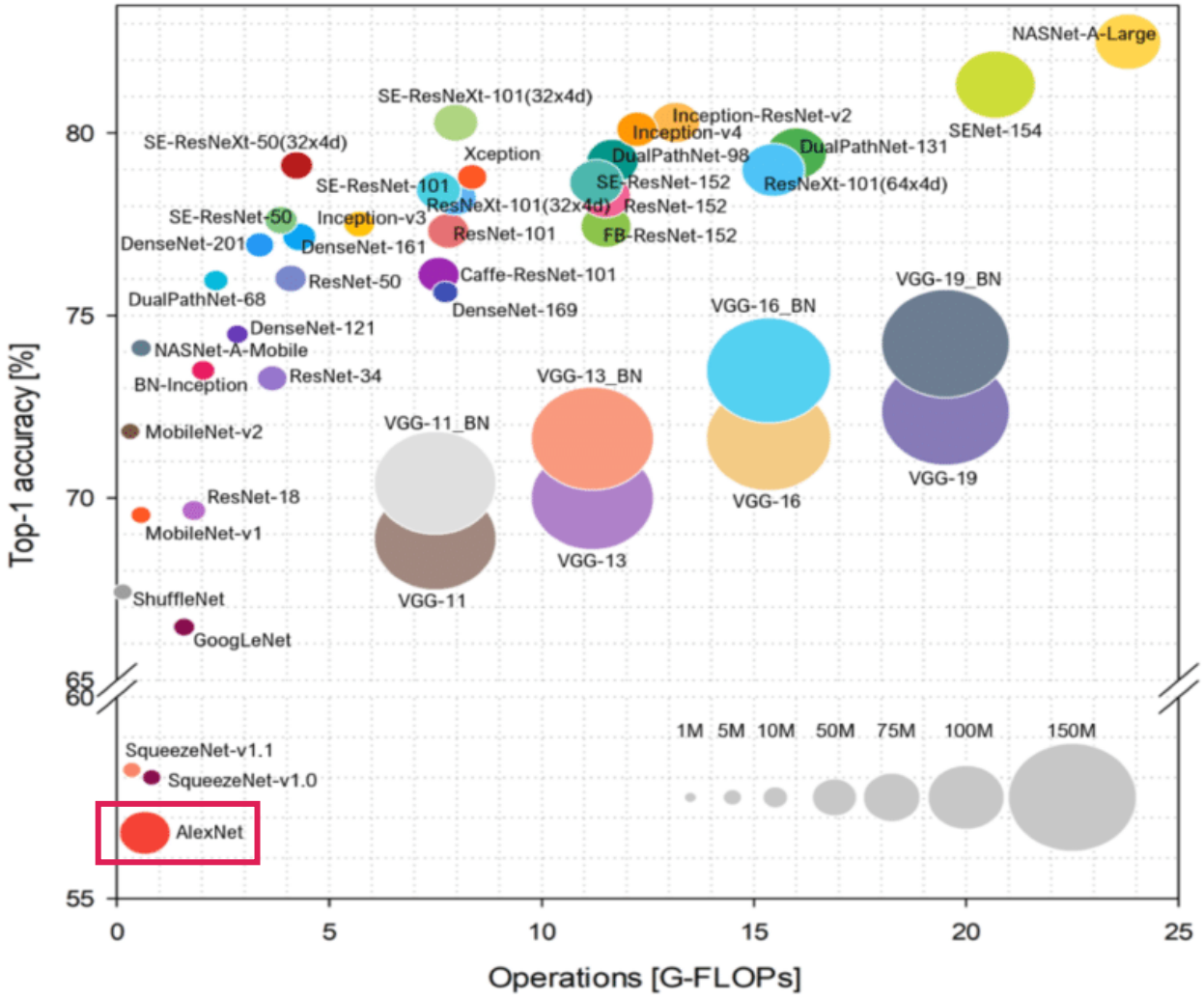
Evolução dos modelos de ML



Fonte: Bianco, Simone, et al. "*Benchmark analysis of representative deep neural network architectures*." *IEEE access* 6 (2018): 64270-64277.

- Para execução em dispositivos tinyML, devemos procurar por modelos com:
  - Poucos parâmetros (i.e., círculos pequenos), pois a quantidade de parâmetros está associada à quantidade de memória demandada pelo modelo.
  - Quantidade necessária de *flops* baixa, pois a quantidade de operações está diretamente relacionada ao consumo de energia. Além disso, a quantidade de operações do modelo deve ser suportada pelo poder computacional da CPU.
  - Alta acurácia.

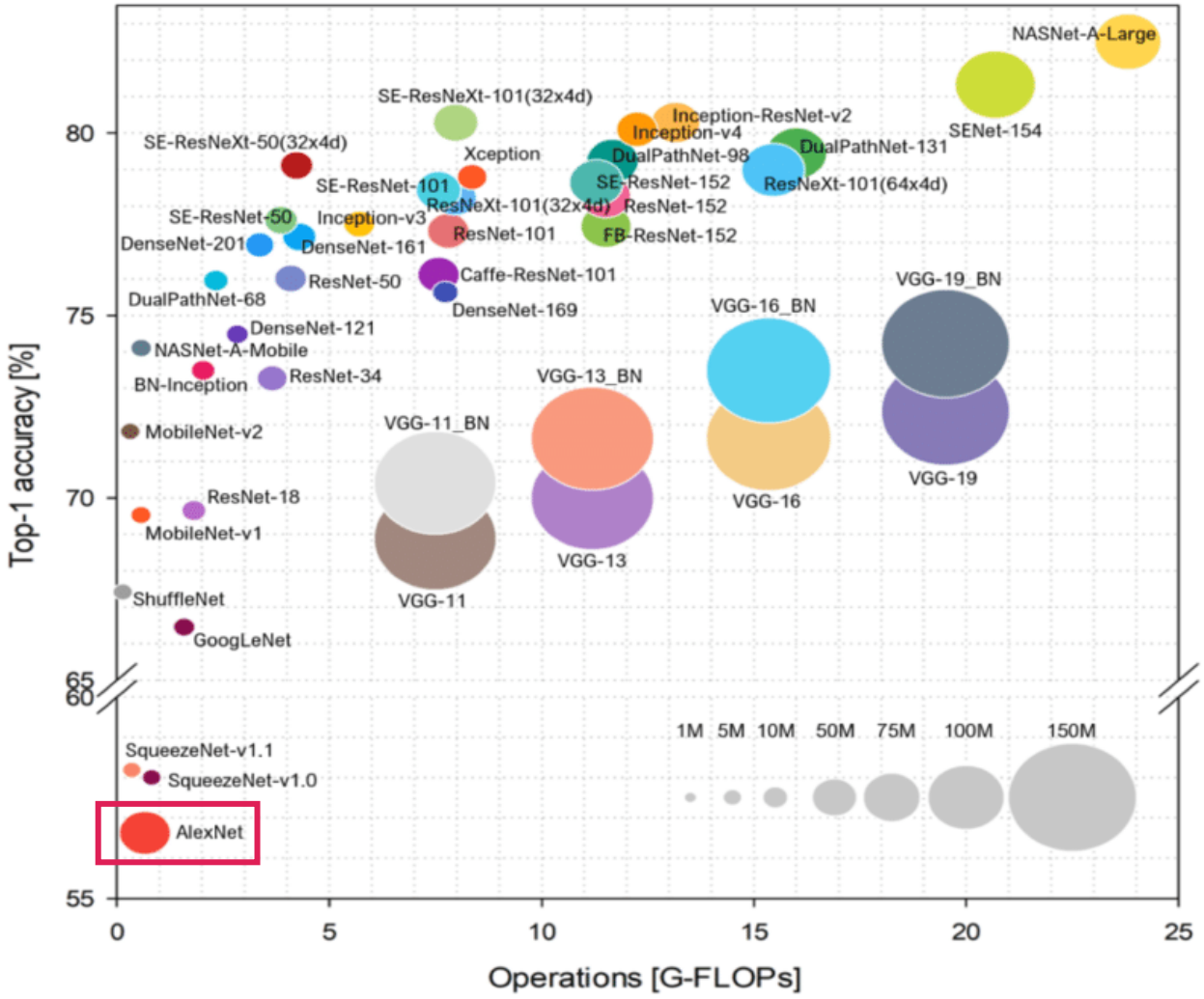
Evolução dos modelos de ML



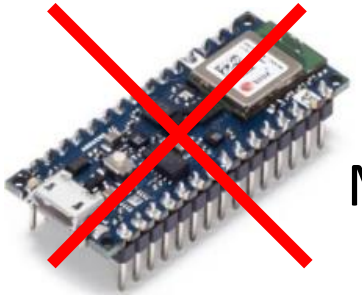
- **AlexNet** (Universidade de Toronto, Canadá - 2012)
  - Acurácia de 57.1%
  - Tamanho: 61 MB
- Camadas: 5 convolucionais e 3 densas.

Fonte: Bianco, Simone, et al. "*Benchmark analysis of representative deep neural network architectures*." *IEEE access* 6 (2018): 64270-64277.

Evolução dos modelos de ML



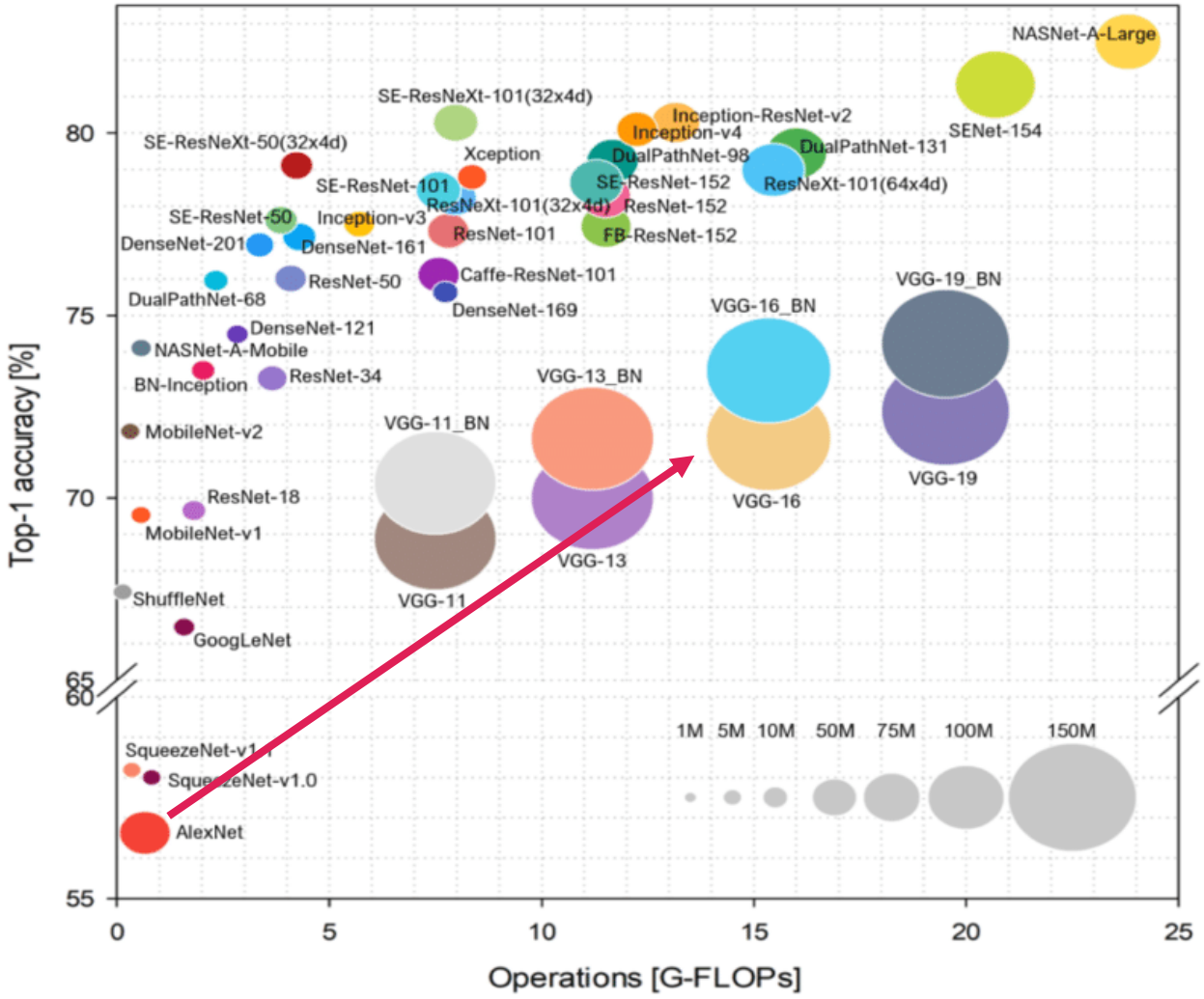
- **AlexNet** (Universidade de Toronto, Canadá - 2012)
  - Acurácia de 57.1%
  - Tamanho: 61 MB



Memória RAM: 256 KB

Fonte: Bianco, Simone, et al. "*Benchmark analysis of representative deep neural network architectures*." *IEEE access* 6 (2018): 64270-64277.

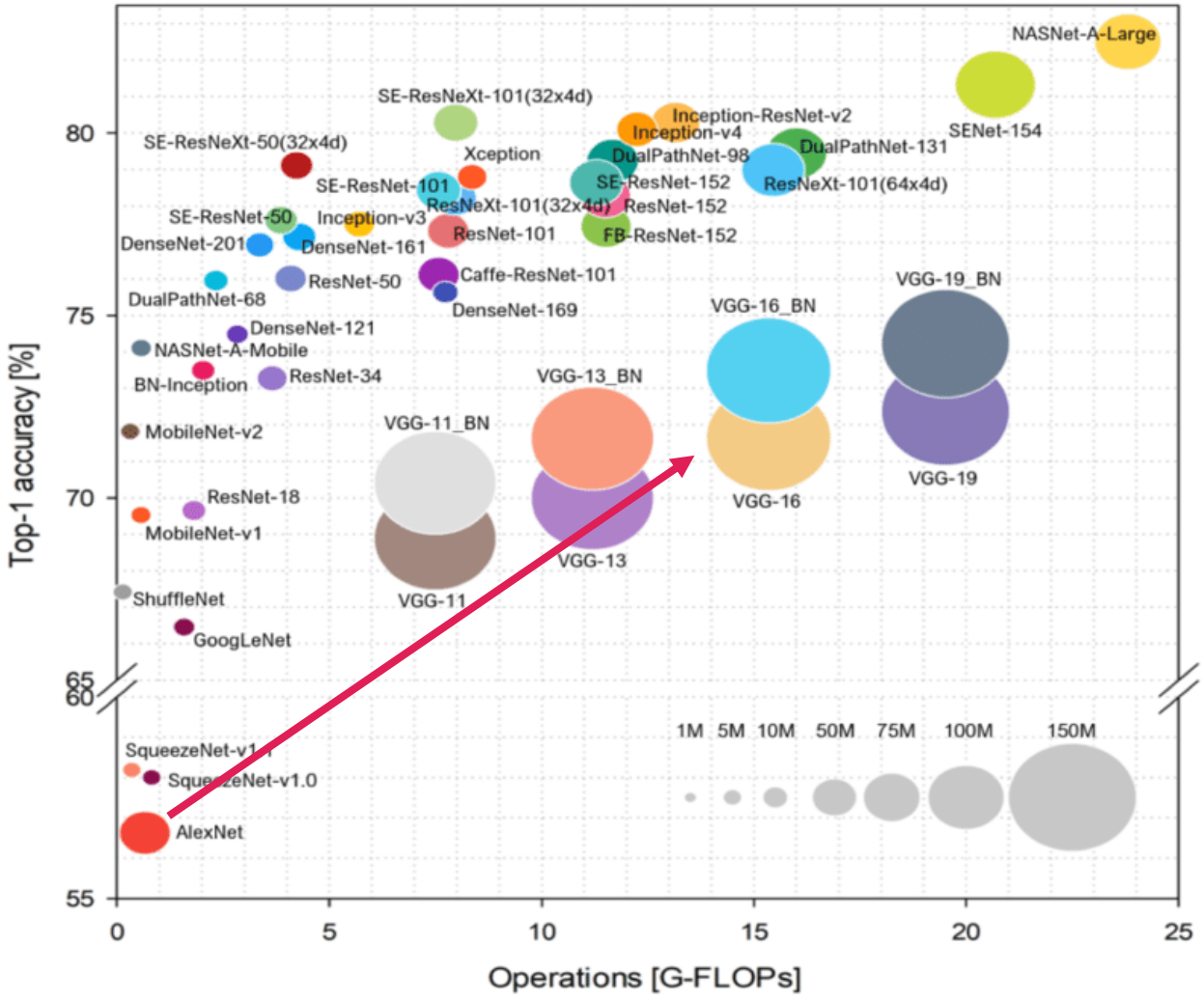
Evolução dos modelos de ML



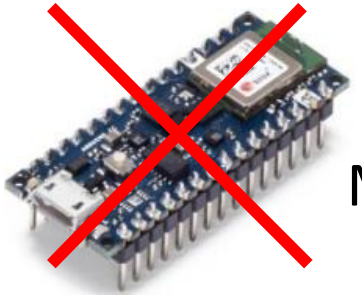
- **VGG-16** (Universidade de Oxford, Reino Unido - 2014)
  - Acurácia de 71.5%
  - Tamanho: 528 MB
- Camadas: 13 convolucionais e 3 densas.
- Em dois anos, saímos de um modelo pequeno, requerendo poucos *flops* (baixo consumo), mas com baixa acurácia para modelos mais precisos, porém mais de 8 vezes maiores e requerendo 15 vezes mais *flops* (alto consumo).

Fonte: Bianco, Simone, et al. "*Benchmark analysis of representative deep neural network architectures*." *IEEE access* 6 (2018): 64270-64277.

Evolução dos modelos de ML



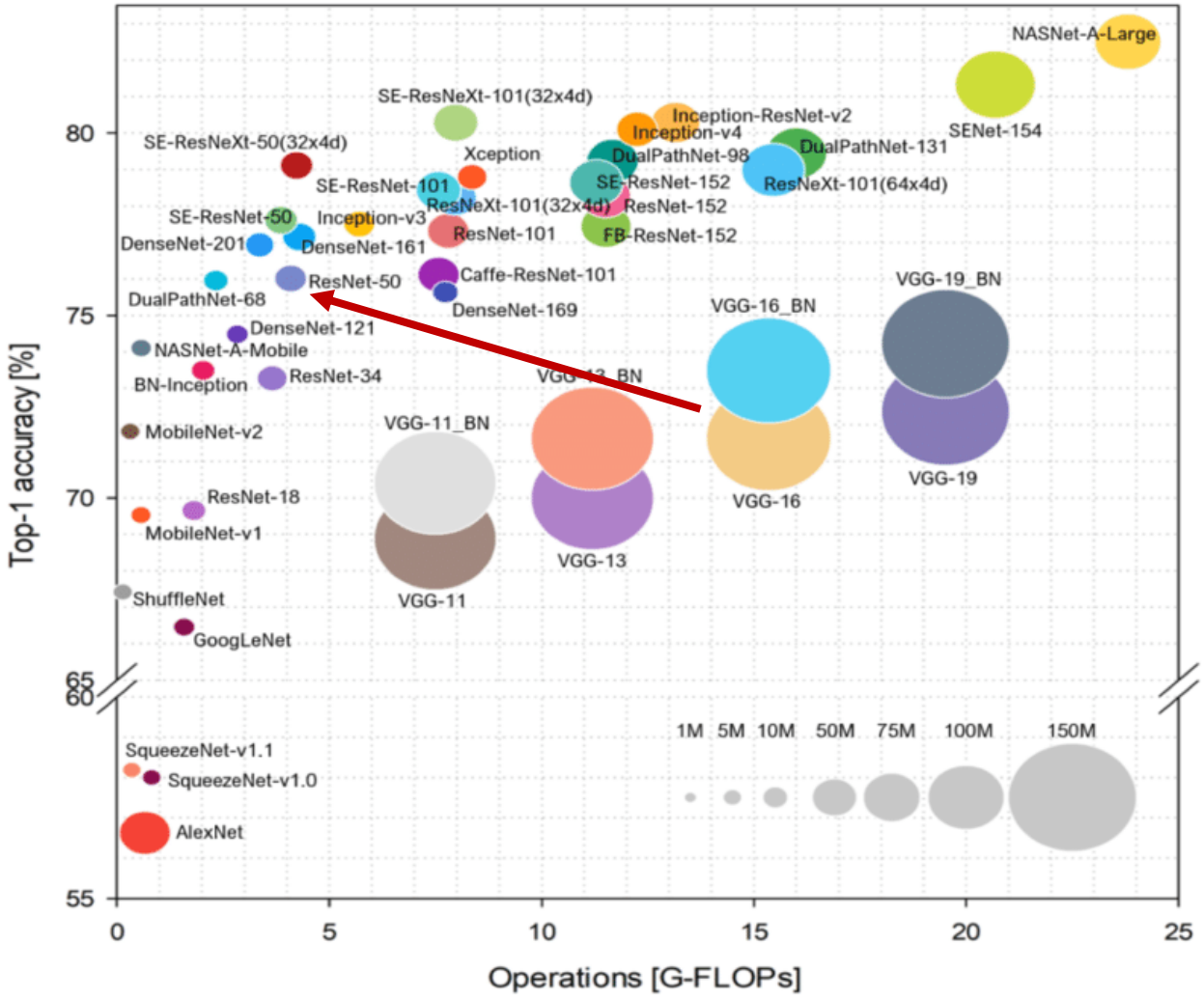
- **VGGNet** (Universidade de Oxford, Reino Unido - 2014)
  - **Acurácia de 71.5%**
  - **Tamanho: 528 MB**



Memória RAM: 256 KB

Fonte: Bianco, Simone, et al. "*Benchmark analysis of representative deep neural network architectures*." *IEEE access* 6 (2018): 64270-64277.

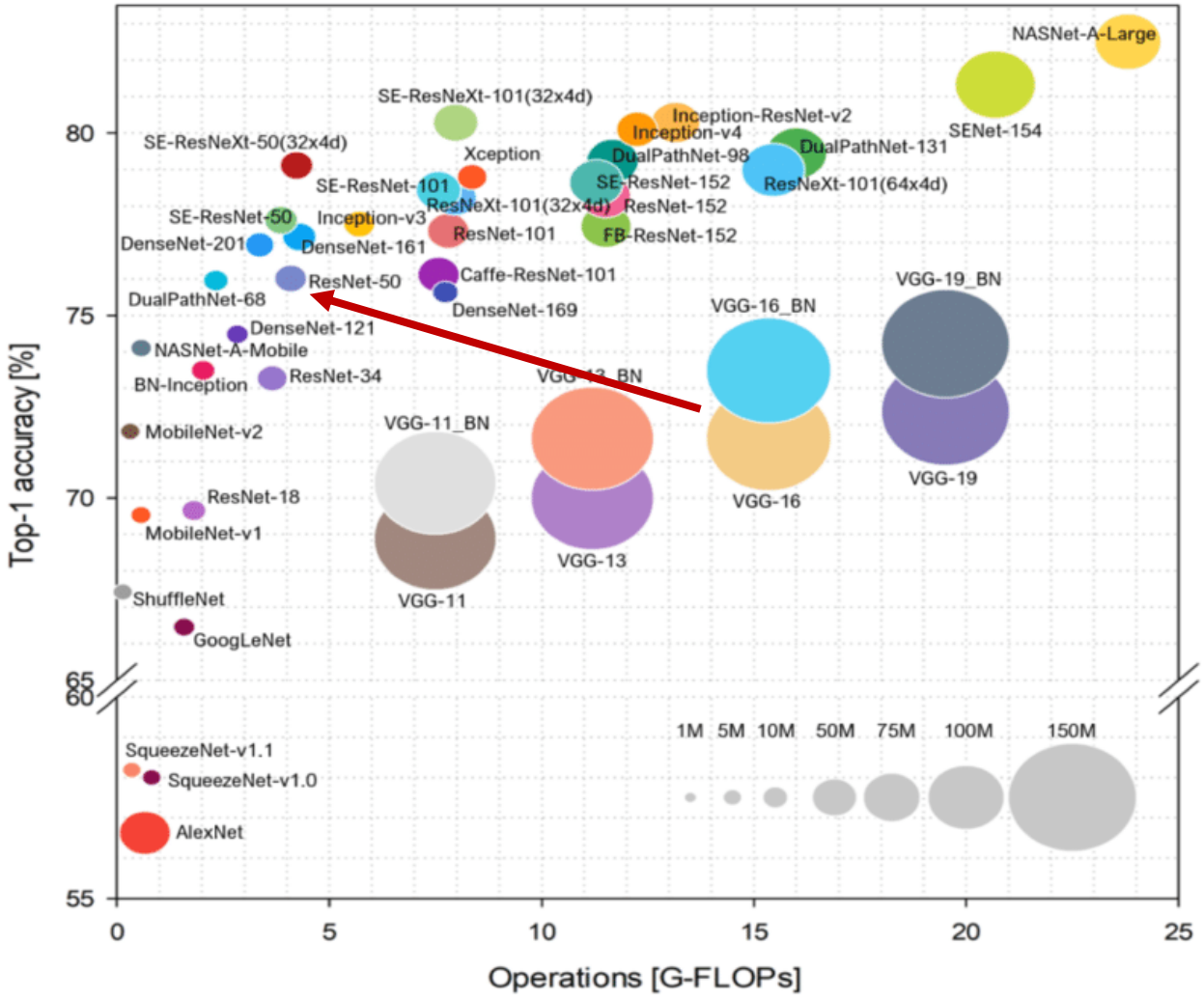
Evolução dos modelos de ML



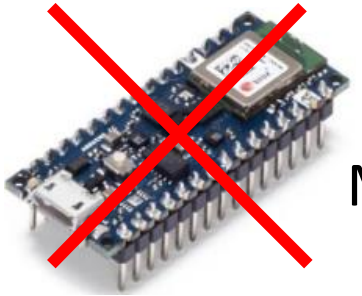
Fonte: Bianco, Simone, et al. "*Benchmark analysis of representative deep neural network architectures*." *IEEE access* 6 (2018): 64270-64277.

- **ResNet-50** (Microsoft - 2015)
  - **Acurácia de 75.8%**
  - **Tamanho: 22.7 MB**
- Camadas: 48 convolucionais, 2 de *pooling* (agrupamento) e 1 densa.
- Arquitetura muito mais eficiente do que a VGGNet pois introduz
  - ***Dropout***: desliga aleatoriamente algumas das conexões da rede durante o treinamento para evitar o sobreajuste.
  - **Blocos residuais**: são atalhos entre camadas convolucionais que mitigam o problema do desaparecimento do gradiente.
- Em um ano, saímos de um modelo extremamente grande e que requer muitos *flops*, mas razoavelmente preciso para um modelo 23 vezes menor, mais preciso e necessitando de 3 vezes menos *flops*.

Evolução dos modelos de ML



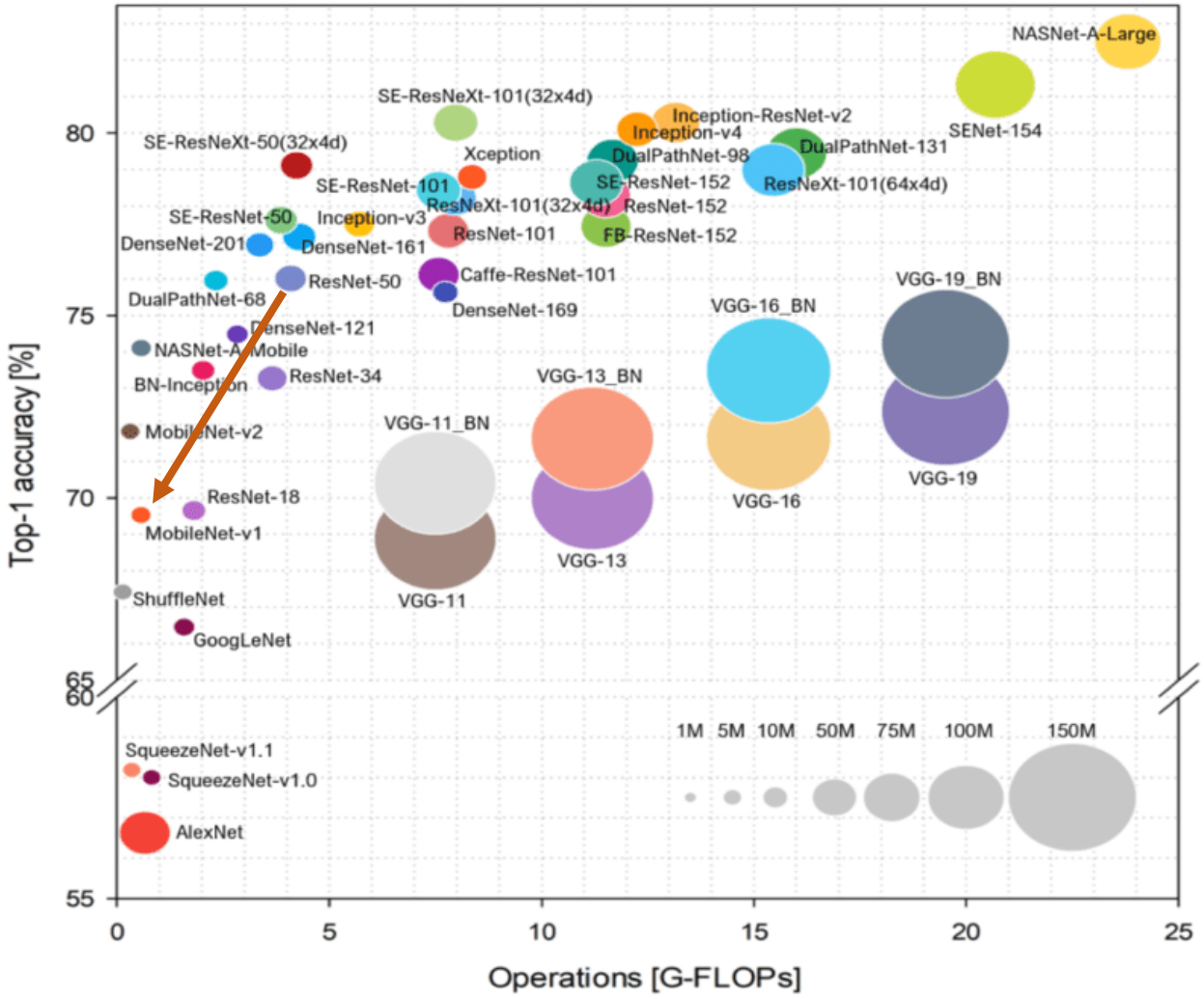
- **ResNet-50** (Microsoft - 2015)
  - **Acurácia de 75.8%**
  - **Tamanho: 22.7 MB**



Memória RAM: 256 KB

Fonte: Bianco, Simone, et al. "*Benchmark analysis of representative deep neural network architectures*." *IEEE access* 6 (2018): 64270-64277.

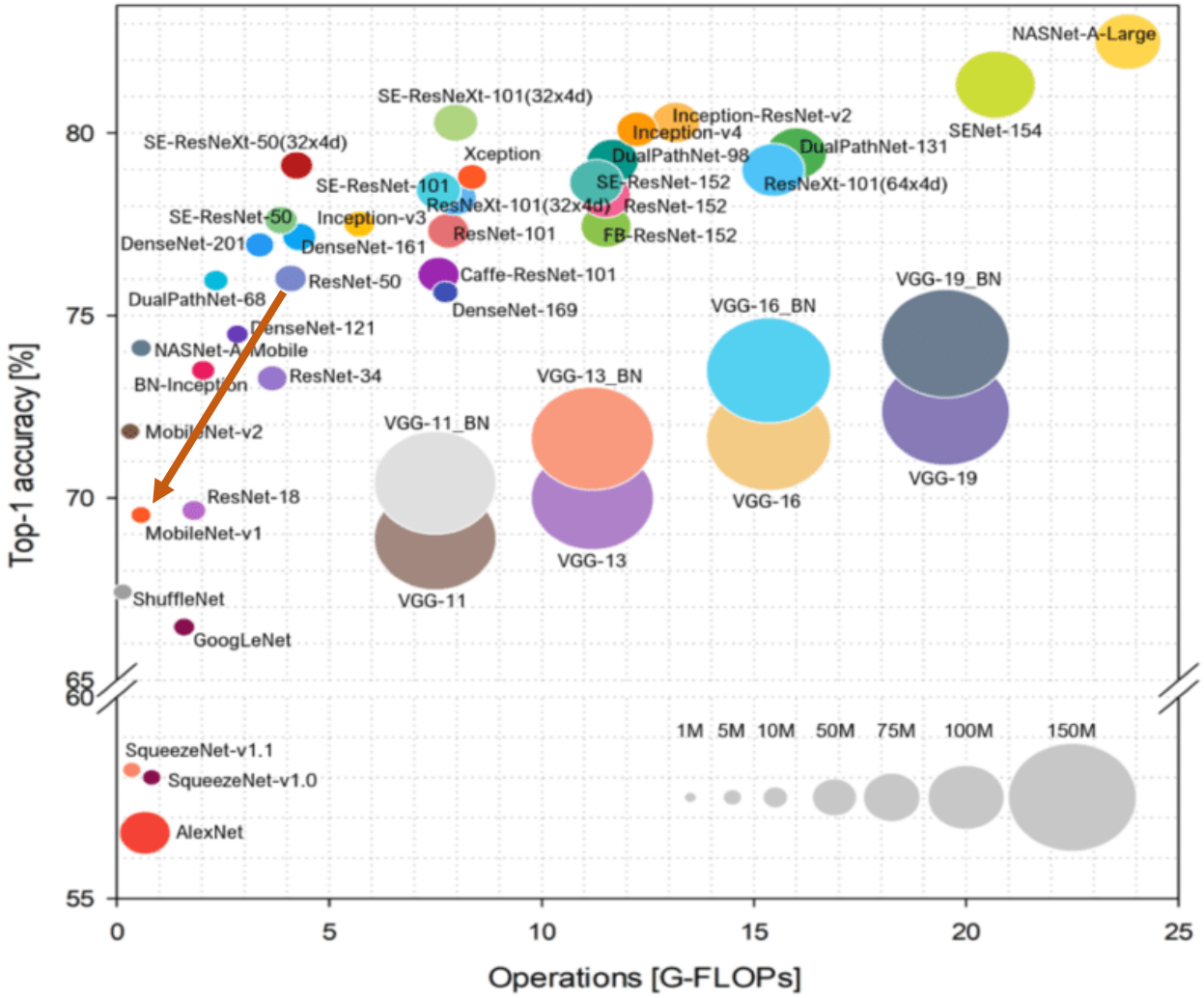
Evolução dos modelos de ML



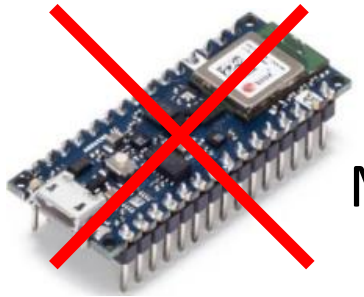
Fonte: Bianco, Simone, et al. "*Benchmark analysis of representative deep neural network architectures*." *IEEE access* 6 (2018): 64270-64277.

- **MobileNetv1** (Google - 2015)
  - **Acurácia de 70.6%**
  - **Tamanho: 16.9 MB**
- Camadas: 27 convolucionais e 2 densas.
- Adequado para aplicações de visão computacional em dispositivos móveis e embarcados (edgeML).
- Usa camadas de ***convolução separável em profundidade*** em vez de convolução comum, reduzindo assim o tamanho do modelo, a quantidade de cálculos e melhorando a eficiência computacional da rede.

Evolução dos modelos de ML



- **MobileNetv1** (Google - 2015)
  - **Acurácia de 70.6%**
  - **Tamanho: 16.9 MB**

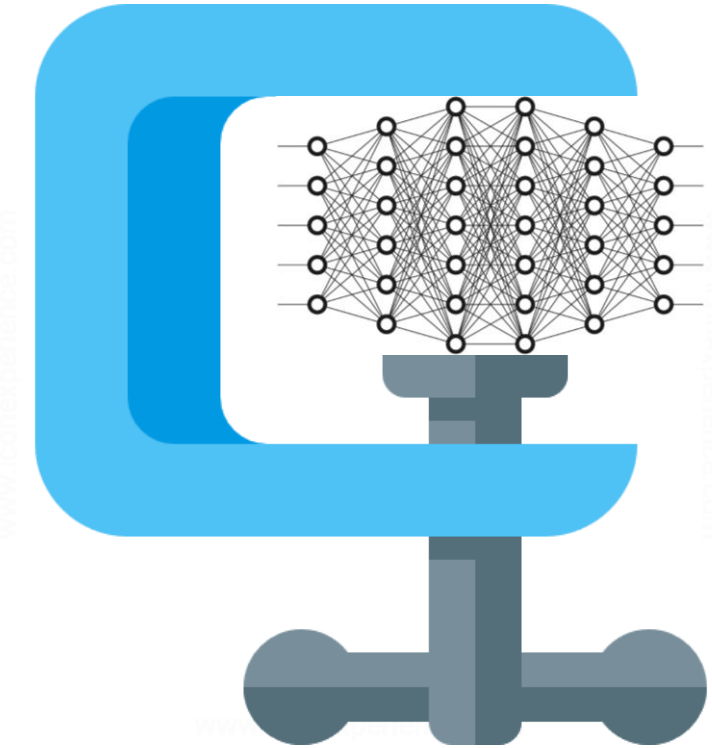


Memória RAM: 256 KB

Os modelos, até este ponto, ainda eram muito grandes para dispositivos IoT com recursos restritos.

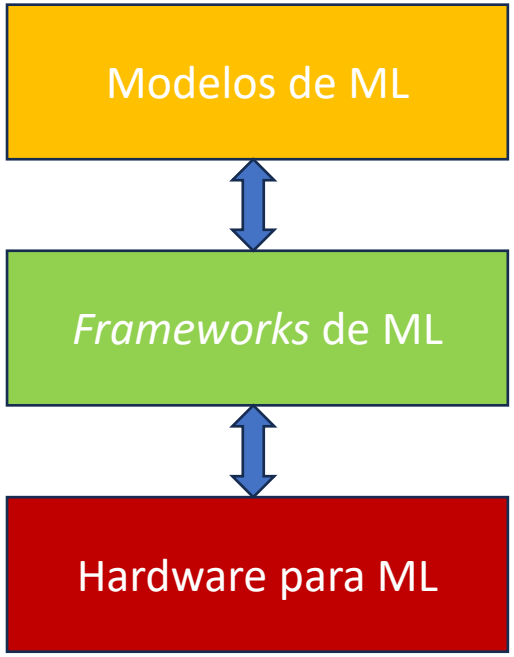
Fonte: Bianco, Simone, et al. "*Benchmark analysis of representative deep neural network architectures*." *IEEE access* 6 (2018): 64270-64277.

Como executar modelos de ML em dispositivos tinyML?



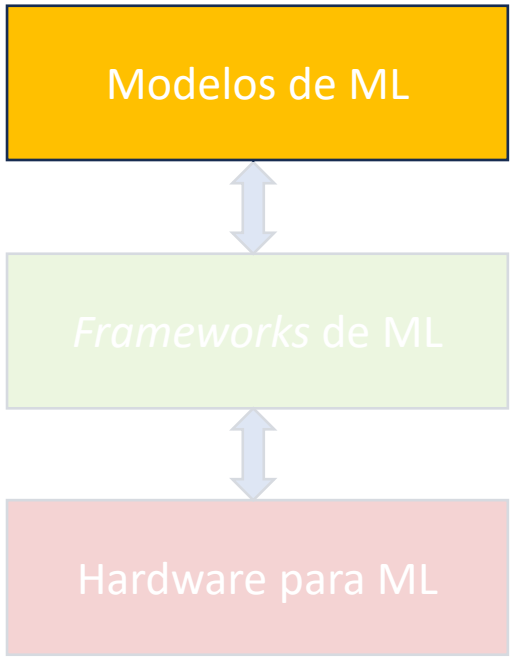
- Nitidamente, mesmo os menores modelos, não cabem em dispositivos tinyML (IoT).
- Portanto, surge a pergunta:
  - *Como podemos reduzir ainda mais esses modelos para que caibam na memória destes dispositivos, possam ser executados por eles e consumam pouca energia?*

Como executar modelos de ML em dispositivos tinyML?



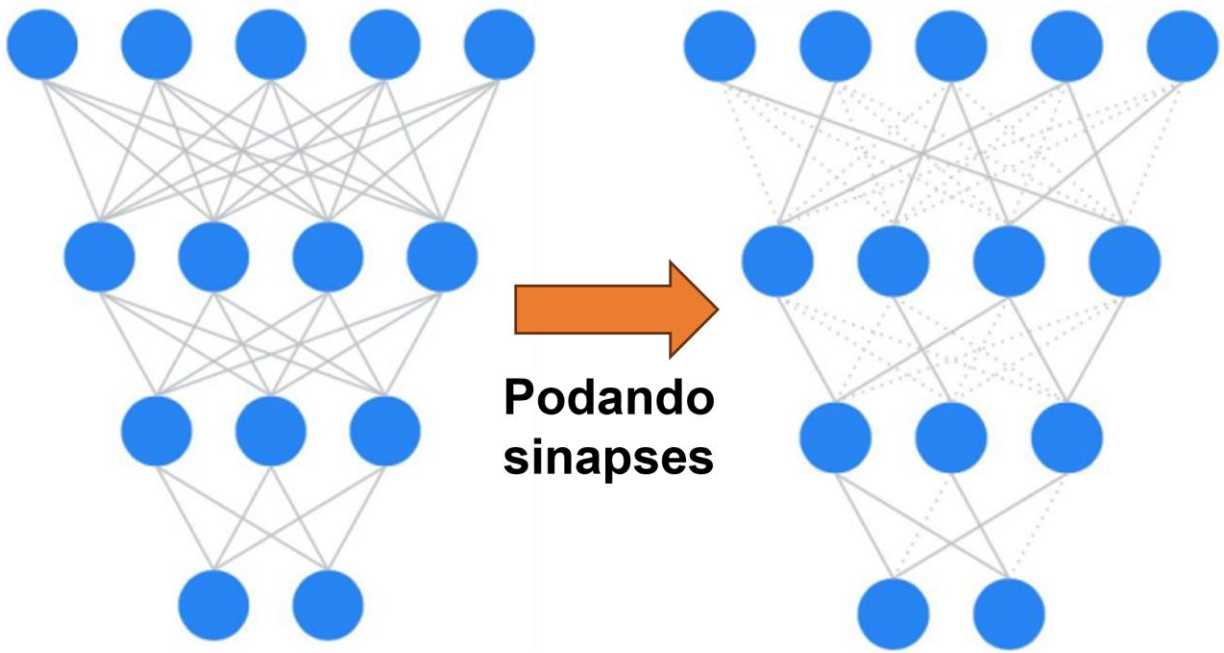
- Podemos lançar mão de três abordagens:
  - Redução/Compressão dos modelos.
  - *Frameworks* mais enxutos e eficientes.
  - Hardwares específicos/customizados para ML.

Técnicas de compressão dos modelos



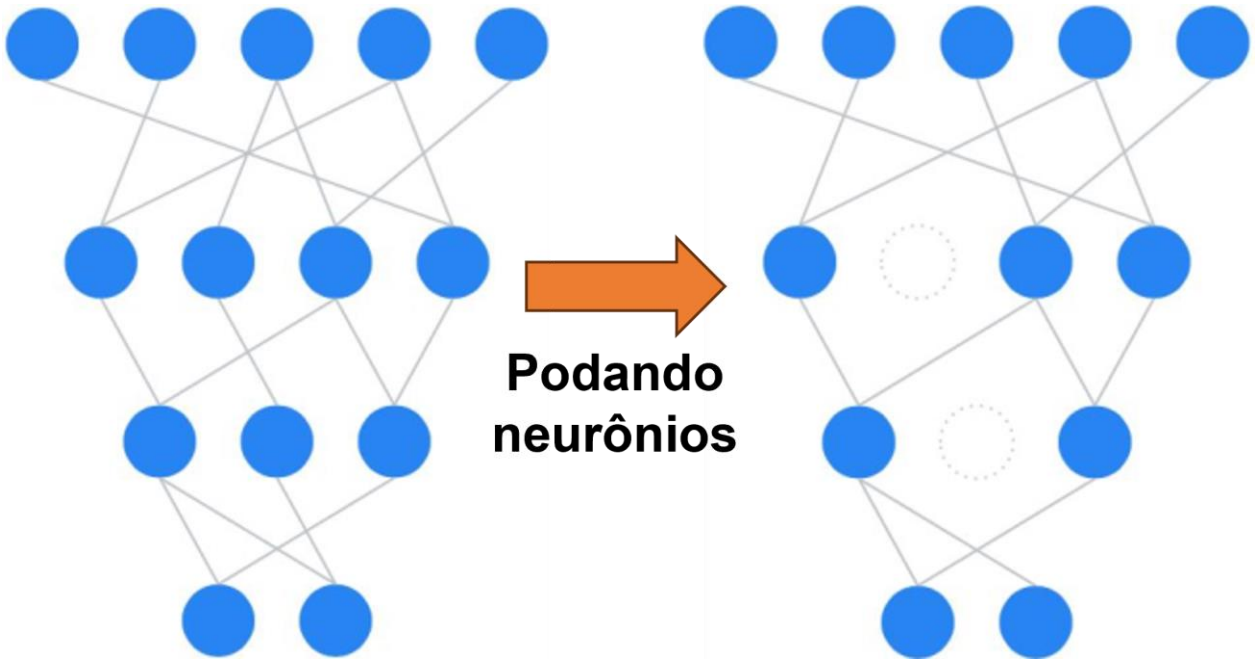
- Em geral, podemos utilizar três técnicas para reduzir os modelos sem perder muito de seu desempenho:
  - *Pruning* (ou poda);
  - Quantização;
  - *Clustering*;
  - *Knowledge Distillation* (ou destilação de conhecimento).

Pruning



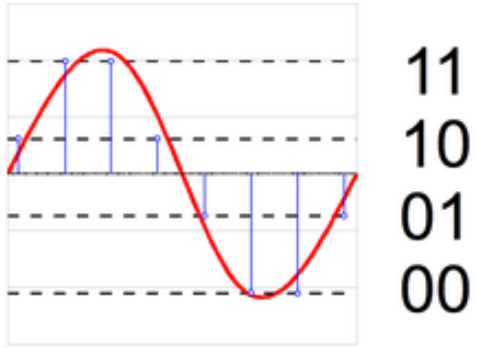
- Remove conexões para reduzir a quantidade de cálculos necessários (i.e., reduz a demanda computacional), e diminuir o tamanho do modelo.
- Aplicado após o treinamento.
- Essa remoção pode ser baseada em critérios como, por exemplo, a magnitude dos pesos.
  - Pesos com magnitudes próximas de zero são removidos do modelo pois podem não interferir no seu desempenho.

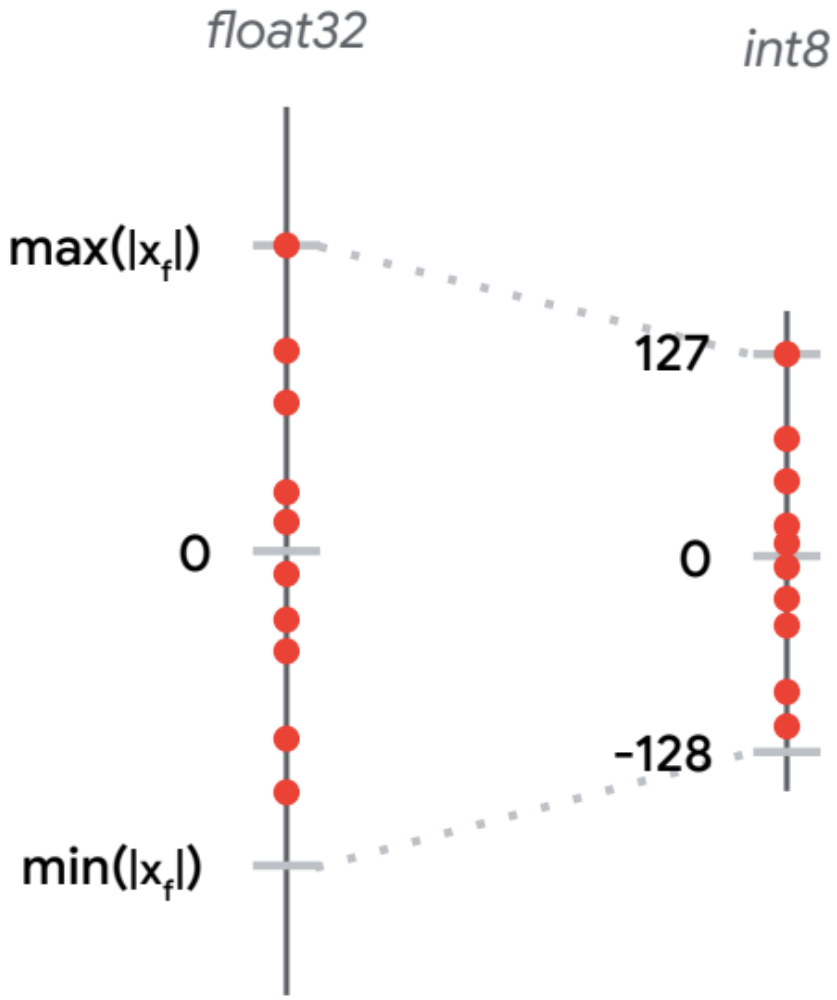
Pruning



- Podemos também ser capazes de ***remover neurônios*** completamente.
- O modelo final pode continuar com o ***mesmo desempenho ou perder parte dele***.
- Portanto, é preciso ***testar e encontrar um balanço*** entre redução do modelo e seu desempenho.
- Ajudar a ***mitigar o problema de sobreajuste***, melhorando a generalização do modelo.
- É uma das técnicas que utilizaremos em breve.

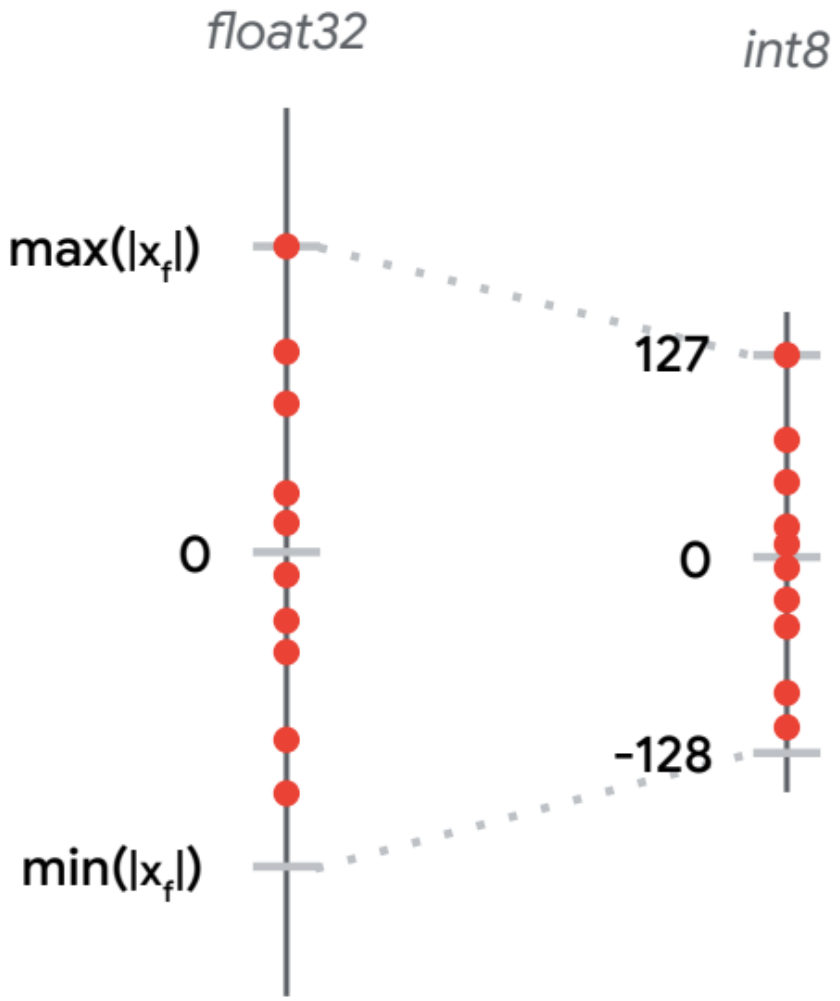
Quantização





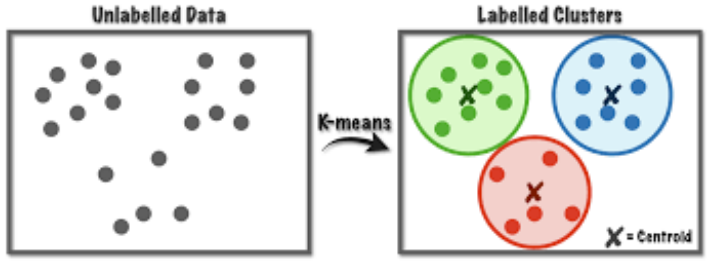
- Lembrando que sistemas embarcados não têm hardware sofisticado (e.g., FPU) e realizam apenas operações simples.
- Assim, outra técnica que iremos usar bastante é chamada de ***quantização***.
  - É a ***modificação da representação numérica*** dos valores envolvidos nos cálculos.
- Ela ***reduz a precisão dos números*** usados para representar os parâmetros (pesos, ativações, etc.) dos modelos.
- Quantizar significa dividir o intervalo de variação de um número em um conjunto discreto de valores.

Quantização



- A quantização transforma uma número com um grande intervalo de variação em outro com um intervalo menor.
  - Por exemplo, transformar uma variável do tipo *float32*, que ocupa 4 bytes e tem intervalo $\pm 3.4 \times 10^{38}$, em uma do tipo *int8*, que ocupa 1 byte e tem intervalo de -128 a 127.
  - Nesse caso, temos uma redução de 4 vezes.
- A quantização pode reduzir o desempenho do modelo, mas, em muitos casos, ela não é significativa e pode ser tolerada quando desejamos utilizar sistemas tinyML.

Clustering



Matriz 4x4 com os pesos (*floats*) de uma camada

Centroides dos clusters

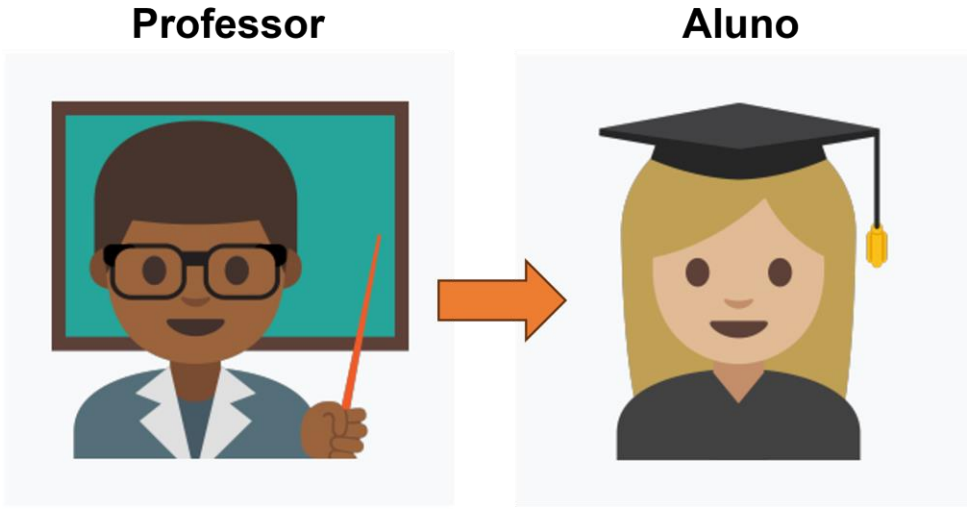
Weight matrix

0.86	-1.88	-1.49	-0.51
-1.7	1.58	0.12	-2.38
1.9	-2.19	1.37	-2.79
1.92	0.46	-2.85	-0.61

Armazena apenas os índices e os centroides

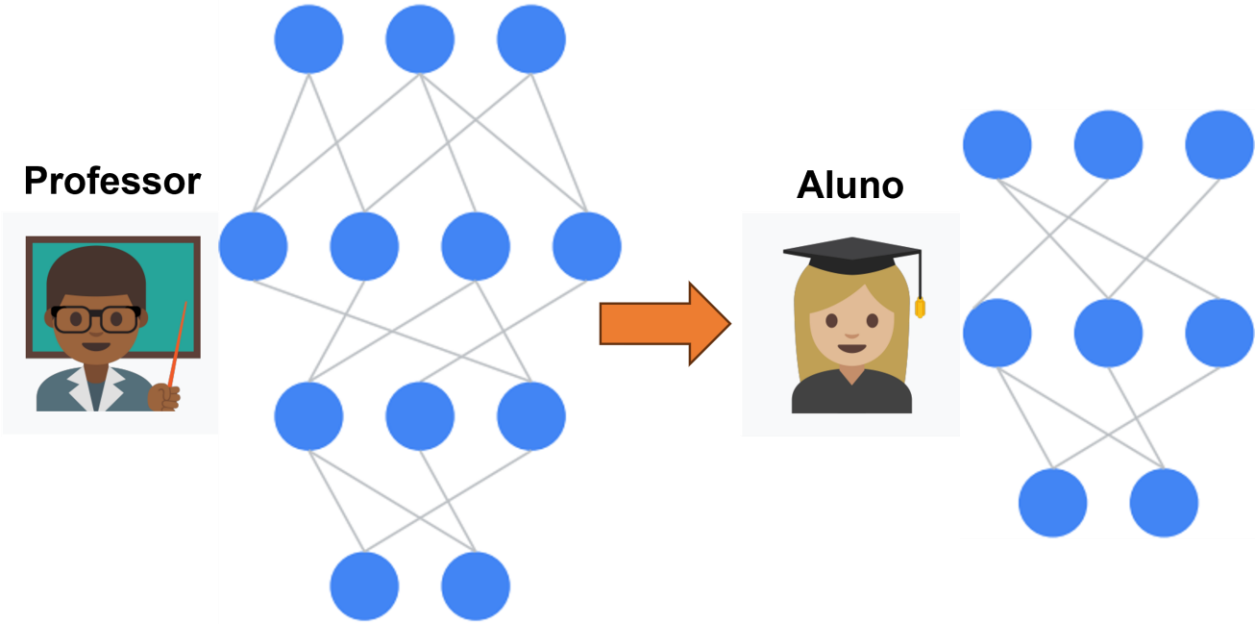
- O *clustering* de pesos reduz o ***tamanho de armazenamento e transferência de rede*** de modelos de ML.
- O *clustering* reduz o tamanho do modelo substituindo pesos semelhantes em uma camada por um mesmo valor (i.e., o centroide mais próximo).
- Podemos armazenar os índices e os centroides:
  - Neste caso, reduzimos o tamanho da matriz de 16 *floats* diferentes para 4 *floats* distintos e 16 índices de 2 bits.
- ou apenas os centroides:
  - Neste caso, reduzimos o tamanho da matriz de 16 *floats* diferentes para 4 *floats* distintos .
  - Assim, ferramentas de compactação, como zip, podem aproveitar a redundância nos dados para obter maior compactação.

Knowledge distillation



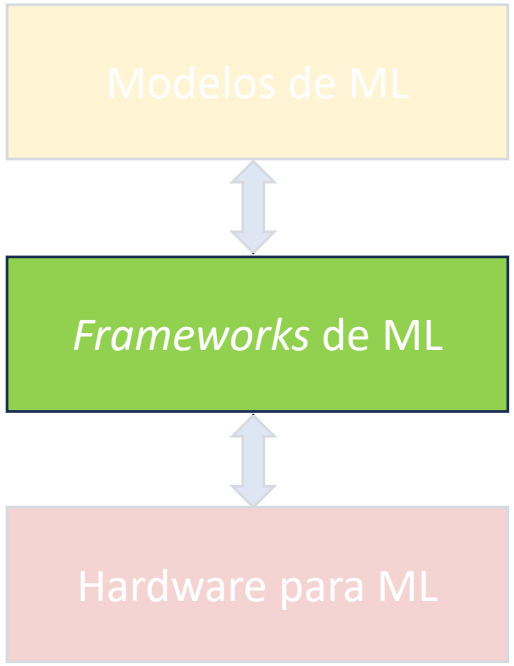
- Também conhecido como “destilação de conhecimento”
- É técnica de treinamento de modelos de ML em que um modelo maior e mais complexo, conhecido como “***professor***", é usado para ensinar (treinar) um modelo menor e mais simples, conhecido como "aluno".
- O objetivo é transferir o conhecimento ou a "sabedoria" do ***professor*** para o ***aluno***, de forma que o ***aluno*** possa obter desempenho comparável ou até mesmo melhor que o ***professor***, mas com menor complexidade e recursos computacionais.

Knowledge distillation



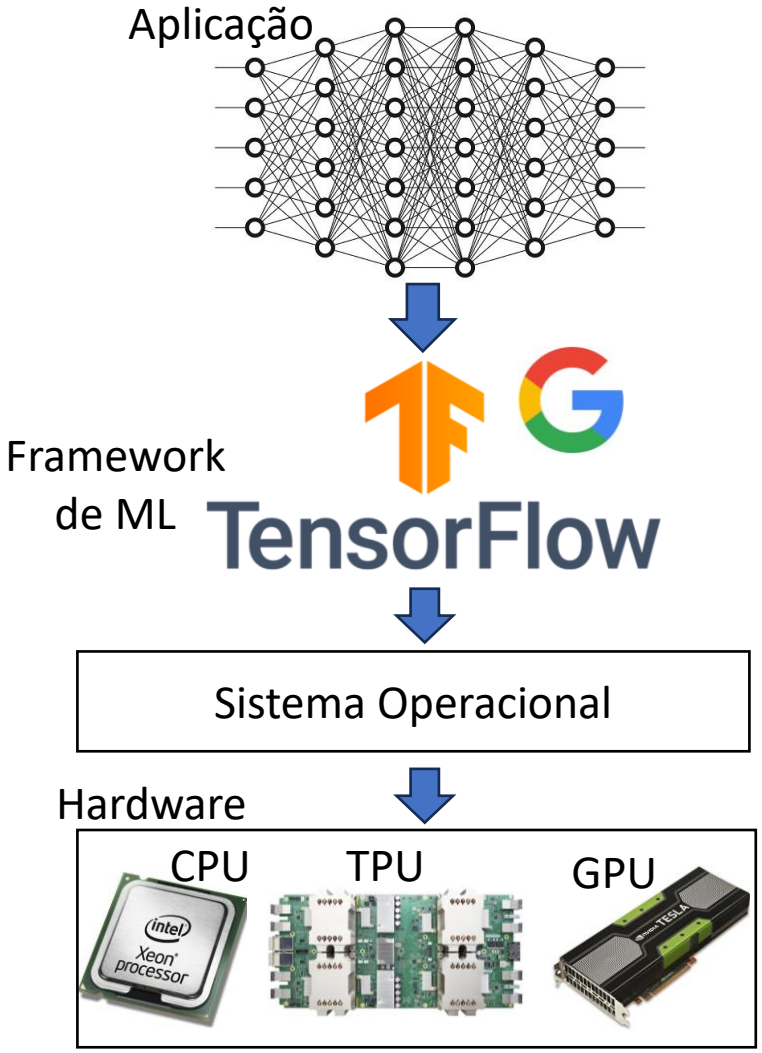
- O processo envolve alimentar o mesmo conjunto de treinamento para ambos os modelos e usar a saída do ***professor*** como um alvo suave (soft target) para guiar o treinamento do modelo aluno.
- Dessa forma, o ***aluno*** é incentivado a aprender com base nas características sutis e nuances aprendidas pelo ***professor***, melhorando assim sua capacidade de generalização e desempenho.

Frameworks de ML



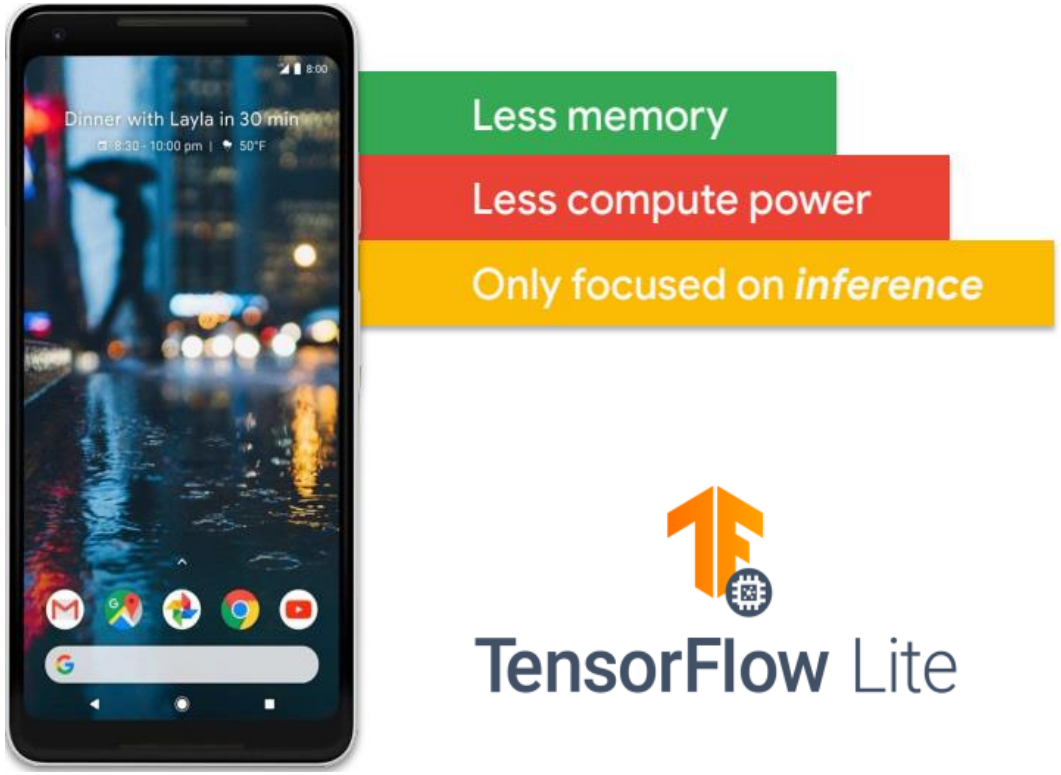
- Um ***framework*** de ML é uma biblioteca ou um conjunto de ferramentas que fornece funcionalidades e abstrações que facilitam a criação, treinamento e implantação (i.e., inferência) de modelos de ML.
- Alguns exemplos populares de ***frameworks*** de ML incluem TensorFlow, PyTorch, Scikit-learn e MXNet, etc.
- Usaremos o ***Tensorflow (TF)***, pois é focado no desenvolvimento de redes neurais (profundas).

Framework de ML



- Quando trabalhamos com modelos de ML em computadores de uso geral, além da aplicação (i.e., modelo), SO e HW, nós usamos o ***TensorFlow*** para a ***criação, treinamento e implantação dos modelos***.
- O ***TensorFlow*** oferece suporte a operações de ponto flutuante e é projetado para tarefas de alto desempenho e uso intensivo dos recursos disponíveis (com CPUs ou TPUs/GPUs).
- Porém, o ***TensorFlow*** é muito grande e quando queremos ***executar modelos de ML em dispositivos com recursos limitados***, precisamos de um ***framework*** mais ***enxuto e eficiente***.

TensorFlow (TF) Lite

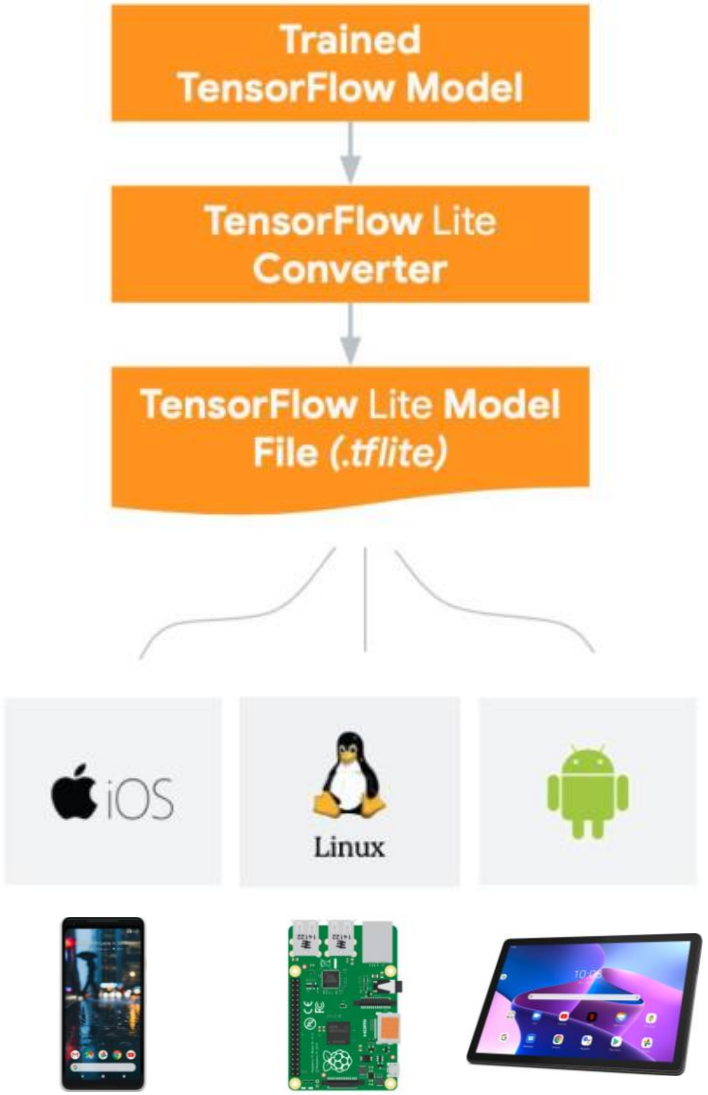


- Usado em smartphones, tablets e sistemas embarcados.
- ***Subconjunto do TF***, portanto, é mais restrito.
- Projetado para ser ***leve e eficiente em termos de uso de CPU e memória***.
- Suporta técnicas de otimização, como ***quantização***, ***pruning*** e ***clustering*** para reduzir o tamanho dos modelos e melhorar a eficiência de execução em HW com recursos limitados.
- Realiza ***apenas inferência*** (i.e., apenas executa o modelo treinado).
  - O modelo deve ser treinados em uma máquina com mais recursos.
- ***Compatível com modelos treinados no TF***, permitindo sua conversão.

Sequência de trabalho com o TF Lite

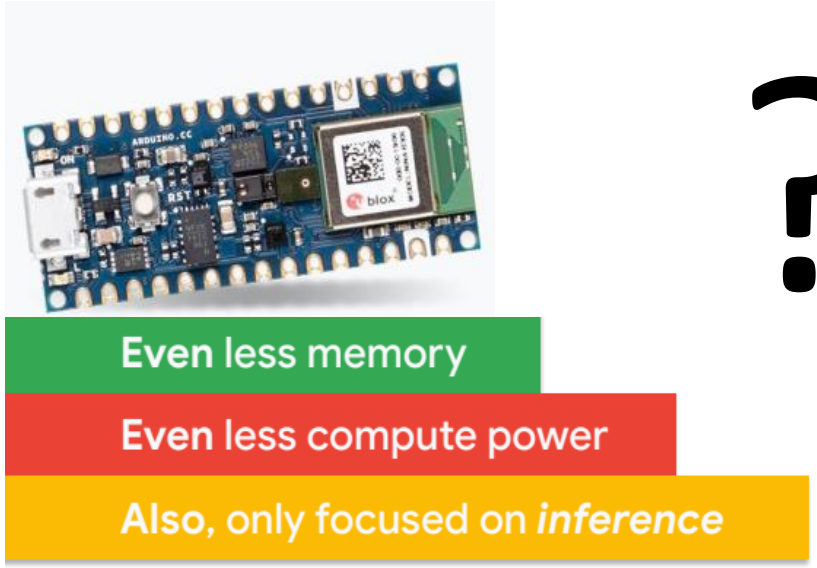
- É possível gerar um modelo do TF Lite de 3 formas:
  - Usando um modelo pré-treinado do TF Lite.
    - ✓ https://www.tensorflow.org/lite/examples?hl=pt-br
  - Criando um modelo do TensorFlow Lite.
    - ✓ Para isso, podemos usar o TensorFlow Lite Model Maker
  - Convertendo um modelo do TF em um do TF Lite.

Sequência de trabalho com o TF Lite



- O primeiro passo é treinar um modelo criado com o TF em um computador com muitos recursos.
- Após, usamos o ***TF Lite converter*** para converter esse modelo grande em um modelo pequeno.
  - Durante a conversão, podemos aplicar otimizações ao modelo.
- Esse processo gera um arquivo com extensão .tflite, o qual contém o modelo reduzido.
- Na sequência, usamos este arquivo para implantar o modelo em smartfones, tablets ou sistemas embarcados.
  - Basta carregar o arquivo com APIs específicas do TF Lite e realizar inferências.

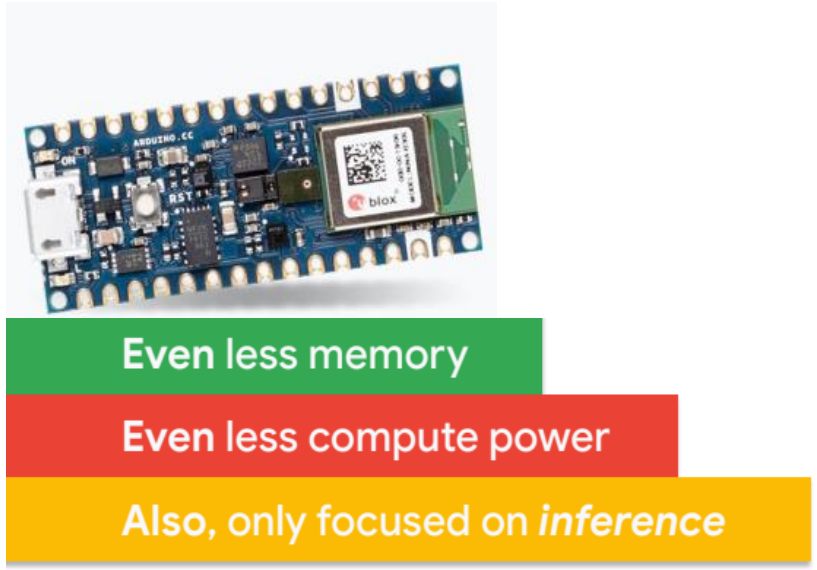
TensorFlow (TF) Lite Micro



?

- Como habilitamos o uso (inferência) de modelos de ML em sistemas embarcados com recursos extremamente limitados e que requerem baixíssimo consumo de energia, como microcontroladores?
- Esses dispositivos têm capacidades computacionais muito menores em comparação com smartphones e tablets.
- Portanto, eles requerem uma implementação ainda mais enxuta e eficiente.

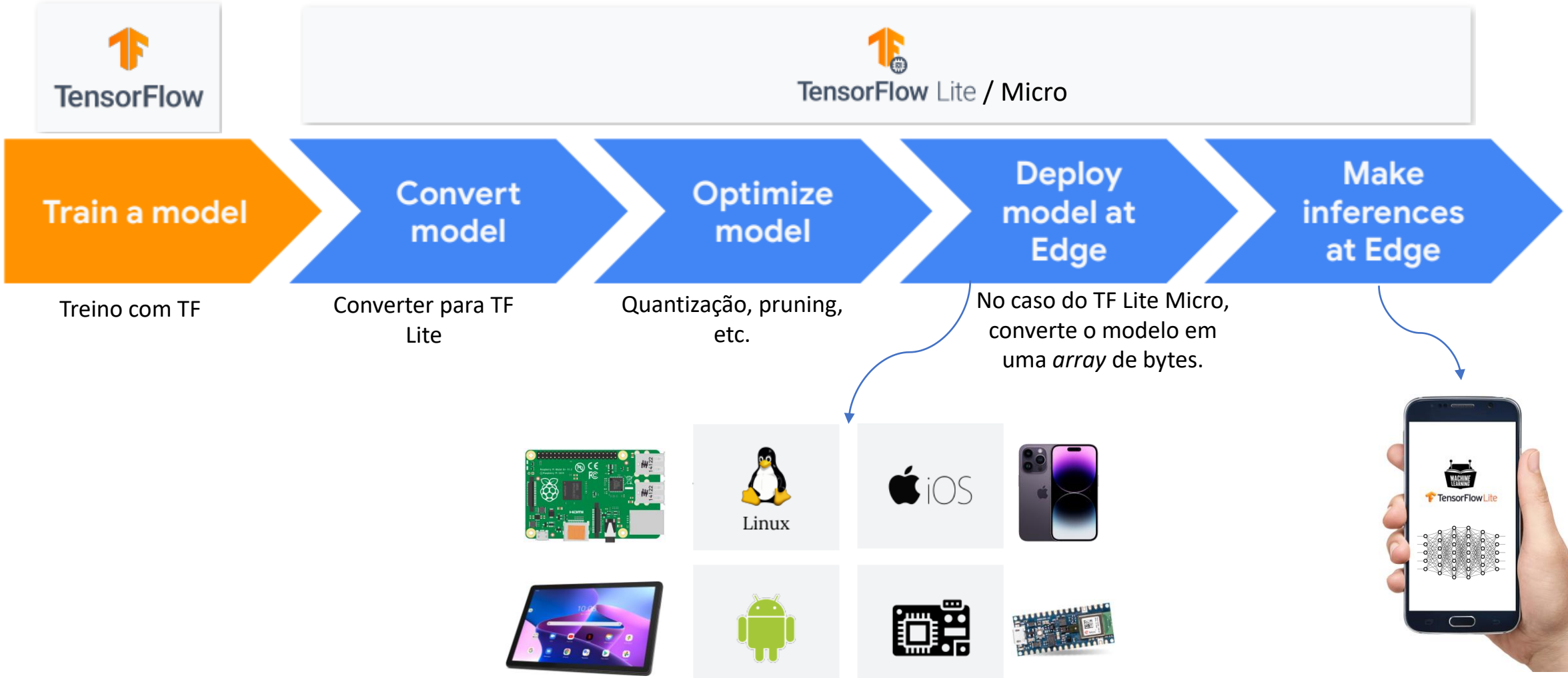
TensorFlow (TF) Lite Micro



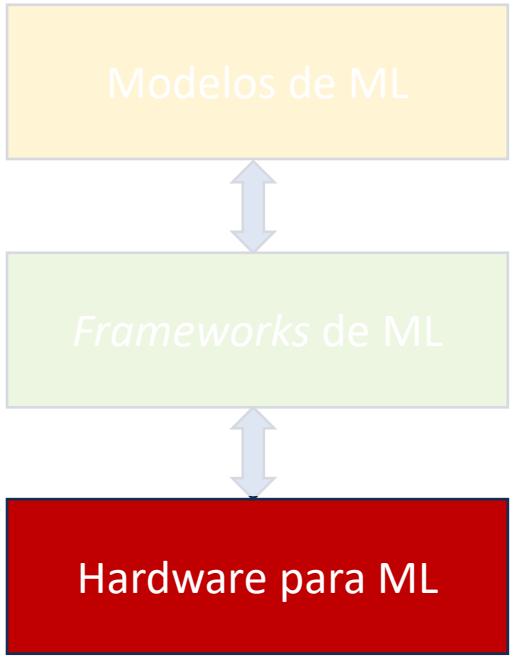
TensorFlow Lite Micro

- O TF Lite Micro é uma versão ainda mais leve e otimizada do TF.
- Projetado para dispositivos embarcados de recursos extremamente limitados (CPU e memória), como microcontroladores.
- Escrito em C++ 17 e roda apenas em plataformas de 32 bits.
- Precisa apenas de 16 KB de memória.
- Não requer suporte de sistema operacional, bibliotecas C ou C++ padrão ou alocação dinâmica de memória.
- Após a conversão do modelo TF em TF Lite, o converte para uma *array* de bytes e o compilá-lo junto ao programa.

Pipeline de desenvolvimento

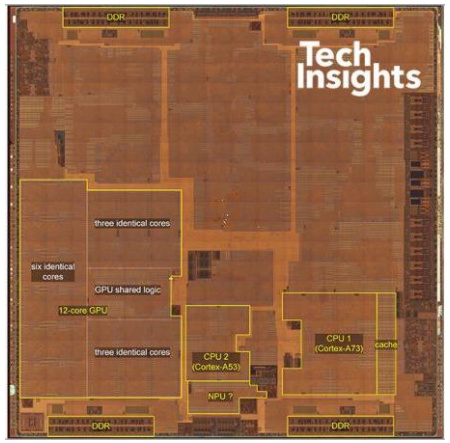


Hardware especializado

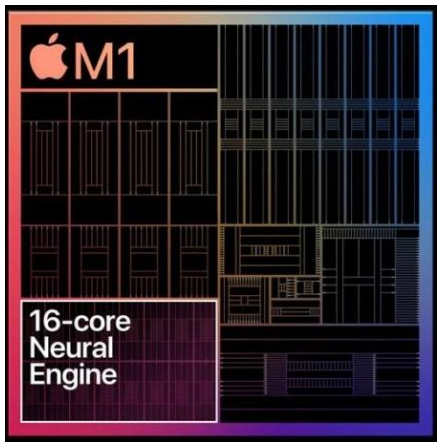


- Outra forma para habilitar o uso de ML em dispositivos embarcados é através de hardwares otimizados para a execução eficiente dos modelos.
- Algumas empresas têm lançados *Systems-on-a-Chip* (SoCs) e microcontroladores equipados com aceleradores de ML, também chamados de *neural processing units* (NPUs) ou *neural engines* (NEs).
- São hardwares altamente especializados e projetados para acelerar aplicações de ML.

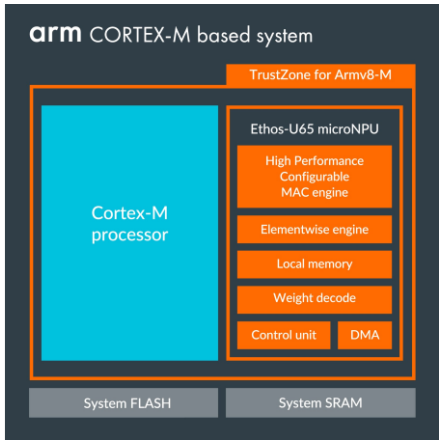
Hardware especializado



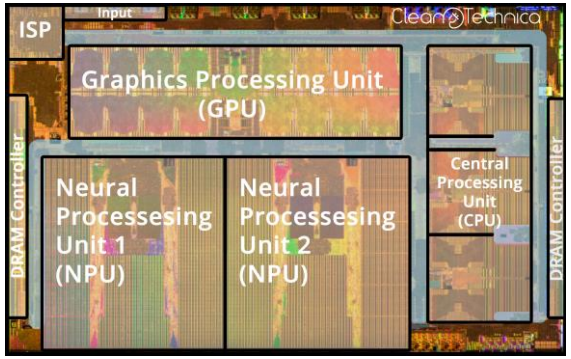
Huawei's Kirin 970



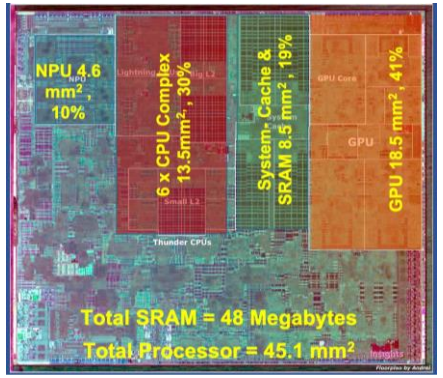
Apple's M1



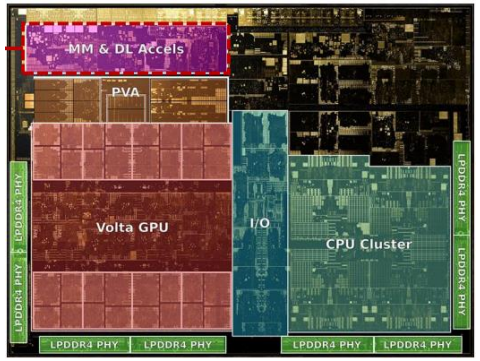
ARM's Cortex-M55 e Ethos-U55



Tesla's FSD



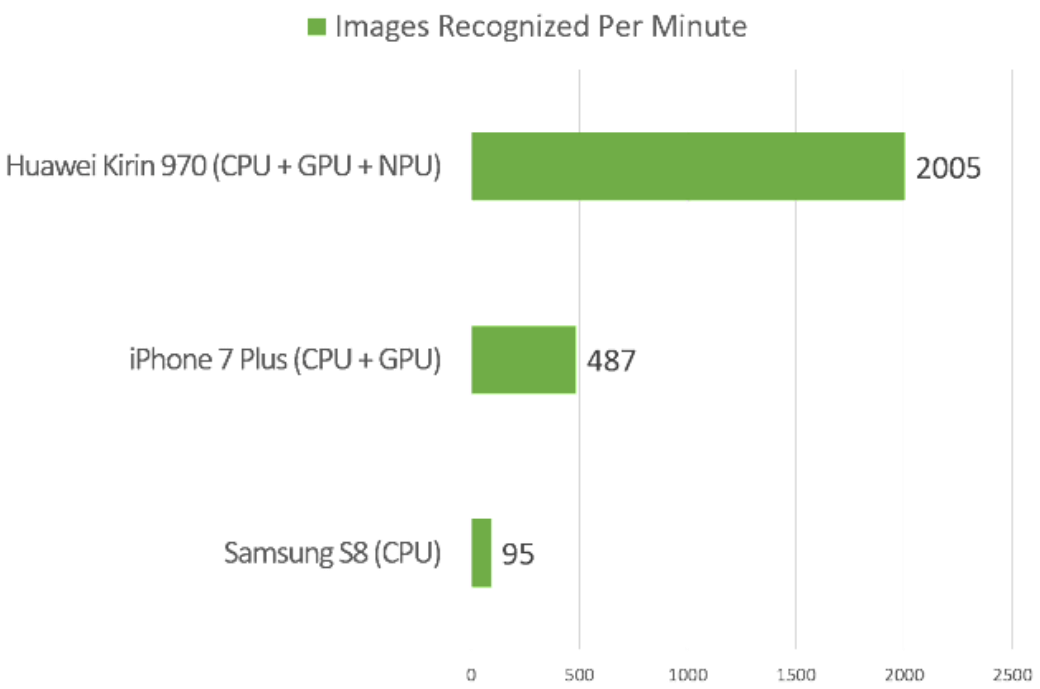
Apple's A13



NVIDIA's Xavier SoC

- Em geral, por serem muito especializadas, as NPUs oferecem uma execução muito mais rápida do que os CPUs e GPUs para tarefas de ML específicas.
- NPUs realizam operações paralelas de multiplicação-acumulação (MAC) e suportam compactação e descompactação de pesos, ajudando a minimizar o uso da memória do sistema.

Hardware especializado



- Esses HWs especializados habilitam aplicações mais complexas como detecção de objetos e reconhecimento de fala em tempo real.
- A figura ao lado mostra que a NPU no Kirin 970 da Huawei faz com que ele supere de longe o iPhone 7 plus na tarefa de reconhecimento de images..

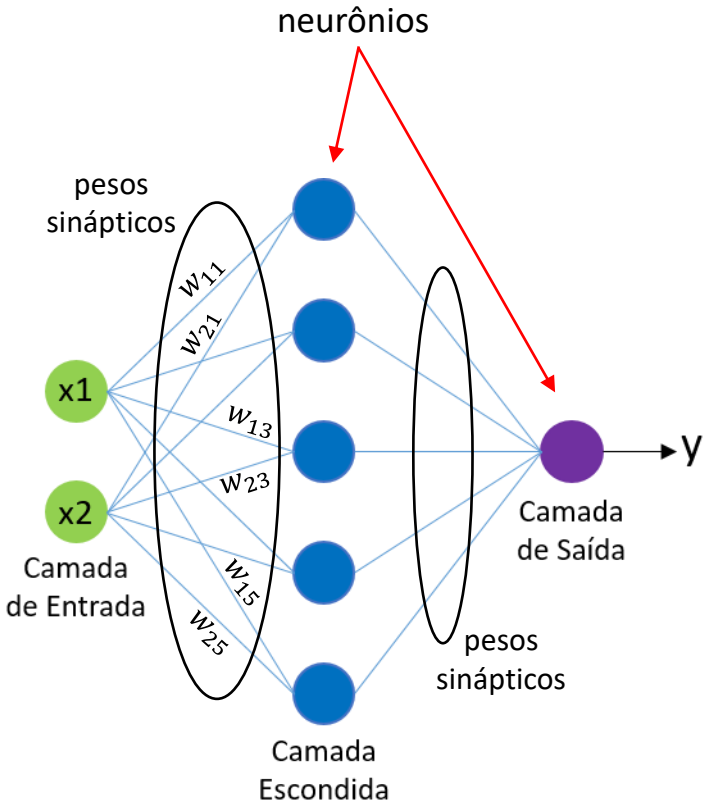
https://thetechrevolutionist.com/2017/09/huaweis-neural-processing-unit-npu-and.html

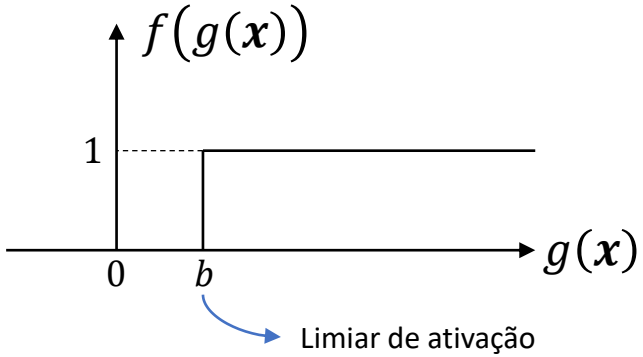
Atividades

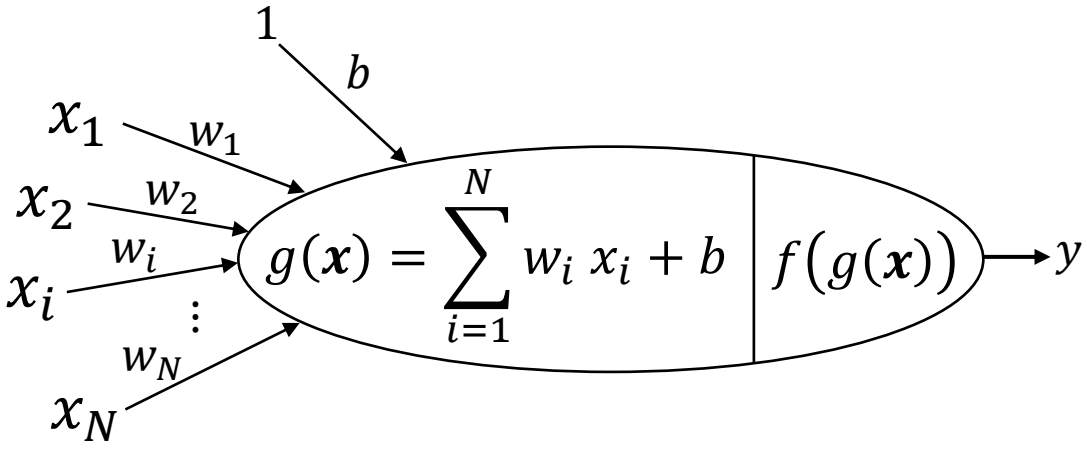
- Quiz: "***TP557 - Desafios do TinyML - Machine Learning***"
- Assistir aos vídeos do Prof. Marcelo Rovai sobre Google Colab e Python.
  - **Google Colab intro**: https://www.youtube.com/watch?v=m_Ueb88yd88&ab_channel=MarceloRovai
  - **Python review**: https://www.youtube.com/watch?v=07tGfteD4_s&ab_channel=MarceloRovai
  - **Notebooks e documentos usados nos vídeos**: https://github.com/Mjrovai/UNIFEI-IESTI01-TinyML-2022.1/tree/main/00_Curse_Folder/1_Fundamentals/Class_04a

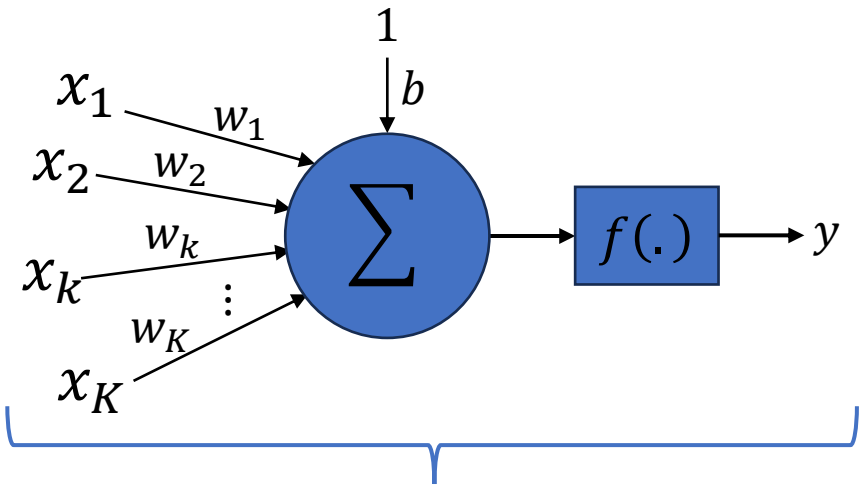
Perguntas?

Obrigado!

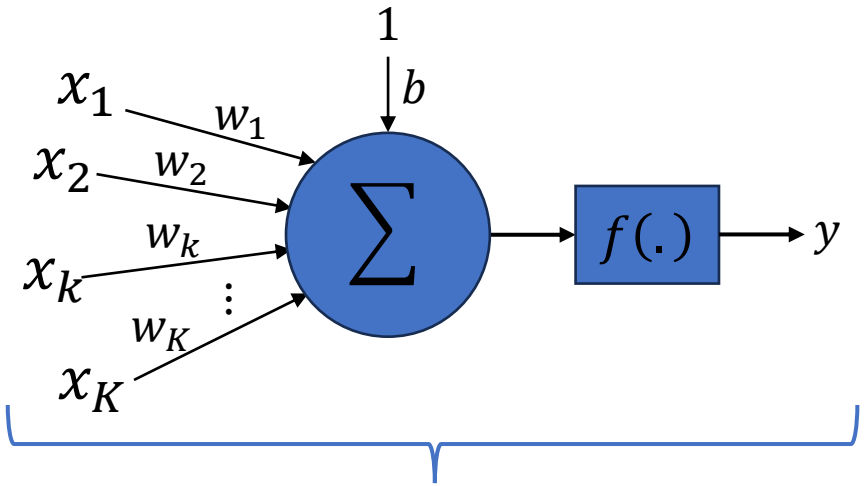

neurônios
pesos
sinápticos
w_{11}
w_{21}
w_{13}
w_{23}
w_{15}
w_{25}
x1
x2
Camada
de Entrada
Camada
Escondida
pesos
sinápticos
Camada
de Saída
y


$f(g(\boldsymbol{x}))$
1
0
b
$g(\boldsymbol{x})$
Limiar de ativação



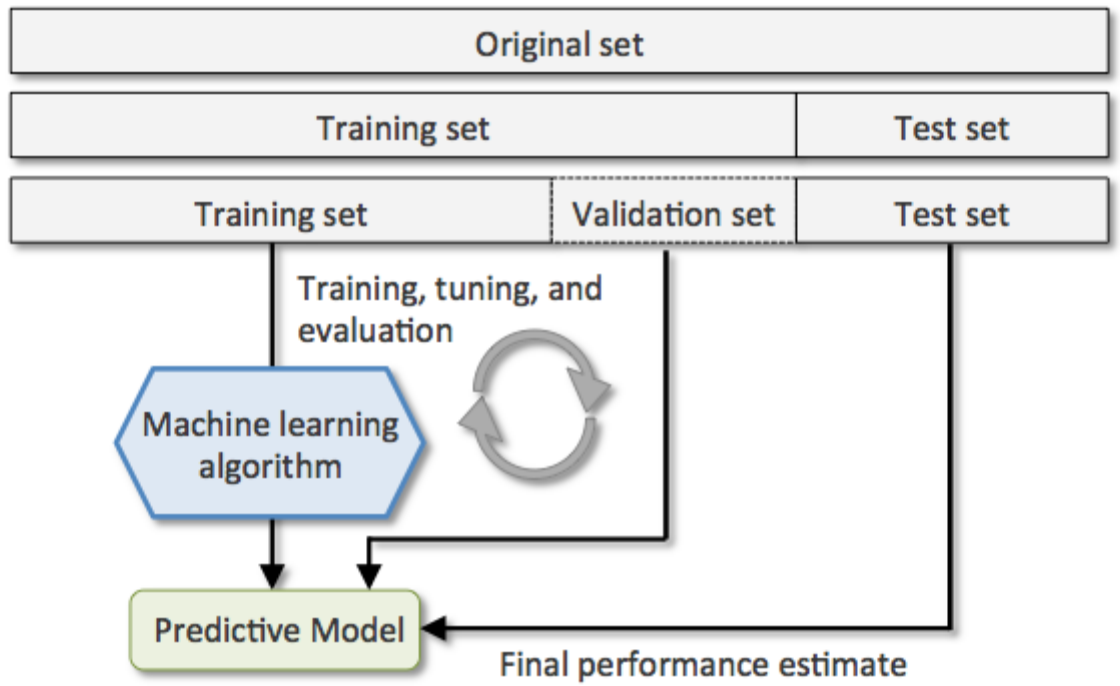


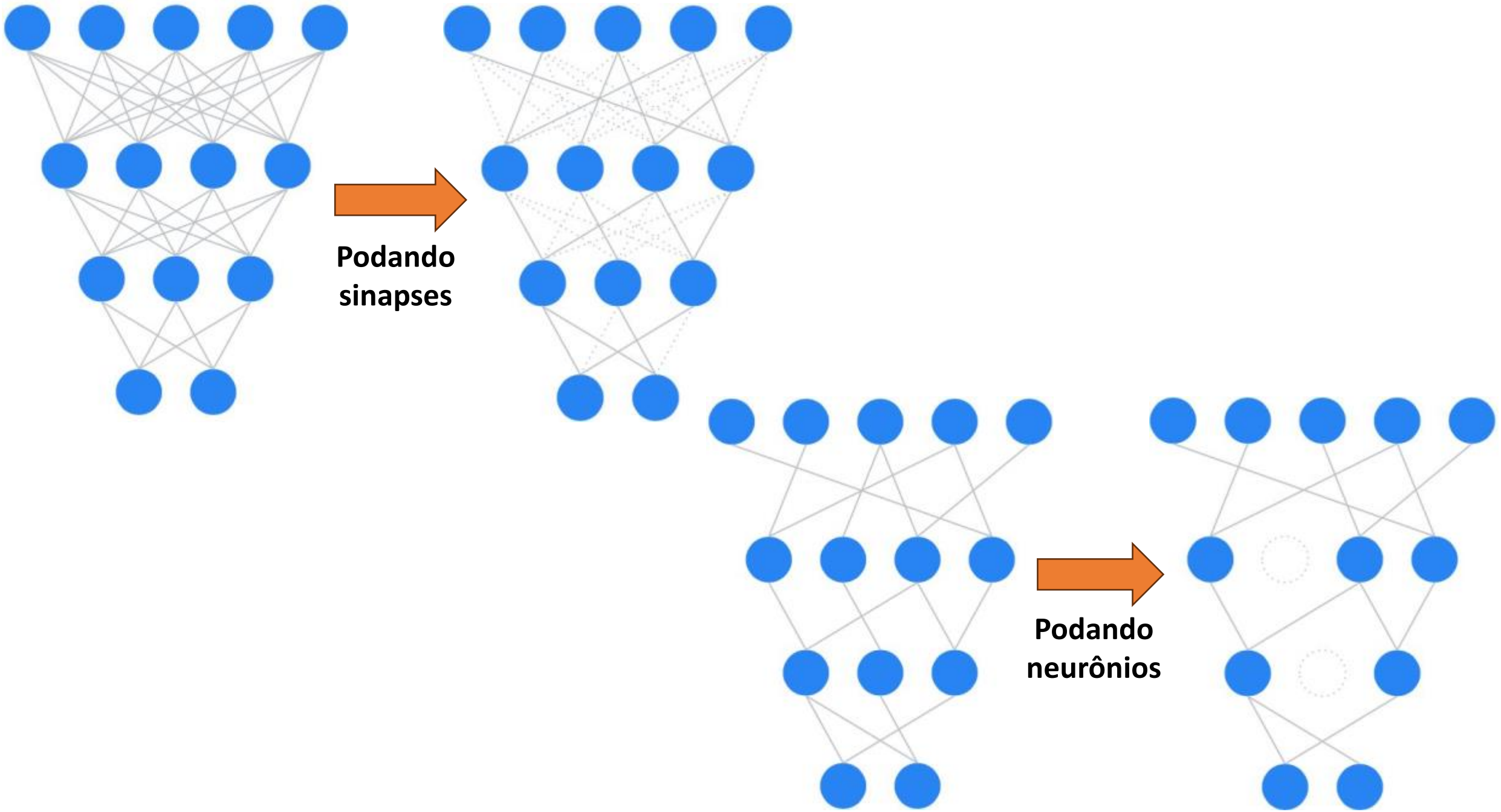
$$y = f\left(\sum_{i=1}^{K} w_i\, x_i + b\right)$$



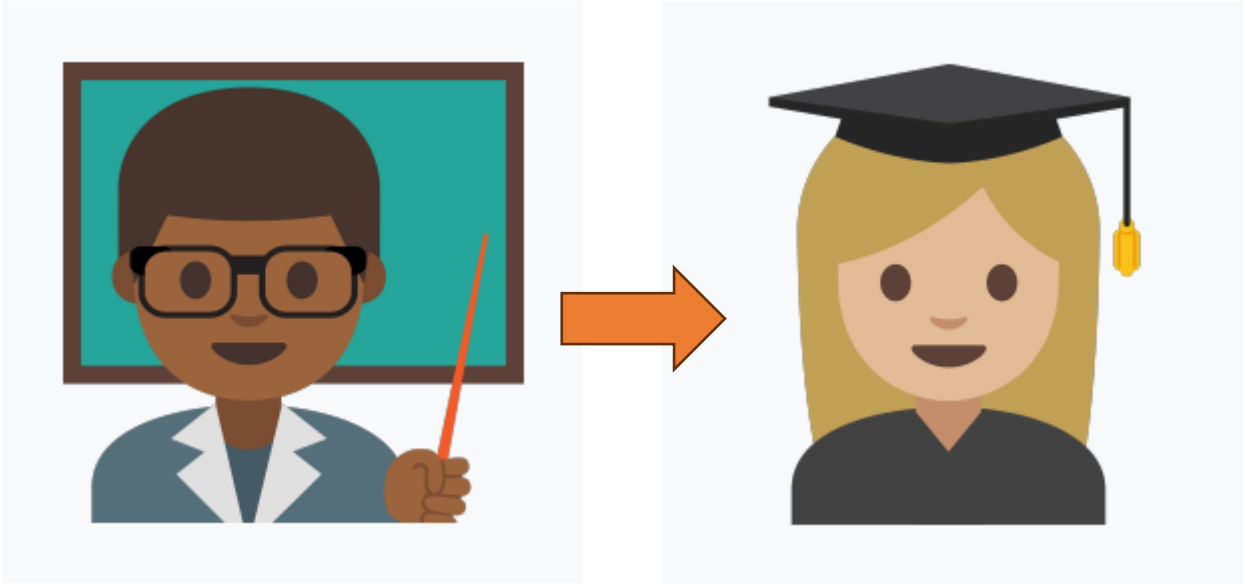
$$y = f\left(\sum_{i=1}^{K} w_i\, x_i + b\right)$$

Pesos ou parâmetros


Original set
Training set
Test set
Training set
Validation set
Test set
Training, tuning, and evaluation
Machine learning algorithm
Predictive Model
Final performance estimate

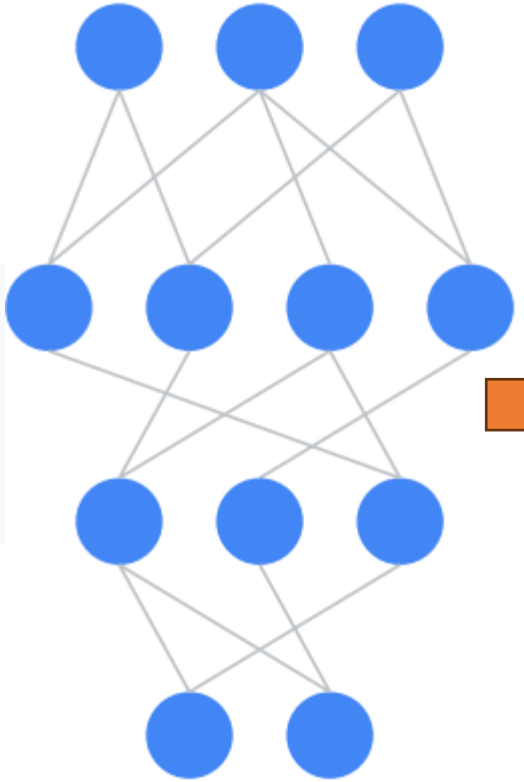

Podando
sinapses
Podando
neurônios

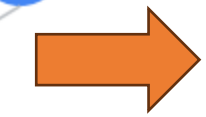
Professor



Aluno

Professor





Aluno