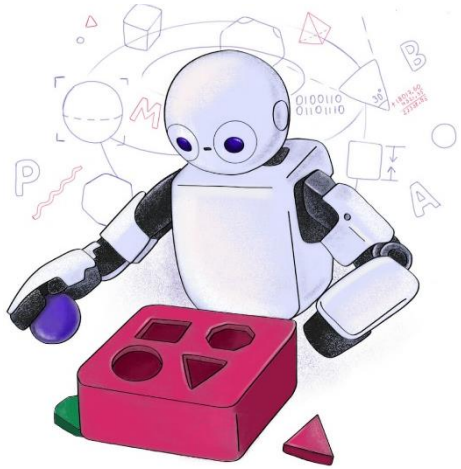
TP555 - Inteligência Artificial e Machine Learning:

Redes Neurais Artificiais com TensorFlow



Inatel

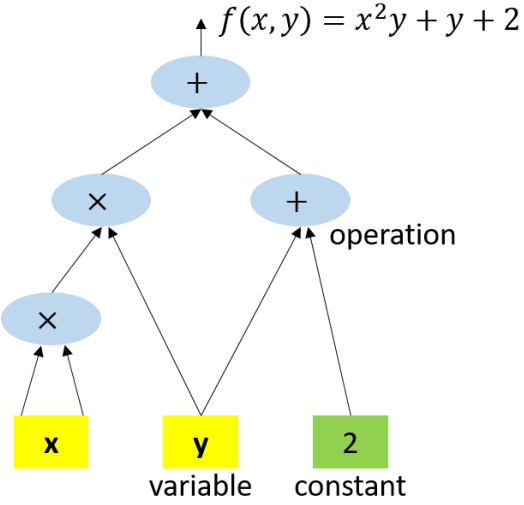
Felipe Augusto Pereira de Figueiredo
felipe.figueiredo@inatel.br

Bibliotecas para construção de redes neurais

- Existem várias bibliotecas ***open-source e gratuitas*** para criação de redes neurais que são escalonáveis e eficientes, como por exemplo:
  - ***Theano***: Universidade de Montreal (primeira versão) e PyMC (versões posteriores sob o nome de Aesara).
  - ***MXNet***: Apache,
  - ***PyTorch***: Meta AI (Facebook),
  - ***TensorFlow***: Google.
- Todas suportam a execução das redes neurais em CPUs ou GPUs e, no caso do TensorFlow, em TPUs também.
  - ***Tensor Processor Unit*** (TPU): ASIC desenvolvido especificamente para a execução eficiente do TensorFlow.
- Dentre elas, a mais difundida é a biblioteca ***TensorFlow***, desenvolvida pelo time da ***Google*** chamado de ***Google Brain***.
- Portanto, neste tópico, veremos como usar a biblioteca ***TensorFlow*** para a implementação e execução de redes neurais.

TensorFlow

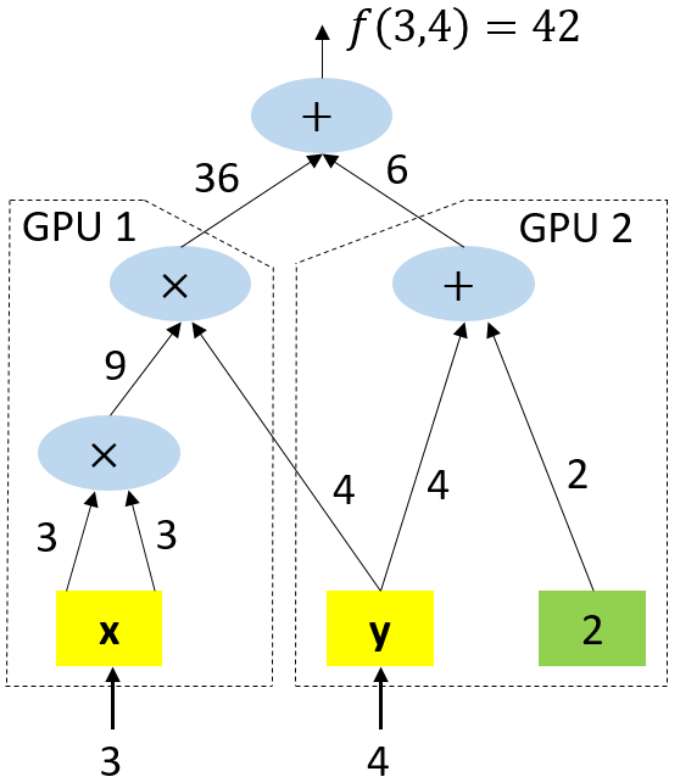
- O ***TensorFlow*** é uma biblioteca para ***computação numérica diferenciável*** usada para a criação, treinamento e execução de algoritmos de ML.
- Possibilita a ***criação*** de algoritmos de ***forma rápida*** e sua ***execução em larga escala***.
- O nome ***TensorFlow*** deriva dos tipos de operações que a biblioteca pode realizar em ***matrizes de dados multidimensionais***, conhecidas como ***tensores***.
- Os cálculos no ***TensorFlow*** são expressos como ***grafos de computação*** (como mostrado na figura ao lado).
- ***TensorFlow*** fornece APIs em Python, C++ e JavaScript.
- Os ***grafos de computação*** podem ser executados em CPUs, GPUs ou TPUs.



Grafo de computação

TensorFlow

- O ***TensorFlow*** possibilita dividir o ***grafo*** em várias partes e executá-las em paralelo em várias CPUs, GPUs ou TPUs, como mostrado na figura ao lado.
- O ***TensorFlow*** também suporta ***computação distribuída***: pode-se treinar ***redes neurais*** gigantescas com ***conjuntos de treinamento*** imensos em um período de tempo razoável, dividindo-se os cálculos através de centenas de servidores.
- O ***TensorFlow*** foi projetado para ser ***flexível, escalável e pronto para produção***.
- Roda no Windows, Linux, macOS e também em dispositivos móveis e embarcados, incluindo iOS, Android e Raspberry Pi com o ***TensorFlow Lite***.

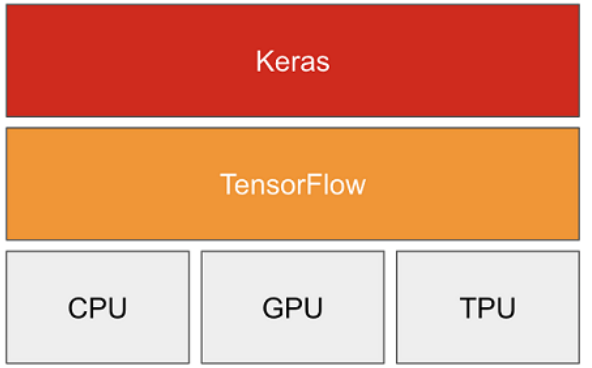


Grafo de computação

TensorFlow

- Inclui implementações em ***C++ altamente eficientes de muitas operações de aprendizado de máquina***, como por exemplo, os ***otimizadores***, que são usados para encontrar os ***parâmetros*** (i.e., pesos) que minimizam a ***função de custo***.
- Os ***otimizadores*** implementam as várias versões do gradiente descendente: GDE, Momentum, RMSProp, Adam, etc.
- As ***operações de otimização*** são muito fáceis de usar, pois o ***TensorFlow*** cuida ***automaticamente do cálculo dos gradientes das funções*** que você define.
  - Isso é chamado de ***diferenciação automática*** (ou ***autodiff***).
- Oferece uma ferramenta de visualização chamada ***TensorBoard***, que permite navegar pelo ***grafo de computação*** e visualizar as ***curvas de aprendizado***.
- Possui uma equipe dedicada de desenvolvedores e uma comunidade crescente que contribui para melhorá-lo cada vez mais.

TensorFlow



- O ***Tensorflow*** oferece dois conjuntos de application programming interfaces (APIs): o nativo (ou original) e o ***Keras***.
- O conjunto de APIs nativas oferece muita flexibilidade ao custo de maior complexidade.
  - API indicada se você quer criar qualquer tipo de grafo de computação, incluindo qualquer arquitetura de ***rede neural*** que você possa imaginar.
  - Porém, a sintaxe das APIs é bastante complicada e prolixa.
- O ***Keras*** é um conjunto de APIs simples e de alto nível construído sobre o ***TensorFlow*** que facilita seu uso (criação, treinamento e validação de modelos) em detrimento de uma menor flexibilidade.
  - Por ser mais fácil de usar e menos flexível do que o conjunto de APIs nativas do TensorFlow, é usado para prototipagem rápida.
- Porém, o ***TensorFlow*** oferece operações muito mais complexas que o ***Keras***, além de ter um debugger especializado, que pode salvar horas de trabalho.
- Entretanto, devido a sua facilidade de uso e rapidez para prototipagem, utilizaremos o ***Keras***.

APIs do Keras

- O ***Keras*** fornece duas APIs para implementar modelos de aprendizado de máquina: ***sequencial*** e ***funcional***.
- A API ***sequencial***, como o nome sugere, permite a criação de modelos camada por camada.
  - É a ***forma mais simples de criar modelos*** de aprendizado de máquina.
  - Porém, é também a ***mais limitada***, pois ***não podemos compartilhar ou ramificar camadas***.
  - Além disso, ***não podemos ter modelos com várias entradas ou saídas***.
- A API ***funcional*** é mais flexível que a sequencial, permitindo a criação de modelos mais complexos.
  - Permite o ***compartilhamento ou ramificação de camadas***. Além de podermos ter ***várias entradas e saídas***.

Classificador com a API sequencial

- Vamos construir um classificador com a API ***sequencial*** do Keras.
- Vamos usá-lo para classificar imagens que representam itens de vestuário como botas, camisetas, bolsas, tênis, etc.
- A base de dados usada será a *Fashion MNIST*, que contém 70.000 imagens em tons de cinza de 28x28 pixels cada, com 10 classes.
- Os rótulos são os IDs das classes, de 0 a 9, onde cada número corresponde às classes listadas ao lado.

"T-shirt/top",
"Trouser",
"Pullover",
"Dress",
"Coat",
"Sandal",
"Shirt",
"Sneaker",
"Bag",
"Ankle boot"

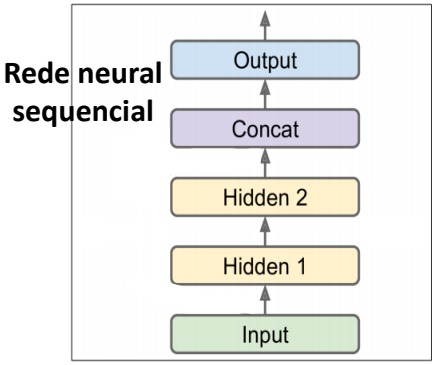
Exemplo: classificador_com_api_sequencial.ipynb

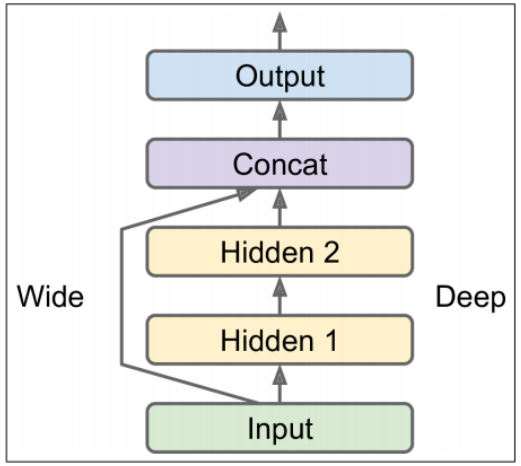
Regressor com a API sequencial

- Vamos construir um regressor com a API ***sequencial*** do Keras.
- Vamos usá-lo para predizer o valor médio de casas.
- O conjunto de dados utilizado é o habitacional da Califórnia.
- Esse conjunto possui 20640 exemplos com 8 atributos e 1 rótulo numéricos.
- O rótulo é o valor médio de casas no estado da Califórnia expresso em centenas de milhares de dólares. Os atributos são os seguintes:
  - ***MedInc***: renda média em um grupo de quarteirões.
  - ***HouseAge***: idade média da casa no grupo de quarteirões.
  - ***AveRooms***: número médio de salas por família.
  - ***AveBedrms***: número médio de quartos por família.
  - ***Population***: população do grupo de quarteirões.
  - ***AveOccup***: número médio de membros da família.
  - ***Latitude***: latitude do grupo de quarteirões.
  - **Longitude**: longitude do grupo de quarteirões.

Exemplo: regressor_com_api_sequencial.ipynb

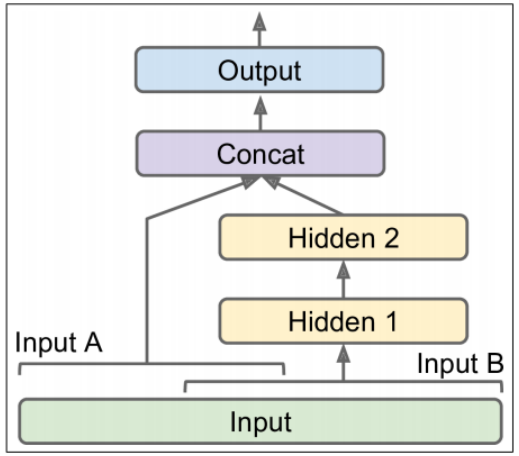
API funcional do Keras

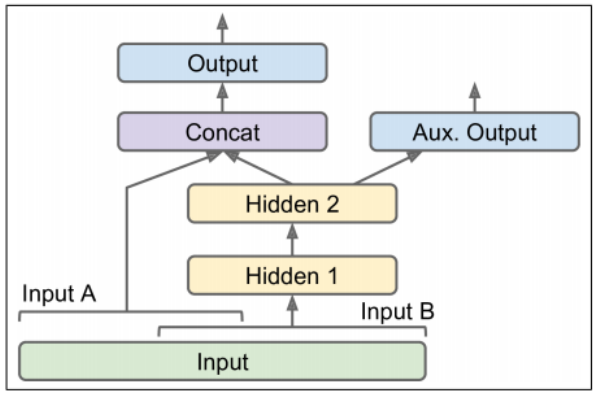




Rede neural Wide & Deep.

- Uma rede neural sequencial força os dados a ***fluir através da sequência de camadas***.
- Em contraste, quando queremos criar modelos não-sequenciais, usamos a API ***funcional***.
- A API funcional possibilita a criação de modelos como os mostrados nas figuras ao lado.
  - Se queremos passar os atributos não apenas para a camada de entrada, mas também para camadas internas da rede neural.
  - Se queremos passar diferentes conjuntos de atributos para diferentes camadas (i.e., múltiplas entradas).
  - Se queremos ter múltiplas saídas.





Exemplo: lidando_com_varias_saidas_com_api_funcional.ipynb

Construindo modelos dinâmicos usando a API de subclasses

- Tanto a API ***sequencial*** quanto a ***funcional*** são ***declarativas***.
  - Inicialmente ***declaramos*** quais camadas desejamos usar e como elas devem ser conectadas, e só então podemos começar a alimentar o modelo com dados para treinamento ou predição.
- Isso tem muitas ***vantagens***: o modelo pode ser facilmente salvo, clonado, compartilhado, sua estrutura pode ser analisada, o *framework* pode inferir dimensões e verificar tipos das *arrays*, para que erros possam ser detectados antecipadamente (ou seja, antes que qualquer dado passe pelo modelo).
- Também é bastante fácil de depurar, pois todo o ***modelo é apenas um grafo estático de camadas***.
- Porém, esta é a grande desvantagem, o modelo é ***estático***.
- Com construímos modelos dinâmicos?

Construindo modelos dinâmicos usando a API de subclasses

- Alguns modelos envolvem laços de repetição, dimensões que se alteram, estruturas condicionais, operações de baixo nível do ***Tensorflow*** e outros comportamentos altamente dinâmicos.
- Para esses casos, deve-se usar a ***API de subclasses***.
- Para usá-la, basta simplesmente ***herdar*** da classe ***Model***, ***criar as camadas necessárias no construtor*** e usá-las para ***realizar os cálculos desejados no*** método ***call()***.
- Os modelos criados com o Keras, incluindo aqueles criados a partir da API de subclasses, podem ser usados como qualquer outra camada.
- Portanto, podemos usá-los para construir arquiteturas complexas, tornando-a uma API excelente para pesquisadores experimentando novas ideias.

Exemplo: modelo_dinâmico_usando_a_api_de_subclasses.ipynb

Construindo modelos dinâmicos usando a API de subclasses

- A flexibilidade extra da API de subclasses tem algumas ***desvantagens***:
  - A arquitetura do modelo está ***escondida*** dentro do método ***call()***, então o Keras
    - não pode inspecioná-lo facilmente,
    - não pode salvá-lo ou cloná-lo,
    - não apresenta nenhuma informação sobre como as camadas estão conectadas umas às outras quando se usa o método ***summary()*** (nesse caso, obtemos apenas uma lista de camadas).
  - Além disso, o Keras não pode verificar tipos e dimensões com antecedência, e, portanto, é mais fácil cometer erros.
- Portanto, a menos que realmente precisemos dessa flexibilidade extra, provavelmente devemos nos ater às APIs ***sequencial*** ou ***funcional***.

Salvando e restaurando um modelo

- Para salvar um modelo treinado, usamos o método ***save()*** da classe ***Model***.
- O Keras salva a ***arquitetura do modelo*** (incluindo os hiperparâmetros de cada camada) e o ***valor de todos os parâmetros do modelo*** para cada camada (por exemplo, pesos sinápticos e de bias e parâmetros não treináveis), usando o formato HDF5 (arquivos com extensão .h5).
- Ele também salva o ***otimizador*** incluindo seus hiperparâmetros e qualquer estado interno que ele possa ter.
- Para salvar, passamos para o método ***save()*** o caminho onde o arquivo HDF5 deve ser salvo.
- Para carregar um modelo salvo, usamos a função ***load_model()*** passando para ela o caminho onde o arquivo HDF5 está salvo.

Exemplo: salvando_e_restaurando_um_modelo.ipynb

Usando callbacks

Exemplo: usando_callbacks.ipynb

- Mas e se o treinamento durar várias horas?
  - Isso é bastante comum, especialmente ao treinar com grandes conjuntos de dados.
- Nesse caso, não devemos apenas salvar um modelo ao final do treinamento, mas também ***salvar pontos de verificação em intervalos regulares*** durante o treinamento.
- Podemos configurar o método ***fit()*** para salvar pontos de verificação, mas como fazer esta configuração?
  - A resposta é: através de ***callbacks***.
  - ***callbacks*** são funções passadas como argumento para outras funções e chamadas quando um evento ocorrer.
- O método ***fit()*** possui um parâmetro chamado ***callbacks***, que permite especificar uma lista de objetos que o Keras ***chamará durante o treinamento*** no ***início e no final do treinamento***, ***no início e no final de cada época*** e até ***antes e depois do processamento de cada mini-batch***.
- Veja todas as ***callbacks*** disponíveis em https://keras.io/callbacks/

Visualização de resultados com TensorBoard

- O ***TensorBoard*** é uma ferramenta de visualização interativa. Ele é usado para:
  - visualizar as curvas de aprendizado,
  - comparar curvas de aprendizado entre várias execuções,
  - visualizar o grafo de computação,
  - analisar estatísticas de treinamento,
  - visualizar imagens geradas pelo seu modelo,
  - visualizar dados multidimensionais projetados em 3D e agrupados automaticamente, etc.

Exemplo:usando_o_tensorboard.ipynb

- Ele é instalado automaticamente quando você instala a biblioteca ***TensorFlow***.
- Para usá-lo, devemos modificar o código para que ele escreva os dados que desejamos visualizar em arquivos de binários chamados de ***arquivos de eventos*** (*event files*).
- Cada registro (i.e., entrada) de dados binários no ***arquivos de eventos*** é chamado de ***resumo*** (*summary*).
- O servidor do ***TensorBoard*** monitora o diretório de logs e coleta automaticamente as alterações e atualiza as visualizações.

Ajuste fino dos hiperparâmetros de um modelo

- A ***flexibilidade de configuração*** das redes neurais também é uma de suas ***principais desvantagens***: existem ***muitos hiperparâmetros*** para se ajustar.
- Nós podemos não apenas usar qualquer tipo de ***topologia*** de rede imaginável (i.e., como os neurônios são interconectados), mas também podemos alterar o ***número de camadas***, o ***número de neurônios*** por camada, o tipo de ***função de ativação*** a ser usada em cada camada, a lógica de ***inicialização dos pesos (sinápticos e de bias)*** e muito mais.
- Sendo assim, como sabemos qual combinação de hiperparâmetros é a melhor para uma dada tarefa?

Ajuste fino dos hiperparâmetros de um modelo

Exemplo: otimização_hiperparametrica.ipynb

- Uma opção simples é testar várias combinações de hiperparâmetros e verificar qual funciona melhor no conjunto de validação.
- Para isso, uma abordagem simples é usar as classes ***GridSearchCV*** ou ***RandomizedSearchCV*** da biblioteca SciKit-Learn para explorar o espaço de hiperparâmetros.
- A classe ***GridSearchCV*** faz uma ***busca exaustiva***, testando todas as possíveis combinações de hiperparâmetros.
  - Porém, como geralmente existem muitos hiperparâmetros para se ajustar, e como treinar uma rede neural com um grande conjunto de dados leva muito tempo, nós poderíamos explorar apenas uma pequena parte do espaço de hiperparâmetros em um período de tempo razoável.
- Já a classe ***RandomizedSearchCV*** usa uma abordagem de ***busca aleatória***.
  - Diferentemente da classe ***GridSearchCV***, nem todos os valores de hiperparâmetros são testados, mas um número fixo de parâmetros é amostrado a partir das distribuições especificadas.
- As duas abordagens funcionam bem para problemas simples, mas demandam um tempo muito longo para problemas mais complexos.

Ajuste fino dos hiperparâmetros de um modelo

- Felizmente, existem muitas técnicas para explorar um espaço de busca com muito mais eficiência do que aleatoriamente ou exaustivamente.
- A ideia central por trás destas ideias é simples: quando uma região do espaço de busca se mostra boa, ela deve ser mais explorada.
- Esse processo leva a soluções boas em muito menos tempo.
- Algumas bibliotecas Python para otimização de hiperparâmetros são:
  - ***Optuna***: implementa abordagem de ***otimização Bayesiana***.
  - ***KerasTuner***: é parte do Keras e implementa abordagens de ***otimização Bayesiana, com parada antecipada e com busca aleatória***.
  - ***Sklearn-Deap***: implementa abordagem de ***otimização baseada em algoritmos evolutivos***.
  - ***XGBoost***: implementa ***otimização baseada no gradiente descendente***, onde o gradiente é calculado com relação aos hiperparâmetros.

[1] https://en.wikipedia.org/wiki/Hyperparameter_optimization

Ajuste fino dos hiperparâmetros de um modelo

- Ter uma ideia de quais valores são razoáveis para cada hiperparâmetro nos ajuda a construir um protótipo rápido e restringir o espaço de busca.
- Assim, na sequência, discutimos algumas diretrizes para escolher bons valores para alguns dos principais hiperparâmetros de uma rede neural.

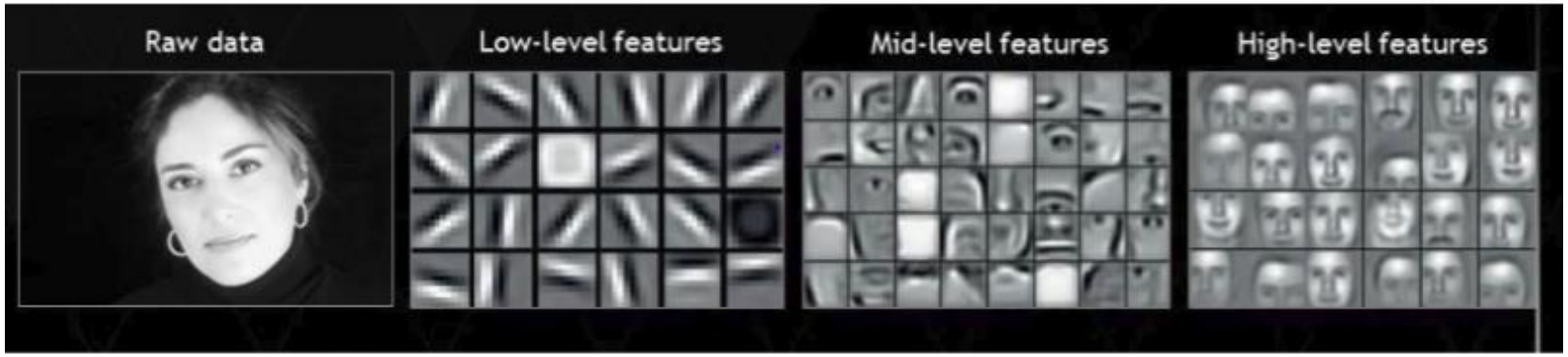
Número de camadas ocultas

- Para muitos problemas, nós podemos começar com apenas uma única camada oculta e mesmo assim obtermos resultados razoáveis.
- Na verdade, foi demonstrado que uma rede MLP com apenas uma única camada oculta pode modelar até as funções mais complexas, desde que a rede possua neurônios suficientes.
  - Funções contínuas são facilmente aproximadas por redes com uma camada, já funções com descontinuidades precisam de pelo menos uma segunda camada oculta.
- Por um longo tempo, esses fatos convenceram os pesquisadores de que não havia necessidade de se investigar redes neurais mais profundas.
- Entretanto eles ignoravam o fato de que as redes profundas têm uma eficiência paramétrica muito maior do que as rasas: elas podem modelar funções complexas usando exponencialmente menos neurônios do que as redes rasas, permitindo que alcancem um desempenho muito melhor com a mesma quantidade de dados de treinamento.

Ajuste fino dos hiperparâmetros de um modelo

Número de camadas ocultas

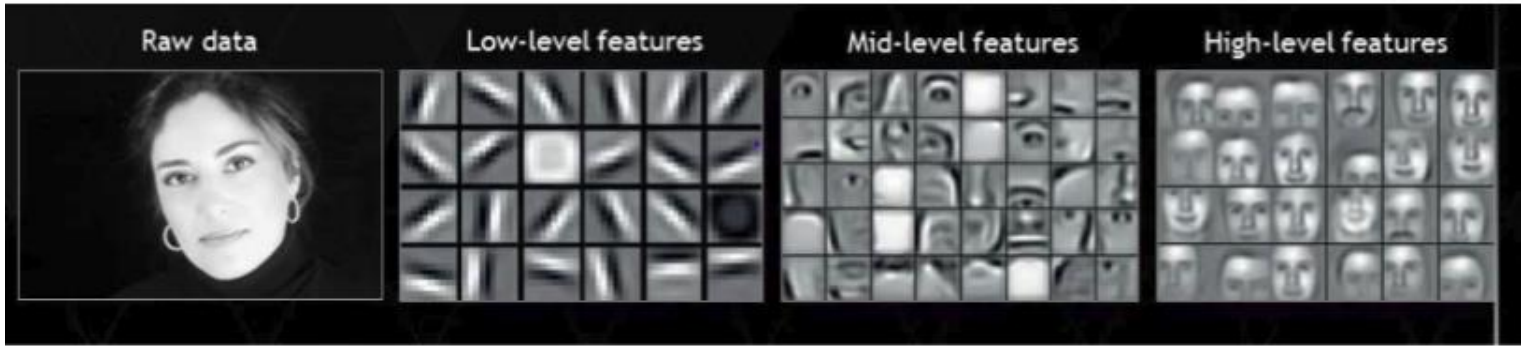
- Além disso, redes profundas tiram proveito da ***estrutura hierárquica*** presente em dados do mundo real.
- Por exemplo, em reconhecimento de faces, camadas ocultas próximas à entrada modelam características de baixo nível: segmentos de linha de várias formas e orientações.
- Camadas ocultas intermediárias combinam essas características de baixo nível para modelar características de nível intermediário: quadrados, círculos, etc.



Ajuste fino dos hiperparâmetros de um modelo

Número de camadas ocultas

- Por fim, as camadas ocultas mais próximas à saída juntamente com a camada de saída combinam essas características intermediárias para modelar características de alto nível: faces, objetos, animais, etc.
- Essa arquitetura hierárquica não apenas ajuda as redes profundas a convergirem mais rapidamente para uma boa solução, como também melhora a capacidade de generalização para ***novos conjuntos de dados***.



Ajuste fino dos hiperparâmetros de um modelo

Número de camadas ocultas

- Por exemplo, se já treinamos um modelo para reconhecer faces e agora desejamos treinar uma nova rede profunda para reconhecer estilos de cabelo, podemos iniciar o treinamento reutilizando as camadas inferiores (ou seja, mais próximas à entrada) da primeira rede.
- Em vez de inicializar aleatoriamente os pesos (sinápticos e de bias) das primeiras camadas da nova rede profunda, podemos inicializá-los com os pesos das camadas inferiores da primeira rede.
- Desta forma, uma rede neural não precisa aprender do zero todas as características de nível baixo/intermediário que ocorrem nos dados (e.g., imagens), ela pode reutilizar os pesos de camadas mais baixas (as quais já aprenderam as estruturas de nível baixo/intermediário) e precisará apenas aprender (treinar o restante das camadas) apenas as características de alto nível (e.g., estilos de cabelo).
- Isso é chamado de ***transfer learning***.

Ajuste fino dos hiperparâmetros de um modelo

Número de camadas ocultas

- Em resumo, para muitos problemas, nós podemos começar com apenas uma ou duas camadas ocultas e com isso obteremos bons resultados.
- Para problemas mais complexos, podemos aumentar gradualmente o número de camadas ocultas, até que a rede comece a sobreajustar demais ao conjunto de treinamento.
- Entretanto, nós raramente precisamos treinar redes profundas do zero: é muito mais comum reutilizar partes de uma rede pré-treinada que executa uma tarefa semelhante.
- Desta forma, o treinamento será muito mais rápido e exigirá muito menos dados.

Ajuste fino dos hiperparâmetros de um modelo

Número de neurônios por camada

- Obviamente, o número de neurônios na camada de saída é determinado pelo tipo de saída que uma determinada tarefa exige.
  - Por exemplo, uma tarefa de classificação com 10 classes exige 10 neurônios na camada de saída. Já uma tarefa de regressão unidimensional, exige apenas um neurônio.
- Quanto às camadas ocultas, uma prática comum é dimensioná-las para formar um ***funil***, com cada vez menos neurônios em cada camada.
- A raciocínio por trás dessa abordagem é que muitas características de baixo nível se unem a um número muito menor de características de alto nível.

Ajuste fino dos hiperparâmetros de um modelo

Número de neurônios por camada

- Assim como no número de camadas, nós podemos tentar aumentar gradualmente o número de neurônios por camada até que a rede comece a ***sobreajustar***.
- Em geral, obtem-se mais retorno aumentando-se o número de camadas do que o número de neurônios por camada.
- Uma abordagem mais simples é escolher um modelo com mais camadas e neurônios do que se realmente precisa e usar alguma técnica de regularização, como o ***early stop*** para evitar que a rede se ***sobreajuste***.

Ajuste fino dos hiperparâmetros de um modelo

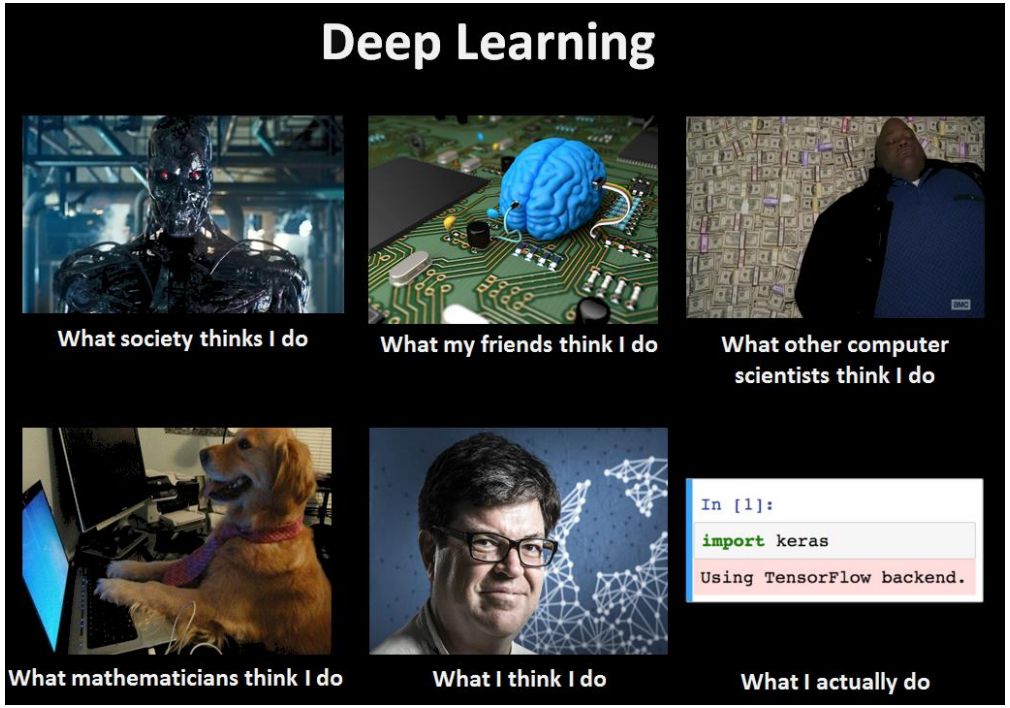
Funções de ativação

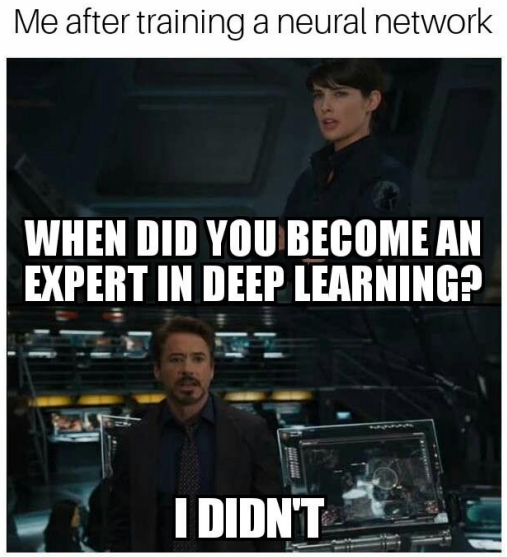
- Na maioria dos casos, principalmente com redes profundas, podemos usar a ***função de ativação*** do tipo ***ReLU*** ou suas variantes nas camadas ocultas.
- A função ***ReLU*** é um pouco mais rápida de se calcular do que outras funções de ativação.
- Além disso, a probabilidade do ***gradiente descendente*** ficar preso em platôs é menor, graças ao fato de a função ***ReLU*** não saturar para valores de entrada grandes (em oposição às ***funções de ativação logística*** ou ***tangente hiperbólica***, que saturam em 1).
- A função de ativação do tipo ***softmax*** é geralmente uma boa opção para a camada de saída em tarefas de classificação (quando as classes são mutuamente exclusivas, ou seja, quando um exemplo pertence somente a uma classe).
- Para tarefas de regressão, não usamos nenhuma função de ativação, que é também chamada de ***função de ativação*** do tipo ***identidade***.

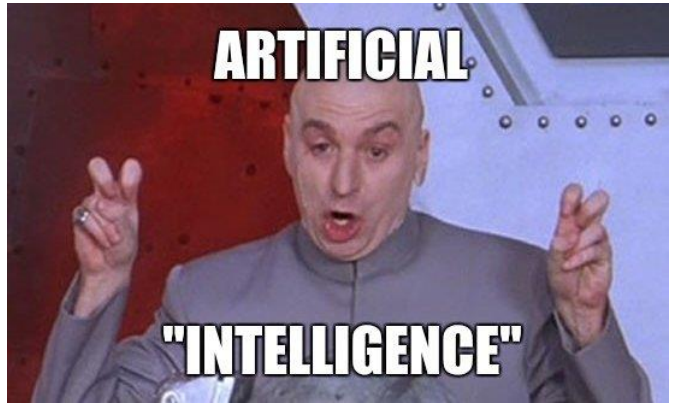
Avisos

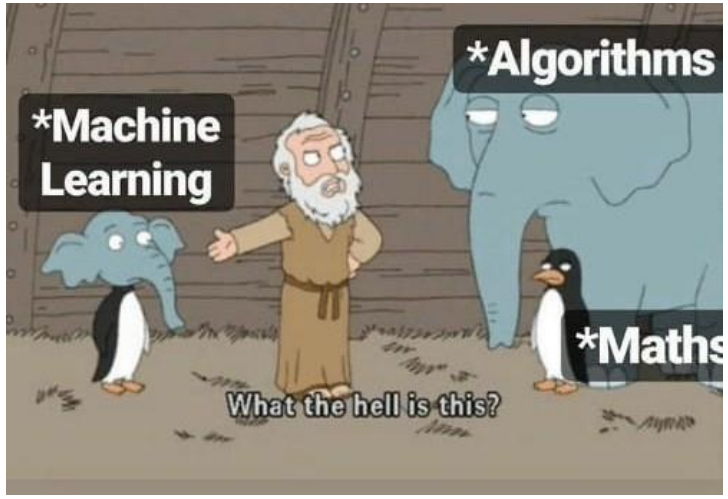
- A lista de exercícios #12 já pode ser feita.

Obrigado!


Deep Learning
What society thinks I do
What my friends think I do
What other computer scientists think I do
What mathematicians think I do
What I think I do
In [1]:
import keras
Using TensorFlow backend.
What I actually do

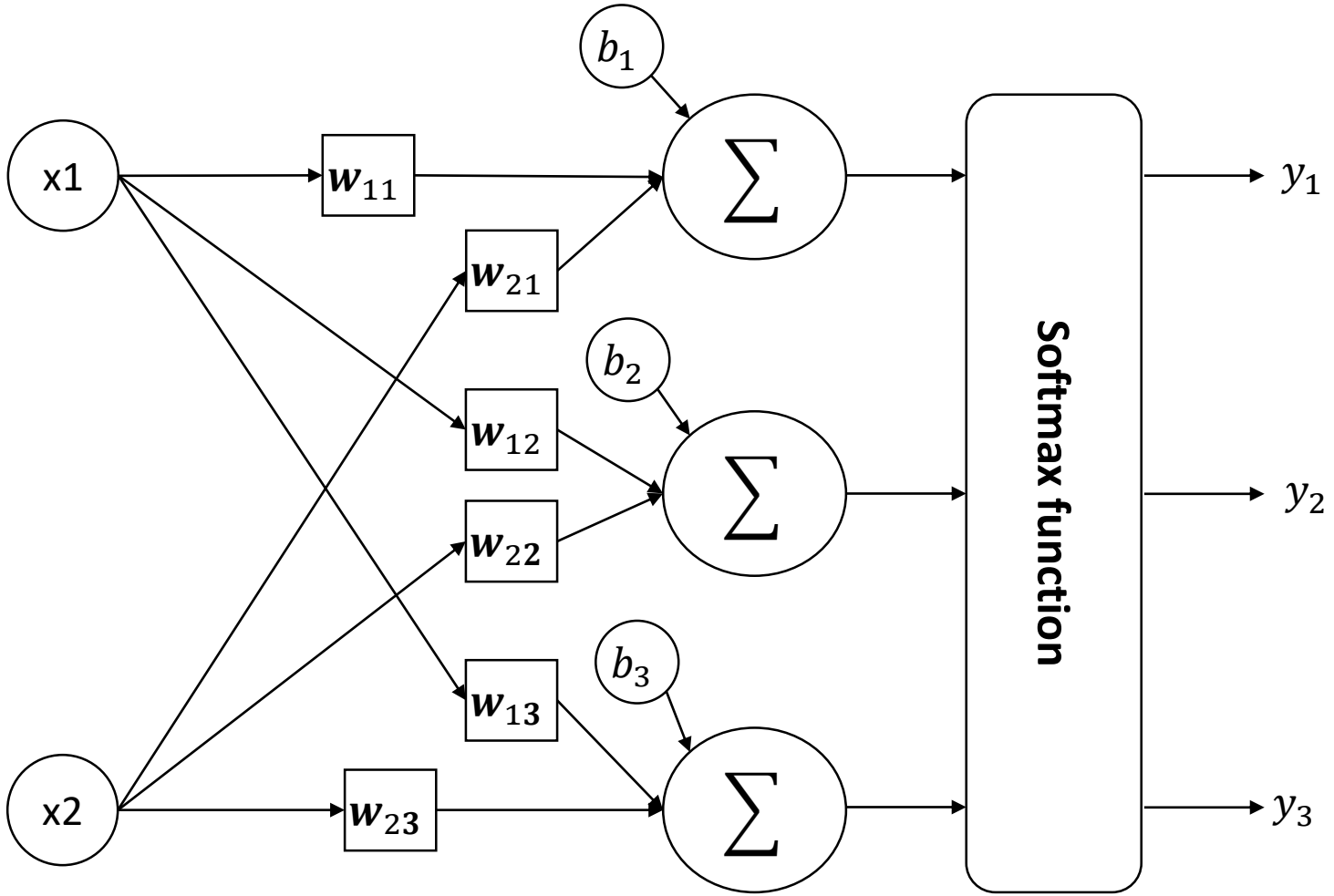

Me after training a neural network
WHEN DID YOU BECOME AN EXPERT IN DEEP LEARNING?
I DIDN'T

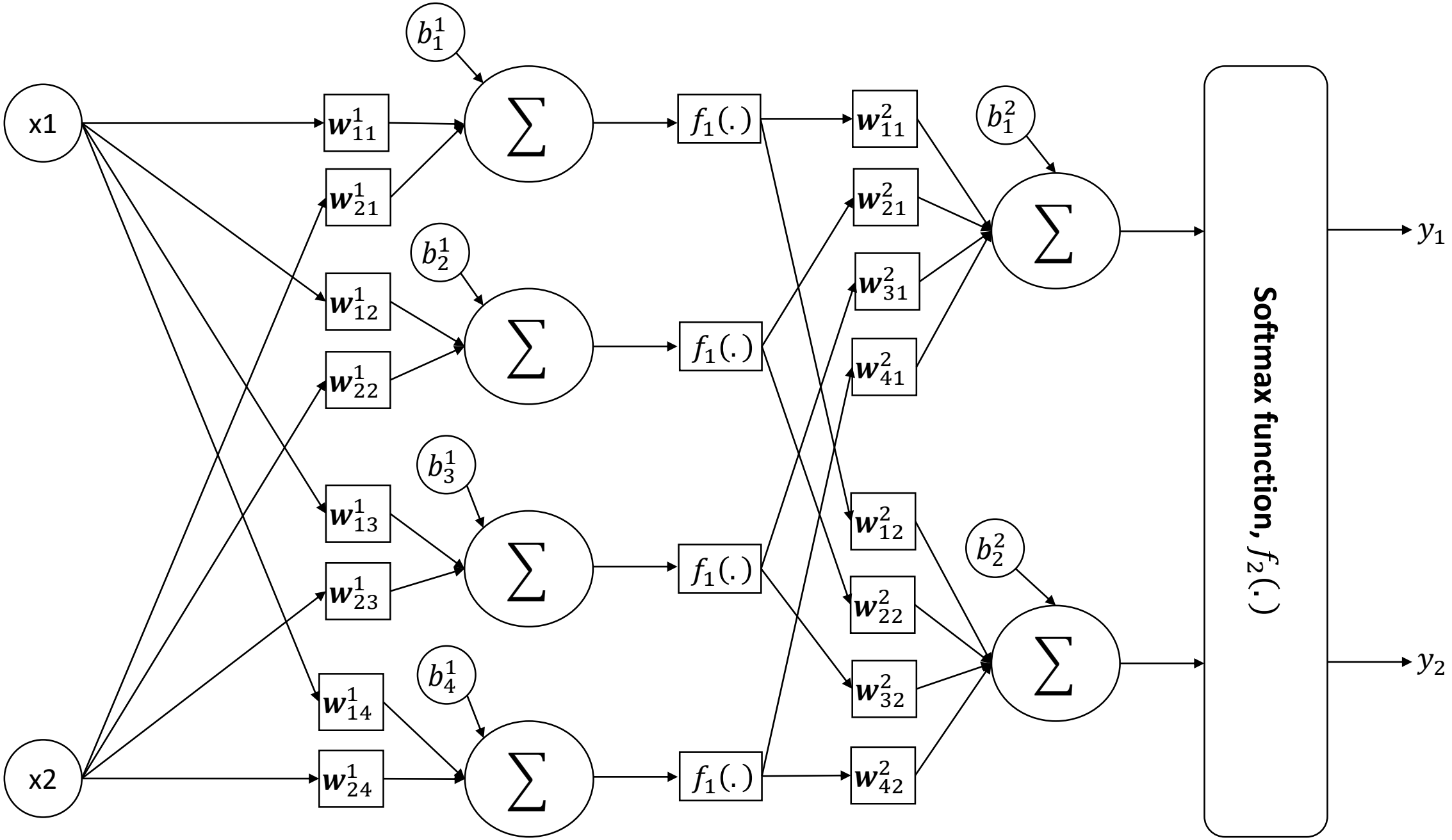

ARTIFICIAL
"INTELLIGENCE"


*Machine Learning
*Algorithms
*Maths
What the hell is this?


Me
Deep learning
Simple problem


Me spending four weeks training a model to 99.9% accuracy and then getting 10% on the test set
stonks


x1
x2
w_{11}
w_{21}
w_{12}
w_{22}
w_{13}
w_{23}
b_1
b_2
b_3
$\sum$
$\sum$
$\sum$
Softmax function
y_1
y_2
y_3


x1
x2
b_1^1
b_2^1
b_3^1
b_4^1
w_{11}^1
w_{21}^1
w_{12}^1
w_{22}^1
w_{13}^1
w_{23}^1
w_{14}^1
w_{24}^1
$\sum$
$f_1(.)$
w_{11}^2
w_{21}^2
w_{31}^2
w_{41}^2
w_{12}^2
w_{22}^2
w_{32}^2
w_{42}^2
b_1^2
b_2^2
Softmax function, $f_2(.)$
y_1
y_2