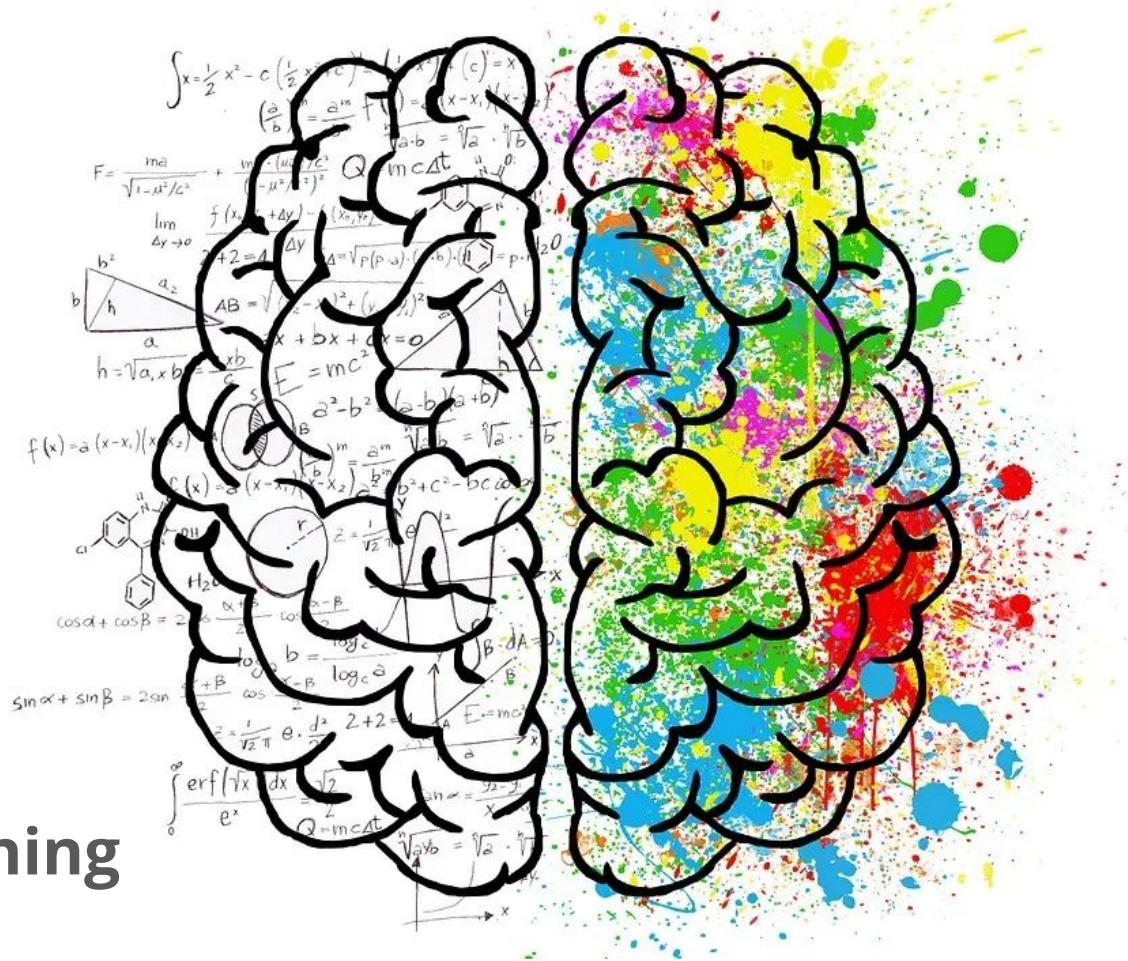
SCC0270 - Neural Networks and Deep Learning

Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo

Professor: Tiago Santana de Nazare
E-mail: tiagosn@alumni.usp.br

1º semestre de 2023

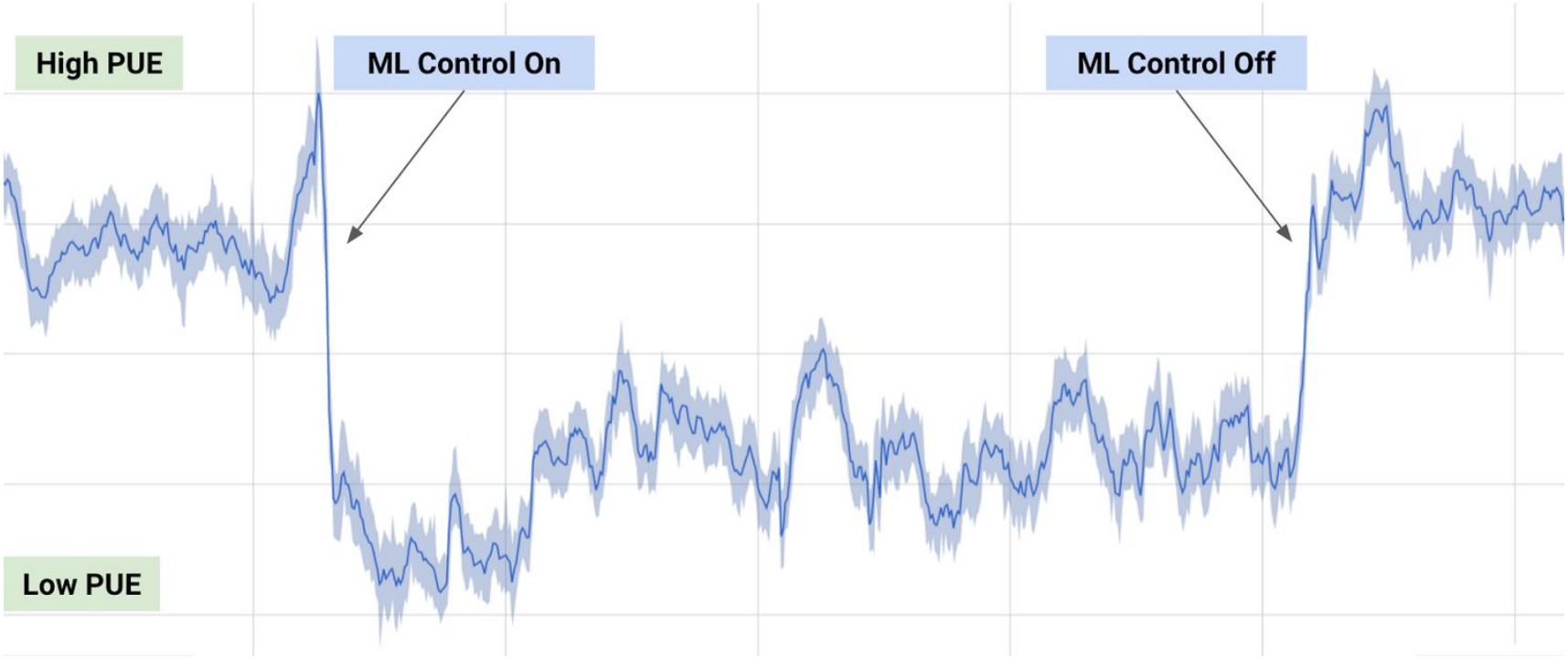


Deep Learning

Algumas aplicações

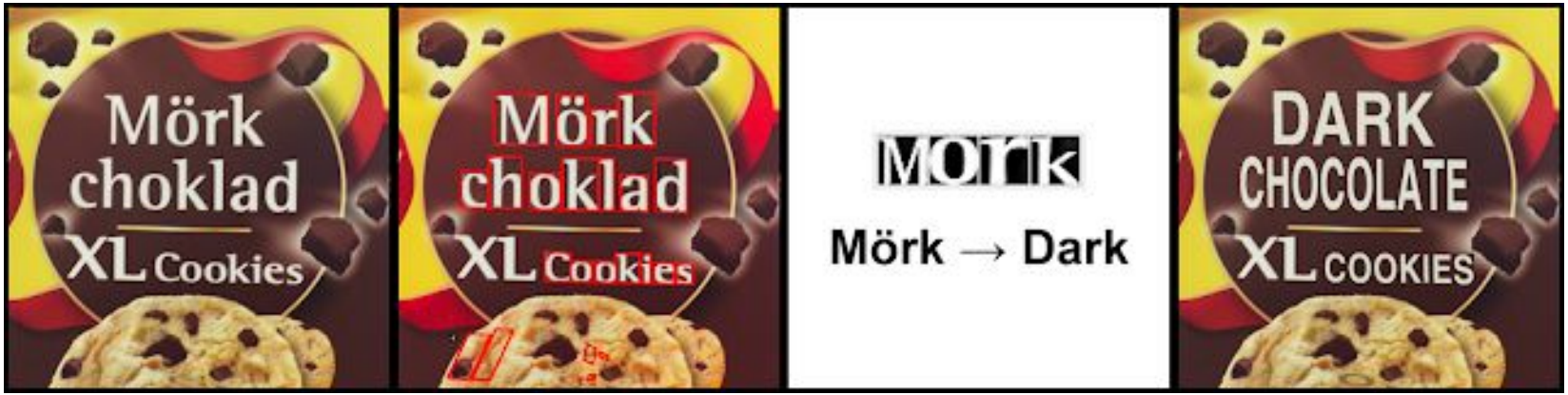

Detecção de veículos

Reduzir a conta de luz



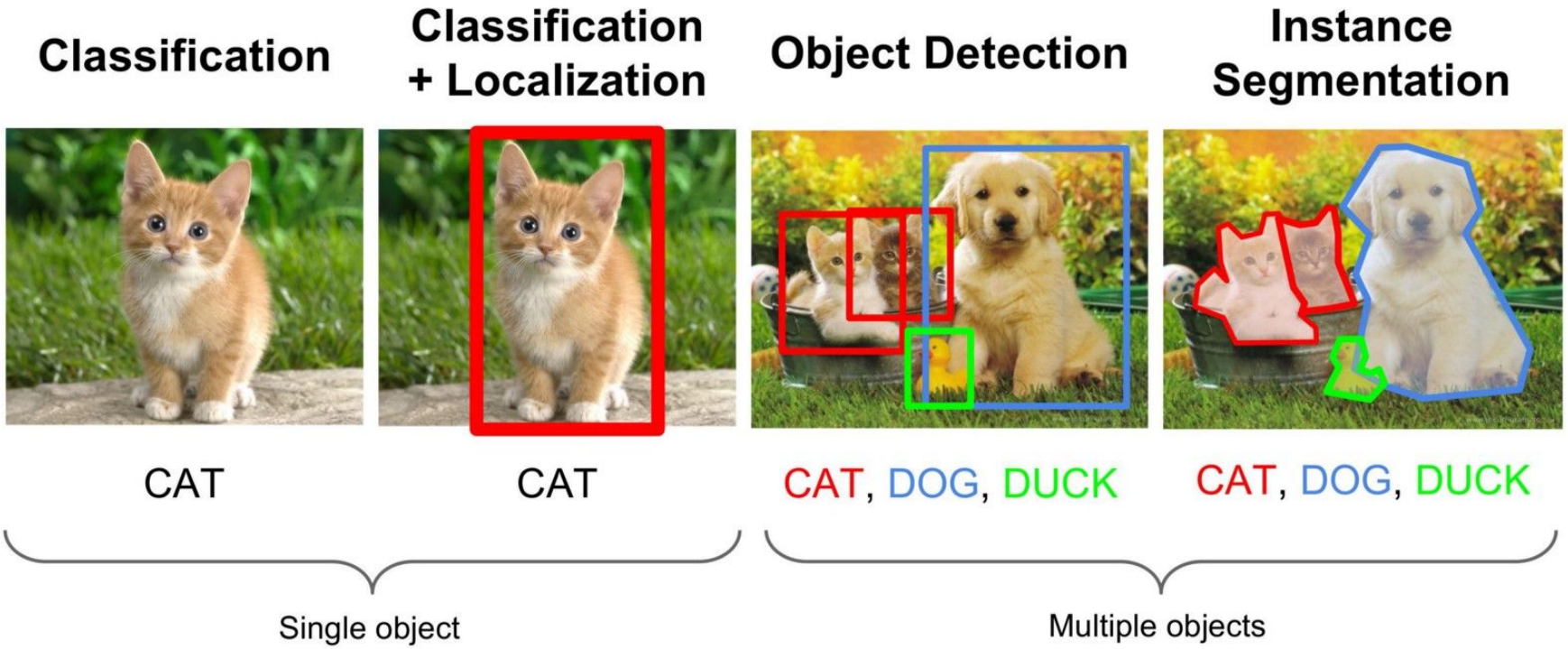
https://deepmind.com/blog/deepmind-ai-reduces-google-data-centre-cooling-bill-40/

“Traduzir” imagens

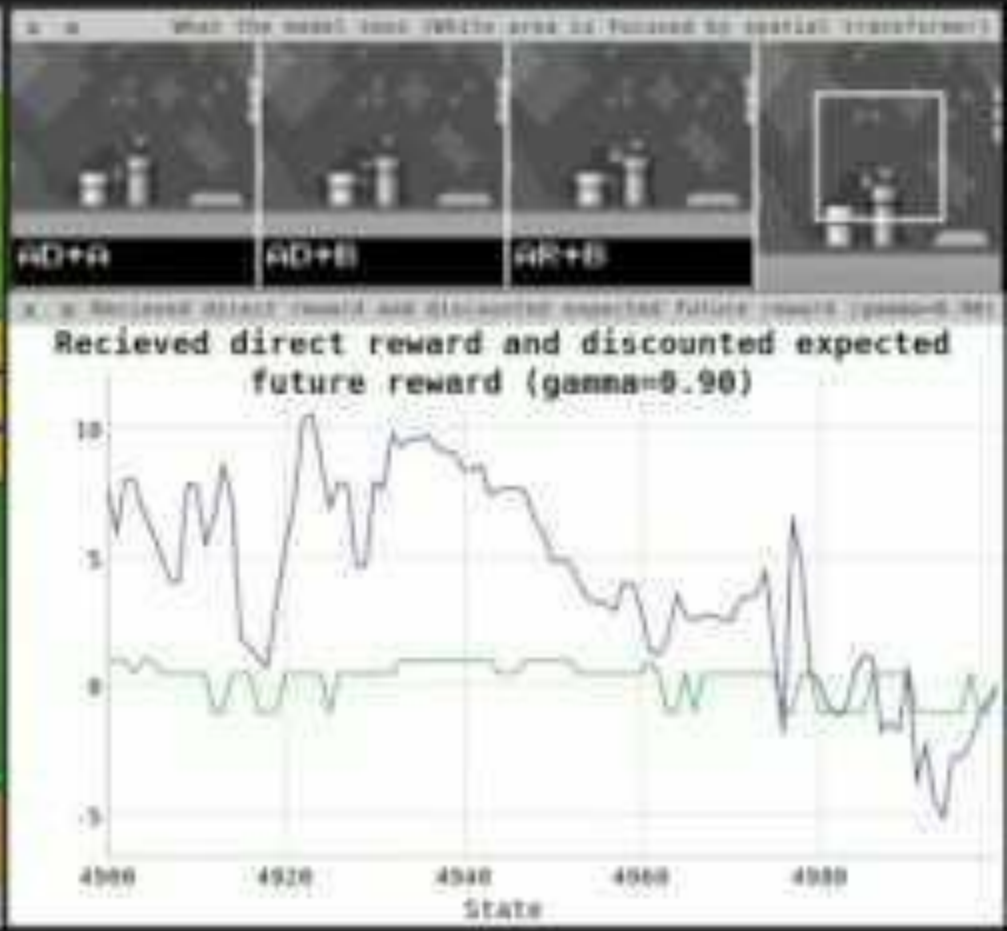


https://research.googleblog.com/2015/07/how-google-translate-squeezes-deep.html

Modelar bases de imagens

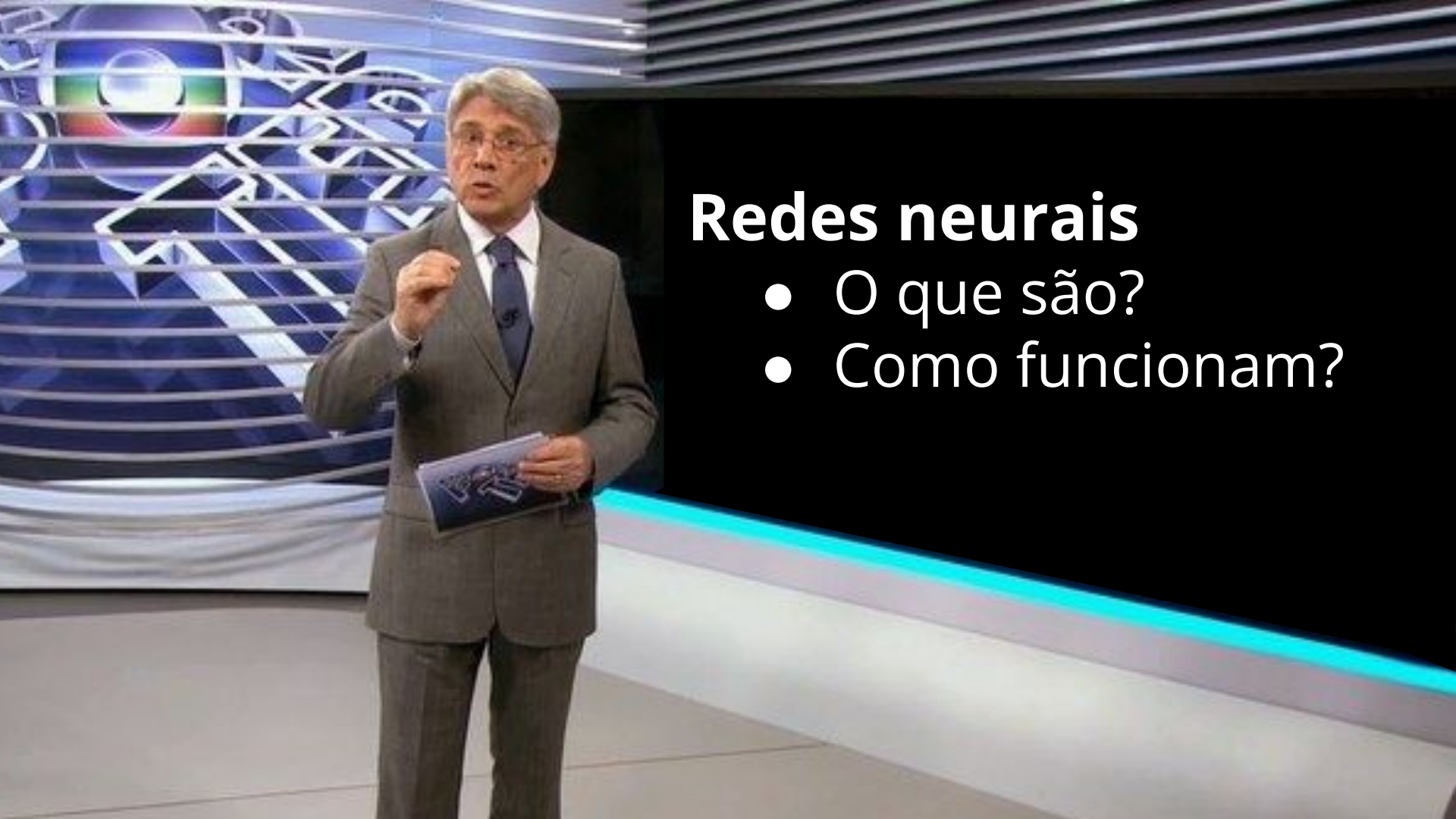



File DMS Movie Speed Tools Configure Help
TIME
4430
Jogar Super Mario


Recieved direct reward and discounted expected future reward (gamma=0.90)
10
5
0
-5
4900
4920
4940
4960
4980
State

Dançar como o Bruno Mars


Source Video
Detected Pose
Source to Target 1 Result
Source to Target 2 Result



Redes neurais

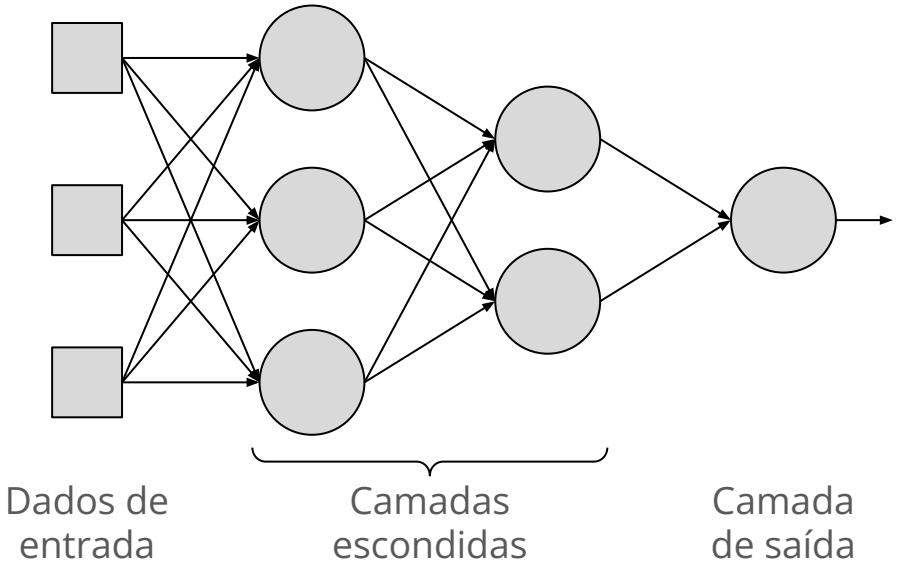
- O que são?
- Como funcionam?

Arquitetura

Entrada: linha da base de dados
Saída: previsão

Neurônios organizados em camadas:

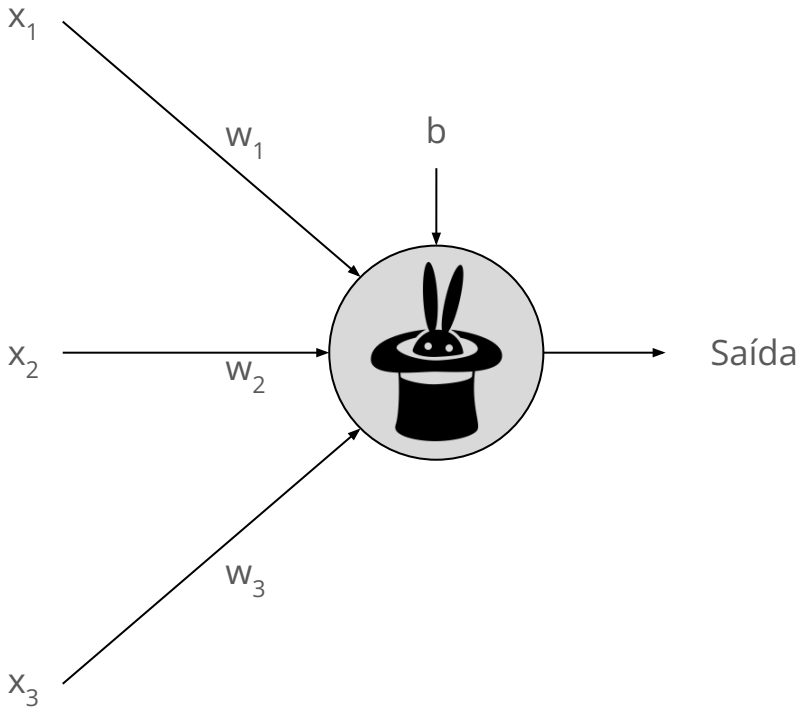
- Conectam com a próxima camada
- Geram uma resposta só (copiada)



Neurônio artificial

Entrada: linha da base de dados ou saída dos neurônios da camada anterior (x_i)
Saída: uma combinação das entradas

- Cada ligação tem um peso (w_i)
- Cada neurônio tem um bias (b)



A saída do neurônio é igual a:

$$f(\sum_i x_i w_i + b) = f(x_1 w_1 + x_2 w_2 + x_3 w_3 + b)$$

Função de ativação

É a função que se aplica a soma ponderada feita pelo neurônio:

$$\sum_i x_i w_i + b$$

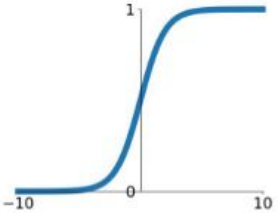
Para classificação e bastante comum usar:

- ReLU para as camadas escondidas
- Softmax para a última camada

$$\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}$$

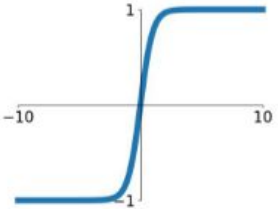
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



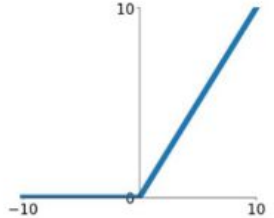
tanh

$$\tanh(x)$$



ReLU

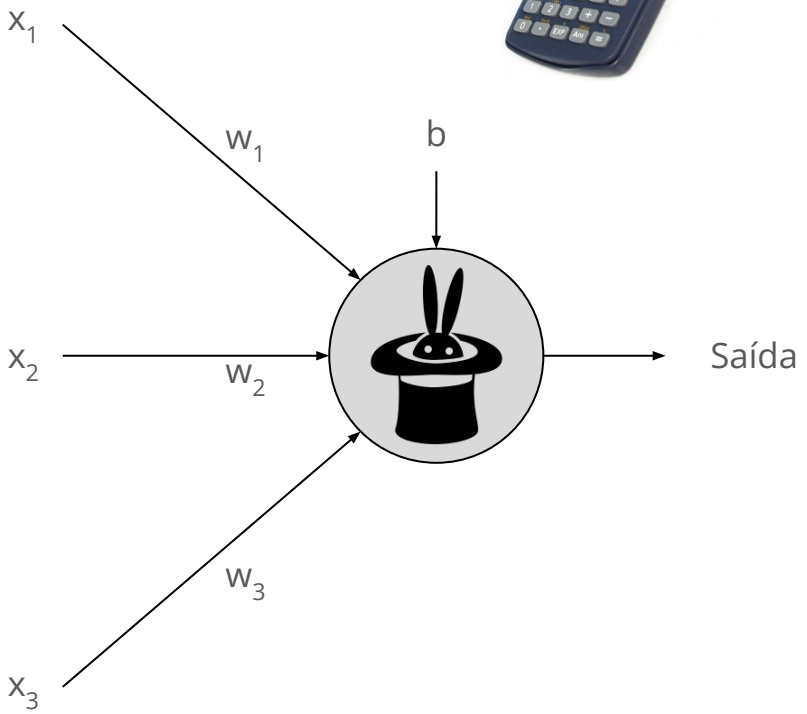
$$\max(0, x)$$



Exemplos cálculo de ativação

Caso 1:

- Ativação: ReLU
- [x1, x2, x3] = [1, 4, 2]
- [w1, w2, w3] = [0.5, -0.8, 0.4]
- b = 0.1



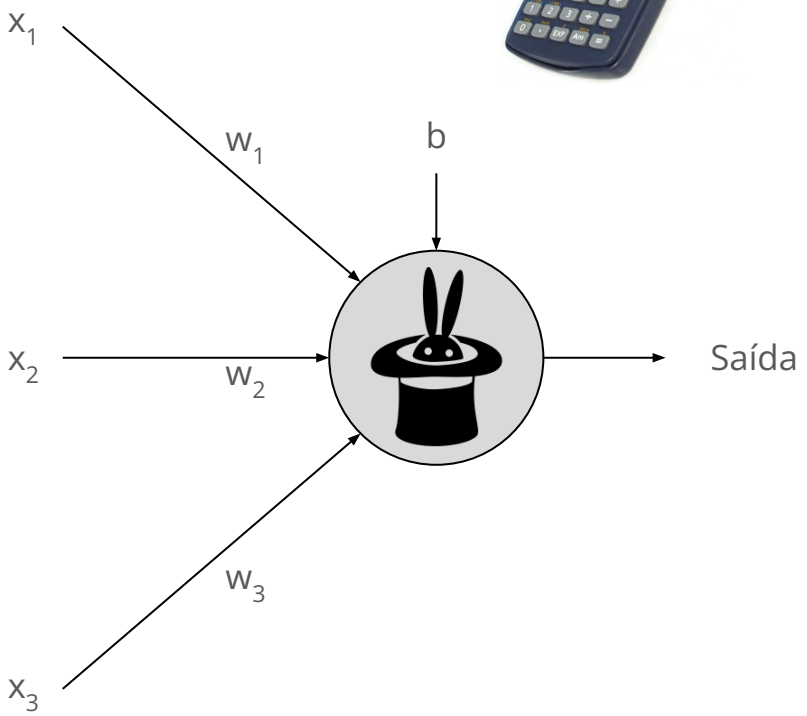
Resposta:

ReLU(1x0.5 + 4x(-0.8) + 2x0.4 + 0.1)
= ReLU(0.5 - 3.2 + 0.8 + 0.1)
= ReLU(-1.8) = **0**

Exemplos cálculo de ativação

Caso 2:

- Ativação: ReLU
- [x1, x2, x3] = [5, 2, 3]
- [w1, w2, w3] = [-0.2, 0.4, 0.7]
- b = -0.2



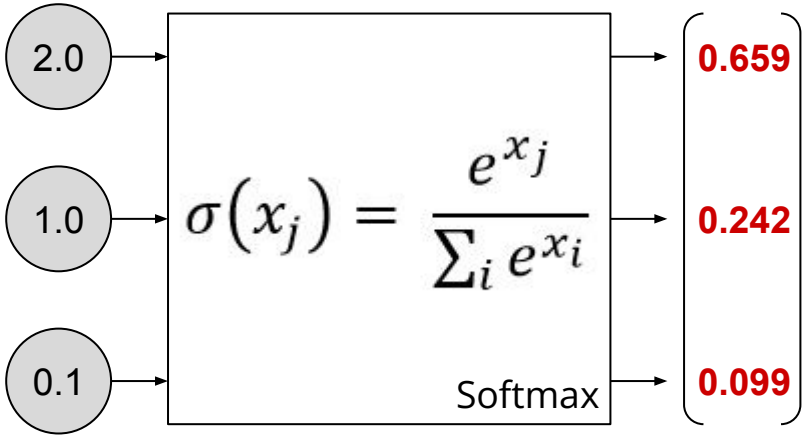
Resposta:

ReLU(5x(-0.2) + 2x(0.4) + 3x0.7 - 0.2)
= ReLU(-1 + 0.8 + 2.1 - 0.2)
= ReLU(1.7) = **1.7**

Exemplos cálculo de ativação

Resposta:

- $e^{2.0} + e^{1.0} + e^{0.1} = 11.21$
- $e^{2.0} = 7.39$
- $e^{1.0} = 2.72$
- $e^{0.1} = 1.11$



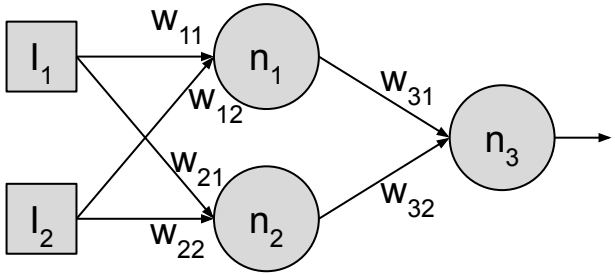
Porque precisamos de uma função de ativação?

Queremos prever o quanto de dinheiro vamos ter no fim do mês

- pode ser um valor positivo (sobrou dinheiro)
- pode ser zero
- ou negativo...

Arquitetura proposta para resolver o problema

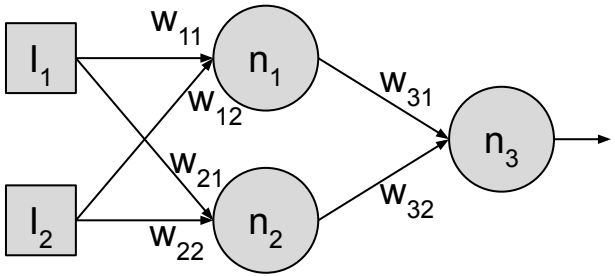
- $out(n_1) = w_{11}I_1 + w_{12}I_2 + b_1$
- $out(n_2) = w_{21}I_1 + w_{22}I_2 + b_2$
- $out(n_3) = w_{31}out(n_1) + w_{32}out(n_2) + b_3$



Porque precisamos de uma função de ativação?

"Sem função de ativação" (identidade):

- $out(n_1) = w_{11}I_1 + w_{12}I_2 + b_1$
- $out(n_2) = w_{21}I_1 + w_{22}I_2 + b_2$
- $out(n_3) = w_{31}out(n_1) + w_{32}out(n_2) + b_3$



Vamos olhar melhor para $out(n_3)$:

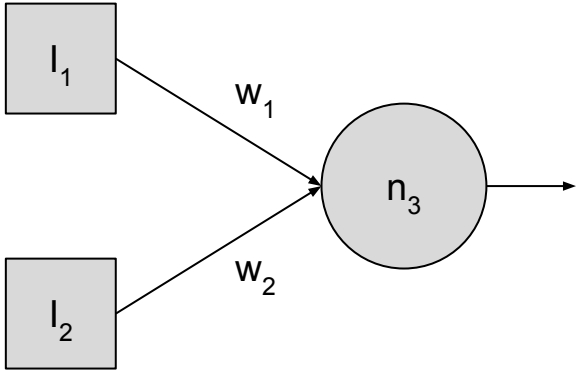
$$
\begin{aligned}
out(n_3) &= w_{31}out(n_1) + w_{32}out(n_2) + b_3 \\
&= w_{31}(w_{11}I_1 + w_{12}I_2 + b_1) + w_{32}(w_{21}I_1 + w_{22}I_2 + b_2) + b_3 \\
&= (w_{31}w_{11} + w_{32}w_{21})I_1 + (w_{31}w_{12} + w_{32}w_{22})I_2 + (w_{31}b_1 + w_{32}b_2 + b_3)
\end{aligned}
$$

Pode ser representado por: $w_1I_1 + w_2I_2 + b$

Porque precisamos de uma função de ativação?

"Sem função de ativação" (identidade):

- $out(n_1) = w_{11}I_1 + w_{12}I_2 + b_1$
- $out(n_2) = w_{21}I_1 + w_{22}I_2 + b_2$
- $out(n_3) = w_{31}out(n_1) + w_{32}out(n_2) + b_3$



Vamos olhar melhor para $out(n_3)$:

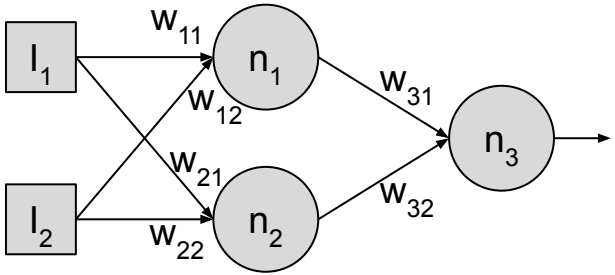
$$
\begin{aligned}
out(n_3) &= w_{31}out(n_1) + w_{32}out(n_2) + b_3 \\
&= w_{31}(w_{11}I_1 + w_{12}I_2 + b_1) + w_{32}(w_{21}I_1 + w_{22}I_2 + b_2) + b_3 \\
&= (w_{31}w_{11} + w_{32}w_{21})I_1 + (w_{31}w_{12} + w_{32}w_{22})I_2 + (w_{31}b_1 + w_{32}b_2 + b_3)
\end{aligned}
$$

Pode ser representado por: $w_1I_1 + w_2I_2 + b$

Porque precisamos de uma função de ativação?

Usando a ReLU:

- $out(n_1) = ReLU(w_{11}I_1 + w_{12}I_2 + b_1)$
- $out(n_2) = ReLU(w_{21}I_1 + w_{22}I_2 + b_2)$
- $out(n_3) = w_{31}out(n_1) + w_{32}out(n_2) + b_3$



Vamos olhar melhor para $out(n_3)$:

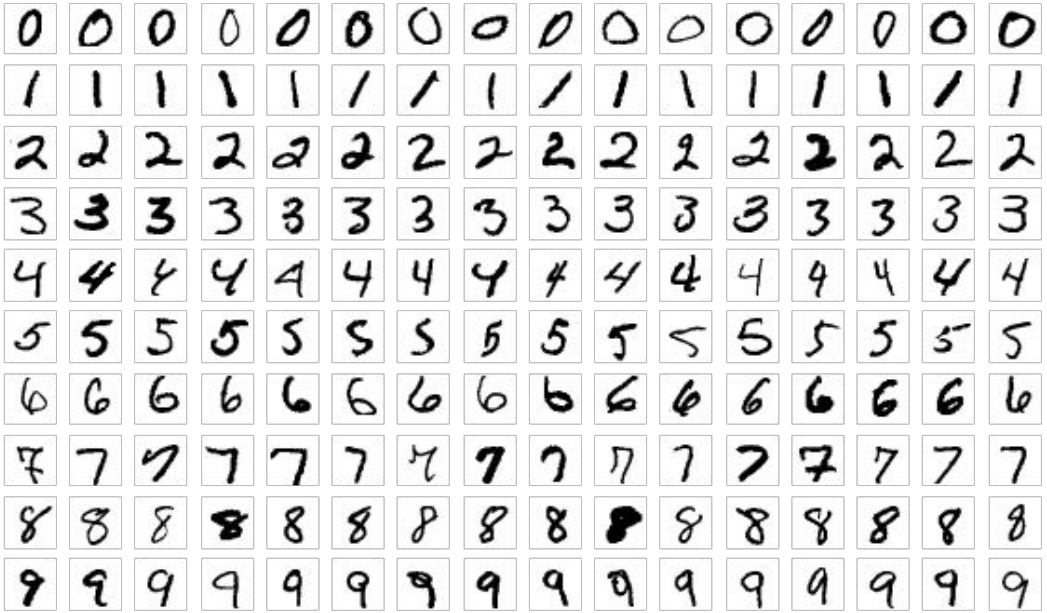
$$\begin{aligned} out(n_3) &= w_{31}out(n_1) + w_{32}out(n_2) + b_3 \\ &= w_{31}ReLU(w_{11}I_1 + w_{12}I_2 + b_1) + w_{32}ReLU(w_{21}I_1 + w_{22}I_2 + b_2) + b_3 \end{aligned}$$

Podemos simplificar?

Classificar dígitos usando redes neurais

Notebooks para estudo:

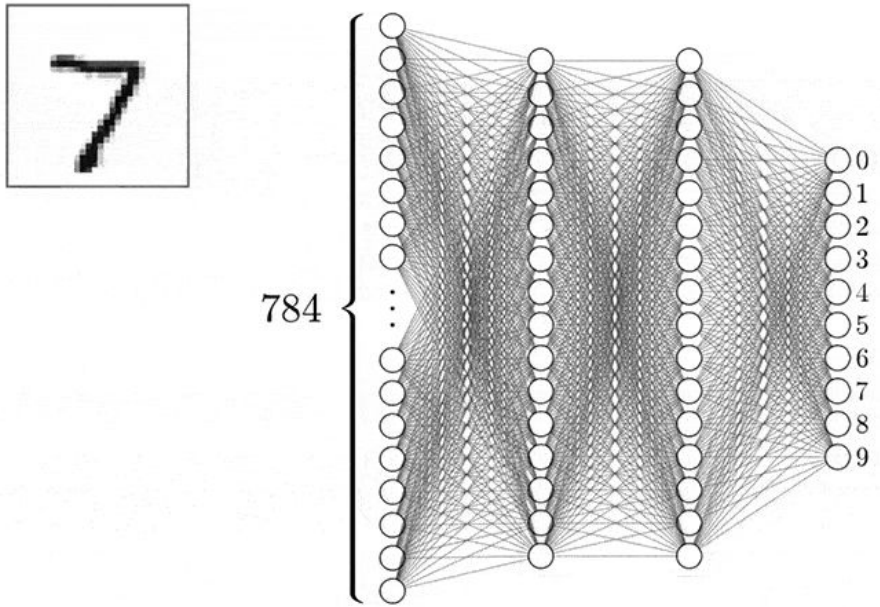
- sklearn_mnist_nn.ipynb
- keras_mnist_nn.ipynb



Treinamento dos modelos

Como uma rede neural aprende

Quando uma rede neural é boa?



Quando a rede produz o resultado correto

Ou seja: **quando os pesos da rede geram as respostas corretas**

Como obter bons pesos para uma rede?

Algoritmo:

1. Iniciar com pesos aleatórios
2. Classificar dados de treinamento
3. Ver onde erramos (medir o erro)
4. Modificar os pesos para melhorar

Medir o erro do modelo

Para medir o erro usamos uma função de custo:

- MSE para regressão:

$$(y_i - \hat{y}_i)^2$$

- Cross entropy para classificação:

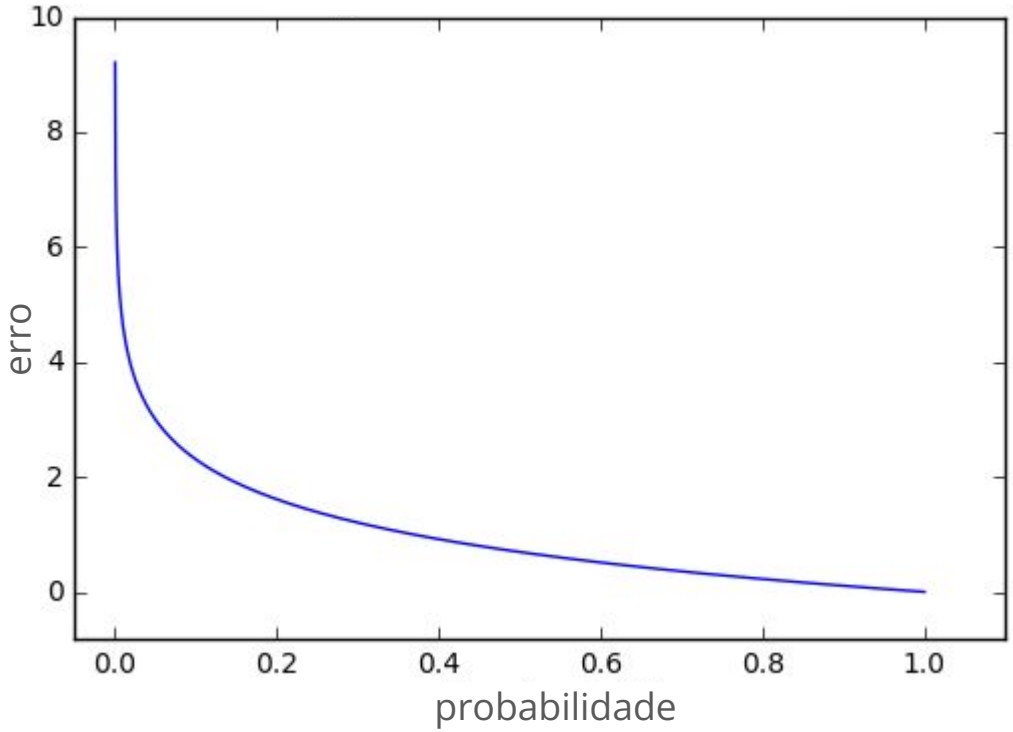
$$-\sum_{i=0}^{C} y_i.log(\hat{y}_i)$$

Fazer a média na amostra

Cross-entropy

Para y = [0, 1]

probs	erro
[0.99, 0.01]	4.61
[0.8, 0.2]	1.60
[0.6, 0.4]	0.92
[0.1, 0.9]	0.10
[0.01, 0.99]	0.01

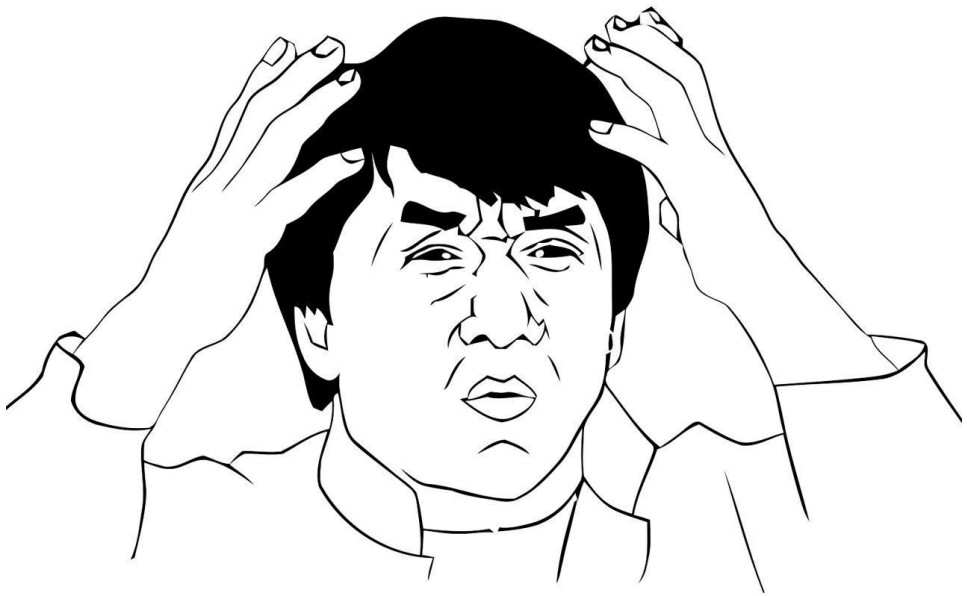


Gradiente descendente

Atualização dos parâmetros do modelo

$$w = w - \alpha \frac{\delta E}{\delta w}$$

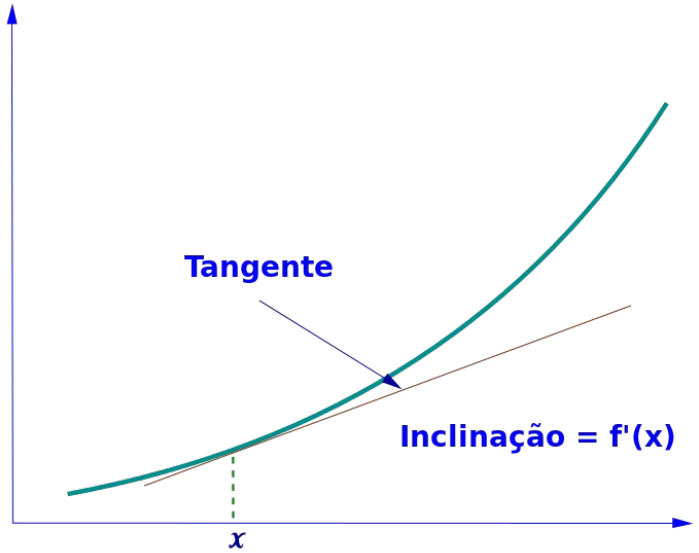
$$b = b - \alpha \frac{\delta E}{\delta b}$$

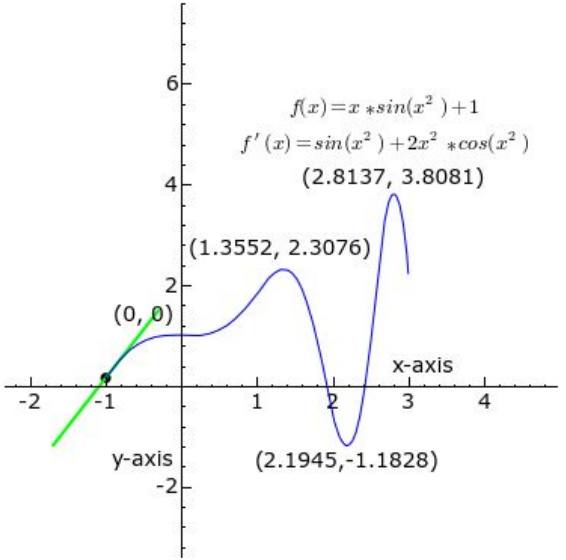


Derivada

$$w = w - \alpha \frac{\delta E}{\delta w}$$

É a taxa de variação da função em um dado ponto x



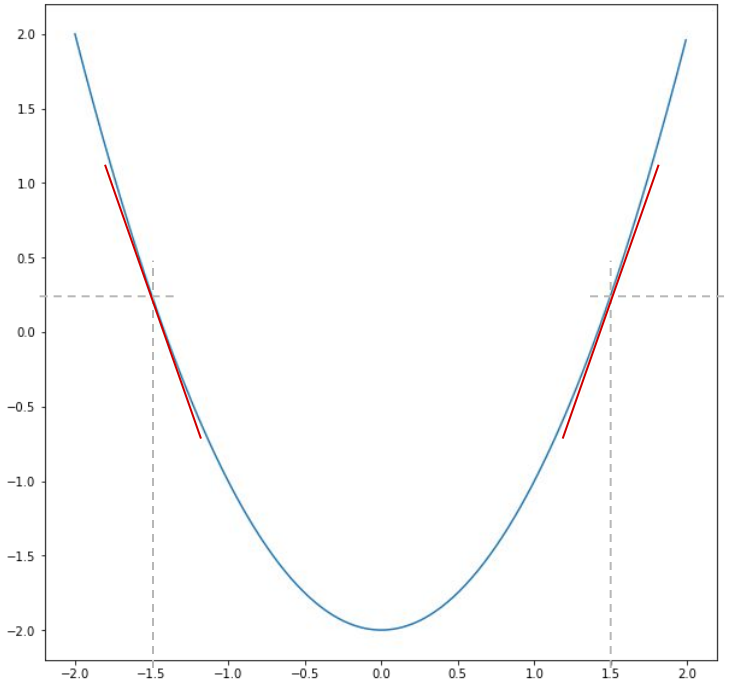


Derivada

$$w = w - \alpha \frac{\delta E}{\delta w}$$

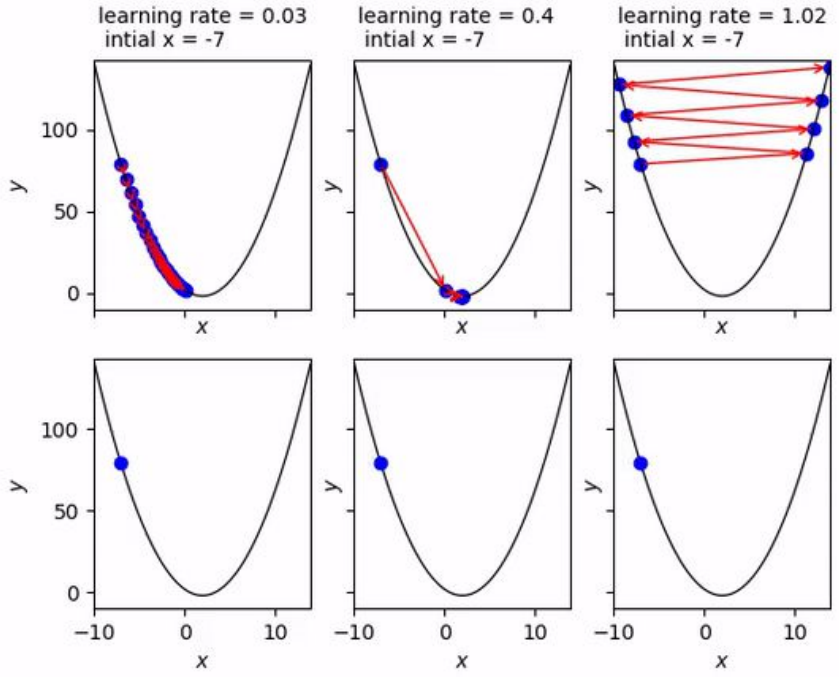
$$f(x) = x^2 - 2, \frac{\delta f(x)}{\delta x} = 2x$$

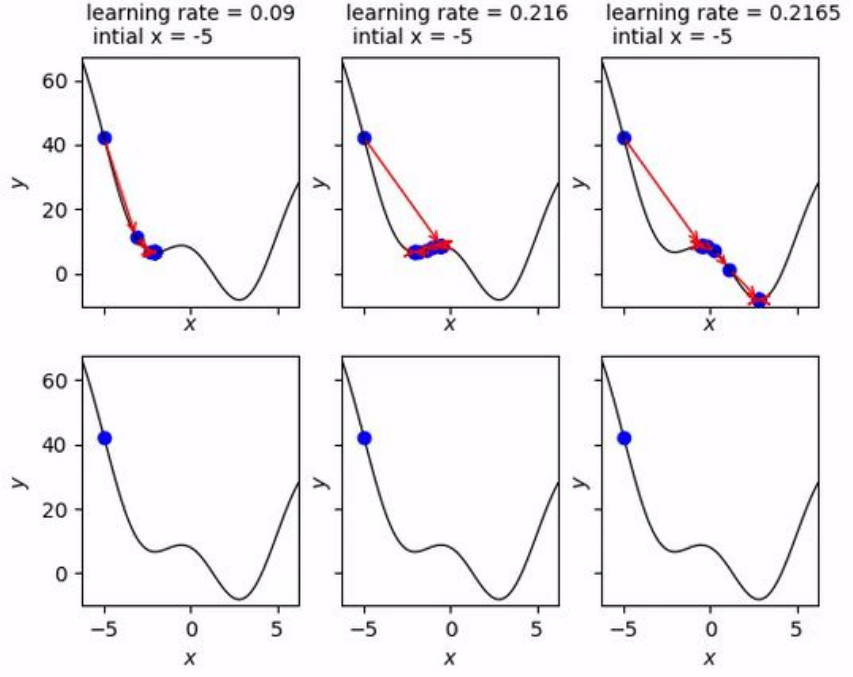
x	$\delta x = 2x$
-1.5	-3.0
0.0	0.0
1.5	3.0



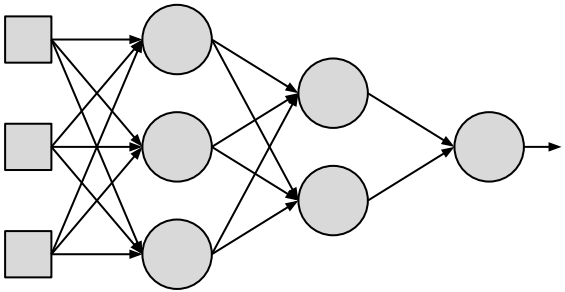
Learning rate

$$w = w - \alpha \frac{\delta E}{\delta w}$$





Resumo - Treinamento do modelo



Entrada:

- Rede neural com parâmetros **W** e **b**, para otimizar
- Um conjunto de treinamento $\mathbf{D}_{\mathbf{train}}$
- Um valor para o learning rate α

Repetir n vezes:

- Computar os erros da rede neural para todos os elementos de $\mathbf{D}_{\mathbf{train}}$
- Atualizar os parâmetros:

$$w = w - \alpha \frac{\delta E}{\delta w} \qquad b = b - \alpha \frac{\delta E}{\delta b}$$

Saída:

- Rede treinada

Tensorflow Playground

Hora de testar alguns conceitos:

https://playground.tensorflow.org

Tensorflow Playground - Atividade 1

Vamos rodar o exemplo padrão:

1. O que significa o termo *epoch* (época)?
2. O que acontece se trocarmos a função de ativação para ReLU?
3. Uma rede com 1 camada de 2 neurônios consegue resolver o problema?
    - E com 3 neurônios?
4. E uma rede gigante com ativação linear?

Tensorflow Playground - Atividade 2

1. Como podemos fazer uma rede de um neurônio ser capaz de modelar o problema???

2. Interprete os pesos dessa rede

3. Porque uma rede com vários neurônios consegue fazer o mesmo sem que precisemos fazer *feature engineering*???

Projetar um modelo

Deixar as coisas mais escaláveis

Batch

Grad. descendente: atualiza os pesos depois de ver o treinamento inteiro

- Pró: estima bem o gradiente (média com mais pontos)
- Contra: Demoraaaaaaa....

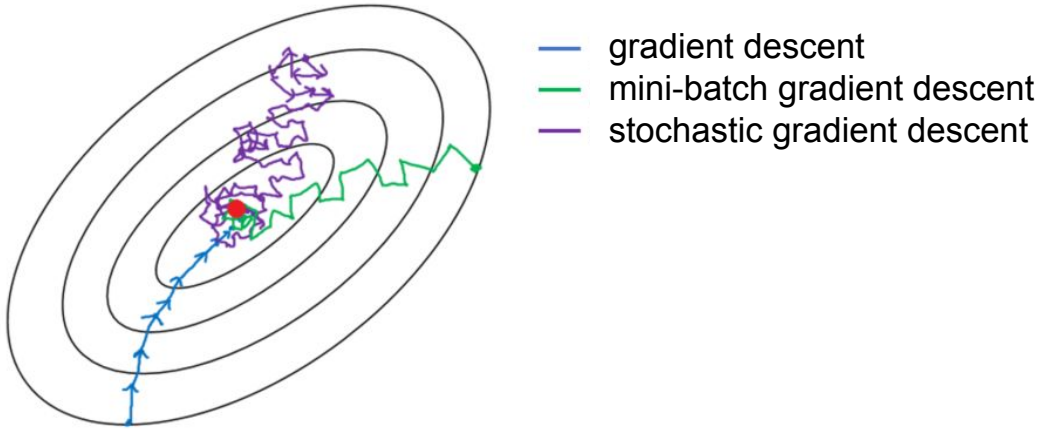
Versão mini-batch: atualiza os pesos depois de ver um batch de exemplos

- Pró: é bem mais rápido
- Contra: não estima o gradiente tão bem

Estocástico: atualiza os pesos depois de **um exemplo**

Batch

Versão	# exemplos para atualizar
gradiente descendente	todos do treinamento
mini-batch	tamanho do batch
estocástico	1



Regularização

Redes neurais têm muitos parâmetros:

- Pode ocorrer overfitting!!!
- Isso quer dizer que devemos abandonar redes mais profundas?!
  - Não =)

Regularização: penalizar a rede por usar complexidade sem precisar

$$\sum_i |w_i| \qquad \sum_i w_i^2$$

Tensorflow Playground - Regularização

Vamos criar uma rede grande no dataset linearmente separável:

1. Sem regularização
2. Com regularização L2
    - Variar o *regularization rate*
3. Com regularização L1
    - Variar o *regularization rate*

O que cada regularização fez com os pesos?

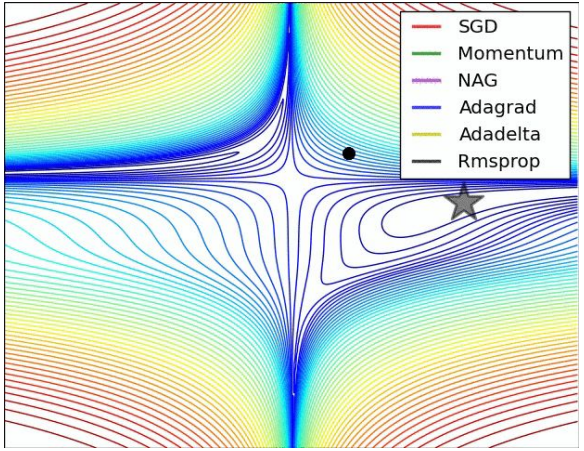
Outros otimizadores

Gradiente descendente atualiza os pesos usando um learning rate fixo:

- Pode ser muito lento no início e muito rápido no final

Existem outros otimizadores:

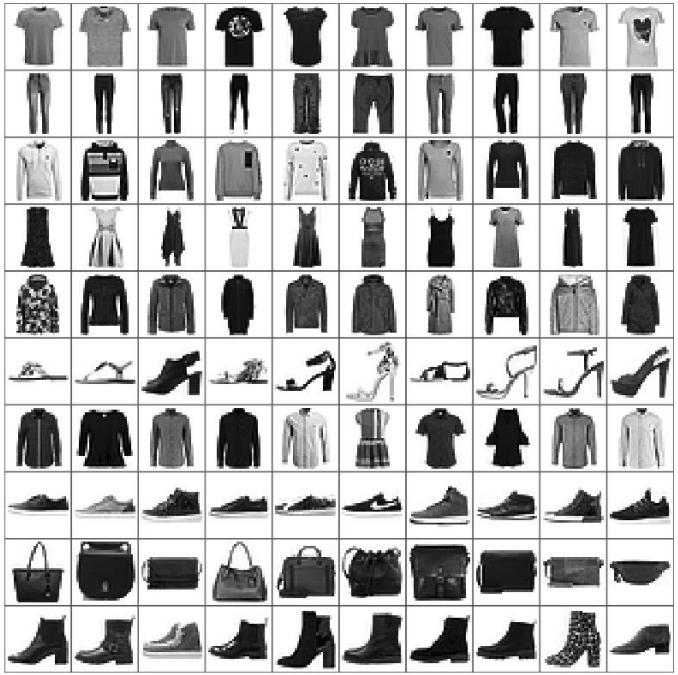
- Reduzem o learning rate ao longo do tempo
  - Adam
  - AdaGrad
  - RMSProp
  - ...



Classificar roupas usando redes neurais

Notebooks para estudo:

- sklearn_fashion-mnist_nn.ipynb
- keras_fashion-mnist_nn.ipynb



Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Links interessantes

- http://neuralnetworksanddeeplearning.com/ ← **Livro texto!!!**
- https://scikit-learn.org/

Scikit-learn - Redes neurais para regressão

Notebook para estudo:

- sklearn_ch_nn.ipynb

Scikit-learn - Classificação de textos

Notebook para estudo:

- sklearn_20newsgroups_nn.ipynb

Keras - Imagens coloridas (RGB)

Notebook para estudo:

- keras_cifar-10_nn.ipynb