PMR5251 - Avaliação do Comportamento Mecânico de Materiais Utilizando uma Abordagem de ML

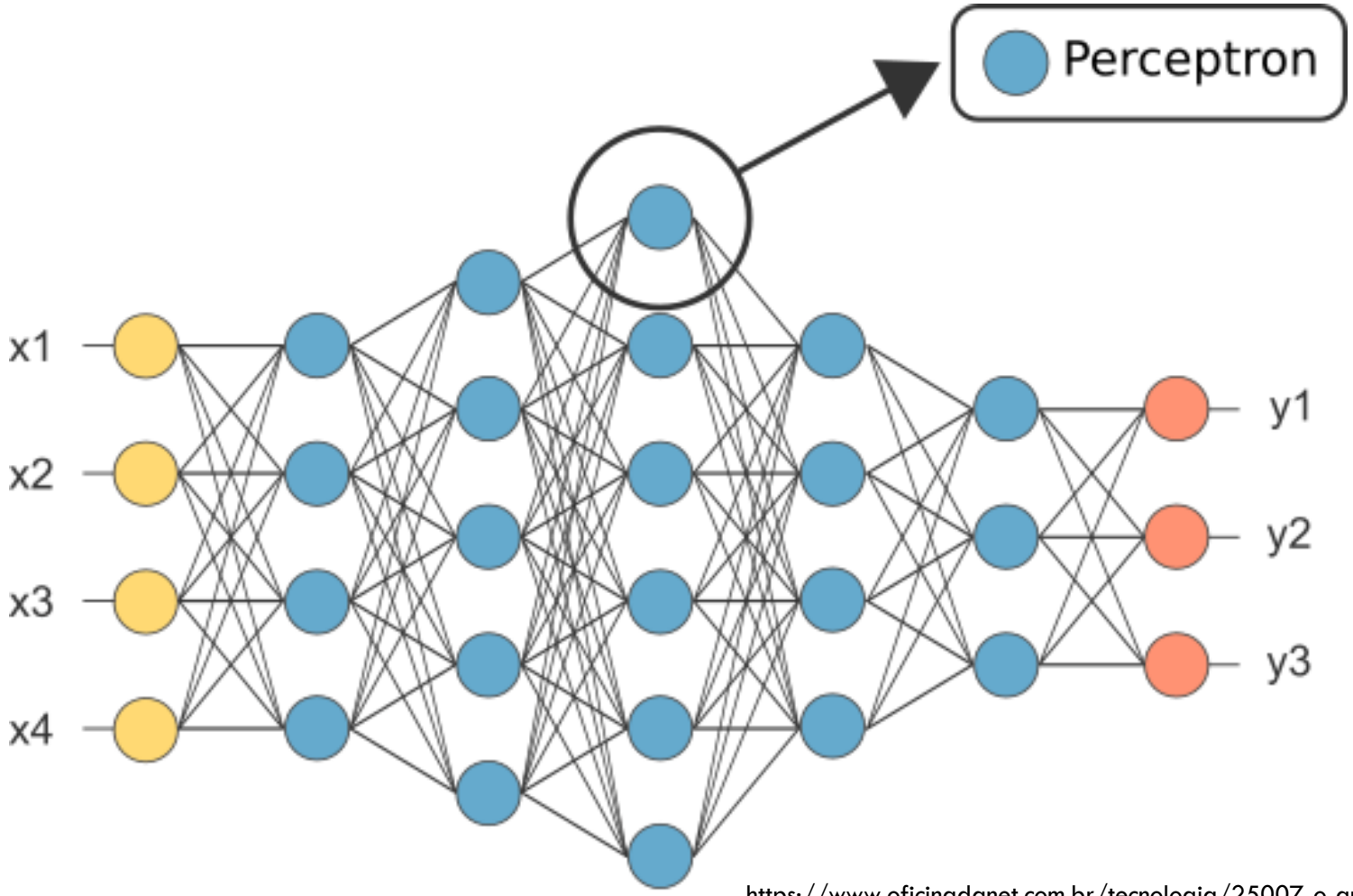
REDES NEURAIS ARTIFICIAIS (RNAs)

Izabel F. Machado
Larissa Driemeier

O QUE É UMA RNA?

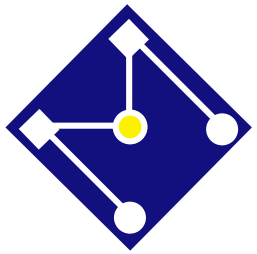
Definição
Estrutura

REDES NEURAIS



https://www.oficinadanet.com.br/tecnologia/25007-o-que-sao-as-redes-neurais-artificiais

PRINCIPAIS CARACTERÍSTICAS DAS REDES NEURAIS BIOLÓGICAS ESTÃO PRESENTES NAS RNAs

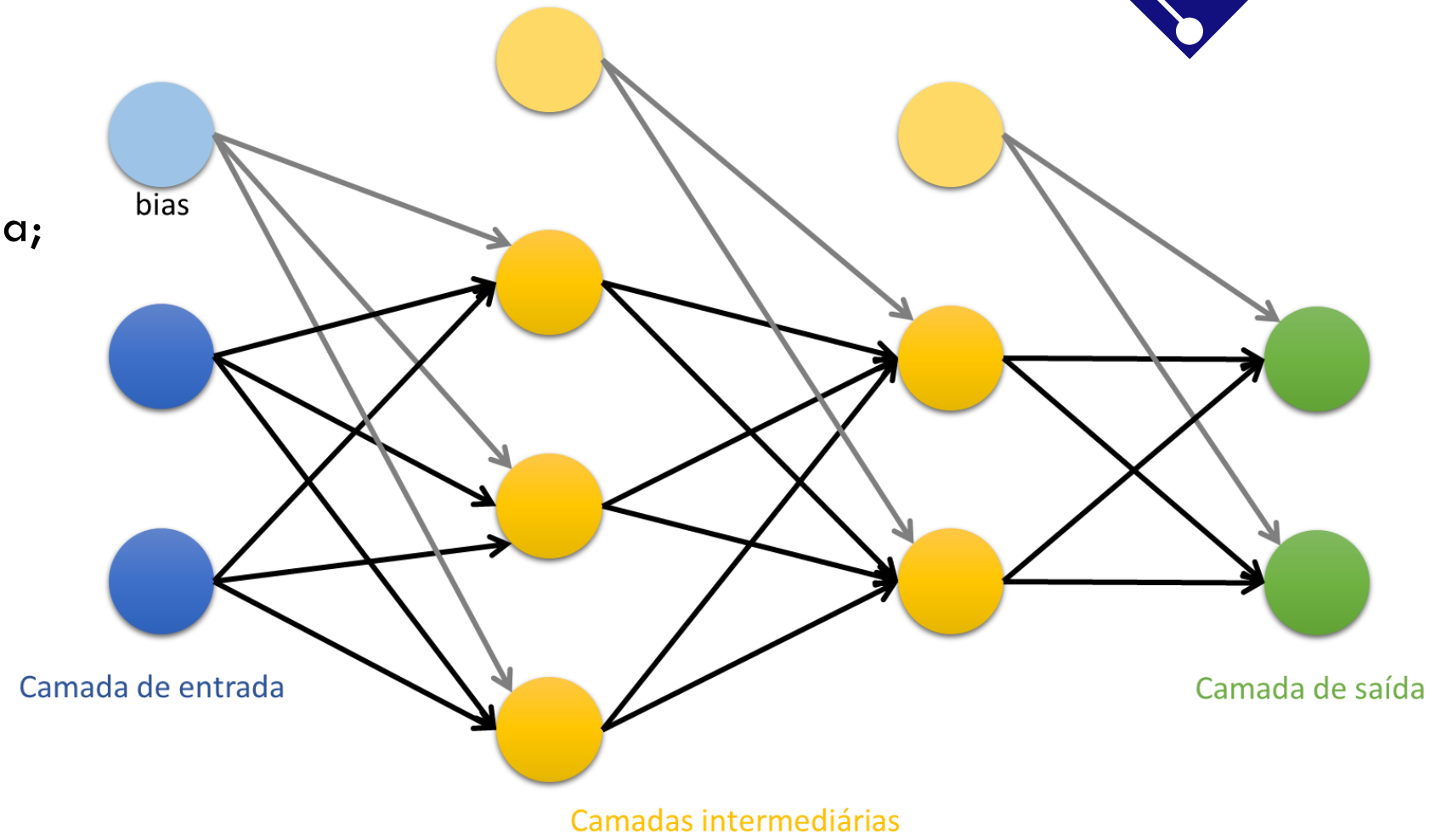


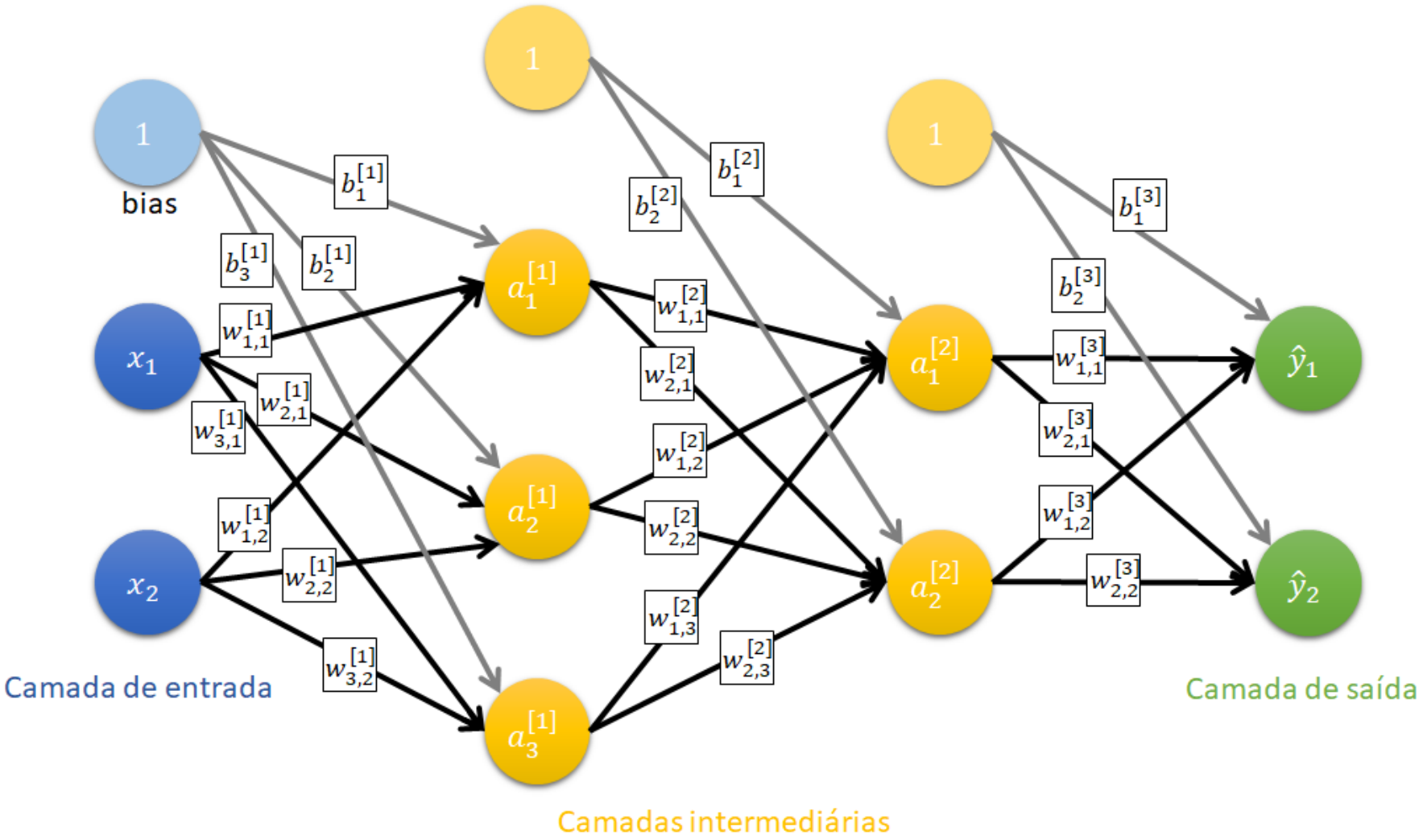
- Uma RNA consiste de uma grande quantidade de unidades de processamento simples (neurônios) interconectados;
- Cada neurônio artificial recebe muitos sinais;
- Os sinais recebidos pelos neurônios são modificados por um peso nas sinapses receptoras;
- Os neurônios artificiais somam de forma ponderada as entradas;
- Os neurônios definem a importância da informação e transmitem uma única saída;
- A saída de um neurônio é transmitida para muitos outros neurônios;
- Uma RNA pode possuir várias camadas de neurônios.

TIPOS DE CAMADAS DE UMA RNA

- Entrada;
- Intermediária ou escondida;
- Saída.

Exemplo de RNA com $L = 3$ camadas intermediárias (veja que a camada de saída entra na conta)



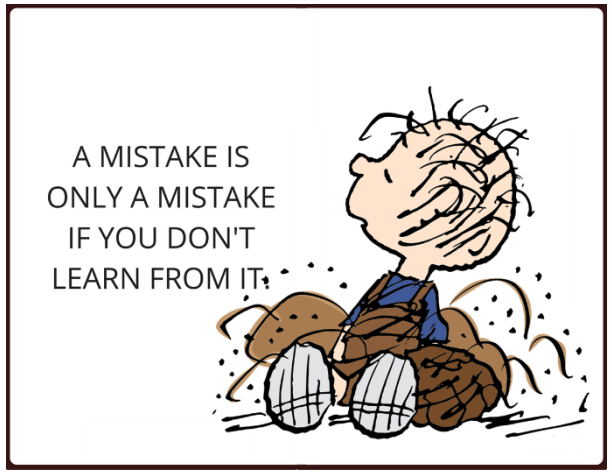

1
bias
x_1
x_2
$b_1^{[1]}$
$b_2^{[1]}$
$b_3^{[1]}$
$w_{1,1}^{[1]}$
$w_{2,1}^{[1]}$
$w_{3,1}^{[1]}$
$w_{1,2}^{[1]}$
$w_{2,2}^{[1]}$
$w_{3,2}^{[1]}$
$a_1^{[1]}$
$a_2^{[1]}$
$a_3^{[1]}$
1
$b_1^{[2]}$
$b_2^{[2]}$
$w_{1,1}^{[2]}$
$w_{2,1}^{[2]}$
$w_{1,2}^{[2]}$
$w_{2,2}^{[2]}$
$w_{1,3}^{[2]}$
$w_{2,3}^{[2]}$
$a_1^{[2]}$
$a_2^{[2]}$
1
$b_1^{[3]}$
$b_2^{[3]}$
$w_{1,1}^{[3]}$
$w_{2,1}^{[3]}$
$w_{1,2}^{[3]}$
$w_{2,2}^{[3]}$
$\hat{y}_1$
$\hat{y}_2$
Camada de entrada
Camadas intermediárias
Camada de saída

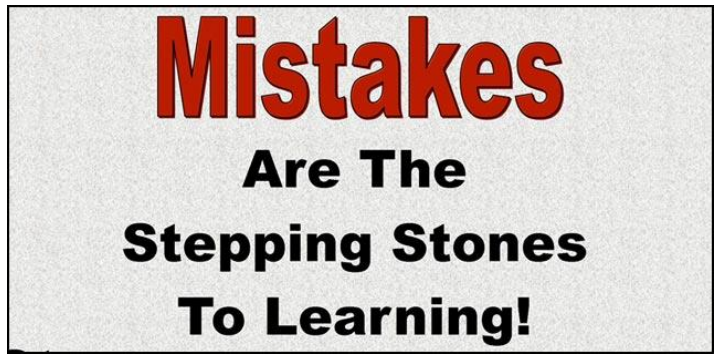
O QUE É UMA RNA?

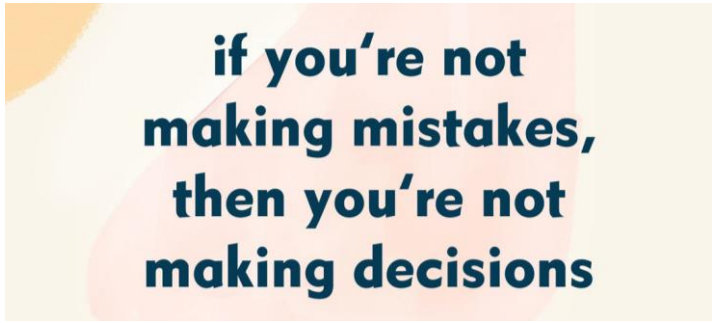
Em poucas palavras, a rede neural é uma grande função que mapeia um conjunto de dados de entrada $\boldsymbol{x}_i \in \boldsymbol{X}$ para o valor alvo desejado $\boldsymbol{y}_i \in \boldsymbol{Y}$. Os dados de entrada são definidos pela matriz $\boldsymbol{X} \in \mathbb{R}^{n_x,m}$, onde n_x é o número de *features* do problema (número de entradas de cada exemplo) e m é o número de dados disponíveis (número total de exemplos de treinamento). Os dados de saída são definidos por $\boldsymbol{Y} \in \mathbb{R}^{n_y,m}$, onde n_y é o número de saídas da RNA.

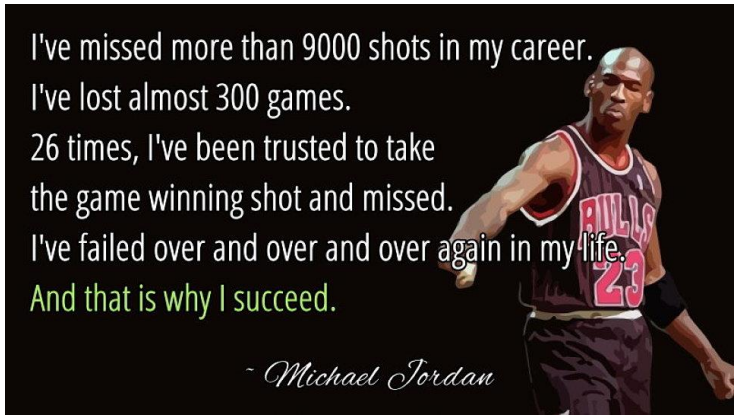
Para uma RNA ser capaz de mapear o problema, seja classificação ou regressão, ela precisa ser **treinada**. O *aprendizado supervisionado* de uma RNA exige, inicialmente, um conjunto de dados rotulados e classificados. Durante o aprendizado as saídas geradas pela RNA são comparadas com as saídas desejadas e as diferenças entre elas são usadas para o treinamento.

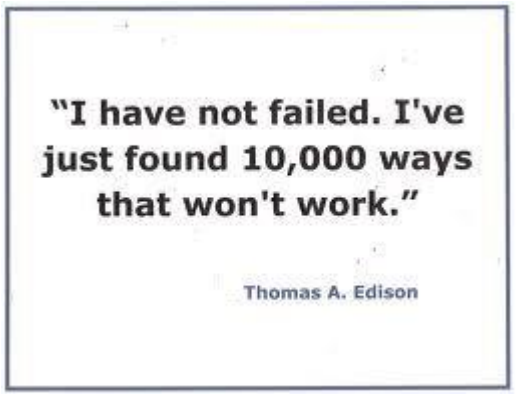
RNAs SABEM DAS COISAS...

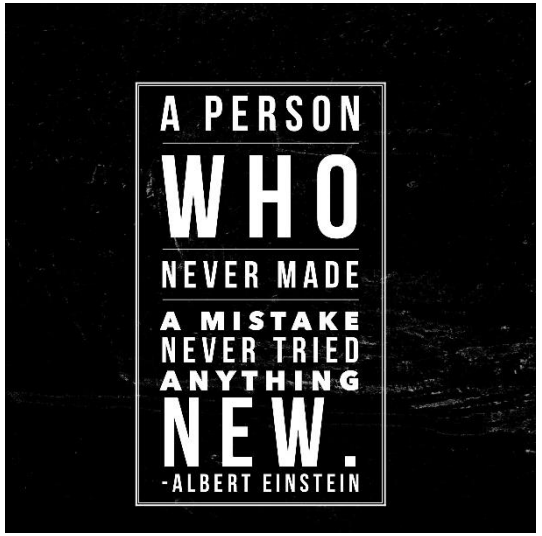






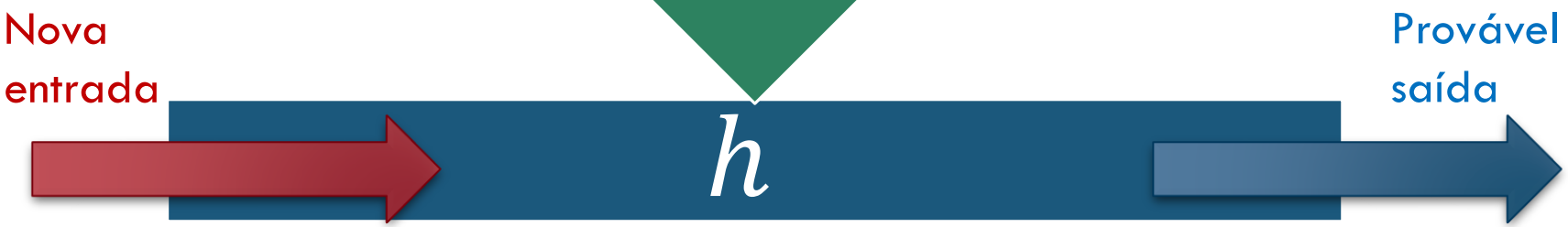


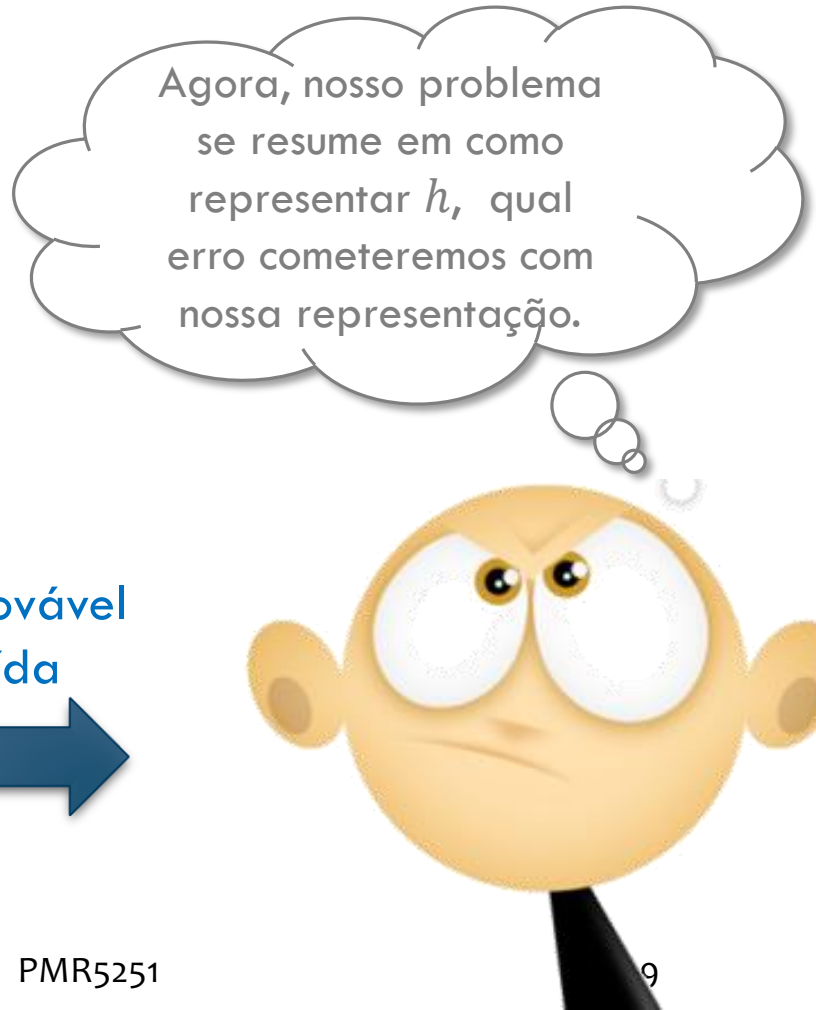




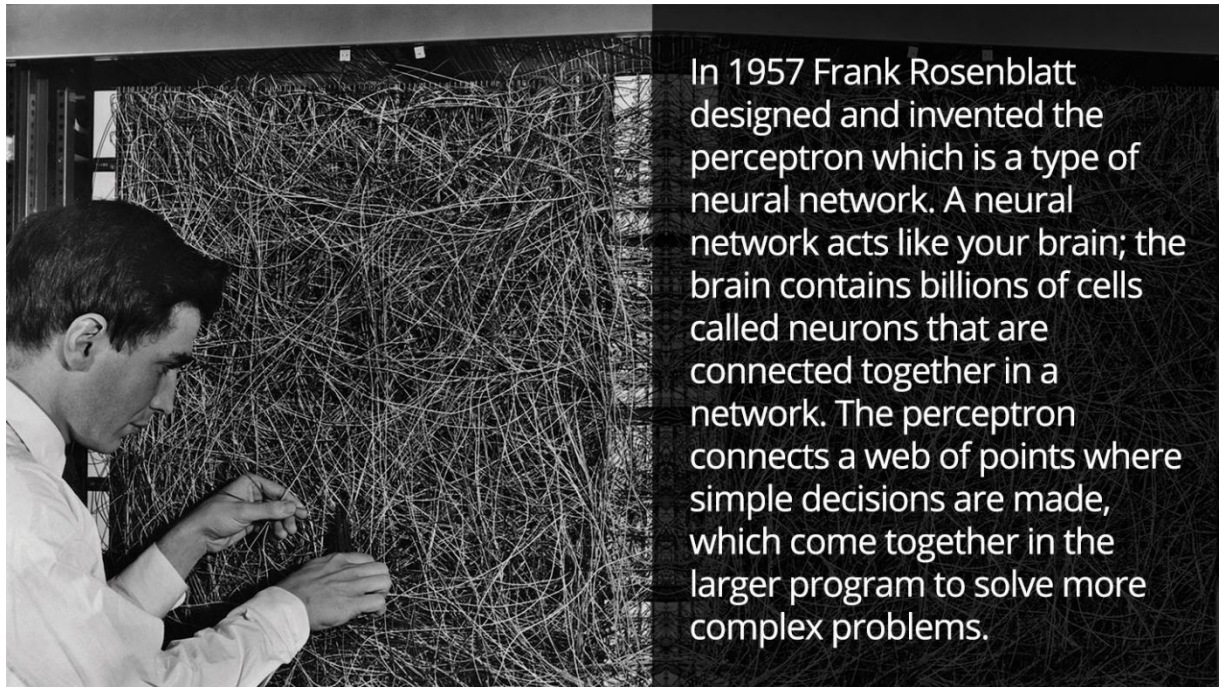

Dados de treinamento
Dados de entrada x
Dados de saída y


Algoritmo de treinamento
Encontrar a hipótese h


Nova entrada
h
Provável saída

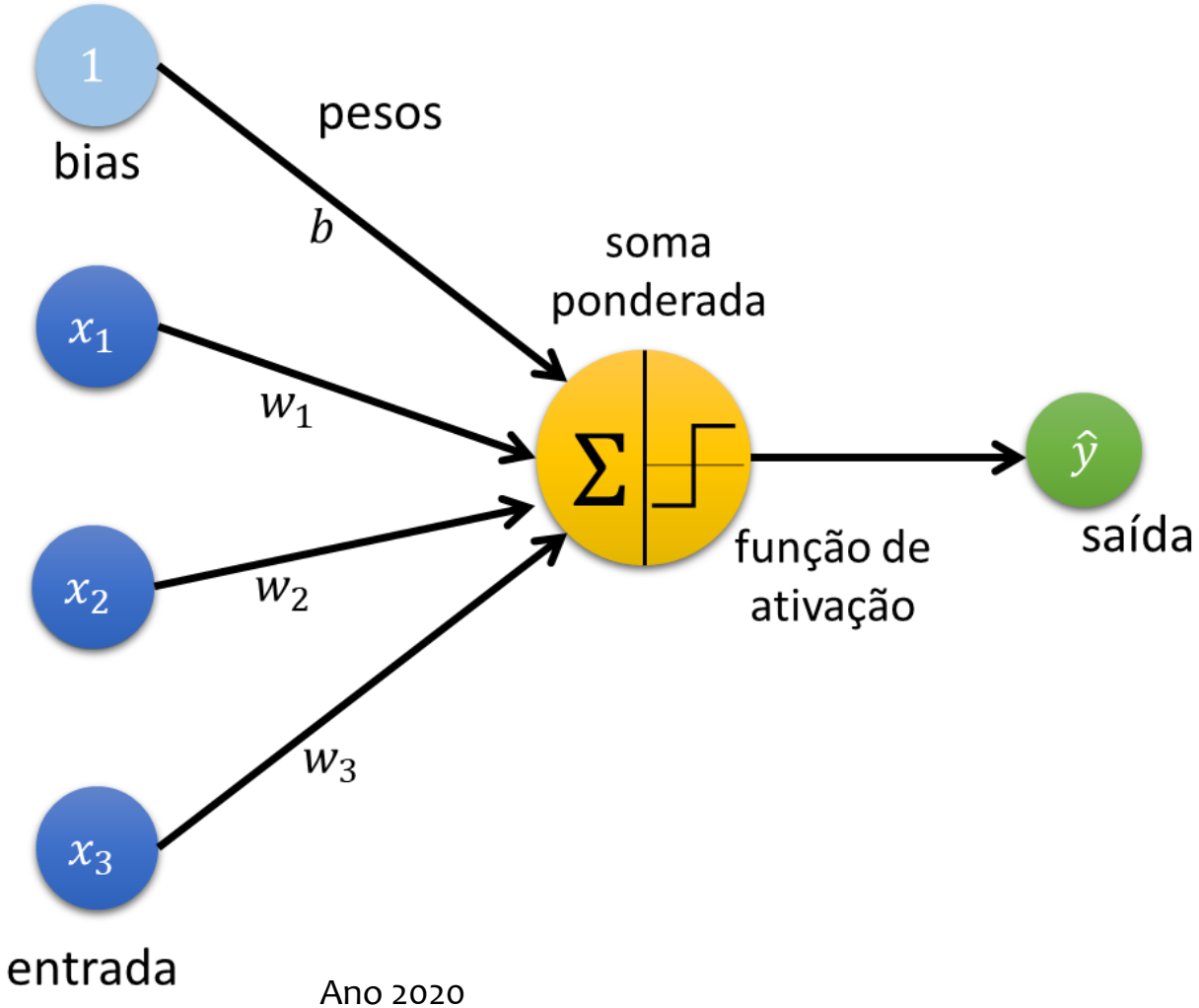

Agora, nosso problema se resume em como representar h, qual erro cometeremos com nossa representação.

ESTRUTURA DE UMA RNA: PERCEPTRON



O modelo mais simples, e bastante antigo, que serviu de base e inspiração para teoria de RNAs é o *Perceptron*, um classificador linear usado para previsões binárias. Foi inventado em 1957 por Frank Rosenblatt no Cornell Aeronautical Laboratory. **Isso significa que, para que funcione, os dados devem ser linearmente separáveis.**

O QUE É O PERCEPTRON?

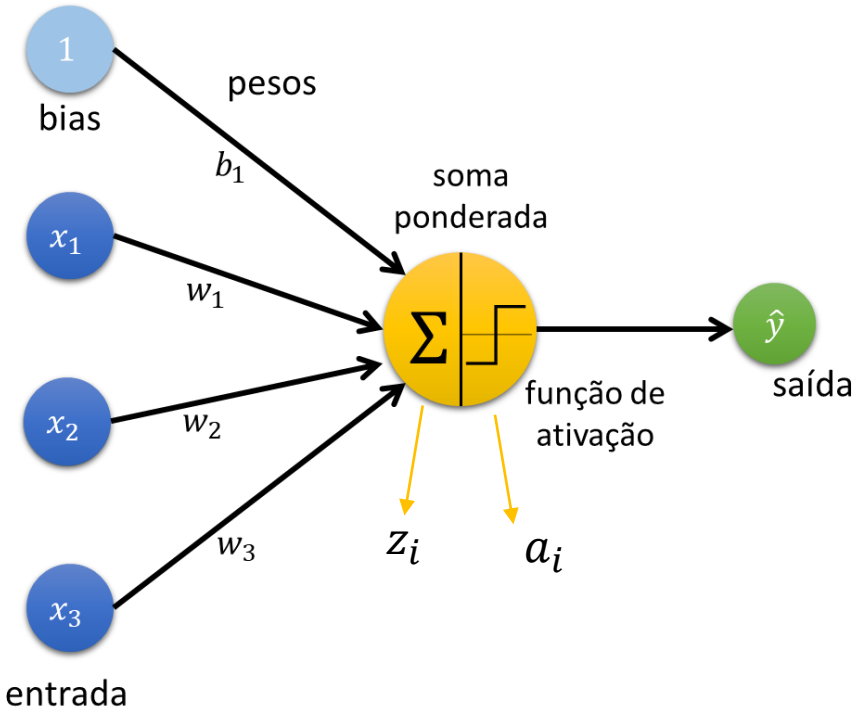


Neurônio de entrada

Neurônio intermediário

Neurônio de saída

MODELO DO PERCEPTRON



Vetor de entrada, de dimensão $(1, n_x)$

$$\mathbf{x}^{(i)} = [x_1, x_2, \cdots, x_{n_x}]$$

Vetor de pesos, de dimensão $(1, n_x)$

$$\mathbf{w} = [w_1, w_2, \cdots, w_{n_x}]$$

Estado do neurônio, escalar

$$z^{(i)} = w_1 x_1 + \cdots + w_{n_x} x_{n_x} + b = \sum_{j=0}^{n_x} x_j^{(i)} w_j + b = \mathbf{w}^T \mathbf{x}^{(i)} + b$$

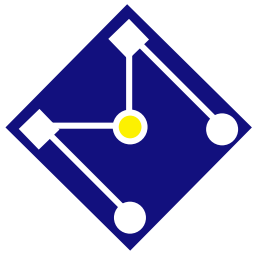
Ativação, escalar

$$a_i = g(z_i)$$

Previsão, escalar

$$\hat{y}_i = a_i$$

FUNÇÃO DE ATIVAÇÃO



O objetivo da função de ativação é introduzir não linearidade na saída de um neurônio.

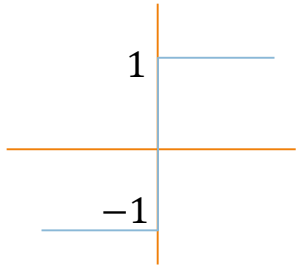
Essa não linearidade é um dos fatores que afetam nossos resultados e a precisão do nosso modelo.

No caso de usarmos Redes Neurais com várias camadas ocultas, uma função de ativação linear simplesmente irá gerar uma série de transformações afins portanto, podemos muito bem não ter nenhuma camada oculta.

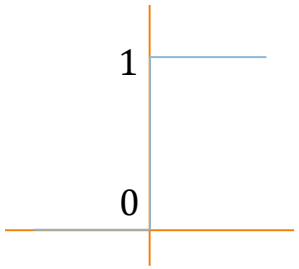
A menos que transmitamos não linearidade, não estamos computando modelos interessantes, mesmo se nos aprofundarmos nas redes neurais.

Degrau Unitário (Unit step):

$$g(z) = \begin{cases} 1 & z \geq 0 \\ -1 & cc \end{cases}$$

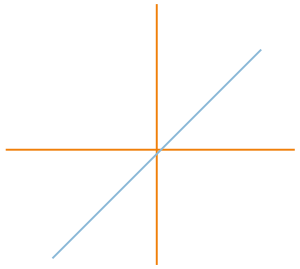


$$g(z) = \begin{cases} 1 & z \geq 0 \\ 0 & cc \end{cases}$$



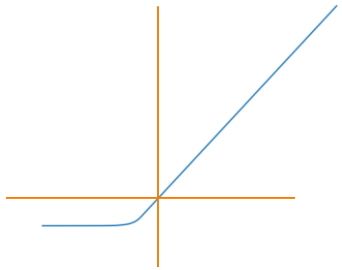
Linear:

$$g(z) = z$$



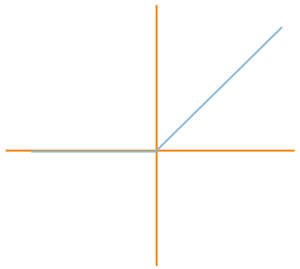
ELU (Exponential Linear Unit):

$$g(z) = \max(\alpha(e^z - 1), z)$$



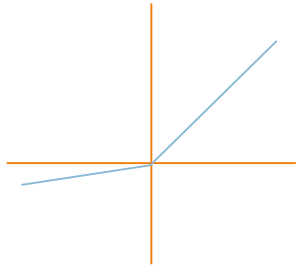
ReLU (Rectified Linear Unit):

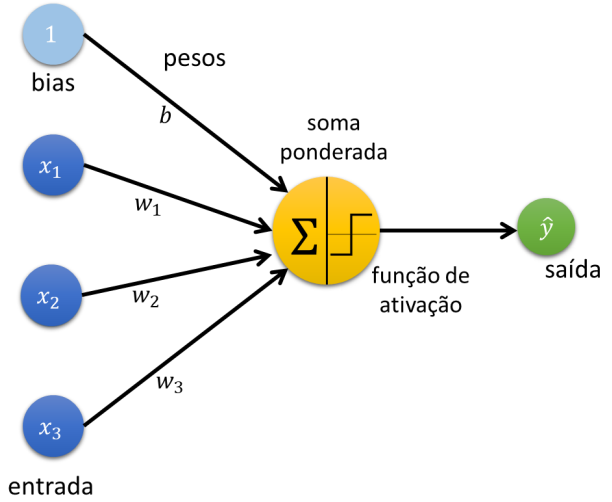
$$g(z) = \max(0, z)$$



Leaky ReLU:

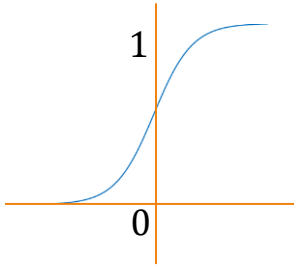
$$g(z) = \max(0.1z, z)$$





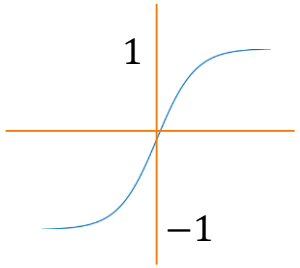
Sigmoide (logistic, sigmoid):

$$g(z) = \frac{1}{1 + e^{-z}}$$



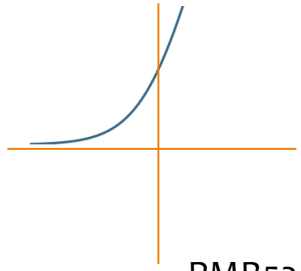
tanh (Hyperbolic tangent):

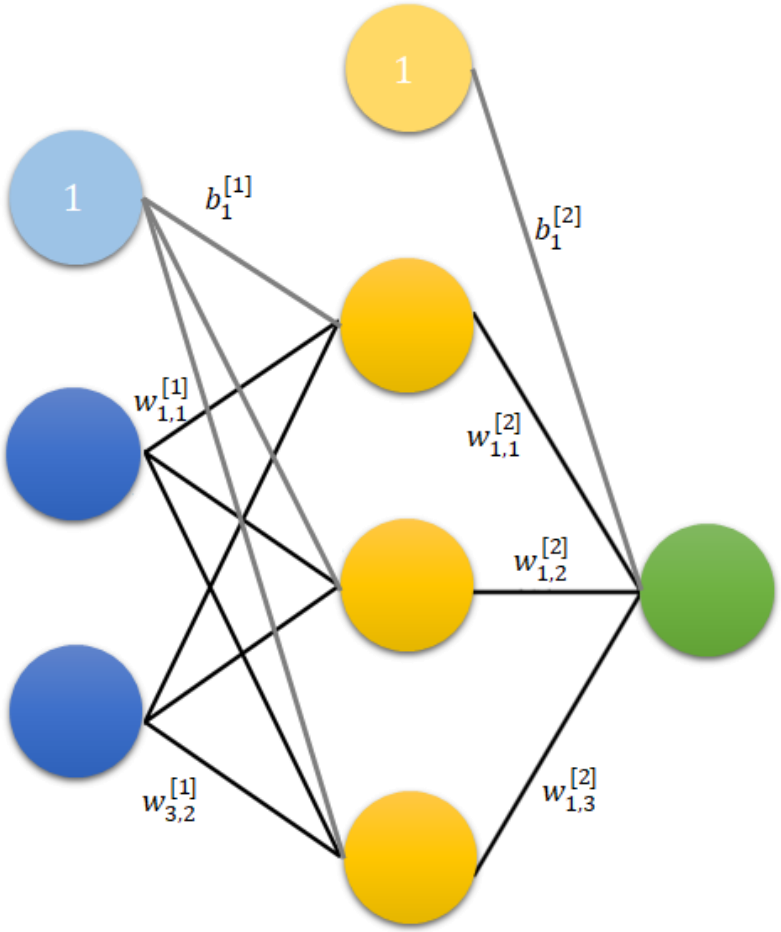
$$g(z) = \frac{e^{2z} - 1}{e^{2z} + 1}$$

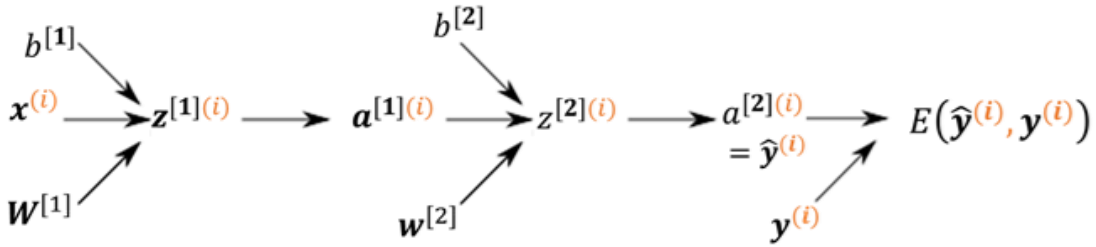


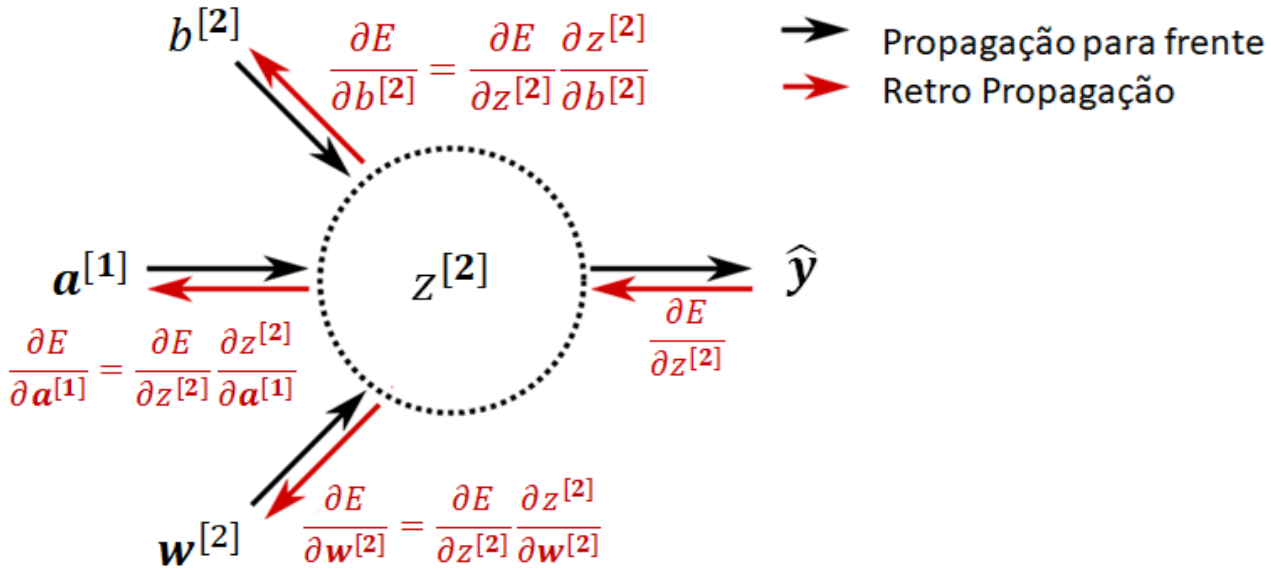
Softplus

$$g(z) = \ln 1 + e^z$$




1
1
$b_1^{[1]}$
$b_1^{[2]}$
$w_{1,1}^{[1]}$
$w_{1,1}^{[2]}$
$w_{1,2}^{[2]}$
$w_{3,2}^{[1]}$
$w_{1,3}^{[2]}$


$b^{[1]}$
$x^{(i)}$
$W^{[1]}$
$z^{[1](i)}$
$a^{[1](i)}$
$b^{[2]}$
$w^{[2]}$
$z^{[2](i)}$
$a^{[2](i)} = \hat{y}^{(i)}$
$y^{(i)}$
$E(\hat{y}^{(i)}, y^{(i)})$


$b^{[2]}$
$\frac{\partial E}{\partial b^{[2]}} = \frac{\partial E}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial b^{[2]}}$
Propagação para frente
Retro Propagação
$a^{[1]}$
$z^{[2]}$
$\hat{y}$
$\frac{\partial E}{\partial a^{[1]}} = \frac{\partial E}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial a^{[1]}}$
$\frac{\partial E}{\partial z^{[2]}}$
$w^{[2]}$
$\frac{\partial E}{\partial w^{[2]}} = \frac{\partial E}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial w^{[2]}}$

COMO NOSSA REDE APRENDE???

O objetivo de treinar uma RNA é calcular os seus parâmetros de forma a minimizar uma **função de custo**.

A **função de custo** é baseada em uma **função do erro** entre a saída desejada e a saída calculada pela RNA para cada exemplo utilizado no treinamento.

Função *Erro Quadrático*

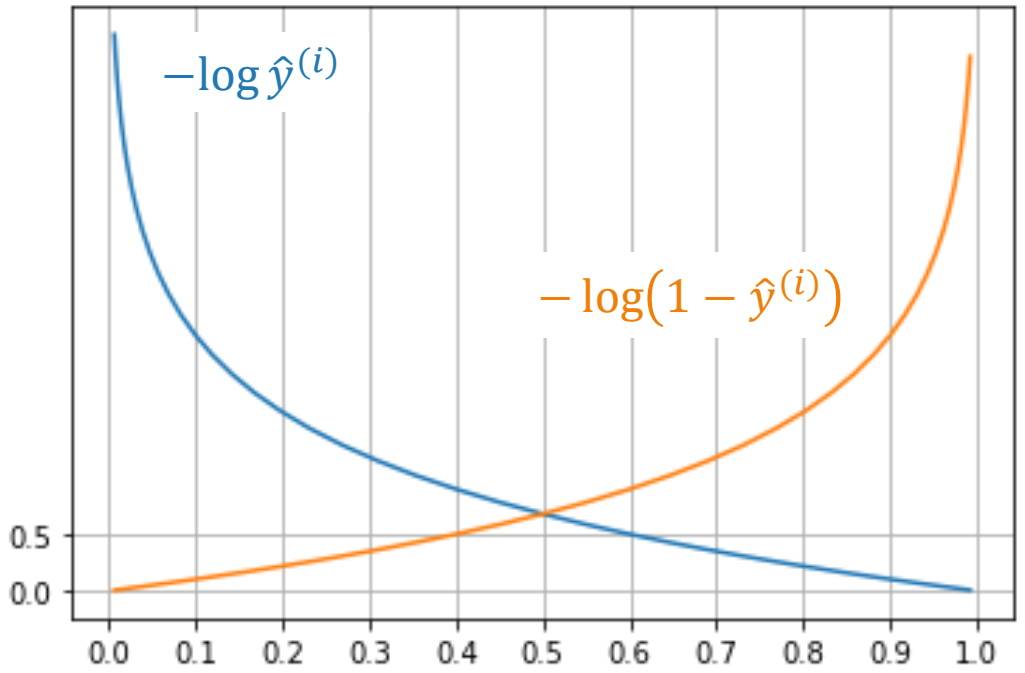
$$E(\hat{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)}) = \sum_{j=1}^{n_y} \left(\hat{y}_j^{(i)} - y_j^{(i)} \right)^2 = \left\| \hat{\mathbf{y}}^{(i)} - \hat{\mathbf{y}}^{(i)} \right\|_2^2$$

Função de Custo

$$J\left(\mathbf{W}, \mathbf{B}\right) = \frac{1}{m} \sum_{i=1}^{m} E(\hat{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)}) = \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n_y} \left(\hat{y}_j^{(i)} - y_j^{(i)} \right)^2 = \frac{1}{m} \sum_{i=1}^{m} \left\| \hat{\mathbf{y}}^{(i)} - \hat{\mathbf{y}}^{(i)} \right\|_2^2$$

Função Logística de *Erro*

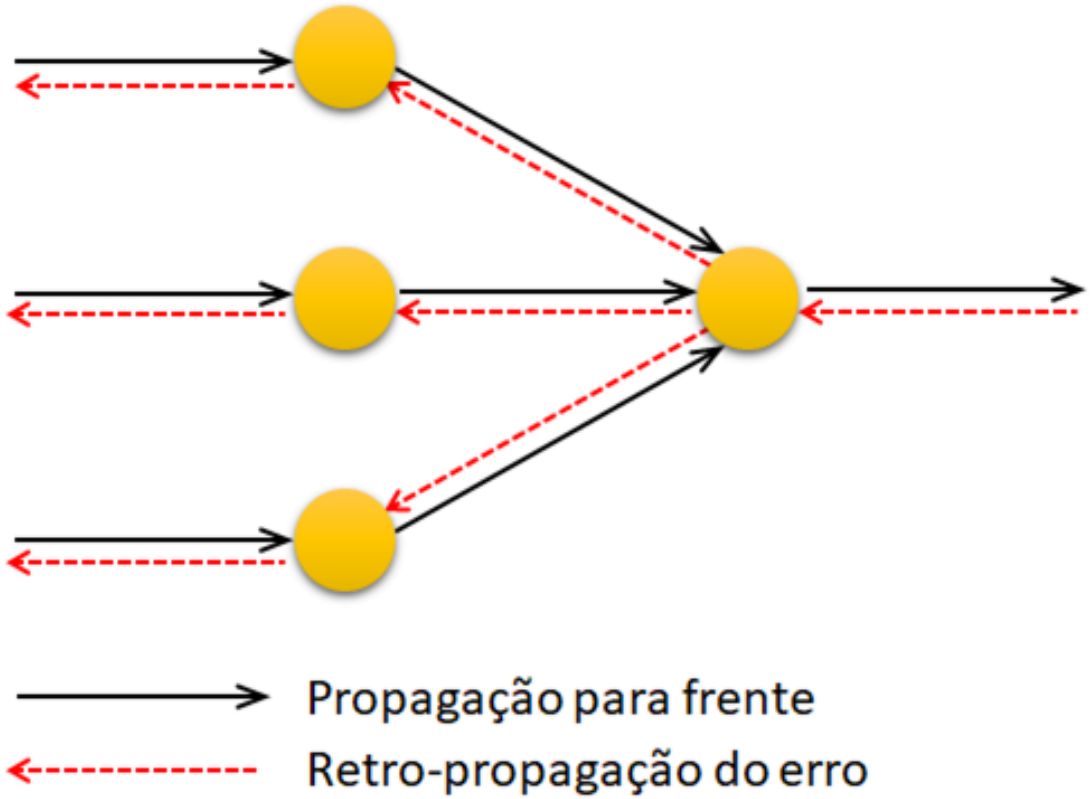
$$L(\hat{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)}) = -y^{(i)} \log \hat{y}^{(i)} - \left(1 - y^{(i)}\right) \log \left(1 - \hat{y}^{(i)}\right)$$



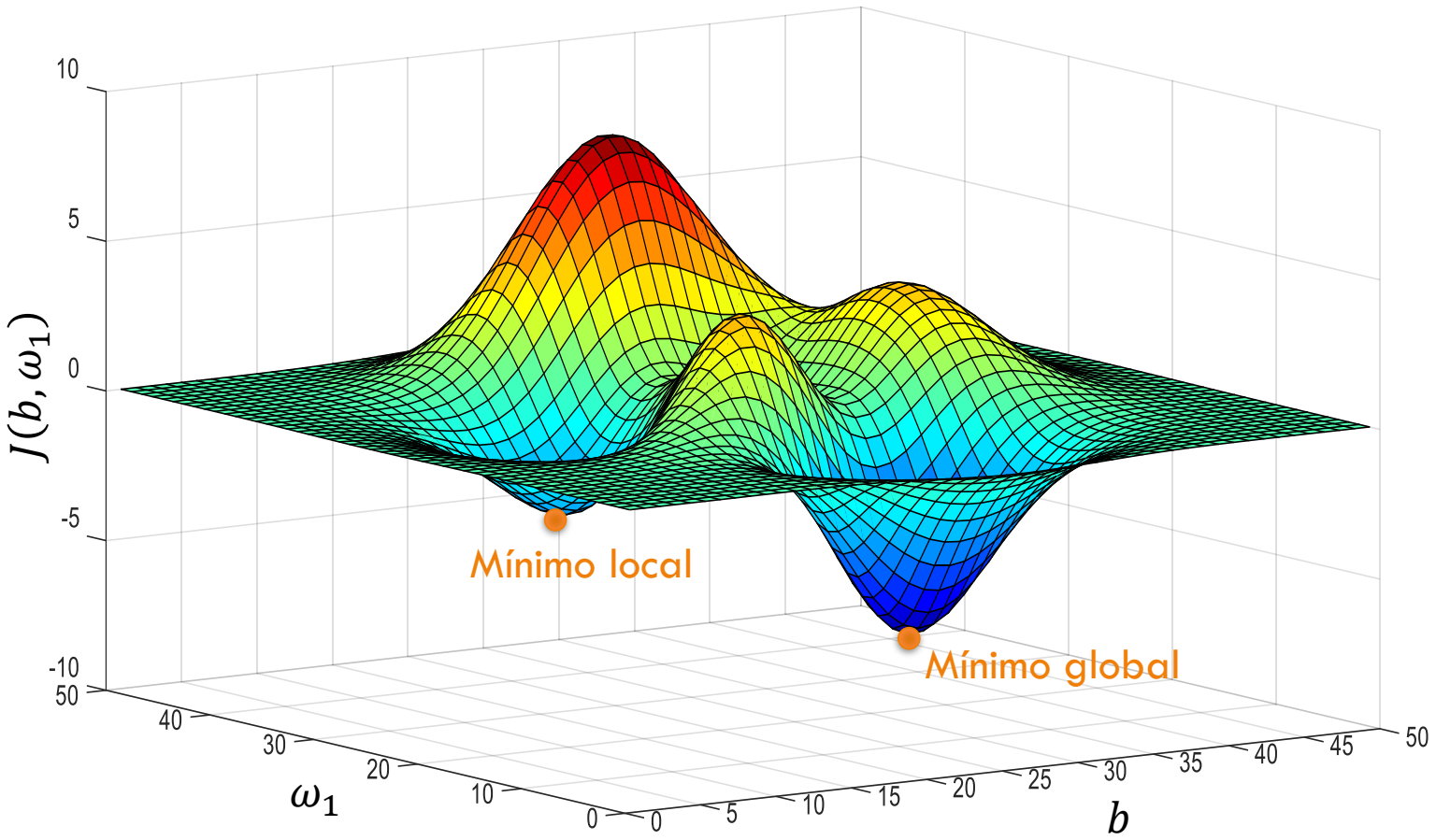
Função de Custo

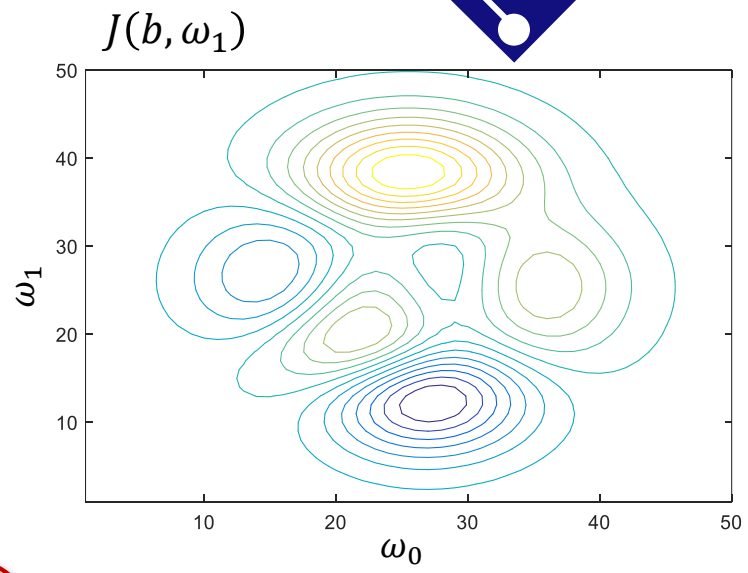
$$J(\mathbf{W}, \mathbf{B}) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)}) = \frac{1}{m} \sum_{i=1}^{m} -y^{(i)} \log \hat{y}^{(i)} - \left(1 - y^{(i)}\right) \log \left(1 - \hat{y}^{(i)}\right)$$

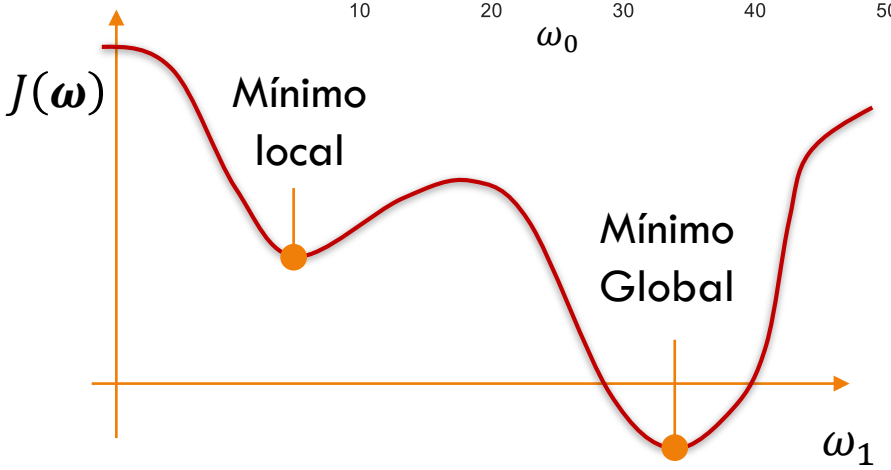
RETROPROPAGAÇÃO (BACKPROPAGATION)



COMO RESOLVER O PROBLEMA DE OTIMIZAÇÃO???







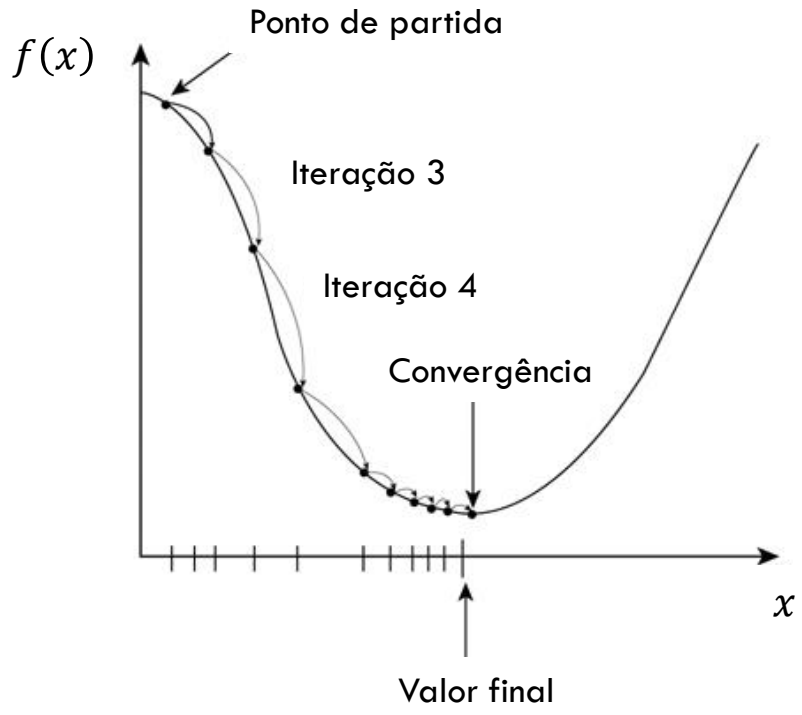
GRADIENTE DESCENDENTE

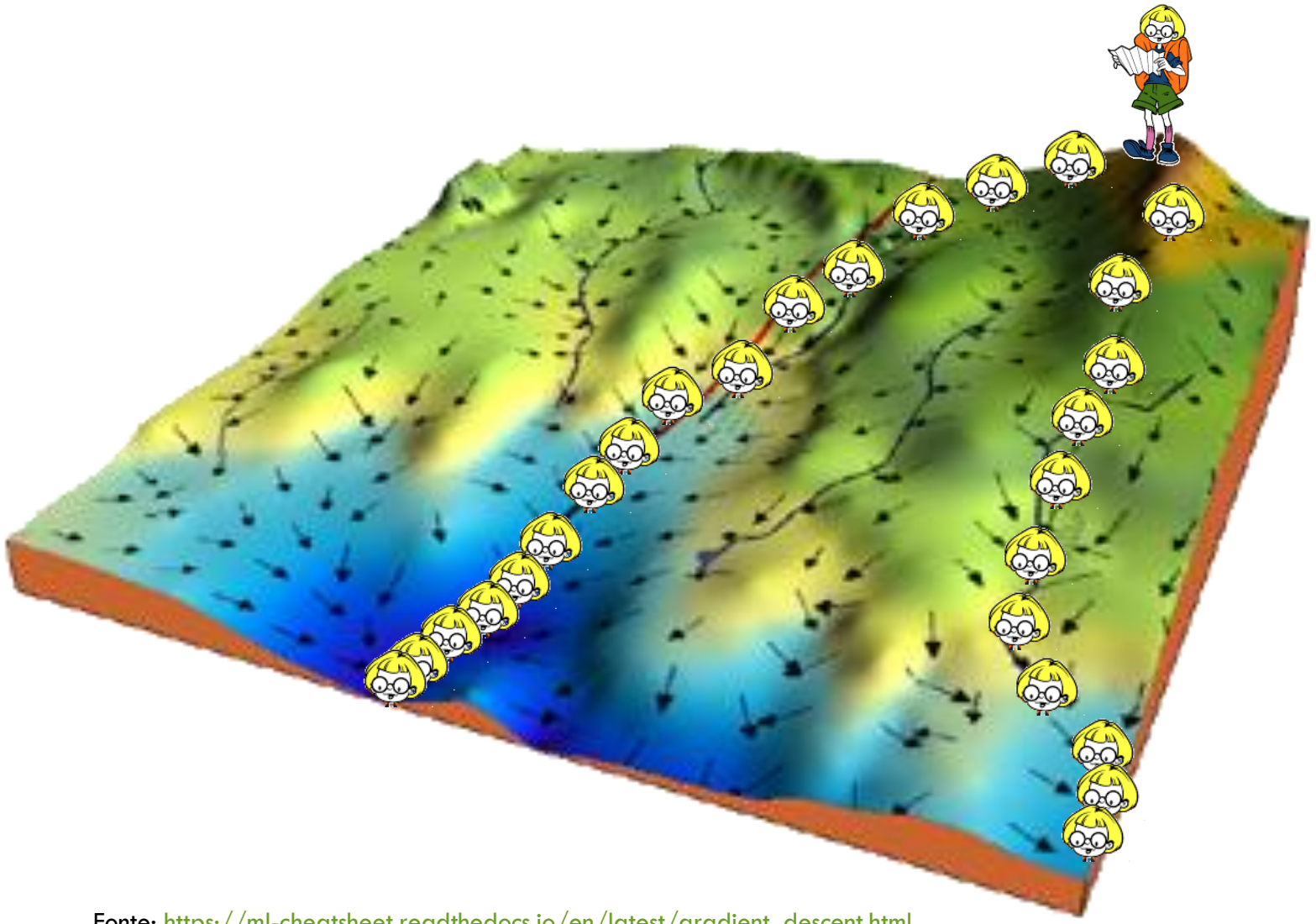
Esse método consiste no *motor* básico das RNAs.

Na prática, é difícil encontrar um problema sem restrições, com um único mínimo...
Entretanto vamos começar *do princípio*....

$$\min_{(x)} f(x)$$

O Gradiente Descendente (GD) é um algoritmo utilizado para encontrar o mínimo de uma função de forma iterativa.



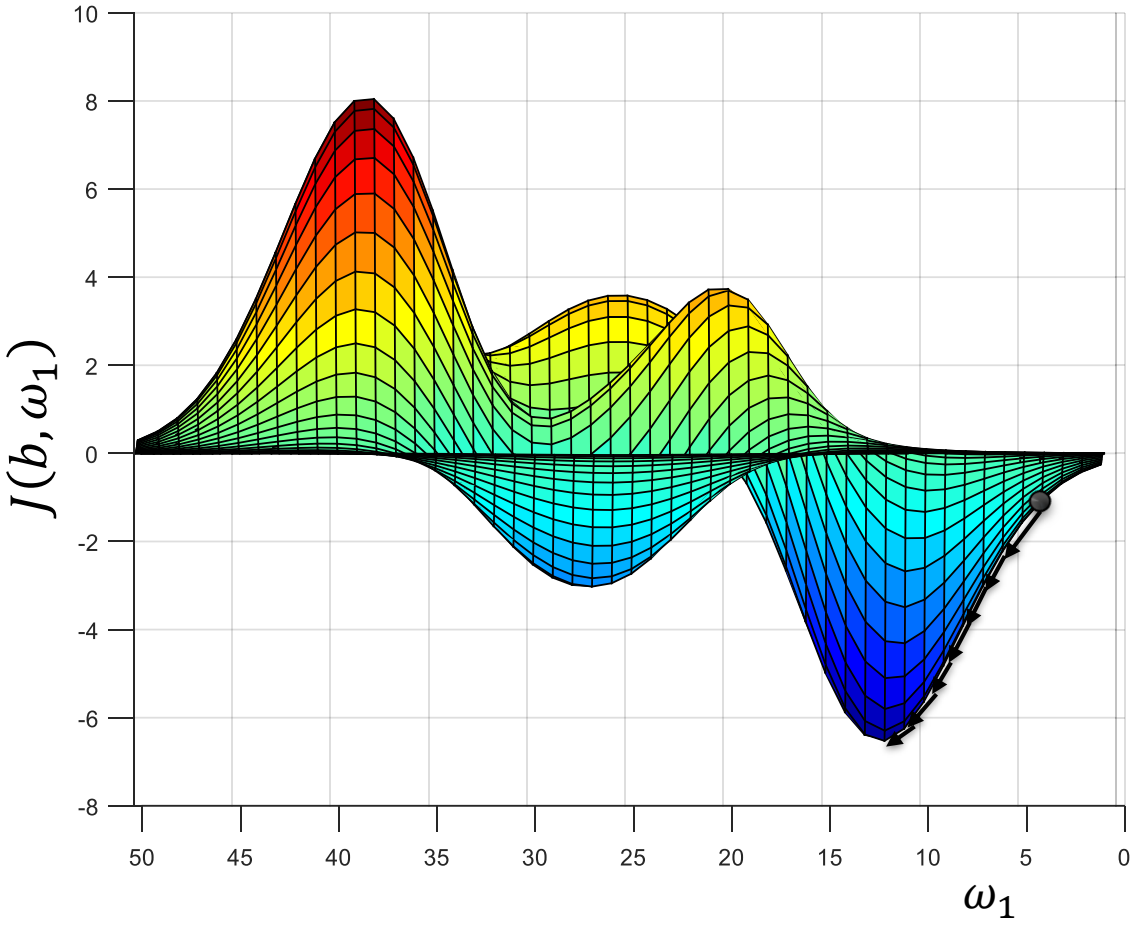


Fonte: https://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html

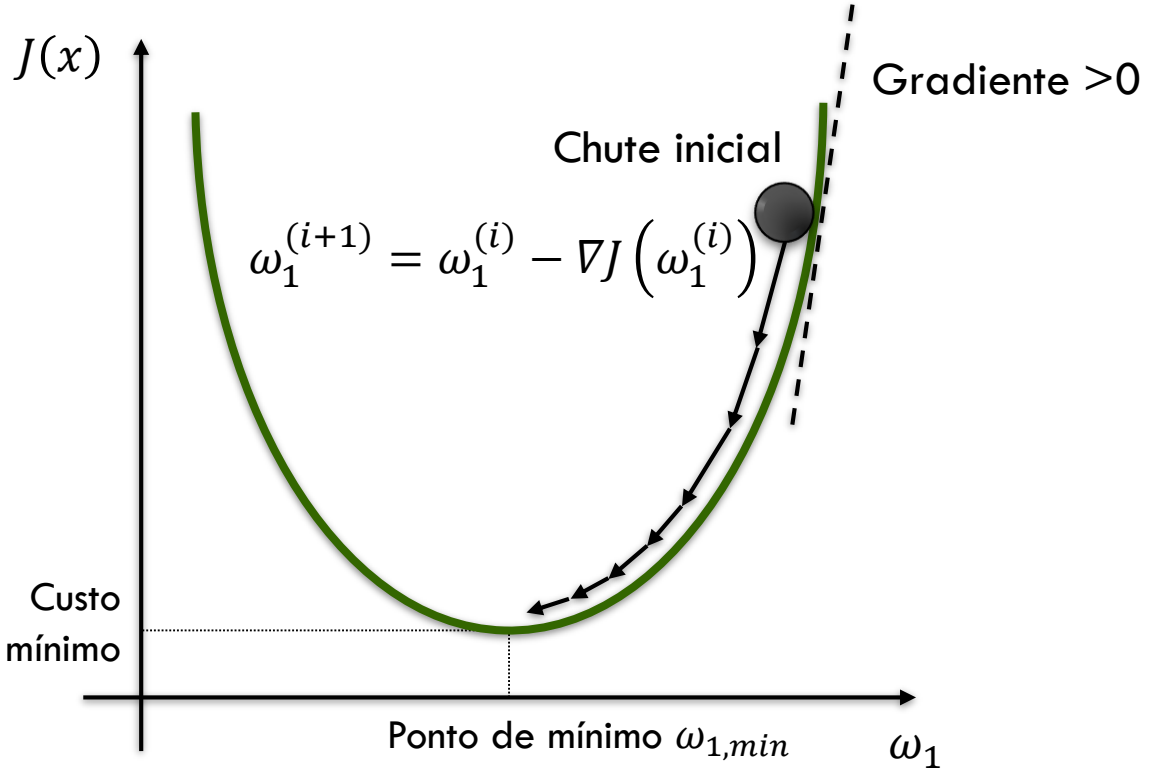
Nosso objetivo é: partindo do alto da montanha, no canto superior direito (alto custo), chegar no mar azul escuro, no canto inferior esquerdo (baixo custo).

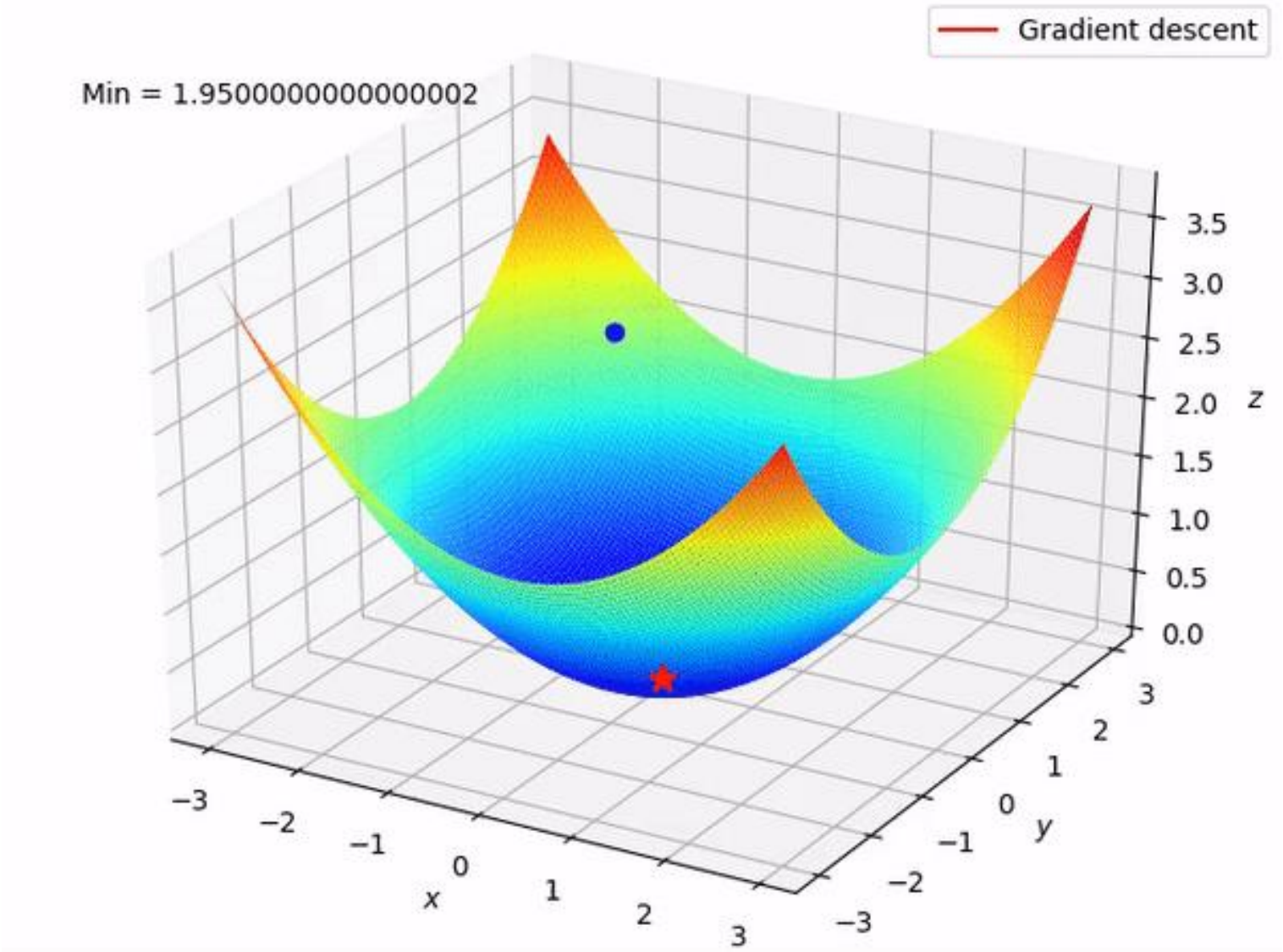
As setas representam a direção da descida mais íngreme (gradiente negativo) de qualquer ponto dado - a direção que diminui a função de custo o mais rápido possível.

Damos nosso primeiro passo em declive na direção especificada pelo gradiente negativo. Em seguida recalculamos o gradiente negativo (através das coordenadas do nosso novo ponto) e damos outro passo na direção que ele especifica. Continuamos este processo de forma iterativa até chegarmos ao fundo do nosso gráfico, ou a um ponto em que não podemos mais nos mover para baixo - um mínimo local.



Supondo $b = 0$ para ilustrar de forma clara o algoritmo





Fonte: https://medium.com/@lucasoliveiras/regress%C3%A3o-linear-do-zero-com-python-ef74a81c4b84

ALGORITMO DE GRADIENTE DESCENDENTE

1. Inicialização dos parâmetros da RNA: Atribua um valor inicial, W, b para os parâmetros;
2. Execução da RNA **para todos os exemplos do conjunto de dados de treinamento**, de forma que dadas as entradas calculam-se as saídas previstas pela RNA;
3. Cálculo da **função de custo** para todos os exemplos de treinamento, através do somatório da função erro;
4. Cálculo do gradiente da função de custo em relação a todos os parâmetros da RNA;
5. Atualização dos parâmetros da RNA na direção oposta ao gradiente de forma a reduzir o valor da função de custo

$$w_{k,j}^{[l]} = w_{k,j}^{[l]} - \alpha \frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial w_{k,j}^{[l]}}$$

$$b_{k}^{[l]} = b_{k}^{[l]} - \alpha \frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial b_{k}^{[l]}}$$

Repetição das etapas 2 a 5 até a obtenção de um valor desejado para a função de custo, ou até o número limite de *épocas*.

TAXA DE APRENDIZAGEM α

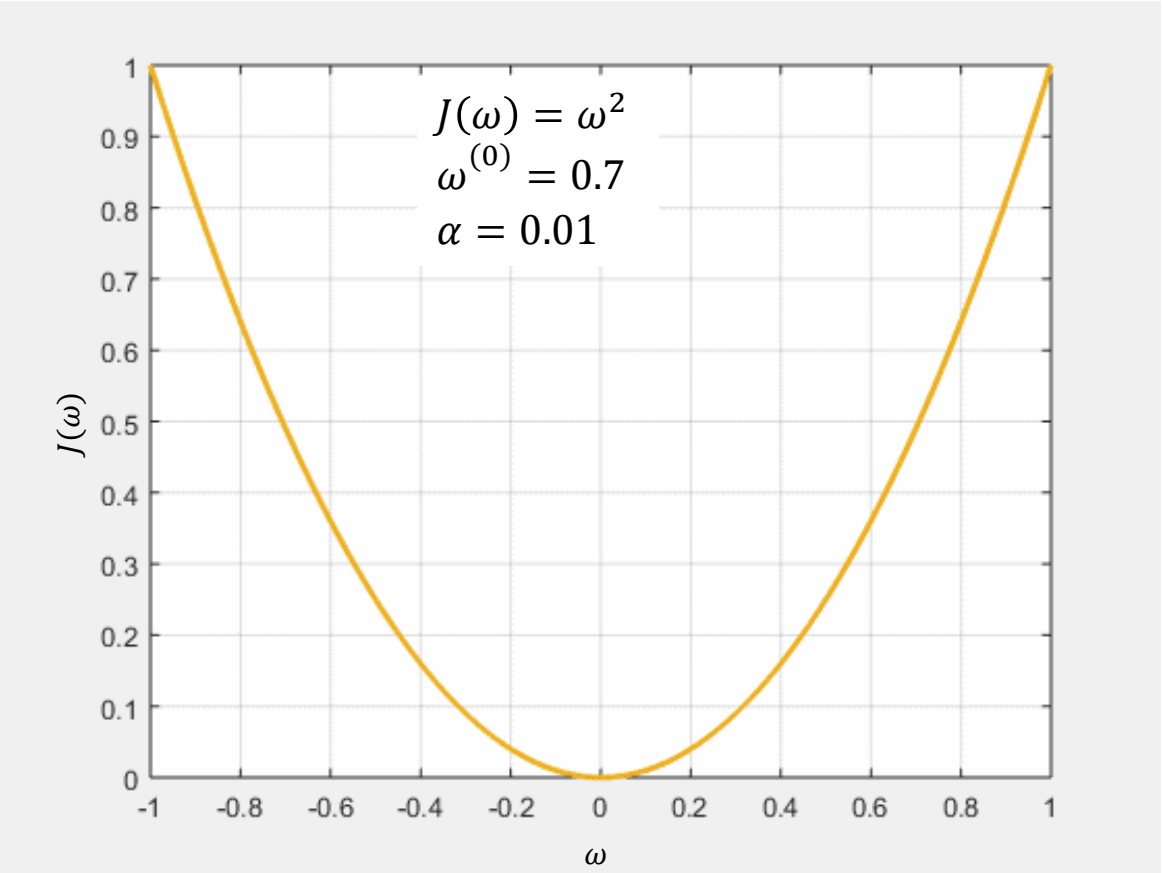
Taxa de aprendizagem α controla o tamanho do passo em cada iteração;

Selecionar o valor correto é uma decisão crítica do modelo:

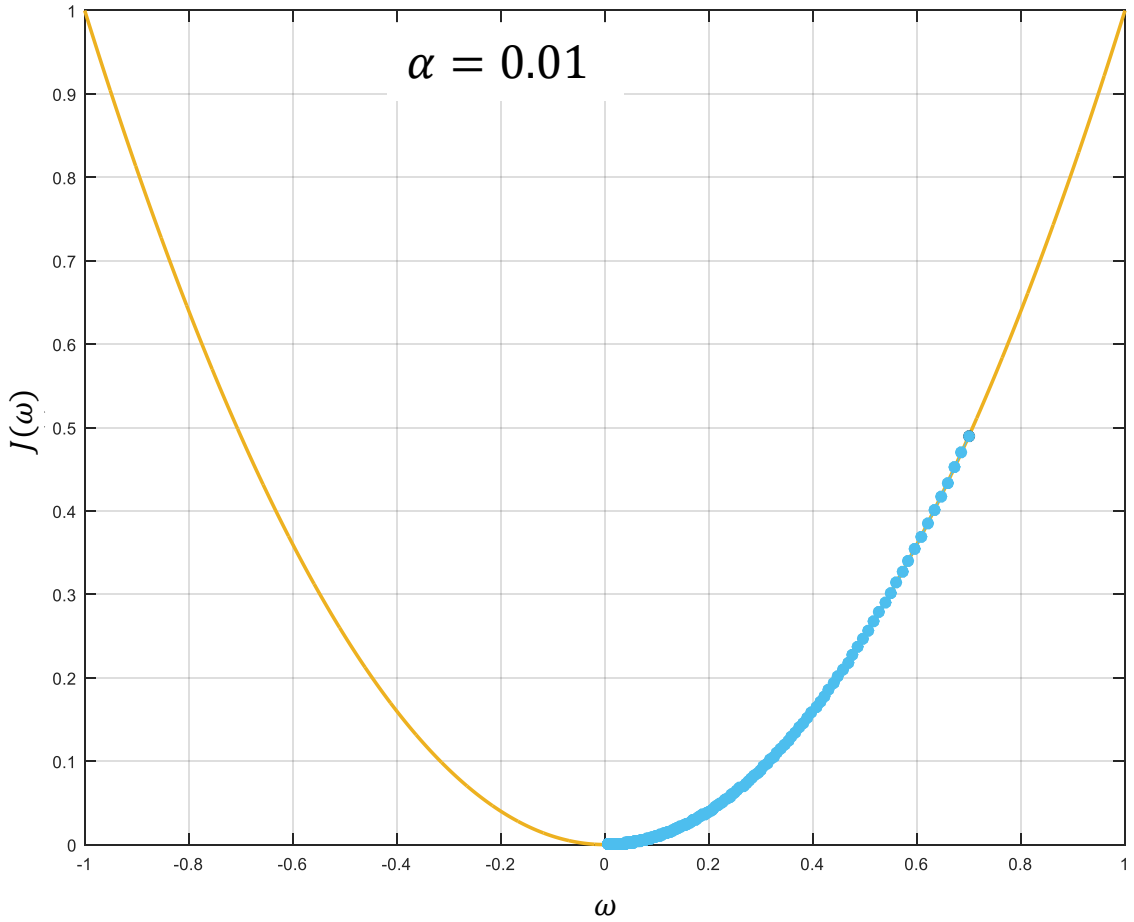
- Pode convergir para um mínimo local se a função de custo não for convexa;
- Quando nos aproximamos de um mínimo local, a descida de gradiente tomará automaticamente passos menores. Portanto, não há necessidade de diminuir α ao longo do tempo;
- Valores de α não podem ser muito grandes, nem muito pequenos...
- Tente … 0,001 – 0,003 – 0,01 – 0,03 – 0,1 – 0,3 – 1 … Suba/Desça ~ 3x em 3x

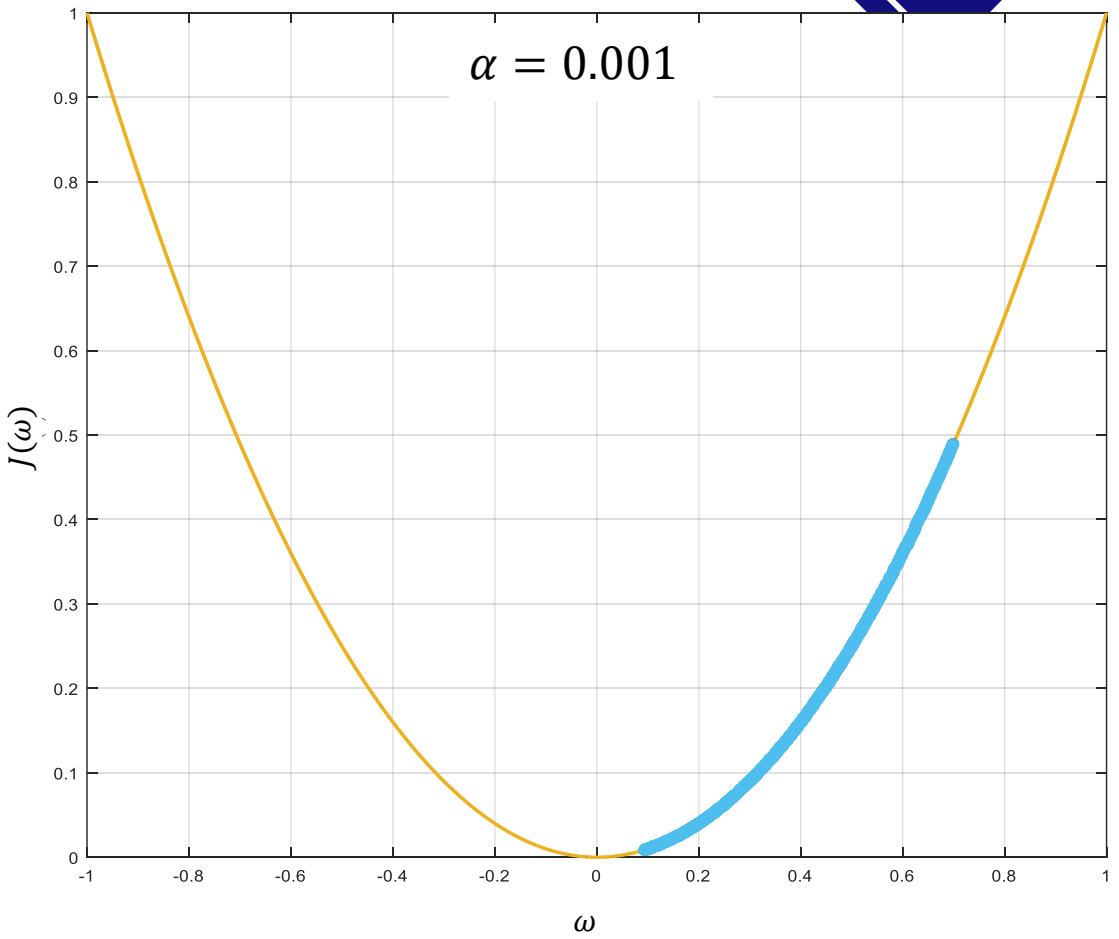
α MUITO PEQUENO

Vamos simplificar a função de custo como algo da forma $J(\omega) = \omega^2$

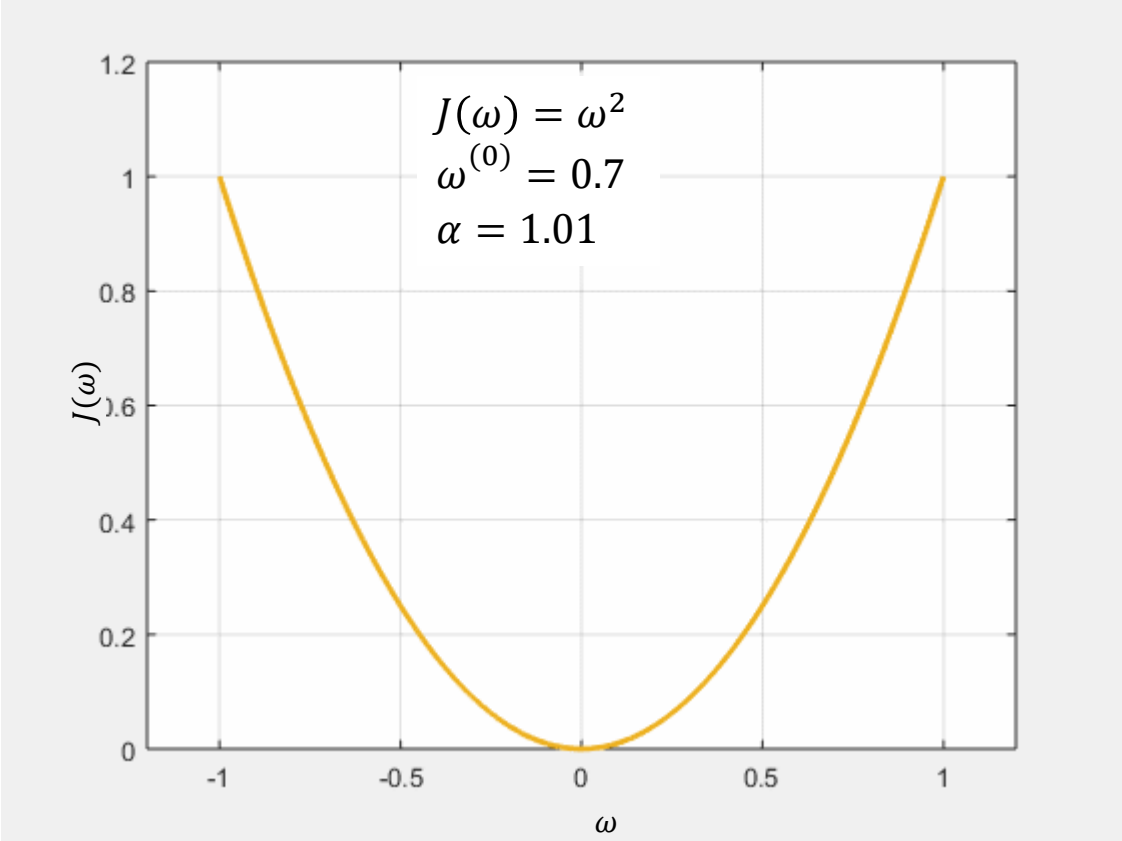


Uma taxa de aprendizado muito baixa é mais precisa, mas o cálculo do gradiente é demorado, então levará muito tempo para chegarmos ao fim. ... Podemos nos mover com confiança na direção do gradiente negativo, pois estamos recalculando-o com muita frequência, mas "desceremos muito devagar";


$\alpha = 0.01$
$J(\omega)$
ω

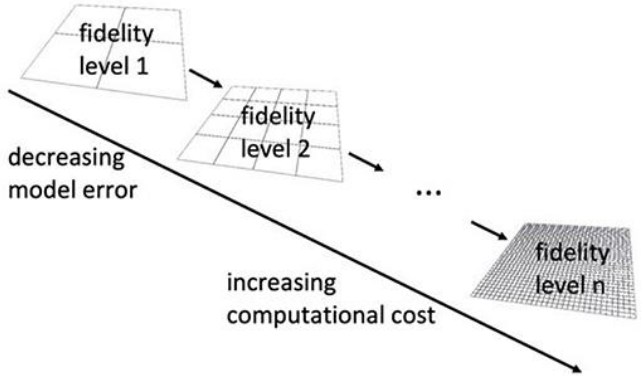

$\alpha = 0.001$
$J(\omega)$
ω

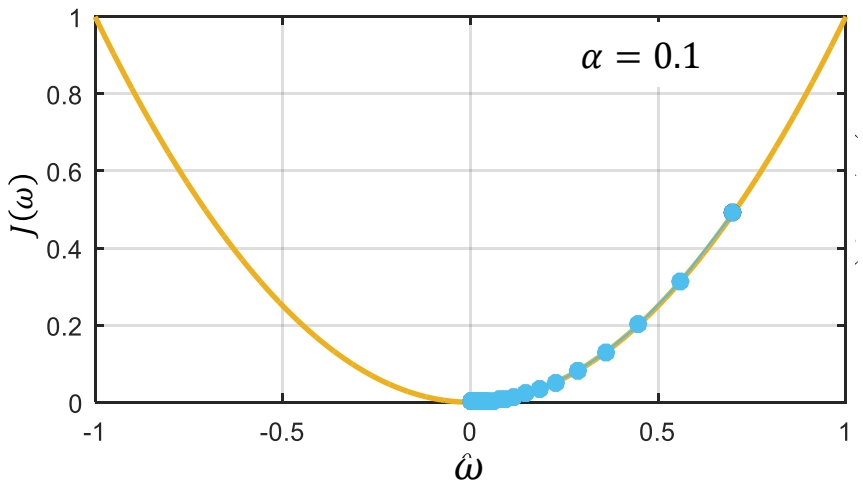
α MUITO GRANDE

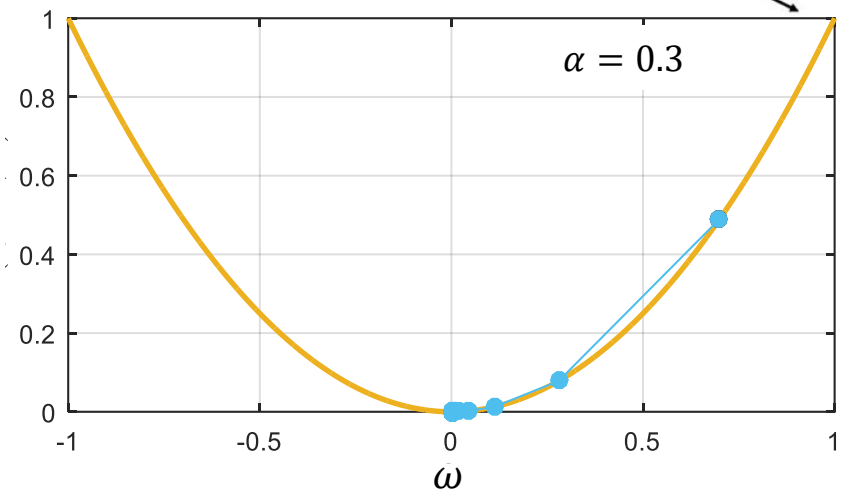


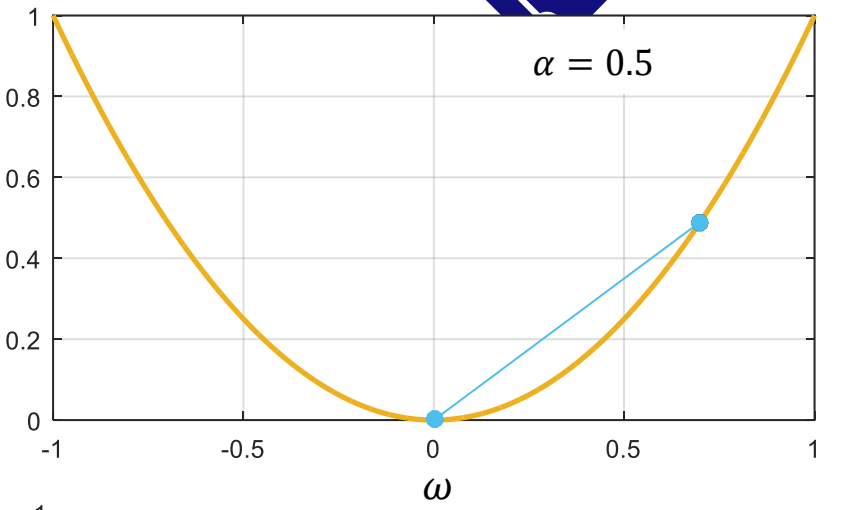
Se utilizarmos uma taxa de aprendizado muito alta, podemos cobrir maior distância a cada passo, mas corremos o risco de ultrapassar o ponto mais baixo, já que o gradiente está mudando constantemente. O método diverge porque estamos "descendo muito rápido".

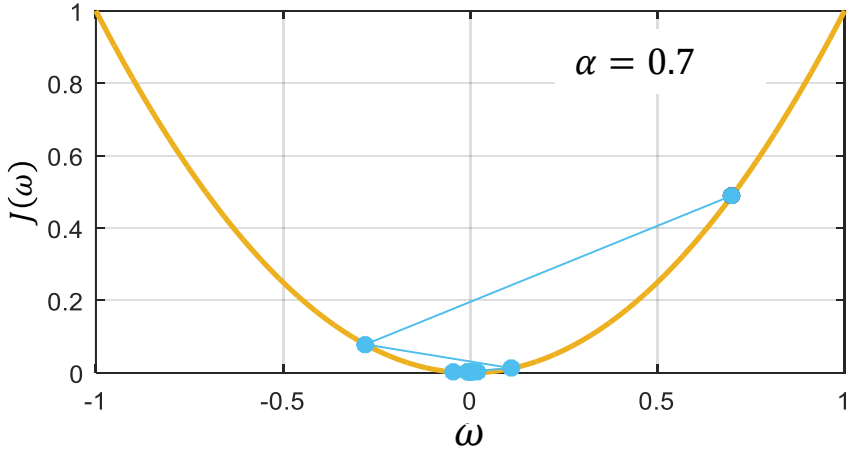
$$J(\omega) = \omega^2$$
$$\omega^{(0)} = 0.7$$

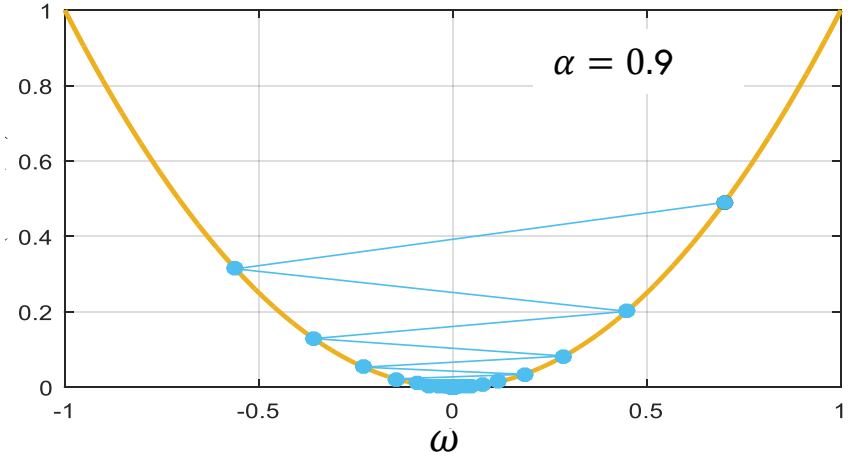


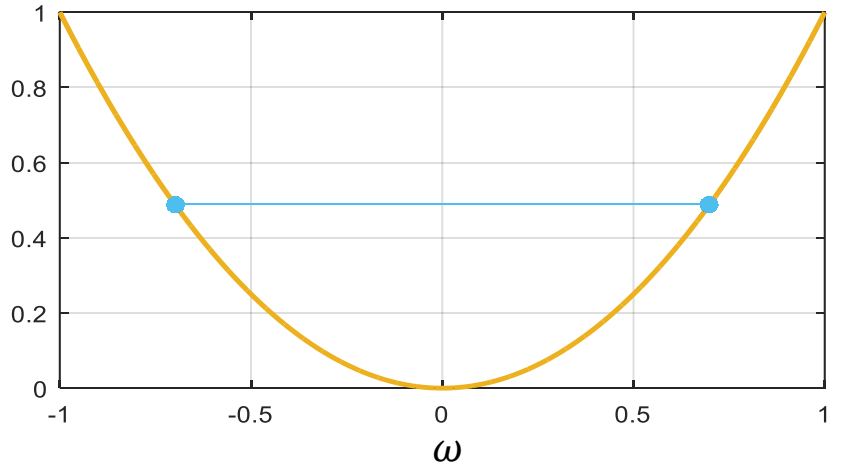








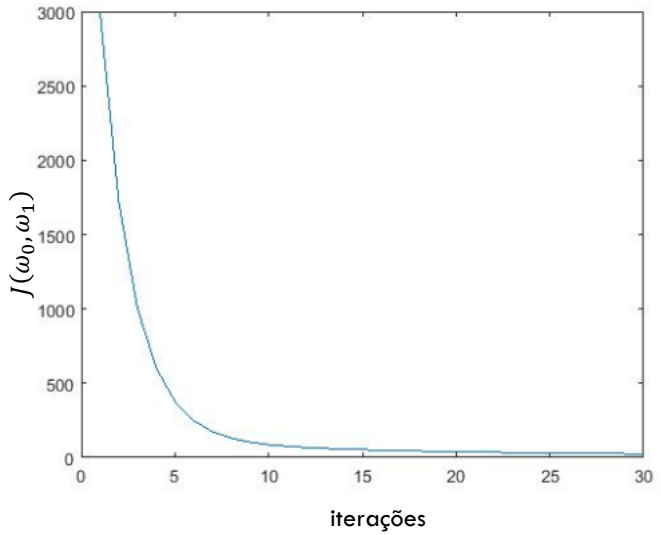




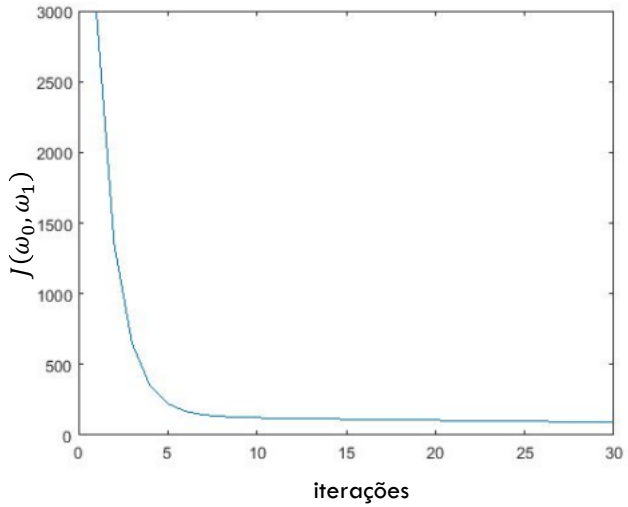
FUNÇÃO CUSTO COM α IDEAL

Diminui acentuadamente e depois se torna mais suave...

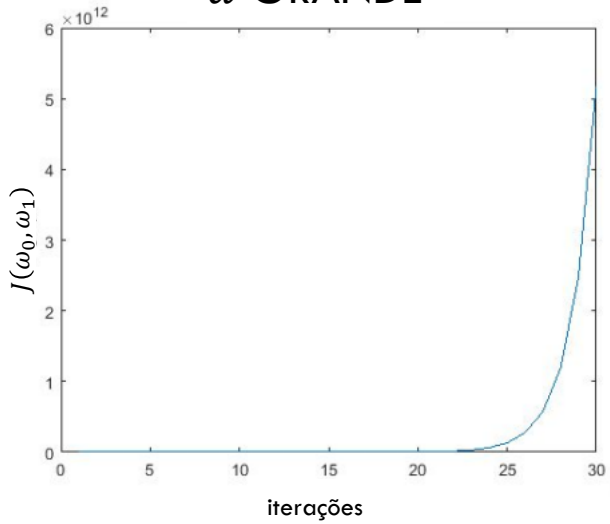
α IDEAL

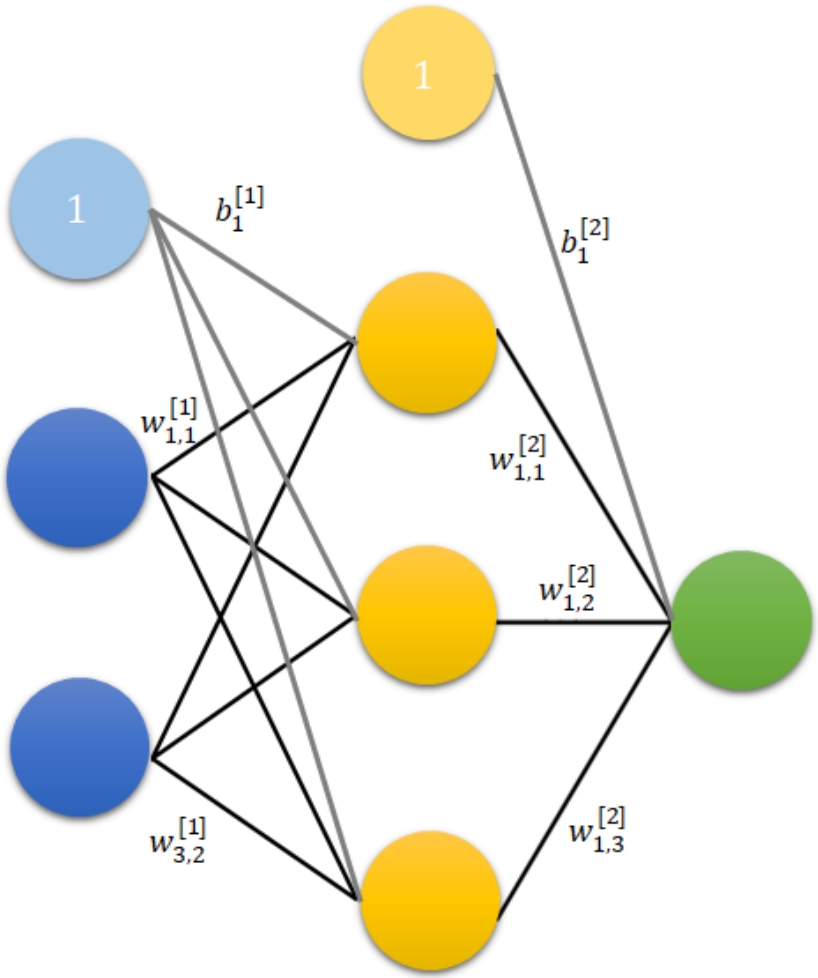


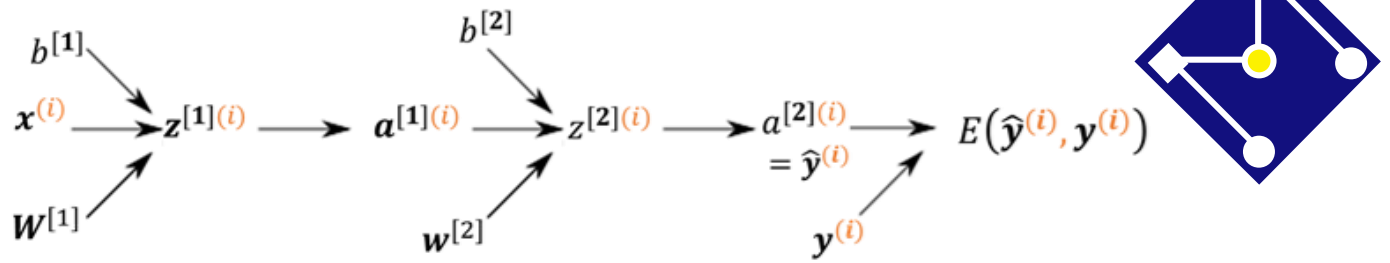
α PEQUENO

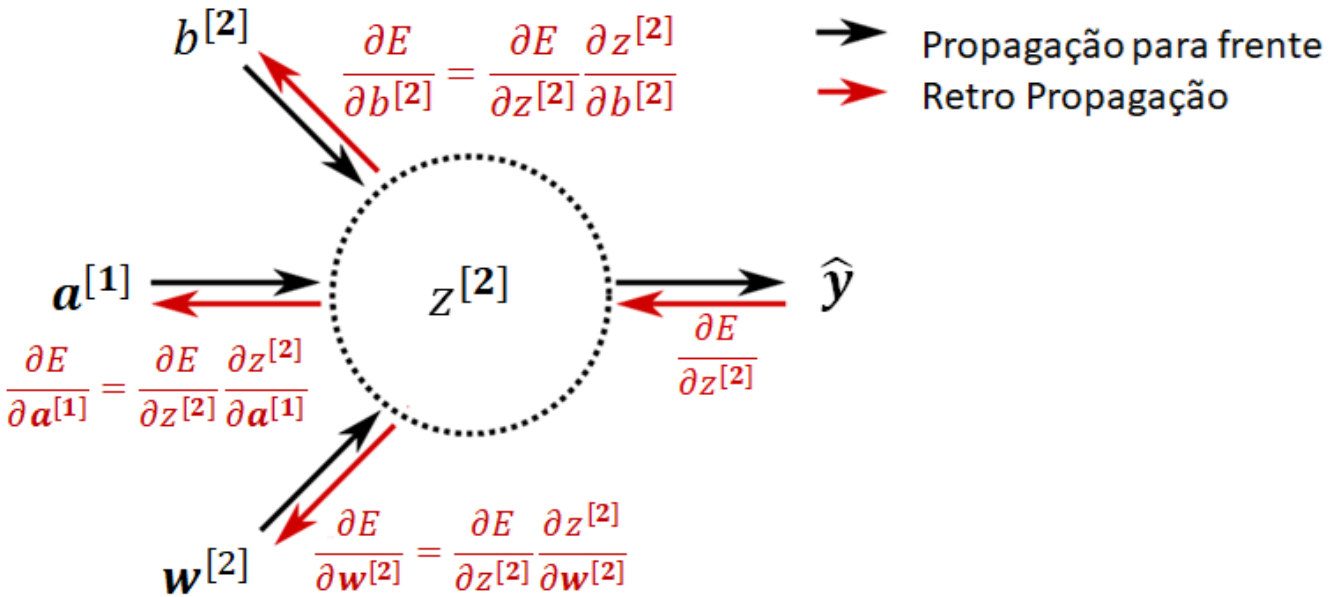


α GRANDE

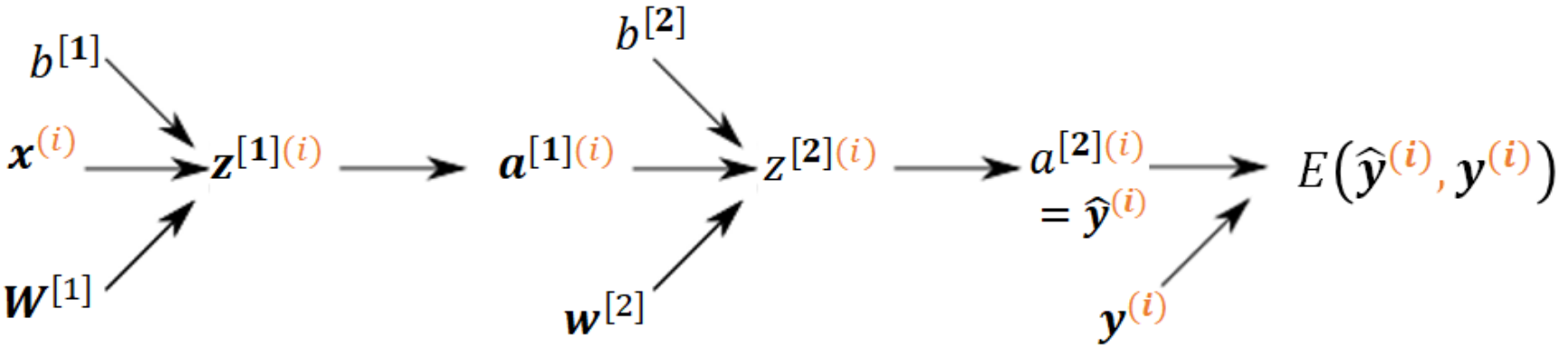



1
1
$b_1^{[1]}$
$b_1^{[2]}$
$w_{1,1}^{[1]}$
$w_{1,1}^{[2]}$
$w_{1,2}^{[2]}$
$w_{3,2}^{[1]}$
$w_{1,3}^{[2]}$


$b^{[1]}$
$x^{(i)}$
$W^{[1]}$
$z^{[1](i)}$
$a^{[1](i)}$
$b^{[2]}$
$w^{[2]}$
$z^{[2](i)}$
$a^{[2](i)} = \hat{y}^{(i)}$
$y^{(i)}$
$E(\hat{y}^{(i)}, y^{(i)})$


$b^{[2]}$
$\frac{\partial E}{\partial b^{[2]}} = \frac{\partial E}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial b^{[2]}}$
Propagação para frente
Retro Propagação
$a^{[1]}$
$z^{[2]}$
$\hat{y}$
$\frac{\partial E}{\partial a^{[1]}} = \frac{\partial E}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial a^{[1]}}$
$\frac{\partial E}{\partial z^{[2]}}$
$w^{[2]}$
$\frac{\partial E}{\partial w^{[2]}} = \frac{\partial E}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial w^{[2]}}$

PORQUE *BACKPROPAGATION*?



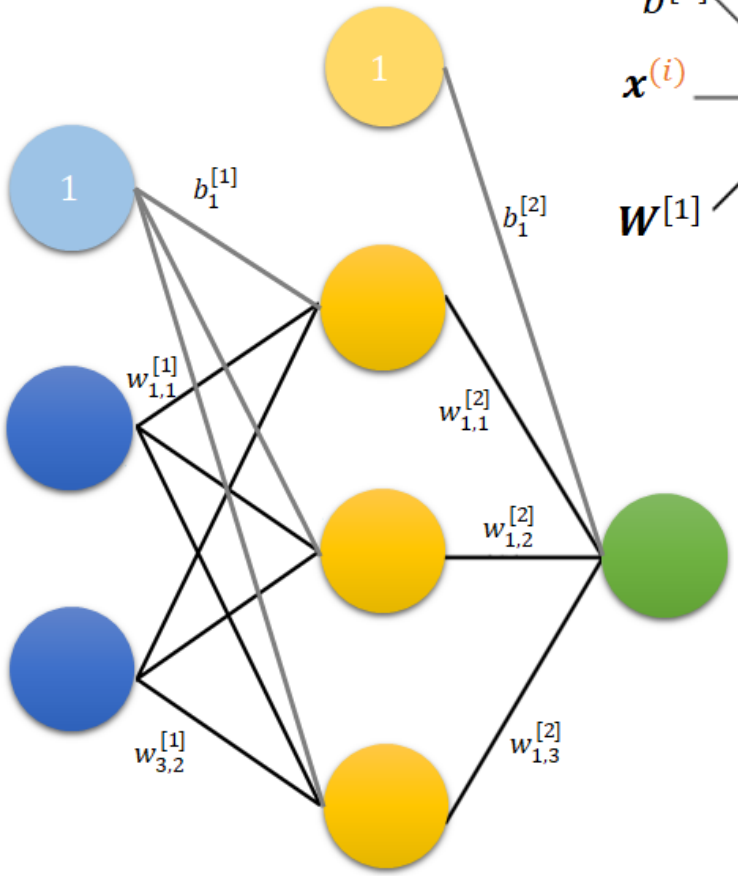
$$J(\mathbf{W}, \mathbf{b}) = \frac{1}{m} \sum_{i=1}^{m} E\left(\hat{y}^{(i)}, y^{(i)}\right)$$

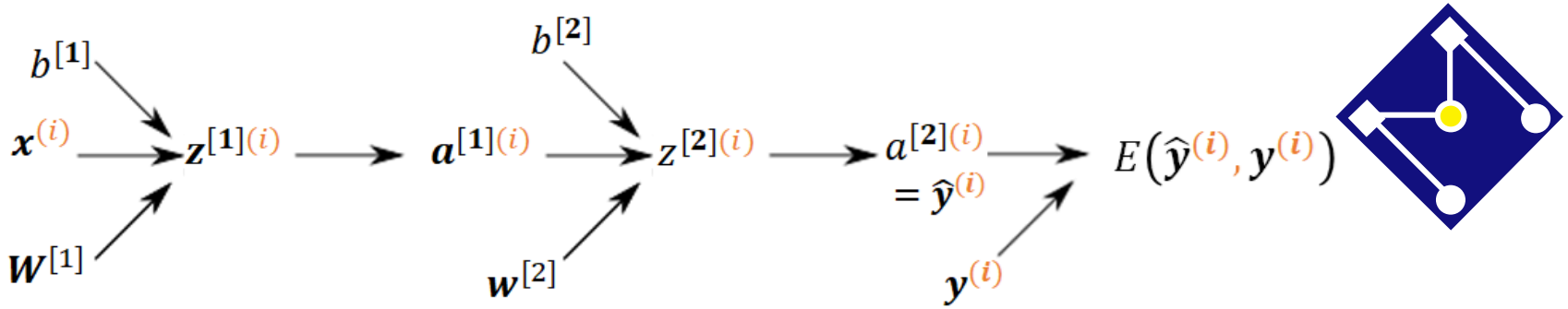
$$\frac{\partial E\left(\hat{y}^{(i)}, y^{(i)}\right)}{\partial w_{k,j}^{[l]}} = \frac{\partial E\left(\hat{y}^{(i)}, y^{(i)}\right)}{\partial a_k^{[l](i)}} \frac{\partial a_k^{[l](i)}}{\partial z_k^{[l](i)}} \frac{\partial z_k^{[l](i)}}{\partial w_{k,j}^{[l]}}$$

$$\frac{\partial E\left(\hat{y}^{(i)}, y^{(i)}\right)}{\partial b_k^{[l]}} = \frac{\partial L\left(\hat{y}^{(i)}, y^{(i)}\right)}{\partial a_k^{[l](i)}} \frac{\partial a_k^{[l](i)}}{\partial z_k^{[l](i)}} \frac{\partial z_k^{[l](i)}}{\partial b_k^{[l]}}$$

$$z_k^{[L](i)} = w_{k,j}^{[L]} a_k^{[L-1](i)} + b_k^{[L]}$$

$$a_k^{[L](i)} = g\left(z_k^{[L](i)}\right) = \hat{y}^{(i)}$$





$$\frac{\partial E}{\partial w_k^{[2]}} = \frac{\partial E}{\partial \hat{y}^{(i)}} \frac{\partial \hat{y}^{(i)}}{\partial w_k^{[2]}} = \frac{\partial E}{\partial \hat{y}^{(i)}} \frac{\partial \hat{y}^{(i)}}{\partial z^{[2](i)}} \frac{\partial z^{[2](i)}}{\partial w_k^{[2]}}$$

$$a^{[2](i)} = g\left(z^{[2](i)}\right)$$

$$z^{[2](i)} = \mathbf{w}^{[2]} \cdot \mathbf{a}^{[1](i)} + b_k^{[2]}$$

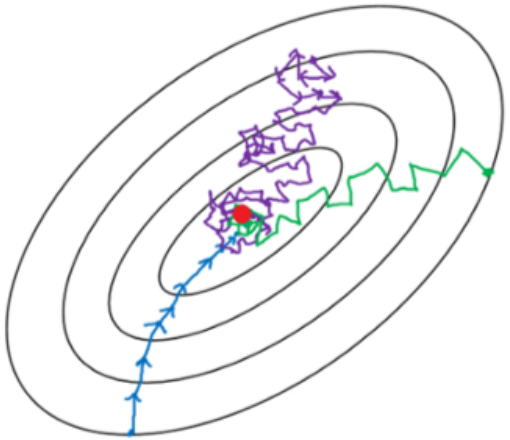
$$\frac{\partial E}{\partial w_{k,j}^{[1]}} = \frac{\partial E}{\partial a^{[2](i)}} \frac{\partial a^{[2](i)}}{\partial w_{k,j}^{[1]}} = \frac{\partial E}{\partial a^{[2](i)}} \frac{\partial a^{[2](i)}}{\partial z^{[2](i)}} \frac{\partial z^{[2](i)}}{\partial w_{k,j}^{[1]}} = \frac{\partial E}{\partial a^{[2](i)}} \frac{\partial a^{[2](i)}}{\partial z^{[2](i)}} \frac{\partial z^{[2](i)}}{\partial a_k^{[1](i)}} \frac{\partial a_k^{[1]}}{\partial w_{k,j}^{[1]}}$$

$$= \boxed{\frac{\partial E}{\partial a^{[2]}} \frac{\partial a^{[2](i)}}{\partial z^{[2](i)}} \frac{\partial z^{[2](i)}}{\partial a_k^{[1](i)}}} \frac{\partial a_k^{[1](i)}}{\partial z_k^{[1](i)}} \frac{\partial z_k^{[1](i)}}{\partial w_{k,j}^{[1]}}$$

Pode-se reutilizar alguns dos cálculos realizados durante a avaliação do gradiente.

VARIAÇÕES DO GRADIENTE DESCENDENTE

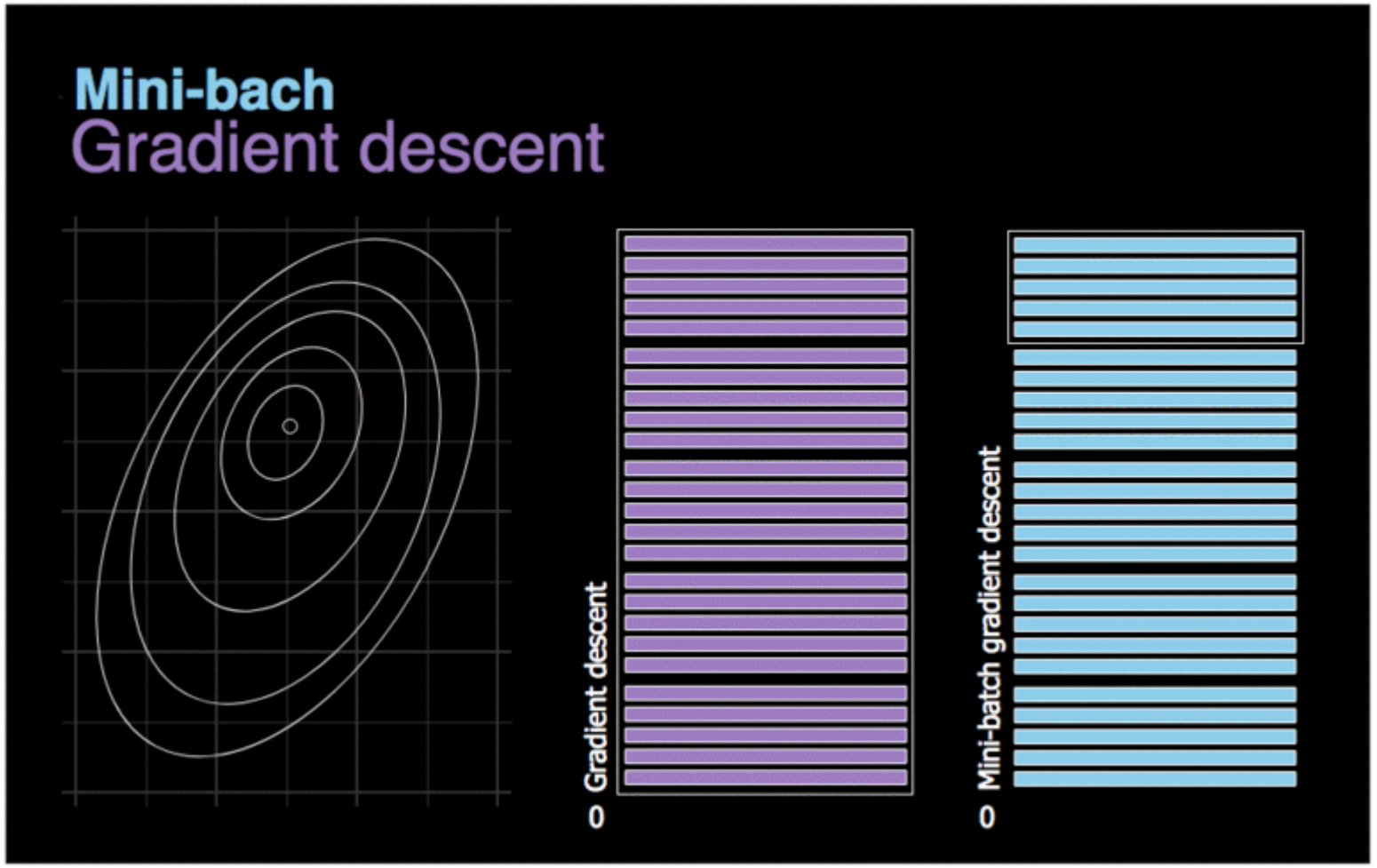
Gradiente descendente em lote (Batch Gradient Descent, BGD): calcula-se o **gradiente** usando **todo o dataset de treinamento** em cada iteração, para atualizar os parâmetros. Portanto, o que fizemos até agora...



Mas se o número de exemplos de treinamento for grande, então o gradiente descendente em lote é computacionalmente muito caro! Imagine se você tem 10000 dados, cada dado com 10 features, são100mil valores para computar a cada iteração, em cada época...

Gradiente descendente em mini lotes (Mini-batch Gradient Descent, MBGD): Este é um tipo de gradiente de descida que funciona mais rápido. Calcula-se o **gradiente** usando $b < m$ **dados do dataset** em cada interação, para atualizar os parâmetros.

Gradiente descendente estocástico (Stochastic Gradient Descent, SGD): calcula-se o **gradiente** usando $b = 1$ **dados aleatórios de treinamento por iteração**, para atualizar os parâmetros. O SGD converge mais rapidamente para conjuntos de dados maiores. Porém, como no SGD usamos apenas um dado de cada vez, não podemos usar implementação vetorizada. Isso pode desacelerar os cálculos.


Mini-bach
Gradient descent
Gradient descent
Mini-batch gradient descent

CADA ÉPOCA (EPOCH)

Gradiente descendente em lote (BGD)
Tomamos a média dos gradientes de **todos os exemplos de treinamento** e usamos esse gradiente médio para atualizar nossos parâmetros.

Gradiente descendente estocástico (SGD):

Tomamos **um exemplo de treinamento** para cálculo do gradiente e usamos esse gradiente médio para atualizar nossos parâmetros.

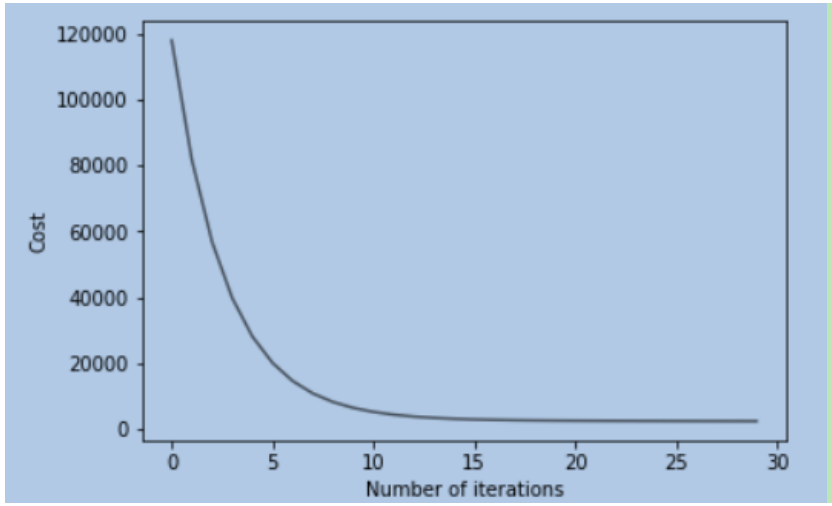
Gradiente descendente em mini lotes (MBGD)

O mini lote tenta encontrar um equilíbrio entre **BGD** e **SGD**.

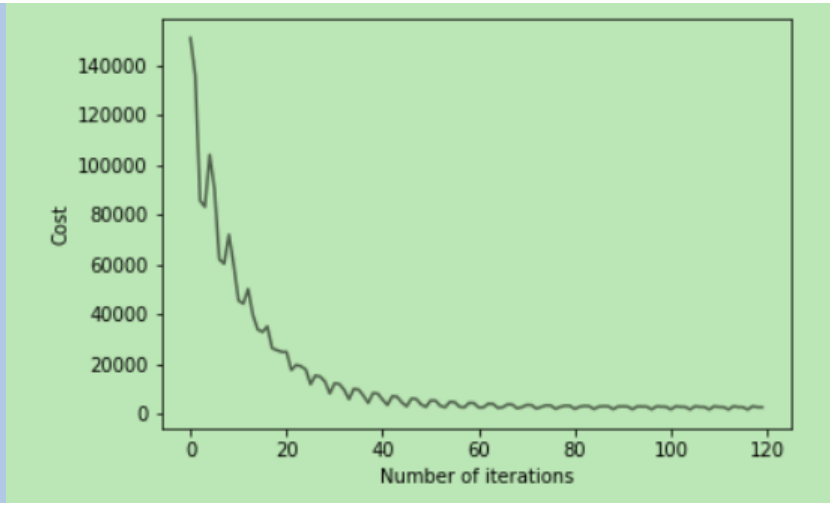
Para cada época:

1. Use os dados de treinamento: **O LOTE**, **1 MINI LOTE** ou **UM ÚNICO DADO**
2. Calcule o gradiente
3. Use o gradiente calculado na para atualizar os pesos
4. Repita as etapas 1 a 3 para todos os exemplos no conjunto de dados de treinamento para o número total de épocas.

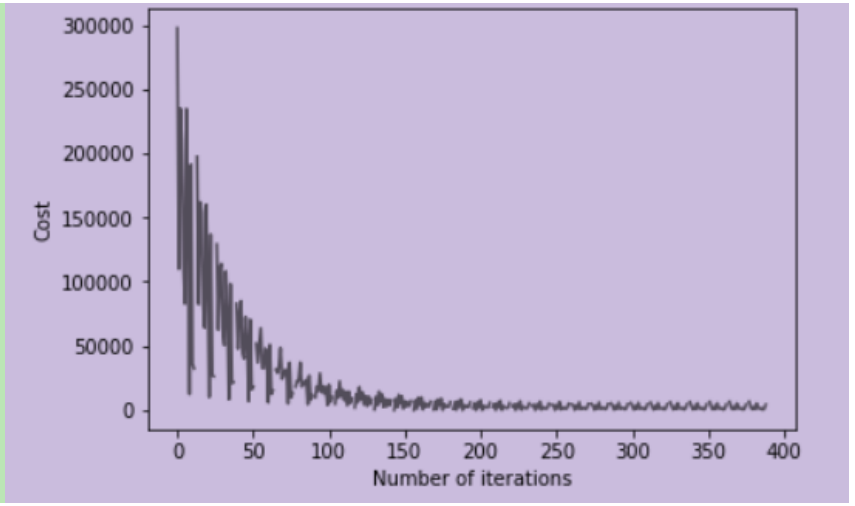
Batch Gradient Descent, BGD

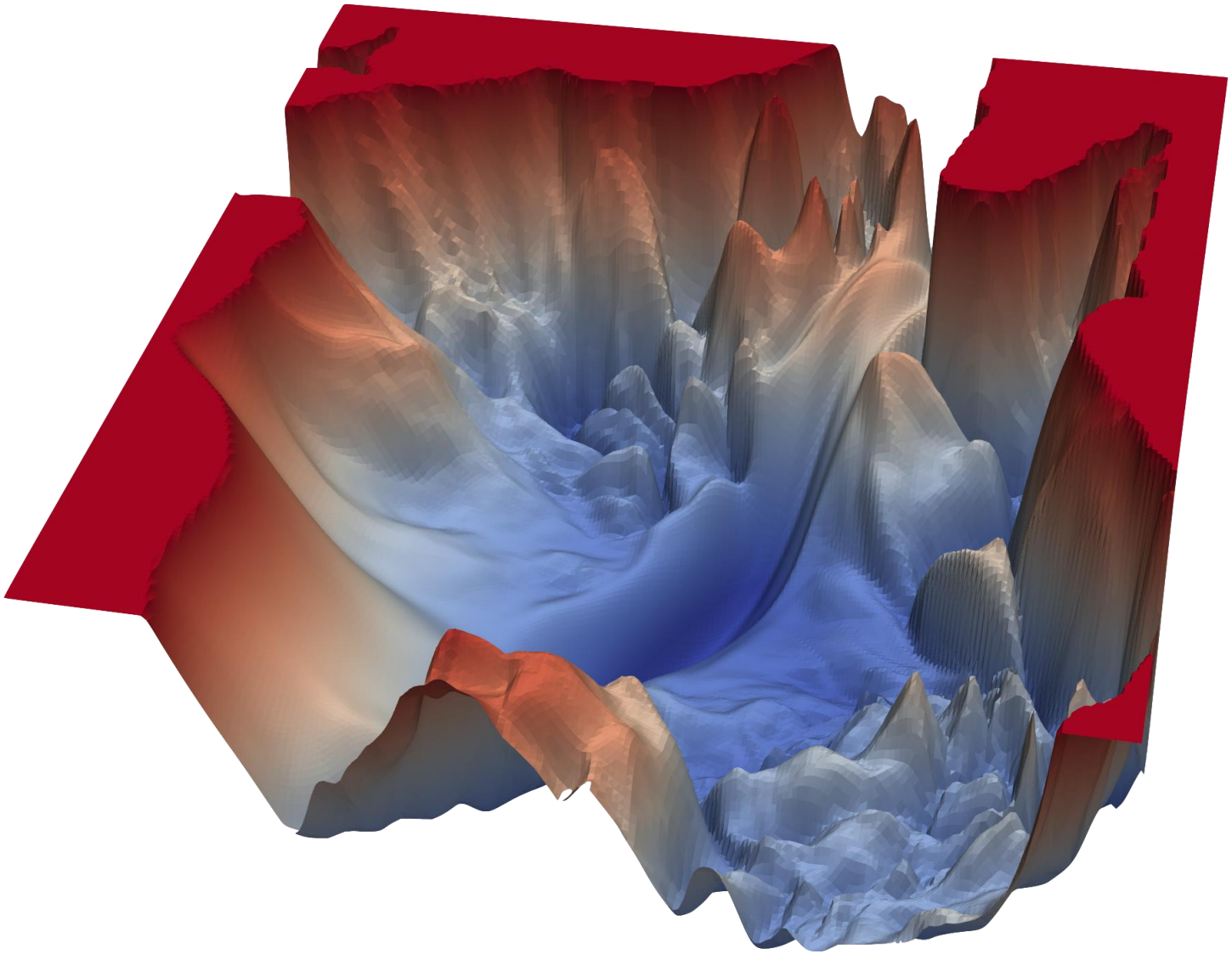


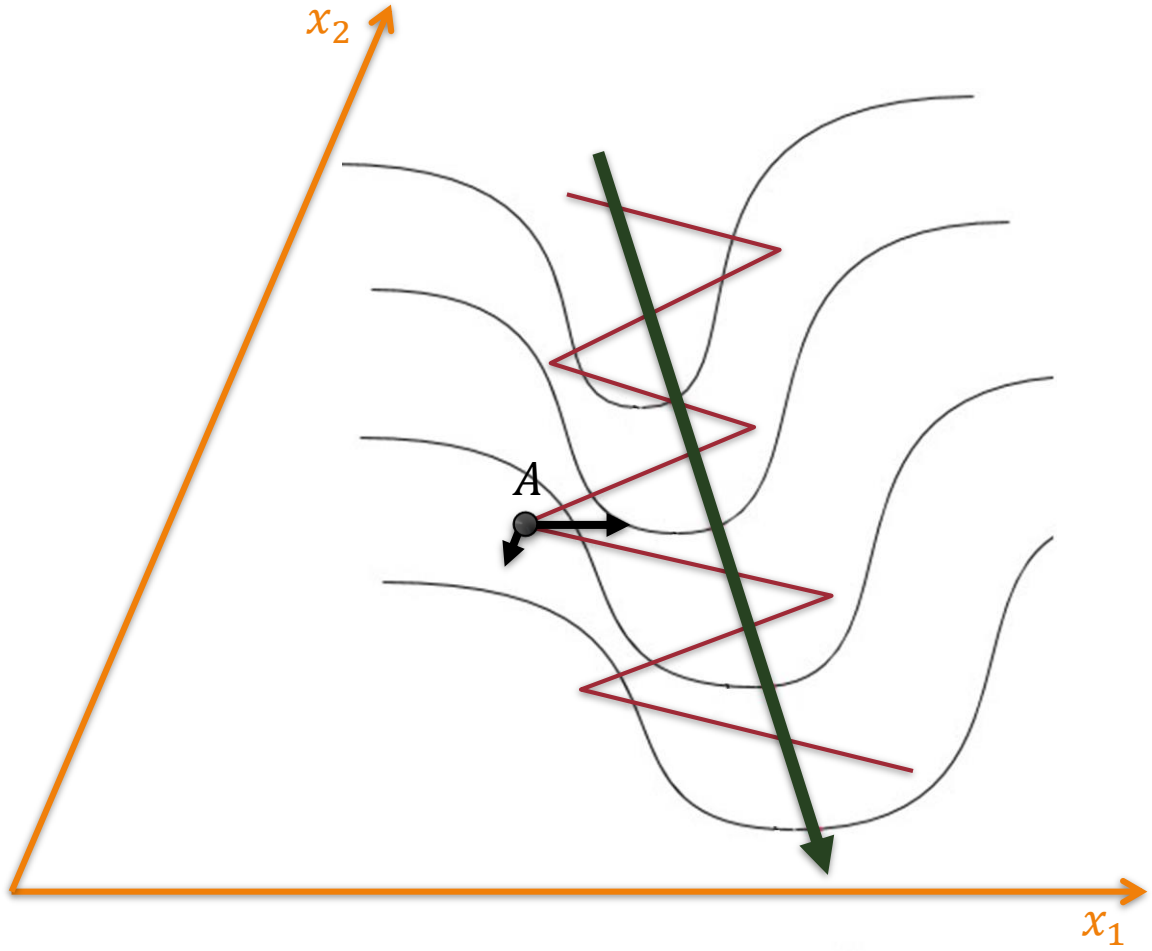
Mini-batch Gradient Descent, MBGD

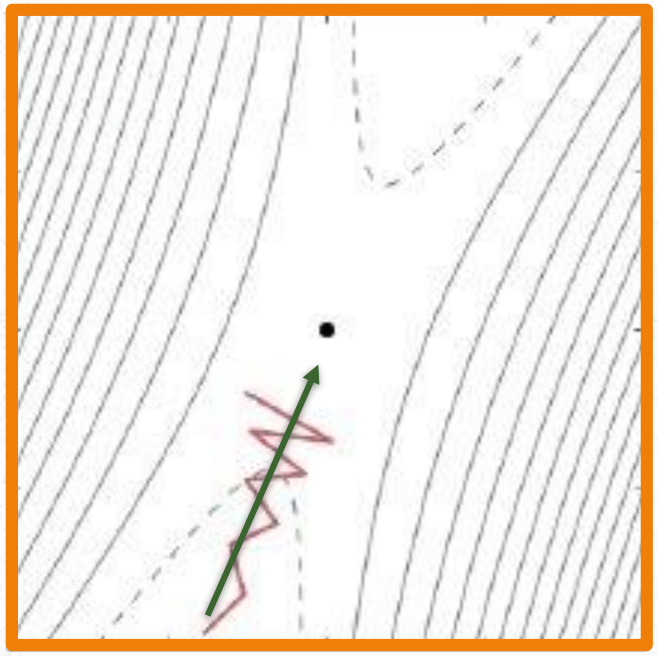


Stochastic Gradient Descent, SGD

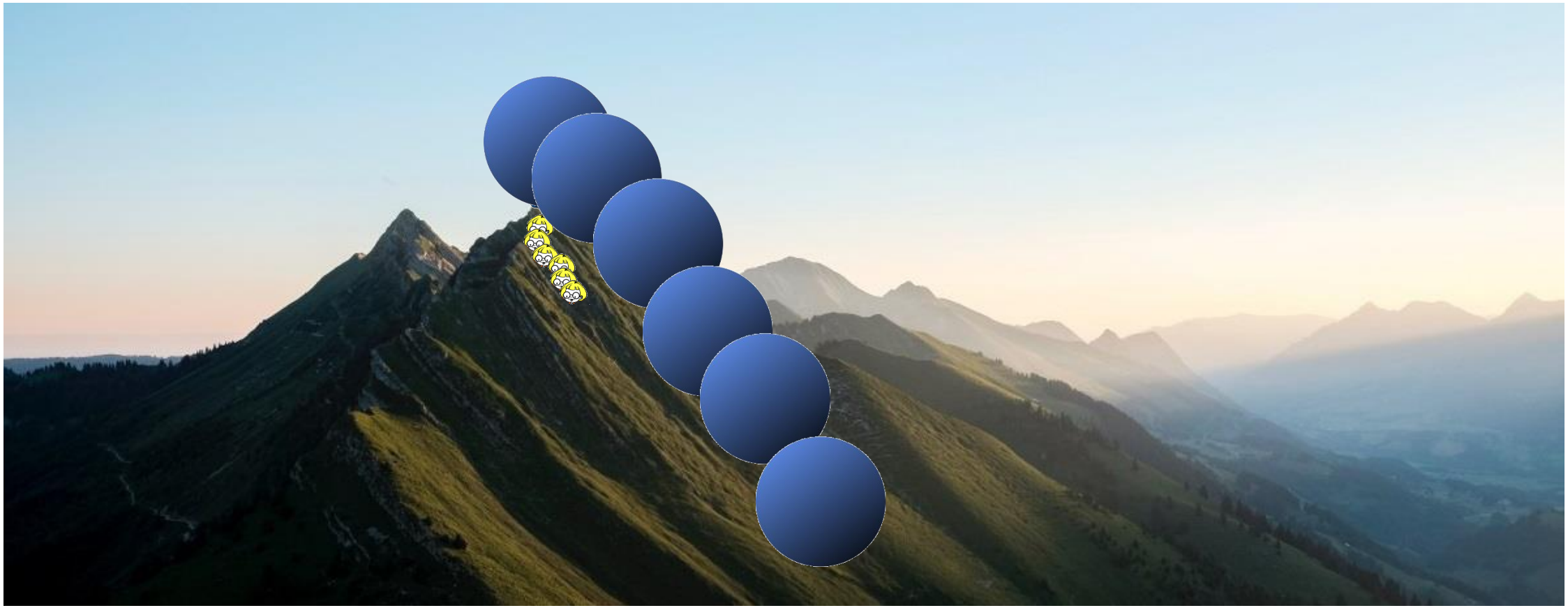


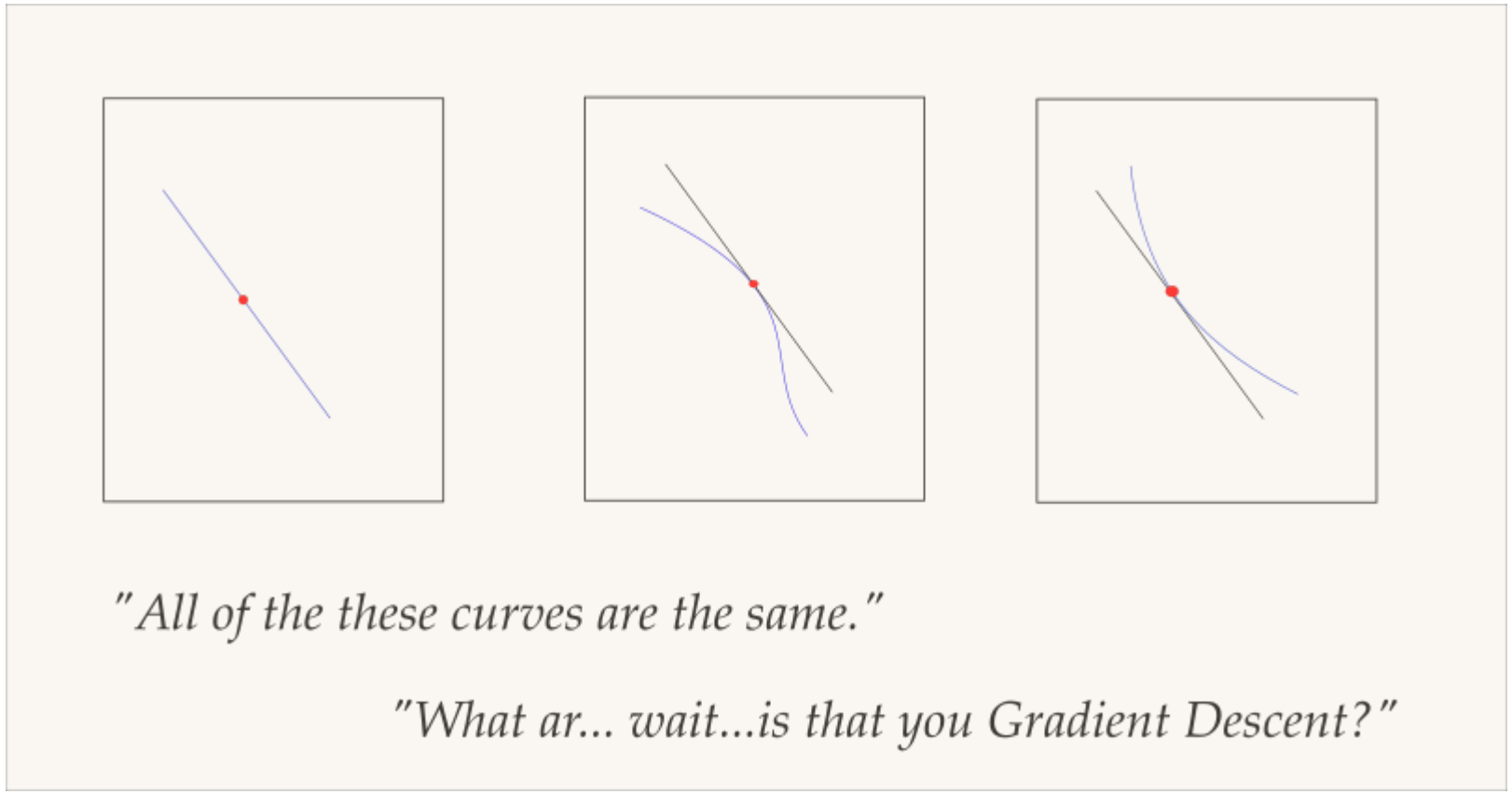



x_2
A
x_1



PROBLEMA DE GRADIENTE DESCENDENTE





"Gradient descent is a **First Order Optimization Method**. It only takes the first order derivatives of the loss function into account and not the higher ones. What this basically means it has no clue about the curvature of the loss function. **It can tell whether the loss is declining and how fast, but cannot differentiate between whether the curve is a plane, curving upwards or curving downwards.**"

https://blog.paperspace.com/intro-to-optimization-momentum-rmsprop-adam/

MOMENTUM

O Momentum propõe o seguinte ajuste para a descida gradiente.

$$m = \beta m - \alpha \nabla J(w)$$

$$w = w + m$$

Gradiente que é mantido nas iterações anteriores. Este gradiente retido é multiplicado por um valor denominado "Coeficiente de Momentum", que é a porcentagem do gradiente retido a cada iteração.

Se definirmos o valor inicial de m como 0 e escolhermos nosso coeficiente como 0.9, as equações de atualização subsequentes serão,

$$m_1 = -G_1$$

$$m_2 = -0.9G_1 - G_2$$

$$m_3 = -0.9(0.9G_1 - G_2) - G_3 = -0.81G_1 - 0.9G_2 - G_3$$

$$G_i = \alpha \nabla J(w_i)$$

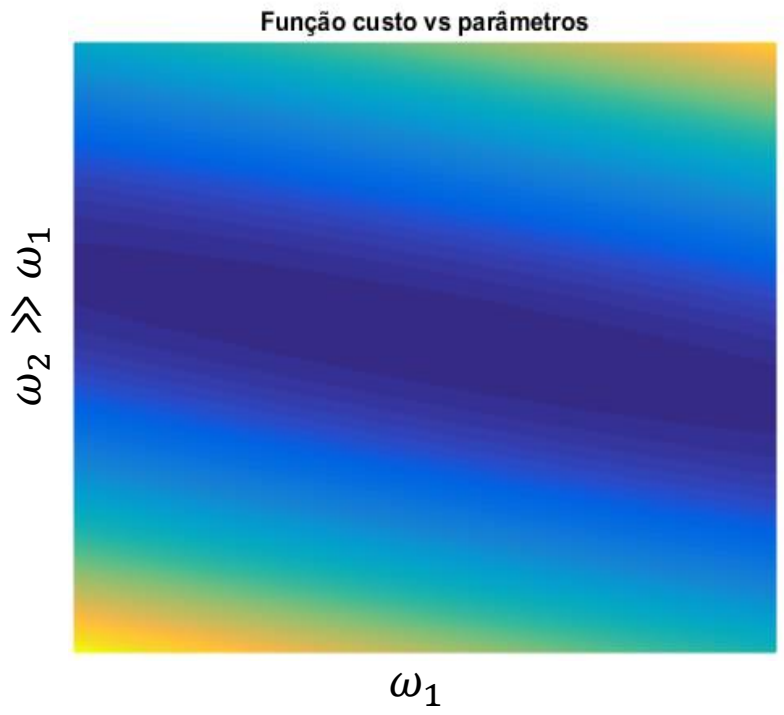
Damos ao gradiente descendente uma memória de curto prazo!

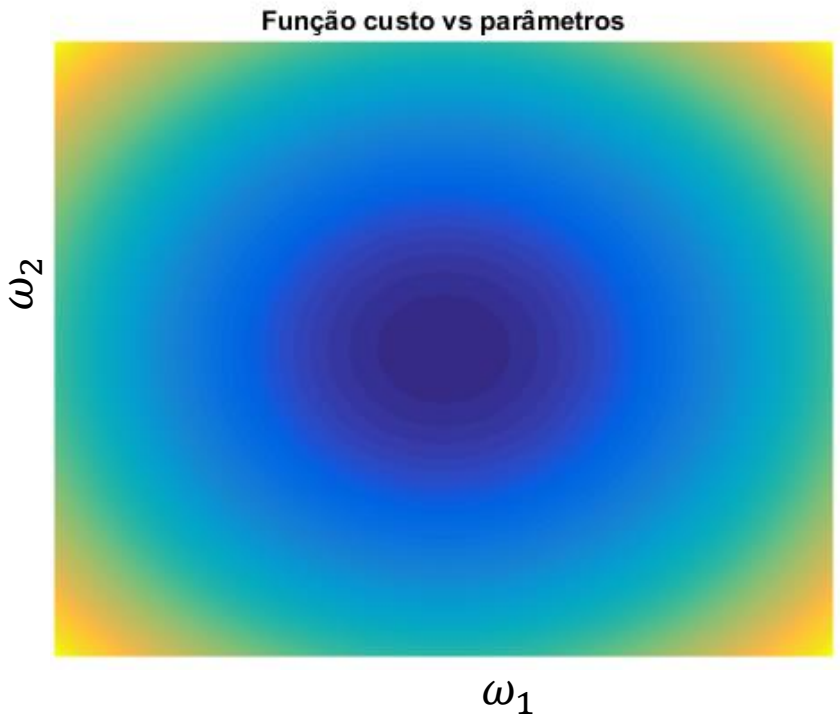
O gradiente descendente é um homem descendo uma colina. Ele segue o caminho mais íngreme para baixo; seu progresso é lento, mas constante. Momentum é uma bola pesada rolando pela mesma colina. A inércia adicionada atua tanto como um suavizador quanto como um acelerador, amortecendo as oscilações e fazendo com que se atravesse direto vales estreitos, pequenas lombadas e mínimos locais.

Figura extraída de:
https://www.itread01.com/content/1543467366.html

ESCALA DOS DADOS DE ENTRADA x_i

Para uma convergência mais rápida, os dados devem estar em uma escala semelhante.





ESCALONAMENTO VS NORMALIZAÇÃO EM ESCORE-Z

Escalonamento/Normalização

$$\bar{x}_i = \frac{x_i - \min \boldsymbol{x}}{\max \boldsymbol{x} - \min \boldsymbol{x}}$$

$$\bar{x}_i = \frac{x_i}{\max \boldsymbol{x}}$$

Padronização

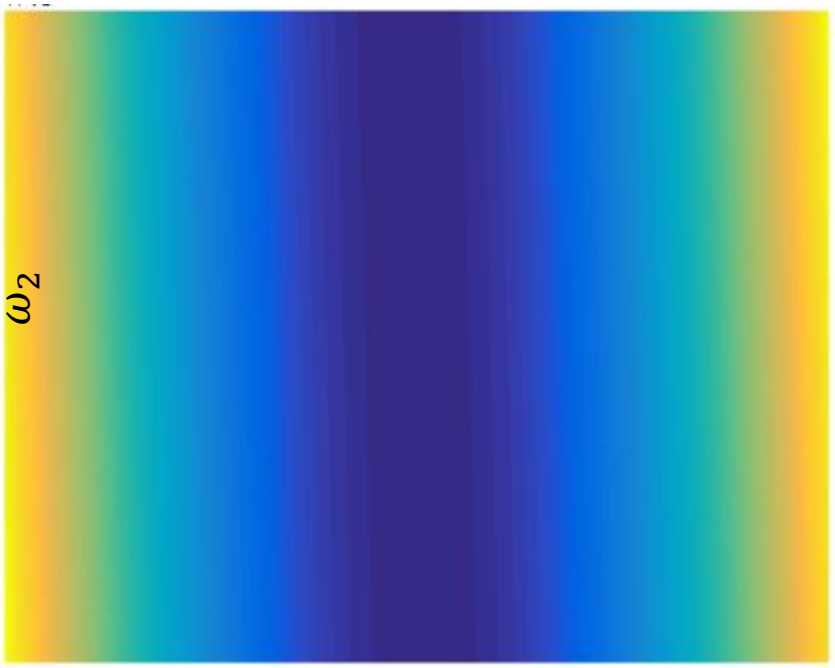
$$\bar{x}_i = \frac{x_i - \mu^{(i)}}{\sigma^{(i)}}$$

Normalização pela média

$$\bar{x}_i = \frac{x_i - \mu^{(i)}}{\max \boldsymbol{x} - \min \boldsymbol{x}}$$

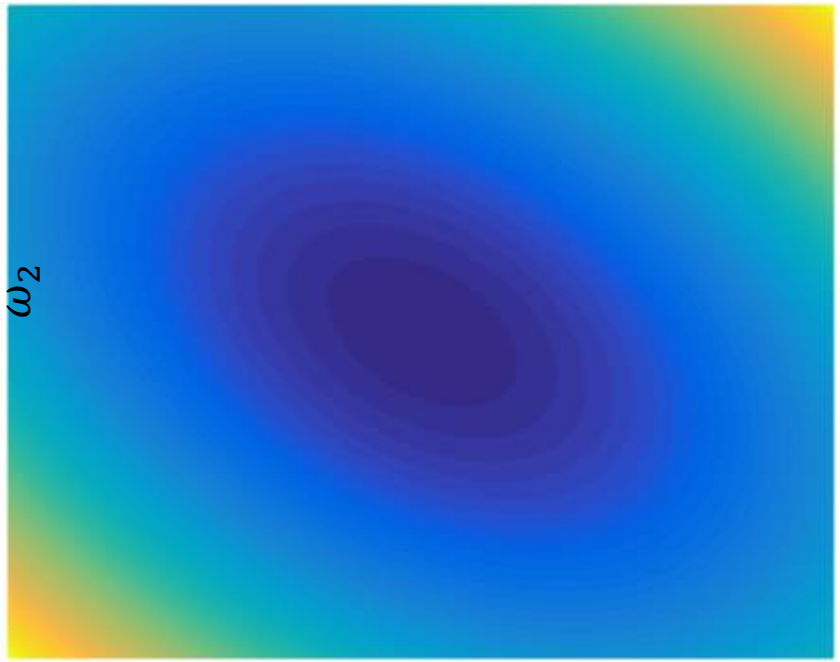
FUNÇÃO CUSTO

Dados brutos



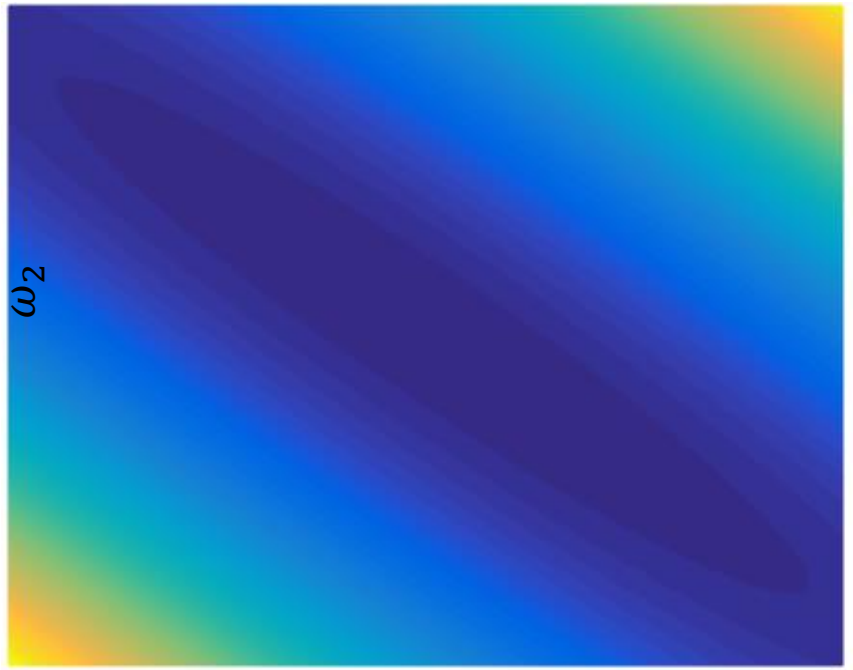
$\omega_1 \gg \omega_2$

Escalonamento



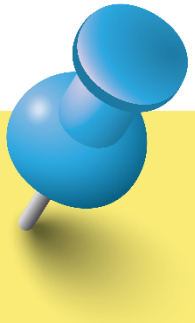
ω_1

Normalização em score-z



ω_1

REVISÃO

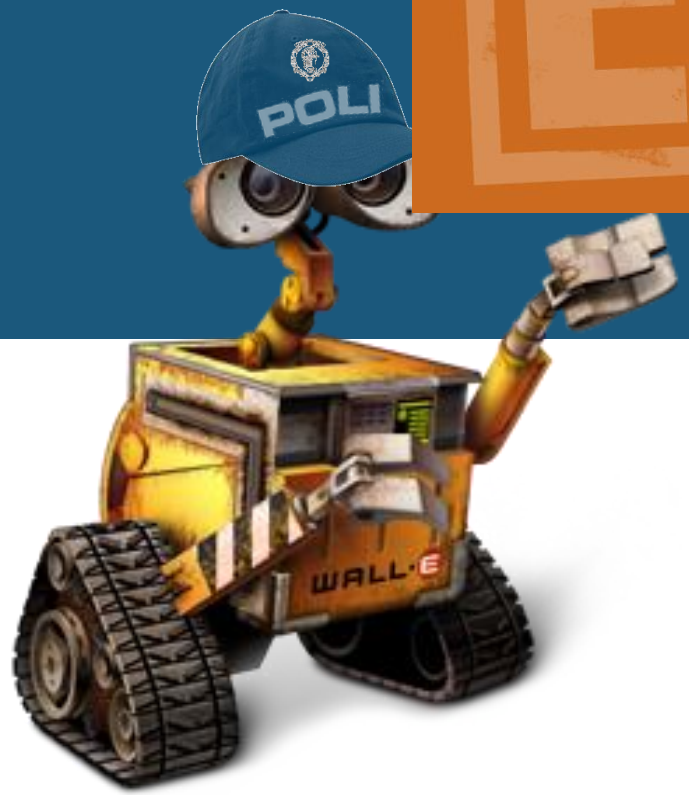


Seu trabalho

Reveja o Notebook e refaça cada etapa.

Faça a lição de casa proposta.

Entrega no Moodle, até 25/10, 23:59. Em dupla.



ENOUGH IS ENOUGH!

ACABOU...

Próxima aula...
Regressão