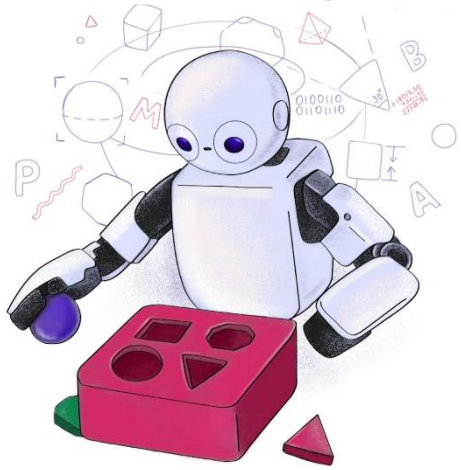
T320 - Introdução ao Aprendizado de Máquina II:
Redes Neurais Artificiais (Parte IV)



Inatel

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

Recapitulando

- No último tópico, discutimos como as redes neurais aprendem.
- Vimos que isso é feito através da minimização de uma função de erro (também chamada de função de custo).
  - Usamos o erro quadrático médio por questões didáticas, mas existem várias outras funções como por exemplo a ***entropia cruzada***, usada para o treinamento de classificadores multi-classes e a ***focal loss*** para o treinamento de detectores de objetos.
- Aprendemos que a minimização da função de erro é realizada de forma iterativamente usando o algoritmo da retropropagação do erro para calcular os vetores gradiente.
- Analisamos como a retropropagação funciona através de um exemplo.
- Neste tópico, iremos discutir algumas questões práticas para o treinamento de redes neurais.

Algumas questões práticas sobre algoritmos de aprendizado

- Podemos dizer que os ***elementos básicos do aprendizado de máquina*** através de ***redes neurais*** foram apresentados até aqui.
- Porém, existem alguns aspectos práticos que nós precisamos discutir.
- Portanto, começamos relembrando sobre a questão do ***cálculo do vetor gradiente***.

Cálculo do vetor gradiente

- Conforme vimos anteriormente, a base para o aprendizado de redes MLP é a obtenção do ***vetor gradiente*** e o estabelecimento de um ***processo iterativo de busca*** dos ***pesos*** que minimizem a ***função de erro***.
- Vimos que a obtenção do ***vetor gradiente*** se dá através do processo de ***retropropagação do erro***, o qual é dividido em duas etapas:
  - Etapa direta (***forward***) onde se apresenta um exemplo de entrada, $\boldsymbol{x}$, e obtém-se a resposta da rede e, consequentemente, o ***erro de saída***.
  - Etapa reversa (***retropropagação***) em que se calculam as derivadas parciais necessárias ao longo das camadas da rede.

Cálculo do vetor gradiente

- Vimos que a derivada parcial do erro em relação a um peso qualquer é a média de ***gradientes particulares (ou locais)***

$$\frac{\partial J(\boldsymbol{W})}{\partial w_{i,j}^{m}} = \frac{1}{N_{\text{dados}} N_M} \sum_{n=1}^{N_{\text{dados}}} \sum_{j=1}^{N_M} \frac{\partial e_j^2(n)}{\partial w_{i,j}^{m}} = \frac{1}{N_{\text{dados}}} \sum_{n=1}^{N_{\text{dados}}} \nabla J_n(\boldsymbol{W}) .$$

Gradiente local

- O ***gradiente local*** é a derivada parcial do erro da j-ésima saída da rede para o n-ésimo exemplo de entrada em relação ao peso $w_{i,j}^{m}$.
- $\nabla J_n(\boldsymbol{W})$ é a média dos N_M ***gradientes locais*** para o n-ésimo exemplo de entrada.
- No entanto, aqui surge um questionamento importante:
  - O que é melhor, usar a ***média dos*** N_M ***gradientes locais***, $\nabla J_n(\boldsymbol{W})$, ***e já dar um passo de otimização***, ou seja, atualizar os pesos, ***reunir o gradiente completo e então dar um passo único e mais preciso*** ou ***um meio termo***?

Cálculo do vetor gradiente

- Esse questionamento gera três abordagens possíveis para o cálculo do vetor gradiente.
  - O cálculo usando todos os exemplos (batelada).
  - O cálculo (i.e., estimativa) usando um único exemplo (estocástica).
  - O cálculo usando um subconjunto de exemplos (mini-*batches*).
- Nas ***redes neurais profundas*** (ou ***deep learning***), usadas com muita frequência em problemas possuem enormes conjuntos de dados, usa-se a abordagem com ***mini-batches***, pois com ela, podemos controlar a complexidade computacional necessária para o treinamento.
- **OBS**.: Os exemplos para estimativa do gradiente das versões ***estocástica*** e ***mini-batch*** devem ser ***aleatoriamente*** escolhidos a partir do conjunto de treinamento.

Variações dos algoritmos de otimização dos pesos

- Existem modificações que podem ser aplicadas às versões estocásticas (mini-*batch* e estocástica) para melhorar seu desempenho sem aumentar muito sua complexidade computacional.
- As modificações mais usadas são:
  - **Redução gradual do passo de aprendizagem.**
  - **Adição do termo momentum.**
  - **Adição do termo momentum de Nesterov.**
  - **Adição de passos de aprendizagem adaptativos.**

Variações dos algoritmos de otimização dos pesos

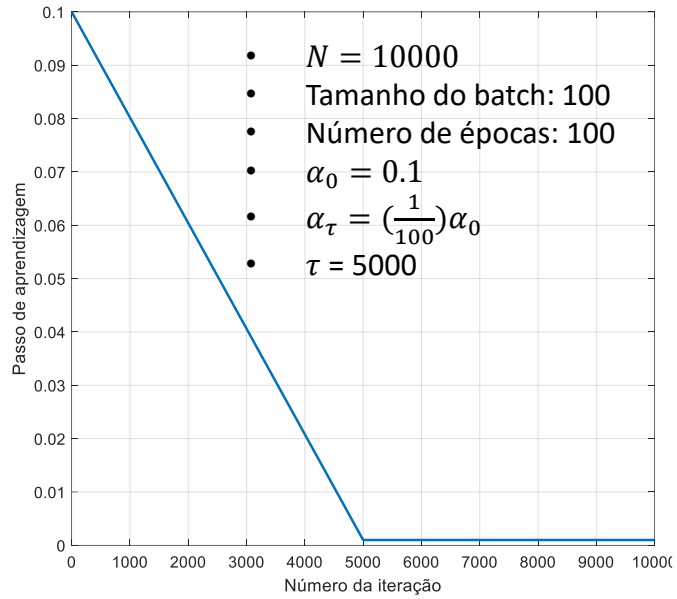
➢Redução gradual do passo de aprendizagem

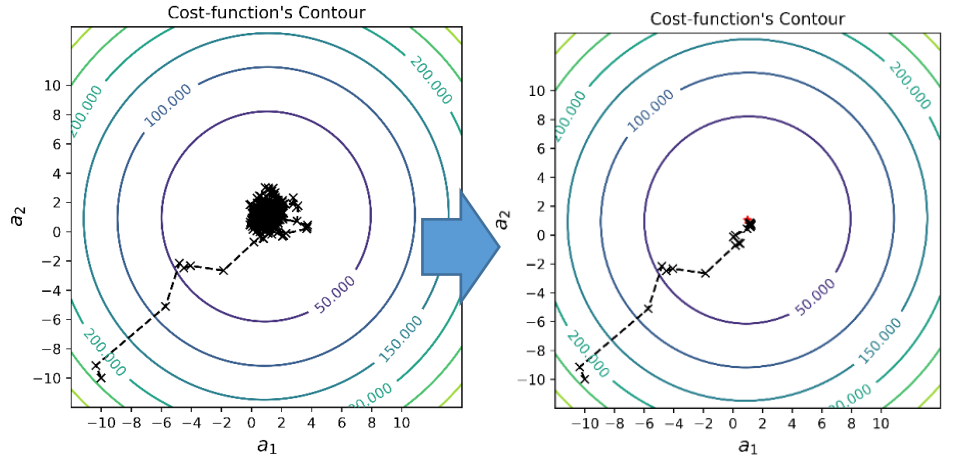
- A escolha do ***passo de aprendizagem***, α, é complicada e exige um compromisso entre velocidade de convergência e estabilidade/precisão.
- Pode-se usar α com um valor fixo, mas, geralmente, para o GDE e MB, se adota uma variação decrescente de um valor α_0 a um valor α_τ (i.e., da iteração 0 à τ-ésima iteração):

$$\alpha_j = \left(1 - \frac{j}{\tau}\right)\alpha_0 + \frac{j}{\tau}\alpha_\tau,$$

onde j é o número da iteração de treinamento.

- Após a τ-ésima iteração, deixa-se o valor do passo de aprendizagem fixo, como mostrado na figura ao lado.
- Porém, a definição dos hiperparâmetros α_0 e α_τ, é mais um problema ***a ser tratado caso-a-caso***.





Variações dos algoritmos de otimização dos pesos

➢ Momentum

- O ***termo momento*** é adicionado à ***equação de atualização dos pesos*** para incorporar ***informação do histórico de gradientes anteriores***.
- Esse termo tem o potencial de ***aumentar a velocidade de convergência*** das versões online e em mini-lotes e ***deixá-las mais estáveis***.
- A ***atualização dos pesos*** com o ***termo momento*** é dada por

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \boldsymbol{v},$$

onde $\boldsymbol{w}$ são os pesos, $\boldsymbol{v}$ é a ***velocidade***, a qual é atualizada da seguinte forma

$$\boldsymbol{v} \leftarrow \mu \boldsymbol{v} + (1 - \mu)\nabla J(\boldsymbol{w}),$$

Média móvel exponencialmente decrescente.

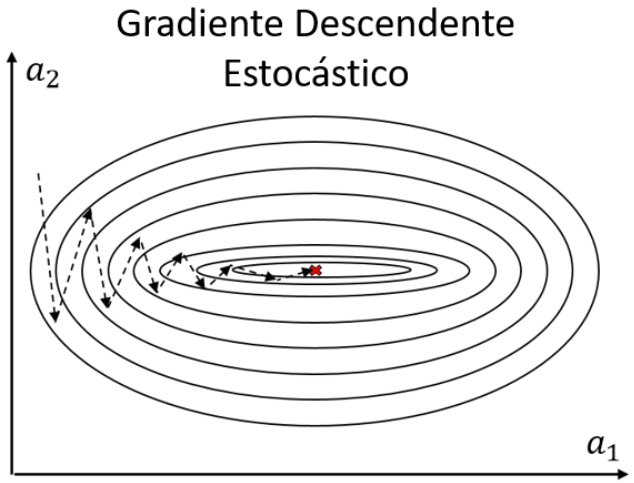
onde, $\nabla J(\boldsymbol{w})$ é o ***vetor gradiente***, α é o ***passo de aprendizagem*** e $\mu \in [0,1)$ é o ***coeficiente de momento*** e determina com que rapidez as contribuições de gradientes anteriores decaem (ou seja, μ é um termo que dita a quantidade de memória).

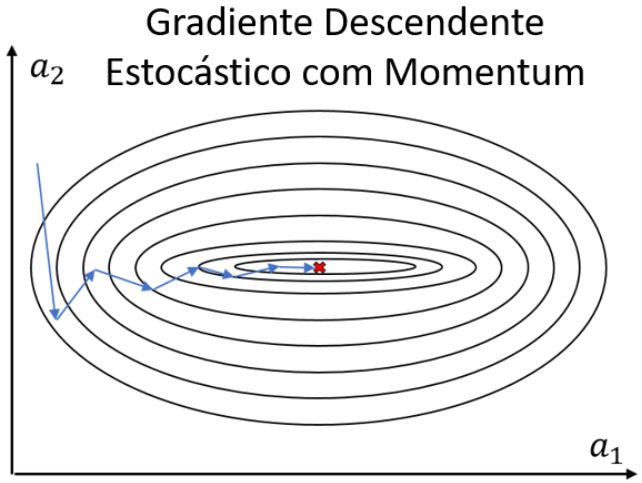
- Quanto maior for μ, maior será a influência de gradientes anteriores na direção atual e quanto menor, menor a influência de gradientes anteriores.
- $\boldsymbol{v}$ dá a ***direção*** e a ***velocidade*** na qual os pesos se movem pelo espaço de pesos.

Variações dos algoritmos de otimização dos pesos

➢ Momentum

- Em física, ***momento*** é igual a ***massa de uma partícula vezes sua velocidade***. A partícula é o vetor de pesos, $\boldsymbol{w}$.
- No algoritmo do momento, assumimos que a massa é unitária, então o vetor velocidade $\boldsymbol{v}$ também pode ser considerado como o momento da partícula.
- O termo momento adiciona uma média dos gradientes anteriores à atualização corrente.
  - Quando o gradiente aponta na mesma direção por várias iterações, o termo aumenta o tamanho dos passos dados naquela direção.
  - Quando o gradiente muda de direção a cada nova iteração, o termo momento suaviza as variações (figura ao lado).
  - Como resultado, temos ***convergência mais rápida e oscilação reduzida***.





Variações dos algoritmos de otimização dos pesos

➢ Momento de Nesterov

- O método do ***momento de Nesterov*** é uma variação do ***método do momento*** em que o cálculo do ***vetor gradiente*** não é feito em relação ao vetor de pesos $\boldsymbol{w}$, mas em relação a $\boldsymbol{w} + \mu\boldsymbol{v}$.
- Essa mudança no cálculo do gradiente faz com que o ***momento de Nesterov*** apresente convergência mais rápida e ajustes mais precisos dos pesos do que o momento clássico.

➢ Modelos com Passo de Aprendizagem Adaptativo

- O ***passo de aprendizagem*** é um ***hiperparâmetro difícil de ser ajustado otimamente*** e **bastante relevante para o sucesso do treinamento** de uma rede neural.
- Isso motivou o surgimento de métodos capazes de ajustá-lo ***dinamicamente***.
- Esses métodos ajustam o passo de acordo com o desempenho da rede, i.e., ***informação dos gradientes passados***.
- Além disso, pode-se ter ***passos diferentes para cada peso do modelo***, os quais são atualizados de forma independente.
- Portanto, esses métodos são adequados para redes neurais, onde a ***superfície de erro é bastante irregular e diferente em diferentes dimensões, tornando a atualização dos pesos mais efetiva***.
- Dentre as técnicas mais populares dessa classe estão ***AdaGrad***, ***RMSProp*** e ***Adam.***

Inicialização dos Pesos

- Uma vez que os métodos de treinamento de ***redes neurais MLP*** são iterativos, eles dependem de uma ***inicialização dos pesos***.
- Como os métodos são de ***busca local***, a inicialização pode afetar drasticamente a qualidade da solução obtida.
- O ***ponto de inicialização*** pode determinar se o algoritmo converge, sendo alguns pontos iniciais tão instáveis que o algoritmo encontra dificuldades numéricas (representações numéricas: ***underflow*** e ***overflow***) e falha completamente em convergir (e.g., ***desaparecimento*** e ***explosão*** dos gradientes).
- O ponto de inicialização também pode fazer com que ocorram variações expressivas na ***velocidade de convergência*** (e.g., platôs, pontos de sela).
- Uma questão importante da inicialização dos pesos é "***quebrar a simetria***" entre os ***nós***, ou seja, ***nós*** com a mesma ***função de ativação*** e conectados às mesmas entradas, devem ter pesos iniciais diferentes, caso contrário, eles terão os mesmos pesos ao longo do treinamento.
- Isso, portanto, sugere uma ***abordagem de inicialização aleatória***.

[1] The importance of effective initialization

Inicialização dos Pesos

- Os pesos iniciais são tipicamente obtidos a partir de ***distribuições gaussianas*** ou ***uniformes***, não importando muito qual é usada.
- No entanto, a ***escala da distribuição inicial*** tem um efeito significativo tanto no resultado da otimização quanto na capacidade de generalização da rede.
- A ordem de grandeza desses pesos levanta algumas discussões:
  - Pesos de maior magnitude criam uma maior distinção entre ***nós*** (i.e., a ***quebra de simetria***). Por outro lado, isso pode causar problemas de ***instabilidade***.
  - Pesos de maior magnitude favorecem a propagação de informação, porém, por outro lado, causam preocupações do ponto de vista de regularização (***overfitting***).
  - Pesos de magnitude elevada podem levar os ***nós*** com ***funções de ativação*** do tipo sigmóide a operarem na região de saturação, comprometendo a convergência do algoritmo (***desaparecimento do gradiente***).
  - Pesos de magnitude elevada podem levar os ***nós*** com ***funções de ativação*** do tipo RELU à ***explosão do gradiente*** ou dos ***valores de saída***, deixando a rede muito sensível a mudanças dos valores de entrada.
- Portanto, na sequência listamos algumas ***heurísticas*** para inicialização dos pesos.

Inicialização dos Pesos

- A ideia por trás destas heurísticas é ***manter a média das ativações igual a zero e suas variâncias constantes ao longo das várias camadas da rede***, pois desta forma evita-se o desaparecimento ou a explosão do gradiente.
- Considerando uma camada com m entradas e n saídas, temos as seguintes ***heurísticas*** para inicializar os ***pesos sinápticos*** de seus nós.

Inicialização	**Funções de ativação**	**Distribuição Uniforme** $U(-r, r)$	**Distribuição Normal** $N(0, \sigma^2)$
Xavier/Glorot	Linear (i.e., nenhuma), Tanh, Logística, Softmax	$r = \sqrt{\frac{6}{m+n}}$	$\sigma^2 = \frac{2}{m+n}$
He	ReLU e suas variantes	$r = \sqrt{\frac{6}{m}}$	$\sigma^2 = \frac{2}{m}$
LeCun	SELU	$r = \sqrt{\frac{3}{m}}$	$\sigma^2 = \frac{1}{m}$

- Uma heurística para a inicialização dos ***pesos de bias*** é inicializá-los com ***valores nulos***. Esta heurística é usada pois se mostra bastante eficiente na maioria dos casos.

[2] How to find appropriate initialization values

Redes Neurais MLP com SciKit-Learn

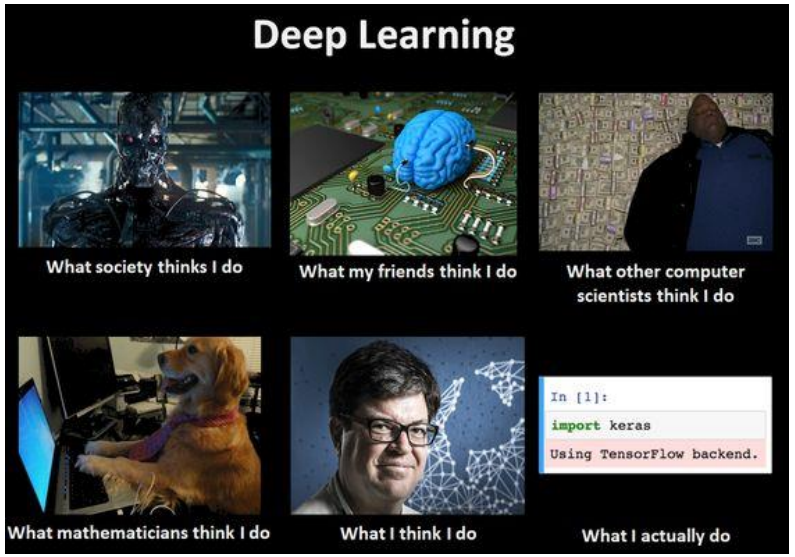
- Como vimos anteriormente, a biblioteca *SciKit-Learn* disponibiliza algumas classes para o treinamento de redes neurais *multi-layer perceptron*.
- Entretanto, suas implementações não são flexíveis e não se destinam a aplicações de larga escala.
  - A biblioteca *SciKit-Learn* não oferece suporte a GPUs.
- Para implementações de ***modelos de aprendizado profundo*** escaláveis, muito mais rápidos, flexíveis e baseados em GPU, devemos utilizar bibliotecas como:
  - ***Tensorflow***: criada pela equipe *Google Brain* do *Google*.
  - ***PyTorch***: criada pela *Meta AI* (antigo *Facebook*).
  - ***MXNet***: criada pela *Apache*.
  - ***Theano***: criada pela Universidade de Montreal (primeira versão) e mantida posteriormente pela equipe de desenvolvedores do pacote PyMC sob o nome de Aesara.
  - Entre outras: https://scikit-learn.org/stable/related_projects.html#related-projects

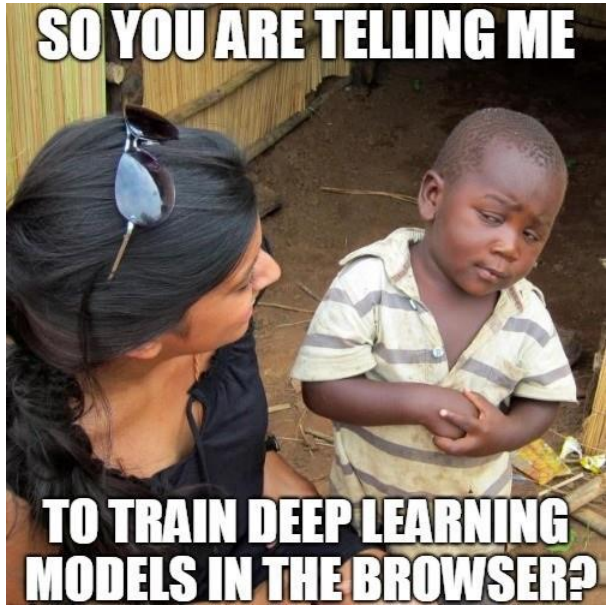
Tarefas

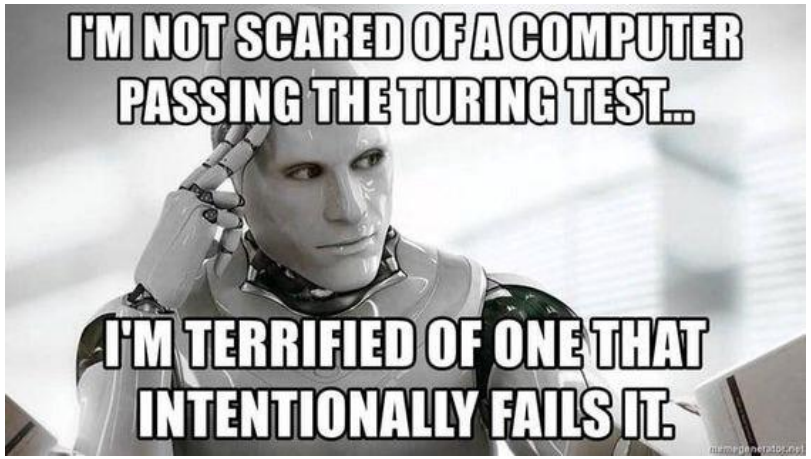
- **Quiz**: “*T320 - Quiz – Redes Neurais Artificiais (Parte VII)*” que se encontra no MS Teams.
- **Projeto**: Projeto #2.
  - Projeto está no github e pode ser feito em grupos de no máximo 3 alunos.
  - **Entrega**: **25/06/2023 até às 23:59**.
  - Leiam os enunciados atentamente.
  - Apenas um integrante do grupo precisa fazer a entrega.
  - **Mas não se esqueçam de colocar os nomes de todos os integrantes do grupo.**

Obrigado!

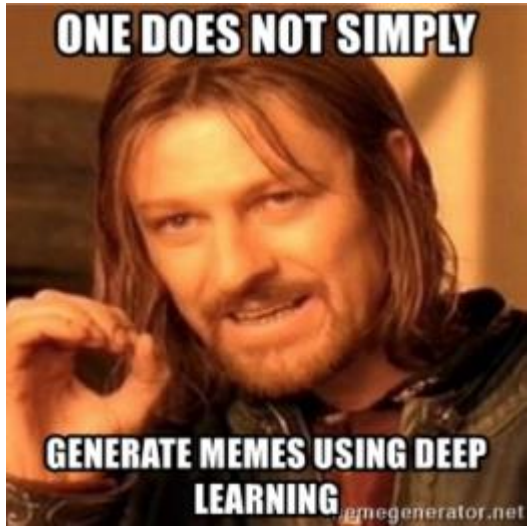

People with no idea about AI, telling me my AI will destroy the world
Me wondering why my neural network is classifying a cat as a dog..
Dog


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

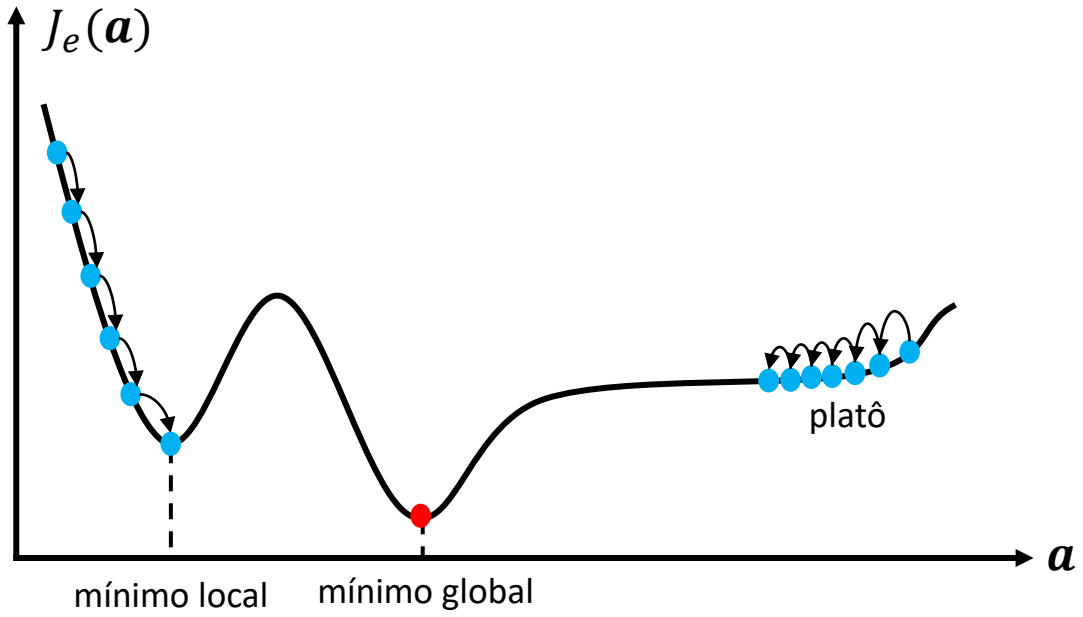

SO YOU ARE TELLING ME
TO TRAIN DEEP LEARNING MODELS IN THE BROWSER?


I'M NOT SCARED OF A COMPUTER PASSING THE TURING TEST...
I'M TERRIFIED OF ONE THAT INTENTIONALLY FAILS IT.


Dog
I NEED GPU FOR MY DUMB NEURAL NETWORK


ONE DOES NOT SIMPLY
GENERATE MEMES USING DEEP LEARNING
memegenerator.net

Figuras


$J_e(\boldsymbol{a})$
platô
$\boldsymbol{a}$
mínimo local
mínimo global

Algumas visões práticas de algoritmos de aprendizado

Versão Online

$$\frac{\partial J(\boldsymbol{x}(n) \mid \boldsymbol{w}(k))}{\partial w_{i,j}^{m}} = \frac{1}{N_M}\sum_{j=1}^{N_M}\frac{\partial\left(d_j(n) - y_j(n) \mid \boldsymbol{w}(k)\right)^2}{\partial w_{i,j}^{m}} = \frac{1}{N_M}\sum_{j=1}^{N_M}\frac{\partial e_j^2(n \mid \boldsymbol{w}(k))}{\partial w_{i,j}^{m}} = \nabla J_n(\boldsymbol{w}(k)).$$