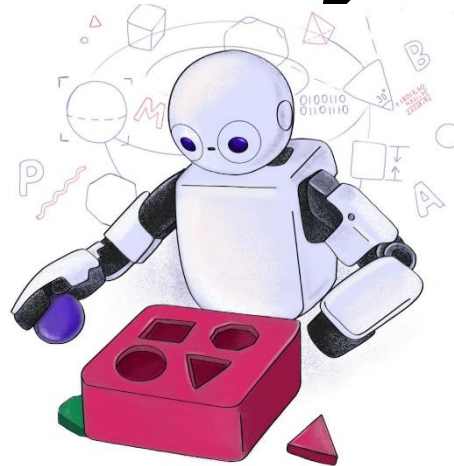


T320 - Introdução ao Aprendizado de Máquina II: *Redes Neurais Artificiais (Parte IV)*



Inatel

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

Recapitulando

- Na última aula, aprendemos como as redes neurais aprendem.
- Vimos que isso é feito através da minimização de uma função de custo.
- Aprendemos qu a minimização é realizada iterativamente com a retropropagação do erro.
- Analisamos como a retropropagação funciona através de um exemplo.
- Nesta aula, iremos discutir algumas visões práticas de algoritmos de aprendizado para redes neurais.

Algumas visões práticas de algoritmos de aprendizado

- Podemos dizer que os elementos básicos do aprendizado de máquina através de redes neurais foram apresentados até aqui.
- Porém, existem importantes aspectos práticos que devem ser comentados de modo que vocês fiquem mais familiarizados com as práticas atuais.
- Começamos falando da questão do cálculo do ***vetor gradiente***.

Algumas visões práticas de algoritmos de aprendizado

Versões Online, Batch e Minibatch

- Conforme vimos nos slides anteriores, a base para o aprendizado em redes MLP é a obtenção do ***vetor gradiente*** e o estabelecimento de um processo iterativo de busca dos ***pesos sinápticos*** que minimizem a ***função de custo***.
- Vimos que a obtenção do ***vetor gradiente*** se dá através de um processo de ***retropropagação do erro*** em que há uma etapa direta (***forward***) de apresentação de um exemplo e obtenção da resposta da rede e uma etapa de ***retropropagação*** em que se calculam as derivadas parciais necessárias.

Algumas visões práticas de algoritmos de aprendizado

Versões Online, Batch e Minibatch

- Vimos também que se calcula o gradiente associado a cada exemplo de entrada e que a combinação de todos esses ***gradientes locais*** leva ao gradiente estimado para o conjunto de exemplos inteiro.

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

- No entanto, surge aqui um questionamento interessante: o que é melhor, usar o gradiente local e já dar um passo de otimização, ou seja, atualizar os pesos, ou reunir o gradiente completo e então dar um passo único e mais preciso?

Algumas visões práticas de algoritmos de aprendizado - Versões Online, Batch e Minibatch

- Nesse questionamento, existem duas abordagens: o cálculo **online** do gradiente (exemplo-a-exemplo) e o cálculo em batelada (**batch**) do gradiente.
- Vejamos inicialmente a noção geral de **adaptação dos pesos sinápticos** com cálculo **online** do gradiente, como expressa o seguinte algoritmo, um método clássico de **primeira ordem**.

- Defina valores iniciais para o vetor de pesos \mathbf{w} e um passo de aprendizagem α pequeno.
- Faça $k = 0, t = 0$ e calcule $J(\mathbf{w}(k))$.
- Enquanto o critério de parada não for atendido, faça:
 - Ordene aleatoriamente os exemplos de entrada/saída.
 - Para l variando de 1 até N , faça:
 - Apresente o exemplo l de entrada à rede.
 - Calcule $J_l(\mathbf{w}(t))$ e $\nabla J_l(\mathbf{w}(t))$.
 - $\mathbf{w}(t + 1) = \mathbf{w}(t) - \alpha \nabla J_l(\mathbf{w}(t)); t = t + 1$.
 - $k = k + 1$.
 - Calcule $J(\mathbf{w}(k))$.

Algumas visões práticas de algoritmos de aprendizado - Versões Online, Batch e Minibatch

- O outro extremo seria utilizar todo o conjunto de dados para estimar o gradiente antes de dar o passo do processo iterativo de aprendizagem.
- Essa é a ideia por trás da abordagem em **batelada (batch)**. O algoritmo abaixo ilustra a operação correspondente (novamente considerando uma metodologia de **primeira ordem**).

- Defina valores iniciais para o vetor de pesos \mathbf{w} e um passo de aprendizagem α pequeno.
- Faça $k = 0$ e calcule $J(\mathbf{w}(k))$.
- Enquanto o critério de parada não for atendido, faça:
 - Para l variando de 1 até N , faça:
 - Apresente o exemplo l de entrada à rede.
 - Calcule $J_l(\mathbf{w}(k))$ e $\nabla J_l(\mathbf{w}(k))$.
 - $\mathbf{w}(k + 1) = \mathbf{w}(k) - \frac{\alpha}{N} \sum_{l=1}^N \nabla J_l(\mathbf{w}(k))$.
 - $k = k + 1$.
 - Calcule $J(\mathbf{w}(k))$.

Algumas visões práticas de algoritmos de aprendizado - Versões Online, Batch e Minibatch

- Nas modernas **redes neurais profundas** (ou **deep learning**), usadas com muita frequência em problemas com conjuntos de dados enormes, a regra é adotar o caminho do meio, usando a abordagem com **mini-batches**.
- Nesse caso, a adaptação dos **pesos** é realizada com um gradiente calculado a partir de um meio-termo entre um exemplo e o número total de exemplos (em geral, este é um valor relativamente pequeno em métodos de **primeira ordem**).
- As amostras que devem compor o **mini-batch** são **aleatoriamente** tomadas do conjunto de dados. O algoritmo abaixo ilustra isso.

- Defina valores iniciais para o vetor de pesos \mathbf{w} e um passo de aprendizagem α pequeno.
- Faça $k = 0$ e calcule $J(\mathbf{w}(k))$.
- Enquanto o critério de parada não for atendido, faça:
 - Para l variando de 1 até m , faça:
 - Apresente o exemplo l de entrada, amostrado aleatoriamente para compor um **minibatch**, à rede.
 - Calcule $J_l(\mathbf{w}(k))$ e $\nabla J_l(\mathbf{w}(k))$.
 - $\mathbf{w}(k + 1) = \mathbf{w}(k) - \frac{\alpha}{m} \sum_{l=1}^m \nabla J_l(\mathbf{w}(k))$.
 - $k = k + 1$.
 - Calcule $J(\mathbf{w}(k))$.

Variações dos algoritmos de otimização dos pesos

- Existem vários algoritmos baseados no **gradiente** que podem ser empregados para otimizar os **pesos sinápticos** de uma rede neural.
- Aqui, vamos nos ater a alguns métodos muito usuais na literatura moderna, que se encontra bastante focada no **apredizado profundo**.
- **Método do Gradiente Estocástico (*Stochastic Gradient Descent*, SGD)**
 - Nos slides anteriores, nós vimos que o método **online** utiliza um único exemplo (que deve ser tomado aleatoriamente) para estimar o gradiente da **função custo**.
 - Este tipo de estimador é o que gera a noção de **gradiente estocástico**. Caso utilizemos **mini-batches**, também teremos uma estimativa do **gradiente**, o qual, a rigor, seria determinístico apenas se usássemos todos os dados (no caso do **batch**).
 - Por esse motivo, esses métodos de **primeira ordem**, como o **online**, são conhecidos como métodos de **stochastic gradient descent** (SGD).

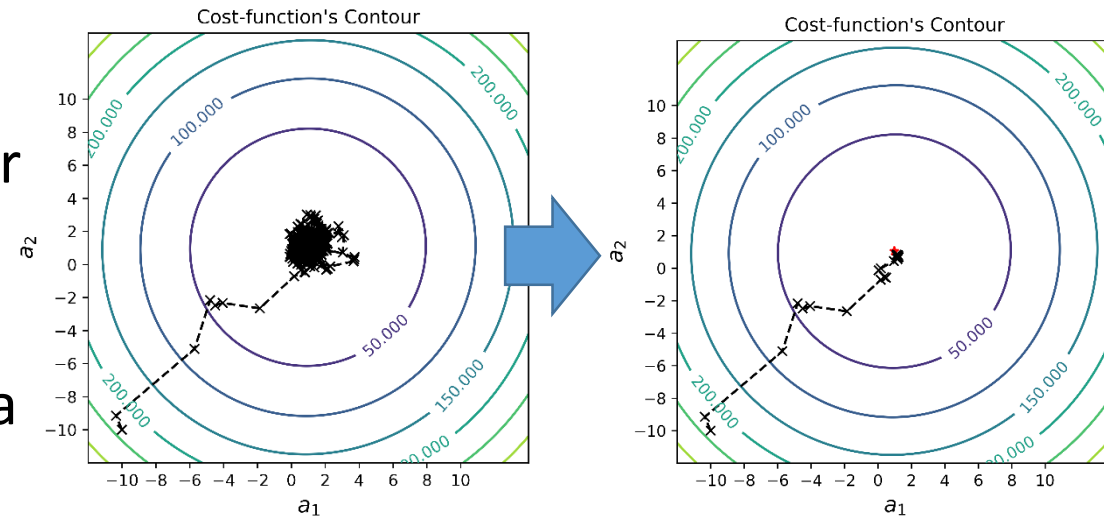
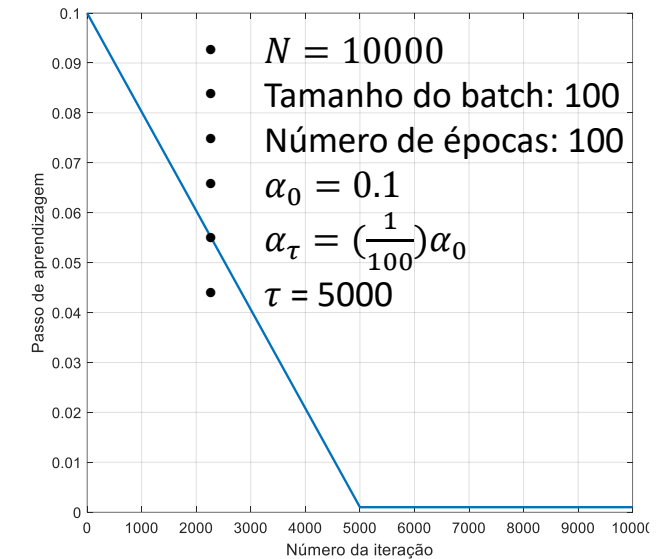
Variações dos algoritmos de otimização dos pesos

- A tarefa de escolha do ***passo de aprendizagem*** é complicada e nos remete ao conhecido compromisso entre velocidade de convergência e estabilidade/precisão.
- Pode-se usar um valor fixo, mas geralmente, se adota um método de variação linear decrescente de um valor α_0 a um valor α_τ (i.e., da iteração 0 à iteração τ):

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

onde j é o número da iteração de treinamento.

- Após a τ -ésima iteração, pode-se deixar o valor do passo de aprendizagem fixo, como mostrado na figura ao lado.
- Naturalmente, a definição dos valores necessários (i.e., 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 trazer ***informação de gradientes anteriores acumulados*** ao ajuste de pesos.
- Isso tem o potencial de melhorar a convergência das versões online e em mini-lotes do gradiente descendente.

- A ***atualização dos pesos*** com o ***termo momento*** é dada por

$$\mathbf{w} \leftarrow \mathbf{w} + \mathbf{v},$$

onde \mathbf{v} é a ***velocidade***, a qual é atualizada da seguinte forma

$$\mathbf{v} \leftarrow \mu \mathbf{v} - \alpha \mathbf{g},$$

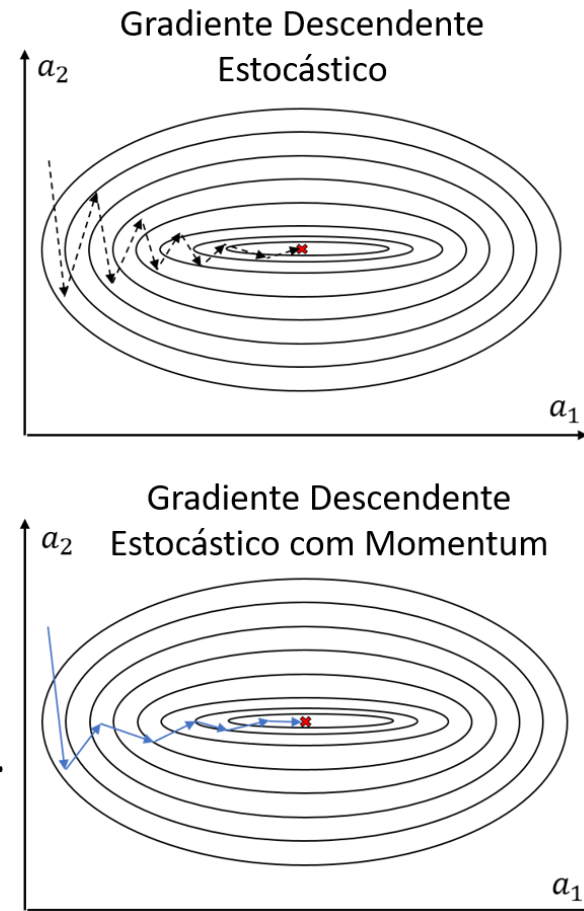
\mathbf{g} é o ***vetor gradiente***, α é o ***passo de aprendizagem*** e $\mu \in [0,1)$ determina com que rapidez as contribuições de gradientes anteriores decaem (ou seja, μ é um termo de memória).

- Quanto maior for μ , maior será a influência de gradientes anteriores na direção atual.
- \mathbf{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

- Momento em física é igual a massa de uma partícula vezes sua velocidade. No algoritmo do momento, assumimos que a massa é unitária, então o vetor velocidade \mathbf{v} também pode ser considerado como o momento da partícula.
- O termo momento adiciona uma fração μ de atualizações anteriores dos pesos.
 - Quando o gradiente continua apontando na mesma direção, isso aumentará o tamanho dos passos dados em direção ao mínimo.
 - Quando o gradiente muda de direção a cada nova iteração, o termo momento suaviza as variações.
 - Como resultado, temos convergência mais rápida e oscilação reduzida.
- O efeito do algoritmo do momento no GDE é ilustrado na figura ao lado.



Variações dos algoritmos de otimização dos pesos

➤ Momento de Nesterov

- O método do ***momento de Nesterov*** pode ser visto, essencialmente, como uma variação do ***método do momento*** em que o cálculo do ***vetor gradiente*** não é feito sobre o vetor de pesos \mathbf{w} , mas sim sobre $\mathbf{w} + \varphi \mathbf{v}$.
- Esse termo adicional funciona como um fator de correção que pode beneficiar, em alguns casos, a velocidade de convergência.

➤ Modelos com Passo de Aprendizagem Adaptativo

- O ***passo de aprendizagem*** é um hiperparâmetro difícil de se ajustar otimamente e bastante relevante para o sucesso do treinamento de uma rede neural.
- Isso motivou o surgimento de um conjunto de métodos com mecanismos capazes de modificá-lo dinamicamente.
- O passo é ajustado de acordo com o desempenho da rede e, além disso, pode-se ter passos diferentes para cada peso do modelo, os quais são atualizados de forma independente.
- Dentre as técnicas mais populares dessa classe estão o ***AdaGrad***, o ***RMSProp*** e o ***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 e falha completamente em convergir.
- Também pode haver variações expressivas na **velocidade de convergência**.
- Um ponto importante da inicialização 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.
- Isso, portanto, sugere uma **abordagem aleatória**.

Inicialização dos Pesos

- Os pesos são tipicamente obtidos de ***distribuições gaussianas*** ou ***uniformes***.
- A ordem de grandeza desses pesos levanta algumas discussões:
 - Pesos de maior magnitude criam 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.
 - Pesos de magnitude elevada podem levar os ***nós*** (no caso de ***funções de ativação*** do tipo sigmóide como a tangente hiperbólica e a função logística) a operarem numa região de saturação, comprometendo a convergência do algoritmo.
- Portanto, na sequência listamos algumas ***heurísticas*** para inicialização dos pesos.

Inicialização dos Pesos

- Considerando uma camada com m entradas e n saídas, uma heurística para inicializar os pesos de nós com função de ativação ***sigmóide*** é

$$w_{i,j} \sim U\left(-\frac{1}{\sqrt{m}}, \frac{1}{\sqrt{m}}\right), \quad \text{Inicialização de Xavier/Glorot}$$

onde $U(.)$ é a ***distribuição uniforme***.

- Outra heurística de inicialização dos pesos de nós com função de ativação ***sigmóide*** é

$$w_{i,j} \sim U\left(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right). \quad \text{Inicialização de Xavier/Glorot Normalizada}$$

- Uma heurística para nós que usam função de ativação ***ReLU*** é dada por

$$w_{i,j} \sim N\left(0, \sqrt{1/m}\right), \quad \text{Inicialização de He}$$

onde $N(.)$ é a ***distribuição Gaussiana***.

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

Redes Neurais MLP com SciKit-Learn

- A biblioteca SciKit-Learn disponibiliza algumas classes para o treinamento de redes neurais multi-layer perceptron.
- Entretanto, as implementações desta biblioteca não se destinam a aplicações de larga escala.
- Em particular, a biblioteca SciKit-Learn não oferece suporte a GPUs.
- Para implementações muito mais rápidas, baseadas em GPU, bem como estruturas que oferecem muito mais flexibilidade para criar arquiteturas de aprendizado profundo, por exemplo, devemos utilizar outras bibliotecas como:
 - **Tensorflow**: biblioteca para desenvolvimento de aplicações eficientes e escaláveis de machine learning.
 - **keras**: uma biblioteca para desenvolvimento de aplicações Deep Learning capaz de rodar sobre o TensorFlow ou o Theano.
 - **skorch**: uma biblioteca de rede neural compatível com o scikit-learn que encapsula a biblioteca PyTorch.
 - 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.
- **Exercício Prático:** [Projeto #2](#).
 - Pode ser baixado do MS Teams ou do GitHub.
 - Pode ser respondido através do link acima (na nuvem) ou localmente.
 - [Instruções para resolução e entrega dos laboratórios](#).
 - **Atividades podem ser feitas em grupo, mas as entregas devem ser individuais.**

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..



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

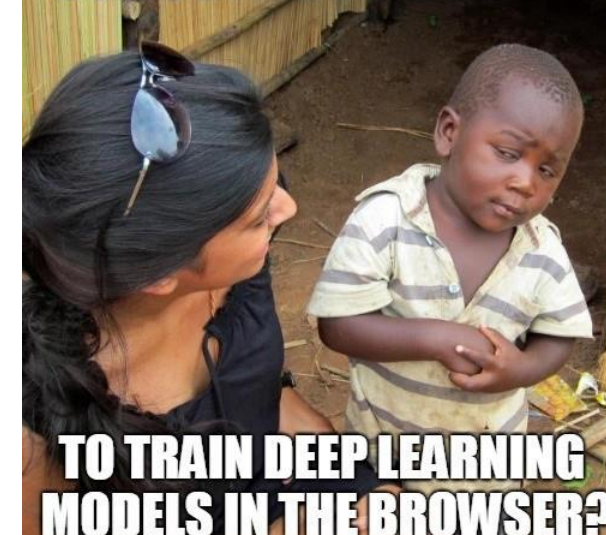


What I think I do

```
In [1]:  
import keras  
Using TensorFlow backend.
```

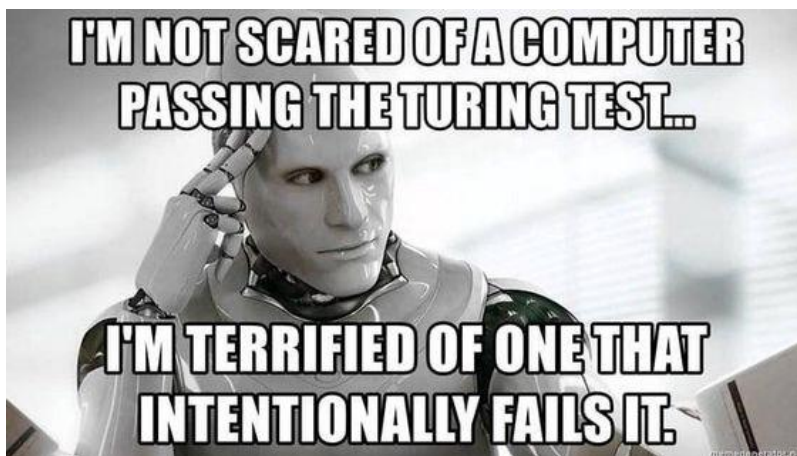
What I actually do

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**

Figuras

