GEX1082 - Tópicos Especiais em Computação XXXIII
Deep Learning

Treinamento

1100/1101 - CIÊNCIA DA COMPUTAÇÃO

Prof. Dr. Giancarlo D. Salton

Forward Pass

Backpropagation

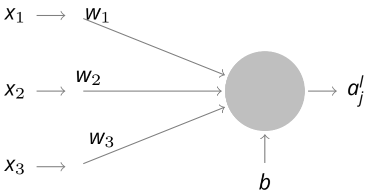
Gradient Descent

Backprop

Forward Pass

- Uma rede neural é um modelo de *machine learning* da família dos métodos baseados em erro.
- Dado uma *dataset* $\mathcal{D}$ com N *datapoints*: $(\mathbf{x}_n, y_n) \in (\mathcal{X}, \mathcal{Y})$
  - ✓ onde $\mathbf{x}_n$ são os *inputs* e y_n é o *target* relativos a um *datapoint* n
- O algoritmo itera sobre o *dataset* e "aprende" uma função parametrizada $f : \mathcal{X} \rightarrow \mathcal{Y}$
  - ✓ esta função descreve a relação entre as *features* e o *target*
  - ✓ parâmetros ("pesos") e controlam a saída retornada pela função
- Para guiar o aprendizado f, o algoritmo usa função de perda/custo $\mathcal{C}(y_n, f(\mathbf{x}_n))$, que interpretamos como o "erro" que a rede neural comete
  - ✓ *cost/loss function*
  - ✓ Aprendizado do modelo também é chamado de *otimização* da função de custo

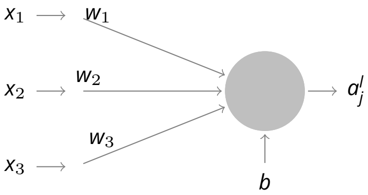
Neurônios



$$z = b + w_1 x_1 + w_2 x_2 + w_3 x_3 = \mathbf{W}^T \mathbf{x} + \mathbf{b}$$

$$a = \frac{1}{1 + e^{-z}} \tag{1}$$

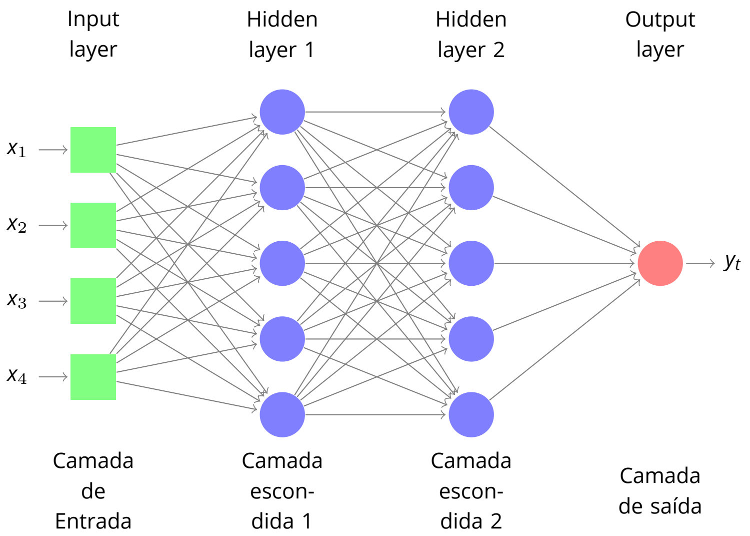
Neurônios



Função sigmoide: $\sigma(\theta)$

$$z = b + w_1 x_1 + w_2 x_2 + w_3 x_3 = \mathbf{W}^T \mathbf{x} + \mathbf{b}$$

$$a = \frac{1}{1 + e^{-z}} \tag{1}$$


Input layer
Hidden layer 1
Hidden layer 2
Output layer
x_1
x_2
x_3
x_4
y_t
Camada de Entrada
Camada escon-dida 1
Camada escon-dida 2
Camada de saída

Forward pass

Exemplo com a rede do slide anterior

$$h^{(1)} = a^1 = g\left(b^{(1)} + W^{T(1)}x\right)$$
$$h^{(2)} = a^2 = g\left(b^{(2)} + W^{T(2)}h^{(1)}\right)$$
$$o = a^3 = g\left(b^{(3)} + W^{T(3)}h^{(2)}\right)$$

onde

- ✓ $g = sigmoide = \frac{1}{1+e^{-z}}$

Backpropagation

- O algoritmo de *backpropagation* foi uma revolução na área de redes neurais
  - ✓ Foi proposto na década de 1970 mas somente em 1986 é que foi demonstrada a sua utilidade

- Possibilitou o treinamento de redes neurais maiores e, especialmente, que a rede neural aprendesse novas *features*!
  - ✓ Como veremos numa das próximas aulas, essa é a base do *Deep learning*

- O conceito do *backpropagation* é bastante simples:

  1. Calculamos o erro que a rede neural comete após um *forward pass* utilizando uma função de custo
  2. Calculamos o quanto cada neurônio é responsável por esse erro
  3. Ajustamos os parâmetros de cada neurônio baseado no erro cometido por ele
  4. Repetimos o processo até o que minimizemos o erro cometido pela rede neural
     - Isto é, até que o erro não diminua mais

Loss Function

Original: *Quadratic cost*

$$\mathbf{C} = \frac{1}{n}\sum_{x}\|y - a^L\|^2$$

Loss Function

a^L é a ativação da cama de saída.

Original: ***Quadratic cost***

$$\mathbf{C} = \frac{1}{n} \sum_x \|y - a^L\|^2$$

Loss Function

Padrão (2 *targets*): *Cross-entropy*

$$\mathbf{C} = -\frac{1}{n}\sum_{x}[y \log a^{L} + (1-y)\log(1-a^{L})]$$

Loss Function

a^L é, novamente a ativação da camada de saída e representa a probabilidade $P(\hat{y})$ predita para o *target*

Padrão (2 *targets*): *Cross-entropy*

$$\mathbf{C} = -\frac{1}{n}\sum_x [y \log a^L + (1-y)\log(1-a^L)]$$

Loss Function

Padrão (2 *targets*): *Cross-entropy*

$$\mathbf{C} = -\frac{1}{n}\sum_x [y \log a^L + (1-y)\log(1-a^L)]$$
$$= -\frac{1}{n}\sum_x [y \log P(\hat{y}) + (1-y)\log(1-P(\hat{y}))]$$

Loss Function

Padrão (3+ *targets*): *Log Loss*

$$\mathbf{C} = -\frac{1}{n}\sum_{i=1}^{N} log(P(y))$$

Loss Function

Log loss é equivalente a *Cross-entropy*!

Padrão (3+ *targets*): *Log Loss*

$$\mathbf{C} = -\frac{1}{n}\sum_{i=1}^{N} log(P(y))$$

- A função de custo precisa obedecer a 2 regras:
  1. precisa ser escrita como uma média das funções de custo para cada *datapoint* no *dataset de treino*

$$\mathbf{C} = \frac{1}{n}\sum_{x} \mathcal{C}_x$$

  2. precisa ser uma função aplicável sobre a ativação dos neurônios da camada de saída

$$\mathbf{C} = C(a^L)$$

onde L indica a camada de saída da rede neural

Gradiente Descendente

- Supondo que tenhamos a função de custo $\mathcal{C}$ e os parâmetros $w_1, w_2, \ldots, w_n$ de uma função contínua qualquer para otimizar
- Definimos o vetor de gradientes como sendo

$$\nabla \mathcal{C} = \left(\frac{\partial \mathcal{C}}{\partial w_1}, \frac{\partial \mathcal{C}}{\partial w_2}, \cdots, \frac{\partial \mathcal{C}}{\partial w_n}, \right)^T$$

Gradiente Descendente

Regra de atualização em notação de matrizes/vetores

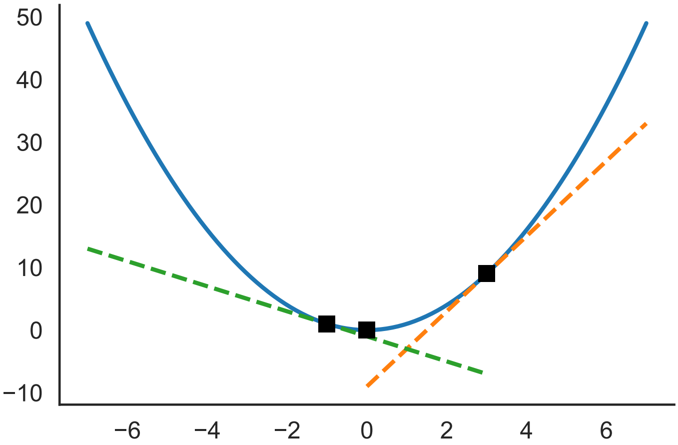
$$\hat{\mathbf{w}} = \mathbf{w} - \alpha \odot \nabla \mathcal{C}$$

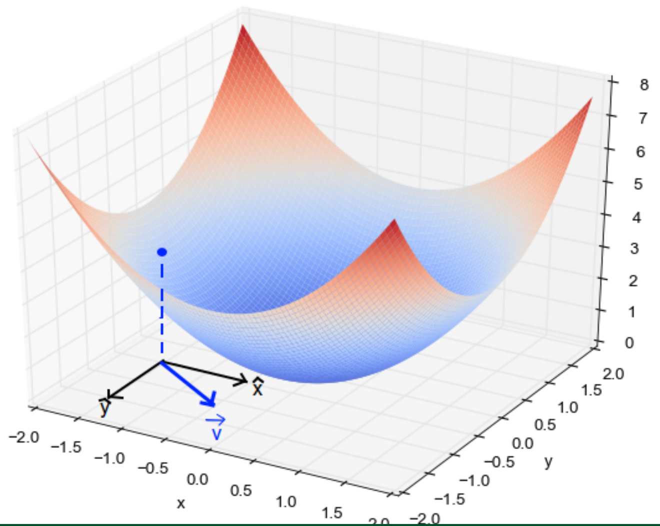
Gradiente Descendente

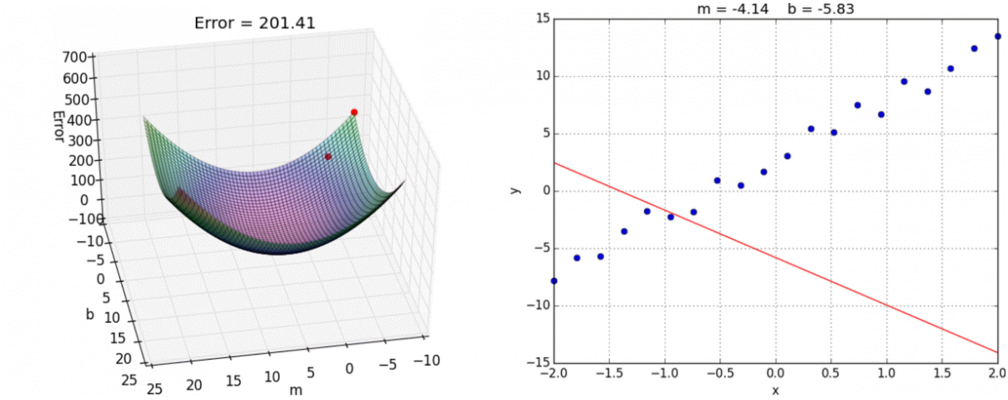
Learning rate, controla o quanto do erro iremos utilizar para atualizar cada parâmetro.

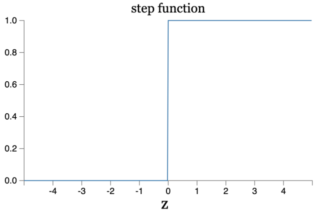
Regra de atualização em notação de matrizes/vetores

$$\hat{\mathbf{w}} = \mathbf{w} - \alpha \odot \nabla \mathcal{C}$$

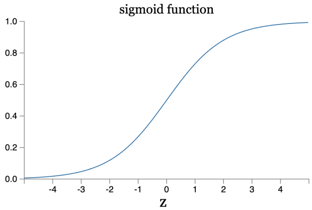

50
40
30
20
10
0
−10
−6
−4
−2
0
2
4
6


8
7
6
5
4
3
2
1
0
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
y
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
x
$\hat{x}$
$\hat{y}$
$\vec{v}$


Error = 201.41
700
600
500
400
300
200
100
0
−100
Error
−10
−5
0
5
b
10
15
20
25
25
20
15
10
5
0
−5
−10
m
m = -4.14   b = -5.83
15
10
5
0
−5
−10
−15
y
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x



(a) Step Function



(b) Sigmoid Function

Perceba como a *Step Function* não possui derivadas nos extremos onde a saída é 0 ou 1 enquanto a *sigmoid* possui derivadas mesmo que sejam muito pequenas (próximas a 0).

- Equação para o erro na camada de saída (para ativação σ)

$$\delta_j^L = \frac{\partial}{\partial a_j^L} \sigma'(z_j^L)$$

- em forma de notação de matrizes:

$$\delta^L = \nabla_a \mathcal{C} \odot \sigma'(\mathbf{z}^l) \tag{BP1}$$

- Equação para o erro em uma camada (δ^l) em termos do erro na camada seguinte (δ^{l+1})

$$\delta^l = ((\mathbf{W}^{l+1})^T \delta^{l+1}) \odot \sigma\prime(z^l) \tag{BP2}$$

- Equação para a alteração no erro com respeito a qualquer bias (*b*) na rede neural

$$\frac{\partial C}{\partial b} = \delta^l \tag{BP3}$$

- Equação para cálculo do erro com respeito a qualquer peso (w) na rede

$$\frac{\partial C}{\partial w} = a^l \delta^{l-1} \tag{BP4}$$

- Resumo das equações:

$$\delta^L = \nabla_a \mathcal{C} \odot \sigma'(\mathbf{z}^l) \tag{BP1}$$

$$\delta^l = ((\mathbf{W}^{l+1})^T \delta^{l+1}) \odot \sigma\prime(z^l) \tag{BP2}$$

$$\frac{\partial C}{\partial b} = \delta^l \tag{BP3}$$

$$\frac{\partial C}{\partial w} = a^l \delta^{l-1} \tag{BP4}$$

- Embora tenha sido usada a função de ativação σ nos exemplos, o processo para outras funções de ativação é o mesmo

- Basta substituir a derivada σ' pela derivada correspondente à função de ativação em uso

$$\sigma(z) = \frac{1}{1 + e^{-z}} \qquad \sigma'(z) = \sigma(z) * (1 - \sigma(z))$$

$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \qquad tanh'(z) = 1 - tanh(z)^2$$

$$ReLu(z) = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{if } z \geqslant 0 \end{cases} \qquad ReLu'(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geqslant 0 \end{cases}$$