

Aula 14

Introdução às Redes Neurais e o Perceptron

Rafael Geraldelli Rossi

Introdução

- Cérebro humano possui características que não estão presentes em computadores:
 - Paralelismo massivo
 - Representação e computação distribuídas
 - Habilidade de aprendizado
 - Habilidade de generalização
 - Adaptabilidade
 - Tolerância a falhas
 - Baixo consumo de energia

Introdução

- A ideia de uma **rede neural é “simular” parte das características do cérebro humano** que podem ser modeladas computacionalmente (paralelismo, habilidade de aprendizado e representação, adaptabilidade) para realizar um aprendizado utilizando computador
- **Modelos neurais, também conhecidos como processamento paralelo distribuído** (PDP, do inglês *Parallel Distributed Processing*), **ou sistemas conexionistas**, consideram que a inteligência surge em um sistema de componentes simples (neurônios biológicos ou artificiais), interativos, por meio de um processo de aprendizado, ou adaptação, pelo qual as conexões entre os componentes são ajustadas

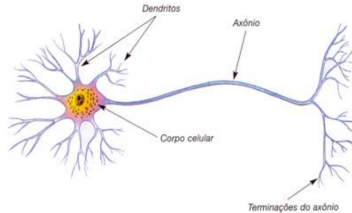
Introdução

- O processamento nesse sistema é distribuído por meio de camadas de neurônios
- A solução é paralela no sentido que todos os neurônios dentro do conjunto ou camada processam as suas entradas simultaneamente e independentemente
- Tarefas para as quais a abordagem neural/conexionista é bem adequada:
 - **Classificação:** decide a categoria ou grupo ao qual pertence um valor de entrada
 - **Reconhecimento de Padrões:** identifica estrutura ou padrões nos dados
 - **Evocação de Memória:** problema da memória endereçável por conteúdo
 - **Regressão:** permite prever valores reais
 - **Otimização:** encontra a melhor conjunto de valores dadas restrições
 - **Filtragem de Ruído:** retira os componentes irrelevantes de um sinal (como ruído de fundo)

Introdução

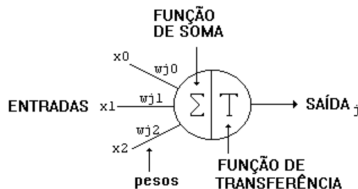
- Existem vários tipos de famílias de redes neurais para as tarefas mencionadas acima
 - Redes recorrentes
 - Mapas auto-organizáveis
 - Perceptrons em uma única camada
 - Perceptrons de múltiplas camadas
 - ...
- Nesta aula veremos famílias de redes que podem ser utilizadas para a maioria das tarefas mencionadas anteriormente:
Perceptrons em uma Única Camada e Perceptrons em Múltiplas Camadas

Modelo de Neurônio Humano Clássico



- Um neurônio possui muitos **dendritos** ramificados que **recebem informação de outros neurônios**
- O **corpo celular** “**processa**” a **informação recebida**
- O neurônio possui um único **axônio** que **propaga a informação processada** a partir de suas terminações
- As sinapses são estímulos que passam de um neurônio para outro por meio de neurotransmissores

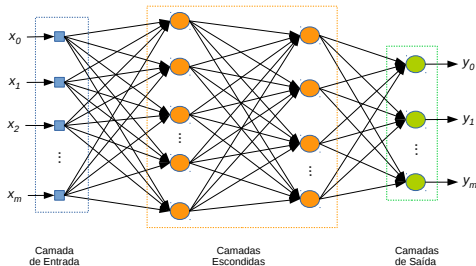
Modelo de Neurônio Artificial Clássico



- Um neurônio artificial é composto por um conjunto de entradas com pesos (representando os dendritos)
- As entradas recebem informações do ambiente ou de outros neurônios que são processadas por uma função soma
- A função de transferência verifica se o estímulo recebido nas entradas será transmitido (função de ativação a qual geralmente utiliza um limiar)

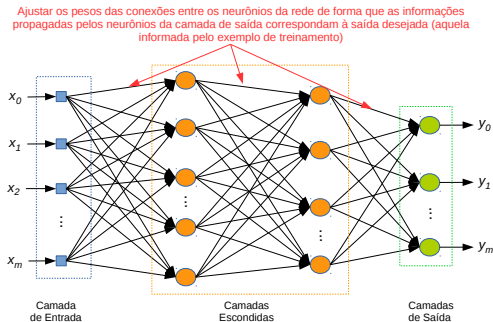
Rede Neural Artificial

- Uma Rede Neural Artificial (RNA) é composta por um conjunto de neurônios interconectados
- Os neurônios podem ser de diferentes tipos dependendo de sua localização nas camadas da rede
 - **Entrada:** recebem um sinal do ambiente
 - **Intermediário:** processam informações entre neurônios
 - **Saída:** devolvem sinal ao ambiente



Rede Neural Artificial

- Treinamento de uma rede neural
 - Identificar os melhores “pesos” das conexões entre neurônios da rede
 - Os pesos definem como são as respostas aos estímulos do ambiente



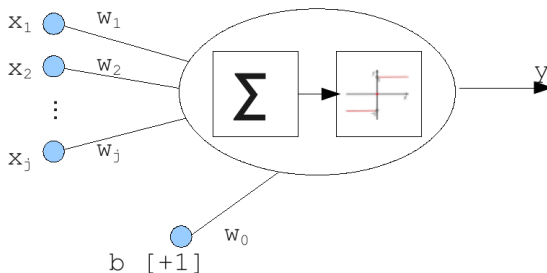
Rede Neural Artificial

- **Existem muitas configurações/topologias de redes neurais**
 - Número de neurônios
 - Conexões entre neurônios
 - Camadas de neurônios
- **Identificar a melhor topologia é um problema experimental**
- **Número de neurônios na cada de saída para problemas de classificação**
 - Classificação binária: um neurônio (saída +1 para a classe positiva e 0 ou -1 para a classe negativa)
 - Classificação multi-classe ou multi-rótulo: um neurônio para a cada classe e dado um exemplo e uma classe c_j , o neurônio y_j deve responder +1 e os demais neurônios devem responder 0 ou -1

Perceptron

- **Primeiro “modelo de neurônio” para aprendizado supervisionado**
- Consistem em:
 - **Sinais de entrada:** corresponde aos valores dos atributos (x_1, x_2, \dots, x_j) para um conjunto de dados com n atributos)
 - **Pesos sinápticos:** pesos que serão inferidos para cada entrada do neurônio (w_1, w_2, \dots, w_j)
 - **Bias (b):** permite deslocar a função induzida pelo plano para melhor se ajustar aos dados de treinamento
 - **Função soma (\sum):** corresponde a uma função linear pesada simples
 - **Função de ativação:** função que irá determinar se o neurônio deve ou propagar um sinal (*spike*) e qual a força do sinal
 - **Sinal de saída:** sinal propagado pela função de ativação (y)

Perceptron



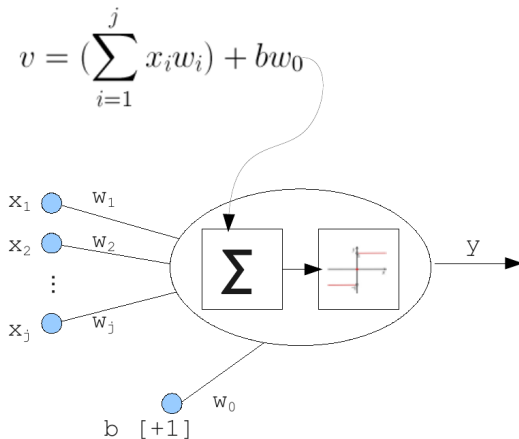
Perceptron

● **Aprendizado Supervisionado no Perceptron:**

- Novamente, o aprendizado consiste em computar os pesos (w_1, w_2, \dots, w_j)
- Cada sinal de entrada é processado, gerando uma determinada saída (y)
- A saída y é comparada com a saída desejada (classe do exemplo)
- Os pesos do neurônio são ajustados na direção da saída desejada, isto é, de forma que minimizem a diferença entre a saída da rede e a saída desejada
- O processo é repetido iterativamente para considerando todos os exemplos de treinamento, várias vezes (épocas) até que a saída y seja próxima a saída desejada

Função Soma

- Calcula a soma ponderada das entradas de uma rede neural



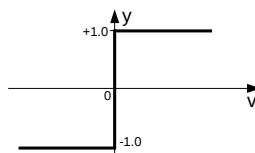
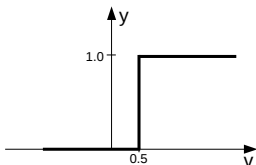
Função de Ativação

- Fornece o valor de saída do neurônio
- Uma das funções mais comuns é a função degrau:

$$y = \begin{cases} 1, & \text{se } v > 0.5 \\ 0 & \text{caso contrário} \end{cases} \quad (1)$$

- Uma variação também muito utilizada da função degrau é dada por:

$$y = \begin{cases} +1, & \text{se } v > 0 \\ -1 & \text{caso contrário} \end{cases} \quad (2)$$



Treinamento do Perceptron

- Inicialização
 - Todos os pesos = 0
 - Pesos definidos aleatoriamente
- Para cada padrão \mathbf{x} apresentado (estímulo) à rede
 - Calcular a função e o sinal de ativação $y(\mathbf{x})$
 - Se a saída for diferente da saída desejada \rightarrow então atualizar pesos
- A atualização dos pesos é feita na direção oposta do gradiente (gradiente descendente) conforme apresentado na [Aula 5](#)
- Critério de parada: até atingir um número máximo de épocas ou até que o erro quadrático método mínimo em uma época esteja abaixo de um limiar

Treinamento do Perceptron

- Atualização do peso w_i é dada por

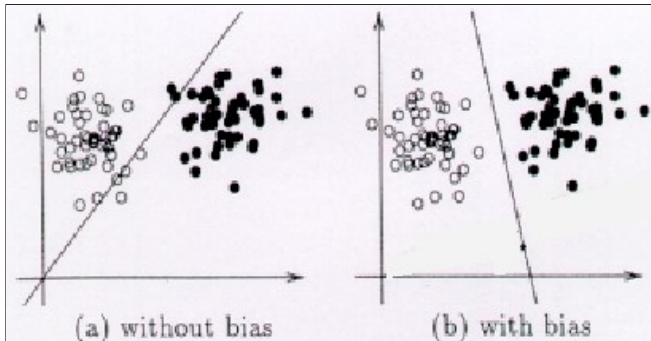
$$w_i^{t+1} = w_i^t + \eta(d(\mathbf{x}) - y(\mathbf{x})) \cdot x_i$$

na qual

- w_i^{t+1} é o novo peso ou o peso na próxima iteração ($t + 1$)
- w_i^t é o peso atual
- $d(\mathbf{x})$ é a classe do exemplo \mathbf{x} (saída desejada)
- $y(\mathbf{x})$ é a saída gerada pelo perceptron para o exemplo \mathbf{x}
- $(d(\mathbf{x}) - y(\mathbf{x}))$ é o erro de classificação do exemplo \mathbf{x}
- x_i é o valor da i -ésima entrada do neurônio (valor do i -ésimo atributo de \mathbf{x})
- η é a taxa de aprendizado ($0 < \eta \leq 1$)

Treinamento do Perceptron

- O resultado do treinamento de um perceptron é um hiperplano de separação linear capaz de separar as classes



Treinando um Perceptron para simular o operador lógico OR

Tabela: Conjunto de treinamento

Estímulo	x_1	x_2	Classe
Entrada 1	0	0	0
Entrada 2	0	1	1
Entrada 3	1	0	1
Entrada 4	1	1	1

- Requisitos:

- Inicializar todos os pesos com 0
- A função de ativação deve ser a degrau

$$y = \begin{cases} 1, & \text{se } v > 0.5 \\ 0 & \text{caso contrário} \end{cases}$$

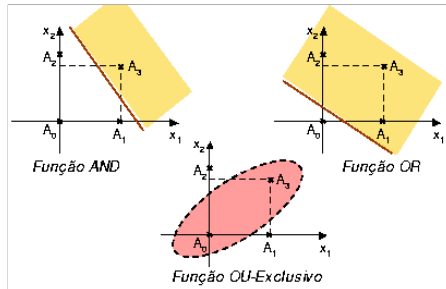
- Taxa de aprendizado $\eta = 0.5$

Classificação de Novos Exemplos

Tanto no perceptron, quanto nas redes neurais que vemos mais a frente, para classificar um novo exemplo basta submeter os valores dos seus atributos na rede e obter a saída da rede (resultado da classificação)

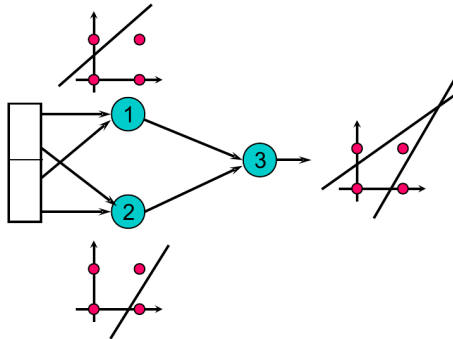
Limitações do Perceptron

- Depois do grande alvoroço do surgimento do Perceptron, descobriu-se que esse é limitado → apenas aprendizado de problemas linearmente separáveis
- Por exemplo, uma perceptron não é capaz de aprender uma função XOR



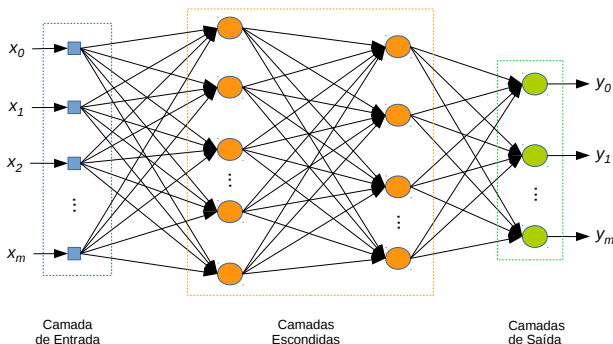
Limitações do Perceptron

- Para solucionar o problema da separação linear dos dados pode-se utilizar
 - Conjunto de Perceptrons
 - Redes Multicamadas (*MLP - Multilayer Perceptron*)



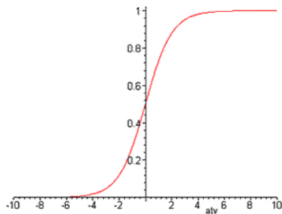
Multilayer Perceptron

- Redes Neurais mais utilizadas em aprendizado supervisionado
- Possuem uma ou mais camadas intermediárias
- Topologia semelhante a apresentada no início dos slides



Multilayer Perceptron

- Nesse tipo de rede, a função de ativação mais popular é a Sigmóide Logística

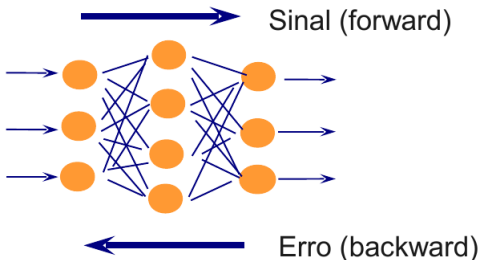


$$y = \frac{1}{1 + \exp(-av)}$$

na qual a determina a inclinação da função sigmoide (conforme se aproxima de 1, a função sigmoide se aproxima de uma reta)

Multilayer Perceptron

- O treinamento mais comum para uma rede MLP é feita por meio do algoritmo Backpropagation
- Duas fases?
 - Fase *forward* → padrão é apresentado e o estímulo percorre a rede (semelhante ao perceptron)
 - Fase *backward* → a partir do erro encontrado os neurônios são atualizados



Multilayer Perceptron

- Para atualizar os pesos das conexões que estão ligadas aos neurônios da camada de saída é fácil: mesma regra do perceptron, pois se sabe qual é o erro produzido pela saída (nó)
- Entretanto, conexões que estão se conectando aos nós das camadas intermediárias, não se sabe qual é o erro em cada nó
- Para isso, é necessário estimar o erro de um nó para atualizar as conexões que se conectam a ele
- **OBSERVAÇÃO:** as derivadas para se atualizar os pesos são apresentadas em e as regras de atualização dos pesos das conexões de acordo com as camadas são apresentadas em [Haykin, 1998]

Multilayer Perceptron

● Desvantagens

- Convergência pode ser muito lenta
- Computacionalmente caro
- Definição dos parâmetros é experimental
 - Número de camadas
 - Número de neurônios por camadas
 - Taxa de aprendizado

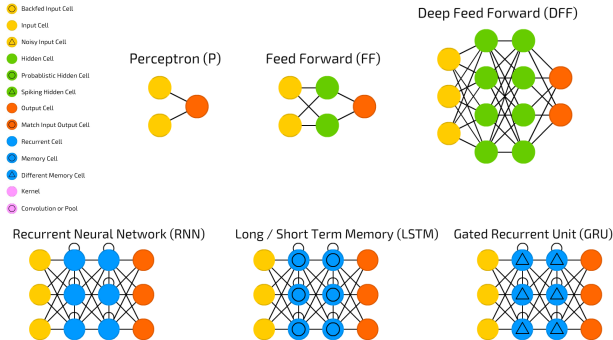
● Vantagens

- Redes MLP com uma única camada oculta permitem a aproximação de qualquer função contínua
- Amplamente aceita na indústria e na academia

Multilayer Perceptron

Simulador de Redes Neurais da Google:
<https://playground.tensorflow.org>

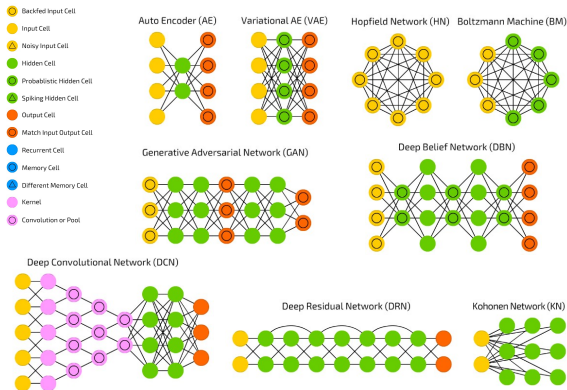
Arquiteturas



[https:](https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464)

[//towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464](https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464)

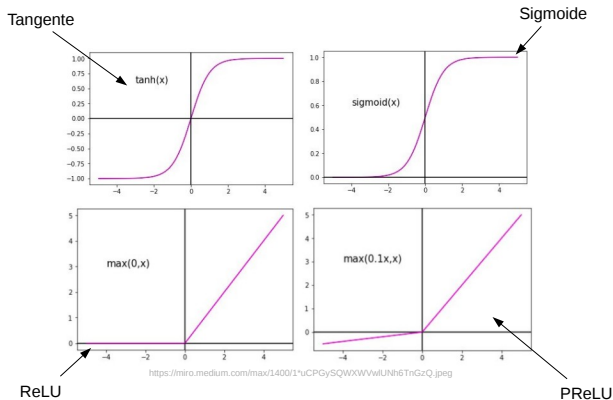
Arquiteturas












[https:](https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464)

[//towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464](https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464)

Funções de Ativação

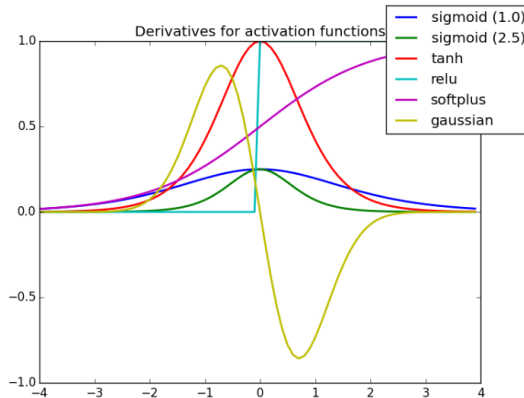


Funções de Ativação

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a. Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tanh		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
Arctan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parametric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

https://miro.medium.com/max/1400/1*p_hyqAtyI8pbt2kEl6si0Q.png

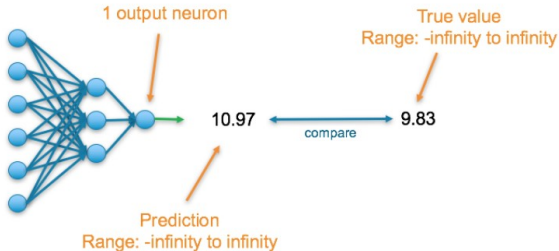
Funções de Ativação



https://miro.medium.com/max/1400/1*n1HFBpwv21FCAzGjmWt1sg.png

Erro Quadrático Médio

Mean squared error (MSE)

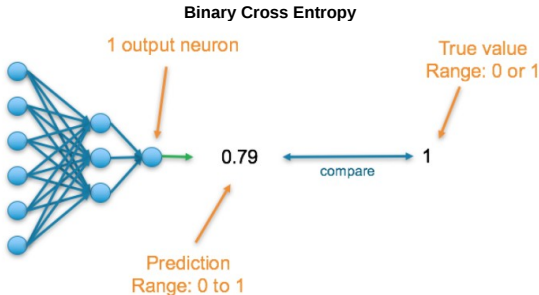


Encontra a diferença média quadrática entre o valor predito e o valor real.

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

Where \hat{y} is the predicted value and y is the true value

Entropia Binária Cruzada

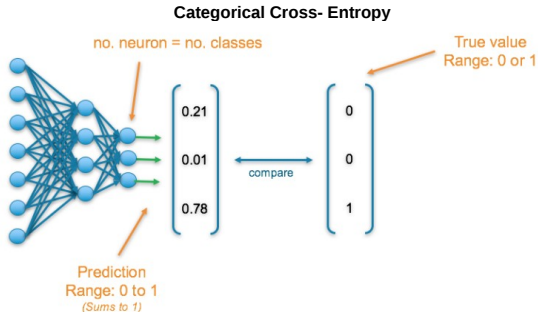


- Compara a distribuição de probabilidades (p e $1-p$)
- Deve ser aplicada em conjunto com a função de ativação Sigmoid ou Tangente.

$$\text{Binary cross entropy} = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$$

Where \hat{y} is the predicted value and y is the true value

Entropia Categórica Cruzada



- Normalmente empregada com uma normalização na saída (softmax) para padronização dos valores.
- Quantifica a diferença entre duas distribuições de probabilidades.

$$\text{Cross entropy} = - \sum_i^M y_i \log(\hat{y}_i)$$

Where \hat{y} is the predicted value, y is the true value and M is the number of classes

Otimizadores

Stochastic gradient descent

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta; x^{(i)}; y^{(i)}).$$

Mini-batch gradient descent

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta; x^{(i:i+n)}; y^{(i:i+n)}).$$

Batch gradient descent

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta).$$

Adagrad

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t.$$

RMSprop

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

Adam

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t.$$
$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

■ ■ ■

Otimizadores

Vejam as animações:

- https://ruder.io/content/images/2016/09/contours_evaluation_optimizers.gif
- https://ruder.io/content/images/2016/09/saddle_point_evaluation_optimizers.gif

Material Complementar

- Rede neural artificial

https://pt.wikipedia.org/wiki/Rede_neural_artificial

- 12a: Neural Nets

<https://www.youtube.com/watch?v=uXt8qF2Zzfo>

- Redes Neurais Artificiais

<http://conteudo.icmc.usp.br/pessoas/andre/research/neural/>

Material Complementar

- A Comprehensive Guide on Activation Functions

<https://towardsdatascience.com/>

[a-comprehensive-guide-on-activation-functions-b45ed37a4fa5](https://towardsdatascience.com/a-comprehensive-guide-on-activation-functions-b45ed37a4fa5)

- Deep Learning: Which Loss and Activation Functions should I use?

<https://towardsdatascience.com/>

[deep-learning-which-loss-and-activation-functions-should-i-use-ac02f1c56a](https://towardsdatascience.com/deep-learning-which-loss-and-activation-functions-should-i-use-ac02f1c56a)

- The mostly complete chart of Neural Networks, explained

<https://towardsdatascience.com/>

[the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464](https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464)

Imagem do Dia



Inteligência Artificial
<http://lives.ufms.br/moodle/>

Rafael Geraldeli Rossi
rafael.g.rossi@ufms.br

Slides baseados no material do Prof. Ricardo Marcacini e nos livros
[Haykin, 1998] e [Luger, 2016]

Referências Bibliográficas I



Haykin, S. (1998).

Neural Networks: A Comprehensive Foundation.

Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition.



Luger, G. F. (2016).

Neural Networks: A Comprehensive Foundation.

Pearson, 6ª edição.