Inteligência Artificial para Robótica Móvel

Aprendizado de Máquina

Professor: Marcos Maximo

Roteiro

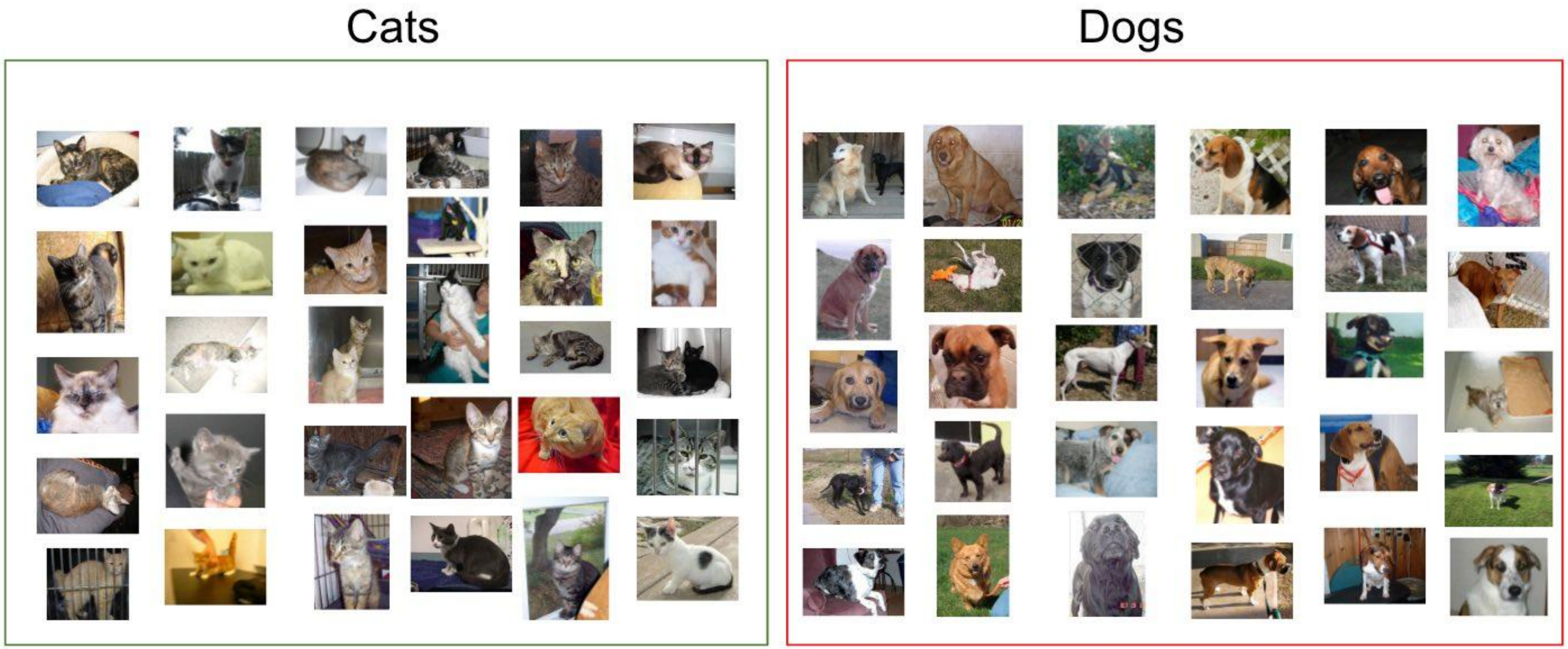
- Motivação;
- Neurônio Artificial;
- Redes Neurais;
- Dicas para Redes Neurais.

Motivação

Motivação

- Inglês: *Machine Learning* (ML).
- Área de IA mais ativa atualmente.
- Aprendizado Supervisionado: mostrando exemplos (professor).
- Aprendizado Não-supervisionado: encontrar padrões em dados.
- Aprendizado por Reforço: através de experiências (recompensas).
- Desempenho super-humano em tarefas complexas:
  - Visão (em dados de competições específicas).
  - Jogos de Atari.
  - Dota.
  - Starcraft.

Aprendizado Supervisionado



Sample of cats & dogs images from Kaggle Dataset

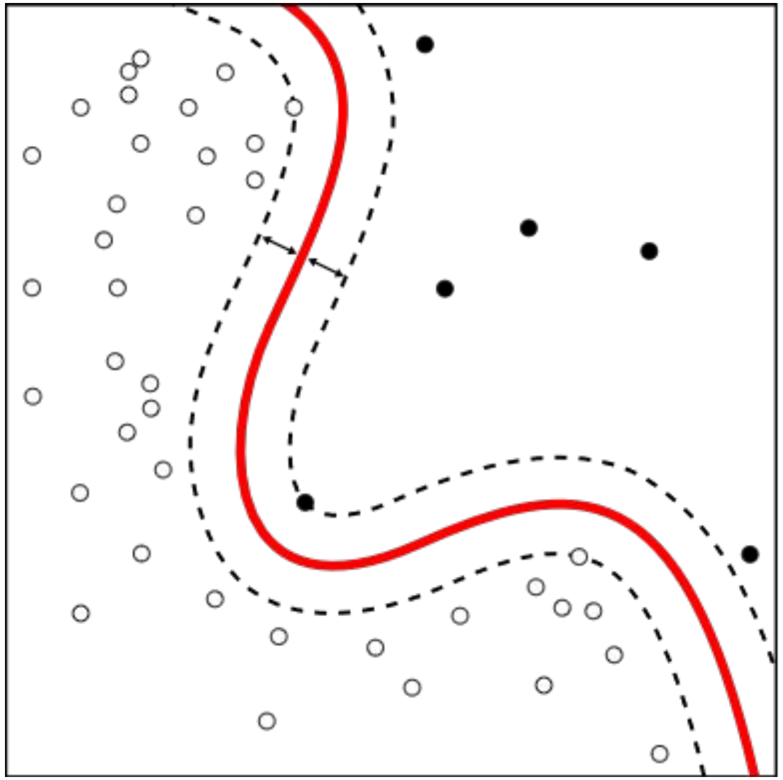
Fonte: http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/

Aprendizado Supervisionado

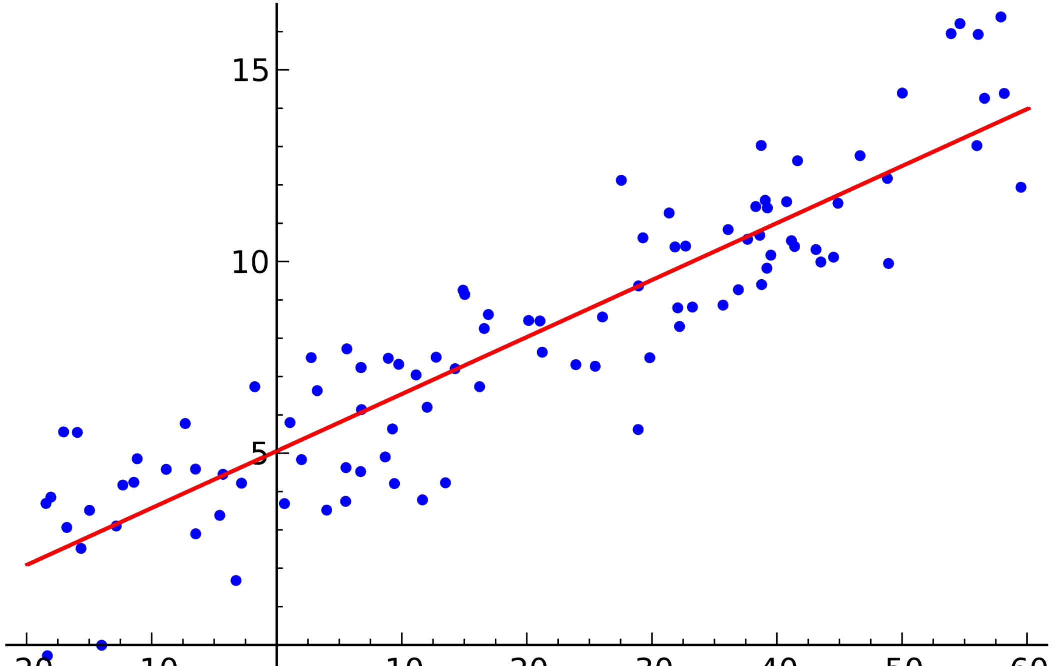
- Ideia matemática: aproximar uma função.
- Função é descrita pelos exemplos de treinamento.
- Algoritmos são estruturas com parâmetros a serem ajustados.
- Usa-se otimização para ajustar os parâmetros (treinamento).
- Espécie de “ajuste de curvas” avançada.
- Funciona muito bem para problemas difíceis de serem descritos formalmente.
- “Algoritmo descrito com dados”.

Aprendizado Supervisionado

Classificação



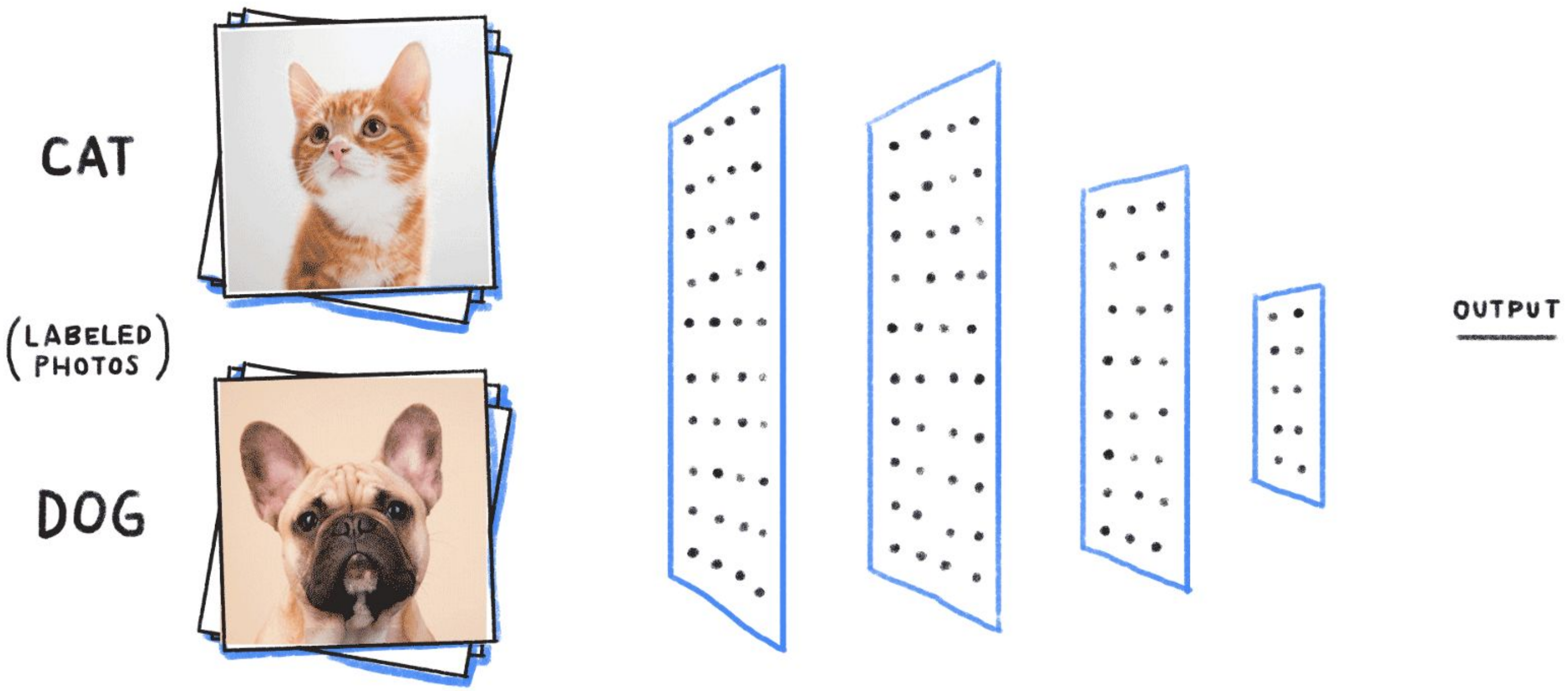
Predição



Aprendizado Supervisionado

- Há um grande conjunto de técnicas de Aprendizado Supervisionado.
  - Support Vector Machines (SVM).
  - Árvores de decisão.
  - Random Florest.
  - Redes neurais.
- Focaremos em redes neurais.
- Motivo: redes neurais são responsáveis pela revolução de Deep Learning.

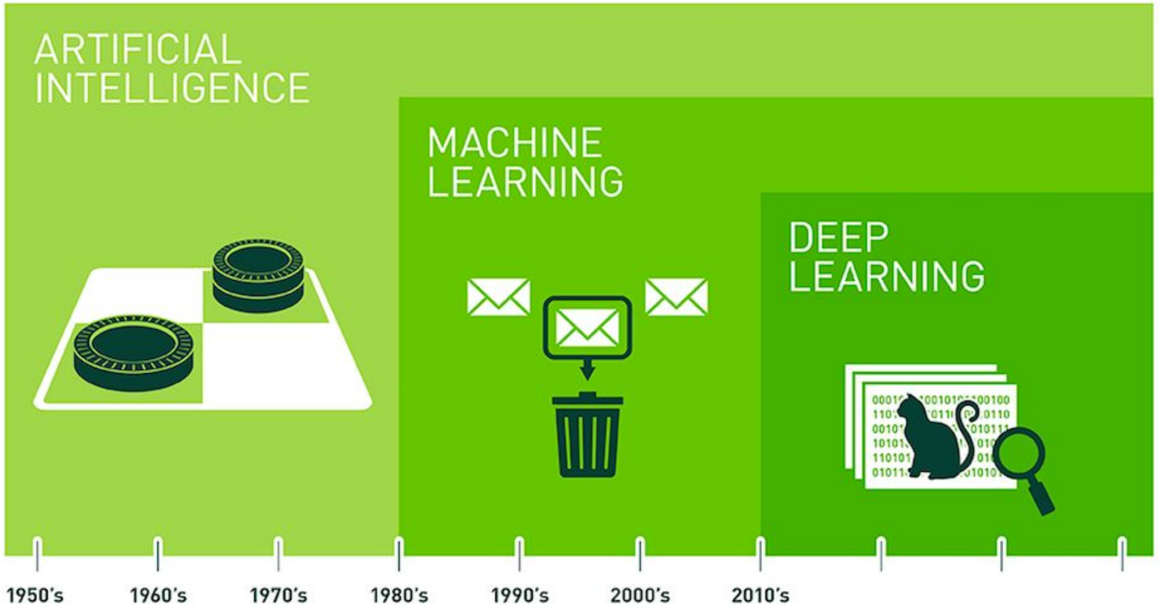
Aprendizado Supervisionado



Fonte:
https://becominghuman.ai/building-an-image-classifier-using-deep-learning-in-python-totally-from-a-beginners-perspective-be8dbaf22dd8

Deep Learning

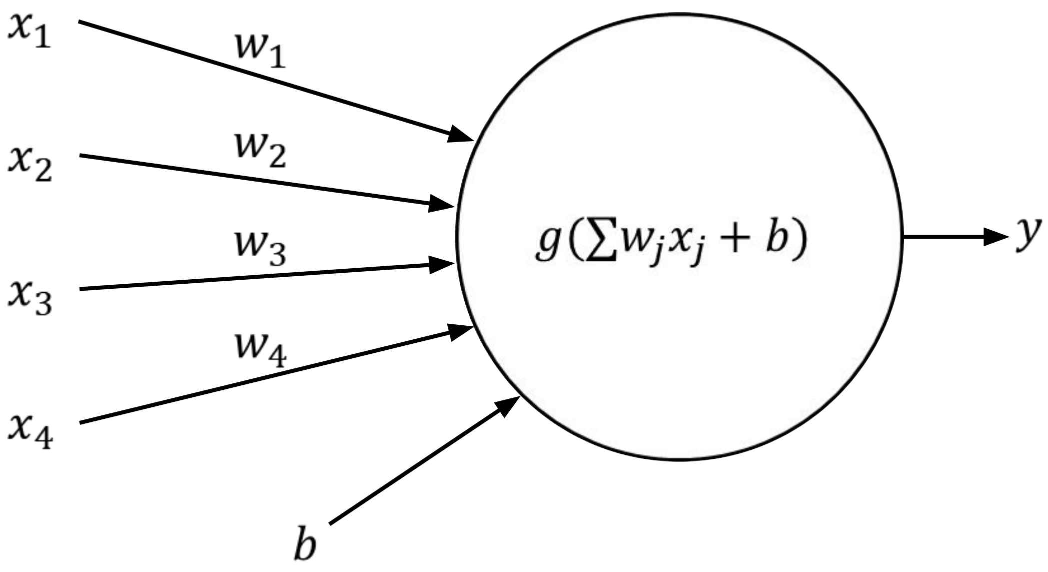
- Conjunto de técnicas que permitiram treinar redes neurais profundas.
- Virou *buzzword*.



Fonte: https://medium.com/data-science-brigade/a-diferen%C3%A7a-entre-intelig%C3%AAncia-artificial-machine-learning-e-deep-learning-930b5cc2aa42

Neurônio Artificial

Neurônio Artificial



- Conta realizada pelo neurônio:

$$y = g\left(\sum w_j x_j + b\right)$$
$$= g(\mathbf{w}^T \mathbf{x} + b) = g(z)$$

$\mathbf{w}$: pesos.

b: bias.

$\mathbf{x}$: entradas (features).

g: função de ativação.

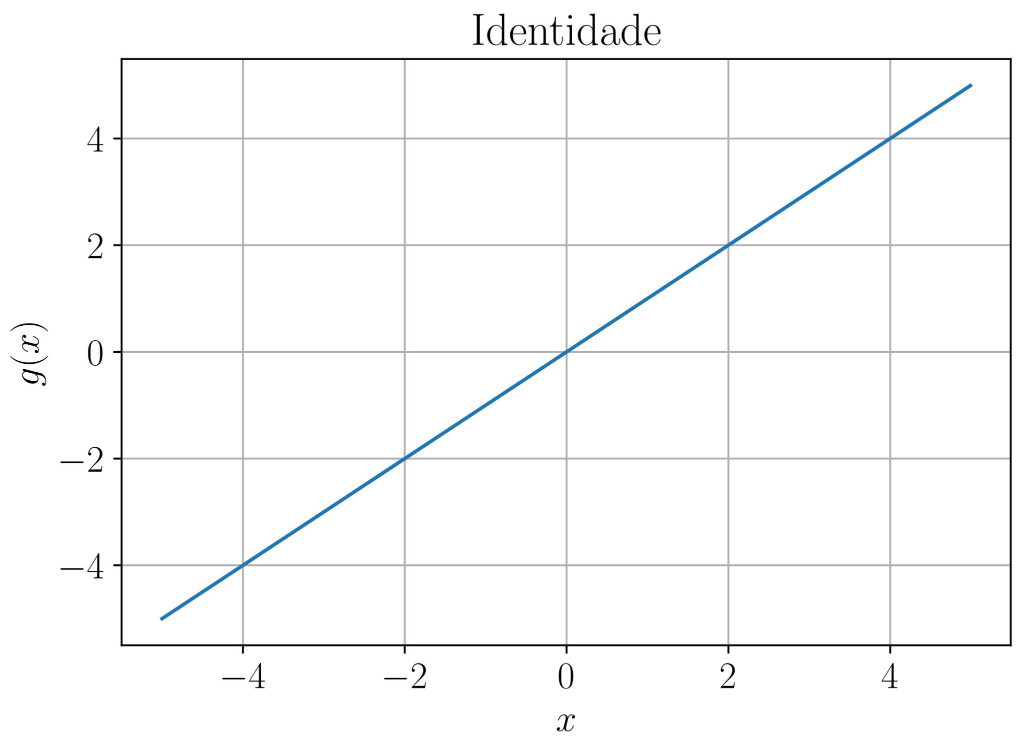
- Objetivo do treinamento é ajustar $\mathbf{w}$ e b para aproximar alguma função $\hat{y}(\mathbf{x})$.

Funções de Ativação

- Identidade:

$$g(x) = x$$

(pré-Deep Learning)

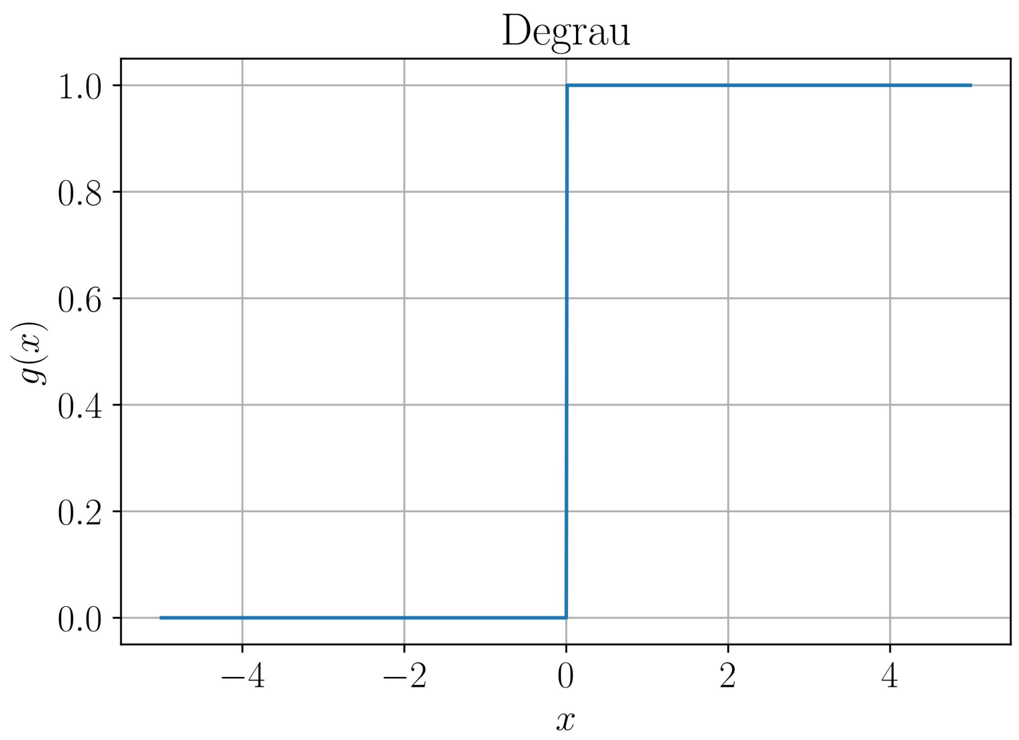


Funções de Ativação

- Degrau:

$$g(x) = \begin{cases} 0, x < 0 \\ 1, x \geq 0 \end{cases}$$

(pré-Deep Learning)



Funções de Ativação

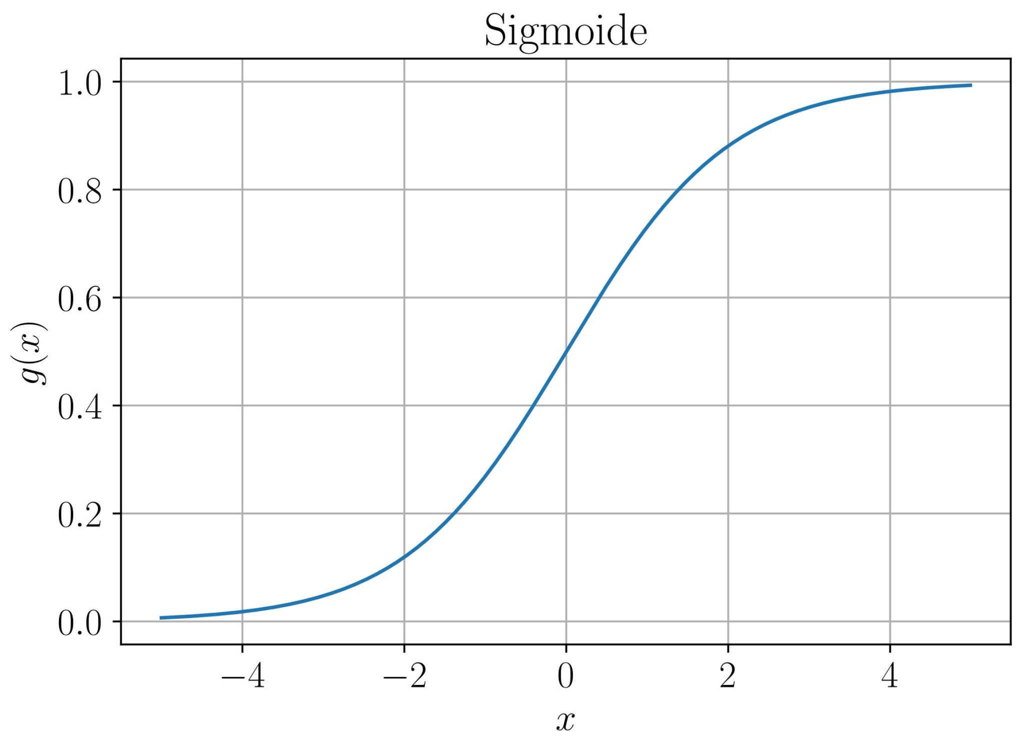
- Sigmóide:

$$\sigma(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1}$$

(pré-Deep Learning)

- Derivada da sigmóide:

$$\sigma'(x) = \sigma(x)\big(1 - \sigma(x)\big)$$

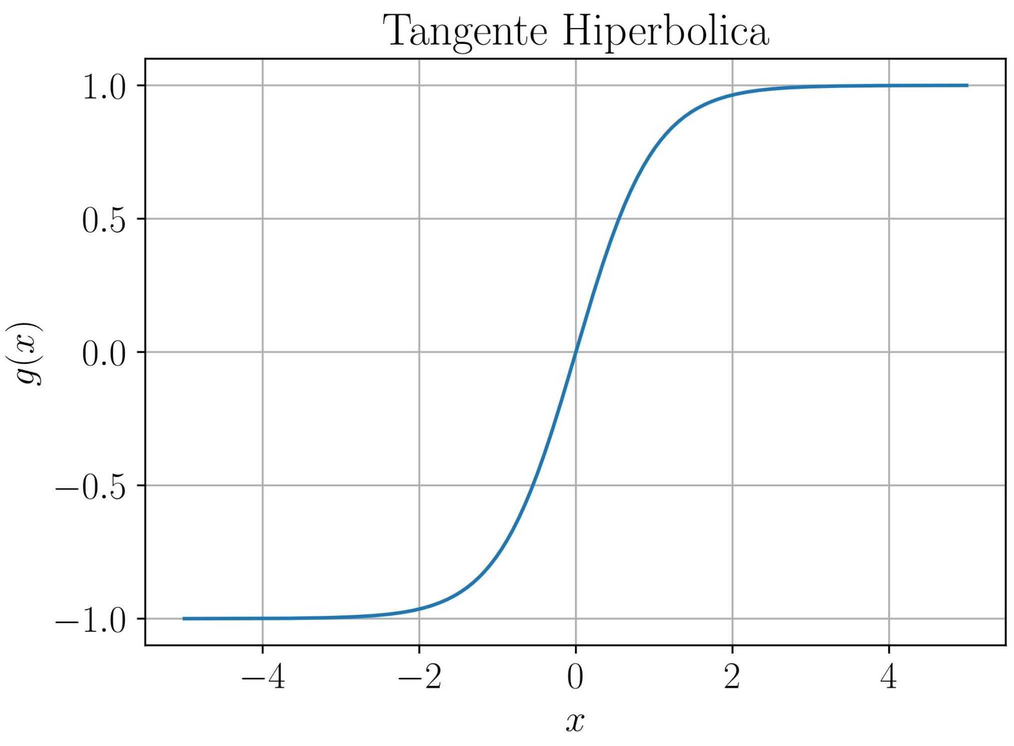


Funções de Ativação

- Tangente Hiperbólica:

$$g(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

(pré-Deep Learning)



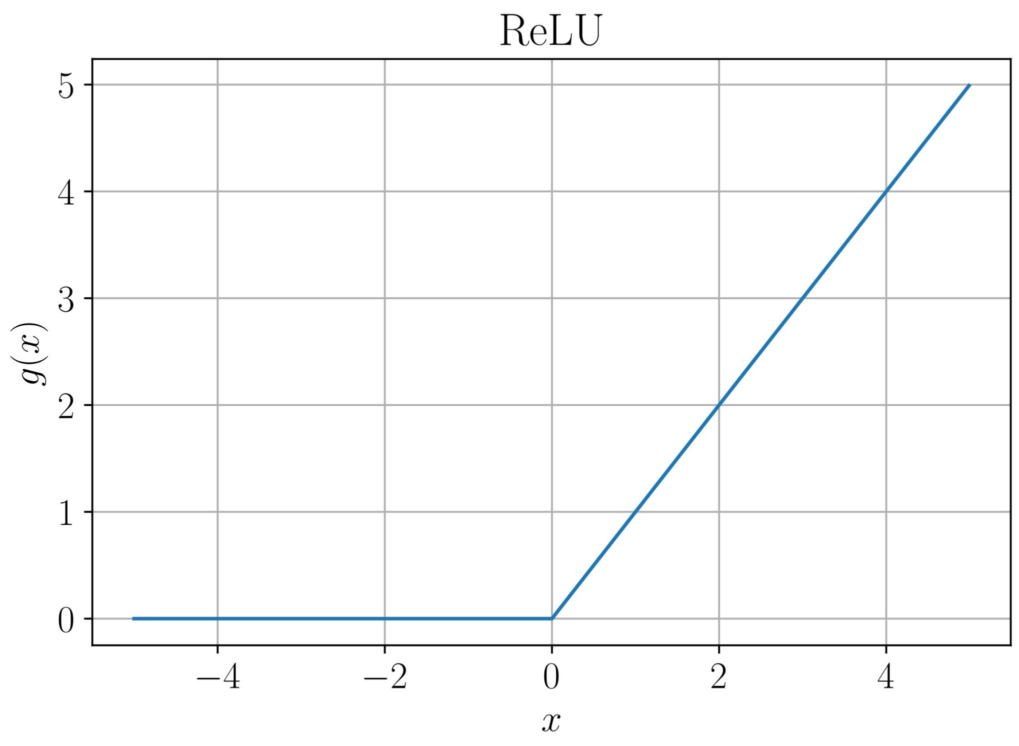
Funções de Ativação

- Rectified Linear Unit (ReLU):

$$g(x) = \begin{cases} 0, x < 0 \\ x, x \geq 0 \end{cases}$$

(Deep Learning)

- Funciona muito bem na prática.

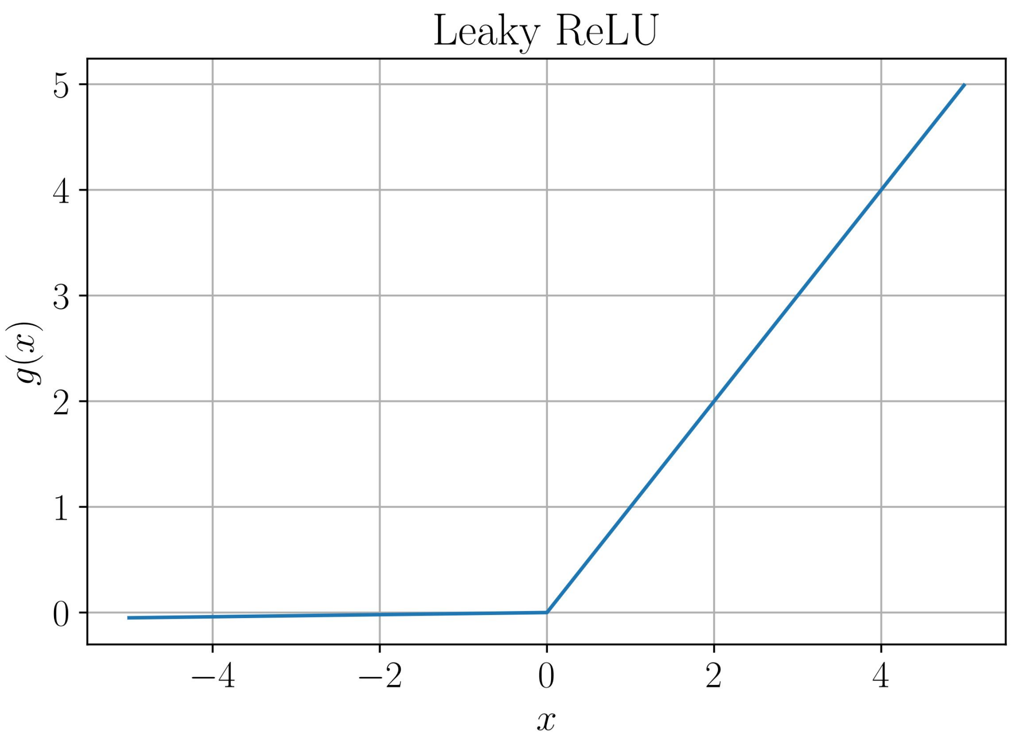


Funções de Ativação

- Leaky ReLU:

$$g(x) = \begin{cases} 0{,}01x;\ x < 0 \\ x;\ x \geq 0 \end{cases}$$

(Deep Learning)



Regressão Linear

- Se usarmos $g(x) = x$, a conta que o neurônio faz é:

$$y = \sum_j w_j x_j + b$$

- Isso é exatamente a expressão da regressão linear múltipla!

Como Treinar o Neurônio?

- Para “ajustar” curvas anteriormente, usamos otimização...
- Otimização é a abordagem mais bem-sucedida de treinamento.
- Vimos antes que Descida de Gradiente é bizu quando conseguimos calcular gradientes.
- Conta do neurônio é combinação linear de pesos e entradas, logo é fácil de derivar.
- O truque então é escolher $g(x)$ fácil de derivar.

Descida de Gradiente para Neurônio

- Ideia do algoritmo: seguir na direção contrária à do gradiente (máximo decrescimento).

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \alpha \frac{\partial J(\boldsymbol{\theta}_n)}{\partial \boldsymbol{\theta}}$$

- α: taxa de aprendizagem (hiperparâmetro).
- $\boldsymbol{\theta} = [\mathbf{w}\ b]^T$ no caso do neurônio são os pesos da rede (incluindo o *bias*).

Função de Custo para Regressão

- Para regressão, usa-se a conhecida função de custo quadrática:

$$J(\boldsymbol{\theta}) = \frac{1}{2m}\sum_{i=1}^{m}\left(\hat{y}^{(i)} - y^{(i)}\right)^2$$

- Na comunidade de ML, é comum usar o termo *loss function* para o custo.
- Pode-se definir também o *loss* de cada exemplo de treinamento:

$$\mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right) = \frac{1}{2}\left(\hat{y}^{(i)} - y^{(i)}\right)^2$$

- Com isso, a função de custo (*loss*) fica:

$$J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right)$$

Gradiente da Regressão Linear

- Com isso, o cálculo do gradiente é parecido com o que fizemos antes:

$$J(\boldsymbol{\theta}) = \frac{1}{2m}\sum_{i=1}^{m}\left(\hat{y}^{(i)} - y^{(i)}\right)^2 = \frac{1}{2m}\sum_{i=1}^{m}\left(\sum_{j} w_j x_j^{(i)} + b - y^{(i)}\right)^2$$

$$\frac{\partial J}{\partial w_k} = \frac{1}{m}\sum_{i=1}^{m}\left(\sum_{j} w_j x_j^{(i)} + b - y^{(i)}\right)x_k^{(i)}$$

$$\frac{\partial J}{\partial b} = \frac{1}{m}\sum_{i=1}^{m}\left(\sum_{j} w_j x_j^{(i)} + b - y^{(i)}\right)$$

Gradiente da Regressão

- Se tivermos uma função de ativação não-linear, precisamos usar a regra da cadeia:

$$J(\boldsymbol{\theta}) = \frac{1}{2m}\sum_{i=1}^{m}\left(g\left(\sum_{j} w_j x_j^{(i)} + b\right) - y^{(i)}\right)^2$$

$$\frac{\partial J}{\partial w_k} = \frac{1}{m}\sum_{i=1}^{m}\left(g\left(\sum_{j} w_j x_j^{(i)} + b\right) - y^{(i)}\right)g'\left(z^{(i)}\right)x_k^{(i)}$$

$$\frac{\partial J}{\partial b} = \frac{1}{m}\sum_{i=1}^{m}\left(g\left(\sum_{j} w_j x_j^{(i)} + b\right) - y^{(i)}\right)g'(z^{(i)})$$

Regressão Logística

- Costuma-se chamar o problema de classificação binária (0 ou 1) usando um neurônio como modelo de regressão logística.
- Nesse caso, é comum o uso de função de ativação sigmoide ou tangente hiperbólica.
- Como a saída é uma sigmoide, usa-se um *threshold* t para determinar se a resposta da rede é 0 ou 1: se saída > t, então é 1, caso contrário, é 0.

Regressão Logística

- Possíveis resultados da classificação: verdadeiro positivo (TP), falso positivo (FP), falso negativo (FN) e verdadeiro negativo (TN).
- Precisão (*precision*):

$$Precision = \frac{TP}{TP + FP}$$

- Sensibilidade (*recall*):

$$Recall = \frac{TP}{TP + FN}$$

- *F1 Score*:

$$F1 = \left(\frac{Precision^{-1} + Recall^{-1}}{2}\right)^{-1} = 2\,\frac{Precision * Recall}{Precision + Recall}$$

Função de Custo para Classificação

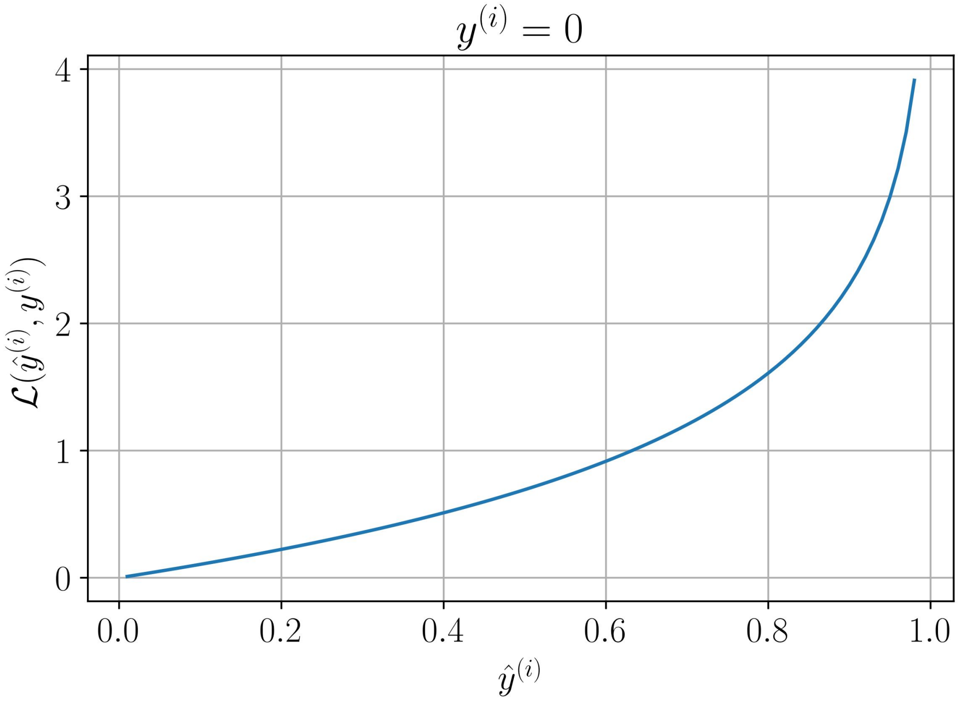
- Para classificação, usa-se a seguinte *loss function*:

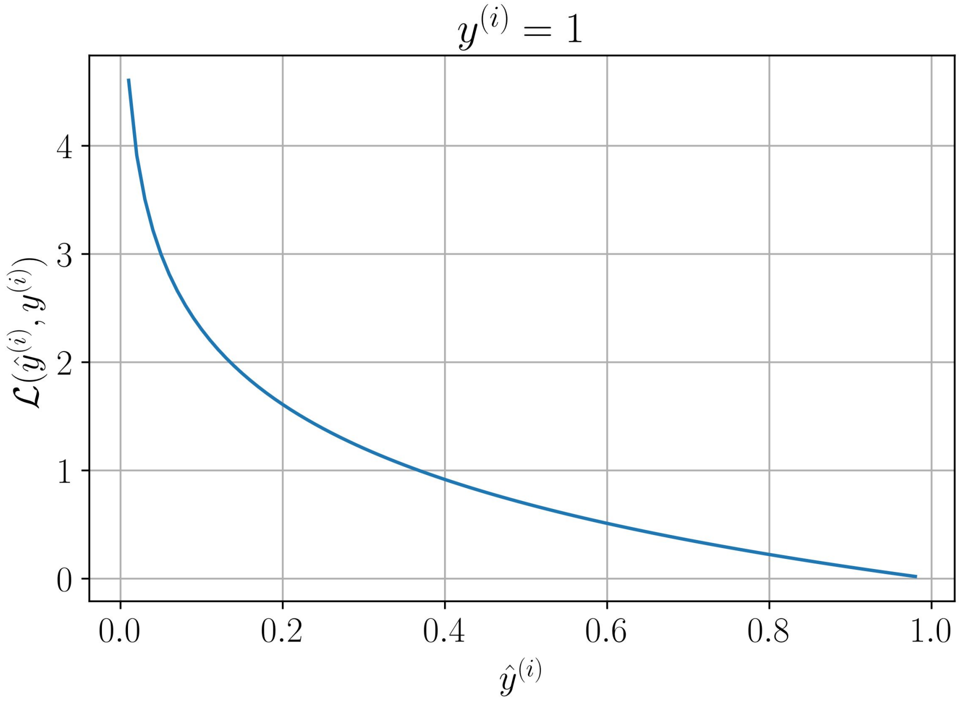
$$\mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right) = -\left[y^{(i)} \log\left(\hat{y}^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - \hat{y}^{(i)}\right)\right]$$

- A base usada no logaritmo é e.
- Se $y^{(i)} = 0$, então $\mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right) = -\log\left(1 - \hat{y}^{(i)}\right)$.
- Se $y^{(i)} = 1$, então $\mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right) = -\log\left(\hat{y}^{(i)}\right)$.

Função de Custo para Classificação (Intuição)

$y^{(i)} = 0\colon \mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right) = -\log\left(1 - \hat{y}^{(i)}\right)$ $y^{(i)} = 1\colon \mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right) = -\log\left(\hat{y}^{(i)}\right)$





Gradiente para Regressão Logística

- Considerando $\sigma(x)$ como função de ativação.

$$\mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right) = -\left[y^{(i)} \log\left(\hat{y}^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - \hat{y}^{(i)}\right)\right]$$

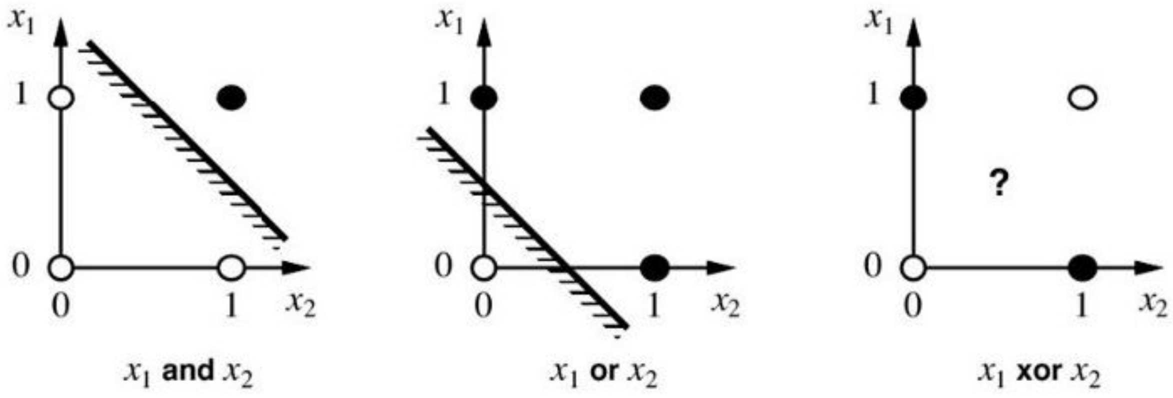
$$\frac{\partial \mathcal{L}}{\partial w_k} = -\left(y \frac{1}{\hat{y}}\, \hat{y}(1 - \hat{y}) - (1 - y) \frac{1}{1 - \hat{y}} \hat{y}(1 - \hat{y})\right) x_k = (\hat{y} - y) x_k$$

$$\frac{\partial \mathcal{L}}{\partial b} = \hat{y} - y$$

Redes Neurais

Redes Neurais

- Pesquisas mostraram que o poder de representação de um único neurônio é muito limitado.
- Pode-se mostrar que um único neurônio é capaz de classificar apenas *datasets* linear separáveis.



Redes Neurais

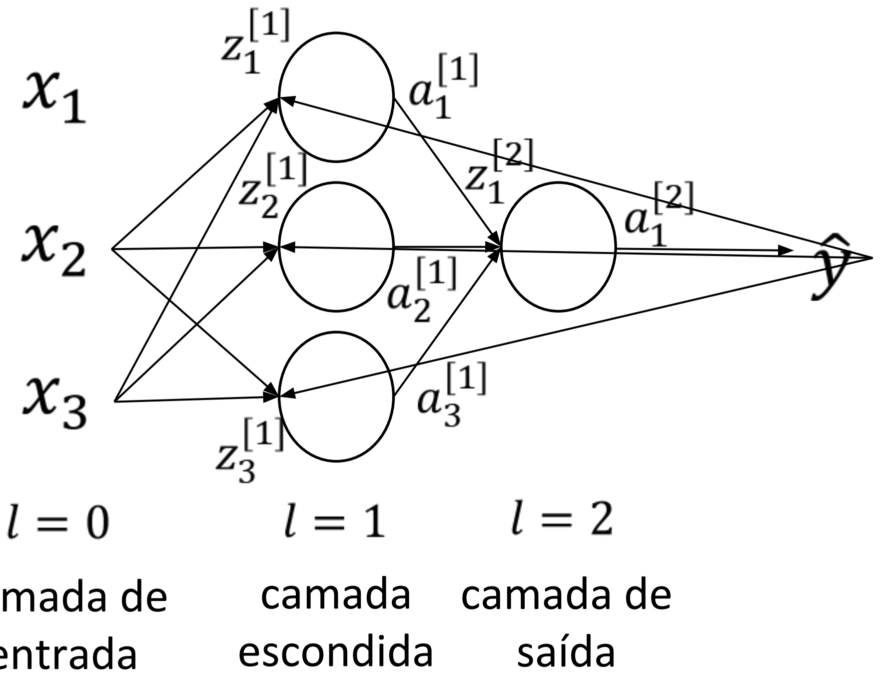
- Redes neurais são construídas ligando neurônios uns nos outros.
- A forma como os neurônios são ligados define a *arquitetura* da rede.
- O tipo de arquitetura mais simples é das *Forward Neural Networks*, em que não há ciclos.
- Em geral, organiza-se os neurônios em camadas, de modo que saídas dos neurônios de uma camada são entradas dos neurônios da próxima camada.
- Quando todos os neurônios de uma camada são conectados com os da camada posterior, chama-se *Fully-Connected Neural Network*.

Redes Neurais

- Cada camada pode ter vários neurônios.
- A rede pode ter múltiplas camadas.
- A rede pode ter múltiplas saídas.
- As camadas “intermediárias” são chamadas de escondidas.
- Teorema da aproximação universal: qualquer função contínua que mapeia um intervalo de números reais em outro intervalo de números reais pode ser aproximada arbitrariamente bem com uma rede neural com uma única camada escondida.
- Esse teorema vale para uma grande classe de funções de ativação.

Redes Neurais

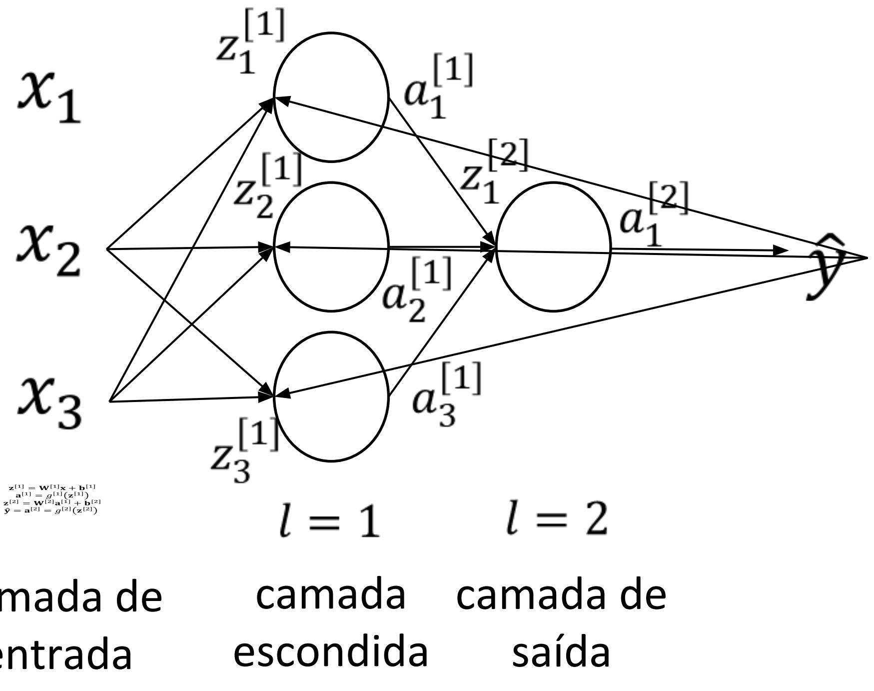
Número de camadas: $L = 2$ (não conta a de entrada)



- $$z_1^{[1]} = w_{11}^{[1]} x_1 + w_{12}^{[1]} x_2 + w_{13}^{[1]} x_3 + b_1^{[1]}$$
  $$z_2^{[1]} = w_{21}^{[1]} x_1 + w_{22}^{[1]} x_2 + w_{23}^{[1]} x_3 + b_2^{[1]}$$
  $$z_3^{[1]} = w_{31}^{[1]} x_1 + w_{32}^{[1]} x_2 + w_{33}^{[1]} x_3 + b_3^{[1]}$$
  $$a_1^{[1]} = g^{[1]}\left(z_1^{[1]}\right)$$
  $$a_2^{[1]} = g^{[1]}\left(z_2^{[1]}\right)$$
  $$a_3^{[1]} = g^{[1]}\left(z_3^{[1]}\right)$$
  $$z_1^{[2]} = w_{11}^{[2]} a_1^{[1]} + w_{12}^{[2]} a_2^{[1]} + w_{13}^{[2]} a_3^{[1]} + b_1^{[2]}$$
  $$\hat{y} = a_1^{[2]} = g^{[2]}\left(z_1^{[2]}\right)$$

Redes Neurais (Vetorização)

Número de camadas: $L = 2$ (não conta a de entrada)



- $$\mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}$$
  $$\mathbf{a}^{[1]} = g^{[1]}\left(\mathbf{z}^{[1]}\right)$$
  $$\mathbf{z}^{[2]} = \mathbf{W}^{[2]}\mathbf{a}^{[1]} + \mathbf{b}^{[2]}$$
  $$\hat{\mathbf{y}} = \mathbf{a}^{[2]} = g^{[2]}\left(\mathbf{z}^{[2]}\right)$$

Redes Neurais (Vetorização)

- Generalizando para L camadas:

$$\mathbf{a}^{[0]} = \mathbf{x}$$

for $l = 1 \colon L$:

$$\mathbf{z}^{[l]} = \mathbf{W}^{[l]}\mathbf{a}^{[l-1]} + \mathbf{b}^{[l]}$$

$$\mathbf{a}^{[l]} = g^{[l]}\left(\mathbf{z}^{[l]}\right)$$

$$\hat{\mathbf{y}} = \mathbf{a}^{[L]}$$

Redes Neurais (Vetorização)

- Generalizando para m exemplos de treinamento:

$\mathbf{A}^{[0]} = \mathbf{X}$

for $l = 1{:}\, L$:

$$\mathbf{Z}^{[l]} = \mathbf{W}^{[l]}\mathbf{A}^{[l-1]} + \mathbf{B}^{[l]}$$

$$\mathbf{A}^{[l]} = g^{[l]}(\mathbf{Z}^{[l]})$$

$\widehat{\mathbf{Y}} = \mathbf{A}^{[L]}$

em que:

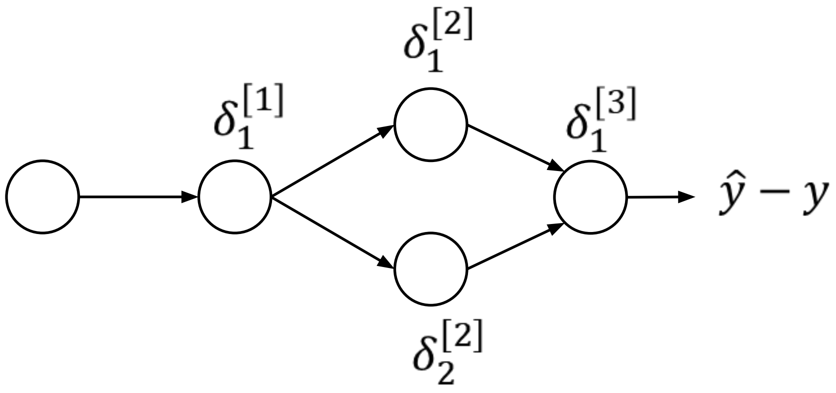
$$\mathbf{X} = \left[\mathbf{x}^{(1)}\ \mathbf{x}^{(2)}\ \cdots\ \mathbf{x}^{(m)}\right], \mathbf{B}^{[l]} = \left[\mathbf{b}^{[l]}\ \mathbf{b}^{[l]} \cdots \mathbf{b}^{[l]}\right]$$

Descida de Gradiente para Redes Neurais

- Considerar *loss function* por exemplo quadrática:

$$\mathcal{L} = \frac{1}{2}\left(\hat{y}^{(i)} - y^{(i)}\right)^2$$

- Vamos assumir rede feedfoward com 1, 1, 2, 1 neurônios:



Descida de Gradiente para Redes Neurais

- Cálculo do gradiente.

$$\frac{\partial \mathcal{L}}{\partial w_{11}^{[3]}} = (\hat{y} - y)\frac{\partial}{\partial w_{11}^{[3]}} g^{[3]}\left(w_{11}^{[3]} a_1^{[2]} + w_{12}^{[2]} a_2^{[2]}\right)$$

$$= \underbrace{\boxed{(\hat{y} - y) g^{[3]'}\left(z_1^{[3]}\right)}}_{= \delta_1^{[3]}} a_1^{[2]}$$

$$\frac{\partial \mathcal{L}}{\partial w_{12}^{[3]}} = \delta_1^{[3]} a_2^{[2]}$$

Descida de Gradiente para Redes Neurais

- $$\frac{\partial \mathcal{L}}{\partial w_{11}^{[2]}} = (\hat{y} - y)\frac{\partial}{\partial w_{11}^{[2]}} g^{[3]}\left(w_{11}^{[3]} g^{[2]}\left(w_{11}^{[2]} a_1^{[1]}\right) + w_{12}^{[2]}\left(w_{21}^{[2]} a_2^{[1]}\right)\right)$$
$$= (\hat{y} - y) g^{[3]\prime}\left(z_1^{[3]}\right) w_{11}^{[3]} g^{[2]\prime}\left(z_1^{[2]}\right) a_1^{[1]} = \underbrace{\boxed{w_{11}^{[3]} \delta_1^{[3]} g^{[2]\prime}\left(z_1^{[2]}\right)}}_{=\delta_1^{[2]}} a_1^{[1]}$$

$$\frac{\partial \mathcal{L}}{\partial w_{21}^{[2]}} = \underbrace{\boxed{w_{11}^{[3]} \delta_1^{[3]} g^{[2]\prime}\left(z_2^{[2]}\right)}}_{=\delta_2^{[2]}} a_1^{[1]}$$

Descida de Gradiente para Redes Neurais

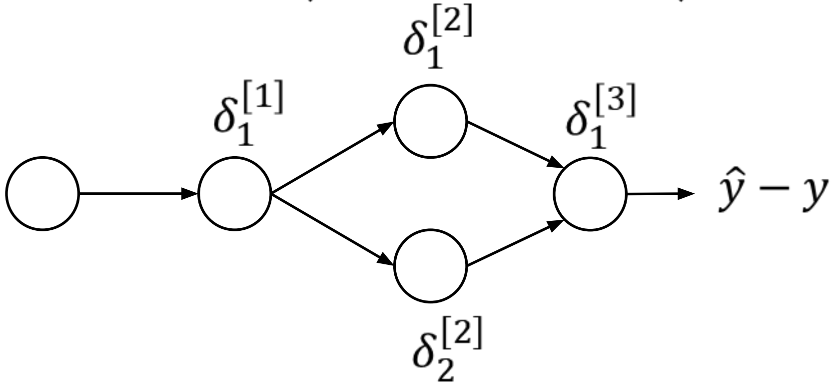
- $\dfrac{\partial \mathcal{L}}{\partial w_{11}^{[1]}} = (\hat{y} - y)\dfrac{\partial}{\partial w_{11}^{[1]}}\left\{g^{[3]}\left(w_{11}^{[3]} g^{[2]}\left(w_{11}^{[2]} g(w_{11}^{[1]} x_1\right) + w_{12}^{[2]}\left(w_{21}^{[2]} g(w_{11}^{[1]} x_1\right)\right)\right\}$

$$\Rightarrow \frac{\partial \mathcal{L}}{\partial w_{11}^{[1]}} = \underbrace{\boxed{\left(\sum_{j=1}^{2} w_{j1}^{[2]} \delta_j^{[2]}\right)}}_{= \delta_1^{[}} g^{[1]\prime}(z_1^{[1]}) x_1$$

Descida de Gradiente para Redes Neurais

- $$\frac{\partial \mathcal{L}}{\partial w_{11}^{[3]}} = \delta_1^{[3]} a_1^{[2]}$$
  $$\frac{\partial \mathcal{L}}{\partial w_{12}^{[3]}} = \delta_1^{[3]} a_1^{[2]}$$
  $$\frac{\partial \mathcal{L}}{\partial w_{11}^{[2]}} = \delta_1^{[2]} a_1^{[1]}$$
  $$\frac{\partial \mathcal{L}}{\partial w_{21}^{[2]}} = \delta_2^{[2]} a_2^{[1]}$$
  $$\frac{\partial \mathcal{L}}{\partial w_{11}^{[1]}} = \delta_1^{[1]} x_1$$

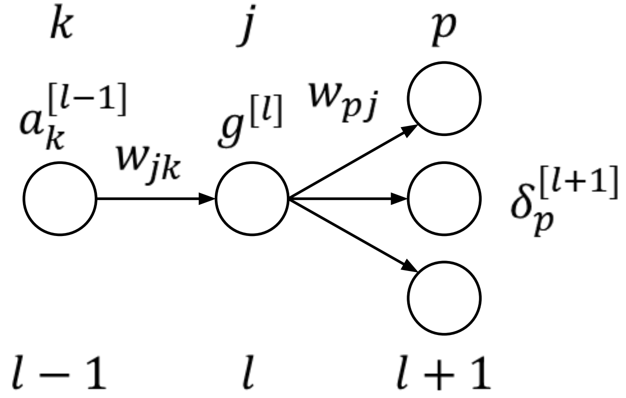
- $$\delta_1^{[3]} = (\hat{y} - y) g^{[3]'}\left(z_1^{[3]}\right)$$
  $$\delta_1^{[2]} = w_{11}^{[3]} \delta_1^{[3]} g^{[2]'}\left(z_1^{[2]}\right)$$
  $$\delta_2^{[2]} = w_{11}^{[3]} \delta_1^{[3]} g^{[2]'}\left(z_2^{[2]}\right)$$
  $$\delta_1^{[1]} = \left(\sum_{j=1}^{2} w_{j1}^{[2]} \delta_j^{[2]}\right) x_1$$



Backpropagation (Fórmula Geral)

- $$\frac{\partial \mathcal{L}}{\partial w_{jk}^{[l]}} = \delta_j^{[l]} a_k^{[l-1]}$$

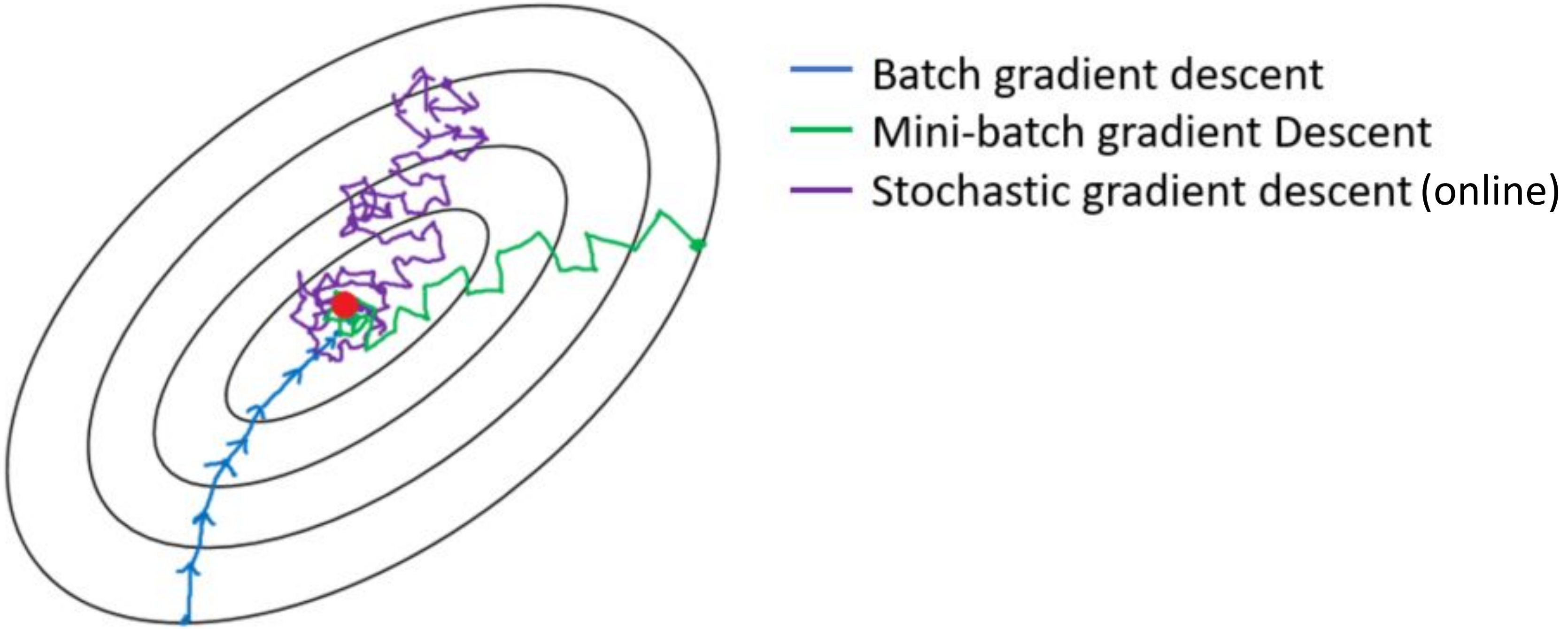
$$\delta_j^{[l]} = \begin{cases} \left(\sum_{p=1}^{n_{l+1}} w_{pj}^{[l+1]} \delta_p^{[l+1]}\right) g^{[l]'}\left(z_j^{[l]}\right), l \neq L-1 \\ \left(\hat{y}_j^{(i)} - y_j^{(i)}\right) g^{[l]'}\left(z_j^{[l]}\right), l = L-1 \end{cases}$$



Descida de Gradiente Estocástica

- Inglês: *Stochastic Gradient Descent*.
- Em ML, é comum um conjunto de dados de treinamento (*dataset*) muito grande, o que torna o passo de treinamento muito pesado.
- No caso de visão, é comum o *dataset* **não** caber na memória RAM.
- Ideia: usar um conjunto menor de dados (mini-batch) a cada passo, com casos escolhidos aleatoriamente dentre o *dataset* original.
- Mais um hiperparâmetro: tamanho do mini-batch.
- Apesar de mais ruidoso, o algoritmo converge em esperança.
- Três modos de treinamento: online, mini-batch e batch.

Descida de Gradiente Estocástica



Dicas para Redes Neurais

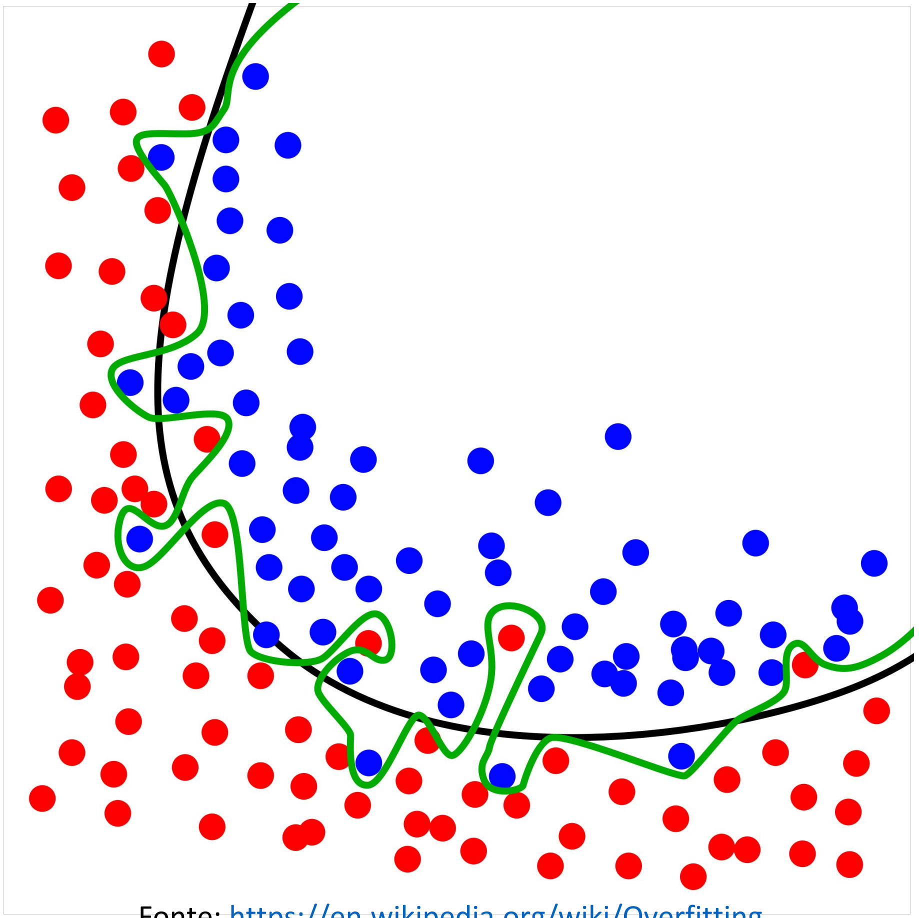
Inicialização dos Pesos

- Bias podem ser inicializados como zero (prática comum).
- Não é bom inicializar pesos como zero.
- Em geral, inicializa-se os pesos com números aleatórios:

$$w_{jk}^{[l]} \sim 0{,}01 * N(0{,}1)$$

Overfitting

- Um problema que acontece com redes neurais, especialmente se forem grandes.
- Após muitas iterações de treinamento, a rede vai tentar se ajustar perfeitamente aos dados de treinamento.
- Isso pode prejudicar sua capacidade de **generalização**.



Fonte: https://en.wikipedia.org/wiki/Overfitting

Overfitting

- Uma forma de identificar *overfitting* é dividir os dados de treinamento em conjunto de treinamento e conjunto de teste.
- Regra de bolso: 70% treinamento/30% teste.
- Assim, se a rede for bem no conjunto de treinamento, mas ruim no de teste, então há *overfitting*.
- Uma técnico que pode ser usada é parar o treinamento assim que a rede começar a piorar no conjunto de teste.

Como Determinar a Arquitetura da Rede?

- Não há métodos formais...
- Em geral, escolha se baseia em experiência e tentativa e erro.
- Uma ideia é buscar “inspiração” em arquiteturas usadas para resolver problemas semelhantes.
- Redes maiores costumam precisar de mais dados.
- Outra ideia é testar várias arquiteturas e escolher a melhor.

Escolha de Modelo

- Durante o teste de diferentes arquiteturas, é importante ter uma métrica para avaliar a qualidade de cada arquitetura.
- Devido ao problema de *overfitting*, não basta ter melhor desempenho no conjunto de treinamento.
- Podemos usar o conjunto de teste então para comparar diferentes modelos, porém no final o próprio conjunto de teste perde um pouco de seu significado porque passa a fazer parte do “treinamento”.
- Costuma-se então separar os dados em 3 conjuntos: treinamento, *cross-validation* e teste. Regra de bolso: 60/20/20 ou 40/30/30.
- Observação: mundo de *Deep Learning*: 90/5/5.

Regularização

- Em geral, para gerar *overfitting*, o algoritmo precisa convergir para pesos muito altos.
- Uma forma de regularização (L_2) envolve adicionar uma parcela que penaliza parâmetros (pesos e biases) muito altos:

$$J(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{m} \mathcal{L}\left(\hat{y}, y^{(i)}\right) + \frac{\lambda}{2m}\sum_{j} \theta_j^2$$

- Dica (Ng): Tentar $\lambda = \{0; 0{,}01; 0{,}02; 0{,}04; \ldots; 10{,}24\}$.
- Se exagerar na regularização, acontece *underfitting*.
- Pode-se usar *cross-validation* para encontrar o melhor valor de λ.

Classificação Multi-Label

- Imagine o caso em que queremos classificar um paciente como portador de diversas doenças possíveis (classes): pneumonia, hipertensão, diabetes e câncer.
- Pode-se associar cada doença como sendo uma saída da rede neural:

Pneumonia: $[1\ 0\ 0\ 0]^T$.

Hipertensão: $[0\ 1\ 0\ 0]^T$.

Diabetes: $[0\ 0\ 1\ 0]^T$.

Câncer: $[0\ 0\ 0\ 1]^T$.

Hipertensão e diabetes: $[0\ 1\ 1\ 0]^T$.

Classificação Multi-Classe

- Imagine o caso em que queremos classificar uma imagem entre um conjunto de possíveis animais (classes mutuamente exclusivas): gato, cachorro, papagaio e tartaruga.
- Truque: considerar vetor c-dimensional, em que c é o número de classes (*one-hot encoding*):

Gato: $[1\ 0\ 0\ 0]^T$.

Cachorro: $[0\ 1\ 0\ 0]^T$.

Papagaio: $[0\ 0\ 1\ 0]^T$.

Tartaruga: $[0\ 0\ 0\ 1]^T$.

Classificação Multi-Classe/Multi-Label

- *Loss function* (multi-label):

$$\mathcal{L}\left(\widehat{\boldsymbol{y}}, \boldsymbol{y}^{(i)}\right) = \sum_{k=1}^{c} -\left[y_k^{(i)} \log\left(\hat{y}_k^{(i)}\right) + \left(1 - y_k^{(i)}\right) \log\left(1 - \hat{y}_k^{(i)}\right)\right]$$

- Vários sigmoides na camada de saída.
- Para classificação multi-classe, costuma-se usar o que der o maior valor de saída.
- Observação: esta não é a melhor forma de fazer classificação multi-classe. Na próxima aula, veremos outra forma de fazer.

Para Saber Mais

- Curso de *Machine Learning* do Andrew Ng no Cousera.
- Especialização de *Deep Learning* do Andrew Ng no Cousera.
- Capítulos 5 e 6 do livro: GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. *Deep Learning*. The MIT Press, 2016.

Laboratório 6

Laboratório 6

- Implementação de rede neural em Python.
- Usando NumPy.
- Problema: segmentação de cores.