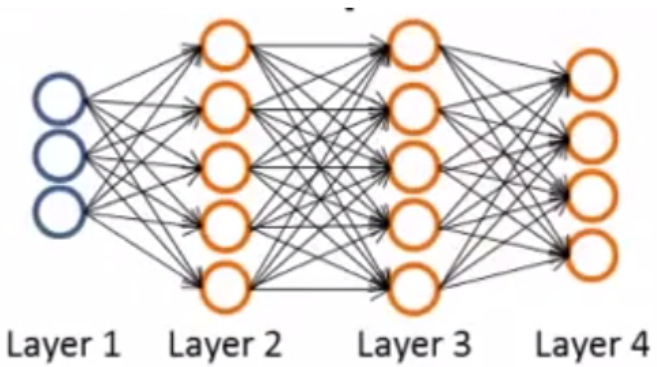
Redes Neurais:
Aprendizagem

Prof.: Eric A. Antonelo

Slides baseados no curso de *Machine Learning* de Andrew Ng

DAS-UFSC

Classificação



$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$

$L =$ Número total de camadas

$s_l =$ Número de unidades na camada *l* (sem contar o *bias*)

Classificação Binária

y = 0 ou 1

1 unidade de saída

Classificação com mais de 2 classes

$y \in \mathbb{R}^K$ E.g. $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

pedestrian car motorcycle truck

K unidades de saída

Função de custo

Regressão Logística:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2$$

Rede Neural:

$$h_\Theta(x) \in \mathbb{R}^K \quad (h_\Theta(x))_i = i^{th} \text{ output}$$

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k)\right]$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

Convenção: não inclui *bias*

Questão

- Suponha que queremos minimizar $J(\Theta)$ como uma função de Θ usando um dos métodos avançados de otimização (gradiente conjugado, BFGS, L-BFGS, etc.). O que precisamos fornecer como código para o método?

Algoritmo de Retro-propagação (*Backpropagation*)

Cálculo do gradiente

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log h_\theta(x^{(i)})_k + (1 - y_k^{(i)}) \log(1 - h_\theta(x^{(i)})_k)\right]$$
$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_j^{(l)})^2$$

$$\min_{\Theta} J(\Theta)$$

É necessário o Código para calcular:

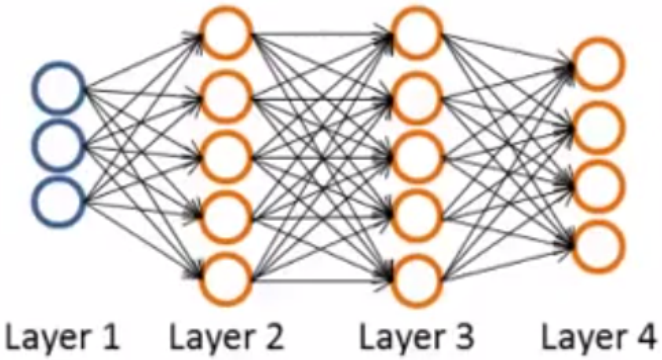
- $J(\Theta)$
- $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$ 

Cálculo do gradiente

Dado um exemplo de treinamento (*x,y*):

Propagação para a frente (*forward*):

$$
\begin{aligned}
a^{(1)} &= x \\
z^{(2)} &= \Theta^{(1)} a^{(1)} \\
a^{(2)} &= g(z^{(2)}) \quad (\text{add } a_0^{(2)}) \\
z^{(3)} &= \Theta^{(2)} a^{(2)} \\
a^{(3)} &= g(z^{(3)}) \quad (\text{add } a_0^{(3)}) \\
z^{(4)} &= \Theta^{(3)} a^{(3)} \\
a^{(4)} &= h_\Theta(x) = g(z^{(4)})
\end{aligned}
$$

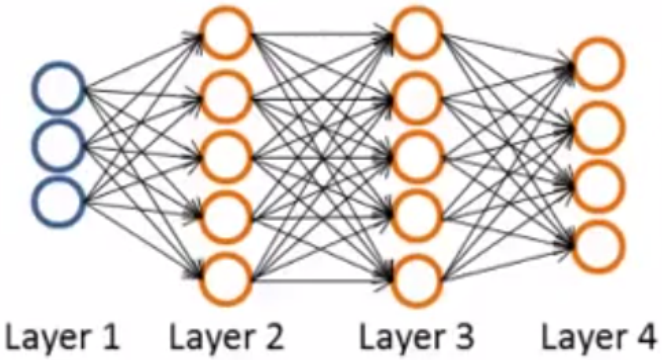


Cálculo do gradiente

Intuição: $\delta_j^{(l)}$ "erro" da unidade j da camada l

Para cada unidade de saída (L=4)

$$\delta_j^{(4)} = a_j^{(4)} - y_j$$

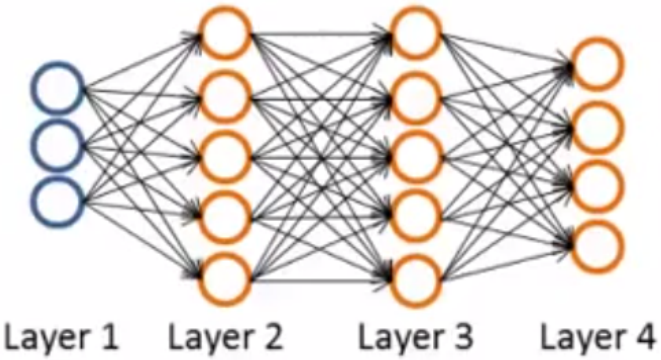


Cálculo do gradiente

Intuição: $\delta_j^{(l)}$ "erro" da unidade *j* da camada *l*

Para cada unidade de saída (L=4)

$$\delta_j^{(4)} = a_j^{(4)} - y_j$$



$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$$

$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$$

Algoritmo de Retro-propagação

Training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

Set $\triangle_{ij}^{(l)} = 0$ (for all l, i, j).

Usado para calcular as derivadas parciais

For $i = 1$ to m

Set $a^{(1)} = x^{(i)}$

Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \ldots, L$

Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$

$\triangle_{ij}^{(l)} := \triangle_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

$$D_{ij}^{(l)} := \frac{1}{m} \triangle_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} \text{ if } j \neq 0$$

$$D_{ij}^{(l)} := \frac{1}{m} \triangle_{ij}^{(l)} \text{ if } j = 0$$

Termos “bias”

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

Verificação do Gradiente

Estimação numérica dos gradientes

quadro

Aproximação do gradiente: Caso geral

$$\frac{\partial}{\partial \theta_1} J(\theta) \approx \frac{J(\theta_1+\epsilon,\theta_2,\theta_3,\ldots,\theta_n)-J(\theta_1-\epsilon,\theta_2,\theta_3,\ldots,\theta_n)}{2\epsilon}$$

$$\frac{\partial}{\partial \theta_2} J(\theta) \approx \frac{J(\theta_1,\theta_2+\epsilon,\theta_3,\ldots,\theta_n)-J(\theta_1,\theta_2-\epsilon,\theta_3,\ldots,\theta_n)}{2\epsilon}$$

$$\vdots$$

$$\frac{\partial}{\partial \theta_n} J(\theta) \approx \frac{J(\theta_1,\theta_2,\theta_3,\ldots,\theta_n+\epsilon)-J(\theta_1,\theta_2,\theta_3,\ldots,\theta_n-\epsilon)}{2\epsilon}$$

Código Matlab/Octave

```
for i = 1:n,
    thetaPlus = theta;
    thetaPlus(i) = thetaPlus(i) + EPSILON;
    thetaMinus = theta;
    thetaMinus(i) = thetaMinus(i) - EPSILON;
    gradApprox(i) = (J(thetaPlus) - J(thetaMinus))
                    /(2*EPSILON);
end;
```

Notas de Implementação

- Implementar **retro-propagação** para calcular o gradiente.
- Implementar **verificação numérica do gradiente** para calcular uma aproximação.
- Checar se ambos fornecem os mesmos valores.
- Remover a verificação numérica (lento) para a etapa de treinamento da rede.

Inicialização aleatória

Valor inicial para os pesos

Para o método do descenso (gradiente) e outros métodos avançados de otimização

```
optTheta = fminunc(@costFunction,
           initialTheta, options)
```

Considere o método do descenso

```
initialTheta = zeros(n,1)
```
?

Inicialização com zeros?

quadro

Inicialização aleatória: Quebra de simetria

Inicializar cada $\Theta_{ij}^{(l)}$ para um valor aleatório $[-\epsilon, \epsilon]$

```
Theta1 =  rand(10,11)*(2*INIT_EPSILON)
          - INIT_EPSILON;
```

Treinando uma rede neural

1. Inicializar os pesos aleatoriamente
2. Implementar propagação para a frente
3. Implementar código para calcular a função de custo
4. Implementar *backprop* para calcular as derivadas parciais
5. Para todos os exemplos de treinamento
    1. Fazer propagação para a frente e backpropagation usando o exemplo atual
    2. Obter as ativações e os termos delta
6. Usar verificação de gradiente
7. Usar método do descenso do gradiente ou outro método avançado para minimizar a função de custo