(2) Perceptrons e Multi-layer Perceptrons

Redes Neurais e Aprendizado Profundo

Moacir Antonelli Ponti

ICMC, Universidade de São Paulo

www.icmc.usp.br/~moacir — moacir@icmc.usp.br

São Carlos-SP/Brasil

Agenda

Introdução: problemas de regressão e classificação

Perceptron

Multi-layer Perceptron

Agenda

Introdução: problemas de regressão e classificação

Perceptron

Multi-layer Perceptron

Modelo linear

- Descreve como os atributos (features) de entrada podem ser transformados numa estimativa do alvo.
- Se o conjunto de funções admissíveis incluir apenas funções lineares temos algo na forma:

$$\hat{y} = wx + b$$

- No caso de existirem duas entradas por exemplo teríamos:

$$\hat{y} = w_1x_1 + w_2x_2 + b$$

Modelo linear: notação algébrica

- A forma geral para um número d de features seria:

$$\hat{y} = w_1 x_1 + \ldots + w_d x_d + b$$

- Como em geral d é de alta dimensionalidade a notação algébrica é mais comum. Seja $\mathrm{x} \in R^d$:

$$\hat{y} = \mathrm{w}^T \mathrm{x} + b$$

- Para um dataset $\mathrm{X} \in R^{n \times d}$, teremos um valor estimado para cada um dos n elementos:

$$\hat{\mathrm{y}} = \mathrm{X}\mathrm{w} + b,$$

OBS: o último b vai ser somado com broadcasting ;)

Relembrando: função de custo/perda

$$\ell^{(i)}(\mathbf{w}, b) = \frac{1}{2}(\hat{y}^{(i)} - y^{(i)})^2$$

- perda quadrática com relação ao exemplo i

$$L(\mathbf{w}, b) = \frac{1}{n}\sum_{i=1}^{n} \ell^{(i)}(\mathbf{w}, b) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{2}(\mathbf{w}^T \mathbf{x}^{(i)} - y^{(i)})^2$$

O que queremos?

Encontrar os parâmetros que minimizem a perda média com relação aos elementos de treinamento

$$\mathrm{w}^*, b^* = \arg\min_{\mathrm{w}, b} L(\mathrm{w}, b)$$

Descida do gradiente estocástica em lote

Minibach stochastic gradient descent

- Definir um minibatch de tamanho B
- Definir um tamanho de passo α
- Inicializar parâmetros
- Usar o negativo do gradiente para atualizar os parâmetros

$$(\mathrm{w}, b) = (\mathrm{w}, b) - \frac{\alpha}{|B|} \sum_{i \in B} \partial_{(\mathrm{w}, b)} \ell^{(i)}(\mathrm{w}, b)$$

Agenda

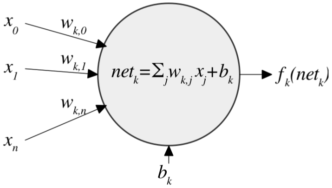
Introdução: problemas de regressão e classificação

Perceptron

Multi-layer Perceptron

Neurônio

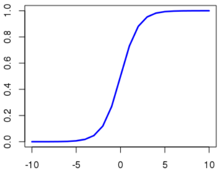
- entrada: valores organizados em um vetor
- saída: um único valor
  - cada valor de entrada é associado a um peso *w* (força da conexão)
  - o bias *b* funciona como intercepto
- aprender é ajustar *w*'s e *b*'s aos dados de treinamento
- há uma função aplicada nessa soma, a qual é chamada função de ativação



Algumas funções de ativação

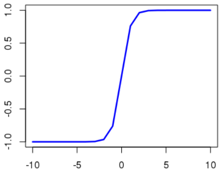
Sigmoid

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



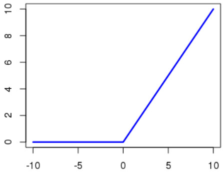
Hiperbolic Tangent

$$f(x) = \tanh(x)$$



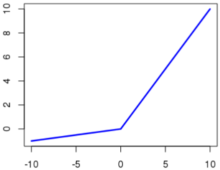
ReLU

$$f(x) = \max(0, x)$$



Leaky ReLU

$$f(x) = \max(0.1x, x)$$



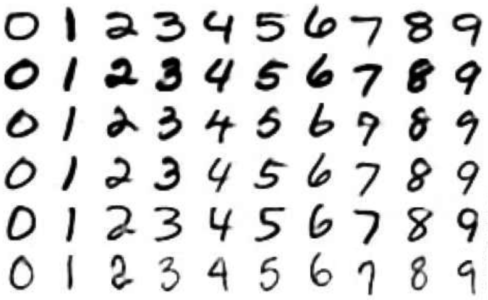
Agenda

Introdução: problemas de regressão e classificação

Perceptron

Multi-layer Perceptron

Exemplo de problema: classificação de dígitos



- Imagens com $28 \times 28 = 784$ pixels,
- Redes do tipo Perceptron,
- Algoritmo SGD com 32 imagens no batch,
- Camada de saída contendo 10 classes do problema.

"Classificador" Softmax

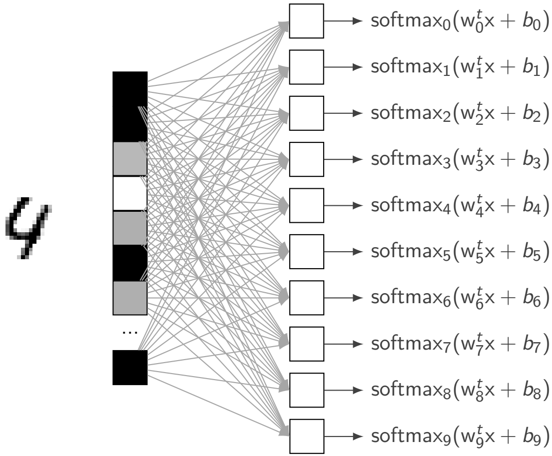
- Normalizar saída de forma a somar 1
- Permite interpretar cada valor como sendo uma probabilidade
- Na saída, temos a distribuição de probabilidades das classes i

$$\text{softmax}(\hat{y}_i) = \frac{e^{\hat{y}_i}}{\sum_j e^{\hat{y}_j}}$$

- O rótulo passa a ser um vetor binário contendo 1 na posição da classe correta (one hot encoding)

Rede neural rasa, com uma única camada

Pixels da imagem organizados em vetor



Formulação da rede neural

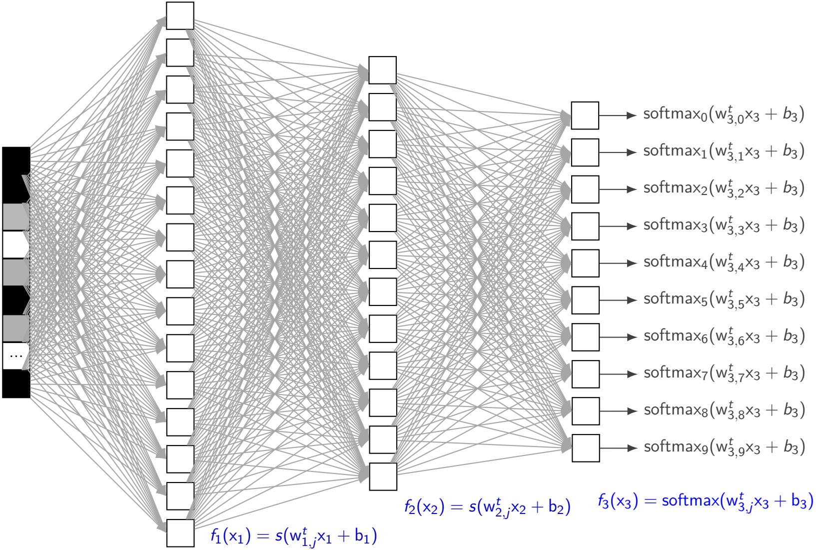
10 classes, batch-size 32, e 784 características (pixels) por imagem

$$\begin{bmatrix} x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,783} \\ x_{1,0} & x_{0,1} & x_{1,2} & \cdots & x_{0,783} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{31,0} & x_{31,1} & x_{31,2} & \cdots & x_{31,783} \end{bmatrix} \begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,9} \\ w_{1,0} & w_{1,1} & \cdots & w_{1,9} \\ w_{2,0} & w_{2,1} & \cdots & w_{2,9} \\ \vdots & \vdots & \ddots & \vdots \\ w_{783,0} & w_{783,1} & \cdots & w_{783,9} \end{bmatrix} + \begin{bmatrix} b_0 \; b_1 \; b_2 \ldots b_9 \end{bmatrix}$$

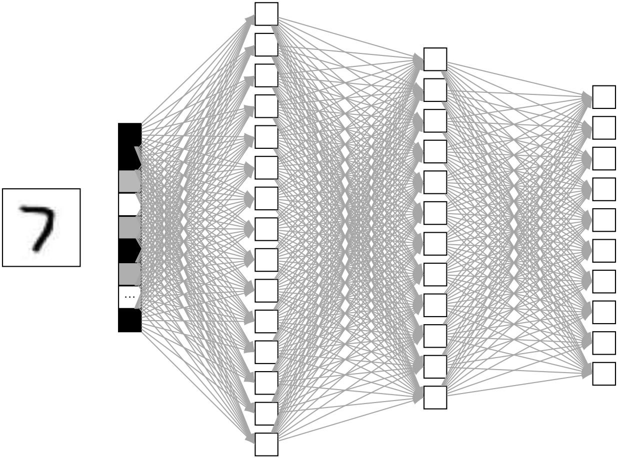
$$\mathrm{Y} = \mathrm{softmax}(\mathrm{X} \cdot \mathrm{W} + \mathrm{b})$$

$$\hat{\mathrm{Y}} = \begin{bmatrix} y_{0,0} & y_{0,1} & y_{0,2} & \cdots & y_{0,9} \\ y_{1,0} & y_{1,1} & y_{1,2} & \cdots & y_{1,9} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ y_{31,0} & y_{31,1} & y_{31,2} & \cdots & y_{31,9} \end{bmatrix}$$

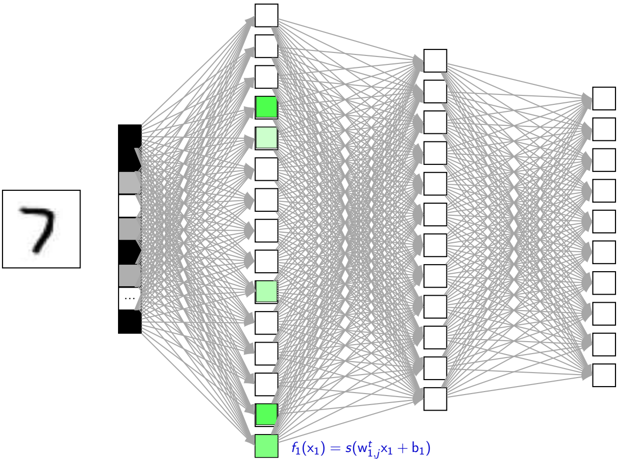
Rede MLP "profunda" com 2 camadas ocultas



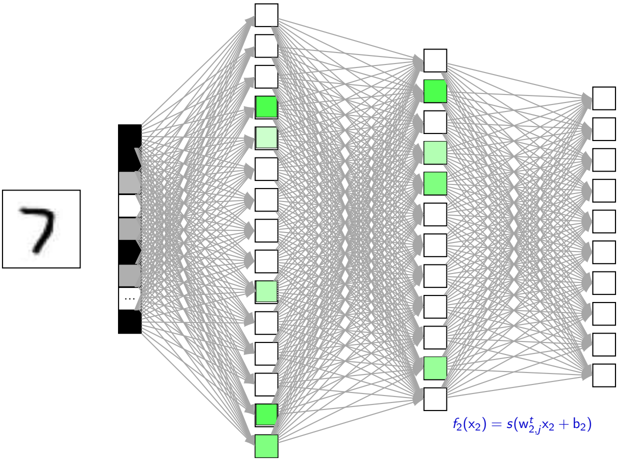
Rede MLP "profunda" com 2 camadas ocultas : Input



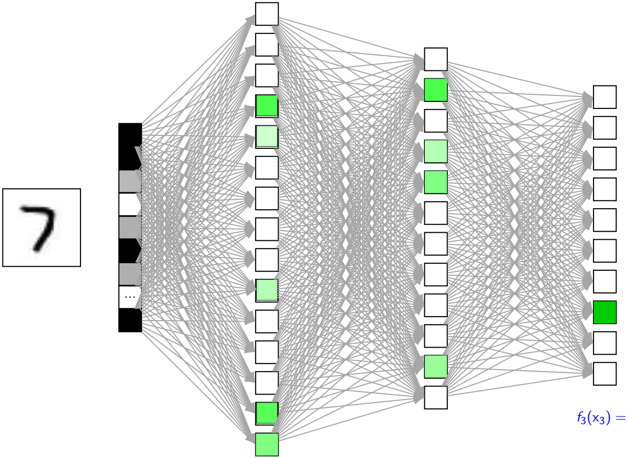
Rede MLP "profunda" com 2 camadas ocultas : Hidden layer 1



Rede MLP "profunda" com 2 camadas ocultas : Hidden layer 2



Rede MLP "profunda" com 2 camadas ocultas : output

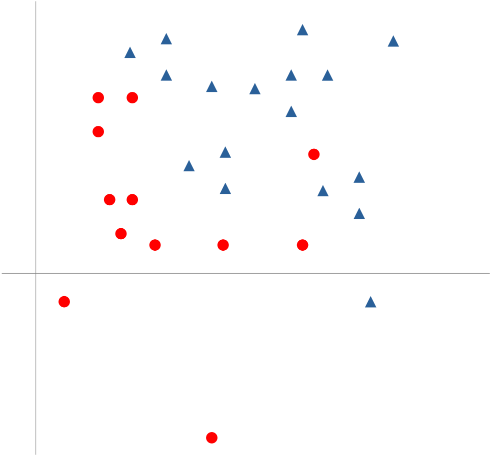


$f_3(x_3) = \text{softmax}(w_{3,j}^t x_3 + b_3)$

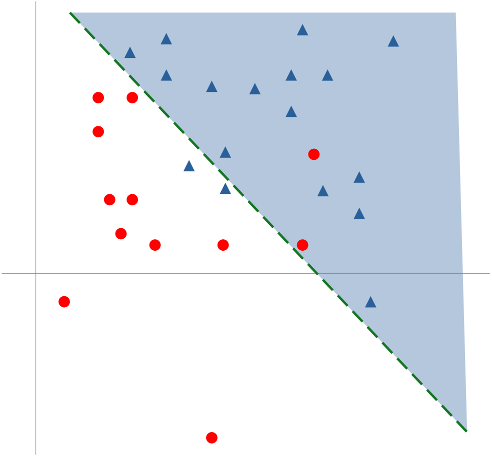
Rede neural Feed-forward

- Recursivamente observa exemplos de entrada e adapta os pesos para todos os parâmetros da rede neural.
- **Feed forward**: todos os neurônios processam a entrada, e computamos a perda
- **Backpropagation**: algoritmo que usa a formulação da regra da cadeia para calcular o gradiente da função de perda, e propaga esse gradiente pelas camadas e neurônios
  - inicia adaptando da última para a primeira camada

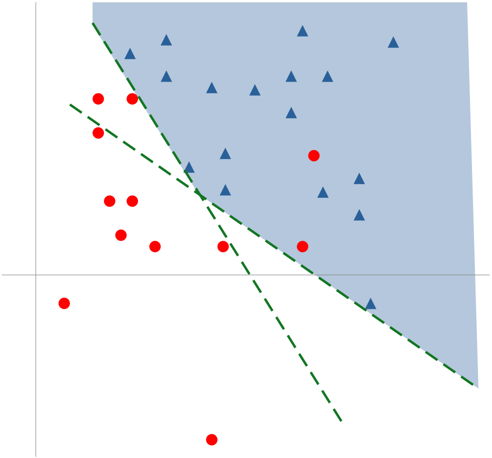
Perceptron Multicamadas: número de neurônios



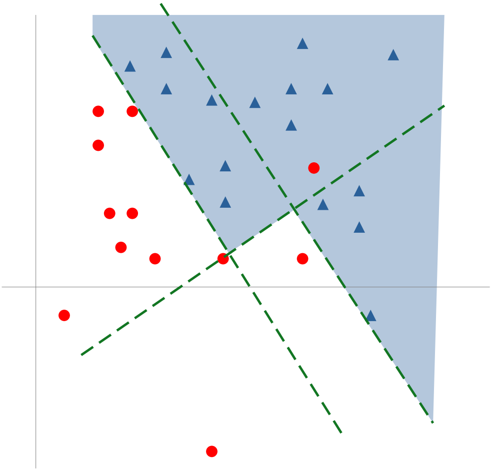
Perceptron Multicamadas: número de neurônios



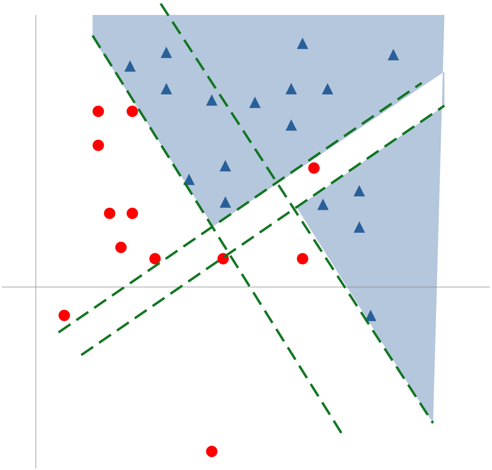
Perceptron Multicamadas: número de neurônios



Perceptron Multicamadas: número de neurônios



Perceptron Multicamadas: número de neurônios



Perda Entropia Cruzada: Entropia

- Quantifica a informação contida nos dados.
- Para uma distribuição P a entropia é:

$$H[P] = \sum_j -P(j) \log P(j)$$

- grosseiramente, o que diz um dos teoremas fundamentais de teoria de informação é que: "para codificar dados amostrados de P precisamos ao menos $H[P]$ bits"

Log-Verossimilhança (*Log-Likelihood*)

- Softmax gera um vetor $\hat{\mathrm{y}}$ com probabilidades condicionais estimadas para cada classe
  - Ex: $\hat{y} = P(y = \mathrm{gato}|\mathrm{x})$
- Podemos comparar $\hat{y}$ com y verificando quão provável as classes reais são com relação à nosso modelo dadas as features (entradas):

$$P(Y|X) = \prod_{i=1}^{n} P(\mathrm{y}^i|\mathrm{x}^i).$$

Log-Verossimilhança (*Log-Likelihood*)

- Assumimos que cada rótulo é amostrado independentemente da sua distribuição $P(\mathrm{y}|\mathrm{x}^i)$
- Como maximizar o produto de termos é pouco conveniente, convertemos em minimização:

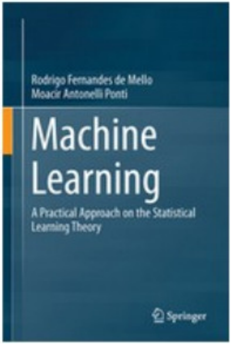
$$-\log P(Y|X) = \sum_{i=1}^{n} -\log P(\mathrm{y}^{(i)}|\mathrm{x}^{(i)}) = \sum_{i=1}^{n} \ell(\mathrm{y}^{(i)}, \hat{\mathrm{y}}^{(i)}).$$

- para cada par de rótulo y e a predição do modelo $\hat{y}$ ao longo de c classes, o custo é:

$$\ell(\mathrm{y}, \hat{\mathrm{y}}) = \sum_{j=1}^{q} -y_j \log \hat{y}_j$$

Bibliografia I

Rodrigo Mello, Moacir A. Ponti. **Machine Learning: a practical approach on the statistical learning theory** Springer, 2018.



Bibliografia II

Moacir A. Ponti, Gabriel Paranhos da Costa. **Como funciona o Deep Learning**
SBC, 2017. Book chapter.
https://arxiv.org/abs/1806.07908

Moacir A. Ponti, Leo Ribeiro, Tiago Nazaré, Tu Bui, John Collomosse. **Everything You Wanted to Know About Deep Learning for Computer Vision but were Afraid to Ask**.
SIBGRAPI-T, 2017. Tutorial.