第五讲
人工神经网络

夏睿
计算机科学与工程学院
南京理工大学
rxia@njust.edu.cn

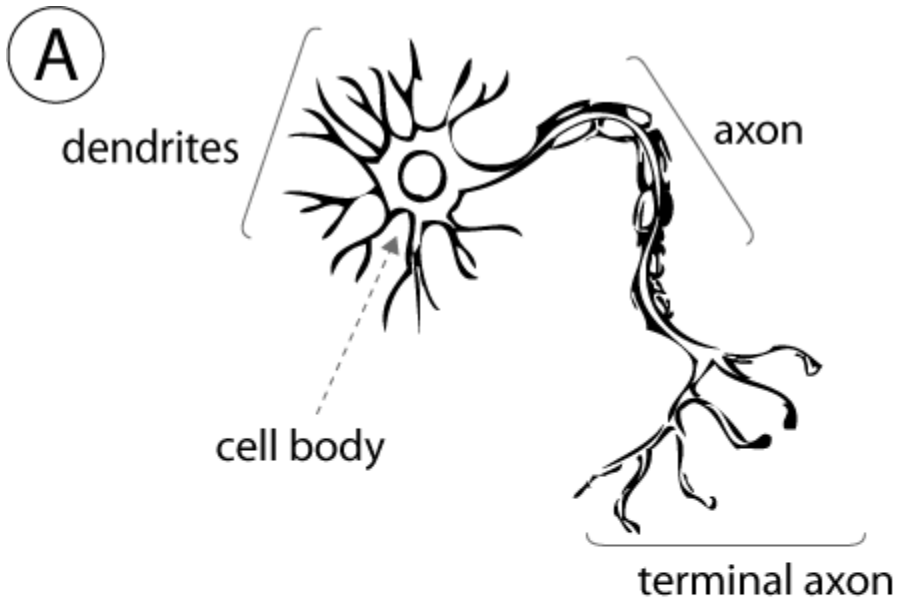
历史简介

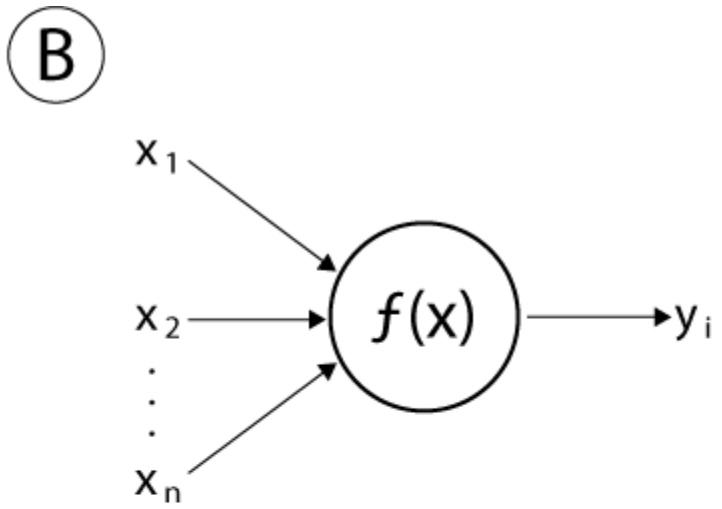
- Rosenblatt（1958）创建了感知器，一种模式识别算法。
- 神经网络研究在Minsky和Papert（1969）的机器学习研究之后停滞不前，他发现了神经网络计算机的两个关键问题。
  - 基本的感知器不能处理异或电路。
  - 计算机没有足够的处理能力来有效地处理大型神经网络所需的工作。
- Paul Werbos（1975）的反向传播算法是对神经网络和学习重新产生兴趣的一个关键触发器。
- 以支持向量机（SVM）为代表的统计学习方法兴起，人工神经网络的研究再次陷入低谷，但仍有学者在此领域持续耕耘。

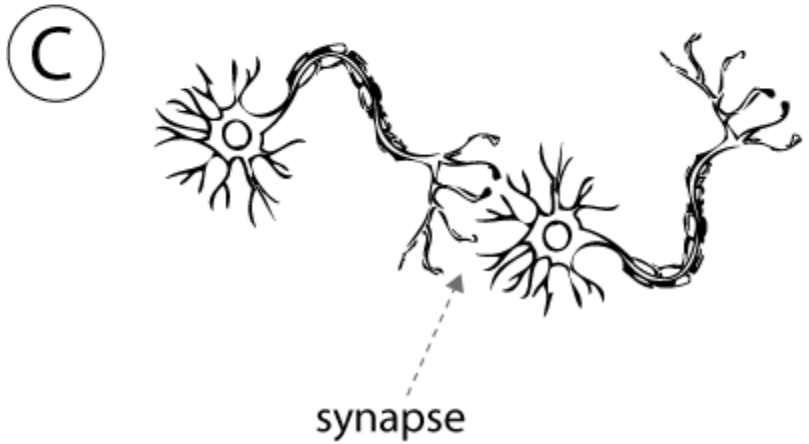
历史简介

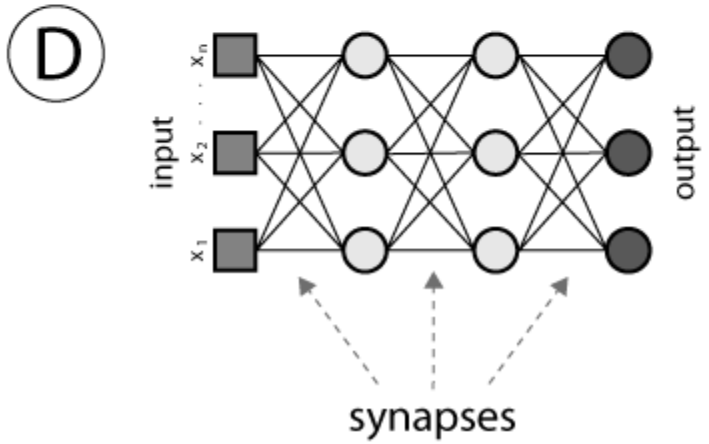
- 2006年，Hinton和Salakhutdinov展示了一个多层前向神经网络如何能够一次有效地预先训练。
- 2009年后，硬件方面的进步为神经网络的重新振兴提供了条件。
- 基于深度学习的语音识别于2010年后开始走向工业。
- 2011 - 2012年，基于深度学习的影像或物体识别产生了重大突破。
- 2013年以后，深度学习方法在许多不同的自然语言处理任务中开始获得很高的性能。
- 目前为止，CNN，RNN，LSTM，GAN等深度学习架构已经应用到很多领域，在这些领域产生了与人类专家相媲美，甚至优于人类专家的成果。

灵感来自神经网络

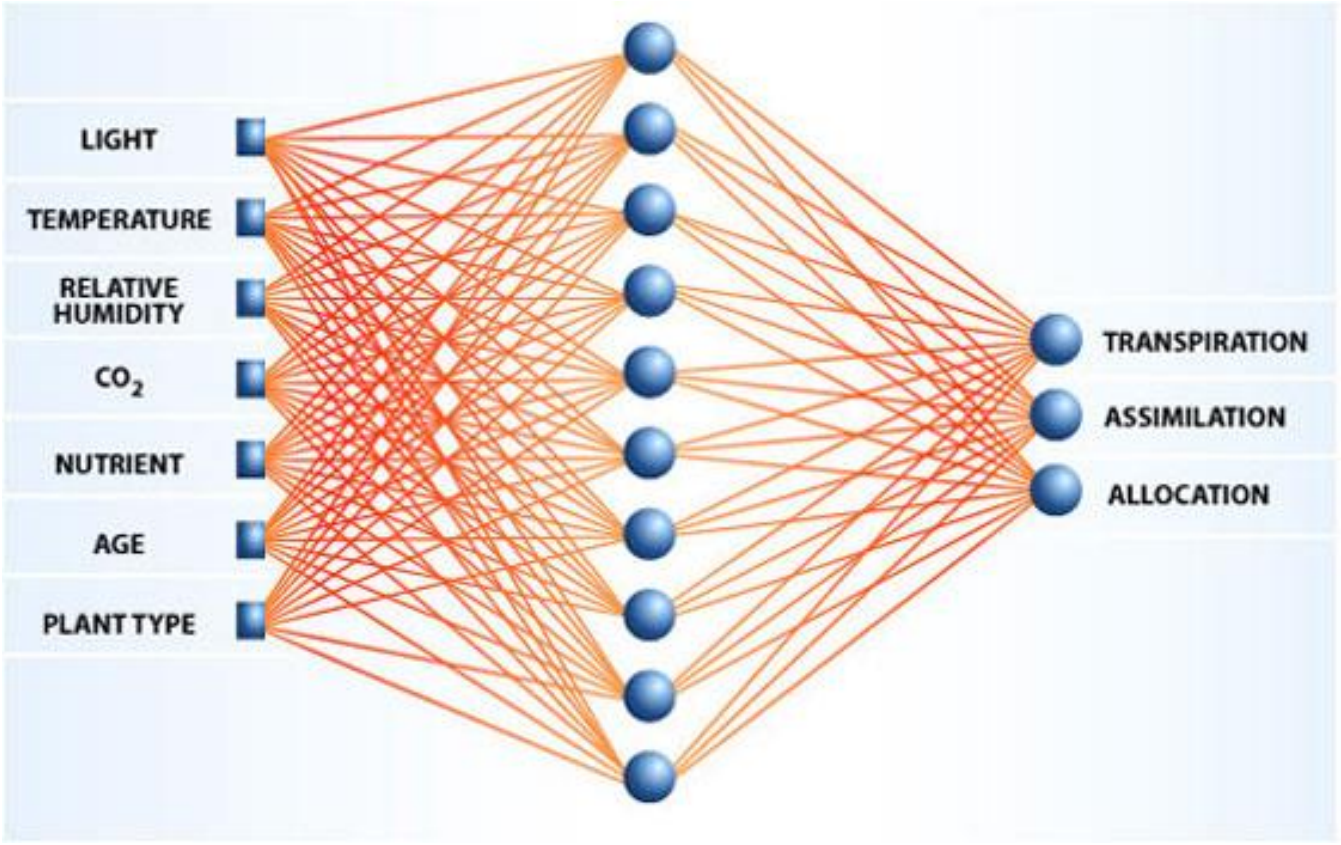

A
dendrites
axon
cell body
terminal axon


B
x 1
x 2
x n
f(x)
y i


C
synapse

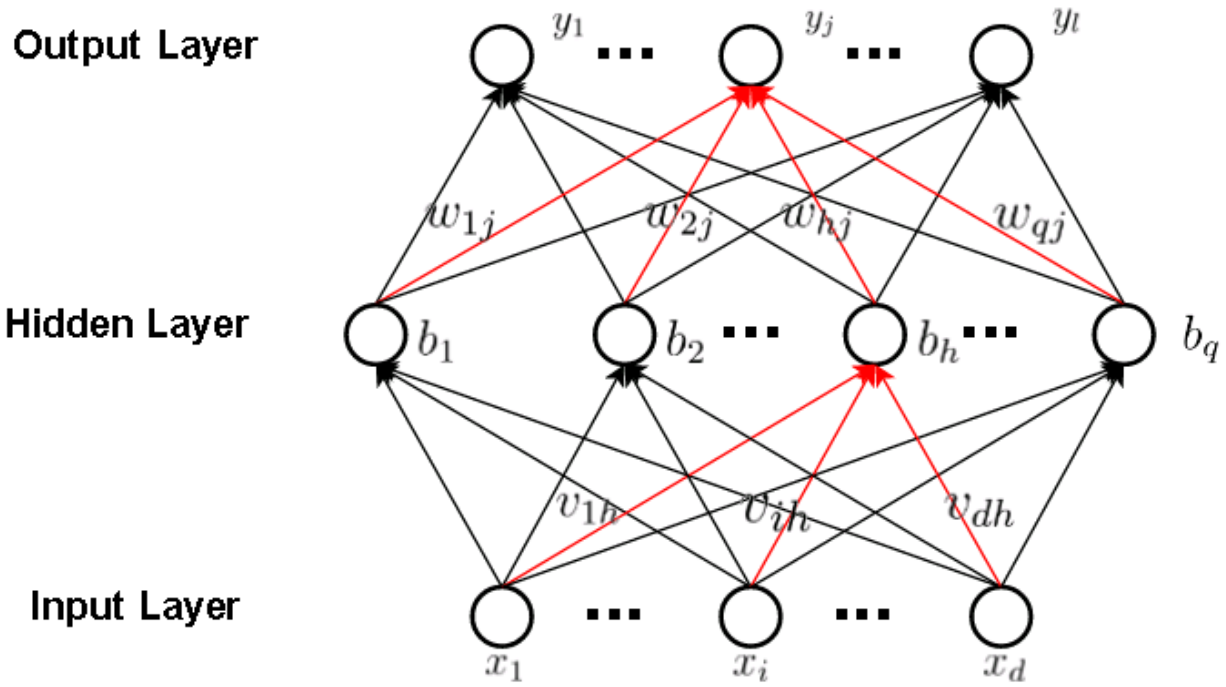

D
input
output
synapses

多层神经网络


LIGHT
TEMPERATURE
RELATIVE HUMIDITY
CO_2
NUTRIENT
AGE
PLANT TYPE
TRANSPIRATION
ASSIMILATION
ALLOCATION

三层前向神经网络

- ANN 结构



- 假设

$$\hat{y}_j = \delta(\beta_j + \theta_j)$$

$$\beta_j = \sum_{h=1}^{q} w_{hj} b_h$$

$$b_h = \delta(\alpha_h + \gamma_h)$$

$$\alpha_h = \sum_{i=1}^{d} v_{ih} x_i$$

学习算法

- 训练集

$$D = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}, \quad x^{(i)} \epsilon R^d, y^{(i)} \epsilon R^l$$

- 代价函数

$$E^{(k)} = \frac{1}{2} \sum_{j=1}^{l} \left(\hat{y}_j^{(k)} - y_j^{(k)} \right)^2$$

- 参数

$$v \epsilon R^{d*q}, \gamma \epsilon R^q, \omega \epsilon R^{q*l}, \theta \epsilon R^l$$

- 梯度计算

$$\frac{\partial E^{(k)}}{\partial v_{ih}}, \frac{\partial E^{(k)}}{\partial \gamma_h}, \frac{\partial E^{(k)}}{\partial \omega_{hj}}, \frac{\partial E^{(k)}}{\partial \theta_j}$$

梯度计算

- 首先，对于ω_{hj}的梯度:

$$\frac{\partial E^{(k)}}{\partial \omega_{hj}} = \frac{\partial E^{(k)}}{\partial \hat{y}_j^{(k)}} \cdot \frac{\partial \hat{y}_j^{(k)}}{\partial(\beta_j + \theta_j)} \cdot \frac{\partial(\beta_j + \theta_j)}{\partial \omega_{hj}}$$

其中，

$$\frac{\partial E^{(k)}}{\partial \hat{y}_j^{(k)}} = \left(\hat{y}_j^{(k)} - y_j^{(k)}\right)$$

$$\frac{\partial \hat{y}_j^{(k)}}{\partial(\beta_j + \theta_j)} = \delta'(\beta_j + \theta_j) = \delta(\beta_j + \theta_j) \cdot \left(1 - \delta(\beta_j + \theta_j)\right) = \hat{y}_j^{(k)} \cdot \left(1 - \hat{y}_j^{(k)}\right)$$

$$\frac{\partial(\beta_j + \theta_j)}{\partial \omega_{hj}} = b_h$$

梯度计算

定义: $$error_j^{OutputLayer} = \frac{\partial E^{(k)}}{\partial(\beta_j + \theta_j)} = \frac{\partial E^{(k)}}{\partial \hat{y}_j^{(k)}} \cdot \frac{\partial \hat{y}_j^{(k)}}{\partial(\beta_j + \theta_j)}$$
$$= \left(\hat{y}_j^{(k)} - y_j^{(k)}\right) \cdot \hat{y}_j^{(k)} \cdot \left(1 - \hat{y}_j^{(k)}\right)$$

然后有: $$\frac{\partial E^{(k)}}{\partial \omega_{hj}} = error_j^{OutputLayer} \cdot b_h$$

- 其次，对于θ_j的梯度:

$$\frac{\partial E^{(k)}}{\partial \theta_j} = \frac{\partial E^{(k)}}{\partial \hat{y}_j^{(k)}} \cdot \frac{\partial \hat{y}_j^{(k)}}{\partial(\beta_j + \theta_j)} \cdot \frac{\partial(\beta_j + \theta_j)}{\partial \theta_j}$$
$$= error_j^{OutputLayer} \cdot 1$$

梯度计算

- 第三，对于v_{ih}的梯度:

$$\frac{\partial E^{(k)}}{\partial v_{ih}} = \sum_{j=1}^{l} \frac{\partial E^{(k)}}{\partial(\beta_j + \theta_j)} \cdot \frac{\partial(\beta_j + \theta_j)}{\partial b_h} \cdot \frac{\partial b_h}{\partial(\alpha_h + \gamma_h)} \cdot \frac{\partial(\alpha_h + \gamma_h)}{\partial v_{ih}}$$

其中，

$$\frac{\partial E^{(k)}}{\partial(\beta_j + \theta_j)} = error_j^{OutputLayer}$$

$$\frac{\partial(\beta_j + \theta_j)}{\partial b_h} = \omega_{hj}$$

$$\frac{\partial b_h}{\partial(\alpha_h + \gamma_h)} = \delta'(\alpha_h + \gamma_h) = \delta(\alpha_h + \gamma_h) \cdot \big(1 - \delta(\alpha_h + \gamma_h)\big) = b_h \cdot (1 - b_h)$$

$$\frac{\partial(\alpha_h + \gamma_h)}{\partial v_{ih}} = x_i^{(k)}$$

梯度计算

定义：

$$error_h^{HiddenLayer} = \frac{\partial E^{(k)}}{\partial(\alpha_h + \gamma_h)}$$

$$= \sum_{j=1}^{l} \frac{\partial E^{(k)}}{\partial(\beta_j + \theta_j)} \cdot \frac{\partial(\beta_j + \theta_j)}{\partial b_h} \cdot \frac{\partial b_h}{\partial(\alpha_h + \gamma_h)}$$

$$= \sum_{j=1}^{l} error_j^{OutputLayer} \cdot \omega_{hj} \cdot \delta'(\alpha_h + \gamma_h)$$

$$= \sum_{j=1}^{l} error_j^{OutputLayer} \cdot \omega_{hj} \cdot b_h \cdot (1 - b_h)$$

然后：

$$\frac{\partial E^{(k)}}{\partial v_{ih}} = error_h^{HiddenLayer} \cdot x_i^{(k)}$$

梯度计算

- 最后，对于 γ_h 的梯度:

$$\frac{\partial E^{(k)}}{\partial \gamma_h} = \sum_{j=1}^{l} \frac{\partial E^{(k)}}{\partial (\beta_j + \theta_j)} \cdot \frac{\partial (\beta_j + \theta_j)}{\partial b_h} \cdot \frac{\partial b_h}{\partial (\alpha_h + \gamma_h)} \cdot \frac{\partial (\alpha_h + \gamma_h)}{\partial \gamma_h}$$

$$= error_h^{HiddenLayer} \cdot 1$$

反向传播算法

梯度更新

$$\omega_{hj} := \omega_{hj} - \eta \cdot \frac{\partial E^{(k)}}{\partial \omega_{hj}}$$

$$\theta_j := \theta_j - \eta \cdot \frac{\partial E^{(k)}}{\partial \theta_j}$$

$$v_{ih} := v_{ih} - \eta \cdot \frac{\partial E^{(k)}}{\partial v_{ih}}$$

$$\gamma_h := \gamma_h - \eta \cdot \frac{\partial E^{(k)}}{\partial \gamma_h}$$

其中 η 是学习率

算法流程图

Input: training set: $\mathcal{D} = \{(x^{(k)}, y^{(k)})\}_{k=1}^{m}$

learning rate η

Steps：

1: initialize all parameters within (0,1)

2: repeat:

3: for all $(x^{(k)}, y^{(k)}) \in \mathcal{D}$ do:

4: calculate $y^{(k)}$

5: calculate $error^{OutputLayer}$:

6: calculate $error^{HiddenLayer}$:

7: update v , θ , v and γ

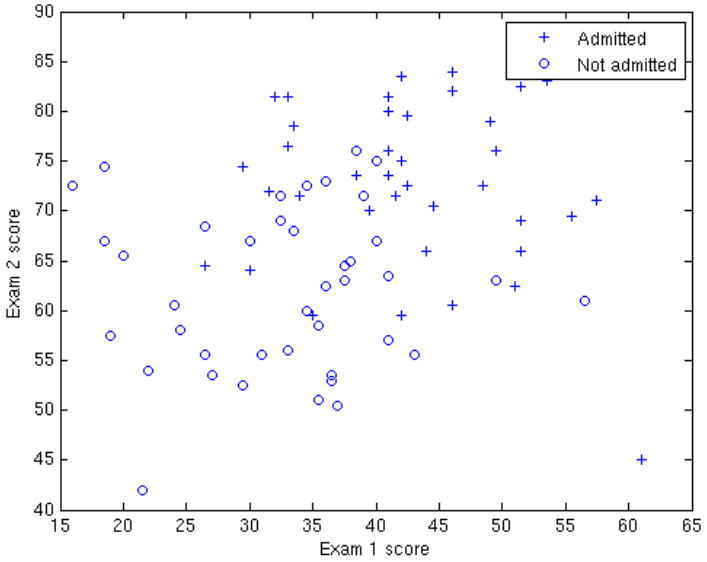
8: end for

9: until reach stop condition

Output: trained ANN

练习#1：基于BP算法的三层前向神经网络

- 给出训练数据:

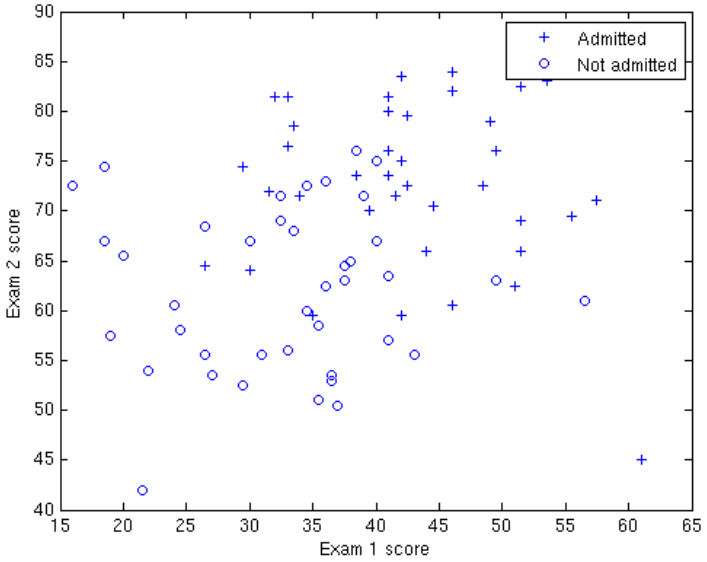


http://openclassroom.stanford.edu/MainFolder/DocumentPage.php?course=DeepLearning&doc=exercises/ex4/ex4.html

- 使用BP算法实现三层前向神经网络 (自己编码，不要使用TensorFlow)，并对结果进行5倍交叉验证;
- 将其与逻辑回归和softmax回归进行比较。.

练习 #2:基于BP算法的三层前向神经网络

- 给出训练数据：



http://openclassroom.stanford.edu/MainFolder/DocumentPage.php?course=DeepLearning&doc=exercises/ex4/ex4.html

- 使用BP算法实现三层前向神经网络，并对结果进行5倍交叉验证(自己编码);
- 使用 Tensorflow完成上面的要求
- 通过使用不同数量的隐藏层和隐藏节点，不同的激活函数，不同的代价函数，不同的学习率来调整模型。

欢迎提问！