Lecture 4
Artificial Neural Networks

Rui Xia
Text **M**ining Group
Nanjing **U**niversity of **S**cience & **T**echnology
rxia@njust.edu.cn

Brief History

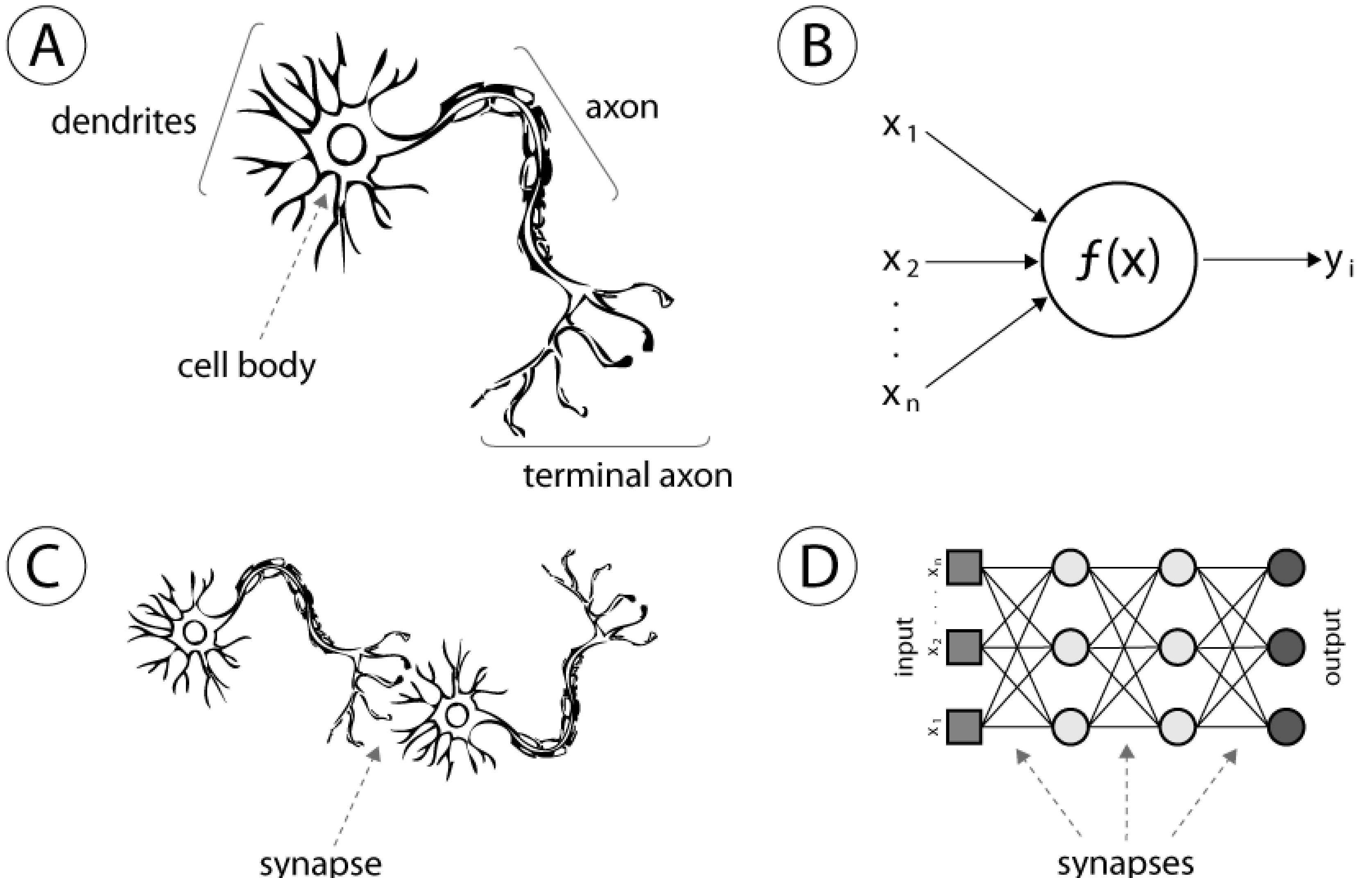
- Rosenblatt (1958) created the perceptron, an algorithm for pattern recognition.
- Neural network research stagnated after machine learning research by Minsky and Papert (1969), who discovered two key issues with the computational machines that processed neural networks.
  - Basic perceptrons were incapable of processing the exclusive-or circuit.
  - Computers didn't have enough processing power to effectively handle the work required by large neural networks.
- A key trigger for the renewed interest in neural networks and learning was Paul Werbos's (1975) **back-propagation** algorithm.
- Both shallow and deep learning (e.g., recurrent nets) of ANNs have been explored for many years.

Brief History

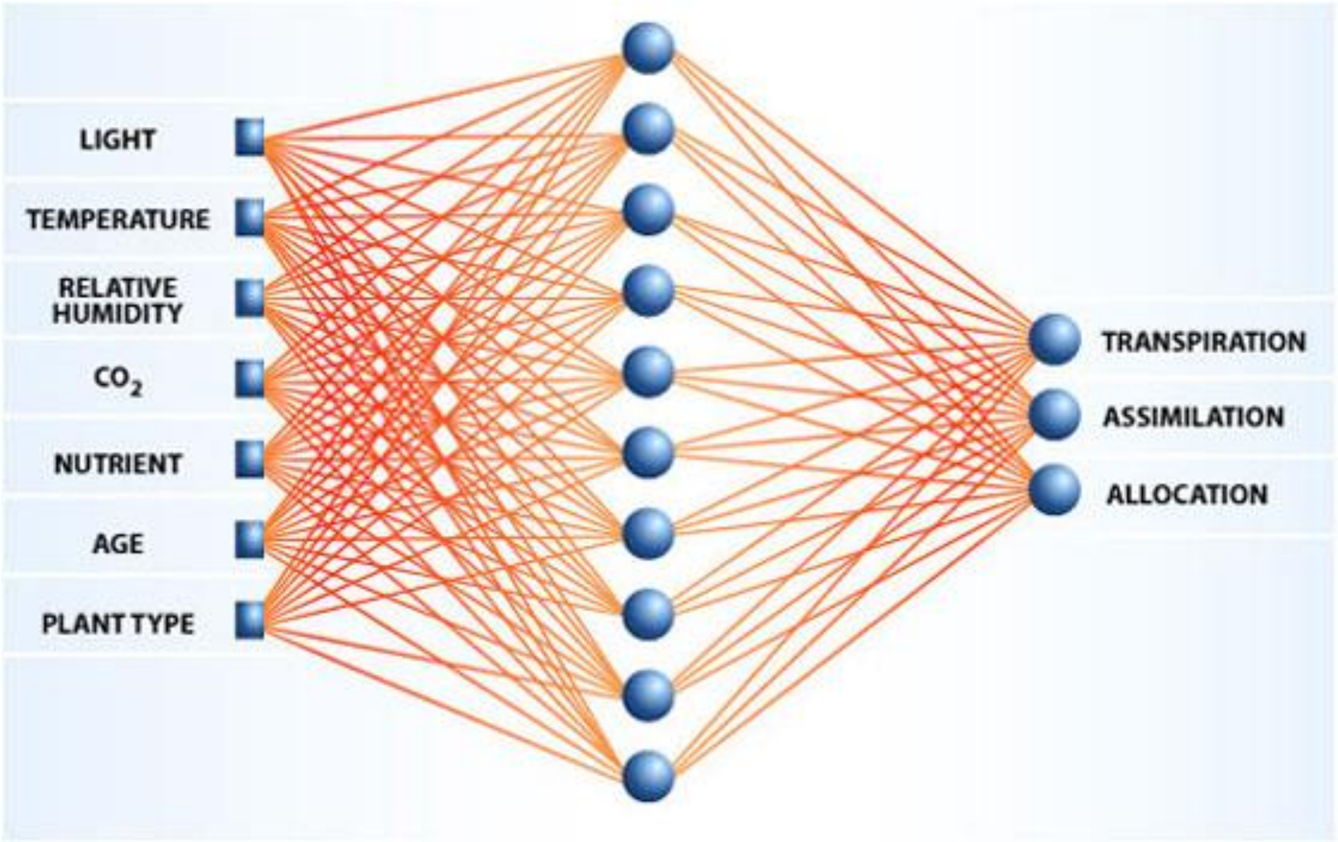
- In 2006, Hinton and Salakhutdinov showed how a many-layered feedforward neural network could be effectively pre-trained one layer at a time.
- Advances in hardware enabled the renewed interest after 2009.
- Industrial applications of deep learning to large-scale speech recognition started around 2010.
- Significant additional impacts in image or object recognition were felt from 2011–2012.
- Deep learning approaches have obtained very high performance across many different natural language processing tasks after 2013.
- Till now, deep learning architectures such as CNN, RNN, LSTM, GAN have been applied to a lot of fields, where they produced results comparable to and in some cases superior to human experts.

NUSTM

Inspired from Neural Networks



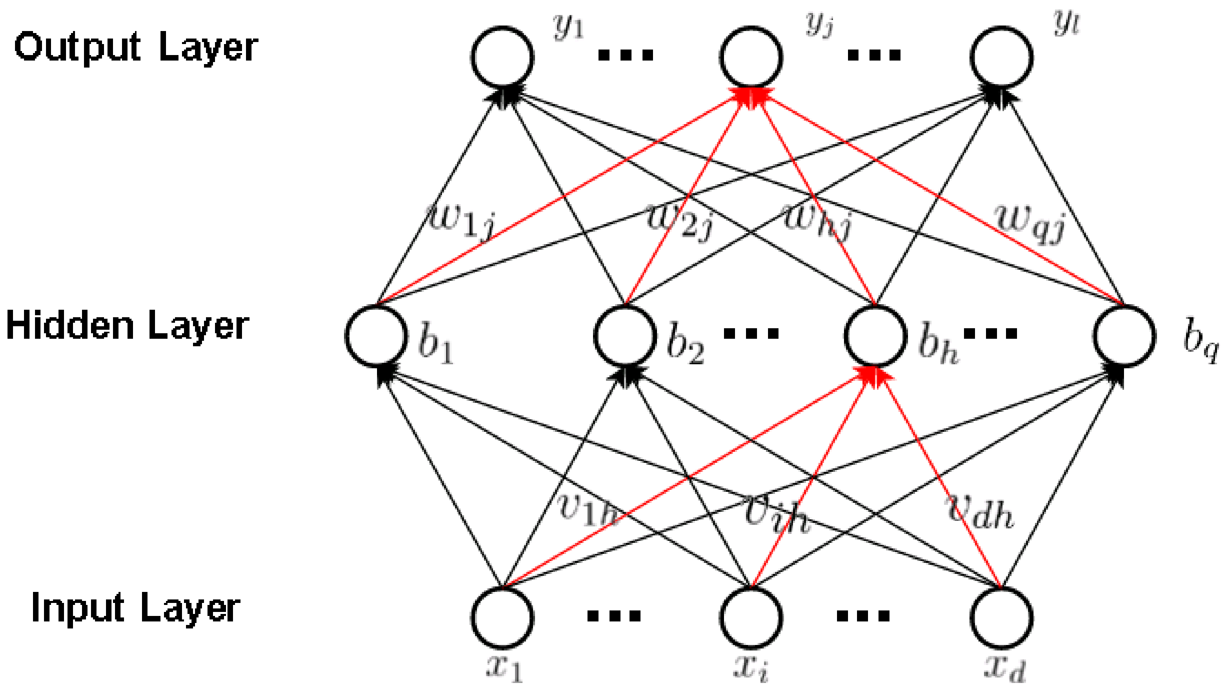
Multi-layer Neural Networks



NUSTM

3-layer Forward Neural Networks

- ANN Structure



- Hypothesis

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

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

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

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

Learning algorithm

- Training Set

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

- Cost function

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

- Parameters

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

- Gradients to calculate

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

Gradient Calculation

- Firstly, gradient with respect to ω_{hj}:

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

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

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

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

Gradient Calculation

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

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

- Secondly, gradient with respect to θ_j:

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

Gradient Calculation

- Thirdly, gradient with respect to v_{ih}:

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

where,

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

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

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

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

Gradient Calculation

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

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

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

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

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

Gradient Calculation

- Finally, gradient with respect to γ_h:

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

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

NUSTM

Back propagation algorithm

weight updating

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

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

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

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

where η is the learning rate

algorithm flowchart

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

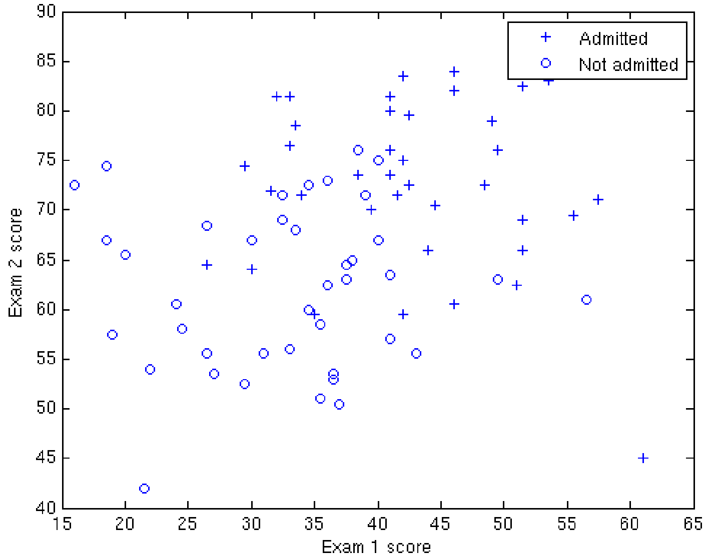
Steps：

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

Practice: 3-layer Forward NN with BP

- Given the following training data:

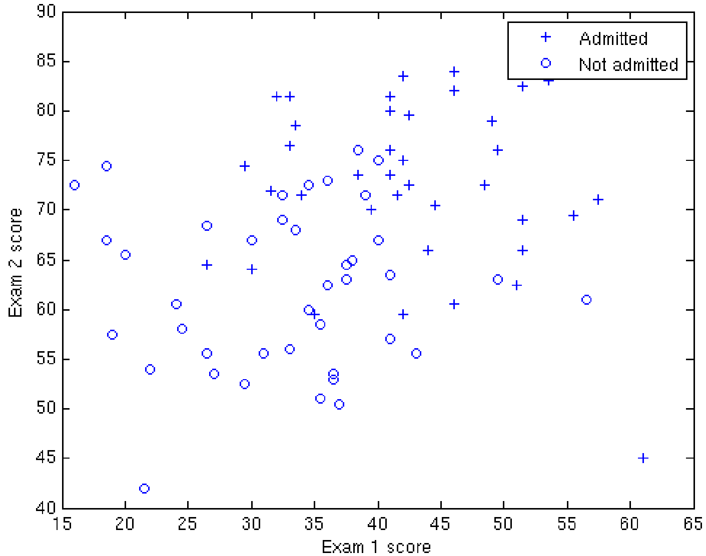


http://openclassroom.stanford.edu/MainFolder/DocumentPage.php?course=DeepLearning&doc=exercises/ex4/ex4.html

- Implement 3-layer Forward Neural Network with Back-Propagation and report the 5-fold cross validation performance (code by yourself, don't use Tensorflow);
- Compare it with logistic regression and softmax regression.

Practice #2: 3-layer Forward NN with BP

- Given the following training data:



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

- Implement multi-layer Forward Neural Network with Back-Propagation and report the 5-fold cross validation performance (code by yourself);
- Do that again (by using Tensorflow)
- Tune the model by using different numbers of hidden layers and hidden nodes, different activation functions, different cost functions, different learning rates.

Questions?