Deep Neural Network

Ting Wu

Deep Learning Platform of IFLYTEK

outline

- **Application**
- **Automatic Speech Recognition**
- **Multi Layer Perceptron**
- **Deep Neural Network**

Application

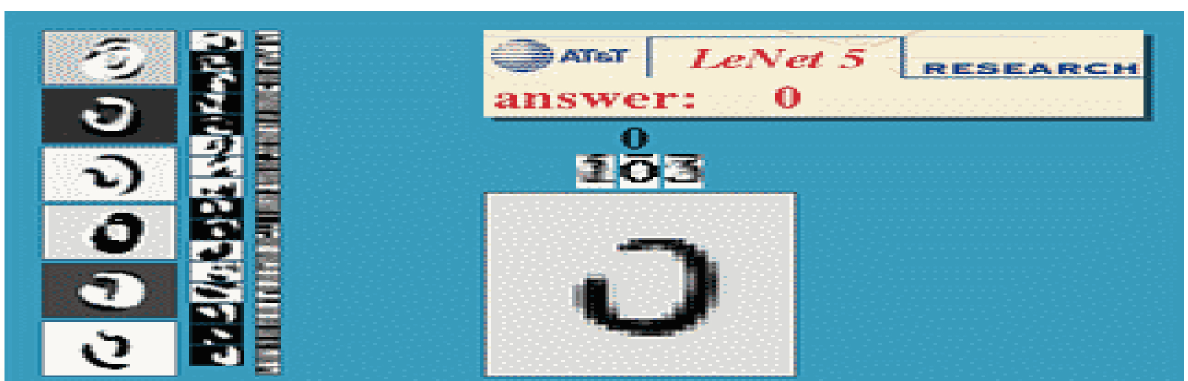
Image Analysis

face recognition

graph-text transformation

A group of **people** sitting on a boat in the water.

handwriting recognition



Speech Analysis

科大讯飞 IFLYTEK

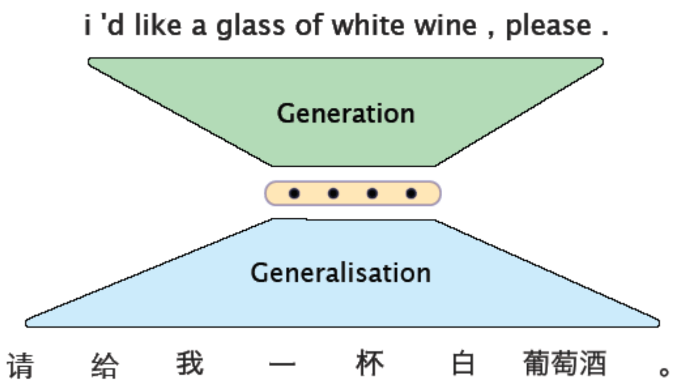
query by humming

automation speech recognition

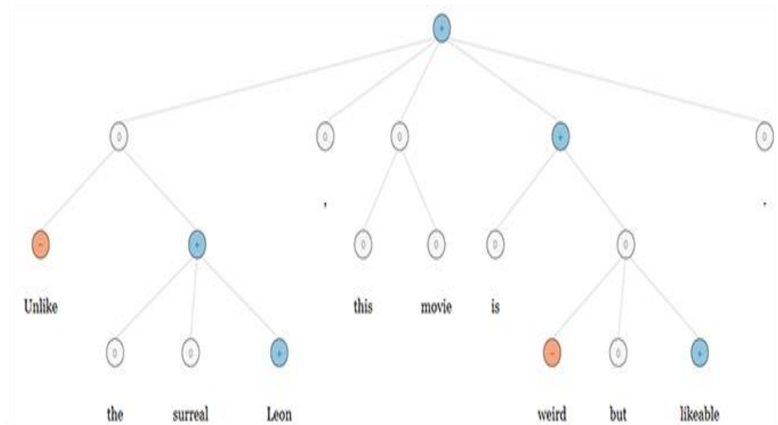
oral testing

Natural Language Processing

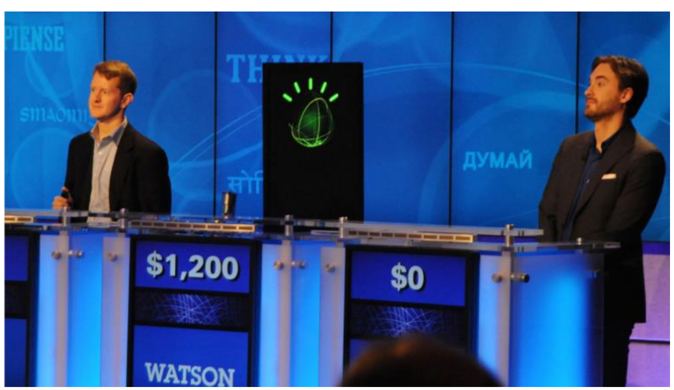
machine translation



sentiment analysis



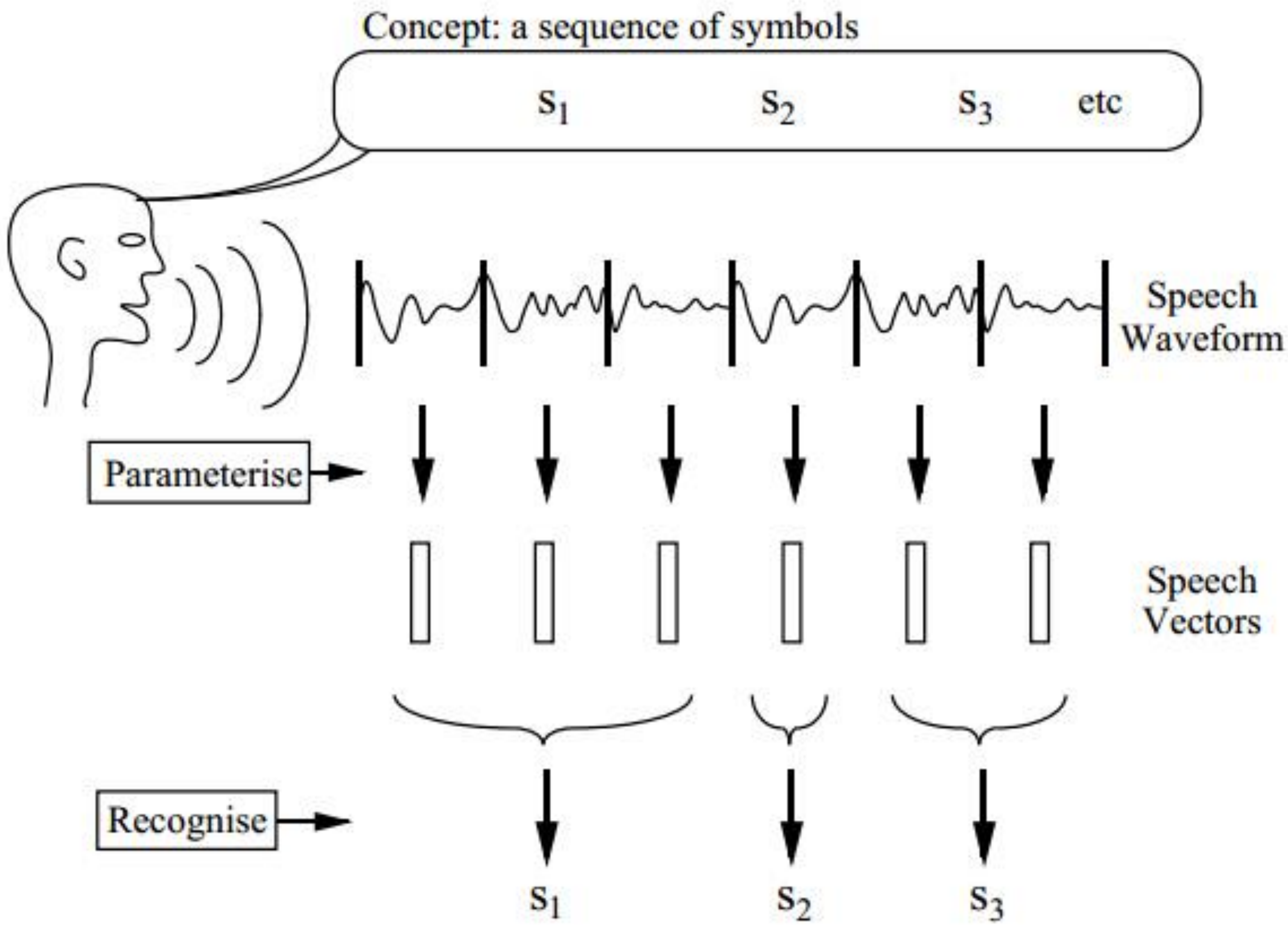
Q&A



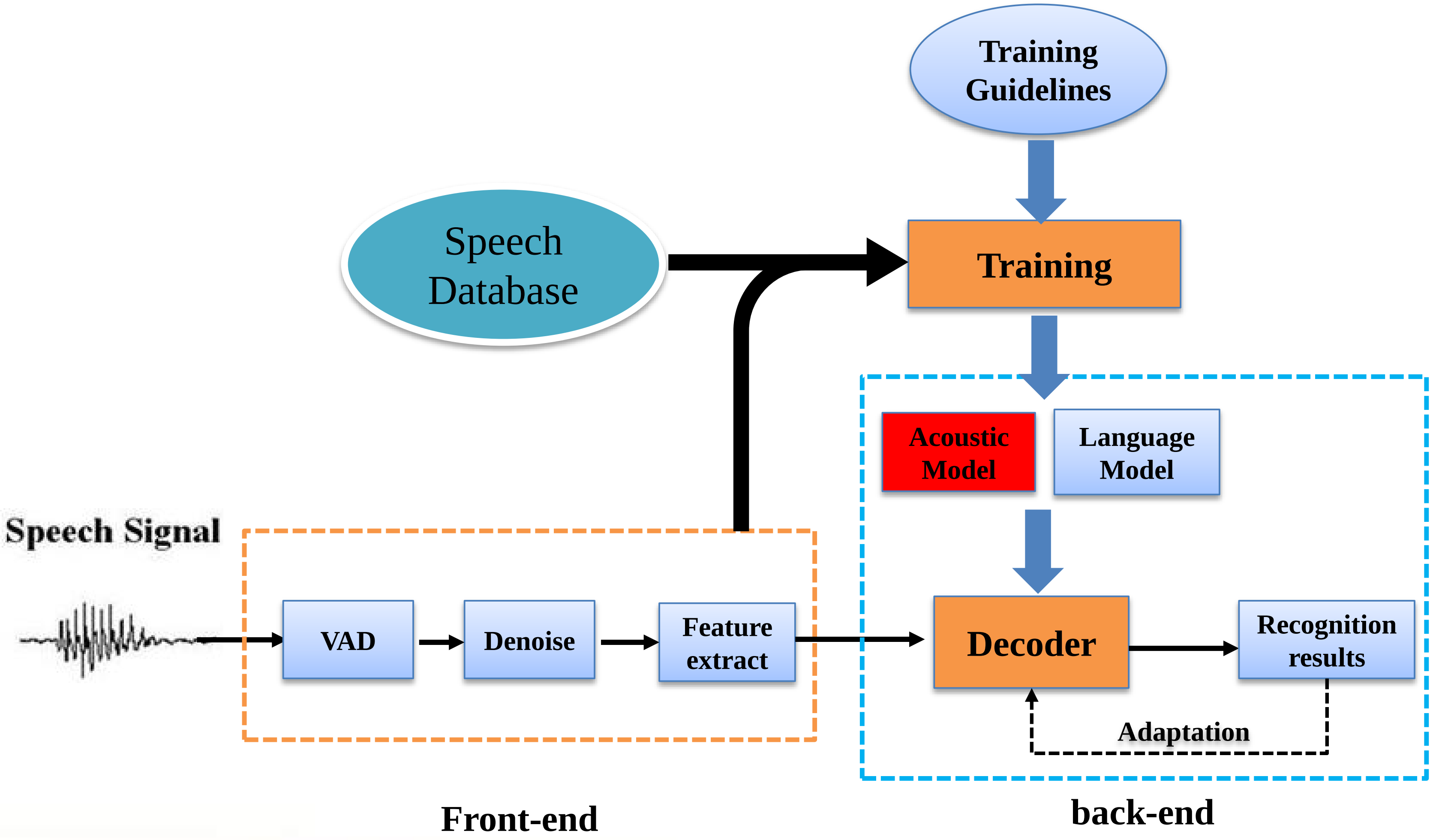
recommendation

Automation Speech Recognition

Automatic Speech Recognition

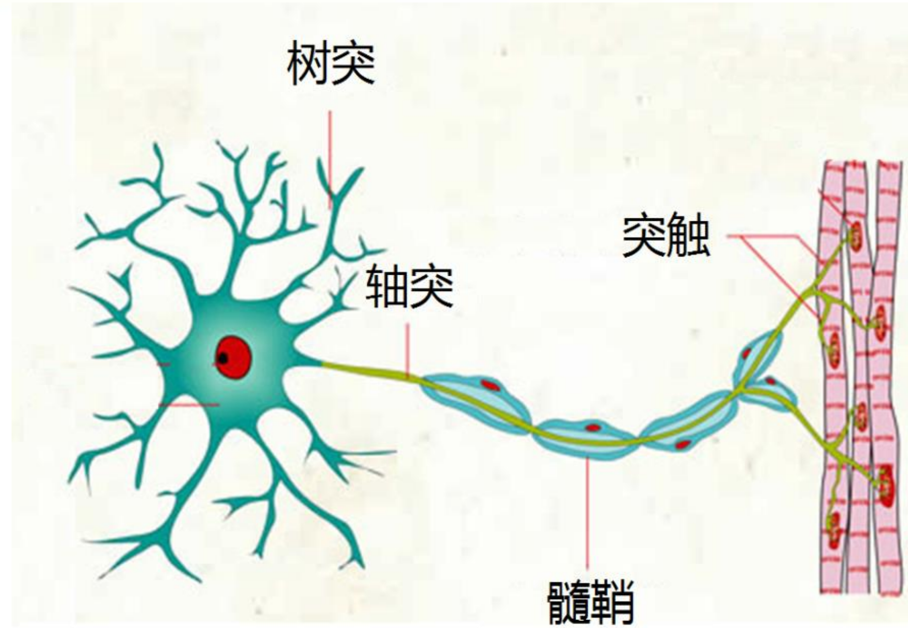


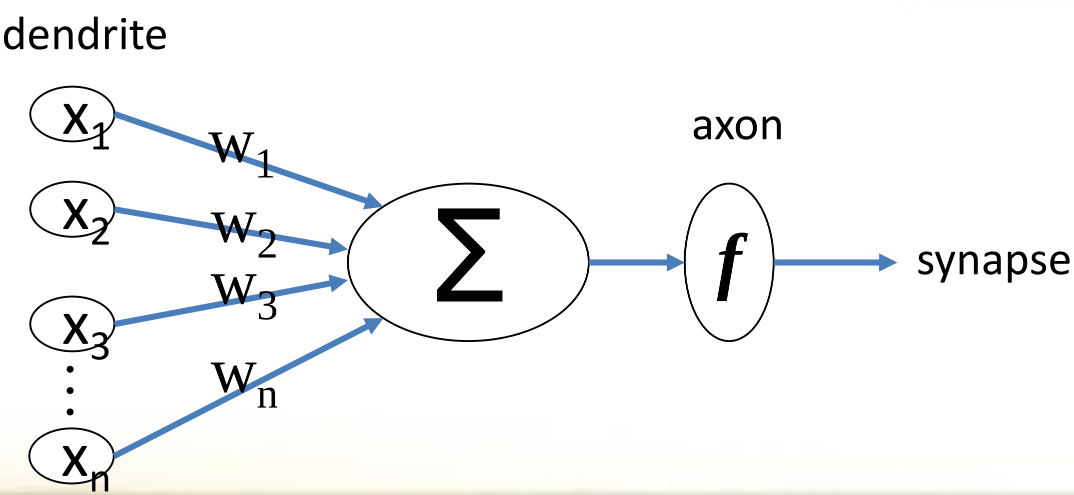
Automatic Speech Recognition



Multi-layer Perceptron

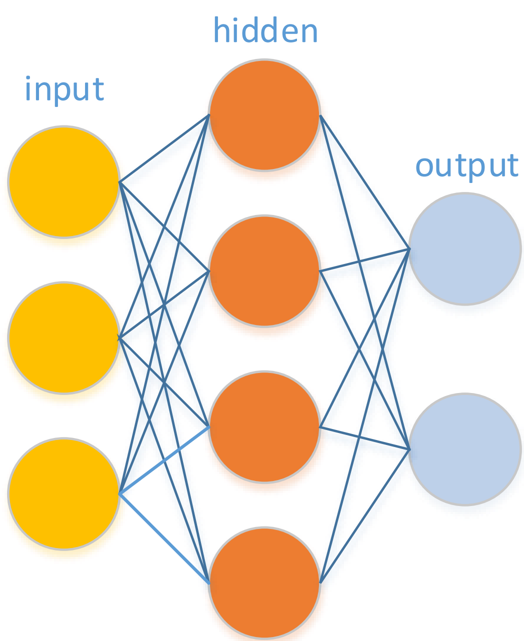
Perceptron





$$y = f\left(\sum_{i=1}^{n} x_i w_i + b\right)$$

MLP



Kolmogonov proves that by appropriately selecting function *G* and *F*, any continuous function *y(x)* can be expressed as

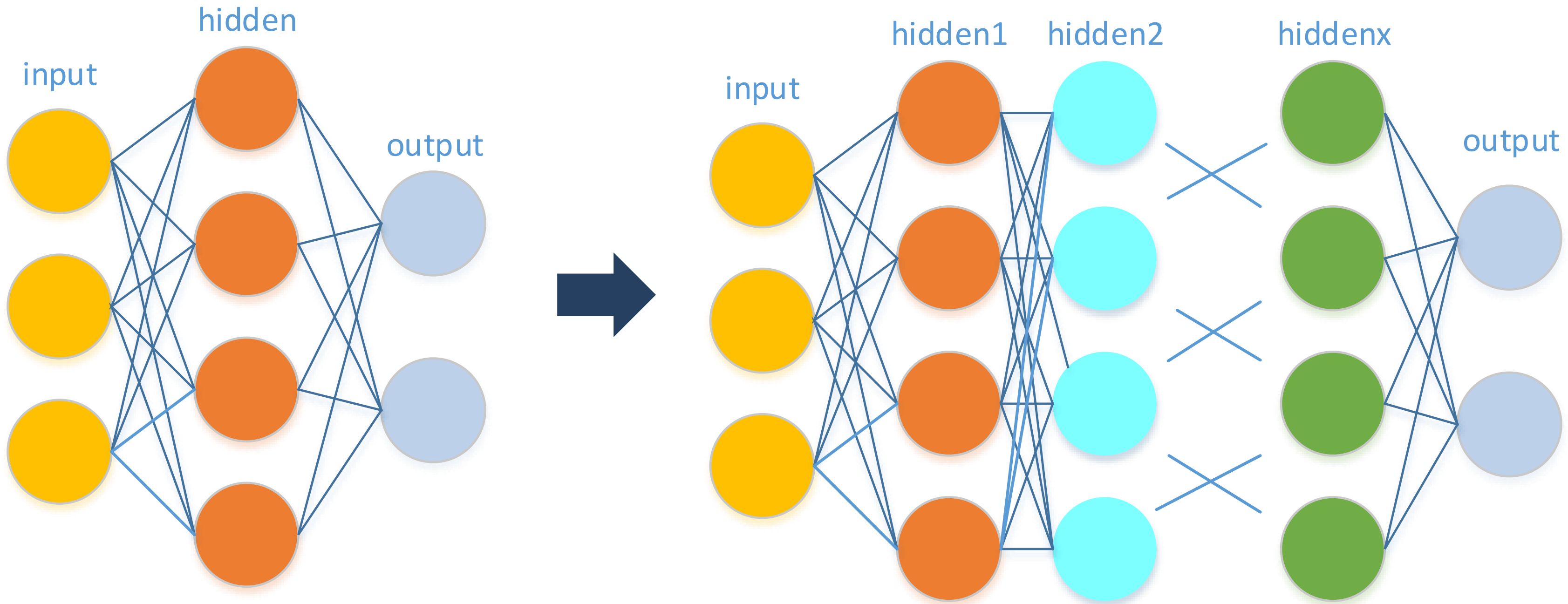
$$y(x) = \sum_j G_j \left(\sum_i F_{ji}(x_i) \right)$$

Discriminant function of MLP can be expressed as

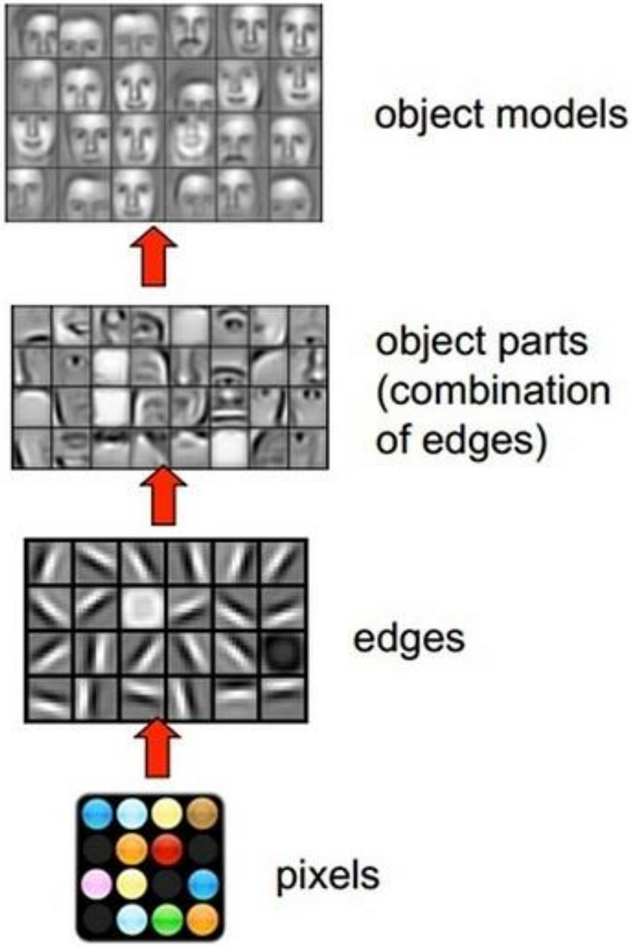
$$g_k(x) = z_k = g\left(\sum_j w_{kj} f\left(\sum_i w_{ji} x_i + w_{j0} \right) + w_{k0} \right)$$

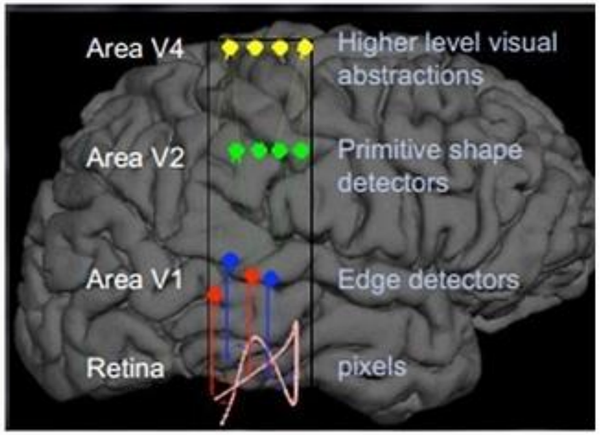
So, MLP can express any continuous function if we choose function *G* and *F* carefully and use many enough hidden units

MLP



MLP

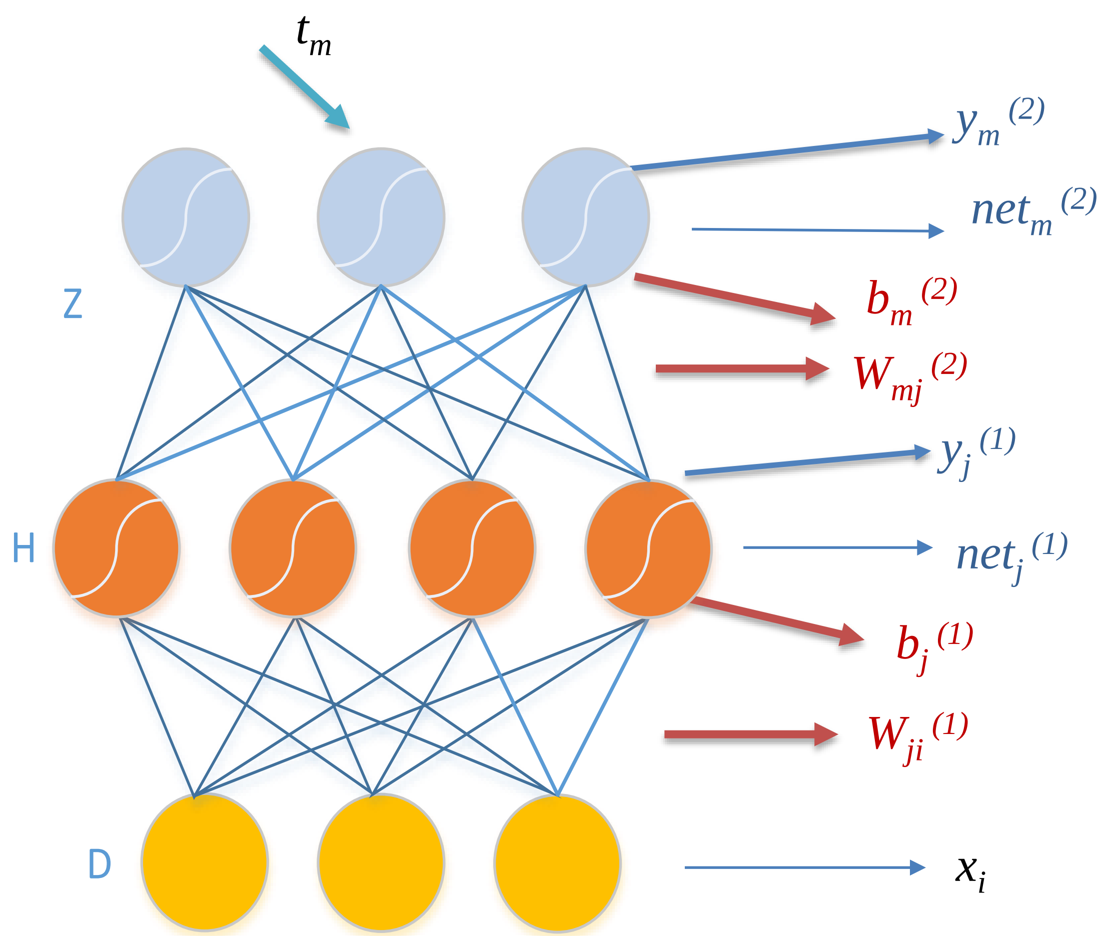




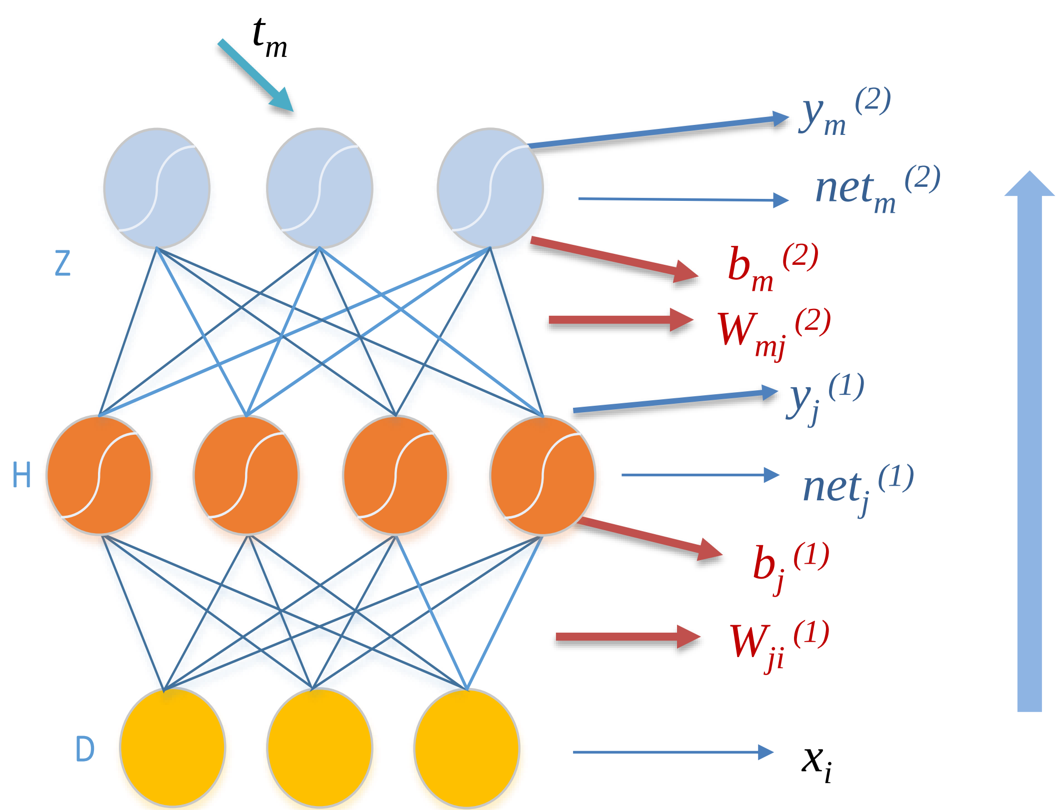
Cognize gradually, abstract progressively

Deep Neural Network

Deep Neural Network(DNN)



Forward



Hidden layer

$$net_j^{(1)} = \sum_{i=1}^{D} x_i w_{ji}^{(1)} + b_j^{(1)}$$

$$y_j^{(1)} = f(net_j^{(1)})$$ → f为隐层激活函数

Output layer

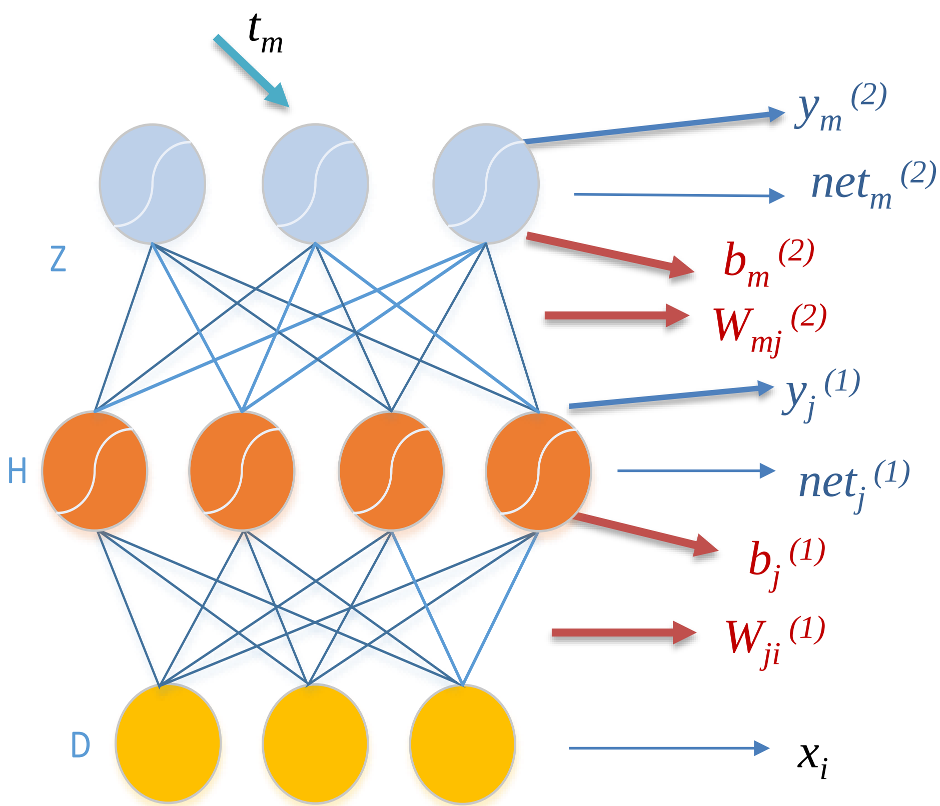
$$net_m^{(2)} = \sum_{j=1}^{H} y_j^{(1)} w_{mj}^{(2)} + b_m^{(2)}$$

$$y_m^{(2)} = g(net_m^{(2)})$$ → g为输出层激活函数

Network error

$$J = \frac{1}{2}\sum_{m=1}^{c}(t_m - y_m^{(2)})^2$$

Backward



$$SensitivityOut: \quad e_s^{k\uparrow} = \frac{\partial J}{\partial y_s^k}$$

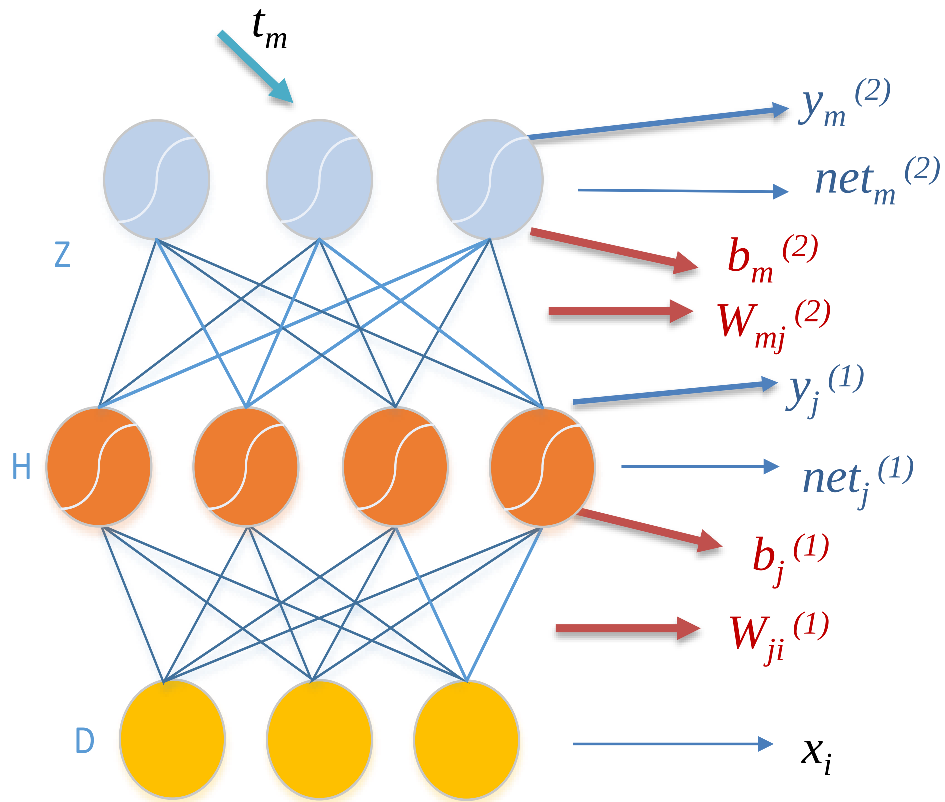
$$SensitivityIn: \quad e_s^{k\downarrow} = \frac{\partial J}{\partial net_s^k}$$

$$y_s^k = f(net_s^k)$$

So $$e_s^{k\downarrow} = \frac{\partial J}{\partial net_s^k} = \frac{\partial J}{\partial y_s^k} \cdot \frac{\partial y_s^k}{\partial net_s^k}$$
$$= e_s^{k\uparrow} \cdot f'(net_s^k)$$

$$SensitivityIn = SensitivityOut * f'$$

Backward



$$J = \frac{1}{2}\sum_{m=1}^{c}(t_m - y_m^{(2)})^2$$

output layer Sensitivity

$$e_m^{(2)\uparrow} = \frac{\partial J}{\partial y_m^{(2)}} = -(t_m - y_m^{(2)})$$

$$e_m^{(2)\downarrow} = \frac{\partial J}{\partial net_m^{(2)}} = \frac{\partial J}{\partial y_m^{(2)}} \cdot \frac{\partial y_m^{(2)}}{\partial net_m^{(2)}}$$

$$= e_m^{(2)\uparrow} \cdot f'(net_m^{(2)})$$

$$= -(t_m - y_m^{(2)}) \cdot f'(net_m^{(2)})$$

hidden layer Sensitivity

$$e_j^{(1)\uparrow} = \frac{\partial J}{\partial y_j^{(1)}} = \frac{\partial J}{\partial net_m^{(2)}} \cdot \frac{\partial net_m^{(2)}}{\partial y_j^{(1)}} = e_m^{(2)\downarrow} \cdot w_{mj}^{(2)}$$

$$e_j^{(1)\downarrow} = \frac{\partial J}{\partial net_j^{(1)}} = \frac{\partial J}{\partial y_j^{(1)}} \cdot \frac{\partial y_j^{(1)}}{\partial net_j^{(1)}}$$

$$= e_j^{(1)\uparrow} \cdot f'(net_j^{(1)})$$

Backward

output layer Sensitivity

$$e_m^{(2)\uparrow} = \frac{\partial J}{\partial y_m^{(2)}} = -(\mathrm{t}_m - y_m^{(2)})$$

$$e_m^{(2)\downarrow} = \frac{\partial J}{\partial net_m^{(2)}} = -(\mathrm{t}_m - y_m^{(2)}) \cdot f'(net_m^{(2)})$$

output layer gradient

$$\Delta w_{mj}^{(2)} = -\eta \frac{\partial J}{\partial w_{mj}^{(2)}} = -\eta \frac{\partial J}{\partial net_m^{(2)}} \cdot \frac{\partial net_m^{(2)}}{\partial w_{mj}^{(2)}}$$

$$= -\eta e_m^{(2)\downarrow} \cdot y_j^{(1)}$$

$$\Delta b_m^{(2)} = -\eta \frac{\partial J}{\partial b_m^{(2)}} = -\eta \frac{\partial J}{\partial net_m^{(2)}} \cdot \frac{\partial net_m^{(2)}}{\partial b_m^{(7)}}$$

$$= -\eta e_m^{(2)\downarrow}$$

hidden layer Sensitivity

$$e_j^{(1)\uparrow} = \frac{\partial J}{\partial y_j^{(1)}} = e_m^{(2)\downarrow} \cdot w_{mj}^{(2)}$$

$$e_j^{(1)\downarrow} = \frac{\partial J}{\partial net_j^{(1)}} = e_j^{(1)\uparrow} \cdot f'(net_j^{(1)})$$

hidden layer gradient

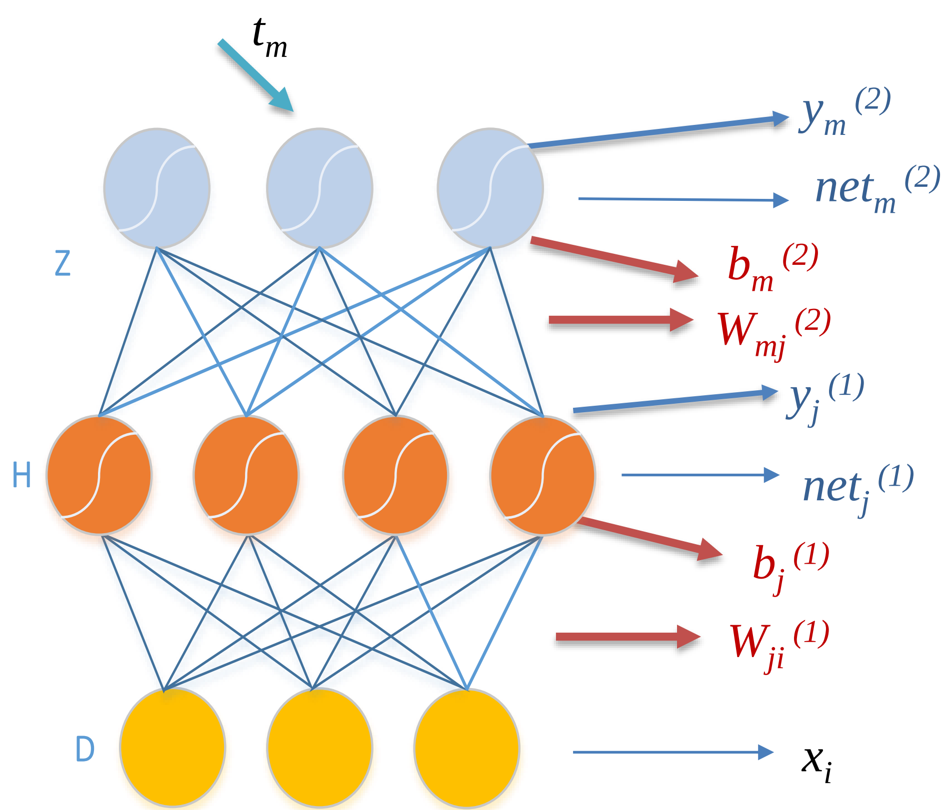
$$\Delta w_{ji}^{(1)} = -\eta \frac{\partial J}{\partial w_{ji}^{(1)}} = -\eta \frac{\partial J}{\partial net_j^{(1)}} \cdot \frac{\partial net_j^{(1)}}{\partial w_{ji}^{(1)}}$$

$$= -\eta e_j^{(1)\downarrow} \cdot x_i$$

$$\Delta b_j^{(1)} = -\eta \frac{\partial J}{\partial b_j^{(1)}} = -\eta \frac{\partial J}{\partial net_j^{(1)}} \cdot \frac{\partial net_j^{(1)}}{\partial b_j^{(1)}}$$

$$= -\eta e_j^{(1)\downarrow}$$

Update



output layer update

$$w_{mj}^{(2)} = w_{mj}^{(2)} + \Delta w_{mj}^{(2)}$$
$$b_m^{(2)} = b_m^{(2)} + \Delta b_m^{(2)}$$

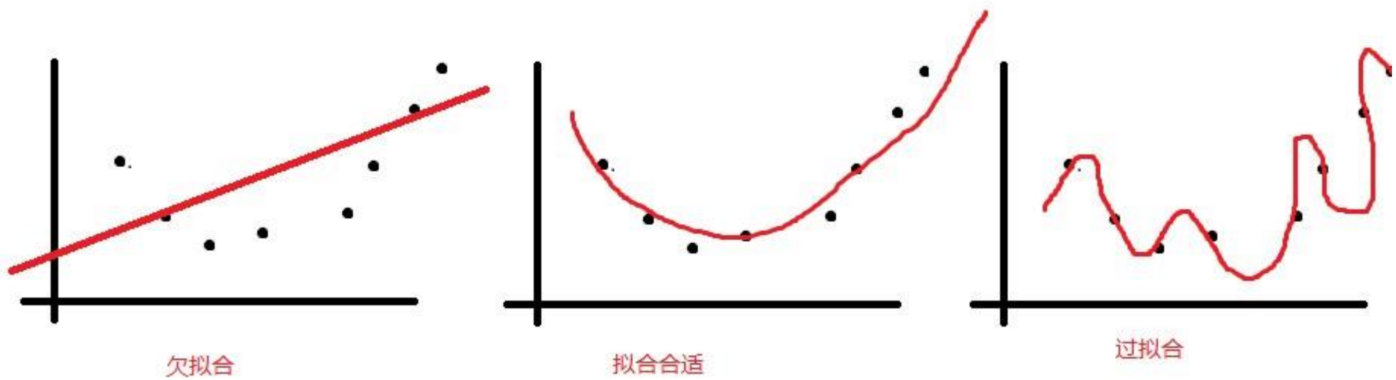
hidden layer update

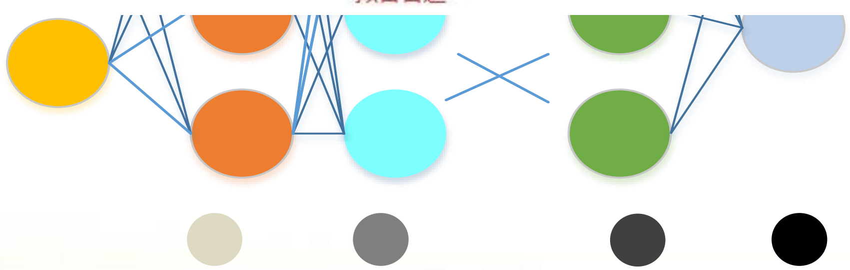
$$w_{ji}^{(1)} = w_{ji}^{(1)} + \Delta w_{ji}^{(1)}$$
$$b_j^{(1)} = b_j^{(1)} + \Delta b_j^{(1)}$$

Problems

- over-fitting
- vanishing gradient

  As errors propagate from layer to layer, they shrink exponentially with the number of layers, So the weights far to output layer is hard to learn well and need a very long time to converge

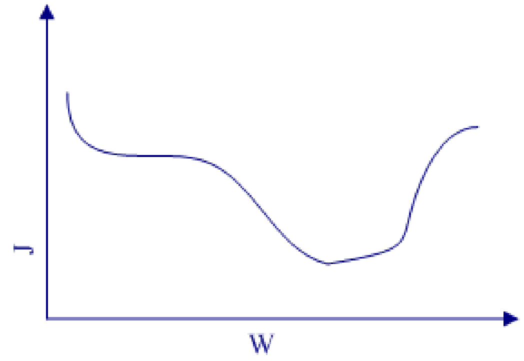




Problems

- Momentum item
  - Sometimes $\frac{\partial J}{\partial W}$ very small or very large
  - it will influence the performance
  - We need smooth the gradient

$$\Delta w_{t+1} = M \cdot \Delta w_t - \eta \frac{\partial J}{\partial W_t}$$



Problems

- Weight decay item
  - When not enough data can be achieved over-fitting always happens
  - A heuristic method is keeping the weights not too large
  - So we can modify the Loss function to

$$J(w)_{new} = J(w) + \varepsilon w^t w$$

So $$\frac{\partial J(w)_{new}}{\partial w} = \frac{\partial J(w)}{\partial w} + 2\varepsilon w$$

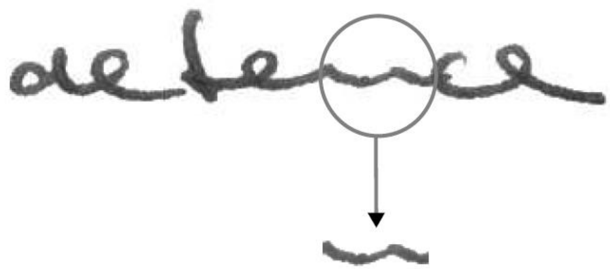
The $2\varepsilon w$ is called weight decay item

Problems

- Pre-training

  If we can get an initial weight which is near an optimum weight so we can modify the weight slightly to optimum.

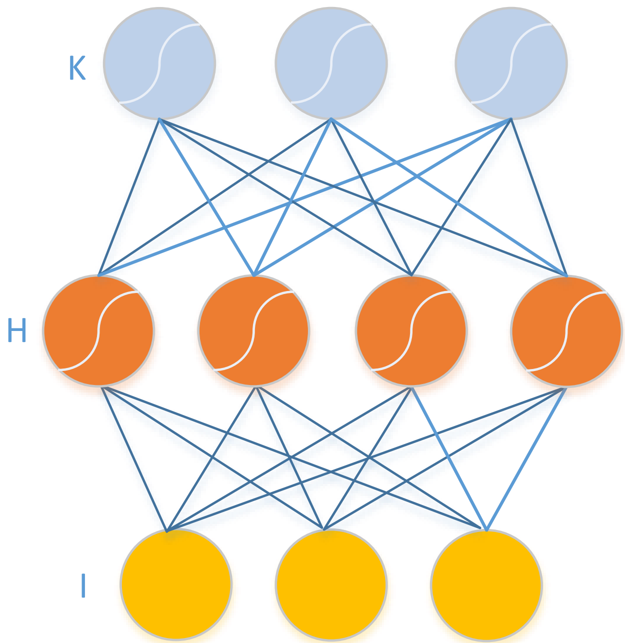
Problem

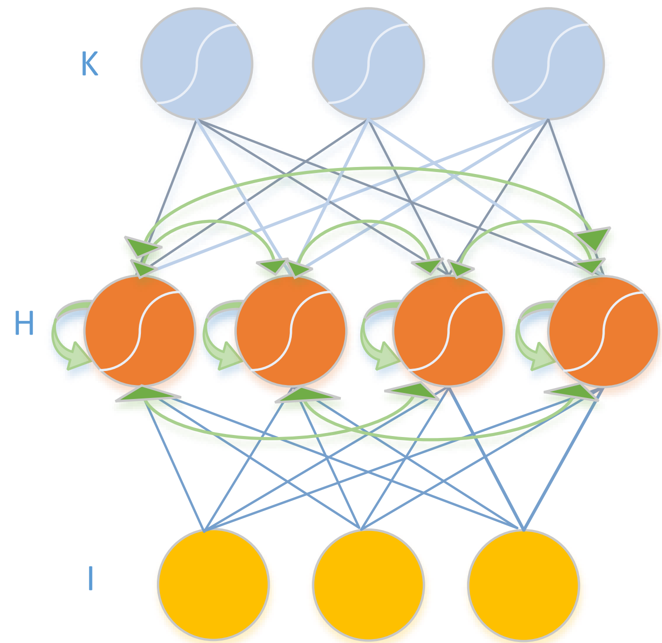


See more, listen more, remember more ?

Make DNN have memories

Recurrent Neural Network





$$DNN: \ a_h^t = \sum_{i=1}^{I} w_{ih} x_i^t$$

$$b_h^t = \theta_h(a_h^t)$$

$$RNN: \ a_h^t = \sum_{i=1}^{I} w_{ih} x_i^t + \boxed{\sum_{h'=1}^{H} w_{h'h} b_{h'}^{t-1}}$$

$$b_h^t = \theta_h(a_h^t)$$

- DNN： map the current input to output
- RNN： map the current input and the current history to output

Thanks

Q & A