Neural Network

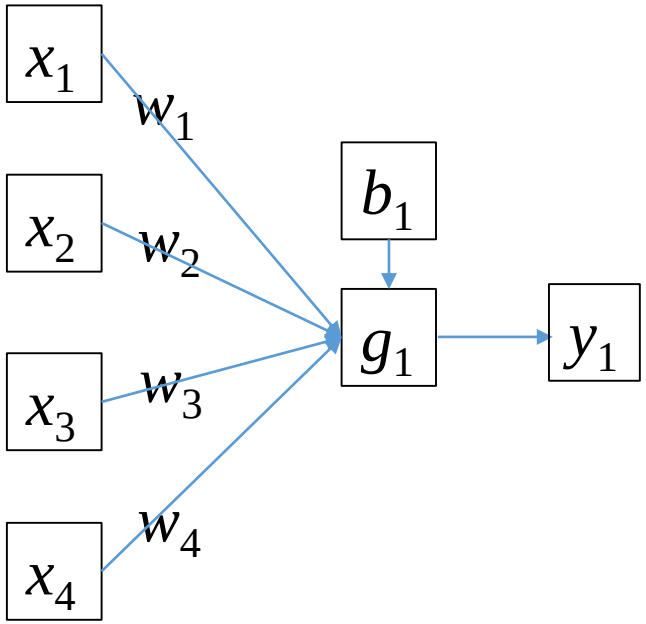
JrPhy

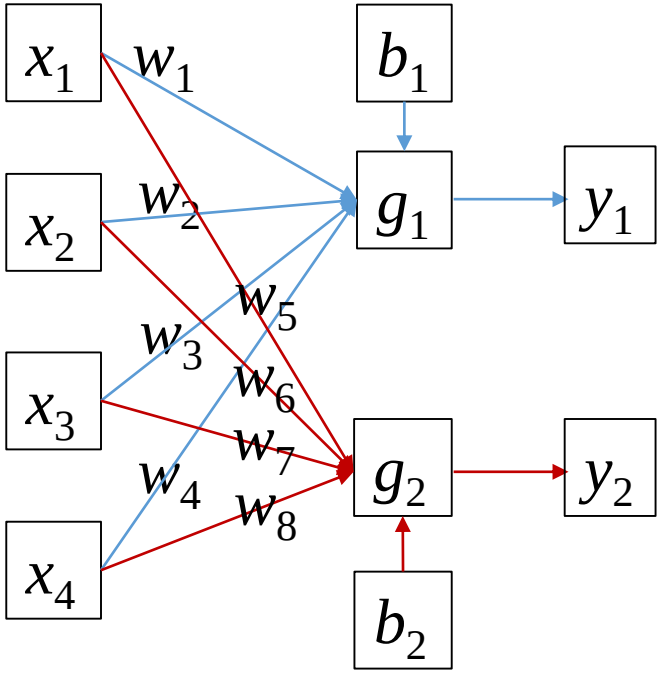
Introduction

- In the SVM, there are so many input, and times the coefficient, plus the bias, then get a number, and put it into some function, like logistic function, then use the output.
- So SVM just inputs a data in the form of vector, then times a coefficient in the form vector, get a number and by a classifier, it’s called a neuron in NN.
- NN is connected by so many SVM or other methods use to classify something. It still input the same data, but you can try another coefficient with different classifier.

Pictorial view of SVM structure

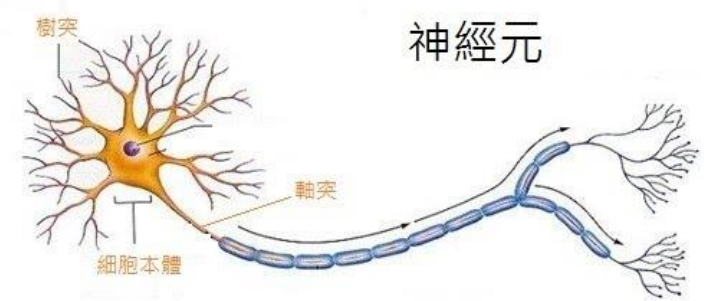
- The mathematical form of SVM is $y = g(\sum_{i=1}^{n} w_i x_i + b)$
- So its structure can be drawn as the left figure, and there are 2 SVM in the right figure, each SVM uses the same input data.



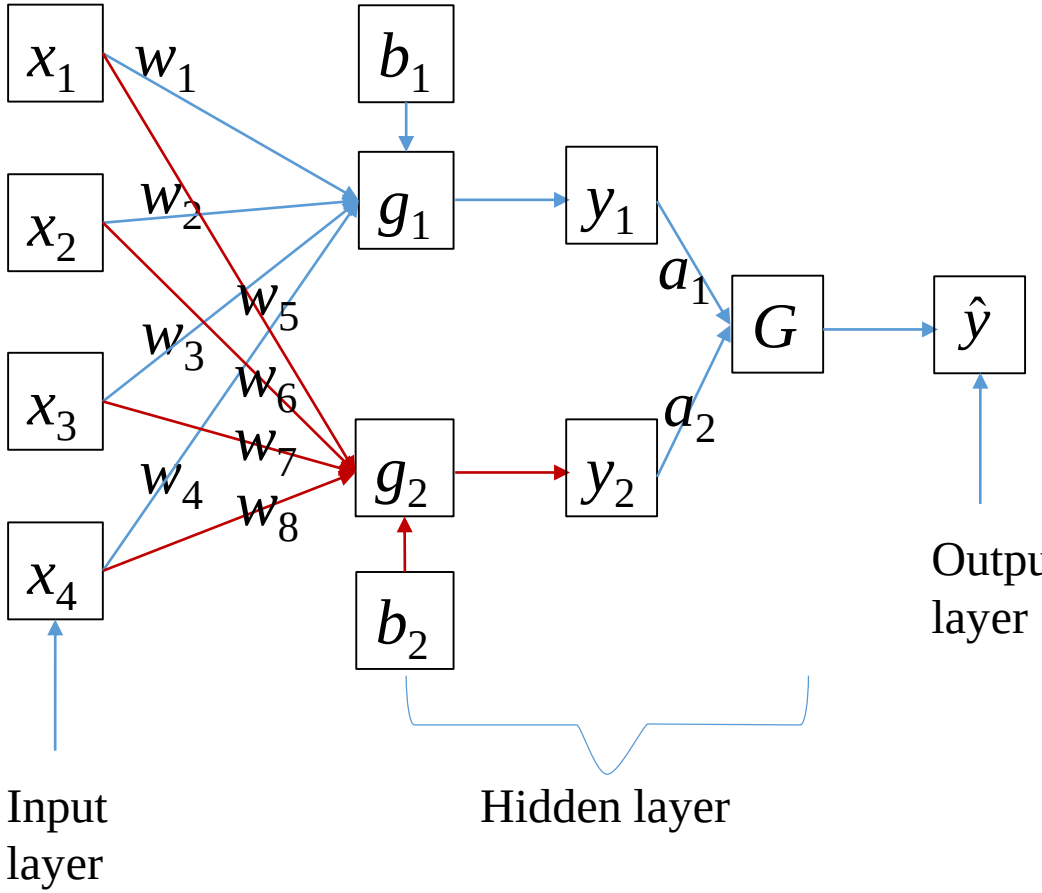


Pictorial view of NN structure

- So a SVM is called neuron, many SVM are connected, it's called neural network.
- The input layer is the data, output layer is the result, and other layers in the middle are called hidden layer, so g and G are hidden layer.



https://sites.google.com/site/2015nthusummercamp16/shen-jing-wang-lu



Mathematical form of NN

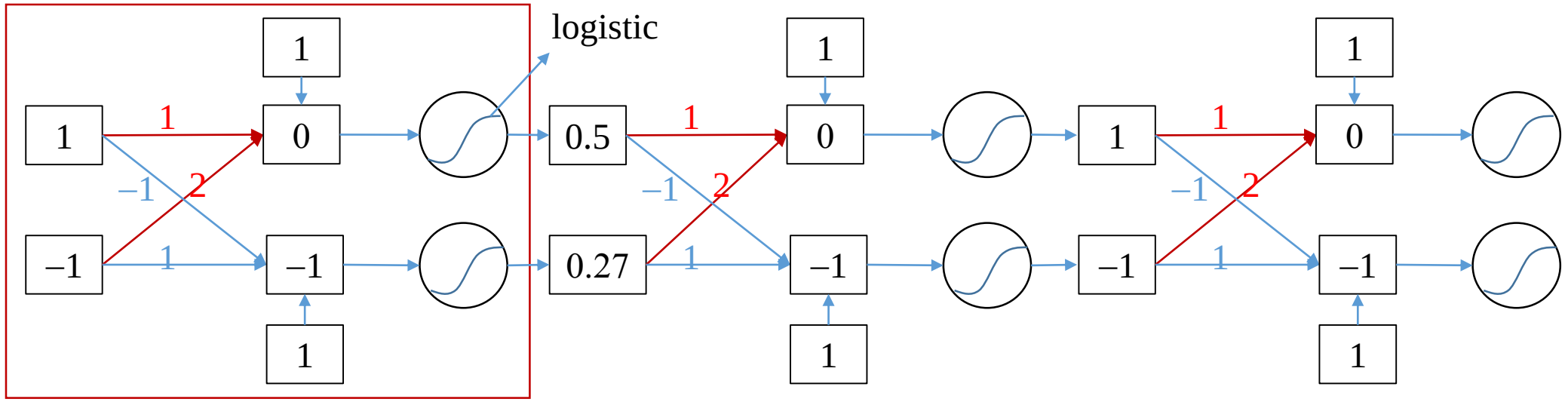
- In general, we use $w_{ij}^{(\ell)}$ to represent the coefficients in NN,
- $$w_{ij}^{(\ell)} = \begin{cases} 1 \le \ell \le L & \text{, layers} \\ 0 \le i \le d^{(\ell-1)} & \text{, input} \\ 1 \le j \le d^{(\ell)} & \text{, output} \end{cases} \quad \text{, score } s = \sum_{j=0}^{\ell-1} w_{ij}^{(\ell)} x_j$$
- The current output is the next input in the hidden layer.
- So the score is gotten by the inner product, that means each hidden layer finds the most likely pattern of input data.

Mathematical form of NN

- Suppose there are 2 inputs, denotes $\boldsymbol{x}$ in vector form, and 2 outputs, so we can write down the system of equations:

$$\begin{cases} w_{11}x_1 + w_{12}x_2 + b_1 = s_1 \\ w_{21}x_1 + w_{22}x_2 + b_2 = s_2 \end{cases} \rightarrow \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 \end{bmatrix}$$

- After throwing the score into some function, get the next input

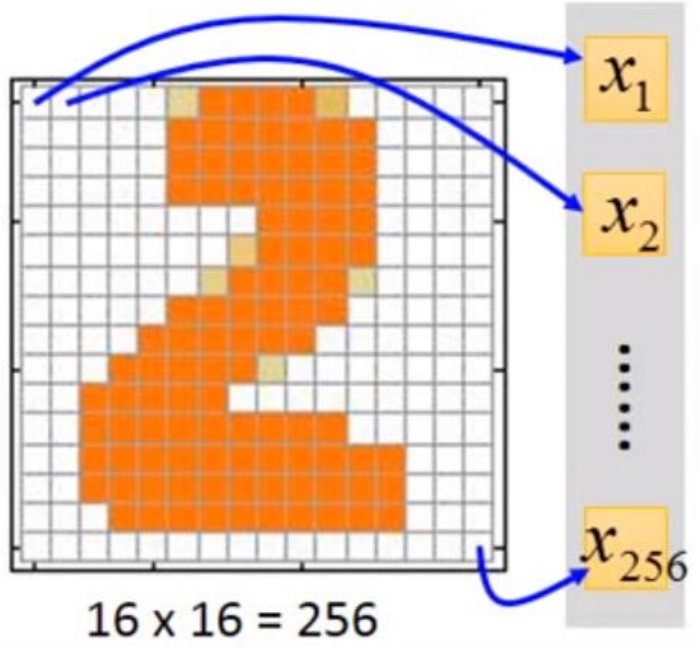


Mathematical form of NN

- The all inputs connect all neurons and it forward propagation, so the structure of that NN is called fully-connected neural network.
- In practical, you can construct your neural network by your problem, in pattern recognized, CNN is popular because it can eliminate some weights which are not important for the pattern, so CNN is efficient in pattern recognized.
- Recurrent neural network (RNN) is usually use for the NLP, so base on the problem to choose network is important.

Mathematical form of NN

- Then the next neuron repeat the step, until the output. So the NN can be seen as a function *f* such that
- $y = f(\boldsymbol{x})$
- $= \sigma(\boldsymbol{w}^{T(L)}\sigma(\boldsymbol{w}^{T(L-1)}\sigma(\boldsymbol{w}^{T(L-2)}\boldsymbol{s}^{(L-3)}\ldots\sigma(\boldsymbol{w}^{T(1)}\boldsymbol{x}^{(0)})))\ldots$
- For example, if you want to classify a pattern in the image, its size is 16 ×16 = 256, so that the dimension of input vector is 256.



https://youtu.be/Dr-WRlEFefw?t=1814

Backward propagation

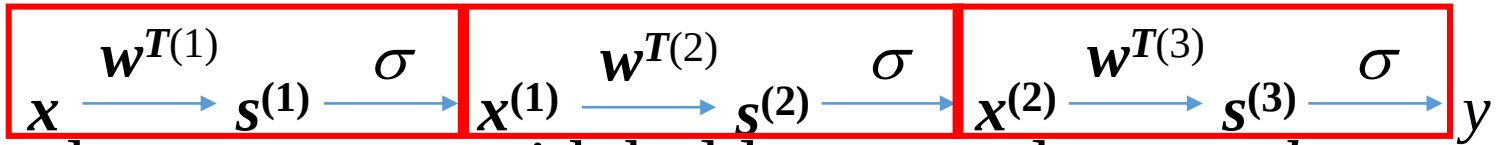
- Same as in other machine learning method, all we want to minimize is the error, and here we still use square error.
- Let's start from the last layer, the error is $e_n = (y_n - s_n^{(L)})^2$.

$$\frac{\partial e_n}{\partial w_{i1}^{(L)}} = \frac{\partial e_n}{\partial s_1^{(L)}} \frac{\partial s_1^{(L)}}{\partial w_{i1}^{(L)}} = -2(y_n - s_1^{(L)}) x_j^{(L-1)} \qquad \frac{\partial e_n}{\partial s_1^{(L)}} = \delta_i^{(L)}$$

- For the hidden layer, suppose we use the function σ to get the score, and 1st to $(L-1)$th are hidden layer, then the final score is
- $y_n = s_n^{(L)} = \sigma(\boldsymbol{w}^{\boldsymbol{T}(L)} \boldsymbol{s}^{(L-1)}) = \sigma(\boldsymbol{w}^{\boldsymbol{T}(L)} \sigma(\boldsymbol{w}^{\boldsymbol{T}(L-1)} \boldsymbol{s}^{(L-2)})) = \ldots$
- $= \sigma(\boldsymbol{w}^{\boldsymbol{T}(L)} \sigma(\boldsymbol{w}^{\boldsymbol{T}(L-1)} \sigma(\boldsymbol{w}^{\boldsymbol{T}(L-2)} \boldsymbol{s}^{(L-3)} \ldots \sigma(\boldsymbol{w}^{\boldsymbol{T}(1)} \boldsymbol{x}^{(0)}))) \ldots)$

Backward propagation

- Suppose there are 3 hidden layers, then



- Here $\boldsymbol{w}$ and $\boldsymbol{x}$ are vector with bold type, and $w_0 x_0 = b$
- Each input times the weight, then get a score, then throw the score into the function to get the next input. For hidden layer, what we want to calculate is

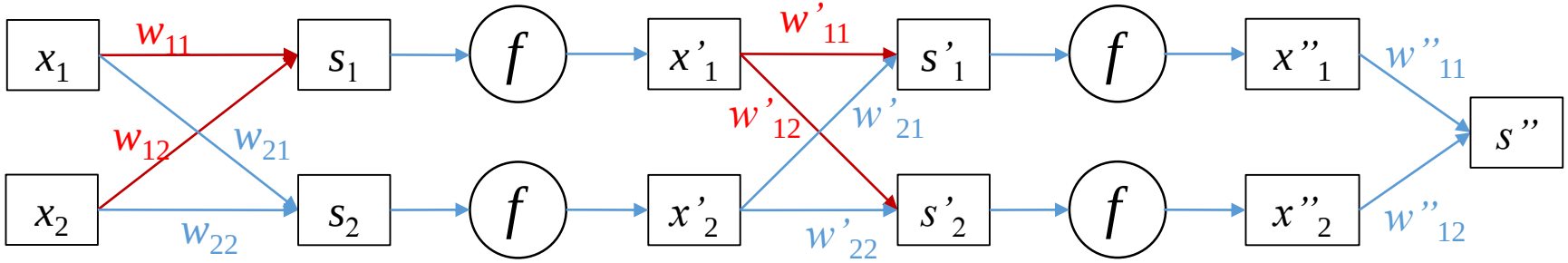
$$\delta_i^{(\ell)} = \frac{\partial e_n}{\partial s_i^{(\ell)}} = \sum_{k=1}^{d^{(\ell+1)}} \frac{\partial e_n}{\partial s_k^{(\ell+1)}} \frac{\partial s_k^{(\ell+1)}}{\partial x_j^{(\ell)}} \frac{\partial x_j^{(\ell)}}{\partial s_j^{(\ell)}} = \sum_k \delta_k^{(\ell+1)} w_{jk}^{(\ell+1)} \sigma'(s_j^{(\ell)})$$

$$\frac{\partial s_k^{(\ell+1)}}{\partial x_j^{(\ell)}} = \frac{\partial \sum_i w_{jk}^{(\ell+1)} x_j^{(\ell)}}{\partial x_j^{(\ell)}} = w_{jk}^{(\ell+1)} \qquad \frac{\partial x_j^{(\ell)}}{\partial s_j^{(\ell)}} = \frac{\partial \sigma(s_j^{(\ell)})}{\partial s_j^{(\ell)}} = \sigma'(s_j^{(\ell)})$$

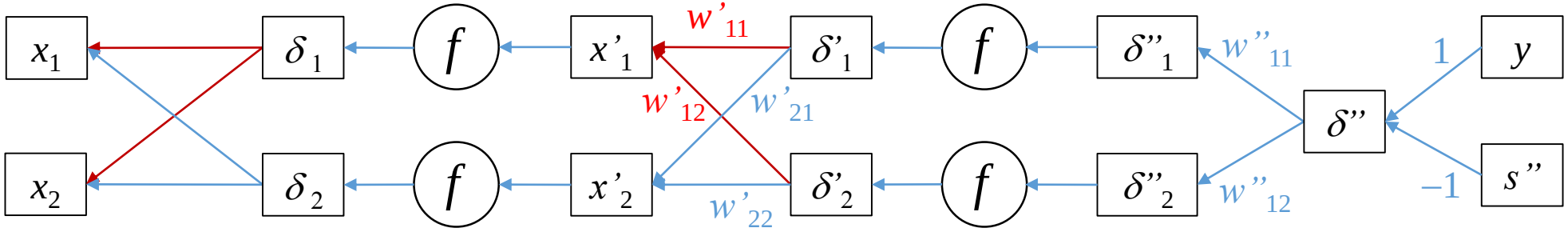
- This is chain rule. It minimizes error from the output to the input, then updates the weight, so it's called backward propagation.

Forward and Backward propagation example

- The weight in the backward propagation in the same as the forward. The figure of forward propagation is below:

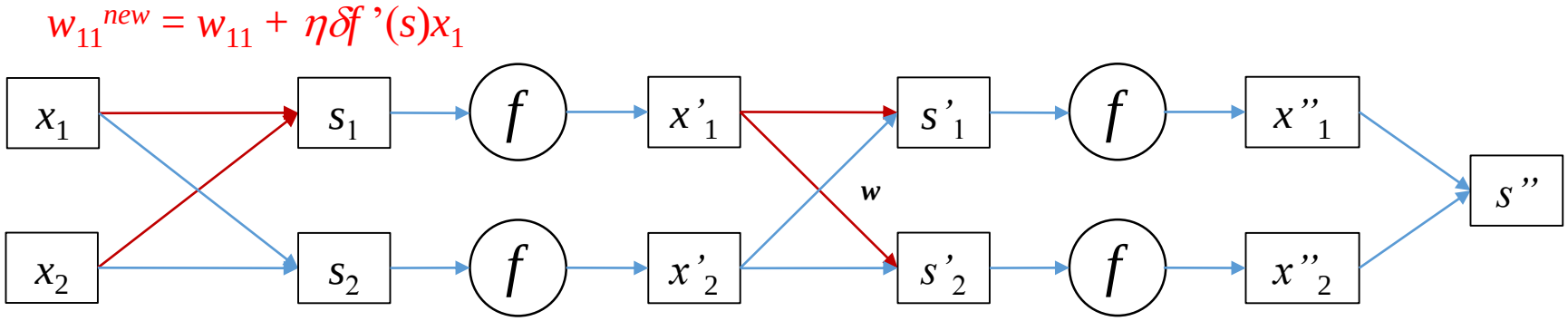


- Then calculate the backward propagation



Forward and Backward propagation example

- At the input layer, update the weight by gradient descent.



- And so do the other weights, so that it's a turn of forward and backward propagation. There are so many tool so that it doesn't need to write it by yourself.

http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html
https://www.youtube.com/watch?v=ibJpTrp5mcE