电子科技大学
University of Electronic Science and Technology of China

Preliminary Study of Deep Learning

Wei Han

Data Mining Lab,
Big Data Research Center, UESTC
Email：wei.hb.han@gmail.com

Outline

- Multi Layer Perceptron
- Convolution Neural Network
- Recurrent Neural Networks
- Long-Short Term dependencies (LSTM)

0. Be motivated in deep neural network

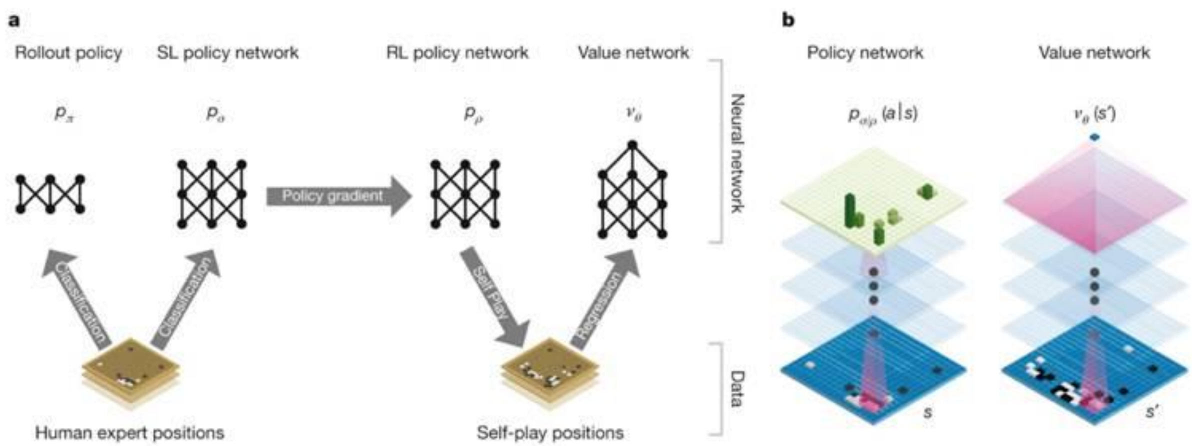
- Overwhelming in performance!!!

- Significantly broaden our available research area and imagination!

0. Be motivated in deep neural network

数据挖掘实验室
Data Mining Lab

Neural network training pipeline and architecture



D Silver *et al.* *Nature* **529**, 484–489 (2016) doi:10.1038/nature16961

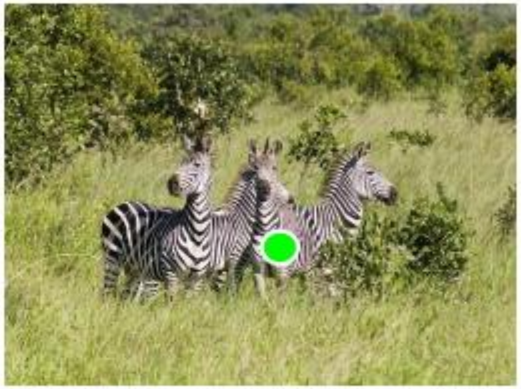
nature

Figure 2-1. Mechanism of AlphaGo

0. Be motivated in deep neural network

- A girl wearing glasses and a pink shirt.
- An Asian girl with a pink shirt eating at the table.

- A boy brushing his hair while looking at his reflection.
- A young male child in pajamas shaking around a hairbrush in the mirror.



- Zebra looking towards the camera.
- A zebra third from the left.

Figure 2-2. Examples of picture talking

0. Be motivated in deep neural network


```
#include<stdio.h>
void main()
{        (1) n
    int x,a[100],i,max1,max2;
    scanf("%d",&n);
    for(i=0;i<=n-1;i++)
    {
        scanf("%d",&a[i]);
        if (a[i]>max1)
            max2=a[i];}
        for(i=1;i<=n;i++)
    {                  (2) 2    (3) <
        if(a[i]>max&&a[i]==max1)
            max2=a[i];
    }

    printf("%d\n%d",max1,max2);
    return 0; (4) [del]
}
```

(a) Generated code

```
#include<stdio.h>
int main(){
    int n,i,j,sz[100],max=0,ci=0;
    scanf("%d",&n);
   for(i=0;i<n;i++){
       scanf("%d",&sz[i]);
       if(sz[i]>max){
          max=sz[i];}}
    for(i=0;i<n;i++){
 if(sz[i]>ci&&sz[i]<max){
       ci=sz[i];}}
 printf("%d\n%d",max,ci);
return 0;
}
```

(b) Training sample 1

```
#include<stdio.h>
void main()
{
    int n,i,a[100],j,max1,max2;
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    max1=a[0];
    for(i=0;i<n;i++)
    {
        if(a[i]>max1)
            max1=a[i];
    }
    for(i=0;i<n;i++)
    {
        if(max1==a[i])
            j=i;
    }
    if(max1!=a[0])
    max2=a[0];
    else max2=a[1];
    for(i=0;i<n;i++)
    {
        if(i==j) continue;
        if(a[i]>max2)
            max2=a[i];
    }
        printf("%d\n%d",max1,max2);
}
```

(c) Training sample 2

Figure 2: (a) Code generated by RNN. The code is almost correct except 4 wrong characters (among ~280 characters in total), highlighted in the figure. (b) Code with the most similar structure in the training set, detected by ccfinder. (c) Code with the most similar identifiers in the training set, also detected by ccfinder. Note that we preserve all indents, spaces and line feeds. The 4 errors are (1) The identifier "x" should be "n"; (2) "max" should be "max2"; (3) "==" should be "<"; (4) return type should be void.

Figure 2-3. Demonstration of Program Generation

1.1.0 Single Neuron

Single neuro:

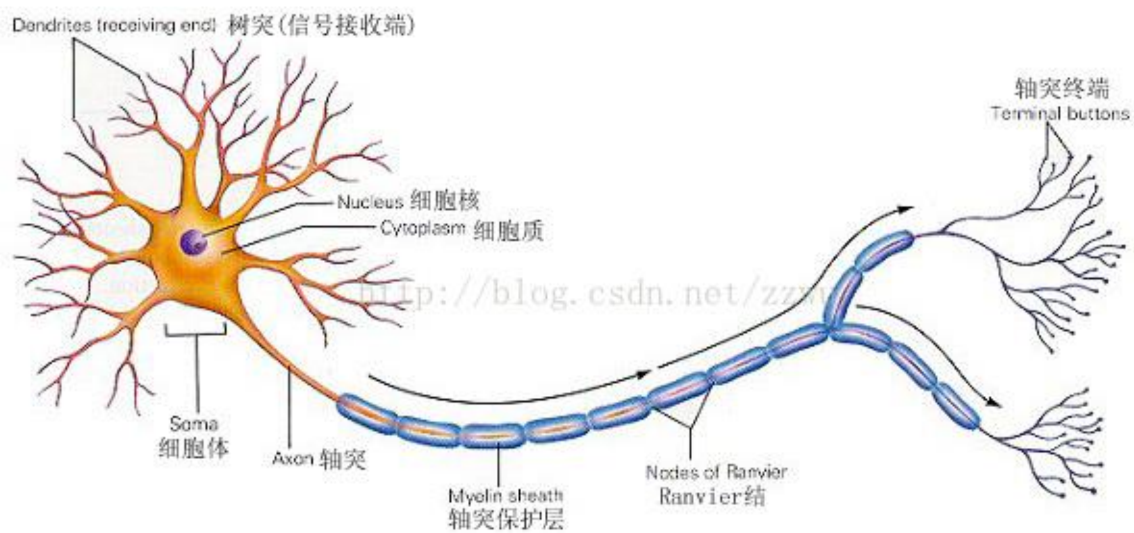


Figure 2-4.
An example of neuro

1.1.1 Perception

The basic unit of computation in a neural network is the **neuron**, often called a **node** or **unit**. It receives input from some other nodes, or from an external source and computes an output.

The function *f* is non-linear and is called the **Activation Function**. The purpose of the activation function is to introduce non-linearity into the output of a neuron.

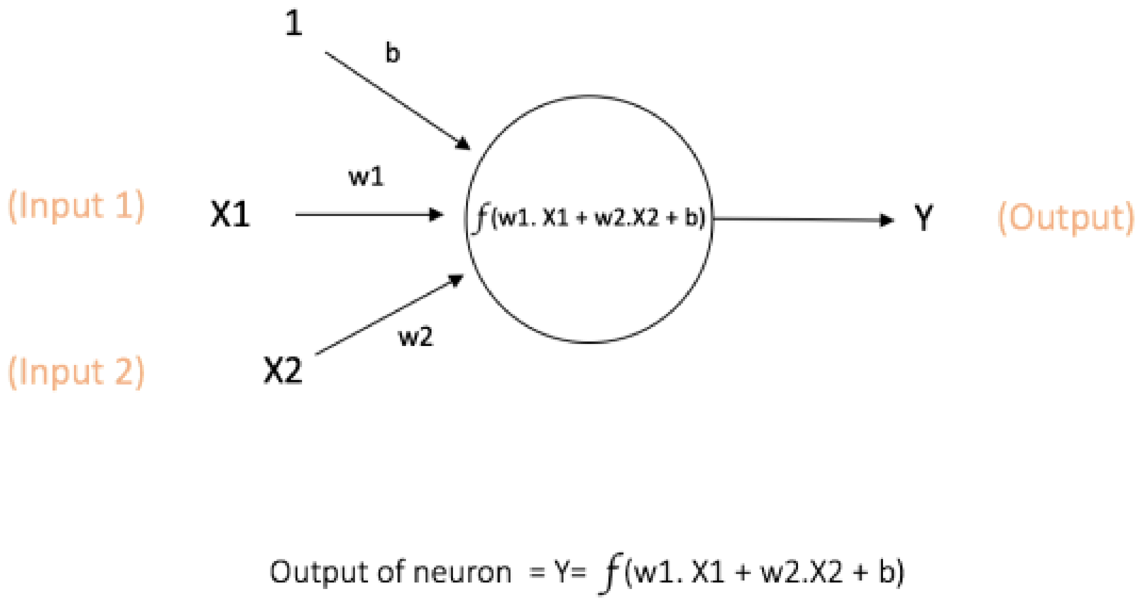


Figure 2-5. A Single Neuron

1.1.2 Activation Function

- **Sigmoid:**

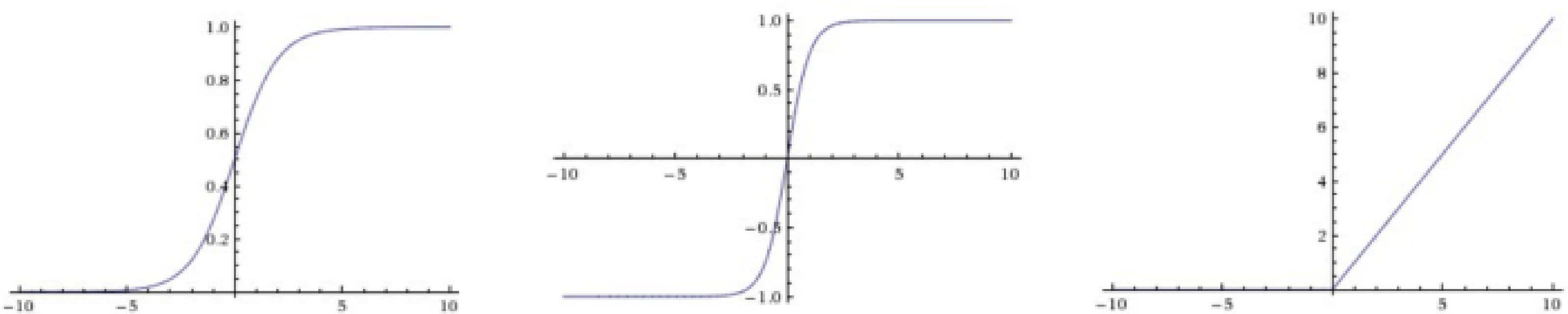
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- **tanh:**

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

- **ReLU (**Rectified Linear Unit**):**

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



Sigmoid tanh ReLU

Figure 2-6. Different activation functions

An Artificial Neural Network (ANN)

is a computational model that is inspired by the way biological neural networks in the human brain process information.

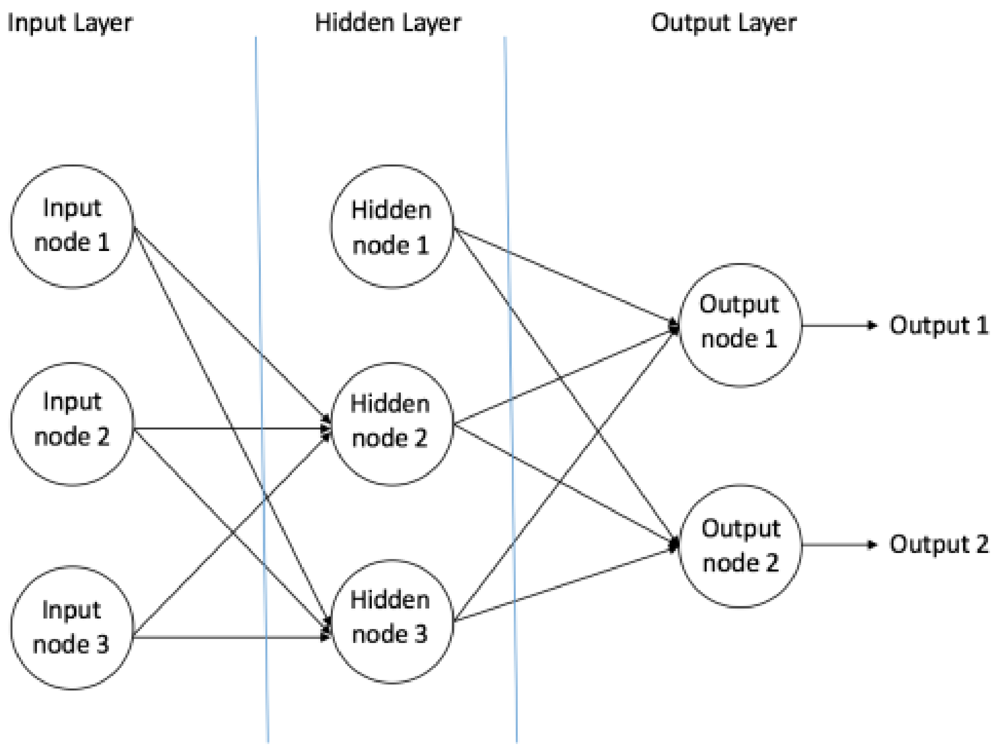


Figure 2-7. An example of feedforward neural network

[A Quick Introduction to Neural Networks]

1.1.4 Multi Layer Perceptron

A Multi Layer Perceptron (MLP) contains one or more hidden layers. While a single layer perceptron can only learn linear functions, a multi layer perceptron can also learn non – linear functions.

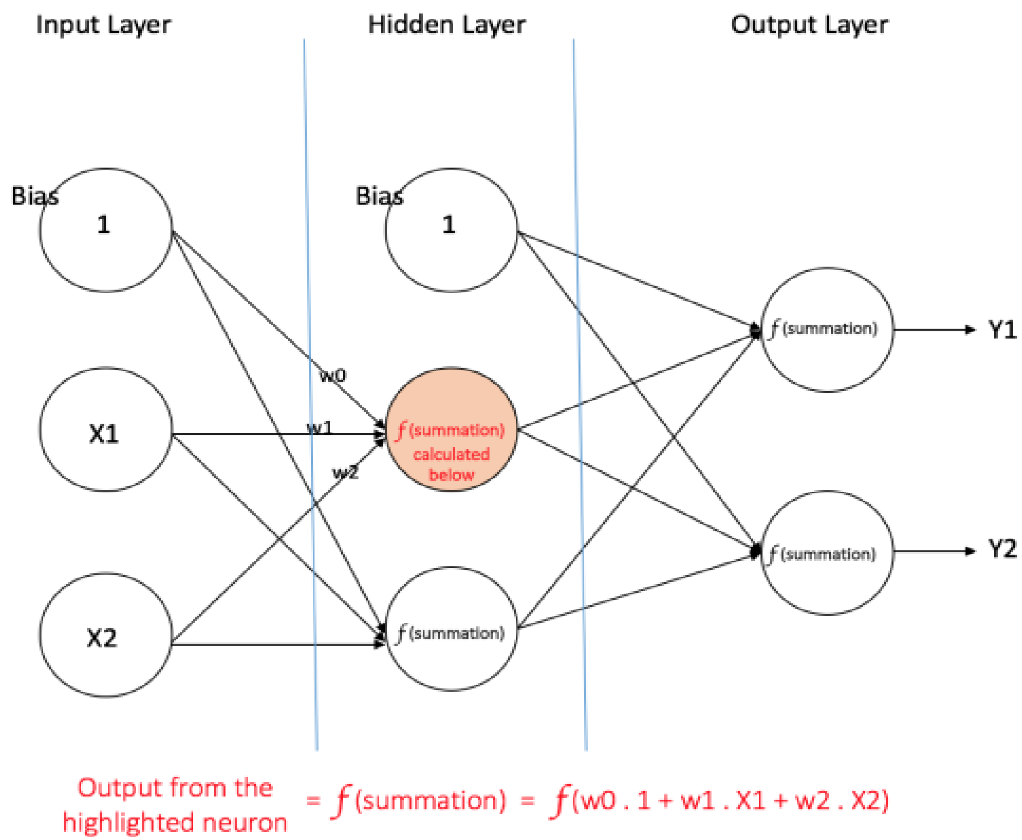


Figure 2-8.
A multi layer perceptron having one hidden layer

1.1.4 The Back-Propagation Algorithm

“learning from mistakes“

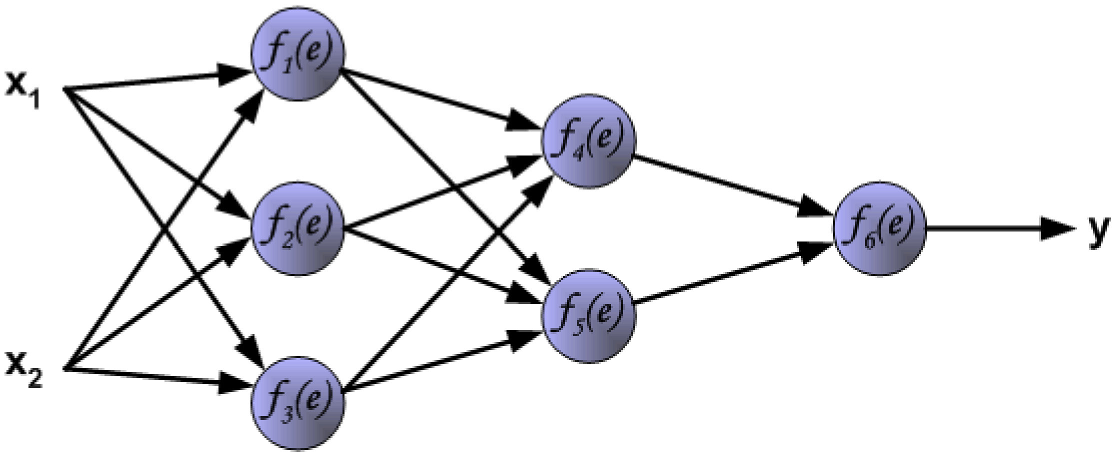
This output is compared with the desired output that we already know, and the error is “propagated” back to the previous layer.

This error is noted and the weights are “adjusted” accordingly.

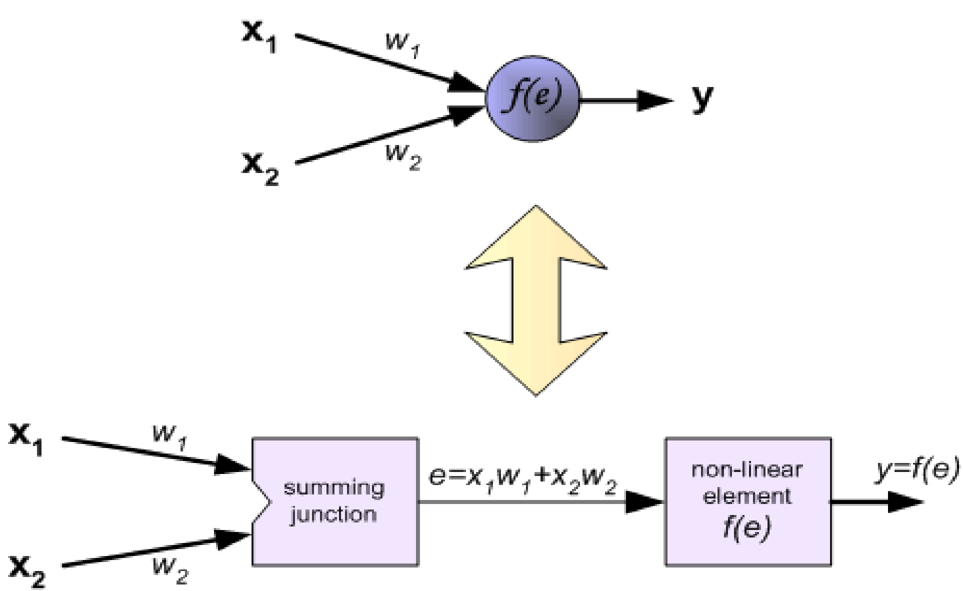
This process is repeated until the output error is below a predetermined threshold.

From Quora

1.1.4 Demonstration of BP Algorithm

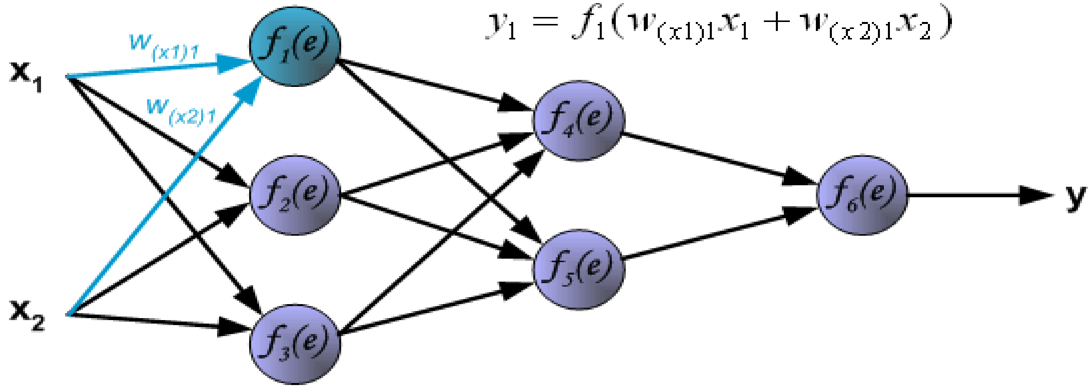


Principles of training multi-layer neural network using backpropagation

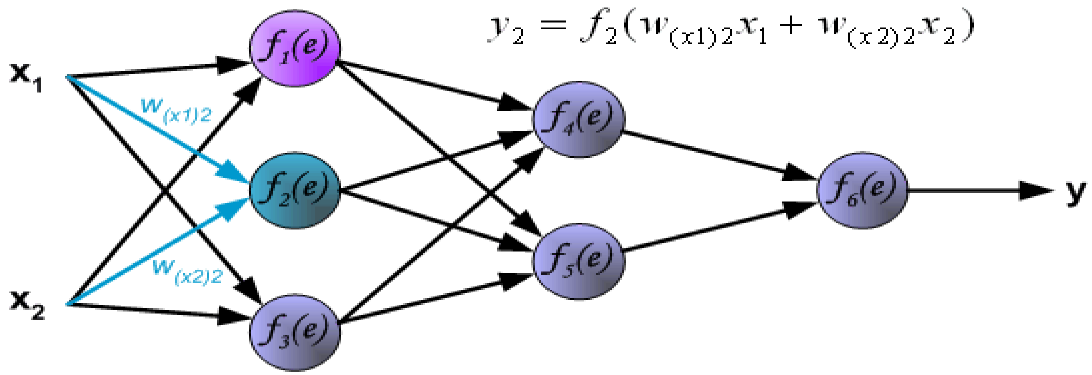


1.1.4 Demonstration of BP Algorithm

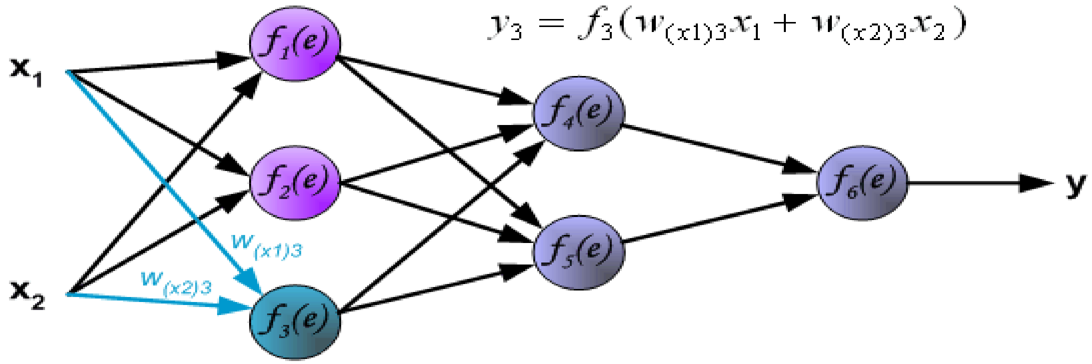
$$y_1 = f_1(w_{(x1)1}x_1 + w_{(x2)1}x_2)$$



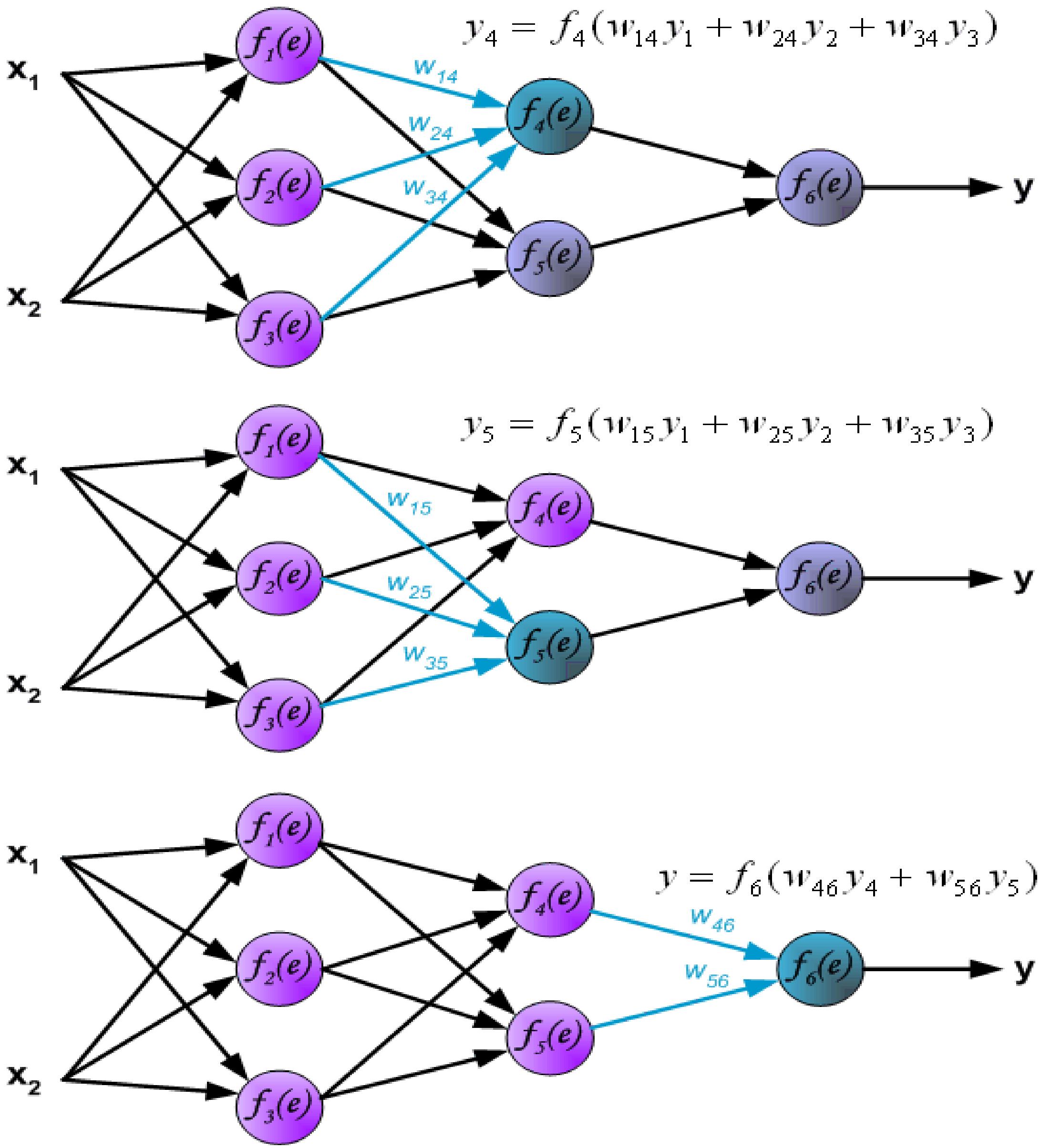
$$y_2 = f_2(w_{(x1)2}x_1 + w_{(x2)2}x_2)$$



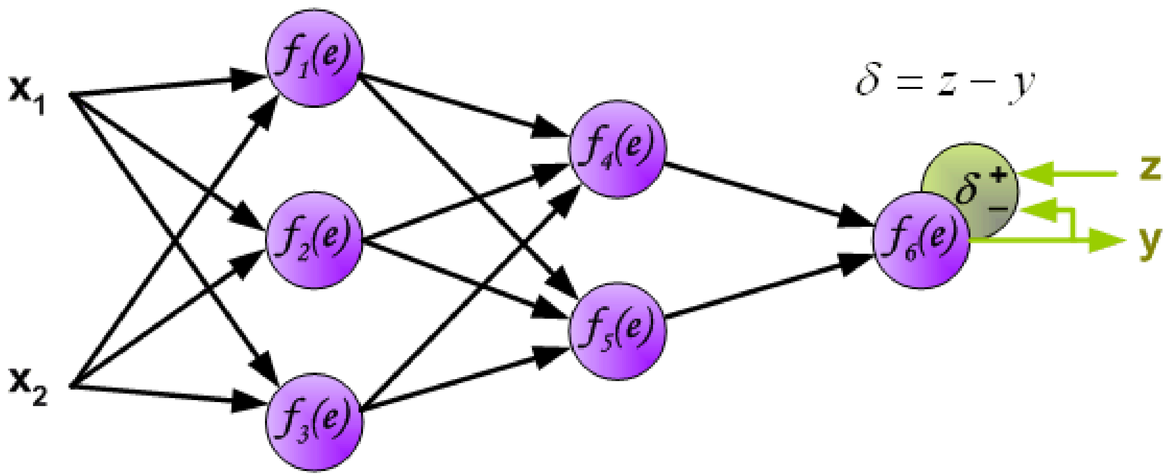
$$y_3 = f_3(w_{(x1)3}x_1 + w_{(x2)3}x_2)$$



1.1.4 Demonstration of BP Algorithm

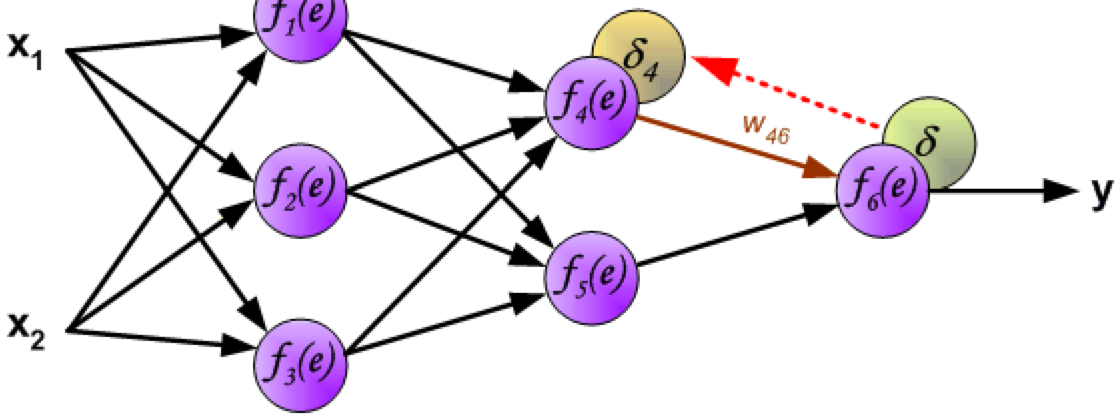


1.1.4 Demonstration of BP Algorithm

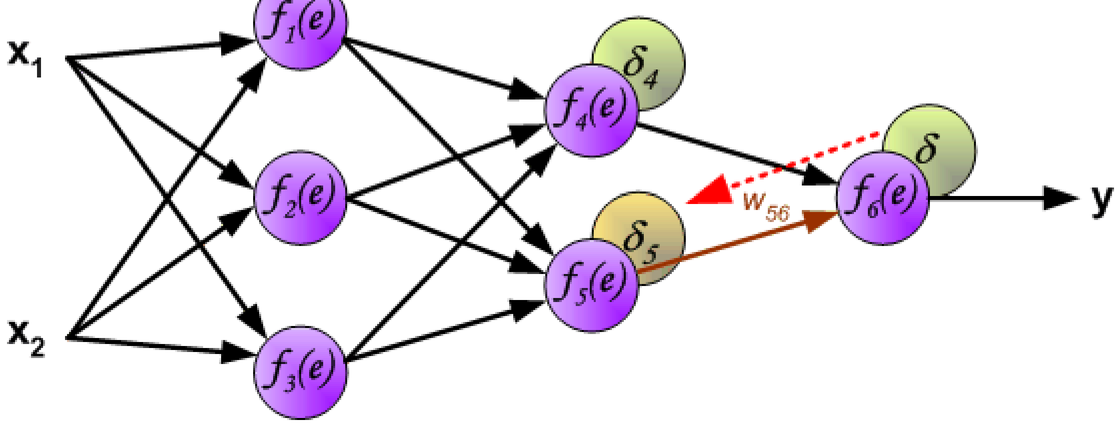


1.1.4 Demonstration of BP Algorithm

$$\delta_4 = w_{46}\delta$$

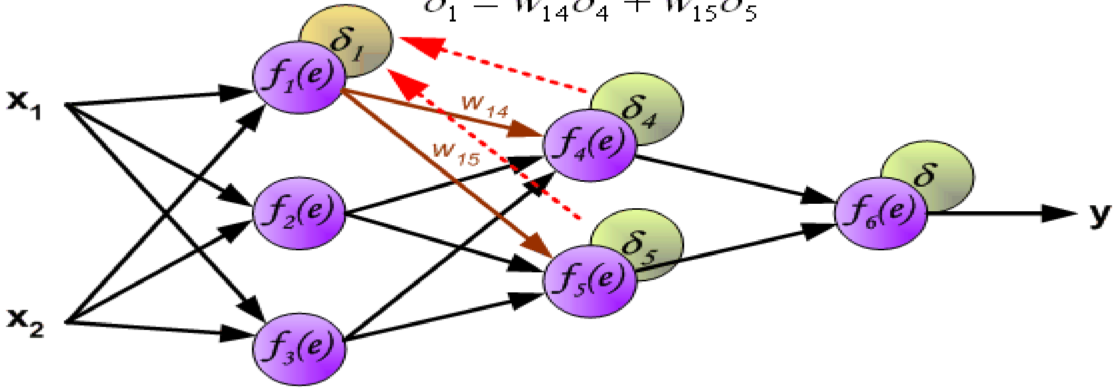


$$\delta_5 = w_{56}\delta$$

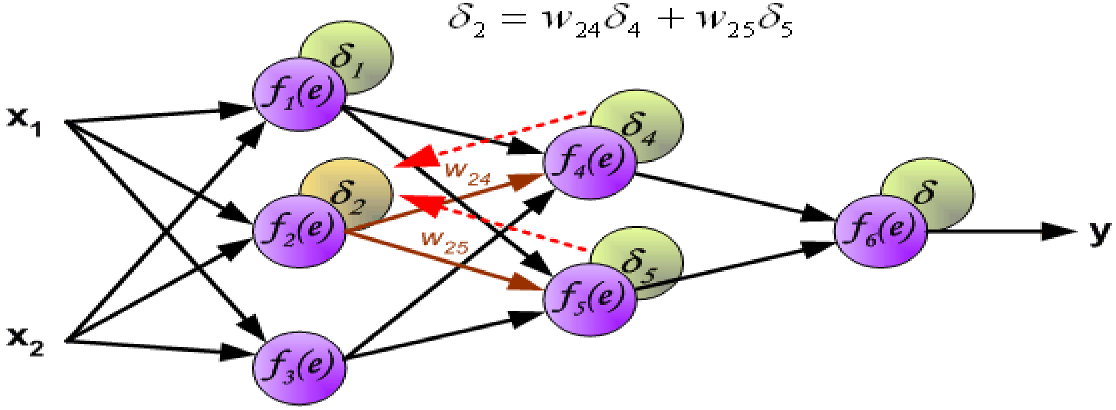


1.1.4 Demonstration of BP Algorithm

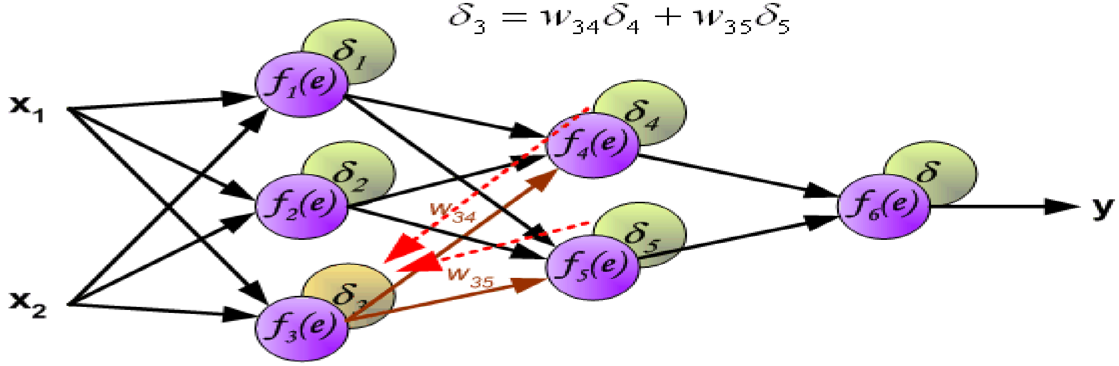
$$\delta_1 = w_{14}\delta_4 + w_{15}\delta_5$$



$$\delta_2 = w_{24}\delta_4 + w_{25}\delta_5$$



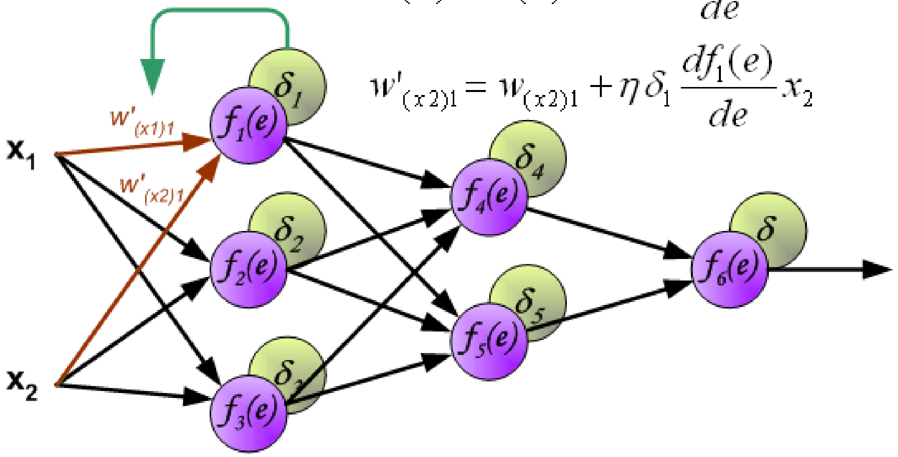
$$\delta_3 = w_{34}\delta_4 + w_{35}\delta_5$$



1.1.4 Demonstration of BP Algorithm

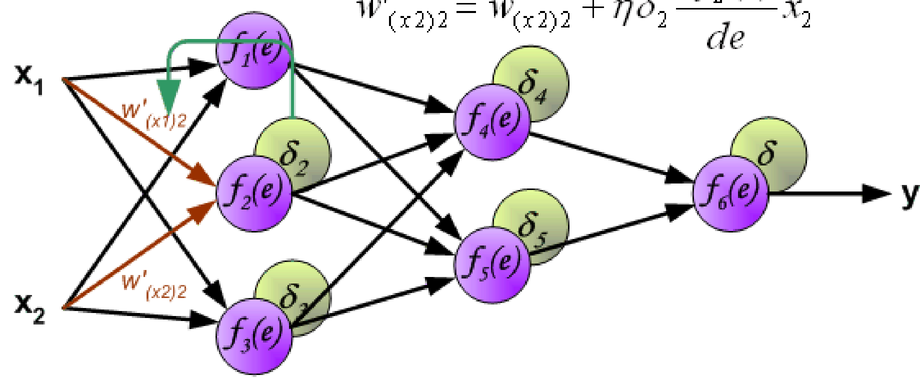
$$w'_{(x1)1} = w_{(x1)1} + \eta \delta_1 \frac{df_1(e)}{de} x_1$$

$$w'_{(x2)1} = w_{(x2)1} + \eta \delta_1 \frac{df_1(e)}{de} x_2$$



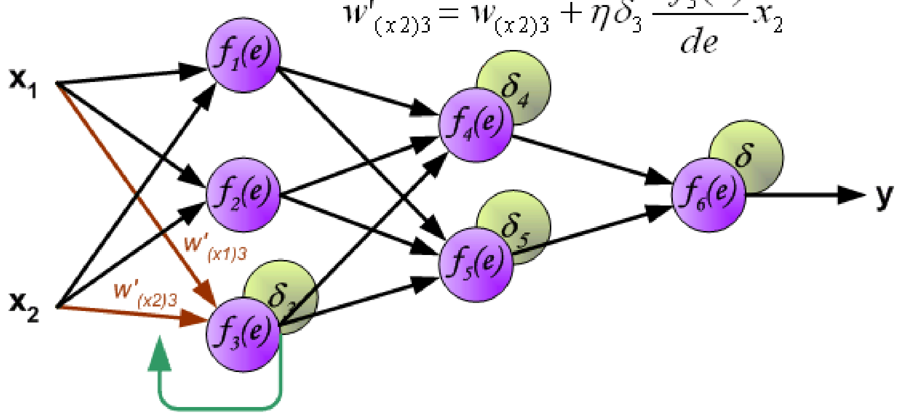
$$w'_{(x1)2} = w_{(x1)2} + \eta \delta_2 \frac{df_2(e)}{de} x_1$$

$$w'_{(x2)2} = w_{(x2)2} + \eta \delta_2 \frac{df_2(e)}{de} x_2$$



$$w'_{(x1)3} = w_{(x1)3} + \eta \delta_3 \frac{df_3(e)}{de} x_1$$

$$w'_{(x2)3} = w_{(x2)3} + \eta \delta_3 \frac{df_3(e)}{de} x_2$$

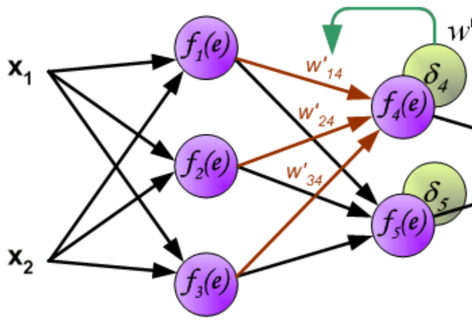


1.1.4 Demonstration of BP Algorithm

$$w'_{14} = w_{14} + \eta\delta_4 \frac{df_4(e)}{de} y_1$$

$$w'_{24} = w_{24} + \eta\delta_4 \frac{df_4(e)}{de} y_2$$

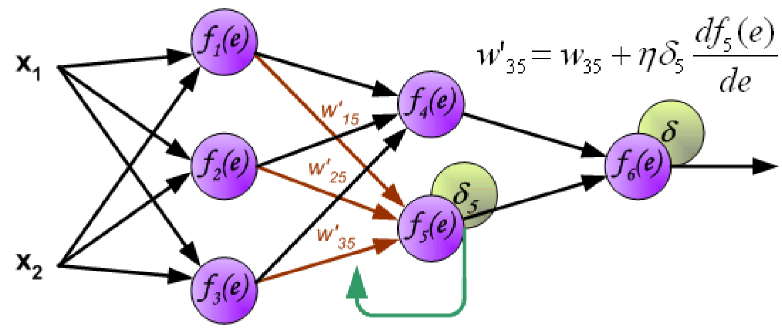
$$w'_{34} = w_{34} + \eta\delta_4 \frac{df_4(e)}{de} y_3$$



$$w'_{15} = w_{15} + \eta\delta_5 \frac{df_5(e)}{de} y_1$$

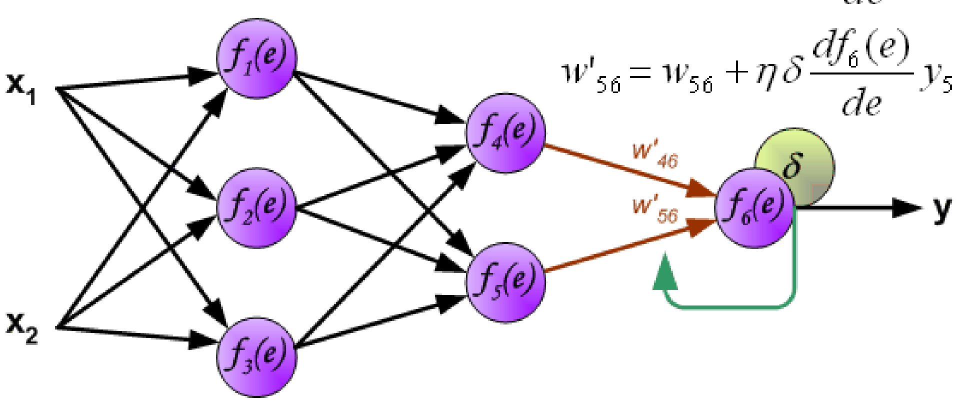
$$w'_{25} = w_{25} + \eta\delta_5 \frac{df_5(e)}{de} y_2$$

$$w'_{35} = w_{35} + \eta\delta_5 \frac{df_5(e)}{de} y_3$$



$$w'_{46} = w_{46} + \eta\delta \frac{df_6(e)}{de} y_4$$

$$w'_{56} = w_{56} + \eta\delta \frac{df_6(e)}{de} y_5$$



1.1.4 Mathematics

For given $x^{(i)}$,$y^{(i)}$, the output of feedforward neural network is $f(x|w,b)$, and then the objective function is:

$$
\begin{aligned}
J(W,b) &= \sum_{i=1}^{N} L(y^{(i)}, f(x^{(i)} \mid W, b)) + \frac{1}{2}\lambda \|W\|_F^2 \\
&= \sum_{i=1}^{N} J(W, b; x^{(i)}, y^{(i)}) + \frac{1}{2}\lambda \|W\|_F^2
\end{aligned}
$$

If we adopt gradient descent method, then

$$
\begin{aligned}
W^{(l)} &= W^{(l)} - \alpha \frac{\partial J(W,b)}{\partial W^{(l)}} \\
&= W^{(l)} - \alpha \sum_{i=1}^{N} \left(\frac{\partial J(W, b; x^{(i)}, y^{(i)})}{\partial W^{(l)}}\right) - \lambda W
\end{aligned}
$$

$$
\begin{aligned}
b^{(l)} &= b^{(l)} - \alpha \frac{\partial J(W,b)}{\partial b^{(l)}} \\
&= b^{(l)} - \alpha \sum_{i=1}^{N} \left(\frac{\partial J(W, b; x^{(i)}, y^{(i)})}{\partial b^{(l)}}\right)
\end{aligned}
$$

1.1.4 Mathematics

By chain rule,

$$\frac{\partial J(W,b;x,y)}{\partial W_{ij}^{(l)}} = tr\left(\left(\frac{\partial J(W,b;x,y)}{\partial z^{(l)}}\right)^T \frac{\partial z^{(l)}}{\partial W_{ij}^{(l)}}\right)$$

Define the first term as a error term $\delta^{(l)}$,

$$\delta^{(l)} = \frac{\partial J(W,b;x,y)}{\partial z^{(l)}} \in \mathbb{R}^{n^{(l)}}$$

Indicating the influence of l$^{\text{th}}$ level neuros to final error.

For the second term $z^{(l)} = W^{(l)} \bullet a^{(l-1)} + b^{(l)}$,

$$\frac{\partial z^{(l)}}{\partial W_{ij}^{(l)}} = \frac{\partial (W^{(l)} \cdot a^{(l-1)} + b^{(l)})}{\partial W_{ij}^{(l)}} = \begin{bmatrix} 0 \\ \vdots \\ a_j^{(l-1)} \\ \vdots \\ 0 \end{bmatrix}$$

Therefore,

$$\frac{\partial J(W,b;x,y)}{\partial W_{ij}^{(l)}} = \delta_i^{(l)} a_j^{(l-1)}$$

and

$$\frac{\partial J(W,b;x,y)}{\partial W^{(l)}} = \delta^{(l)} (a^{(l-1)})^T$$

Similarly, the gradient of b:

$$\frac{\partial J(W,b;x,y)}{\partial b^{(l)}} = \delta^{(l)}$$

Finally, the error term of l$^{\text{th}}$ level, $\delta^{(l)}$

$$\delta^{(l)} \triangleq \frac{\partial J(W,b;x,y)}{\partial z^{(l)}}$$

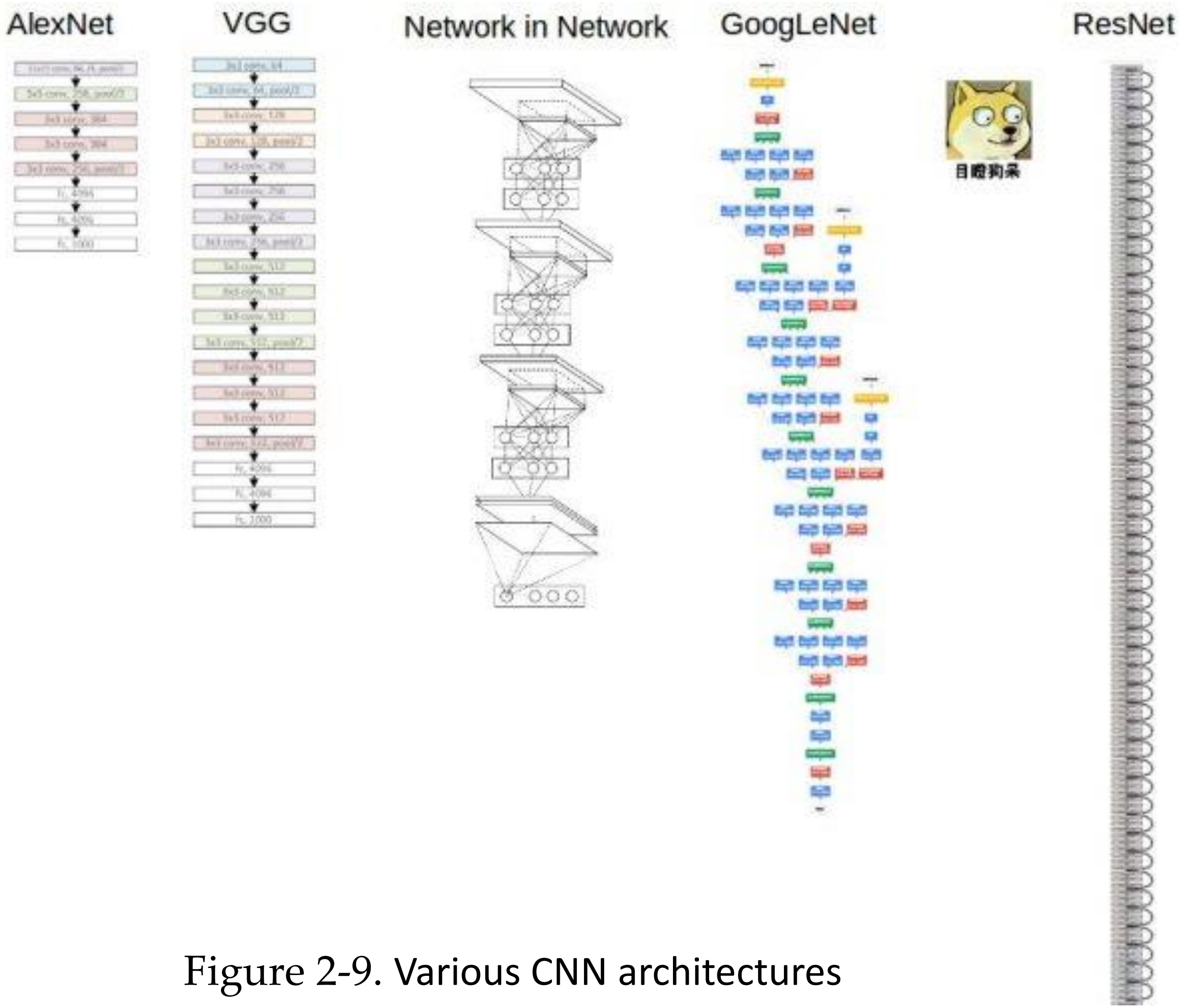
$$= \frac{\partial a^{(l)}}{\partial z^{(l)}} \cdot \frac{\partial z^{(l+1)}}{\partial a^{(l)}} \cdot \frac{\partial J(W,b;x,y)}{\partial z^{(l+1)}}$$

$$= diag(f_l^{'}(z^{(l)})) \cdot (W^{(l+1)})^T \cdot \delta^{(l+1)}$$

$$= f_l^{'}(z^{(l)}) \odot ((W^{(l+1)})^T \cdot \delta^{(l+1)})$$

As is shown, the error term of l$^{\text{th}}$ level can be calculated from the one of (l+1)$^{\text{th}}$ level as back propagation.

1.2.0 Convolutional Neural Networks



152层！！

Figure 2-9. Various CNN architectures

1.2.1 LeNet intuitive introduction

数据挖掘实验室
Data Mining Lab

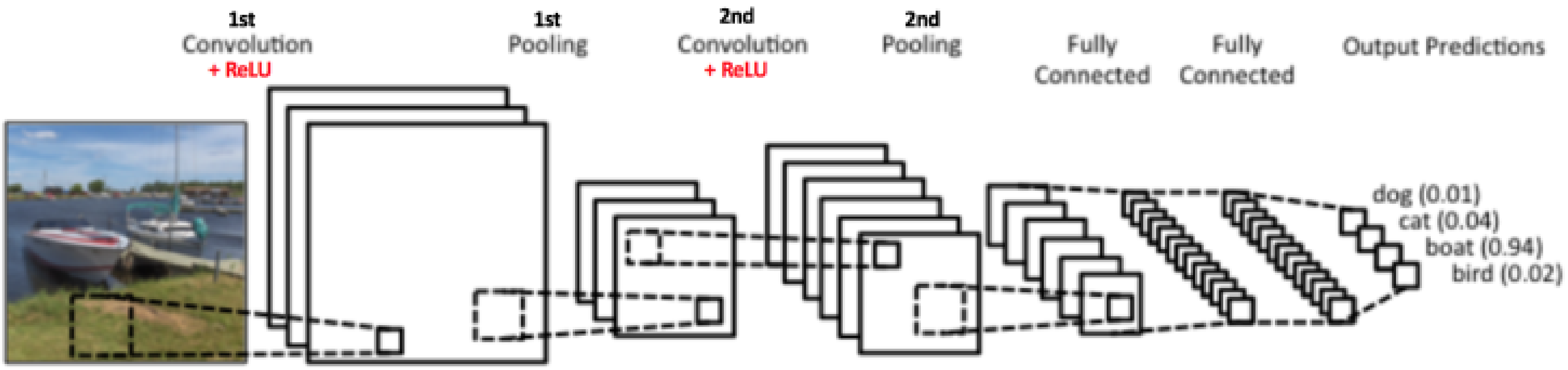


Figure 2-10. The architecture of LeNet

[An Intuitive Explanation of Convolutional Neural Networks]

1.2.1 Start from raw data

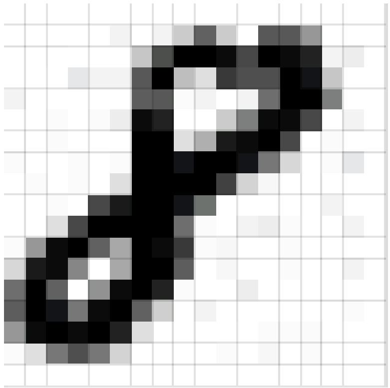


Figure 2-11. The raw data of handwriting

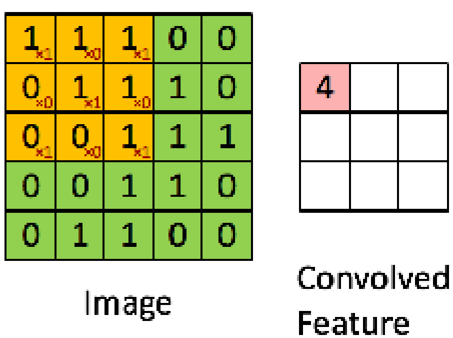
1.2.2 Convolution layer

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Figure 2-12. The raw data

1	0	1
0	1	0
1	0	1

Figure 2-13. The convolution kernel

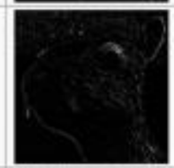


Key words:

- Convolution kernel
- Stride
- Zero-padding

Figure 2-14. The demonstration of convolution

1.2.2 Convolution layer

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

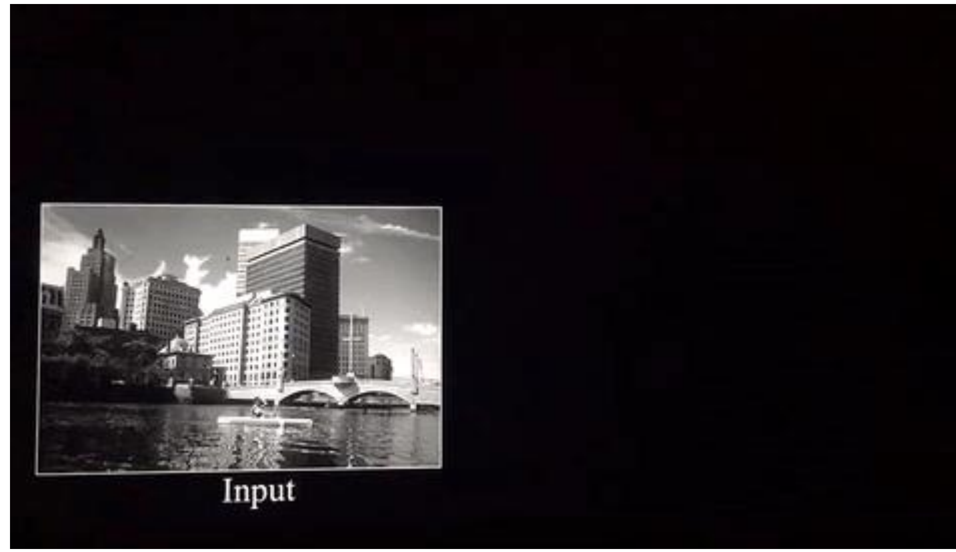


Figure 2-16. The demonstration of future extract

Figure 2-15. different feature filter (kernel)

1.2.3 The Pooling Step

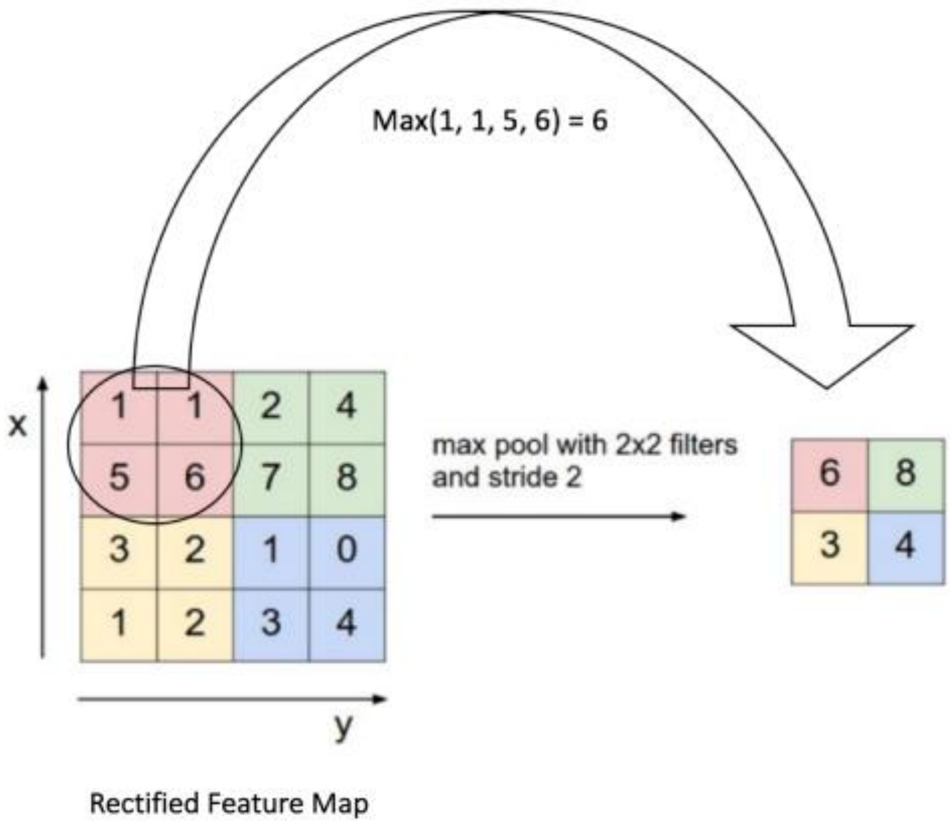


Figure 2-17. Max Pooling Source

1.2.3 Story so far

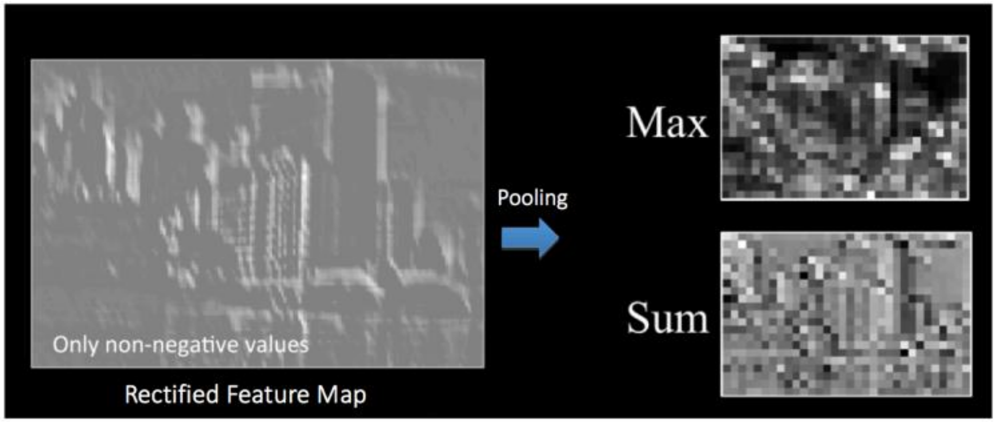


Figure 2-18. Pooling Source

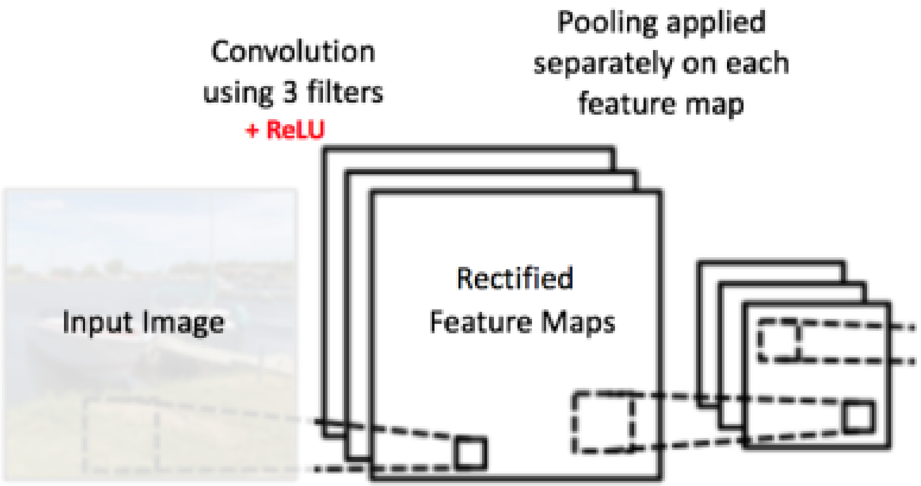


Figure 2-19. Pooling applied to Rectified Feature Maps

1.2.3 Convolution layer mathematics

数据挖掘实验室
Data Mining Lab

Replace fully connected mode by convolution layer:

$$a^{(l)} = f(w^{(l)} \otimes a^{(l-1)} + b^{(l)})$$

where $w^{(l)} \in \mathbb{R}^m$ is m-dimensional feature filter and share the same value for all neuros in the l^{th} convolution layer. So, we need only m+1 parameters. Usually, the number of neuro in the $(l+1)^{th}$ layer is designed as $n^{(l+1)} = n^{(l)} - m + 1$.

In l^{th} layer and k^{th} set feature map:

$$X^{(l,k)} = f(\sum_{p=1}^{n^{l-1}} (w^{(l,k,p)} \otimes X^{(l-1,p)}) + b^{(l,k)})$$

where $w^{(l,k,p)}$ is the map parameter of p^{th} set feature vector in $(l-1)^{th}$ layer to of l^{th} set feature vector in l^{th} layer.

1.2.3 Convolution layer mathematics

数据挖掘实验室
Data Mining Lab

For gradient computation, we assume that the l^{th} layer is the convolution layer and the down sample layer is placed at the $(l+1)^{th}$ layer.

To derivate the k^{th} error term in the l^{th} layer $\delta^{(l,k)}$:

$$\delta^{(l,k)} \triangleq \frac{\partial J(W,b;X,y)}{\partial Z^{(l,k)}}$$

$$= \frac{\partial X^{(l,k)}}{\partial Z^{(l,k)}} \cdot \frac{\partial Z^{(l+1,k)}}{\partial X^{(l,k)}} \cdot \frac{\partial J(W,b;X,y)}{\partial Z^{(l+1,k)}}$$

$$= f_l^{'}(Z^{(l)}) \odot (up(w^{(l+1,k)}\delta^{(l+1)}))$$

$$= w^{(l+1,k)}(f_l^{'}(Z^{(l)}) \odot up(\delta^{(l+1)}))$$

1.2.3 Pooling layer mathematics

After the convolution layer, we get a feature map $X^{(l)}$. Divide $X^{(l)}$ into a series of areas R_k. k=1,…,K

$$X_k^{l+1} = f(Z_k^{l+1})$$

$$= f(w^{l+1} \cdot down(R_k) + b^{l+1})$$

therefore,

$$X^{l+1} = f(w^{l+1} \cdot down(X^l) + b^{l+1})$$

Usually, down sample function down(·) is Maximum Pooling or Average Pooling.

$$pool_{\max}(R_k) = \max_{i \in R_k} a_i$$

$$pool_{avg}(R_k) = \frac{1}{|R_k|} \sum_{i \in R_k} a_i$$

1.2.3 Pooling layer mathematics

For gradient computation, we assume that the l^{th} layer is the down sample layer and the convolution layer is placed at the $(l+1)^{th}$ layer.

$$Z^{(l+1,k)} = \sum_{p,T_{p,k}=1} (W^{(l+1,k,p)} \otimes X^{(l,p)}) + b^{(l+1,k)}$$

To derivate the k^{th} error term in the l^{th} layer $\delta^{(l,k)}$:

$$\delta^{(l,k)} \triangleq \frac{\partial J(W,b;X,y)}{\partial Z^{(l,k)}}$$

$$= \frac{\partial X^{(l,k)}}{\partial Z^{(l,k)}} \cdot \frac{\partial Z^{(l+1,k)}}{\partial X^{(l,k)}} \cdot \frac{\partial J(W,b;X,y)}{\partial Z^{(l+1,k)}}$$

$$= f_l^{'}(Z^{(l)}) \odot \left(\sum_{p,T_{p,k}=1} (\delta^{(l+1,p)}) \otimes rot180(W^{(l,k,p)}) \right)$$

1.2.4 Classifier

- Full Connected Layer

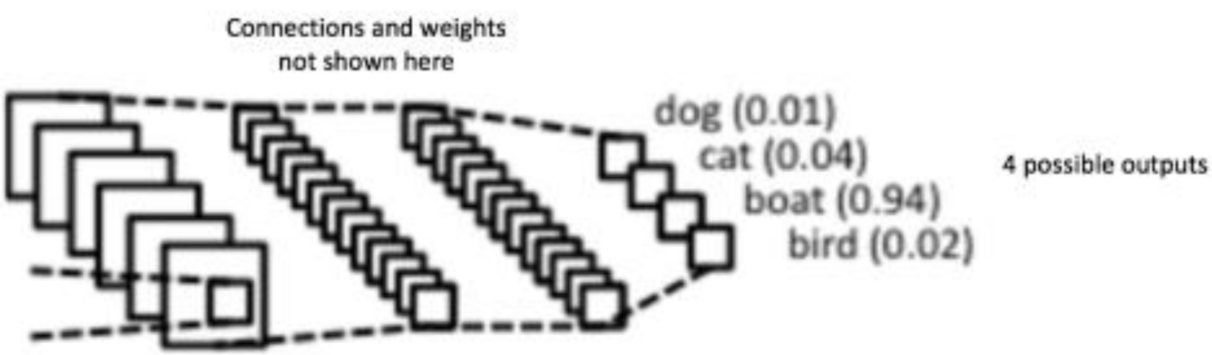


Figure 2-20. Full Connected Layer

- Global Average Pooling + Softmax

1.2.5 Think twice

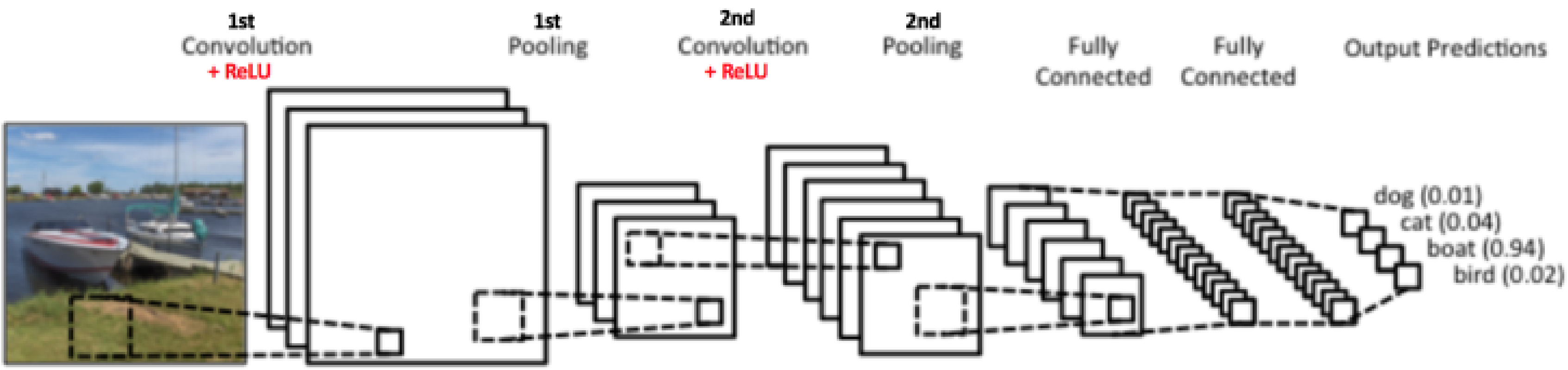


Figure 2-21. The architecture of LeNet

- Why is it successful?
- Why should it be deeper?
- Is that OK?

1.2.5 Think twice

- Why is it successful?
  - Neuroscience
    - ✓ receptive field in vision area
    - ✓ Top-down processing
    - ✓ Hierarchical structure
  - Five space transformation operations
    - ✓ increasing/reducing dimensionality
    - ✓ Zooming
    - ✓ Rotating
    - ✓ Translation
    - ✓ Bending
  - Dropout

1.2.5 Think twice

- Why is it successful?

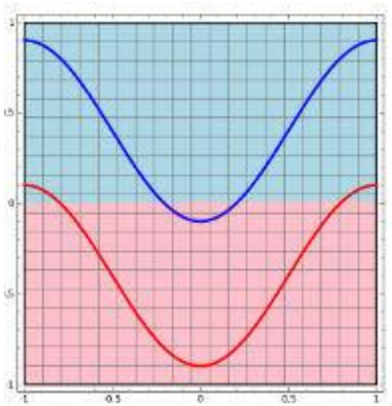


Figure 2-22.
Raw data space

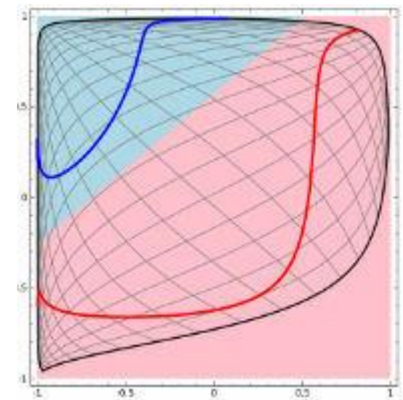


Figure 2-23.
Transformed data space

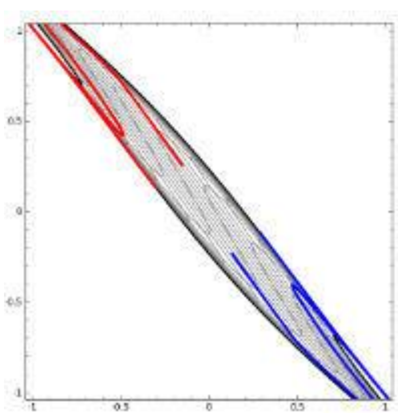


Figure 2-24.
Stronger transformation

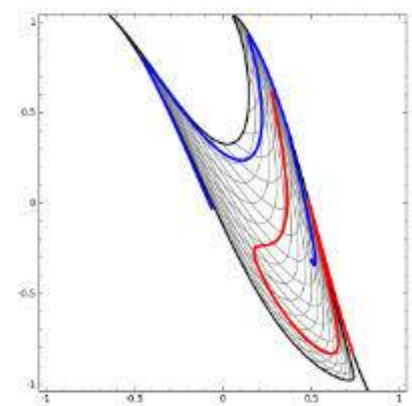


Figure 2-25.
Failed transformation

1.2.5 Think twice

- Why should it be deeper?

  The most direct explanation is that the number of critical point varies with the size of network which is proportional to

$$\sqrt{width} \times (depth)^{width/2}$$

  Therefore, the increase of width leads to explosive critical points while the increase of depth outcomes a slower increasing.

1.2.5 Think twice

- Is that OK?

2.1 Recurrent Neural Networks

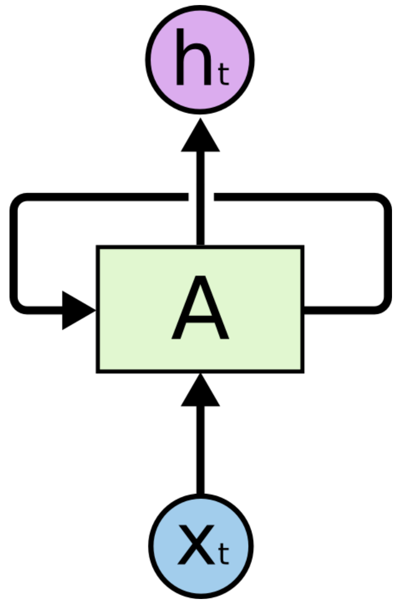


Figure 2-26. Recurrent Neural Networks

Understanding LSTM Networks

2.1 Recurrent Neural Networks

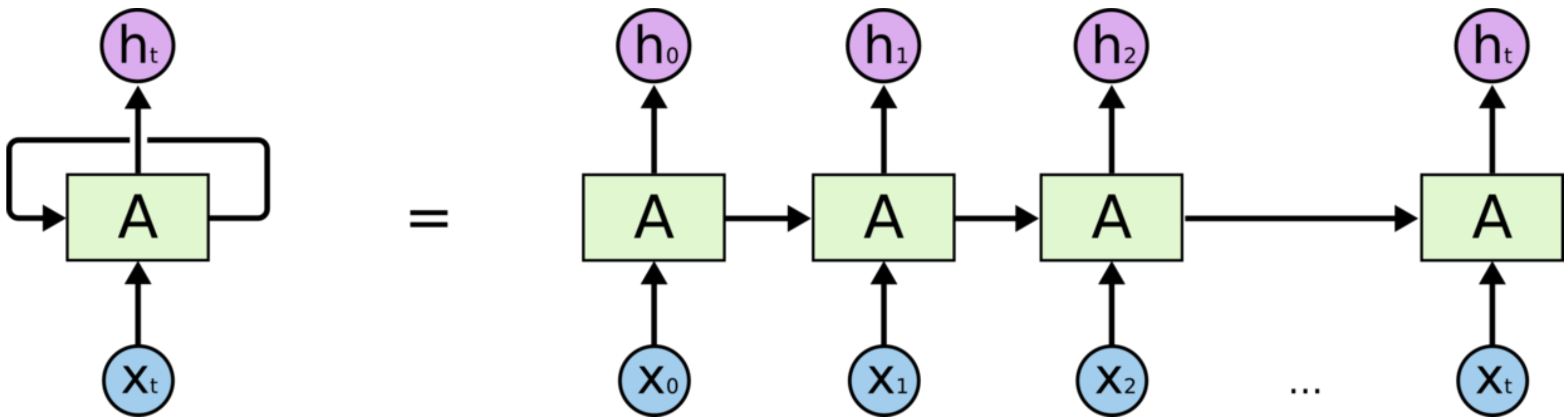


Figure 2-27. An unrolled recurrent neural network

2.1 Long-Term Dependency Problem

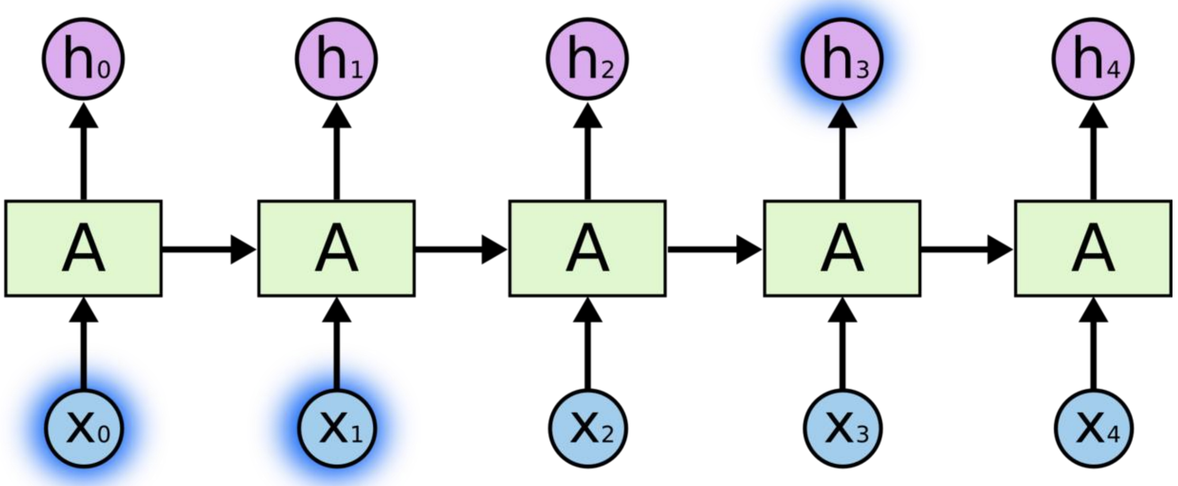


Figure 2-28. Short-term dependency

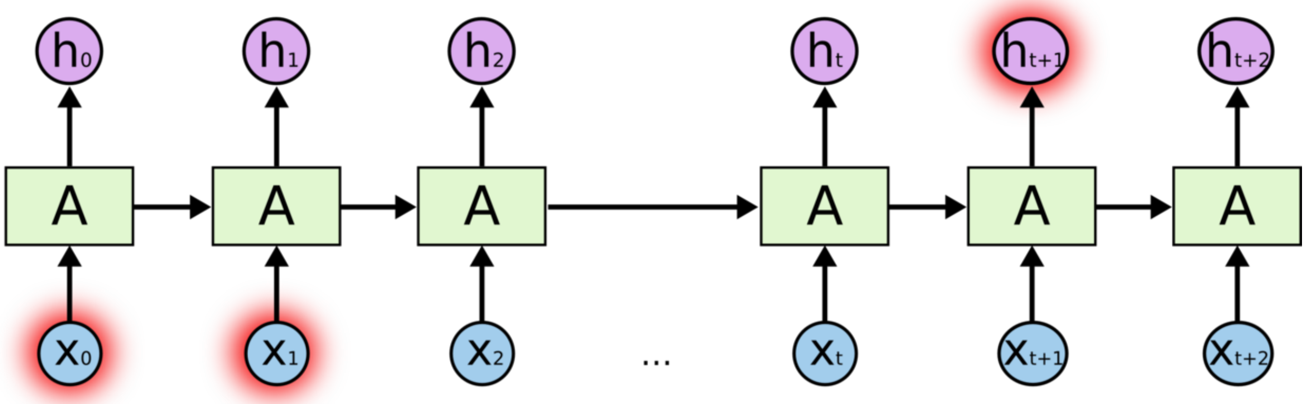


Figure 2-29. Long-term dependency

2.2 Long Short Term (LSTM)

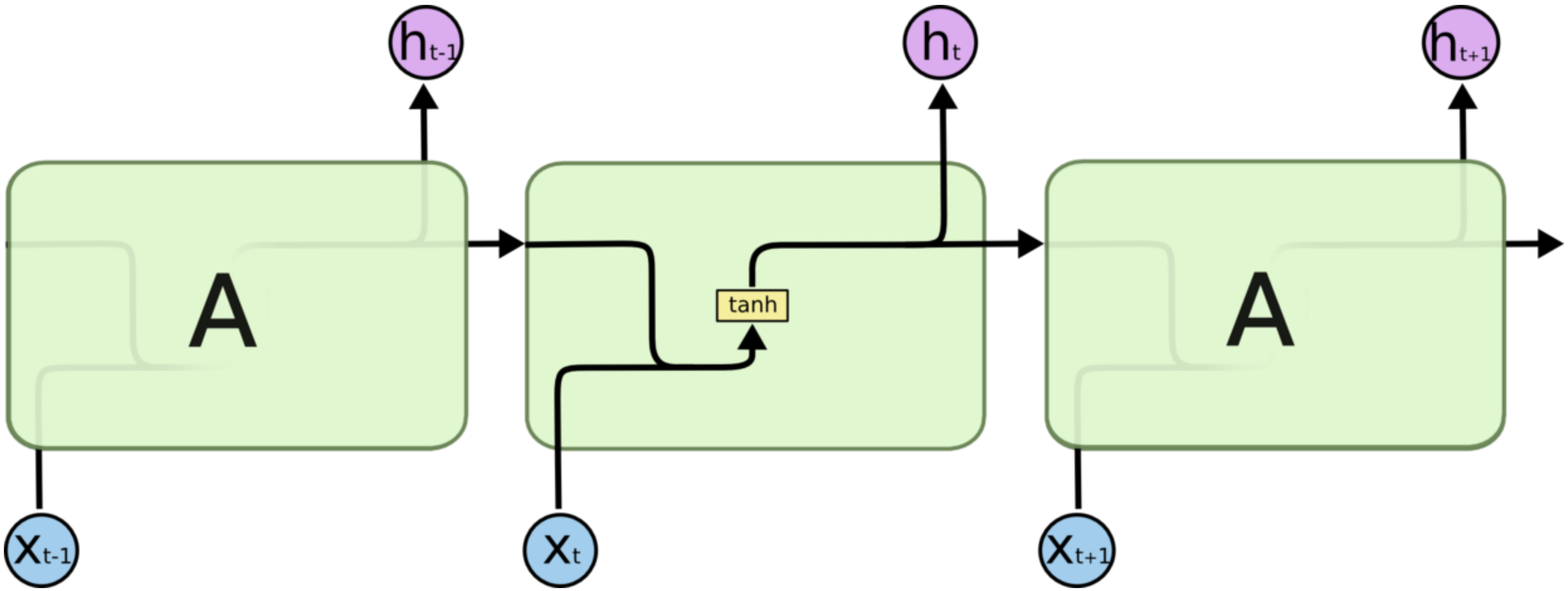


Figure 2-30. Standard RNN with single layer

2.2 Long Short Term (LSTM)

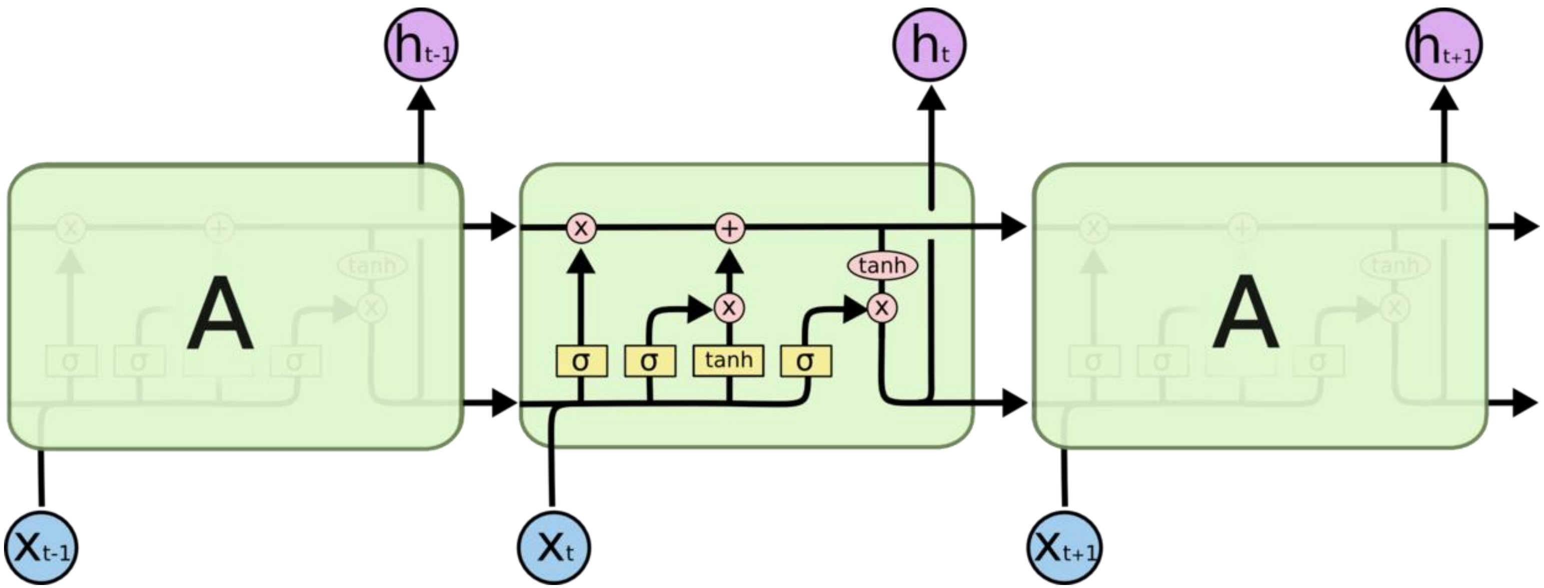


Figure 2-31. Well designed Long Short Term

2.2 Illustrate LSTM Structure

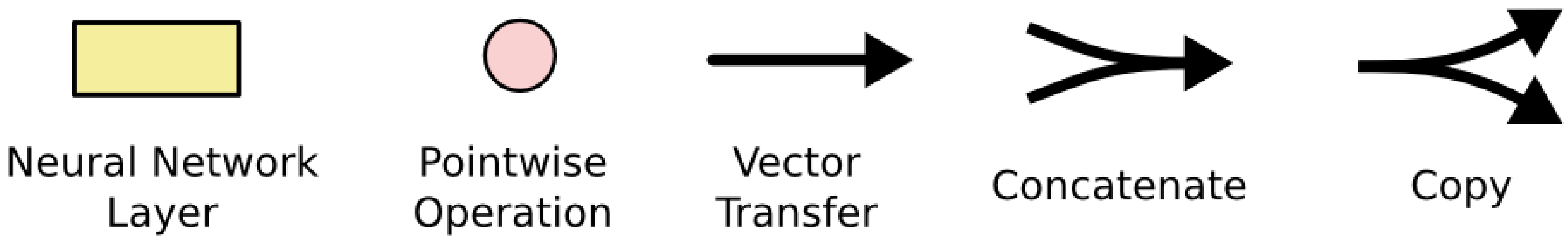


Figure 2-32. Corresponding meaning of diagram

2.2 Core Idea behind LSTM

数据挖掘实验室
Data Mining Lab

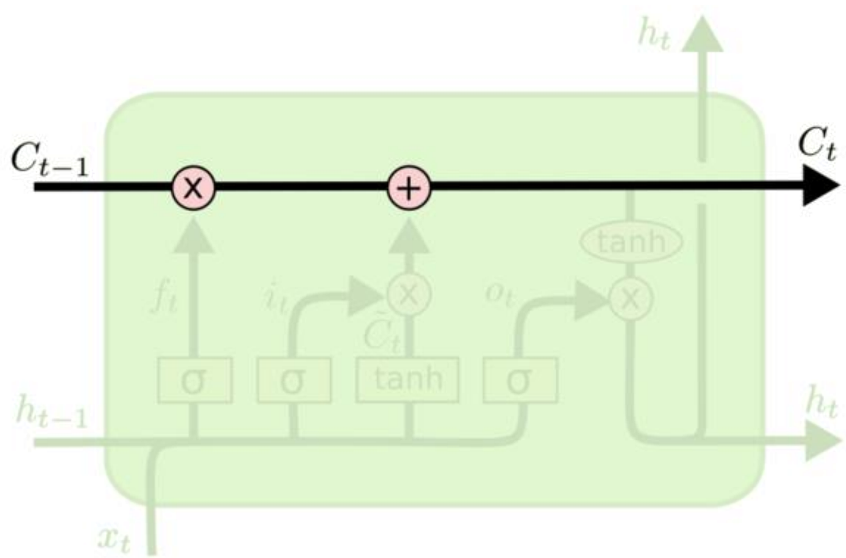


Figure 2-33. the cell state horizontally running through the top of the diagram

2.2 Illustrate LSTM Structure

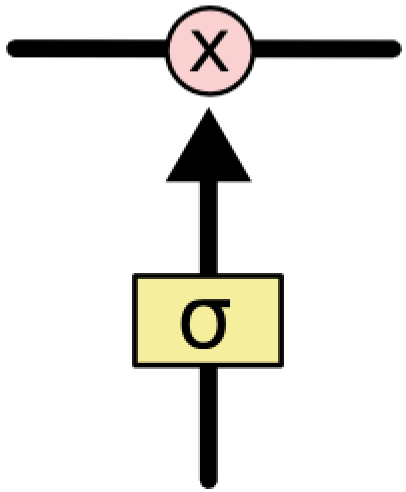
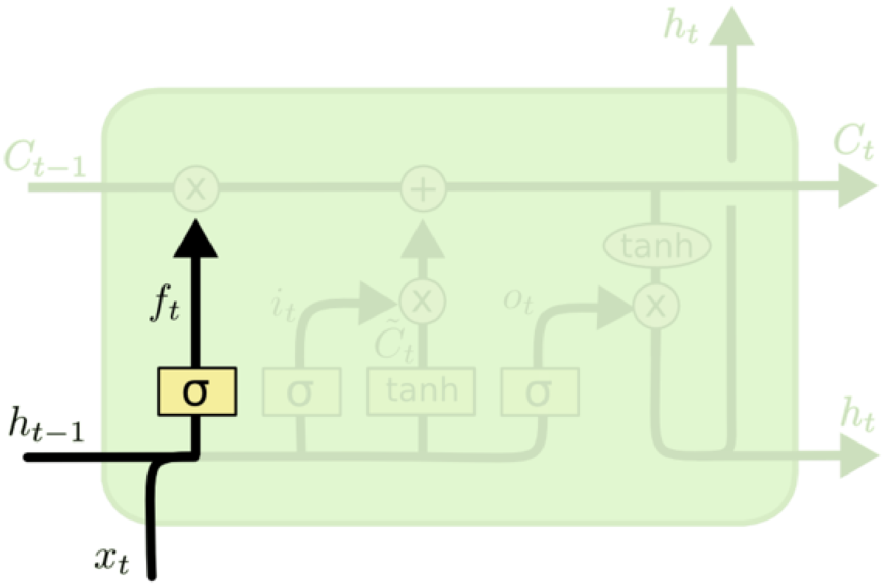


Figure 2-34. The gates structures

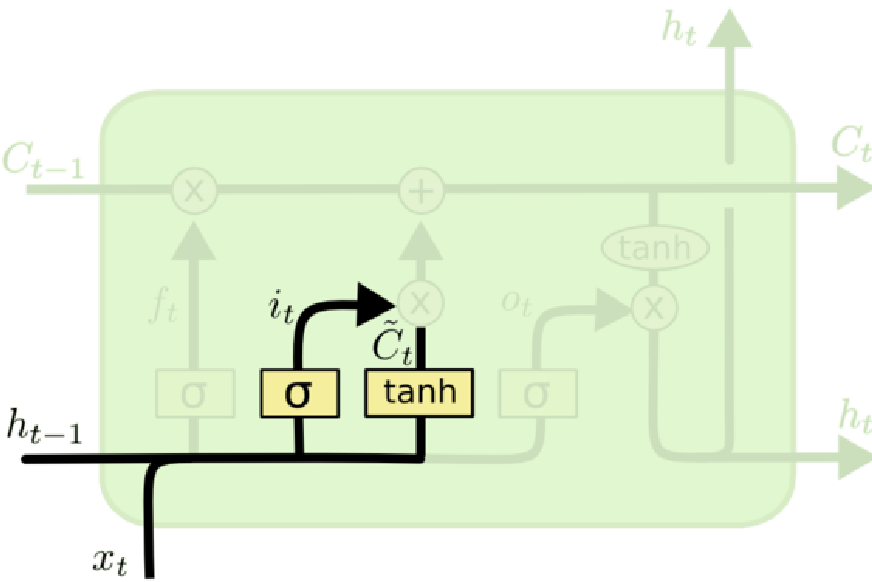
2.2 Illustrate LSTM Structure



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Figure 2-35. The forget gate layer of LSTM

2.2 Illustrate LSTM Structure

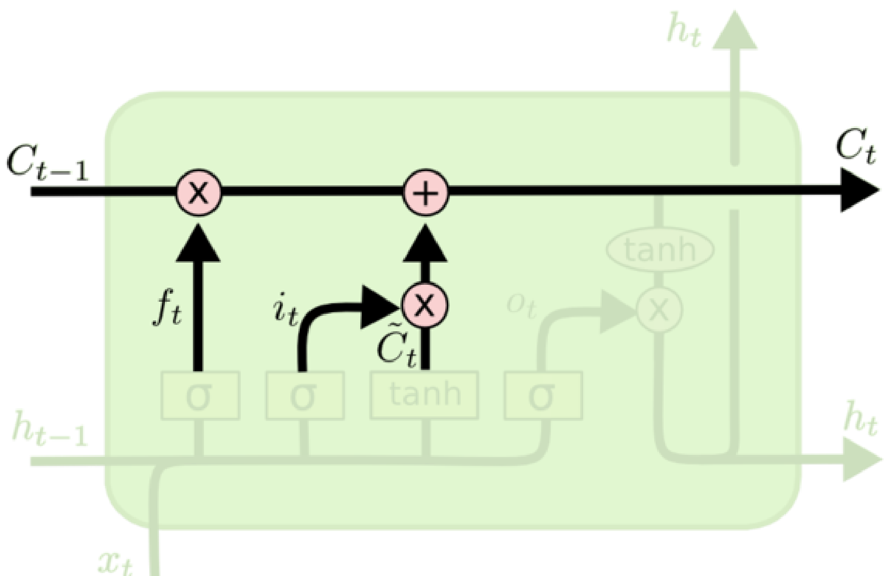


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Figure 2-36. The input gate layer of LSTM

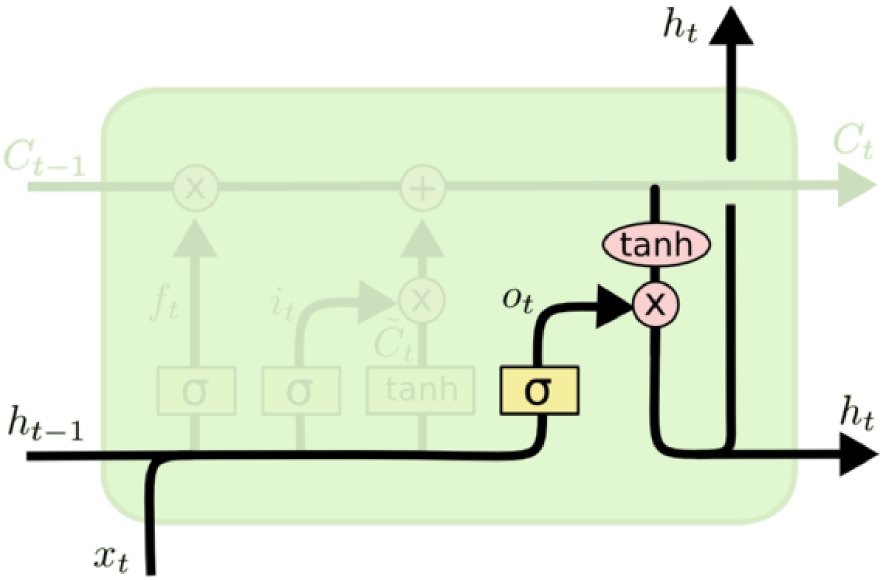
2.2 Illustrate LSTM Structure



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 2-37. The implement gate layer of LSTM

2.2 Illustrate LSTM Structure



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 2-38. The output gate layer of LSTM

2.2 Long Short Term (LSTM)

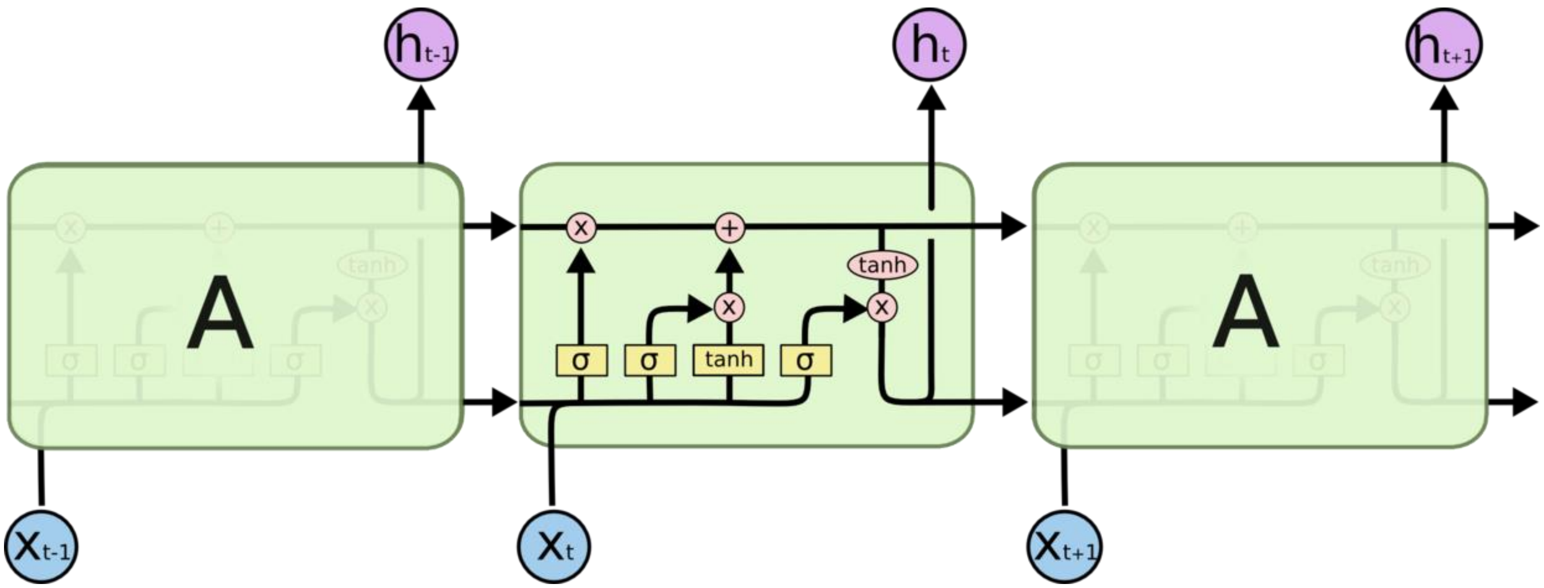
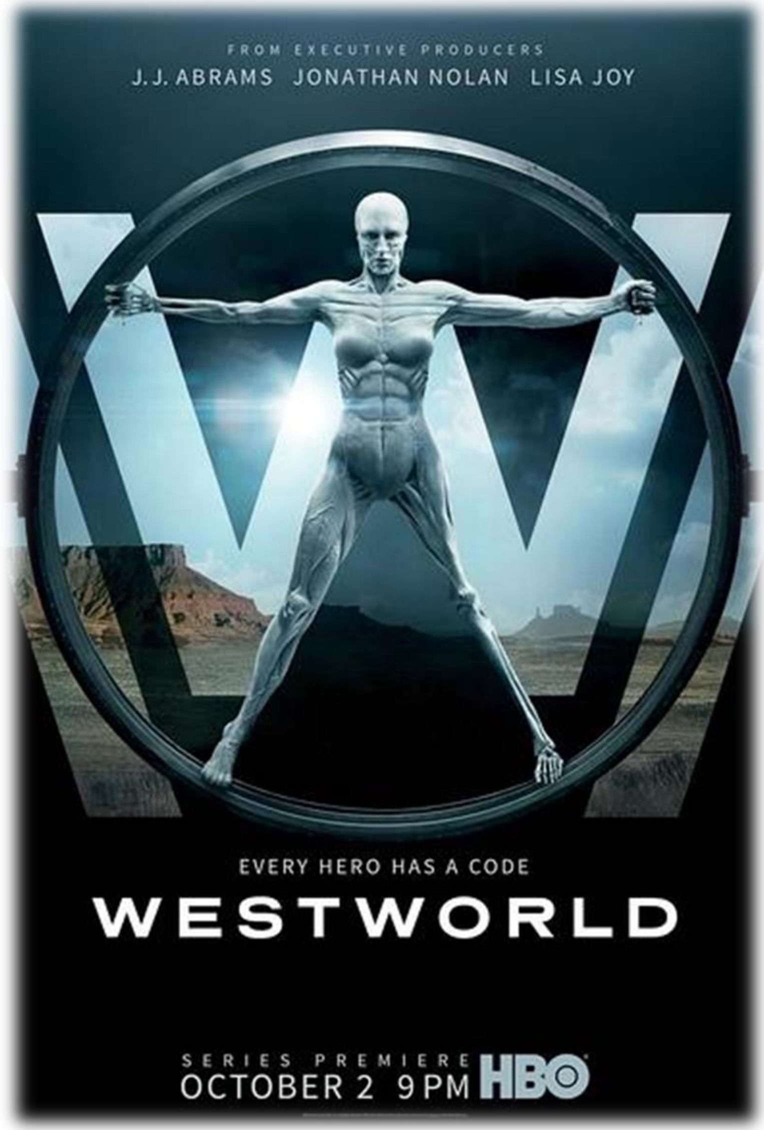


Figure 2-39. Well designed Long Short Term

Message

We are in the best of times.

How about the top of tides!



Thanks