Big Data Technology and its Applications

Neural Network

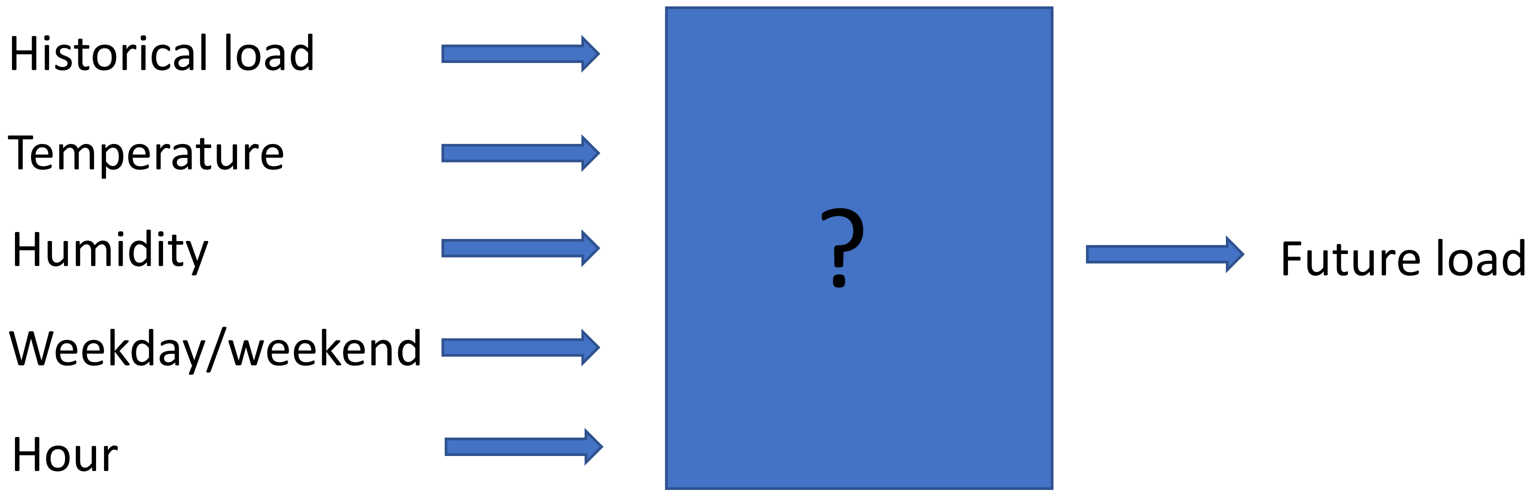
张宁 ningzhang@tsinghua.edu.cn

Outline

- The example of load forecasting
- Feed forward neural network
- Back propagation training
- Convolutional neural network
- Recurrent neural network
- Load forecasting using LSTM

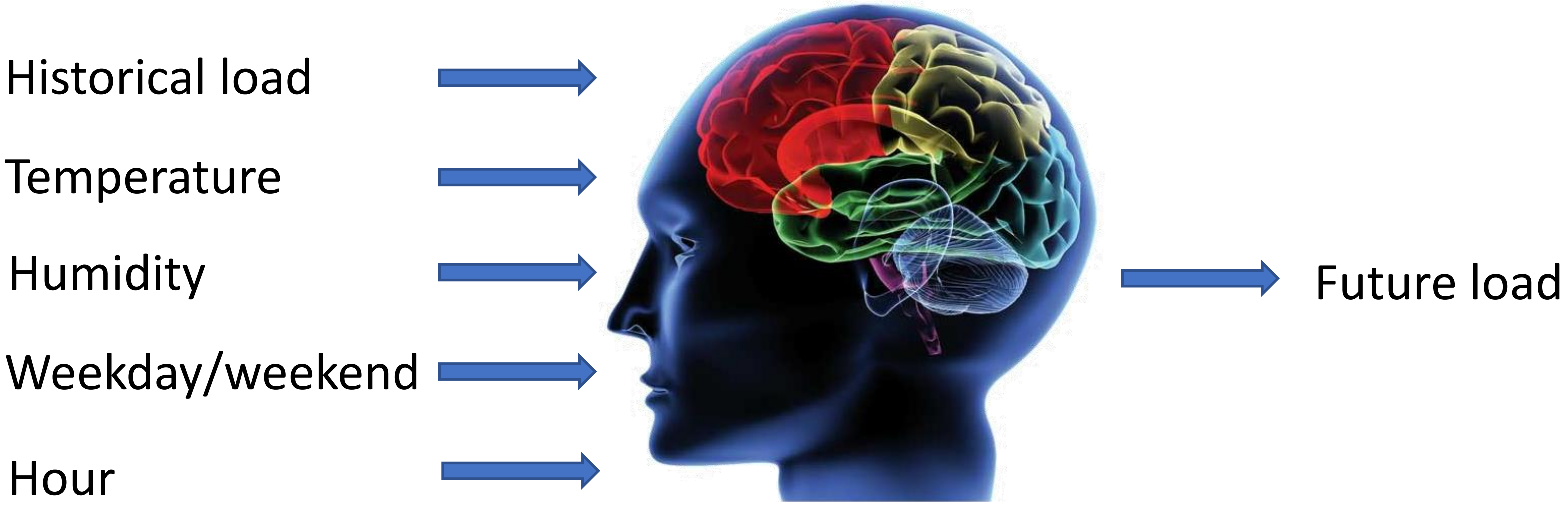
An example — load forecasting

- How to forecast the load of the power system given the historical data?
- The forecasting results of future loads are very important to the operation of power systems.



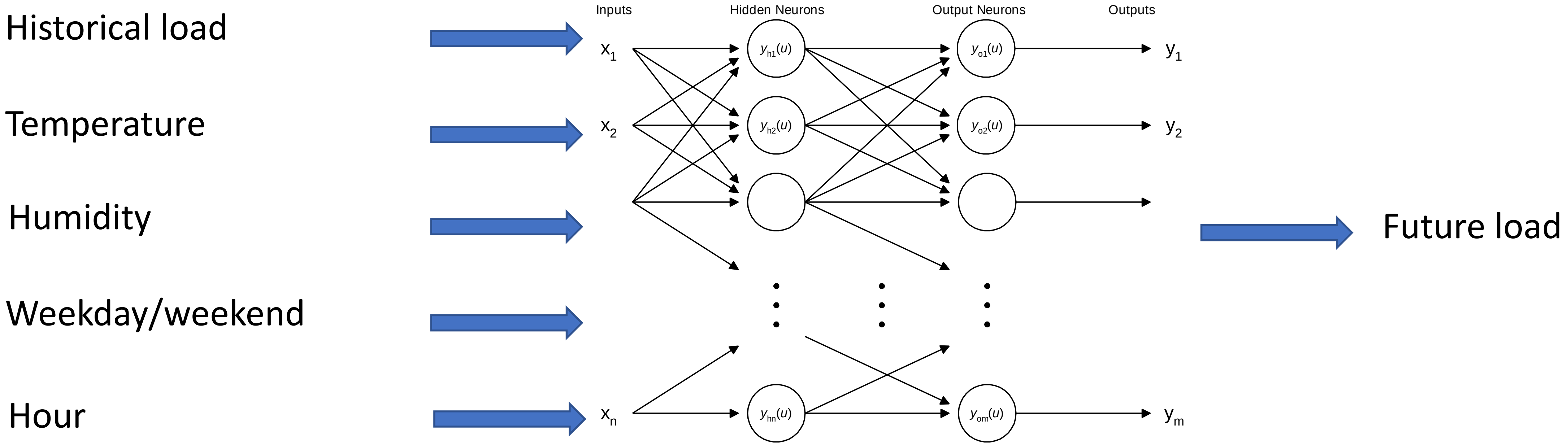
An example — load forecasting

- One may use human experience to forecast future loads

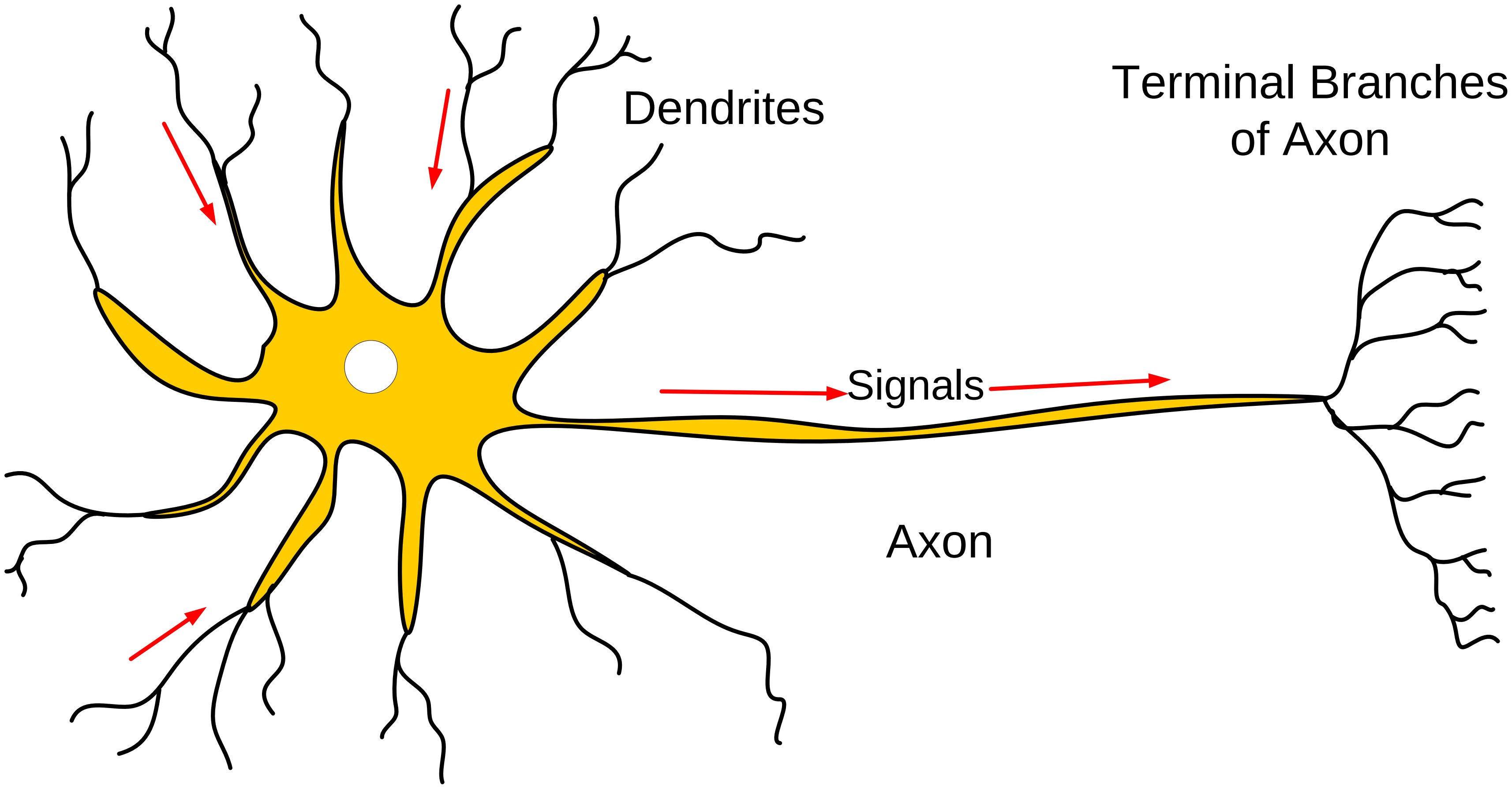


An example — load forecasting

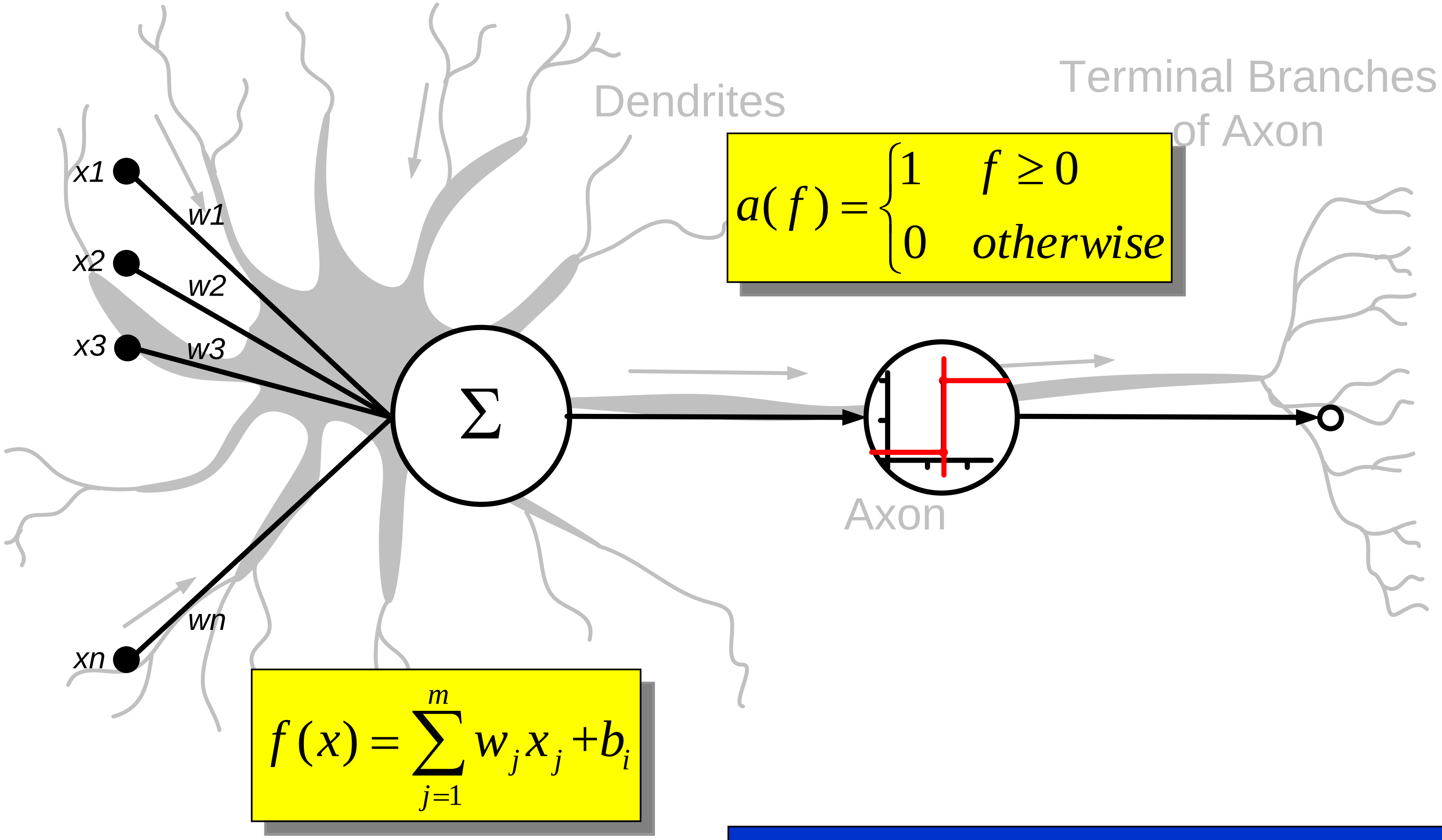
- We can also use artificial neural networks to replace human brains.



Feed forward neural network



Feed forward neural network



McCulloch and Pitts[1943]

Feed forward neural network

$$f(z) = \frac{1}{1+e^{-\lambda z}}$$

Activation function

Why we need activation function?

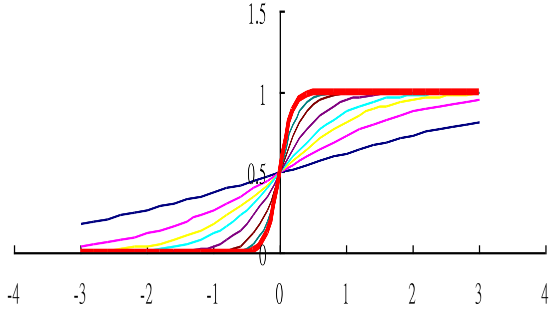
-sigmoid

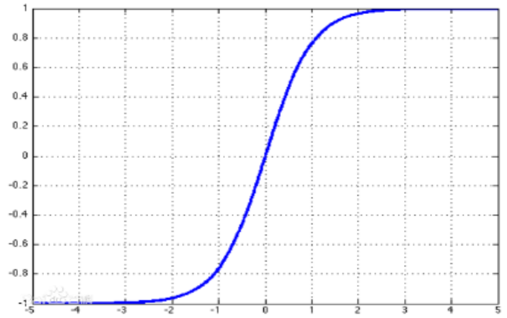
The outputs are between (0, 1)

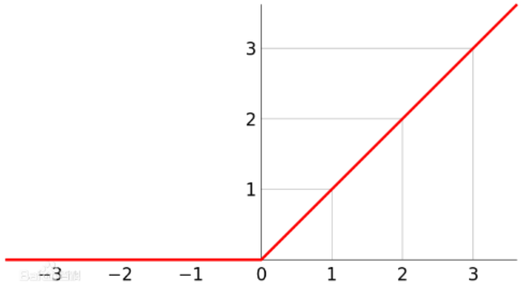
The derivative is $f'(z) = f(z)(1 - f(z))$

-tanh

-relu







Feed forward neural network

- Single input neuron

Scalar input: x
Scalar weight: w
Bias: b

Net input: e
Activation function: $f(\mathrm{e}) = \dfrac{1}{1+e^{-\mathrm{e}}}$
Output: y

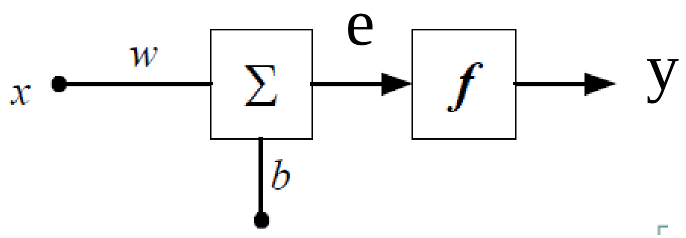
$$\mathrm{e} = wx + b$$

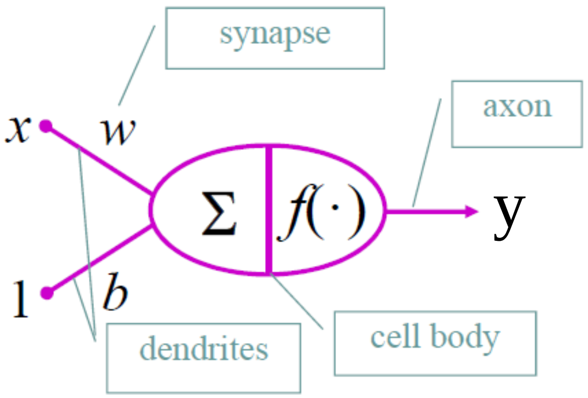
$$y = f(\mathrm{e}) = f(wz + b)$$

$$w = 3, x = 2, b = -1.5$$

$$\mathrm{e} = 3 \times 2 - 1.5 = 4.5$$

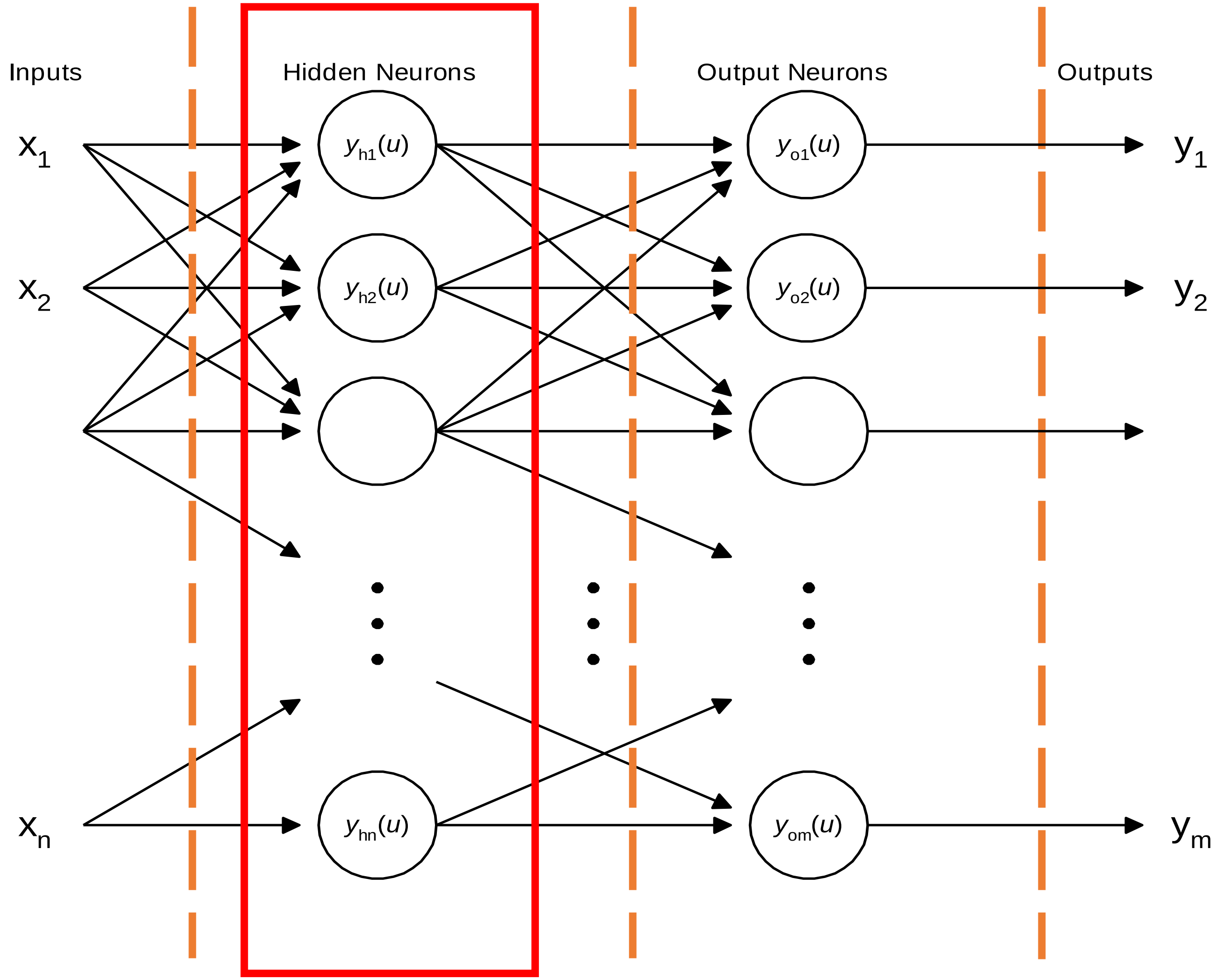
$$y = f(4.5) = \frac{1}{1+e^{-\mathrm{e}}} = 0.989$$





Feed forward neural network

- Multi-layer perceptron

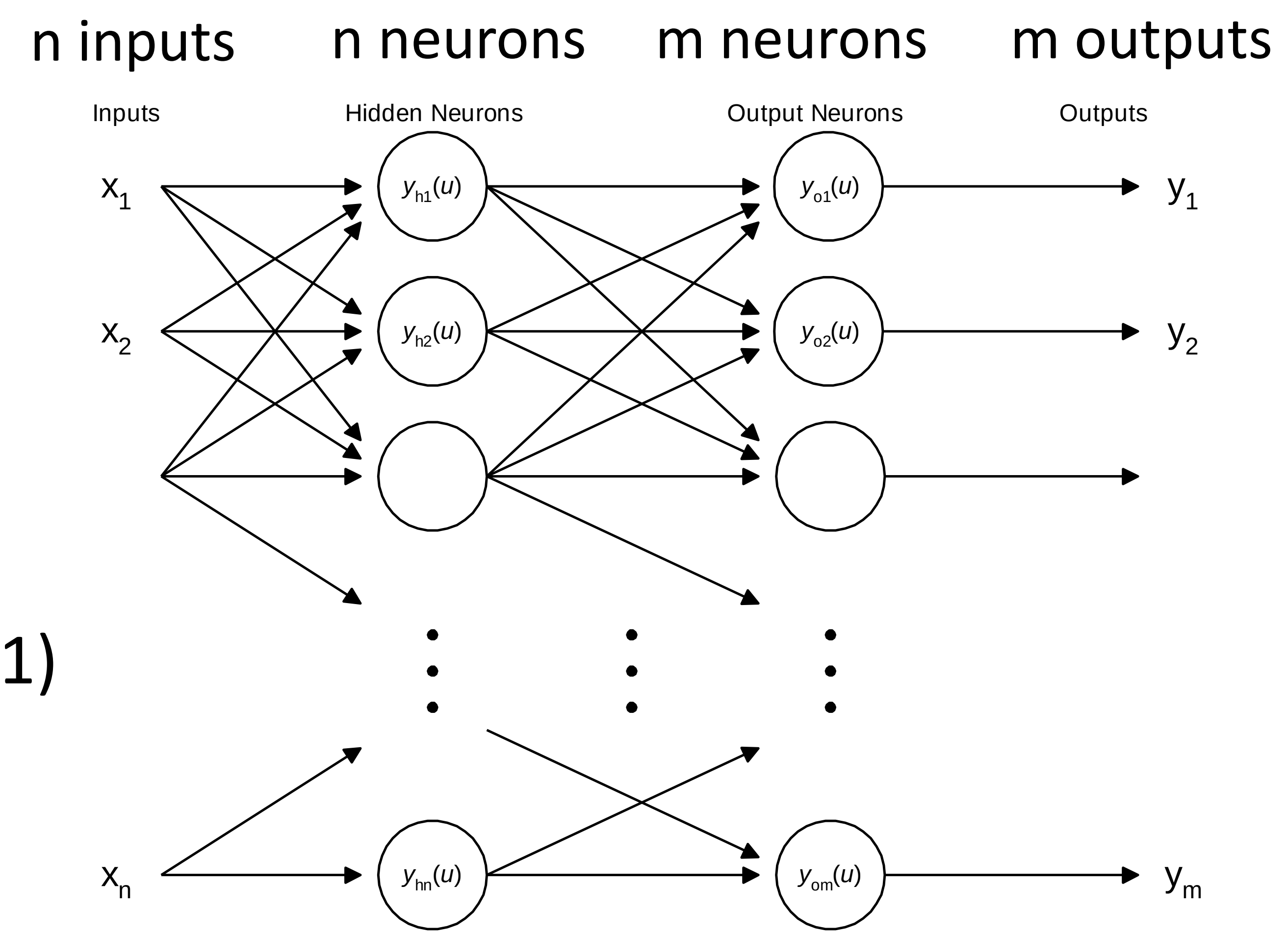


单选题 1分

Setting

The number of parameters to be trained:

$$y = f\left(\sum_{j=1}^{m} w_j x_j + b_i\right)$$



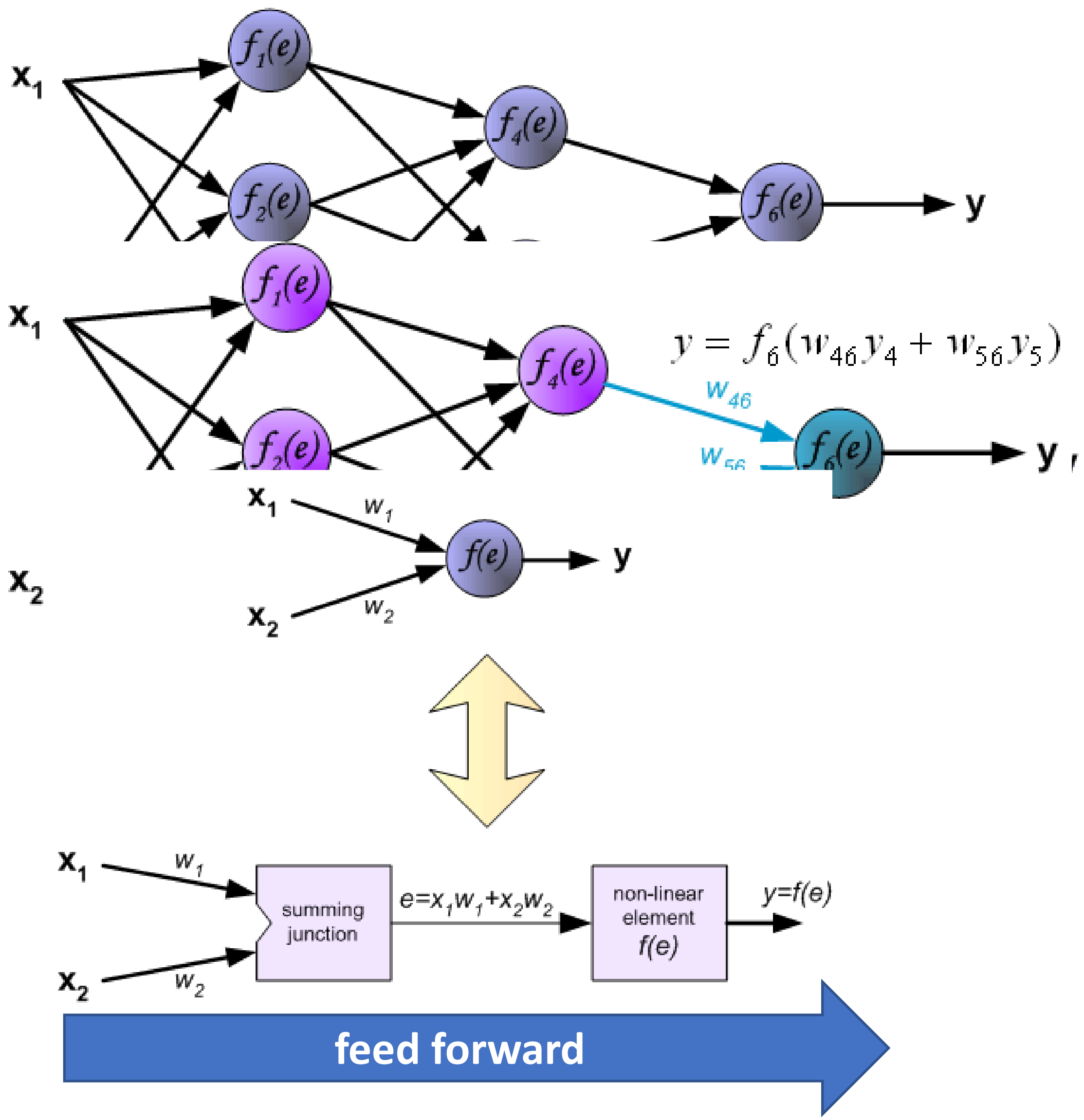
A n*n+ m*n

B n*(n+1)+m*(n+1)

C 2*n*m

Submit

Back propagation training



Back propagation training

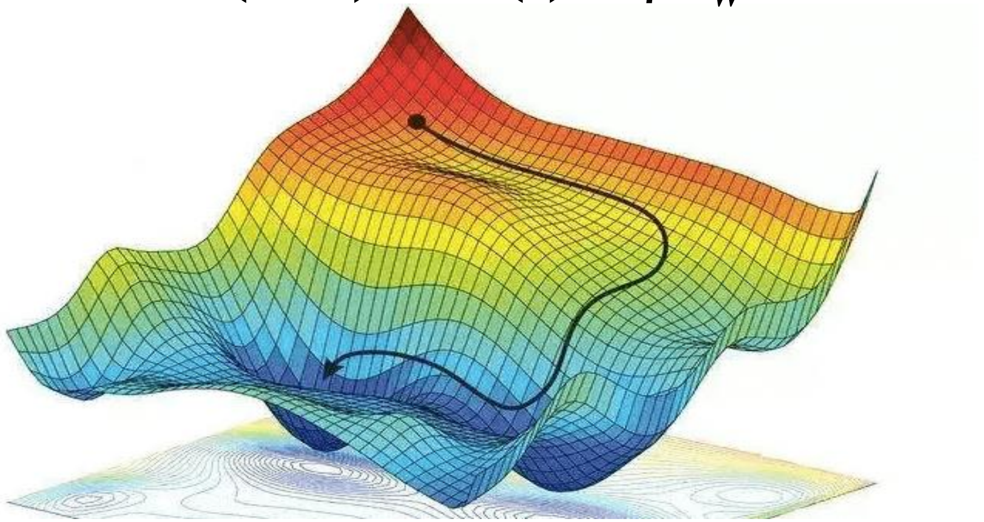
How to train the weights $\mathcal{W} = \{\mathbf{W_i}\}_{i=1,\ldots,l}$ so that the outputs can best fit the real outputs?

We need to define a loss function L to evaluate how "well" the weights are, for example, we can use the sum-of-squares as the loss function.

$$L = \frac{1}{2}\sum_{i=1}^{N}\left(y_i - z_i\right)^2$$

We then use the gradient-based optimizer, such as the gradient descent:

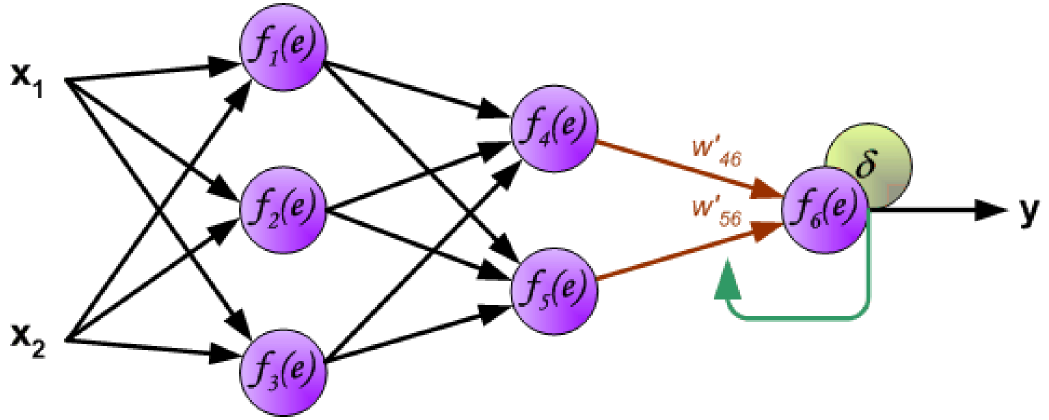
$$W(t+1) = W(t) - \eta \nabla_W L$$



Back propagation training

But how to calculate the gradient of the loss function $\nabla_W L$?

We use the back propagation technique.



$$w'_{46} = w_{46} - \eta \frac{\partial L}{\partial w_{46}}$$

$$w'_{56} = w_{56} - \eta \frac{\partial L}{\partial w_{56}}$$

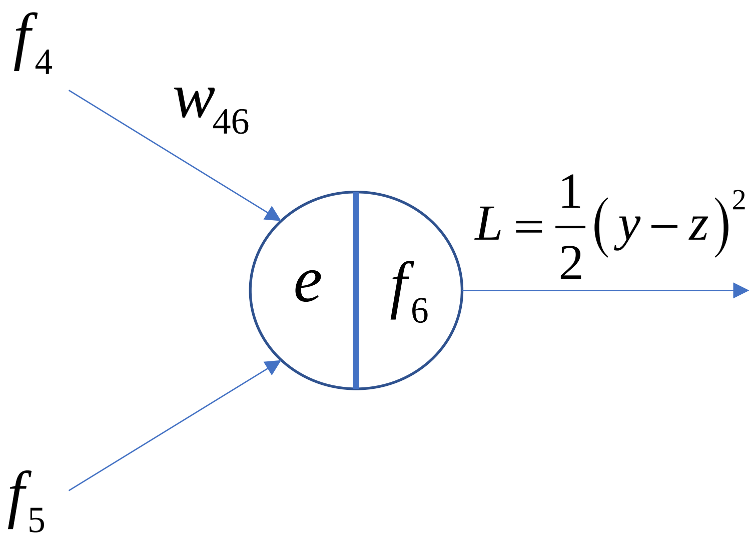
We define $\delta_i^{(l)}$ measures how much node i in layer l is "responsible" for any errors of the network's output.

$$\delta_i^{(l)} = \frac{\partial L}{\partial e_i^{(l)}}$$

For output layer in the above figure, we have:

$$\delta = \frac{\partial L}{\partial e_6} = \frac{\partial L}{\partial f_6}\frac{\partial f_6}{\partial e_6} = \ (y-z)\ \ f_6\left(1 - f_6\right)$$

Back propagation training



$$\frac{\partial L}{\partial w_{46}} = \frac{\partial L}{\partial e_6}\frac{\partial e_6}{\partial w_{46}}$$

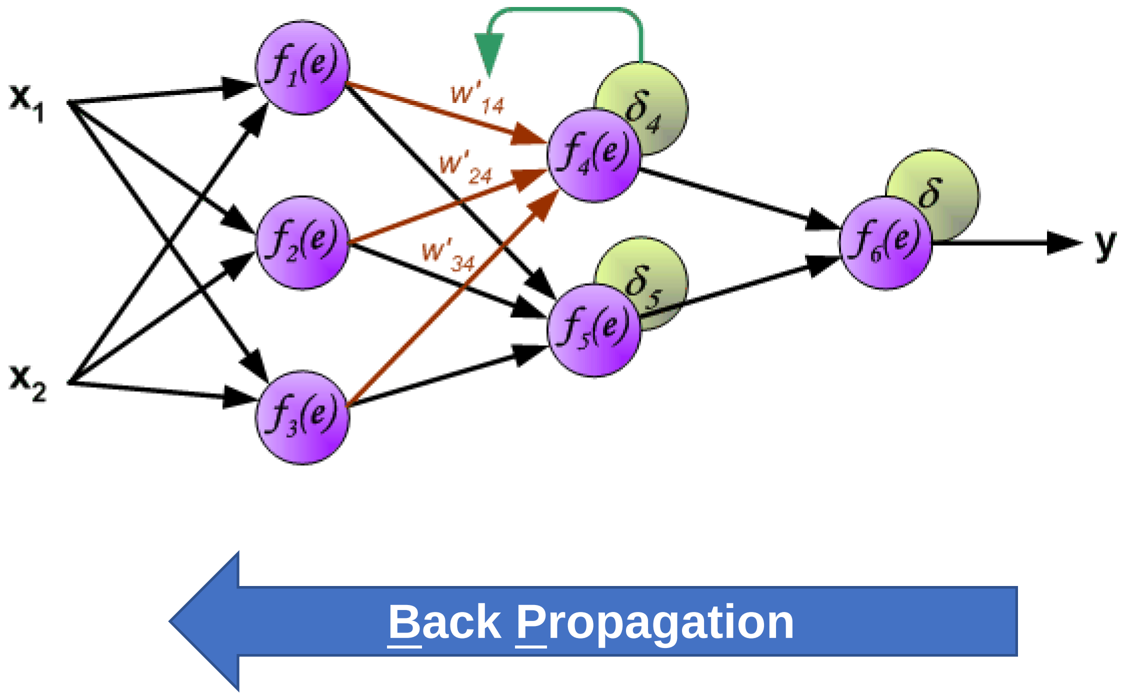
$$\frac{\partial L}{\partial e_6} = \delta = (y-z)\ f_6(1-f_6)$$

$$\frac{\partial e_6}{\partial w_{46}} = f_4$$

$$\frac{\partial L}{\partial w_{46}} = (y-z)\ f_6(1-f_6)\, f_4 = \delta f_4$$

Back propagation training

$$\delta_4 = \frac{\partial L}{\partial e_4} = \frac{\partial L}{\partial e_6}\frac{\partial e_6}{\partial f_4}\frac{\partial f_4}{\partial e_4} = \delta w_{46} f_4 (1 - f_4)$$



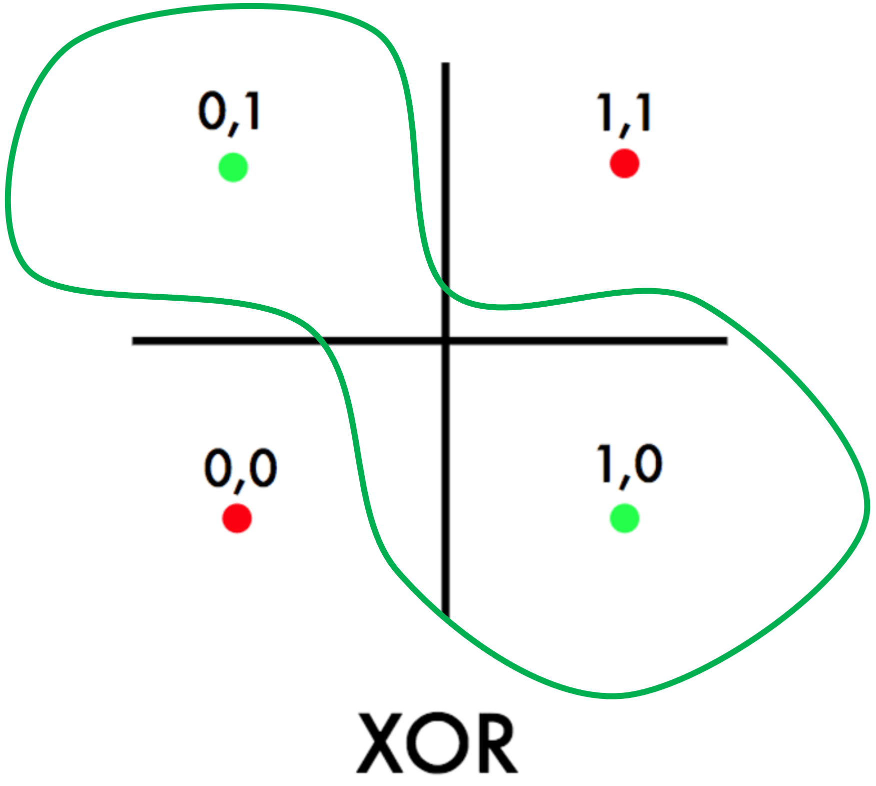
$$w'_{14} = w_{14} - \eta \frac{\partial L}{\partial w_{14}}$$

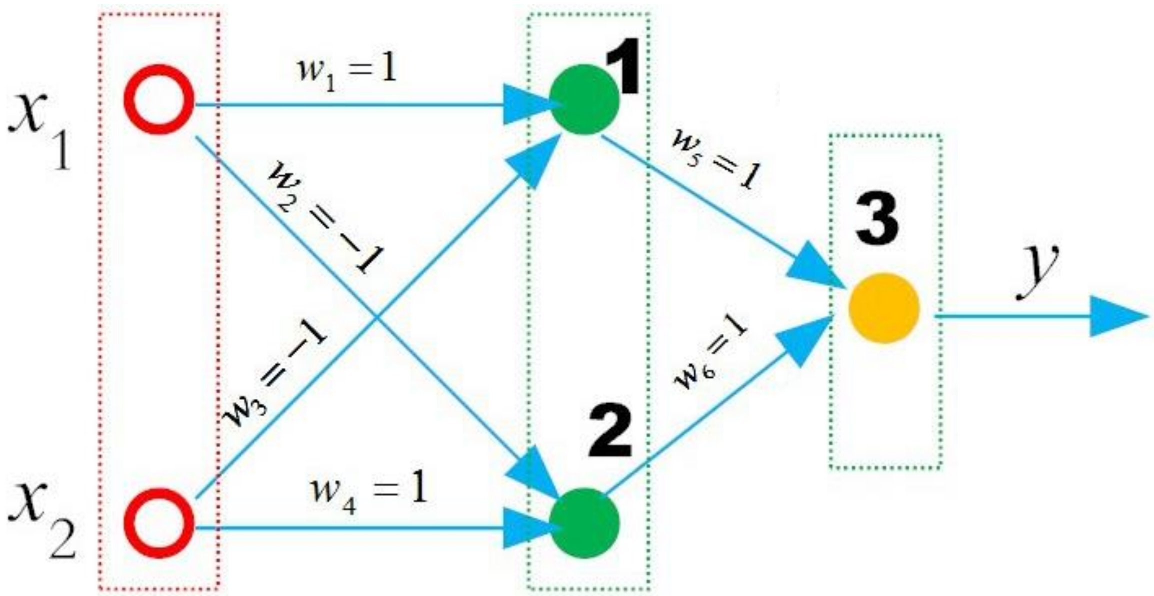
$$w'_{24} = w_{24} - \eta \frac{\partial L}{\partial w_{24}}$$

$$w'_{34} = w_{34} - \eta \frac{\partial L}{\partial w_{34}}$$

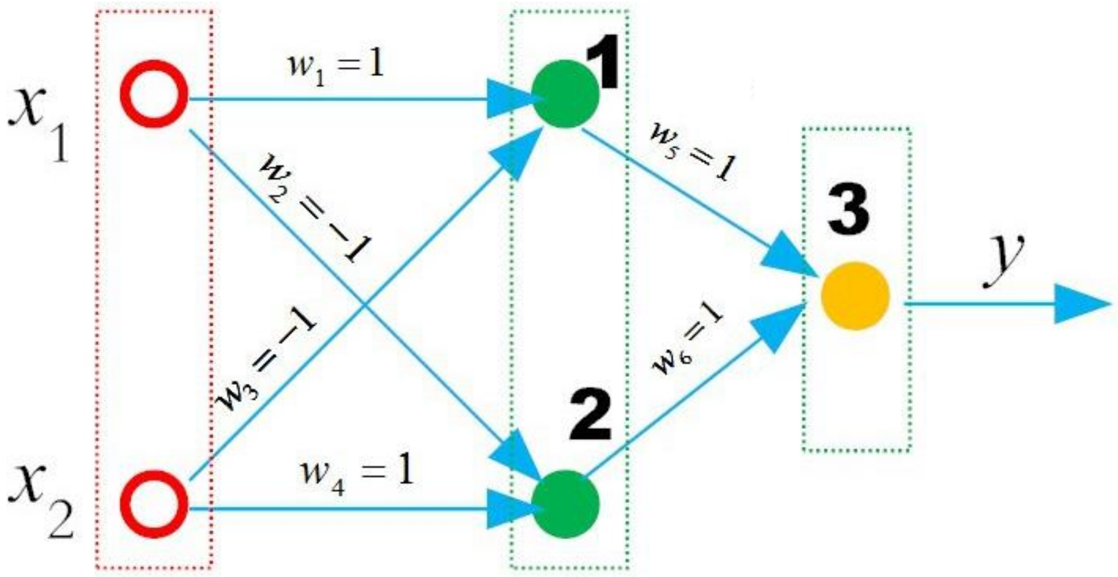
How to calculate?

Non-linear classification example: y=XOR(x1,x2)





Non-linear classification example: XOR



$for\ x_1 = 1,\ x_2 = 1,\ y = 0$

$$\begin{aligned} f_1 &= \operatorname{sgn}\left(x_1 w_1 + x_2 w_2 - \theta\right) \\ &= \operatorname{sgn}(1\times 1 + 1\times(-1) - 0.5) \\ &= \operatorname{sgn}(-0.5) \\ &= 0 \end{aligned}$$

$$\begin{aligned} f_2 &= \operatorname{sgn}\left(x_1 w_3 + x_2 w_4 - \theta\right) \\ &= \operatorname{sgn}(1\times(-1) + 1\times 1 - 0.5) \\ &= \operatorname{sgn}(-0.5) \\ &= 0 \end{aligned}$$

$$\begin{aligned} y = f_3 &= \operatorname{sgn}\left(f_1 w_5 + f_2 w_6 - \theta\right) \\ &= \operatorname{sgn}(0\times 1 + 0\times(1) - 0.5) \\ &= \operatorname{sgn}(-0.5) \\ &= 0 \end{aligned}$$

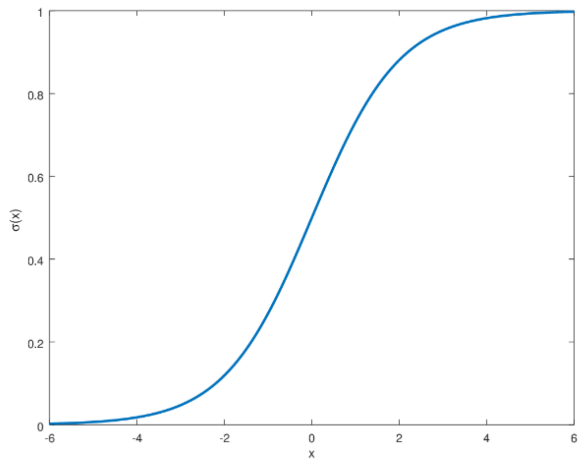
https://zhuanlan.zhihu.com/p/42041080

Back propagation training

Problem: gradient vanishing

The gradient of sigmoid function

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$



sigmoid

$\sigma'(x) \in (0, 1)$, if network is deep, the gradient decrease after error signal backpropagating through several layers.

When sigmoid function is saturated (i.e. is sufficiently large), the gradient is near zero. This means saturated neurons “kill” the gradients. This makes it hard to train deep networks.

Back propagation training

This is one of the reasons that ReLU is proposed.

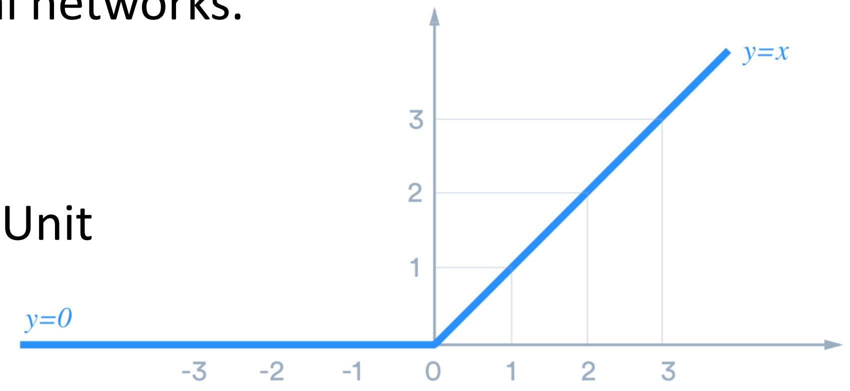
ReLU will never be saturated, and its gradient is always 1, when x>0

ReLU in backpropagation serves as a gradient switch:

$$\mathrm{ReLU}'(x) = \begin{cases} 1 & x > 0 \\ 0 & x < 0 \end{cases}$$

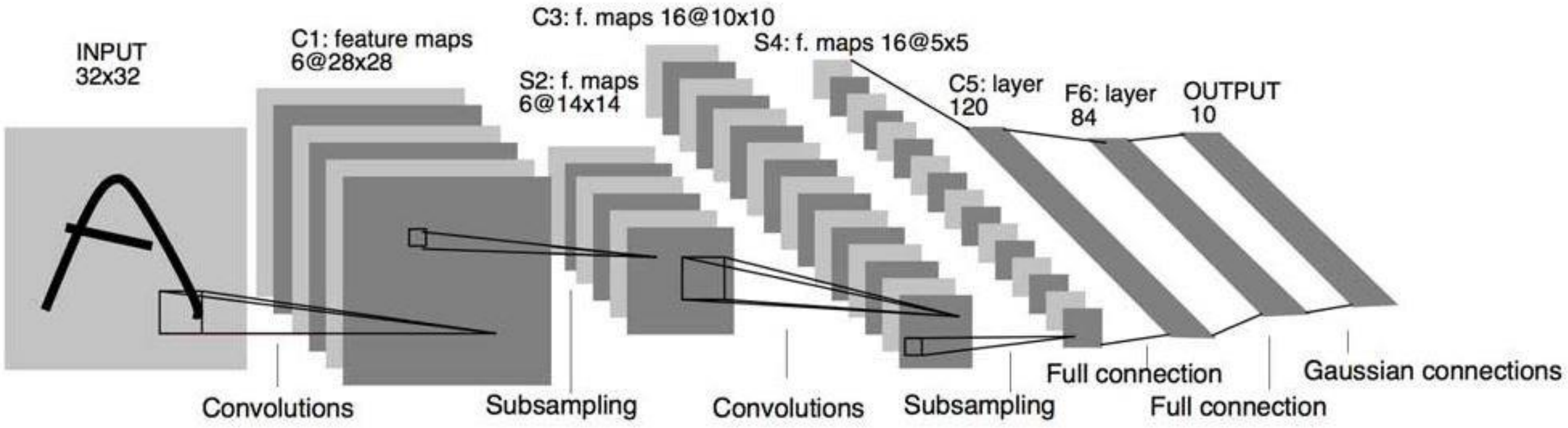
ReLU is widely-used in recent deep neural networks.

ReLU:
Rectified Linear Unit



https://blog.csdn.net/c123_sensing/article/details/81531519

Convolutional Neural Network



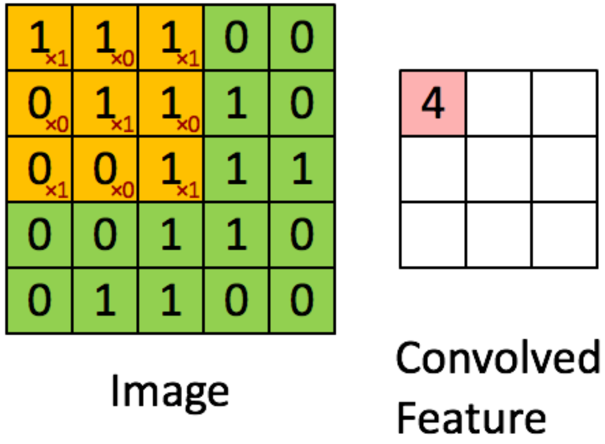
- Mainly applied in processing image input.
- Cascade of convolution, ReLU, pooling (subsampling) and fully-connected layer.
- Cascaded convolutional layers extract the high-level features of input image. Subsequent fully-connected layers serve as classifier.

Convolutional Neural Network

Convolution

$$(f * g)(t) \triangleq \int_{-\infty}^{\infty} f(\tau) g(t - \tau) \, d\tau.$$

2D-convolution on image



Convolutional kernel (filter): the spatial size of convolutional kernel is usually much smaller than input image. (7x7, 5x5, 3x3 kernel)

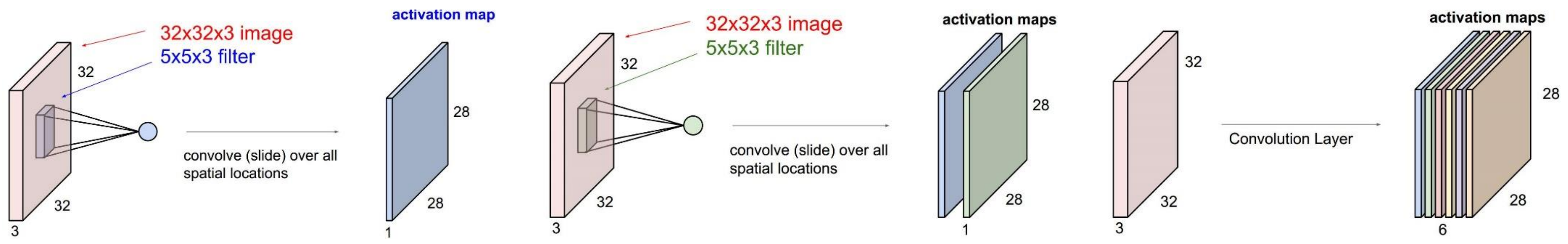
$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

I – input, *K – kernel (size k x k)*

S – output, *m, n in [-k, k]*

(i, j) the spatial axes of output

Convolutional Neural Network

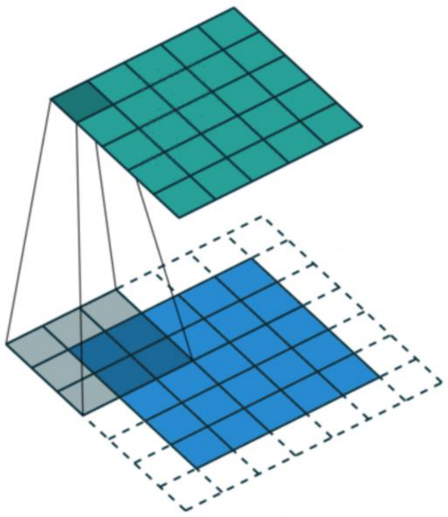


Multiple filters will form multiple output map. Feature map is obtained by concatenating these output maps and activating them with activation function.

In practice, to avoid feature map spatial size decreasing, we usually padding the input with zero.

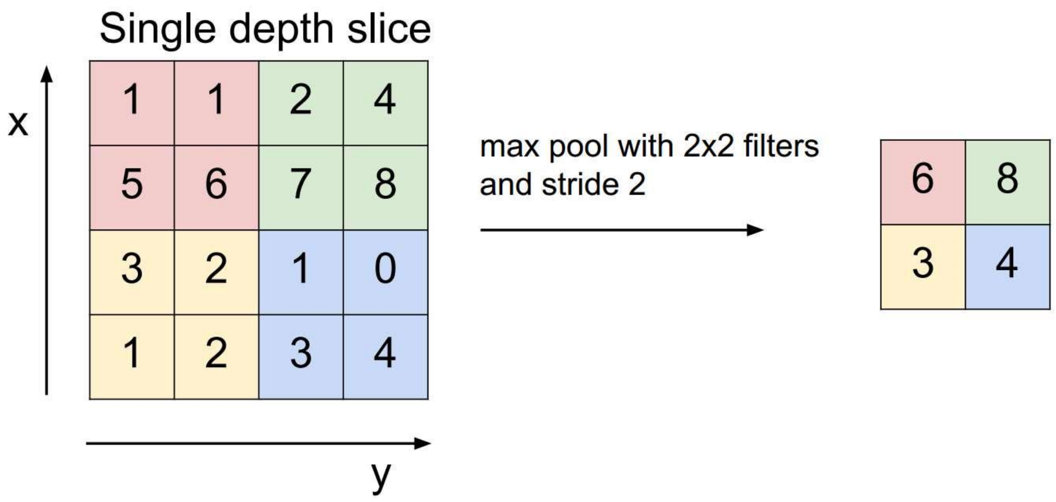
Difference from fully-connected network:

sparse connection and parameter sharing.



Convolutional Neural Network

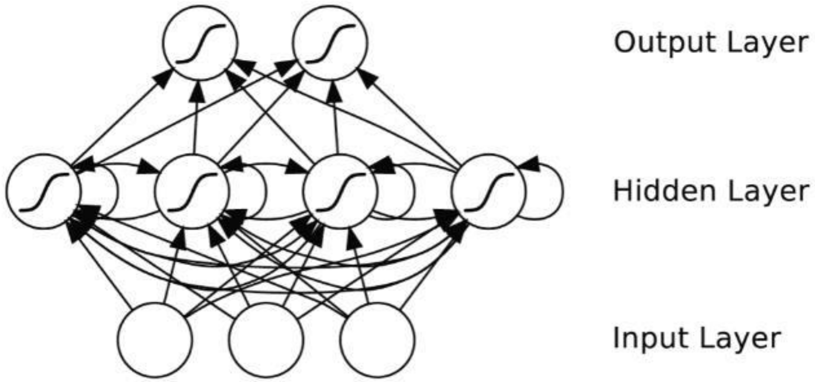
Max-pooling

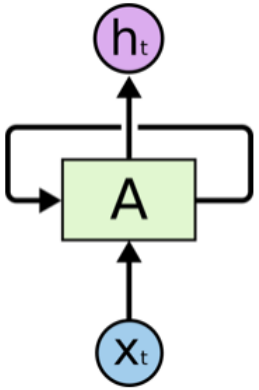


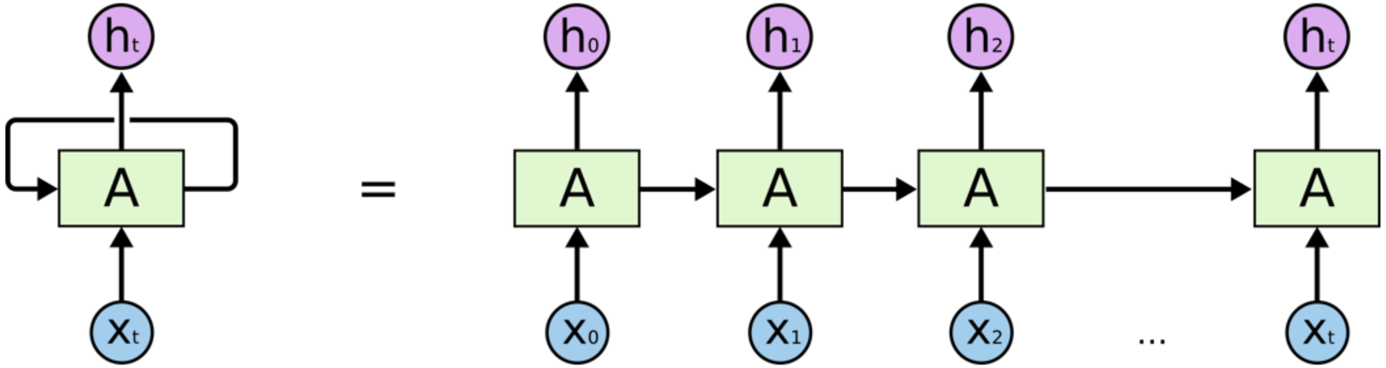
Make the feature map smaller and more manageable. Acquire some invariance properties.

Other pooling: average pooling. Replace the max with average.

Recurrent Neural Network



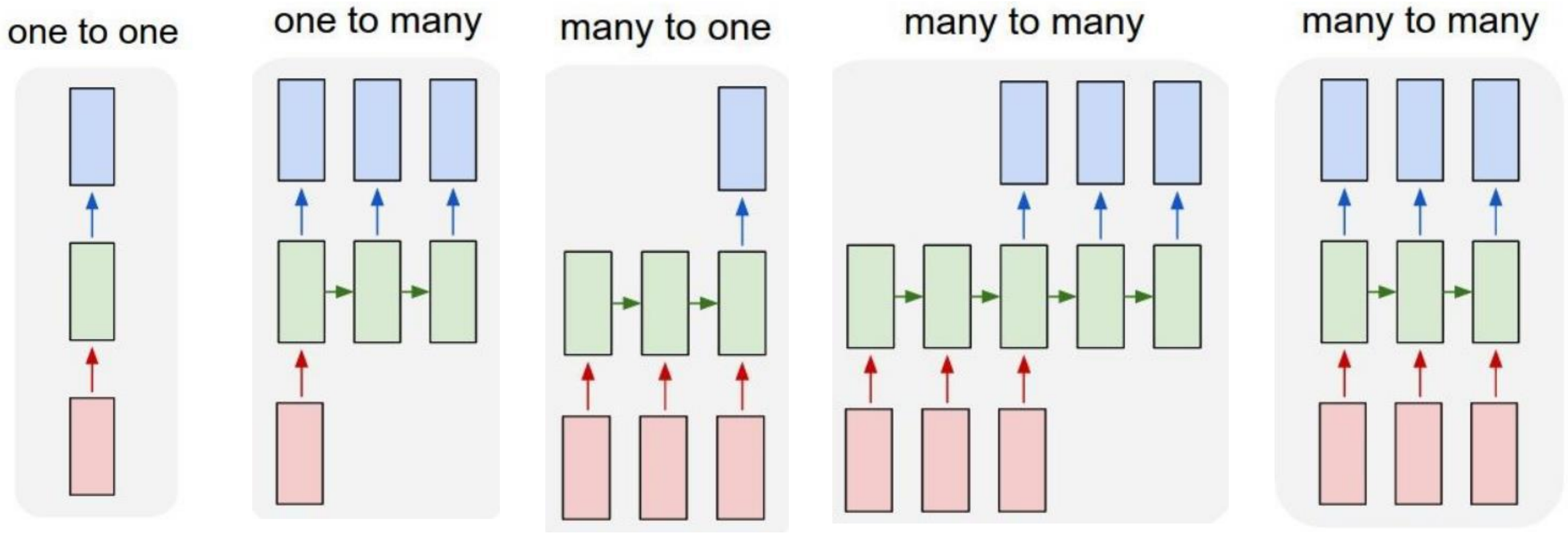




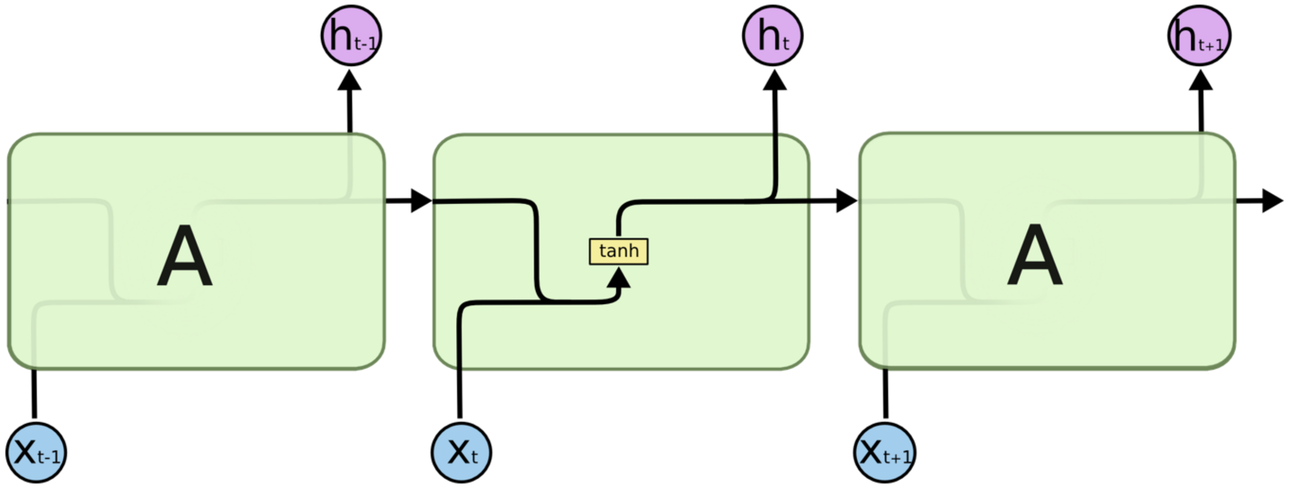
https://blog.csdn.net/qq_29462849/article/details/80633975

Recurrent Neural Network

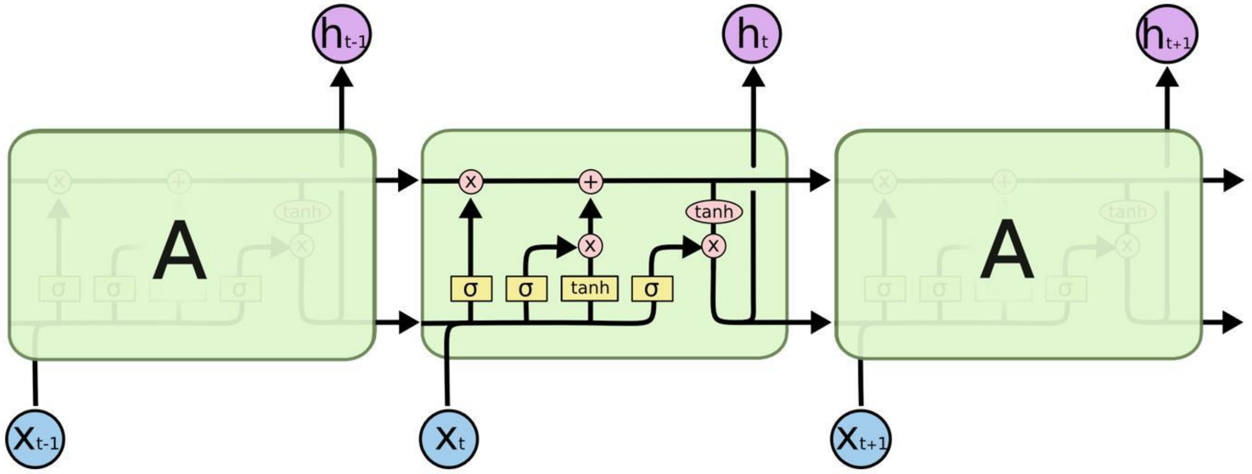
Different forms of recurrent neural networks



Recurrent Neural Network



The repeating module in a standard RNN contains a single layer.

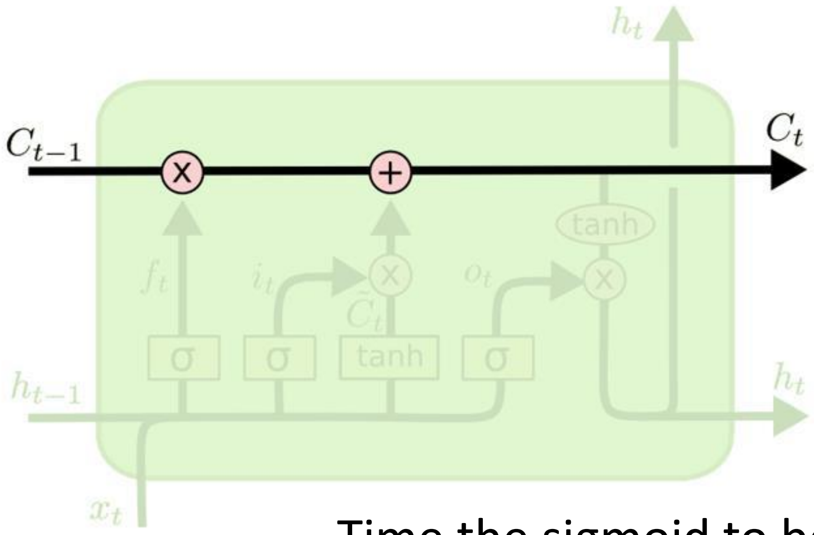


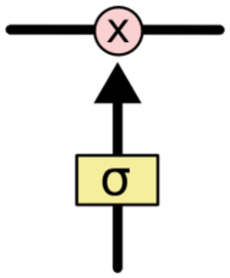
The repeating module in an LSTM contains four interacting layers.

Cell state
(change slowly)

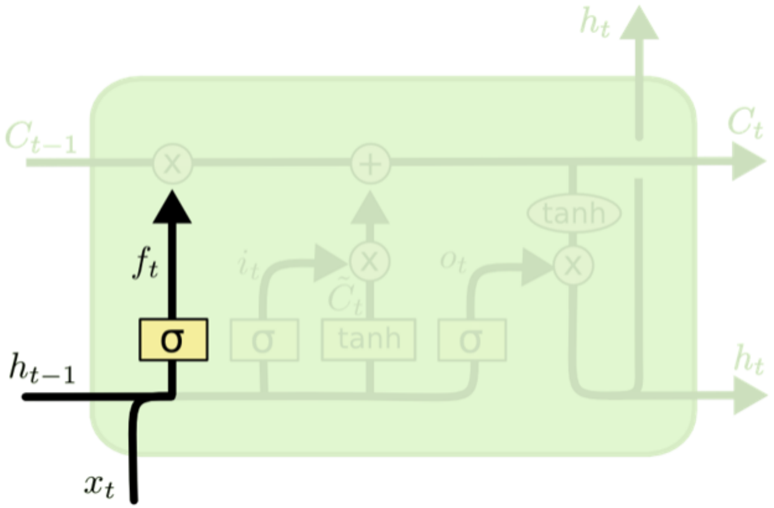
Hidden state
(change quickly)

Recurrent Neural Network



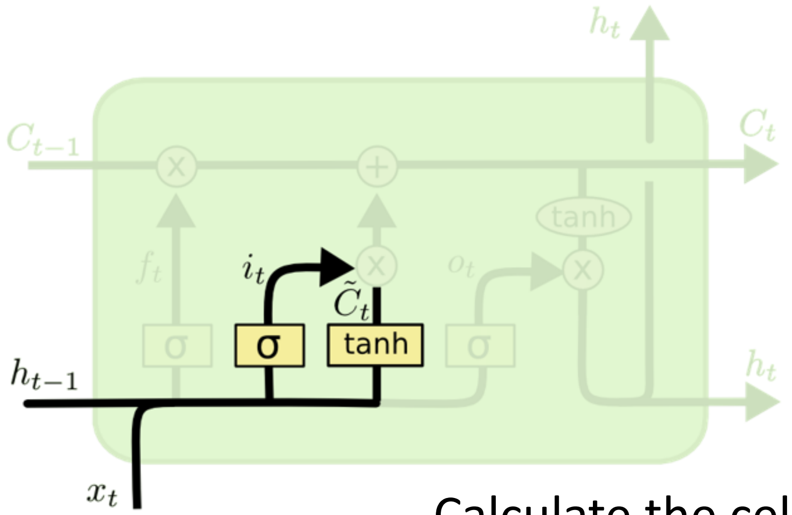


Time the sigmoid to be able to forget the previous cell state



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

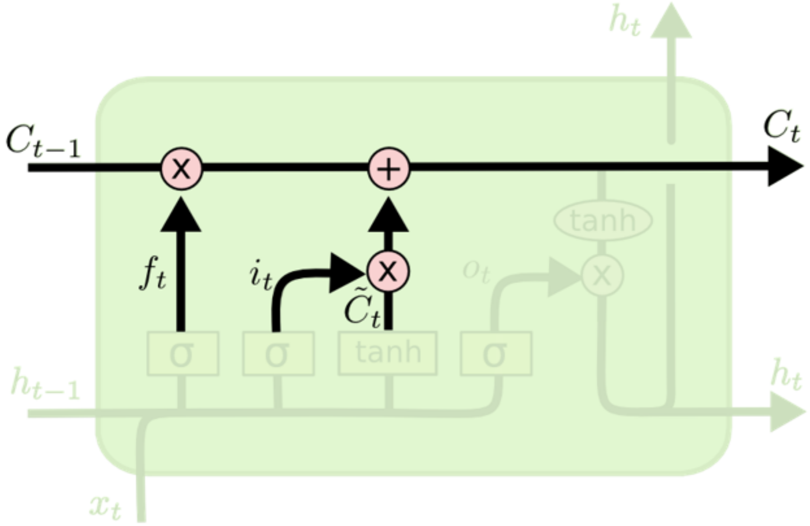
Recurrent Neural Network



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

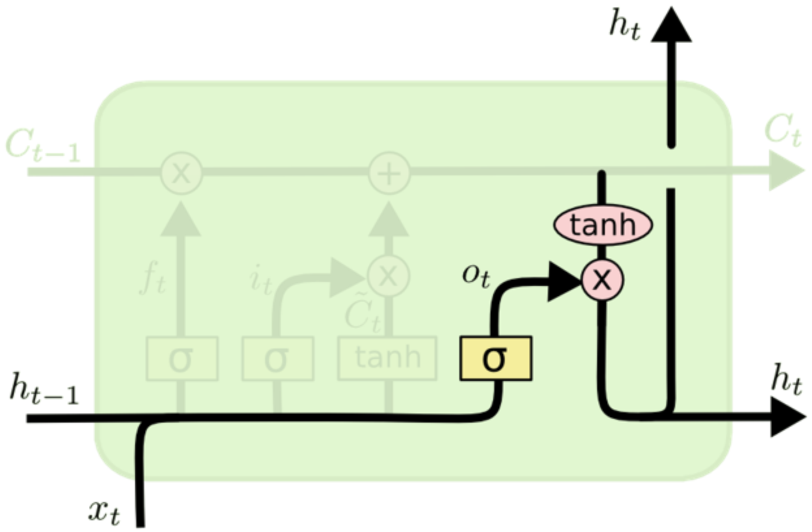
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Calculate the cell state of this step from the input of this step, the hidden state of the last step, and the cell state of last step.



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

Recurrent Neural Network



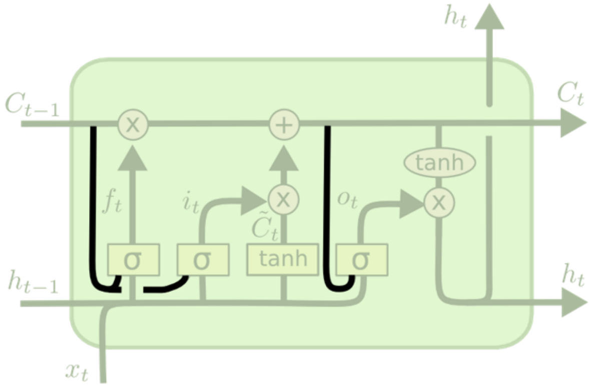
$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$

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

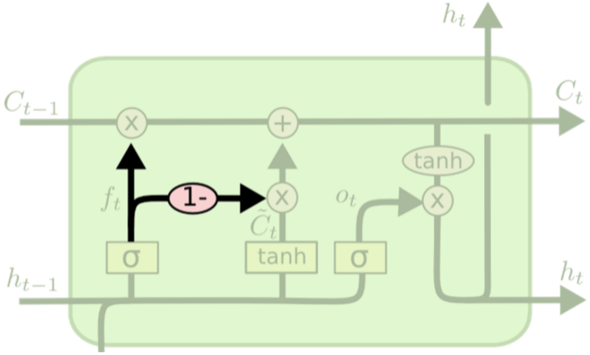
Output the hidden state of this step from the hidden state of the last step, the input, and the cell state of this step.

Recurrent Neural Network

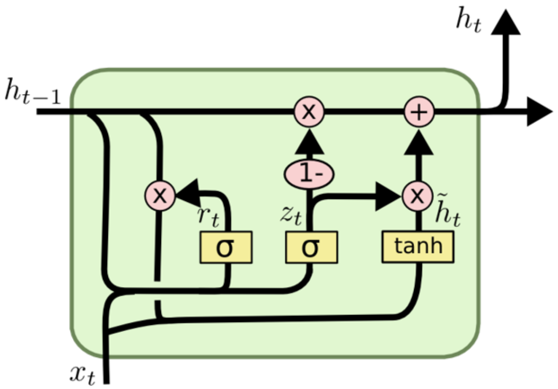
Some other form of LSTM



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$



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



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Gers & Schmidhuber (2000)
Cho, et al. (2014).

Load forecasting using LSTM

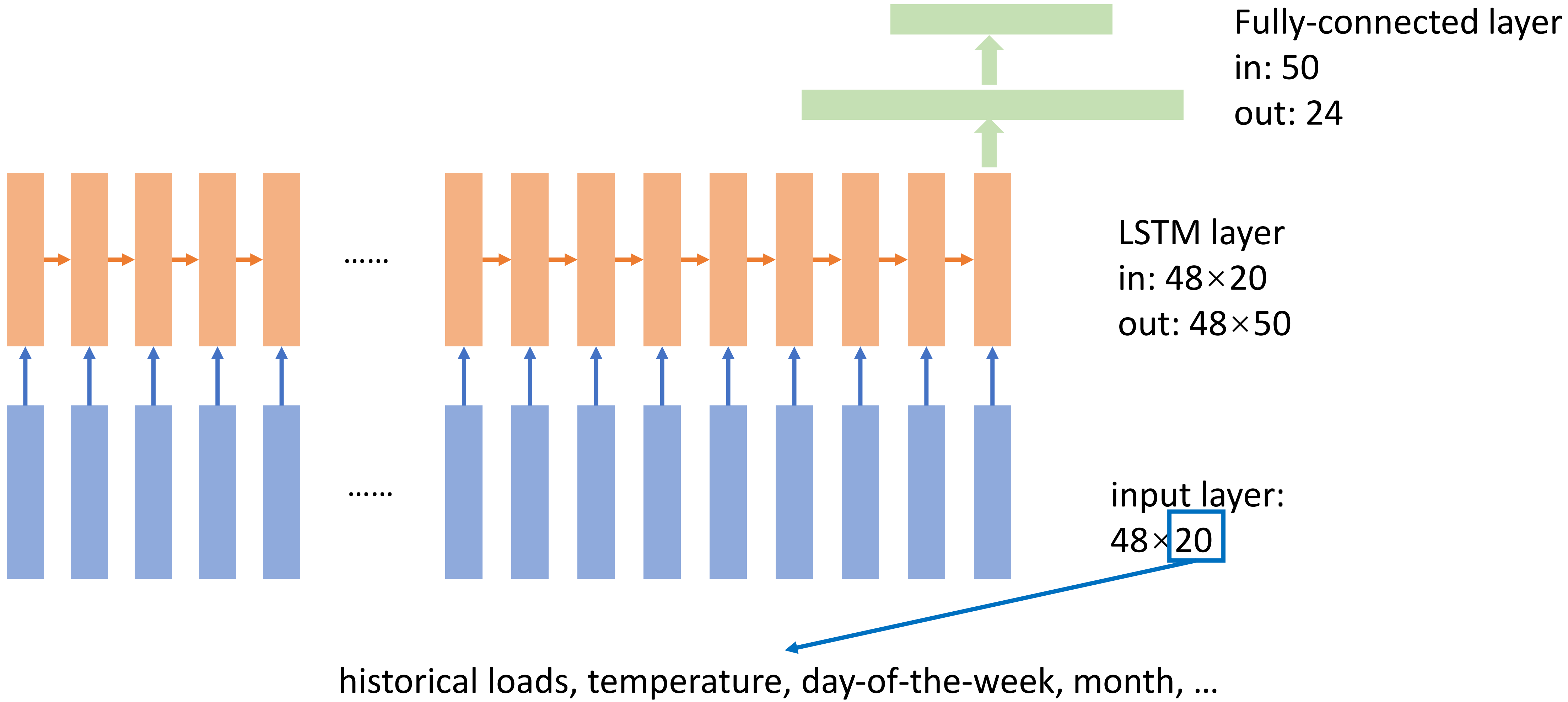
The task is to provide one-month ahead hourly forecasts given the historical hourly load data and temperature data.

The task was a load forecasting competition GEFCom2014, which can be found https://www.sciencedirect.com/science/article/pii/S0169207016000133?via%3Dihub

We implement the code from Jingrui Xie https://github.com/Timbasa/GEFCom2014_Load_Forecasting_SVRG_BB

The code was implemented by pytorch, which is a popular deep learning tool based on python. https://pytorch.org/

Load forecasting using LSTM



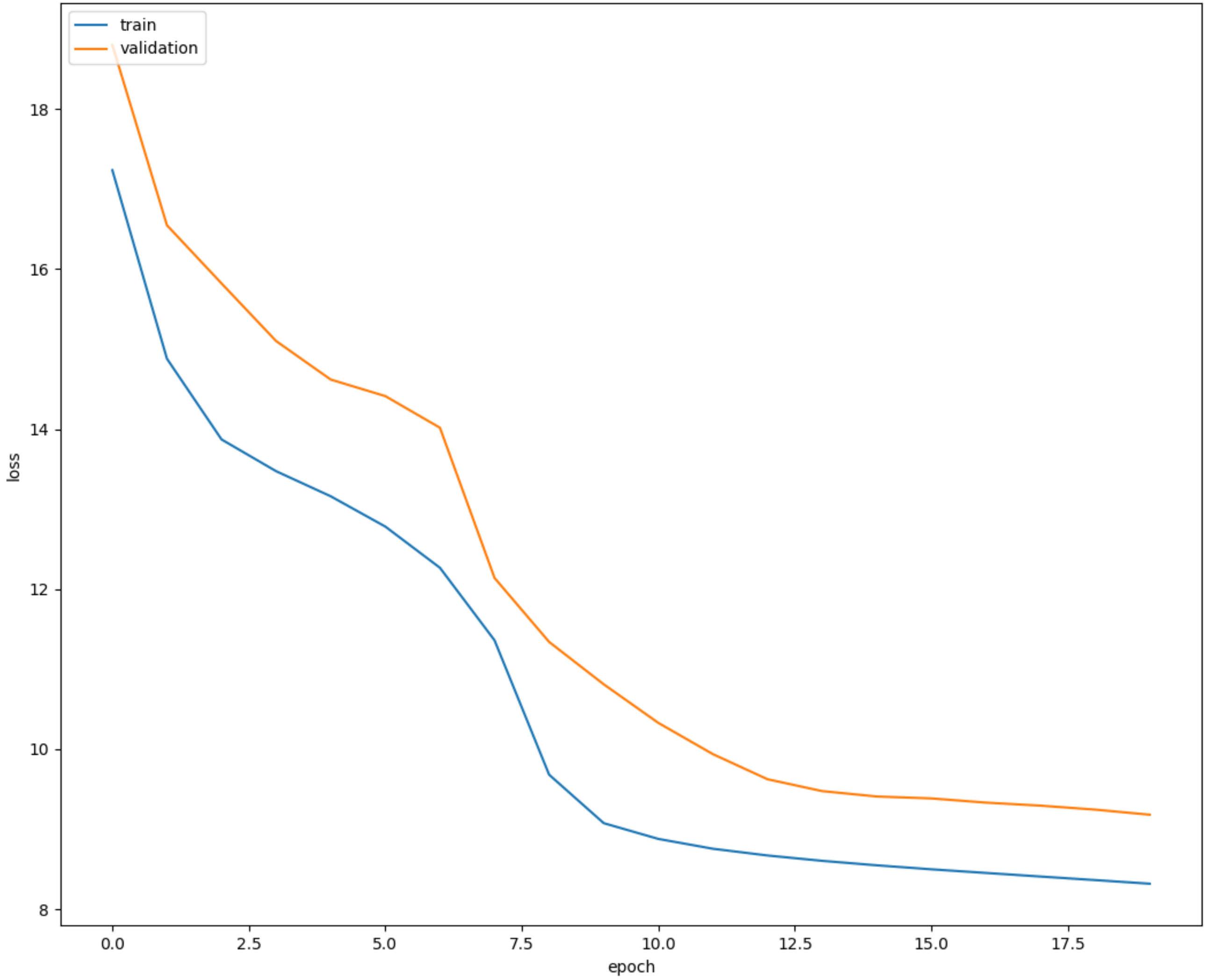
Load forecasting using LSTM

```
def __init__(self, input_size, hidden_size, number_layer,
 output_size, output_layer):
    super(LSTM, self).__init__()
    self.output_size = output_size
    self.output_layer = output_layer
    self.lstm = nn.LSTM(input_size=input_size,
                        hidden_size=hidden_size,
                        num_layers=number_layer,
                        batch_first=True,
                        dropout=0.2)
    self.out = nn.ModuleList([nn.Linear(hidden_size, outp
ut_size) for _ in range(output_layer)])
```

Load forecasting using LSTM

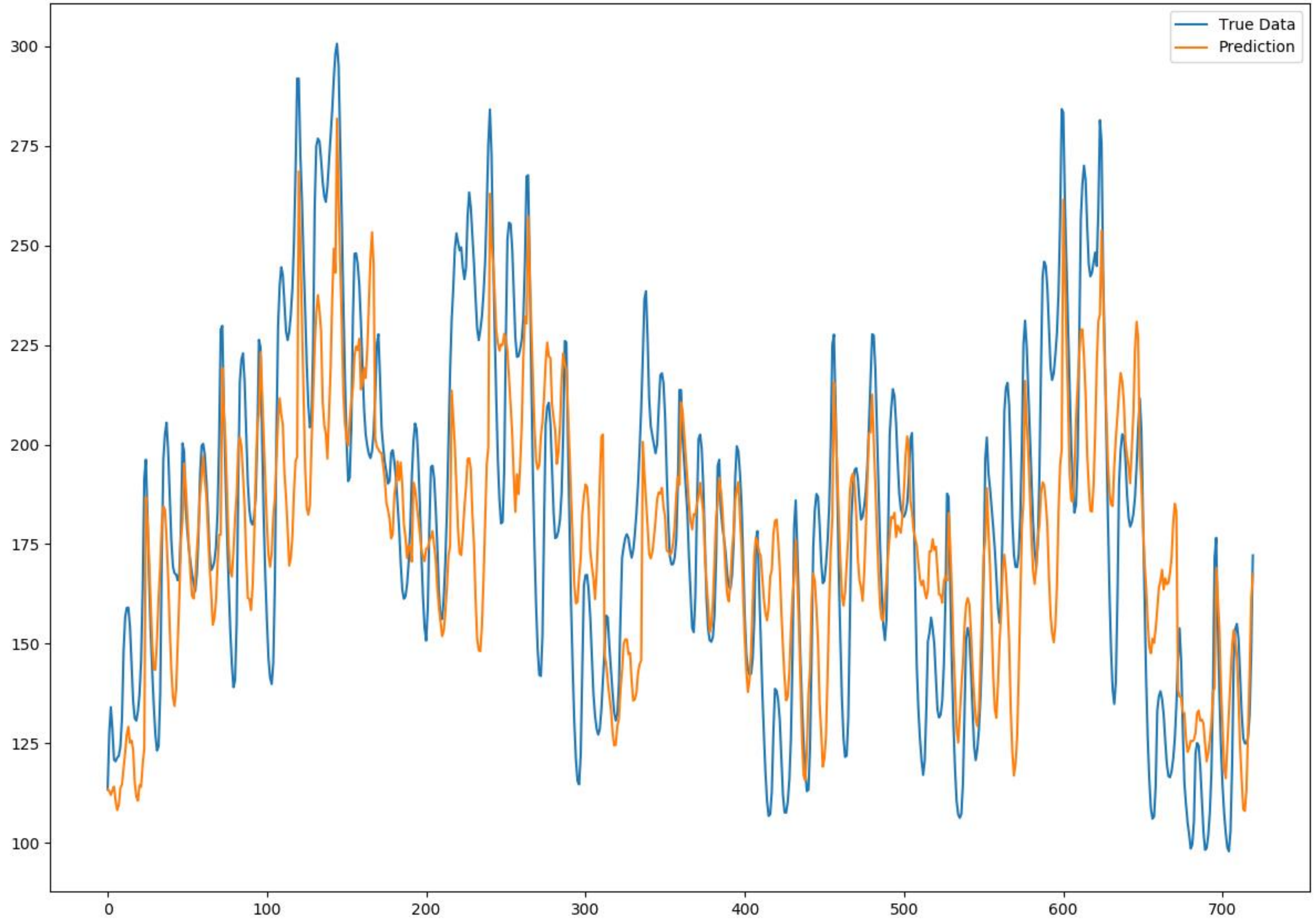
```
def forward(self, x):
    out, _ = self.lstm(x, None)
    out = torch.cat([layer(out[:, -
1, :]) for layer in self.out], dim=1)
    out = out.view(out.size(0), self.output_size, self.ou
tput_layer)
    return out
```

Load forecasting using LSTM



training process

Load forecasting using LSTM



forecasting results

forecast loss:8.433679580688477

Homework

Compulsory: Current codes split the dataset into the train dataset and the validation one. Please modify codes to evaluate the accuracy on the test dataset.
Choose one:
Modify the above load forecasting program and improve the model accuracy, possible modifications includes:

1. Change the structure of the networks
2. Change the hyper-parameter of the algorithm
3. Change the inputs
4. Change the training methods
5. And so on.

Pytorch is required to run the program, see the following link for pytorch installation:
https://pytorch.org/get-started/locally/

Q&A