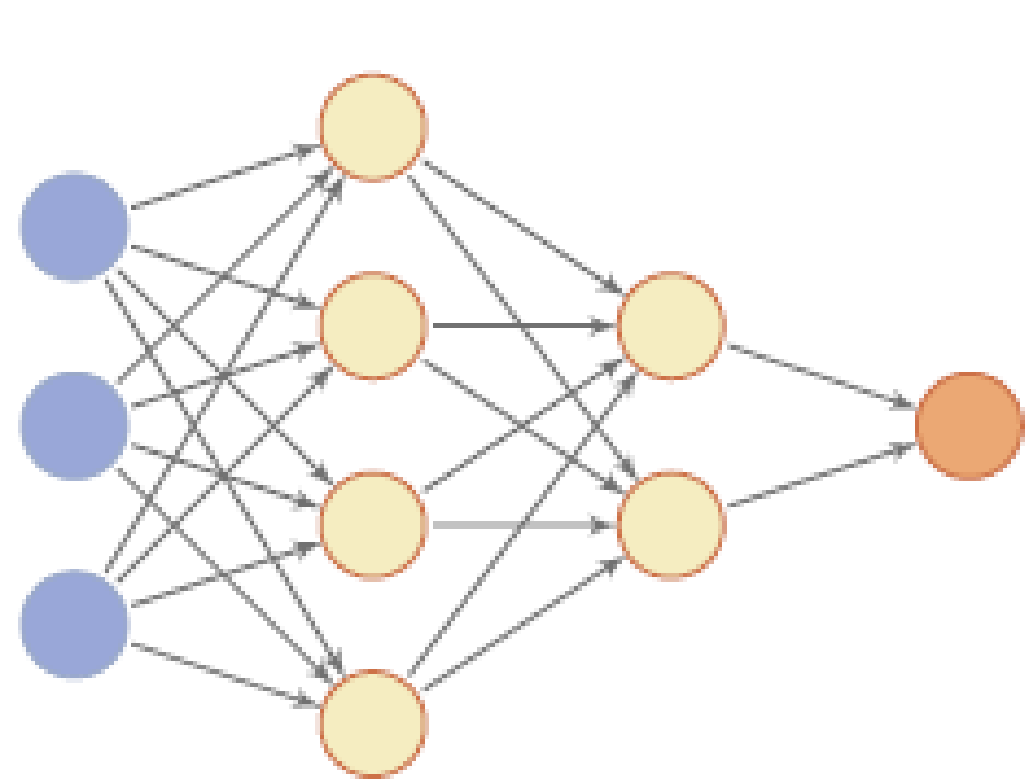


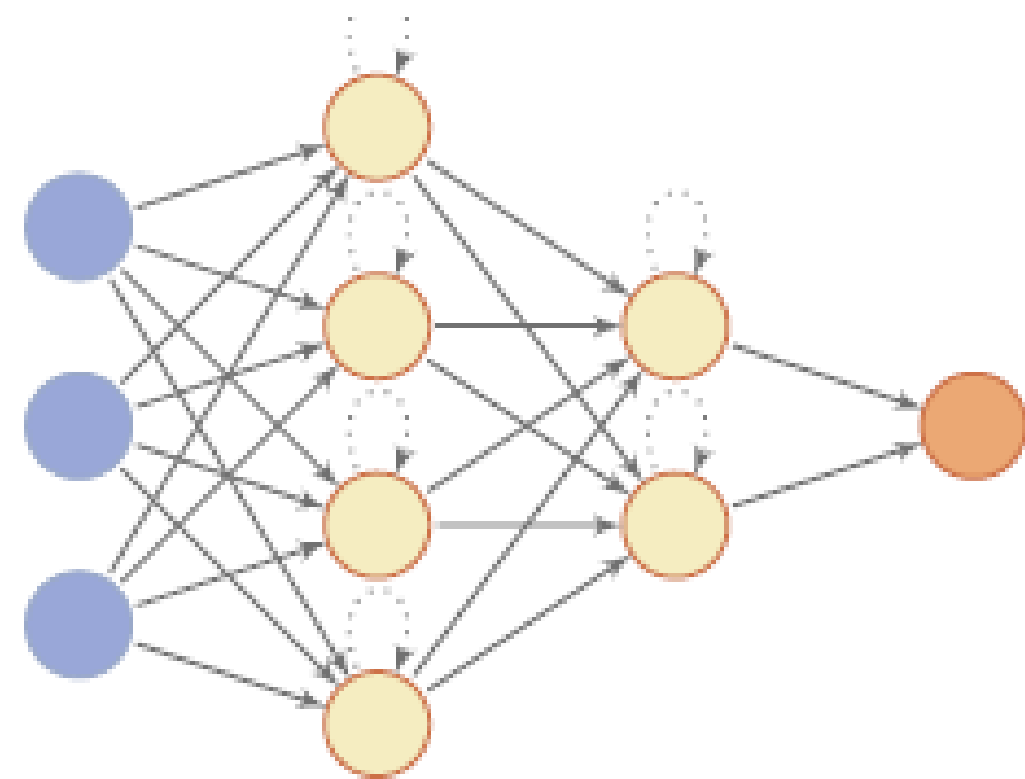
Dense & CNN & RNN

# 人工神经网络

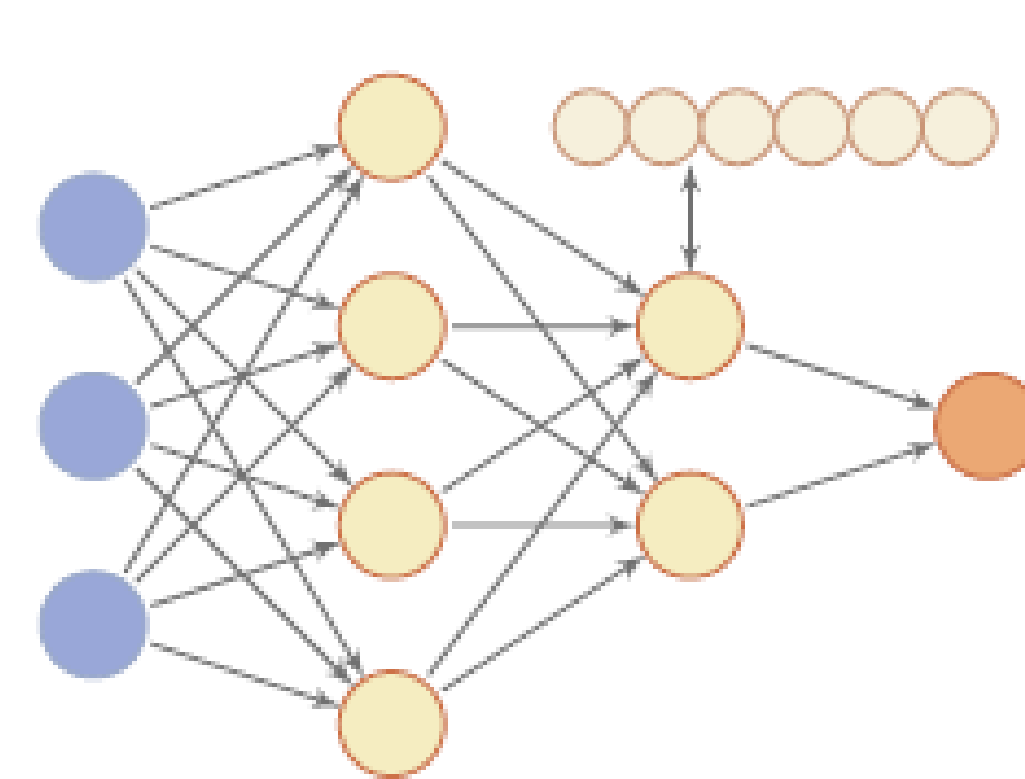
- 连接主义观点：
  - 将大量的神经元之间，按照拓扑连接结构，构成人工神经网络。
- 网络的拓扑结构
  - 不同神经元之间的连接关系。
  - 前馈网络（feedforward）、反馈网络（feedback）和记忆网络（memory network）



(a) 前馈网络

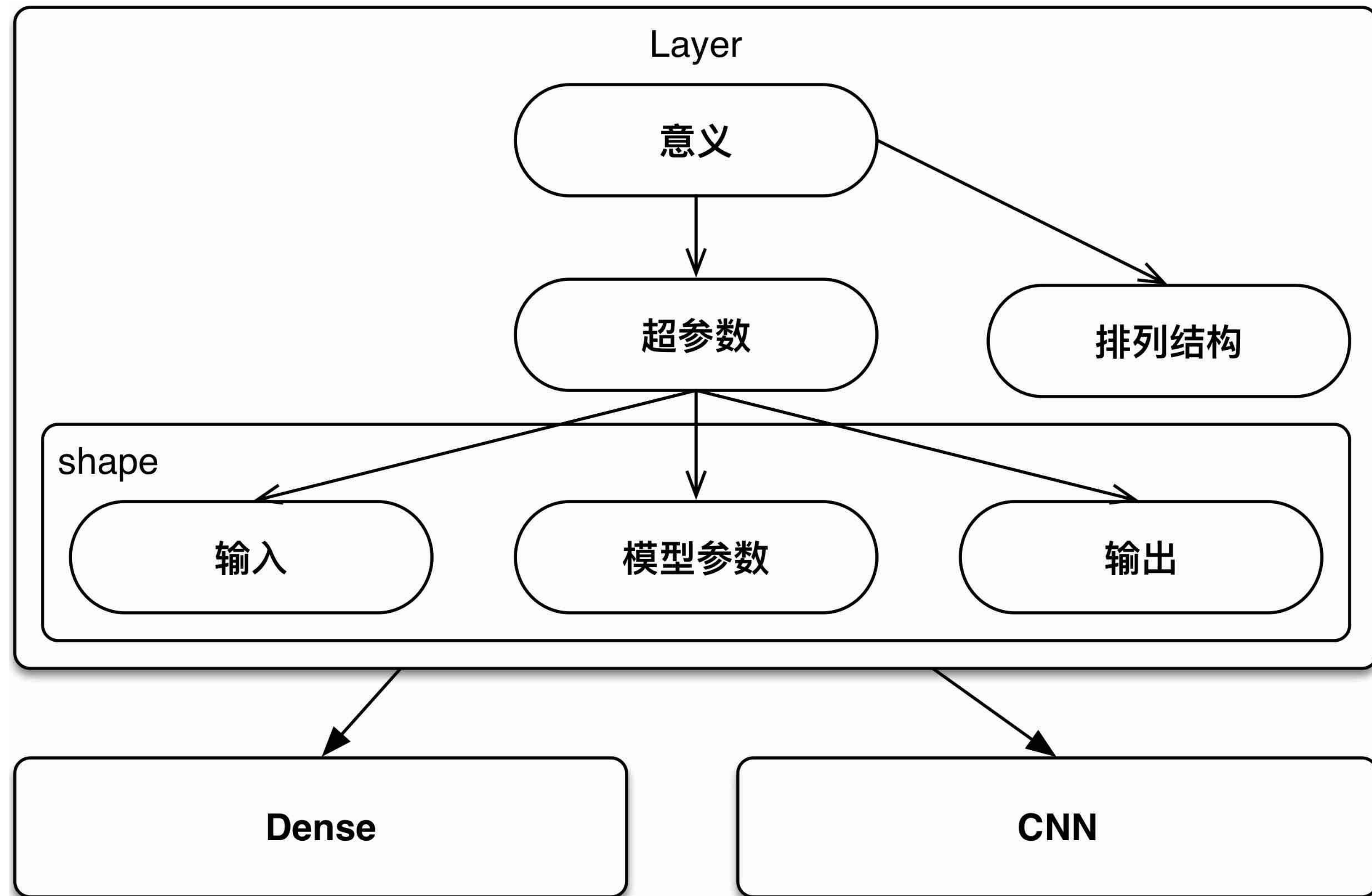


(b) 反馈网络

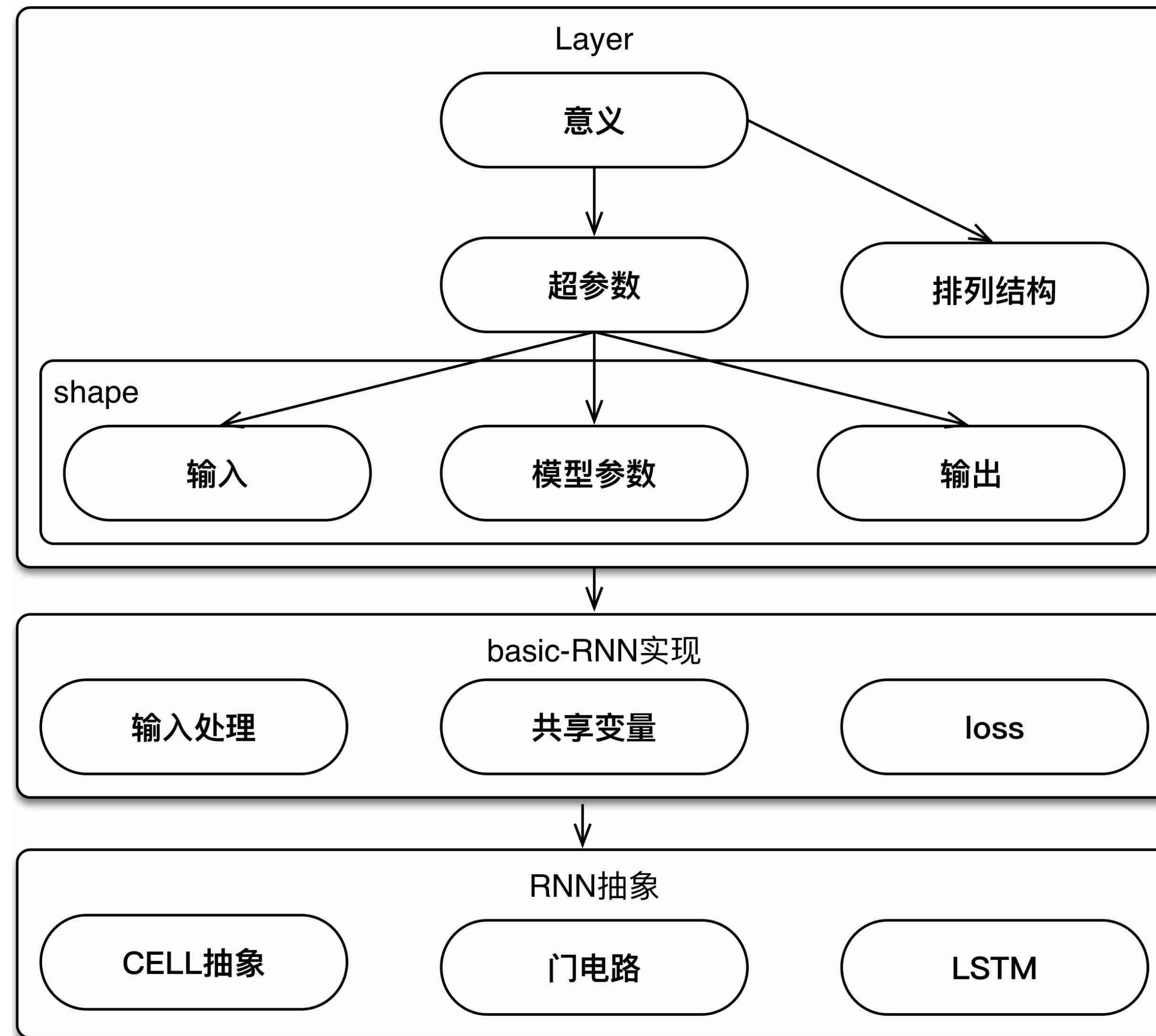


(c) 记忆网络

# TensorFlow学习路线： Dense与CNN

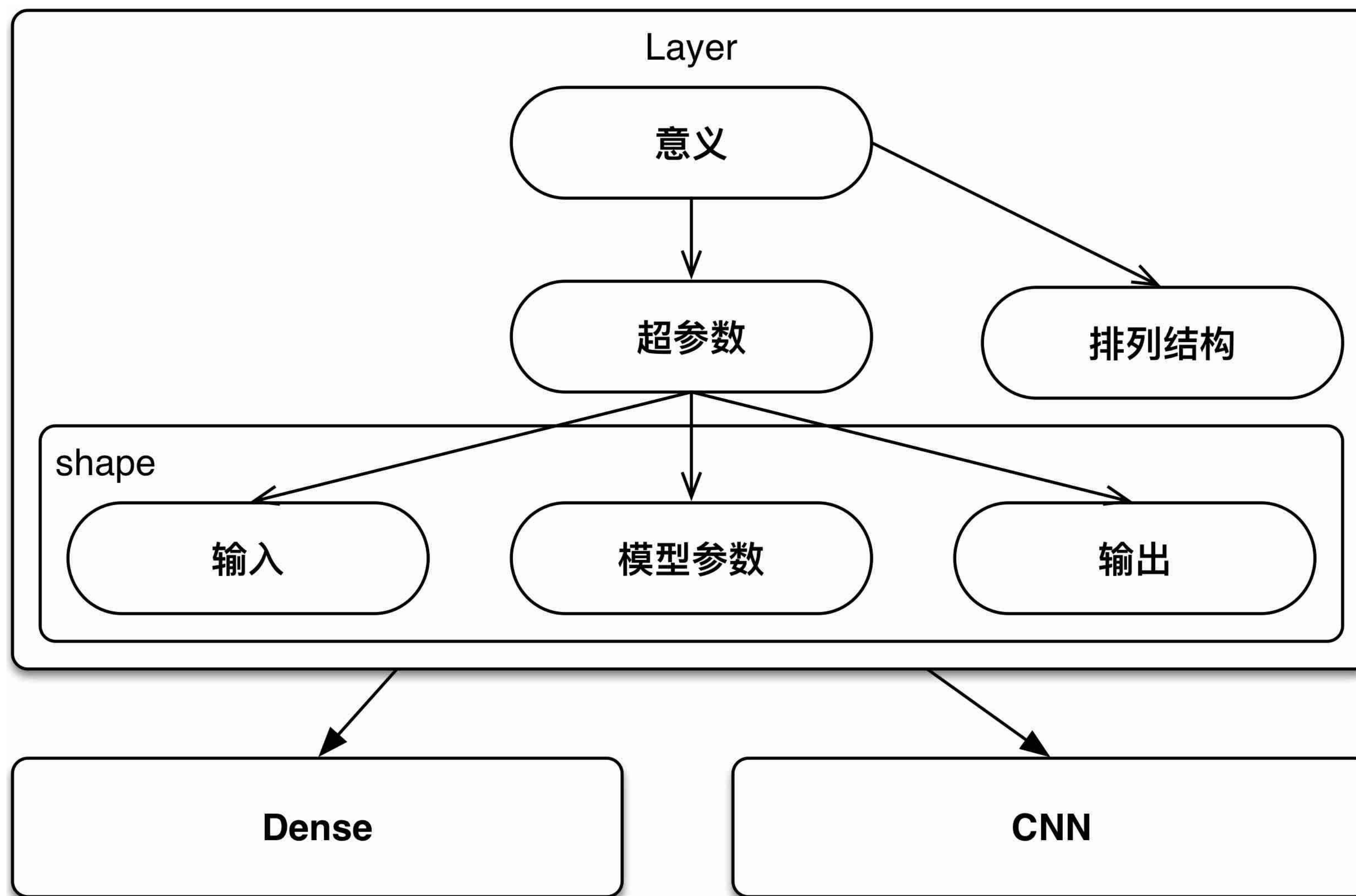


# TensorFlow学习路线： RNN与LSTM



Dense

# 学习路线



# Tensor形象化表示

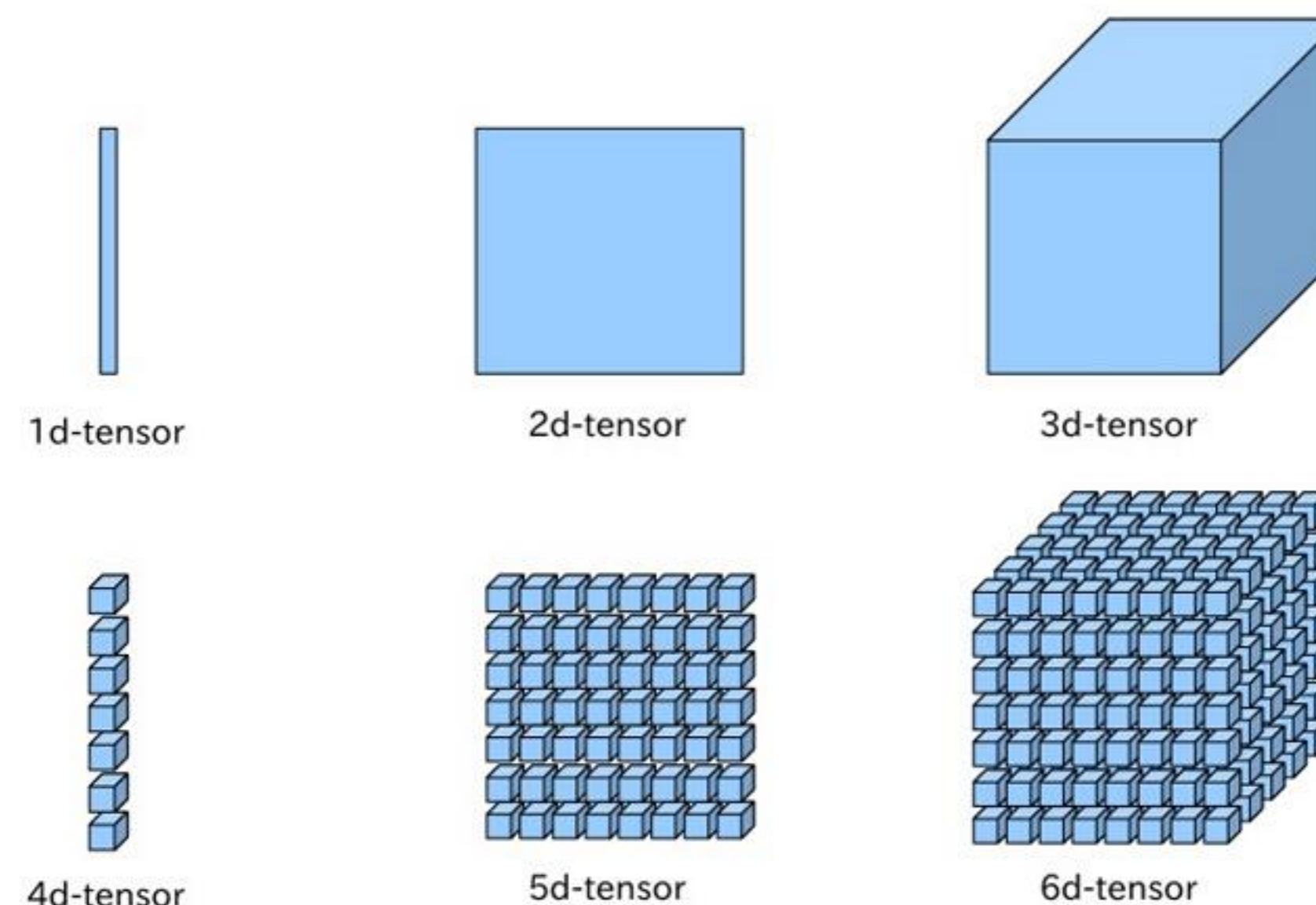
- 对于一个4\*5\*6的Tensor

- rank : 3d

- length: 4, 5, 6

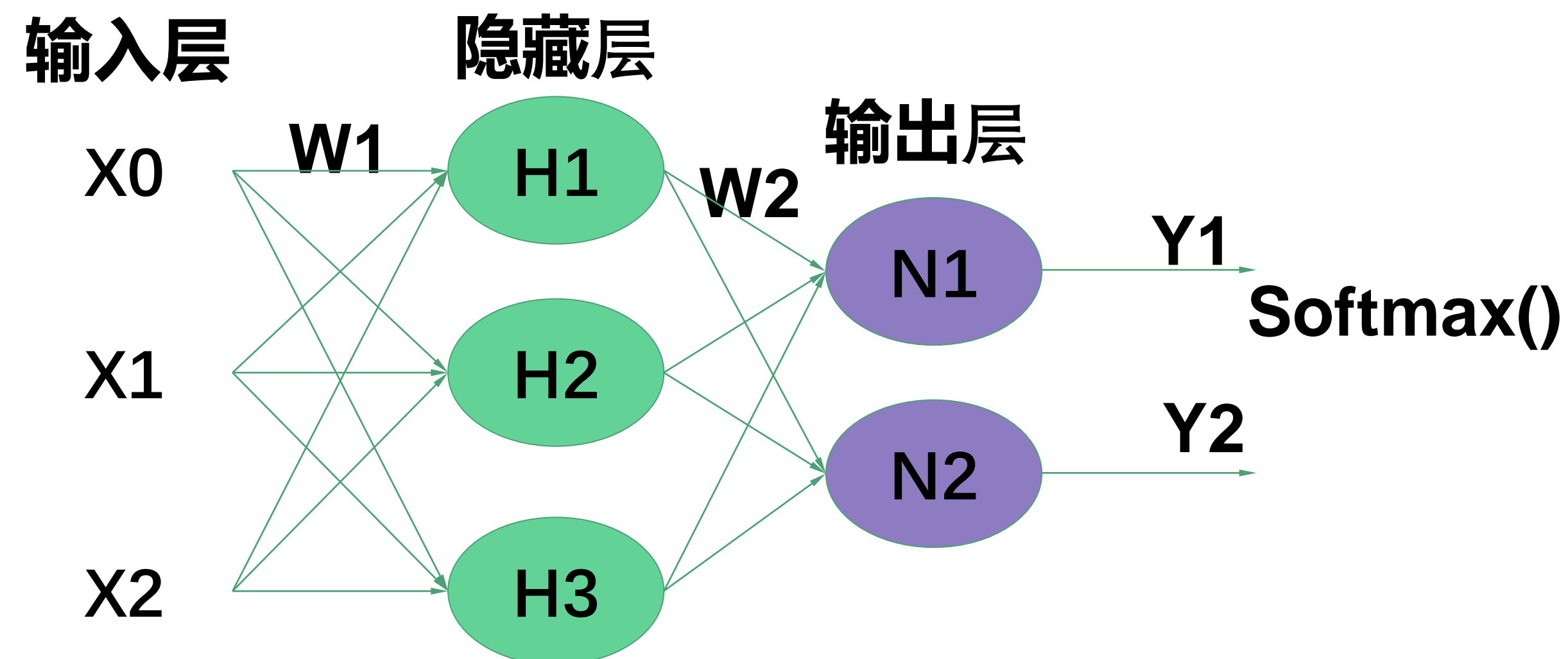
- shape: [4, 5, 6]

- volume:  $4*5*6=120$



# 多层前馈网络 (Multilayer feedforward networks)

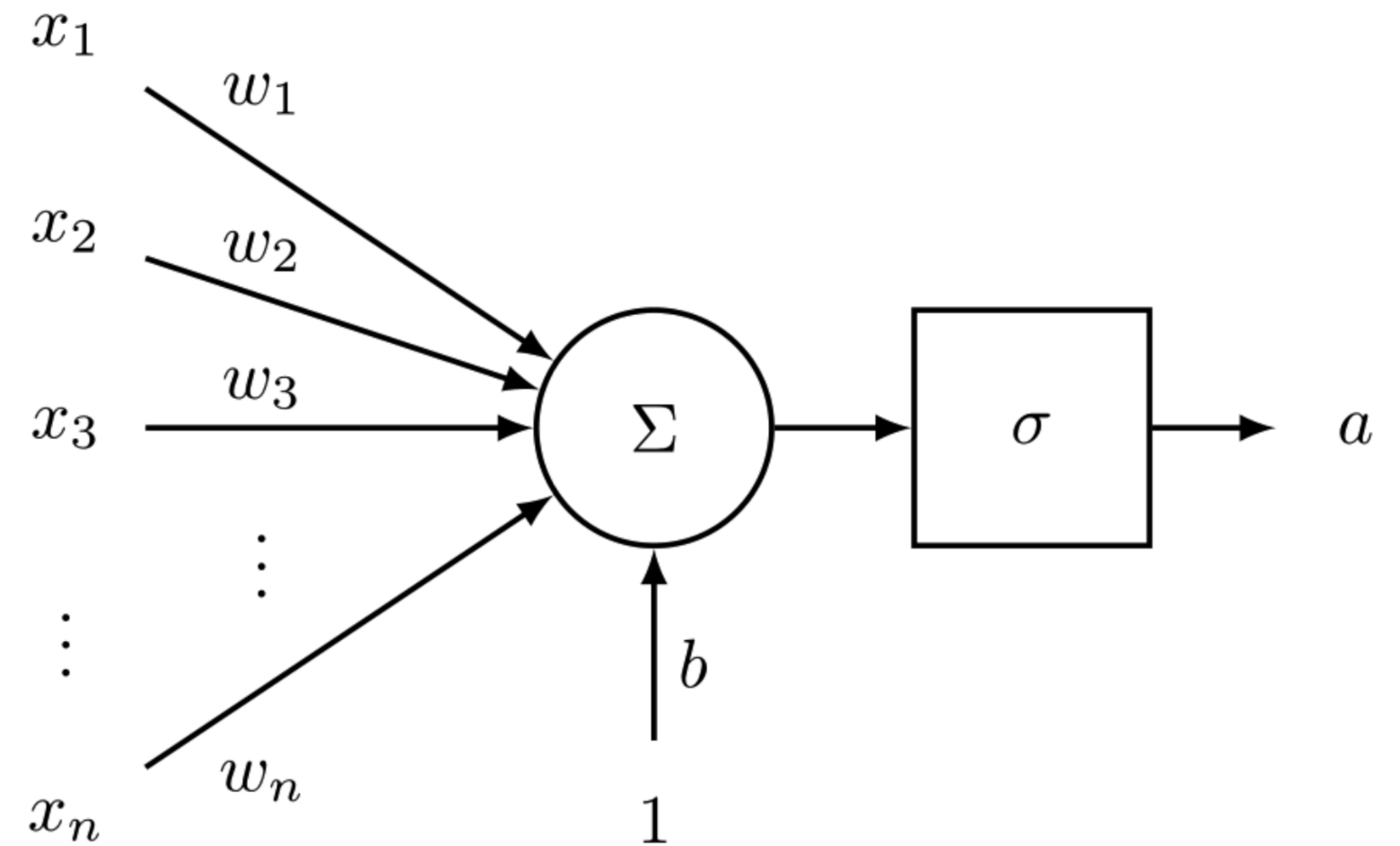
- 别名：前馈网络（FNN），多层全连接网络（FCN）、多层感知机（MLP）、多个密集层网络（Dense）
- 示例网络：有3个输入，2个输出；输入层、隐藏层，输出层；
- 隐藏层、输入层，共有5个神经元。
- 前馈网络结构可以表示为Tensor的**计算图**（Computational Graph）





# Tensor表示: Neuron $\rightarrow$ Layer

- 单个神经元:
  - 输入是1d, 参数是1d, 输出是0d
- 一层神经元构成一个Layer
  - 显然输出的shape和Layer的shape一致.
- batch\_size
  - 会影响输出的shape
  - 并不会影响参数的shape



# Dense

- 排列结构: Layer的结构是1d
- 超参数: 神经元的个数U
- shape:
  - $\text{input} = L$
  - $\text{weights} = L * U$
  - $\text{output} = U$
- 意义: 多个Dense层构成MLP (Multi-layer Perceptron, 多层感知机), 用于解分类问题.

```
tf.layers.dense(  
    inputs,  
    units,  
    activation=None,  
    use_bias=True,  
    kernel_initializer=None,  
    bias_initializer=tf.zeros_initializer(),  
    kernel_regularizer=None,  
    bias_regularizer=None,  
    activity_regularizer=None,  
    kernel_constraint=None,  
    bias_constraint=None,  
    trainable=True,  
    name=None,  
    reuse=None  
)
```

# Softmax处理

- 输出层的Softmax 处理，计算出一个概率分布：

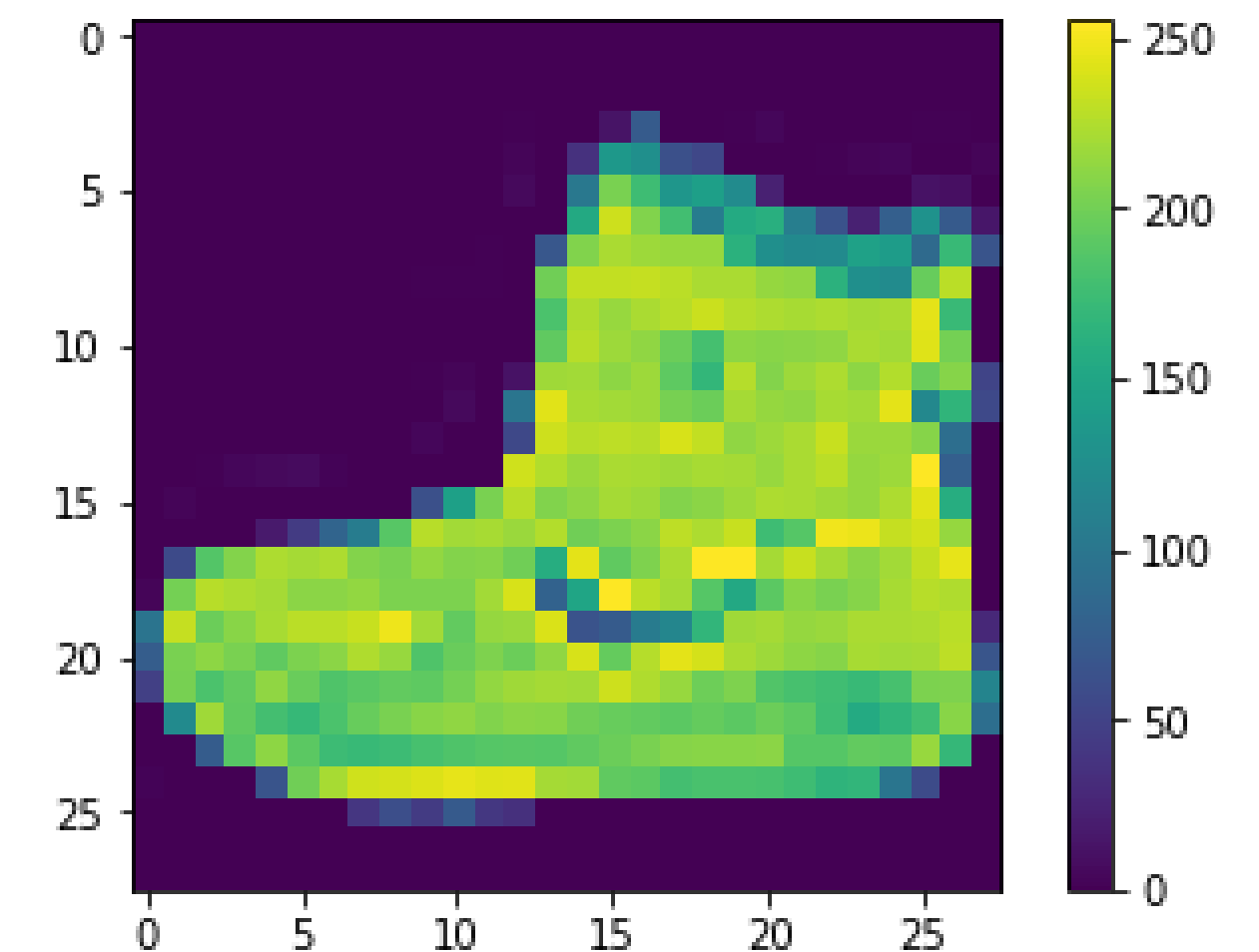
$$g(z_m) = \frac{e^{z_m}}{\sum_k e^{z_k}}$$

- 所有分量之和为 1, 所有输出的数值是正的。

`numpy_activation_function.ipynb`

# 图像处理

- 图像表示
  - 矩阵表示，每个像素是数字
- 手写字体MNIST数据集
  - 灰度图像，
  - 二值图像，黑白：
  - 0代表黑色， 1代表白色
- 时尚MNIST数据集（Fashion MNIST）
  - 彩色图像（RGB）：红(Red)，绿 (Green)， 蓝(Blue)
  - 对应的值域从0到255，对应8位2进制数字
  - 24位二进制数字

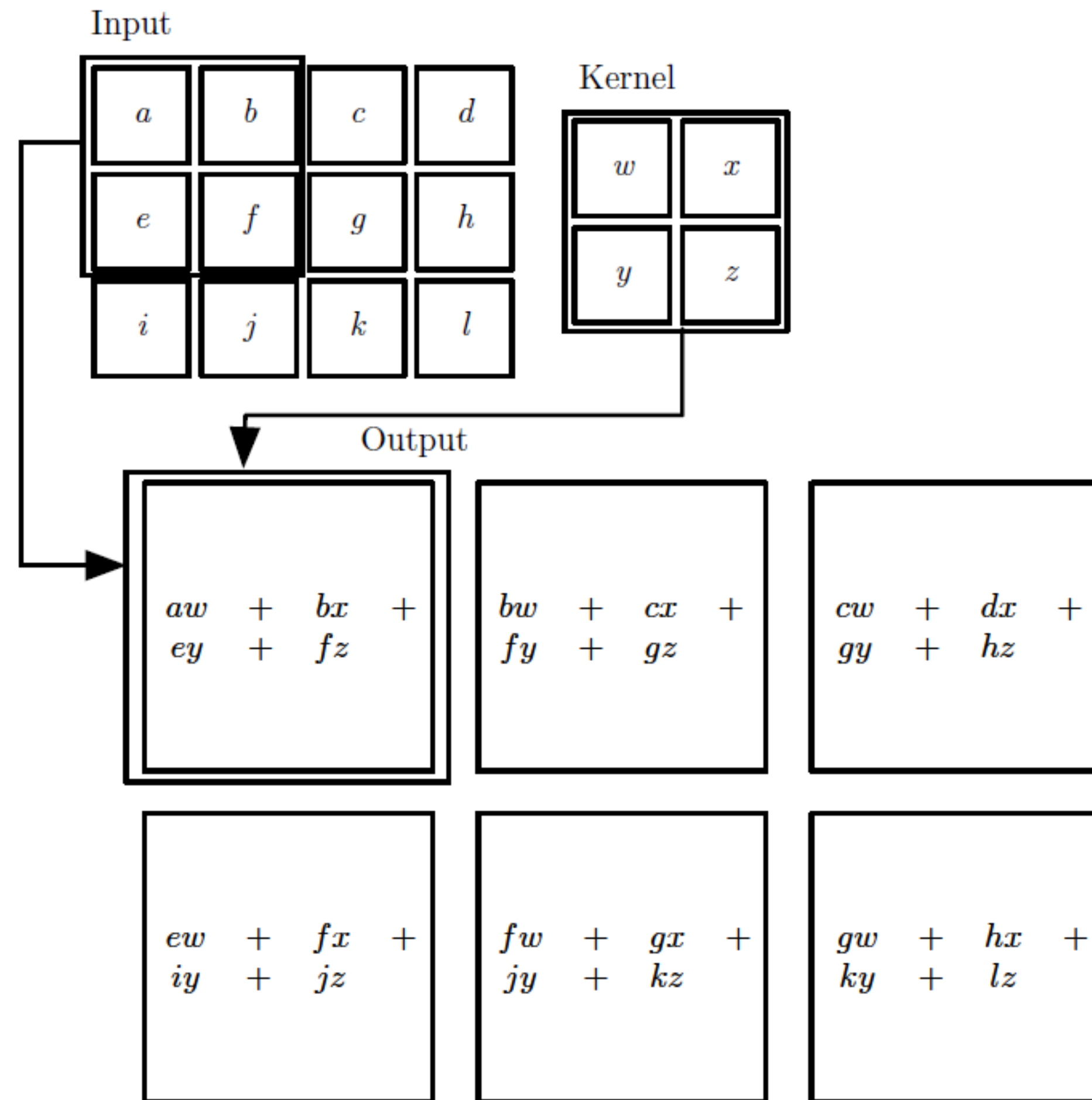


# MLP处理图像

- MLP 分类器（多层Dense）
- MLP来处理一张图像
  - 图像单个像素点的颜色RGB值表示
  - 一张200x200x3的图片
  - 单个神经元有 $200*200*3 = 120,000$ 参数！（参数量太大！怎么办？）

CNN

# 卷积运算 (Convolution)

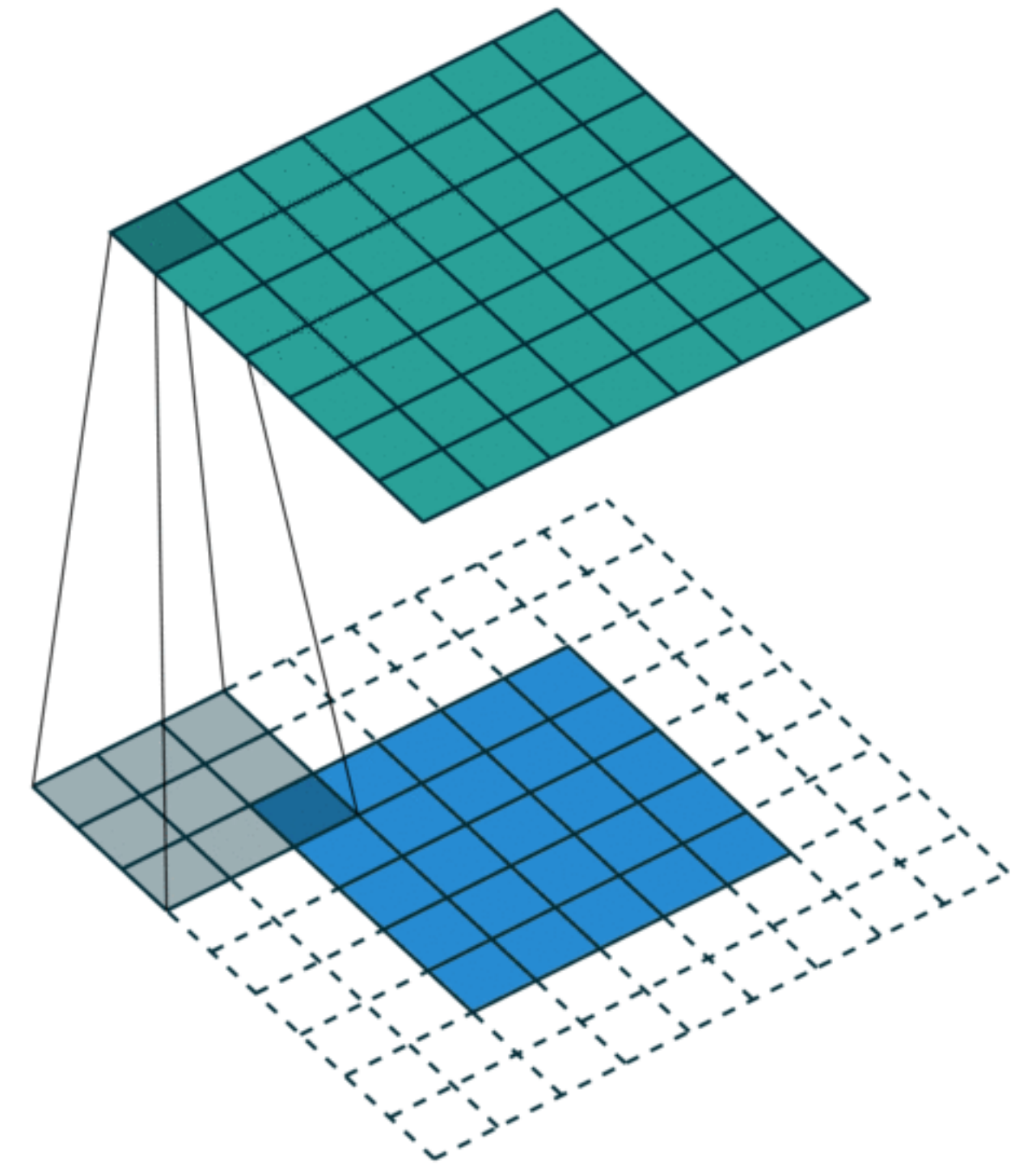


- 卷积是一种张量运算
- 输入是多维数组的数据 (Tensor)
- 卷积核 (Kernal) 是一个多维数组
- 卷积核是由学习算法得到的参数
- 卷积核数目一般选32、64等
- 这些多维数组都是张量 (Tensor)。



# 2d卷积核

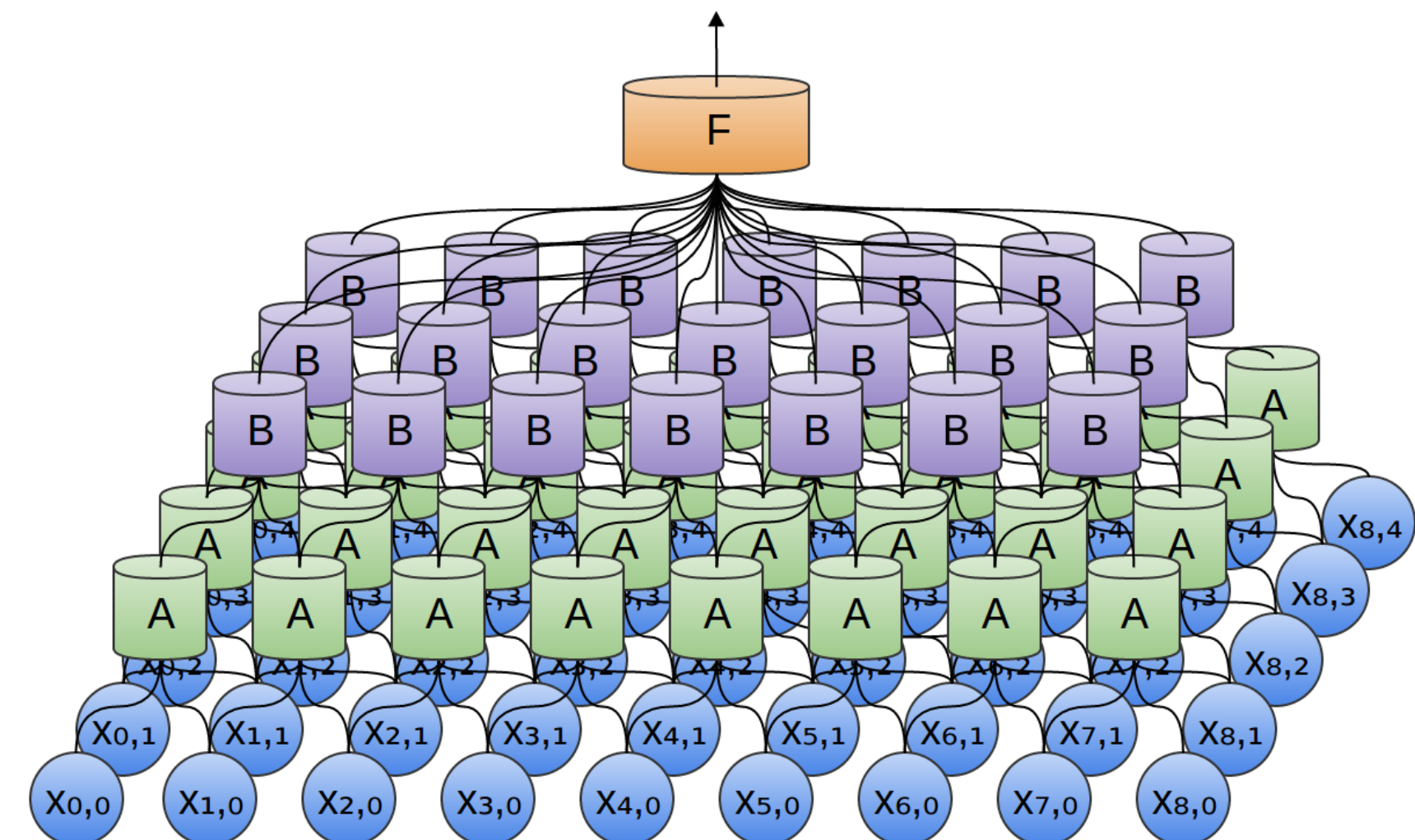
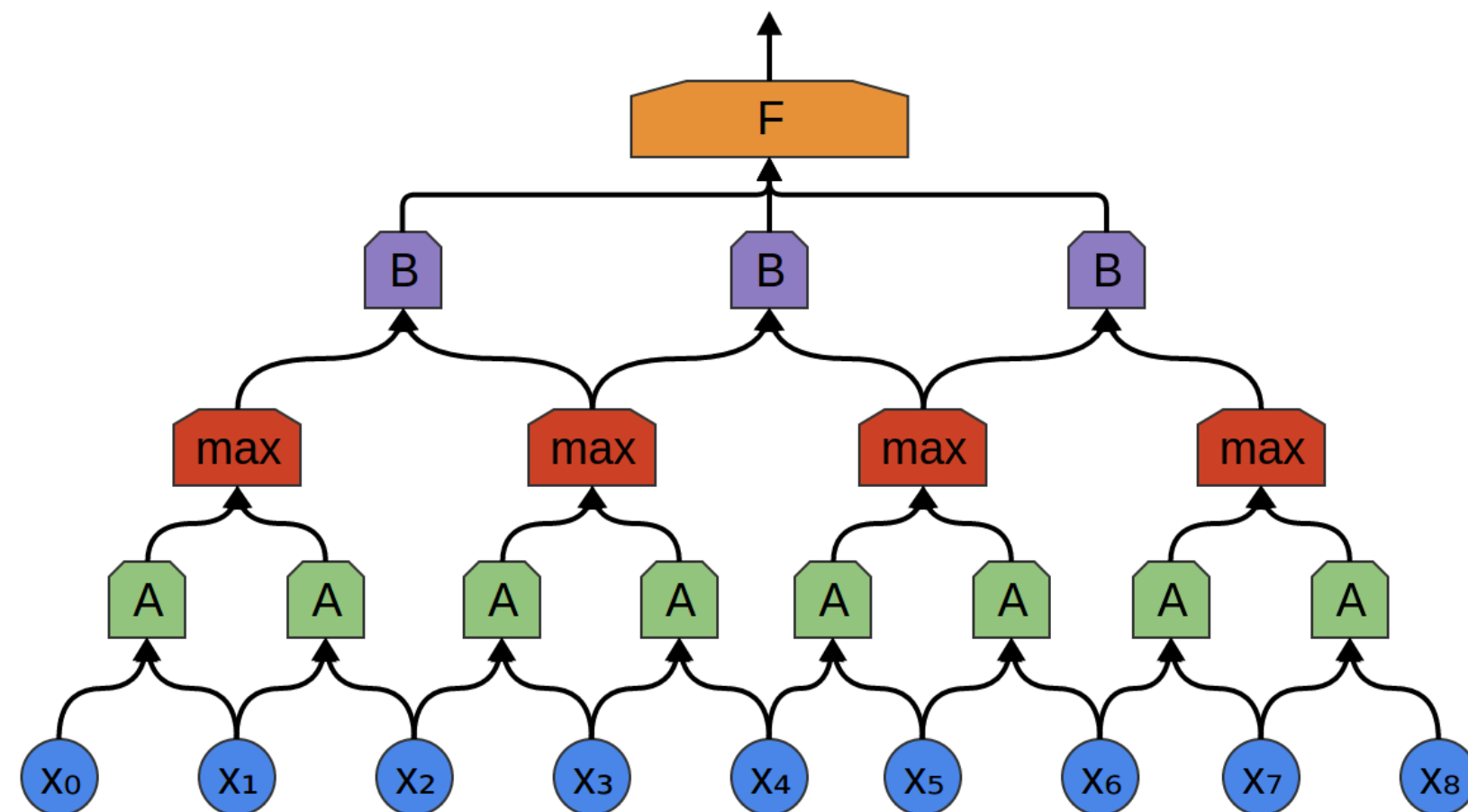
- 卷积核是一个多维数组Tensor，参数由学习算法得到的
- 定义输入的长度(W)，卷积核的大小(F)，核移动的步长stride(S)，zero padding(P)
- 输出的长度 $L = (W-F+2P)/S+1$
- 并行化：做一个和输出一样大小的Layer，Layer里面所有的神经元参数都一样！



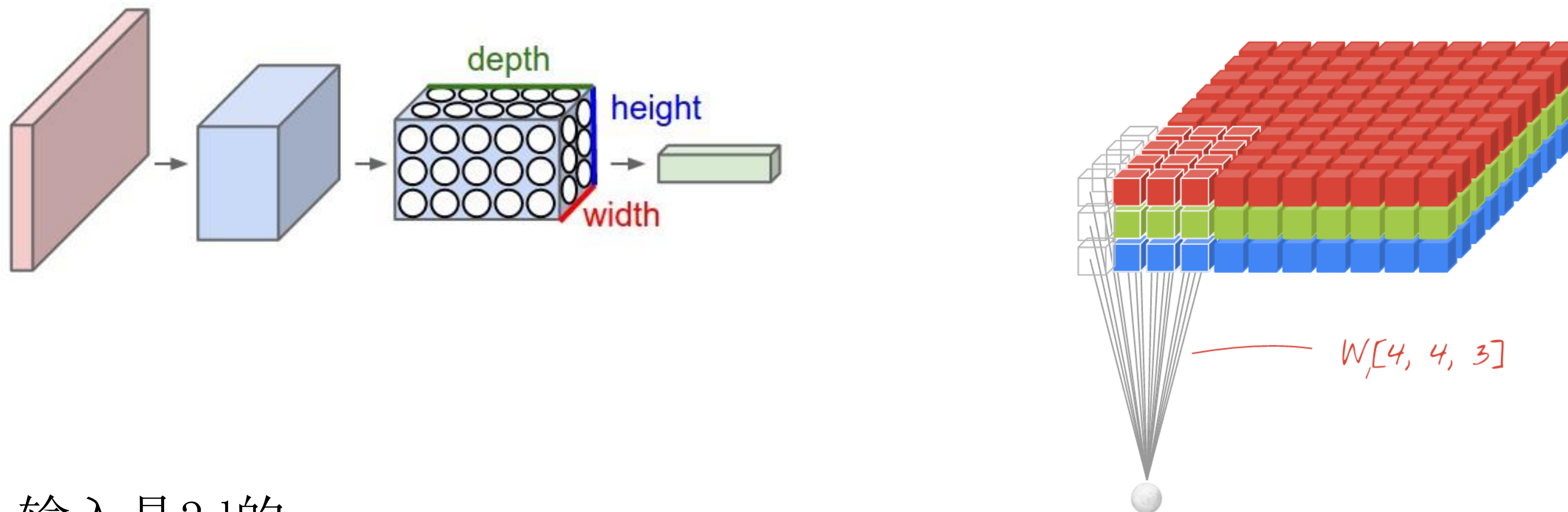


# CNN

- 卷积网络 (Convolutional neural network, CNN)
- 特点: 局部区域的权重 $W$ 共用 (**weight sharing**) (空间维度)
- 每一个卷积层后通常紧跟着一个下采样层 (subsample), 如最大池化 (max-pooling) 方法完成下采样。



# 3d卷积核



- 输入是3d的
- 有多个卷积核

# CNN组成

- 卷积层 (convolutional layers)
- 采样层 (pooling layers)
- 正则层 (normalization layers) (如 dropout)

# 卷积层

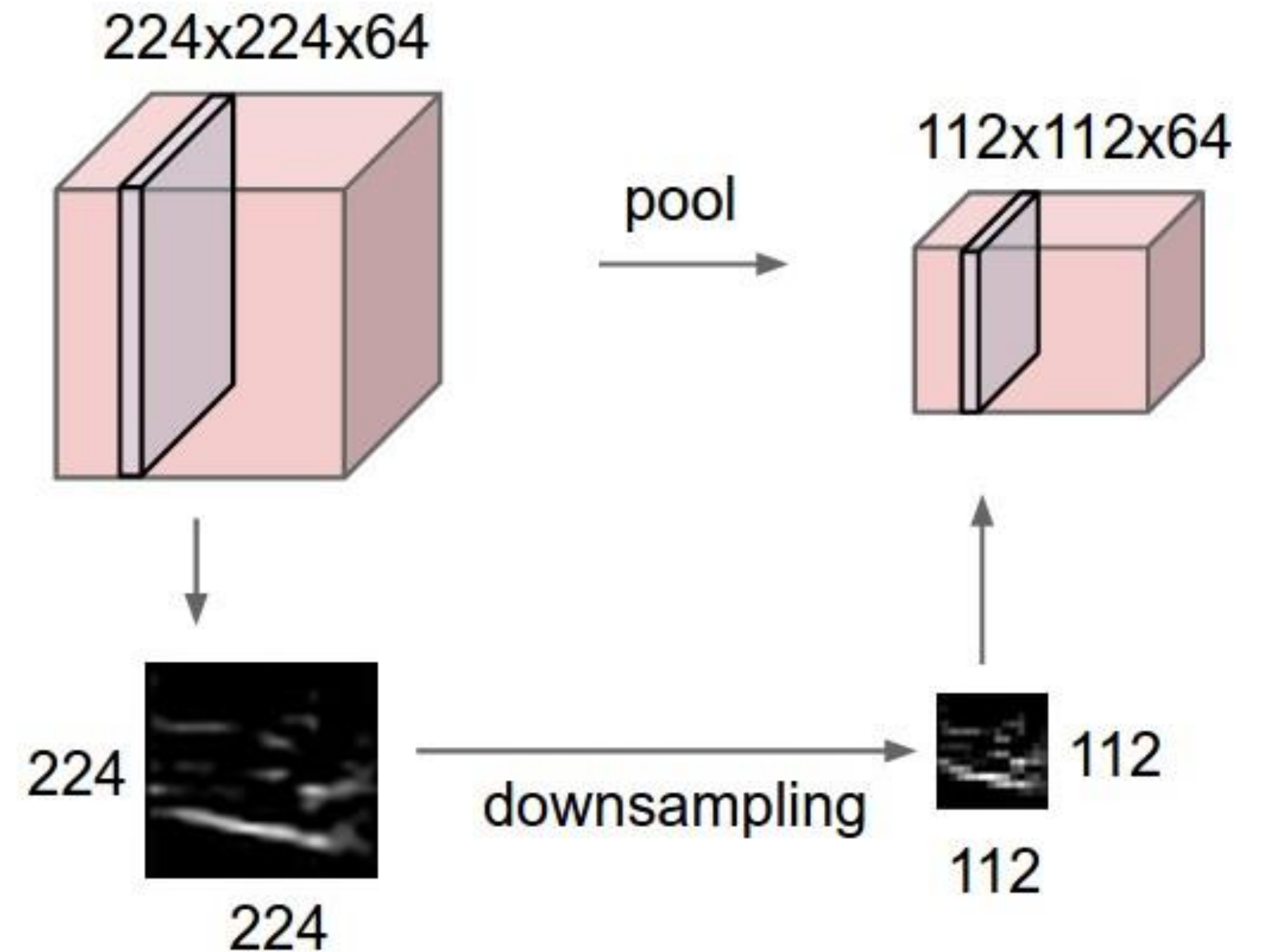
- 意义：用于处理图像.
- 排列结构：Layer的结构是3d
- 超参数：卷积核个数(D)，核大小(F)，padding(P)，strides(S)
- shape：
  - Input =  $W \times W \times 3$
  - $L = (W - F + 2P) / S + 1$
  - Layer =  $L \times L \times D$
  - Weights =  $F \times F \times D$
  - Output =  $L \times L \times D$

```
tf.layers.conv2d(  
    inputs,  
    filters,  
    kernel_size,  
    strides=(1, 1),  
    padding='valid',  
    data_format='channels_last',  
    dilation_rate=(1, 1),  
    activation=None,  
    use_bias=True,  
    kernel_initializer=None,  
    bias_initializer=tf.zeros_initializer(),  
    kernel_regularizer=None,  
    bias_regularizer=None,  
    activity_regularizer=None,  
    kernel_constraint=None,  
    bias_constraint=None,  
    trainable=True,  
    name=None,  
    reuse=None  
)
```



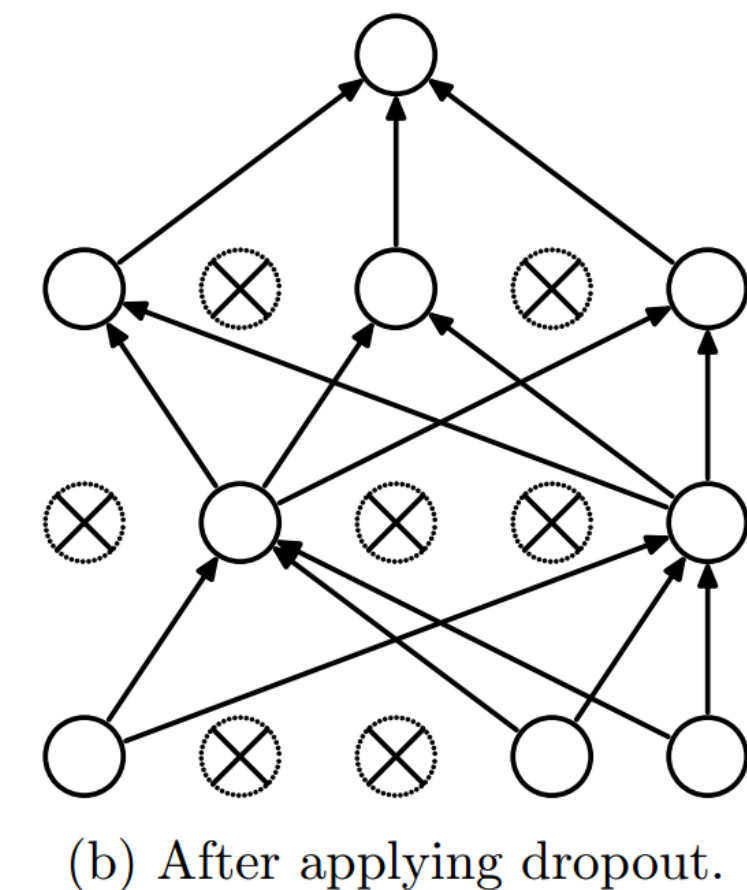
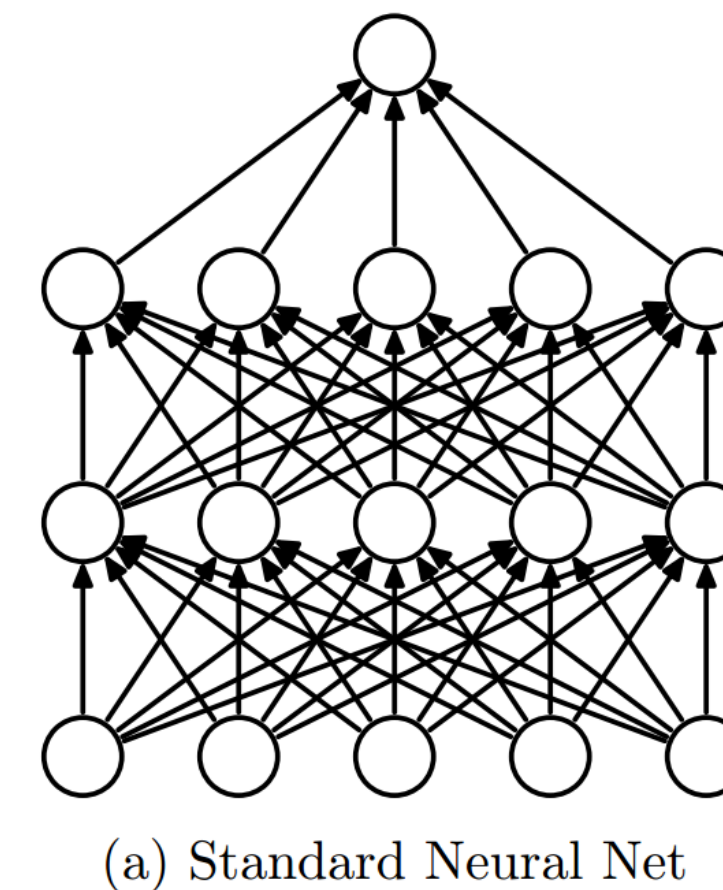
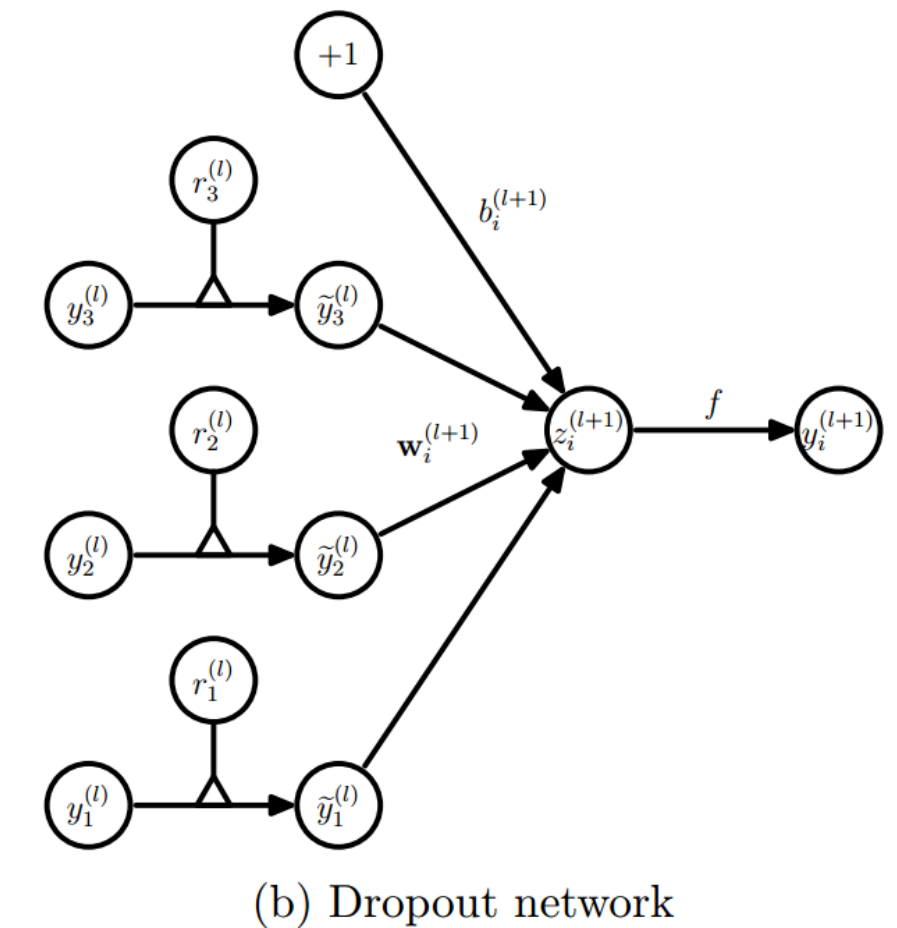
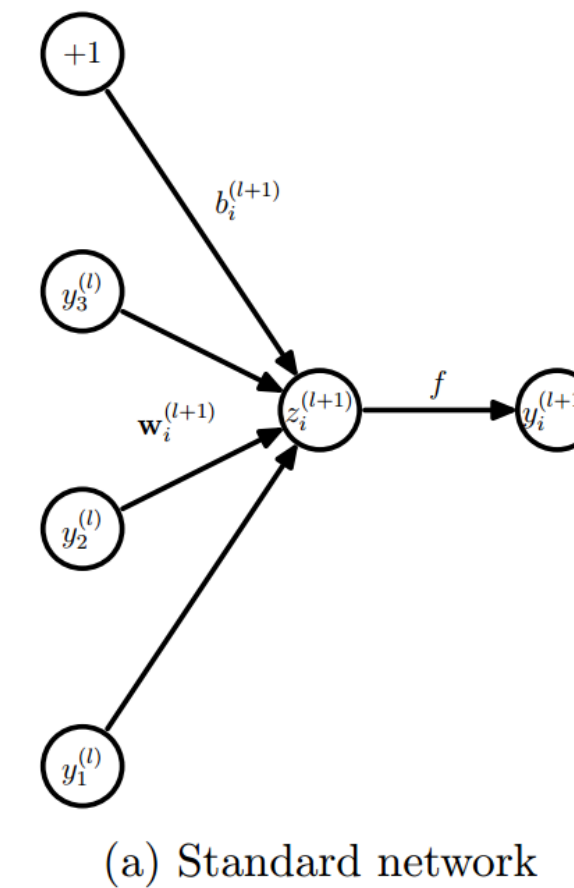
# Pooling层

- 意义：采样, 缩小模型大小
- 排列结构：Layer的结构是3d
- 超参数： `pooling_type`,  
`window_shape`, `padding`, `strides`
- 一个2\*2核, `strides=2`的pooling层,  
等于减少75%的输出
- pooling层并不会改变tensor的深度



# Dropout层

- 意义：减少CNN过拟合问题
- 超参数：keep\_prob 丢弃率
- 对于所有的输入，有keep\_prob概率保留并乘以 $1/\text{keep\_prob}$ ，以保证前后总和大致相等，否则输出0



# 共享Variable

```
def my_image_filter(input_images):  
    with tf.variable_scope("conv1"):  
        # Variables created here will be named "conv1/weights", "conv1/biases".  
        relu1 = conv_relu(input_images, [5, 5, 32, 32], [32])  
    with tf.variable_scope("conv2"):  
        # Variables created here will be named "conv2/weights", "conv2/biases".  
        return conv_relu(relu1, [5, 5, 32, 32], [32])
```



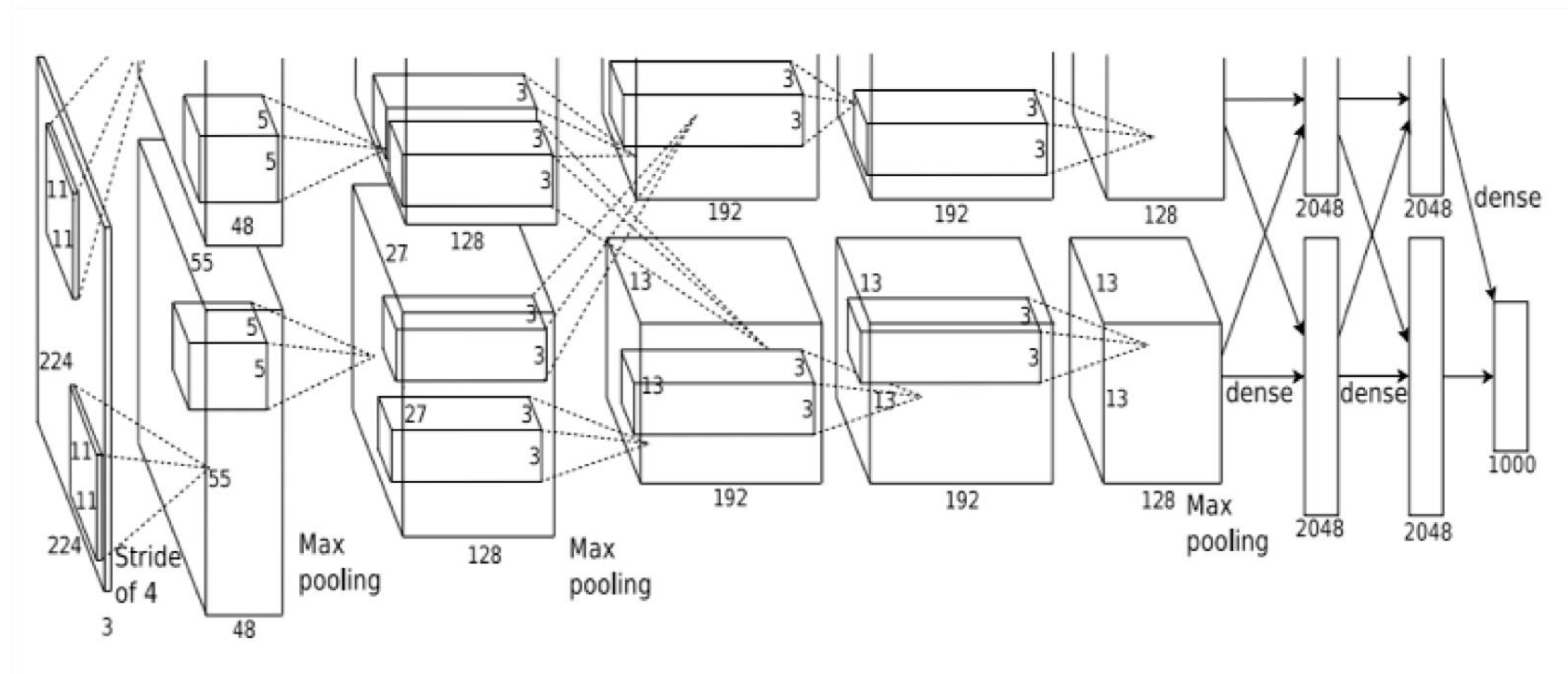
```
with tf.variable_scope("model") as scope:  
    output1 = my_image_filter(input1)  
scope.reuse_variables()  
    output2 = my_image_filter(input2)
```

生成两套参数

```
with tf.variable_scope("model") as scope:  
    output1 = my_image_filter(input1)  
    scope.reuse_variables()  
    output2 = my_image_filter(input2)
```

共享一套参数

# 课后阅读作业

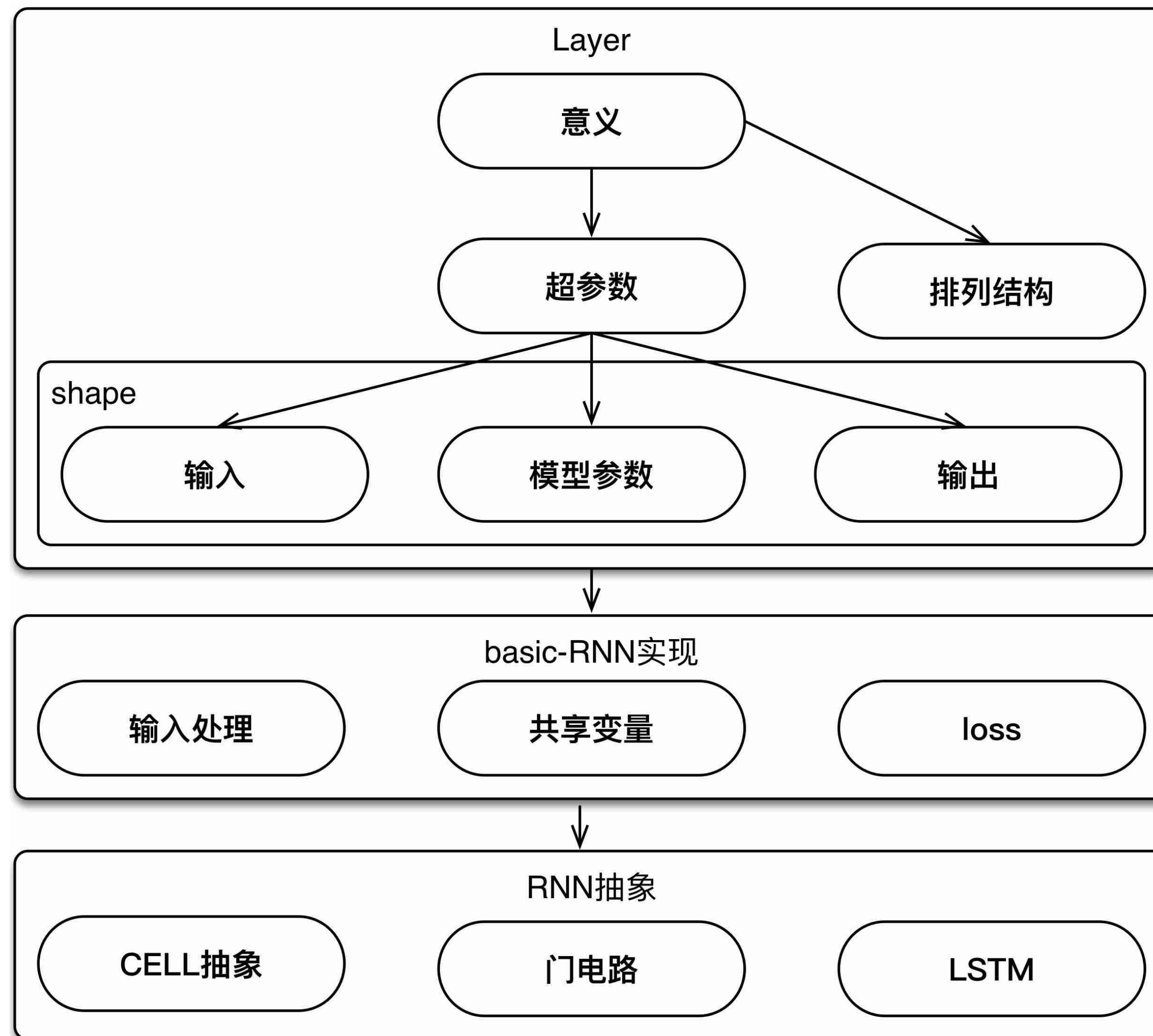


Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton. "ImageNet classification with deep convolutional neural networks." NIPS 2012.



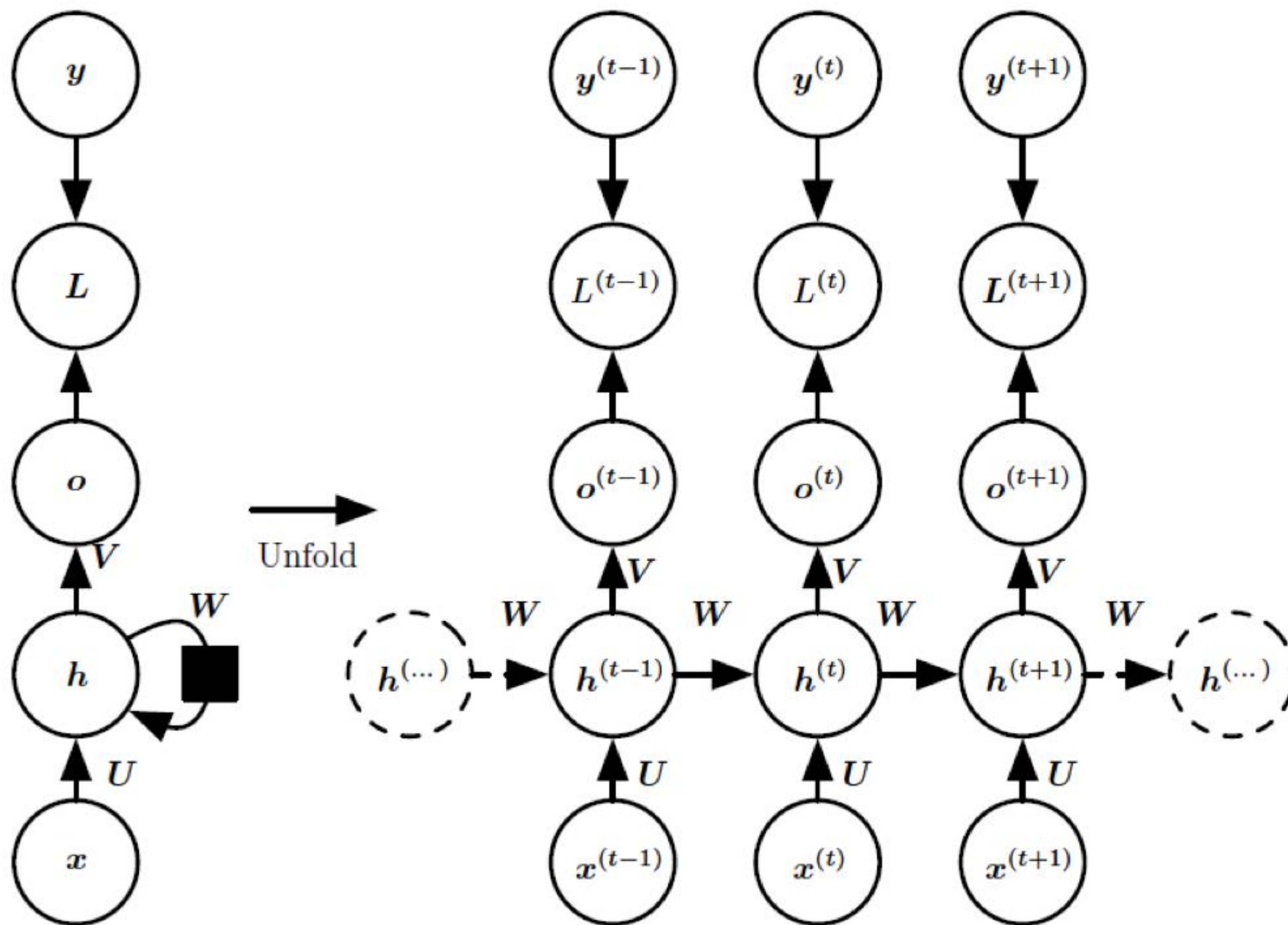
Simple RNN

# 学习路线



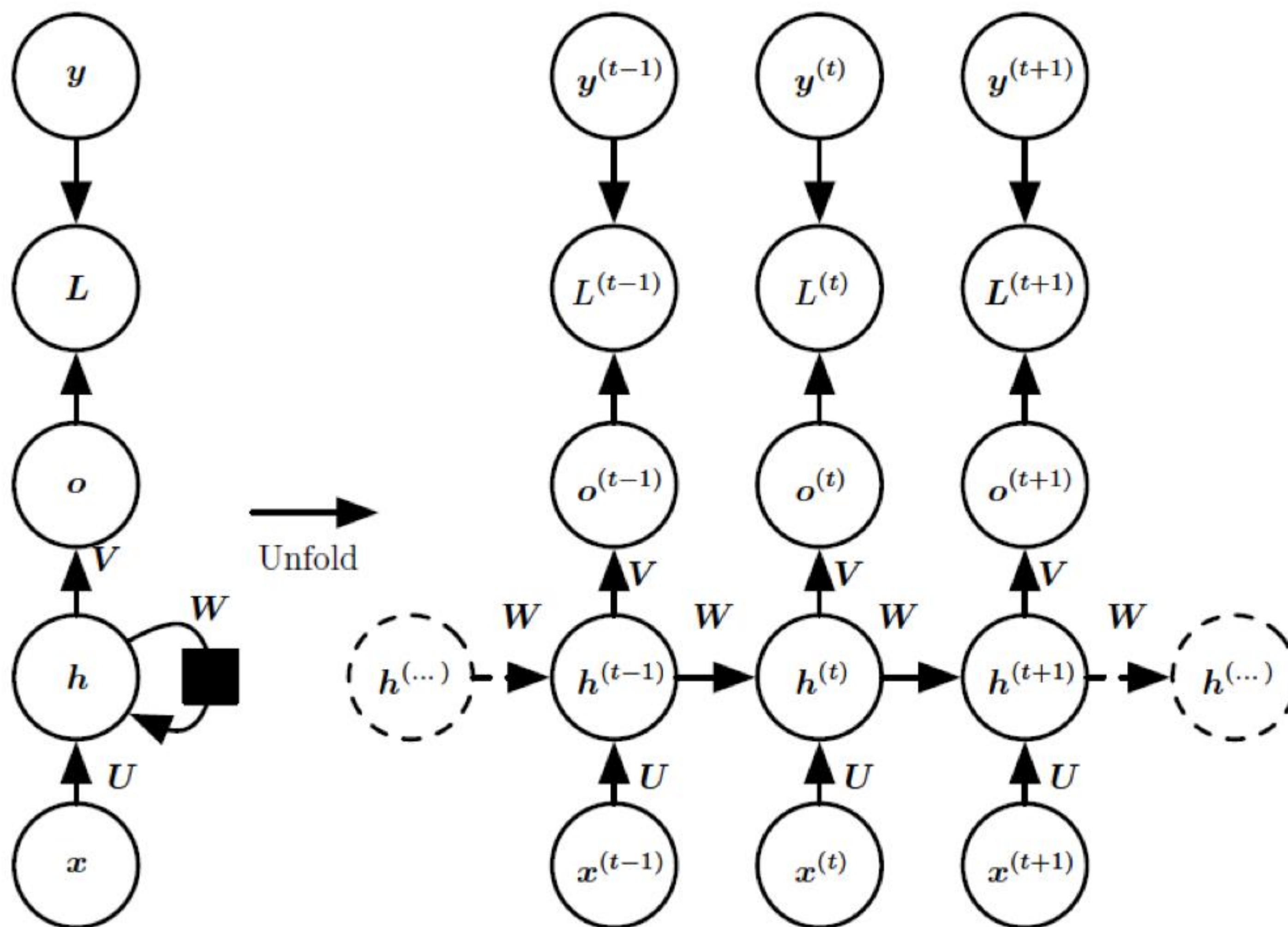
# RNN

- 循环网络结构
  - $y$  是训练目标
  - $L$  是损失函数
  - $o$  是网络输出
  - $h$  是状态（隐藏单元）
  - $x$  是网络输入
- 计算图的时间步上展开
- 举例：天气预测



# 循环网络-权重共享

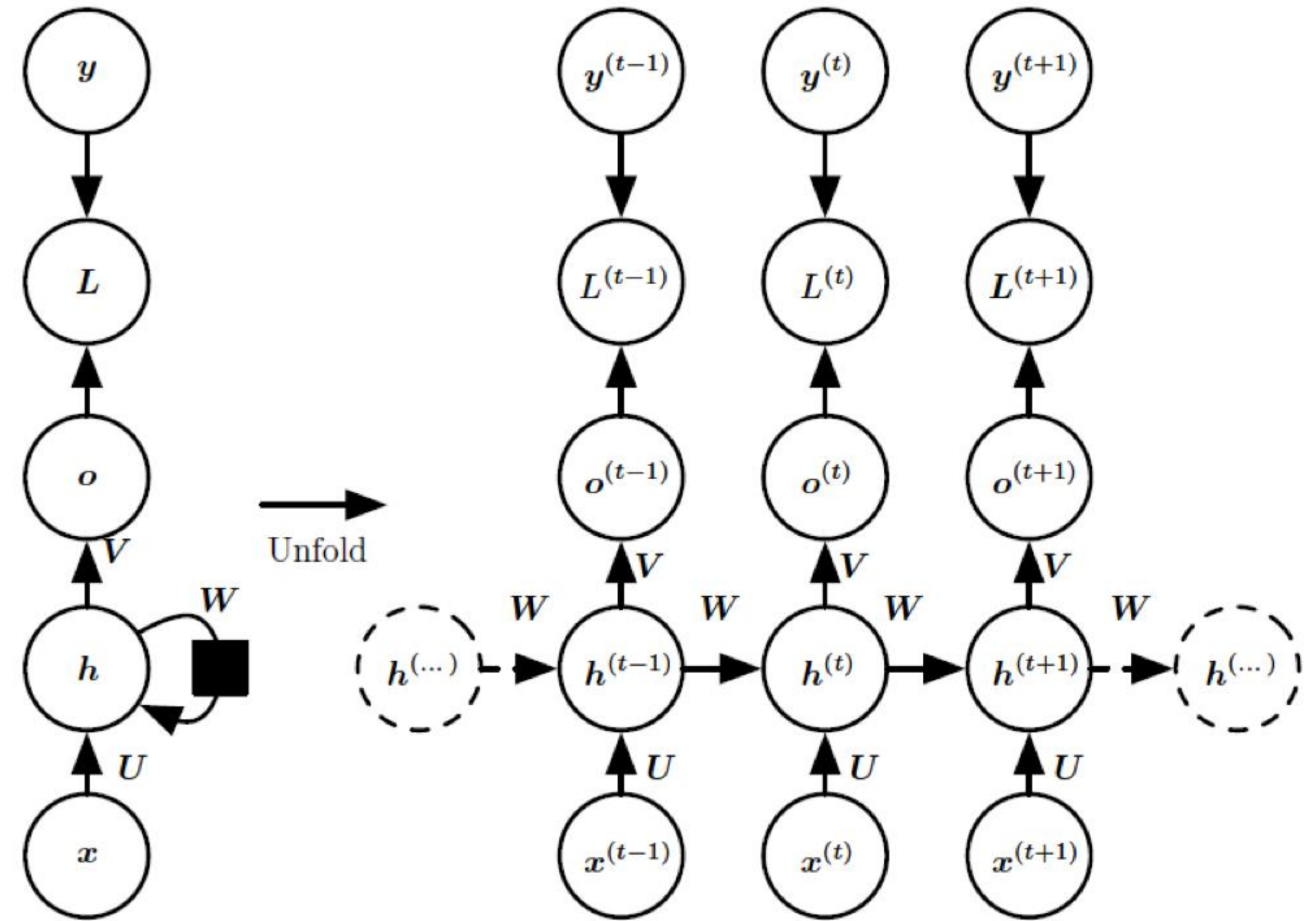
- 在不同的时间步上采用相同的U、V、W权重矩阵
- U: 输入到隐藏的连接的参数化的权重矩阵
- V: 隐藏到输出的连接的参数化的权重矩阵
- W: 隐藏到隐藏的循环连接的参数化的权重矩阵





# 计算图

$$\begin{aligned}a^{(t)} &= b + Wh^{(t-1)} + Ux^{(t)}, \\h^{(t)} &= \tanh(a^{(t)}), \\o^{(t)} &= c + Vh^{(t)}, \\\hat{y}^{(t)} &= \text{softmax}(o^{(t)}),\end{aligned}$$



- 循环网络将一个输入序列映射到相同长度的输出序列。
- 信息流动路径：信息在时间上向前（计算输出和损失）和向后（计算梯度）的思想。
- $U$ 、 $V$  和  $W$  分别对应于输入到隐藏、隐藏到输出和隐藏到隐藏的连接权重矩阵。
- $b$  和  $c$  是偏置向量。

# basic-rnn 实现

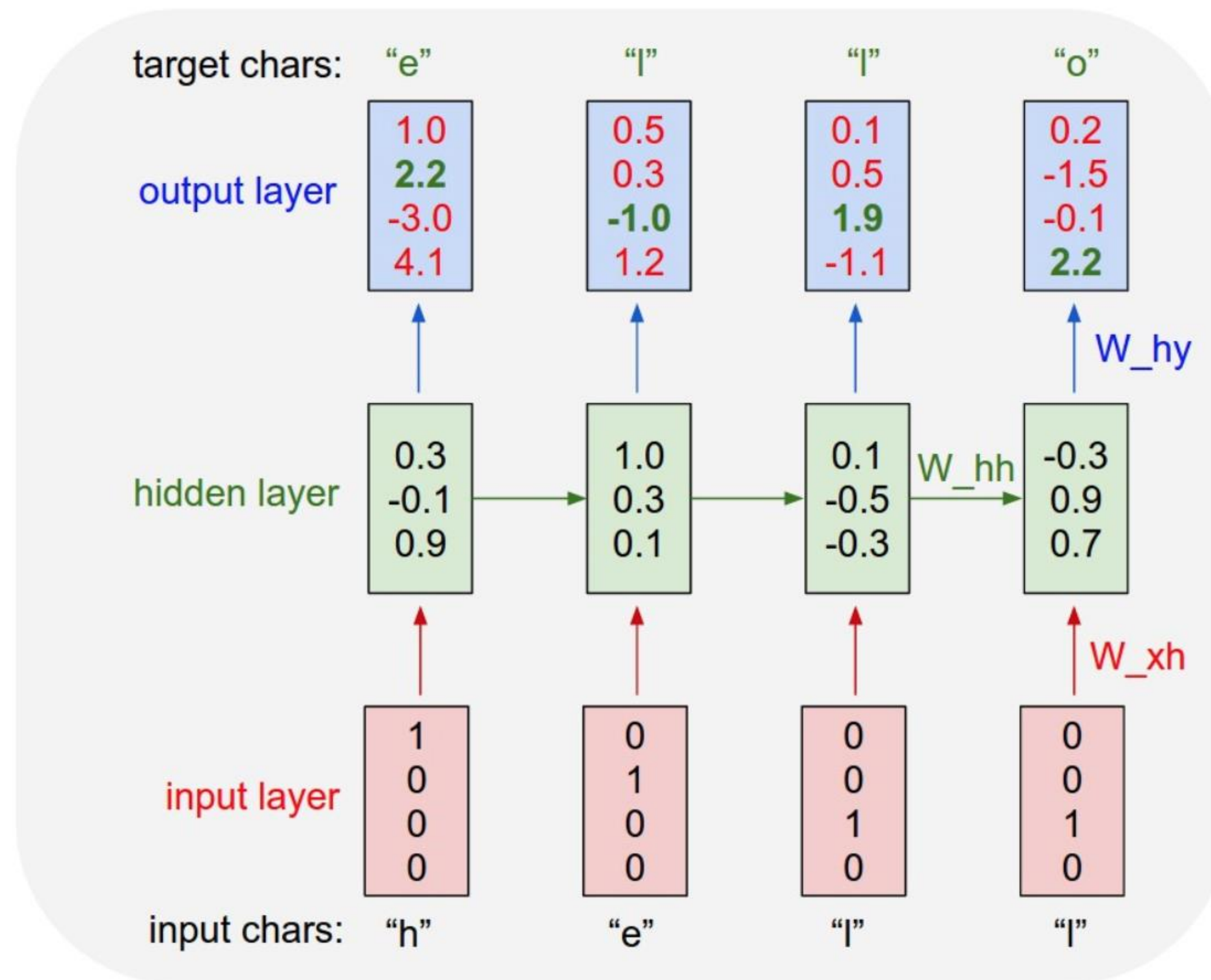
```
iter: 646500, p: 1300, loss: 0.019042
----
y = tf.get_variable("by", [vocab_size], initializer=initializer)
    by = tf.get_variable("by", [vocab_size], dtype=tf.float32, name="state:
hprev_val = np.zeros([1, hidden_size])

while True:
```

- Andrej Karpathy的min-char-rnn tf版本实现
- 实现了一个自动写代码的程序，输入程序就是本身

# RNN示例

- RNN在不同的时间步上采用相同的U、V、W参数
- 即 $W_{xh}$ ,  $W_{hh}$ ,  $W_{hy}$
- 尼采的文集示例





# 输入和loss处理

- 给定序列长度(模型超参数), 把输入序列化
- one-hot 离散化处理
- U, V, W 共享权重
- 收集所有时刻的输出, 计算的loss
- 梯度截取预防梯度爆炸

$$\begin{aligned} L(\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(\tau)}\}, \{\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(\tau)}\}) \\ &= \sum_t L^{(t)} \\ &= - \sum_t \log p_{\text{model}}(y^{(t)} \mid \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(t)}\}) \end{aligned}$$



# rnn-cell抽象

```
__init__(  
    num_units,  
    activation=None,  
    reuse=None,  
    name=None  
)
```

Cell超参数: num\_units

```
__call__(  
    inputs,  
    state,  
    scope=None,  
    *args,  
    **kwargs  
)
```

调用时刻要输入state

## rnn-example

```
tf.nn.static_rnn(  
    cell,  
    inputs,  
    initial_state=None,  
    dtype=None,  
    sequence_length=None,  
    scope=None  
)
```

static-rnn抽象

```
state = cell.zero_state(...)  
outputs = []  
for input_ in inputs:  
    output, state = cell(input_, state)  
    outputs.append(output)  
return (outputs, state)
```

[https://tensorflow.google.cn/api\\_docs/python/tf/keras/layers/SimpleRNNCell](https://tensorflow.google.cn/api_docs/python/tf/keras/layers/SimpleRNNCell)

[https://tensorflow.google.cn/api\\_docs/python/tf/contrib/rnn/BasicRNNCell](https://tensorflow.google.cn/api_docs/python/tf/contrib/rnn/BasicRNNCell)

# rnn-cell抽象

- `hidden-units` : 模型的容量大小
- $I(\text{input}) + S(\text{state}) \rightarrow O(\text{output}) + S(\text{new\_state})$
- `inputs`: 输入
- `state`: 隐含了之前所有的输出信息
- 当前的输出完全取决于`state`和当前的输入

# keras.layers.RNN(cell)

## Class SimpleRNN

Fully-connected RNN where the output is to be fed back to input.

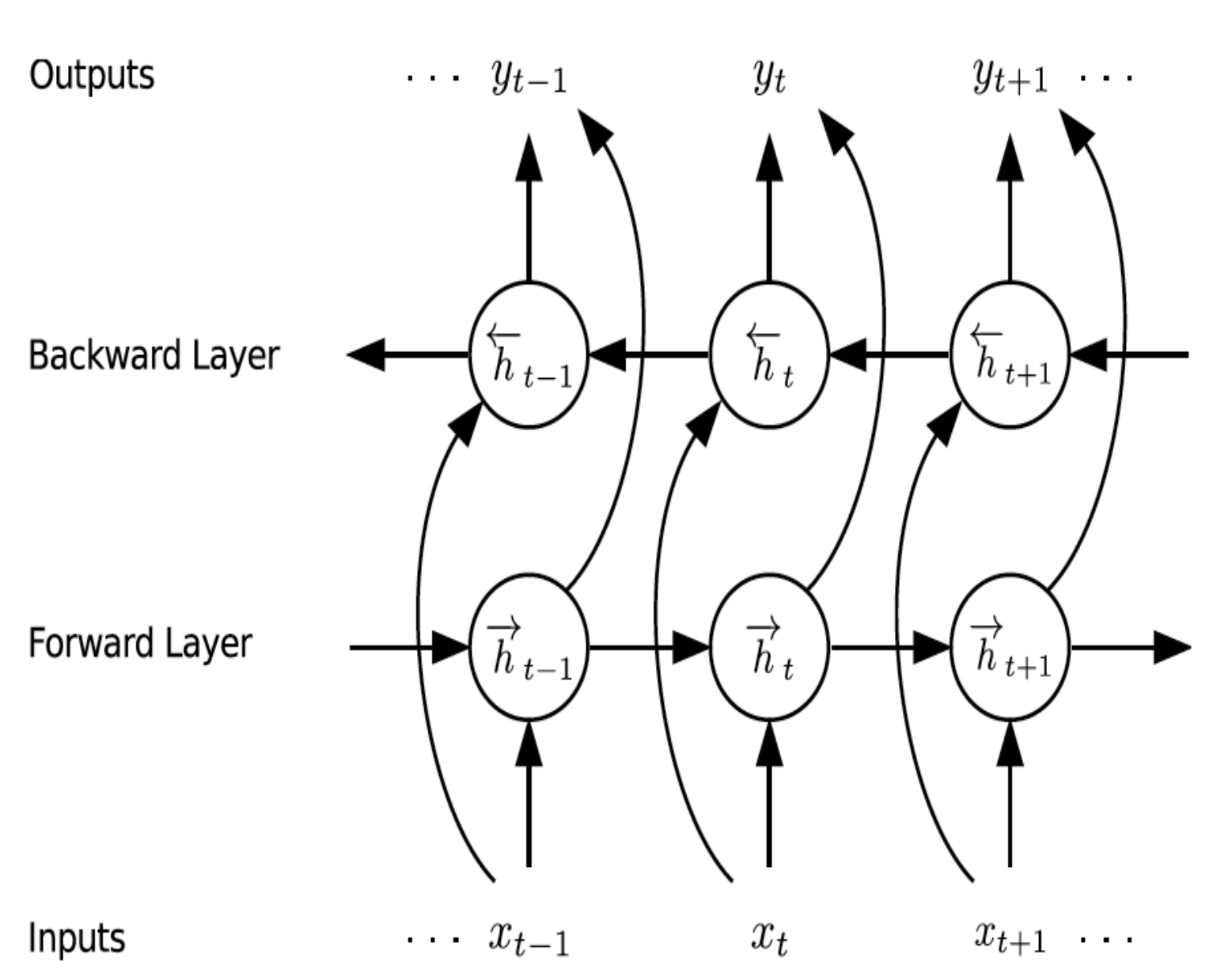
units: Positive integer, dimensionality of the output space.

[https://tensorflow.google.cn/api\\_docs/python/tf/keras/layers/SimpleRNN](https://tensorflow.google.cn/api_docs/python/tf/keras/layers/SimpleRNN)

[https://tensorflow.google.cn/api\\_docs/python/tf/keras/layers/RNN?hl=en](https://tensorflow.google.cn/api_docs/python/tf/keras/layers/RNN?hl=en)

```
__init__(  
    units,  
    activation='tanh',  
    use_bias=True,  
    kernel_initializer='glorot_uniform',  
    recurrent_initializer='orthogonal',  
    bias_initializer='zeros',  
    kernel_regularizer=None,  
    recurrent_regularizer=None,  
    bias_regularizer=None,  
    activity_regularizer=None,  
    kernel_constraint=None,  
    recurrent_constraint=None,  
    bias_constraint=None,  
    dropout=0.0,  
    recurrent_dropout=0.0,  
    return_sequences=False,  
    return_state=False,  
    go_backwards=False,  
    stateful=False,  
    unroll=False,  
    **kwargs  
)
```

# 课后作业



$$\vec{h}_t = \mathcal{H} \left( W_{x\vec{h}} x_t + W_{\vec{h}\vec{h}} \vec{h}_{t-1} + b_{\vec{h}} \right)$$

$$\overleftarrow{h}_t = \mathcal{H} \left( W_{x\overleftarrow{h}} x_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h}_{t+1} + b_{\overleftarrow{h}} \right)$$

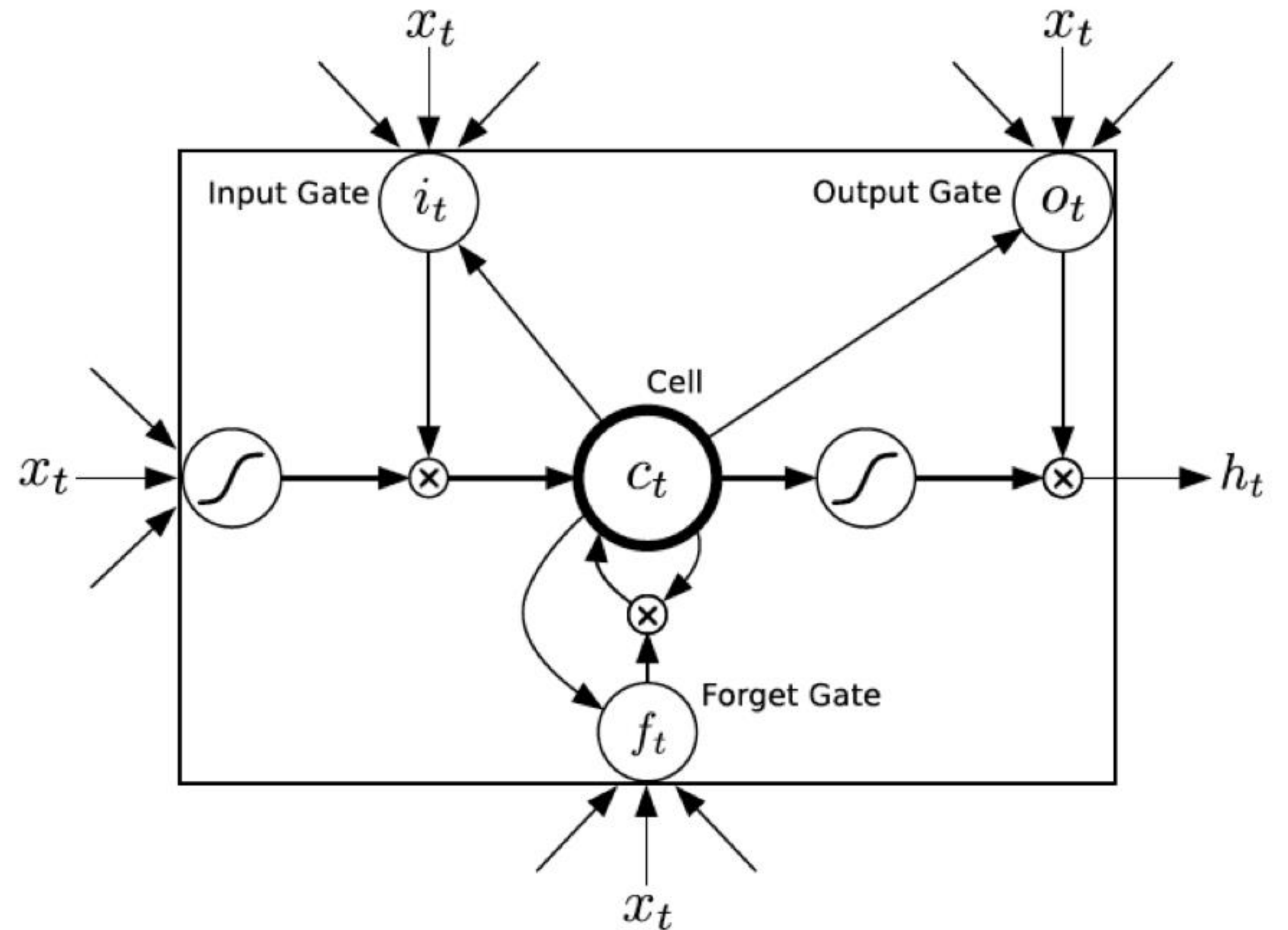
$$y_t = W_{\vec{h}y} \vec{h}_t + W_{\overleftarrow{h}y} \overleftarrow{h}_t + b_y$$

- [x] Alex Graves et al., Speech recognition with deep recurrent neural networks, ICASSP 2013.

# RNN & LSTM

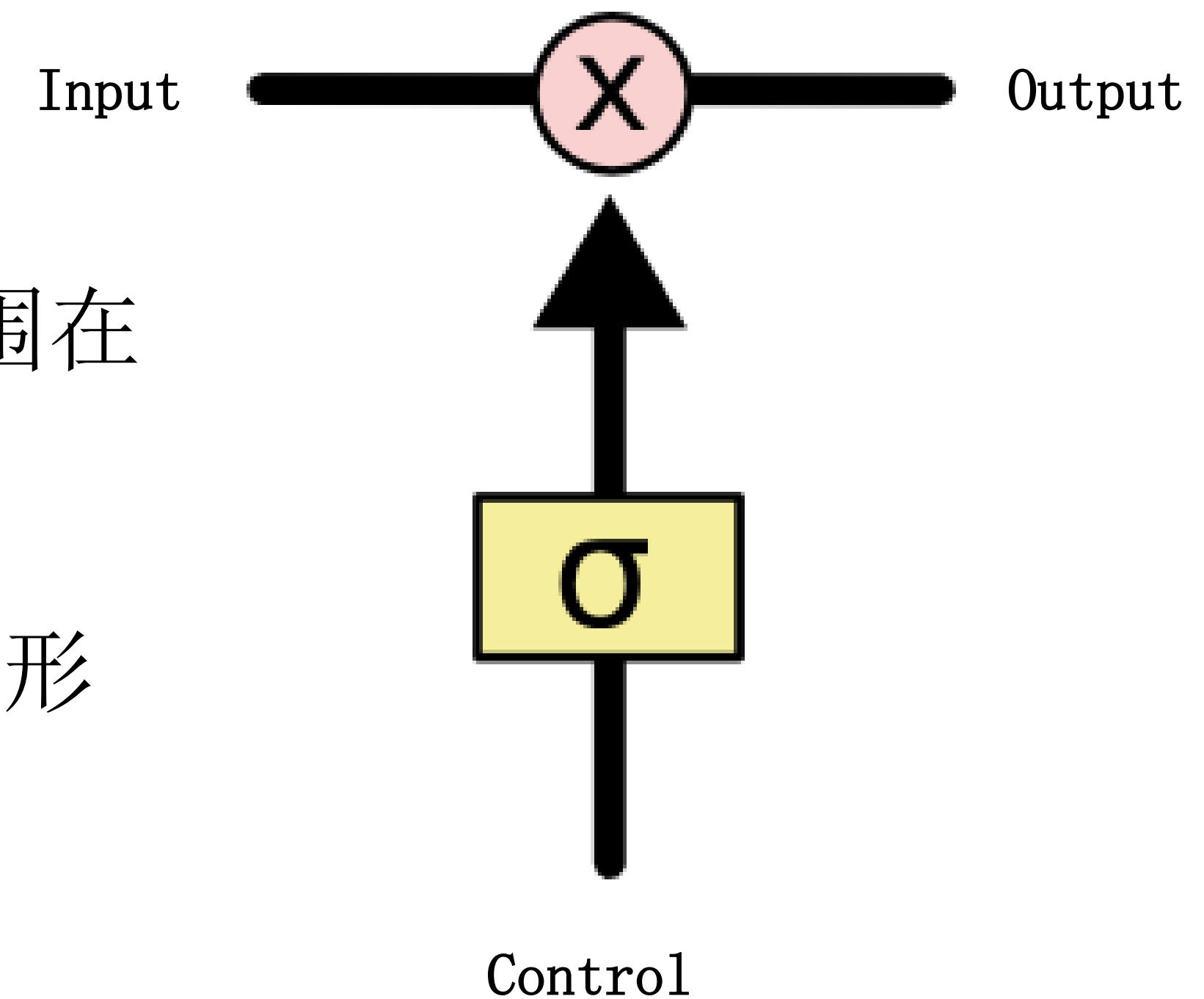
# LSTM

- RNN训练有以下问题
  - RNN梯度爆炸
  - RNN梯度消失
- LSTM解决以上问题



# 门电路

- Input和Control形状一致
- Control经过Sigmoid函数后，变成一个范围在0-1之间的一个同形状的Tensor
- Input和  $\sigma$  (Control) 元素相乘得到一个同形的Output



# LSTM

- LSTM是RNN的一个改进，LSTM增加了一个辅助记忆单元和其他三个辅助的门限输入单元。
- 输入门（Input gate）控制是否输入，遗忘门（Forget gate）控制是否存储，输出门（Output gate）控制是否输出。
- 辅助记忆单元可以寄存时间序列的输入，在训练过程中会利用后向传播的方式进行。
- 记忆单元和门单元的组合，提升了RNN处理远距离依赖问题的能力，解决RNN网络收敛慢的问题。

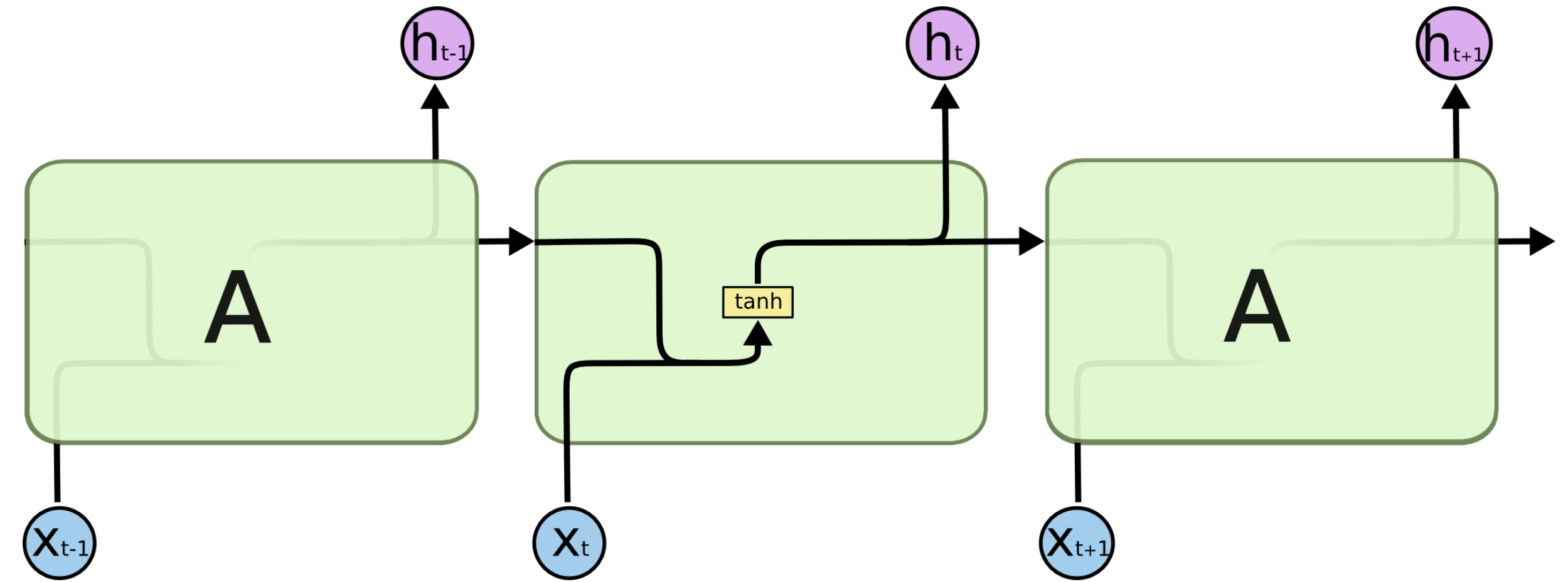


# RNN与LSTM

- RNN

$$h_t = \mathcal{H}(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \quad (1)$$

$$y_t = W_{hy}h_t + b_y \quad (2)$$



- LSTM

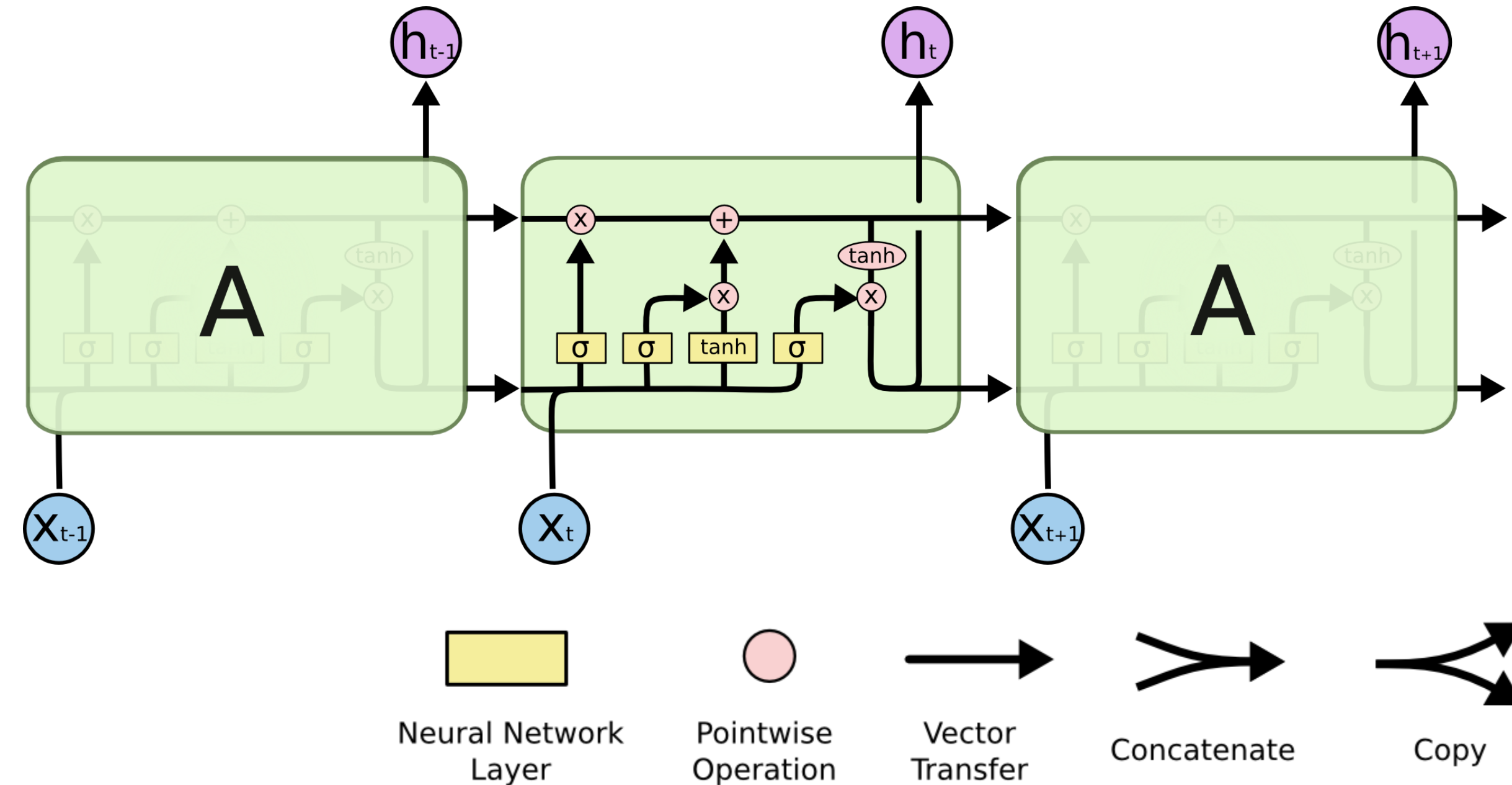
$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \quad (3)$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \quad (4)$$

$$c_t = f_t c_{t-1} + i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \quad (5)$$

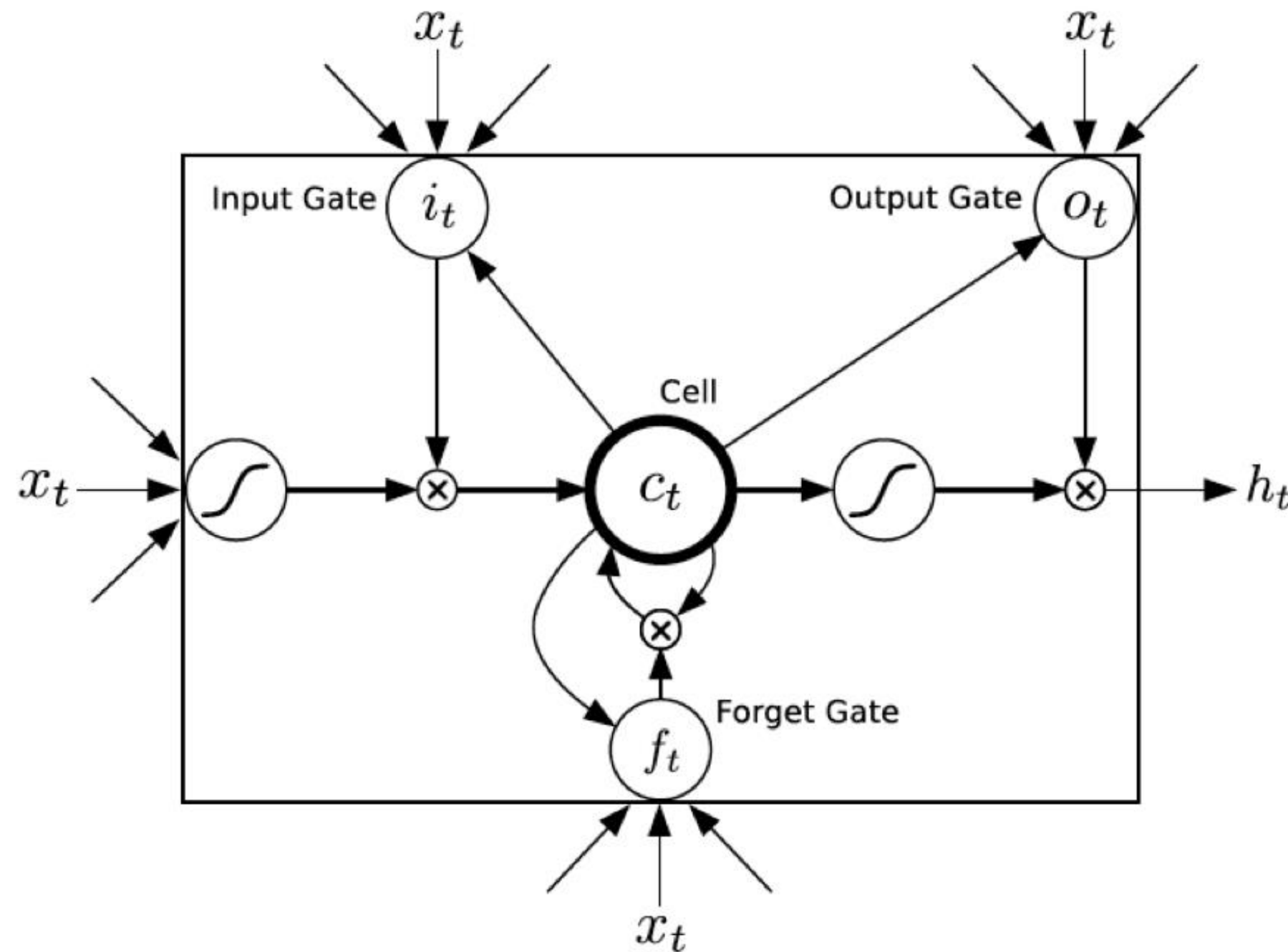
$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \quad (6)$$

$$h_t = o_t \tanh(c_t) \quad (7)$$



# LSTMCe11

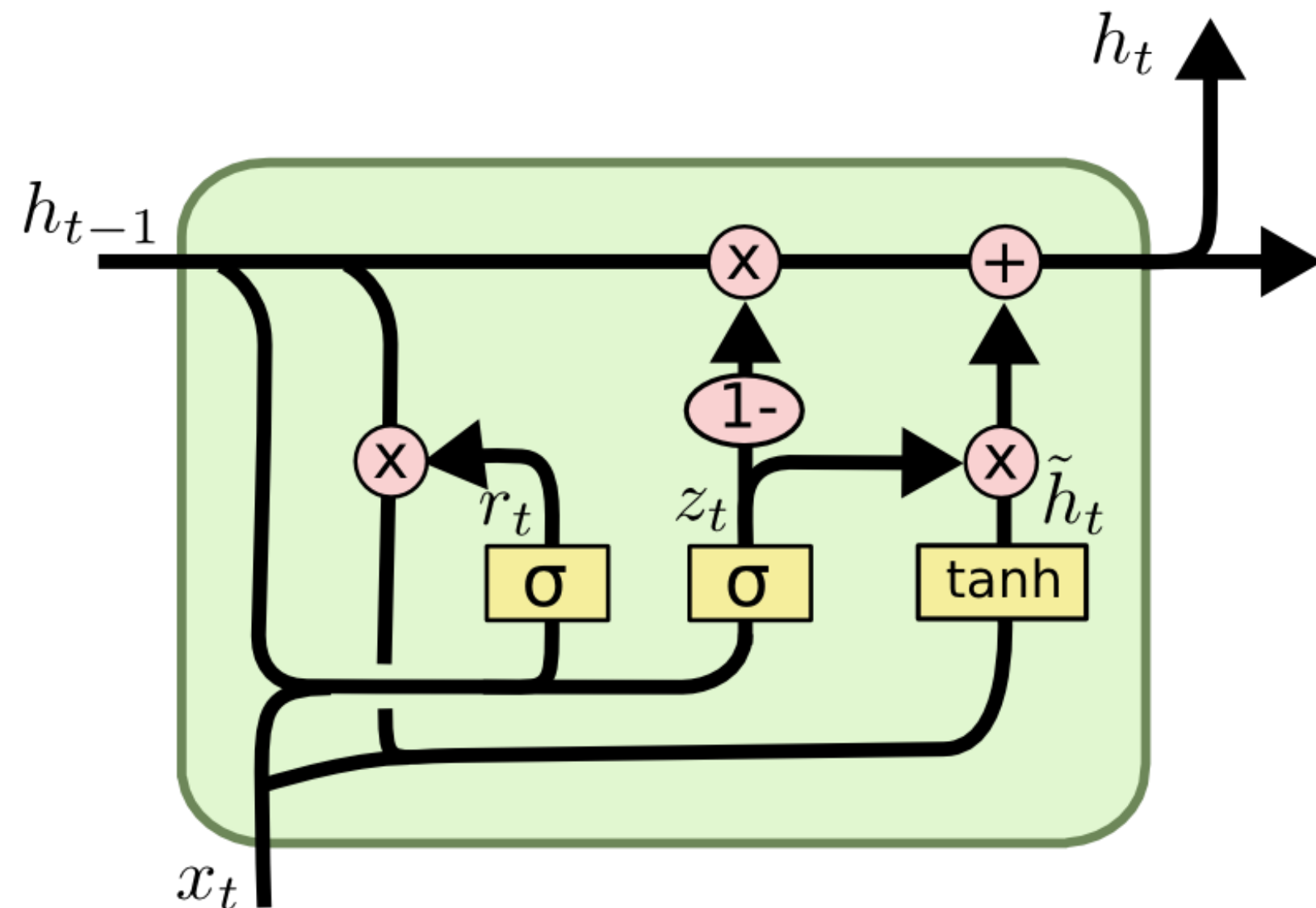
- LSTM internal



```
__init__(  
    units,  
    activation='tanh',  
    recurrent_activation='sigmoid',  
    use_bias=True,  
    kernel_initializer='glorot_uniform',  
    recurrent_initializer='orthogonal',  
    bias_initializer='zeros',  
    unit_forget_bias=True,  
    kernel_regularizer=None,  
    recurrent_regularizer=None,  
    bias_regularizer=None,  
    kernel_constraint=None,  
    recurrent_constraint=None,  
    bias_constraint=None,  
    dropout=0.0,  
    recurrent_dropout=0.0,  
    implementation=2,  
    **kwargs  
)
```

# GRU

- Gated Recurrent Unit (GRU)
- 组合遗忘门 (forget gates) 和输入门 (input gates) 为单一的更新门 (Update gates)
- 合并LSTM的cell状态 ( cell state ) 和隐藏态 (hidden state)



$$z_t = \sigma (W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma (W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = \tanh (W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

# 推荐阅读

- Understanding Convolutions
  - <http://colah.github.io/posts/2014-07-Understanding-Convolutions/>
- Conv Nets: A Modular Perspective
  - <http://colah.github.io/posts/2014-07-Conv-Nets-Modular/>
- The Unreasonable Effectiveness of Recurrent Neural Networks
  - <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>
- Understanding-LSTMs
  - <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

谢谢！