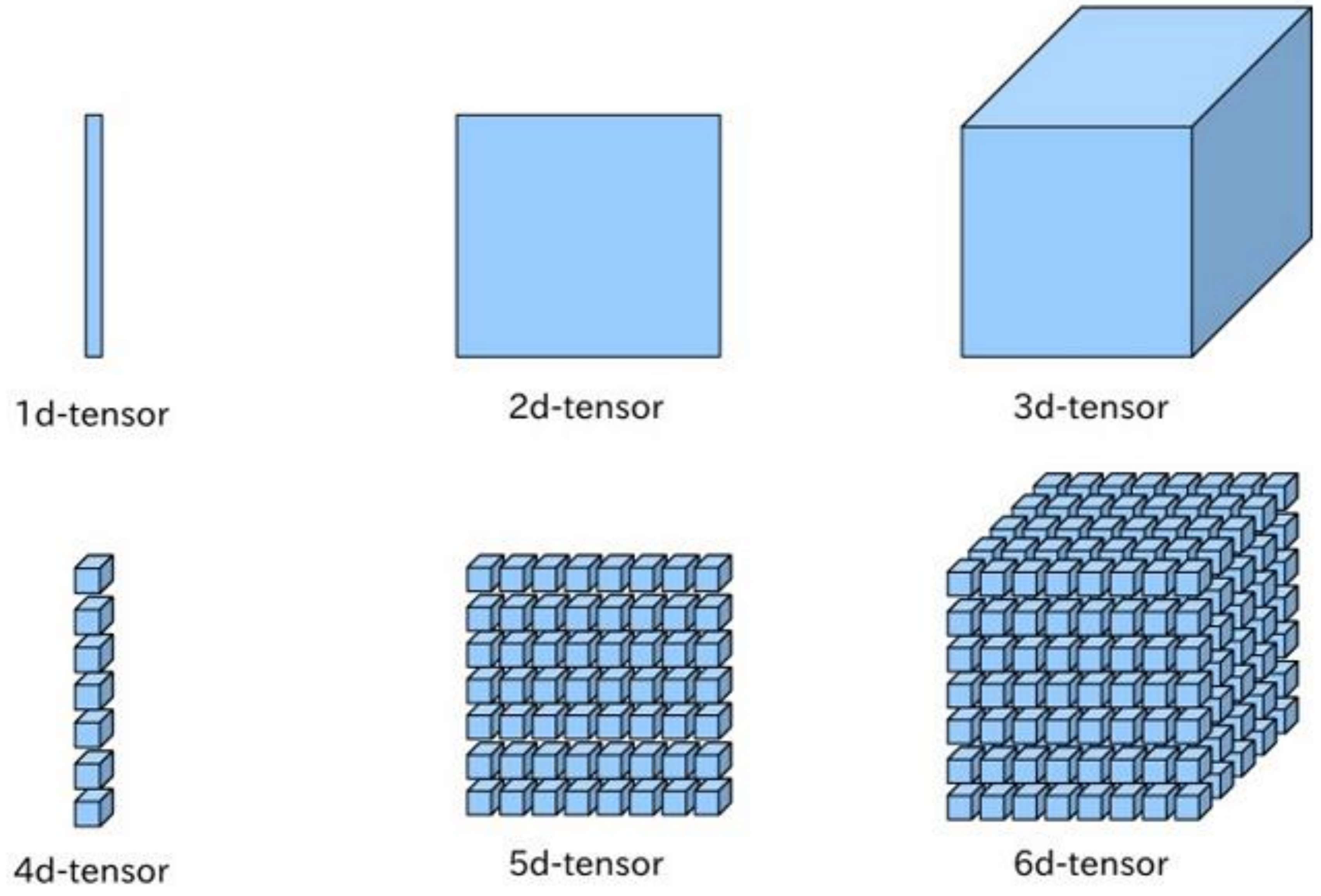
Dense & CNN & RNN

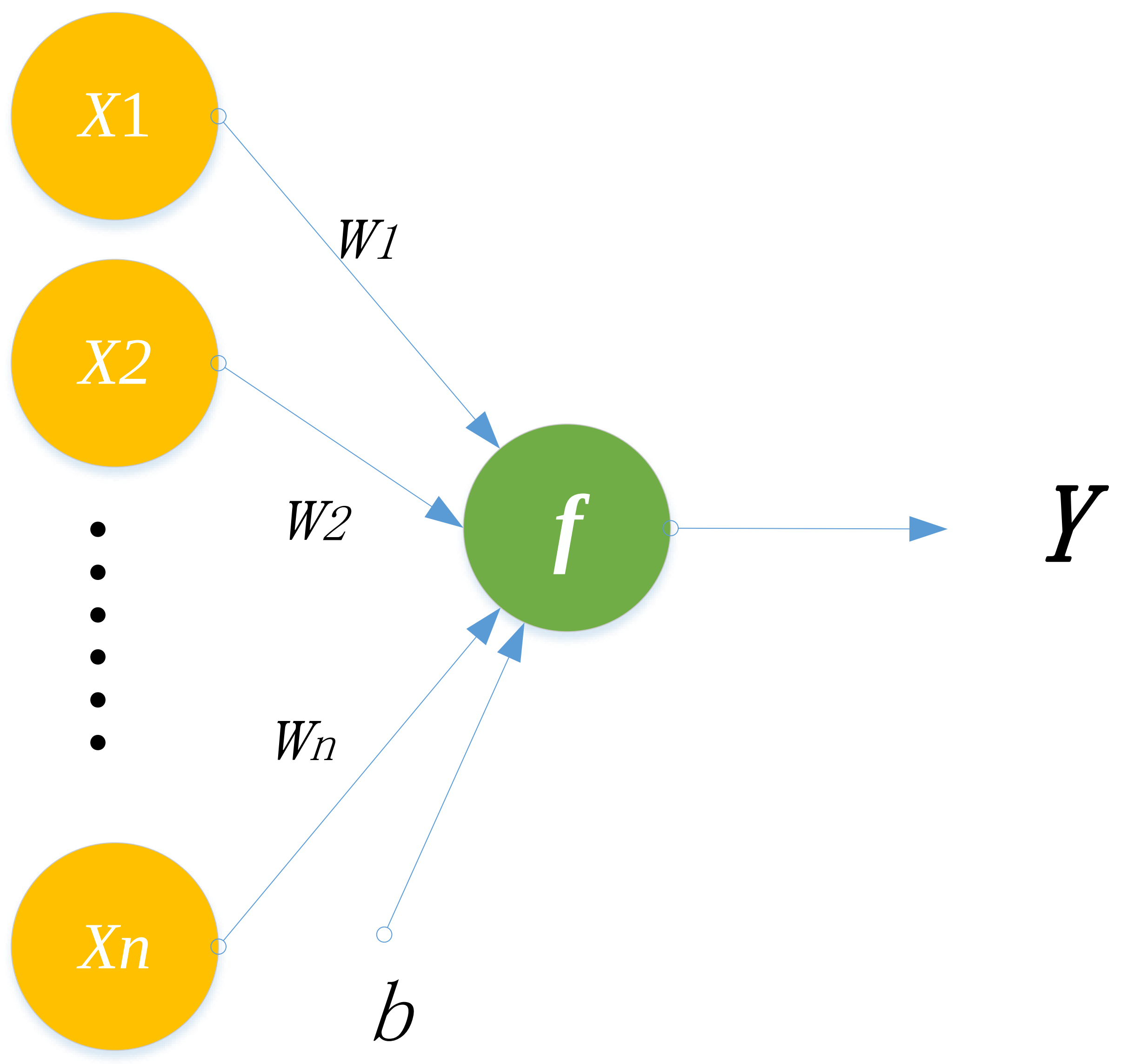
Tensor形象化表示

- 在TensorFlow 2中，Tensor是基础
- 对于一个4*5*6的Tensor，以下属性值
  - rank ： 3d
  - length: 4, 5, 6
  - shape: [4, 5, 6]
  - volume: 4*5*6=120



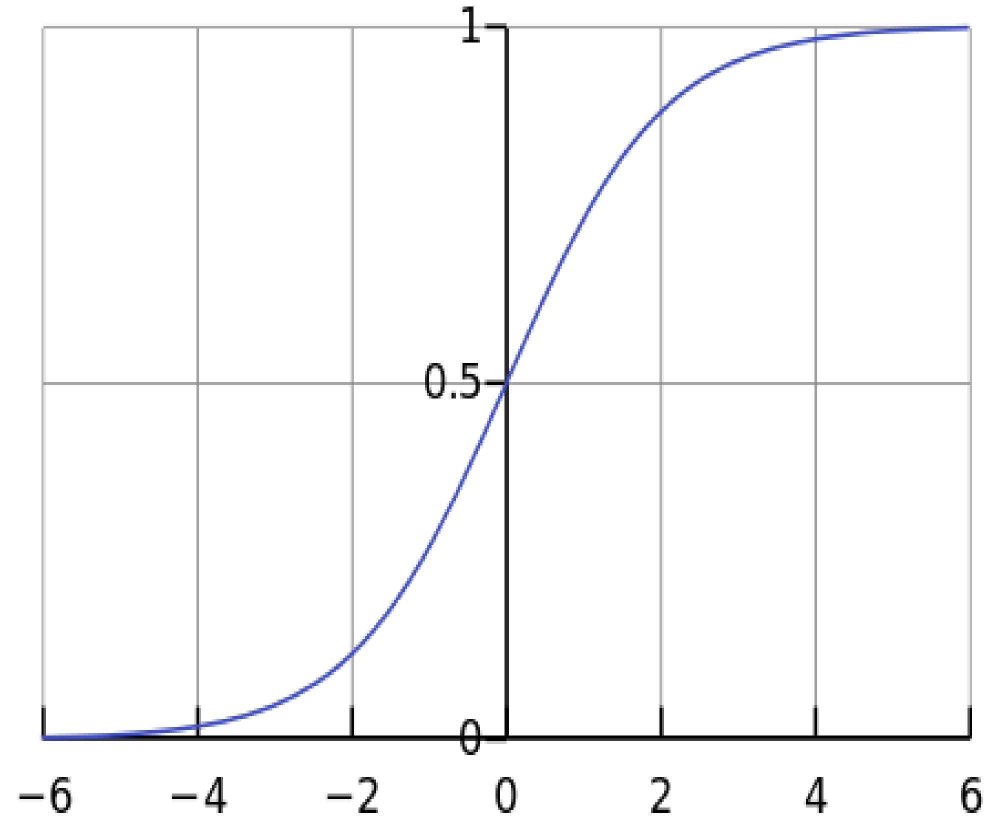
单个人工神经元

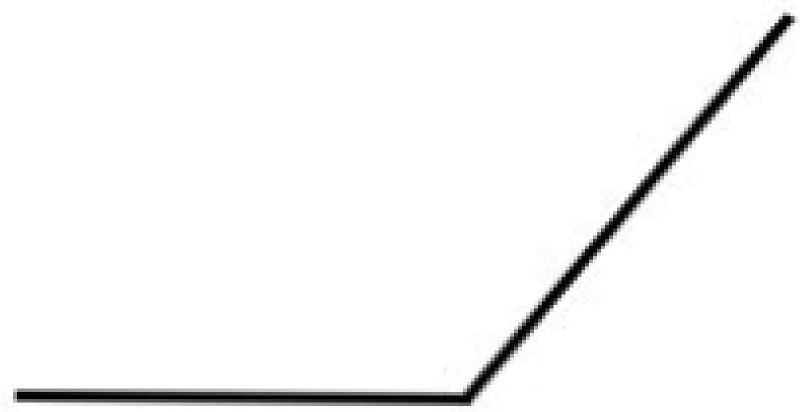
- 单个人工神经元（Artificial Neuron）
  - 一组输入(Input)的线性加权叠加（Sum）
  - 经过一个非线性变换f，输出变换值(Output)



$$y=f\left(\sum_{i=1}^{N} W_i X_i + b\right)$$

logistic_regression.ipynb

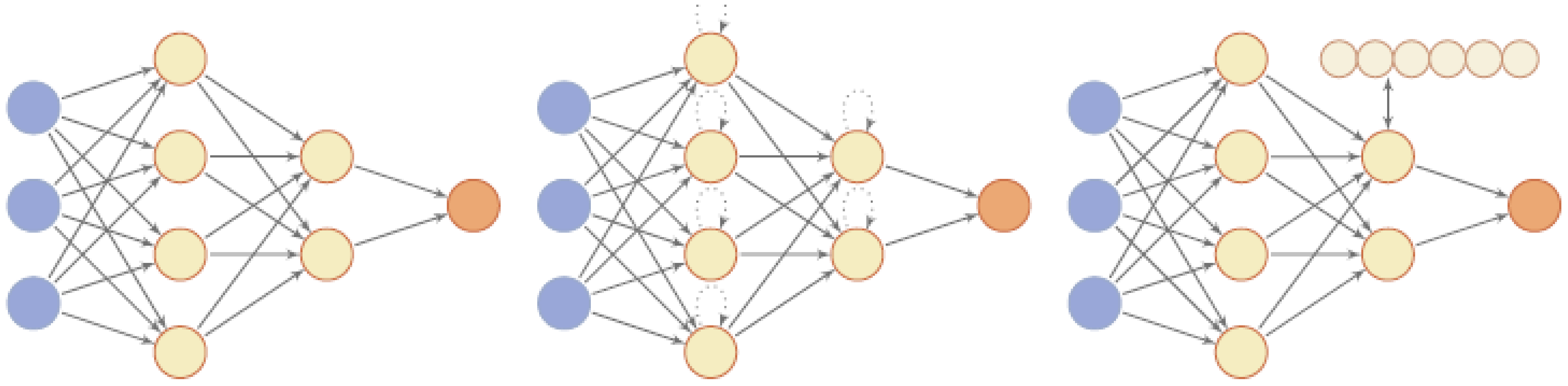




$F(x) = \max(0, x)$

深度网络

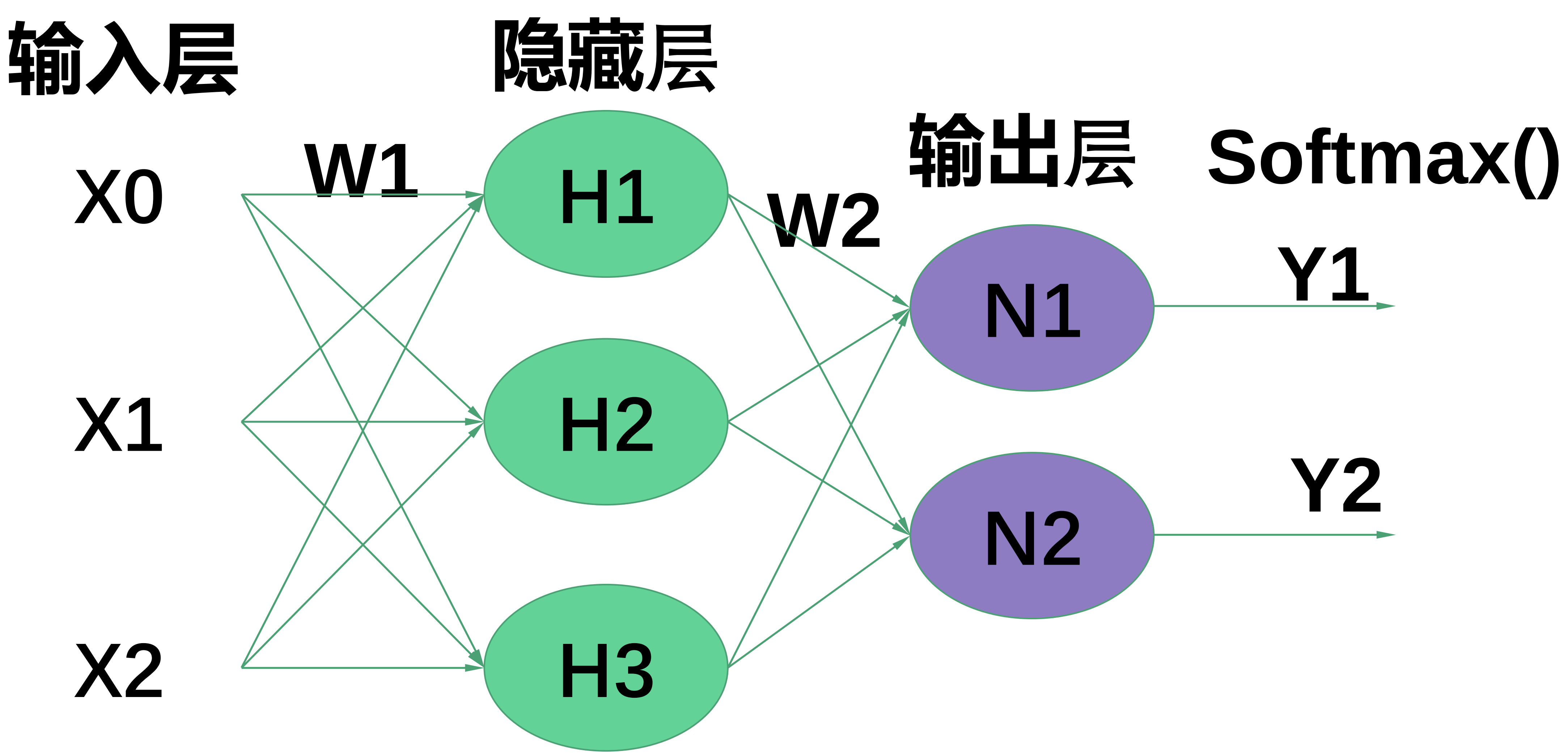
- 深度网络（Deep Networks）
  - 按照拓扑连接结构，将大量的神经元组织起来，构成规模化的深度的网络。
- 网络的拓扑结构，即每层的连接关系（Layers）
  - 每层神经元之间的连接关系，即层间的连接关系(Layers)。
- 前馈网络（feedforward）、反馈网络（feedback）和记忆网络（memory network）



(a) 前馈网络　(b) 反馈网络　(c) 记忆网络

多层网络（Multilayer networks）

- 前馈网络（Multilayer feedforward networks，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(Input layer)、隐藏层(hidden layer)，输出层(Output Layer)；
  - 隐藏层、输入层，共有5个神经元。



neural_network.ipynb

Tensorflow and Keras

- Keras 是一套高级API，用来**快速搭建**深度神经网络；
- http://Keras.io
- tf.keras 是 基于TensorFlow实现的程序库

```
import tensorflow as tf

from tensorflow import keras
```

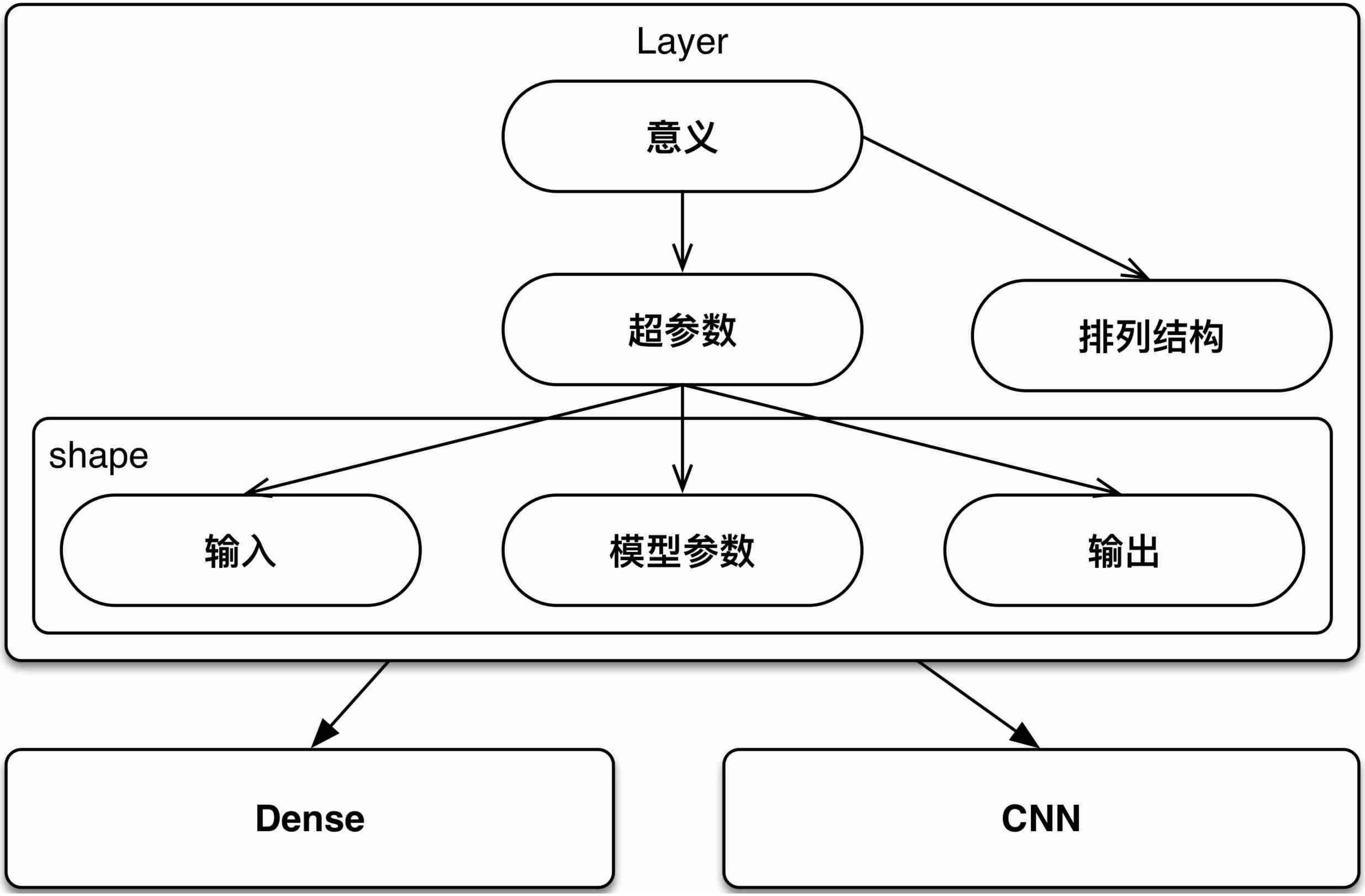
Keras

TensorFlow

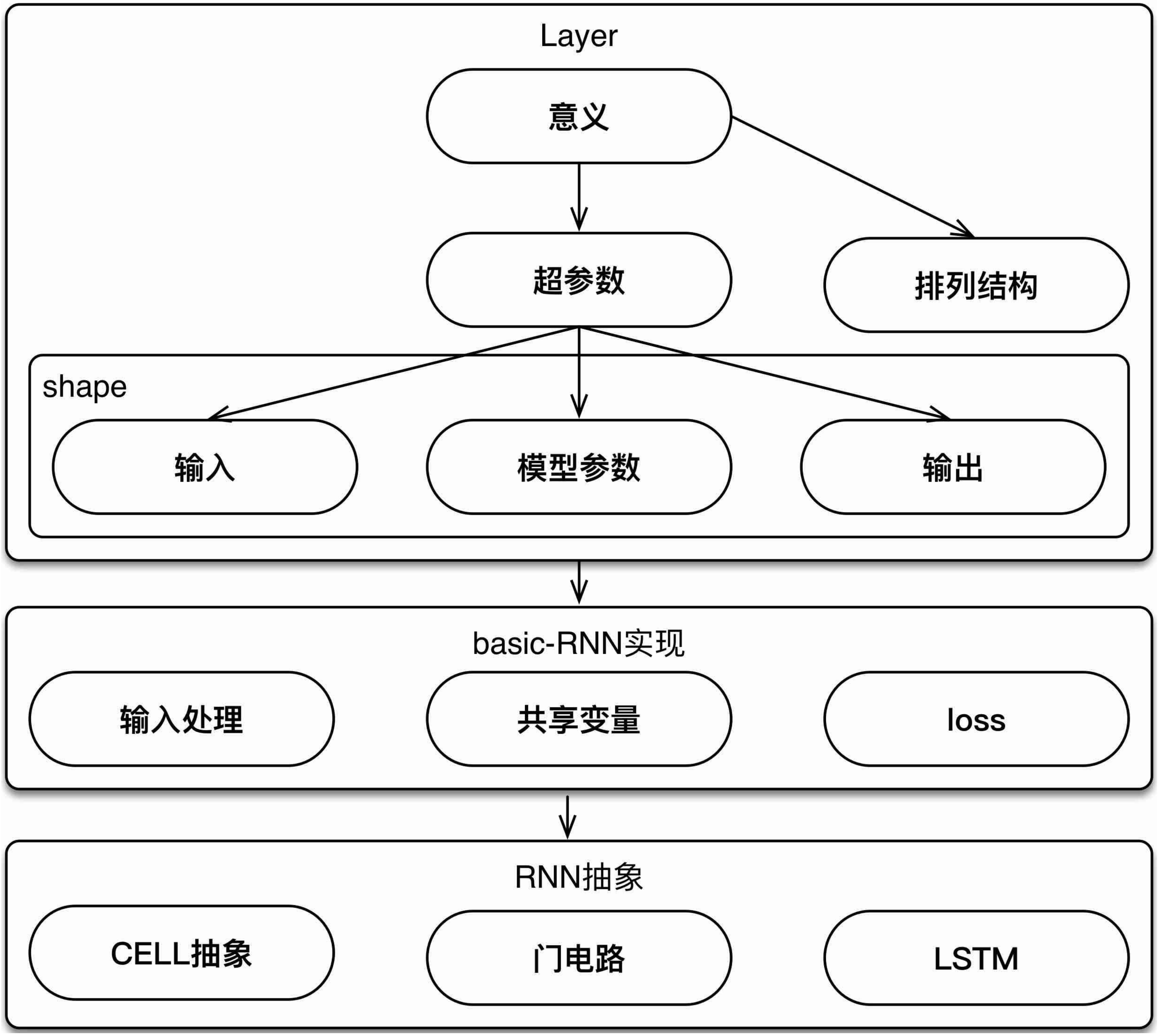
Python

https://tensorflow.google.cn/guide/keras/overview

TensorFlow学习路线： Dense与CNN

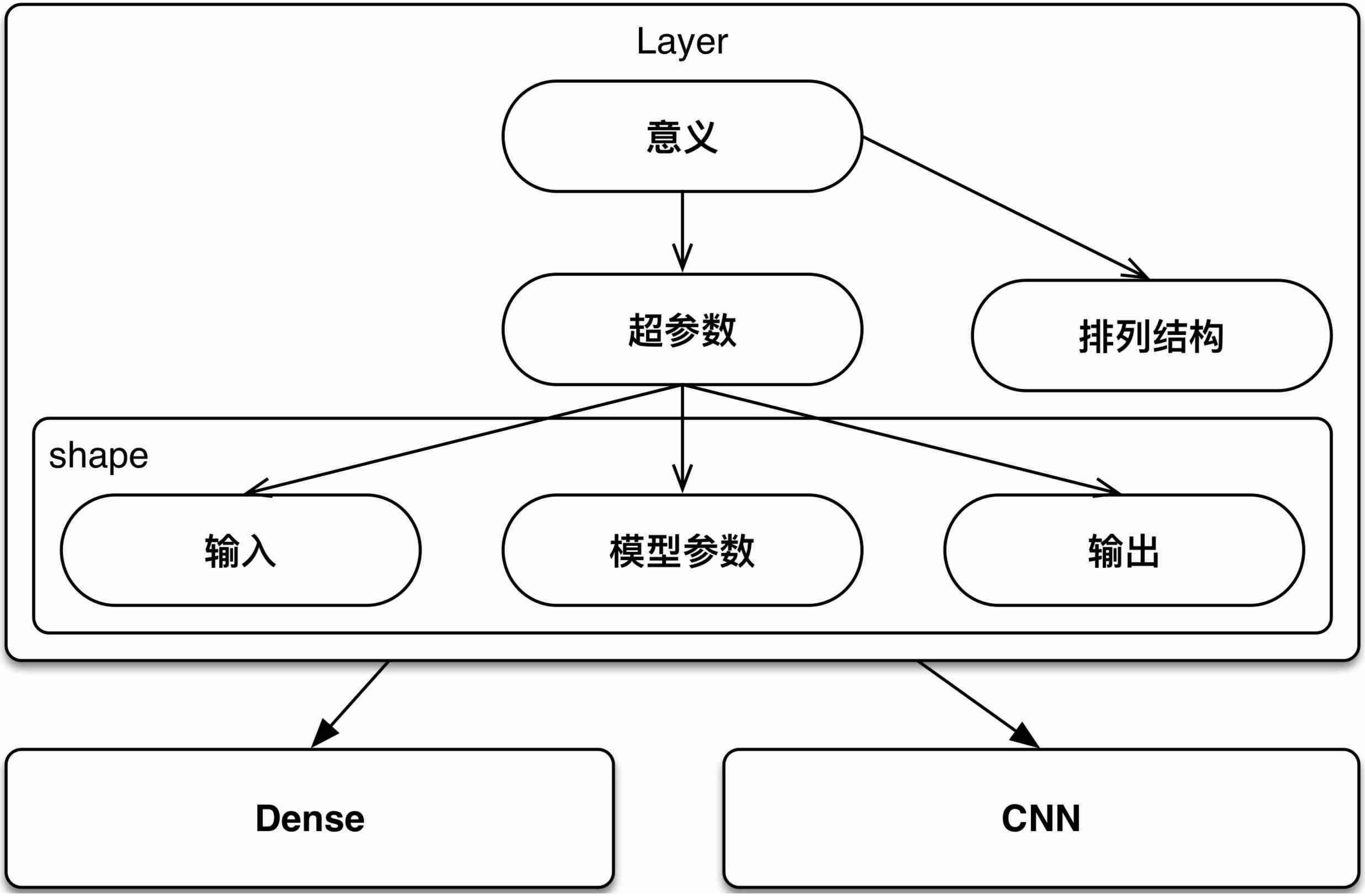


TensorFlow学习路线： RNN与LSTM



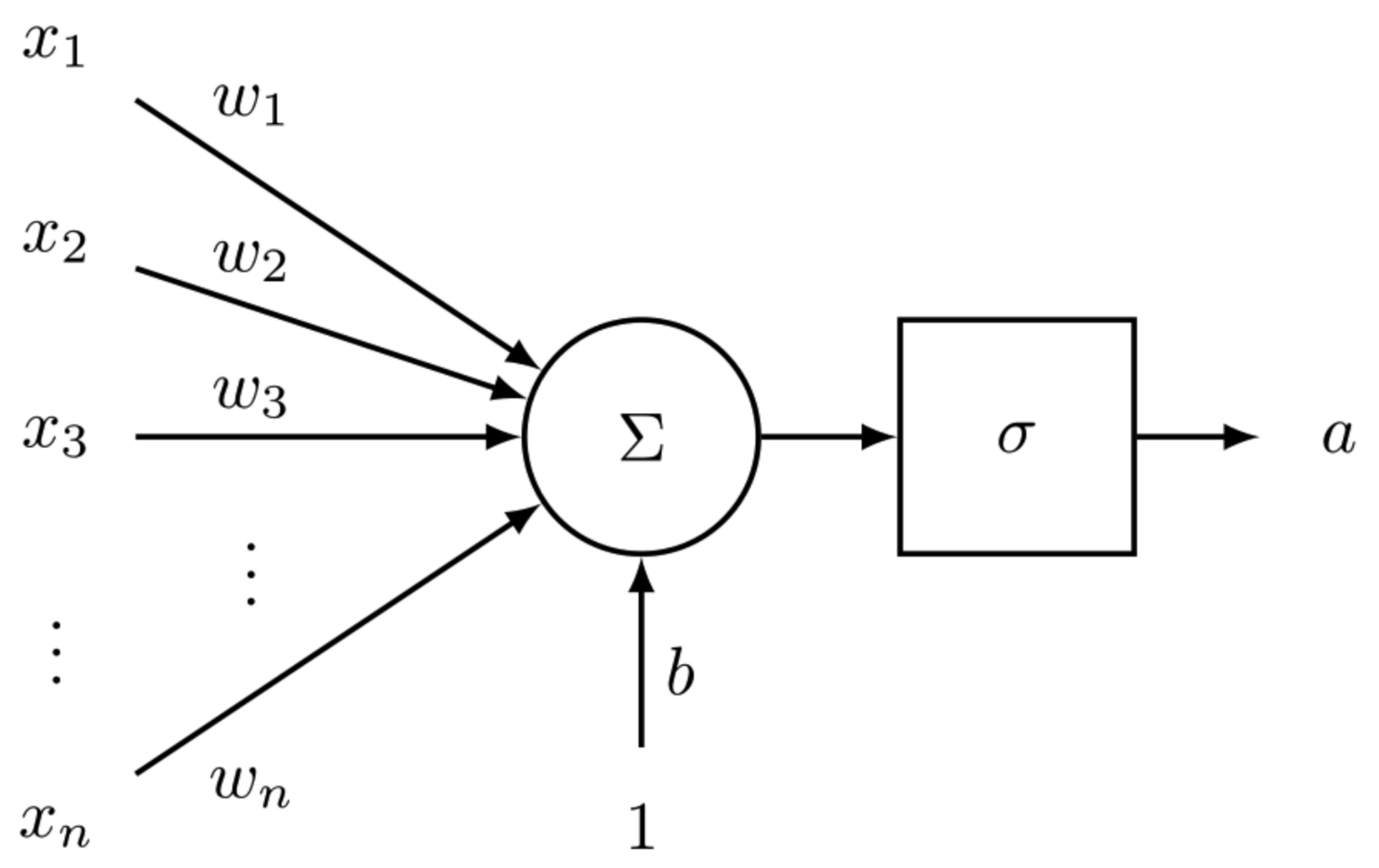
Dense

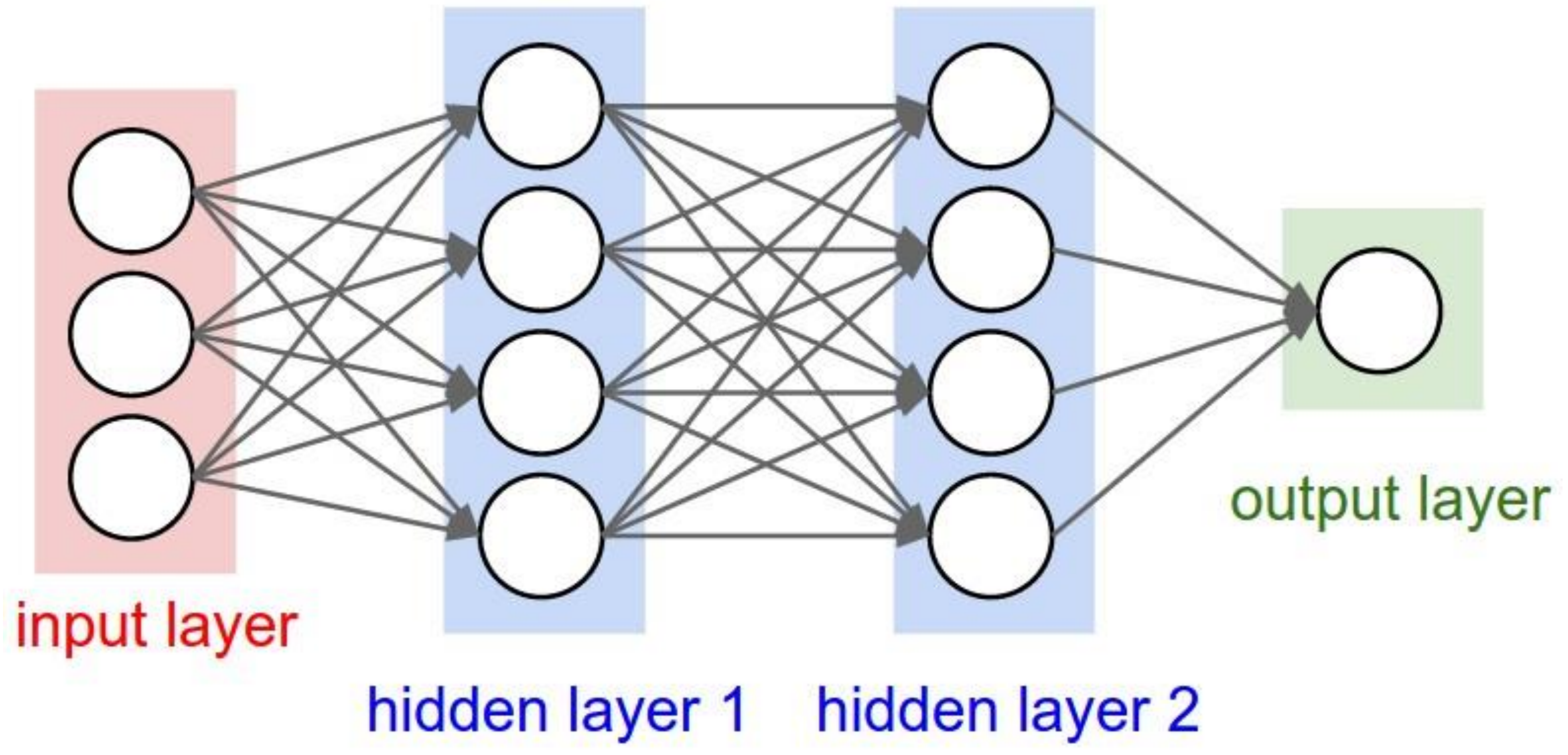
学习路线



Neuron → Layer

- Tensor表示：
- 单个人工神经元:
  - 输入是1d，参数是1d，输出是标量（0d）
- 一层人工神经元构成一个Layer
  - 输出的shape和Layer的shape一致.
- batch_size
  - 会影响输出的shape
  - 不影响参数的shape





Dense layer

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

```
tf.keras.layers.Dense(
    units, activation=None, use_bias=True, kernel_initializer='glorot_uniform',
    bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=None,
    activity_regularizer=None, kernel_constraint=None, bias_constraint=None,
    **kwargs
)
```

https://tensorflow.google.cn/api_docs/python/tf/keras/layers/Dense

Softmax层

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

- 输出的Softmax层处理，计算出一个概率分布向量：
- 所有输出的数值是正的，所有分量之和为 1。

```
tf.nn.softmax(
    logits, axis=None, name=None
)
```

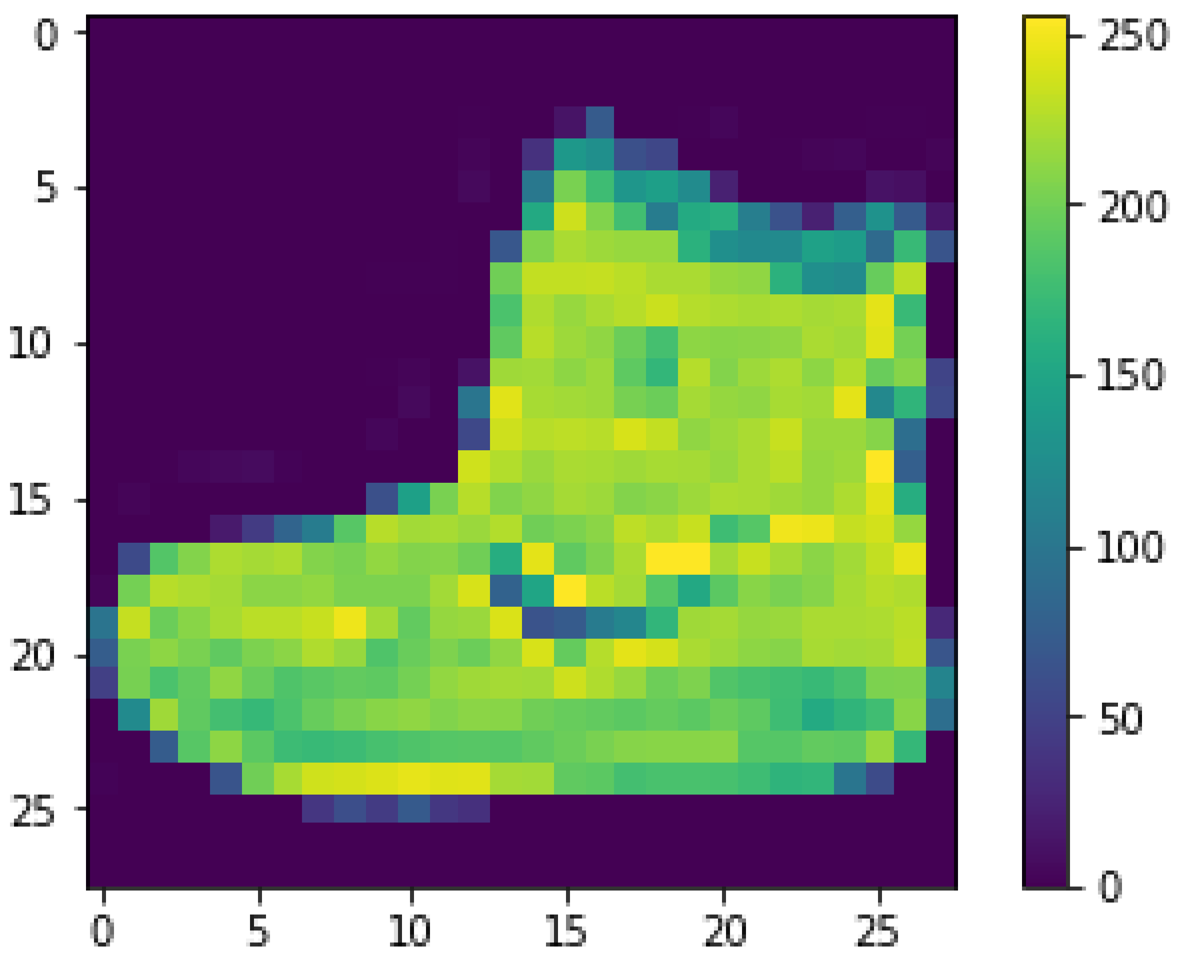
Args:
- `logits`: A non-empty `Tensor`. Must be one of the following types: `float32`, `float64`.
- `axis`: The dimension softmax would be performed on. The default is -1 which indicates the last dimension.
- `name`: A name for the operation (optional).

Returns:
A `Tensor`. Has the same type and shape as `logits`.

https://tensorflow.google.cn/api_docs/python/tf/keras/layers/Softmax

图像分类

- 图像的表示
  - 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



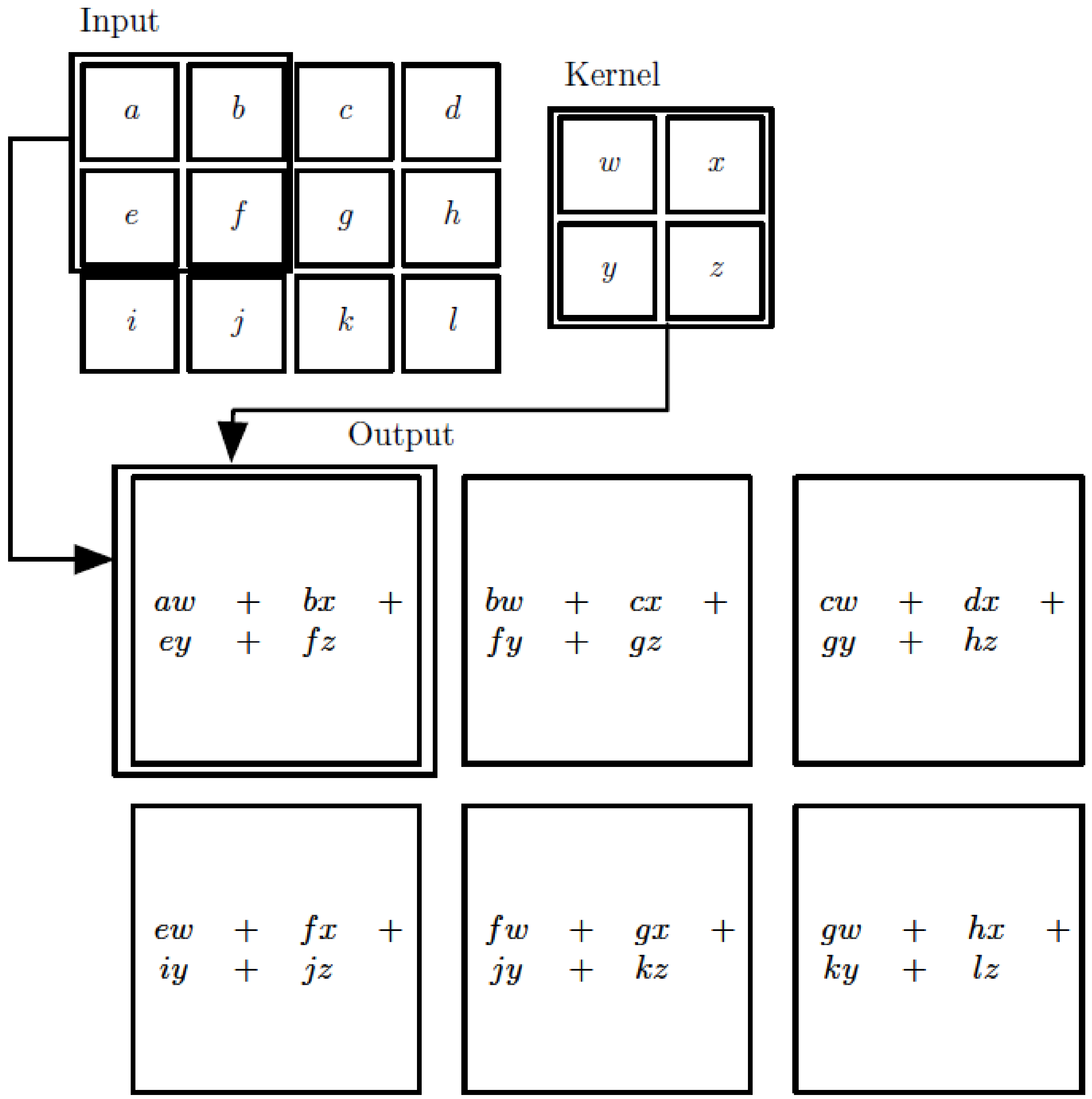
MLP实现图像分类

- 多层感知机MLP，由多层Dense组成，是一个常用的分类器（classifier）
- 多层感知机MLP进行图像分类（Image Classification），其中：
  - 彩色图像的每个像素点的颜色由RGB值表示
  - 一张200x200x3的图片，采用全连接Dense层
  - 单个神经元有200*200*3 = 120,000参数!（参数量太大！怎么办？）

CNN
convolutional neural networks

卷积运算（Convolution）

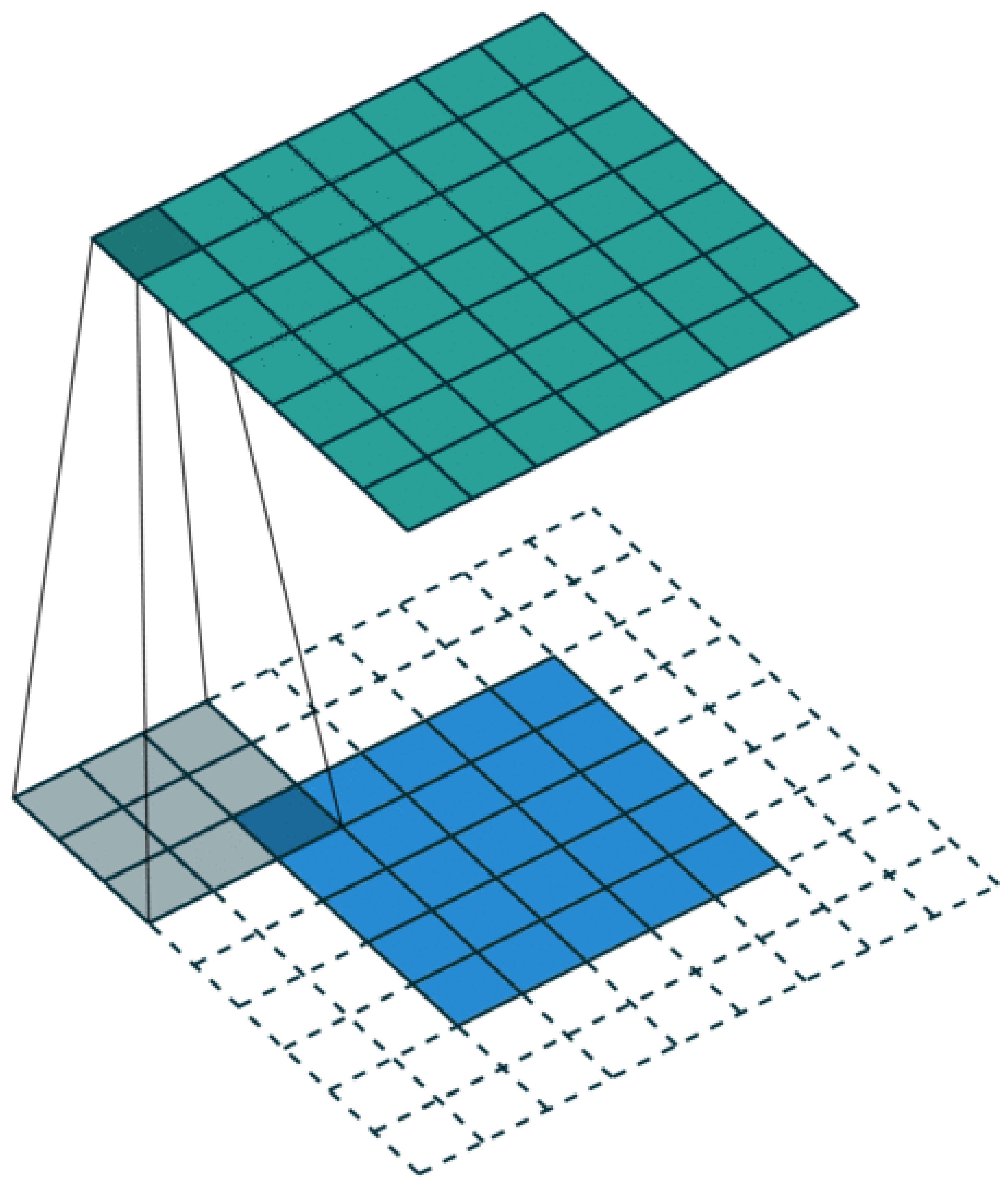
- 卷积运算是一种张量运算
  - 输入(Input)是多维数组(即张量Tensor)
  - 卷积核(Kernel)也是多维数组(即张量Tensor)
- 卷积核是由学习算法得到的权重参数
- 卷积核数目一般选16、32、64等



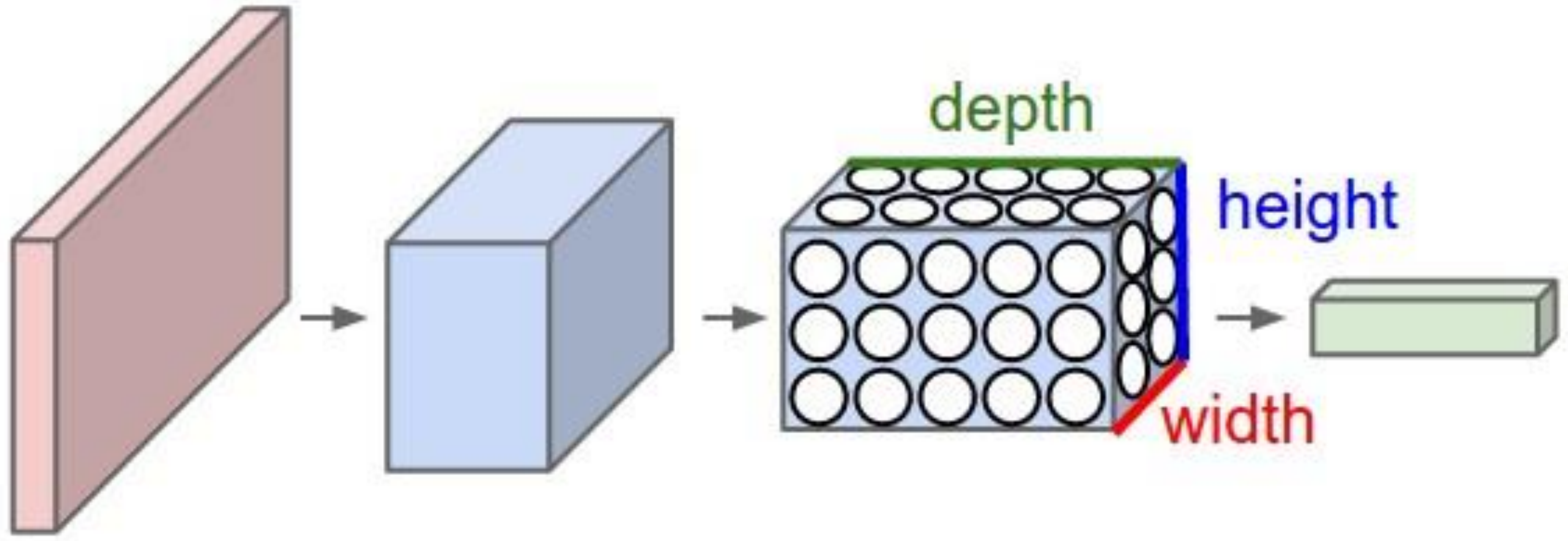
[Understanding Convolutions, http://colah.github.io/posts/2014-07-Understanding-Convolutions/](http://colah.github.io/posts/2014-07-Understanding-Convolutions/)

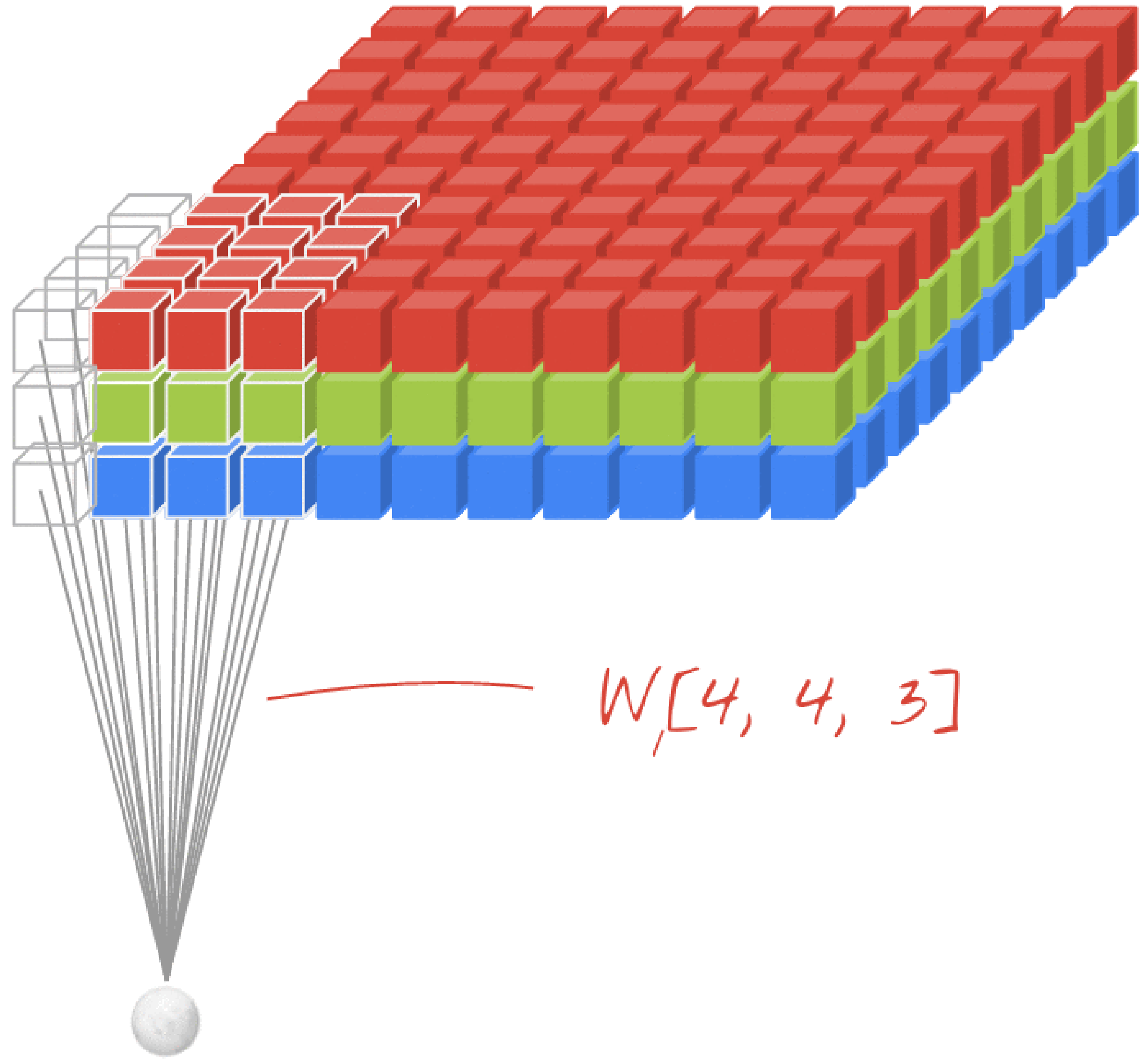
2d卷积核

- 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！



3d卷积核

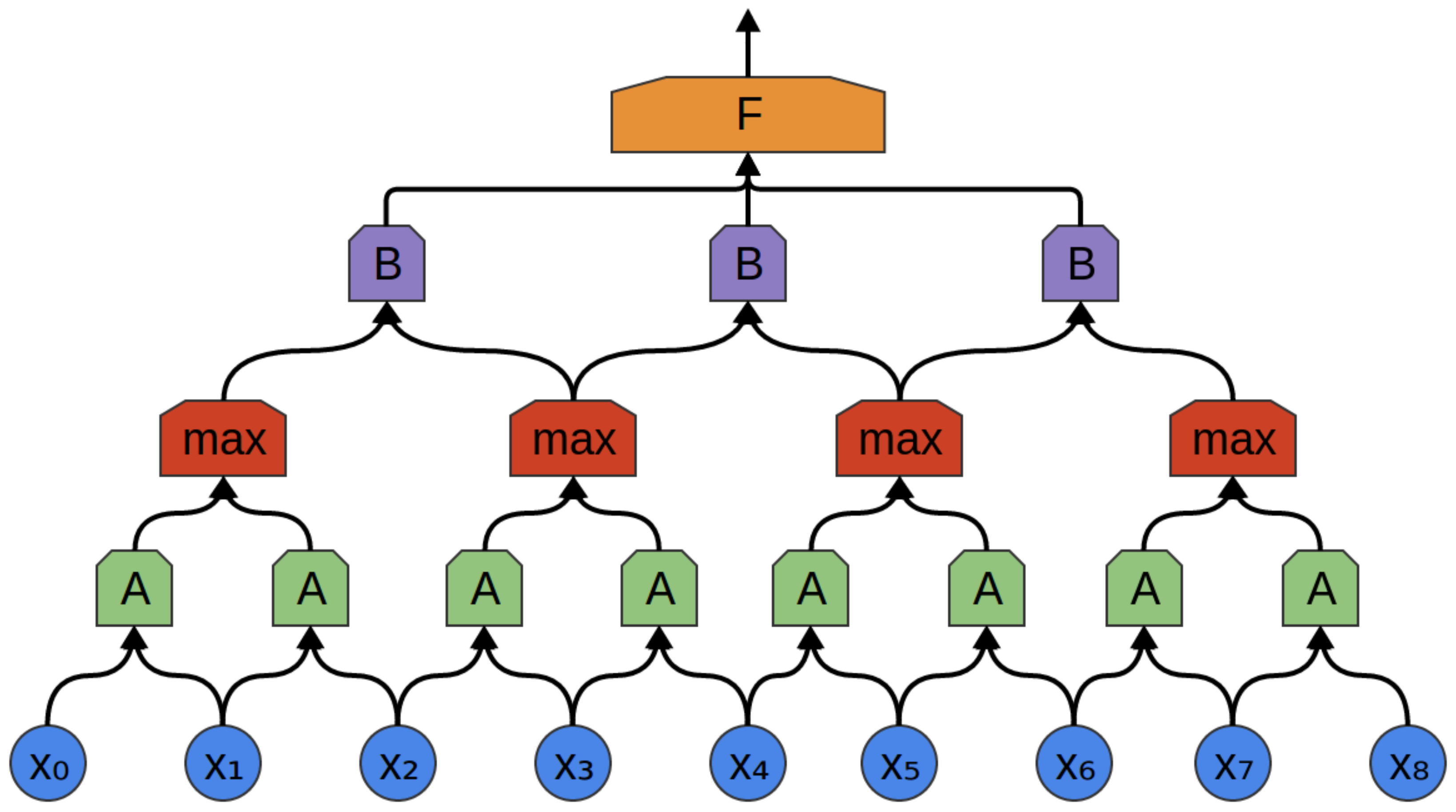


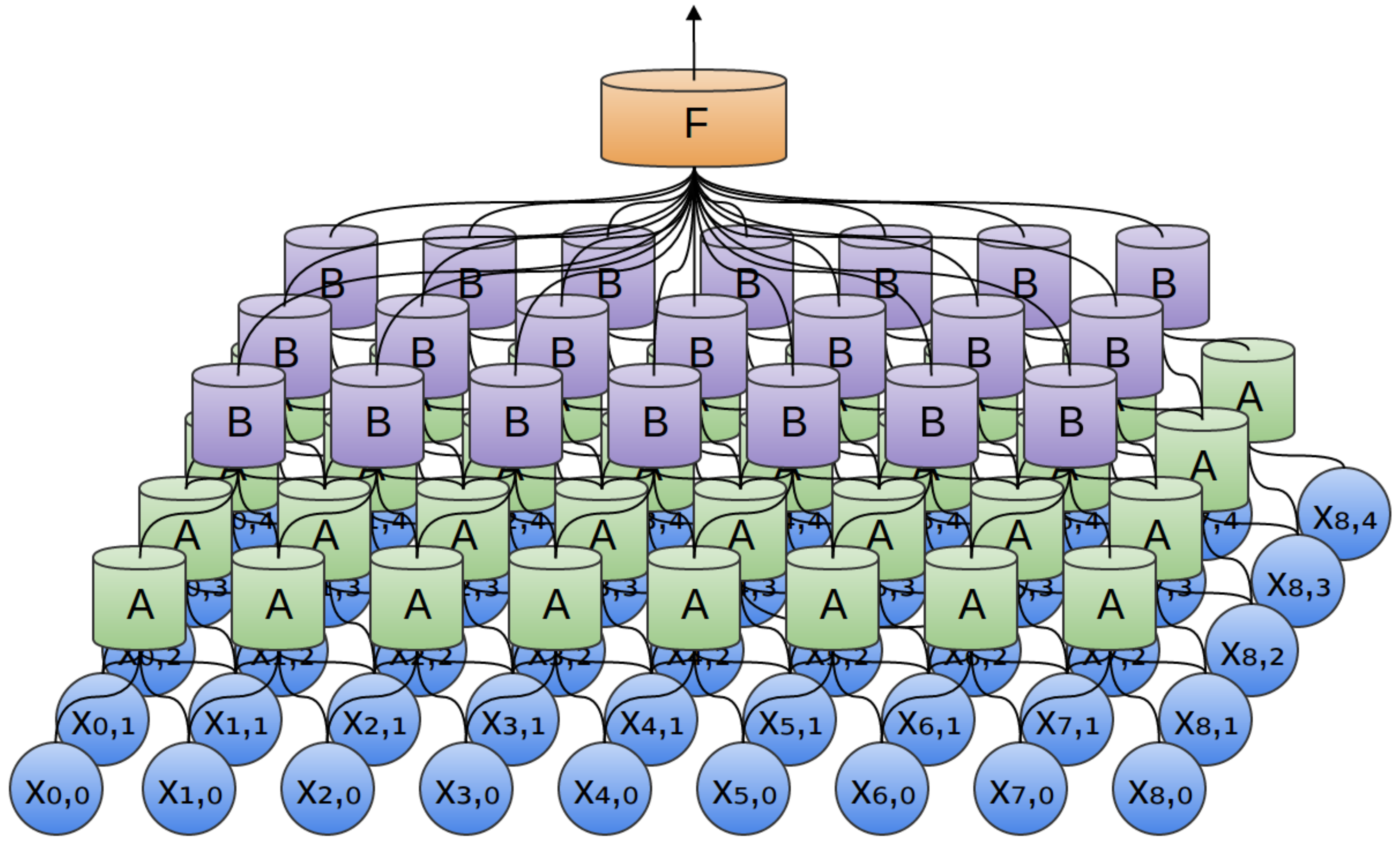


- 输入是3d的
- 有多个卷积核

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。





Conv Nets: A Modular Perspective, http://colah.github.io/posts/2014-07-Conv-Nets-Modular/

CNN layer

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

卷积层

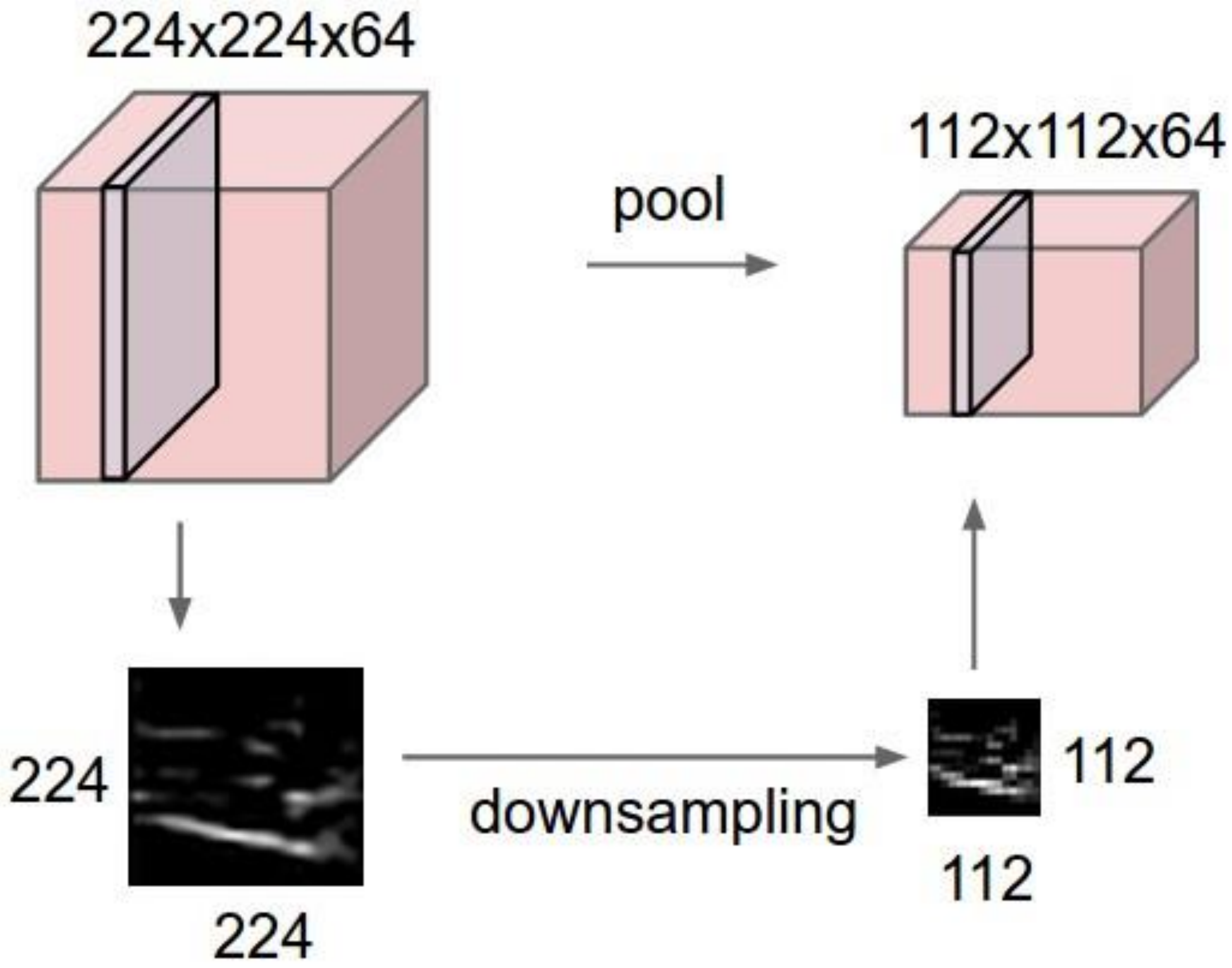
- 意义：用于处理图像.
- 排列结构：Layer的结构是3d
- 超参数：卷积核个数(D)，核大小(F)，padding(P)，strides(S)
- shape:
  - Input = W*W*3
  - L = (W-F+2P)/S+1
  - Layer = L*L*D
  - Weights = F*F*D
  - Output = L*L*D

```
tf.keras.layers.Conv2D(
    filters, kernel_size, strides=(1, 1), padding='valid', data_format=None,
    dilation_rate=(1, 1), activation=None, use_bias=True,
    kernel_initializer='glorot_uniform', bias_initializer='zeros',
    kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None,
    kernel_constraint=None, bias_constraint=None, **kwargs
)
```

https://tensorflow.google.cn/api_docs/python/tf/keras/layers/Conv2D

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



https://tensorflow.google.cn/api_docs/python/tf/keras/layers/MaxPool2D

Pool Layer

- AvgPool和MaxPool

```
tf.keras.layers.AveragePooling3D(
    pool_size=(2, 2, 2), strides=None, padding='valid', data_format=None, **kwargs
)
```

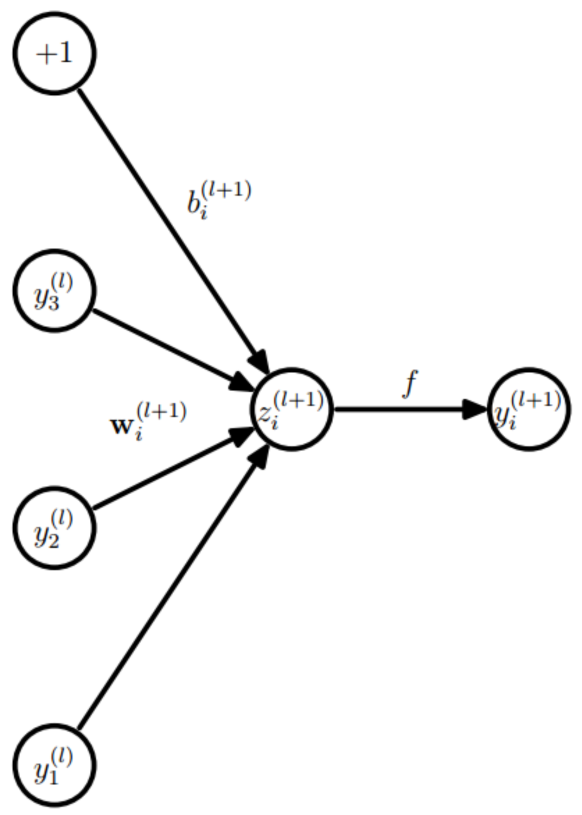
https://tensorflow.google.cn/api_docs/python/tf/keras/layers/AveragePooling3D

```
tf.keras.layers.MaxPool3D(
    pool_size=(2, 2, 2), strides=None, padding='valid', data_format=None, **kwargs
)
```

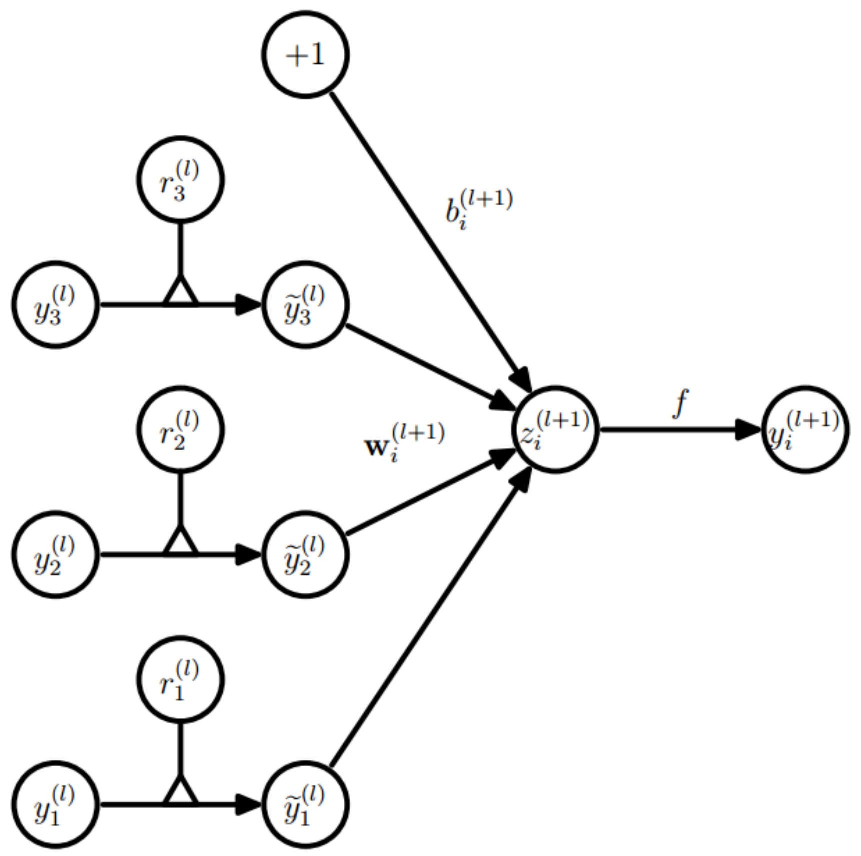
https://tensorflow.google.cn/api_docs/python/tf/keras/layers/MaxPool3D

Dropout层

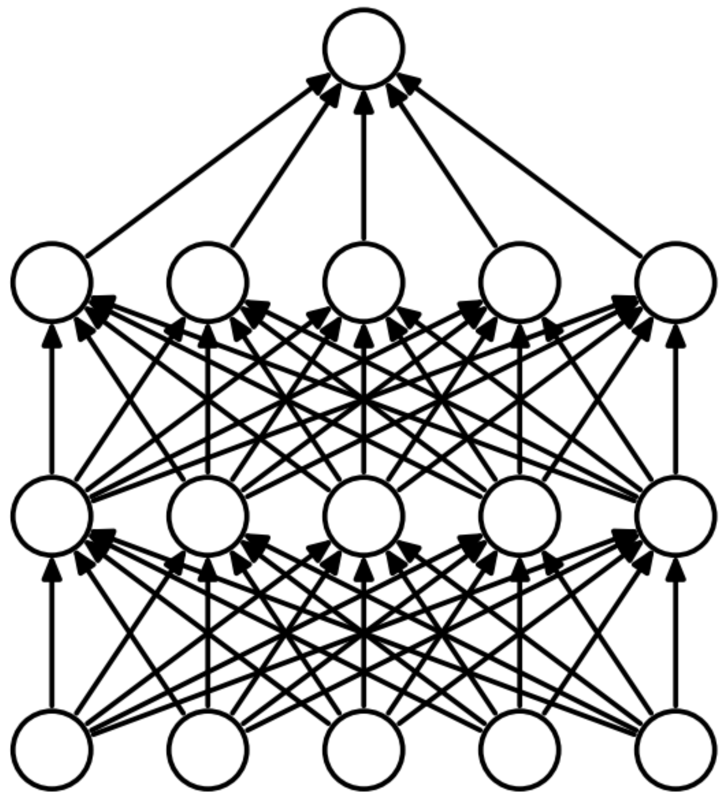
- 意义：减少CNN过拟合问题
- 超参数：keep_prob 丢弃率
- 对于所有的输入，有keep_prob概率保留并乘以1/keep_prob，以保证前后总和大致相等，否则输出0



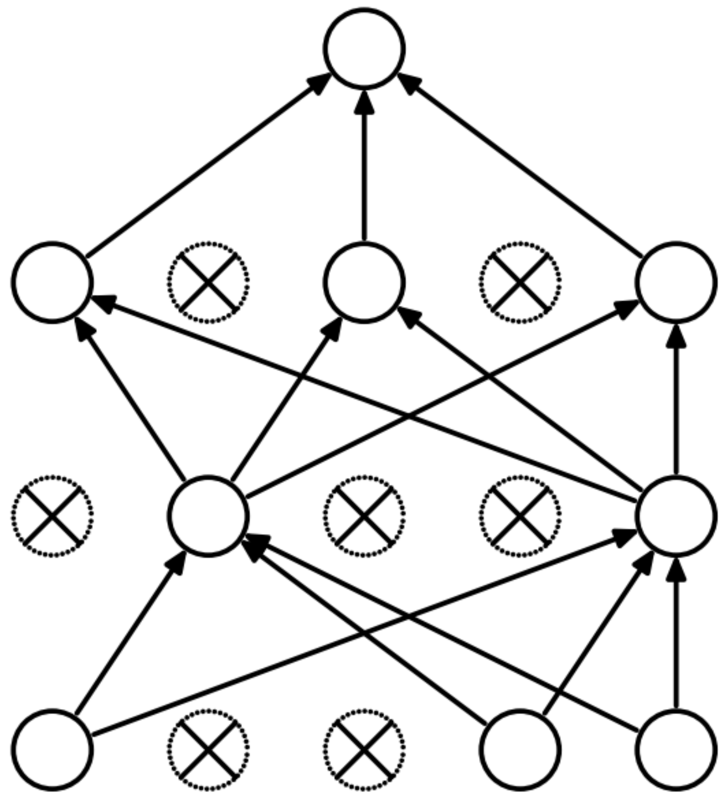
(a) Standard network



(b) Dropout network



(a) Standard Neural Net



(b) After applying dropout.

https://tensorflow.google.cn/api_docs/python/tf/keras/layers/Dropout

Dropout Layer

- Apply Dropout to input to prevent overfitting.

```
tf.keras.layers.Dropout(
    rate, noise_shape=None, seed=None, **kwargs
)
```

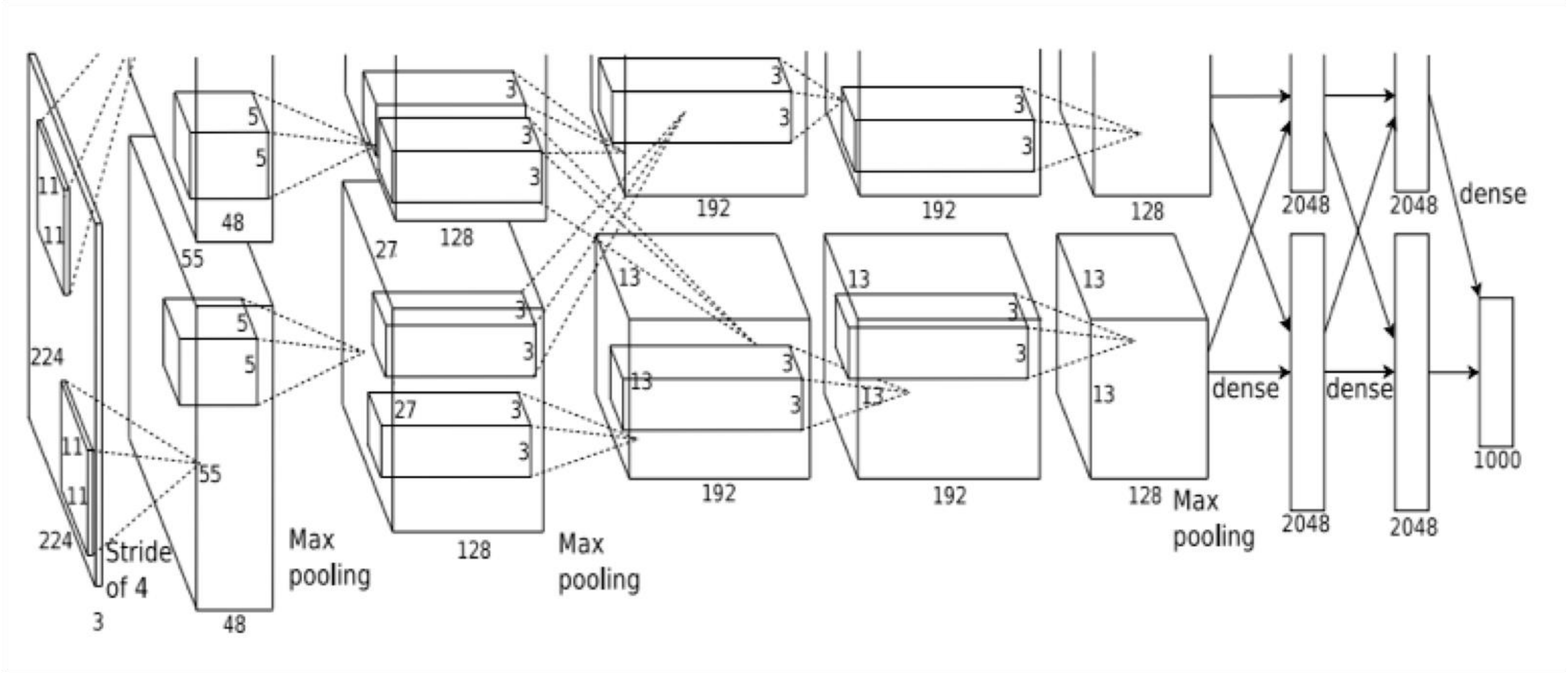
https://tensorflow.google.cn/api_docs/python/tf/keras/layers/Dropout

CNN示例

- CNN处理Fashion MNIST/MNIST的例子；

https://tensorflow.google.cn/tutorials/keras/classification

课后阅读作业1

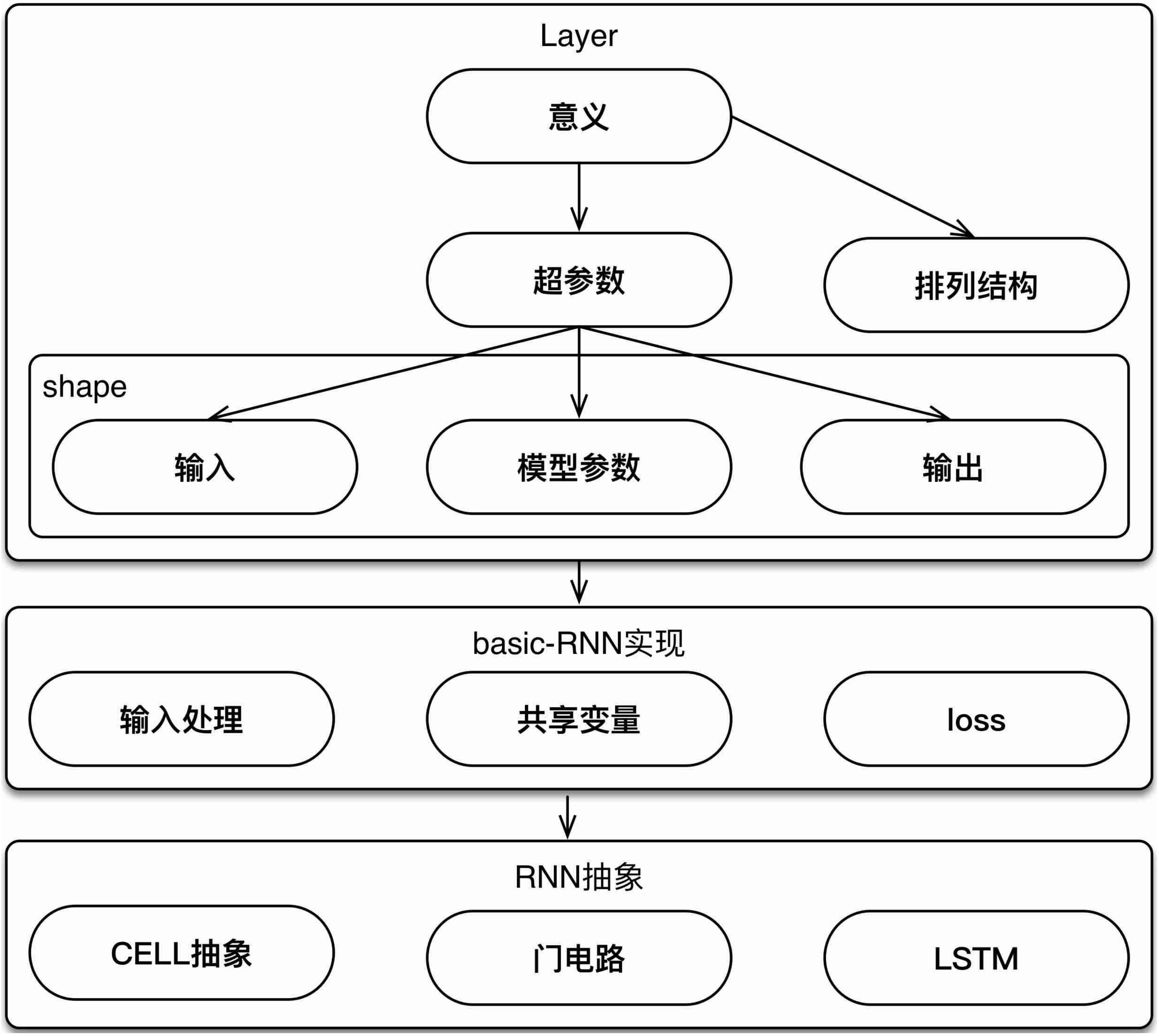


[1] Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton. ″ImageNet classification with deep convolutional neural networks.″ NIPS 2012.

RNN

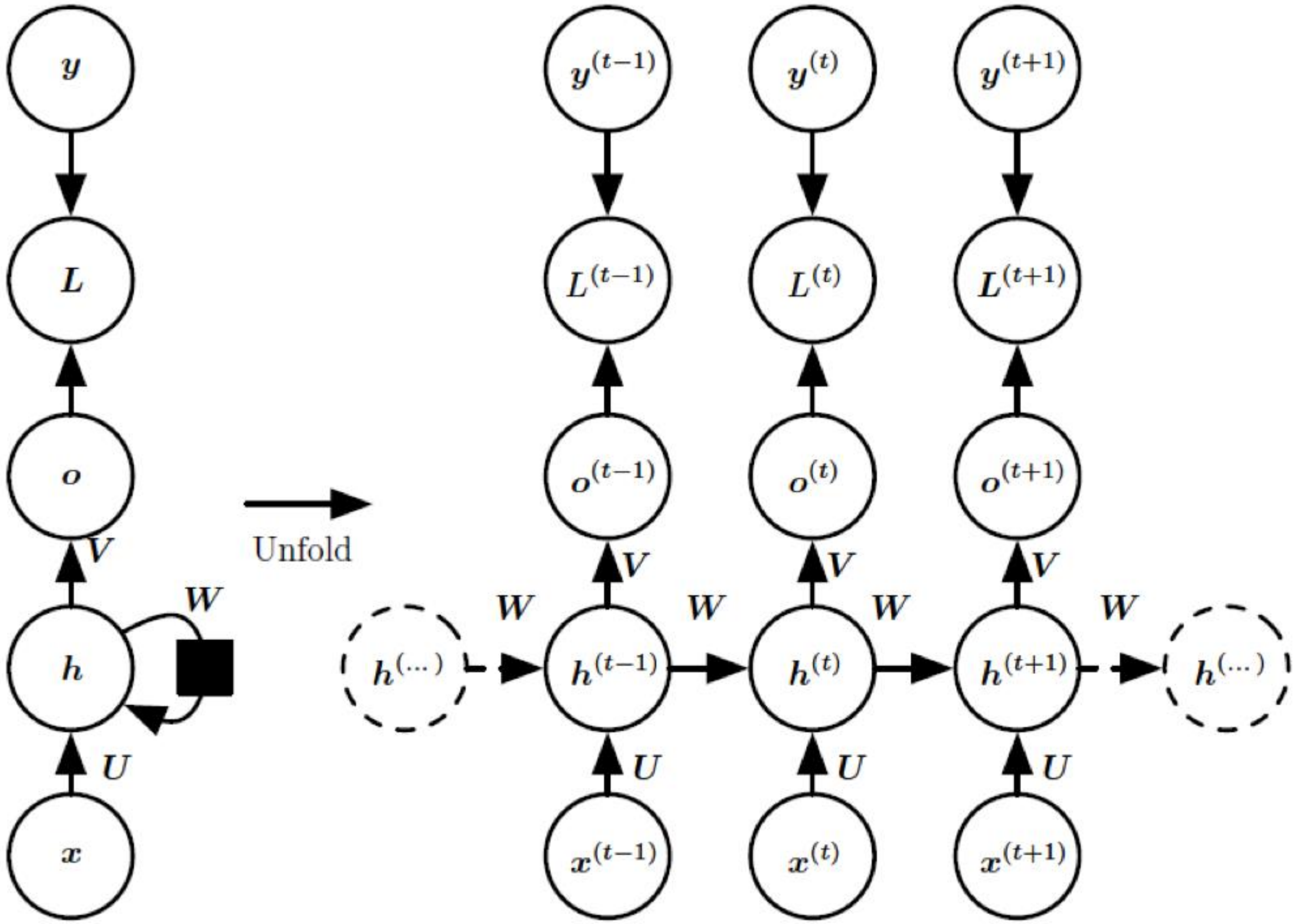
recurrent neural networks

学习路线


Layer
意义
超参数
排列结构
shape
输入
模型参数
输出
basic-RNN实现
输入处理
共享变量
loss
RNN抽象
CELL抽象
门电路
LSTM

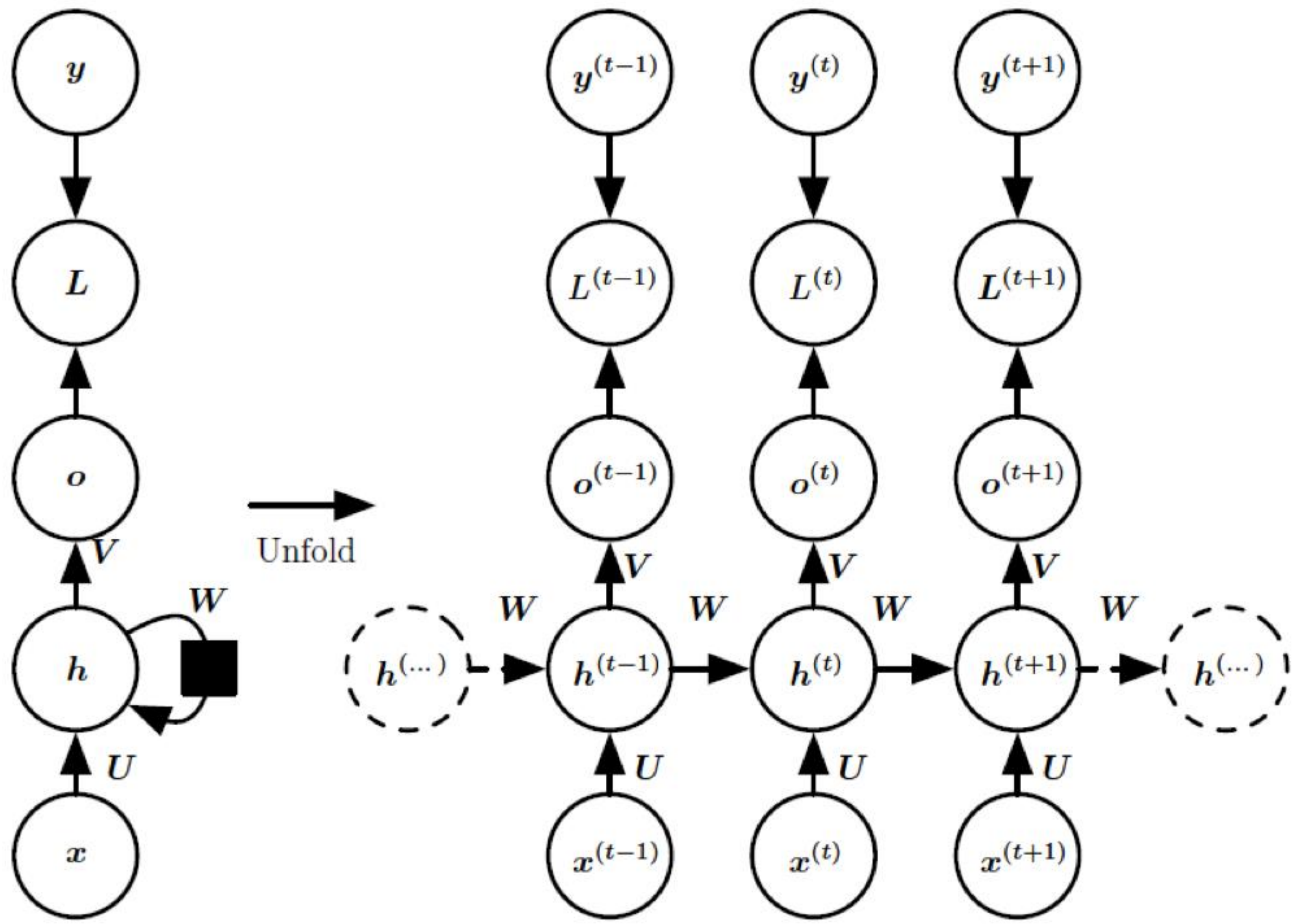
RNN

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



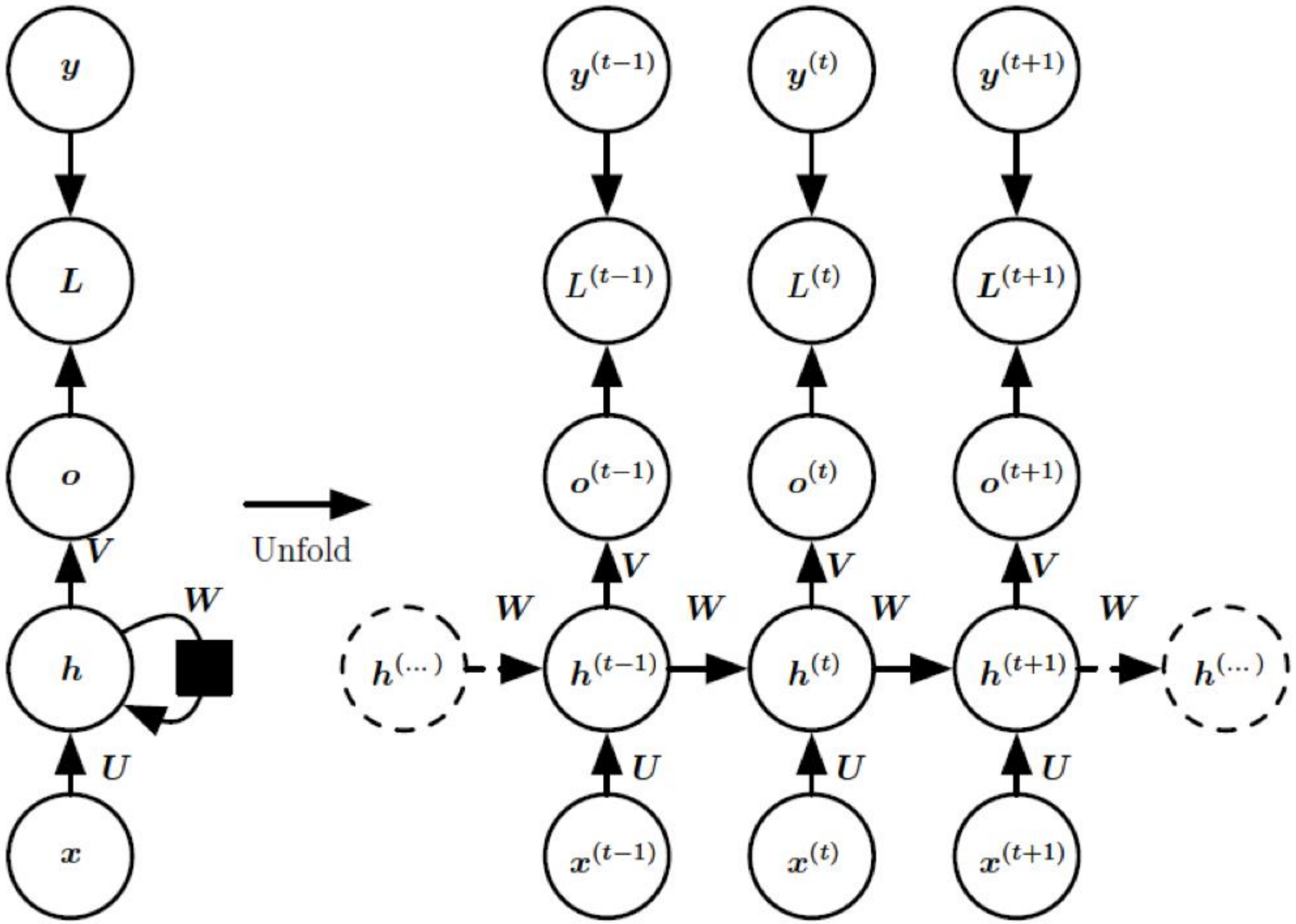
循环网络-权重共享

- 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
  - W：隐藏到隐藏的循环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



计算图

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



- 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的权重矩阵。
- b 和c 是偏置向量。
- 循环网络将一个输入序列映射到相同长度的输出序列。

输入和loss处理

- 给定序列长度(模型超参数)，把输入序列化
- 输入进行离散化处理（ one-hot ）
- 时间步上权重矩阵U, W, V 权重共享
- 收集所有时刻的输出,计算的loss
  - 与x序列配对的y的总损失就是所有时间步的损失之和
  - 损失是给定$x_1, \cdots, x_t$后y_t负对数似然

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

basic-rnn 实现

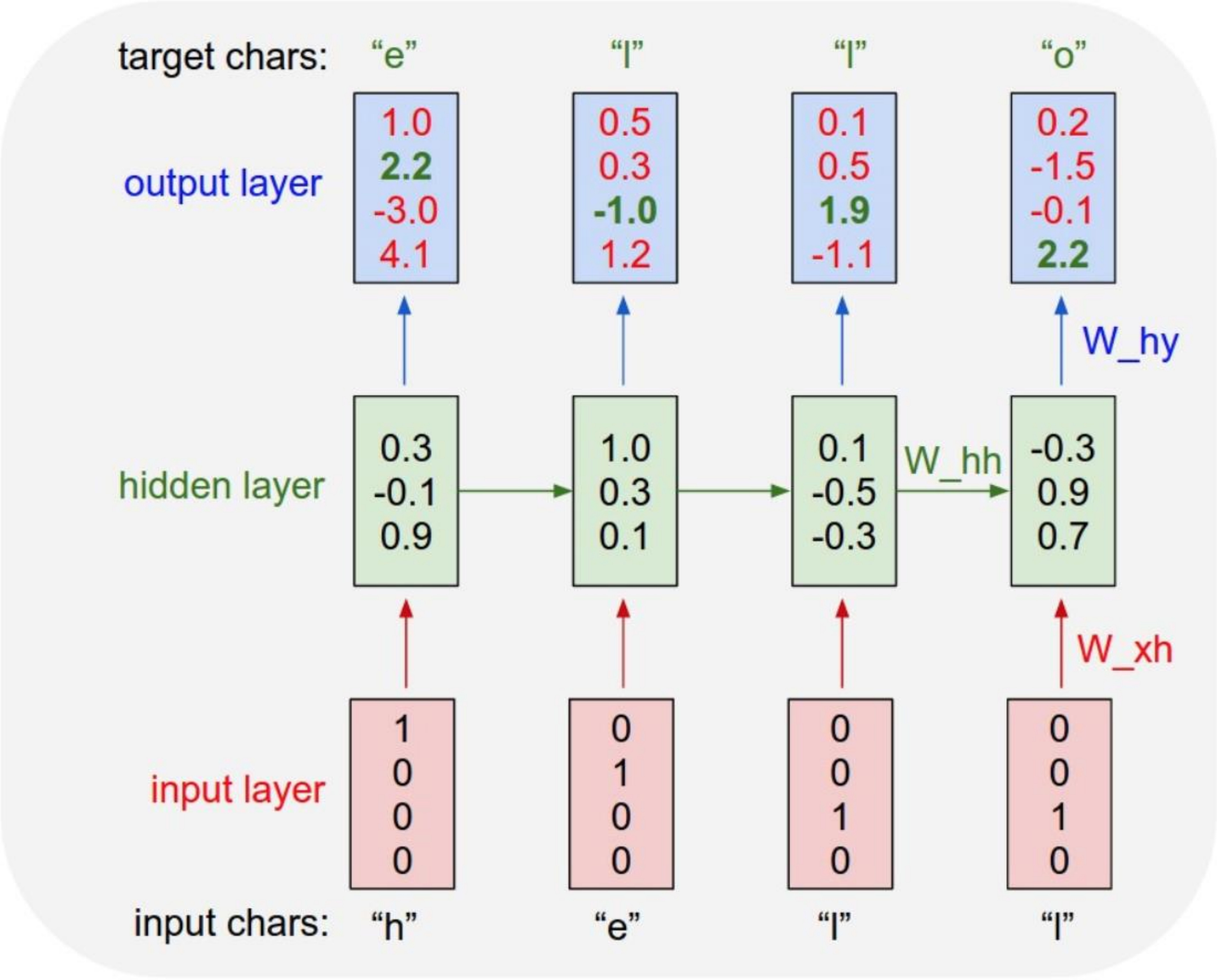
```
iter: 646500, p: 1300, loss: 0.019042
----
 y = tf.get_variable("by", [vocab_size], initializer=initializer)
        by  = tf.get_variable("by", [vocab_size], dtype=tf.float32, name="state:
 hprev_val = np.zeros([1, hidden_size])

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

min-char-rnn-tensorflow.py

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



rnn-cell抽象

- hidden-units : 模型的容量大小
- I(input) + S(state) -> O(output) + S(new_state)
- inputs: 输入
- Outputs:当前的输出完全取决于state和当前的输入
- state: 隐含了之前所有的输出信息

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.

```
tf.keras.layers.SimpleRNN(
    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
)
```

https://tensorflow.google.cn/api_docs/python/tf/keras/layers/SimpleRNN

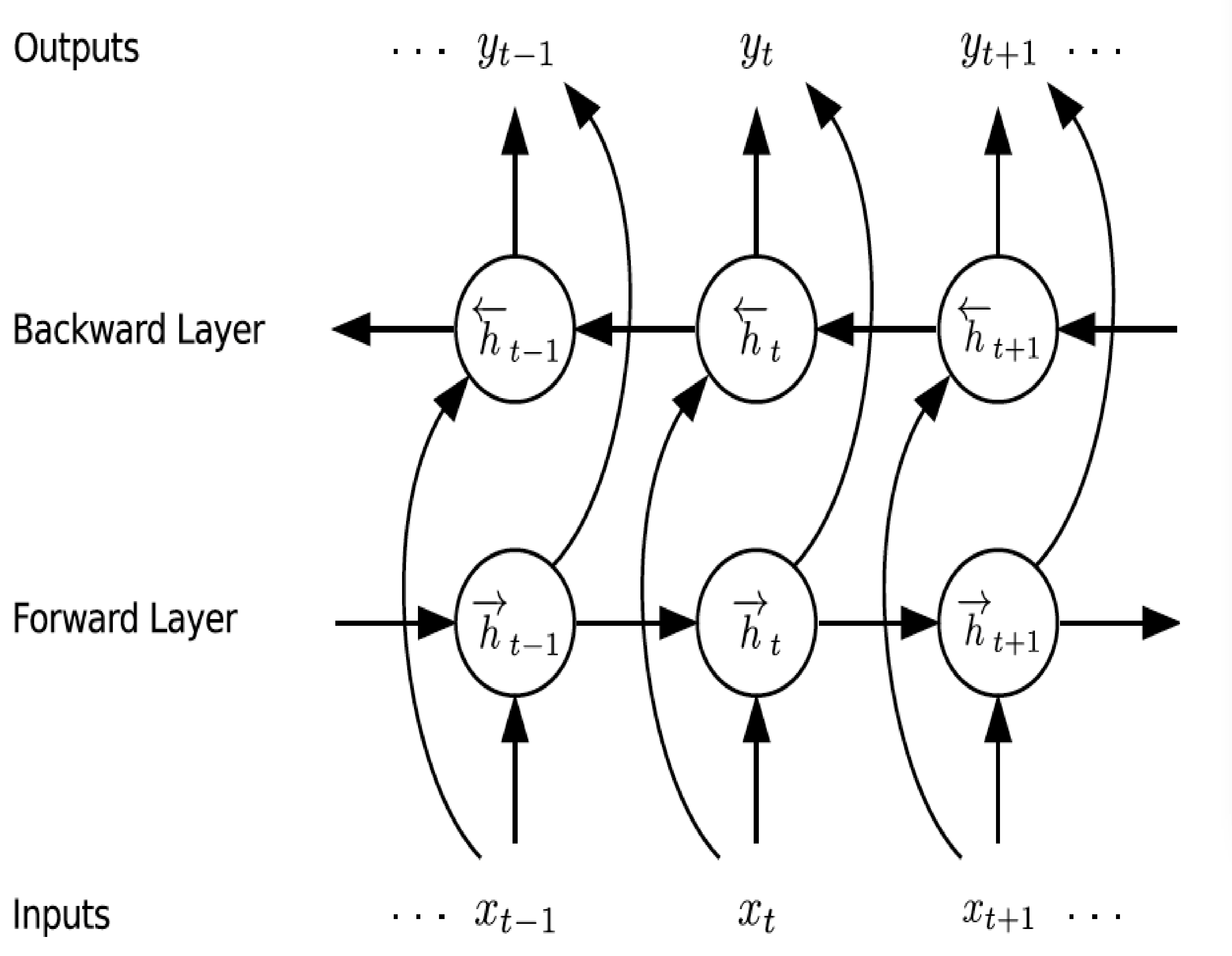
keras. layers. RNN

- Class RNN
- Fully-connected RNN where the output is to be fed back to input.
- units: Positive integer, dimensionality of the output space.

```
tf.keras.layers.RNN(
    cell, return_sequences=False, return_state=False,
    go_backwards=False, stateful=False, unroll=False,
    time_major=False, **kwargs
)
```

https://tensorflow.google.cn/api_docs/python/tf/keras/layers/RNN

课后阅读作业2



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

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

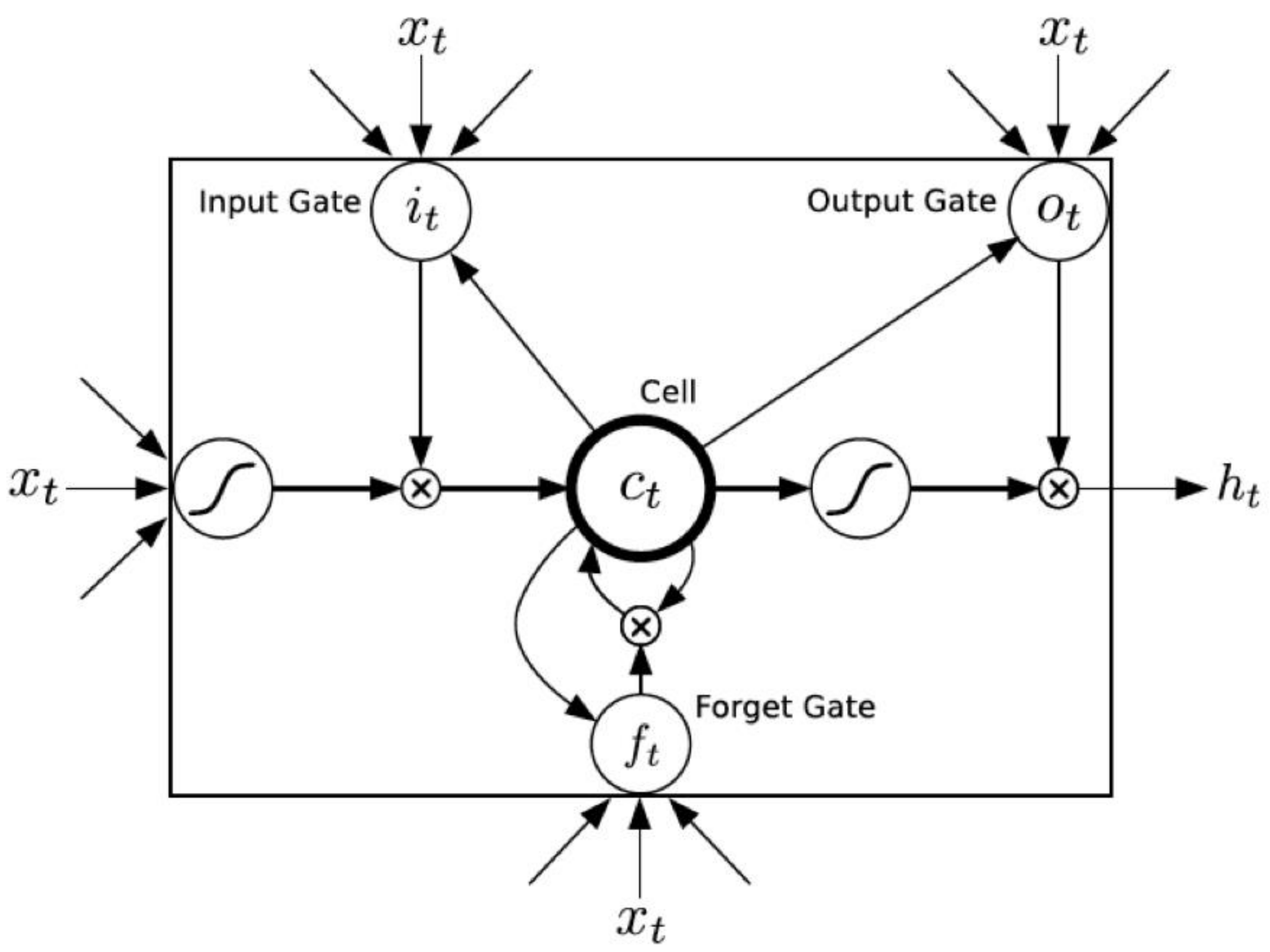
$$y_t = W_{\overrightarrow{h}y} \overrightarrow{h}_t + W_{\overleftarrow{h}y} \overleftarrow{h}_t + b_y$$

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

LSTM
Long Short-Term Memory

RNN→LSTM

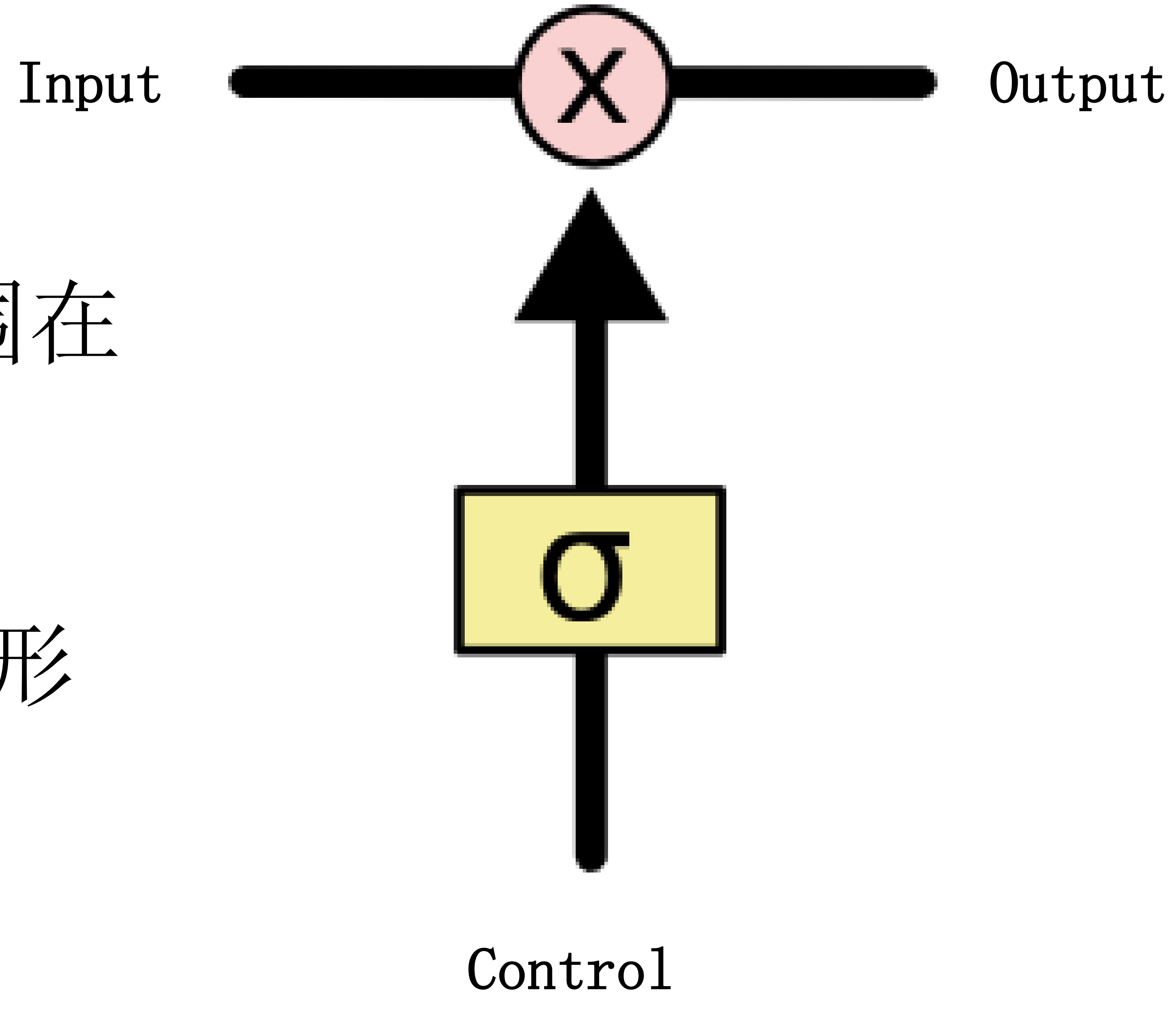
- RNN训练有以下问题
  - RNN梯度爆炸
  - RNN梯度消失
- LSTM解决以上问题



https://distill.pub/2019/memorization-in-rnns/

门单元Gate

- 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σ（Control）元素相乘等到一个同形的Output

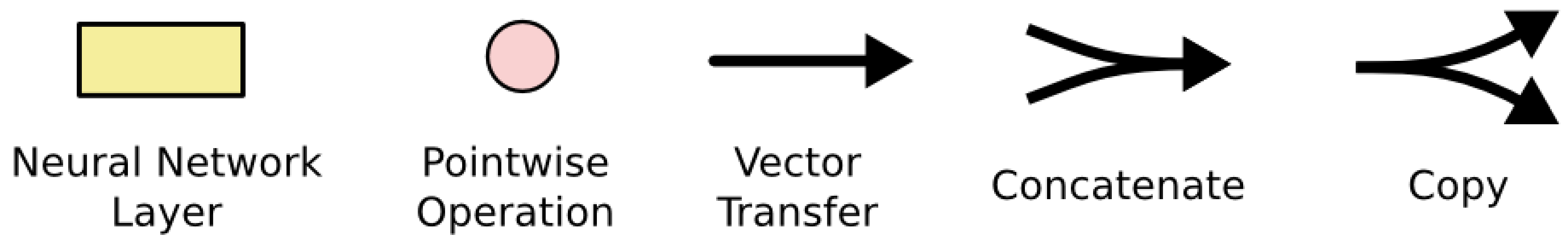


LSTM

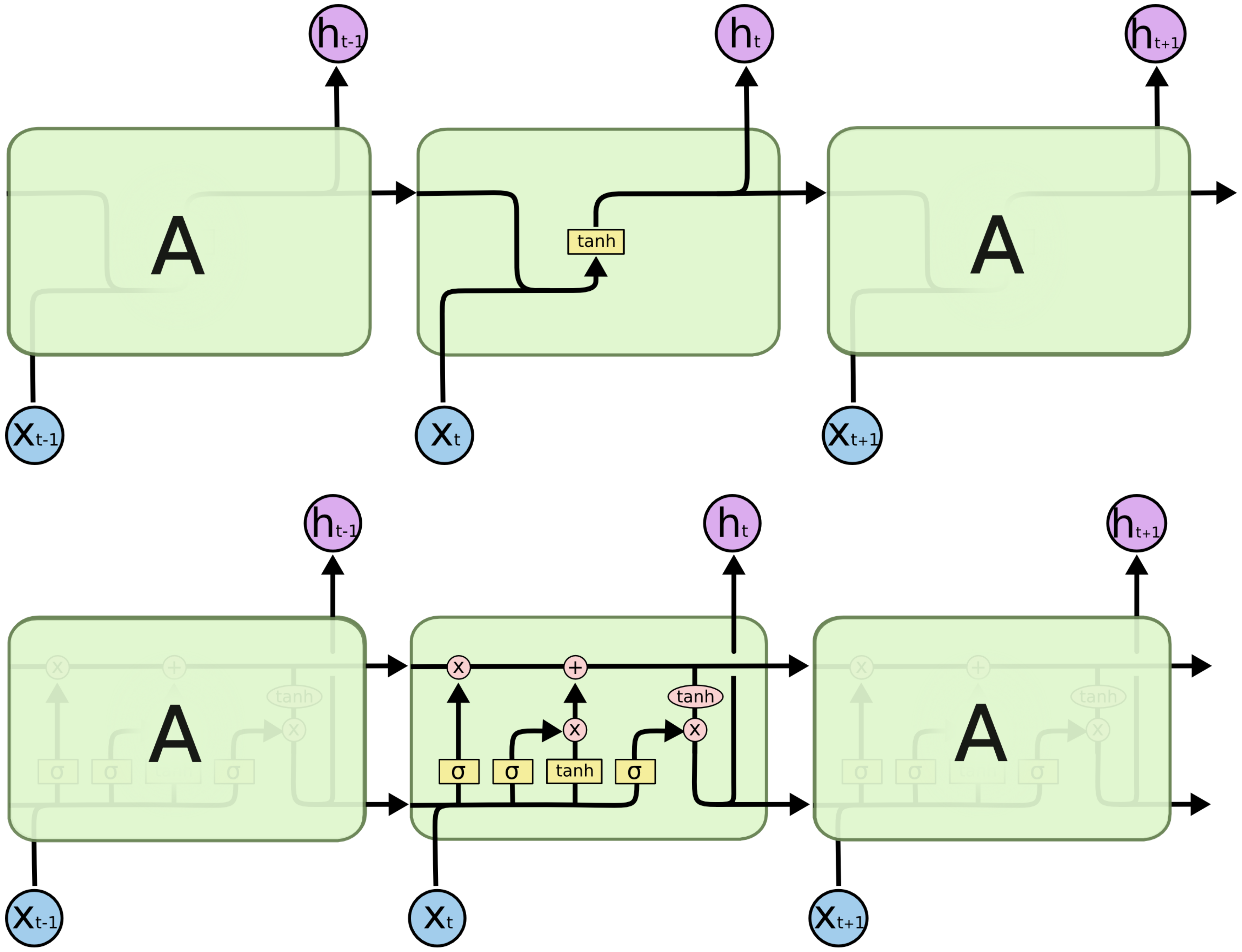
- LSTM增加了一个**辅助记忆单元**和三个辅助的**门单元**，对vanilla RNN进行改进。
  - 输入门（Input gate）控制是否输入，
  - 遗忘门（Forget gate）控制是否存储，
  - 输出门（Output gate）控制是否输出。
- 辅助记忆单元可以寄存时间序列的输入，在训练过程中会利用后向传播的方式进行。
- 记忆单元和门单元的组合，提升了RNN处理远距离依赖问题的能力 ，解决RNN网络收敛慢的问题。

Hochreiter, S , and J. Schmidhuber . "Long Short-Term Memory." *Neural computation* 9.8(1997):1735-1780.

RNN与LSTM

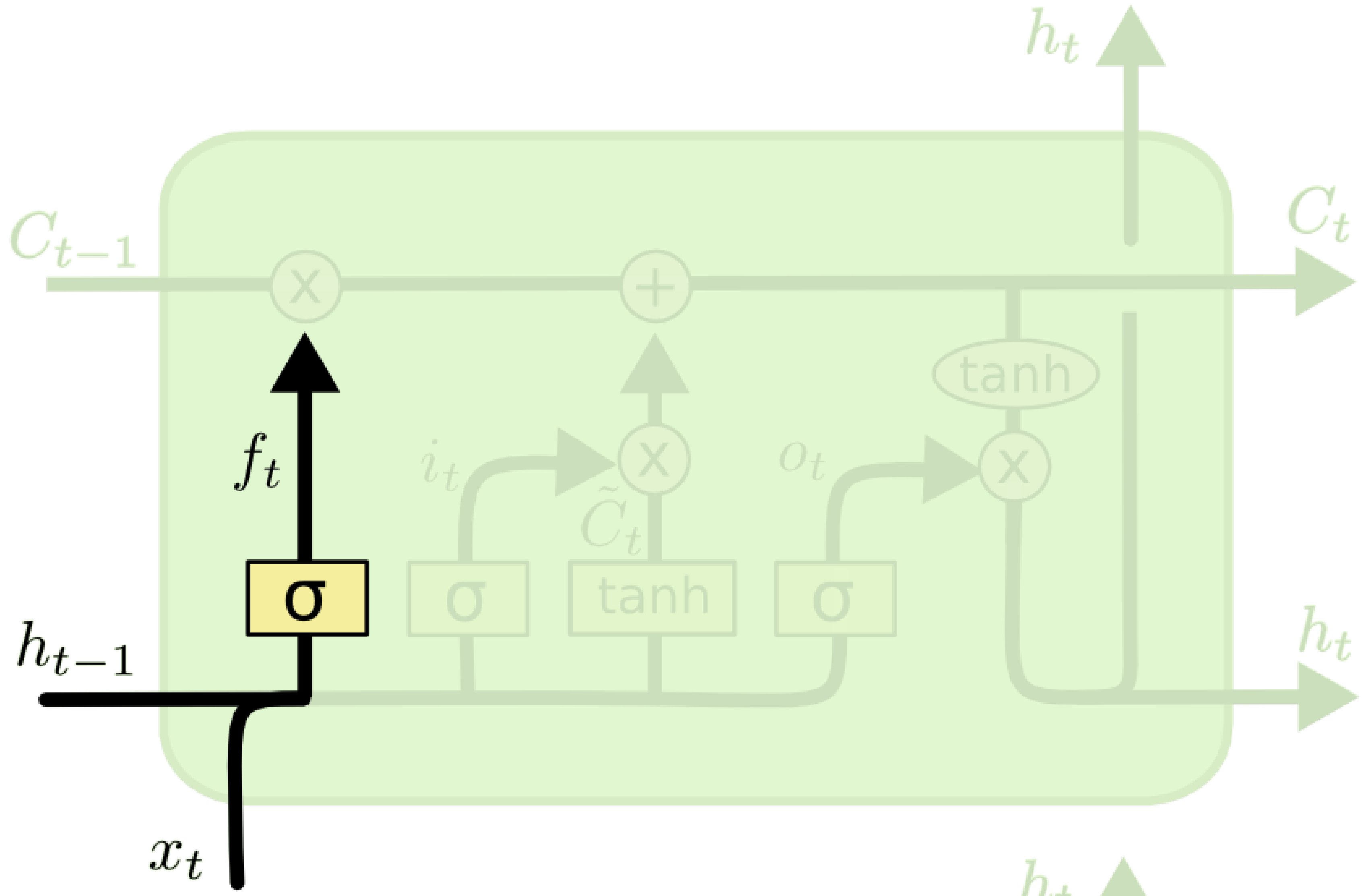


- 记忆单元C
- 遗忘门f
- 输入门i
- 输出门o

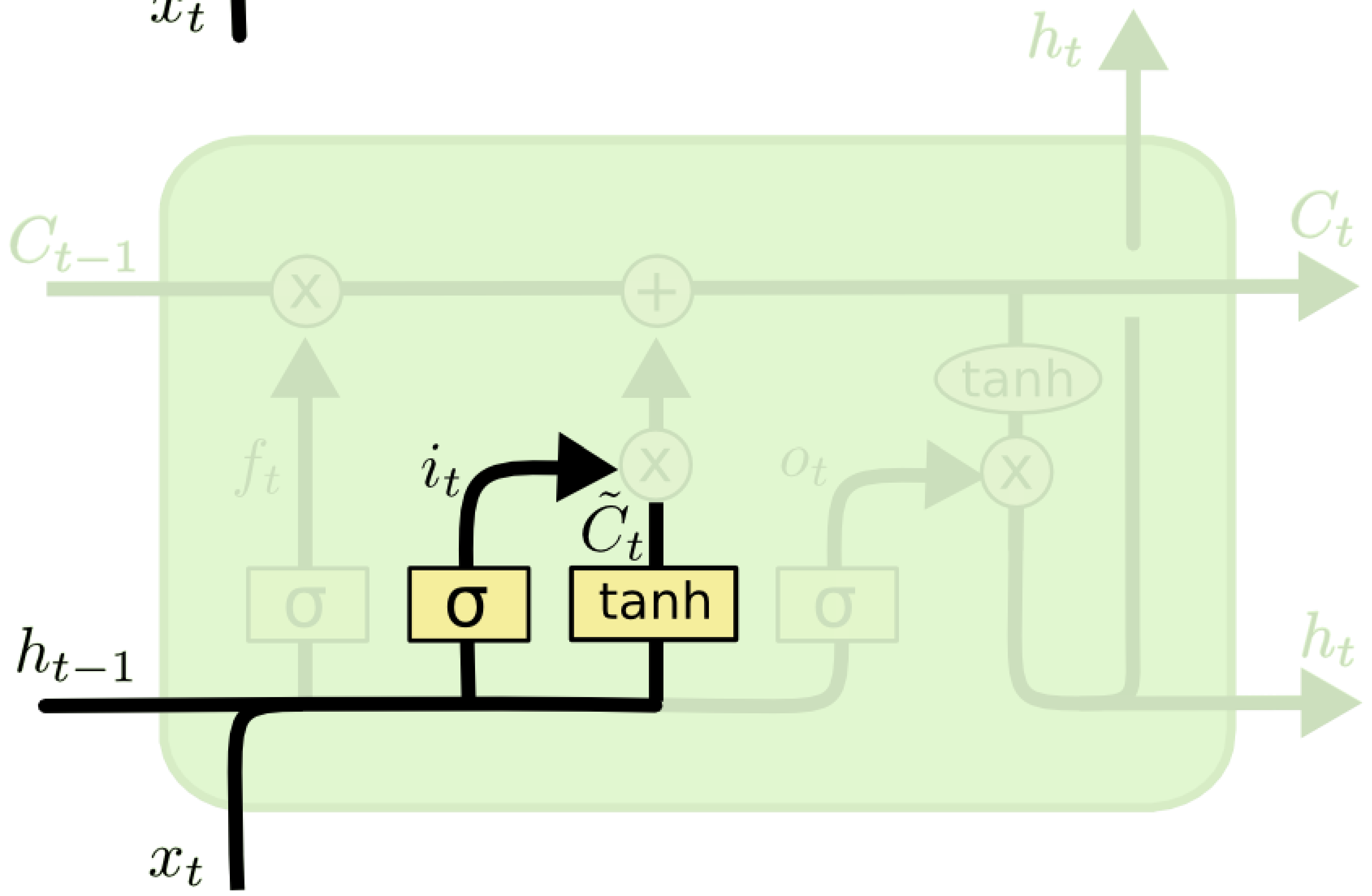


[Understanding-LSTMs, http://colah.github.io/posts/2015-08-Understanding-LSTMs/](http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTM-1



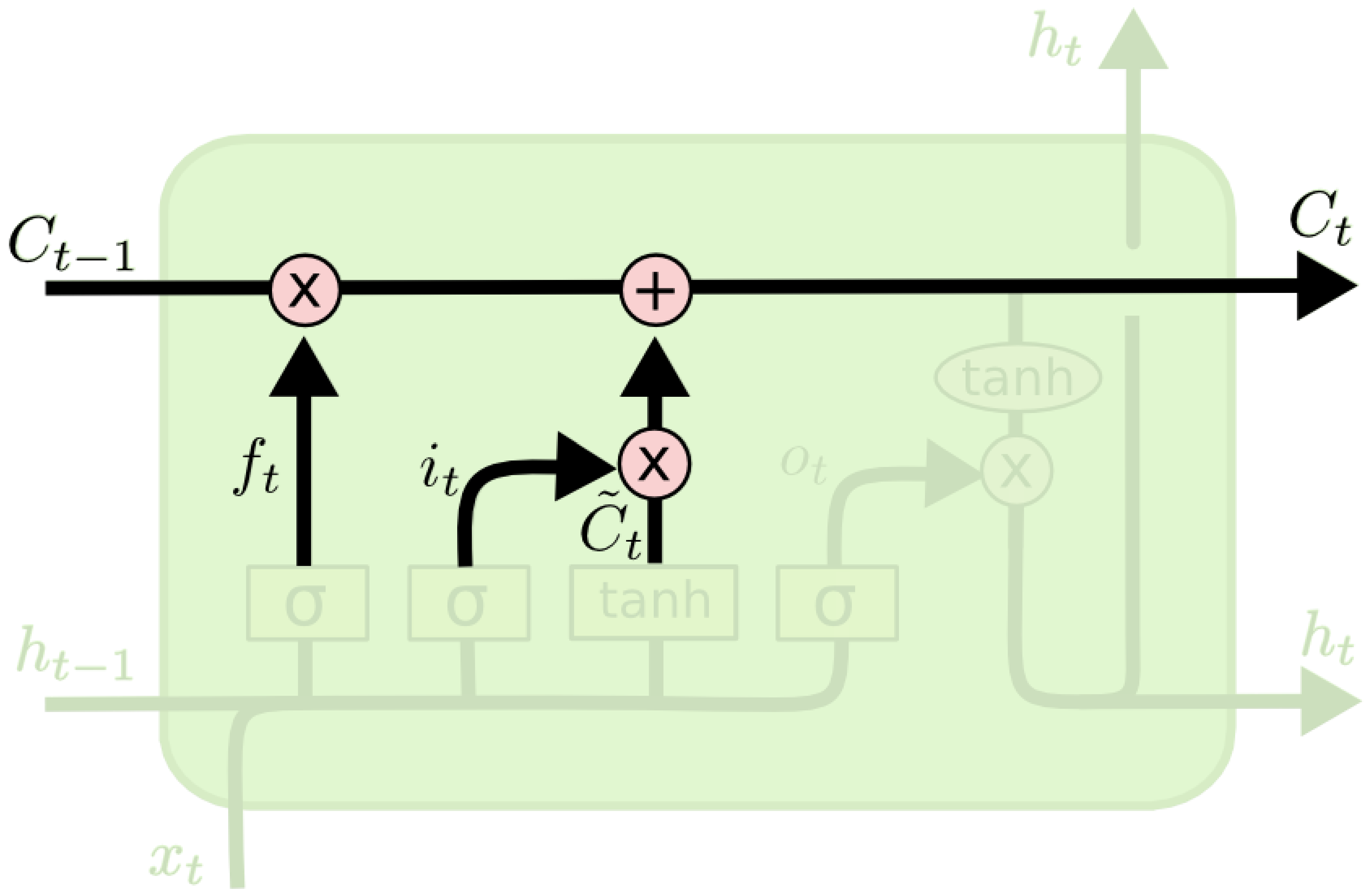
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$



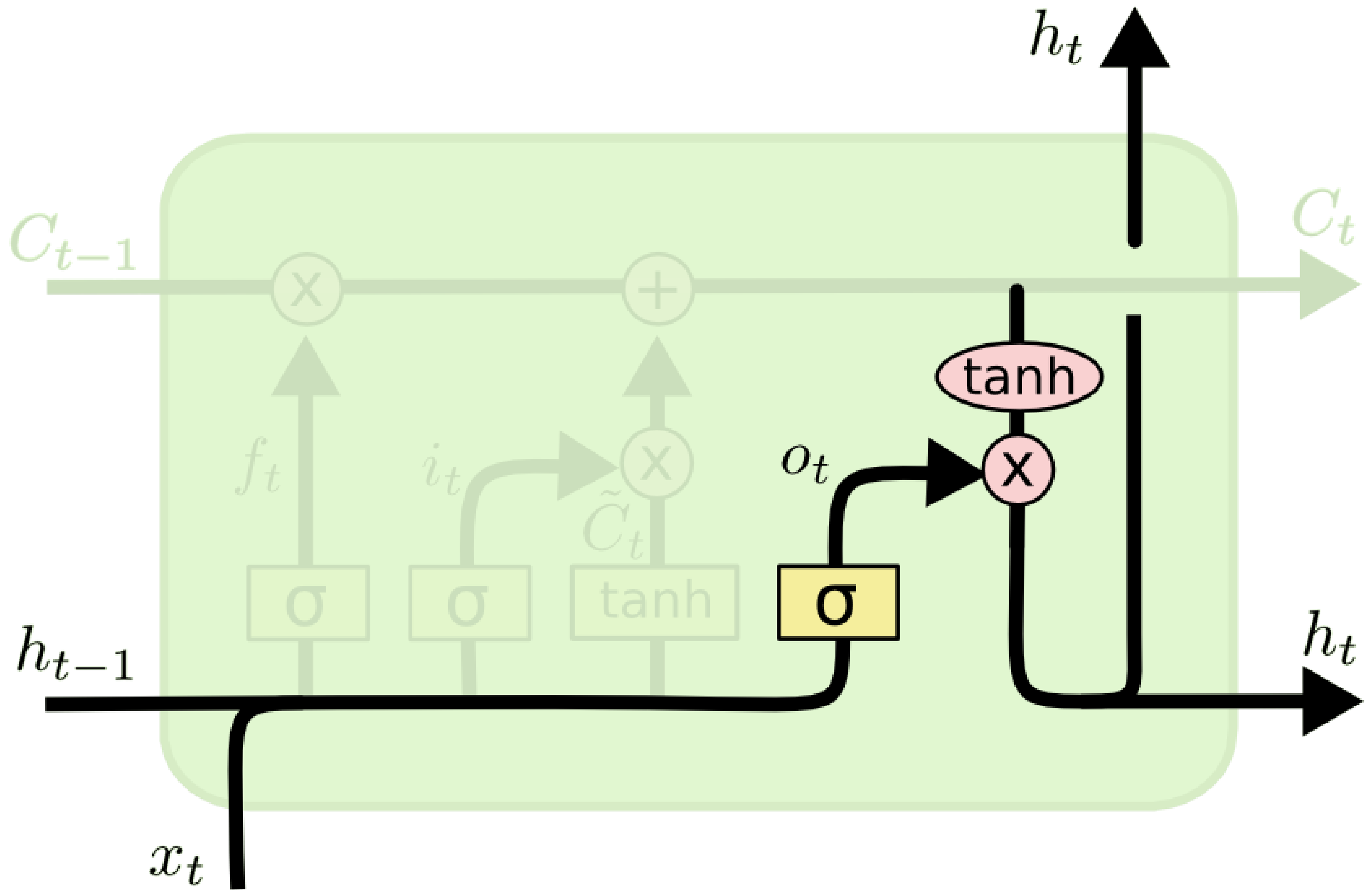
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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

LSTM-2



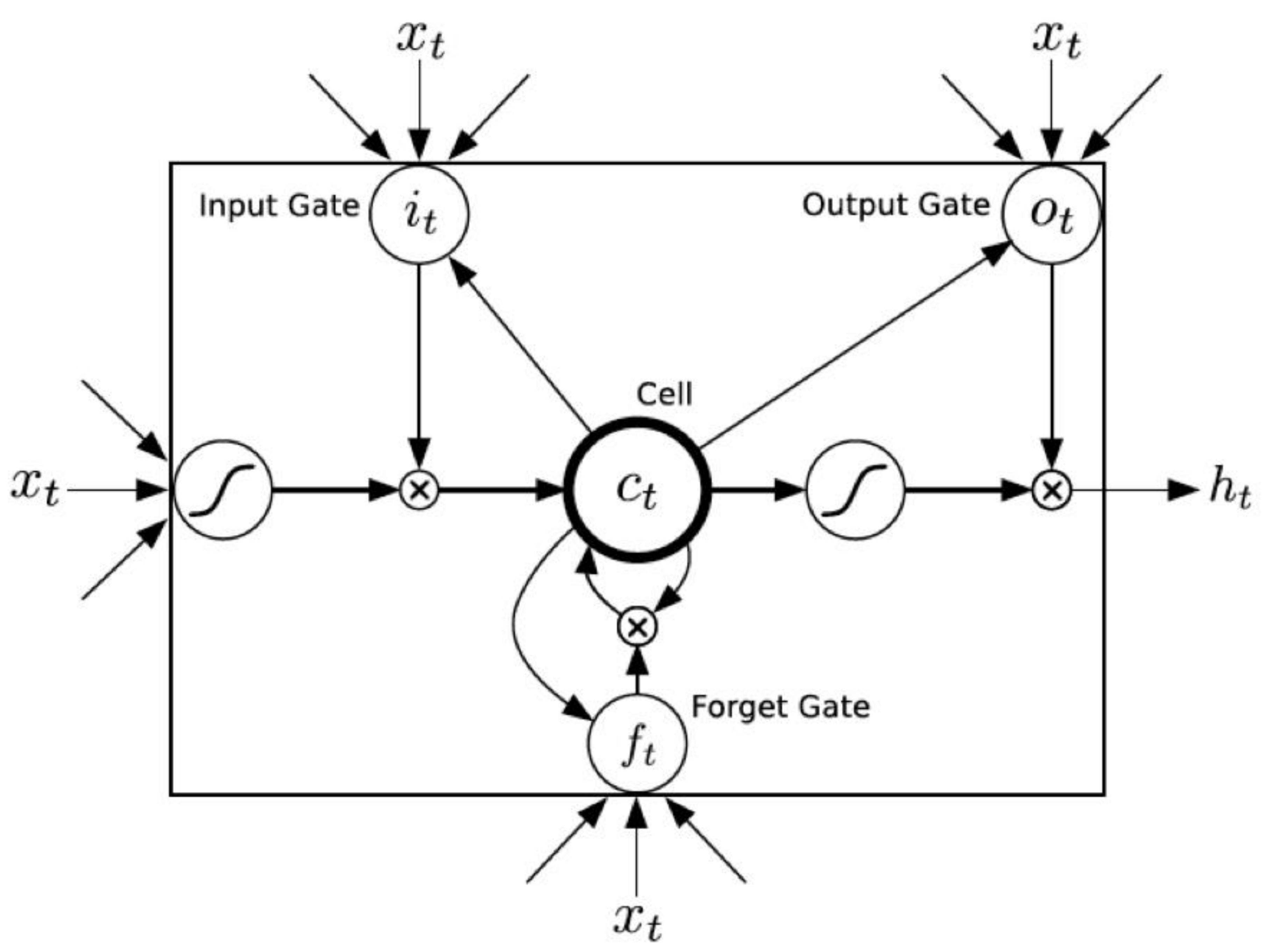
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



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

LSTM layer

- LSTM internal

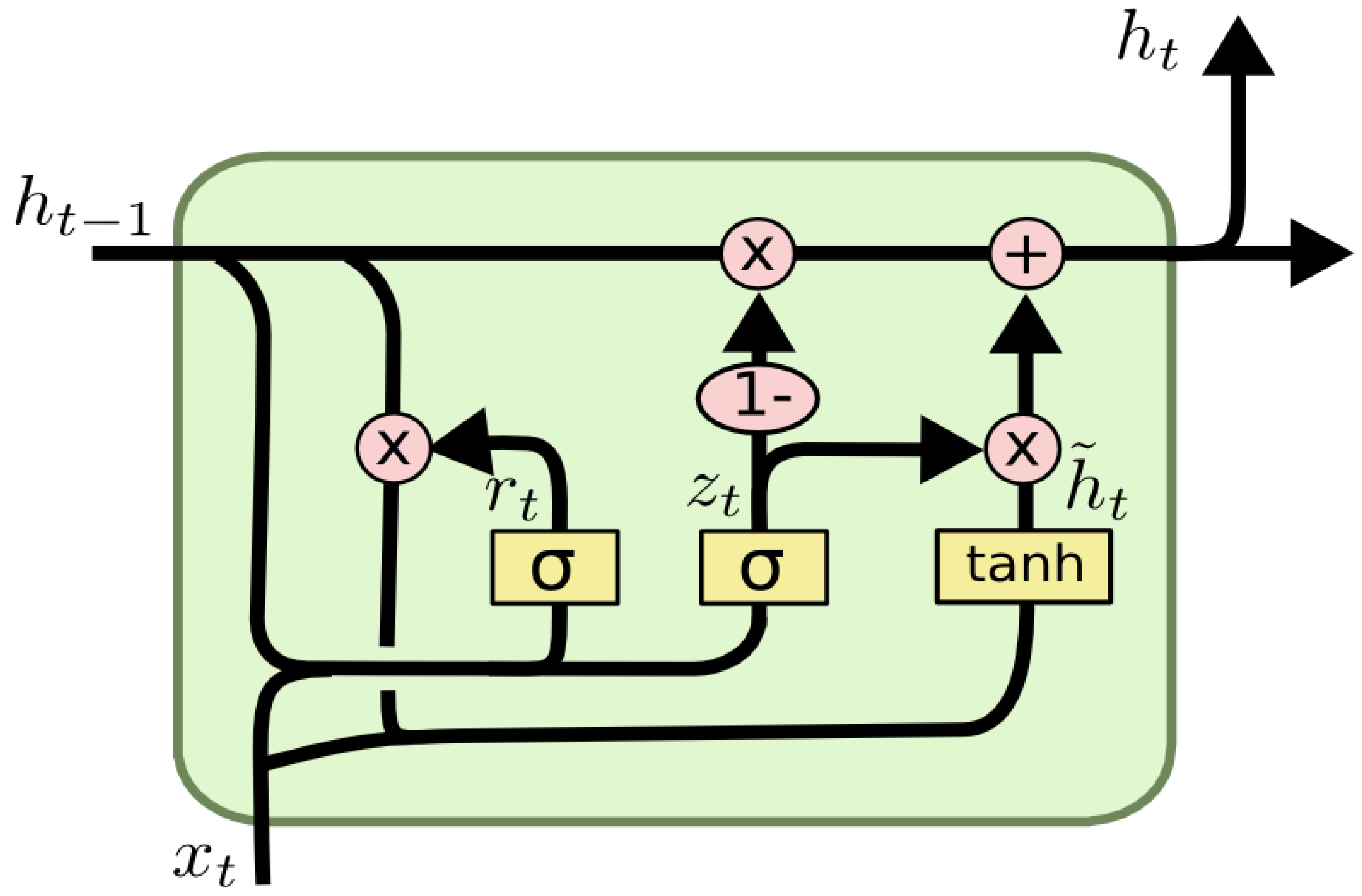



```
tf.keras.layers.LSTM(
    units, activation='tanh', recurrent_activation='sigmoid', use_bias=True,
    kernel_initializer='glorot_uniform', recurrent_initializer='orthogonal',
    bias_initializer='zeros', unit_forget_bias=True, kernel_regularizer=None,
    recurrent_regularizer=None, bias_regularizer=None, activity_regularizer=None,
    kernel_constraint=None, recurrent_constraint=None, bias_constraint=None,
    dropout=0.0, recurrent_dropout=0.0, implementation=2, return_sequences=False,
    return_state=False, go_backwards=False, stateful=False, time_major=False,
    unroll=False, **kwargs
)
```

https://tensorflow.google.cn/api_docs/python/tf/keras/layers/LSTM

GRU layer

- GRU (Gated Recurrent Unit)只有两个门：
  - LSTM的遗忘门（forget gates）和输入门（input gates）组合为单一的**更新门（Update gates）**
  - **重置门（reset gate）**，如果重置门关闭，会忽略掉历史信息
- LSTM的cell状态（ cell state ）和隐藏态（hidden state）合并为一个状态



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

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

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

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

K. Cho et al., On the Properties of Neural Machine Translation Encoder-Decoder Approaches, SSST-8, 2014.

GRU layer

- inputs: A 3D tensor, with shape [batch, timesteps, feature]

```
tf.keras.layers.GRU(
    units, activation='tanh', recurrent_activation='sigmoid', 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, implementation=2, return_sequences=False,
    return_state=False, go_backwards=False, stateful=False, unroll=False,
    time_major=False, reset_after=True, **kwargs
)
```

https://tensorflow.google.cn/api_docs/python/tf/keras/layers/GRU

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/

TensorFlow 2.0 API

Dense: https://tensorflow.google.cn/api_docs/python/tf/keras/layers/Dense

Softmax: https://tensorflow.google.cn/api_docs/python/tf/keras/layers/Softmax

Conv. Layer: https://tensorflow.google.cn/api_docs/python/tf/keras/layers/Conv2D

Pooling: https://tensorflow.google.cn/api_docs/python/tf/keras/layers/MaxPool2D

Dropout: https://tensorflow.google.cn/api_docs/python/tf/keras/layers/Dropout

SimpleRNNCell: https://tensorflow.google.cn/api_docs/python/tf/keras/layers/SimpleRNNCell

SimpleRNN: https://tensorflow.google.cn/api_docs/python/tf/keras/layers/SimpleRNN

RNN: https://tensorflow.google.cn/api_docs/python/tf/keras/layers/RNN

LSTM: https://tensorflow.google.cn/api_docs/python/tf/keras/layers/LSTM

谢谢指正！