# MACHINE LEARNING
# 机器学习
## Neural Networks

## 深度学习简介


参考：
《神经网络与深度学习》

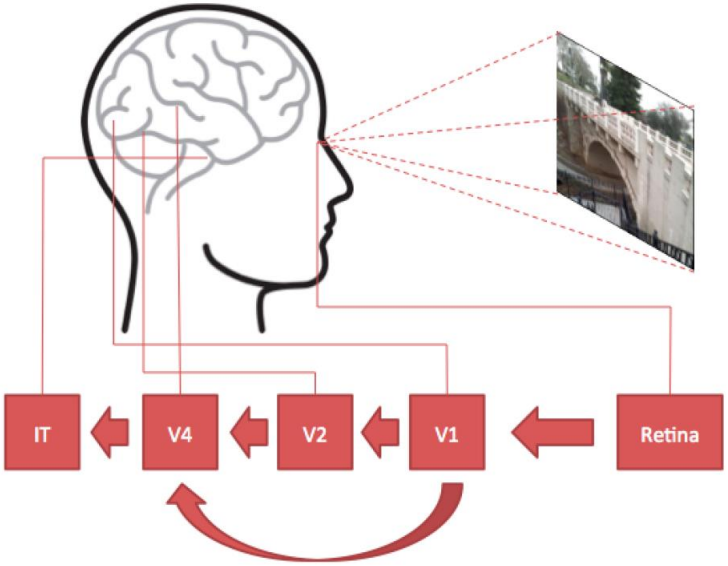
Machine Learning Course

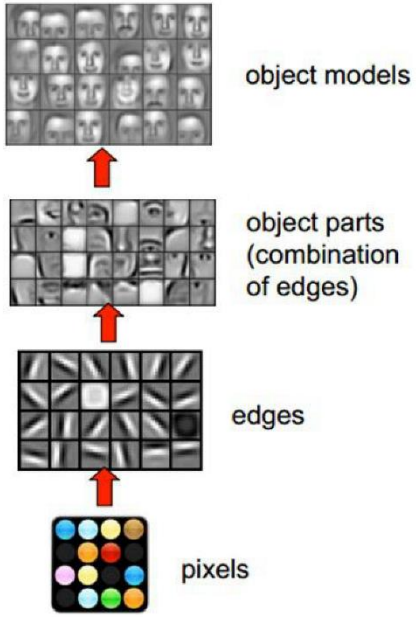
# 动机

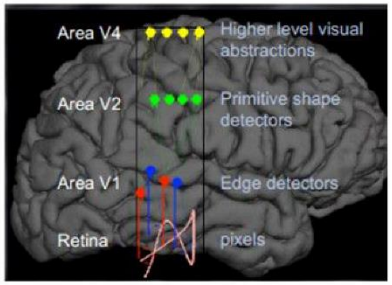
- 表示学习

为了提高机器学习系统的准确率，我们就需要将输入信息转换为有效的特征，或者更一般性地称为表示（Representation）．如果有一种算法可以自动地学习出有效的特征，并提高最终机器学习模型的性能，那么这种学习就可以叫作表示学习（Representation Learning）

人类的可视皮层具有“深度结构/层次结构”(deep architecture)




从低级的 V1区提取边缘特征，再到 V2区的形状或者目标的部分等，再到更高层，整个目标、目标的行为等。也就是说高层的特征是低层特征的组合，从低层到高层的特征表示越来越抽象，越来越能表现语义或者意图。而抽象层面越高，就越利于分类。

# 动机

- 表示学习

一般而言，一个好的表示具有以下几个优点：

（1）一个好的表示应该具有很强的表示能力，即同样大小的向量可以表示更多信息.

（2）一个好的表示应该使后续的学习任务变得简单，即需要包含更高层的语义信息.

（3）一个好的表示应该具有一般性，是任务或领域独立的. 虽然目前的大部分表示学习方法还是基于某个任务来学习，但我们期望其学到的表示可以比较容易地迁移到其他任务上. ）

# 动机

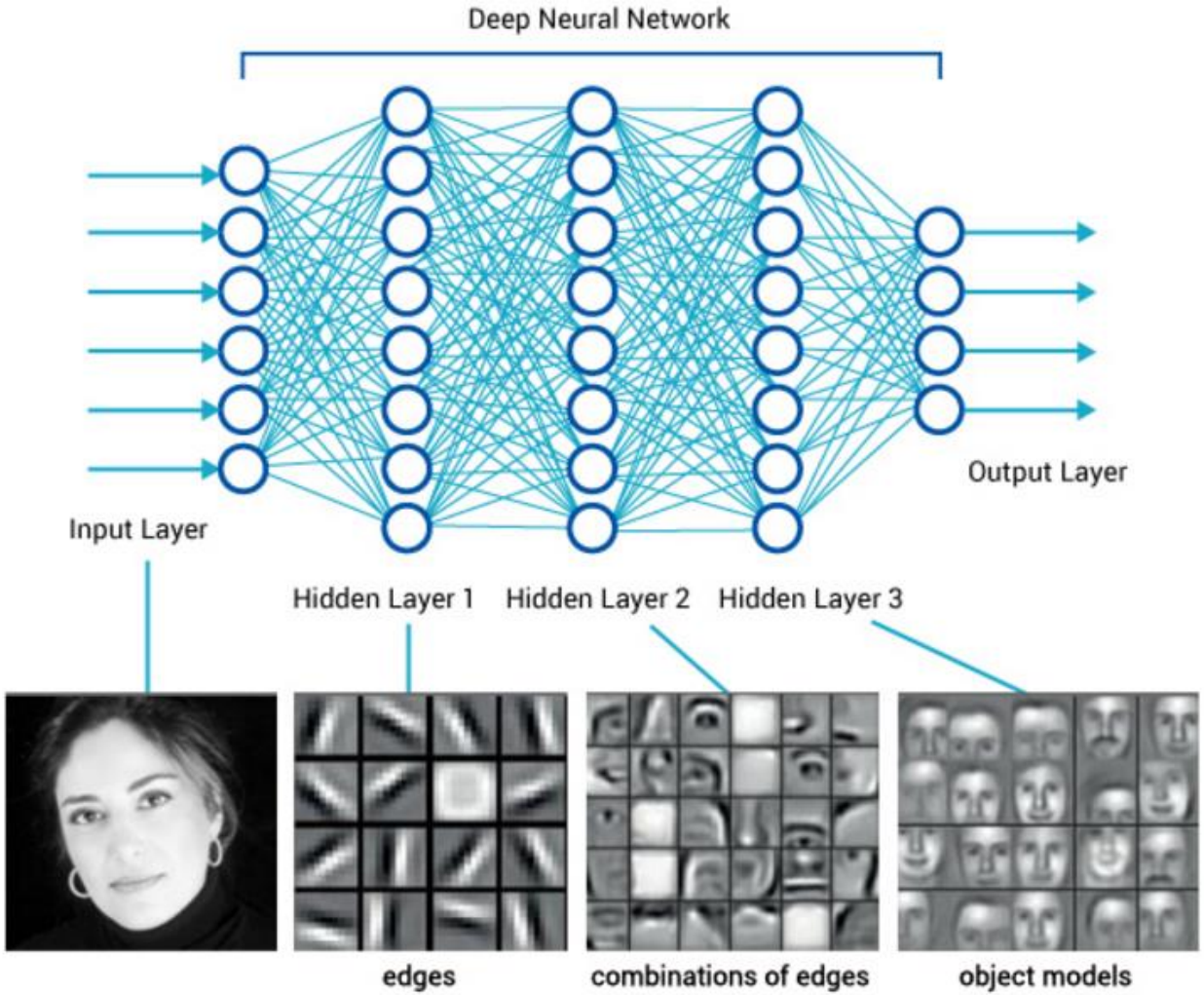
- 表示学习

要学习到一种好的高层语义表示（一般为分布式表示），通常需要从底层特征开始，经过多步非线性转换才能得到.

深层结构的优点是可以增加特征的重用性，从而指数级地增加表示能力．因此，表示学习的关键是构建具有一定深度的多层次特征表示 [Bengio et al., 2013]

# 深层困境

- 深度全连接前馈网络



# 深层困境

- 深度全连接前馈网络面临的难点
  - 参数过多
  - 梯度消失
  - 非凸优化
  - 解释性
  - 对数据和计算资源的需求

# 卷积神经网络

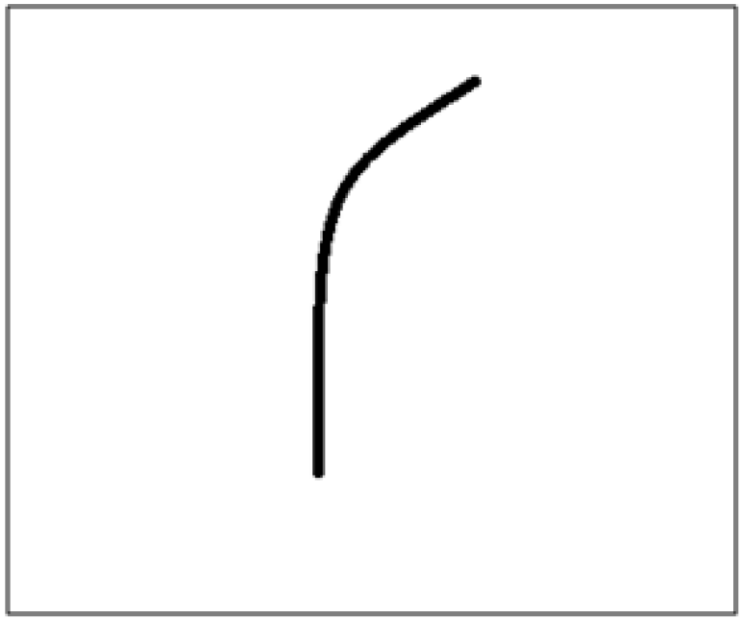
- 特点

卷积神经网络 *Convolutional Neural Network (CNN or ConvNet)* 是一种具有**局部连接、权重共享**等特性的**深层前馈神经网络**。

# 卷积神经网络

- 特点

卷积神经网络 *Convolutional Neural Network (CNN or ConvNet)* 是一种具有**局部连接、权重共享**等特性的**深层前馈神经网络**。

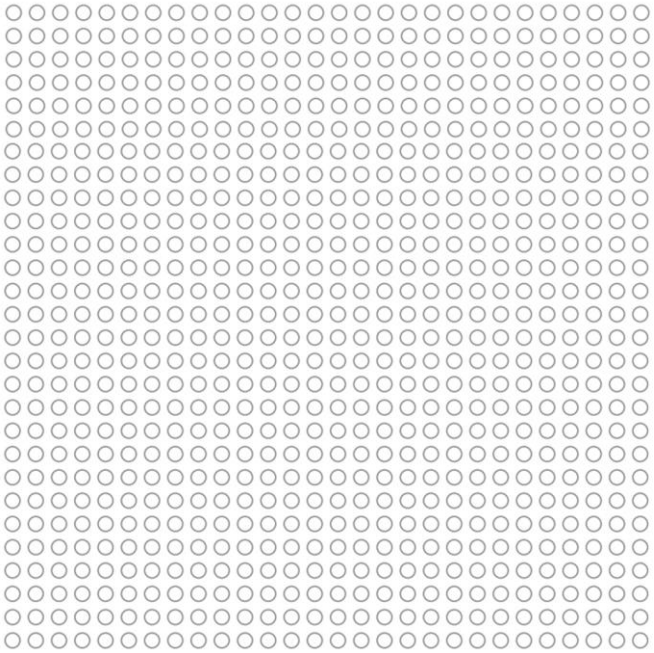


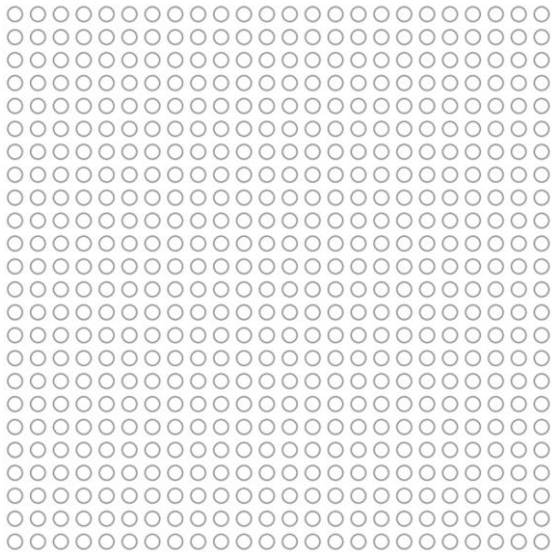
| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 30 | 0 |
| 0 | 0 | 0 | 0 | 30 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# 卷积神经网络

- 特点

卷积神经网络 *Convolutional Neural Network (CNN or ConvNet)* 是一种具有**局部连接、权重共享**等特性的**深层前馈神经网络**。

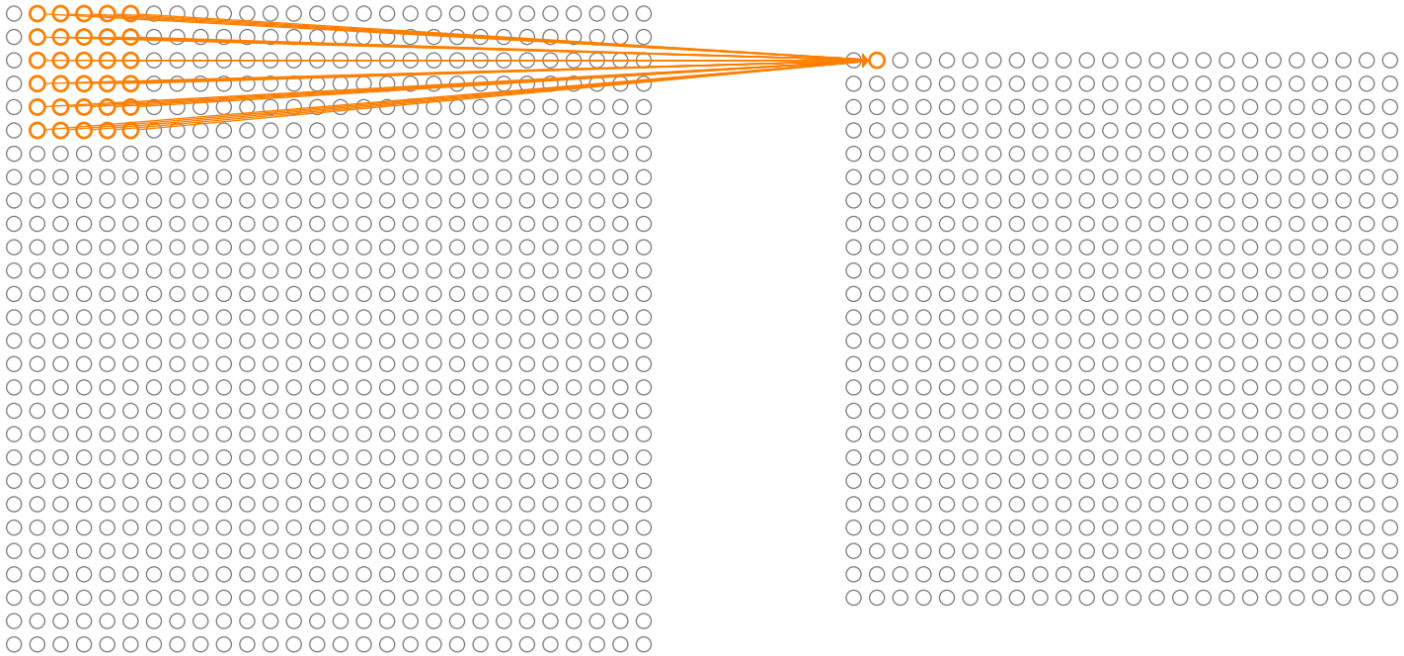




# 卷积神经网络

- 特点

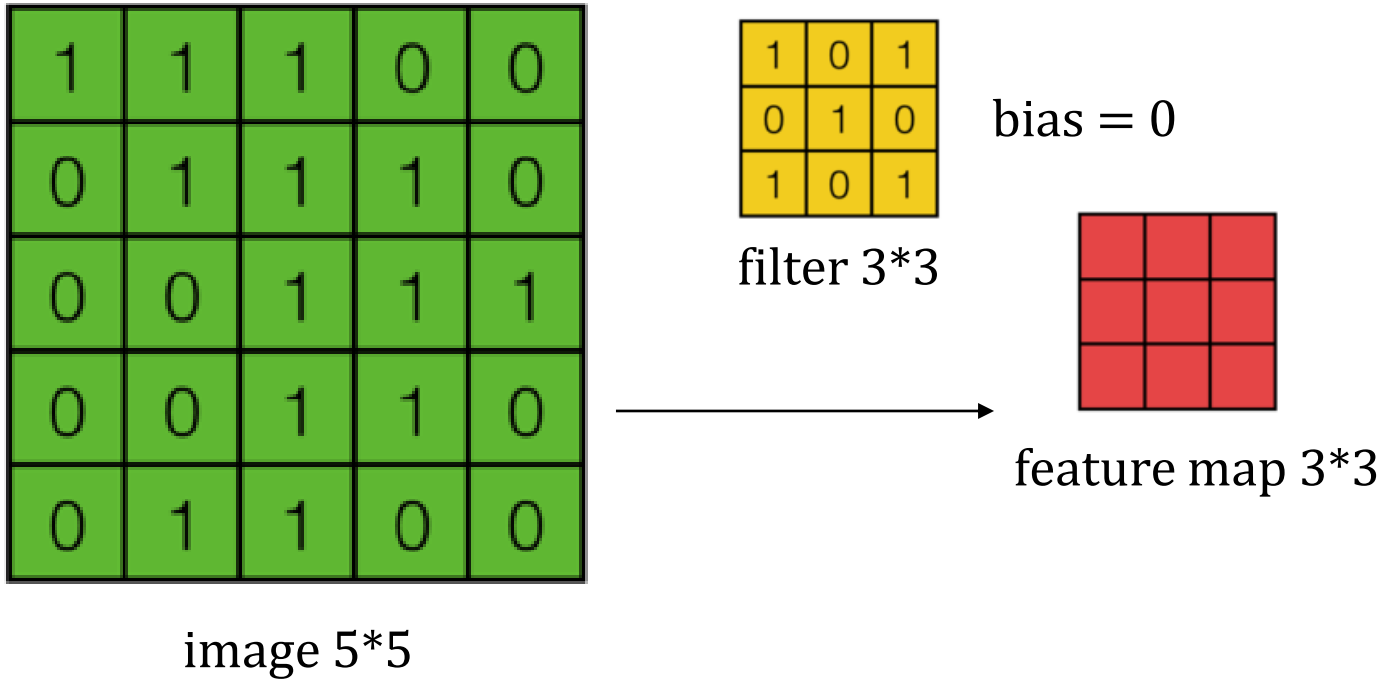
卷积神经网络 *Convolutional Neural Network (CNN or ConvNet)* 是一种具有**局部连接、权重共享**等特性的**深层前馈神经网络**。



# 卷积神经网络

- 卷积层

卷积层由卷积核作为算子实现数据表示的转换



# 卷积神经网络

- 卷积层

卷积层由卷积核作为算子实现数据表示的转换

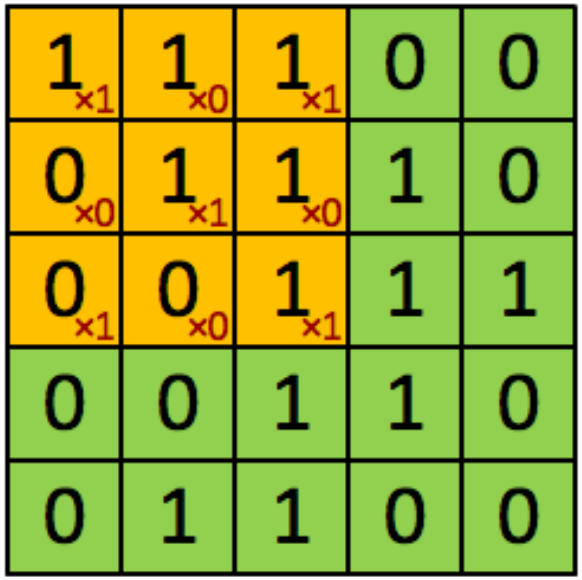
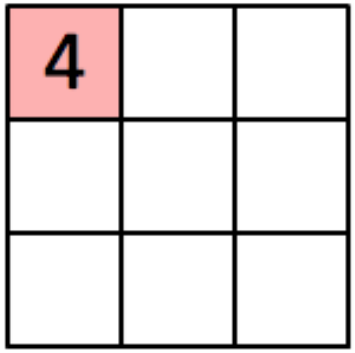



image 5*5

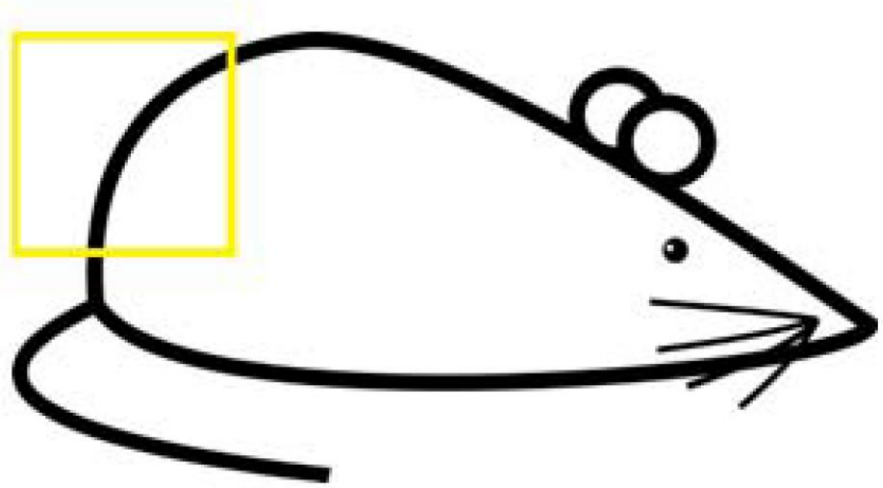



feature map 3*3

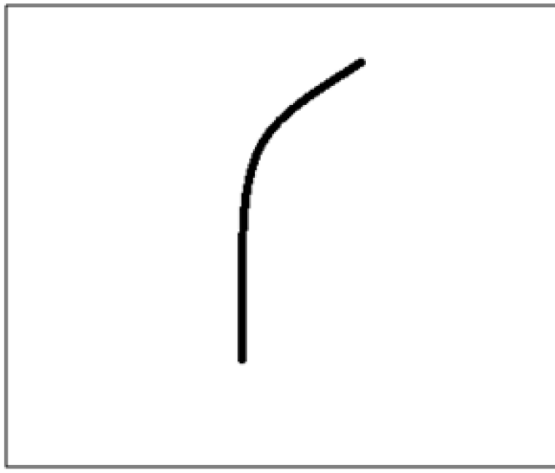
# 卷积神经网络

- 卷积层

卷积层由卷积核作为算子实现数据表示的转换



| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 30 | 0 |
| 0 | 0 | 0 | 0 | 30 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |



$$(50*30)+(50*30)+(50*30)+(20*30)+(50*30)=6600$$

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 30 |
| 0 | 0 | 0 | 0 | 50 | 50 | 50 |
| 0 | 0 | 0 | 20 | 50 | 0 | 0 |
| 0 | 0 | 0 | 50 | 50 | 0 | 0 |
| 0 | 0 | 0 | 50 | 50 | 0 | 0 |
| 0 | 0 | 0 | 50 | 50 | 0 | 0 |
| 0 | 0 | 0 | 50 | 50 | 0 | 0 |

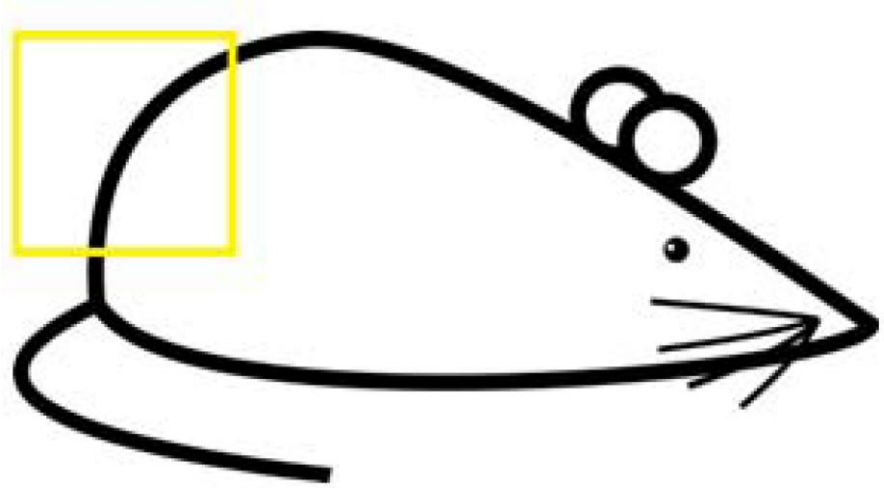
$*$

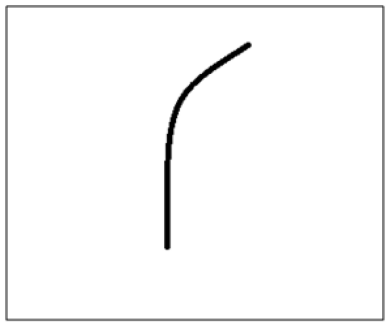
| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 30 | 0 |
| 0 | 0 | 0 | 0 | 30 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# 卷积神经网络

- 卷积层

卷积层由卷积核作为算子实现数据表示的转换





*

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 30 | 0 |
| 0 | 0 | 0 | 0 | 30 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

*

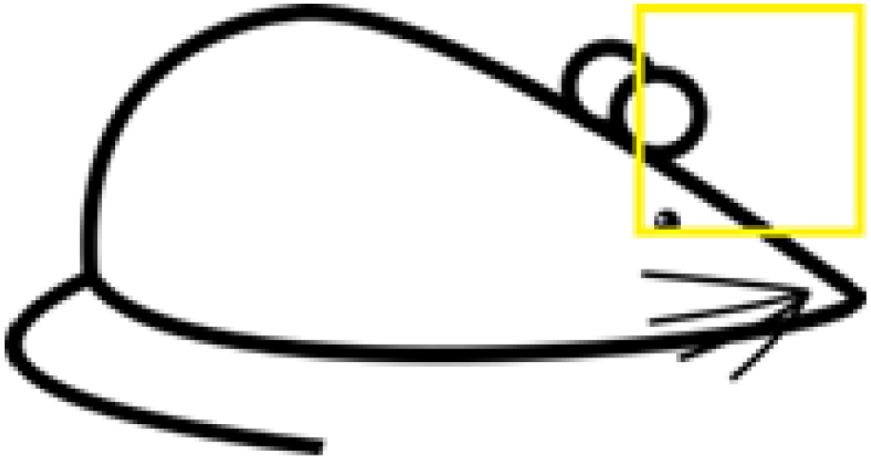
| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 30 |
| 0 | 0 | 0 | 0 | 50 | 50 | 50 |
| 0 | 0 | 0 | 20 | 50 | 0 | 0 |
| 0 | 0 | 0 | 50 | 50 | 0 | 0 |
| 0 | 0 | 0 | 50 | 50 | 0 | 0 |
| 0 | 0 | 0 | 50 | 50 | 0 | 0 |
| 0 | 0 | 0 | 50 | 50 | 0 | 0 |

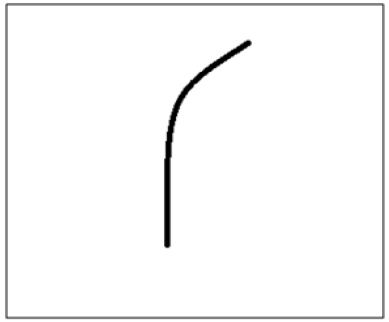
$$(50^*30) + (50^*30) + (50^*30) + (20^*30) + (50^*30) = 6600$$

# 卷积神经网络

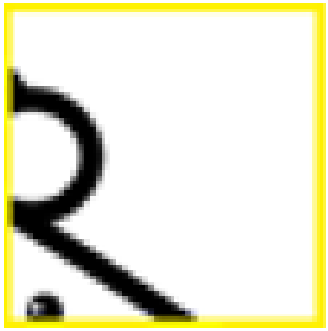
- 卷积层

卷积层由卷积核作为算子实现数据表示的转换





*



| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 30 | 0 |
| 0 | 0 | 0 | 0 | 30 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

*

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 40 | 0 | 0 | 0 | 0 | 0 |
| 40 | 0 | 40 | 0 | 0 | 0 | 0 |
| 40 | 20 | 0 | 0 | 0 | 0 | 0 |
| 0 | 50 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 50 | 0 | 0 | 0 | 0 |
| 25 | 25 | 0 | 50 | 0 | 0 | 0 |

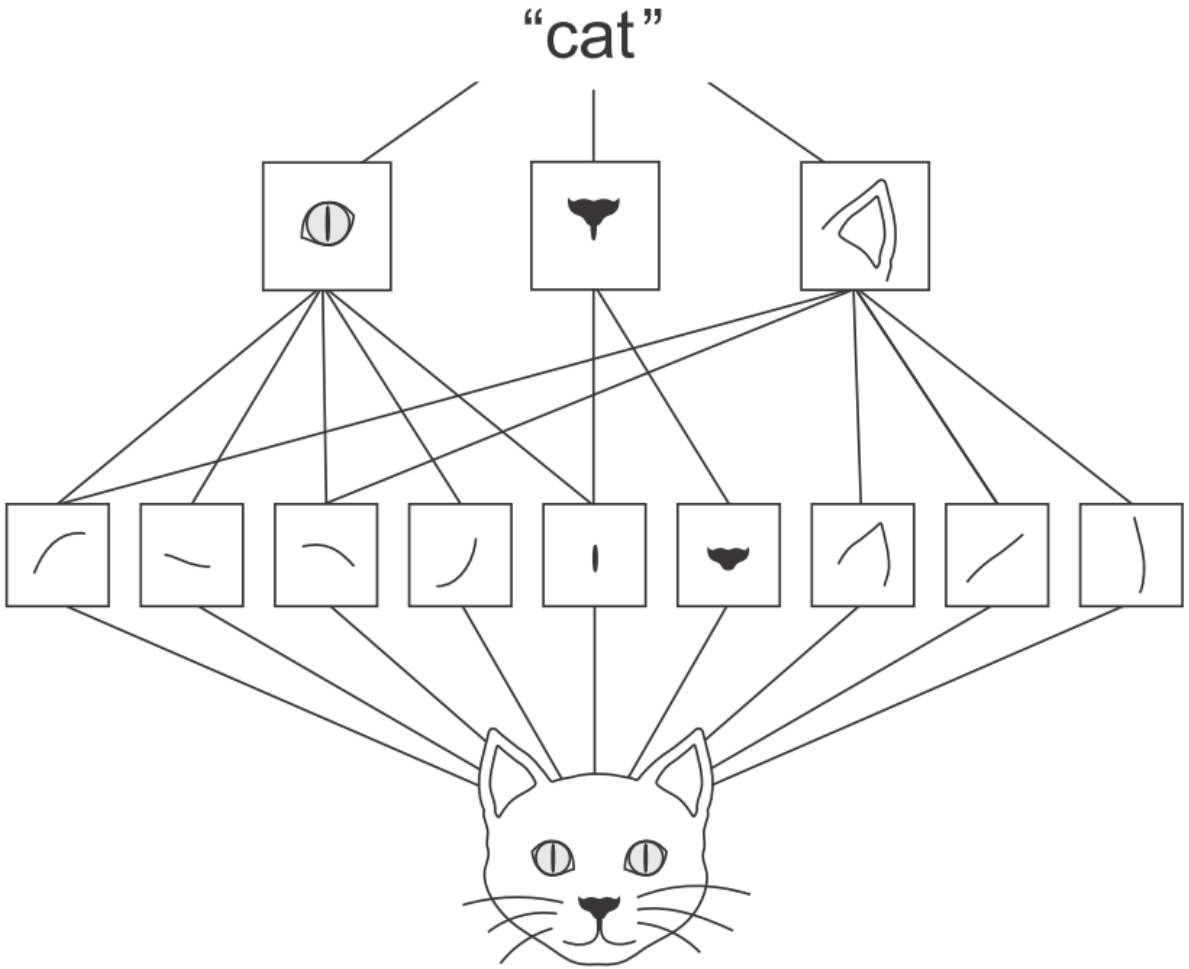
0

# 卷积神经网络

- 卷积层

一层卷积层由多个卷积核共同作用实现提取数据不同角度的抽象特征

卷积层超参数之 卷积核个数

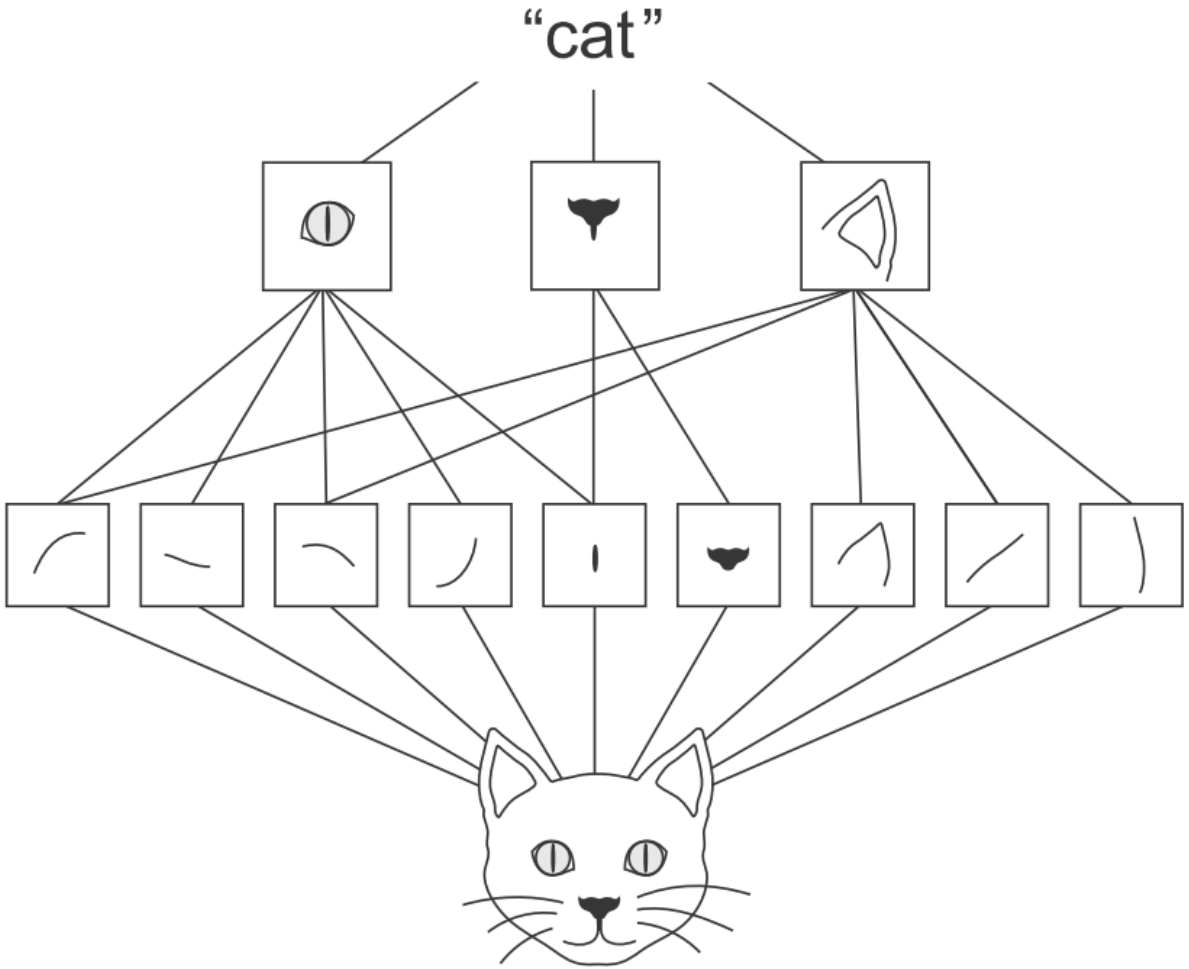


# 卷积神经网络

- 卷积层

一层卷积层由多个卷积核共同作用实现提取数据不同角度的抽象特征

卷积层超参数之 卷积核个数



# 卷积神经网络

- 卷积层

卷积层超参数之 卷积核形状

- 长，宽，深

| | | | | | |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |

6 x 6 black & white picture image

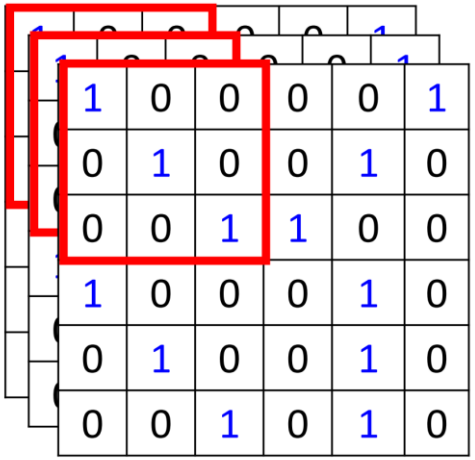
$$a_{i,j} = f\left(\sum_{m=0}^{F-1}\sum_{n=0}^{F-1} w_{m,n}x_{i+m,j+n} + w_b\right)$$

# 卷积神经网络

- 卷积层

卷积层超参数之 卷积核形状

- 长，宽，深




6 x 6 colorful image

$$a_{i,j} = f\left(\sum_{d=0}^{D-1}\sum_{m=0}^{F-1}\sum_{n=0}^{F-1} w_{d,m,n}x_{d,i+m,j+n} + w_b\right)$$

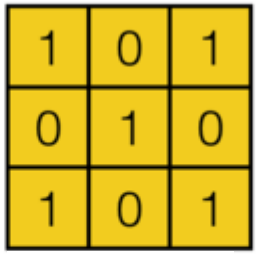
# 卷积神经网络

- 卷积层

卷积层超参数之 卷积核步幅 *Stride*

- 卷积核滑动时的间隔


image 5*5




bias = 0

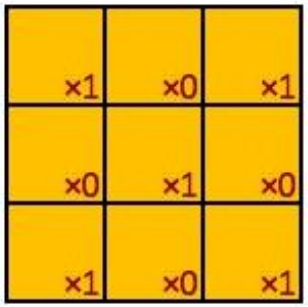
filter 3*3


feature map 2*2

# 卷积神经网络

- 卷积层

卷积层超参数之 卷积核填充 *Padding*



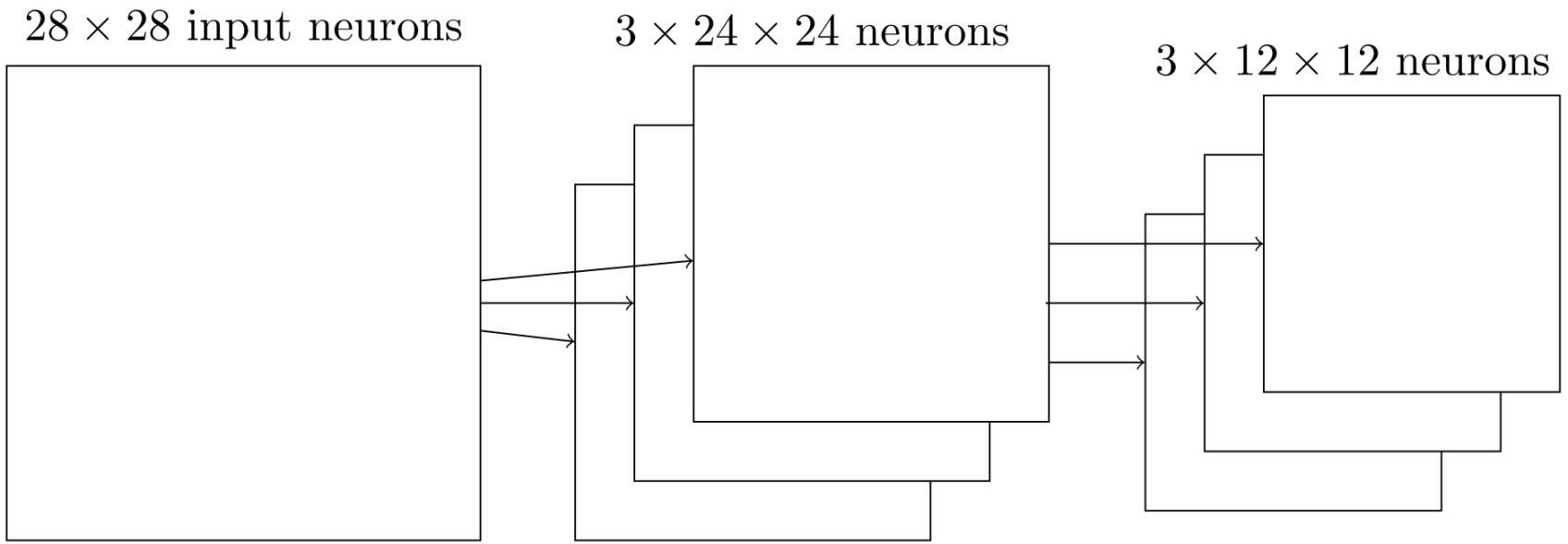

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | | | | |
|---|---|---|---|---|
| 2 | 2 | 3 | 1 | 1 |
| 1 | 4 | 3 | 4 | 1 |
| 2 | 2 | 4 | 3 | 3 |
| 1 | 2 | 3 | 4 | 1 |
| 1 | 2 | 3 | 1 | 1 |

# 卷积神经网络

- 卷积层

示例

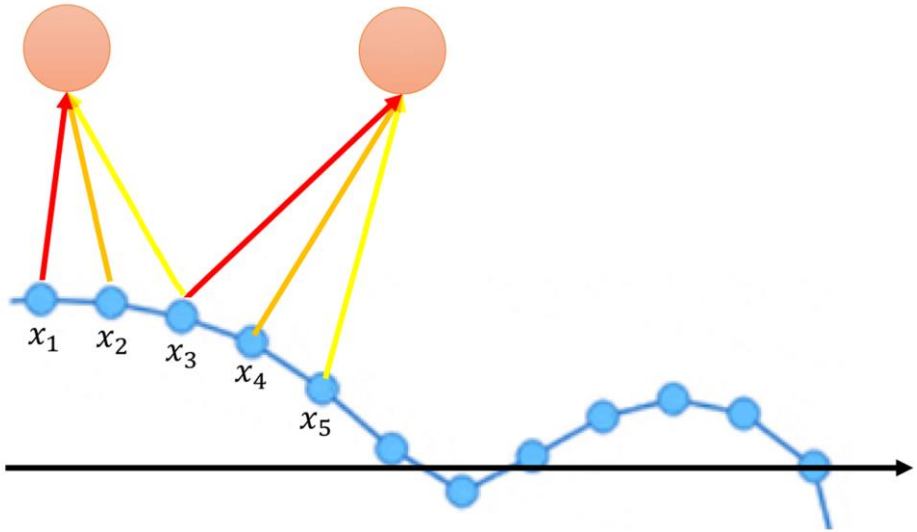


# 卷积神经网络

- 卷积层

卷积层超参数之 卷积核维度

例如 *Conv2D*？*Conv1D*？



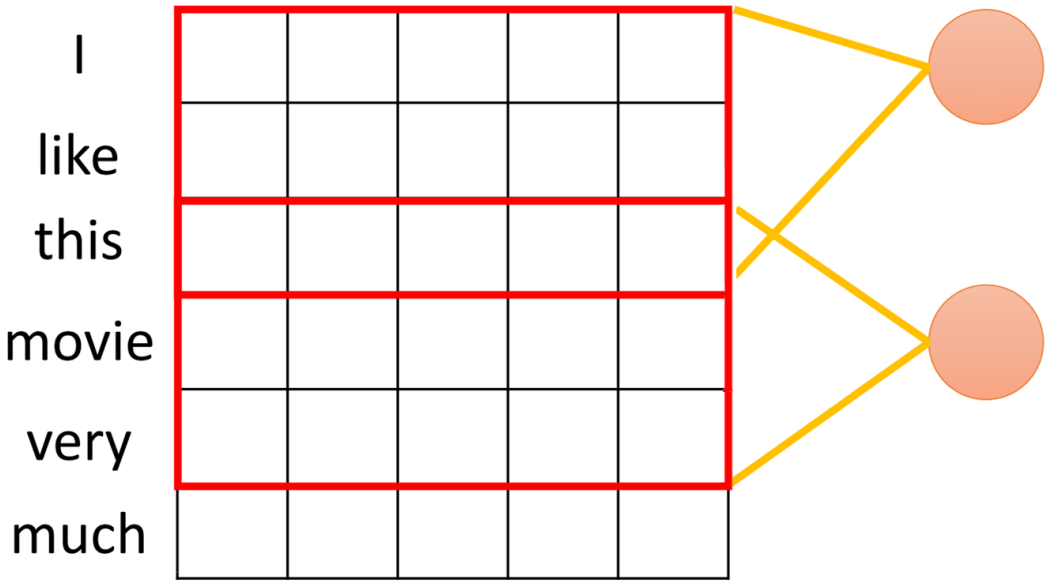

$$y_t = w_1 \times x_t + w_2 \times x_{t-1} + w_3 \times x_{t-2}$$

# 卷积神经网络

- 卷积层

卷积层超参数之 卷积核维度

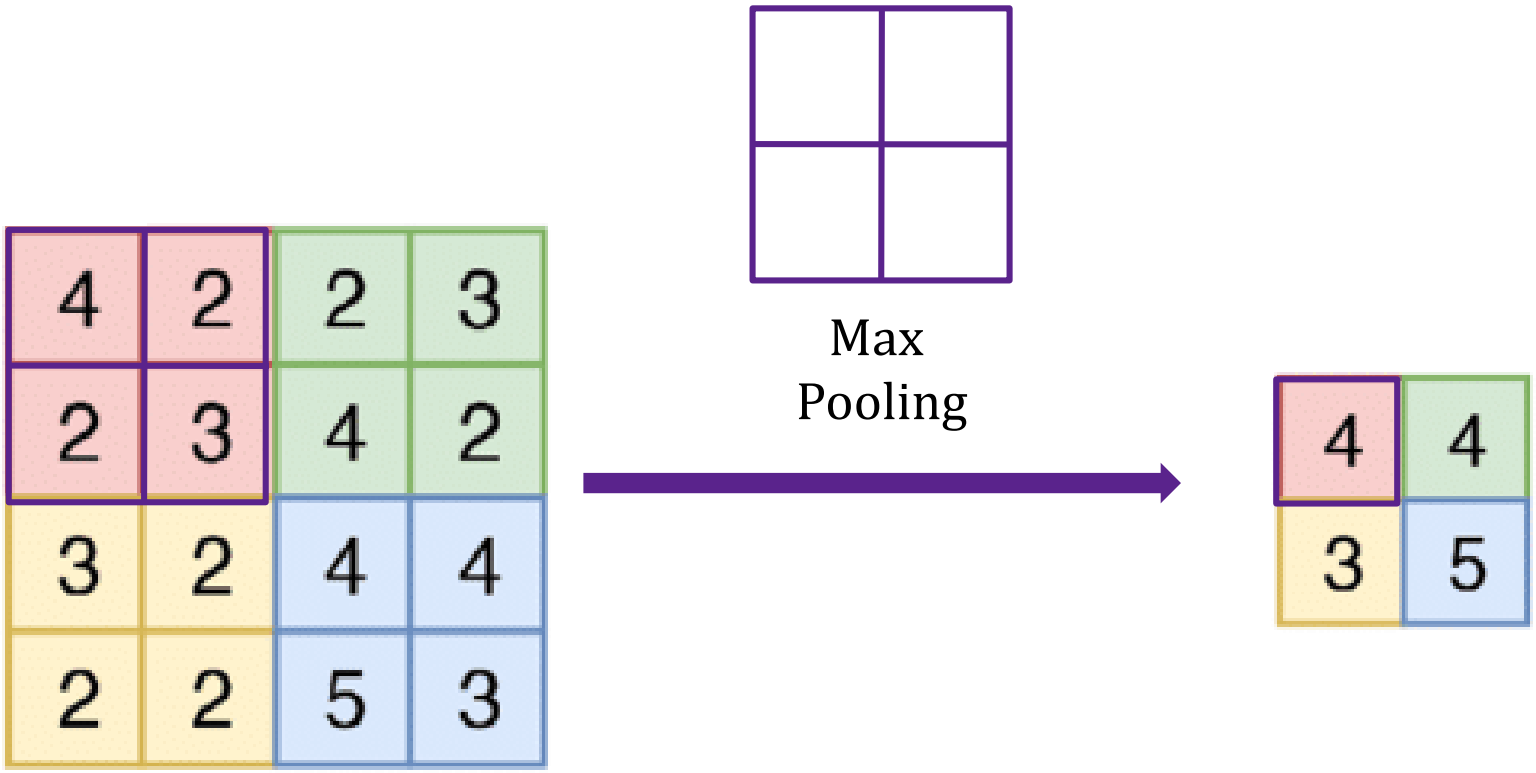
例如 *Conv2D*?*Conv1D*?



# 卷积神经网络

- 汇聚层

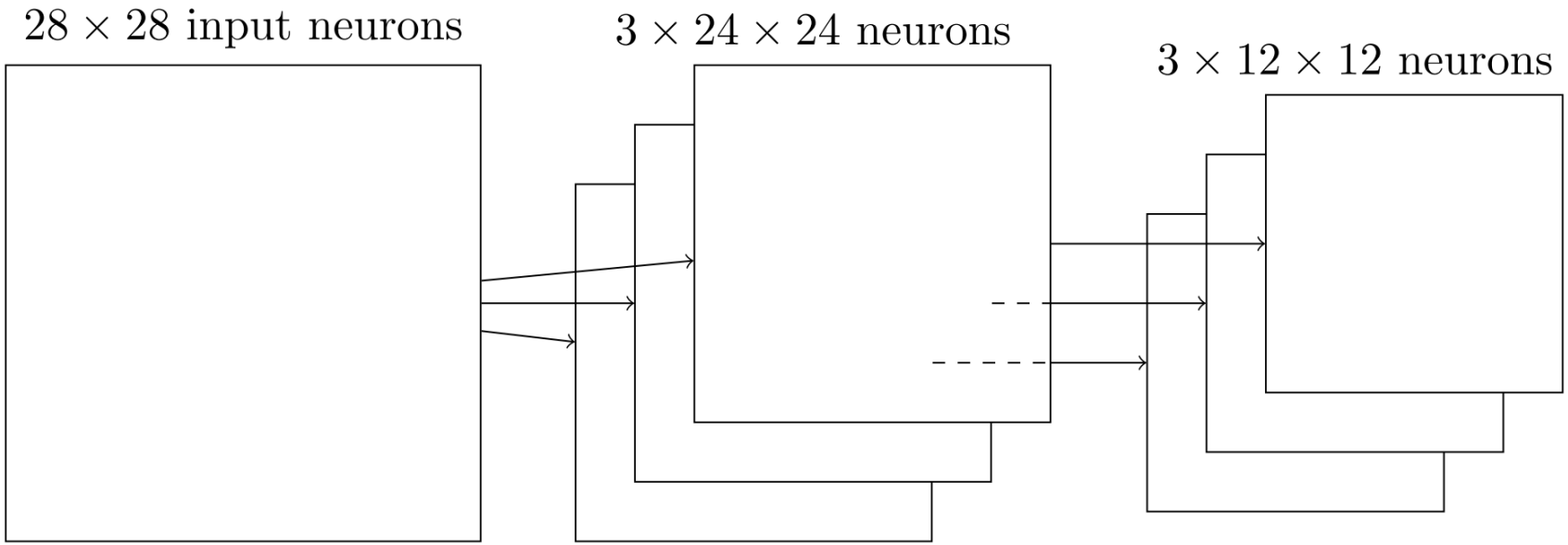
卷积层超参数之 卷积核维度



# 卷积神经网络

- 汇聚层

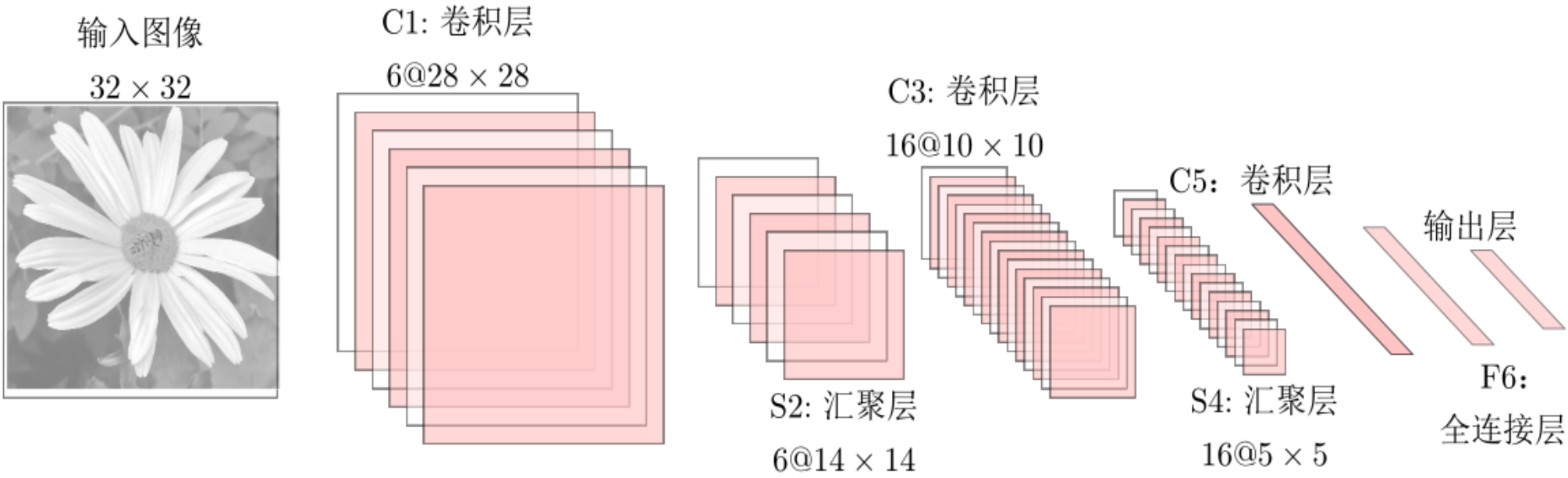
卷积层超参数之 卷积核维度



# 卷积神经网络

- 经典卷积神经网络

LeNet-5模型



# MACHINE LEARNING

# 实践

## Practice


参考：
《Python深度学习》

# 深度学习实践

- 数据/张量
  - 标量（0D 张量） *Scalars (0D tensors)*

仅包含一个数字的张量叫作标量（scalar，也叫标量张量、零维张量、0D 张量）。在 Numpy 中，一个 float32 或 float64 的数字就是一个标量张量（或标量数组）。你可以用 ndim 属性来查看一个 Numpy 张量的轴的个数。标量张量有 0 个轴（ndim == 0）。张量轴的个数也叫作阶 *rank*。下面定义了一个 Numpy 标量：

```
>>> import numpy as np
>>> x = np.array(12)
>>> x
array(12)
>>> x.ndim
0
```

# 深度学习实践

- 数据/张量
  - 向量（1D 张量） *Vectors (1D tensors)*

数字组成的数组叫作向量（vector）或一维张量（1D 张量）。一维张量只有一个轴。下面定义了一个 Numpy 向量：

```
>>> x = np.array([12, 3, 6, 14])
>>> x
array([12, 3, 6, 14])
>>> x.ndim
1
```

这个向量有 5 个元素，所以被称为 5D 向量。不要把 5D 向量*5D vector*和 5D 张量***5D tensor***弄混！ 5D 向量只有一个轴 *axis*，沿着轴有 5 个维度 *dimensions*，而 5D 张量有 5 个轴（沿着每个轴可能有任意个维度）。维度 *dimensionality* 可以表示沿着某个轴上的元素个数（比如 5D 向量），也可以表示张量中轴的个数（比如 5D 张量），这有时会令人感到混乱。对于后一种情况，技术上更准确的说法是 5 阶张量 *a tensor of rank 5*（张量的阶数即轴的个数），但 5D 张量这种模糊的写法更常见。

# 深度学习实践

- 数据/张量
  - 矩阵（2D 张量）*Matrices (2D tensors)*

向量组成的数组叫作矩阵（matrix）或二维张量（2D 张量）。矩阵有 2 个轴（通常叫作行和列）。可以将矩阵直观地理解为数字组成的矩形网格。下面是一个 Numpy 矩阵。

```
>>> x = np.array([[5, 78, 2, 34, 0],
[6, 79, 3, 35, 1],
[7, 80, 4, 36, 2]])
>>> x.ndim
2
```

一批1D数据点（即每个数据点被编码称为一个向量）可以被编码成为一个2D张量（即向量组成的数组），矩阵的两个轴可以看作是样本轴和特征轴。例如：

1. 人口统计数据集：其中包括每个人的年龄、邮编和收入。每个人可以表示为包含 3 个值的向量，而整个数据集包含 100 000 个人，因此可以存储在形状为 (100000, 3) 的 2D张量中。
2. 文档数据集：我们将每个文档表示为每个单词在其中出现的次数（字典中包含20,000 个常见单词）。每个文档可以被编码为包含 20,000 个值的向量（每个值对应于字典中每个单词的出现次数），整个数据集包含 500 个文档，因此可以存储在形状为(500, 20000) 的张量中。

# 深度学习实践

- 数据/张量
  - 3D 张量 *3D tensors*（或更高阶张量）

将多个矩阵组合成一个新的数组，可以得到一个 3D 张量，你可以将其直观地理解为数字组成的立方体。下面是一个 Numpy 的 3D 张量。

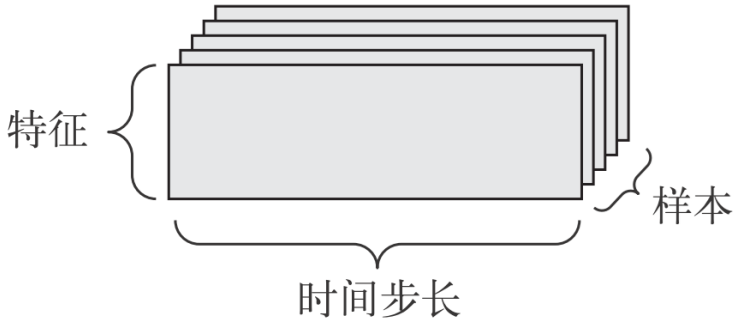
```
>>> x = np.array([[[5, 78, 2, 34, 0],
[6, 79, 3, 35, 1],
[7, 80, 4, 36, 2]],
[[5, 78, 2, 34, 0],
[6, 79, 3, 35, 1],
[7, 80, 4, 36, 2]],
[[5, 78, 2, 34, 0],
[6, 79, 3, 35, 1],
[7, 80, 4, 36, 2]]])
>>> x.ndim
3
```

将多个*N*D张量组合成一个数组，可以创建一个*N+1*D张量

# 深度学习实践

- 数据/张量
  - 3D 张量 *3D tensors*（或更高阶张量）

当时间（或序列顺序）对于数据很重要时，应该将数据存储在带有时间轴的 3D 张量中。每个样本可以被编码为一个向量序列（即 2D 张量），因此一个数据批量就被编码为一个 3D 张量：




根据惯例，时间轴始终是第 2 个轴（索引为 1 的轴）。示例：

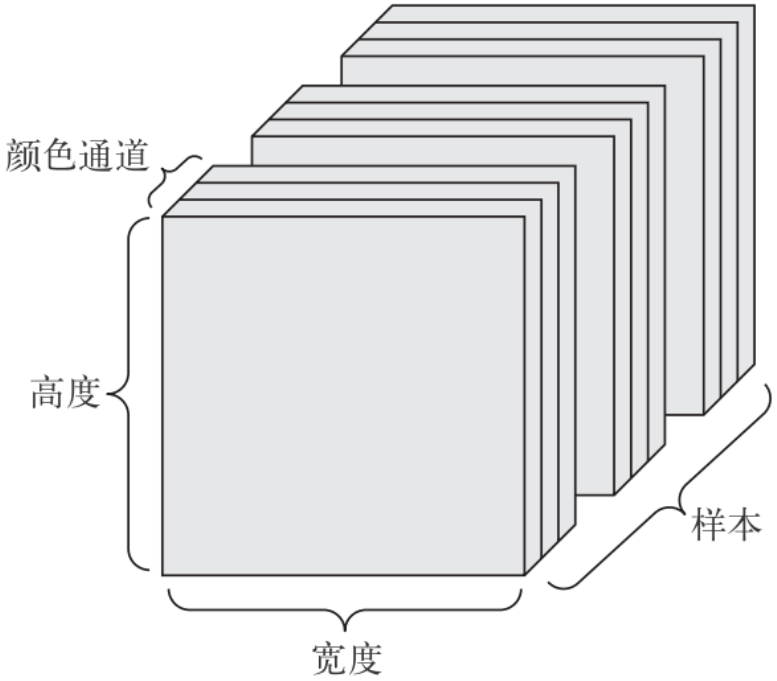
1.股票价格数据集：每一分钟，我们将股票的当前价格、前一分钟的最高价格和前一分钟的最低价格保存下来。因此每分钟被编码为一个 3D 向量，整个交易日被编码为一个形状为 (390, 3) 的 2D 张量（一个交易日有 390 分钟），而 250 天的数据则可以保存在一个形状为 (250, 390, 3) 的 3D 张量中。这里每个样本是一天的股票数据。

2.推文数据集。我们将每条推文编码为 280 个字符组成的序列，而每个字符又来自于 128个字符组成的字母表。在这种情况下，每个字符可以被编码为大小为 128 的二进制向量（只有在该字符对应的索引位置取值为 1，其他元素都为 0）。那么每条推文可以被编码为一个形状为 (280, 128) 的 2D 张量，而包含 100 万条推文的数据集则可以存储在一个形状为 (1000000, 280, 128) 的张量中。

# 深度学习实践

- 数据/张量
  - 3D 张量 *3D tensors*（或更高阶张量）

图像通常具有三个维度：高度、宽度和颜色深度。虽然灰度图像（比如 MNIST 数字图像）只有一个颜色通道，因此可以保存在 2D 张量中，但按照惯例，图像张量始终都是 3D 张量，灰度图像的彩色通道只有一维。因此，如果图像大小为 256×256，那么 128 张灰度图像组成的批量可以保存在一个形状为 (128, 256, 256, 1) 的张量中，而 128 张彩色图像组成的批量则可以保存在一个形状为 (128, 256, 256, 3) 的张量中：




图像张量的形状有两种约定：通道在后（channels-last）的约定（在 TensorFlow 中使用）和通道在前（channels-first）的约定（在 Theano 中使用）。Google 的TensorFlow 机器学习框架将颜色深度轴放在最后：(samples, height, width, color_depth)。与此相反，Theano将图像深度轴放在批量轴之后：(samples, color_depth, height, width)。如果采用 Theano 约定，前面的两个例子将变成 (128, 1, 256, 256) 和 (128, 3, 256, 256)。Keras 框架同时支持这两种格式。

# 深度学习实践

- 数据/张量
  - 3D 张量 *3D tensors*（或更高阶张量）

视频数据是现实生活中需要用到 5D 张量的少数数据类型之一。视频可以看作一系列帧，每一帧都是一张彩色图像。由于每一帧都可以保存在一个形状为 (height, width, color_depth) 的 3D 张量中，因此一系列帧可以保存在一个形状为 (frames, height, width, color_depth) 的 4D 张量中，而不同视频组成的批量则可以保存在一个 5D 张量中，其形状为 (samples, frames, height, width, color_depth)。

# 深度学习实践

- 数据/张量
- 属性
- Number of axes (rank)

*>>> print(train_images.ndim) # 3*

- Shape

*>>> print(train_images.shape) # (60000, 28, 28)*

- Data type

*>>> print(train_images.dtype) # uint8*

# 深度学习实践

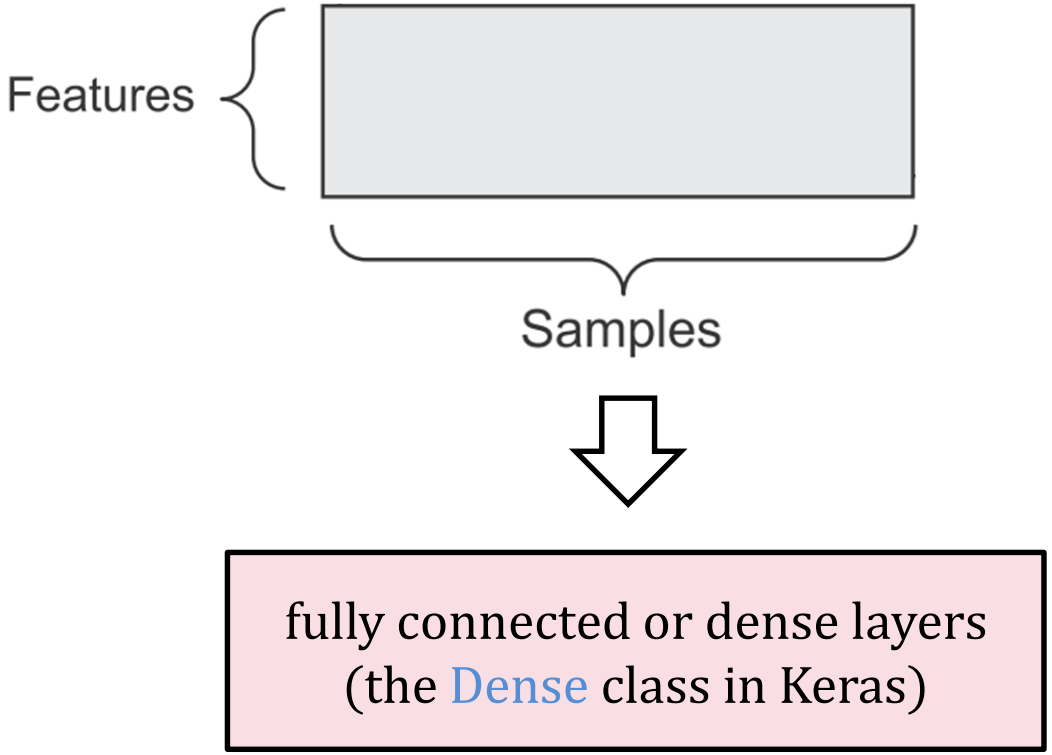
- 层

神经网络的基本数据结构是层。层是一个数据处理模块，将一个或多个输入张量转换为一个或多个输出张量。有些层是无状态的，但大多数的层是有**状态**的，即层的**权重**。权重是利用随机梯度下降学到的一个或多个张量，其中包含网络的知识。

# 深度学习实践

- 层

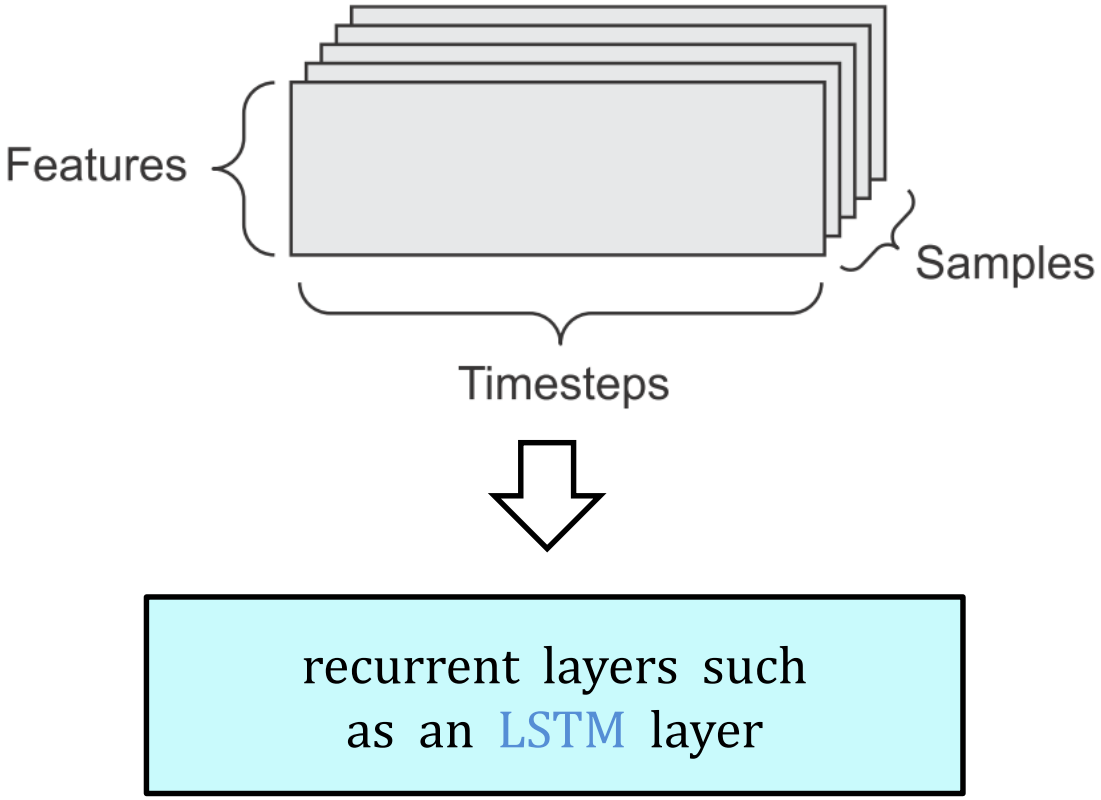
不同的张量格式与不同的数据处理类型需要用到不同的层。
简单的向量数据保存在形状为 (samples, features) 的 2D 张量中，通常用密集连接层*densely connected layer*，也叫全连接层 f*ully connected layer* 或密集层 *dense layer*，对应于 Keras 的 Dense 类来处理。



# 深度学习实践

- 层

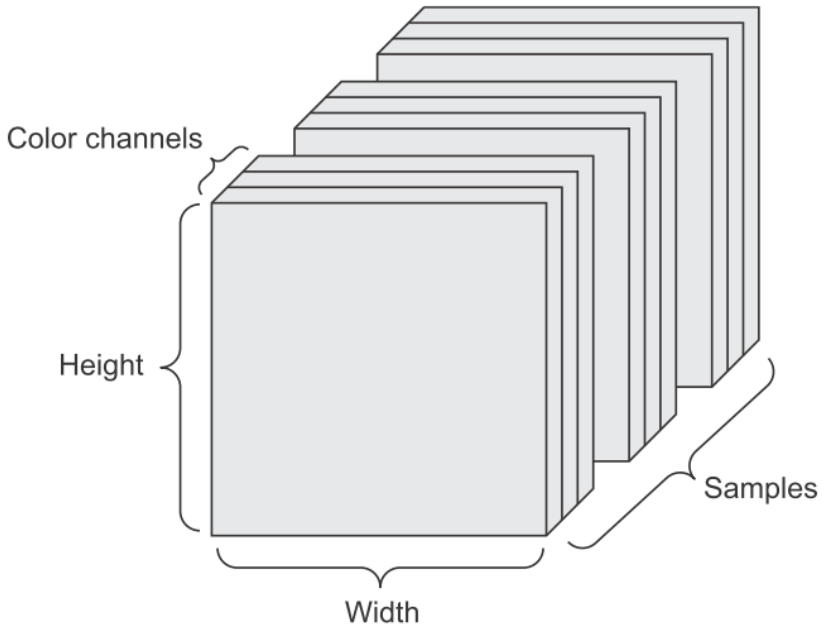
不同的张量格式与不同的数据处理类型需要用到不同的层。
序列数据保存在形状为 (samples, timesteps, features) 的 3D 张量中，通常用循环层（recurrent layer，比如 Keras 的 LSTM 层）来处理。



# 深度学习实践

- 层

不同的张量格式与不同的数据处理类型需要用到不同的层。
图像数据保存在 4D 张量中，通常用二维卷积层（Keras 的 Conv2D）来处理。




2D convolution layers (Conv2D)

# 深度学习实践

- 层兼容性 *layer compatibility*

我们可以将层看作深度学习的乐高积木，Keras 等框架则将这种比喻具体化。在 Keras 中，构建深度学习模型就是将相互兼容的多个层拼接在一起，以建立有用的数据变换流程。这里层兼容性（layer compatibility）具体指的是每一层只接受特定形状的输入张量，并返回特定形状的输出张量。

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(32, input_shape=(825,)))
model.add(layers.Dense(32))
```

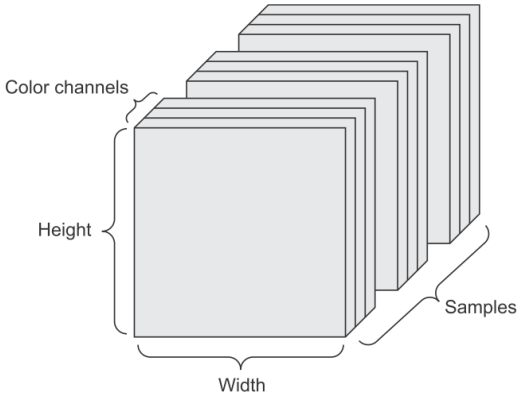
第一层只接受维度大小为825的2D张量（第 0 轴是批量维度，其大小没有指定，因此可以任意取值）作为输入。这个层将返回一个张量，第一个维度的大小变成了 32。

因此，这个层后面只能连接一个接受 32 维向量作为输入的层。使用 Keras 时，你无须担心兼容性，因为向模型中添加的层都会自动匹配输入层的形状。例如这里的第二层没有输入形状（input_shape）的参数，相反，它可以自动推导出输入形状等于上一层的输出形状。

# 深度学习实践

- 卷积神经网络实践

```
from keras import layers
from keras import models
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
print(model.summary())
```




(image_height, image_width, image_channels)

2D convolution layers (Conv2D)

keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding='valid', ...)

keras.layers.MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid', data_format=None)

# 深度学习实践

- 卷积神经网络实践

```
from keras import layers
from keras import models
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 3, 3, 64)          36928
=================================================================
Total params: 55,744
Trainable params: 55,744
Non-trainable params: 0
_________________________________________________________________
```

# 深度学习实践

- 卷积神经网络实践

```
from keras import layers
from keras import models
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
print(model.summary())

model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
print(model.summary())
```

# 深度学习实践

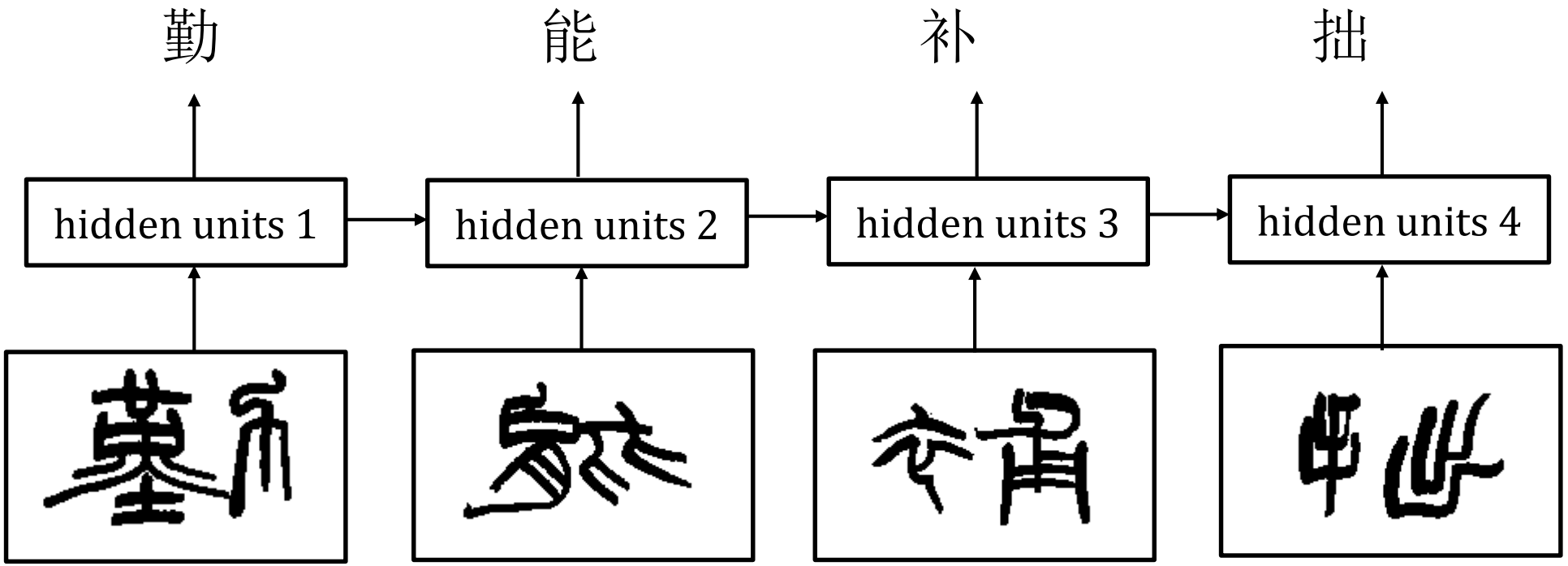
- 卷积神经网络实践

```
from keras.datasets import mnist
from keras.utils import to_categorical
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
model.compile(optimizer='rmsprop',
loss='categorical_crossentropy',
metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5, batch_size=64)

test_loss, test_acc = model.evaluate(test_images, test_labels)
print(test_acc)
#0.9908000000000001
```

# 循环神经网络

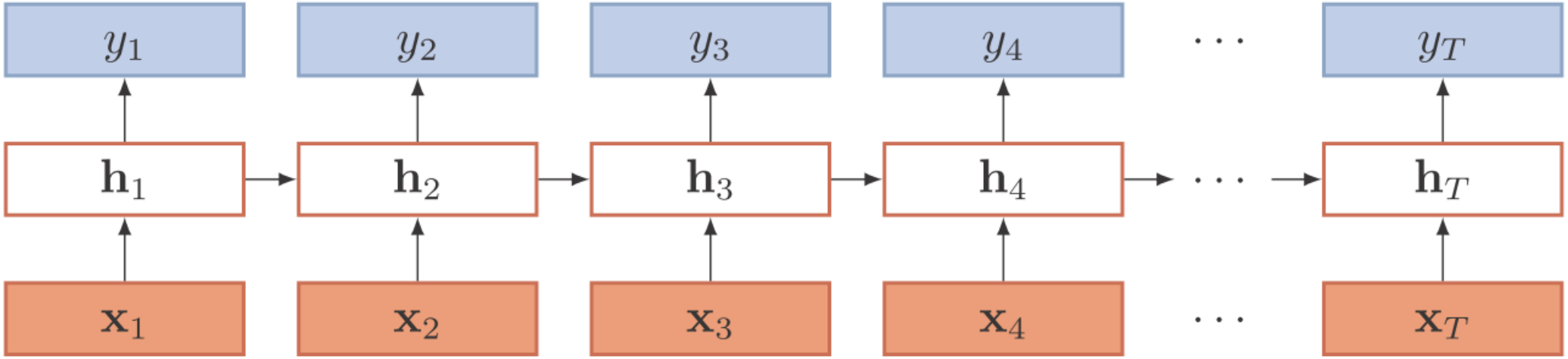
- 简单循环神经网络




前馈神经网络每次输入都是独立的，即网络的输出只依赖于当前的输入。
但是在很多现实任务中，网络的输入不仅和当前时刻的输入相关，也和其过去一段时间的输出相关。比如一个有限状态自动机，其下一个时刻的状态（输出）不仅仅和当前输入相关，也和当前状态（上一个时刻的输出）相关

# 循环神经网络

- 简单循环神经网络

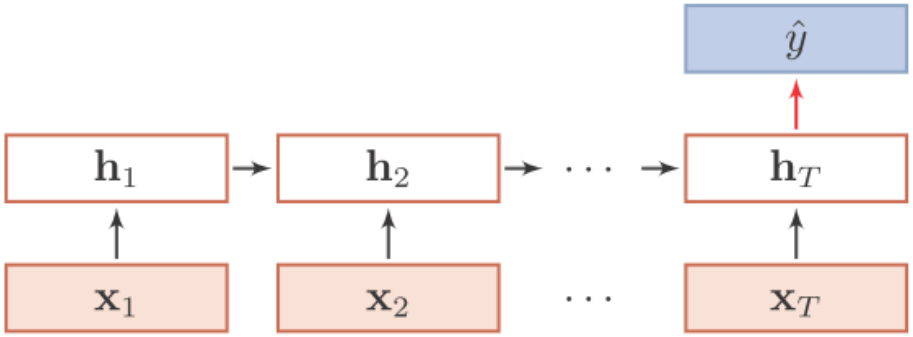



$$\mathbf{h}_t = \mathrm{f}(\mathbf{z}_t)$$
$$\boldsymbol{z}_t = U\boldsymbol{h}_{t-1} + W\boldsymbol{x}_t + b$$
$$\boldsymbol{y}_t = V\boldsymbol{h}_t$$

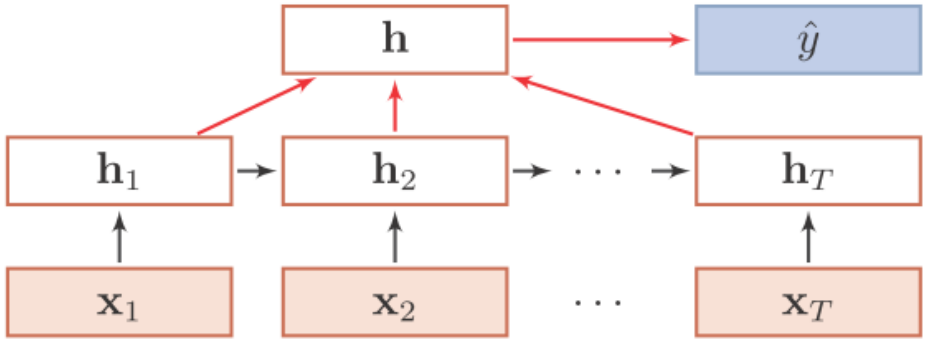
$U, V$在不同time step上是共享的

# 循环神经网络

- 序列到点




$$\hat{y} = g(h_T)$$

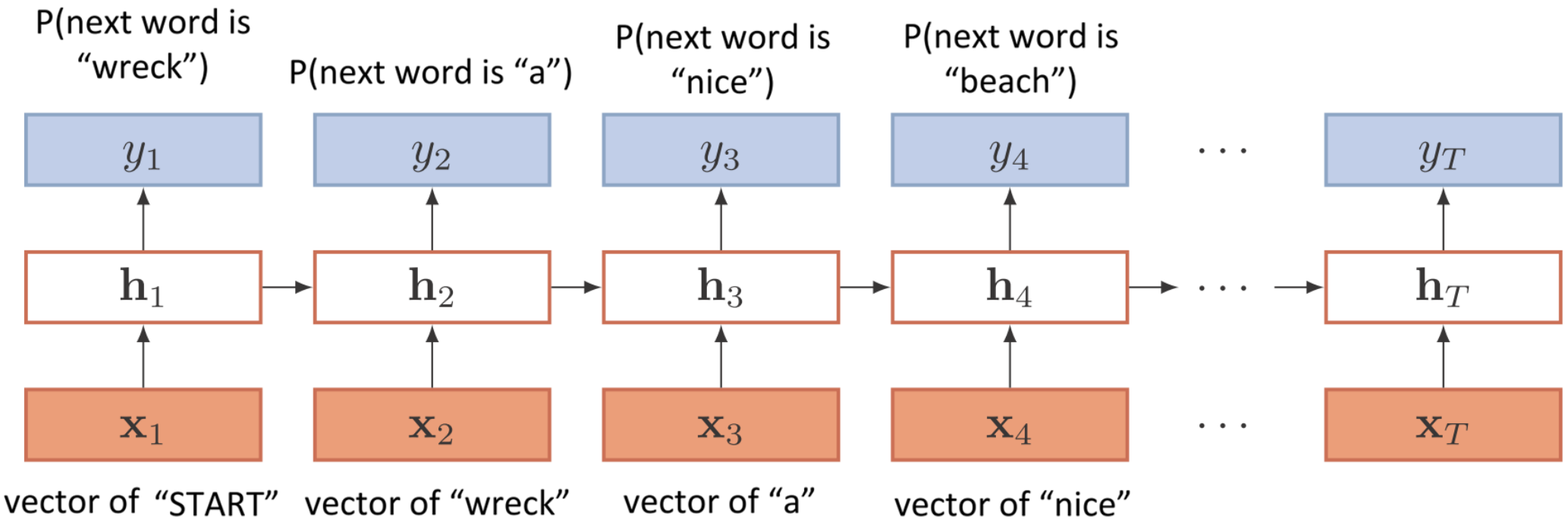



$$\hat{y} = g\left(\frac{1}{T}\sum_{t=1}^{T} h_t\right)$$

# 循环神经网络

- 序列到序列（同步）

$$x = (x_1, \cdots, x_T) \longrightarrow y = (y_1, \cdots, y_T)$$

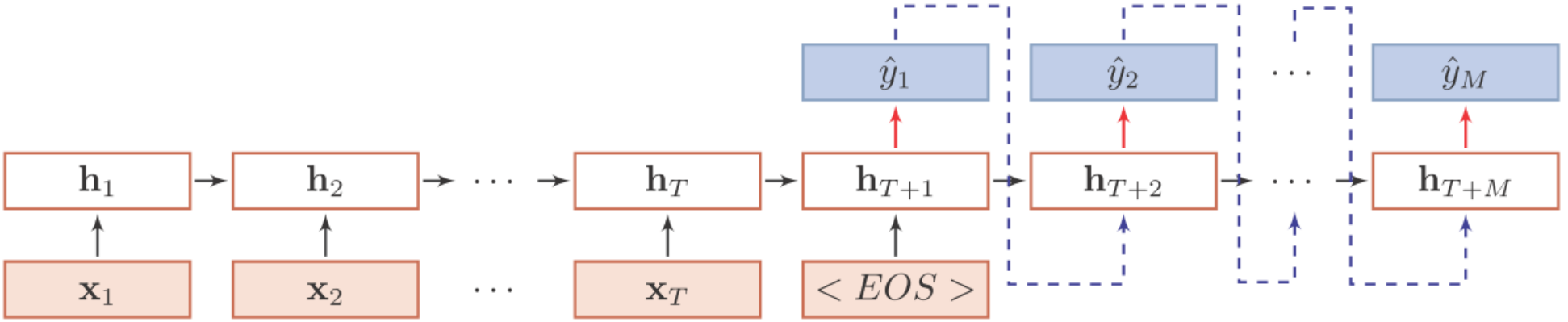


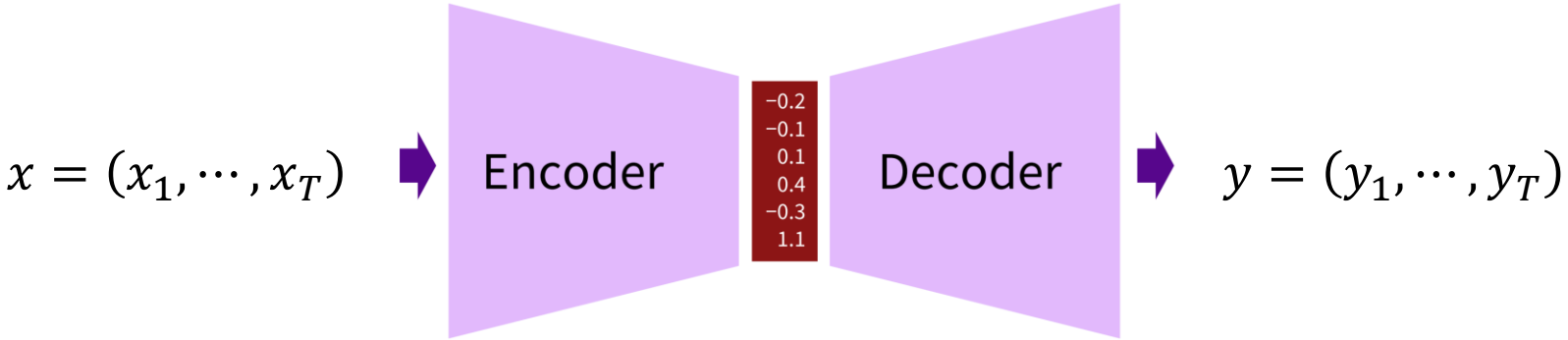

$$\hat{y}_t = g(h_t), \quad \forall t \in [1, T]$$

# 循环神经网络

- 序列到序列（异步）

$$x = (x_1, \cdots, x_T) \longrightarrow y = (y_1, \cdots, y_T)$$

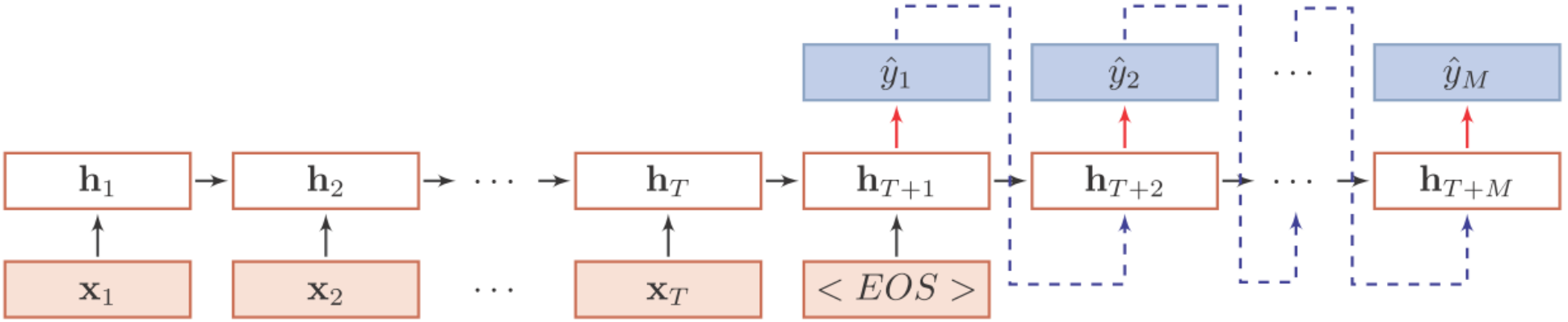


# 循环神经网络

- 序列到序列（异步）

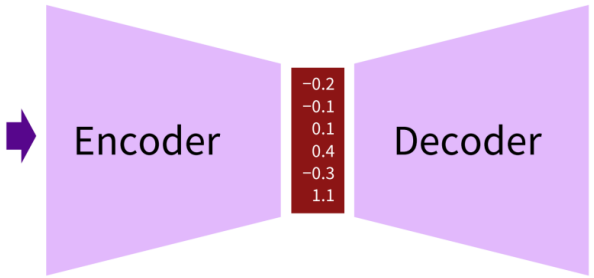
$$x = (x_1, \cdots, x_T) \longrightarrow y = (y_1, \cdots, y_T)$$




$$h_t = f_1(h_{t-1}, x_t), \forall t \in [1, T]$$

$$h_{T+t} = f_2(h_{T+t-1}, \hat{\boldsymbol{y}}_{t-1})$$
$$\hat{y}_t = g(h_{T+t}), \forall t \in [1, M]$$

$$x = (x_1, \cdots, x_T)$$

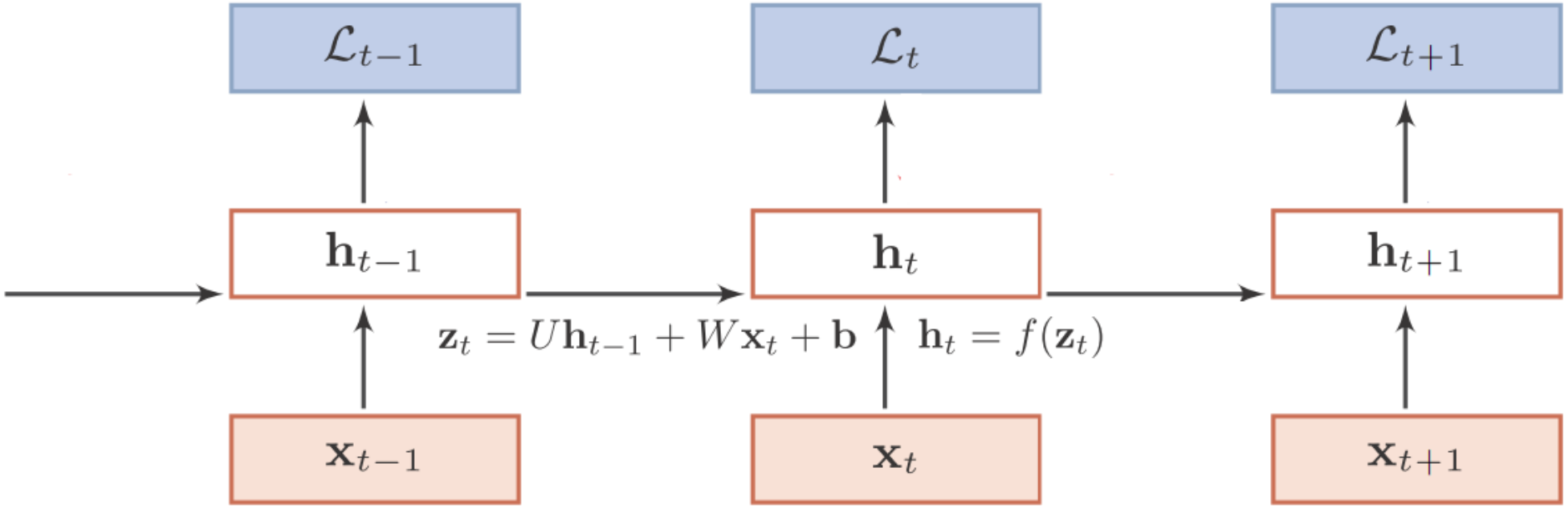



$$y = (y_1, \cdots, y_T)$$

# 循环神经网络

- 基于门控制的循环神经网络

短期记忆与长期记忆




简单循环神经网络中的隐状态 h 存储了历史信息，可以看作是一种记忆。在简单循环网络中，隐状态每个时刻都会被重写，因此可以看作是一种短期记忆 Short-Term Memory

# 循环神经网络

- 基于门控制的循环神经网络

短期记忆与长期记忆

$$\tilde{h}_t = \tanh(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h)$$

$r_t \in [0,1]$为重置门 *Reset Gate*，用来控制候选状态$\tilde{h}_t$ 的计算是否依赖上一时刻的状态$h_{t-1}$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r)$$

当重置门为$= 0$ 时，候选状态只和当前输入相关，和历史状态无关。
当重置门为$= 1$ 时，候选状态只和当前输入相关和历史状态相关。

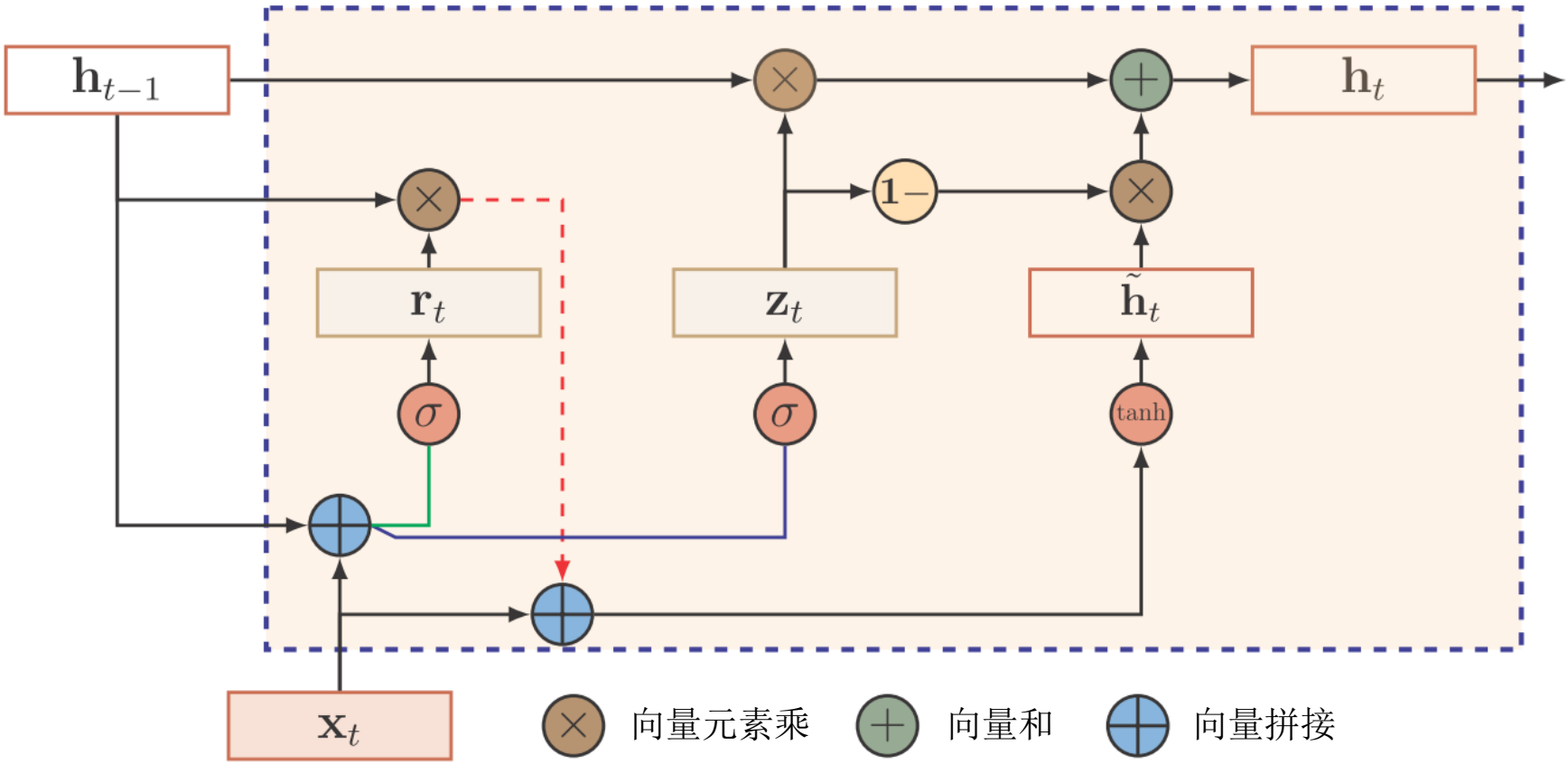
$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t$$

$z_t \in [0,1]$为更新门 *Update Gate*，用来控制着输入和遗忘之间的平衡。

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$

# 循环神经网络

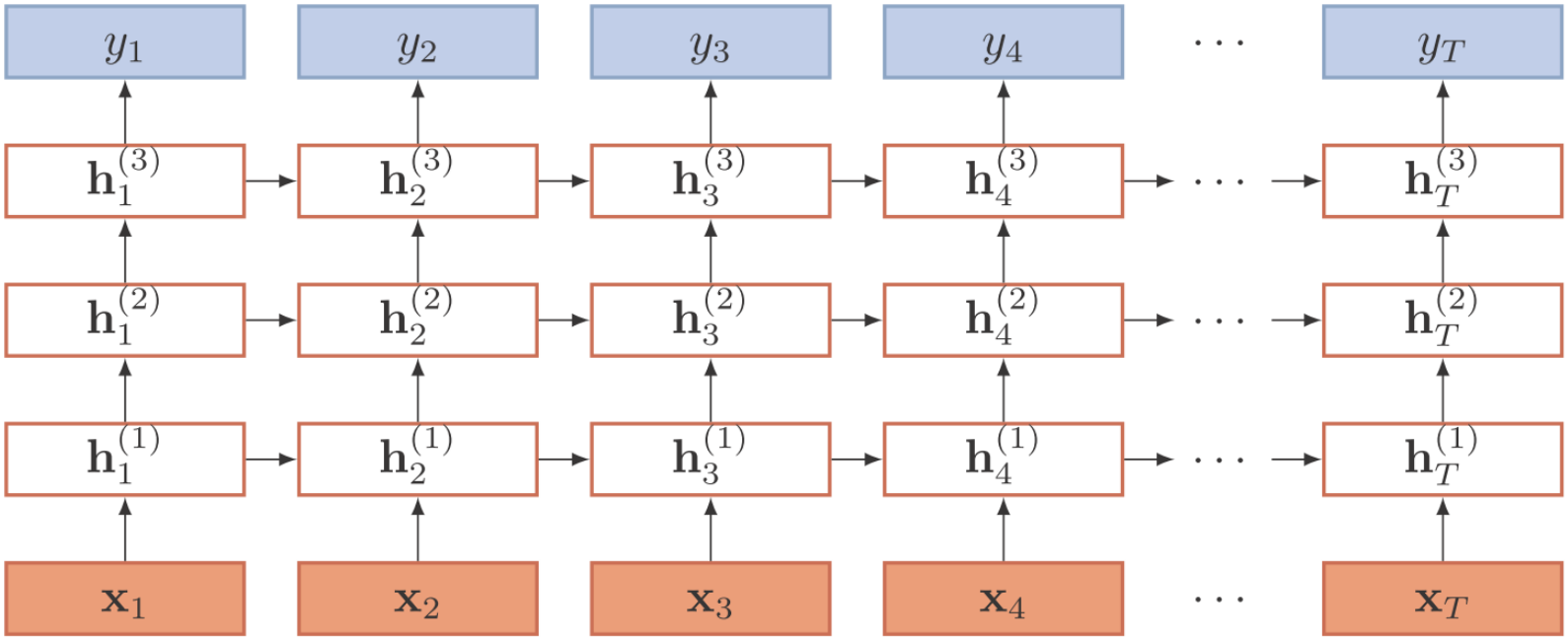
- 基于门控制的循环神经网络



# 循环神经网络

- 深层循环神经网络

> 堆叠循环神经网络

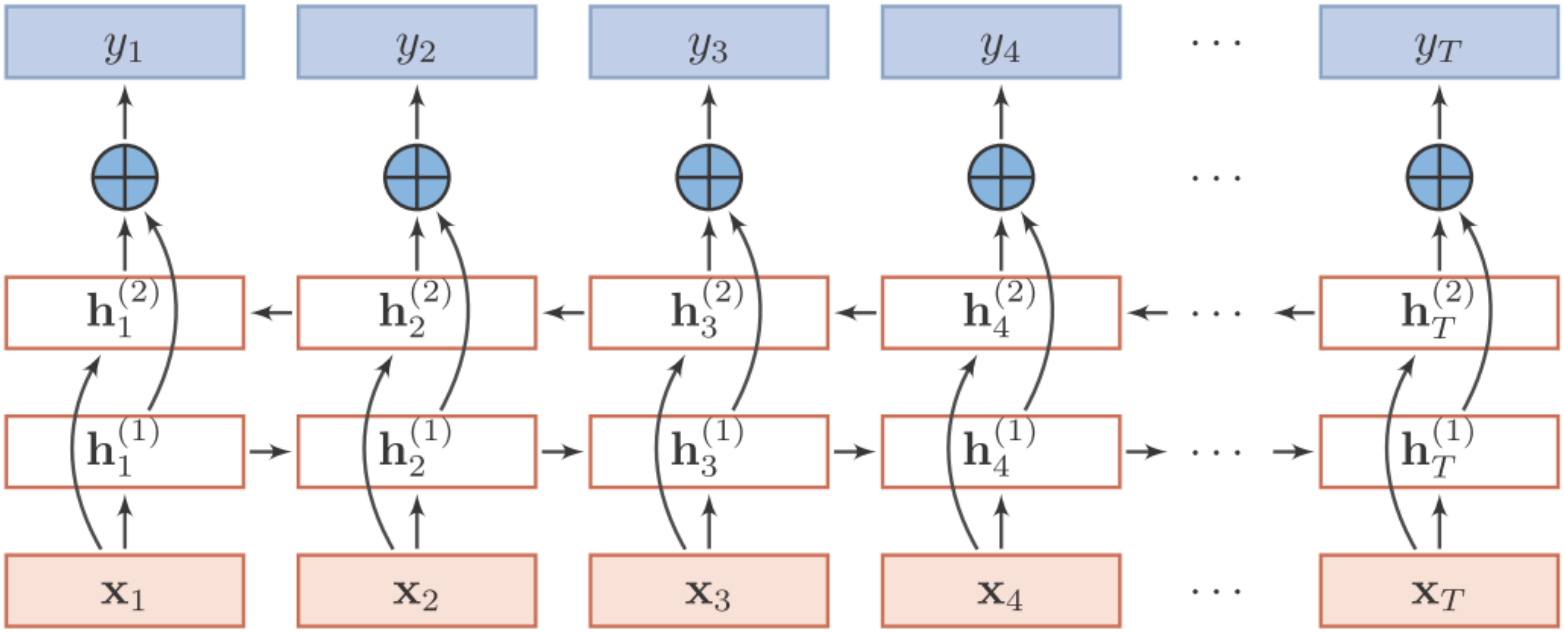



$$h_t^{(l)} = f\left(U^{(l)} h_{t-1}^{(l)} + W^{(l)} h_t^{(l-1)} + b^{(l)}\right)$$

# 循环神经网络

- 深层循环神经网络

> 双向循环神经网络

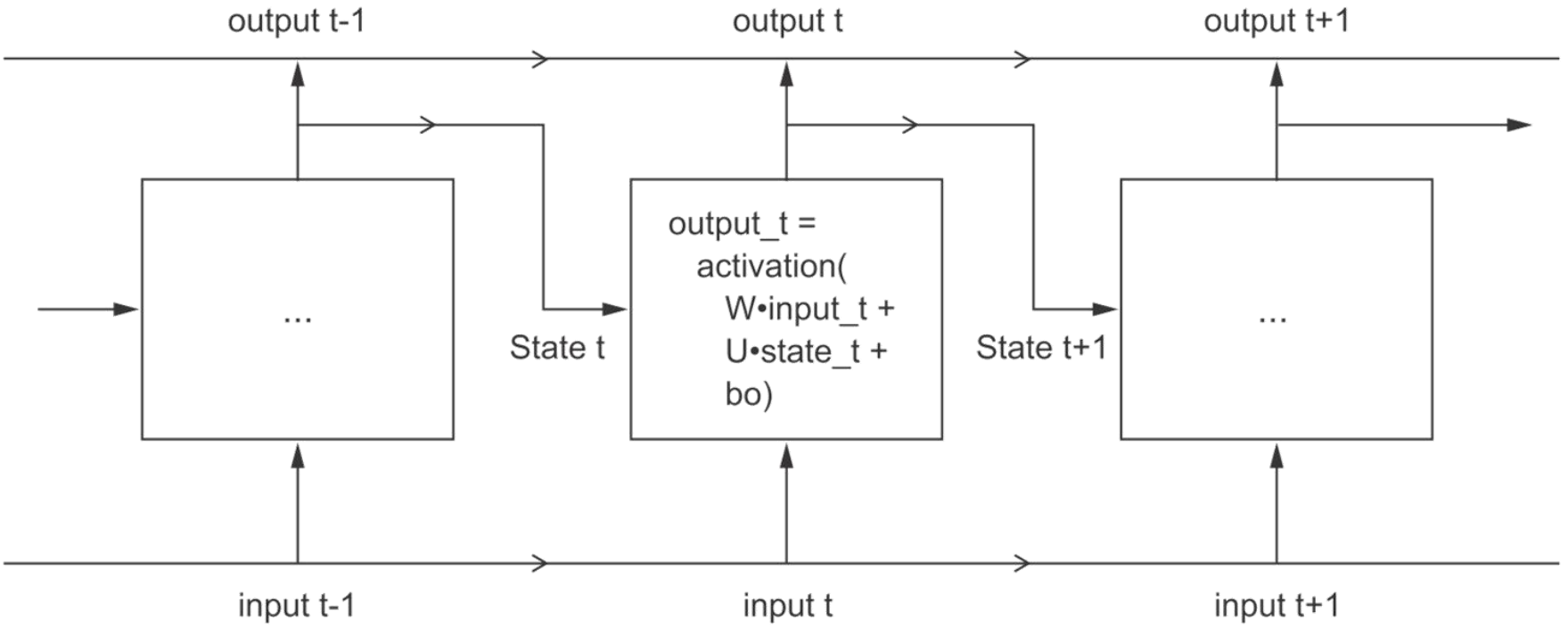


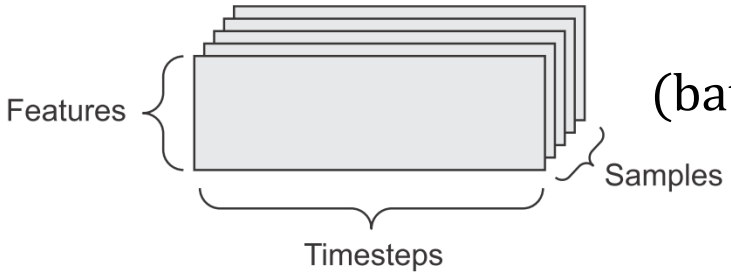

$$h_t^{(1)} = f\left(U^{(1)}h_{t-1}^{(1)} + W^{(1)}x_t + b^{(1)}\right)$$
$$h_t^{(2)} = f\left(U^{(2)}h_{t-1}^{(2)} + W^{(2)}x_t + b^{(2)}\right)$$
$$h_t = h_t^{(1)} \oplus h_t^{(2)}$$

# 深度学习实践

- RNN layer in Keras

```
from keras.layers import SimpleRNN
```




(batch_size, timesteps, input_features)