MACHINE LEARNING
机器学习
Neural Networks

神经网络

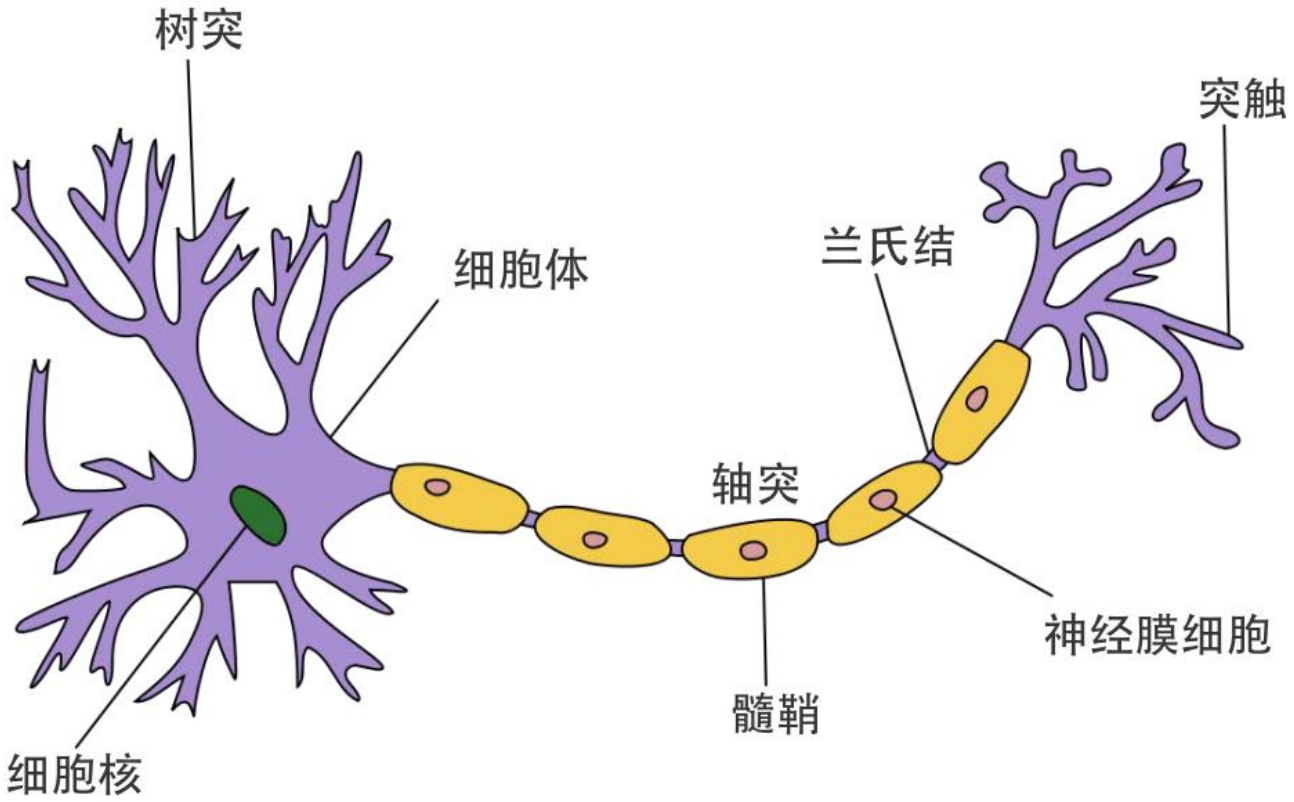
参考：
《神经网络与深度学习》

Machine Learning Course

发展历史

- 神经元

每个神经元与其他神经元相连，当它“兴奋”时，就会向相连的神经元发送化学物质，从而改变这些神经元内的电位； 如果某神经元的电位超过了一个“阈值” *threshold*, 那么它就会被激活，即“兴奋” 起来，向其他神经元发送化学物质.

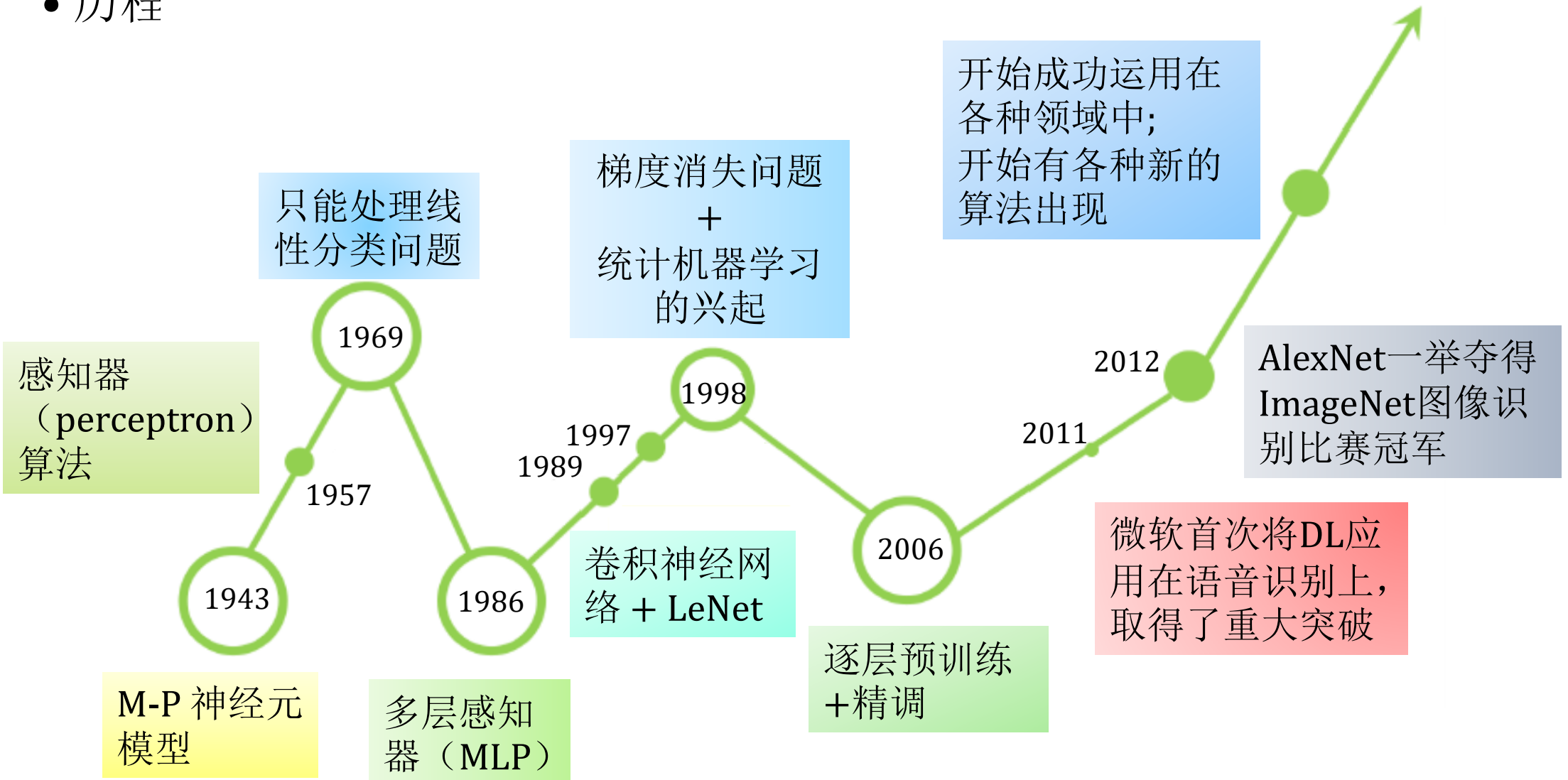


树突和细胞体的表膜都有接受刺激的功能

轴突把从树突和细胞表面传入细胞体的神经冲动传出到其他神经元或效应器

发展历史

- 历程



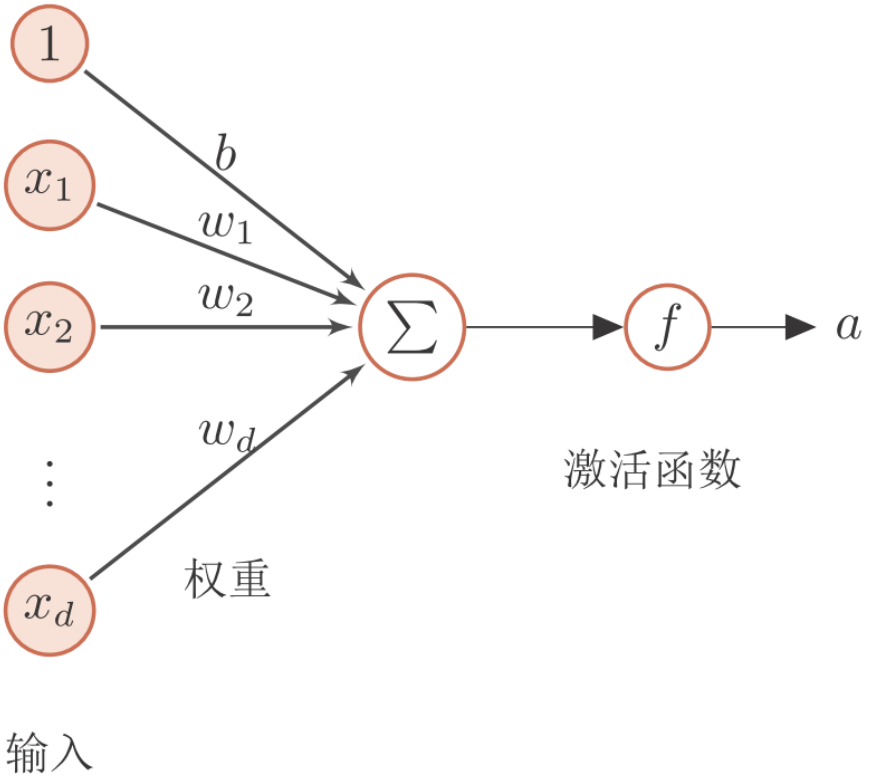
(深度学习大讲堂)

发展历史

- M-P 神经元模型

[McCulloch and Pitts, 1943] 设计了“ M-P 神经元模型”来模拟生物神经系统的神经元功能. 在这个模型中，神经元接收到来自 N 个其他神经元传递过来的输入信号，这些输入信号通过带权重的连接进行传递，神经元接收到的总输入值将与神经元的阈值 *threshold* 进行比较，然后通过“激活函数” *activation function* 处理以产生神经元的输出。

★ 阈值也可以刻画成符号相反的偏置 *bias*。



★ 阶跃型激活函数

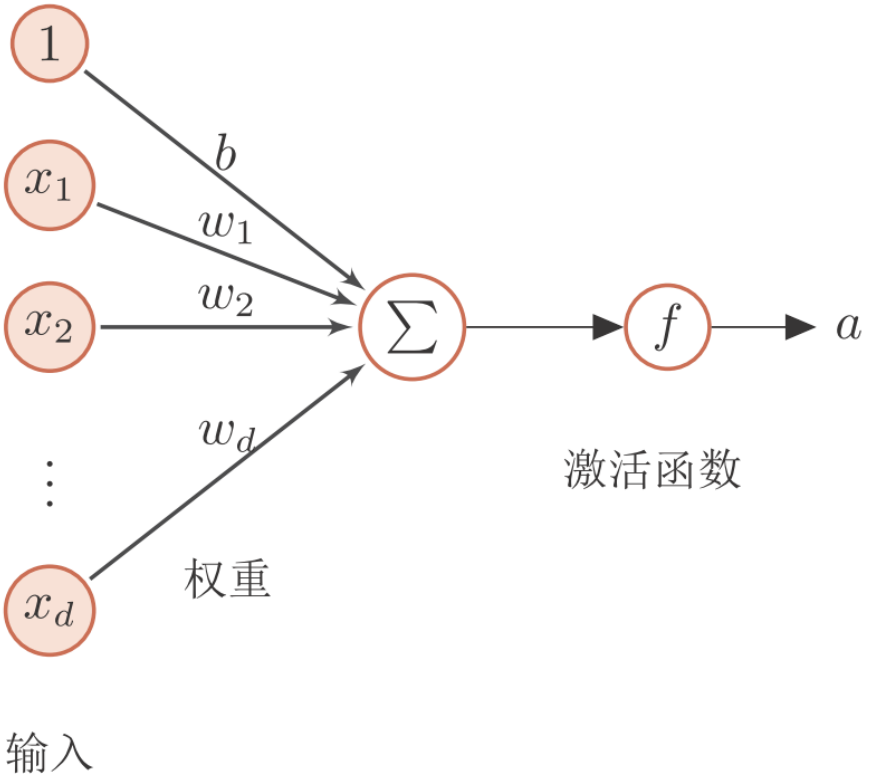
$$\sigma(\boldsymbol{w}^{\top}\boldsymbol{x}+b)=\begin{cases}\boldsymbol{w}^{\top}\boldsymbol{x}+b\geq 0\\ \boldsymbol{w}^{\top}\boldsymbol{x}+b<0\end{cases}$$

发展历史

- 激活函数

激活函数在神经元中非常重要的。为了增强网络的表示能力和学习能力，激活函数需要具备以下几点性质：

- 连续并可导（允许少数点上不可导）的非线性函数。
- 激活函数及其导函数要尽可能的简单
- 激活函数的导函数的值域要在一个合适的区间内，不能太大也不能太小



★ Sigmoid 型激活函数

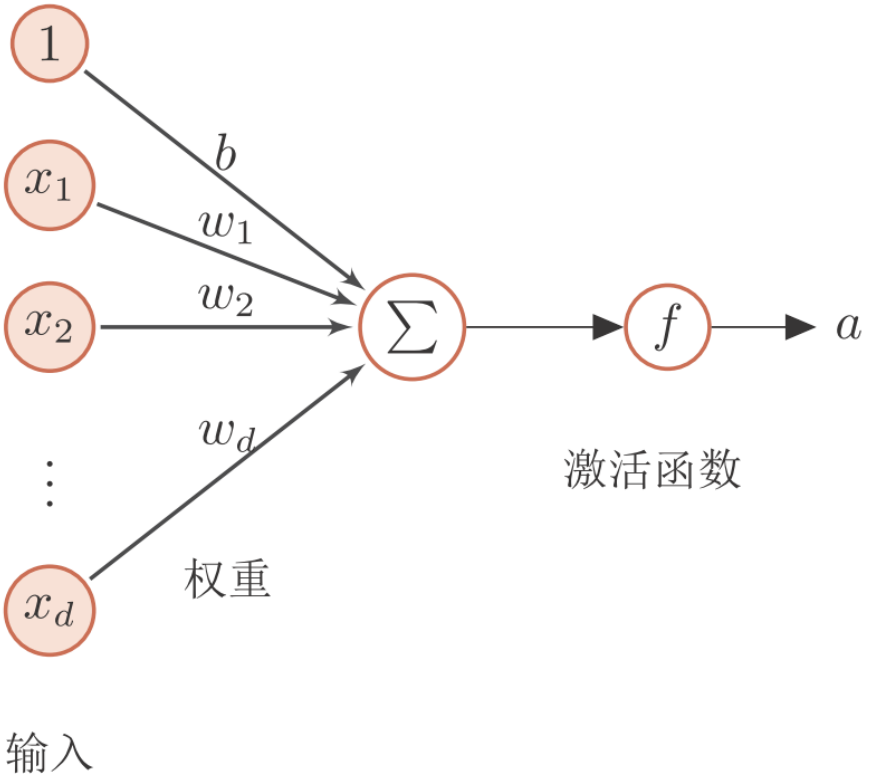
$$\sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b)=\frac{1}{1+\exp(-(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b))}$$

发展历史

- 激活函数

激活函数在神经元中非常重要的。为了增强网络的表示能力和学习能力，激活函数需要具备以下几点性质：

- 连续并可导（允许少数点上不可导）的非线性函数。
- 激活函数及其导函数要尽可能的简单
- 激活函数的导函数的值域要在一个合适的区间内，不能太大也不能太小



★ 双曲正切型激活函数

$$\tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)}$$

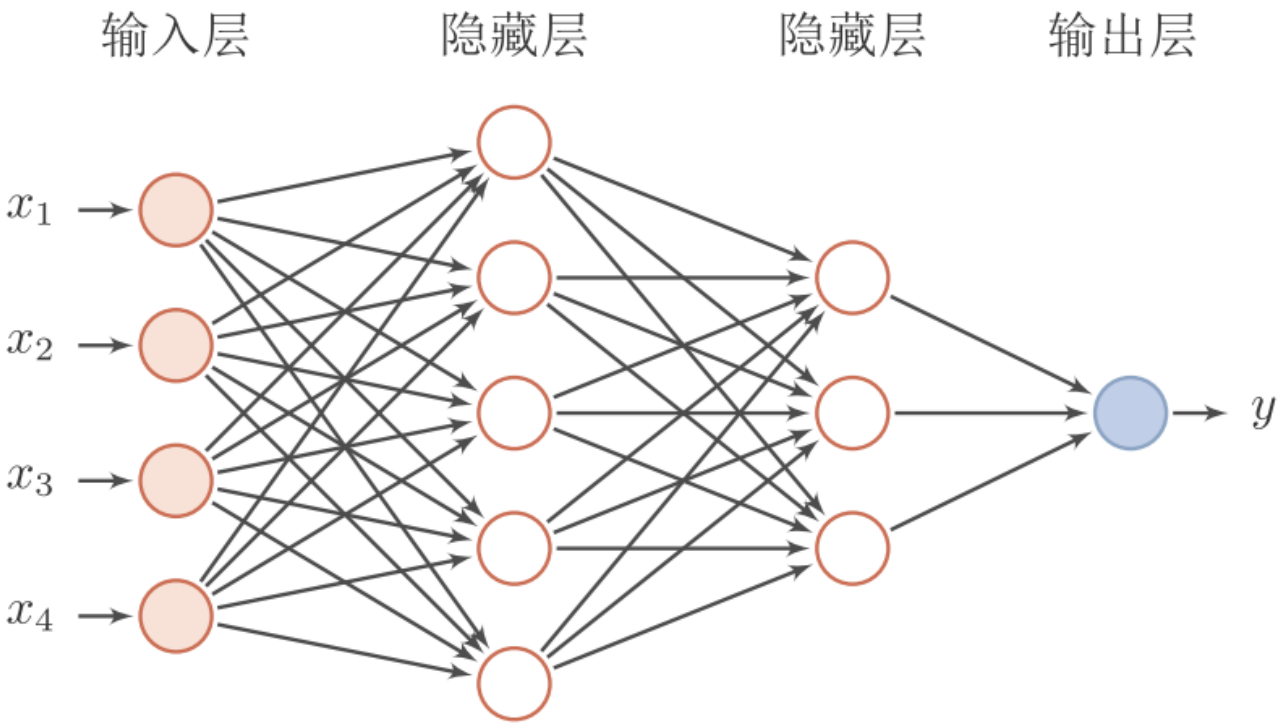
★ 修正线性单元（Rectified Linear Unit，ReLU）

$$\mathrm{ReLU}(z) = \max(0, z)$$

前馈神经网络

- 特点

人工神经网络的真正魅力在于它将多个神经元结构构成网络，从而获得强大的表示能力。
根据网络的结构特性，神经网络有多种类型。较流行的是前馈神经网络。
前馈神经网络中，各神经元从输入层开始，接收前一级输入，并输入到下一级，直至输出层。整个网络中无反馈，可用一个有向无环图表示。



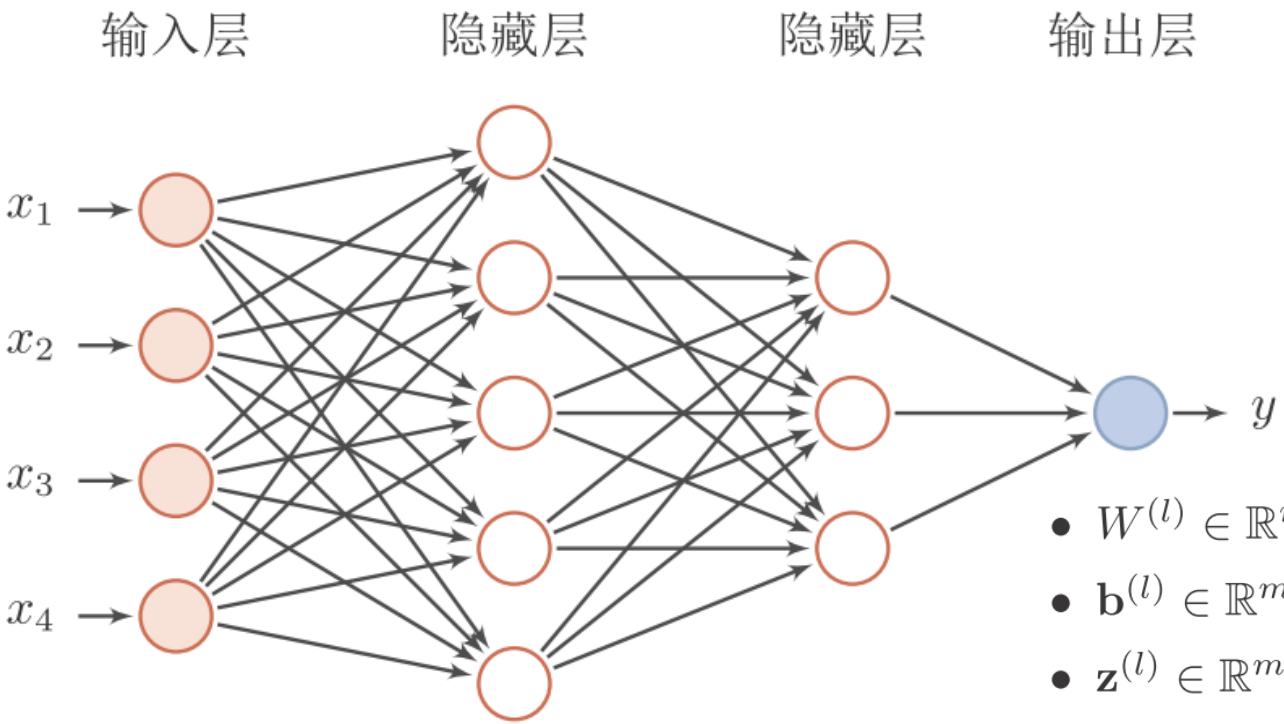
前馈神经网络

- 作用

神经网络学习了一套数据从原始到抽象的变换过程

$$\mathbf{z}^{(l)} = W^{(l)} \cdot \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)} \qquad \mathbf{a}^{(l)} = f_l(\mathbf{z}^{(l)})$$

$$\mathbf{x} = \mathbf{a}^{(0)} \to \mathbf{z}^{(1)} \to \mathbf{a}^{(1)} \to \mathbf{z}^{(2)} \to \cdots \to \mathbf{a}^{(L-1)} \to \mathbf{z}^{(L)} \to \mathbf{a}^{(L)} = \varphi(\mathbf{x}; W, \mathbf{b}))$$



- L：表示神经网络的层数；
- $m^{(l)}$：表示第 l 层神经元的个数；
- $f_l(\cdot)$：表示 l 层神经元的激活函数；
- $W^{(l)} \in \mathbb{R}^{m^{(l)} \times m^{l-1}}$：表示 $l-1$ 层到第 l 层的权重矩阵；
- $\mathbf{b}^{(l)} \in \mathbb{R}^{m^l}$：表示 $l-1$ 层到第 l 层的偏置；
- $\mathbf{z}^{(l)} \in \mathbb{R}^{m^l}$：表示 l 层神经元的净输入（净活性值）；
- $\mathbf{a}^{(l)} \in \mathbb{R}^{m^l}$：表示 l 层神经元的输出（活性值）。

前馈神经网络

- 如何学习：基于误差反向传播的学习方法

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial w_{ij}^{(l)}} = \frac{\partial \mathbf{z}^{(l)}}{\partial w_{ij}^{(l)}} \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}}$$

$$\frac{\partial \mathbf{z}^{(l)}}{\partial w_{ij}^{(l)}} = \left[\frac{\partial z_1^{(l)}}{\partial w_{ij}^{(l)}}, \cdots, \frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l)}}, \cdots, \frac{\partial z_{m^{(l)}}^{(l)}}{\partial w_{ij}^{(l)}} \right] \quad (\mathbf{z}^{(l)} = W^{(l)} \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)})$$

$$= \left[0, \cdots, \frac{\partial (\mathbf{w}_{i:}^{(l)} \mathbf{a}^{(l-1)} + b_i^{(l)})}{\partial w_{ij}^{(l)}}, \cdots, 0 \right]$$

$$= \left[0, \cdots, a_j^{(l-1)}, \cdots, 0 \right]$$

前馈神经网络

- 如何学习：基于误差反向传播的学习方法

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial w_{ij}^{(l)}} = \frac{\partial \mathbf{z}^{(l)}}{\partial w_{ij}^{(l)}} \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}}$$

$$\begin{aligned}
\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}} = \delta^{(l)} &= \frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} \cdot \frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{a}^{(l)}} \cdot \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l+1)}} \\
&= \mathrm{diag}(f_l'(\mathbf{z}^{(l)})) \cdot (W^{(l+1)})^{\mathrm{T}} \cdot \delta^{(l+1)} \\
&= f_l'(\mathbf{z}^{(l)}) \odot \left((W^{(l+1)})^{\mathrm{T}} \delta^{(l+1)}\right)
\end{aligned}$$

$\odot$：点积运算符，表示每个元素相乘

前馈神经网络

- 如何学习：基于误差反向传播的学习方法

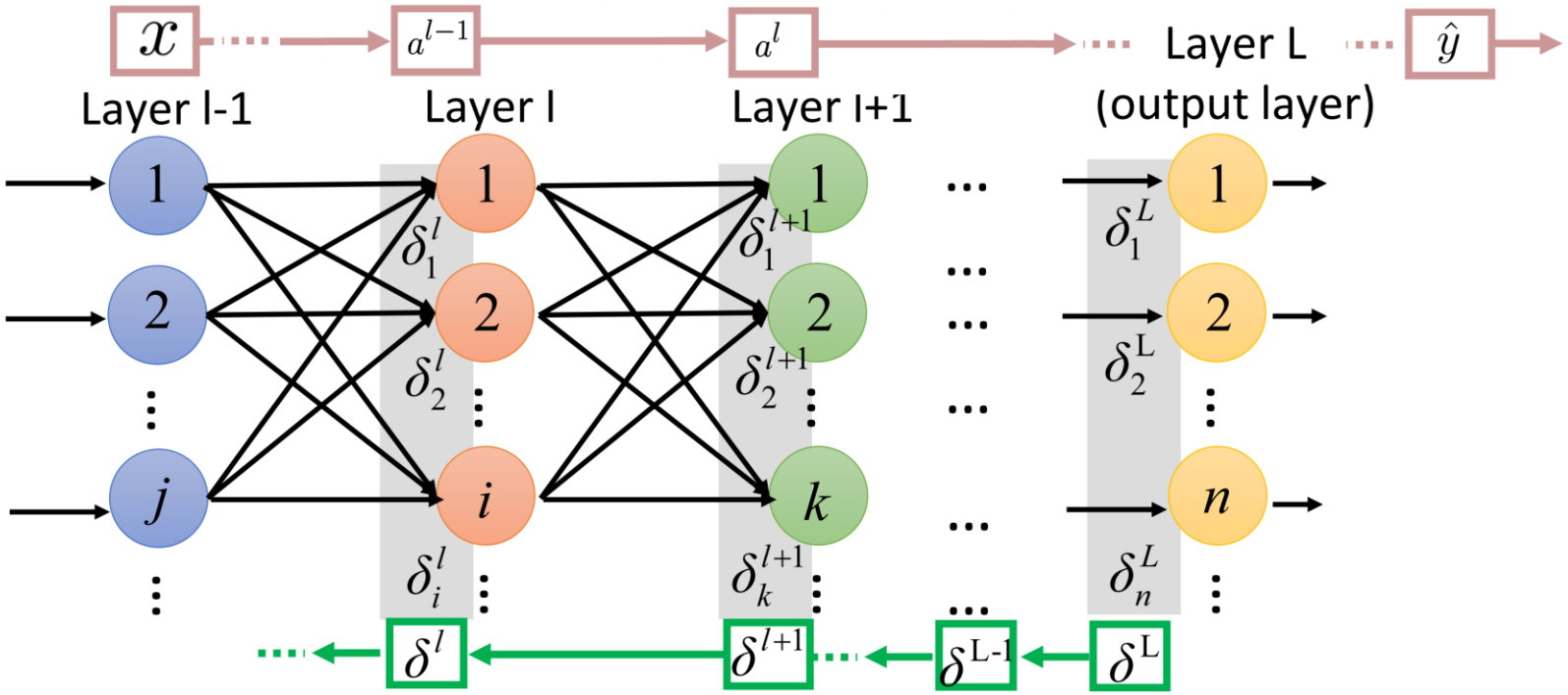
$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{b}^{(l)}} = \frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}} \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}}$$

$$\frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}} = \mathbf{I}_{m^{(l)}} \quad (\ \mathbf{z}^{(l)} = W^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}\ )$$

$$\delta^{(l)} = \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}}$$

前馈神经网络

- 如何学习：基于误差反向传播的学习方法



1. 前馈计算每一层的净输入 $\mathbf{z}^{(l)}$ 和激活值 $\mathbf{a}^{(l)}$，直到最后一层；
2. 反向传播计算每一层的误差项 $\delta^{(l)}$；
3. 计算每一层参数的偏导数，并更新参数。

前馈神经网络

- 如何学习：更多学习细节

■ 权重初始化

假设有个神经元有n个输入权重。常会使用均值为0, 标准差为1/sqrt(n)的高斯随机分布初始化这些权重。

■ Mini-batch方式

每次随机选取一部分的训练样本，称它们为一个“小批量” mini-batch 数据。最小化在它们上面的累积误差。当每次用完了所有的训练样本，则称完成了一个训练迭代周期 epoch。

■ 优化技术

SGD 基于Hessian和momentum技术上的改进

L-BFGS（受限的BFGS）

CG（共轭梯度法）

前馈神经网络

- 如何学习：更多学习细节
  - 早停 *early stopping*

隔出验证集，若训练误差降低但泛化误差升高，则停止训练。并返回具有最小验证集误差的权重和阈值。

  - 丢弃技术 *dropout*

dropout是指在深度学习网络的训练过程中，对于神经网络单元，按照一定的概率将其暂时从网络中丢弃。注意是暂时，对于随机梯度下降来说，由于是随机丢弃，故而每一个mini-batch都在训练不同的网络。

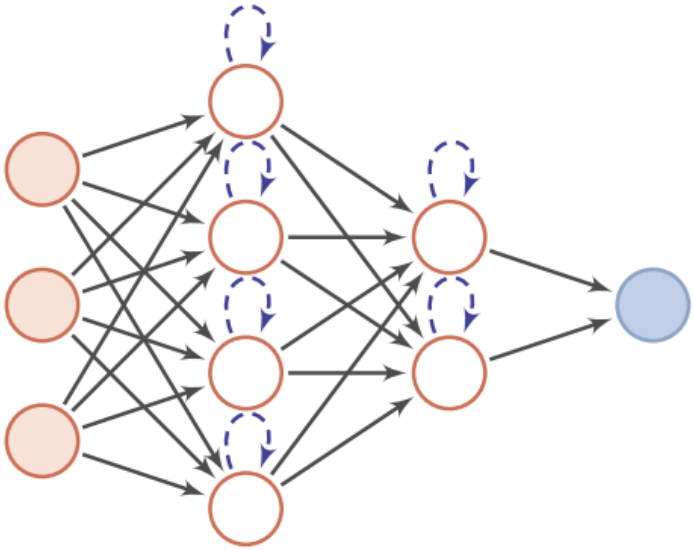
  - 数据增强 *data augmentation*

通过应用反映真实世界变化的操作（例如通过增加背景噪声）来扩展训练数据

其他网络结构

- 记忆网络

记忆网络，也称为反馈网络，网络中的神经元不但可以接收其它神经元的信息，也可以接收自己的历史信息。和前馈网络相比，记忆网络中的神经元具有记忆功能，在不同的时刻具有不同的状态。

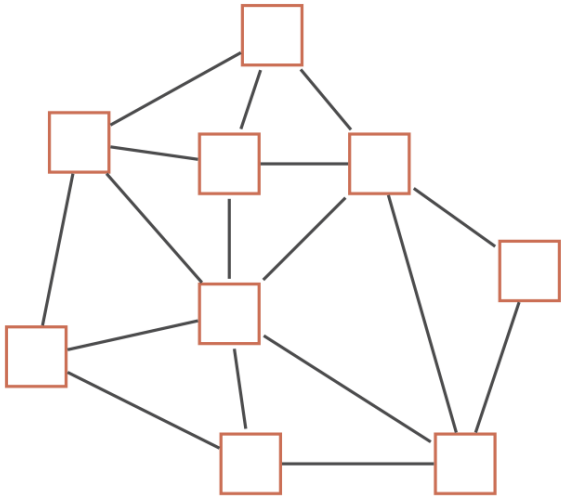


记忆神经网络中的信息传播可以是单向或双向传递，因此可用一个有向循环图或无向图来表示。

其他网络结构

- 图网络

前馈网络和记忆网络的输入都可以表示为向量或向量序列。图网络是定义在图结构数据上的神经网络。图中每个节点都由一个或一组神经元构成。节点之间的连接可以是有向的，也可以是无向的。每个节点可以收到来自相邻节点或自身的信息。



MACHINE LEARNING

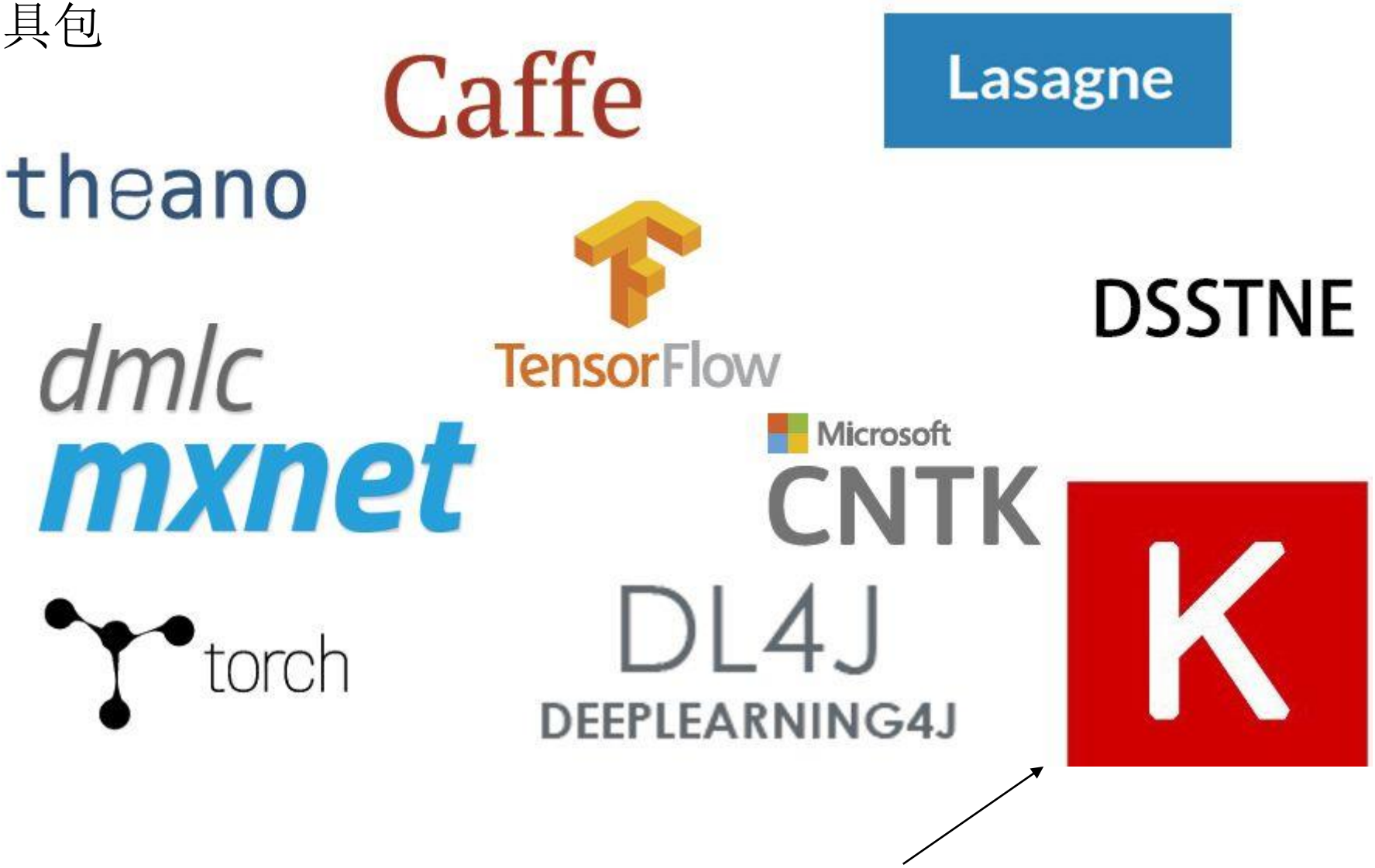
实践

Practice

参考：
《Python深度学习》

神经网络实践

- 工具包

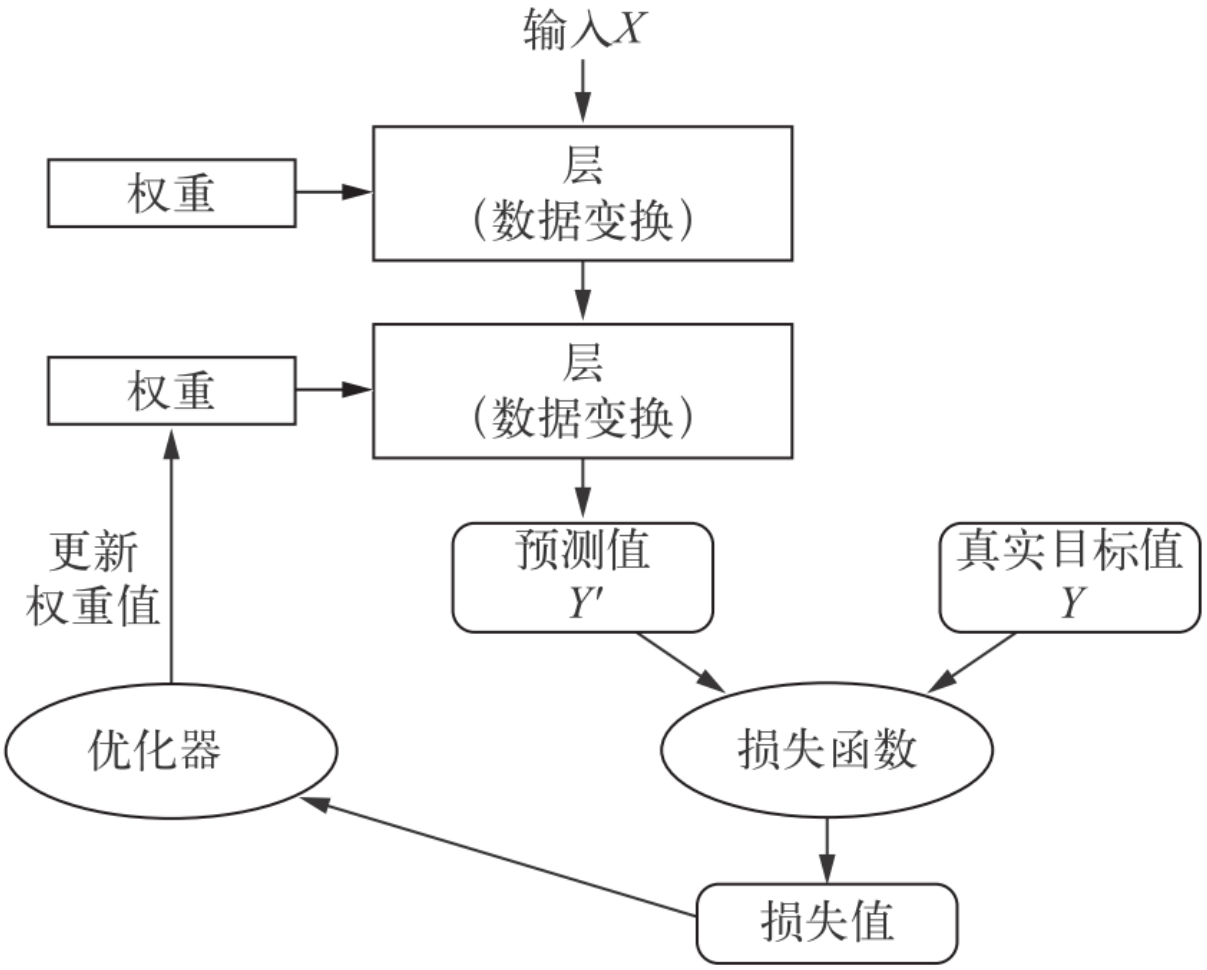


最简单易用的工具包之一

神经网络实践

- 要素

- 数据
- 层
- 结构
- 损失函数
- 优化器



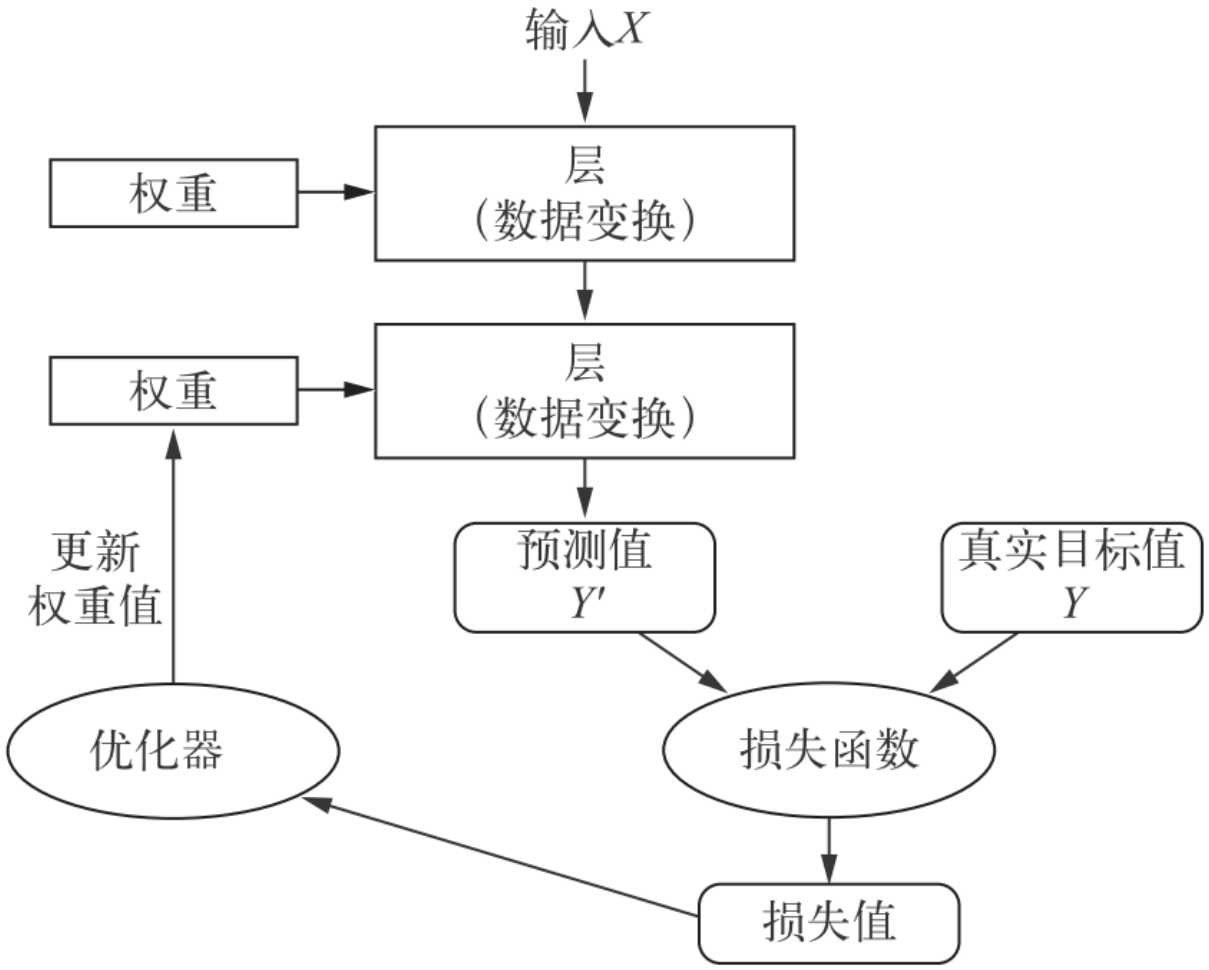
神经网络实践

- 步骤

- 准备数据
- 定义由层+结构刻画的网络
- 设定学习细节：

1 损失函数
2 优化方法
3 评估指标

- 网络学习
- 网络存储
- 网络使用



神经网络实践

- 示例：多层前馈神经网络
  - 准备数据

Loading the Boston housing dataset

```
from keras.datasets import boston_housing

(train_data, train_targets), (test_data, test_targets) =
boston_housing.load_data()

print(train_data.shape)
```

Normalizing the data

```
mean = train_data.mean(axis=0)
train_data -= mean
std = train_data.std(axis=0)
train_data /= std

test_data -= mean
test_data /= std
```

神经网络实践

- 示例：多层前馈神经网络
  - 定义网络

Model definition

```
from keras import models
from keras import layers
def build_model():
    model = models.Sequential()
    model.add(layers.Dense(64, activation='relu',

input_shape=(train_data.shape[1],)))
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(1))
    model.compile(optimizer='rmsprop', loss='mse',
metrics=['mae'])
    return model
```

神经网络实践

- 示例：多层前馈神经网络
  - 查看网络

Visual neural network

```
model = build_model()
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 64)                896
_________________________________________________________________
dense_2 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 65
=================================================================
Total params: 5,121
Trainable params: 5,121
Non-trainable params: 0
```

神经网络实践

- 示例：多层前馈神经网络
  - 网络学习与使用

Training and test

```
model.fit(train_data, train_targets,
          epochs=80, batch_size=16, verbose=0)
test_mse_score, test_mae_score = model.evaluate(test_data,
test_targets)
print(test_mae_score)
```

```
 32/102 [========>.....................] - ETA: 0s
102/102 [==============================] - 0s 127us/step
2.64116473291
```

神经网络实践

- 示例：多层前馈神经网络
  - 网络学习与使用

网络存储与复用

```
model.save('simpprog_model.h5')
model = load_model('simpprog_model.h5')
model.summary()
from keras.models import Model
dense1_layer_model =
Model(inputs=model.input,outputs=model.get_layer('dense_1').
output)
test_feas_dense1 = dense1_layer_model.predict(test_data)
print(test_feas_dense1.shape)
print(test_feas_dense1[0])
```