

### 3 深度学习(Deep Learning)

#### 3.1 神经网络(Neural Networks)

#### 3.2 卷积神经网络(Convolutional Neural Networks)

#### 3.3 递归神经网络(Recurrent Neural Networks)

#### 3.4 强化学习与控制(Reinforcement Learning and Control)

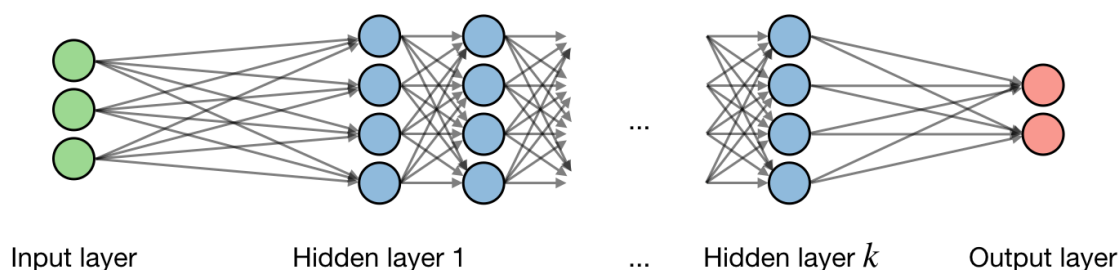
## 3 深度学习(Deep Learning)

### 3.1 神经网络(Neural Networks)

神经网络是一类用层构建的模型。常用的神经网络类型包括卷积神经网络和递归神经网络。

- 结构

关于神经网络架构的描述如下图所示：



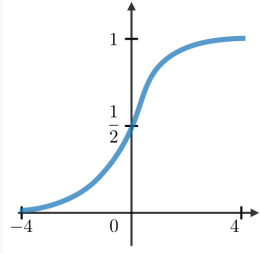
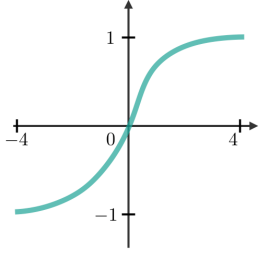
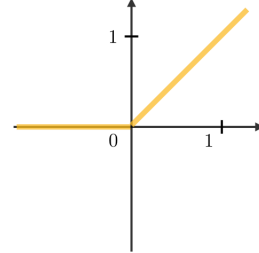
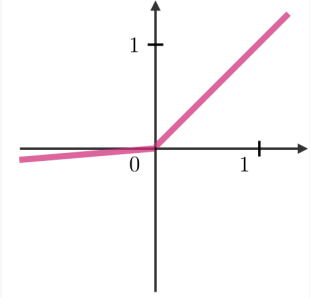
记  $i$  为网络的第  $i$  层,  $j$  为一层中隐藏的第  $j$  个单元, 得到：

$$z_j^{[i]} = \omega_j^{(i)T} x + b_j^{([i])}$$

式中  $\omega, b, z$  分别表示权重, 偏移和输出。

- 激活函数

在隐含单元的末端使用激活函数向模型引入非线性复杂性。以下是最常见的几种:

Sigmoid	Tanh	ReLU	Leaky ReLU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$	$g(z) = \max(\varepsilon z, z), \varepsilon \ll 1$
			

- 交叉熵损失(Cross-entropy loss)

在神经网络中，交叉熵损失  $L(z, y)$  是常用的，定义如下：

$$L(z, y) = -[y \log(z) + (1 - y) \log(1 - z)]$$

- 学习率(Learning rate)

学习率通常记作  $\eta$ ，表示在哪一步权重得到了更新。这个可以使固定的，也可以是自适应变化的。目前最流行的方法是 Adam，这是一种自适应学习率的方法。

- 反向传播(Backpropagation)

反向传播是一种通过考虑实际输出和期望输出更新神经网络权重的方法。权重  $\omega$  的导数用链式法则计算(chain rule)，它的形式如下：

$$\frac{\partial L(z, y)}{\partial \omega} = \frac{\partial L(z, y)}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial \omega}$$

因此权重更新如下：

$$\omega \leftarrow \omega - \eta \frac{\partial L(z, y)}{\partial \omega}$$

- 更新权重

在神经网络中，权重的更新方式如下：

第一步：对训练数据取一批(batch)；第二步：进行正向传播以获得相应的损失；第三步：反向传播损失，得到梯度；第四步：使用梯度更新网络的权重。

- 丢弃(Dropout)

它是一种通过在神经网络中删除单元来防止过度拟合训练数据的技术。实际应用中，单元被删除的概率是  $p$ ，或被保留的概率是  $1 - p$ 。

## 3.2 卷积神经网络(Convolutional Neural Networks)

- 卷积层需求

记  $W$  为输入量大小,  $F$  为卷积层神经元大小,  $P$  为 zero padding 数量, 那么匹配给定体积输入的神经元数量  $N$  为:

$$N = \frac{W - F + 2P}{S} + 1$$

- 批量正则化(Batch normalization)

这一步是超参数(hyperparameter)  $\gamma, \beta$  正则化批量  $\{x_i\}$ 。记  $\mu_B, \sigma_B^2$  分别为批量值的平均值和方差, 正则化表示如下:

$$x_i \leftarrow \gamma \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta$$

它通常用于完全连接或卷积层之后, 且在非线性层之前。目的是允许更高的学习率, 减少初始化的强依赖。

### 3.3 递归神经网络(Recurrent Neural Networks)

- 门类型(Types of gates)

以下是在我们碰到的典型递归神经网络中存在的不同类型的门:

输入门(Input gate)	忘记门(Forget gate)	输出门(Output gate)	门(Gate)
是否写入神经元?	是否擦出神经元?	是否显示神经元?	写入多少

- 长短期记忆网络(LSTM, Long Short-Term Memory)

长短期记忆网络是RNN模型的一种, 它通过添加“忘记”门来避免梯度消失问题。

### 3.4 强化学习与控制(Reinforcement Learning and Control)

强化学习的目标是让代理(agent)学会如何在环境中进化。

- 马尔科夫决策过程(Markov decision processes)

马尔科夫决策过程(MDP)是一个5元组  $(S, A, \{P_{sa}\}, \gamma, R)$ , 其中:

$S$  是一组状态。

$A$  是一组行为。

$\{P_{sa}\}$ , 是  $s \in S$  和  $a \in A$  的状态转换概率。

$\gamma \in [0, 1]$  是discount系数。

$R: S \times A \rightarrow \mathbb{R}$  或者  $R: S \rightarrow \mathbb{R}$  是算法要最大化的奖励函数。

- 策略(Policy)

策略  $\pi$  是一个映射状态到行为的函数  $\pi: S \rightarrow A$ 。

备注: 我们说, 如果给定一个状态  $s$ , 我们执行一个给定的策略  $\pi$ , 得到的行为是  $a = \pi(s)$ 。

- 价值函数(Value function)

对于给定的策略  $\pi$  和状态  $s$  , 我们定义价值函数如下  $V^\pi$  :

$$V^\pi = E [R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots | s_0 = s, \pi]$$

- 贝尔曼方程(Bellman equation)

最优贝尔曼方程描述了最优策略  $\pi^*$  的价值函数  $V^{\pi^*}$  :

$$V^{\pi^*} = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^{\pi^*}(s')$$

备注：对于给定的状态  $s$  , 我们记最优策略  $\pi^*$  为：

$$\pi^* = \arg \max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^{\pi^*}(s')$$

- 价值迭代算法(Value iteration algorithm)

算法包含2步：

第一步，初始化价值：

$$V_0(s) = 0$$

第二步，基于之前的驾驶进行迭代：

$$V_{i+1}(s) = R(s) + \max_{a \in A} \left[ \sum_{s' \in S} P_{sa}(s') V_i(s') \right]$$

- 最大似然估计(Maximum likelihood estimate)

状态转移概率的最大似然估计如下:

$$P_{sa}(s') = \frac{\text{状态 } s \text{ 到 } s' \text{ 行为 } a \text{ 的次数}}{\text{状态 } s \text{ 的行为 } a \text{ 次数}}$$

- Q-learning

Q-learning是  $Q$  一种无模型，公式如下：

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[ R(s, a, s') + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$$