

VIP Cheatsheet: Deep Learning

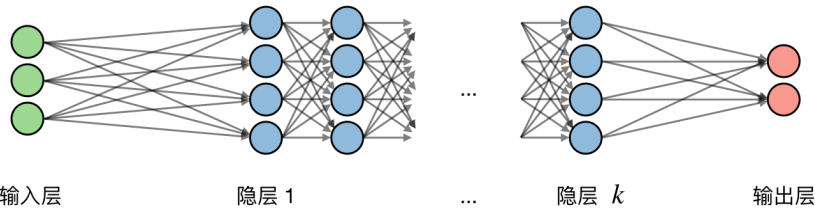
Afshine AMIDI and Shervine AMIDI

October 13, 2018

神经网络

神经网络是一类按层结构搭建的模型。常用的神经网络包括卷积神经网络和递归神经网络。

□ **架构** – 下表列举了用来描述神经网络架构的词汇：



已知 i 是网络的第 i 层, j 是网络的第 j 层, 我们有:

$$z_j^{[i]} = w_j^{[i]T} x + b_j^{[i]}$$

我们用 w, b, z 分别表示权重, 偏差和输出。

□ **激活函数** – 激活函数被用在隐含单元之后来向模型引入非线性复杂度。比较常见的如下所示:

逻辑函数(Sigmoid)	双曲正切函数(Tanh)	ReLU	Leaky ReLU
$g(z) = \frac{1}{1 + e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$

□ **交叉熵损失** – 在神经网络中, 交叉熵损失 $L(z, y)$ 通常如下定义:

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

□ **学习率** – 学习率, 通常记为 α 或 η , 表示权重更新的速度。它可以被修复或自适应改变。现阶段最常用的方法是一种调整学习率的算法, 叫做Adam。

□ **反向传播** – 反向传播是一种通过考虑实际输出和期望输出来更新神经网络中权重的方法。权重 w 的导数由链式法则计算, 模式如下:

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

作为结果, 权重更新如下:

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

□ **更新权重** – 在一个神经网络中, 权重如下所示更新:

- 第一步: 分出第一批次的训练数据。
- 第二步: 通过前向传播来得到相关损失。
- 第三步: 通过反向传播损失来得到梯度。
- 第四步: 利用梯度更新网络的权重。

□ **随机丢弃(Dropout)** – 随机丢弃是一种通过丢弃神经网络单元来防止训练数据过拟合的技术。实际上, 神经元以概率 p 被丢弃或以概率 $1 - p$ 被保留。

卷积神经网络

□ **卷积神经网络要求** – 记 W 为输入图像尺寸, F 为卷积层神经元尺寸, P 为零填充的大小, 那么给定的输入图像能够容纳的神经元数目 N 为:

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

□ **批量规范化** – 它是超参数 γ, β 标准化样本批 $\{x_i\}$ 的一个步骤。将我们希望修正这一批样本的均值和方差记作 μ_B, σ_B^2 , 会得到:

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

它通常在线性层之前和全连接/卷积层后应用, 目的是允许更高的学习率和减少初始化的强相关。

递归神经网络

□ **门控的种类** – 在一个典型的递归神经网络中有多种门控结构:

输入门	忘记门	门控	输出门
要不要写入单元?	要不要清空单元?	在单元写入多少?	从单元泄露多少?

□ **LSTM** – 长短期记忆网络是递归神经网络的一种, 它可以通过增加“遗忘”门控来避免梯度消失问题。

强化学习和控制

强化学习的目标是让代理学习如何在环境中进化。

□ **马尔可夫决策过程** – 一个马尔可夫决策过程(MDP)是一个5维元组 $(\mathcal{S}, \mathcal{A}, \{P_{sa}\}, \gamma, R)$ ，即：

- \mathcal{S} 是状态的集合
- \mathcal{A} 是动作的集合
- $\{P_{sa}\}$ 是对于 s 属于 \mathcal{S} 并且 a 属于 \mathcal{A} 的状态转换概率
- $\gamma \in [0, 1[$ 是折扣系数
- $R: \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ 或 $R: \mathcal{S} \rightarrow \mathbb{R}$ 是算法希望最大化的回报函数

□ **策略** – 策略 π 是映射状态到动作的 $\pi: \mathcal{S} \rightarrow \mathcal{A}$ 函数。

注意：如果对于一个指定的状态 s 我们完成了行动 $a = \pi(s)$ ，我们认为执行了一个指定的策略 π 。

□ **价值函数** – 对于一个指定的策略 π 和指定的状态 s ，我们定义如下价值函数 V^π ：

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

□ **贝尔曼方程** – 最优贝尔曼方程描述了最优策略 π^* 的价值方程 V^{π^*} ：

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

注意：我们注意到对于一个特定的状态 s 的最优策略 π^* 是：

$$\pi^*(s) = \operatorname{argmax}_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P_{sa}(s') V^*(s')$$

□ **值迭代算法** – 值迭代算法分为两步：

- 首先我们初始化值：

$$V_0(s) = 0$$

- 我们通过之前的值进行迭代：

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

□ **极大似然估计** – 状态转移概率的极大似然估计如下：

$$P_{sa}(s') = \frac{\text{\#状态 } s \text{ 下进行动作 } a \text{ 并且进入状态 } s' \text{ 的次数}}{\text{\#状态 } s \text{ 下进行动作 } a \text{ 的次数}}$$

□ **Q 学习** – Q 学习是一种 Q 的无模型(model-free)估计，如下所示：

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