

目录

0 简介	3
1 循环神经网络	3
1.1 RNN 的一般结构	3
1.2 简单循环网络	4
1.3 RNN 的应用	5
1.3.1 文本分类 (序列 - 类别)	5
1.3.2 词性标注 (序列 - 序列, 同步)	5
1.3.3 机器翻译 (序列 - 序列, 异步)	6
1.4 长短期神经网络	6
1.5 代码实现	7
2 卷积神经网络	8
2.1 卷积	8
2.1.1 一维卷积	9
2.1.2 二维卷积	9
2.1.3 互相关	10
2.1.4 卷积的变种	10
2.2 卷积神经网络结构	11
2.2.1 卷积层	11
2.2.2 汇聚层	11
2.2.3 卷积网络的一般结构	12
2.3 代码实现	12
3 无监督学习	13
3.1 变分自动编码机	14
3.1.1 VAE 损失函数	14
3.1.2 重新参数化技巧	15
3.2 去偏变分自动编码机	15
3.2.1 DB-VAE 损失函数	16
3.2.2 自适应重采样	16
3.3 代码实现	17
4 强化学习	19
4.1 Deep Q-Learning Network (DQN)	19
4.1.1 基本概念	19
4.1.2 Q-Learning	20
4.1.3 DQN	21
4.1.4 ε - greedy 策略	21
4.2 Cartpole 问题及代码实现	22

0 简介

本次 Deep Learning 学习使用的教材为 [《神经网络与深度学习》邱锡鹏](#), 该报告上部分图片来源于该教材. 网络资源使用的是 2022 年麻省理工公开课 [MIT 6.S191](#), 部分代码参考来源于此. 使用的神经网络框架为 [TensorFlow](#) 版本为 2.6.0, Python 版本为 3.9.13, 版本在 3.8.0 以上均可. 使用到的第三方 Python 库有: tensorflow, keras, numpy, matplotlib, pandas, mitdeeplearning, tqdm, gym.

配置方面 CPU 为 AMD Ryzen Threadripper 3990X 64-Core Processor, GPU 为 NVIDIA GeForce RTX 3090, RAM 大小为 64GB, 系统为 Windows 10.

该报告全部代码均托管于[GitHub 仓库 - DeepLearning-Summer](#), 除了最后一个项目代码直接为.py 文件, 其他代码均为.ipynb 文件, 该文件同时包含了笔记和代码, 便于阅读, 推荐使用 Jupyter Notebook 查看、运行, Pycharm 和 VScode 也都同样支持, GitHub 支持在线查看功能.

1 循环神经网络

循环神经网络 (Recurrent Neural Networks, RNN) 是对前馈型全连接神经网络的改进. 全连接神经网络的输入维度在确定网络结构的时候已经固定, 而且当特征之间具有的潜在关联性也无法解决. 循环神经网络支持输入任意长度的特征, 并且可以通过处理特征之间具有的关联性, 这也称为神经网络的记忆力.

一般的 RNN 具有短期记忆力, 使用**门控机制** (Gating Mechanism) 可以使其具有更长的记忆, 例如 LSTM 和 GRU, 它们可以解决在长序列下发生的梯度爆炸和消失的问题, 也称为**长程依赖问题**.

1.1 RNN 的一般结构

为解决前馈型全连接神经网络无法处理与输入顺序相关的特征输入, 而通过加入短期记忆力可以解决该问题: 使用**带自反馈神经元**处理任意长度的**时序数据**.

时序数据: 有时间上相关性的数据, x_1, x_2, \dots, x_t 有前后时间相关性, 即在每个时刻 t 下, 神经网络只能看到得到 t 时刻之前的全部数据, 即 x_t, x_{t-1}, \dots, x_1 的特征输入.

图1就是 RNN 的一般结构, 看起来非常简单, 我们可以将文字用数学符号表示, 并将其展开如图2.

将隐藏层中神经元的活性值 h_t 称为**隐状态** (hidden state), 定义式为

$$h_t = \begin{cases} f(h_{t-1}, x_t), & t \geq 1, \\ 0, & t = 0. \end{cases}$$

其中 $f(\cdot)$ 为非线性函数, 它可以是一个复杂前馈网络, 也可以是一个简单的 sigmoid 函数. 对于不同的 $f(\cdot)$ 我们可以得到不同版本的 RNN.

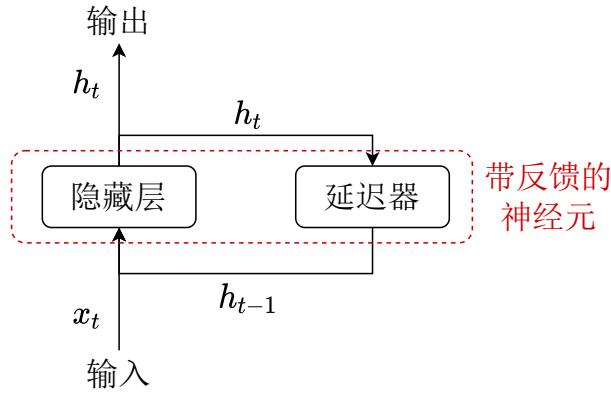


图 1: RNN 简化结构

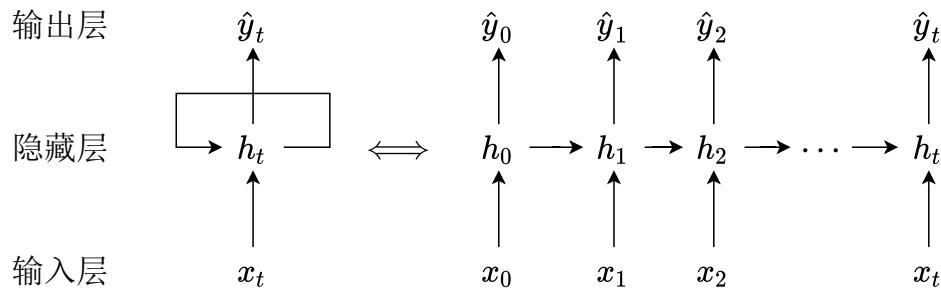


图 2: RNN 结构

1.2 简单循环网络

简单循环网络 (Simple Recurrent Network, SRN) . 设 $x_t \in \mathbb{R}^M$ 为 t 时刻的输入, $h_t \in \mathbb{R}^D$ 为隐状态 (活性值), 则

$$\begin{cases} y_t = Vh_t \\ z_t = Uh_{t-1} + Wx_t + b \\ h_t = f(z_t) \end{cases} \quad (1.1)$$

其中 $U \in \mathbb{R}^{D \times D}$, $W \in \mathbb{R}^{D \times M}$ 为权重矩阵, 前者也称为状态-状态矩阵, 后者也称为状态-输入矩阵, $b \in \mathbb{R}^D$ 为偏置向量, $f(\cdot)$ 为非线性函数 (sigmoid 或 tanh) . 如图3所示 (方框为非线性变化) .

这里不加证明地给出与 RNN 相关的两个定义, 大致含义就是 RNN 可以描述一个给定的空间中所有的点随时间状态变化的情况 (动力系统) .

定理 1.1 (RNN 通用近似定理 Haykin). 一个全连接 RNN 可以以任意准确率近似任一非线性动力系统.

定理 1.2 (Turing 完备). 一个使用 sigmoid 型激活函数的全连接 RNN 可以模拟所有图灵机 (解决所有可计算问题) .

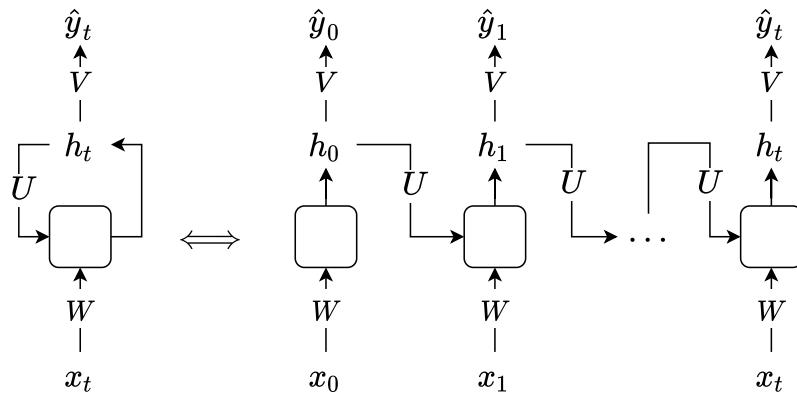


图 3: 简单循环神经网络结构

1.3 RNN 的应用

1.3.1 文本分类 (序列 - 类别)

样本特征: 长度 T 的时间序列 $\mathbf{x} = (x_1, \dots, x_T) \in \mathbb{R}^T$, 标签: 分类类别 $y \in \{1, 2, \dots, c\}$. 可以将文本信息作为输入, 然后将 RNN 的输出连接到全连接神经网络进行分类. 有两种网络结构如图4所示.

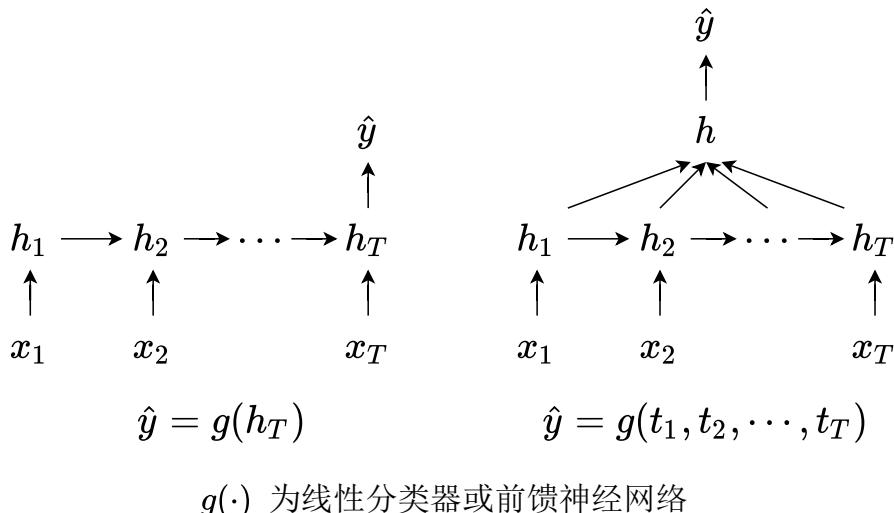


图 4: “序列-类别” 网络结构

代码实现: [语义识别 - 利用 RNN 判断正面和负面的电影评论.](#)

1.3.2 词性标注 (序列 - 序列, 同步)

输入变量个数和输出变量个数一一对应, 样本特征: 长度为 T 的时间序列 $\mathbf{x} = (x_1, \dots, x_T) \in \mathbb{R}^T$, 标签: $\mathbf{y} = (y_1, \dots, y_T) \in \mathbb{R}^T$. 网络结构如图5所示.

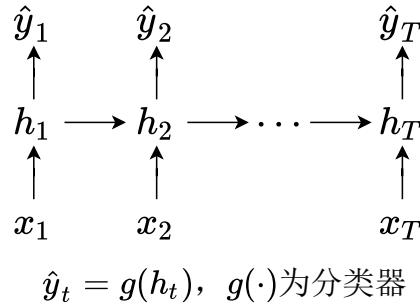


图 5: “序列-序列，同步” 网络结构

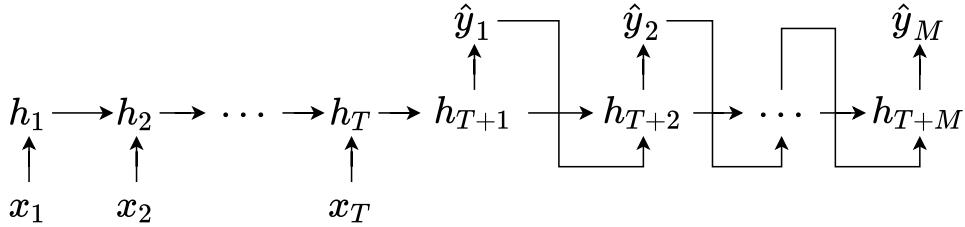


图 6: “序列-序列，异步” 网络结构

1.3.3 机器翻译（序列 - 序列，异步）

序列 - 序列网络结构也称为编码器 - 解码器 (Encoder - Decoder)，没有严格的对应关系，无需保持相同长度，样本特征：长度为 T 的时间序列 $x = (x_1, \dots, x_T) \in \mathbb{R}^T$ ，标签 $y = (y_1, \dots, y_M) \in \mathbb{R}^M$. 网络结构如图6所示，图中变量之间的关系由 (1.2) 式给出.

$$\begin{cases} h_t = f_1(h_{t-1}, x_t), & t \in [1, T], \\ h_{T+t} = f_2(h_{T+t-1}, \hat{y}_{t-1}), & t \in [1, M], \\ \hat{y}_t = g(h_{T+t}) & t \in [1, M], \\ h_0 = \hat{y}_0 = 0. \end{cases} \quad (1.2)$$

1.4 长短期神经网络

长短期神经网络 (Long Short Term Memory Network, LSTM)，是简单 RNN 神经网络的一种变体，具有更长的记忆力.

在简单 RNN 中，整个神经网络使用的是相同的权矩阵 U ，由于 $h_t = f(Uh_{t-1} + Wh_t + b)$ ，当 $\|U\|_2 < 1$ 时，隐状态 $\|h_t\|_2 \rightarrow 0$ ($t \rightarrow \infty$) (梯度消失)，当 $\|U\|_2 > 1$ 时，隐状态 $\|h_t\|_2 \rightarrow \infty$ ($t \rightarrow \infty$) (梯度爆炸). 所以简单 RNN 无法获得两个长时间差的隐状态之间的关联性 (长程问题)，为了解决这种问题，需要引入 LSTM 算法.

LSTM 是一种引入门控机制 (Gating Mechanism) 的算法，它使用了三种门控单元，分别为：遗忘门 f_t ，输入门 i_t ，输出门 o_t ，新的内部状态 $c_t \in \mathbb{R}^D$ 用于线性循环信息传递，输出信息到外部状态 $h_t \in \mathbb{R}^D$ ， $\tilde{c}_t \in \mathbb{R}^D$ 为候选状态. 它们具有以下关系式：

$$\begin{cases} \tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c), \\ f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f), \\ i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i), \\ o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o), \\ c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \\ h_t = o_t \odot \tanh(c_t). \end{cases} \iff \begin{cases} \begin{bmatrix} \tilde{c}_t \\ f_t \\ i_t \\ o_t \end{bmatrix} = \begin{bmatrix} \tanh \\ \sigma \\ \sigma \\ \sigma \end{bmatrix} \left(W \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix} + b \right) \\ c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \\ h_t = o_t \odot \tanh(c_t). \end{cases} \quad (1.3)$$

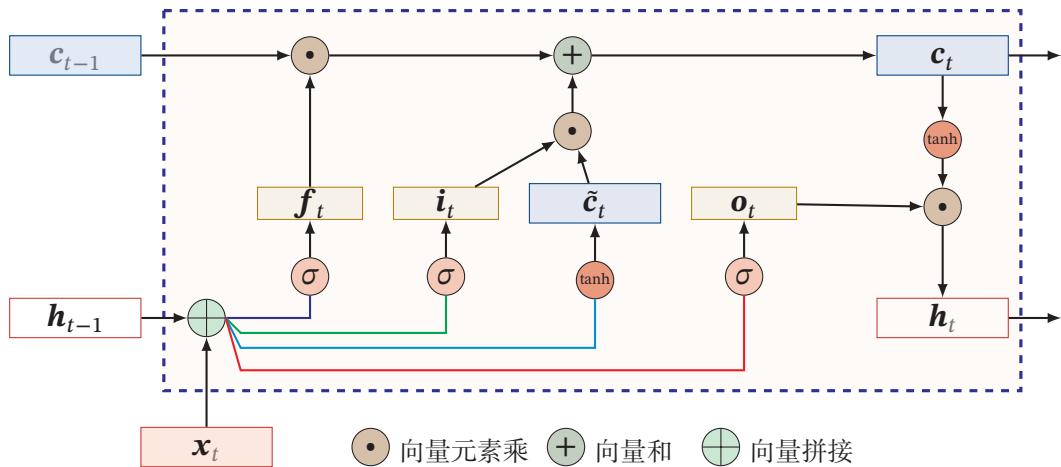


图 7: LSTM

这里的门控机制并非传统的 01 门，而是一种“软”门，取值在 $(0, 1)$ 之间，用于信息的筛选，每个门都有各自的含义：

- 遗忘门 f_t 控制上个时刻内部状态 c_{t-1} 需要遗忘多少信息。当 $f_t = 0$ 时完全清空历史信息。
- 输入门 i_t 控制当前的候选状态 \tilde{c}_t 有多少信息需要保存。当 $f_t = 1, i_t = 0$ 时完全复制上一个时刻的信息。
- 输出门 o_t 控制当前的内部状态 c_t 有多少信息需要输出到外部状态 h_t 。

在隐状态 h 中存储了历史信息，可以视为一种记忆（Memory），LSTM 只是增长了短期的记忆，通过中间记忆单元 c_t 作为媒介减缓更新速度，将 h_t 的更新周期加长，延长生命周期，但是这种做法仍然无法达到真正的长期记忆（保持极长时间的记忆），所以只能称其具有更长的短期记忆，简称长短期记忆（Long Short-Term Memory）。

1.5 代码实现

完整代码：[音乐生成 - 利用 RNN 学习爱尔兰民谣曲谱进行作曲](#).

具体过程可以分为以下几步：

1. 预处理数据集:

- 构建单词库 `vacab`, 以频率高低设置对应索引, 将字符串转为数字.
- 构建训练集 `batch`, 包含 `sequence_length` 和 `batch_size` 两个参数. 每个 `batch` 中的样本序列的开头 `start` 为 $[0, n-\text{len}-1]$ 中随机选取的, 其中 $n=\text{vocab_size}$ 词库大小.
每个样本的特征为数据集的子串 `[start, start+len-1]`,
标签为子串 `[start+1, start+len]`.

2. 搭建模型: `embedding` 层, 参数 `embedding_dimensionality` → `LSTM` 层, 参数 `rnn_units` → `Dense` 层, 参数 `units=vocab_size`.

3. 定义损失函数, 使用交叉熵函数.

超参数配置: `training_iterations`, `learning_rate`. 构建训练函数:

- 使用 `tf.GradientTape` 对变量进行观测, 计算 $\mathcal{L}(y, \hat{y})$.
- 求出 $\frac{\partial \mathcal{L}}{\partial W}$, W 为全体可学习参数 `model.trainable_variables`.
- 使用 `optimizer` 对梯度进行更新.
- 开始训练: 执行训练函数 `training_iterations` 次, 用 `tqdm` 可视化进度条, 在记录点保存模型.

4. 生成歌曲, 根据启动种子 `start_text` 作为预测序列的开头,
用 `tf.random.categorical` 以输出的结果作为概率通过多项分布选出一个预测值, 作为下一次预测的输入值. 反复执行此操作得到曲谱.
生成的曲谱无法保证每次都是符合规范的, 可以通过修改预测序列的开头, 多次重复运行程序, 从而得到一些乐曲, 这里是我们生成出的一些[乐曲](#).

2 卷积神经网络

卷积神经网络 (Convolution Neural Networks, CNN, ConvNet), 结构特性: 局部连接, 权重共享, 汇聚信息. 主要适用于图像处理的一种神经网络, 其想法来源来自于生物模型中的**感受野** (Receptive Field), 即视觉神经元只会接收到其所支配的刺激区域的信号, 即获得某个区域内的加权平均结果, 这种操作在数学中就是卷积.

2.1 卷积

这里的卷积指的是**离散型**的卷积形式.

2.1.1 一维卷积

设 $\{w_i\}, \{x_i\}$ 为两个数列, $k \in \mathbb{R}$, 定义 $\{w_i\}$ 与 $\{x_i\}$ 的有限卷积为以下数列

$$y_t = \sum_{k=1}^K w_k x_{t-k+1}, \quad (t \geq K) \quad (2.1)$$

其中 $\{w_i\}$ 称为滤波器 (Filter) 或卷积核 (Convolution Kernel), $\{x_i\}$ 为信号序列, K 为滤波器长度.

如果我们将数列记为对应的函数值: $w(i) = w_i (1 \leq i \leq K)$, $x(i) = x_i (1 \leq i)$, $y(t) = y_t (K \leq t)$. 则上述定义可视为: 数列 $\{w_i\}, \{x_i\}$ 在 \mathbb{R} 上的零延拓, 即
 $w(i) = \begin{cases} w_i, & 1 \leq i \leq K, \\ 0, & \text{otherwise.} \end{cases}$ 用更形象的方式将其列出如下

$$\begin{aligned} i &= \cdots, -1, 0, 1, 2, \cdots, K, K+1, K+2, \cdots \\ w(i) &= \cdots, 0, w_1, w_2, w_3, \cdots, w_K, 0, 0, \cdots \\ x(i) &= \cdots, 0, x_1, x_2, x_3, \cdots, x_K, x_{K+1}, x_{K+2}, \cdots \end{aligned}$$

定义两个离散数列 $\{w_i\}, \{x_i\}$ 的卷积如下:

$$w * x := \sum_{i=-\infty}^{\infty} w_i x_{t-i+1} \stackrel{i=t-j+1}{=} \sum_{j=-\infty}^{\infty} x_j w_{t-j+1} = x * w \quad (2.2)$$

$$= \sum_{i=1}^K w_i x_{t-i+1} \quad (2.3)$$

通过 (2.2) 式可知卷积具有可交换性, (2.3) 式表明 (2.1) 式中定义的有限卷积其实就是在数列零延拓下的卷积, 再截取 $t \geq K$ 这一段的结果.

卷积操作在信号处理方面有不错的效果, 可以通过不同的卷积核, 对不同的信号进行提取. 下面是几个简单例子.

1. 简单移动平移: $w = [1/k, 1/k, \dots, 1/k]$ (用于时间序列中消除数据的随机波动).
2. 二阶微分近似: $w = [1, -2, 1]$, 由数值分析的知识可知, 连续二阶可微函数 $x(t)$, 有如下近似式

$$x''(t) \approx \frac{x(t-h) - 2x(t) + x(t+h)}{h^2} \stackrel{\text{令 } h=1}{=} x(t-1) - 2x(t) + x(t+1)$$

2.1.2 二维卷积

常用于图像处理, 设图像 $x \in \mathbb{R}^{M \times N}$, 卷积核 $w \in \mathbb{R}^{U \times V}$, 一般有 $U \ll M, V \ll N$, 类比一维卷积定义, 二维卷积定义如下:

$$y_{st} = \sum_{i=1}^U \sum_{j=1}^V w_{ij} x_{s-i+1, t-j+1} = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} w'_{ij} x'_{s-i+1, t-j+1} =: w * x \quad (2.4)$$

其中 w'_{ij}, x'_{ij} 分别为 w_{ij}, x_{ij} 的零延拓, 记 $y = w * x \in \mathbb{R}$. 图8是几种不同卷积核作用在一张图片上的效果.

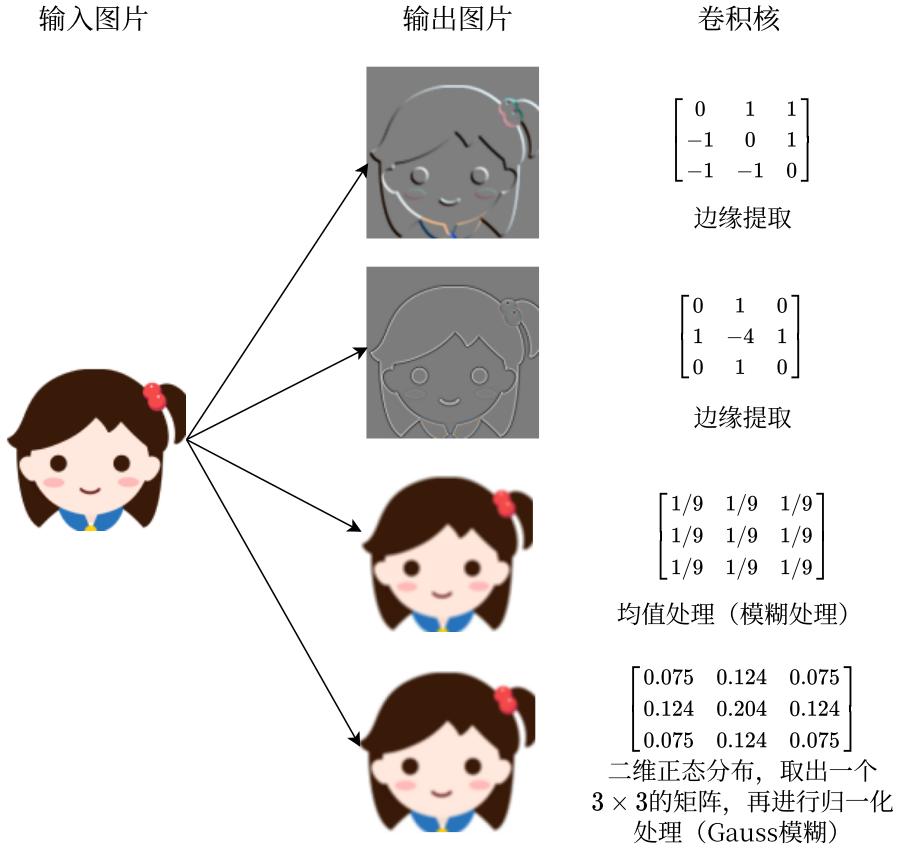


图 8: 不同卷积核处理效果

2.1.3 互相关

在机器学习和图像处理中，卷积的作用主要是通过在一个图像上滑动一个卷积核，通过卷积操作得到一个新的图像。在计算卷积过程中，需要对卷积核进行反转操作，即对卷积核旋转 π 大小。这个操作就显得多余了，所以在计算机中经常将卷积视为互相关 (Cross-Correlation) 操作，即直接对卷积核和原图进行点积操作 (对应位相乘)。

设图像 $x \in \mathbb{R}^{M \times N}$ ，卷积核 $w \in \mathbb{R}^{U \times V}$ ，则它们的互相关为：

$$y_{st} = \sum_{i=1}^U \sum_{j=1}^V w_{ij} x_{s+i-1, t+j-1} = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} w'_{ij} x'_{s+i-1, t+j-1} =: w \otimes x \quad (2.5)$$

和 (2.4) 式对照可知，互相关和卷积的区别仅仅在于卷积核是否需要翻转，即 $w \otimes x = \text{rot}(w) * x$ ， $\text{rot}(w)$ 表示将矩阵 w 旋转 π 以后的结果。因此互相关也称为不翻转卷积。

2.1.4 卷积的变种

在卷积的基础上，还可以引入步长和零填充增加卷积的多样性，以便更灵活地提取图像特征。

- **步长 (Stride)** 指卷积核在滑动时的时间间隔。如图9(a)。
- **零填充 (Zero Padding)** 指对输入矩阵的边缘进行零填充。如图9(b)。

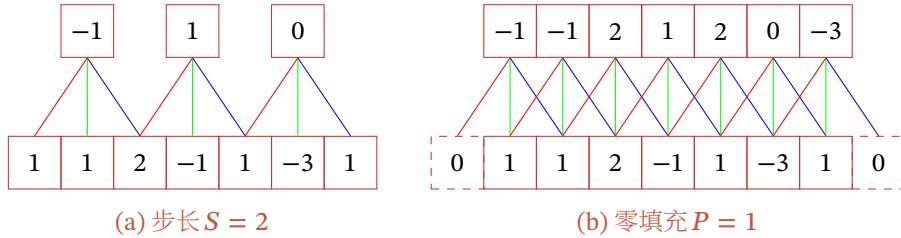


图 9: 步长和零填充

设卷积层的输入向量维数为 M , 卷积大小为 K , 步长为 S , 在输入两端各填补 P 个 0, 则输出向量维度为 $(M - K + 2P)/S + 1$,

常用卷积有以下三种:

1. 窄卷积 (Narrow Convolution): $S = 1, P = 0$, 输出维度为 $M - K + 1$. (普通卷积)
2. 宽卷积 (Wide Convolution): $S = 1, P = K - 1$, 输出维度为 $M + K - 1$.
3. 等宽卷积 (Equal-Width Convolution): $S = 1, P = (K - 1)/2$, 输出维度为 K . 如图9(b) 就是一种等宽卷积.

2.2 卷积神经网络结构

2.2.1 卷积层

卷积层的作用是提取局部区域的特征, 将输入卷积层的矩阵称为**输入特征**, 将通过卷积层后的输出称为**输出特征**, 也称**特征映射** (Feature Map) .

一般的图片每个像素由 RGB 三原色 (颜色通道数为 3) 构成, 假设图片的宽度和高度分别为 N, M , 颜色通道数为 D , 则一张图片 $x \in \mathbb{R}^{N \times M \times D}$, 由于图片的像素值一般为无符号 8 位整型, 即 $x_{ijk} \in [0, 255]$, 所以也有 $x \in [0, 255]^{N \times M \times D}$, 当我们对图片进行归一化处理后, 即 $x \leftarrow x/256$, 就有 $x \in [0, 1]^{N \times M \times D}$.

卷积层中, 假设每个卷积核大小为 $U \times V$, 且每个颜色通道上都对应有 P 个卷积核, 则卷积核 $w \in \mathbb{R}^{U \times V \times P \times D}$, 令第 d 个颜色通道上的第 p 个卷积核为 $w_{d,p}$. 由于每个卷积核 $w_{d,p}$ 作用在图片 x 上都会得到一个输出 y_p , 所以一共有 P 个输出特征, 所以特征映射 $y \in \mathbb{R}^{N' \times M' \times P}$, $N' \times M'$ 为卷积核 $U \times V$ 作用在 $N \times M$ 矩阵后的维度. 可以参考下图更好地理解.

2.2.2 汇聚层

汇聚层 (Pooling Layer) 也称池化层, 子采样层 (Subsampling Layer) . 起作用是对卷积层输出的特征映射进一步进行特征选取, 降低特征数量, 减少参数数量.

设汇聚层的输入特征 $x \in \mathbb{R}^{N \times M \times D}$, 对于其中每一个颜色通道中的图像 x^d , 划分为很多的区域 $\{R_{ij}^d\}$, 满足 $\bigcup_{ij} R_{ij}^d \subset \{x_{ij}\}$, 这些区域可以是不交的, 也可以有交集. 汇聚 (Pooling) 是指对每个区域进行下采样 (Down Sampling) 操作得到的值, 作为该区域的概括.

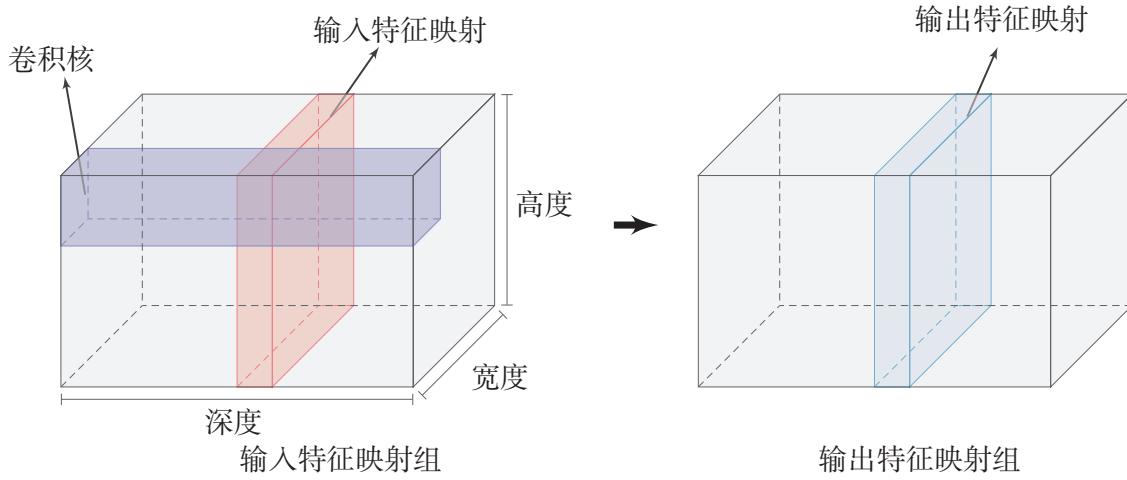


图 10: 卷积层的三维结构

常用的汇聚操作有以下两种:

1. **最大汇聚 (Maximum Pooling)**: 对于一个区域 R_{ij}^d , 选择这个区域内所有神经元的最大活性值作为这个区域的表示, 即

$$y_{ij}^d = \max_{x \in R_{ij}^d} x \quad (2.6)$$

2. **平均汇聚 (Mean Pooling)**: 取该区域内的所有活性值的平均值作为该区域的表示, 即

$$y_{ij}^d = \frac{1}{|R_{ij}^d|} \sum_{x \in R_{ij}^d} x \quad (2.7)$$

其中 $|R_{ij}^d|$ 表示集合 R_{ij}^d 的基数, 即该集合中所包含元素的个数.

2.2.3 卷积网络的一般结构

一个经典卷积网络由卷积层、汇聚层、全连接层堆叠而成, 常用卷积神经网络结构如图11所示. 一个**卷积块**为一组连续 M 个卷积层和 b 个汇聚层构成 (M 取值通常为 $2 \sim 5$, 且卷积核大小逐层增大, 个数逐层增多, b 通常取为 0 或 1), 卷积神经网络堆叠 N 个连续的卷积块, 然后连接 K 个全连接层 (N 通常取为 $1 \sim 100$ 或更大, K 一般取为 $0 \sim 2$)

卷积网络的卷积核大小一般取为 2×2 或 3×3 , 以及更多的数量如 32 个或更多. 由于卷积可以设置步长减少输出特征的大小, 所以汇聚层的作用并不显著了, 可以通过增加步长来替代.

2.3 代码实现

完整代码: 1. 基于前馈型全连接神经网络的数字识别; 2. 基于 RNN 的数字识别 (数据增强) .

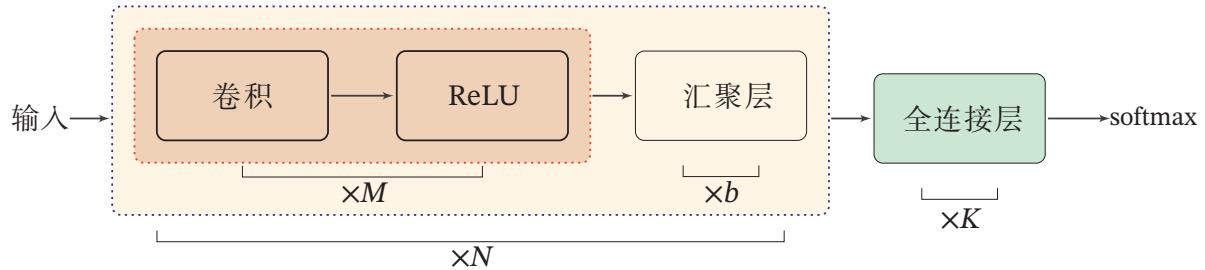


图 11: 经典卷积网络结构

第一个版本是最简单的全连接神经网络模型，实现较为简单，对数据集的识别率已经达到 95% 以上，但是如果自定义输入数字，识别效果并不好。所以第二个版本，在加入 RNN 的基础上，进行了**数据增强操作**。

数据增强简单来说就是对原有数据集的图像增加噪声，随机添加轻微扰动后再加入训练集，从而提高模型的鲁棒性。常见的扰动操作有旋转，平移，拉伸，缩放等，图12就举出了一些例子，最左端为原始图片，右侧均为经过变换后的图片。



图 12: 数据增强

优化后的算法准确率达到 98% 甚至更高，如图13所示，模型对自定义数字输入识别正确率也极高。

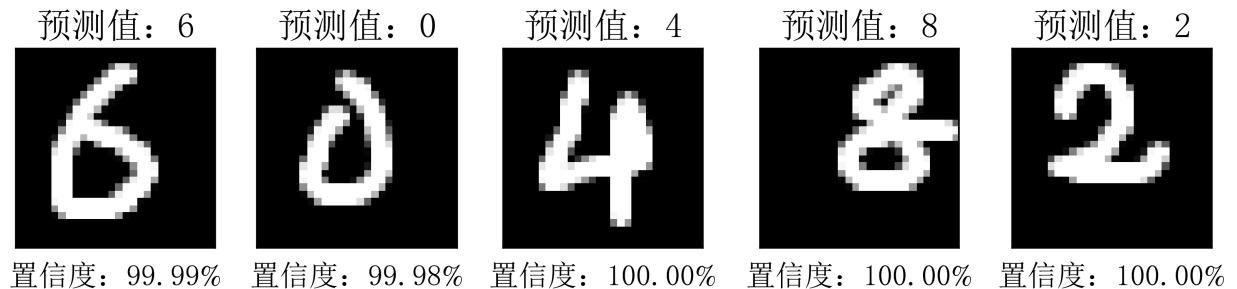


图 13: 自定义手写数字分类

3 无监督学习

之前我们所学习的神经网络算法均为**监督学习**（Supervised Learning），即每个训练样本均对应一个标签。而**无监督学习**（Unsupervised Learning）没有标签的概念，仅给出训练样本，然后模型对其进行处理，下面的变分自动编码机就是一种无监督学习算法。

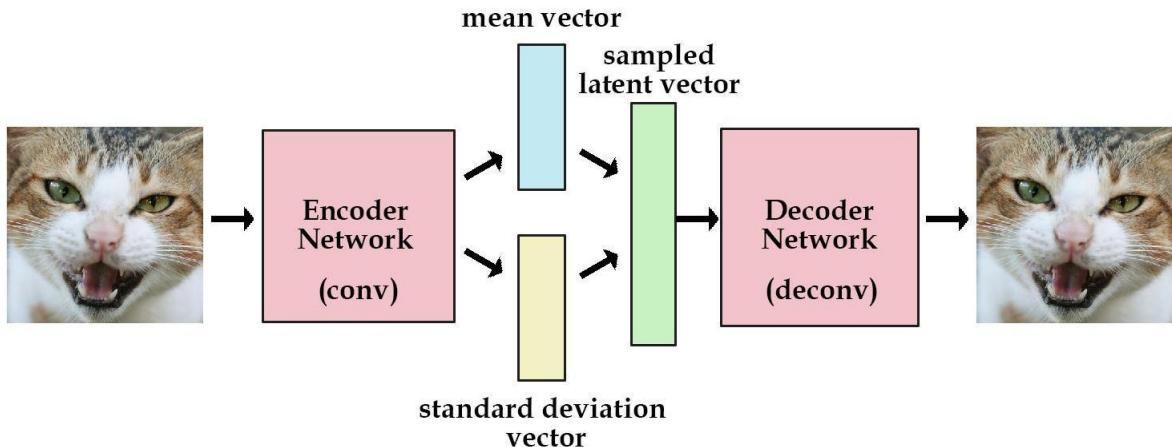


图 14: VAE 基本结构

3.1 变分自动编码机

变分自动编码机 (Variational AutoEncoder, VAE), 是一种通过完全无监督的方式学习图片中的潜在特征编码.

通过图14和 MIT 6.S191 第四讲可知, VAE 通过**编码-解码** (Encoder-Decoder) 结构来学习输入数据的潜在表示. 在计算机视觉中, **编码网络** (Encoder Network) 用于接受输入图像, 将它们编码为一系列由**均值** μ (Mean) 和**标准差** σ (Standard Deviation), 通过这两个参数就可以定义出**潜空间** (Latent Space, 概率分布函数, 通常使用 **Gauss 分布**), 然后从该空间中进行**采样** (Sample, 根据概率分布随机取样), 得到一组**潜变量** (Latent Variables). 然后通过**解码网络** (Decoder Network) 对这些潜变量进行解码, 从而得到输入图像的重建结果. 我们期望输出的结果与输入图像能够尽可能地相似. 其中编码过程可以通过卷积神经网络实现, 解码层可以通过**转置卷积神经网络** (卷积的逆运算) 完成.

设输入图像为 x , 编码过程相当于计算出概率分布 $q_\phi(z|x)$ (潜空间), 然后对 $q_\phi(z|x)$ 进行采样得到编码 z , 然后对 z 进行解码计算出 \hat{x} , 解码器也可以抽象为一个概率分布 $p_\theta(x|z)$. 我们期望输入图像与输出图像差别尽可能小, 即 $\|x - \hat{x}\|^2$ 尽可能小, 且希望潜空间 $q_\phi(z|x)$ 近似于某个期望的分布 $p(z)$, 即 $D(q_\phi(z|x)||p(z))$ 最小, $D(q||p)$ 用于衡量两个概率分布的差距, 一般取为 KL 散度.

在训练模型的过程中, 可以通过 VAE 识别哪些潜变量对模型训练更加重要. 下面让我们将具体分析 VAE 的两个关键部分的损失函数, 并讨论如何对其参数进行梯度更新.

3.1.1 VAE 损失函数

潜空间就是潜变量的概率分布函数, 可以通过在潜空间采样获得潜变量, 我们需要将潜空间 $\mathcal{N}(\mu, \sigma^2 I)$ 向一个标准 Gauss 分布 $\mathcal{N}(0, I)$ 近似, 这样可以使得潜变量更具有连续性, 避免其分布过于分散. 这里需要对可学习参数进行更新, 所以我们需要定义第一个损失函数 (Loss Function). 并且 VAE 用这些参数进行图像重建后, 还需考虑和输入图像的匹配程度, 这里需要第二个损失函数. 因此我们 VAE 的损失函数具有两项:

1. **潜损失 Latent Loss L_{KL}** : 用于衡量潜空间和标准 Gauss 分布的匹配程度, 这里由 Kullback-Leibler (KL) 散度所定义.

2. **重建损失 Reconstruction Loss** $L_x(x, \hat{x})$: 用于衡量重建所得到的图片与输入图片的匹配程度, 由 L^1 范数所定义.

潜损失的表达式 (KL 散度, μ, σ 分别为编码的均值和标准差):

$$L_{KL}(\mu, \sigma) = \frac{1}{2} \sum_{j=0}^{k-1} (\sigma_j + \mu_j^2 - 1 - \log \sigma_j) \quad (3.1)$$

重建损失的表达式 (L^1 范数, 其中 x 为输入特征, \hat{x} 为重建输出):

$$L_x(x, \hat{x}) = \|x - \hat{x}\|_1 \quad (3.2)$$

综上, VAE 损失为:

$$L_{VAE} = c \cdot L_{KL} + L_x(x, \hat{x}) \quad (3.3)$$

其中 c 为潜损失的权系数, 即用于正则化的加权系数.

3.1.2 重新参数化技巧

VAE 需要使用“重新参数化技巧”(Reparameterization Trick) 对潜变量取样, 由于潜变量 $z \sim q(z|x)$, 而梯度下降法中不能出现随机变量, 所以需要利用该技巧, 将 z 固定下来. 由于 $q(z|x)$ 可由 Gauss 分布近似, 则可以对 z 按照特定均值和方差的 Gauss 分布进行取样, 从而可以进行梯度下降法对参数进行学习. 假设 VAE 编码中生成的均值和方差分别为 μ, σ , 则潜变量 $z \sim \mathcal{N}(\mu, \sigma^2 I)$, 可以通过多维标准正态分布 $\varepsilon \sim \mathcal{N}(\mu, I)$ 平移和等比放缩得到.

$$z = \mu + e^{\frac{1}{2} \log \Sigma} \circ \varepsilon \quad (3.4)$$

其中 $\Sigma = \sigma^2 I$ 为随机变量 z 的协方差矩阵.

3.2 去偏变分自动编码机

去偏变分自动编码机 (Debiasing Variational AutoEncoder, DB-VAE) 为 VAE 的一个增强版, 在传统 VAE 基础上, 它增加了去偏的功能: 通过自适应重采样 (自动选择数据, 进行重复性训练) 减轻训练集中的潜在偏差. 例如: 面部识别训练集中, 大多数图片的人脸都是正面图像, 而侧脸的图像偏少, 如果将它们均等地训练, 训练出的模型可能对正脸识别效果优于侧脸的效果, 这就是**数据偏差** (Debiasing). 为了平衡这种偏差有两种方法, 一是使用人工处理, 提高数据集中偏差数据的训练数量, 但操作十分复杂, 而且人无法判断哪些数据是偏差数据; 二是通过机器自动识别偏差数据, 然后自我调整数据的训练数量, 这就是 DB-VAE 的提升之处. DB-VAE 的示意图如图15所示, 图片来源论文 [Uncovering and Mitigating Algorithmic Bias through Learned Latent Structure](#).

注意到, DB-VAE 编码部分有一个单独输出的有监督变量 z_0 , 例如, 该变量可以用于判断是否该图片是人脸图像. 而一般的 VAE 并不具有有监督变量输出的功能, 这也是 DB-VAE 与传统 VAE 不同之处.

需要注意如果是数据集中既有人脸图像也有非人脸图像, 我们仅想学习人脸相关的潜变量, 对数据集做去偏操作, 并做一个二分类问题. 所以我们要确保模型仅对人脸图

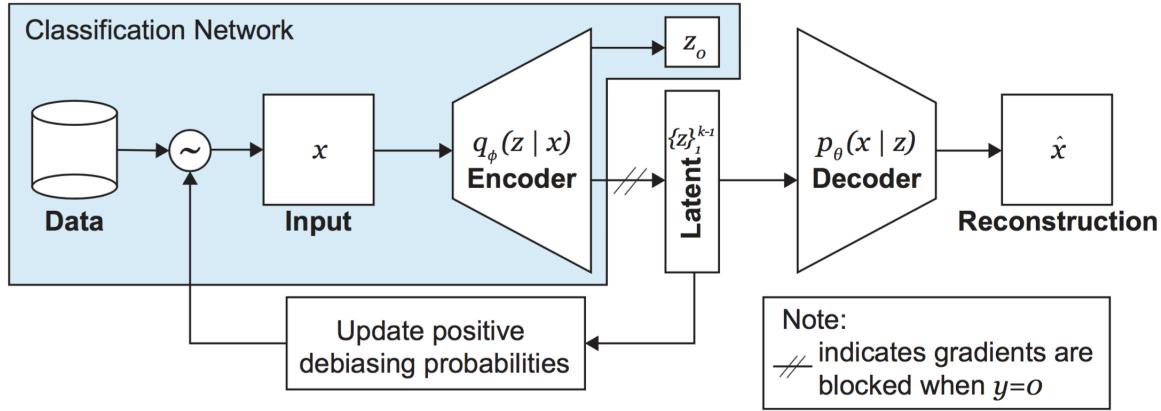


图 15: DB-VAE 基本结构

片从分布 $q_\phi(z|x)$ 中获取无监督潜变量的表示，并且输出一个有监督的分类预测 z_0 ，而对于非人脸图片，我们只需要输出一个预测 z_0 即可。

3.2.1 DB-VAE 损失函数

我们需要对 DB-VAE 的损失函数进行一些改进，损失函数要与是否是人脸图片相关。

对于人脸图片，我们的损失函数将包含两项：

1. 传统 VAE 损失函数 L_{VAE} : 包含潜损失和重建损失.
2. 分类损失 $L_y(y, \hat{y})$: 二分类问题的标准交叉熵损失函数.

相反地，对于非人脸图片，我们的损失函数仅有分类损失这一项。则 DB-VAE 损失函数为：

$$L_{total} = L_y(y, \hat{y}) + \chi_{image}(y) \cdot L_{VAE} \quad (3.5)$$

其中 $\chi_{image}(y) = \begin{cases} 1, & y = 1, \text{ 训练样本为人脸图片,} \\ 0, & y = 0, \text{ 训练样本为非人脸图片.} \end{cases}$

3.2.2 自适应重采样

回想 DB-VAE 的架构：当图像通过网络输入时，编码器会学习得到潜空间中 $q_\phi(z|x)$ 的估计。我们希望通过增加对潜空间中代表性不足区域的采样，从而增加稀有数据的相对训练次数。我们可以通过每个学习到的潜变量的频率分布对 $q_\phi(z|x)$ 进行近似，根据**中心极限定理**（随机变量部分和分布渐近与 Gauss 分布）近似结果应该趋近于 Gauss 分布，从该近似结果中我们可以得到出现每个潜变量的出现频率占比，然后将出现频率取倒数（提高出现频率低的样本的重采样率），再归一化处理，将这个概率分布将用于数据的重新采样。

3.3 代码实现

[完整代码及解释 - Face Detection VAE.](#)

目标为识别输入照片是否是人脸图像，我们使用了两个数据集：

1. 正训练集：[CelebA Dataset](#)，包含超过二十万张名人照片。
2. 负训练集：[ImageNet](#)，该网站上有非常多不同分类的图片，我们将从非人脸类别中选取负样本。通过 [Fitzpatrick 度量法](#) 对肤色进行分类，将图片标记为“Lighter”或“Darker”。

然后我们使用了经典 CNN 和 DE-VAE 神经网络对图片进行识别，图16体现了去偏后的训练效果。

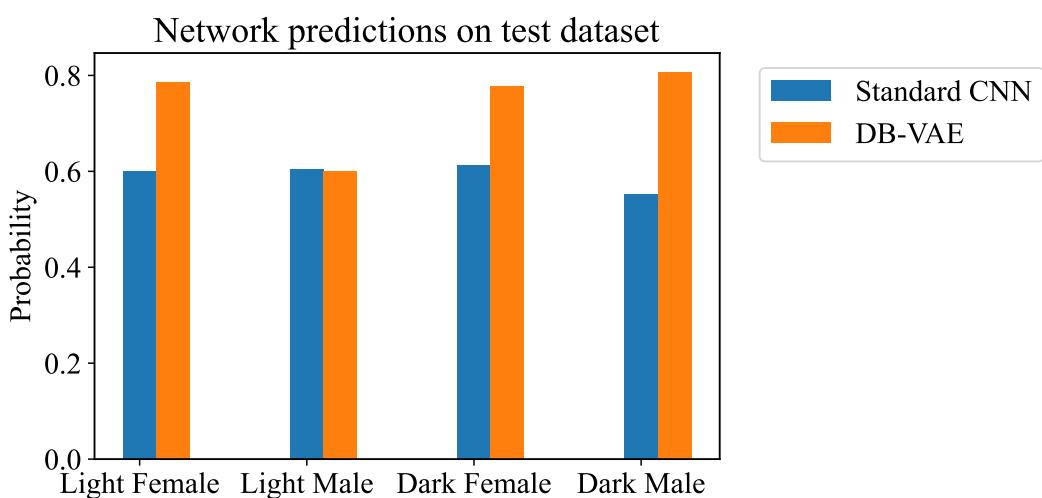


图 16: CNN 与 VAE 算法在带偏差的数据下分类效果比对

图17展示了 VAE 的图像渐变转化功能（变脸效果），清晰的图像为输入的图片（左右两端），较为模糊的图像为 VAE 输出的预测结果。

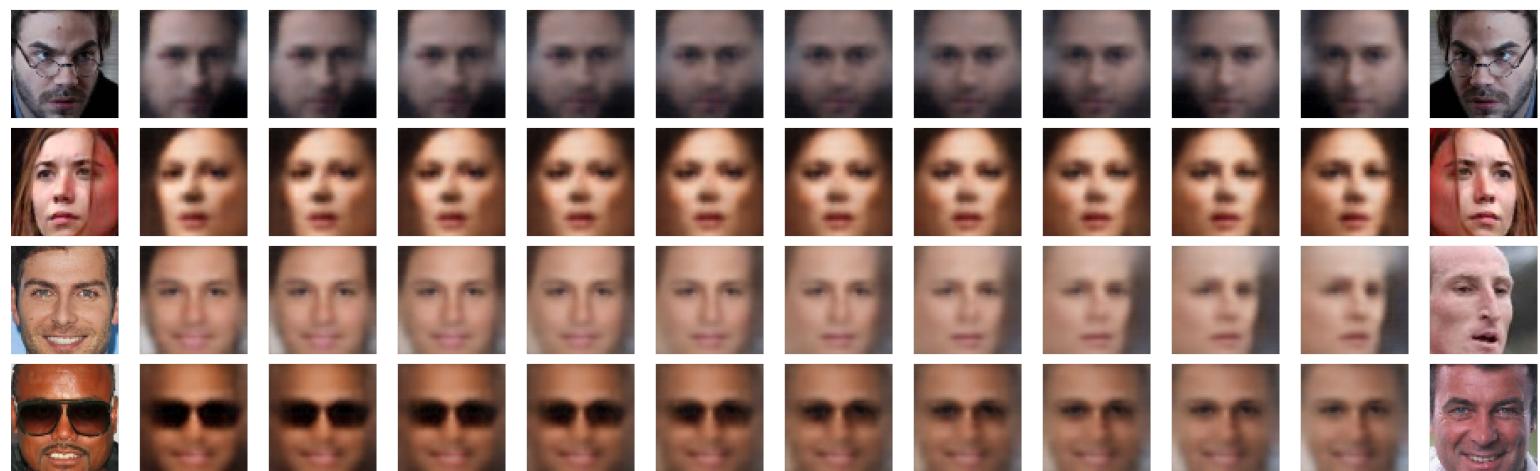


图 17: 图像渐变效果

4 强化学习

强化学习 (Reinforcement Learning, RL) 是一种通过不断试错，并从中不断改进从而提升效果的学习算法。

一般来说，游戏或电脑中模拟的现实情况称为环境 (Environment), 智能体 (Agent) 在环境中可以做出行动 (Action) 从而最大化累积奖励 (Reward), 在每次行动后，智能体可以通过观察 (Observe) 环境得到状态 (State) 的变化并获得当前行动的奖励。这段过程可以随着时间序列反复进行，可以表示为图18的 Markov 链形式：

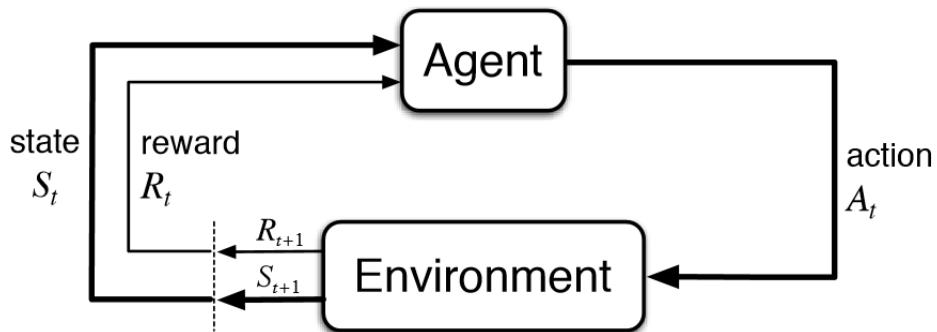


图 18: SARSA' Markov 链示意图

我们通过初始化环境 s_0 开始，对于每一次迭代，智能体会观察当前的状态 s_t ，通过预测算法，执行最好的行动 a_t 。随后环境会返回当前行动对应的奖励 r_t 和一个新的状态 s_{t+1} ，并且可以得到当前游戏是否已经结束。上述过程可以反复执行，直到游戏结束。

4.1 Deep Q-Learning Network (DQN)

DQN 算法原文连接：[2013 版 \(arxiv\)](#), [2015 版 \(nature\)](#)，下面将详细介绍 DQN 算法。

4.1.1 基本概念

设环境状态空间维数为 N ，行为空间维数为 M . 在第 t 时刻的状态记为 s_t ，做出的行动为 a_t ，得到的奖励为 r_t ，其中 $t \in \mathbb{N}, s_t \in \mathbb{R}^N, a_t \in \mathbb{R}^M, r_t \in \mathbb{R}$.

策略 Policy 策略 π 指的是智能体的行动函数 $\pi(s) : \mathbb{R}^N \rightarrow \mathbb{R}^M$ ，从状态空间映射到行动空间，表示在 s 状态下，智能体做出的行动。强化学习中有两种 Policy，分别是**确定性策略** (Deterministic policy) 和**不确定性策略** (Stochastic policy)，前者根据状态，给出确定的行动，而后者给出每个行动的概率。

阶段 Episode 一个阶段指的是：游戏从开始到结束的过程，结束时刻可以是游戏终止或者是到达时间上限。一个阶段由每个时刻下的状态，行为，奖励构成：

$$(s_1, a_1, r_1, s_2, a_2, r_2, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T), \quad T \in \mathbb{N}$$

回报函数 Return Function 又称累积折扣奖励 (Cumulative Discounted Reward), 定义在 t_0 时刻下的 return 为:

$$G_{t_0} = \sum_{t=t_0}^{\infty} \gamma^{t-t_0} r_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \quad (4.1)$$

其中 $\gamma \in (0, 1]$ 表示折扣因子 (Discount Factor), 在训练前固定的常数. 上述公式类似于金融中的贴现金额, γ 类似于贴现因子, 相同价值的物品, 随着时间越久, 其相对价值会随之降低.

当 $\gamma \in (0, 1]$ 时, G_t 为累积奖励函数, 智能体的学习目标就是最大化累积奖励的期望, 即下文的值函数.

值函数 Value Function 值函数分为两种:

第一种称为状态值函数, $V_{\pi}(s) : \mathbb{R}^N \rightarrow \mathbb{R}$, 定义为

$$V_{\pi}(s_t) = \mathbf{E}_{\pi}(G_t | S = s_t) \quad (4.2)$$

上式表明从 t 时刻的状态 s_t 开始, 遵循策略 π 时的回报函数 G_t 的期望值.

第二种称为状态行动值函数, $Q_{\pi}(s_t, a_t) : \mathbb{R}^{N+M} \rightarrow \mathbb{R}$, 定义为

$$Q_{\pi}(s_t, a_t) = \mathbf{E}_{\pi}(G_t | S = s_t, A = a_t) \quad (4.3)$$

上式表示从 t 时刻的状态 s_t 开始采取行动 a_t 遵循策略 π 时回报函数 G_t 的期望值, 与第一种值函数的区别在于多确定了一步的行动 a_t . 用神经网络对值函数 $Q_{\pi}(s_t, a_t)$ 进行近似是 DQN 的核心思路.

强化学习是一个反复重复上述迭代的过程, 在每次迭代时, 需要解决两个问题: 通过值函数更新策略, 和根据策略更新值函数.

4.1.2 Q-Learning

首先弄清楚 Q-Learning 算法原理, 假设我们不清楚具体策略 π 是什么, 通过随机给定的一个函数 $Q(s, a)$ 去近似 $Q_{\pi}(s, a)$, 考虑之前梯度下降算法的思路, 通过求偏导可以得到学习参数的下降方向, 类似的, 这里也要找到 $Q(s, a)$ 合理的下降方向. 通过变化 $Q_{\pi}(s_t, a_t)$ 的定义式我们可以发现有意思的结果, 当 $t+1$ 时刻不是终止时刻时:

$$\begin{aligned} Q_{\pi}(s_t, a_t) &= \mathbf{E}_{\pi}(G_t | S = s_t, A = a_t) \\ &= \mathbf{E}_{\pi}(r_t + \gamma G_{t+1} | S = s_t, A = a_t) \\ &= r_t + \gamma \mathbf{E}_{\pi}(G_{t+1} | S = s_t, A = a_t) \\ &= r_t + \gamma \mathbf{E}_{\pi}(G_{t+1} | S = s_{t+1}) \\ &= r_t + \gamma \max_{a' \in \mathbb{R}^M} \mathbf{E}_{\pi}(G_{t+1} | S = s_{t+1}, A = a') \\ &= r_t + \gamma \max_{a' \in \mathbb{R}^M} Q_{\pi}(s_{t+1}, a') \end{aligned} \quad (4.4)$$

其中 r_t, s_{t+1} 分别为 s_t 状态下执行行动 a_t 所得到的奖励和新的状态. 当 $t+1$ 时刻是终止时刻时, 则 $Q_{\pi}(s_t, a_t) = r_t$. 于是我们得到了 $Q(s_t, a_t)$ 的更新目标: 当 $t+1$ 不是

终止时刻时, $y_t = r_t + \gamma \max_{a' \in \mathbb{R}^M} Q(s_{t+1}, a')$, 当 $t+1$ 是终止时刻时, $y_t = r_t$. 记下降方向 $\mathbf{d} = y_t - Q(s_t, a_t)$, 则更新公式为

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \mathbf{d} \quad (4.5)$$

其中 $\alpha \in (0, 1)$, 称为学习因子或步长.

4.1.3 DQN

DQN 其实就是 Q-Learning 的变体, 将值函数用神经网络去近似. 设神经网络函数为 $Q(s, a; \omega)$, 其中 ω 为该网络的权值. 则损失函数为

$$\mathcal{L}(s_t, a_t, y_t; \omega) = (y_t - Q(s_t, a_t; \omega))^2 \quad (4.6)$$

$$\text{其中 } y_t = \begin{cases} r_t, & t+1 \text{ 步为终止时刻,} \\ r_t + \gamma \max_{a \in \mathbb{R}^M} Q(s_{t+1}, a; \omega), & \text{否则.} \end{cases}$$

和原来的神经网络训练问题比较, 这里的 s_t, a_t 可以视为输入特征, y_t 为对应的标签. 在一般的神经网络中, 标签一般是固定的, 而在这个训练问题中, 标签与当前的神经网络相关, 可能导致算法不收敛. 于是有一种方法是, 在创建一个网络结构与 Q 一模一样的神经网络, 但只有参数值更新速度比 Q 慢, 称为目标神经网络, 用于生成 y_t . 我们记目标神经网络的参数为 ω^- , 当前神经网络的参数为 ω . 则损失函数为

$$\mathcal{L}(s_t, a_t, y_t; \omega) = (y_t - Q(s_t, a_t; \omega))^2, \quad (4.7)$$

$$\text{其中 } y_t = \begin{cases} r_t, & t+1 \text{ 步为终止时刻,} \\ r_t + \gamma \max_{a \in \mathbb{R}^M} Q(s_{t+1}, a; \omega^-), & \text{否则.} \end{cases}$$

区别只在 y_t 的计算来源发生了变化. 在 2013 年发表的论文中没有使用目标神经网络, 而 2015 年发表的论文中才提出目标神经网络.

4.1.4 ε -greedy 策略

greedy 策略指的就是贪心策略, 其每次选取的均为值函数最大的行动, 即 $a_t = \arg\max_{a \in \mathbb{R}^M} Q(s_t, a; \omega)$. 但是这样会导致对于采样中没有出现过的 (s, a) , 由于没有 Q 值, 之后可能就不会被采样到.

这里与强化学习中的重要概念相关, 叫探索与利用 (Exploration & Exploitation). 前者强调发现环境中的更多信息, 不仅局限于已知的信息中; 后者强调从已知的信息中最大化奖励. 而 greedy 策略只注重了后者, 而没涉及到前者.

而 ε -greedy 策略则是以 ε 的概率从行动空间中随机返回一个行动, 以 $1-\varepsilon$ 的概率选择贪心策略, 这样就能更具有随机性. ε 可以随时间的增长而下降, 例如 $\varepsilon = \delta^t$, $\delta \in (0, 1)$.

4.2 Cartpole 问题及代码实现

这是一个在[OpenAI gym](#)上的经典环境，首先需要安装 `gym` 库，该库中包含非常多经典游戏、物理模型，可以用于强化训练学习。

`Cartpole` 问题可以视为一个[倒立摆问题](#)，倒立摆是一个重心高于其枢轴点的摆。它是不稳定的，但是可以通过移动枢轴点位置以保持该系统的稳定性。我们的目标是尽可能长地保持摆的垂直状态。如图19所示，游戏规则如下：

一个摆通过一个无摩擦的枢轴连接到推车上，该推车沿着水平方向的无摩擦轨道移动。通过对推车施加 $+1$ 和 -1 的推力来维持该系统平衡，保持摆的直立状态。当杆在一个时间戳保持直立，则获得 $+1$ 的奖励。当杆与竖直方向的夹角超过 15° ，或者小车相对中心的移动距离超过 2.4 个单位时，游戏结束。

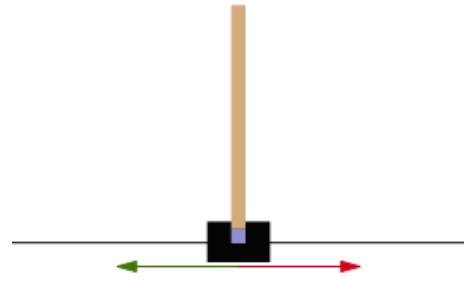


图 19: Cartpole 示意图

- 紫色方块代表枢纽点。
- 红色和绿色箭头分别表示枢纽点可移动的水平方向。

完整代码及解释：[Cartpole 完整模块代码](#)。此部分代码使用的是模块式写法，包含以下 4 个模块：

1. `run.py` 为主程序，用于实例化模块，调用训练函数、测试训练效果。
2. `cartpole.py` 为环境操作及过程性数据保存代码，包含环境创建、预测模型创建、与环境进行交互、获取环境状态、调用模型进行训练、保存训练结果、测试模型效果等功能。
3. `dqn.py` 为智能体算法代码，包含创建神经网络、预测下一步的行动、通过记忆回溯对参数进行训练等功能。
4. `constant.py` 保存所有代码中使用到的常量，包含神经网络所用的超参数、最大训练步数、保存文件的目录等。

可以通过查看 `training_checkpoints` 文件夹下的图片实时查看训练效果，每次重复开始 80 次游戏，完成全部训练在 30 分钟左右，游戏最大时长为 500 帧，我们的模型平均在重复 30 次游戏时就能达到游戏最大时长，训练时间在 15 分钟左右。图20是某次训练的效果曲线，`score` 表示该模型游戏时长（帧），`Q Value` 为预测模型对 Q 值进行的预测结果。

动图效果请见：[动图 1](#), [动图 2](#).

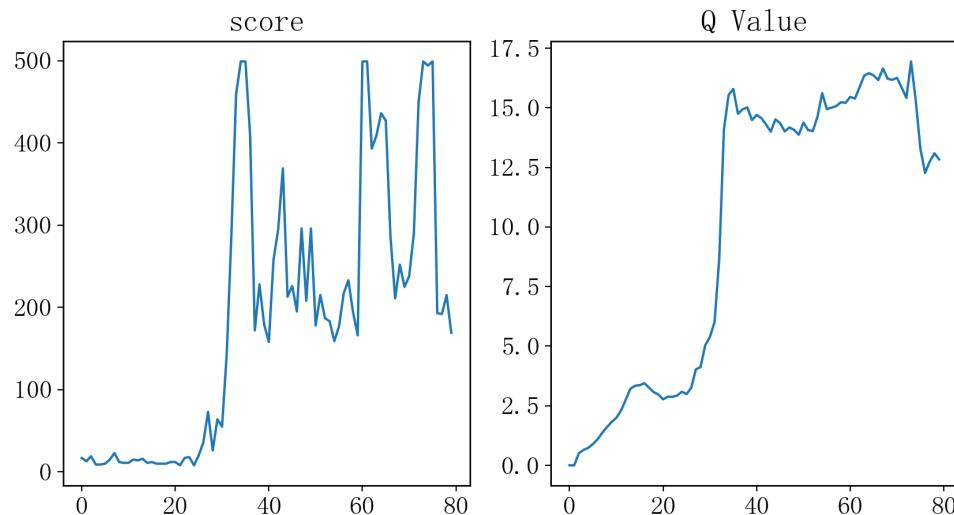


图 20: DQN 训练曲线图

参考资料:

1. [Wanjun's blog - 强化学习—DQN 算法原理详解.](#)

代码参考:

2. [Cartpole - Introduction to Reinforcement Learning \(DQN - Deep Q-Learning\).](#)
3. [PyLessons - Introduction to Reinforcement Learning.](#)