该资料内容为 Introduction to Deep Learning 的复习知识点，以老师给出的概念为复习中心进行的知识点总结，6、7、8 三道题较为复杂仅列出部分知识点，需要自己进行总结提升。希望大家可以一起交流学习，更希望大佬可以完善内容。

祝期末复习愉快~Author@森酱皮皮虾

1.深度学习基本概念。

机器学习概念：专门研究计算机怎样模拟或实现人类的学习行为，以获取新的知识或技能，并重新组织已有的知识结构使之不断改善自身的性能。机器学习算法奏效的原因是人们手工设计特征。然后机器学习就变成了对权重的优化来得到一个最好的预测值。

深度学习：通过级联的单层网络来学习数据的表达。

机器学习通常包含的**任务类型**：分类、回归、降维等。

机器学习**名词解释**：训练(Training)、优化(Optimization)、泛化(Generalization)、欠拟合(Under-fitting)、过拟合(Over-fitting)、维数灾难、正则化(Regularization)。

设计神经网络的步骤：a.选择一个优化模型、代价函数以及输出单元的形式，b.选择用于计算隐藏层值的激活函数，c.设计网络的结构，多少层、每层如何连接以及每层的单元数。

神经网络的非线性导致大多数我们感兴趣的代价函数都变得非凸，利用梯度下降训练时往往得到一个局部的极小值。对于前馈神经网络，将所有的权重值初始化为小随机数是很重要的。

常见的激活函数：sigmoid 函数、tanh(x)函数、ReLu 函数{y(z)=max{0, z}}等。

通常整流线性单元(ReLu)是极好的默认选择，它的优势：其二阶导数处处为 0，

并且在整流线性单元处于激活状态时，它的一阶导数处处为 1，有利于学习。它的劣势：不能通过基于梯度的方法学习那些使它们激活为零的样本。

ReLu 的三个扩展：绝对值 ReLu、Leaky ReLu、Parametric ReLu

2.反向传播算法 BP（Back Propagation）

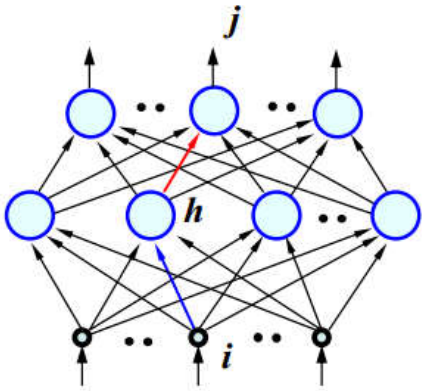
BP 算法的核心是梯度下降法，学习的本质：对网络各连接权重作动态调整

主要计算规则：链式以及复合函数求导法则。

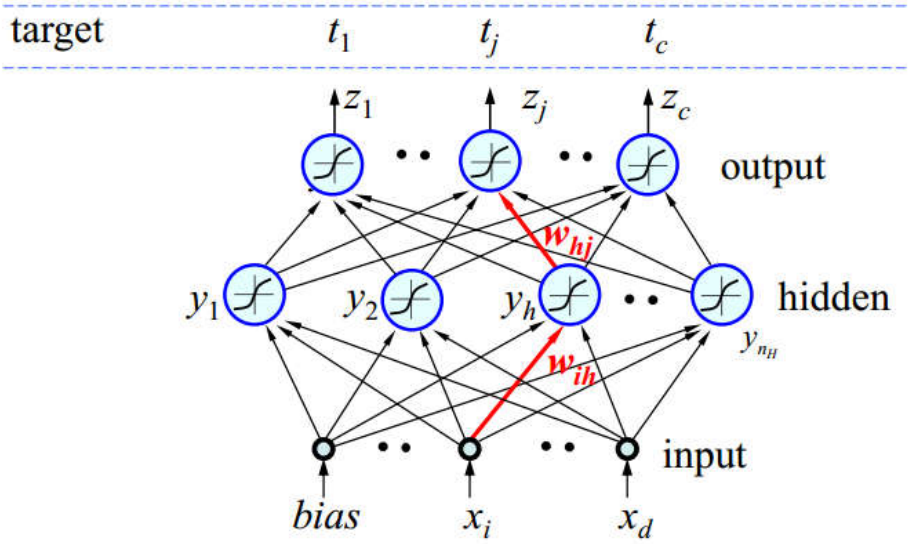
用一个最简单的三层网络结构来展示 BP 算法的整个流程。

三层网络的描述

- 训练数据输入输出对：$\{x_i^k, t_j^k\}$
- 输出层结点的输出：z_j^k
- 隐含层结点的输出：y_h^k
- 输入信号：x_i^k
- 输入端点数目：$d+1$
- 输入层结点 i 至隐含层结点 h 的权重：w_{ih}
- 隐含层结点 h 至输出层结点 j 的加权表示：w_{hj}
- 上标 k 表示训练对的序号，$k=1, 2, \ldots, n$



Hope: $z_1 \approx t_1$, $\cdots$, $z_c \approx t_c$, **for all samples:** $J(\mathbf{w}) = \frac{1}{2}\sum_{j=1}^{c}(t_j - z_j)^2 \approx 0$



- 网络描述-每个样本所经历的计算

上标 ***k*** 联系
第 ***k*** 个样本

对第 k 个样本，隐含层 h 结点的输入加权和为：

$$net_h^k = \sum_i w_{ih} x_i^k$$

经过激励，隐含层 h 结点的输出：

$$y_h^k = f(net_h^k) = f\left(\sum_i w_{ih} x_i^k\right)$$

输出层 j 结点的输入加权和为：

$$net_j^k = \sum_h w_{hj} y_h^k = \sum_h w_{hj} f\left(\sum_i w_{ih} x_i^k\right)$$

经过激励，输出层 j 结点的输出：

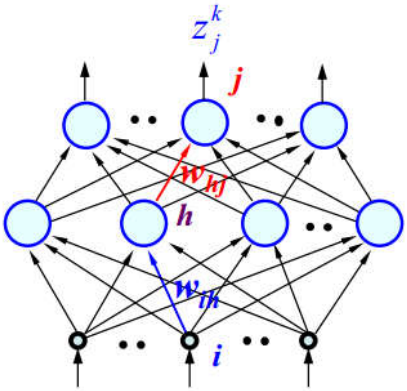
$$z_j^k = f(net_j^k) = f\left(\sum_h w_{hj} y_h^k\right) = f\left(\sum_h w_{hj} f\left(\sum_i w_{ih} x_i^k\right)\right)$$

- 网络训练：隐含层－输出层

隐含层到输出层的连接权重调节量：

$$\Delta w_{hj} = -\eta \frac{\partial E}{\partial w_{hj}} = -\eta \sum_k \frac{\partial E}{\partial net_j^k} \frac{\partial net_j^k}{\partial w_{hj}}$$
$$= \eta \sum_k (t_j^k - z_j^k) f'(net_j^k) y_h^k$$
$$= \eta \sum_k \delta_j^k y_h^k$$

(δ规则)



（接前一页）

$$\Delta w_{ih} = \eta \sum_{k,j} \left(t_j^k - z_j^k\right) f'\left(net_j^k\right) w_{hj} \frac{\partial y_h^k}{\partial w_{ih}}$$
$$= \eta \sum_{k,j} \left(t_j^k - z_j^k\right) f'\left(net_j^k\right) w_{hj} \frac{\partial y_h^k}{\partial net_h^k} \frac{\partial net_h^k}{\partial w_{ih}}$$
$$= \eta \sum_{k,j} \left(t_j^k - z_j^k\right) f'\left(net_j^k\right) w_{hj} f'\left(net_h^k\right) x_i^k$$
$$= \eta \sum_{k,j} \delta_j^k w_{hj} f'(net_h^k) x_i^k$$

$$\delta_j^k = f'\left(net_j^k\right)\left(t_j^k - z_j^k\right)$$

$$= \eta \sum_k \left(f'(net_h^k) \sum_j \delta_j^k w_{hj} \right) x_i^k$$
$$= \eta \sum_k \delta_h^k x_i^k$$

更新权重：
$$\begin{cases} W_{hj} := W_{hj} + \Delta W_{hj} \\ W_{ih} := W_{ih} + \Delta W_{ih} \end{cases}$$

3.CNN 结构分析，设计一个网络，计算 feature map、channel 数等网络结构的参数。

CNN 特点： 局部连接、权值共享来大量降低网络权重的数量。

CNN 操作：

卷积（Convolution）：局部特征提取；池化（Pooling）：降低数据维度，避免过拟合

卷积：原始输入与 Kernel 做卷积和然后加偏置后进行激励（也可在 pooling 之后激励）

池化（Pooling）：两种 Mean Pooling 与 Max Pooling，下采样。例如 2x2 窗口取平均作为这四个值的代表，2x2 窗口取最大值作为这四个值的代表。

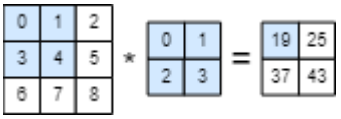
计算涉及相关名词：Input size(N)、Kernel、Kernel size(K)、stride(S)、padding(P)、Output size(O)

卷积或者 Pooling 操作之后的输出大小：$O = \frac{N - K + 2P}{S} + 1$

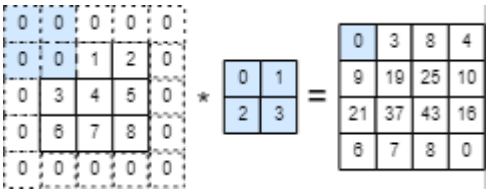
每个卷积核得到一副特征图像被称为一个 Feature Map。 Feature Map 的数量等于计算它之前 Kernel(Filter)的数量乘以输入数据的 channel 数。

具体计算举例

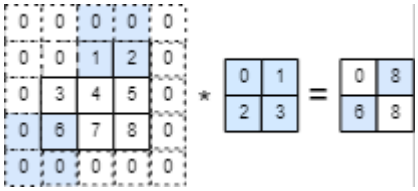
二维卷积运算符（Convolution）一个二维相关运算符将一个二维核（kernel）数组作用在一个二维输入数据上来计算一个二维数组输出。



步长为 1 的卷积和运算

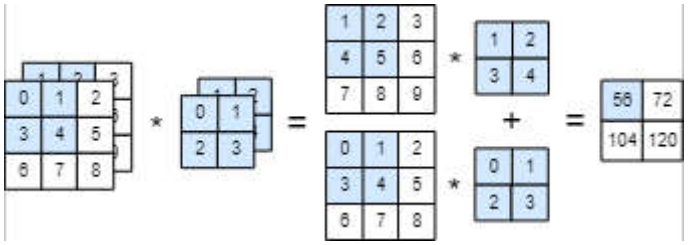


在输入的高和宽两侧分别填充了 0 的二维相关计算（padding），步长为 1



高上使用步幅 3，宽上使用步幅 2

2×2 最大池化层



输入通道为 2 的二维相关计算（多通道输入单通道输出）

某一层的参数数量：kernel_size * kernel_size * channels * kernel_number

4.RNN 的两个变体 LSTM 与 GRU 差别以及各自的优势。

GRU（Gated Recurrent Neural Network），门控循环神经网络。与 RNN 相比，

它引入了两个门（重置门与更新门）的概念，从而修改了循环神经网络中隐藏状态的计算方式。

重置门与更新门介绍:

假设隐藏单元个数为 h，给定时间步 t 的小批量输入$\mathbf{X}_t \in R^{n\times x}$（样本数为 n，输入维度为 x)，和上一时间步隐藏状态$\mathbf{H}_{t-1} \in R^{n\times h}$。重置门 (reset gate) $\mathbf{R}_t \in R^{n\times h}$ 和更新门（update gate）$\mathbf{Z}_t \in R^{n\times h}$的计算如下:

$$\mathbf{R}_t = \sigma\left(\mathbf{X}_t\mathbf{W}_{x\tau} + \mathbf{H}_{t-1}\mathbf{W}_{h\tau} + \mathbf{b}_\tau\right)$$
$$\mathbf{Z}_t = \sigma\left(\mathbf{X}_t\mathbf{W}_{xz} + \mathbf{H}_{t-1}\mathbf{W}_{hz} + \mathbf{b}_z\right)$$

其中$\mathbf{W}_{x\tau}, \mathbf{W}_{xz} \in R^{x\times h}$和$\mathbf{W}_{h\tau}, \mathbf{W}_{hz} \in R^{h\times h}$是权重参数，$\mathbf{b}_\tau, \mathbf{b}_z \in R^{1\times h}$是偏移参数。激活函数$\sigma$是 sigmoid 函数，因此重置门$\mathbf{R}_t$与更新门$\mathbf{Z}_t$中每个元素的值域都是[0,1]。

接下来时间步 t 的候选隐藏状态

$$\tilde{\mathbf{H}}_t = \tanh\left(\mathbf{X}_t\mathbf{W}_{xh} + \mathbf{R}_t \odot \mathbf{H}_{t-1}\mathbf{W}_{hh} + \mathbf{b}_h\right),$$

其中$\mathbf{W}_{xh} \in R^{x\times h}$和$\mathbf{W}_{hh} \in R^{h\times h}$是权重参数，$\mathbf{b}_h \in R^{1\times h}$是偏移参数。需要注意的是，候选隐藏状态使用了重置门，从而控制包含时间序列历史信息的上一个时间步的隐藏状态如何流入当前时间步的候选隐藏状态。如果重置门近似 0，上一个隐藏状态将被丢弃。因此，重置门可以丢弃与预测未来无关的历史信息。

最后，隐藏状态$\mathbf{H}_t \in R^{n\times h}$的计算使用更新门$\mathbf{Z}_t$来对上一时间步的隐藏状态$\mathbf{H}_{t-1}$和当前时间步的候选隐藏状态$\tilde{\mathbf{H}}_t$做组合:

$$\mathbf{H}_t = \mathbf{Z}_t \odot \mathbf{H}_{t-1} + \left(1 - \mathbf{Z}_t\right) \odot \tilde{\mathbf{H}}_t。$$

值得注意的是，更新门可以控制隐藏状态应该如何被包含当前时间步信息的候选隐藏状态所更新。假设更新门在时间步t'到 t（t'<t）之间一直近似 1。那么，在时间步t'到 t 之间的输入信息几乎没有流入时间步 t 的隐藏状态$\mathbf{H}_t$。 实际上，这可以看作是较早时刻的隐藏状态$\mathbf{H}_{t'-1}$ 一直通过时间保存并传递至当前时间步

t。 这个设计可以应对循环神经网络中的梯度衰减问题，并更好地捕捉时间序列中时间步距离较大的依赖关系。

GRU 总结：重置门有助于捕捉时间序列里短期的依赖关系。更新门有助于捕捉时间序列里长期的依赖关系。

LSTM（Long Short-Term Memory），长短期记忆。

它修改了循环神经网络隐藏状态的计算方式，并引入了与隐藏状态形状相同的记忆细胞。

三个门：输入门(Input gate)、遗忘门(Forget gate)、输出门(Output gate)。

假设隐藏单元个数为 h，给定时间步 t 的小批量输入$\mathbf{X}_t \in R^{n\times x}$（样本数为 n，输入维度为 x)，和上一时间步隐藏状态$\mathbf{H}_{t-1} \in R^{n\times h}$。时间步 t 的输入门$\mathbf{I}_t \in R^{n\times h}$、遗忘门$\mathbf{F}_t \in R^{n\times h}$和输出门$\mathbf{O}_t \in R^{n\times h}$分别计算如下：

$$\mathbf{I}_t = \sigma\left(\mathbf{X}_t\mathbf{W}_{xi} + \mathbf{H}_{t-1}\mathbf{W}_{hi} + \mathbf{b}_i\right)$$
$$\mathbf{F}_t = \sigma\left(\mathbf{X}_t\mathbf{W}_{xf} + \mathbf{H}_{t-1}\mathbf{W}_{hf} + \mathbf{b}_f\right)$$
$$\mathbf{O}_t = \sigma\left(\mathbf{X}_t\mathbf{W}_{xo} + \mathbf{H}_{t-1}\mathbf{W}_{ho} + \mathbf{b}_o\right)$$

其中的$\mathbf{W}_{xi}$, $\mathbf{W}_{xf}$, $\mathbf{W}_{xo} \in R^{x\times h}$和$\mathbf{W}_{hi}$, $\mathbf{W}_{hf}$, $\mathbf{W}_{ho} \in R^{h\times h}$是权重参数，$\mathbf{b}_i$, $\mathbf{b}_f$, $\mathbf{b}_o \in R^{1\times h}$是偏移参数。激活函数$\sigma$是 sigmoid 函数。输入门、遗忘门和输出门中每个元素的值域都是[0,1]。

和门控循环单元中的候选隐藏状态一样，时间步 t 的的候选记忆细胞$\tilde{\mathbf{C}}_t \in R^{n\times h}$也使用了值域在$[-1,1]$的 tanh 函数做激活函数。

$$\tilde{\mathbf{C}}_t = \tanh\left(\mathbf{X}_t\mathbf{W}_{xc} + \mathbf{H}_{t-1}\mathbf{W}_{hc} + \mathbf{b}_c\right)$$

其中$\mathbf{W}_{xc} \in R^{x\times h}$和$\mathbf{W}_{xc} \in R^{h\times h}$是权重参数，$\mathbf{b}_c \in R^{1\times h}$是偏移参数。

当前时间步记忆细胞$\mathbf{C}_t \in R^{n\times h}$的计算组合了上一时间步记忆细胞和当前时间步候选记忆细胞的信息，并通过遗忘门和输入门来控制信息的流动：

$$\mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{t-1} + \mathbf{I}_t \odot \tilde{\mathbf{C}}_t$$

需要注意的是，如果遗忘门一直近似 1 且输入门一直近似 0，过去的记忆细胞将一直通过时间保存并传递至当前时间步。 这个设计可以应对循环神经网络中的梯度衰减问题，并更好地捕捉时序数据中间隔较大的依赖关系。

有了记忆细胞以后，接下来我们还可以通过输出门来控制从记忆细胞到隐藏状态 $\mathbf{H}_t \in R^{n\times h}$ 的信息的流动：

$$\mathbf{H}_t = \mathbf{O}_t \odot \tanh\left(\mathbf{C}_t\right)$$

这里的 tanh 函数确保隐藏状态元素值在-1 到 1 之间。需要注意的是，当输出门近似 1，记忆细胞信息将传递到隐藏状态供输出层使用；当输出门近似 0，记忆细胞信息只自己保留。在时间步 t，长短期记忆的输出层计算和之前描述的循环神经网络输出层计算一样：我们只需将该时刻的隐藏状态 $\mathbf{H}_t$ 传递进输出层，从而计算时间步 t 的输出。

LSTM 总结：长短期记忆的可以应对循环神经网络中的梯度衰减问题，并更好地捕捉时序数据中间隔较大的依赖关系。

GRU 与 LSTM 对比分析：

a.性能上不分伯仲

b.GRU 参数更少因此更容易收敛，但是在数据集很大的情况下 LSTM 表现更好

c.从结构上来说，GRU 只有两个门，LSTM 有三个门，GRU 直接将隐藏状态传给下一个单元，而 LSTM 则用 memory cell 把隐藏状态包含起来传递信息

d.对记忆的控制：LSTM 用输出门控制，GRU 直接传递给下一个单元不做任何控制

e.LSTM 计算记忆细胞 $\mathbf{C}_t$ 时不对上一时刻的信息做任何控制而是用遗忘门独立地

实现这点，GRU 计算新的记忆单元$\tilde{\mathbf{H}}_t$时利用重置门对上一时刻的信息进行控制。

5.神经网络的优化算法。适用场景。

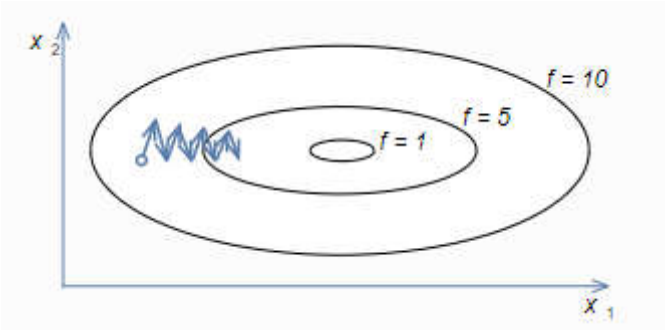
绝大多数深度学习中的目标函数都很复杂。因此，很多优化问题并不存在解析解，而需要使用基于数值方法的优化算法找到近似解。这类优化算法一般通过不断迭代更新解的数值来找到近似解。我们讨论的优化算法都是这类基于数值方法的算法。局部最小值点与鞍点。

最常见梯度下降法（GD），梯度下降每次迭代的计算开销随着样本数 n 线性增长，因此，当训练数据样本很大时，梯度下降每次的计算开销很高。因此我们使用随机梯度下降（SGD），由于$\nabla f(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n}\nabla f_i(\mathbf{x})$，我们可以每次取（随机均匀采样 i）其中的一份$\nabla f_i(\mathbf{x})$来迭代 x，$\mathbf{x} \leftarrow \mathbf{x} - \eta\nabla f_i(\mathbf{x})$ 。**SGD** 速度很快，在大数据集上收敛速度非常快，但是对于小样本速度一般，因为它可能偏离了一些方向。于是我们可以采用小批量的梯度下降法来综合 GD 与 SGD 的问题。每一次迭代可以随机均匀采样一个由训练数据样本索引所组成的**小批量（mini-batch）** B。因此我们可以使用$\nabla f_B(\mathbf{x}) = \frac{1}{|\mathbf{B}|}\sum_{i\in\mathbf{B}}\nabla f_i(\mathbf{x})$来迭代 x：

$\mathbf{x} \leftarrow \mathbf{x} - \eta\nabla f_{\mathbf{B}}(\mathbf{x})$。每次计算开销为$0(|\mathbf{B}|)$。当训练数据集的样本较少时，我们可以使用梯度下降；当样本较多时，我们可以使用小批量梯度下降并依据计算资源选择合适的批量大小。我们通常在每个迭代周期（epoch）开始前随机打乱数据集中样本的先后顺序，然后在同一个迭代周期中依次读取小批量的样本。

梯度下降的问题：给定目标函数，在梯度下降中，自变量的迭代方向仅仅取决于

自变量当前位置。这可能会带来一些问题。如下所示为一个等高线示意图。



由于目标函数在竖直方向（x2 轴方向）比在水平方向（x1 轴方向）更弯曲，给定学习率，梯度下降迭代自变量时会使自变量在竖直方向比在水平方向移动幅度更大。因此，我们需要一个较小的学习率从而避免自变量在竖直方向上越过目标函数最优解。然而，这造成了图中自变量向最优解移动较慢。

为解决该问题，**动量法(momentum)**应用而生。

我们对小批量随机梯度算法在每次迭代的步骤做如下修改：

$$\mathbf{v} \leftarrow \gamma \mathbf{v} + \eta \nabla f_{\mathrm{B}}(\mathbf{x})$$
$$\mathbf{x} \leftarrow \mathbf{x} - \mathbf{v}$$

其中，v 是速度变量，动量超参数γ满足$0 \le \gamma \le 1$。η为学习率。γ一般取 0.8 或 0.9.

动量法使用了指数加权移动平均的思想。

Adagrad 就是一个在迭代过程中不断自我调整学习率，并让模型参数中每个元素都使用不同学习率的优化算法。

Adagrad 的算法会使用一个小批量随机梯度按元素平方的累加变量 s，并将其中每个元素初始化为 0。在每次迭代中，首先计算小批量随机梯度 g，然后将该梯度按元素平方后累加到变量 s：$\mathbf{s} \leftarrow \mathbf{s} + \mathbf{g} \odot \mathbf{g}$

然后，我们将目标函数自变量中每个元素的学习率通过按元素运算重新调整一下：

$$\mathbf{g}' \leftarrow \frac{\eta}{\sqrt{\mathbf{s} + \varepsilon}} \odot \mathbf{g}$$

最后，自变量的迭代步骤与小批量随机梯度下降类似。只是这里梯度前的学习率已经被调整过了：

$$\mathbf{x} \leftarrow \mathbf{x} - \mathbf{g}'$$

由于 s 一直在累加按元素平方的梯度，自变量中每个元素的学习率在迭代过程中一直在降低（或不变）。所以，当学习率在迭代早期降得较快且当前解依然不佳时，Adagrad 在迭代后期由于学习率过小，可能较难找到一个有用的解。

RMSProp 对 Adagrad 中的第一步 s 的计算作了优化：

$$\mathbf{s} \leftarrow \gamma \mathbf{s} + (1 - \gamma)\, \mathbf{g} \odot \mathbf{g}$$

后续步骤与 Adagrad 步骤一致。自变量每个元素的学习率在迭代过程中避免了“直降不升”的问题。

Adam 算法使用了动量变量 v 和 RMSProp 中小批量随机梯度按元素平方的指数加权移动平均变量 s，并将它们中每个元素初始化为 0。

和动量法类似，给定超参数 β_1 且满足 $0 \le \beta_1 < 1$（算法作者建议设为 0.9），将小批量随机梯度的指数加权移动平均记作动量变量 v，并将它在时刻 t 的值记作 $\mathbf{v}_t$ ：

$$\mathbf{v}_t \leftarrow \beta_1 \mathbf{v}_{t-1} + (1 - \beta_1)\, \mathbf{g}_t$$

给定超参数 β_2 且满足 $0 \le \beta_2 < 1$（算法作者建议设为 0.999），将小批量随机梯度按元素平方后做指数加权移动平均得到 s，并将它在时刻 t 的值记作 $\mathbf{s}_t$ ：

$$\mathbf{s}_t \leftarrow \beta_2 \mathbf{s}_{t-1} + (1 - \beta_2)\, \mathbf{g}_t \odot \mathbf{g}_t$$

在 Adam 算法中，我们对变量 v 和 s 均作偏差修正：

$$\hat{\mathbf{v}}_t \leftarrow \frac{\mathbf{v}_t}{1-\beta_1^t}$$

$$\hat{\mathbf{s}}_t \leftarrow \frac{\mathbf{s}_t}{1-\beta_2^t}$$

将模型参数中每个元素的学习率通过按元素运算重新调整：

$$\mathbf{g}_t' \leftarrow \frac{\eta \hat{\mathbf{v}}_t}{\sqrt{\hat{\mathbf{s}}_t + \varepsilon}}$$

最后，时刻 t 的自变量的迭代步骤与小批量随机梯度下降类似：$\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} - \mathbf{g}_t'$

Adam 算法组合了动量法和 RMSProp，使用了偏差修正。

Batch Normalization（批量归一化），BN 就是通过一定的规范化手段，把每层神经网络任意神经元这个输入值的分布强行拉回到均值为 0 方差为 1 的标准正太分布，其实就是把越来越偏的分布强制拉回比较标准的分布，这样使得激活输入值落在非线性函数对输入比较敏感的区域，这样输入的小变化就会导致损失函数较大的变化，意思是这样让梯度变大，避免梯度消失问题产生，而且梯度变大意味着学习收敛速度快，能大大加快训练速度。

6.马尔可夫蒙特卡罗（MCMC）与变分推导（variational inference）。

MCMC：对于某概率分布 π，生成一个能够收敛到概率分布 π 的马尔可夫状态转移矩阵 P，则经过有限次迭代，一定可以得到概率分布 π。

- 根据马尔科夫过程的定义：$\pi(j)=\sum_{i=1}^{n}\pi(i)\cdot P(i,j)$
- 根据细致平稳条件：

$$\forall i,j,\pi(i)P(i,j)=\pi(j)P(j,i)$$

- 得：

$$\pi(j)=\sum_{i=1}^{n}\pi(j)\cdot P(j,i)$$

- 从而：

$$\pi=\pi\cdot P$$

- 假定当前马尔科夫过程的转移矩阵为Q，对于给定分布p，一般的说，$p(i)q(i,j)\neq p(j)q(j,i)$
- 通过加入因子α的方式，使得上式满足细致平稳条件 $p(i)q(i,j)\alpha(i,j)=p(j)q(j,i)\alpha(j,i)$
- 满足等式的因子α有很多，根据对称性，可以取：$\alpha(i,j)=p(j)q(j,i)$，$\alpha(j,i)=p(i)q(i,j)$
- 根据接受率α改造转移矩阵Q：

$$p(i)\underline{q(i,j)\alpha(i,j)}=p(j)\underline{q(j,i)\alpha(j,i)}$$

- 根据需要满足的细致平稳条件

$$p(i)q(i,j)\alpha(i,j)=p(j)q(j,i)\alpha(j,i)$$

- 若令α(j,i)=1，则有：$p(i)q(i,j)\alpha(i,j)=p(j)q(j,i)$
- 从而：

$$\alpha(i,j)=\frac{p(j)q(j,i)}{p(i)q(i,j)}$$

- 将接受率置为恒小于1，从而

$$\alpha(i,j)=\min\left(\frac{p(j)q(j,i)}{p(i)q(i,j)},1\right)$$

Metropolis-Hastings算法

- 初始化马尔科夫过程初始状态$I=i_0$
- 对t=0,1,2,3…
  - 第t时刻马尔科夫过程初始状态i_t,采样$q=q(j|i_t)$
  - 从均匀分布中采样$u \in [0,1]$
  - 如果 $u < \alpha(i,j) = \min\left(\frac{p(j)q(j,i)}{p(i)q(i,j)},1\right)$

    则接受状态j，即$i_{t+1}=j$

    否则，不接受状态j，即$i_{t+1}=i$

变分推导（variational inference）是一般的确定性的近似推导算法。基本思想：选择一个容易计算的近似分布 $q(x)$，它能够尽可能的接近真正的后验分布 $p(x \mid D)$ 。通过降低约束条件，在精度和速度上折中。

关键点：描述两个分布之间的相似度。------一个显然的损失函数是最小化 KL 散度。

$$KL(p^* \| q) = \sum_x p^*(x) \log \frac{p^*(x)}{q(x)} = E_{p^*(x)}\left(\log \frac{p^*(x)}{q(x)}\right)$$

- 上式关于后验概率p*的期望是不容易计算的，作为替代，将上述KL散度变成“逆KL散度”(reverse KL divergence)

$$KL(q \| p^*) = \sum_x q(x) \log \frac{q(x)}{p^*(x)} = E_{q(x)}\left(\log \frac{q(x)}{p^*(x)}\right)$$

- 第二个式子的主要优点是转换为计算关于q的期望(而q是关于未知参数的简单分布)；进一步，由于p(D)是归一化因子

$$p^*(x) = p(x \mid D) = \frac{p(x,D)}{p(D)} \overset{\Delta}{=\!=} \frac{\tilde{p}(x)}{Z} \Rightarrow \tilde{p}(x) = Z \cdot p^*(x)$$

- 上式变成：$J(q) = KL(q \| \tilde{p}) = \sum_x q(x) \log \frac{q(x)}{\tilde{p}(x)}$

新目标函数的可行性 $J(q)=KL(q\|\tilde{p})$

$$J(q)=KL(q\|\tilde{p})=\sum_x q(x)\log\frac{q(x)}{\tilde{p}(x)}$$
$$=\sum_x q(x)\log\frac{q(x)}{Z\cdot p^*(x)}$$
$$=\sum_x q(x)\log\frac{q(x)}{p^*(x)}+\sum_x q(x)\log\frac{1}{Z}$$
$$=\sum_x q(x)\log\frac{q(x)}{p^*(x)}-\log Z$$
$$=KL(q\|p^*(x))-\log Z$$

- 由于Z是常数，通过最小化J(q)，能够使得q接近p*。

- 定义能量 $E(x)=-\log\tilde{p}(x)$
- 目标函数是能量的期望减去系统的熵。J(q)被叫做“变分自由能”或“Helmholtz free energy”。

$$J(q)=KL(q\|\tilde{p})=\sum_x q(x)\log\frac{q(x)}{\tilde{p}(x)}$$
$$=E_q\left(\log\frac{q(x)}{\tilde{p}(x)}\right)=E_q(\log q(x)-\log\tilde{p}(x))$$
$$=E_q(\log q(x))+E_q(-\log\tilde{p}(x))$$
$$\overset{\triangle}{==}-H(X)+E_q(E(x))$$

平均场方法(Mean field method)

- 最流行的变分方法之一是平均场近似。在这种方法中，假定后验概率能够近似分解为若干因子的乘积。
  - 思考：无向图中的“最大团” Hammersley-Clifford定理

$$q(x)=\prod_i q_i(x_i)$$

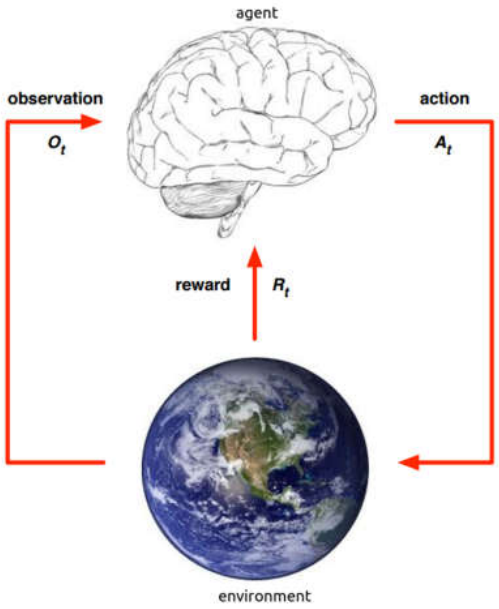
- 我们的目标是解决最优化问题：$\min_{q_1,\cdots q_D} KL(q\|p)$
- 平均场方法使得可以在若干边界分布qi上进行(依次)优化。事实上，很快将得知，有如下近似等式：

$$\log q_j(x_j)=E_{-q_j}[\log\tilde{p}(x)]+const$$

- 其中，未正则化的后验概率 $\tilde{p}(x)=p(x,D)$
- 关于除了x_j的所有其他变量的f(x)的期望 $E_{-q_j}[f(x)]$

7.强化学习。适用于哪些场景，基本要素，如何用强化学习对实际问题建模。

强化学习涉及智能体（agent）与环境的交互和序列化决策（sequential decision making）。



强化学习适用的场景：

a. 控制物理系统：行走、飞行、驾驶、游泳

b. 与用户进行交互：客户维护、个性化频道、用户体验优化

c. 解决物流问题：规划、带宽分配、电梯控制、认知无线电、电力优化

d. 玩游戏：扑克、围棋、星际、Dota2

e. 学习序列化算法：注意力、记忆、条件计算、激活

基本要素：

智能体(Agent)、环境(Environment)、奖励(Rewards)、状态(State)、动作(Action)

奖励（Rewards）：Agent 在 t 步时执行某个行动（action）A_t，环境会根据潜在的

奖励函数反馈给 Agent 一个奖励（reward）R_t，奖励R_t为一个标量反馈信号，用于反映 Agent 在 t 步时行动的好坏。

强化学习的任务：如何调整 Agent 的行动策略函数，来最大化积累奖励（cumulative reward）。

状态（State）：在 Agent 和环境进行交互的时候，会产生一连串的观测，动作和奖励，即历史（history）$H_t = O_1, R_1, A_1, \ldots, A_{t-1}, O_t, R_t$，将要发生的事情(Agent 要做什么动作、环境将产生什么观测和奖励)，完全取决于历史。而状态(state)就是决定将要发生事情的信息。通常状态是历史的函数：$S_t = f\left(H_t\right)$。PS：状态的马尔可夫性：$\mathrm{P}\left[S_{t+1} \mid S_t\right] = \mathrm{P}\left[S_{t+1} \mid S_1, \ldots, S_t\right]$，即给定现在，未来与过去独立。

智能体（Agent）的构成：

强化学习中的 Agent 通常涉及如下一个或多个部分，包括：

a. 策略（Policy）：决定智能体如何行动，智能体状态到行动的映射

   确定型策略$a = \pi\left(s\right)$，随机型策略$\pi\left(a \mid s\right) = \mathrm{P}\left(A_t = a \mid S_t = s\right)$

b. 值函数（Value function）：未来奖励的预测函数，用来评价 Agent 所处状态的好坏，可用于挑选下一步的行动

   $v_\pi\left(s\right) = \mathrm{E}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s\right]$

c. 模型（model）：智能体对环境的建模，模型可能有缺陷

   模型并不是环境本身，但是对于预测环境变化很有用处，这有利于 Agent 决定下一步的行动，但模型并不是构建 Agent 所必需的。

   模型通常涉及：

   ① 预测状态转移$P_{ss'}^a = \mathrm{P}\left[S_{t+1} = s' \mid S_t = s, A_t = a\right]$

② 预测即时奖赏 $R_{ss'}^{a} = \mathrm{P}\left[R_{t+1} \mid S_t = s, A_t = a\right]$

智能体的分类：

a. 基于值函数的智能体（Value Based）仅有值函数

b. 基于策略函数的智能体（Policy Based）仅有策略函数

c. 演员-批评者（Actor Critic）既有值函数也有策略函数

d. 有模型的智能体（Model Based）

e. 无模型的智能体（Model Free）

强化学习类似于一个试错的学习，Agent 需要从其与环境的交互中发现一个好的策略，同时又不至于在试错的过程中丢失太多的奖励。探索和利用是 Agent 进行决策时需要平衡的两个方面，探索：用于获取更多的环境信息，利用：利用已获取的环境信息来最大化奖赏。强化学习中的两个相关联任务：预测：给定策略函数，对该策略的未来表现进行评估，控制：寻找最优的策略函数，使得未来所获得的累积奖励最大化。几乎所有的强化学习问题都可以用 MDPs（马尔可夫决策过程）来刻画。

对实际问题建模举例：a=动作，p=转移矩阵中的元素，r=奖励。

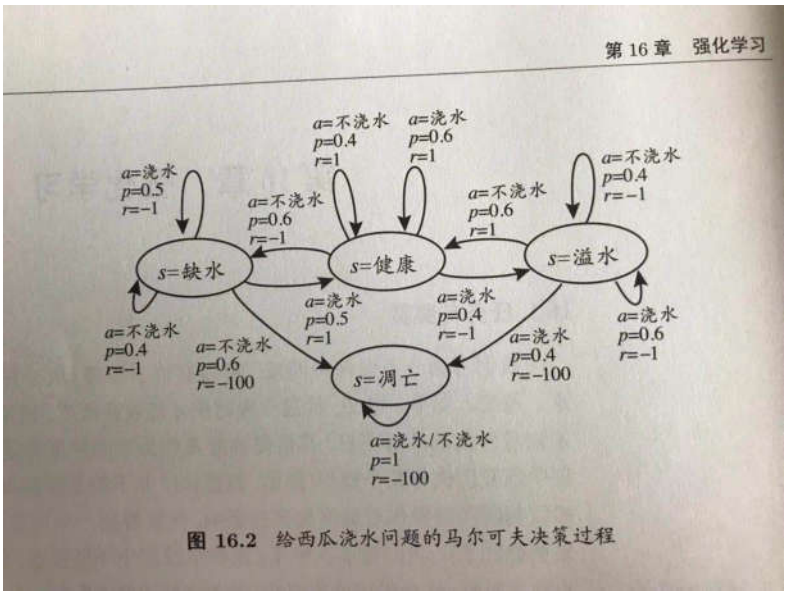


图 16.2 给西瓜浇水问题的马尔可夫决策过程

对实际的建模属于 Open 问题，更多灵感可以参考贝尔曼方程、Q-Learning 算法等。

8.给定实际问题，利用 CNN 或 LSTM 等涉及一个可行网络，如何训练及测试。

Open 问题，可以参考经典网络 AlexNet 的整个流程。