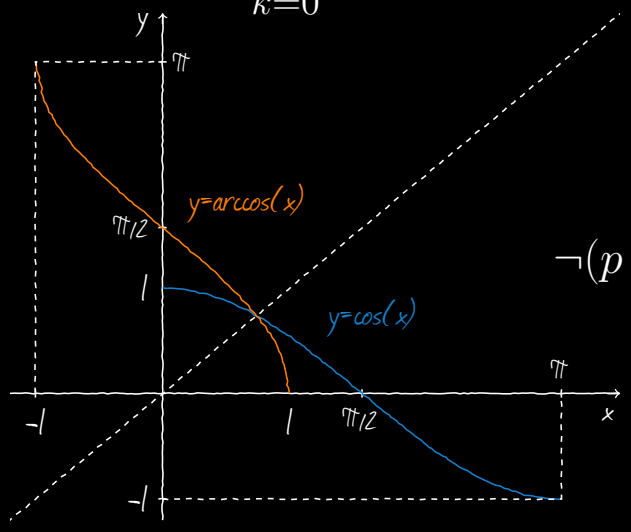
学习笔记

深度学习：从 MLP 到 GNN

李开运

version 1.1

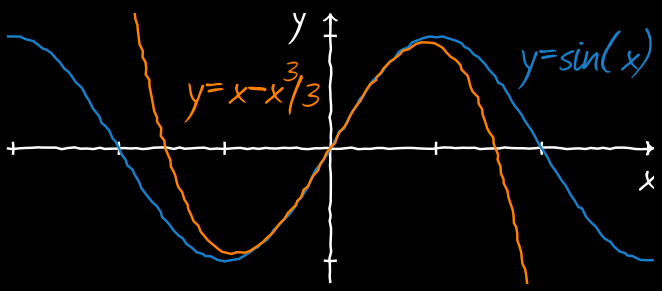
$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k}$$



$$\zeta_k = |a|^{1/n} \mathrm{e}^{i(\arg(a)+2k\pi)/n}$$

$$\mathrm{e}^{i\pi} + 1 = 0$$

$$\neg(p \vee q) \equiv (\neg p) \wedge (\neg q)$$



$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Contents

Contents

引言部分

Part I

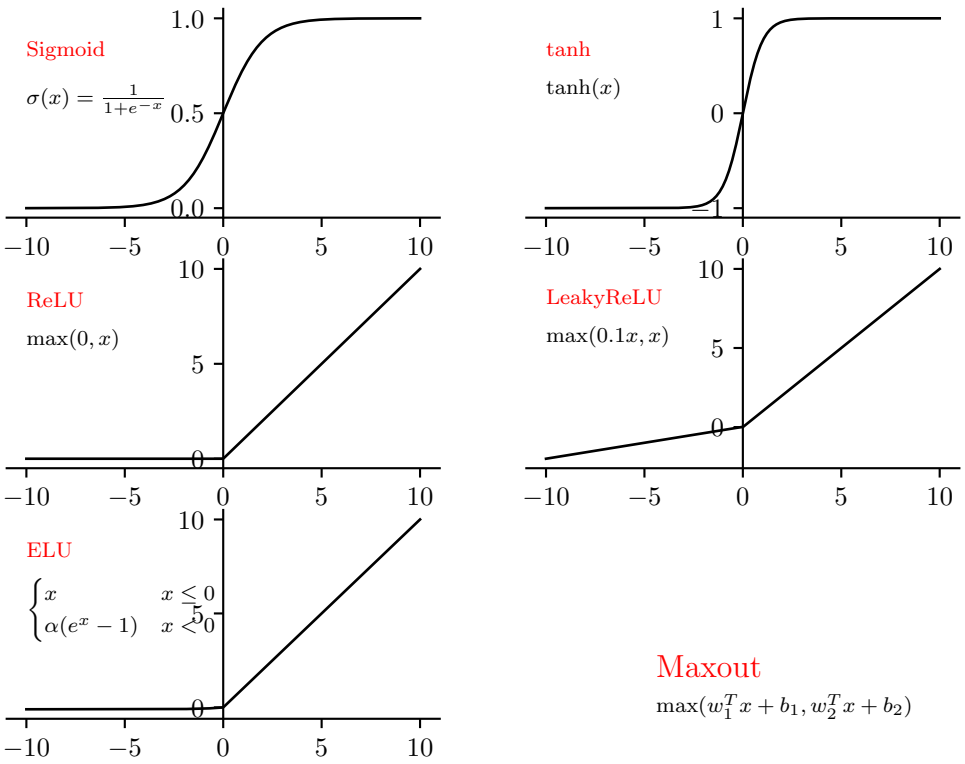
深度学习基础

2.1 多层感知机

test

2.2 激活函数

激活函数的作用是为模型引入非线性，提高模型的表达能力。



2.3 损失函数

损失函数 (loss function) 是用来估量模型的预测值 $f(x)$ 与真实值 Y 的不一致程度，损失函数越小，一般就代表模型的鲁棒性越好，正是损失函数指导了模型的学习。

2.4 优化算法

梯度下降是目前神经网络使用最为广泛的优化算法之一。可以用下述步骤描述梯度下降的流程：

1. 计算目标函数关于参数的梯度

$$\boldsymbol{g}_t = \nabla_\theta J(\theta) \tag{2.1}$$

2. 根据历史梯度计算一阶和二阶动量

$$\begin{aligned} \boldsymbol{m}_t &= \phi(\boldsymbol{g}_1, \boldsymbol{g}_2, \ldots, \boldsymbol{g}_t) \\ \boldsymbol{v}_t &= \psi(\boldsymbol{g}_1, \boldsymbol{g}_2, \ldots, \boldsymbol{g}_t) \end{aligned} \tag{2.2}$$

3. 更新模型参数

$$\theta_{t+1} = \theta_t - \frac{1}{\sqrt{v_t + \epsilon}} \boldsymbol{m}_t \tag{2.3}$$

其中，ϵ 为平滑项，防止分母为零，通常取 1e-8。

2.4.1 小批量随机梯度下降

小批量随机梯度下降在每次迭代中随机均匀采样多个样本来组成一个小批量，然后使用这个小批量来计算梯度。没有动量的概念，即 $\boldsymbol{m}_t = \eta \boldsymbol{g}_t, \boldsymbol{v}_t = I^2, \epsilon = 0$

$$\begin{aligned} \boldsymbol{g}_t \leftarrow \nabla J_{\mathcal{B}_t}(\boldsymbol{x}_{t-1}) &= \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}_t} \nabla J_i(\boldsymbol{x}_{t-1}) \\ \theta_{t+1} &= \theta_t - \eta \boldsymbol{g}_t \end{aligned} \tag{2.4}$$

SGD 的缺点在于收敛速度慢，可能在鞍点处震荡。并且，如何合理的选择学习率是 SGD 的一大难点。

2.4.2 动量法

SGD 在遇到沟壑时容易陷入震荡。为此，可以为其引入动量 Momentum，加速 SGD 在正确方向的下降并抑制震荡。

$$\begin{aligned} \boldsymbol{m}_t &= \gamma \boldsymbol{m}_{t-1} + \eta \boldsymbol{g}_t \\ \theta_{t+1} &= \theta_t - \boldsymbol{m}_t \end{aligned} \tag{2.5}$$

这意味着参数更新方向不仅由当前的梯度决定，也与此前累积的下降方向有关。这使得参数中那些梯度方向变化不大的维度可以更新，并减少梯度方向变较大的维度上的更新幅度。由此产生了加速收敛和减小震荡的效果。

动量法使用了指数加权移动平均的思想。它将过去时间步的梯度做了加权平均，且权重按时间步指数衰减。动量法使得相邻时间步的自变量更新在方向上更加一致。

2.4.3 AdaGrad 算法

深度学习模型中往往涉及大量的参数，不同参数的更新频率往往有所区别。对于更新不频繁的参数，我们希望单次步长更大，多学习一些知识；对于更新频率的参数，我们则希望步长较小，使得学习到的参数更稳定，不至于被单个样本影响太多。

Adagrad 算法引入了二阶动量：

$$\begin{aligned} \boldsymbol{v}_t &= \operatorname{diag}\left(\sum_{i=1}^{t} \boldsymbol{g}_{i,1}^2, \sum_{i=1}^{t} \boldsymbol{g}_{i,2}^2, \ldots, \sum_{i=1}^{t} \boldsymbol{g}_{i,d}^2\right) \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{v_t + \epsilon}} \boldsymbol{g}_t \end{aligned} \tag{2.6}$$

AdaGrad 算法在迭代过程中不断调整学习率，并让目标函数自变量中每个元素都分别拥有自己的学习率。使用 AdaGrad 算法时，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)。这一方法在稀疏数据的场景下表现很好。

2.5 RMSProp 算法

在 Adagrad 中，v_t 是单增的，使得学习率逐渐递减至 0，可能导致训练过程提前结束。为了改进这一缺点，可以考虑在计算二阶动量时不累积全部历史梯度，而只关注最近某一时间窗口内的下降梯度。根据此思想有了 RMSprop。有

$$\begin{aligned} v_t &= \gamma v_{t-1} + (1-\gamma)\cdot \mathrm{diag}(g_t \odot g_t) \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{v_t+\epsilon}} \boldsymbol{g}_t \end{aligned} \tag{2.7}$$

其二阶动量采用指数移动平均公式计算，这样可避免二阶动量持续累积的问题。

2.5.1 Adam 算法

Adam 可以认为是 RMSprop 和 Momentum 的结合。和 RMSprop 对二阶动量使用指数移动平均类似，Adam 中对一阶动量也是用指数移动平均计算。

$$\begin{aligned} \boldsymbol{m}_t &= \eta[\beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t] \\ \boldsymbol{v}_t &= \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)\cdot \mathrm{diag}(\boldsymbol{g}_t \odot \boldsymbol{g}_t) \end{aligned} \tag{2.8}$$

其中，初值 $\boldsymbol{m}_0 = 0, \boldsymbol{v}_0 = 0$。注意到，在迭代初始阶段，$\boldsymbol{m}_t, \boldsymbol{v}_t$ 有一个向初值的偏移 (过多地偏向 0)。因此，可以做偏置校正

$$\begin{aligned} \hat{\boldsymbol{m}}_t &= \frac{\boldsymbol{m}_t}{1-\beta_1^t} \\ \hat{\boldsymbol{v}}_t &= \frac{\boldsymbol{v}_t}{1-\beta_2^t} \end{aligned} \tag{2.9}$$

再更新

$$\theta_{t+1} = \theta_t - \frac{1}{\sqrt{\hat{\boldsymbol{v}}_t+\epsilon}} \hat{\boldsymbol{m}}_t \tag{2.10}$$

2.6 反向传播

前向传递输入信号直至产生误差，反向传播误差信息更新权重矩阵。误差反向传播的目标是寻找一计算前馈神经网络的误差函数 $E(\boldsymbol{w})$ 的梯度的一种高效的方法。
许多实际应用中使用的误差函数，例如针对一组独立同分布的数据的最大似然方法定义的误差函数，由若干的求和式组成，每一项对应于训练集的一个数据点，即

$$E(\boldsymbol{w}) = \sum_{n=1}^{N} E_n(\boldsymbol{w}) \tag{2.11}$$

这里，我们要考虑的是计算 $\nabla E_n(\boldsymbol{w})$ 的问题。这可以使用顺序优化的方法计算，或者使用批处理方法在训练集上进行累加。

首先考虑一个简单的线性模型，其中输出 y_k 是输入变量 x_i 的线性组合，即，

$$y_k = \sum_i w_{ki} x_i \tag{2.12}$$

对于一个特定的输入模式 n，误差函数的形式为

$$E_n = \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2 \tag{2.13}$$

其中 $y_{nk} = y_k(\boldsymbol{x}_n, \boldsymbol{w})$。这个误差函数关于一个权值 w_{ji} 的梯度为

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj}) x_{ni} \tag{2.14}$$

它可以表示为链接 w_{ji} 的输出端相关联的“误差信号” $y_{nj} - t_{nj}$ 和与链接的输入端相关联的变量 x_{nj} 的乘积。反向传播算法可以总结如下

1. 对于网络的一个输入向量 $\boldsymbol{x}_n$，使用下列进行正向传播，找到所有隐含单元和输出单元的激活。

$$a_j = \sum_i w_{ji} z_i \tag{2.15}$$

$$z_j = h(a_j) \tag{2.16}$$

   其中 z_i 是一个单元的激活，或者是输入。它向单元 j 发送一个链接，w_{ji} 是与这个链接关联的权值。

2. 计算所有输出单元的 δ_k

$$\delta_k = y_k - t_k \tag{2.17}$$

3. 获得网络中所有隐含单元的 δ_j

$$\begin{aligned} \delta_j \equiv \frac{\partial E_n}{\partial a_j} &= \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j} \\ &= h^{'}(a_j) \sum_k w_{kj} \delta_k \end{aligned} \tag{2.18}$$

   其中求和式的作用对象是所有向单元 j 发送链接的单元 k。注意，单元 k 可以包含其他的隐含单元和输出单元。

4. 计算导数

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i \tag{2.19}$$

对于批处理方法，总误差函数 E 的导数可以通过下面的方式得到：对于训练集里的每个模式，重复上面的步骤，然后对所有的模式求和，即

$$\frac{\partial E}{\partial w_{ji}} = \sum_n \frac{\partial E_n}{\partial w_{ji}} \tag{2.20}$$

上面的推导中，我们隐式地假设网络中的每个隐含单元或输入单元相同的激活函数 $h(\cdot)$。

Part II

深度模型架构

3.1 CNN

3.1.1 卷积

信号处理中，卷积被定义为：一个函数经过翻转和移动后与另一个函数的乘积的积分.

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t - tau)\mathrm{d}\tau \tag{3.1}$$

在深度学习中的卷积本质上是信号处理中的互相关 (cross-correlation) 操作。执行卷积的目的是从输入中提取有用的特征。卷积神经网络通过卷积在训练期间使用自动学习权重的函数来提取特征，所有这些提取出来的特征之后会被组合在一起做出决策。

1. 2D 卷积，如图1

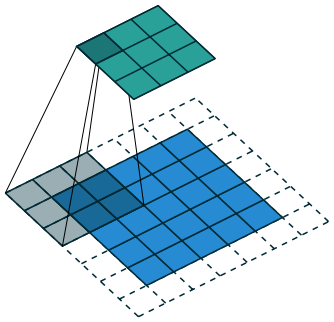


Figure 3.1: 2D 卷积

2. 多通道卷积 (空间卷积)，如图2

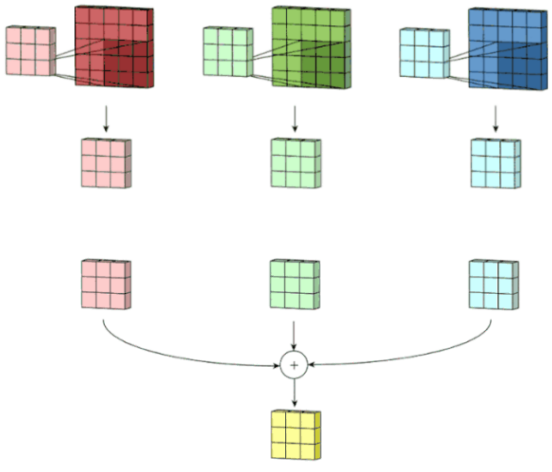


Figure 3.2: 多通道卷积

3. 3D 卷积，上面两种虽然实现了空间卷积，但是本质上还是 2D 卷积。而在 3D 卷积中，过滤器的深度要比输入层的深度要小，结果是，3D 过滤器可以沿着 3 个方向移动。输出的数值同样也以 3D 空间的形式呈现，最终输出一个 3D 数据。如图3.

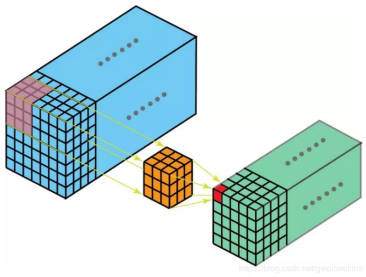


Figure 3.3: 3D 卷积

4. 1×1 卷积，计算上等于各个通道的加权和，同时可以改变特征通道数。如图4

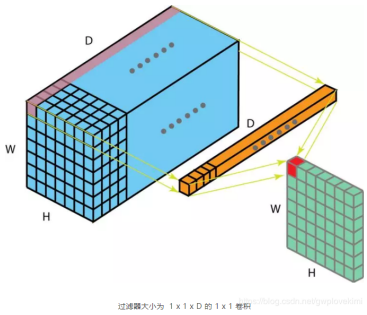


Figure 3.4: 1×1 卷积

5. 空间可分离卷积 (Depthwise Separable Convolutions)，我们可以将可分离卷积操作拆成多个步骤。减少计算量。如图5

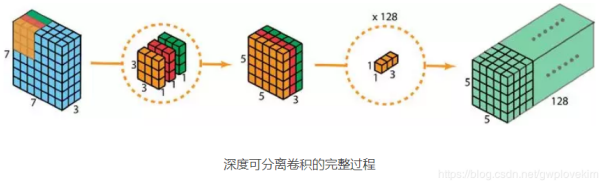


Figure 3.5: 空间可分离卷积

6. 分组卷积 (Group Convolutions)，过滤器被拆分为不同中的组，每一个组负责具有一定深度的传统 2D 卷积的工作。另外，分组卷积也许有提供比标准完整 2D 卷积更好的模型。原因和稀疏过滤器有关。如图6

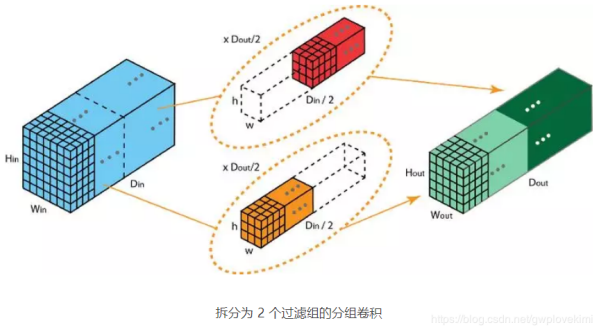


Figure 3.6: 分组卷积

7. 扩张卷积 (空洞卷积 Dilated Convolutions)，引入另一个卷积层的参数被称为扩张率。这定义了内核中值之间的间距。扩张率为 2 的 3×3 的内核具有与 5×5 的内核相同的视野，而只使用 9 个参数。系统能以相同的计算成本，提供更大的感受野。如图7

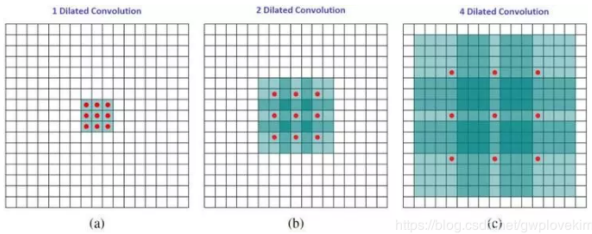


Figure 3.7: 扩张卷积

8. 反卷积 (转置卷积 Transposed Convolutions)，转置卷积并非信号处理中的卷积运算的逆运算。对于很多网络架构的很多应用言，我们往往需要进行与普通卷积方向相反的转换，即我们希望执行上采样。此时可以使用转置卷积操作。如图8

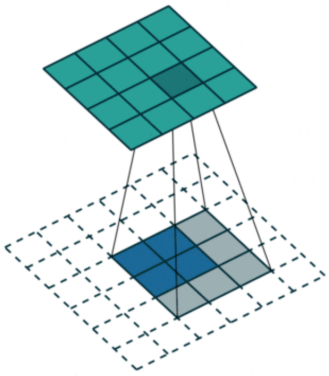


Figure 3.8: 转置卷积

3.1.2 卷积层的前向传播

为简单起见，考虑单通道的情况

$$\boldsymbol{X} * \boldsymbol{K} = \boldsymbol{Y} \tag{3.2}$$

即

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{pmatrix} * \begin{pmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{pmatrix} = \begin{pmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{pmatrix} \tag{3.3}$$

这里

$$\begin{aligned} \begin{pmatrix} y_{11} \\ y_{12} \\ y_{21} \\ y_{22} \end{pmatrix} &= \begin{pmatrix} x_{11}k_{11} + x_{12}k_{12} + x_{21}k_{21} + x_{22}k_{22} \\ x_{12}k_{11} + x_{13}k_{12} + x_{22}k_{21} + x_{23}k_{22} \\ x_{21}k_{11} + x_{22}k_{12} + x_{31}k_{21} + x_{32}k_{22} \\ x_{22}k_{11} + x_{23}k_{12} + x_{32}k_{21} + x_{33}k_{22} \end{pmatrix} \\ &= \begin{pmatrix} x_{11} & x_{12} & x_{21} & x_{22} \\ x_{12} & x_{13} & x_{22} & x_{23} \\ x_{21} & x_{22} & x_{31} & x_{32} \\ x_{22} & x_{23} & x_{32} & x_{33} \end{pmatrix} \cdot \begin{pmatrix} k_{11} \\ k_{12} \\ k_{21} \\ k_{22} \end{pmatrix} \end{aligned} \tag{3.4}$$

所以，卷积运算最终转化为矩阵运算。需要对原始的 $\boldsymbol{X}, \boldsymbol{K}, \boldsymbol{Y}$ 进行变形操作，相应的记作 $\boldsymbol{XC}, \boldsymbol{KC}, \boldsymbol{YC}$。多通道的情况只需要在维度上将操作扩展即可。

3.1.3 卷积层的反向传播

分析 δ 误差反向传播过程可以简单的记忆为：如果神经网络 $l+1$ 层某个结点的 δ 误差要传到 l 层，我们就去找到前向传播时 $l+1$ 层的这个结点和第 l 层的哪些结点有关系，权重是多少，那么反向传播时，δ 误差就会乘上相同的权重传播回来。

为了书写方便，记

$$\begin{pmatrix} \delta_{11} \\ \delta_{12} \\ \delta_{21} \\ \delta_{22} \end{pmatrix} = \nabla \boldsymbol{YC} = \begin{pmatrix} \nabla y_{11} \\ \nabla y_{12} \\ \nabla y_{21} \\ \nabla y_{22} \end{pmatrix} \tag{3.5}$$

在反向传播中，δ 是从后面一层（一般是激活函数层或池化层）传过来的，是一个已知量，在此基础上求 $\nabla \boldsymbol{K}, \nabla \boldsymbol{X}$

1. 求 $\nabla \boldsymbol{K}$

$$\nabla \boldsymbol{KC} = \boldsymbol{XC}^T \cdot \nabla \boldsymbol{YC} \tag{3.6}$$

$\nabla \boldsymbol{KC}$ 只要 reshape 一下就可以得到 $\nabla \boldsymbol{K}$

2. 求 $\nabla \boldsymbol{X}$

根据反向传播公式，

$$\nabla \boldsymbol{XC} = \nabla \boldsymbol{YC} \cdot \boldsymbol{KC}^T \tag{3.7}$$

但是，从 $\nabla \boldsymbol{XC}$ 还原到 $\nabla \boldsymbol{X}$ 不是一件容易的事，所以考虑新的计算方式。

根据前向传播

$$\begin{pmatrix} y_{11} \\ y_{12} \\ y_{21} \\ y_{22} \end{pmatrix} = \begin{pmatrix} x_{11}k_{11} + x_{12}k_{12} + x_{21}k_{21} + x_{22}k_{22} \\ x_{12}k_{11} + x_{13}k_{12} + x_{22}k_{21} + x_{23}k_{22} \\ x_{21}k_{11} + x_{22}k_{12} + x_{31}k_{21} + x_{32}k_{22} \\ x_{22}k_{11} + x_{23}k_{12} + x_{32}k_{21} + x_{33}k_{22} \end{pmatrix} \tag{3.8}$$

可以计算每个 x_{ij} 的导数

$$\begin{aligned} \begin{pmatrix} \nabla x_{11} \\ \nabla x_{12} \\ \nabla x_{13} \\ \nabla x_{21} \\ \nabla x_{22} \\ \nabla x_{23} \\ \nabla x_{31} \\ \nabla x_{32} \\ \nabla x_{33} \end{pmatrix} &= \begin{pmatrix} k_{22}0 + k_{21}0 + K_{12}0 + k_{11}\delta_{11} \\ k_{22}0 + k_{21}0 + K_{12}\delta_{11} + k_{11}\delta_{12} \\ k_{22}0 + k_{21}0 + K_{12}\delta_{12} + k_{11}0 \\ k_{22}0 + k_{21}\delta_{11} + K_{12}0 + k_{11}\delta_{21} \\ k_{22}\delta_{11} + k_{21}\delta_{12} + K_{12}\delta_{21} + k_{11}\delta_{22} \\ k_{22}\delta_{12} + k_{21}0 + K_{12}\delta_{22} + k_{11}0 \\ k_{22}0 + k_{21}0 + K_{12}\delta_{21} + k_{11}0 \\ k_{22}\delta_{21} + k_{21}\delta_{22} + K_{12}0 + k_{11}0 \\ k_{22}\delta_{22} + k_{21}0 + K_{12}0 + k_{11}0 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 0 & 0 & \delta_{11} \\ 0 & 0 & \delta_{11} & \delta_{12} \\ 0 & 0 & \delta_{12} & 0 \\ 0 & \delta_{11} & 0 & \delta_{21} \\ \delta_{11} & \delta_{12} & \delta_{21} & \delta_{22} \\ \delta_{12} & 0 & \delta_{22} & 0 \\ 0 & \delta_{21} & 0 & 0 \\ \delta_{21} & \delta_{22} & 0 & 0 \\ \delta_{22} & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} k_{11} \\ k_{12} \\ k_{21} \\ k_{22} \end{pmatrix} \end{aligned} \tag{3.9}$$

设上面三个矩阵分别为 $\nabla \boldsymbol{X}', \nabla \boldsymbol{Y}', \nabla \boldsymbol{K}'$，即 $\nabla \boldsymbol{X}' = \nabla \boldsymbol{Y}' \cdot \nabla \boldsymbol{K}'$。从而可见

$$\nabla \boldsymbol{X} = \begin{pmatrix} \nabla x_{11} & \nabla x_{12} & \nabla x_{13} \\ \nabla x_{21} & \nabla x_{22} & \nabla x_{23} \\ \nabla x_{31} & \nabla x_{32} & \nabla x_{33} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & \delta_{11} & \delta_{12} & 0 \\ 0 & \delta_{21} & \delta_{22} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} * \begin{pmatrix} k_{22} & k_{21} \\ k_{12} & k_{11} \end{pmatrix} \tag{3.10}$$

这是一个卷积运算。不同的是对 $\nabla \boldsymbol{Y}$ 进行卷积，从后向前卷积，有的文章称为逆向卷积。

3.1.4 池化层及池化层的反向传播

5.1 RNN

5.1.1 RNN 前向传播

简单起见，我们考虑一个无偏差项的循环神经网络，且激活函数为恒等映射（$\phi(x) = x$）。设时间步 t 的输入为单样本 $\boldsymbol{x}_t \in \mathbb{R}^d$，标签为 y_t，那么隐藏状态 $\boldsymbol{h}_t \in \mathbb{R}^h$ 的计算表达式为

$$\boldsymbol{h}_t = \boldsymbol{W}_{hx}\boldsymbol{x}_t + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1},$$

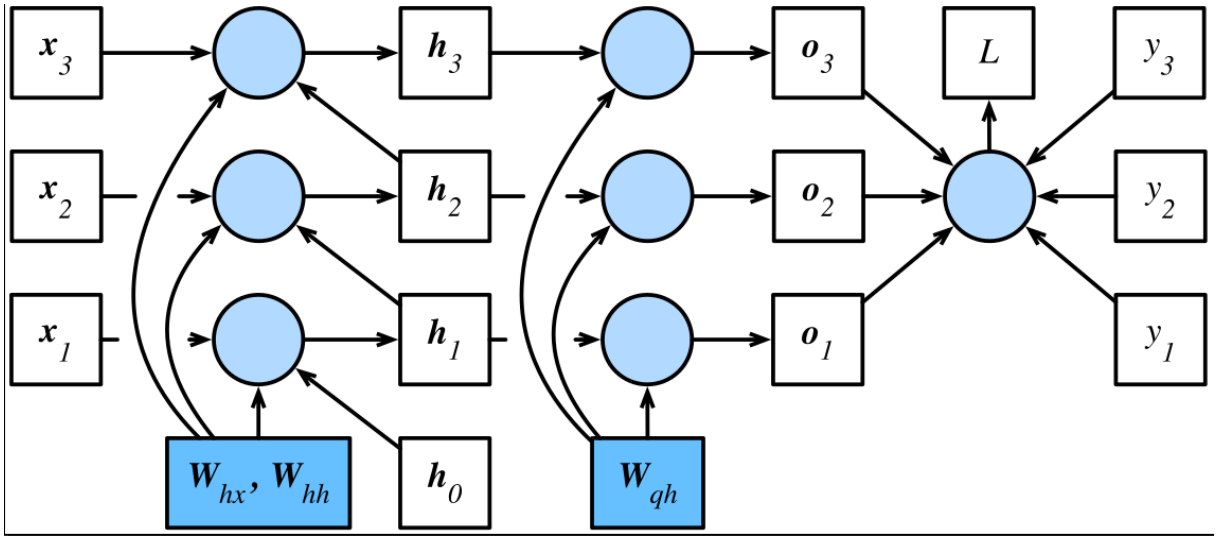
其中 $\boldsymbol{W}_{hx} \in \mathbb{R}^{h\times d}$ 和 $\boldsymbol{W}_{hh} \in \mathbb{R}^{h\times h}$ 是隐藏层权重参数。设输出层权重参数 $\boldsymbol{W}_{qh} \in \mathbb{R}^{q\times h}$，时间步 t 的输出层变量 $\boldsymbol{o}_t \in \mathbb{R}^q$ 计算为

$$\boldsymbol{o}_t = \boldsymbol{W}_{qh}\boldsymbol{h}_t.$$

设时间步 t 的损失为 $\ell(\boldsymbol{o}_t, y_t)$。时间步数为 T 的损失函数 L 定义为

$$L = \frac{1}{T}\sum_{t=1}^{T} \ell(\boldsymbol{o}_t, y_t).$$

我们将 L 称为有关给定时间步的数据样本的目标函数。



5.1.2 通过时间反向传播

模型的参数是 $\boldsymbol{W}_{hx}$, $\boldsymbol{W}_{hh}$ 和 $\boldsymbol{W}_{qh}$。链式法则的运算符用 prod 表示。

1. $\partial L/\partial \boldsymbol{W}_{qh}$

$$\frac{\partial L}{\partial \boldsymbol{W}_{qh}} = \sum_{t=1}^{T} \text{prod}\left(\frac{\partial L}{\partial \boldsymbol{o}_t}, \frac{\partial \boldsymbol{o}_t}{\partial \boldsymbol{W}_{qh}}\right) = \sum_{t=1}^{T} \frac{\partial L}{\partial \boldsymbol{o}_t}\boldsymbol{h}_t^\top. \tag{5.1}$$

2. $\partial L/\partial \boldsymbol{W}_{hx}$

$$\frac{\partial L}{\partial \boldsymbol{W}_{hx}} = \sum_{t=1}^{T} \text{prod}\left(\frac{\partial L}{\partial \boldsymbol{h}_t}, \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{W}_{hx}}\right) = \sum_{t=1}^{T} \frac{\partial L}{\partial \boldsymbol{h}_t}\boldsymbol{x}_t^\top \tag{5.2}$$

3. $\partial L/\partial \boldsymbol{W}_{hh}$

$$\frac{\partial L}{\partial \boldsymbol{W}_{hh}} = \sum_{t=1}^{T} \text{prod}\left(\frac{\partial L}{\partial \boldsymbol{h}_t}, \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{W}_{hh}}\right) = \sum_{t=1}^{T} \frac{\partial L}{\partial \boldsymbol{h}_t} \boldsymbol{h}_{t-1}^{\top}. \tag{5.3}$$

其中，目标函数有关各时间步输出层变量的梯度 $\partial L/\partial \boldsymbol{o}_t \in \mathbb{R}^q$ 很容易计算：

$$\frac{\partial L}{\partial \boldsymbol{o}_t} = \frac{\partial \ell(\boldsymbol{o}_t, y_t)}{T \cdot \partial \boldsymbol{o}_t}. \tag{5.4}$$

目标函数有关时间步隐藏状态的梯度 $\partial L/\partial \boldsymbol{h}_t \in \mathbb{R}^h$。依据链式法则，我们得到

$$\frac{\partial L}{\partial \boldsymbol{h}_t} = \sum_{i=t}^{T} \left(\boldsymbol{W}_{hh}^{\top}\right)^{T-i} \boldsymbol{W}_{qh}^{\top} \frac{\partial L}{\partial \boldsymbol{o}_{T+t-i}}. \tag{5.5}$$

由上式中的指数项可见，当时间步数 T 较大或者时间步 t 较小时，目标函数有关隐藏状态的梯度较容易出现衰减和爆炸。这也会影响其他包含 $\partial L/\partial \boldsymbol{h}_t$ 项的梯度。

6.1 GNN

Part III

深度学习应用

6.2 计算机视觉

6.3 自然语言处理

6.4 社会计算