CV-Learning Notes

张荐科 2022.1

一.基础算法

1.矩阵演算重要定义

梯度：

$$对m*n矩阵A，\ f:\mathbb{R}^{m*n}\rightarrow R$$

$$\frac{\partial f(A)}{\partial A}=\nabla_A f(A)=\begin{bmatrix}\frac{\partial f(A)}{\partial A_{11}} & \frac{\partial f(A)}{\partial A_{12}} & \cdots & \frac{\partial f(A)}{\partial A_{1n}}\\ \frac{\partial f(A)}{\partial A_{21}} & \frac{\partial f(A)}{\partial A_{22}} & \cdots & \frac{\partial f(A)}{\partial A_{2n}}\\ \vdots & \vdots & \ddots & \vdots\\ \frac{\partial f(A)}{\partial A_{m1}} & \frac{\partial f(A)}{\partial A_{m2}} & \cdots & \frac{\partial f(A)}{\partial A_{mn}}\end{bmatrix}$$

向量梯度即矩阵梯度的特例：

$$对向量x：$$

$$\nabla_x f(x)=\begin{bmatrix}\frac{\partial f(x)}{\partial x_1}\\ \frac{\partial f(x)}{\partial x_2}\\ \vdots\\ \frac{\partial f(x)}{\partial x_n}\end{bmatrix}$$

一些结论：

$$\frac{\partial b^Tx}{\partial x}=b$$

$$\frac{\partial a^TXb}{\partial X}=ab^T$$

$$\frac{\partial a^TX^Tb}{\partial X}=ba^T$$

$$\frac{\partial x^TBx}{\partial x}=(B+B^T)x$$

Hessian矩阵：

对n维向量x，对应$Hessian$矩阵为$n*n$矩阵，记为：

$$\nabla_x^2 f(x) = \begin{bmatrix} \frac{\partial^2 f(x)}{\partial x_1^2} & \frac{\partial^2 f(x)}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(x)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(x)}{\partial x_2^2} & \cdots & \frac{\partial^2 f(x)}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \frac{\partial^2 f(x)}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_n^2} \end{bmatrix}$$

重要结论：对$f(x) = x^T Ax$,其中x为n维向量，A为n阶方阵，有

$$\nabla_x^2 f(x) = 2A$$

2.线性系统

定义：将图像矩阵进行线性变换，记为：

$$S(f(m,n)) = g(m,n)$$

线性系统要求每个新的像素是原来像素的加权和，并且每个像素使用相同的权重集

为了更好地描述线性系统，引入脉冲响应，在某一处取1，其他位置为0

二维δ：$\delta_2[m,n] = \begin{cases} 1 & m=0且n=0 \\ 0 & 其他 \end{cases}$

常用的“线性位移不变系统”-LSI：整体移动输入也会等量地移动输出，可以用脉冲响应的线性组合进行表示。

二维卷积运算：具有交换性

$$f[n,m] * *h[n,m] = \sum_k \sum_l f[k,l] \cdot h[n-k, m-l]$$

对于二维信号，可以使用卷积处理一个线性、移动不变的系统中：

对输入信号x：经线性移动不变系统处理后输出可表为：

$y[n,m] = \Sigma_{i=-\infty}^{\infty} \Sigma_{j=-\infty}^{\infty} x[i,j] h[n-i, m-j]$

其中h为信号x与脉冲响应的二维卷积

3.边缘检测

一维离散导数

- 后向

$$\frac{df}{dx} = f(x) - f(x-1) = f'(x)$$

- 前向

$$\frac{df}{dx} = f(x) - f(x+1) = f'(x)$$

- 中心

$$\frac{df}{dx}=\frac{f(x)-f(x-1)}{2}=f'(x)$$

二维离散导数（梯度）

$$\nabla f(x,y)=\begin{bmatrix}f'_x(x,y)\\f'_y(x,y)\end{bmatrix}$$

$$|\nabla f(x,y)|=\sqrt{f'^2_x+f'^2_y}$$

通过梯度可以大致判断边缘特征：在垂直边缘处，x方向导数相较于y方向大很多；在竖直边缘处，y方向导数很大相较于x方向大很多

基于梯度的一种滤波方法：每个点用该处的梯度的模代替（可适当放大缩小）

- 该方法可用于提取图像的轮廓线，舍去平滑的细节
- 无法对有过多噪声的图像进行处理
- 为了解决这个问题，一般需要先对图像进行平滑化处理，即模糊滤波过程，如高斯模糊，使用二维高斯卷积核处理图像即可

Canny边缘检测器

- Sobel算子（一般可用于替代求某点梯度的两个分量，sobel梯度幅度图像更加粗大明亮）

$$\text{使用}3*3\text{的卷积核}(\text{平滑}*\text{微分})$$

$$G_x=\begin{bmatrix}1&0&-1\\2&0&-2\\1&0&-1\end{bmatrix}=\begin{bmatrix}1\\2\\1\end{bmatrix}\begin{bmatrix}1&0&-1\end{bmatrix}$$

$$G_y=\begin{bmatrix}1&2&1\\0&0&0\\-1&-2&-1\end{bmatrix}=\begin{bmatrix}1\\0\\1\end{bmatrix}\begin{bmatrix}1&2&1\end{bmatrix}$$

$$\text{对于倾斜边缘及定位精度效果不好}$$

- Canny边缘检测：5个步骤，可以得到精确的单边缘图像

1. 高斯模糊：

   通过模糊算子将噪声图像进行平滑处理，降低噪点导致的伪边缘。这一步中高斯核的大小选择很重要，对于边缘检测一般不能过大，否则边缘的模糊会导致后续算法效果下降。（实现很简单，算法略）

2. 梯度幅度与方向：

   计算每个点的梯度值和梯度方向。

   实现细节：用两个表记录幅值和方向，方向进行离散化处理，可取0～7作为一个点的8个可能方向。

   （这一步可以使用Sobel等其他算子代替常规的梯度计算方法，各有不同特点）

3. 非最大抑制：

进行初步的边缘细化，遍历每个梯度幅度像素点，判断该点沿着正负梯度方向的3个点中该点是否是最大的，如果是则保存为255，否则抑制为0。（该步骤可以与4一同一次遍历完成）

4. 滞后阈值（双阀值）

   对3中的抑制后边缘图按照梯度进行细分，如果幅度高于"高阀值"，则标记为强边缘，如果介于"高阀值"和"低阀值"间，则标记为弱边缘，其他的边缘删除。

   3、4实现细节：（将梯度幅度图一次拆为不交的强边缘图和弱边缘图，二者均初始化为0）

   一次遍历梯度幅度图，对每个点

   - 先按其梯度方向判断是否是极大值，若不是，不进行修改
     - 若是，根据其幅度和高低阀值的大小关系在强边缘图和弱边缘图中存为255
     - 若不是，不进行修改

5. 滞后边界跟踪

   强边缘认为是真的边缘，弱边缘则可能是有噪声引起的，因此需要排除噪声弱边缘。一般来说噪声弱边缘是不与强边缘相交的，因此只需要对每个连通的弱边缘判断是否有一点与强边缘相邻即可，一般使用非递归dfs/bfs寻找判断。

   实现细节：

   - 遍历弱边缘图的每一个点，并使用vis标记
     - 使用dfs/bfs一次性找到该点出发连通的所有弱边缘点，每次加入时判断是否与一个强连通点相邻，并用connected记录结果
     - 根据connected的结果将本次找到的所有连通弱边缘点加入加入真边缘点图
   - 将强边缘点加入到真边缘点图中，得到Canny边界检测结果

4.Hough变换

用于进行直线检测，将边缘图像的每个点转化为另一个空间的一条线，如

若x,y为边缘图的直线上一点，那么有$y=ax+b$

变换为：$b=-ax+y$

相当于每个(x,y)点都会在a,b空间中画出一条直线，而这些直线的交点则是(x,y)空间中参数a,b的值

但是这种变换会造成对于竖直线的a，b空间很大，不太方便处理，因此普遍使用极坐标变换。

极坐标变换方法：对点(x,y),变换到$r-\theta$空间的一条正弦曲线

$$r=xcos\theta+ysin\theta$$

$x-y$空间下的一条直线变换后则是一个正弦曲线族，它们经过同一个点(r_0,θ_0)，分别表示原点到该直线的距离和该直线的倾斜角

实现细节（采用二维数组计数算法）：对极坐标初始化一个二维数组，一个方向表示垂直距离，大小约为图片最大对角线长度，另一个方向表示倾斜角，根据需要细分-180-180，然后对边缘二值图像里每个像素点，对每个角度计算r，然后该格子的计数器+1，最后提取二维数组里计数器大于一定值的参数则是原图里的特征直线。

5.Ransac方法

随机抽样方法，一般用于图像拼接等拟合估计模型中，主要思想就是从样本中任意选取样本子集（较小），然后据此计算拟合结果，然后判断其他样本点与结果的相合性（可用一些评价值），据此记录该拟合结果的内点（与结果偏差较小的点），重复此过程选择内点数最大或与总样本数之比高于一定阈值（称为“内点分数”）的作为最佳拟合。

该方法针对不同拟合模型有不同的实现细节，需要分别讨论，同时也对样本数量有一定下限要求：

$p<1$表示拟合结果的置信度，$W<1$表示可用的拟合所需内点分数，n为选取的样本子集点数，则所需最小的样本数量：

$$k=\frac{log(1-p)}{log(1-W^n)}$$

这种方法对于高噪声的数据处理效率和效果很差。

6.局部不变特征

即通过算法找到两张图片某一特征的相似性，比如旋转，缩放等变换后的相似度，可用于进行图片特征匹配。

具体分为5个步骤：

1. 寻找两个图片的关键点（并非随机，有某些特殊特点的点）
2. 在关键点周围确定一个范围——邻域
3. 规范化两个关键点的邻域数据，使二者具有可比性
4. 计算两个邻域的descriptors（描述符），即抽象后的描述局部特征的一组参数
5. 通过descriptors的相似性进行特征匹配

6.1确定关键点：角点检测（Harris算子）

用于进行尺度相同两张图片的关键点检测。

角点特征：以该点为中心选取一个合适的窗口，进行小量平移后平均变化的总合最大:

对以(x_0,y_0)为中心的窗口，令$E(u,v)=\sum_{x,y}w(x,y)[I(x+u,y+v)-I(x,y)]^2$取最大值

其中x,y为窗口内的所有点，u,v为选择的微小平移量，w为选取的权重核$(一般取高斯核)$

将该极值问题推导化简：

$$E(u,v)\approx\sum_{x,y}w(x,y)[\frac{\partial I}{\partial x}(x,y)u+\frac{\partial I}{\partial y}(x,y)v]^2$$

$$=[u\quad v]H\begin{bmatrix}u\\v\end{bmatrix}$$

$$其中H=\sum_{x,y}w(x,y)\begin{bmatrix}I_x^2&I_xI_y\\I_xI_y&I_y^2\end{bmatrix}$$

对H进行特征值分析：

设H特征值为λ_1,λ_1

若λ_1和λ_2均趋于0，则中心点在光滑区域

若二者有一个较大，另一个接近0，那么中心点在边缘区域（线边缘）

若二者均比较大，则中心点位于角点区域

为了避免计算特征值，对二阶方阵，特征值之积等于矩阵行列式的值，特征值之和等于矩阵迹线，因此使用下式进行计算：

$$Harris角点准则：C = det(H) - k * tr^2(H) = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2, 其中k为0.4 - 0.6一个常数$$

实现细节：

1. 平滑滤波处理
2. 计算每个点的两个方向导数（水平和竖直）
3. 对该点的窗口所有点求和，计算每个点的Harris矩阵（高斯卷积核*导数矩阵），
4. 计算Harris角点响应值（随后可进行非极大抑制，即在两个方向上判断该点是否是极值点，抑制非极值点）
5. 选择响应值高于一定阈值的点作为特征点

6.2 SIFT尺度不变特征变换

用于尺度变换了的两张图片关键点检测。

核心原理：在Harris算子基础上，考虑两张图片进行缩放旋转后特征点的提取，需要对选取的窗口大小进行相应调整。

主要步骤：

1.尺度空间极值检测

通过高斯差分函数搜索所有尺度的图像位置，识别出对于尺度和方向不变的潜在兴趣点。

高斯金字塔的建立

高斯金字塔时一组由大小递减的图像形成的塔状结构，每组内的图片大小相同，是由最底层图片分别对5*5大小不同方差的高斯核卷积生成，一般一组内有3个同大小方差成比例增大的图片，相邻两组的图片大小2倍递减，使用“降采样”，即隔行隔列取，图像线度减半，共约有log(原尺寸)组数的图片：

例：对一张$128 * 128$的图片构建高斯金字塔：

令最上层为$32 * 32$(可变)

序号	图片大小	高斯核的标准差（σ可变）	构建方式
3.3	32*32	2^(8/3)σ	1/4原图*G(2^(8/3)σ)
3.2	32*32	2^(7/3)σ	1/4原图*G(2^(7/3)σ)
3.1	32*32	4σ	1/4原图*G(4σ)
2.3	64*64	2^(5/3)σ	1/2原图*G(2^(5/3)σ)
2.2	64*64	2^(4/3)σ	1/2原图*G(2^(4/3)σ)
2.1	64*64	2σ	1/2原图*G(2σ)
1.3	128*128	2^(2/3)σ	原图*G(2^(2/3)σ)
1.2	128*128	2^(1/3)σ	原图*G(2^(1/3)σ)

序号	图片大小	高斯核的标准差（σ可变）	构建方式
1.1	128*128	σ	原图*G(σ)

差分取极大值

本质上就是同一张图片进行方差不同的高斯卷积后做差：

$$\text{高斯差}DOG(\sigma) = (G(k\sigma) - G(\sigma)) * I$$

然后将金字塔每组3个往上扩展两个（方差按比例增），变成5个，然后组内相邻两张图片做差，变成4张差分图片，对这四张图片的每个像素点判断3*3 *3内是否是极值点（即与周围26个点比较），选取全部的极值点作为候选点。

2.关键点定位

在每个候选位置上，利用拟合模型确定关键点位置和尺度。

首先通过差值思想滤掉非极值的候选点，对图像取二阶泰勒展开后求导，解得极值点位置：

$$\vec{x} = -\frac{\partial^2 D^{-1}}{\partial x^2}\frac{\partial D}{\partial x}$$

之后根据Hessian矩阵除去低对比度和边缘效应极值点：

$$\text{计算}(x,y)\text{处的}Hessian\text{矩阵：}H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

一个合理的极值点H的两个特征值之比应该接近1，即两个特征值相近。

说明极值点处两方向变化类似，不是边缘造成的，为了描述这个比例特点，让极值点满足：

$$\frac{tr^2(H)}{det(H)} < \frac{(r+1)^2}{r}, \quad r\text{取}10\text{左右}$$

3.方向匹配

给予局部图像的梯度方向，为每个关键点分配一个或多个方向，从而确定相对关键点的尺度和方向变换，进而获得尺度不变特征。

对2中最后选取的每个特征点，计算它的主方向，即计算4.5*σ半径内全部像素的幅值和幅角，将幅角离散化处理（8个或者10个），将全部邻域像素相同幅角的幅值相加（即直方图），幅值最大的方向即"该特征点的主方向"。（还有辅方向等附加处理，比较细节）

以上处理全部是在高斯金字塔中的图像中，并非是对应的原图！

4.关键点描述符

在每个关键点邻域内，以选定的尺度计算局部图像梯度，并扩展为可以允许较大局部形状变形和光照变化的表示。

- 首先将每个特征点的邻域（取16*16）旋转主方向的角度，即使该点的方向归0，保证特征的旋转不变性。
- 然后对旋转后的图像再次求去特征点邻域像素的梯度方向（8向离散化）和梯度幅值，然后对以特征点为中心把邻域分为16个区域，每个区域为16个像素的正方形，16个正方形组成16*16的邻域。

- 对每个区域统计8个方向的幅度值之和，最后每个邻域拥有代表8个方向幅值和的8维向量
- 16个区域的全部128个向量进行归一化，（排除光照等因素）作为该特征点的128维描述符

Sift描述符的128维向量是区域图像的抽象，满足缩放、旋转、光照强度不变性。

7.Seam Carving图像变换系列算法

7.1缩放

思想就是删减平滑区域，保留边缘，让改变的像素不会引起很强的视觉变化。

每次删去图像中一条从顶部到下的像素线，直到图像宽度删减为想要的宽度，如果想要压缩长度，可将图像进行转置，进而压缩长度。

算法步骤：

1.进行n-n'次循环（每次删去一条竖着的曲线）

2.计算当前图像的能量图：

每个像素有一个能量值：

$$E(I) = |\frac{\partial I}{\partial x}| + |\frac{\partial I}{\partial y}|$$

偏导数可用$Sobel$偏导计算，注意直接绝对值相加，而非平方和开根

3寻找最小能量线（也就是从上到下通过8个方向连着的一条线能量之和最小的线），使用动态规划算法，转移方程很简单：

$m[i][j]$表示从顶部一点到i，j线的最小能量，顶层设置为自身能量，边界外设置为0

转移方程$m[i][j] = E[i][j] + min(m[i-1][j-1], m[i-1][j], m[i-1][j+1])$

4.回溯法依次删除删除点，并将该点左右两侧的像素更新为与裂缝线的像素值的平均，使裂缝更加平滑。

5.返回1

算法对于如人脸等图像的效果不好，对于风景类对畸变不敏感的图片更适用，因此在一些情形中需要增设约束，让删除的能量线尽量不涉及关键部位，避免图像大面积失真。

另外有一种"forward energy"的处理手段可以用来避免删除导致图像能量上升，锯齿化明显的现象。

7.2扩展

选择前n-n'个最小能量线，将其进行复制，从而扩展图像。（不能用迭代进行复制，否则很有可能将一条线重复很多次）

7.3对象移除

在计算时只删除通过待删对象边缘的那些能量线。

（具体算法没有细究，想要实现上述功能还需要记录不少信息）

8.图像分割-聚类

图像分割的两种目的：

- 将图像分割成连贯的对象
- 将图像分割成“super pixel",即多个像素组成的块状像素，可以更快速的对图像进行处理

这实际上就是聚类在CV中的体现

8.1 凝聚聚类

主要思路：

1. 将每个点初始化为自己的群集
2. 找到最相似的一对群集
3. 将这对相似群集合并生成一个父群集
4. 重复2、3直到只有一个群集

群集间距离的不同度量方法：

1.单链接（single link)

用两个群集中两点的最近距离表示群集距离

$$d(C_i, C_j) = min_{x \epsilon C_i, x' \epsilon C_j} d(x, x')$$

使用这种距离方法的聚类构造聚类树实际上就是一个图的最小生成树构造过程，与Kruskal算法完全一致，因此可以使用Prim算法等各种最小生成树算法。

特点：这种方式倾向于生成长而细的群集（在特征点的空间中看）（类似扁平的椭圆形）

2.完整链接(complete link)

与单链接刚好相反，使用最远距离表示

$$d(C_i, C_j) = max_{x \epsilon C_i, x' \epsilon C_j} d(x, x')$$

该方式则倾向于生成均匀丰满的群集（类似于正圆）

3.平均链接（average link)

使用两群集内所有点对的平均距离表示群集距离

$$d(C_i, C_j) = \frac{\sum_{x \epsilon C_i, x' \epsilon C_j} d(x, x')}{|C_i||C_j|}$$

8.2 k均值聚类

与凝聚聚类的区别在于用一个聚类中心代表一个群集，一个点到一个群集的距离就是点到该群集中心的距离。

k为给定的类别数，聚类要确定的就是这k个聚类中心。

k均值聚类满足一下两个约束：

- 每个聚类的所有点到它所属的聚类中心距离是到其他k-1中心最小的
- 每个类别群集中聚类中心是该群集全部点的均值（质心）

k均值聚类算法流程：

- 初始化k个聚类中心（一般随机选取k个点）
    - 将数据集中每个点分配到最近的那个中心（可使用各种距离度量方法）
    - 更新每个群集的中心（用质心点代替）
- 达到最大迭代次数或中心不再改变时结束

注意，该方法的结果并不唯一，即满足两个约束的聚类结果可能有多种，因此需要多次尝试选取不同初值，然后选择合理的迭代聚类结果

8.3Mean-shift均值漂移

与k均值分类不同，均值漂移的迭代是多次迭代的迭代，每个单独的的迭代会确定一个群集，不同的迭代确定的中心距离较小时将两个群集合并，否则作为一个新的群集。而单个迭代寻找聚类中心的过程是"漂移"的过程，漂移中经历的所有点都属于这个群集，即中心按照点的密度由低到高进行挪动直到稳定在附近的密度极值点，而聚类群集就是一个沿密度上升方向的一类数据点。

Mean-shift主要步骤：

- 重复一下步骤，直到所有点被标记
    - 重复以下步骤，直到位移量m小于阈值
        - 在未标记的数据点中随机选择一个点作为中心center
        - 找出center周围以r（一个定值半径）半径圆内的所有点，将他们加入群集（标记），并计算平均位移m（公式见下）
        - 更新center=center+m
    - 最终收敛到的群集center与已有群集中心距离小于阈值时，将这两个群集合并，否则这个新的center群集作为新的聚类

$$\vec{m}=\frac{\sum_{i=1}^{n}\vec{x}_i\cdot g(|\frac{\vec{x}-\vec{x}_i}{h}|^2)}{\sum_{i=1}^{n}g(|\frac{\vec{x}-\vec{x}_i}{h}|^2)}-\vec{x}$$

其中g为一个权重核函数，x为当前$center$位置，x_i为圆内点位置

9.目标识别

9.1k近邻分类

思路：选择与检测点最近的k个样本点，并将其分配给附近最多的那个类别。

k的选择：

- k值过低导致对噪声点很敏感
- k值过高导致结果边界模糊，不准确性增高
- 因此一般使用交叉验证尝试不同k值最后确定最优k值（这部分在Python大作业已完成，不再赘述）

9.2奇异值分解SVD

对非方阵矩阵进行的对角化变换：

$$A = UDV^T$$

A为$m * n$矩阵

U为m阶方阵，且为正交矩阵，其列向量为\\左奇异向量//，是AA^T的特征向量

D为$m * n$矩阵，且为对角矩阵（认为对角线是左上到右下的45度线）,对角线的元素为\\奇异值//

V为n阶方阵，且为正交矩阵，其列向量为\\右奇异向量//，是A^TA的特征向量

SVD可以用于计算任意矩阵的“伪逆”。

9.3主成分分析

见深度学习部分，此处略。

二.深度学习

前2节课主要介绍了计算机视觉的发展历程和近些年产生的CNN卷积神经网络的发展。值得思考的是这一部分中解决计算机视觉问题的研究思路和想法。人们从数字中提取数值外的整体特征的方法多种多样，针对不同问题的解决思路也各不相同。

目前主流的问题是图像识别和分类。这类问题一般需要两类求解器，其一是训练器，用于根据训练集生成一个“模型”，也就是机器自己学习已有“知识”进行提炼的过程。其二是预测器，用来讲已有模型用于未知的数据进行分类预测。另外，为了验证可靠性，人们一般也会根据问题设置验证器，对训练集中未使用的一部分——“验证集”进行模型的效果检验。

算法实现居多，具体见作业部分。

1.KNN图像分类

这一部分在课程中主要用于熟悉numpy数组操作，核心算法简单，但是想要编写快速（python中尽量不出现循环）的向量话操作是一件很有挑战性的工作，需要更多的练习和使用。

Knn算法流程：

- 改变图片格式，考虑到后面的图片距离计算，一般直接将3通道变成一维数组，整个数据集就是一个二维数组，每一行是一张图片
- 训练train：在knn类中记录训练集图片及其label
- 生成dist数组：给出每一张测试集图片和训练集图片的距离，为一个二维数组，距离定义有L1和L2两种
- 预测predict：每张测试集图片前k个最相似的作为它的预测标签，实际上就是dist数组中每一行中前k个最多的label

一般来说，knn的k值需要通过交叉验证进行测试：

- 将训练集分成kfold折（作业中使用5折）
- 去出其中任意一折，然后将剩余的作为训练集，这一折作为验证集
- 给出每种取法的不同k值的正确率，即一个二维数组，一维表示k的值，另一维表示选取的验证集

作业总结：

1. assignment1中的knn实现主要是用于熟悉numpy库数组函数的使用，但是对于具体数据接口和可视化等部分直接给出，这有利于理解knn算法的核心部分，但是对于api等操作并不能很好地理解
2. 作业中每个函数也分布编写，一方面理解了科研中算法实现的模块化方法，同时编写思路也体现了knn从模型到预测，以及参数确定的验证步骤

2.SVM支持向量机

超平面：寻找一个高维空间的平面（即仅含一个约束），将空间内所有点分在平面两侧（一侧表示一种类别），并且离超平面最近的两侧的两个点到超平面的距离相等且达到最大（本质上就是能够将空间点最显著地分成两部分的界面），数学表示与最优化问题：

设x是空间中的一个向量，平面可表示为：$\omega^T x+b=0$

用$y=\pm1$表示类别，则有：$\begin{cases}\frac{\omega^T x+b}{|\omega|}\geq d, & y=1\\ \frac{\omega^T x+b}{|\omega|}\leq -d, & y=-1\end{cases}$

令$|\omega|d=1,$进行归一化，d达到最大值转化为如下最优化问题：

$$\forall\text{样本向量}x_i\text{和标签}y_i\text{(只有正负1)，}y_i(\omega^T x_i+b)\geq 1,\text{求}min\frac{1}{2}|\omega|^2$$

SVM优化求解步骤：

1.构造拉格朗日函数

$$L(\omega,b,\lambda)=\frac{1}{2}|\omega|^2+\sum_{i=1}^{n}\lambda_i[1-y_i(\omega^T x_i+b)],\text{其中}\lambda_i\geq 0$$

2.强对偶性转化并求偏导

$$\frac{\partial L}{\partial \omega}=0\Rightarrow\omega=\sum_{i=1}^{n}\lambda_i x_i y_i$$

$$\frac{\partial L}{\partial b}=0\Rightarrow\sum_{i=1}^{n}\lambda_i y_i=0$$

将结果带入L，化简为

$$L=\sum_{j=1}^{n}\lambda_i-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\lambda_i\lambda_j y_i y_j(x_i\cdot x_j)$$

3.通过SMO算法求解λ

每次固定两个$\lambda,$迭代求导至收敛，最终得到最优解λ^*

4.带入求b和ω：

$$\omega=\sum_{i=1}^{m}\lambda_i y_i x_i$$

b可以用约束并去平均（使用每个量）求取：$b=\frac{1}{|S|}\sum_{s\in S}(y_s-\omega x_s)$

最大分割超平面：$\omega^T x+b=0$

关于算法的优化：一方面可能需要平面不完全划分特征点，允许一些噪声或杂点越界，因此可以在约束条件中增加松弛变量，并将松弛变量加入优化问题一同求解；另一方面SMO算法的时间和空间复杂度都很大，因此需要采用核函数的方法避免计算每两个向量的内积，最终采用核函数的时空复杂度均是O(N^2)的。

常见SVM优化核函数：

线性核函数 $k(x_i, x_j) = x_i^T x_j$

多项式核函数 $k(x_i, x_j) = (x_i^T x_j)^d$

高斯核函数 $k(x_i, x_j) = e^{-\frac{|x_i - x_j|^2}{2\sigma^2}}$

图像处理中的SVM

评价思路：

使用一个权重矩阵W对一张图片的每个像素求加权，得到这张图片的每个类别的得分，分值越高，越可能属于这个类：

$S = Wx$ 如图片$x(3072 * 1)$,一共10个类别，那么W就是$10 * 3072$的权重矩阵，S是$10 * 1$的得分矩阵

损失函数：用于评估训练集中W的误差值：

每个错误类别误差：

第一种：$L_i = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + 1)$

第二种$(softmax$的交叉误差函数$)$：$L_i = -log(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}})$,

其中s_{y_i}是正确分类的得分，实际上就是全部得分取e指数后再求平均，最后再取负对数保证结果为正并且越高训练约准确

总误差：$L = \frac{1}{N} \sum_{i=1}^{N} L_i + reg \cdot \sum_k W_k^2$，前一项为求均值，后一项为正则项，防止过拟合

算法流程：

- 1.随机初始化一个值较小的W
- 迭代计算直到loss足够小或到达循环次数上限
    - 遍历每张图片
    - 计算当前Wx的得分结果并计算loss
    - 遍历每个类别：根据loss_i更新dW并叠加计算当前W对应这张图的总loss

1.折页max的$\nabla_W L$计算方法：

每次遍历图片循环中dW的更新量：

$Loss_i > 0$时，将这张图的像素行加到dW对应不正确分类的那一列中，将dW的正确类别的那一列减去这张图像像素；

$Loss_i <= 0$时,dW不更新

最后再在dW中更新正则项的导数，也就是$2 * W * reg$

2. $softmax$的$\nabla_W L$计算方法：

首先进行求导解析解的推导：(以下是我的推导方法，由于对向量化操作不熟悉，因此用单个变量求导)

$\frac{\partial L_k}{\partial w_{ij}} = -\frac{\sum_j e^{s_j}}{e^{s_{y_k}}} \cdot \frac{\partial(\frac{e^{s_{y_k}}}{\sum_j e^{s_j}})}{\partial w_{ij}}$，　w_{ij}即第i个像素的第j个分类的权重值，k表示第k张图片的$loss$

若$j = y_k$(即j是i张图片的正确分类)：

$$\frac{\partial L_k}{\partial w_{ij}} = -\frac{\sum_j e^{s_j}}{e^{s_{y_k}}} \cdot \frac{\partial(1 - \frac{(\sum_j e^{s_j}) - e^{s_{y_k}}}{\sum_j e^{s_j}})}{\partial w_{ij}}$$

考虑$s = xW$可以知道分子：$(\sum_j e^{s_j}) - e^{s_{y_k}}$不含$w_{ij}$,故只需对分母求导：

$$\frac{\partial L_k}{\partial w_{ij}} = -\frac{\sum_j e^{s_j}}{e^{s_{y_k}}} \cdot \frac{(\sum_j e^{s_j}) - e^{s_{y_k}}}{(\sum_j e^{s_j})^2} \cdot \frac{\partial e_{s_{y_k}}}{\partial w_{ij}} = -\frac{(\sum_j e^{s_j}) - e^{s_{y_k}}}{e^{s_{y_k}} \cdot \sum_j e^{s_j}} \cdot e_{s_{y_k}} \cdot x_{ki}$$

$$= (\frac{e_{s_{y_k}}}{\sum_j e^{s_j}} - 1) \cdot x_{ki} = (\frac{e_{s_j}}{\sum_j e^{s_j}} - 1) \cdot x_{ki}$$

若$j \neq y_k$(即j是i张图片的正确分类)：

$$\frac{\partial L_k}{\partial w_{ij}} = -\frac{\sum_j e^{s_j}}{e^{s_{y_k}}} \cdot \frac{\partial(\frac{e^{s_{y_k}}}{\sum_j e^{s_j}})}{\partial w_{ij}}$$

考虑$s = xW$可以知道分子：$e^{s_{y_k}}$不含w_{ij},故只需对分母求导：

$$\frac{\partial L_k}{\partial w_{ij}} = -\frac{\sum_j e^{s_j}}{e^{s_{y_k}}} \cdot \frac{e^{s_{y_k}}}{(\sum_j e^{s_j})^2} \cdot \frac{\partial e_{s_{y_k}}}{\partial w_{ij}} = \frac{1}{\sum_j e^{s_j}} \cdot e_{s_j} \cdot x_{ki} = (\frac{e_{s_j}}{\sum_j e^{s_j}}) \cdot x_{ki}$$

以上推到可以直接转换为程序，遍历每一张图片和每一个分类，找出s行向量，

然后每次可以直接更新dW的一列（因为一行s的生成只与w的一列相关）

根据这一操作思路可以类似转化为向量化操作

最后再在dW中更新正则项的导数，也就是$2 * W * reg$

注意，使用max误差函数以上循环全部可以进行向量化操作，注意使用广播和矩阵乘法转化dW的更新

3.反向传播和神经网络

反向传播用于高效计算结果对于每个网络节点的导数，在正向传播时计算每个节点的值，并存储本地导数（也就是这个节点对上一个节点的导数，这个值是固定的，需要人为提前计算并进行存储），反向传播时从结果按逆拓扑序用前一节点导数值乘以本地节点导数值，由链式法则可知就是这一节点的导数值。

反向传播的原理很简单，不过在对矩阵进行求导计算时需要细心推导，下面给出常用的结论：

对向量x和线性变换W，定义：$f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

$$\Rightarrow \nabla_W f = 2W \cdot x \cdot x^T$$

$$\Rightarrow \nabla_x f = 2W^T \cdot q$$

传统神经网络：Fully Connected Net

Fully Connected Net一半由多组Affine 和Relu以及中间的优化（包括批量归一化等操作）和最后的一个Affine及softmax组成。

$$Fully-Connected-Net：(Affine+[batch-norm]+Relu+[..])\times L+Affine+softmax$$

常用的神经元：

- sigmoid：$\sigma(x)=\frac{1}{1+e^{-x}}$，$\sigma'(x)=\sigma(x)(1-\sigma(x))$
- softmax-loss：$Loss=\frac{1}{N}\sum_{i=1}^{N}-log(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}})+reg\cdot\sum_k W_k^2$
- Relu：$H=max(0,X)$
- Affine：$H=WX+b$
- Batch-Normalization (批量归一化BN)：消除线性层或者卷积层导致的特征尺度累计效应，同时也具有一定的正则化效果

  对输入的N张图片$(每张有D维)$，考虑第k列（即N张图片的同一维位置的值）的归一化：

  $$\hat{x}_k=\frac{x_k-E(x_k)}{\sqrt{D(x_k)}}$$

  BN具体算法：

  选取一个$mini-batch$用于计算一个特征维度的均值和方差，取为$B=\{x_1,x_2\ldots\}$

  $$\mu_j\leftarrow\frac{1}{m}\sum_{i=1}^{m}x_{ij}\quad;\quad\sigma_j^2\leftarrow\frac{1}{m}\sum_{i=1}^{m}(x_{ij}-\mu_j)^2$$

  对于处理全部的N个训练输入，对输入按列进行归一化：$\hat{x}_{ij}=\frac{x_{ij}-\mu_j}{\sqrt{\sigma_j^2+\epsilon}}$

  最后再进行变换，需要进行变换参数γ和β的学习$(并非矩阵乘法，二个参数均为长度D的一维向量，即X的每一列拥有相同的值)$：

  $$y_{ij}=\gamma_j\hat{x}_{ij}+\beta_j\equiv BN_{\gamma,\beta}(x_{ij})$$

  另外，实现中还有一个细节：

  训练时每次使用$mini-batch$的均值和方差，但是要在迭代中动态更新一个$running-mean$和$running-var$

  在迭代结束时这两个值可以用来当作全体训练集的均值和方差，用于测试时使用，更新方程：$(momentum一般取0.9)$

  $$RunningMean=momentum*RunningMean+(1-momentum)*SampleMean$$

  $$RunningVar=momentum*RunningVar+(1-momentum)*SampleVar$$

  BN反向传播：

记上一层返回的$\frac{\partial L}{\partial y} \equiv dout$, 大小为$N \times D$

根据计算图计算反向传播得出$\frac{\partial L}{\partial \gamma}, \frac{\partial L}{\partial \beta}, \frac{\partial L}{\partial x}$：

$$\frac{\partial L}{\partial \gamma} = \sum_{i=1}^{n}(\frac{\partial L}{\partial y} \cdot \hat{x})_i, \text{即点乘后按列相加}$$

$$\frac{\partial L}{\partial \beta} = \sum_{i=1}^{n}(\frac{\partial L}{\partial y})_i, \text{即} dout \text{按列相加}$$

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial x_1} + \frac{\partial L}{\partial x_2}$$

其中：

$$\frac{\partial L}{\partial x_1} = \frac{1}{N}\begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix}^{N \times D} \frac{\partial L}{\partial \mu}$$

$$\frac{\partial L}{\partial x_2} = \frac{\partial L}{\partial y}\frac{\gamma}{\sqrt{\sigma^2+\epsilon}} + \frac{2}{N}\begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix}^{N \times D} \frac{\partial L}{\partial \sigma^2}(x-\mu)$$

$$\frac{\partial L}{\partial \mu} = -\frac{\gamma}{\sqrt{\sigma^2+\epsilon}}\sum_{i=1}^{N}\frac{\partial L}{\partial y} - \frac{2}{N}\frac{\partial L}{\partial \sigma^2}\sum_{i=1}^{N}(x-\mu)$$

$$\frac{\partial L}{\partial \sigma^2} = -\frac{1}{2}\sum_{i=1}^{N}\frac{\partial L}{\partial y}\frac{\gamma(x-\mu)}{(\sigma^2+\epsilon)^{\frac{3}{2}}}$$

- Layer Normalization：另一种归一化，对输入数据点的特征为轴计算均值和方差进行归一化，BN是纵向的归一化，而LN是横向的归一化，总体上效果不如BN但是更省时，算法上只需要将BN的数据转置后计算。（另外不再设置running-mean和var了，训练和测试的归一化方式完全相同）
- Dropout：防止过拟合采取的另一种手段，将一层中的某一些神经元置为零（实际上是每个神经元以概率p伯努利分布失活，即直接让输出的特征中部分变为0）。在测试时前向计算时不进行Dropout，因为本质上不需要在这时防止过拟合。

网络中正向传播用于顺着网络计算最终的得分矩阵，反向传播用于倒推每个Affine节点系数矩阵的偏导数，然后用梯度下降一次次迭代训练这些系数矩阵，降低loss，提高神经网络分类能力。

四种梯度下降方法算法

1.SGD随机梯度下降：$W \leftarrow W - \eta\frac{\partial L}{\partial W}$,对于呈延伸状函数搜索效率很低

2.SGD-Momentum: 通过动量（梯度的积累）有效地处理局部极值和鞍点，同时也有效地降低了mini-batch造成的噪声随机性，路径更加稳定

$$v \leftarrow \alpha v - \eta\frac{\partial L}{\partial W}$$
$$W \leftarrow W + v$$

3.RMSProp(Ada优化)：具有记忆性，在梯度大的方向学习率降低，梯度小的方向学习率提高

$$R \leftarrow \rho R + (1-\rho)\frac{\partial L}{\partial W} \odot \frac{\partial L}{\partial W}, \quad R\text{与}W\text{形状相同}$$
$$W \leftarrow W - \eta \frac{1}{\epsilon + \sqrt{R}} \odot \frac{\partial L}{\partial W}, \quad \sqrt{R}\text{指对每个元素开根,}\epsilon\text{是个小量，避免分母为}0$$

4.Adam

- s:历史梯度指数衰减平均，代表了动量的积累$s \leftarrow \rho_1 s + (1-\rho_1)\frac{\partial L}{\partial W}$
- r：历史梯度平方的指数衰减平均，$r \leftarrow \rho_2 r + (1-\rho)\frac{\partial L}{\partial W} \odot \frac{\partial L}{\partial W}$
- 修正偏差，防止训练初期s和r太小（ρ_1和ρ_2一般都比较接近于1): $\hat{s} \leftarrow \frac{s}{1-\rho_1^t}; \hat{r} \leftarrow \frac{r}{1-\rho_2^t}$，t表示当前迭代次数

算法：

$$s \leftarrow \rho_1 s + (1-\rho_1)\frac{\partial L}{\partial W}$$
$$r \leftarrow \rho_2 r + (1-\rho_2)\frac{\partial L}{\partial W} \odot \frac{\partial L}{\partial W}$$
$$W \leftarrow W - \eta \frac{s/(1-\rho_1^t)}{\epsilon + \sqrt{r/(1-\rho_2^t)}}$$

4.卷积神经网络CNN

重要声明：一般神经网络里所谓“卷积”就是对应位置加权求和，算法中不对核进行翻转！！！一下公式里的“$\circ$”卷积全部使用点乘实现，可以避免很多繁琐的操作。

卷积层-Conv：使用卷积加偏差替代神经网络中节点的输出算法，对于彩色图像（一般称“深度”为3，rgb），卷积核也应是三维的，每个卷积层中有多个卷积核，浅层的卷积核训练后能够展现图片的简单细节，而更深层的卷积核则体现更复杂的细节。另外，默认图像边界填充0像素，保证结果的形状不会变小。

$$H_{i,j} = I \circ g + b$$
例如：对$32 \times 32 \times 3$的图片，某一层使用10个5×5(暗指$5 \times 5 \times 3$)卷积核
输出的结果为$32 \times 32 \times 10$,总共$(5 \times 5 \times 3 + 1) \times 10 = 760$个参数

卷积层的求导：具体推导可以手撕，这里给出结论

设卷积层$Y = X \circ W + b$ （X为输入，Y为输出，W为核，b为偏差)有：

$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Y} \circ W$,其中$\frac{\partial L}{\partial Y}$需要补$0$，每两个元素以及外侧需要插入$stride-1$个$0$，同时最外侧额外补全$pad$个$0$

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial Y} \circ X\text{(如果使用的是卷积而不是点乘，则需要额外将结果中心对称一下)}$$
$$\frac{\partial L}{\partial b} = \sum_{i,j}(\frac{\partial L}{\partial Y})_{i,j}$$

需要注意，对dout行间插入0的部分没有库函数，需要自己写循环插入（cs231中的数值验证只对stride=1有效，因此大部分只用numpy实现的conv-backward都只能在stride=1下运行）。

针对卷积层，batchnormalization的均值和方差取为这个通道的全部像素（即最后的γ、β都是C维向量（C是卷积层的核数量）

同时，为了减少计算，对于卷积网络也有batchnorm的变形（与layernorm不同），称为**Group Norm**，可以理解为在layer Norm的基础上，输入维度为（N,C,H,W），对C进行分组，即为（N,G,C//G,H,W），对N个通道G个组进行归一化，输出维度为（N,G,C//G,H,W），再还原输出维度为（N,C,H,W）

池化层：一类特殊的卷积核，对输入的平面尺寸进行压缩（深度方向不变）即将采样，这里的卷积计算不填充0，并且使用的步长不是1，而是使进行了卷积计算的区域不发生重叠的步长（就是用卷积核均匀覆盖整张图片，无重叠部分）。

（例子——常用的Max Pooling：取2*2大小的像素块，将其用4个像素中最大的那个代替，最终图像被压缩为一半。最大值池化能够体现神经元在某一区域的激发程度，更好地描述特征。）

$$ConvNets：[(Conv+Relu)\times N+Pool]\times M+(FC+Relu)\times K+softmax$$

5.Pytorch网络构建

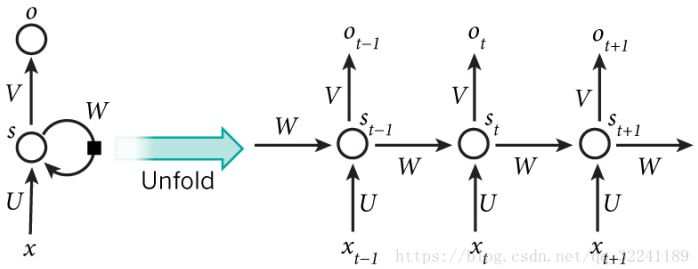
使用pytorch构建cnn的方法：

1. 定义网络类（nn.Module的子类，如"$classTwoLayerFC(nn.Module)$"）
    1. *init*函数，需要先用super（），其中设置网络的结构和初始化方法（内封初始化）
    2. 定义forward，由输入x计算scores
2. 编写Train函数，定义使用的优化算法，进行前向和后向计算
3. 倒入数据集，数据预处理，调用方法，输出结果

6.循环神经网络RNN

1.朴素RNN（Vanilla RNN）

作用：输出的结果不仅仅与当前样本有关，还与之前输入的样本有关，这种样本输入顺序称为时间序列，主要用于考虑上下文语义，前后数据关系分析特征。



$$\begin{cases}s_t=f(Ux_t+Ws_{t-1}+b)\\o_t=g(Vs_t+c)\end{cases}$$

W,U,V在循环中是同一个值

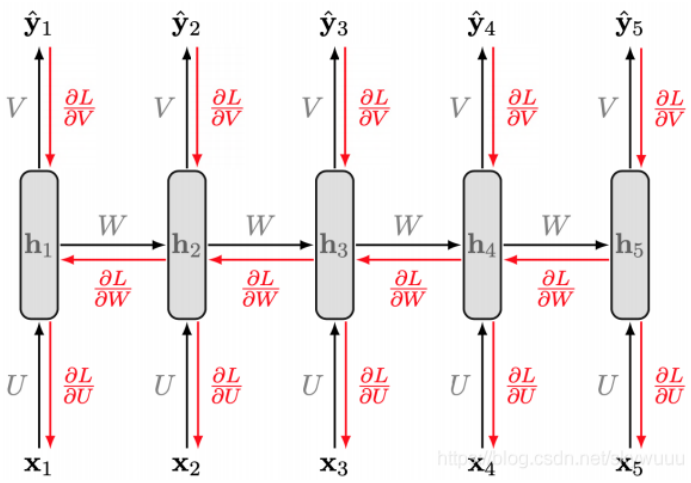
f,g是激活函数

f可为$tanh$，$relu$，$sigmoid$

g一般为$softmax$

Vanilla RNN中，f函数取tanh，g函数取softmax。

RNN反向传播BPTT（**Bach Propogation through Time**）



注意，这里导数得按时间序列分别求，最后统一更改权重，因此一次只能操作第t个输入的反向传播L_t(并非之前的一次性计算所有样本)

$$\frac{\partial L}{\partial V}=\sum_t\frac{\partial L_t}{\partial V};\qquad \frac{\partial L}{\partial c}=\sum_t\frac{\partial L_t}{\partial c};\qquad \frac{\partial L}{\partial U}=\sum_t\frac{\partial L_t}{\partial U};\qquad \frac{\partial L}{\partial W}=\sum_t\frac{\partial L_t}{\partial W};\qquad \frac{\partial L}{\partial b}=\sum_t\frac{\partial L_t}{\partial b};$$

o、V、c的求导就是正常的$softmax$求导：

$$\begin{cases}\dfrac{\partial L}{\partial c}=\sum_t\frac{\partial L_t}{\partial c}=\sum_t\hat{y}_t-y_t\\ \dfrac{\partial L}{\partial V}=\sum_t\frac{\partial L_t}{\partial c}=\sum_t\hat{(}y_t-y_t)h_t^T\end{cases}$$

$$\bigstar\frac{\partial L}{\partial h_t}=V^T(\hat{y}_t-y_t)+W^Tdiag(1-h_{t+1}^2)\frac{\partial L}{\partial h_{t+1}}$$

$$\frac{\partial L}{\partial W}=\sum_t diag(1-h_t^2)\frac{\partial L}{\partial h_t}h_{t-1}^T;$$

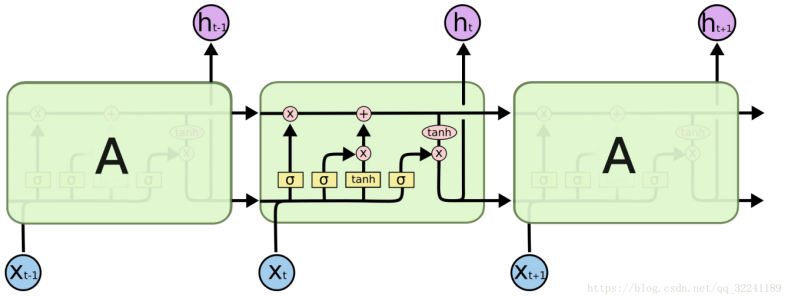
$$\frac{\partial L}{\partial b}=\sum_t diag(1-h_t^2)\frac{\partial L}{\partial h_t};$$

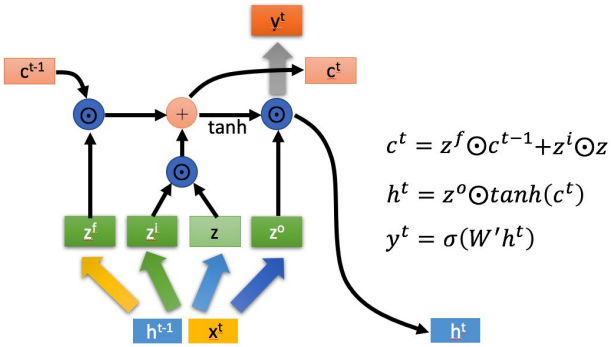
$$\frac{\partial L}{\partial U}=\sum_t diag(1-h_t^2)\frac{\partial L}{\partial h_t}x_t^T;$$

image-captioning中，h_0就是图片特征线性变换后而来，每个输入的一个词特征，输出是这个词后面的那个词，训练时可以用输出进行softmax的loss优化，测试时x_1输入“start”标识，然后输出第一个词，讲这个输出作为第二个输入，这样最终输出一整句话。

2.LSTM（长短期记忆网络）

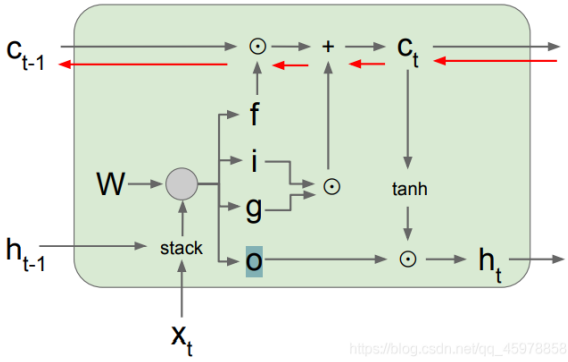
LSTM是一种特殊的RNN，主要是为了解决长序列训练过程中的梯度消失和梯度爆炸问题。简单来说，就是相比普通的RNN，LSTM能够在更长的序列中有更好的表现。





$$c^t = z^f \odot c^{t-1} + z^i \odot z$$
$$h^t = z^o \odot tanh(c^t)$$
$$y^t = \sigma(W' h^t)$$

反向传播相对简单，可通过下图加以实现：



3.transformer

目的：把无法并行运算的RNN或者LSTM等循环网络改为能够进行并行运算的网络，本质上就是设计了self-attention层进行替代。

Transformer 模型主要分为两大部分，分别是 **Encoder** 和 **Decoder**。**Encoder** 负责把输入（语言序列）隐射成**隐藏层**（下图中第 2 步用九宫格代表的部分），然后解码器再把隐藏层映射为自然语言序列。

Positional Encoding

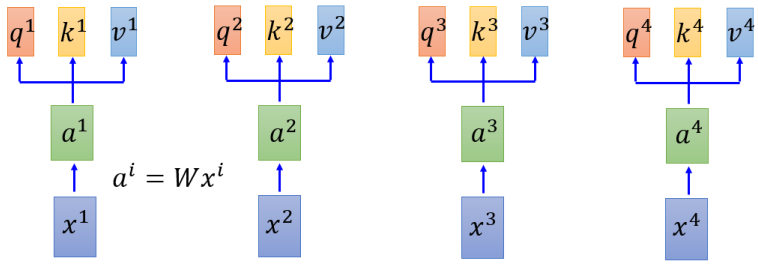
由于 Transformer 模型**没有**循环神经网络的迭代操作，所以我们必须提供每个字的位置信息给 Transformer，这样它才能识别出语言中的顺序关系。定义一个**位置嵌入**的概念，也就是 Positional Encoding，位置嵌入的维度为 `[max_sequence_length, embedding_dimension]`，max_sequence_length` 属于超参数，指的是限定每个句子最长由多少个词构成。

$$P_{i,j} = sin(i \cdot 10000^{-j/d}), \text{当}j\text{是偶数}$$

$$P_{i,j} = cos(i \cdot 10000^{-(j-1)/d}), \text{当}j\text{是奇数}$$

$$\text{其中：} i \in [0, max-sequence-length], j \in [0, embedding-dimension]$$

Self-attention Layer



q: query (to match others)

$$q^i = W^q a^i$$

k: key (to be matched)

$$k^i = W^k a^i$$

v: information to be extracted

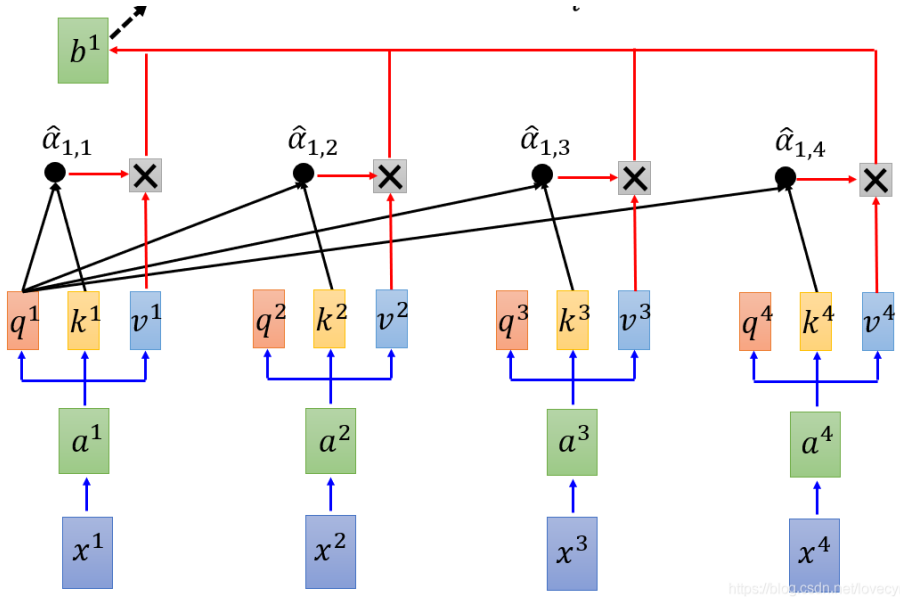
$$v^i = W^v a^i$$

每个$input\ a^i$分别乘上3个不同的Matrix得到3个不同的Vector，这个3个Vector分别命名为q、k、v，各个矩阵作用如图。

拿每个$query\ q$对每个$key\ k$去做Attention：

$$Dot-Product\ Attention: \alpha_{i,j} = \frac{q_i k_j^T}{\sqrt{d}}, \text{其中}d\text{为}q\text{、}k\text{的特征维数}$$

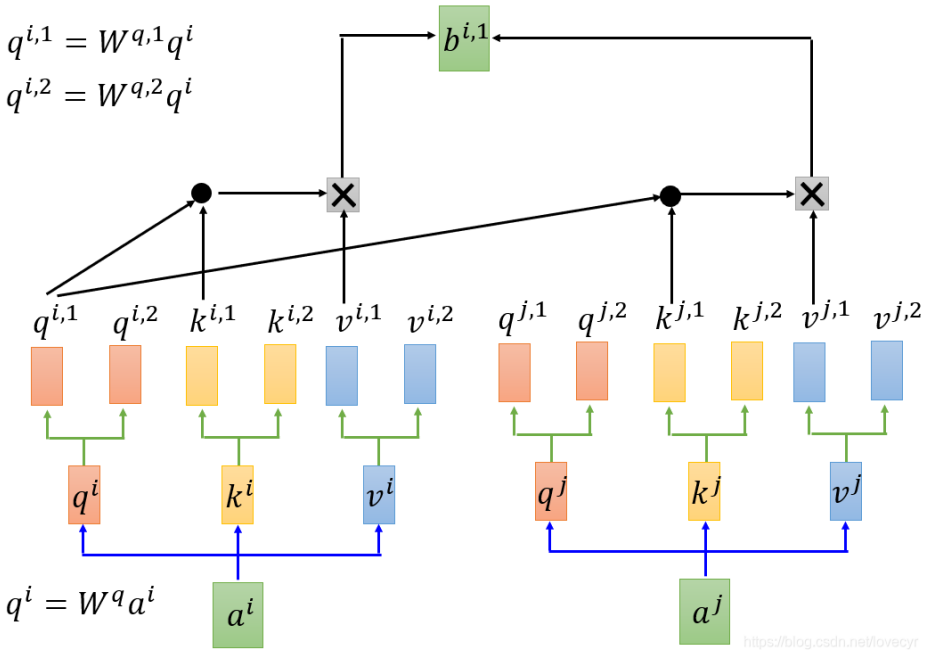
$$\text{输出层为：} y_i = \sum_j softmax(\frac{q_i k_j^T}{\sqrt{d}}) v_j$$



Multi-head Attention

可以关注更多更好的特征：

q,v,k变为两个，$q^{i,1}$只和$k^{j,1}$做$Attention$，$q^{i,2}$只和$k^{j,2}$做$Attention$，将输出的b_1和$b_2\cdots$拼接（有时可能使用dropout，只随机选择一部分作为输出），组合并经一个线性层得到最终的结果Y。



7.生成对抗网络Generative Adversarial Networks

概述：对抗是指GAN的交替训练的过程。先让生成器生成一些假图片和真图片，一起交给判别器判别，让它学习区分两者，给真的高分，给假的低分，当判别器能够熟练判断现有的数据后，再让生成器以从判别器处获得高分为目标，不断生成更好的假图片，直到能骗过判别器，重复这一过程，直到判别器对任何图片的预测概率都接近0.5，也就是无法判别图片的真假，就可以停止训练了。

目的：训练一个GAN的最终目标就是获得一个足够好的生成器，也就生成一个足够已经乱真的内容，能完成类似功能的还有玻尔兹曼机、变分自编码器等，这些都被成为生成模型。

就是**通过生成网络G（Generator）和判别网络D（Discriminator）不断博弈，进而使G学习到数据的分布**，如果用到图片生成上，则训练完成后，G可以从一段随机数中生成逼真的图像。G， D的主要功能是：

- G是一个生成式的网络，它接收一个随机的噪声z（随机数），通过这个噪声生成图像

- D是一个判别网络，判别一张图片是不是"真实的"。它的输入参数是x，x代表一张图片，输出$D(x)$代表x为真实图片的概率，如果为1，就代表100%是真实的图片，而输出为0，就代表不可能是真实的图片

bce-loss：

结合sigmoid转化形式使得计算中间结果不会爆炸

x是分数矩阵，z是0、1标签向

$$
\begin{aligned}
bce-loss &= -z*log(sigmoid(x))-(1-z)*log(1-sigmoid(x)) \\
&= z*-log(\frac{1}{1+e^{-x}})+(1-z)*-log(\frac{e^{-x}}{1+e^{-x}}) \\
&= z*log(1+e^{-x})+(1-z)*(-loge^{-x}+log(1+e^{-x}) \\
&= z*log(1+e^{-x})+(1-z)*(x+log(1+e^{-x})) \\
&= (1-z)*x+log(1+e^{-x}) \\
&= x-x*z+log(1+e^{-x})
\end{aligned}
$$

同理，当$x<0,$为防止e^{-x}爆炸，进一步转化为为：$-x*z+log(1+e^{x})$

综上：$bce-loss=max(x,0)-x*z+log(1+exp(-abs(x)))$

训练时，优化目标如下：

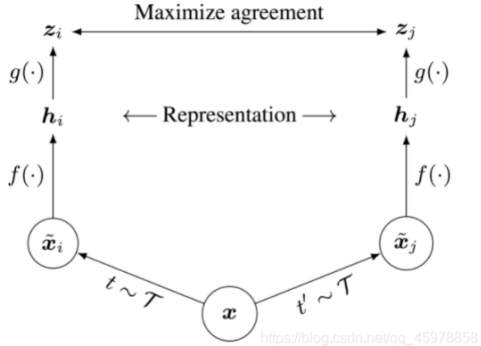
$$
l_G = bce(G\text{生成假图片结果标签结果}) : \ell_G = -\mathbb{E}_{z\sim p(z)}\left[\log D(G(z))\right]
$$

$$
l_D = bce(\text{所有图片标签结果}) : \ell_D = -\mathbb{E}_{x\sim p_{\text{data}}}\left[\log D(x)\right] - \mathbb{E}_{z\sim p(z)}\left[\log\left(1-D(G(z))\right)\right]
$$

更新时注意分辨器D反向传播时要在G生成的$fake-pic$处Z截断反向传播流$(model.detach()$方法$)$，防止l_D的优化改变生成器G的参数，只优化D的参数，使用l_G优化生成器G的参数，实现"对抗"。

8.自监督学习Self-Supervised Learning

给定图像x, SimCLR使用两种不同的数据增强方案t和t'生成正数对图像$\hat{x}i$和$\hat{x}j$。f是一个基本的编码器网络，它从扩展的数据样本中提取表示向量，分别生成h_i和h_j。最后，一个小的神经网络投影头g将表示向量映射到应用对比损耗的空间。对比损失的目标是使最终向量$z_i=g(h_i)$和$z_j=g(h_j)$之间的一致性最大化。(这里的f是一个概率图像变换器，因此可以随机讲一张图片变换为不同的图片，从而据此提取图片中不变的特征)



SimCLR: Contrastive Loss对比损失

一小批训练图像N共生成$2N$数据增强实例。对于每一个增强例子的正数对(i,j)，对比损失函数的目标是使向量z_i和z_j的一致性最大化。具体来说，损失是标准化temperature-scaled交叉熵的损失，目的是最大化zi和zj相对于批内所有其他增强示例的一致性:

$$
l\ (i,j) = -\log\frac{\exp(\ \text{sim}(z_i,z_j)\ /\ \tau)}{\sum_{k=1}^{2N}\mathbb{1}_{k\neq i}\exp(\ \text{sim}(z_i,z_k)\ /\ \tau)}
$$

$$
sim(z_i,z_j) = \frac{z_i\cdot z_j}{||z_i||\cdot||z_j||}
$$

其中$\mathbb{1}\in 0,1$是指示函数，如果$k\neq i$，则输出1，否则输出0。τ是一个$temperature$参数，它决定了指数增长的速度。

三.Diffusion Models

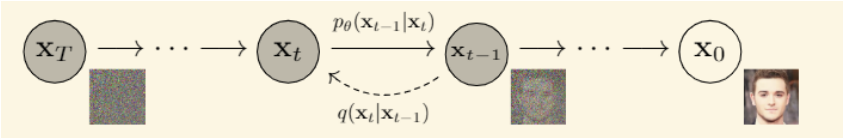
Denoising Diffusion Probabilistic Models——DDPM

Intro

目的：将给定噪声图片生成新的图片（同维度）

原理

基本结构



1. **前向过程：将输入的原图片x_0经过T轮转化为纯噪声x_T，该步用于构建训练样本**

   输入图像x_0，依次得到$x_1, x_2, \cdots, x_T$；给定一系列高斯分布的超参数$\{\beta_t \in (0,1)\}_{t=1}^T$

   每个时刻x_t只与x_{t-1}有关，构成马尔科夫链：

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$

$$q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1})$$

$$\text{其中}\beta_1 < \beta_2 < \cdots < \beta_T$$

   为了能够让每一步可微，采用随机变量重参数的方法取代上述的高斯采样：

$$z \sim \mathcal{N}(z; \mu, \sigma^2) \rightarrow z = \mu + \sigma \cdot \epsilon, \text{其中}\epsilon \sim \mathcal{N}(0,1)$$

   扩展到噪声图像的马尔可夫链中，每个x_t都可以递推的由x_0, β表示：

$$\begin{aligned}
&\text{假设随即参数 } z_1, z_2, \cdots \sim \mathcal{N}(0, I)\\
&\text{由}q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)\\
&\text{可得：} x_t = \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}z_1\\
&\quad = \sqrt{1-\beta_t}(\sqrt{1-\beta_{t-1}}x_{t-2} + \sqrt{\beta_{t-1}}z_2) + \sqrt{\beta_t}z_1\\
&\quad = \sqrt{(1-\beta_t)(1-\beta_{t-1})}x_{t-2} + (\sqrt{\beta_{t-1}(1-\beta_t)}z_2 + \sqrt{\beta_t}z_1)\\
&\quad = \sqrt{(1-\beta_t)(1-\beta_{t-1})}x_{t-2} + (\sqrt{1-(1-\beta_{t-1})(1-\beta_t)}\bar{z}_2\\
&\text{其中}\bar{z}_0, \bar{z}_1, \cdots \sim \mathcal{N}(0, I)\\
&\text{简单起见，记}\alpha_t = 1-\beta_t, \bar{\alpha}_t = \prod_{i=1}^{t}(1-\beta_i) = \prod_{i=1}^{t}\alpha_i\\
&x_t = \sqrt{\alpha_t\alpha_{t-1}}x_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\bar{z}_2, \text{其中}\bar{z}_2 \sim \mathcal{N}(0, I)\\
&\quad = \cdots\\
&\quad = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\bar{z}_t, \text{其中}\bar{z}_t \sim \mathcal{N}(0, I)
\end{aligned}$$

   因此，最终可以得到由x_0到x_t的转移为：

$$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\bar{z}_t, \text{其中}\bar{z}_t \sim \mathcal{N}(0, I)$$
$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I)$$

可以看到$\lim_{t\to\infty} q(x_t|x_0) = \mathcal{N}(0, I)$，因此最终前向加噪会越来越趋近于纯高斯噪声。

2. **逆向过程：从噪声x_T逐渐推断还原出图片x_0**

前向过程中$q(x_t|x_{t-1})$已知，如果知道了$q(x_{t-1}|x_t)$就可以完全还原图片（该分布近似于高斯分布，但是其参数不能轻易确定）。因此利用深度学习模型学习$q(x_{t-1}|x_t)$的近似$p_\theta(x_{t-1}|x_t)$。假设网络参数θ，有预测目标：

$$p_\theta(X_{0:T}) = p(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$$
$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \sum_\theta(x_t, t))$$

另外由贝叶斯公式有：

$$q(x_{t-1}|x_t, x_0) = q(x_t|x_{t-1}, x_0)\frac{q(x_{t-1}|x_0)}{q(x_t|x_0)}$$

带入前向扩散$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$，有：

$$q(x_{t-1}|x_t, x_0) \propto exp(-\frac{1}{2}(\frac{(x_t - \sqrt{\alpha}_t x_{t-1})^2}{\beta_t} + \frac{(x_{t-1} - \sqrt{\bar{\alpha}_{t-1}}x_0)^2}{1-\bar{\alpha}_{t-1}} - \frac{(x_t - \sqrt{\alpha}_t x_0)^2}{1-\bar{\alpha}_t}))$$
$$= exp(-\frac{1}{2}((\frac{\alpha_t}{\beta_t} + \frac{1}{1-\bar{\alpha}_{t-1}})x_{t-1}^2 - (\frac{2\sqrt{\alpha_t}}{\beta_t}x_t + \frac{2\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_{t-1}}x_0)x_{t-1} + C(x_t, x_0))$$

观察上式，可以看到$q(x_{t-1}|x_t, x_0)$依旧满足高斯分布：

$$q(x_{t-1}|x_t, x_0) = \mathcal{N}(x_{t-1}; \widetilde{\mu}_t(x_t, x_0), \widetilde{\beta}_t I)$$
$$\text{其中}\begin{cases}\widetilde{\mu}_t(x_t, x_0) = \dfrac{\sqrt{\alpha}_t(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}x_t + \dfrac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}x_0 \\ \widetilde{\beta}_t = \dfrac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t\end{cases}$$

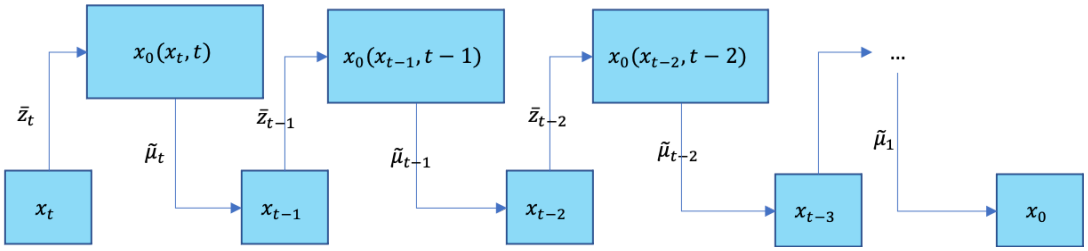
使用前向扩散的结论$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\bar{z}_t$化简$\mu$，消去$x_0$：

$$\widetilde{\mu}_t(x_t) = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\bar{z}_t)$$

即使用深度网络来预测$\bar{z}_t$噪声为$z_\theta(x_t, t)$，最后用于去噪的逆向高斯分布均值μ为：

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\bar{z}_\theta(x_t, t))$$

因此逆向推断的过程包括：



1. 每一步通过x_t和t预测高斯噪声$z_\theta(x_t,t)$，进而得到噪声均值$\mu_\theta(x_t,t)$
2. 直接将$\sum_\theta(x_t,t)$视为$\widetilde{\beta}_t$，并认为$\widetilde{\beta}_t=\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t\approx\beta_t$（后续的工作中有的将这一步也采用深度网络进行预测）
3. 根据$p_\theta(x_{t-1}|x_t)=\mathcal{N}(x_{t-1};\mu_\theta(x_t,t),\sum_\theta(x_t,t))$得到近似的$q(x_{t-1}|x_t)$，进而得到$x_{t-1}$

目标损失

本部分说明如何设置损失函数确保训练得到可靠的$z_\theta(x_t,t)$和$\sum_\theta(x_t,t)$。

真实数据$x_0\sim q(x_0)$，模型预测分布$p_\theta(x_0)$，因此总体优化目标即为二者交叉熵：

$$\mathcal{L}=\mathbb{E}_{x_0\sim q(x_0)}[-logp_\theta(x_0)]$$

之后使用变分下限VLB转化优化目标：

$$\begin{aligned}\mathcal{L}&=\mathbb{E}_{x_0\sim q(x_0)}[-logp_\theta(x_0)]=-\mathbb{E}_{q(x_0)}[log(p_\theta(x_0)\cdot\int p_\theta(x_{1:T})dx_{1:T})]\\&=-\mathbb{E}_{q(x_0)}[log(\int p_\theta(x_{0:T})dx_{1:T})]\\&=-\mathbb{E}_{q(x_0)}[log(\int q(x_{1:T}|x_0)\frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)}dx_{1:T})]\\&=-\mathbb{E}_{q(x_0)}[log(\mathbb{E}_{q(x_{1:T}|x_0)}\frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)})]\end{aligned}$$

又由琴声不等式（概率论），有：

$$\begin{aligned}&\leq-\mathbb{E}_{q(x_{0:T})}[log(\frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)})]\\&=\mathbb{E}_{q(x_{0:T})}[log(\frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})})]=\mathcal{L}_{VLB}\end{aligned}$$

进一步将$\mathcal{L}_{VLB}$转化为可计算的形式：

$$
\begin{aligned}
\mathcal{L} \leq \mathcal{L}_{VLB} &= \mathbb{E}_{q(x_{0:T})}[log(\frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})})] \\
&= \mathbb{E}_{q(x_{0:T})}[log(\frac{\prod_{t=1}^{T} q(x_t|x_{t-1})}{p_\theta(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)})] \\
&= \mathbb{E}_{q(x_{0:T})}[-log(p_\theta(x_T)) + \sum_{t=1}^{T} log(\frac{q(x_t|x_{t-1})}{p_\theta(x_{t-1}|x_t)})] \\
&= \mathbb{E}_{q(x_{0:T})}[-log(p_\theta(x_T)) + log(\frac{q(x_1|x_0)}{p_\theta(x_0|x_1)}) + \sum_{t=2}^{T} log(\frac{q(x_t|x_{t-1})}{p_\theta(x_{t-1}|x_t)})]
\end{aligned}
$$

又因为$q(x_t|x_{t-1}) \cdot q(x_{t-1}|x_0) = q(x_t, x_{t-1}|x_0) = q(x_t|x_0) \cdot q(x_{t-1}|x_t, x_0)$

（上面这个式子和前面有点歧义，我感觉应该只是表示方法不统一，大致意思是没问题的）

$$
\begin{aligned}
&= \mathbb{E}_{q(x_{0:T})}[-log(p_\theta(x_T)) + log(\frac{q(x_1|x_0)}{p_\theta(x_0|x_1)}) + \sum_{t=2}^{T} log(\frac{q(x_t|x_0) \cdot q(x_{t-1}|x_t, x_0))}{p_\theta(x_{t-1}|x_t) \cdot q(x_{t-1}|x_0)})] \\
&= \mathbb{E}_{q(x_{0:T})}[-log(p_\theta(x_T)) + log(\frac{q(x_1|x_0)}{p_\theta(x_0|x_1)}) + \sum_{t=2}^{T} log(\frac{q(x_t|x_0)}{q(x_{t-1}|x_0)}) + \sum_{t=2}^{T} log(\frac{q(x_{t-1}|x_t, x_0)}{p_\theta(x_{t-1}|x_t)})] \\
&= \mathbb{E}_{q(x_{0:T})}[-log(p_\theta(x_T)) + log(\frac{q(x_1|x_0)}{p_\theta(x_0|x_1)}) + log(\frac{q(x_T|x_0)}{q(x_1|x_0)}) + \sum_{t=2}^{T} log(\frac{q(x_{t-1}|x_t, x_0)}{p_\theta(x_{t-1}|x_t)})] \\
&= \mathbb{E}_{q(x_{0:T})}[log(\frac{q(x_T|x_0)}{p_\theta(x_T)}) - log(p_\theta(x_0|x_1)) + \sum_{t=2}^{T} log(\frac{q(x_{t-1}|x_t, x_0)}{p_\theta(x_{t-1}|x_t)})] \\
&= \mathbb{E}_{q(x_{0:T})}[D_{KL}(q(x_T|x_0))||p_\theta(x_T)) + \sum_{t=2}^{T} D_{KL}(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t)) - log(p_\theta(x_0|x_1))] \\
&= \mathbb{E}_{q(x_{0:T})}[L_T + \sum_{t=2}^{T} L_t + L_0]
\end{aligned}
$$

上面式子中$L_T = D_{KL}(q(x_T|x_0))||p_\theta(x_T))$中$x_T, x_0$都是已知的，因此优化时可以忽略这一项，$L_t$项则迫使$p_\theta$逼近于要求的$q(x_{t-1}|x_t, x_0) = \mathcal{N}(x_{t-1}; \widetilde{\mu}_t(x_t, x_0), \widetilde{\beta}_t I)$。

损失参数化

整理前后向推导，有：

$$
\begin{aligned}
p_\theta(x_{t-1}|x_t) &= \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\bar{z}_\theta(x_t, t)), \Sigma_\theta(x_t, t)) \\
q(x_{t-1}|x_t, x_0) &= \mathcal{N}(x_{t-1}; \widetilde{\mu}_t(x_t, x_0) = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\bar{z}_t), \Sigma_\theta(x_t, t))
\end{aligned}
$$

因此，定义参数化的L_t来降低μ_θ和$\widetilde{\mu}_t$之间的差距：

$$
\begin{aligned}
L_t &= \mathbb{E}_{x_0,\bar{z}_t}[\frac{1}{2||\Sigma_\theta(x_t,t)||_2^2}||\widetilde{\mu}_t(x_t,x_0)-\mu_\theta(x_t,t)||_2^2] \\
&= \mathbb{E}_{x_0,\bar{z}_t}[\frac{1}{2||\Sigma_\theta(x_t,t)||_2^2}||\frac{1}{\sqrt{\alpha_t}}(x_t-\frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\bar{z}_t)-\frac{1}{\sqrt{\alpha_t}}(x_t-\frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\bar{z}_\theta(x_t,t))||_2^2] \\
&= \mathbb{E}_{x_0,\bar{z}_t}[\frac{\beta_t^2}{2\alpha_t(1-\bar{\alpha}_t)||\Sigma_\theta(x_t,t)||_2^2}||(\bar{z}_\theta(x_t,t)-\bar{z}_t)||_2^2] \\
&= \mathbb{E}_{x_0,\bar{z}_t}[\frac{\beta_t^2}{2\alpha_t(1-\bar{\alpha}_t)||\Sigma_\theta(x_t,t)||_2^2}||(\bar{z}_\theta(\sqrt{\bar{\alpha}_t}x_0+\sqrt{1-\bar{\alpha}_t}\bar{z}_t,t)-\bar{z}_t)||_2^2]
\end{aligned}
$$

Model

实际上，最终使用神经网络优化的是简化版的L_t（略去了常数项）：

$$
L_t^{simple}=\mathbb{E}_{t\sim[1,T],x_0,\bar{z}_t}[||\bar{z}_t-\bar{z}_\theta(\sqrt{\bar{\alpha}_t}x_0+\sqrt{1-\bar{\alpha}_t}\bar{z}_t,t)||^2]
$$

具体的算法步骤如下：

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1,\dots,T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0},\mathbf{I})$
5: Take gradient descent step on
$\nabla_\theta \left\|\boldsymbol{\epsilon}-\boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0+\sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon},t)\right\|^2$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0},\mathbf{I})$
2: **for** $t=T,\dots,1$ **do**
3: $\mathbf{z}\sim\mathcal{N}(\mathbf{0},\mathbf{I})$ if $t>1$, else $\mathbf{z}=\mathbf{0}$
4: $\mathbf{x}_{t-1}=\frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t-\frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t,t)\right)+\sigma_t\mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Denoising Diffusion Implicit Models——DDIM

Intro

目的：将上文DDPM中的方法进行优化，加速训练。

原理

DDPM的高质量生成依赖于较大的T （一般为1000或以上），这就导致diffusion的前向过程非常缓慢。DDIM中提出了一种牺牲多样性来换取更快推断的手段。

由DDPM中前向推导可知：

$$
\begin{aligned}
&x_{t-1}=\sqrt{\bar{\alpha}_{t-1}}x_0+\sqrt{1-\bar{\alpha}_{t-1}}\bar{z}_{t-1} \\
&\text{将}z_t\text{的一部分固定为由}x_t\text{和}x_0\text{所决定的定值，缩小方差：} \\
&x_{t-1}=\sqrt{\bar{\alpha}_{t-1}}x_0+\sqrt{1-\bar{\alpha}_{t-1}-\sigma_t^2}\bar{z}_t+\sigma_t z_t \\
&\quad=\sqrt{\bar{\alpha}_{t-1}}x_0+(\sqrt{1-\bar{\alpha}_{t-1}-\sigma_t^2})\cdot\frac{x_t-\sqrt{\bar{\alpha}_t}x_0}{\sqrt{1-\bar{\alpha}_t}}+\sigma_t z_t \\
&\Rightarrow q_\sigma(x_{t-1}|x_t,x_0)=\mathcal{N}(x_{t-1};(\sqrt{1-\bar{\alpha}_{t-1}-\sigma_t^2})\cdot\frac{x_t-\sqrt{\bar{\alpha}_t}x_0}{\sqrt{1-\bar{\alpha}_t}},\sigma_t^2 I)
\end{aligned}
$$

实际上DDIM中将σ_t设置为可变的用来平衡结果随机性和生成速度：

$$
\sigma_t^2(\eta)=\eta\cdot\widetilde{\beta}_t=\eta\cdot(\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t)
$$

- 当$\eta = 1$，DDIM与DDPM完全一致
- 当$\eta = 0$，DDIM生成的结果是不具有随机性的

1） $\sigma_{t,\theta}^2 = \Sigma_\theta(x_t, t)$ 相当于模型学习的方差，DDPM称为learned，实际没有使用（但是GLIDE使用的是这种方差）。

2） $\sigma_{t,s}^2 = \tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$ ，由(8-1)得到，DDPM称为fixedsmall，用于celebahq和lsun。

3） $\sigma_{t,l}^2 = \beta_t$ ，DDPM称为fixedlarge，用于cifar10，注意 $\sigma_{t,l} > \sigma_{t,s}$ ，**fixedlarge的方差大于fixedsmall的。**

High-Resolution Image Synthesis with Latent Diffusion Models——LDM

Intro

目的：将Diffusion Model中原本对于图像维度进行的操作转换到Latent Space中，从而减少训练成本和推理速度，实现更高分辨率图片的生成。另外，论文还将多模态的Transformer模块移植到DM中预测均值的Unet中，实现图像文本生成。

Model

LDM主要分为两个阶段：

1. 训练一个自动编码器，将图像空间压缩到Latent Space中，这一部分的结果可以迁移到其他下游任务中，不需要多次训练

   主要步骤：

   1. 使用一个编码器将输入图像压缩到潜在空间：$z = \mathcal{E}(x) \in \mathbb{R}^{h \times w \times c}$，一般缩放率$f = 2^m$，即 $h = \frac{H}{2^m}, w = \frac{W}{2^m}$
   2. 利用 一个解码器重建图片$\tilde{x} = \mathcal{D}(z)$
   3. loss中添加合理的正则项，论文中考虑了KL-reg和VQ-reg两种
2. 在Latent Space中训练一个经过改造的DM（Unet中插入transformer块）

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\Big(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}\Big) \cdot \mathbf{V}$$
$$\text{where } \mathbf{Q} = \mathbf{W}_Q^{(i)} \cdot \varphi_i(\mathbf{z}_i),\ \mathbf{K} = \mathbf{W}_K^{(i)} \cdot \tau_\theta(y),\ \mathbf{V} = \mathbf{W}_V^{(i)} \cdot \tau_\theta(y)$$
$$\text{and } \mathbf{W}_Q^{(i)} \in \mathbb{R}^{d \times d_\epsilon^i},\ \mathbf{W}_K^{(i)}, \mathbf{W}_V^{(i)} \in \mathbb{R}^{d \times d_\tau},\ \varphi_i(\mathbf{z}_i) \in \mathbb{R}^{N \times d_\epsilon^i},\ \tau_\theta(y) \in \mathbb{R}^{M \times d_\tau}$$

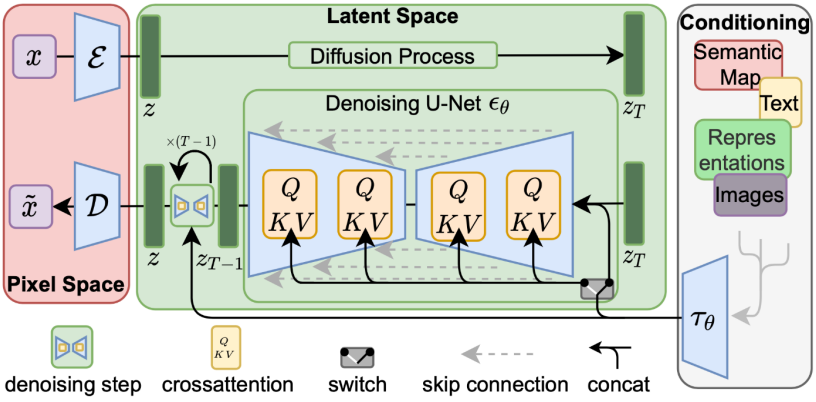


Fig. 9. The architecture of latent diffusion model. (Image source: Rombach & Blattmann, et al. 2022)

其优化目标为：

$$L_{LDM} = \mathbb{E}_{\mathcal{E}(x),y,\sim N(0,1),t}[||\epsilon - \epsilon_\theta(z_t, t, \tau_\theta(y))||_2^2]$$

Results

论文对比了不同latent压缩率对于生成结果质量FID的影响，提出了Classifier free的重要性。同时比较了在UNet中加入cross-attention或不加入两种情况的生成情况。最后，论文中还讨论了模型在低分辨率图像训练后用于生成高分辨率图像的能力，这使得模型异常强大。