信息量

All events have their own quality of information.

任何**事件**都会承载着一定的信息量，包括已经发生的事件和未发生的事件。如昨天下雨这个已知事件，因为已经发生，既定事实，那么它的信息量就为0。如明天会下雨这个事件，因为未有发生，那么这个事件的信息量就大。

从上面例子可以看出信息量与事件的发生概率相关，事件发生的概率越小，其信息量越大。这也很好理解，比如"狗咬人"事件不如"人咬狗"事件信息量大。

An event with a high probability to happen get much more amount of quality of information. So,how can we measure this? The mystery log function.

我们已知某个**事件**的信息量是与它发生的概率有关，那我们可以通过如下公式计算信息量：

假X是一个离散型随机变量，其取值集合为χ,概率分布函数$p(x) = Pr(X = x), x\epsilon\chi$,则定义事件$X = x_0$的信息量为：$I(x_0) = -log(p(x_0))$

熵

熵最早是物理学的概念。

在**热力学**中，用于表示一个系统的无序程度。

一个系统越无序，熵越高。每一个孤立的系统一定会随着时间的发展而总混乱程度也会随之增加，而且过程是单向的，像时间一去不复返一样不可逆。

熵增是一个从有序到无序的过程，而熵减则相反，从无序到有序的过程。

我们随着时间的流逝一天天在变老，身体一天天在变差，这就是一个熵增的过程。

在**信息论**中，用与衡量一个随机变量的不确定性。一个随机变量不确定性越高，熵越高。

我们知道：当一个事件发生的概率为 $\boldsymbol{p(x)}$ ，那么它的信息量是 $\boldsymbol{-log(p(x))}$ 。

那么如果我们把这个事件的所有可能性罗列出来，就可以求得该事件信息量的期望，

针对随机变量来说，**信息量的期望就是熵**，所以熵的公式为：

假设 事件X共有n种可能，发生 $\boldsymbol{x_i}$ X_i的概率为$p(X_i)$ $\boldsymbol{p(x_i)}$ ，那么该事件的熵$H(X)$ $\boldsymbol{H(X)}$ 为：

$$H(x) = -\sum_{i=1}^{n} p(x_i)log(p(x_i))$$

然而有一类比较特殊的问题，比如投掷硬币只有两种可能，字朝上或花朝上。买彩票只有两种可能，中奖或不中奖。我们称之为0-1分布问题（伯努利分布/二项分布的特例），对于这类问题，熵的计算方法可以简化为如下算式：

$$H(x) = -\sum_{i=1}^{n} p(x_i)log(p(x_i)) = -p(x)log(p(x)) - (1-p(x))log(1-p(x))$$

考虑一个**categorical** 分布一个表格：

$p(x_0)$	$p(x_1)$	$p(x_2)$	**熵**
1	0	0	0
$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{3}{2}log2$
$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$log(3)$

一个分布很集中，则熵低，对应系统很有序；一个分布很分散（各种情况都有可能）则熵高，对应系统很无序。

KL散度

设随机变量有两个单独的概率分布， $p(x)$和$q(x)$衡量两个分布之间的差异，我们怎么做呢？KL散度

在机器学习中，P往往用来表示样本的真实分布用来， Q表示模型所预测的分布，那么KL散度就可以计算两个分布的差异，即$Loss$损失值。

$$D_{KL}(p||q) = \sum_{i=1}^{n} p(x_i) log(\frac{p(x_i)}{q(x_i)})$$

从KL散度公式中可以看到Q的分布越接近P（Q分布越拟合P），那么散度值越小，即损失值越小。

因为对数函数是凸函数，所以KL散度的值为非负数。

当$P = Q$时，KL散度等于0，两个分布越远，KL散度值越大，无上限。

有时会将KL散度称为KL距离，但严格来说，散度不能称为距离，因为它并不满足距离的性质：

1. KL散度不是对称的:$D_{KL}(p||q) \neq D_{KL}(q||p)$
2. KL散度不满足三角不等式。

针对散度的信息论解释是：用概率分布Q来近似概率分布P 时所造成的信息量的损失。--信息损耗

交叉熵

我们将KL散度公式进行变形：

$$D_{KL}(p||q) = \sum_{i=1}^{n} p(x_i) log(\frac{p(x_i)}{q(x_i)}) = \sum_{i=1}^{n} p(x_i) log(p(x_i)) - \sum_{i=1}^{n} p(x_i) log(q(x_i))$$
$$= -H(p(x)) + [-\sum_{i=1}^{n} p(x_i) log(q(x_i))]$$

等式的前一部分恰巧就是p的熵，等式的后一部分，就是交叉熵：

$$H(p,q) = -\sum_{i=1}^{n} p(x_i) log(q(x_i))$$

在机器学习中，我们需要评估label和predicts之间的差距，使用KL散度刚刚好，即 $\boldsymbol{D_{KL}(y||\tilde{y})}$ ，由于KL散度中的前一部分$-H(y)$是label 的分布，与predicts无关，故在优化过程中，只需要关注交叉熵就可以了。所以一般在机器学习中直接用用交叉熵做loss，评估模型。

CrossEntropyLoss in Pytorch

假设我们现在需要处理一个K分类问题，可以在一个网络的最后直接一个K个神经元的全连接层，然后直接使用$nn.CrossEntropyLoss$ 定义损失函数

```
class Net(torch.nn.Module):
```

```
    def __init__(self, n_feature, n_hidden, n_output):
        super(Net, self).__init__()
        self.hidden = torch.nn.Linear(n_feature, n_hidden)   # hidden layer
        self.out = torch.nn.Linear(n_hidden, n_output)   # output layer

    def forward(self, x):
        x = F.relu(self.hidden(x))      # activation function for hidden layer
        x = self.out(x)
        return x

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# Forward pass
outputs = model(images)
loss = criterion(outputs, labels)

# Backward and optimize
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

针对一个分类实例来说，outputs为每一个神经元输出组成的向量，

softmax

$$softmax(x_i) = \frac{exp(x_i)}{\sum_{j=0}^{n} exp(x_j)}$$

完成从一系列数值到概率的归一， 并且在归一后能够凸显最大值，并抑制远低于最大值的其他分量。例如输入[1,2,3,4,1,2,3]对应softmax 函数的输出值为[0.024, 0.064, 0.175, **0.475**, 0.024, 0.064, 0.175]

NLLLOSS（The negative log likelihood loss. It is useful to train a classification problem with C classes.)

相当于求取log概率向量同label 之间"差异"。

$$NLLLoss = L(P,Q) = -\sum_{y} P(y)Q(y)$$

注意这里不是交叉熵(H)，没有log

交叉熵：

$$CrossEntropyLoss(P,Q) = H(P, softmax(Q)) = -\sum_{y} P(y)log(softmax(Q(y)))$$
$$= NLLLoss(P, logsoftmax(Q))$$

This criterion combines `nn.LogSoftmax()` and `nn.NLLLoss()` in one single class.

```
import torch
import torch.nn.functional as F

output = torch.randn(3, 5, requires_grad=True)
target = torch.tensor([1, 0, 4])


y1 = F.cross_entropy(output,target)
y2 = F.nll_loss(F.log_softmax(output,dim=1), target)

# y1 == y2
print(y1)
print(y2)
```

CrossEntropyLoss 之前不需要加softmax 层，且标签要是one -hot 形式 ,例如分类问题。

举例来看一下CrossEntropyLoss 的计算过程

在图片单标签分类时，输入m张图片，输出一个m*N的Tensor，其中N是分类个数。比如输入3张图片，分三类，最后的输出是一个*3*3的Tensor，举个例子

```
[131]: input=torch.randn(3,3)
       input
```

```
[131]: tensor([[-0.1342, -2.5835, -0.9810],
               [ 0.1867, -1.4513, -0.3225],
               [ 0.6272, -0.1120,  0.3048]])
```


第123行分别是第123张图片的结果，假设第123列分别是猫、狗和猪的分类得分。
可以看出模型认为第123张都更可能是猫。

此时，我们不能简单取最大， 认为是猫就OK 了， 在训练过程中我们要致力于提升模型的泛化能力，所以我们必须要去度量模型的输出同标签之间的距离。

因此，交叉熵上场！交叉熵是如何运算的呢?

首先，对每一行使用Softmax，这样可以得到每张图片的概率分布。

```
[132]: sm=nn.Softmax(dim=1)
```

```
[133]: sm(input)
```

```
[133]: tensor([[0.6600, 0.0570, 0.2830],
               [0.5570, 0.1083, 0.3347],
               [0.4542, 0.2169, 0.3290]])
```


这里dim的意思是计算Softmax的维度，这里设置dim=1，可以看到每一行的加和为1。比如第一行0.6600+0.0570+0.2830=1。
然后对Softmax的结果取自然对数：

```
[139]: torch.log(sm(input))

[139]: tensor([[-0.4155, -2.8648, -1.2623],
               [-0.5852, -2.2232, -1.0945],
               [-0.7893, -1.5285, -1.1117]])
```


Softmax后的数值都在0~1之间，所以ln之后值域是负无穷到0。
NLLLoss的结果就是把上面的输出与Label对应的那个值拿出来，取负，求均值。
假设我们现在Target是[0,2,1]（第一张图片是猫，第二张是猪，第三张是狗）。第一行取第0个元素，第二行取第2个，第三行取第1个，去掉负号，结果是：[0.4155,1.0945,1.5285]。再求个均值，结果是：

```
[155]: (0.4155+1.0945+1.5285)/3

[155]: 1.0128333333333333
```

下面使用NLLLoss函数验证一下：

```
[156]: loss=nn.NLLLoss()

[157]: target=torch.tensor([0,2,1])

[159]: loss(input,target)

[159]: tensor(1.0128)
```


CrossEntropyLoss就是把以上Softmax–Log–NLLLoss合并成一步，我们用刚刚随机出来的input直接验证一下结果是不是1.0128：

```
[160]: loss=nn.CrossEntropyLoss()

[167]: input=torch.tensor([[-0.1342, -2.5835, -0.9810],
                           [ 0.1867, -1.4513, -0.3225],
                           [ 0.6272, -0.1120,  0.3048]])

[168]: target=torch.tensor([0,2,1])

[169]: loss(input,target)

[169]: tensor(1.0128)
```


JS divergence:

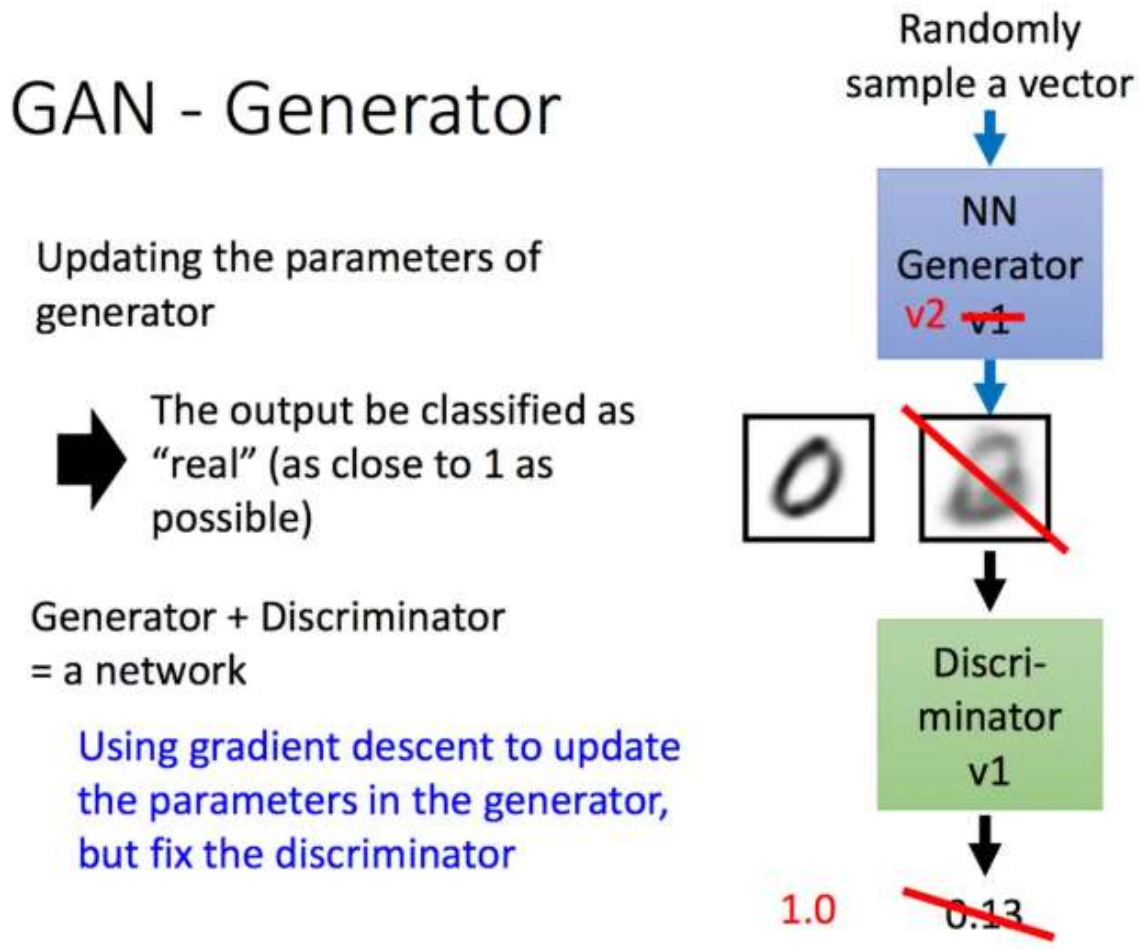
$$JS(P||Q) = \frac{1}{2}KL(p||\frac{p+q}{2}) + \frac{1}{2}KL(q||\frac{p+q}{2})$$

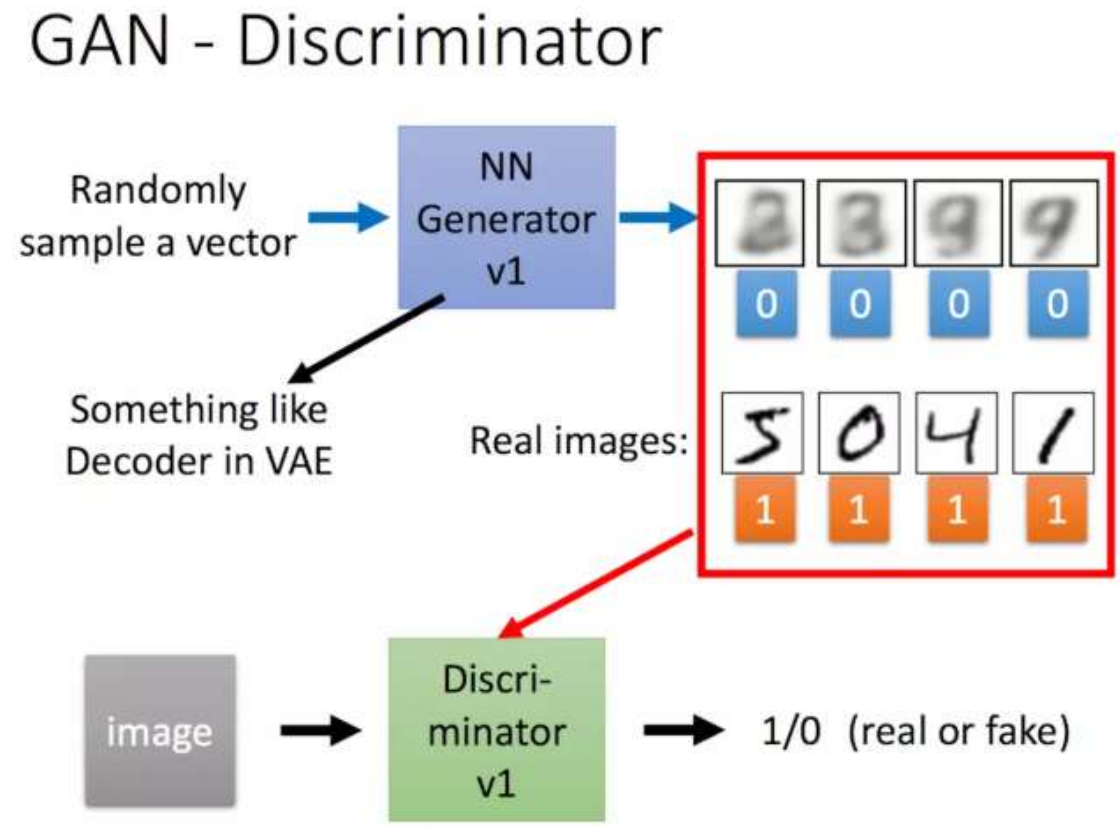
该散度是对称的。

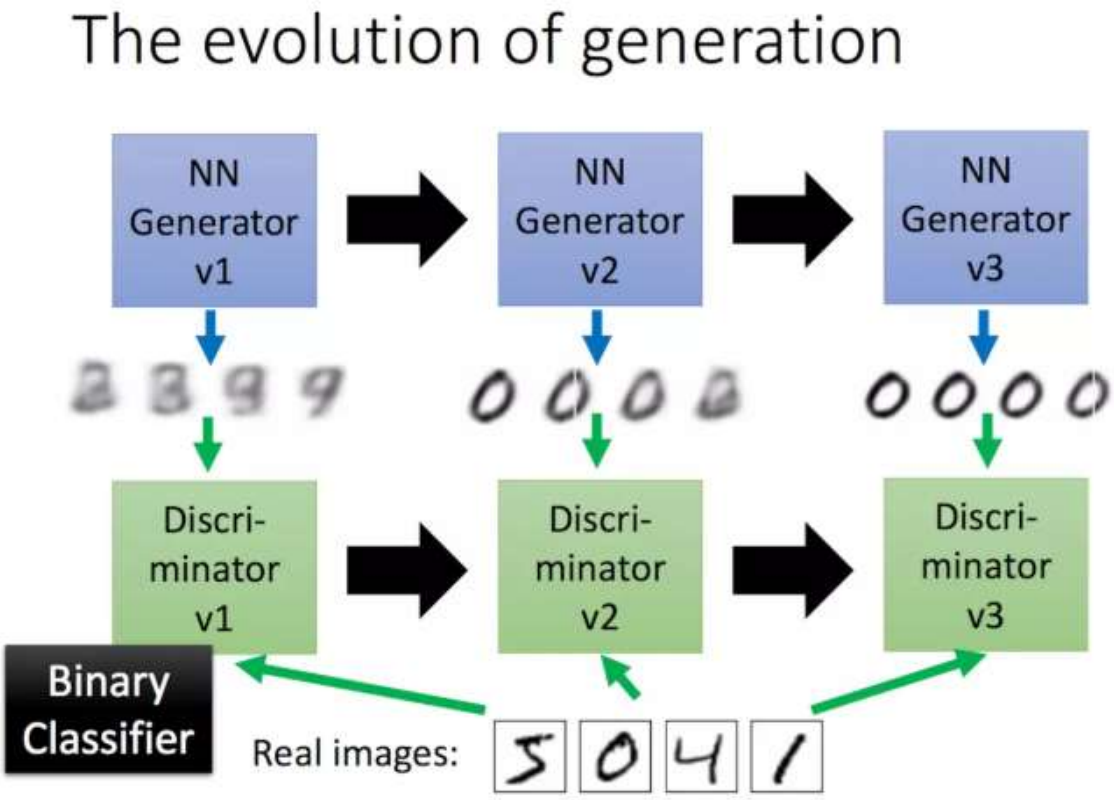
GAN

GAN中文叫做生成对抗网络，就是双方互相博弈（生成器和鉴别器，鉴别器要最大化，求出分布差异，生成器要最小化，降低差异），最后达到一种平衡状态。我们用GAN做图片生成，做风格学习，做语音识别或者做语音和声纹分离，本质上都是在做概率分布的学习，我们希望我们的模型可以去接近真实分布。

简单的说，就是我希望训练一个模型，模型用来学习真实样本的分布，很多时候我们是不知道真实样本的分布的，但是它确实是存在的。那我们要怎么来学习这个分布呢，最常见的就是用极大似然估计，即我们希望我们的模型生成的样本，看起来很最像是真实的分布生成的样本。换句话说，我们希望模型的样本分布和真实样本分布差距越来越小，因此引入了散度的概念，即两个分布之间的差异，不是距离，应该是信息损耗。







为什么需要D， 只有G 行不行？？？

如果D 太强，那么G 不知道如何去靠近

如果D 太弱，那么G 生成的数据可能永远无法向真实的分布靠近

如何把握D 的平衡？？？

模型

GAN's objective:

$$\min_G \max_D \Big(\mathcal{L}(D,G) \equiv \mathbb{E}_{\mathbf{x}\sim p_r(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))] \Big)$$

$$= \mathbb{E}_{\mathbf{x}\sim p_r(\mathbf{x})}[\log D(\mathbf{x})] \underbrace{+\mathbb{E}_{x\sim p_g(x)}[\log(1-D(\mathbf{x})]}_{\text{alternative expression}}$$

推导

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
  - Sample minibatch of m examples $\{x^{(1)}, \ldots, x^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$.
  - Update the discriminator by ascending its stochastic gradient: 随机梯度.

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right) \right].$$

  end for

  - Sample minibatch of m noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

4.1 Global Optimality of $p_g = p_{\text{data}}$

$\int_x P_{data}(x)\log D(x) + P_G(x)\log(1-D(x))\,dx$

We first consider the optimal discriminator D for any given generator G.

Proposition 1. *For G fixed, the optimal discriminator D is*

$$D_G^*(\boldsymbol{x}) = \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})} \quad (2)$$

求导，令=0.

Proof. The training criterion for the discriminator D, given any generator G, is to maximize the quantity $V(G, D)$

$$\begin{aligned} V(G,D) &= \int_{\boldsymbol{x}} p_{\text{data}}(\boldsymbol{x}) \log(D(\boldsymbol{x}))dx + \int_z p_z(z)\log(1-D(g(z)))dz \\ &= \int_{\boldsymbol{x}} p_{\text{data}}(\boldsymbol{x}) \log(D(\boldsymbol{x})) + p_g(\boldsymbol{x})\log(1-D(\boldsymbol{x}))dx \end{aligned} \quad (3)$$

For any $(a, b) \in \mathbb{R}^2 \setminus \{0, 0\}$, the function $y \to a\log(y) + b\log(1-y)$ achieves its maximum in $[0, 1]$ at $\frac{a}{a+b}$. The discriminator does not need to be defined outside of $Supp(p_{\text{data}}) \cup Supp(p_g)$, concluding the proof. □ ⇒ 求导

Note that the training objective for D can be interpreted as maximizing the log-likelihood for estimating the conditional probability $P(Y = y|\boldsymbol{x})$, where Y indicates whether $\boldsymbol{x}$ comes from p_{data} (with $y = 1$) or from p_g (with $y = 0$). The minimax game in Eq. 1 can now be reformulated as:

$$\begin{aligned} C(G) &= \max_D V(G,D) \\ &= \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}[\log D_G^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}[\log(1 - D_G^*(G(\boldsymbol{z})))] \\ &= \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}[\log D_G^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim p_g}[\log(1 - D_G^*(\boldsymbol{x}))] \\ &= \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\left[\log \frac{p_{\text{data}}(\boldsymbol{x})}{P_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right] + \mathbb{E}_{\boldsymbol{x}\sim p_g}\left[\log \frac{p_g(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right] \end{aligned} \quad (4)$$

$$* \Rightarrow \int_x P_{data}(x)\cdot\log\frac{P_{data}(x)}{\frac{P_{data}(x)+P_G(x)}{2}} + \int_x P_G(x)\cdot\log\frac{P_G(x)}{\frac{P_{data}(x)+P_G(x)}{2}}$$

$$= \int_x P_{data}(x)\cdot\log\frac{P_{data}(x)}{P_{data}(x)+P_G(x)} + \int_x P_G(x)\log\frac{P_G(x)}{P_{data}(x)+P_G(x)} + \log 4$$

Theorem 1. *The global minimum of the virtual training criterion $C(G)$ is achieved if and only if $p_g = p_{data}$. At that point, $C(G)$ achieves the value $-\log 4$.*

Proof. For $p_g = p_{\text{data}}$, $D_G^*(\boldsymbol{x}) = \frac{1}{2}$, (consider Eq. 2). Hence, by inspecting Eq. 4 at $D_G^*(\boldsymbol{x}) = \frac{1}{2}$, we find $C(G) = \log\frac{1}{2} + \log\frac{1}{2} = -\log 4$. To see that this is the best possible value of $C(G)$, reached only for $p_g = p_{\text{data}}$, observe that

$$\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}[-\log 2] + \mathbb{E}_{\boldsymbol{x}\sim p_g}[-\log 2] = -\log 4$$

and that by subtracting this expression from $C(G) = V(D_G^*, G)$, we obtain:

$$C(G) = -\log(4) + KL\left(p_{\text{data}} \middle\| \frac{p_{\text{data}}+p_g}{2}\right) + KL\left(p_g \middle\| \frac{p_{\text{data}}+p_g}{2}\right) \tag{5}$$

where KL is the Kullback–Leibler divergence. We recognize in the previous expression the Jensen–Shannon divergence between the model's distribution and the data generating process:

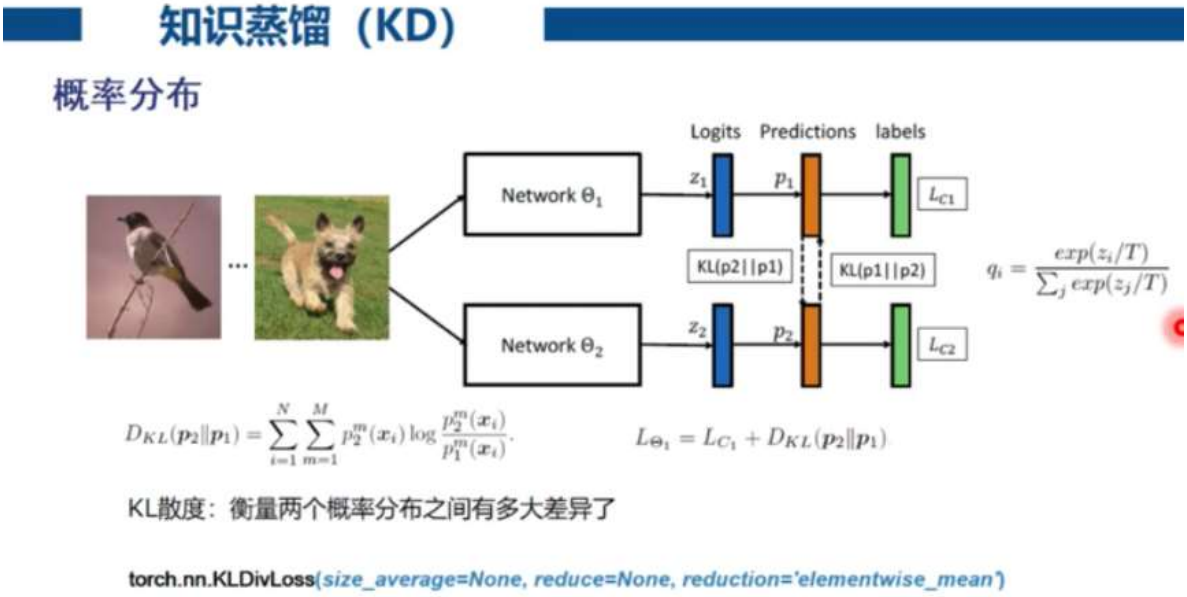
$$C(G) = -\log(4) + 2\cdot JSD\left(p_{\text{data}} \| p_g\right) \tag{6}$$

Since the Jensen–Shannon divergence between two distributions is always non-negative and zero only when they are equal, we have shown that $C^* = -\log(4)$ is the global minimum of $C(G)$ and that the only solution is $p_g = p_{\text{data}}$, i.e., the generative model perfectly replicating the data generating process. □

4.2 Convergence of Algorithm 1

Proposition 2. *If G and D have enough capacity, and at each step of Algorithm 1, the discriminator is allowed to reach its optimum given G, and p_g is updated so as to improve the criterion*

知识蒸馏



ACTOR-CRITIC

演员-评论员算法（Actor-Critic Algorithm）是一种结合策略梯度和时序差分学习的强化学习方法。其中演员（Actor）是指策略函数 $\pi_\theta(s,a)$，即学习一个策略来得到尽量高的回报，评论员（Critic）是指值函数 $V_\phi(s)$，对当前策略的值函数进行估计，即评估 actor 的好坏。借助于值函数，Actor-Critic 算法可以进行单步更新参数，不需要等到回合结束才进行更新。

在 Actor-Critic 算法中的策略函数 $\pi_\theta(s,a)$ 和值函数 $V_\phi(s)$ 都是待学习的函数，需要在训练过程中同时学习。

Actor（玩家）：为了玩转这个游戏得到尽量高的reward，需要一个策略：输入state，输出action，即上面的第2步。（可以用神经网络来近似这个函数。剩下的任务就是如何训练神经网络，得更高的reward。这个网络就被称为actor）

Critic（评委）：因为actor是基于策略policy的所以需要critic来计算出对应actor的value来反馈给actor，告诉他表现得好不好。所以就要使用到之前的Q值。（当然这个Q-function所以也可以用神经网络来近似。这个网络被称为critic。）