图像生成报告

1. VAE 的原理

1.1. 从 AE 的思想开始

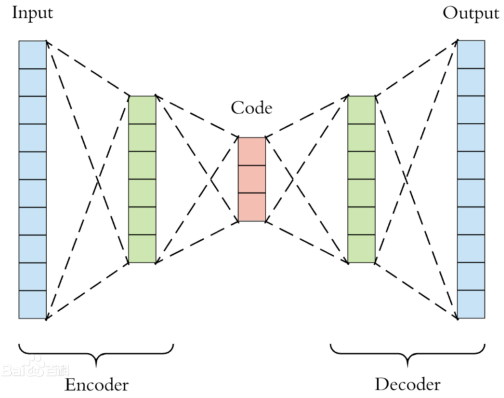

图 1: 自编码器

AE 即 auto-encoder，自编码器，是一种无监督的数据维度压缩和数据特征表达方法。

以生成图片的任务为例，AE 的思想是，输入一张图片，经过 Encoder 编码得到 Code，对 Code 经过 Decoder 解码，得到另一张图片，我们训练的目标是让 Input 和 Output 越相近越好。完成训练后我们就可以单独将 Decoder 拿出来，通过人为设置不同的 Code，从而生成不同的图像。

总的来说 AE 通过将输入数据压缩为一个中间表示，再将该中间表示解码为与原始数据相似的重构数据，从而实现对原始数据的压缩和还原。

1.2. VAE 的原理

实际上，AE 太简单，它的效果通常不会很好。因此产生了 VAE。VAE 是指 Variational Autoencoder，也是一种无监督的机器学习技术，类似于 AE，但其利用了概率方法，使得对于生成数据具有更强的表达能力和样本生成能力。

VAE 中的 Encoder 和 Decoder 和 AE 中的结构类似，没有太复杂的改动。但是中间的部分使用了一些特殊的技巧。Encoder 会生成两组向量，

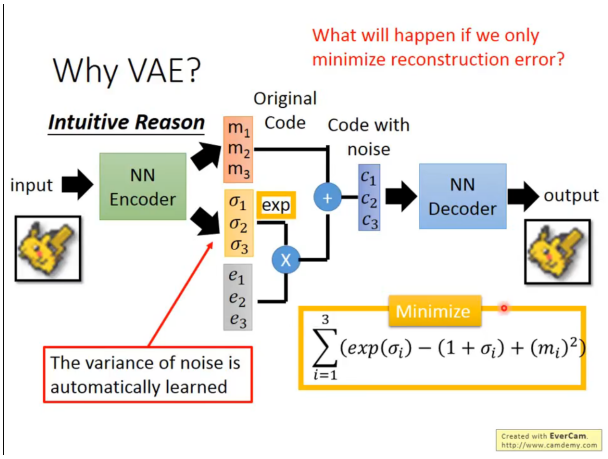

图 2: 自编码器

m 和 sigma，m 代表的就是 code，这和 AE 中的想法一样；sigma 代表了噪声的方差，这一部分是交给机器学习的。同时，还会有第三个向量 e，它是从一个 normal distribution 中采样出来的，将 sigma 向量取 e 指数然后和 e 向量相乘，此时得到的值就代表了一个随机的噪声，之后将这个噪声加到原本的 code 向量 m 上，得到最终的带着噪声的 code，记为 c。

以上的过程公式如下：

$$c_i = exp(\sigma_i) \times e_i + m_i \tag{1}$$

以上的这个步骤称为重参数化 (reparametrize)，之后这个新的 Code 会被送入 Decoder，进行解码后得到 output，之后优化重构误差，这里由于中间重参数化的复杂性，重构误差的函数也进行了部分的更改，保证误差反传的正确性和模型的正确性。

之后和 AE 一样，我们可以通过调整 code 来生成不同的图像。VAE 由于使用了随机的取样，因此它一个显著的优点就是创造了连续的 code space，我们可以人为调整 code 来观察到随着 code 的变化，图像的变化趋势。这也是我们后面任务中想要探究的内容。

VAE 也有一个很显著的缺陷，它从来没有学着怎样去生成一个图像，它学的内容一直是怎样让结果和输入越接近越好，因此它可能甚至不懂图像是什么，但它只要照着做就可以完成任务。后面的相关研究，生成对抗网络 GAN 等等，就解决了这个缺陷，取得了更好的结果。

2. 实验过程与分析

首先我先编写了 vae 的大体结构，编写训练和评估以及生成图片的代码。

自己训练了一些后发现模型的 loss 很大，生成的图片效果也比较一般，因此我考虑先优化模型，再进行前面两个任务的探究。

2.1. 优化模型

我逐步进行了模型的优化。

第一个我想到的是对图像进行正则化，我尝试使用批量归一化 BatchNorm 提高模型的鲁棒性。具体的做法是在模型中加入 bn 层。

加入正则化后的 loss 为：

epoch_average_batch_loss : 3786.125208

这比最初的默认效果好，所以我保留了正则化 bn 层。

第二我想到的是使用更深的网络。因为 MNIST 的数据本身不复杂，因此过于太深的网络也不会有太大的提升，所以我只将网络加深了一层。

加深后的 loss 为：

epoch_average_batch_loss : 3496.538722

这比其不加深的网络有所提升，因此我保留了更深的网络。

第三我想到的是更换不同的优化器，不同的优化器在不同的任务上会呈现出不同的性能。这一点调参还是比较基础的。我尝试了几种不同的优化器

可以看，其中一些优化器的性能偏差，其他

表 1: 表标题

optimizer	*epoch_average_batch_loss*
adam	3496.538722
adagrad	3915.571762
RMSprop	3499.479338
adamW	3471.771419
adadelta	4120.296003

的几个优化器的效果都很优异，没有过于明显的差距。因此在后面的所有实验里我都统一使用 adamW 优化器。

第四点我想到了调整 batchsize 的大小，在之前的实验中我就体会过 batchsize 对模型的性能和训练速率的影响，之前我默认使用了 100 的 batchsize，这里我调整为 64 的 batchsize，得到了这样的结果。

epoch_average_batch_loss : 2237.570835

效果有了显著的提升，之后的训练我都使用 64 的 batchsize。

前面的调参调优都是比较一般的方法，对于每一种模型都可以进行这些步骤。第五点我就想到要从模型本身的特点出发来进行调参。在 VAE 中隐层的维度决定了 code space 的大小，在之前我都使用的是维度为 2 的 code space。我想到增加维度，这样可以扩大 code space，使模型的表征能力改变，这对模型的性能会有影响。

我尝试了将隐层的维度改为 10 维，得到了如下的结果：

epoch_average_batch_loss : 1800.064726

我又改成了 20 维，得到了如下的结果：

可以看到，增大了隐层的维度缺失让模型性能有一定程度的改变，因为 code space 变大，这个 code 的表征能力就会更强，能做到更细节的表征。但是也可以看到，20 维度效果和 10 维度其实没什么差别。我个人认为是 MNIST 的数据本身比较简单，不需要太复杂的表征。

虽然增大隐层的维度会对提升模型的性能，但是对模型的客观性会有比较大的挑战，它生成的

epoch_average_batch_loss : 1800.181220

图片就比较奇怪，这一点我会在后面继续讨论。

第六点我还想到了使用 dropout 技术，我在 20 隐层维度的条件下，给网络中加入了参数为 0.2 的 dropout 层，结果如下：

epoch_average_batch_loss : 2171.465144

效果并不是很好，这有可能是参数调整不当导致的。但是对 dropout 的太过精细的调整没有十分必要，因此我选择不适用 dropout 层。

第七点我选择了很大幅度的修改，因为我发现比较常规的调参方法我都试验过了，已经可以说是比较完整了，因此我不能再从超参数上继续改动了。我仔细观察了网络结构，发现网络的 encoder 和 decoder 层使用的是全连接层。我仔细思考了编码器和解码器的作用，发现可以将它们的全连接层改成卷积层，第一这样不会改变模型的功能，第二卷积层在处理图像的数据时往往会比全连接层有更好的效果。

我更改了所有 encoder 和 decoder 的结构，将其换成了卷积层，这里我使用了 10 维的隐层，得到的结果如下：

epoch_average_batch_loss : 1789.080109

我发现更改为卷积层后，对于 10 维的情况，loss 的提升并没有很大。我又将维度减小为 2，再次运行，得到了以下的结果。

可以看到，在隐层维度设置为 2 时，模型的 loss 比全连接层更高，效果其实不如以前。

我的理解如下：MNIST 数据集本身比较简单，对它的表征不需要过于复杂的结构。增大隐层的维度可以提高表征能力，更换为卷积层也是在提高表征能力。一旦模型能够很好表征数据集后，再增强表征能力也不会有性能的提升了。这也就是为什么在隐层维度为 10 时，模型性能差别不大了，因为此时 10 隐层维度已经能够很好表征，再更换卷积层也不会有太高的提升。而在隐层维度为 2 时，卷积不如全链接层，这一点我认为是我的网络搭建的模型，因为卷积网络中还有很多超参数需要调整，可以看到卷积的 loss 和全连接相差没有很大，我认为通过模型的优化能够取得比全连接更好的效果。

epoch_average_batch_loss : 2434.495113

使用卷积有一个比较好的地方就是，卷积能更好提取图像中的特征，因此它生成的图片质量整体来说是比全连接好很多的。

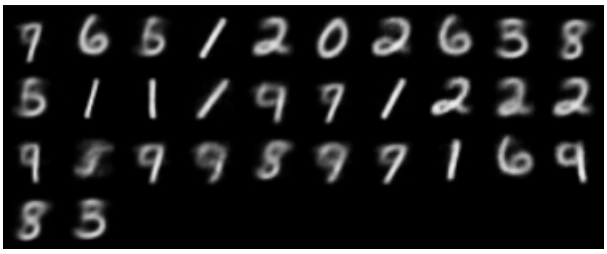

图 3: 隐层向量为 *2* 维时的卷积层生成图像

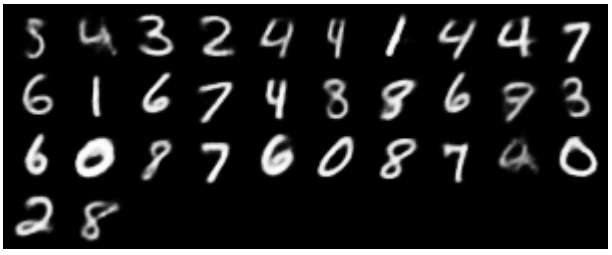

图 4: 隐层向量为 *10* 维时的卷积层生成图像

尤其是隐层为 10 时，模型生成的很多图像，都很清晰，亮度很高。但是全连接层生成的图像大多都很模糊。下面这个是全连接生成的图像。

我们确实可以看到卷积层在处理二维图像时很有优势。

2.2. 其他模型优化思路

在上面的探究中我在七个方面来改进模型：加入归一化、使用更深的网络、更换不同的优化器、调整 batchsize 的大小、更改隐层的维度、使用 dropout、将 encoder 和 decoder 改为卷积网络。

还有一些其他的改进方法我在这里阐述一下思路，其中有一些因为实现难度和时间的原因我经过了一些尝试但没有成功，另一些我查阅了资

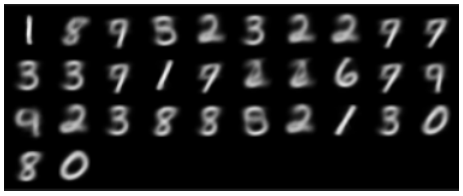

图 5: 隐层向量为 *1* 时的全连接层生成图像

料但没有实操。

1、对卷积层调参优化。卷积层具有对平移不变性的强大建模能力，这使得它们非常适合处理图像和其他类型的空间数据。相比之下，全连接层没有这种内置的平移不变性，并且通常需要更多的参数来对图像进行建模。

然而，在某些情况下，全连接层仍然可以表现得很好，卷积网络可能需要更多的深度和训练时间才能正确捕捉到图像中的结构信息。因此可以对卷积网络继续调参以获得更好的效果。同时也可能需要尝试调整模型架构、损失函数或训练策略，以及超参数以获得更好的结果。

2、数据增强。MNIST 数据集比较简单，可以考虑使用数据增强的方法。常用的数据增强技术包括随机裁剪、水平或垂直翻转、缩放、旋转等。这些方法可以帮助增加数据集的多样性，增加模型泛化能力，从而提高模型的训练效果。

但我其实有一点疑惑就是，手写数字，如果对其进行翻转的数据增强，其实就不是一个正确的数字了，而缩放会导致输入维度变化，如果使用 padding 的话会导致一定程度的失真。所以我并没有想到实现简单且效果更好的数据增强实现方法。

3、注意到，vae 的 loss 函数由两个部分组成，一个是原本的重建误差，一个是我们为了使生成的 sigma 向量更加有效而引入的散度误差，在基础的 vae 版本中我们将其直接相加。

但是实际上我在训练中感觉到，其实二者对训练的影响是不一样的，也就是说我们应该将 loss 写成

$$loss = \lambda_1 loss_1 + \lambda_2 loss_2$$

我们对两个 loss 加权得到最终的 loss，对其中的权重，我们可以认为设置。但是这样需要在一个很大的区间里暴力搜索，耗时耗力。

因此有一个比较直观的想法就是让机器自己去学习权重，有一种方法叫 gradnorm，可以实现自己学习权重的实现。

我认为这也是一种改进的方法，会对模型会有一些提升和优化。

这里有一个简短的对这个方法的讲解: https://zhuanlan.zhihu.com/p/542296680。

2.3. 隐层为 1

我将隐层设置为 1，使用前面优化好的全连接层，使用 linspace 函数在-20 到 20 之间生成 400 项的等差数列，将每一项作为 code 输入模型中，将生成的图片组合在一起，得到了以下的结果。

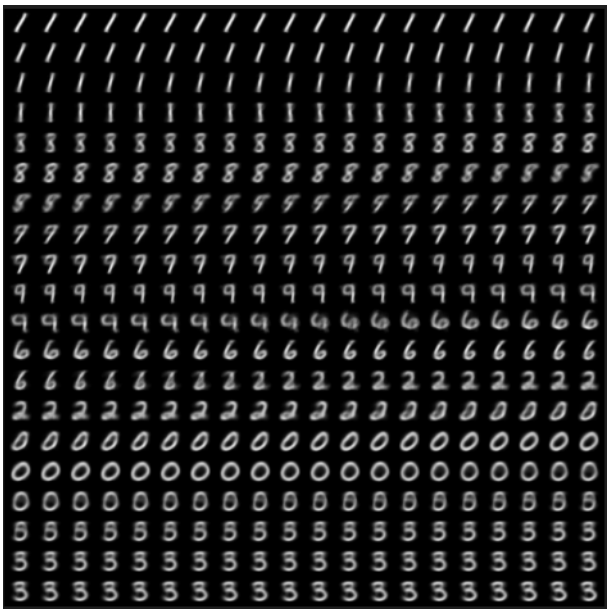

图 6: 隐层向量为 *1* 维生成的图像

根据 vae 的原理，它会将不同的 z 的取值空间映射到图像的空间。也就说我的模型将-20 的 z 值映射到了 1 的图像空间附近，将大概-10 左右的 z 值映射到了 8 的图像空间附近，其他的以此类推。

在之前我们说到了，vae 可以创建连续的 code space，这一点很有趣，我们可以看到，当我们输入从 1 的图像空间对应的 z 值和 8 的图像空间对应的 z 值中间的这些值时，我们可以看到模型生成了既像 1 又像 8 的这样的图像，我们观察图中的

这一系列图像我们甚至可以看到图像 1 变化到 8 的中间的变化过程，1 会逐渐扭曲产生 8 的形状，之后慢慢变成 8 的样子。这非常有趣，因为这些东西是我们数据中没有的，但是因为 code space 的连续性，我们生成了这样的新的图片。

2.4. 隐层为 2

和上面的过程类似，我对隐层向量的两个维度在-5 到 5 之间均匀采样，生成了 400 个 code，将 code 输入 vae 模型中生成图片，并组合在在一起，得到了这样的结果。

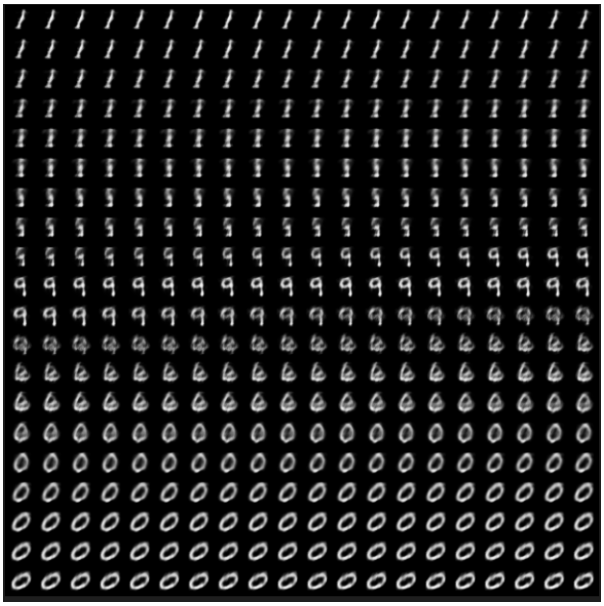

图 7: 隐层向量为 2 维生成的图像

可以看到，在左上到右下的过程中，也就是从 (-5,-5) 到 (5,5) 的过程中，生成的图像从 1 变成了 0。如果我们固定 x 看 y，固定 y 看 x，我们可以观察到连续空间中 1 变化到 0 的过程，我的模型中，1 会先把上部分扭曲成圈变成 9 的样子，然后会下部分扭曲变成 6 的样子，最后两个扭曲结合在一起变成了 0。

我们的任务是 MNIST，这里面的图像都是数字，但是我们还是生成了一些我们没见过的数字，这也是连续 code space 导致的。这些没见过的数字对我们来说没有什么意义。但是如果换成其他的任务，这些生成的新的东西可能成为一些有用的信息，这也是 vae 重要的特点之一。