content

content

VAE变分自编码器Keras实现

2020-07-23 22:44 847 0
编辑 收藏

变分自编码器（variational autoencoder，VAE）是一种生成模型，训练模型分为编码器和解码器两部分。

编码器将输入样本映射为某个低维分布，这个低维分布通常是不同维度之间相互独立的多元高斯分布，因此编码器的输出为这个高斯分布的均值与对数方差（因为方差总是大于0，为了将它映射到$(-\infty,\infty)$，所以加了对数）。在编码器的分布中抽样后，解码器做的事是将从这个低维抽样重新解码，生成与输入样本相似的数据。数据可以是图像、文字、音频等。

VAE模型的结构不难理解，关键在于它的**损失函数**的定义。我们要让解码器的输出与编码器的输入尽量相似，这个损失可以由这二者之间的二[illegible]y）来[illegible]的目标

昵称： 顾周
园龄： 396.000000011天
粉丝： 13
关注： 3

积分 - 33448
排名 - 32385

QQ: 582476065
QQ群: 941030564
欢迎加入

随笔分类

- GAN(5)
- LaTeX(2)
- Python(7)
- TF/Keras/Pytorch(8)
- 概率论(11)
- 高等数学(4)
- 高性能集群(2)
- 机器学习(16)
- 矩阵运算(4)
- 李宏毅视频笔记(1)
- 论文笔记(4)
- 爬虫(1)
- 深度学习(17)
- 统计学习方法(9)
- 凸优化(1)
- 线性代数(6)
- 信息论(3)
- 英语(4)
- 智能算法(1)

阅读排行榜

1. 特征值之积等于矩阵行列式、特征值之和等于矩阵的迹(2518)

函数是不够的。在这样的目标函数下，不断的梯度下降，会使编码器在不同输入下的输出均值之间的差别越来越大，输出方差则会不断地趋向于0，也就是对数方差趋向于负无穷。因为只有这样才会使从生成分布获取的抽样更加明确，从而让解码器能生成与输入数据更接近的数据，以使损失变得更小。但是这就与生成器的初衷有悖了，生成器的初衷实际上是为了生成更多“全新”的数据，而不是为了生成与输入数据“更像”的数据。所以，我们还要再给目标函数加上编码器生成分布的“正则化损失”：生成分布与标准正态分布之间的KL散度（相对熵）。让生成分布不至于“太极端、太确定”，从而让不同输入数据的生成分布之间有交叉。于是解码器通过这些交叉的“缓冲带”上的抽样，能够生成“中间数据”，产生意想不到的效果。

详细的分析请看：**变分自编码器VAE：原来是这么一回事 - 知乎**

以下使用Keras实现VAE生成图像，数据集是MNIST。

1 代码实现

1.1 编码器

为2维的正态分布均值与对数方差。

简单堆叠卷积层与全连接层即可，代码如下：


```
#%%编码器
import numpy as np
import keras
from keras import layers,Mode
from keras import backend as
from keras.datasets import mn

img_shape = (28,28,1)
latent_dim = 2

input_img = layers.Input(shap
x = layers.Conv2D(32,3,paddin
x = layers.Conv2D(64,3,paddin
x = layers.Conv2D(64,3,paddin
x = layers.Conv2D(64,3,paddin
inter_shape = K.int_shape(x)
x = layers.Flatten()(x)
x = layers.Dense(32,activatio

encode_mean = layers.Dense(2,
encode_log_var = layers.Dense

encoder = Model(input_img,[en
```

1.2 解码器

解码器接受2维向量，将这个向量“解码”为图像。同样也是简单的堆叠卷积层、逆卷积层与全连接层即可，代码如下：

```
#%%解码器
input_code = layers.Input(sha
x = layers.Dense(np.prod(inte
x = layers.Reshape(target_sha
```

```
decoder = Model(input_code,x,
```

content

整体待训练模型

整个待训练模型包括编码器、抽样、解码器。中间的抽样操作在获取编码器传出的均值与方差后，通过一个自定义的lambda层来实现。这个抽样是先从标准正态分布中抽样，再通过乘生成分布的标准差，加上均值来获得。因此这个操作并不会把反向传播中断，可以将编码器与解码器的张量流连接起来。

定义好模型后是损失的定义，如前面所说，最终损失（目标函数）是生成图像与原图像之间的二元交叉熵和生成分布的正则化的平均值。使用add_loss方法来添加模型的损失，具体的自定义损失方法看**链接**。

代码如下：

```
#%%整体待训练模型
def sampling(arg):
  mean = arg[0]
  logvar = arg[1]
  epsilon = K.random_normal(s
  return mean + K.exp(0.5*log
input_img = layers.Input(shap
code_mean, code_log_var = enc
x = layers.Lambda(sampling,na
x = decoder(x)
training_model = Model(input_
```

```
training_model.compile(optimi
```


因为损失函数并没有定义真实数据与预测数据直接的损失，因此fit方法不需传入输入即可（不用输出）。

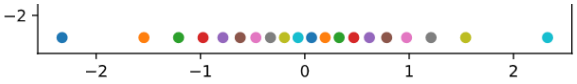
代码如下：

```
#%%读取数据集训练
(x_train,y_train),(x_test,y_t
x_train = x_train.astype('flo
x_train = x_train[:,:,:,np.ne

training_model.fit(
    x_train,
    batch_size=512,
    epochs=100,
    validation_data=(x_train[:2
```

1.5 生成测试

使用scipy.stats中的**norm.ppf方法**在概率区间(0.01,0.99)内生成20*20个解码器输入，这个方法类似在标准正态分布中抽样，但并不是随机的，是正态分布下的等概率。生成的二维点分布如下图：



content

习抽样为了和编码器的生成分布契合，因为编码器正态后生成的分布是靠近标准正态分布的。然后用解码器生成图片，这一部分的代码如下：

```
#%%测试
from scipy.stats import norm
import numpy as np
import matplotlib.pyplot as p
n = 20
x = y = norm.ppf(np.linspace(
X,Y = np.meshgrid(x,y)
X = X.reshape([-1,1])
Y = Y.reshape([-1,1])
input_points = np.concatenate
for i in input_points:
  plt.scatter(i[0],i[1])
plt.show()

img_size = 28
predict_img = decoder.predict
pic = np.empty([img_size*n,im
for i in range(n):
  for j in range(n):
    pic[img_size*i:img_size*(
plt.figure(figsize=(10,10))
plt.axis('off')
pic = np.squeeze(pic)
plt.imshow(pic,cmap='bone')
plt.show()
```

生成的400张图：

content

可以看出来，二维坐标系中**某个**
本文的编码是可以使解码器的输出从
一个数字变换到另一个数字的。

分类: 深度学习, TF/Keras/Pytorch

« 上一篇： Tensorflow/Keras、Pytorch 杂记
» 下一篇： 遗传算法详解与实验