原 变分自动编码机(VAE)理解和实现(Tensorflow)

2018年05月04日 10:21:58 ppp8300885 阅读数：3855 标签： 变分自动编码机 VAE 变分下界 mnist 生成模型 更多
个人分类： 计算机视觉 深度学习 机器学习

你需要知道的：

1. 自动编码机Auto-Encoder (AE)由两部分encoder和decoder组成，encoder输入x数据，输出潜在变量z，decoder输入z然后输出一个x’，目的是让x’与布尽量一致，当两者完全一样时，中间的潜在变量z可以看作是x的一种压缩状态，包含了x的全部feature特征，此时监督信号就是原数据x本身。
2. 变分自动编码机VAE是自动编码机的一种扩展，它假设输出的潜在变量z服从一种先验分布，如高斯分布。这样，在训练完模型后，我们可以通过采样这种布得到z’，这个z’可能是训练过程中没有出现过的，但是我们依然能在解码器中通过这个z’获得x’，从而得到一些符合原数据x分布的新样本，具有“生成“新样本的能力。
3. VAE是一种生成模型，它的目标是要得到$p(z|x)$分布，即给定输入数据x的分布，得到潜在变量x的分布，与其他的生成模型一样，它计算的是x和z的联合概率分布$p(x,z)$（如朴素贝叶斯模型通过计算$p(x,z)/p(x)$得到$p(z|x)$），当然它不是直接计算这个联合概率分布，而是借助一些公式变换求解。

从简单的例子理解VAE/AE的意义：

前面讲过，变分自动编码机的目的是想知道观测数据x背后的潜在变量z分布，即$p(z|x)$，举个简单的例子，比如天气是我们的观测数据x，但我们想知道影响天气变化背后的一些无法观测的因素z，这个z就像自然法则一样能够左右最后观测到的天气，这样我们以后描述某个天气，就可以完全量化为对应的潜在变量z。对于这个例子，VAE/AE都能完成这个事情，但如果现在我们想生成一些新的天气样本来作为研究，这个时候只有VAE可以很容易做这个事情：拟合现有样本分布的一个潜在变量的先验分布，通过采样这个先验分布来获得新的样本；而对于AE这个事情就比较难了：由于每个样本x被固定编码为对应的z，我们无法知道潜在样本的分布（若此时我们知道了z的分布，就等于知道了真实数据x的分布，这显然是不可能的，相比VAE的解决方案是把真实数据x对应的潜在分布映射到一个先验分布上），若AE硬要获得新样本怎么做呢，此时只能随机采样z了，很显然我们无法验证：根据这个z是否能正确地还原出一个符合真实样本x的新样本。

除了单纯“生成“新的样本用途，生成模型还可以用来去噪声，比如现在的图片里有雾霾，我们想把图片里的雾霾去掉，还原没有雾霾的样子，就可以用VAE/AE做：把有雾霾的图片当作输入x，对应的无雾霾的图片（假设我们能够在天气好的时候获得）作为最后要还原的x’训练VAE模型，如果训练的足够好的话，以后再任意拿一张有雾霾的图片，VAE能够还原出这个图片没有雾霾的样子，这就是生成模型的优势。**当然，判别模型也能做这个事情：在给定原图像的情况下，尽量拟合原图像的变换图像**，但是若测试时出现了之前训练过程中没有出现的图像，效果会不好，因为判别模型是基于条件概率$p(x'|x)$，若新的条件x模型都没见过，效果肯定不好啊，所以判别模型更注重泛化能力。而生成模型会去拟合x和x’联合概率分布$p(x,x')$，因此$p(x'|x)$的计算只需要除以边缘概率分布$p(x)$即可，而对于VAE来说，它拟合的其实是x和潜在变量z的联合概率分布$p(x,z)$，获得$p(z|x)$从而间接生成x’

VAE推导

为了求解真实的后验$p(z|x)$概率分布，VAE引入一个识别模型$q(z|x)$去近似$p(z|x)$，那么衡量这两个分布之间的差异自然就是相对熵了，也就是KL散度，VAE的目的就是要让这个相对熵越小，因此推导从相对熵开始：

$$
\begin{aligned}
KL(q(z|x)||p(z|x)) &= \int q(z|x) \log \frac{q(z|x)}{p(z|x)} dz \\
&= \int q(z|x) \left(\log q(z|x) - \log \frac{p(z,x)}{p(x)} \right) dz \\
&= \int q(z|x) \left(\log q(z|x) - \log p(z,x) + \log p(x) \right) dz \\
&= \int q(z|x) \left(\log q(z|x) - \log p(z,x) \right) dz + \log p(x) \\
&= E_{z\sim q(z|x)} \log \frac{q(z|x)}{p(z,x)} + \log p(x)
\end{aligned}
$$

我们把两个分布的KL散度展开后得到了两项，第一项是一个期望，第二个是真实样本概率的对数$\log p(x)$，虽然我们不知道它的值是多少，但是我们知道它的值是一个定值。我们将上述结果稍微调换位置得到如下：

$$
L(x) = E_{z\sim q(z|x)} \log \frac{p(z,x)}{q(z|x)} = \log p(x) - KL(q(z|x)||p(z|x))
$$

令$L(x)$为上述期望, 它等于一个固定值减去KL散度，由于KL散度值是恒大于0的（当两个分布完全一致时，KL散度为0），因此有$L(x) \leqslant \log p(x)$，此时$L(x)$作是真实概率log值的一个下界，原文叫做变分下界(variational lower bound)。我们目的当然是最优化这个下界，当下界越靠近$\log p(x)$时，KL散度越小，此时我们$q(z|x)$就能够越准确地估计$p(z|x)$。

现在我们继续研究这个下界L，发现里面有个联合概率分布$p(z,x)$，这个东西可不好求，因此继续把它用贝叶斯公式展开，然后合并成如下样子：

$$
\begin{aligned}
L(x) &= E_{z\sim q(z|x)} \log \frac{p(z,x)}{q(z|x)} \\
&= E_{z\sim q(z|x)} \log \frac{p(x|z)p(z)}{q(z|x)} \\
&= \int q(z|x)(\log p(z) - \log q(z|x) + \log p(x|z))dz \\
&= -\int q(z|x)\left(\log \frac{q(z|x)}{p(z)}\right)dz + \int q(z|x)\log p(x|z))dz \\
&= -KL(q(z|x)||p(z)) + E_{z\sim q(z|x)}(\log p(x|z))
\end{aligned}
$$

经过变换，我们把这个变分下界$L(x)$用一个期望和KL散度的差表示，我们先看这个期望怎么求，这个期望表示的是在已获得的z变量的情况下输出x的log似然期 也可以看作是解码器的损失函数，因为我们希望解码器能通过z尽量的还原出x，也就是尽量使这个对数似然在z服从$q(z|x)$分布情况下最大，那么这个期望怎么求 最简单的就是蒙特卡洛采样了：对于样本x，用$q(z|x)$分布采样出L个z，对于每个z算出$p(x|z)$概率的log值，然后取平均即为所求期望，而且当采样次数L越大， 值越接近于真实的期望值：

$$E_{z\sim q(z|x)}(\log p(x|z)) \approx \frac{1}{L}\sum_{l=1}^{L} \log p(x|z_l), z_l \sim q(z_l|x)$$

但是这种简单的蒙特卡洛采样的缺点是估计出来的值方差太大(high variance)，也就是说采样出的z与z之间相差比较大，导致最后估计值波动性太大，而且这种直接采样的方法通常是不可求导的，所以不实用。因此，VAE把对z的采样分成两部分来求：一部分是固定的值比如标准差σ和均值μ，另一部分是一个随机的高斯噪声ϵ；具体来说，用一个函数$g(x,\epsilon)$表示最后采样出的z值，这个函数由两部分的和组成：$g(x,\epsilon) = \mu_x + \sigma_x \odot \epsilon$，其中$\epsilon \sim N(0,1)$，$\mu_x$和$\sigma_x$是两个关于$x$的向量，一般可以理解为网络在输入x样本后输出的两个向量，$\odot$表示点乘；这样，z的采样由于被固定的μ_x和σ_x值决定着其均值和方差，而随机的部分只由高斯分布决定，因此减小了方差，而且这种情况下，我们还能计算μ_x和σ_x的梯度用于更新，这种trick叫做重参数化(reparameterization trick)，当然上述只是$g(x,\epsilon)$的一种形式，论文给出了构造$g(x,\epsilon)$的一般约束。

其实上述的期望换一种角度理解，本质上描述了解码器的性能，z相当于是从编码器获得的潜在变量，而解码器要做的就是尽量让z能还原出原来的x，也就是尽可能让$\log p(x|z)$最大化，因此它的损失函数就是$ p(x|z)$与真实x分布的交叉熵。

那么我们回到变分下界$L(x)$，我们已经知道了如何最大化式子中第二项的期望，那么如何最小化第一项呢？我们知道KL散度是恒大于0的，因此我们只需要最小化KL散度即可，此时变分下界最大。由于KL散度描述着两个分布之间的差距，VAE因此让$p(z)$服从一个先验的高斯分布$N(0,1)$，便直接可以展开式子计算$q(z|x)$与$p(z)$的KL散度，这是因为$q(z|x)$其实就是一种高斯均值为μ_x，方差为σ_x的高斯分布（由上述$g(x,\epsilon)$的求法可得），衡量两个高斯分布的差异可以通过它们的密度函数展开推导出来，有兴趣的可以尝试推一下：

$$-KL(q(z|x)||p(z)) = \frac{1}{2}\sum_{j=1}^{J}(1+\log((\sigma_j)^2) - (\mu_j)^2 - (\sigma_j)^2)$$

这里σ_j和μ_j分别表示向量σ_x和μ_x的第j个值，这个KL散度本质上描述了编码器的损失：VAE强制让输出的z变量服从先验的高斯分布$N(0,1)$，因此损失函数即为当前输出的z分布与标准高斯分布之间的距离，也就是这个KL散度。

最后$L(x)$被写成：

$$
\begin{aligned}
L(x) &= -KL(q(z|x)||p(z)) + E_{z\sim q(z|x)}(\log p(x|z)) \\
&= \frac{1}{2}\sum_{j=1}^{J}(1+\log((\sigma_j)^2) - (\mu_j)^2 - (\sigma_j)^2) + \frac{1}{L}\sum_{l=1}^{L}\log p(x|z_l)
\end{aligned}
$$

总结：最优化$L(x)$变分下界意味着让编码器输出的z值符合先验的高斯分布的情况下，同时也让解码器能够最大可能的用z还原出原来的x，这就是VAE的整个流程，有非常漂亮的理论依据。

VAE的实现(Tensorflow)

这里主要写一下实现中比较重要的部分，源码请参考这个github，使用的mnist手写体识别的数据集，输入的是一张张手写图片，输出的是经过潜在变量z还原后的图片。

编码器：

```
def gaussian_MLP_encoder(...):
    # 1st hidden layer
    ...

    # 2nd hidden layer
    ...
```

```

    # output layer
    wo = tf.get_variable('wo', [h1.get_shape()[1], n_output * 2], initializer=w_init)
    bo = tf.get_variable('bo', [n_output * 2], initializer=b_init)
    gaussian_params = tf.matmul(h1, wo) + bo

    # The mean parameter is unconstrained
    mean = gaussian_params[:, :n_output]
    # The standard deviation must be positive. Parametrize with a softplus and
    # add a small epsilon for numerical stability
    stddev = 1e-6 + tf.nn.softplus(gaussian_params[:, n_output:])

```


编码器的输出分两部分，一部分表示mean，一部分表示标准差std，其中由于标准差是恒大于0，因此用了softplus激活函数：

解码器：

```
def bernoulli_MLP_decoder(...):
    # 1st hidden layer
    ...

    # 2nd hidden layer
    w1 = tf.get_variable('w1', [h0.get_shape()[1], n_hidden], initializer=w_init)
    b1 = tf.get_variable('b1', [n_hidden], initializer=b_init)
    h1 = tf.matmul(h0, w1) + b1
    h1 = tf.nn.elu(h1)
    h1 = tf.nn.dropout(h1, keep_prob)

    # output layer-mean
    wo = tf.get_variable('wo', [h1.get_shape()[1], n_output], initializer=w_init)
    bo = tf.get_variable('bo', [n_output], initializer=b_init)
    y = tf.sigmoid(tf.matmul(h1, wo) + bo)
```

输出的大小与输入一致，其中每个元素代表着此位置的像素值为0的概率(或者255，根据输入来定)，所以用sigmoid激活函数

损失函数

```
# 编码器得到标准差和均值向量
mu, sigma = gaussian_MLP_encoder(x_hat, n_hidden, dim_z, keep_prob)

# reparameterization 重参数采样得到z
z = mu + sigma * tf.random_normal(tf.shape(mu), 0, 1, dtype=tf.float32)

# 解码器传入z，输出y
y = bernoulli_MLP_decoder(z, n_hidden, dim_img, keep_prob)
y = tf.clip_by_value(y, 1e-8, 1 - 1e-8)

# marginal_likelihood loss为y与输入数据x之间交叉熵，即解码器的损失
marginal_likelihood = tf.reduce_sum(x * tf.log(y) + (1 - x) * tf.log(1 - y), 1)
marginal_likelihood = tf.reduce_mean(marginal_likelihood)

# KL_divergence为z与标准高斯分布之间的差距，即编码器的损失
KL_divergence = 0.5 * tf.reduce_sum(tf.square(mu) + tf.square(sigma) - tf.log(1e-8 + tf.square(sigma)) - 1, 1)
KL_divergence = tf.reduce_mean(KL_divergence)

# 变分下界L(x)，目标最大化
ELBO = marginal_likelihood - KL_divergence

# 令损失函数为-L(x)，目标梯度下降最小化
loss = -ELBO
```

训练过程

```
# 定义更新器，最小化loss
train_op = tf.train.AdamOptimizer(learn_rate).minimize(loss)

with tf.Session() as sess:
```

```
    sess.run(tf.global_variables_initializer(), feed_dict={keep_prob : 0.9})

    for epoch in range(n_epochs):

        # Random shuffling
        np.random.shuffle(train_total_data)
        train_data_ = train_total_data[:, :-mnist_data.NUM_LABELS]

        # Loop over all batches
        for i in range(total_batch):
            # Compute the offset of the current minibatch in the data.
            offset = (i * batch_size) % (n_samples)
            batch_xs_input = train_data_[offset:(offset + batch_size), :]

            # 输出label等于输入值
            batch_xs_target = batch_xs_input

            # 可以在输入中加入噪音，让VAE从带有噪音的x还原真实的x
            if ADD_NOISE:
                batch_xs_input = batch_xs_input * np.random.randint(2, size=batch_xs_input.shape)
                batch_xs_input += np.random.randint(2, size=batch_xs_input.shape)

            # forward + backword 过程，记录总的loss，编码器和解码器loss
            _, tot_loss, loss_likelihood, loss_divergence = sess.run(
                (train_op, loss, neg_marginal_likelihood, KL_divergence),
                feed_dict={x_hat: batch_xs_input, x: batch_xs_target, keep_prob : 0.9})
```


结果

输入数据：

输出（第0个epoch）：