Deep learning

7.4. Variational Autoencoder

François Fleuret

https://fleuret.org/dlc/

Coming back to generating a signal, instead of training an autoencoder and modeling the distribution of Z, we can try an alternative approach:

Impose a distribution for Z and then train a decoder g so that $g(Z)$ matches the training data.

This can be done with a **Variational Autoencoder** (Kingma and Welling, 2013).

Notes

We saw in lecture 7.2. "Deep Autoencoders" that autoencoders can, to some extent, model the data distribution by first mapping the data to a smaller dimension latent space, and then fitting a density model.
As seen with the experiments on MNIST, this is not satisfying because,

- either the latent space is of very small dimension, in which case it makes sense to use a parameterized density model over the latent representation, but the input signal is not modeled properly, or
- the latent space is of higher dimension, in which case it becomes harder to properly model the latent representation.

We want to train a model $p(X = x \mid Z = z; w)$ such that, with $p(Z = z)$ fixed, for instance to $\mathcal{N}(0, I)$, the marginal

$$p(X = x; w) = \int p(X = x \mid Z = z; w) p(Z = z) dz$$

match the training data, hence maximizes

$$\sum_n \log p(X = x_n; w).$$

This value is sometimes referred to as the (log of the) **model evidence**.

The model for $p(X = x \mid Z = z)$ plays the role of a decoder: Given the latent representation z, it estimates the signal x.

A form that echoes Gaussian mixture models, is to take

$$p(X \mid Z = z; w) = \mathcal{N}(\mu^g(z; w), \text{diag}(\sigma^g(z; w))).$$

where μ^g and σ^g are of same shape as X and are computed by a deep model g.

The key technical issue is that there is no tractable form for the marginalized quantity $p(X = x; w)$.

What we can do is to estimate it by sampling. Indeed, with any distribution $q(Z)$, we have

$$\begin{aligned} p(X = x) &= \int p(X = x, Z = z; w) dz \\ &= \int \frac{p(X = x, Z = z; w)}{q(Z = z)} q(Z = z) dz \\ &= \mathbb{E}_{z \sim q(Z)} \left[\frac{p(X = x, Z = z; w)}{q(Z = z)} \right]. \end{aligned}$$

Hence, if we sample one $z \sim q(Z)$, the quantity

$$\frac{p(X = x, Z = z; w)}{q(Z = z)}$$

is an unbiased estimator of $p(X = x; w)$.

However we want to maximize the fit to the training set, which corresponds to maximizing the likelihood of the training data

$$\sum_n \log p(X = x_n).$$

Due to its convexity the log of our unbiased estimator of $p(X = x; w)$ is not an unbiased estimator of $\log p(X = x; w)$.

We can look at that more precisely:

$$\begin{aligned}
&\mathbb{E}_{z\sim q(Z)}\left[\log \frac{p(X=x,Z=z;w)}{q(Z=z)}\right] \\
&= \mathbb{E}_{z\sim q(Z)}\left[\log \frac{p(Z=z\mid X=x;w)p(X=x;w)}{q(Z=z)}\right] \\
&= \log p(X=x;w) + \mathbb{E}_{z\sim q(Z)}\left[\log \frac{p(Z=z\mid X=x;w)}{q(Z=z)}\right] \\
&= \log p(X=x;w) - \mathbb{D}_{\mathrm{KL}}\left(q(Z)\,\|\,p(Z\mid X=x;w)\right).
\end{aligned}$$

Where

$$\mathbb{D}_{\mathrm{KL}}(a\,\|\,b) = \int a(u)\log\frac{a(u)}{b(u)}du = -\int a(u)\log\frac{b(u)}{a(u)}du$$

is the **Kullback-Leibler divergence.**

This quantity is non-negative, hence the expectation of the log of our estimator is a lower bound of $\log p(X=x;w)$, called the **Evidence Lower Bound** (ELBO).

Hence, to have the model fit the data when we optimize the ELBO, we need a $q(Z)$ that makes $\mathbb{D}_{\mathrm{KL}}\left(q(Z) \,\|\, p(Z \mid X = x; w)\right)$ as small as possible.

All the derivations remain valid if q is a function of X. The quantity we want to maximize is then

$$\log p(X = x; w) - \mathbb{D}_{\mathrm{KL}}\left(q(Z \mid X = x; w') \,\|\, p(Z \mid X = x; w)\right)$$

and maximizing it will both maximize $\log p(X = x; w)$, and minimize the KL term, hence will bring $q(Z \mid X = x; w')$ close to $p(Z \mid X = x; w)$.

The role of $q(Z \mid X = x; w')$ is very similar to that of an encoder: Given the signal x, it estimates what z are consistent with the decoding.

We can again use a Gaussian whose parameters are computed by a deep model f

$$q(Z \mid X = x; w') \sim \mathcal{N}(\mu^f(x; w'), \mathrm{diag}(\sigma^f(x; w'))).$$

One last technical point is that we can rewrite the ELBO as

$$\begin{aligned}
&\mathbb{E}_{z\sim q(Z|X=x;w')}\left[\log \frac{p(X=x,Z=z;w)}{q(Z=z\mid X=x;w')}\right] \\
&= \mathbb{E}_{z\sim q(Z|X=x;w')}\left[\log \frac{p(X=x\mid Z=z;w)p(Z=z)}{q(Z=z\mid X=x;w')}\right] \\
&= \mathbb{E}_{z\sim q(Z|X=x;w')}\left[\log p(X=x\mid Z=z;w) - \log \frac{q(Z=z\mid X=x;w')}{p(Z=z)}\right] \\
&= \mathbb{E}_{z\sim q(Z|X=x;w')}\Big[\log p(X=x\mid Z=z;w)\Big] - \mathbb{D}_{\mathrm{KL}}\left(q(Z\mid X=x;w')\,\|\,p(Z)\right).
\end{aligned}$$

This form allows to take advantage of the closed-form expression of the KL divergence between Gaussians to get a less noisy estimate:

$$\begin{aligned}
&\mathbb{D}_{\mathrm{KL}}\left(\mathcal{N}(\mu_1,\Sigma_1),\mathcal{N}(\mu_2,\Sigma_2)\right) \\
&\qquad = \frac{1}{2}\left[\log\frac{|\Sigma_1|}{|\Sigma_2|} - D + (\mu_1-\mu_2)^\top \Sigma_2^{-1}(\mu_1-\mu_2) + \mathrm{Tr}\left(\Sigma_2^{-1}\Sigma_1\right)\right].
\end{aligned}$$

So the final loss is

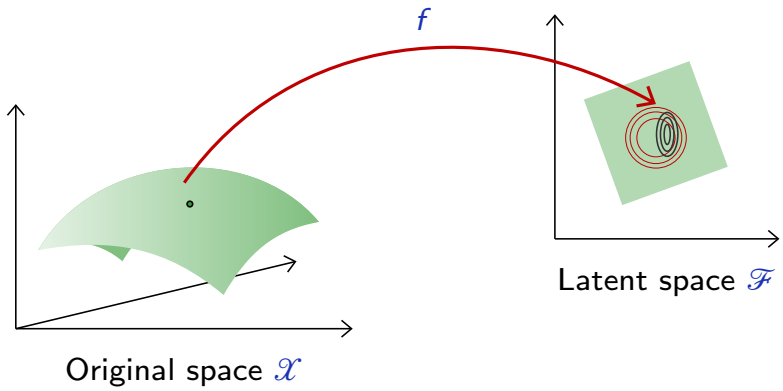
$$\mathscr{L}(w, w') = \frac{1}{N} \sum_n \mathbb{D}_{\mathrm{KL}} \left(q(Z \mid X = x_n; w') \,\|\, p(Z) \right) - \log p(X = x_n \mid Z = z_n; w)$$

where $\forall n,\ z_n \sim q(Z \mid X = x_n; w')$.

Minimizing the first term

$$\mathbb{D}_{\mathrm{KL}}\left(q(Z \mid X = x_n; w') \,\|\, p(Z)\right)$$

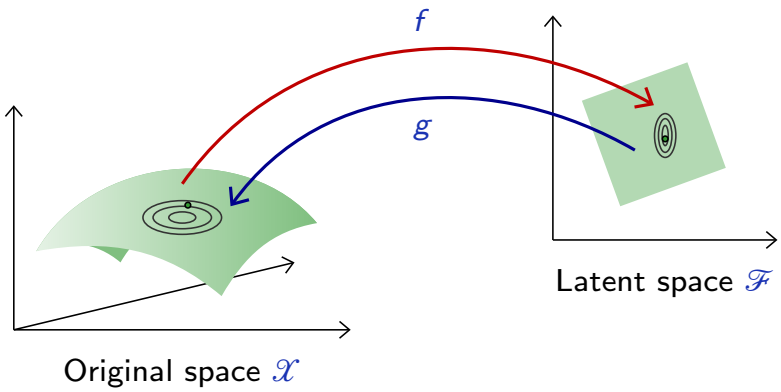
brings $q(Z \mid X = x_n; w')$ close to $p(Z) = \mathcal{N}(0, I)$.



Minimizing the second term for a $z_n \sim q(Z \mid X = x_n; w')$

$$-\log p(X = x_n \mid Z = z_n; w)$$

maximizes the likelihood of the original data point x_n under $p(X \mid Z = z_n; w)$.



The assumption of independence between the component of $P(X \mid Z = z)$ allows the model to overfit the variance and additionally leads to grainy samples.

We fix this by forcing a variance of 1 during training and 0 during sampling.

```
class VariationalAutoEncoder(nn.Module):
    def __init__(self, nb_channels, latent_dim):
        super().__init__()

        self.encoder = nn.Sequential(
            nn.Conv2d(1, nb_channels, kernel_size=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(nb_channels, nb_channels, kernel_size=5),
            nn.ReLU(inplace=True),
            nn.Conv2d(nb_channels, nb_channels, kernel_size=5),
            nn.ReLU(inplace=True),
            nn.Conv2d(nb_channels, nb_channels, kernel_size=4, stride=2),
            nn.ReLU(inplace=True),
            nn.Conv2d(nb_channels, nb_channels, kernel_size=3, stride=2),
            nn.ReLU(inplace=True),
            nn.Conv2d(nb_channels, 2 * latent_dim, kernel_size=4),
        )

        self.decoder = nn.Sequential(
            nn.ConvTranspose2d(latent_dim, nb_channels, kernel_size=4),
            nn.ReLU(inplace=True),
            nn.ConvTranspose2d(nb_channels, nb_channels, kernel_size=3, stride=2),
            nn.ReLU(inplace=True),
            nn.ConvTranspose2d(nb_channels, nb_channels, kernel_size=4, stride=2),
            nn.ReLU(inplace=True),
            nn.ConvTranspose2d(nb_channels, nb_channels, kernel_size=5),
            nn.ReLU(inplace=True),
            nn.ConvTranspose2d(nb_channels, 1, kernel_size=5),
        )
```

```
def encode(self, x):
    output = self.encoder(x).view(x.size(0), 2, -1)
    mu, log_var = output[:, 0], output[:, 1]
    return mu, log_var

def decode(self, z):
    mu = self.decoder(z.view(z.size(0), -1, 1, 1))
    return mu, mu.new_zeros(mu.size())
```

```
def sample_gaussian(param):
    mean, log_var = param
    std = log_var.mul(0.5).exp()
    return torch.randn(mean.size(), device=mean.device) * std + mean


def log_p_gaussian(x, param):
    mean, log_var, x = param[0].flatten(1), param[1].flatten(1), x.flatten(1)
    var = log_var.exp()
    return -0.5 * (((x - mean).pow(2) / var) + log_var + math.log(2 * math.pi)).sum(1)


def dkl_gaussians(param_a, param_b):
    mean_a, log_var_a = param_a[0].flatten(1), param_a[1].flatten(1)
    mean_b, log_var_b = param_b[0].flatten(1), param_b[1].flatten(1)
    var_a = log_var_a.exp()
    var_b = log_var_b.exp()
    return 0.5 * (
        log_var_b - log_var_a - 1 + (mean_a - mean_b).pow(2) / var_b + var_a / var_b
    ).sum(1)
```

Note in particular the **re-parameterization trick**:

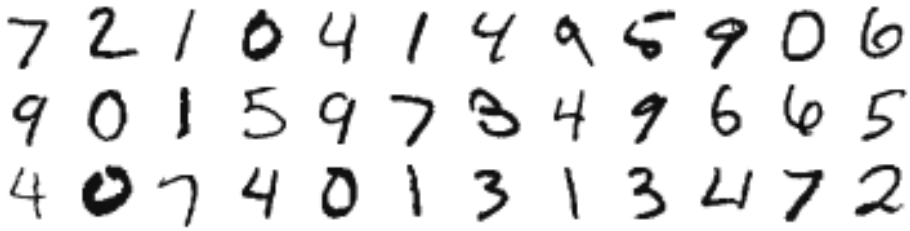
```
def sample_gaussian(param):
    mean, log_var = param
    std = log_var.mul(0.5).exp()
    return torch.randn(mean.size(), device=mean.device) * std + mean
```

Implementing the sampling of z that way allows to compute the gradient w.r.t the density's parameters without any particular property of `randn()`.

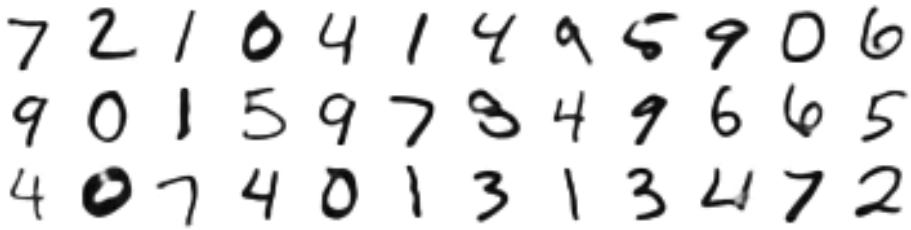
```
for x in train_input.split(args.batch_size):
    param_q_Z_given_x = model.encode(x)
    z = sample_gaussian(param_q_Z_given_x)
    param_p_X_given_z = model.decode(z)
    log_p_x_given_z = log_p_gaussian(x, param_p_X_given_z)

    dkl_q_Z_given_x_from_p_Z = dkl_gaussians(param_q_Z_given_x, param_p_Z)
    loss = -(log_p_x_given_z - dkl_q_Z_given_x_from_p_Z).mean()

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

```
parser.add_argument("--nb_epochs", type=int, default=25)

parser.add_argument("--learning_rate", type=float, default=1e-3)

parser.add_argument("--batch_size", type=int, default=100)

parser.add_argument("--latent_dim", type=int, default=32)

parser.add_argument("--nb_channels", type=int, default=32)
```

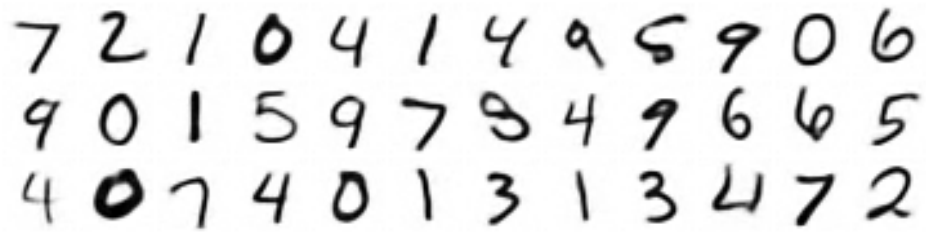
Original



Autoencoder reconstruction ($d = 32$)



Variational Autoencoder reconstruction ($d = 32$)

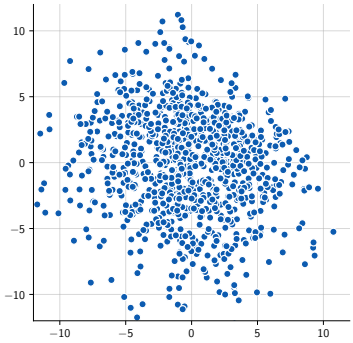


Notes

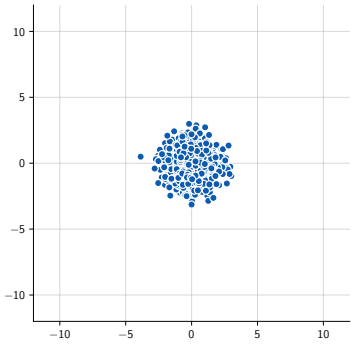
The images at the top are original test MNIST samples. The images in the middle are the reconstructed samples with the standard autoencoder as seen in lecture 7.2. "Deep Autoencoders". The images at the bottom are the reconstructed samples obtained with the variational autoencoder.
The results are not as good as with the standard autoencoder, which is not surprising since there is an additional constraint on the distribution in the latent space.

We can look at two latent features to check that they are Normal for the VAE.

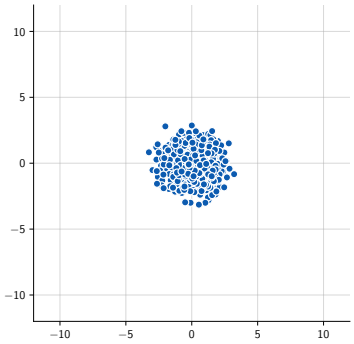
Autoencoder



Variational autoencoder



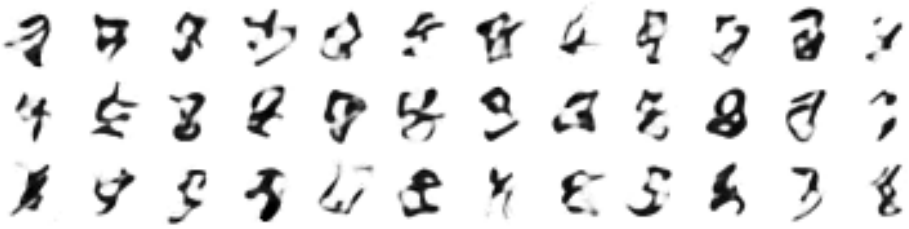
$\mathcal{N}(0,1)$



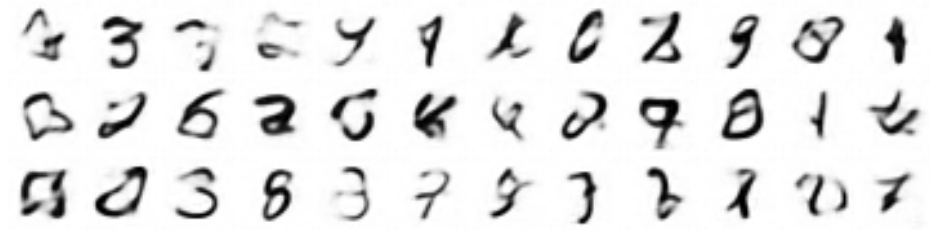
Notes

The first scatter plot on the top-left shows the empirical distribution of two latent dimensions when the encoder is from the standard autoencoder. The plot was generated by choosing at random two dimensions among the 32 ones of the latent space, encode 1000 MNIST samples, and draw a point at the two resulting coordinates. We can see that this is not a normal distribution. If the same process is done with the variational autoencoder we get the scatter plot at the bottom left, that we can compare with a similar scatter plot obtained by sampling independent normal coordinates, shown at the bottom right.
The variational autoencoder did its job of making the embedded representation follow a Gaussian distribution.

Autoencoder sampling ($d = 32$)



Variational Autoencoder sampling ($d = 32$)



Notes

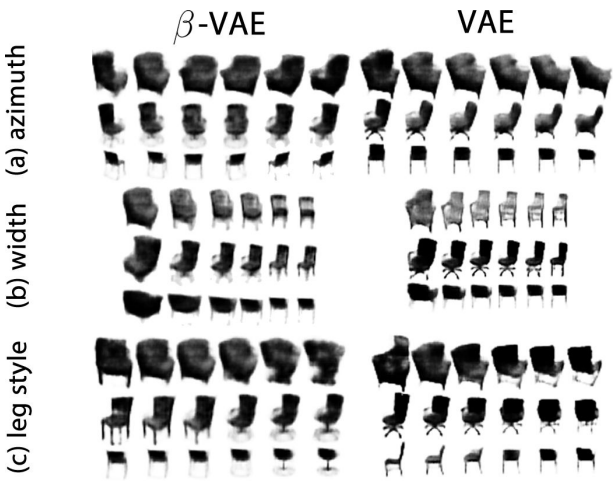
We can compare the result of sampling data points in the latent space and map them back in the original space with the decoder. To that purpose, we first generate a random Gaussian vector of dimension equal to that of the latent space, $d = 32$, and then we run the decoder on that random sample, which produces an image in the original space.
The images on the top are obtained with the decoder from the standard autoencoder, and the images on the bottom are for the variational autoencoder. Although these small-scale experiments are far from the state of the art, the latter samples are more realistic.

Making the embedding $\sim \mathcal{N}(0, 1)$, often results in "disentangled" representations.

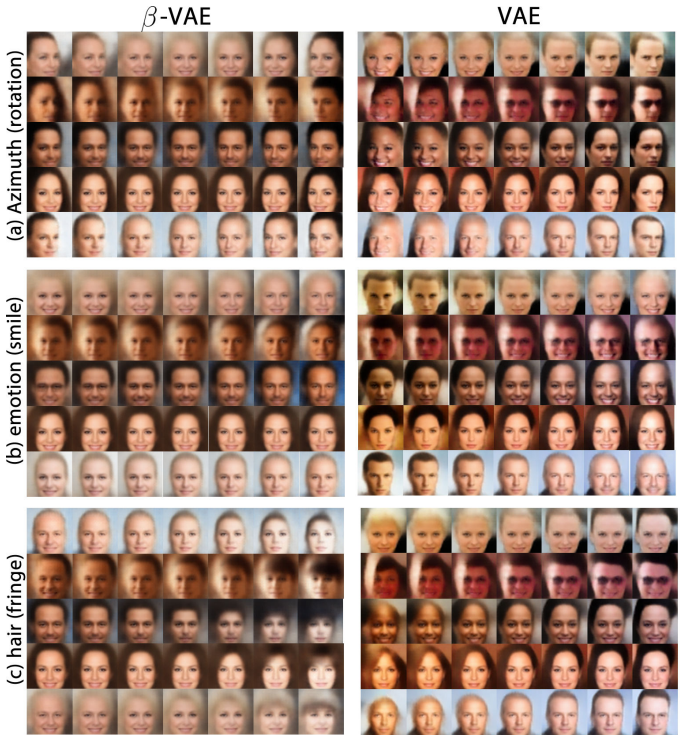
This effect can be reinforced with a greater weight of the KL term

$$\frac{1}{N} \sum_n \beta \, \mathbb{D}_{\mathrm{KL}} \left(q(Z \mid X = x_n; w') \,\|\, p(Z) \right) - \log p(X = x_n \mid Z = z_n; w)$$

resulting in the β-VAE proposed by Higgins et al. (2017).



(Higgins et al., 2017)



(Higgins et al., 2017)

References

I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. **beta-vae: Learning basic visual concepts with a constrained variational framework**. In International Conference on Learning Representations (ICLR), 2017.

D. P. Kingma and M. Welling. **Auto-encoding variational bayes**. CoRR, abs/1312.6114, 2013.