EE-559 – Deep learning

9.3. Denoising and variational autoencoders

François Fleuret

https://fleuret.org/ee559/

Tue Apr 16 10:20:30 UTC 2019

Denoising Autoencoders

Vincent et al. (2010) interpret training the autoencoder as maximizing the mutual information between the input and the latent states.

Vincent et al. (2010) interpret training the autoencoder as maximizing the mutual information between the input and the latent states.

Let X be a sample, $Z = f(X; \theta)$ its latent representation, and $q(x, z)$ the distribution of (X, Z).

Vincent et al. (2010) interpret training the autoencoder as maximizing the mutual information between the input and the latent states.

Let X be a sample, $Z = f(X; \theta)$ its latent representation, and $q(x, z)$ the distribution of (X, Z).

We have

$$\underset{\theta}{\operatorname{argmax}}\ \mathbb{I}(X; Z) = \underset{\theta}{\operatorname{argmax}}\ \mathbb{E}_{q(X,Z)}\Big[\log q(X \mid Z)\Big].$$

However, there is no expression of $q(X \mid Z)$ in any reasonable setup.

Given $(X, Z) \sim q_\theta$, for any distribution p we have

$$\mathbb{E}_{q(X,Z)}\Big[\log q(X \mid Z)\Big] \geq \mathbb{E}_{q(X,Z)}\Big[\log p(X \mid Z)\Big].$$

Given $(X, Z) \sim q_\theta$, for any distribution p we have

$$\mathbb{E}_{q(X,Z)}\Big[\log q(X \mid Z)\Big] \geq \mathbb{E}_{q(X,Z)}\Big[\log p(X \mid Z)\Big].$$

So we can in particular try to find a "good p", so that the left term is a good approximation of the right one.

If we consider the following model for p

$$p\,(\,\cdot \mid Z = z) = \mathcal{N}(g(z;\eta), \sigma)$$

where g is deterministic and σ fixed

If we consider the following model for p

$$p\,(\,\cdot \mid Z = z) = \mathcal{N}(g(z;\eta), \sigma)$$

where g is deterministic and σ fixed, we get

$$\mathbb{E}_{q(X,Z)}\Big[\log p(X \mid Z)\Big] = -\frac{1}{2\sigma^2}\mathbb{E}_{q(X,Z)}\left[\|X - g(f(X);\eta)\|^2\right] + k.$$

If we consider the following model for p

$$p\left(\cdot \mid Z=z\right)=\mathscr{N}(g(z;\eta),\sigma)$$

where g is deterministic and σ fixed, we get

$$\mathbb{E}_{q(X,Z)}\Big[\log p(X \mid Z)\Big]=-\frac{1}{2\sigma^2}\mathbb{E}_{q(X,Z)}\left[\|X-g(f(X);\eta)\|^2\right]+k.$$

If optimizing η makes the bound tight, the final loss is the reconstruction error

$$\underset{\theta}{\operatorname{argmax}}\ \mathbb{I}(X;Z)\simeq\underset{\theta}{\operatorname{argmin}}\left(\min_{\eta}\ \frac{1}{N}\sum_{n=1}^{N}\|x_n-g(f(x_n);\eta)\|^2\right).$$

If we consider the following model for p

$$p\,(\cdot \mid Z = z) = \mathcal{N}(g(z;\eta), \sigma)$$

where g is deterministic and σ fixed, we get

$$\mathbb{E}_{q(X,Z)}\Big[\log p(X \mid Z)\Big] = -\frac{1}{2\sigma^2}\mathbb{E}_{q(X,Z)}\left[\|X - g(f(X);\eta)\|^2\right] + k.$$

If optimizing η makes the bound tight, the final loss is the reconstruction error

$$\underset{\theta}{\operatorname{argmax}}\ \mathbb{I}(X;Z) \simeq \underset{\theta}{\operatorname{argmin}}\left(\min_{\eta}\ \frac{1}{N}\sum_{n=1}^{N}\|x_n - g(f(x_n);\eta)\|^2\right).$$

This abstract view of the encoder as “maximizing information” justifies its use to build generic encoding layers.

In the perspective of building a good feature representation, just retaining information is not enough, otherwise the identity would be a good choice.

In the perspective of building a good feature representation, just retaining information is not enough, otherwise the identity would be a good choice.

In their work, Vincent et al. propose to degrade the signal with noise before feeding it to the encoder, but to keep the MSE to the original signal.

This makes the encoder retain information about structures beyond local noise.

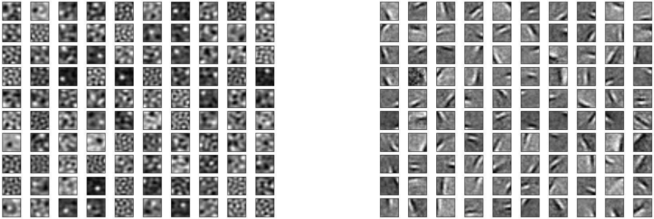


Figure 6: Weight decay vs. Gaussian noise. We show typical filters learnt from natural image patches in the over-complete case (200 hidden units). *Left:* regular autoencoder with weight decay. We tried a wide range of weight-decay values and learning rates: filters never appeared to capture a more interesting structure than what is shown here. Note that some local blob detectors are recovered compared to using no weight decay at all (Figure 5 right). *Right:* a denoising autoencoder with additive Gaussian noise ($\sigma = 0.5$) learns Gabor-like local oriented edge detectors. Clearly the filters learnt are qualitatively very different in the two cases.

(Vincent et al., 2010)

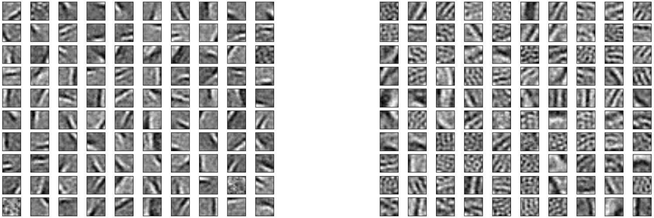
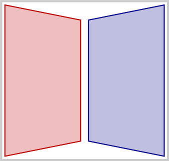


Figure 7: Filters obtained on natural image patches by denoising autoencoders using other noise types. *Left:* with 10% salt-and-pepper noise, we obtain oriented Gabor-like filters. They appear slightly less localized than when using Gaussian noise (contrast with Figure 6 right). *Right:* with 55% zero-masking noise we obtain filters that look like oriented gratings. For the three considered noise types, denoising training appears to learn filters that capture meaningful natural image statistics structure.

(Vincent et al., 2010)

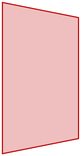
Vincent et al. build deep MLPs whose layers are initialized successively as encoders trained within a noisy autoencoder.

Vincent et al. build deep MLPs whose layers are initialized successively as encoders trained within a noisy autoencoder.

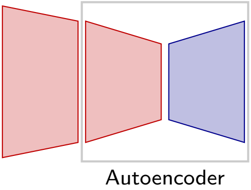


Autoencoder

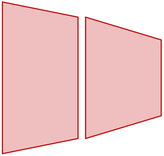
Vincent et al. build deep MLPs whose layers are initialized successively as encoders trained within a noisy autoencoder.



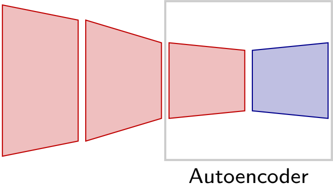
Vincent et al. build deep MLPs whose layers are initialized successively as encoders trained within a noisy autoencoder.



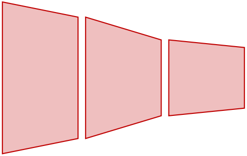
Vincent et al. build deep MLPs whose layers are initialized successively as encoders trained within a noisy autoencoder.



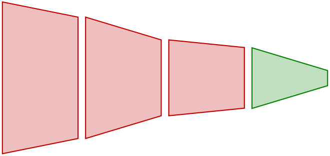
Vincent et al. build deep MLPs whose layers are initialized successively as encoders trained within a noisy autoencoder.



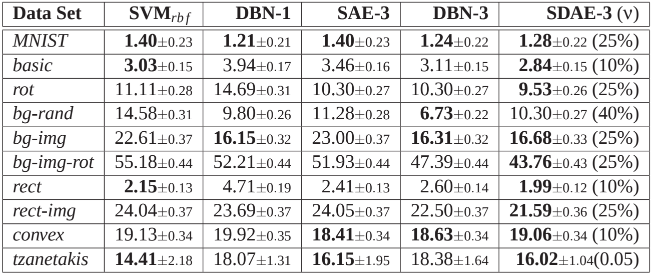
Vincent et al. build deep MLPs whose layers are initialized successively as encoders trained within a noisy autoencoder.



Vincent et al. build deep MLPs whose layers are initialized successively as encoders trained within a noisy autoencoder.



A final classifying layer is added and the full structure can be fine-tuned.



Data Set	**SVM**$_{rbf}$	**DBN-1**	**SAE-3**	**DBN-3**	**SDAE-3** (ν)
MNIST	**1.40**±0.23	**1.21**±0.21	**1.40**±0.23	**1.24**±0.22	**1.28**±0.22 (25%)
basic	**3.03**±0.15	3.94±0.17	3.46±0.16	3.11±0.15	**2.84**±0.15 (10%)
rot	11.11±0.28	14.69±0.31	10.30±0.27	10.30±0.27	**9.53**±0.26 (25%)
bg-rand	14.58±0.31	9.80±0.26	11.28±0.28	**6.73**±0.22	10.30±0.27 (40%)
bg-img	22.61±0.37	**16.15**±0.32	23.00±0.37	**16.31**±0.32	**16.68**±0.33 (25%)
bg-img-rot	55.18±0.44	52.21±0.44	51.93±0.44	47.39±0.44	**43.76**±0.43 (25%)
rect	**2.15**±0.13	4.71±0.19	2.41±0.13	2.60±0.14	**1.99**±0.12 (10%)
rect-img	24.04±0.37	23.69±0.37	24.05±0.37	22.50±0.37	**21.59**±0.36 (25%)
convex	19.13±0.34	19.92±0.35	**18.41**±0.34	**18.63**±0.34	**19.06**±0.34 (10%)
tzanetakis	**14.41**±2.18	18.07±1.31	**16.15**±1.95	18.38±1.64	**16.02**±1.04(0.05)

(Vincent et al., 2010)

Variational Autoencoders

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.

We consider the following two distributions:

- p is the distribution on $\mathscr{X} \times \mathbb{R}^d$ of a pair (X, Z) composed of an encoding state $Z \sim \mathscr{N}(0, I)$ and the output of the decoder g on it.
- q is the distribution on $\mathscr{X} \times \mathbb{R}^d$ of a pair (X, Z) composed of a sample X taken from the data distribution and the output of the encoder on it,

We consider the following two distributions:

- p is the distribution on $\mathscr{X} \times \mathbb{R}^d$ of a pair (X, Z) composed of an encoding state $Z \sim \mathscr{N}(0, I)$ and the output of the decoder g on it.
- q is the distribution on $\mathscr{X} \times \mathbb{R}^d$ of a pair (X, Z) composed of a sample X taken from the data distribution and the output of the encoder on it,

Our goal is that $p(X)$ mimics the data-distribution $q(X)$, that is to find g that maximizes the log-likelihood

$$\frac{1}{N} \sum_n \log p(x_n) = \hat{\mathbb{E}}_{q(X)} \Big[\log p(X) \Big].$$

We consider the following two distributions:

- p is the distribution on $\mathscr{X} \times \mathbb{R}^d$ of a pair (X, Z) composed of an encoding state $Z \sim \mathscr{N}(0, I)$ and the output of the decoder g on it.
- q is the distribution on $\mathscr{X} \times \mathbb{R}^d$ of a pair (X, Z) composed of a sample X taken from the data distribution and the output of the encoder on it,

Our goal is that $p(X)$ mimics the data-distribution $q(X)$, that is to find g that maximizes the log-likelihood

$$\frac{1}{N} \sum_n \log p(x_n) = \hat{\mathbb{E}}_{q(X)} \Big[\log p(X) \Big].$$

However, with a complicated g, we can sample z and compute $g(z)$, but cannot compute $p(x)$ for a given x, and even less compute its derivatives.

The **Variational Autoencoder** proposed by Kingma and Welling (2013) relies on a tractable approximation of this log-likelihood.

Note that their framework involves **stochastic** encoder f, and decoder g, whose outputs depend on both their inputs and additional randomness.

Remember that $q(X)$ is the data distribution, and $q(Z \mid X = x)$ is the distribution of the latent encoding $f(x)$. We want to maximize

$$\mathbb{E}_{q(X)}\Big[\log p(X)\Big],$$

or equivalently minimize

$$\begin{aligned}\mathbb{E}_{q(X)}\Big[\log q(X) - \log p(X)\Big] &= \mathbb{D}_{\mathsf{KL}}(q(X) \,\|\, p(X)) \\ &\leq \mathbb{D}_{\mathsf{KL}}(q(X,Z) \,\|\, p(X,Z)) \,.\end{aligned}$$

Remember that $q(X)$ is the data distribution, and $q(Z \mid X = x)$ is the distribution of the latent encoding $f(x)$. We want to maximize

$$\mathbb{E}_{q(X)}\Big[\log p(X)\Big],$$

or equivalently minimize

$$\begin{aligned}\mathbb{E}_{q(X)}\Big[\log q(X) - \log p(X)\Big] &= \mathbb{D}_{\mathsf{KL}}(q(X) \,\|\, p(X)) \\ &\leq \mathbb{D}_{\mathsf{KL}}(q(X, Z) \,\|\, p(X, Z)) \,.\end{aligned}$$

We will minimize this latter bound, that can be rewritten as

$$\begin{aligned}&\mathbb{D}_{\mathsf{KL}}(q(X, Z) \,\|\, p(X, Z)) = \\ &\quad \mathbb{E}_{q(X)}\Big[\mathbb{D}_{\mathsf{KL}}(q(Z \mid X) \,\|\, p(Z))\Big] - \mathbb{E}_{q(X,Z)}\Big[\log p(X \mid Z)\Big] + \mathbb{H}(q(X)).\end{aligned}$$

Kingma and Welling model both $q(Z \mid X)$ and $p(X \mid Z)$ with Gaussians with diagonal covariance.

The first term of $\mathscr{L}$ is the average of

$$\mathbb{D}_{\mathsf{KL}}\big(\underbrace{q(Z \mid X = x)}_{\mathcal{N}(\mu^f(x),\sigma^f(x))} \parallel \underbrace{p(Z)}_{\mathcal{N}(0,I)}\big) = -\frac{1}{2}\sum_d \left(1 + 2\log\sigma^f_d(x) - \left(\mu^f_d(x)\right)^2 - \left(\sigma^f_d(x)\right)^2\right).$$

over the x_ns.

Kingma and Welling model both $q(Z \mid X)$ and $p(X \mid Z)$ with Gaussians with diagonal covariance.

The first term of $\mathscr{L}$ is the average of

$$\mathbb{D}_{\mathsf{KL}}\big(\underbrace{q(Z \mid X = x)}_{\mathscr{N}(\mu^f(x),\sigma^f(x))} \parallel \underbrace{p(Z)}_{\mathscr{N}(0,I)}\big) = -\frac{1}{2}\sum_d \left(1 + 2\log \sigma_d^f(x) - \left(\mu_d^f(x)\right)^2 - \left(\sigma_d^f(x)\right)^2\right).$$

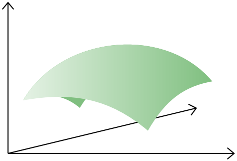
over the x_ns.

The second term is the average of

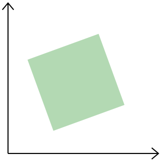
$$-\log \underbrace{p(X = x \mid Z = z)}_{\mathscr{N}(\mu^g(x),\sigma^g(x))(z)} = \frac{1}{2}\sum_d \left(\log 2\pi + 2\log \sigma_d^g(z) + \frac{(x_d - \mu_d^g(z))^2}{\left(\sigma_d^g(z)\right)^2}\right)$$

over the x_n, with one z_n sampled for each (could be more)

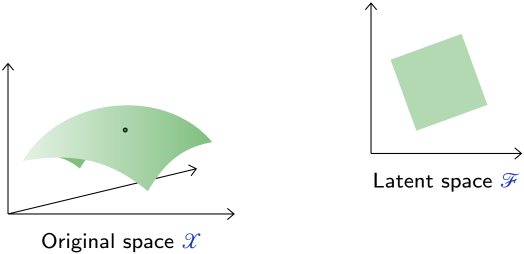
$$z_n \sim \mathscr{N}\left(\mu^f(x_n), \sigma^f(x_n)\right),\ n = 1, \dots, N.$$

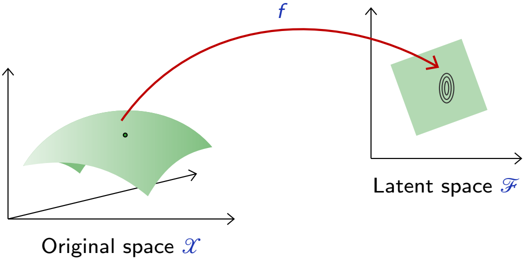


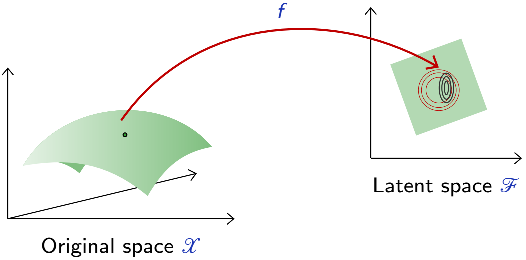
Original space $\mathscr{X}$

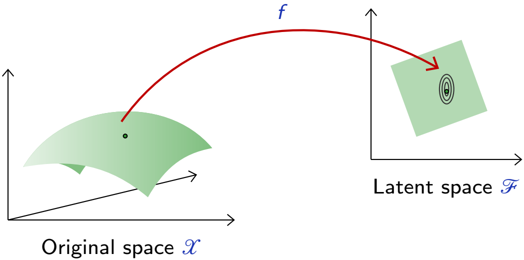


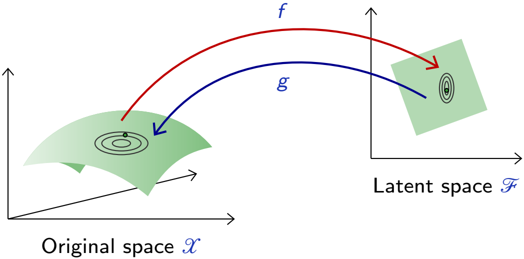
Latent space $\mathscr{F}$


Original space $\mathscr{X}$
Latent space $\mathscr{F}$


f
Original space $\mathscr{X}$
Latent space $\mathscr{F}$


f
Original space $\mathscr{X}$
Latent space $\mathscr{F}$


f
Latent space $\mathscr{F}$
Original space $\mathscr{X}$


f
g
Latent space $\mathscr{F}$
Original space $\mathscr{X}$

Regarding implementation: the encoder now maps to twice the number of dimensions, which corresponds to the μ^fs and $\log\left((\sigma^f)^2\right)$s.

Regarding implementation: the encoder now maps to twice the number of dimensions, which corresponds to the μ^fs and $\log\left((\sigma^f)^2\right)$s.

Also, as in Kingma and Welling (2013), **we use a fixed variance of 1 for the decoder.** So it outputs the μ^gs alone, and its dimension remains unchanged.

The first term of the loss is the average of

$$\mathbb{D}_{\mathrm{KL}}\left(q(Z \mid X = x) \,\|\, p(Z)\right) = -\frac{1}{2} \sum_d \left(1 + 2 \log \sigma_d^f(x) - \left(\mu_d^f(x)\right)^2 - \left(\sigma_d^f(x)\right)^2\right).$$

over the x_n.

The first term of the loss is the average of

$$\mathbb{D}_{\mathsf{KL}}\left(q(Z \mid X = x) \,\|\, p(Z)\right) = -\frac{1}{2} \sum_{d} \left(1 + 2 \log \sigma_d^f(x) - \left(\mu_d^f(x)\right)^2 - \left(\sigma_d^f(x)\right)^2\right).$$

over the x_n.

This can be implemented as

```
param_f = model.encode(input)
mu_f, logvar_f = param_f.split(param_f.size(1)//2, 1)

kl = - 0.5 * (1 + logvar_f - mu_f.pow(2) - logvar_f.exp())
kl_loss = kl.sum() / input.size(0)
```

Since we use a constant variance of 1 for the decoder, the second term of $\mathscr{L}$ becomes the average of

$$-\log p(X = x \mid Z = z) = \frac{1}{2} \sum_{d} (x_d - \mu^g_d(z))^2 + \text{cst}$$

over the x_n, with one z_n sampled for each, *i.e.*

$$z_n \sim \mathcal{N}\left(\mu^f(x_n), \sigma^f(x_n)\right),\ n = 1, \dots, N.$$

Since we use a constant variance of 1 for the decoder, the second term of $\mathscr{L}$ becomes the average of

$$-\log p(X = x \mid Z = z) = \frac{1}{2} \sum_d (x_d - \mu^g_d(z))^2 + \text{cst}$$

over the x_n, with one z_n sampled for each, *i.e.*

$$z_n \sim \mathscr{N}\left(\mu^f(x_n), \sigma^f(x_n)\right), \; n = 1, \dots, N.$$

This can be implemented as

```
std_f = torch.exp(0.5 * logvar_f)
z = torch.empty_like(mu_f).normal_() * std_f + mu_f
output = model.decode(z)

fit = 0.5 * (output - input).pow(2)
fit_loss = fit.sum() / input.size(0)
```

We had for the standard autoencoder

```
z = model.encode(input)
output = model.decode(z)
loss = 0.5 * (output - input).pow(2).sum() / input.size(0)
```

We had for the standard autoencoder

```
z = model.encode(input)
output = model.decode(z)
loss = 0.5 * (output - input).pow(2).sum() / input.size(0)
```

and putting everything together we get for the VAE

```
param_f = model.encode(input)
mu_f, logvar_f = param_f.split(param_f.size(1)//2, 1)

kl = - 0.5 * (1 + logvar_f - mu_f.pow(2) - logvar_f.exp())
kl_loss = kl.sum() / input.size(0)

std_f = torch.exp(0.5 * logvar_f)
z = torch.empty_like(mu_f).normal_() * std_f + mu_f
output = model.decode(z)

fit = 0.5 * (output - input).pow(2)
fit_loss = fit.sum() / input.size(0)

loss = kl_loss + fit_loss
```

We had for the standard autoencoder

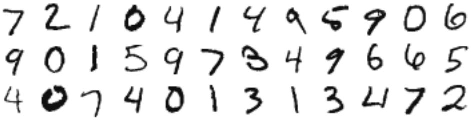
```
z = model.encode(input)
output = model.decode(z)
loss = 0.5 * (output - input).pow(2).sum() / input.size(0)
```

and putting everything together we get for the VAE

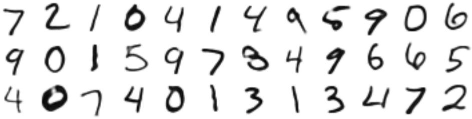
```
param_f = model.encode(input)
mu_f, logvar_f = param_f.split(param_f.size(1)//2, 1)

kl = - 0.5 * (1 + logvar_f - mu_f.pow(2) - logvar_f.exp())
kl_loss = kl.sum() / input.size(0)

std_f = torch.exp(0.5 * logvar_f)
z = torch.empty_like(mu_f).normal_() * std_f + mu_f
output = model.decode(z)

fit = 0.5 * (output - input).pow(2)
fit_loss = fit.sum() / input.size(0)

loss = kl_loss + fit_loss
```

During inference we do not sample, and instead use μ^f and μ^g as prediction.

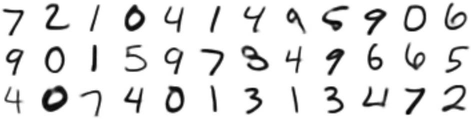
Original



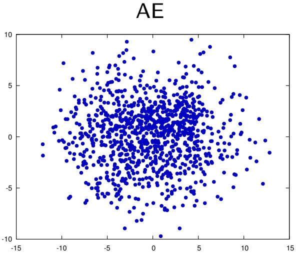
Autoencoder reconstruction ($d = 32$)



Variational Autoencoder reconstruction ($d = 32$)

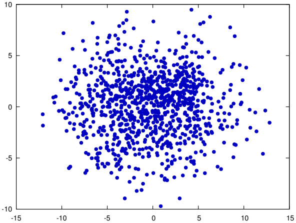


We can look at two latent features to check that they are Normal for the VAE.

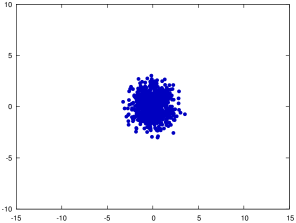


We can look at two latent features to check that they are Normal for the VAE.

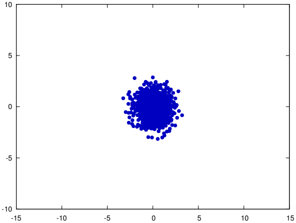
AE



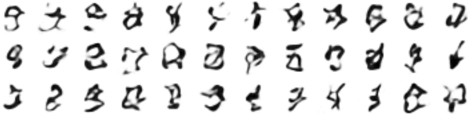
VAE



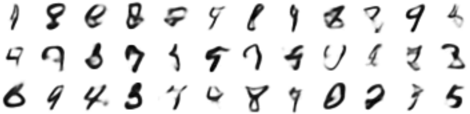
$\mathcal{N}(0,1)$



Autoencoder sampling ($d = 32$)



Variational Autoencoder sampling ($d = 32$)



The end

References

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

P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research (JMLR)*, 11:3371–3408, 2010.