Applied Deep Learning

Chapter 8: Representation and Generation

Ali Bereyhi

ali.bereyhi@utoronto.ca

Department of Electrical and Computer Engineering

University of Toronto

Fall 2025

Generating New Data via AEs

Let's keep the track of their applications

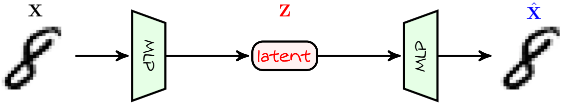
1. *Compression*
2. *Finding a sparse representation of data*
3. *Denoising*
4. *Data Generation*
   - ↳ *We intend to generate a new sample by our decoder from a seed*
     - ↳ *for instance we generate a random seed and give it to decoder*
     - ↳ *the decoder returns an image which was not in the dataset*

\+ *That sounds crazy!*

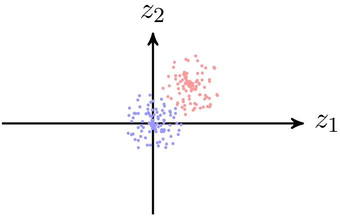
\- Well! It's not as crazy as it sounds

Looking into *Latent Space*

Let's get back to our MNIST example: *assume that we set the dataset to only contain images of handwritten 1 and 8, and train an AE to compress them into 2-dimensional latent representations*

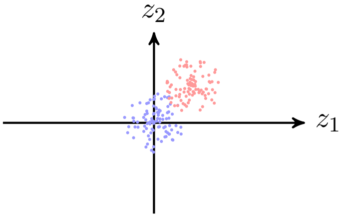


We now do a simple experiment: we pass all images of 1 and 8 that we have and mark their latent representations with blue and red



Looking into *Latent Space*

These points show a specific behavior: *for each class, they are concentrated within an specific region*



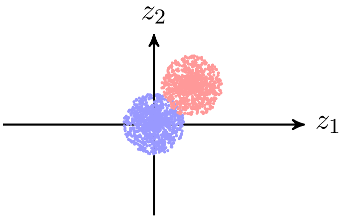
Recall: Data Space and Distribution

In Chapter 3 we said that we can look at our dataset as a set of samples drawn by some distribution from a data space that contains all possible data-points

This means that we have actually lots of other possible handwritten 1 and 8 that are not available in our dataset!

Looking into *Latent Space*

+ *What happens if we send all of them through our AE?*
- Well! We can't say, as we have no access to them, but we may guess!



They are probably some compact regions

we call the union of those regions the latent space

Similar to data space, we cannot access it! We just imagine it!

First Try for Generating Data

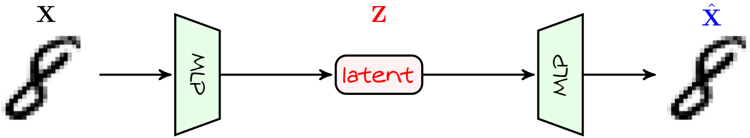
We could use this behavior to generate a new data

- *We sample a new point in the region that we guess is the latent space*
- *We send this sample over the decoder of AE: if we are lucky*
    - ↳ *This sample is latent representation of a data-point that is out of our dataset*
    - ↳ *The decoder is trained well and can reconstruct that data-point*
    - ↳ *We have now a data-point out of our dataset*

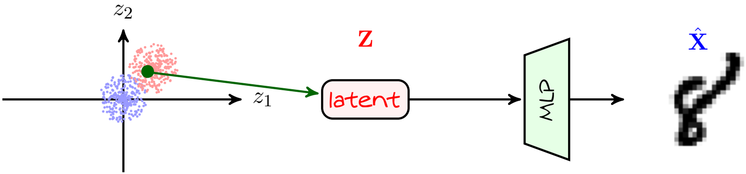
We have generated data out of some random seed $\equiv$ *latent sample*

First Try for Generating Data

We first train



We then sample the latent space



Drawbacks of Generation via *Vanilla AEs*

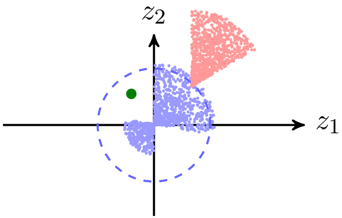
Even though the idea seems to be intuitive: *it turns out that it does not work very well when we use basic AE architectures*

- *Frequency of invalid generated data is quite high*
  - ↳ *For instance, the decoder returns an image which is not a digit*
- *This is not due to bad training: it happens even if AE compresses perfectly*

The main reason is our significant lack of knowledge about latent space

- *We guessed that latent space is compact and smoothly shaped*
  - ↳ *Apparently, this is not the case!*
- *By extensive experimental investigations, we could see*
  - ↳ *Latent space can be extremely asymmetric*
  - ↳ *It can be hugely discontinuous*

Drawbacks of Generation via *Vanilla AEs*



When we sample from the postulated latent space

- *with high chance we could sample from a region out of true latent space*
  ↳ *We hence send a compressed version of invalid image*
- *decoder returns an invalid data-point!*

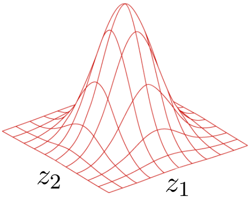
\+ *How can we resolve this issue?*

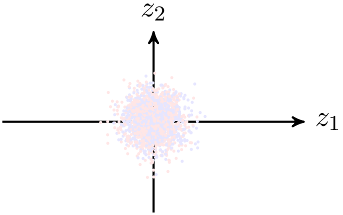
– *We may restrict encoder to encode into compact and symmetric region*

Generating via Variational AEs

Variational AEs apply some trick to *make sure that*

latent representation look like samples of a Gaussian distribution

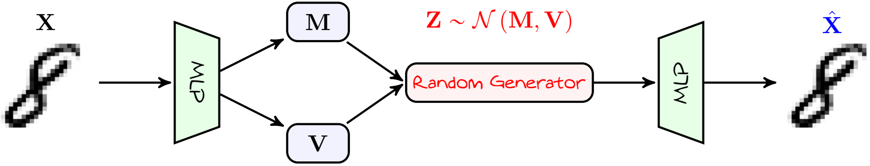




Specifically, a Gaussian distribution with mean zero and variance one: $\mathcal{N}(\mathbf{0}, \mathbf{1})$

+ *How can we do it?*
– Well! The trick is quite sophisticated!

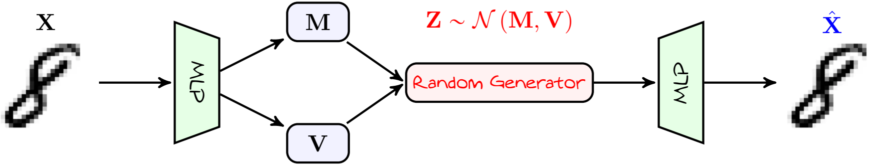
Variational AE: *Architecture*



Let's formulate: *say input is* $\mathbf{X}$, *latent representation is* $\mathbf{Z}$, *and* $\hat{\mathbf{X}}$ *is output*

- *We start by encoding: encoder gets* $\mathbf{X}$ *and returns*
  - ↳ $\mathbf{M}$ *which is of same shape as* $\mathbf{Z}$*: this plays the role of mean*
  - ↳ $\mathbf{V}$ *which is of same shape as* $\mathbf{Z}$*: this plays the role of -variance*
- *We also then generate latent representations at random*
  - ↳ $\mathbf{Z}$ *is generated from a Gaussian distribution*
  - ↳ *Mean of* $\mathbf{Z}$ *is* $\mathbf{M}$ *and its variance is* $\mathbf{V}$
- *We give latent representations to the decoder*
- *We train such that decoder recovers the input data*

Variational AE: *Loss*



Let's specify the loss

- *We need to recover from latent representation, i.e., we want* $\hat{\mathbf{X}} = \mathbf{X}$
  - ↳ *Loss is proportional to the difference between* $\mathbf{X}$ *and* $\hat{\mathbf{X}}$
- *We want a zero-mean and unit-variance Gaussian latent representation*
  - ↳ *Distribution of* $\mathbf{Z}$ *should be* $\mathcal{N}(\mathbf{0}, \mathbf{1})$
  - ↳ *But* $\mathbf{Z}$ *is generated as* $\mathcal{N}(\mathbf{M}, \mathbf{V})$
  - ↳ *Loss should be penalized by difference between the two distribution*

Loss in VAEs

Loss is proportional to recovery error and difference between actual and intended distributions of $\mathbf{Z}$ $\equiv$ *let's call them* $p_{\mathbf{Z}}$ *and* $q_{\mathbf{Z}}$*, respectively*

$$\hat{R} = \mathcal{L}(\hat{\mathbf{X}}, \mathbf{X}) + \lambda \mathrm{Div}\left(p_{\mathbf{Z}}, q_{\mathbf{Z}}\right)$$

for regularizer λ *and a difference measure* $\mathrm{Div}\left(p_{\mathbf{Z}}, q_{\mathbf{Z}}\right)$

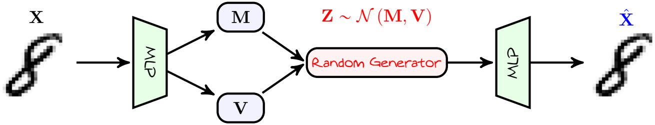
The classical choice for $\mathrm{Div}\left(p_{\mathbf{Z}}, q_{\mathbf{Z}}\right)$ *is the KL-divergence*

$$\begin{aligned}\mathrm{Div}\left(p_{\mathbf{Z}}, q_{\mathbf{Z}}\right) &= \mathrm{KL}\left(p_{\mathbf{Z}} \| q_{\mathbf{Z}}\right) \\ &= \int p_{\mathbf{Z}}\left(\mathbf{Z}\right) \log \frac{p_{\mathbf{Z}}\left(\mathbf{Z}\right)}{q_{\mathbf{Z}}\left(\mathbf{Z}\right)} \mathrm{d}\mathbf{Z} = F\left(\mathbf{M}, \mathbf{V}\right)\end{aligned}$$

So, we basically train by minimizing

$$\hat{R} = \mathcal{L}(\hat{\mathbf{X}}, \mathbf{X}) + \lambda F\left(\mathbf{M}, \mathbf{V}\right)$$

Training VAEs



Let's see how training looks: *say we are training with single sample* $\mathbf{X}$

- *Pass forward* $\mathbf{X}$ *through encoder and decoder*
- *Backpropagate by first computing* $\nabla_{\hat{\mathbf{X}}} \hat{R}$
  - ↳ *Backpropagate till the latent space*
  - ↳ *At the bottleneck, we need to compute* $\nabla_{\mathbf{M}} \hat{R}$ *and* $\nabla_{\mathbf{V}} \hat{R}$

$$\nabla_{\mathbf{M}} \hat{R} = \underbrace{\nabla_{\hat{\mathbf{X}}} \hat{R} \circ \nabla_{\mathbf{M}} \hat{\mathbf{X}}}_{\text{computed by backpropagation}} + \lambda \nabla_{\mathbf{M}} F(\mathbf{M}, \mathbf{V})$$

  - ↳ *Start from* $\nabla_{\mathbf{M}} \hat{R}$ *and* $\nabla_{\mathbf{V}} \hat{R}$ *backpropagate till input*
- *Update weights and go for the next round*

VAEs: *Final Remarks*

Attention!

We have skipped too much details to make it very simple: the concrete approach to understand VAEs is to

1. *Start with looking at the NNs as machines that realize distributions*
2. *Get to the problem of Variational Inference*
3. *Develop an AE that performs Variational Inference*

We then end up with VAEs

The above approach will be taken in the course Generative AI

But for know: *you have the main tools* to *implement a VAE*

↳ *You may just be unsure about some details, e.g.,*

   ↳ *Why particular expressions are defined that way?!*

↳ *You can find the answers in the course Generative AI*

The End!

Remember that *you have the main tools to apply deep learning*

- ↳ *Always search for the main three components*
  - ↳ *Model, Dataset and Loss*
- ↳ *Always imagine how to backpropagate over the architecture*
- ↳ *You got into new challenges?*
  - ↳ *Search online* ☺
  - ↳ *Reach out to me! I would be more than happy!*

Next in line . . .

- ↳ *This Summer Semester*
  - ↳ *Generative AI*
- ↳ *Next Fall Semester*
  - ↳ *Creative Applications of NLP*
  - ↳ *Reinforcement Learning*