ECE 1508S2: Applied Deep Learning

Chapter 8: Auto Encoders

Ali Bereyhi

ali.bereyhi@utoronto.ca

Department of Electrical and Computer Engineering

University of Toronto

Winter 2024

Road-map: *Auto Encoders*

In this chapter, we get to know *Auto Encoders* which are very powerful architectures for feature extraction and data generation:

Road-map: *Auto Encoders*

In this chapter, we get to know *Auto Encoders* which are very powerful architectures for feature extraction and data generation: *this chapter is just an introduction, and we will see more details in next Fall Semester*

Road-map: *Auto Encoders*

In this chapter, we get to know *Auto Encoders* which are very powerful architectures for feature extraction and data generation: *this chapter is just an introduction, and we will see more details in next Fall Semester*

- *There's going to be an ECE course on Generative Models*
  - ↳ *We start from this point there!*

Road-map: *Auto Encoders*

In this chapter, we get to know *Auto Encoders* which are very powerful architectures for feature extraction and data generation: *this chapter is just an introduction, and we will see more details in next Fall Semester*

- *There's going to be an ECE course on Generative Models*
  - ↳ *We start from this point there!*

The best way to understand Auto Encoders is to look at a simple representation problem, but before that let's make an agreement

$$\textit{Auto Encoders} \equiv \textit{Autoencoders} \equiv \textit{AEs}$$

from now on!

Simple Problem: *Representation in Smaller Dimension*

Consider a simple problem: *we have a dataset of two-dimensional data-points*

$$\mathbb{D} = \{\mathbf{x}_b : b = 1, \dots, B\}$$

where $\mathbf{x}_b \in \mathbb{R}^2$, *and we want to find two weight vectors*

Simple Problem: *Representation in Smaller Dimension*

Consider a simple problem: *we have a dataset of two-dimensional data-points*

$$\mathbb{D} = \{\mathbf{x}_b : b = 1, \dots, B\}$$

where $\mathbf{x}_b \in \mathbb{R}^2$*, and we want to find two weight vectors*

- *A weight vector* $\mathbf{w} \in \mathbb{R}^2$ *that compresses* $\mathbf{x}_b$ *into a single number* z_b *as*

$$z_b = \mathbf{w}^\mathsf{T} \mathbf{x}_b$$

Simple Problem: *Representation in Smaller Dimension*

Consider a simple problem: *we have a dataset of two-dimensional data-points*

$$\mathbb{D} = \{\mathbf{x}_b : b = 1, \dots, B\}$$

where $\mathbf{x}_b \in \mathbb{R}^2$, and we want to find two weight vectors

- *A weight vector $\mathbf{w} \in \mathbb{R}^2$ that compresses $\mathbf{x}_b$ into a single number z_b as*

$$z_b = \mathbf{w}^\mathsf{T} \mathbf{x}_b$$

- *A weight vector $\hat{\mathbf{w}}$ that decompresses $\mathbf{x}_b$ from z_b*

$$\hat{\mathbf{x}}_b = \hat{\mathbf{w}} z_b$$

Simple Problem: *Representation in Smaller Dimension*

Consider a simple problem: *we have a dataset of two-dimensional data-points*

$$\mathbb{D} = \{\mathbf{x}_b : b = 1, \dots, B\}$$

where $\mathbf{x}_b \in \mathbb{R}^2$, and we want to find two weight vectors

- *A weight vector $\mathbf{w} \in \mathbb{R}^2$ that compresses $\mathbf{x}_b$ into a single number z_b as*

$$z_b = \mathbf{w}^{\mathsf{T}} \mathbf{x}_b$$

- *A weight vector $\hat{\mathbf{w}}$ that decompresses $\mathbf{x}_b$ from z_b*

$$\hat{\mathbf{x}}_b = \hat{\mathbf{w}} z_b$$

\+ *Can we do this?*

– Well! Not always!

Simple Problem: *Representation in Smaller Dimension*

Let's try our ML knowledge:

Simple Problem: *Representation in Smaller Dimension*

Let's try our ML knowledge: *say we set the the compression vector to $\mathbf{w}$; then, the compressed version of the dataset is*

$$\begin{bmatrix} z_1 & \ldots & z_B \end{bmatrix} = \begin{bmatrix} \mathbf{w}^\mathsf{T}\mathbf{x}_1 & \ldots & \mathbf{w}^\mathsf{T}\mathbf{x}_B \end{bmatrix}$$

Simple Problem: *Representation in Smaller Dimension*

Let's try our ML knowledge: *say we set the the compression vector to $\mathbf{w}$; then, the compressed version of the dataset is*

$$\begin{aligned}\begin{bmatrix} z_1 & \dots & z_B \end{bmatrix} &= \begin{bmatrix} \mathbf{w}^\mathsf{T}\mathbf{x}_1 & \dots & \mathbf{w}^\mathsf{T}\mathbf{x}_B \end{bmatrix} \\ &= \mathbf{w}^\mathsf{T} \underbrace{\begin{bmatrix} \mathbf{x}_1 & \dots & \mathbf{x}_B \end{bmatrix}}_{\mathbf{X}} =\end{aligned}$$

Simple Problem: *Representation in Smaller Dimension*

Let's try our ML knowledge: *say we set the the compression vector to* $\mathbf{w}$*; then, the compressed version of the dataset is*

$$\begin{bmatrix} z_1 & \dots & z_B \end{bmatrix} = \begin{bmatrix} \mathbf{w}^\mathsf{T}\mathbf{x}_1 & \dots & \mathbf{w}^\mathsf{T}\mathbf{x}_B \end{bmatrix}$$
$$= \mathbf{w}^\mathsf{T} \underbrace{\begin{bmatrix} \mathbf{x}_1 & \dots & \mathbf{x}_B \end{bmatrix}}_{\mathbf{X} \in \mathbb{R}^{2 \times B}} = \mathbf{w}^\mathsf{T}\mathbf{X}$$

Simple Problem: *Representation in Smaller Dimension*

Let's try our ML knowledge: *say we set the the compression vector to* $\mathbf{w}$*; then, the compressed version of the dataset is*

$$\begin{bmatrix} z_1 & \dots & z_B \end{bmatrix} = \begin{bmatrix} \mathbf{w}^\mathsf{T}\mathbf{x}_1 & \dots & \mathbf{w}^\mathsf{T}\mathbf{x}_B \end{bmatrix}$$
$$= \mathbf{w}^\mathsf{T} \underbrace{\begin{bmatrix} \mathbf{x}_1 & \dots & \mathbf{x}_B \end{bmatrix}}_{\mathbf{X} \in \mathbb{R}^{2\times B}} = \mathbf{w}^\mathsf{T}\mathbf{X}$$

Now, let's find the decompressed versions using $\hat{\mathbf{w}}$

$$\underbrace{\begin{bmatrix} \hat{\mathbf{x}}_1 & \dots & \hat{\mathbf{x}}_B \end{bmatrix}}_{\hat{\mathbf{X}} \in \mathbb{R}^{2\times B}} = \begin{bmatrix} \hat{\mathbf{w}} z_1 & \dots & \hat{\mathbf{w}} z_B \end{bmatrix}$$

Simple Problem: *Representation in Smaller Dimension*

Let's try our ML knowledge: *say we set the the compression vector to* $\mathbf{w}$*; then, the compressed version of the dataset is*

$$\begin{bmatrix} z_1 & \dots & z_B \end{bmatrix} = \begin{bmatrix} \mathbf{w}^\mathsf{T}\mathbf{x}_1 & \dots & \mathbf{w}^\mathsf{T}\mathbf{x}_B \end{bmatrix}$$
$$= \mathbf{w}^\mathsf{T} \underbrace{\begin{bmatrix} \mathbf{x}_1 & \dots & \mathbf{x}_B \end{bmatrix}}_{\mathbf{X}\in\mathbb{R}^{2\times B}} = \mathbf{w}^\mathsf{T}\mathbf{X}$$

Now, let's find the decompressed versions using $\hat{\mathbf{w}}$

$$\underbrace{\begin{bmatrix} \hat{\mathbf{x}}_1 & \dots & \hat{\mathbf{x}}_B \end{bmatrix}}_{\hat{\mathbf{X}}\in\mathbb{R}^{2\times B}} = \begin{bmatrix} \hat{\mathbf{w}} z_1 & \dots & \hat{\mathbf{w}} z_B \end{bmatrix}$$
$$= \hat{\mathbf{w}} \begin{bmatrix} z_1 & \dots & z_B \end{bmatrix} = \hat{\mathbf{w}}\mathbf{w}^\mathsf{T}\mathbf{X}$$

Simple Problem: *Representation in Smaller Dimension*

Let's try our ML knowledge: *say we set the the compression vector to $\mathbf{w}$; then, the compressed version of the dataset is*

$$\begin{bmatrix} z_1 & \dots & z_B \end{bmatrix} = \begin{bmatrix} \mathbf{w}^\mathsf{T}\mathbf{x}_1 & \dots & \mathbf{w}^\mathsf{T}\mathbf{x}_B \end{bmatrix} = \mathbf{w}^\mathsf{T} \underbrace{\begin{bmatrix} \mathbf{x}_1 & \dots & \mathbf{x}_B \end{bmatrix}}_{\mathbf{X} \in \mathbb{R}^{2\times B}} = \mathbf{w}^\mathsf{T}\mathbf{X}$$

Now, let's find the decompressed versions using $\hat{\mathbf{w}}$

$$\underbrace{\begin{bmatrix} \hat{\mathbf{x}}_1 & \dots & \hat{\mathbf{x}}_B \end{bmatrix}}_{\hat{\mathbf{X}} \in \mathbb{R}^{2\times B}} = \begin{bmatrix} \hat{\mathbf{w}} z_1 & \dots & \hat{\mathbf{w}} z_B \end{bmatrix} = \hat{\mathbf{w}} \begin{bmatrix} z_1 & \dots & z_B \end{bmatrix} = \hat{\mathbf{w}}\mathbf{w}^\mathsf{T}\mathbf{X}$$

After decompression, we get $\hat{\mathbf{X}} = \hat{\mathbf{w}}\mathbf{w}^\mathsf{T}\mathbf{X}$ which we want to be the same as $\mathbf{X}$

Simple Problem: *Representation in Smaller Dimension*

Let's try our ML knowledge: *we could define a loss and minimize it via SGD*

$$\hat{R} = \mathcal{L}(\hat{\mathbf{X}}, \mathbf{X}) = \|\hat{\mathbf{w}}\mathbf{w}^\mathsf{T}\mathbf{X} - \mathbf{X}\|^2$$

Simple Problem: *Representation in Smaller Dimension*

Let's try our ML knowledge: *we could define a loss and minimize it via SGD*

$$\hat{R} = \mathcal{L}(\hat{\mathbf{X}}, \mathbf{X}) = \|\hat{\mathbf{w}}\mathbf{w}^\mathsf{T}\mathbf{X} - \mathbf{X}\|^2$$

But, we actually do not need to go that far! We decompress $\hat{\mathbf{X}}$ *simply if*

$$\hat{\mathbf{w}}\mathbf{w}^\mathsf{T}\mathbf{X} \stackrel{!}{=} \mathbf{X}$$

Simple Problem: *Representation in Smaller Dimension*

Let's try our ML knowledge: *we could define a loss and minimize it via SGD*

$$\hat{R} = \mathcal{L}(\hat{\mathbf{X}}, \mathbf{X}) = \left\| \hat{\mathbf{w}} \mathbf{w}^{\mathsf{T}} \mathbf{X} - \mathbf{X} \right\|^2$$

But, we actually do not need to go that far! We decompress $\hat{\mathbf{X}}$ *simply if*

$$\hat{\mathbf{w}} \mathbf{w}^{\mathsf{T}} \mathbf{X} \overset{!}{=} \mathbf{X}$$

Or alternatively if we have $\hat{\mathbf{w}} \mathbf{w}^{\mathsf{T}}$ *to be an identity transform*

$$\hat{\mathbf{w}} \mathbf{w}^{\mathsf{T}} \overset{!}{=} \mathbf{I}$$

Simple Problem: *Representation in Smaller Dimension*

Let's try our ML knowledge: *we could define a loss and minimize it via SGD*

$$\hat{R} = \mathcal{L}(\hat{\mathbf{X}}, \mathbf{X}) = \|\hat{\mathbf{w}}\mathbf{w}^\mathsf{T}\mathbf{X} - \mathbf{X}\|^2$$

But, we actually do not need to go that far! We decompress $\hat{\mathbf{X}}$ *simply if*

$$\hat{\mathbf{w}}\mathbf{w}^\mathsf{T}\mathbf{X} \stackrel{!}{=} \mathbf{X}$$

Or alternatively if we have $\hat{\mathbf{w}}\mathbf{w}^\mathsf{T}$ *to be an identity transform*

$$\hat{\mathbf{w}}\mathbf{w}^\mathsf{T} \stackrel{!}{=} \mathbf{I}$$

+ *But it's impossible!*
– Yes! $\hat{\mathbf{w}}\mathbf{w}^\mathsf{T}$ can never be $\mathbf{I}$ *since* $\hat{\mathbf{w}}\mathbf{w}^\mathsf{T}$ *is rank one and* $\mathbf{I}$ *is full-rank!*

Simple Problem: *Representation in Smaller Dimension*

Now, let's assume that we are given the following piece of information: *we know that every single point in the dataset is of the form*

$$\mathbf{x}_b = \begin{bmatrix} 2\alpha_b \\ \alpha_b \end{bmatrix}$$

for some real-valued α_b

Simple Problem: *Representation in Smaller Dimension*

Now, let's assume that we are given the following piece of information: *we know that every single point in the dataset is of the form*

$$\mathbf{x}_b = \begin{bmatrix} 2\alpha_b \\ \alpha_b \end{bmatrix}$$

for some real-valued α_b

> *it turns out we can now do compression and decompression successfully*

Simple Problem: *Representation in Smaller Dimension*

Now, let's assume that we are given the following piece of information: *we know that every single point in the dataset is of the form*

$$\mathbf{x}_b = \begin{bmatrix} 2\alpha_b \\ \alpha_b \end{bmatrix}$$

for some real-valued α_b

it turns out we can now do compression and decompression successfully

---

Let's see how this happens: *let's set compression and decompression vectors to* $\mathbf{w} = [1, -1]^\mathsf{T}$ *and* $\hat{\mathbf{w}} = [2, 1]^\mathsf{T}$. *We can then say*

Simple Problem: *Representation in Smaller Dimension*

Now, let's assume that we are given the following piece of information: *we know that every single point in the dataset is of the form*

$$\mathbf{x}_b = \begin{bmatrix} 2\alpha_b \\ \alpha_b \end{bmatrix}$$

for some real-valued α_b

> *it turns out we can now do compression and decompression successfully*

Let's see how this happens: *let's set compression and decompression vectors to* $\mathbf{w} = [1, -1]^\mathsf{T}$ *and* $\hat{\mathbf{w}} = [2, 1]^\mathsf{T}$. *We can then say*

$$z_b = \begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} 2\alpha_b \\ \alpha_b \end{bmatrix} = \alpha_b$$

Simple Problem: *Representation in Smaller Dimension*

Now, let's assume that we are given the following piece of information: *we know that every single point in the dataset is of the form*

$$\mathbf{x}_b = \begin{bmatrix} 2\alpha_b \\ \alpha_b \end{bmatrix}$$

for some real-valued α_b

> *it turns out we can now do compression and decompression successfully*

Let's see how this happens: *let's set compression and decompression vectors to* $\mathbf{w} = [1, -1]^{\mathsf{T}}$ *and* $\hat{\mathbf{w}} = [2, 1]^{\mathsf{T}}$. *We can then say*

$$z_b = \begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} 2\alpha_b \\ \alpha_b \end{bmatrix} = \alpha_b \rightsquigarrow \hat{\mathbf{x}}_b = \begin{bmatrix} 2 \\ 1 \end{bmatrix} z_b$$

Simple Problem: *Representation in Smaller Dimension*

Now, let's assume that we are given the following piece of information: *we know that every single point in the dataset is of the form*

$$\mathbf{x}_b = \begin{bmatrix} 2\alpha_b \\ \alpha_b \end{bmatrix}$$

for some real-valued α_b

it turns out we can now do compression and decompression successfully

---

Let's see how this happens: *let's set compression and decompression vectors to* $\mathbf{w} = [1, -1]^{\mathsf{T}}$ *and* $\hat{\mathbf{w}} = [2, 1]^{\mathsf{T}}$. *We can then say*

$$z_b = \begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} 2\alpha_b \\ \alpha_b \end{bmatrix} = \alpha_b \rightsquigarrow \hat{\mathbf{x}}_b = \begin{bmatrix} 2 \\ 1 \end{bmatrix} z_b = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \alpha_b = \mathbf{x}_b$$

Simple Problem: *Principle Component*

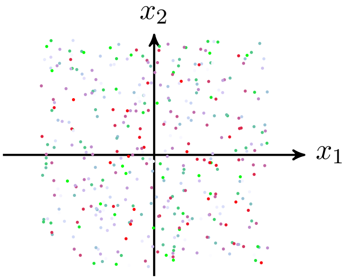
+ *Wait a moment! Why did it happen? We don't have $\hat{\mathbf{w}}\mathbf{w}^\mathsf{T} = \mathbf{I}$!*
- Yes! It happened because we don't need $\hat{\mathbf{w}}\mathbf{w}^\mathsf{T} = \mathbf{I}$ in this case

Simple Problem: *Principle Component*

+ *Wait a moment! Why did it happen? We don't have $\hat{\mathbf{w}}\mathbf{w}^{\mathsf{T}} = \mathbf{I}$!*
– Yes! It happened because we don't need $\hat{\mathbf{w}}\mathbf{w}^{\mathsf{T}} = \mathbf{I}$ in this case

---

A general dataset of 2-dimensional vectors look like



and this dataset cannot be projected on a single axis!

Simple Problem: *Principle Component*

We were successful when *we knew that every data-point is of the form*

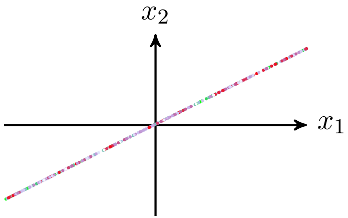
$$\mathbf{x}_b = \begin{bmatrix} 2\alpha_b \\ \alpha_b \end{bmatrix}$$

Simple Problem: *Principle Component*

We were successful when *we knew that every data-point is of the form*

$$\mathbf{x}_b = \begin{bmatrix} 2\alpha_b \\ \alpha_b \end{bmatrix}$$

In this case, the dataset is already on a single rotated axis



We just need to find the value of each point on that axis, i.e., α_b

Principle Component Analysis: *Dimensionality Reduction*

This is in fact *a very basic example of*

$$\textit{Principle Component Analysis} \equiv \textit{PCA}$$

Principle Component Analysis: *Dimensionality Reduction*

This is in fact *a very basic example of*

Principle Component Analysis $\equiv$ *PCA*

PCA: *Minimum Error Formulation*

In PCA, we have a dataset of N-dimensional data-points and learn a weight matrix $\mathbf{W} \in \mathbb{R}^{L \times N}$ that for each $\mathbf{x}_b$ in the dataset generates a latent variable $\mathbf{z}_b = \mathbf{W}\mathbf{x}_b$ such that by linear reconstruction of $\mathbf{x}_b$ from $\mathbf{z}_b$ has minimum error

Principle Component Analysis: *Dimensionality Reduction*

This is in fact *a very basic example of*

Principle Component Analysis $\equiv$ *PCA*

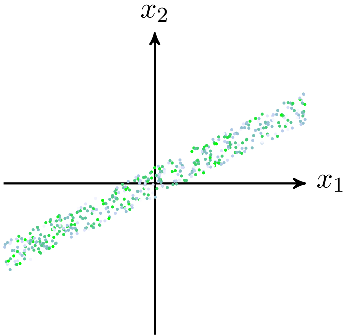
PCA: *Minimum Error Formulation*

In PCA, we have a dataset of N-dimensional data-points and learn a weight matrix $\mathbf{W} \in \mathbb{R}^{L \times N}$ that for each $\mathbf{x}_b$ in the dataset generates a latent variable $\mathbf{z}_b = \mathbf{W}\mathbf{x}_b$ such that by linear reconstruction of $\mathbf{x}_b$ from $\mathbf{z}_b$ has minimum error

+ *Do we always have such a properly in our dataset?!*
– Perfectly no, but approximately yes!

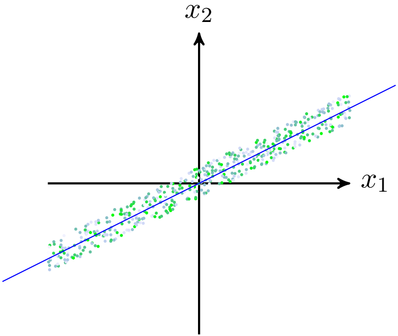
Principle Component Analysis: *Dimensionality Reduction*

We could in practice have compressible dataset with its main variation being on a reduced dimension, e.g., 2-dimensional points that lie around a single line



Principle Component Analysis: *Dimensionality Reduction*

We could in practice have compressible dataset with its main variation being on a reduced dimension, e.g., 2-dimensional points that lie around a single line



We could a smaller latent variable for each data-point

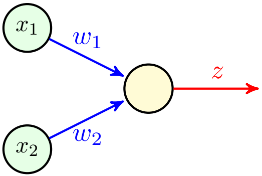
↳ *We accept a minor reconstruction error*

PCA as *Neural Network*

Since we like NNs, *let's represent our simple example as an NN architecture and look at its training*

PCA as *Neural Network*

Since we like NNs, *let's represent our simple example as an NN architecture and look at its training*

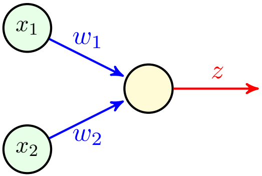


The above NN describe PCA

- *We have two learnable parameters* w_1 *and* w_2

PCA as *Neural Network*

Since we like NNs, *let's represent our simple example as an NN architecture and look at its training*

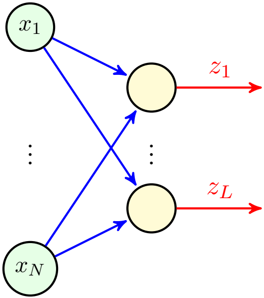


The above NN describe PCA

- *We have two learnable parameters* w_1 *and* w_2
- *We want to train this NN such that* z *is the best representation*
  - ↳ z *is a compression for* $\mathbf{x}$

PCA as *Neural Network*

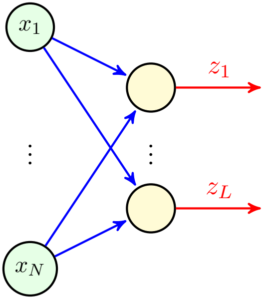
For more general setting, we have N *inputs and* L *latent variables*



The above NN describe more generic PCA setting

PCA as *Neural Network*

For more general setting, we have N *inputs and* L *latent variables*

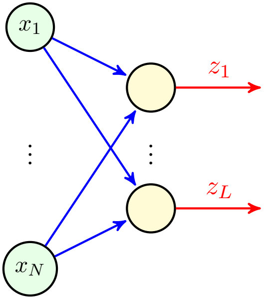


The above NN describe more generic PCA setting

- *We have* $(N+1)L$ *learnable parameters:* NL *weights and* L *biases*

PCA as *Neural Network*

For more general setting, we have N *inputs and* L *latent variables*

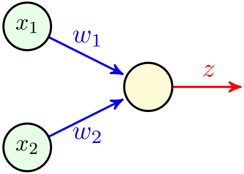


The above NN describe more generic PCA setting

- *We have* $(N+1)L$ *learnable parameters:* NL *weights and* L *biases*
- *We want to train this NN such that* $\mathbf{z}$ *is the best representation*
  - ↳ $\mathbf{z}$ *is lower dimensional representation of* $\mathbf{x}$

PCA as *Neural Network*

+ *But how can we train this NN? We do not have any label!*
– This is because it's an *unsupervised learning problem*

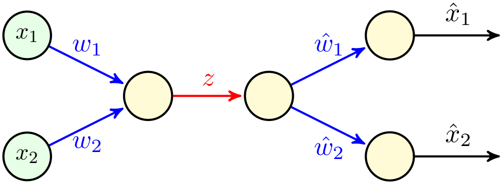


To train this model, we should make some labels

- *We intend to recover* $\mathbf{x}$ *from* z *at the end of the day*

PCA as *Neural Network*

+ *But how can we train this NN? We do not have any label!*
– This is because it's an *unsupervised learning problem*

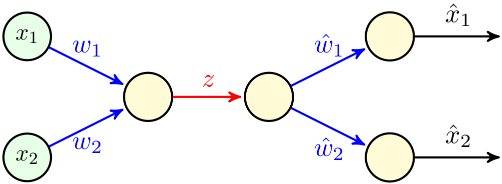


To train this model, we should make some labels

- *We intend to recover* $\mathbf{x}$ *from* z *at the end of the day*
  - ↳ *We can learn further the decompression*

PCA as *Neural Network*

+ *But how can we train this NN? We do not have any label!*
– This is because it's an *unsupervised learning problem*

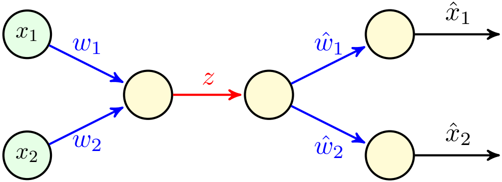


To train this model, we should make some labels

- *We intend to recover* $\mathbf{x}$ *from* z *at the end of the day*
  ↳ *We can learn further the decompression*
- *We now have some labels*
  ↳ *We train using the loss* $\hat{R} = \mathcal{L}(\mathbf{x}; \hat{\mathbf{x}})$

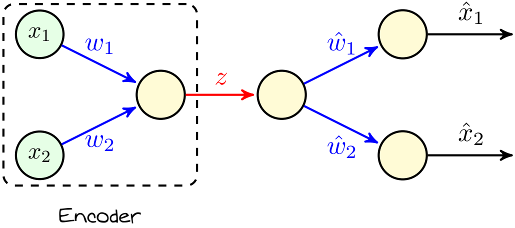
PCA as *Neural Network*

+ *This looks like something we had before!*
– Yes! It's an *encoder-decoder architecture whose label is the input*



PCA as *Neural Network*

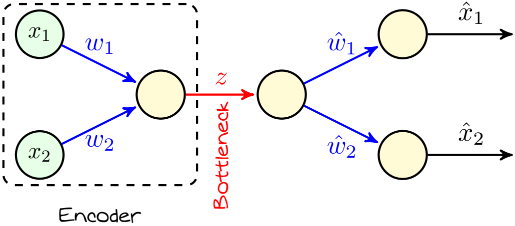
+ *This looks like something we had before!*
– Yes! It's an *encoder-decoder architecture whose label is the input*



- *Encoder represents input in a lower dimension*
  ↳ *It somehow compresses the input*

PCA as *Neural Network*

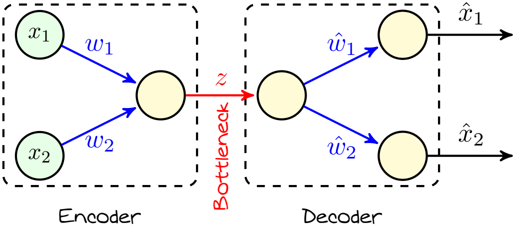
+ *This looks like something we had before!*
– Yes! It's an *encoder-decoder architecture whose label is the input*



- *Encoder represents input in a lower dimension*
  ↳ *It somehow compresses the input*
- *Bottleneck contains the latent variables*

PCA as *Neural Network*

+ *This looks like something we had before!*
– Yes! It's an *encoder-decoder architecture whose label is the input*



- *Encoder represents input in a lower dimension*
  ↳ *It somehow compresses the input*
- *Bottleneck contains the latent variables*
- *Decoder can return back the data from its lo-dimensional representation*
  ↳ *It decompresses the latent variables*

Nonlinear Compression

Encoder and decoder are both *shallow* and *linear* in PCA: *we may however need deep nonlinear encoder and decoder*

Nonlinear Compression

Encoder and decoder are both *shallow* and *linear* in PCA: *we may however need deep nonlinear encoder and decoder*

Let's consider an example: *consider our initial simple example and now assume that data-points are of the form*

$$\mathbf{x} = \begin{bmatrix} \sin\theta \\ \cos\theta \end{bmatrix}$$

Obviously PCA fails to compress $\mathbf{x}$ *error-less into one dimension*

Nonlinear Compression

Encoder and decoder are both *shallow* and *linear* in PCA: *we may however need deep nonlinear encoder and decoder*

Let's consider an example: *consider our initial simple example and now assume that data-points are of the form*

$$\mathbf{x} = \begin{bmatrix} \sin\theta \\ \cos\theta \end{bmatrix}$$

Obviously PCA fails to compress $\mathbf{x}$ error-less into one dimension

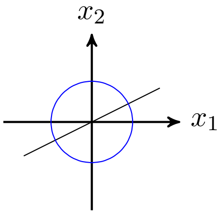
- *PCA searches for the line that represent the dataset*
  - ↳ *In general, it finds the line with minimum error*

Nonlinear Compression

Encoder and decoder are both *shallow* and *linear* in PCA: *we may however need deep nonlinear encoder and decoder*

Let's consider an example: *consider our initial simple example and now assume that data-points are of the form*

$$\mathbf{x} = \begin{bmatrix} \sin\theta \\ \cos\theta \end{bmatrix}$$



Obviously PCA fails to compress $\mathbf{x}$ error-less into one dimension

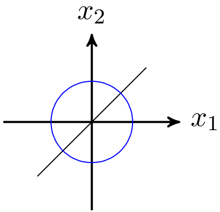
- *PCA searches for the line that represent the dataset*
  ↳ *In general, it finds the line with minimum error*
- *Such a line does not exists in this problem*

Nonlinear Compression

Encoder and decoder are both *shallow* and *linear* in PCA: *we may however need deep nonlinear encoder and decoder*

Let's consider an example: *consider our initial simple example and now assume that data-points are of the form*

$$\mathbf{x} = \begin{bmatrix} \sin\theta \\ \cos\theta \end{bmatrix}$$



Obviously PCA fails to compress $\mathbf{x}$ *error-less into one dimension*

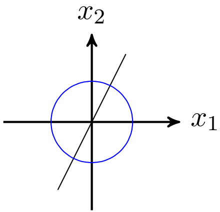
- *PCA searches for the line that represent the dataset*
  - ↳ *In general, it finds the line with minimum error*
- *Such a line does not exists in this problem*

Nonlinear Compression

Encoder and decoder are both *shallow* and *linear* in PCA: *we may however need deep nonlinear encoder and decoder*

Let's consider an example: *consider our initial simple example and now assume that data-points are of the form*

$$\mathbf{x} = \begin{bmatrix} \sin\theta \\ \cos\theta \end{bmatrix}$$



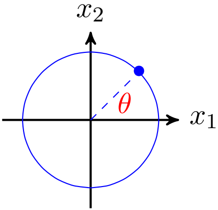
Obviously PCA fails to compress $\mathbf{x}$ *error-less into one dimension*

- *PCA searches for the line that represent the dataset*
  - ↳ *In general, it finds the line with minimum error*
- *Such a line does not exists in this problem*

Nonlinear Compression

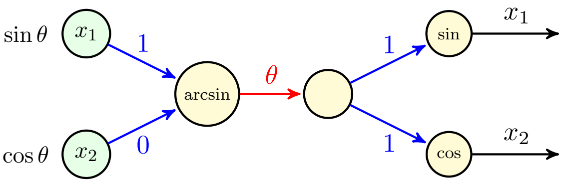
We can however do this by a nonlinear representation

We just need to know the angle θ

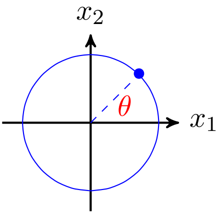


Nonlinear Compression

We can however do this by a nonlinear representation

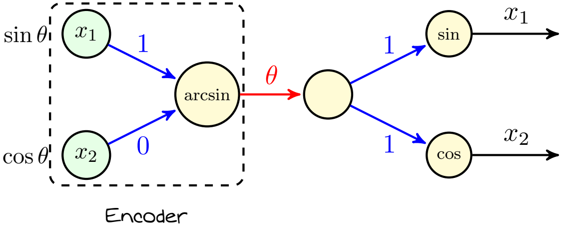


We just need to know the angle θ

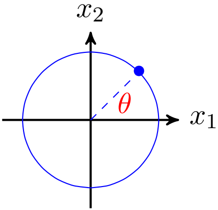


Nonlinear Compression

We can however do this by a nonlinear representation

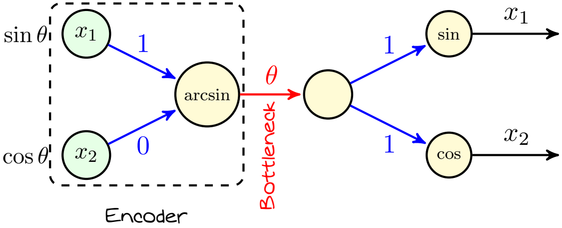


We just need to know the angle θ

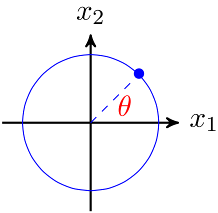


Nonlinear Compression

We can however do this by a nonlinear representation

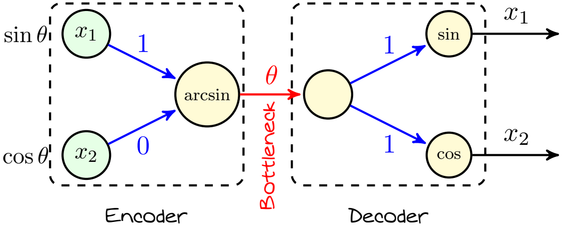


We just need to know the angle θ

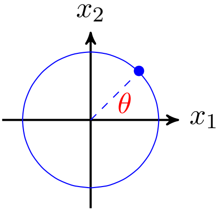


Nonlinear Compression

We can however do this by a nonlinear representation

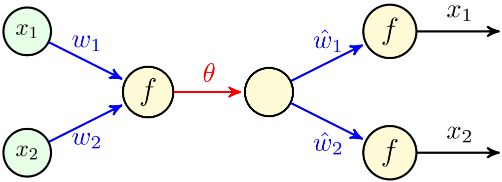


We just need to know the angle θ

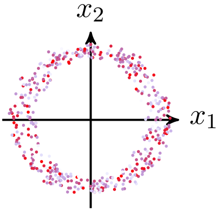


Nonlinear PCA

We can use this NN to extract principle components of other datasets

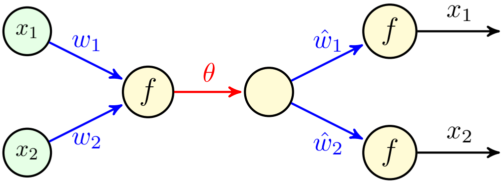


Latent variable θ represents data in lower dimension for minimal recovery error

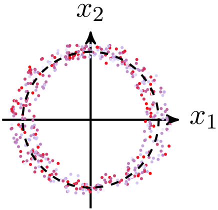


Nonlinear PCA

We can use this NN to extract principle components of other datasets

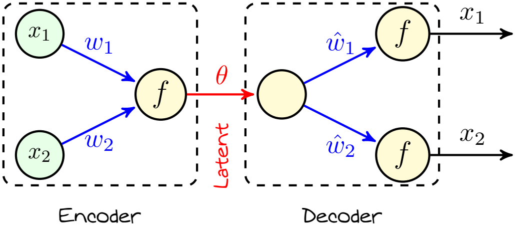


Latent variable θ represents data in lower dimension for minimal recovery error

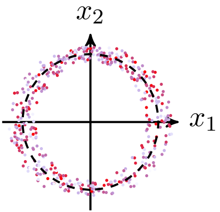


Nonlinear PCA

We can use this NN to extract principle components of other datasets

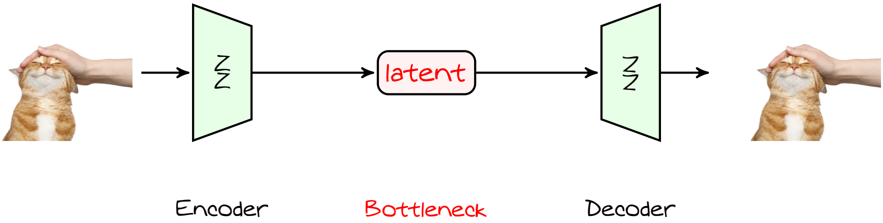


Latent variable θ represents data in lower dimension for minimal recovery error



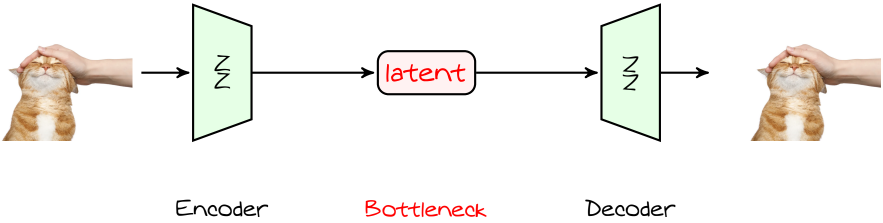
Auto Encoder: *Deep PCA with Encoder-Decoder*

AE is in principle a deep encoder-decoder architecture used for nonlinear PCA



Auto Encoder: *Deep PCA with Encoder-Decoder*

AE is in principle a deep encoder-decoder architecture used for nonlinear PCA



Vanilla AE finds a latent space that is very smaller in dimension

- *Each data-point is encoded to its low-dimensional latent representation*
  - ↳ *Latent representation contains in fact the principle features of data*
- *Latent representation can be re-generate the data-point via the decoder*

Vanilla AE

Auto Encoder (AE)

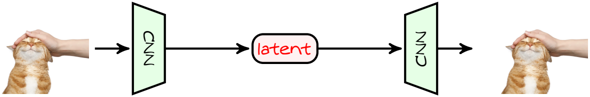
AE is an encoder-decoder architecture in which bottleneck features, also called latent representation, have lower dimension than the input and output of NN

Vanilla AE

Auto Encoder (AE)

AE is an encoder-decoder architecture in which bottleneck features, also called latent representation, have lower dimension than the input and output of NN

We can implement encoder and decoder by simple NNs, e.g.,



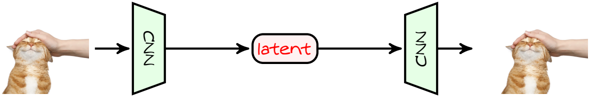
Such an architecture is a vanilla AE mainly used for compression

Vanilla AE

Auto Encoder (AE)

AE is an encoder-decoder architecture in which bottleneck features, also called latent representation, have lower dimension than the input and output of NN

We can implement encoder and decoder by simple NNs, e.g.,



Such an architecture is a vanilla AE mainly used for compression

---

+ *Is compression so crucial that AEs become so important?*
- Naive answer: Yes! Better answer: *AEs can do much more than compression in fact!*

Training AEs: *General Approach*

We typically use AEs in unsupervised settings: *it means that we have no labels in the dataset*

- *In AE both latent representation and decoded data are outputs*

Training AEs: *General Approach*

We typically use AEs in unsupervised settings: *it means that we have no labels in the dataset*

- *In AE both latent representation and decoded data are outputs*
- *For training we need to compute loss between the outputs and a reference*
  - ↳ *We cannot compare the latent representation with any reference*
    - ↳ *We have no true latent representation*

Training AEs: *General Approach*

We typically use AEs in unsupervised settings: *it means that we have no labels in the dataset*

- *In AE both latent representation and decoded data are outputs*
- *For training we need to compute loss between the outputs and a reference*
  - ↳ *We cannot compare the latent representation with any reference*
    - ↳ *We have no true latent representation*
- *We should extract some reference from our dataset*
  - ↳ *This reference depends on our target application*
  - ↳ *We are going to consider three types in this chapter*

Training AEs: *General Approach*

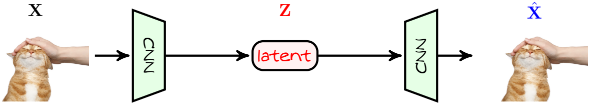
We typically use AEs in unsupervised settings: *it means that we have no labels in the dataset*

- *In AE both latent representation and decoded data are outputs*
- *For training we need to compute loss between the outputs and a reference*
  - ↳ *We cannot compare the latent representation with any reference*
    - ↳ *We have no true latent representation*
- *We should extract some reference from our dataset*
  - ↳ *This reference depends on our target application*
  - ↳ *We are going to consider three types in this chapter*

To go on with the training of AEs, let's keep the track of their applications

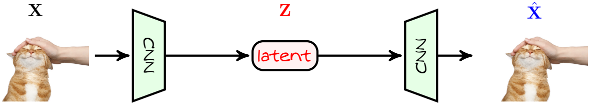
1. *Compression*
   - ↳ *We intend to compress data into a lower-dimensional subspace*
   - ↳ *For loss computation, we compare decoded data with its ground truth*

Training AEs *for Compression*



Let's name variables: *say the input is* $\mathbf{X}$*, e.g., RGB image, latent representation is* $\mathbf{Z}$*, e.g., multi-channel tensor, and* $\hat{\mathbf{X}}$ *is decoded output, e.g., RGB image*

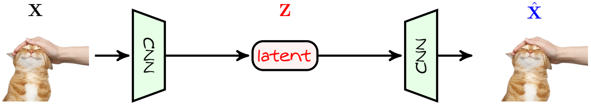
Training AEs *for Compression*



Let's name variables: *say the input is* $\mathbf{X}$*, e.g., RGB image, latent representation is* $\mathbf{Z}$*, e.g., multi-channel tensor, and* $\hat{\mathbf{X}}$ *is decoded output, e.g., RGB image*

- *For compression we wish to recover* $\hat{\mathbf{X}} = \mathbf{X}$
  - ↳ *Loss is proportional to the difference between* $\mathbf{X}$ *and* $\hat{\mathbf{X}}$

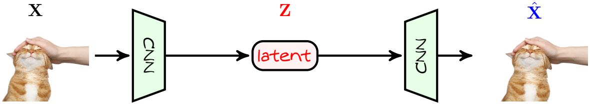
Training AEs *for Compression*



Let's name variables: *say the input is* $\mathbf{X}$, *e.g., RGB image, latent representation is* $\mathbf{Z}$, *e.g., multi-channel tensor, and* $\hat{\mathbf{X}}$ *is decoded output, e.g., RGB image*

- *For compression we wish to recover* $\hat{\mathbf{X}} = \mathbf{X}$
  - ↳ *Loss is proportional to the difference between* $\mathbf{X}$ *and* $\hat{\mathbf{X}}$
- *We are indifferent about the behavior of latent representation*
  - ↳ *We do not need to include* $\mathbf{Z}$ *directly in loss computation*
  - ↳ $\mathbf{Z}$ *contributes to loss indirectly through* $\hat{\mathbf{X}}$

Training AEs *for Compression*



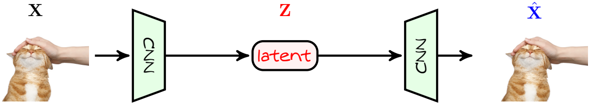
Let's name variables: *say the input is* $\mathbf{X}$, *e.g., RGB image, latent representation is* $\mathbf{Z}$, *e.g., multi-channel tensor, and* $\hat{\mathbf{X}}$ *is decoded output, e.g., RGB image*

- *For compression we wish to recover* $\hat{\mathbf{X}} = \mathbf{X}$
  - ↳ *Loss is proportional to the difference between* $\mathbf{X}$ *and* $\hat{\mathbf{X}}$
- *We are indifferent about the behavior of latent representation*
  - ↳ *We do not need to include* $\mathbf{Z}$ *directly in loss computation*
  - ↳ $\mathbf{Z}$ *contributes to loss indirectly through* $\hat{\mathbf{X}}$

So, the loss in this case is compute as

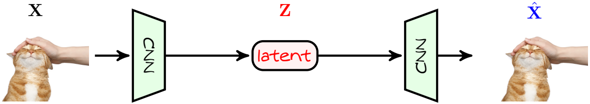
$$\hat{R} = \mathcal{L}(\hat{\mathbf{X}}, \mathbf{X})$$

Training AEs *for Compression*



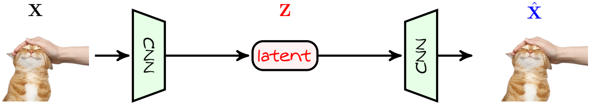
We know the loss:

Training AEs *for Compression*



We know the loss: *we can analytically compute* $\nabla_{\hat{\mathbf{x}}} \hat{R}$, *so training is done by standard forward and backward pass*

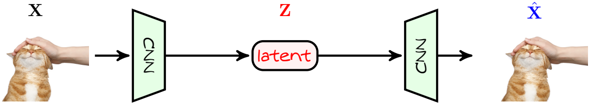
Training AEs *for Compression*



We know the loss: *we can analytically compute* $\nabla_{\hat{\mathbf{x}}} \hat{R}$*, so training is done by standard forward and backward pass*

1. *Pass* $\mathbf{X}$ *forward through the encoder*
   ↳ *Compute output of all layer as well as* $\mathbf{Z}$
2. *Pass* $\mathbf{Z}$ *forward through the decoder*
   ↳ *Compute output of all layer as well as* $\hat{\mathbf{X}}$

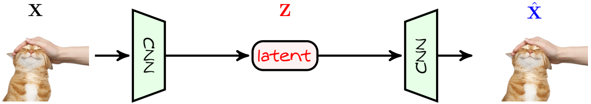
Training AEs *for Compression*



We know the loss: *we can analytically compute $\nabla_{\hat{\mathbf{x}}} \hat{R}$, so training is done by standard forward and backward pass*

1. *Pass $\mathbf{X}$ forward through the encoder*
   - ↳ *Compute output of all layer as well as $\mathbf{Z}$*
2. *Pass $\mathbf{Z}$ forward through the decoder*
   - ↳ *Compute output of all layer as well as $\hat{\mathbf{X}}$*
3. *Compute $\nabla_{\hat{\mathbf{x}}} \hat{R}$ and backpropagate through decoder*

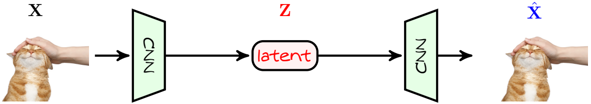
Training AEs *for Compression*



We know the loss: *we can analytically compute $\nabla_{\hat{\mathbf{x}}} \hat{R}$, so training is done by standard forward and backward pass*

1. *Pass $\mathbf{X}$ forward through the encoder*
   - ↳ *Compute output of all layer as well as $\mathbf{Z}$*
2. *Pass $\mathbf{Z}$ forward through the decoder*
   - ↳ *Compute output of all layer as well as $\hat{\mathbf{X}}$*
3. *Compute $\nabla_{\hat{\mathbf{X}}} \hat{R}$ and backpropagate through decoder*
4. *Compute $\nabla_{\mathbf{Z}} \hat{R}$ from the gradient at the first layer of decoder*
5. *Starting from $\nabla_{\mathbf{Z}} \hat{R}$ backpropagate through encoder*

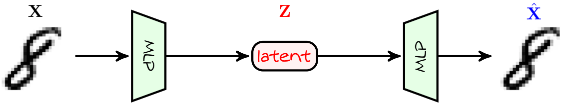
Training AEs *for Compression*



We know the loss: *we can analytically compute $\nabla_{\hat{\mathbf{x}}} \hat{R}$, so training is done by standard forward and backward pass*

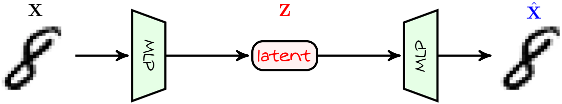
1. *Pass $\mathbf{X}$ forward through the encoder*
   ↳ *Compute output of all layer as well as $\mathbf{Z}$*
2. *Pass $\mathbf{Z}$ forward through the decoder*
   ↳ *Compute output of all layer as well as $\hat{\mathbf{X}}$*
3. *Compute $\nabla_{\hat{\mathbf{x}}} \hat{R}$ and backpropagate through decoder*
4. *Compute $\nabla_{\mathbf{z}} \hat{R}$ from the gradient at the first layer of decoder*
5. *Starting from $\nabla_{\mathbf{z}} \hat{R}$ backpropagate through encoder*
6. *Update all weights and go for the next round*

Example: *Compressing MNIST*



A simple practice can be done on MNIST:

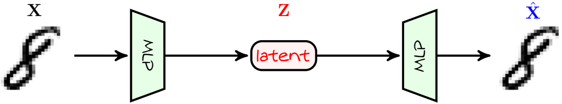
Example: *Compressing MNIST*



A simple practice can be done on MNIST: *we try to represent MNIST images in a 2-dimensional latent space. For encoding we use the following MLP*

1. *It has four hidden layer*
   - ↳ *The widths of layers gradually reduce*
   - ↳ *The last layer has only two outputs*
2. *All neurons are activated via sigmoid*
3. *We do not use any dropout or batch-normalization*

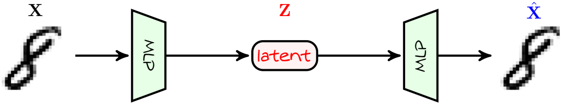
Example: *Compressing MNIST*



For decoding we use another MLP to invert the encoder

1. *The decoder has four hidden layer*
   - ↳ *The widths of layers gradually increase*
   - ↳ *The last layer has 784 neurons*
2. *All neurons are activated via sigmoid*
3. *We finally sort the output into a 28 $\times$ 28 matrix*

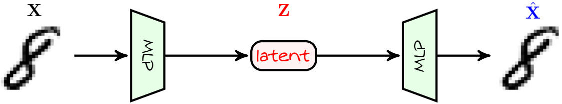
Example: *Compressing MNIST*



Training then follows the standard approach: this is in fact an 8-layer MLP

1. *Pass each training image forward through all layers*

Example: *Compressing MNIST*

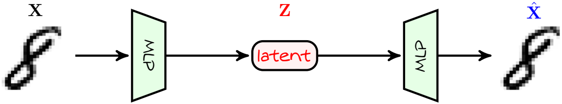


Training then follows the standard approach: this is in fact an 8-layer MLP

1. *Pass each training image forward through all layers*
2. *Compute loss between the output and true image, e.g.,*

$$\mathcal{L}\left(\hat{\mathbf{X}}, \mathbf{X}\right) = \|\hat{\mathbf{X}} - \mathbf{X}\|^2$$

Example: *Compressing MNIST*



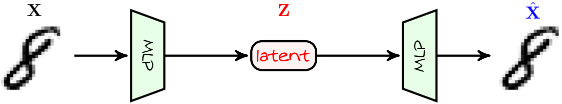
Training then follows the standard approach: this is in fact an 8-layer MLP

1. *Pass each training image forward through all layers*
2. *Compute loss between the output and true image, e.g.,*

$$\mathcal{L}\left(\hat{\mathbf{X}}, \mathbf{X}\right) = \|\hat{\mathbf{X}} - \mathbf{X}\|^2$$

3. *Compute* $\nabla_{\hat{\mathbf{X}}} \hat{R}$ *and backpropagate*

Example: *Compressing MNIST*



Training then follows the standard approach: this is in fact an 8-layer MLP

1. *Pass each training image forward through all layers*
2. *Compute loss between the output and true image, e.g.,*

$$\mathcal{L}\left(\hat{\mathbf{x}}, \mathbf{x}\right) = \|\hat{\mathbf{x}} - \mathbf{x}\|^2$$

3. *Compute* $\nabla_{\hat{\mathbf{x}}} \hat{R}$ *and backpropagate*
4. *Update all weights and go for the next round*

Example: *Compressing MNIST*

We can then test our AE

1. *Pass a test image forward through encoder*
2. *Compute latent representation*

Example: *Compressing MNIST*

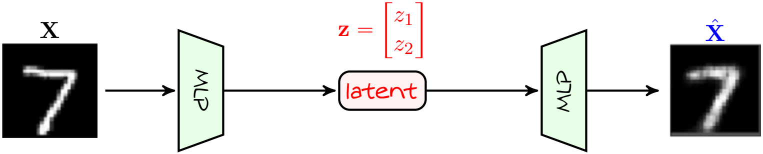
We can then test our AE

1. *Pass a test image forward through encoder*
2. *Compute latent representation*
3. *Pass the latent representation forward through decoder*
4. *Compare the images*

Example: *Compressing MNIST*

We can then test our AE

1. *Pass a test image forward through encoder*
2. *Compute latent representation*
3. *Pass the latent representation forward through decoder*
4. *Compare the images*



Obvious Compression via Vanilla AE

For compression it is important that we set

the latent representation to be of smaller dimension than input

Obvious Compression via Vanilla AE

For compression it is important that we set

the latent representation to be of smaller dimension than input

If we set it larger or equal to the input size, we end up with an obvious solution

$$\text{Decoder}\left(\text{Encoder}\left(\cdot\right)\right) = \text{Identity}\left(\cdot\right)$$

Obvious Compression via Vanilla AE

For compression it is important that we set

the latent representation to be of smaller dimension than input

If we set it larger or equal to the input size, we end up with an obvious solution

$$\text{Decoder}\left(\text{Encoder}\left(\cdot\right)\right) = \text{Identity}\left(\cdot\right)$$

- *We want to recover the original data after decoding*
  - ↳ *Identity is always an obvious solution*
- *With larger latent space we cal always realize identity*
  - ↳ *We simply set* $\mathbf{Z} = \mathbf{X}$ *and* $\hat{\mathbf{X}} = \mathbf{Z}$
- *This is however useless since we do not compress*

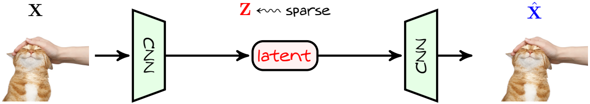
Sparse AEs

Let's keep the track of their applications

1. *Compression*
2. *Finding a sparse representation of data*
   - ↳ *We intend to represent data with a sparse object*
     - ↳ *for instance, we intend to represent input $\mathbf{x} \in \mathbb{R}^{100}$ with another 100-dimensional vector whose most of entries are zero*
     - ↳ *we may want to further compress, i.e., represent $\mathbf{x} \in \mathbb{R}^{100}$ with an 80-dimensional sparse vector*
   - ↳ *For loss computation, we should also take a look at the latent representation*
     - ↳ *we want the latent representation to be sparse*
     - ↳ *in vanilla AE there is no guarantee that this happens*

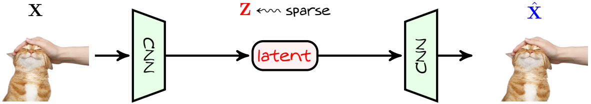
For such application we use sparse AEs

Training AEs *for Sparse Representation*



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

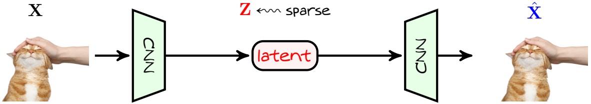
Training AEs *for Sparse Representation*



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

- *We still 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 $\hat{\mathbf{X}}$*

Training AEs *for Sparse Representation*



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

- *We still 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 also want to have sparse latent representation*
  - ↳ $\mathbf{Z}$ *should contribute directly to loss*
  - ↳ *Loss should also be proportional to the sparsity of* $\mathbf{Z}$

Training Sparse AEs

Loss is proportional to difference between $\mathbf{X}$ *and* $\hat{\mathbf{X}}$, *and sparsity of* $\mathbf{Z}$

Training Sparse AEs

Loss is proportional to difference between $\mathbf{X}$ and $\hat{\mathbf{X}}$, and sparsity of $\mathbf{Z}$

So, the loss in this case should be

$$\hat{R} = \mathcal{L}(\hat{\mathbf{X}}, \mathbf{X}) +$$

Training Sparse AEs

Loss is proportional to difference between $\mathbf{X}$ and $\hat{\mathbf{X}}$, and sparsity of $\mathbf{Z}$

So, the loss in this case should be

$$\hat{R} = \mathcal{L}(\hat{\mathbf{X}}, \mathbf{X}) + \lambda S(\mathbf{Z})$$

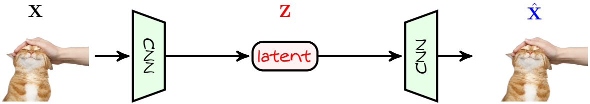
for some function $S(\cdot)$ that is proportional to sparsity, i.e.,

if $\mathbf{Z}$ has less zeros $\rightsquigarrow$ $S(\mathbf{Z})$ should increase

and regularizer λ that is a hyperparameter

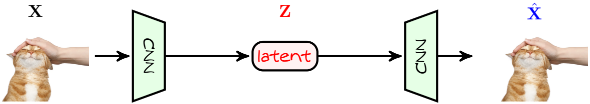
- $S(\mathbf{Z}) = \|\mathbf{Z}\|_0 \rightsquigarrow$ *non-differentiable* ✗
- $S(\mathbf{Z}) = \|\mathbf{Z}\|_1 \rightsquigarrow$ *convex* ✓
- $S(\mathbf{Z}) = \mathrm{KL}(p_{\mathbf{Z}} \| \mathrm{Ber}_\rho) \rightsquigarrow$ *convex* ✓
  - ↳ Ber_ρ *is a Bernoulli distribution with probability of zero being* ρ
  - ↳ $p_{\mathbf{Z}}$ *is the empirical distribution of the support of* $\mathbf{Z}$

Training Sparse AEs



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

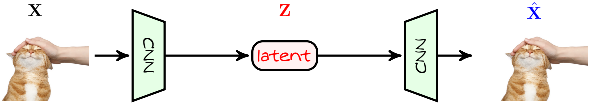
Training Sparse AEs



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

- *Pass forward* $\mathbf{X}$ *through encoder and decoder*

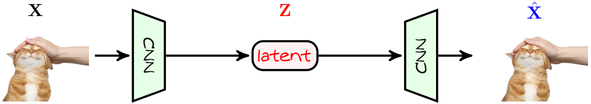
Training Sparse AEs



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 bottleneck*

Training Sparse AEs

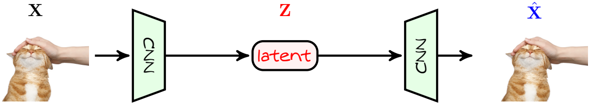


Let's see how training looks: *say we are training with single sample* $\mathbf{X}$

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

$$\nabla_{\mathbf{Z}} \hat{R} = \underbrace{\nabla_{\hat{\mathbf{X}}} \hat{R} \circ \nabla_{\mathbf{Z}} \hat{\mathbf{X}}}_{\text{computed by backpropagation}} + \lambda \nabla_{\mathbf{Z}} S\left(\mathbf{Z}\right)$$

Training Sparse AEs



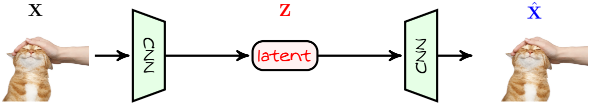
Let's see how training looks: *say we are training with single sample* $\mathbf{X}$

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

$$\nabla_{\mathbf{Z}} \hat{R} = \underbrace{\nabla_{\hat{\mathbf{X}}} \hat{R} \circ \nabla_{\mathbf{Z}} \hat{\mathbf{X}}}_{\text{computed by backpropagation}} + \lambda \nabla_{\mathbf{Z}} S(\mathbf{Z})$$

- ↳ *Start from* $\nabla_{\mathbf{Z}} \hat{R}$ *and backpropagate till input*

Training Sparse AEs



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 bottleneck*
  - ↳ *At the bottleneck, we need to compute* $\nabla_{\mathbf{Z}} \hat{R}$

$$\nabla_{\mathbf{Z}} \hat{R} = \underbrace{\nabla_{\hat{\mathbf{X}}} \hat{R} \circ \nabla_{\mathbf{Z}} \hat{\mathbf{X}}}_{\text{computed by backpropagation}} + \lambda \nabla_{\mathbf{Z}} S\left(\mathbf{Z}\right)$$

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

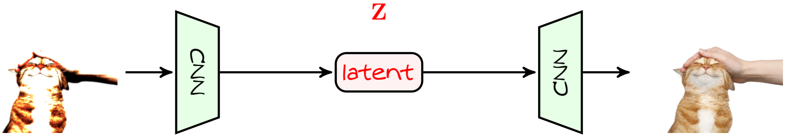
Using AE for Noise Removal

Let's keep the track of their applications

1. *Compression*
2. *Finding a sparse representation of data*
3. *Denoising*
   - ↳ *We intend to find a representation that can refine a noisy dataset*
     - ↳ *for instance we want to remove background noise from an image*
     - ↳ *for instance we want to increase the resolution of an image*
     - ↳ *for instance we want to color a gray image*
   - ↳ *For loss computation, we should*
     - ↳ *be able the recover the refined data at the decoder*
     - ↳ *unlike vanilla AE we can start with distorted data*

We call such AE architecture a denoising AE

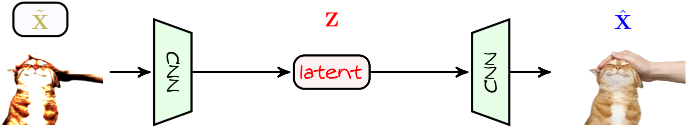
Training Denoising AEs



We can train a denoising AE using *degraded samples*

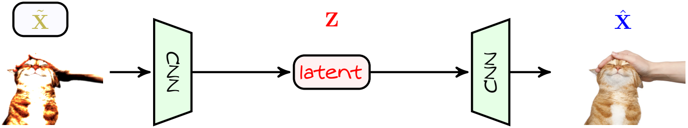
- *For each sample we generate its degraded counterpart, e.g.,*
  ↳ *for each image we also produce a noisy, or low resolution or gray version*
- *We give this noisy version to the encoder*
- *We set loss to compute difference between original samples and output*
  ↳ *the decoded image and the original RGB image in dataset*

Training Denoising AEs



Let's formulate the problem: *say the sample is* $\mathbf{X}$, *and its corrupted version is* $\tilde{\mathbf{X}}$. *Also, denote latent representation by* $\mathbf{Z}$ *and decoded output by* $\hat{\mathbf{X}}$

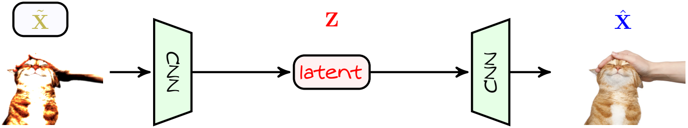
Training Denoising AEs



Let's formulate the problem: *say the sample is $\mathbf{X}$, and its corrupted version is $\tilde{\mathbf{X}}$. Also, denote latent representation by $\mathbf{Z}$ and decoded output by $\hat{\mathbf{X}}$*

- *We want to recover original data from latent representation, i.e., $\hat{\mathbf{X}} = \mathbf{X}$*
  - ↳ *Loss is proportional to the difference between $\mathbf{X}$ and $\hat{\mathbf{X}}$*

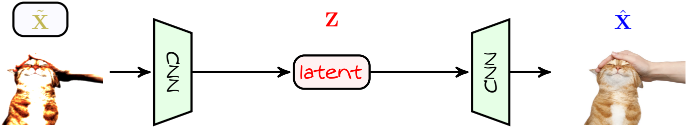
Training Denoising AEs



Let's formulate the problem: *say the sample is* $\mathbf{X}$, *and its corrupted version is* $\tilde{\mathbf{X}}$. *Also, denote latent representation by* $\mathbf{Z}$ *and decoded output by* $\hat{\mathbf{X}}$

- *We want to recover original data from latent representation, i.e.,* $\hat{\mathbf{X}} = \mathbf{X}$
  - ↳ *Loss is proportional to the difference between* $\mathbf{X}$ *and* $\hat{\mathbf{X}}$
- *We may want our representation to be sparse*
  - ↳ *We could add a penalty proportional to sparsity of* $\mathbf{Z}$

Training Denoising AEs



Let's formulate the problem: *say the sample is* $\mathbf{X}$, *and its corrupted version is* $\tilde{\mathbf{X}}$. *Also, denote latent representation by* $\mathbf{Z}$ *and decoded output by* $\hat{\mathbf{X}}$

- *We want to recover original data from latent representation, i.e.,* $\hat{\mathbf{X}} = \mathbf{X}$
  ↳ *Loss is proportional to the difference between* $\mathbf{X}$ *and* $\hat{\mathbf{X}}$
- *We may want our representation to be sparse*
  ↳ *We could add a penalty proportional to sparsity of* $\mathbf{Z}$

So, we set the loss to

$$\hat{R} = \mathcal{L}(\hat{\mathbf{X}}, \mathbf{X}) + \lambda S(\mathbf{Z})$$

Training AEs: *Summary*

We could have various form of AEs depending on the target application

- *Vanilla* AEs
  - ↳ *Encoder-decoder with both input and label being data*
  - ↳ *Loss computes difference between input and output* $\equiv$ *recovery error*
  - ↳ *We can use it for compression*

Training AEs: *Summary*

We could have various form of AEs depending on the target application

- *Vanilla* AEs
  - ↳ *Encoder-decoder with both input and label being data*
  - ↳ *Loss computes difference between input and output* $\equiv$ *recovery error*
  - ↳ *We can use it for compression*
- *Sparse* AEs
  - ↳ *Encoder-decoder with both input and label being data*
  - ↳ *Loss computes recovery error plus a sparsity penalty*
  - ↳ *We can use it for sparse representation of data*

Training AEs: *Summary*

We could have various form of AEs depending on the target application

- *Vanilla* AEs
  - ↳ *Encoder-decoder with both input and label being data*
  - ↳ *Loss computes difference between input and output* $\equiv$ *recovery error*
  - ↳ *We can use it for compression*
- *Sparse* AEs
  - ↳ *Encoder-decoder with both input and label being data*
  - ↳ *Loss computes recovery error plus a sparsity penalty*
  - ↳ *We can use it for sparse representation of data*
- *Denoising* AEs
  - ↳ *Encoder-decoder with input being noisy data and label being data*
  - ↳ *Loss computes recovery error*
  - ↳ *We can also regularize with a sparsity penalty if we need sparse latent*
  - ↳ *We can use it for noise removal, resolution increasing and other similar applications*

Generating New Data via AEs

Let's keep the track of their applications

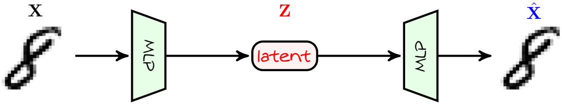
1. *Compression*
2. *Finding a sparse representation of data*
3. *Denoising*
4. *Data Generation*
   - ↳ *We intend to generate a new sample by our decoder from a seed*
     - ↳ *for instance we generate a random seed and give it to decoder*
     - ↳ *the decoder returns an image which was not in the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

\+ *That sounds crazy!*

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

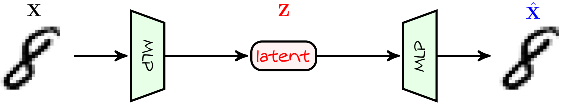
Looking into *Latent Space*

Let's get back to out 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*

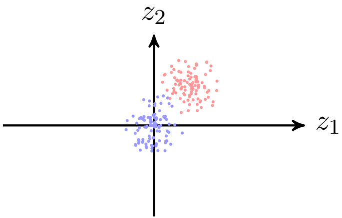


Looking into *Latent Space*

Let's get back to out 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*

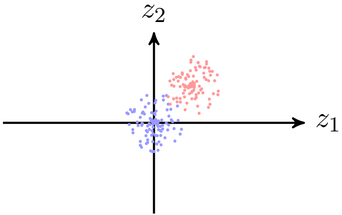


We now do a simple experiment: we pass all images of 1 and 8 that we have and mark their latent representations with blue and red



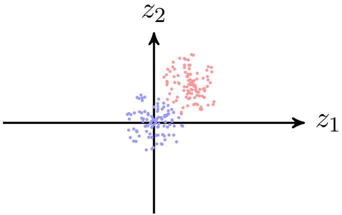
Looking into *Latent Space*

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



Looking into *Latent Space*

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



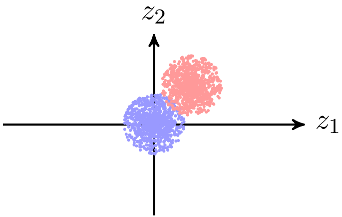
Recall: Data Space and Distribution

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!

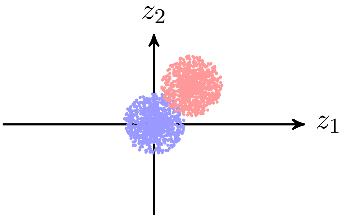
Looking into *Latent Space*

+ *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!



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

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*

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*

First Try for Generating Data

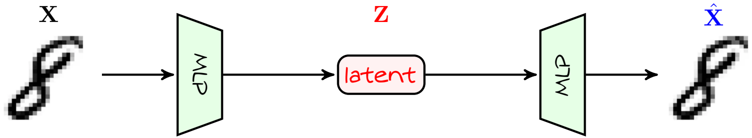
We could use this behavior to generate a new data

- *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*

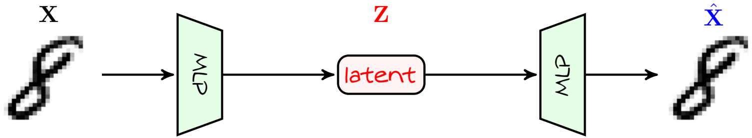
First Try for Generating Data

We first train

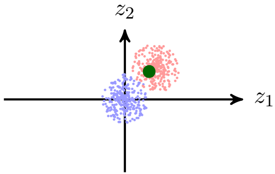


First Try for Generating Data

We first train

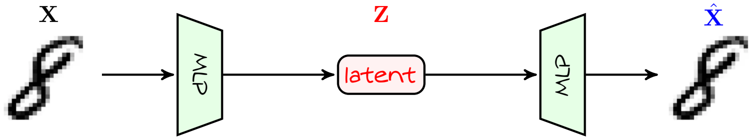


We then sample the latent space

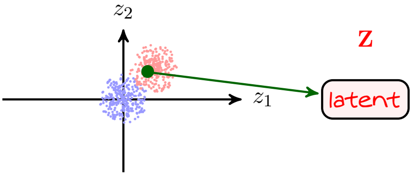


First Try for Generating Data

We first train

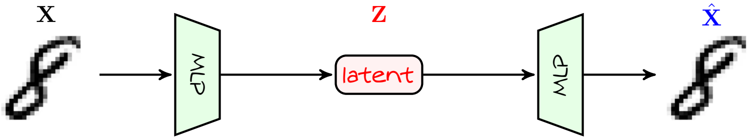


We then sample the latent space

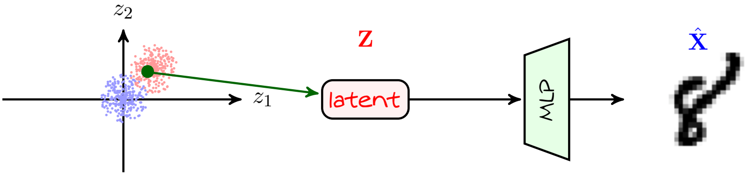


First Try for Generating Data

We first train



We then sample the latent space



Drawbacks of Generation via *Vanilla AEs*

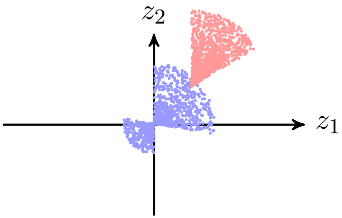
Even though the idea seems to be intuitive: *it turns out that it does not work very well when we use basic AE architectures*

- *Frequency of invalid generated data is quite high*
  ↳ *For instance, the decoder returns an image which is not 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*



So, 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 compress version of invalid image*
- *decoder returns an invalid data-point!*

+ *How can we resolve this issue?*
– *We may restrict encoder to compress into compact and symmetric region*

Generating via Variational AEs

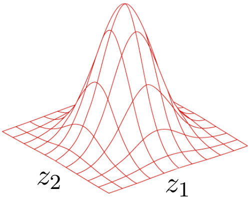
Variational AEs apply some trick to *make sure that*

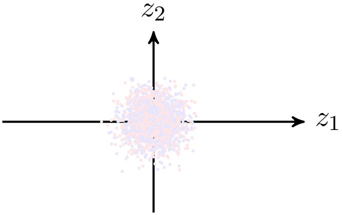
latent representation look like samples of a Gaussian distribution

Generating via Variational AEs

Variational AEs apply some trick to *make sure that*

latent representation look like samples of a Gaussian distribution



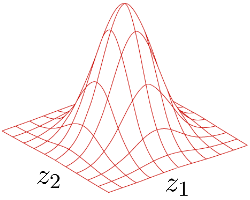


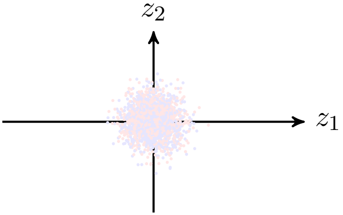
Specifically, a Gaussian distribution with mean zero and variance one: $\mathcal{N}(\mathbf{0}, \mathbf{1})$

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: $\mathcal{N}\left(\mathbf{0}, \mathbf{1}\right)$

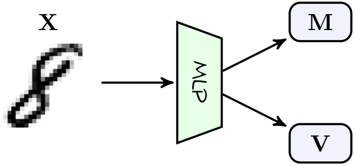
+ *How can we do it?*
– Well! The trick is quite sophisticated!

Variational AE: *Architecture*

$\mathbf{X}$

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

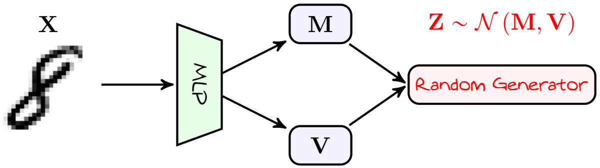
Variational AE: *Architecture*



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*

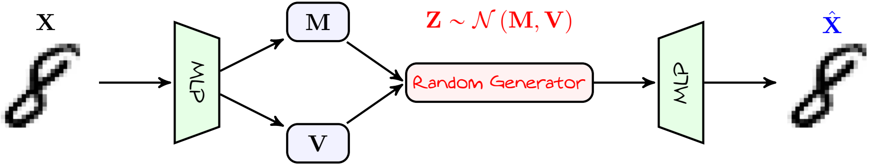
Variational AE: *Architecture*



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* $\mathbf{V}$

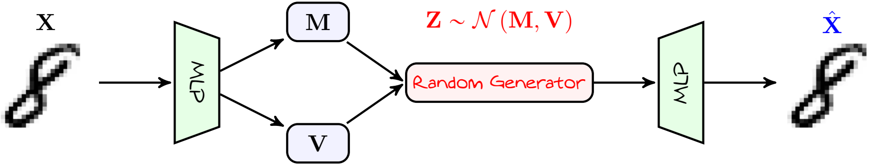
Variational AE: *Architecture*



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*

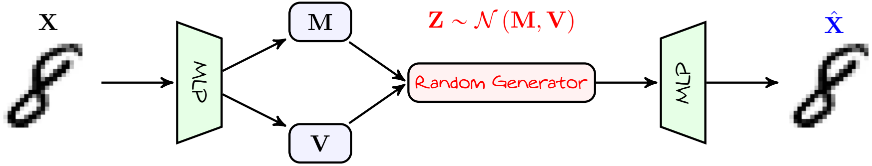
Variational AE: *Architecture*



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

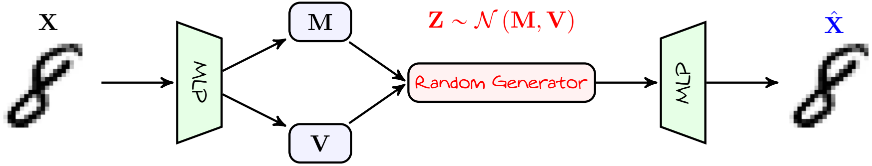
- *We start by encoding: encoder gets* $\mathbf{X}$ *and returns*
  - ↳ $\mathbf{M}$ *which is of same shape as* $\mathbf{Z}$*: this plays the role of mean*
  - ↳ $\mathbf{V}$ *which is of same shape as* $\mathbf{Z}$*: this plays the role of variance*
- *We also then generate latent representations at 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

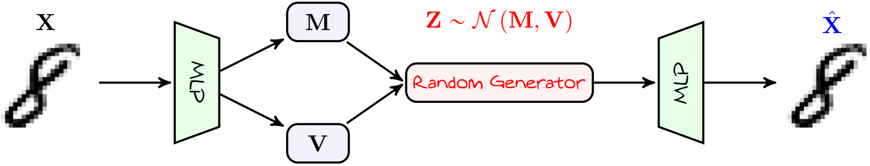
Variational AE: *Loss*



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* $\hat{\mathbf{X}}$

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 to have 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 distribution of $\mathbf{Z}$ $\equiv$ *let's call them* $p_{\mathbf{Z}}$ *and* $q_{\mathbf{Z}}$*, respectively*

$$\hat{R} = \mathcal{L}(\hat{\mathbf{X}}, \mathbf{X}) +$$

Loss in VAEs

Loss is proportional to recovery error and difference between actual and distribution 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* $\mathrm{Div}\left(p_{\mathbf{Z}}, q_{\mathbf{Z}}\right)$

Loss in VAEs

Loss is proportional to recovery error and difference between actual and distribution 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*

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

Loss in VAEs

Loss is proportional to recovery error and difference between actual and distribution 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*

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

Loss in VAEs

Loss is proportional to recovery error and difference between actual and distribution 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* $\mathrm{Div}\left(p_{\mathbf{Z}}, q_{\mathbf{Z}}\right)$

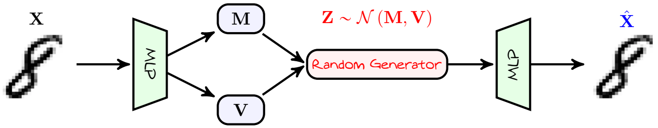
The classical choice for $\mathrm{Div}\left(p_{\mathbf{Z}}, q_{\mathbf{Z}}\right)$ *is the KL-divergence*

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

So, we basically train by minimizing

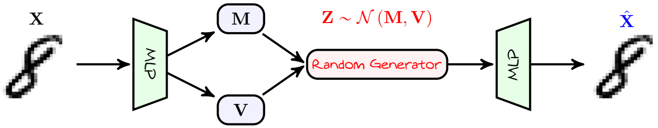
$$\hat{R} = \mathcal{L}(\hat{\mathbf{X}}, \mathbf{X}) + \lambda F\left(\mathbf{M}, \mathbf{V}\right)$$

Training VAEs



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

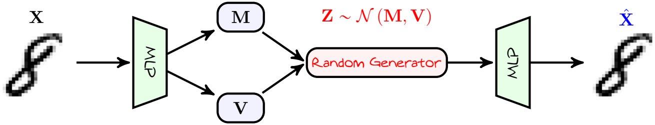
Training VAEs



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

- *Pass forward* $\mathbf{X}$ *through encoder and decoder*

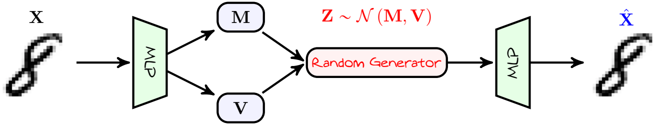
Training VAEs



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*

Training VAEs

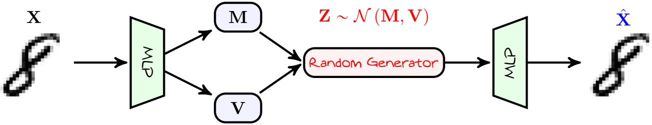


Let's see how training looks: *say we are training with single sample* $\mathbf{X}$

- *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* $\nabla_{\mathbf{V}} \hat{R}$

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

Training VAEs



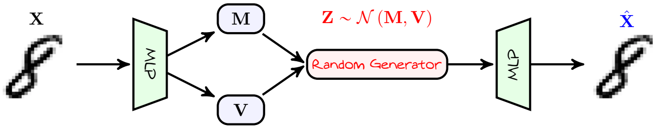
Let's see how training looks: *say we are training with single sample* $\mathbf{X}$

- *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*

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 Models

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 Models

---

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 Models*

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 in your mind*
- ↳ *You got into new challenges?*
  - ↳ *Search online*
  - ↳ *Reach out to me! I would be more than happy!*

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 in your mind*

↳ *You got into new challenges?*

  ↳ *Search online*

  ↳ *Reach out to me! I would be more than happy!*

Next in line . . .

↳ *This Summer Semester*

  ↳ *Reinforcement Learning*

↳ *This Fall Semester*

  ↳ *Creative Applications of NLP*

  ↳ *Generative Models*