Lecture 10: Autoencoders

Madhavan Mukund and Pranabendu Misra

Advanced Machine Learning 2021

Learning (Sensible) Compressed Representations

Learning a sensible and compressed representation of a given dataset is an important task:

- Given an image, learn about the objects in the image
  - Then we can add / remove / modify the objects in the image, thereby generating a new image
  - We can generate similar images
  - We can create sensible interpolation between two images

Learning (Sensible) Compressed Representations

Learning a sensible and compressed representation of a given dataset is an important task:

- Given an image, learn about the objects in the image
  - Then we can add / remove / modify the objects in the image, thereby generating a new image
  - We can generate similar images
  - We can create sensible interpolation between two images

- Given an music file, learn about the instruments in the music
  - Remove noise and distortions
  - Add or remove other instruments
  - and so on ...

This is an unsupervised learning task

- Consider the following two images

- A usual interpolation between the two images could be:

- A more sensible interpolation between the two images would be:

- A more sensible interpolation between the two images would be:

- This is something that, e.g. a child will imagine, a dog transforming into bird looks like.

- Example: shifting the image

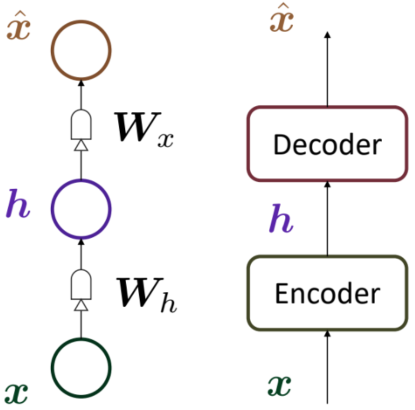
- Example: Night to Day

- Super Resolution

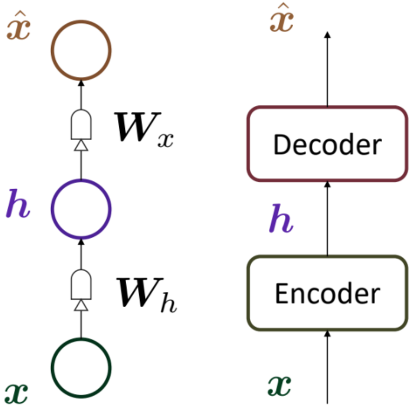
- Caption to Image:

Input Caption: A bright blue bird with white belly

- The idea is to learn how to encode the input into a compressed form, and then re-generate the input from the encoding

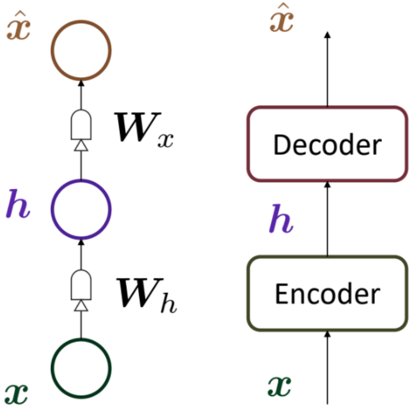


- The idea is to learn how to encode the input into a compressed form, and then re-generate the input from the encoding
- An encoder network learns how to map the input x to a code h in the latent space of far smaller dimension.



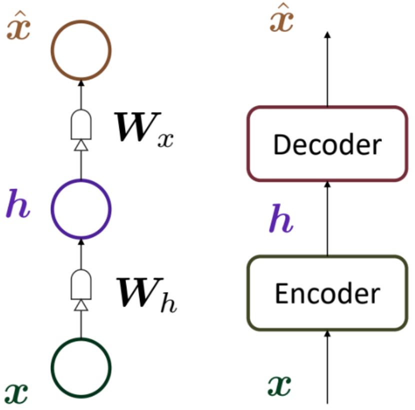
Autoencoders

- The idea is to learn how to encode the input into a compressed form, and then re-generate the input from the encoding
- An encoder network learns how to map the input x to a code h in the latent space of far smaller dimension.
- And at the same time, a decoder network learns to map the code h back to an approximate from of the input $\hat{x}$



Autoencoders

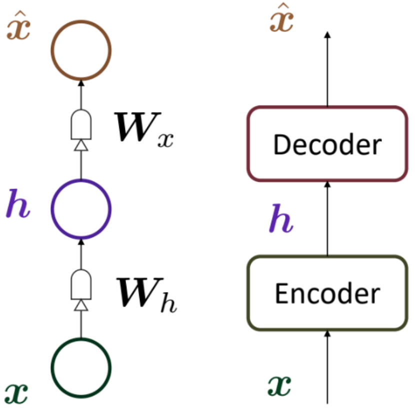
- The idea is to learn how to encode the input into a compressed form, and then re-generate the input from the encoding
- An encoder network learns how to map the input x to a code h in the latent space of far smaller dimension.
- And at the same time, a decoder network learns to map the code h back to an approximate from of the input $\hat{x}$
- This is an *Unsupervised Learning* task.



- In equations:

$$h = f(W_h x)$$
$$\hat{x} = g(W_x h)$$

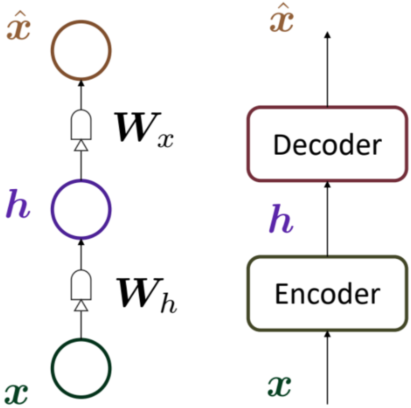
- Here f and g are the non-linear activation functions



- In equations:

$$h = f(W_h x)$$
$$\hat{x} = g(W_x h)$$

- Here f and g are the non-linear activation functions
- Without these non-linear activations, and $W_x = W_h^\top$ the above equations would define PCA (Principal Component Analysis), where the dimension of the vector h is the number of principal components.

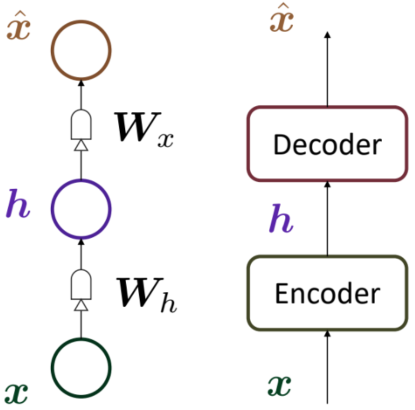


Autoencoders

- In equations:

$$h = f(W_h x)$$
$$\hat{x} = g(W_x h)$$

- Here f and g are the non-linear activation functions
- Without these non-linear activations, and $W_x = W_h^\top$ the above equations would define PCA (Principal Component Analysis), where the dimension of the vector h is the number of principal components.
- So Autoencoders can be thought of as a generalization of PCA



Autoencoders

Loss functions:

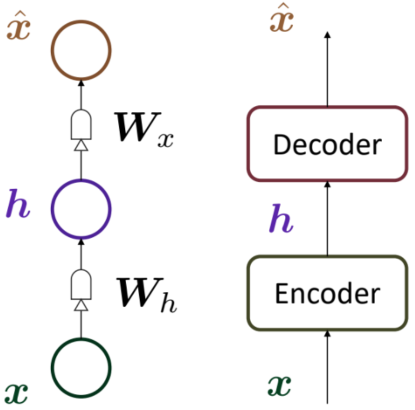
- Mean Squared Error: $\ell(x, \hat{x}) = \frac{1}{2}\|x - \hat{x}\|^2$
  Suitable for $x \in \mathbb{R}^n$, e.g. an image.

- Cross-Entropy loss:

  $$\ell(x, \hat{x}) = -\sum_{i=1}^{n} x_i \log(\hat{x}_i) + (1 - \hat{x}_i) \log(1 - x_i)$$

  suitable for categorical x, e.g. binary.

Reconstruction Errors



Autoencoders

Loss functions:

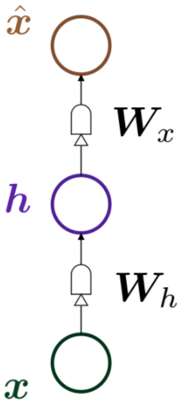
- Mean Squared Error: $\ell(x, \hat{x}) = \frac{1}{2}\|x - \hat{x}\|^2$
  Suitable for $x \in \mathbb{R}^n$, e.g. an image.

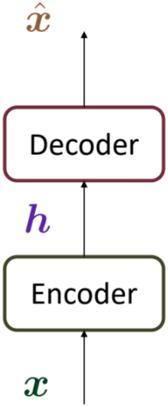
- Cross-Entropy loss:

  $$\ell(x, \hat{x}) = -\sum_{i=1}^{n} x_i \log(\hat{x}_i) + (1 - \hat{x}_i)\log(1 - x_i)$$

  suitable for categorical x, e.g. binary.

- Clearly, minimizing the loss-function means that the encoder and decoder learn to compress and reconstruct the inputs in the training set.





Autoencoders

Loss functions:

- Mean Squared Error: $\ell(x, \hat{x}) = \frac{1}{2}\|x - \hat{x}\|^2$
  Suitable for $x \in \mathbb{R}^n$, e.g. an image.

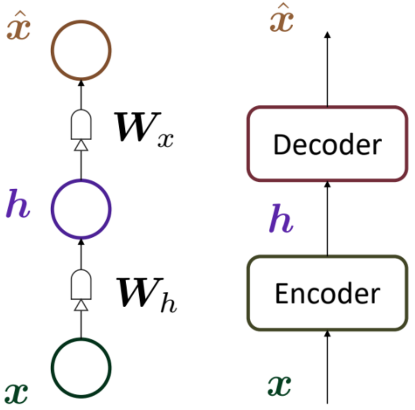
- Cross-Entropy loss:

  $$\ell(x, \hat{x}) = -\sum_{i=1}^{n} x_i \log(\hat{x}_i) + (1 - \hat{x}_i) \log(1 - x_i)$$

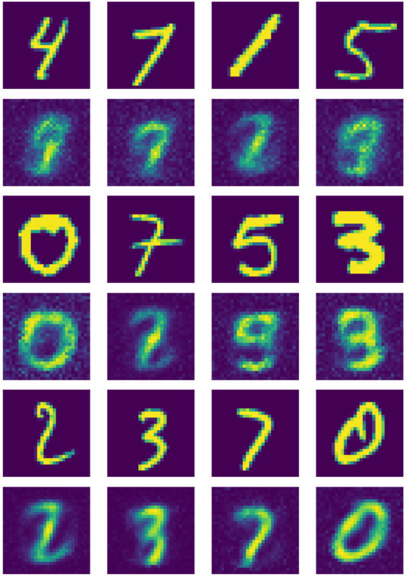
  suitable for categorical x, e.g. binary.

- Clearly, minimizing the loss-function means that the encoder and decoder learn to compress and reconstruct the inputs in the training set.

- We can then validate using the test set.

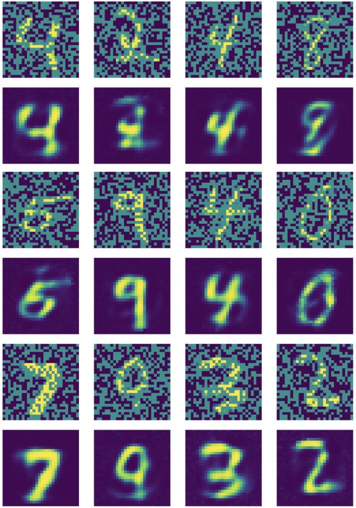


The outputs generated by an autoencoder trained on MNIST.



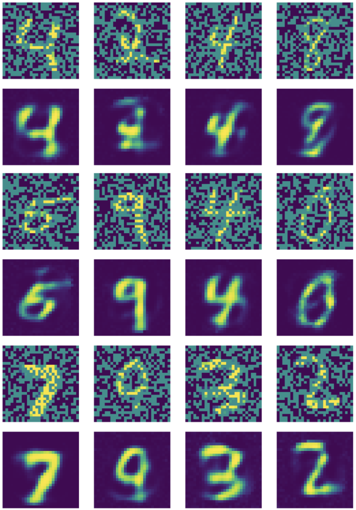
De-noising Autoencoder:

- The input to the encoder is $x + r$ where r is some small amount of random noise.



De-noising Autoencoder:

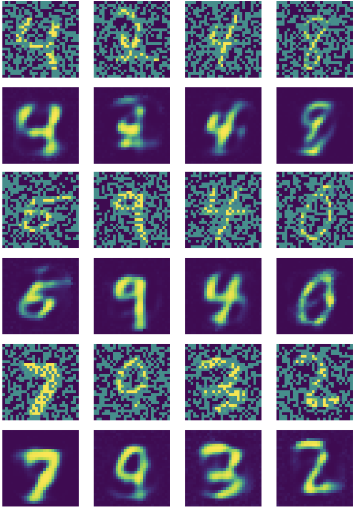
- The input to the encoder is $x + r$ where r is some small amount of random noise.
- The Loss Function still compares x and $\hat{x}$



Autoencoders: Variants

De-noising Autoencoder:

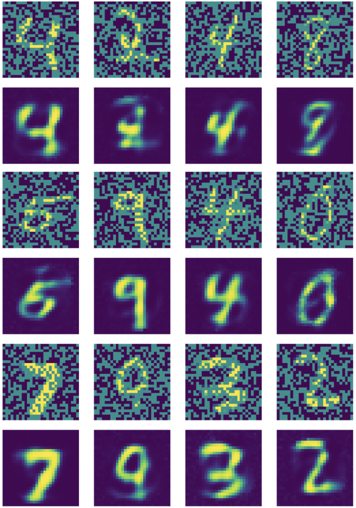
- The input to the encoder is $x + r$ where r is some small amount of random noise.
- The Loss Function still compares x and $\hat{x}$
- The idea is the encoder will learn to separate out the noise r from x, thus learning useful features of the input and generating $\hat{x}$ from those.



Autoencoders: Variants

De-noising Autoencoder:

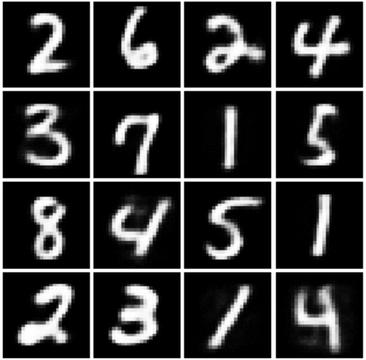
- The input to the encoder is $x + r$ where r is some small amount of random noise.
- The Loss Function still compares x and $\hat{x}$
- The idea is the encoder will learn to separate out the noise r from x, thus learning useful features of the input and generating $\hat{x}$ from those.
- This filters out some of the noise that was present in the input data.



Autoencoders: Variants

Contractive Autoencoder:

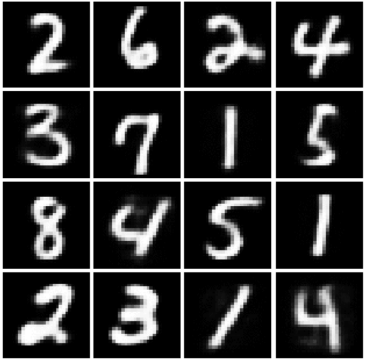
- Enforces the notion that two similar inputs x and x' should map to similar codes h and h' in the latent space.



Autoencoders: Variants

Contractive Autoencoder:

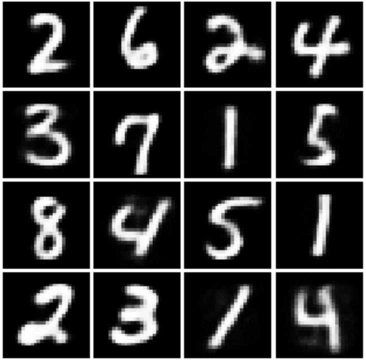
- Enforces the notion that two similar inputs x and x' should map to similar codes h and h' in the latent space.
- The Loss Function has an extra Regularization term, which is $\lambda \cdot \|J_h(x)\|^2$



Autoencoders: Variants

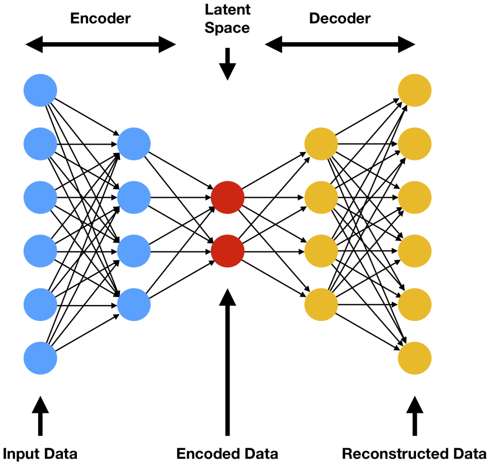
Contractive Autoencoder:

- Enforces the notion that two similar inputs x and x' should map to similar codes h and h' in the latent space.
- The Loss Function has an extra Regularization term, which is $\lambda \cdot \|J_h(x)\|^2$
- Recall that $J_h(x)$, the Jacobian, is the matrix of partial derivatives of h w.r.t. x.
- So the above term in the loss forces smaller derivatives for h. Hence, small changes in x should lead to small changes in h.



Autoencoders: The Latent Space

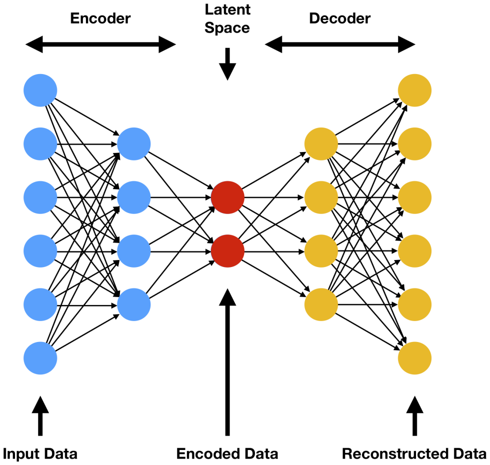
- The input data-points x are mapped to points h in the latent-space.



Autoencoders: The Latent Space

- The input data-points x are mapped to points h in the latent-space.
- We choose the dimension of the latent-space to reflect the representational dimension of the input data.

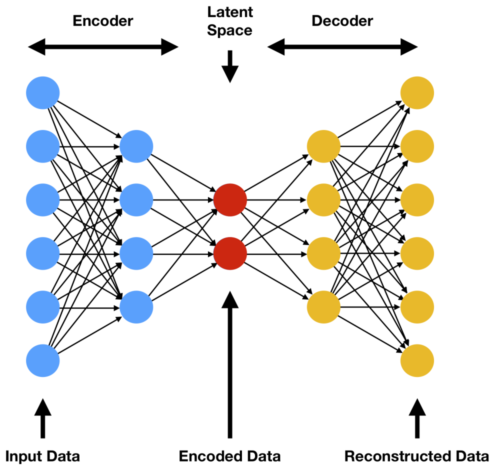
  e.g. for a data-set of human faces, a latent space dimension of 50 would be reasonable.



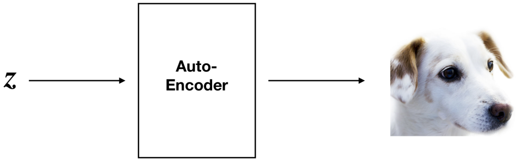
Autoencoders: The Latent Space

- The input data-points x are mapped to points h in the latent-space.
- We choose the dimension of the latent-space to reflect the representational dimension of the input data.

  e.g. for a data-set of human faces, a latent space dimension of 50 would be reasonable.
- Is the latent space sensible? If we sample a point from the latent-space, will the decoder produce an interesting image out of it?

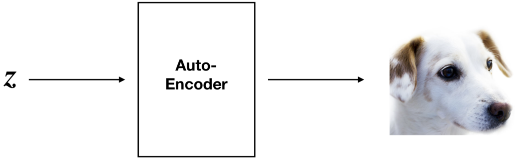


Expectation:

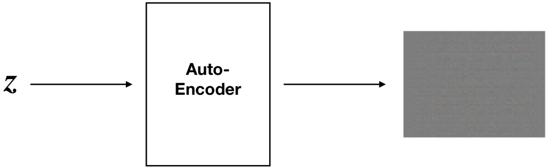


Autoencoders: Data generation from random point in Latent Space?

Expectation:

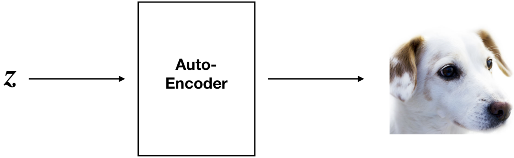


Reality:

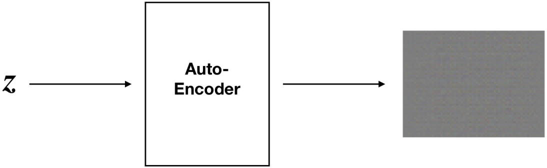


Autoencoders: Data generation from random point in Latent Space?

Expectation:



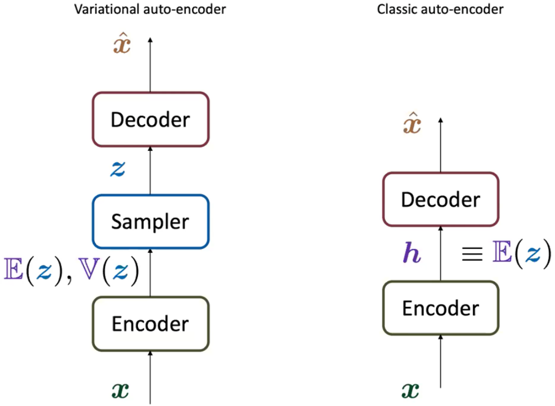
Reality:



Turns out that auto-encoders learn something like jpeg compression, most random latent-space points are meaning-less.

Variational Autoencoders (VAE)

The generation aspect is at the center of this model.

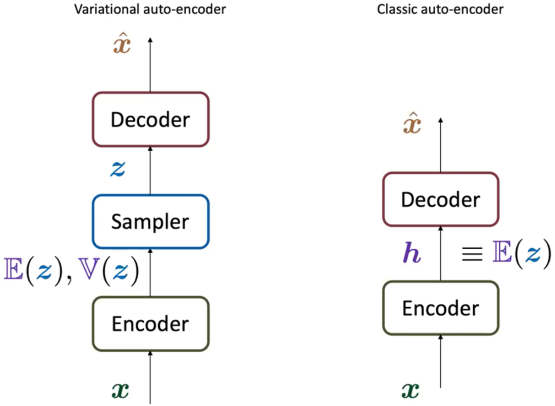


Variational Autoencoders (VAE)

The generation aspect is at the center of this model.

- The encoder maps x to a code h which has two parts $E(z)$ and $V(z)$.

E and V stand for expectation and variance



Variational Autoencoders (VAE)

The generation aspect is at the center of this model.

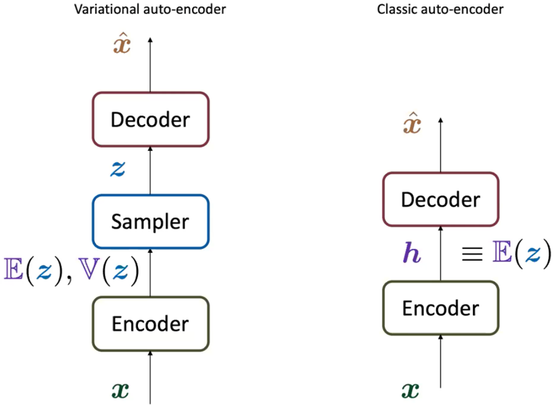
- The encoder maps x to a code h which has two parts $E(z)$ and $V(z)$.

  E and V stand for expectation and variance

- The next step is to sample a point z from a *Gaussian distribution* with expectation $E(z)$ and variance $V(z)$.

$$z = E(z) + \epsilon \odot \sqrt{V(z)}$$

  where $\epsilon \sim N(0, I_d)$ and d is the dimension of the latent-space.



Variational Autoencoders (VAE)

The generation aspect is at the center of this model.

- The encoder maps x to a code h which has two parts $E(z)$ and $V(z)$.

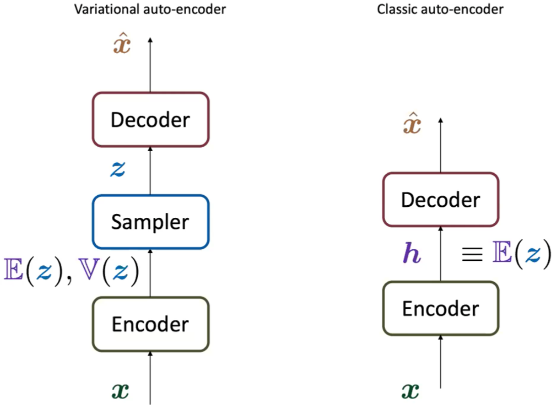
  E and V stand for expectation and variance

- The next step is to sample a point z from a *Gaussian distribution* with expectation $E(z)$ and variance $V(z)$.

  $$z = E(z) + \epsilon \odot \sqrt{V(z)}$$

  where $\epsilon \sim N(0, I_d)$ and d is the dimension of the latent-space.

- Finally, the decoder maps this randomly sampled point z to $\hat{x}$



Variational Autoencoders (VAE)

The generation aspect is at the center of this model.

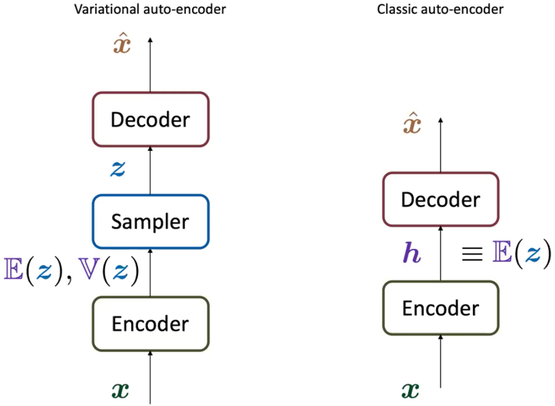
- The encoder maps x to a code h which has two parts $E(z)$ and $V(z)$.

  E and V stand for expectation and variance

- The next step is to sample a point z from a *Gaussian distribution* with expectation $E(z)$ and variance $V(z)$.

$$z = E(z) + \epsilon \odot \sqrt{V(z)}$$

  where $\epsilon \sim N(0, I_d)$ and d is the dimension of the latent-space.



So the code in the latent-space specifies a probability distribution.

Variational Autoencoders (VAE)

The generation aspect is at the center of this model.

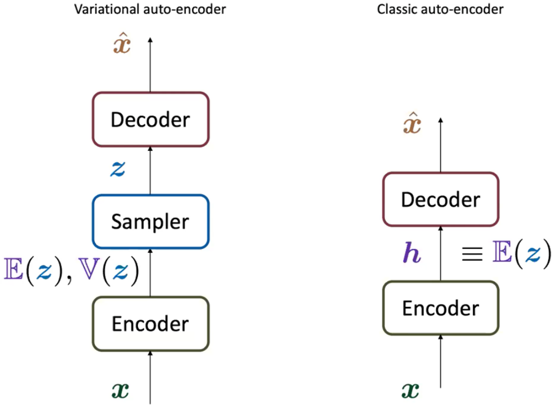
- The encoder maps x to a code h which has two parts $E(z)$ and $V(z)$.

  E and V stand for expectation and variance

- The next step is to sample a point z from a *Gaussian distribution* with expectation $E(z)$ and variance $V(z)$.

$$z = E(z) + \epsilon \odot \sqrt{V(z)}$$

where $\epsilon \sim N(0, I_d)$ and d is the dimension of the latent-space.



So the code in the latent-space specifies a probability distribution.

Variational Autoencoders (VAE)

Loss functions for VAE:

- Apart from the usual loss, there is an extra regularization term in the loss

$$\beta \cdot \ell_{KL}(z, N(0, I_d))$$

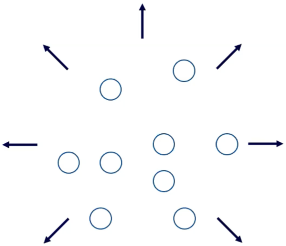
Variational Autoencoders (VAE)

Loss functions for VAE:

- Apart from the usual loss, there is an extra regularization term in the loss

$$\beta \cdot \ell_{KL}(z, N(0, I_d))$$

- The reconstruction error $\ell(x, \hat{x}) = \frac{1}{2}\|x - \hat{x}\|^2$ pushes the "probability balls" apart, because it penalizes overlapping.
  We also have no control over the "size" of the bubbles.



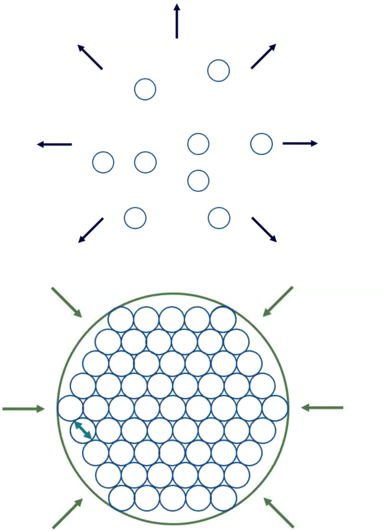
Loss functions for VAE:

- Apart from the usual loss, there is an extra regularization term in the loss

$$\beta \cdot \ell_{KL}(z, N(0, I_d))$$

- The reconstruction error $\ell(x, \hat{x}) = \frac{1}{2}\|x - \hat{x}\|^2$ pushes the "probability balls" apart, because it penalizes overlapping.
  We also have no control over the "size" of the bubbles.

- Introducing regularization term in the loss counters this.



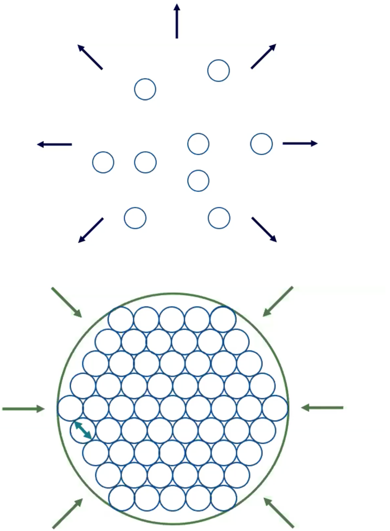
Loss functions for VAE:

- Apart from the usual loss, there is an extra regularization term in the loss

$$\ell_{KL}(z, N(0, I_d)) =$$

$$\frac{1}{2}\sum_{i=1}^{d}(V(z_i) - \log(V(z_i)) - 1 + E(z_i)^2)$$

divergence b/w distribution of z and $N(0, I_d)$.



Variational Autoencoders (VAE)

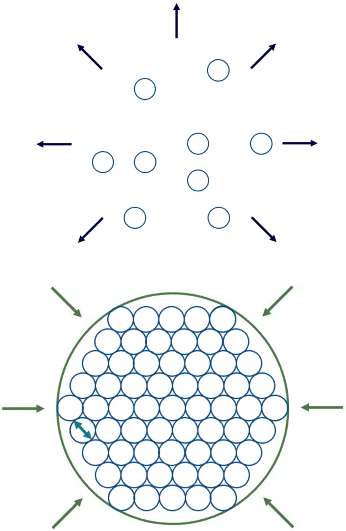
Loss functions for VAE:

- Apart from the usual loss, there is an extra regularization term in the loss

$$\ell_{KL}(z, N(0, I_d)) =$$
$$\frac{1}{2}\sum_{i=1}^{d}(V(z_i) - \log(V(z_i)) - 1 + E(z_i)^2)$$

divergence b/w distribution of z and $N(0, I_d)$.

- The term $(V(z_i) - \log(V(z_i)) - 1)$ is minimized when $V(z_i) = 1$. This makes the "size" of the balls radius-1.



Variational Autoencoders (VAE)

Loss functions for VAE:

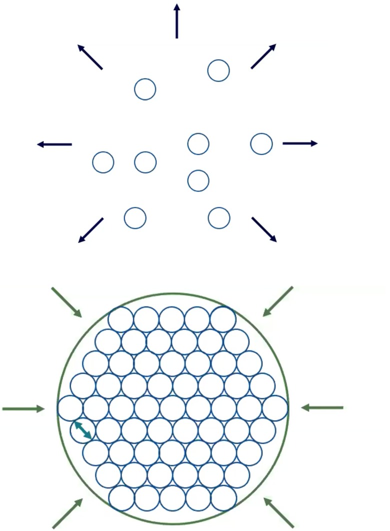
- Apart from the usual loss, there is an extra regularization term in the loss

  $\ell_{KL}(z, N(0, I_d)) =$
  $$\frac{1}{2}\sum_{i=1}^{d}(V(z_i) - \log(V(z_i)) - 1 + E(z_i)^2)$$

  divergence b/w distribution of z and $N(0, I_d)$.

- The term $(V(z_i) - \log(V(z_i)) - 1)$ is minimized when $V(z_i) = 1$. This makes the "size" of the balls radius-1.

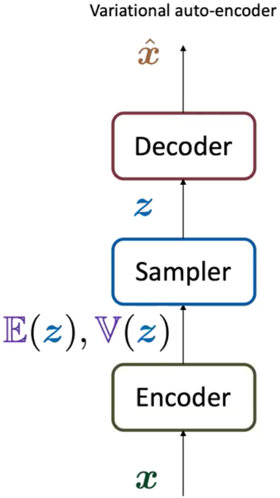
- The term $E(z_i)^2$ penalizes large values, so the balls are tightly packed as much as possible.



Variational Autoencoders (VAE)

Loss functions for VAE:

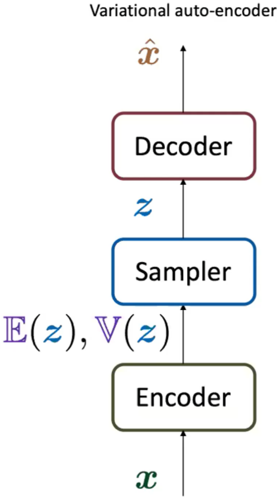
- Overall, the loss function leads to a *clustering* of similar inputs in the latent-space, and these clusters *densely packed*



Variational Autoencoders (VAE)

Loss functions for VAE:

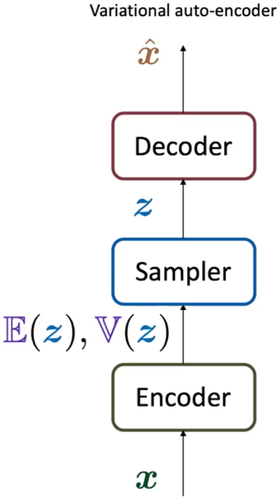
- Overall, the loss function leads to a *clustering* of similar inputs in the latent-space, and these clusters *densely packed*
- This means that if we pick a point in the latent-space, it can generate a good-quality image.



Variational Autoencoders (VAE)

Loss functions for VAE:

- Overall, the loss function leads to a *clustering* of similar inputs in the latent-space, and these clusters *densely packed*
- This means that if we pick a point in the latent-space, it can generate a good-quality image.
- We can also *interpolate* between two input-points in the latent-space to generate new images.



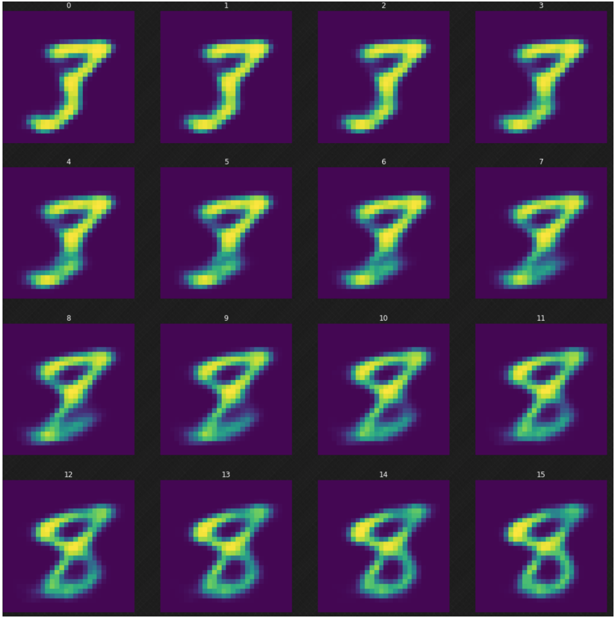
Examples: VAE trained on Celebrity Faces

Interpolating between hair-colors:

Examples: VAE trained on Celebrity Faces

Interpolating between hair-colors:

Interpolating between no glasses and glasses


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

References:

These slides are based on:

- https://www.compthree.com/blog/autoencoder/
- https://atcold.github.io/pytorch-Deep-Learning/en/week07/07-3/
- https://atcold.github.io/pytorch-Deep-Learning/en/week08/08-3/