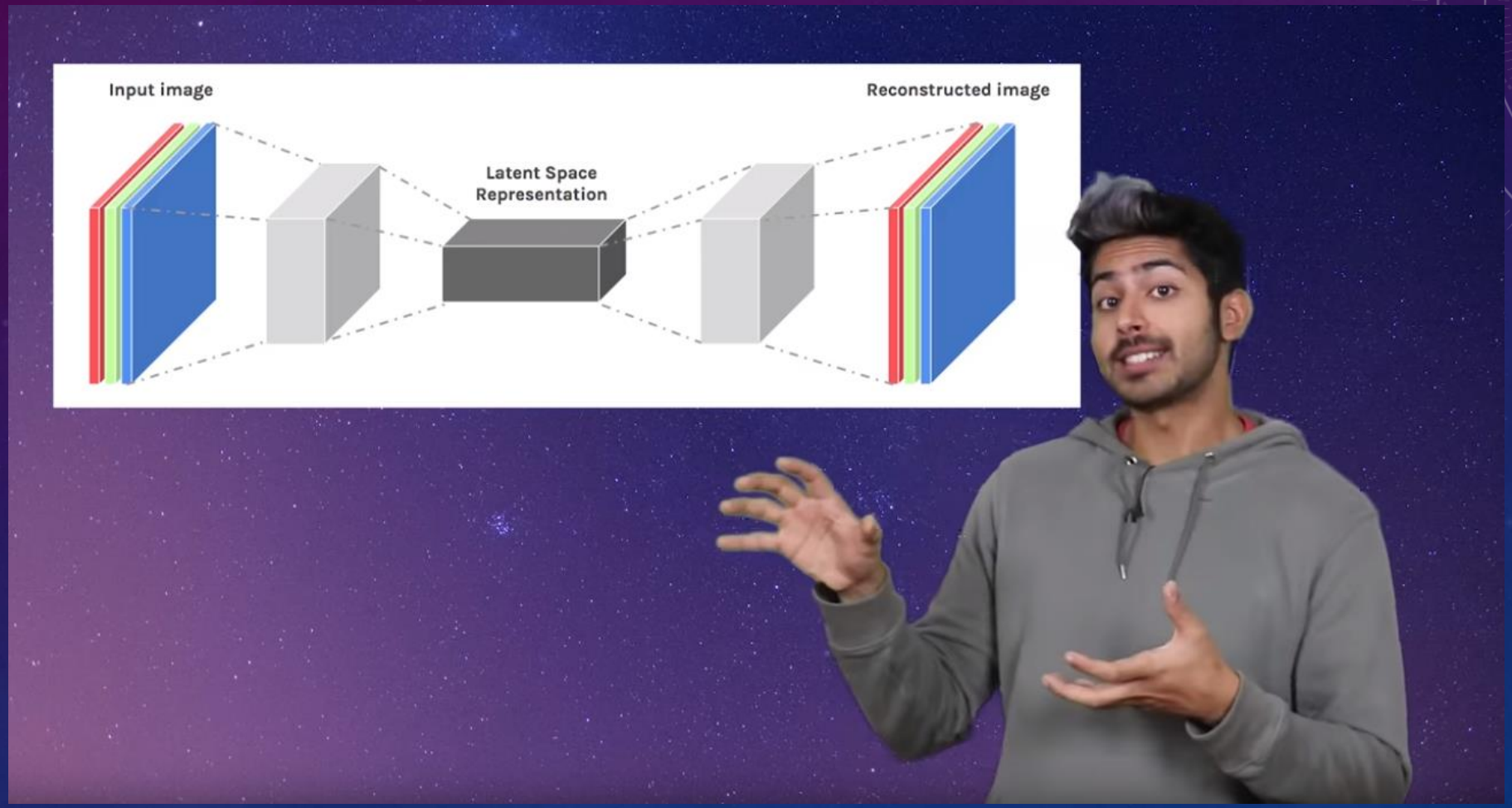
Digital Surveillance Systems and Application

AUTOENCODERS

Jison G.S. Hsu
Artificial Vision Laboratory
Taiwan Tech

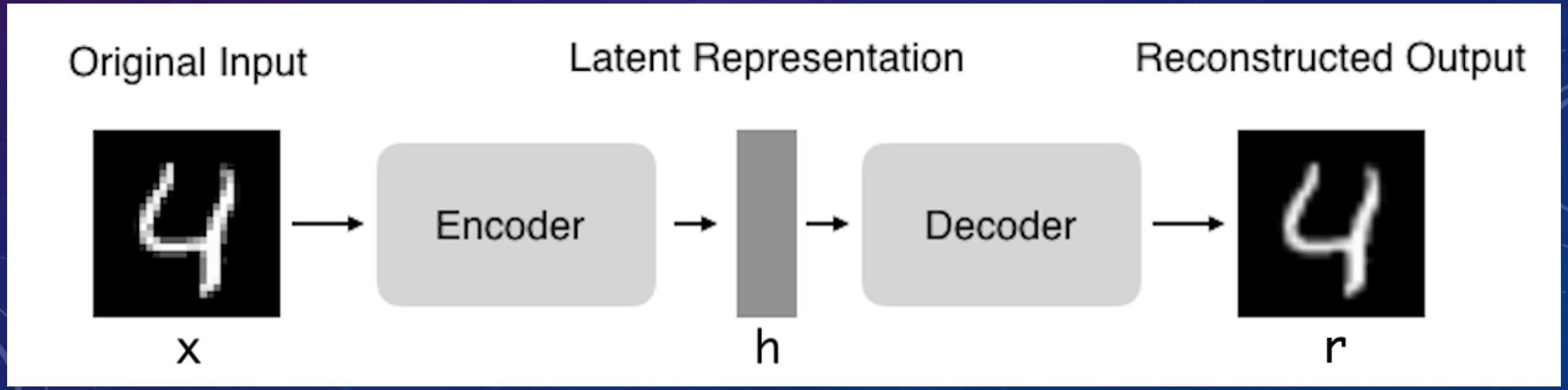
Autoencoder Explained



https://youtu.be/H1AllrJ-_30 [8:41]

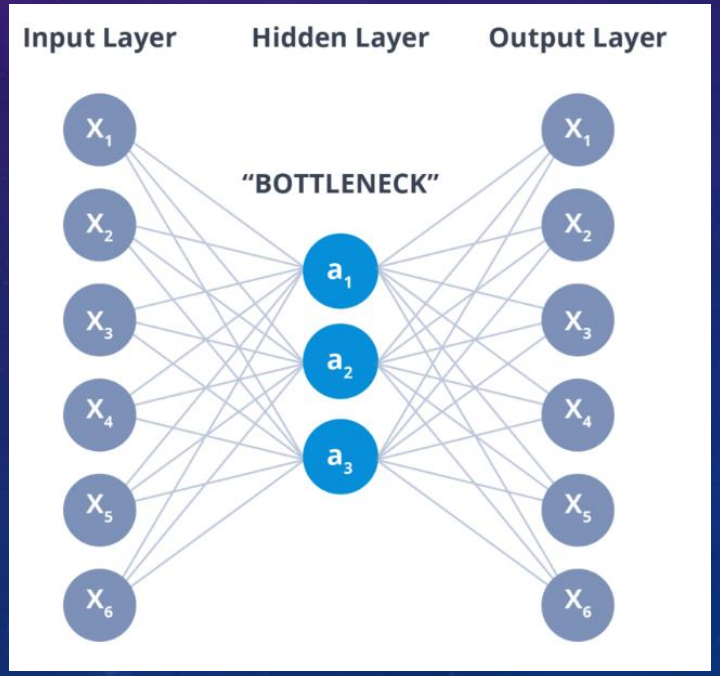
Autoencoders (AE)

- Autoencoders (AE) are neural networks that aims to copy their inputs to their outputs. They work by compressing the input into a **latent-space representation**, and then reconstructing the output from this representation.
  - **Encoder:** Compresses the input into a latent-space representation. It can be represented by an encoding function $h = f(x)$.
  - **Decoder:** Reconstruct the input from the latent space representation. It can be represented by a decoding function $r = g(h)$.



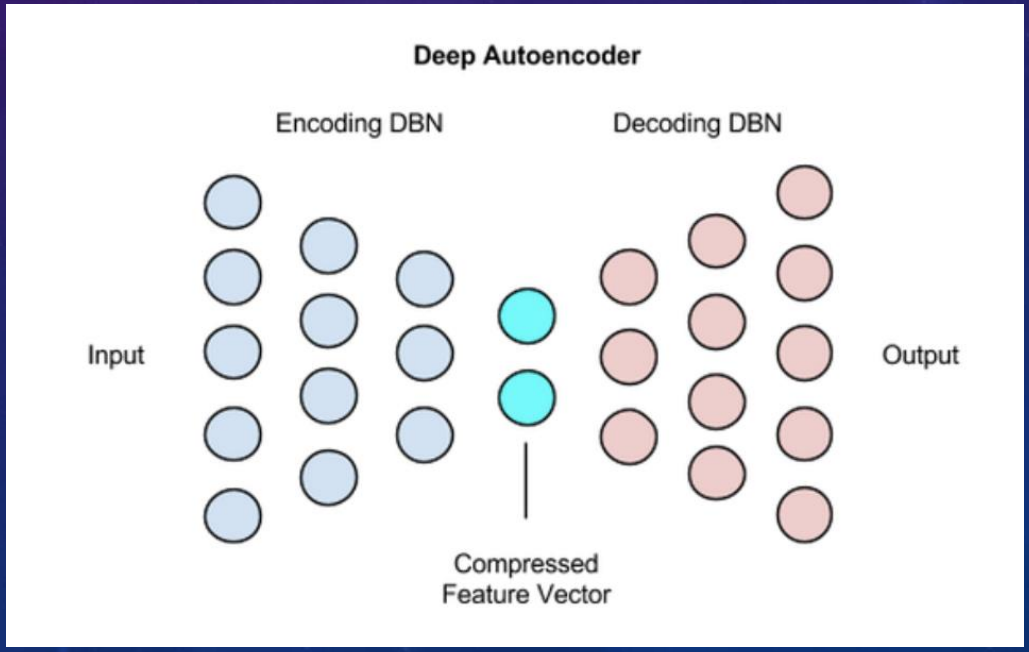
Bottleneck

- The layer between the encoder and decoder, ie. the code is also known as **Bottleneck**. This is a well-designed approach to decide which aspects of observed data are relevant information and what aspects can be discarded.
- It does this by balancing two criteria :
  - Compactness of representation, measured as the compressibility.
  - It retains some behaviourally relevant variables from the input.



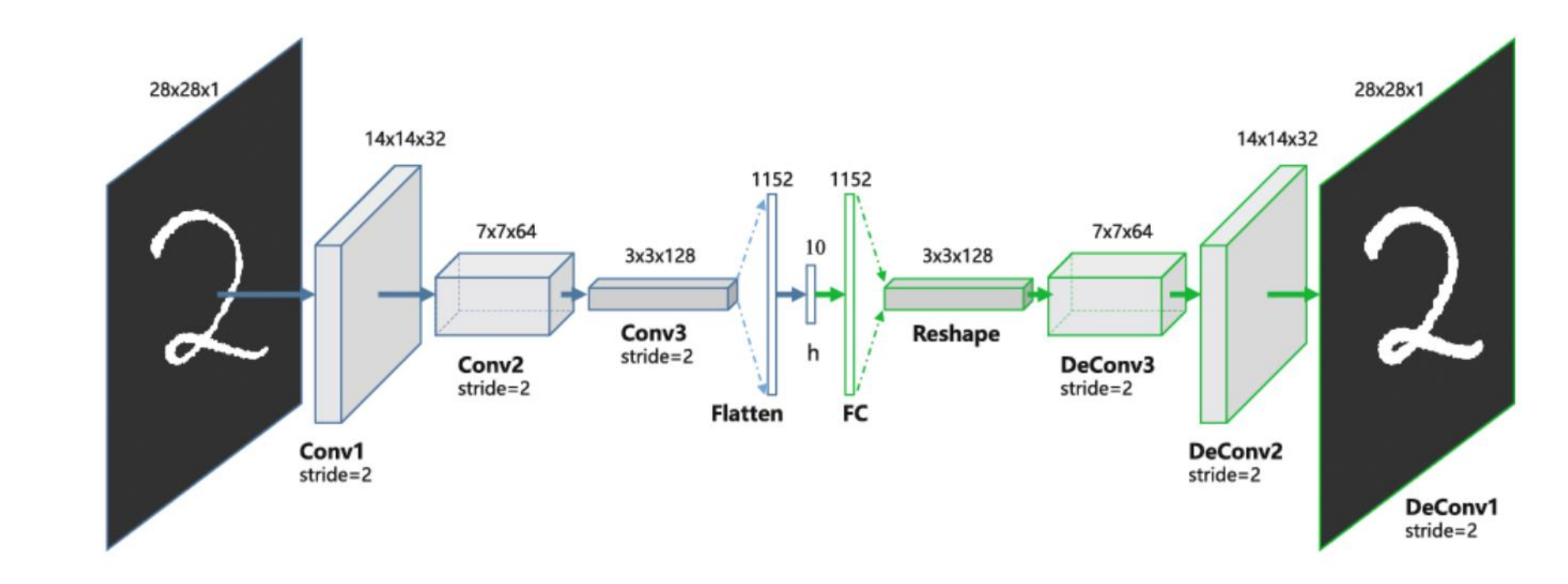
Deep Autoencoders

- The extension of the simple Autoencoder is the **Deep Autoencoder**. Deeper layers of the Deep Autoencoder tend to learn even higher-order features.
- A **deep autoencoder** is composed of two, symmetrical deep-belief networks-
  - First four or five shallow layers representing the encoding half of the net.
  - The second set of four or five layers that make up the decoding half.



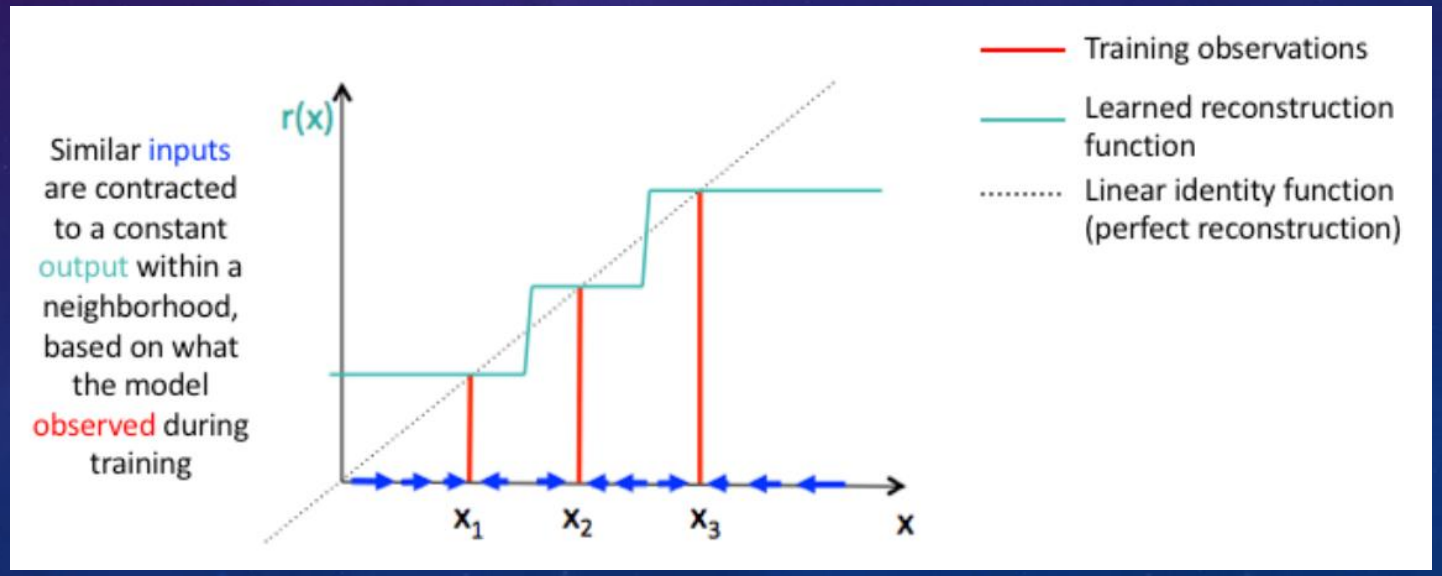
Convolution Autoencoders

- Convolutional Autoencoders use the convolution operator to exploit this observation. They learn to encode the input in a set of simple signals and then try to reconstruct the input from them, modify the geometry or the reflectance of the image.



Contractive Autoencoders

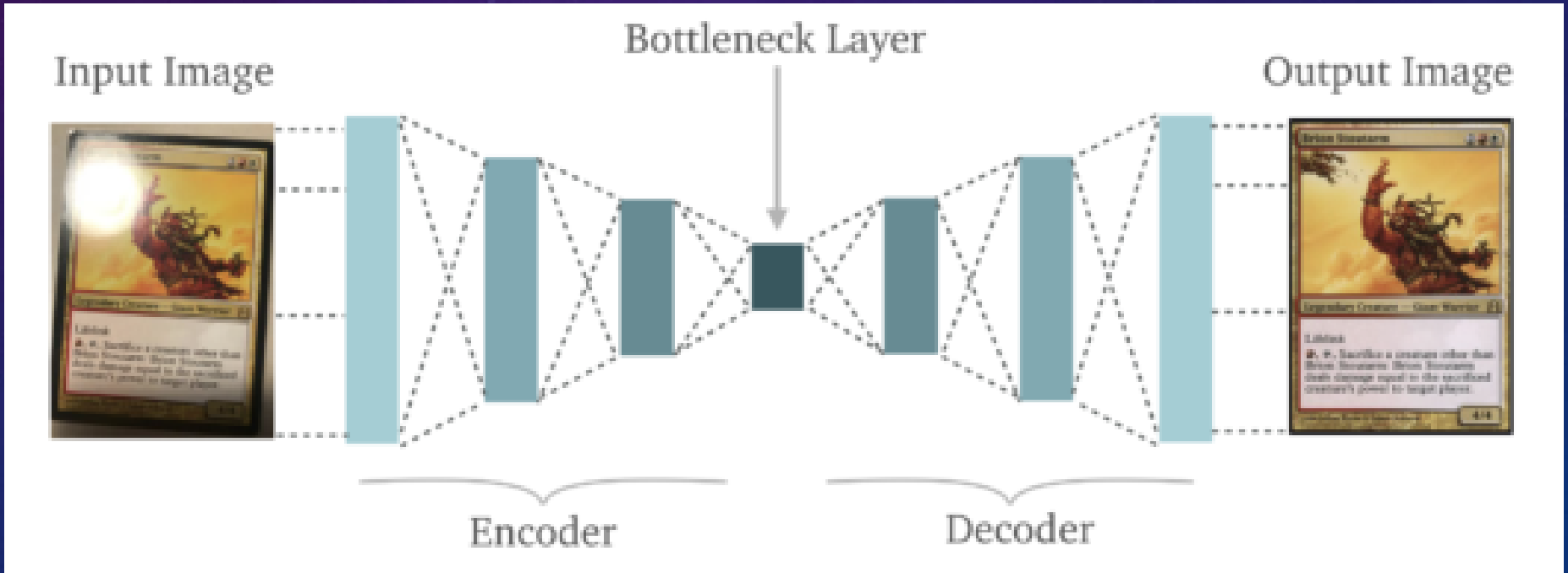
- A **contractive autoencoder** is an unsupervised deep learning technique that helps a neural network encode unlabeled training data.
- This is accomplished by constructing a **loss term** which penalizes large derivatives of our hidden layer activations with respect to the input training examples, **essentially penalizing** instances where a small change in the input leads to a large change in the encoding space.



Applications of Autoencoders

- Feature variation

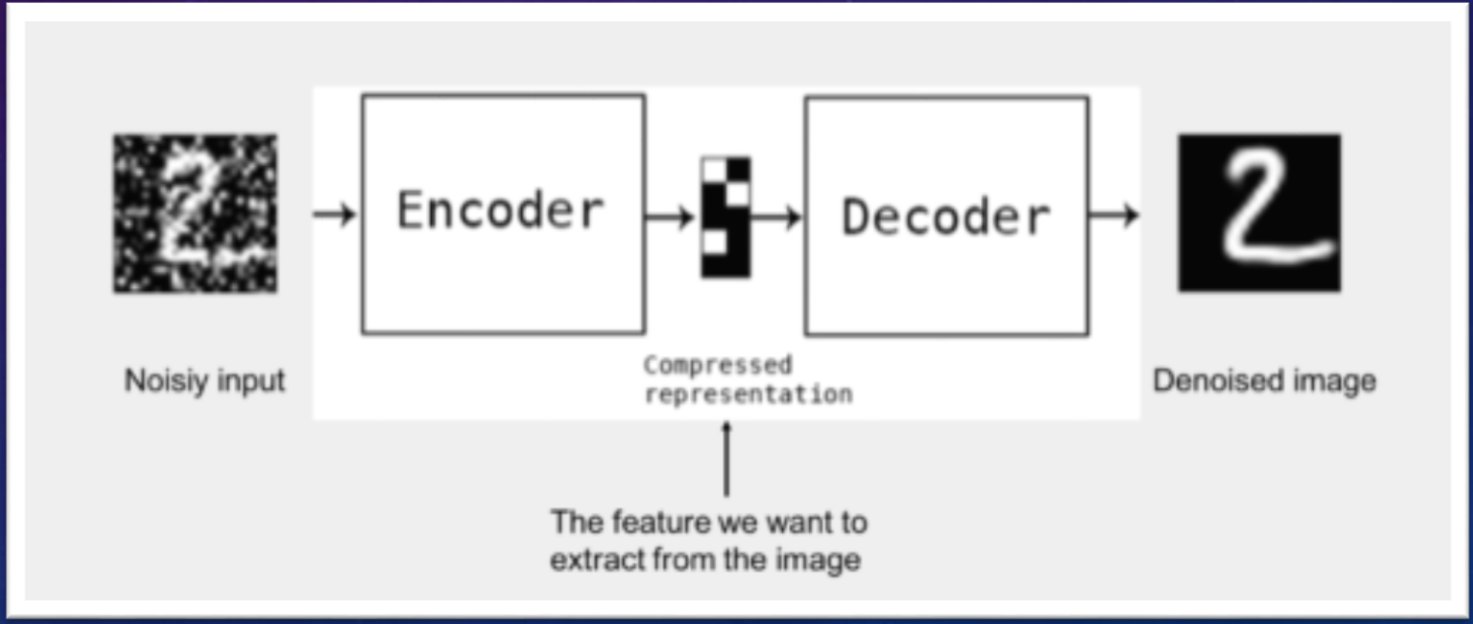
  It extracts only the required features of an image and generates the output by removing any noise or unnecessary interruption.



Applications of Autoencoders

- Denoising Image

  The input seen by the autoencoder is not the raw input but a stochastically corrupted version. A denoising autoencoder is thus trained to reconstruct the original input from the noisy version.



Variational Autoencoders

https://youtu.be/9zKuYvjFFS8 [15:05]

Lecture 13 | Generative Models

https://youtu.be/5WoItGTWV54 [1:17:40]

Lecture 13 | Generative Models

- OUTLINE:
- 1:12 - Supervised learning
- 5:33 - Generative models
- 11:00 - PixelRNN and PixelCNN
- 19:36 - Traditional and variational autoencoders (VAEs)
- 50:00 - Generative adversarial networks (GANs)

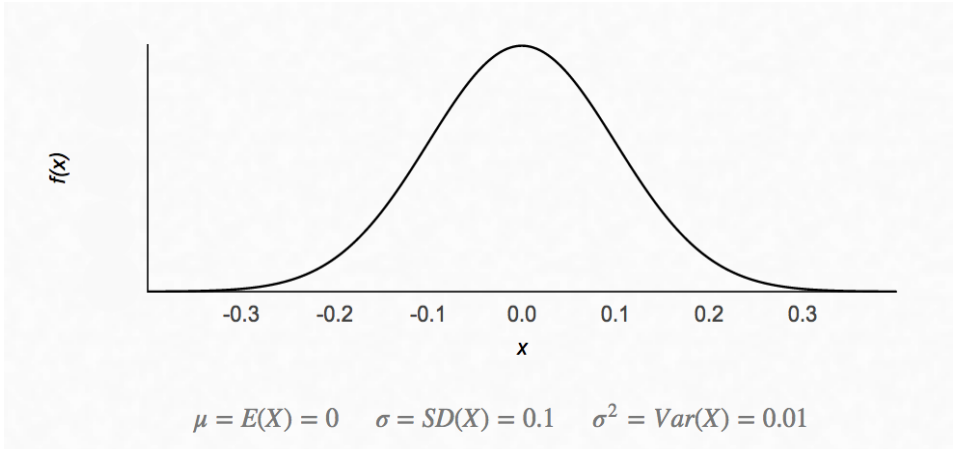
Variational Autoencoders

- Variational autoencoders use gaussian models to generate images.
- There are some backgrounds we need to know:
  - Gaussian distribution
  - Autoencoders
  - KL-divergence

https://jhui.github.io/2017/03/06/Variational-autoencoders/

Gaussian distribution

- In the following diagram, we assume the probability of X equal to a certain value x, $p(X=x)$, follows a gaussian distribution:



- We can sample data using the PDF above. We use the following notation for sample data using a gaussian distribution with mean μ and standard deviation σ.

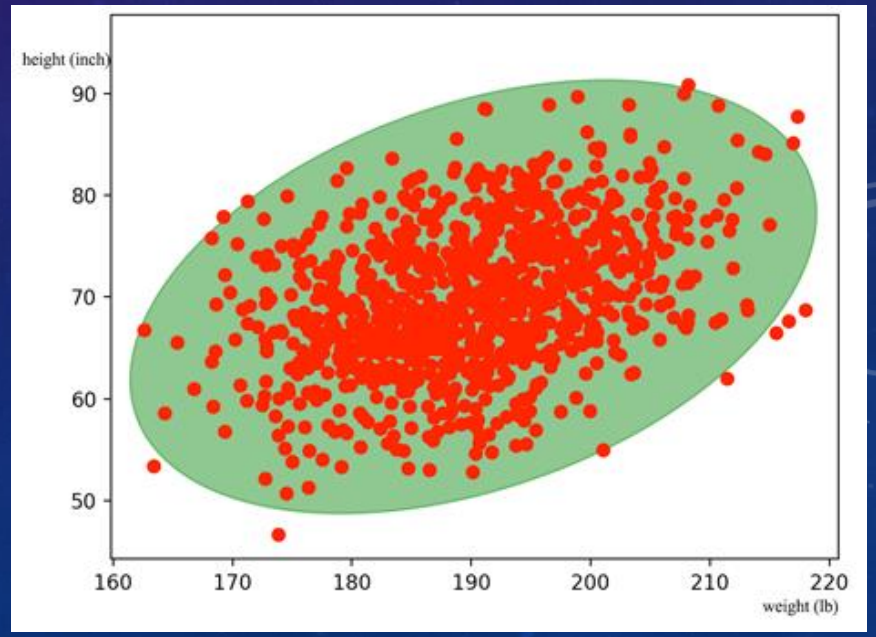
$$x \sim N(\mu, \sigma)$$

Gaussian distribution

- In the example above, mean: μ=0, standard deviation: σ=0.1:

  In many real world examples, the data sample distribution follows a gaussian distribution.

- Now, we generalize it with multiple variables. For example, we want to model the relationship between the body height and the body weight for San Francisco residents.
- We collect the information from 1000 adult residents and plot the data below with each red dot represents 1 person:

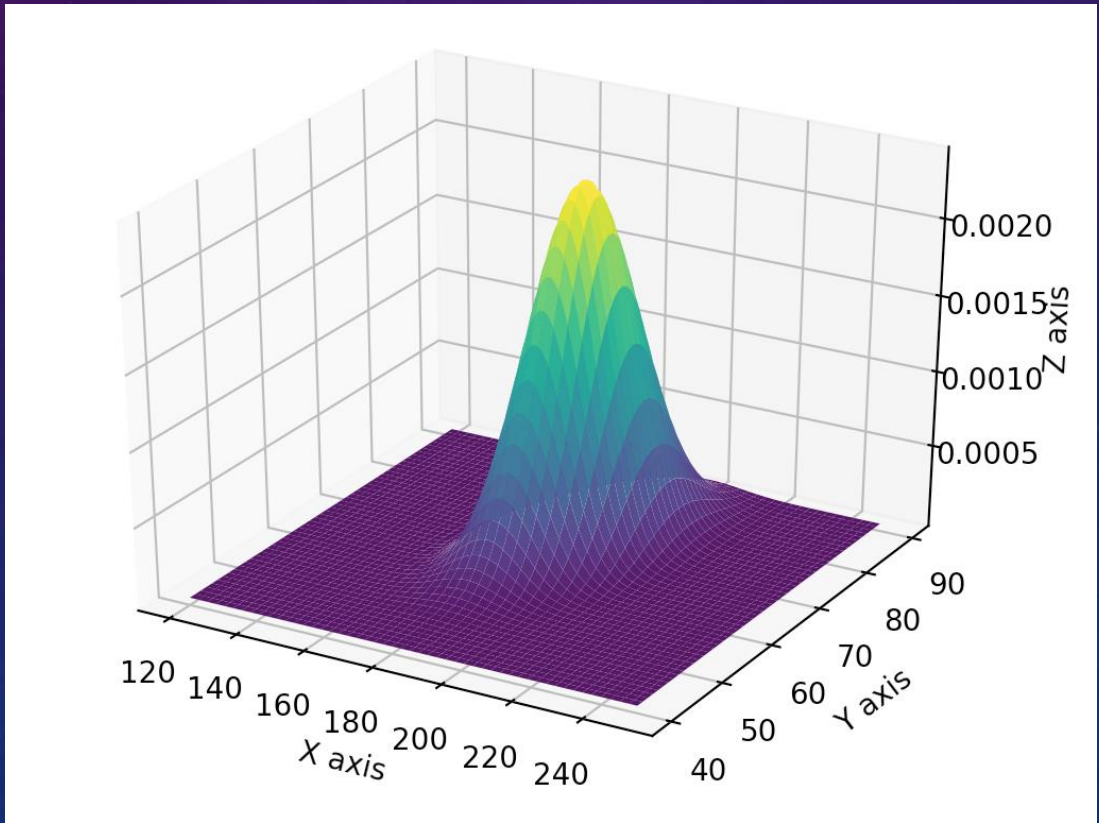


Gaussian distribution

- We can plot the corresponding probability density function

PDF = probability(height = h, weight = w)

- in 3D:



Gaussian distribution

- We can model such a probability density function using a gaussian distribution function. The PDF with p variables is:

$$z = \begin{pmatrix} \mathrm{x}_1 \\ \vdots \\ \mathrm{x}_p \end{pmatrix}, \; f(x) = \frac{1}{(2\pi)^{p/2}|\sum|^{1/2}} \exp\{-\frac{1}{2}(x-\mu)^T \sum^{-1}(x-\mu)\}$$

- with covariance matrix $\sum$:

$$\sum = \begin{pmatrix} E[(x_1-\mu_1)(x_1-\mu_1)] & E[(x_1-\mu_1)(x_2-\mu_2)] & \dots & E[(x_1-\mu_1)(x_p-\mu_p)] \\ E[(x_2-\mu_2)(x_1-\mu_1)] & E[(x_2-\mu_2)(x_2-\mu_2)] & \dots & E[(x_2-\mu_2)(x_p-\mu_p)] \\ \vdots & \vdots & \ddots & \vdots \\ E[(x_p-\mu_p)(x_1-\mu_1)] & E[(x_p-\mu_p)(x_2-\mu_2)] & \dots & E[(x_n-\mu_p)(x_p-\mu_p)] \end{pmatrix}$$

Gaussian distribution

- The notation for sampling x is:

$$x \sim \mathcal{N}(\mu, \sum)$$

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_p \end{pmatrix} \sim \mathcal{N}(\mu, \sum) = \mathcal{N}\left(\begin{pmatrix} \mu_1 \\ \vdots \\ \mu_p \end{pmatrix}, \sum\right)$$

- Let's go back to our weight and height example to illustrate it

$$x = \begin{pmatrix} weight \\ height \end{pmatrix}$$

- From the data, we compute the mean weight is 190 lb and mean height is 70 inches:

$$\mu = \begin{pmatrix} 190 \\ 70 \end{pmatrix}$$

Gaussian distribution

- For the covariance matrix $\sum$, here, we illustrate how to compute one of the element E_{21}

$$E_{21} = E[(x_2 - \mu_2)(x_1 - \mu_1)] = E\left[\left(x_{height} - 70\right)\left(x_{weight} - 190\right)\right]$$

- which E is the expected value. Let say we have only 2 datapoints (200 lb, 80 inches) and (180 lb, 60 inches)

$$E_{21} = E\left[\left(x_{height} - 70\right)\left(x_{weight} - 190\right)\right]$$
$$= \frac{1}{2}((80 - 70) \times (200 - 190) + (60 - 70) \times (180 - 190))$$

- After computing all 1000 data, here is the value of $\sum$:

$$\sum = \begin{pmatrix} 100 & 25 \\ 25 & 50 \end{pmatrix}, \qquad x \sim N\left(\begin{pmatrix} 190 \\ 70 \end{pmatrix}, \begin{pmatrix} 100 \\ 50 \end{pmatrix}\right)$$

Gaussian distribution

- E_{21}measures the co-relationship between variables x_2 and x_1. Positive values means both are positively related. With not surprise, E_{21} is positive because weight increases with height. If two variables are independent of each other, it should be 0 like:

$$\sum = \begin{pmatrix} 100 & 0 \\ 0 & 50 \end{pmatrix}$$

- and we will simplify the gaussian distribution notation here as:

$$x \sim N\left(\begin{pmatrix} 190 \\ 70 \end{pmatrix}, \begin{pmatrix} 100 \\ 50 \end{pmatrix}\right)$$

Autoencoders

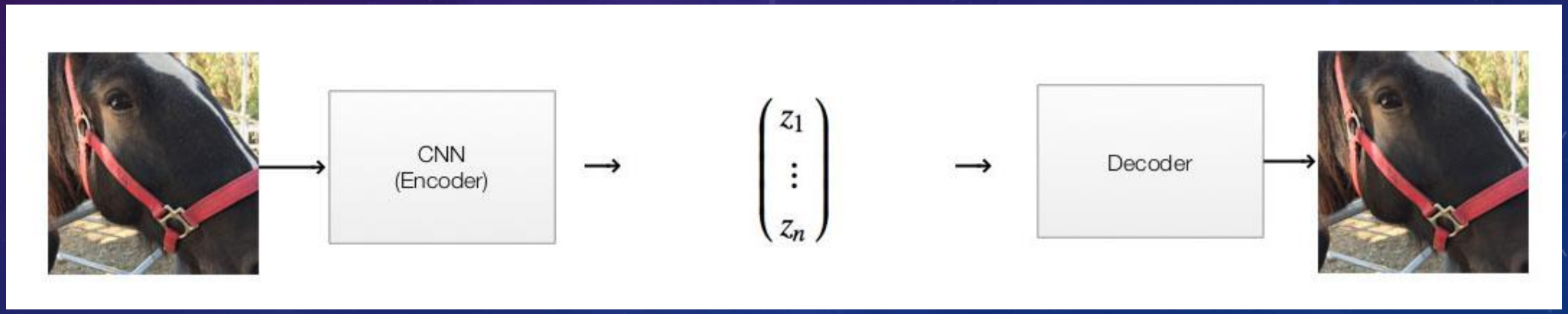
- In an autoencoders, we use a deep network to map the input image (for example 256x256 pixels = 256x256 = 65536 dimension) to a lower dimension latent variables (latent vector say 100-D vector: $(x_1, x_2, \cdots x_{100})$).

- We use another deep network to decode the latent variables to restore the image. We train both encoder and decoder network to minimize the difference between the original image and the decoded image.

- By forcing the image to a lower dimension, we hope the network learns to encode the image by extracting core features.

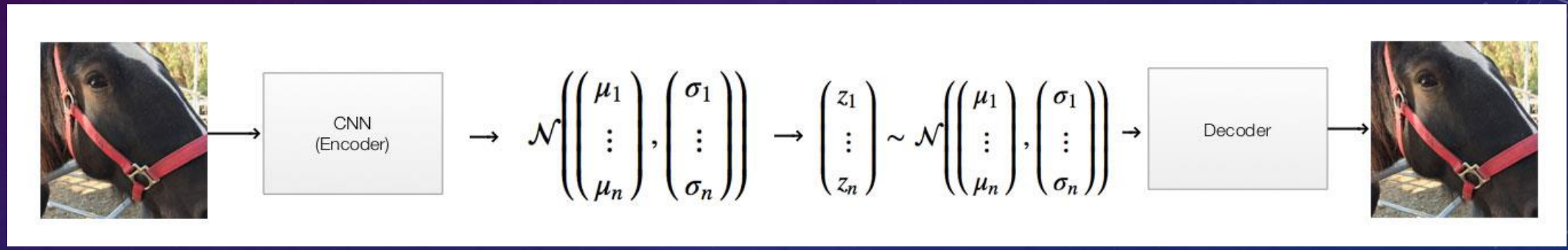
Autoencoders

- For example, we enter a 256x256 image to the encoder, we use a CNN to encode the image to 20-D latent variables $(x_1, x_2, \ldots, x_{20})$=(0.1, 0, ..., −0.05).
- We use another network to decode the latent variables into a 256x256 image.
- We use backpropagation with cost function comparing the decoded and input image to train both encoding and decoding network



Variational Autoencoders (VAEs)

- For VAEs, we replace the middle part with a stochastic model using a gaussian distribution. Let's get into an example to demonstrate the flow:



- For a variation autoencoder, we replace the middle part with 2 separate steps. VAE does not generate the latent vector directly.

Variational Autoencoders (VAEs)

- It generates 100 Gaussian distributions each represented by a mean (μ_i) and a standard deviation (σ_i).
- Then it samples a latent vector, say (0.1, 0.03, …, -0.01), from these distributions.
- For example, if element x_i of the latent vector has μ_i =0.1 and σ_i =0.5. We randomly select x_i with probability based on this Gaussian distribution:

$$p(X = x_i) = \frac{e^{-(x_i-\mu_i)^2/(2\sigma_i^2)}}{\sigma_i\sqrt{2\pi}}$$

$$z = \begin{pmatrix} z_1 \\ \vdots \\ z_{20} \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \mu_1 \\ \vdots \\ \mu_{20} \end{pmatrix}, \begin{pmatrix} \sigma_1 \\ \vdots \\ \sigma_{20} \end{pmatrix}\right)$$

Variational Autoencoders (VAEs)

- Say, the encoder generates μ = (0, −0.01, ..., 0.2) and σ = (0.05, 0.01, ..., 0.02). We can sample a value from this distribution:

$$N\left(\begin{pmatrix} 0 \\ -0.01 \\ \vdots \\ 0.2 \end{pmatrix}, \begin{pmatrix} 0.05 \\ 0.01 \\ \vdots \\ 0.02 \end{pmatrix}\right)$$

- with the latent variables as (say) :

$$z = \begin{pmatrix} Z_1 \\ \vdots \\ Z_{20} \end{pmatrix} = \begin{pmatrix} 0.03 \\ -0.015 \\ \vdots \\ 0.197 \end{pmatrix}$$

Variational Autoencoders (VAEs)

- The autoencoder in the previous section is very hard to train with not much guarantee that the network is generalize enough to make good predictions.(We say the network simply memorize the training data.)
- In VAEs, we add a constrain to make sure:
  1. The latent variable are relative independent of each other, i.e. the 20 variables are relatively independent of each other (not co-related). This maximizes what a 20-D latent vectors can represent.
  2. Latent variables z which are similar in values should generate similar looking images. This is a good indication that the network is not trying to memorize individual image.

Variational Autoencoders (VAEs)

- To achieve this, we want the gaussian distribution model generated by the encoder to be as close to a normal gaussian distribution function.
- We penalize the cost function if the gaussian function is deviate from a normal distribution. This is very similar to the L_2 regularization in a fully connected network in avoiding overfitting.

$$z \sim N(0,1) = normal\ distribution$$

Variational Autoencoders (VAEs)

- In a normal gaussian distribution, the covariance E_{ij} is 0 for $i \neq j$. That is the latent variables are independent of each other.
- If the distribution is normalize, the distance between different z will be a good measure of its similarity.
- With sampling and the gaussian distribution, we encourage the network to have similar value of z for similar images.

Cost Function in Detail

- In VAE, we want to model the data distribution $p(x)$ with an encoder $q_\emptyset(z|x)$, a decoder $p_\theta(z|x)$ and a latent variable model $p(z)$ through the VAE objective function:

$$\log p(x) \approx E_q[logp_\theta(x|z)] - D_{KL}[q_\emptyset(z|x)||p(z)]$$

- To draw this conclusion, we start with the KL divergence which measures the difference of 2 distributions. By definition, KL divergence is defined as:

$$D_{KL}(q||p) = \sum_x q(x) \log\left(\frac{q(x)}{p(x)}\right) = E_q[\log\big(q(x)\big) - \log\big(p(x)\big)]$$

- Apply it with:

$$D_{KL}[\big(q\big|\big|p\big)\big|\big|p(z|x)\big] = E[logq(z|x) - logp(z|x)]$$

Cost Function in Detail

- Let $q_\lambda(z|x)$ be the distribution of z predicted by our encoder deep network. We want it to match the true distribution $p(z|x)$. We want the distribution approximated by the deep network has little divergence from the true distribution. i.e. we want to optimize λ with the smallest KL divergence.

$$D_{KL}[(q_\lambda(z|x)||p(z|x)] = E[logq_\lambda(z|x) - logp(z|x)]$$

- Apply:

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)}$$

$$\begin{aligned}
D_{KL}[q_\lambda(z|x)\|p(z|x)] &= \mathbb{E}_q[\log q_\lambda(z|x) - \log \frac{p(x|z)p(z)}{p(x)}] \\
&= \mathbb{E}_q[\log q_\lambda(z|x) - \log p(x|z) - \log p(z) + \log p(x)] \\
&= \mathbb{E}_q[\log q_\lambda(z|x) - \log p(x|z) - \log p(z)] + \log p(x) \\
D_{KL}[q_\lambda(z|x)\|p(z|x)] - \log p(x) &= \mathbb{E}_q[\log q_\lambda(z|x) - \log p(x|z) - \log p(z)] \\
\log p(x) - D_{KL}[q_\lambda(z|x)\|p(z|x)] &= \mathbb{E}_q[\log p(x|z) - (\log q_\lambda(z|x) - \log p(z))] \\
&= \mathbb{E}_q[\log p(x|z)] - \mathbb{E}_q[\log q_\lambda(z|x) - \log p(z))] \\
&= \mathbb{E}_q[\log p(x|z)] - D_{KL}[q_\lambda(z|x)\|p(z)]
\end{aligned}$$

Cost Function in Detail

- Define the term ELBO (Evidence lower bound) as:

$$ELBO(\lambda) = E_q[logp(x|z)] - D_{KL}[q_\lambda(z|x)||p(z)]$$

$$logp(x) - D_{KL}[q_\lambda(z|x)||p(z|x)] = ELBO(\lambda)$$

- We call ELBO the evidence lower bound because:

$$logp(x) - D_{KL}[q_\lambda(z|x)||p(z|x)] = ELBO(\lambda)$$

$logp(x) \geq ELBO(\lambda)$ since KL is always positive

- Here, we define our VAE objective function

$$logp(x) - D_{KL}[q_\lambda(z|x)||p(z|x)] = E_q[logp(x|z)] - D_{KL}[q_\lambda(z|x)||p(z)]$$

Cost Function in Detail

- Instead of the distribution $p(x)$, we can model the data x with $\log p(x)$. With the error term, $D_{KL}[q_\lambda(z|x)||p(z|x)]$, we can establish a lower bound ELBO for $\log p(x)$ which in practice is good enough in modeling the data distribution.
- In the VAE objective function, maximize our model probability $\log p(x)$. is the same as maximize $\log p(x|z)$. while minimize the divergence of $D_{KL}[q_\lambda(z|x)||p(z)]$.

Cost Function in Detail

- Maximizing $\log p(x|z)$ can be done by building a decoder network and maximize its likelihood. So with an encoder $q_{\emptyset}(z|x)$, a decoder $p_{\theta}(z|x)$, our objective become optimizing:

$$ELBO(\theta, \emptyset) = E_{q\theta(z|x)}[\log(p_{\theta}(x_i|z))] - D_{KL}[q_{\emptyset}(z|x)||p(z)]$$

- We can apply a constrain to $p(z)$ such that we can evaluate $D_{KL}[q_{\emptyset}(z|x)||p(z)]$ easily. In VAE, we use $p(z) = N(0,1)$. For optimal solution, we want $q_{\emptyset}(z|x)$ to be as close as $N(0,1)$.

Cost Function in Detail

- In VAE, we model $q_{\emptyset}(z|x)$ as $N(\mu, \Sigma)$

$$
\begin{aligned}
D_{KL}[q_\phi(z|x)\|p(z)] &= D_{KL}[N(\mu,\Sigma)\|N(0,1)] \\
&= \frac{1}{2}\left(\mathrm{tr}(\Sigma) + \mu^T\mu - k - \log\det(\Sigma)\right) \\
&= \frac{1}{2}\left(\sum_k \Sigma + \sum_k \mu^2 - \sum_k 1 - \log\prod_k \Sigma\right) \\
&= \frac{1}{2}\left(\sum_k \Sigma(X) + \sum_k \mu^2(X) - \sum_k 1 - \sum_k \log\Sigma(X)\right) \\
&= \frac{1}{2}\sum_k\left(\Sigma + \mu^2 - 1 - \log\Sigma\right)
\end{aligned}
$$