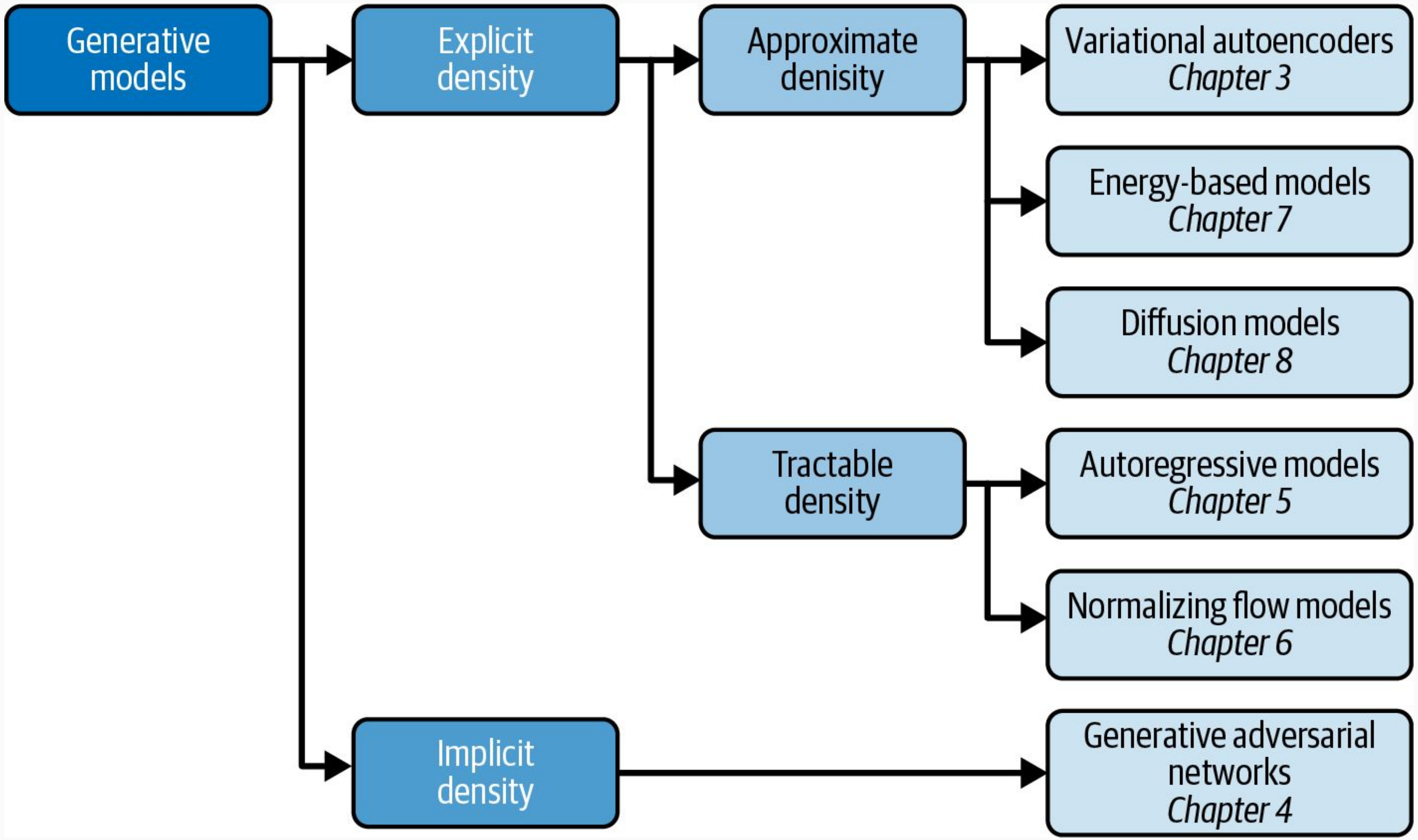
Autoencoders

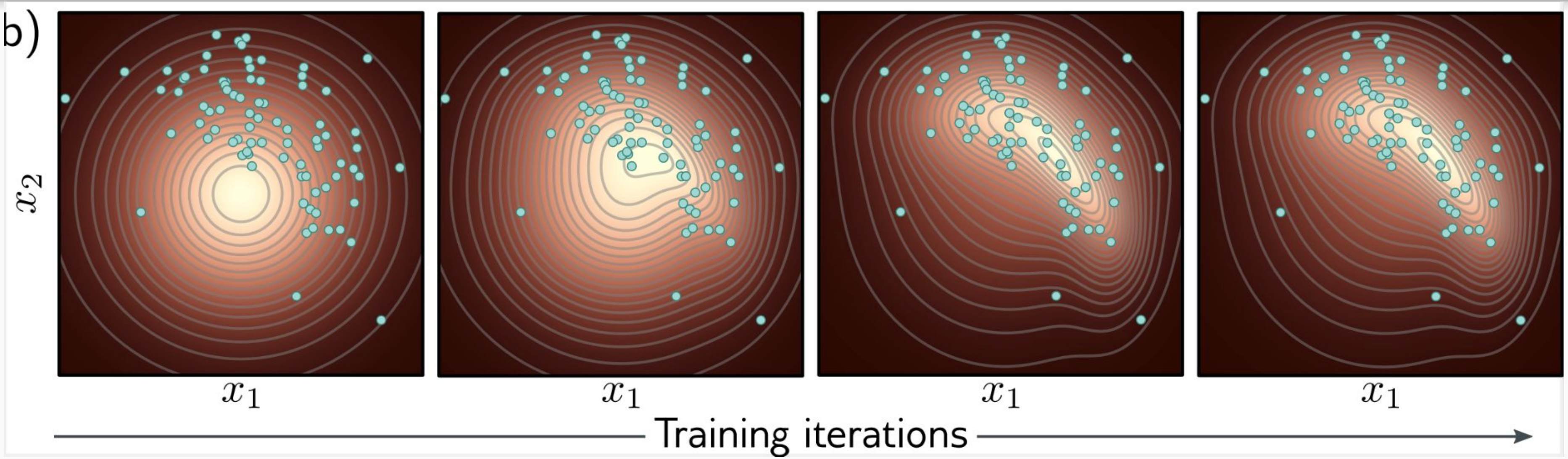
Vijay Raghavan

Agenda

Generative models -> Explicit Density -> Approximate Density -> VAE


Generative models
Explicit density
Approximate denisity
Variational autoencoders *Chapter 3*
Energy-based models *Chapter 7*
Diffusion models *Chapter 8*
Tractable density
Autoregressive models *Chapter 5*
Normalizing flow models *Chapter 6*
Implicit density
Generative adversarial networks *Chapter 4*

Probabilistic models - Variational autoencoders



- Probabilistic models such as variational autoencoders learn a probability distribution over the training data.
- As training proceeds (left to right), the likelihood of the real examples increases under this distribution, which can be used to draw new samples and assess the probability of new data points.

Prince, S. J. (2023). *Understanding Deep Learning*. MIT press.

Theoretical Foundation

Latent variable models

- These models take an indirect approach to describing a probability distribution ***Pr(x)*** over a multi-dimensional variable ***x***.
- Instead of directly writing the expression for ***Pr(x)***, they model a joint distribution ***Pr(x, z)*** of the data ***x*** and a latent variable ***z***.
- They describe the probability of ***Pr(x)*** as a marginalization of this joint probability so that:

$$Pr(\mathbf{x}) = \int Pr(\mathbf{x}, \mathbf{z}) d\mathbf{z}.$$

Latent variables and Prior

The joint probability ***Pr(x, z)*** is broken down using the rules of conditional probability into

- the likelihood of the data with respect to the latent variables term ***Pr(x|z)***
- and the prior ***Pr(z)***

$$Pr(\mathbf{x}) = \int Pr(\mathbf{x}|\mathbf{z})Pr(\mathbf{z})d\mathbf{z}.$$

Prince, S. J. (2023). *Understanding Deep Learning*. MIT press.

1D mixture of Gaussians

- The latent variable ***z*** is discrete, and the prior ***Pr(z)*** is a categorical distribution with one probability $\boldsymbol{\lambda_n}$ for each possible value of ***z***.
- The likelihood ***Pr(x|z = n)*** of the data ***x*** given that the latent variable ***z*** takes value n is normally distributed with mean $\boldsymbol{\mu_n}$ and variance $\boldsymbol{\sigma^2_n}$:

$$
\begin{aligned}
Pr(z=n) &= \lambda_n \\
Pr(x|z=n) &= \text{Norm}_x\left[\mu_n, \sigma_n^2\right]
\end{aligned}
$$

Example: Gaussian Mixtures

We want to model complex data that has multiple peaks or blobs.

- For example, data with groups of points clustered in a few spots.
- To do this, we'll use a mixture of simpler bell curve distributions or Gaussians.
- Imagine we have 3 Gaussians - Gaussian 1, Gaussian 2, and Gaussian 3.
  - Each one looks like a bell curve with one peak.
  - We take these basic building blocks and mix them together to get a complex multi-peak distribution.

Core Idea

- Each Gaussian has a mean (center point) and variance (spread) that determines its shape.
- Each Gaussian gets a mixing weight that says how much it contributes.
    - For example, Gaussian 1 gets 70% weight, Gaussian 2 gets 20% weight, Gaussian 3 gets 10%.
- We add up these weighted Gaussians to get the full mixture distribution.
- The different means and variances cause multiple peaks and blobs.

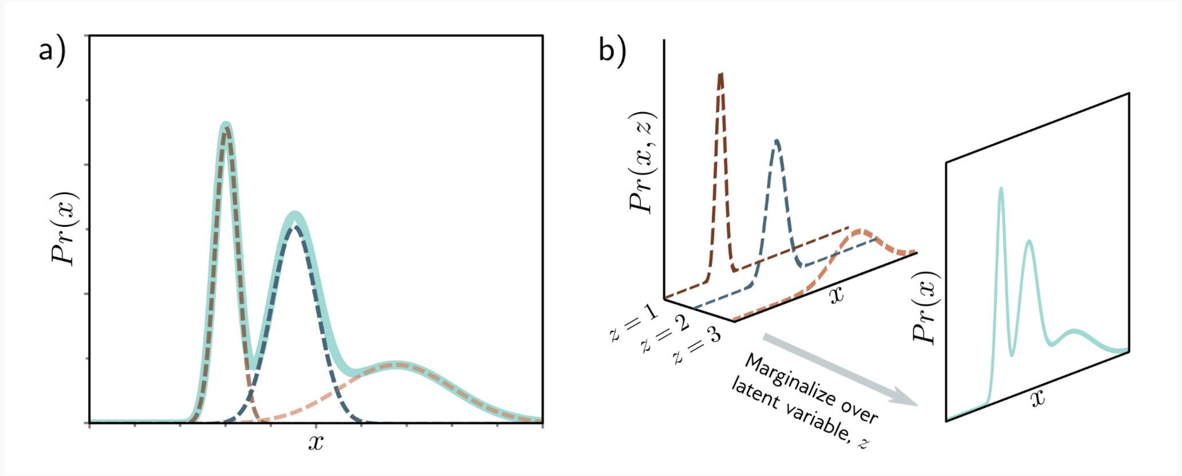
In the end, a mixture model lets us build complex data distributions through weighted combinations of simpler single-peak Gaussians. The more Gaussians we mix, the more complex shapes we can make.

Prince, S. J. (2023). *Understanding Deep Learning*. MIT press.

Mixture of Gaussians (MoG)

a) The MoG describes a complex probability distribution (cyan curve) as a weighted sum of Gaussian components (dashed curves).

b) This sum is the marginalization of the joint density *Pr(x, z)* between the continuous observed data *x* and a discrete latent variable *z*.



Likelihood *Pr(x)* is given by the marginalization over the latent variable z

- The latent variable is discrete, so we sum over its possible values to marginalize.

$$
\begin{aligned}
Pr(x) &= \sum_{n=1}^{N} Pr(x, z = n) \\
&= \sum_{n=1}^{N} Pr(x|z = n) \cdot Pr(z = n) \\
&= \sum_{n=1}^{N} \lambda_n \cdot \mathrm{Norm}_x\left[\mu_n, \sigma_n^2\right].
\end{aligned}
$$

Multivariate data

- Multivariate data has more than 2 variables. For example, an image with pixel values for red, green, blue channels.
- So in multivariate data, each data point is a vector with multiple values, rather than just a single scalar. This allows it to encode more complex interactions between variables.
- In the context of the latent variable model:
  - Both the observed data ***x*** and the latent variables ***z*** are multivariate - meaning they are not single numbers but vectors with multiple continuous dimensions.
  - This allows them to represent more complex data like images or other structured high-dimensional datasets, rather than just 1D data.

Simple low-D latent space -> Flexible neural network generator -> Complex high-D distribution

- We have some complex, messy data in a high-dimensional space - a dataset of images.
- We want to discover some hidden low-dimensional structure underlying this complex data - latent space.
- Assume there is a continuous random latent variable ***z*** in this low-D space, and ***z*** has a spherical Gaussian (bell curve).
- Then there is a generator function ***f*** that maps points ***z*** from the latent space into the data space.
  - This function ***f*** adds all the complexity needed to match the structure of the real data.
  - By pushing points through ***f***, simple low-D points get turned into complex structured high-D data. For example plain samples turn into detailed images.
  - The mapping ***f*** is highly flexible because it's represented by a neural network. By marginalizing over all values of ***z*** weighted by ***P(z)***, we get an extremely complex distribution over data space.
- The model discovers the latent structure and uses it generate new samples.

Nonlinear latent variable model

In the nonlinear latent variable model, both the data ***x*** and the latent variable ***z*** are multivariate. The prior ***Pr(z)*** is a standard multivariate normal:

$$Pr(\mathbf{z}) = \text{Norm}_{\mathbf{z}}[\mathbf{0}, \mathbf{I}]$$

The likelihood ***Pr(x|z, ϕ)*** is also normally distributed; its mean is a nonlinear function ***f[z, ϕ]*** of the latent variable, and its covariance ***σ²I*** is spherical:

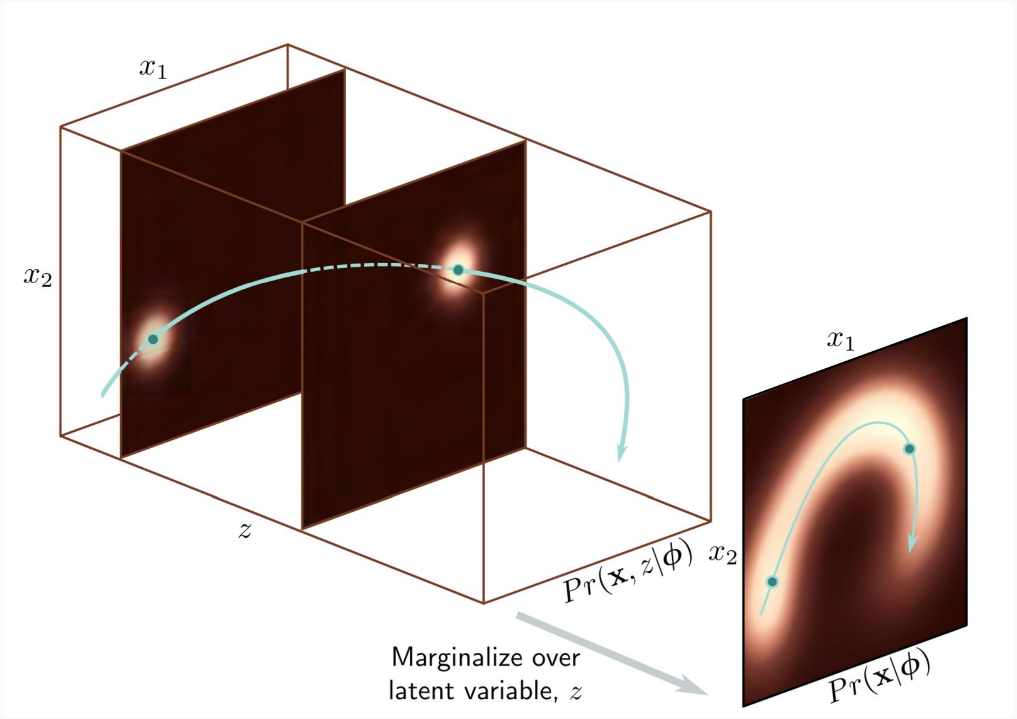
$$Pr(\mathbf{x}|\mathbf{z}, \phi) = \text{Norm}_{\mathbf{x}}\left[\mathbf{f}[\mathbf{z}, \phi], \sigma^2\mathbf{I}\right]$$

Discovering *f* using Deep Neural Network

- The function ***f[z, ϕ]*** is described by a deep network with parameters ***ϕ***.
- The latent variable ***z*** is lower dimensional than the data ***x***.
- The model ***f[z, ϕ]*** describes the important aspects of the data, and the remaining unmodeled aspects are ascribed to the noise ***σ²I***.
- The data probability ***Pr(x|ϕ)*** is found by marginalizing over the latent variable **z**.
- This can be viewed as an infinite weighted sum (i.e., an infinite mixture) of spherical Gaussians with different means, where the weights are ***Pr(z)*** and the means are the network outputs ***f[z, ϕ]***

$$
\begin{aligned}
Pr(\mathbf{x}|\phi) &= \int Pr(\mathbf{x},\mathbf{z}|\phi)d\mathbf{z} \\
&= \int Pr(\mathbf{x}|\mathbf{z},\phi)\cdot Pr(\mathbf{z})d\mathbf{z} \\
&= \int \text{Norm}_{\mathbf{x}}\left[\mathbf{f}[\mathbf{z},\phi],\sigma^2\mathbf{I}\right]\cdot \text{Norm}_{\mathbf{z}}\left[\mathbf{0},\mathbf{I}\right]d\mathbf{z}.
\end{aligned}
$$

Nonlinear latent variable model



- ***Pr(x)*** (right) is created as the marginalization of the joint distribution ***Pr(x, z)*** (left) over the latent variable ***z***.
- For each ***z***, the distribution over ***x*** is a spherical Gaussian (two slices shown) with a mean ***f[z, ϕ]*** that is a nonlinear function of ***z*** and depends on parameters ***ϕ***.
- The distribution ***Pr(x)*** is a weighted sum of these Gaussians

Ancestral Sampling

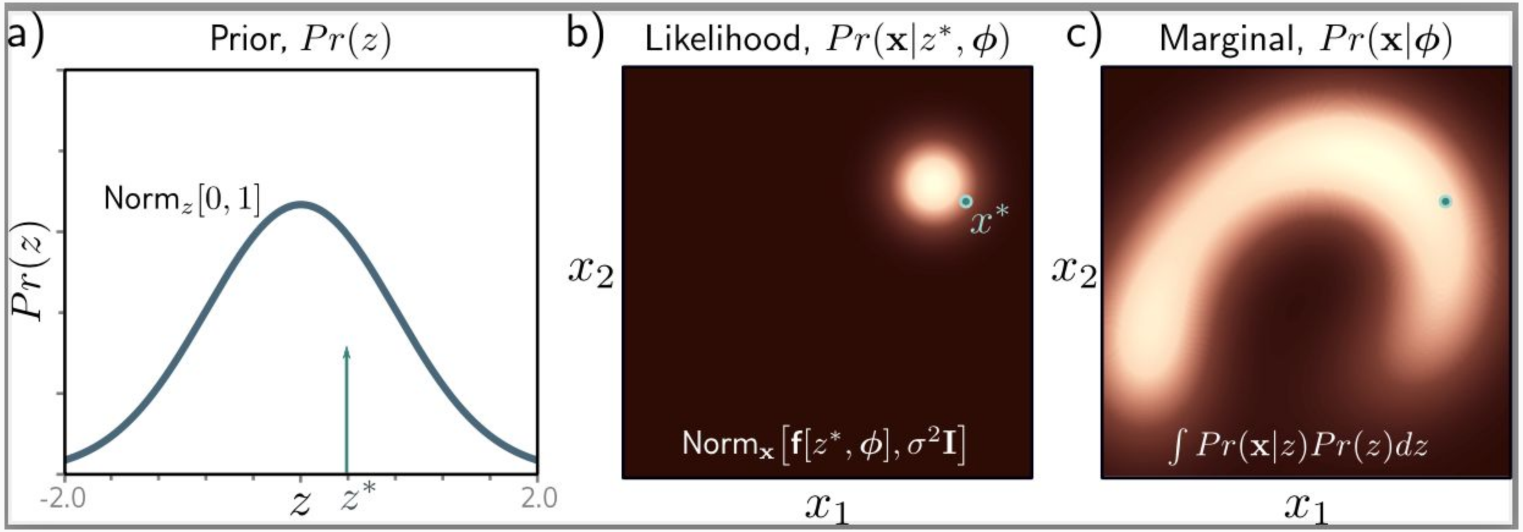
- When the joint distribution can be factored into a series of conditional probabilities, we can generate samples using ancestral sampling.
- The basic idea is to generate a sample from the root variable(s) and then sample from the subsequent conditional distributions based on this instantiation.
- This process is known as ancestral sampling and is easiest to understand with an example. Consider a joint distribution over three variables, ***x, y, and z***, where the distribution factors as:

$$Pr(x, y, z) = Pr(x)Pr(y|x)P\ (z|y).$$

- To sample from this joint distribution, we first draw a sample x^* from $Pr(x)$.
- Then we draw a sample y^* from $Pr(y|x^*)$.
- Finally, we draw a sample z^* from $Pr(z|y^*)$.

Generation from nonlinear latent variable model

- A new example x* can be generated using ancestral sampling.
- We draw ***z**** from the prior ***Pr(z) (fig a)*** and pass this through the network ***f[z*, ϕ] (fig b)***to compute the mean of the likelihood ***Pr(x|z* , ϕ) (fig c)*** from which we draw ***x****.
- Both the prior and likelihood are normal distributions, so this is straightforward



Training

- To train the model, we maximize the log-likelihood over a training dataset $\{ \boldsymbol{x}_i \}_{i=1}^{I}$ with respect to the model parameters.
- For simplicity, we assume that the variance term $\boldsymbol{\sigma^2}$ in the likelihood expression is known and concentrate on learning $\boldsymbol{\phi}$:

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\mathrm{argmax}} \left[\sum_{i=1}^{I} \log\left[Pr(\mathbf{x}_i|\boldsymbol{\phi}) \right] \right],$$

where:

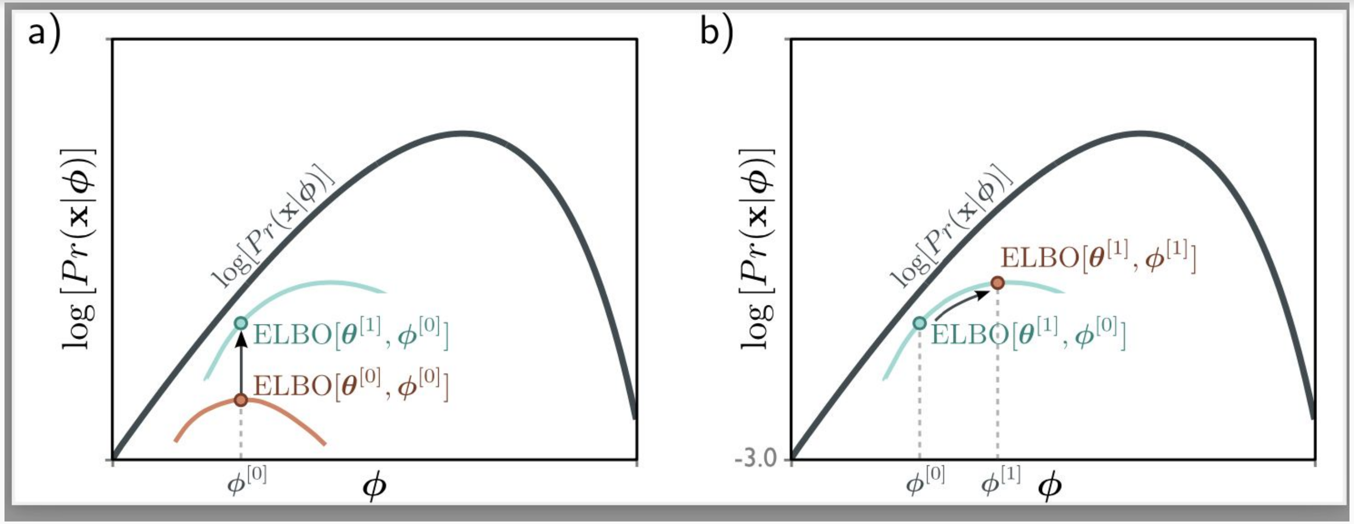
$$Pr(\mathbf{x}_i|\boldsymbol{\phi}) = \int \mathrm{Norm}_{\mathbf{x}_i}[\mathbf{f}[\mathbf{z},\boldsymbol{\phi}], \sigma^2\mathbf{I}] \cdot \mathrm{Norm}_{\mathbf{z}}[\mathbf{0},\mathbf{I}] d\mathbf{z}.$$

- Unfortunately, this is intractable.
- There is no closed-form expression for the integral and no easy way to evaluate it for a particular value of $\boldsymbol{x}$.

Evidence lower bound (ELBO)

- To make progress, we define a lower bound on the log-likelihood.
- This is a function that is always less than or equal to the log-likelihood for a given value of $\boldsymbol{\phi}$ and will also depend on some other parameters $\boldsymbol{\theta}$.
- The goal is to maximize the log likelihood ***log [Pr(x|*$\boldsymbol{\phi}$*)]*** (black curve) with respect to the parameters $\boldsymbol{\phi}$**.**
- The ELBO is a function that lies everywhere below the log-likelihood.
- It is a function of both ϕ and a second set of parameters θ.
- For fixed θ, we get a function of ϕ (two colored curves for different values of θ).
- Consequently, we can increase the log-likelihood by either improving the ELBO with respect to
  - the new parameters θ (moving from colored curve to colored curve) or the original parameters ϕ (moving along the current colored curve).

Evidence lower bound (ELBO)



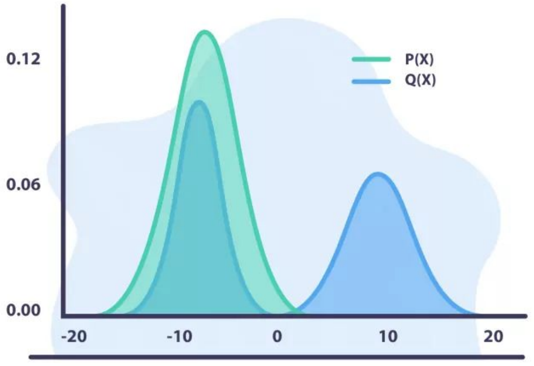
- The goal is to maximize the log- likelihood log[Pr(x|φ)] (black curve) with respect to the parameters φ.
- The ELBO is a function that lies everywhere below the log-likelihood.
- It is a function of both φ and a second set of parameters θ.
- For fixed θ, we get a function of φ (two colored curves for different values of θ).
- Consequently, we can increase the log-likelihood by either improving the ELBO with respect to

  a) the new parameters θ (moving from colored curve to colored curve)

  or b) the original parameters φ (moving along the current colored curve).

Kullback-Leibler (KL) divergence

- The KL divergence is a way to measure how different two probability distributions are from each other.
- Imagine we have two models that try to describe some data - they will each make predictions of the probability of different outcomes.
- We can use the KL divergence between the probability predictions of the two models to get a numerical score for how different the models are.
- It specifically measures the information lost when using one distribution ***(q)*** to represent the other ***(p)***.



Kullback–Leibler Divergence

https://www.aporia.com/glossary/kullback-leibler-divergence/

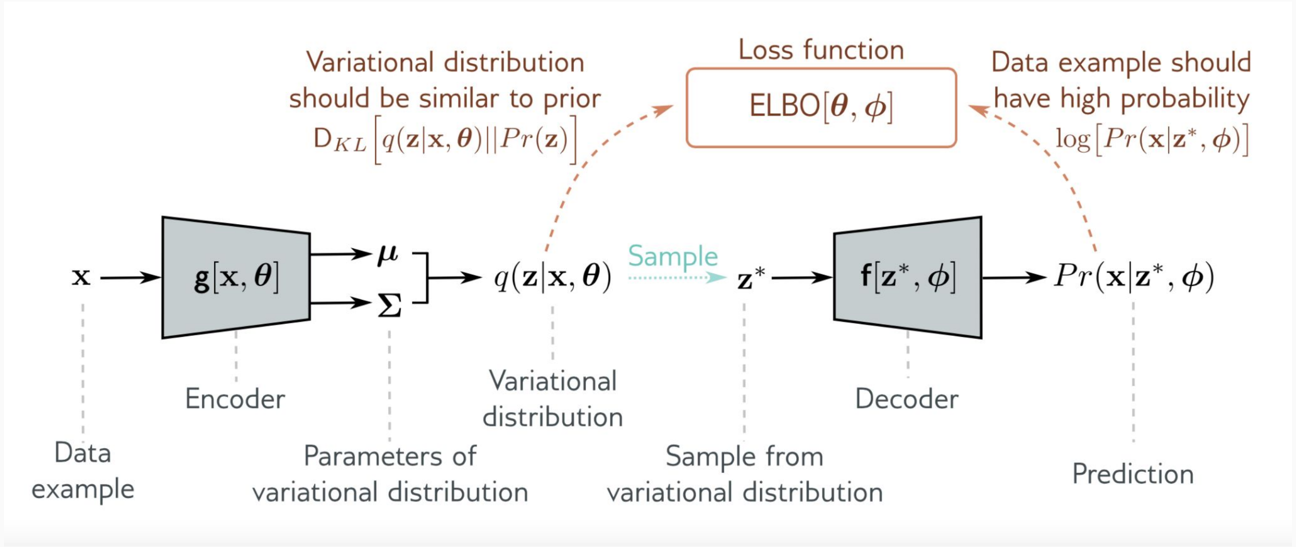
KL divergence - Mechanics

- It works by taking the p distribution, and for each possible outcome, looking at the probability p assigns to that outcome versus the probability q assigns.
- It then sums over all possible outcomes.
- The bigger this sum is, the more "divergent" the two distributions are.
- A key property is that the KL divergence score is always greater than or equal to 0. This means the two distribution predictions can't be more similar than identical.

KL divergence - issue

- One issue that comes up is that if there is some outcome ***p*** says is possible but ***q*** says is impossible, then the KL divergence score becomes infinite.
- This causes problems in some applications where we want to minimize the KL divergence.
- It means the optimization will sometimes drive ***q's*** predictions to ***0*** incorrectly just to get an infinitely low (but meaningless) score.

Variational Autoencoder


Variational distribution should be similar to prior
$D_{KL}\left[q(\mathbf{z}|\mathbf{x},\boldsymbol{\theta})||Pr(\mathbf{z})\right]$
Loss function
$\text{ELBO}[\boldsymbol{\theta},\phi]$
Data example should have high probability
$\log\left[Pr(\mathbf{x}|\mathbf{z}^*,\phi)\right]$
$\mathbf{x}$
$\mathbf{g}[\mathbf{x},\boldsymbol{\theta}]$
$\boldsymbol{\mu}$
$\boldsymbol{\Sigma}$
$q(\mathbf{z}|\mathbf{x},\boldsymbol{\theta})$
Sample
$\mathbf{z}^*$
$\mathbf{f}[\mathbf{z}^*,\phi]$
$Pr(\mathbf{x}|\mathbf{z}^*,\phi)$
Encoder
Variational distribution
Decoder
Data example
Parameters of variational distribution
Sample from variational distribution
Prediction

- The encoder ***g[x, θ]*** takes a training example x and predicts the parameters ***μ, Σ*** of the variational distribution ***q(z|x,θ)***.
- We sample from this distribution and then use the decoder ***f[z, φ]*** to predict the data **x**.
- The loss function is the negative ELBO, which depends on how accurate this prediction is and how similar the variational distribution ***q(z|x,θ)*** is to the prior ***Pr(z)*** .

Prince, S. J. (2023). *Understanding Deep Learning*. MIT press.

ELBO as reconstruction loss minus KL distance to prior

- One way is to consider the bound as reconstruction error minus the distance to the prior:
  - In this formulation, the first term measures the average agreement Pr(x|z, ϕ) of the latent variable and the data - this is termed the reconstruction loss.
  - The second term measures the degree to which the auxiliary distribution q(z|θ) matches the prior.

$$
\begin{aligned}
\mathrm{ELBO}[\boldsymbol{\theta}, \phi] &= \int q(\mathbf{z}|\boldsymbol{\theta}) \log \left[\frac{Pr(\mathbf{x}, \mathbf{z}|\phi)}{q(\mathbf{z}|\boldsymbol{\theta})}\right] d\mathbf{z} \\
&= \int q(\mathbf{z}|\boldsymbol{\theta}) \log \left[\frac{Pr(\mathbf{x}|\mathbf{z}, \phi)Pr(\mathbf{z})}{q(\mathbf{z}|\boldsymbol{\theta})}\right] d\mathbf{z} \\
&= \int q(\mathbf{z}|\boldsymbol{\theta}) \log \left[Pr(\mathbf{x}|\mathbf{z}, \phi)\right] d\mathbf{z} + \int q(\mathbf{z}|\boldsymbol{\theta}) \log \left[\frac{Pr(\mathbf{z})}{q(\mathbf{z}|\boldsymbol{\theta})}\right] d\mathbf{z} \\
&= \int q(\mathbf{z}|\boldsymbol{\theta}) \log \big[Pr(\mathbf{x}|\mathbf{z}, \phi)\big] d\mathbf{z} - \mathrm{D}_{KL}\Big[q(\mathbf{z}|\boldsymbol{\theta}) \Big|\Big| Pr(\mathbf{z})\Big],
\end{aligned}
$$

Variational approximation

- ELBO is tight when $q(z|\theta)$ is the posterior $Pr(z|x, \phi)$.
- In principle, we can compute the posterior using Bayes' rule: $Pr(z|x, \phi) = (Pr(x|z, \phi)Pr(z)) / Pr(x|\phi)$, but in practice, this is intractable because we can't evaluate the data likelihood in the denominator
- One solution is to make a variational approximation:
  - Choose a simple parametric form for $q(z|\theta)$ and use this to approximate the true posterior.
  - Here, we choose a Multivariate normal multivariate normal distribution with mean μ and diagonal covariance Σ.
  - This will not always match the posterior well but will be better for some values of μ and Σ than others.
  - During training, we will find the normal distribution that is "closest" to the true posterior $Pr(z|x)$.
  - This corresponds to minimizing the KL divergence and moving the colored curves upwards.

Encoder - the Neural Network to map data to latent variable

- Since the optimal choice for ***q(z|θ)*** was the posterior ***Pr(z|x)***
- And this depends on the data example ***x***, the variational approximation should do the same, so we choose:

$$q(\mathbf{z}|\mathbf{x},\boldsymbol{\theta}) = \text{Norm}_{\mathbf{z}}\left[\mathbf{g}_{\boldsymbol{\mu}}[\mathbf{x},\boldsymbol{\theta}], \mathbf{g}_{\boldsymbol{\Sigma}}[\mathbf{x},\boldsymbol{\theta}]\right]$$

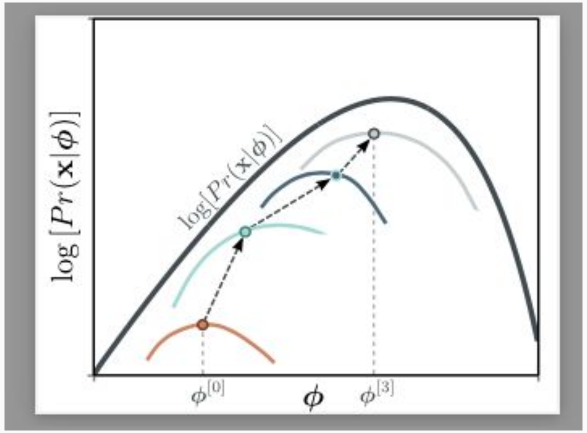
- where ***g[x, θ]*** is a neural network with parameters ***θ*** that predicts the mean ***μ*** and variance ***Σ*** of the normal variational approximation.
- In other words, the encoder ***g[x, θ]*** takes a training example ***x*** and predicts the parameters ***μ, Σ*** of the variational distribution ***q(z|x, θ)***.

Decoder - the Neural Network to map the latent variable to data

- The encoder ***g[x, θ]*** takes a training example x and predicts the parameters ***μ, Σ*** of the variational distribution ***q(z|x, θ)***.
- We sample from this distribution and then use the decoder ***f[z, ϕ]*** to predict the data ***x***.
- The loss function is the negative ELBO, which depends on how accurate the prediction of ***x*** is and how similar the variational distribution ***q(z|x, θ)*** is to the prior ***Pr(z)***

Workflow

- The VAE computes the ELBO as a function of both ϕ and θ.
- To maximize this bound, we run mini-batches of samples through the network and update these parameters with an optimization algorithm such as SGD or Adam.
- The gradients of the ELBO with respect to the parameters are computed as usual using automatic differentiation.
- During this process, we are both moving between the colored curves (changing θ) and along them (changing ϕ).
- During this process, the parameters ϕ change to assign the data a higher likelihood in the nonlinear latent variable model.



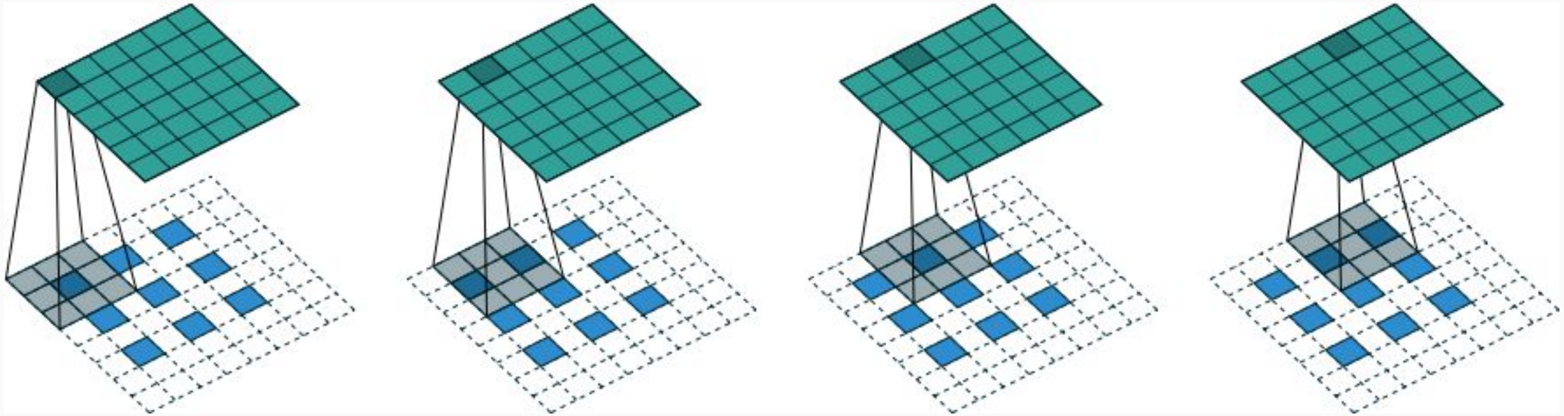
The Autoencoder Architecture

Convolutional transpose layers

A convolutional transpose layer works by taking an input feature map and producing a larger output feature map. This works by sliding a kernel over the input, but with some key differences:

- The strides are fractional - e.g taking bigger jumps when sliding the kernel. This makes the output larger.
- There is implicit upsampling happening with nearest neighbor interpolation to increase the spatial dimensions.
- The output channels from the input feature map is treated as input channels in the deconv layer. And the number of kernels/filters in the layer determines the number of output channels.
- The parameters of a convolutional transpose layer include number of filters, kernel size, strides and padding - just like a conv layer but with the output dimensions increased.

Convolutional Transpose Layers

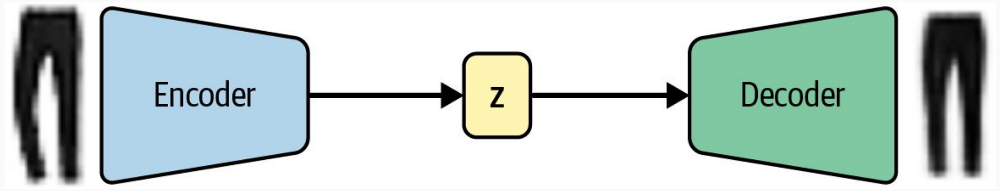


1. Takes smaller spatial input, more channels
2. Fractionally strided convolution to upsample
3. Produces larger spatial output, fewer channels

Infinite Wardrobe

- Your entire wardrobe is organized on the floor of an infinite, two-dimensional space.
- Locations within this space represent the concepts and attributes that can be learned about clothing.
- Over time, you request various items of clothing by location, while your stylist retrieves or "generates" them.
- Through repeated exchanges, you develop a shared semantic understanding of how the wardrobe space encodes knowledge about fashion.
- Eventually, your stylist becomes so adept at leveraging these spatial relationships that he can synthesize entirely new garments, simply by extrapolating from locations never requested before.

Autoencoders for Creative Generation



- **Your role:** Encode clothing items to locations in wardrobe (embedding space)
- **Stylist's role:** Decode locations back to clothing (reconstruction)
- **During Training:**
  - Repeated encode-decode cycle trains the autoencoder model
  - Autoencoder learns to embed inputs and regenerate them
- **After training:**
  - Decoder can take any point in embedding space and generate novel outputs
  - Interpolating between learned embeddings enables creative synthesis

Why Train an Autoencoder?

- Input image --> Encoded to lower-dim embedding --> Decoded to image
- Autoencoder learns to reconstruct original inputs

Seems pointless at first...we already have the images!

- The embedding space captures important features and concepts
- Sampling this latent space allows generating new examples
- Creative synthesis happens by navigating learned embeddings

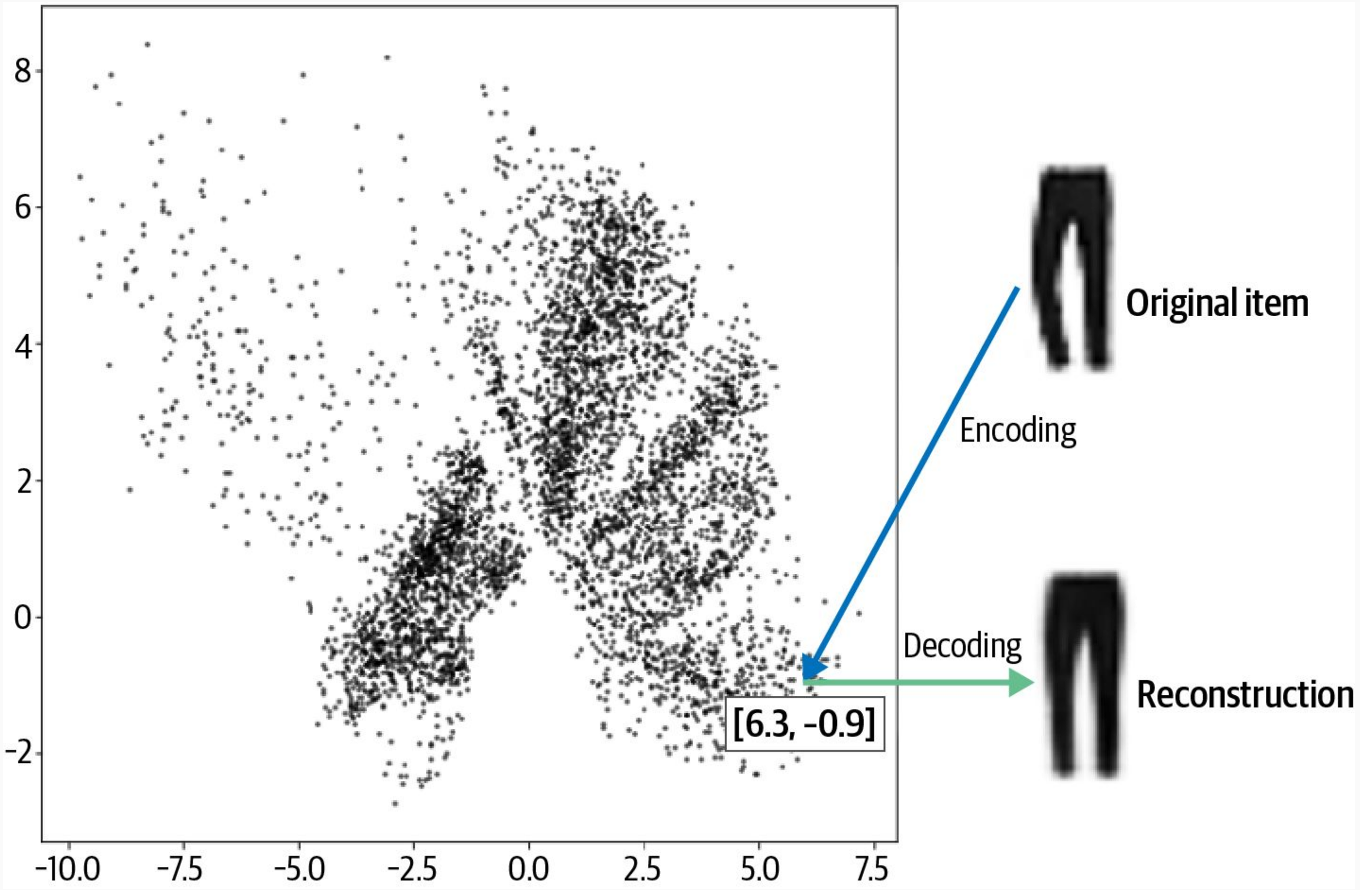
Understanding Embeddings

- Embeddings are a compressed representation in a lower-dimensional latent space
- The autoencoder encoder maps images to this space
- Choosing any point in the latent space allows the decoder to generate a new image
- It has learned conversions between the latent space and viable images
- We embed images in 2D to visualize the space
- But practical autoencoders use higher dimensions to enable greater nuance

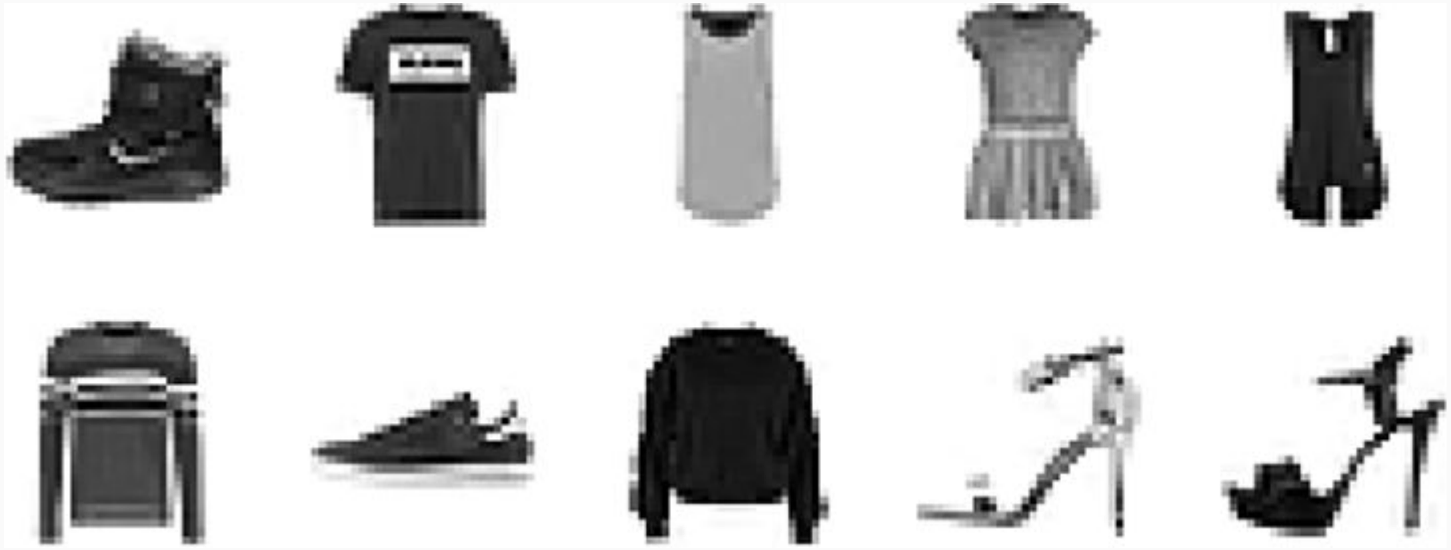
Image embedding - MobileNet

```
-0.118, -0.353, 0.282, 0.401, 0.0786, 0.160, -0.354, -0.369, 0.200, 0.563, -0.359, 0.926, 0.739, 0.600, -0.216, -0.0427,
-0.362, 0.219, -0.282, 0.00, 0.319, -0.372, -0.119, -0.0578, 0.846, 0.975, 0.440, -0.277, 0.418, -0.375, -0.321, -0.263,
-0.308, -0.292, 0.0688, -0.374, 0.0757, 0.471, 0.862, -0.147, 0.440, 0.0947, -0.375, 0.304, -0.223, 0.536, -0.346, -0.373,
2.02, 1.50, -0.375, 0.518, -0.339, 0.182, 0.150, -0.133, -0.127, 0.00, 0.715, 0.00, -0.293, -0.196, -0.279, -0.325, -0.346,
-0.243, -0.285, 0.549, 0.194, -0.336, 0.140, -0.146, -0.306, 0.578, 0.486, 0.242, 0.567, 0.0264, -0.374, -0.373, -0.288,
-0.319, 0.661, -0.338, -0.0939, 1.05, 1.24, 0.326, 1.16, -0.375, 0.183, -0.253, -0.328, -0.364, 0.636, 0.157, 0.237, -0.206,
-0.298, 0.212, 0.00, 0.656, -0.103, -0.304, -0.248, -0.187, 0.697, 0.623, -0.319, -0.371, -0.242, -0.0247, -0.143, 0.594,
0.569, -0.287, -0.232, 0.125, 0.582, 0.0652, 0.612, 0.00, -0.0112, 0.00, 0.00, 0.405, -0.258, 0.501, 0.0260, -0.257, 0.0777,
-0.373, -0.300, -0.219, -0.234, -0.373, -0.191, -0.375, -0.117, -0.334, -0.309, 0.0106, -0.157, 0.00, -0.369, 0.629, -0.120,
0.00, 0.429, 0.00, -0.358, -0.369, -0.187, -0.320, 0.513, 2.23, 0.0417, -0.340, -0.298, -0.305, -0.315, 0.154, -0.366, 0.0586,
1.19, -0.331, 1.61, 0.938, -0.189, 0.324, -0.113, -0.288, -0.271, 0.327, -0.371, 0.389, 0.00, -0.366, 0.00, 0.00, -0.319,
-0.374, -0.315, 0.00, -0.348, -0.334, -0.233, -0.220, -0.343, 0.240, -0.161, -0.0506, 0.0830, 0.266, 0.0587, -0.0329, -0.193,
-0.283, -0.222, 0.256, -0.350, -0.249, 0.140, -0.311, -0.373, -0.118, 0.436, 0.00, 0.483, -0.363, -0.182, 0.00674, 0.0956,
-0.374, 1.02, -0.148, -0.307, 1.17, -0.152, -0.263, -0.206, -0.351, 0.121, 0.389, 0.305, -0.0997, 0.727, -0.367, 1.37, -0.257,
0.0761, -0.298, 0.151, -0.303, -0.124, -0.204, -0.202, -0.154, 1.19, 0.359, 1.15, -0.374, 0.364, 0.189, 1.07, -0.363, 0.183,
-0.242, -0.110, -0.356, -0.00924, -0.00658, -0.373, -0.187, 0.00, 0.256, 0.552, -0.344, -0.306, 0.00, 0.548, -0.101, -0.238,
-0.360, -0.0768, 1.28, 0.317, -0.178, -0.330, -0.277, -0.316, -0.370, -0.135, -0.267, -0.158, -0.168, -0.280, 0.170, -0.171,
2.28, 0.775, -0.0721, -0.312, 0.464, -0.365, 0.180, -0.375, -0.130, -0.371, 0.00, 0.585, 0.453, -0.240, 0.333, 0.441, -0.374,
-0.353, -0.0798, -0.301, -0.200, -0.357, -0.112, 0.00, 0.131, -0.327, -0.319, -0.0352, 0.0745, 0.00, 0.00, 0.00, 0.573, -0.203,
-0.125, -0.362, -0.240, -0.374, 0.00, 0.573, -0.269, -0.346, 0.162, -0.0433, 0.0186, 0.00, 0.0507, 0.178, -0.127, 0.00, -0.189,
0.150, 1.62, 0.866, -0.116, 0.0939, -0.161, 0.183, 0.00, -0.288, -0.370, 0.619, 0.00, 1.08, -0.0366, -0.354, -0.367, 0.113,
-0.0860, 1.40, 0.219, 1.76, -0.374, -0.282, 0.00, -0.371, 0.123, -0.375, -0.259, 0.00, -0.202, -0.276, 0.156, -0.230, -0.202,
0.248, -0.210, 0.174, 1.76, 0.605, -0.373, 0.184, -0.149, 0.270, 0.534, -0.341, -0.228, -0.367, 0.188, 0.00, 0.00879, -0.170,
-0.00450, -0.134, -0.0854, -0.345, -0.339, -0.364, -0.0487, 0.736, 0.00, -0.368, -0.290, -0.331, -0.362, 0.00, -0.305, -0.371,
0.222, 0.172, -0.340, 0.00, 0.454, 0.0375, 0.00, -0.184, 1.19, -0.249, 0.893, 0.00, 0.394, -0.269, -0.0102, -0.329, -0.347,
0.355, -0.374, 0.0853, -0.195, 0.0141, 0.211, 0.00, -0.0637, 0.00, -0.143, 0.588, 0.00, -0.312, 0.463, 0.356, -0.290, -0.372,
0.469, 0.568, 0.00, -0.292, 0.457, -0.339, -0.0837, -0.314, -0.283, 0.280, -0.371, 1.90, -0.374, 0.490, 0.310, 0.423, 0.969,
-0.0366, 0.00, -0.145, -0.328, -0.263, 0.123, -0.294, 0.146, 1.40, -0.280, 0.284, 0.847, 0.741, 0.409, -0.165, -0.349, -0.144,
0.127, -0.313, -0.0239, -0.372, -0.0903, 0.773, -0.353, -0.0842, -0.0827, -0.133, -0.172, 0.0450, 0.00, -0.344, -0.368, 0.0432,
1.03, -0.343, 0.781, -0.0542, 0.111, -0.317, -0.0436, -0.146, 0.615, -0.374, 0.00, -0.355, 0.170, -0.318, -0.363, -0.374,
-0.374, -0.273, 0.142, 0.760, -0.232, -0.0468, 0.00, -0.108, -0.355, -0.300, -0.0637, -0.356, -0.282, -0.205, 0.00, 0.128,
-0.363, 0.594, 0.0208, 0.00, -0.235, -0.188, 0.712, 1.25, -0.109, -0.274, 0.238, 0.792, -0.356, -0.366, 0.316, 0.373, 0.0795,
-0.374, 0.505, -0.123, -0.311, -0.324, -0.267, 0.390, -0.164, -0.368, 0.171, -0.264, 0.186, 0.985, 0.00996, -0.331, -0.306,
0.00618, -0.136, 0.590, 0.463, -0.128, -0.371, 0.00747, -0.325, 1.71, -0.375, -0.208, -0.373, -0.101, -0.310, -0.261, 0.00,
0.00, 0.841, 0.163, 0.460, -0.369, 0.00, -0.136, 0.917, 0.886, 0.373, -0.0490, -0.179, -0.151, 0.00, -0.323, 0.0372, -0.319,
0.609, 0.179, -0.365, 0.0811, -0.364, -0.0856, -0.375, 0.00, 0.210, 0.425, -0.0221, 0.479, -0.352, -0.222, -0.361, 0.0239,
-0.358, -0.308, -0.317, 1.92, -0.179, -0.202, -0.125, -0.282, -0.181, 0.114, -0.0338, 0.00, 0.165, 0.000381, -0.304, 0.461,
0.186, 1.09, 0.767, -0.372, 0.550, -0.100, -0.284, 0.321, -0.303, 0.132, -0.109, 1.22, -0.193, -0.231, -0.375, 0.283, 0.379,
0.682, 0.114, -0.370, -0.284, 0.529, -0.196, -0.361, -0.279, -0.326, 0.210, -0.241, -0.360, -0.155, 0.111, 0.226, -0.244,
0.297, -0.139, -0.295, 0.493, -0.320, -0.175, -0.375, 0.00, 0.00, -0.335, 0.214, -0.242, -0.367, -0.331, -0.0671, -0.327,
-0.285, -0.274, -0.174, -0.0814, -0.0636, 0.454, 1.04, -0.372, -0.375, 0.0319, -0.218, 0.289, -0.259, 0.463, -0.109, -0.359,
0.129, -0.362, -0.349, -0.249, -0.332, 0.350, 1.17, -0.215, -0.260, 0.717, 0.703, -0.358, -0.167, 0.568, -0.172, -0.372,
-0.197, 0.00, -0.230, -0.310, -0.288, -0.375, 0.498, -0.374, -0.351, -0.370, -0.221, 0.645, 0.0318, -0.345, 0.221, -0.245,
-0.375, -0.163, 0.216, -0.304, 0.00, -0.375, -0.153, -0.364, -0.373, 0.150, -0.315, -0.350, -0.241, 0.815, -0.342, 0.0622,
-0.200, 0.00, -0.374, -0.367, 0.0772, -0.239, -0.346, -0.0626, -0.271, -0.0884, -0.335, -0.223, 0.774, -0.0182, -0.329, 0.00,
0.128, 0.434, 0.651, -0.308, -0.253, 0.109, -0.141, -0.365, -0.0470, 0.354, -0.373, -0.288, 0.0433, 0.285, 0.406, -0.362,
0.559, -0.302, 0.929, 0.00, 0.00, -0.353, -0.0834, -0.290, -0.0894, 0.00, -0.165, 0.00, -0.108, 0.00, -0.285, -0.353, -0.368,
0.270, 0.213, -0.375, -0.204, -0.372, -0.271, -0.372, 0.651, -0.356, -0.301, 0.0944, -0.346, 0.548, 0.00, -0.345, 0.00, -0.121,
-0.238, 1.14, -0.182, 0.242, 0.608, -0.230, -0.237, -0.339, -0.272, -0.369, -0.0903, -0.0536, -0.188, -0.306, 0.107, 0.549,
-0.253, -0.244, 0.586, 0.960, 0.860, -0.153, -0.310, -0.301, -0.325, 0.0145, 0.00, -0.350, -0.0564, 1.51, -0.280, 0.473,
-0.103, -0.243, 0.112, 0.394, -0.0399, -0.0858, -0.211, 0.00, -0.205, 0.359, 0.308, -0.0726, -0.250, 0.00, 0.650, 0.00, -0.234,
-0.123, -0.0537, 0.00, -0.356, 0.111, -0.374, 0.00, -0.322, -0.206, 0.00, -0.230, 1.83, -0.0933, -0.365, -0.240, -0.365,
-0.347, -0.0452, -0.354, -0.263, -0.372, -0.0656, -0.264, 1.02, -0.357, 0.381, -0.137, 1.56, -0.0332, -0.337, -0.239, -0.157,
1.01, 2.24, -0.350, 0.258, -0.371, 0.0499, 0.779, -0.331, -0.374, -0.187, -0.368, -0.147, 0.0966, 0.748, -0.0729, 0.00, 1.21,
-0.114, -0.161, -0.297, 0.203, 0.806, -0.374, -0.199, -0.0475, -0.261, -0.357, -0.164, -0.201, -0.359, -0.277, -0.363, 0.00,
0.0801, -0.321, -0.327, -0.342, -0.265, -0.369, -0.198, -0.118, 0.386, -0.00678, -0.171, 0.330, -0.138, -0.326, -0.325, -0.132,
0.760, 0.00, -0.164, -0.218, -0.252, -0.237, 0.0482, -0.356, 0.429, -0.290, 0.0287, -0.179, -0.194, 0.175, -0.367, -0.347,
-0.292, 0.431, -0.321, -0.336, -0.196, -0.150, 0.137, -0.0450, -0.357, -0.300, -0.374, 0.475, -0.268, 1.42, -0.135, -0.00570,
-0.316, 1.01, 0.356, -0.152, -0.345, -0.372, -0.358, 0.753, 0.438, -0.367, 0.193, -0.155, -0.180, -0.360, -0.319, 0.00,
-0.0797, -0.360, 0.00, -0.293, -0.0643, -0.337, -0.149, -0.138, 0.338, -0.291, -0.280, -0.0430, 0.561, -0.375, 0.494, -0.354,
-0.243, -0.240, -0.298, -0.134, 0.0789, -0.338, 0.438, 0.00, -0.270, -0.351, 0.0984, -0.197, -0.294, 0.521, 0.0840, -0.362,
-0.363, -0.0324, 0.0127, -0.187, -0.0913, 0.592, 0.889, -0.218, 1.29, 0.0529, -0.354, 0.00, -0.121, 0.244, 0.644, 0.0901,
-0.375, -0.328, 0.179, -0.212, -0.307, -0.296, -0.00977, -0.0960
```

Autoencoder for Wardrobe

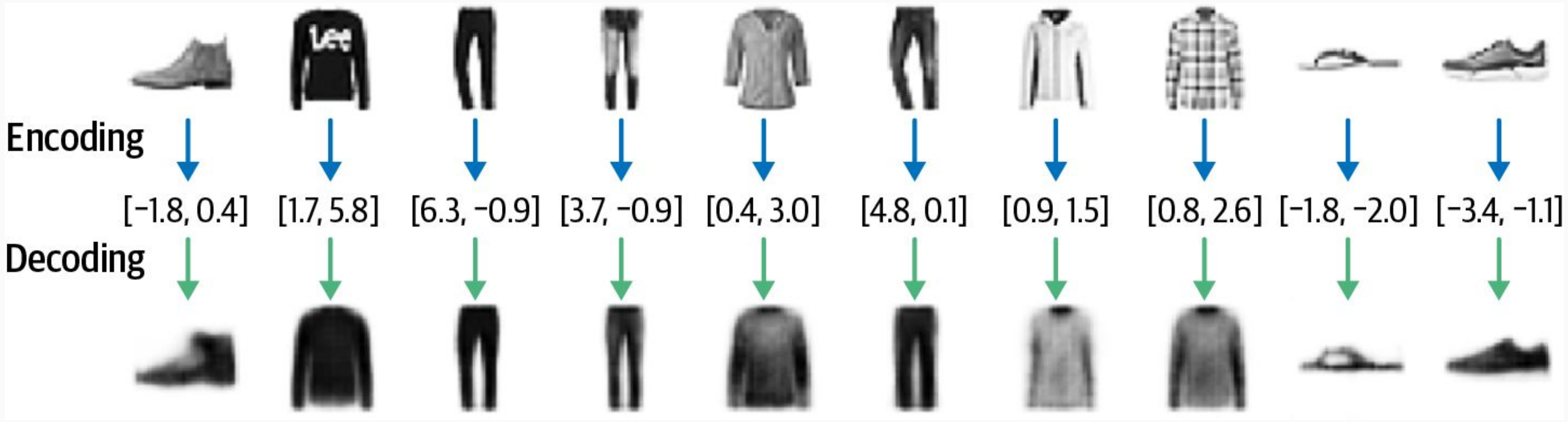


Fashion-MNIST Dataset



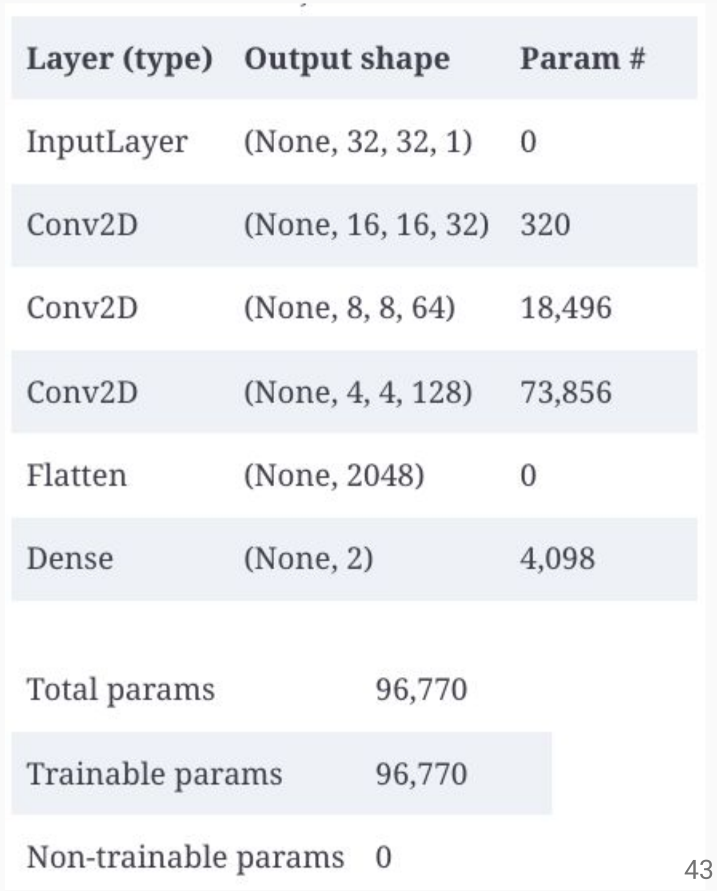
https://www.tensorflow.org/datasets/catalog/fashion_mnist

Reconstructing images using the autoencoder



The Encoder

- We first create an Input layer for the image and pass this through three Conv2D layers in sequence, each capturing increasingly high-level features.
- We use a stride of 2 to halve the size of the output of each layer, while increasing the number of channels.
- The last convolutional layer is flattened and connected to a Dense layer of size 2, which represents our two-dimensional latent space.



Layer (type)	Output shape	Param #
InputLayer	(None, 32, 32, 1)	0
Conv2D	(None, 16, 16, 32)	320
Conv2D	(None, 8, 8, 64)	18,496
Conv2D	(None, 4, 4, 128)	73,856
Flatten	(None, 2048)	0
Dense	(None, 2)	4,098

Total params	96,770
Trainable params	96,770
Non-trainable params	0

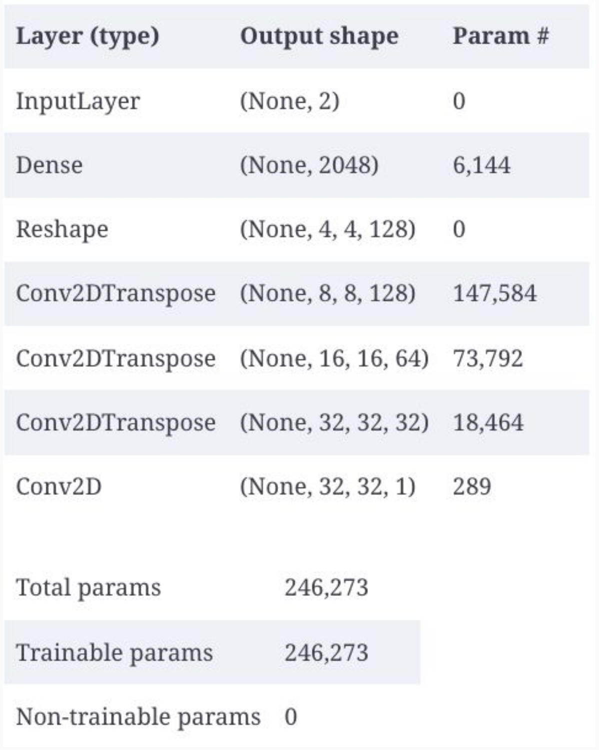
The Encoder - Code

1. Define the Input layer of the encoder (the image).
2. Stack Conv2D layers sequentially on top of each other.
3. Flatten the last convolutional layer to a vector.
4. Connect this vector to the 2D embeddings with a Dense layer.
5. The Keras Model that defines the encoder—a model that takes an input image and encodes it into a 2D embedding.

```
1.   encoder_input = layers.Input(shape=(32, 32, 1), name = "encoder_input")
2.   x = layers.Conv2D(32, (3, 3), strides = 2, activation = 'relu', padding="same")(encoder_input)
     x = layers.Conv2D(64, (3, 3), strides = 2, activation = 'relu', padding="same")(x)
     x = layers.Conv2D(128, (3, 3), strides = 2, activation = 'relu', padding="same")(x)
     shape_before_flattening = K.int_shape(x)[1:]
3.   x = layers.Flatten()(x)
4.   encoder_output = layers.Dense(2, name="encoder_output")(x)
5.   encoder = models.Model(encoder_input, encoder_output
```

The Decoder

The decoder is a mirror image of the encoder—instead of convolutional layers, we use convolution transpose layers



Layer (type)	Output shape	Param #
InputLayer	(None, 2)	0
Dense	(None, 2048)	6,144
Reshape	(None, 4, 4, 128)	0
Conv2DTranspose	(None, 8, 8, 128)	147,584
Conv2DTranspose	(None, 16, 16, 64)	73,792
Conv2DTranspose	(None, 32, 32, 32)	18,464
Conv2D	(None, 32, 32, 1)	289

Total params	246,273
Trainable params	246,273
Non-trainable params	0

The Decoder - Code

1. Define the Input layer of the decoder (the embedding).
2. Connect the input to a Dense layer.
3. Reshape this vector into a tensor that can be fed as input into the first Conv2DTranspose layer.
4. Stack Conv2DTranspose layers on top of each other.
5. The Keras Model that defines the decoder—a model that takes an embedding in the latent space and decodes it into the original image domain.

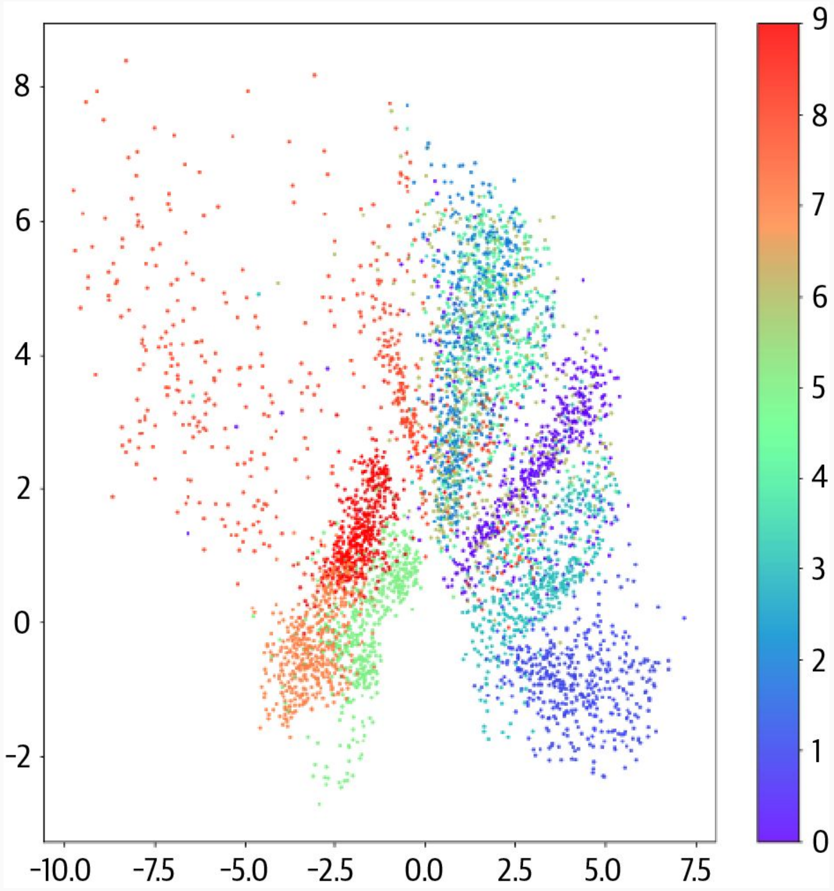
```
1.    decoder_input = layers.Input(shape=(2,), name="decoder_input")
2.    x =
      layers.Dense(np.prod(shape_before_flattening))(decoder_input)
3.    x = layers.Reshape(shape_before_flattening)(x)
4.    x = layers.Conv2DTranspose(
          128, (3, 3), strides=2, activation = 'relu',
      padding="same")(x)
      x = layers.Conv2DTranspose(
          64, (3, 3), strides=2, activation = 'relu',
      padding="same")(x)
      x = layers.Conv2DTranspose(
          32, (3, 3), strides=2, activation = 'relu',
      padding="same")(x)
      decoder_output = layers.Conv2D(
          1,
          (3, 3),
          strides = 1,
          activation="sigmoid",
          padding="same",
          name="decoder_output"
      )(x)
5.    decoder = models.Model(decoder_input, decoder_output)
```

Training and Testing

```python
# Joining the Encoder to the Decoder
autoencoder = Model(encoder_input, decoder(encoder_output))

# Compile the autoencoder
autoencoder.compile(optimizer="adam", loss="binary_crossentropy")

# Training the autoencoder
autoencoder.fit(
    x_train,
    x_train,
    epochs=5,
    batch_size=100,
    shuffle=True,
    validation_data=(x_test, x_test),
)

# Testing
example_images = x_test[:5000]
predictions = autoencoder.predict(example_images)
```

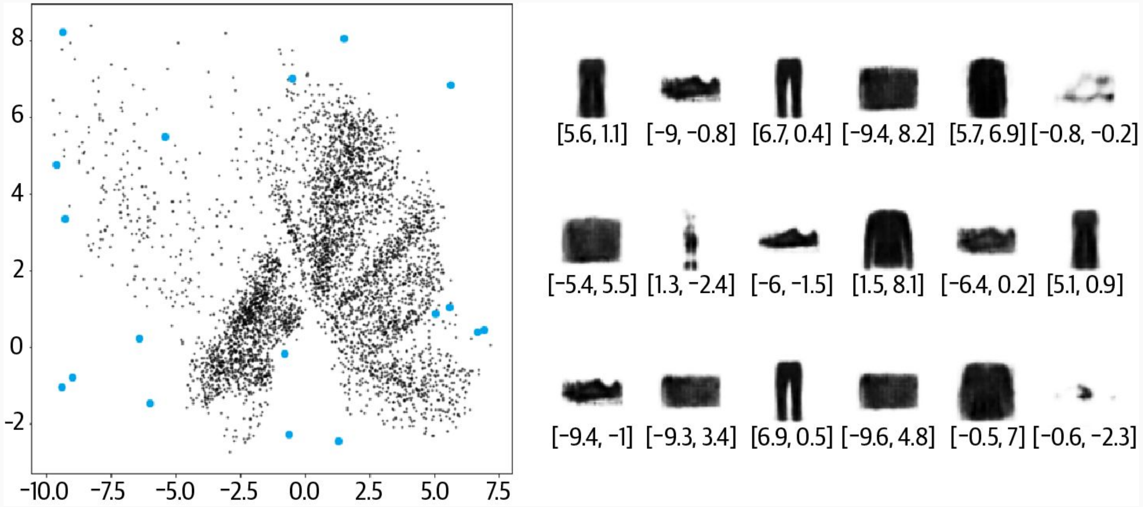
Visualizing the Latent Space

Plot of the latent space, colored by clothing label

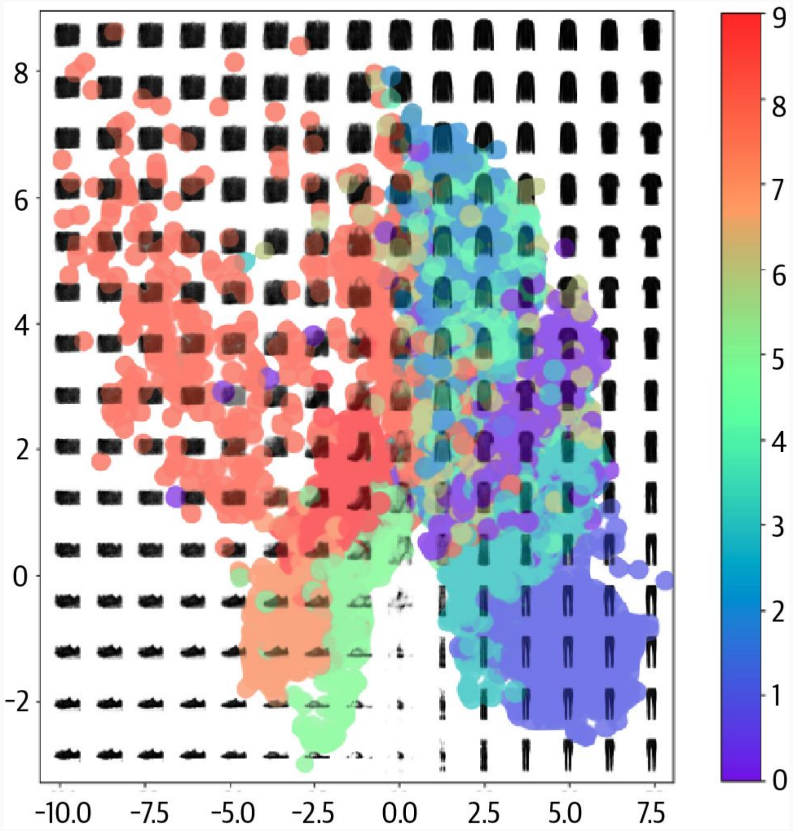


Generating New Images

```
# Generating novel images using the decoder
mins, maxs = np.min(embeddings, axis=0), np.max(embeddings, axis=0)
sample = np.random.uniform(mins, maxs, size=(18, 2))
reconstructions = decoder.predict(sample)
```



Embeddings colored by item type



1. Some clothing items are represented over a very small area and others over a much larger area.
2. The distribution is not symmetrical about the point (0, 0), or bounded. For example, there are far more points with positive y-axis values than negative, and some points even extend to a y-axis value > 8.
3. There are large gaps between colors containing few points.

Variational Autoencoders

Revisiting The Infinite Wardrobe - Improving the Autoencoder

Previously, each item was placed at a single location

A different approach:

- Allocate general areas for each item
  - Allows more diversity
  - Avoids issues with discontinuities
- Add constraints:
  - Areas center near the middle
  - Limit size of areas
  - Penalize being outside the guidelines
- Results:
  - Generates more diverse and realistic items
  - Constraints prevent low quality outputs

TL;DR: Adding some "fuzziness" to embeddings while regularizing helps autoencoders achieve better creative generation.

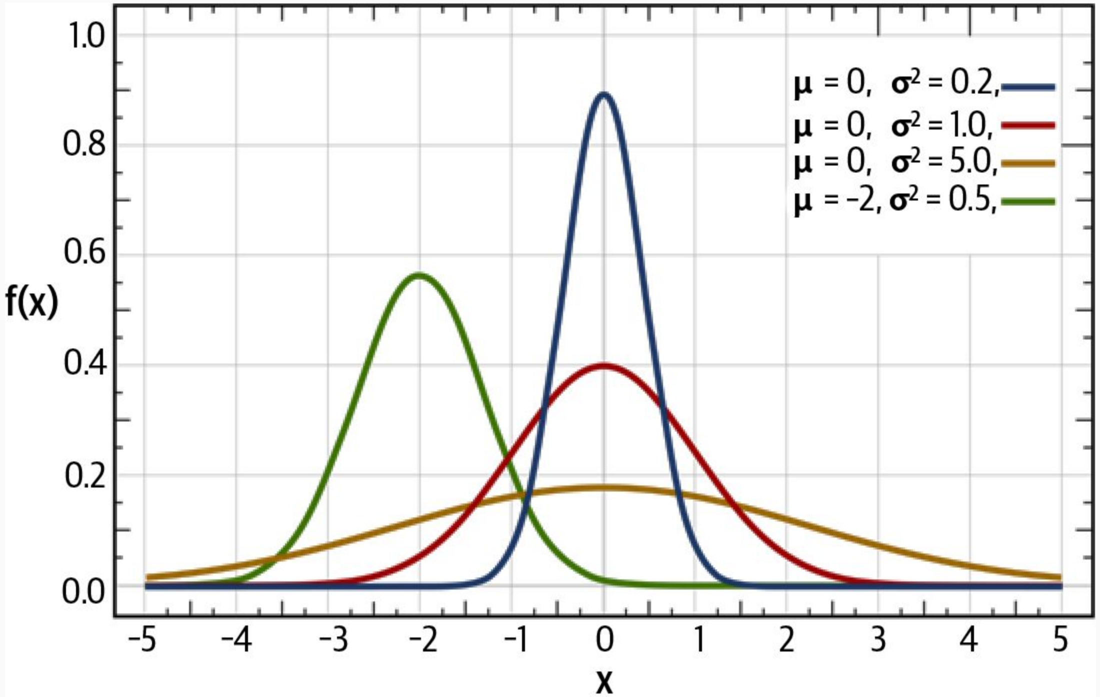
The Multivariate Normal Distribution

A *normal distribution* (or *Gaussian distribution*) $\mathcal{N}(\mu, \sigma)$ is a probability distribution characterized by a distinctive *bell curve* shape, defined by two variables: the *mean* (μ) and the *variance* (σ^2). The *standard deviation* (σ) is the square root of the variance.

The probability density function of the normal distribution in one dimension is:

$$f\left(x \mid \mu, \sigma^2\right) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Normal distributions



- Several normal distributions in one dimension, for different values of the mean and variance.
- The red curve is the standard normal (or unit normal) $\mathcal{N}(0, 1)$ —the normal distribution with mean equal to 0 and variance equal to 1.

Covariance

Covariance in a normal distribution measures the directional relationship between two random variables.

The covariance measures how two variables in a normal distribution change together - whether they tend to move in the same direction or opposite directions.

If covariance is:

1) Positive - the variables tend to increase and decrease together

2) Negative - as one variable increases, the other decreases

3) Zero - the variables are independent.

Symmetric covariance matrix

A covariance matrix captures the covariance between multiple random variables. For example, in a dataset with 3 features X, Y, and Z:

The covariance matrix elements show the covariance between each pair of features. The matrix is square (N x N) if there are N features.It is always symmetric around the main diagonal. That means: ***cov(X, Y) = cov(Y, X)***

In matrix form:

$$\begin{array}{cccc} & X & Y & Z \\ X & \mathrm{cov}(X,X) & \mathrm{cov}(X,Y) & \mathrm{cov}(X,Z) \\ Y & \mathrm{cov}(Y,X) & \mathrm{cov}(Y,Y) & \mathrm{cov}(Y,Z) \\ Z & \mathrm{cov}(Z,X) & \mathrm{cov}(Z,Y) & \mathrm{cov}(Z,Z) \end{array}$$

For the matrix to be symmetric: ***cov(X, Y) = cov(Y, X) & cov(X, Z) = cov(Z, X)***

Multivariate standard normal distribution

A multivariate standard normal distribution $\mathcal{N}(0, \boldsymbol{I})$ is a distribution with a zero-valued mean vector and identity covariance matrix.

For example, a 3x3 identity covariance matrix:

1	0	0
0	1	0
0	0	1

This structures means that:

- The variance of each variable is 1.
- There is no covariance between any pair of variables (all independent).
- It assumes all dimensions have equal univariate variance.

P.D.F - Multivariate normal distribution

The concept of a normal distribution extends to more than one dimension—the probability density function for a multivariate normal distribution (or multivariate Gaussian distribution) $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ in $\boldsymbol{k}$ dimensions with mean vector and symmetric covariance matrix Σ is as follows:

$$f\left(x_1, \ldots, x_k\right) = \frac{\exp\ -\frac{1}{2}(\mathbf{x}-\mu)^{\mathrm{T}} \Sigma^{-1} (\mathbf{x}-\mu)\Big)}{\sqrt{(2\pi)^k |\Sigma|}}$$

Variational inference - Kingma, D. P., & Welling, M. (2013)

- We have a complex model that tries to explain our data using hidden variables ***z***.
- However, getting values for ***z*** that best explain the data is very hard.
- Variational inference makes this easier by using an approximation.
  - It introduces a simple distribution ***q(z)*** that hopes to mimic the true complicated distribution ***p(z)***.
  - We then optimize ***q(z)*** to be as close as possible to ***p(z)***.
  - This gives us approximate values of ***z*** that try to explain the data.
- TL;DR:
  - True posterior ***p(z)*** is complex and intractable
  - Variational inference uses a simpler approximate posterior ***q(z)***
  - Optimizes ***q(z)*** to be as close as possible to ***p(z)***

This makes it much more efficient to infer the hidden variables ***z*** versus directly tackling the true complex distribution ***p(z)***.

Reparameterization trick - Kingma, D. P., & Welling, M. (2013)

Normally, to do variational inference we have to compute a difficult integral over the distribution ***q(z)*** - this is very slow and hard to optimize.

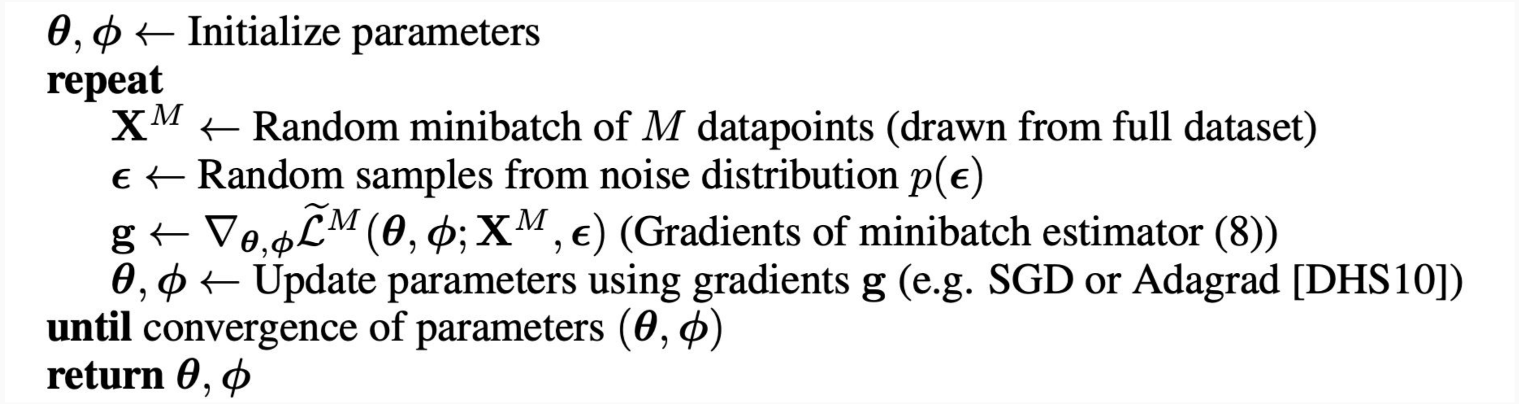
The trick provides an alternative way to sample from ***q(z)*** that avoids this integral. It works like this:

1. Sample epsilon from a simple distribution $p(\epsilon)$
2. Plug epsilon into a function ***g()*** to get your sample z = $g(\epsilon)$
3. Now z is a sample from ***q(z)*** and we avoided the painful integral!

This turns the problem of sampling from ***q(z)*** into sampling from $p(\epsilon)$ which is much much easier.

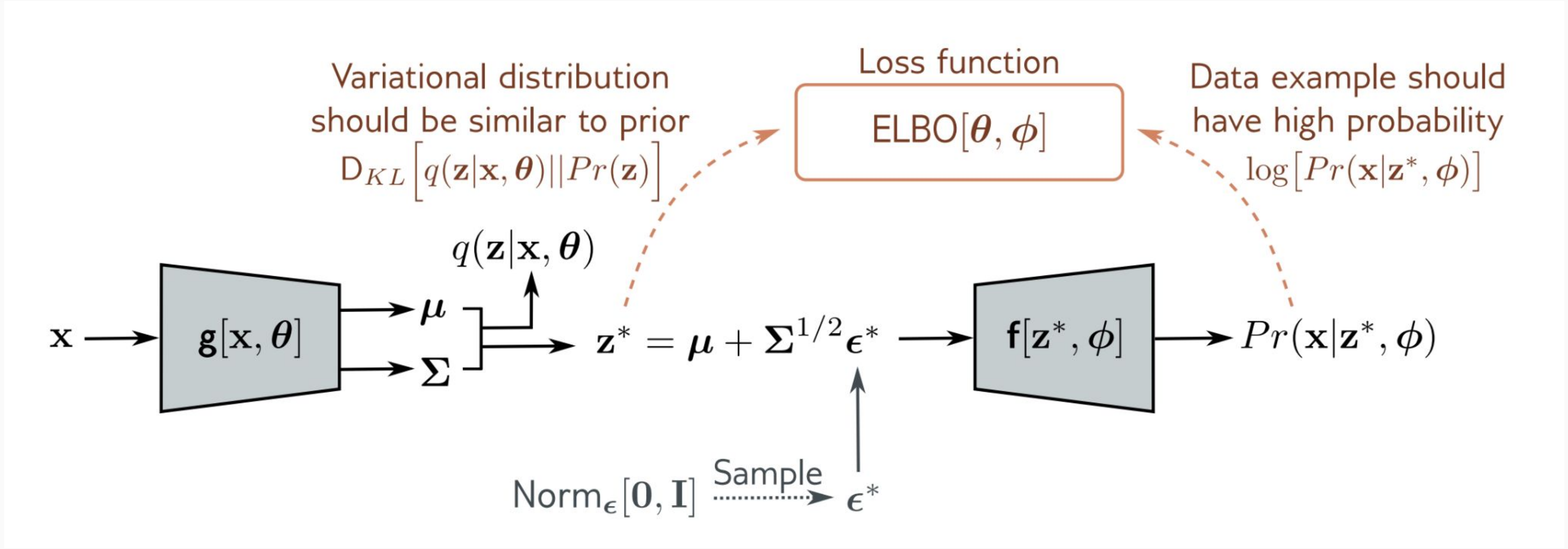
We can repeat steps 1-2 many times to get many samples of ***z***.

Algorithm - Kingma, D. P., & Welling, M. (2013)



```
$\boldsymbol{\theta}, \boldsymbol{\phi} \leftarrow$ Initialize parameters
**repeat**
    $\mathbf{X}^M \leftarrow$ Random minibatch of $M$ datapoints (drawn from full dataset)
    $\boldsymbol{\epsilon} \leftarrow$ Random samples from noise distribution $p(\boldsymbol{\epsilon})$
    $\mathbf{g} \leftarrow \nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} \widetilde{\mathcal{L}}^M(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{X}^M, \boldsymbol{\epsilon})$ (Gradients of minibatch estimator (8))
    $\boldsymbol{\theta}, \boldsymbol{\phi} \leftarrow$ Update parameters using gradients $\mathbf{g}$ (e.g. SGD or Adagrad [DHS10])
**until** convergence of parameters $(\boldsymbol{\theta}, \boldsymbol{\phi})$
**return** $\boldsymbol{\theta}, \boldsymbol{\phi}$
```

Reparameterization trick


Variational distribution
should be similar to prior
$D_{KL}\left[q(\mathbf{z}|\mathbf{x},\boldsymbol{\theta})||Pr(\mathbf{z})\right]$
Loss function
$\text{ELBO}[\boldsymbol{\theta},\boldsymbol{\phi}]$
Data example should
have high probability
$\log\left[Pr(\mathbf{x}|\mathbf{z}^*,\boldsymbol{\phi})\right]$
$q(\mathbf{z}|\mathbf{x},\boldsymbol{\theta})$
$\mathbf{x}$
$\mathbf{g}[\mathbf{x},\boldsymbol{\theta}]$
$\boldsymbol{\mu}$
$\boldsymbol{\Sigma}$
$\mathbf{z}^* = \boldsymbol{\mu} + \boldsymbol{\Sigma}^{1/2}\boldsymbol{\epsilon}^*$
$\mathbf{f}[\mathbf{z}^*,\boldsymbol{\phi}]$
$Pr(\mathbf{x}|\mathbf{z}^*,\boldsymbol{\phi})$
$\text{Norm}_{\boldsymbol{\epsilon}}[\mathbf{0},\mathbf{I}]$
Sample
$\boldsymbol{\epsilon}^*$

Encoder and Decoder

Encoder:

- The encoder only needs to map each input to a mean vector and a variance vector and does not need to worry about covariance between dimensions.
- Variational autoencoders assume that there is no correlation between dimensions in the latent space.
- Variance values are always positive, so we actually choose to map to the logarithm of the variance, as this can take any real number in the range (-∞, ∞).
- This way we can use a neural network as the encoder to perform the mapping from the input image to the mean and log variance vectors.
- The encoder will take each input image and encode it to two vectors that together define a multivariate normal distribution in the latent space

Decoder: The decoder of a variational autoencoder is identical to the decoder of a plain autoencoder

Mechanics

The mean point of the distribution: *z_mean*

The logarithm of the variance of each dimension: *z_log_var*

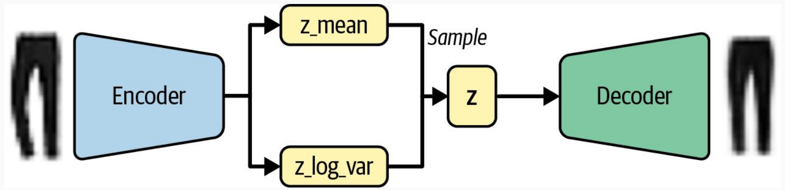
We can sample a point *z* from the distribution defined by these values by:

*z = z_mean + z_sigma * epsilon*

Where: *z_sigma = exp(z_log_var * 0.5), epsilon ~ N(0,I)*

The derivation of the relationship between z_sigma and *z_log_var is:*

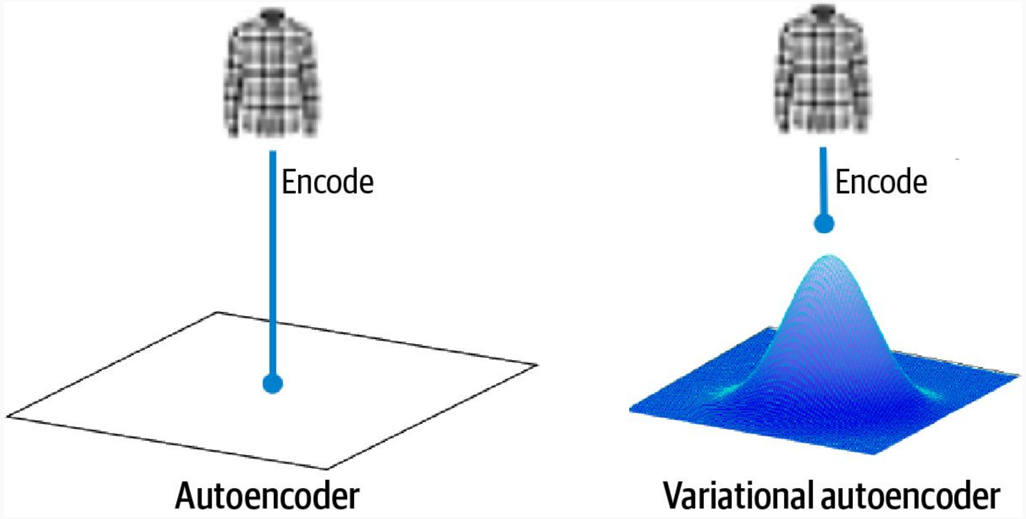
$$\sigma = \exp\left(\log\left(\sigma\right)\right) = \exp\left(2\log\left(\sigma\right)/2\right) = \exp\left(\log\left(\sigma^2\right)/2\right)$$



Why does this change to the encoder help?

- Previously, we saw that there was no requirement for the latent space to be continuous—even if the point (−2, 2) decodes to a well-formed image of a sandal, there's no requirement for (−2.1, 2.1) to look similar.
- Now, since we are sampling a random point from an area around z_mean, the decoder must ensure that all points in the same neighborhood produce very similar images when decoded, so that the reconstruction loss remains small.
- This is a very nice property that ensures that even when we choose a point in the latent space that has never been seen by the decoder, it is likely to decode to an image that is well formed.

The Encoder



- In an autoencoder, each image is mapped directly to one point in the latent space.
- In a variational autoencoder, each image is instead mapped to a multivariate normal distribution around a point in the latent space.

KL divergence

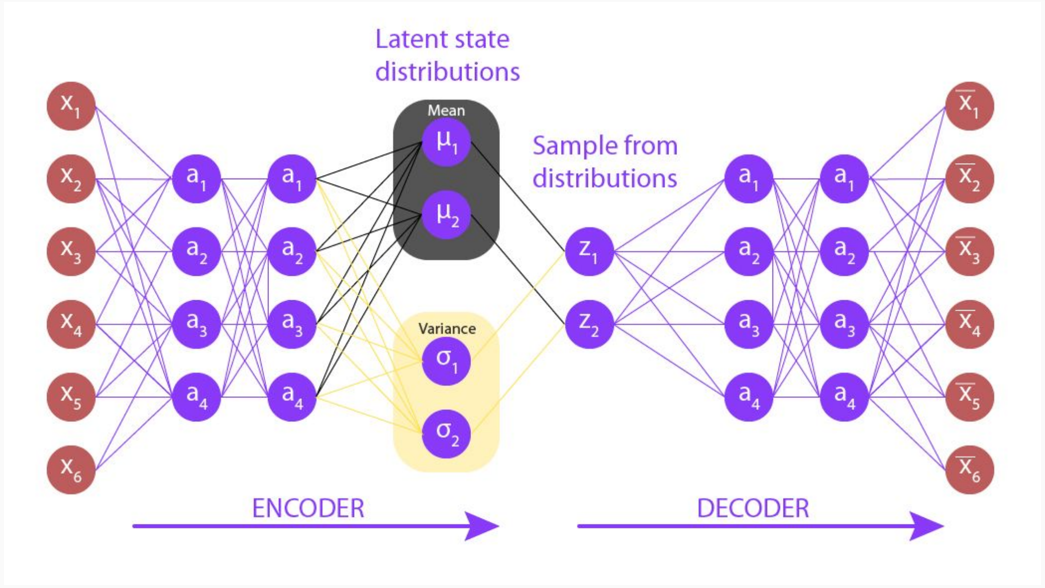
- Measures difference between two distributions
- Penalizes large differences from normal distribution
- The mathematical equation is:

$$D_{KL}[N(\mu, \sigma \parallel N(0,1)] = -\frac{1}{2}\sum\left(1 + log\left(\sigma^2\right) - \mu^2 - \sigma^2\right)$$

- In closed form:

  *Kl_loss = -0.5 * sum(1 + z_log_var - z_mean ^ 2 - exp(z_log_var))*

Workflow


Latent state distributions
Mean
μ_1
μ_2
Variance
σ_1
σ_2
Sample from distributions
x_1 x_2 x_3 x_4 x_5 x_6
a_1 a_2 a_3 a_4
z_1 z_2
$\bar{x}_1$ $\bar{x}_2$ $\bar{x}_3$ $\bar{x}_4$ $\bar{x}_5$ $\bar{x}_6$
ENCODER
DECODER

http://bit.ly/47NVdI2

Building a VAE

Sampling Layer and Reparameterization trick

- Sampling layer: to sample from the distribution defined by z_mean and z_log_var
- Reparameterization trick: Rather than sample directly from a normal distribution with parameters z_mean and z_log_var, we can sample epsilon from a standard normal and then manually adjust the sample to have the correct mean and variance.

Loss Function

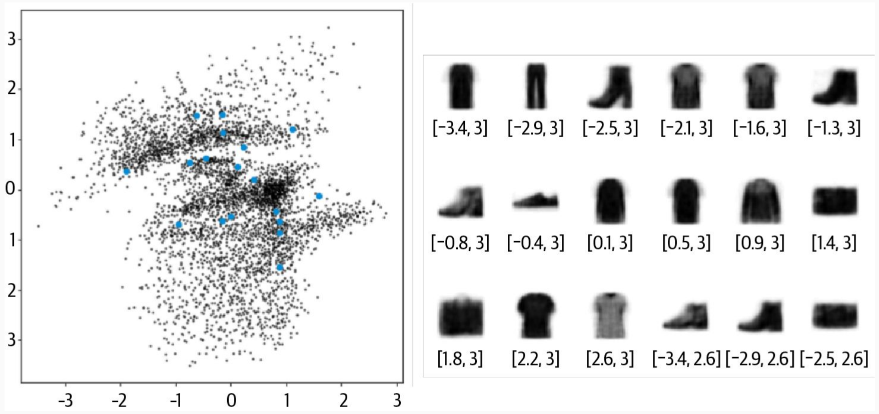
1. Reconstruction loss - between images and their attempted copies after being passed through the encoder and decoder
2. KL loss - KL divergence is a way of measuring how much one probability distribution differs from another. In a VAE, we want to measure how much our normal distribution with parameters z_mean and z_log_var differs from a standard normal distribution.

How does KL loss help?

- We now have a well-defined distribution that we can use for choosing points in the latent space—the standard normal distribution.
- Since this term tries to force all encoded distributions toward the standard normal distribution, there is less chance that large gaps will form between point clusters.
- Instead, the encoder will try to use the space around the origin symmetrically and efficiently.

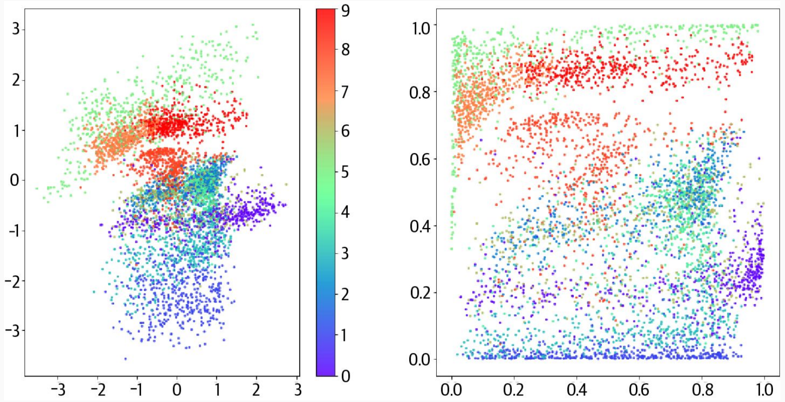
Analysis of the Variational Autoencoder

The New latent space



- The black dots show the z_mean value of each encoded image.
- The blue dots show some sampled points in the latent space
- KL divergence loss term ensures that the z_mean and z_log_var values of the encoded images never stray too far from a standard normal distribution.
- There are not so many poorly formed images as the latent space is now much more continuous.

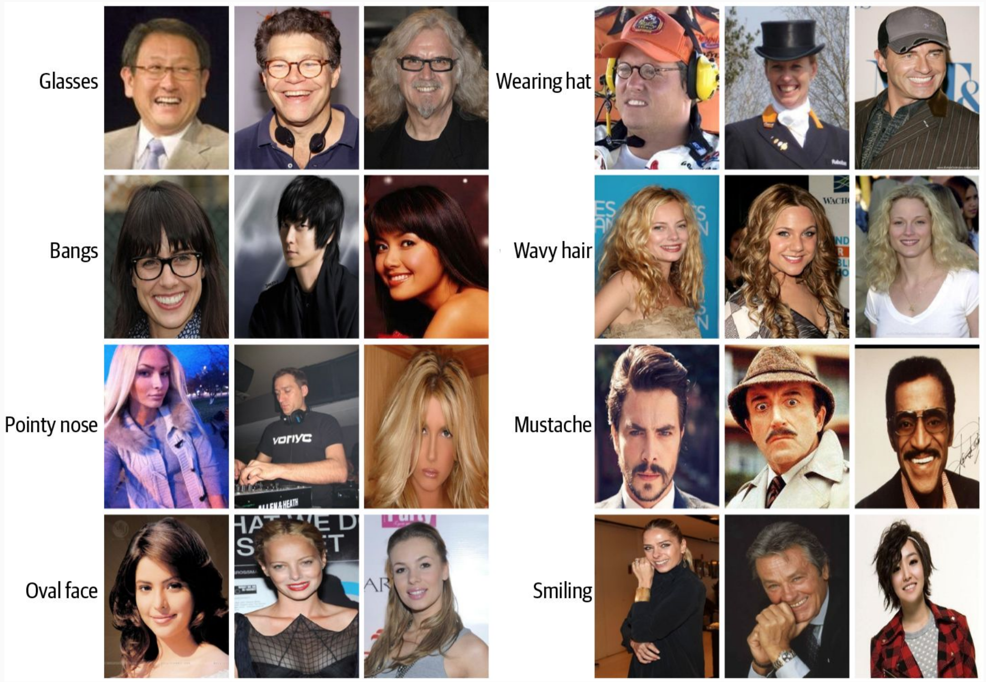
The latent space of the VAE colored by clothing type



- There is no preferential treatment of any one type.
- The right hand plot shows the space transformed into p-values—we can see that each color is approximately equally represented.
- Labels were not used at training
- VAE has learned the various forms of clothing by itself in order to help minimize reconstruction loss.

Exploring the Latent Space

The CelebA Dataset

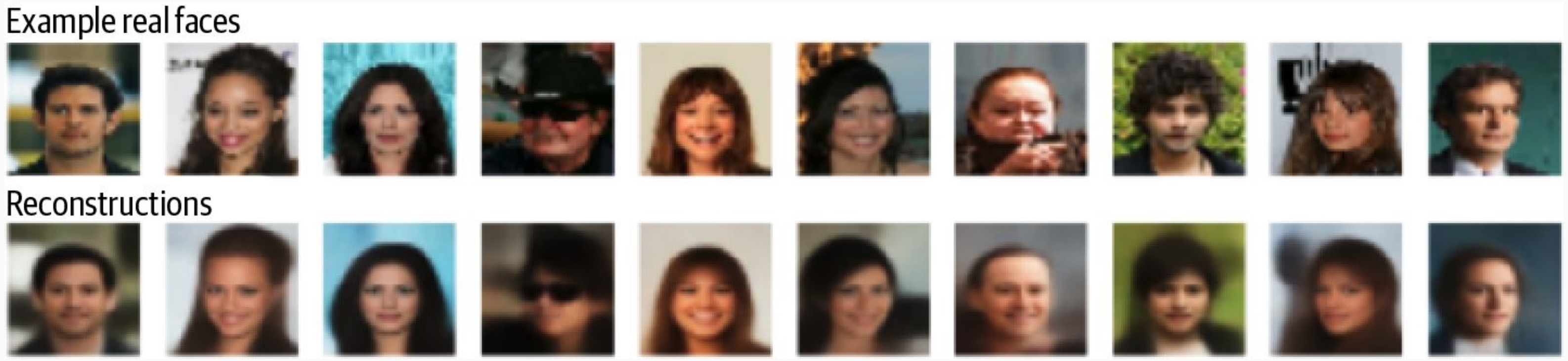


Some examples from the CelebA dataset (source: Liu et al., 2015)

Training the Variational Autoencoder

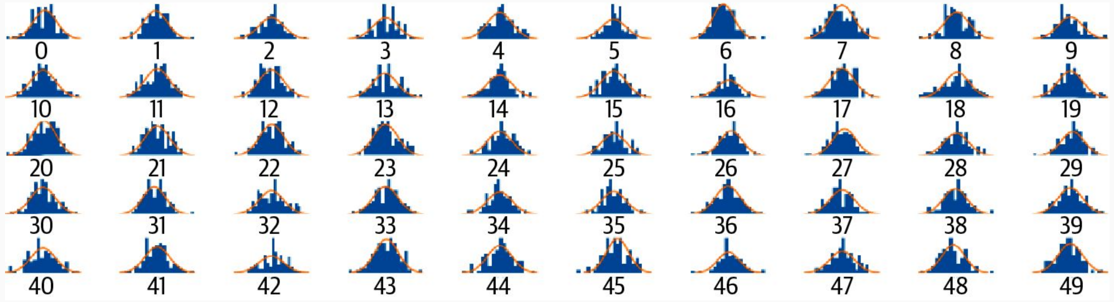
- The data now has three input channels (RGB) instead of one (grayscale).
- This means we need to change the number of channels in the final convolutional transpose layer of the decoder to 3.
- We will using a latent space with 200 dimensions instead of 2. Since faces are much more complex than the Fashion-MNIST images, we increase the dimensionality of the latent space so that the network can encode a satisfactory amount of detail from the images.
- There are batch normalization layers after each convolutional layer to stabilize training. Even though each batch takes a longer time to run, the number of batches required to reach the same loss is greatly reduced.
- We increase the β factor for the KL divergence to 2,000. This is a parameter that requires tuning; for this dataset and architecture this value was found to generate good results.

Analysis of the Variational Autoencoder



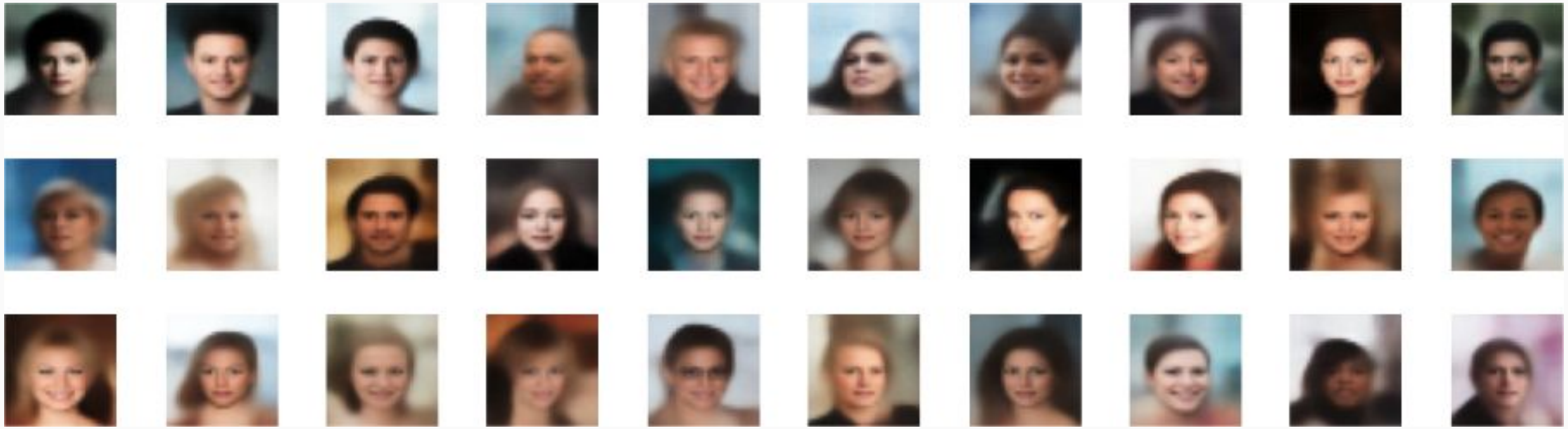
Reconstructed faces, after passing through the encoder and decoder

Distributions of points



Distributions of points for the first 50 dimensions in the latent space

New generated faces

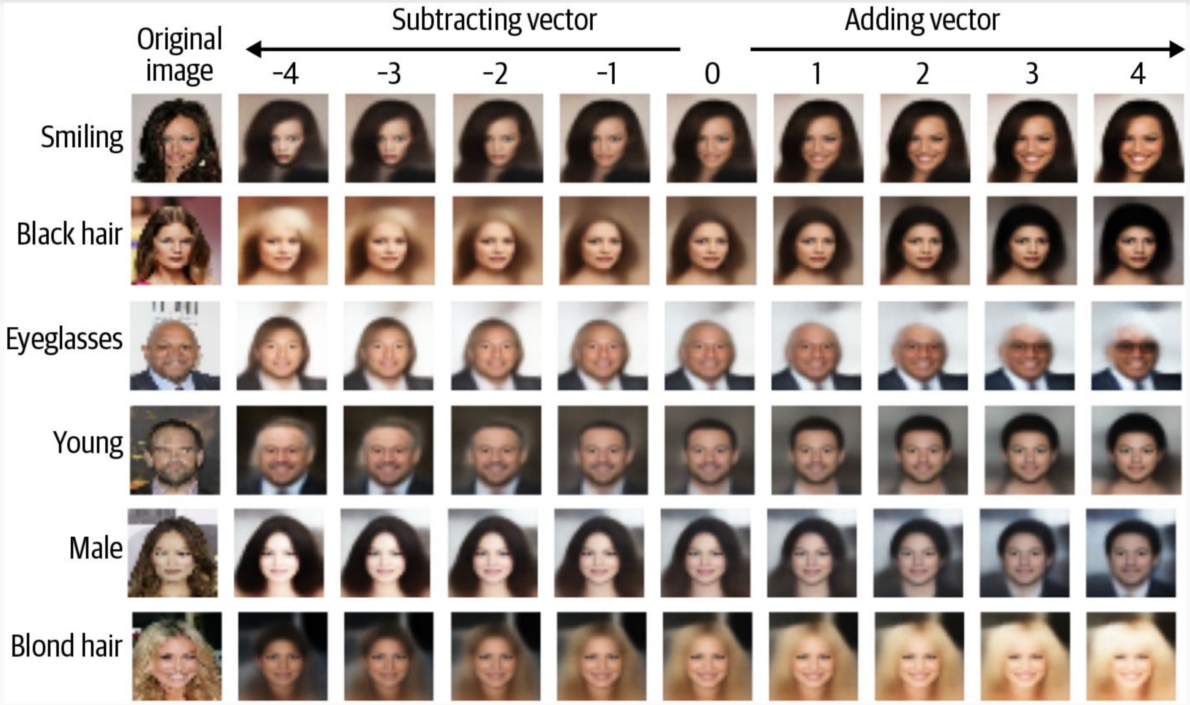


Latent Space Arithmetic

- Latent space allows arithmetic operations between embeddings
- Can find conceptual vectors:
  - Take average smiling embedding
  - Subtract average non-smiling embedding
- Gives "smile vector" pointing in that concept direction
- Adding smile vector to any image makes it more smiley!
- Conceptually, we are performing the following vector arithmetic in the latent space, where alpha is a factor that determines how much of the feature vector is added or subtracted:

$$z\_new = z + alpha * feature\_vector$$

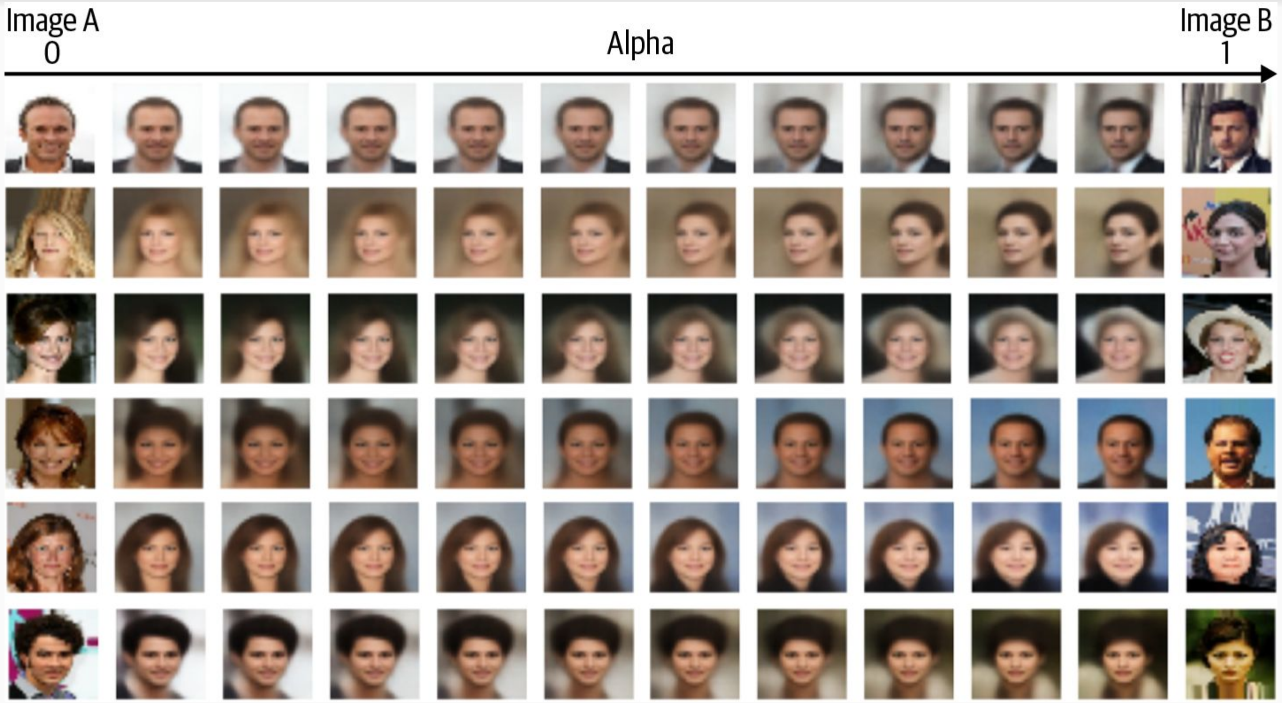
Adding and subtracting features to and from faces



Morphing

- Imagine two points in the latent space, A and B, that represent two images.
- If you started at point A and walked toward point B in a straight line, decoding each point on the line as you went, you would see a gradual transition from the starting face to the end face.
- Mathematically, we are traversing a straight line, which can be described by the following equation: *z_new = z_A * (1- alpha) + z_B * alpha*
- Alpha is a number between 0 and 1 that determines how far along the line we are, away from point A.

Morphing Between Faces



This Week - Homework

1. Read Multimodal Variational Auto-encoder based Audio-Visual Segmentation paper and identify an approach from the paper that you thought was novel.
2. Run Chap_3_VAE_MNIST.ipynb with different batch sizes and see if there is any difference in the outcome.

References

- Diederik P. Kingma and Max Welling, “Auto-Encoding Variational Bayes,” December 20, 2013, *https://arxiv.org/abs/1312.6114*.
- Vincent Dumoulin and Francesco Visin, “A Guide to Convolution Arithmetic for Deep Learning,” January 12, 2018, *https://arxiv.org/abs/1603.07285*.
- Ziwei Liu et al., “Large-Scale CelebFaces Attributes (CelebA) Dataset,” 2015, *http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html*.
- Mao, Y., Zhang, J., Xiang, M., Zhong, Y., & Dai, Y. (2023). Multimodal variational auto-encoder based audio-visual segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 954-965).

Multimodal variational auto-encoder

Mao, Y., Zhang, J., Xiang, M., Zhong, Y., & Dai, Y. (2023). Multimodal variational auto-encoder based audio-visual segmentation. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 954-965).

Introduction

- Audio-Visual Segmentation (AVS) aims to segment sound sources in a video sequence
- Existing AVS methods focus on implicit feature fusion strategies
- Models are trained to fit discrete samples in the dataset
  - Limited and less diverse datasets lead to unsatisfactory performance
  - ECMVAE (Explicit Conditional Multimodal Variational Auto-Encoder) approaches AVS from an effective representation learning perspective
- Explicitly models the contribution of each modality
- Addresses the limitations of existing methods

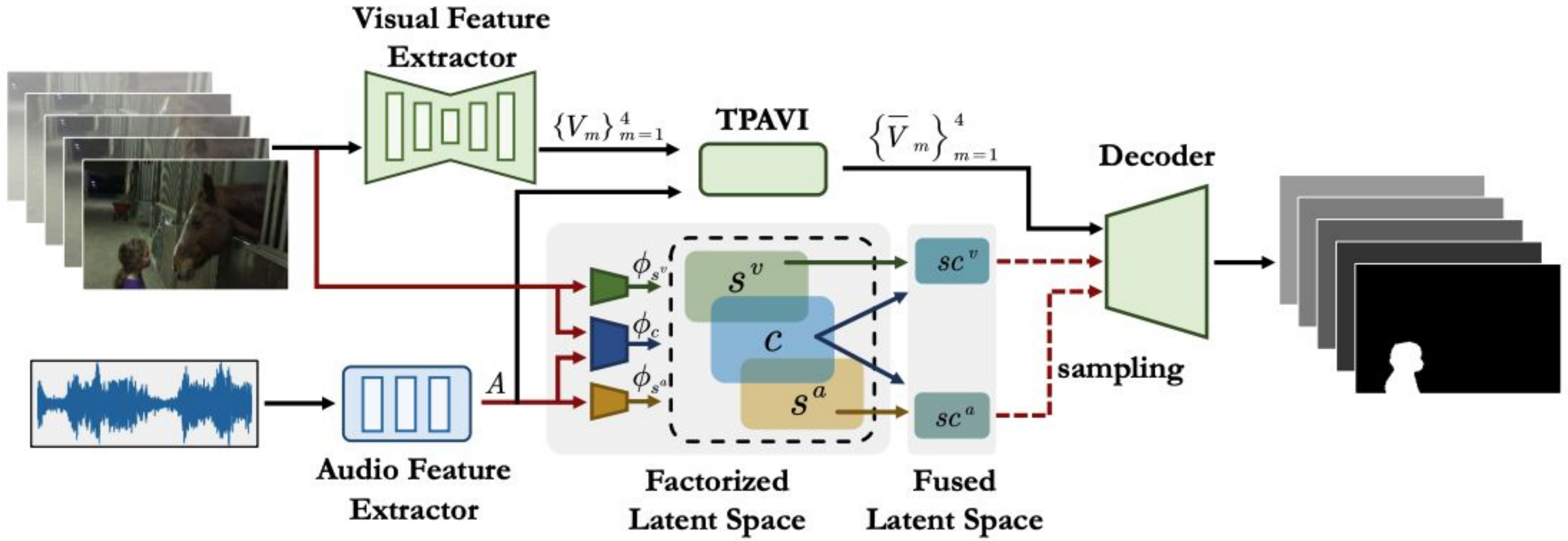
Key Insights

- Audio contains critical category information of sound producers
  - Provides information about the type of sound being produced
- Visual data provides candidate sound producer(s)
  - Shows the appearance of the entire scene, with sound producers taking a small portion
- Shared information between modalities corresponds to target sound producer(s)
  - Represents the common information related to the sound producers
- Cross-modal shared representation learning is crucial for AVS
  - Enables the model to effectively leverage information from both modalities

ECMVAE Overview

- Factorizes representations of each modality into shared and specific representations
  - Shared representation (c) captures common information between modalities
  - Modality-specific representations (s_a, s_v) capture unique information for each modality
- Applies orthogonality constraint between shared and specific representations
  - Maintains the exclusive attribute of the factorized latent code
  - Ensures that shared and specific representations capture different aspects of the input
- Introduces mutual information maximization regularizer for extensive exploration of modalities
  - Encourages the model to extract maximally informative representations from each modality
- Built upon a multimodal variational auto-encoder (MVAE) framework
  - Leverages the advantages of variational auto-encoders for representation learning
- Uses Jensen-Shannon divergence for trade-off between sampling efficiency and quality
  - Achieves a balance between computational efficiency and the quality of generated samples

ECMVAE for audio-visual segmentation



- The feature extractors are used to extract backbone features for the two modalities.
- Three latent encoders ϕsv , ϕsa , ϕc to achieve latent space factorization and obtain both task-driven shared representation (c) and modality-related specific representation (s a , sv), achieving explicit multimodal representation learning.
- The decoder is introduced to obtain the final segmentation maps, indicating the sound producers of the audio-visual data.

Model Components

- Modal Encoding:
    - Visual branch: ImageNet pre-trained backbone + convolutional neck
    - Extracts multi-scale visual features from the input video frames
- Audio branch: Frozen VGGish model pre-trained on AudioSet
    - Processes the spectrogram of the input audio to extract audio features
- Latent Space Encoding:
    - Learns task-driven shared (c) and modality-specific (s_a, s_v) representations
    - Shared representation captures common information relevant to the AVS task
    - Modality-specific representations capture unique information from each modality
- Decoder:
    - Takes deterministic features from TPAVI (Temporal Pixel-wise Audio-Visual Interaction) module and expanded latent codes
    - TPAVI module performs cross-modal attention-based feature fusion
    - Generates final segmentation maps
    - Predicts the pixel-wise segmentation masks for the sound producers
- Objective Function:
    - Optimizes ELBO (Evidence Lower Bound) with practical constraints for latent space representation
    - Includes reconstruction term and KL divergence term for variational inference
    - Incorporates orthogonality constraint and mutual information maximization regularizer

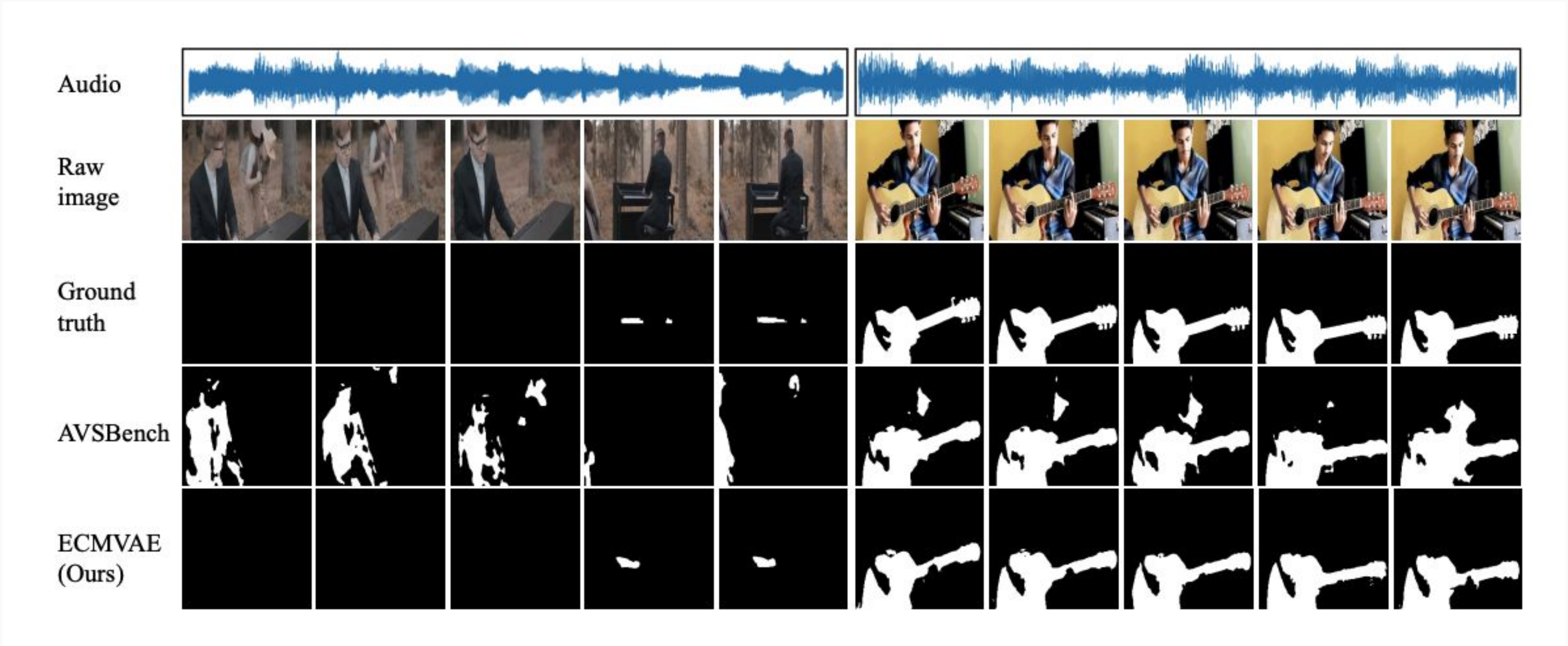
Latent Space Factorization

- Factorizes latent space into modality-shared (c) and modality-specific (s_a, s_v) representations
  - Enables the model to capture both common and unique information from each modality
- Orthogonality constraint maintains exclusivity of factorized latent code
  - Ensures that shared and specific representations capture different aspects of the input
- Implemented using a difference loss between the representations
  - Mutual information maximization regularizer explores contribution of each modality
- Encourages the model to extract maximally informative representations from each modality
  - Achieved by maximizing the mutual information between the fused representations of each modality
  - Latent space factorization allows for effective modeling of the contribution of each modality

Experimental Setup - AVSBench dataset

- Contains 5,356 video sequences with corresponding audio data and binary per-pixel annotations
- Includes two settings: S4 (Single Sound Source Segmentation) and MS3 (Multiple Sound Source Segmentation)
- Compares ECMVAE with state-of-the-art methods
- Baseline AVS models and methods from related tasks (e.g., video object segmentation, salient object detection)
- Evaluates segmentation performance using mIoU (Mean Intersection over Union) and F-score metrics
- mIoU measures the overlap between predicted and ground-truth segmentation masks
- F-score combines precision and recall to assess the quality of the segmentation
- Qualitative comparisons for audio temporal and spatial localization quality
- Visual assessment of the segmentation results and the ability to localize sound producers

Qualitative comparison between ECMVAE and AVSBench



The method competently achieves high segmentation performance with better audio temporal and spatial localization quality and detail handling

Results

- ECMVAE consistently achieves superior segmentation performance
- Outperforms state-of-the-art methods on the AVSBench dataset
- Significant improvements in mIoU and F-score compared to state-of-the-art methods
- Achieves 3.00 and 3.84 higher mIoU than the previous AVS method on S4 and MS3 settings, respectively (with PVTv2 backbone)
- Also shows consistent performance improvement with ResNet backbone
- Better audio temporal and spatial localization quality
- Provides more accurate segmentation of sound producers
- Handles background noise and captures richer foreground details
- Qualitative comparisons demonstrate the effectiveness of ECMVAE in AVS

Summary: The Power of Variational Autoencoders

- Plain autoencoders map images to a latent feature space
- But sampling the learned space struggles to generate realistic novel images
- Variational autoencoders solve this by:
  - Introducing randomness
  - Constraining the latent distribution
- Transforms the autoencoder into a powerful generative model
- Decoding random points synthesizes new examples
- The structured latent space also enables:
  - Intuitive feature vector arithmetic
  - Face morphing and editing applications