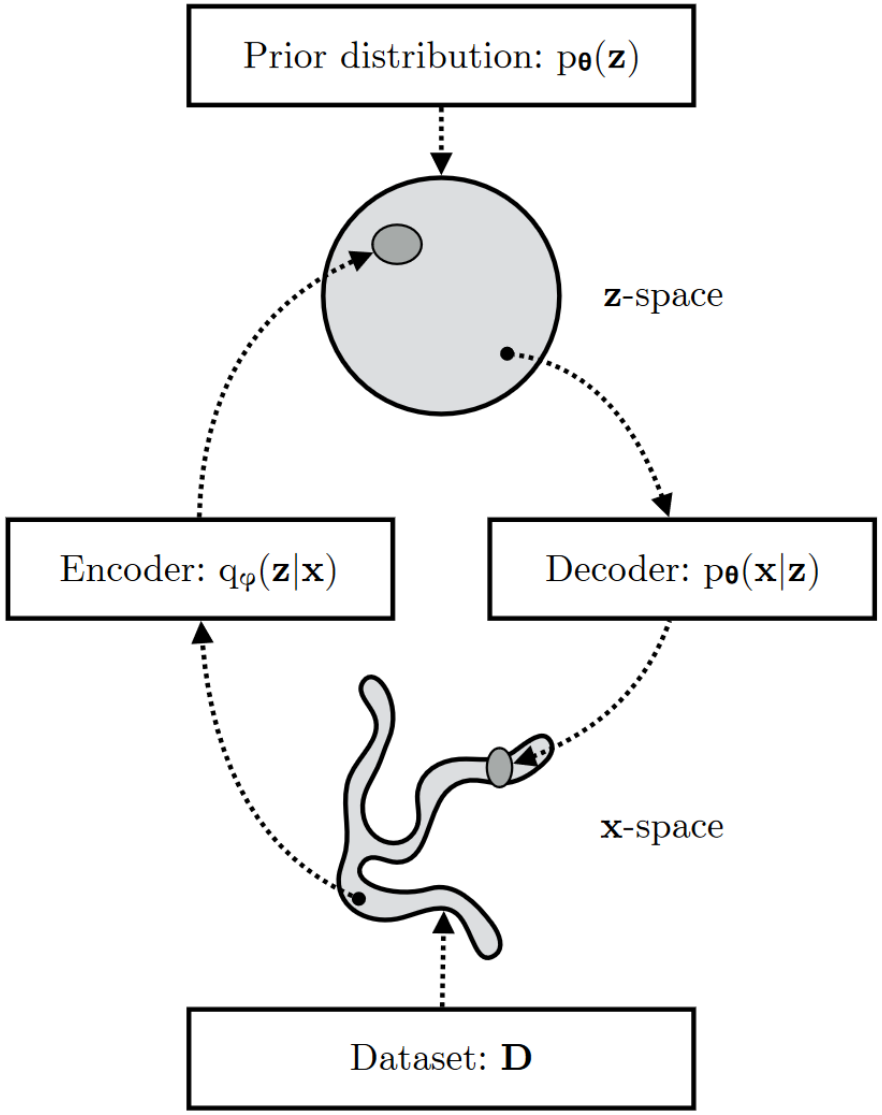


Kingma, D.P. and Welling, M., 2019. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, *12*(4), pp.307-392.

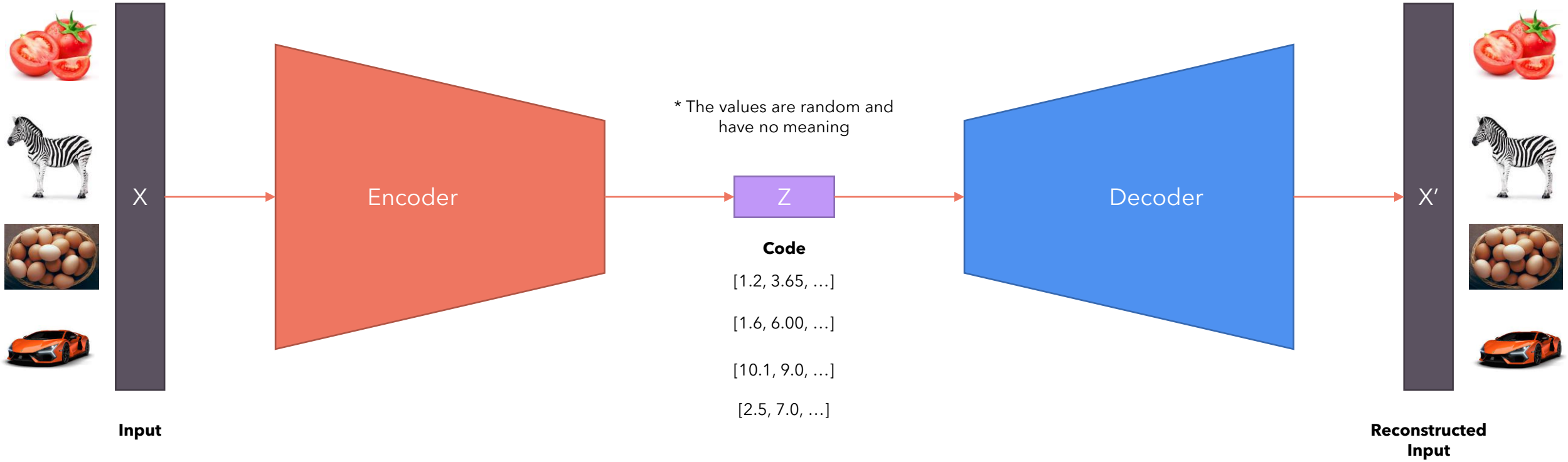
Variational Autoencoder from scratch

Umar Jamil

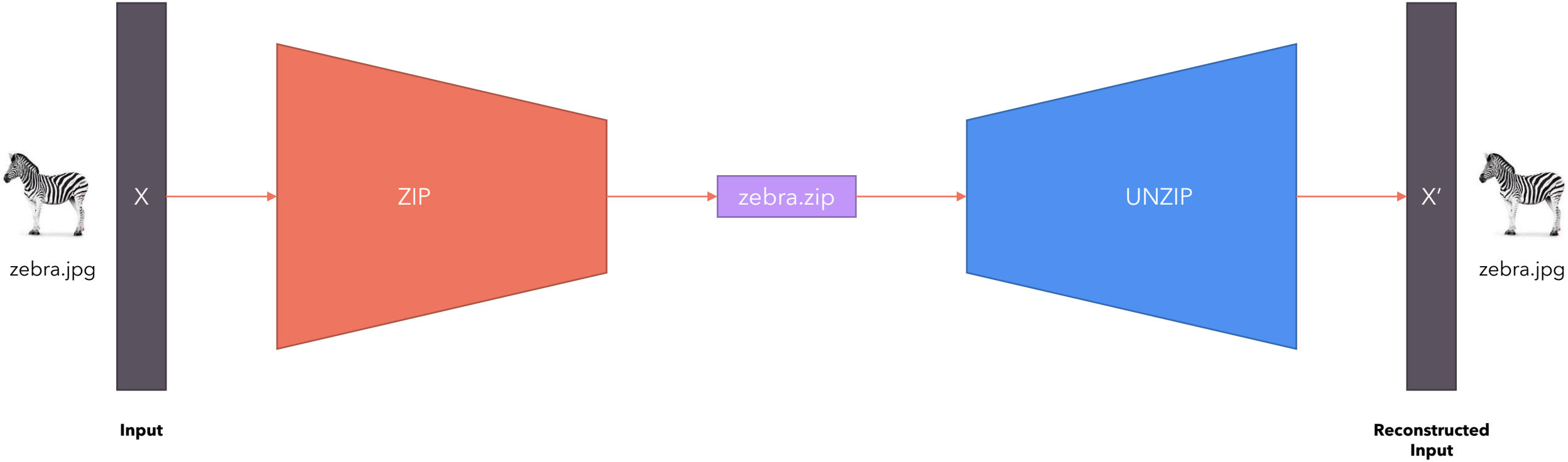
Video: https://youtu.be/iwEzwTTalbg

Not for commercial use

What is an Autoencoder?



Analogy with file compression



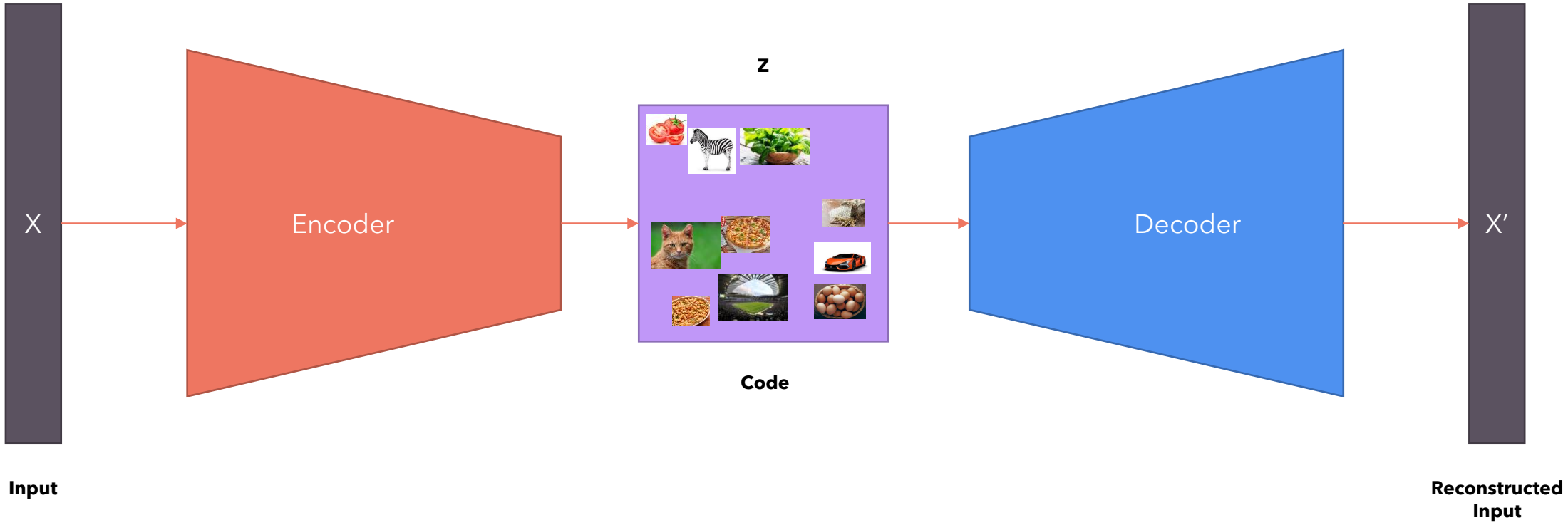
What makes a good Autoencoder?

- The code should be as small as possible, that is, the dimension of the Z vector should be as small as possible.
- The reconstructed input should as close as possible to the original input.

Umar Jamil - https://github.com/hkproj/vae-from-scratch-notes

What's the problem with Autoencoders?

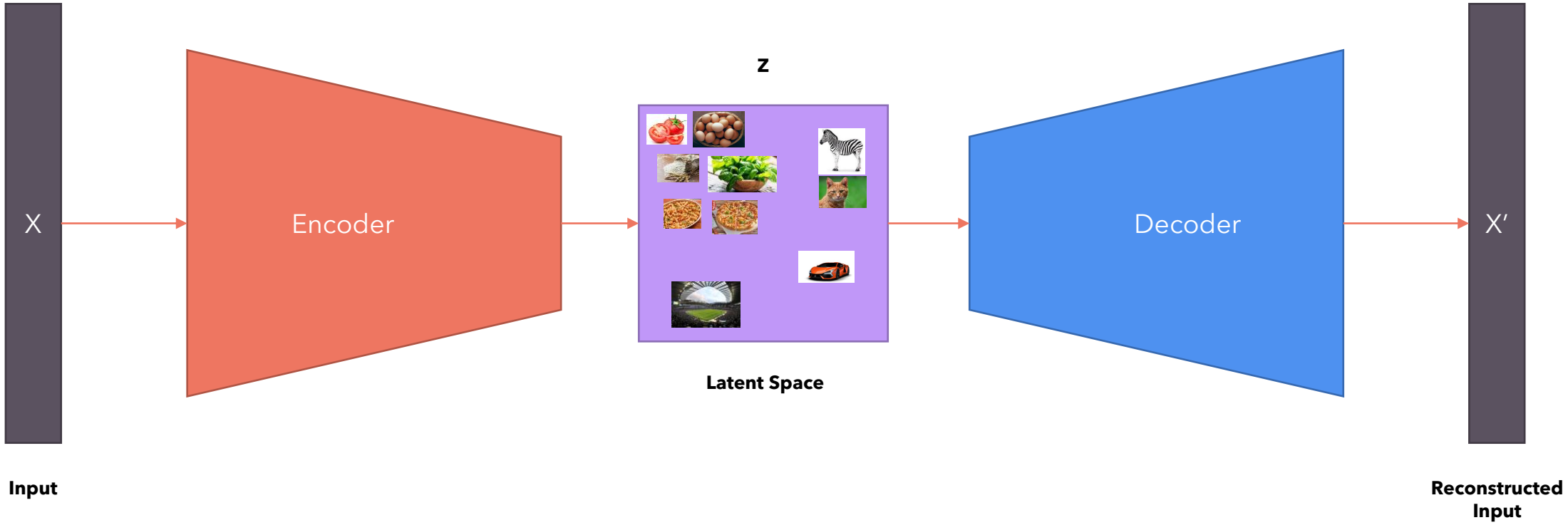
The code learned by the model **makes no sense**. That is, the model can just assign any vector to the inputs without the numbers in the vector representing any pattern. The model doesn't capture any **semantic relationship** between the data.



Umar Jamil - https://github.com/hkproj/vae-from-scratch-notes

Introducing the Variational Autoencoder

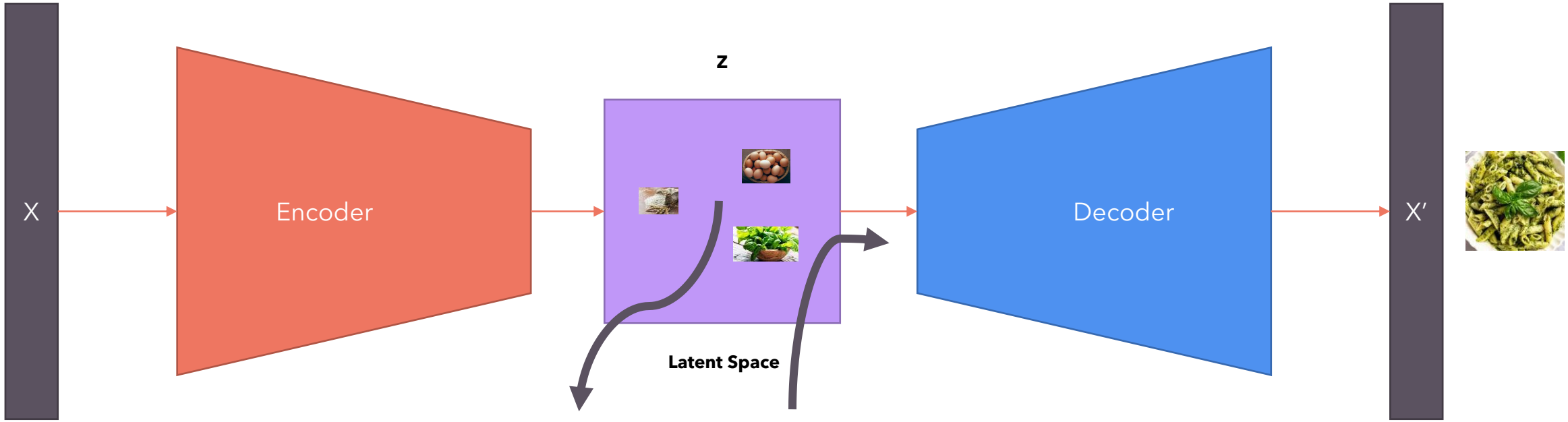
The variational autoencoder, instead of learning a code, learns a "**latent space**". The latent space represents the parameters of a (multivariate) distribution.



Umar Jamil - https://github.com/hkproj/vae-from-scratch-notes

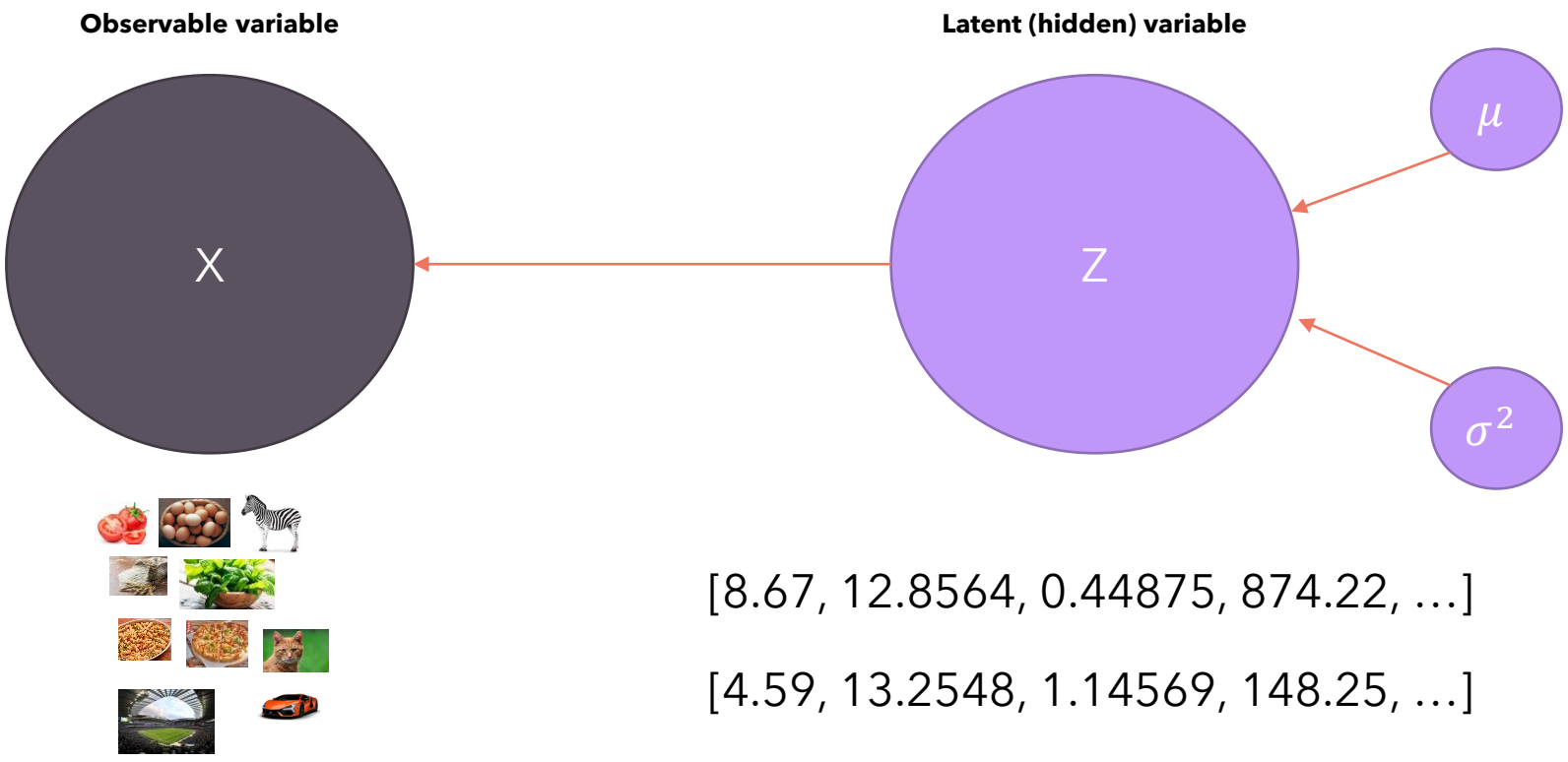
Sampling the latent space

Just like when you use Python to generate a random number between 1 and 100, you're sampling from a uniform (pseudo)random distribution between 1 and 100. In the same way, we can sample from the latent space in order to generate a random vector, give it to the decoder and generate new data.

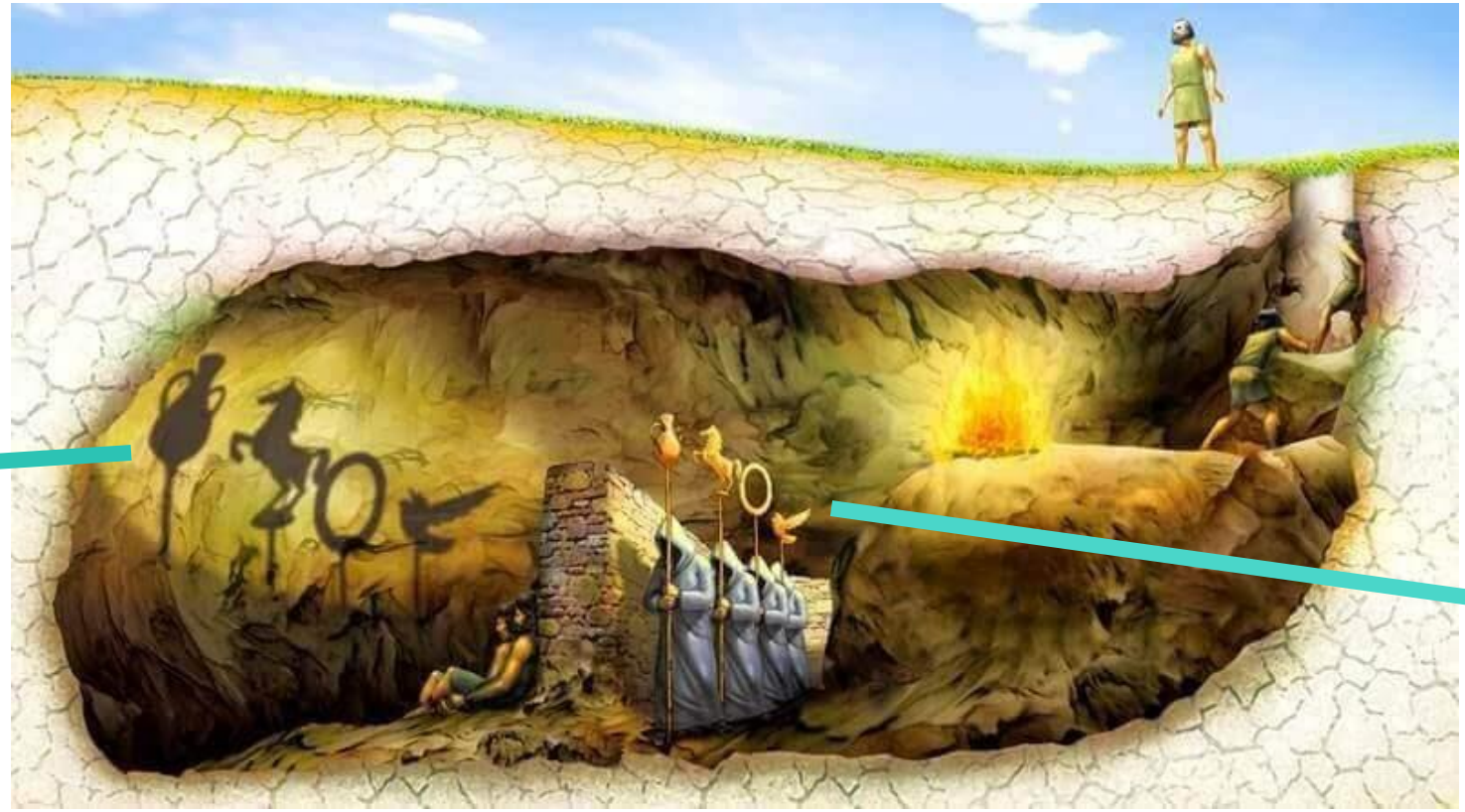


Umar Jamil - https://github.com/hkproj/vae-from-scratch-notes

Why is it called latent space?



Plato's allegory of the cave



Observable variable

Latent (hidden) variable

[8.67, 12.8564, 0.44875, 874.22, …]

[4.59, 13.2548, 1.14569, 148.25, …]

[1.74, 32.3476, 5.18469, 358.14, …]

Umar Jamil - https://github.com/hkproj/vae-from-scratch-notes

Pep talk

1. VAE is the most important component of Stable Diffusion models. Concepts like ELBO also come in Stable Diffusion.
2. In 2023 you shouldn't be memorizing things without understanding, ChatGPT can do that faster and better than any human being. You need to be human to compete with a machine, you can't compete with a machine by acting like one.
3. You should try to learn how things work not only for curiosity, but because that's the true engine of innovation and creativity.
4. Math is fun.

Math Concepts

Expectation of a random variable

$$E_x[f(x)] = \int xf(x)dx$$

Chain rule of probability

$$P(x, y) = P(x|y)P(y)$$

Bayes' Theorem

$$P(x \mid y) = \frac{P(y|x)P(x)}{P(y)}$$

Kullback-Leibler Divergence

$$D_{KL}(P\|Q) = \int p(x) \log\left(\frac{p(x)}{q(x)}\right) dx$$

Properties:

- Not symmetric.
- Always ≥ 0
- It is equal to 0 if and only if $P = Q$

Let's define our model

We can define the likelihood of our data as the marginalization over the joint probability with respect to the latent variable

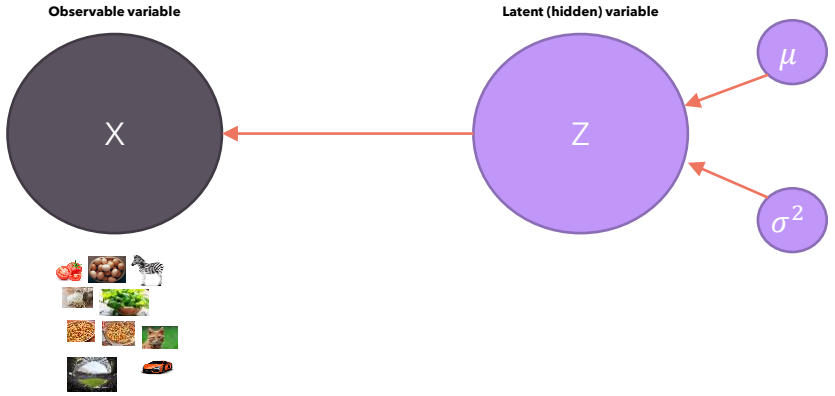
$$p(\boldsymbol{x}) = \int p(\boldsymbol{x}, \boldsymbol{z}) d\boldsymbol{z}$$

Is **intractable** because we would need to evaluate this integral over all latent variables Z.

… or we can use the Chain rule of probability

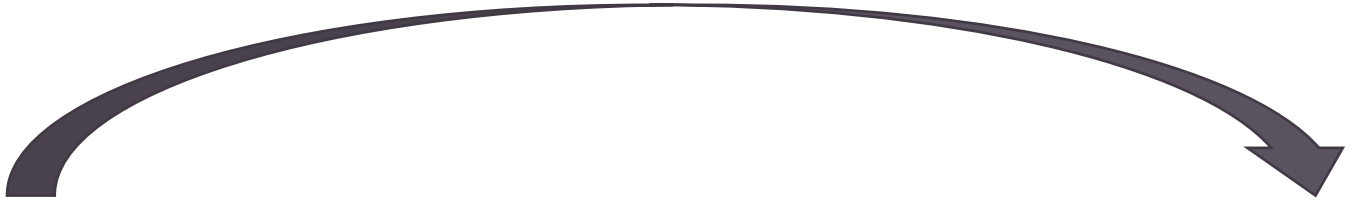
$$p(\boldsymbol{x}) = \frac{p(\boldsymbol{x}, \boldsymbol{z})}{p(\boldsymbol{z}|\boldsymbol{x})}$$

We **don't** have a ground truth $p(\boldsymbol{z}|\boldsymbol{x})$
… which is also what we're trying to find!



Intractable problem = a problem that can be solved in theory (e.g. given large but finite resources, especially time), but for which in practice any solution takes too many resources to be useful, is known as an intractable problem.

A chicken and egg problem



In order to have a tractable $p(\boldsymbol{x})$ we need a tractable $p(\boldsymbol{z}|\boldsymbol{x})$

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

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

In order to have a tractable $p(\boldsymbol{z}|\boldsymbol{x})$ we need a tractable $p(\boldsymbol{x})$

Can we find a surrogate?

An **approximate** posterior.
Parametrized by φ.

$$p_\theta(\boldsymbol{z}|\boldsymbol{x}) \approx q_\varphi(\boldsymbol{z}|\boldsymbol{x})$$

Our **true** posterior (that we can't evaluate due to its intractability)
Parametrized by θ.

Let's do some maths…

$$\log p_\theta(\boldsymbol{x}) = \log p_\theta\ (\boldsymbol{x})$$

$$= \log p_\theta\ (\boldsymbol{x}) \int q_\varphi(\boldsymbol{z}|\boldsymbol{x}) d\boldsymbol{z}$$

Multiply by 1

$$= \int \log p_\theta\ (\boldsymbol{x})\, q_\varphi(\boldsymbol{z}|\boldsymbol{x}) d\boldsymbol{z}$$

Bring inside the integral

$$= E_{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta\ (\boldsymbol{x})]$$

Definition of expectation

$$= E_{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p_\theta(\boldsymbol{x},\boldsymbol{z})}{p_\theta(\boldsymbol{z}|\boldsymbol{x})}\right]$$

Apply the equation $p_\theta(\boldsymbol{x}) = \dfrac{p_\theta(\boldsymbol{x},\boldsymbol{z})}{p_\theta(\boldsymbol{z}|\boldsymbol{x})}$

$$= E_{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p_\theta(\boldsymbol{x},\boldsymbol{z}) q_\varphi(\boldsymbol{z}|\boldsymbol{x})}{p_\theta(\boldsymbol{z}|\boldsymbol{x}) q_\varphi(\boldsymbol{z}|\boldsymbol{x})}\right]$$

Multiply by 1

$$= E_{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p_\theta(\boldsymbol{x},\boldsymbol{z})}{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}\right] + E_{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}{p_\theta(\boldsymbol{z}|\boldsymbol{x})}\right]$$

Split the expectation

$$= E_{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p_\theta(\boldsymbol{x},\boldsymbol{z})}{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}\right] + \underbrace{D_{KL}\left(q_\varphi(\boldsymbol{z}|\boldsymbol{x}) \| p_\theta(\boldsymbol{z}|\boldsymbol{x})\right)}_{\geq 0}$$

Definition of KL divergence

What can we infer?

$$\log p_\theta(\boldsymbol{x}) = \underbrace{E_{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p_\theta(\boldsymbol{x},\boldsymbol{z})}{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}\right]}_{\textbf{ELBO}} + \underbrace{D_{KL}\left(q_\varphi(\boldsymbol{z}|\boldsymbol{x}) \| p_\theta(\boldsymbol{z}|\boldsymbol{x})\right)}_{\geq 0}$$

ELBO = Evidence Lower Bound

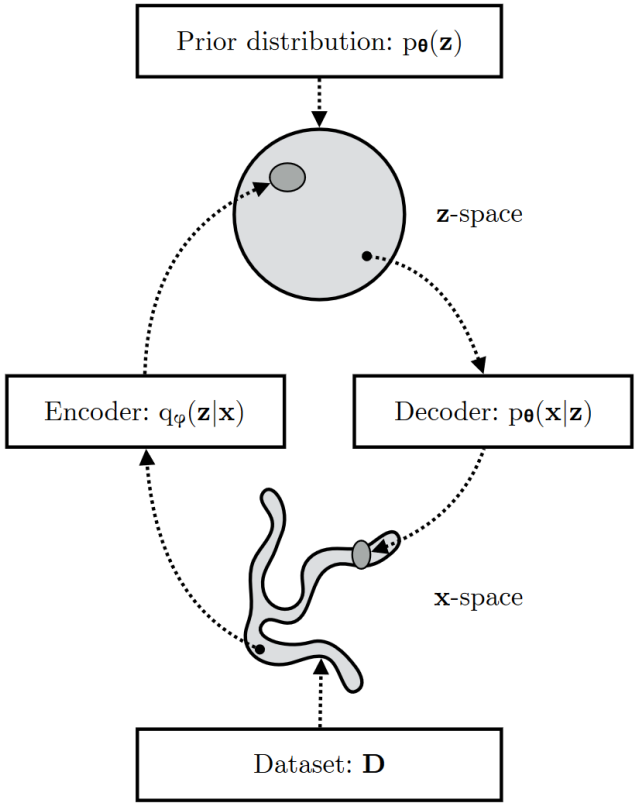
Total Compensation = Base Salary + $\underbrace{\text{Bonus}}_{\geq 0}$

We can for sure deduce the following:

Total Compensation $\geq$ Base Salary

$$\log p_\theta(\boldsymbol{x}) \geq E_{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p_\theta(\boldsymbol{x},\boldsymbol{z})}{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}\right]$$

ELBO in detail



$$\log p_\theta(\boldsymbol{x}) \geq E_{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p_\theta(\boldsymbol{x},\boldsymbol{z})}{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}\right] = E_{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p_\theta(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}\right]$$ Chain rule of probability

$$= E_{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})] + E_{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{z})}{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}\right]$$ Split the expectation

$$= E_{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})] - D_{KL}\left(q_\varphi(\boldsymbol{z}|\boldsymbol{x})\|p_\theta(\boldsymbol{z})\right)$$ Definition of KL divergence

Profit = Revenue - Costs

Maximizing the ELBO means:

1. Maximizing the first term: maximizing the reconstruction likelihood of the decoder
2. Minimizing the second term: minimizing the distance between the learned distribution and the prior belief we have over the latent variable.

Kingma, D.P. and Welling, M., 2019. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, *12*(4), pp.307-392.

Maximizing the ELBO: A little introduction to estimators

- When we have a function we want to **maximize**, we usually take the gradient and adjust the weights of the model so that they move **along** the gradient direction.
- When we have a function we want to **minimize**, we usually take the gradient, and adjust the weights of the model so that they move **against** the gradient direction.

Stochastic Gradient Descent

When used to minimize the above function, a standard (or "batch") gradient descent method would perform the following iterations:

$$w := w - \eta \nabla Q(w) = w - \frac{\eta}{n} \sum_{i=1}^{n} \nabla Q_i(w),$$

where η is a step size (sometimes called the *learning rate* in machine learning).

SGD is stochastic because we choose the minibatch randomly from our dataset and we then average the loss over the minibatch

$$L(\theta, \varphi, \boldsymbol{x}) = E_{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p_\theta(\boldsymbol{x}, \boldsymbol{z})}{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}\right] = E_{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})] - D_{KL}\left(q_\varphi(\boldsymbol{z}|\boldsymbol{x}) \| p_\theta(\boldsymbol{z})\right)$$

Umar Jamil - https://github.com/hkproj/vae-from-scratch-notes

How to maximize the ELBO?

ELBO

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}^{(i)}) = -D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})||p_{\boldsymbol{\theta}}(\mathbf{z})) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z})\right] \quad (3)$$

We want to differentiate and optimize the lower bound $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}^{(i)})$ w.r.t. both the variational parameters $\boldsymbol{\phi}$ and generative parameters $\boldsymbol{\theta}$. However, the gradient of the lower bound w.r.t. $\boldsymbol{\phi}$ is a bit problematic. The usual (naïve) Monte Carlo gradient estimator for this type of problem is: $\nabla_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z})}\left[f(\mathbf{z})\right] = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z})}\left[f(\mathbf{z}) \nabla_{q_{\boldsymbol{\phi}}(\mathbf{z})} \log q_{\boldsymbol{\phi}}(\mathbf{z})\right] \simeq \frac{1}{L} \sum_{l=1}^{L} f(\mathbf{z}) \nabla_{q_{\boldsymbol{\phi}}(\mathbf{z}^{(l)})} \log q_{\boldsymbol{\phi}}(\mathbf{z}^{(l)})$ where $\mathbf{z}^{(l)} \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})$. This gradient estimator exhibits exhibits very high variance (see e.g. [BJP12]) and is impractical for our purposes.

SCORE estimator

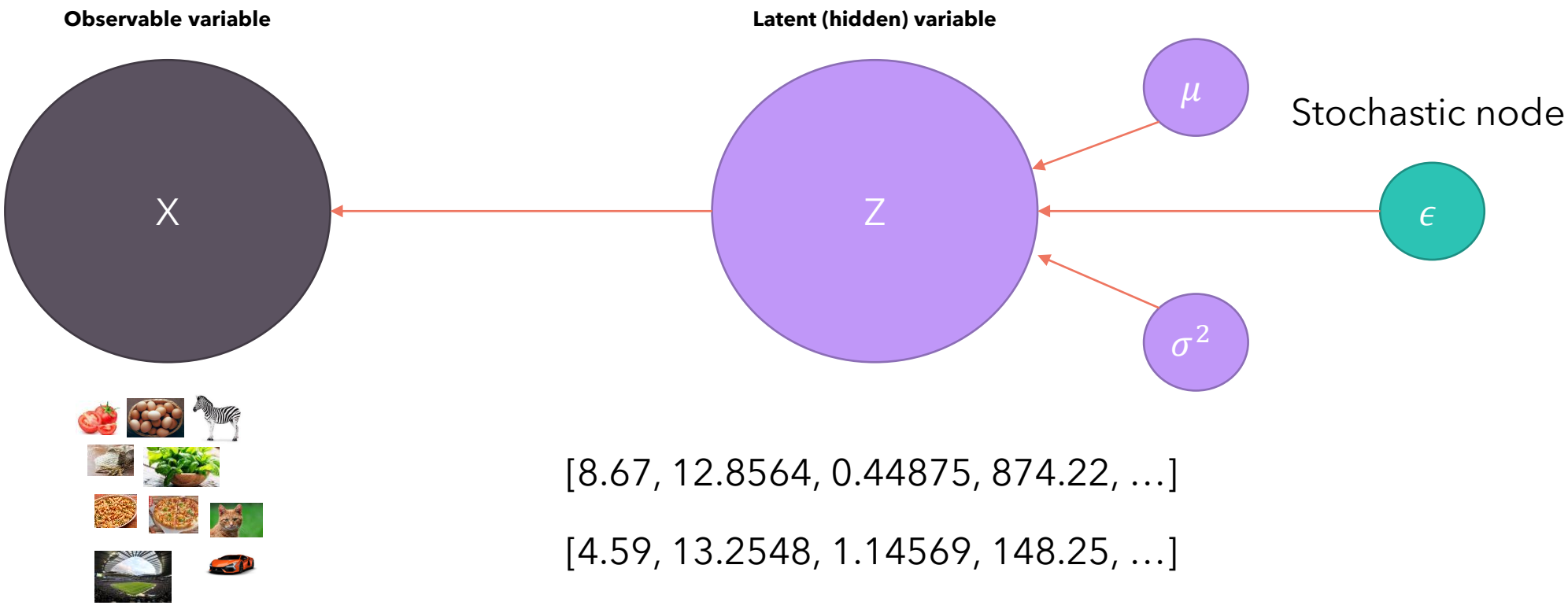
Kingma, D.P. and Welling, M., 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

This estimator is **unbiased**, meaning that even if at every step it may not be equal to the true expectation, on average it will converge to it, but as it is stochastic, it also has a variance and it happens to be high for practical use. Plus, we can't run backpropagation through it!

We need a new estimator!

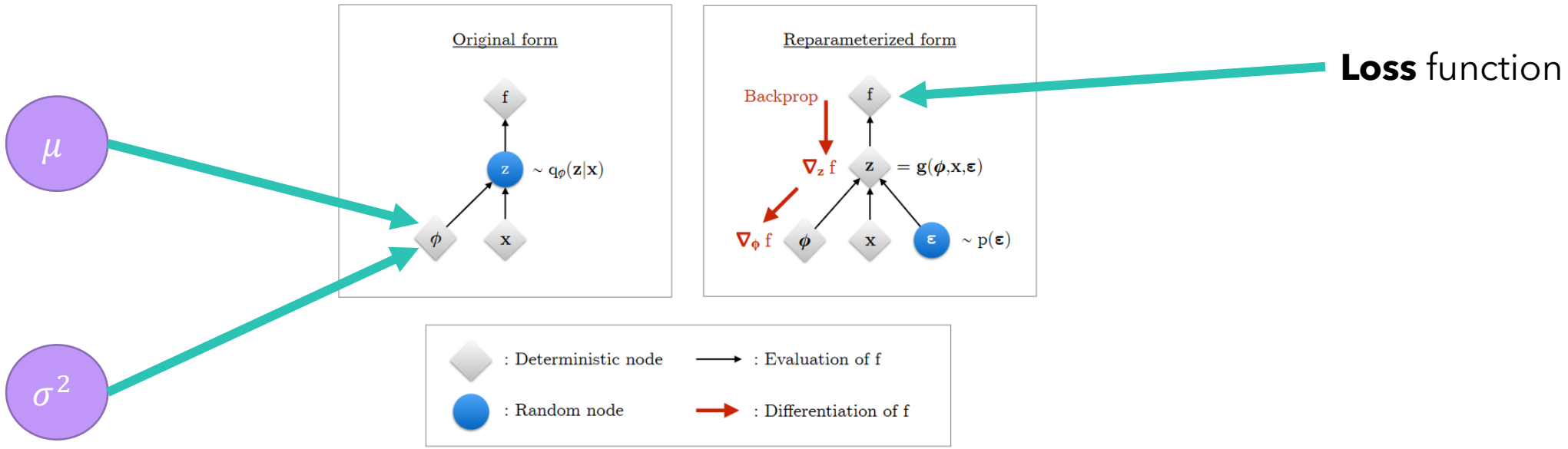
Umar Jamil - https://github.com/hkproj/vae-from-scratch-notes

The reparameterization trick



Umar Jamil - https://github.com/hkproj/vae-from-scratch-notes

Running backpropagation on the reparametrized model



Loss function

Figure 2.3: Illustration of the reparameterization trick. The variational parameters ϕ affect the objective f through the random variable $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$. We wish to compute gradients $\nabla_\phi f$ to optimize the objective with SGD. In the original form (left), we cannot differentiate f w.r.t. ϕ, because we cannot directly backpropagate gradients through the random variable $\mathbf{z}$. We can 'externalize' the randomness in $\mathbf{z}$ by re-parameterizing the variable as a deterministic and differentiable function of ϕ, $\mathbf{x}$, and a newly introduced random variable ϵ. This allows us to 'backprop through $\mathbf{z}$', and compute gradients $\nabla_\phi f$.

Kingma, D.P. and Welling, M., 2019. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, *12*(4), pp.307-392.

Umar Jamil - https://github.com/hkproj/vae-from-scratch-notes

A new estimator!

$$L(\theta, \varphi, \boldsymbol{x}) = \underbrace{E_{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p_\theta(\boldsymbol{x}, \boldsymbol{z})}{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}\right]}_{\text{ELBO}} = E_{p(\epsilon)}\left[\log \frac{p_\theta(\boldsymbol{x}, \boldsymbol{z})}{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}\right]$$

$$\epsilon \approx p(\epsilon)$$

$$z = g(\varphi, x, \epsilon)$$

$$E_{p(\epsilon)}\left[\log \frac{p_\theta(\boldsymbol{x}, \boldsymbol{z})}{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}\right] \cong \tilde{L}(\theta, \varphi, \boldsymbol{x}) = \log \frac{p_\theta(\boldsymbol{x}, \boldsymbol{z})}{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}$$

aging the approximate posterior to be close to the prior $p_{\boldsymbol{\theta}}(\mathbf{z})$. This yields a second version of the SGVB estimator $\widetilde{\mathcal{L}}^B(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}^{(i)}) \simeq \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}^{(i)})$, corresponding to eq. (3), which typically has less variance than the generic estimator:

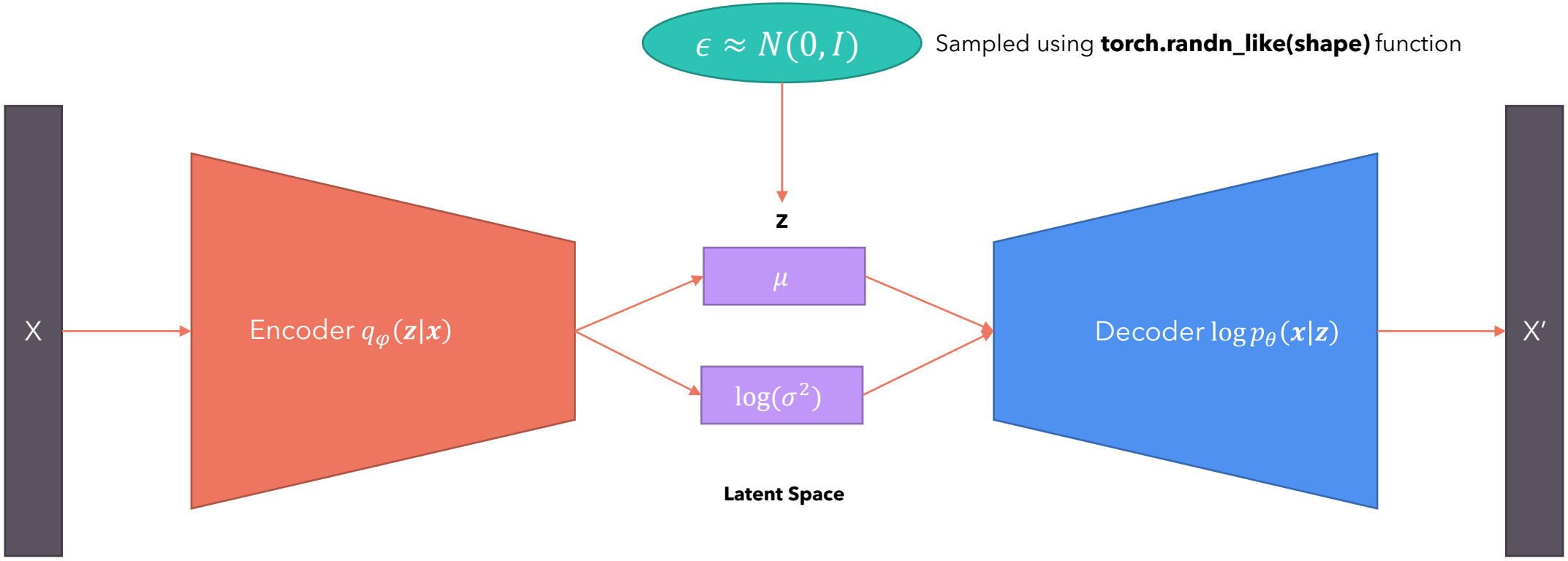
Kingma, D.P. and Welling, M., 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Is the new estimator unbiased?

$$L(\theta, \varphi, \boldsymbol{x}) = E_{p(\epsilon)}\left[\log \frac{p_\theta(\boldsymbol{x}, \boldsymbol{z})}{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}\right] \qquad \tilde{L}(\theta, \varphi, \boldsymbol{x}) = \log \frac{p_\theta(\boldsymbol{x}, \boldsymbol{z})}{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}$$

$$\begin{aligned} E_{p(\epsilon)}\left[\nabla_{\theta,\varphi} \tilde{L}(\theta, \varphi, \boldsymbol{x})\right] &= E_{p(\epsilon)}\left[\nabla_{\theta,\varphi} \log \frac{p_\theta(\boldsymbol{x}, \boldsymbol{z})}{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}\right] \\ &= \nabla_{\theta,\varphi}\left(E_{p(\epsilon)}\left[\log \frac{p_\theta(\boldsymbol{x}, \boldsymbol{z})}{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}\right]\right) \\ &= \nabla_{\theta,\varphi}(L(\theta, \varphi, \boldsymbol{x})) \end{aligned}$$

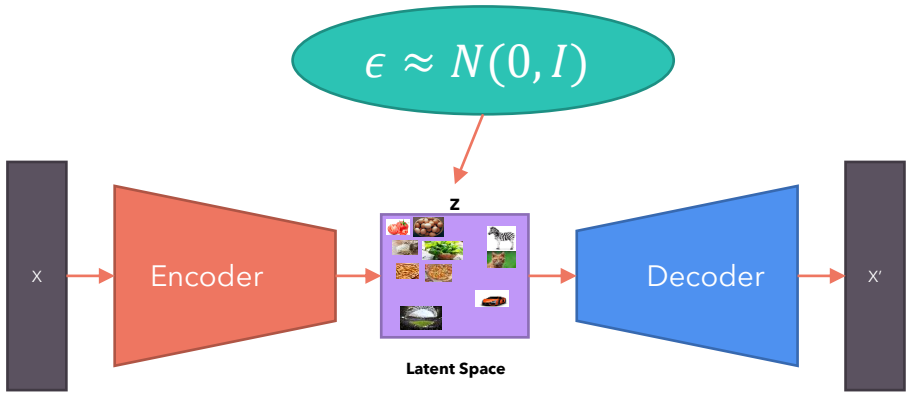
Example network



We prefer learning $\log(\sigma^2)$ because it can be negative, so the model doesn't need to be forced to produce only positive values for it.

Show me the loss already!

MLP = Multi Layer Perceptron



Let the prior over the latent variables be the centered isotropic multivariate Gaussian $p_{\boldsymbol{\theta}}(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$. Note that in this case, the prior lacks parameters. We let $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ be a multivariate Gaussian (in case of real-valued data) or Bernoulli (in case of binary data) whose distribution parameters are computed from $\mathbf{z}$ with a MLP (a fully-connected neural network with a single hidden layer, see appendix C). Note the true posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ is in this case intractable. While there is much freedom in the form $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$, we'll assume the true (but intractable) posterior takes on a approximate Gaussian form with an approximately diagonal covariance. In this case, we can let the variational approximate posterior be a multivariate Gaussian with a diagonal covariance structure[2]:

$$\log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)}) = \log \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{2(i)}\mathbf{I}) \tag{9}$$

where the mean and s.d. of the approximate posterior, $\boldsymbol{\mu}^{(i)}$ and $\boldsymbol{\sigma}^{(i)}$, are outputs of the encoding MLP, i.e. nonlinear functions of datapoint $\mathbf{x}^{(i)}$ and the variational parameters $\boldsymbol{\phi}$ (see appendix C).

As explained in section 2.4, we sample from the posterior $\mathbf{z}^{(i,l)} \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})$ using $\mathbf{z}^{(i,l)} = g_{\boldsymbol{\phi}}(\mathbf{x}^{(i)}, \boldsymbol{\epsilon}^{(l)}) = \boldsymbol{\mu}^{(i)} + \boldsymbol{\sigma}^{(i)} \odot \boldsymbol{\epsilon}^{(l)}$ where $\boldsymbol{\epsilon}^{(l)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. With $\odot$ we signify an element-wise product. In this model both $p_{\boldsymbol{\theta}}(\mathbf{z})$ (the prior) and $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ are Gaussian; in this case, we can use the estimator of eq. (7) where the KL divergence can be computed and differentiated without estimation (see appendix B). The resulting estimator for this model and datapoint $\mathbf{x}^{(i)}$ is:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}^{(i)}) \simeq \frac{1}{2}\sum_{j=1}^{J}\left(1 + \log((\sigma_j^{(i)})^2) - (\mu_j^{(i)})^2 - (\sigma_j^{(i)})^2\right) + \frac{1}{L}\sum_{l=1}^{L}\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z}^{(i,l)})$$

$$\text{where} \quad \mathbf{z}^{(i,l)} = \boldsymbol{\mu}^{(i)} + \boldsymbol{\sigma}^{(i)} \odot \boldsymbol{\epsilon}^{(l)} \quad \text{and} \quad \boldsymbol{\epsilon}^{(l)} \sim \mathcal{N}(0, \mathbf{I}) \tag{10}$$

As explained above and in appendix C, the decoding term $\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z}^{(i,l)})$ is a Bernoulli or Gaussian MLP, depending on the type of data we are modelling.

Kingma, D.P. and Welling, M., 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

How to derive the loss function?

48

The encoder distribution is $q(z|x) = \mathcal{N}(z|\mu(x), \Sigma(x))$ where $\Sigma = \mathrm{diag}(\sigma_1^2, \ldots, \sigma_n^2)$.

The latent prior is given by $p(z) = \mathcal{N}(0, I)$.

Both are multivariate Gaussians of dimension n, for which in general the KL divergence is:

$$\mathfrak{D}_{\mathrm{KL}}[p_1 \mid\mid p_2] = \frac{1}{2}\left[\log\frac{|\Sigma_2|}{|\Sigma_1|} - n + \mathrm{tr}\{\Sigma_2^{-1}\Sigma_1\} + (\mu_2 - \mu_1)^T\Sigma_2^{-1}(\mu_2 - \mu_1)\right]$$

where $p_1 = \mathcal{N}(\mu_1, \Sigma_1)$ and $p_2 = \mathcal{N}(\mu_2, \Sigma_2)$.

In the VAE case, $p_1 = q(z|x)$ and $p_2 = p(z)$, so $\mu_1 = \mu$, $\Sigma_1 = \Sigma$, $\mu_2 = \vec{0}$, $\Sigma_2 = I$. Thus:

$$\begin{aligned}
\mathfrak{D}_{\mathrm{KL}}[q(z|x) \mid\mid p(z)] &= \frac{1}{2}\left[\log\frac{|\Sigma_2|}{|\Sigma_1|} - n + \mathrm{tr}\{\Sigma_2^{-1}\Sigma_1\} + (\mu_2 - \mu_1)^T\Sigma_2^{-1}(\mu_2 - \mu_1)\right] \\
&= \frac{1}{2}\left[\log\frac{|I|}{|\Sigma|} - n + \mathrm{tr}\{I^{-1}\Sigma\} + (\vec{0} - \mu)^T I^{-1}(\vec{0} - \mu)\right] \\
&= \frac{1}{2}\left[-\log|\Sigma| - n + \mathrm{tr}\{\Sigma\} + \mu^T\mu\right] \\
&= \frac{1}{2}\left[-\log\prod_i \sigma_i^2 - n + \sum_i \sigma_i^2 + \sum_i \mu_i^2\right] \\
&= \frac{1}{2}\left[-\sum_i \log\sigma_i^2 - n + \sum_i \sigma_i^2 + \sum_i \mu_i^2\right] \\
&= \frac{1}{2}\left[-\sum_i \left(\log\sigma_i^2 + 1\right) + \sum_i \sigma_i^2 + \sum_i \mu_i^2\right]
\end{aligned}$$

https://stats.stackexchange.com/questions/318748/deriving-the-kl-divergence-loss-for-vaes

Thanks for watching!
Don’t forget to subscribe for more amazing content on AI and Machine Learning!