

---

# Deep learning

Unpacking Transformers, LLMs and image generation

Session 6

# Syllabus

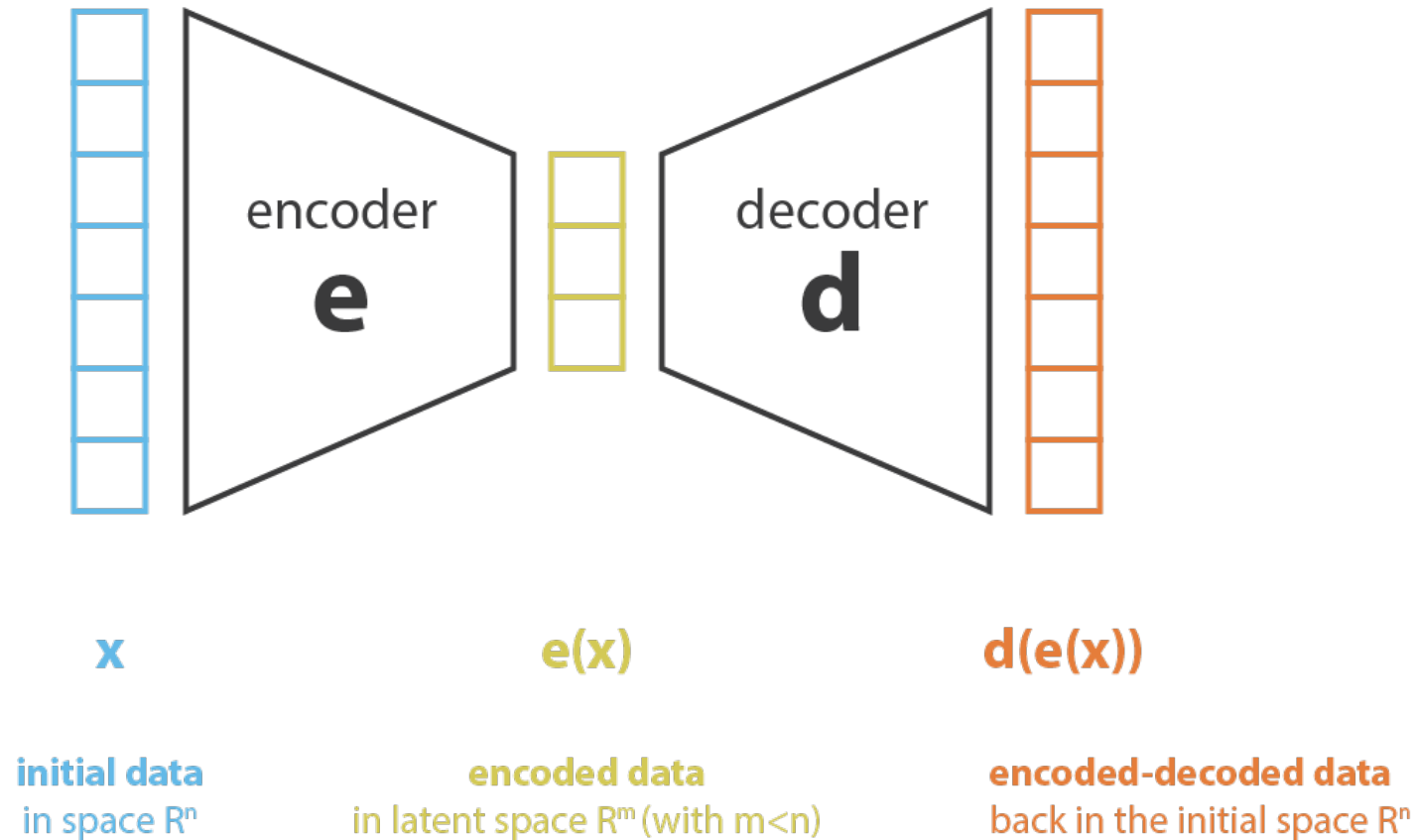
Session	Lecture	TP
1	Intro to DL Gradient descent and backprop	Intro to micrograd  (*) fit the $x^2$ function
2	DL fundamentals I <ul style="list-style-type: none"><li>• Backprop</li><li>• Loss functions</li><li>• Neural Probabilistic Language Model (Bengio 2003)</li></ul>	Bigram model and MLP for next-character prediction  (*) extend to tri-gram
3	DL fundamentals II <ul style="list-style-type: none"><li>• Activation function</li><li>• Regularization</li><li>• Initialization</li><li>• Residual networks</li><li>• Normalization</li></ul> Recurrent Neural Networks	Backprop ninja MLP in pytorch  (*) add batchnorm to TP2
4	Attention and Transformers	GPT from scratch
5	DL for computer vision: convnets, unets	Convnets for CIFAR-10
6	VAE & Diffusion models	Diffusion from scratch  <b>Quiz</b>

# Agenda for today

---

1. VAEs
2. Diffusion

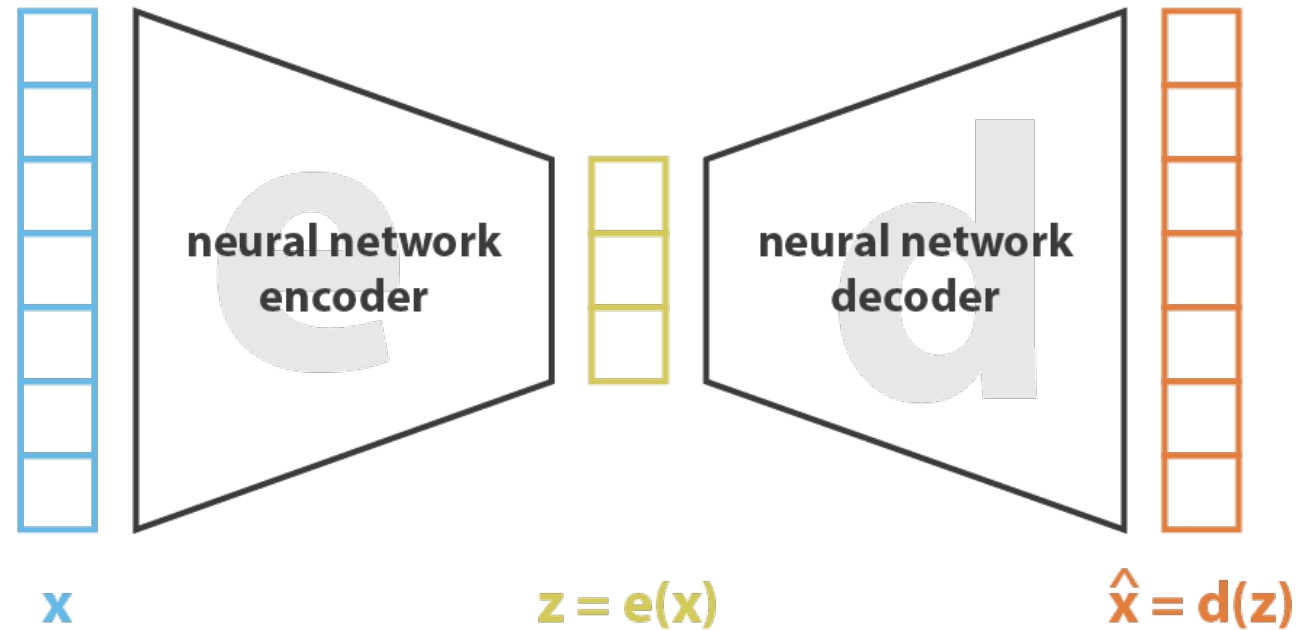
# Auto-encoders



$x = d(e(x))$  ➔ **lossless encoding**  
no information is lost  
when reducing the  
number of dimensions

$x \neq d(e(x))$  ➔ **lossy encoding**  
some information is lost  
when reducing the  
number of dimensions and  
can't be recovered later

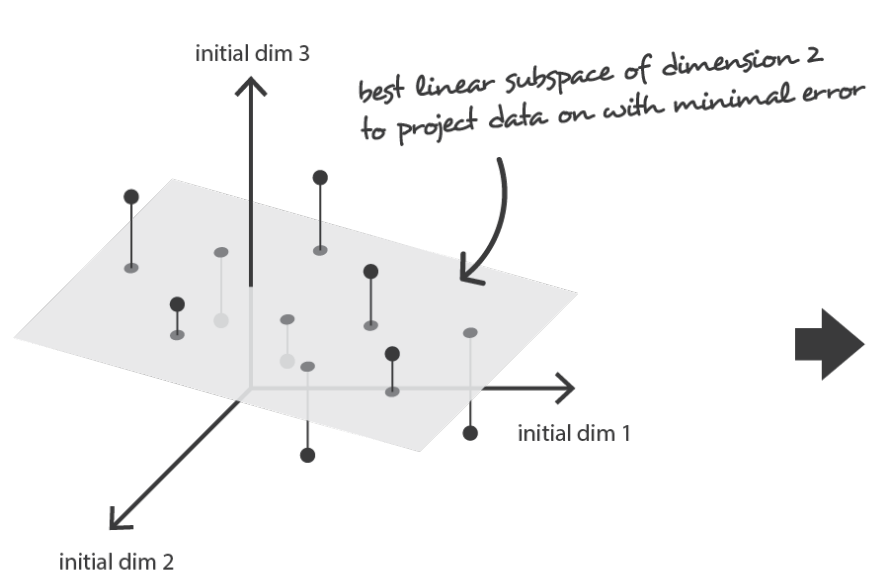
# Auto-encoders



---

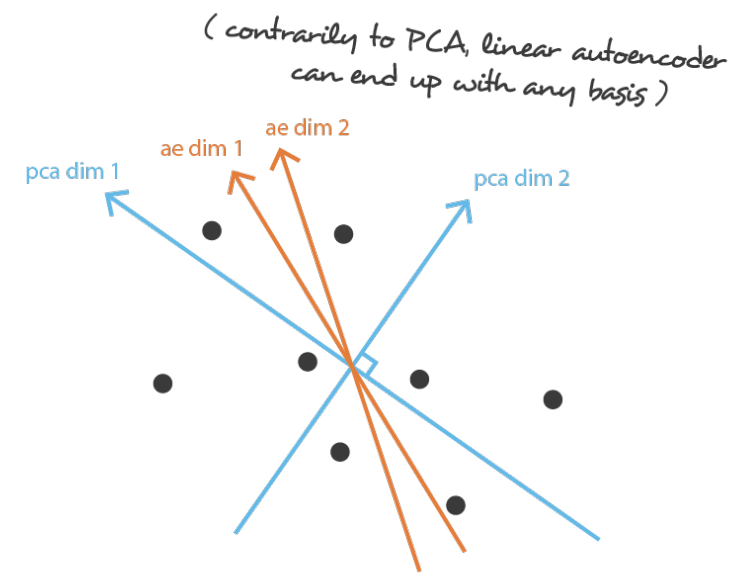
$$\text{loss} = || \mathbf{x} - \hat{\mathbf{x}} ||^2 = || \mathbf{x} - d(\mathbf{z}) ||^2 = || \mathbf{x} - d(e(\mathbf{x})) ||^2$$

# Auto-encoders



**Data in the full initial space**

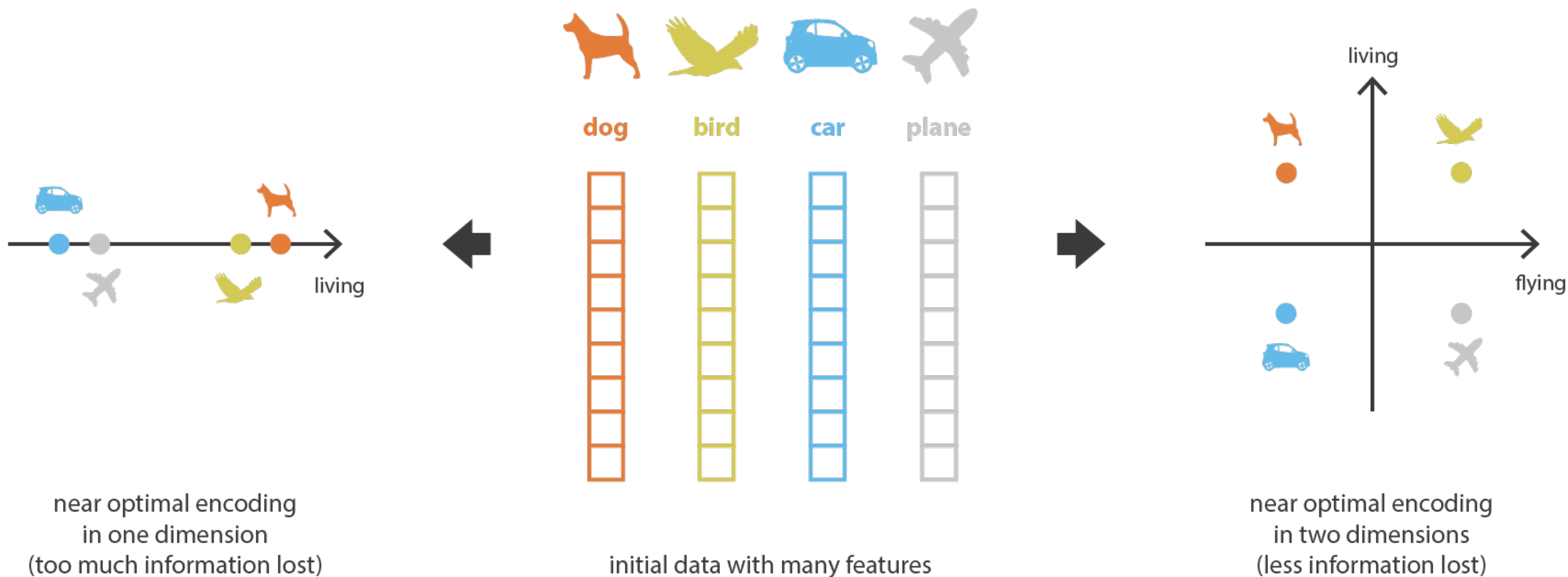
In order to reduce dimensionality, PCA and linear autoencoder target, in theory, the same optimal subspace to project data on...



**Data projected on the best linear subspace**

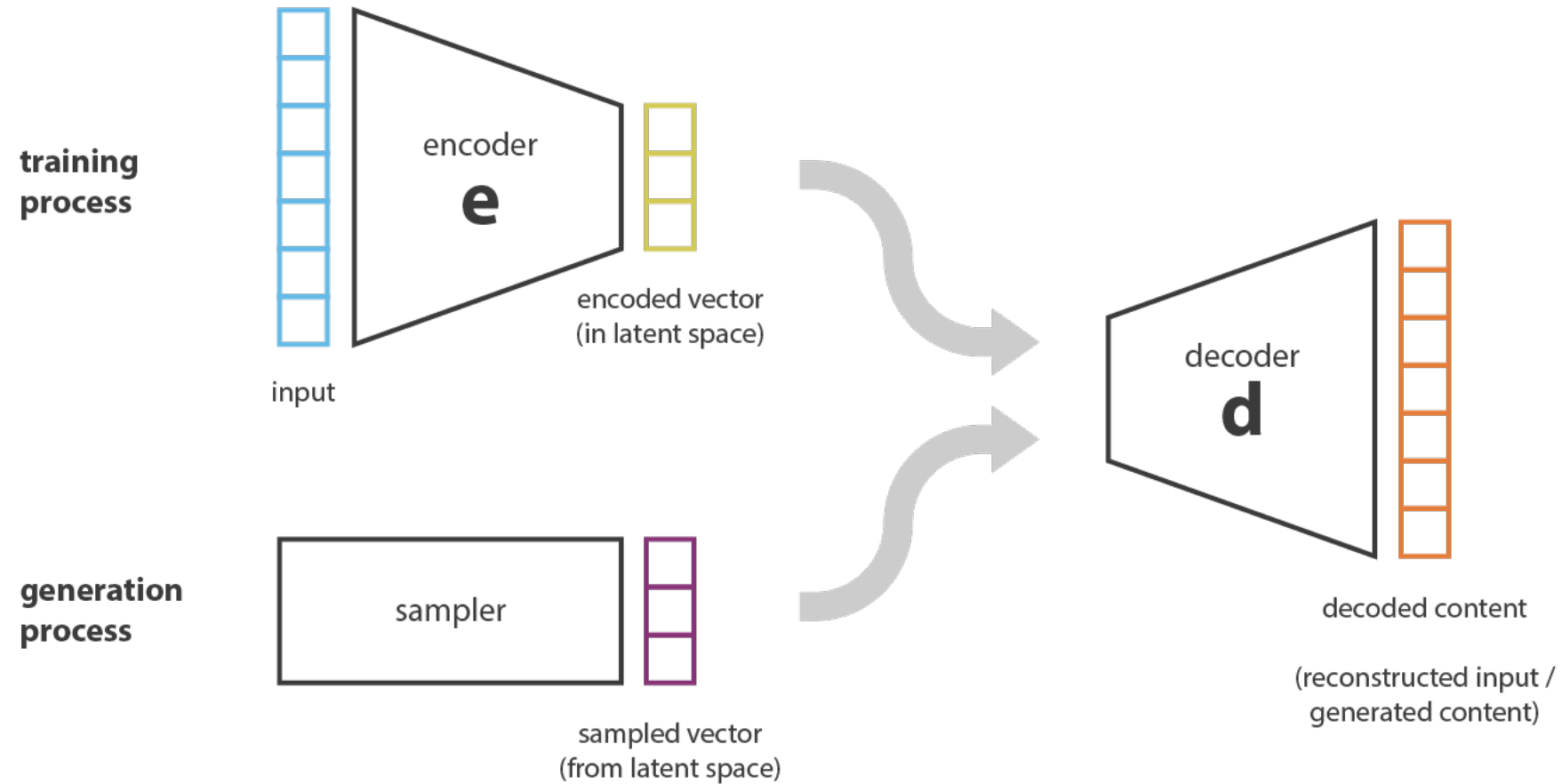
... but not necessarily with the same basis due to different constraints (in PCA the first component is the one that explains the maximum of variance and components are orthogonal)

# Auto-encoders



When reducing dimensionality, we want to keep the main structure there exists among the data.

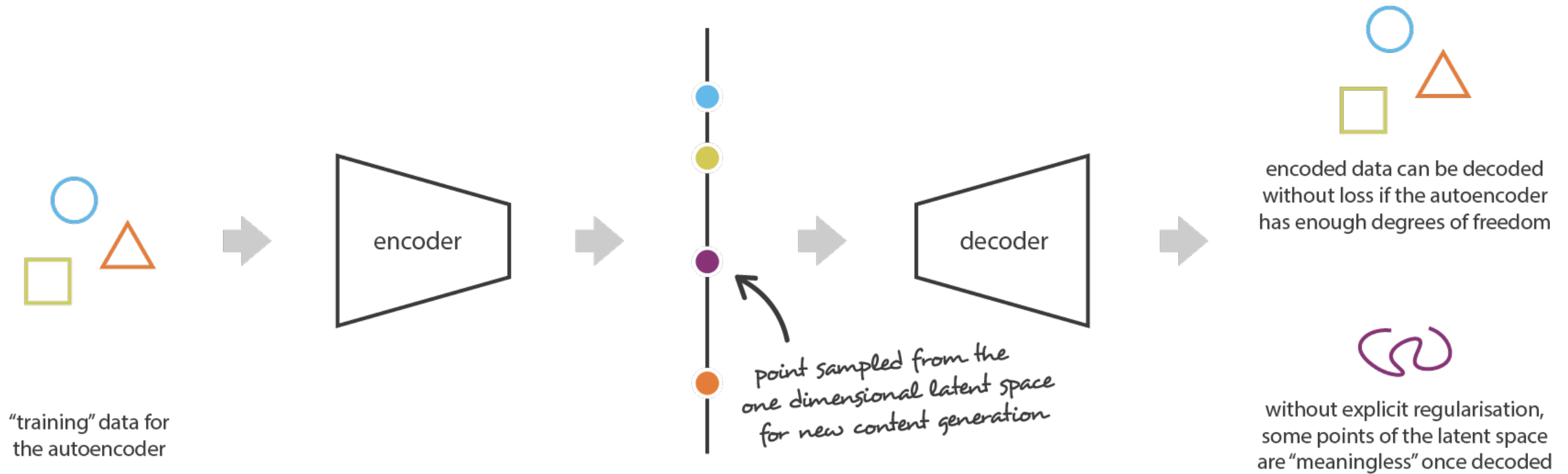
# Auto-encoders



In theory, auto-encoders can be used for data generation.



# Auto-encoders



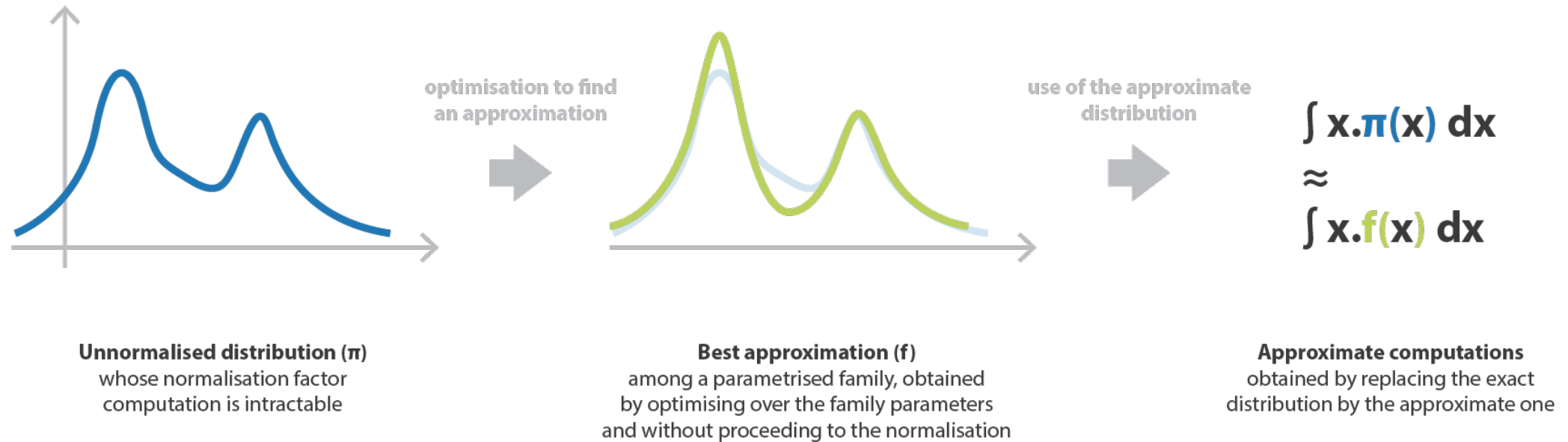
Irregular latent space prevent us from using autoencoder for new content generation.

# Variational auto-encoders

---

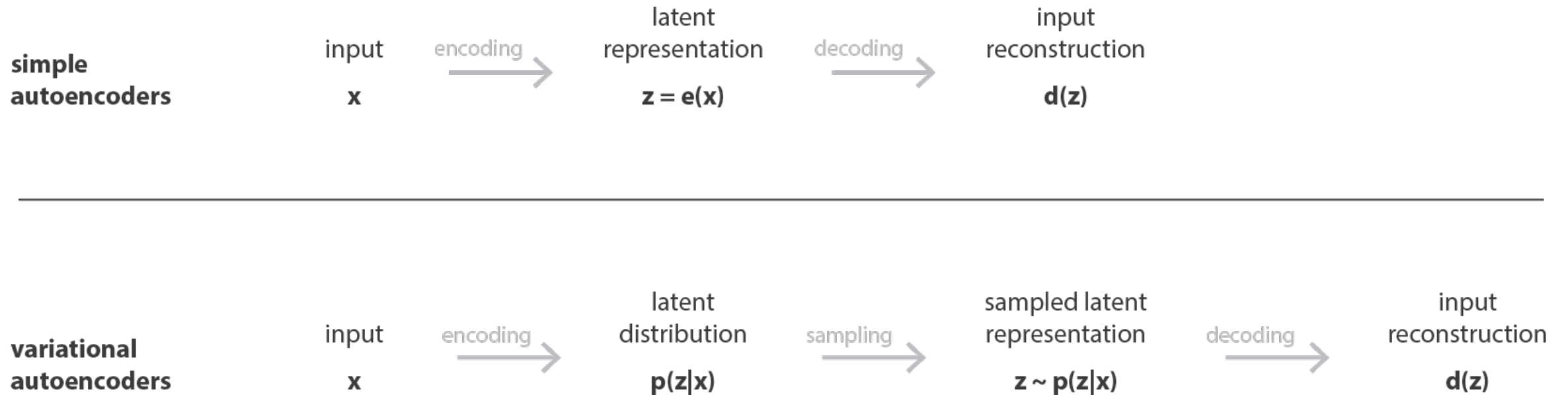
A variational autoencoder can be defined as being an autoencoder whose training is regularised to avoid overfitting and ensure that the latent space has good properties that enable generative process.

# Variational inference



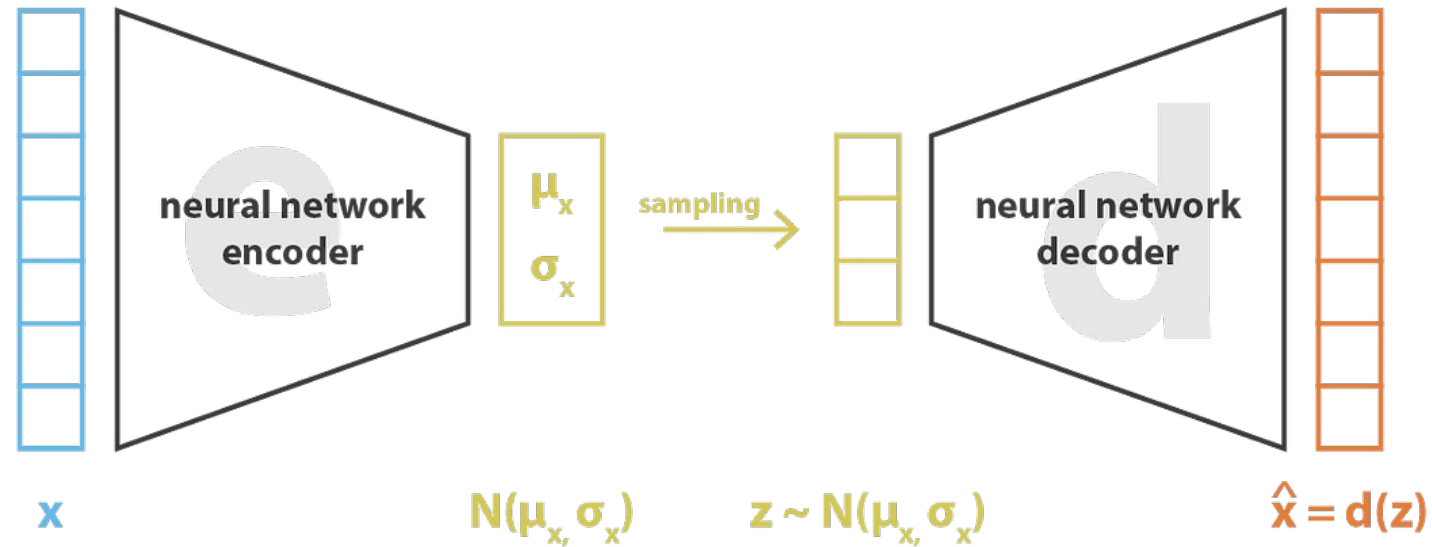
Variational Inference methods that consist in finding the best approximation of a distribution among a parametrised family.

# Variational Auto-encoders (VAEs)



Difference between autoencoder (deterministic) and variational autoencoder (probabilistic).

# Variational Auto-encoders (VAEs)

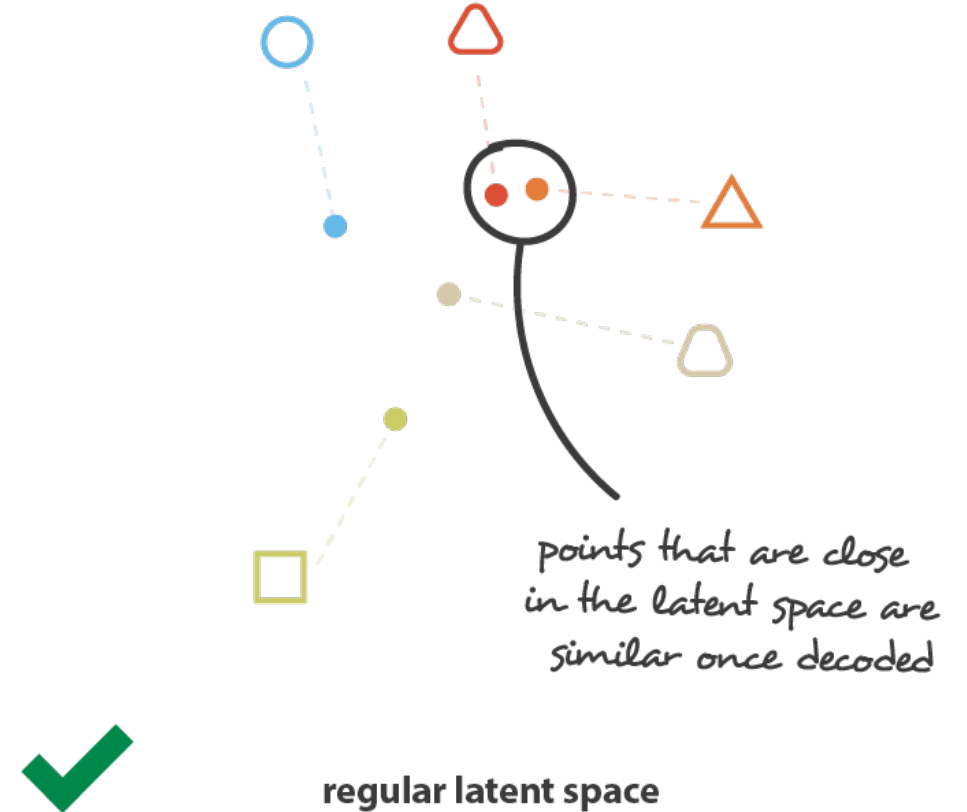
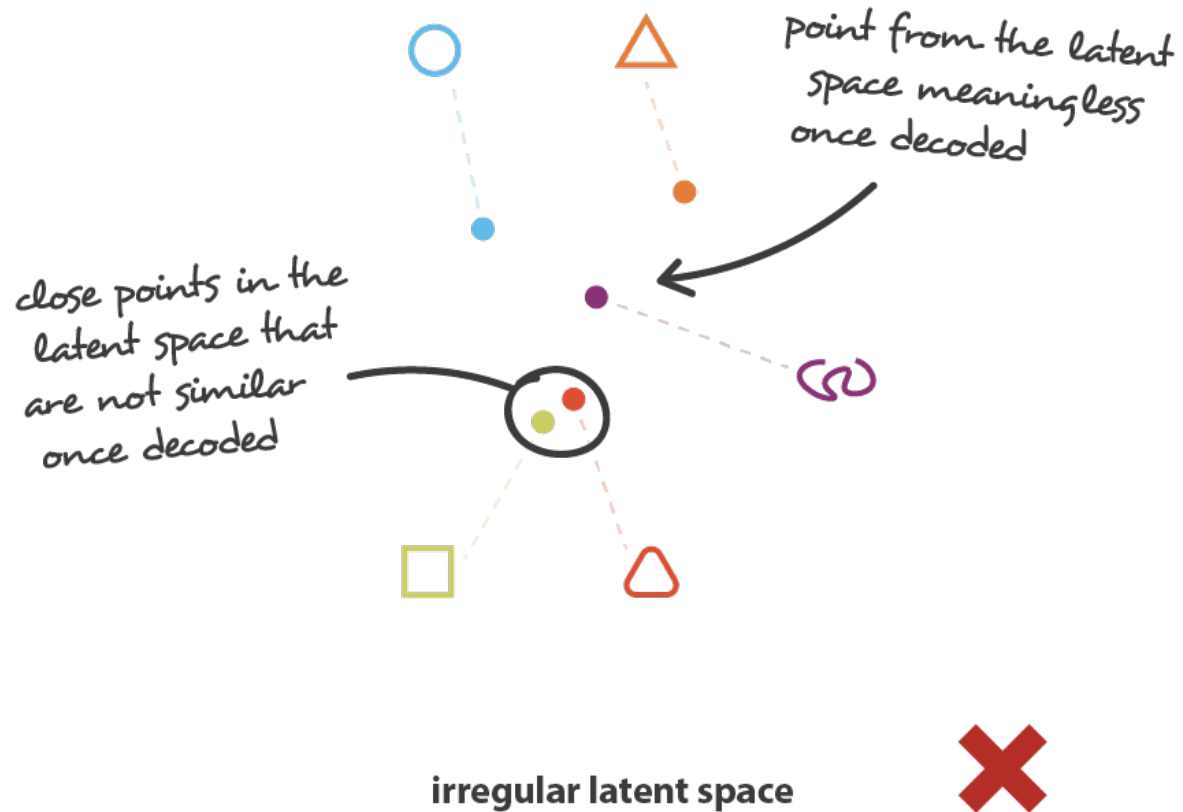


---

$$\text{loss} = ||x - \hat{x}||^2 + \text{KL}[N(\mu_x, \sigma_x), N(0, I)] = ||x - d(z)||^2 + \text{KL}[N(\mu_x, \sigma_x), N(0, I)]$$

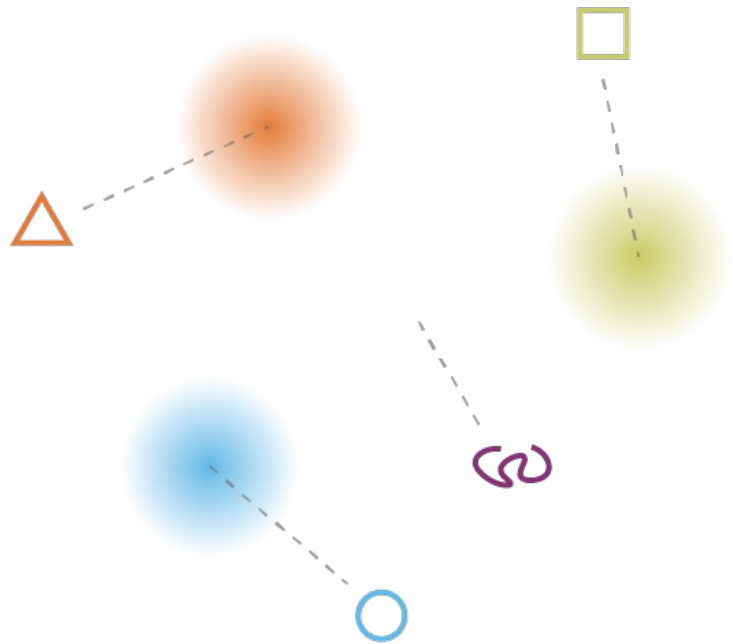
In variational autoencoders, the loss function is composed of a **reconstruction term** (that makes the encoding-decoding scheme efficient) and a **regularisation term** (that makes the latent space regular).

# Variational Auto-encoders (VAEs)

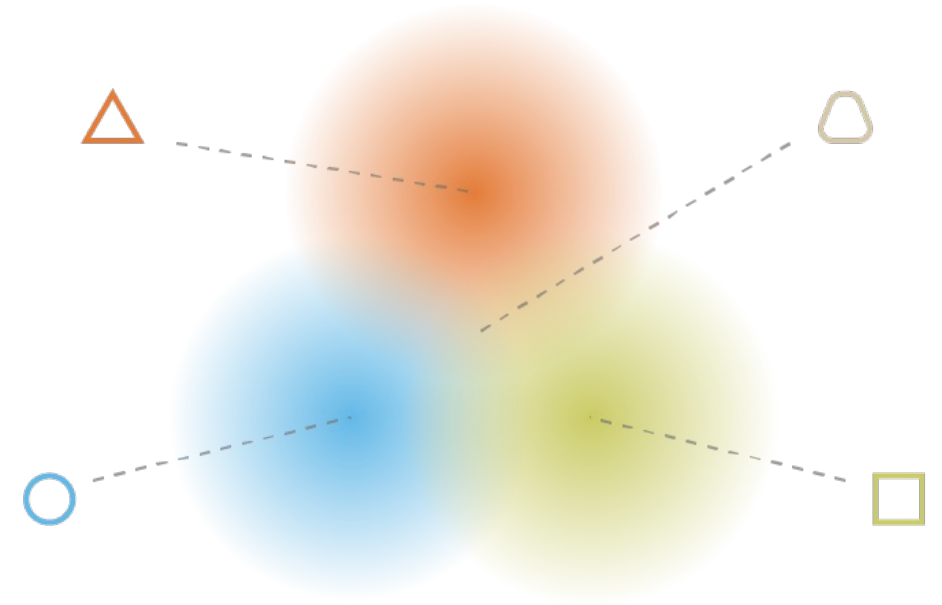


Difference between a “regular” and an “irregular” latent space.

# Variational Auto-encoders (VAEs)



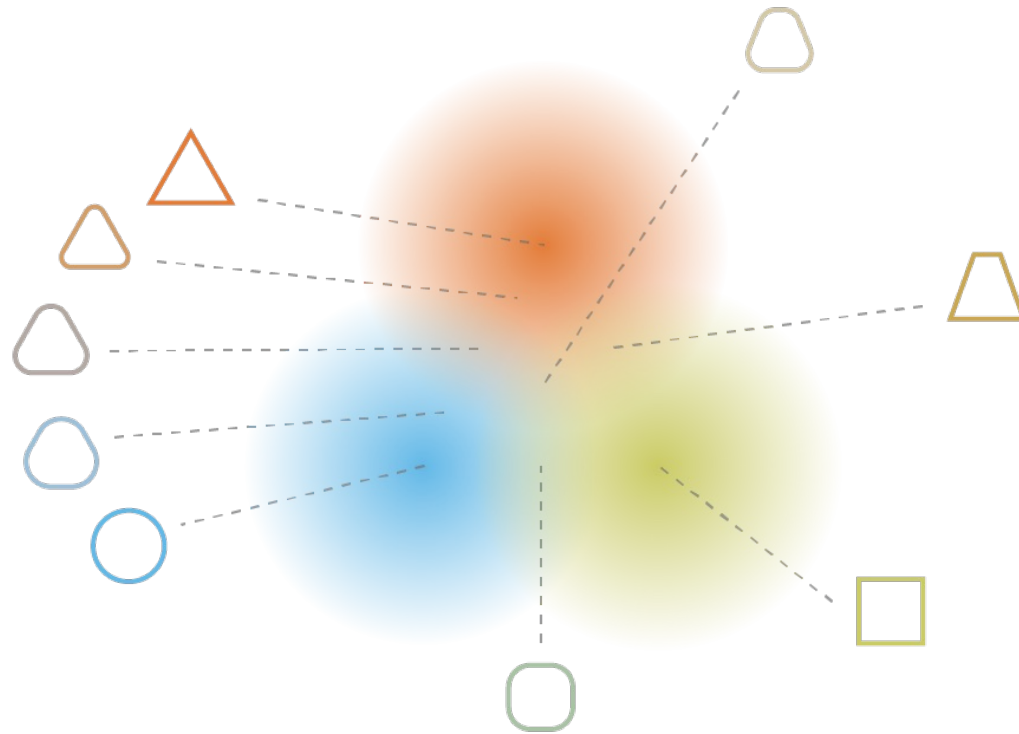
what can happen without regularisation



what we want to obtain with regularisation

The returned distributions of VAEs have to be regularised to obtain a latent space with good properties.

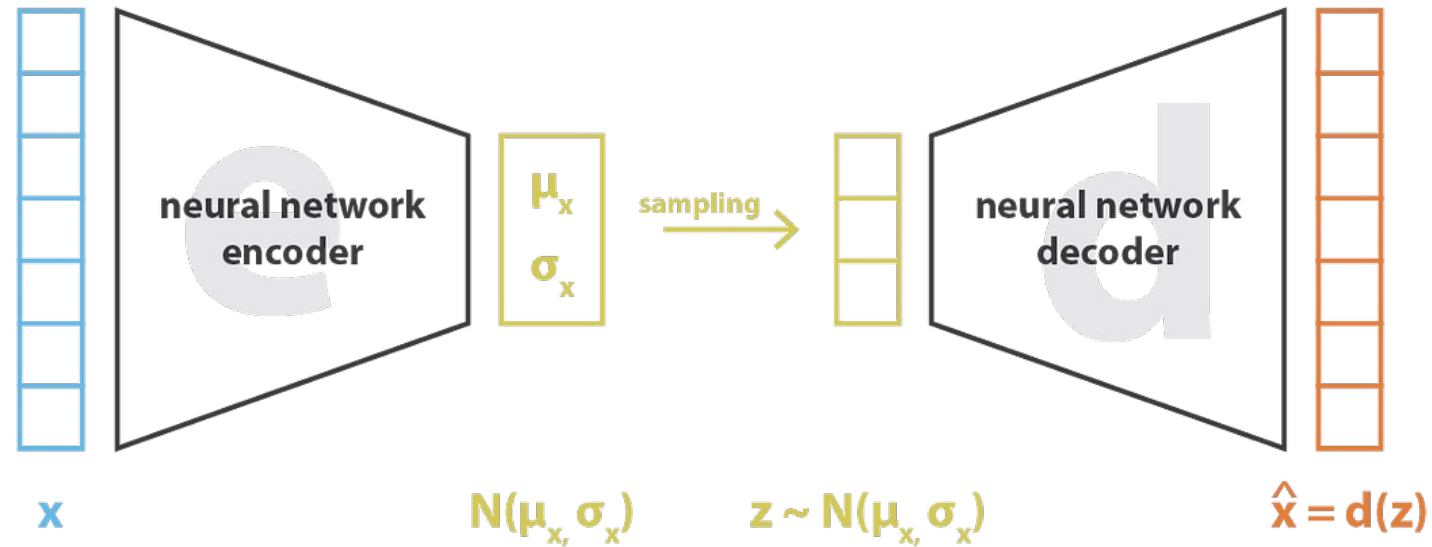
# Variational Auto-encoders (VAEs)



Regularisation tends to create a “gradient” over the information encoded in the latent space.



# Variational Auto-encoders (VAEs)

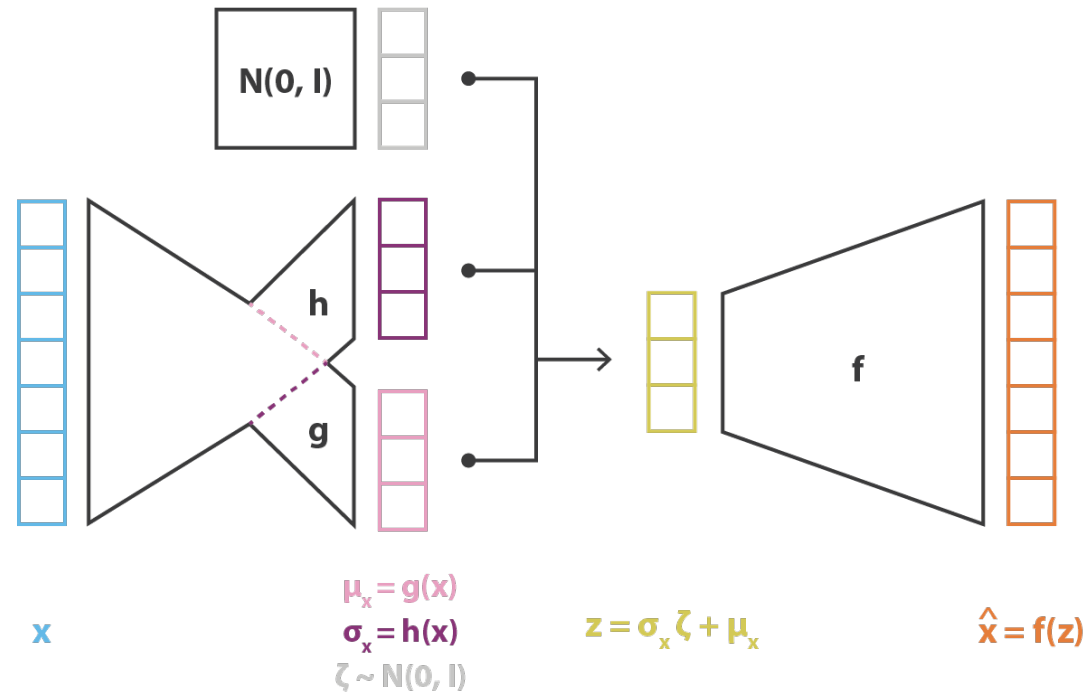


---

$$\text{loss} = ||x - \hat{x}||^2 + \text{KL}[N(\mu_x, \sigma_x), N(0, I)] = ||x - d(z)||^2 + \text{KL}[N(\mu_x, \sigma_x), N(0, I)]$$

In variational autoencoders, the loss function is composed of a **reconstruction term** (that makes the encoding-decoding scheme efficient) and a **regularisation term** (that makes the latent space regular).

# Variational Auto-encoders (VAEs)



---

$$\text{loss} = C \|x - \hat{x}\|^2 + \text{KL}[N(\mu_x, \sigma_x), N(0, I)] = C \|x - f(z)\|^2 + \text{KL}[N(g(x), h(x)), N(0, I)]$$

# VAEs for image generation

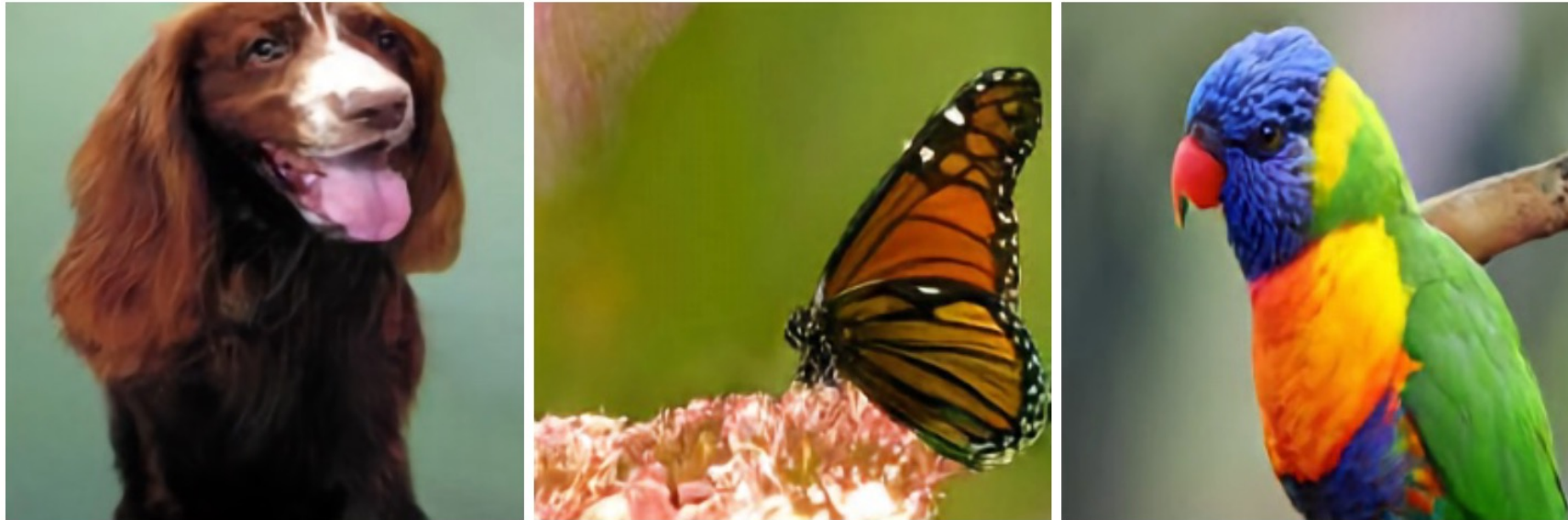


Figure 1: Class-conditional 256x256 image samples from a two-level model trained on ImageNet.

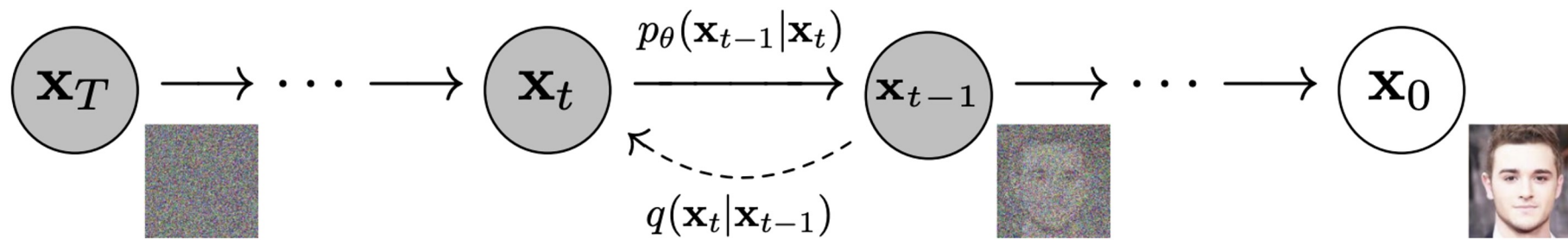
# Agenda for today

---

1. VAEs

2. Diffusion

# The principle of diffusion



# Schedulers

Schedulers define the methodology for iteratively adding noise to an image or for updating a sample based on model outputs.

- How to add noise for training
- How to update a sample based on an output from a pretrained model for inference

Schedulers are often defined by a *noise schedule* and an *update rule* to solve the differential equation solution.

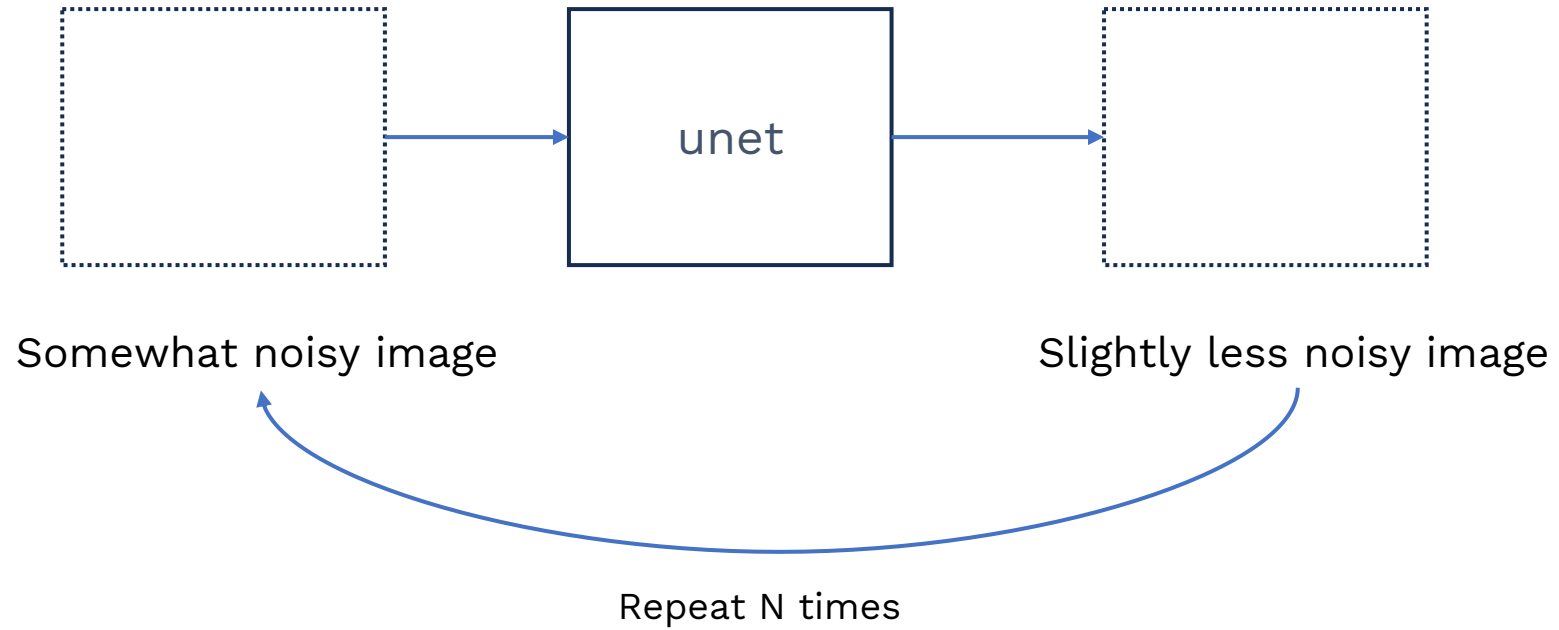
**Linear**



**Cosine**

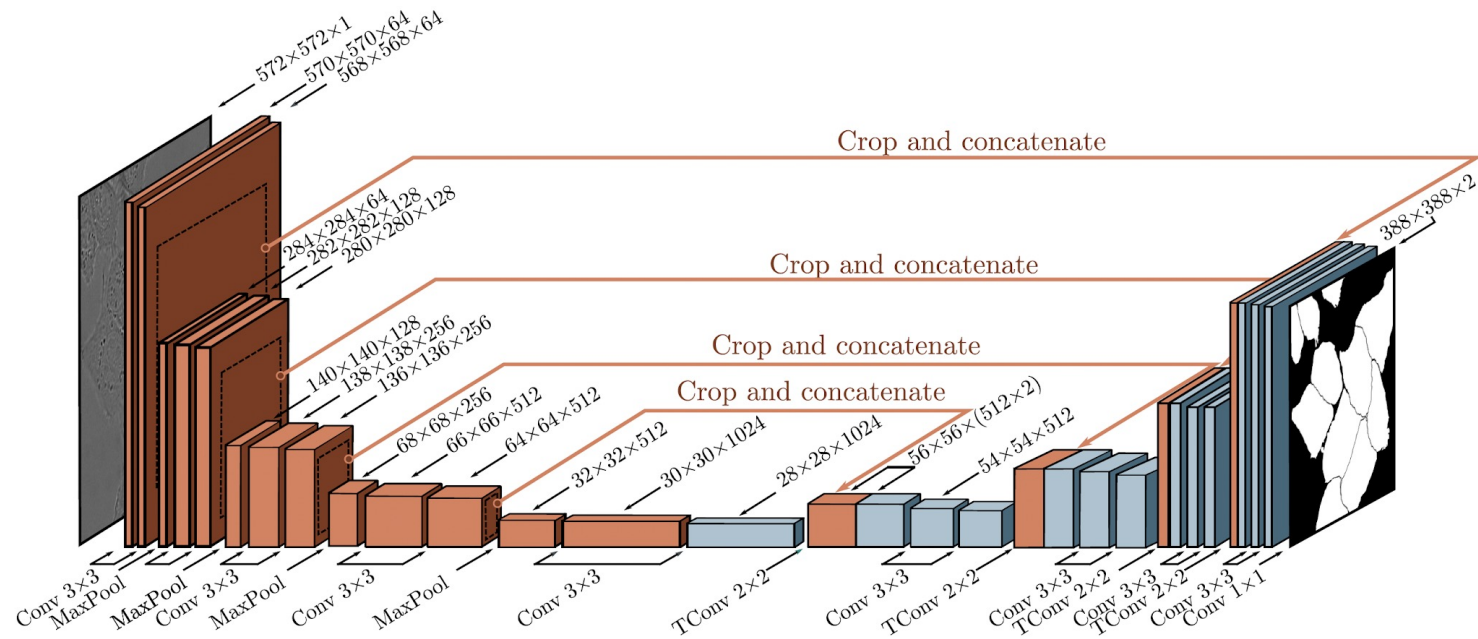
# Stable Diffusion

---





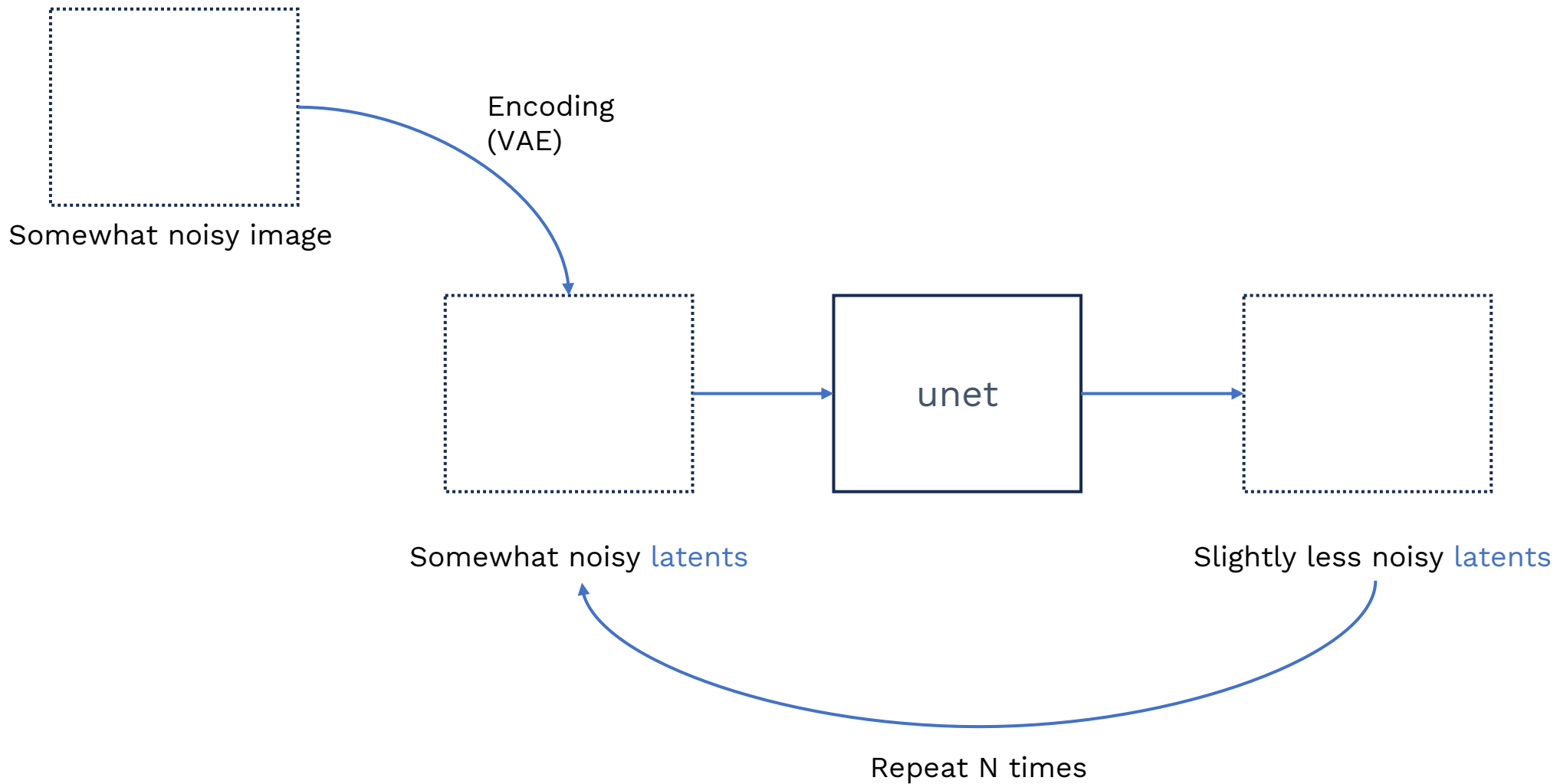
# U-Net



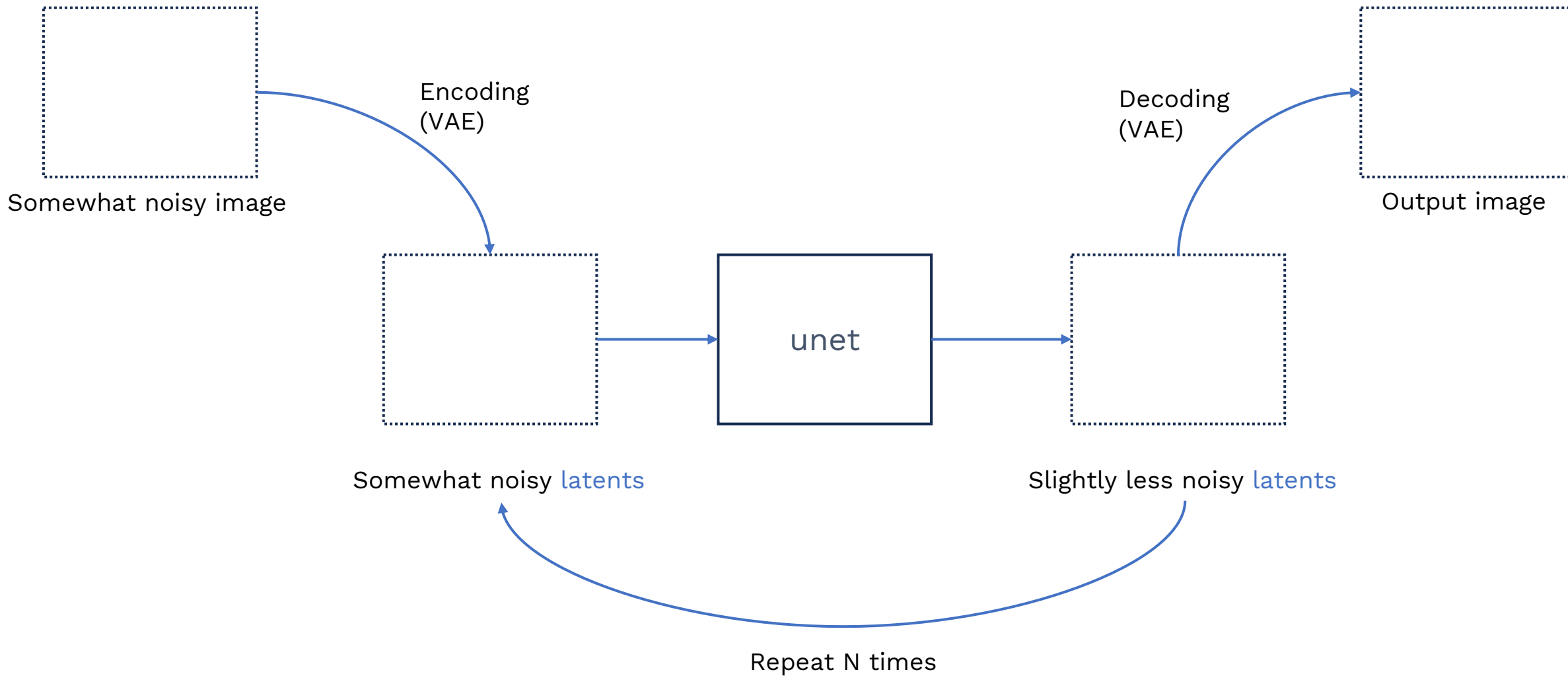
**Figure 11.10** U-Net for segmenting HeLa cells. The U-Net has an encoder-decoder structure, in which the representation is downsampled (orange blocks) and then re-upsampled (blue blocks). The encoder uses regular convolutions, and the decoder uses transposed convolutions. Residual connections append the last representation at each scale in the encoder to the first representation at the same scale in the decoder (orange arrows). The original U-Net used “valid” convolutions, so the size decreased slightly with each layer, even without downsampling. Hence, the representations from the encoder were cropped (dashed squares) before appending to the decoder. Adapted from Ronneberger et al. (2015).



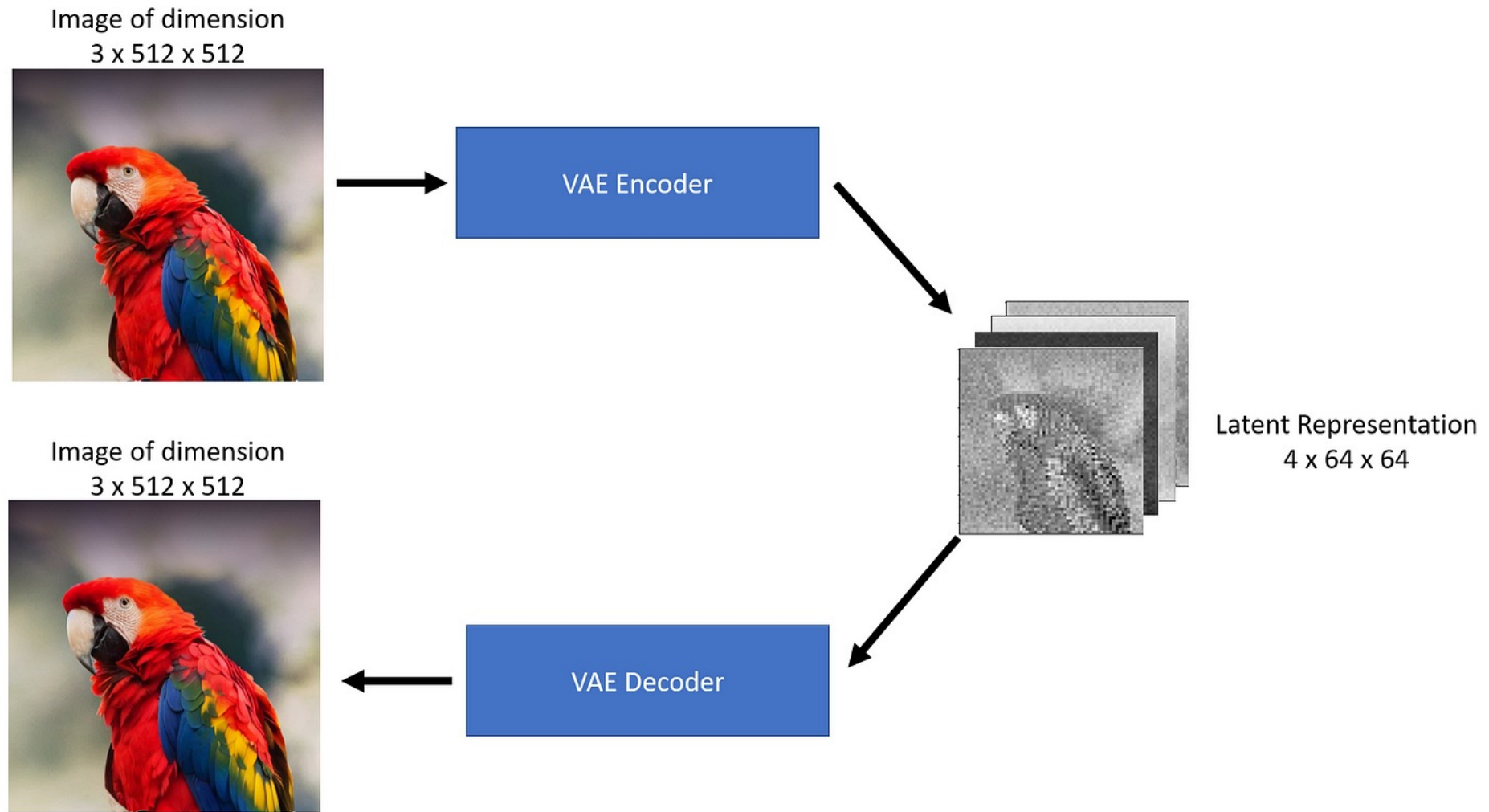
# Stable Diffusion



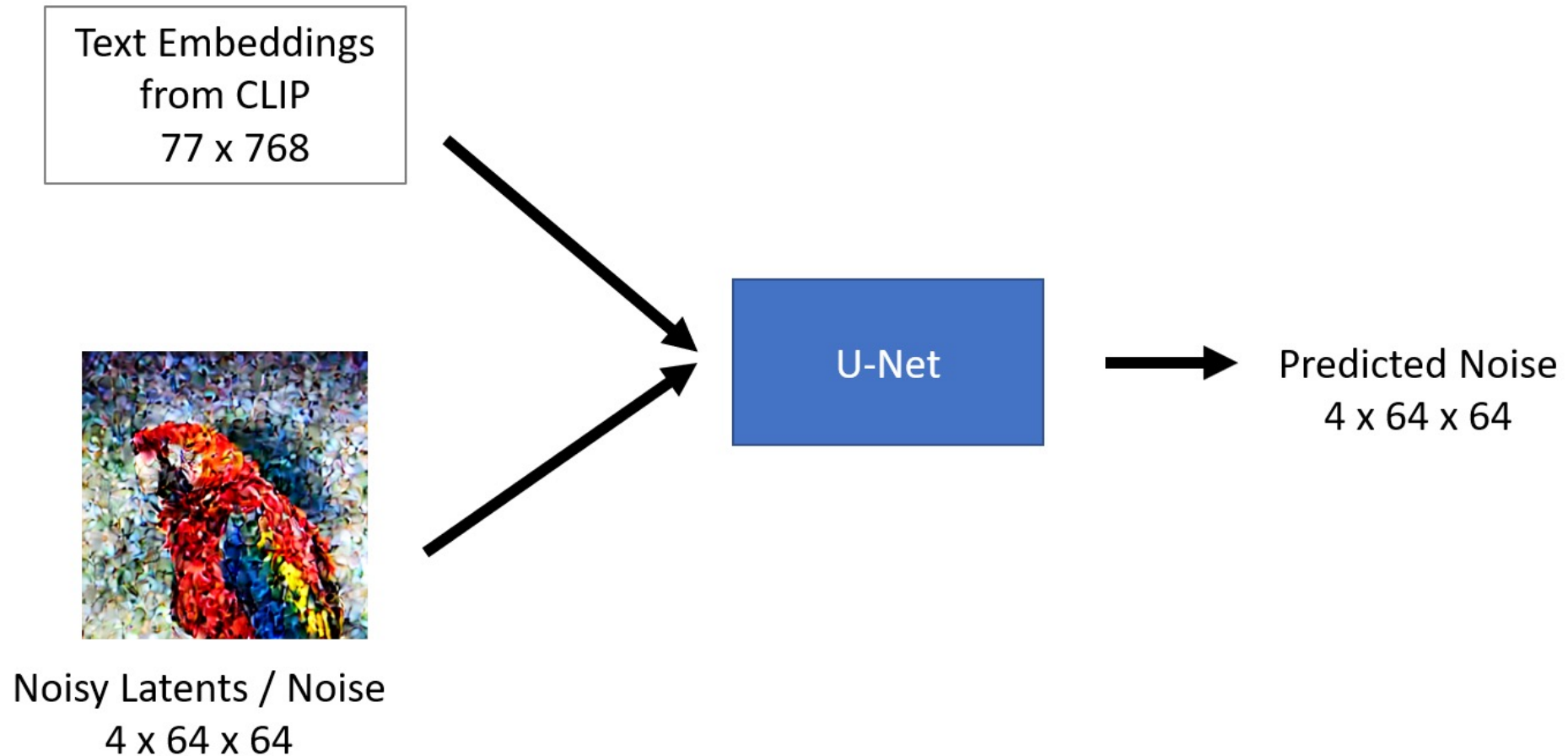
# Stable Diffusion



# VAE Encoder

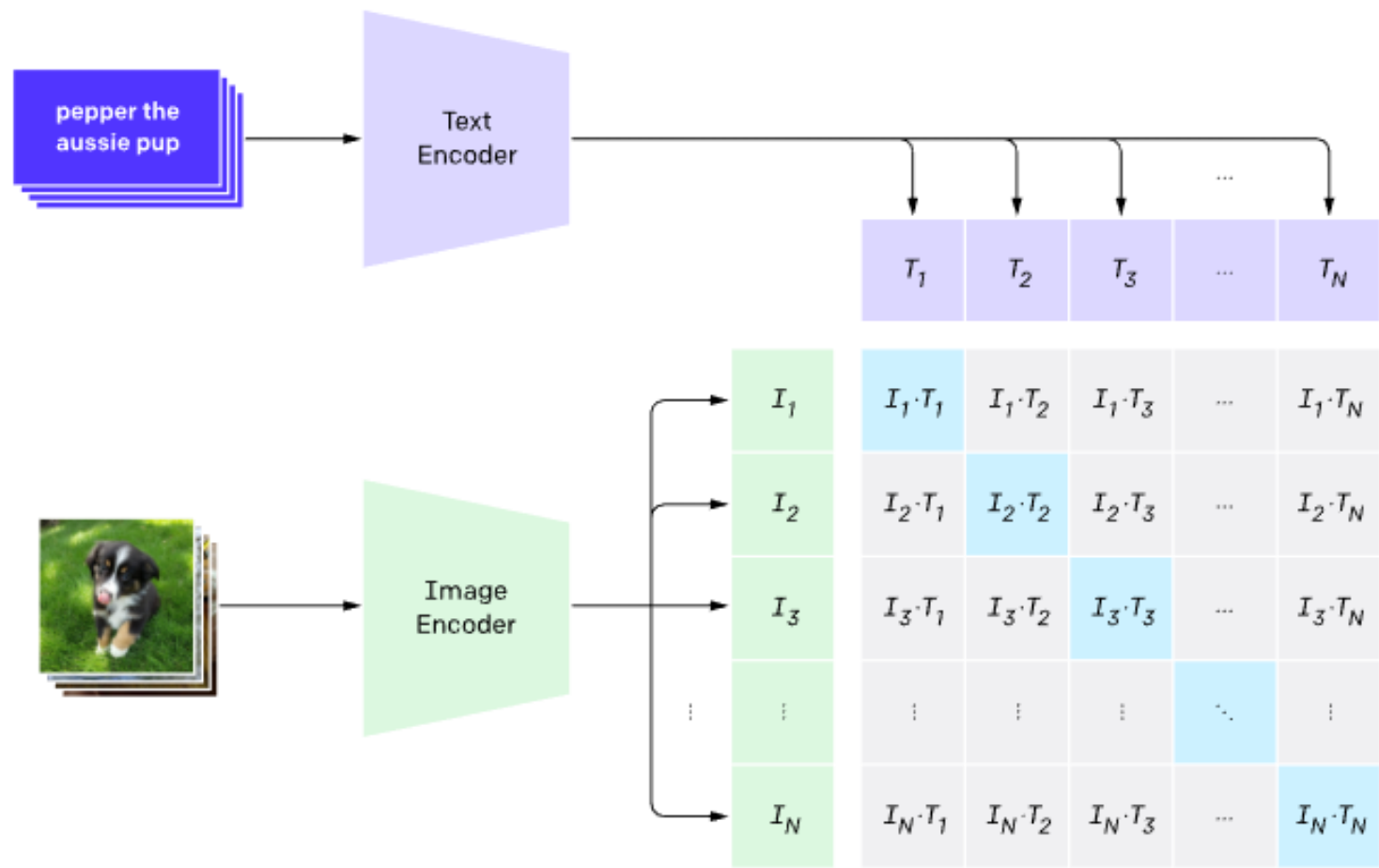


# Guiding the diffusion with CLIP text encoding

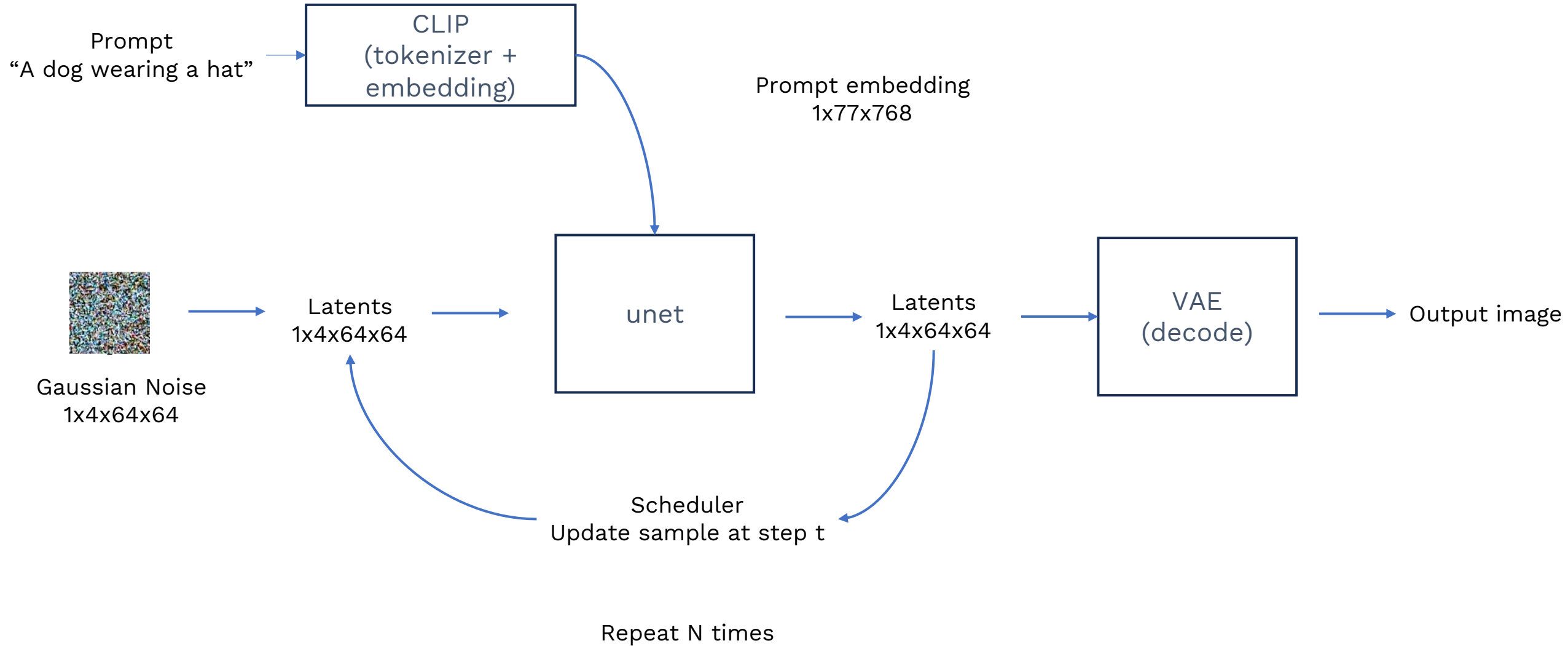


# CLIP (Contrastive Language-Image Pretraining)

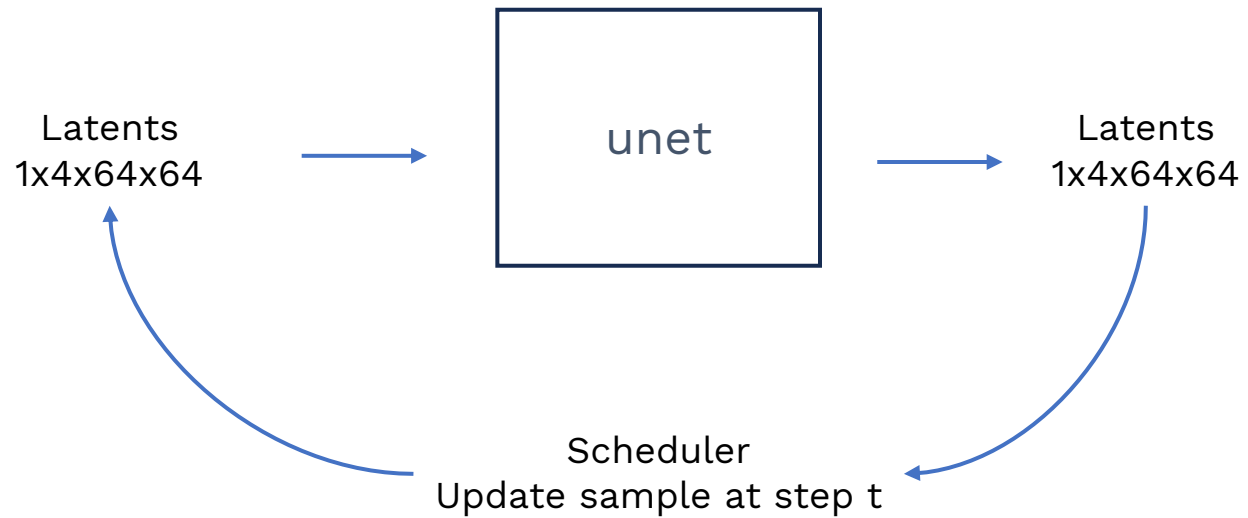
## 1. Contrastive pre-training



# Stable Diffusion - Putting it all together



# Common misconception



The U-Net **does not predict noise between step t-1 and step t**

The U-Net predicts the *entire* noise

The scheduler takes care of removing part of the noise

See [demo](#)

## Stable Diffusion in 15 lines of code

```
tokenizer = CLIPTokenizer.from_pretrained("openai/clip-vit-large-patch14",  
torch_dtype=torch.float16)
```

```
text_encoder = CLIPTextModel.from_pretrained("openai/clip-vit-large-patch14",  
torch_dtype=torch.float16).to("cuda")
```

```
vae = AutoencoderKL.from_pretrained("stabilityai/sd-vae-ft-ema",  
torch_dtype=torch.float16).to("cuda")
```

```
unet = UNet2DConditionModel.from_pretrained("CompVis/stable-diffusion-v1-4",  
subfolder="unet", torch_dtype=torch.float16).to("cuda")
```

```
beta_start, beta_end = 0.00085, 0.012
```

```
scheduler = LMSDiscreteScheduler(beta_start=beta_start, beta_end=beta_end,  
beta_schedule="scaled_linear", num_train_timesteps=1000)
```



## Stable Diffusion in 15 lines of code

```
bs = len(prompts)
text = text_enc(prompts)
uncond = text_enc([""] * bs, text.shape[1])
emb = torch.cat([uncond, text])

latents = torch.randn((bs, unet.in_channels, height//8, width//8))
scheduler.set_timesteps(steps)
latents = latents.to("cuda").half() * scheduler.init_noise_sigma

for i,ts in enumerate(tqdm(scheduler.timesteps)):
    inp = scheduler.scale_model_input(torch.cat([latents] * 2), ts)

    with torch.no_grad():
        u,t = unet(inp, ts, encoder_hidden_states=emb).sample.chunk(2)
        pred = u + g*(t-u)
        latents = scheduler.step(pred, ts, latents).prev_sample

with torch.no_grad():
    return vae.decode(1 / 0.18215 * latents).sample
```

# Going deeper

---

The [maths of diffusion](#) by Lilian Weng

Classifier-free guidance ([CFG](#))

Denoising Diffusion Implicit Models ([DDIM](#))

Diffusion Models for Text Generation ([AR-Diffusion](#))

Diffusion Models in Reinforcement Learning ([DDPO](#))

Applications in Molecular Design ([GaUDI](#))

## Practical 6: let's build a diffusion model from scratch!

---