Variational auto-encoders

Felix Cheysson[1] Tristan Mary-Huard[2]

[1] Université Gustave Eiffel, CNRS, UMR 8050, LAMA.
[2] Université Paris-Saclay, AgroParisTech, INRAE, UMR 518, MIA-Paris-Saclay.

StateoftheR
March 10th 2023

Outline

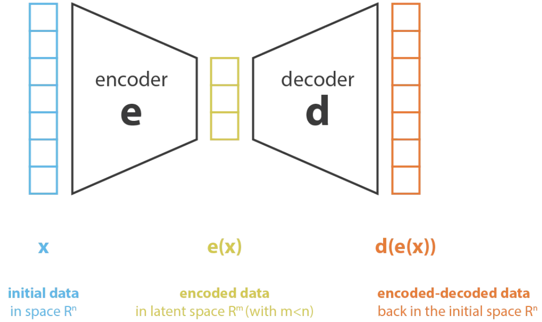
1. Some background
   - Autoencoders
   - The latent variable model and its variational formulation
   - The Evidence (Variational) Lower Bound
2. Variational Auto-Encoders (VAE)
   - The reparameterisation trick
   - Assumptions in our Variational Auto-Encoder
3. Coding time!
   - Setting up the decoder
   - Setting up the encoder
   - Completing the VAE
   - Building the loss function

Dimensionality reduction via autoencoders

An **autoencoder**, encoder-decoder pair (e, d), aims to minimise the **reconstruction error measure** between an input data $x \in \mathcal{X}$ and the encoded-decoded data $d(e(x)) \in \mathcal{X}$:

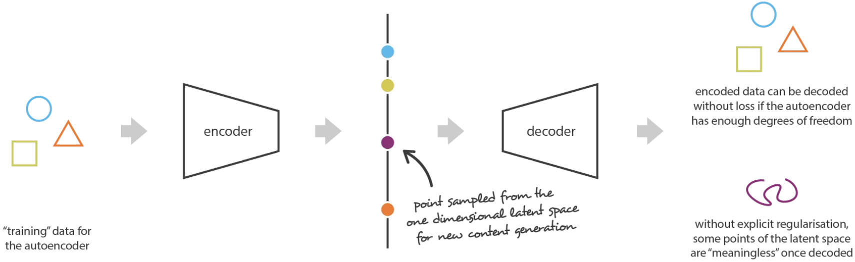
$$(e^*, d^*) = \underset{(e,d)\in E\times D}{\arg\min}\ L\big(x, d(e(x))\big).$$

Example: in PCA, $e = P' \in \mathcal{O}_{m\times n}$ and $d = P$.



Shortcomings of autoencoders

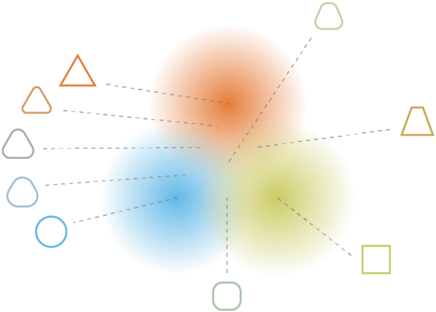
- Denote $z = e(x) \in \mathcal{Z}$ the encoded data, call it a **latent variable**.
- **Problem**: without any constraint on the encoder-decoder pair (e, d) (e.g. neural networks), the autoencoder may lead to gross overfitting.
  - Example: mapping $(x_1, \ldots, x_n)$ to integers $(1, \ldots, n)$ and back.
- This leads to a lack of interpretable and exploitable structure in the latent space $\mathcal{Z}$, without generative purpose.



Goal of variational autoencoders

- **Idea**: Introduce probabilistic model on (x, z) such that the latent space $\mathcal{Z}$ becomes structured.
- Akin to a regularisation during the training process, of the form

$$(e^*, d^*) = \underset{(e,d)\in E\times D}{\arg\min}\ L\big(x, d(e(x))\big) + D_{\mathsf{KL}}\big(p(z) \,\|\, \mathcal{N}(0,1)\big).$$



Context: latent variable model

Objective: We are interested in the joint distribution of a couple (X, Z).

- $X \in \mathbb{R}^n$ is the **input data**, $Z \in \mathbb{R}^m$ are **latent variables**.
- Z usually represents some unobservable (or unobserved) information about X (e.g. in image recognition, X would be the image, Z the correct label).
- Example: the Gaussian mixture model,

$$p(x \mid z) = \mathcal{N}\left(\mu(z), \sigma^2(z)\right)$$
$$p(z) = \text{Categorical}(\pi_1, \dots, \pi_k).$$

- Estimation usually follows from **Expectation-Maximisation** (EM) algorithms.

$$\log p_\theta(x) = \mathbb{E}_{p_\theta(z|x)}[\log p_\theta(x, z)].$$

  - Given θ_n obtained at n^{th} iteration:
  - *E-step*: Compute $\mathbb{E}_{p_{\theta_n}(z|x)}[\log p_\theta(x, z)]$.
  - *M-step*: Find $\theta_{n+1} = \arg\max_\theta \mathbb{E}_{p_{\theta_n}(z|x)}[\log p_\theta(x, z)]$.

Variational formulation

- **Problem**: $p(z \mid x)$ may be intractable.
- **Idea**: approximate $p(z \mid x)$ by $q_\phi(z \mid x)$ (*variational distribution*).

$$
\begin{aligned}
\log p_\theta(x) &= \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x)] \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log\left(\frac{p_\theta(x,z)}{p_\theta(z \mid x)}\right)\right] \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log\left(\frac{p_\theta(x,z)}{q_\phi(z \mid x)}\frac{q_\phi(z \mid x)}{p_\theta(z \mid x)}\right)\right] \\
&= \underbrace{\mathbb{E}_{q_\phi(z|x)}\left[\log\left(\frac{p_\theta(x,z)}{q_\phi(z \mid x)}\right)\right]}_{\substack{=\mathcal{L}_{\theta,\phi}(x) \\ \text{(ELBO)}}} + \underbrace{\mathbb{E}_{q_\phi(z|x)}\left[\log\left(\frac{q_\phi(z \mid x)}{p_\theta(z \mid x)}\right)\right]}_{=D_{\mathrm{KL}}(q_\phi(z|x)\,\|\,p_\theta(z|x))}
\end{aligned}
$$

ELBO: Evidence (Variational) Lower Bound

The variational formulation of the latent variable model consists in maximising the ELBO as a lower bound on $\log p_\theta(x)$,

$$\mathcal{L}_{\theta,\phi}(x) = \log p_\theta(x) - D_{\mathrm{KL}}(q_\phi(z \mid x) \,\|\, p_\theta(z \mid x)).$$

- Optimisation over ϕ will keep ELBO tight around $\log p_\theta(x)$.
- Optimisation over θ will keep pushing the lower bound (and hence $\log p_\theta(x)$) up.

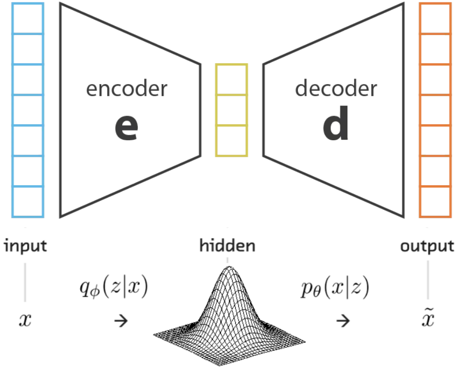
$$\begin{aligned}\mathcal{L}_{\theta,\phi}(x) &= \mathbb{E}_{q_\phi(z|x)}\left[\log\left(\frac{p_\theta(x,z)}{q_\phi(z \mid x)}\right)\right] \\ &= \underbrace{\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x \mid z)]}_{\text{Reconstruction loss}} - \underbrace{D_{\mathsf{KL}}(q_\phi(z \mid x) \,\|\, p(z))}_{\text{Regularisation term}}\end{aligned}$$

Outline

1. Some background
   - Autoencoders
   - The latent variable model and its variational formulation
   - The Evidence (Variational) Lower Bound
2. Variational Auto-Encoders (VAE)
   - The reparameterisation trick
   - Assumptions in our Variational Auto-Encoder
3. Coding time!
   - Setting up the decoder
   - Setting up the encoder
   - Completing the VAE
   - Building the loss function

The Variational Auto-Encoder (VAE)

$$\mathcal{L}_{\theta,\phi}(x) = \underbrace{\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x \mid z)]}_{\substack{\text{Reconstruction loss} \\ \text{Decoder } p_\theta(x|z)}} - \underbrace{D_{\mathsf{KL}}(q_\phi(z \mid x) \,\|\, p(z))}_{\substack{\text{Regularisation term} \\ \text{Encoder } q_\phi(z|x)}}$$



Stochastic gradient-based optimisation of the ELBO

$$\begin{aligned}\mathcal{L}_{\theta,\phi}(x) &= \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x \mid z)] - D_{\mathsf{KL}}(q_\phi(z \mid x) \,\|\, p(z)) \\ &= \mathbb{E}_{q_\phi(z|x)}[f_{\theta,\phi}(x,z)]\end{aligned}$$

Gradient-based optimisation of the ELBO requires partial derivatives with respect to θ and ϕ, given by

$$\begin{aligned}\nabla_\theta \mathcal{L}_{\theta,\phi}(x) &= \nabla_\theta \mathbb{E}_{q_\phi(z|x)}[f_{\theta,\phi}(x,z)] \\ &= \mathbb{E}_{q_\phi(z|x)}[\nabla_\theta f_{\theta,\phi}(x,z)]\end{aligned}$$

but

$$\begin{aligned}\nabla_\phi \mathcal{L}_{\theta,\phi}(x) &= \nabla_\phi \mathbb{E}_{q_\phi(z|x)}[f_{\theta,\phi}(x,z)] \\ &\neq \mathbb{E}_{q_\phi(z|x)}[\nabla_\phi f_{\theta,\phi}(x,z)].\end{aligned}$$

Reparameterisation trick

Instead of $z \sim q_\phi(z \mid x)$, define

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

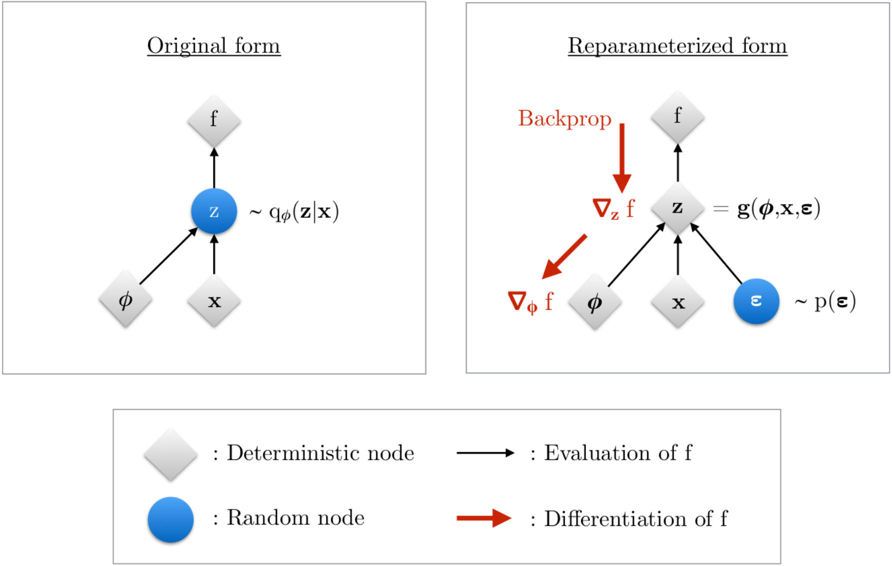
with $\epsilon \sim p(\epsilon)$ independent of x, for example:

- if $q_\phi(z \mid x)$ one-dimensional, $\epsilon \sim \mathcal{U}(0, 1)$ with $g(\epsilon, \phi, x) = F_{q_\phi(z|x)}(\epsilon)$.
- if $q_\phi(z \mid x) = \mathcal{N}(\mu, \mathrm{diag}(\sigma^2))$, $\epsilon \sim \mathcal{N}(0, 1)$ with $g(\epsilon, \phi, x) = \mu + \sigma \odot \epsilon$.

Then,

$$\begin{aligned}
\nabla_\phi \mathcal{L}_{\theta,\phi}(x) &= \nabla_\phi \mathbb{E}_{q_\phi(z|x)}[f_{\theta,\phi}(x, z)] \\
&= \nabla_\phi \mathbb{E}_{p(\epsilon)}[f_{\theta,\phi}(x, z)] \\
&= \mathbb{E}_{p(\epsilon)}[\nabla_\phi f_{\theta,\phi}(x, z)]
\end{aligned}$$

The trick allows us to “backpropagate through z”



Monte Carlo approximation

Both partial derivatives are estimated through Monte Carlo approximation:

- Draw $\epsilon_i \sim p(\epsilon)$ such that $z_i = g(\epsilon_i, \phi, x) \sim q_\phi(z \mid x)$.
- Estimate via Monte Carlo approximation

$$
\begin{aligned}
\nabla_\theta \mathcal{L}_{\theta,\phi}(x) &= \mathbb{E}_{p(\epsilon)}[\nabla_\theta f_{\theta,\phi}(x, z)] \\
&\simeq \frac{1}{n} \sum_i \nabla_\theta f_{\theta,\phi}(x, z_i), \\
\nabla_\phi \mathcal{L}_{\theta,\phi}(x) &= \mathbb{E}_{p(\epsilon)}[\nabla_\phi f_{\theta,\phi}(x, z)] \\
&\simeq \frac{1}{n} \sum_i \nabla_\phi f_{\theta,\phi}(x, z_i).
\end{aligned}
$$

In practice, the Monte Carlo approximation is done with $n = 1$.

Assumptions in our Variational Auto-Encoder

Assume the following choice for $q_\phi(z \mid x)$, $p(z)$ and $p_\theta(x \mid z)$:

$$q_\phi(z \mid x) = \mathcal{N}\big(\mu_\phi(x), \mathrm{diag}(\sigma^2_\phi(x))\big) \qquad \text{(encoder)}$$

$$p(z) = \mathcal{N}\big(0, I\big) \qquad \text{(latent space)}$$

$$p_\theta(x \mid z) = \mathcal{N}\big(\mu_\theta(z), \sigma^2\big). \qquad \text{(decoder)}$$

(Compound Gaussian mixture model)

Then, the ELBO becomes

$$\begin{aligned}
\mathcal{L}_{\theta,\phi}(x) &= \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x \mid z)] - D_{\mathsf{KL}}(q_\phi(z \mid x) \,\|\, p(z)) \\
&\propto \mathbb{E}_{q_\phi(z|x)}\left[-\frac{\|x - \mu_\theta(z)\|_2^2}{2\sigma^2}\right] - \frac{1}{2}\left\|\mu_\phi^2(x) + \sigma_\phi^2(x) - \log \sigma_\phi^2(x)\right\|_1 \\
&\simeq -\frac{\|x - \mu_\theta(z)\|_2^2}{2\sigma^2} - \frac{1}{2}\left\|\mu_\phi^2(x) + \sigma_\phi^2(x) - \log \sigma_\phi^2(x)\right\|_1
\end{aligned}$$

(Monte Carlo approx.)

Possible extensions

Non-exhaustive list of possible improvements to the VAE:

- Improving **the variational bound**: increasing flexibility and accuracy of $q_\phi(z \mid x)$ with improve the tightness of the variational bound (e.g. Inverse Autoregressive Flow, see Kingma and Welling, 2019).
- Improving **encoder and decoder algorithms** which approximate $p_\theta(x \mid z)$ and $q_\phi(z \mid x)$.
- Changing the **structure on the latent space** to better fit the problem at hand.
- Improving **optimisation algorithms** to both accelerate and find better solutions to the problem.

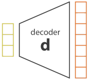
Outline

1. Some background
   - Autoencoders
   - The latent variable model and its variational formulation
   - The Evidence (Variational) Lower Bound
2. Variational Auto-Encoders (VAE)
   - The reparameterisation trick
   - Assumptions in our Variational Auto-Encoder
3. Coding time!
   - Setting up the decoder
   - Setting up the encoder
   - Completing the VAE
   - Building the loss function

Setting up the decoder

- **Model for the decoder**: $p_\theta(x \mid z) = \mathcal{N}(\mu_\theta(z), \sigma^2)$.
- Start with a decoder **generator**:

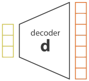
```
> decoder_gen = nn_module(
>   classname = "decoder",
>
>   ## Define the architecture of the decoder
>   initialize = function(latent_dim, input_dim) {
>     self$decompressor = decompressor_gen(latent_dim, input_dim)
>   },
>
>   ## Define the forward method
>   forward = function(input) {
>     input %>% self$decompressor()
>   }
> )
```



Setting up the decoder

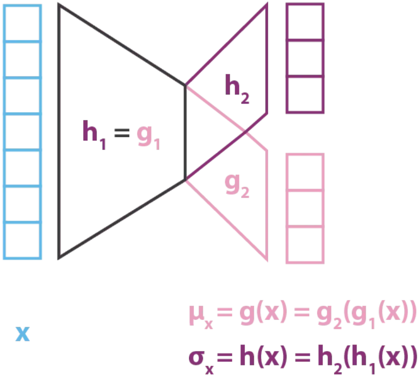
- **Model for the decoder**: $p_\theta(x \mid z) = \mathcal{N}\big(\mu_\theta(z) , \sigma^2\big)$.
- Fill in with your favorite **neural network**:

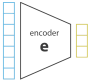
```
> decompressor_gen <- function(latent_dim, input_dim) {
>
>   nn_sequential(
>     nn_linear(latent_dim, 32),
>     nn_relu(),
>     nn_linear(32, 256),
>     nn_relu(),
>     nn_linear(256, input_dim),
>     nn_sigmoid()
>   )
>
> }
```



Setting up the encoder

- **Model for the encoder**: $q_\phi(z \mid x) = \mathcal{N}(\mu_\phi(x), \text{diag}(\sigma^2_\phi(x)))$
- We let $\mu_\phi(x)$ and $\sigma^2_\phi(x)$ share a part of their architecture:

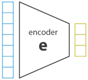




Setting up the encoder

- **Model for the encoder**: $q_\phi(z \mid x) = \mathcal{N}(\mu_\phi(x), \mathrm{diag}(\sigma^2_\phi(x)))$
- Start with an encoder **generator**:

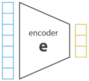
```
> encoder_gen = nn_module(
>   classname = "encoder",
>
>   initialize = function(input_dim, shared_dim, latent_dim) {
>     self$compressor = compressor_gen(          ,              )
>     self$mean = nn_linear(            ,             )
>     self$log_var = nn_linear(            ,             )
>   },
>
>   forward = function(input) {
>     shared_layer = input %>% self$compressor()
>     mean =
>     log_var =
>     list(mean = mean, log_var = log_var)
>   }
> )
```



Setting up the encoder (solution)

- **Model for the encoder**: $q_\phi(z \mid x) = \mathcal{N}(\mu_\phi(x), \mathrm{diag}(\sigma^2_\phi(x)))$
- Start with an encoder **generator**:

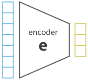
```
> encoder_gen = nn_module(
>   classname = "encoder",
>
>   initialize = function(input_dim, shared_dim, latent_dim) {
>     self$compressor = compressor_gen(input_dim, shared_dim)
>     self$mean = nn_linear(shared_dim, latent_dim)
>     self$log_var = nn_linear(shared_dim, latent_dim)
>   },
>
>   forward = function(input) {
>     shared_layer = input %>% self$compressor()
>     mean = shared_layer %>% self$mean()
>     log_var = shared_layer %>% self$log_var()
>     list(mean = mean, log_var = log_var)
>   }
> )
```



Setting up the encoder

- **Model for the encoder**: $q_\phi(z \mid x) = \mathcal{N}(\mu_\phi(x), \mathrm{diag}(\sigma^2_\phi(x)))$
- Fill in with your favorite **neural network**:

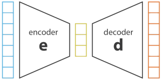
```
> compressor_gen = function(input_dim, shared_dim) {
>
>   nn_sequential(
>     nn_linear(input_dim, 256),
>     nn_relu(),
>     nn_linear(256, shared_dim),
>     nn_relu()
>   )
>
> }
```



Completing the VAE

- Start with a VAE **generator**:

```
> vae_gen = nn_module(
>   classname = "vae",
>
>   initialize = function(input_dim, shared_dim, latent_dim) {
>     self$latent_dim = latent_dim
>     self$encoder = encoder_gen(input_dim, shared_dim, latent_dim)
>     self$decoder = decoder_gen(latent_dim, input_dim)
>   },
>
>   forward = function(input) {
>     [...]
>     return list(output = output, z = z,
>                 mean = mean, log_var = log_var)
>   }
> )
```

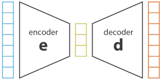


Completing the VAE

Monte Carlo approximation and reparameterisation trick

Draw $\epsilon \sim \mathcal{N}(0, 1)$ and set $z = \mu_\phi(x) + \sigma_\phi(x) \odot \epsilon \sim q_\phi(z \mid x)$.

```
>   forward = function(input) {
>     ## Compressing data
>     latent = self$encoder(input)
>     mean =
>     log_var =
>
>     ## Sampling in latent space (Monte Carlo approx.)
>     z =
>
>
>     ## Decompressing latent representation
>     output =
>
>     return(list(output = output, z = z,
>                 mean = mean, log_var = log_var))
>   }
```

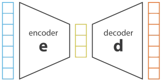


Completing the VAE (solution)

Monte Carlo approximation and reparameterisation trick

Draw $\epsilon \sim \mathcal{N}(0, 1)$ and set $z = \mu_\phi(x) + \sigma_\phi(x) \odot \epsilon \sim q_\phi(z \mid x)$.

```
>    forward = function(input) {
>      ## Compressing data
>      latent = self$encoder(input)
>      mean = latent$mean
>      log_var = latent$log_var
>
>      ## Sampling in latent space (Monte Carlo approx.)
>      z = mean + torch_exp(log_var$mul(0.5))
>                 * torch_randn(c(dim(input)[1], self$latent_dim))
>
>      ## Decompressing latent representation
>      output = self$decoder(z)
>
>      return(list(output = output, z = z,
>                  mean = mean, log_var = log_var))
>    }
```

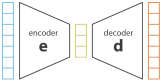


The ELBO function

Evidence (Variational) Lower Bound (ELBO)

$$-\mathcal{L}_{\theta,\phi}(x) \simeq \underbrace{\frac{\|x - \mu_\theta(z)\|_2^2}{\sigma^2}}_{\text{Reconstruction loss}} + \underbrace{\left\| \mu_\phi^2(x) + \sigma_\phi^2(x) - \log \sigma_\phi^2(x) \right\|_1}_{\text{Regularisation term}}$$

```
> loss_fn = function(prediction, target, mean, log_var, kl_weight) {
>
>   l2 = nn_mse_loss(reduction = "sum")
>   l2_eval =
>
>   kl_div =
>   kl_div = kl_div$sum()
>
>   return(l2_eval + kl_weight * kl_div)
>
> }
```

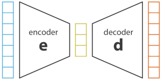


The ELBO function (solution)

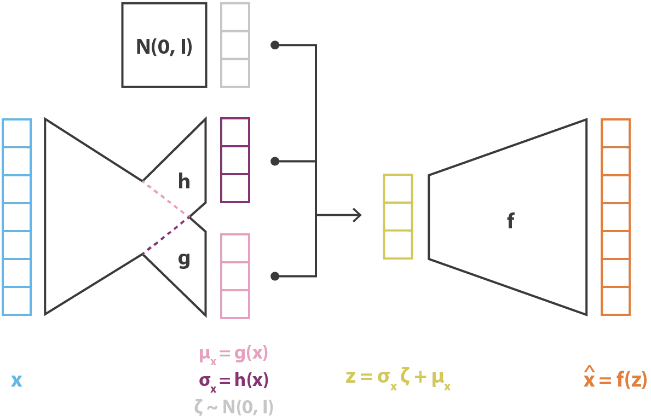
Evidence (Variational) Lower Bound (ELBO)

$$-\mathcal{L}_{\theta,\phi}(x) \simeq \underbrace{\frac{\|x-\mu_\theta(z)\|_2^2}{\sigma^2}}_{\text{Reconstruction loss}} + \underbrace{\left\|\mu_\phi^2(x)+\sigma_\phi^2(x)-\log\sigma_\phi^2(x)\right\|_1}_{\text{Regularisation term}}$$

```
> loss_fn = function(prediction, target, mean, log_var, kl_weight) {
>
>   l2 = nn_mse_loss(reduction = "sum")
>   l2_eval = l2(prediction, target)
>
>   kl_div = mean$square() + log_var$exp() - log_var
>   kl_div = kl_div$sum()
>
>   return(l2_eval + kl_weight * kl_div)
>
> }
```

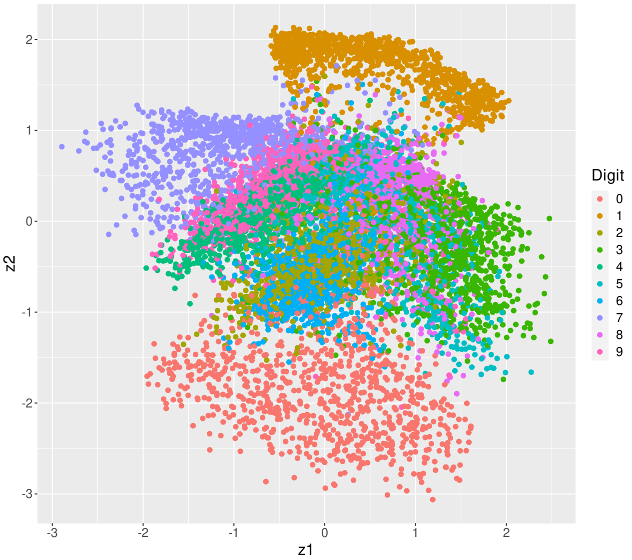


The full VAE architecture in a nutshell

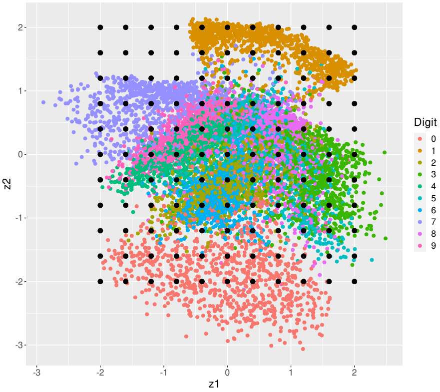


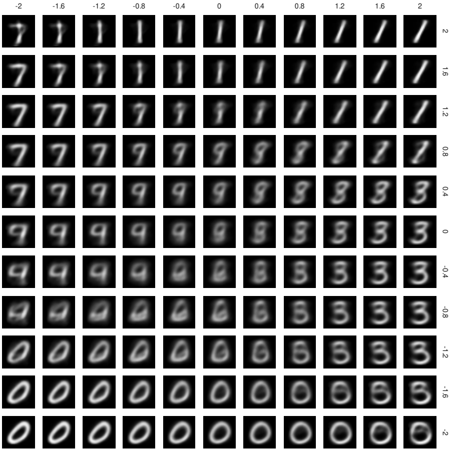
$$\text{loss} = C \,\| x - \hat{x} \|^2 + KL[\, N(\mu_x, \sigma_x), N(0, I) \,] = C \,\| x - f(z) \|^2 + KL[\, N(g(x), h(x)), N(0, I) \,]$$

Representation of the latent space for MNIST



Sampling the latent space for MNIST





For Further Reading I

- Aubert, Julie and Sophie Donnet (2021). *A gentle introduction to the Variational Neural Networks*. https://stateofther.netlify.app/post/intro-variational-autoencoder/.
- Gupta, Rishabh (2017). *Variational Auto Encoders*.
- Kingma, Diederik P. and Max Welling (2019). "An introduction to variational autoencoders". In: *Foundations and Trends in Machine Learning* 12.4, pp. 307–392. DOI: 10.1561/2200000056.
- Kuleshov, Volodymyr and Stefano Ermon (2023). *The variational auto-encoder*. https://ermongroup.github.io/cs228-notes/extras/vae/.
- Rocca, Joseph (2019). *Understanding Variational Autoencoders (VAEs)*. https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73.