Learning Probabilities and Causality

Lab on Variational Autoencoders

Xavier Alameda-Pineda

Defining your VAE

You will be using a VAE to represent speech data, more precisely in the form of a short-time Fourier transform.

$\Rightarrow$ Every observation $\boldsymbol{x}$ will be an F-dimensional **complex** vector: $\boldsymbol{x} \in \mathbb{C}^F$.
The low-dimensional latent variable will be **real** of dimension D: $\boldsymbol{z} \in \mathbb{R}^D$.

The generative model:

- $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$,
- $\boldsymbol{x}|\boldsymbol{z} \sim \mathcal{N}_c(\boldsymbol{x}; \mathbf{0}, \boldsymbol{\Sigma}_{\boldsymbol{\Theta}}(\boldsymbol{z}))$, (see in 2 slides)

where $\boldsymbol{\Sigma}_{\boldsymbol{\Theta}}(\boldsymbol{z})$ will be implemented with a **deep neural network** parametrised by $\boldsymbol{\Theta}$ with input $\boldsymbol{z}$.

Recalling the log-variance

How can we ensure that $\boldsymbol{\Sigma}_{\boldsymbol{\Theta}}(\boldsymbol{z})$ is a covariance matrix?

- The covariance matrix is assumed to be diagonal:

$$\boldsymbol{\Sigma}_{\boldsymbol{\Theta}}(\boldsymbol{z}) = \begin{pmatrix} \nu_{\boldsymbol{\Theta}}^{(1)}(\boldsymbol{z}) & 0 & \cdots & 0 \\ 0 & \nu_{\boldsymbol{\Theta}}^{(2)}(\boldsymbol{z}) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \nu_{\boldsymbol{\Theta}}^{(F)}(\boldsymbol{z}) \end{pmatrix} \tag{1}$$

  Reduces complexity and memory, but also expressivity.

- We estimate the log-variance: $\eta_{\boldsymbol{\Theta}}^{(f)}(\boldsymbol{z}) = \log \nu_{\boldsymbol{\Theta}}^{(f)}(\boldsymbol{z})$:

$$\boldsymbol{\Sigma}_{\boldsymbol{\Theta}}(\boldsymbol{z}) = \mathrm{diag}_{\boldsymbol{d}} \left(\exp \left(\eta_{\boldsymbol{\Theta}}^{(f)}(\boldsymbol{z}) \right) \right) \tag{2}$$

  The values of $\eta_{\boldsymbol{\Theta}}^{(f)}(\boldsymbol{z})$ can be positive or negative.

Complex Gaussian distribution – Reconstruction Loss

We will only need the 1D complex Gaussian distribution:

$$\mathcal{N}_c(x^{(f)}; 0, \nu_{\boldsymbol{\Theta}}^{(f)}(\boldsymbol{z})) = \frac{1}{\pi \nu_{\boldsymbol{\Theta}}^{(f)}(\boldsymbol{z})} \exp\left(-\frac{|x^{(f)}|^2}{\nu_{\boldsymbol{\Theta}}^{(f)}(\boldsymbol{z})}\right). \tag{3}$$

Very close to the real Gaussian distribution but missing some "$\frac{1}{2}$."

We will only use the modulus of the speech vector representation.

And recall the output of the decoder network produces the log-variance: $\nu_{\boldsymbol{\Theta}}^{(f)}(\boldsymbol{z}) = \exp\left(\eta_{\boldsymbol{\Theta}}^{(f)}(\boldsymbol{z})\right)$.

[In function `get_loss`, implement `loss_recon`]

Posterior distribution – Regularisation Loss

The exact posterior is not analytic, we approximate with **another** feed-forward network parametrised with $\mathbf{\Phi}$:

$$p(\boldsymbol{z}|\boldsymbol{x}) \approx q(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}; \tilde{\boldsymbol{\mu}}_{\mathbf{\Phi}}(\boldsymbol{x}), \tilde{\mathbf{\Sigma}}_{\mathbf{\Phi}}(\boldsymbol{x})) \tag{4}$$

Recall that we will estimate the log-variances of a diagonal covariance matrix.

The regularisation terms is based on the Kullback-Leibler divergence, which for two D-dimensional real Gaussian distributions writes:

$$\mathcal{D}_{\mathrm{KL}}(\mathcal{N}_0 \| \mathcal{N}_1) = \frac{1}{2}\left(\mathrm{Tr}(\mathbf{\Sigma}_1^{-1}\mathbf{\Sigma}_0) - D + (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)^\top \mathbf{\Sigma}_1^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0) + \log \frac{|\mathbf{\Sigma}_1|}{|\mathbf{\Sigma}_0|}\right)$$

[In function `get_loss`, implement `loss_KLD`]

The encoder and decoder networks

The decoder network consists of **five** layers:

- The first four have output dimensions 32, 64, 128 and 256, respectively, and each of them is followed up with a hyperbolic tangent activation. They are implemented in the `mlp_z_x`.
- The fifth layer has dimension F, and has no activation layer. It is implemented in `gen_logvar`.

The overall generative process uses these two subnetworks in `generation_x` (be sure to use torch operators for functions).

The encoder network has a symmetric design:

- Four layers with output dim 256, 128, 64, and 32 activated with `tanh`. Implemented in `mlp_x_z`.
- Two extra layer with no activation to output **both** mean and the log-variance of z. Implemented in `z_mean` and `z_logvar` (both D).

The overall inference process uses these two subnetworks in `inference`.

Finishing the implementation

The only thing missing is to implement the reparametrisation trick that we've seen in class.

The forward and train are already implemented, and you will only need to run experiments/plot results.

The skeleton will be provided, together with a link to the data, and the pre-trained model (used for some experiments).

Reports due before vacation break, that is on **December 22nd** (Xmas lotery day in Spain) at latest.