Halvtid presentation

ULR

HIS

March 7, 2023

Contents

Autoencoder

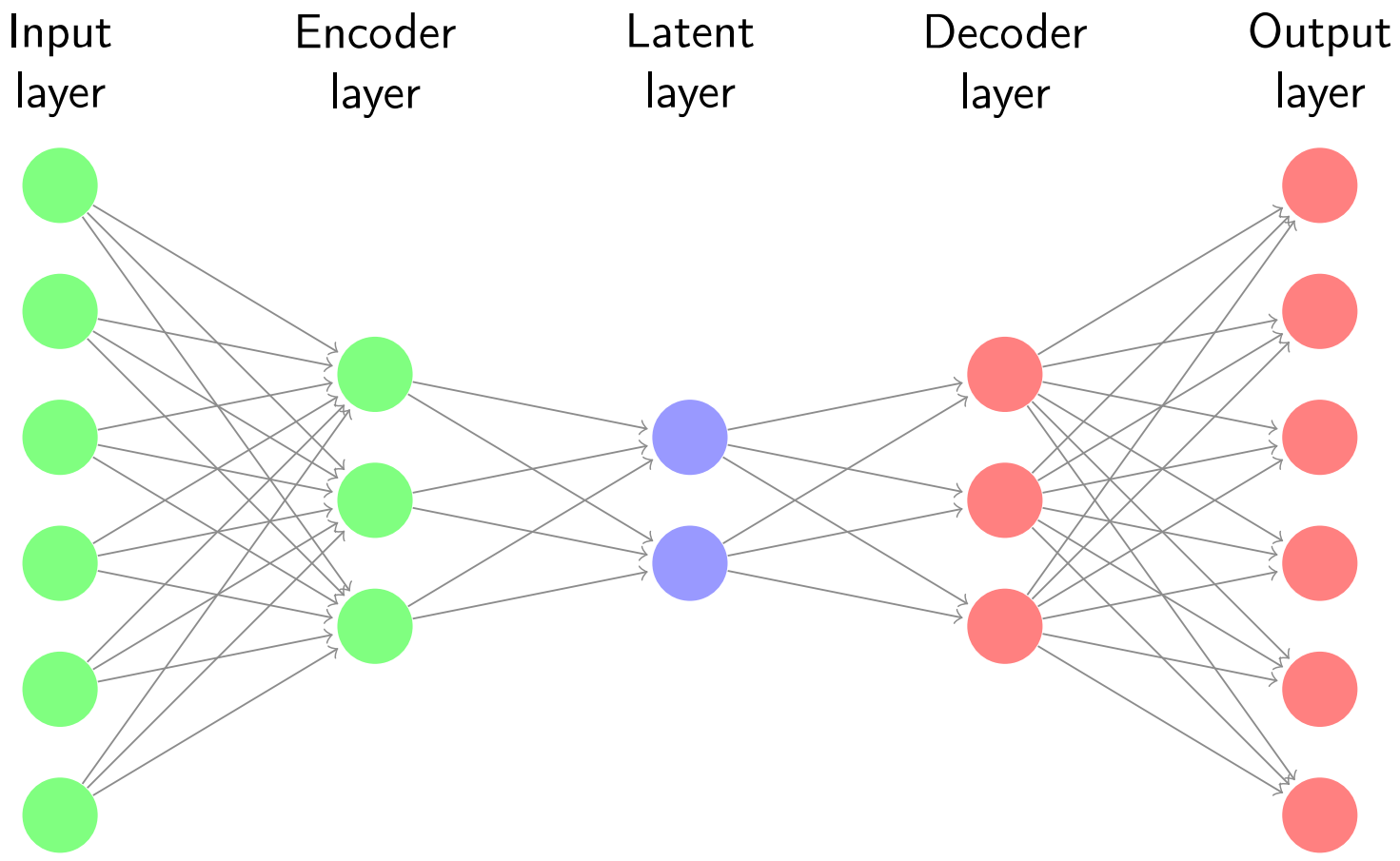
Basics

- "been part of the historical landscape of of neural networks for decades" [3]
- Hidden layers h reassemble a function $f(X)$ which transform their input X to a encoded representation z so that another reconstruction function $g(z)$ is able to produce $g(z) \approx X$.

Usage

- Denoising
- Dimensionality reduction
- Feature learning
- Generative learning

Autoencoder

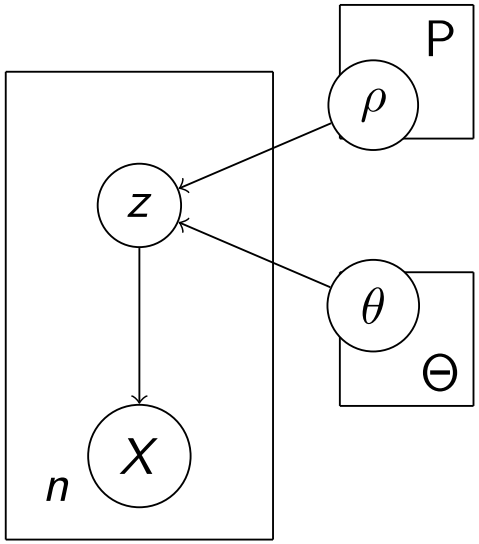


Variational Autoencoder

Method

Traditionally unsupervised variational Bayesian method

Sampling process



- Θ Model parameter space
- ρ Probability distribution $P(z)$
- z Latent variable sampled n times

Variational Autoencoder

Bayes theorem

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)} \tag{1}$$

Objective

- With every example X from a set $\mathcal{X}$ we aim to generate a space z to recreate X.
- z gets sampled from $\mathcal{Z}$ according to $\rho \sim P(z)$.
- $f(z;\theta), f : \mathcal{Z} \times \Theta \to \mathcal{X}$ helps to optimize θ
- maximize:

$$P(x) = \int P(X \mid z;\theta)\, P(z)\, dz \tag{2}$$

- Where $f(z;\theta)$ gets replaced by $P(X \mid z;\theta)$ to enable maximum likelihood framework

Variational Autoencoder

Objective

Based on some input sample values of z likely to produce X.

Method

Therefor a function $Q(z \mid X)$ is introduced to generate z's likely to produce X.
This space of z's should be smaller than that one under the prior $P(X)$

Variational Autoencoder

Optimisation

The framework introduced by Kingma and Welling [4] does the magic. It allows us to construct a differentiable estimator:

$$\int Q(z \mid X)\, f(z)\, dz \tag{3}$$

Kullback Leibler Divergence

Allows to compare probability distributions

P, Q probability distribution functions

$$\mathcal{D}(P \parallel Q) = \mathcal{KL} = \sum_{x \in X} P(x) \cdot \log \frac{P(x)}{Q(x)} \tag{4}$$

Helpers

Marginal likelihood of individual data points

(Likelihood function integrated over parameter space)

$$\log P\left(X^i\right) = \mathcal{D}\left(Q_\phi\left(z \mid X^i\right) \parallel P_\theta\left(z \mid X^i\right)\right) + \mathcal{L}\left(\theta, \phi, X^i\right) \tag{5}$$

Where:

$$\mathcal{L}(\theta, \phi, X) = \mathcal{E}_{z \sim Q}\left[\log P_\theta(z \mid X) - \log Q_\phi(z)\right] \tag{6}$$

Evidence lower bound

$$\begin{aligned} \log P_\theta\left(X^i\right) &\geq \mathcal{L}\left(\theta, \phi, X^i\right) \\ &= \mathcal{E}_{Q_\phi(z|X)}\left[-\log Q_\phi(z \mid X) + \log P_\theta(z, X)\right] \end{aligned} \tag{7}$$

Relationship between Q and P

The Evidence lower bound (ELBO) or just lower variational bound:

$$\mathcal{D}\left(Q(z) \parallel P(z \mid X)\right) = \mathcal{E}_{z\sim Q}\left[\log Q(z) - \log P(z \mid X)\right] \tag{8}$$

Introduce $P(x)$ and $P(X \mid z)$:

$$\begin{aligned} &\mathcal{D}\left(Q(z) \parallel P(z \mid X)\right) \\ &\quad = \mathcal{E}_{z\sim Q}\left[\log Q(z) - \log P(z \mid X) - \log P(z)\right] + \log P(X) \end{aligned} \tag{9}$$

Rearrange the ELBO and get another $\mathcal{KL}$-term:

$$\begin{aligned} \log P(X) - &\mathcal{D}\left(Q(z) \parallel P(z \mid X)\right) \\ &= \mathcal{E}_{z\sim Q}\left[\log P(X \mid z)\right] - \mathcal{D}\left(Q(z) \parallel P(z)\right) \end{aligned} \tag{10}$$

And introduce a dependency of $Q(z)$ on X to make it useful:

$$\begin{aligned} \log P(X) - &\mathcal{D}\left(Q(z \mid X) \parallel P(z \mid X)\right) \\ &= \mathcal{E}_{z\sim Q}\left[\log P(X \mid z)\right] - \mathcal{D}\left(Q(z \mid X) \parallel P(z)\right) \end{aligned} \tag{11}$$

Autoencoding variational Bayes

Backpropagation only works with continuous operations, stochastic units aren't that.

The reparametrisation trick

With $\mu(X)$ and $\sigma(x)$ parametrising $Q(z \mid X)$ sampling from $\mathcal{N}(\mu(X), \sigma(X))$ we can draw $\epsilon \sim \mathcal{N}(0,1)$ and compute $z_{\epsilon \sim Q} = \mu(X) + \sqrt{\sigma}(X) \cdot \epsilon$ and perform SGD

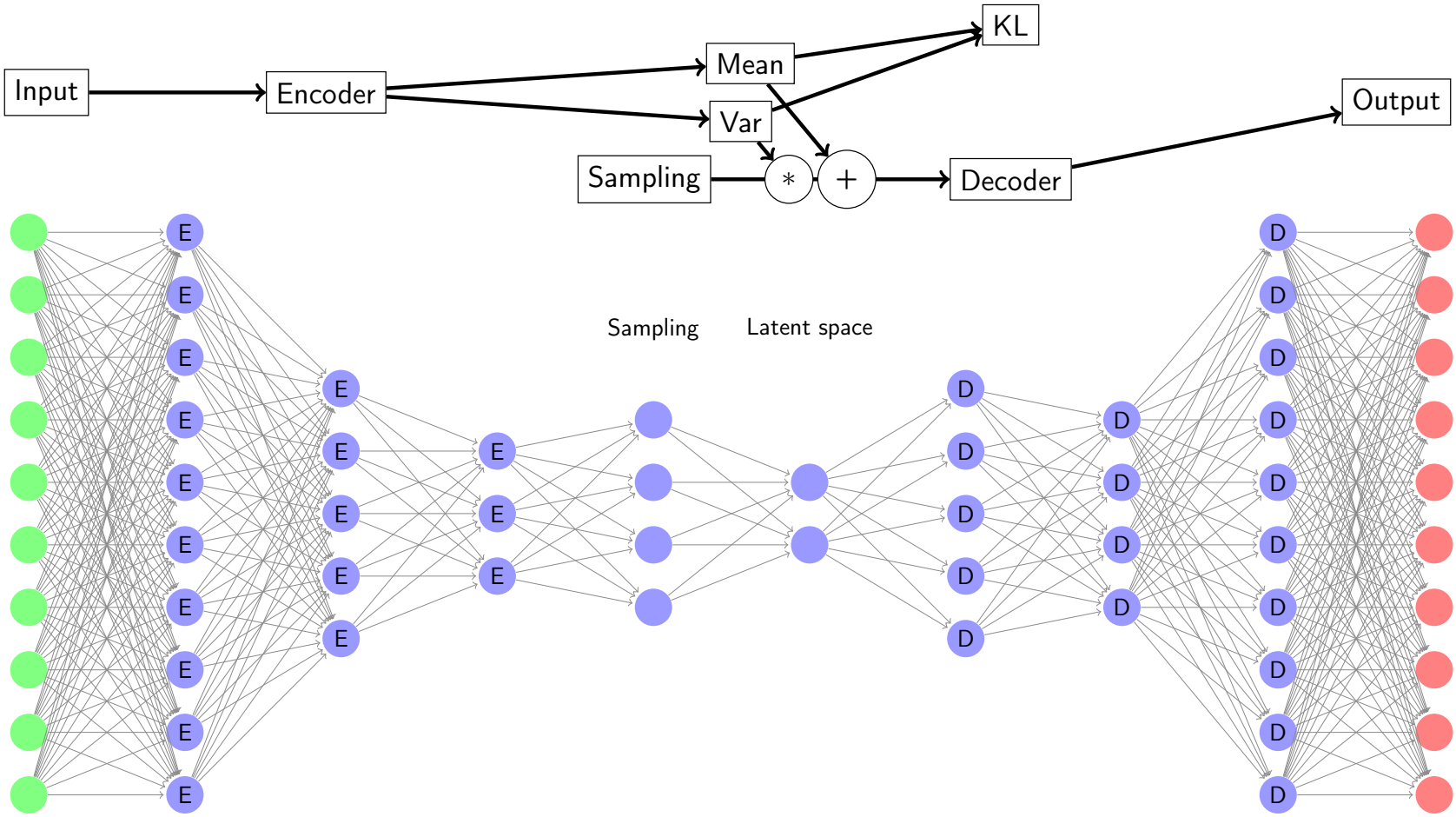
$$\mathcal{E}_{z \sim \mathcal{D}} \left[\mathcal{E}_{\epsilon \sim \mathcal{N}(0,1)} \left[\log P(X \mid z_{\epsilon \sim Q} - \mathcal{D} \left(Q(z \mid X) \parallel P(z) \right) \right] \right] \tag{12}$$

$$\begin{aligned} \mathcal{L}(\theta, \phi, X) &= \mathcal{E}_{z \sim Q} \left[\log P_\theta(z \mid X) - \log Q_\phi(z) \right] \\ &\approx \tilde{\mathcal{L}}^m(\theta, \phi, X^m) = \frac{n}{m} \sum_{i=1}^{m} \mathcal{L}(\theta, \phi, X_i) \end{aligned} \tag{13}$$

Pseudocode

$\theta, \phi \leftarrow$ random parameter
repeat
    $X^m \leftarrow$ random mini batch with size m
    $\epsilon \leftarrow$ random samples from noise distribution $P(\epsilon)$
    $g \leftarrow \nabla_{\theta,\phi} \tilde{\mathcal{L}}^m(\theta, \phi, X^m, \epsilon)$
    $\theta, \phi \leftarrow$ update parameters with g
until convergence of params θ, ϕ
return θ, ϕ

Variational Autoencoder



Implementation

- Using `tensorflow.keras` functional API [2]
- Currently *relu* activations and unbiased layers
- Hidden layers constructed iterative
- Deploying a `._build` method, returns encoder, decoder and z
- `Model` gets trained in customised step
- Sampling layer forms an extra `Class`

Training step

```
def train_step(self, data):
   with tf.GradientTape() as tape:
       z_mean, z_logvar, z = self.encoder(data)
       reconstruction = self.decoder(z)
       reconstruction_loss = mean(K.square(data - reconstruction),
                                  axis=1)
       kl_loss = -.5 * (1 + z_logvar - square(z_mean) - exp(
           z_logvar)
                         )
       kl_loss = sum(kl_loss)
       total_loss = kl_loss + reconstruction_loss
   grads = tape.gradient(total_loss, self.trainable_weights)
   self.optimizer.apply_gradients(zip(grads, self.trainable_weights))
   self.total_loss_tracker.update_state(total_loss)
   self.reconstruction_loss_tracker.update_state(reconstruction_loss)
   self.kl_loss_tracker.update_state(kl_loss)
   return {
           "loss": self.total_loss_tracker.result(),
           "reconstruction_loss": self.reconstruction_loss_tracker.result(),
           "kl_loss": self.kl_loss_tracker.result()}
```

Sampling layer

```
class Sampling_Layer(tf.keras.layers.Layer):
    """
    A sampling layer using the mean and log(var) to sample from an input
    """
    def call(self, inputs):
        mean, logvar = inputs
        batch = tf.shape(mean)[0]
        dim = tf.shape(mean)[1]
        epsilon = tf.random.normal(shape=(batch, dim), mean=0., stddev=1.)
        return mean + tf.exp(logvar * .5) * epsilon
```

Training with a dummy

```
import tensorflow as tf
import numpy as np
import generic_VAE

# create some random, uniformly distributed tensors:
training_dummys = np.asarray(
        [tf.random.uniform((400,)) for item in range(100000)])

# create some testing tensors:
testing_dummys = np.asarray(
        [tf.random.uniform((400,)) for item in range(50000)])

# this is completely senseless, the part where a training-set and one for
# testing was created the VAE is unsupervised (in this case)
dummy_data = np.concatenate((training_dummys, testing_dummys))

dummy_vae = generic_VAE.Builder(
        input_shape=(400, ),
        encoder_shape=[400, 200, 100, 40, 20, 4],
        decoder_shape=[4, 20, 40, 100, 200, 400],
        latent_dims=2,
        dropout_rate=0)

vae = generic_VAE.VAE(dummy_vae.decoder_model, dummy_vae.encoder_model)
vae.compile()
vae.fit(dummy_data, epochs=10, batch_size=2000)
```

Training with a dummy

```
Epoch 1/10
75/75 [==============================] - 4s 32ms/step - loss: 3.6552 - reconstruction_loss: 0.1831 - kl_loss: 3.4721
Epoch 2/10
75/75 [==============================] - 3s 34ms/step - loss: 0.1436 - reconstruction_loss: 0.1432 - kl_loss: 4.0093e-04
Epoch 3/10
75/75 [==============================] - 3s 38ms/step - loss: 0.1430 - reconstruction_loss: 0.1422 - kl_loss: 7.2372e-04
Epoch 4/10
75/75 [==============================] - 3s 33ms/step - loss: 0.1417 - reconstruction_loss: 0.1410 - kl_loss: 7.1526e-04
Epoch 5/10
75/75 [==============================] - 3s 34ms/step - loss: 0.1405 - reconstruction_loss: 0.1398 - kl_loss: 7.1526e-04
Epoch 6/10
75/75 [==============================] - 3s 35ms/step - loss: 0.1403 - reconstruction_loss: 0.1396 - kl_loss: 7.1526e-04
Epoch 7/10
75/75 [==============================] - 3s 42ms/step - loss: 0.1397 - reconstruction_loss: 0.1389 - kl_loss: 7.1526e-04
Epoch 8/10
75/75 [==============================] - 3s 36ms/step - loss: 0.1395 - reconstruction_loss: 0.1388 - kl_loss: 7.1526e-04
Epoch 9/10
75/75 [==============================] - 3s 35ms/step - loss: 0.1395 - reconstruction_loss: 0.1388 - kl_loss: 7.1526e-04
Epoch 10/10
75/75 [==============================] - 3s 33ms/step - loss: 0.1391 - reconstruction_loss: 0.1384 - kl_loss: 7.1526e-04
```

Training with GED

[5]

Next step

[1] []

Bibliography I

Yongin Choi, Ruoxin Li, and Gerald Quon.
Interpretable deep generative models for genomics.
unpublished, September 2021.

Francois Chollet.
Deep learning with python.
Manning Publications, New York, NY, October 2017.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville.
Deep Learning.
MIT Press, 2016.
http://www.deeplearningbook.org.

Diederik P Kingma and Max Welling.
Auto-encoding variational bayes, 2013.

Bibliography II

Qingguo Wang, Joshua Armenia, Chao Zhang, Alexander V Penson, Ed Reznik, Liguo Zhang, Thais Minet, Angelica Ochoa, Benjamin E Gross, Christine A Iacobuzio-Donahue, Doron Betel, Barry S Taylor, Jianjiong Gao, and Nikolaus Schultz.
Unifying cancer and normal RNA sequencing data from different sources.
Sci. Data, 5:180061, April 2018.