



Latent variable models and variational autoencoders

Dr. Luca Ambrogioni





Part I: Probabilistic latent variables models

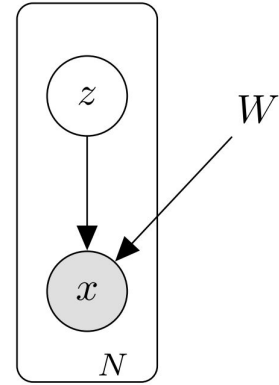
Dr. Luca Ambrogioni



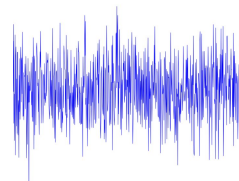
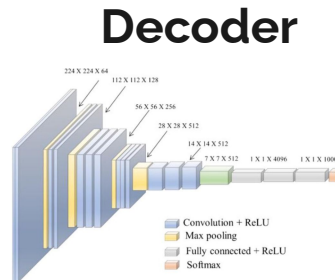
Deep unsupervised learning

Our aims:

- Uncover latent structures in the data
- Generate synthetic data



Observable data



Latent code



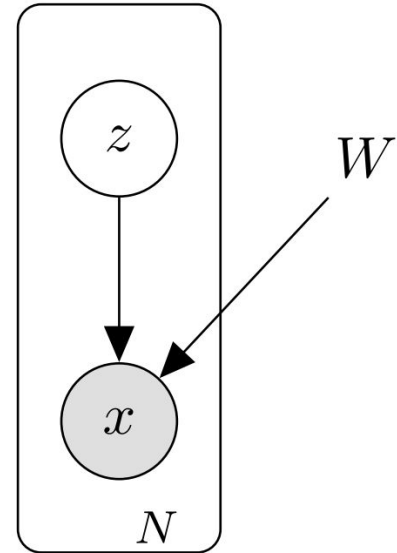
Latent variable models

Observable variable

Learnable
parameters

$$p(x_n | z_n; W)p(z_n)$$

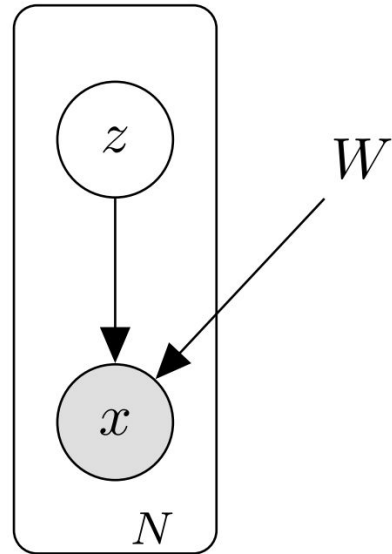
Latent variable



The marginal likelihood

In order to compute the likelihood, we need to marginalize out the latent variables

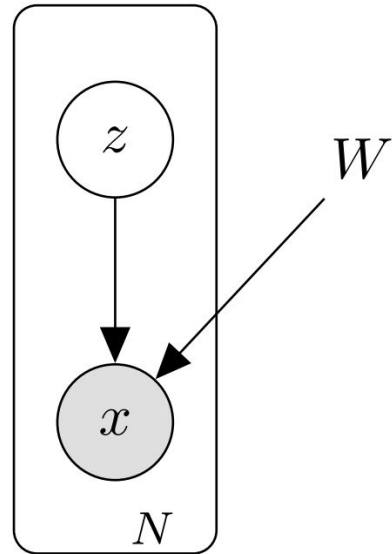
$$p(D \mid W) = \prod_{n=1}^N \int p(x_n \mid z_n; W) p(z_n) dz_n$$



The marginal log-likelihood loss

We can train the latent variable model by minimizing the negative marginal log-likelihood

$$\mathcal{L}(W) = - \sum_{n=1}^N \log \left(\int p(x_n | z_n; W) p(z_n) dz_n \right)$$



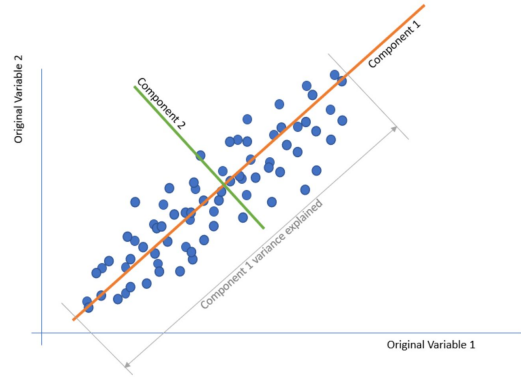
Probabilistic principal component analysis

Correlated data vector

$$p(x_n | z_n; W) = \mathcal{N}(W z_n, \sigma^2 I)$$

Uncorrelated latent variable

$$p(z_n) = \mathcal{N}(0, \nu^2 I)$$



Source image:

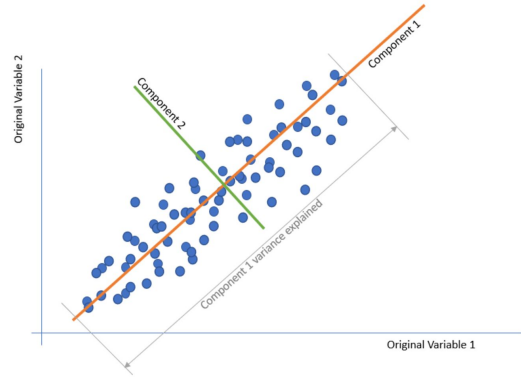
<https://towardsdatascience.com/what-is-the-difference-between-pca-and-factor-analysis-5362ef6fa6f9>

Probabilistic principal component analysis

Correlations arise from linear mixing

$$x_n = Wz_n + \xi_n$$

Observation noise



PCA: Marginalizing the latent variable

$$p(x_n) \propto \int_{-\infty}^{\infty} e^{-\frac{1}{2\sigma^2} \|x_n - Wz_n\|_2^2} e^{-\frac{1}{2\nu^2} \|z_n\|_2^2} dz_n$$

This is a Gaussian integral that can be solved analytically. It looks difficult though...

Gaussian marginalization: The simple way

A sum of two Gaussian variables is always another Gaussian variable!

Gaussian variable one

$$x_n = Wz_n + \xi_n$$

Gaussian variable two



Step one: Computing the mean

We can easily compute the mean of this marginalized Gaussian variable!

$$\mathbb{E}[x_n] = \mathbb{E}[Wz_n + \xi_n] = W\mathbb{E}[z_n] + \mathbb{E}[\xi_n] = 0$$

↑
Linearity of expectation

Step two: Computing the covariance matrix I

We can similarly compute the covariance matrix

**Correlations between latent and
noise = 0**

$$\begin{aligned}\mathbb{E}[x_n x_n^T] &= \mathbb{E}[(W z_n + \xi_n)(W z_n + \xi_n)^T] \\ &= W \mathbb{E}[z_n z_n^T] W^T + 2W \mathbb{E}[z_n \xi_n^T] + \mathbb{E}[\xi_n \xi_n^T]\end{aligned}$$

Latent covariance

Noise covariance

Step two: Computing the covariance matrix II

We can similarly compute the covariance matrix

Correlations (non-diagonal covariance matrix) arise from linear mixing of components

$$\mathbb{E}[x_n x_n^T] = W(\sigma^2 I)W^T + \nu^2 I = \sigma^2 W W^T + \nu^2 I$$

Latent covariance

Noise covariance



The PCA marginal likelihood

Et voila! We obtained the marginal likelihood without touching the integral!

$$p(x_n) = \mathcal{N}(x_n \mid 0, K)$$

$$K = \sigma^2 W W^T + \nu^2 I$$



PCA: Marginal log-likelihood

$$\mathcal{L}(W) = \frac{1}{2} \sum_{n=1}^N x_n^T K^{-1} x_n + \frac{1}{2} \log \det K + c$$

$$K = \sigma^2 W W^T + \nu^2 I$$



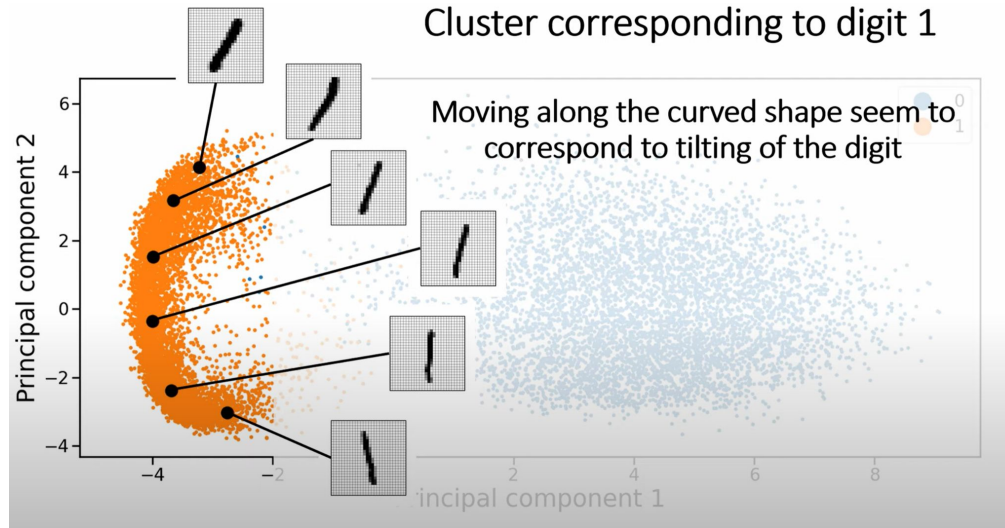
PCA by gradient descent

This isn't the most efficient way since the result can be obtained by eigendecomposition of the data covariance matrix


$$\mathcal{L}(W) = \frac{1}{2} \sum_{n=1}^N x_n^T K^{-1} x_n + \frac{1}{2} \log \det K + c$$

$$W_{k+1} = W_k - \eta \nabla \mathcal{L}(W_n)$$

The PCA uncovers latent structures in the data



Video: <https://www.youtube.com/watch?v=4UNlDjWV3ls>



Part II: From important sampling to VAEs

Dr. Luca Ambrogioni



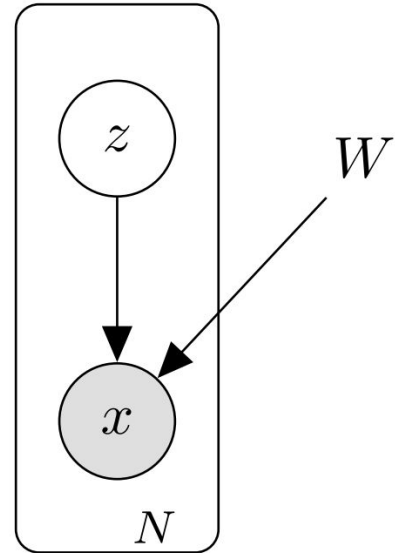
Back to latent variable models

Observable variable

Learnable
parameters

$$p(x_n | z_n; W)p(z_n)$$

Latent variable

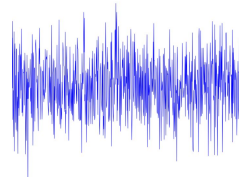
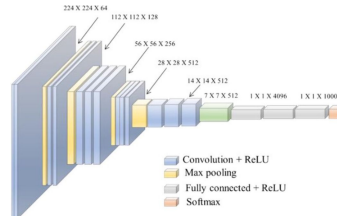


Deep latent variable models

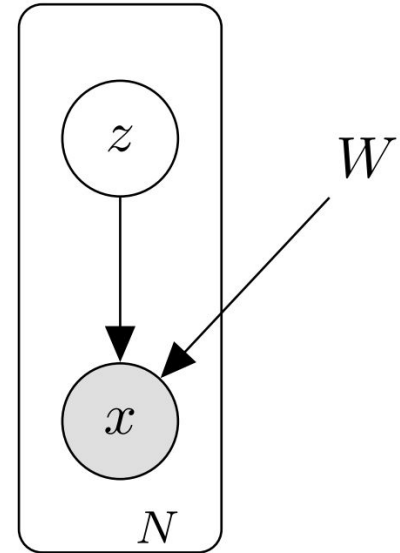


Observable data

Decoder

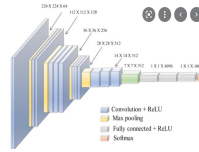


Latent code



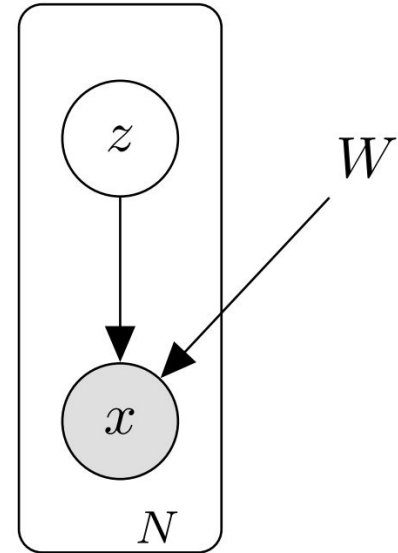
Deep latent variable models

Deep decoder



$$p(x_n | z_n, W) = \mathcal{N}(x_n | f_W(z_n), \sigma^2 I)$$

$$p(z_n) = \mathcal{N}(z_n | 0, I)$$





Deep marginal log-likelihood

Problem: The loss is now expressed through an intractable integral (usually) without closed form solutions


$$\mathcal{L}(W) = - \sum_{n=1}^N \log \left(\int \mathcal{N}(x_n | f_W(z_n), \sigma^2 I) \mathcal{N}(z_n | 0, I) dz_n \right)$$



Possible solution: Using a point estimate for the latent variable

We make the approximate assumption that the variable can be expressed as a deterministic function of the observable data

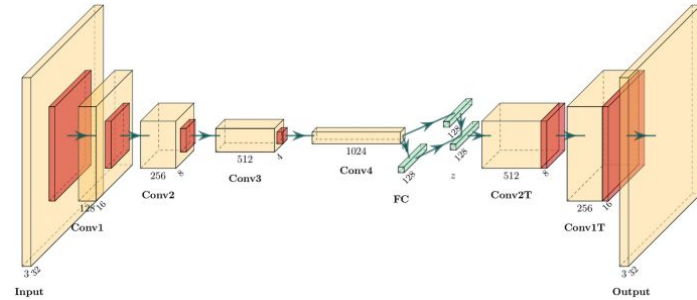
$$\hat{z}_n = g_{\Psi}(x_n)$$



Parameterized
encoder

A deep autoencoder loss

Encoder parameters



$$\mathcal{L}(W, \Psi) \approx - \sum_{n=1}^N \log (\mathcal{N}(x_n | f_W(g_\Psi(x_n)), \sigma^2 I) \mathcal{N}(g_\Psi(x_n) | 0, I))$$

$$= \frac{1}{2\sigma^2} \sum_{n=1}^N (x_n - f_W(g_\Psi(x_n)))^2 + \frac{1}{2} g_\Psi(x_n)^2 + c$$

Autoencoder
reconstruction loss

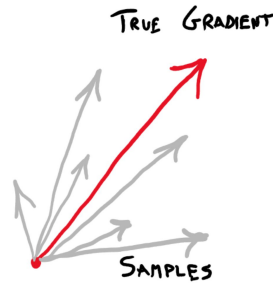
Regularization (keeps
the latents near zero)

Another approach: Monte Carlo sampling

Instead of using a point estimate and obtaining an approximate loss, we can use an unbiased stochastic estimate of the true loss. The simplest choice is to sample latents from the prior:

$$\mathcal{L}(W) \approx \sum_{n=1}^N \log \left(\frac{1}{J} \sum_{j=1}^J \mathcal{N}(x_n \mid f_W(z_{n,j}), \sigma^2 I) \right)$$

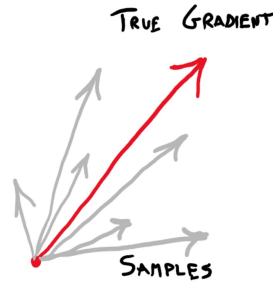
$$z_{n,j} \sim p(z)$$



Problem I: We need a better sampler!

This approach does not work! The estimate is unbiased but the variance is too high! Using random latents is too inefficient!

$$\mathcal{L}(W) \approx \sum_{n=1}^N \log \left(\frac{1}{J} \sum_{j=1}^J \mathcal{N}(x_n \mid f_W(z_{n,j}), \sigma^2 I) \right)$$



$z_{n,j} \sim p(z)$ ←

The sampler should depend on x . Now it just guesses it at random!



Possible solution: Importance sampling

We can sample from another distribution and then correct for the bias

Bias correction

$$\mathcal{L}(W) \approx \sum_{n=1}^N \log \left(\frac{1}{J} \sum_{j=1}^J \overset{\text{Bias correction}}{\downarrow} w(z_{n,j}) \mathcal{N}(x_n \mid f_W(z_{n,j}), \sigma^2 I) \right)$$

$$w(z) = \frac{p(z)}{q(z \mid x; \Psi)}, \quad z_{n,j} \sim q(z \mid x_n; \Psi)$$



The exact posterior is the best sampler

If the sampler is the exact posterior, the estimate is deterministically equal to the exact marginal log-likelihood

$$\mathbb{E}_z[w(z)p(x | z; W)] = \mathbb{E}_z \left[\frac{p(z)p(x; W)}{p(x | z; W)p(z)} p(x | z; W) \right] = p(x; W)$$



Problems with importance sampling

- The variance is very high unless the sampler is extremely close to the true posterior. The weights will take values that differ from order of magnitudes so that only a few weights will dominate the estimate.
- It is not clear how to choose a good sampler.



Interlude: The Jensen inequality

$$\log \mathbb{E}[f(x)] \geq \mathbb{E}[\log f(x)]$$



From importance sampling to the evidence lower bound

If we use Jensen inequality on the marginal log-likelihood, we obtain a much more tractable lower bound of the exact loss

$$\log \mathbb{E}_{q(z|x; \Psi)} [w(z) \mathcal{N}(x | f_W(z), \sigma^2 I)] \geq \mathbb{E}_{q(z|x; \Psi)} \left[\log \frac{\mathcal{N}(x | f_W(z), \sigma^2 I) p(z)}{q(z | x; \Psi)} \right]$$



The negative ELBO

Moving the log inside the expectation makes the importance sampling weights much “tamer”. The price is a non-zero bias.

$$L(W, \Psi) = - \sum_{n=1}^N \mathbb{E}_{q(z|x_n; \Psi)} \left[\log \frac{\mathcal{N}(x_n | f_W(z), \sigma^2 I) p(z)}{q(z | x; \Psi)} \right]$$



The negative ELBO: Autoencoder loss + Regularization

The final result is not much different from our previous deterministic autoencoder approximation

$$L(W, \Psi) = - \sum_{n=1}^N \mathbb{E}_{q(z|x; \Psi)} \left[-\frac{1}{2\sigma^2} (x_n - f_W(z))^2 \right] + D_{\text{KL}}(q(z | x_n; \Psi), p(z))$$



**Probabilistic
autoencoder loss**



The ELBO, our old friend

In the last lecture, we saw that minimizing the negative ELBO trains the sampler towards the true posterior. This means that we can jointly train the parameters of the sampler and the model parameters by minimizing the negative ELBO!

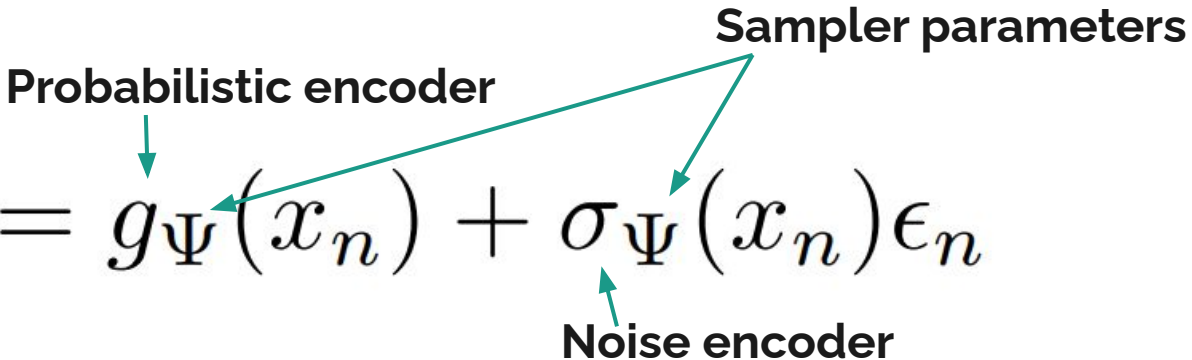
$$W_{k+1} = W_k - \eta \nabla_W L(W_k, \Psi_k)$$

$$\Psi_{k+1} = \Psi_k - \eta \nabla_\Psi L(W_k, \Psi_k)$$



Estimating the gradient, the reparameterization trick

In order to estimate the gradient, we can use the reparameterization trick. The only difference is that now the posterior parameters must depend on the data S .


$$z_n = g_{\Psi}(x_n) + \sigma_{\Psi}(x_n)\epsilon_n$$

Labels and arrows in the diagram:

- Probabilistic encoder** points to g_{Ψ}
- Noise encoder** points to σ_{Ψ}
- Sampler parameters** points to the entire expression $z_n = g_{\Psi}(x_n) + \sigma_{\Psi}(x_n)\epsilon_n$



Finally, the VAE loss

We finally arrived to variational autoencoders! It is common to use a single sample of ϵ

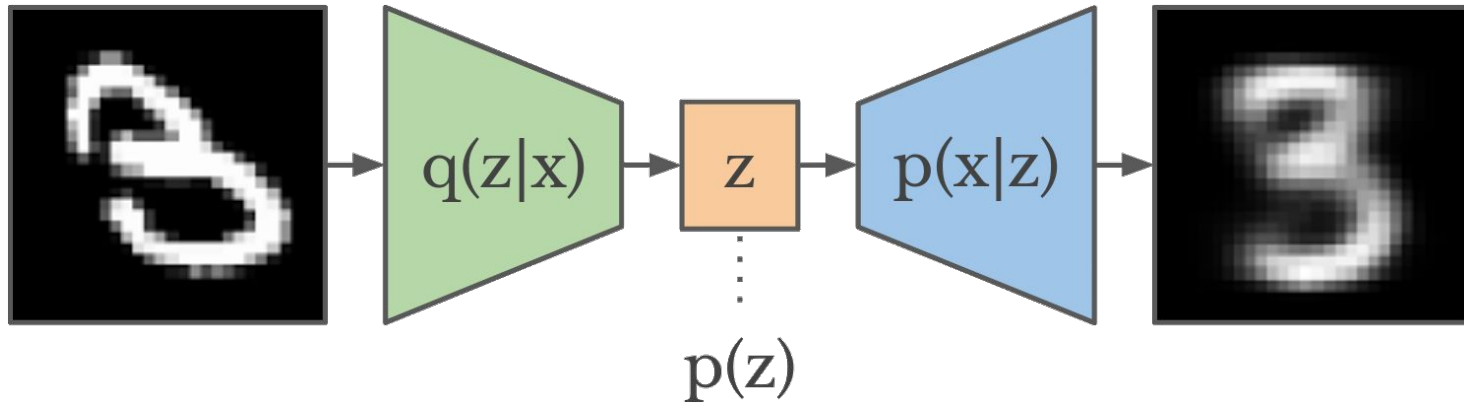
$$L(W, \Psi) \approx -\frac{1}{2\sigma^2} \sum_{n=1}^N \left(x_n - f_W(g_\Psi(x_n) + \sigma_\Psi(x_n)\epsilon) \right)^2 + D_{\text{KL}}(q(z | x_n; \Psi), p(z))$$

**Probabilistic encoder
loss**

$$\epsilon \sim \mathcal{N}(0, I)$$



VAEs in practice



Blogpost:

<https://danijar.com/building-variational-auto-encoders-in-tensorflow/>



Thank you.

