Advanced Topic: Unsupervised Learning

Machine Learning

Michael Wand

michael.wand@idsia.ch

Dalle Molle Institute for Artificial Intelligence Studies (IDSIA) USI - SUPSI

Fall Semester 2024

Introduction

- In this lecture, we consider techniques of unsupervised learning (UL).
- Unsupervised tasks are defined by *unannotated* data, i.e. there are no targets (like for classification or regression) to be learned.
- Instead, unsupervised learning deals with discovering *structure* in data.
- We distinguish passive UL, where we discover regularities and structure in *given* data, and active UL, where data (from still images to movements of an actor in a complex world) is actively generated.
- Unsupervised learning can stand on its own, or it can be a step in a more complex pipeline.

Introduction

Classical tasks of Unsupervised Learning:

- Clustering and anomaly detection
- Probability density estimation
- **Data compression / dimensionality reduction**
- **Active unsupervised learning with Neural Networks: VAE and GAN**

USI/SUPSI
Istituto
Dalle Molle
di studi
sull'intelligenza
artificiale
IDSIA

Data Compression and Dimensionality Reduction

Data Compression and Dimensionality Reduction

- Compressing data is a fundamental task in Data Analysis and Machine Learning.
- Elementary applications: efficient storage, feature extraction, avoiding curse of dimensionality, understanding and visualizing structure, making predictions on the future
- Advanced applications: driving a learning agent to explore new possibilities, artificial curiosity / art / creativity

PCA: Introduction

- Principal Component Analysis (PCA) is *the* linear method for dimensionality reduction, lossy data compression, feature extraction, and also data visualization.
- Two equivalent definitions:
  - ➔ project data (orthogonally) onto a lower-dimensional subspace to that the *variance* of the projected data is maximized
  - ➔ project data (orthogonally) onto a lower-dimensional subspace to that the *projection cost* is minimized.
- We will soon see that these definitions are equivalent.
- In this section, we always assume that our data has mean zero (otherwise the projection subspace will simply be displaced by a fixed vector).

PCA

- Images: General projection (left) / orthogonal projection (right).
- Any projection from a D-dimensional space to an M-dimensional subspace can be described using a suitable *basis* $\mathbf{b}_i$, $i = 1, \ldots, D$: If $\mathbf{x} = \sum_{i=1}^{D} \alpha_i \mathbf{b}_i$, then the projection is $\tilde{\mathbf{x}} = \sum_{i=1}^{M} \alpha_i \mathbf{b}_i$.
- In the specific case of an orthogonal projection, the basis can be chosen to be orthonormal, i.e. $\mathbf{b}_i^T \mathbf{b}_j = 0$ for $i \neq j$ and $\mathbf{b}_i^T \mathbf{b}_j = 1$.

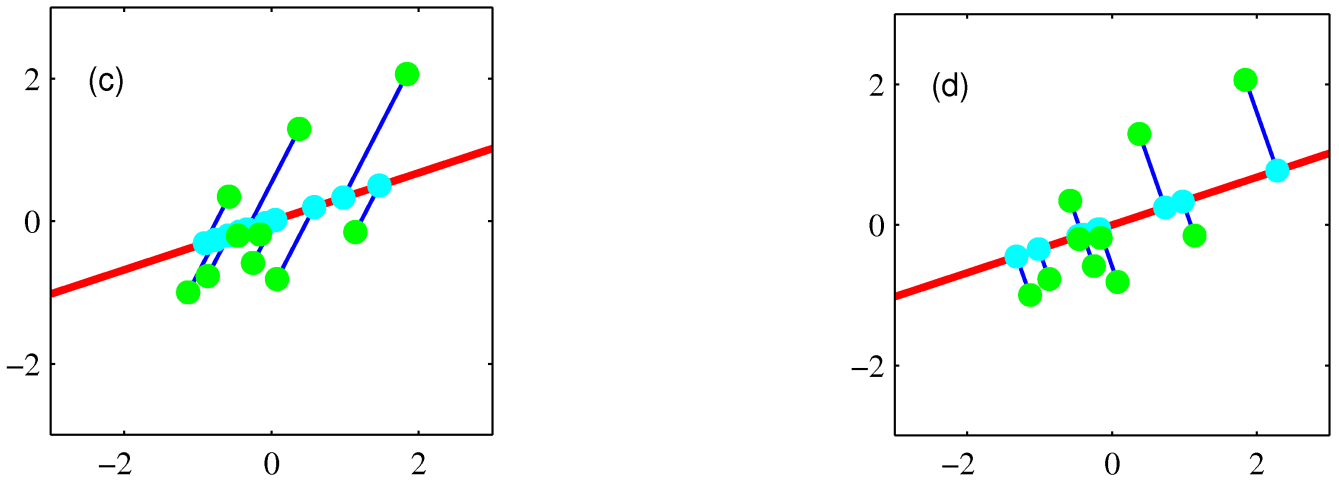


Image source: Bishop, figure 12.12 (part)

PCA

- For a sample $\mathbf{x}$ which is projected to $\tilde{\mathbf{x}}$, the projection cost is defined as the squared distance between data point and its projection $||\mathbf{x} - \tilde{\mathbf{x}}||^2$.
- Now assume we have a set of samples $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}$.
- We will minimize the projection cost over all samples, i.e. we minimize

$$J = \frac{1}{N} \sum_{n=1}^{N} ||\mathbf{x}^{(n)} - \tilde{\mathbf{x}}^{(n)}||^2.$$

- We take from basic linear algebra that for any fixed projection subspace, the projection with smallest projection cost will be an orthogonal projection.
- Thus, we only look for orthogonal projections.

PCA

- Assume $\mathbf{b}_1, \ldots, \mathbf{b}_D$ is an orthonormal basis of the space.
- For each $\mathbf{x}^{(n)}$, we have the representation $\mathbf{x}^{(n)} = \sum_{i=1}^{D} \alpha_i^{(n)} \mathbf{b}_i$.
- The sum $\sum_{i=1}^{D} (\alpha_i^{(n)})^2$ is the squared distance of $\mathbf{x}^{(n)}$ to the origin, it does *not depend* on the concrete choice of the orthonormal basis $\mathbf{b}_i$.
- Consequently, the sum

$$V = \frac{1}{N} \sum_{i=1}^{D} \sum_{n=1}^{N} (\alpha_i^{(n)})^2$$

  is also basis-independent.
- Since the data has mean zero, $\frac{1}{N} \sum_{n=1}^{N} (\alpha_i^{(n)})^2$ is the *variance* of the data in direction i, and V is the total variance of the data.

PCA

- For a given basis, consider the projection

$$\mathbf{x}^{(n)} = \sum_{i=1}^{D} \alpha_i^{(n)} \mathbf{b}_i \qquad \Rightarrow \qquad \tilde{\mathbf{x}}^{(n)} = \sum_{i=1}^{M} \alpha_i^{(n)} \mathbf{b}_i .$$

- Since the basis is orthogonal, the projection cost is

$$\frac{1}{N} \sum_{n=1}^{N} ||\mathbf{x}^{(n)} - \tilde{\mathbf{x}}^{(n)}||^2 = \frac{1}{N} \sum_{n=1}^{N} \sum_{i=M+1}^{D} (\alpha_i^{(n)})^2$$

  and the variance of the projected data is

$$\frac{1}{N} \sum_{n=1}^{N} ||\tilde{\mathbf{x}}^{(n)}||^2 = \frac{1}{N} \sum_{n=1}^{N} \sum_{i=1}^{M} (\alpha_i^{(n)})^2 .$$

- Since their sum is constant (the total variance V), minimizing the projection cost is equivalent to maximizing the variance of the projected data!

PCA

- Given the projection dimensionality M, how to find a basis $\mathbf{b}_i$ such that the variance of the projected data is maximized / the projection cost is minimized?
- Consider the *covariance matrix* of the data

$$\Sigma = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}^{(n)} (\mathbf{x}^{(n)})^T.$$

- This matrix can be orthogonally diagonalized, leading to an orthonormal basis $\mathbf{u}_1, \ldots, \mathbf{u}_D$ of eigenvectors of Σ, where the eigenvalues $\lambda_1, \ldots, \lambda_D$ are the data variances in the directions of $\mathbf{u}_1, \ldots, \mathbf{u}_D$.
- We sort the basis vectors such that $\lambda_1 > \ldots > \lambda_D$. (If some of the λ are equal, the solution might not be unique any more.)

PCA

- We choose the basis of the projection subspace as the first M eigenvectors of the data covariance matrix Σ!
- The total variance of the data *in the projection subspace* will be $\lambda_1 + \ldots + \lambda_M$, and the projection cost will be $\lambda_{M+1} + \ldots + \lambda_D$.
- It is intuitively clear (and not difficult to show) that for fixed M, any other choice of the projection subspace increases the projection cost / reduces the variance of the projected data.
- How to choose M? Often by inspection.
  - ➔ Compute all eigenvalues and eigenvectors of Σ.
  - ➔ Frequently, a large part of the data variance is found in the first few directions, so that one can choose the "cutoff" M to be relatively low.

PCA for Data Exploration

Istituto Dalle Molle di studi sull'intelligenza artificiale
USI/SUPSI
IDSIA

Considering PCA as a method for detecting directions of high variance offers a way to use it for *exploring* and *visualizing* data:

- Take any dataset (example: faces, see below). We are interested in understanding the factors of variation in the data.
- Compute the data mean, and the first few PCA dimensions (i.e. dimensions of high variance).
- In some cases, this is enough to extract interesting factors from the data.
- Consider the example below (*Eigenfaces*): The leftmost image is an "average" face, the other ones are the first few principal components. Du you find the one which corresponds to having a beard?

Source: Shakhnarovich & Moghaddam, Face Recognition in Subspaces Mitsubishi Electric Research Laboratories, Technical Report, 2004

Relationship to LDA

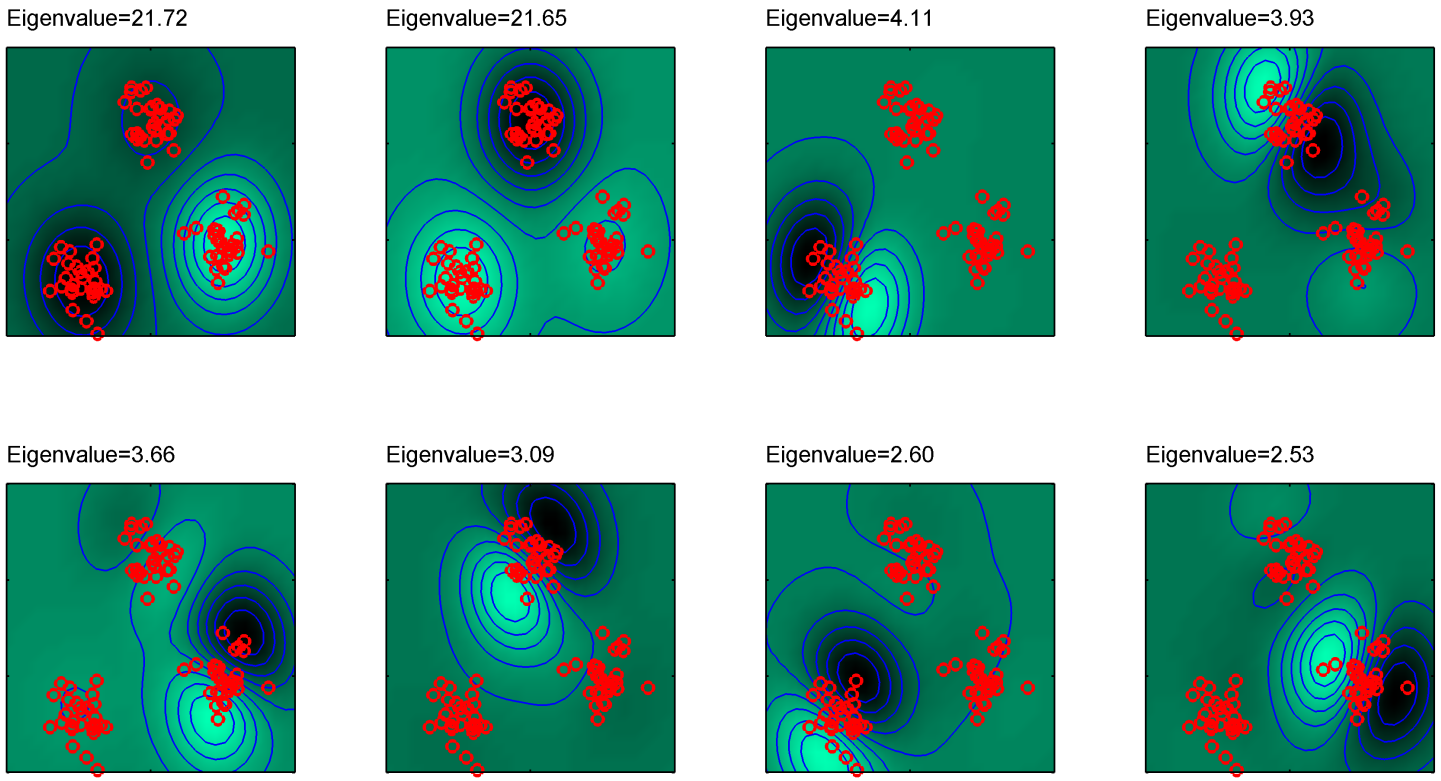
- For the PCA, we have obtained a basis of the projection space by considering the eigenvalues of the data covariance matrix.
- This is related to the optimization criterion for the Linear Discriminant Analysis (LDA, or Fisher's Linear Discriminant); where we had the ratio of two covariance matrices (within-class covariance and between-class covariance):

$$J_{\text{LDA}}(\mathbf{W}) = \text{Tr}((\mathbf{W}\mathbf{S}_\mathbf{W}\mathbf{W}^T)^{-1}(\mathbf{W}\mathbf{S}_\mathbf{B}\mathbf{W}^T)).$$

- The LDA projection space is spanned by the M largest eigenvectors of $\mathbf{S}_\mathbf{W}{}^{-1}\mathbf{S}_\mathbf{B}$.
- One can use both PCA and LDA for data visualization and dimensionality reduction in the same way (project to a low-dimensional subspace which best matches the criterion).
- To some extent, the PCA can be considered an unsupervised version of the LDA.

Kernel PCA

- The PCA can be made nonlinear, for example by using a *kernel* formulation.
- A nice example (using a Gaussian kernel) is given in the image below.



Source: Bishop, figure 12.17

USI/SUPSI
Istituto
Dalle Molle
di studi
sull'intelligenza
artificiale
IDSIA

The Autoencoder

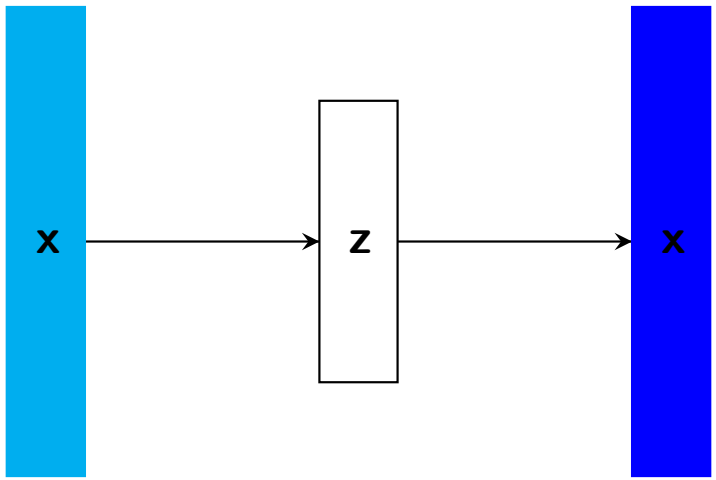
Introduction

- We aim at using neural networks for unsupervised representation learning: The goal is to get a representation of the input with desirable properties.
- Since neural network training always requires to compute some loss w.r.t. some kind of target, we need to define a of surrogate loss which will be used for training by gradient descent.
- Usually, we use a *reconstruction loss*: We evaluate whether the network is able to (re)construct the training data items.
- Major differences lie in the architecture and the input, as we will see.

Autoencoder Definition

- We consider the Autoencoder (AE), where the input and the output are identical: The system is trained to *encode and reconstruct* the data items which are passed to it.
- We are usually interested in the input representation in the hidden layer(s).
- The setup is the one we know from other regression tasks (final linear layer, MSE loss, gradient descent training).

- In this simple example, the network needs to learn to encode all information from the input **x** in the (smaller) layer **z**.
- In more complex setups, both the encoder and the decoder part could contain e.g. convolutions, skip connections, etc.



The Linear Autoencoder

- Normally, one would choose the usual nonlinearities (tanh, ReLU, etc.) for the inner layers of the autoencoder.
- An interesting observation is made if all layers are *linear*:
  - ➔ In this case, the autoencoder computes a linear projection on a subspace whose dimensionality is given by the size of the smallest inner layer.
  - ➔ This linear projection spans the same space as the PCA; the only difference is that the neural network is not constrained to find an orthonormal basis.
- In the nonlinear case, the autoencoder learns a nonlinear compression of the input data.

The Denoising Autoencoder

- If the inner layer is *greater* than the input, the autoencoder might simply learn the *identity* function.
- In this case, each input sample is reconstructed (almost) perfectly, but the autoencoder learns no interesting structure!
- Even if the inner layer is smaller than the input, there may still be some overfitting.
- A simple way to push the autoencoder towards learning stable high-level representations of the input: The Denoising Autoencoder.
  - ➔ During training, the input data is corrupted by noise (e.g. Gaussian noise, random masking, etc).
  - ➔ The target is the *uncorrupted* version of the input.
  - ➔ In the test phase, the uncorrupted input data is fed into the network to extract representations.

Autoencoder: Conclusion

- A variety of other autoencoder variations has been proposed (contractive AE, sparse AE), usually constraining the repesentations in some way.
- The usual regularization methods are likewise applicable.

USI/SUPSI
Istituto
Dalle Molle
di studi
sull'intelligenza
artificiale
IDSIA

Active Unsupervised Learning with a Probabilistic Model: The Variational Autoencoder

Introduction

- So far, we have done *passive unsupervised learning*: No data was actively generated.
- We now consider an architecture which allows data *generation* from random noise: The Variational Autoencoder (VAE).
- This will be our first example of *active unsupervised learning*.
- Examples will be mostly from image generation, i.e. we generate images which *resemble* a set of training images, but are not identical to the training images.

For the Variational Autoencoder, I follow Carl Doersch's tutorial (`https://arxiv.org/pdf/1606.05908.pdf`) and occasionally Joseph Rocca's Tutorial (`https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73`)

The Variational Autoencoder

- An elementary take of generative modelling:
  - ➔ Describe the training set $\mathcal{X}$ by a probability distribution $P(X)$
  - ➔ Sample from $P(X)$ to generate new images!
- What could be the problem of such an approach?
  - ➔ we need to estimate a distribution on a high-dimensional space with a limited amount of training data
  - ➔ finding a metric (deterministic or probabilistic) which describes similarity between images is notoriously difficult
  - ➔ if the distribution $P(X)$ is complicated, sampling might be intractable.
- We will solve the problem by learning an underlying low-dimensional structure of the data.
- We know that neural networks can do that! Let us see how to use them in a probabilistic setting.

The Variational Autoencoder

- Idea:
  - ➔ Introduce a *latent vector* as a low-dimensional representation of the data.
  - ➔ Use a suitable neural network architecture to convert this latent representation into the desired image (we should also think on how to *get* a suitable latent representation).
- Assuming that we can sample from this latent space, we have solved all three issues from the last slide:
  - ➔ we have obtained a low-dimensional representation of the training images, mitigating dimensionality issues
  - ➔ if we have done it properly, the neural network has learned the structure of the image space
  - ➔ we can sample in the latent space, and then convert this sample to a proper image.

The Variational Autoencoder

- The (standard) autoencoder creates a low-dimensional representation from the training data!
- Can we use the autoencoder to solve our problem?
- **No**, since the latent space (the representations in some inner layer) of the autoencoder does not have a regular structure!
  - ➔ Some areas might not correspond to sensible images when decoded.
  - ➔ Also, it is not clear whether a small/large distance in the latent space corresponds to a small/large distance in the data space.

Image source: Rocca's VAE tutorial

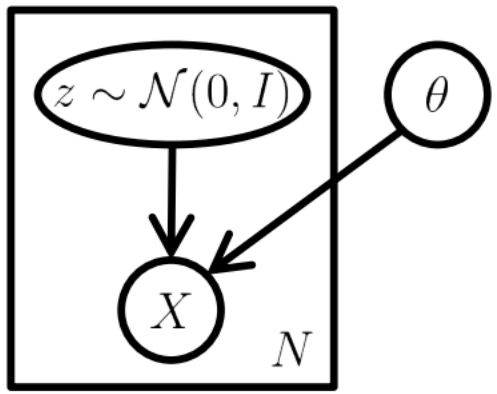
The Variational Autoencoder

- The variational autoencoder follows a somewhat different pattern.
- It is best introduced from a perspective of Bayesian modeling.
- We describe the latent variable model as

$$P(X) = \int_z P(X|z;\theta)P(z)\,\mathrm{d}z$$

with data X and latent vector z.

- We make the mapping from the latent space to the image probabilistic in order to formalize the notion that we generate images which are *like* the training data.
- Commonly, $P(X|z;\theta) = \mathcal{N}(X|f(z;\theta), \sigma^2 I)$ with a deterministic function $f(z,\theta) = f(z)$ and a hyperparameter σ.



Sampling as a graphical process, the "plate" means that we generate N samples, keeping the parameters θ fixed.

The Variational Autoencoder

- How to choose the latent variables?
- In common purely probabilistic models, the latent variables have a semantic interpretation.
- But in the case of neural networks, we usually do not want to hand-craft such an interpretation...
- For the VAE, we let the latent vector z follow a normal distribution:

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

  and let the "decoding" neural network $f(z)$ learn how to map these z to some reasonable image!
- If there are semantic categories which help creating reasonable images, we expect the neural network to automatically learn them!

The Variational Autoencoder

- We could possibly train such a model by gradient descent.
  - ➔ start with a training image X (or a batch of images)
  - ➔ sample a batch $z_1, \ldots, z_n$
  - ➔ compute $P(X) \approx \frac{1}{n} \sum_i P(X_i|z_i)$
  - ➔ compute the gradient and perform gradient ascent.
- What is still missing?
- The z are sampled without taking X into account!
- This means that

$$P(X|z) = \frac{1}{(2\pi)^{D/2}\sigma^D} \exp\left(-\frac{1}{2\sigma^2}(X - f(z))^T(X - f(z))\right)$$

  will almost always be very small! (D is the dimensionality of the latent vector.)
- In order to learn a reasonable function $f(z)$, one would have to consider *extremely* many samples!

The Variational Autoencoder

- We solve the training problem by making the sampling dependent on the input image.

$\Rightarrow$ Compute $P(X) \approx \frac{1}{n} \sum_i P(X_i|z_i)$ only from values of z which are *likely* to have produced an image similar to X.

- Requires an *encoder* probability distribution $Q(z|X)$.
- Just like the decoding part, Q is normally distributed:

$$Q(z|X) = \mathcal{N}(z|\mu(X;\psi), \Sigma(X;\psi))$$

  where μ and Σ are parametrized by neural networks.
- Only remaining issue: If we take samples according to Q, we easily compute $E_{z\sim Q} P(X|z)$.
- How do we relate this to $P(X)$?
- We formulate the relationship between $E_{z\sim Q} P(X|z)$ and $P(X)$ using Bayesian modeling!

The Variational Autoencoder

- We have

$$D_{\mathrm{KL}}(Q(z|X)||P(z|X)) = E_{z\sim Q}(\log Q(z|X) - \log P(z|X))$$
$$= E_{z\sim Q}(\log Q(z|X) - \log P(X|z) - \log P(z)) + \log P(X).$$

- This is easily rearranged to

$$\log P(X) - D_{\mathrm{KL}}(Q(z|X)||P(z|X))$$
$$= E_{z\sim Q}(\log P(X|z)) - D_{\mathrm{KL}}(Q(z|X)||P(z)).$$

- Consider this equation:
  - ➔ $P(X)$ is what we want to optimize.
  - ➔ $D_{\mathrm{KL}}(Q(z|X)||P(z|X))$ is an error term that is small if training samples can be reconstructed, it is small when Q is versatile enough.
  - ➔ Optimize the right-hand side!

The Variational Autoencoder

- We need to optimize

$$E_{z\sim Q}(\log P(X|z)) - D_{\mathrm{KL}}(Q(z|X)||P(z)).$$

- The last term is the KLD between two multivariate Gaussian distributions, which can be computed exactly (see Doersch, formulas (6) and (7) for details).
- As usual in stochastic gradient descent, $E_{z\sim Q}(\log P(X|z))$ is computed on a per-sample basis, or by averaging over a mini-batch, i.e.

$$E_{z\sim Q}(\log P(X|z)) \approx \frac{1}{n}\sum_{n} \log P(X_n|z_n)$$

where the z_n are sampled from the distribution $Q(z|X_n)$.

The Variational Autoencoder

- The full processing chain is shown in the image.
- There are two losses (in blue), the KLD loss and the reconstruction loss.
- Computing the gradient is now easy. . . ?

$\Rightarrow$ . . . for the *Decoder* part (and the KLD loss).

- However, we cannot backpropogate across the sampling operation to the Encoder!

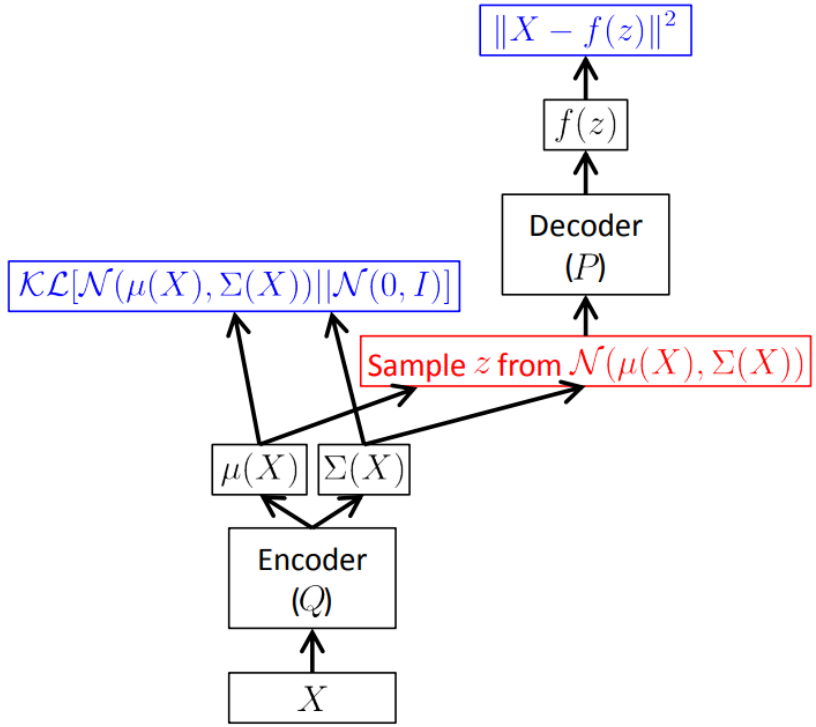


Image source: Doersch, Fig. 4 (partial)

The Variational Autoencoder

- We observe that the trainable parameters of the Encoder are just the mean and the covariance of the Gaussian distributions.
- The backpropagation problem can be solved by moving the sampling operation to a separate input layer (reparametrization trick).
- The image shows the final processing chain for training the variational autoencoder.
- As always, there is a variety of hyperparameters (network topology, learning rate, *size of latent vector z*, . . .)

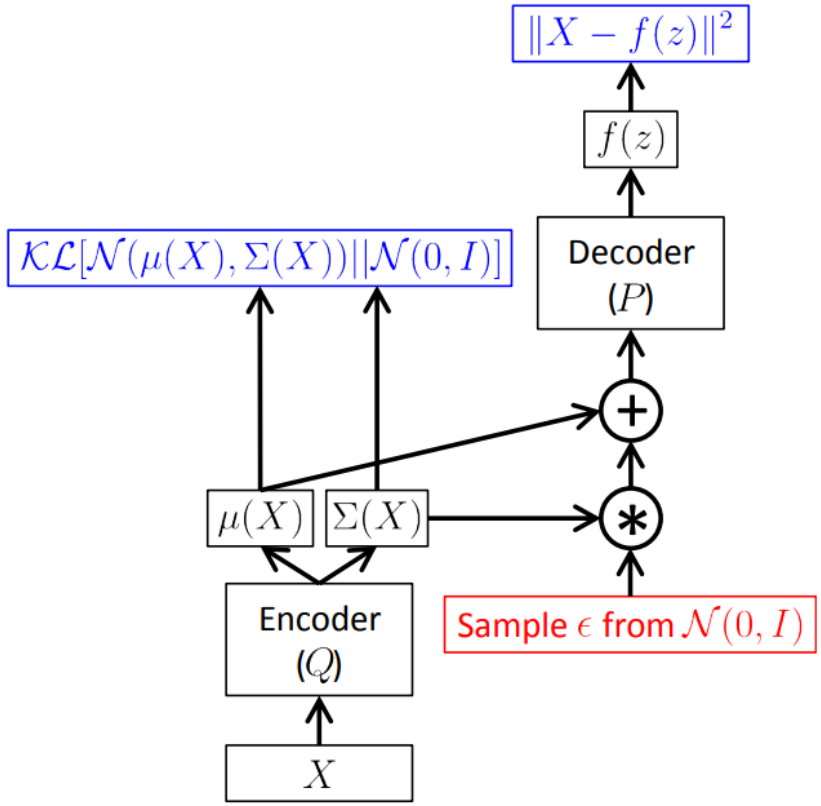


Image source: Doersch, Fig. 4 (partial)

The Variational Autoencoder

- Sampling from the trained variational autoencoder is very simple:
  1. Sample from a standard normal distribution
  2. Perform the forward pass of the VAE.
- The encoder part is no longer needed.

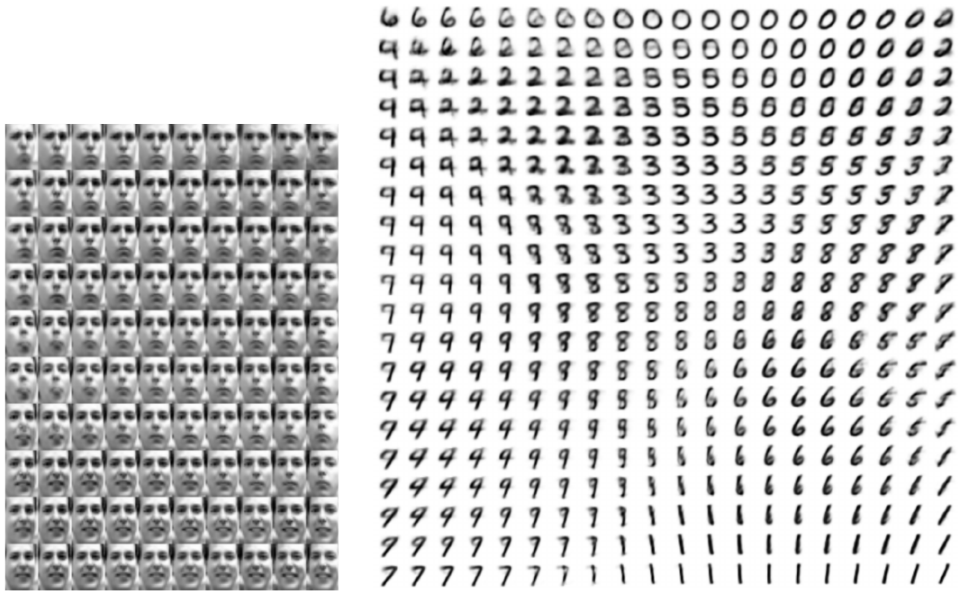
The Variational Autoencoder - Alternative View

An alternative view of the VAE (from Rocca's tutorial) creates a connection to the standard autoencoder.

- Key idea: Encode an input sample not as a point, but as a *distribution* in latent space!
- Thus, the entire architecture is trained towards a sampling-and-decoding paradigm (unlike the entirely deterministic standard autoencoder)!
- The encoding of data points as distributions is however not enough to ensure a regularized latent space.
  - ➔ The input data points could be encoded as distributions with very small variance, or very different means, thus imitating a classical autoencoder.
- The latent space is regularized by augmenting the loss with a *regularization term*, namely the KL divergence (over all training data) between the latent representation and a standardized Gaussian.
- This ensures that all regions of the latent space are meaningful.
- Since the VAE also learns how to reconstruct samples, different regions of the latent space will have different semantic interpretations.

The Variational Autoencoder - Example

- No talk about generative models comes without an example :-)
- These examples are taken from the original VAE paper; where the VAE learned to encode a) faces, b) MNIST digits into a two-dimensional (!) latent space. Can you see the regularity?



(a) Learned Frey Face manifold (b) Learned MNIST manifold

Source: Kingma and Welling, *Auto-Encoding Variational Bayes*. Proc. ICLR 2014

The Variational Autoencoder - Conclusion

- With the variational autoencoder, we have successfully unified probabilistic modeling and neural networks!
- The VAE is considered one of the standard architectures for data generation.
- Many variations exist; in particular, the Conditional VAE allows to condition the sampling process (for example, for sampling a *specific* handwritten character, or a face with a specific expression).

USI/SUPSI
Istituto
Dalle Molle
di studi
sull'intelligenza
artificiale
IDSIA

Adversarial Learning for Unsupervised Data Generation and Adaptation

Adversarial Learning

- Gradient-based Adversarial learning[1] deals with pitting two subnetworks against each other.
- One of the networks is trained to solve the task we are interested in. The other one acts as an *adversary*: It takes the output, or some intermediate representation, of the first network, and is trained towards an objective which is adversarial to the underlying main task.
- The first network tries to "fool" the second one. More formally, the second network computes a dynamic *loss* for the first network.
- Training often alternates between the two sub-networks.

[1] Schmidhuber: *Making the world differentiable: On using fully recurrent self-supervised neural networks for dynamic reinforcement learning and planning in non-stationary environments*. Tech. Report, 1990.
Schmidhuber: *A possibility for implementing curiosity and boredom in model-building neural controllers*. In Proc. of the International Conference on Simulation of Adaptive Behavior: From Animals to Animats, 1991.
Schmidhuber: *Learning Factorial Codes by Predictability Minimization*. Neural Computation, 1992

Predictability Minimization

- One of the first concrete applications: A neural network is trained to compute an invertible representation of the input, so that the components of this representation (i.e. the values of the output neurons) are *statistically independent*.
- This means that independent factors are extracted from the input (which is a hot topic even today!).
- It is not easy to formulate a loss which measures statistical independence.
- Idea: The secondary network tries to *predict* each output neuron from the other output neurons.
- The performance of this prediction, which changes in each step, acts as the loss for the first network, which thus learns to *minimize the predictability* of its outputs.

This idea has seen a large number of applications. We get to know two of them, namely *Generative Adversarial Networks* (GAN) and unsupervised adversarial domain adaptation.

Generative Adversarial Networks

- The Generative Adversarial Network[2] aims at generating samples (e.g. images) from random noise, just like the VAE.
- This is a special case of *Artificial Curiosity* (will be covered later).
- Adversarial training idea: A generator is trained to produce samples from random noise. The adversarial discriminator tries to distinguish these "fake" samples from "real" samples from the training data set.

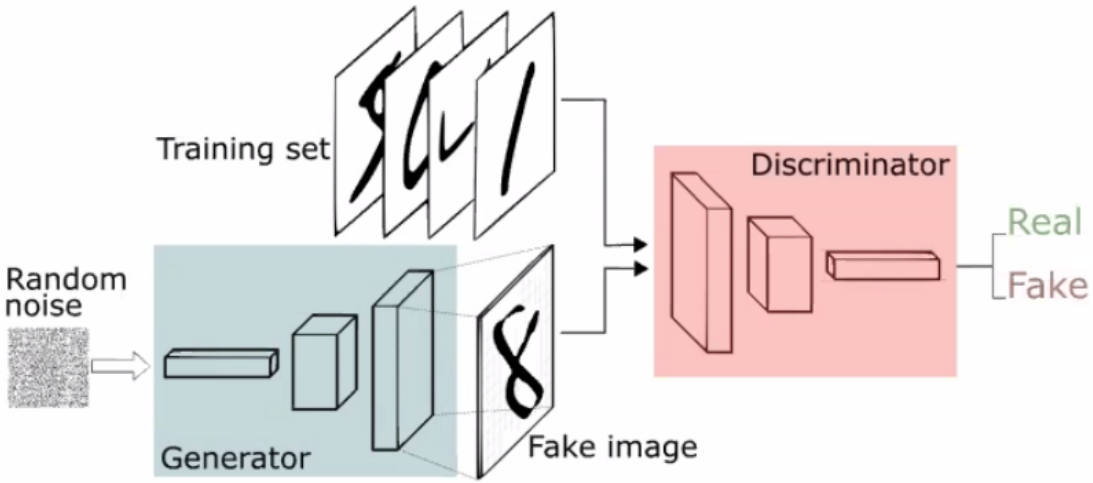


Illustration of a GAN (image credit: deeplearning4j.org)

[2] Goodfellow et al.: *Generative Adversarial Nets*. Proc. NIPS, 2014

Generative Adversarial Networks

- The discriminator is trained to distinguish real from fake samples (usually with standard cross-entropy loss). But it also acts as a loss for the generator.
  - ➔ if the generator produces "bad" samples, the discriminator works well, but the backpropagated loss should be high
  - ➔ if the generator produces "good" samples, the discriminator works badly, but the backpropagated loss should be low.
- There are several ways to cause such a behavior of the loss, e.g. inverting the backpropagated gradient, thus making the generator perform gradient ascent instead of gradient descent.
- Eventually, the goal is to solve the following minimax problem:

$$\min_G \max_D E_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}[\log D(\mathbf{x})] + E_{\mathbf{z}\sim p_{\mathbf{z}}}[\log(1 - D(G(\mathbf{z})))]$$

with the generator G, the discriminator D, the data $\mathbf{x}$, and the latent samples $\mathbf{z}$.

Generative Adversarial Networks

- While the VAE follows a Bayesian probabilistic model, the GAN is very engineering-oriented: It aims at finding a game-theoretic Nash equilibrium between the two subnetworks.
- This comes with several advantages and disadvantages compared to the VAE:
  - + The model is conceptually simple, and sampling is straightforward.
  - + Architectural variations (e.g. conditioning) are easily integrated into the model.
  - - The model is often difficult to train. In particular, if the discriminator works perfectly, the backpropagated gradient gets close to zero.
  - - No explicit modeling of the underlying probabilistic distributions.

Conclusion

- We have covered some elements of unsupervised learning (UL), focusing on Representation Learning and Generative Models.
- Regarding passive UL, we went from the linear PCA to nonlinear neural networks (autoencoders).
- Active UL (data generation) was covered by the VAE and the GAN.

We will revisit unsupervised learning in the context of Artificial Curiosity.