Deep Learning autoencoders and generative models

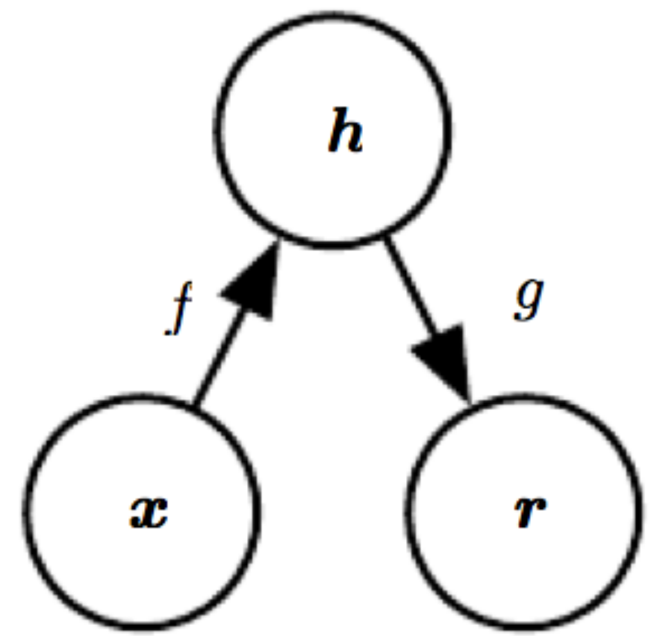
Luca Bortolussi

DMG, University of Trieste, IT
Modelling and Simulation, Saarland University, DE

DSSC @ TRIESTE

Autoencoders

Autoencoders are simply (deep) neural networks trained to
copy the input x into the output: y=x.

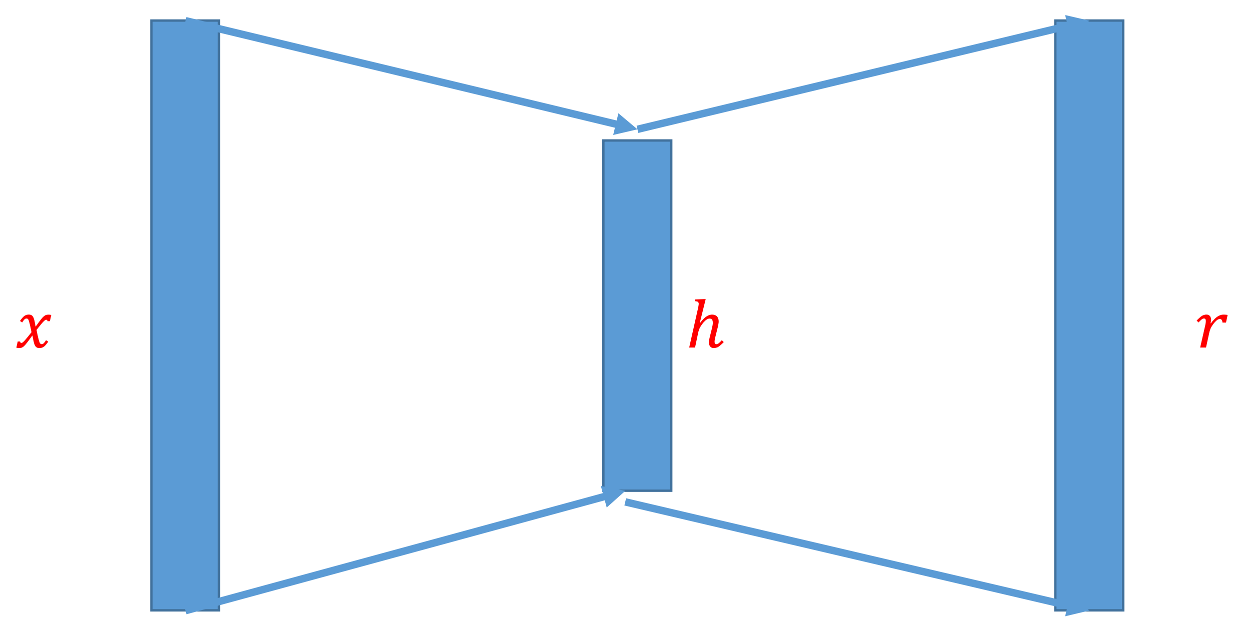


The hidden layer h is the interesting point: it learns a non-linear representation of the input. Autoencoders are thus **unsupervised learning methods for dimensionality reduction** (generalising PCA).

Autoencoders with too much capacity will just learn the identity map. Hence one has to use a lower dimension hidden layer or **regularise** them.

Sparse Autoencoders

$$L(x,r) = L(x, g(f(x)))$$



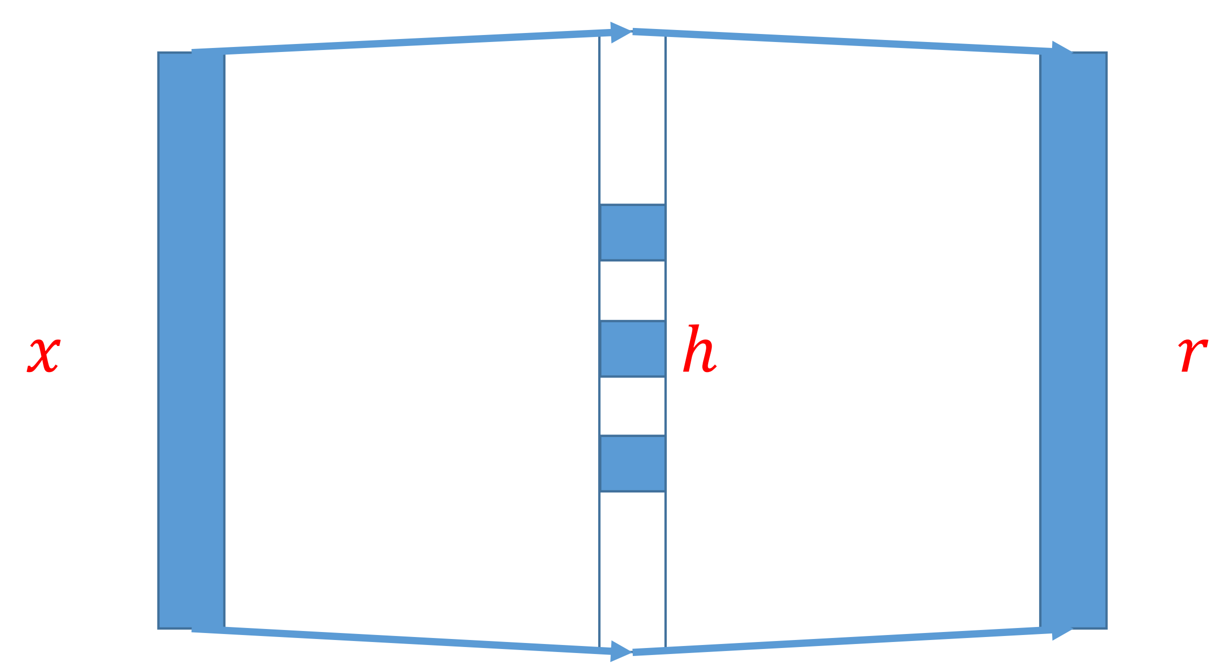
Undercomplete autoencoders

- dimension of hidden layer is less than input.
- without nonlinearities and with square loss, it is the PCA.
- nonlinearity improves but model capacity has to be taken into control.

$$L_R = L(x, g(f(x))) + R(h)$$

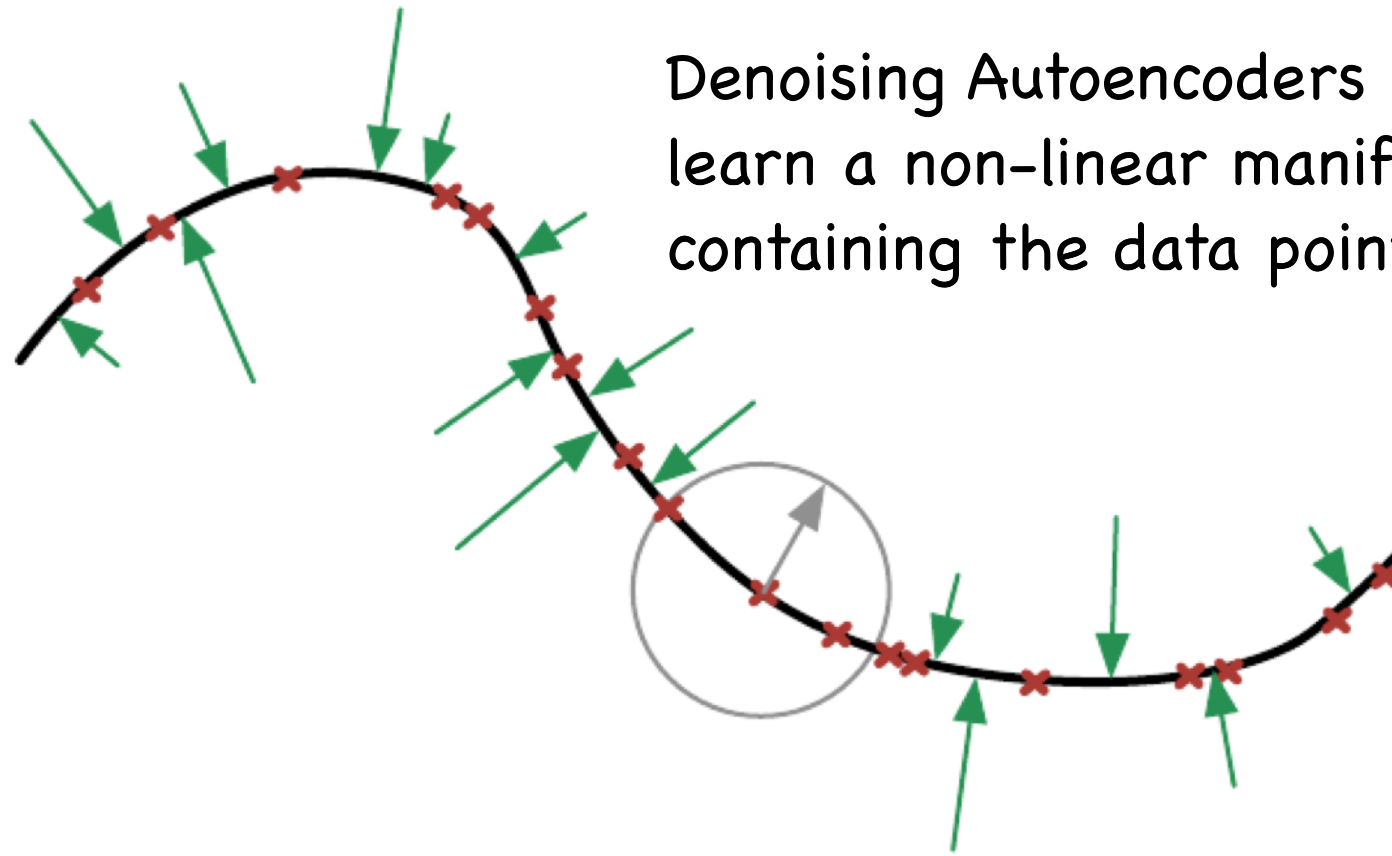
Sparse autoencoders

- higher capacity on the hidden layer
- regularise by L1 penalty on the hidden layer.



Denoising Autoencoders

Denoising autoencoders:
perturb the input and learn a noise correcting map.
Minimise the loss L(x,g(f(x*))), where x*= x + noise.



Denoising Autoencoders learn a non-linear manifold containing the data points.

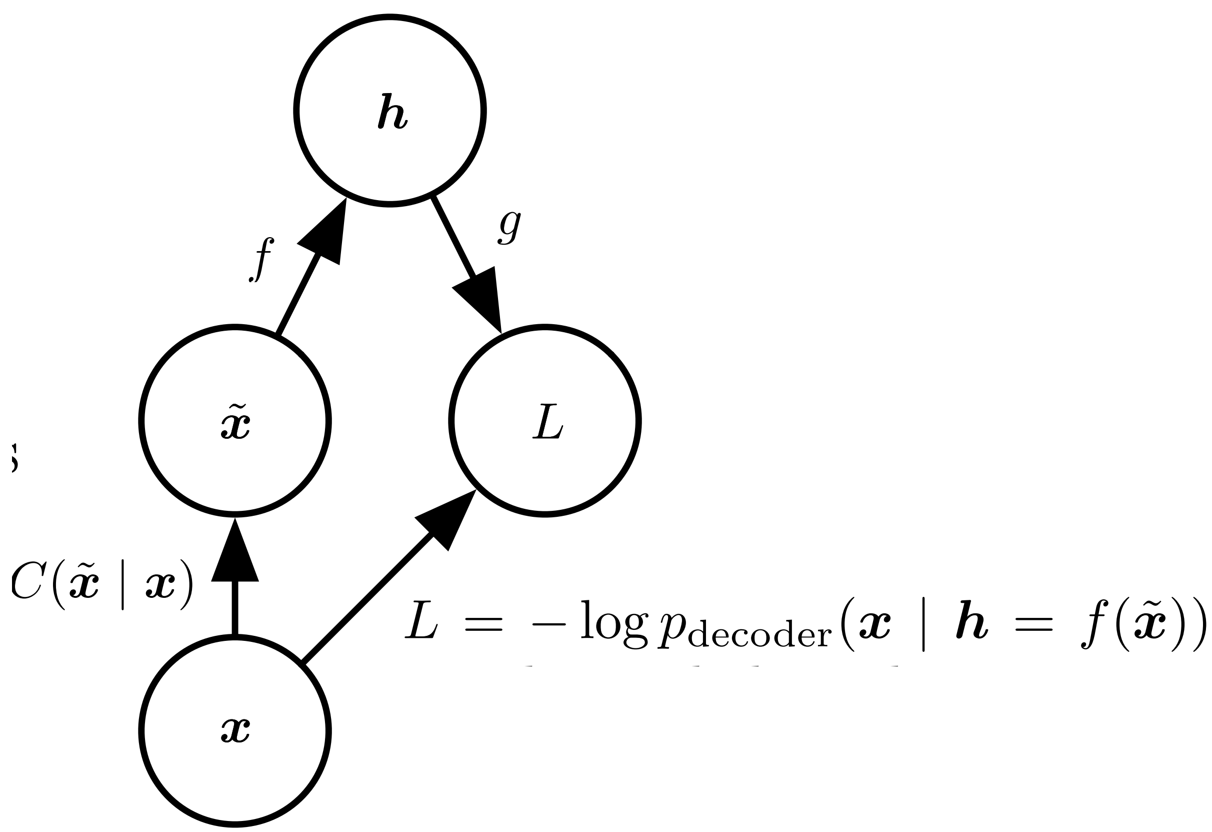
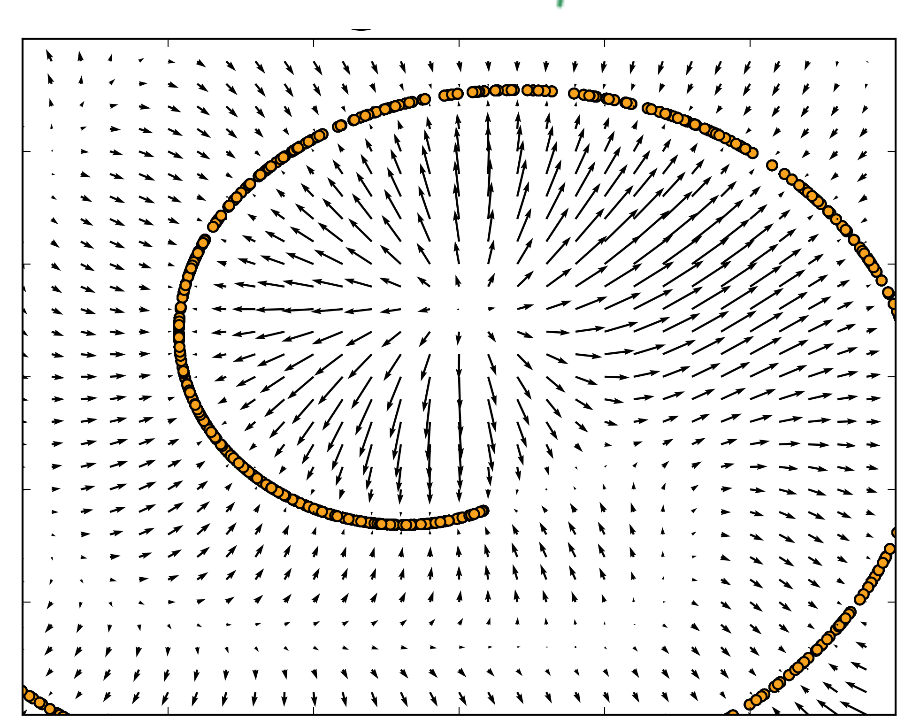
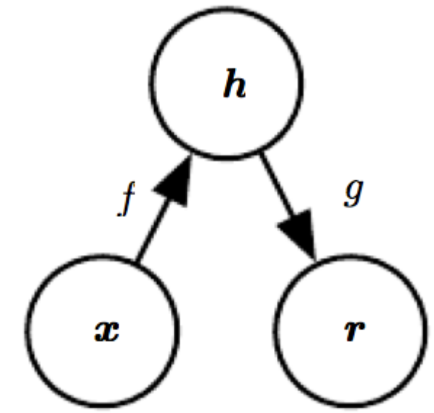


Figure 14.3

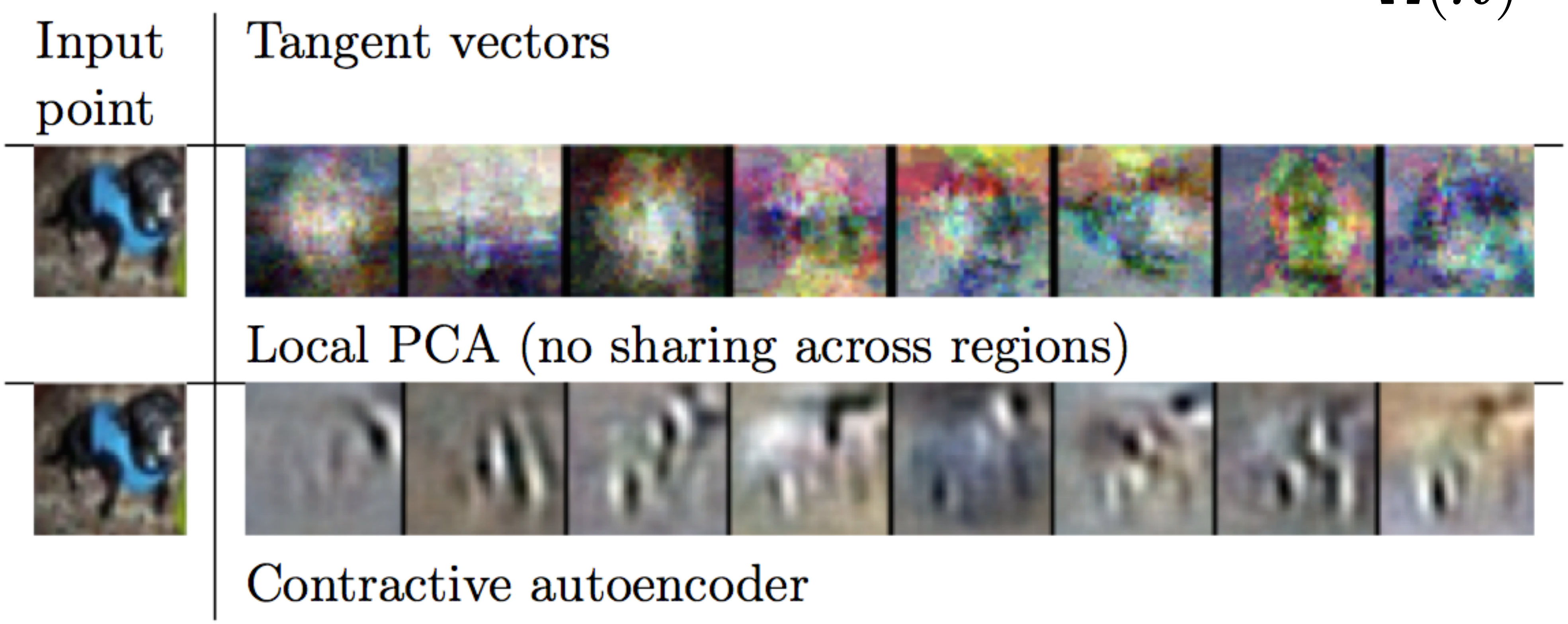


Contractive Autoencoders

Contractive autoencoders
penalise with Frobenis norm of the Jacobian of h.



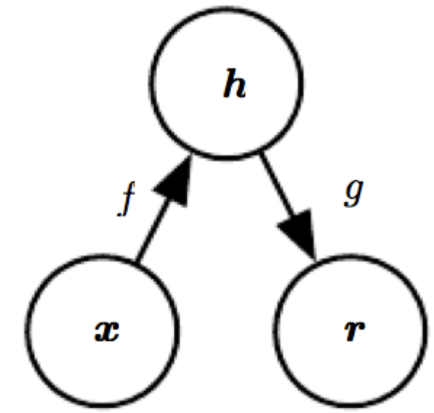
$$\Omega(\boldsymbol{h}) = \lambda \left\| \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} \right\|_F^2 .$$



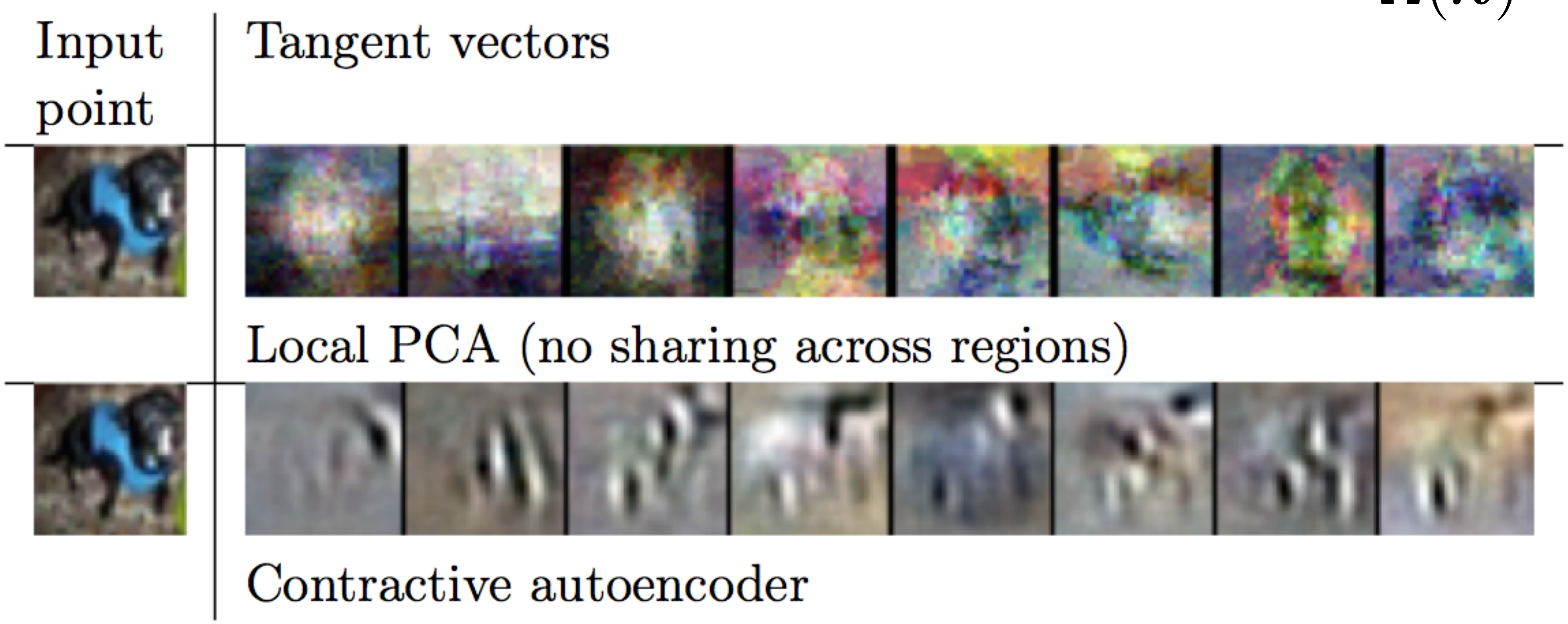
CAE also learn a non-linear manifold containing the data points.

Contractive Autoencoders

Contractive autoencoders
penalise with Frobenis norm of the Jacobian of h.



$$\Omega(\boldsymbol{h}) = \lambda \left\| \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} \right\|_F^2 .$$



CAE also learn a non-linear manifold containing the data points.

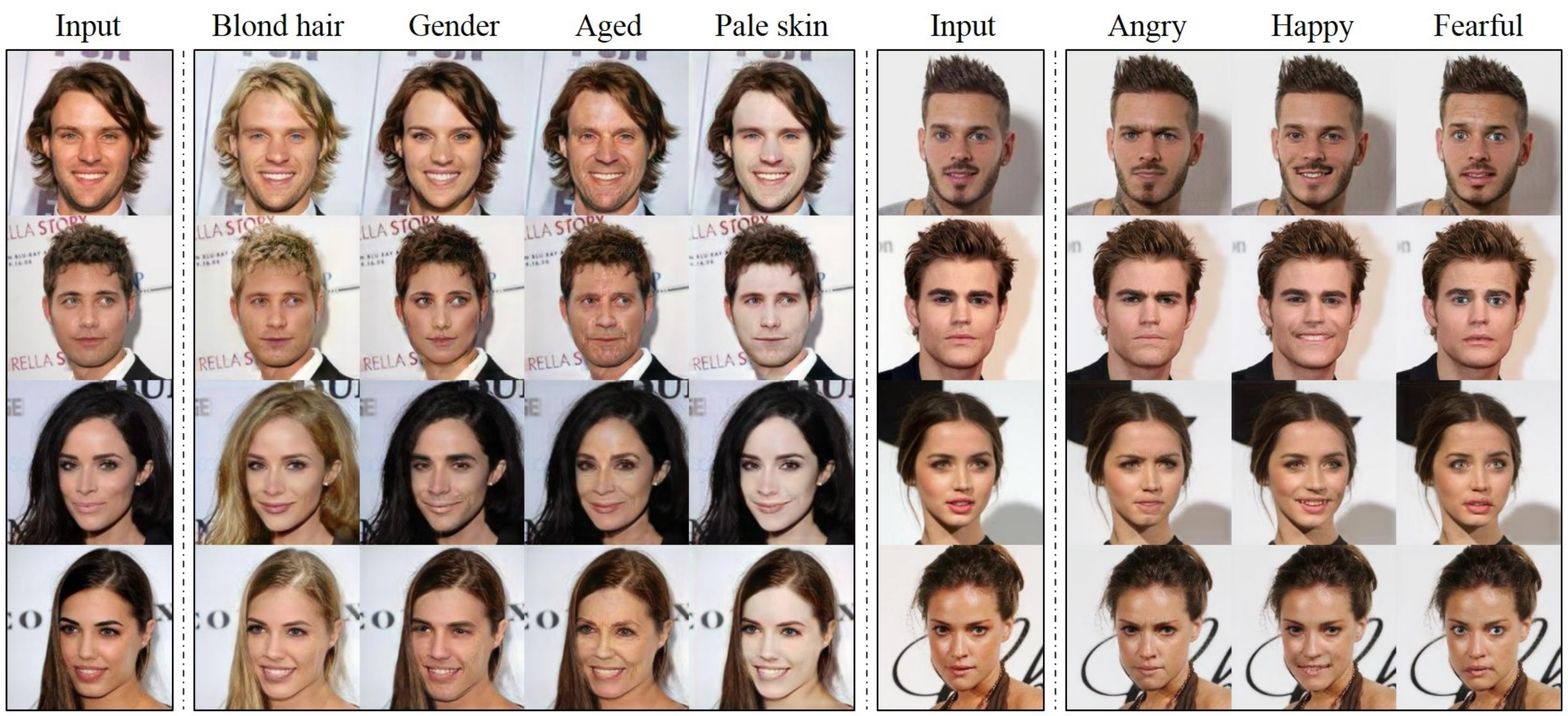
Generative Models

Goal is to learn a generative distribution p(x) of the input data.

We can learn also the joint input/output p(x,y);

we may want to learn the conditioning distribution w.r.t the class: p(x|y).

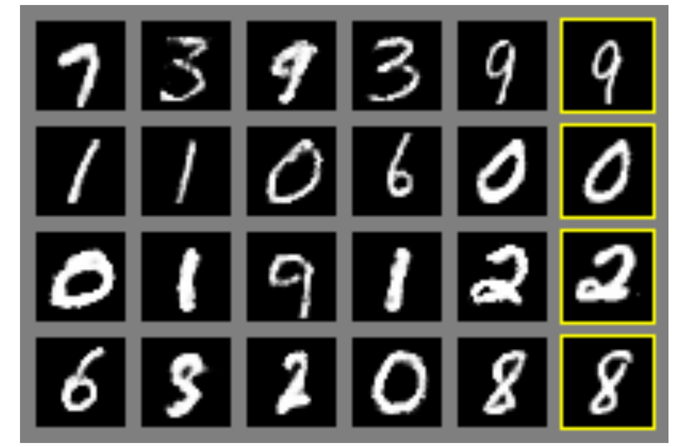
Or also p(x'|x"), conditioning on some input features x".



Many approaches using DNN:

- Boltzmann machines
- Variational Autoencoders
- GAN

Generative Adversarial Networks



Main Idea

Model learning as a two player game:

- a generator which generates new data points, starting from a random latent seed
- a discriminator that tries to distinguish real inputs from generated ones

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

D(x): probability of being authentic
G(z): transformed latent variable z

Goals: for G is to deceive D,
for D is to discriminate correctly.
Both are DNNs

Training alternates are SGD steps for G and one step for D.

Boltzmann Machines

- Introduced by Ackley *et al.* (1985)

- General “connectionist” approach to learning arbitrary probability distributions over binary vectors

- Energy model:

$$p(x) = \frac{\exp(-E(x))}{Z}$$

- Boltzmann machine: special case of energy model with

$$E(x) = -x^T U x - b^T x$$

where U is the weight matrix and b is the bias parameter

Limitation: they encode only linear dependency among variables.

Boltzmann Machines (with latent variables)

- Some variables are not observed

$$x = (x_v, x_h), \qquad x_v \text{ visible, } x_h \text{ hidden}$$

$$E(x) = -x_v^T R x_v - x_v^T W x_h - x_h^T S x_h - b^T x_v - c^T x_h$$

- Universal approximator of probability mass functions

- Suppose we are given data $X = (x_v^1, x_v^2, \dots, x_v^n)$
- Maximum likelihood is to maximize

$$\log p(X) = \sum_i \log p(x_v^i)$$

where

$$p(x_v) = \sum_{x_h} p(x_v, x_h) = \sum_{x_h} \frac{1}{Z} \exp(-E(x_v, x_h))$$

- $Z = \sum \exp(-E(x_v, x_h))$: partition function, difficult to compute

Restricted Boltzmann Machines

- Invented under the name *harmonium* (Smolensky, 1986)
- Popularized by Hinton and collaborators to *Restricted Boltzmann machine*
- Special case of Boltzmann machine with latent variables:

$$p(v,h) = \frac{\exp(-E(v,h))}{Z}$$

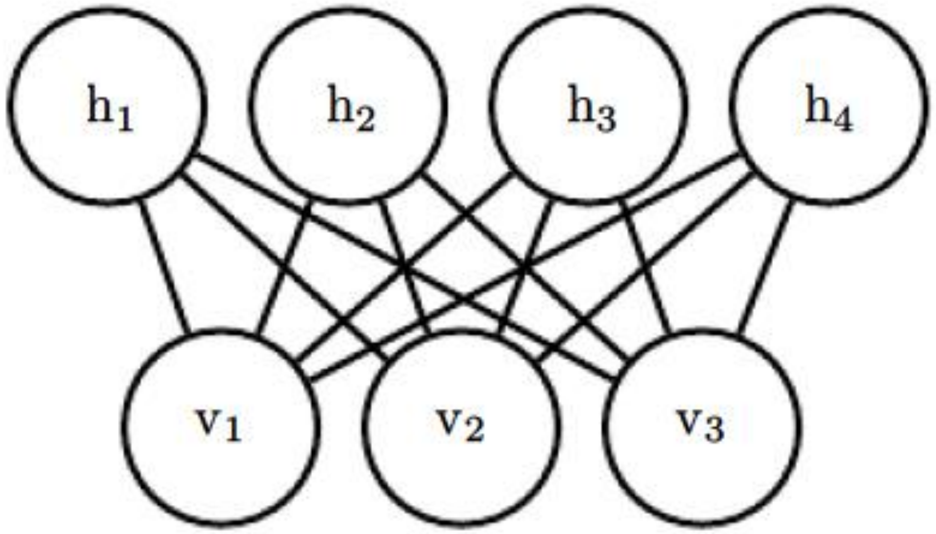
  where the energy function is

$$E(v,h) = -v^T W h - b^T v - c^T h$$

  with the weight matrix W and the bias b, c

- Partition function

$$Z = \sum_v \sum_h \exp(-E(v,h))$$



Restricted Boltzmann Machines

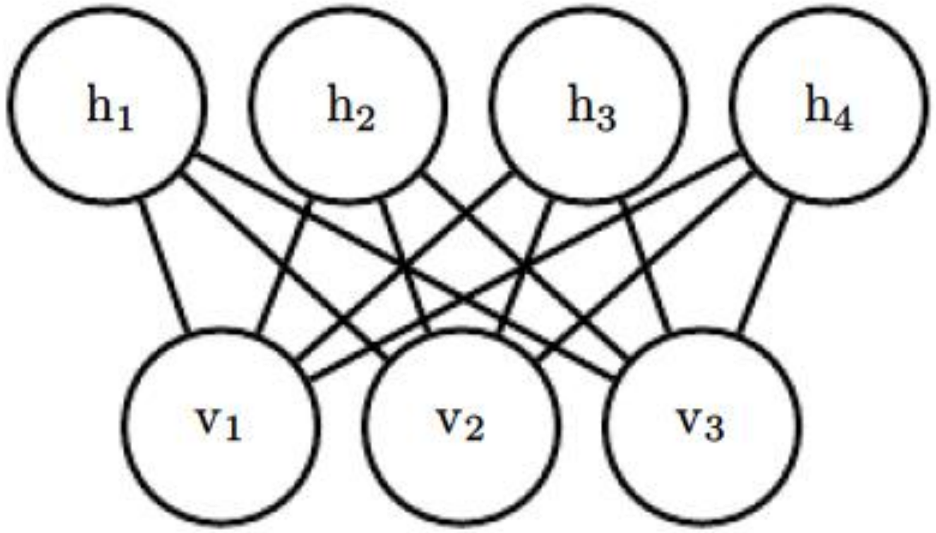
- Conditional distribution is factorial

$$p(h|v) = \frac{p(v,h)}{p(v)} = \prod_j p(h_j|v)$$

and

$$p\left(h_j = 1|v\right) = \sigma\left(c_j + v^T W_{:,j}\right)$$

is logistic function



- Similarly,

$$p(v|h) = \frac{p(v,h)}{p(h)} = \prod_i p(v_i|h)$$

and

$$p(v_i = 1|h) = \sigma\left(b_i + W_{i,:}h\right)$$

is logistic function

Deep Boltzmann Machines

- Special case of energy model. Take 3 hidden layers and ignore bias:

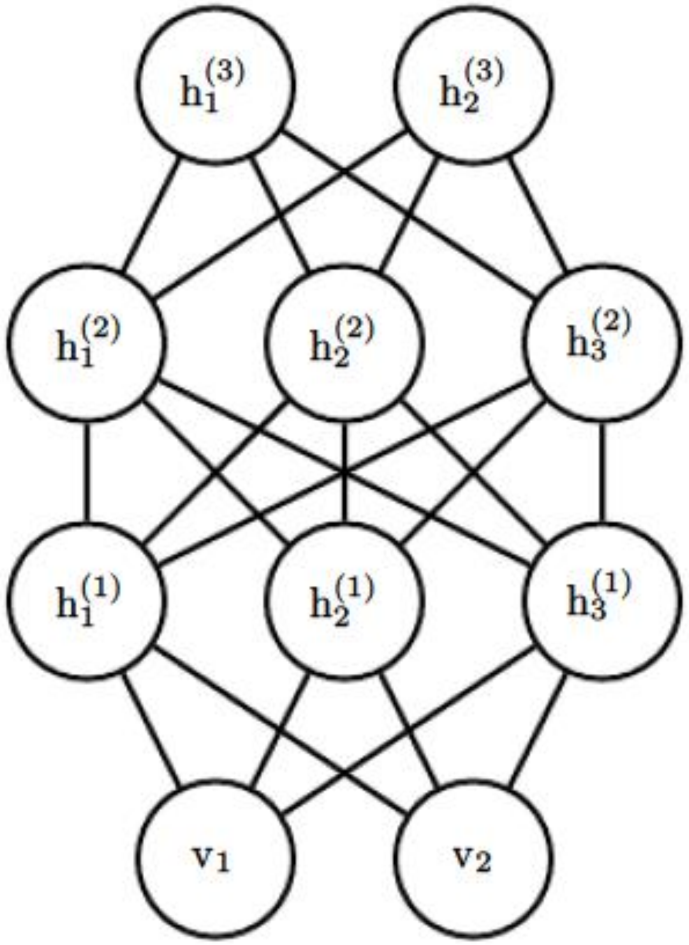
$$p(v,h^1,h^2,h^3) = \frac{\exp(-E(v,h^1,h^2,h^3))}{Z}$$

- Energy function

$$E(v,h^1,h^2,h^3) = -v^T W^1 h^1 - (h^1)^T W^2 h^2 - (h^2)^T W^3 h^3$$

  with the weight matrices W^1, W^2, W^3
- Partition function

$$Z = \sum_{v,h^1,h^2,h^3} \exp(-E(v,h^1,h^2,h^3))$$



Thanks for your attention!!!

Thanks for your attention!!!