Chapter 5

Deep Generative Models

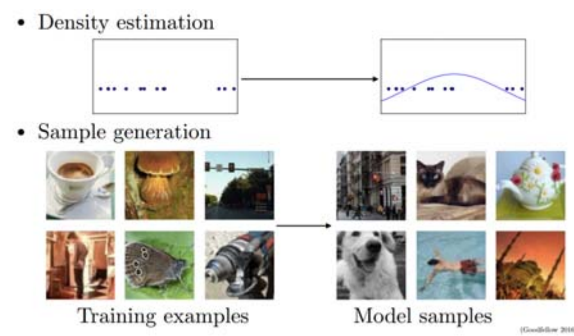
Qiang Ji

Introduction

- Discriminative models: map high-dimensional, rich sensory input x to a class label y, i.e., p(y|x) for classification . Learning is supervised. CNN is a discriminative model.
- Generative models: represent joint probability distributions over input x, i.e., p(x). No output labels are needed and learning is unsupervised

Table of Contents

Generative Modeling



Why study generative models

- Provide a powerful means to represent data concisely
- Allow to produce realistic synthetic data for visualization or to train discriminative models
- Allow to restore or reconstruct data from incomplete input data or infer data from a different modality, given one modality

Maximum Likelihood Learning

- Objective: marginal (log-)likelihood of observations

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, P_g(D|\theta) = \underset{\theta}{\operatorname{argmax}}\, \log P_g(D|\theta)$$

- Notation:
  - Observed data: $D = \{x_i\}$
  - Generative Model distribution: $P_g(x|\theta)$
  - True data distribution: $P_d(x)$

Generative Model Learning

Given training data **D**, learn a generative model p(x|Θ) that best capture p(**D**) by minimizing

- the KL-divergence between p(x|Θ) and p(**D**) – maximum likelihood learning
- the Jenson divergence between p(x|Θ) and p(**D**)- adversarial learning
- the reconstruction errors - auto-encoder

Maximum Likelihood Learning

- Maximum Likelihood learning is equivalent to minimizing Kullback–Leibler divergence (KLD) from model $P_g(x|\theta)$ to $P_d(x)$
- Derivation of the equivalence

$$\begin{aligned} KLD(P_d(x)||P_g(x|\theta)) &= E_{P_d(x)}[\log \frac{P_d(x)}{P_g(x|\theta)}] \\ &= \int_x P_d(x)\log P_d(x)dx - \int_x P_d(x)\log P_g(x|\theta)dx \\ &= -H(P_d(x)) - \int_x P_d(x)\log P_g(x|\theta)dx \\ &\approx C - \frac{1}{N}\sum_{i=1}^{N}\log P_g(x_i|\theta) \end{aligned}$$

  - where $C = -H(P_d(x))$ is a constant w.r.t. θ
  - The last term is average log-likelihood of observations
  - We see minimizing KLD is equivalent to ML

Adversarial Learning

- Objective:

$$\{\theta^*, \phi^*\} = \min_{\theta} \max_{\phi} E_{x\sim P_d(x)}[\log y(x|\phi)] + E_{x\sim P_g(x|\theta)}[\log(1 - y(x|\phi))]$$

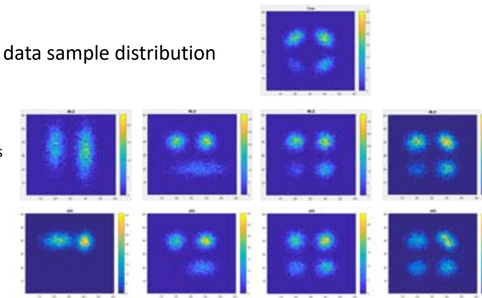
- Notation:
  - Generative Model distribution: $P_g(x|\theta)$
  - True data distribution: $P_d(x)$
  - Discriminative Model output: $y(x|\phi)$
    - This indicates the probability of x being a real data sample

ML Learning v.s. Adversarial Learning

- Generative Model: a HMM or MoG
  - True data sample distribution

ML samples

AL samples



Hidden state #: 2 3 4 5

Adversarial Learning

- Adversarial learning (AL) is equivalent to minimizing Jensen-Shannon divergence (JSD) between model $P_g(x|\theta)$ and $P_d(x)$

- Equivalence of AL and minimizing JSD:

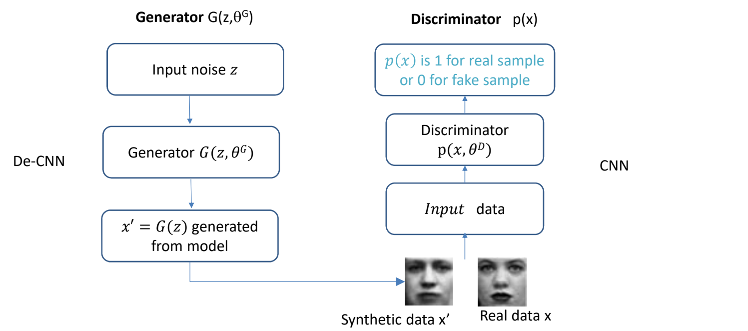
$$\begin{aligned} JSD(P_d\|P_g) &\triangleq \frac{1}{2}KL(P_d\|\frac{P_d+P_g}{2}) + \frac{1}{2}KL(P_g\|\frac{P_d+P_g}{2}) \\ &= -\log 2 + \frac{1}{2}(KL(P_d\|P_d+P_g) + KL(P_g\|P_d+P_g)) \\ &= -\log 2 + \frac{1}{2}(E_{x\sim P_d}[\log\frac{P_d}{P_d+P_g}] + E_{x\sim P_g}[\log\frac{P_g}{P_d+P_g}]) \\ &= -\log 2 + \frac{1}{2}(E_{x\sim P_d}[\log f(x)] + E_{x\sim P_g}[\log(1-f(x))]) \end{aligned}$$

  - where we use the notation, $f(x) = \frac{P_d(x)}{P_d(x) + P_g(x)} = y(x|\phi^*)$, i.e., the discriminator

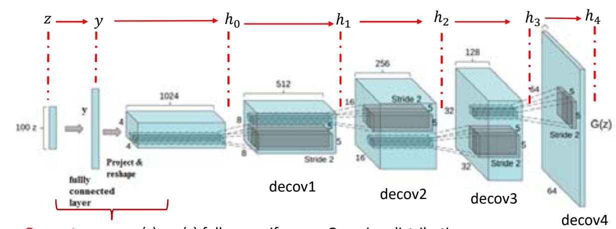
Generative Adversarial Nets (GANs)

- a generative model that approximates the underlying probability distribution of the input data by minimizing the Jensen divergence
- it is implemented a de-convolutional neural network plus a probabilistic input
- It is trained by competing with a discriminative model

GAN Framework



Generator Network



Generate z : z ~ p(z) - p(z) follows uniform or Gaussian distribution

Fully-connected layer: **y**$=ReLU(w^T z + b)$

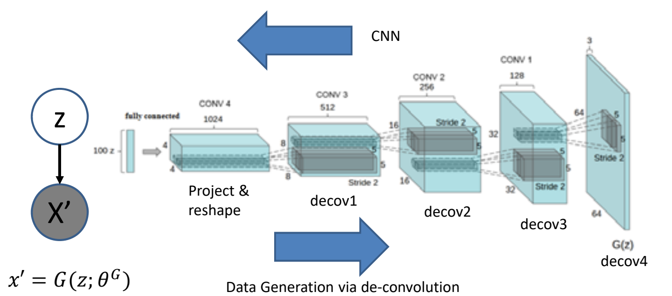
Reshape: $h_0 = Reshape(\boldsymbol{y})$

h_0 can be considered as a volume with 1024 feature maps, each map is of size 4x4.

For this architecture:

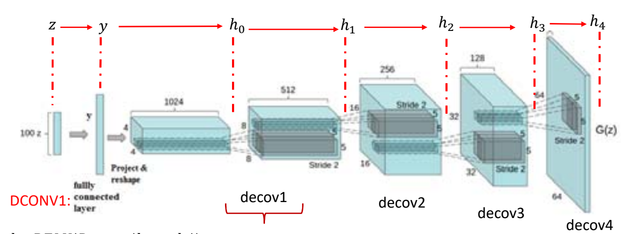
$$z \in R^{100\times1}, y \in R^{16484\times1}, w \in R^{100\times16484}, b \in R^{16384x1}, h_0 \in R^{4\times4\times1024},$$

Generator Network



DCGAN Radford et al 2015

Generator Network



DCONV1:

$h_1=RELU(Deconv(h_0;w_0,b_0))$

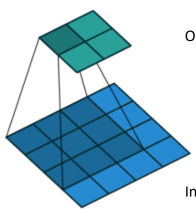
Deconvolve h_0 with 512 filters of size 5x5x1024 with stride 1, followed by ReLU activation

$w_0 \in R^{5\times5\times1024\times512}, b_0 \in R^{8x8\ x\ 512}$

No pooling, feature map size is doubled to 8x8.

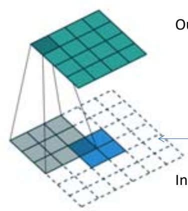
Input dimension: M
Filter Dimension K
Stride size: s
Output dimension:
N=(M-1)*s +K

Deconvolution --- Visualization



Input dimension: M
Filter Dimension K
Stride size: s
Output dimension:
N=(M-K)/s +1

Convolution with a 3x3 filter



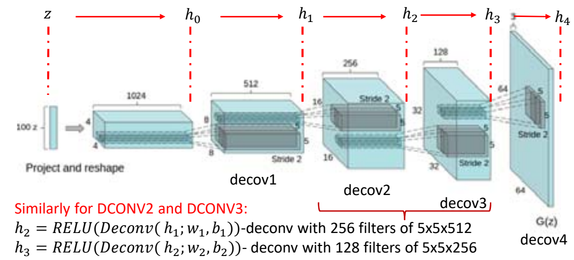
Deconvolution with 3x3 filter

Input dimension: M
Filter Dimension K
Stride size: s
Output dimension:
N=(M-1)*s +K

Blue is input
Green is output

Gif images from https://github.com/vdumoulin/conv_arithmetic

Generator Network



Similarly for DCONV2 and DCONV3:

$h_2 = RELU(Deconv(h_1; w_1, b_1))$-deconv with 256 filters of 5x5x512

$h_3 = RELU(Deconv(h_2; w_2, b_2))$- deconv with 128 filters of 5x5x256

Input dimension: M
Filter Dimension K
Stride size: s
Output dimension:
N=(M-1)*s +K

$w_1 \in R^{5\times5\times512\times256}, b_1 \in R^{16x16x256}$.

$w_2 \in R^{5\times5\times256\times128}, b_2 \in R^{32x32x128}$.

Finally, we have a 128 feature maps of size 32x32

Deconvolution --- Computation

- Consider filter $W \in R^{3\times3}, input\ X = R^{2\times2}, output\ Y = R^{4\times4}$
- X and Y are reshaped to a column vector (**X** and **Y**) in following matrix product.

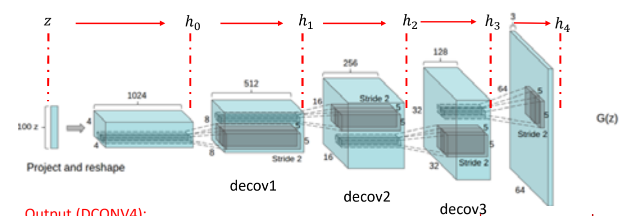
– Convolution: $Y \rightarrow X$, $\boldsymbol{X} = \mathrm{Conv(Y, W)} = \mathbf{CY}$

Deconvolution is also called transposed convolution

– Deconvolution: $\mathrm{X} \rightarrow \mathrm{Y}$, $\boldsymbol{Y} = \mathrm{Deconv(X, W)} = \mathrm{C^T}\mathbf{X}$

- $\mathrm{C} \in R^{4\times16}$ is the induced matrix from W
- $$C = \begin{pmatrix} w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 \\ 0 & 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} \end{pmatrix}$$

Generator Network



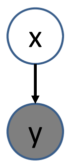
Output (DCONV4):

$h_4 = Deconv(h_3; w_3, h_3)$-deconv with 3 filters of size 5x5x128

$w_3 \in R^{5\times5\times128\times3}, b_3 \in R^{64x64x3}$.

decov4

Discriminator Network



$y = p(x; \theta^D)$

- Binary classification, conventional CNN architecture.
- Real data samples with label y = 1
- Model samples $x' = G(z; \theta^G)$ with label y = 0

Break down

- Discriminator $y = p(x; \theta^D) = D(x; \theta^D)$: the probability of the input sample to be real.
- Solve discriminator D: given m pairs of real and fake samples $\{x_i, x_i'\}$

$$\theta^{D*} = \arg\max_{\theta^D} \frac{1}{m}\sum_{i=1}^{m} \log p(x_i; \theta^D) + \log\left(1 - p(x_i'; \theta^D)\right)$$

- Solve θ^G for generator: minimize the probability of fake samples of being fake
  - $\theta^{G*} = \arg\min_{\theta^G} \frac{1}{m}\sum_{i=1}^{m} \log(1 - p(G(z; \theta^G); \theta^D))$

Model learning

- Learn the generator network parameter θ^G and discriminator network parameter θ^D simultaneously.
- Minimax Game
  - Simultaneously update discriminator and generator network
  - Discriminator tries to distinguish real data samples (x) from model (fake) samples (x')
  - Genetator tries to generate realistic synthetic sample (x')

Learning algrithm

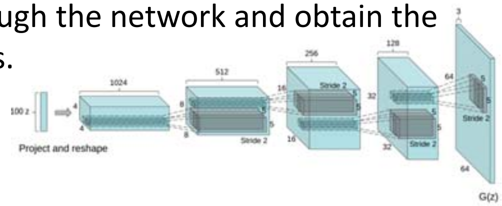
- For number of iterations:
  - Sample m noise vectors $\{z\}_{i=1}^{m}$, and m generated samples $\{x'\}_{i=1}^{m}$, where $x'_i = G(z_i; \theta^G)$.
  - Sample m data samples $\{x\}_{i=1}^{m}$.
  - Update Discriminator by the stochastic gradient ascent

$$\theta^D \leftarrow \theta^D + \lambda^D \frac{\partial \frac{1}{m}\sum_{i=1}^{m} \log p(x_i; \theta^D) + \log\left(1 - p(x'_i; \theta^D)\right)}{\partial \theta^D}$$

  - Update generator by ascending the stochastic gradient
  - $\theta^G \leftarrow \theta^G + \lambda^G \frac{\partial \frac{1}{m}\sum_{i=1}^{m} \log p\left(G\left(z_i; \theta^G\right); \theta^D\right)}{\partial \theta^G}$

Inference

- Inference is to generate samples given an input noise vector
- It is a straightforward process by feeding the input vector through the network and obtain the genrated samples.



Examples --- Generate realistic images

Minibatch GAN on CIFAR

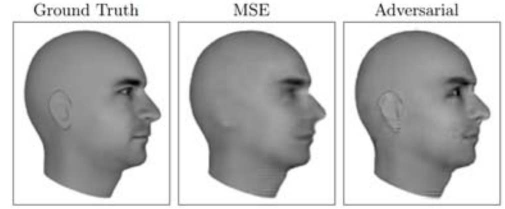
(Salimans et al 2016)

Examples --- Generate realistic images

DCGANs for LSUN Bedrooms

(Radford et al 2015)

Example --- Adversarial loss to produce sharp images



(Lotter et al 2016)

Next frame prediction

Traditional using MSE loss:

$$L_G^{MSE} = \left|\left|f(I^{1:t};\theta) - I^{t+1}\right|\right|^2$$

Adding adversarial loss:

$$L_G^{AL} = \log\Big(1 - p\big(G(I^{1:t};\theta^G)\big)\Big)$$

Total loss:

$$L_{\mathrm{G}} = \mathrm{L}_{\mathrm{G}}^{\mathrm{MSE}} + \lambda L_G^{AL}$$

The output has multiple possible solutions:

- MSE averages all possible output and results in blurry image
- AL knows there are multiple solutions, and each one is sharp and realistic.

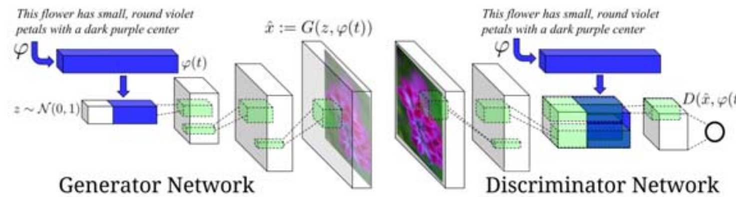
Example --- Adversarial loss to produce sharp images

(Ledig et al 2016)

Deep residual network with MSE loss

Deep residual network with adversarial loss

Conditional GAN --- Generate meaningful images



Conditional GAN allows us to generate meaningful images

To encode conditions:

- Concatenate conditions with random noise in Generator
- Concatenate conditions with feature maps in Discriminator

Video demos

- GAN Demo: Training to synthesize newspaper pages
  https://www.youtube.com/watch?v=lJoZR3GnHqY

- GAN Demo: Training to synthesize faces
  https://www.youtube.com/watch?v=Co2ukCewKkE

Examples --- Generate meaningful images

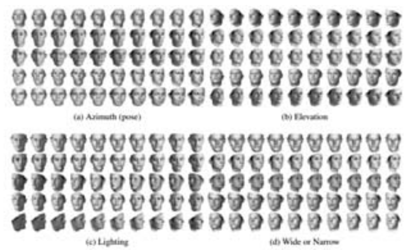
Text to Image Reed et al 2016

InfoGAN --- Generate meaningful images

Use interpretable latent nodes to control data generation

Decompose random noise z to two parts:
- Latent codes : $z_1 - z_3$
- Original noise z_4- z_{100}

Leverage on information theory to learn the latent meaningful codes.



InfoGAN (Chen et al 2016)

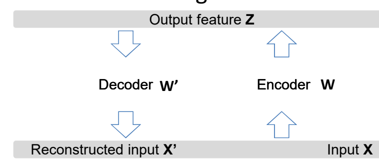
(Goodfellow 2016)

Conclusions

- GAN is a generative model for generating realistic synthetic images
- It combines a feed forward NN with a noise input vector to model the underlying data distribution
- It employs an adversarial training strategy that involves the competition between the GAN and a discriminative model to produce the GAN that maximally confuses the discriminative model
- Compared to the traditional learning criteria such as MSE or negative log-likelihood, GAN can produce much sharper photo-realistic images
- Recent GAN extensions allow GAN to produce images with meaningful contexts.

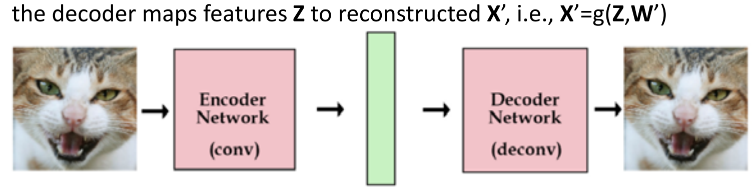
AutoEncoders and Decoders

- An autoencoder is an unsupervised learning method to train a model
- The trained model can be used for data representation, dimensionality reduction or un-supervised feature learning
- It consists of two parts: an encoder (e.g. the NN) and a decoder
- Decoder can be used for data generation



AutoEncoders and Decoders (cont'd)

- The encoder is parameterized by **W** and through its mapping function f(), the encoder maps input **X** in [0,1] to output feature **Z,** i.e., Z=f(**X**,**W**)
- The decoder is parameterized by **W'** and through its mapping function g(), the decoder maps features **Z** to reconstructed **X'**, i.e., **X'**=g(**Z**,**W'**)



- The simplest encoder function is $\mathbf{Z}=f(\mathbf{X},\mathbf{W})=\sigma(\mathbf{W}^t\mathbf{X}+\mathbf{b})$
- The simplest decoder is $\mathbf{X}'=g(\mathbf{Z},\mathbf{W}')=\sigma(\mathbf{W}'^t\mathbf{Z}+\mathbf{b}')$, **W'** is often constrained to transpose of W.

AutoEncoder Learning

- Given training data D={$\mathbf{X}_i$}, i=1,2, .., N, autoencoder learning is to simultaneously learn **W** and **W**' by minimizing the total reconstruction errors, i.e.,

$$\mathbf{W}^*, \mathbf{W'}^* = \arg\min_{\mathbf{W},\mathbf{W'}} \sum_{i=1}^{N} (\mathbf{X}_i - \mathbf{X'}_i)^2 = \arg\min_{\mathbf{W},\mathbf{W'}} \sum_{i=1}^{N} (\mathbf{X}_i - g(f(\mathbf{X}_i, \mathbf{W}), \mathbf{W}'))^2$$

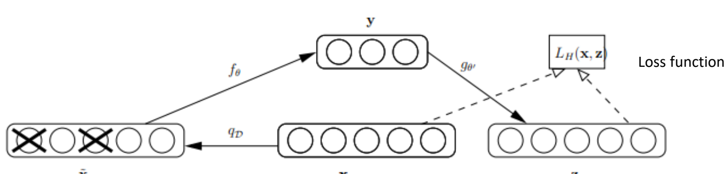
- Given the loss function, back propagation may be used for the training.

- Once trained, the encoder can be used to generate features, while the decoder may be used to generate synthetic data

Variants of AutoEncoders

1. Denoising Autoencoders

- Noise is added to the input data to learn the encoder, producing robust encoders that can better recover from noisy or corrupted inputs



Note : 1) x is perturbed randomly, and 2) two hidden layers y and z

Variants of AutoEncoders (cont'd)

2. Sparse autoencoders

- L1 norm is imposed on the output of the encoder to learn sparse features

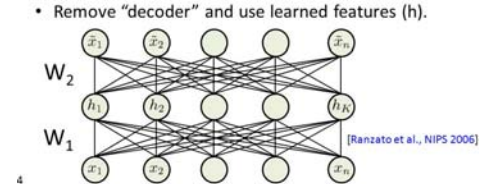
- Train two-layer neural network by minimizing:

$$\underset{W_1, W_2}{\text{minimize}} \sum_i ||W_2 h(W_1 x^{(i)}) - x^{(i)}||_2^2 + \lambda ||h(W_1 x^{(i)})||_1$$

$$h(z) = \mathrm{ReLu}(z)$$

Note bias is ignored.

- Remove "decoder" and use learned features (h).

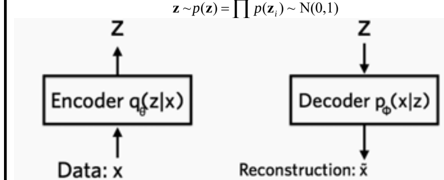


Variants of AutoEncoders (cont'd)

3. **Variational autoencoders** -one of the most popular unsupervised learning methods and is comparable to GANs

- The generative (decoder) model-p_ϕ (x,z) and the encoding model q_θ(z|x).

$$\mathbf{z} \sim p(\mathbf{z}) = \prod p(\mathbf{z}_i) \sim \mathrm{N}(0,1)$$



Data generation

1) $\mathbf{z} \sim \mathrm{N}(0, \mathrm{I})$
2) $\mathbf{x} \sim \mathrm{p}(\mathbf{x} \mid \mathbf{z}, \theta)$

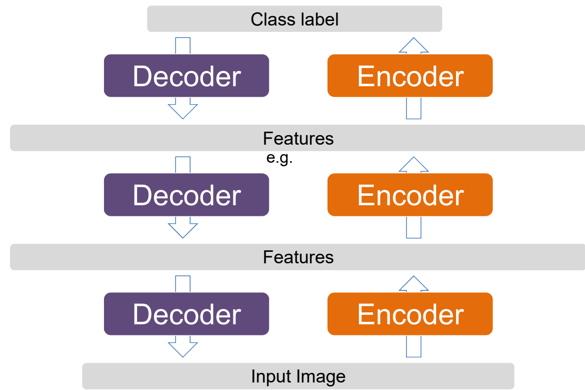
Generated fake faces

Training :

$$\varphi^*, \theta^* = \max p_\varphi(\mathbf{x}) = \max_{\theta,\varphi} \mathrm{E}_{q(z|x)}(p_\varphi(\mathbf{x} \mid \mathbf{z})) - KL(q_\theta(\mathbf{z} \mid \mathbf{x}) \,||\, p(\mathbf{z}))$$

The first term is expected reconstruction loss and the second term is KL divergence

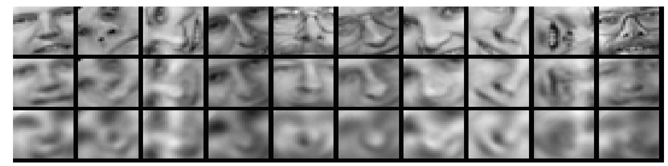
Stacked Auto-Encoders (Deep Autoencoders)



From http://cs.nyu.edu/~fergus/tutorials/deep_learning_cvpr12/

Image Reconstruction with Deep Autoencoders

•We used 25x25 – 2000 – 1000 – 500 – 30 autoencoder to extract 30-D real-valued codes for Olivetti face patches.



- **Top**: Random samples from the test dataset.
- **Middle**: Reconstructions by the 30-dimensional deep autoencoder.
- **BoHom**: Reconstructions by the 30-dimentinoal PCA.

Courtesy of Russ Salakhutdinov of CMU

Feature Learning, Generation, and Reconstruction