Deep Learning - 2019

Unsupervised Approaches

Prof. Avishek Anand

What we learnt so far…

- Supervised learning: discover patterns in the data that relate data attributes with a target (class) attribute.
  - These patterns are then utilized to predict the values of the target attribute in future data instances.

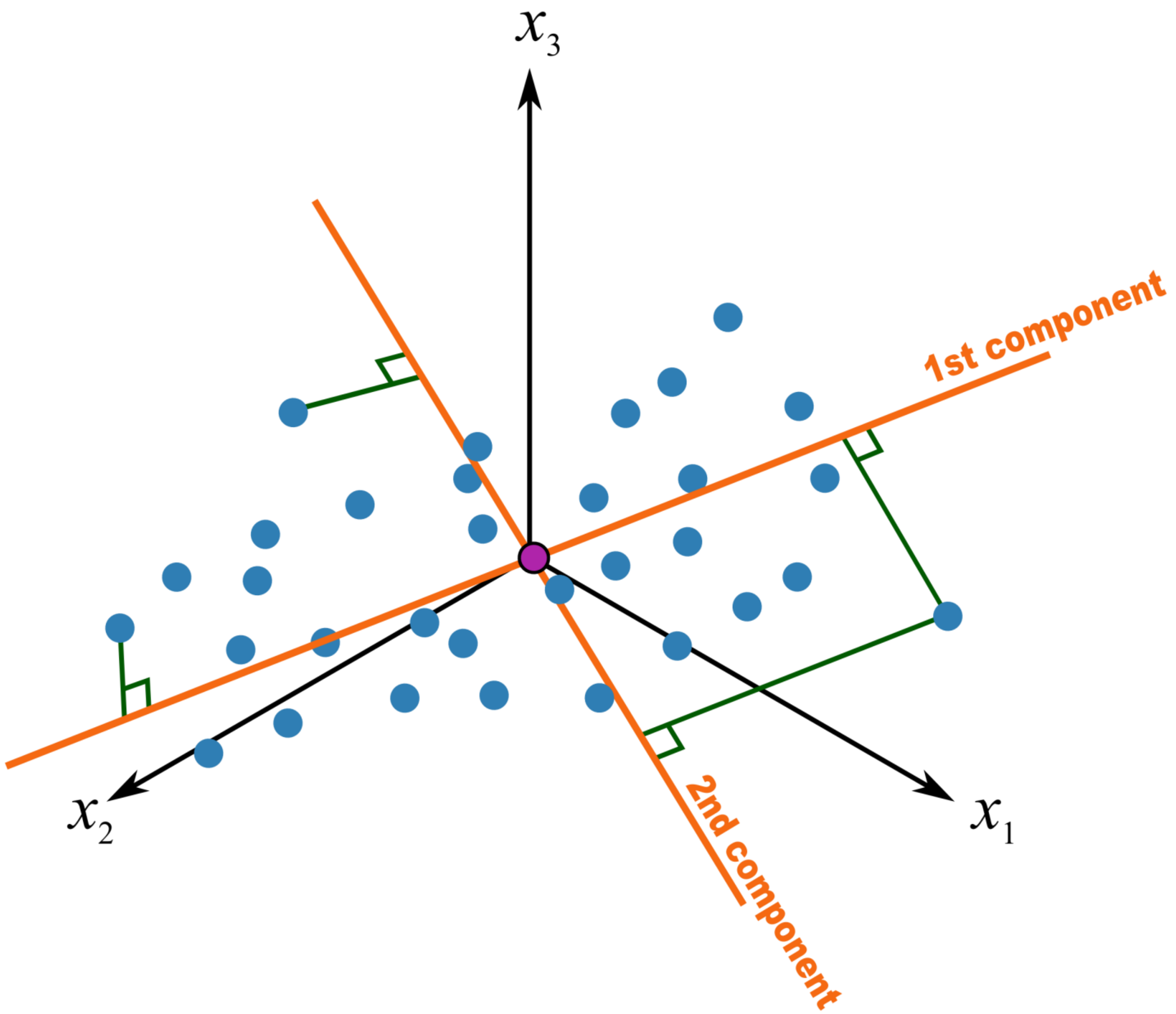
- Unsupervised learning: The data have no target attribute.
  - We want to explore the data to find some intrinsic structures in them.
  - Today we cover neural models that do dimensionality reduction, generative modelling

Principle Components Analysis

- Statistical approach for data compression and visualization
- Invented by Karl Pearson in 1901

- N-dimensional data and finds the M orthogonal directions in which the data have the most variance.

Aim of PCA: *Can we obtain another basis that is a linear combination of the original basis and that re-expresses the data optimally?*



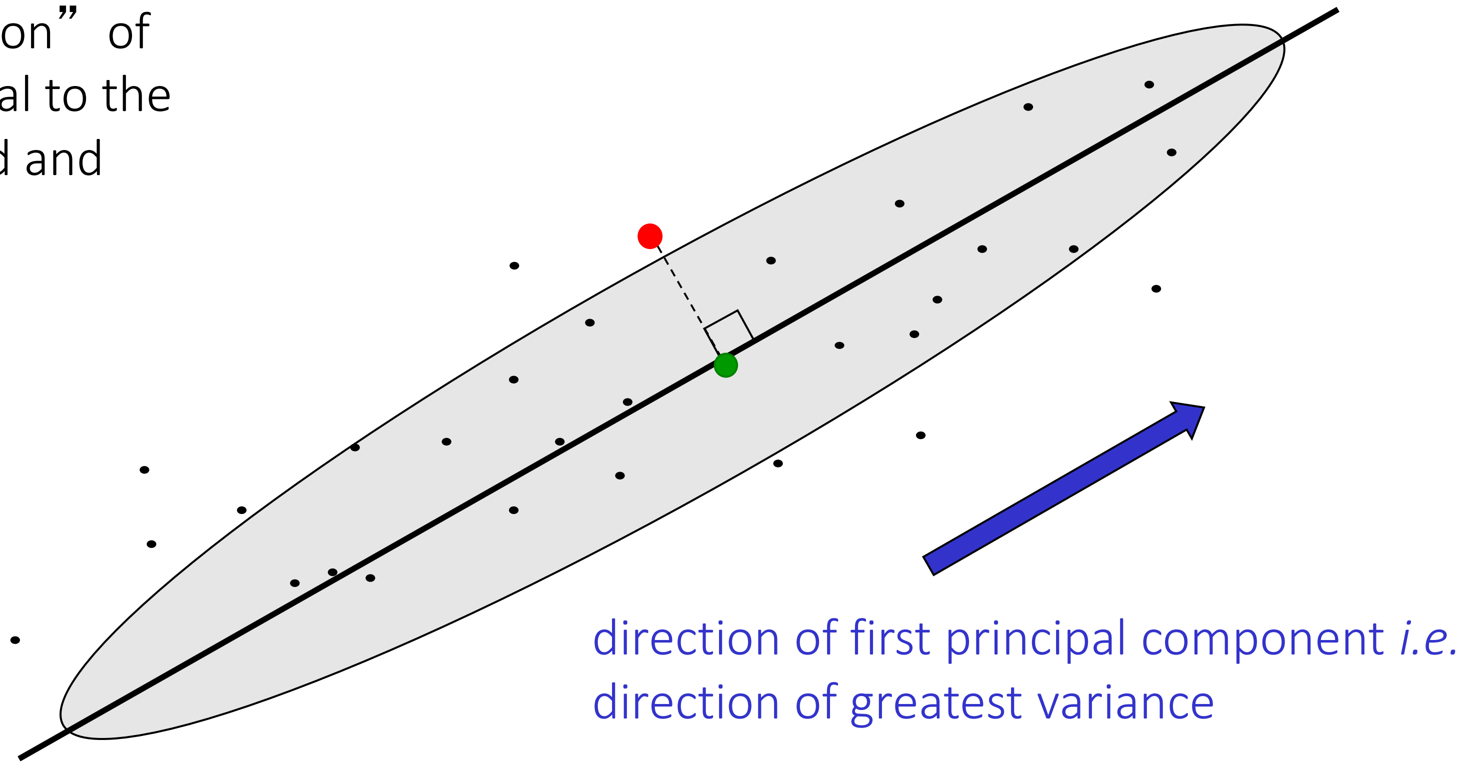
Principle Components Analysis (PCA)

Aim of PCA: *Can we obtain another basis that is a linear combination of the original basis and that re-expresses the data optimally?*

- PCA takes N-dimensional data and finds the M orthogonal directions in which the data have the most variance
  - These M principal directions form a lower-dimensional subspace.
  - Linear Transformation: We can represent an N-dimensional data point by its projections onto the M principal directions.
- We reconstruct by using the mean value (over all the data) on the N-M directions that are not represented
  - The reconstruction error is the sum over all these unrepresented directions of the squared differences of the data point from the mean.

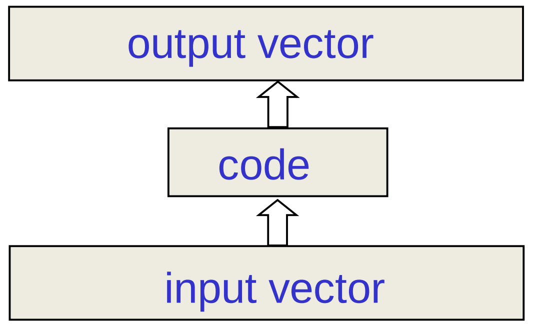
PCA

The red point is represented by the green point. The “reconstruction” of the red point has an error equal to the squared distance between red and green points.


direction of first principal component *i.e.* direction of greatest variance

Using NN to implement PCA

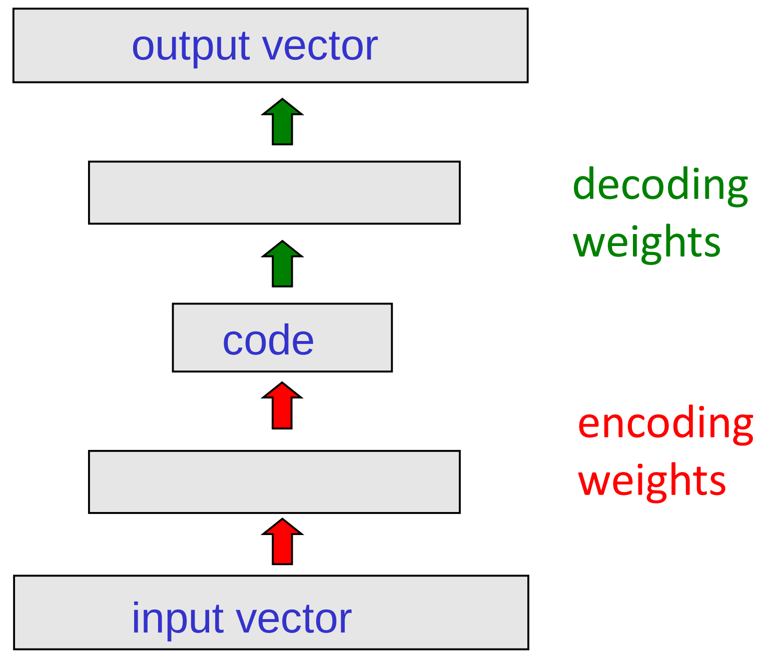
- Try to make the output be the same as the input in a network with a central bottleneck.
- The activities of the hidden units in the bottleneck form an efficient code.
- If the hidden and output layers are linear, it will learn hidden units that are a linear function of the data and minimize the squared reconstruction error.
  - This is exactly what PCA does.
- The M hidden units will span the same space as the first M components found by PCA
  - Their weight vectors may not be orthogonal.
  - They will tend to have equal variances.



Generalizing PCA

- With non-linear layers before and after the code, it should be possible to efficiently represent data that lies on or near a non-linear manifold.

  - The encoder converts coordinates in the input space to coordinates on the manifold.
  - The decoder does the inverse mapping.



Auto-encoders

- Given data x (no labels) we would like to learn the functions f (encoder) and g (decoder) where:

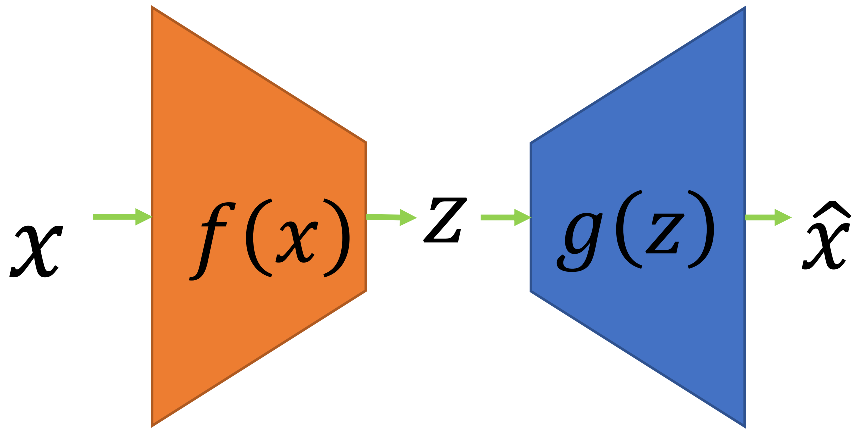
$$f(x) = s(wx + b) = z$$

and

$$g(z) = s(w'z + b') = \hat{x}$$

$$\text{s.t } h(x) = g\big(f(x)\big) = \hat{x}$$

where h is an **approximation** of the identity function.



(z is some **latent** representation or **code** and s is a non-linearity such as the sigmoid)

($\hat{x}$ is x's reconstruction)

Training the Auto-Encoder

Using **Gradient Descent** we can simply train the model as any other FC NN with:

- Traditionally with *squared error* loss function

$$L(x, \hat{x}) = \|x - \hat{x}\|^2$$

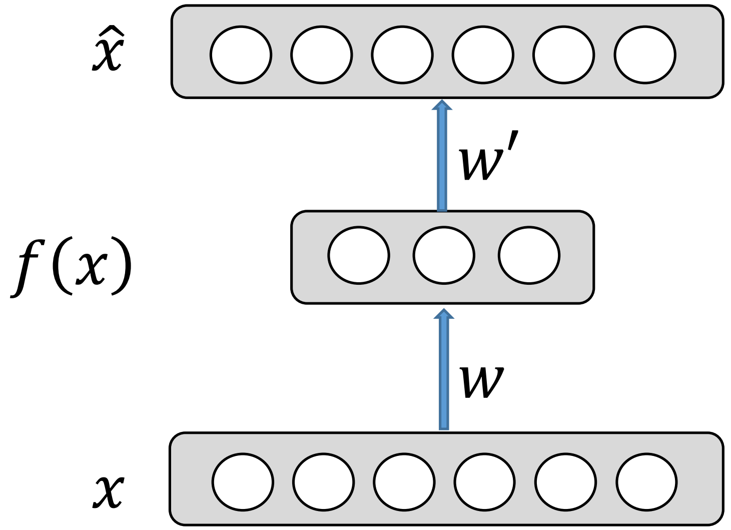
- If our input is interpreted as bit vectors or vectors of bit probabilities the *cross entropy* can be used

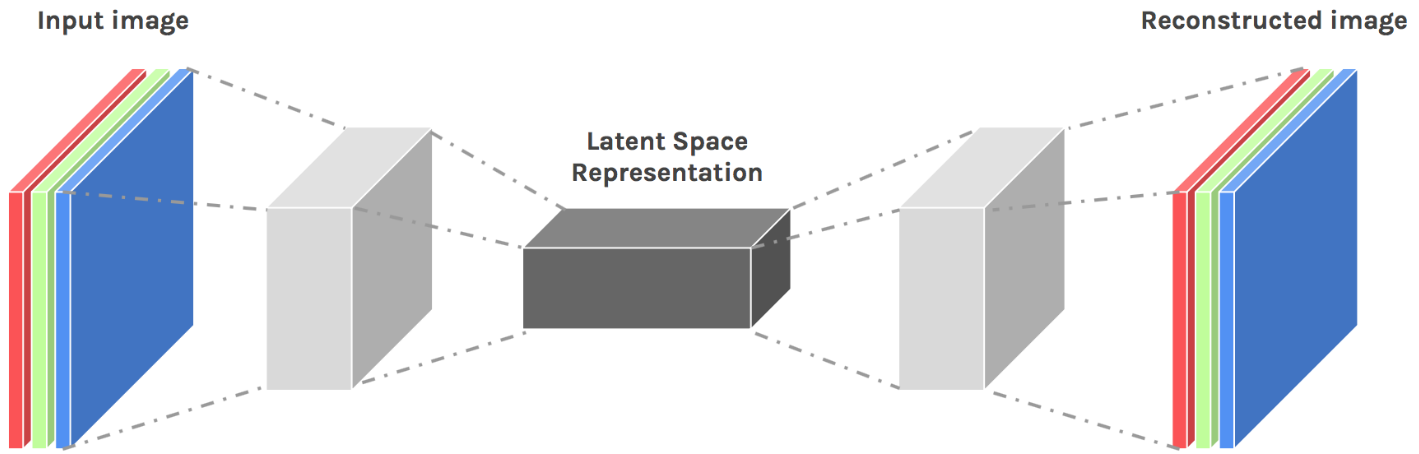
$$H(p, q) = -\sum_{x} p(x) \log q(x)$$

Under-Complete AE

- Hidden layer is **Undercomplete** if hidden layer is smaller than the input layer
  - Compresses the input
  - Compresses well only for the training dist.

- Hidden nodes will be
  - Good features for the training distribution.
  - Bad for other types on input





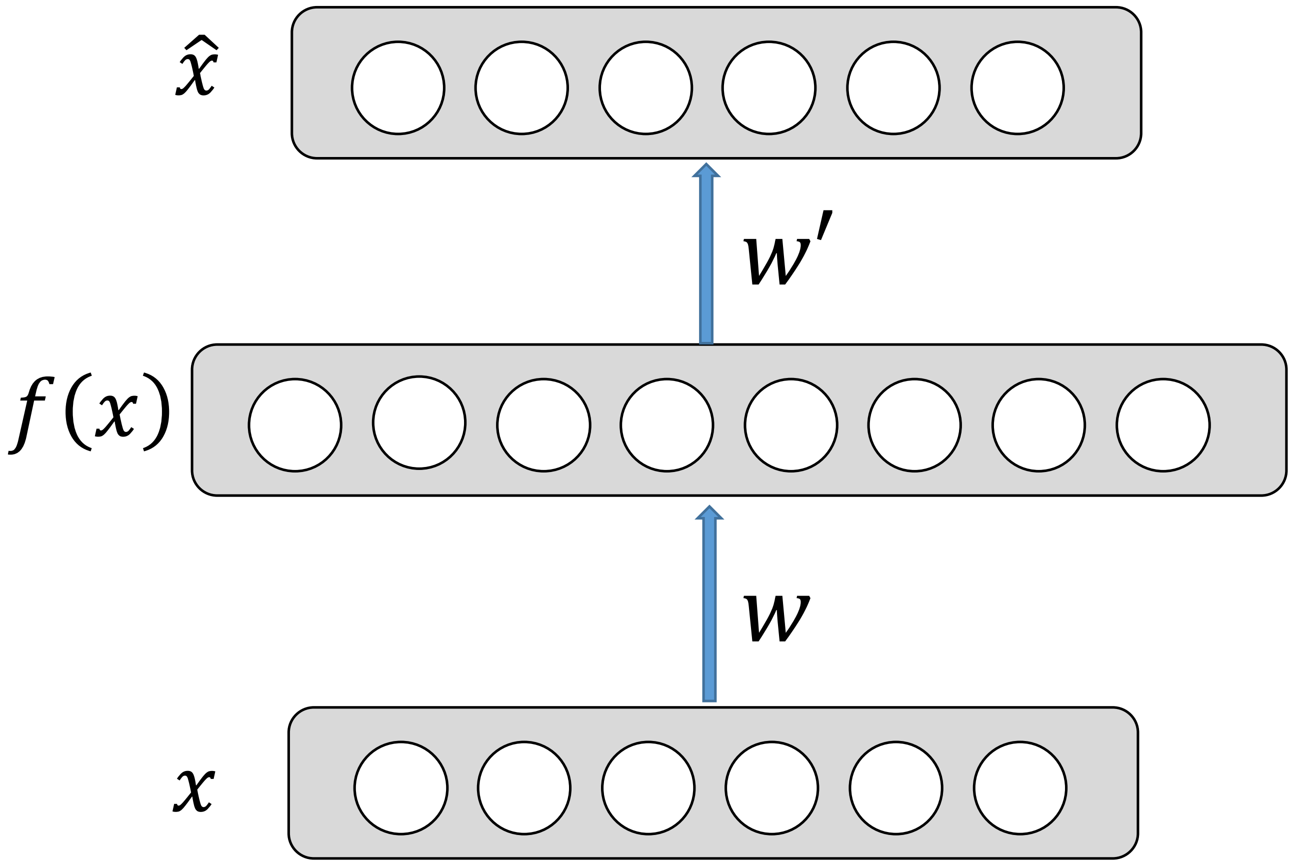
Convolutional AE

Overcomplete Auto-encoders

- Hidden layer is **Overcomplete** if greater than the input layer
  - No compression in hidden layer.
  - Each hidden unit could copy a different input component.

- No guarantee that the hidden units will extract meaningful structure.

- Adding dimensions is good for training a linear classifier (XOR case example).
- A higher dimension code helps model a more complex distribution.



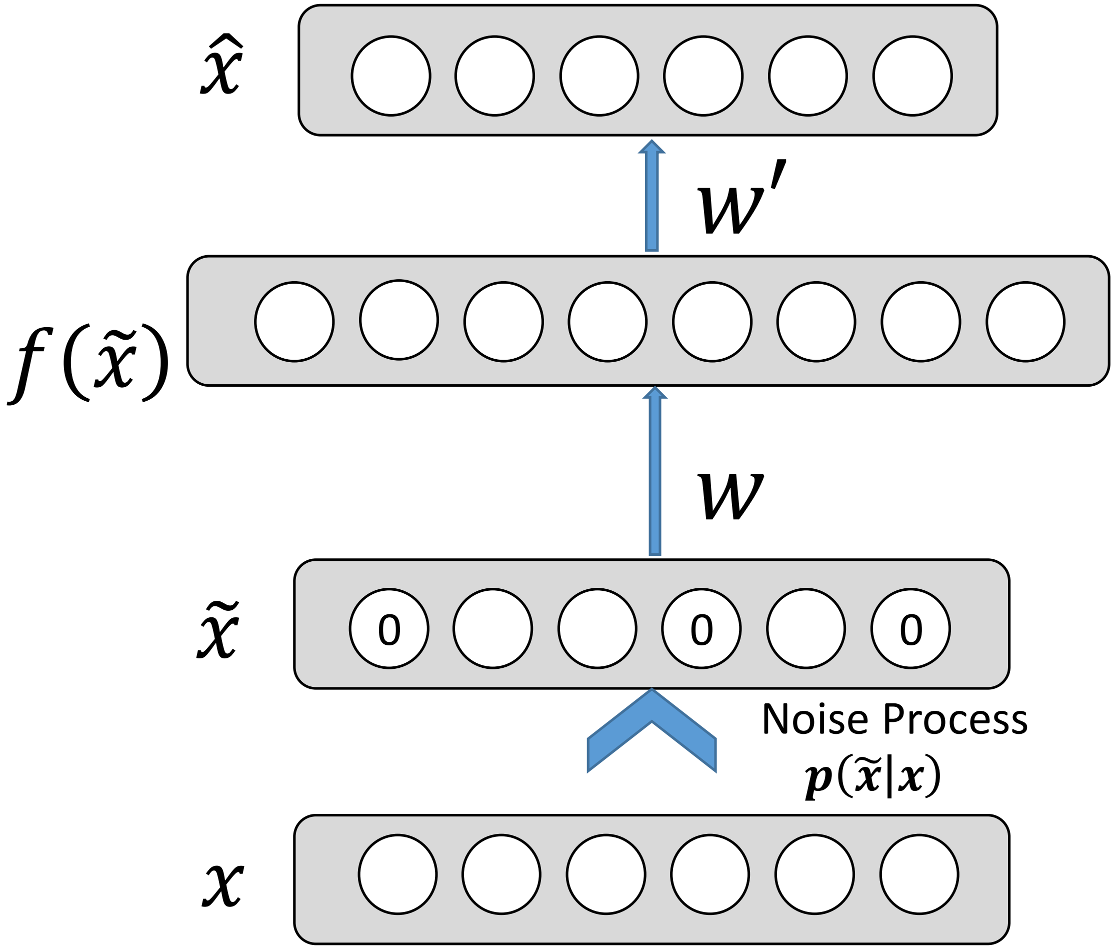
De-noising Auto-encoders

- Random assignment of subset of inputs to 0, with probability v.
- Gaussian additive noise.

- Reconstruction $\hat{x}$ computed from the corrupted input $\tilde{x}$.
- Loss function compares $\hat{x}$ reconstruction with the noiseless x

- The autoencoder cannot fully trust each feature of x independently so it must learn the correlations of x's features.
- Based on those relations we can predict a more 'not prune to changes' model.

We are forcing the hidden layer to learn a generalized structure of the data



Generative Adversarial Networks

Density estimation : Maximum Likelihood Estimator

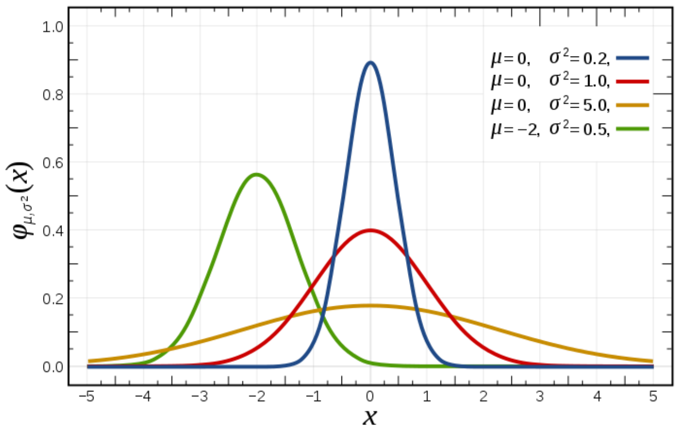
- An attempt to make the model distribution match the empirical distribution
  - Parameterization for the model that best fits the data
- Assume data is independent and identically distributed s(i.i.d) so that p factorizes

Parametric Family of distributions

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} p_{\mathrm{model}}(\mathbb{X}; \boldsymbol{\theta})$$

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}\sim\hat{p}_{\mathrm{data}}} \log p_{\mathrm{model}}(\boldsymbol{x}; \boldsymbol{\theta})$$

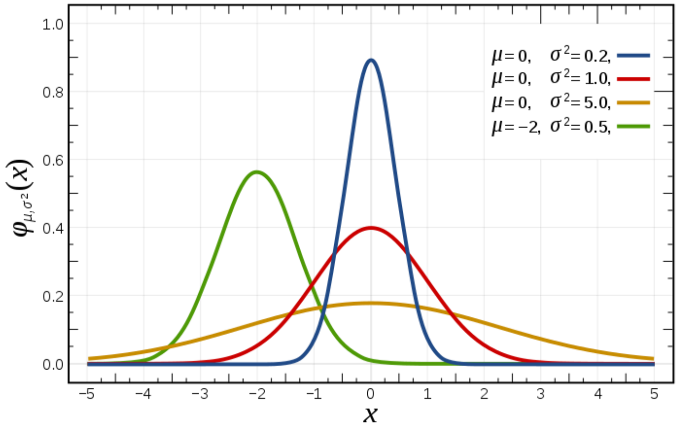
- MLE Example: Gaussian distribution (on white board)
- Parameter space view

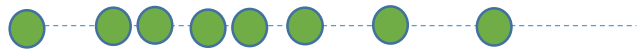


MLE Example

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}\sim\hat{p}_{\mathrm{data}}} \log p_{\mathrm{model}}(\boldsymbol{x};\boldsymbol{\theta})$$

- Given a set of input data = {x_1, …., x_n} find the MLE if we assume it fits the Gaussian N(μ, σ^2)





$$p(x_1,\dots,x_N|\mu,\sigma^2) = \prod_{n=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{\frac{-(x_n-\mu)^2}{2\sigma^2}\right\}$$

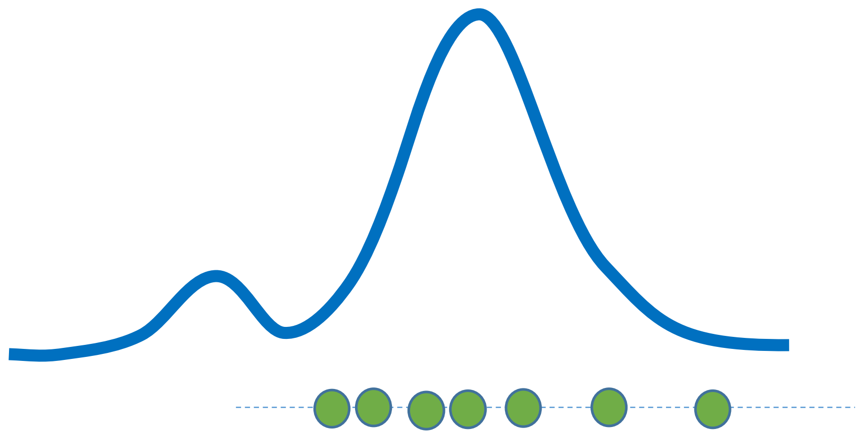
$$\mathcal{L}(\mu,\sigma) = -\frac{1}{2}N\log(2\pi\sigma^2) - \sum_{n=1}^{N}\frac{(x_n-\mu)^2}{2\sigma^2}$$

- What are the partial derivatives w.r.t paramters ?
  - Set them to zero to analytically solve for the optimal parameters
  - Not always possible for all distributions of Likelihoods

Generative Models

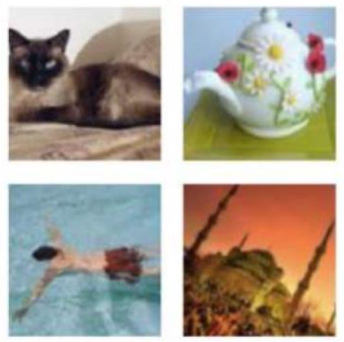
Goal: Take as input training samples from some distribution and learn a model that represents that distribution

Density Estimation



Sample Generation

Input samples

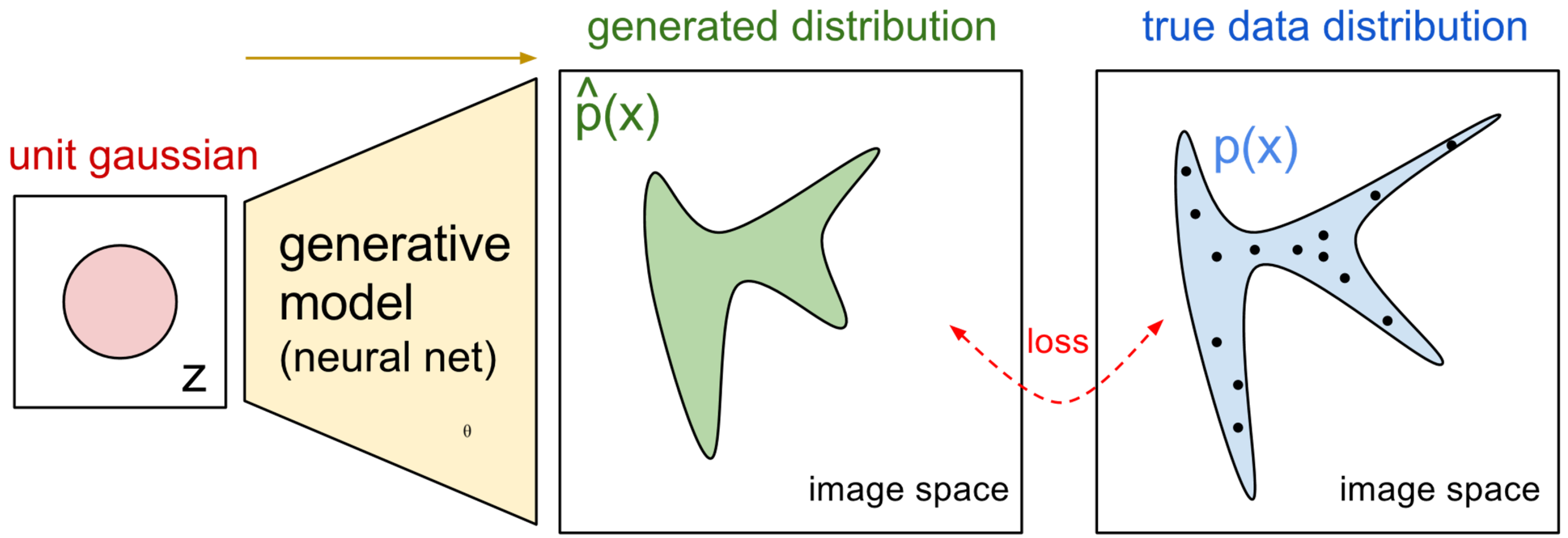


Generated samples

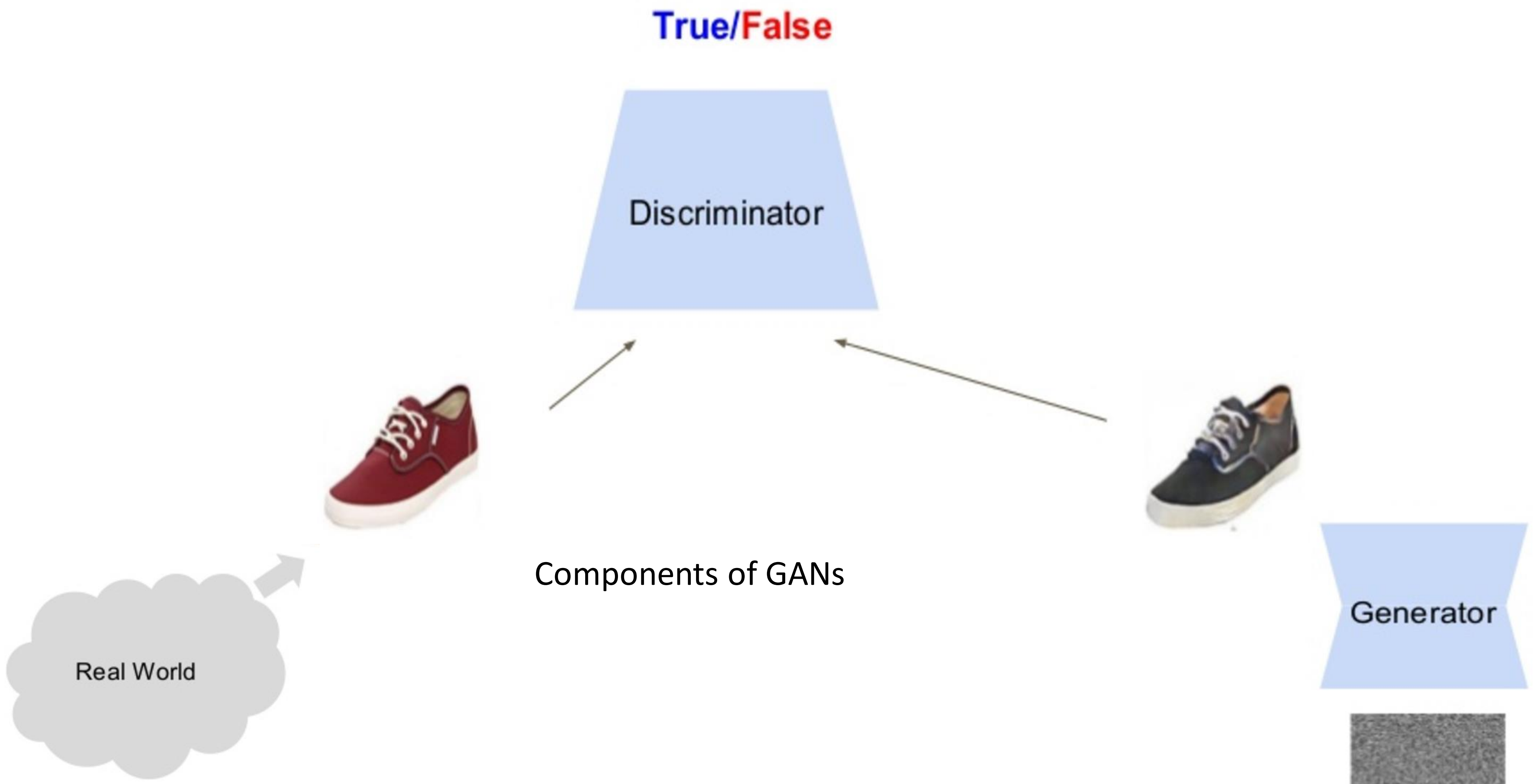
Generative Adversarial Networks

Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution, e.g. random noise. Learn transformation to training distribution.

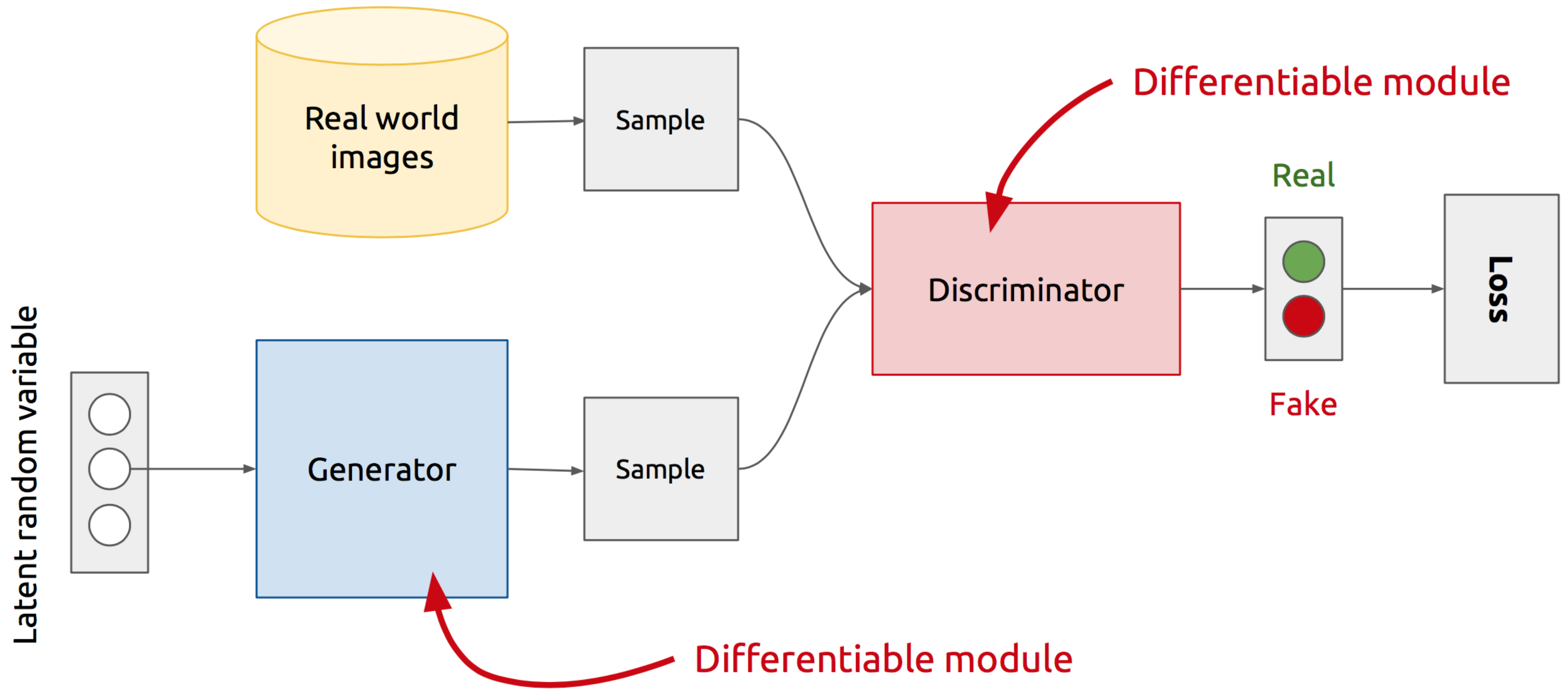


GAN Overview



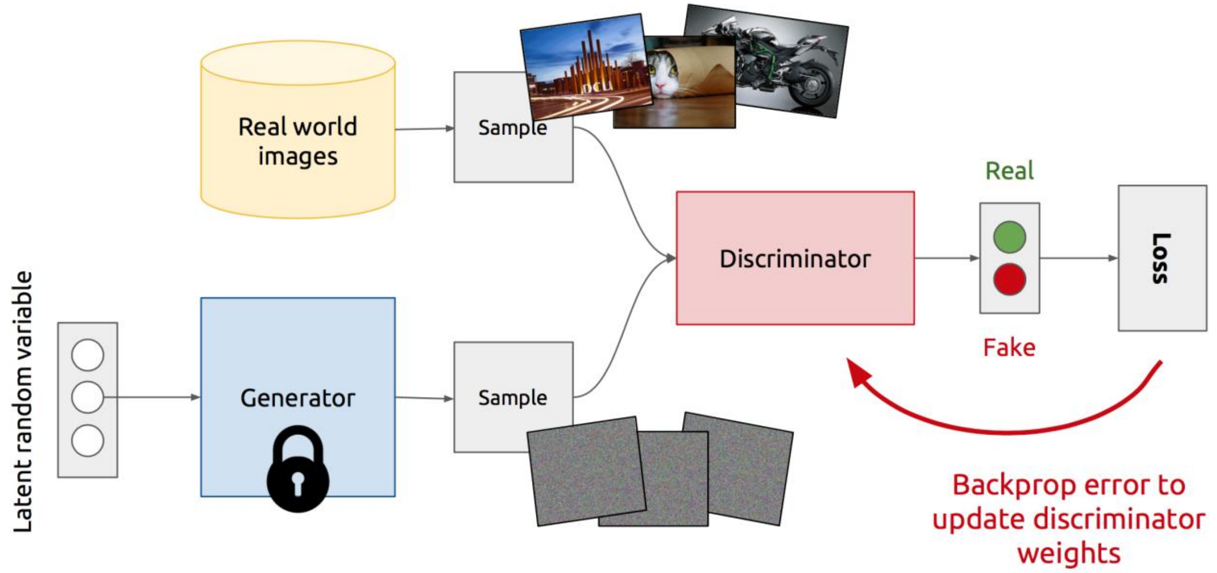
Components of GANs

GAN General Architecture

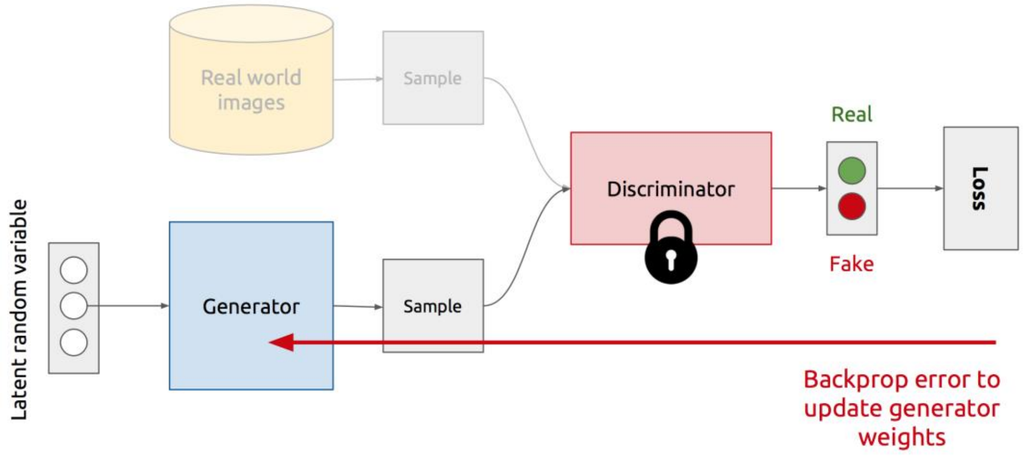


Training GANS

Training the discriminator



Training the Generator



Training Objective

- Discriminator tries to identify real data from fakes created by the generator
- Generator tries to create imitations of data to trick the discriminator
- **Train** GAN jointly via **minimax game**:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log \underbrace{D_{\theta_d}(x)}_{\text{Discriminator output for real data x}} + \mathbb{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data G(z)}}) \right]$$

- Discriminator wants to maximize objective such that
  - D(x) is close to 1 (real) and D(G(z)) is close to 0 (fake)
- Generator wants to minimize objective such that
  - D(G(z)) is close to 1 (discriminator is fooled into thinking generated G(z) is real)

Training Procedure

- Optimize

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log \underbrace{D_{\theta_d}(x)}_{\text{Discriminator output for real data x}} + \mathbb{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data G(z)}}) \right]$$

Alternate between:

- Gradient ascent on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

- Gradient descent on generator

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

Training Procedure

Alternate between:

- Gradient ascent on discriminator

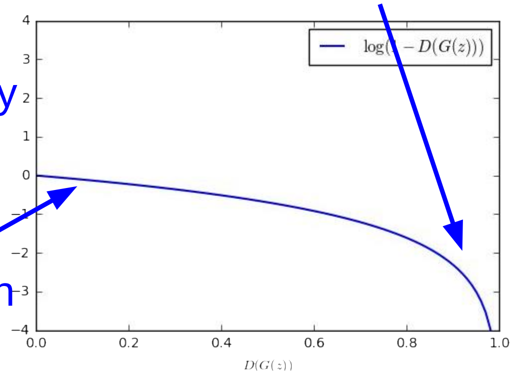
$$\max_{\theta_d} \left[\mathbb{E}_{x\sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z\sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))\right]$$

- Gradient descent on generator

$$\min_{\theta_g} \mathbb{E}_{z\sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

Gradient signal dominated by region where sample is already good

When sample is likely fake, want to learn from it to improve generator. But gradient in this region is relatively flat!



In practice, optimizing this generator objective does not work well!

Training Procedure

Alternate between:

- Gradient ascent on discriminator

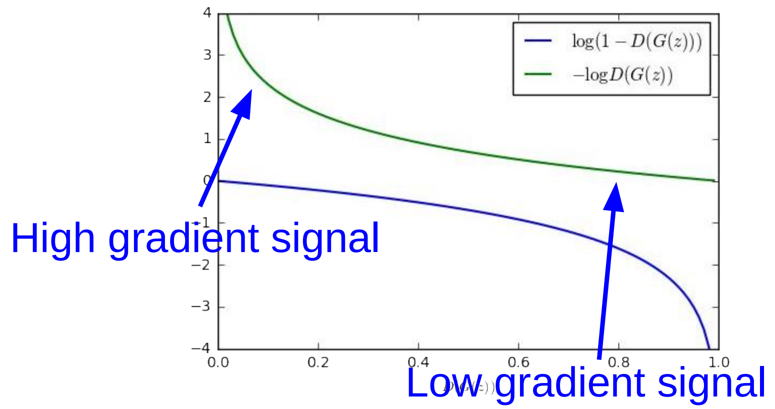
$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

- Gradient ascent on generator

$$\log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

Instead of minimizing likelihood of discriminator being correct, now maximize likelihood of discriminator being wrong.
Same objective of fooling discriminator, but now higher gradient signal for bad samples => works much better! Standard in practice.

Aside: Jointly training two networks is challenging, can be unstable. Choosing objectives with better loss landscapes helps training, is an active area of research.

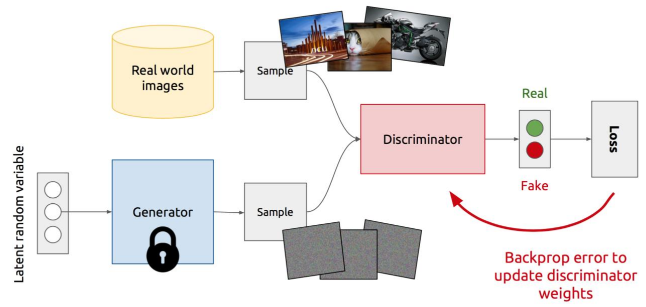


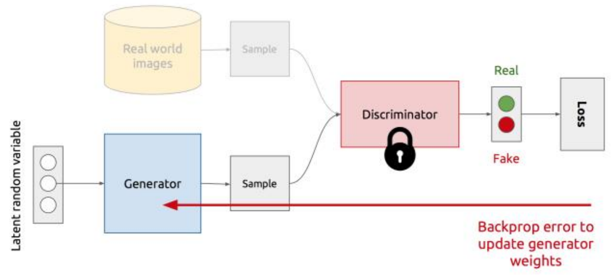
Training Procedure

Discriminator Training

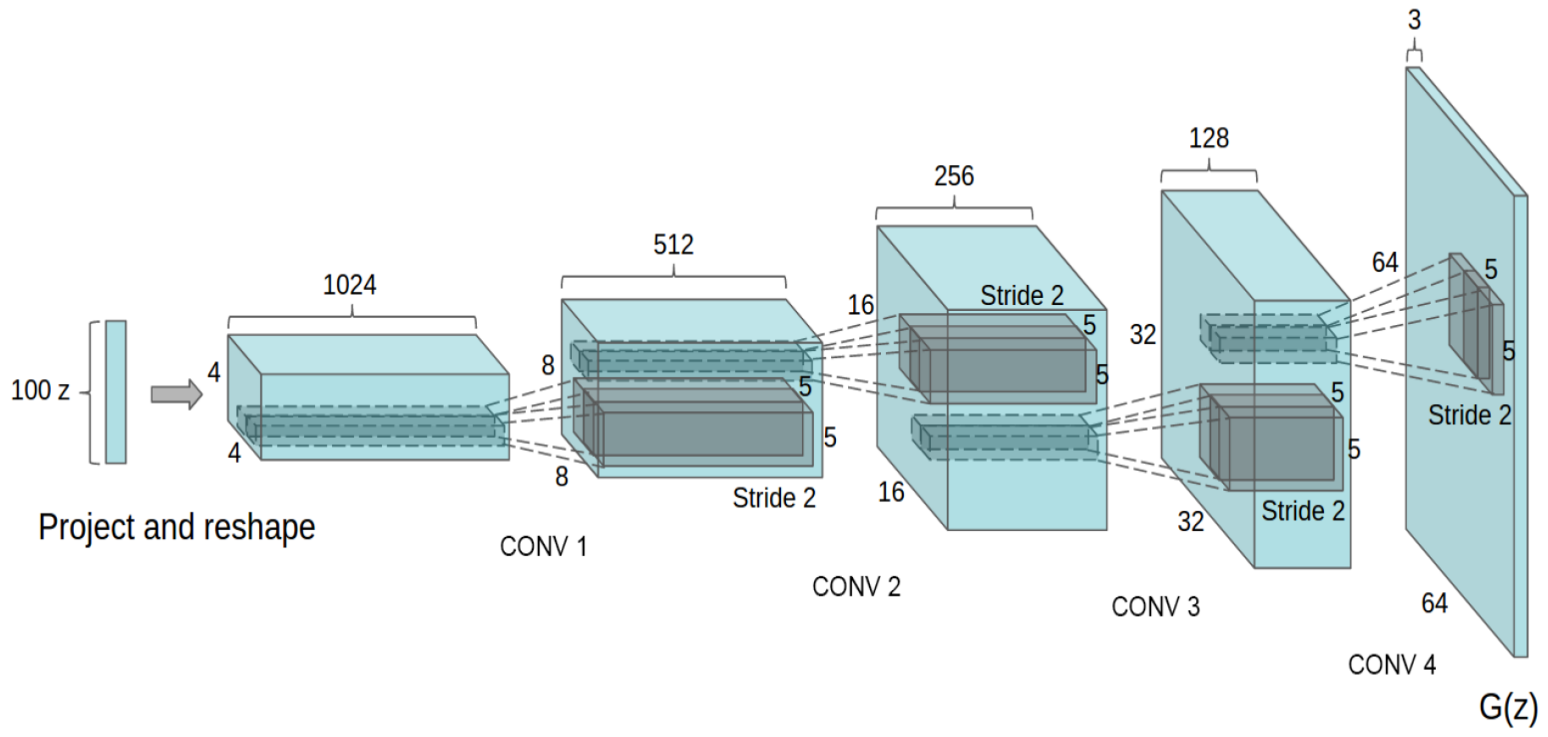
Generator Training

```
for number of training iterations do
    for $k$ steps do
        • Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
        • Sample minibatch of $m$ examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
        • Update the discriminator by ascending its stochastic gradient:
            $$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$
    end for
    • Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
    • Update the generator by ascending its stochastic gradient (improved objective):
        $$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$
end for
```





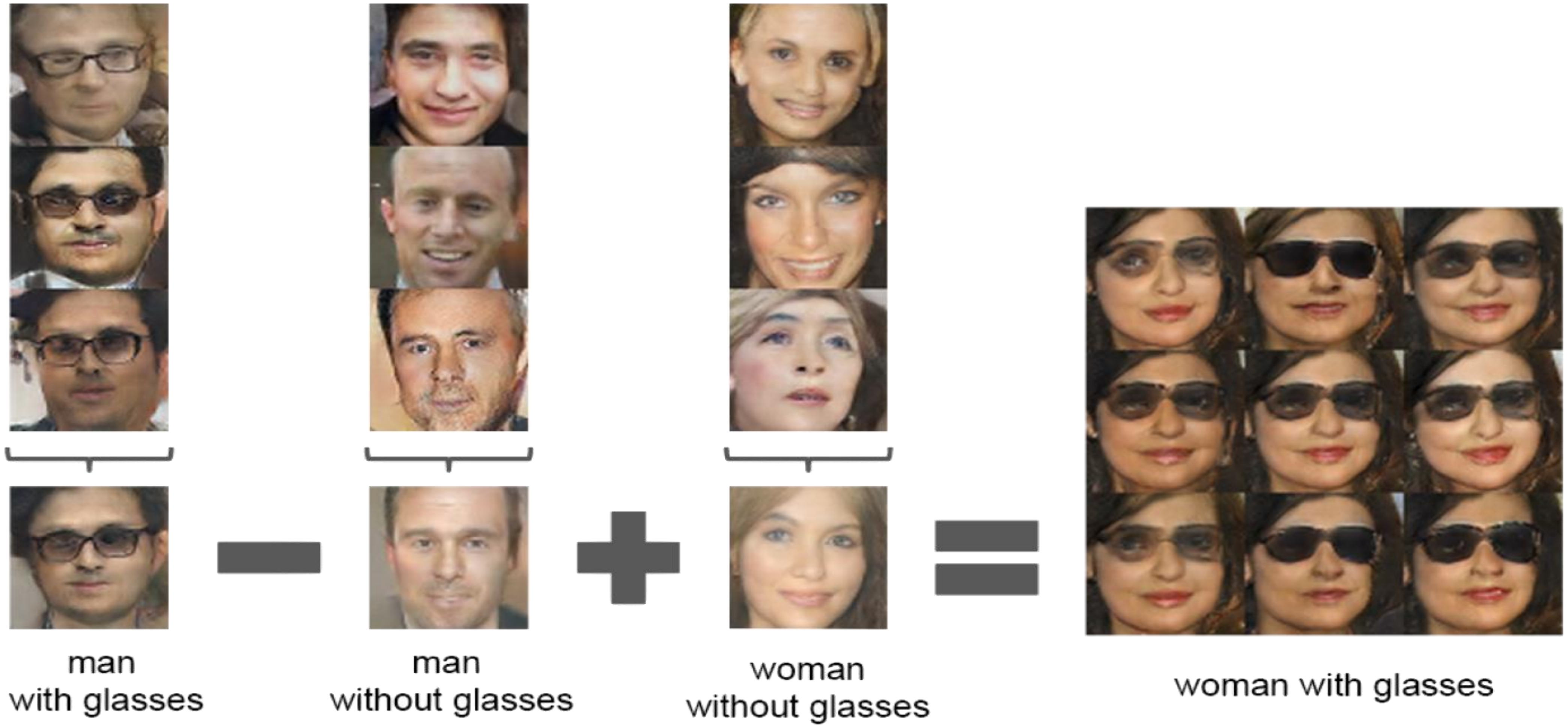
DC-GAN



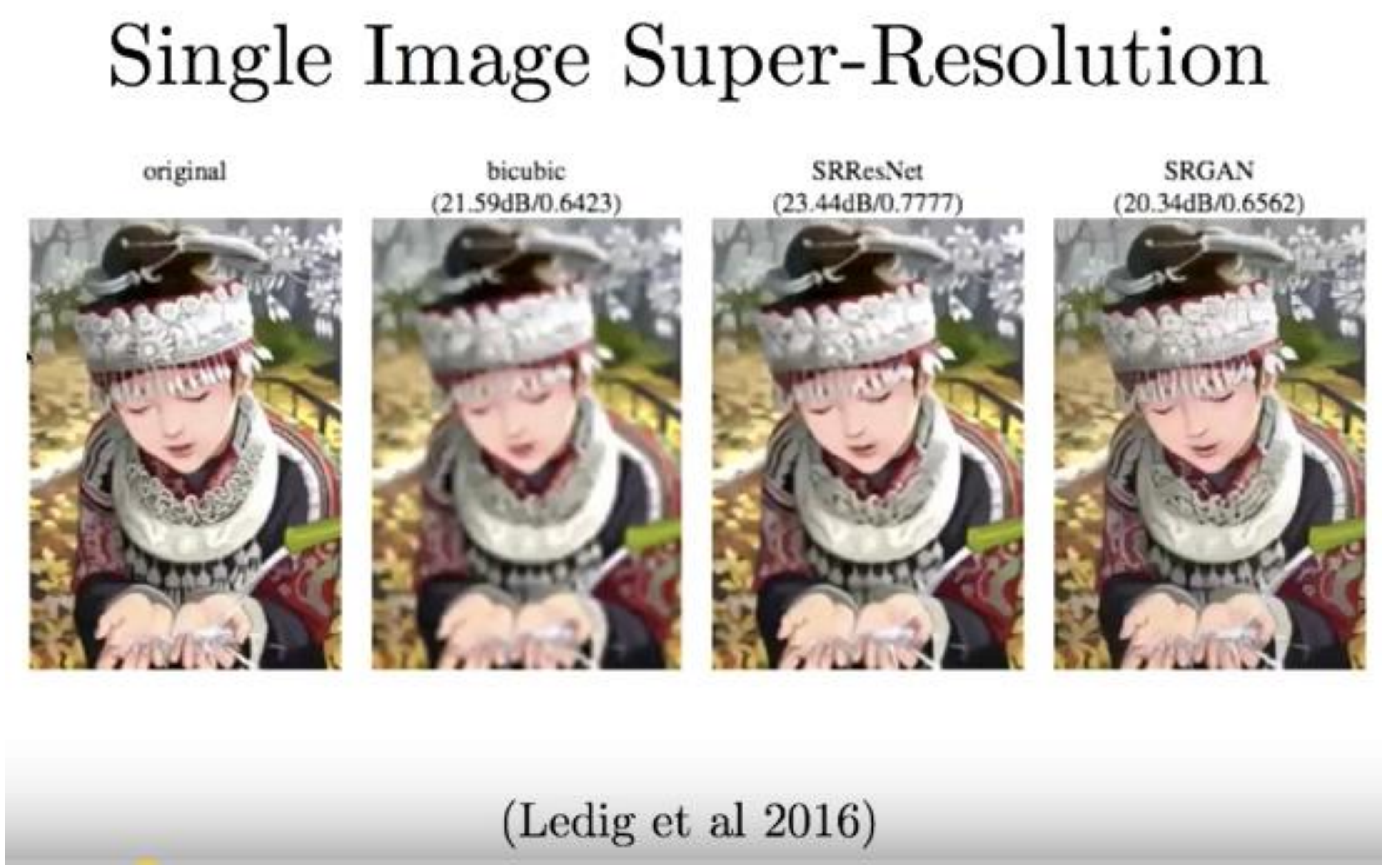
Key ideas:

- Replace FC hidden layers with Convolutions
  - Generator: Fractional-Strided convolutions
- Use Batch Normalization after each layer
- Inside Generator
  - Use ReLU for hidden layers
  - Use Tanh for the output layer

Man with Glasses - Man + Woman = ?



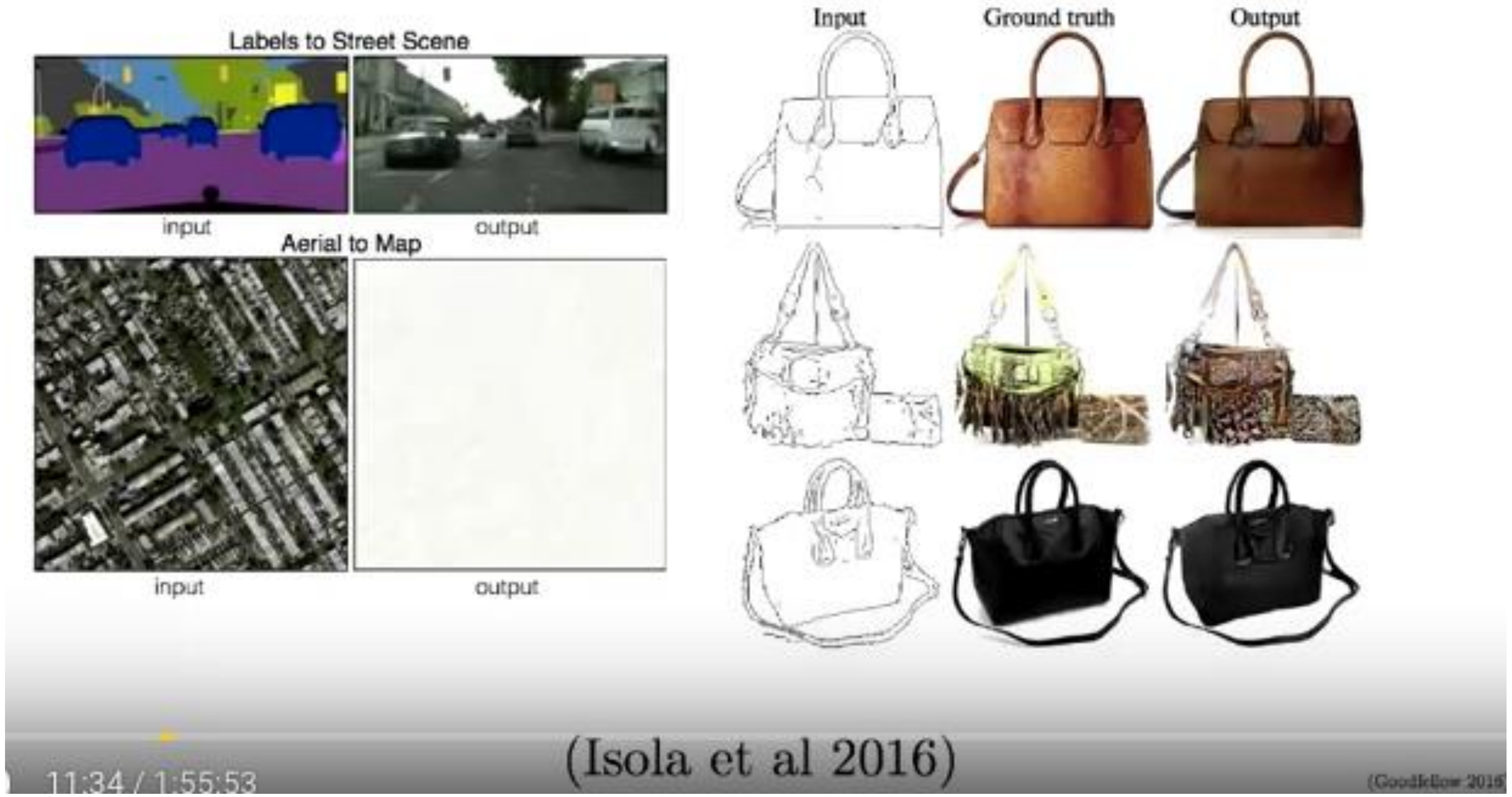
Applications fo GAN



Single Image Super-Resolution

(Ledig et al 2016)

Image to Image Translation



Advantages of GANs

- Plenty of existing work on Deep Generative Models but GANs
  - Don’t take explicit density function
  - Game theoretic Approach
  - More crisp and better Sample generation

- Why GANs?
  - Sampling (or generation) is straightforward
  - Training doesn't involve Maximum Likelihood estimation
  - Robust to Overfitting since Generator never sees the training data
  - Empirically, GANs are good at capturing the modes of the distribution

Disadvantages of GANs

- Probability Distribution is Implicit
  - Not straightforward to compute P(X).
  - Thus Vanilla GANs are only good for Sampling/Generation.

- Training is Hard
  - Non-Convergence
  - Mode-Collapse
    - Can focus on a few realistic images from the training dataset
  - Evaluation is hard
  - Game theoretic objectives are new and still evolving

References

- https://arxiv.org/abs/1804.10253
- Chapter on Autoencoders in Deep Learning Book
- https://github.com/soumith/ganhacks