Generative models

Lecture 10

Henning Petzka

Models of machine learning

Increasing complexity

- **Discriminant function**:
  - Find a discriminative function mapping a sample x to its class label (e.g. binary classification function: perceptron, SVM,...)

Models of machine learning

Increasing complexity

- **Discriminant function**:
  - Find a discriminative function mapping a sample x to its class label (e.g. binary classification function: perceptron, SVM,...)
- **Discriminative models**:
  - Find a (posterior) probability distribution $p(C_k|x)$ determining for each sample x a probability for belonging to class C_k.
  - Choosing the class with highest probability gives a discriminant function.

Models of machine learning

Increasing complexity

- **Discriminant function**:
  - Find a discriminative function mapping a sample x to its class label (e.g. binary classification function: perceptron, SVM,...)
- **Discriminative models**:
  - Find a (posterior) probability distribution $p(C_k|x)$ determining for each sample x a probability for belonging to class C_k.
  - Choosing the class with highest probability gives a discriminant function.
- **Generative models**:
  - Find the probability distribution $p(C_k, x)$.
  - Normalizing over p(x) gives

$$\frac{p(C_k, x)}{p(x)} = p(C_k|x)$$

Models of machine learning

Advantages of complex models

- **Discriminant functions**
  - provides little insight into the model and is fixed after learning.

Models of machine learning

Advantages of complex models

- **Discriminant functions**
  - provides little insight into the model and is fixed after learning.

- **Discriminative models**
  - describing $p(C_k|x)$ allows the application of different loss functions
  - provides us with a confidence value for classification.

Models of machine learning

Advantages of complex models

- **Discriminant functions**
  - provides little insight into the model and is fixed after learning.

- **Discriminative models**
  - describing $p(C_k|x)$ allows the application of different loss functions
  - provides us with a confidence value for classification.

- **Generative models**
  - computing the joint probability distribution $p(C_k, x)$ makes the model „generative“, since it has the power of generating data by sampling from the distribution.
  - also allow outlier detection, i.e. to find points with very low probabilities.

  ! By solving a Generative model, we automatically get a Discriminative model and a Discriminant function !

A little probability theory

Bayes Theorem

- From $p(x, y) = p(x|y) \cdot p(y)$
- we get:

$$p(y|x) = \frac{p(y, x)}{p(x)} = \frac{p(x|y)p(y)}{p(x)}$$

- Applied to binary classification into classes C_1 and C_2

$$p(C_1|x) = \frac{p(C_1|x) \cdot p(C_1)}{p(x)} = \frac{p(x|C_1) \cdot p(C_1)}{p(x|C_1) \cdot p(C_1) + p(x|C_2) \cdot p(C_2)}$$

Probabilistic view on machine learning

Probabilistic binary classification

$$p(C_1|x) = \frac{p(x|C_1) \cdot p(C_1)}{p(x|C_1) \cdot p(C_1) + p(x|C_2) \cdot p(C_2)}$$

$$\text{Define } a = \ln\left(\frac{p(x|C_1) \cdot p(C_1)}{p(x|C_2) \cdot p(C_2)}\right)$$

$$p(C_1|x) = \frac{1}{1 + \exp(-a)}$$

Probabilistic view on machine learning

Probabilistic binary classification

$$p(C_1|x) = \frac{p(x|C_1) \cdot p(C_1)}{p(x|C_1) \cdot p(C_1) + p(x|C_2) \cdot p(C_2)}$$

$$\text{Define } a = \ln\left(\frac{p(x|C_1) \cdot p(C_1)}{p(x|C_2) \cdot p(C_2)}\right)$$

$$p(C_1|x) = \frac{1}{1 + \exp(-a)} = \sigma(a) \text{ with } \sigma \text{ the sigmoid function.}$$

Probabilistic view on machine learning

Probabilistic binary classification

Assume probability distributions of all classes are Gaussian distributions with the same variance.

$$p(x|C_k) = \frac{1}{((2\pi)^d|\Sigma|)^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu_k)^T\Sigma^{-1}(x-\mu_k)\right)$$

$$\text{Define } w = \Sigma^{-1}(\mu_1 - \mu_2),$$

$$w_0 = -\frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 + \frac{1}{2}\mu_2^T\Sigma^{-1}\mu_2 + \ln\left(\frac{p(C_1)}{p(C_2)}\right)$$

$$\text{Then } p(C_1|x) = \sigma(w^T x + w_0)$$

Probabilistic view on machine learning

Probabilistic binary classification

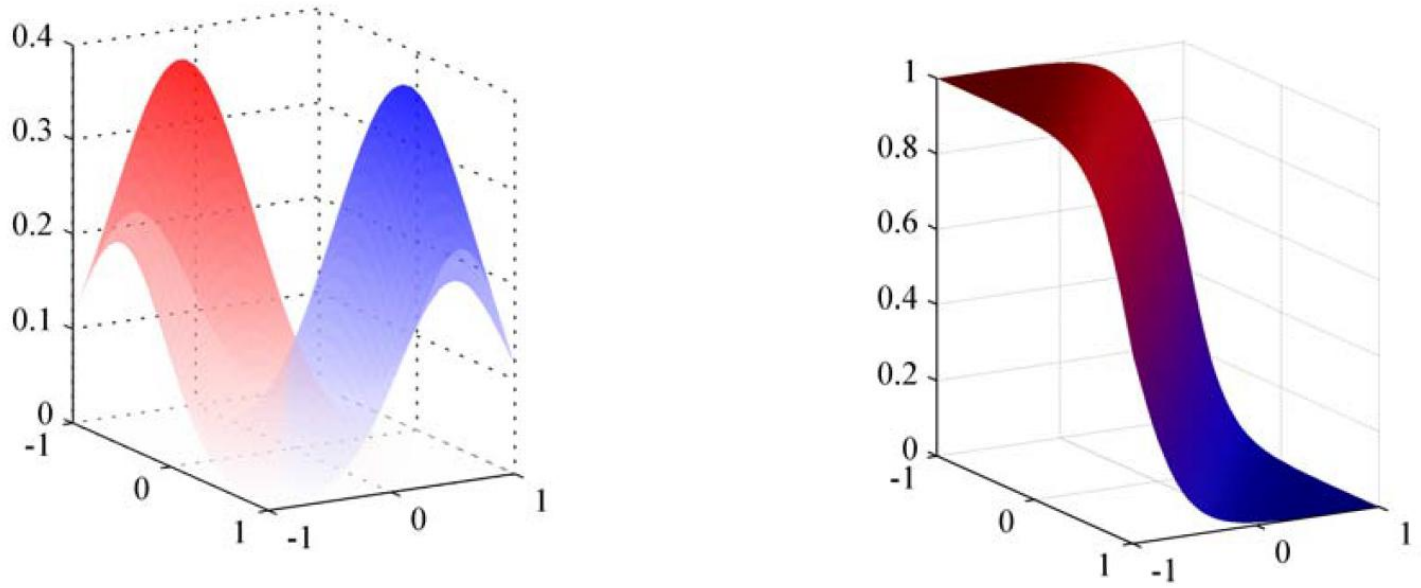


Figure 4.10 The left-hand plot shows the class-conditional densities for two classes, denoted red and blue. On the right is the corresponding posterior probability $p(\mathcal{C}_1|\mathbf{x})$, which is given by a logistic sigmoid of a linear function of $\mathbf{x}$. The surface in the right-hand plot is coloured using a proportion of red ink given by $p(\mathcal{C}_1|\mathbf{x})$ and a proportion of blue ink given by $p(\mathcal{C}_2|\mathbf{x}) = 1 - p(\mathcal{C}_1|\mathbf{x})$.

from Bishop's book „Pattern matching and machine learning“

Probabilistic view on machine learning

Probabilistic multiclass classification

$$p(C_k|x) = \frac{p(x|C_k) \cdot p(C_k)}{\sum_j p(x|C_j) \cdot p(C_j)}$$

$$\text{Define } a_k = \ln\left(p(x|C_k) \cdot p(C_k)\right)$$

$$p(C_k|x) = \frac{\exp\left(a_k\right)}{\sum_j \exp(a_j)}$$

Probabilistic view on machine learning

Probabilistic multiclass classification

$$p(C_k|x) = \frac{p(x|C_k) \cdot p(C_k)}{\sum_j p(x|C_j) \cdot p(C_j)}$$

$$\text{Define } a_k = \ln\left(p(x|C_k) \cdot p(C_k)\right)$$

$$p(C_k|x) = \frac{\exp\left(a_k\right)}{\sum_j \exp(a_j)} = \text{softmax}_k(a_1, \ldots, a_j, \ldots, a_n)$$

Probabilistic view on machine learning

Probabilistic multiclass classification

Assume probability distributions of all classes are Gaussian distributions with the same variance.

$$p(x|C_k) = \frac{1}{((2\pi)^d|\Sigma|)^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu_k)^T\Sigma^{-1}(x-\mu_k)\right)$$

$$\text{Define } w_k = \Sigma^{-1}\mu_k$$

$$b_k = -\frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \ln\left(p(C_k)\right)$$

$$\text{Then } p(C_k|x) = \text{softmax}(w_1^T x + b_1, \ldots, w_n^T x + b_n)$$

Probabilistic view on machine learning

Probabilistic multiclass classification

When we allow the different classes to have different covariance matrices $\sum_k$

then the decision boundaries become quadratic.

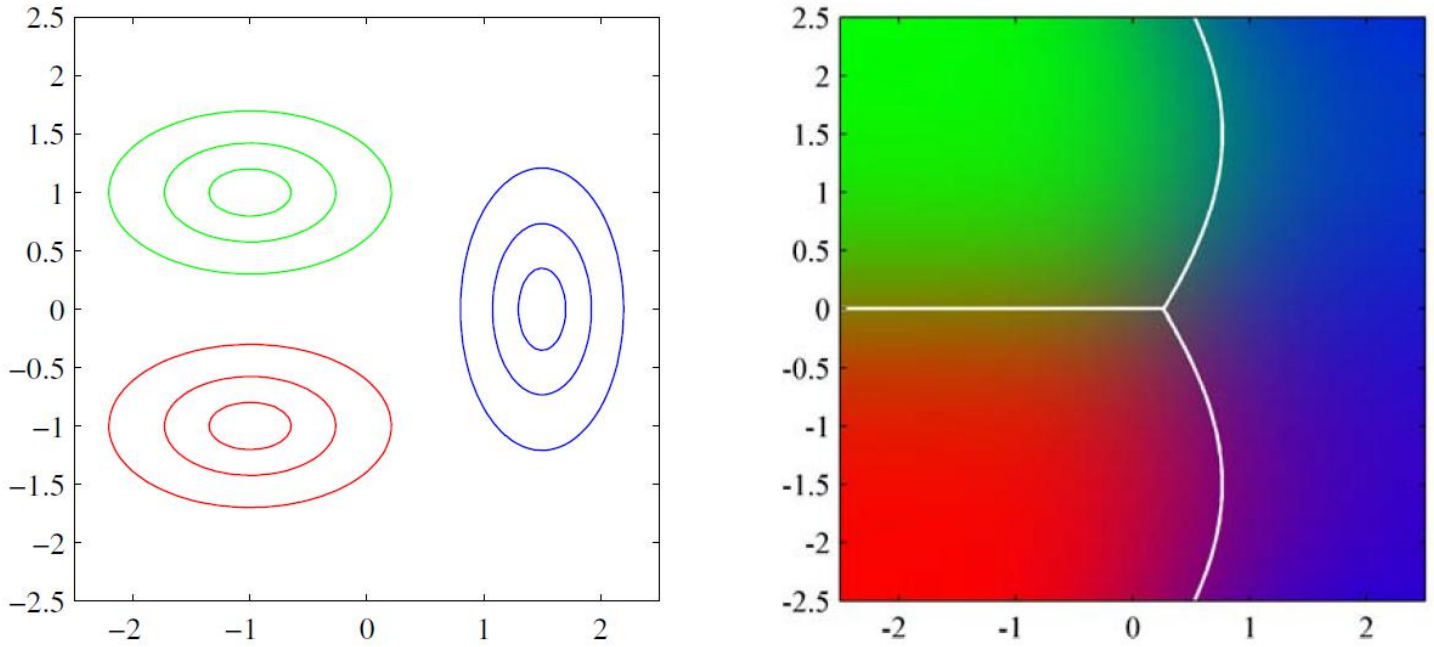


Figure 4.11 The left-hand plot shows the class-conditional densities for three classes each having a Gaussian distribution, coloured red, green, and blue, in which the red and green classes have the same covariance matrix. The right-hand plot shows the corresponding posterior probabilities, in which the RGB colour vector represents the posterior probabilities for the respective three classes. The decision boundaries are also shown. Notice that the boundary between the red and green classes, which have the same covariance matrix, is linear, whereas those between the other pairs of classes are quadratic.

from Bishop's book „Pattern matching and machine learning“

Probabilistic view on machine learning

Probabilistic classification

- The models we have seen can be motivated from a probabilistic viewpoint as discriminative models

- The sigmoid and the softmax function appear naturally in this context.

Probabilistic models

How to train them?

- We have a parameterized description of the likelihoods $p(C_k|x)$
  - For example, in the example above, the parameters were the means μ_k

- Given labeled samples (x_i,$C_{k(i)}$), we want to maximize all

$$p(C_{k(i)}|x_i)$$

- Assuming independence of the sample, we want to maximize

$$\prod_i p(C_{k(i)}|x_i)$$

over the parameters describing the probabilities.

Sequential tasks

Example setup

- Assume, there are three types of weather (sunny, rainy, cloudy)
- Task: Predict wether of next day.
- Training set: A sequence

Day	1	2	3	4	5	6	7	8	9
Weather									

Sequential tasks

Example setup

- Assume, there are three types of weather (sunny, rainy, cloudy)
- Task: Predict wether of next day.
- Training set: A sequence

Day	1	2	3	4	5	6	7	8	9
Weather									

- If we assume the data to be i.i.d, then to predict the weather we can only use the relative frequencies
- However, we would like to take trends into account:

Sequential tasks

Example setup

- Assume, there are three types of weather (sunny, rainy, cloudy)
- Task: Predict wether of next day.
- Training set: A sequence

Day	1	2	3	4	5	6	7	8	9
Weather									

- If we assume the data to be i.i.d, then to predict the weather we can only use the relative frequencies
- However, we would like to take trends into account:
- q_n = weather of day n, want a model $p(q_{n+1} | q_n, q_{n-1}, q_{n-2},\ldots,q_1)$

Markov models

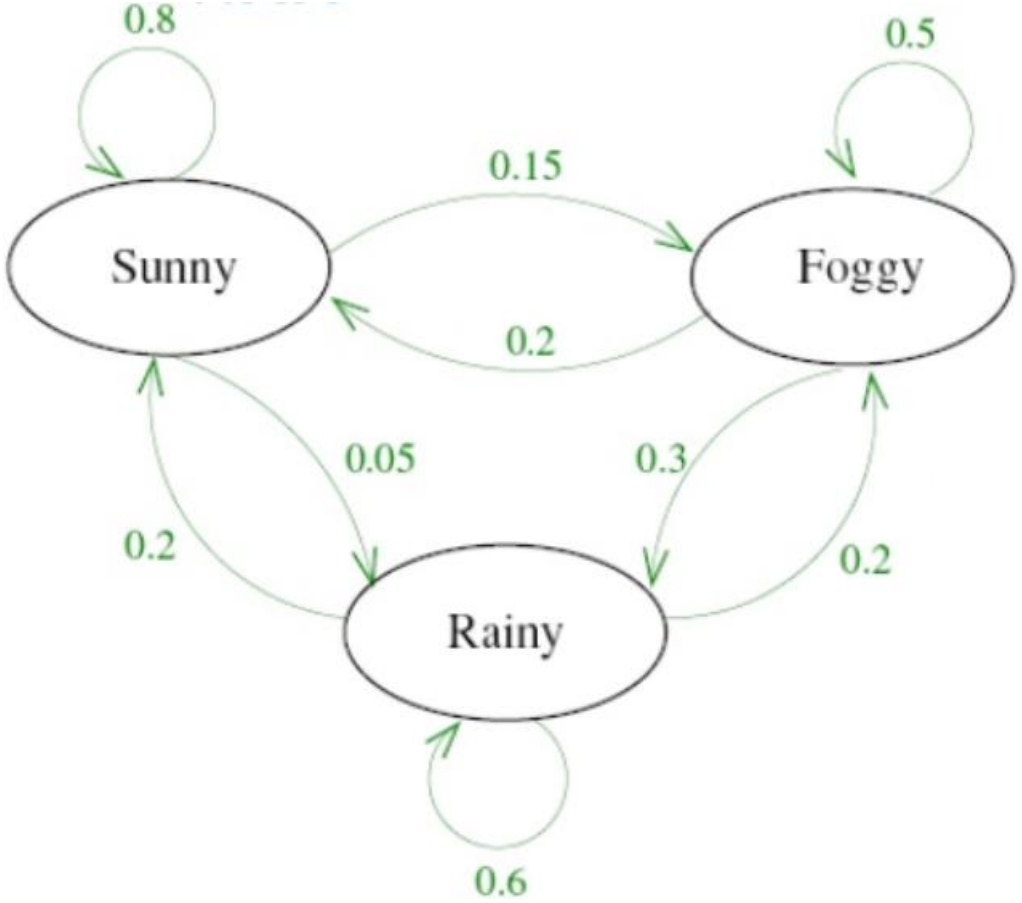
Markov property

- q_n = weather of day n, want a model $p(q_{n+1}|q_n, q_{n-1}, q_{n-2},\ldots,q_1)$

- Markov assumption (Markov property)
  - Future states depend only on the current state, not on the events that occurred before it
  - Here: Tomorrow's weather only depend on today's weather
  - $p(q_{n+1}|q_n, q_{n-1}, q_{n-2},\ldots,q_1)=p(q_{n+1}|q_n)$

Markov models

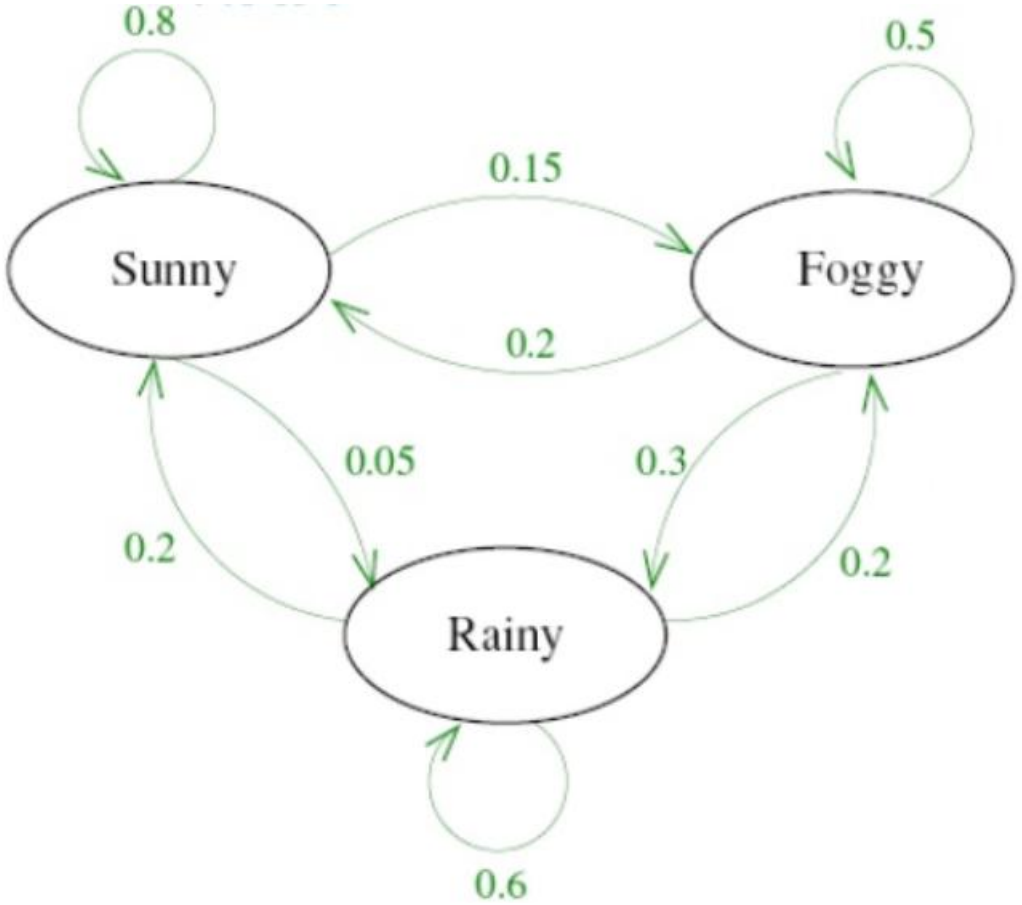
Example



Today's weather	Tomorrow's weather		
	0.8	0.05	0.15
	0.2	0.6	0.2
	0.2	0.3	0.5

Markov models

Example



Today's weather	Tomorrow's weather		
☀	0.8	0.05	0.15
🌧	0.2	0.6	0.2
☁	0.2	0.3	0.5

This leads to a (simple) generative model

$$p(C_k|x)=p(q_n|q_{n-1})$$

By sampling from it, we can **generate** a sequence of weather predictions:

☀ ☀ ☀ ☁ ☁ 🌧

p = 0.8 * 0.8 * 0.15 * 0.5 * 0.3

Hidden Markov models

The model

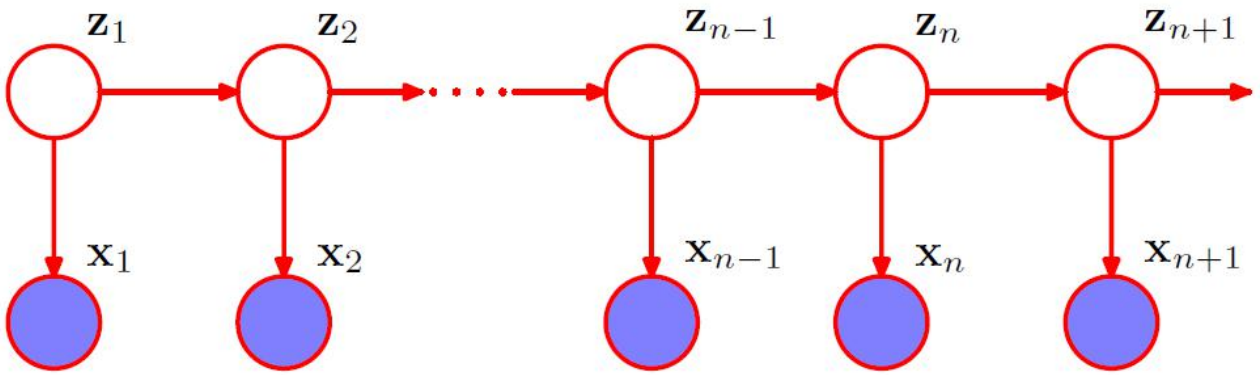
- Markov models seem extremely limiting.
- Even with a k-th step Markov model $p(q_{n+1}|q_n, q_{n-1}, q_{n-2},\ldots,q_{n-k})$ we are limited to k previous observations.

Hidden Markov models

The model

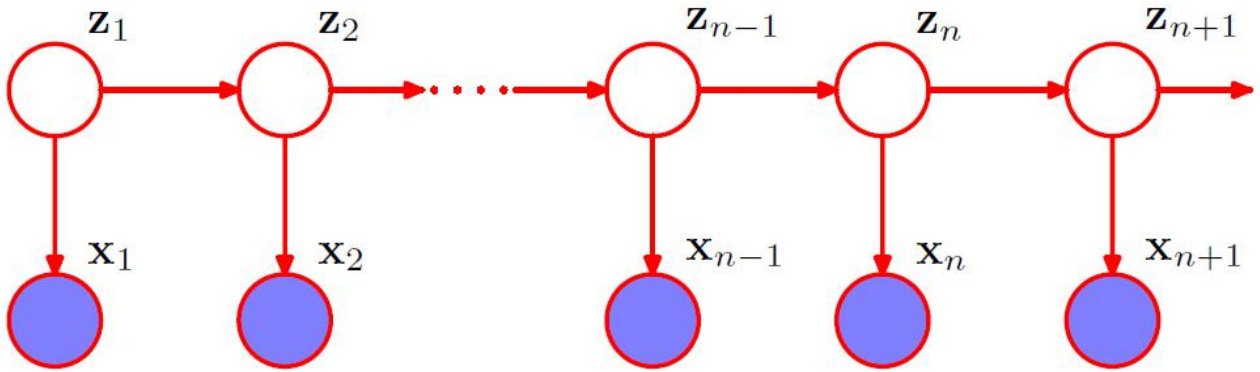
- Markov models seem extremely limiting.
- Even with a k-th step Markov model $p(q_{n+1}|q_n, q_{n-1}, q_{n-2},\ldots,q_{n-k})$ we are limited to k previous observations.

- Idea: Introduce latent variables to permit a rich class of models
- For each observation x_n, we introduce a corresponding latent variable z_n.
- We now assume that it is the latent variables that form a Markov chain
- Assuming the latent variable to be discrete, we get the Hidden Markov model



Hidden Markov models

The model



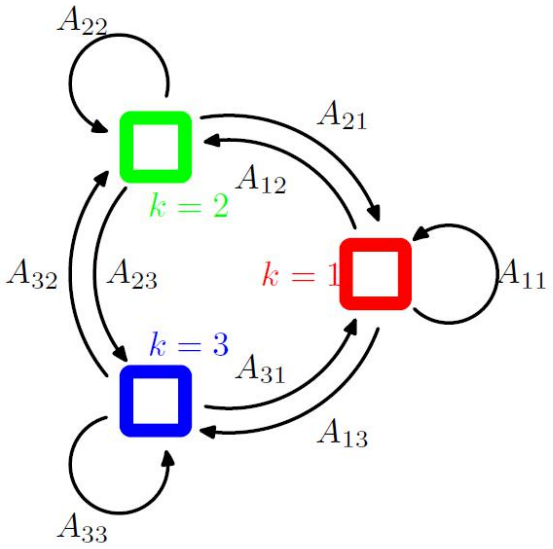
$$p(x_1, \ldots, x_n, z_1, \ldots, z_n) = p(z_1) \left(\prod_{j=2}^{n} p(z_j|z_{j-1}) \right) \left(\prod_{j=1}^{n} p(x_j|z_j) \right)$$

Hidden Markov models

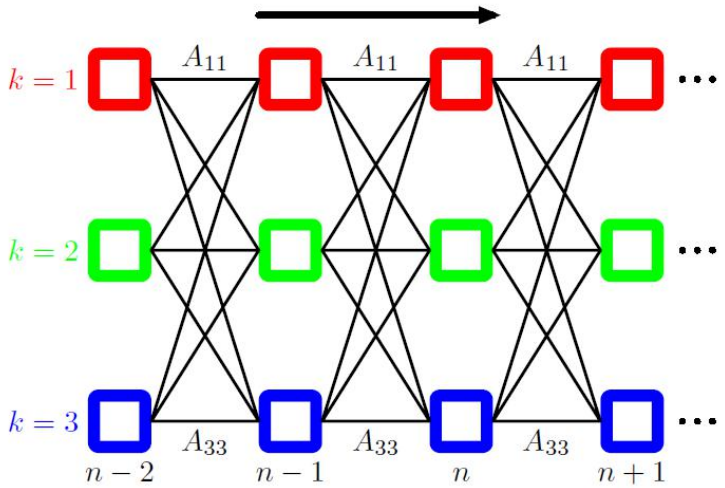
The model

- We encode the discrete latent variables z_j by a K-dimensional vector where $z_{j,k}$ =1 if z_j takes on class k and 0 otherwise
- The transition between hidden states is encoded by a matrix A of size (K,K) where A_{ij} encodes the probability of transition from state j to state i:

$$A_{i,j} = p\left(z_{n,i} = 1 \mid z_{n-1,j} = 1\right)$$



unfolded in time:



Hidden Markov models

The model

- Only left to specify and parameterize the distribution of the observed variable conditional on the latent variable

  $p(x_n|z_n,\phi)$, where ϕ is a set of parameters

- For example, ϕ could specify Gaussian distributions as before.
- Parameters:

$$\theta = \{A, \phi\}$$

- Model:

$$p(X, Z|\theta) = p(z_1) \left(\prod_{i=2}^{n} p(z_i|z_{i-1}, A) \right) \left(\prod_{j=1}^{n} p(x_j|z_j, \phi) \right)$$

Hidden Markov models

HMMs as generative models

$$p(X, Z|\theta) = p(z_1) \left(\prod_{i=2}^{n} p(z_i|z_{i-1}, A) \right) \left(\prod_{j=1}^{n} p(x_j|z_j, \phi) \right)$$

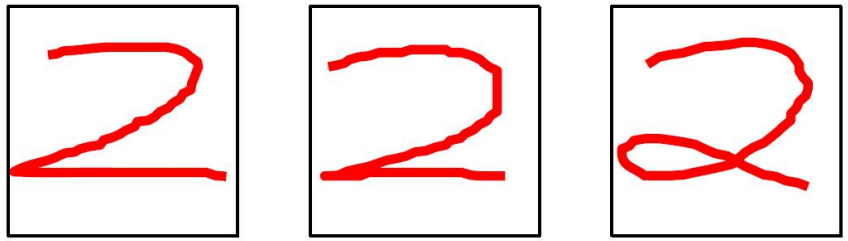
- Sample z_1
- Sample x_1 from z_1 using parameters ϕ
- Sample z_2 from z_1 using transition probabilities A
- Sample x_2 from z_2 using parameters ϕ
- .....

Hidden Markov models

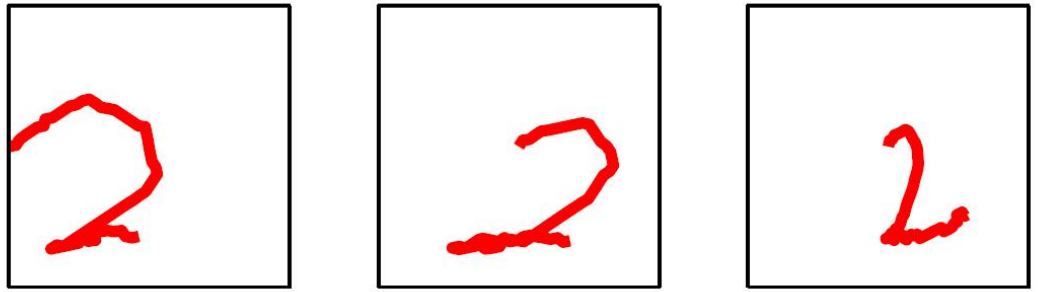
HMMs as generative models

Example: Hand-written digits

- Each digit is encoded by a trajectory of pen movements in form of a sequence of pen coordinates
- Dataset: 45 trajectories to write a '2'



- Latent variable has K=16 possible states corresponding to 16 different angles of pen movement
- Results:



- We have generated realistically looking 2's from a very simple generative model.

Generative models

Limitations of probabilistic models

- We have seen HMM as a probabilistic generative model
- There are, of course, more complex generative models

Generative models

Limitations of probabilistic models

- We have seen HMM as a probabilistic generative model
- There are, of course, more complex generative models
- However, they typically have quite restrictive assumption on the data.
  - Mixture of Gaussians, hidden Markov model, naive Bayes.

Generative models

Limitations of probabilistic models

- We have seen HMM as a probabilistic generative model
- There are, of course, more complex generative models
- However, they typically have quite restrictive assumption on the data.
  - Mixture of Gaussians, hidden Markov model, naive Bayes.
- For the distribution of visually appealing images, all of the assumptions of the mentioned methods are unrealistic

Generative models

Generative Models in Deep Learning

- In Deep Learning, the term „Generative Model“ is not used as strictly as to model $p(C_k,x)$.

- There are Deep Learning models that do not allow the explicit evalutation of a computed probability distribution function,....

Generative models

Generative Models in Deep Learning

- In Deep Learning, the term „Generative Model“ is not used as strictly as to model $p(C_k,x)$.

- There are Deep Learning models that do not allow the explicit evalutation of a computed probability distribution function,....
- ... but they support operations that implicitly require knowledge of it, such as drawing samples from the distribution

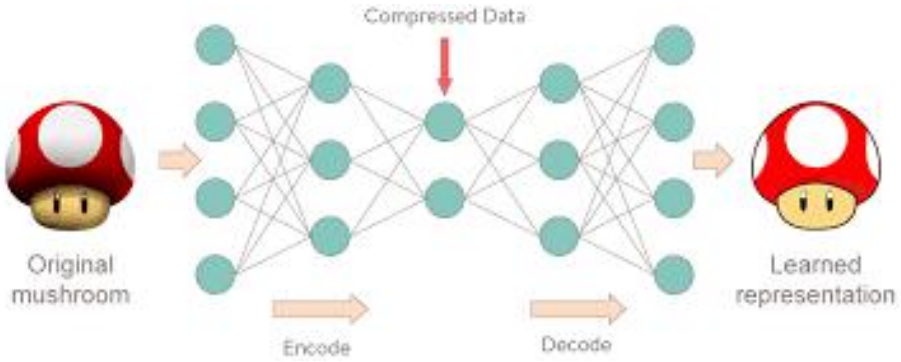
Generative models

Generative Models in Deep Learning

- In Deep Learning, the term „Generative Model“ is not used as strictly as to model $p(C_k,x)$.
- There are Deep Learning models that do not allow the explicit evalutation of a computed probability distribution function,....
- ... but they support operations that implicitly require knowledge of it, such as drawing samples from the distribution
- Simply put: In Deep Learning, models are called generative if they have the power of generating new data.
- So let us consider neural network approaches to generate data.

Autoenoder

The network

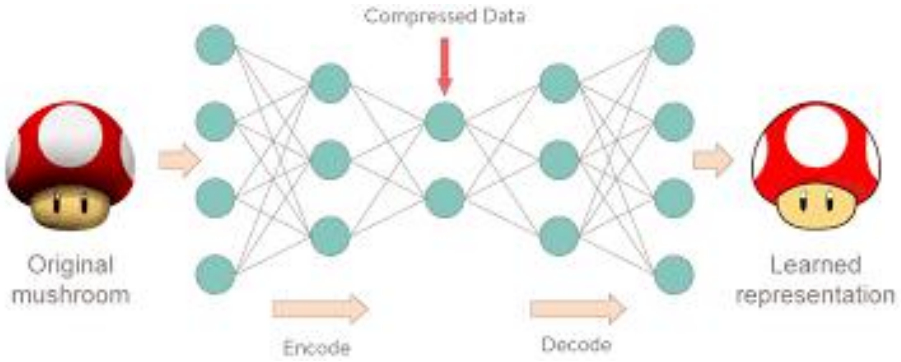


https://towardsdatascience.com/deep-autoencoders-using-tensorflow-c68f075fd1a3

- Neural network with a low-dimensional hidden layer
- Dimension of input layer equals the dimension of the output layer
- Training objective: Reconstruct the input
- Small hidden layer is called bottleneck, because the model needs to compress the information in the data in a low-dimensional space such that a reproduction of the original is possible

Autoenoder

Training objective



https://towardsdatascience.com/deep-autoencoders-using-tensorflow-c68f075fd1a3

Objective: $\min_{dec,\ enc} \sum_{i=1}^{N} ||f(x_i) - x_i||^2$

$f : \mathbb{R}^D \to \mathbb{R}^D$

$f = dec \circ enc$

$enc : \mathbb{R}^D \to \mathbb{R}^d$, encoder function, $d < D$

$dec : \mathbb{R}^d \to \mathbb{R}^D$, decoder function

Training set $\{x_i | i = 1, \ldots, N\} \subseteq \mathbb{R}^D$

Trained by backprop as usual.

Autoenoder

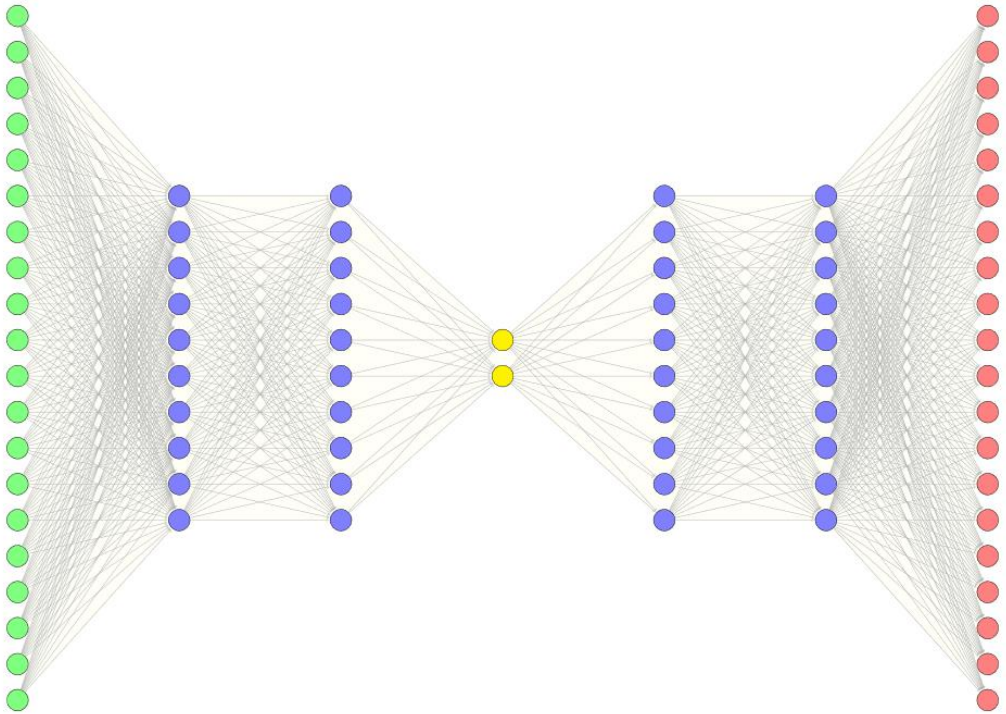
MNIST example

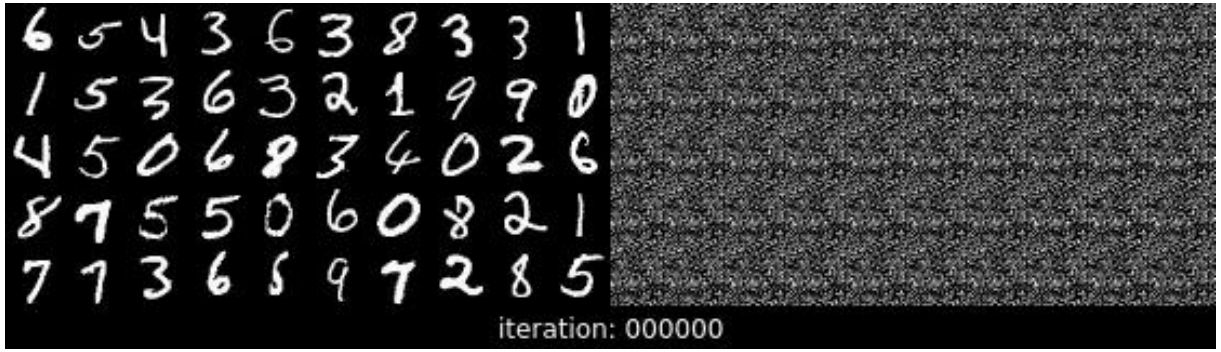
32 dimensions

We can even conpress to two dimenions.

Autoenoder

MNIST example

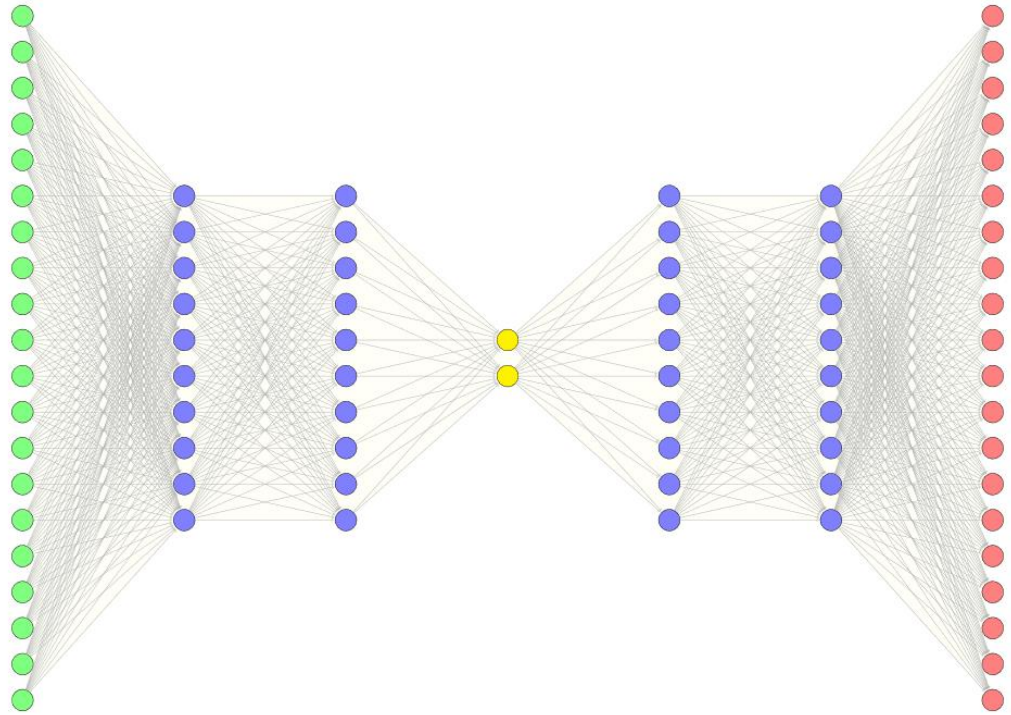


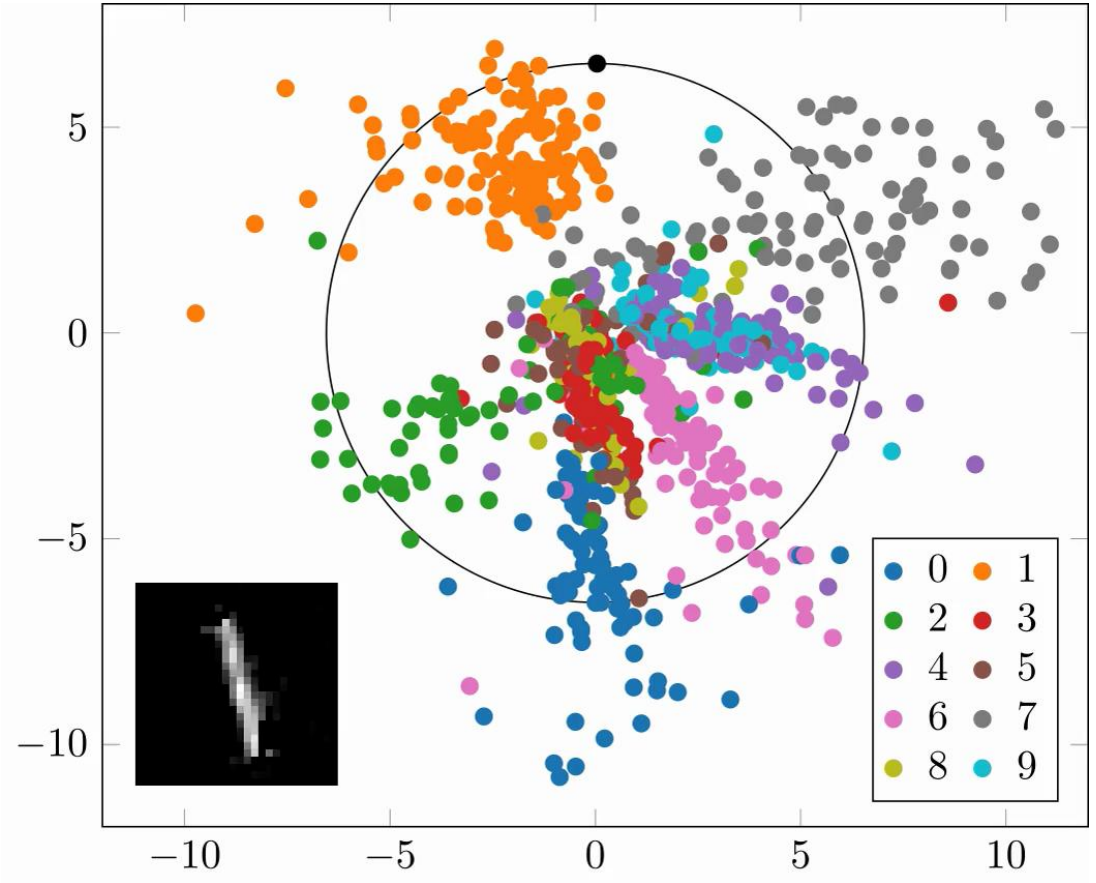


https://gertjanvandenburg.com/blog/autoencoder/

Autoenoder

MNIST example





https://gertjanvandenburg.com/blog/autoencoder/

Autoenoder

Applications

- Dimensionality reduction
- Denoising
  - Train the autoencoder on good quality data
  - Feed in noisy data
  - Noise gets forgotten in bottleneck and disapppears in reconstruction

Autoenoder

Applications

- Dimensionality reduction
- Denoising
  - Train the autoencoder on good quality data
  - Feed in noisy data
  - Noise gets forgotten in bottleneck and disapppears in reconstruction
- Outlier detection
  - Train the autoencoder on typical data
  - Autoencoder cannot reproduce outliers
  - Outliers will have high reconstruction loss

Autoenoder

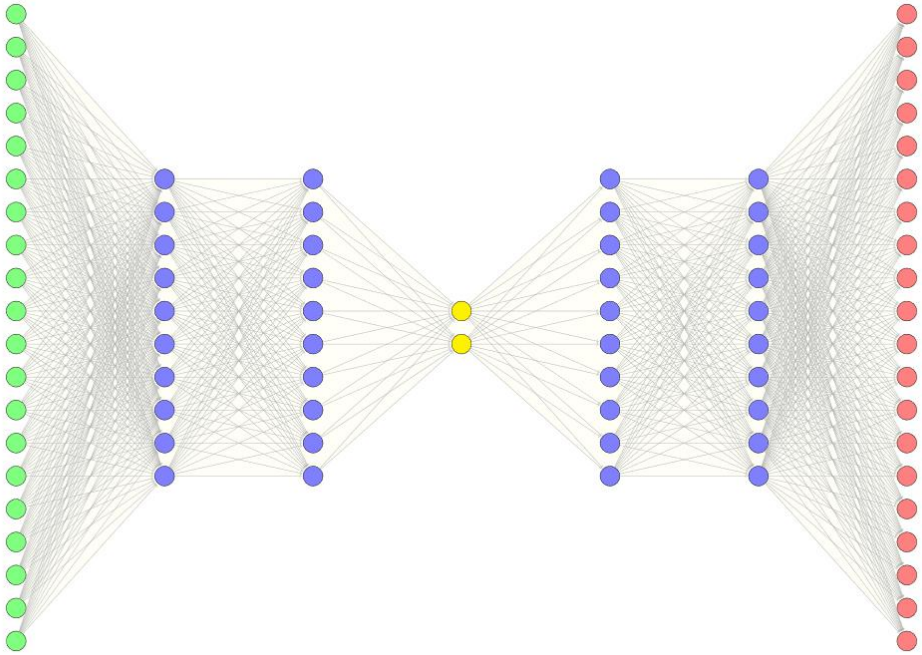
Applications

- Dimensionality reduction
- Denoising
  - Train the autoencoder on good quality data
  - Feed in noisy data
  - Noise gets forgotten in bottleneck and disapppears in reconstruction
- Outlier detection
  - Train the autoencoder on typical data
  - Autoencoder cannot reproduce outliers
  - Outliers will have high reconstruction loss
- Generation of new data? Is it a generative model?

Autoenoder

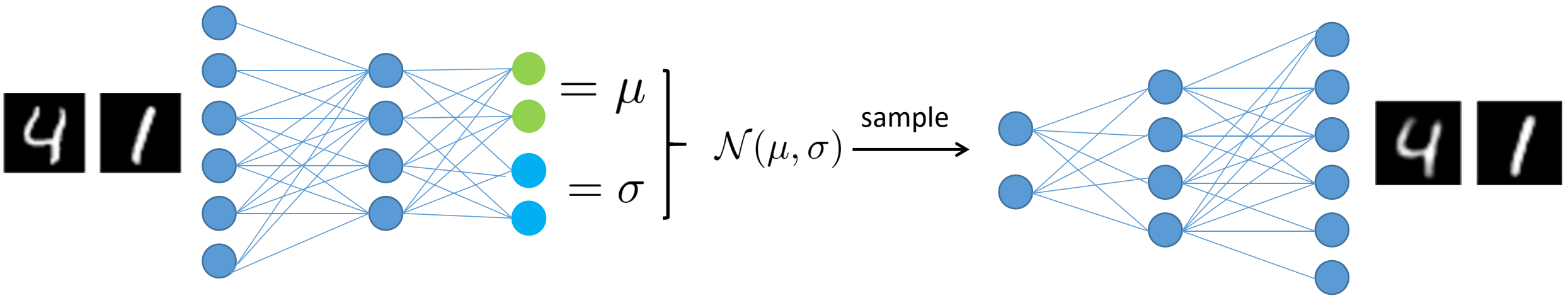
As a generative model?

- We would like to sample from hidden layer to generate new images.
- There is no probabilistic model at the hidden layer that we can sample from
- We do not even know the image of the encoder, so from where in the hidden layer can realistic points be sampled?
- Idea: Enforce a probabilistic model at the hidden layer



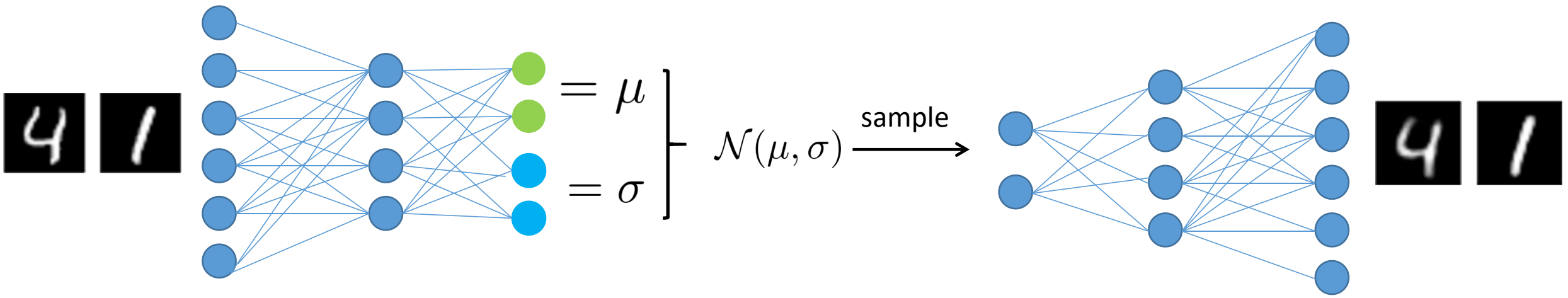
Variational Autoenoder

The model - Combining a probabistic model and DL



Variational Autoenoder

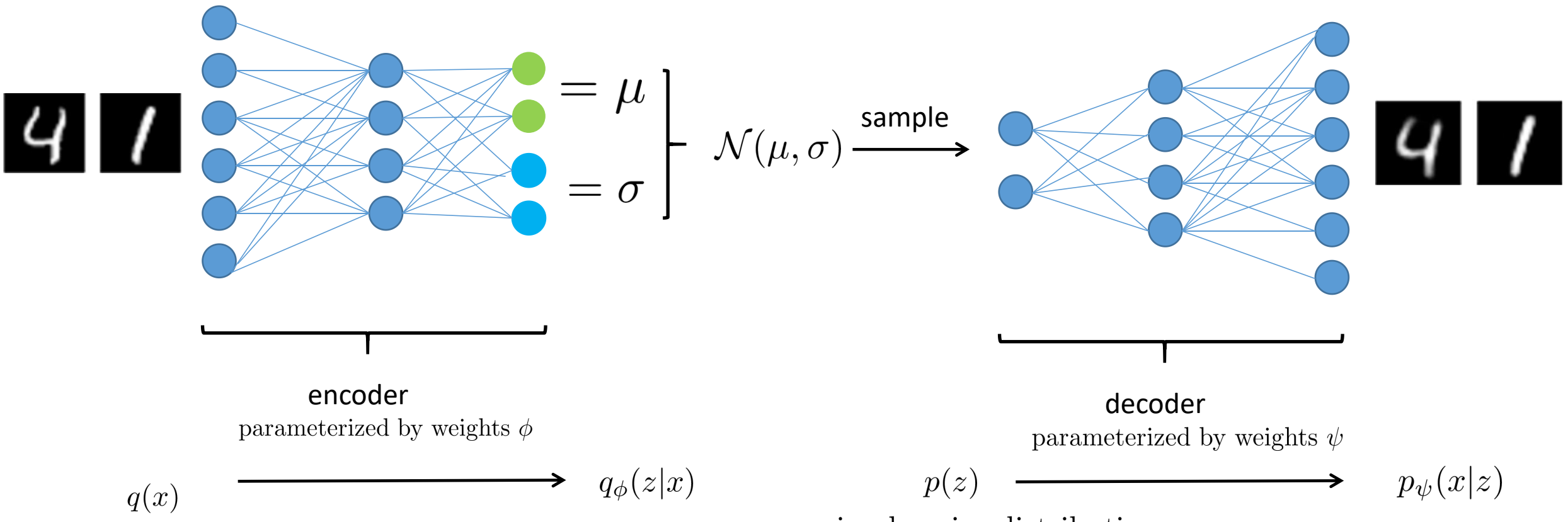
The model - Combining a probabistic model and DL



- At the bottleneck, we learn the mean and the standard deviation of a multinomial Gaussian distribution.
- We can then sample from the Gaussian and decode from the sample and try to reconstruct the original.
- We will also try to enforce a simple unit-variance Gaussian at the bottleneck over all samples

Variational Autoenoder

The model - Combining a probabistic model and DL

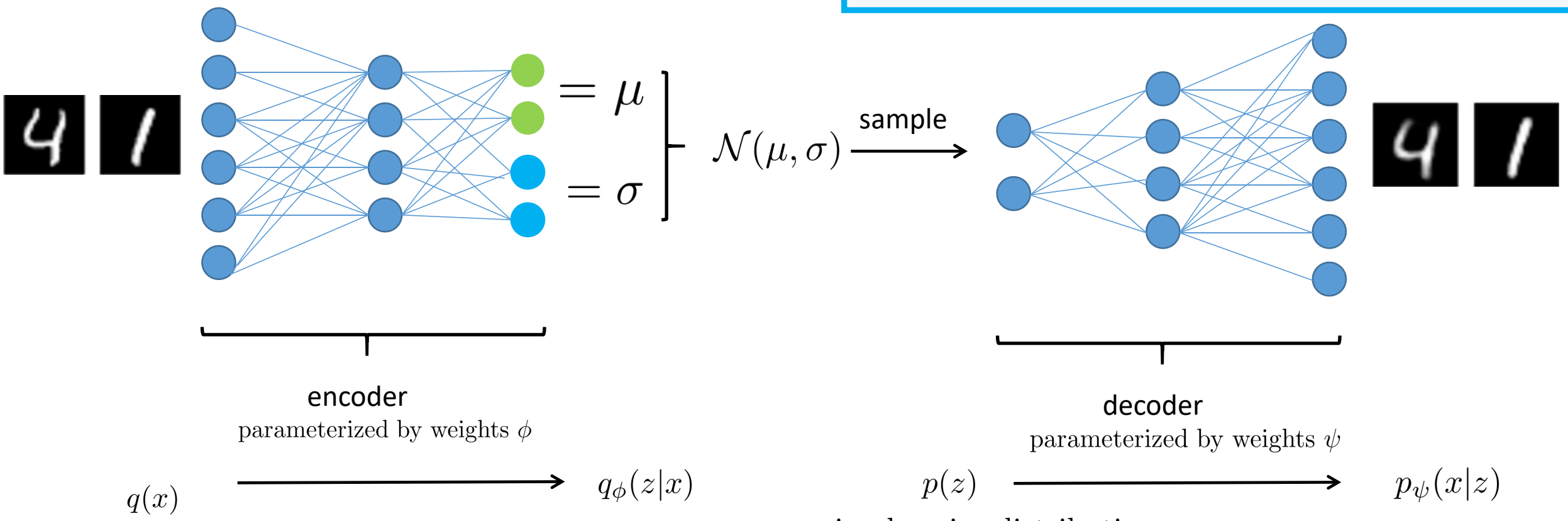


Variational Autoenoder

The model - Combining a probabistic mo

Training objective:

$$\underbrace{-\mathbb{E}_{z\sim q_\theta(z|x_i)}\left[\log p_\phi(x_i|z)\right]}_{\text{reconstruction loss}} + \underbrace{\mathbb{KL}(q_\theta(z|x_i)||p(z))}_{regularizer}$$



Variational Autoenoder

The objective

$$\mathbb{KL}(q_\theta(z|x_i)||p(z)) = -\int_{z\in Z} q_\theta(z|x_i)\log\left(\frac{p(z)}{q_\theta(z|x_i)}\right)$$

Training objective:

$$\underbrace{-\mathbb{E}_{z\sim q_\theta(z|x_i)}\left[\log p_\phi(x_i|z)\right]}_{\text{reconstruction loss}} + \underbrace{\mathbb{KL}(q_\theta(z|x_i)||p(z))}_{regularizer}$$

Variational Autoenoder

The objective

Training objective:

$$\underbrace{-\mathbb{E}_{z\sim q_\theta(z|x_i)}\left[\log p_\phi(x_i|z)\right]}_{\text{reconstruction loss}} + \underbrace{\mathbb{KL}(q_\theta(z|x_i)||p(z))}_{regularizer}$$

$$\mathbb{KL}(q_\theta(z|x_i)||p(z)) = -\int_{z\in Z} q_\theta(z|x_i)\log\left(\frac{p(z)}{q_\theta(z|x_i)}\right)$$

With $p(z) = \mathcal{N}(0,1)$ unit variance Gaussian around 0 this is analytically solvable.

$$= -\frac{1}{2}\ln(\sigma(x_i)) + \frac{1}{2}\sigma(x_i)^2 + \frac{1}{2}\mu(x_i)^2 - \frac{1}{2}$$

Variational Autoenoder

The objective

Training objective:

$$\underbrace{-\mathbb{E}_{z\sim q_\theta(z|x_i)}\left[\log p_\phi(x_i|z)\right]}_{\text{reconstruction loss}} + \underbrace{\mathbb{KL}(q_\theta(z|x_i)||p(z))}_{regularizer}$$

$$\mathbb{KL}(q_\theta(z|x_i)||p(z)) = -\int_{z\in Z} q_\theta(z|x_i)\log\left(\frac{p(z)}{q_\theta(z|x_i)}\right)$$

With $p(z) = \mathcal{N}(0,1)$ unit variance Gaussian around 0 this is analytically solvable.

$$= -\frac{1}{2}\ln(\sigma(x_i)) + \frac{1}{2}\sigma(x_i)^2 + \frac{1}{2}\mu(x_i)^2 - \frac{1}{2}$$

For the reconstruction loss, we take training sample x_i and feed it through the network to obtain $\hat{x}_i$.

By modelling $p_\phi(x_i|z) = \exp\left(-||dec(z) - x_i||^2\right)$ our loss for sample x_i turns into $||\hat{x}_i - x_i||^2$

Variational Autoenoder

The objective

Training objective:

$$\underbrace{-\mathbb{E}_{z\sim q_\theta(z|x_i)}\left[\log p_\phi(x_i|z)\right]}_{\text{reconstruction loss}} + \underbrace{\mathbb{KL}(q_\theta(z|x_i)||p(z))}_{regularizer}$$

$$\mathbb{KL}(q_\theta(z|x_i)||p(z)) = -\int_{z\in Z} q_\theta(z|x_i)\log\left(\frac{p(z)}{q_\theta(z|x_i)}\right)$$

With $p(z) = \mathcal{N}(0,1)$ unit variance Gaussian around 0 this is analytically solvable.

$$= -\frac{1}{2}\ln(\sigma(x_i)) + \frac{1}{2}\sigma(x_i)^2 + \frac{1}{2}\mu(x_i)^2 - \frac{1}{2}$$

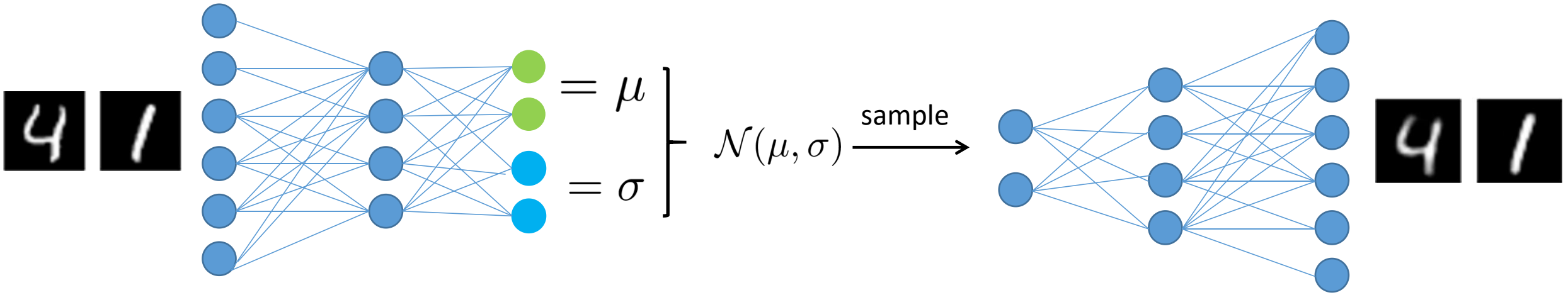
For the reconstruction loss, we take training sample x_i and feed it through the network to obtain $\hat{x}_i$.

By modelling $p_\phi(x_i|z) = \exp\left(-||dec(z) - x_i||^2\right)$ our loss for sample x_i turns into $||\hat{x}_i - x_i||^2$

$$\sum_{x_i} ||x_i - dec(enc(x_i))||^2 - \frac{1}{2}\ln\left(z_\sigma(x_i)\right) + \frac{1}{2}z_\sigma(x_i)^2 + \frac{1}{2}z_\mu(x_i)^2 - \frac{1}{2}$$

Variational Autoenoder

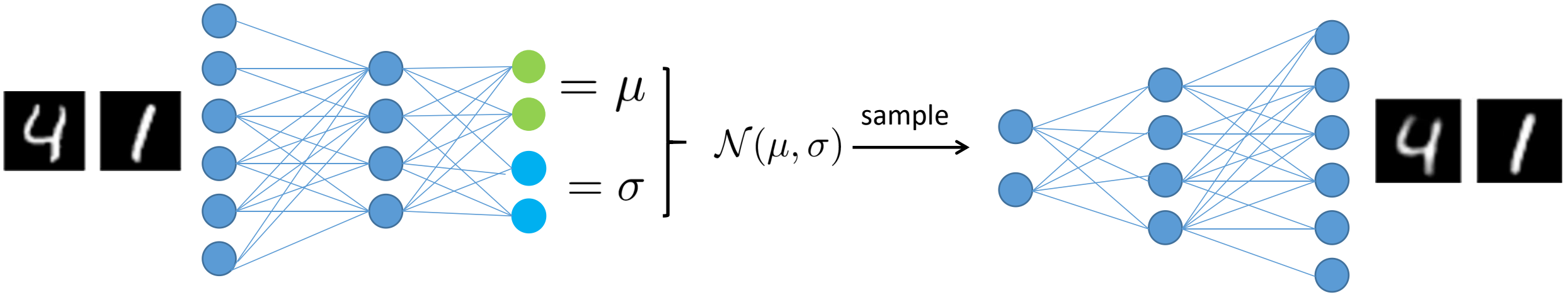
Training VAEs



But how to train it?

Variational Autoenoder

Training VAEs

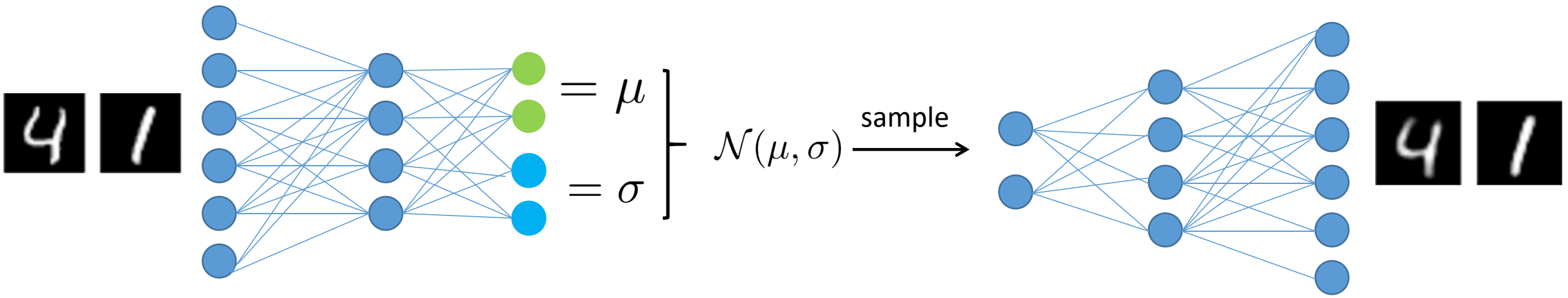


But how to train it?

Problem: We cannot do backpropagation through the sampling step.

Variational Autoenoder

Training VAEs



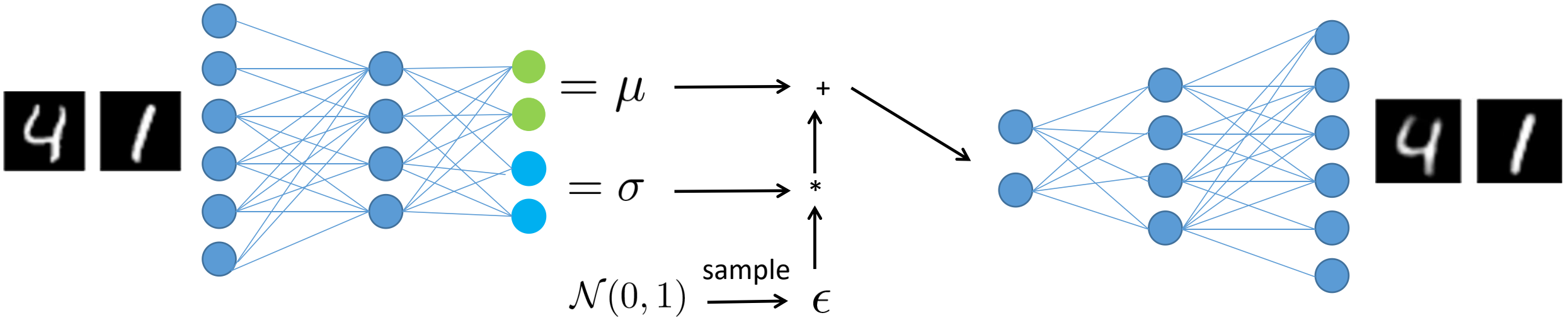
But how to train it?

Problem: We cannot do backpropagation through the sampling step.

Reparameterization Trick $\mathcal{N}(\mu, \sigma) = \mu + \sigma \cdot \mathcal{N}(0, 1)$

Variational Autoenoder

Training VAEs



But how to train it?

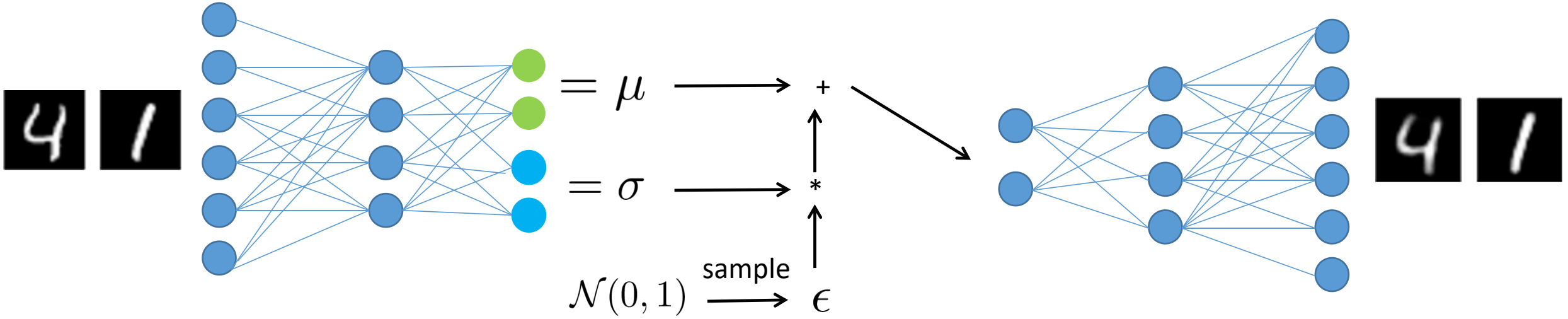
Problem: We cannot do backpropagation through the sampling step.

Reparameterization Trick $\mathcal{N}(\mu, \sigma) = \mu + \sigma \cdot \mathcal{N}(0,1)$

Variational Autoenoder

Training VAEs

No more need to backprop through sampling step



But how to train it?

Problem: We cannot do backpropagation through the sampling step.

Reparameterization Trick $\mathcal{N}(\mu, \sigma) = \mu + \sigma \cdot \mathcal{N}(0,1)$

Generative models

A good objective for generating real-looking samples?

Both Autoencoder and Variational Autoencoder are trained with L2 (sometimes L1) loss, which determines what it means for the reconstructed image to be similar to the original image.

Do L1 or L2 losses adequately model image similarity? if an image is shifted 10 pixels to the right, is it the same image? We really want to capture more high level information of the image rather than just comparing pixel values.

Also, is the idea of reproducing the original the best method to learn a generative model?

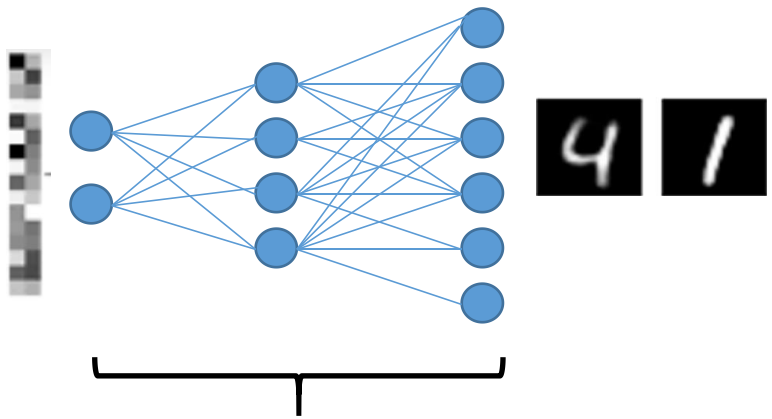
Let us concentrate on the generator network.

Generative models

A good objective for generating real-looking samples?

For the generator, we want:

- Ability to generate „realistic“ data from random noise. A new random input should give a new realistic image.
- To train the network, we need to decide on a loss function that measures how „realistic“ the generated sample is

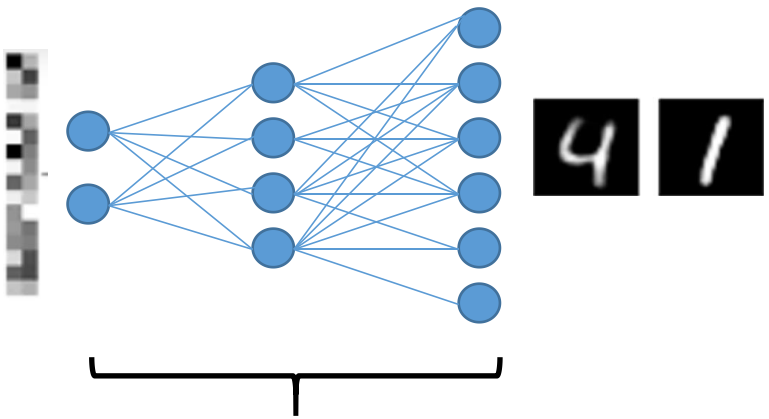


Generative models

A good objective for generating real-looking samples?

For the generator, we want:

- Ability to generate „realistic“ data from random noise. A new random input should give a new realistic image.
- To train the network, we need to decide on a loss function that measures how „realistic“ the generated sample is
- We assume we have real data to compare with.
- What are good features to compare real with generated images?



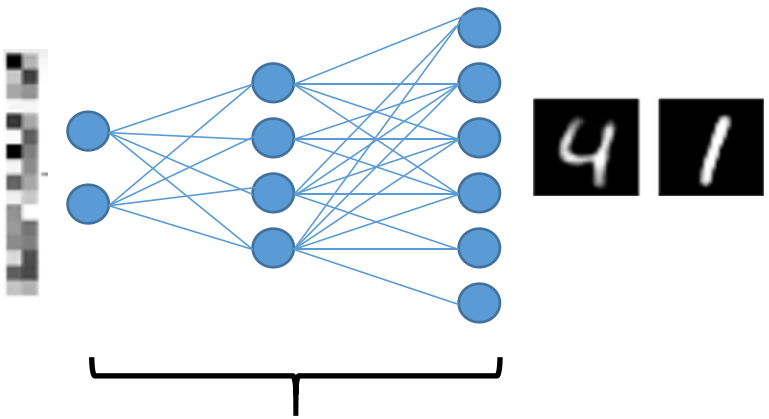
Generative models

A good objective for generating real-looking samples?

For the generator, we want:

- Ability to generate „realistic“ data from random noise. A new random input should give a new realistic image.
- To train the network, we need to decide on a loss function that measures how „realistic“ the generated sample is
- We assume we have real data to compare with.
- What are good features to compare real with generated images?

  Very hard task!
  What was the trick to use when it is hard to create good features manually?

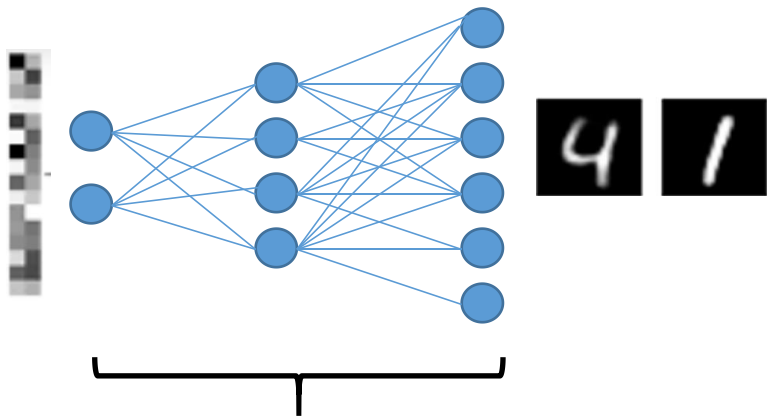


Generative models

A good objective for generating real-looking samples?

For the generator, we want:

- Ability to generate „realistic“ data from random noise. A new random input should give a new realistic image.
- To train the network, we need to decide on a loss function that measures how „realistic“ the generated sample is
- We assume we have real data to compare with.
- What are good features to compare real with generated images?



Very hard task!
What was the trick to use when it is hard to create good features manually?

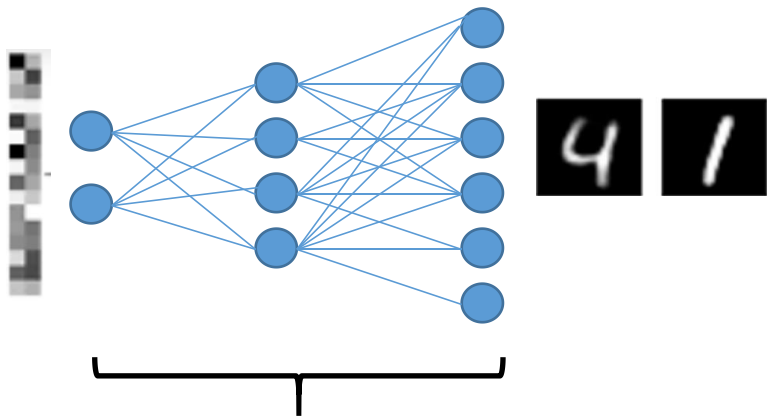
Let a neural network learn what good features are.

Generative models

A good objective for generating real-looking samples?

For the generator, we want:

- Ability to generate „realistic“ data from random noise. A new random input should give a new realistic image.
- To train the network, we need to decide on a loss function that measures how „realistic“ the generated sample is
- We assume we have real data to compare with.
- What are good features to compare real with generated images?



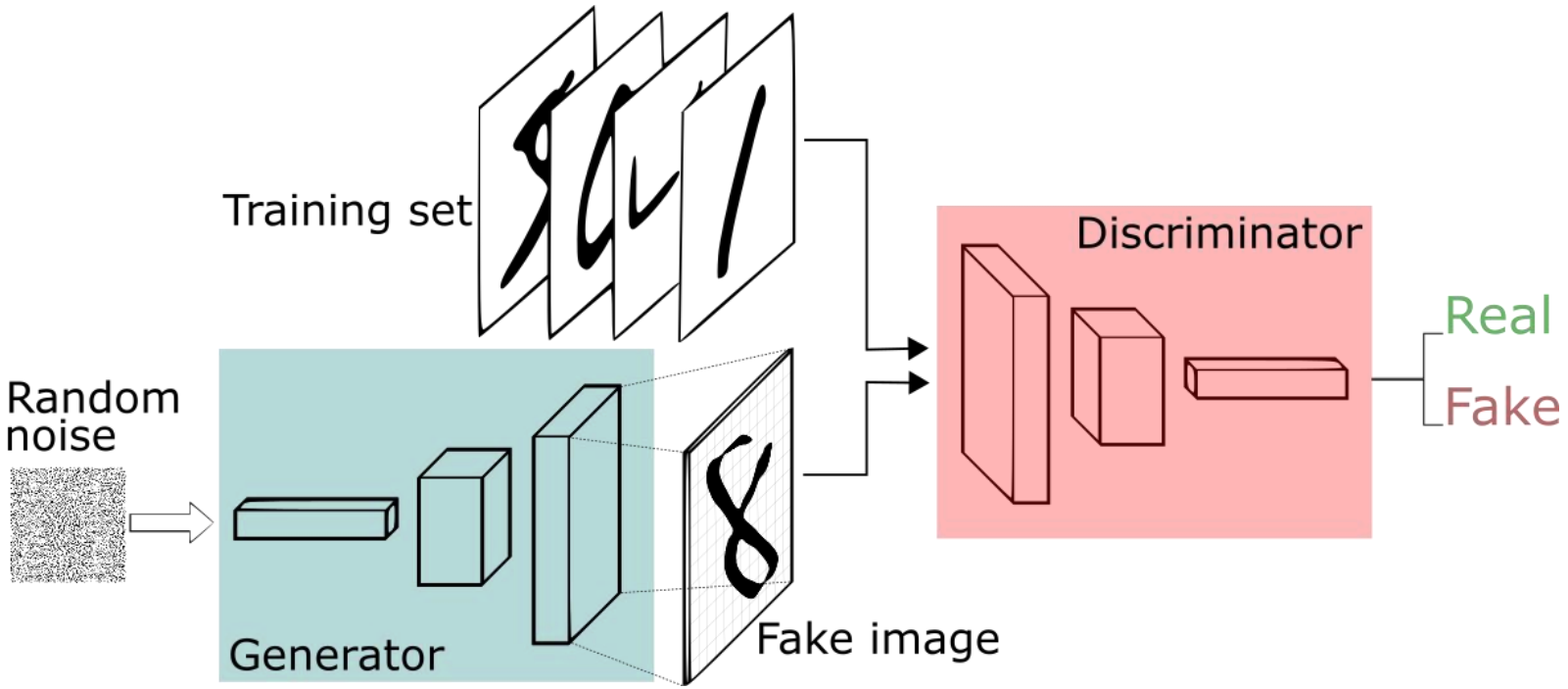
Very hard task!
What was the trick to use when it is hard to create good features manually?

Let a neural network learn what good features are.

Here: Let the neural network learn a good loss function for the generator

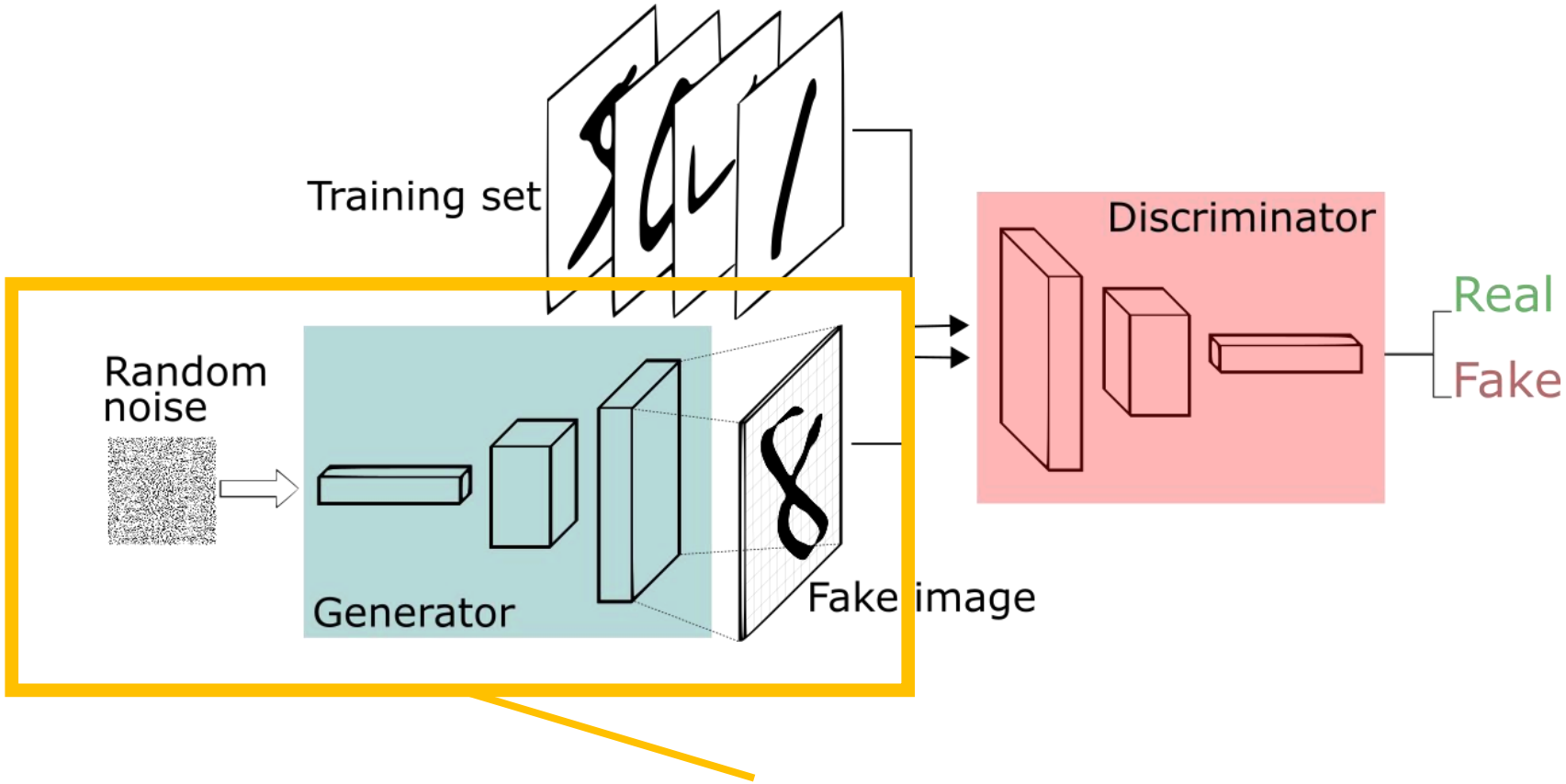
Generative adversarial networks (GANs)

The model



Generative adversarial networks (GANs)

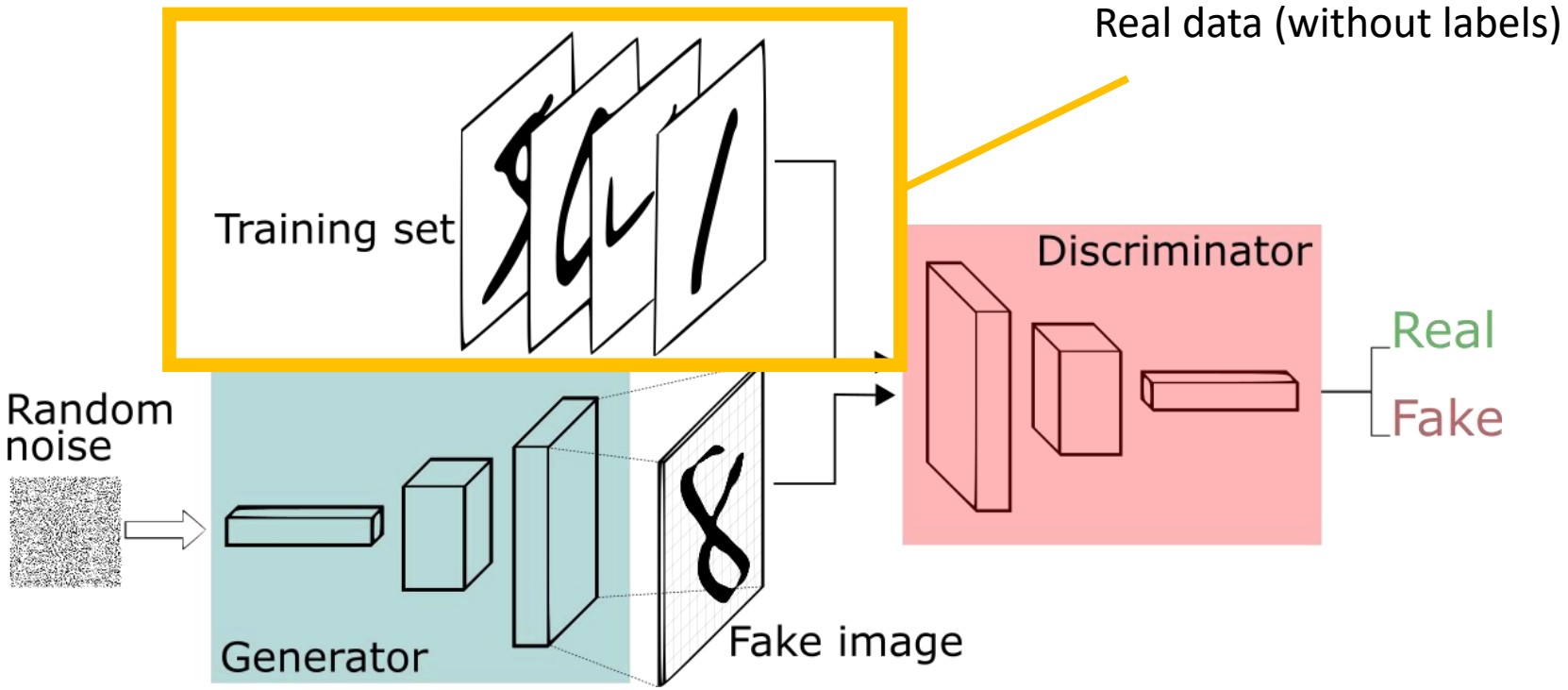
The model



Generator network. Maps latent noise to image.
Noise samples are generated from a Gaussian or uniform distribution.

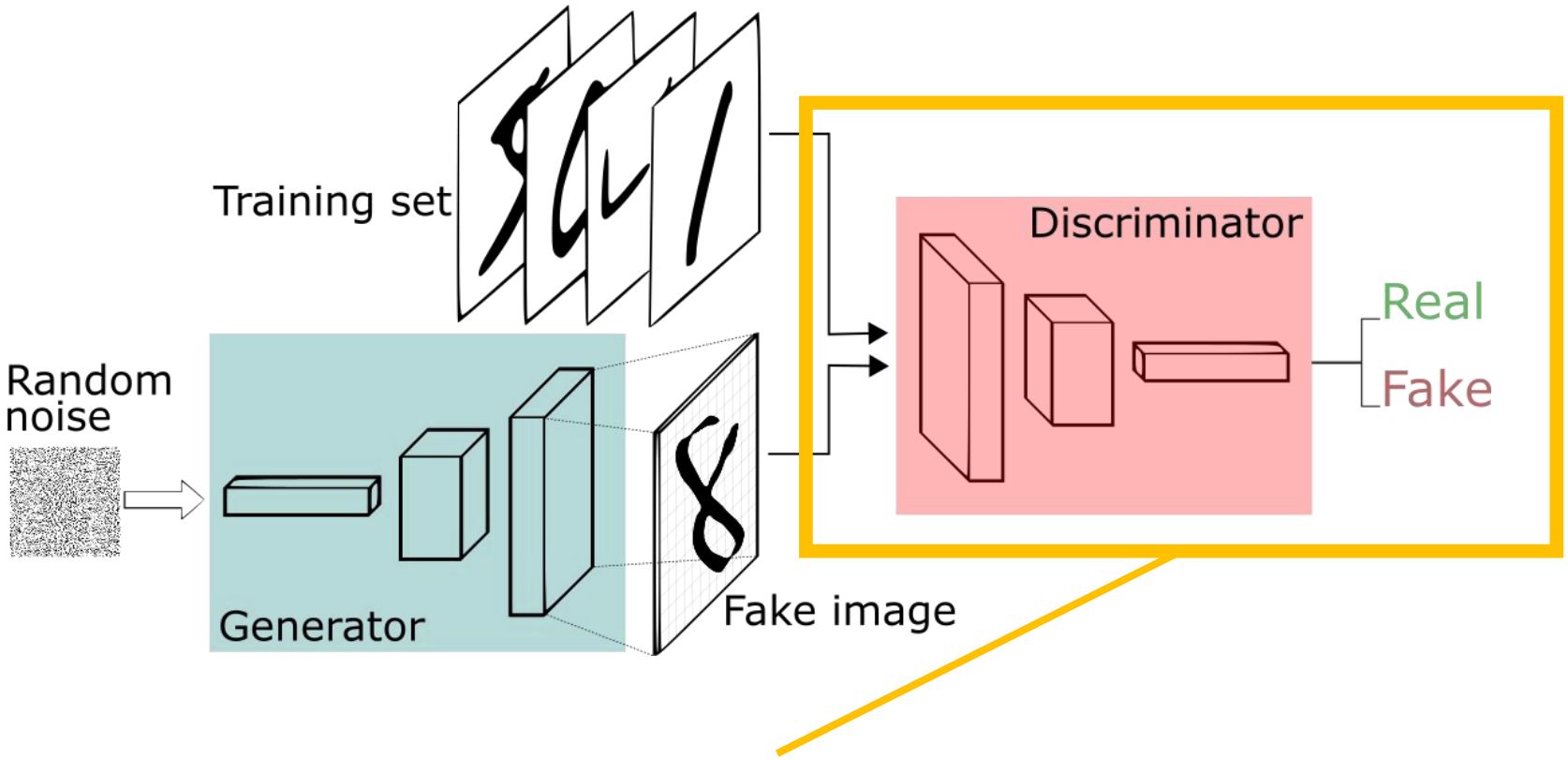
Generative adversarial networks (GANs)

The model



Generative adversarial networks (GANs)

The model



Discriminator network. Has the task to tell real images and generated/fake images apart.

Generative adversarial networks (GANs)

The loss function

Generator network and discriminator network with competing objectives.

Discriminator aims to assign high values (≈1) to real images and low values (≈0) to fake images.

$$\max_D \left(\mathbb{E}_{x \sim p_{data}(x)} [\log(D(x))] + \mathbb{E}_{x \sim p_{gen}(x)} [\log(1 - D(x))] \right)$$

Generative adversarial networks (GANs)

The loss function

Generator network and discriminator network with competing objectives.

Discriminator aims to assign high values (≈1) to real images and low values (≈0) to fake images.

$$\max_D \left(\mathbb{E}_{x \sim p_{data}(x)} [\log(D(x))] + \mathbb{E}_{x \sim p_{gen}(x)} [\log(1 - D(x))] \right)$$

Discriminator output in interval [0,1] (e.g. by sigmoid activation function at output).

Generative adversarial networks (GANs)

The loss function

Generator network and discriminator network with competing objectives.

Discriminator aims to assign high values (≈1) to real images and low values (≈0) to fake images.

$$\max_D \left(\mathbb{E}_{x \sim p_{data}(x)} [\log(D(x))] + \mathbb{E}_{x \sim p_{gen}(x)} [\log(1 - D(x))] \right)$$

Generator aims to fool the discriminator and obtain high values (≈1) from discriminator:

$$\min_G \mathbb{E}_{z \sim p(z)} [\log(1 - D(G(z)))]$$

Generative adversarial networks (GANs)

The loss function

Generator network and discriminator network with competing objectives.

Discriminator aims to assign high values (≈1) to real images and low values (≈0) to fake images.

$$\max_D \left(\mathbb{E}_{x \sim p_{data}(x)} [\log(D(x))] + \mathbb{E}_{x \sim p_{gen}(x)} [\log(1 - D(x))] \right)$$

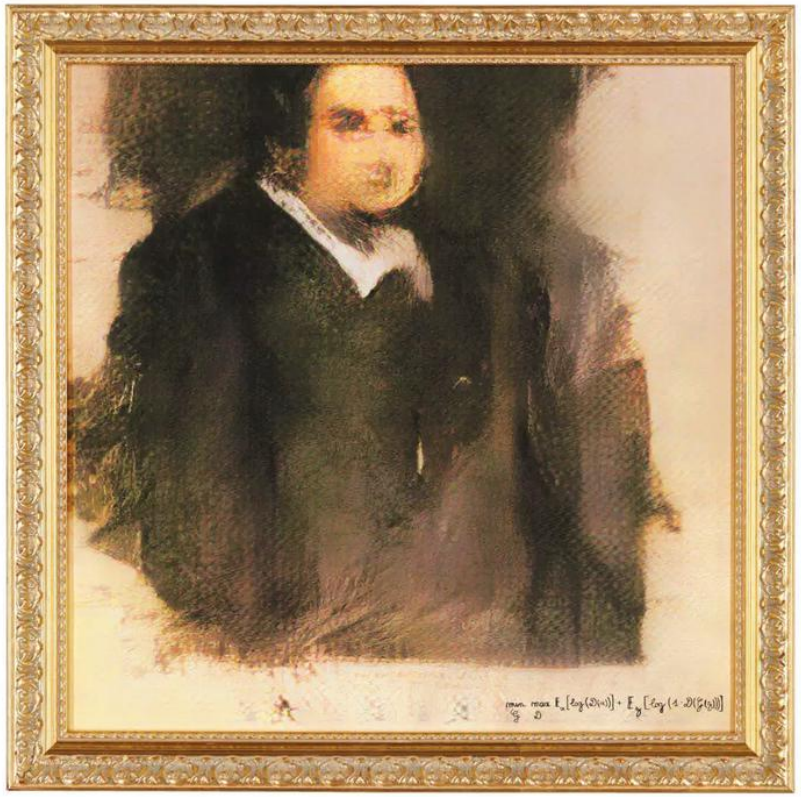
Generator aims to fool the discriminator and obtain high values (≈1) from discriminator:

$$\min_G \mathbb{E}_{z \sim p(z)} [\log(1 - D(G(z)))]$$

Combined objective:

$$\min_G \max_D \left(\mathbb{E}_{x \sim p_{data}(x)} [\log(D(x))] + \mathbb{E}_{z \sim p(z))} [\log(1 - D(G(z)))] \right)$$

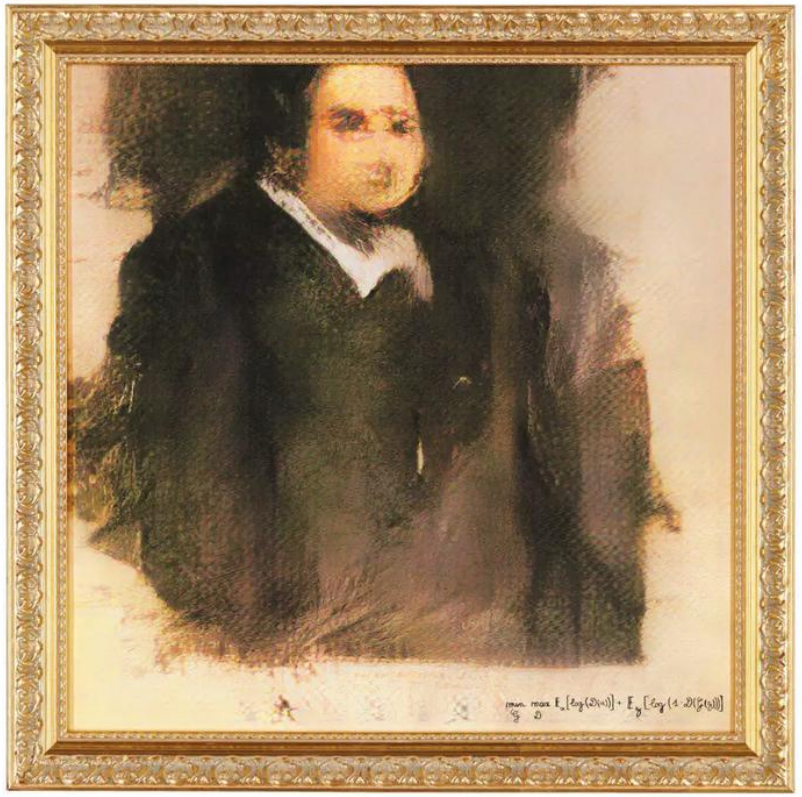
Goodfellow, Pouget-Abadie, Mirza, Xu, Warde-Farley, Ozair, Courville, Bengio (2014). Generative Adversarial Nets. NIPS 2014



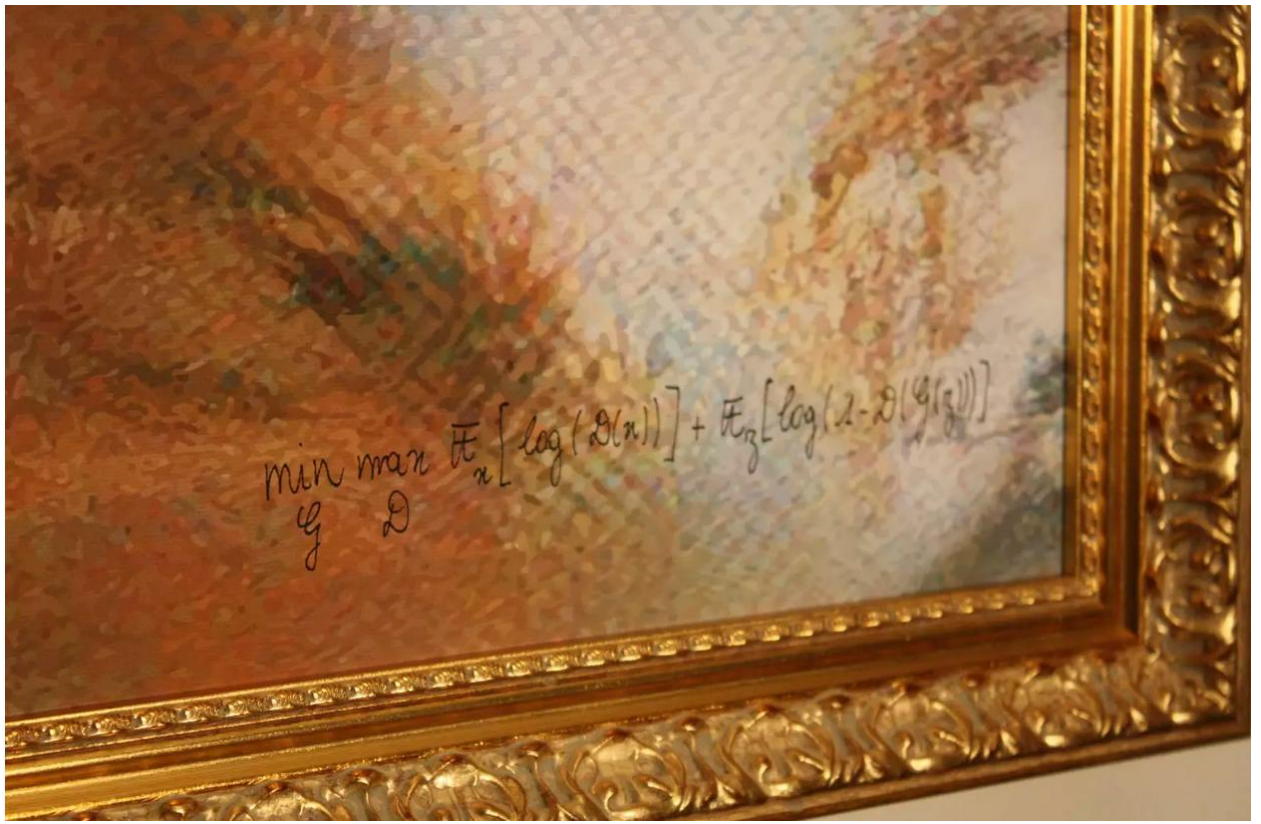
„Artist“ Belamy

$$\min_G \max_D \left(\mathbb{E}_{x \sim p_{data}(x)} [\log(D(x))] + \mathbb{E}_{z \sim p(z))} [\log(1 - D(G(z)))] \right)$$

Goodfellow, Pouget-Abadie, Mirza, Xu, Warde-Farley, Ozair, Courville, Bengio (2014). Generative Adversarial Nets. NIPS 2014



„Artist“ Belamy



$$\min_G \max_D \left(\mathbb{E}_{x \sim p_{data}(x)}[\log(D(x))] + \mathbb{E}_{z \sim p(z))}[\log(1 - D(G(z)))] \right)$$

Goodfellow, Pouget-Abadie, Mirza, Xu, Warde-Farley, Ozair, Courville, Bengio (2014). Generative Adversarial Nets. NIPS 2014

Generative adversarial networks (GANs)

How to train them?

$$\min_G \max_D \left(\mathbb{E}_{x \sim p_{data}(x)} [\log(D(x))] + \mathbb{E}_{z \sim p(z))} [\log(1 - D(G(z)))] \right)$$

Loss function of form min-max from competing objectives:

Train iteratively:

- Perform n update steps of **gradient ascent** for parameters for **discriminator** network D.
- Perform n update steps of **gradient descent** for parameters for **generator** network **G**.

Generative adversarial networks (GANs)

How to train them?

$$\min_G \max_D \left(\mathbb{E}_{x \sim p_{data}(x)} [\log(D(x))] + \mathbb{E}_{z \sim p(z))} [\log(1 - D(G(z)))] \right)$$

Loss function of form min-max from competing objectives:

Train iteratively:

- Perform n update steps of **gradient ascent** for parameters for **discriminator** network D.
- Perform n update steps of **gradient descent** for parameters for **generator** network **G**.

<u>In theory:</u>

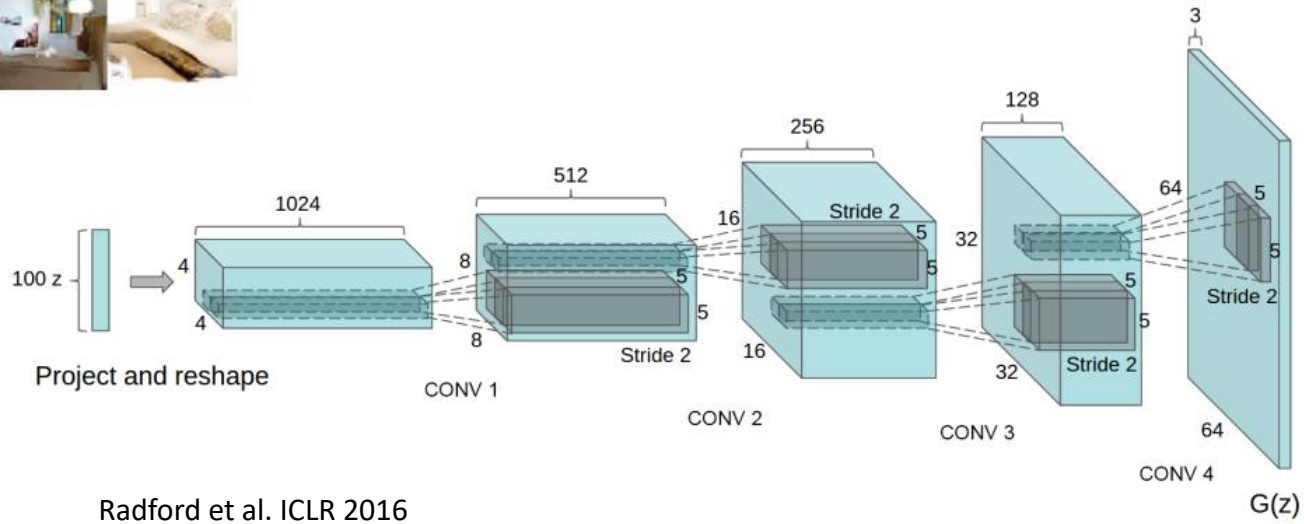
If G and D have enough capacity, and D converges to a maximum of its objective function, then G will converge to $p_g = p_d$, i.e. the generator will sample from the true data distribution.

<u>In practice:</u>

Reputation of being hard to train. Many tricks of the trade. Many more stable variants.
When they work, very impressive results can be achieved.

DCGAN

A competitive GAN network with convolutions



Radford et al. ICLR 2016

ProGAN

A competitive GAN network by NVIDIA

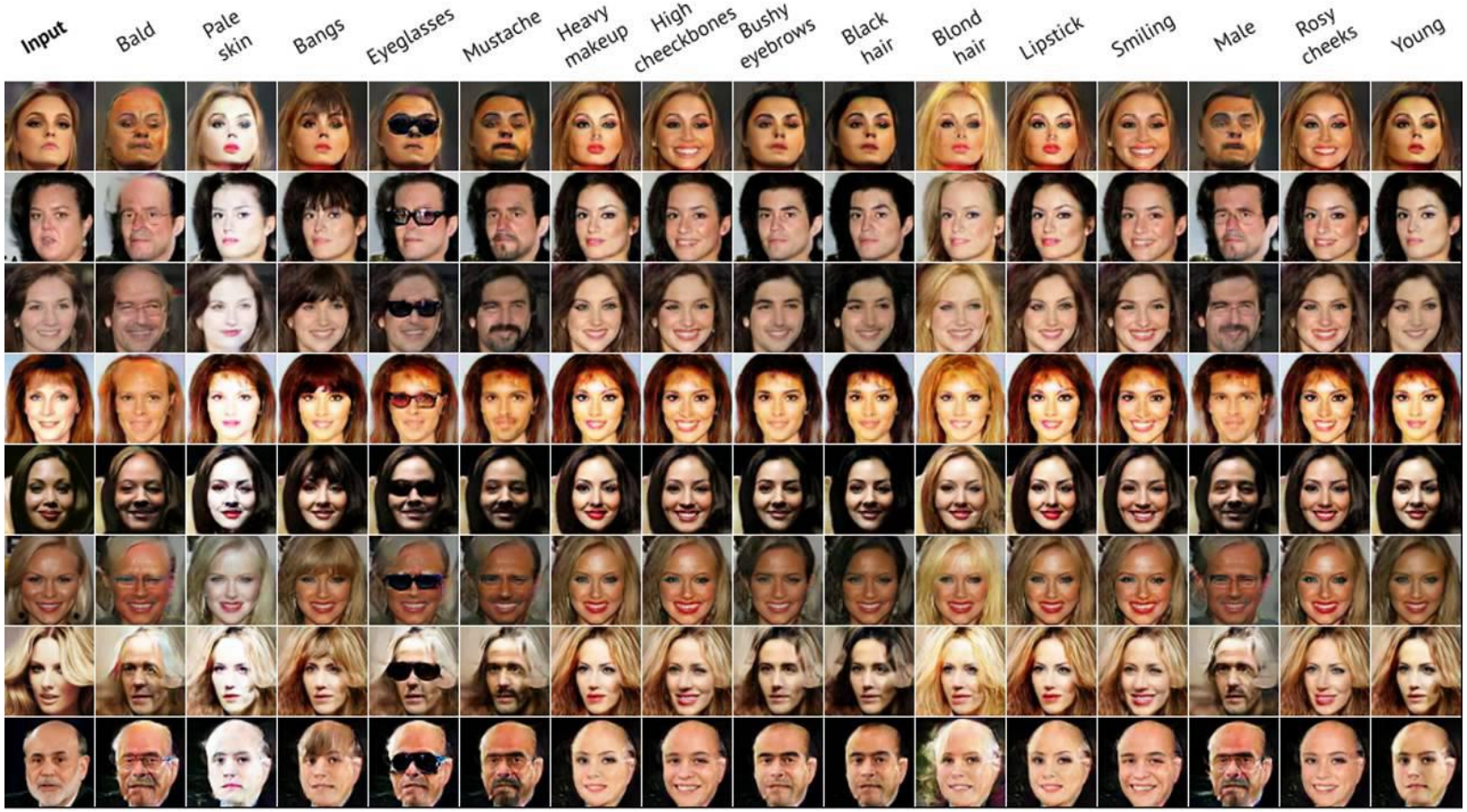
Images of Faces generated by a GAN (State of the art)

One hour video of generated images:

https://www.youtube.com/watch?v=36lE9tV9vm0

GAN

Latent space exploration



GAN application

Image to Image translation

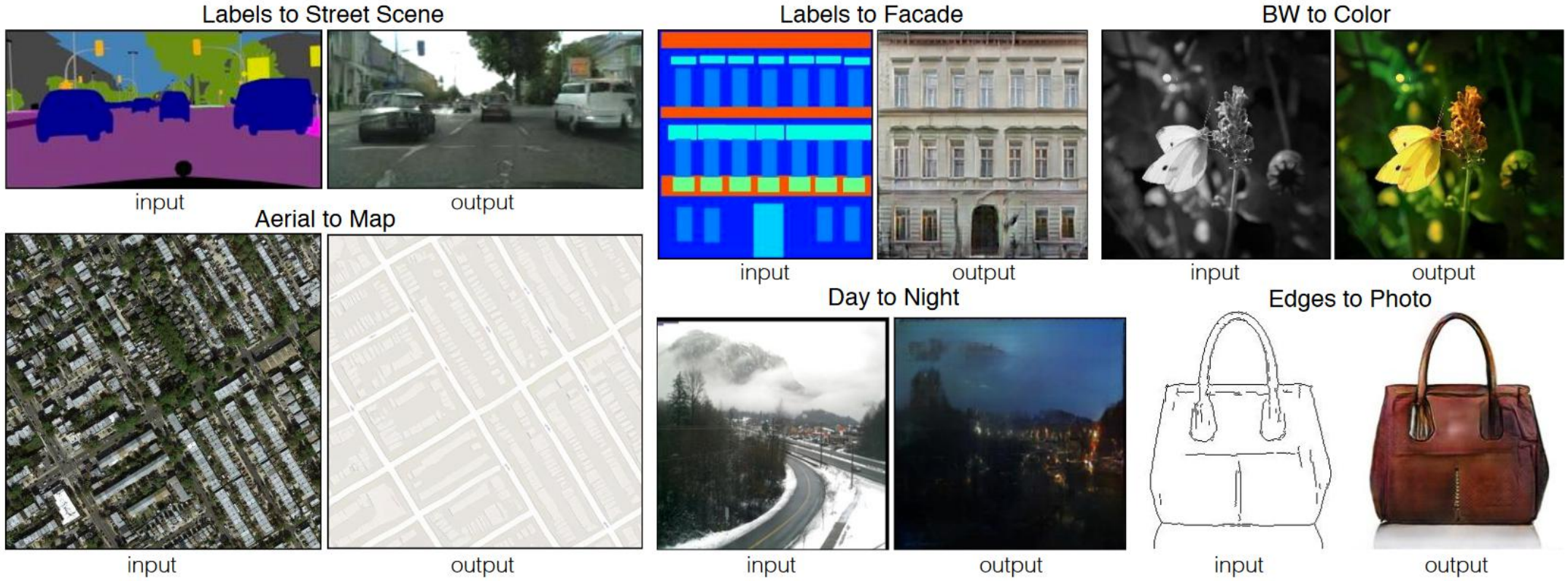
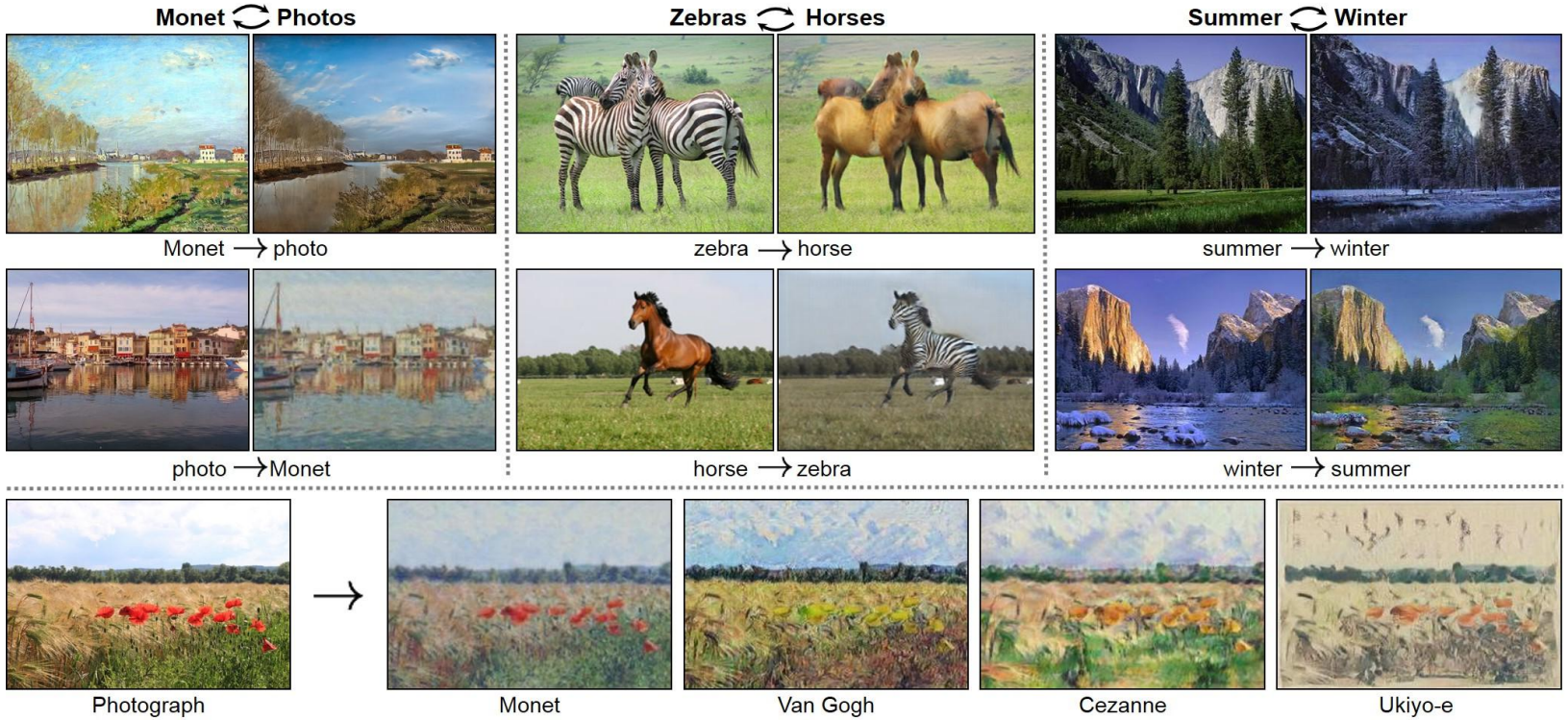


Image-to-Image Translation with Conditional Adversarial Networks, Isola et al. CVPR 2017

GAN application

CycleGAN



GAN application

Pose to photo

Pose Guided Person Image Generation, Ma et al. NIPS 2017

GAN application

GANimation