Mathematics of Deep Learning

Generative models

Lessons: Kevin Scaman

Dauphine | PSL
UNIVERSITÉ PARIS

Class overview

Exam

1. Next week (19/03/2023).
2. Documents allowed.
3. From 8:30am to 10:30am (2h).
4. Similar to the homework.

Generative models

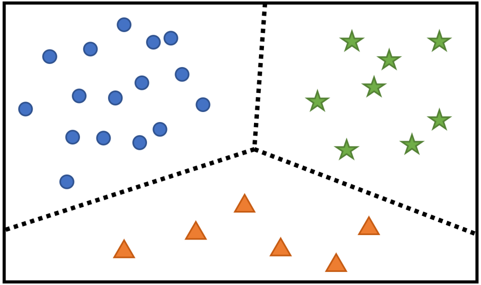
Beyond classification tasks

What is a generative model?

Generative vs. discriminative

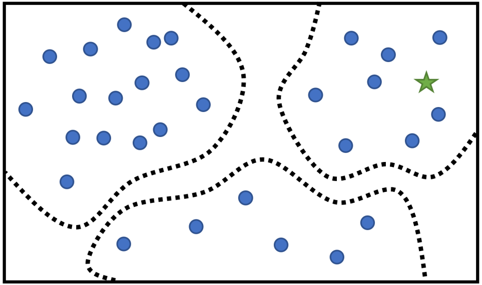
- Discriminative tasks such as classification aim at **separating** data.
- Generative tasks aim at **creating** new data.

Discriminative tasks



Classification (access to (X,y) pairs)

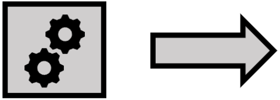
Generative tasks



Sampling (access to X only)

Examples of generative models

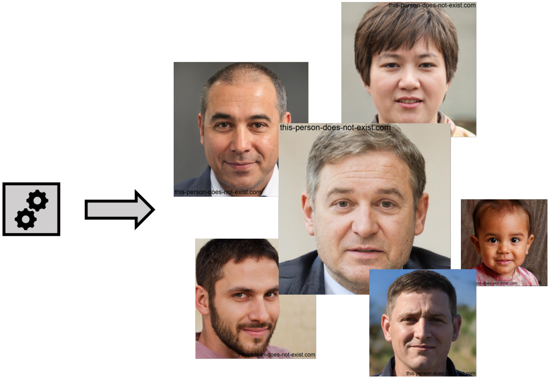
- Image generation (face generation, deepfakes, ...).



source: `https://this-person-does-not-exist.com/en`

Examples of generative models

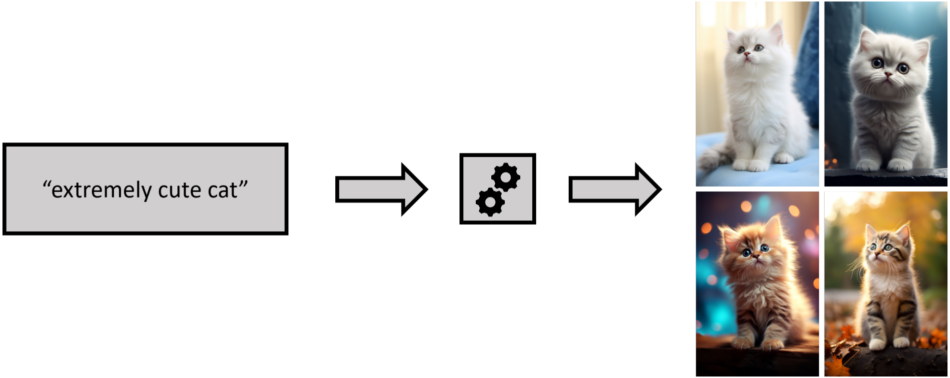
- Image generation (face generation, deepfakes, ...).



source: `https://this-person-does-not-exist.com/en`

Examples of generative models

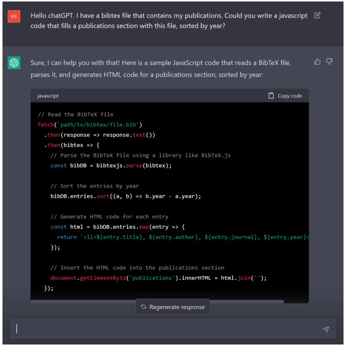
- Image generation (face generation, deepfakes, ...).
- Prompt-based image generation (Dalle2, Imagen, MidjourneyAI, ...).



source: MidjourneyAI. `https://midjourney.com/`

Examples of generative models

- Image generation (face generation, deepfakes, ...).
- Prompt-based image generation (Dalle2, Imagen, MidjourneyAI, ...).
- Text generation (Bert, GPT2, GPT3, ChatGPT, Bard, Sparrow, ...).



source: ChatGPT. `https://chat.openai.com/`

Neural architectures for generative tasks

Key aspects of a generative model

- We want to **output complex data** (e.g. images, text, ...).
- We want to **sample random outputs** from a learnt distribution.
- Usually involves more **difficult optimization problems** than standard ERM.
- How do we measure **performance**?

Neural architectures for generative tasks

Key aspects of a generative model

- We want to **output complex data** (e.g. images, text, ...).
- We want to **sample random outputs** from a learnt distribution.
- Usually involves more **difficult optimization problems** than standard ERM.
- How do we measure **performance**?

Three main approaches

1. Variational auto-encoders (VAEs)
2. Generative Adversarial Networks (GANs)
3. Score-based generative models / diffusion models

Generating random variables

Classical approaches to sampling probability distributions

Generative models

Approximating distributions with NNs

- **Intuition:** How do we create models whose outputs are **random variables**?

Generative models

Approximating distributions with NNs

- **Intuition:** How do we create models whose outputs are **random variables**?
- **Data:** $\hat{\mathcal{D}}_n = (X_1, \ldots, X_n)$ i.i.d. according to some target distribution $\mathcal{D}$.

Generative models

Approximating distributions with NNs

- **Intuition:** How do we create models whose outputs are **random variables**?
- **Data:** $\hat{\mathcal{D}}_n = (X_1, \ldots, X_n)$ i.i.d. according to some target distribution $\mathcal{D}$.
- **Objective:** sample new elements $\tilde{X} \sim \mathcal{D}$ from the target distribution.

Generative models

Approximating distributions with NNs

- **Intuition:** How do we create models whose outputs are **random variables**?
- **Data:** $\hat{\mathcal{D}}_n = (X_1, \ldots, X_n)$ i.i.d. according to some target distribution $\mathcal{D}$.
- **Objective:** sample new elements $\tilde{X} \sim \mathcal{D}$ from the target distribution.

Extensions

- **Prompt-based models:** one data distribution per input query. Equivalent to supervised learning with a random output.

Generative models

Approximating distributions with NNs

- **Intuition:** How do we create models whose outputs are **random variables**?
- **Data:** $\hat{\mathcal{D}}_n = (X_1, \ldots, X_n)$ i.i.d. according to some target distribution $\mathcal{D}$.
- **Objective:** sample new elements $\tilde{X} \sim \mathcal{D}$ from the target distribution.

Extensions

- **Prompt-based models:** one data distribution per input query. Equivalent to supervised learning with a random output.
- **Learn a density function:** some models also provide a density function.

No clear cut: classification tasks also generate probability distributions...

How to sample from a known distribution $\mathcal{D}$?

How to sample from a known distribution $\mathcal{D}$?

Standard approaches

- **Parametric families of distributions:** sampled by a simple function of a base distribution. E.g. Gaussian $X = \mu + \sigma Y$ where $Y \sim \mathcal{N}(0,1)$.

How to sample from a known distribution $\mathcal{D}$?

Standard approaches

- **Parametric families of distributions:** sampled by a simple function of a base distribution. E.g. Gaussian $X = \mu + \sigma Y$ where $Y \sim \mathcal{N}(0, 1)$.
- **Inversion sampling:** For 1D r.v., we have $X = F^{-1}(Y)$ where $Y \sim \mathcal{U}([0, 1])$ is uniform in $[0, 1]$ and F is the cumulative distribution function of $\mathcal{D}$.

How to sample from a known distribution $\mathcal{D}$?

Standard approaches

- **Parametric families of distributions:** sampled by a simple function of a base distribution. E.g. Gaussian $X = \mu + \sigma Y$ where $Y \sim \mathcal{N}(0,1)$.
- **Inversion sampling:** For 1D r.v., we have $X = F^{-1}(Y)$ where $Y \sim \mathcal{U}([0,1])$ is uniform in $[0,1]$ and F is the cumulative distribution function of $\mathcal{D}$.
- **Monte-Carlo Markov Chains:** Start with a base distribution (e.g. Gaussian), and iteratively refine it to get closer and closer to the target distribution $\mathcal{D}$.

How to sample from a known distribution $\mathcal{D}$?

Standard approaches

- **Parametric families of distributions:** sampled by a simple function of a base distribution. E.g. Gaussian $X = \mu + \sigma Y$ where $Y \sim \mathcal{N}(0, 1)$.
- **Inversion sampling:** For 1D r.v., we have $X = F^{-1}(Y)$ where $Y \sim \mathcal{U}([0, 1])$ is uniform in $[0, 1]$ and F is the cumulative distribution function of $\mathcal{D}$.
- **Monte-Carlo Markov Chains:** Start with a base distribution (e.g. Gaussian), and iteratively refine it to get closer and closer to the target distribution $\mathcal{D}$.

How to use it for generative models?

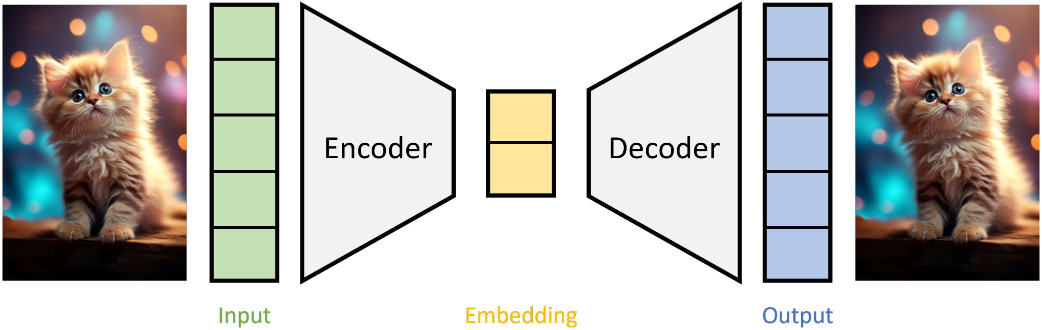
- **Parameter modelling:** Learn the parameters $(\mu, \sigma) = g_\theta(x)$ to generate $\mathcal{N}(\mu, \sigma)$.
- **Transformation:** generate with $g_\theta(Y) \sim \mathcal{D}$ where $Y \sim \mathcal{N}(0, I)$ (**VAEs**, **GANs**).
- **Dynamics:** Learn iterative refinements that transform $\mathcal{N}(0, I)$ into $\mathcal{D}$ (**diffusion**).

Variational Autoencoders (VAEs)

From compression to generation

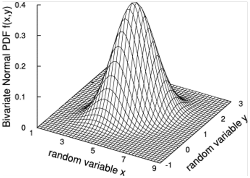
But first... what is an autoencoder?

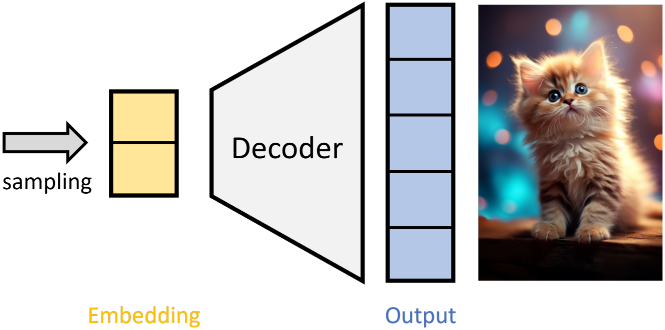
- **Objective:** Learn a **compressed data representation** in an unsupervised manner.
- **Idea:** Map **data points to themselves** $g_\theta(x) = x$ with **small inner representation**.
- **Loss:** Let $e_\theta, d_{\theta'}$ be two NNs, we want to minimize $\mathbb{E}(\|X - d_{\theta'}(e_\theta(X))\|^2)$.



But first... what is an autoencoder?

- **Compression:** If latent space is smaller than input space, information is **compressed**.
- **Generation:** We can sample from the **latent space**.





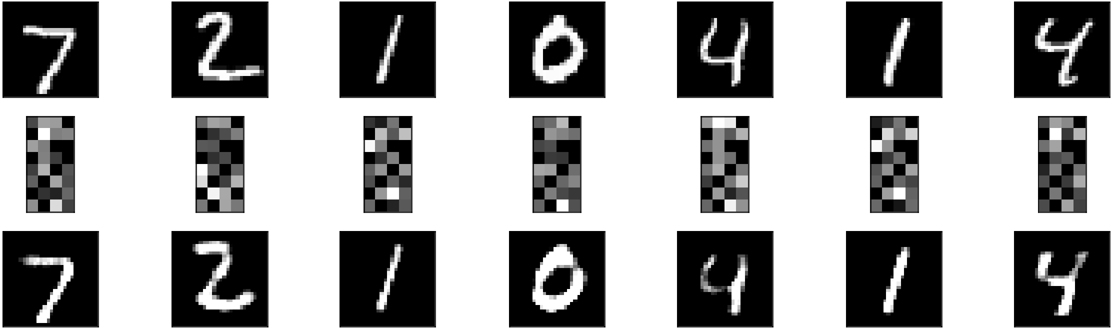
Autoencoders in PyTorch

The simplest possible autoencoder with a **single affine layer** as encoder and as decoder:

```
class AutoEncoder(nn.Module):
    def __init__(self, input_dim, encoding_dim):
        super(AutoEncoder, self).__init__()
        self.encoder = nn.Linear(input_dim, encoding_dim)
        self.decoder = nn.Linear(encoding_dim, input_dim)
    def forward(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded
```

Autoencoders in PyTorch

After training, we obtain:

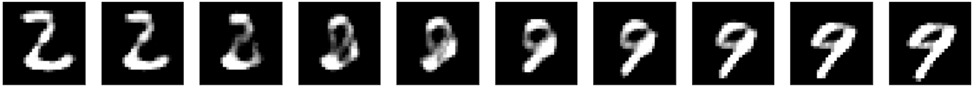


Representation learning with autoencoders

- **Interpolation in latent space:** We can interpolate between two images x and y with

$$x_\alpha \quad = \quad d_{\theta'}\Big(\alpha\, e_\theta(x) + (1-\alpha)\, e_\theta(y)\Big)$$

  for $\alpha \in [0, 1]$.
- **Results:** Interpolation between digits 2 and 9.



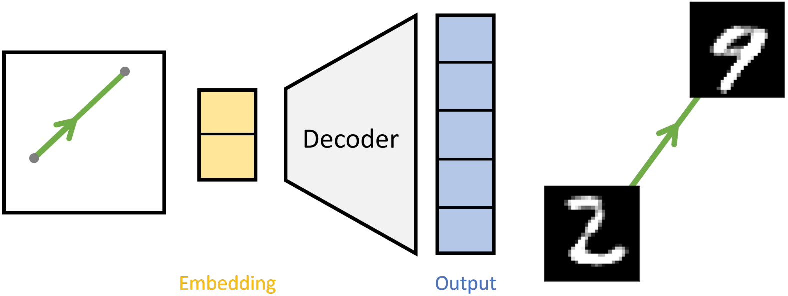
- Better than in the **pixel space**, but not perfect still...

Representation learning with autoencoders

- **Interpolation in latent space:** We can interpolate between two images x and y with

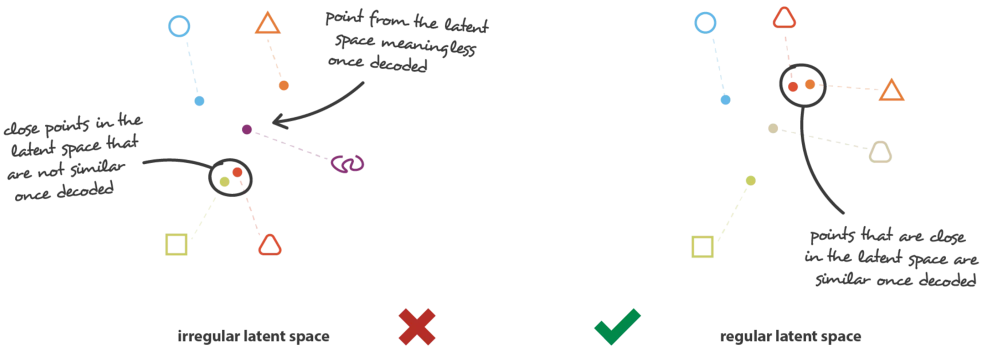
$$x_\alpha \quad = \quad d_{\theta'}\Big(\alpha\, e_\theta(x) + (1-\alpha)\, e_\theta(y)\Big)$$

for $\alpha \in [0, 1]$.



Is this a good generative model?

- **Limitations:** There is **no constraint** on the **regularity** of the latent space embedding.



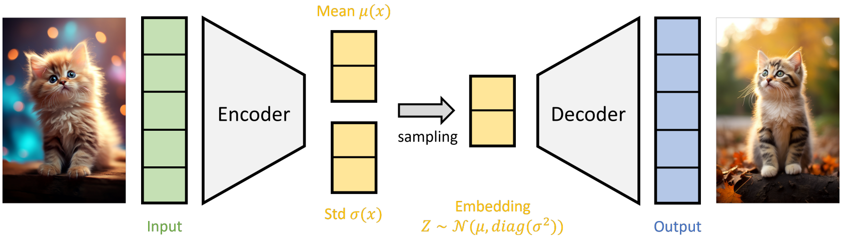
source: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

Variational Autoencoders (VAEs)

- **Objective:** Regularize by forcing the embedding to be **robust to noise**.
- **Idea:** The encoder returns the parameters $(\mu_x, \sigma_x) = e_\theta(x)$ of a Gaussian distribution. We sample $Z_x \sim \mathcal{N}(\mu_x, \sigma_x)$ and minimize

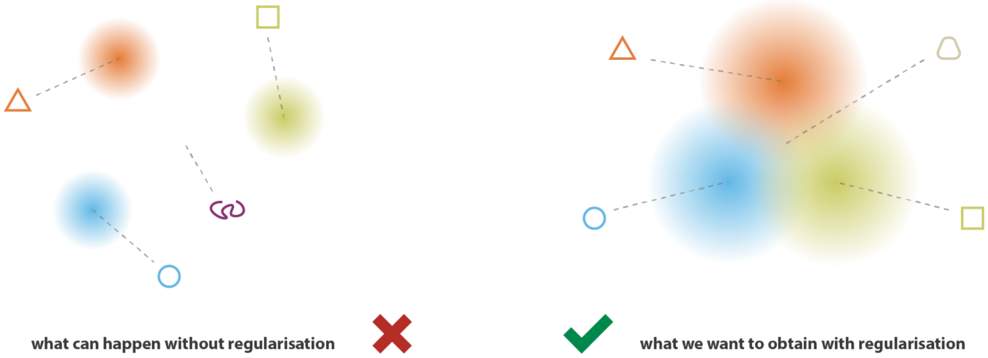
$$\min_{\theta,\theta'} \frac{1}{n} \sum_{i=1}^{n} \|x_i - d_{\theta'}(Z_{x_i})\|^2 + d_{\mathrm{KL}}\Big(\mathcal{N}(\mu_{x_i}, \sigma_{x_i}), \mathcal{N}(0, I)\Big)$$

where $d_{\mathrm{KL}}(p, q) = \mathbb{E}_{X\sim p}(\log(p(X)/q(X)))$ measures the "distance" between p and q.



Regularization with KL divergence

- **Benefits:** Each image is pushed to be mapped to a normal distribution.
- **Sampling:** We can sample new images with $d_{\theta'}(Z)$ where $Z \sim \mathcal{N}(0, I)$.

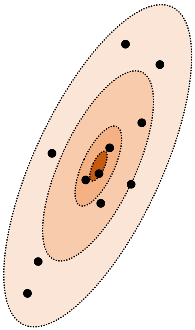


Performance measures

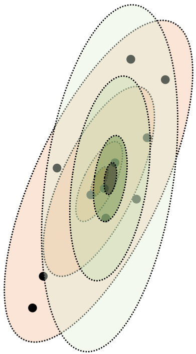
When is our model good enough?

Comparing data distribution and generated distribution

- **Question:** How should we measure distances between real and generated distributions?



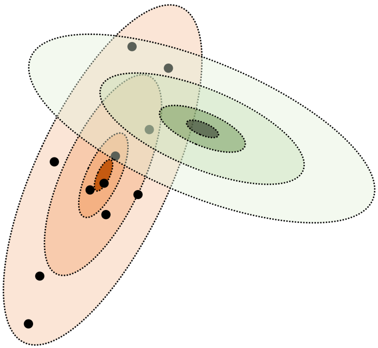
Training dataset and
underlying distribution



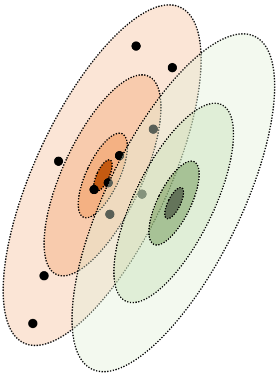
Good fit!

Comparing data distribution and generated distribution

- **Question:** How should we measure distances between real and generated distributions?



Bad fit?



Better fit!?

Likelihood of the data distribution

- We already saw that we can train a probabilistic model by **maximizing the likelihood** of the training data points (approach used by most of statistics!).

Likelihood of the data distribution

- We already saw that we can train a probabilistic model by **maximizing the likelihood** of the training data points (approach used by most of statistics!).
- Equivalent to minimizing the **negative log-likelihood**:

$$\min_{\theta} - \sum_{i=1}^{n} \log p_\theta(x_i)$$

where $(x_1, \ldots, x_n)$ are the training data points and p_θ is the density of the distribution.

Likelihood of the data distribution

- We already saw that we can train a probabilistic model by **maximizing the likelihood** of the training data points (approach used by most of statistics!).
- Equivalent to minimizing the **negative log-likelihood**:

$$\min_{\theta} - \sum_{i=1}^{n} \log p_\theta(x_i)$$

  where $(x_1, \ldots, x_n)$ are the training data points and p_θ is the density of the distribution.
- **Cons:** Requires to have access to the density p_θ. Can overfit training data.

Likelihood of the data distribution

- We already saw that we can train a probabilistic model by **maximizing the likelihood** of the training data points (approach used by most of statistics!).
- Equivalent to minimizing the **negative log-likelihood**:

$$\min_\theta - \sum_{i=1}^{n} \log p_\theta(x_i)$$

  where $(x_1, \ldots, x_n)$ are the training data points and p_θ is the density of the distribution.
- **Cons:** Requires to have access to the density p_θ. Can overfit training data.
- This is equivalent to minimizing the **Kullback-Leibler divergence** $d_{KL}(\hat{p}_n, p_\theta)$, where:

$$d_{KL}(p, q) = \mathbb{E}\left(\ln\left(\frac{p(X)}{q(X)}\right)\right)$$

  where $\hat{p}_n = \frac{1}{n}\sum_i \delta_{x_i}$ and $X \sim p$.

Other performance metrics

Wasserstein distance

- Measures how similar are the two measures via **evaluation functions**:

$$d_W(\mu, \nu) = \sup_{f \in \mathsf{Lip}_1} |\mathbb{E}(f(X)) - \mathbb{E}(f(Y))|$$

where $X \sim \mu$, $Y \sim \nu$ and Lip_1 is the space of 1-Lipschitz functions.

Other performance metrics

Wasserstein distance

- Measures how similar are the two measures via **evaluation functions**:

$$d_W(\mu, \nu) = \sup_{f \in \mathsf{Lip}_1} |\mathbb{E}(f(X)) - \mathbb{E}(f(Y))|$$

  where $X \sim \mu$, $Y \sim \nu$ and Lip_1 is the space of 1-Lipschitz functions.
- Measures are similar if there is **no way to distinguish** them with (Lipschitz) statistics.
- Another (equivalent) definition via **optimal transport**.

Other performance metrics

Wasserstein distance

- Measures how similar are the two measures via **evaluation functions**:

$$d_W(\mu, \nu) = \sup_{f \in \mathsf{Lip}_1} |\mathbb{E}(f(X)) - \mathbb{E}(f(Y))|$$

  where $X \sim \mu$, $Y \sim \nu$ and Lip_1 is the space of 1-Lipschitz functions.
- Measures are similar if there is **no way to distinguish** them with (Lipschitz) statistics.
- Another (equivalent) definition via **optimal transport**.

Human evaluation

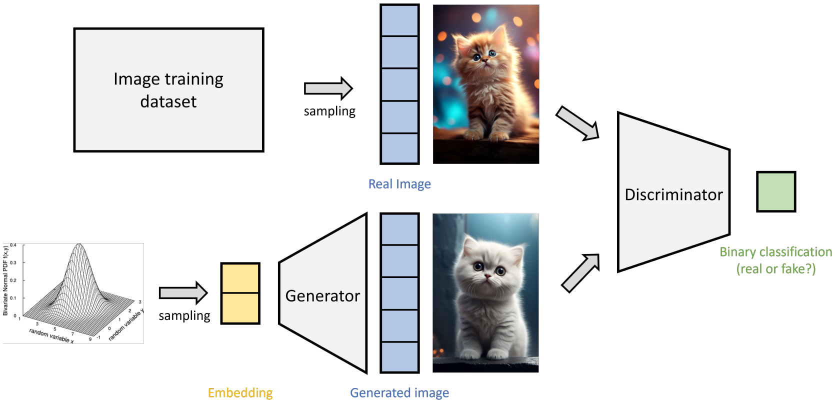
- Compare the outputs and decide which generative model you prefer...
- **Limitations:** subjective, and difficult to assess **diversity**.

Generative Adversarial Networks (GANs)

Asking another NN if your NN is good enough

Generative Adversarial Networks (Goodfellow et.al., 2014)

- **Idea:** Use another NN (discriminator) to **compare true and generated images**.
- Discriminator finds **mistakes** in the generation, and generator learns to **fool** the critic.



Traning GANs: a min-max optimization problem

- **Generator:** g_θ generates a fake sample $g_\theta(Z)$ with a Gaussian r.v. $Z \sim \mathcal{N}(0, I)$.

Traning GANs: a min-max optimization problem

- **Generator:** g_θ generates a fake sample $g_\theta(Z)$ with a Gaussian r.v. $Z \sim \mathcal{N}(0, I)$.
- **Discriminator:** $d_{\theta'}$ is a **classifier** and $d_{\theta'}(x)$ is the probability for x to be a real sample.

Traning GANs: a min-max optimization problem

- **Generator:** g_θ generates a fake sample $g_\theta(Z)$ with a Gaussian r.v. $Z \sim \mathcal{N}(0, I)$.
- **Discriminator:** $d_{\theta'}$ is a **classifier** and $d_{\theta'}(x)$ is the probability for x to be a real sample.
- **Learning:** g_θ and $d_{\theta'}$ are learnt **alternatively**, i.e. one is fixed when the other is learnt.
- **Loss:** For real images $(x_1, \ldots, x_n)$ and generated images $(g_\theta(Z_1), \ldots, g_\theta(Z_n))$, we want

$$\max_{\theta} \min_{\theta'} \mathcal{L}(\theta, \theta') = -\frac{1}{n} \sum_{i=1}^{n} \log \Big(d_{\theta'}(x_i) \Big) + \log \Big(1 - d_{\theta'}(g_\theta(z_i)) \Big)$$

Traning GANs: a min-max optimization problem

- **Generator:** g_θ generates a fake sample $g_\theta(Z)$ with a Gaussian r.v. $Z \sim \mathcal{N}(0, I)$.
- **Discriminator:** $d_{\theta'}$ is a **classifier** and $d_{\theta'}(x)$ is the probability for x to be a real sample.
- **Learning:** g_θ and $d_{\theta'}$ are learnt **alternatively**, i.e. one is fixed when the other is learnt.
- **Loss:** For real images $(x_1, \ldots, x_n)$ and generated images $(g_\theta(Z_1), \ldots, g_\theta(Z_n))$, we want

$$\max_{\theta} \min_{\theta'} \mathcal{L}(\theta, \theta') = -\frac{1}{n} \sum_{i=1}^{n} \log\Big(d_{\theta'}(x_i)\Big) + \log\Big(1 - d_{\theta'}(g_\theta(z_i))\Big)$$

- **Interpretation:** Discriminator minimizes its **BCE loss**, generator tries to **maximize** it.

Learning algorithm and technical details

- **Descriptor:** For a fixed generator g_θ, the optimal discriminator is $\theta'_\star = \mathrm{argmin}_{\theta'} \mathcal{L}(\theta, \theta')$.
- **Generator:** For a fixed $d_{\theta'}$, optimal gen. is $\theta_\star = \mathrm{argmax}_\theta -\frac{1}{n}\sum_{i=1}^n \log\Big(1 - d_{\theta'}(g_\theta(z_i))\Big)$.

Learning algorithm and technical details

- **Descriptor:** For a fixed generator g_θ, the optimal discriminator is $\theta'_\star = \mathrm{argmin}_{\theta'} \mathcal{L}(\theta, \theta')$.
- **Generator:** For a fixed $d_{\theta'}$, optimal gen. is $\theta_\star = \mathrm{argmax}_\theta -\frac{1}{n}\sum_{i=1}^n \log\Big(1 - d_{\theta'}(g_\theta(z_i))\Big)$.
- **Practice:** This is often replaced by $\theta_\star = \mathrm{argmax}_\theta \frac{1}{n}\sum_{i=1}^n \log\Big(d_{\theta'}(g_\theta(z_i))\Big)$.
- **Intuition:** When generator is **weak** compared to discriminator (i.e. $d_{\theta'}(g_\theta(z)) \ll 1$), the modified loss boosts the learning of the generator thanks to its large grad. at 0.

Learning algorithm and technical details

- **Descriptor:** For a fixed generator g_θ, the optimal discriminator is $\theta'_\star = \mathrm{argmin}_{\theta'} \mathcal{L}(\theta, \theta')$.
- **Generator:** For a fixed $d_{\theta'}$, optimal gen. is $\theta_\star = \mathrm{argmax}_\theta -\frac{1}{n}\sum_{i=1}^n \log\Big(1 - d_{\theta'}(g_\theta(z_i))\Big)$.
- **Practice:** This is often replaced by $\theta_\star = \mathrm{argmax}_\theta \frac{1}{n}\sum_{i=1}^n \log\Big(d_{\theta'}(g_\theta(z_i))\Big)$.
- **Intuition:** When generator is **weak** compared to discriminator (i.e. $d_{\theta'}(g_\theta(z)) \ll 1$), the modified loss boosts the learning of the generator thanks to its large grad. at 0.
- **Limitation:** training **extremely unstable**, potential **mode collapse**.

Learning algorithm and technical details

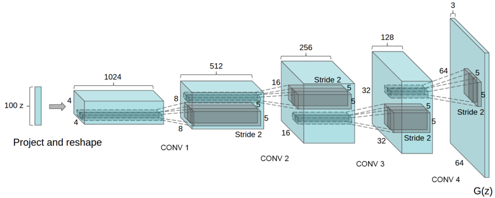
- **Descriptor:** For a fixed generator g_θ, the optimal discriminator is $\theta'_\star = \mathrm{argmin}_{\theta'} \mathcal{L}(\theta, \theta')$.
- **Generator:** For a fixed $d_{\theta'}$, optimal gen. is $\theta_\star = \mathrm{argmax}_\theta -\frac{1}{n}\sum_{i=1}^n \log\Big(1 - d_{\theta'}(g_\theta(z_i))\Big)$.
- **Practice:** This is often replaced by $\theta_\star = \mathrm{argmax}_\theta \frac{1}{n}\sum_{i=1}^n \log\Big(d_{\theta'}(g_\theta(z_i))\Big)$.
- **Intuition:** When generator is **weak** compared to discriminator (i.e. $d_{\theta'}(g_\theta(z)) \ll 1$), the modified loss boosts the learning of the generator thanks to its large grad. at 0.
- **Limitation:** training **extremely unstable**, potential **mode collapse**.
- **Extensions:** Wasserstein GANs (Arjovsky et.al., 2017) view the discriminator as the probe function in Wasserstein distance. More **principled** and **stable** in practice.

Deep Convolutional GAN (Radford et al., 2015)

- "Historical attempts to scale up GANs using CNNs to model images have been **unsuccessful**. [...] However, after **extensive model exploration** we identified a family of architectures that resulted in **stable training** across a range of datasets and allowed for training higher resolution and deeper generative models."

Deep Convolutional GAN (Radford et al., 2015)

- "Historical attempts to scale up GANs using CNNs to model images have been **unsuccessful**. [...] However, after **extensive model exploration** we identified a family of architectures that resulted in **stable training** across a range of datasets and allowed for training higher resolution and deeper generative models."
- **Heuristics:** 1) Replace pooling layers with **strided** (transposed) convolutions. 2) Use **batchnorm** in both g_θ and $d_{\theta'}$. 3) Remove linear layers. 4) use ReLU in g_θ except for the output using Tanh, and **LeakyReLU** in $d_{\theta'}$.



source: Radford et al. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. 2015.

Recap

- Generative models rely on **learning to sample** probability distributions.
- VAEs use an **Encoder-Decoder** architecture to learn a **low-dimensional latent representation** of the data distribution.
- GANs use two adversarial networks trained alternatively (**Generator** and **Discriminator**).
- To create images from low-dimensional vectors, we need to use **transposed convolutions**.
- Training is very **unstable**, and requires lots of tricks in practice.