Todays lecture

Until now:

- Dense NNs
- Supervised learning
- Unsupervised Learning
- Hyperparameter tuning
- Implementations

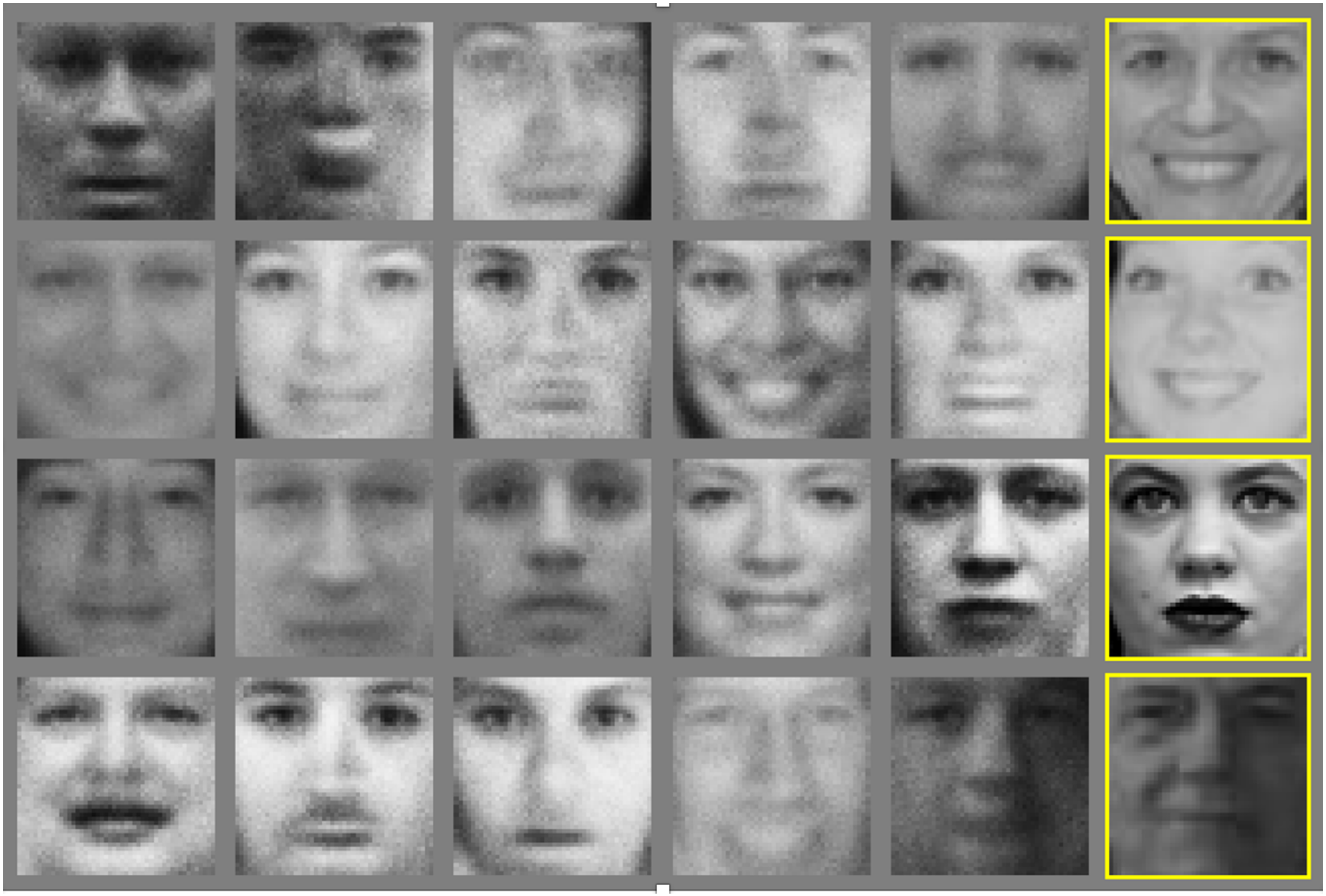
Today:

- Generative Models
  - GANs
  - Wasserstein GANs
  - Alternative approaches

Generative Adversarial Networks - History

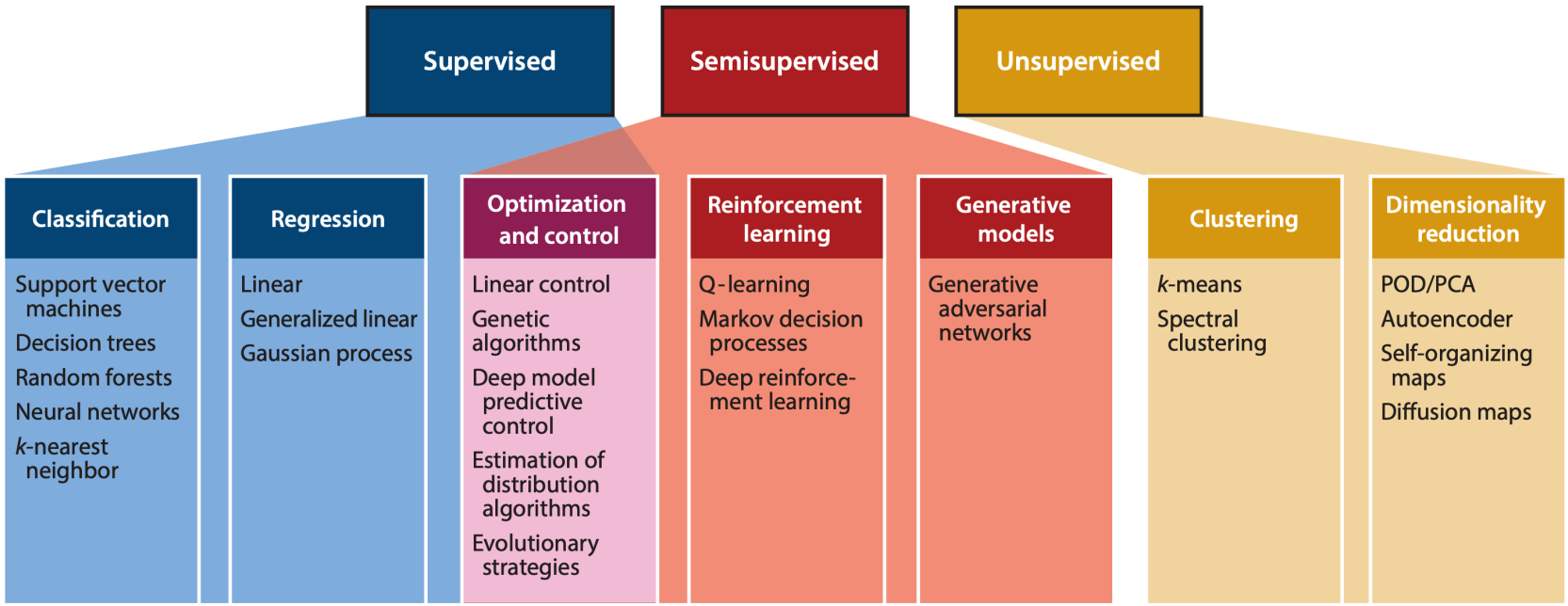
- Original paper: "Generative adversarial nets" by Ian Goodfellow et al. (2014)
  - 65882 citations!
- Much development since then!
- First model to actually generate plausible images
- Attention in public media - deep fakes

GAN - 2014



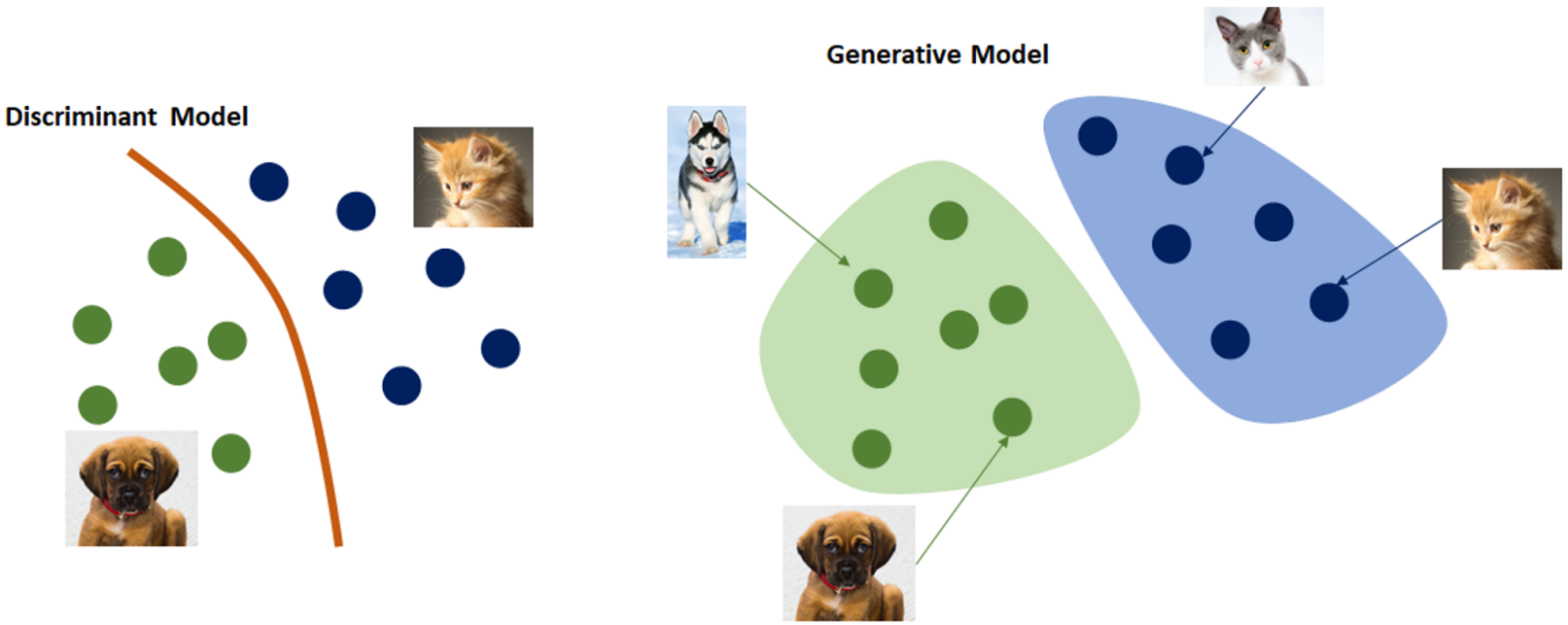
GAN - 2019

Learning Approaches



Discriminative vs Generative modeling

- Discriminative/regression model: Make predictions on unseen data based on conditional probability
- Generative model: Learn a distribution of a dataset with the goal of generating new samples



Introduction to Generative Modelling

- Real data comes a distribution $\mathbb{P}_{data}$
- We don't have access to $\mathbb{P}_{data}$
- But we have access to samples from $\mathbb{P}_{data}$
- A generative model, $\mathbb{P}_{model}$, learns to approximate $\mathbb{P}_{data}$ from samples
- $\mathbb{P}_{model} \approx \mathbb{P}_{data}$
- QUESTION: How do we learn a distribution?

Learn distribution directly?

- Assume a certain family of distributions
- Learn parameters of distribution
- E.g. via maximum likelihood

Learn distribution directly?

Learn distribution directly?

- Assume a certain family of distributions
- Learn parameters of distribution
- E.g. via maximum likelihood

- We gain full understanding of the distribution

Learn distribution directly?

- Assume a certain family of distributions
- Learn parameters of distribution
- E.g. via maximum likelihood

- We gain full understanding of the distribution
- Struggles with high-dimensional distributions (curse of dimensionality)
- (VERY!) limiting if distribution is complicated

Learn distribution indirectly?

- Learn to **sample** from the unknown distribution
- E.g. via monte carlo methods or pushforward maps

Learn distribution indirectly?

- Learn to **sample** from the unknown distribution
- E.g. via monte carlo methods or pushforward maps

- Less restrictive
- Typically better with respect to the curse of dimensionality

Learn distribution indirectly?

- Learn to **sample** from the unknown distribution
- E.g. via monte carlo methods or pushforward maps

- Less restrictive
- Typically better with respect to the curse of dimensionality

- Difficult to analyse generated distribution

Pushforward distribution

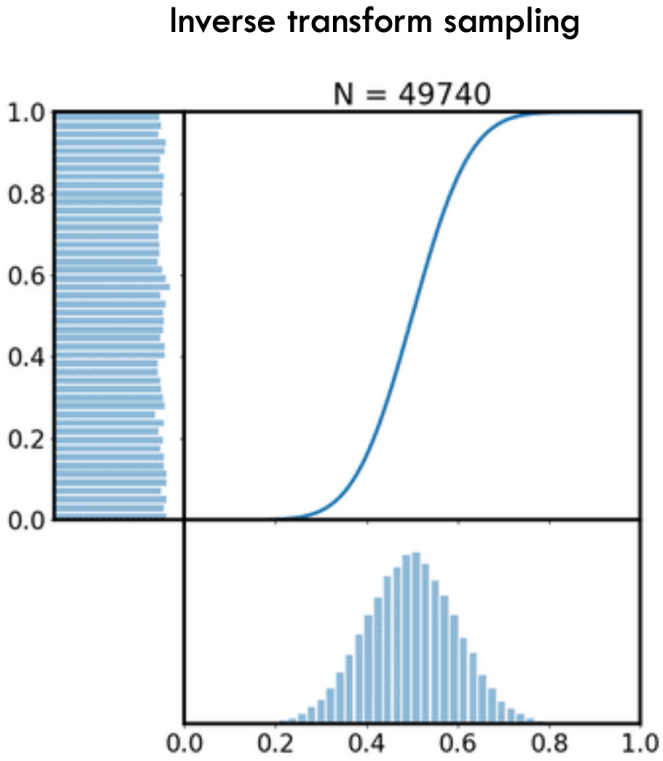
- Transform simple distributions to complicated ones
- Transformation is deterministic
- Example (Normal distribution): Inverse CDF is the **pushforward map**
  - CDF:

$$F(x) = \frac{1}{2} + \frac{1}{2} erf\left(\frac{x}{\sqrt{2}}\right)$$

  - Inverse CDF:

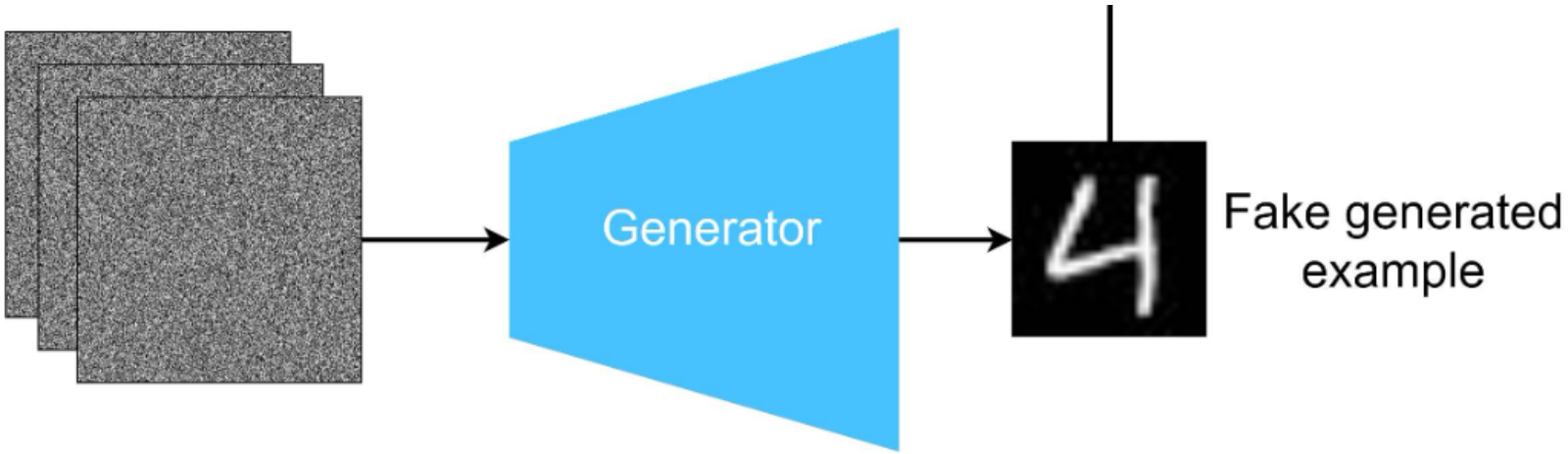
$$F^{-1}(y) = \sqrt{(2)} erfinv(2y - 1)$$

- What if we do not know the CDF or its inverse?



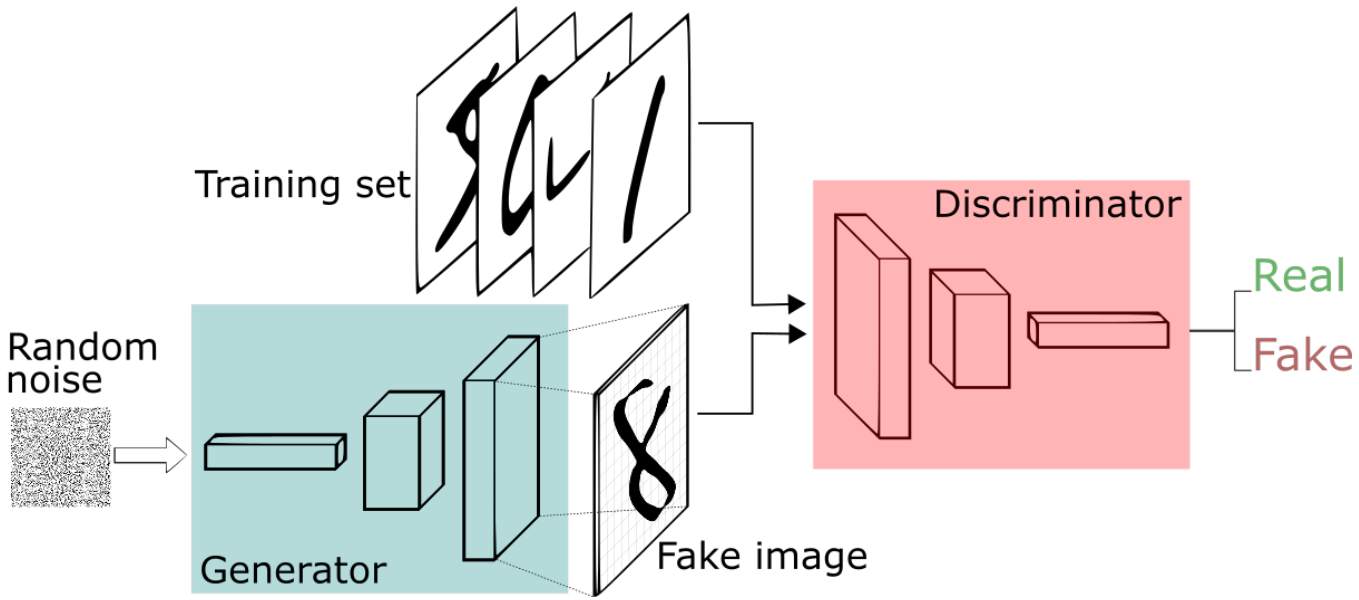
Generative Adversarial Networks

- GANs learn a deterministic pushforward map
- "Inverse CDF"
- Only needs samples to be trained



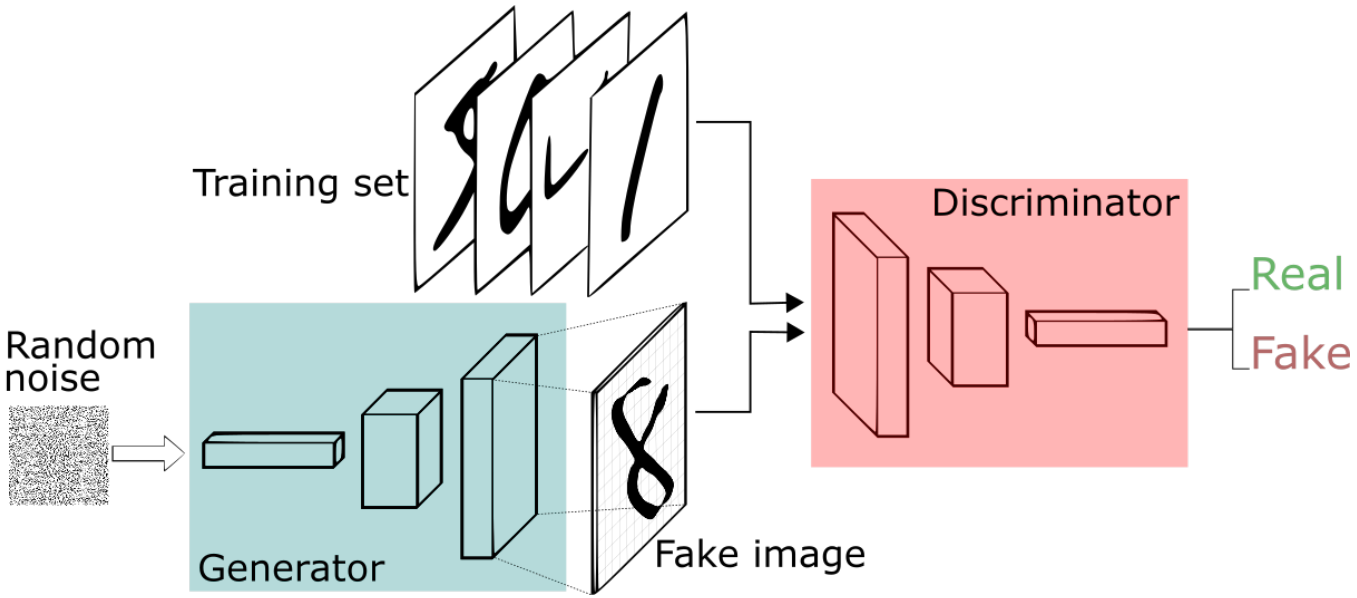
The GAN framework

- Game between two players
- **Generator**, G, generates samples
- **Discriminator**, D, discriminates between real and genrated samples
  - Uses traditional supervised learning techniques to determine whether samples are real or fake



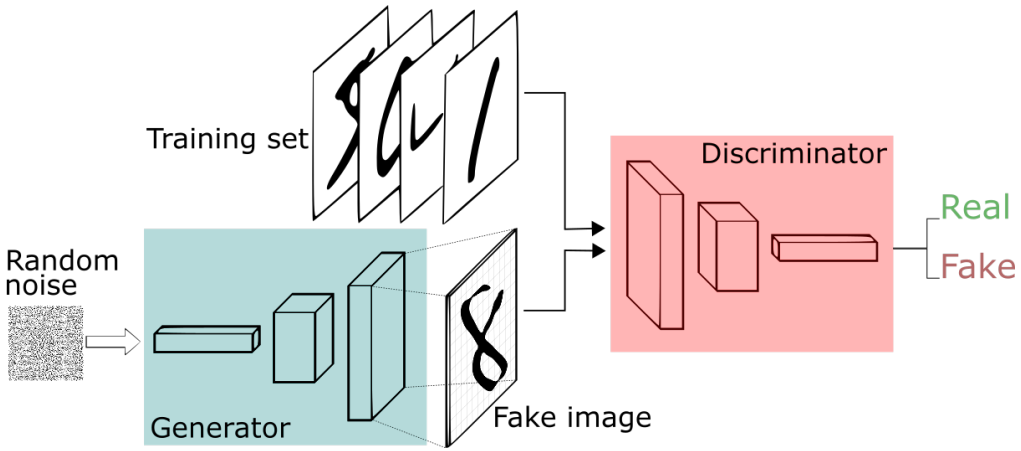
The GAN framework

Suppose we want to have a model that generates images of hand-written numbers. Generator G would start by producing blurry images that the discriminator D would classify as fake. The discriminator provides feedback to the generator during training, such that the generator would learn how to make realistic hand-written numbers that can fool the discriminator.



Latent Distribution

- **Latent** distribution $\mathbb{P}_z(z)$
- Also referred to as the **prior**
- $\mathbb{P}_z(z)$ is of **lower dimension** than $\mathbb{P}_{data}$
- $\mathbb{P}_z(z)$ is (almost) always chosen to be $N(0, 1)$
- Samples are random **noise**



Generator

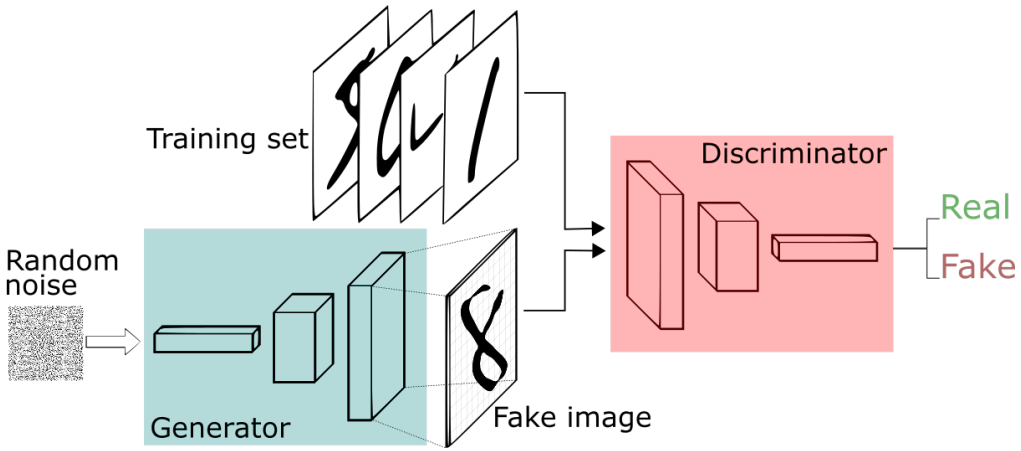
- Deep neural networks are used to represent G.
- G takes **latent samples**, $z \sim \mathbb{P}_z$, and generates a sample

$$x = G(z) \sim P_{model}$$

- $G(z) = x$ is of same dimension as the training data.
- G must be differentiable (these are the only requirements).
- The generator defines a pushforward distribution

$$\mathbb{P}_{model} = G_{\#}\mathbb{P}_z$$

- We want $\mathbb{P}_{model} = \mathbb{P}_{data}$



Generator as a parametric map

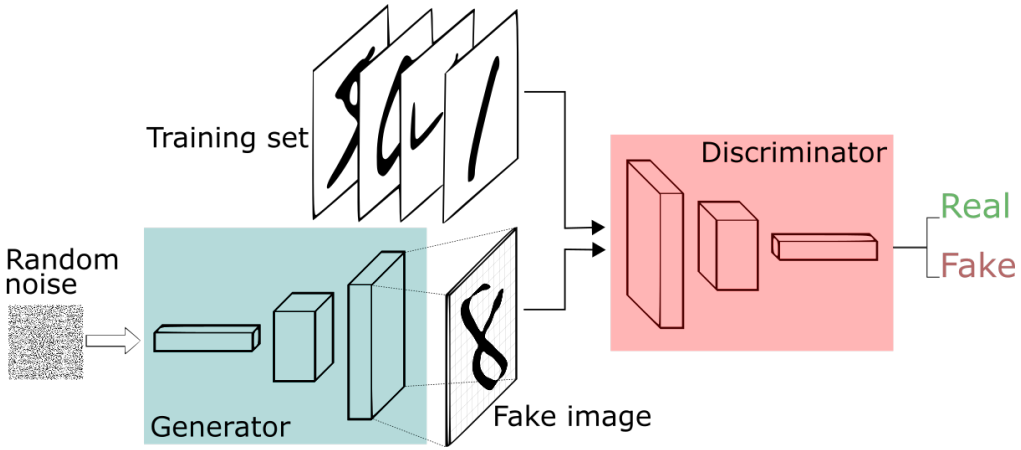
- The GAN is part of a class of methods for approximating the distribution of a target random variable $X \sim P_X$, starting from a prior $Z \sim P_Z$.
- Assume $X, Z \in \mathbb{R}^n$; the model that approximates the target is defined by a map $\varphi_\theta : \mathbb{R}^n \to \mathbb{R}^n$, $z \mapsto \varphi_\theta(z)$, with parameter set $\theta \in \mathbb{R}^p$, for some integer dimensions n, p.
- With the distribution of $\varphi_\theta(Z)$ given by P_θ, i.e. the distribution of the output samples is P_θ, the goal is to change the parameters θ such that $P_\theta \overset{d}{\approx} P_X$.
- In our case, the role of φ_θ is taken by the GAN generator.

Discriminator

- Deep neural networks are used to represent D
- The neural network takes real and fake samples as input and output a probability

$$D(x) \in [0, 1]$$

- $D(x) = 1$: Believes sample is **real**
- $D(x) = 0$: Believes sample is **fake**
- The discriminator is a **differentiable** function D
- Goal is to classify the real and fake samples accurately.
- Discriminator is **discarded** after training



Training

- We train a GAN the same way as any other neural network:
  1. Define loss function
  2. Evaluate loss function
  3. Use stochastic gradient descent to minimize loss function
- ... There are a few things that are different though

Training

- Generator has parameters $W^{(G)}$
- Discriminator has parameters $W^{(D)}$
- We define a loss function, $L(W^{(D)}, W^{(G)})$
- Generator aims to **maximize** $L(W^{(D)}, W^{(G)})$, while controlling only $W^{(G)}$.
- Discriminator D aims to **minimize** $L(W^{(D)}, W^{(G)})$ while controlling only $W^{(D)}$.
- The two networks have competing goals - this scenario can best be described as a **game**.

Loss Function

- Two player minimax game:

$$\min_{W^{(G)}} \max_{W^{(D)}} L(W^{(D)}, W^{(G)}) = \mathbb{E}_{x\sim\mathbb{P}_{data}} \log D(x) + \mathbb{E}_{z\sim\mathbb{P}_z} \log(1 - D(G(z)))$$

- This is the binary cross entropy for binary classification with a sigmoid output
- When training, we divide into two loss functions:
  - Generator loss

$$L^{(G)}(W^{(D)}, W^{(G)}) = \mathbb{E}_{z\sim\mathbb{P}_z} \log(1 - D(G(z)))$$

  - Discriminator loss

$$L^{(D)}(W^{(D)}, W^{(G)}) = -\mathbb{E}_{x\sim\mathbb{P}_{data}} \log D(x) - \mathbb{E}_{z\sim\mathbb{P}_z} \log(1 - D(G(z)))$$

Generator Loss Function

- Generator aims to fool the discriminator by maximizing the chance of the discriminator classifying generated data as *real*
- Taking only the part of the loss function that involves G
- Loss function:

$$L^{(G)}(W^{(D)}, W^{(G)}) = \mathbb{E}_{z \sim \mathbb{P}_z} \log(1 - D(G(z)))$$

Discriminator Loss Function

- Aims to classify real samples as *real* and generated samples as *fake*
- The cost used for the discriminator reads:

$$L^{(D)}(W^{(D)}, W^{(G)}) = -\mathbb{E}_{x\sim\mathbb{P}_{data}} \log D(x) - \mathbb{E}_{z\sim\mathbb{P}_z} \log(1 - D(G(z)))$$

- This is the cross-entropy cost that is minimized when training a binary classifier with a sigmoid output.
- The difference is that the classifier is trained on two mini-batches of data; one from a data-set with labels 1 for all examples, and one from the generator, where the label is 0 for all examples.

Training

1. Two mini-batches are sampled:
   - Real data samples, $x \sim \mathbb{P}_{data}$
   - Latent data samples, $z \sim \mathbb{P}_z$
2. Update discriminator:
   - Evaluate $L^{(D)}(W^{(D)}, W^{(G)})$
   - Update $W^{(D)}$ using stochastic gradient descent
   - Aim:
$$\min_{W^{(D)}} L^{(D)}(W^{(D)}, W^{(G)})$$
3. Update generator:
   - Evaluate $L^{(G)}(W^{(D)}, W^{(G)})$
   - Update $W^{(G)}$ using stochastic gradient descent
   - Aim:
$$\min_{W^{(G)}} L^{(G)}(W^{(D)}, W^{(G)})$$

It is common to update the discriminator several times before updating the generator

Training

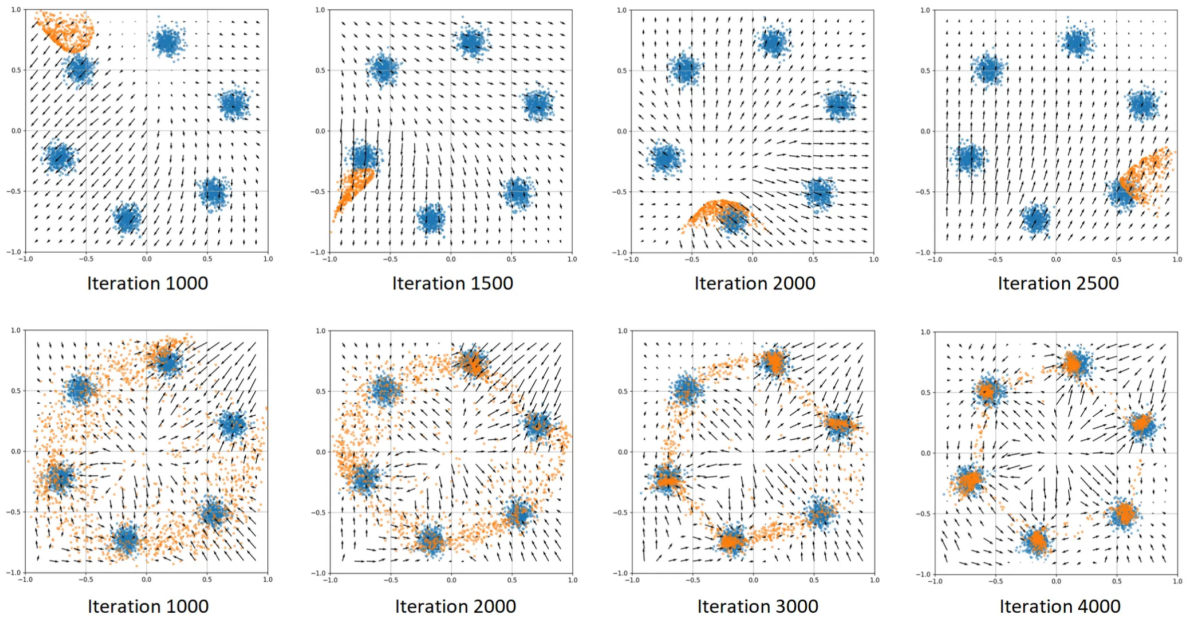
- The tuple $(W^{(D)}, W^{(G)})$ is a local minimum of $L^{(D)}$ w.r.t $W^{(D)}$ and a local minimum of $L^{(G)}$ w.r.t $W^{(G)}$.
- The solution to an optimization problem is a (local) point in parameter space where all neighboring points have greater or equal cost.
- The solution to a game is a **Nash equilibrium**.

Convergence stagnation in GANs

- Deep models are trained using optimization that usually makes reliable downhill progress.
- GANs require finding the equilibrium to a game with two players.
- One player moving downhill might move the other player uphill.
- Sometimes players reach an equilibrium, but in other scenarios they repeatedly undo each others' progress.
- Currently, there is neither a theoretical argument that GAN games should converge when the updates are made to parameters of deep neural networks, nor a theoretical argument that the games should not converge.

Mode Collapse

- Probably the most common form of harmful non-convergence encountered in the GAN game is mode collapse.
- Mode collapse is a phenomenon that arises when training results in failure. It may manifest when a generator maps several input z values to the same output region, or when a generator misses certain regions of target distribution $\mathbb{P}_{data}$.



Mode Collapse

- Definition: A target distribution $\mathbb{P}_{data}$ and a model distribution $\mathbb{P}_{model}$ produced by a generator G exhibit (ε, δ)-mode collapse for some $0 \leq \varepsilon < \delta \leq 1$, if there exists a set $S \subseteq X$ such that $\mathbb{P}_{data}(S) \geq \delta$ and $P_{model}(S) \leq \varepsilon$.
- Intuitively, larger δ and smaller ε indicate more severe mode collapse. The mode collapse problem is among the most important issues with GANs that researchers addressed.

Vanishing gradient

- When the discriminator is perfect, $D(x) = 1, \forall x \in \mathbb{P}_{data}$ and $D(x) = 0, \forall x \in \mathbb{P}_{model}$. Loss function $L \to 0$ and we end up with no gradient to update the loss during learning iterations.
- GAN Dilemma: If D behaves badly, G does not have accurate feedback and the loss function cannot represent reality. If D converges well, the gradient of the loss function drops down to almost zero and the learning becomes very slow.

Theoretical analysis of GAN

- We cannot use the generator and discriminator loss to asses performance
- We need another measure
- In supervised learning:
  - Regression: RMSE, MAE, ...
  - Classification: Accuracy, F-score, ...
- Generative learning:
  - KL divergence
  - JS divergence
  - Wasserstein distance (comes later!)

Definition (Kullback-Leibler Divergence)

- Def. Let $\mathbb{P}$ and $\mathbb{Q}$ be distributions of two absolutely continuous random variables with probability densities p and q. Their Kullback-Leibler (KL) divergence is defined to be:

$$D_{KL}(\mathbb{P}||\mathbb{Q}) = \int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{q(x)}\right) \mathrm{d}x.$$

- Regarding GANs, $D_{KL}(\mathbb{P}_{data}||\mathbb{P}_{model})$ has a unique minimum at $p_{model} = p_{data}$, and it doesn't require knowledge of the unknown p_{data} to estimate it (only samples).

Kullback-Leibler Divergence

$$D_{KL}(\mathbb{P}||\mathbb{Q}) = \int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{q(x)}\right) \mathrm{d}x.$$

- KL divergence is not symmetrical between $\mathbb{P}_{data}$ and $\mathbb{P}_{model}$:
- If $p_{data}(x) > 0$ but $p_{model}(x) \to 0 \Rightarrow KL \to \infty$
  - Loss function punishes generated samples hard
- If $p_{data} \to 0$ and $p_{model}(x) > 0 \Rightarrow KL \to 0$
  - Loss function will pay an extremely low cost for generating fake looking samples.

Jensen-Shannon Divergence

- Definition (Jensen-Shannon Divergence). Let $\mathbb{P}$ and $\mathbb{Q}$ be distributions of two different absolutely continuous random variables with probability densities p and q. Then, their Jensen-Shannon (JS) divergence is defined to be:

$$D_{JS}(p||q) = \frac{1}{2}D_{KL}\left(p||\frac{p+q}{2}\right) + \frac{1}{2}D_{KL}\left(q||\frac{p+q}{2}\right)$$

- The Jensen-Shannon divergence is symmetric and is connected to GAN training.

JS and KL Divergence

- JS is defined for absolutely continuous distribution measures P and Q through the Kullback-Leibler (KL) divergence:

$$\begin{aligned} JS(P\|Q) &= \frac{1}{2}\left(KL(P\|M) + KL(Q\|M)\right), \\ KL(P\|Q) &= \int_{\mathcal{X}} \log\left(\frac{p(x)}{q(x)}\right) q(x)dx, \end{aligned}$$

  where p and q are densities associated with P and Q, $M = \frac{P+Q}{2}$ and $\mathcal{X} \subseteq \mathbb{R}^n$ is the support.
- It can be shown that $JS(P\|Q) \geq 0$, with equality iff $P = Q$.
- The goal is to choose θ such that it minimises $JS(P_\theta\|P_X)$.

Example

- When the generator is trained to its optimal value, p_{model} gets close to p_{data}. When $p_{model} = p_{data}$, $D^*(x)$ becomes $1/2$.
- When both G and D are at their optimal values, we have $p_{model} = p_{data}$, $D(x) = 1/2$ and the loss function $L(G^*, D^*) =$

$$
\begin{aligned}
&= \int_x (p_{data}(x) \log(D^*(x)) + p_{model}(x) \log(1 - D^*(x)))\mathrm{d}x \\
&= \log \frac{1}{2} \int_x p_{data}(x)\mathrm{d}x + \log \frac{1}{2} \int_x p_{model}(x)\mathrm{d}x \\
&= -2 \log 2.
\end{aligned}
$$

Example

- If we calculate the JS divergence for $\mathbb{P}_{model}$ and $\mathbb{P}_{data}$, we have that $D_{JS}(\mathbb{P}_{data}||\mathbb{P}_{model}) =$

$$
\begin{aligned}
&= \frac{1}{2} D_{KL}\left(p_{data}||\frac{p_{data}+p_{model}}{2}\right) \\
&+ \frac{1}{2} D_{KL}\left(p_{model}||\frac{p_{data}+p_{model}}{2}\right) \\
&= \left(\log 2 + \int_x p_{data}(x) \log \frac{p_{data}(x)}{p_{data}(x)+p_{model}(x)} \mathrm{d}x\right) + \\
&\quad \left(\log 2 + \int_x p_{model}(x) \log \frac{p_{model}(x)}{p_{data}(x)+p_{model}(x)} \mathrm{d}x\right) \\
&= \frac{1}{2}(\log 4 + L(G, D^*)).
\end{aligned}
$$

- Thus, $L(G, D^*) = 2D_{JS}(\mathbb{P}_{data}||\mathbb{P}_{model}) - 2\log 2$.

Conditional GANs

- Generating conditional distributions
- Generator and discriminator receives a vector with conditional information as an additional input, which allows the GAN to learn how the output should vary based on a condition label, e.g. generating images of apples or oranges based on the given input.
- If we let $y \in \mathbb{R}^{n_c}$ be a vector with n_c conditional inputs, the joint objective function becomes:

$$\min_{W^{(G)}} \max_{W^{(D)}} L(W^{(D)}, W^{(G)}) =$$

$$\mathrm{E}_{x,y\sim\mathbb{P}_{data}} \log D(x,y) + \mathbb{E}_{z\sim\mathbb{P}_z, y\sim\mathbb{P}_y} \log(1 - D(G(z,y),y))$$

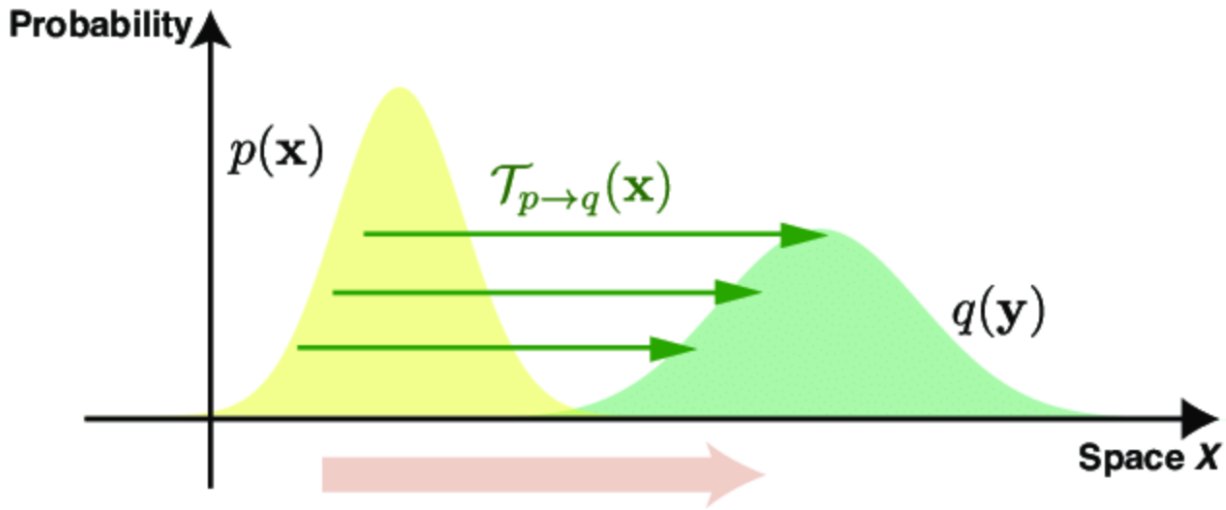
Let's take a break!

Wasserstein GAN (WGAN)

- A variation of GANs
- Based on the Wasserstein metric
- "Solves" mode collapse

Wasserstein Metric

- Distance between two probability distribution
- Sometimes referred to as the "earth mover" distance
- Deeply rooted in optimal transport theory



Wasserstein Metric

- Wasserstein-p metric

$$W^p(\mu,\nu) = \left(\inf_{\gamma\in\Gamma(\mu,\nu)} \int_M d(x,y)^p d\gamma(x,y)\right)^{1/p}$$

  Γ is the collection of all measures on $M \times M$ with marginals μ and ν.

- Equivalently defined by

$$W^p(\mu,\nu) = (\inf \mathbb{E}[d(X,Y)^p])^{1/p}$$

  where X and Y have μ and ν as marginals

Wasserstein Metric

- Wasserstein-2 metric

$$W^2(\mu,\nu) = \left(\inf_{\gamma \in \Gamma(\mu,\nu)} \int_M d(x,y)^2 d\gamma(x,y) \right)^{1/2}$$

- Wasserstein-1 metric

$$W^1(\mu,\nu) = \inf_{\gamma \in \Gamma(\mu,\nu)} \int_M d(x,y) d\gamma(x,y)$$

- WGANs use the Wasserstein-1 metric

Wasserstein Metric - Absolutely continuous measures

- Absolutely continuous measures with respect to the Lebesque measure, λ, have well-defined Radon-Nikodym derivatives
- $\longrightarrow$ a probability density function (PDF), f, exists
- The PDF satisfies

$$\mu(A) = \int_M f(x) d\lambda(x)$$

- On Euclidean spaces, we can use the Riemann integral

$$\mu(A) = \int_M f(x) dx$$

- Wasserstein-1 with Riemann integral

$$W^1(\mu, \nu) = \inf_{h \in H(\mu,\nu)} \int_M d(x,y)^2 h(x,y) dx dy$$

Wasserstein-1 Metric

- Wasserstein-1 metric

$$W^1(\mu,\nu) = \inf_{\gamma\in\Gamma(\mu,\nu)} \int_M d(x,x_0)d\gamma(x,y)$$

- Kantorovich-Rubenstein duality

$$W^1(\mu,\nu) = \sup_{Lip(f)\leq 1} \int_M f(x)d(\mu-\nu)$$

- In Euclidean space

$$W^1(\mu,\nu) = \sup_{Lip(f)\leq 1} \int_M f(x)(g(x)-h(y))dxdy$$

where g is the PDF corresponding to μ and h is the PDF corresponding to ν

Wasserstein-1 Metric - Relation to weighted L^1 space

- The Wasserstein-1 metric can be written in terms of norms in the weighted $L^1_r(M)$ space:

$$
\begin{aligned}
W^1(\mu,\nu) &= \sup_{Lip(f)\leq 1} \int_M f(x)(g(x)-h(y))dxdy \\
&= \sup_{Lip(f)\leq 1} \left(\int_M f(x)g(x)dx - \int_M f(x)h(x)dx \right) \\
&= \sup_{Lip(f)\leq 1} \left(||f||_{L^1_g} - ||f||_{L^1_h} \right) \\
&= \sup_{Lip(f)\leq 1} \left(\mathbb{E}_{x\sim\mu}[f(x)] - \mathbb{E}_{x\sim\nu}[f(x)] \right)
\end{aligned}
$$

Wasserstein GAN

- The WGAN aims to minimize the the Wasserstein-1 metric

$$W^1(\mathbb{P}_{data}, \mathbb{P}_{model}) = \sup_{Lip(f)\leq 1} (\mathbb{E}_{x\sim\mathbb{P}_{data}}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_{model}}[f(x)])$$

- Goal: Find $\mathbb{P}_{model}$ that minimizes the distance
- Challenge: How do we compute $W^1(\mathbb{P}_{data}, \mathbb{P}_{model})$?
  - Supremum over all functions f is not feasible!
  - We cannot compute the integrals
- Solution: Replace bottlenecks with NNs!

Wasserstein GAN - Critic

- *Critic*, $f(x) \in \mathbb{R}$

$$W^1(\mathbb{P}_{data}, \mathbb{P}_{model}) = \sup_{Lip(f) \leq 1} \left(\mathbb{E}_{x \sim \mathbb{P}_{data}} [\underbrace{f(x)}_{critic}] - \mathbb{E}_{x \sim \mathbb{P}_{model}} [\underbrace{f(x)}_{critic}] \right)$$

- The critic approximates all Lipschitz functions
- The critic is a neural network

Wasserstein GAN - Generator

- Generator is the same as Vanilla GAN

$$W^1(\mathbb{P}_{data}, \mathbb{P}_{model}) = \sup_{Lip(f)\leq 1} \left(\mathbb{E}_{x\sim\mathbb{P}_{data}}[f(x)] - \mathbb{E}_{z\sim\mathbb{P}_z}[f(\underbrace{G(z)}_{generator})] \right)$$

Wasserstein GAN - Training

- Critic is parametrized by it's weights, ω, - f_ω
- Critic wants to maximize w.r.t. f, in order to approximate W^1:

$$\max_{\omega} \mathbb{E}_{x \sim \mathbb{P}_{data}}[f_\omega(x)] - \mathbb{E}_{z \sim \mathbb{P}_z}[f_\omega(G(z))] \approx W^1(\mathbb{P}_{data}, \mathbb{P}_{model})$$

- Generator is parametrized by it's weights, ω - G_θ
- Generator wants to minimize $W^1(\mathbb{P}_{data}, \mathbb{P}_{model})$ in order to approximate $\mathbb{P}_{data}$:

$$\min_{\theta} \max_{\omega} \mathbb{E}_{x \sim \mathbb{P}_{data}}[f_\omega(x)] - \mathbb{E}_{z \sim \mathbb{P}_z}[f_\omega(G_\theta(z))] \approx \min_{\mathbb{P}_{model}} W^1(\mathbb{P}_{data}, \mathbb{P}_{model})$$

Wasserstein GAN - Training

- We now have a min-max game.. again!

$$L(G, f) = \min_{\theta} \max_{\omega} \mathbb{E}_{x \sim \mathbb{P}_{data}}[f_\omega(x)] - \mathbb{E}_{z \sim \mathbb{P}_z}[f_\omega(G_\theta(z))]$$

- Decompose into a generator and a critic part:

$$\begin{aligned} \min_{\theta} L_G &= -\mathbb{E}_{z \sim \mathbb{P}_z}[f_\omega(G_\theta(z))] \\ \min_{\omega} L_f &= -\mathbb{E}_{x \sim \mathbb{P}_{data}}[f_\omega(x)] + \mathbb{E}_{z \sim \mathbb{P}_z}[f_\omega(G_\theta(z))] \end{aligned}$$

NOTE: max is changed to min due to sign change!

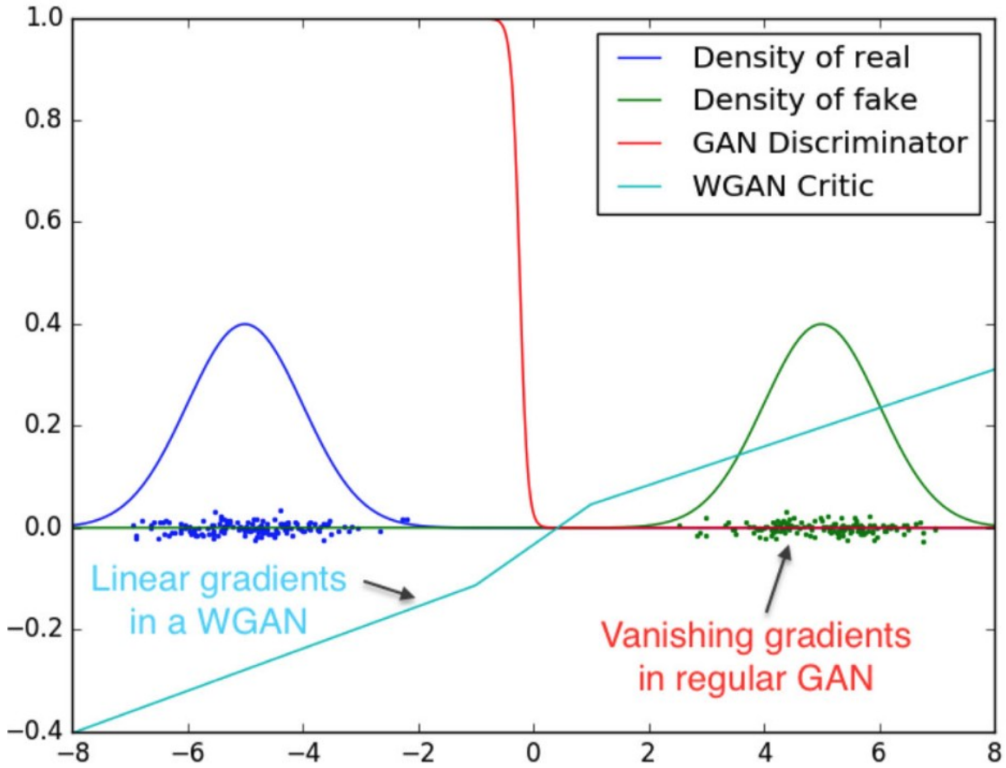
Wasserstein GAN - Lipschitz continuity?

- The critic must be Lipschitz continuous
- Three options:
  - Weight clipping
  - Spectral normalization
  - Regularization / gradient penalty

$$L_f + \lambda \mathbb{E}_{\hat{x} \sim \mathbb{P}}[||\nabla_{\hat{x}} f(\hat{x})||_2 - 1]$$

WGAN - Advantages

- Much smaller probability of mode collapse
- The Wasserstein-1 metric is "better" at scoring when distributions are far from each other



WGAN - Disadvantages

- The Wasserstein-1 metric is weaker than KL divergence
- Convergence:

$$\text{Total variation} \Leftrightarrow \text{KL divergence} \Rightarrow \text{Wasserstein-1}$$

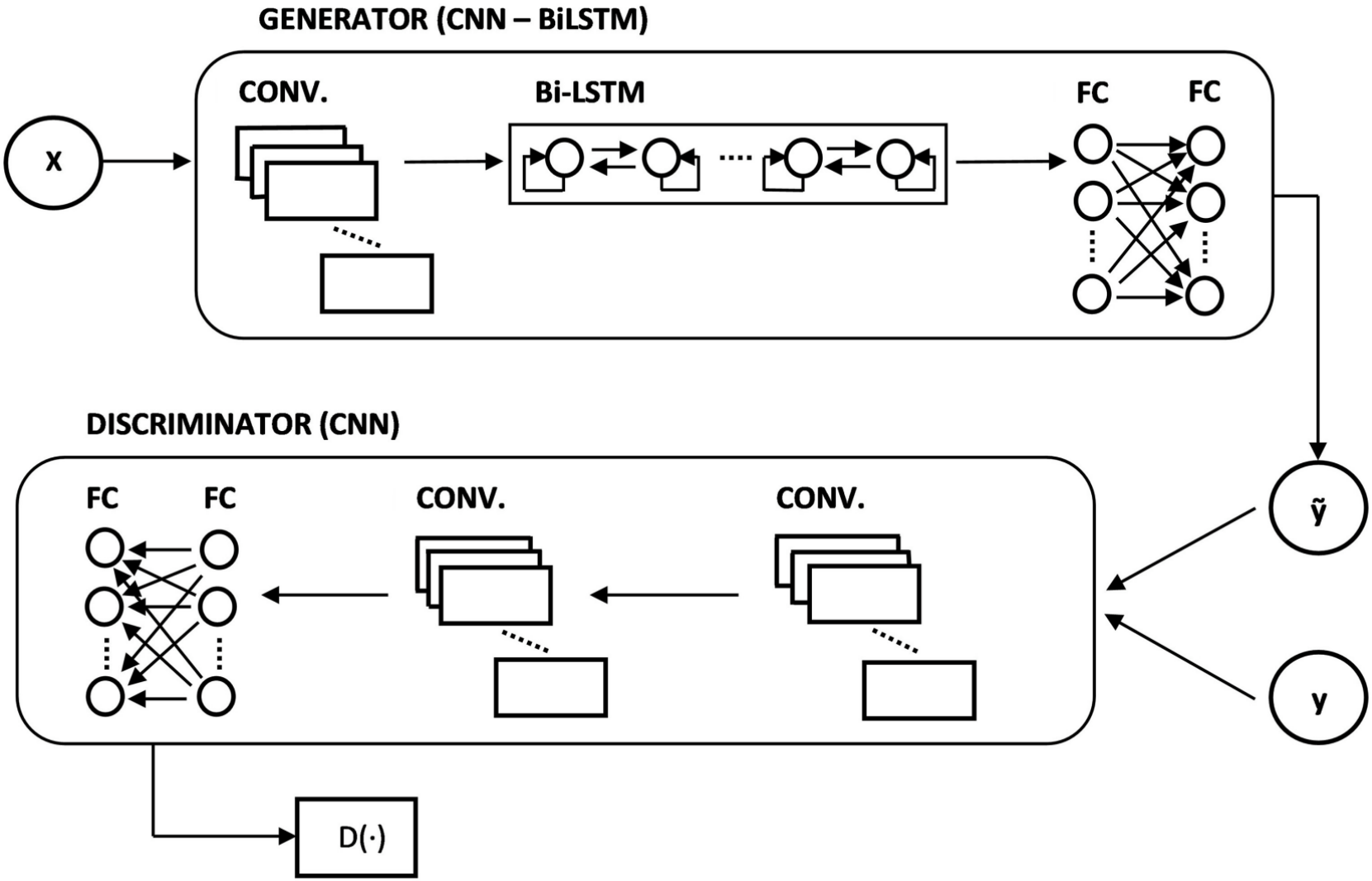
Stock Price Forecasting by a Deep Convolutional Generative Adversarial Network

- GAN for time series forecasting
- Consider time series, $(y_1, y_2, \ldots, y_t)$
- Input to generator: t time steps of i variables:

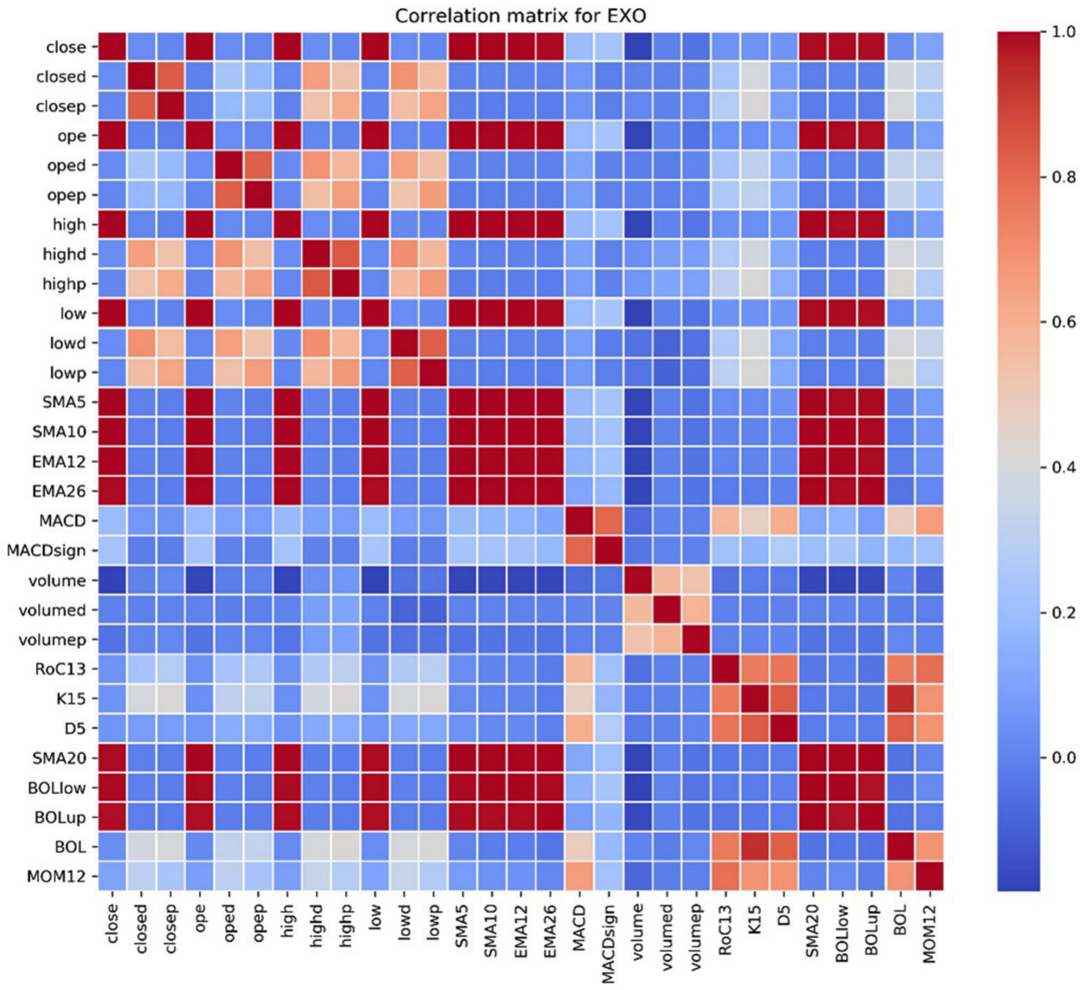
$$X = \begin{bmatrix} x_{1,1} & x_{2,1} \ldots & x_{i,1} \\ x_{1,2} & x_{2,2} \ldots & x_{i,2} \\ \vdots & \vdots & \ddots \\ x_{1,t} & x_{2,t} \ldots & x_{i,t} \end{bmatrix}$$

- Generator output: $G(X) = \hat{y}_{t+1}$
- Discriminator input:
  - From generator: $(y_1, y_2, \ldots, y_t, \hat{y}_{t+1})$
  - Real data: $(y_1, y_2, \ldots, y_t, \hat{y}_{t+1})$

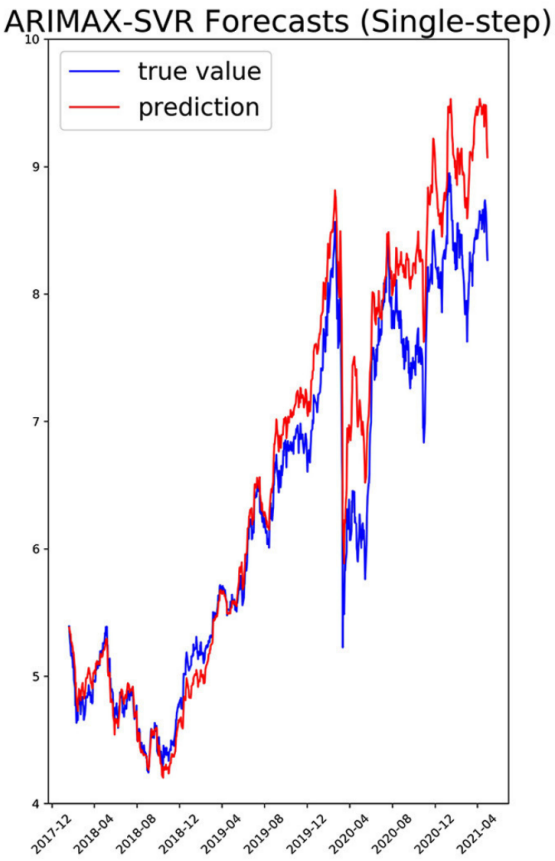
Stock Price Forecasting by a Deep Convolutional Generative Adversarial Network

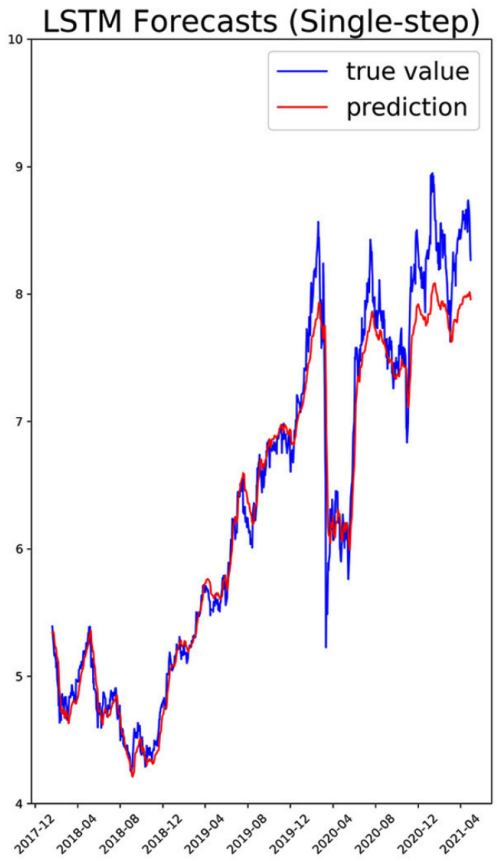


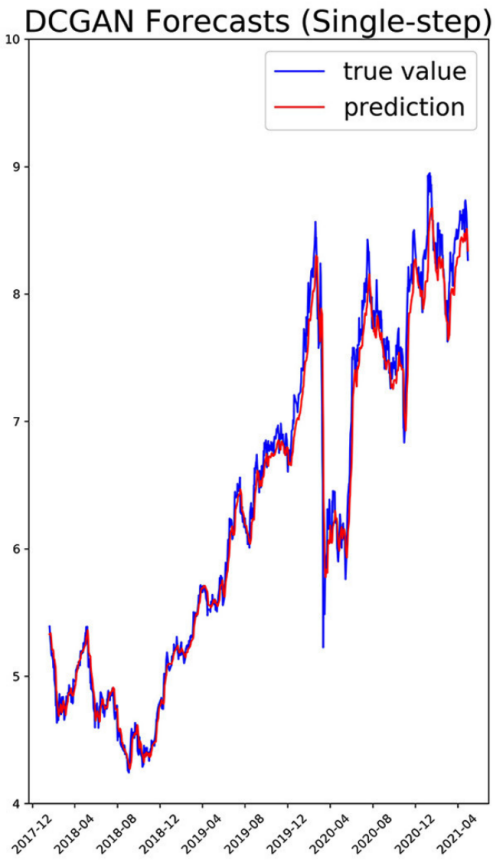
Stock Price Forecasting by a Deep Convolutional Generative Adversarial Network



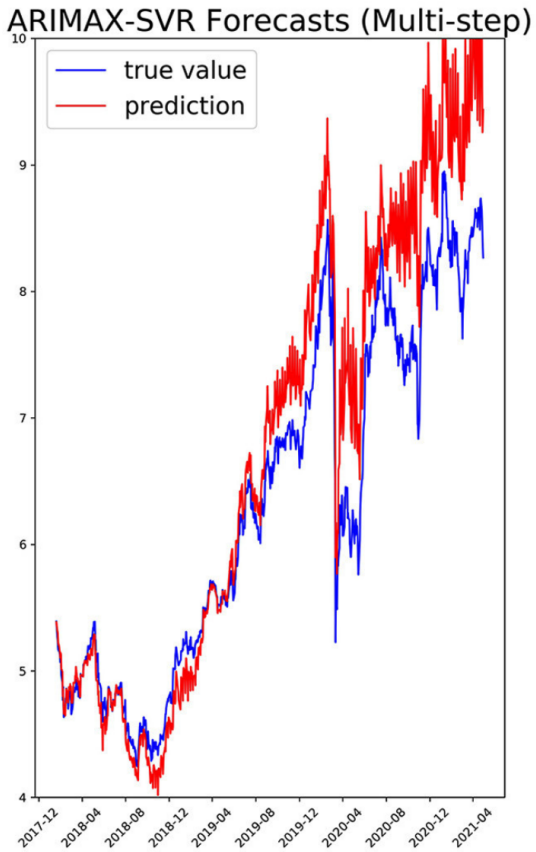
Stock Price Forecasting by a Deep Convolutional Generative Adversarial Network

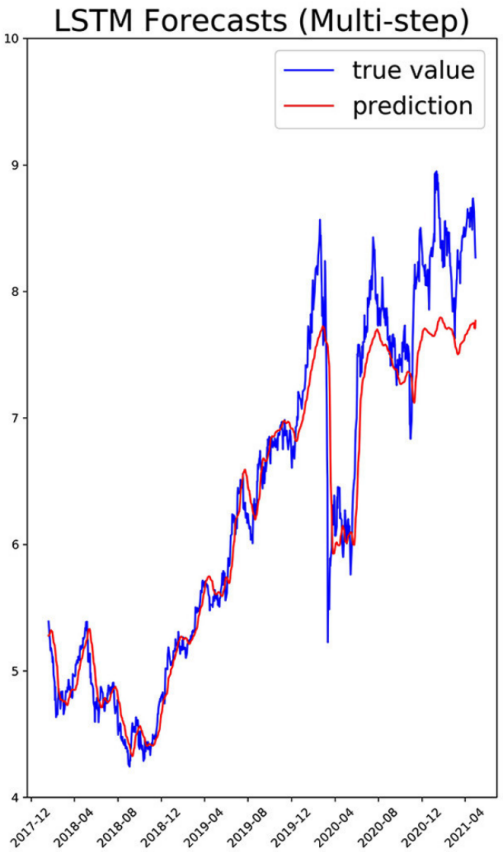


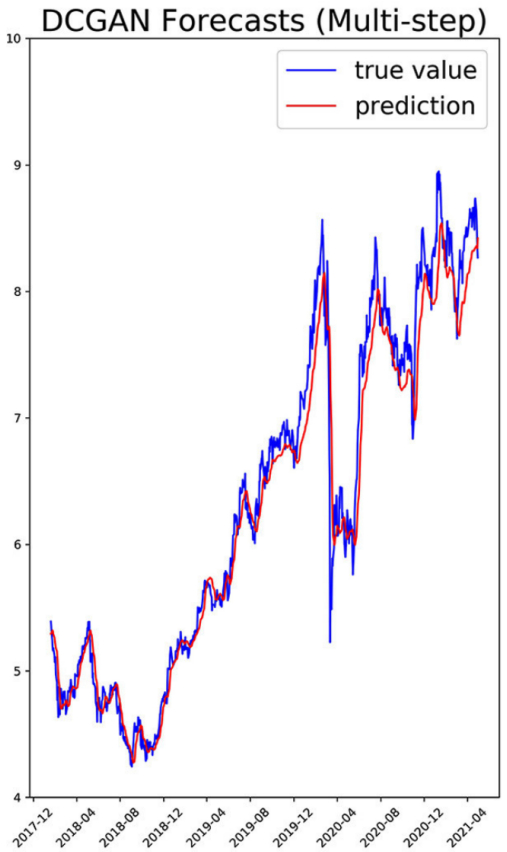




Stock Price Forecasting by a Deep Convolutional Generative Adversarial Network

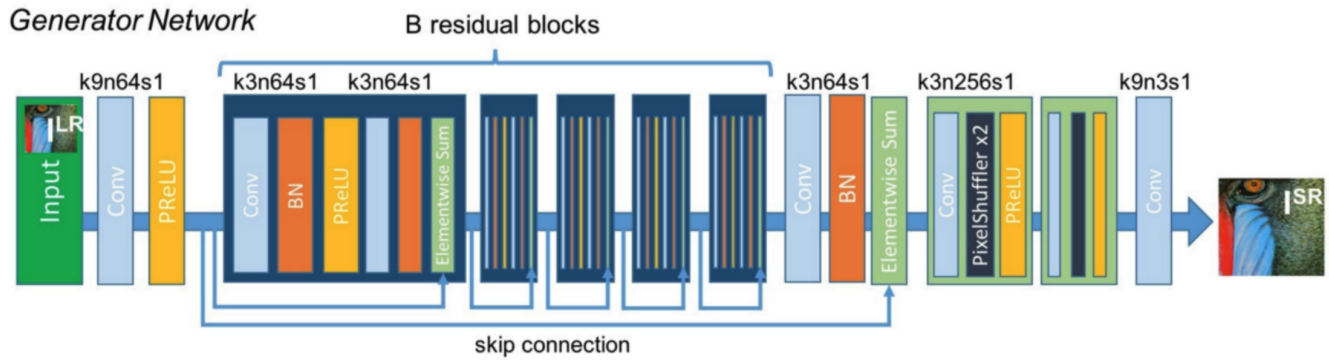


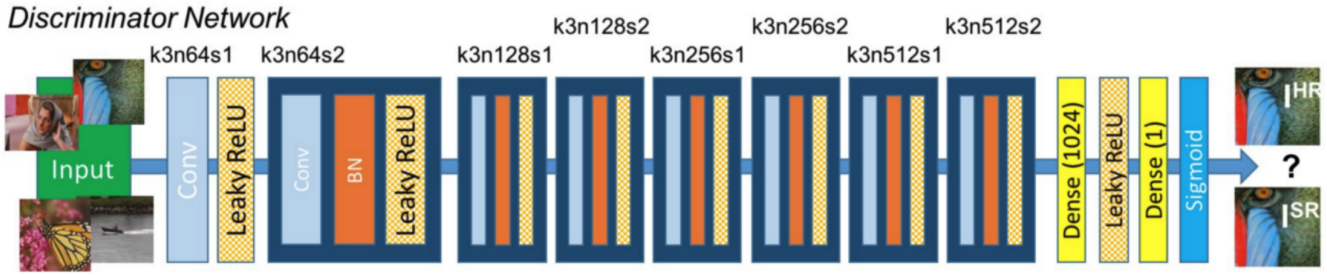


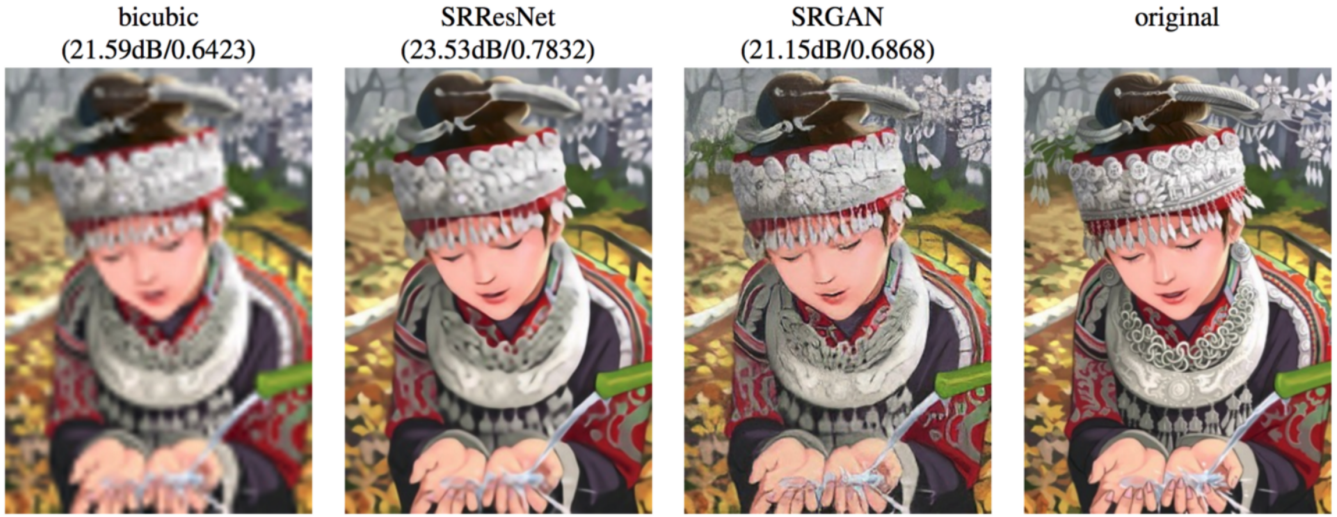


GAN Variations - Style GAN

GAN Variations - SRGAN







GAN Variations - CycleGAN

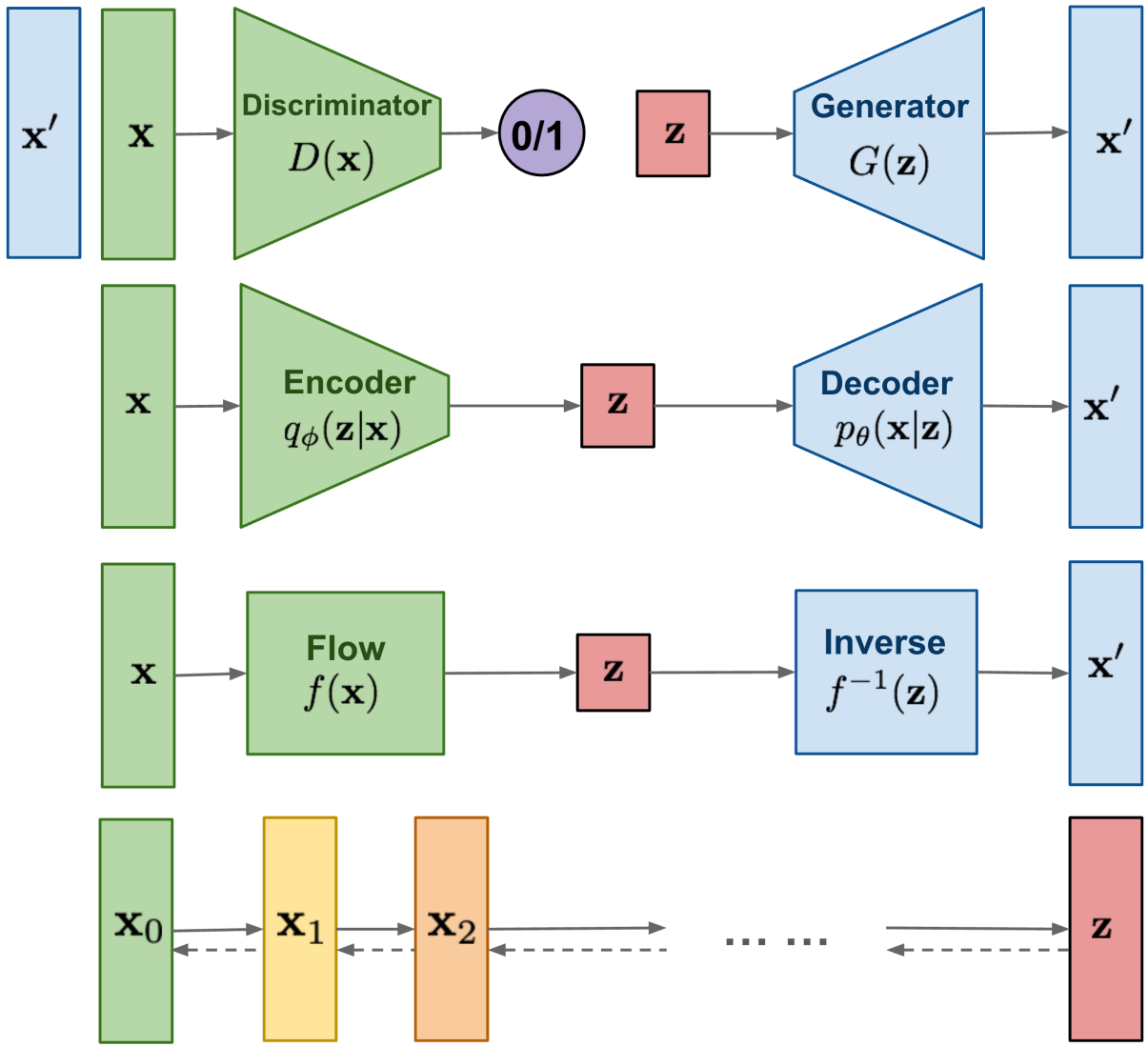
Alternative Methods

GAN: Adversarial training

VAE: maximize variational lower bound

Flow-based models: Invertible transform of distributions

Diffusion models: Gradually add Gaussian noise and then reverse



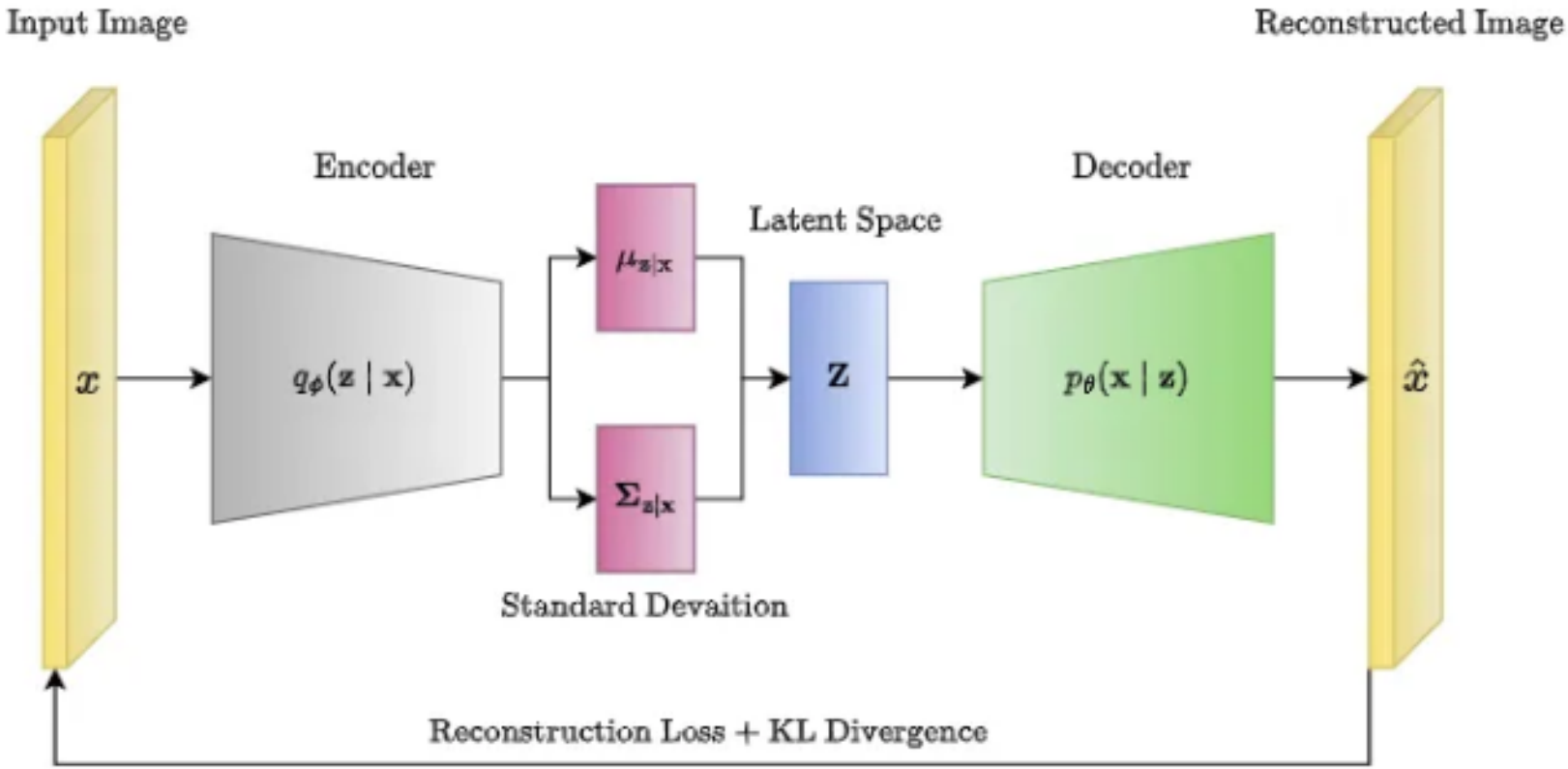
Alternative Methods - Variational Autoencoders

- Extension to Autoencoder framework
- Autoencoder: Reduce data into a low-dim vector
- Variation autoencoder: Reduce into a low-dim normal distribution
- Encoder reduces the data into a vector of means and variances

$$Enc(X) = (\mu(X), \sigma(X))$$

- Decoder takes samples from $\mathcal{N}(\mu(X), \sigma(X))$ and reconstructs X
- Repeatedly sampling from $\mathcal{N}(\mu(X), \sigma(X))$ generates a distribution $\mathbb{P}(X)$

Alternative Methods - Variational Autoencoders

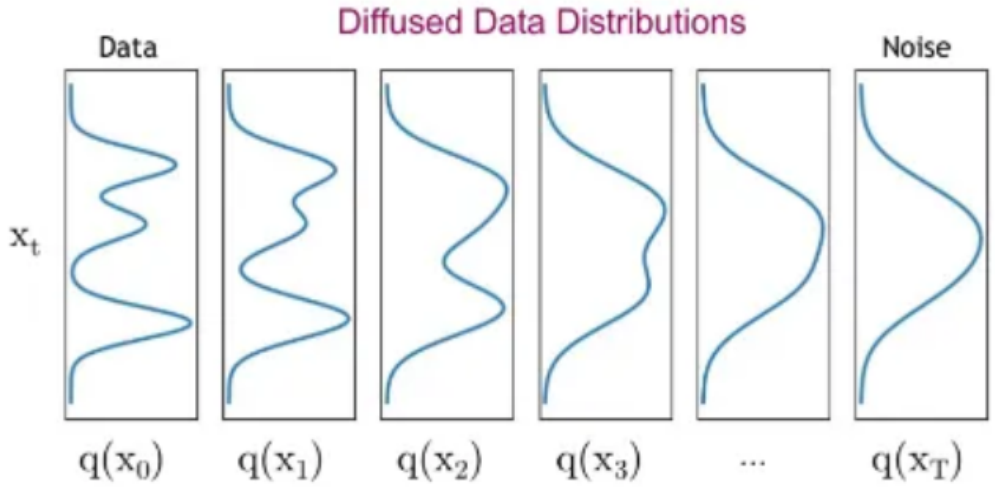


Alternative Methods - Denoising Diffusion Models (DDMs)

- Starts with an image X_0 and adds noise step by step

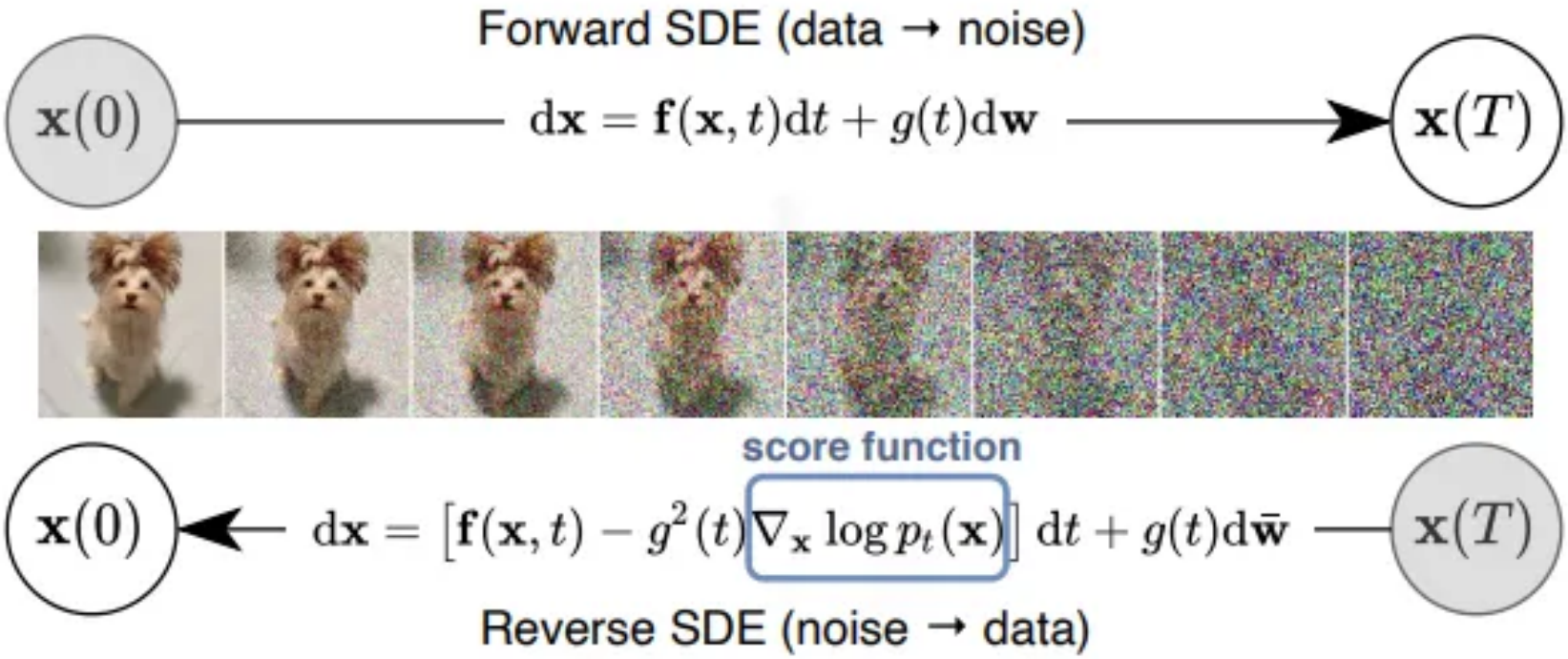
$$X_0 \to X_1 = X_0 + \epsilon_1 \to X_2 = X_1 + \epsilon_2 \to \cdots \to X_T \qquad (1)$$

- $\epsilon_i \sim \mathcal{N}(0, \sigma)$
- $p(X_t|X_{t-1})$ is known since noise distribution is known
- $p(X_{t-1}|X_t)$ is not known but is learned!
- after training, one start with an image, X_T, which is pure noise, and then one "reconstructs" and image, X_0, by successively applying $p(X_{t-1}|X_t)$



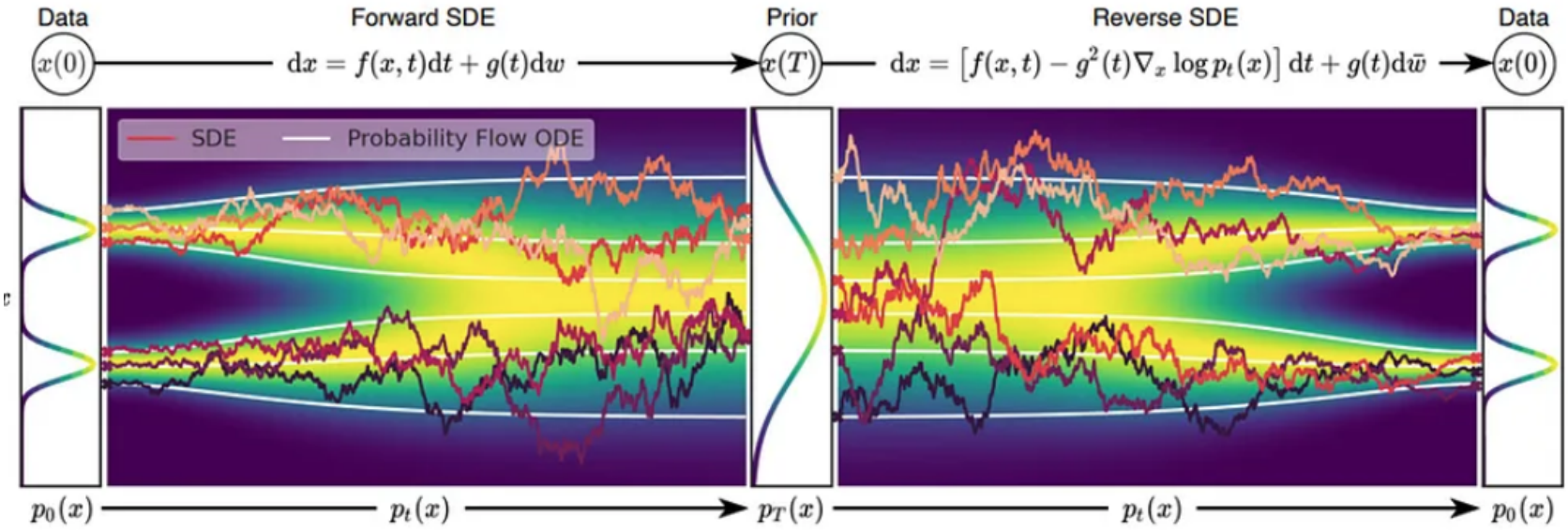
Alternative Methods - Denoising Diffusion Models (DDMs)

- The noising and denoising can be modelled as stochastic differential equations (SDEs)



Alternative Methods - Denoising Diffusion Models (DDMs)

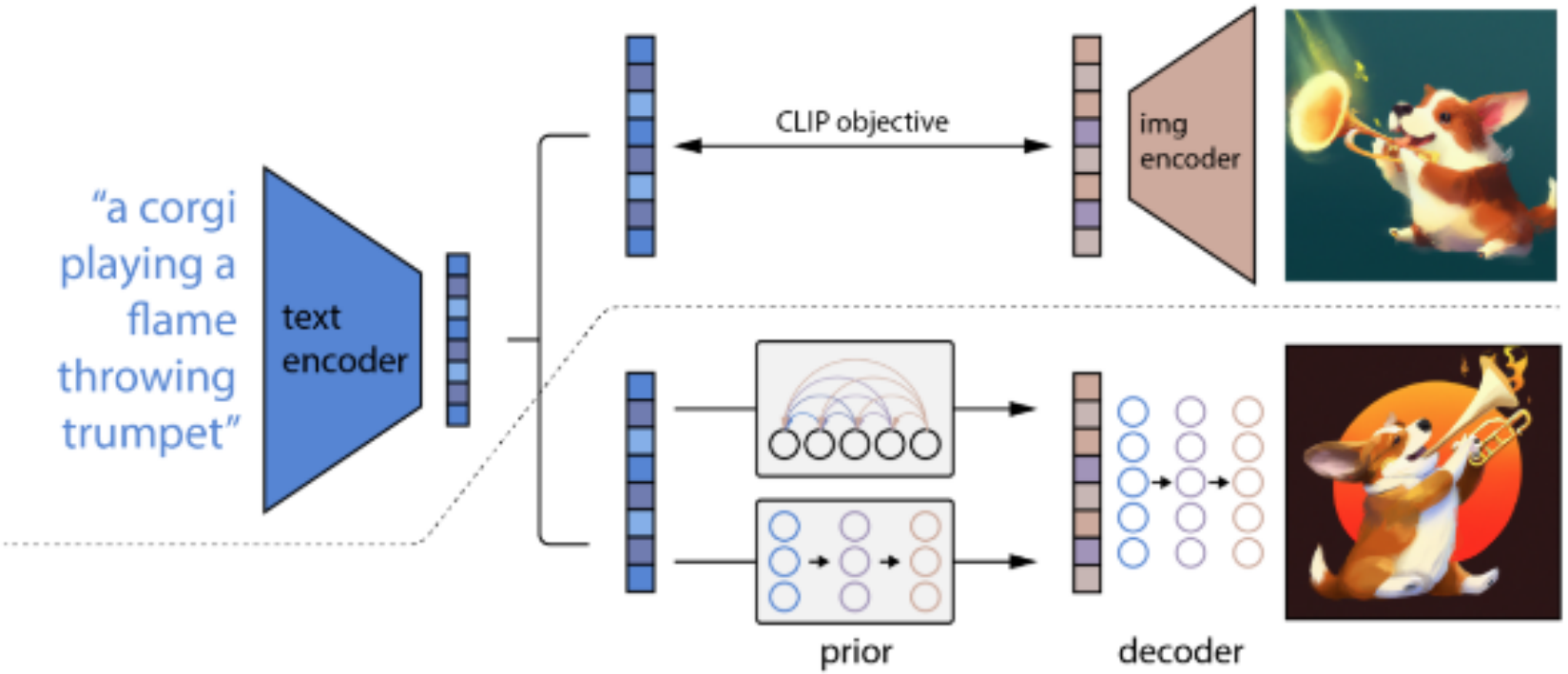
- The noising and denoising can be modelled as stochastic differential equations (SDEs)



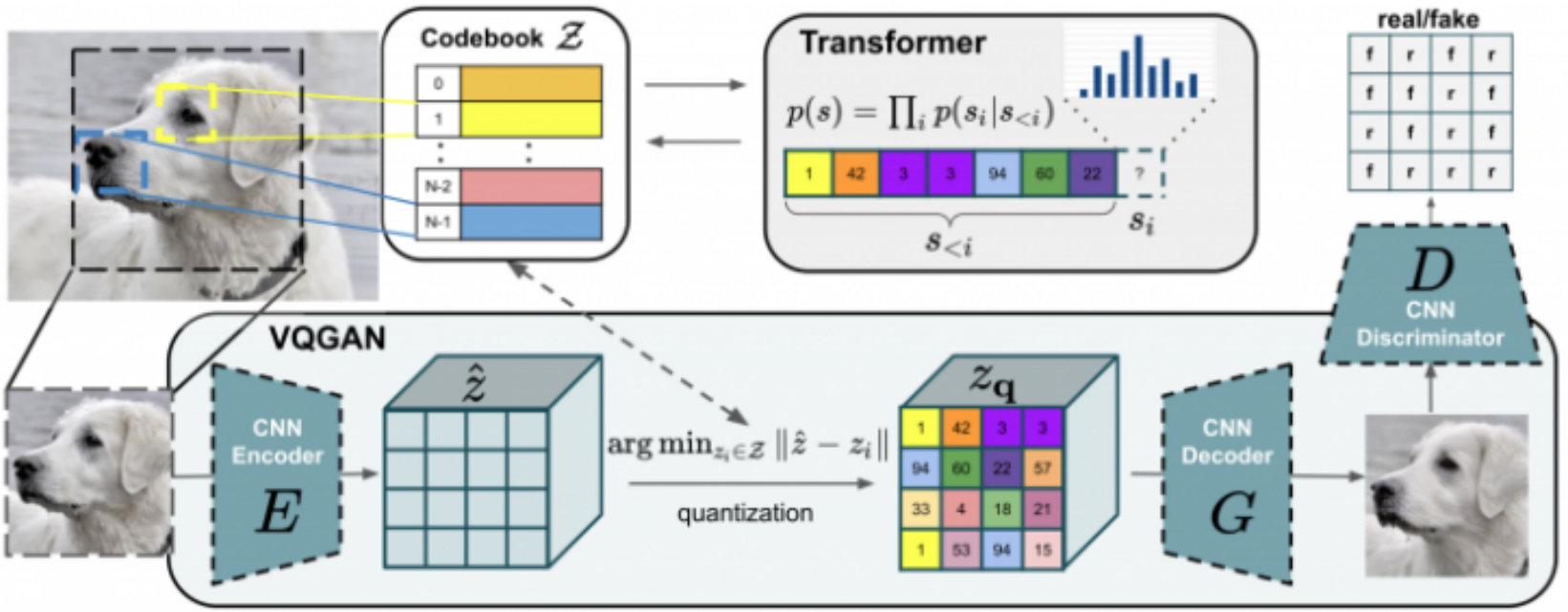
Alternative Methods - Denoising Diffusion Models (DDMs)

- DDMs are more expensive to train
- ... and MUCH more expensive to sample from (compared to GANs)
  - We need to simulate a high-dimensional stochastic differential equation for every sample!
- DDMs are much better at generating conditional samples

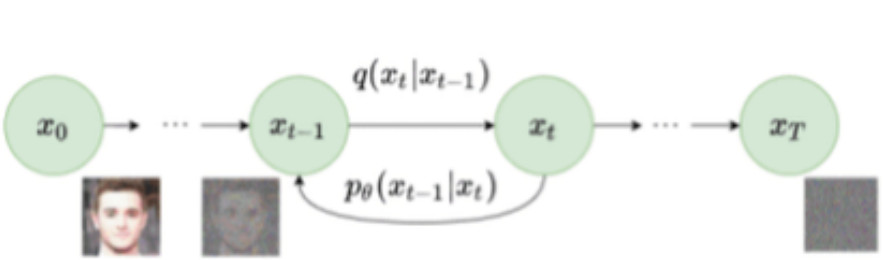
Alternative Methods - DALL-E 2

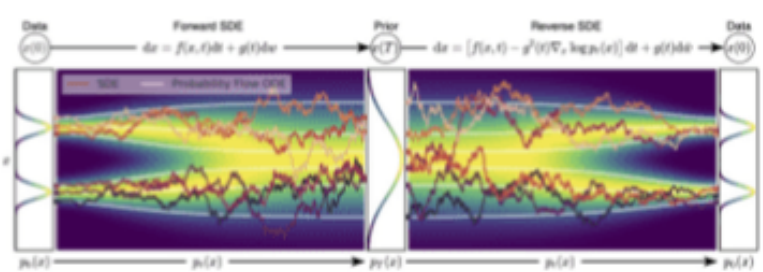


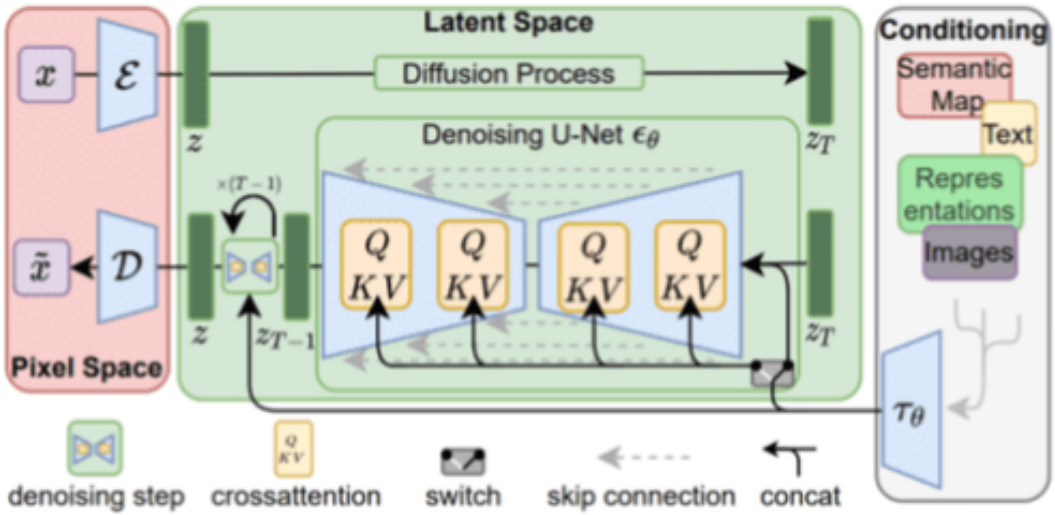
Alternative Methods - Vector Quantized GAN (VQGAN)



Alternative Methods - Stable Diffusion







Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$