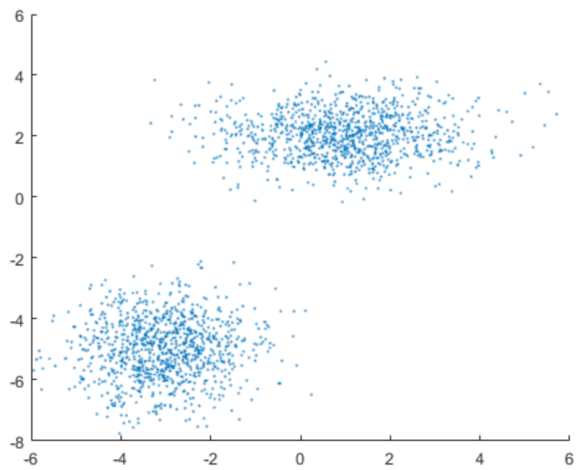
Outline

Example: Gaussian mixture



Consider two Gaussian distributions

$$X_1 \sim \mathcal{N}(\mu_1, \sigma_1^2), \text{ and } X_2 \sim \mathcal{N}(\mu_2, \sigma_2^2), \tag{61}$$

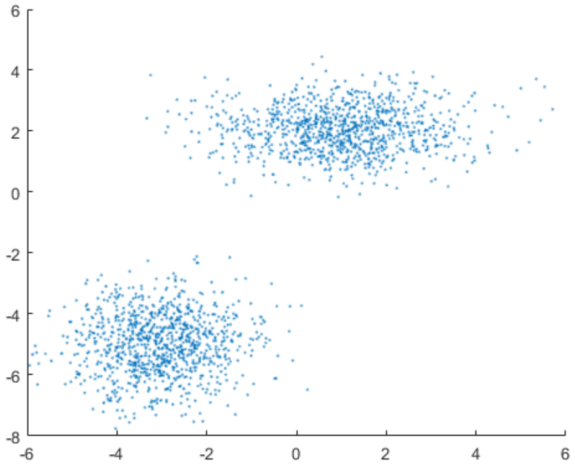
and the mixture

$$X = (1 - \Delta)X_1 + \Delta X_2, \tag{62}$$

where $\Delta \in \{0, 1\}$ with $\Pr[\Delta = 1] = \pi$ and Δ and (X_1, X_2) are independent.
Let ϕ_θ denote the density of a Gaussian random variable parametrized by $\theta = (\mu, \sigma^2)$.

Exercise: How do we estimate θ_1, θ_2?

Solution



Then, the density of X satisfies

$$f_X(x) = (1-\pi)\phi_{\theta_1}(x) + \pi\phi_{\theta_2}(x). \tag{63}$$

One can estimate the parameters $\mu_1, \mu_2, \sigma_1^2, \sigma_2^2$ by maximizing the log likelihood

$$\ell(\theta_1, \theta_2, \mathcal{D}_n) = \sum_{i=1}^{n} \log[(1-\pi)\phi_{\theta_1}(x_i) + \pi\phi_{\theta_2}(x_i)]. \tag{64}$$

Explicit maximization of this expression is numerically difficult, because of the sum of terms inside the logarithm.

Simpler approach: latent variables

Suppose that we know the value Δ_i for all $i = 1, \ldots, n$, that is for each observation we know if it comes from X_1 ($\Delta = 0$) or X_2 ($\Delta = 1$). In that case, the *complete* log likelihood writes

Simpler approach: latent variables

Suppose that we know the value Δ_i for all $i = 1, \dots, n$, that is for each observation we know if it comes from X_1 ($\Delta = 0$) or X_2 ($\Delta = 1$). In that case, the *complete* log likelihood writes

$$\ell_0(\theta_1, \theta_2, \mathcal{D}_n, \Delta) = \sum_{i=1}^{n} \log[(1 - \Delta_i)\phi_{\theta_1}(x_i) + \Delta_i \phi_{\theta_2}(x_i)] \tag{65}$$

$$+ \sum_{i=1}^{n} [\Delta_i \log \pi + (1 - \Delta_i) \log(1 - \pi)]. \tag{66}$$

Since we do not know the values of Δ_i, we are going to replace them by their expected values

Simpler approach: latent variables

Suppose that we know the value Δ_i for all $i = 1, \dots, n$, that is for each observation we know if it comes from X_1 ($\Delta = 0$) or X_2 ($\Delta = 1$). In that case, the *complete* log likelihood writes

$$\ell_0(\theta_1, \theta_2, \mathcal{D}_n, \Delta) = \sum_{i=1}^{n} \log[(1 - \Delta_i)\phi_{\theta_1}(x_i) + \Delta_i \phi_{\theta_2}(x_i)] \tag{65}$$

$$+ \sum_{i=1}^{n} [\Delta_i \log \pi + (1 - \Delta_i) \log(1 - \pi)]. \tag{66}$$

Since we do not know the values of Δ_i, we are going to replace them by their expected values

$$\gamma_i(\theta) = \mathbb{E}[\Delta_i | \theta, \mathcal{D}_n] = \Pr[\Delta_i = 1 | \theta, \mathcal{D}_n]. \tag{67}$$

called the responsibility of model 2 for the ith observation.

We will use an iterative procedure called Expectation-Maximization (EM) algorithm to find estimations of θ_1, θ_2, π.

EM algorithm for Gaussian mixture

1. Initialize $\hat{\mu}_1, \hat{\mu}_2, \hat{\sigma}_1^2, \hat{\sigma}_2^2, \hat{\pi}$.
2. *Expectation step*: compute the responsibilities

$$\hat{\gamma}_i = \frac{\hat{\pi}\phi_{\hat{\theta}_2(x_i)}}{(1-\hat{\pi})\phi_{\hat{\theta}_1(x_i)} + \hat{\pi}\phi_{\hat{\theta}_2(x_i)}} \tag{68}$$

3. *Maximization step*: compute the weighted means and variances

$$\hat{\mu}_1 = \frac{\sum_{i=1}^n (1-\hat{\gamma}_i)x_i}{\sum_{i=1}^n (1-\hat{\gamma}_i)} \quad \hat{\mu}_2 = \frac{\sum_{i=1}^n \hat{\gamma}_i x_i}{\sum_{i=1}^n \hat{\gamma}_i} \tag{69}$$

and

$$\hat{\sigma}_1^2 = \frac{\sum_{i=1}^n (1-\hat{\gamma}_i)(x_i-\hat{\mu}_1)^2}{\sum_{i=1}^n (1-\hat{\gamma}_i)} \quad \hat{\sigma}_2^2 = \frac{\sum_{i=1}^n \hat{\gamma}_i (x_i-\hat{\mu}_2)^2}{\sum_{i=1}^n \hat{\gamma}_i} \tag{70}$$

and the mixing probabilities

$$\hat{\pi} = \frac{1}{n}\sum_{i=1}^n \hat{\gamma}_i. \tag{71}$$

4. Iterate until convergence.

General framework for EM

- A data set $\mathcal{D}_n = \{\mathbf{X}_1, \ldots, \mathbf{X}_n\}$
- The observed log likelihood is given by $\ell(\theta, \mathcal{D}_n)$
- We do not observe latent variables Z_i that drive the behaviour of the observed variables $\mathbf{X}_i$
- If we were to observed Z_i, the log likelihood (called complete log likelihood) would be $\ell_0(\theta, \mathcal{D}_n, \mathbf{Z})$.

General framework for EM

- A data set $\mathcal{D}_n = \{\mathbf{X}_1, \ldots, \mathbf{X}_n\}$
- The observed log likelihood is given by $\ell(\theta, \mathcal{D}_n)$
- We do not observe latent variables Z_i that drive the behaviour of the observed variables $\mathbf{X}_i$
- If we were to observed Z_i, the log likelihood (called complete log likelihood) would be $\ell_0(\theta, \mathcal{D}_n, \mathbf{Z})$.

EM algorithm

1. Start by initializing $\hat{\theta}^{(0)}$.
2. *Expectation step*: at the jth iteration, compute

$$Q(\theta, \hat{\theta}^{(j)}) = \mathbb{E}_{p_{\hat{\theta}^{(j)}}(\mathbf{z}|\mathbf{x})}(\log p_\theta(\mathbf{x}, \mathbf{z})|\mathbf{x}). \tag{72}$$

3. *Maximization step*: at the jth iteration, compute

$$\hat{\theta}^{(j+1)} \in \underset{\theta}{\operatorname{argmax}}\, Q(\theta, \hat{\theta}^{(j)}). \tag{73}$$

4. Repeat step $2-3$ until convergence.

Exercise: Prove that this algorithm converges.

Prove that EM algorithm increases the log-likelihood

We have

$$Q(\theta, \theta') = \mathbb{E}_{p_{\theta'}(\mathbf{z}|\mathbf{x})}(\log p_\theta(\mathbf{x}, \mathbf{z})|\mathbf{x}) \tag{74}$$

$$= \mathbb{E}_{p_{\theta'}(\mathbf{z}|\mathbf{x})}(\log p_\theta(\mathbf{z}|\mathbf{x})|\mathbf{x}) + \mathbb{E}_{p_{\theta'}(\mathbf{z}|\mathbf{x})}(\log p_\theta(\mathbf{x})|\mathbf{x}) \tag{75}$$

$$= \underbrace{\mathbb{E}_{p_{\theta'}(\mathbf{z}|\mathbf{x})}(\log p_\theta(\mathbf{z}|\mathbf{x})|\mathbf{x})}_{R(\theta,\theta')} + \underbrace{\log p_\theta(\mathbf{x})}_{\ell(\theta,\mathcal{D}_n)}. \tag{76}$$

Consequently, by definition of the maximization step, if $\theta \in \operatorname{argmax}_u Q(u, \theta')$,

$$\ell(\theta, \mathcal{D}_n) - \ell(\theta', \mathcal{D}_n) = (Q(\theta, \theta') - Q(\theta', \theta')) - (R(\theta, \theta') - R(\theta', \theta')) \tag{77}$$

$$\geq -(R(\theta, \theta') - R(\theta', \theta')), \tag{78}$$

Besides,

$$R(\theta, \theta') - R(\theta', \theta') = \mathbb{E}_{p_{\theta'}(\mathbf{z}|\mathbf{x})}[\log(p_\theta(\mathbf{z}|\mathbf{x}))|\mathbf{x}] - \mathbb{E}_{p_{\theta'}(\mathbf{z}|\mathbf{x})}[\log(p_{\theta'}(\mathbf{z}|\mathbf{x}))|\mathbf{x}] \tag{79}$$

$$= \mathbb{E}_{p_{\theta'}(\mathbf{z}|\mathbf{x})}\left[\log\left(\frac{p_\theta(\mathbf{z}|\mathbf{x})}{p_{\theta'}(\mathbf{z}|\mathbf{x})}\right)\middle|\mathbf{x}\right] \leq \log\left(\mathbb{E}_{p_{\theta'}(\mathbf{z}|\mathbf{x})}\left[\frac{p_\theta(\mathbf{z}|\mathbf{x})}{p_{\theta'}(\mathbf{z}|\mathbf{x})}\middle|\mathbf{x}\right]\right) \tag{80}$$

$$\leq 0, \tag{81}$$

which proves that,

$$\ell(\theta, \mathcal{D}_n) - \ell(\theta', \mathcal{D}_n) \geq 0. \tag{82}$$

Mathematical Framework - Bayesian point of view

Observed variables: $\mathbf{x} = (x_1, \ldots, x_n)$

(Unobserved) Latent variables $\mathbf{z} = (z_1, \ldots, z_m)$

What we specify:

- Prior on latent variables: $p(\mathbf{z})$
- Likelihood $p(\mathbf{x}|\mathbf{z})$

What we want to know:

- The posterior distribution $p(\mathbf{z}|\mathbf{x})$ (inference)
- The distribution $p(\mathbf{x})$ (density estimation)
- Or, at least, being able to sample from a distribution close to $p(\mathbf{x})$ (generating data)

Solutions

How to determine

- The posterior distribution $p(\mathbf{z}|\mathbf{x})$ (inference)
- The distribution $p(\mathbf{x})$ (density estimation)
- Or, at least, being able to sample from a distribution close to $p(\mathbf{x})$ (generating data)

Easy via Bayes formula

- $p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{x},\mathbf{z})}{p(\mathbf{x})}$
- $p(\mathbf{x}) = \int_{\mathbf{z}} p(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z}$

However, computing the last integral is often intractable (one precise example later).

Two solutions:

- Use Monte Carlo Markov Chain to estimate $\int_{\mathbf{z}} p(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z}$.
- Use variational inference (or GAN) to compute an approximation of $p(\mathbf{z}|\mathbf{x})$.

Differences between MCMC and Variational inference?

Outline

Mathematical Framework - Bayesian point of view

Observed variables: $\mathbf{x} = (x_1, \ldots, x_n)$

(Unobserved) Latent variables $\mathbf{z} = (z_1, \ldots, z_m)$

What we specify:

- Prior on latent variables: $p(\mathbf{z})$
- Complete Likelihood $p_\theta(\mathbf{x}, \mathbf{z})$
- Observed likelihood $p_\theta(\mathbf{x})$

What we want to know:

- The posterior distribution $p_\theta(\mathbf{z}|\mathbf{x})$ (inference)
- The distribution $p_\theta(\mathbf{x})$ (density estimation)
- Or, at least, being able to sample from a distribution close to $p_\theta(\mathbf{x})$ (generating data)

Distribution on latent variables

- Latent informations are really hard to grasp from an image for example: angle, width, color, shape, texture...

- We do not want to create manually our own latent variable
  ["Deep convolutional inverse graphics network", Kulkarni et al. 2015]
  They encourage neurons to represent specific transformation

- Difficult to model dependencies between each component of $\mathbf{Z}$.
  $\rightarrow$ Latent variables do not have clear interpretation and can be drawn from a classic distribution, typically $\mathcal{N}(0, I)$.

  Key argument: Any distribution in d dimensions can be generated by applying a function to a normally distributed vector u of dimension d.

Generate random variables from univariate Gaussian

Any distribution in d dimensions can be generated by applying a function to a normally distributed vector u of dimension d.

Assume that we want to model the distribution of X, which admits a density

$$f(x_1, \dots, x_d) = f_1(x_1) f_2(x_2|x_1) f_3(x_3|x_1, x_2) \dots f_d(x_d|x_1, \dots, x_{d-1}). \tag{83}$$

Let $Z = (Z_1, \dots, Z_d) \sim \mathcal{N}(0, I)$, F_1 be the cumulative distribution function of X_1 and ϕ be the cumulative distribution function of a standard normal random variable. Then,

$$\phi^{-1}(F_1(X_1)) \sim Z_1 \tag{84}$$

that is,

$$X_1 \sim F_1^{-1}(\phi(Z_1)) \tag{85}$$

Now, denote by F_{2,x_1} the cdf of X_2 conditional on $X_1 = x_1$. Then,

$$X_2 \sim F_{2,x_1}^{-1}(\phi(Z_2)) \tag{86}$$

And so on...

More on generating random variables: ["Sample-based non-uniform random variate generation", Devroye 1986]

Rewritting the likelihood

First, we need a distance between distribution, for example the Kullback-Leibler divergence between two densities p and q, defined as

$$KL(p||q) = \int p(\mathbf{z}) \log \left(\frac{p(\mathbf{z})}{q(\mathbf{z})} \right) d\mathbf{z}. \tag{87}$$

- Prove that, the observed log likelihood can be written as

$$\log p_\theta(\mathbf{x}) = ELBO(q_\lambda) + KL(q_\lambda(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})), \tag{88}$$

  where

$$\begin{aligned} ELBO(q_\lambda) &= \log p_\theta(\mathbf{x}) - KL(q_\lambda(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})) && (89)\\ &= \mathbb{E}_{q_\lambda}[\log p_\theta(\mathbf{x}, \mathbf{z})] - \mathbb{E}_{q_\lambda}[\log q_\lambda(\mathbf{z}|\mathbf{x})] && (90)\\ &= -KL(q_\lambda(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{x}, \mathbf{z})) && (91)\\ &= \mathbb{E}_{q_\lambda}[\log p_\theta(\mathbf{x}|\mathbf{z})] - KL(q_\lambda(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})). && (92) \end{aligned}$$

- Prove that $\log p_\theta(\mathbf{x}) \geq ELBO(q_\lambda)$.

Our aim is to maximize $\log p_\theta(\mathbf{x})$. However, computing

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z} \tag{93}$$

is often intractable, we instead want to maximize the lower bound

$$ELBO(\lambda, \theta) = \mathbb{E}_{q_\lambda}[\log p_\theta(\mathbf{x}|\mathbf{z})] - KL(q_\lambda(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})). \tag{94}$$

First optimization idea

New algorithm

Start with $\theta^{(0)}, \lambda^{(0)}$. While $\theta^{(t)}, \lambda^{(t)}$ do not converge, do

1. Find $\lambda^{(t+1)}$ maximizing
$$\mathbb{E}_{q_\lambda}[\log p_{\theta^{(t)}}(\mathbf{x}|\mathbf{z})] - KL(q_\lambda(\mathbf{z}|\mathbf{x})||p_{\theta^{(t)}}(\mathbf{z}))$$
2. Find $\theta^{(t+1)}$ maximizing
$$\mathbb{E}_{q_{\lambda^{(t+1)}}}[\log p_\theta(\mathbf{x}|\mathbf{z})] - KL(q_{\lambda^{(t+1)}}(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}))$$

Exercise: Prove that this algorithm converges.

Proof of convergence

Assume that we are given $\theta^{(t)}, \lambda^{(t)}$. According to the first step, using the first expression of ELBO,

$$\lambda^{(t+1)} \in \underset{\lambda}{\text{argmax}} \left(\log p_{\theta^{(t)}}(\mathbf{x}) - KL(q_\lambda(\mathbf{z}|\mathbf{x})||p_{\theta^{(t)}}(\mathbf{z}|\mathbf{x})) \right) \tag{95}$$

$$\Leftrightarrow \lambda^{(t+1)} \in \underset{\lambda}{\text{argmin}}\, KL(q_\lambda(\mathbf{z}|\mathbf{x})||p_{\theta^{(t)}}(\mathbf{z}|\mathbf{x})) \tag{96}$$

$$\Leftrightarrow \lambda^{(t+1)} \text{ s.t. } q_{\lambda^{(t+1)}}(\mathbf{z}|\mathbf{x}) = p_{\theta^{(t)}}(\mathbf{z}|\mathbf{x}). \tag{97}$$

Thus, after the first step,

$$ELBO(\lambda^{(t+1)}, \theta^{(t)}) = \log p_{\theta^{(t)}}(\mathbf{x}). \tag{98}$$

By definition of the second optimization step, we have

$$ELBO(\lambda^{(t+1)}, \theta^{(t+1)}) \geq ELBO(\lambda^{(t+1)}, \theta^{(t)}) = \log p_{\theta^{(t)}}(\mathbf{x}). \tag{99}$$

However

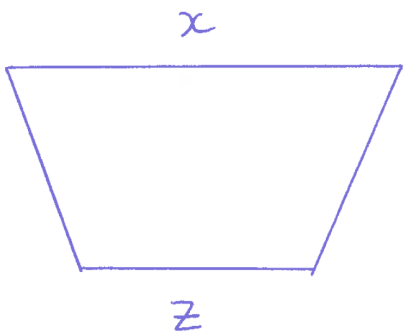
$$ELBO(\lambda^{(t+1)}, \theta^{(t+1)}) = \log p_{\theta^{(t+1)}}(\mathbf{x}) - KL(q_{\lambda^{(t+1)}}(\mathbf{z}|\mathbf{x})||p_{\theta^{(t+1)}}(\mathbf{z}|\mathbf{x})) \tag{100}$$

$$\leq \log p_{\theta^{(t+1)}}(\mathbf{x}). \tag{101}$$

Consequently, after the M step,

$$\log p_{\theta^{(t+1)}}(\mathbf{x}) \geq \log p_{\theta^{(t)}}(\mathbf{x}). \tag{102}$$

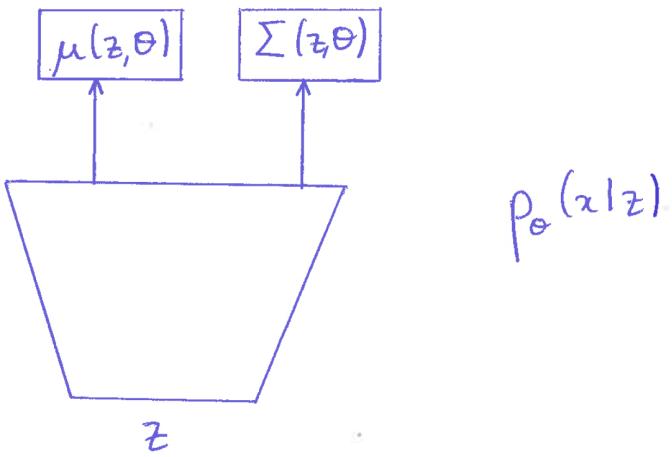
But where are neural networks?



$p_\theta(x|z)$

We model the probability $p_\theta(x|z)$ by a Gaussian distribution whose parameters are estimated with a neural network parametrized by the bias/weights θ.

But where are neural networks?



We model the probability $p_\theta(x|z)$ by a Gaussian distribution whose parameters are estimated with a neural network parametrized by the bias/weights θ.

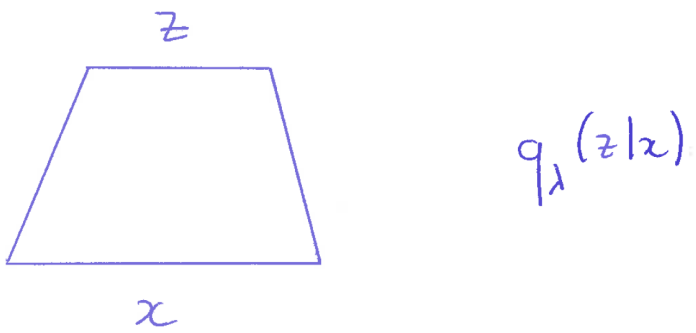
Precisely,

$$p_\theta(x|z) = f_{\mu(z,\theta),\Sigma(z,\theta)}(x),$$

where $f_{\mu(z,\theta),\Sigma(z,\theta)}$ is the density of a Gaussian of mean $\mu(z,\theta)$ and covariance matrix $\Sigma(z,\theta)$.

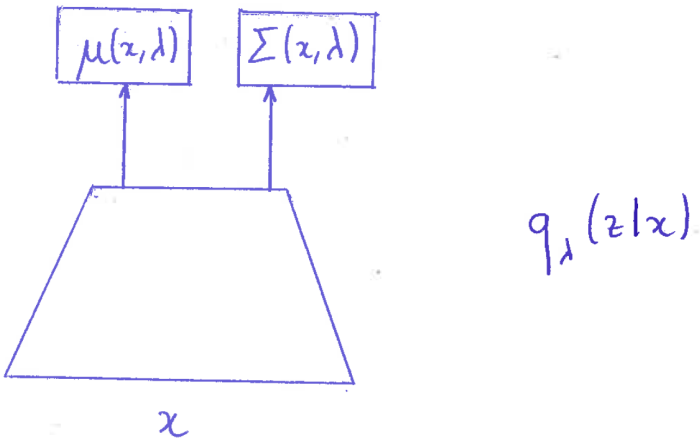
Here, $\mu(z,\theta)$ and $\Sigma(z,\theta)$ are the output of a neural network parametrized by θ whose input is z.

But where are neural networks?



Similarly, we model the probability $q_\lambda(z|x)$ by a Gaussian distribution whose parameters are estimated with a neural network parametrized by the bias/weights λ.

But where are neural networks?



Similarly, we model the probability $q_\lambda(z|x)$ by a Gaussian distribution whose parameters are estimated with a neural network parametrized by the bias/weights λ.

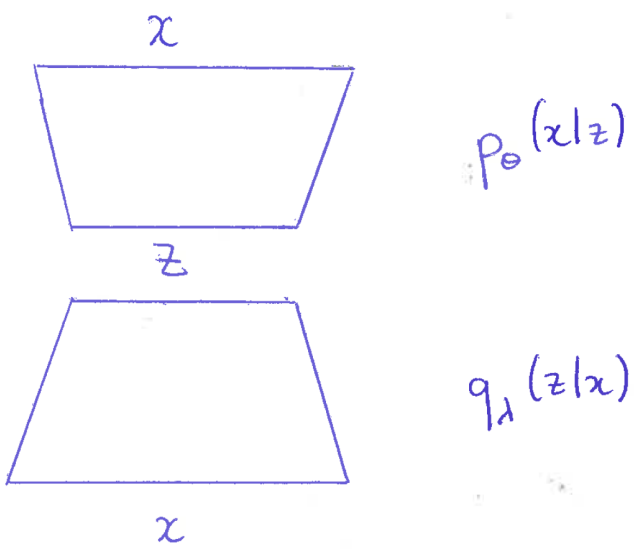
Precisely,

$$q_\lambda(z|x) = f_{\mu(x,\lambda),\Sigma(x,\lambda)}(z),$$

where $f_{\mu(x,\lambda),\Sigma(z,\lambda)}$ is the density of a Gaussian of mean $\mu(x,\lambda)$ and covariance matrix $\Sigma(z,\lambda)$.

Here, $\Sigma(z,\lambda)$ and $\Sigma(z,\lambda)$ are the output of a neural network parametrized by λ whose input is x.

Variational autoencoder

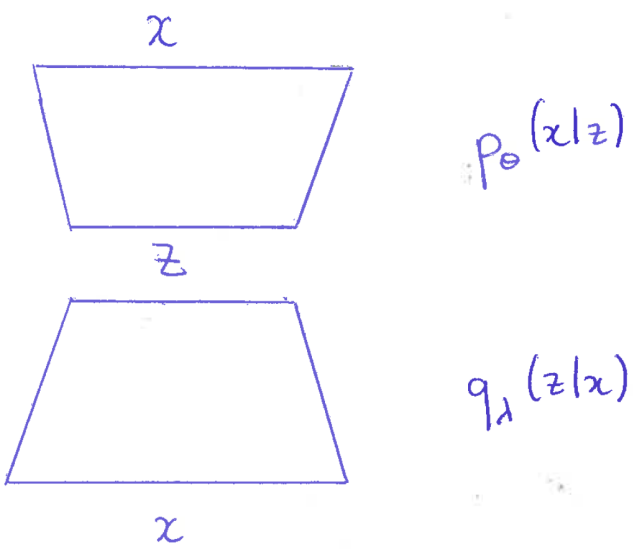


Optimization procedure should be:

Start with $\theta^{(0)}, \lambda^{(0)}$. Then iterate:

1. Find $\lambda^{(t+1)}$ maximizing
$$\mathbb{E}_{q_\lambda}[\log p_{\theta^{(t)}}(\mathbf{x}|\mathbf{z})] - KL(q_\lambda(\mathbf{z}|\mathbf{x})||p_{\theta^{(t)}}(\mathbf{z}))$$
2. Find $\theta^{(t+1)}$ maximizing
$$\mathbb{E}_{q_{\lambda^{(t+1)}}}[\log p_\theta(\mathbf{x}|\mathbf{z})] - KL(q_{\lambda^{(t+1)}}(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}))$$

Variational autoencoder



Optimization procedure should be:

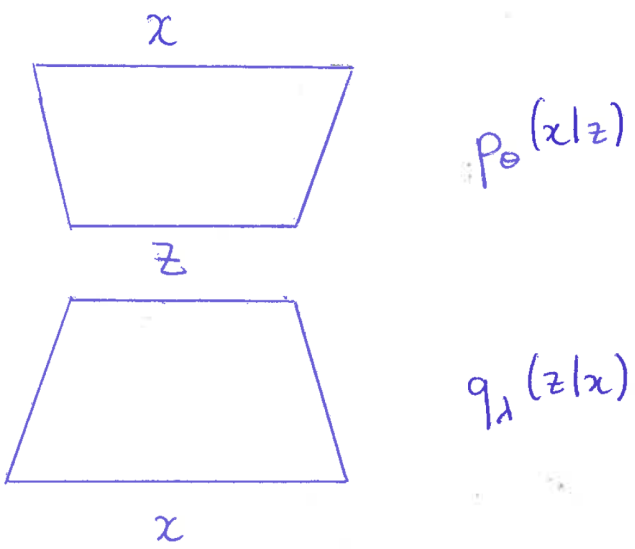
Start with $\theta^{(0)}, \lambda^{(0)}$. Then iterate:

1. Find $\lambda^{(t+1)}$ maximizing
$$\mathbb{E}_{q_\lambda}[\log p_{\theta^{(t)}}(\mathbf{x}|\mathbf{z})] - KL(q_\lambda(\mathbf{z}|\mathbf{x})||p_{\theta^{(t)}}(\mathbf{z}))$$
2. Find $\theta^{(t+1)}$ maximizing
$$\mathbb{E}_{q_{\lambda^{(t+1)}}}[\log p_\theta(\mathbf{x}|\mathbf{z})] - KL(q_{\lambda^{(t+1)}}(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}))$$

Too time consuming: amounts to fully optimizing two neural networks at each iteration.

Hint: do stochastic gradient descent instead!

Variational autoencoder - stochastic gradient descent



Metric: $\mathbb{E}_{q_\lambda}[\log p_{\theta^{(t)}}(\mathbf{x}|\mathbf{z})] - KL(q_\lambda(\mathbf{z}|\mathbf{x})||p_{\theta^{(t)}}(\mathbf{z}))$

Procedure:

Start with $\theta^{(0)}, \lambda^{(0)}$. Then iterate:

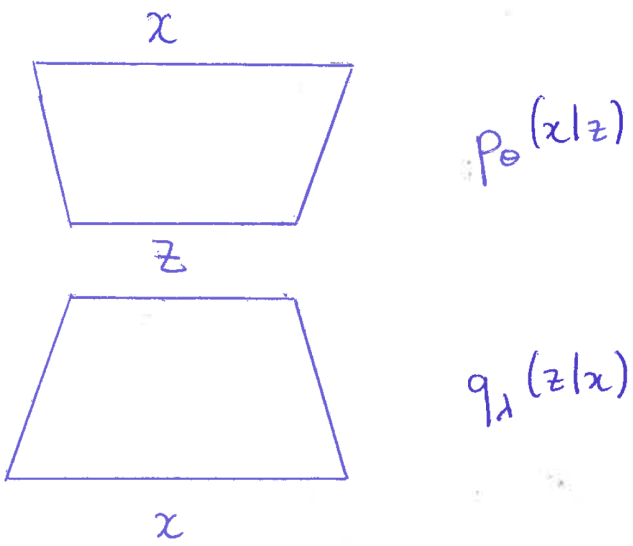
1. Sample $i \in \{1, \ldots, n\}$
2. Draw m $i.i.d.$ samples $z_{i,j} \sim q_{\lambda^{(t)}}(z|x_i)$ for $j = 1, \ldots, m$.
3. Compute the empirical ELBO version at x_i

$$\widetilde{ELBO}(\lambda^{(t)}, \theta^{(t)}, x_i) = \frac{1}{m}\sum_{j=1}^{m} \log p_{\theta^{(t)}}(x_i|z_{i,j}) - KL(q_{\lambda^{(t)}}(z_i|x_i)||p(z_i)). \quad (103)$$

4. Backpropagate the error through the network.
5. Update at once θ and λ.

Any problem?

Reparametrization trick



Metric: $\mathbb{E}_{q_\lambda}[\log p_{\theta^{(t)}}(\mathbf{x}|\mathbf{z})] - KL(q_\lambda(\mathbf{z}|\mathbf{x})||p_{\theta^{(t)}}(\mathbf{z}))$
The gradient flow is blocked in the middle of the autoencoder: z_i depends randomly of the output of the encoder.
We need to reparametrized the network. Instead of drawing $z_i \sim q_\lambda(z|x_i)$ we let

$$z_i = g(\varepsilon_i, x_i, \lambda),$$

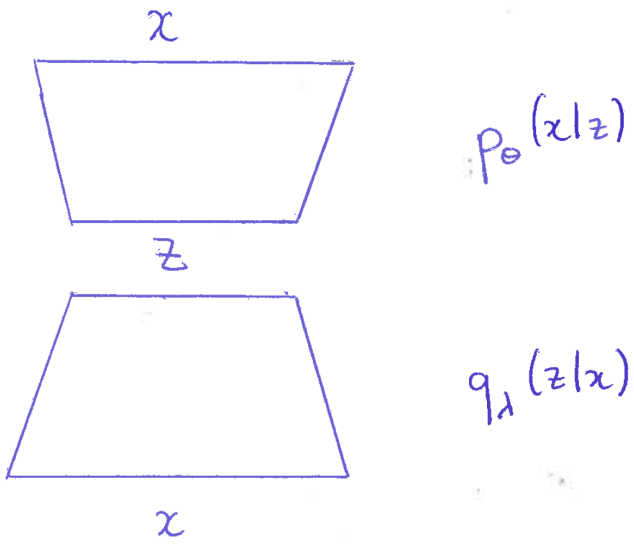
and draw $\varepsilon_i \sim p(\varepsilon)$. In that way, z_i is still random but depends deterministically of the parameters λ.
Example: $Z \sim \mathcal{N}(\mu, \sigma^2)$ is equivalent to $Z = g(\varepsilon)$ where

- $\varepsilon \sim \mathcal{N}(0, 1)$,
- $g(u) = \mu + u\sigma$.

Need to replace expectations with respect to q_λ by expectations with respect to p that is replacing

$$\mathbb{E}_{q_\lambda(z|x)}[f(z)] \text{ by } \mathbb{E}_{p(\varepsilon)}[f(g(\varepsilon, x_i, \lambda))].$$

Final Variational autoencoder



Metric: $\mathbb{E}_{q_\lambda}[\log p_{\theta^{(t)}}(\mathbf{x}|\mathbf{z})] - KL(q_\lambda(\mathbf{z}|\mathbf{x})||p_{\theta^{(t)}}(\mathbf{z}))$

Procedure:
Start with $\theta^{(0)}, \lambda^{(0)}$. Then iterate:

1. Sample $i \in \{1, \ldots, n\}$
2. Draw m $i.i.d.$ samples $\varepsilon_{i,j} \sim p(\varepsilon)$ for $j = 1, \ldots, m$ and set $z_{i,j} = g(\varepsilon_{i,j}, \mathbf{x}_i, \lambda)$.
3. Compute the empirical ELBO version at x_i

$$\widetilde{ELBO}(\lambda^{(t)}, \theta^{(t)}, x_i) = \frac{1}{m}\sum_{j=1}^{m} \log p_{\theta^{(t)}}(x_i|z_{i,j}) - KL(q_{\lambda^{(t)}}(z_i|x_i)||p(z_i)).$$

4. Backpropagate the error through the network.
5. Update at once θ and λ.

No more problem with gradient flow.

How to compute KL divergence for Gaussian

Solving

$$\lambda^\star \in \underset{\lambda}{\text{argmax}} \Big(\mathbb{E}_{q_\lambda}[\log p(\mathbf{x}|\mathbf{z})] - KL(q_\lambda(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) \Big). \tag{104}$$

Regarding the second term, recall that we assume

- $p(\mathbf{z}) \sim \mathcal{N}(0, I)$
- $q_\lambda(\mathbf{z}|\mathbf{x}) \sim \mathcal{N}(\mu(\mathbf{x}, \lambda), \Sigma(\mathbf{x}, \lambda))$.

We have

$$KL(\mathcal{N}(\mu_0, \Sigma_0)||\mathcal{N}(\mu_1, \Sigma_1)) = \frac{1}{2}\Big(Tr(\Sigma_1^{-1}\Sigma_0) + (\mu_1 - \mu_0)^T \Sigma_1^{-1}(\mu_1 - \mu_0) - d + \log\left(\frac{\det \Sigma_1}{\det \Sigma_0}\right)\Big). \tag{105}$$

In our case,

$$KL(q_\lambda(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) = \frac{1}{2}\Big(Tr(\Sigma(\mathbf{x}, \lambda)) + (\mu(\mathbf{x}, \lambda))^T(\mu(\mathbf{x}, \lambda)) - d - \log\det \Sigma(\mathbf{x}, \lambda)\Big) \tag{106}$$

Optimization problem as a penalization

Solving

$$(\lambda^\star, \theta^\star) \in \underset{\lambda,\theta}{\text{argmax}}\, ELBO(\lambda, \theta)$$

where

$$ELBO(\lambda, \theta) = \mathbb{E}_{q_\lambda}[\log p_\theta(\mathbf{x}|\mathbf{z})] - KL(q_\lambda(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})).$$

- The first term is an expected likelihood: it forces the distribution $q_\lambda^\star$ to be such that the probability of observing $\mathbf{x}$ is large.
- The second term can be seen as a penalty: it forces the distribution $q_\lambda^\star$ to be close to the prior $p_\theta(\mathbf{z})$.

Note that there is no global latent variable: $\mathbf{X}_1, \ldots, \mathbf{X}_n$ are independent and $\mathbf{X}_i$ depends only on $\mathbf{Z}_i$ and not on the other $\mathbf{Z}_j$, for $j \neq i$.

Variational Auto Encoders: remarks

Requirements for VAE

- Prior distribution $p(z)$ must be easy to sample from.
- Conditional likelihood $p_\theta(x|z)$ must be computable.

In practice, these two distributions are often uniform or Gaussian.

Benefits of VAE

- The prior on the latent space allow to inject information in the distribution of z
- Possibility to estimate variance/uncertainty in predictions.

Outline

Generative Adversarial Networks (GAN)

["Generative adversarial nets", Goodfellow et al. 2014]

Data: $X_1, \ldots, X_n$ *i.i.d.* drawn from $p^\star$.

Aim: generating new data whose distribution is (approximately) $p^\star$.

GAN: Two devices working together/against each other.

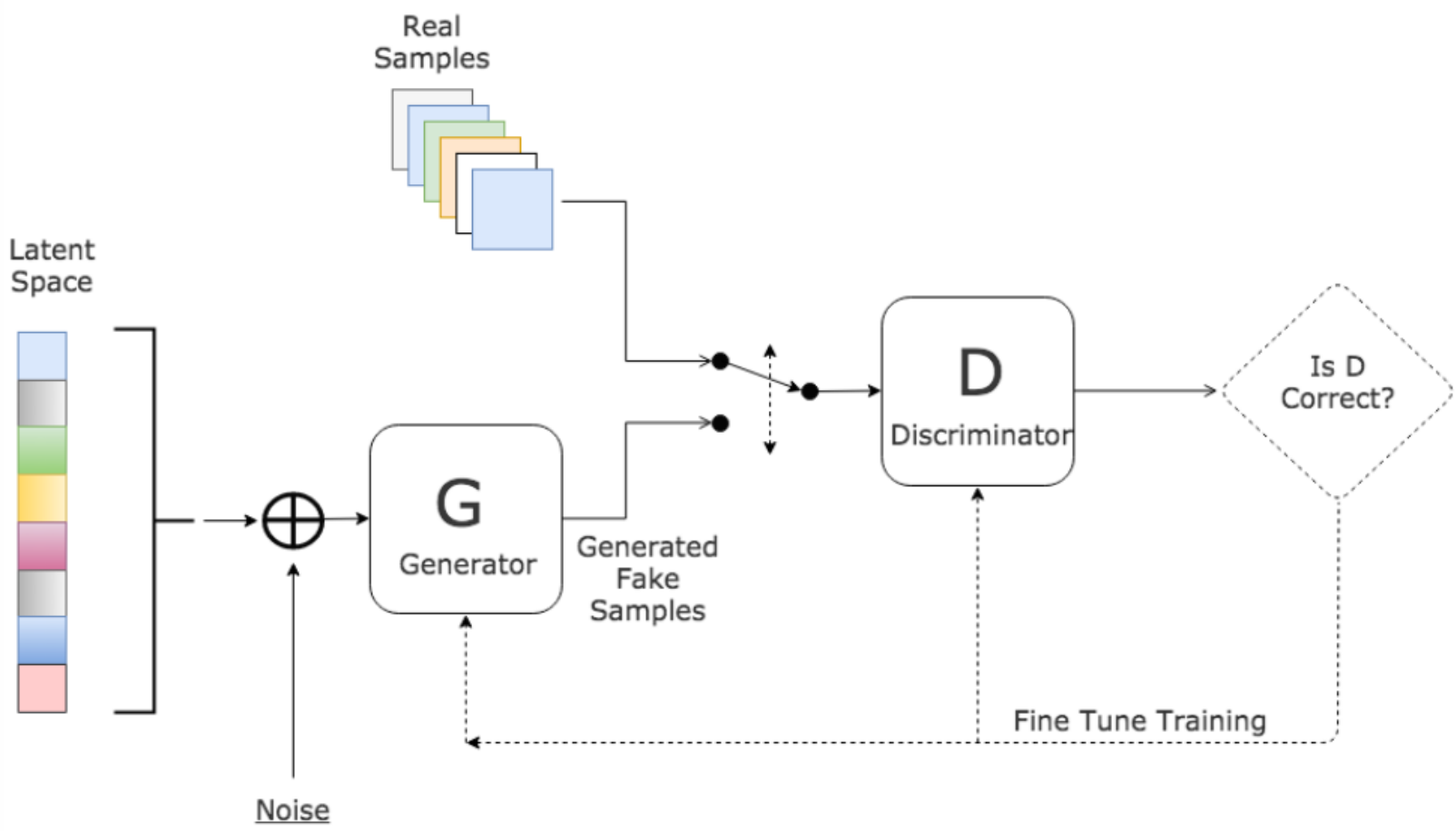
- **A generator** G that takes low-dimensional noise/latent random variable as input (usually Gaussian or uniformly distributed) and turn them into fake data that closely resemble the original data set.

- **A discriminator** D that takes as input the original data and the fake data and try to distinguish between them.

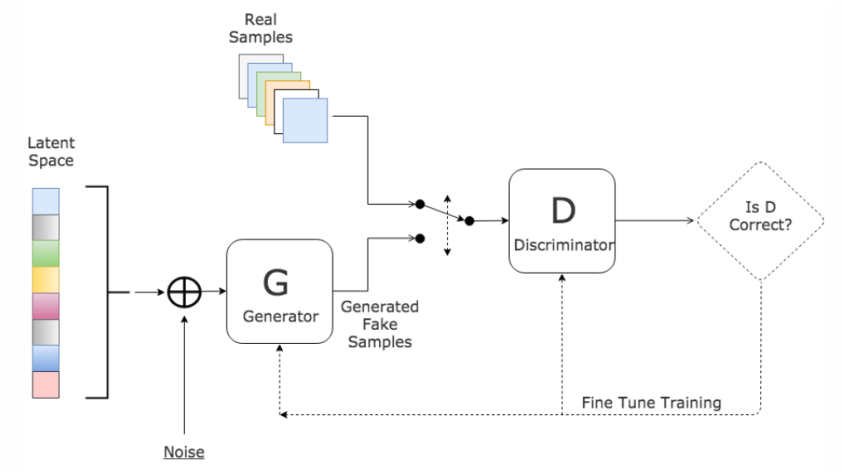
The **goal of the generator** is to fool the discriminator whereas the **goal of the discriminator** is to discriminate the true samples from the fake one. Hence, the name adversarial learning. (cops and robbers analogy)

The game stops when an **equilibrium is reached**. The ideal situation would be the one in which the generator has learned to replicate the original sample and the discriminator cannot differentiate true and fake data.

Picture of adversarial learning



Definitions



In GAN, the generator (resp. the discriminator) is a neural network parametrized by θ_g (resp. θ_d).

The latent/input variable for the generator is denoted by $\mathbf{Z}$. For a given realization $\mathbf{z}$, the generator outputs $G(\mathbf{z}, \theta_g)$.

For a true or a fake sample $\mathbf{x}$, the discriminator outputs a value $D(\mathbf{x}, \theta_d)$, ranging from 0 to 1, which is the estimated probability that the sample $\mathbf{x}$ belongs to the original data set (rather than being a fake copy).

Parameter estimation

How to estimate $\boldsymbol{\theta}_g$ and $\boldsymbol{\theta}_d$?

We use the log likelihood to evaluate the **performance of the discriminator**. The expectation of the log likelihood of the discriminator is thus

$$\mathbb{E}_{\mathbf{x}\sim p^\star}[\log D(\mathbf{x}, \boldsymbol{\theta}_d)] + \mathbb{E}_{\mathbf{z}\sim p_z}[\log(1 - D(G(\mathbf{z}, \boldsymbol{\theta}_g), \boldsymbol{\theta}_d))]. \tag{107}$$

Besides, we want the **generator to fool the discriminator**, that is minimizing

$$\mathbb{E}_{\mathbf{z}\sim p_\mathbf{z}}[\log(1 - D(G(\mathbf{z}, \boldsymbol{\theta}_g), \boldsymbol{\theta}_d))]. \tag{108}$$

All in all, the problem is equivalent to

$$(\boldsymbol{\theta}_g^\star, \boldsymbol{\theta}_d^\star) \in \underset{\boldsymbol{\theta}_g}{\operatorname{argmin}} \underset{\boldsymbol{\theta}_d}{\operatorname{argmax}}\, V(\boldsymbol{\theta}_g, \boldsymbol{\theta}_d), \tag{109}$$

where $V(\boldsymbol{\theta}_g, \boldsymbol{\theta}_d)$ is the objective/value function defined as

$$V(\boldsymbol{\theta}_g, \boldsymbol{\theta}_d) = \mathbb{E}_{\mathbf{x}\sim p^\star}[\log D(\mathbf{x}, \boldsymbol{\theta}_d)] + \mathbb{E}_{\mathbf{z}\sim p_\mathbf{z}}[\log(1 - D(G(\mathbf{z}, \boldsymbol{\theta}_g), \boldsymbol{\theta}_d))]. \tag{110}$$

First theoretical results

The optimization problem we want to solve is

$$(\boldsymbol{\theta}_g^\star, \boldsymbol{\theta}_d^\star) \in \underset{\boldsymbol{\theta}_g}{\operatorname{argmin}} \underset{\boldsymbol{\theta}_d}{\operatorname{argmax}} V(\boldsymbol{\theta}_g, \boldsymbol{\theta}_d), \tag{111}$$

where $V(\boldsymbol{\theta}_g, \boldsymbol{\theta}_d)$ is the objective/value function defined as

$$V(\boldsymbol{\theta}_g, \boldsymbol{\theta}_d) = \mathbb{E}_{\mathbf{x}\sim p^\star}[\log D(\mathbf{x}, \boldsymbol{\theta}_d)] + \mathbb{E}_{\mathbf{z}\sim p_\mathbf{z}}[\log(1 - D(G(\mathbf{z}, \boldsymbol{\theta}_g), \boldsymbol{\theta}_d))]. \tag{112}$$

Exercise.

1. For a fixed generator, the optimal discriminator $D^\star$ is given by

$$D^\star(\mathbf{x}) = \frac{p^\star(\mathbf{x})}{p^\star(\mathbf{x}) + p_G(\mathbf{x})}. \tag{113}$$

2. Let $C(G) = \max_D V(G, D)$. Then the global minimum of C is achieved if and only if

$$p_G = p^\star.$$

   At that point, $C(G) = -\log 4$.

Proof

First question.

Consider a fixed generator G. Then the optimal discriminator is a maximizer of

$$V(G,D) = \int_{\mathbf{x}} p^\star(\mathbf{x}) \log(D(\mathbf{x}))d\mathbf{x} + \int_{\mathbf{z}} p_{\mathbf{z}}(\mathbf{z}) \log(1 - D(G(\mathbf{z})))d\mathbf{z} \tag{114}$$

$$= \int_{\mathbf{x}} (p^\star(\mathbf{x}) \log(D(\mathbf{x})) + p_G(\mathbf{x}) \log(1 - D(\mathbf{x})))dz. \tag{115}$$

But, for any $(a, b) \neq (0, 0)$, the function $u \mapsto a \log(u) + b \log(1 - u)$ achieves its maximum in $[0, 1]$ at $a/(a + b)$.

This gives the result for all $\mathbf{x}$ such that $(p^\star(\mathbf{x}), p_G(\mathbf{x})) \neq (0, 0)$.

Since the discriminator needs not to be defined outside $\mathrm{Supp}(p^\star + p_G)$ this concludes the proof.

Proof

Second question.

According to the previous result, for $p_G = p^\star$, $D_G^\star(\mathbf{x}) = 1/2$. Note that

$$C(G) = \max_D V(G, D) \tag{116}$$

$$= \mathbb{E}_{\mathbf{x}\sim p^\star}[\log(D_G^\star)] + \mathbb{E}_{\mathbf{z}\sim p_\mathbf{z}}[\log(1 - D_G^\star(G(\mathbf{z})))] \tag{117}$$

$$= \mathbb{E}_{\mathbf{x}\sim p^\star}[\log(D_G^\star)] + \mathbb{E}_{\mathbf{x}\sim p_G}[\log(1 - D_G^\star(\mathbf{x}))] \tag{118}$$

$$= \mathbb{E}_{\mathbf{x}\sim p^\star}\left[\log\left(\frac{p^\star(\mathbf{x})}{p^\star(\mathbf{x}) + p_G(\mathbf{x})}\right)\right] + \mathbb{E}_{\mathbf{x}\sim p_G}\left[\log\left(\frac{p_G(\mathbf{x})}{p^\star(\mathbf{x}) + p_G(\mathbf{x})}\right)\right]. \tag{119}$$

According to the second equality, $C(G) = -\log 4$ if $D_G^\star(\mathbf{x}) = 1/2$. To prove that this is the optimal value, note that

$$C(G) = -\log 4 + KL\left(p^\star \middle\| \frac{p^\star + p_G}{2}\right) + KL\left(p_G \middle\| \frac{p^\star + p_G}{2}\right). \tag{120}$$

Consequently $C(G) \geq -\log 4$ with equality if and only if $p_G = p^\star$ (according to the properties of KL divergence).

GAN Algorithm

Inputs: prior distribution $p_{\mathbf{z}}$ on noise/latent variables $\mathbf{Z}$, initialization of the parameters $\boldsymbol{\theta}_d^{(0)}$ of the discriminator and $\boldsymbol{\theta}_g^{(0)}$ of the generator, learning rate η, minibatch size m.

1. Discriminator optimization
   1. Sample m noise samples $\{\mathbf{z}_1, \ldots, \mathbf{z}_m\}$ from noise prior $p_{\mathbf{z}}$.
   2. Sample m examples $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ from the original data distribution.
   3. Update the discriminator by stochastic gradient ascent
2. Generator optimization
   1. Sample m noise samples $\{\mathbf{z}_1, \ldots, \mathbf{z}_m\}$ from noise prior $p_{\mathbf{z}}$.
   2. Update the generator by stochastic gradient descent

$$\boldsymbol{\theta}_G^{(t+1)} = \boldsymbol{\theta}_G^{(t)} - \eta \left(\frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}_G} [\log(1 - D(G(\mathbf{z}_i, \boldsymbol{\theta}_G), \boldsymbol{\theta}_d^{(t+1)}))] \right).$$

Note that any version of stochastic gradient descent can be used. Momentum was used in the seminal paper

["Generative adversarial nets", Goodfellow et al. 2014]

Kullback-Leibler divergence and Jensen Shannon divergence

Recall that the Kullback-Leibler divergence between two densities p and q is defined as

$$KL(p||q) = \int \log\left(\frac{p(x)}{q(x)}\right) p(x)dx. \tag{121}$$

Properties

$KL(p||q) \geq 0$ with equality if and only if $p = q$ almost everywhere.

The Jensen-Shannon divergence between two densities p and q is defined as

$$JSD(p||q) = \frac{1}{2}KL\left(p||\frac{p+q}{2}\right) + \frac{1}{2}KL\left(q||\frac{p+q}{2}\right). \tag{122}$$

Properties

The Jensen-Shannon divergence is symmetric and satisfies

$$0 \leq JSD(p||q) \leq \log 2. \tag{123}$$

The optimization problem can be rewritten using Jensen-Shannon divergence:

$$C(G) = \max_D V(G, D) = -\log 4 + 2JSD(p^\star||p_G). \tag{124}$$

Proof: basic divergence inequalities

KL properties.

$$KL(p||q) = \int \log\left(\frac{p(x)}{q(x)}\right)p(x)dx = -\int \log\left(\frac{q(x)}{p(x)}\right)p(x)dx \tag{125}$$

$$\geq -\log\left(\int \frac{q(x)}{p(x)}p(x)dx\right) \geq 0. \tag{126}$$

where the inequality results from the Jensen's inequality, with equality if and only if the function is linear or the random variable is degenerated almost everywhere.

Here the function is $-\log$, thus equality occurs if there exists some constant α such that $p = \alpha q$. Since p and q are densities, it is equivalent to $p = q$.

JSD properties. The positivity of Jensen-Shannon divergence results from that of KL.

Besides, since $p, q \geq 0$,

$$JSD(p||q) = \frac{1}{2}KL\left(p||\frac{p+q}{2}\right) + \frac{1}{2}KL\left(q||\frac{p+q}{2}\right) \tag{127}$$

$$= \log 2 + \frac{1}{2}\int \log\left(\frac{p}{p+q}\right)p + +\frac{1}{2}\int \log\left(\frac{q}{p+q}\right)q \tag{128}$$

$$\leq \log 2. \tag{129}$$

Comparison of KL and JS divergences

$$KL(p||q) = \int \log\left(\frac{p(x)}{q(x)}\right) p(x) dx, \tag{130}$$

$$JSD(p||q) = \frac{1}{2} KL\left(p||\frac{p+q}{2}\right) + \frac{1}{2} KL\left(q||\frac{p+q}{2}\right). \tag{131}$$

KL is asymmetric. If $p(x)$ is very small but $q(x)$ is far from zero, the corresponding term of KL divergence is close to zero. This is not the case for Jensen-Shannon divergence which is symmetric.

MLE estimates can be seen as minimizers of KL divergence between the proposed distribution and the true distribution. This may be a reason explaining why GAN have good performance: they are based on JS divergence rather than on KL divergence.

Training GAN and related issues

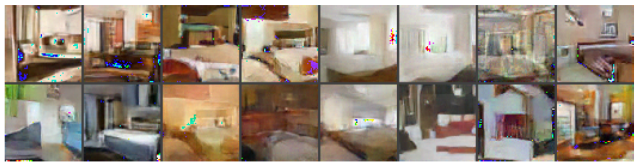
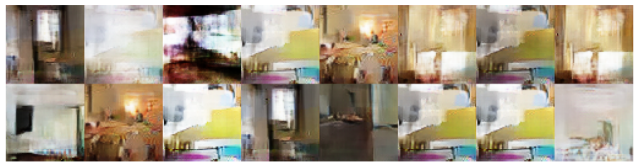
The training process for GAN is known to be slow and unstable.

Vanishing gradient.

- Consider a perfect discriminator, that is $D(x) = 1$ if $x \sim p^\star$ and $D(x) = 0$ if $x \sim p_G$. Thus, the loss of the discriminator is zero. Hence the gradient descent step for the generator is useless.
- On the other hand, if the discriminator is not good enough, the generator cannot learn to approximate the true data distribution.

Conclusion: the discriminator must be good but not too good... Really easy to implement!

Mode collapse
The generator outputs are very similar to each other and thus are not representative of the true distribution of data.

["Improved techniques for training gans", Salimans et al. 2016]

Improving training

How to address mode collapse?

- **Feature matching.** Instead of directly maximizing the output of the discriminator, the new objective requires the generator to generate data that matches the statistics of the real data, here the expected value of the features on an intermediate layer of the discriminator.
- **Minibatch discrimination.** Compute statistics over the whole minibatch and allow the discriminator to use that information to make an estimation.
- **Historical averaging.** Modify the cost of each player to include $\|\theta - \frac{1}{t}\sum_{i=1}^{t}\theta^{(i)}\|_2^2$.
- **One-sided label smoothing.** Replace the labels 1 by $\alpha \in (0, 1)$ (typically, $\alpha = 0.9$).
- **Virtual batch normalization.**

Difference between VAE and GAN

Both are based on backpropagation/gradient descent.

- Gradient flow in GAN requires that the data output by the generator are continuous. Therefore GAN cannot model discrete data.
- Gradient flow in VAE requires that the data output by the encoder are continuous. Therefore latent variables in VAE cannot be discrete.

More references:

["Towards principled methods for training generative adversarial networks", Arjovsky and Bottou 2017]

["Approximation and convergence properties of generative adversarial learning", Liu et al. 2017]

["On the discrimination-generalization tradeoff in GANs", Zhang et al. 2017]

["Improved training of wasserstein gans", Gulrajani et al. 2017]

`https://poloclub.github.io/ganlab/`

Outline

Create animate characters

["Towards the Automatic Anime Characters Creation with Generative Adversarial Networks", Jin et al. 2017]

Figure: Original images

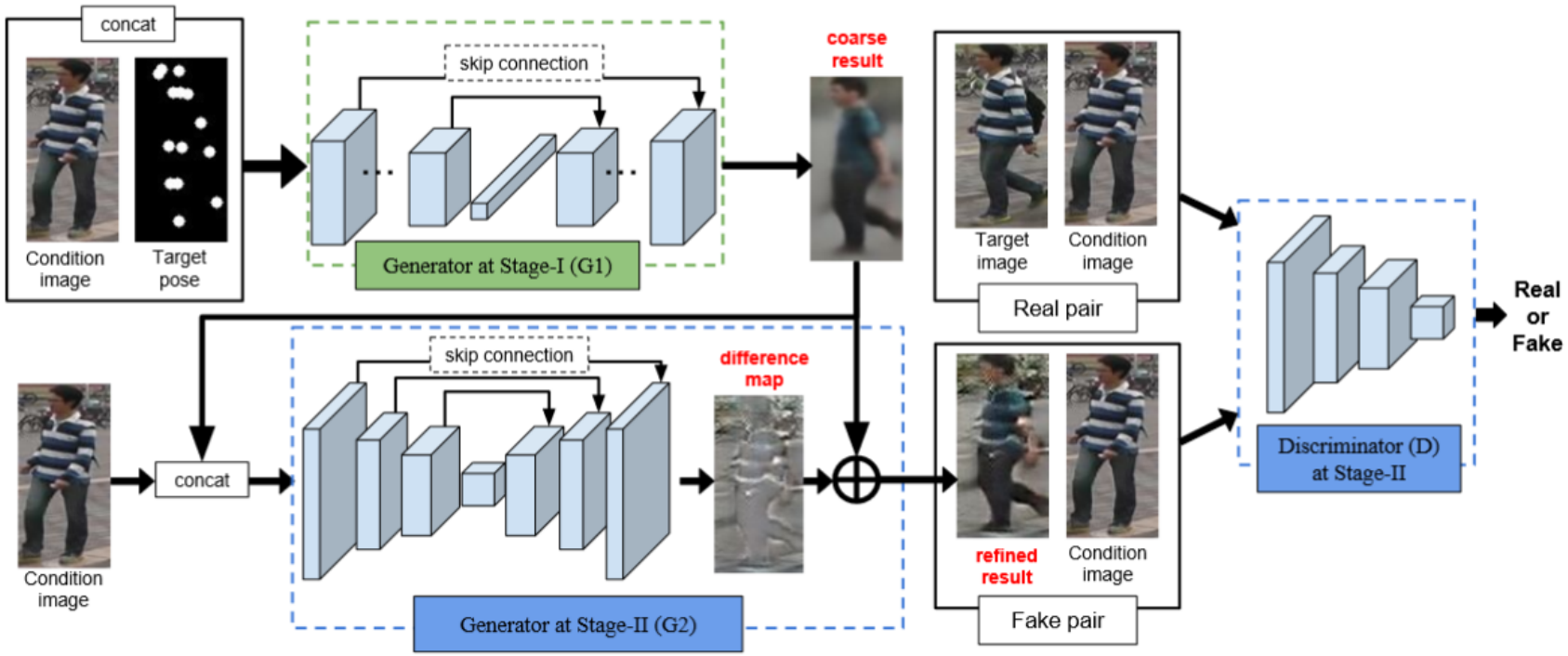
Figure: Generated images

Particularity: a vector of 34 attributes (describing hair, eyes... learned with a CNN) is given as random input for the generator, together with random noise $\rightarrow$ allow for generating specific type of characters.
The loss of the discriminator is also modified to take into account the 34 classes.

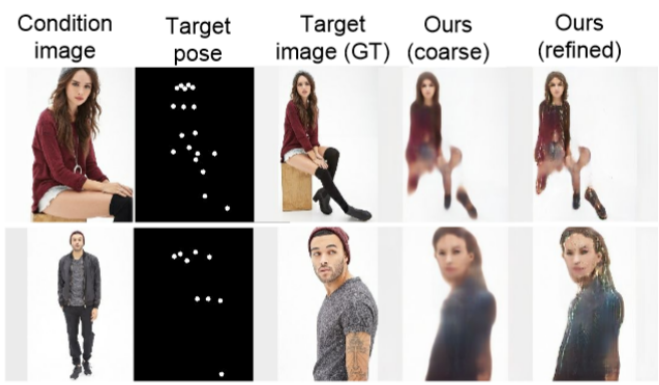
`https://make.girls.moe/`

Generating poses

["Pose guided person image generation", Ma et al. 2017]

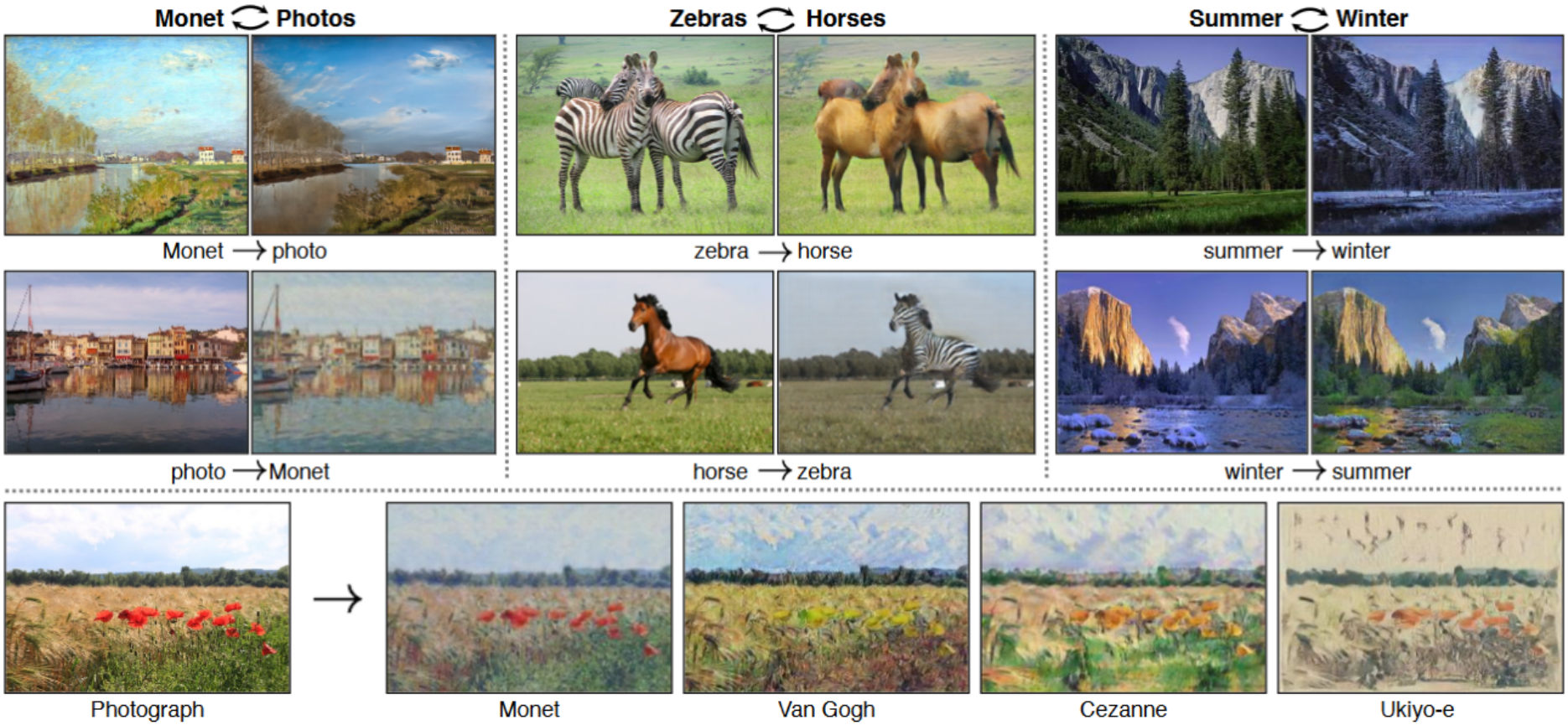


Generating poses



Domain transfer

["Unpaired image-to-image translation using cycle-consistent adversarial networks", Zhu et al. 2017]



Easy if you have a training set composed of pairs (X_i, Y_i)

Domain transfer

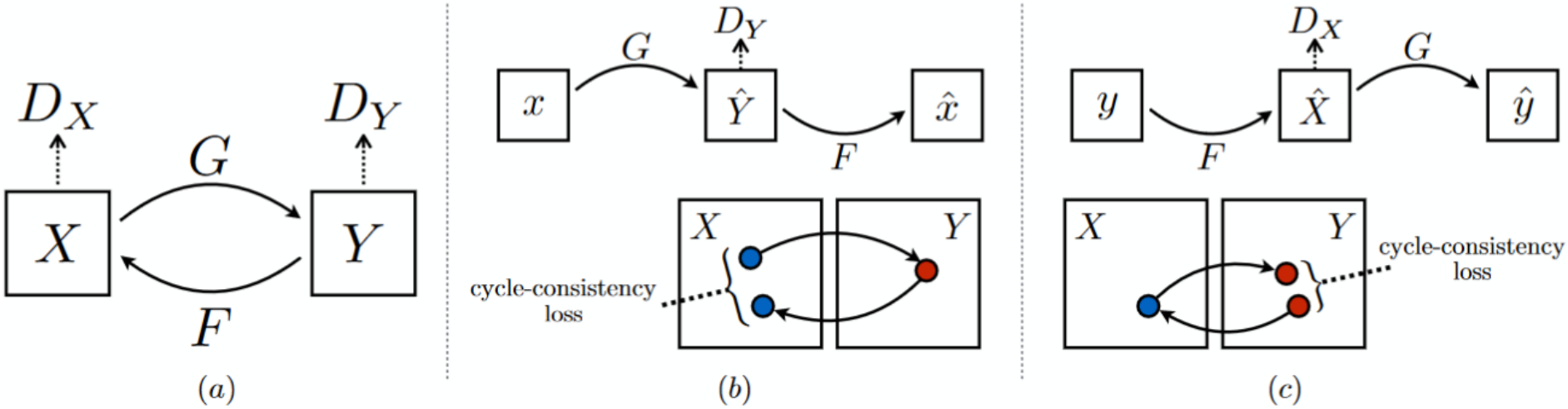
We have two domains X and Y. We want to learn a mapping $G : X \rightarrow Y$ that produces $\hat{y} = G(x)$ that an adversarial network cannot distinguish from true samples $\{Y_1, \ldots, Y_n\}$.

Problem: no guarantee to recover the corresponding image of x. Besides, there are optimization issues.

Solution: impose that the mapping should be invertible and train simultaneously $G : X \rightarrow Y$ and $F : Y \rightarrow X$ such that

- $F(G(x)) \simeq x$
- $G(F(y)) \simeq y$
- the distribution of $F(x)$ is close to that of Y.

Domain transfer



Loss:

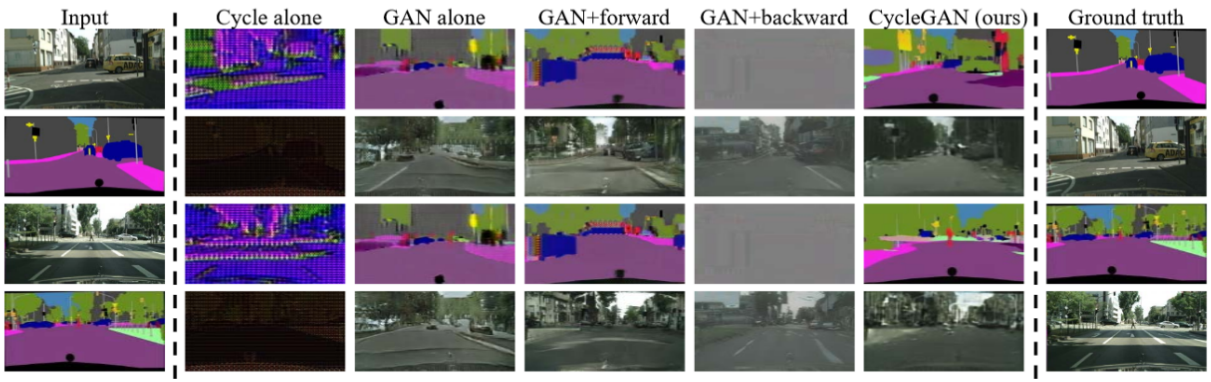
$$\mathcal{L} = \mathcal{L}_{GAN}(G, D_Y, X, Y) + \mathcal{L}_{GAN}(F, D_X, Y, X) + \lambda \mathcal{L}_{cycle}(G, F),$$

where $\mathcal{L}_{GAN}$ is the standard adversarial loss and

$$\mathcal{L}_{cycle}(G, F) = \mathbb{E}_{x \sim p_X^\star} \|F(G(x)) - x\|_1 + \mathbb{E}_{y \sim p_Y^\star} \|G(F(y)) - y\|_1.$$

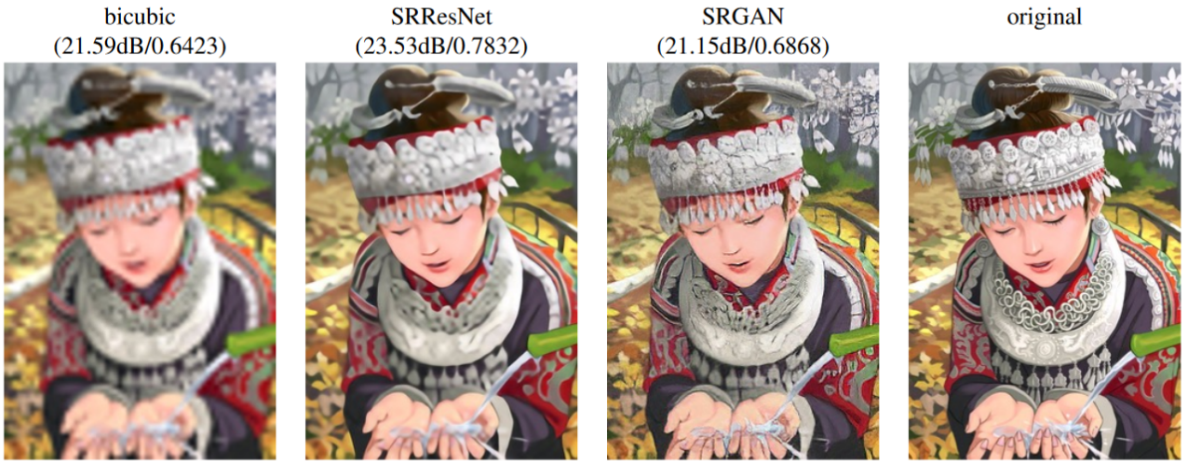
Domain transfer

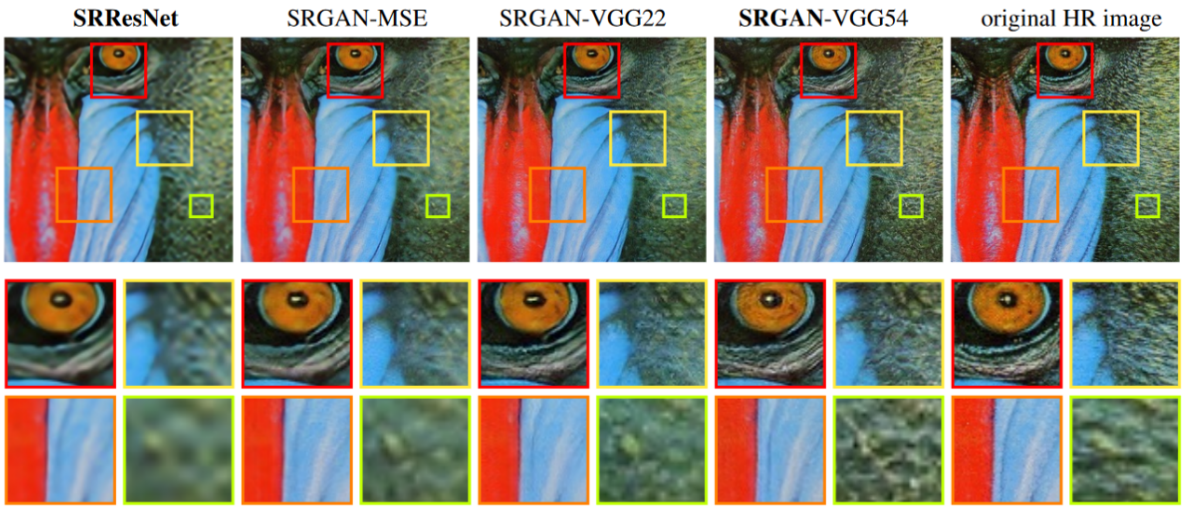
Figure: Paired data set - close to performance of supervised learner pix2pix



Create high-resolution pictures

["Photo-realistic single image super-resolution using a generative adversarial network", Ledig et al. 2017]





Generating celebrities

["Progressive growing of gans for improved quality, stability, and variation", Karras et al. 2017]

Figure: Last five rows: five closest pictures of the generated one (first row) in the data set

`https://www.youtube.com/watch?v=XOxxPcy5Gr4`

Natural Language Processing (NLP)

For statistical language modeling, the input typically consists of incomplete sequences of words rather than complete sentences.

Translation

["Sequence to sequence learning with neural networks", Sutskever et al. 2014]

["Neural machine translation in linear time", Kalchbrenner et al. 2016]

Recently, Kalchbrenner et al. 2016 propose a CNN-based architecture for sequence processing called ByteNet, which is a stack of two dilated CNNs. Like WaveNet, ByteNet also benefits from convolutions with dilations to increase the receptive field size, thus can model sequential data with long-term dependencies. It also has the advantage that the computational time only linearly depends on the length of the sequences. Compared with recurrent neural networks, CNNs not only can get long-range information but also get a hierarchical representation of the input words

Inspired by the gating mechanism in LSTM net- works, the gated CNN in Dauphin et al. 2016 uses a gating mechanism to control the path through which information flows in the network, and achieves the state-of-the-art on WiKiText-103.

However, this framework is still under the recurrent framework, and the input window size of their network are of limited size. How to capture the specific long-term dependencies as well as hierarchical representation of history words is still an open problem.

References

Martin Arjovsky and Léon Bottou. "Towards principled methods for training generative adversarial networks". In: *arXiv preprint arXiv:1701.04862* (2017).

Yoshua Bengio and Samy Bengio. "Modeling high-dimensional discrete data with multi-layer neural networks". In: *Advances in Neural Information Processing Systems*. 2000, pp. 400–406.

Julius R Blum, J Kiefer, and Murray Rosenblatt. "Distribution free tests of independence based on the sample distribution function". In: *The annals of mathematical statistics* (1961), pp. 485–498.

Yann N Dauphin et al. "Language modeling with gated convolutional networks". In: *arXiv preprint arXiv:1612.08083* (2016).

Luc Devroye. "Sample-based non-uniform random variate generation". In: *Proceedings of the 18th conference on Winter simulation*. ACM. 1986, pp. 260–265.

Brendan J Frey and Brendan J Frey. *Graphical models for machine learning and digital communication*. MIT press, 1998.

References

Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

Ishaan Gulrajani et al. "Improved training of wasserstein gans". In: *Advances in Neural Information Processing Systems*. 2017, pp. 5767–5777.

Yanghua Jin et al. "Towards the Automatic Anime Characters Creation with Generative Adversarial Networks". In: *arXiv preprint arXiv:1708.05509* (2017).

Nal Kalchbrenner et al. "Neural machine translation in linear time". In: *arXiv preprint arXiv:1610.10099* (2016).

Tero Karras et al. "Progressive growing of gans for improved quality, stability, and variation". In: *arXiv preprint arXiv:1710.10196* (2017).

Tejas D Kulkarni et al. "Deep convolutional inverse graphics network". In: *Advances in neural information processing systems*. 2015, pp. 2539–2547.

Shuang Liu, Olivier Bousquet, and Kamalika Chaudhuri. "Approximation and convergence properties of generative adversarial learning". In: *Advances in Neural Information Processing Systems*. 2017, pp. 5545–5553.

References

Christian Ledig et al. "Photo-realistic single image super-resolution using a generative adversarial network". In: *arXiv preprint* (2017).

Liqian Ma et al. "Pose guided person image generation". In: *Advances in Neural Information Processing Systems*. 2017, pp. 406–416.

Tim Salimans et al. "Improved techniques for training gans". In: *Advances in Neural Information Processing Systems*. 2016, pp. 2234–2242.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to sequence learning with neural networks". In: *Advances in neural information processing systems*. 2014, pp. 3104–3112.

Pengchuan Zhang et al. "On the discrimination-generalization tradeoff in GANs". In: *arXiv preprint arXiv:1711.02771* (2017).

Jun-Yan Zhu et al. "Unpaired image-to-image translation using cycle-consistent adversarial networks". In: *arXiv preprint* (2017).