Generative Modelling with Neural Networks

M2DS Alternants Research Seminar Course 2022

Outline

Outline

Image Generation

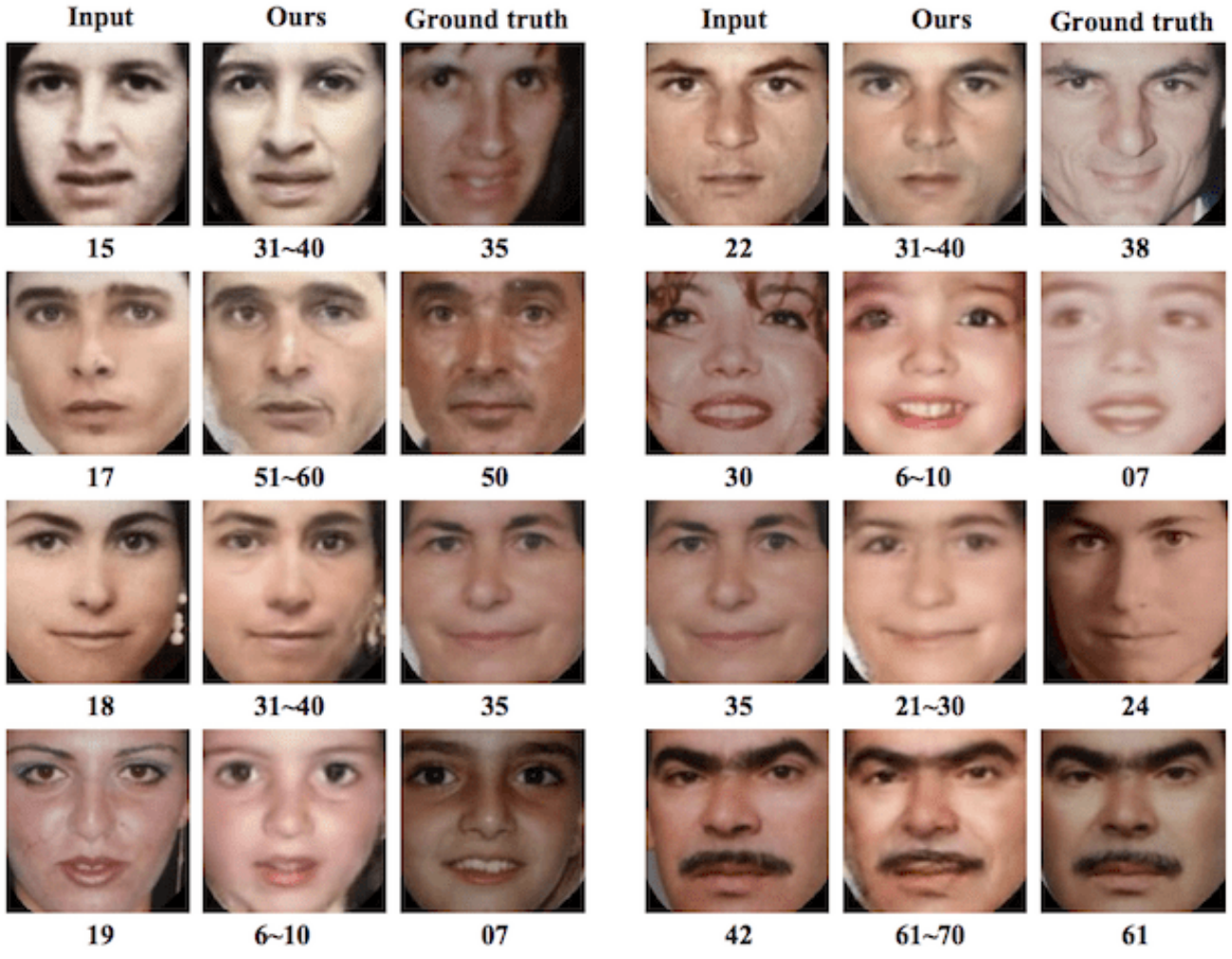
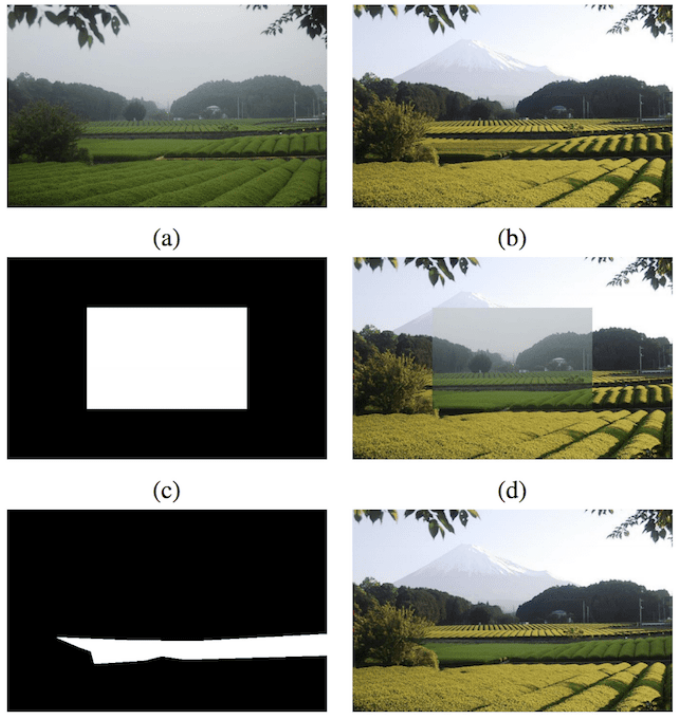
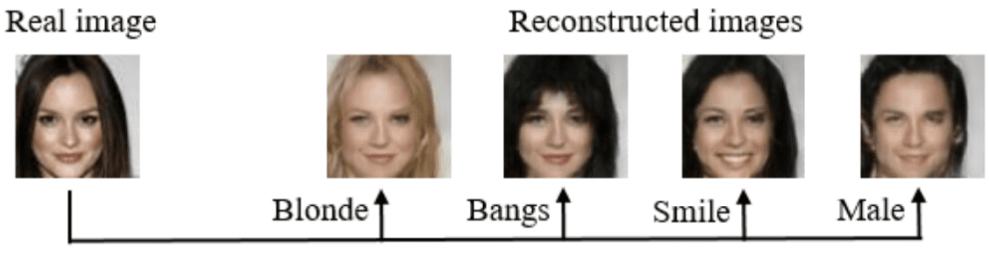


Photo Editting





Text to Image Synthesis

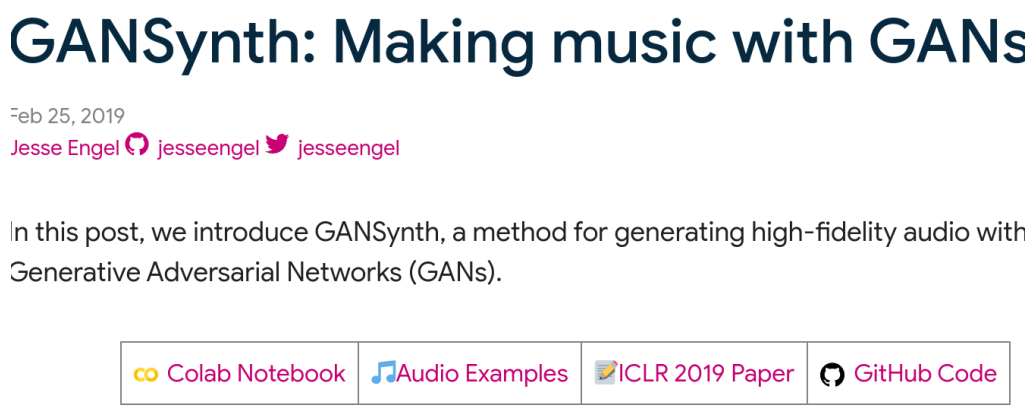
this small bird has a pink breast and crown, and black primaries and secondaries.

this magnificent fellow is almost all black with a red crest, and white cheek patch.

the flower has petals that are bright pinkish purple with white stigma

this white and yellow flower have thin white petals and a round yellow stamen

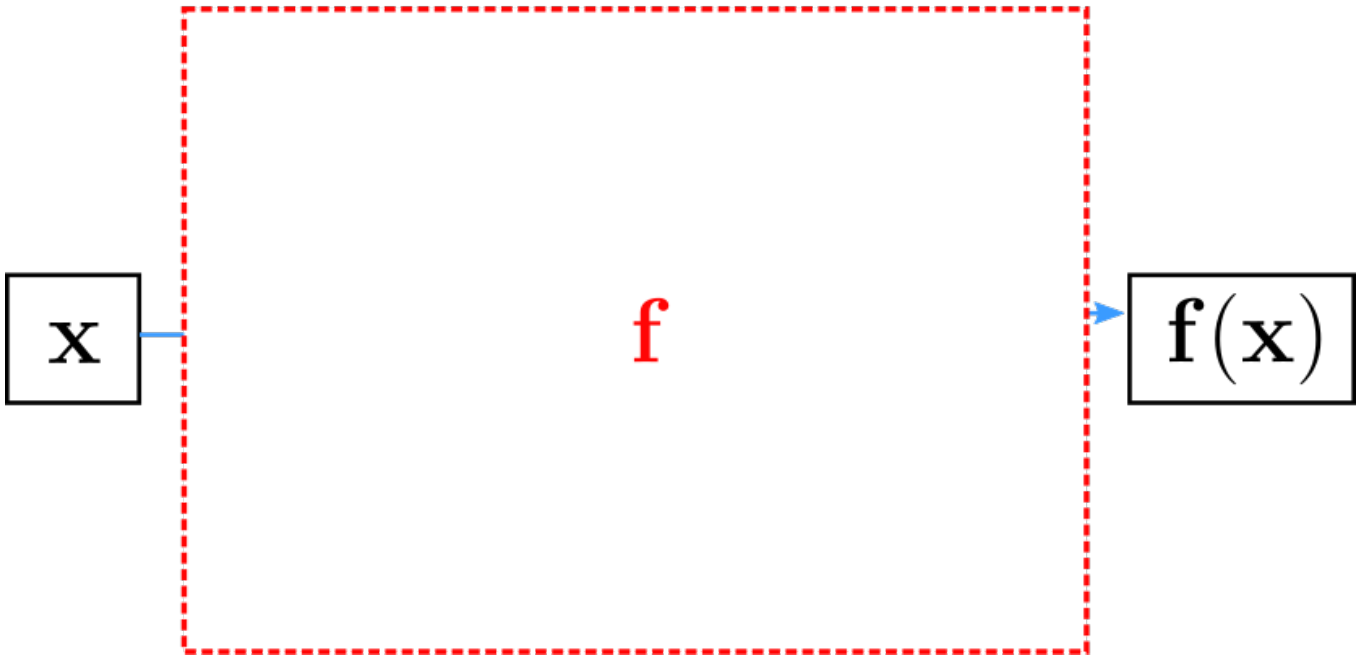
Making Music



https://magenta.tensorflow.org/gansynth

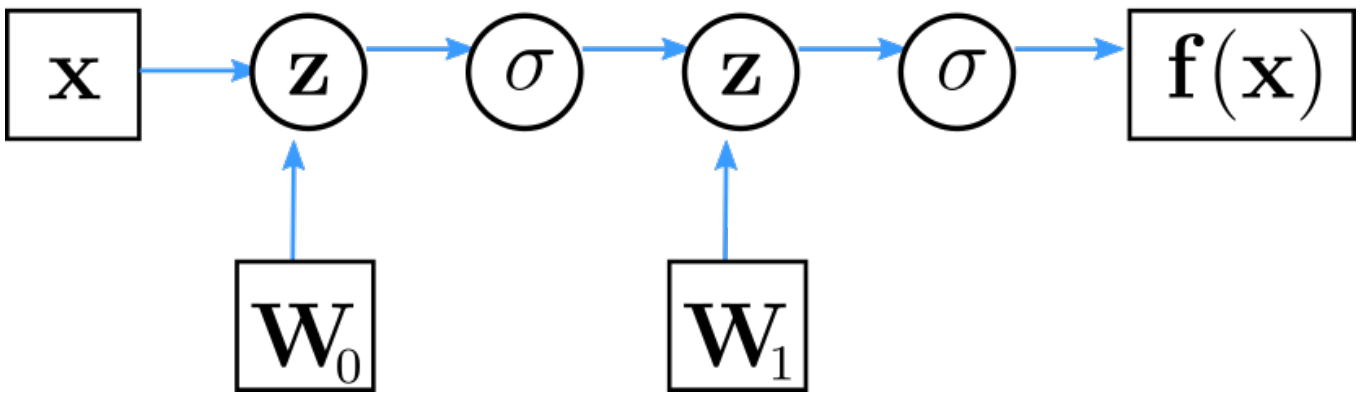
Outline

Computation Graph



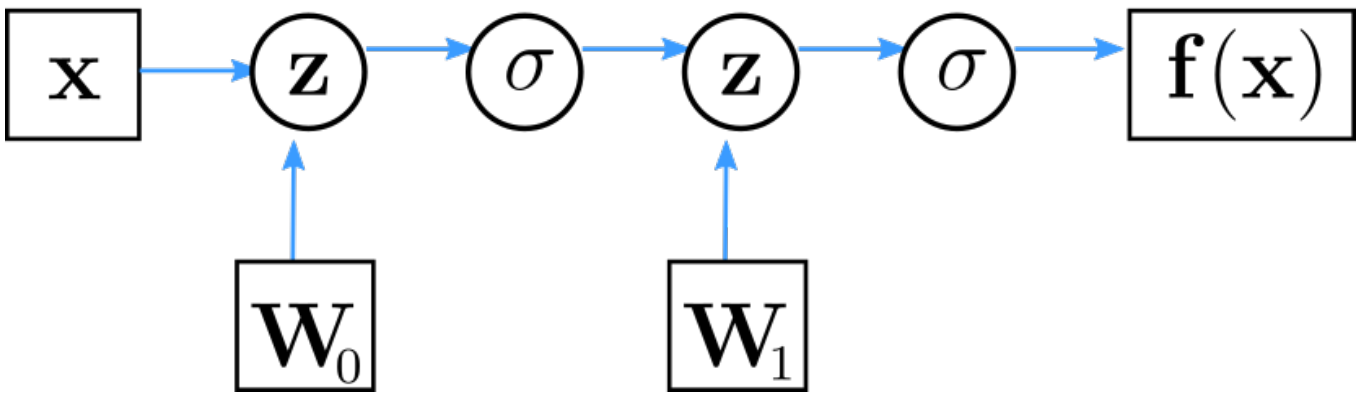
Neural network = parametrized, non-linear function

Computation Graph



Computation graph: Directed graph of functions, depending on parameters (neuron weights)

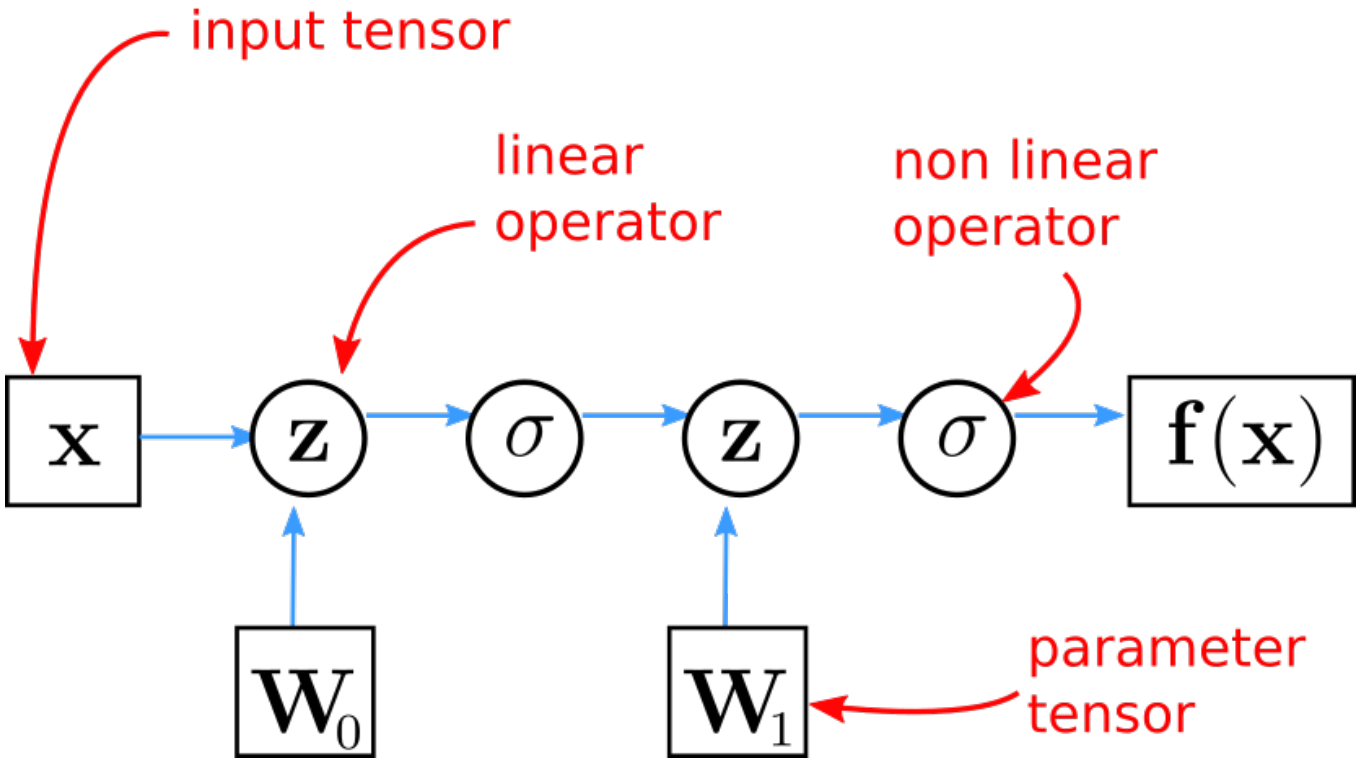
Computation Graph



Computation graph: Directed graph of functions, depending on parameters (neuron weights)

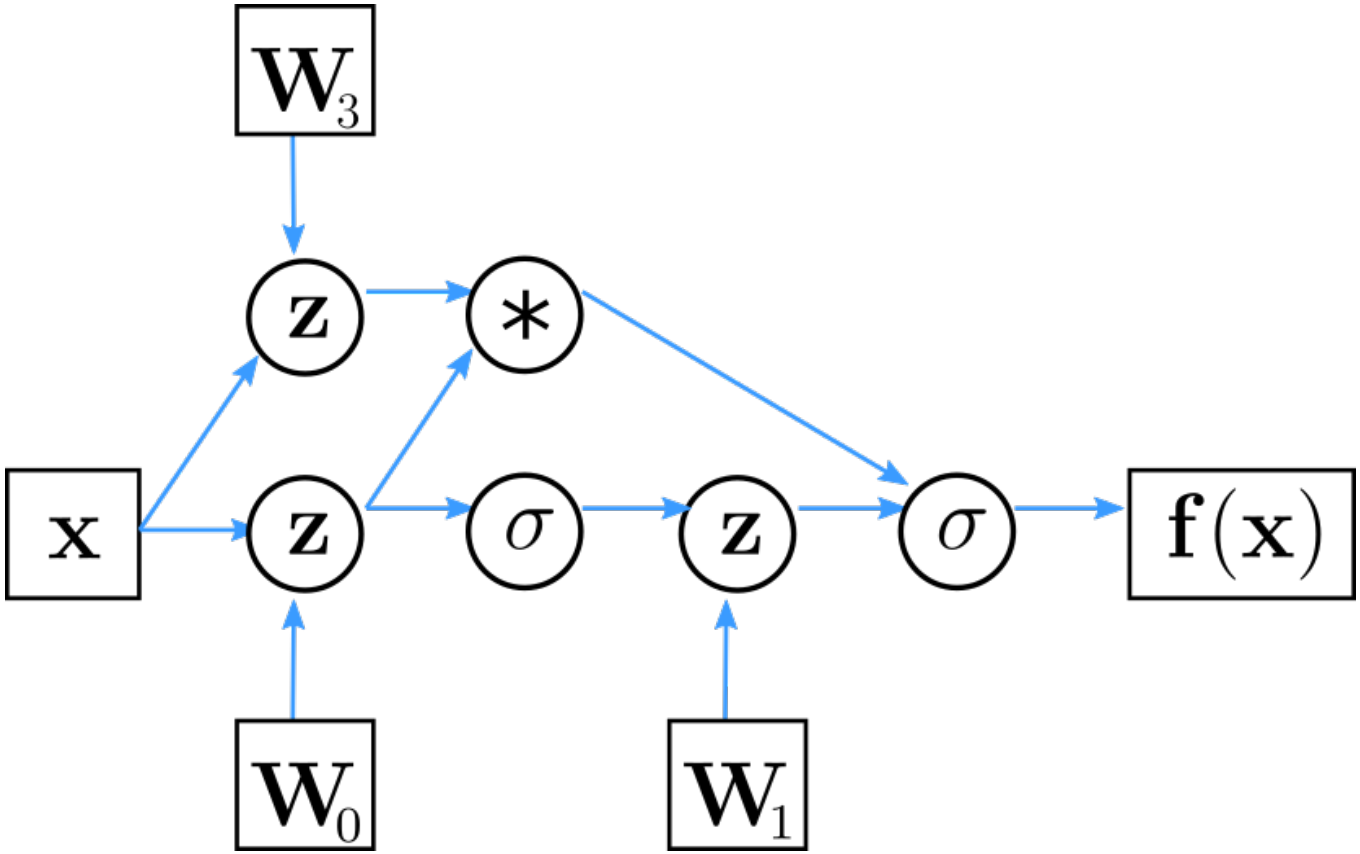
Question: can you write the full expression of $f(x)$ for this graph?

Computation Graph



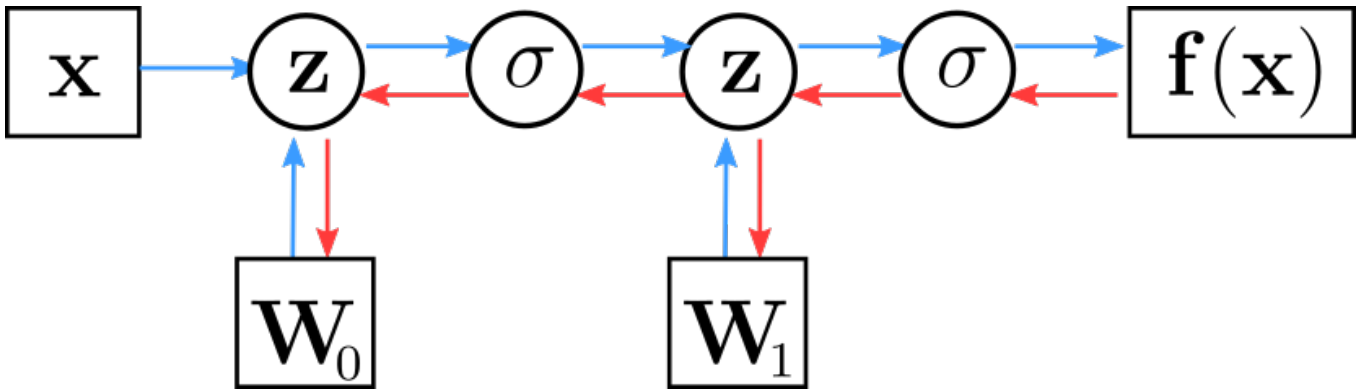
Combination of linear (parametrized) and non-linear functions

Computation Graph



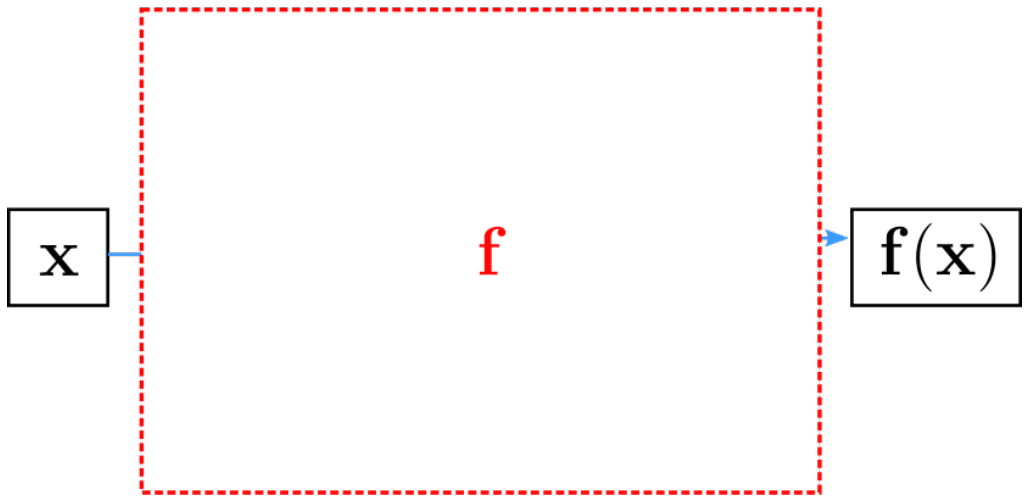
Not only sequential application of functions

Computation Graph



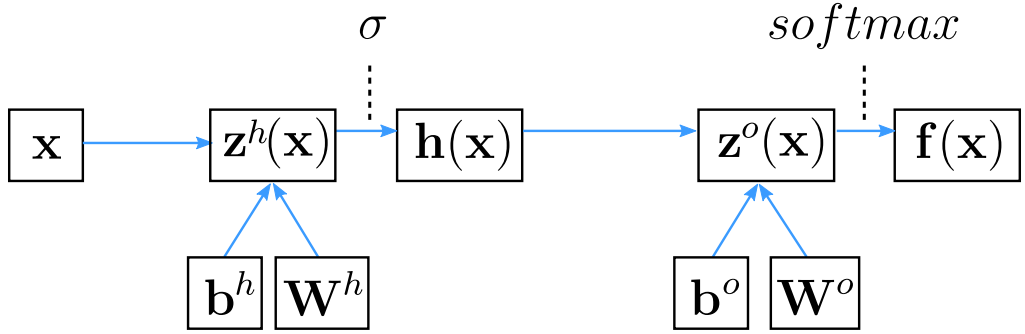
- Automatic computation of gradients: all modules are *differentiable*!
- Vector computation on CPU and accelerators (GPU and TPU).

Formalization for Classification Problem



- Input: $\{\mathrm{x}, y\}_{i=1}^{n}$ with $y \in \{0, 1\}$
- $f(\cdot, \theta) : \mathbb{R}^p \to (0, 1)$
- Output: $f(\mathrm{x}_i, \theta) = \mathbb{P}(y_i = 1 \mid \mathrm{x} = \mathrm{x}_i)$

Simple example: One-hidden layer network



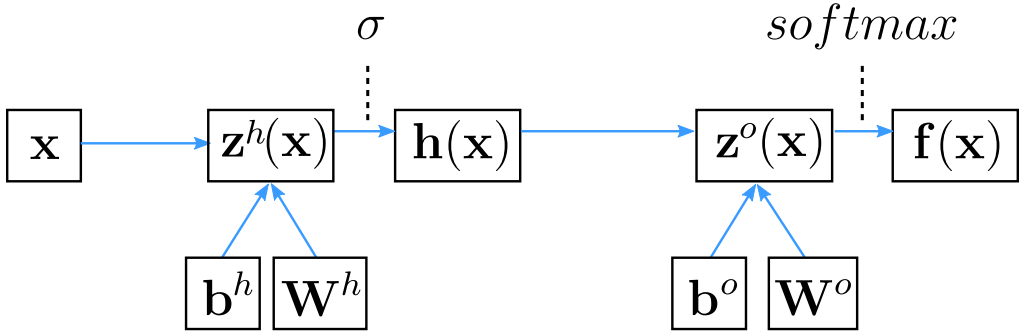
$$f(\mathrm{x}) = \sigma(\mathrm{w}^T\mathrm{x} + b)$$

- $\mathrm{z}^h(\mathrm{x}) = \mathrm{W}^h\mathrm{x} + \mathrm{b}^h$
- $\mathrm{h}(\mathrm{x}) = g(\mathrm{z}^h(\mathrm{x})) = g(\mathrm{W}^h\mathrm{x} + \mathrm{b}^h)$
- $\mathrm{z}^o(\mathrm{x}) = \mathrm{W}^o\mathrm{h}(\mathrm{x}) + \mathrm{b}^o$
- $\mathrm{f}(\mathrm{x}) = softmax(\mathrm{z}^o) = softmax(\mathrm{W}^o\mathrm{h}(\mathrm{x}) + \mathrm{b}^o)$

Where:

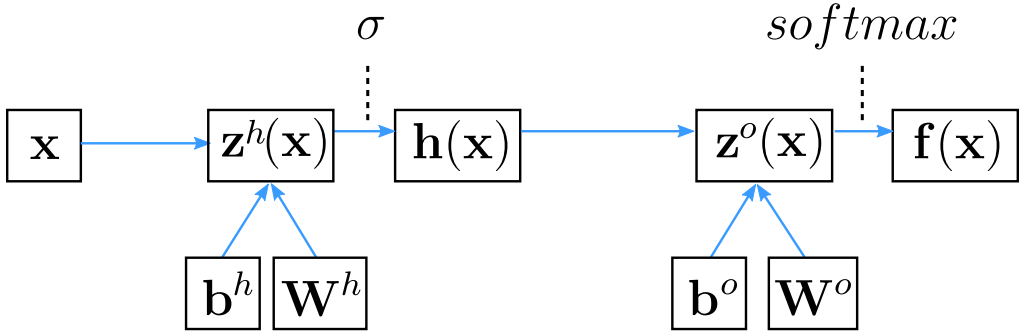
- ($z(\cdot)$ pre-activation)
- w, b weights and bias
- σ activation function

Simple example: One-hidden layer network



Question: What is this `softmax` thing doing here?

Simple example: One-hidden layer network



Question: What is this `softmax` thing doing here?

$$softmax(\mathrm{x}) = \frac{1}{\sum_{i=1}^{n} e^{x_i}} \cdot \begin{bmatrix} e^{x_1} \\ e^{x_2} \\ \vdots \\ e^{x_n} \end{bmatrix}$$

$$\frac{\partial softmax(\mathrm{x})_i}{\partial x_j} = \begin{cases} softmax(\mathrm{x})_i \cdot (1 - softmax(\mathrm{x})_i) & i = j \\ -softmax(\mathrm{x})_i \cdot softmax(\mathrm{x})_j & i \neq j \end{cases}$$

Training the network

Objective: Finding parameters $\theta = (\mathrm{W}^h; \mathrm{b}^h; \mathrm{W}^o; \mathrm{b}^o)$ that minimize the *negative log likelihood*

The loss function for a given sample $s \in S$:

$$\ell(\mathrm{f}(\mathrm{x}^s; \theta), y^s) = -\log \mathrm{f}(\mathrm{x}^s; \theta)_{y^s}$$

Training the network

Objective: Finding parameters $\theta = (\mathrm{W}^h; \mathrm{b}^h; \mathrm{W}^o; \mathrm{b}^o)$ that minimize the *negative log likelihood*

The loss function for a given sample $s \in S$:

$$\ell(\mathrm{f}(\mathrm{x}^s; \theta), y^s) = -\log \mathrm{f}(\mathrm{x}^s; \theta)_{y^s}$$

$\rightarrow$ when computed on full training set:

$$L(\theta) = -\frac{1}{n} \sum_{i=1}^{n} \log \mathrm{f}(\mathrm{x}^s; \theta)_{y^s}$$

or even with ℓ_2-regularization term

$$L(\theta) = -\frac{1}{n} \sum_{i=1}^{n} \log \mathrm{f}(\mathrm{x}^s; \theta)_{y^s} + \lambda(\| W^h \|^2 + \| W^o \|^2)$$

Training the network

Objective: Finding parameters $\theta = (\mathrm{W}^h; \mathrm{b}^h; \mathrm{W}^o; \mathrm{b}^o)$ that minimize the *negative log likelihood*
The loss function for a given sample $s \in S$:

$$\ell(\mathrm{f}(\mathrm{x}^s; \theta), y^s) = -\log \mathrm{f}(\mathrm{x}^s; \theta)_{y^s}$$

$\rightarrow$ when computed on full training set:

$$L(\theta) = -\frac{1}{n} \sum_{i=1}^{n} \log \mathrm{f}(\mathrm{x}^s; \theta)_{y^s}$$

or even with ℓ_2-regularization term

$$L(\theta) = -\frac{1}{n} \sum_{i=1}^{n} \log \mathrm{f}(\mathrm{x}^s; \theta)_{y^s} + \lambda(\| W^h \|^2 + \| W^o \|^2)$$

Question (reminder): how to find θ?

Gradient Descent

Gradient descent:

$$\theta^{(k+1)} = \theta^{(k)} - \eta \nabla_\theta L(\theta^{(k)})$$

Question: what is the potential problem with this?

Gradient Descent

Gradient descent:

$$\theta^{(k+1)} = \theta^{(k)} - \eta \nabla_\theta L(\theta^{(k)})$$

Question: what is the potential problem with this?
$\longrightarrow$ problematic with large n: taking long time to train the network

Stochastic Gradient Descent

1. Initialize θ randomly
2. For K epochs perform:
   - Important: randomly select a small batch of samples ($B \subset [n]$)
   - Do gradient descent with only this batch of samples:

$$\theta^{(k+1)} = \theta^{(k)} - \eta \nabla_\theta L_B(\theta^{(k)})$$

3. Stop when reaching criterion:
   $L(\theta^{(k)})$ stops decreasing when computed on validation set

Backpropagation: Computing Gradients Efficiently

Output Weights:

$$\frac{\partial \ell(\mathrm{f}(\mathrm{x}), y)}{\partial W^o_{i,j}}$$

Hidden Weights:

$$\frac{\partial \ell(\mathrm{f}(\mathrm{x}), y)}{\partial W^h_{i,j}}$$

Output bias:

$$\frac{\partial \ell(\mathrm{f}(\mathrm{x}), y)}{\partial b^o_i}$$

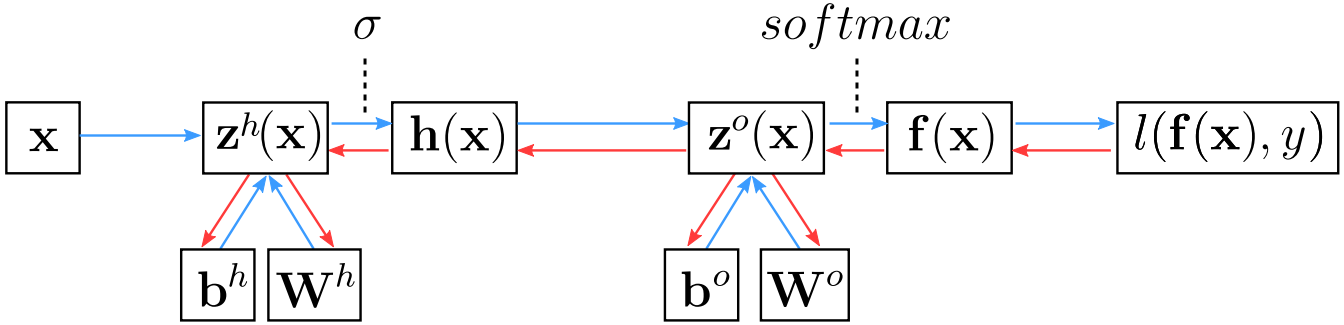
Hidden bias:

$$\frac{\partial \ell(\mathrm{f}(\mathrm{x}), y)}{\partial b^h_i}$$

$\longrightarrow$ differentiate with *chain-rule* – with $z(x) = u(v(x))$:

$$\frac{\partial z}{\partial x} = \left[\frac{\partial u}{\partial v}\right]^\top \frac{\partial v}{\partial x}$$

Backpropagation: For one hidden-layer network



Compute activation gradients

$$\nabla_{\mathbf{z}^o(\mathbf{x})}\ell = \mathbf{f}(\mathbf{x}) - \text{one-hot-encode}(y)$$

Compute layer params gradients

$$\nabla_{\mathbf{W}^o}\ell = \nabla_{\mathbf{z}^o(\mathbf{x})}\ell \cdot \mathbf{h}(\mathbf{x})^\top$$

$$\nabla_{\mathbf{b}^o}\ell = \nabla_{\mathbf{z}^o(\mathbf{x})}\ell$$

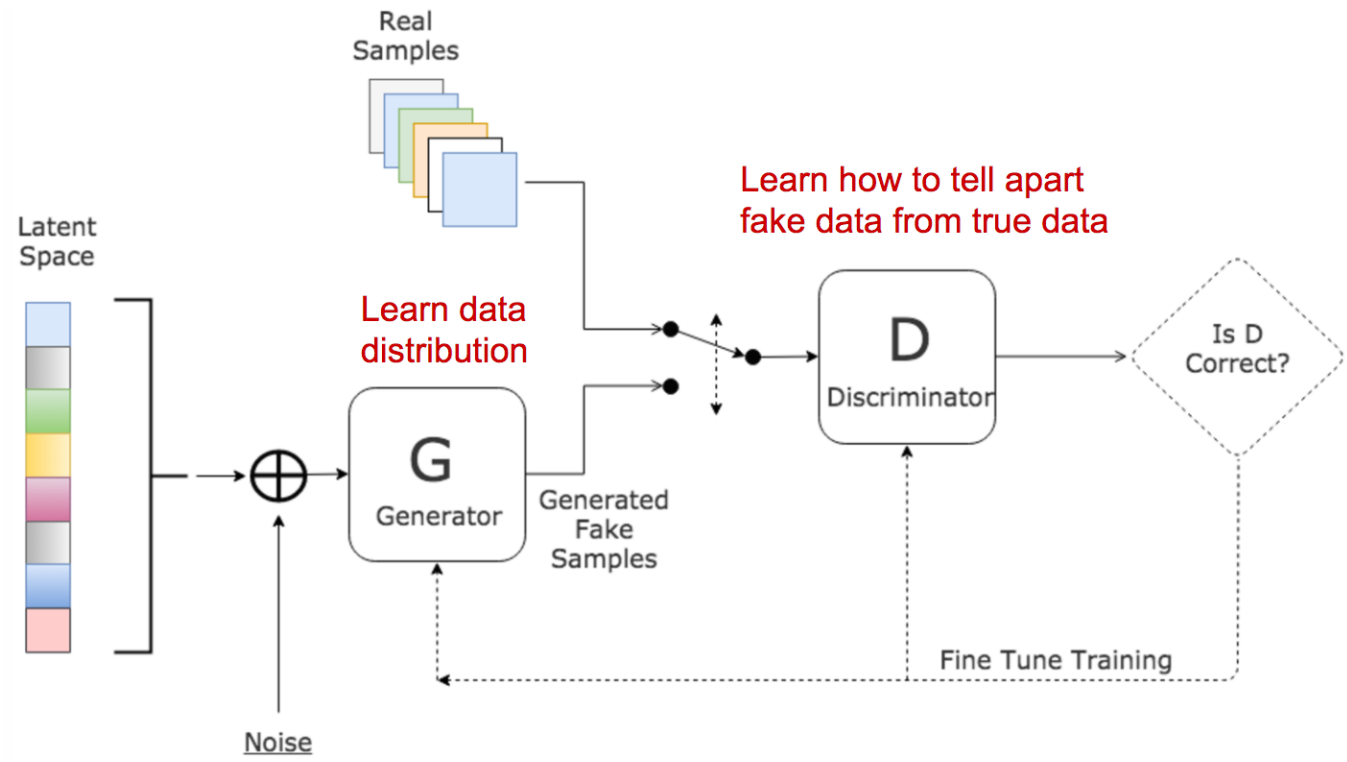
Compute prev layer activation gradients

$$\nabla_{\mathbf{h}(\mathbf{x})}\ell = \mathbf{W}^{o\top}\nabla_{\mathbf{z}^o(\mathbf{x})}\ell$$

$$\nabla_{\mathbf{z}^h(\mathbf{x})}\ell = \nabla_{\mathbf{h}(\mathbf{x})}\ell \odot \sigma'(\mathbf{z}^{\mathbf{h}}(\mathbf{x}))$$

Outline

Generative Adversarial Network (GAN)



Alternate training between:

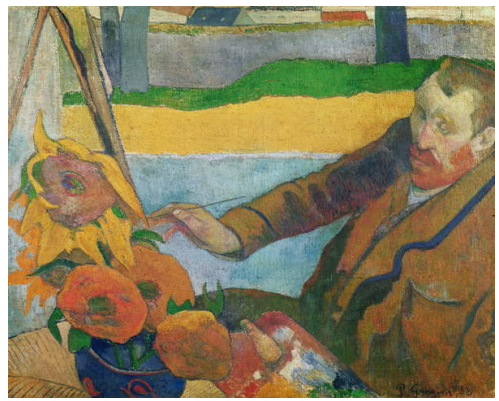
- A *generative network* G (generator), and
- A *discrimininative network* D (discriminator)

Goodfellow, Ian, et al. Generative Adversarial Nets. NeurIPS 2014

What does these two networks do?

(intuitively): Discriminator

- Estimates the probability of a given sample coming from the real dataset
- Optimized to tell the fake samples from the real ones



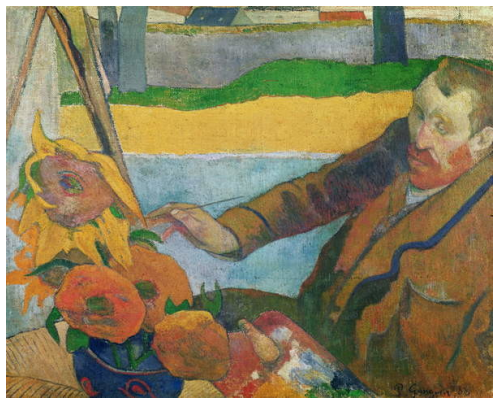
What does these two networks do?

(intuitively): Discriminator

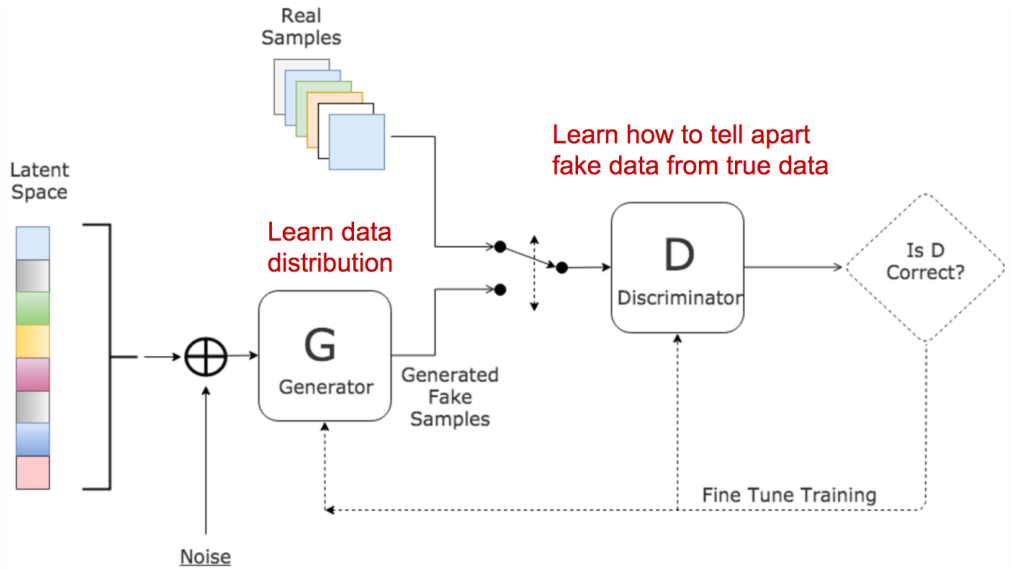
- Estimates the probability of a given sample coming from the real dataset
- Optimized to tell the fake samples from the real ones

Generator

- outputs synthetic samples given a noise variable input (brings in potential output diversity)
- trained to capture the real data distribution so to generate samples can be as real as possible



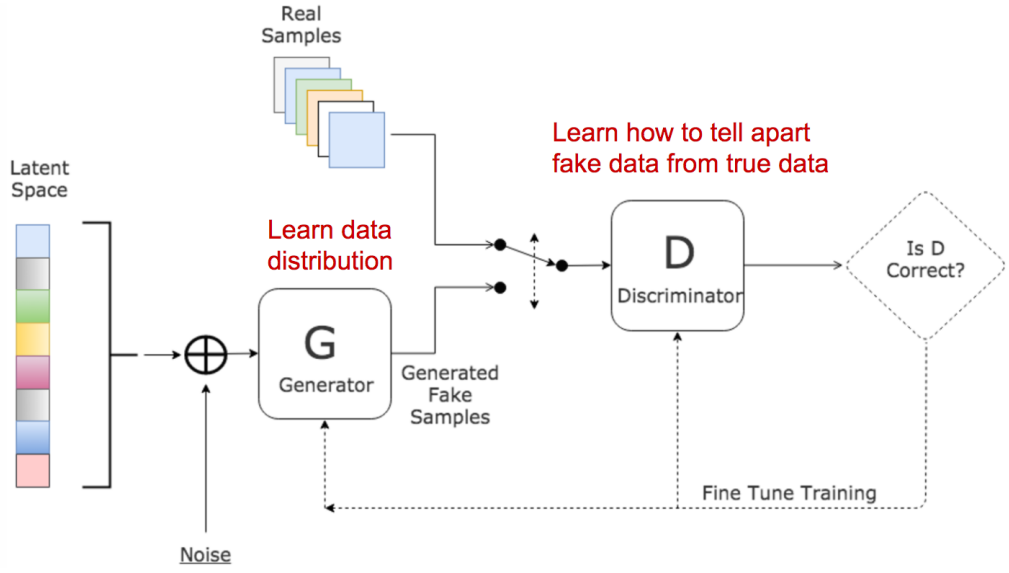
What does these two networks do?



More formally:

- p_z: Noise distribution
- p_g: Generator's distribution over data x
- p_r: Real data x's distribution

What does these two networks do?



Discriminator:

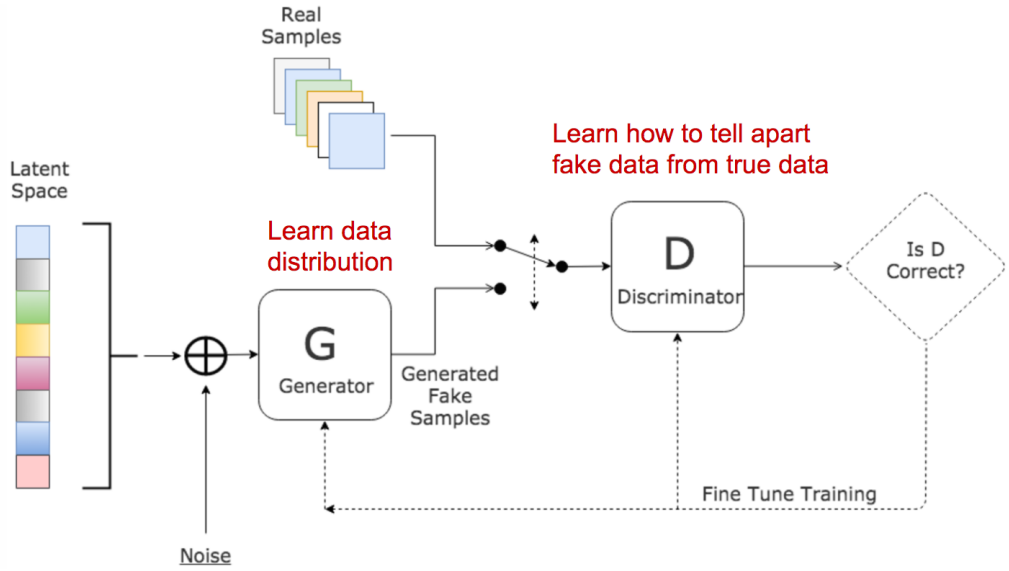
- Make sure decisions over real data are accurate by

$$\max \mathbb{E}_{x \sim p_r(x)}[\log D(x)]$$

- At the same time: given a fake sample $G(z)$ with $z \sim p_z$

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

What does these two networks do?

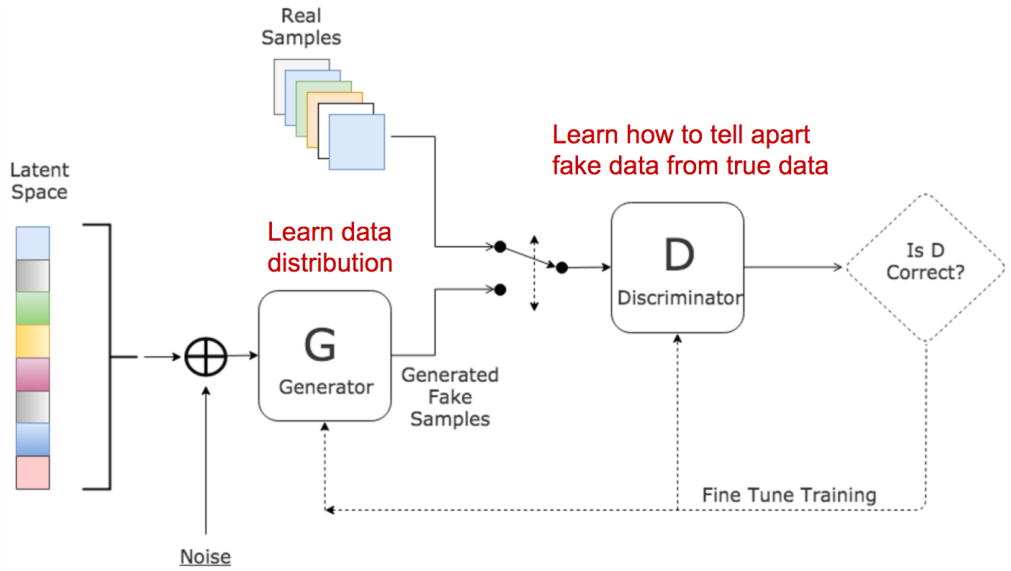


Generator:

- Try its best to fool the discriminator:

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

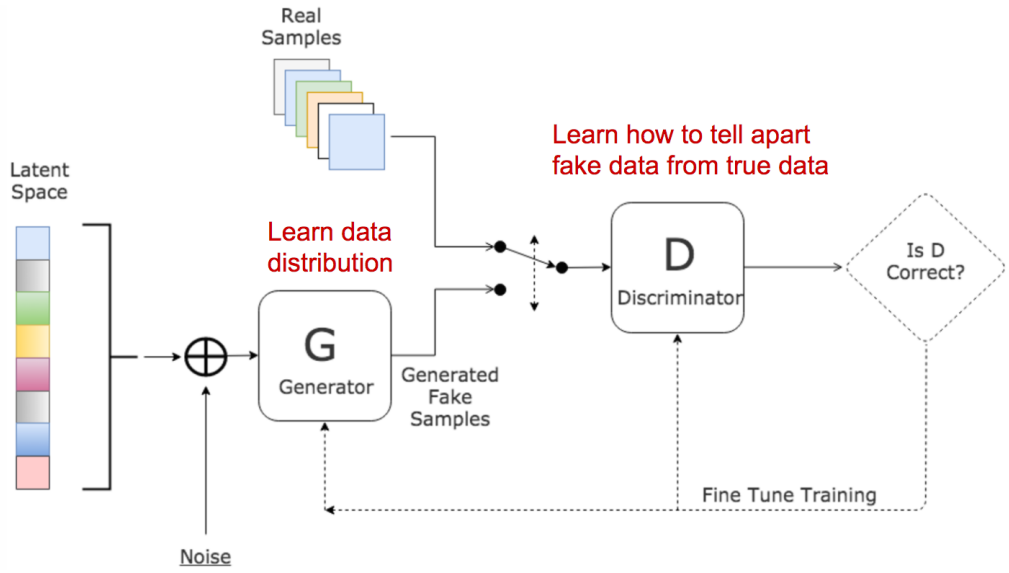
What does these two networks do?



Putting together:

$$L(D,G) = \mathbb{E}_{x\sim p_r(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(x)}[\log(1 - D(G(z)))]$$
$$= \mathbb{E}_{x\sim p_r(x)}[\log D(x)] + \mathbb{E}_{x\sim p_g(x)}[\log(1 - D(x))]$$

What does these two networks do?



Putting together: playing a *minimax* (zero-sum) game

$$\min_G \max_D L(D, G) = \mathbb{E}_{x \sim p_r(x)}[\log D(x)] + \mathbb{E}_{x \sim p_g(x)}[\log(1 - D(x))]$$

Let's examine the loss function

$$L(D, G) = \mathbb{E}_{x \sim p_r(x)}[\log D(x)] + \mathbb{E}_{x \sim p_g(x)}[\log(1 - D(x))]$$

Let's examine the loss function

$$
\begin{aligned}
L(D, G) &= \mathbb{E}_{x \sim p_r(x)}[\log D(x)] + \mathbb{E}_{x \sim p_g(x)}[\log(1 - D(x))] \\
&= \int_x \left(p_r(x) \log D(x) + p_g(x) \log(1 - D(x)) \right) dx
\end{aligned}
$$

Let's examine the loss function

$$L(D, G) = \mathbb{E}_{x \sim p_r(x)}[\log D(x)] + \mathbb{E}_{x \sim p_g(x)}[\log(1 - D(x))]$$
$$= \int_x \left(p_r(x) \log D(x) + p_g(x) \log(1 - D(x))\right) dx$$

Since we are summing all over sample x, what we should pay attention to is

$$\min_G \max_D f(x) \stackrel{\text{def.}}{=} p_r(x) \log D(x) + p_g(x) \log(1 - D(x))$$

Question: finding optimal D is equivalent with?

Let's examine the loss function

$$L(D, G) = \mathbb{E}_{x \sim p_r(x)}[\log D(x)] + \mathbb{E}_{x \sim p_g(x)}[\log(1 - D(x))]$$
$$= \int_x \left(p_r(x) \log D(x) + p_g(x) \log(1 - D(x))\right) dx$$

Since we are summing all over sample x, what we should pay attention to is

$$\min_G \max_D f(x) \stackrel{\text{def.}}{=} p_r(x) \log D(x) + p_g(x) \log(1 - D(x))$$

Question: finding optimal D is equivalent with?

$$\frac{\partial f(x)}{\partial D(x)} = 0$$

Optimal value for D

$$\frac{\partial f(x)}{\partial D(x)} =$$

Optimal value for D

$$\frac{\partial f(x)}{\partial D(x)} = p_r(x)\frac{1}{\ln 10}\frac{1}{D(x)} - p_g(x)\frac{1}{\ln 10}\frac{1}{1 - D(x)}$$

Optimal value for D

$$\begin{aligned}\frac{\partial f(x)}{\partial D(x)} &= p_r(x)\frac{1}{\ln 10}\frac{1}{D(x)} - p_g(x)\frac{1}{\ln 10}\frac{1}{1-D(x)} \\ &= \frac{1}{\ln 10}\left(\frac{p_r(x)}{D(x)} - \frac{p_g(x)}{1-D(x)}\right)\end{aligned}$$

Optimal value for D

$$\begin{aligned}
\frac{\partial f(x)}{\partial D(x)} &= p_r(x)\frac{1}{\ln 10}\frac{1}{D(x)} - p_g(x)\frac{1}{\ln 10}\frac{1}{1-D(x)} \\
&= \frac{1}{\ln 10}\left(\frac{p_r(x)}{D(x)} - \frac{p_g(x)}{1-D(x)}\right) \\
&= \frac{1}{\ln 10}\left(\frac{p_r(x) - (p_r(x)+p_g(x))D(x)}{D(x)(1-D(x))}\right)
\end{aligned}$$

Optimal value for D

$$\begin{aligned}\frac{\partial f(x)}{\partial D(x)} &= p_r(x)\frac{1}{\ln 10}\frac{1}{D(x)} - p_g(x)\frac{1}{\ln 10}\frac{1}{1-D(x)} \\ &= \frac{1}{\ln 10}\left(\frac{p_r(x)}{D(x)} - \frac{p_g(x)}{1-D(x)}\right) \\ &= \frac{1}{\ln 10}\left(\frac{p_r(x) - (p_r(x)+p_g(x))D(x)}{D(x)(1-D(x))}\right)\end{aligned}$$

and so setting this equals to zero gives

$$D^*(x) = \frac{p_r(x)}{p_r(x)+p_g(x)}$$

Global Optimal

$$D^*(x) = \frac{p_r(x)}{p_r(x) + p_g(x)}$$

plugging in the loss we have

$$\begin{aligned}
&L(G, D^*) \\
&= \mathbb{E}_{x \sim p_r(x)}[\log D^*(x)] + \mathbb{E}_{x \sim p_g(x)}[\log(1 - D^*(x))] \\
&= \mathbb{E}_{x \sim p_r(x)}\left[\log \frac{p_r(x)}{p_r(x) + p_g(x)}\right] + \mathbb{E}_{x \sim p_g(x)}\left[\frac{p_g(x)}{p_r(x) + p_g(x)}\right]
\end{aligned}$$

Global Optimal

$$D^*(x) = \frac{p_r(x)}{p_r(x) + p_g(x)}$$

plugging in the loss we have

$$\begin{aligned}
&L(G, D^*) \\
&= \mathbb{E}_{x \sim p_r(x)}[\log D^*(x)] + \mathbb{E}_{x \sim p_g(x)}[\log(1 - D^*(x))] \\
&= \mathbb{E}_{x \sim p_r(x)}\left[\log \frac{p_r(x)}{p_r(x) + p_g(x)}\right] + \mathbb{E}_{x \sim p_g(x)}\left[\frac{p_g(x)}{p_r(x) + p_g(x)}\right] \\
&= -2\log(2) + \mathbb{E}_{x \sim p_r(x)}\left[\log \frac{2p_r(x)}{p_r(x) + p_g(x)}\right] + \mathbb{E}_{x \sim p_g(x)}\left[\frac{2p_g(x)}{p_r(x) + p_g(x)}\right]
\end{aligned}$$

Global Optimal

$$D^*(x) = \frac{p_r(x)}{p_r(x) + p_g(x)}$$

plugging in the loss we have

$$\begin{aligned}
&L(G, D^*) \\
&= \mathbb{E}_{x \sim p_r(x)}[\log D^*(x)] + \mathbb{E}_{x \sim p_g(x)}[\log(1 - D^*(x))] \\
&= \mathbb{E}_{x \sim p_r(x)}\left[\log \frac{p_r(x)}{p_r(x) + p_g(x)}\right] + \mathbb{E}_{x \sim p_g(x)}\left[\frac{p_g(x)}{p_r(x) + p_g(x)}\right] \\
&= -2\log(2) + \mathbb{E}_{x \sim p_r(x)}\left[\log \frac{2p_r(x)}{p_r(x) + p_g(x)}\right] + \mathbb{E}_{x \sim p_g(x)}\left[\frac{2p_g(x)}{p_r(x) + p_g(x)}\right] \\
&= -\log(4) + \mathrm{KL}(p_r \,||\, (p_r(x) + p_g(x))/2) + \mathrm{KL}(p_g \,||\, (p_r(x) + p_g(x))/2)
\end{aligned}$$

Digression: Statistical Divergence

- A statistical divergence measures how a probability distribution P differs from another probability distribution Q
- Kullback-Leibler divergence:

$$\mathrm{KL}(P \mid\mid Q) \stackrel{\text{def.}}{=} E_P\left[log\left(\frac{P}{Q}\right)\right]$$

- Jensen-Shannon divergence:

$$\mathrm{JS}(P \mid\mid Q) \stackrel{\text{def.}}{=} \frac{1}{2}\mathrm{KL}(P \mid\mid (P+Q)/2) + \frac{1}{2}\mathrm{KL}(Q \mid\mid (P+Q)/2)$$

$\longrightarrow$ goes to 0 when P and Q are the same

Back to Global Optimal...

$$D^*(x) = \frac{p_r(x)}{p_r(x) + p_g(x)}$$

plugging in the loss we have

$$\begin{aligned} & L(G, D^*) \\ & = -\log(4) + \mathrm{KL}(p_r \;||\; (p_r(x) + p_g(x))/2) + \mathrm{KL}(p_g \;||\; (p_r(x) + p_g(x))/2) \\ & = -\log(4) + \mathrm{JS}(p_r \;||\; p_g) \end{aligned}$$

$\longrightarrow$ when p_g is trained to be very close to p_r, we achieve the theoretical global optimum $L(G^*, D^*) = -\log(4)$

Training GAN

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Goodfellow, Ian, et al. Generative Adversarial Nets. NeurIPS 2014

Training GAN

Proposition 2. *If G and D have enough capacity, and at each step of Algorithm 1, the discriminator is allowed to reach its optimum given G, and p_g is updated so as to improve the criterion*

$$\mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log D_G^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim p_g}[\log(1 - D_G^*(\boldsymbol{x}))]$$

then p_g converges to p_{data}

Goodfellow, Ian, et al. Generative Adversarial Nets. NeurIPS 2014

Training GAN

Proposition 2. *If G and D have enough capacity, and at each step of Algorithm 1, the discriminator is allowed to reach its optimum given G, and p_g is updated so as to improve the criterion*

$$\mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log D_G^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim p_g}[\log(1 - D_G^*(\boldsymbol{x}))]$$

then p_g converges to p_{data}

$\longrightarrow$ but reality is far from theory...

Goodfellow, Ian, et al. Generative Adversarial Nets. NeurIPS 2014

Outline

Hard to achieve a global optimum (Nash Equilibrium)

Proposition 2. *If G and D have enough capacity, and at each step of Algorithm 1, the discriminator is allowed to reach its optimum given G, and p_g is updated so as to improve the criterion*

$$\mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log D_G^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim p_g}[\log(1 - D_G^*(\boldsymbol{x}))]$$

then p_g converges to p_{data}

Simple example:

$$\min_x \max_y f(x, y) = xy$$

Goodfellow, Ian, et al. Generative Adversarial Nets. NeurIPS 2014

Hard to achieve a global optimum (Nash Equilibrium)

Proposition 2. *If G and D have enough capacity, and at each step of Algorithm 1, the discriminator is allowed to reach its optimum given G, and p_g is updated so as to improve the criterion*

$$\mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log D_G^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim p_g}[\log(1 - D_G^*(\boldsymbol{x}))]$$

then p_g converges to p_{data}

Simple example:

$$\min_x \max_y f(x, y) = xy$$

equivalent with $\min_x xy$ and $\min_y -xy$

Goodfellow, Ian, et al. Generative Adversarial Nets. NeurIPS 2014

Hard to achieve a global optimum (Nash Equilibrium)

Simple example:

$$\min_x \max_y f(x, y) = xy$$

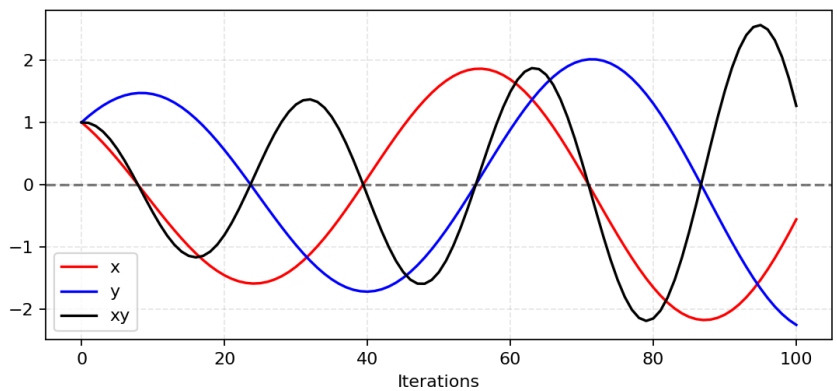
equivalent with $\min_x xy$ and $\min_y -xy$

Differentiating gives:

$$\frac{\partial f}{\partial x} = y \quad \frac{\partial f}{\partial y} = -x$$

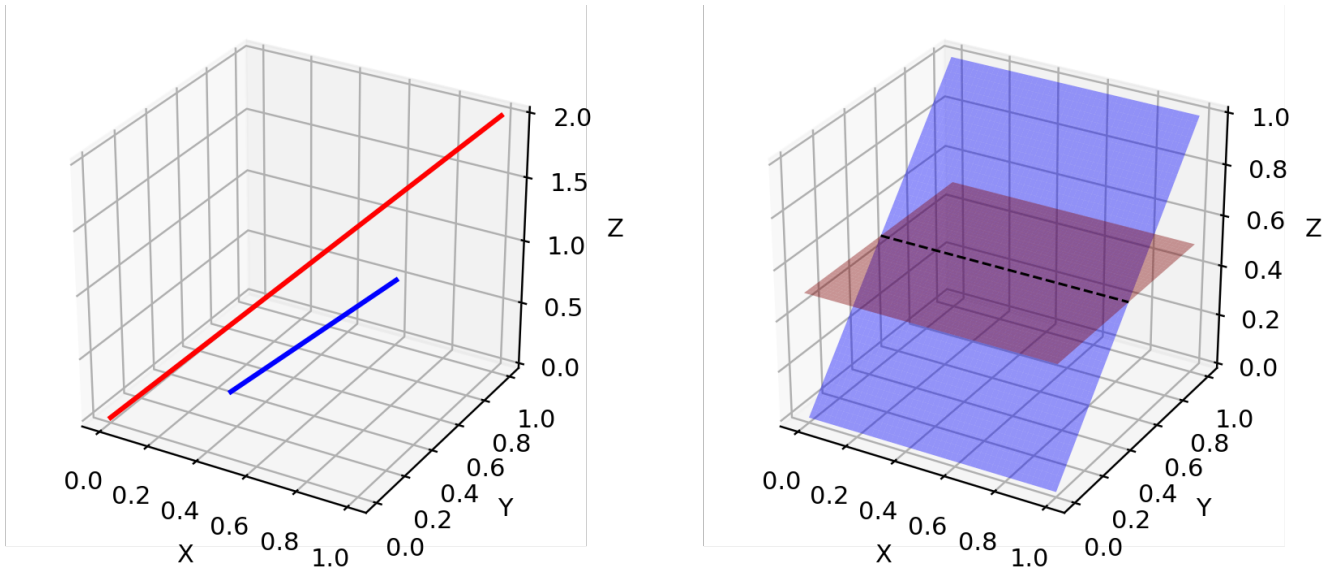
So when doing GD:

$$x^{(k+1)} = x^{(k)} - \eta y \quad y^{(k+1)} = y^{(k)} + \eta x$$



Salimans, Tim, et al. "Improved techniques for training gans." NeurIPS 2016

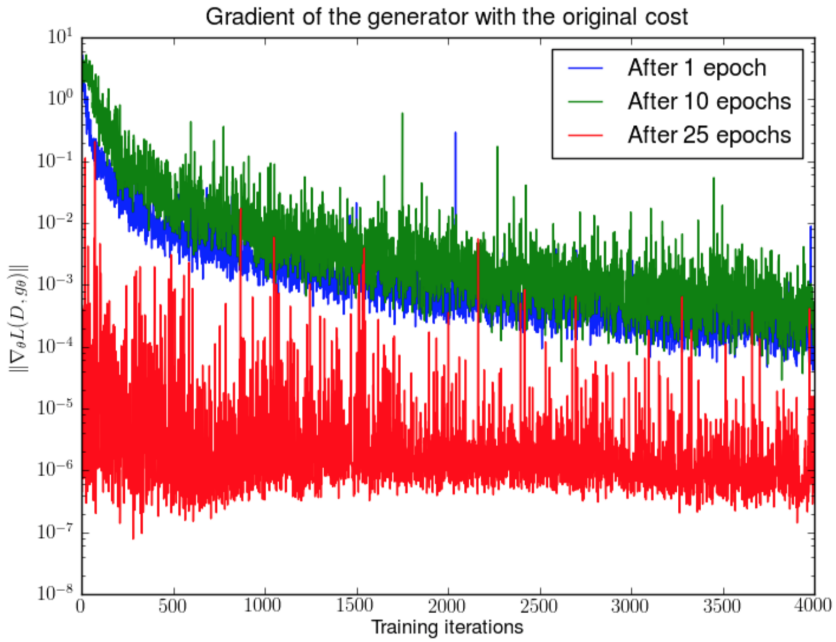
Supports of p_r and p_g concentrates in lower dimension



Meaning: for high-dimensional data, p_r and p_g can be very hard to train to match each others

$\longrightarrow$ our discriminator has easy time to distinguish real and fake samples

Vanishing Gradient



Recall that

$$\min_G \max_D L(D, G) = \mathbb{E}_{x \sim p_r(x)}[\log D(x)] + \mathbb{E}_{x \sim p_g(x)}[\log(1 - D(x))]$$

$\longrightarrow$ if the discriminator does a great job ($D(x) = 1$ for all $x \sim p_r(x)$ and $D(x) = 0$ for all $x \sim p_g(x)$), the gradient of the loss function drops down to close to zero

$\longrightarrow$ SGD update no longer works

Mode Collapse

Mode Collapse: the generator always produces same outputs in training

---

Arjovsky et. al. "Wasserstein generative adversarial networks." ICML 2017

Lack of Proper Evaluation Metric

When training GANs:

- We can check whether if the loss decreases – but it does not tell the whole story (recall vanishing gradient slide)
- How to evaluate the generated images' quality? (asking human being?)
- *i.e.* lack an objective function to compare performance of different architectures

Outline

To read more

- This part is based on Sectio 3 of Salimans, Tim, et al. "Improved techniques for training gans." NeurIPS 2016
- Related: Arjovsky and Bottou. "Towards Principled Methods for Training Generative Adversarial Networks." ICLR 2017.
- Also related: Huszár "How (not) to train your generative model: Scheduled sampling, likelihood, adversary?." ICLR 2015.
- Arguably the original GAN proposal of Goodfellow et al. (2014) worked because they switched the loss from using asymmetric Kullback-Leibler divergence to symmetric Jensen-Shannon divergence

Feature Matching

- Objective: lessen the stability of GANs training
- Idea: discriminator to inspect whether the generator's output matches expected statistics of the real samples
- Implementation: training the generator to match the expected value of the features on an intermediate layer of the discriminator
- Denoting $f(\mathrm{x})$ the summary on an intermediate layer of the discriminator, now the objective is

$$\|\mathbb{E}_{x\sim p_r} f(x) - \mathbb{E}_{x\sim p_z} f(G(z))\|_2^2$$

- In practice: $f(x)$ can be mean or median (summary statistics) of the features

Minibatch Discrimination

- Objective: avoid mode-collapse in the generator
- Idea: the discriminator is able to account for the relationship between training data points in one batch, instead of each point independently
- Implementation: In one minibatch, we approximate the closeness between every pair of samples, denoting $c_b(x_i, x_j)_{i,j\in[n]}$, and get the overall summary of one data point by summing up how close it is to other samples in the same batch

$$o_b(x_i) = \sum_j c_b(x_i, x_j)$$

- Then we add $o_b(x_i)$ to the input of the model

Historical Averaging

- Objective: bypass the low-dimensional manifold support problem and saddle points for the non-convex minimax optimization
- Idea: adding to the generator and the discriminator's cost a term

$$\|\theta - \frac{1}{t}\sum_{i=1}^{t}\theta(i)\|_2^2$$

  where $\theta(i)$ is the parameters at the previous time i
- Implementation: scale better (*i.e.* faster training for large dataset) by adding only $\theta(i-1)$ to the i-th training iteartion (online learning)

One-sided label smoothing

- Objective: soften the discriminator value to avoid network's vulnerability
- Idea: smoothing label of discriminator network with α for positive label (or 1 if the input is real data), and β for negative label
- The optimal D is then

$$D(\mathrm{x}) = \frac{\alpha p_r(x) + \beta p_g(x)}{p_r(x) + p_g(x}$$

- Example: $\alpha = 0.9$ and $\beta = 0.1$

Virtual Batch Normalization

- Batch-norm[1] is one of the most popular tricks for fastening and stabilizing the training of deep neural networks
- The idea of batch-norm is very simple: normalized each layer's input (activation) with their mean and covariance
- However: batch-norm can make output of a neural network highly dependent on some other inputs in the same minibatch
- Idea: in virtual batch-norm, we normalize the input based on a collection of fixed inputs (called reference batch)
- Implementation: virtual batch-norm is more expensive (forward pass on two minbatches of data) – so only used in the generator network

[1] Ioffe and Szegedy . "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift" (2015)

Better metric accounting for Distribution Similarity

- We already argue KL is not the best divergence for GAN loss, but also the JS divergence: when the two distributions lies on low dimensional manifold, it is hard to match them with gradient descent
- This is why the concept of Wasserstein GAN was introduced [2]

[2] Arjovsky et. al. "Wasserstein generative adversarial networks." ICML 2017

Outline

Previously

$$\min_G \max_D L(D, G) = \mathbb{E}_{x \sim p_r(x)}[\log D(x)] + \mathbb{E}_{x \sim p_g(x)}[\log(1 - D(x))]$$

GANs training suffers from many problems:

- Stability of the optimization scheme
- Vanishing gradient
- Mode collapse

with many training fixes

Better metric accounting for Distribution Similarity

- We already argue KL is not the best divergence for GAN loss, but also the JS divergence: when the two distributions lies on low dimensional manifold, it is hard to match them with gradient descent
- This is why the concept of Wasserstein GAN was introduced [3]

Wasserstein Generative Adversarial Networks

Martin Arjovsky [1] **Soumith Chintala** [2] **Léon Bottou** [1 2]

Abstract

We introduce a new algorithm named WGAN, an alternative to traditional GAN training. In this new model, we show that we can improve the stability of learning, get rid of problems like mode collapse, and provide meaningful learning curves useful for debugging and hyperparameter searches. Furthermore, we show that the corresponding optimization problem is sound, and provide extensive theoretical work highlighting the deep connections to different distances between distributions.

The typical remedy is to add a noise term to the model distribution. This is why virtually all generative models described in the classical machine learning literature include a noise component. In the simplest case, one assumes a Gaussian noise with relatively high bandwidth in order to cover all the examples. It is well known, for instance, that in the case of image generation models, this noise degrades the quality of the samples and makes them blurry. For example, we can see in the recent paper (Wu et al., 2016) that the optimal standard deviation of the noise added to the model when maximizing likelihood is around 0.1 to each pixel in a generated image, when the pixels were already normalized to be in the range $[0, 1]$. This is a very

[3] Arjovsky et. al. "Wasserstein generative adversarial networks." ICML 2017

Statistical Distance

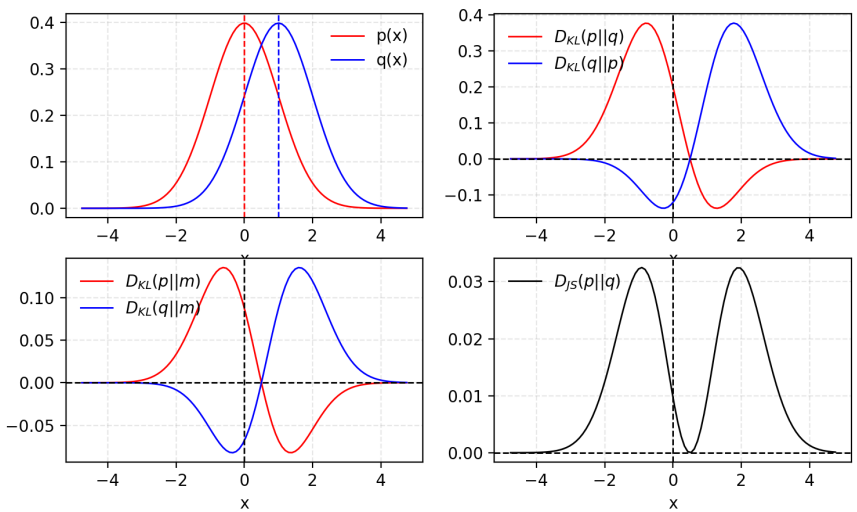
- Distance in $\mathbb{R}^d$? What is the distance between two vectors? Two matrices?

Statistical Distance

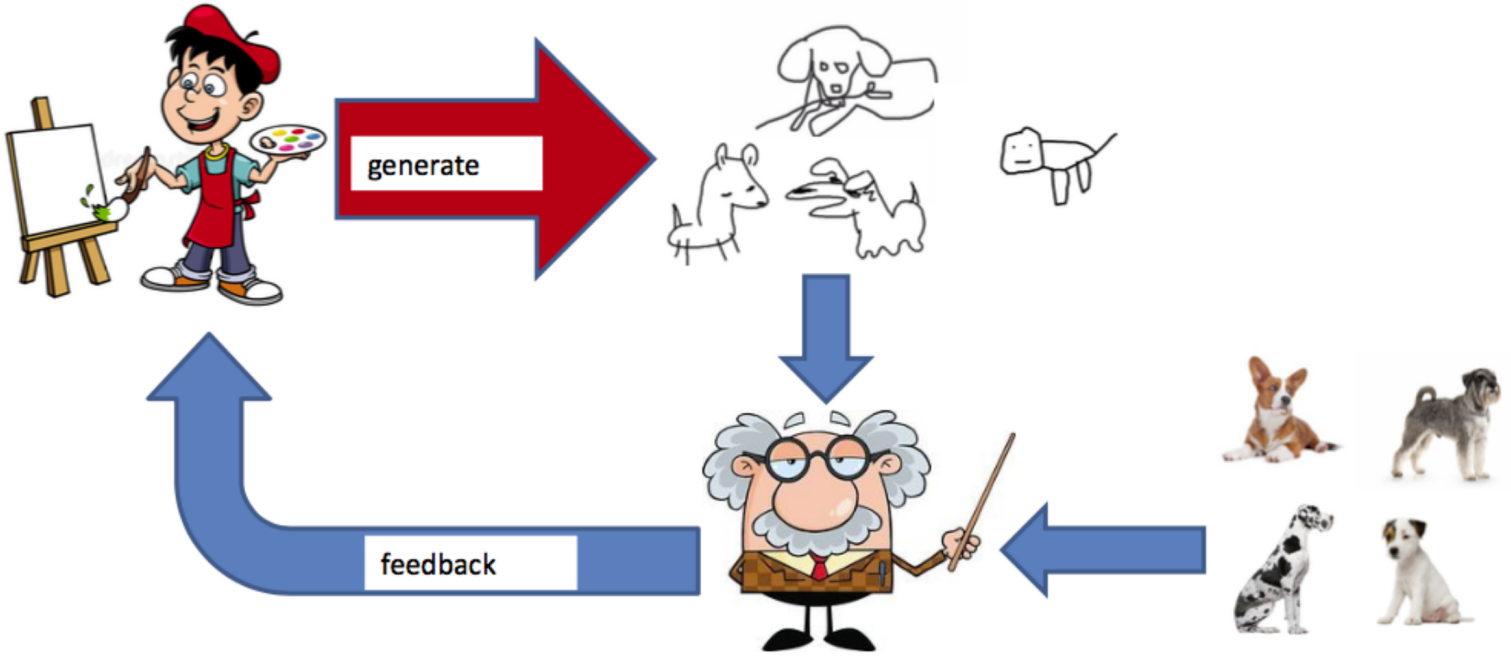
- Distance in $\mathbb{R}^d$? What is the distance between two vectors? Two matrices?
- What if we want to measure the distance between two probability distributions P and Q?

Statistical Distance

- Distance in $\mathbb{R}^d$? What is the distance between two vectors? Two matrices?
- What if we want to measure the distance between two probability distributions P and Q?
- Last week: Kullback-Leibler and Jensen-Shannon divergences



Reminder: Why do we care about the prob. distance?



In GAN architecure, we have $p_r(x)$ and $p_g(x)$
where $p_r(x)$ the distribution of real dataset
$p_g(x)$ the generator learned distribution over data x

Background on Optimal Transport – next topic

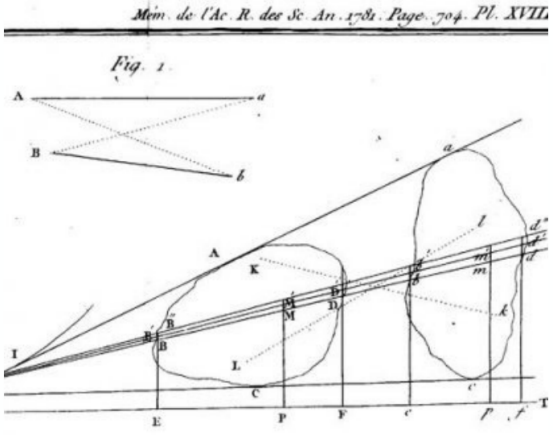
(1784)

MÉMOIRE
SUR LA
THÉORIE DES DÉBLAIS
ET DES REMBLAIS.

Par M. MONGE.

LORSQU'ON doit transporter des terres d'un lieu dans un autre, on a coutume de donner le nom de *Déblai* au volume des terres que l'on doit transporter, & le nom de *Remblai* à l'espace qu'elles doivent occuper après le transport.

Le prix du transport d'une molécule étant, toutes choses d'ailleurs égales, proportionnel à son poids & à l'espace qu'on lui fait parcourir, & par conséquent le prix du transport total devant être proportionnel à la somme des produits des molécules multipliées chacune par l'espace parcouru, il s'ensuit que le déblai & le remblai étant donnés de figure & de position, il n'est pas indifférent que telle molécule du déblai soit transportée dans tel ou tel autre endroit du remblai, mais qu'il y a une certaine distribution à faire des molécules du premier dans le second, d'après laquelle la somme de ces produits sera la moindre possible, & le prix du transport total sera un *minimum*.



Wasserstein-1 distance as a GAN metric

$$\mathcal{W}_1(p_r, p_g) \overset{\text{def.}}{=} \min_{\gamma \in \Pi(p_r, p_g)} \mathbb{E}_{(x,x') \sim \gamma} \|x - x'\|$$

- Also called Earth-Mover distance (EMD)
- Imagine the two distributions are a certain amount of piles of dirt over a region
- γ is one way to transport one pile of dirt to the other
- Therefore we have the constraint

$$\sum_x \gamma(x, x') = p_g(x')$$

$$\sum_{x'} \gamma(x, x') = p_r(x)$$

- EMD is the minimum cost of moving them

Wasserstein-1 distance as a GAN metric

$$\mathcal{W}_1(p_r, p_g) \overset{\text{def.}}{=} \min_{\gamma \in \Pi(p_r, p_g)} \mathbb{E}_{(x,x') \sim \gamma} \|x - x'\|$$

$\longrightarrow$ empirical version

$$W_1(p_r, p_g) \overset{\text{def.}}{=} \min_{\gamma \in \Pi(p_r, p_g)} \sum_{x,x'} \gamma(x, x') \|x - x'\|$$

When using W1 as a metric, we have the WGAN

Arjovsky et. al. "Wasserstein generative adversarial networks." ICML 2017

Wasserstein-1 vs JS/KL as a GAN loss?

It's all about the smoothness (*i.e.* differentiable). Example 1 in the paper

- Suppose $z \sim \mathcal{U}[0,1]$,
- $\mathbb{P}_0$ is the distribution of $(0, z) \in \mathbb{R}^2$
- $\mathbb{P}_\theta$ this distribution of $g_\theta(z) = (\theta, z)$ with $\theta \in (0,1)$ (can view this as our generator distribution)

Arjovsky et. al. "Wasserstein generative adversarial networks." ICML 2017

Wasserstein-1 vs JS/KL as a GAN loss?

What happens when one calculate these divergence? When $\theta \neq 0$

- Kullback-Leibler:

$$KL(\mathbb{P}_0, \mathbb{P}_\theta) = KL(\mathbb{P}_\theta, \mathbb{P}_0) = \sum_{x=0, y\sim\mathcal{U}[0,1]} p_x \log\left(\frac{1}{0}\right) = +\infty$$

- Jensen-Shannon:

$$JS(\mathbb{P}_0, \mathbb{P}_\theta) = 1/2 \left(\sum_{x=0, y\sim\mathcal{U}[0,1]} 1 \log \frac{1}{1/2} + \sum_{x=\theta, y\sim\mathcal{U}[0,1]} 1 \log \frac{1}{1/2} \right) = log2$$

- EMD:

$$W(\mathbb{P}_0, \mathbb{P}_\theta) = |\theta|$$

When $\theta = 0 \longrightarrow KL(\mathbb{P}_0, \mathbb{P}_\theta) = JS(\mathbb{P}_0, \mathbb{P}_\theta) = 0$ but

$$W(\mathbb{P}_0, \mathbb{P}_\theta) = 0 = |\theta|$$

Wasserstein-1 vs JS/KL as a GAN loss?

Conclusion from the simple example:

- KL and JS divergences are not smooth (differentiable) everywhere
- Using Wasserstein-1 distance fix this issue

Wasserstein-1 as a GAN loss?

$$\mathcal{W}_1(p_r, p_g) \overset{\text{def.}}{=} \min_{\gamma \in \Pi(p_r, p_g)} \mathbb{E}_{(x,x') \sim \gamma} \|x - x'\|$$

Problem: this formulation is highly intractable

$\longrightarrow$ Kantorovich-Rubinstein duality for W1 distance

$$\mathcal{W}_1(p_r, p_g) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim p_r}[f(x)] - \mathbb{E}_{x \sim p_g}[f(x)]$$

Remark 6.5, Villani, Cédric. Optimal transport: old and new. Vol. 338. Berlin: springer, 2009.

Wasserstein-1 as a GAN loss?

In other words, with the W1 distance, we move from the loss

$$\min_G \max_D L(D, G) = \mathbb{E}_{x \sim p_r(x)}[\log D(x)] + \mathbb{E}_{x \sim p_g(x)}[\log(1 - D(x))]$$

to

$$\max_w \mathbb{E}_{x \sim p_r}[f_w(x)] - \mathbb{E}_{z \sim p_z}[f_w(g_\theta(z))]$$

Important improvements:

- From minimax loss (hard to solve) to single maximization
- f_w is a parameterization: now we can use any architectures of neural networks to train

Wasserstein-1 as a GAN loss?

Theorem 3: with the loss

$$\mathcal{W}_1(p_r, p_g) = \max_w \mathbb{E}_{x \sim p_r}[f_w(x)] - \mathbb{E}_{z \sim p_z}[f_w(g_\theta(z))]$$

Under additional technical assumption (we skip it here, but it relates to local Lipschitz continuity), we have

$$\nabla_w \mathcal{W}_1(p_r, p_g) = -\mathbb{E}_{z \sim p_z}[\nabla_w f_w(g_\theta(z))]$$

Arjovsky et. al. "Wasserstein generative adversarial networks." ICML 2017

Wasserstein-1 as a GAN loss?

$$\nabla_w \mathcal{W}_1(p_r, p_g) = -\mathbb{E}_{z \sim p_z}[\nabla_w f_w(g_\theta(z))]$$

Intuition:

- Now the "discriminator" D is not a direct critic of telling the fake samples apart from the real ones anymore.
- Instead, it is trained to learn f_w to help compute Wasserstein distance.
- Loss function decreases $\longrightarrow$ Wasserstein-1 distance gets smaller
- Generator g_θ output becomes closer to the real data distribution (or p_g becomes very similar to p_r)

Training WGAN

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

Arjovsky et. al. "Wasserstein generative adversarial networks." ICML 2017

Fixing vanishing gradient in some settings

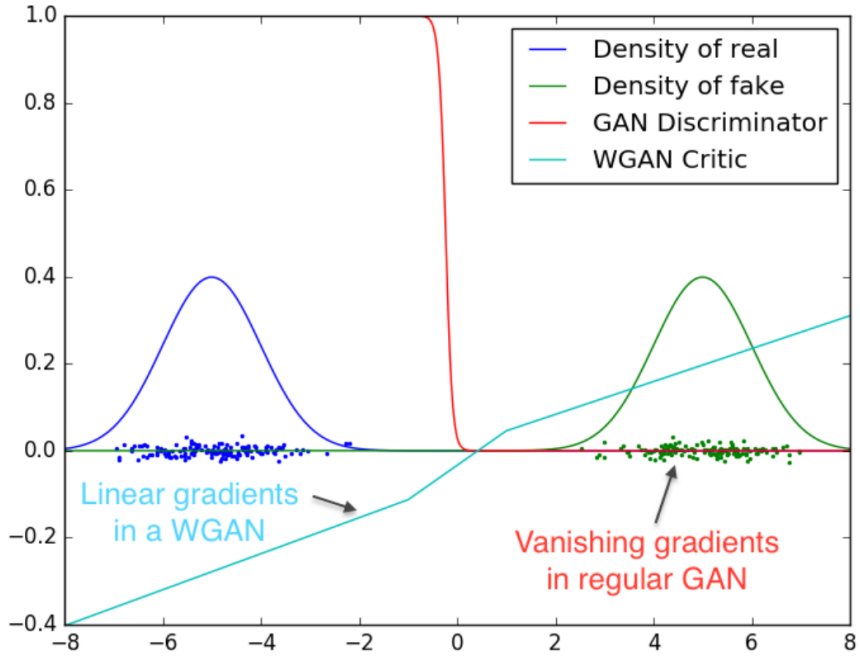


Figure 2: Optimal discriminator and critic when learning to differentiate two Gaussians. As we can see, the discriminator of a minimax GAN saturates and results in vanishing gradients. Our WGAN critic provides very clean gradients on all parts of the space.

Arjovsky et. al. "Wasserstein generative adversarial networks." ICML 2017

Better at image generation task

Figure 6: Algorithms trained with a DCGAN generator. Left: WGAN algorithm. Right: standard GAN formulation. Both algorithms produce high quality samples.

Figure 7: Algorithms trained with a generator without batch normalization and constant number of filters at every layer (as opposed to duplicating them every time as in (Radford et al., 2015)). Aside from taking out batch normalization, the number of parameters is therefore reduced by a bit more than an order of magnitude. Left: WGAN algorithm. Right: standard GAN formulation. As we can see the standard GAN failed to learn while the WGAN still was able to produce samples.

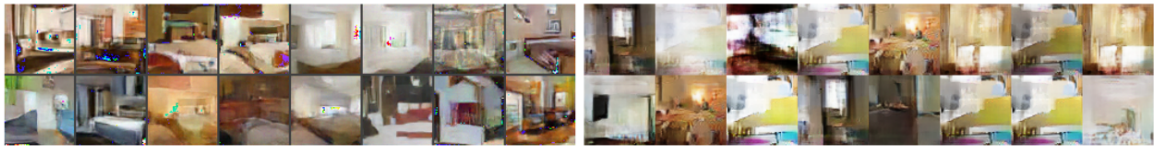


Figure 8: Algorithms trained with an MLP generator with 4 layers and 512 units with ReLU nonlinearities. The number of parameters is similar to that of a DCGAN, but it lacks a strong inductive bias for image generation. Left: WGAN algorithm. Right: standard GAN formulation. The WGAN method still was able to produce samples, lower quality than the DCGAN, and of higher quality than the MLP of the standard GAN. Note the significant degree of mode collapse in the GAN MLP.

Arjovsky et. al. "Wasserstein generative adversarial networks." ICML 2017

However...

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

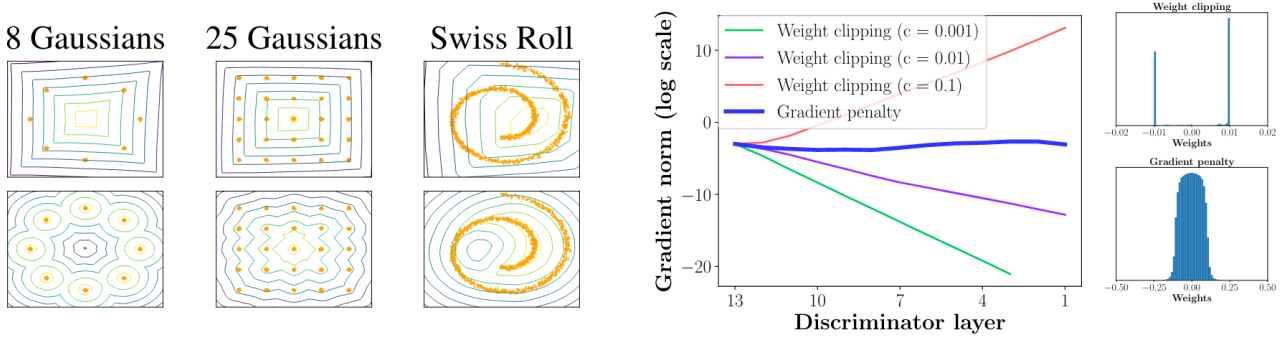
while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

From the paper: 'Weight clipping is a clearly terrible way to enforce a Lipschitz constrain [...] we leave the topic of enforcing Lipschitz constraints in a NN setting for further investigation'

Arjovsky et. al. "Wasserstein generative adversarial networks." ICML 2017

Difficulties with Weight Constraints

- Clipping the weights inside a range (*e.g.* (-0.01, 0.01)) actually can lead to optimization difficulties



(a) Value surfaces of WGAN critics trained to optimality on toy datasets using (top) weight clipping and (bottom) gradient penalty. Critics trained with weight clipping fail to capture higher moments of the data distribution. The 'generator' is held fixed at the real data plus Gaussian noise.

(b) (left) Gradient norms of deep WGAN critics during training on the Swiss Roll dataset either explode or vanish when using weight clipping, but not when using a gradient penalty. (right) Weight clipping (top) pushes weights towards two values (the extremes of the clipping range), unlike gradient penalty (bottom).

Figure 1: Gradient penalty in WGANs does not exhibit undesired behavior like weight clipping.

Gulrajani, Ishaan, et al. "Improved training of wasserstein gans." NeuRIPS 2017.

WGAN-GP: a variant of GAN variant...

Introducing a regularization term that penalize the gradient to the loss

$$\mathcal{W}_1(p_r, p_g) = \max_w \mathbb{E}_{x\sim p_r}[f_w(x)] - \mathbb{E}_{z\sim p_z}[f_w(g_\theta(z))] + \lambda \mathbb{E}_{z\sim p_z}[(\|\nabla_w f_w(g_\theta(z))\|_2 - 1)^2]$$

- This is equivalent to constraining the function f to be 1-Lipschitz
- Regularization terms is still differentiable

Training WGAN-GP

Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

Require: The gradient penalty coefficient λ, the number of critic iterations per generator iteration n_{critic}, the batch size m, Adam hyperparameters α, β_1, β_2.
Require: initial critic parameters w_0, initial generator parameters θ_0.

while θ has not converged **do**
  for $t = 1, ..., n_{\text{critic}}$ **do**
    for $i = 1, ..., m$ **do**
      Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
      $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
      $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
      $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
    end for
    $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
  end for
  Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p(\boldsymbol{z})$.
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
end while

Gulrajani, Ishaan, et al. "Improved training of wasserstein gans." NeuRIPS 2017.

WGAN-GP vs. WGAN vs. other GANs

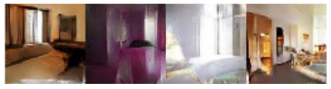
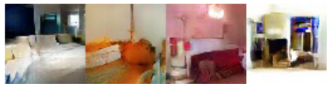
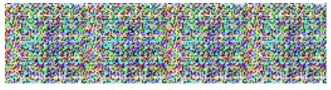
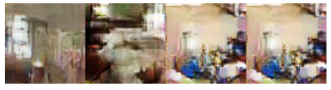
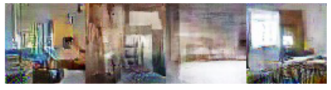
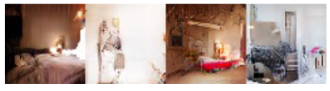
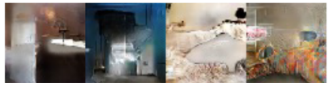
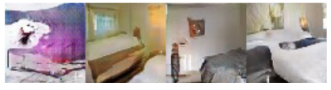
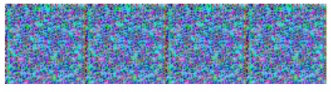
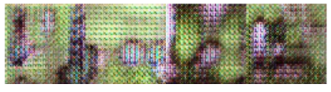
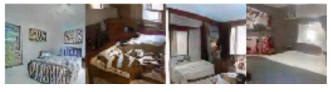
DCGAN	LSGAN	WGAN (clipping)	WGAN-GP (ours)
Baseline (G: DCGAN, D: DCGAN)			
			
G: No BN and a constant number of filters, D: DCGAN			
			
G: 4-layer 512-dim ReLU MLP, D: DCGAN			
			
No normalization in either G or D			
			
Gated multiplicative nonlinearities everywhere in G and D			
			
tanh nonlinearities everywhere in G and D			
			
101-layer ResNet G and D			
			

Figure 2: Different GAN architectures trained with different methods. We only succeeded in training every architecture with a shared set of hyperparameters using WGAN-GP.

Conclusion

- Using Wasserstein-1 distance is better for GAN training (taken into account we are using the same architecture for generator), which leads to significantly improvement in image-generation task
- However, clipping weight as a way to enforce Lipschitz continuity is some kind of mysterious engineering technique
- With adding regularization term to enforce the norm of gradient to be equal to 1, WGAN-GP suffers less instability problem in training