Unsupervised Learning & Generative AI

Shan-Hung Wu
shwu@cs.nthu.edu.tw

Department of Computer Science,
National Tsing Hua University, Taiwan

Machine Learning

Outline

1. **Unsupervised Learning**
   - Text Models
   - Image Models
2. **ChatGPT**
3. **Autoencoders (AE)**
   - Manifold Learning*
4. **Variational Autoencoders (VAE)**
5. **Flow-based Models**
6. **Diffusion Models**
7. **Generative Adversarial Networks***
   - Basic Architecture
   - Challenges
   - More GANs

Outline

Unsupervised Learning

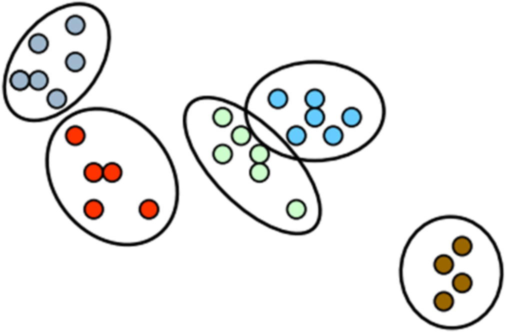
- Dataset: $\mathbb{X} = \{\boldsymbol{x}^{(i)}\}_i$, where $\boldsymbol{x}^{(i)}$'s are i.i.d. samples of $\mathbf{x}$
  - No supervision $\boldsymbol{y}^{(i)}$ (labels)
- What can we learn without labels?

Unsupervised Learning

- Dataset: $\mathbb{X} = \{\boldsymbol{x}^{(i)}\}_i$, where $\boldsymbol{x}^{(i)}$'s are i.i.d. samples of $\mathbf{x}$
  - No supervision $\boldsymbol{y}^{(i)}$ (labels)
- What can we learn without labels? The ***structures*** in $\mathbb{X}$
  - Inter-sample structures
  - Intra-sample structures

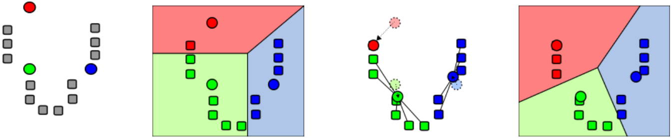
Clustering I

- Goal: to divide $\boldsymbol{x}^{(i)}$'s into K groups/***clusters***
  - Based on some similarity/distance measure between $\boldsymbol{x}^{(i)}$ and $\boldsymbol{x}^{(j)}$



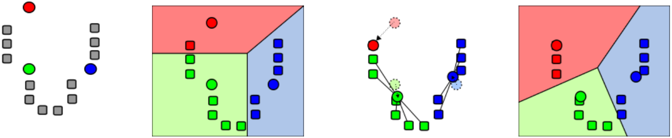
Clustering II

- K-means algorithm (K fixed): iteratively move clusterheads until convergence

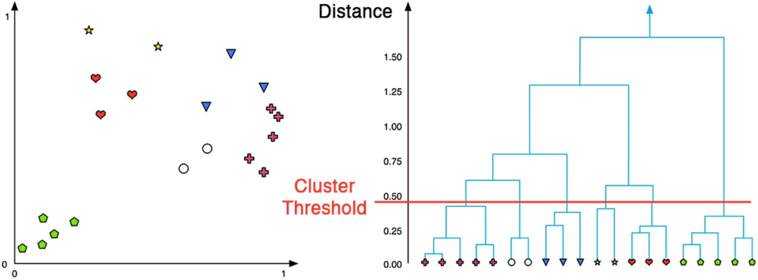


Clustering II

- K-means algorithm (K fixed): iteratively move clusterheads until convergence



- Hierarchical clustering (variable K): iteratively merge two points/groups,then cut



Factorization for Tabular $\mathbb{X}$

- Let $\boldsymbol{X}$ be a rating matrix where $\boldsymbol{X}_{i,:} = \boldsymbol{x}^{(i)}$

Items

Users

User/Item	A	B	C	D
Ted		4		3
Carol	3		2	
Bob		5		2
Alice			4	

Factors

Ted	1.4	.9
Carol	1.2	1
Bob	1.5	.9
Alice	1.2	.8

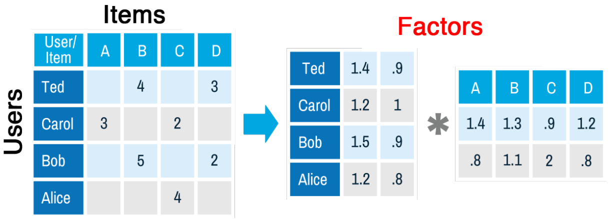
*

A	B	C	D
1.4	1.3	.9	1.2
.8	1.1	2	.8

- Goal: to approximate $\boldsymbol{X}$ with a dense matrix $\hat{\boldsymbol{X}} = \boldsymbol{W}\boldsymbol{H}$
  - To make recommendations based on $\hat{\boldsymbol{X}}_{i,:}$ for user i, known as ***collaborative filtering***
- How?

Factorization for Tabular $\mathbb{X}$

- Let $\boldsymbol{X}$ be a rating matrix where $\boldsymbol{X}_{i,:} = \boldsymbol{x}^{(i)}$



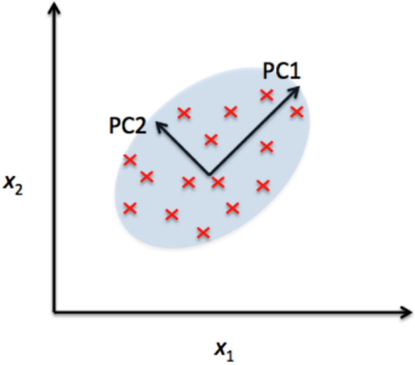
- Goal: to approximate $\boldsymbol{X}$ with a dense matrix $\hat{\boldsymbol{X}} = \boldsymbol{W}\boldsymbol{H}$
  - To make recommendations based on $\hat{\boldsymbol{X}}_{i,:}$ for user i, known as ***collaborative filtering***
- How? Non-negative matrix factorization (NMF) [19, 20]:

$$\arg \min_{\boldsymbol{W} \geq \boldsymbol{O}, \boldsymbol{H} \geq \boldsymbol{O}} \|\boldsymbol{X} - \boldsymbol{W}\boldsymbol{H}\|_F$$

  - So, positive elements in $\boldsymbol{W}$ and $\boldsymbol{H}$ can be seen as ***factor*** degrees

Dimension Reduction

- Goal: to learn a low dimensional representation $\mathbf{z}$ of $\mathbf{x}$
  - E.g., PCA



Self-Supervised Learning

- Goal: to learn a model that is able to "fill in the blanks"

$\boldsymbol{x}^{(i)}$  $\boldsymbol{y}^{(i)}$

Self-Supervised Learning

- Goal: to learn a model that is able to "fill in the blanks"
- Links unsupervised tasks with supervised models
  - Much more training data for models
  - Representation learning with deep networks

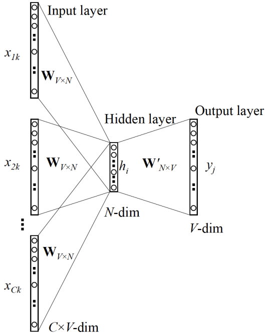
$\boldsymbol{x}^{(i)}$  $\boldsymbol{y}^{(i)}$

Example I: Word2Vec

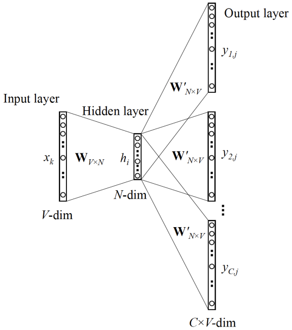
- Goal: to learn a model for blank filling

Example I: Word2Vec

- Goal: to learn a model for blank filling
  - E.g., word2vec [26, 25]: "... *the cat sat* **on**..."



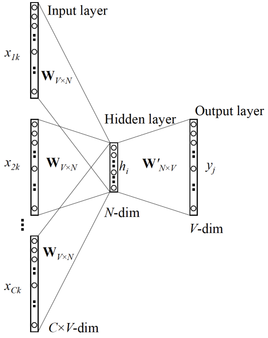
CBOW



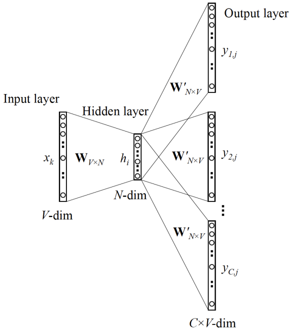
Skip Gram

Example I: Word2Vec

- Goal: to learn a model for blank filling
  - E.g., word2vec [26, 25]: "... *the cat sat* **<u>on</u>**..."



CBOW



Skip Gram

- Latent representation $\mathbf{h}$ encodes the ***semantics*** of a word
  - No need for synonym dictionary; big data tell that already

Example II: Doc2Vec

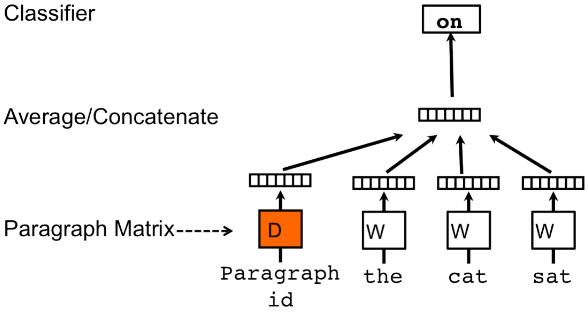
- How to encode a document?

Example II: Doc2Vec

- How to encode a document?
- Bag of words (TF-IDF), average word2vec, etc.
  - Do ***not*** capture the semantics due to sentence/paragraph/doc structure
    - "*John likes* Mary" $\neq$ "*Mary likes* John"

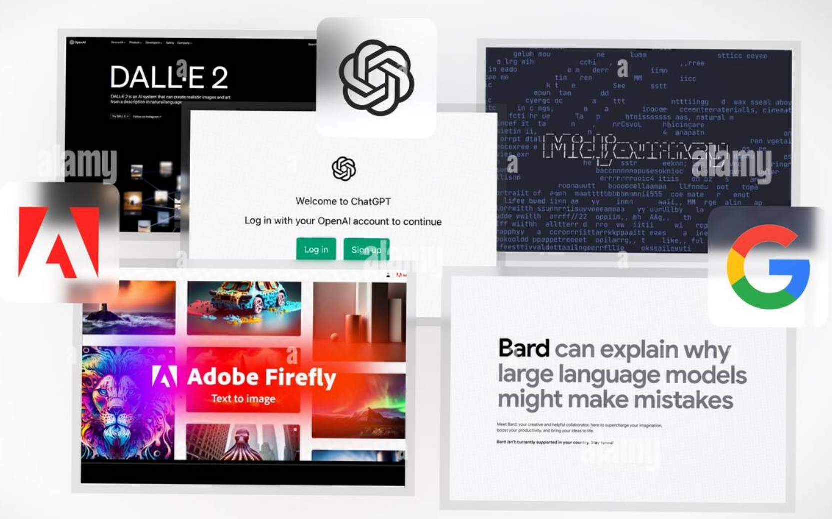
Example II: Doc2Vec

- How to encode a document?
- Bag of words (TF-IDF), average word2vec, etc.
  - Do ***not*** capture the semantics due to sentence/paragraph/doc structure
    - "*John likes* Mary" $\neq$ "*Mary likes* John"
- Why not apply self-supervised learning to docs?
  - Doc2vec [17]: to capture the ***context*** not explained by words
  - ***Transductive*** rather than inductive; does not work with unseen docs



Generative Models

- Goal: to generate new samples of $\mathbf{x}$
  - Can be conditioned on instructions (input)
- Largely based on self-supervised learning



Outline

1. **Unsupervised Learning**
    - Text Models
    - Image Models
2. **ChatGPT**
3. **Autoencoders (AE)**
    - Manifold Learning*
4. **Variational Autoencoders (VAE)**
5. **Flow-based Models**
6. **Diffusion Models**
7. **Generative Adversarial Networks***
    - Basic Architecture
    - Challenges
    - More GANs

Generating Text

- With large (self-supervised) training data, ***transformers*** [46] have shown to perform better than RNNs and CNNs
  - Mainly due to the $O(1)$ point distance
- Two common text models based on transformer:
  - BERT [7]: non-autoregressive
  - GPT [31]: autoregressive

BERT [7]

- Massively trained ***encoder*** of the original transformer
  - Non-autoregressive
- Pre-training tasks:
  - Masked language model ("Cloze" task)
    - "A quick brown [MASK] jumps over the lazy dog" → "fox" 11%, "ant" 5%, ...
  - Next sentence prediction
    - "[MASK] go to store [SEP] to buy a [MASK] of milk"→ True 93%, False 7%

One Pre-training, Multiple Fine-tuning Tasks

- Special input to identify downstream task: [CLS] token

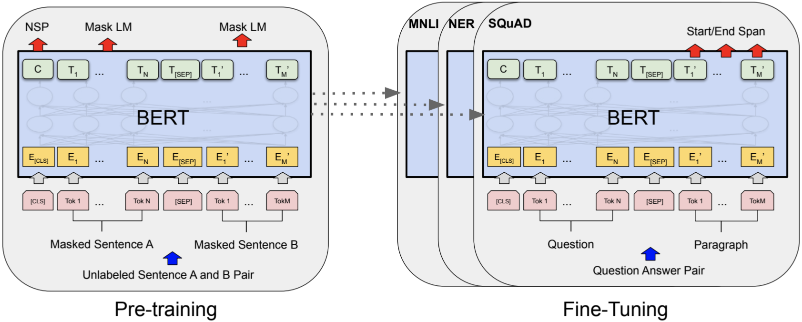
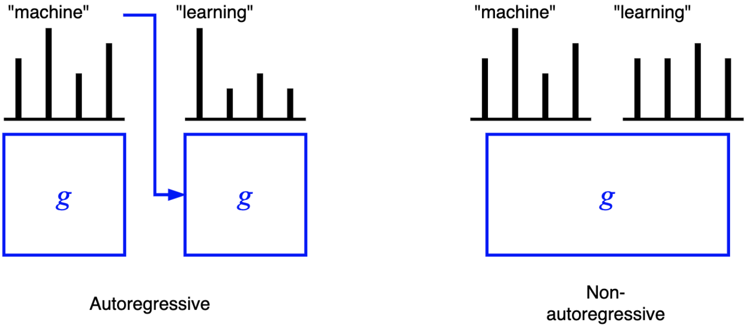


Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. `[CLS]` is a special symbol added in front of every input example, and `[SEP]` is a special separator token (e.g. separating questions/answers).

GPT [31]

- Massively trained ***decoder*** of the original transformer
  - Autoregressive
- Multitask pre-training by maximizing $\Pr(\text{output}|\text{input}, \text{task})$:
  - Translation: $\Pr(\text{french text}|\text{en text}, \text{translation})$
  - Question answering: $\Pr(\text{answer}|\text{question}, \text{qa})$
  - Reading comprehension: $\Pr(\text{answer}|\text{document}, \text{question}, \text{reading})$
- Usage for downstream task: fine-tuning or ***prompting***

Autoregressive or Not?

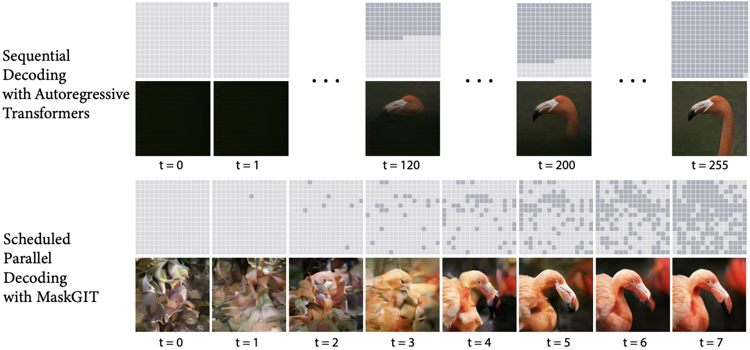


- It's easier for an autoregressive model to generate coherent text
- In this lecture, we focus on GPT and its variants

Outline

Autoregressive or Not?

- Autoregressive models can still give good (if not better) performance
  - E.g., PixelRNN [45], PixelCNN [44], or MaskGIT [5]
- But unlike in text domain, generating images pixel-by-pixel is ***very slow***
- Speed-up?
  - Scheduled ***parallel-pixel*** generation (e.g., MaskGIT)
  - ***Stepwise*** generation of ***whole*** images (e.g., flow-based and diffusion models)



Whole-Image Generation I

- Goal: given $\mathbb{X}$, to learn a ***generator function*** g such that $\hat{\boldsymbol{x}} = g(\boldsymbol{c};\Theta_g)$ looks like a real image in $\mathbb{X}$
  - $\boldsymbol{c}$ is a code or condition
  - Θ_g represents parameters of g
- Objective from Information Theory perspective:

$$\arg\min_g \mathrm{D}(\mathrm{P}_{\mathsf{data}} \| \mathrm{P}_g)$$

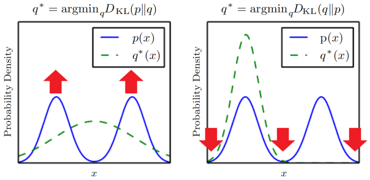
  - $\mathrm{P}_{\mathsf{data}}$ is the distribution of real data in the ground truth
  - P_g is the distribution of generated data
  - D is a divergence measure, e.g., D_{KL}

Whole-Image Generation I

- Goal: given $\mathbb{X}$, to learn a ***generator function*** g such that $\hat{\boldsymbol{x}} = g(\boldsymbol{c}; \Theta_g)$ looks like a real image in $\mathbb{X}$
  - $\boldsymbol{c}$ is a code or condition
  - Θ_g represents parameters of g
- Objective from Information Theory perspective:

$$\arg\min_g \mathrm{D}(\mathrm{P}_{\mathsf{data}} \| \mathrm{P}_g)$$

  - $\mathrm{P}_{\mathsf{data}}$ is the distribution of real data in the ground truth
  - P_g is the distribution of generated data
  - D is a divergence measure, e.g., D_{KL}
- Why not $\mathrm{D}_{\mathsf{KL}}(\mathrm{P}_g \| \mathrm{P}_{\mathsf{data}})$?



Whole-Image Generation II

- Minimizing $\mathrm{D_{KL}}(\mathrm{P_{data}} \| \mathrm{P}_g)$ amounts to maximizing $\mathrm{P}(\mathbb{X}|\Theta_g)$, the log likelihood of Θ_g:

$$
\begin{aligned}
g^* &= \arg\min_g \mathrm{D_{KL}}(\mathrm{P_{data}} \| \mathrm{P}_g) \\
&= \arg\max_g \mathrm{E}_{\mathbf{x}\sim \mathrm{P_{data}}}[\log \mathrm{P}_g(\mathbf{x})] + \mathrm{H}(\mathbf{x} \sim \mathrm{P_{data}}) \\
&= \arg\max_g \mathrm{E}_{\mathbf{x}\sim \mathrm{P_{data}}}[\log \mathrm{P}_g(\mathbf{x})] \\
&\approx \arg\max_g \textstyle\sum_{\boldsymbol{x}^{(i)} \in \mathbb{X}} \log \mathrm{P}_g(\boldsymbol{x}^{(i)}) \\
&= \arg\max_g \log \textstyle\prod_{\boldsymbol{x}^{(i)} \in \mathbb{X}} \mathrm{P}_g(\boldsymbol{x}^{(i)})
\end{aligned}
$$

$$
\begin{aligned}
\Theta_g^* &= \arg\max_{\Theta_g} \log \textstyle\prod_{\boldsymbol{x}^{(i)} \in \mathbb{X}} \mathrm{P}(\boldsymbol{x}^{(i)} | \Theta_g) \\
&= \arg\max_{\Theta_g} \log \mathrm{P}(\mathbb{X} | \Theta_g)
\end{aligned}
$$

- Other divergence measures can lead to similar results

Common Whole-Image Generation Methods

- Autoencoder (single-step: $\hat{\boldsymbol{x}} = g(\boldsymbol{c})$)
  - Pros: easy
  - Cons: but no creativity, blurry images
- Variational Autoencoder (single-step: $\hat{\boldsymbol{x}} = g(\boldsymbol{c})$)
  - Pros: creative
  - Cons: only maximizes a lower bound of $\mathrm{P}(\mathbb{X}|\Theta_g)$; blurry images
- Flow-based methods (multi-step: $\hat{\boldsymbol{x}} = g^{(T)}(\cdots g^{(2)}(g^{(1)}(\boldsymbol{c})))$)
  - Pros: maximizes $\mathrm{P}(\mathbb{X}|\Theta_g)$ directly
  - Cons: limited expressiveness of g for invertibility; slow training and inference
- GANs (single-step: $\hat{\boldsymbol{x}} = g(\boldsymbol{c})$)
  - Pros: good image quality (sharp and coherent)
  - Cons: difficult to train (convergence issue, mode collapse, vanishing gradients, etc.)
- Diffusion models (multi-step: $\hat{\boldsymbol{x}} = g^{(T)}(\cdots g^{(2)}(g^{(1)}(\boldsymbol{c})))$)
  - Pros: good image quality; efficient & stable to train, can be made conditional easily
  - Cons: slow inference

Outline

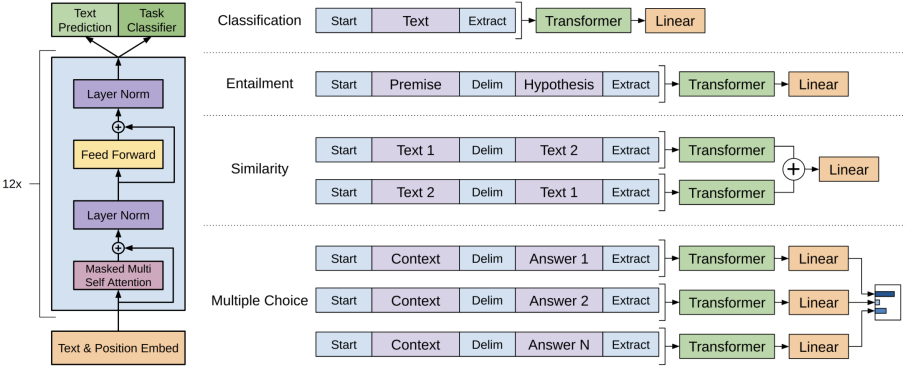
Evolutions

- 2018 GPTv1 [31]
  - Self-supervised pre-training
- 2019 GPTv2 [32]
  - ***Multitask*** pre-training
- 2020 GPTv3 [4]
  - ***Few-shot*** & ***in-context*** learning
- 2022 GPTv3.5 [29]
  - ***Alignment*** using (supervised) instruction tuning + reinforcement learning from human feedback (RLHF)
- 2023 GPT4: mixture of experts

GPTv1 [31]

- Aims at 2-step training process:
  1. Self-supervised pre-training on unlabeled data
     - To predict next word in a sentence (language model)
  2. Discriminative fine-tuning on labeled data in downstream tasks
     - Here, fine-tuning could also mean training a new model based on extracted features

Example Fine-tuning Tasks



Does Self-supervised Pre-training Help?

Table 5: Analysis of various model ablations on different tasks. Avg. score is a unweighted average of all the results. (*mc*= Mathews correlation, *acc*=Accuracy, *pc*=Pearson correlation)

Method	Avg. Score	CoLA (mc)	SST2 (acc)	MRPC (F1)	STSB (pc)	QQP (F1)	MNLI (acc)	QNLI (acc)	RTE (acc)
Transformer w/ aux LM (full)	74.7	45.4	91.3	82.3	82.0	**70.3**	**81.8**	**88.1**	**56.0**
Transformer w/o pre-training	59.9	18.9	84.0	79.4	30.9	65.5	75.7	71.2	53.8
Transformer w/o aux LM	**75.0**	**47.9**	**92.0**	**84.9**	**83.2**	69.8	81.1	86.9	54.4
LSTM w/ aux LM	69.1	30.3	90.5	83.2	71.8	68.1	73.7	81.1	54.6

GPTv2 [32]

- Many NLP tasks can be formulated as the problem of maximizing $\Pr(\text{output}|\text{input}, \text{task})$
  - Translation: $\Pr(\text{french text}|\text{en text}, \text{translation})$
  - Question answering: $\Pr(\text{answer}|\text{question}, \text{qa})$
  - Reading comprehension: $\Pr(\text{answer}|\text{document}, \text{question}, \text{reading})$
- GPTv3 [4]: ***few-shot*** & ***in-context*** learning
- GPTv3.5 [29]: ***alignment*** using (supervised) instruction tuning & reinforcement learning from human feedback (RLHF)
- GPT4: mixture of experts

GPTv3 [4]

- The model learns from few shots (examples), even ***in context***
  - Enables ***prompting*** techniques for downstream tasks
  - E.g.,few shots, chain of thought (CoT), or simply "Let's work this out step-by-step to ensure the answer is correct"

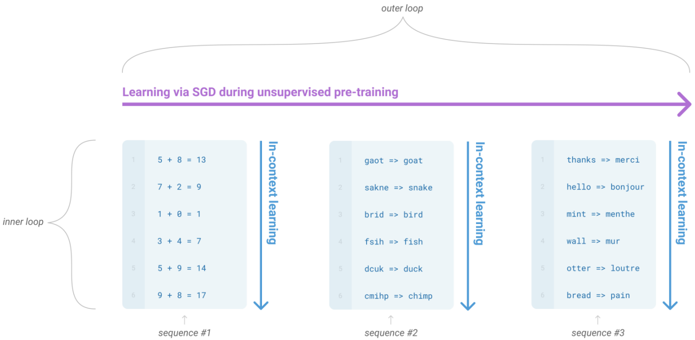
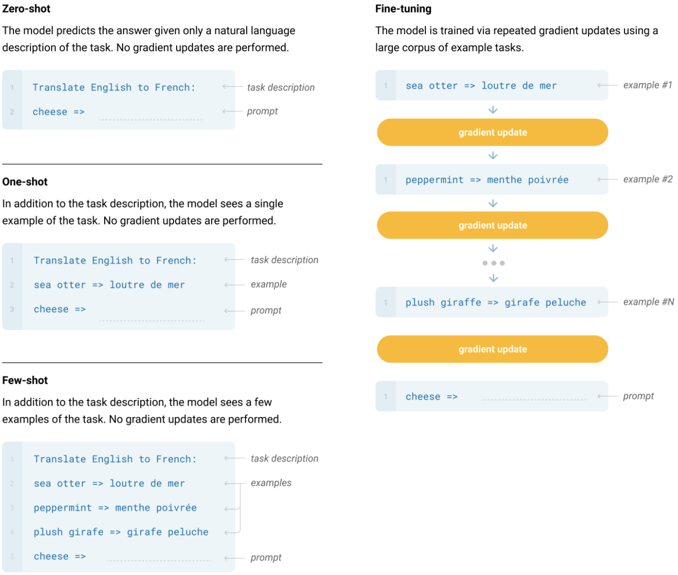


Figure 1.1: Language model meta-learning. During unsupervised pre-training, a language model develops a broad set of skills and pattern recognition abilities. It then uses these abilities at inference time to rapidly adapt to or recognize the desired task. We use the term "in-context learning" to describe the inner loop of this process, which occurs within the forward-pass upon each sequence. The sequences in this diagram are not intended to be representative of the data a model would see during pre-training, but are intended to show that there are sometimes repeated sub-tasks embedded within a single sequence.

Zero/Few Shot Prompting vs. Fine-tuning



Performance vs. Model Size

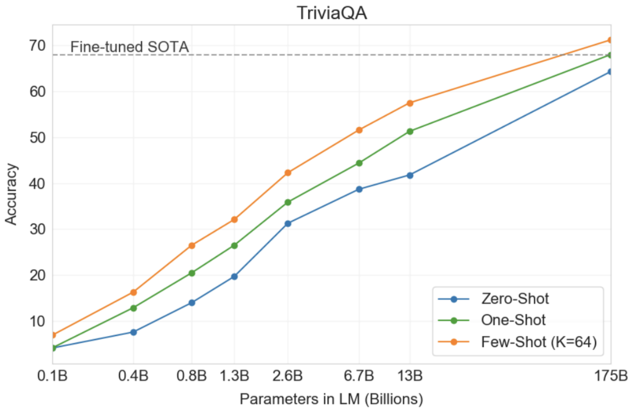
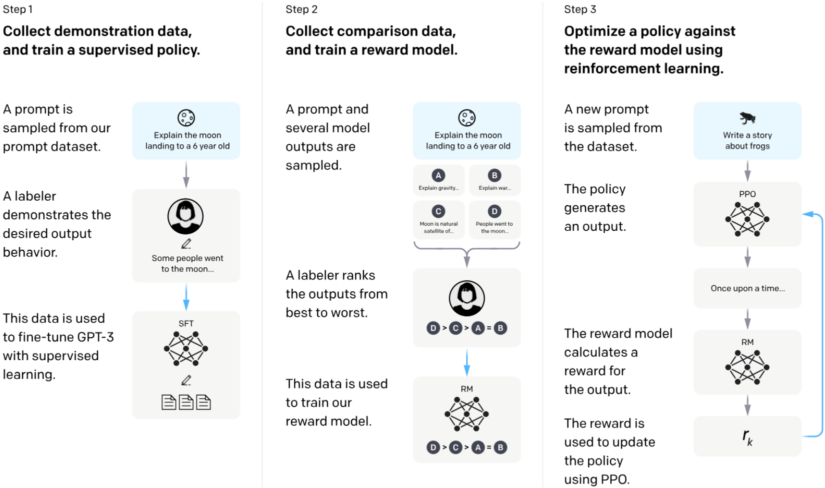


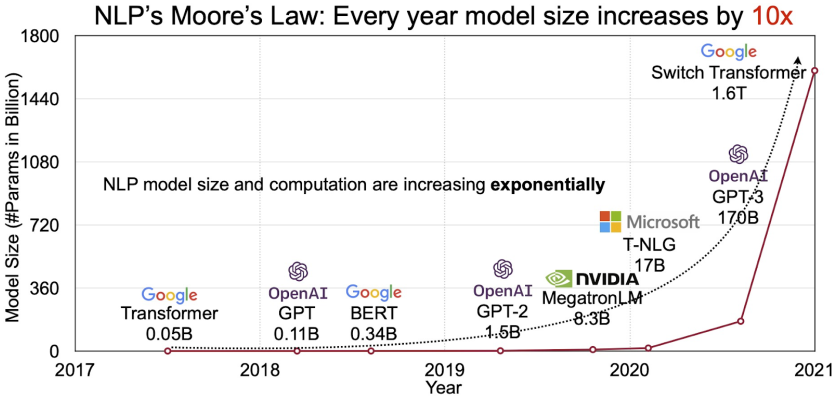
Figure 3.3: On TriviaQA GPT3's performance grows smoothly with model size, suggesting that language models continue to absorb knowledge as their capacity increases. One-shot and few-shot performance make significant gains over zero-shot behavior, matching and exceeding the performance of the SOTA fine-tuned open-domain model, RAG [LPP+20]

GPTv3.5 [29]

- From GPT to "ChatGPT" through ***alignment***
  - Supervised (multi-task) instruction tuning
  - Reinforcement learning from human feedback (RLHF)



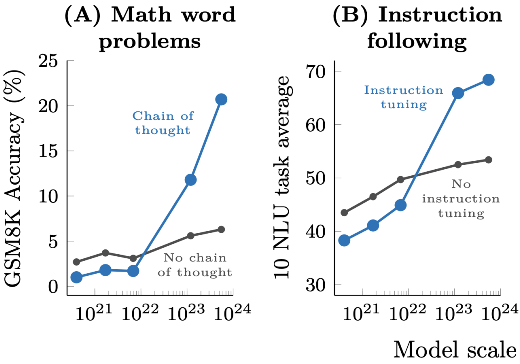
Sizes of Large Language Models (LLMs)



- Training costs [42]:
  - 110M params: \$2.5k–\$50k
  - 340M params: \$10k–\$200k
  - 1.5B param: \$80k–\$1.6m

Size Does Matter!

- Emerging abilities of LLMs [48]



- A balance: 70B parameters + 1.4T training tokens [11]

GPT Variants

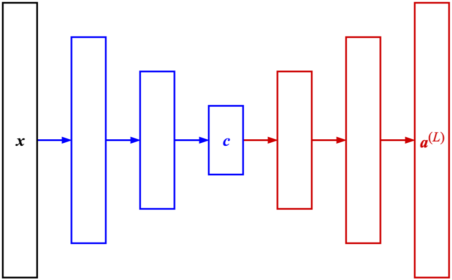
- Low-rank adaptation (LoRA) [12]: to efficiently fine-tune LLMs
- WebGPT [28]: GPT that can search the web
- Retrieval-augmented generation (RAG) [21]: GPT to query external knowledge (vector) base
- GPTs that can reflect on their answers [2, 27]

Outline

1. Unsupervised Learning
   - Text Models
   - Image Models
2. ChatGPT
3. **Autoencoders (AE)**
   - Manifold Learning*
4. Variational Autoencoders (VAE)
5. Flow-based Models
6. Diffusion Models
7. Generative Adversarial Networks*
   - Basic Architecture
   - Challenges
   - More GANs

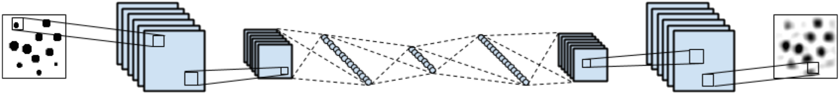
Autoencoders (AE)

- ***Encoder***: to learn a low dimensional representation $\mathbf{c}$ (called *code*) of input $\mathbf{x}$
- ***Decoder***: to reconstruct $\mathbf{x}$ from $\mathbf{c}$
- Objective: $\arg\max_{\Theta} \log \mathrm{P}(\mathbb{X} \,|\, \Theta) = \arg\max_{\Theta} \sum_i \log \mathrm{P}(\boldsymbol{x}^{(i)} \,|\, \Theta)$
- Assuming that $\mathbf{x} \sim \mathcal{N}(\mu, \cdot)$, we have linear output units $\boldsymbol{a}^{(L)} = \boldsymbol{z}^{(L)} = \hat{\mu}$
  - $\log \mathrm{P}(\boldsymbol{x}^{(i)} \,|\, \Theta) \propto -\|\boldsymbol{x}^{(i)} - \boldsymbol{a}^{(i,L)}\|^2$
  - $\arg\max_{\Theta} \sum_i \log \mathrm{P}(\boldsymbol{x}^{(i)} \,|\, \Theta) = \arg\min_{\Theta} \sum_i \|\boldsymbol{x}^{(i)} - \boldsymbol{a}^{(i,L)}\|^2$ (minimizing reconstruct error)



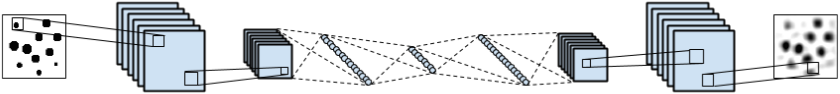
Convolutional Autoencoders

- Convolution + deconvolution layers:



Convolutional Autoencoders

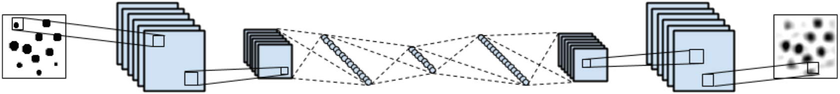
- Convolution + deconvolution layers:



- Decoder is a simplified DeconvNet [49] trained from scratch:

Convolutional Autoencoders

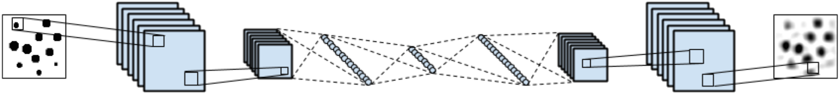
- Convolution + deconvolution layers:



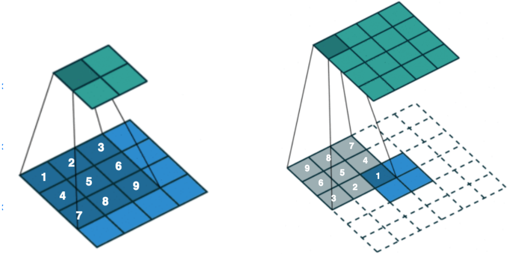
- Decoder is a simplified DeconvNet [49] trained from scratch:
  - Uppooling → upsampling (no need to remember max positions)

Convolutional Autoencoders

- Convolution + deconvolution layers:

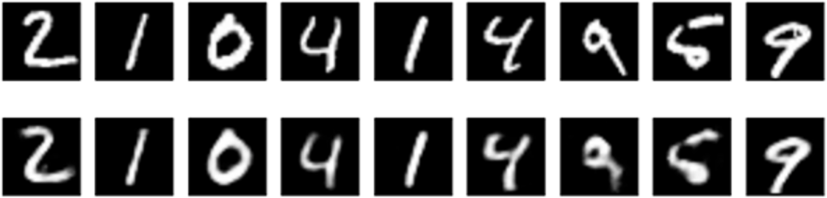


- Decoder is a simplified DeconvNet [49] trained from scratch:
  - Uppooling → upsampling (no need to remember max positions)
  - Deconvolution → convolution



Codes & Reconstructed $\mathbf{x}$

- A 32-bit code can roughly represents a 32×32 (1024 dimensional) MNIST image



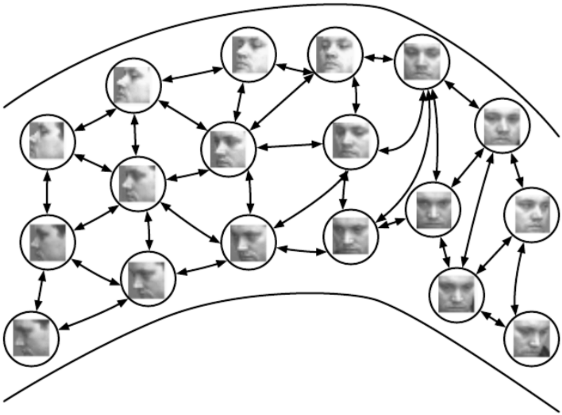
Outline

Manifolds I

- In many applications, data concentrate around one or more low-dimensional ***manifolds***

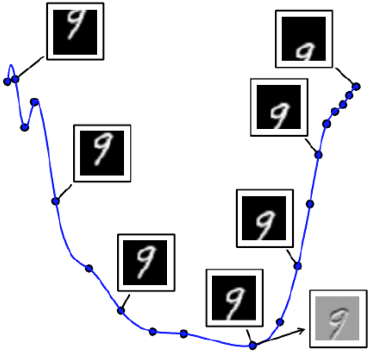
Manifolds I

- In many applications, data concentrate around one or more low-dimensional ***manifolds***
- A manifold is a topological space that are ***linear locally***



Manifolds II

- For each point $\boldsymbol{x}$ on a manifold, we have its ***tangent space*** spanned by ***tangent vectors***
  - Local directions specify how one can change $\boldsymbol{x}$ infinitesimally while staying on the manifold



Learning Manifolds I

- How to make $\mathbf{c}$ produced by autoencoders denote a ***coordinate*** of a dimensional manifold?

Learning Manifolds I

- How to make $\mathbf{c}$ produced by autoencoders denote a ***coordinate*** of a dimensional manifold?
- Contractive autoencoder [35]: regularizes the code $\mathbf{c}$ such that it is invariant to local changes of $\mathbf{x}$:

$$\Omega(\mathbf{c}) = \sum_i \left\| \frac{\partial \boldsymbol{c}^{(i)}}{\partial \boldsymbol{x}^{(i)}} \right\|_F^2$$

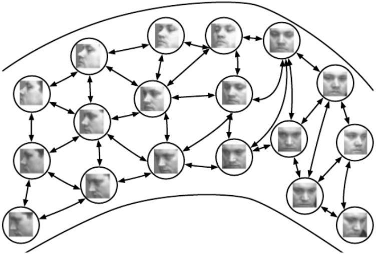
  - $\partial \boldsymbol{c}^{(i)} / \partial \boldsymbol{x}^{(i)}$ is a Jacobian matrix

Learning Manifolds I

- How to make $\mathbf{c}$ produced by autoencoders denote a ***coordinate*** of a dimensional manifold?
- Contractive autoencoder [35]: regularizes the code $\mathbf{c}$ such that it is invariant to local changes of $\mathbf{x}$:

$$\Omega(\mathbf{c}) = \sum_i \left\| \frac{\partial \boldsymbol{c}^{(i)}}{\partial \boldsymbol{x}^{(i)}} \right\|_F^2$$

  - $\partial \boldsymbol{c}^{(i)} / \partial \boldsymbol{x}^{(i)}$ is a Jacobian matrix
- Hence, $\mathbf{c}$ represents only the variations needed to reconstruct $\mathbf{x}$
  - I.e., $\mathbf{c}$ changes most along tangent vectors



Learning Manifolds II

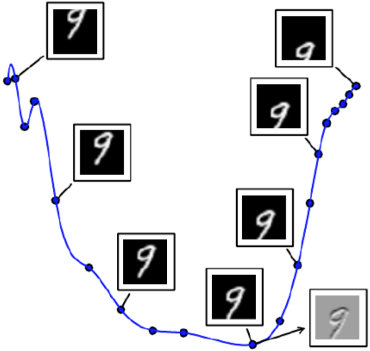
- In practice, it is easier to train a denoising autoencoder [47]:

Learning Manifolds II

- In practice, it is easier to train a denoising autoencoder [47]:
  - Encoder: to encode $\mathbf{x}$ ***with*** random noises
  - Decoder: to reconstruct $\mathbf{x}$ ***without*** noises

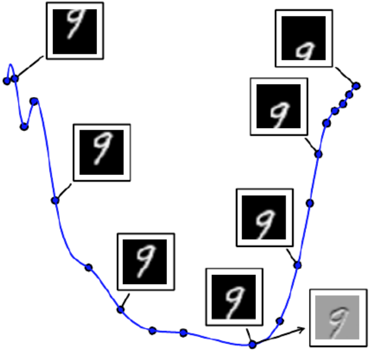
Getting Tangent Vectors I

- The code $\mathbf{c}$ represents a coordinate on a low dimensional manifold
  - E.g., the blue line
- How to get the tangent vectors of a given $\boldsymbol{c}$?



Getting Tangent Vectors II

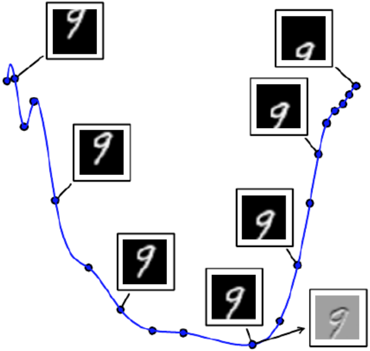
- Recall: directions in the input space that ***changes c most*** should be tangent vectors



Getting Tangent Vectors II

- Recall: directions in the input space that ***changes c most*** should be tangent vectors

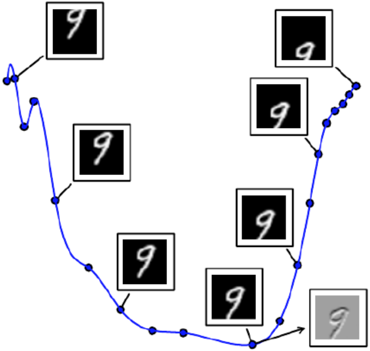
- Given a point $\boldsymbol{x}$, let $\boldsymbol{c}$ be the code of $\boldsymbol{x}$ and $\boldsymbol{J}(\boldsymbol{x}) = \frac{\partial \boldsymbol{c}}{\partial \boldsymbol{x}}$ be the Jacobian matrix of $\boldsymbol{c}$ at $\boldsymbol{x}$



Getting Tangent Vectors II

- Recall: directions in the input space that ***changes c most*** should be tangent vectors

- Given a point $\boldsymbol{x}$, let $\boldsymbol{c}$ be the code of $\boldsymbol{x}$ and $\boldsymbol{J}(\boldsymbol{x}) = \frac{\partial \boldsymbol{c}}{\partial \boldsymbol{x}}$ be the Jacobian matrix of $\boldsymbol{c}$ at $\boldsymbol{x}$
  - $\boldsymbol{J}(\boldsymbol{x})$ summarizes how $\boldsymbol{c}$ changes in terms of $\boldsymbol{x}$

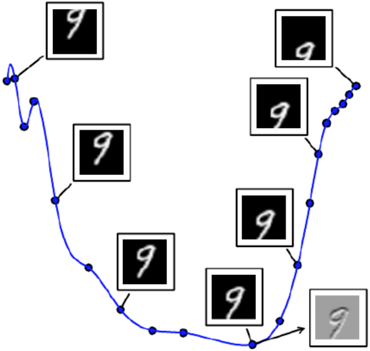


Getting Tangent Vectors II

- Recall: directions in the input space that ***changes c most*** should be tangent vectors

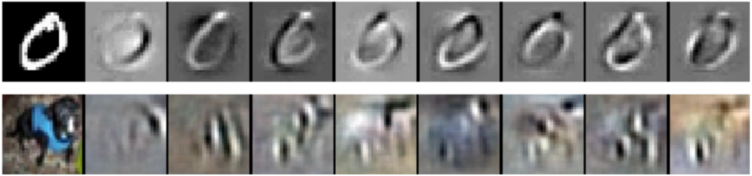
- Given a point $\boldsymbol{x}$, let $\boldsymbol{c}$ be the code of $\boldsymbol{x}$ and $\boldsymbol{J}(\boldsymbol{x}) = \frac{\partial \boldsymbol{c}}{\partial \boldsymbol{x}}$ be the Jacobian matrix of $\boldsymbol{c}$ at $\boldsymbol{x}$
  - $\boldsymbol{J}(\boldsymbol{x})$ summarizes how $\boldsymbol{c}$ changes in terms of $\boldsymbol{x}$

1. Decompose $\boldsymbol{J}(\boldsymbol{x})$ using SVD such that $\boldsymbol{J}(\boldsymbol{x}) = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top$
2. Let tangent vectors be ***rows of $\boldsymbol{V}$ corresponding to the largest singular values in $\boldsymbol{D}$***



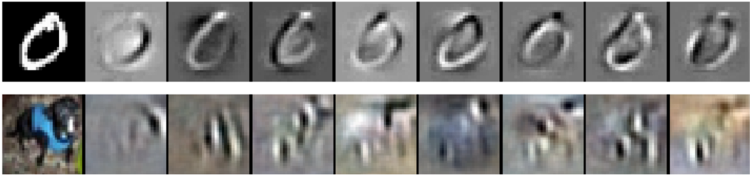
Getting Tangent Vectors III

- In practice, $\boldsymbol{J}(\boldsymbol{x})$ usually has few large singular values
- Tangent vectors found by contractive/denoising autoencoders:



Getting Tangent Vectors III

- In practice, $\boldsymbol{J}(\boldsymbol{x})$ usually has few large singular values
- Tangent vectors found by contractive/denoising autoencoders:

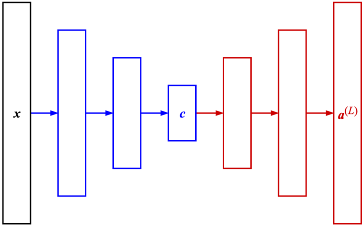


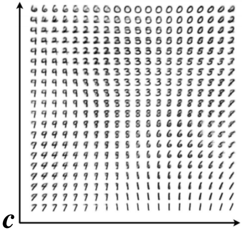
- Can be used by Tangent Prop [43]:
- Let $\{\boldsymbol{v}^{(i,j)}\}_j$ be tangent vectors of each example $\boldsymbol{x}^{(i)}$
- Trains an NN classifier f with cost penalty: $\Omega[f] = \sum_{i,j} \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(i)})^\top \boldsymbol{v}^{(i,j)}$
  - Points in the same manifold share the same label

Outline

Problems of Autoencoder I

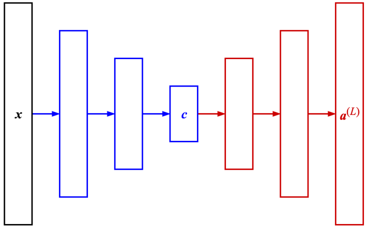
- Ideally, the decoder of an autoencoder can be used to generate images even with ***synthetic codes***

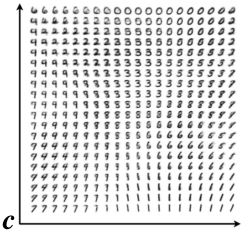




Problems of Autoencoder I

- Ideally, the decoder of an autoencoder can be used to generate images even with ***synthetic codes***





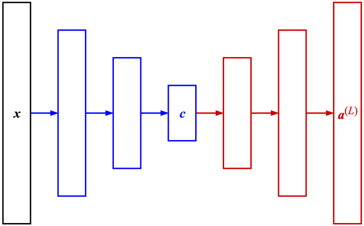
- In reality, the learnt $\mathbf{c}$ in code space has many "holes" that fail to map to images
  - More complex image patterns
  - Training data are never enough

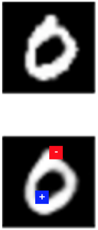
Problems of Autoencoder II

- Blurry images: the objective

$$\arg\max_{\Theta} \sum_i \log \mathrm{P}(\boldsymbol{x}^{(i)} \mid \Theta) = \arg\min_{\Theta} \sum_i \|\boldsymbol{x}^{(i)} - \boldsymbol{a}^{(i,L)}\|^2$$

does not panelize Gaussian pixel noises in $\boldsymbol{a}^{(i,L)}$





Variational Autoencoders (VAE) [15]

- Encoder $f(\cdot;\Theta_f)$: maps each sample of $\mathbf{x}$ (i.e., $\boldsymbol{x}^{(i)} \in \mathbb{X}$) to an ***axis-aligned normal distribution*** $\mathcal{N}(\mu, \sigma)$
  - Each code dimension is independent with each other
  - $f(\boldsymbol{x}) = (\mu, \sigma)$
- Decoder $g(\cdot;\Theta_g)$: same as that of AE
- How to minimize the objective $\arg\max_{\Theta_f,\Theta_g} \log \mathrm{P}(\mathbb{X} \mid \Theta_f, \Theta_g))$?

VAE Objective

- Objective:
  $\arg\max_{\Theta_f,\Theta_g} \log \mathrm{P}(\mathbb{X} \mid \Theta_f, \Theta_g) = \arg\max_{\Theta_f,\Theta_g} \sum_i \log \mathrm{P}(\boldsymbol{x}^{(i)} \mid \Theta_f, \Theta_g)$
- Considering $\log \mathrm{P}(\boldsymbol{x})$ for any sample $\boldsymbol{x}$, we have

$$
\begin{aligned}
\log \mathrm{P}(\boldsymbol{x}) &= \int_{\boldsymbol{c}} \mathrm{Q}(\boldsymbol{c}|\boldsymbol{x}) \log \mathrm{P}(\boldsymbol{x})\, d\boldsymbol{c} \quad \text{// Q can be any distribution} \\
&= \int_{\boldsymbol{c}} \mathrm{Q}(\boldsymbol{c}|\boldsymbol{x}) \log \left(\frac{\mathrm{P}(\boldsymbol{c},\boldsymbol{x})}{\mathrm{P}(\boldsymbol{c}|\boldsymbol{x})} \right) d\boldsymbol{c} = \int_{\boldsymbol{c}} \mathrm{Q}(\boldsymbol{c}|\boldsymbol{x}) \log \left(\frac{\mathrm{P}(\boldsymbol{c},\boldsymbol{x})}{\mathrm{Q}(\boldsymbol{c}|\boldsymbol{x})} \frac{\mathrm{Q}(\boldsymbol{c}|\boldsymbol{x})}{\mathrm{P}(\boldsymbol{c}|\boldsymbol{x})} \right) d\boldsymbol{c} \\
&= \int_{\boldsymbol{c}} \mathrm{Q}(\boldsymbol{c}|\boldsymbol{x}) \log \left(\frac{\mathrm{P}(\boldsymbol{c},\boldsymbol{x})}{\mathrm{Q}(\boldsymbol{c}|\boldsymbol{x})} \right) d\boldsymbol{c} + \int_{\boldsymbol{c}} \mathrm{Q}(\boldsymbol{c}|\boldsymbol{x}) \log \left(\frac{\mathrm{Q}(\boldsymbol{c}|\boldsymbol{x})}{\mathrm{P}(\boldsymbol{c}|\boldsymbol{x})} \right) d\boldsymbol{c} \\
&= \int_{\boldsymbol{c}} \mathrm{Q}(\boldsymbol{c}|\boldsymbol{x}) \log \left(\frac{\mathrm{P}(\boldsymbol{c},\boldsymbol{x})}{\mathrm{Q}(\boldsymbol{c}|\boldsymbol{x})} \right) d\boldsymbol{c} + \mathrm{D}_{\mathrm{KL}} \left(\mathrm{Q}(\mathbf{c}|\boldsymbol{x}) \| \mathrm{P}(\mathbf{c}|\boldsymbol{x}) \right) \\
&\geq \int_{\boldsymbol{c}} \mathrm{Q}(\boldsymbol{c}|\boldsymbol{x}) \log \left(\frac{\mathrm{P}(\boldsymbol{c},\boldsymbol{x})}{\mathrm{Q}(\boldsymbol{c}|\boldsymbol{x})} \right) d\boldsymbol{c} \quad \text{// lower bound} \\
&= \int_{\boldsymbol{c}} \mathrm{Q}(\boldsymbol{c}|\boldsymbol{x}) \log \left(\frac{\mathrm{P}(\boldsymbol{x}|\boldsymbol{c})\mathrm{P}(\boldsymbol{c})}{\mathrm{Q}(\boldsymbol{c}|\boldsymbol{x})} \right) d\boldsymbol{c}
\end{aligned}
$$

- VAE lets $\mathrm{Q}(\cdot|\boldsymbol{x}) = \mathcal{N}(f(\boldsymbol{x};\Theta_f))$ and $\mathrm{P}(\cdot|\boldsymbol{c}) = \mathcal{N}(g(\boldsymbol{c};\Theta_g))$
  - So $\mathrm{Q}(\boldsymbol{c}|\boldsymbol{x}) = \mathrm{Q}(\boldsymbol{c}|\boldsymbol{x},\Theta_f)$ and $\mathrm{P}(\boldsymbol{x}|\boldsymbol{c}) = \mathrm{P}(\boldsymbol{x}|\boldsymbol{c},\Theta_g)$
- New objective: finds Θ_f and Θ_g that maximize the lower bound
  - Not necessarily maximize $\log \mathrm{P}(\boldsymbol{x} \mid \Theta_f, \Theta_g)$

Maximizing Lower Bound

$$
\begin{aligned}
&\int_{\boldsymbol{c}} \mathrm{Q}(\boldsymbol{c}|\boldsymbol{x}) \log\left(\frac{\mathrm{P}(\boldsymbol{x}|\boldsymbol{c})\mathrm{P}(\boldsymbol{c})}{\mathrm{Q}(\boldsymbol{c}|\boldsymbol{x})}\right) d\boldsymbol{c} \\
&\quad = \int_{\boldsymbol{c}} \mathrm{Q}(\boldsymbol{c}|\boldsymbol{x},\Theta_f) \log\left(\frac{\mathrm{P}(\boldsymbol{x}|\boldsymbol{c},\Theta_g)\mathrm{P}(\boldsymbol{c})}{\mathrm{Q}(\boldsymbol{c}|\boldsymbol{x},\Theta_f)}\right) d\boldsymbol{c} \\
&\quad = \int_{\boldsymbol{c}} \mathrm{Q}(\boldsymbol{c}|\boldsymbol{x},\Theta_f) \log\left(\frac{\mathrm{P}(\boldsymbol{c})}{Q(\boldsymbol{c}|\boldsymbol{x},\Theta_f)}\right) d\boldsymbol{c} + \int_{\boldsymbol{c}} \mathrm{Q}(\boldsymbol{c}|\boldsymbol{x},\Theta_f) \log \mathrm{P}(\boldsymbol{x}|\boldsymbol{c},\Theta_g) d\boldsymbol{c} \\
&\quad = -\mathrm{D}_{\mathrm{KL}}\left(\mathrm{Q}(\mathbf{c}|\boldsymbol{x},\Theta_f)\|\mathrm{P}(\mathbf{c})\right) + \mathrm{E}_{(\mathbf{c}|\boldsymbol{x},\Theta_f)\sim\mathrm{Q}}\left[\log \mathrm{P}(\boldsymbol{x}|\mathbf{c},\Theta_g)\right]
\end{aligned}
$$

- For $\mathrm{P}(\mathbf{c}) = \mathcal{N}(\mathbf{0},\boldsymbol{I})$:
- To minimize the first term, the encoder $f(\boldsymbol{x};\Theta_f) = (\mu,\sigma)$ has a loss term [15]

$$\exp(\sigma) - (\mathbf{1}+\sigma) + \|\mu\|^2$$

- To maximize the second term, the decoder
  1. Samples $\boldsymbol{c}$ from $\mathcal{N}(\mu,\sigma)$ to get $g(\boldsymbol{c};\Theta_g)$, the mean of output $\mathcal{N}$
  2. Minimizes a loss term $\|\boldsymbol{x}-\boldsymbol{a}^{(L)}\|^2$ as in AE

Outline

1. Unsupervised Learning
   - Text Models
   - Image Models
2. ChatGPT
3. Autoencoders (AE)
   - Manifold Learning*
4. Variational Autoencoders (VAE)
5. **Flow-based Models**
6. Diffusion Models
7. Generative Adversarial Networks*
   - Basic Architecture
   - Challenges
   - More GANs

Problems of VAE

- Only maximize a lower bound of the likelihood $\mathrm{P}(\mathbb{X}|\Theta_f, \Theta_g)$
  - Θ_f and Θ_g are encoder and decoder weights, respectively
- Still blurry images
  - The decoder's loss is the same with that of AE

Flow-based Models

- Idea: let the decoder $g(\cdot;\Theta_g)$ be a ***deterministic invertible*** function
  - Given $\mathbf{c} \sim \mathcal{N}(\mathbf{0},\boldsymbol{I})$, we have $\mathbf{x} = g(\mathbf{c};\Theta_g)$ of complex distribution
  - Conversely, given $\mathbf{x}$, we have $\mathbf{c} = g^{-1}(\mathbf{x};\Theta_g)$
- The likelihood can be maximize directly:

$$\begin{aligned}
&\arg\max_g \log \mathrm{P}_g(\mathbb{X}) \\
&\quad = \arg\max_g \textstyle\sum_i \log \mathrm{P}_{\mathbf{x}}(g(\boldsymbol{c}^{(i)})) \\
&\quad = \arg\max_g \textstyle\sum_i \log \left[\mathrm{P}_{\mathbf{c}}(g^{-1}(\boldsymbol{x}^{(i)}))\,|\det\left(\boldsymbol{J}(g^{-1})(\boldsymbol{x}^{(i)})\right)|\right] \\
&\quad = \arg\max_g \textstyle\sum_i \log \mathrm{P}_{\mathbf{c}}(g^{-1}(\boldsymbol{x}^{(i)})) + \log|\det\left(\boldsymbol{J}(g^{-1})(\boldsymbol{x}^{(i)})\right)|
\end{aligned}$$

- First term: finds g (Θ_g) that maps all $\boldsymbol{x}^{(i)}$ to 0
- Second term: prevents g (Θ_g) from mapping all $\boldsymbol{x}^{(i)}$ to 0
  - $\log|\det(\boldsymbol{O})| = -\inf$

Flow-based Models

- Idea: let the decoder $g(\cdot;\Theta_g)$ be a ***deterministic invertible*** function
  - Given $\mathbf{c} \sim \mathcal{N}(\mathbf{0},\boldsymbol{I})$, we have $\mathbf{x} = g(\mathbf{c};\Theta_g)$ of complex distribution
  - Conversely, given $\mathbf{x}$, we have $\mathbf{c} = g^{-1}(\mathbf{x};\Theta_g)$
- The likelihood can be maximize directly:

$$
\begin{aligned}
&\arg\max_g \log \mathrm{P}_g(\mathbb{X}) \\
&\quad = \arg\max_g \textstyle\sum_i \log \mathrm{P}_{\mathbf{x}}(g(\boldsymbol{c}^{(i)})) \\
&\quad = \arg\max_g \textstyle\sum_i \log \left[\mathrm{P}_{\mathbf{c}}(g^{-1}(\boldsymbol{x}^{(i)}))\,|\det\left(\boldsymbol{J}(g^{-1})(\boldsymbol{x}^{(i)})\right)|\right] \\
&\quad = \arg\max_g \textstyle\sum_i \log \mathrm{P}_{\mathbf{c}}(g^{-1}(\boldsymbol{x}^{(i)})) + \log|\det\left(\boldsymbol{J}(g^{-1})(\boldsymbol{x}^{(i)})\right)|
\end{aligned}
$$

- First term: finds g (Θ_g) that maps all $\boldsymbol{x}^{(i)}$ to 0
- Second term: prevents g (Θ_g) from mapping all $\boldsymbol{x}^{(i)}$ to 0
  - $\log|\det(\boldsymbol{O})| = -\inf$
- But how to ensure the followings during training?
  - g is invertible
  - $\det\left(\boldsymbol{J}(g^{-1})(\cdot)\right)$ can be easily computed

Ensuring Invertibility

- Glow [16]: make g^{-1} an 1×1 convolution layer
  - $\mathbf{x}_{i,j,:} = \begin{bmatrix} \mathrm{x}_{i,j,1} \\ \mathrm{x}_{i,j,2} \\ \mathrm{x}_{i,j,3} \end{bmatrix}, \quad g^{-1} = \boldsymbol{W}_{3\times 3},$
$$g^{-1}(\mathbf{x}_{i,j,:}) = \boldsymbol{W}_{3\times 3} \begin{bmatrix} \mathrm{x}_{i,j,1} \\ \mathrm{x}_{i,j,2} \\ \mathrm{x}_{i,j,3} \end{bmatrix} = \begin{bmatrix} \mathrm{c}_{i,j,1} \\ \mathrm{c}_{i,j,2} \\ \mathrm{c}_{i,j,3} \end{bmatrix}$$
- At training time, initialize $\boldsymbol{W}_{3\times 3}$ as an invertible matrix
  - $g^{-1} = \boldsymbol{W}_{3\times 3}$ is likely to be invertible after SGD updates
  - Determinant is easy to compute: $\det\left(\boldsymbol{J}(g^{-1})(\boldsymbol{x}^{(i)})\right) = \det(\boldsymbol{W}_{3\times 3})^{W\times H}$
- At inference time, use $g = \boldsymbol{W}_{3\times 3}^{-1}$ to generate images

Ensuring Invertibility

- Glow [16]: make g^{-1} an 1×1 convolution layer
  - $\mathbf{x}_{i,j,:} = \begin{bmatrix} \mathrm{x}_{i,j,1} \\ \mathrm{x}_{i,j,2} \\ \mathrm{x}_{i,j,3} \end{bmatrix}, \quad g^{-1} = \boldsymbol{W}_{3\times 3},$

    $$g^{-1}(\mathbf{x}_{i,j,:}) = \boldsymbol{W}_{3\times 3} \begin{bmatrix} \mathrm{x}_{i,j,1} \\ \mathrm{x}_{i,j,2} \\ \mathrm{x}_{i,j,3} \end{bmatrix} = \begin{bmatrix} \mathrm{c}_{i,j,1} \\ \mathrm{c}_{i,j,2} \\ \mathrm{c}_{i,j,3} \end{bmatrix}$$
- At training time, initialize $\boldsymbol{W}_{3\times 3}$ as an invertible matrix
  - $g^{-1} = \boldsymbol{W}_{3\times 3}$ is likely to be invertible after SGD updates
  - Determinant is easy to compute: $\det\left(\boldsymbol{J}(g^{-1})(\boldsymbol{x}^{(i)})\right) = \det(\boldsymbol{W}_{3\times 3})^{W\times H}$
- At inference time, use $g = \boldsymbol{W}_{3\times 3}^{-1}$ to generate images
- Problem: g has limited expressiveness

Step-wise Generation

- Cascade multiple invertible g to have a more complex one:

$$\mathbf{x} = g^{(T)}(\cdots g^{(2)}(g^{(1)}(\mathbf{c})))$$

Step-wise Generation

- Cascade multiple invertible g to have a more complex one:

$$\mathbf{x} = g^{(T)}(\cdots g^{(2)}(g^{(1)}(\mathbf{c})))$$

- Result: sharp images

Outline

1. **Unsupervised Learning**
   - Text Models
   - Image Models
2. **ChatGPT**
3. **Autoencoders (AE)**
   - Manifold Learning*
4. **Variational Autoencoders (VAE)**
5. **Flow-based Models**
6. **Diffusion Models**
7. **Generative Adversarial Networks***
   - Basic Architecture
   - Challenges
   - More GANs

Problems of Flow-based Models

- Limited model expressiveness
- Slow training and inference
    - #steps can be large

Denoising Diffusion Probabilistic Models (DDPM) [10]

- Borrow some good ideas from previous works
  - Probabilistic formulation of VAE that models encoder in the objective
  - Step-wise encoding/decoding in generative flows
- But, unlike flows, the encoding steps
  - Are ***predefined***; no parameter to learn
  - Can be simplified to ***one encoding step***

Encoding

- Predefined encoding functions $f^{(t)}(\cdot;\beta^{(t)})$, $t=1,\cdots,T$:
  - $\beta^{(t)}$, $\forall t$, are hyperparameters; no learning needed
- Let each sample $\boldsymbol{x}=\boldsymbol{x}^{(0)}$
- $\mathbf{x}^{(1)}=f^{(1)}(\boldsymbol{x}^{(0)};\beta^{(1)})=\sqrt{1-\beta^{(1)}}\boldsymbol{x}^{(0)}+\sqrt{\beta^{(1)}}\boldsymbol{\varepsilon}$, $\boldsymbol{\varepsilon}\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})$
  - So, $(\mathbf{x}^{(1)}|\mathbf{x}^{(0)}=\boldsymbol{x}^{(0)})\sim\mathcal{N}(\sqrt{1-\beta^{(1)}}\boldsymbol{x}^{(0)},\beta^{(1)}\boldsymbol{I})$

Encoding

- Predefined encoding functions $f^{(t)}(\cdot;\beta^{(t)})$, $t = 1, \cdots, T$:
  - $\beta^{(t)}$, $\forall t$, are hyperparameters; no learning needed
- Let each sample $\boldsymbol{x} = \boldsymbol{x}^{(0)}$
- $\mathbf{x}^{(1)} = f^{(1)}(\boldsymbol{x}^{(0)};\beta^{(1)}) = \sqrt{1-\beta^{(1)}}\boldsymbol{x}^{(0)} + \sqrt{\beta^{(1)}}\varepsilon$, $\varepsilon \sim \mathcal{N}(\mathbf{0},\boldsymbol{I})$
  - So, $(\mathbf{x}^{(1)}|\mathbf{x}^{(0)} = \boldsymbol{x}^{(0)}) \sim \mathcal{N}(\sqrt{1-\beta^{(1)}}\boldsymbol{x}^{(0)}, \beta^{(1)}\boldsymbol{I})$
- $$\begin{aligned} \mathbf{x}^{(2)} &= f^{(2)}(\boldsymbol{x}^{(1)};\beta^{(t)}) = \sqrt{1-\beta^{(2)}}\boldsymbol{x}^{(1)} + \sqrt{\beta^{(2)}}\varepsilon \\ &= \sqrt{1-\beta^{(2)}}\sqrt{1-\beta^{(1)}}\boldsymbol{x}^{(0)} + \sqrt{1-(1-\beta^{(2)})(1-\beta^{(1)})}\varepsilon \\ &= \sqrt{\alpha^{(2)}\alpha^{(1)}}\boldsymbol{x}^{(0)} + \sqrt{1-\alpha^{(2)}\alpha^{(1)}}\varepsilon \end{aligned}$$
  - $\sqrt{1-\beta^{(1)}}\sqrt{\beta^{(2)}}\varepsilon + \sqrt{\beta^{(2)}}\varepsilon \sim \mathcal{N}(\mathbf{0},((1-\beta^{(1)})\beta^{(2)}+\beta^{(2)})\boldsymbol{I})$
  - Let $\alpha^{(t)} = 1-\beta^{(t)}$ (derived hyperparameter)
  - Only one ε is added

Encoding

- Predefined encoding functions $f^{(t)}(\cdot;\beta^{(t)})$, $t=1,\cdots,T$:
  - $\beta^{(t)}$, $\forall t$, are hyperparameters; no learning needed
- Let each sample $\boldsymbol{x}=\boldsymbol{x}^{(0)}$
- $\mathbf{x}^{(1)}=f^{(1)}(\boldsymbol{x}^{(0)};\beta^{(1)})=\sqrt{1-\beta^{(1)}}\boldsymbol{x}^{(0)}+\sqrt{\beta^{(1)}}\boldsymbol{\varepsilon}$, $\boldsymbol{\varepsilon}\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})$
  - So, $(\mathbf{x}^{(1)}|\mathbf{x}^{(0)}=\boldsymbol{x}^{(0)})\sim\mathcal{N}(\sqrt{1-\beta^{(1)}}\boldsymbol{x}^{(0)},\beta^{(1)}\boldsymbol{I})$
- $$\begin{aligned}\mathbf{x}^{(2)} &= f^{(2)}(\boldsymbol{x}^{(1)};\beta^{(t)})=\sqrt{1-\beta^{(2)}}\boldsymbol{x}^{(1)}+\sqrt{\beta^{(2)}}\boldsymbol{\varepsilon}\\ &= \sqrt{1-\beta^{(2)}}\sqrt{1-\beta^{(1)}}\boldsymbol{x}^{(0)}+\sqrt{1-(1-\beta^{(2)})(1-\beta^{(1)})}\boldsymbol{\varepsilon}\\ &= \sqrt{\alpha^{(2)}\alpha^{(1)}}\boldsymbol{x}^{(0)}+\sqrt{1-\alpha^{(2)}\alpha^{(1)}}\boldsymbol{\varepsilon}\end{aligned}$$
  - $\sqrt{1-\beta^{(1)}}\sqrt{\beta^{(2)}}\boldsymbol{\varepsilon}+\sqrt{\beta^{(2)}}\boldsymbol{\varepsilon}\sim\mathcal{N}(\mathbf{0},((1-\beta^{(1)})\beta^{(2)}+\beta^{(2)})\boldsymbol{I})$
  - Let $\alpha^{(t)}=1-\beta^{(t)}$ (derived hyperparameter)
  - Only one $\boldsymbol{\varepsilon}$ is added
- $\mathbf{x}^{(t)}=\sqrt{\bar{\alpha}^{(t)}}\boldsymbol{x}^{(t-1)}+\sqrt{1-\bar{\alpha}^{(t)}}\boldsymbol{\varepsilon}$
  - Let $\bar{\alpha}^{(t)}=\alpha^{(t)}\alpha^{(t-1)}\cdots\alpha^{(1)}$ (derived hyperparameter)

Objective

- As VAE, DDPM maximizes a lower bound of $\mathrm{P}(\mathbb{X})$
- VAE: for each $\boldsymbol{x}$, where $\mathrm{P}(\boldsymbol{x}) = \int_{\boldsymbol{c}} \mathrm{Q}(\boldsymbol{c}|\boldsymbol{x}) \log \mathrm{P}(\boldsymbol{x})\, d\boldsymbol{c}$, maximize:

$$\int_{\boldsymbol{c}} \mathrm{Q}(\boldsymbol{c}|\boldsymbol{x}) \log\left(\frac{\mathrm{P}(\boldsymbol{x},\boldsymbol{c})}{\mathrm{Q}(\boldsymbol{c}|\boldsymbol{x})}\right) d\boldsymbol{c} = \mathrm{E}_{\mathbf{c}|\boldsymbol{x}\sim\mathrm{Q}}\left[\log\left(\frac{\mathrm{P}(\boldsymbol{x},\mathbf{c})}{\mathrm{Q}(\mathbf{c}|\boldsymbol{x})}\right)\right] \leq \mathrm{P}(\boldsymbol{x})$$

Objective

- As VAE, DDPM maximizes a lower bound of $\mathrm{P}(\mathbb{X})$
- VAE: for each $\boldsymbol{x}$, where $\mathrm{P}(\boldsymbol{x}) = \int_{\boldsymbol{c}} \mathrm{Q}(\boldsymbol{c}|\boldsymbol{x}) \log \mathrm{P}(\boldsymbol{x})\, d\boldsymbol{c}$, maximize:

$$\int_{\boldsymbol{c}} \mathrm{Q}(\boldsymbol{c}|\boldsymbol{x}) \log\left(\frac{\mathrm{P}(\boldsymbol{x},\boldsymbol{c})}{\mathrm{Q}(\boldsymbol{c}|\boldsymbol{x})}\right) d\boldsymbol{c} = \mathrm{E}_{\mathbf{c}|\boldsymbol{x}\sim\mathrm{Q}}\left[\log\left(\frac{\mathrm{P}(\boldsymbol{x},\mathbf{c})}{\mathrm{Q}(\mathbf{c}|\boldsymbol{x})}\right)\right] \leq \mathrm{P}(\boldsymbol{x})$$

- DDPM: for each $\boldsymbol{x}$ where $\mathrm{P}(\boldsymbol{x}^{(0)}) = \int_{\boldsymbol{x}^{(1)},\cdots,\boldsymbol{x}^{(T)}} \mathrm{Q}(\boldsymbol{x}^{(1)},\cdots,\boldsymbol{x}^{(T)}|\boldsymbol{x}^{(0)}) \log \mathrm{P}(\boldsymbol{x}^{(0)})\, d\boldsymbol{x}^{(1)}\cdots d\boldsymbol{x}^{(T)}$, maximize:

$$\mathrm{E}_{(\mathbf{x}^{(1)},\cdots,\mathbf{x}^{(T)}|\boldsymbol{x}^{(0)})\sim\mathrm{Q}}\left[\log\left(\frac{\mathrm{P}(\boldsymbol{x}^{(0)},\mathbf{x}^{(1)},\cdots,\mathbf{x}^{(T)})}{\mathrm{Q}(\mathbf{x}^{(1)},\cdots,\mathbf{x}^{(T)}|\boldsymbol{x}^{(0)})}\right)\right],$$

  which can be simplified to [23]:

$$-\mathrm{D}_{\mathrm{KL}}\left(\mathrm{Q}(\mathbf{x}^{(T)}|\boldsymbol{x}^{(0)})\|\mathrm{P}(\mathbf{x}^{(T)})\right) + \mathrm{E}_{\mathbf{x}^{(1)}|\boldsymbol{x}^{(0)}\sim\mathrm{Q}}\left[\log \mathrm{P}(\boldsymbol{x}^{(0)}|\mathbf{x}^{(1)})\right] + \\ -\sum_{t=2}^{T} \mathrm{E}_{\mathbf{x}^{(t)}|\boldsymbol{x}^{(0)}\sim\mathrm{Q}}\left[\mathrm{D}_{\mathrm{KL}}\left(\mathrm{Q}(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)},\boldsymbol{x}^{(0)})\|\mathrm{P}(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)})\right)\right]$$

  - First term controlled by encoding process (predefined)
  - Second & third term controlled by denoising process (learnable)

Denoising I

- For simplicity, we focus on maximizing the third term:

$$-\sum_{t=2}^{T} \mathrm{E}_{\mathbf{x}^{(t)}|\boldsymbol{x}^{(0)}\sim \mathrm{Q}}\left[\mathrm{D}_{\mathrm{KL}}\left(\mathrm{Q}(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)},\boldsymbol{x}^{(0)})\|\mathrm{P}(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)})\right)\right]$$

- Goal: for each observed $\boldsymbol{x}^{(t)}$, minimize

$$\mathrm{D}_{\mathrm{KL}}\left(\mathrm{Q}(\mathbf{x}^{(t-1)}|\color{red}{\boldsymbol{x}^{(t)}},\boldsymbol{x}^{(0)})\|\mathrm{P}(\mathbf{x}^{(t-1)}|\color{red}{\boldsymbol{x}^{(t)}})\right)$$

- Note that

$$\begin{aligned}
\mathrm{Q}(\mathbf{x}^{(t-1)}|\boldsymbol{x}^{(t)},\boldsymbol{x}^{(0)}) &= \frac{\mathrm{Q}(\mathbf{x}^{(t-1)},\boldsymbol{x}^{(t)},\boldsymbol{x}^{(0)})}{\mathrm{Q}(\boldsymbol{x}^{(t)},\boldsymbol{x}^{(0)})} \\
&= \frac{\mathrm{Q}(\boldsymbol{x}^{(t)}|\mathbf{x}^{(t-1)})\mathrm{Q}(\mathbf{x}^{(t-1)}|\boldsymbol{x}^{(0)})\mathrm{Q}(\boldsymbol{x}^{(0)})}{\mathrm{Q}(\boldsymbol{x}^{(t)}|\boldsymbol{x}^{(0)})\mathrm{Q}(\boldsymbol{x}^{(0)})} \\
&= \frac{\mathrm{Q}(\boldsymbol{x}^{(t)}|\mathbf{x}^{(t-1)})\mathrm{Q}(\mathbf{x}^{(t-1)}|\boldsymbol{x}^{(0)})}{\mathrm{Q}(\boldsymbol{x}^{(t)}|\boldsymbol{x}^{(0)})}
\end{aligned}$$

- Since $\mathrm{Q}(\boldsymbol{x}^{(t)}|\mathbf{x}^{(t-1)})$ & $\mathrm{Q}(\mathbf{x}^{(t-1)}|\boldsymbol{x}^{(0)})$ are Gaussian, we have [23]:
$\mathrm{Q}(\mathbf{x}^{(t-1)}|\boldsymbol{x}^{(t)},\boldsymbol{x}^{(0)}) = \mathcal{N}\left(\frac{\sqrt{\alpha^{(t)}}(1-\alpha^{(t-1)})\boldsymbol{x}^{(t)}+\sqrt{\bar{\alpha}^{(t-1)}}\beta^{(t)}\boldsymbol{x}^{(0)}}{1-\bar{\alpha}^{(t)}}, \frac{1-\bar{\alpha}^{(t-1)}}{1-\bar{\alpha}^{(t)}}\beta^{(t)}\boldsymbol{I}\right)$

Denoising II

- Goal: for each observed $\boldsymbol{x}^{(t)}$, minimize

$$\mathrm{D_{KL}}\left(\mathrm{Q}(\mathbf{x}^{(t-1)}|\boldsymbol{x}^{(t)},\boldsymbol{x}^{(0)})\|\mathrm{P}(\mathbf{x}^{(t-1)}|\boldsymbol{x}^{(t)})\right),$$

  where $\mathrm{Q}(\cdots) = \mathscr{N}\left(\frac{\sqrt{\alpha^{(t)}}(1-\alpha^{(t-1)})\boldsymbol{x}^{(t)}+\sqrt{\bar{\alpha}^{(t-1)}}\beta^{(t)}\boldsymbol{x}^{(0)}}{1-\bar{\alpha}^{(t)}},\cdots\right)$ is ***fixed***
- DDPM finds Θ that move the mean of $\mathrm{P}(\mathbf{x}^{(t-1)}|\boldsymbol{x}^{(t)},\Theta)$ (also Gaussian) toward $\mathrm{Q}(\cdots)$'s mean:

$$\frac{\sqrt{\alpha^{(t)}}(1-\alpha^{(t-1)})\boldsymbol{x}^{(t)}+\sqrt{\bar{\alpha}^{(t-1)}}\beta^{(t)}\boldsymbol{x}^{(0)}}{1-\bar{\alpha}^{(t)}}$$

Noise Predictor

- Θ moves $\mathrm{P}(\mathbf{x}^{(t-1)}|\boldsymbol{x}^{(t)},\Theta)$'s mean toward

$$
\begin{aligned}
&\frac{\sqrt{\alpha^{(t)}}(1-\alpha^{(t-1)})\boldsymbol{x}^{(t)}+\sqrt{\bar{\alpha}^{(t-1)}}\beta^{(t)}\boldsymbol{x}^{(0)}}{1-\bar{\alpha}^{(t)}} \\
&= \frac{\sqrt{\alpha^{(t)}}(1-\alpha^{(t-1)})\boldsymbol{x}^{(t)}+\sqrt{\bar{\alpha}^{(t-1)}}\beta^{(t)}\frac{\boldsymbol{x}^{(t)}-\sqrt{1-\bar{\alpha}^{(t)}}\varepsilon}{\sqrt{\bar{\alpha}^{(t)}}}}{1-\bar{\alpha}^{(t)}} \\
&= \frac{1}{\sqrt{\alpha^{(t)}}}\left(\boldsymbol{x}^{(t)}-\frac{1-\alpha^{(t)}}{\sqrt{1-\bar{\alpha}^{(t)}}}\varepsilon\right)
\end{aligned}
$$

- What's its corresponding network?

Noise Predictor

- Θ moves $\mathrm{P}(\mathbf{x}^{(t-1)}|\boldsymbol{x}^{(t)},\Theta)$'s mean toward

$$\begin{aligned}
&\frac{\sqrt{\alpha^{(t)}}(1-\alpha^{(t-1)})\boldsymbol{x}^{(t)}+\sqrt{\bar{\alpha}^{(t-1)}}\beta^{(t)}\boldsymbol{x}^{(0)}}{1-\bar{\alpha}^{(t)}} \\
&= \frac{\sqrt{\alpha^{(t)}}(1-\alpha^{(t-1)})\boldsymbol{x}^{(t)}+\sqrt{\bar{\alpha}^{(t-1)}}\beta^{(t)}\frac{\boldsymbol{x}^{(t)}-\sqrt{1-\bar{\alpha}^{(t)}}\varepsilon}{\sqrt{\bar{\alpha}^{(t)}}}}{1-\bar{\alpha}^{(t)}} \\
&= \frac{1}{\sqrt{\alpha^{(t)}}}\left(\boldsymbol{x}^{(t)}-\frac{1-\alpha^{(t)}}{\sqrt{1-\bar{\alpha}^{(t)}}}\varepsilon\right)
\end{aligned}$$

- What's its corresponding network?
- By definition: let Θ parametrize a network outputting $\boldsymbol{x}^{(t-1)}$ given $\boldsymbol{x}^{(t)}$
  - But the input $\boldsymbol{x}^{(t)}$ also resides in the output target

Noise Predictor

- Θ moves $\mathrm{P}(\mathbf{x}^{(t-1)}|\boldsymbol{x}^{(t)},\Theta)$'s mean toward

$$
\begin{aligned}
&\frac{\sqrt{\alpha^{(t)}}(1-\alpha^{(t-1)})\boldsymbol{x}^{(t)}+\sqrt{\bar{\alpha}^{(t-1)}}\beta^{(t)}\boldsymbol{x}^{(0)}}{1-\bar{\alpha}^{(t)}} \\
&= \frac{\sqrt{\alpha^{(t)}}(1-\alpha^{(t-1)})\boldsymbol{x}^{(t)}+\sqrt{\bar{\alpha}^{(t-1)}}\beta^{(t)}\frac{\boldsymbol{x}^{(t)}-\sqrt{1-\bar{\alpha}^{(t)}}\varepsilon}{\sqrt{\bar{\alpha}^{(t)}}}}{1-\bar{\alpha}^{(t)}} \\
&= \frac{1}{\sqrt{\alpha^{(t)}}}\left(\boldsymbol{x}^{(t)}-\frac{1-\alpha^{(t)}}{\sqrt{1-\bar{\alpha}^{(t)}}}\varepsilon\right)
\end{aligned}
$$

- What's its corresponding network?
- By definition: let Θ parametrize a network outputting $\boldsymbol{x}^{(t-1)}$ given $\boldsymbol{x}^{(t)}$
  - But the input $\boldsymbol{x}^{(t)}$ also resides in the output target
- DDPM: let Θ parametrize a ***noise predictor*** outputting ε given $\boldsymbol{x}^{(t)}$
  - Objective: $\arg\min_{\Theta}\|\varepsilon-e(\boldsymbol{x}^{(t)},t;\Theta)\|^2$
  - Of U-Net [37] architecture
  - Shared between all t

Training & Inference Algorithms

- One-step encoding during training time
- Muti-step inference (sampling), with each intermediate decoding step t, $t > 1$, comes with extra noise $\sigma^{(t)}z$
  - Similar to output token sampling in GPT
  - Improves performance empirically

Algorithm 1 Training

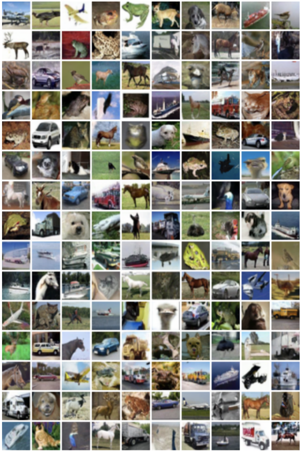
repeat
  $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
  $t \sim \mathrm{Uniform}(\{1, \dots, T\})$
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  Take gradient descent step on
  $\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0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

Algorithm 2 Sampling

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

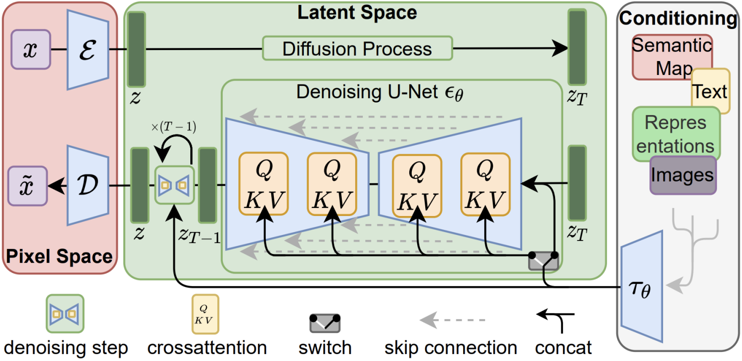
Results

- Sharp and coherent



Conditioning & Scaling

- Stable Diffusion [36]: separately train 3 networks
  - Text/condition embedding: only needs text data
  - Diffusion: needs paired (text-image) data but works at ***latent*** (low dimensional) space
  - Image encoder/decoder: only needs image data
- Text embedding as extra input for denoising net
- Other models like DALL-E [33] also use similar strategies



Results

Text-to-Image Synthesis on LAION. 1.45B Model.

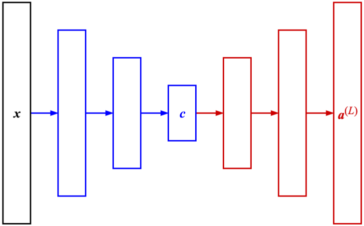
'A street sign that reads "Latent Diffusion" '	'A zombie in the style of Picasso'	'An image of an animal half mouse half octopus'	'An illustration of a slightly conscious neural network'	'A painting of a squirrel eating a burger'	'A watercolor painting of a chair that looks like an octopus'	'A shirt with the inscription: "I love generative models!" '

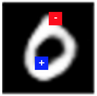
- Even the smallest text-image training set (LAION [41, 40]) has >400M samples!
  - MS COCO [22] "only" has 328K images

Outline

Why Do AE and VAE Give Blurry Images?

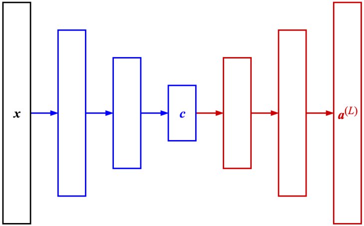
- Objective $\arg\max_{\Theta} \log \mathrm{P}(\mathbb{X}|\Theta) = \arg\min_{\Theta} \sum_n \|\boldsymbol{x}^{(n)} - \boldsymbol{a}^{(n,L)}\|^2$ does not panelize Gaussian pixel noises in $\boldsymbol{a}^{(i,L)}$
  - Root cause: assumption that $\mathbf{x} \sim \mathcal{N}$

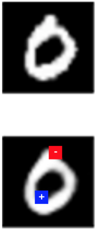




Why Do AE and VAE Give Blurry Images?

- Objective $\arg\max_{\Theta} \log \mathrm{P}(\mathbb{X}|\Theta) = \arg\min_{\Theta} \sum_n \|\boldsymbol{x}^{(n)} - \boldsymbol{a}^{(n,L)}\|^2$ does not panelize Gaussian pixel noises in $\boldsymbol{a}^{(i,L)}$
  - Root cause: assumption that $\mathbf{x} \sim \mathcal{N}$

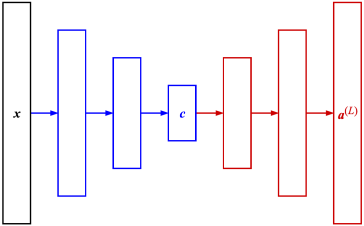


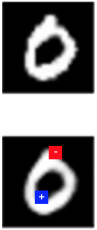


- Fix 1: maximizing the likelihood $\mathrm{P}(\mathbb{X}|\Theta)$ without assuming $\mathbf{x} \sim \mathcal{N}$
  - Flow-based methods, diffusion models

Why Do AE and VAE Give Blurry Images?

- Objective $\arg\max_{\Theta} \log \mathrm{P}(\mathbb{X}|\Theta) = \arg\min_{\Theta} \sum_n \|\boldsymbol{x}^{(n)} - \boldsymbol{a}^{(n,L)}\|^2$ does not panelize Gaussian pixel noises in $\boldsymbol{a}^{(i,L)}$
  - Root cause: assumption that $\mathbf{x} \sim \mathcal{N}$



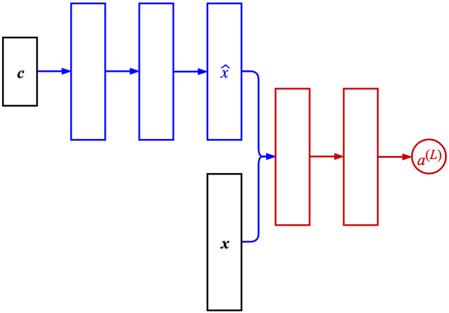


- Fix 1: maximizing the likelihood $\mathrm{P}(\mathbb{X}|\Theta)$ without assuming $\mathbf{x} \sim \mathcal{N}$
  - Flow-based methods, diffusion models
- Fix 2: maximizing the chance that $a^{(L)}$ ***is a true image from another NN point of view***

Outline

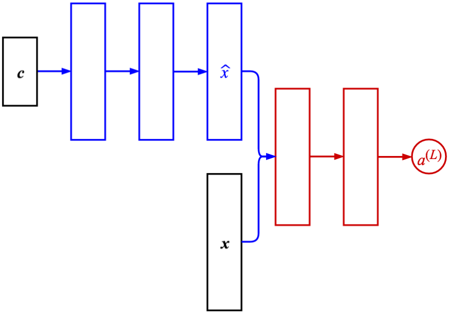
Network Design

- ***Generative adversarial networks*** (***GAN***) [8] consist of:



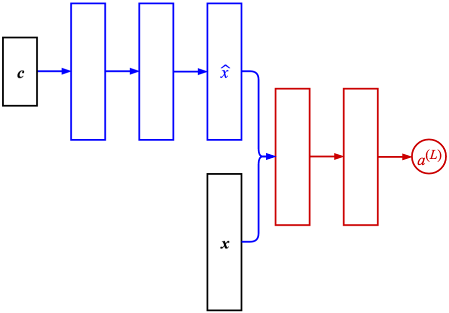
Network Design

- ***Generative adversarial networks*** (***GAN***) [8] consist of:
- ***Generator*** g: to generate data points from random codes
  - No need for "encoder" since the task is data synthesis



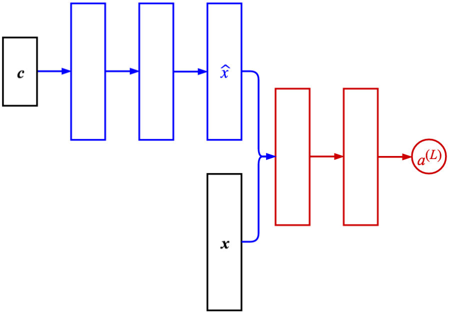
Network Design

- ***Generative adversarial networks*** (***GAN***) [8] consist of:
- ***Generator*** g: to generate data points from random codes
  - No need for "encoder" since the task is data synthesis
- ***Discriminator*** f: to separate generated points from real ones
  - Weights for $\boldsymbol{x}$ and $\hat{\boldsymbol{x}}$ are tied
  - A binary classifier with Sigmoid output unit $a^{(L)} = \hat{\rho}$ for $\mathrm{P}(\mathrm{y} = \text{true point} \,|\, \mathbf{x}) \sim \mathrm{Bernoulli}(\rho)$



Network Design

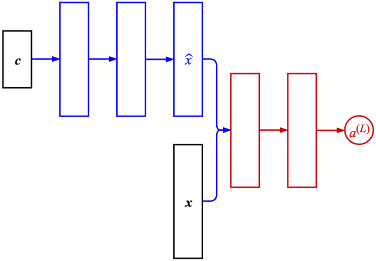
- *Generative adversarial networks* (*GAN*) [8] consist of:
- *Generator* g: to generate data points from random codes
  - No need for "encoder" since the task is data synthesis
- *Discriminator* f: to separate generated points from real ones
  - Weights for $\boldsymbol{x}$ and $\hat{\boldsymbol{x}}$ are tied
  - A binary classifier with Sigmoid output unit $a^{(L)} = \hat{\rho}$ for $\mathrm{P}(\mathrm{y} = \text{true point} \,|\, \mathbf{x}) \sim \mathrm{Bernoulli}(\rho)$
- Goal: to train a g that tricks f into believing $g(\boldsymbol{c})$ is real



Cost Function

- Given N real training points and N generated points:

$$\arg\min_{\Theta_g}\max_{\Theta_f}\log\mathrm{P}(\mathbb{X}\,|\,\Theta_g,\Theta_f)$$
$$=\arg\min_{\Theta_g}\max_{\Theta_f}\sum_n\log f(\boldsymbol{x}^{(n)})+\sum_m\log(1-f(g(\boldsymbol{c}^{(m)})))$$

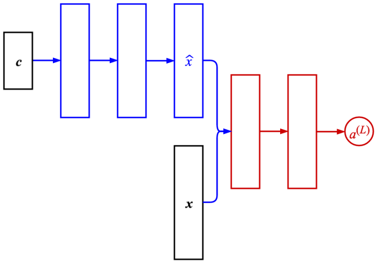


Cost Function

- Given N real training points and N generated points:

$$
\begin{aligned}
&\arg\min_{\Theta_g}\max_{\Theta_f}\log \mathrm{P}(\mathbb{X}\,|\,\Theta_g,\Theta_f)\\
=&\arg\min_{\Theta_g}\max_{\Theta_f}\sum_n \log f(\boldsymbol{x}^{(n)})+\sum_m \log(1-f(g(\boldsymbol{c}^{(m)})))\\
=&\arg\min_{\Theta_g}\max_{\Theta_f}\sum_{n=1}^N \log\hat{\rho}^{(n)}+\sum_{m=1}^N \log(1-\hat{\rho}^{(m)})
\end{aligned}
$$

- Recall that f maximizes the log likelihood
$\log \mathrm{P}(\mathbb{X}\,|\,\Theta)\propto\sum_n \log \mathrm{P}(y^{(n)}\,|\,\boldsymbol{x}^{(n)},\Theta)=\sum_n \log\left[(\hat{\rho}^{(n)})^{y^{(n)}}(1-\hat{\rho}^{(n)})^{(1-y^{(n)})}\right]$



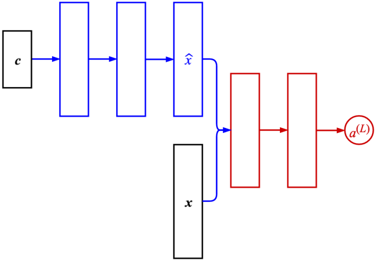
Cost Function

- Given N real training points and N generated points:

$$
\begin{aligned}
&\arg\min_{\Theta_g}\max_{\Theta_f}\log \mathrm{P}(\mathbb{X}\,|\,\Theta_g,\Theta_f)\\
&=\arg\min_{\Theta_g}\max_{\Theta_f}\sum_n \log f(\boldsymbol{x}^{(n)})+\sum_m \log(1-f(g(\boldsymbol{c}^{(m)})))\\
&=\arg\min_{\Theta_g}\max_{\Theta_f}\sum_{n=1}^N \log\hat{\rho}^{(n)}+\sum_{m=1}^N \log(1-\hat{\rho}^{(m)})
\end{aligned}
$$

- Recall that f maximizes the log likelihood
$\log \mathrm{P}(\mathbb{X}\,|\,\Theta)\propto\sum_n \log \mathrm{P}(y^{(n)}\,|\,\boldsymbol{x}^{(n)},\Theta)=\sum_n \log\left[(\hat{\rho}^{(n)})^{y^{(n)}}(1-\hat{\rho}^{(n)})^{(1-y^{(n)})}\right]$

- Inner max first, then outer min



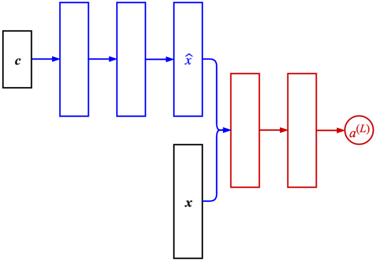
Cost Function

- Given N real training points and N generated points:

$$\arg\min_{\Theta_g}\max_{\Theta_f}\log\mathrm{P}(\mathbb{X}\,|\,\Theta_g,\Theta_f)$$
$$=\arg\min_{\Theta_g}\max_{\Theta_f}\sum_n\log f(\boldsymbol{x}^{(n)})+\sum_m\log(1-f(g(\boldsymbol{c}^{(m)})))$$
$$=\arg\min_{\Theta_g}\max_{\Theta_f}\sum_{n=1}^N\log\hat{\rho}^{(n)}+\sum_{m=1}^N\log(1-\hat{\rho}^{(m)})$$

- Recall that f maximizes the log likelihood
$$\log\mathrm{P}(\mathbb{X}\,|\,\Theta)\propto\sum_n\log\mathrm{P}(y^{(n)}\,|\,\boldsymbol{x}^{(n)},\Theta)=\sum_n\log\left[(\hat{\rho}^{(n)})^{y^{(n)}}(1-\hat{\rho}^{(n)})^{(1-y^{(n)})}\right]$$

- Inner max first, then outer min
- $\hat{\rho}^{(n)}$ depends on Θ_f only
- $\hat{\rho}^{(m)}$ depends on both Θ_f and Θ_g



Training: Alternative SGD

$$\arg\min_{\Theta_g}\max_{\Theta_f}\sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))$$

- Initialize Θ_g for g and Θ_f for f
- At each SGD step/iteration:

Training: Alternative SGD

$$\arg\min_{\Theta_g}\max_{\Theta_f}\sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))$$

- Initialize Θ_g for g and Θ_f for f
- At each SGD step/iteration:

1. ***Repeat K times (with fixed Θ_g):***
    1. Sample N real points $\{\boldsymbol{x}^{(n)}\}_n$ from $\mathbb{X}$ and N codes from $\mathbf{c} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
    2. $\Theta_f \leftarrow \Theta_f + \eta \nabla_{\Theta_f}[\sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))]$

Training: Alternative SGD

$$\arg\min_{\Theta_g}\max_{\Theta_f}\sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))$$

- Initialize Θ_g for g and Θ_f for f
- At each SGD step/iteration:

1. ***Repeat K times (with fixed Θ_g):***
    1. Sample N real points $\{\boldsymbol{x}^{(n)}\}_n$ from $\mathbb{X}$ and N codes from $\mathbf{c} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
    2. $\Theta_f \leftarrow \Theta_f + \eta \nabla_{\Theta_f}[\sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))]$
2. ***Execute once (with fixed Θ_f):***
    1. Sample N codes from $\mathbf{c} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
    2. $\Theta_g \leftarrow \Theta_g - \eta \nabla_{\Theta_g}[\sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))]$

Training: Alternative SGD

$$\arg\min_{\Theta_g}\max_{\Theta_f}\sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1-f(g(\boldsymbol{c}^{(m)})))$$

- Initialize Θ_g for g and Θ_f for f
- At each SGD step/iteration:

1. ***Repeat K times (with fixed Θ_g):***
    1. Sample N real points $\{\boldsymbol{x}^{(n)}\}_n$ from $\mathbb{X}$ and N codes from $\mathbf{c} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
    2. $\Theta_f \leftarrow \Theta_f + \eta \nabla_{\Theta_f}[\sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1-f(g(\boldsymbol{c}^{(m)})))]$
2. ***Execute once (with fixed Θ_f):***
    1. Sample N codes from $\mathbf{c} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
    2. $\Theta_g \leftarrow \Theta_g - \eta \nabla_{\Theta_g}[\sum_m \log(1-f(g(\boldsymbol{c}^{(m)})))]$

- Why limiting the steps (K) when updating Θ_f?

Training: Alternative SGD

$$\arg\min_{\Theta_g}\max_{\Theta_f}\sum_n \log f(\boldsymbol{x}^{(n)})+\sum_m \log(1-f(g(\boldsymbol{c}^{(m)})))$$

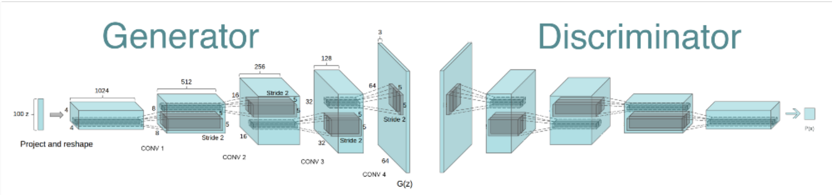
- Initialize Θ_g for g and Θ_f for f
- At each SGD step/iteration:

1. ***Repeat K times (with fixed Θ_g):***
   1. Sample N real points $\{\boldsymbol{x}^{(n)}\}_n$ from $\mathbb{X}$ and N codes from $\mathbf{c}\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})$
   2. $\Theta_f \leftarrow \Theta_f + \eta\nabla_{\Theta_f}[\sum_n \log f(\boldsymbol{x}^{(n)})+\sum_m \log(1-f(g(\boldsymbol{c}^{(m)})))]$
2. ***Execute once (with fixed Θ_f):***
   1. Sample N codes from $\mathbf{c}\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})$
   2. $\Theta_g \leftarrow \Theta_g - \eta\nabla_{\Theta_g}[\sum_m \log(1-f(g(\boldsymbol{c}^{(m)})))]$

- Why limiting the steps (K) when updating Θ_f?
  - f may overfit data and give very different values once g is updated
  - Limiting K so to prevent g from being updated for "wrong" target

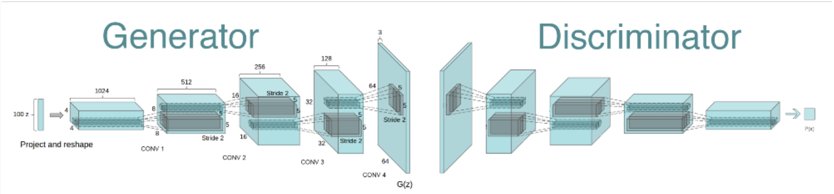
Results

- Domain-specific architecture, e.g., DC-GAN [30]



Results

- Domain-specific architecture, e.g., DC-GAN [30]



GANs Are Hard to Train!

- Tips for Training Stable GANs
- Keep Calm and train a GAN. Pitfalls and Tips...
- 10 Lessons I Learned Training GANs for one Year
- GAN hacks on GitHub

Outline

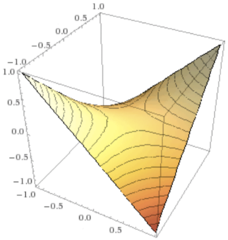
Challenge: Non-Convergence

- The GAN training may ***not*** converge

Challenge: Non-Convergence

- The GAN training may ***not*** converge
- The goal of GAN is to find a saddle point

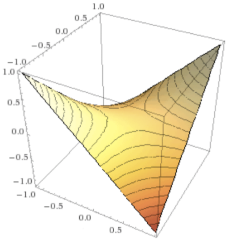
$$\arg\min_{\Theta_g}\max_{\Theta_f}\sum_n \log f(\boldsymbol{x}^{(n)})+\sum_m \log(1-f(g(\boldsymbol{c}^{(m)})))$$



Challenge: Non-Convergence

- The GAN training may ***not*** converge
- The goal of GAN is to find a saddle point

$$\arg\min_{\Theta_g}\max_{\Theta_f}\sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))$$

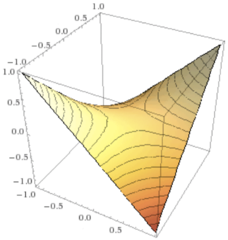


- The updated Θ_f and Θ_g may cancel each other's progress

Challenge: Non-Convergence

- The GAN training may ***not*** converge
- The goal of GAN is to find a saddle point

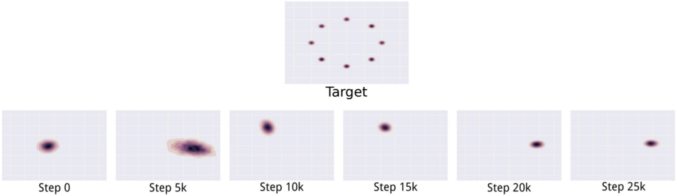
$$\arg\min_{\Theta_g}\max_{\Theta_f}\sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))$$



- The updated Θ_f and Θ_g may cancel each other's progress
- Requires human monitoring and termination

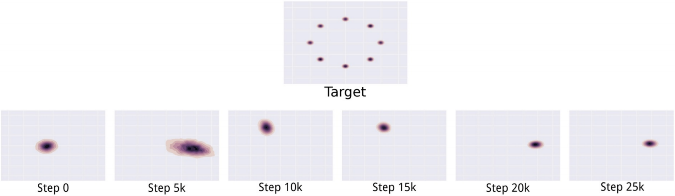
Mode Collapsing

- Even worse: ***mode collapsing***
  - g may oscillate from generating one kind of points to generating another kind of points



Mode Collapsing

- Even worse: ***mode collapsing***
  - g may oscillate from generating one kind of points to generating another kind of points



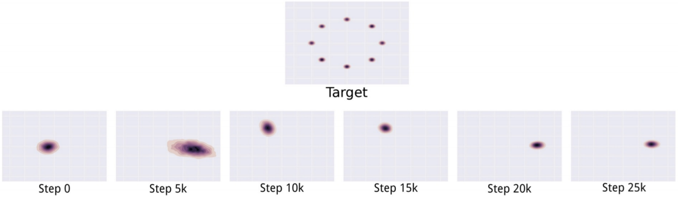
- When K is small, alternate SGD does not distinguish between $\min_{\Theta_g}\max_{\Theta_f}$ and $\max_{\Theta_f}\min_{\Theta_g}$

$$\arg\min_{\Theta_g}\max_{\Theta_f}\sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))$$

- $\max_{\Theta_f}\min_{\Theta_g}$?

Mode Collapsing

- Even worse: ***mode collapsing***
  - g may oscillate from generating one kind of points to generating another kind of points



- When K is small, alternate SGD does not distinguish between $\min_{\Theta_g}\max_{\Theta_f}$ and $\max_{\Theta_f}\min_{\Theta_g}$

$$\arg\min_{\Theta_g}\max_{\Theta_f}\sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))$$

- $\max_{\Theta_f}\min_{\Theta_g}$? g is encouraged to map every code to the "mode" that f believes is most likely to be real

Solutions

- Minibatch discrimination [39]
  - In $\max_{\Theta_f} \min_{\Theta_g}$ case, g collapses because $\nabla_{\Theta_f} C$ are computed independently for each point
  - Why not augment each $\boldsymbol{x}^{(n)}/\hat{\boldsymbol{x}}^{(n)}$ with ***batch features***?

Solutions

- Minibatch discrimination [39]
  - In $\max_{\Theta_f} \min_{\Theta_g}$ case, g collapses because $\nabla_{\Theta_f} C$ are computed independently for each point
  - Why not augment each $\boldsymbol{x}^{(n)}/\hat{\boldsymbol{x}}^{(n)}$ with ***batch features***?
  - If g collapses, f can tell this from batch features and reject fake points
  - Now, g needs to generate dissimilar points to fool f

Solutions

- Minibatch discrimination [39]
  - In $\max_{\Theta_f} \min_{\Theta_g}$ case, g collapses because $\nabla_{\Theta_f} C$ are computed independently for each point
  - Why not augment each $\boldsymbol{x}^{(n)}/\hat{\boldsymbol{x}}^{(n)}$ with ***batch features***?
  - If g collapses, f can tell this from batch features and reject fake points
  - Now, g needs to generate dissimilar points to fool f

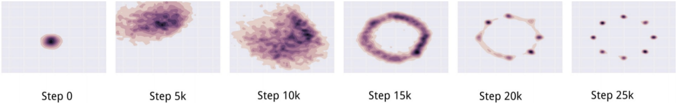
without   with

Solutions

- Minibatch discrimination [39]
  - In $\max_{\Theta_f} \min_{\Theta_g}$ case, g collapses because $\nabla_{\Theta_f} C$ are computed independently for each point
  - Why not augment each $\boldsymbol{x}^{(n)}/\hat{\boldsymbol{x}}^{(n)}$ with ***batch features***?
  - If g collapses, f can tell this from batch features and reject fake points
  - Now, g needs to generate dissimilar points to fool f

without  with

- Unrolled GANs [24]: to back-propagate through ***several*** max ***steps*** when computing $\nabla_{\Theta_g} C$



Challenge: Balance between g and f

$$\arg\min_{\Theta_g}\max_{\Theta_f}\sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1-f(g(\boldsymbol{c}^{(m)})))$$

- Alternate SGD:
  - $\Theta_f \leftarrow \Theta_f + \eta\nabla_{\Theta_f}[\sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1-f(g(\boldsymbol{c}^{(m)})))]$ for K times
  - $\Theta_g \leftarrow \Theta_g - \eta\nabla_{\Theta_g}[\sum_m \log(1-f(g(\boldsymbol{c}^{(m)})))]$

Challenge: Balance between g and f

$$\arg\min_{\Theta_g}\max_{\Theta_f}\sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))$$

- Alternate SGD:
  - $\Theta_f \leftarrow \Theta_f + \eta \nabla_{\Theta_f}[\sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))]$ for K times
  - $\Theta_g \leftarrow \Theta_g - \eta \nabla_{\Theta_g}[\sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))]$
- Why limiting K when updating Θ_f?
- Too large K:
- Too small K:

Challenge: Balance between g and f

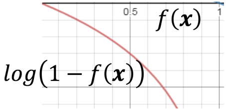
$$\arg\min_{\Theta_g}\max_{\Theta_f}\sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))$$

- Alternate SGD:
  - $\Theta_f \leftarrow \Theta_f + \eta\nabla_{\Theta_f}[\sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))]$ for K times
  - $\Theta_g \leftarrow \Theta_g - \eta\nabla_{\Theta_g}[\sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))]$
- Why limiting K when updating Θ_f?
- Too large K:
  - f may overfit data, making g updated for "wrong" target f
- Too small K:

Challenge: Balance between g and f

$$\arg\min_{\Theta_g}\max_{\Theta_f}\sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))$$

- Alternate SGD:
  - $\Theta_f \leftarrow \Theta_f + \eta \nabla_{\Theta_f}[\sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))]$ for K times
  - $\Theta_g \leftarrow \Theta_g - \eta \nabla_{\Theta_g}[\sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))]$
- Why limiting K when updating Θ_f?
- Too large K:
  - f may overfit data, making g updated for "wrong" target f
  - Vanishing gradients: $\nabla_{\Theta_g}[\sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))]$ too small to learn

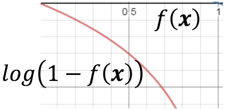


- Too small K:

Challenge: Balance between g and f

$$\arg\min_{\Theta_g}\max_{\Theta_f}\sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1-f(g(\boldsymbol{c}^{(m)})))$$

- Alternate SGD:
  - $\Theta_f \leftarrow \Theta_f + \eta\nabla_{\Theta_f}[\sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1-f(g(\boldsymbol{c}^{(m)})))]$ for K times
  - $\Theta_g \leftarrow \Theta_g - \eta\nabla_{\Theta_g}[\sum_m \log(1-f(g(\boldsymbol{c}^{(m)})))]$
- Why limiting K when updating Θ_f?
- Too large K:
  - f may overfit data, making g updated for "wrong" target f
  - Vanishing gradients: $\nabla_{\Theta_g}[\sum_m \log(1-f(g(\boldsymbol{c}^{(m)})))]$ too small to learn



- Too small K:
  - g updated for "meaningless" f

Solution: Wasserstein GAN [1]

- Let f be a regressor ***without*** the sigmoid output layer
- Cost function:

$$\arg\min_{\Theta_g}\max_{\Theta_f}\sum_n f(\boldsymbol{x}^{(n)}) - \sum_m f(g(\boldsymbol{c}^{(m)}))$$

- Initialize Θ_g for g and Θ_f for f
- At each SGD step/iteration:

Solution: Wasserstein GAN [1]

- Let f be a regressor ***without*** the sigmoid output layer
- Cost function:

$$\arg\min_{\Theta_g}\max_{\Theta_f}\sum_n f(\boldsymbol{x}^{(n)})-\sum_m f(g(\boldsymbol{c}^{(m)}))$$

- Initialize Θ_g for g and Θ_f for f
- At each SGD step/iteration:

1. Repeat K times (with fixed Θ_g):
   1. Sample N real points $\{\boldsymbol{x}^{(n)}\}_n$ from $\mathbb{X}$ and N codes from $\mathbf{c}\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})$
   2. $\Theta_f \leftarrow \Theta_f + \eta\, clip(\nabla_{\Theta_f}[\sum_n f(\boldsymbol{x}^{(n)})-\sum_m f(g(\boldsymbol{c}^{(m)}))])$

Solution: Wasserstein GAN [1]

- Let f be a regressor ***without*** the sigmoid output layer
- Cost function:

$$\arg\min_{\Theta_g}\max_{\Theta_f} \sum_n f(\boldsymbol{x}^{(n)}) - \sum_m f(g(\boldsymbol{c}^{(m)}))$$

- Initialize Θ_g for g and Θ_f for f
- At each SGD step/iteration:

1. Repeat K times (with fixed Θ_g):
   1. Sample N real points $\{\boldsymbol{x}^{(n)}\}_n$ from $\mathbb{X}$ and N codes from $\mathbf{c} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
   2. $\Theta_f \leftarrow \Theta_f + \eta\, clip(\nabla_{\Theta_f}[\sum_n f(\boldsymbol{x}^{(n)}) - \sum_m f(g(\boldsymbol{c}^{(m)}))])$
2. Execute once (with fixed Θ_f):
   1. Sample N codes from $\mathbf{c} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
   2. $\Theta_g \leftarrow \Theta_g - \eta \nabla_{\Theta_g}[-\sum_m f(g(\boldsymbol{c}^{(m)}))]$

GANs from Information Theory Perspective

- Review the Information Theory first!

GANs from Information Theory Perspective

- Review the Information Theory first!
- Let $\mathrm{P}_{\mathsf{data}}$ / P_g be distribution of $\mathbf{x}$ / $\hat{\mathbf{x}} = g(\mathbf{c})$

GANs from Information Theory Perspective

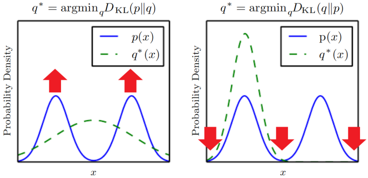
- Review the Information Theory first!
- Let $\mathrm{P}_{\mathsf{data}}$ / P_g be distribution of $\mathbf{x}$ / $\hat{\mathbf{x}} = g(\mathbf{c})$
- A way to find g: $\arg\min_{\Theta_g} \mathrm{D}_{\mathsf{KL}}(\mathrm{P}_{\mathsf{data}} \| \mathrm{P}_g)$

GANs from Information Theory Perspective

- Review the Information Theory first!
- Let $\mathrm{P}_{\mathsf{data}}$ / P_g be distribution of $\mathbf{x}$ / $\hat{\mathbf{x}} = g(\mathbf{c})$
- A way to find g: $\arg\min_{\Theta_g} \mathrm{D}_{\mathsf{KL}}(\mathrm{P}_{\mathsf{data}} \| \mathrm{P}_g)$
  - Why not $\arg\min_{\Theta_g} \mathrm{D}_{\mathsf{KL}}(\mathrm{P}_g \| \mathrm{P}_{\mathsf{data}})$?

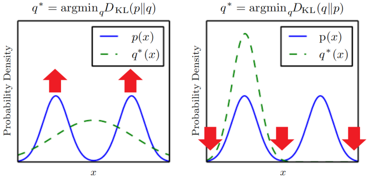
GANs from Information Theory Perspective

- Review the Information Theory first!
- Let P_{data} / P_g be distribution of $\mathbf{x}$ / $\hat{\mathbf{x}} = g(\mathbf{c})$
- A way to find g: $\arg\min_{\Theta_g} \mathrm{D}_{\text{KL}}(\mathrm{P}_{\text{data}} \| \mathrm{P}_g)$
  - Why not $\arg\min_{\Theta_g} \mathrm{D}_{\text{KL}}(\mathrm{P}_g \| \mathrm{P}_{\text{data}})$?



GANs from Information Theory Perspective

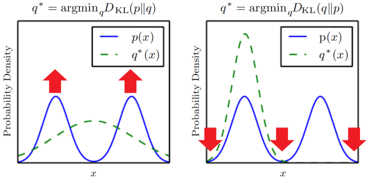
- Review the Information Theory first!
- Let $\mathrm{P}_{\mathrm{data}}$ / P_g be distribution of $\mathbf{x}$ / $\hat{\mathbf{x}} = g(\mathbf{c})$
- A way to find g: $\arg\min_{\Theta_g} \mathrm{D}_{\mathrm{KL}}(\mathrm{P}_{\mathrm{data}} \| \mathrm{P}_g)$
  - Why not $\arg\min_{\Theta_g} \mathrm{D}_{\mathrm{KL}}(\mathrm{P}_g \| \mathrm{P}_{\mathrm{data}})$?



- GAN: $\arg\min_{\Theta_g} \max_{\Theta_f} \sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))$

GANs from Information Theory Perspective

- Review the Information Theory first!
- Let P_{data} / P_g be distribution of $\mathbf{x}$ / $\hat{\mathbf{x}} = g(\mathbf{c})$
- A way to find g: $\arg\min_{\Theta_g} \mathrm{D}_{\mathrm{KL}}(\mathrm{P}_{\text{data}} \| \mathrm{P}_g)$
  - Why not $\arg\min_{\Theta_g} \mathrm{D}_{\mathrm{KL}}(\mathrm{P}_g \| \mathrm{P}_{\text{data}})$?



- GAN: $\arg\min_{\Theta_g} \max_{\Theta_f} \sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))$
- Actually, the max term measures Jensen-Shannon divergence (a.k.a. symmetric KL divergence):

$$\mathrm{D}_{\mathrm{JS}}(\mathrm{P}_{\text{data}} \| \mathrm{P}_g) = \frac{1}{2}\mathrm{D}_{\mathrm{KL}}(\mathrm{P}_{\text{data}} \| \mathrm{Q}) + \frac{1}{2}\mathrm{D}_{\mathrm{KL}}(\mathrm{P}_g \| \mathrm{Q}), \text{ where } \mathrm{Q} = \frac{1}{2}(\mathrm{P}_g + \mathrm{P}_{\text{data}})$$

Why Jensen-Shannon Divergence? I

- Given a fixed g, we have

$$C^* \quad = \max_{\Theta_f} \sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))$$

Why Jensen-Shannon Divergence? I

- Given a fixed g, we have

$$\begin{aligned} C^* &= \max_{\Theta_f} \sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1 - f(g(\boldsymbol{c}^{(m)}))) \\ &= \max_{\Theta_f} \frac{1}{N} \sum_n \log f(\boldsymbol{x}^{(n)}) + \frac{1}{N} \sum_m \log(1 - f(\hat{\boldsymbol{x}}^{(m)})) \end{aligned}$$

Why Jensen-Shannon Divergence? I

- Given a fixed g, we have

$$\begin{aligned} C^* &= \max_{\Theta_f} \sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1 - f(g(\boldsymbol{c}^{(m)}))) \\ &= \max_{\Theta_f} \frac{1}{N} \sum_n \log f(\boldsymbol{x}^{(n)}) + \frac{1}{N} \sum_m \log(1 - f(\hat{\boldsymbol{x}}^{(m)})) \\ &\approx \max_{\Theta_f} \mathrm{E}_{\mathbf{x} \sim \mathrm{P}_{\mathsf{data}}}[\log f(\mathbf{x})] + \mathrm{E}_{\mathbf{x} \sim \mathrm{P}_g}[\log(1 - f(\mathbf{x}))] \end{aligned}$$

Why Jensen-Shannon Divergence? I

- Given a fixed g, we have

$$\begin{aligned} C^* &= \max_{\Theta_f} \sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1 - f(g(\boldsymbol{c}^{(m)}))) \\ &= \max_{\Theta_f} \frac{1}{N} \sum_n \log f(\boldsymbol{x}^{(n)}) + \frac{1}{N} \sum_m \log(1 - f(\hat{\boldsymbol{x}}^{(m)})) \\ &\approx \max_{\Theta_f} \mathrm{E}_{\mathbf{x} \sim \mathrm{P}_{\mathsf{data}}} [\log f(\mathbf{x})] + \mathrm{E}_{\mathbf{x} \sim \mathrm{P}_g} [\log(1 - f(\mathbf{x}))] \\ &= \max_{\Theta_f} \int_{\boldsymbol{x}} \mathrm{P}_{\mathsf{data}}(\boldsymbol{x}) \log f(\boldsymbol{x}) d\boldsymbol{x} + \int_{\boldsymbol{x}} \mathrm{P}_g(\boldsymbol{x}) \log(1 - f(\boldsymbol{x})) d\boldsymbol{x} \end{aligned}$$

Why Jensen-Shannon Divergence? I

- Given a fixed g, we have

$$
\begin{aligned}
C^* \quad &= \max_{\Theta_f} \sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1 - f(g(\boldsymbol{c}^{(m)}))) \\
&= \max_{\Theta_f} \frac{1}{N} \sum_n \log f(\boldsymbol{x}^{(n)}) + \frac{1}{N} \sum_m \log(1 - f(\hat{\boldsymbol{x}}^{(m)})) \\
&\approx \max_{\Theta_f} \mathrm{E}_{\mathbf{x} \sim \mathrm{P}_{\mathsf{data}}} [\log f(\mathbf{x})] + \mathrm{E}_{\mathbf{x} \sim \mathrm{P}_g} [\log(1 - f(\mathbf{x}))] \\
&= \max_{\Theta_f} \int_{\boldsymbol{x}} \mathrm{P}_{\mathsf{data}}(\boldsymbol{x}) \log f(\boldsymbol{x}) d\boldsymbol{x} + \int_{\boldsymbol{x}} \mathrm{P}_g(\boldsymbol{x}) \log(1 - f(\boldsymbol{x})) d\boldsymbol{x} \\
&= \max_{\Theta_f} \int_{\boldsymbol{x}} [\mathrm{P}_{\mathsf{data}}(\boldsymbol{x}) \log f(\boldsymbol{x}) + \mathrm{P}_g(\boldsymbol{x}) \log(1 - f(\boldsymbol{x}))] d\boldsymbol{x}
\end{aligned}
$$

Why Jensen-Shannon Divergence? I

- Given a fixed g, we have

$$
\begin{aligned}
C^* \quad & = \max_{\Theta_f} \sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1 - f(g(\boldsymbol{c}^{(m)}))) \\
& = \max_{\Theta_f} \frac{1}{N} \sum_n \log f(\boldsymbol{x}^{(n)}) + \frac{1}{N} \sum_m \log(1 - f(\hat{\boldsymbol{x}}^{(m)})) \\
& \approx \max_{\Theta_f} \mathrm{E}_{\mathbf{x} \sim \mathrm{P}_{\mathsf{data}}} [\log f(\mathbf{x})] + \mathrm{E}_{\mathbf{x} \sim \mathrm{P}_g} [\log(1 - f(\mathbf{x}))] \\
& = \max_{\Theta_f} \int_{\boldsymbol{x}} \mathrm{P}_{\mathsf{data}}(\boldsymbol{x}) \log f(\boldsymbol{x}) d\boldsymbol{x} + \int_{\boldsymbol{x}} \mathrm{P}_g(\boldsymbol{x}) \log(1 - f(\boldsymbol{x})) d\boldsymbol{x} \\
& = \max_{\Theta_f} \int_{\boldsymbol{x}} [\mathrm{P}_{\mathsf{data}}(\boldsymbol{x}) \log f(\boldsymbol{x}) + \mathrm{P}_g(\boldsymbol{x}) \log(1 - f(\boldsymbol{x}))] d\boldsymbol{x}
\end{aligned}
$$

- To have C^*, we can find f maximizing

$$
\mathrm{P}_{\mathsf{data}}(\boldsymbol{x}) \log f(\boldsymbol{x}) + \mathrm{P}_g(\boldsymbol{x}) \log(1 - f(\boldsymbol{x}))
$$

  for each $\boldsymbol{x}$
  - Assuming that f has infinite capacity

Why Jensen-Shannon Divergence? II

Why Jensen-Shannon Divergence? II

- Given P_{data}, P_g, and $\boldsymbol{x}$, what is the $f(\boldsymbol{x})$ that maximizes

$$\mathrm{P}_{\text{data}}(\boldsymbol{x})\log f(\boldsymbol{x}) + \mathrm{P}_g(\boldsymbol{x})\log(1 - f(\boldsymbol{x}))?$$

- $f^*(\boldsymbol{x}) = \frac{\mathrm{P}_{\text{data}}(\boldsymbol{x})}{\mathrm{P}_{\text{data}}(\boldsymbol{x}) + \mathrm{P}_g(\boldsymbol{x})} \in [0,1]$ [Proof]
- That is,

$$C^* = \max_{\Theta_f} \int_{\boldsymbol{x}} [\mathrm{P}_{\text{data}}(\boldsymbol{x})\log f(\boldsymbol{x}) + \mathrm{P}_g(\boldsymbol{x})\log(1 - f(\boldsymbol{x}))]d\boldsymbol{x}$$

Why Jensen-Shannon Divergence? II

- Given P_{data}, P_g, and $\boldsymbol{x}$, what is the $f(\boldsymbol{x})$ that maximizes

$$\mathrm{P}_{\text{data}}(\boldsymbol{x})\log f(\boldsymbol{x}) + \mathrm{P}_g(\boldsymbol{x})\log(1-f(\boldsymbol{x}))?$$

- $f^*(\boldsymbol{x}) = \frac{\mathrm{P}_{\text{data}}(\boldsymbol{x})}{\mathrm{P}_{\text{data}}(\boldsymbol{x})+\mathrm{P}_g(\boldsymbol{x})} \in [0,1]$ [Proof]
- That is,

$$\begin{aligned} C^* &= \max_{\Theta_f} \int_{\boldsymbol{x}} [\mathrm{P}_{\text{data}}(\boldsymbol{x})\log f(\boldsymbol{x}) + \mathrm{P}_g(\boldsymbol{x})\log(1-f(\boldsymbol{x}))]d\boldsymbol{x} \\ &= \int_{\boldsymbol{x}} \mathrm{P}_{\text{data}}(\boldsymbol{x})\log \frac{\mathrm{P}_{\text{data}}(\boldsymbol{x})}{\mathrm{P}_{\text{data}}(\boldsymbol{x})+\mathrm{P}_g(\boldsymbol{x})} d\boldsymbol{x} \\ &\qquad + \int_{\boldsymbol{x}} \mathrm{P}_g(\boldsymbol{x})\log(\frac{\mathrm{P}_g(\boldsymbol{x})}{\mathrm{P}_{\text{data}}(\boldsymbol{x})+\mathrm{P}_g(\boldsymbol{x})})d\boldsymbol{x} \end{aligned}$$

Why Jensen-Shannon Divergence? II

- Given P_{data}, P_g, and $\boldsymbol{x}$, what is the $f(\boldsymbol{x})$ that maximizes

$$\mathrm{P}_{\text{data}}(\boldsymbol{x})\log f(\boldsymbol{x}) + \mathrm{P}_g(\boldsymbol{x})\log(1-f(\boldsymbol{x}))?$$

- $f^*(\boldsymbol{x}) = \frac{\mathrm{P}_{\text{data}}(\boldsymbol{x})}{\mathrm{P}_{\text{data}}(\boldsymbol{x})+\mathrm{P}_g(\boldsymbol{x})} \in [0,1]$ [Proof]
- That is,

$$\begin{aligned}
C^* &= \max_{\Theta_f} \int_{\boldsymbol{x}} [\mathrm{P}_{\text{data}}(\boldsymbol{x})\log f(\boldsymbol{x}) + \mathrm{P}_g(\boldsymbol{x})\log(1-f(\boldsymbol{x}))] d\boldsymbol{x} \\
&= \int_{\boldsymbol{x}} \mathrm{P}_{\text{data}}(\boldsymbol{x}) \log \frac{\mathrm{P}_{\text{data}}(\boldsymbol{x})}{\mathrm{P}_{\text{data}}(\boldsymbol{x})+\mathrm{P}_g(\boldsymbol{x})} d\boldsymbol{x} \\
&\qquad + \int_{\boldsymbol{x}} \mathrm{P}_g(\boldsymbol{x}) \log(\frac{\mathrm{P}_g(\boldsymbol{x})}{\mathrm{P}_{\text{data}}(\boldsymbol{x})+\mathrm{P}_g(\boldsymbol{x})}) d\boldsymbol{x} \\
&= -2\log 2 + \int_{\boldsymbol{x}} \mathrm{P}_{\text{data}}(\boldsymbol{x}) \log \frac{\mathrm{P}_{\text{data}}(\boldsymbol{x})}{(\mathrm{P}_{\text{data}}(\boldsymbol{x})+\mathrm{P}_g(\boldsymbol{x}))/2} d\boldsymbol{x} \\
&\qquad + \int_{\boldsymbol{x}} \mathrm{P}_g(\boldsymbol{x}) \log(\frac{\mathrm{P}_g(\boldsymbol{x})}{(\mathrm{P}_{\text{data}}(\boldsymbol{x})+\mathrm{P}_g(\boldsymbol{x}))/2}) d\boldsymbol{x}
\end{aligned}$$

Why Jensen-Shannon Divergence? II

- Given P_{data}, P_g, and $\boldsymbol{x}$, what is the $f(\boldsymbol{x})$ that maximizes

$$\mathrm{P}_{\text{data}}(\boldsymbol{x})\log f(\boldsymbol{x})+\mathrm{P}_g(\boldsymbol{x})\log(1-f(\boldsymbol{x}))?$$

- $f^*(\boldsymbol{x})=\frac{\mathrm{P}_{\text{data}}(\boldsymbol{x})}{\mathrm{P}_{\text{data}}(\boldsymbol{x})+\mathrm{P}_g(\boldsymbol{x})}\in[0,1]$ [Proof]
- That is,

$$\begin{aligned}
C^* &= \max_{\Theta_f}\int_{\boldsymbol{x}}[\mathrm{P}_{\text{data}}(\boldsymbol{x})\log f(\boldsymbol{x})+\mathrm{P}_g(\boldsymbol{x})\log(1-f(\boldsymbol{x}))]d\boldsymbol{x}\\
&= \int_{\boldsymbol{x}}\mathrm{P}_{\text{data}}(\boldsymbol{x})\log\frac{\mathrm{P}_{\text{data}}(\boldsymbol{x})}{\mathrm{P}_{\text{data}}(\boldsymbol{x})+\mathrm{P}_g(\boldsymbol{x})}d\boldsymbol{x}\\
&\qquad+\int_{\boldsymbol{x}}\mathrm{P}_g(\boldsymbol{x})\log(\frac{\mathrm{P}_g(\boldsymbol{x})}{\mathrm{P}_{\text{data}}(\boldsymbol{x})+\mathrm{P}_g(\boldsymbol{x})})d\boldsymbol{x}\\
&= -2\log 2+\int_{\boldsymbol{x}}\mathrm{P}_{\text{data}}(\boldsymbol{x})\log\frac{\mathrm{P}_{\text{data}}(\boldsymbol{x})}{(\mathrm{P}_{\text{data}}(\boldsymbol{x})+\mathrm{P}_g(\boldsymbol{x}))/2}d\boldsymbol{x}\\
&\qquad+\int_{\boldsymbol{x}}\mathrm{P}_g(\boldsymbol{x})\log(\frac{\mathrm{P}_g(\boldsymbol{x})}{(\mathrm{P}_{\text{data}}(\boldsymbol{x})+\mathrm{P}_g(\boldsymbol{x}))/2})d\boldsymbol{x}\\
&= -2\log 2+2\mathrm{D}_{\text{JS}}(\mathrm{P}_{\text{data}}\|\mathrm{P}_g)
\end{aligned}$$

Balance between g and f, Revisited I

- Cost function of GAN:

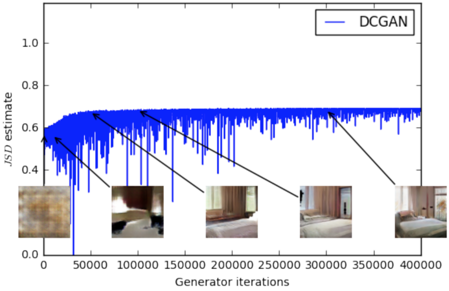
$$\arg\min_{\Theta_g} \max_{\Theta_f} \sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))$$
$$= \arg\min_{\Theta_g} -2\log 2 + 2\mathrm{D}_{\mathsf{JS}}(\mathrm{P}_{\mathsf{data}} \| \mathrm{P}_g)$$

Balance between g and f, Revisited I

- Cost function of GAN:

$$\arg\min_{\Theta_g} \max_{\Theta_f} \sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))$$
$$= \arg\min_{\Theta_g} -2\log 2 + 2\mathrm{D}_{\mathsf{JS}}(\mathrm{P}_{\mathsf{data}} \| \mathrm{P}_g)$$

- However, no mater how g changes, $\mathrm{D}_{\mathsf{JS}}(\mathrm{P}_{\mathsf{data}} \| \mathrm{P}_g)$ remains high during the GAN training process

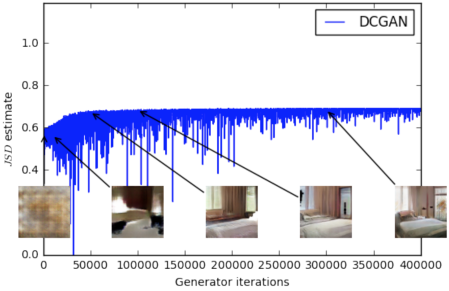


Balance between g and f, Revisited I

- Cost function of GAN:

$$\arg\min_{\Theta_g} \max_{\Theta_f} \sum_n \log f(\boldsymbol{x}^{(n)}) + \sum_m \log(1 - f(g(\boldsymbol{c}^{(m)})))$$
$$= \arg\min_{\Theta_g} -2\log 2 + 2\mathrm{D}_{\mathrm{JS}}(\mathrm{P}_{\mathrm{data}} \| \mathrm{P}_g)$$

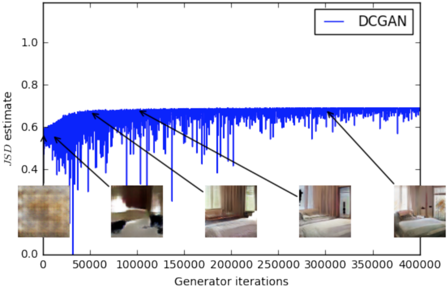
- However, no mater how g changes, $\mathrm{D}_{\mathrm{JS}}(\mathrm{P}_{\mathrm{data}} \| \mathrm{P}_g)$ remains high during the GAN training process



- There's something wrong with the design of the inner max problem!

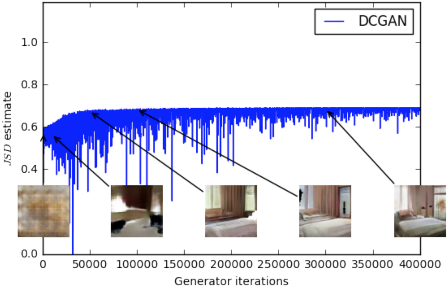
Balance between g and f, Revisited II

- GAN: $\arg\min_{\Theta_g} D_{JS}(P_{data} \| P_g)$



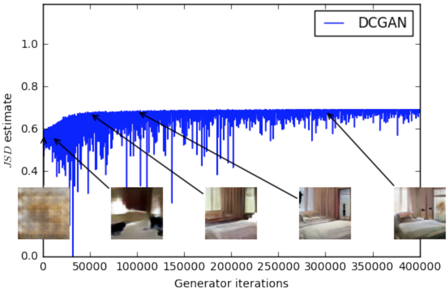
Balance between g and f, Revisited II

- GAN: $\arg\min_{\Theta_g} \mathrm{D}_{\mathrm{JS}}(\mathrm{P}_{\mathrm{data}} \| \mathrm{P}_g)$
- $0 \leq \mathrm{D}_{\mathrm{JS}}(\mathrm{P}_{\mathrm{data}} \| \mathrm{P}_g) \leq \log 2 \approx 0.69$



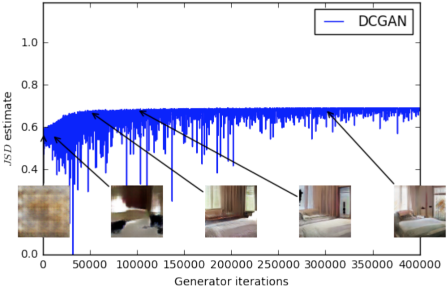
Balance between g and f, Revisited II

- GAN: $\arg\min_{\Theta_g} \mathrm{D}_{\mathrm{JS}}(\mathrm{P}_{\mathrm{data}} \| \mathrm{P}_g)$
- $0 \leq \mathrm{D}_{\mathrm{JS}}(\mathrm{P}_{\mathrm{data}} \| \mathrm{P}_g) \leq \log 2 \approx 0.69$
- When does $\mathrm{D}_{\mathrm{JS}}(\mathrm{P}_{\mathrm{data}} \| \mathrm{P}_g)$ reach its maximum value?



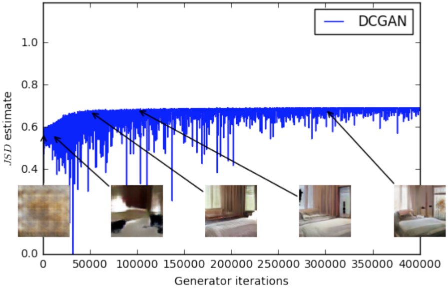
Balance between g and f, Revisited II

- GAN: $\arg\min_{\Theta_g} \mathrm{D}_{\mathrm{JS}}(\mathrm{P}_{\mathrm{data}} \| \mathrm{P}_g)$
- $0 \leq \mathrm{D}_{\mathrm{JS}}(\mathrm{P}_{\mathrm{data}} \| \mathrm{P}_g) \leq \log 2 \approx 0.69$
- When does $\mathrm{D}_{\mathrm{JS}}(\mathrm{P}_{\mathrm{data}} \| \mathrm{P}_g)$ reach its maximum value?
- Let's see the case when P_g and $\mathrm{P}_{\mathrm{data}}$ are "disjointed"



Balance between g and f, Revisited II

- GAN: $\arg\min_{\Theta_g} \mathrm{D}_{\mathrm{JS}}(\mathrm{P}_{\mathrm{data}} \| \mathrm{P}_g)$
- $0 \leq \mathrm{D}_{\mathrm{JS}}(\mathrm{P}_{\mathrm{data}} \| \mathrm{P}_g) \leq \log 2 \approx 0.69$
- When does $\mathrm{D}_{\mathrm{JS}}(\mathrm{P}_{\mathrm{data}} \| \mathrm{P}_g)$ reach its maximum value?
- Let's see the case when P_g and $\mathrm{P}_{\mathrm{data}}$ are "disjointed"

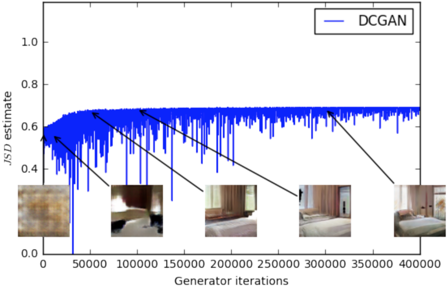


- Suppose $\mathrm{P}_g(\boldsymbol{x}) \neq 0 \Leftrightarrow \mathrm{P}_{\mathrm{data}}(\boldsymbol{x}) = 0$ and $\mathrm{P}_g(\boldsymbol{x}) = 0 \Leftrightarrow \mathrm{P}_{\mathrm{data}}(\boldsymbol{x}) \neq 0$, we have

$$\mathrm{D}_{\mathrm{JS}}(\mathrm{P}_g \| \mathrm{P}_{\mathrm{data}}) = \frac{1}{2}\mathrm{D}_{\mathrm{KL}}\left(\mathrm{P}_g \middle\| \frac{\mathrm{P}_g + \mathrm{P}_{\mathrm{data}}}{2}\right) + \frac{1}{2}\mathrm{D}_{\mathrm{KL}}\left(\mathrm{P}_{\mathrm{data}} \middle\| \frac{\mathrm{P}_g + \mathrm{P}_{\mathrm{data}}}{2}\right)$$

Balance between g and f, Revisited II

- GAN: $\arg\min_{\Theta_g} \mathrm{D}_{\mathrm{JS}}(\mathrm{P}_{\mathrm{data}} \| \mathrm{P}_g)$
- $0 \leq \mathrm{D}_{\mathrm{JS}}(\mathrm{P}_{\mathrm{data}} \| \mathrm{P}_g) \leq \log 2 \approx 0.69$
- When does $\mathrm{D}_{\mathrm{JS}}(\mathrm{P}_{\mathrm{data}} \| \mathrm{P}_g)$ reach its maximum value?
- Let's see the case when P_g and $\mathrm{P}_{\mathrm{data}}$ are "disjointed"

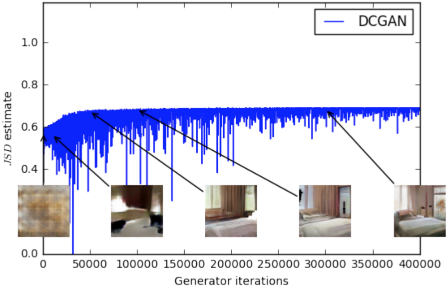


- Suppose $\mathrm{P}_g(\boldsymbol{x}) \neq 0 \Leftrightarrow \mathrm{P}_{\mathrm{data}}(\boldsymbol{x}) = 0$ and $\mathrm{P}_g(\boldsymbol{x}) = 0 \Leftrightarrow \mathrm{P}_{\mathrm{data}}(\boldsymbol{x}) \neq 0$, we have

$$
\begin{aligned}
\mathrm{D}_{\mathrm{JS}}(\mathrm{P}_g \| \mathrm{P}_{\mathrm{data}}) &= \tfrac{1}{2}\mathrm{D}_{\mathrm{KL}}(\mathrm{P}_g \| \tfrac{\mathrm{P}_g + \mathrm{P}_{\mathrm{data}}}{2}) + \tfrac{1}{2}\mathrm{D}_{\mathrm{KL}}(\mathrm{P}_{\mathrm{data}} \| \tfrac{\mathrm{P}_g + \mathrm{P}_{\mathrm{data}}}{2}) \\
&= \tfrac{1}{2}\int_{\boldsymbol{x}} \mathrm{P}_g(\boldsymbol{x}) \log \frac{2\mathrm{P}_g(\boldsymbol{x})}{\mathrm{P}_g(\boldsymbol{x}) + \mathrm{P}_{\mathrm{data}}(\boldsymbol{x})} d\boldsymbol{x} \\
&\quad + \tfrac{1}{2}\int_{\boldsymbol{x}} \mathrm{P}_{\mathrm{data}}(\boldsymbol{x}) \log \frac{2\mathrm{P}_{\mathrm{data}}(\boldsymbol{x})}{\mathrm{P}_g(\boldsymbol{x}) + \mathrm{P}_{\mathrm{data}}(\boldsymbol{x})} d\boldsymbol{x}
\end{aligned}
$$

Balance between g and f, Revisited II

- GAN: $\arg\min_{\Theta_g} \mathrm{D}_{\mathrm{JS}}(\mathrm{P}_{\mathrm{data}} \| \mathrm{P}_g)$
- $0 \leq \mathrm{D}_{\mathrm{JS}}(\mathrm{P}_{\mathrm{data}} \| \mathrm{P}_g) \leq \log 2 \approx 0.69$
- When does $\mathrm{D}_{\mathrm{JS}}(\mathrm{P}_{\mathrm{data}} \| \mathrm{P}_g)$ reach its maximum value?
- Let's see the case when P_g and $\mathrm{P}_{\mathrm{data}}$ are "disjointed"

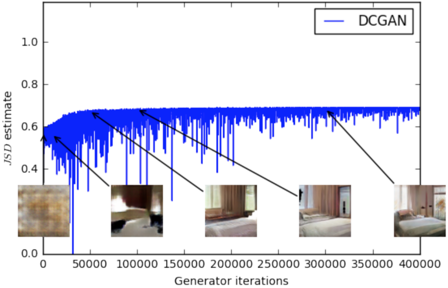


- Suppose $\mathrm{P}_g(\boldsymbol{x}) \neq 0 \Leftrightarrow \mathrm{P}_{\mathrm{data}}(\boldsymbol{x}) = 0$ and $\mathrm{P}_g(\boldsymbol{x}) = 0 \Leftrightarrow \mathrm{P}_{\mathrm{data}}(\boldsymbol{x}) \neq 0$, we have

$$
\begin{aligned}
\mathrm{D}_{\mathrm{JS}}(\mathrm{P}_g \| \mathrm{P}_{\mathrm{data}}) &= \tfrac{1}{2}\mathrm{D}_{\mathrm{KL}}\left(\mathrm{P}_g \| \tfrac{\mathrm{P}_g + \mathrm{P}_{\mathrm{data}}}{2}\right) + \tfrac{1}{2}\mathrm{D}_{\mathrm{KL}}\left(\mathrm{P}_{\mathrm{data}} \| \tfrac{\mathrm{P}_g + \mathrm{P}_{\mathrm{data}}}{2}\right) \\
&= \tfrac{1}{2}\int_{\boldsymbol{x}} \mathrm{P}_g(\boldsymbol{x}) \log \frac{2\mathrm{P}_g(\boldsymbol{x})}{\mathrm{P}_g(\boldsymbol{x}) + \mathrm{P}_{\mathrm{data}}(\boldsymbol{x})} d\boldsymbol{x} \\
&\quad + \tfrac{1}{2}\int_{\boldsymbol{x}} \mathrm{P}_{\mathrm{data}}(\boldsymbol{x}) \log \frac{2\mathrm{P}_{\mathrm{data}}(\boldsymbol{x})}{\mathrm{P}_g(\boldsymbol{x}) + \mathrm{P}_{\mathrm{data}}(\boldsymbol{x})} d\boldsymbol{x} \\
&= \tfrac{1}{2}\log 2 + \tfrac{1}{2}\log 2 = \log 2
\end{aligned}
$$

Balance between g and f, Revisited II

- GAN: $\arg\min_{\Theta_g} \mathrm{D}_{\mathrm{JS}}(\mathrm{P}_{\text{data}} \| \mathrm{P}_g)$
- $0 \leq \mathrm{D}_{\mathrm{JS}}(\mathrm{P}_{\text{data}} \| \mathrm{P}_g) \leq \log 2 \approx 0.69$
- When does $\mathrm{D}_{\mathrm{JS}}(\mathrm{P}_{\text{data}} \| \mathrm{P}_g)$ reach its maximum value?
- Let's see the case when P_g and P_{data} are "disjointed"



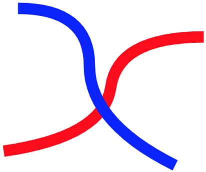
- Suppose $\mathrm{P}_g(\boldsymbol{x}) \neq 0 \Leftrightarrow \mathrm{P}_{\text{data}}(\boldsymbol{x}) = 0$ and $\mathrm{P}_g(\boldsymbol{x}) = 0 \Leftrightarrow \mathrm{P}_{\text{data}}(\boldsymbol{x}) \neq 0$, we have

$$\begin{aligned}
\mathrm{D}_{\mathrm{JS}}(\mathrm{P}_g \| \mathrm{P}_{\text{data}}) &= \tfrac{1}{2}\mathrm{D}_{\mathrm{KL}}(\mathrm{P}_g \| \tfrac{\mathrm{P}_g + \mathrm{P}_{\text{data}}}{2}) + \tfrac{1}{2}\mathrm{D}_{\mathrm{KL}}(\mathrm{P}_{\text{data}} \| \tfrac{\mathrm{P}_g + \mathrm{P}_{\text{data}}}{2}) \\
&= \tfrac{1}{2}\int_{\boldsymbol{x}} \mathrm{P}_g(\boldsymbol{x}) \log \frac{2\mathrm{P}_g(\boldsymbol{x})}{\mathrm{P}_g(\boldsymbol{x}) + \mathrm{P}_{\text{data}}(\boldsymbol{x})} d\boldsymbol{x} \\
&\quad + \tfrac{1}{2}\int_{\boldsymbol{x}} \mathrm{P}_{\text{data}}(\boldsymbol{x}) \log \frac{2\mathrm{P}_{\text{data}}(\boldsymbol{x})}{\mathrm{P}_g(\boldsymbol{x}) + \mathrm{P}_{\text{data}}(\boldsymbol{x})} d\boldsymbol{x} \\
&= \tfrac{1}{2}\log 2 + \tfrac{1}{2}\log 2 = \log 2
\end{aligned}$$

- Are P_g and P_{data} really disjointed during the GAN training?

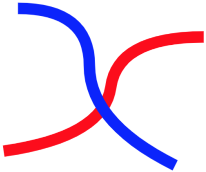
Disjoining $\mathbf{P}_g$ and $\mathbf{P}_{\text{data}}$

- In a high dimensional space, $\mathbf{x}$ and $g(\mathbf{z})$ may resides in low dimensional manifolds
  - P_g and P_{data} may have values only on the manifolds



Disjoining $\mathbf{P}_g$ and $\mathbf{P}_{\text{data}}$

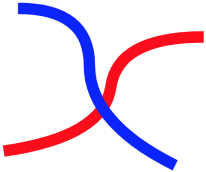
- In a high dimensional space, $\mathbf{x}$ and $g(\mathbf{z})$ may resides in low dimensional manifolds
  - P_g and P_{data} may have values only on the manifolds



- P_g and P_{data} can be very different initially during GAN training

Disjoining $\mathbf{P}_g$ and $\mathbf{P}_{\text{data}}$

- In a high dimensional space, $\mathbf{x}$ and $g(\mathbf{z})$ may resides in low dimensional manifolds
  - P_g and P_{data} may have values only on the manifolds



- P_g and P_{data} can be very different initially during GAN training
- The intersections where $\mathrm{P}_g(\boldsymbol{x}) \neq 0$ and $\mathrm{P}_{\text{data}}(\boldsymbol{x}) \neq 0$ can be neglected
  - $\mathrm{P}_g(\boldsymbol{x}) \neq 0 \Leftrightarrow \mathrm{P}_{\text{data}}(\boldsymbol{x}) = 0$ and $\mathrm{P}_g(\boldsymbol{x}) = 0 \Leftrightarrow \mathrm{P}_{\text{data}}(\boldsymbol{x}) \neq 0$ almost surely
  - Maximum JS divergence at all time

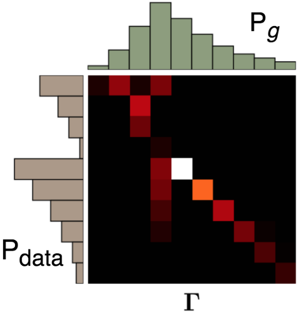
Better Divergence Measure?

- GAN: $\arg\min_{\Theta_g} \mathrm{D}_{\mathsf{JS}}(\mathrm{P}_{\mathsf{data}} \| \mathrm{P}_g)$
  - $\mathrm{D}_{\mathsf{JS}}(\mathrm{P}_{\mathsf{data}} \| \mathrm{P}_g)$ does ***not*** bring P_g and $\mathrm{P}_{\mathsf{data}}$ closer during GAN training

Better Divergence Measure?

- GAN: $\arg\min_{\Theta_g} \mathrm{D}_{\mathrm{JS}}(\mathrm{P}_{\mathrm{data}} \| \mathrm{P}_g)$
  - $\mathrm{D}_{\mathrm{JS}}(\mathrm{P}_{\mathrm{data}} \| \mathrm{P}_g)$ does ***not*** bring P_g and $\mathrm{P}_{\mathrm{data}}$ closer during GAN training
- Wasserstein (or earth-mover) distance:

$$
\begin{aligned}
\mathrm{W}(\mathrm{P}_{\mathrm{data}}, \mathrm{P}_g) &= \inf_{\mathrm{Q} \in \Gamma(\mathrm{P}_{\mathrm{data}}, \mathrm{P}_g)} \mathrm{E}_{(\mathbf{x}, \hat{\mathbf{x}}) \sim \mathrm{Q}}[\|\mathbf{x} - \hat{\mathbf{x}}\|] \\
&= \inf_{\mathrm{Q} \in \Gamma(\mathrm{P}_{\mathrm{data}}, \mathrm{P}_g)} \int_{(\boldsymbol{x}, \hat{\boldsymbol{x}})} \mathrm{Q}(\boldsymbol{x}, \hat{\boldsymbol{x}}) \|\boldsymbol{x} - \hat{\boldsymbol{x}}\| d(\boldsymbol{x}, \hat{\boldsymbol{x}})
\end{aligned}
$$

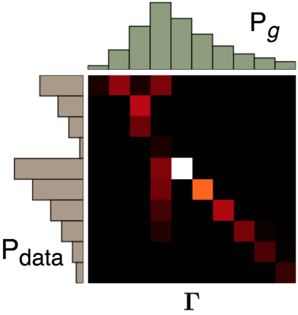


Better Divergence Measure?

- GAN: $\arg\min_{\Theta_g} D_{\text{JS}}(P_{\text{data}} \| P_g)$
  - $D_{\text{JS}}(P_{\text{data}} \| P_g)$ does ***not*** bring P_g and P_{data} closer during GAN training
- Wasserstein (or earth-mover) distance:

$$\begin{aligned} W(P_{\text{data}}, P_g) &= \inf_{Q \in \Gamma(P_{\text{data}}, P_g)} E_{(\mathbf{x}, \hat{\mathbf{x}}) \sim Q}[\|\mathbf{x} - \hat{\mathbf{x}}\|] \\ &= \inf_{Q \in \Gamma(P_{\text{data}}, P_g)} \int_{(\boldsymbol{x}, \hat{\boldsymbol{x}})} Q(\boldsymbol{x}, \hat{\boldsymbol{x}}) \|\boldsymbol{x} - \hat{\boldsymbol{x}}\| d(\boldsymbol{x}, \hat{\boldsymbol{x}}) \end{aligned}$$

- Intuitively, the minimal "cost" to change P_{data} into P_g

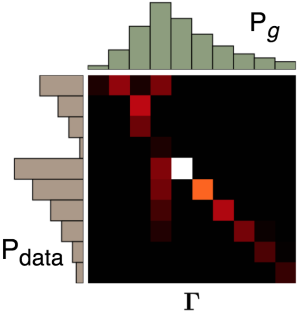


Better Divergence Measure?

- GAN: $\arg\min_{\Theta_g} \mathrm{D}_{\mathrm{JS}}(\mathrm{P}_{\text{data}} \| \mathrm{P}_g)$
  - $\mathrm{D}_{\mathrm{JS}}(\mathrm{P}_{\text{data}} \| \mathrm{P}_g)$ does ***not*** bring P_g and P_{data} closer during GAN training
- Wasserstein (or earth-mover) distance:

$$\begin{aligned} \mathrm{W}(\mathrm{P}_{\text{data}}, \mathrm{P}_g) &= \inf_{\mathrm{Q} \in \Gamma(\mathrm{P}_{\text{data}}, \mathrm{P}_g)} \mathrm{E}_{(\mathbf{x}, \hat{\mathbf{x}}) \sim \mathrm{Q}}[\|\mathbf{x} - \hat{\mathbf{x}}\|] \\ &= \inf_{\mathrm{Q} \in \Gamma(\mathrm{P}_{\text{data}}, \mathrm{P}_g)} \int_{(\boldsymbol{x}, \hat{\boldsymbol{x}})} \mathrm{Q}(\boldsymbol{x}, \hat{\boldsymbol{x}}) \|\boldsymbol{x} - \hat{\boldsymbol{x}}\| d(\boldsymbol{x}, \hat{\boldsymbol{x}}) \end{aligned}$$

  - Intuitively, the minimal "cost" to change P_{data} into P_g
  - $\mathrm{W}(\mathrm{P}_{\text{data}}, \mathrm{P}_g)$ measures the "divergence" between P_g and P_{data} ***even when they are disjointed***



Wasserstein GAN I

- W-GAN: $\arg\min_{\Theta_g} \mathrm{W}(\mathrm{P}_{\mathsf{data}}, \mathrm{P}_g)$
  - $\mathrm{W}(\mathrm{P}_{\mathsf{data}}, \mathrm{P}_g) = \inf_{\mathrm{Q} \in \Gamma(\mathrm{P}_{\mathsf{data}}, \mathrm{P}_g)} \mathrm{E}_{(\mathbf{x}, \hat{\mathbf{x}}) \sim \mathrm{Q}}[\|\mathbf{x} - \hat{\mathbf{x}}\|]$

Wasserstein GAN I

- W-GAN: $\arg\min_{\Theta_g} \mathrm{W}(\mathrm{P}_{\mathsf{data}}, \mathrm{P}_g)$
  - $\mathrm{W}(\mathrm{P}_{\mathsf{data}}, \mathrm{P}_g) = \inf_{\mathrm{Q} \in \Gamma(\mathrm{P}_{\mathsf{data}}, \mathrm{P}_g)} \mathrm{E}_{(\mathbf{x}, \hat{\mathbf{x}}) \sim \mathrm{Q}}[\|\mathbf{x} - \hat{\mathbf{x}}\|]$
- Unfortunately, $\mathrm{W}(\mathrm{P}_{\mathsf{data}}, \mathrm{P}_g)$ is hard to solve directly

Wasserstein GAN II

Theorem

Consider f's that are Lipschitz continuous with constant 1, i.e.,

$$|f(\boldsymbol{x}) - f(\hat{\boldsymbol{x}})| \leq 1 \cdot \|\boldsymbol{x} - \hat{\boldsymbol{x}}\|, \forall \boldsymbol{x}, \hat{\boldsymbol{x}},$$

we have [a]

$$W(P_{\text{data}}, P_g) \quad = \sup_f E_{\mathbf{x} \sim P_{\text{data}}}[f(\mathbf{x})] - E_{\mathbf{x} \sim P_g}[f(\mathbf{x})]$$

Wasserstein GAN II

Theorem

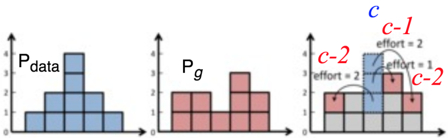
Consider f's that are Lipschitz continuous with constant 1, i.e.,

$$|f(\boldsymbol{x}) - f(\hat{\boldsymbol{x}})| \leq 1 \cdot \|\boldsymbol{x} - \hat{\boldsymbol{x}}\|, \forall \boldsymbol{x}, \hat{\boldsymbol{x}},$$

we have [a]

$$\begin{aligned} W(P_{\text{data}}, P_g) &= \sup_f E_{\mathbf{x} \sim P_{\text{data}}}[f(\mathbf{x})] - E_{\mathbf{x} \sim P_g}[f(\mathbf{x})] \\ &= \sup_f \int_{\boldsymbol{x}} (P_{\text{data}}(\boldsymbol{x}) - P_g(\boldsymbol{x})) f(\boldsymbol{x}) d\boldsymbol{x}. \end{aligned}$$

[a] https://vincentherrmann.github.io/blog/wasserstein/



Wasserstein GAN II

Theorem

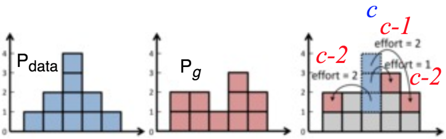
Consider f's that are Lipschitz continuous with constant 1, i.e.,

$$|f(\boldsymbol{x}) - f(\hat{\boldsymbol{x}})| \leq 1 \cdot \|\boldsymbol{x} - \hat{\boldsymbol{x}}\|, \forall \boldsymbol{x}, \hat{\boldsymbol{x}},$$

we have [a]

$$\begin{aligned} W(P_{\text{data}}, P_g) &= \sup_f E_{\mathbf{x} \sim P_{\text{data}}}[f(\mathbf{x})] - E_{\mathbf{x} \sim P_g}[f(\mathbf{x})] \\ &= \sup_f \int_{\boldsymbol{x}} (P_{\text{data}}(\boldsymbol{x}) - P_g(\boldsymbol{x})) f(\boldsymbol{x}) d\boldsymbol{x}. \end{aligned}$$

[a] https://vincentherrmann.github.io/blog/wasserstein/



- W-GAN [1]: $\arg\min_{\Theta_g} \max_{\Theta_f} \mathrm{E}_{\mathbf{x} \sim \mathrm{P}_{\text{data}}}[f(\mathbf{x})] - \mathrm{E}_{\mathbf{x} \sim \mathrm{P}_g}[f(\mathbf{x})]$

Alternate SGD for W-GAN

$$\arg\min_{\Theta_g}\max_{\Theta_f} \mathrm{E}_{\mathbf{x}\sim \mathrm{P}_{\mathsf{data}}}[f(\mathbf{x})] - \mathrm{E}_{\mathbf{x}\sim \mathrm{P}_g}[f(\mathbf{x})]$$
$$= \arg\min_{\Theta_g}\max_{\Theta_f} \sum_n f(\boldsymbol{x}^{(n)}) - \sum_m f(g(\boldsymbol{c}^{(m)}))$$

- f a regressor ***without*** the sigmoid output layer

Alternate SGD for W-GAN

$$\arg\min_{\Theta_g} \max_{\Theta_f} \mathrm{E}_{\mathbf{x}\sim \mathrm{P}_{\text{data}}}[f(\mathbf{x})] - \mathrm{E}_{\mathbf{x}\sim \mathrm{P}_g}[f(\mathbf{x})]$$
$$= \arg\min_{\Theta_g} \max_{\Theta_f} \sum_n f(\boldsymbol{x}^{(n)}) - \sum_m f(g(\boldsymbol{c}^{(m)}))$$

- f a regressor ***without*** the sigmoid output layer
- Initialize Θ_g for g and Θ_f for f
- At each SGD step/iteration:

1. Repeat K times (with fixed Θ_g):
    1. Sample N real points $\{\boldsymbol{x}^{(n)}\}_n$ from $\mathbb{X}$ and N codes from $\mathbf{c} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
    2. $\Theta_f \leftarrow \Theta_f + \eta\, \mathit{clip}(\nabla_{\Theta_f}[\sum_n f(\boldsymbol{x}^{(n)}) - \sum_m f(g(\boldsymbol{c}^{(m)}))])$
2. Execute once (with fixed Θ_f):
    1. Sample N codes from $\mathbf{c} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
    2. $\Theta_g \leftarrow \Theta_g - \eta \nabla_{\Theta_g}[-\sum_m f(g(\boldsymbol{c}^{(m)}))]$

Why Gradient Clipping?

- Update rule for Θ_f:

$$\Theta_f \leftarrow \Theta_f + \eta\, clip(\nabla_{\Theta_f}[\sum_n f(\boldsymbol{x}^{(n)}) - \sum_m f(g(\boldsymbol{c}^{(m)}))])$$

- Gradient clipping: $\forall w \in \Theta_f$, $clip(w) = \max(\min(w, \tau), -\tau)$ for some threshold $\tau > 0$
- Why?

Why Gradient Clipping?

- Update rule for Θ_f:

$$\Theta_f \leftarrow \Theta_f + \eta\, clip(\nabla_{\Theta_f}[\sum_n f(\boldsymbol{x}^{(n)}) - \sum_m f(g(\boldsymbol{c}^{(m)}))])$$

- Gradient clipping: $\forall w \in \Theta_f$, $clip(w) = \max(\min(w, \tau), -\tau)$ for some threshold $\tau > 0$
- Why?
- In W-GAN, we have :

$$\arg\min_{\Theta_g} \mathrm{W}(\mathrm{P}_{\text{data}}, \mathrm{P}_g)$$
$$= \arg\min_{\Theta_g} \max_{\Theta_f} \sum_n f(\boldsymbol{x}^{(n)}) - \sum_m f(g(\boldsymbol{c}^{(m)}))$$

only if f is 1-Lipschitz continuous

Why Gradient Clipping?

- Update rule for Θ_f:

$$\Theta_f \leftarrow \Theta_f + \eta\, clip(\nabla_{\Theta_f}[\sum_n f(\boldsymbol{x}^{(n)}) - \sum_m f(g(\boldsymbol{c}^{(m)}))])$$

- Gradient clipping: $\forall w \in \Theta_f$, $clip(w) = \max(\min(w, \tau), -\tau)$ for some threshold $\tau > 0$
- Why?
- In W-GAN, we have :

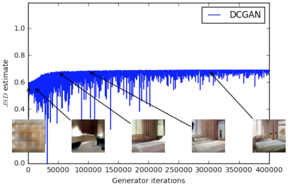
$$\arg\min_{\Theta_g} \mathrm{W}(\mathrm{P}_{\text{data}}, \mathrm{P}_g)$$
$$= \arg\min_{\Theta_g} \max_{\Theta_f} \sum_n f(\boldsymbol{x}^{(n)}) - \sum_m f(g(\boldsymbol{c}^{(m)}))$$

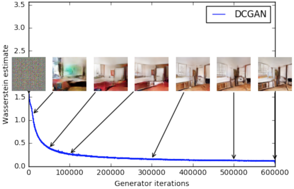
only if f is 1-Lipschitz continuous

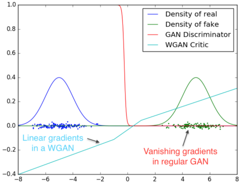
- Heuristics for making f 1-Lipchitz

Advantages of W-GAN

- "Corrects" the inner max problem
  - Wasserstein distance guides g even when P_g and P_{data} "disjointed"
  - Training less sensitive to K (balance between g and f)
- The max value can be used as a "stop" indicator
- f a regressor, avoids vanishing gradients for g







Improved W-GAN I

- In practice, W-GAN training converges slowly and is unstable to τ

Improved W-GAN I

- In practice, W-GAN training converges slowly and is unstable to τ
- W-GAN use a small τ to make f 1-Lipschitz continuous
- However, too small a τ severely limits the capacity of f such it cannot actually maximize

$$\max_{\Theta_f} \mathrm{E}_{\mathbf{x}\sim \mathrm{P}_{\mathsf{data}}}[f(\mathbf{x})] - \mathrm{E}_{\mathbf{x}\sim \mathrm{P}_g}[f(\mathbf{x})]$$

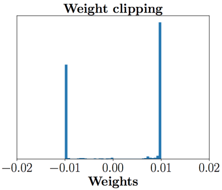
- g is not updated for minimizing $\mathrm{W}(\mathrm{P}_{\mathsf{data}}, \mathrm{P}_g)$

Improved W-GAN I

- In practice, W-GAN training converges slowly and is unstable to τ
- W-GAN use a small τ to make f 1-Lipschitz continuous
- However, too small a τ severely limits the capacity of f such it cannot actually maximize

$$\max_{\Theta_f} \mathrm{E}_{\mathbf{x}\sim \mathrm{P}_{\text{data}}}[f(\mathbf{x})] - \mathrm{E}_{\mathbf{x}\sim \mathrm{P}_g}[f(\mathbf{x})]$$

- g is not updated for minimizing $\mathrm{W}(\mathrm{P}_{\text{data}}, \mathrm{P}_g)$
- Distribution of weight values of f $(\tau = 0.01)$:



- Exploding and vanishing gradients

Improved W-GAN II

- If f is 1-Lipschitz, then $\|\nabla f(\boldsymbol{x})\| \leq 1$ for all $\boldsymbol{x}$
- Why not just panelize $\|\nabla f(\boldsymbol{x})\| > 1$ for all $\boldsymbol{x}$?
- Cost function:

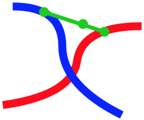
$$\arg\min_{\Theta_g} \max_{\Theta_f} \mathrm{E}_{\mathbf{x} \sim \mathrm{P}_{\mathsf{data}}}[f(\mathbf{x})] - \mathrm{E}_{\mathbf{x} \sim \mathrm{P}_g}[f(\mathbf{x})]$$
$$-\lambda \mathrm{E}_{\mathbf{x} \sim \mathrm{P}_{\mathsf{penalty}}}[\max(0, \|\nabla f(\mathbf{x})\| - 1)]$$

Improved W-GAN II

- If f is 1-Lipschitz, then $\|\nabla f(\boldsymbol{x})\| \leq 1$ for all $\boldsymbol{x}$
- Why not just panelize $\|\nabla f(\boldsymbol{x})\| > 1$ for all $\boldsymbol{x}$?
- Cost function:

$$\arg\min_{\Theta_g} \max_{\Theta_f} \mathrm{E}_{\mathbf{x}\sim \mathrm{P}_{\mathsf{data}}}[f(\mathbf{x})] - \mathrm{E}_{\mathbf{x}\sim \mathrm{P}_g}[f(\mathbf{x})] \\ -\lambda \mathrm{E}_{\mathbf{x}\sim \mathrm{P}_{\mathsf{penalty}}}[\max(0, \|\nabla f(\mathbf{x})\| - 1)]$$

- $\mathrm{P}_{\mathsf{penalty}}$?

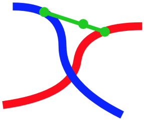


Improved W-GAN II

- If f is 1-Lipschitz, then $\|\nabla f(\boldsymbol{x})\| \leq 1$ for all $\boldsymbol{x}$
- Why not just panelize $\|\nabla f(\boldsymbol{x})\| > 1$ for all $\boldsymbol{x}$?
- Cost function:

$$\arg\min_{\Theta_g} \max_{\Theta_f} \mathrm{E}_{\mathbf{x}\sim \mathrm{P}_{\mathsf{data}}}[f(\mathbf{x})] - \mathrm{E}_{\mathbf{x}\sim \mathrm{P}_g}[f(\mathbf{x})] \\ -\lambda \mathrm{E}_{\mathbf{x}\sim \mathrm{P}_{\mathsf{penalty}}}[\max(0, \|\nabla f(\mathbf{x})\| - 1)]$$

- $\mathrm{P}_{\mathsf{penalty}}$?

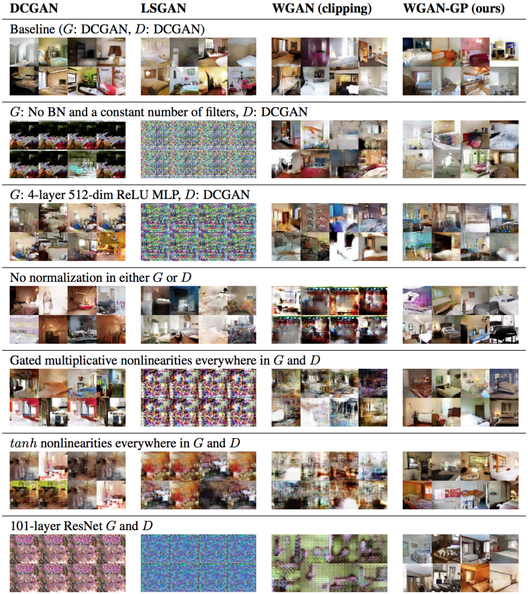


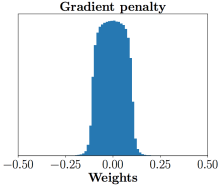
- W-GAN-GP [9]:

$$\arg\min_{\Theta_g}\max_{\Theta_f} \sum_n f(\boldsymbol{x}^{(n)}) - \sum_m f(g(\boldsymbol{c}^{(m)})) - \lambda \sum_p (\|\nabla f(\boldsymbol{x}^{(p)})\| - 1)^2$$

  - The larger $f(\boldsymbol{x}^{(p)})$ the better (subject to $\|\nabla f(\boldsymbol{x}^{(p)})\| \leq 1$)
  - Faster convergence

Results


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


Gradient penalty
−0.50
−0.25
0.00
0.25
0.50
Weights

Challenge: Global Coherence

Challenge: Global Coherence

- Large images generated by GANs usually lack global coherency

Counting

Perspective

Shape

Challenge: Global Coherence

- Large images generated by GANs usually lack global coherency

Counting

Perspective

Shape

- A CNN, when used as f, detects ***existence*** of patterns more than their ***relative positions***
    - f loses track of the position of a pattern after several pooling layers
    - Relative position of patterns in $\mathbb{X}$ may change due to different view angels
- Solutions?

Challenge: Global Coherence

- Large images generated by GANs usually lack global coherency

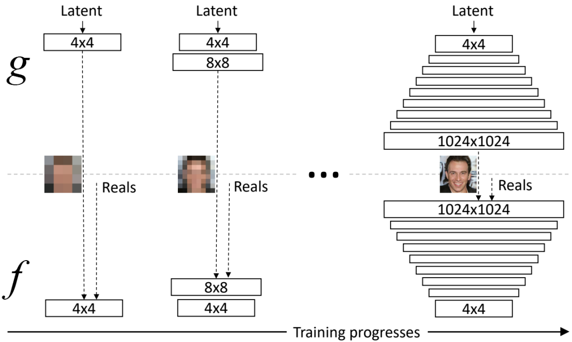
Counting

Perspective

Shape

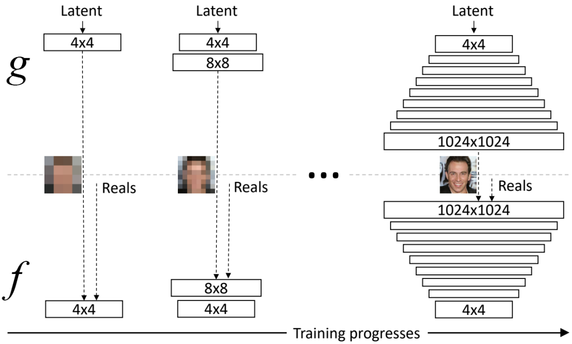
- A CNN, when used as f, detects ***existence*** of patterns more than their ***relative positions***
  - f loses track of the position of a pattern after several pooling layers
  - Relative position of patterns in $\mathbb{X}$ may change due to different view angels
- Solutions? A better f, such as the CapsuleNet [38]?

Progressive Growing of GANs [14]



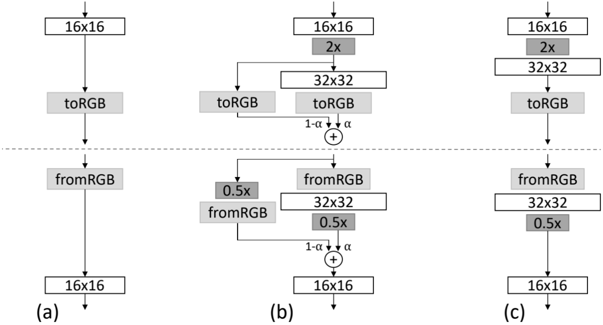
- Incrementally adds new layers in sequential GAN trainings
  - Convolution + upsampling for g each time
  - Convolution + downpooling for f each time
- Real images are downscaled to match the current resolution of g

Progressive Growing of GANs [14]



- Incrementally adds new layers in sequential GAN trainings
  - Convolution + upsampling for g each time
  - Convolution + downpooling for f each time
- Real images are downscaled to match the current resolution of g
- Goal: to let new layers ***add details without ruining the context***

Transition when Adding a New Layer



- Gradually increases α
  - New convolution layer in g/f learns to generate/detect details first
  - Then learns the "context"

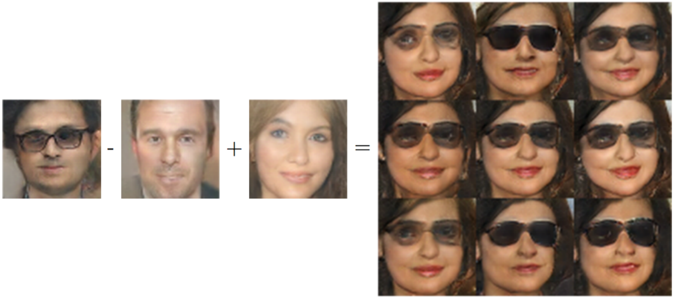
Results

- Minibatch discrimination [39] + W-GAN-GP [9] + progressive growing [14] + other tricks
- $2 \sim 6$ times faster

Outline

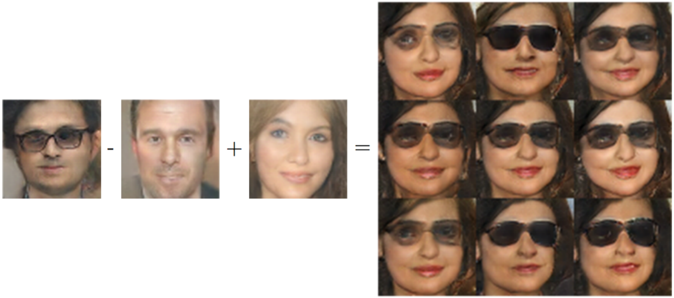
Code Space Arithmetics

- DC-GAN [30] can learn to use codes in meaningful ways:



Code Space Arithmetics

- DC-GAN [30] can learn to use codes in meaningful ways:



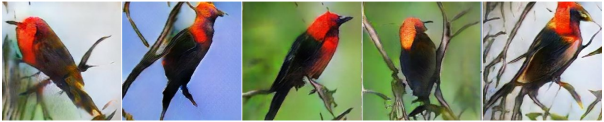
- Finding codes for images with constraints [50, 3] ‣ Demo 1 ‣ Demo 2

$$\arg\min_{\boldsymbol{c}} \|mask(g(\boldsymbol{c})) - \text{constraint}\|_F$$

Conditional GAN I

- Text to image synthesis [34]: $\mathbb{X} = \{(\boldsymbol{x}^{(n)}, \boldsymbol{\phi}^{(n)})\}_n$

  "*This bird is completely red with black wings and pointy beak.*"

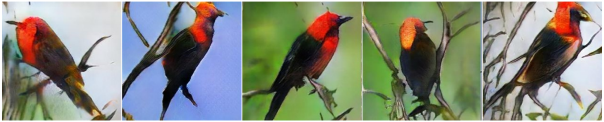


- How?

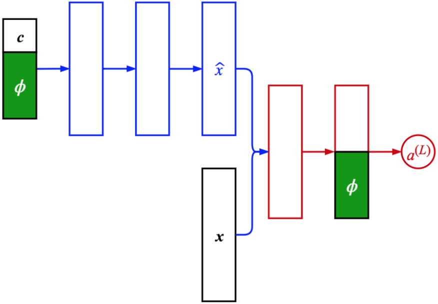
Conditional GAN I

- Text to image synthesis [34]: $\mathbb{X} = \{(\boldsymbol{x}^{(n)}, \boldsymbol{\phi}^{(n)})\}_n$

  "This bird is completely red with black wings and pointy beak."

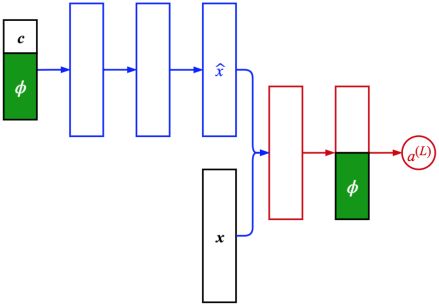


- How?



Conditional GAN II

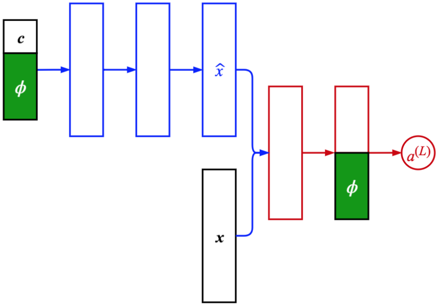
- Pitfall: g and f can choose to ignore the condition ϕ altogether
- Solution?



Conditional GAN II

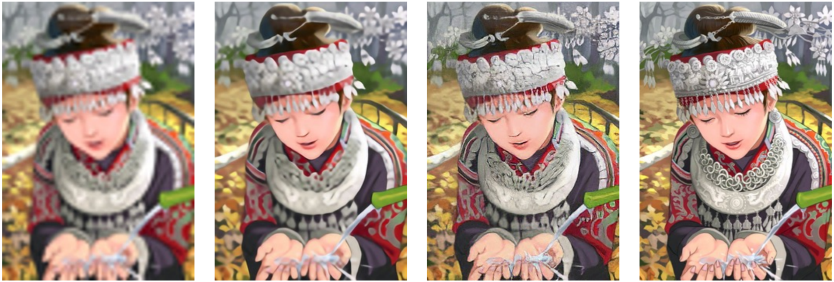
- Pitfall: g and f can choose to ignore the condition ϕ altogether
- Solution? Conditioned labeling

$$
\begin{aligned}
&(\boldsymbol{x}^{(n)}, \phi^{(n)}) \Rightarrow \text{true} \\
&(\boldsymbol{x}^{(n)}, \phi') \Rightarrow \text{false}, \forall \phi' \neq \phi^{(n)} \\
&(\hat{\boldsymbol{x}}^{(m)}, \phi^{(m)}) \Rightarrow \text{false}
\end{aligned}
$$



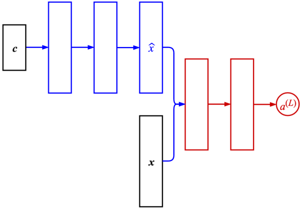
Super Resolution [18]

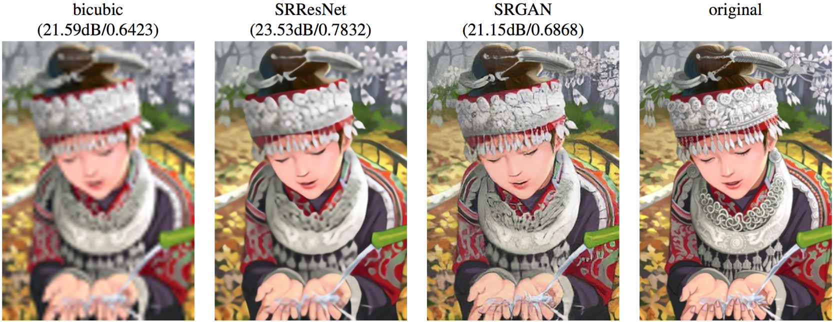
bicubic (21.59dB/0.6423) | SRResNet (23.53dB/0.7832) | SRGAN (21.15dB/0.6868) | original



Super Resolution [18]

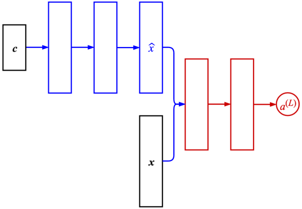
- g : low res img $\to$ high res img
  - Training: $\boldsymbol{c}$'s are downscaled images





Super Resolution [18]

- g : low res img $\rightarrow$ high res img
  - Training: $\boldsymbol{c}$'s are downscaled images
- No "creativity," f acts as a better lose metric



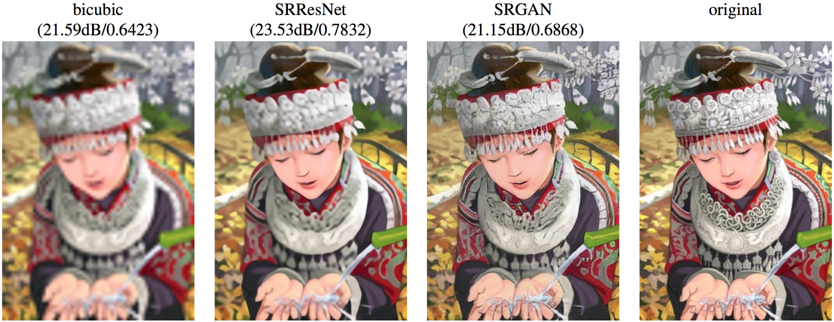


Image-to-Image Translation [13] I

- Given an image $\boldsymbol{x}_{\text{src}}$ in source domain, generate image(s) $\boldsymbol{x}_{\text{target}}$ in target domain
  - $\boldsymbol{x}_{\text{src}}$ and $\boldsymbol{x}_{\text{target}}$ are semantically aligned

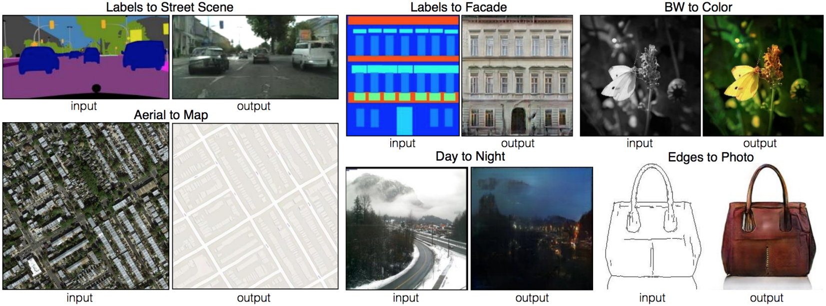


Image-to-Image Translation [13] II

- Based on conditional GAN:
  - $\boldsymbol{c} = \boldsymbol{x}_{\text{src}}$; $g(\boldsymbol{c}) = \boldsymbol{x}_{\text{target}}$
  - Conditioned labels
  - Uses dropout layers to create diversity (if needed)

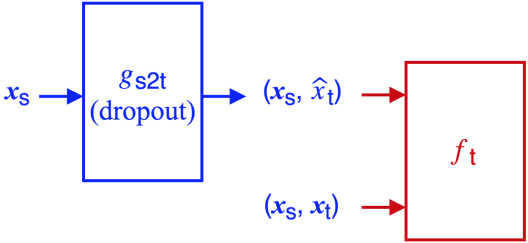
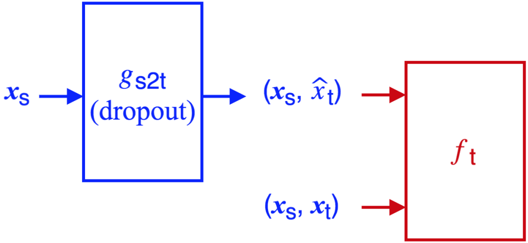


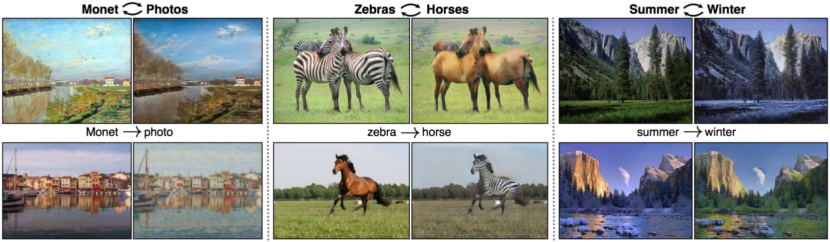
Image-to-Image Translation [13] II

- Based on conditional GAN:
  - $\boldsymbol{c} = \boldsymbol{x}_{\text{src}}$; $g(\boldsymbol{c}) = \boldsymbol{x}_{\text{target}}$
  - Conditioned labels
  - Uses dropout layers to create diversity (if needed)
- Requires ***paired*** examples $\mathbb{X} = \{(\boldsymbol{x}_{\text{src}}^{(n)}, \boldsymbol{x}_{\text{target}}^{(n)})\}_n$



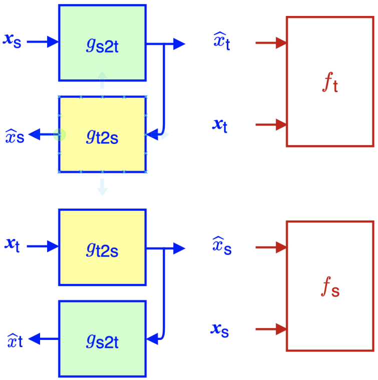
Unpaired Image-to-Image Translation I

- What if the images in different domains are ***unpaired***?
  - $\mathbb{X} = \{\boldsymbol{x}_{\text{src}}^{(n)}\}_n \cup \{\boldsymbol{x}_{\text{target}}^{(n)}\}_n$



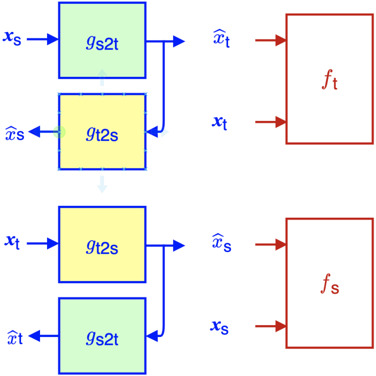
Unpaired Image-to-Image Translation II

- Cycle GAN [51]: to train two generators $g_{\text{src2target}}$ and $g_{\text{target2src}}$ simultaneously in two GANs



Unpaired Image-to-Image Translation II

- Cycle GAN [51]: to train two generators $g_{\text{src2target}}$ and $g_{\text{target2src}}$ simultaneously in two GANs
- Add a loss term $\sum_n \|\boldsymbol{x}^{(n)}_{\text{src}} - g_{\text{target2src}}(g_{\text{src2target}}(\boldsymbol{x}^{(n)}_{\text{src}}))\|_F$ and $\sum_n \|\boldsymbol{x}^{(n)}_{\text{target}} - g_{\text{src2target}}(g_{\text{target2src}}(\boldsymbol{x}^{(n)}_{\text{target}}))\|_F$ for $g_{\text{src2target}}$ and $g_{\text{target2src}}$



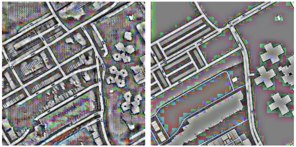
How Does Cycle GAN Work?

How Does Cycle GAN Work?

- Ideal: $g_{\text{src2target}}$ and $g_{\text{target2src}}$ learns to translate images

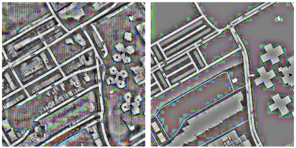
How Does Cycle GAN Work?

- Ideal: $g_{\text{src2target}}$ and $g_{\text{target2src}}$ learns to translate images
- Reality: $g_{\text{src2target}}$ and $g_{\text{target2src}}$ learns to hide inoformation [6]



How Does Cycle GAN Work?

- Ideal: $g_{\text{src2target}}$ and $g_{\text{target2src}}$ learns to translate images
- Reality: $g_{\text{src2target}}$ and $g_{\text{target2src}}$ learns to hide inoformation [6]



- *Unsupervised DNN models, including GANs, may not work as one may expect* quality?

Reference I

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou.
Wasserstein generative adversarial networks.
In *International Conference on Machine Learning*, pages 214–223, 2017.

[2] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al.
Constitutional ai: Harmlessness from ai feedback.
arXiv preprint arXiv:2212.08073, 2022.

[3] Andrew Brock, Theodore Lim, JM Ritchie, and Nick Weston.
Neural photo editing with introspective adversarial networks.
arXiv preprint arXiv:1609.07093, 2016.

Reference II

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al.
Language models are few-shot learners.
Advances in neural information processing systems, 33:1877–1901, 2020.

[5] Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T Freeman.
Maskgit: Masked generative image transformer.
In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11315–11325, 2022.

[6] Casey Chu, Andrey Zhmoginov, and Mark Sandler.
Cyclegan, a master of steganography.
arXiv preprint arXiv:1712.02950, 2017.

Reference III

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova.
Bert: Pre-training of deep bidirectional transformers for language understanding.
arXiv preprint arXiv:1810.04805, 2018.

[8] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio.
Generative adversarial nets.
In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

[9] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville.
Improved training of wasserstein gans.
arXiv preprint arXiv:1704.00028, 2017.

Reference IV

[10] Jonathan Ho, Ajay Jain, and Pieter Abbeel.
Denoising diffusion probabilistic models.
Advances in neural information processing systems, 33:6840–6851, 2020.

[11] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al.
Training compute-optimal large language models.
arXiv preprint arXiv:2203.15556, 2022.

[12] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen.
Lora: Low-rank adaptation of large language models.
arXiv preprint arXiv:2106.09685, 2021.

Reference V

[13] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros.
Image-to-image translation with conditional adversarial networks.
arXiv preprint arXiv:1611.07004, 2016.

[14] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen.
Progressive growing of gans for improved quality, stability, and variation.
arXiv preprint arXiv:1710.10196, 2017.

[15] Diederik P Kingma and Max Welling.
Auto-encoding variational bayes.
arXiv preprint arXiv:1312.6114, 2013.

[16] Durk P Kingma and Prafulla Dhariwal.
Glow: Generative flow with invertible 1x1 convolutions.
Advances in neural information processing systems, 31, 2018.

Reference VI

[17] Quoc V Le and Tomas Mikolov.
Distributed representations of sentences and documents.
In *ICML*, volume 14, pages 1188–1196, 2014.

[18] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al.
Photo-realistic single image super-resolution using a generative adversarial network.
arXiv preprint arXiv:1609.04802, 2016.

[19] Daniel D Lee and H Sebastian Seung.
Learning the parts of objects by non-negative matrix factorization.
Nature, 401(6755):788–791, 1999.

Reference VII

[20] Daniel D Lee and H Sebastian Seung.
Algorithms for non-negative matrix factorization.
In *Advances in neural information processing systems*, pages 556–562, 2001.

[21] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al.
Retrieval-augmented generation for knowledge-intensive nlp tasks.
Advances in Neural Information Processing Systems, 33:9459–9474, 2020.

Reference VIII

[22] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick.
Microsoft coco: Common objects in context.
In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.

[23] Calvin Luo.
Understanding diffusion models: A unified perspective.
arXiv preprint arXiv:2208.11970, 2022.

[24] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein.
Unrolled generative adversarial networks.
arXiv preprint arXiv:1611.02163, 2016.

[25] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean.
Efficient estimation of word representations in vector space.
arXiv preprint arXiv:1301.3781, 2013.

Reference IX

[26] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean.
Distributed representations of words and phrases and their compositionality.
In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[27] Varun Nair, Elliot Schumacher, Geoffrey Tso, and Anitha Kannan.
Dera: enhancing large language model completions with dialog-enabled resolving agents.
arXiv preprint arXiv:2303.17071, 2023.

[28] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al.
Webgpt: Browser-assisted question-answering with human feedback.
arXiv preprint arXiv:2112.09332, 2021.

Reference X

[29] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al.
Training language models to follow instructions with human feedback.
Advances in Neural Information Processing Systems, 35:27730–27744, 2022.

[30] Alec Radford, Luke Metz, and Soumith Chintala.
Unsupervised representation learning with deep convolutional generative adversarial networks.
arXiv preprint arXiv:1511.06434, 2015.

[31] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al.
Improving language understanding by generative pre-training.
OpenAI blog, 2018.

Reference XI

[32] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al.
Language models are unsupervised multitask learners.
OpenAI blog, 2019.

[33] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen.
Hierarchical text-conditional image generation with clip latents.
arXiv preprint arXiv:2204.06125, 2022.

[34] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee.
Generative adversarial text to image synthesis.
arXiv preprint arXiv:1605.05396, 2016.

Reference XII

[35] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio.
Contractive auto-encoders: Explicit invariance during feature extraction.
In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 833–840, 2011.

[36] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer.
High-resolution image synthesis with latent diffusion models.
In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

Reference XIII

[37] Olaf Ronneberger, Philipp Fischer, and Thomas Brox.
U-net: Convolutional networks for biomedical image segmentation.
In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[38] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton.
Dynamic routing between capsules.
In *Advances in Neural Information Processing Systems*, pages 3857–3867, 2017.

[39] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen.
Improved techniques for training gans.
In *Advances in Neural Information Processing Systems*, pages 2226–2234, 2016.

Reference XIV

[40] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, et al.
Laion-5b: An open large-scale dataset for training next generation image-text models.
Advances in Neural Information Processing Systems, 35:25278–25294, 2022.

[41] Christoph Schuhmann, Richard Vencu, Romain Beaumont, Robert Kaczmarczyk, Clayton Mullis, Aarush Katta, Theo Coombes, Jenia Jitsev, and Aran Komatsuzaki.
Laion-400m: Open dataset of clip-filtered 400 million image-text pairs.
arXiv preprint arXiv:2111.02114, 2021.

Reference XV

[42] Or Sharir, Barak Peleg, and Yoav Shoham.
The cost of training nlp models: A concise overview.
arXiv preprint arXiv:2004.08900, 2020.

[43] Patrice Simard, Bernard Victorri, Yann LeCun, and John S Denker.
Tangent prop-a formalism for specifying selected invariances in an adaptive network.
In *NIPS*, volume 91, pages 895–903, 1991.

[44] Aaron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al.
Conditional image generation with pixelcnn decoders.
In *Advances in Neural Information Processing Systems*, pages 4790–4798, 2016.

Reference XVI

[45] Aaron Van Oord, Nal Kalchbrenner, and Koray Kavukcuoglu.
Pixel recurrent neural networks.
In *International Conference on Machine Learning*, pages 1747–1756, 2016.

[46] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin.
Attention is all you need.
Advances in neural information processing systems, 30, 2017.

[47] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol.
Extracting and composing robust features with denoising autoencoders.
In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

Reference XVII

[48] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al.
Emergent abilities of large language models.
arXiv preprint arXiv:2206.07682, 2022.

[49] Matthew D Zeiler and Rob Fergus.
Visualizing and understanding convolutional networks.
In *European conference on computer vision*, pages 818–833. Springer, 2014.

[50] Jun-Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A Efros.
Generative visual manipulation on the natural image manifold.
In *European Conference on Computer Vision*, pages 597–613. Springer, 2016.

Reference XVIII

[51] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros.
Unpaired image-to-image translation using cycle-consistent adversarial networks.
arXiv preprint arXiv:1703.10593, 2017.