Introduction to Generative Adversarial Networks

Nikita Popov, Maxim Ryabinin

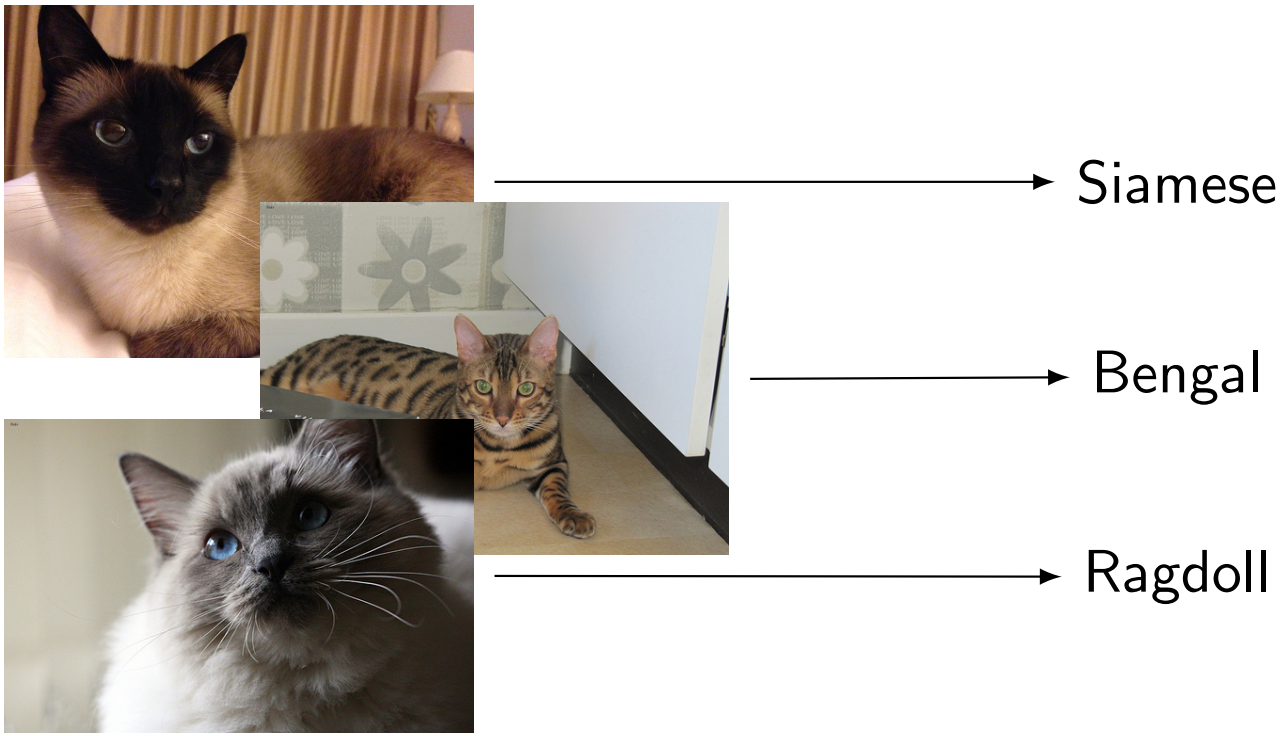
January 26, 2018

Talk outline

Supervised setting

Suppose we want to classify cat images by breed and have a (large enough) labeled dataset[1]. Easy to solve —

```
from torchvision.models import resnet152
```



In probabilistic setting this corresponds to learning $p(\text{label}|\text{features})$. Such models are called **discriminative**.

[1] http://www.robots.ox.ac.uk/~vgg/data/pets/

Moving to data generation

What if the task we want to solve is not "Which breed is that cat?", but "What is a cat? Show me an example"?

Training data

Generated sample

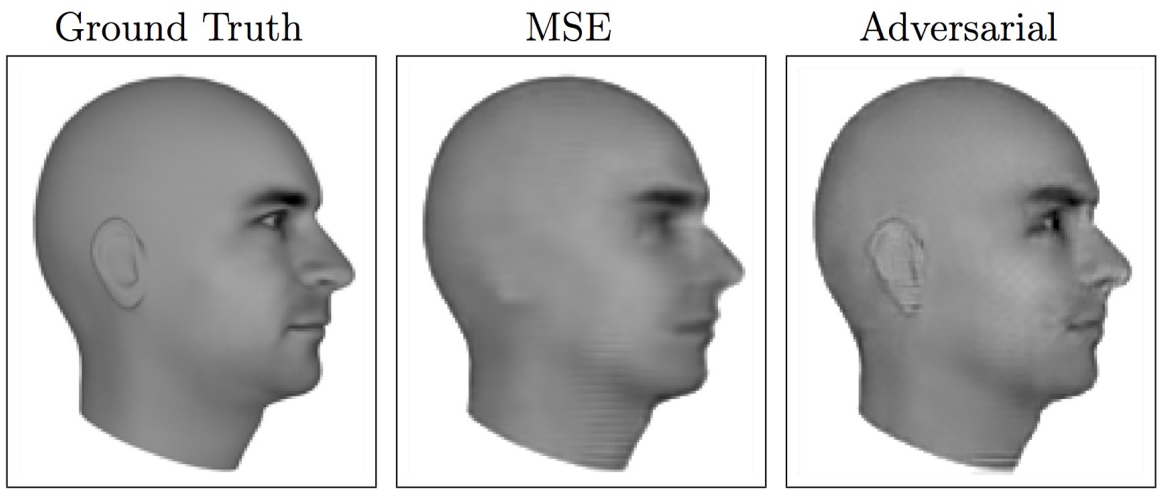
In other words, we want to learn $p_{\text{model}} \approx p_{\text{data}}$. Approaches that model the distribution of inputs are known as **generative**.

Comparison with discriminative approach

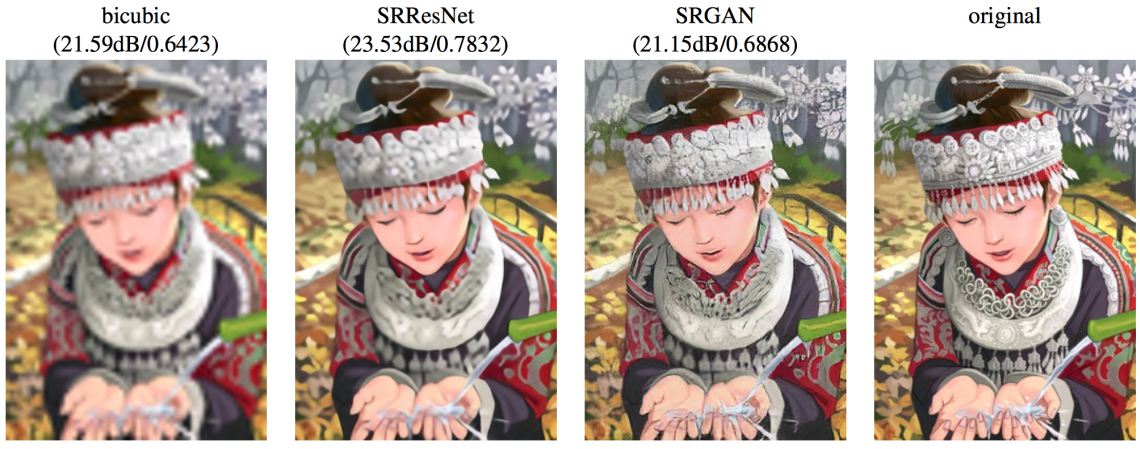
Discriminative models	Generative models
Learn $p(y\|x)$	Learn $p(x, y)$ (or $p(x)$ if no explicit labels)
Suitable for any supervised learning task	Possible to work in unsupervised setting
Many fast and accurate algorithms	Hard to obtain good quality on complex data
Need all features to be present for prediction	May predict for data with missing features
Not possible to generate new data	Can sample from learned distribution

Why study generative modeling?

- Can be trained with missing data (and labels)
- Useful for planning in reinforcement learning
- Works better with multimodal outputs
- Required by some tasks, for example, image super-resolution



Next video frame prediction

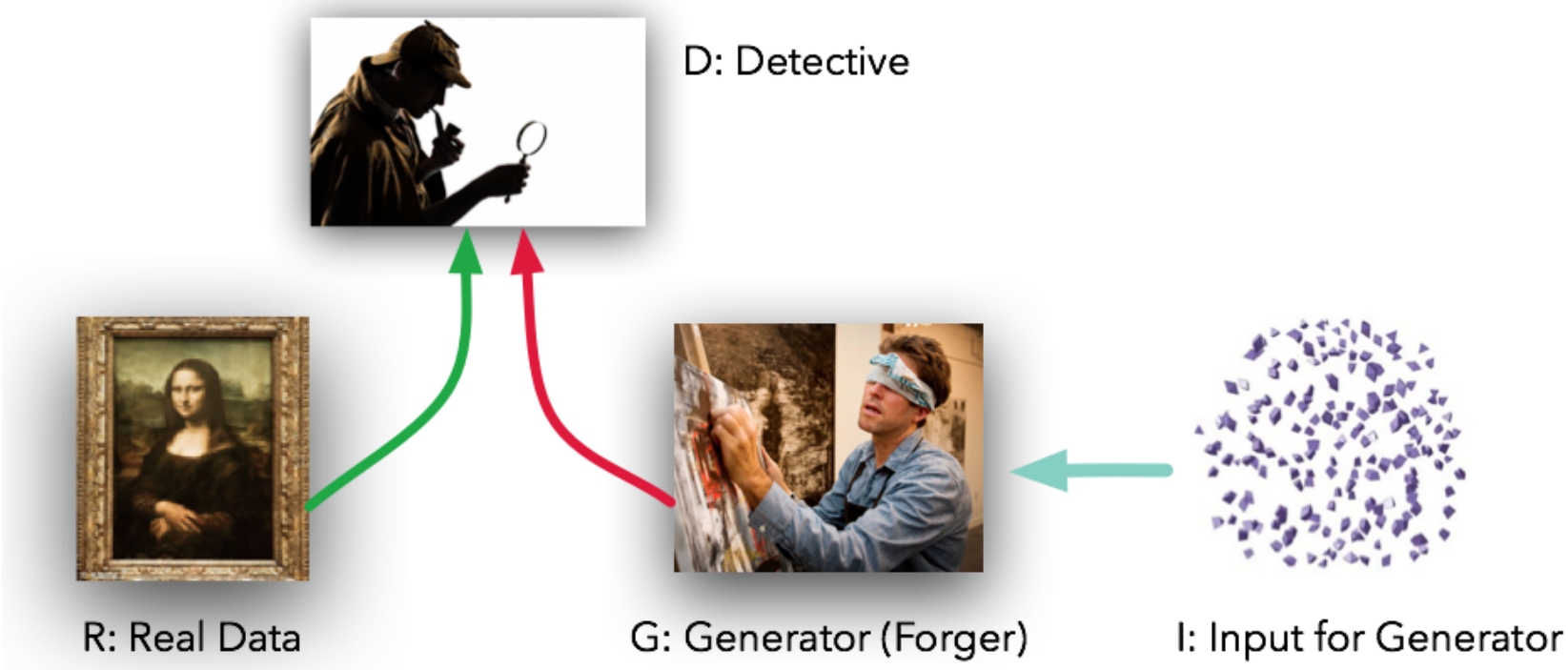


Super-resolution

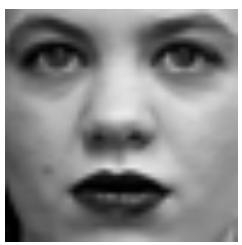
Talk outline

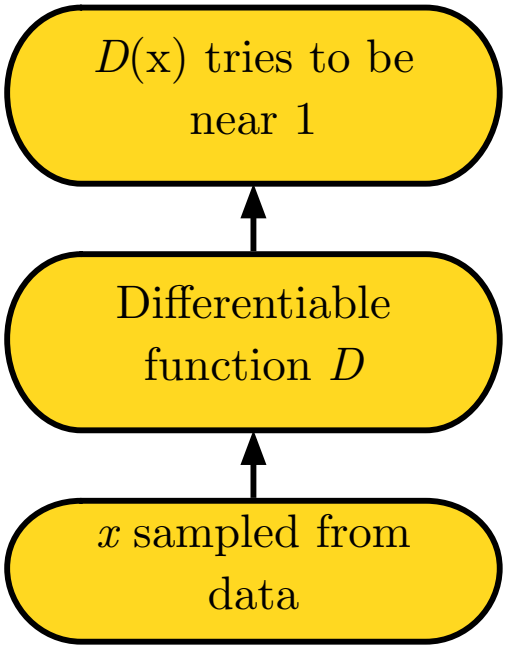
Basic idea

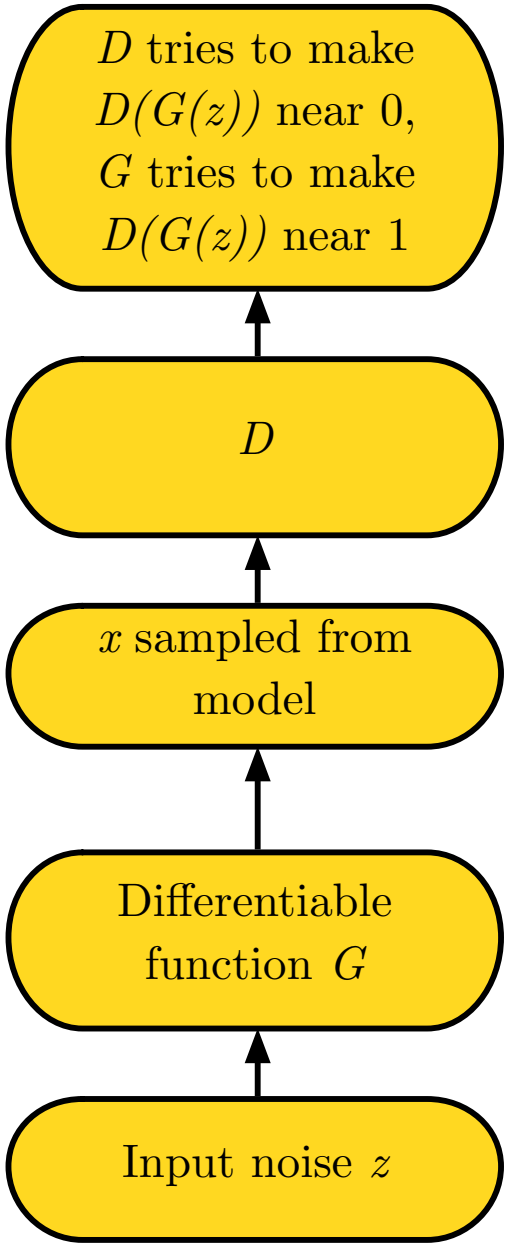
Imagine we have two **adversaries** (represented by **networks**):
detective (**discriminator**) and forger (**generator**)
They compete until forger learns to paint pictures close to real



Basic idea




D(x) tries to be near 1
Differentiable function D
x sampled from data


D tries to make $D(G(z))$ near 0, G tries to make $D(G(z))$ near 1
D
x sampled from model
Differentiable function G
Input noise z

Training objective

This is a two-player game with the following objective:

$$\min_G \max_D \mathbb{E}_{x \sim p_{\text{data}}} \log[D(x)] + \mathbb{E}_{z \sim p_z} \log[1 - D(G(z))]$$

From discriminator's point of view it is equivalent to maximizing data log-likelihood!

θ_G, θ_D — parameters of generator and discriminator respectively,
J_G, J_D — their loss functions

Loss functions

$$J_D = -\frac{1}{2}\mathbb{E}_{x\sim p_{\text{data}}} \log[D(x)] - \frac{1}{2}\mathbb{E}_{z\sim p_z} \log[1 - D(G(z))]$$

Regular cross-entropy

$$J_G = -J_D$$

Zero-sum aka minimax game

- Resembles minimizing Jensen-Shannon divergence between p_{data} and p_{model}

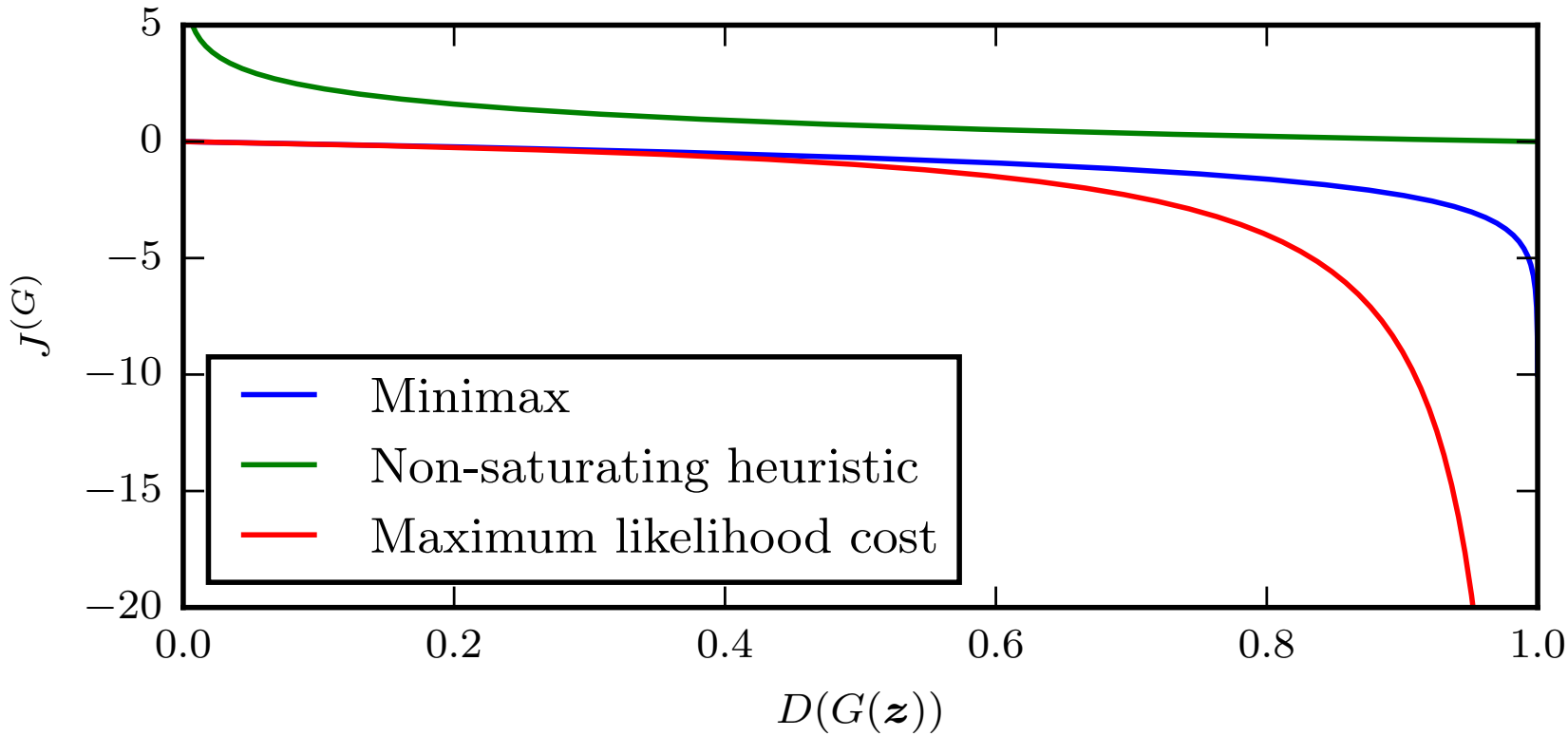
$$JSD(P, Q) = \frac{1}{2} D_{KL}\left(P \middle|\middle| \frac{P+Q}{2}\right) + \frac{1}{2} D_{KL}\left(Q \middle|\middle| \frac{P+Q}{2}\right)$$

- Have convergence guarantees (if updates are done in function space, not NN parameters)

Seems good, but there is a problem

Alternative generator losses

Early on, when model outputs are clearly identifiable, the gradient vanishes. What if we flip the expression inside $\log[1 - D(G(z))]$ instead of flipping the sign?

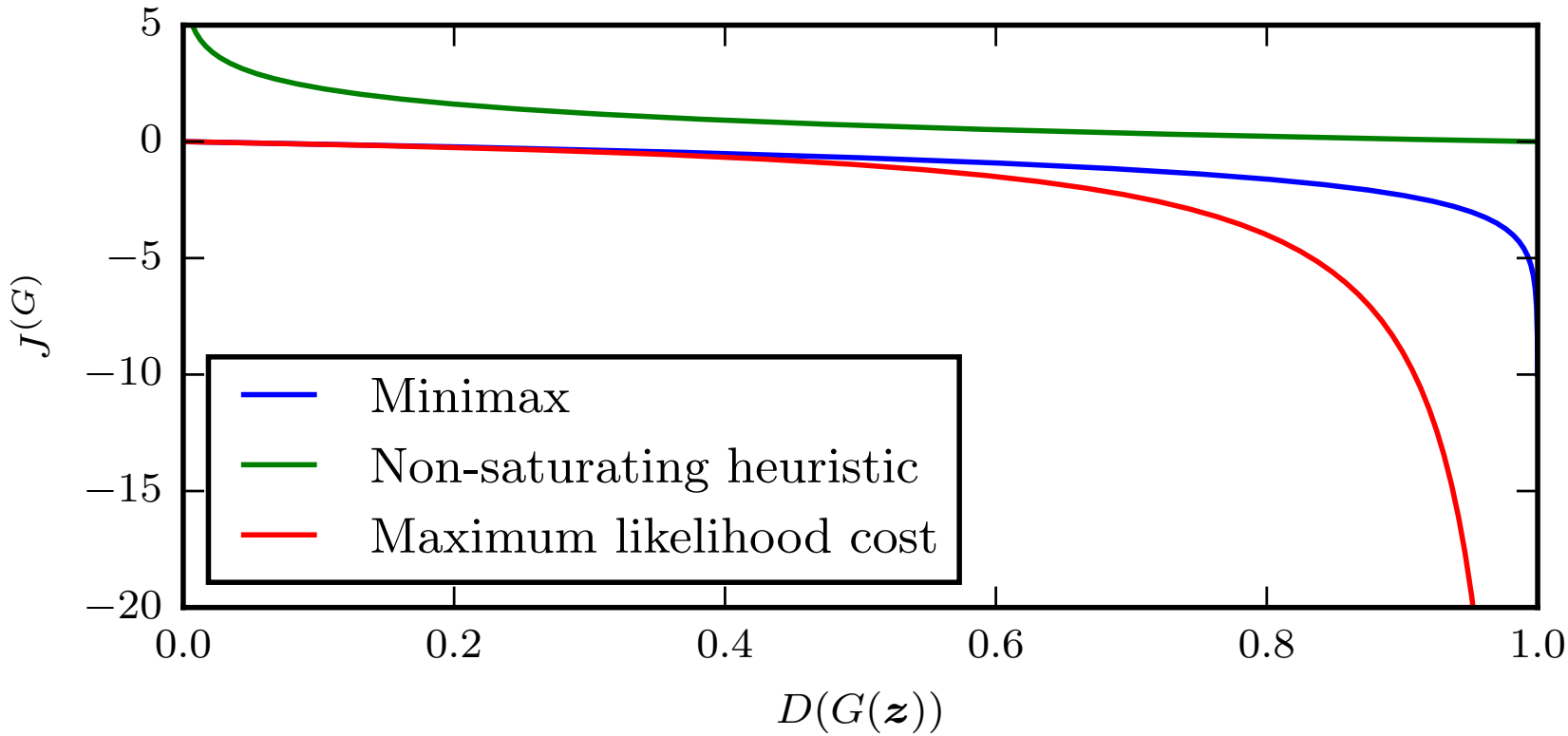


Comparison of generator losses

Alternative generator losses

Our new non-saturating loss is

$$J_G = -\frac{1}{2}\mathbb{E}_{z\sim p_z} \log[D(G(z))]$$

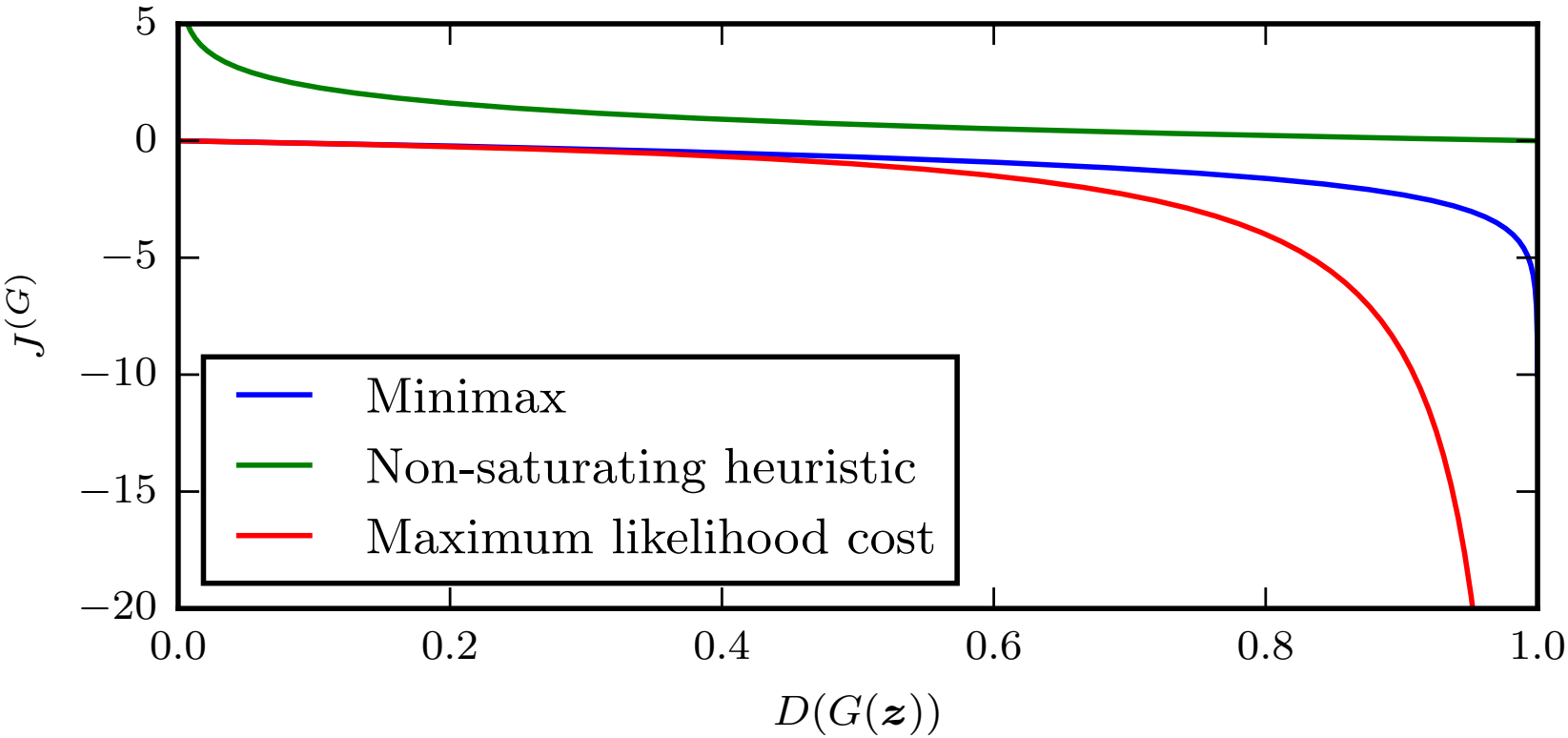


Comparison of generator losses

Alternative generator losses

Suppose we want to optimize likelihood with generator as well. Then, if D is optimal and its activation is $\sigma(x)$ (regular sigmoid), we can use

$$J_G = -\frac{1}{2}\mathbb{E}_{z\sim p_z} \exp\left(\sigma^{-1}(D(G(z)))\right)$$

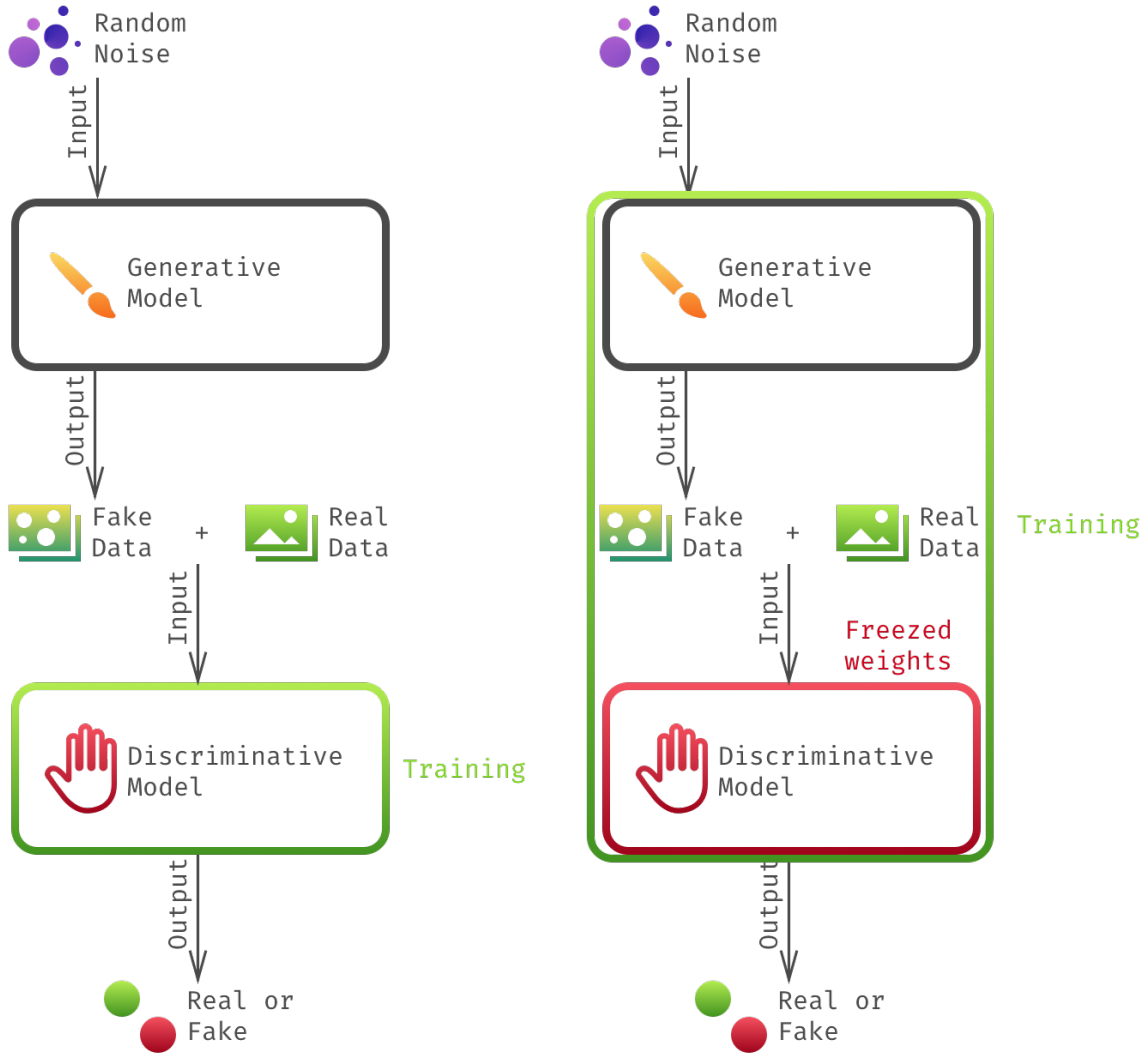


Comparison of generator losses

Training procedure

Simultaneous SGD: sample two batches of real and generated data, then at the same time:

- update θ_D to reduce J_D
- update θ_G to reduce J_G



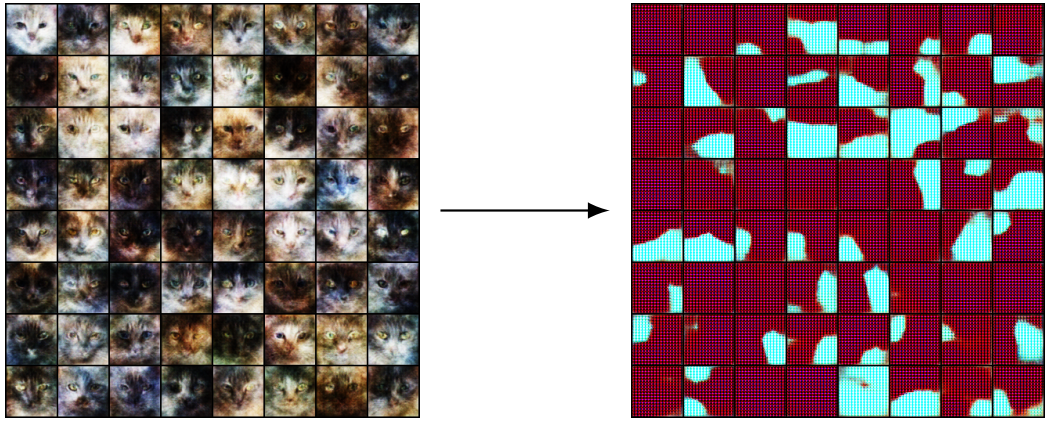
Conclusion

Pros of GANs

- A powerful technique for complex distributions approximation
- Unparalleled visual quality compared to other methods
- Generator loss function is learned by discriminator

Cons of GANs

- Finding equilibrium in high-dimensional non-convex games is difficult
- No reliable evaluation metrics (yet)
- Very tricky to train



References I

Generative Adversarial Networks
Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio
arXiv:1406.2661 [stat.ML]

NIPS 2016 Tutorial: Generative Adversarial Networks
Ian Goodfellow
arXiv:1701.00160 [cs.LG]

Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks
Alec Radford, Luke Metz, Soumith Chintala
arXiv:1511.06434 [cs.LG]

References II


Pattern Recognition and Machine Learning
Christopher M. Bishop
Springer, 2011


Machine Learning: A Probabilistic Perspective
Kevin P. Murphy
MIT Press, 2012


Generative Adversarial Networks (GANs) in 50 lines of code (PyTorch)
Dev Nag
https://medium.com/@devnag/generative-adversarial-networks-gans-in-50-lines-of-code-pytorch-e81b79659e3f

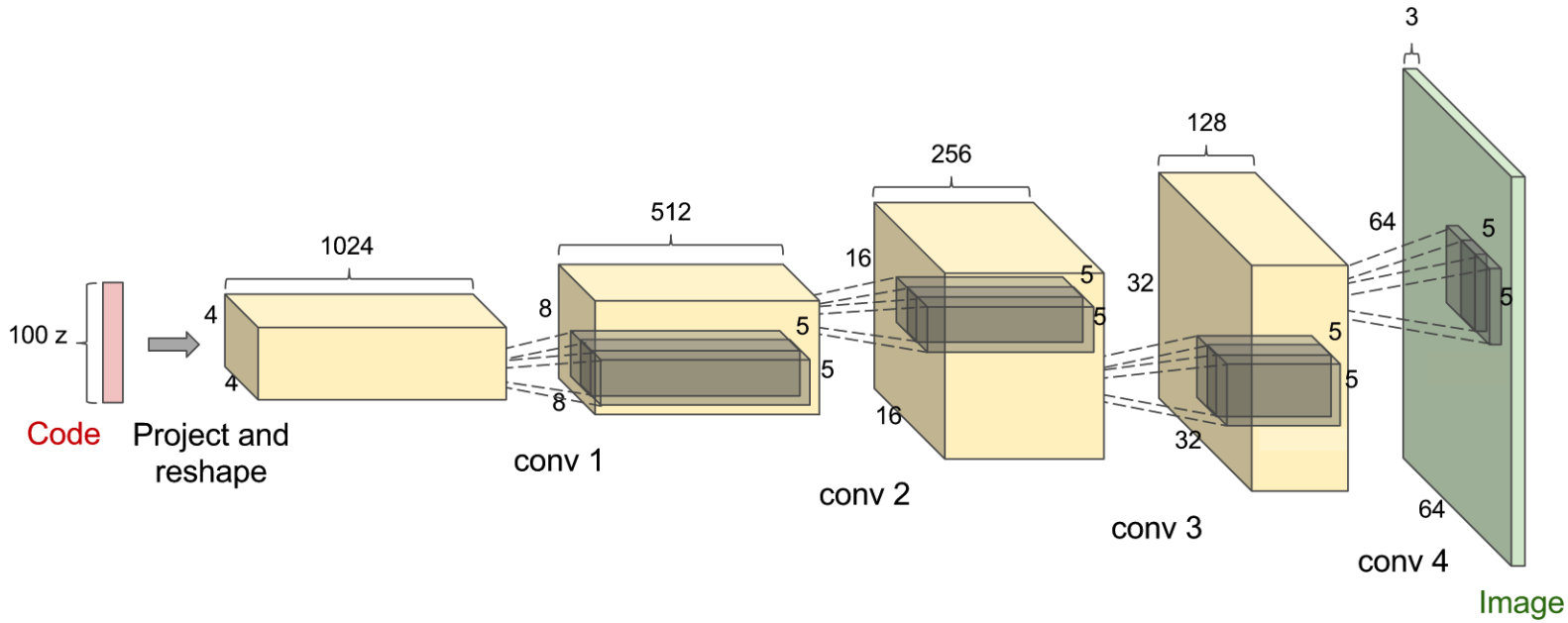

Generative Adversarial Networks (GANs), Some Open Questions)
Sanjeev Arora
http://www.offconvex.org/2017/03/15/GANs/

References III


Generative Adversarial Networks Part 1 - Understanding GANs
Robin Ricard
http://www.rricard.me/machine/learning/generative/adversarial/netw
part1.html

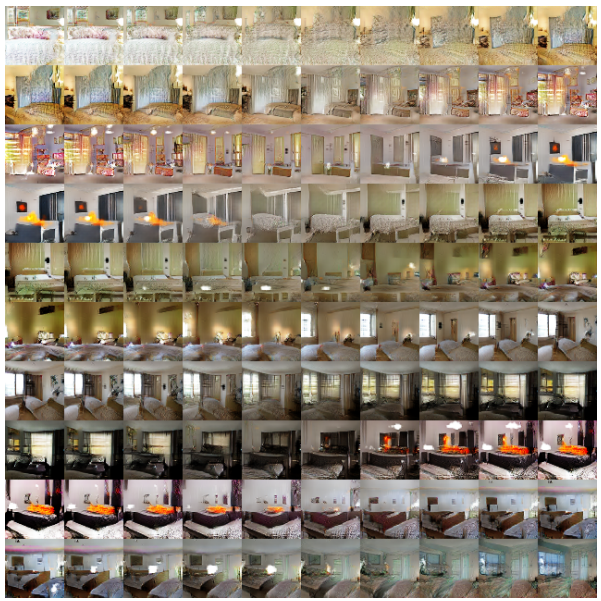

Stanford University CS231n: Convolutional Neural Networks for Visual Recognition Lecture 13: Generative Models
Fei-Fei Li, Justin Johnson, Serena Yeung L
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Bonus: DCGAN

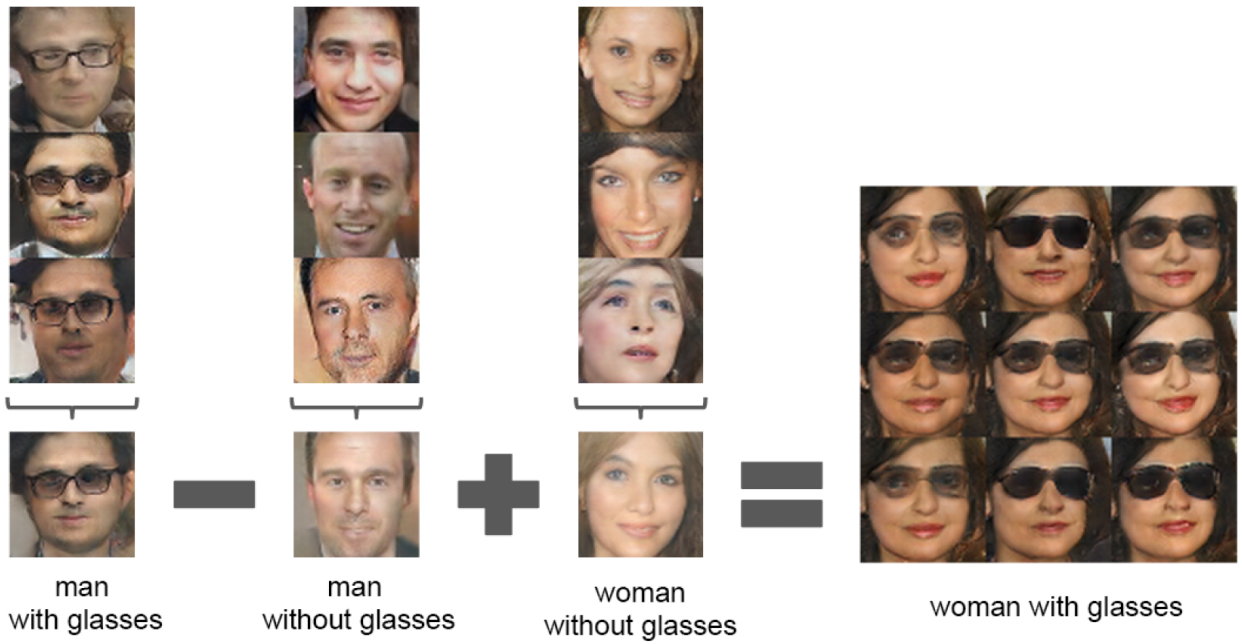


- Batch normalization for almost all layers in encoder and decoder (except last generator and first discriminator layers)
- ReLU in generator, LeakyReLU in discriminator
- All-convolutional networks
- Adam instead of momentum SGD

Representation properties



Smooth latent space transitions



Vector arithmetic