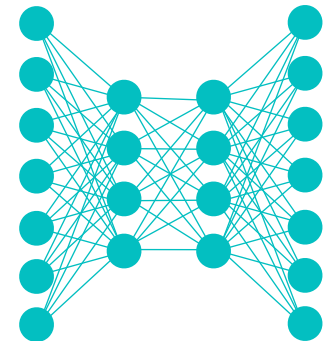


Lecture Notes for **Neural Networks and Machine Learning**



Practical GANs
and LSGAN



Logistics and Agenda

- Logistics
 - Student Paper: Glow (two lectures away)
- Agenda
 - LSGAN
 - Practical GANs
 - Wasserstein GAN (next time)
 - WGAN-GP (next time)
 - BigGAN (next next time)
 - More GAN Examples (Holy GAN-zooks)



LSGAN

Least Squares Generative Adversarial Networks

Xudong Mao^{*1}, Qing Li^{†1}, Haoran Xie^{‡2}, Raymond Y.K. Lau^{§3},
Zhen Wang^{¶4}, and Stephen Paul Smolley^{||5}

¹Department of Computer Science, City University of Hong Kong

²Department of Mathematics and Information Technology, The
Education University of Hong Kong

³Department of Information Systems, City University of Hong
Kong

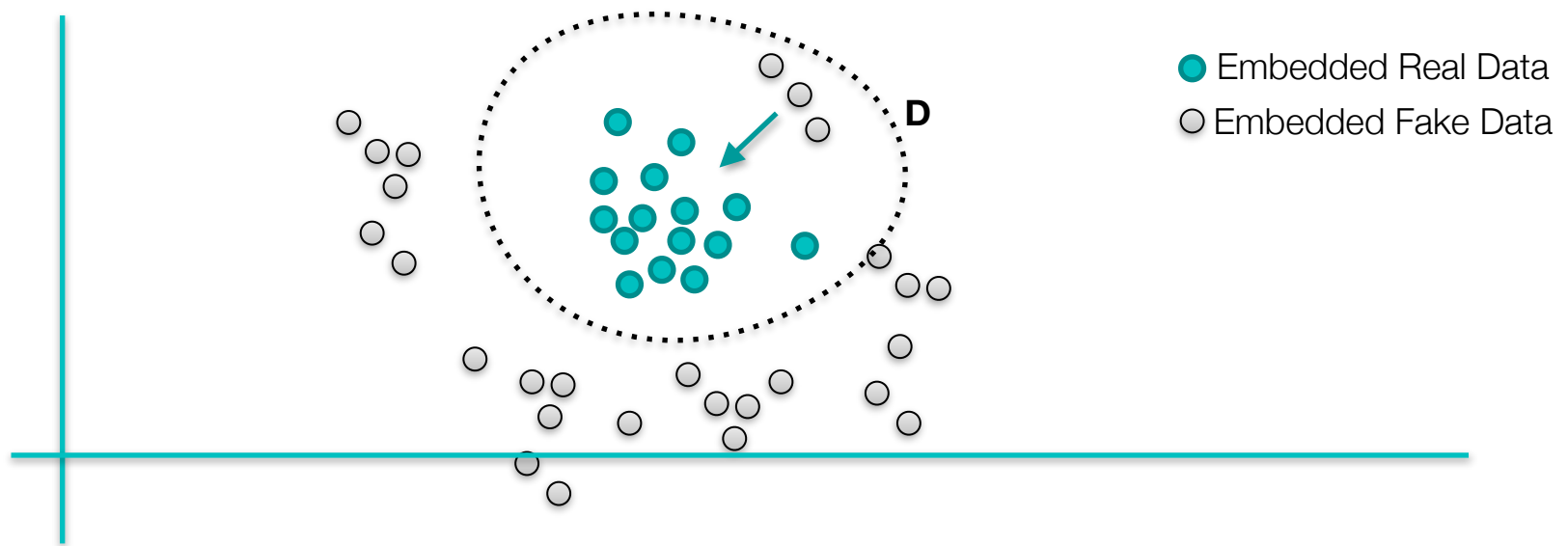
⁴Center for OPTical IMagery Analysis and Learning (OPTIMAL),
Northwestern Polytechnical University

⁵CodeHatch Corp.



The Least Squares GAN

- **Observation:** Generated points may (by chance) be classified as real by Discriminator—but they are still not representative of the real data
- **Solution:** Incentivize even correctly classified labels to move toward real data distribution



Incentivizing with Least Squares

- Assume a =fake label, b =real label, c =misleading label
- The new loss function is:

$$\min_D V_{\text{LSGAN}}(D) = \frac{1}{2} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [(D(\mathbf{x}) - b)^2] + \frac{1}{2} \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [(D(G(\mathbf{z})) - a)^2]$$

$$\min_G V_{\text{LSGAN}}(G) = \frac{1}{2} \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [(D(G(\mathbf{z})) - c)^2],$$

- Here we can take advantage of the labels, even when they are classified correct/incorrect
 - ...because we have a distance to margin
 - ...that's it!



But that results is not publishable!

- We need to find a way to complicate the math to make it less approachable and therefore publishable
 - Yay for ethics!
- **Discussion:** is this wrong for the authors to do?

In the original GAN paper [7], the authors has shown that minimizing Equation 1 yields minimizing the Jensen-Shannon divergence:

$$C(G) = KL\left(p_{\text{data}} \left\| \frac{p_{\text{data}} + p_G}{2} \right\| \right) + KL\left(p_G \left\| \frac{p_{\text{data}} + p_G}{2} \right\| \right) - \log(4). \quad (3)$$

Here we also explore the relation between LSGANs and f-divergence. Consider the following extension of Equation 2:

$$\begin{aligned} \min_D V_{\text{LSGAN}}(D) &= \frac{1}{2} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [(D(\mathbf{x}) - b)^2] + \frac{1}{2} \mathbb{E}_{\mathbf{z} \sim p_G(\mathbf{z})} [(D(G(\mathbf{z})) - a)^2] \\ \min_G V_{\text{LSGAN}}(G) &= \frac{1}{2} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [(D(\mathbf{x}) - c)^2] + \frac{1}{2} \mathbb{E}_{\mathbf{z} \sim p_G(\mathbf{z})} [(D(G(\mathbf{z})) - c)^2]. \end{aligned} \quad (4)$$

Note that adding the term $\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [(D(\mathbf{x}) - c)^2]$ to $V_{\text{LSGAN}}(G)$ does not change the optimal values since this term does not contain parameters of G .

We first derive the optimal discriminator D for a fixed G as below :

$$D^*(\mathbf{x}) = \frac{bp_{\text{data}}(\mathbf{x}) + ap_G(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_G(\mathbf{x})}. \quad (5)$$

In the following equations we use p_d to denote p_{data} for simplicity. Then we can reformulate Equation 4 as follows:

$$\begin{aligned} 2C(G) &= \mathbb{E}_{\mathbf{x} \sim p_d} [(D^*(\mathbf{x}) - c)^2] + \mathbb{E}_{\mathbf{z} \sim p_g} [(D^*(\mathbf{z}) - c)^2] \\ &= \mathbb{E}_{\mathbf{x} \sim p_d} \left[\left(\frac{bp_d(\mathbf{x}) + ap_g(\mathbf{x})}{p_d(\mathbf{x}) + p_g(\mathbf{x})} - c \right)^2 \right] + \mathbb{E}_{\mathbf{z} \sim p_g} \left[\left(\frac{bp_d(\mathbf{z}) + ap_g(\mathbf{z})}{p_d(\mathbf{z}) + p_g(\mathbf{z})} - c \right)^2 \right] \\ &= \int_{\mathcal{X}} p_d(\mathbf{x}) \left(\frac{(b-c)p_d(\mathbf{x}) + (a-c)p_g(\mathbf{x})}{p_d(\mathbf{x}) + p_g(\mathbf{x})} \right)^2 d\mathbf{x} + \int_{\mathcal{X}} p_g(\mathbf{x}) \left(\frac{(b-c)p_d(\mathbf{x}) + (a-c)p_g(\mathbf{x})}{p_d(\mathbf{x}) + p_g(\mathbf{x})} \right)^2 d\mathbf{x} \\ &= \int_{\mathcal{X}} \frac{((b-c)p_d(\mathbf{x}) + (a-c)p_g(\mathbf{x}))^2}{p_d(\mathbf{x}) + p_g(\mathbf{x})} d\mathbf{x} \\ &= \int_{\mathcal{X}} \frac{((b-c)(p_d(\mathbf{x}) + p_g(\mathbf{x})) - (b-a)p_g(\mathbf{x}))^2}{p_d(\mathbf{x}) + p_g(\mathbf{x})} d\mathbf{x}. \end{aligned} \quad (6)$$

If we set $b - c = 1$ and $b - a = 2$, then

$$\begin{aligned} 2C(G) &= \int_{\mathcal{X}} \frac{(2p_g(\mathbf{x}) - (p_d(\mathbf{x}) + p_g(\mathbf{x})))^2}{p_d(\mathbf{x}) + p_g(\mathbf{x})} d\mathbf{x} \\ &= \chi^2_{\text{Pearson}}(p_d + p_g \| 2p_g), \end{aligned} \quad (7)$$

where χ^2_{Pearson} is the Pearson χ^2 divergence. Thus minimizing Equation 4 yields minimizing the Pearson χ^2 divergence between $p_d + p_g$ and $2p_g$ if a , b , and c satisfy the conditions of $b - c = 1$ and $b - a = 2$.



LS-GAN Parameter Selection

3.2.3 Parameters Selection

One method to determine the values of a , b , and c in Equation [2] is to satisfy the conditions of $b - c = 1$ and $b - a = 2$, such that minimizing Equation [2] yields minimizing the Pearson χ^2 divergence between $p_d + p_g$ and $2p_g$. For example, by setting $a = -1$, $b = 1$, and $c = 0$, we get the following objective functions:

$$\begin{aligned}\min_D V_{\text{LSGAN}}(D) &= \frac{1}{2} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [(D(\mathbf{x}) - 1)^2] + \frac{1}{2} \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [(D(G(\mathbf{z})) + 1)^2] \\ \min_G V_{\text{LSGAN}}(G) &= \frac{1}{2} \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [(D(G(\mathbf{z})))^2].\end{aligned}\tag{8}$$

Another method is to make G generate samples as real as possible by setting $c = b$. For example, by using the 0-1 binary coding scheme, we get the following objective functions:

$$\begin{aligned}\min_D V_{\text{LSGAN}}(D) &= \frac{1}{2} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [(D(\mathbf{x}) - 1)^2] + \frac{1}{2} \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [(D(G(\mathbf{z})))^2] \\ \min_G V_{\text{LSGAN}}(G) &= \frac{1}{2} \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [(D(G(\mathbf{z})) - 1)^2].\end{aligned}\tag{9}$$

In practice, we observe that Equation [8] and Equation [9] show similar performance. Thus either one can be selected. In the following sections, we use Equation [9] to train the models.



LS-GAN Results

- Some takeaways:
 - RMSProp works better than Adam
 - Reasonable values for a, b, c have similar performance
 - Mode collapse is still a problem, but not nearly as bad as regular GANs

Experiment:

Generate 2D samples from known Mix of Gaussian Distributions, then train 3 layer GANs to generate the same 2D data.

Does one learn the distribution?

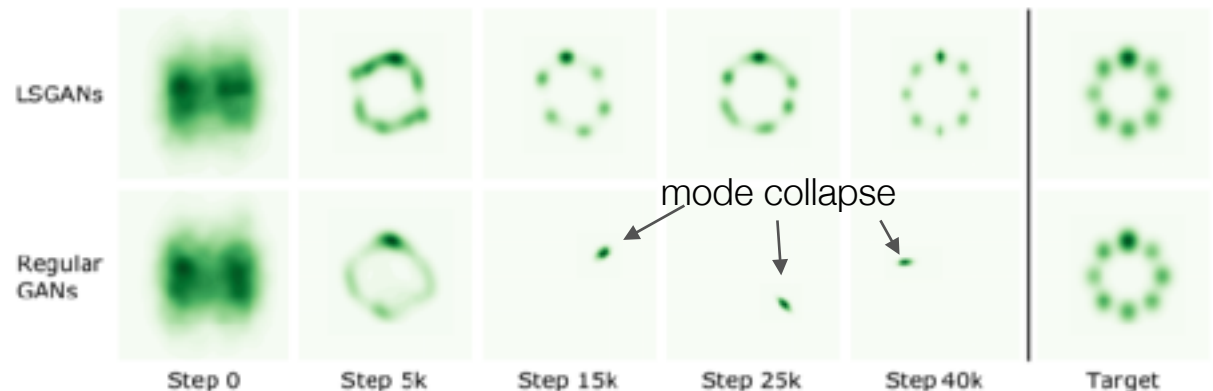
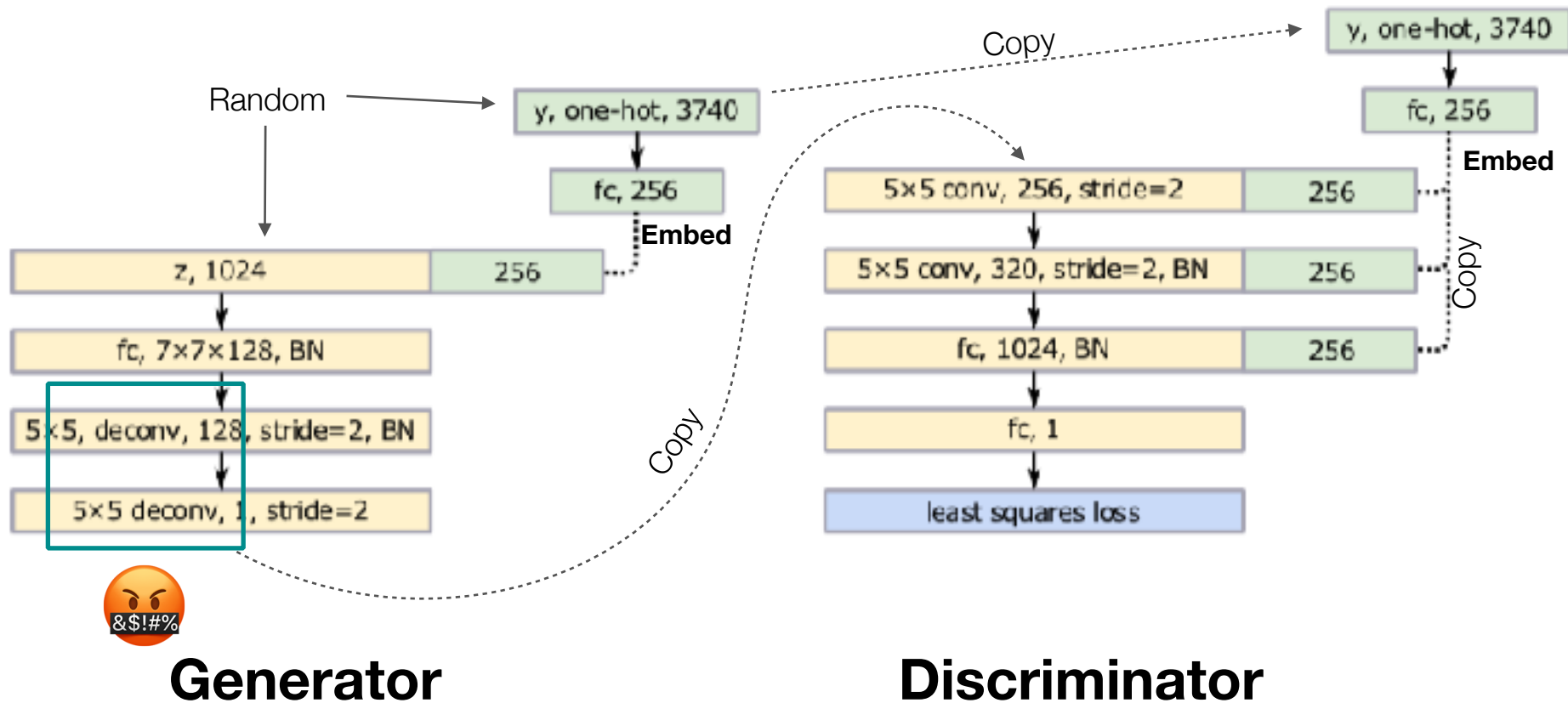


Figure 8: Dynamic results of Gaussian kernel estimation for LSGANs and regular GANs. The final column shows the real data distribution.



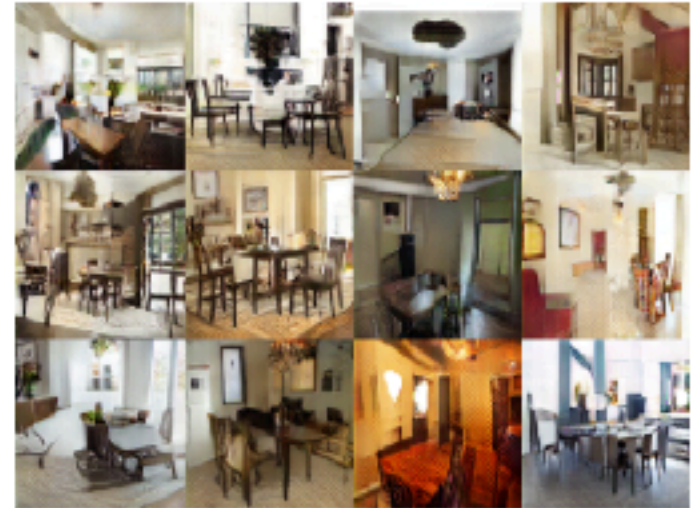
Architecture Employed



Qualitative Results



(a) Church outdoor.



(b) Dining room.



(c) Kitchen.



(d) Conference room.



More Qualitative Results



(a) Generated by LSGANs.



(b) Generated by DCGANs (Reported in [11]).



How can we trust the results?

- Do we trust that the authors are not cherry picking the results?
- If I perform random seed optimization and run my algorithms for longer, can I always claim its better
 - ...but maybe not for the reasons I publish...
- Without strong quantitative evaluation criteria for image generation, can there be a solution to this?
 - Require human subjects evaluation?
 - But can't they still tune the results for their algorithm before the human subjects testing?
 - What about open sourcing the weights of a tuned algorithm?



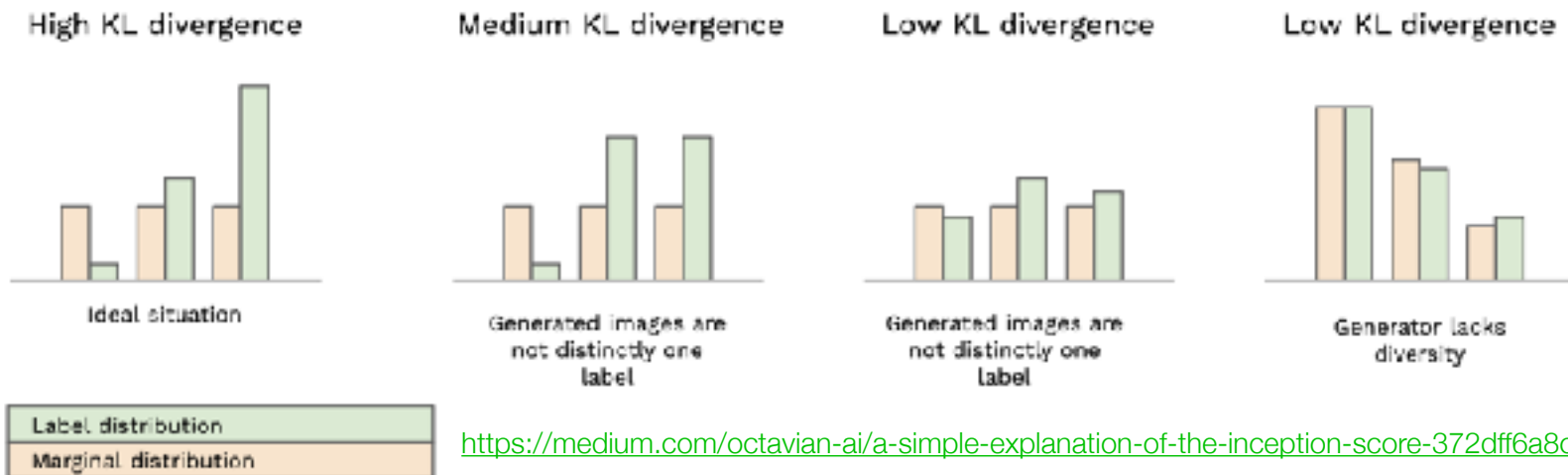
An Accepted Measure: Inception Score

$$\hat{p}(y) = \frac{1}{N} \sum_i p(y | \mathbf{x}^{(i)})$$

Expected class distribution through a trained CNN, like VGG should be **nearly uniform** in ideal case

$$IS(G) \approx \exp \left(\frac{1}{N} \sum_i D_{KL} (p(y | \mathbf{x}^{(i)}) || \hat{p}(y)) \right)$$

average KL Divergence of generated images with marginal, should **differ dramatically**, ideally



<https://medium.com/octavian-ai/a-simple-explanation-of-the-inception-score-372dff6a8c7a>



LSGAN Inception Score

Discriminator Feature-based Inference by Recycling the Discriminator of GANs

Generative adversarial networks (GANs) successfully generate high quality data by learning a mapping from a latent vector to the data. Various studies assert that the latent space of a GAN is semantically meaningful and can be utilized for advanced

Metric	DCGAN	LSGAN
Inception score	6.50	5.98

Higher is better

<https://paperswithcode.com/paper/high-quality-bidirectional-generative/review/>

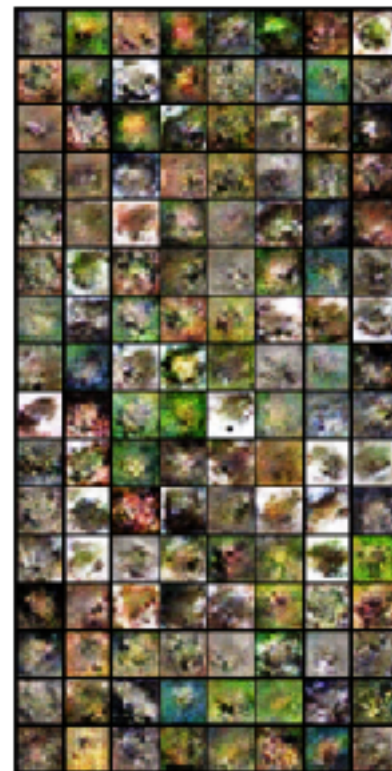
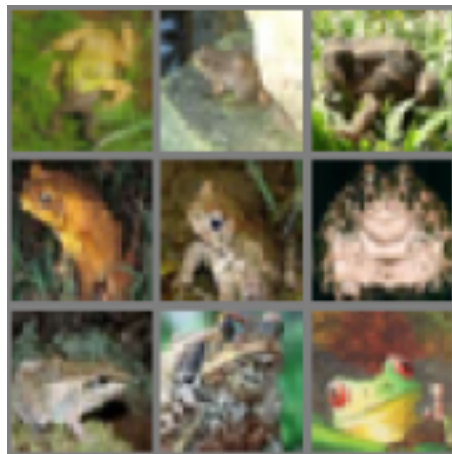




GANs in PyTorch

Master Repository:

[07c GANsWithTorch.ipynb](#)



Revisiting Demo with the LS-GAN architecture



GANs Loss

Abstract

Recent work in meta-learning has set the deep learning community alight. From minute gains on few-shot learning tasks, to discovering architectures that are slightly better than chance, to solving intelligence itself¹, meta-learning is proving a popular solution to every conceivable problem ever conceivably conceived ever. In this paper we venture deeper into the computational insanity that is meta-learning, and potentially risk exiting the simulation of reality itself, by attempting to meta-learn at a third learning level. We showcase the resulting approach—which we call *meta-meta-learning*—for neural architecture search. Crucially, instead of *meta-learning* a neural architecture differentially as in DARTS (Liu et al., 2018) we *meta-meta-learn* an architecture by searching through arXiv. This *arXiv descent* is GPU-free and only requires a handful of graduate students. Further, we introduce a regulariser, called *college-dropout*, which works by randomly removing a single graduate student from our system. As a consequence, procrastination levels decrease significantly, due to the increased workload and sense of responsibility each student attains.

The code for our experiments is publicly available at [\[redacted\]](#).

Edit: we have decided not to release our code as we are concerned that it may be used for malicious purposes.

Meta-meta-learning for Neural Architecture Search through arXiv Descent

Andreas Antoniou
MetaMind
aa@mm.ai

Nick Pawlowski
Google x²
nick@x.z

Jack Turner
slow.ai
jack@slow.ai

James Owers
Facebook AI Research Team
jim@fart.org

Joseph Mellor
Institute of Yellow Jumpers
joe@anditwasall.yellow

Elliot J. Crowley
ClosedAI
elliott@closed.ai



Why are GANs so difficult to train?

- Why did we need to add noise to labels ?
- Why were sparse gradients needed?
- Does using least squares really solve the problem?
- Formalization:
 - GANs will not converge
 - GAN outputs all might be similar (**mode collapse**)
 - Slow training: gradient vanishes, sometimes not recoverable
 - Theory behind this is still developing, but a good start:

Generative Adversarial Networks (GANs): What it can generate and What it cannot?

P. Manisha
manisha.padala@research.iiit.ac.in

Sujit Gujar
sujit.gujar@iiit.ac.in

TOWARDS PRINCIPLED METHODS FOR TRAINING
GENERATIVE ADVERSARIAL NETWORKS

Martin Arjovsky
Courant Institute of Mathematical Sciences
martinarjovsky@gmail.com

Léon Bottou
Facebook AI Research
leonb@fb.com



The Optimal Discriminator

- Discriminator is maximizing:

$$\max_d \mathbf{E}_{x \leftarrow p_{data}} [\log d(x)] + \mathbf{E}_{x \leftarrow g(z)} [\log(1 - d(x))]$$

$$\max_d \int_{x \in [P_{data}, P_g]} \left(p_{data}(x) \cdot \log d(x) + p_g(x) \cdot \log(1 - d(x)) \right) dx$$

$$\nabla_f = 0 = \frac{1}{\partial d(x)} \left(p_{data}(x) \cdot \log d(x) + p_g(x) \cdot \log(1 - d(x)) \right)$$

... math ...

$$d(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}$$

-Conclusion: Discriminator is optimal when this condition occurs.

Generator is optimal when

p_g and p_{data} are close together.

So we want to optimize the overlap of these distributions



Optimality of Generator

- The optimal generator for $d(x) = p_{data} / (p_{data} + p_g)$ is:

$$C(G) = -\log(4) + KL\left(p_{data} \parallel \frac{p_{data} + p_g}{2}\right) + KL\left(p_g \parallel \frac{p_{data} + p_g}{2}\right)$$

$$C(G) = -\log(4) + 2.JSD(p_{data} \parallel p_g) \quad \text{Jenson-Shannon Divergence}$$

- But that is not what we optimize, we optimize (for better gradients):

$$\max_{G_\theta} \log(D(G(z))) \text{ rather than training } G \text{ to minimize } \log(1 - D(G(z)))$$

- Vanishing gradients: if $p_{data} \parallel p_g \approx 0$ then JSD saturates and the gradient is basically zero. Optimization has no idea what direction to move in...
- What if we use some tricks to keep the optimization moving even when we do not know which direction to go?



GAN Tricks

When your GAN suffers from mode collapse



The teacher adding "and just have fun!" at the end of the assignment :



The Art of Science: Using tricks

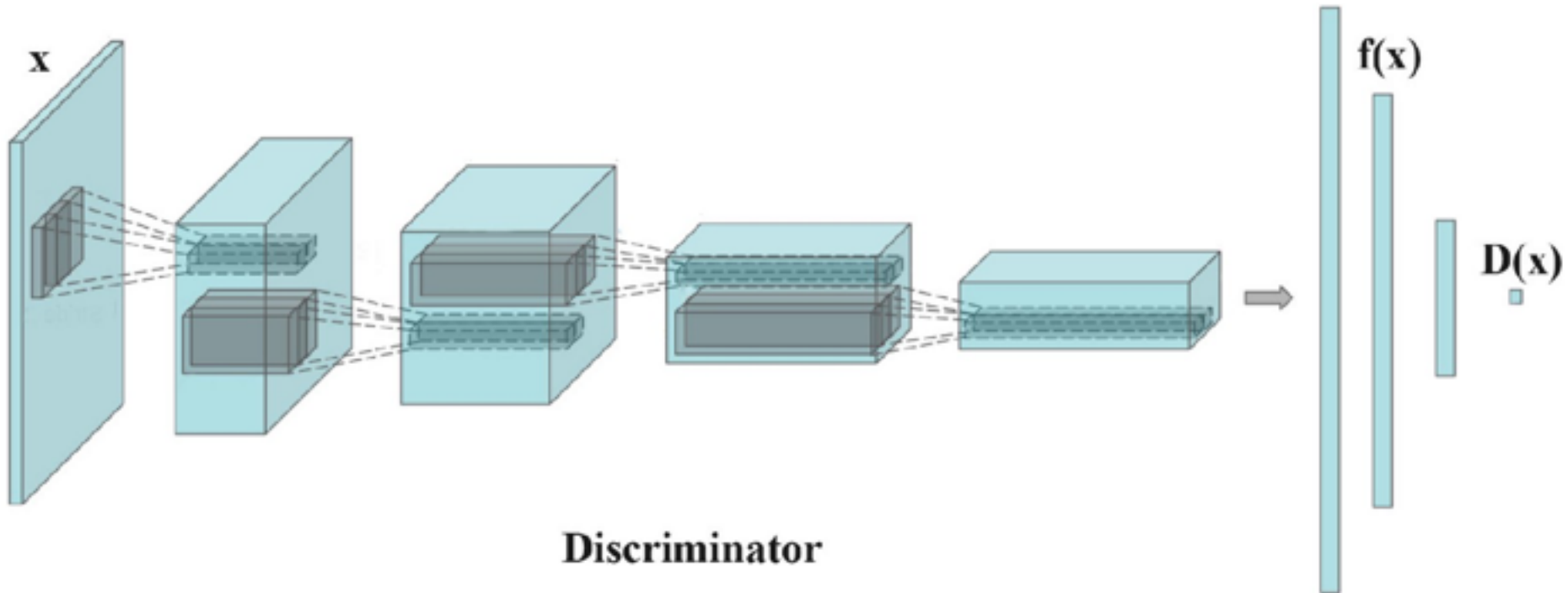
- From Salimans, Goodfellow, et al., :
 - *In this work, we introduce several techniques intended to encourage convergence of the GANs game. These techniques are motivated by a heuristic understanding of the non-convergence problem. They lead to improved semi-supervised learning performance and improved sample generation. We hope that some of them may form the basis for future work, providing formal guarantees of convergence.*
- A collection of Heuristics and Observations that is more Alchemy than Science
 - Feature Matching loss
 - Mini-batch discrimination (*we will skip this...*)
 - Historical averaging
 - Virtual Batch normalization
 - Experience Replay
 - Manipulating the Loss Function (*next lecture*)

Improved Techniques for Training GANs

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen

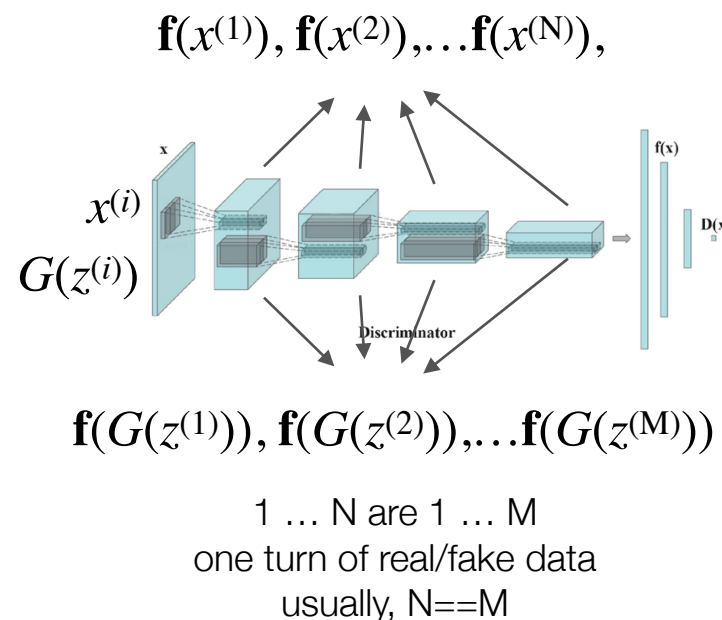


Before we go too far



Feature Matching

- Loss function for generator is really difficult and unstable
 - hard to optimize based on feedback only from discriminator output!
- Since generator is trying to statistically match features inside the discriminator (ideally, to fool it)
 - try to make generator match statistics of discriminator activations



$$\left\| \mathbf{E}_{x \leftarrow \text{data}}[\mathbf{f}(x)] - \mathbf{E}_{x \leftarrow q(z)}[\mathbf{f}(G(z))] \right\|^2$$

mean discriminator activations from **real** data


mean discriminator activations from **fake** data

```
real_mean = K.reduce_mean(real_features, axis = 0)
fake_mean = K.reduce_mean(fake_features, axis = 0)
feat_match = K.reduce_mean(tf.square(real_mean-fake_mean))
```



Historical Averaging

- Because we cannot converge, parameters should constantly be exploring different solutions and exploiting current found solutions
- Solution: Add explore/exploit tradeoff directly in loss function

$$\mathcal{L}(w)_{orig} - \lambda_T \left\| \underset{\text{current weights}}{w[T]} - \underset{\text{average weights of model over window}}{\frac{1}{n} \sum_{i=T-n}^{T-1} w[t]} \right\|^2$$


start: incentivize new weights to be different. Large value.

end: reduce term to zero so that we exploit current solutions.



Review: Batch Normalization

- Normalize and rescale activations before going to new layer

One Batch: $\mu_B = \frac{1}{m} \sum_{i=1}^m \underbrace{x_i}_{\text{input}}$

$$\sigma_B = \frac{1}{m} \sum_{i=1}^m (\mu_B - x_i)^2$$

Normalized x: $\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$

Scaled activations
with trainable
parameters:

$$y = \gamma \hat{x} + \beta$$

During inference: μ_B and σ_B are fixed based on the training set population



Virtual Batch Normalization

- Batch normalization is really swell
- But for GANs, it makes current x's dependent on other generated x's (and it was already hard to train...)
- Why not use a reference batch for statistics? And just use statistics of the reference batch activations for normalizing?
 - Every time we update W's in the network, we need to recompute the statistics for the reference batch
 - That's it!

$$\mu^{ref} = \frac{1}{M} \sum_i a_i^{ref} \quad \sigma^{ref^2} = \frac{1}{M} \sum_i (a_i^{ref} - \mu^{ref})^2 \quad z_i = \gamma \cdot \frac{(a_i - \mu^{ref})}{\sqrt{\sigma^{ref^2} + \epsilon}} + \beta$$



Experience Replay

- The discriminator can overpower the generator quickly and it over-learns to a particular epoch
- Solution: feed it more than just the current epoch
- Save previous epochs and randomly grab from them when updating the generator!
- Why might this work?
 - Because the gradient was zero, and this moves us around the latent space to (hopefully) find better gradient



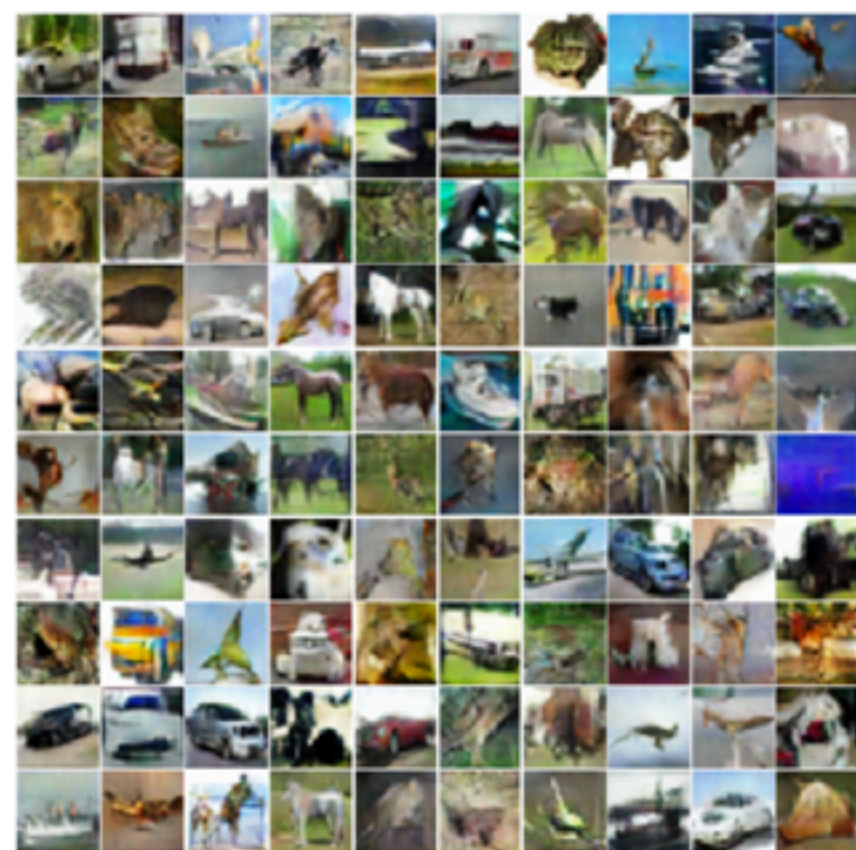


Figure 4: Samples generated during semi-supervised training on CIFAR-10 with feature matching (Section 3.1, *left*) and minibatch discrimination (Section 3.2, *right*).



Is there a better way than tricks?

- All of the things we have discussed help to move the optimization along even when **the output distribution does not overlap with the actual data distribution**
- Can we formalize this mathematically?
 - Some if it, yes. The Wasserstein GAN
 - Will discuss next lecture (or now, if time)

