



HOW TO BE A GANGSTA

CHRISTOPHER BECKHAM

WHAT ARE GANS?

- GANs are a class of algorithms used in unsupervised learning in which the goal is to learn (implicitly) the data distribution
- This is done through two adversaries (generator and discriminator) which compete to out-perform each other
- Discriminator detects real/fake, generator takes in noise and spits out data
- (Ideally) the game finishes when no network performs better than the other
- Theoretically, the game that GANs play is equivalent to minimising some measure of divergence between distribution learned by the generator and the data distribution

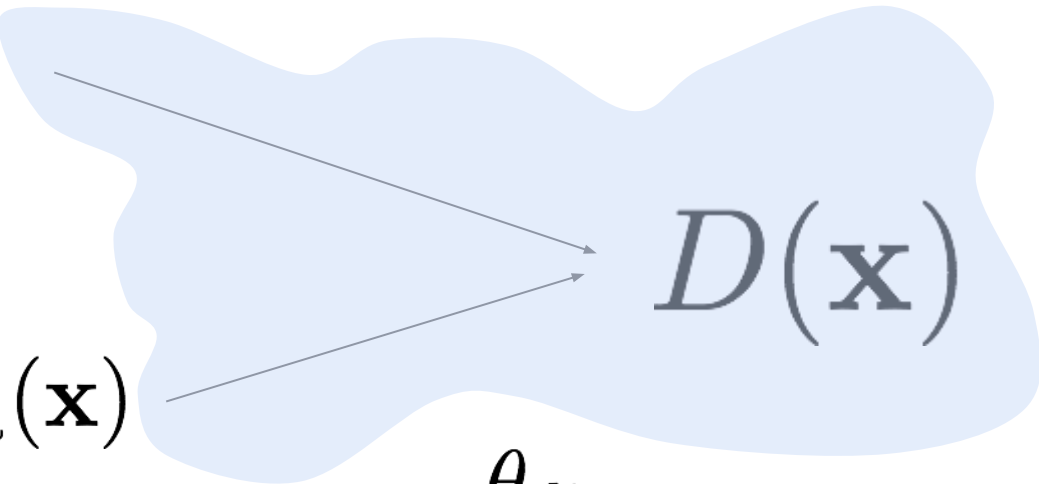
$$\mathbf{z} \sim p(\mathbf{z})$$



$$\tilde{\mathbf{x}} = G(\mathbf{z})$$

$$\mathbf{x} \sim p_{\text{data}}(\mathbf{x})$$

```
z = sample_z()  
x_fake = g(z)  
x_real = sample_data()  
d_fake, d_real = d(x_fake), d(x_real)  
loss = binary_xent(d_fake, 0) + binary_xent(d_real, 1)  
loss.backward()  
optim_d.step()
```



θ_{disc}

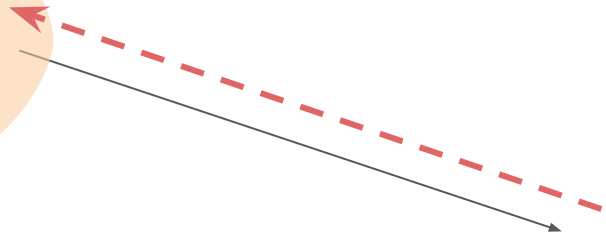
$$\mathbf{z} \sim p(\mathbf{z})$$



$$\tilde{\mathbf{x}} = G(\mathbf{z})$$

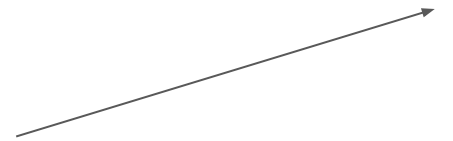
θ_{gen}

```
z = sample_z()  
x_fake = g(z)  
D_fake = d(x_fake)  
loss = binary_xent(d_fake, 1)  
loss.backward()  
optim_g.step()
```



$$D(\mathbf{x})$$

$$\mathbf{x} \sim p_{\text{data}}(\mathbf{x})$$



```
for iter in range(N):
```

```
    # train generator
```

```
    optim_g.zero_grad()
```

```
    optim_d.zero_grad()
```

```
    for _ in range(G_steps):
```

```
        z = sample_z()
```

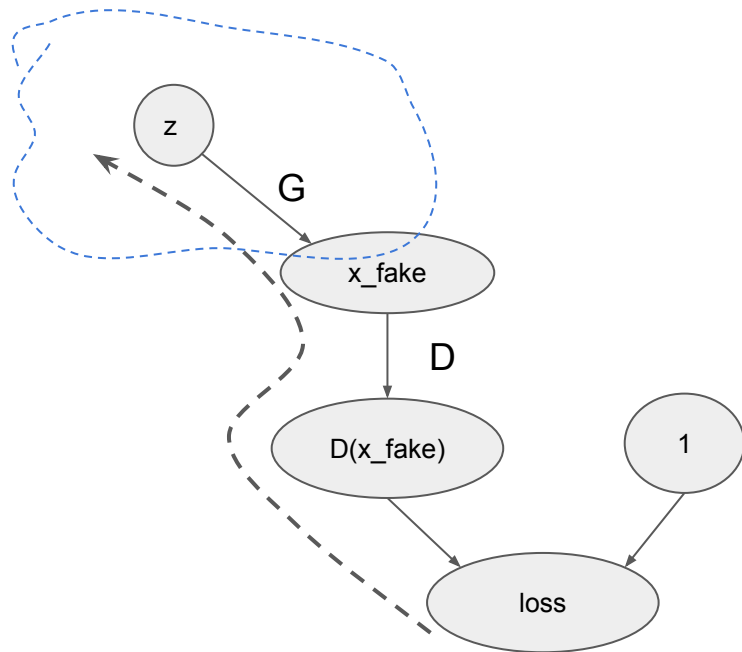
```
        x_fake = g(z)
```

```
        D_fake = d(x_fake)
```

```
        loss = binary_xent(d_fake, 1)
```

```
        loss.backward()
```

```
        optim_g.step()
```



```
for iter in range(N):
```

```
    # train generator
```

```
    ...
```

```
    # train discriminator
```

```
    optim_d.zero_grad()
```

```
    for _ in range(D_steps):
```

```
        z = sample_z()
```

```
        x_fake = g(z)
```

```
        x_real = sample_data()
```

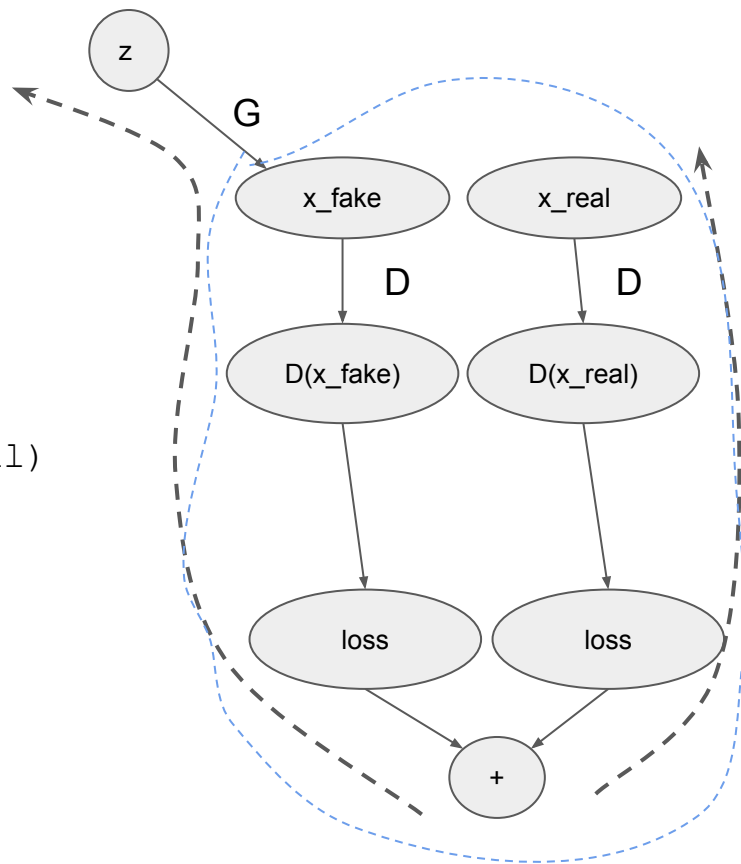
```
        d_fake, d_real = d(x_fake), d(x_real)
```

```
        loss = binary_xent(d_fake, 0) + \
```

```
                binary_xent(d_real, 1)
```

```
        loss.backward()
```

```
        optim_d.step()
```



GENERATIVE ADVERSARIAL NETWORKS (2014)

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

```
min_D binary_xent(d_fake, 0) + binary_xent(d_real, 1)
```

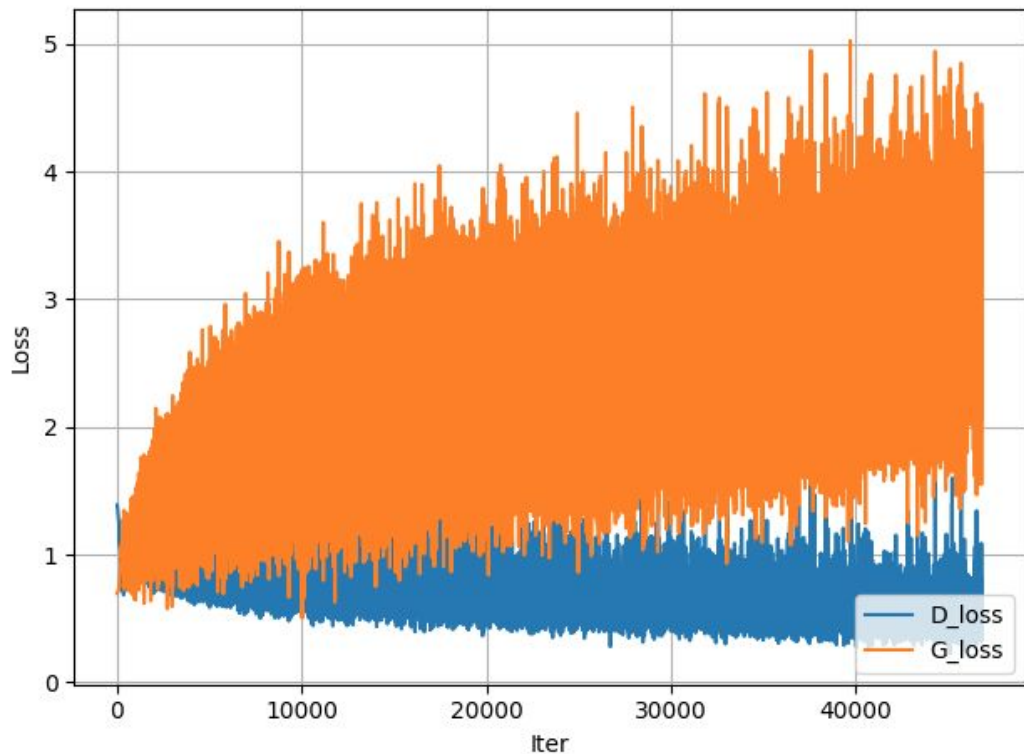
```
min_G binary_xent(d_fake, 1)
```

- Training a GAN (roughly speaking) minimises the Jensen-Shannon divergence between the real and generated distribution
- The loss function determines what divergence this is, e.g.
 - `binary_xent` => JSGAN (Goodfellow et al, 2014)
 - `squared_error` => LSGAN (Mao et al, 2016)
 - See f-divergences paper: <https://arxiv.org/pdf/1606.00709.pdf>

GENERATIVE ADVERSARIAL NETWORKS (2014)

- GANs used to be quite difficult to train (and may still be, but to a much lesser extent now)
- Problems included:
 - Careful balancing of generator / discriminator
 - If discriminator became 'too good' then the GAN would not converge (vanishing gradients)
 - Have to 'cripple' discriminator by reducing its # parameters, or reduce D_steps
 - Mode dropping (generator doesn't completely cover data distribution)
 - Mode collapse (generator outputs the same image)
- How do you evaluate quality of samples?

GENERATIVE ADVERSARIAL NETWORKS (2014)



How to Train a GAN? Tips and tricks to make GANs work

While research in Generative Adversarial Networks (GANs) continues to improve the fundamental stability of these models, we use a bunch of tricks to train them and make them stable day to day. **4: BatchNorm**

Here are a summary of some of the tricks.

[Here's a link to the authors of this document](#)

If you find a trick that is particularly useful in practice, be reasonable and verified, we will merge it in.

1. Normalize the inputs

- normalize
- Tanh as

3: Use a spherical Z

- Don't sample from a Uniform distribution

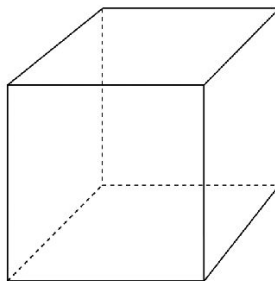
2: A model

In GAN papers

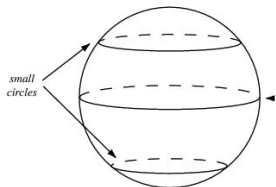
- because
- Goodfellow

In practice, we

- Flip labels



- Sample from a gaussian distribution



- When doing interpolations, do the interpolation
- Tom White's [Sampling Generative Networks](#)

5: Avoid Sparse Gradients: ReLU, MaxPool

- the stability of the GAN game suffers if you have sparse gradients
- LeakyReLU = good (in both G and D)
- For Downsampling, use: Average Pooling, Conv2d + stride
- For Upsampling, use: PixelShuffle, ConvTranspose2d + stride
 - PixelShuffle: <https://arxiv.org/abs/1609.05158>

6: Use Soft and Noisy Labels

- Label Smoothing, i.e. if you have two target labels: Real=1 and Fake=0, then for each replace the label with a random number between 0.7 and 1.2, and if it is a fake sample example).
 - Salimans et. al. 2016
- make the labels the noisy for the discriminator: occasionally flip the labels when training

7: DCGAN / Hybrid Models

- Use DCGAN when you can. It works!
- if you can't use DCGANs and no model is stable, use a hybrid model: KL + GAN or

8: Use stability tricks from RL

- Experience Replay
 - Keep a replay buffer of past generations and occasionally show them
 - Keep checkpoints from the past of G and D and occasionally swap them out for
- All stability tricks that work for deep deterministic policy gradients

Old tricks: <https://github.com/soumith/ganhacks> (Dec 2016)

- optim.Adam rules!
 - See Radford et. al. 2015
- Use SGD for discriminator and ADAM for generator

11: Don't balance loss via statistics (unless you have a good reason to)

- Don't try to find a (number of G / number of D) schedule to uncollapse training
- It's hard and we've all tried it.
- If you do try it, have a principled approach to it, rather than intuition

For example

```
while lossD > A:
    train D
while lossG > B:
    train G
```

12: If you have labels, use them

- if you have labels available, training the discriminator to also classify the samples: auxiliary GANs

13: Add noise to inputs, decay over time

- Add some artificial noise to inputs to D (Arjovsky et. al., Huszar, 2016)
 - <http://www.inference.vc/instance-noise-a-trick-for-stabilising-gan-training/>
 - https://openreview.net/forum?id=Hk4_qw5xe
- adding gaussian noise to every layer of generator (Zhao et. al. EBGAN)
 - Improved GANs: OpenAI code also has it (commented out)

14: [notsure] Train discriminator more (sometimes)

- especially when you have noise
- hard to find a schedule of number of D iterations vs G iterations

15: [notsure] Batch Discrimination

- Mixed results

16: Discrete variables in Conditional GANs

◦ match image channel size

17: Use Dropouts in G in both train and test phase

- Provide noise in the form of dropout (50%).
- Apply on several layers of our generator at both training and test time

WASSERSTEIN GENERATIVE ADVERSARIAL NETS (2017)

- Arjovsky et al proposed the Wasserstein GAN, a very stable GAN variant (and backed by lots of theory)
- Essentially:
 - There exist generated distributions which **do not converge** to the data distribution under non-Wasserstein divergences (JS, KL, etc.)
 - Ones that do converge under non-Wasserstein also converge for Wasserstein
 - The discriminator under a WGAN gives **non-saturating** (clean) gradients **everywhere**
 - The only requirement is that the discriminator is “K-Lipschitz” (more on this soon)

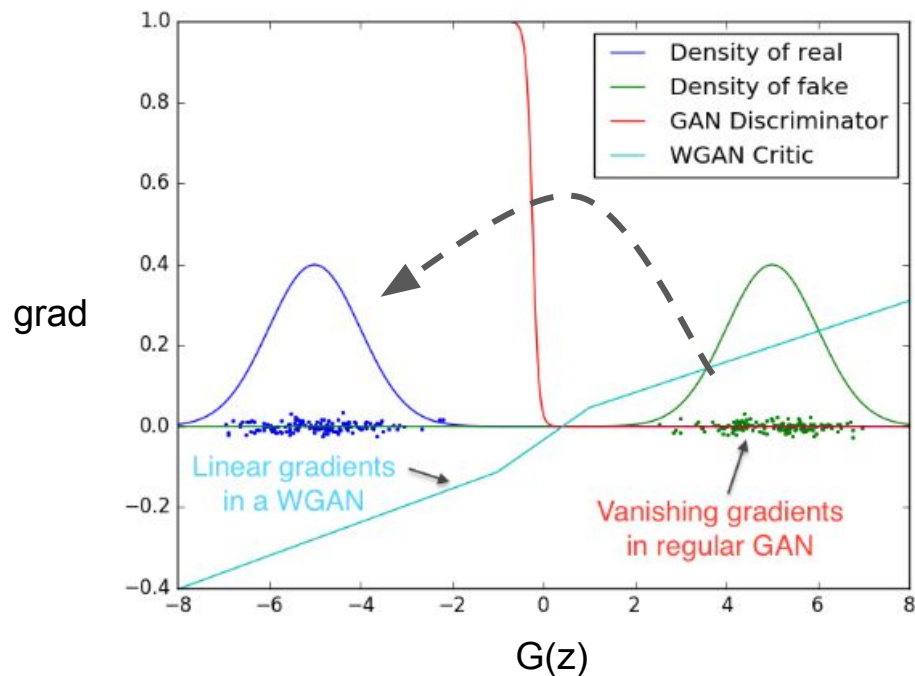
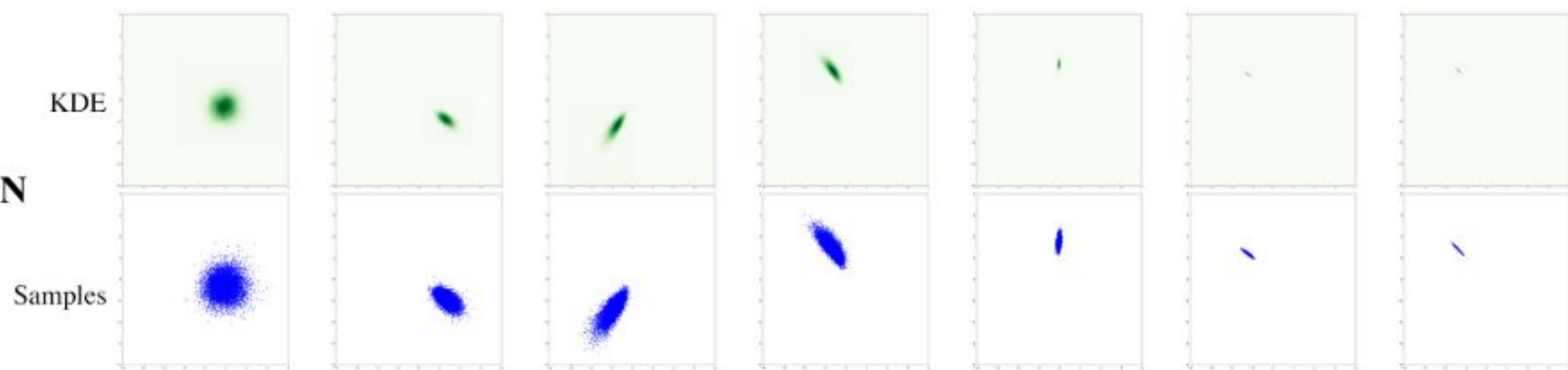
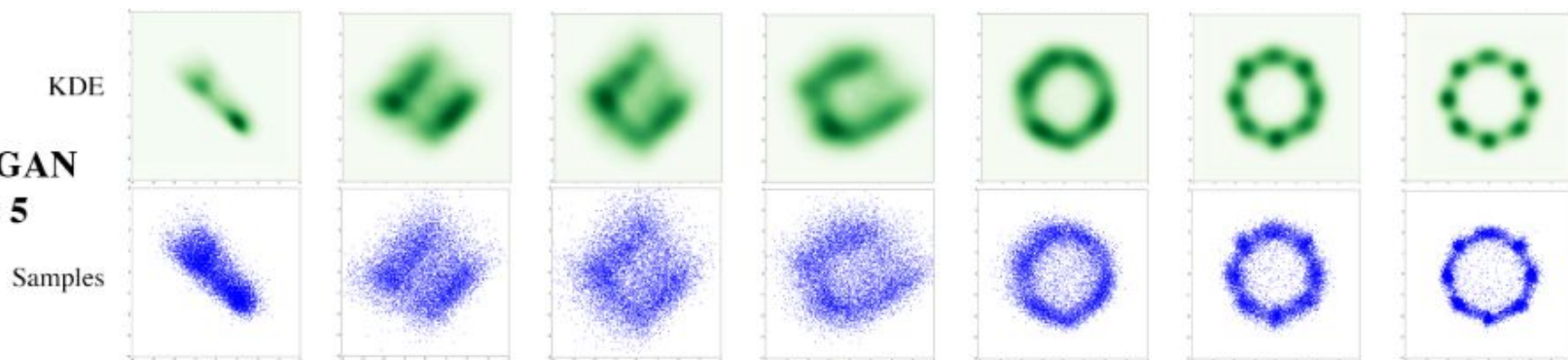


Figure 3: Optimal discriminator and critic when learning to differentiate two Gaussians. As we can see, the traditional GAN discriminator *saturates* and results in vanishing gradients. Our WGAN critic provides very clean gradients on all parts of the space.

Standard GAN



Wasserstein GAN N_critic = 5



epochs

A horizontal arrow pointing to the right, indicating the progression of epochs from left to right across the columns of the figure.

Let d_X and d_Y be distance functions on spaces X and Y . A function $f : X \rightarrow Y$ is K -Lipschitz if for all $x_1, x_2 \in X$,


$$d_Y(f(x_1), f(x_2)) \leq K d_X(x_1, x_2)$$

Intuitively, the slope of a K -Lipschitz function never exceeds K , for a more general definition of slope.

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L \leq 1} \underbrace{\mathbb{E}_{x \sim \mathbb{P}_r}[f(x)] - \mathbb{E}_{x \sim \mathbb{P}_\theta}[f(x)]}$$

Considering all functions whose Lipschitz constants are ≤ 1 (or $\leq K$)

Discriminator tries to maximise this

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L \leq 1} \underbrace{\mathbb{E}_{x \sim \mathbb{P}_r}[f(x)] - \mathbb{E}_{x \sim \mathbb{P}_\theta}[f(x)]}$$


Considering all functions whose Lipschitz constants are ≤ 1 (or $\leq K$)

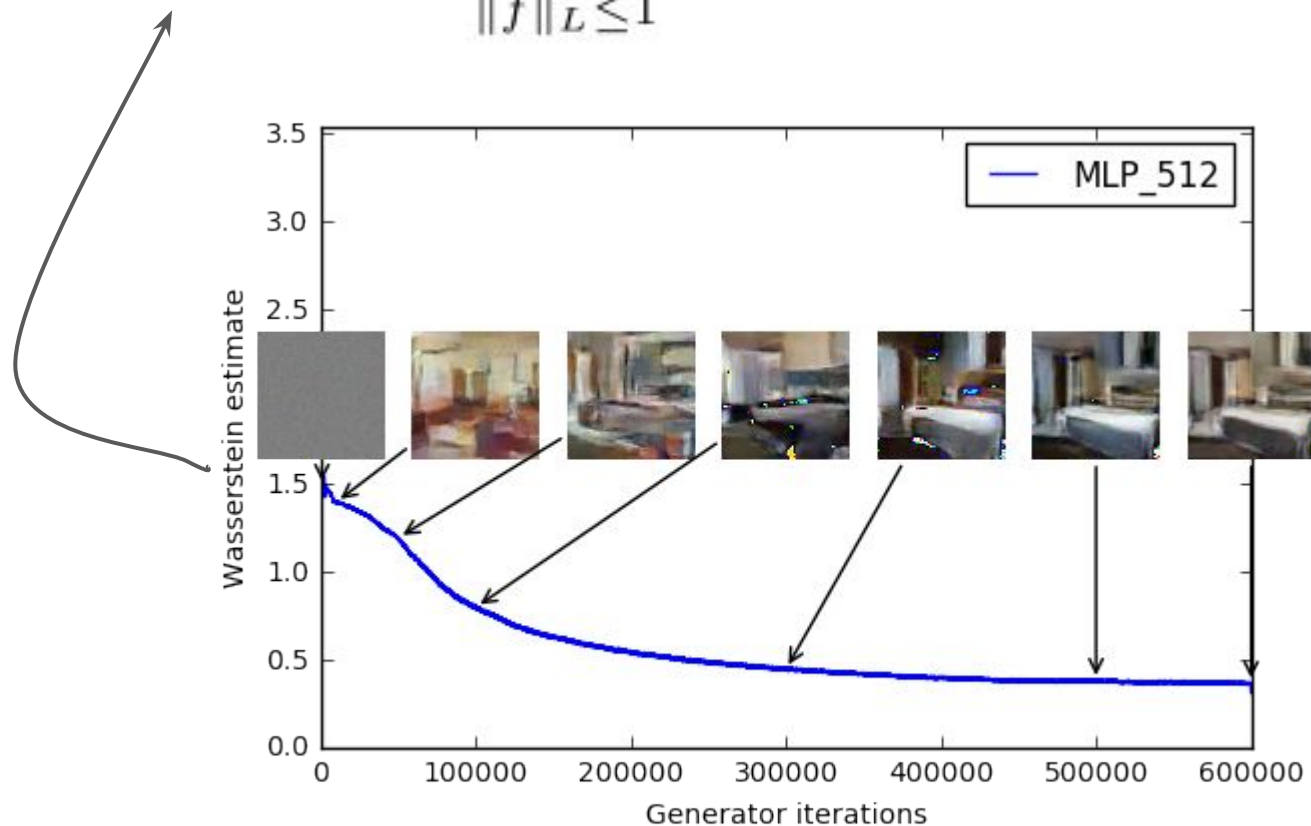
Discriminator tries to maximise this

minimise the following

```
d_loss = -(torch.mean(D(x)) - torch.mean(D(G(z))))
```

```
g_loss = -torch.mean(D(G(z)))
```

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim \mathbb{P}_r}[f(x)] - \mathbb{E}_{x \sim \mathbb{P}_\theta}[f(x)]$$



WASSERSTEIN GENERATIVE ADVERSARIAL NETS (2017)

- How do we make the discriminator K-Lipschitz?
 - WGAN: clip weights between [-c, c] after
 - WGAN-GP (Gulrajani et al, 2017) (<https://arxiv.org/pdf/1704.00028.pdf>)
 - Observation: a function is K-Lipschitz if it has gradients with norm at most =K everywhere
 - Propose a gradient penalty term to the discriminator:

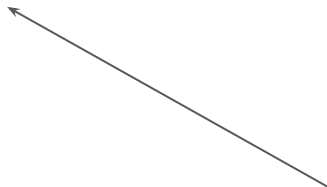
$$\lambda(\|\nabla_x D(x)\|_2 - K)^2$$

Requires backward pass during forward pass

WHAT ABOUT JS-GAN WITH LIPSCHITZ PENALTY?

To conclude, if the clipping is small enough, the network is quite literally a WGAN, and if it's large it will saturate and fail to take into account information between samples that are far away (much like a normal GAN when the discriminator is trained till optimum).

As to the similarities between the difference vs cross-entropy on the loss of the discriminator or critic: if the supports of \mathbb{P}_r and \mathbb{P}_θ are essentially disjoint (which was shown to happen in (Arjovsky & Bottou, 2017) in the usual setting of low dimensional supports), with both cost functions the f will simply be trained to attain the maximum value possible in the real and the minimum possible in the fake, without surpassing the Lipschitz constraint. Therefore, CE and the difference might behave more similarly than we expect in the typical 'learning low dimensional distributions' setting, provided we have a strong Lipschitz constraint.



Appendix G of WGAN paper

SPECTRAL NORM FOR GANS (2018)

- Miyato et al, 2018 (<https://arxiv.org/pdf/1802.05957.pdf>)
- Spectral norm of a matrix A (denoted $\sigma(A)$) is its largest singular value
- The Lipschitz constant of a network f can be bounded as such:

$$\|f\|_{\text{Lip}} \leq \prod_{l=1}^L \sigma(W^l)$$

- Idea: normalise the spectral norm of W by dividing by its spectral norm
 - $W := W / \sigma(W)$
- Benefits: simple, much faster to compute than gradient penalty

SPECTRAL NORM FOR GANS (2018)

```
# simply wrap existing layers with  
# spec_norm() function
```

```
layer = nn.Linear(N, M)  
layer = spec_norm(layer)
```

```
layer = nn.Conv2d(...)  
layer = spec_norm(layer)
```

PRACTICAL TIPS FOR TRAINING

- Store images as uint8s [0,255]
- Each minibatch, scale data to be [-1, 1]
 - $x_scaled = ((x/255.) - 0.5) / 0.5$
 - No need to scale by mean/std, etc.
- Output activation function of generator is tanh, so $G(z)$ is also in [-1,1]
- Use JS-GAN loss / hinge loss
- Add spectral norm to all layers in discriminator
 - Don't use batch norm / instance norm / etc.
- Make $D_iters > G_iters$ (e.g. 5 to 1)
- Use ADAM optimiser

REFERENCES / ACKNOWLEDGEMENTS

- Understanding saturation in JSGANs:
 - <https://danieltakeshi.github.io/2017/03/05/understanding-generative-adversarial-networks/>
- WGAN explained nicely:
 - <https://www.alexirpan.com/2017/02/22/wasserstein-gan.html>
- Spectral norm GAN implementation (close to SOTA):
 - <https://github.com/christiancosgrove/pytorch-spectral-normalization-gan>
 - (Note: PyTorch now has its own spectral norm module built-in)
- GAN collections (easy to read code):
 - <https://github.com/znxlwm/pytorch-generative-model-collections>
- @me on Slack if you have any questions