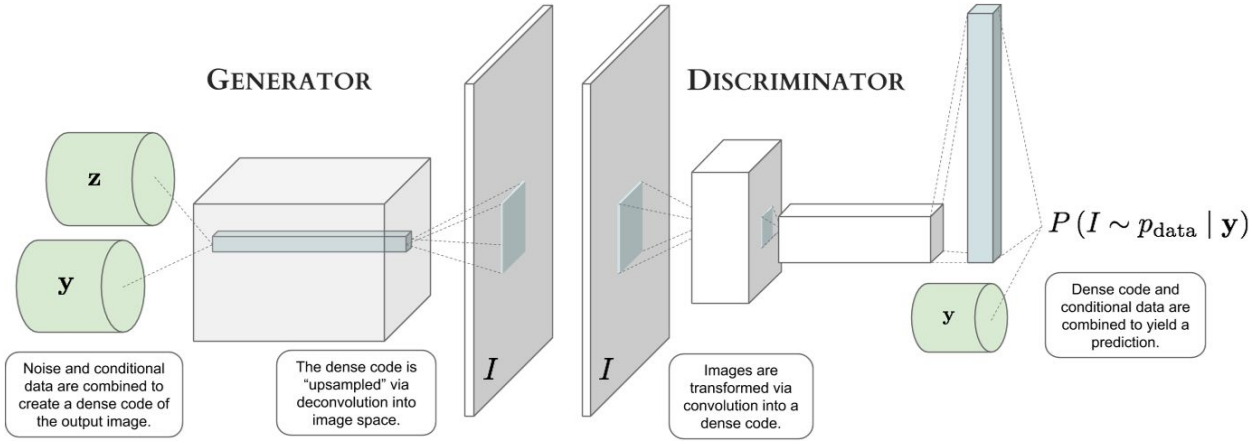
Simple GANS



- **Overview**
  - Generator network vs Discriminator network
  - Generator tries to make fake images to fool the discriminator into thinking they are real
  - Discriminate tries to find real images vs fake images
  - Not really an optimization problem, more like a “game” between generator and discriminator

- **Generator**
  - Receives ‘z’ or random gaussian noise
  - Some networks can also take an original image as input, and then transform that
  - Upsample the input into an image
    - Uses transposed convolutions to upsample (See notes)
  - Generator learns how to best take a random value from the distribution and best turn it into a realistic looking image

- **Discriminator**
  - Standard CNN
    - down samples image, outputs one logit which is the percent that the input image is real (1 for real, 0 for fake, usually)

Generator Loss:

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

- GENERATOR WANTS TO MINIMIZE THIS ONE
- Upside down triangle = gradient
- Generator wants to make images that the discriminator will think are real, so it wants the discriminator to output a '1' for its images.
    - We feed in a random 'z' into the generator, and feed the output of the generator into the discriminator
    - Optimally, the discriminator will output a 1 for this input, so we can subtract 1 from the discriminator's output so we get 0, which is optimal.
    - Then, we want to MINIMIZE this function, so that we get the most zeros, and the discriminator always thinks generator's images are real
- Wait, but why log(x)?
    - Gradient methods work better optimizing log functions because the gradient is usually better scaled.
    - Also, the discriminator may output very small probabilities such as 0.0001 for input images it thinks are fake, and we have limited floating point precision. The log function gets rid of small numbers, since log(0.0001) is 4, and approaches negative infinity as x approahces 0.

Discriminator Loss:

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

- DISCRIMINATOR WANTS TO MAXIMIZE THIS ONE
- First term:
    - Discriminator wants to correctly categorize real data as '1'
- Second term:

- Discriminator wants to categorize generated output as ‘0’. Generated output is defined by putting random noise ‘z’ through the Generator network. Since we want to categorize as ‘0’, we subtract it from 1, which will maximize that number.

PROBLEMS DURING TRAINING

1. Discriminator loss approaches zero
    a. No gradients left for generator to optimize, like vanishing gradient
2. Discriminator loss keeps increasing
    a. Exploding gradient
3. If discriminator classifies everything as real or fake
4. If discriminator becomes too powerful

Tips to improve training: (src https://github.com/soumith/ganhacks)

1. Normalize inputs -1 to -1 and use tanh in last layer of generator
2. Flip labels when training generator
3. Sample from gaussian not uniform
4. Batch norm is OP (and dropout)
5. Avoid sparse gradients such as Relu or max pool
    a. Use leaky relu and average pooling instead
6. Label smoothing
    a. Instead of 0 use random uniform from 0 to 0.3
    b. Instead of 1, random uniform 0.7 to 1.3