Generative Adversarial Networks

“the coolest idea in machine learning in the last twenty years”.

Yann Lecun, Director of AI at Facebook

What's a GAN?!

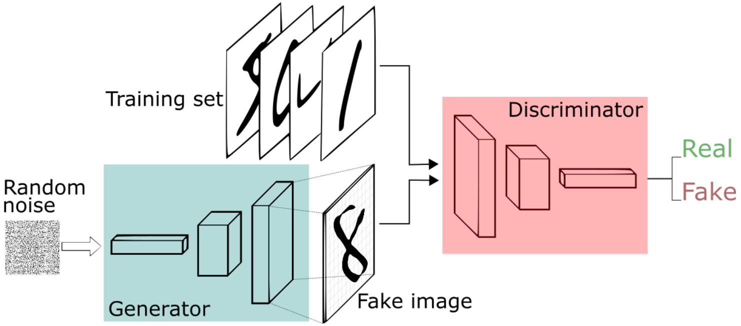

Figure: Generative Adversarial Network

As per the original paper, GANs are a type of machine learning model designed to generate new instances of some class of images.

What's a GAN?!

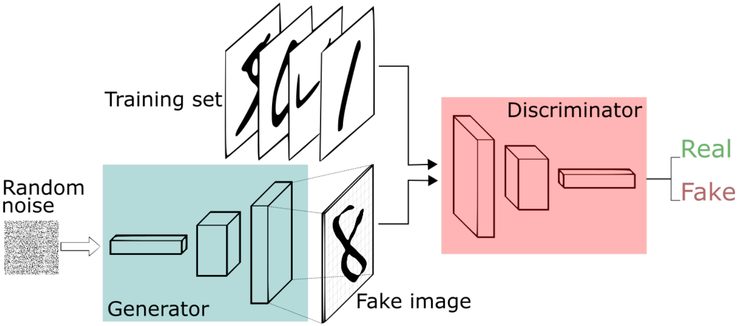

Figure: Generative Adversarial Network

In practice, two neural networks have a mini competition. The generator attempts to approximate the true distribution of the data in question and the discriminator attempts to successfully tell real and fake images apart.

A simple example...

I've written some code in PyTorch that might you follow along, but the essence of what happens is that:

1. The random noise is passed through the generator, G
2. Real images and the G are passed through the discriminator, D
3. The loss (binary cross entropy here) is calculated for both G and D
4. The loss is backpropagated through both networks
5. Stop once the loss has stabilized for both networks

Let's look at this in more detail!

Generator - Forward Pass

In our abstracted example with batch size of 64:

1. Generate a 64 by 100 (convention) array, with z $\sim (-1, 1)$
2. Pass each of the 1*100 vectors through the network (128-728) with Leaky ReLU
3. Output tanh activation of this vector of length 728 and reshape to 28*28

Where Leaky ReLU:

$$f(x) = 1_{x \leq 0}(\alpha x) + 1_{x>0}(x) \tag{1}$$

For some small positive constant α

Discriminator - Forward Pass

1. Flatten 28*28 image to 1*728
2. Pass this through a network (728-128-1 for example)
3. Store both the raw output of the network as well as its sigmoid transformation (so you have two numbers for each image in the batch)

Crucially, we run this on the fake images from the generator as well as real images from our training set. This means that we end up with a set of outputs for both the fake and real images - we need to remember this for the backpropagation steps

Loss functions

General form

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

Discriminator

$$\frac{1}{m}\sum_{i=1}^{m}[logD(x^{(i)}) + log(1-D(G(z^{(i)})))]$$

Generator

$$\frac{1}{m}\sum_{i=1}^{m}[-log(D(G(z^{(i)})))]$$

Backpropagation - Discriminator

For the first layer:

$$\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} [logD(x^{(i)}) + log(1 - D(G(z^{(i)})))]$$

$$\frac{\partial f(x,z)}{\partial x} = -\frac{1}{1 + D(x) + \epsilon} \qquad \text{[Real inputs]}$$

$$\frac{\partial f(x,z)}{\partial z} = -\frac{1}{1 + D(G(z)) + \epsilon} \qquad \text{[Fake inputs]}$$

Backpropagation - Discriminator

- From here, we really just backpropagate like usual
- Only one change - we simulataneously calculate gradients for both the real and fake inputs
- Weight and bias updates using SGD then just consist of the sum of the gradients multiplied by the learning rate

Backpropagation - Generator

Again, for the first layer:

$$\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} [-log(D(G(z^{(i)})))]$$

$$\frac{\partial f(x,z)}{\partial x} = -\frac{1}{D(G(z)) + \epsilon}$$

Backpropagation - Generator

A twist here is that we need some information from the discriminator to calculate the derivative of D(G(z)).

In our example (because GAN variants like WGAN use a different loss function), this simply involves calculating the gradient of the discriminator loss again but not updating its parameters.

And there you have it!

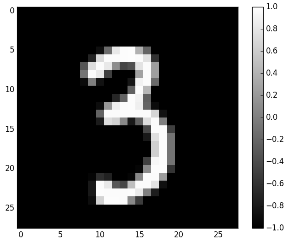

Figure: Human drawn 3

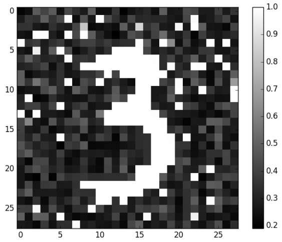

Figure: Computer generated 3

Bear in mind that this is just a simple feedforward network - things get much more interesting when you start to add convolution layers to both the generator and discriminator!