IMAGE EDITING

Last time

- Spoke about Seam Carving and Poisson Image Editing.
- Non-learned methods that allow advanced image editing abilities and output photorealistic images.
- Train neural networks to allow such editing:
  - How do you train to incorporate user input?
  - How do you generate photorealistic images?
- Second question: By using GANs.

GENERATIVE ADVERSARIAL NETWORKS

Review

- We want to produce images (or other objects) that are "photo-realistic".
- But what does it mean for an image to be photo-realistic?
- Let \( X \) represent an HxWx3 array ... not all such arrays will "look" like images.
- What we want is essentially an image prior: \( p(X) \) which takes an \( H \times W \times 3 \) array, and tells us if it’s a photograph.
- We’ve talked about different image priors: functions that give you a ‘score’ given an input \( X \). And we’ve talked about “denoiser” priors.
- Alternate Strategy: Learn a network that learns to “sample” from \( p(X) \).

GANs learn a Generator: \( X = G(z, \theta) \).

- \( G \) is a neural network, and \( \theta \) are its learnable parameters.
- \( z \) is an input "noise" vector that is sampled from a known distribution \( p_z(\cdot) \).
  - For example, \( z \) may be a 100 dimensional vector with each element sampled iid from a uniform distribution between \([-1,1]\).
- The network learns to map these \( z \) from a known simple distribution to images \( X \) which are sampled from \( p_X(X) \).

\[
\{z_i\} \sim p_z \Rightarrow \{G(z_i)\} \sim p_X
\]

Side Note: \( G \) can be thought of as similar to the "inverse CDF" of \( p(x) \).
**GENERATIVE ADVERSARIAL NETWORKS**

Review

How do you learn $G$? Assume you are given a set of samples $\{X_1, X_2, \ldots\}$ of real images.

For training, you will sample different $z \sim p_Z$ and you want to make sure their outputs have the same "distribution" as the training set. But there is no one-to-one correspondence between your sampled $z$’s and training examples.

Actually, variational auto-encoders try to learn a strategy of mapping each image to a $z$, and ensuring that the $z$’s corresponding to your training set follow $p_z$.

In GANs, we will learn a second network, called the discriminator, which will determine if the outputs of $G$ match the distribution of your training set.

The generator will get its gradients through the discriminator—by using a loss based on the output of its discriminator.

**GENERATIVE ADVERSARIAL NETWORKS**

Review

$$L(\theta, \phi) = -E_{z \sim p_Z} \log(1 - D(G(z; \theta); \phi)) - E_{X \sim p_X} \log D(X; \phi)$$

Quick Theoretical Analysis: Let’s say $D$ and $G$ both had infinite capacity.

- Given a specific generator $G$, let $p_G$ represent the distribution of $G(z; \theta)$, $z \sim p_Z$.
- Let’s assume $D$ knows this distribution and the true distribution $p_X$.
- Given an input $X$, what should the output of $D$ be? What is the optimal output that minimizes the loss? $D(X) = \frac{p_X(X)}{p_X(X) + p_G(X)}$.
- Now if $G$ is training against an optimal discriminator, what should be its optimal distribution $p_G(X)$?
- It should try to maximize $-E_{x \sim p_G} \log 1 - D(X)$, which is $\log(p_G(X) + p_G(X)) - \log(p_G(X))$.
- It has to be a proper distribution: $\int p_G(X) dX = 1$.
- The optimal answer is when $p_G(X) = p_X(X)$, when the discriminator will be forced to output 0.5 (i.e., chance) for all inputs.

**GENERATIVE ADVERSARIAL NETWORKS**

Review

- Training Set: $\{X_1, X_2, X_3, \ldots\} \in \mathbb{R}^{HxW}$ that come from some $p_X$.
- Generator network: $G(z; \theta) \rightarrow \mathbb{R}^{HxW}$, where $z \sim p_Z$.
- Discriminator Network: $D(X; \phi) \rightarrow \mathbb{R}$ scores its input to say if it came from $p_X$ (the training set) or from $G$.

$$L(\theta, \phi) = -E_{z \sim p_Z} \log(1 - D(G(z; \theta); \phi)) - E_{X \sim p_X} \log D(X; \phi)$$

- This is basically a binary classification cross-entropy on the discriminator network: we want it to output 1 if the image comes from $p_X$ and 0 if it comes from $G$.
- We will train the parameters of $D$ to minimize this loss.
- And the parameters of $G$ to maximize this loss.

In practice, sample a bunch of $\{z_1, z_2, \ldots\} \sim p_Z$.

$$L(\theta, \phi) = -\frac{1}{N} \sum \log(1 - D(G(z_i; \theta); \phi)) - \frac{1}{N} \sum \log D(X; \phi)$$

$$\theta = \arg \max \min \ L(\theta, \phi)$$

**GENERATIVE ADVERSARIAL NETWORKS**

Review

What about learning conditional distributions?

- I want to model $p(Y|X)$. That is, learn the distribution of possible outputs $Y$, given a specific input $X$.
- Conditional GANs

Generator takes both noise vector and $x$ as input: $y = G(x, z; \theta)$.

Discriminator takes generated image, or real $y$, AND the input $x$ as input:

$$L(\theta, \phi) = -\frac{1}{N} \sum \log(1 - D(G(x_i, z_i; \theta), x_i; \phi)) - \frac{1}{N} \sum \log D(y_i, x_i; \phi)$$

- So now, the discriminator is evaluating whether the generated vs true $y_i$ goes with the true $x_i$. 

- The Problem: Training generators that output high-resolution images is unstable.
- So think of the decomposition of an image into a Laplacian pyramid:

\[ X \leftrightarrow [L_0, L_1, L_2, L_3, G_4] \]

\[ G_0 = X, G_1 = G_{i-1} \downarrow_2. \text{ Where down-sampling is by convolution with a Gaussian filter with stride 2.} \]

\[ L_i = G_i - G_{i+1} \uparrow_2. \text{ Where up-sampling is by bilinear up-sampling.} \]

- Recursive decomposition and re-construction.
  - Each \( L_i \) has only the 'detail' image at that level: lower magnitudes that corresponding \( G_i \)
  - \( G_i \) is lower resolution (fewer numbers) than \( G_{i+1} \).

\[ p(X) = p(L_0, L_1, L_2, L_3, G_4) = p(G_4)p(L_4|G_4)p(L_3|L_4, G_3)p(L_2|L_3, G_2)p(L_1|L_2, L_3, G_3)p(L_0|L_3, L_2, L_1, G_4) \]

\[ p(X) = p(L_0, L_1, L_2, L_3, G_4) = p(G_4)p(L_3|G_4)p(L_2|L_3, G_3)p(L_1|L_2, G_2)p(L_0|L_1, L_2, L_3, G_4) \]

- Learn a low-resolution generator for \( p(G_4) \), and conditional generators for each level \( p(L_i|G_{i+1}) \).

In practice, modern GANs can generate high-resolution images without this kind of conditioning.

While the original GAN paper used "fully connected" layers in the generator and discriminator, it turns out it's better to use convolution and "transpose" convolution layers.


DC-GAN does well, but starts to run into trouble once you get to very high resolutions (typically above 64x64 or 128x128).

Another paper uses the low-resolution decomposition idea, but as a means of training and not an explicit factorization into conditional decompositions.

The basic idea is, you learn smaller generators and discriminators to generate a low-resolution version of your image.

Then use these layers as initialization for a slightly larger generator and discriminator pair for images at the next resolution.
Given their ability to generate realistic outputs, GANs have been used for a number of image generator tasks. In painting, fill in a large missing region inside an image. Here, the loss includes a discriminator that looks at the context+inpainted block (i.e., it's a conditional generator).

The conditional generator idea can now be used to generate images from different kinds of inputs: where the user input, that the generator and discriminators are conditioned on, is an image shaped object.


But ideally, we want more control on allowing user input to modify images.

The sketch to image translation example is close to what we want, but it doesn’t constrain the output image enough. There are many plausible images that could correspond to that sketch.


Basic idea: train a GAN to reconstruct an image from a contour / edge map. Image = G(Contour Map).

Edge map is zero everywhere except at edge pixels. Edge pixels are determined by a standard edge-detector.

At edges, don’t just store ones, but information about the edge neighborhood.

So the edge map is a multi-channel image which is zero at all non-edge pixels, and at edge-pixels has some feature vector.

Features

- 6-values corresponding to RGB intensities on either side of the contour. Either side here is defined by edge direction.
- Gradients: x- and y- gradients for each of the RGB channels from the original image at the edge pixels.
- Learned Features: The paper also considers low-dimensional features (3 per edge pixel) that are learned along with the generator.

But the gradient features turn out to be the best for image editing.

So this is the basic setup: Given an image, run an edge detector and extract features at edge pixels.

Learn to reconstruct the image from this sparse edge-feature map.

Turns out this works well, which suggests that locations of edges + local gradient information is enough information to reconstruct the entire image. (Kind of like the poisson solver).
You will notice though that some detail in regions without contours are missing in the reconstruction. But because this is a GAN, those regions are photorealistic and not completely smooth (as would happen if you tried to do a poisson reconstruction).

But the quality of the reconstruction will depend on how many edge pixels you retain.

What’s the point?

The edge map is much easier to edit than the image itself. You can move edges around in the same image, or copy edges from multiple images (like in Poisson Image Editing).

Green shows moved/edited contours.

You can reshape contours (and move the associated feature values with them).
IMAGE EDITING WITH GANS


VISION + LANGUAGE: IMAGE GENERATION

- **Already Seen**: Given an image, generate a caption.
- **New Task**: Given a caption, generate an image!
- **Basic idea**: use an RNN to encode the sentence into a feature vector.

Learn a conditional GAN that takes this encoding as input.

VISION + LANGUAGE: IMAGE GENERATION

VISION + LANGUAGE: IMAGE GENERATION


<table>
<thead>
<tr>
<th>Caption</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>this vibrant red bird has a pointed black beak</td>
<td>![Image of red birds with black beak]</td>
</tr>
<tr>
<td>this bird is yellowish orange with black wings</td>
<td>![Image of birds with orange color]</td>
</tr>
<tr>
<td>the bright blue bird has a white colored belly</td>
<td>![Image of blue birds with white belly]</td>
</tr>
</tbody>
</table>

VISION + LANGUAGE: IMAGE GENERATION


<table>
<thead>
<tr>
<th>Caption</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>this flower has white petals and a yellow stamen</td>
<td>![Image of flower with white petals]</td>
</tr>
<tr>
<td>the center is yellow surrounded by wavy dark purple petals</td>
<td>![Image of flower with purple petals]</td>
</tr>
<tr>
<td>this flower has lots of small round pink petals</td>
<td>![Image of flower with pink petals]</td>
</tr>
</tbody>
</table>

MOTION MAGNIFICATION

- The task: "magnify" motions.
- Don't make them faster, rather exaggerate small movements.