GENERAL

- PSET 3 grades posted, PSET 4 to be posted shortly.
- PSET 5 due today.
- No office hours on Friday.

- Project reports due December 9.
- Submit through git
  
  ```git clone cse559@euclid.seas.wustl.edu:wustl.key/project```

- Include only your report PDF. Do not submit your code or other files.
  - But hang on to your code in case we ask for it later.

GENERATIVE ADVERSARIAL NETWORKS

- So far, we have looked at networks that given an input $x$, produce an output $y$.
- All $(x, y)$ pairs come from some joint distribution $p(x, y)$.
- A function that maps $x \rightarrow y$ is then reasoning with the distribution $p(y|x)$
- And producing a single guess $\hat{y}$ which minimizes $\mathbb{E}_{p(y|x)} L(y, \hat{y})$.
- But if $p(y|x)$ is not deterministic, this expected loss won't go to zero: Bayes Error
- What if I didn't want my network to produce a "best" guess, but tell me about this distribution ?

- One option: choose a parametric form for $p(y|x) = f(y; \theta)$.
- Have a network $g$ that predicts $\theta = g(x)$ for a specific $x$.
- The other option is to train a sampler. A network that given an input $x$, produces "samples" from $p(y|x)$.
- How do you produce samples, or how do you produce multiple outputs for the same input ?
- You give your network access to a random generator: a noise source.
Let’s ignore conditional distributions. Consider the task of generating samples from $p_x(x)$.

You don’t know $p(x)$, but you have training examples that are samples from $p_x(x)$. You want to learn a “generator” network $G(z; \theta)$ which

- Takes in random inputs $z$ from a known distribution $p_z(z)$
- And produces outputs $x$ from $p(x)$
- Has learnable parameters $\theta$

You want to select $\theta$ such that the distribution of $G(z; \theta)$ to match $p_x(x)$. But you don’t have the data distribution, only samples from it.

Set this up as a min-max objective with a second network, a “discriminator” $D(x, \phi)$

The discriminator is a binary classifier. Tries to determine if

- The input $x$ is “real”, i.e., it came from the training set.
- Or “fake”, i.e., it was the output of $G$

Train both networks simultaneously, against the following loss:

$$L(\theta, \phi) = -E_{z \sim p_z} \log(1 - D(G(z; \theta); \phi)) - E_{x \sim p_x} \log D(x; \phi)$$

This is the cross-entropy loss on the discriminator saying outputs of $G$ should be labeled 0.

What about examples that should be labeled 1?

Theoretical Analysis

- Let’s say your discriminator and generator had infinite capacity and you had infinite training data.
- For a given input $x$, what should the optimal output of your discriminator $D(x)$ be?

  - Say you know $p_x(x)$.
  - You also know $p_z(x)$ and $G$, and therefore $p_g(x)$: probability of $x$ being an output from the generator.

$$q = D(x) = \arg \max_q \log(1 - q) - p_x(x) \log q$$

- What $q$ minimizes this, for $q \in [0, 1]$?

$$q = \frac{p_x(x)}{p_x(x) + p_g(x)}$$

- Let’s replace $D$ with this optimal value in the loss function, and figure out what $G$ should do.
Generative Adversarial Networks

\[ G = \arg \max_G \min_D -E_{z \sim p_Z} \log(1 - D(G(z))) - E_{x \sim p_X} \log D(x) \]

\[ G = \arg \min_D \mathbb{E}_{z \sim p_Z} \log \frac{p_{\theta}(G(z))}{p_{\theta}(G(z)) + p_{\phi}(G(z))} + E_{x \sim p_X} \log \frac{p_{\phi}(x)}{p_{\phi}(x) + p_{\theta}(x)} \]

- Remember that \( p_{\theta} \) also depends on \( G \). In fact, you can replace this as an optimization on \( p_{\theta} \).

\[ p_{\theta} = \arg \min_{\theta} \int \left[ p_{\theta}(x) \log \frac{p_{\theta}(x)}{p_{\theta}(x) + p_{\phi}(x)} + p_{\phi}(x) \log \frac{p_{\theta}(x)}{p_{\theta}(x) + p_{\phi}(x)} \right] dx \]

- You can relate this to KL-divergences:

\[ KL(p_{\theta} \parallel \frac{p_{\theta} + p_{\phi}}{2}) + KL(p_{\phi} \parallel \frac{p_{\theta} + p_{\phi}}{2}) \]

- Called the "Jensen-Shannon" Divergence
- Minimized when \( p_{\theta} \) matches \( p_{\phi} \).

Generative Adversarial Networks

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

Practical Concerns

- So the procedure is, set up your generator and discriminator networks. Define the loss.
- At each iteration, pick a batch of \( z \) values from a known distribution (typically a vector of uniformly or Gaussian distributed values)
- And a batch of training samples
- Compute gradients for the discriminator and update.
- Compute gradients for the generator, by back-propagating through the discriminator, and update.
- A common issue is that the discriminator has a much "easier" task
  - In the initial iterations, your generator will be producing junk.
  - Very easy for the discriminator to identify fake samples with high confidence.
  - At that point, \( \log 1 - D(G(z)) \) will saturate.
  - No gradients to generator.

Generative Adversarial Networks

Conditional GANs: now want to sample from \( p(x | s) \) for a given \( s \)

\[ G(s) = \arg \max \mathbb{E}_{z \sim p_Z} \log(1 - D(G(s; z); \phi)) \]

- Same adversarial setting but \( s \) is given as an input to both generator and discriminator
- \( G(s, z) \) and \( D(s) \)
- Sometimes a noise source is simply replaced by dropout
- Or no noise at all: called an "Adversarial Loss". The generator is producing a deterministic output, but being trained with a distribution matching loss rather than \( L_1 \) or \( L_2 \).
  - Can be useful when the true \( p(x | s) \) is multi-modal.
  - Regular networks would average the modes, adversarial loss promotes picking one of the modes.

Generative Adversarial Networks

Practical Concerns

- One common approach: minimize \( G \) with respect to a different loss
  - Instead of \( \max - \log(1 - D(G(z))) \)
  - Do \( \min - \log D(G(z)) \)
- View as minimizing cross-entropy wrt wrong label, rather than maximizing wrt true label.
- Other approaches:
  - Reduce capacity of discriminator
  - Make fewer updates to discriminator, or have lower learning rate
  - Provide additional losses to generator to help it train: e.g., separate network that predicts intermediate features of the discriminator.
  - Other losses: See Wasserstein GANs.
- Also need to be careful how you use Batch Normalization. (Don’t let the discriminator use batch statistics to tell real and fake apart!)