CS6501: Deep Learning for Visual Recognition
Generative Adversarial Networks (GANs)
Today’s Class

- Adversarial Examples – Input Optimization
- Generative Adversarial Networks (GANs)
- Conditional GANs
- Style-Transfer Networks
What we have been doing: Optimize weights in the network to predict bus (correct class).

\[ \mathbf{I} \quad y = f(I; w) \quad L(y, \text{bus}) \]

\[ w = w - \lambda \frac{\partial L}{\partial w} \]
New Idea: Create Adversarial Inputs by optimizing the input image to predict ostrich (wrong class).

\[ I \quad y = f(I; w) \quad L(y, ostrich) \]

\[ I = I - \lambda \frac{\partial L}{\partial I} \]

Work on Adversarial examples by Goodfellow et al., Szegedy et al., etc.
Convnets (optimize input to predict ostrich)

Work on Adversarial examples by Goodfellow et al., Szegedy et. al., etc.
All get predicted as ostrich
Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images

Anh Nguyen, Jason Yosinski, Jeff Clune, 2014

Figure 13. Images found by maximizing the softmax output for classes via gradient ascent [11, 26]. Optimization begins at the ImageNet mean (plus small Gaussian noise to break symmetry) and continues until the DNN confidence for the target class reaches 99.99%. Images are shown with the mean subtracted. Adding regularization makes images more recognizable but results in slightly lower confidence scores (see supplementary material).
New Idea: Create Adversarial Inputs by optimizing the input image to predict ostrich (wrong class).

\[ I = I - \lambda \frac{\partial L}{\partial I} \]

Work on Adversarial examples by Goodfellow et al., Szegedy et al., etc.
Total Variation Regularization

A second richer regulariser is total variation (TV) \( R_{V_\beta}(x) \), encouraging images to consist of piece-wise constant patches. For continuous functions (or distributions) \( f : \mathbb{R}^{H \times W} \supset \Omega \rightarrow \mathbb{R} \), the TV norm is given by:

\[
R_{V_\beta}(f) = \int_\Omega \left( \left( \frac{\partial f}{\partial u}(u, v) \right)^2 + \left( \frac{\partial f}{\partial v}(u, v) \right)^2 \right)^{\frac{\beta}{2}} du dv
\]

where \( \beta = 1 \). Here images are discrete (\( x \in \mathbb{R}^{H \times W} \)) and the TV norm is replaced by the finite-difference approximation:

\[
R_{V_\beta}(x) = \sum_{i,j} \left( (x_{i,j+1} - x_{ij})^2 + (x_{i+1,j} - x_{ij})^2 \right)^{\frac{\beta}{2}}.
\]

Figure 1. What is encoded by a CNN? The figure shows five possible reconstructions of the reference image obtained from the 1,000-dimensional code extracted at the penultimate layer of a reference CNN[13] (before the softmax is applied) trained on the ImageNet data. From the viewpoint of the model, all these images are practically equivalent. This image is best viewed in color/screen.
Taking the idea to the extreme: Google’s DeepDream

https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html

Generate your own in Pytorch: https://github.com/XavierLinNow/deepdream_pytorch
Generative Adversarial Networks (GAN) [Goodfellow et al 2014]

https://deeplearning4j.org/generative-adversarial-network
Generative Network (closer look)

Generative Network (closer look)

- Deconvolutional Layers
- Upconvolutional Layers
- Backwards Strided Convolutional Layers
- Fractionally Strided Convolutional Layers
- Transposed Convolutional Layers
- Spatial Full Convolutional Layers

Generative Adversarial Networks (GAN) [Goodfellow et al.]
Generative Adversarial Networks (GAN) [Goodfellow et al.]

https://deeplearning4j.org/generative-adversarial-network
**Algorithm 1** Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, $k$, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations do
  for $k$ steps do
    • Sample minibatch of $m$ noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
    • Sample minibatch of $m$ examples $\{x^{(1)}, \ldots, x^{(m)}\}$ from data generating distribution $p_{data}(x)$.
    • Update the discriminator by ascending its stochastic gradient:
      \[
      \nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D(x^{(i)}) + \log \left(1 - D\left(G\left(z^{(i)}\right)\right)\right) \right].
      \]
  end for
  • Sample minibatch of $m$ noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
  • Update the generator by descending its stochastic gradient:
    \[
    \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log \left(1 - D\left(G\left(z^{(i)}\right)\right)\right).
    \]
end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.
Algorithm 1: Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, \( k \), is a hyperparameter. We used \( k = 1 \), the least expensive option, in our experiments.

for number of training iterations do
  for \( k \) steps do
    • Sample minibatch of \( m \) noise samples \( \{z^{(1)}, \ldots, z^{(m)}\} \) from noise prior \( p_g(z) \).
    • Sample minibatch of \( m \) examples \( \{x^{(1)}, \ldots, x^{(m)}\} \) from data generating distribution \( p_{data}(x) \).
    • Update the discriminator by ascending its stochastic gradient:
      \[
      \nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D(x^{(i)}) + \log \left( 1 - D(G(z^{(i)})) \right) \right].
      \]
  end for
  • Sample minibatch of \( m \) noise samples \( \{z^{(1)}, \ldots, z^{(m)}\} \) from noise prior \( p_g(z) \).
  • Update the generator by descending its stochastic gradient:
    \[
    \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log \left( 1 - D(G(z^{(i)})) \right).\]
end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.
Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, $k$, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations do
  for $k$ steps do
    • Sample minibatch of $m$ noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
    • Sample minibatch of $m$ examples $\{x^{(1)}, \ldots, x^{(m)}\}$ from data generating distribution $p_{data}(x)$.
    • Update the discriminator by ascending its stochastic gradient:
      \[
      \nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D\left(x^{(i)}\right) + \log \left(1 - D\left(G\left(z^{(i)}\right)\right)\right) \right].
      \]
  end for
  • Sample minibatch of $m$ noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
  • Update the generator by descending its stochastic gradient:
    \[
    \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log \left(1 - D\left(G\left(z^{(i)}\right)\right)\right).
    \]
end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.
**Algorithm 1** Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, $k$, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

**for number of training iterations do**

**for** $k$ steps **do**

- Sample minibatch of $m$ noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
- Sample minibatch of $m$ examples $\{x^{(1)}, \ldots, x^{(m)}\}$ from data generating distribution $p_{data}(x)$.
- Update the discriminator by ascending its stochastic gradient:

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

**end for**

- Sample minibatch of $m$ noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
- Update the generator by descending its stochastic gradient:

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

**end for**

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.
Generative Adversarial Networks (GAN) [Goodfellow et al.]

https://deeplearning4j.org/generative-adversarial-network
Generative Adversarial Networks (GAN) [Goodfellow et al.]

- GANs are hard to train, loss for the discriminator and generator might fluctuate.
- There are many choices for loss, and other auxiliary signals.
- Training of these models is even less well understood than for other deep models.

https://deeplearning4j.org/generative-adversarial-network
Basic GAN Results (Example implementation is provided in Pytorch’s examples)

http://torch.ch/blog/2015/11/13/gan.html
NVidia’s progressive GANs ICLR 2018
Google’s BigGAN
Google’s BigGAN

Teddy Bear

Microphone

Conditional GANs: Input is not just Noise

Isola et al. CVPR 2017: Image-to-Image Translation with Conditional Adversarial Networks
Conditional GANs: Also Hard to Train

Result they obtained with a regular Fully Convolutional Network

Result they obtained with a U-Net network (with skip-connections)

Isola et al. CVPR 2017: Image-to-Image Translation with Conditional Adversarial Networks
Conditional GANs: Also Hard to Train

Ronneberger et al. MICCAI 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation
Questions?