CS4501: Introduction to Computer Vision
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).

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

\[ w = w - \lambda \frac{\partial L}{\partial w} \]
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
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.]

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.]
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.

\begin{algorithm}
\begin{algorithmic}
\For {number of training iterations}
  \For {$k$ steps}
    \begin{itemize}
    \item Sample minibatch of $m$ noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
    \item Sample minibatch of $m$ examples $\{x^{(1)}, \ldots, x^{(m)}\}$ from data generating distribution $p_{data}(x)$.
    \item Update the discriminator by ascending its stochastic gradient:
      \begin{equation}
      \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{equation}
    \end{itemize}
  \EndFor
  \item Sample minibatch of $m$ noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
  \item Update the generator by descending its stochastic gradient:
    \begin{equation}
    \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log \left( 1 - D \left( G \left( z^{(i)} \right) \right) \right).
    \end{equation}
  \EndFor
\end{algorithmic}
\end{algorithm}

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

\[ \textbf{for number of training iterations do} \]

\[ \textbf{for } k \text{ steps do} \]

- 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 \( p_{\text{data}}(x) \).
- Update the discriminator by ascending its stochastic gradient:

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

\[ \textbf{end for} \]

- 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:

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

\[ \textbf{end for} \]

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
More on the Idea of Feature Space Optimization

Gatys et. al. Image Style Transfer Using Convolutional Neural Networks. CVPR 2016
Idea 1: Image Reconstruction from Features

\[ \mathcal{L}_{\text{total}} = \alpha \mathcal{L}_{\text{content}} \]

\[ \mathcal{L}_{\text{content}} = \sum (F^l - P^l)^2 \]

\[ \mathcal{L}_{\text{total}} \]

\[ \frac{\partial \mathcal{L}_{\text{total}}}{\partial \bar{x}} \]

Gradient descent

\[ \bar{x} = \bar{x} - \lambda \frac{\partial \mathcal{L}_{\text{total}}}{\partial \bar{x}} \]

\[ \bar{p} = \]
Idea 1: Image Reconstruction from Features

\[
\mathcal{L}_{content} = \sum (F^l - P^l)^2
\]
Idea 1: Image Reconstruction from Features

\[ L_{total} = \alpha L_{content} \]

\[ \frac{\partial E_I}{\partial F^{L-1}} \]

\[ L_{content} = \sum (F^l - P^l)^2 \]

\[ \text{conv5}_{1,2} \]

\[ \text{pool4} \]

\[ \text{conv4}_{3,4} \]

\[ \text{pool3} \]

\[ \text{conv3}_{3,4} \]

\[ \text{pool2} \]

\[ \text{conv2}_{3,4} \]

\[ \text{pool1} \]

\[ \text{conv1}_{3,4} \]

Input

\[ \vec{p} = \]

\[ \vec{x} := \vec{x} - \lambda \frac{\partial L_{total}}{\partial \vec{x}} \]
\[ \mathcal{L}_{\text{content}} = \sum (F^l - P^l)^2 \]
Idea 2: Backpropagation of Style
Idea 2: Backpropagation of Style

$$L_{total} = \beta L_{style}$$

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l$$

$$E_L = \sum (G^L - A^L)^2$$
Idea 2: Backpropagation of Style

\[ G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l \cdot \]

\[ E_L = \sum (G^L - A^L)^2 \]
\[ E_L = \sum (G^L - A^L)^2 \]

\[ \mathcal{L}_{total} = \alpha \mathcal{L}_{content} + \beta \mathcal{L}_{style} \]

\[ \mathcal{L}_{style} = \sum_l w_l E_l \]

\[ \tilde{a} = \]

\[ \tilde{x} = \]

\[ \tilde{p} = \]
Questions?