Lecture 22: Adversarial Image, Adversarial Training, Variational Autoencoders, and Generative Adversarial Networks

ECE 417: Multimedia Signal Processing

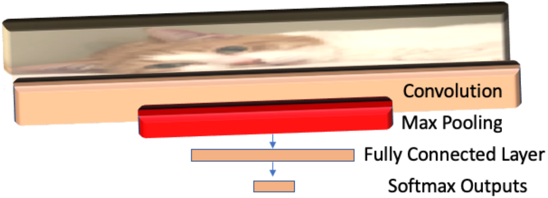
Mark Hasegawa-Johnson

University of Illinois

Nov. 8, 2018

Outline

Review: ConvNet



Today's notation (like the MP, different from last week's lectures):

- $\hat{y}_\ell \in (0,1)$ is the network output for label ℓ, $1 \le \ell \le L$
  - Softmax output layer ensures that $\hat{y}_\ell \ge 0$ and $\sum_\ell \hat{y}_\ell = 1$.
- $y_\ell \in \{0,1\}$ is the reference bit for label ℓ, $1 \le \ell \le L$
  - One-hot encoding ensures that $y_\ell \ge 0$ and $\sum_\ell y_\ell = 1$.

Review: ConvNet Training

For some convolutional weight u_{jkmn} (weight connecting j^{th} input channel to k^{th} output channel, pixel (m, n)),

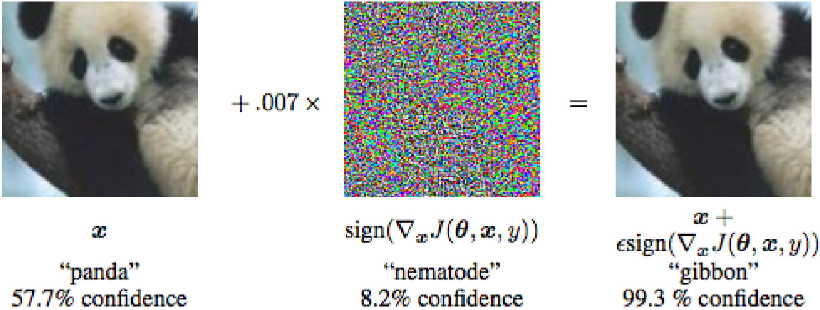
$$u_{kmn} \leftarrow u_{jkmn} - \eta \frac{\partial E}{\partial u_{jkmn}}$$

where

$$E = -\ln \hat{y}_{\mathrm{true}},$$

$$\mathrm{true} \in \{1, \ldots, L\} \text{ is the true label}$$

"Breaking" a ConvNet: Adversarial Examples



Credit: http://karpathy.github.io/2015/03/30/breaking-convnets/

Interesting Current Research Topics

- Suppose the CIA is recording your phone calls, and processing each one using a speaker-ID system. Can you make it believe that you are somebody else?
- Can you do that without knowing exactly what the CIA's neural net parameters are?
- Can you figure out whether news is fake versus real? Can the fake-news providers fool you?
- You have a speech recognizer trained for English. Can you "fool" it into believing that Swahili is English, in such a way that it generates correct Swahili transcriptions?
- You have a system that breaks in mysterious ways. Can you use adversarial examples to figure out why it's breaking?

"Breaking" a ConvNet

Modify the image $x_j[m,n]$ (j^{th} channel, pixel (m,n)) as

$$x_j[m,n] \leftarrow x_j[m,n] + \eta \frac{\partial E}{\partial x_j[m,n]}$$

where

$$E = -\ln \hat{y}_{\text{true}}$$

The result: with very small η, we can make the network believe the image is something else.

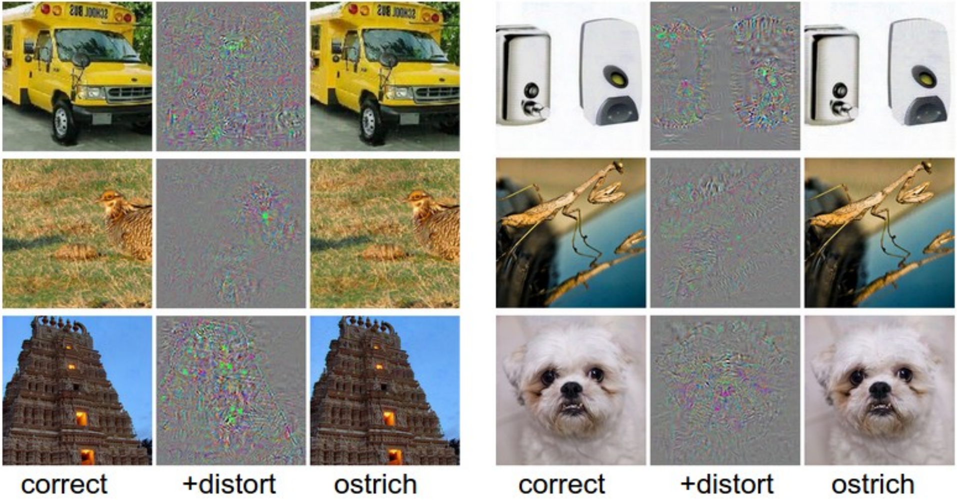
Intentionally Generating the Mistakes We Want

Suppose, instead, we do this:

$$x_k[m,n] \leftarrow x_k[m,n] + \eta \frac{\partial}{\partial x_k[m,n]} \left(\ln \hat{y}_{\text{fake}} - \ln \hat{y}_{\text{true}}\right)$$

Then we can force the network to believe that the image is of category "fake" instead of category "true."

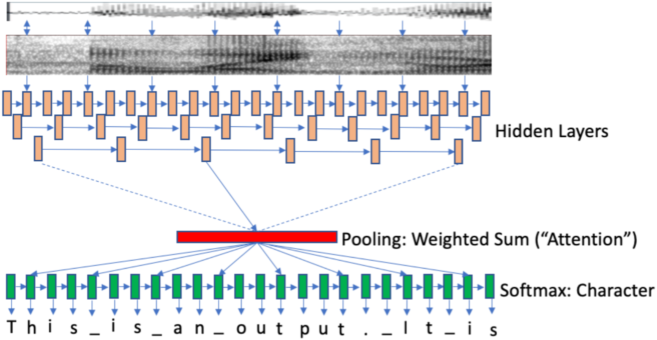
Intentionally Generating the Mistakes We Want



Credit: http://karpathy.github.io/2015/03/30/breaking-convnets/

Outline

Example of a Modern Speech Recognizer: Listen Attend & Spell (LAS). https://arxiv.org/abs/1508.01211



- Input: MFCC vectors $\vec{x}_t$ at time t.
- Output: English characters (letters, spaces, punctuation).
- $\hat{y}_{t,\text{true}}$ is the softmax output at time t, for the character it's supposed to output at time t.
- Training Criterion: $E = -\sum_t \ln \hat{y}_{t,\text{true}}$

A problem that all speech recognizers have: speaker variation

- The problem: LAS is sometimes fooled by differences between different speakers, e.g., if a speaker has an unusual pronunciation pattern, or a really deep voice or something.
- The solution: force the hidden layers to contain **as little information as possible** about the speaker ID, while still containing **as much information as possible** about the words.

What is "Adversarial Training"?

- The basic idea: make a neural network robust to some particular type of noise.
- How: Force one of its hidden layers to be really really **bad** at classifying that type of noise.
  1. Train an "adversary" neural net that observes the hidden layer, and from it, figures out which one of the noise signals is present in the input.
  2. Train the hidden layers in order to **increase** the error rate of the adversary.

Adversarial Training: General Idea

- ADVERSARY: The adversary tries to minimize its error rate,

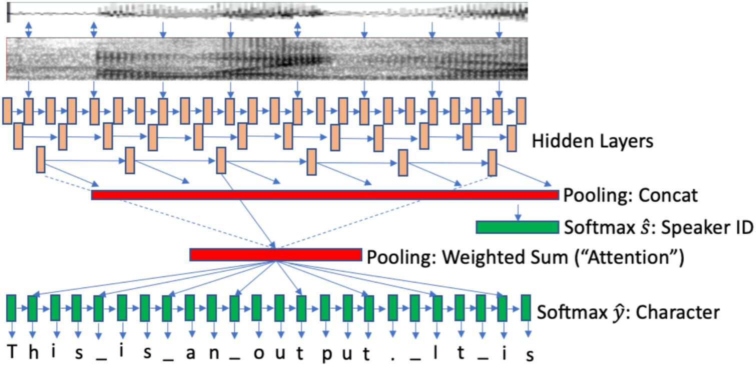
$$E_{\text{adversary}} = -\ln \hat{s}_{\text{true}}$$

- NOISE-ROBUST MAIN SYSTEM: The main system tries to **minimize** the primary error rate, while simultaneously **maximizing** the error rate of the adversary:

$$E_{\text{primary}} = -\ln \hat{y}_{\text{true}}$$

$$E_{\text{noise-robust-primary}} = E_{\text{primary}} - E_{\text{adversary}}$$

Example: LAS with Adversarial Training



$$E_{\text{adversary}} = -\ln \hat{s}_{\text{true}}$$

$$E_{\text{noise-robust-primary}} = \ln \hat{s}_{\text{true}} - \sum_t \ln \hat{y}_{t,\text{true}}$$

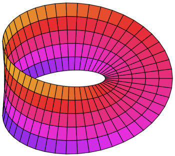
Example Research Topics

- Speech recognition, robust to speaker variation.
- . . . or background noise; or even language ID. . .
- Image style: Identify the person who painted a particular image, regardless of what type of object is in the painting.
- Melody extraction: identify the melody being played, regardless of what type of instrument is playing it.

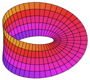
Outline

And now, for something completely different

Credit: https://commons.wikimedia.org/wiki/File:Moebius_strip.svg



Until now, we've been studying the relationship between $\vec{x}$ and $\vec{y}$. The goal of an auto-encoder is just to learn $\vec{x}$. Specifically, if $\vec{x} \in \Re^p$ is actually limited to a q-dimensional manifold, where $q < p$, then an auto-encoder learns the manifold.



Autoencoder: Basic Idea

Given input $\vec{x}_i \in \Re^p$, compute a shorter hidden state vector $\vec{z}_i = f(\vec{x}_i)$, where $\vec{z}_i \in \Re^q$, $q < p$, such that $\vec{z}_i$ captures all of the "useful" information about $\vec{x}_i$.

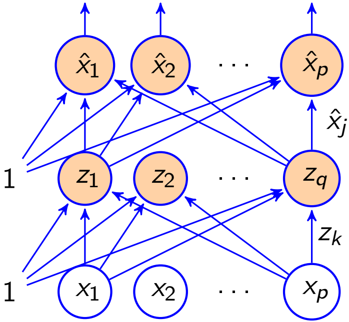
The Autoencoder Training Criterion: Mean Squared Error

$\vec{z}_i$ is passed through a second neural net to compute $\hat{x}_i = g(\vec{z}_i)$, and then we train the neural net to minimize

$$E = \frac{1}{n}\sum_{i=1}^{n} \|\vec{x}_i - \hat{x}_i\|^2$$

Two-Layer Linear Autoencoder

$$E = \|\vec{x} - \hat{x}\|^2 = \sum_{j=1}^{p}(x_j - \hat{x}_j)^2$$



$$\hat{x}_j = u_{j0} + \sum_{k=1}^{q} u_{kj} z_k \qquad \hat{x} = U^T \vec{z}$$

$$z_k = \sum_{j=1}^{p} u_{kj}(x_j - u_{j0}) \qquad \vec{z} = U\vec{x}$$

$\vec{x}$ is the input vector

Analyzing the Two-Layer Linear Autoencoder

Define the data matrices:

$$\begin{aligned} X &= [\vec{x}_1, \ldots, \vec{x}_n] \\ Z &= [\vec{z}_1, \ldots, \vec{z}_n] = UX \\ \hat{X} &= [\hat{x}_1, \ldots, \hat{x}_n] = U^T Z = U^T UX \end{aligned}$$

Then the error criterion is

$$E = \frac{1}{n}\sum_i (\vec{x}_i - \hat{x}_i)^T (\vec{x}_i - \hat{x}_i) = \frac{1}{n}\text{trace}\left((X - \hat{X})^T (X - \hat{X})\right)$$

$$E = \frac{1}{n}\text{trace}\left((X - \hat{X})^T(X - \hat{X})\right)$$

By the trace equality,

$$\begin{aligned} E &= \frac{1}{n}\text{trace}\left((X - \hat{X})(X - \hat{X})^T\right) \\ &= \frac{1}{n}\text{trace}\left(XX^T - U^TUXX^T - XX^TU^TU + U^TUXX^TU^TU\right) \end{aligned}$$

Covariance matrix:

$$\Sigma = \frac{1}{n} X X^T = \frac{1}{n} \sum_{i=1}^{n} \vec{x}_i \vec{x}_i^T$$

Then the auto-encoder training criterion is just

$$E = \text{trace}\left(\Sigma - U^T U \Sigma - \Sigma U^T U + U^T U \Sigma U^T U\right)$$

Suppose we set $U = [\vec{u}_1, \ldots, \vec{u}_q]^T$ to be the first q eigenvectors of Σ (the ones with highest eigenvalues, λ_j). Then

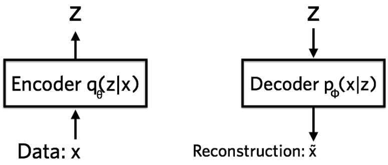
$$E = \text{trace}\left(\Sigma - \sum_{k=1}^{q} \lambda_k \vec{u}_k \vec{u}_k^T\right)$$

...and any other choice has a worse error! Therefore the unique optimum value of U is a principal component analysis.

Deep Autoencoder

If an autoencoder has more than two layers, then it finds a sort of "nonlinear principal components:" a nonlinear manifold, $\vec{z} = f(\vec{x})$, that minimizes the error term

$$E = \frac{1}{n} \sum_{i=1}^{n} \|\vec{x}_i - g(\vec{z}_i)\|^2$$



Credit: https://jaan.io/what-is-variational-autoencoder-vae-tutorial/

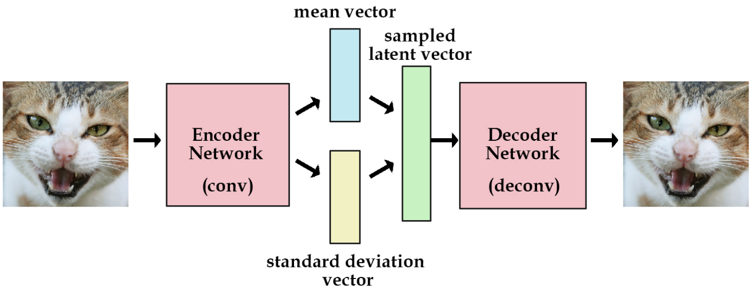
Outline

Autoencoder pros and cons

- Things that work:
  - The reconstruction, $\hat{x} = g(\vec{z})$, reconstructs $\hat{x} \approx \vec{x}$ with the smallest possible MSE.
  - In that sense, the hidden vector $\vec{z}$ (often called the "embedding") represents as much information about $\vec{x}$ as it's possible to represent in a q-dimensional vector.
- Things that fail:
  - The input space $\Re^p$ is infinite, but the training dataset X is finite; with enough trainable parameters, a deep auto-encoder can learn an embedding such that every training token is reconstructed with zero error. That's not very interesting.
  - If you pick some other $\vec{z}$ at random and generate $g(\vec{z})$, you don't get a very good image.

The Solution: Variational Autoencoder

Instead of just $\vec{z} = f(\vec{x})$, a VAE learns $(\vec{\mu}, \Sigma) = f(\vec{x})$. It then forces $\mu \approx \vec{0}$ and $\Sigma \approx I$, so that we can use $\vec{z} \sim \mathcal{N}(\vec{\mu}, \Sigma)$ to generate "fake" images that are similar to the real ones.

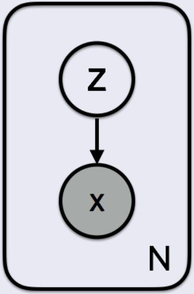


Credit: https://www.doc.ic.ac.uk/ js4416/163/website/autoencoders/variational.html

VAE Generative Model

Credit: https://jaan.io/what-is-variational-autoencoder-vae-tutorial/

VAE Generative Model



VAE Generative Model

For each token in the training database:

- PRIOR: Choose a mean and covariance, $(\vec{\mu}_i, \Sigma_i) \sim p(\vec{\mu}, \Sigma)$.
- HIDDEN: Choose a hidden vector $\vec{z}_i \sim p(\vec{z}|\vec{\mu}, \Sigma)$.
- OBSERVED: Choose an observed vector $\vec{x}_i \sim p(\vec{x}|\vec{z})$.

VAE Generative Model

- PRIOR: $\vec{\mu}$ is Gaussian, with zero mean and identity covariance. Σ is inverse-Wishart, with identity mean.

$$p(\vec{\mu}_i, \Sigma_i) \propto \prod_{k=1}^{q} \sigma_{ik} e^{-\frac{1}{2}\left(\mu_{ik}^2 + \sigma_{ik}^2 - 1\right)}$$

- HIDDEN: $\vec{z}_i$ is Gaussian, with mean $\vec{\mu}_i$ and covariance Σ_i.

$$p(\vec{z}_i | \vec{\mu}_i, \Sigma_i) = \mathcal{N}\left(\vec{\mu}_i, \Sigma_i\right)$$

- OBSERVED: $\vec{x}_i$ is Gaussian, with mean $g(\vec{z}_i)$, and identity covariance.

$$p(\vec{x}_i | \vec{z}_i) \propto e^{-\frac{1}{2}\|\vec{x}_i - g(\vec{z}_i)\|^2}$$

VAE Training Procedure

- SAMPLE X: choose $\vec{x}_i$ from the training database.
- GENERATE MU, SIGMA as $[\mu_{ij}, \sigma_{ij}] = f(\vec{x}_i)$, then penalize their error:

$$E_i^{(f)} = -\ln p(\vec{\mu}_i, \Sigma_i) = \frac{1}{2}\sum_{k=1}^{q}\left(\mu_{ik}^2 + \sigma_{ik}^2 - \ln\sigma_{ik}^2 - 1\right)$$

- SAMPLE Z: randomly from the distribution $\vec{z}_i \sim \mathcal{N}(\vec{\mu}_i, \Sigma_i)$.
- GENERATE X-HAT as $\hat{x}_i = g(\vec{z}_i)$, then penalize its error

$$E_i^{(g)} = -\ln p(\vec{x}_i|\vec{z}_i) = \frac{1}{2}\sum_{j=1}^{p}(x_{ij} - \hat{x}_{ij})^2$$

- TOTAL: The total cost function is

$$E_i = E_i^{(f)} + E_i^{(g)}$$

VAE Generative Tests

The result of training is that you can generate pretty good new images by doing the following:

- Generate $(\vec{\mu}, \Sigma)$ at random according to the known prior,
- Generate $\vec{z}$ at random as $\mathcal{N}(\vec{\mu}, \Sigma)$,
- Generate $\hat{x} = g(\vec{z})$ with your neural net.

Credit: https://www.youtube.com/watch?v=XNZIN7Jh3Sg

Outline

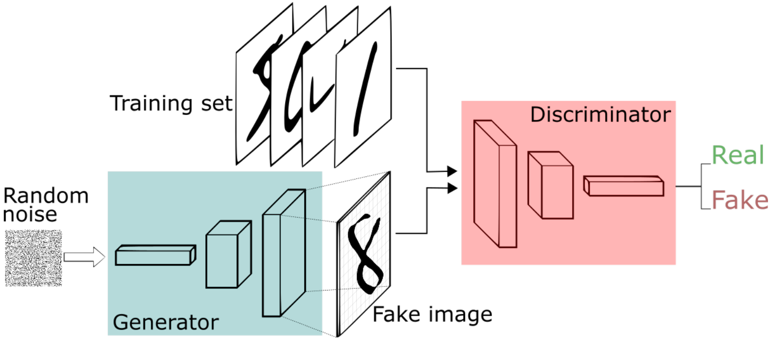
- **Review: Data Augmentation**
  - For every example in your training corpus, $\vec{x}_i$ with label $\vec{y}_i$,...
  - generate as many "fake examples" as you can, $\hat{x}_i$, such that all of the fake examples have the same label...
  - then re-train your network using these new fake examples, as well as the real examples.
- **Data Augmentation Using a VAE**
  - Can we use a VAE to generate "fake examples" for data augmentation?
  - THE PROBLEM: VAE doesn't know what types of variability will change the label.
  - POSSIBLE SOLUTION: Can we train another network, to tell us whether the fake example has the same label or not?

Generative Adversarial Network (GAN)

Steps to train a GAN:

1. Train a generator to generate fake examples.
2. Train an adversary to distinguish fake versus real training examples.
3. Re-train the whole thing all together:
   - The adversary is trying to correctly distinguish true data versus fake data.
   - The generator is trying to generate fake data that fools the adversary.

Generative Adversarial Network (GAN)



Credit: https://skymind.ai/wiki/generative-adversarial-network-gan

GAN Training Criterion

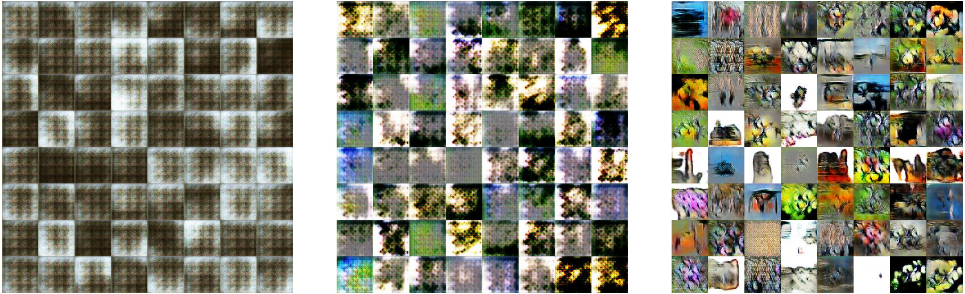
- ADVERSARY: Suppose that $y_i = 1$ if $\vec{x}_i$ is a true image, and $y_i = 0$ if $\vec{x}_i$ is a fake image. Suppose the adversary computes $\hat{y}_i = D(\vec{x}_i, \theta)$. The adversary wants to minimize the cross-entropy $H(y_i \| \hat{y}_i) = -y_i \ln \hat{y}_i - (1 - y_i) \ln(1 - \hat{y}_i)$:

$$\theta \leftarrow \theta - \frac{\partial H(y_i \| \hat{y}_i)}{\partial \theta}$$

- GENERATOR: The generator wants to make fake images $\hat{x}_i = g(\vec{z}_i)$ that fool the adversary, i.e., it wants to MAXIMIZE the cross-entropy:

$$g \leftarrow g + \frac{\partial H(y_i \| \hat{y}_i)}{\partial g}$$

GAN: How well does it work?



Imagenet fake images generated by a GAN on epochs 300, 800, and 5800.

Credit: http://kvfrans.com/generative-adversial-networks-explained/

.

Outline

Conclusions

- Adversarial images: modify the image in order to increase ConvNet error.
- Adversarial training: make the hidden layers robust to noise by training them to fool an adversary.
- Auto-encoder: a two-layer auto-encoder is PCA. A deep auto-encoder is a kind of nonlinear PCA.
- Variational autoencoder: Force your autoencoder to have a latent space distributed like $\vec{z} \sim \mathcal{N}(0, I)$, so that you can easily generate realistic fake images.
- Generative adversarial network: Train the VAE so it can fool a "real versus fake" discriminative adversary.