Lecture 14:

Videos

Unsupervised Learning
Administrative

- Everyone should be done with Assignment 3 now
- Milestone grades will go out soon
Last class

Segmentation

Spatial Transformer

Soft Attention
Videos
ConvNets for images
Feature-based approaches to Activity Recognition

Dense trajectories and motion boundary descriptors for action recognition
Wang et al., 2013

Action Recognition with Improved Trajectories
Wang and Schmid, 2013

(code available!)

Dense trajectories
Dense trajectories and motion boundary descriptors for action recognition

Wang et al., 2013

detect feature points  track features with optical flow  extract HOG/HOF/MBH features in the (stabilized) coordinate system of each tracklet
Dense trajectories and motion boundary descriptors for action recognition
Wang et al., 2013

[J. Shi and C. Tomasi, “Good features to track,” CVPR 1994]
[Ivan Laptev 2005]
Dense trajectories and motion boundary descriptors for action recognition
Wang et al., 2013

track each keypoint using optical flow.

[G. Farnebäck, “Two-frame motion estimation based on polynomial expansion,” 2003]
Dense trajectories and motion boundary descriptors for action recognition
Wang et al., 2013

Extract features in the local coordinate system of each tracklet.

Accumulate into histograms, separately according to multiple spatio-temporal layouts.
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images

**First layer (CONV1):** 96 11x11 filters applied at stride 4

=>

Output volume [55x55x96]

Q: What if the input is now a small chunk of video? E.g. [227x227x3x15]?
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images

**First layer** (CONV1): 96 11x11 filters applied at stride 4

=>

Output volume **[55x55x96]**

Q: What if the input is now a small chunk of video? E.g. [227x227x3x15]?

A: Extend the convolutional filters in time, perform spatio-temporal convolutions!

E.g. can have 11x11xT filters, where T = 2..15.
Spatio-Temporal ConvNets

Figure 3. A 3D CNN architecture for human action recognition. This architecture consists of 1 hardwired layer, 3 convolution layers, 2 subsampling layers, and 1 full connection layer. Detailed descriptions are given in the text.

[3D Convolutional Neural Networks for Human Action Recognition, Ji et al., 2010]
Spatio-Temporal ConvNets

Sequential Deep Learning for Human Action Recognition, Baccouche et al., 2011
Spatio-Temporal ConvNets

spatio-temporal convolutions; worked best.

[Large-scale Video Classification with Convolutional Neural Networks, Karpathy et al., 2014]
Spatio-Temporal ConvNets

Learned filters on the first layer

[Large-scale Video Classification with Convolutional Neural Networks, Karpathy et al., 2014]
Spatio-Temporal ConvNets

Large-scale Video Classification with Convolutional Neural Networks, Karpathy et al., 2014

1 million videos
487 sports classes
Spatio-Temporal ConvNets

<table>
<thead>
<tr>
<th>Model</th>
<th>Clip Hit@1</th>
<th>Video Hit@1</th>
<th>Video Hit@5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature Histograms + Neural Net</td>
<td>-</td>
<td>55.3</td>
<td>-</td>
</tr>
<tr>
<td>Single-Frame</td>
<td>41.1</td>
<td><strong>59.3</strong></td>
<td>77.7</td>
</tr>
<tr>
<td>Single-Frame + Multires</td>
<td>42.4</td>
<td>60.0</td>
<td>78.5</td>
</tr>
<tr>
<td>Single-Frame Fovea Only</td>
<td>30.0</td>
<td>49.9</td>
<td>72.8</td>
</tr>
<tr>
<td>Single-Frame Context Only</td>
<td>38.1</td>
<td>56.0</td>
<td>77.2</td>
</tr>
<tr>
<td>Early Fusion</td>
<td>38.9</td>
<td>57.7</td>
<td>76.8</td>
</tr>
<tr>
<td>Late Fusion</td>
<td>40.7</td>
<td>59.3</td>
<td>78.7</td>
</tr>
<tr>
<td>Slow Fusion</td>
<td><strong>41.9</strong></td>
<td>60.9</td>
<td><strong>80.2</strong></td>
</tr>
<tr>
<td>CNN Average (Single+Early+Late+Slow)</td>
<td>41.4</td>
<td>63.9</td>
<td>82.4</td>
</tr>
</tbody>
</table>

The motion information didn’t add all that much...

[Large-scale Video Classification with Convolutional Neural Networks, Karpathy et al., 2014]
Spatio-Temporal ConvNets
**Spatio-Temporal ConvNets**

**Figure 3. C3D architecture.** C3D net has 8 convolution, 5 max-pooling, and 2 fully connected layers, followed by a softmax output layer. All 3D convolution kernels are $3 \times 3 \times 3$ with stride 1 in both spatial and temporal dimensions. Number of filters are denoted in each box. The 3D pooling layers are denoted from pool1 to pool5. All pooling kernels are $2 \times 2 \times 2$, except for pool1 is $1 \times 2 \times 2$. Each fully connected layer has 4096 output units.

| Conv1a  | 64 | Pool1 | 64 | Conv2a | 128 | Pool2 | 128 | Conv3a | 256 | Pool3 | 256 | Conv3b | 256 | Pool4 | 256 | Conv4a | 512 | Pool5 | 512 | Conv4b | 512 | | Conv5a | 512 | Pool6 | 512 | Conv5b | 512 | Pool7 | 512 | fc6    | 4096 | fc7    | 4096 | softmax |

**3D VGGNet, basically.**

[Learning Spatiotemporal Features with 3D Convolutional Networks, Tran et al. 2015]
Spatio-Temporal ConvNets

(Two-Stream Convolutional Networks for Action Recognition in Videos, Simonyan and Zisserman 2014)

Spatio-Temporal ConvNets

<table>
<thead>
<tr>
<th></th>
<th>Spatial stream ConvNet</th>
<th>Temporal stream ConvNet</th>
<th>Two-stream model (fusion by averaging)</th>
<th>Two-stream model (fusion by SVM)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>73.0%</td>
<td>40.5%</td>
<td>86.9%</td>
<td>88.0%</td>
</tr>
<tr>
<td>Temporal stream ConvNet</td>
<td>83.7%</td>
<td>54.6%</td>
<td>58.0%</td>
<td></td>
</tr>
<tr>
<td>Two-stream model (fusion by averaging)</td>
<td>86.9%</td>
<td>58.0%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Two-stream model (fusion by SVM)</td>
<td>88.0%</td>
<td>59.4%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Two-stream version works much better than either alone.

[Two-Stream Convolutional Networks for Action Recognition in Videos, Simonyan and Zisserman 2014]

Long-time Spatio-Temporal ConvNets

All 3D ConvNets so far used local motion cues to get extra accuracy (e.g. half a second or so)

Q: what if the temporal dependencies of interest are much much longer? E.g. several seconds?
Long-time Spatio-Temporal ConvNets

(This paper was way ahead of its time. Cited 65 times.)

Sequential Deep Learning for Human Action Recognition, Baccouche et al., 2011
Long-time Spatio-Temporal ConvNets

(This paper was way ahead of its time. Cited 65 times.)

Sequential Deep Learning for Human Action Recognition, Baccouche et al., 2011
Long-time Spatio-Temporal ConvNets

[Long-term Recurrent Convolutional Networks for Visual Recognition and Description, Donahue et al., 2015]
Long-time Spatio-Temporal ConvNets

[Beyond Short Snippets: Deep Networks for Video Classification, Ng et al., 2015]
Summary so far

We looked at two types of architectural patterns:

1. Model temporal motion locally (3D CONV)
2. Model temporal motion globally (LSTM / RNN)

+ Fusions of both approaches at the same time.
Summary so far

We looked at two types of architectural patterns:

1. Model temporal motion locally (3D CONV)

2. Model temporal motion globally (LSTM / RNN)

+ Fusions of both approaches at the same time.

There is another (cleaner) way!
Finite temporal extent (neurons that are only a function of finitely many video frames in the past)

Infinite (in theory) temporal extent (neurons that are function of all video frames in the past)
Long-time Spatio-Temporal ConvNets

Beautiful:
All neurons in the ConvNet are recurrent.

\[
\begin{align*}
z_t^l &= \sigma(W_z^l \ast x_t^l + U_z^l \ast h_t^{l-1}), \\
r_t^l &= \sigma(W_r^l \ast x_t^l + U_r^l \ast h_t^{l-1}), \\
h_t^l &= \text{tanh}(W_t^l \ast x_t^l + U \ast (r_t^l \odot h_t^{l-1})), \\
h_t^l &= (1 - z_t^l)h_{t-1}^{l} + z_t^l h_t^l,
\end{align*}
\]

Only requires (existing) 2D CONV routines. No need for 3D spatio-temporal CONV.

[Delving Deeper into Convolutional Networks for Learning Video Representations, Ballas et al., 2016]
Long-time Spatio-Temporal ConvNets

Normal ConvNet:

[Delving Deeper into Convolutional Networks for Learning Video Representations, Ballas et al., 2016]
Long-time Spatio-Temporal ConvNets

[Delving Deeper into Convolutional Networks for Learning Video Representations, Ballas et al., 2016]

layer N

CONV

layer N+1 at previous timestep

CONV

RNN-like recurrence (GRU)

layer N+1
Long-time Spatio-Temporal ConvNets

Recall: RNNs

\[ h_t = f_W(h_{t-1}, x_t) \]

Vanilla RNN

\[ h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t) \]

GRU

\[
\begin{align*}
  z_t &= \sigma(W_z x_t + U_z h_{t-1}), \\
  r_t &= \sigma(W_r x_t + U_r h_{t-1}), \\
  \tilde{h}_t &= \tanh(Wx_t + U(r_t \odot h_{t-1}) \\
  h_t &= (1 - z_t)h_{t-1} + z_t\tilde{h}_t,
\end{align*}
\]

LSTM

\[
\begin{pmatrix}
  i \\
  f \\
  o \\
  g
\end{pmatrix} = \begin{pmatrix}
  \text{sigmoid} \\
  \text{sigmoid} \\
  \text{sigmoid} \\
  \text{tanh}
\end{pmatrix} W^l \begin{pmatrix}
  h_{t-1}^l \\
  h_{t-1}^l
\end{pmatrix}
\]

\[
\begin{align*}
  c_t^l &= f \odot c_{t-1}^l + i \odot g \\
  h_t^l &= o \odot \tanh(c_t^l)
\end{align*}
\]

[Delving Deeper into Convolutional Networks for Learning Video Representations, Ballas et al., 2016]
Long-time Spatio-Temporal ConvNets

Recall: RNNs

\[ h_t = f_W(h_{t-1}, x_t) \]

\[
\begin{align*}
    z_t &= \sigma(W_z x_t + U_z h_{t-1}), \\
    r_t &= \sigma(W_r x_t + U_r h_{t-1}), \\
    \tilde{h}_t &= \tanh(W x_t + U (r_t \odot h_{t-1})) \\
    h_t &= (1 - z_t) h_{t-1} + z_t \tilde{h}_t,
\end{align*}
\]

Matrix multiply

\[
\begin{align*}
    z_t^l &= \sigma(W_z^l x_t^l + U_z^l h_{t-1}^l), \\
    r_t^l &= \sigma(W_r^l x_t^l + U_r^l h_{t-1}^l), \\
    \tilde{h}_t^l &= \tanh(W^l x_t^l + U (r_t^l \odot h_{t-1}^l)), \\
    h_t^l &= (1 - z_t^l) h_{t-1}^l + z_t^l \tilde{h}_t^l,
\end{align*}
\]

[Delving Deeper into Convolutional Networks for Learning Video Representations, Ballas et al., 2016]
Finite temporal extent (neurons that are only a function of finitely many video frames in the past)

Infinite (in theory) temporal extent (neurons that are function of all video frames in the past)
i.e. we obtain:

Infinite (in theory) temporal extent
(neurons that are function of all video frames in the past)
Summary

- You think you need a Spatio-Temporal Fancy Video ConvNet
- STOP. Do you really?
- Okay fine: do you want to model:
  - **local motion?** (use 3D CONV), or
  - **global motion?** (use LSTM).
- Try out using Optical Flow in a second stream (can work better sometimes)
- Try out GRU-RCN! (imo best model)
Unsupervised Learning
Unsupervised Learning Overview

● Definitions
● Autoencoders
  ○ Vanilla
  ○ Variational
● Adversarial Networks
Supervised vs Unsupervised

Supervised Learning

Data: \((x, y)\)
x is data, y is label

Goal: Learn a function to
map \(x \rightarrow y\)

Examples: Classification,
regression, object detection,
semantic segmentation, image
captioning, etc
Supervised vs Unsupervised

**Supervised Learning**

**Data:** \((x, y)\)  
\(x\) is data, \(y\) is label

**Goal:** Learn a *function* to map \(x \rightarrow y\)

**Examples:** Classification, regression, object detection, semantic segmentation, image captioning, etc.

**Unsupervised Learning**

**Data:** \(x\)  
Just data, no labels!

**Goal:** Learn some *structure* of the data

**Examples:** Clustering, dimensionality reduction, feature learning, generative models, etc.
Unsupervised Learning

- Autoencoders
  - Traditional: feature learning
  - Variational: generate samples
- Generative Adversarial Networks: Generate samples
Autoencoders

Features

Input data

Encoder

z

x

Autoencoders
Autoencoders

Originally: Linear + nonlinearity (sigmoid)
Later: Deep, fully-connected
Later: ReLU CNN
Autoencoders

$z$ usually smaller than $x$ (dimensionality reduction)

- Originally: Linear + nonlinearity (sigmoid)
- Later: Deep, fully-connected
- Later: ReLU CNN

Input data $x$ → Features $z$ → Encoder
Autoencoders

Reconstructed input data

Features

Input data

Encoder

Decoder

xx

z

x

47

29 Feb 2016
Autoencoders

Originally: Linear + nonlinearity (sigmoid)
Later: Deep, fully-connected
Later: ReLU CNN (upconv)

Encoder: 4-layer conv
Decoder: 4-layer upconv

Input data  \[ x \]  \rightarrow \text{Encoder}  \rightarrow \text{Features}  \rightarrow \text{Decoder}  \rightarrow \text{Reconstructed input data}
Autoencoders

Originally: Linear + nonlinearity (sigmoid)
Later: Deep, fully-connected
Later: ReLU CNN (upconv)

Encoder / decoder sometimes share weights

Example:
\( \text{dim}(x) = D \)
\( \text{dim}(z) = H \)
\( w_e : H \times D \)
\( w_d : D \times H = w_e^T \)

Train for reconstruction with no labels!
Autoencoders

Loss function (Often L2)

Reconstructed input data

Decoder

Features

Encoder

Input data

Reconstructed input data

Loss function

Train for reconstruction with no labels!
Autoencoders

After training, throw away decoder!
Autoencoders

Use encoder to initialize a **supervised** model

Use encoder to initialize a supervised model

Feedforward

Input data

Features

Predicted Label

Loss function (Softmax, etc)

Encoder

Classifier

Fine-tune encoder jointly with classifier

Train for final task (sometimes with small data)
Autoencoders: Greedy Training

In mid 2000s layer-wise pretraining with Restricted Boltzmann Machines (RBM) was common.

Training deep nets was hard in 2006!

It is difficult to optimize the weights in nonlinear autoencoders that have multiple hidden layers (2–4). With large initial weights, autoencoders typically find poor local minima; with small initial weights, the gradients in the early layers are tiny, making it infeasible to train autoencoders with many hidden layers. If

Hinton and Salakhutdinov, “Reducing the Dimensionality of Data with Neural Networks”, Science 2006
Autoencoders: Greedy Training

In mid 2000s layer-wise pretraining with Restricted Boltzmann Machines (RBM) was common.

Training deep nets was hard in 2006!

It is difficult to optimize the weights in nonlinear autoencoders that have multiple hidden layers (2-4). With large initial weights, autoencoders typically find poor local minima; with small initial weights, the gradients in the early layers are tiny, making it infeasible to train autoencoders with many hidden layers. If

Hinton and Salakhutdinov, “Reducing the Dimensionality of Data with Neural Networks”, Science 2006
Autoencoders

Autoencoders can reconstruct data, and can learn features to initialize a supervised model.

Can we generate images from an autoencoder?
Variational Autoencoder

A Bayesian spin on an autoencoder - lets us generate data!

Assume our data \( \{x^{(i)}\}_{i=1}^{N} \) is generated like this:

\[
\begin{align*}
\text{Sample from true prior} &\quad \mathcal{p}_{\theta^*}(z) \\
\text{Sample from true conditional} &\quad \mathcal{p}_{\theta^*}(x \mid z^{(i)})
\end{align*}
\]

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014
Variational Autoencoder

A Bayesian spin on an autoencoder!

Assume our data $\{x^{(i)}\}_{i=1}^N$ is generated like this:

Sample from true prior $p_{\theta^*}(z)$

Sample from true conditional $p_{\theta^*}(x \mid z^{(i)})$

Intuition: $x$ is an image, $z$ gives class, orientation, attributes, etc.

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014
Variational Autoencoder

A Bayesian spin on an autoencoder!

Assume our data \( \{x^{(i)}\}_{i=1}^{N} \) is generated like this:

\[
\text{Sample from true prior } \quad p_{\theta^*}(z) \\
\text{Sample from true conditional } \quad p_{\theta^*}(x \mid z^{(i)}) \\
\text{Sample from } \{x^{(i)}\} \\
\text{Problem: Estimate } \theta \text{ without access to latent states } z^{(i)}
\]

Intuition: \( x \) is an image, \( z \) gives class, orientation, attributes, etc

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014
Variational Autoencoder

Prior: Assume $p_\theta(z)$ is a unit Gaussian

Kingma and Welling, ICLR 2014
Variational Autoencoder

**Prior:** Assume $p_{\theta}(z)$ is a unit Gaussian

**Conditional:** Assume $p_{\theta}(x \mid z)$ is a diagonal Gaussian, predict mean and variance with neural net

Kingma and Welling, ICLR 2014
Variational Autoencoder

**Prior:** Assume \( p_\theta(z) \) is a unit Gaussian

**Conditional:** Assume \( p_\theta(x \mid z) \) is a diagonal Gaussian, predict mean and variance with neural net

Mean and (diagonal) covariance of \( p_\theta(x \mid z) \)

Decoder network with parameters \( \theta \)

Latent state

Kingma and Welling, ICLR 2014
Variational Autoencoder

**Prior**: Assume $p_\theta(z)$ is a unit Gaussian

**Conditional**: Assume $p_\theta(x \mid z)$ is a diagonal Gaussian, predict mean and variance with neural net

Mean and (diagonal) covariance of $p_\theta(x \mid z)$

- $\mu^x$
- $\Sigma^x$

Fully-connected or upconvolutional

Latent state

Decoder network with parameters $\theta$

Kingma and Welling, ICLR 2014
Variational Autoencoder: Encoder

By Bayes Rule the posterior is:

\[ p_\theta(z \mid x) = \frac{p_\theta(x \mid z)p_\theta(z)}{p_\theta(x)} \]

Kingma and Welling, ICLR 2014
Variational Autoencoder: Encoder

By Bayes Rule the posterior is:

$$p_\theta(z \mid x) = \frac{p_\theta(x \mid z) p_\theta(z)}{p_\theta(x)}$$

Use decoder network =)
Gaussian =)
Intractible integral =(  

Kingma and Welling,  
ICLR 2014
Variational Autoencoder: Encoder

By Bayes Rule the posterior is:

\[
p_\theta(z \mid x) = \frac{p_\theta(x \mid z) p_\theta(z)}{p_\theta(x)}
\]

Use decoder network =)
Gaussian =)
Intractible integral =()

Mean and (diagonal) covariance of

\[
q_\phi(z \mid x) = \mu^z \quad \Sigma^z
\]

Encoder network with parameters \(\phi\)

Data point
Variational Autoencoder: Encoder

By Bayes Rule the posterior is:

\[ p_\theta(z \mid x) = \frac{p_\theta(x \mid z)p_\theta(z)}{p_\theta(x)} \]

Use decoder network =)
Gaussian =)
Intractible integral =(

Mean and (diagonal) covariance of
\[ q_\phi(z \mid x) = \mu^z \Sigma^z \]

Encoder network with parameters \( \phi \)

Approximate posterior with encoder network \( q_\phi(z \mid x) \)

Data point

Kingma and Welling, ICLR 2014
Variational Autoencoder: Encoder

By Bayes Rule the posterior is:

\[
p_\theta(z \mid x) = \frac{p_\theta(x \mid z)p_\theta(z)}{p_\theta(x)}
\]

Use decoder network =)
Gaussian =)
Intractible integral =(

Mean and (diagonal) covariance of
\[
q_\phi(z \mid x) = \mu^z \quad \Sigma^z
\]

Encoder network with parameters \( \phi \)

Approximate posterior with encoder network
\[
q_\phi(z \mid x)
\]

Fully-connected or convolutional

Kingma and Welling, ICLR 2014
Variational Autoencoder

Data point $x$

Kingma and Welling, ICLR 2014
Variational Autoencoder

Encoder network

Data point

Mean and (diagonal) covariance of $q_\phi(z | x)$

$\mu^z$ $\Sigma^z$

Kingma and Welling, ICLR 2014
Variational Autoencoder

Sample from $q_\phi(z \mid x)$

Mean and (diagonal) covariance of $q_\phi(z \mid x)$

Kingma and Welling, ICLR 2014
Variational Autoencoder

\[ \mathbf{\mu}^x \quad \Sigma^x \]

Decoder network

\[ \mathbf{z} \]

Sample from \( q_\phi(z \mid x) \)

\[ \mathbf{\mu}^z \quad \Sigma^z \]

Encoder network

\[ \mathbf{x} \]

Mean and (diagonal) covariance of \( p_\theta(x \mid z) \)

Mean and (diagonal) covariance of \( q_\phi(z \mid x) \)

Kingma and Welling, ICLR 2014
Variational Autoencoder

Data point $x$ → Encoder network → Sample from $q_\phi(z \mid x)$ → $z$ → Decoder network → Sample from $p_\theta(x \mid z)$ → Reconstructed $x$

Mean and (diagonal) covariance of $p_\theta(x \mid z)$
Mean and (diagonal) covariance of $q_\phi(z \mid x)$

Kingma and Welling, ICLR 2014
Variational Autoencoder

Training like a normal autoencoder: reconstruction loss at the end, regularization toward prior in middle

Mean and (diagonal) covariance of \( p_\theta(x \mid z) \)
(should be close to data x)

Mean and (diagonal) covariance of \( q_\phi(z \mid x) \)
(should be close to prior \( p_\theta(z) \))

Kingma and Welling, ICLR 2014
Variational Autoencoder: Generate Data!

After network is trained:

Sample from prior $p_\theta(z)$
Variational Autoencoder: Generate Data!

After network is trained:

\[
\mu^x \quad \Sigma^x
\]

Decoder network

Sample from prior \( p_\theta(z) \)
Variational Autoencoder: Generate Data!

After network is trained:

Sample from $p_{\theta}(x | z)$

Sample from prior $p_{\theta}(z)$
Variational Autoencoder: Generate Data!

After network is trained:

- Sample from prior $p_\theta(z)$
- Sample from $p_\theta(x \mid z)$
- Generated $x$
Variational Autoencoder: Generate Data!

After network is trained:

Sample from $p_{\theta}(x | z)$

Sample from prior $p_{\theta}(z)$

Generated data $x$

Decoder network
Variational Autoencoder: Generate Data!

After network is trained:

Generated $xx$

Sample from $p_\theta(x \mid z)$

Decoder network

Sample from prior $p_\theta(z)$

Diagonal prior on $z \Rightarrow$ independent latent variables
Variational Autoencoder: Math
Maximum Likelihood?

\[ \theta^* = \arg \max_{\theta} \prod_{i=1}^{N} p_{\theta}(x^{(i)}) \]
Maximize likelihood of dataset \( \{x^{(i)}\}_{i=1}^{N} \)

Kingma and Welling, ICLR 2014
Variational Autoencoder: Math

Maximum Likelihood?

\[ \theta^* = \arg \max_\theta \prod_{i=1}^{N} p_\theta(x^{(i)}) \]

Maximize likelihood of dataset \( \{x^{(i)}\}_{i=1}^{N} \)

\[ = \arg \max_\theta \sum_{i=1}^{N} \log p_\theta(x^{(i)}) \]

Maximize log-likelihood instead because sums are nicer

Kingma and Welling, ICLR 2014
Variational Autoencoder: Math

Maximum Likelihood?

\[ \theta^* = \arg \max_{\theta} \prod_{i=1}^{N} p_\theta(x^{(i)}) \quad \text{Maximize likelihood of dataset} \quad \{x^{(i)}\}_{i=1}^{N} \]

\[ = \arg \max_{\theta} \sum_{i=1}^{N} \log p_\theta(x^{(i)}) \quad \text{Maximize log-likelihood instead because sums are nicer} \]

\[ p_\theta(x^{(i)}) = \int p_\theta(x^{(i)}, z) dz \quad \text{Marginalize joint distribution} \]

Kingma and Welling, ICLR 2014
Variational Autoencoder: Math

Maximum Likelihood?

\[ \theta^* = \arg \max_{\theta} \prod_{i=1}^{N} p_\theta(x^{(i)}) \]  
Maximize likelihood of dataset \( \{x^{(i)}\}_{i=1}^{N} \)

\[ = \arg \max_{\theta} \sum_{i=1}^{N} \log p_\theta(x^{(i)}) \]  
Maximize log-likelihood instead because sums are nicer

\[ p_\theta(x^{(i)}) = \int p_\theta(x^{(i)}, z) dz = \int p_\theta(x^{(i)} | z)p_\theta(z)dz \]  
Intractible integral \( = (\)
Variational Autoencoder: Math

\[ \log p_\theta(x^{(i)}) \]
Variational Autoencoder: Math

$$\log p_\theta(x^{(i)}) = E_{z \sim q_\phi(z|x^{(i)})} \left[ \log p_\theta(x^{(i)}) \right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$
Variational Autoencoder: Math

$$\log p_\theta(x^{(i)}) = \mathbb{E}_{z \sim q_\phi(z|x^{(i)})} \left[ \log p_\theta(x^{(i)}) \right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

$$= \mathbb{E}_z \left[ \log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})} \right] \quad (\text{Bayes’ Rule})$$
Variational Autoencoder: Math

\[
\log p_{\theta}(x^{(i)}) = E_{z \sim q_{\phi}(z | x^{(i)})} \left[ \log p_{\theta}(x^{(i)}) \right] \quad (p_{\theta}(x^{(i)}) \text{ Does not depend on } z)
\]

\[
= E_{z} \left[ \log \frac{p_{\theta}(x^{(i)} | z)p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} \right] \quad (\text{Bayes’ Rule})
\]

\[
= E_{z} \left[ \log \frac{p_{\theta}(x^{(i)} | z)p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} \frac{q_{\phi}(z | x^{(i)})}{q_{\phi}(z | x^{(i)})} \right] \quad (\text{Multiply by constant})
\]
Variational Autoencoder: Math

\[ \log p_\theta(x^{(i)}) = E_{z \sim q_\phi(z|x^{(i)})} \left[ \log p_\theta(x^{(i)}) \right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z) \]

\[ = E_z \left[ \log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \right] \quad \text{(Bayes’ Rule)} \]

\[ = E_z \left[ \log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \frac{q_\phi(z | x^{(i)})}{q_\phi(z | x^{(i)})} \right] \quad \text{(Multiply by constant)} \]

\[ = E_z \left[ \log p_\theta(x^{(i)} | z) \right] - E_z \left[ \log \frac{q_\phi(z | x^{(i)})}{p_\theta(z)} \right] + E_z \left[ \log \frac{q_\phi(z | x^{(i)})}{p_\theta(z | x^{(i)})} \right] \quad \text{(Logarithms)} \]
Variational Autoencoder: Math

\[
\log p_\theta(x^{(i)}) = \mathbb{E}_{z \sim q_\phi(z|x^{(i)})} \left[ \log p_\theta(x^{(i)}) \right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)
\]

\[
= \mathbb{E}_z \left[ \log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \right] \quad \text{(Bayes’ Rule)}
\]

\[
= \mathbb{E}_z \left[ \log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \frac{q_\phi(z | x^{(i)})}{q_\phi(z | x^{(i)})} \right] \quad \text{(Multiply by constant)}
\]

\[
= \mathbb{E}_z \left[ \log p_\theta(x^{(i)} | z) \right] - \mathbb{E}_z \left[ \log \frac{q_\phi(z | x^{(i)})}{p_\theta(z)} \right] + \mathbb{E}_z \left[ \log \frac{q_\phi(z | x^{(i)})}{p_\theta(z | x^{(i)})} \right] \quad \text{(Logarithms)}
\]

\[
= \mathbb{E}_z \left[ \log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) \| p_\theta(z)) + D_{KL}(q_\phi(z | x^{(i)}) \| p_\theta(z | x^{(i)}))
\]
Variational Autoencoder: Math

\[ \log p_\theta(x^{(i)}) = \mathbb{E}_{z \sim q_\phi(z \mid x^{(i)})} \left[ \log p_\theta(x^{(i)}) \right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z) \]

\[ = \mathbb{E}_z \left[ \log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})} \right] \quad \text{(Bayes’ Rule)} \]

\[ = \mathbb{E}_z \left[ \log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})} \frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})} \right] \quad \text{(Multiply by constant)} \]

\[ = \mathbb{E}_z \left[ \log p_\theta(x^{(i)} \mid z) \right] - \mathbb{E}_z \left[ \log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)} \right] + \mathbb{E}_z \left[ \log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})} \right] \quad \text{(Logarithms)} \]

\[ = \mathbb{E}_z \left[ \log p_\theta(x^{(i)} \mid z) \right] - D_{KL}(q_\phi(z \mid x^{(i)}) \mid \mid p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \mid \mid p_\theta(z \mid x^{(i)})) \]

\[ \mathcal{L}(x^{(i)}, \theta, \phi) \quad \text{“Elbow”} \]
Variational Autoencoder: Math

\[
\log p_\theta(x^{(i)}) = \mathbb{E}_{z \sim q_\phi(z|x^{(i)})} \left[ \log p_\theta(x^{(i)}) \right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)
\]

\[
= \mathbb{E}_z \left[ \log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \right] \quad \text{(Bayes’ Rule)}
\]

\[
= \mathbb{E}_z \left[ \log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} q_\phi(z | x^{(i)}) \right] \quad \text{(Multiply by constant)}
\]

\[
= \mathbb{E}_z \left[ \log p_\theta(x^{(i)} | z) \right] - \mathbb{E}_z \left[ \log \frac{q_\phi(z | x^{(i)})}{p_\theta(z)} \right] + \mathbb{E}_z \left[ \log \frac{q_\phi(z | x^{(i)})}{p_\theta(z | x^{(i)})} \right] \quad \text{(Logarithms)}
\]

\[
= \mathbb{E}_z \left[ \log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) \| p_\theta(z)) + D_{KL}(q_\phi(z | x^{(i)}) \| p_\theta(z | x^{(i)}))
\]

\[
\mathcal{L}(x^{(i)}, \theta, \phi) \quad \text{“Elbow”}
\]

\[
\geq 0
\]
Variational Autoencoder: Math

\[
\log p_\theta(x^{(i)}) = E_{z \sim q_\phi(z|x^{(i)})} \left[ \log p_\theta(x^{(i)}) \right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)
\]

\[
= E_z \left[ \log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \right] \quad (\text{Bayes’ Rule})
\]

\[
= E_z \left[ \log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \right] \cdot \frac{q_\phi(z | x^{(i)})}{q_\phi(z | x^{(i)})} \quad (\text{Multiply by constant})
\]

\[
= E_z \left[ \log p_\theta(x^{(i)} | z) \right] - E_z \left[ \log \frac{q_\phi(z | x^{(i)})}{p_\theta(z | x^{(i)})} \right] + E_z \left[ \log \frac{q_\phi(z | x^{(i)})}{p_\theta(z | x^{(i)})} \right] \quad (\text{Logarithms})
\]

\[
= E_z \left[ \log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) \| p_\theta(z)) + D_{KL}(q_\phi(z | x^{(i)}) \| p_\theta(z | x^{(i)})) \geq 0
\]

\[\mathcal{L}(x^{(i)}, \theta, \phi) \quad \text{“Elbow”}\]

\[
\log p_\theta(x^{(i)}) \geq \mathcal{L}(x^{(i)}, \theta, \phi)
\]

Variational lower bound (elbow)
Variational Autoencoder: Math

\[
\log p_\theta(x^{(i)}) = E_{z \sim q_\phi(z|x^{(i)})} \left[ \log p_\theta(x^{(i)}) \right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)
\]

\[
= E_z \left[ \log \frac{p_\theta(x^{(i)} | z) p_\theta(z)}{p_\theta(z | x^{(i)})} \right] \quad \text{(Bayes’ Rule)}
\]

\[
= E_z \left[ \log \frac{p_\theta(x^{(i)} | z) p_\theta(z)}{p_\theta(z | x^{(i)})} q_\phi(z | x^{(i)}) \right] \quad \text{(Multiply by constant)}
\]

\[
= E_z \left[ \log p_\theta(x^{(i)} | z) \right] - E_z \left[ \log \frac{q_\phi(z | x^{(i)})}{p_\theta(z)} \right] + E_z \left[ \log \frac{q_\phi(z | x^{(i)})}{p_\theta(z | x^{(i)})} \right] \quad \text{(Logarithms)}
\]

\[
\mathcal{L}(x^{(i)}, \theta, \phi) \quad \text{“Elbow”}
\]

\[
\mathcal{L}(x^{(i)}, \theta, \phi) \geq 0
\]

\[
\log p_\theta(x^{(i)}) \geq \mathcal{L}(x^{(i)}, \theta, \phi)
\]

Variational lower bound (elbow)

Training: Maximize lower bound
**Variational lower bound (elbow)**

Training: Maximize lower bound

\[ \theta^*, \phi^* = \arg \max_{\theta, \phi} \sum_{i=1}^{N} \mathcal{L}(x^{(i)}, \theta, \phi) \]

```
\[
\log p_{\theta}(x^{(i)}) \geq \mathcal{L}(x^{(i)}, \theta, \phi) = \mathbb{E}_z \left[ \log p_{\theta}(x^{(i)} | z) \right] - \mathbb{E}_z \left[ \log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z)} \right] - \text{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z))
\]
```

Bayes’ Rule

\[
\log p_{\theta}(x^{(i)}) = \mathbb{E}_z \log \frac{p_{\theta}(x^{(i)} | z)}{q_{\phi}(z | x^{(i)})} + \mathbb{E}_z \log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z)}
\]

Reconstruct the input data

\[
\log p_{\theta}(x^{(i)}) = \mathbb{E}_z \log \frac{p_{\theta}(x^{(i)} | z)}{q_{\phi}(z | x^{(i)})}
\]

```
(Variational Autoencoder: Math)

Reconstruct the input data

\[
\log p_{\theta}(x^{(i)}) = \mathbb{E}_z \log \frac{p_{\theta}(x^{(i)} | z)}{q_{\phi}(z | x^{(i)})}
\]

(Bayes’ Rule)

\[
\log p_{\theta}(x^{(i)}) = \mathbb{E}_z \log \frac{p_{\theta}(x^{(i)} | z)}{q_{\phi}(z | x^{(i)})}
\]

Does not depend on \( z \)

(Multiply by constant)

\[
\log p_{\theta}(x^{(i)}) = \mathbb{E}_z \log \frac{p_{\theta}(x^{(i)} | z)}{q_{\phi}(z | x^{(i)})}
\]

(Logarithms)

\[
\log p_{\theta}(x^{(i)}) = \mathbb{E}_z \log \frac{p_{\theta}(x^{(i)} | z)}{q_{\phi}(z | x^{(i)})}
\]
Variational Autoencoder: Math

Reconstruct the input data

\[
\log p_\theta(x^{(i)}) = \mathbb{E}_{z \sim q_\phi(z|x^{(i)})} \left[ \log p_\theta(x^{(i)}) \right]
\]
\[
= \mathbb{E}_z \left[ \log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \right]
\]
\[
= \mathbb{E}_z \left[ \log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \frac{q_\phi(z | x^{(i)})}{q_\phi(z | x^{(i)})} \right]
\]
\[
= \mathbb{E}_z \left[ \log p_\theta(x^{(i)} | z) \right] - \mathbb{E}_z \left[ \log \frac{q_\phi(z | x^{(i)})}{p_\theta(z | x^{(i)})} \right] + \mathbb{E}_z \left[ \log \frac{q_\phi(z | x^{(i)})}{p_\theta(z | x^{(i)})} \right]
\]
\[
= \mathbb{E}_z \left[ \log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) \parallel p_\theta(z)) + D_{KL}(q_\phi(z | x^{(i)}) \parallel p_\theta(z | x^{(i)})) \geq 0
\]

\[
\mathcal{L}(x^{(i)}, \theta, \phi) \quad \text{“Elbow”}
\]

\[
\log p_\theta(x^{(i)}) \geq \mathcal{L}(x^{(i)}, \theta, \phi)
\]

Variational lower bound (elbow)

\[
\theta^*, \phi^* = \arg \max_{\theta, \phi} \sum_{i=1}^{N} \mathcal{L}(x^{(i)}, \theta, \phi)
\]

Training: Maximize lower bound
Variational Autoencoder: Math

\[ \log p_\theta(x^{(i)}) = E_{z \sim q_\phi(z|x^{(i)})} \left[ \log p_\theta(x^{(i)}) \right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z) \]

Reconstruct the input data

\[ E_z \left[ \log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \right] \quad (\text{Bayes’ Rule}) \]

Latent states should follow the prior

Sampling with reparam. trick (see paper)

\[ E_z \left[ \log p_\theta(x^{(i)} | z) \right] - E_z \left[ \log \frac{q_\phi(z | x^{(i)})}{p_\theta(z | x^{(i)})} \right] + E_z \left[ \log \frac{q_\phi(z | x^{(i)})}{p_\theta(z | x^{(i)})} \right] \quad (\text{Logarithms}) \]

\[ \mathcal{L}(x^{(i)}, \theta, \phi) \quad \text{“Elbow”} \]

\[ \log p_\theta(x^{(i)}) \geq \mathcal{L}(x^{(i)}, \theta, \phi) \]

Variational lower bound (elbow)

\[ \theta^*, \phi^* = \arg \max_{\theta, \phi} \sum_{i=1}^{N} \mathcal{L}(x^{(i)}, \theta, \phi) \]

Training: Maximize lower bound
Variational Autoencoder: Math

Reconstruct the input data

Sampling with reparam. trick (see paper)

Latent states should follow the prior

Everything is Gaussian, closed form solution!

Variational lower bound (elbow)

\[
\log p_\theta(x^{(i)}) \geq \mathcal{L}(x^{(i)}, \theta, \phi)
\]

Training: Maximize lower bound

\[
\theta^*, \phi^* = \arg \max_{\theta, \phi} \sum_{i=1}^{N} \mathcal{L}(x^{(i)}, \theta, \phi)
\]
Autoencoder Overview

● Traditional Autoencoders
  ○ Try to reconstruct input
  ○ Used to learn features, initialize supervised model
  ○ Not used much anymore

● Variational Autoencoders
  ○ Bayesian meets deep learning
  ○ Sample from model to generate images
Generative Adversarial Nets

Can we generate images with less math?

Random noise $z$
Generative Adversarial Nets

Can we generate images with less math?


Fake image $X$

Generator

Random noise $Z$
Generative Adversarial Nets

Can we generate images with less math?

Real or fake? y

Discriminator

Fake image x

Generator

Random noise z

Generative Adversarial Nets

Can we generate images with less math?

Real or fake? $y$

Discriminator

Fake image $x$

Generator

Random noise $z$

Fake examples: from generator
Real examples: from dataset

Generative Adversarial Nets

Can we generate images with less math?

Train generator and discriminator jointly
After training, easy to generate images

Real or fake?

Discriminator

Fake image

Generator

Real image

Fake examples: from generator
Real examples: from dataset

Random noise

Generative Adversarial Nets

Generated samples

Nearest neighbor from training set

Generative Adversarial Nets

Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Nearest neighbor from training set

Generated samples (CIFAR-10)
Generative Adversarial Nets: Multiscale

Generative Adversarial Nets: Multiscale

Generative Adversarial Nets: Multiscale

Generative Adversarial Nets: Multiscale


Fei-Fei Li & Andrej Karpathy & Justin Johnson

Lecture 14 - 109   29 Feb 2016
Generative Adversarial Nets: Multiscale

Generative Adversarial Nets: Multiscale


Fei-Fei Li & Andrej Karpathy & Justin Johnson
Generative Adversarial Nets: Multiscale

Generative Adversarial Nets: Multiscale

Discriminators work at every scale!

Denton et al, NIPS 2015
Generative Adversarial Nets: Multiscale

Train separate model per-class on CIFAR-10

Denton et al, NIPS 2015
Generative Adversarial Nets: Simplifying

Generator is an upsampling network with fractionally-strided convolutions
Discriminator is a convolutional network

Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

Generative Adversarial Nets: Simplifying

Generator

Generative Adversarial Nets: Simplifying

Samples from the model look amazing!

Radford et al, ICLR 2016
Generative Adversarial Nets: Simplifying

Interpolating between random points in latent space

Radford et al, ICLR 2016
Generative Adversarial Nets: Vector Math

Samples from the model

Smiling woman  Neutral woman  Neutral man

Radford et al, ICLR 2016
Generative Adversarial Nets: Vector Math

Smiling woman  Neutral woman  Neutral man

Samples from the model

Average Z vectors, do arithmetic

Radford et al, ICLR 2016
Generative Adversarial Nets: Vector Math

Smiling woman  Neutral woman  Neutral man

Samples from the model

Average Z vectors, do arithmetic

Smiling Man

Radford et al, ICLR 2016
Generative Adversarial Nets: Vector Math

Glasses man  No glasses man  No glasses woman

Radford et al, ICLR 2016
Generative Adversarial Nets: Vector Math

Glasses man  No glasses man  No glasses woman

Woman with glasses

Radford et al, ICLR 2016
Putting everything together

Variational Autoencoder

Pixel loss

Putting everything together

Real or Generated

Discriminator network

Pixel loss

Variational Autoencoder

Putting everything together

Variational Autoencoder

Discriminator network

Pixel loss

Real or Generated

Pretrained AlexNet

Putting everything together

Real or Generated

Discriminator network → Pixel loss

Variational Autoencoder

\[ y \]

\[ x \]

\[ \mu^x \]

\[ \Sigma^x \]

\[ \mu^z \]

\[ \Sigma^z \]


Pretrained AlexNet

Features of real image → Features of reconstructed image

Fei-Fei Li & Andrej Karpathy & Justin Johnson
Putting everything together

Real or Generated

Discriminator network

Pixel loss

Variational Autoencoder

\[ x \rightarrow \mu^x, \Sigma^x \rightarrow z \rightarrow \mu^z, \Sigma^z \rightarrow x \]

Pretrained AlexNet

Features of real image

\[ x_f, x_{xf} \rightarrow \text{L2 loss} \]

Features of reconstructed image

Putting everything together

Samples from the model, trained on ImageNet

Recap

● Videos
● Unsupervised learning
  ○ Autoencoders: Traditional / variational
  ○ Generative Adversarial Networks
● Next time: Guest lecture from Jeff Dean