CS 188: Artificial Intelligence
Special Topics: NLP/CV/RL
Instructor: Nicholas Tomlin

[Slides courtesy of Dan Klein, Abigail See, Greg Durrett, Yejin Choi, John DeNero, Eric Wallace, Kevin Lin, Fei-Fei Li, Sergey Levine, Pieter Abbeel, and many others]
What tasks do we care about?

- Object detection and classification
- Semantic segmentation
- Image captioning
- Visual question answering
- Video classification and understanding
- Image generation
- ...

...
Image Classification

cat
dog
horse
person
airplane
house
...

[Image of a cat]
Beyond Image Classification

**Classification**
- CAT
  - No spatial extent

**Semantic Segmentation**
- GRASS, CAT, TREE, SKY
  - No objects, just pixels

**Object Detection**
- DOG, DOG, CAT
  - Multiple Object

**Instance Segmentation**
- DOG, DOG, CAT
  - This image is CC0 public domain
<table>
<thead>
<tr>
<th>TEXT PROMPT</th>
<th>an armchair in the shape of an avocado, an armchair imitating an avocado.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI-GENERATED IMAGES</td>
<td><img src="image.png" alt="Image of avocado-shaped armchairs" /></td>
</tr>
</tbody>
</table>
Recall: MNIST Digit Classification

Task specification:
- Input features: binary pixel values
- Output: a digit classification (0-9)

Issues with Naïve Bayes classifier:
- Can overfit to individual pixels
- Not robust to scaling, movement left/right, etc.
Convolutional Neural Networks

- **Image**: 28 x 28
- **Convolution**: padding = 1, kernel = 3x3, stride = 1 + ReLU
- **Max pooling**: Kernel = 2x2, Stride = 2
- **Convolution**: padding = 1, kernel = 3x3, stride = 1 + ReLU
- **Max pooling**: Kernel = 2x2, Stride = 2
- **Flatten**: 3136 x 128
- **Output**: 128 x 10
Convolution in 1D

- Basic idea: define a new function by averaging over a sliding window
- Example in one dimension: smoothing
Convolution in 1D

- Moving average:

\[ c[i] = \frac{1}{2r + 1} \sum_{j=i-r}^{i+r} a[j] \]

- Convolution: same idea but with weighted average

\[ (a \ast b)[i] = \sum_{j} a[j] \cdot b[i - j] \]

called a filter
Convolution in 1D

- Filters in one dimension:
  - Box filter: $[... , 0, 0, 1, 1, 1, 1, 0, 0,...]/5$
  - Gaussian filter: $[... , 0, 0, 1, 4, 6, 4, 1, 0, 0,...]/16$
Filters in two dimensions: same idea but apply over a square patch of inputs (often 3x3 or 5x5)

Applications:
- Blurring
- Sharpening
- Feature detection
- ...
Convolutional Neural Networks

- Key idea: learn the filter weights via backprop
Benchmarking on ImageNet

- 2010: Lin et al
- 2011: Sanchez & Perronnin
- 2012: Krizhevsky et al (AlexNet)
- 2013: Zeiler & Fergus
- 2014: Simonyan & Zisserman (VGG)
- 2014: Szegedy et al (GoogLeNet)
- 2015: He et al (ResNet)
- 2016: Shao et al
- 2017: Hu et al (SENNet)
- Human

First CNN-based winner: 152 layers, 152 layers, 152 layers

Layers:
- shallow
- 8 layers
- 19 layers
- 22 layers
- 3.6
- 3
- 2.3
- 5.1
ResNet (He, et al. 2015)

- **Key idea:**
  - Want deeper networks with more parameters, but training signal becomes weak
  - Add “skip” connections between layers so that there are shorter paths between early parameters and the final loss function

- **ResNet:**
  - 152-layer model for ImageNet
  - Massive improvement over all previous CNN-based classification models circa 2015
Image Classification

cat
dog
horse
person
airplane
house
...

The diagram shows a cat, which is classified under different categories.
Image Captioning

a cat standing on a desk
Input: Image I
Output: Sequence $y = y_1, y_2, ..., y_T$

Encoder: $h_0 = f_w(z)$
where $z$ is spatial CNN features
$f_w(\cdot)$ is an MLP

Decoder: $y_t = g_v(y_{t-1}, h_{t-1}, c)$
where context vector $c$ is often $c = h_0$

Image Captioning with RNNs + Attention

This entire process is differentiable.
- model chooses its own attention weights. No attention supervision is required.

Image Captioning with Transformers

**Input:** Image \( I \)

**Output:** Sequence \( y = y_1, y_2, \ldots, y_T \)

**Encoder:** \( c = T_w(z) \)
where \( z \) is spatial CNN features
\( T_w(\cdot) \) is the transformer encoder

Extract spatial features from a pretrained CNN

**Features:** \( H \times W \times D \)

**Decoder:** \( y_t = T_D(y_{0:t-1}, c) \)
where \( T_D(\cdot) \) is the transformer decoder

```
<table>
<thead>
<tr>
<th>y_0</th>
<th>y_1</th>
<th>y_2</th>
<th>y_3</th>
<th>y_4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>person</td>
<td>wearing</td>
<td>hat</td>
<td>[END]</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>y_0</th>
<th>y_1</th>
<th>y_2</th>
<th>y_3</th>
<th>y_4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Transformer decoder

```
<table>
<thead>
<tr>
<th>y_0</th>
<th>y_1</th>
<th>y_2</th>
<th>y_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>[START]</td>
<td>person</td>
<td>wearing</td>
<td>hat</td>
</tr>
</tbody>
</table>
```

Transformer encoder
Image Captioning with Vision Transformers

Need to learn these tokens
Representation Learning

1. Learn good feature extractors from self-supervised pretext tasks, e.g., predicting image rotations

2. Attach a shallow network on the feature extractor; train the shallow network on the target task with small amount of labeled data
Representation Learning: SimCLR

(a) Original  (b) Crop and resize  (c) Crop, resize (and flip)  (d) Color distort. (drop)
(e) Color distort. (jitter)

(f) Rotate \{90°, 180°, 270°\}  (g) Cutout  (h) Gaussian noise  (i) Gaussian blur
(j) Sobel filtering
Representation Learning: SimCLR

(a) Original (b) Crop and resize (c) Crop, resize (and flip) (d) Color distort. (drop) (e) Color distort. (jitter)

(f) Rotate \{90^\circ, 180^\circ, 270^\circ\} (g) Cutout (h) Gaussian noise (i) Gaussian blur (j) Sobel filtering
Key idea: take N images, make 2N augmented versions, and then try to learn all the pairwise matchings.
Key idea: treat latent space in image as a sequence of patches and learn to predict future patches from previous ones.
Autoencoders

Ideally they are identical. \( x \approx x' \)

An compressed low dimensional representation of the input.
Denoising Autoencoder

Original input

Partially destroyed input

Input

Reconstructed input

\[ x \approx x' \]

Encoder \( g_\phi \)

Decoder \( f_\theta \)

Bottleneck!

An compressed low dimensional representation of the input.
Variational Autoencoder

- **Input**
- **Probabilistic Encoder**
  - $q_\phi(z|x)$
  - Mean: $\mu$
  - Std. dev: $\sigma$
  - $z = \mu + \sigma \odot \epsilon$
  - $\epsilon \sim \mathcal{N}(0, I)$
- **Sampled latent vector**
- **Probabilistic Decoder**
  - $p_\theta(x|z)$
- **Reconstructed input**

Ideally they are identical.

$x \approx x'$

An compressed low dimensional representation of the input.

https://lilianweng.github.io/posts/2018-08-12-vae/
**Generative Adversarial Networks**

**Idea:** Train a **network** to guess which images are real and which are fake!

This model can then serve as a loss function for the generator!
Generative Adversarial Networks

1. get a “True” dataset $D_T = \{(x_i)\}$
2. get a generator $G_\theta(z)$
3. sample a “False” dataset $D_F$: $z \sim p(z)$, $x = G(z)$
4. update $D_\phi(x) = p_\phi(y|x)$ using $D_T$ and $D_F$ (1 SGD step)
5. use $D(x)$ to update $G(z)$ (1 SGD step)

“classic” GAN 2-player game:

$$
\min_G \max_D V(D, G) = E_{x \sim p_{\text{data}}(x)}[\log D(x)] + E_{z \sim p(z)}[\log(1 - D(G(z)))]
$$

$$
\approx \frac{1}{N} \sum_{i=1}^{N} \log D(x_i), \quad x_i \in D_T \\
\approx \frac{1}{N} \sum_{j=1}^{N} \log(1 - D(x_j)), \quad x_j = G(z_j)
$$
Diffusion Models
CLIP and DALL-E

1. Contrastive pre-training

2. Create dataset classifier from label text

3. Use for zero-shot prediction