CSE/STAT 416
Convolutional Neural Networks
Pre-Lecture Videos

Hunter Schafer
Paul G. Allen School of Computer Science & Engineering
University of Washington

May 3, 2023
Video 1
Recap Neural Networks
A lot of the buzz about ML recently has come from recent advancements in **deep learning**.

When people talk about “deep learning” they are generally talking about a class of models called **neural networks** that are a loose approximation of how our brains work.
Compute the output for input $(0, 1)$. There is a sign activation function on the hidden layers and output layer.

\[ v_1 = \text{sign} \left( 2 + 2 \cdot 0 + 3 \cdot 1 \right) = \text{sign}(5) = 1 \]
\[ v_2 = \text{sign} \left( 1 + 1 \cdot 0 - 2 \cdot 1 \right) = \text{sign}(-1) = 0 \]
\[ y = \text{sign} \left( -1 + 2 \cdot 1 + 3 \cdot 0 \right) = \text{sign}(1) = 1 \]
Neural Networks implicitly find these low level features for us!

Each layer learns more and more complex features

[Zeiler & Fergus ‘13]
Hyperparameter Optimization

How do we choose hyperparameters to train and evaluate?

Grid search:
- Hyperparameters on 2d uniform grid

Random search:
- Hyperparameters randomly chosen

Bayesian Optimization:
- Hyperparameters adaptively chosen
Images are extremely high dimensional

- **CIFAR-10** dataset are very small: \(3@32\times32\)
  - # inputs: \(3 \times 32 \times 32 = 3072\) input neurons

Hidden Layer of Size 4:
\[
3072 \cdot 4 + 4 = 12,791\] parameters

- For moderate sized images: \(3@200\times200\)
  - # inputs:
\[
3 \cdot 200 \cdot 200 = 120,000
\]

Images are structured, we should leverage this.
Idea: Reduce the number of weights that need to be learned by looking at local neighborhoods of image.

Use the idea of a convolution to reduce the number of inputs by combing information about local pixels.
Convolution

Use a **kernel** that slides across the image, computing the sum of the element-wise product between the kernel and the overlapping part of the image.

\[
3 \cdot 0 + 3 \cdot 1 + 2 \cdot 2 + 0 \cdot 2 + 0 \cdot 2 + 1 \cdot 0 + 3 \cdot 0 + 1 \cdot 1 + 2 \cdot 2 = 12
\]

Image

<table>
<thead>
<tr>
<th>3</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Kernel

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Output = \[
\begin{bmatrix}
12
\end{bmatrix}
\]
Convolution

The input image (blue), the kernel (dark blue, numbers lower right) slide over the image to produce a result (green)
Convolution

The input image (blue), the kernel (dark blue, numbers lower right) slide over the image to produce a result (green)
Convolution

The input image (blue), the kernel (dark blue, numbers lower right) slide over the image to produce a result (green)
Convolution

The input image (blue), the kernel (dark blue, numbers lower right) slide over the image to produce a result (green)
Convolution

The input image (blue), the kernel (dark blue, numbers lower right) slide over the image to produce a result (green)
Convolution

The input image (blue), the kernel (dark blue, numbers lower right) slide over the image to produce a result (green)
Convolution

The input image (blue), the kernel (dark blue, numbers lower right) slide over the image to produce a result (green)
Convolution

The input image (blue), the kernel (dark blue, numbers lower right) slide over the image to produce a result (green)
End of one convolution with a particular kernel

The input image (blue), the kernel (dark blue, numbers lower right) slide over the image to produce a result (green)
Special Kernels

The numbers in the kernels determine special properties:

- **Identity**:
  \[
  \begin{bmatrix}
  0 & 0 & 0 \\
  0 & 1 & 0 \\
  0 & 0 & 0 \\
  \end{bmatrix}
  \]

- **Edge Detection**:
  \[
  \begin{bmatrix}
  -1 & -1 & -1 \\
  -1 & 8 & -1 \\
  -1 & -1 & -1 \\
  \end{bmatrix}
  \]

- **Sharpen**:
  \[
  \begin{bmatrix}
  0 & -1 & 0 \\
  -1 & 5 & -1 \\
  0 & -1 & 0 \\
  \end{bmatrix}
  \]

- **Box Blur**:
  \[
  \frac{1}{9}
  \begin{bmatrix}
  1 & 1 & 1 \\
  1 & 1 & 1 \\
  1 & 1 & 1 \\
  \end{bmatrix}
  \]

Convolutional Neural Networks (CNNs) learn the right weights for each kernel they use! Generally not interpretable!
Hyper-parameters of a Single Convolution

You can specify a few more things about a kernel

- Kernel dimensions
- Padding size and padding values
- Stride (how far to jump) values

For example, a 3x3 kernel applied to a 5x5 image with 1x1 zero padding and a 2x2 stride
- Nobody
- Google Colab:

You get a GPU and you get a GPU
Everyone gets a GPU
Deep Learning

A lot of the buzz about ML recently has come from recent advancements in **deep learning**.

When people talk about “deep learning” they are generally talking about a class of models called **neural networks** that are a loose approximation of how our brains work.
Features in computer vision are local detectors
- Combine features to make prediction

In reality, these features are much more low level (e.g. Corner?)
A popular approach to computer vision was to make hand-crafted features for object detection.

Relies on coming up with these features by hand (yuck!)
NNs to the Rescue

Neural Networks implicitly find these low level features for us!

Each layer learns more and more complex features

[Zeiler & Fergus ‘13]
Idea: Reduce the number of weights that need to be learned by looking at local neighborhoods of image.

Use the idea of a convolution to reduce the number of inputs by combing information about local pixels.
You can specify a few more things about a kernel:
- Kernel dimensions
- Padding size and padding values
- Stride (how far to jump) values

For example, a 3x3 kernel applied to a 5x5 image with 1x1 zero padding and a 2x2 stride.
What is the result of applying a convolution using this kernel on this input image?

Use 1x1 zero padding and a 2x2 stride

<table>
<thead>
<tr>
<th>Image</th>
<th>Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4</td>
<td></td>
</tr>
<tr>
<td>5 6 7 8</td>
<td></td>
</tr>
<tr>
<td>9 10 11 12</td>
<td></td>
</tr>
<tr>
<td>13 14 15 16</td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td></td>
</tr>
<tr>
<td>0 2</td>
<td></td>
</tr>
</tbody>
</table>
Convolutional Neural Networks
Another core operation that is similar to a convolution is a pool.

- Idea is to down sample an image using some operation
- Combine local pixels using some operation (e.g. max, min, average, median, etc.)

Typical to use max pool with 2x2 filter and stride 2

- Tends to work better than average pool
Convolutional Neural Network

Combine convolutions and pools into pre-processing layers on image to learn a smaller, information dense representation.

Example architecture for hand-written digit recognition
- Each convolution section uses many different kernels (increasing depth of channels)
- Pooling layers downsample each channel separately
- Usually ends with fully connected neural network
Why does this help?

- Only need to learn a small number of values (kernel weights) that get applied to the entire image region by region
  - This is called weight-sharing
  - Gives efficiency + shift invariance

- Pooling lets us focus on features from larger and larger regions of the original image.
- **Input**: 10x10x1 image (grayscale image of 10x10 pixels)
- **Convolution**: 5x5 kernel, stride 1
- **MaxPool**: 2x2, stride 2
- What is the size of the resulting image?
- **Input:** 10x10x1 image (grayscale image of 10x10 pixels)
- **Convolution:** 5x5 kernel, stride 1
- **MaxPool:** 2x2, stride 2
- What is the size of the resulting image?
Number of Weights / Parameters

38
How does this work if there is more than one input channel?

- Usually, use a 3-dimensional tensor as the kernel to combine information from each input channel.

**Kernel:** $3 \times 3 \times 3$ (or $3 \times 3 \times 3$)

**#weights:** $3 \times 3 \times 3 = 27$, (with bias) $= 27 + 1 = 28$
Another way of thinking about this process is each kernel is a (hidden-layer) neuron that looks at the kernel-size pixels in a neighborhood.

If there are 5 output channels in a conv layer, only need to learn the weights for the 5 neurons.

- These neurons are a bit different since they look at the pixels that overlap with the window at each position.

Suppose 6x6 kernels

Total weights (w/o bias) = \(108 \cdot 5 = 540\) weights per neuron

\(6 \cdot 6 \cdot 3 = 108\)
Consider a plain neural network below, how many weights need to be learned?

Completely ignore intercept terms

Task: Binary Classification

Total w/o bias
\[15 \times 10 + 10 \times 5 + 5 \times 2 = 210\]

Total w/ bias
\[210 + 10 + 5 + 2 = 227\]
Consider solving a digit recognition task on 28x28 grayscale images. Suppose I wanted to use a fully connected hidden layer with 84 neurons.

Without Convolutions:

\[
\begin{align*}
28 \times 28 &= 784 & 84 & & 10 \\
\text{Num weights (w/o bias)} &= 784 \times 84 + 84 \times 10 = 66,696
\end{align*}
\]
Weight Sharing

Consider solving a digit recognition task on 28x28 images. Suppose I wanted to use a fully connected hidden layer with 84 neurons

**With Convolutions** (assume n1=10, n2=20) (not counting intercepts)

\[ \text{# inputs flattened: } 4 \cdot 4 \cdot 20 = 320 \]

\[ \text{FC layers # weights} \]

\[ 320 \cdot 84 + 84 \cdot 10 = 27,720 \]
CNN Applications & Transfer Learning
CNNs usually* have architectures that look like the following

- A series of Convolution + Activation Functions and Pooling layers. It’s very common to do a pool after each convolution.
- Each set of operations lowers the size of the image but increases the number of features.
- Then after some number of these operations, flatten the image to work with the final neural network.
The learned kernels are exactly the “features” for computer vision! They start simple (corners, edges) and get more complex after more layers.

Example detectors learned

Example interest points detected

[Zeiler & Fergus ‘13]
CNNs have had remarkable success in practice

LeNet, 1990s
LeNet made 82 errors on MNIST (popular hand-written digit dataset of size 60K). 99.86% accuracy
CNN Success

ImageNet 2012 competition:
- 1.2M training images
- 1000 categories

Winner: SuperVision
- 8 layers, 60M parameters
- [Krizhevsky et al. ‘12]
- Top-5 Error: 17%
CNN Success

Won 2014 ImageNet challenge with 6.66% top-5 error rate

Huge CNN depth has proven helpful in recognition systems... Maybe because images contain hierarchical structure (faces contain eyes contain edges, etc.)
Applications

Image Classification

Input: $x$  
Image pixels

Output: $y$  
Predicted object

Object Localization

Scene Parsing [Farabet et al. ‘13]
Applications


Product Recommendation
Not discussed in class. See next slide

For each of the Computer Vision Tasks below, what do you think the output layer of the neural network would look like? What would each output neuron represent?

- **Image Classification**: Given an image with a single object, output the class of the object.

- **Object Localization**: Given an image with a single object, output the class and bounding box \((x,y,w,h)\) of the object.

- **Object Detection**: Given an image with possibly multiple objects, output the bounding box and class for each object.
**Image Classification**: Given an image with a single object, output the class of the object.

**Object Localization**: Given an image with a single object, output the class and bounding box \((x, y, w, h)\) of the object.

**Object Detection**: Given an image with possibly multiple objects, output the bounding box and class for each object.

**Output Layer**

- \(C\) neurons where \(C\) is the number of classes

Each neuron represents the probability that the image is of that class.

- \(C+4\) neurons
  - First \(C\) are same as last
  - Also need outputs for bounding box \((x, y, w, h)\)

More complex, search Youtube for YOLO algorithm explained!
If we don’t have a lot of data for Task 2, what can we do?

Idea: Use a model that was trained for one task to help learn another task.

- An old idea, explored for deep learning by Donahue et al. ‘14 & others
What is learned in a neural network?

Initial layers are low-level and very general.

- Usually not sensitive/specific to the task at hand

Neural net trained for Task 1: cat vs. dog

More generic
Can be used as feature extractor

Very specific
to Task 1
Should be ignored for other tasks
Transfer Learning

Share the weights for the general part of the network

Neural net trained for Task 1: cat vs. dog

Use simple classifier e.g., logistic regression, SVMs, nearest neighbor,...

More generic
Can be used as feature extractor

Very specific to Task 1
Should be ignored for other tasks

Keep weights fixed!

Re-train
If done successfully, transfer learning can really help. Can give you

- A higher start
- A higher slope
- A higher asymptote
Deep Learning in Practice
Pros

No need to manually engineer features, enable automated learning of features

Impressive performance gains
- Image processing
- Natural Language Processing
- Speech recognition

Making huge impacts in most fields
Cons

Requires a LOT of data

Computationally really expensive
  - Environmentally, extremely expensive (Green AI)

Hard to tune hyper-parameters
  - Choice of architecture (we’ve added even more hyper-parameters)
    - Size of kernels, stride, 0 padding, number of conv layers, depth of outputs of conv layers,
  - Learning algorithm

Still not very interpretable
While NNs have had amazing success, they also have some baffling failures.

“No one adds noise to things in real applications”

**Not true!**

- Hackers will hack
- Sensors (cameras) are noisy!
They even fail with “natural” transformations of images

NN Failures

Objects can be created to trick neural networks!
Datasets, like ImageNet, are generally biased. One approach is to augment your dataset to add random permutations of data to avoid bias.
Demo: Adversarial Neural Networks to Promote Fairness


Dataset: Adult UCI

- Predict whether a person’s income will be > $50K or ≤ $50K based on factors like:
  - Age
  - Education level
  - Marital status
  - Served in Armed Services?
  - Hours per week worked
  - Occupation sector
  - Etc.
Further Readings on Deep Learning

Dealing with Variable Length Sequences (e.g. language)
- Recurrent Neural Networks (RNNs)
- Long Short Term Memory Nets (LSTMs)
- [http://colah.github.io/posts/2015-08-Understanding-LSTMs/](http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

Reinforcement Learning
- [Google DeepMind AlphaGo Zero](https://deepmind.com/blog/alphago-zero)

Generative Adversarial Networks
- [How to learn synthetic data](https://www.tensorflow.org/tutorials/generative/adversarial_networks)

Green AI
Recap

Theme: Details of convolutional neural networks

Ideas:
- Convolutions
- MaxPool
- Number of Parameters in a (C)NN
- Weight Sharing
- CNN Applications
- Transfer Learning
- NN Failures
- Using NNs to promote algorithmic fairness