Topics:

• Convolutional Neural Networks: Past and Present
• Convolution Layers
Administrative:

• Assignment due on Sep 19\textsuperscript{th} (with 48hr grace period)
• Will release proposal template today
• Proposal due Sep 26\textsuperscript{th} 11:59pm \textbf{(No Grace Period)}
• Start finding a project team if you haven’t!
Recap: Vector derivatives

Scalar to Scalar
\[ x \in \mathbb{R}, \ y \in \mathbb{R} \]
Regular derivative:
\[ \frac{\partial y}{\partial x} \in \mathbb{R} \]
If \( x \) changes by a small amount, how much will \( y \) change?

Vector to Scalar
\[ x \in \mathbb{R}^N, \ y \in \mathbb{R} \]
Derivative is **Gradient**:
\[ \frac{\partial y}{\partial x} \in \mathbb{R}^N \]
For **each** element of \( x \), if it changes by a small amount, how much will \( y \) change?

Vector to Vector
\[ x \in \mathbb{R}^N, \ y \in \mathbb{R}^M \]
Derivative is **Jacobian**:
\[ \frac{\partial y}{\partial x} \in \mathbb{R}^{M \times N} \]
For **each** element of \( x \), if it changes by a small amount, how much will each element of \( y \) change?

Slide credit: Stanford CS231n Instructors
Backprop with Vectors

Matrix multiplication

matmul([1 x D_z], [[D_z x D_x]])

“local gradients”

[D_z x D_x] [D_z x D_y]

Loss L still a scalar!

“Downstream gradients”

What about \( \frac{\partial L}{\partial x} \) and \( \frac{\partial L}{\partial y} \)?

“Upstream gradient”

Slide credit: Stanford CS231n Instructors
Jacobians

Given a function \( f : \mathbb{R}^n \to \mathbb{R}^m \), we have the Jacobian matrix \( J \) of shape \( m \times n \), where \( J_{i,j} = \frac{\partial f_i}{\partial x_j} \)

\[
J = \begin{bmatrix}
\frac{\partial f}{\partial x_1} & \cdots & \frac{\partial f}{\partial x_n} \\
\nabla^T f_1 \\
\vdots \\
\nabla^T f_m
\end{bmatrix} = \begin{bmatrix}
\frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\
\vdots & \ddots & \vdots \\
\frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n}
\end{bmatrix}
\]
Backprop with Vectors

4D input $x$:

$$\begin{bmatrix}
1 \\
-2 \\
3 \\
-1
\end{bmatrix}$$

4D output $z$:

$$\begin{bmatrix}
1 \\
0 \\
3 \\
0
\end{bmatrix}$$

$f(x) = \max(0, x)$

\textit{(elementwise)}

Slide credit: Stanford CS231n Instructors
Backprop with Vectors

4D input x:

\[
\begin{bmatrix}
  1 \\
  -2 \\
  3 \\
  -1 \\
\end{bmatrix}
\]

4D output z:

\[
\begin{bmatrix}
  1 \\
  0 \\
  3 \\
  0 \\
\end{bmatrix}
\]

\( f(x) = \max(0,x) \) (elementwise)

4D dL/dx:

\[
\begin{bmatrix}
  4 & 0 & 5 & 0 \\
\end{bmatrix}
\]

[\( dL/dz \) [\( dz/dx \)]

\[
\begin{bmatrix}
  4 & -1 & 5 & 9 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  1 & 0 & 0 & 0 \\
  0 & 0 & 0 & 0 \\
  0 & 0 & 1 & 0 \\
  0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  4 & -1 & 5 & 9 \\
\end{bmatrix}
\]

Upstream gradient

Slide credit: Stanford CS231n Instructors
For element-wise ops, Jacobian is **sparse**: off-diagonal entries always zero!

Never **explicitly** form Jacobian -- instead use Hadamard (element-wise) multiplication

**f(x) = max(0, x) (elementwise)**

4D input x:

\[
\begin{bmatrix}
1 \\
-2 \\
3 \\
-1 \\
\end{bmatrix}
\]

4D output z:

\[
\begin{bmatrix}
1 \\
0 \\
3 \\
0 \\
\end{bmatrix}
\]

4D dL/dx:

\[
\begin{bmatrix}
4 & 0 & 5 & 0
\end{bmatrix}
\]

\[
\left(\frac{\partial L}{\partial x}\right)_i = \begin{cases} 
\left(\frac{\partial L}{\partial z}\right)_i & \text{if } x_i > 0 \\
0 & \text{otherwise}
\end{cases}
\]

4D [dL/dz]:

\[
\begin{bmatrix}
4 & -1 & 5 & 9
\end{bmatrix}
\]

Upstream gradient
Backprop with Matrices (or Tensors)

**“Local gradients”**

\[
\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x}
\]

\[
\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial y}
\]

\[
[(D_z \times M_z) \times (D_x \times M_x)]
\]

\[
[(D_z \times M_z) \times (D_y \times M_y)]
\]

**“Downstream gradients”**

\[
[D_x \times M_x]
\]

\[
[D_y \times M_y]
\]

**“Upstream gradient”**

For each element of \( z \), how much does it influence \( L \)?

Flatten the two matrices \( \rightarrow \) vector-vector gradients \( \rightarrow \) jacobian matrices!

Loss \( L \) still a scalar!

d\( L \)/d\( x \) always has the same shape as \( x \)!

Slide credit: Stanford CS231n Instructors
Summary (Lecture 5 – here):

• Neural networks, activation functions
• NNs as Universal Function Approximators
• Neurons as biological inspirations to DNNs
• Vector Calculus
• Backpropagation through vectors / matrices
Next: Convolutional Neural Networks
A bit of history...

The **Mark I Perceptron** machine was the first implementation of the perceptron algorithm.

The machine was connected to a camera that used 20×20 photocells to produce a 400-pixel image.

recognized letters of the alphabet

update rule:

\[ w_i(t + 1) = w_i(t) + \alpha (d_j - y_j(t))x_{j,i} \]

Frank Rosenblatt, ~1957: Perceptron
A bit of history...

Widrow and Hoff, ~1960: Adaline/Madaline
A bit of history...

\[ \frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial w_{ji}} \]

Rumelhart et al., 1986: First time back-propagation became popular

Illustration of Rumelhart et al., 1986 by Lane McIntosh, copyright CS231n 2017
A bit of history...

Reinvigorated research in Deep Learning

[Hinton and Salakhutdinov 2006]

Illustration of Hinton and Salakhutdinov 2006 by Lane McIntosh, copyright CS231n 2017

Slide credit: Stanford CS231n Instructors
First strong results

**Acoustic Modeling using Deep Belief Networks**
Abdel-rahman Mohamed, George Dahl, Geoffrey Hinton, 2010

**Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition**
George Dahl, Dong Yu, Li Deng, Alex Acero, 2012

**Imagenet classification with deep convolutional neural networks**

A bit of history:

Hubel & Wiesel, 1959
RECEPTIVE FIELDS OF SINGLE NEURONES IN THE CAT'S STRIATE CORTEX

1962
RECEPTIVE FIELDS, BINOCULAR INTERACTION AND FUNCTIONAL ARCHITECTURE IN THE CAT'S VISUAL CORTEX

1968...
Hierarchical organization

Simple cells: Response to light orientation

Complex cells: Response to light orientation and movement

Hypercomplex cells: response to movement with an end point

Illustration of hierarchical organization in early visual pathways by Lane McIntosh, copyright CS231n 2017

Slide credit: Stanford CS231n Instructors
A bit of history:

**Neocognitron**

[**Fukushima 1980**]

“sandwich” architecture (SCSCSC...)  
simple cells: modifiable parameters  
complex cells: perform pooling
A bit of history:
Gradient-based learning applied to document recognition
[LeCun, Bottou, Bengio, Haffner 1998]
A bit of history:

ImageNet Classification with Deep Convolutional Neural Networks

[Krizhevsky, Sutskever, Hinton, 2012]

“AlexNet”
Fast-forward to today: ConvNets are everywhere

Classification

Retrieval

Fast-forward to today: ConvNets are everywhere

Detection

Segmentation

Figures copyright Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun, 2015. Reproduced with permission.

[Faster R-CNN: Ren, He, Girshick, Sun 2015]


[Farabet et al., 2012]
Fast-forward to today: ConvNets are everywhere

**Autonomous Driving:** GPUs & specialized chips are fast and compact enough for on-board compute!

[Image of autonomous driving technology]


Fast-forward to today: ConvNets are everywhere

Images are examples of pose estimation, not actually from Toshev & Szegedy 2014. Copyright Lane McIntosh.

[Toshev, Szegedy 2014]

[Guo et al. 2014]

Fast-forward to today: ConvNets are everywhere

**Generalized convolution:** spatial convolution

Choi et al., 2019

Kamnitsas et al., 2015
Fast-forward to today: ConvNets are everywhere

Generalized convolution: \textbf{temporal convolution}

Bai et al., 2018
Fast-forward to today: ConvNets are everywhere

Generalized convolution: **graph convolution**

Kipf et al., 2017
A white teddy bear sitting in the grass

A man in a baseball uniform throwing a ball

A woman is holding a cat in her hand

A man riding a wave on top of a surfboard

A cat sitting on a suitcase on the floor

A woman standing on a beach holding a surfboard

[Neuraltalk2](https://github.com/MalteJung/Neuraltalk2)
“An avocado armchair”
Convolutional Neural Networks
The connectivity in linear layers doesn’t always make sense

How many parameters?

- $M \times N$ (weights) + $N$ (bias)

Hundreds of millions of parameters for just one layer

More parameters $\Rightarrow$ More data needed & slower to train / inference

Is this necessary?
Image features are spatially localized!

- Smaller features repeated across the image
  - Edges
  - Color
  - Motifs (corners, etc.)
- No reason to believe one feature tends to appear in a fixed location. Need to search in entire image.

Can we induce a **bias** in the design of a neural network layer to reflect this?
Convolution: A 1D Visual Example

From https://en.wikipedia.org/wiki/Convolution
Convolution

1-D Convolution is defined as the \textbf{integral} of the \textbf{product} of two functions after one is reflected about the y-axis and shifted.

Cross-correlation is convolution without the y-axis reflection.

\textbf{Intuitively}: given function $f$ and filter $g$. How similar is $g(-x)$ with the part of $f(x)$ that it’s operating on.

For ConvNets, we don’t flip filters, so we are really using Cross-Correlation Nets!

\textit{From https://en.wikipedia.org/wiki/Convolution}
Convolution in Computer Vision (non-Deep)

Locality of Features

Convolution with Gaussian Filter (Gaussian Blur)

Convolution with Sobel Filter (Edge Detection)
Convolution: A 1D Visual Example

\[ g() : \text{filter / pattern template} \]

\[ f() : \text{signal / observed data} \]

\[ f \ast g() : \text{how well data matches with the template} \]

For Convolution Layers in NN, think of:

- \[ g() \] as the weights to learn
- \[ f() \] as the input to the layer
- \[ f \ast g() \] as the output of the layer (result of convolution)

From https://en.wikipedia.org/wiki/Convolution
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

input

\[ Wx \]

10 x 3072 weights

activation

1 number: the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)
Convolution Layer

32x32x3 image -> preserve spatial structure
Convolution Layer

32x32x3 image

5x5x3 filter

Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”
Convolution Layer

32x32x3 image

5x5x3 filter

Filters always extend the full depth of the input volume

Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”
Convolution Layer

32x32x3 image
5x5x3 filter \( w \)

1 number: the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. \( 5 \times 5 \times 3 = 75 \)-dimensional dot product + bias)

\[ w^T x + b \]
Convolution Layer

3 32

3 32
Convolution Layer
Convolution Layer
Convolution Layer
Convolution Layer

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

activation map
Convolution Layer

Consider a second, green filter

32x32x3 image
5x5x3 filter

Convolve (slide) over all spatial locations

Activation maps

32
For example, if we had 6 5x5 filters, we’ll get 6 separate activation maps:

We stack these up to get a “new image” of size 28x28x6!
ConvNet is a sequence of Convolution Layers, interspersed with activation functions.

CONV, ReLU

e.g. 6 5x5x3 filters
ConvNet is a sequence of Convolution Layers, interspersed with activation functions.

- **CONV, ReLU** e.g. 6 5x5x3 filters
- **CONV, ReLU** e.g. 10 5x5x6 filters
- **CONV, ReLU**
Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].
We call the layer convolutional because it is related to convolution of two signals:

\[ f[x,y] \ast g[x,y] = \sum_{n_1} \sum_{n_2} f[n_1,n_2] \cdot g[x-n_1,y-n_2] \]

Elementwise multiplication and sum of a filter and the signal (image)
A closer look at spatial dimensions:

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

activation map
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter

The # of grid that the filter shifts is called **stride**.

E.g., here we have stride = 1
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter \textbf{with stride} = 1
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter with stride = 1

=> 5x5 output
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter with stride = 1

=> 5x5 output

But what about the features at the border?
In practice: Common to zero pad the border

e.g. input 7x7
3x3 filter, applied with stride 1
pad with 1 pixel border => what is the output?
In practice: Common to zero pad the border e.g. input 7x7
3x3 filter, applied with \textbf{stride 1}

\textbf{pad with 1 pixel} border => what is the output?

\textbf{7x7 output!}
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with \((F-1)/2\). (will preserve size spatially)
e.g. \(F = 3\) => zero pad with 1
   \(F = 5\) => zero pad with 2
   \(F = 7\) => zero pad with 3
In practice: Common to zero pad the border

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

e.g. input 7x7
3x3 filter, applied with stride 1
**pad with 1 pixel** border => what is the output?

7x7 output!

\[
N = \text{input dimension} \\
P = \text{padding size} \\
F = \text{filter size} \\
\text{Output size} = \frac{N - F + 2P}{\text{stride}} + 1 \\
= \frac{7 - 3 + 2 \times 1}{1} + 1 = 7
\]
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 2
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 2
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 2 => 3x3 output!
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied **with stride 3?**
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 3?

doesn’t fit!
cannot apply 3x3 filter on 7x7 input with stride 3.
Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 \Rightarrow (7 - 3)/1 + 1 = 5
stride 2 \Rightarrow (7 - 3)/2 + 1 = 3
stride 3 \Rightarrow (7 - 3)/3 + 1 = 2.33

With padding of 1 x 1:
stride 3 \Rightarrow (7 - 3 + 2)/3 + 1 = 3
Remember back to…
E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn’t work well.
Remember back to…

With padding, we can keep the same spatial feature dimension throughout the convolution layers.

CONV, ReLU

e.g. 6
5x5x3 filters with 2 x 2 padding

CONV, ReLU

e.g. 10
5x5x6 filters with 2 x 2 padding

CONV, ReLU

....
Examples time:

Input volume: $32 \times 32 \times 3$
Conv layer: 10 5x5 filters with stride 1, pad 2

Output volume size: ?
Examples time:

Input volume: \(32 \times 32 \times 3\)
Conv layer: \(10\) 5x5 filters with stride 1, pad 2

Output volume size:
\[(32+2\times2-5)/1+1 = 32\] spatially, so \(32 \times 32 \times 10\)
Examples time:

Input volume: $32 \times 32 \times 3$
Conv layer: 10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?
Examples time:

Input volume: $32\times32\times3$

10 $5\times5$ filters with stride 1, pad 2

Number of parameters in this layer?

Each filter has $5\times5\times3 + 1 = 76$ params (+1 for bias)

$=> 76 \times 10 = 760$
Convolution layer: summary

Let’s assume input is $W_1 \times H_1 \times C$

Conv layer needs 4 hyperparameters:
- Number of filters $K$
- The filter size $F$
- The stride $S$
- The zero padding $P$

This will produce an output of $W_2 \times H_2 \times K$

where:
- $W_2 = \frac{(W_1 - F + 2P)}{S} + 1$
- $H_2 = \frac{(H_1 - F + 2P)}{S} + 1$

Number of parameters: $F^2CK$ and $K$ biases
Convolution layer: summary

Let’s assume input is $W_1 \times H_1 \times C$
Conv layer needs 4 hyperparameters:
- Number of filters $K$
- The filter size $F$
- The stride $S$
- The zero padding $P$
This will produce an output of $W_2 \times H_2 \times K$
where:
- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$
Number of parameters: $F^2CK$ and $K$ biases

Common settings:
- $K = (\text{powers of 2}, \text{e.g. 32, 64, 128, 512})$
- $F = 3, S = 1, P = 1$
- $F = 5, S = 1, P = 2$
- $F = 5, S = 2, P = ?$ (whatever fits)
- $F = 1, S = 1, P = 0$
(btw, 1x1 convolution layers make perfect sense)

1x1 CONV with 32 filters

(each filter has size 1x1x64, and performs a 64-dimensional dot product)
(btw, 1x1 convolution layers make perfect sense)

1x1 CONV with 32 filters

(each filter has size 1x1x64, and performs a 64-dimensional dot product)

Grows or shrinks feature channel dimension
Example: CONV layer in PyTorch

Conv layer needs 4 hyperparameters:
- Number of filters $K$
- The filter size $F$
- The stride $S$
- The zero padding $P$
Next Time:
• Pooling
• Convolutional Neural Nets!