Topics:
• Backpropagation through vectors (Cont.)
• Convolutional Neural Networks: Past and Present
• Convolution Layers
Administrative:

- Poll on piazza about local vs. colab setup for HWs
- Assignment due on Sep 19th (with 48hr grace period)
- Will release proposal template today
- 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**:

\[ \frac{\partial y}{\partial x} \in \mathbb{R}^N \quad \left( \frac{\partial y}{\partial x} \right)_n = \frac{\partial y}{\partial x_n} \]

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**:

\[ \frac{\partial y}{\partial x} \in \mathbb{R}^{N \times M} \quad \left( \frac{\partial y}{\partial x} \right)_{n,m} = \frac{\partial y_m}{\partial x_n} \]

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

For each element of $z$, how much does it influence $L$?

"Downstream gradients"

Matrix-vector multiply

Loss $L$ still a scalar!

"Upstream gradient"

For each element of $z$, how much does it influence $L$?

Slide credit: Stanford CS231n Instructors
Gradients loss of wrt a variable have same dims as the original variable

For each element of $z$, how much does it influence $L$?

"Downstream gradients"

"Upstream gradient"

For each element of $z$, how much does it influence $L$?

Slide credit: Stanford CS231n Instructors
Backprop with Vectors

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

4D input x:

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

4D output z:

\[
\begin{bmatrix}
1 \\
0 \\
3 \\
0
\end{bmatrix}
\]
Backprop with Vectors

4D input $x$:

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

$f(x) = \max(0, x) \quad \text{(elementwise)}$

4D output $z$:

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

4D $dL/dz$:

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

\[
\frac{\partial z}{\partial x} = \begin{bmatrix}
\frac{dz_1}{dx_1} & \ldots & \frac{dz_1}{dx_4} \\
\vdots & \ddots & \vdots \\
\frac{dz_4}{dx_1} & \ldots & \frac{dz_4}{dx_4} \\
\end{bmatrix}
\]

What does $\frac{\partial z}{\partial x}$ look like?

Slide credit: Stanford CS231n Instructors
Backprop with Vectors

4D input $x$:

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

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

4D output $z$:

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

Jacobian $\frac{dz}{dx}$:

\[
\begin{bmatrix}
\frac{dz_1}{dx_1} & \cdots & \frac{dz_1}{dx_4} \\
\vdots & \ddots & \vdots \\
\frac{dz_4}{dx_1} & \cdots & \frac{dz_4}{dx_4} \\
\end{bmatrix}
\]

4D $dL/dz$:

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

$\frac{\partial z}{\partial x} = \frac{dL}{dz} \frac{dz}{dx}$

Upstream gradient

Slide credit: Stanford CS231n Instructors
Backprop with Vectors

4D input $x$:

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

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

4D output $z$:

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

4D $dL/dz$:

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

$[dz/dx] \ [dL/dz]$:

\[
\begin{bmatrix}
  1 & 0 & 0 & 0 \\
  0 & 0 & 0 & 0 \\
  0 & 0 & 1 & 0 \\
  0 & 0 & 0 & 0 \\
\end{bmatrix}
\begin{bmatrix}
  4 \\
  -1 \\
  5 \\
  9 \\
\end{bmatrix}
\]

Upstream gradient

Slide credit: Stanford CS231n Instructors
Backprop with Vectors

4D input x:

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

4D output z:

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

4D dL/dx:

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

4D dL/dz:

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

\[f(x) = \max(0,x)\] (elementwise)
Backprop with Vectors

4D input x:

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

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

4D output z:

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

4D dL/dx:

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

4D [dz/dx] [dL/dz]:

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 4 \\
0 & 0 & 0 & 0 & -1 \\
0 & 0 & 1 & 0 & 5 \\
0 & 0 & 0 & 0 & 9 \\
\end{bmatrix}
\]

4D dL/dz:

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

For element-wise ops, Jacobian is **sparse**: off-diagonal entries always zero!

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

*Upstream gradient*

*Slide credit: Stanford CS231n Instructors*
Backprop with Vectors

4D input $x$:

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

4D output $z$:

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

4D $dL/dx$:

$$
\begin{bmatrix}
4 \\
0 \\
5 \\
0 \\
\end{bmatrix}
$$

$dz/dx$ $dL/dz$:

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

Upstream gradient

4D $dL/dz$:

$$
\begin{bmatrix}
4 \\
-1 \\
5 \\
9 \\
\end{bmatrix}
$$

For element-wise ops, Jacobian is **sparse**: off-diagonal entries always zero! Never explicitly form Jacobian -- instead use Hadamard (element-wise) multiplication.
Backprop with Matrices (or Tensors)

$[D_x \times M_x] \ x$

$[D_y \times M_y] \ y$

$[D_z \times M_z] \ z$

Loss $L$ still a scalar!

d$L$/dx always has the same shape as $x$!

Jacobian matrices
Backprop with Matrices (or Tensors)

Loss $L$ still a scalar!

$dL/dx$ always has the same shape as $x$!

For each element of $z$, how much does it influence $L$?

"Downstream gradients"

"Upstream gradient"

Slide credit: Stanford CS231n Instructors
Backprop with Matrices (or Tensors)

Loss L still a scalar!

dL/dx always has the same shape as x!

For each element of y, how much does it influence each element of z?

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

Slide credit: Stanford CS231n Instructors
Backprop with Matrices (or Tensors)

\[
\begin{align*}
[D_x \times M_x] & \quad [D_y \times M_y] \\
\frac{\partial L}{\partial x} & = \frac{\partial z}{\partial x} \frac{\partial L}{\partial x} \\
\frac{\partial L}{\partial y} & = \frac{\partial z}{\partial y} \frac{\partial L}{\partial z}
\end{align*}
\]

"local gradients"

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

Jacobian matrices

Flatten the two matrices -> vector-vector gradients -> jacobian matrices!

"Downstream gradients"

"Upstream gradient"

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

\[
[D_z \times M_z]
\]

Loss \( L \) still a scalar!

\[
\frac{dL}{dx} \text{ always has the same shape as } x!
\]

Slide credit: Stanford CS231n Instructors
Backprop with Matrices

\[ x: [N \times D] \]
\[ \begin{bmatrix} 2 & 1 & -3 \\ -3 & 4 & 2 \end{bmatrix} \]

\[ w: [D \times M] \]
\[ \begin{bmatrix} 3 & 2 & 1 & -1 \\ 2 & 1 & 3 & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix} \]

Matrix Multiply

\[ y_{n,m} = \sum_d x_{n,d}w_{d,m} \]

\[ y: [N \times M] \]
\[ \begin{bmatrix} 13 & 9 & -2 & -6 \\ 5 & 2 & 17 & 1 \end{bmatrix} \]

\[ dL/dy: [N \times M] \]
\[ \begin{bmatrix} 2 & 3 & -3 & 9 \\ -8 & 1 & 4 & 6 \end{bmatrix} \]
Backprop with Matrices

\[ x: [N \times D] \]
\[ \begin{bmatrix}
  2 & 1 & -3 \\
  -3 & 4 & 2 
\end{bmatrix} \]

\[ w: [D \times M] \]
\[ \begin{bmatrix}
  3 & 2 & 1 & -1 \\
  2 & 1 & 3 & 2 \\
  3 & 2 & 1 & -2 
\end{bmatrix} \]

**Matrix Multiply**

\[ y_{n,m} = \sum_d x_{n,d} w_{d,m} \]

\[ y: [N \times M] \]
\[ \begin{bmatrix}
  13 & 9 & -2 & -6 \\
  5 & 2 & 17 & 1 
\end{bmatrix} \]

\[ dL/dy: [N \times M] \]
\[ \begin{bmatrix}
  2 & 3 & -3 & 9 \\
  -8 & 1 & 4 & 6 
\end{bmatrix} \]

**Jacobians:**

\[ dy/dx: [(N \times D) \times (N \times M)] \]

\[ dy/dw: [(D \times M) \times (N \times M)] \]

What does the jacobian matrix look like?

*Slide credit: Stanford CS231n Instructors*
Backprop with Matrices

Matrix Multiply

\[ y_{n,m} = \sum_d x_{n,d} w_{d,m} \]

Jacobians:

\[ \frac{dy}{dx}: [(N \times D) \times (N \times M)] \]
\[ \frac{dy}{dw}: [(D \times M) \times (N \times M)] \]

For a neural net with
\[ N=64, \ D=M=4096 \]
Each Jacobian takes 256 GB of memory!
Must exploit its sparsity!
Backprop with Matrices

\[ x: [N \times D] \]
\[
\begin{bmatrix}
2 & 1 & -3 \\
-3 & 4 & 2
\end{bmatrix}
\]

\[ w: [D \times M] \]
\[
\begin{bmatrix}
3 & 2 & 1 & -1 \\
2 & 1 & 3 & 2 \\
3 & 2 & 1 & -2
\end{bmatrix}
\]

**Q**: What parts of \( y \) are affected by one element of \( x \)?

\( y: [N \times M] \)
\[
\begin{bmatrix}
13 & 9 & -2 & -6 \\
5 & 2 & 17 & 1
\end{bmatrix}
\]

\( dL/dy: [N \times M] \)
\[
\begin{bmatrix}
2 & 3 & -3 & 9 \\
-8 & 1 & 4 & 6
\end{bmatrix}
\]

Recall the branching gradient rule!
Backprop with Matrices

\[ x: [N \times D] \]
\[
\begin{bmatrix}
2 & 1 & -3 \\
-3 & 4 & 2
\end{bmatrix}
\]

\[ w: [D \times M] \]
\[
\begin{bmatrix}
3 & 2 & 1 & -1 \\
2 & 1 & 3 & 2 \\
3 & 2 & 1 & -2
\end{bmatrix}
\]

\[ y: [N \times M] \]
\[
\begin{bmatrix}
13 & 9 & -2 & -6 \\
5 & 2 & 17 & 1
\end{bmatrix}
\]

\[ dL/dy: [N \times M] \]
\[
\begin{bmatrix}
2 & 3 & -3 & 9 \\
-8 & 1 & 4 & 6
\end{bmatrix}
\]

Q: What parts of y are affected by one element of x?
A: \( x_{n,d} \) affects the whole row \( y_n \).

Recall the branching gradient rule!

\[
\frac{\partial L}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} \frac{\partial y_{n,m}}{\partial x_{n,d}}
\]
Backprop with Matrices

**x:** \([N \times D]\)
\[
\begin{bmatrix}
2 & 1 & -3 \\
-3 & 4 & 2
\end{bmatrix}
\]

**w:** \([D \times M]\)
\[
\begin{bmatrix}
3 & 2 & 1 & -1 \\
2 & 1 & 3 & 2 \\
3 & 2 & 1 & -2
\end{bmatrix}
\]

**Matrix Multiply**
\[
y_{n,m} = \sum_{d} x_{n,d} w_{d,m}
\]

**y:** \([N \times M]\)
\[
\begin{bmatrix}
13 & 9 & -2 & -6 \\
5 & 2 & 17 & 1
\end{bmatrix}
\]

**dL/dy:** \([N \times M]\)
\[
\begin{bmatrix}
2 & 3 & -3 & 9 \\
-8 & 1 & 4 & 6
\end{bmatrix}
\]

**Q:** What parts of \(y\) are affected by one element of \(x\)?

**A:** \(x_{n,d}\) affects the whole row \(y_n,:\)

\[
\frac{\partial L}{\partial x_{n,d}} = \sum_{m} \frac{\partial L}{\partial y_{n,m}} \frac{\partial y_{n,m}}{\partial x_{n,d}}
\]

Upstream gradient  local gradient

*Slide credit: Stanford CS231n Instructors*
Backprop with Matrices

\[ x: [N \times D] \]
\[
\begin{bmatrix}
2 & 1 & -3 \\
-3 & 4 & 2 \\
\end{bmatrix}
\]

\[ w: [D \times M] \]
\[
\begin{bmatrix}
3 & 2 & 1 & -1 \\
2 & 1 & 3 & 2 \\
3 & 2 & 1 & -2 \\
\end{bmatrix}
\]

\[ y: [N \times M] \]
\[
\begin{bmatrix}
13 & 9 & -2 & -6 \\
5 & 2 & 17 & 1 \\
\end{bmatrix}
\]

\[ dL/dy: [N \times M] \]
\[
\begin{bmatrix}
2 & 3 & -3 & 9 \\
-8 & 1 & 4 & 6 \\
\end{bmatrix}
\]

**Matrix Multiply**

\[ y_{n,m} = \sum_d x_{n,d} w_{d,m} \]

**Q:** What parts of \( y \) are affected by one element of \( x \)?

**A:** \( x_{n,d} \) affects the whole row \( y_n \).

**Q:** How much does \( x_{n,d} \) affect \( y_{n,m} \)?

\[
\frac{\partial L}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} \frac{\partial y_{n,m}}{\partial x_{n,d}}
\]

*Slide credit: Stanford CS231n Instructors*
Backprop with Matrices

\[ x: [N \times D] \]
\[
\begin{bmatrix}
2 & 1 & -3 \\
-3 & 4 & 2 \\
\end{bmatrix}
\]

\[ w: [D \times M] \]
\[
\begin{bmatrix}
3 & 2 & 1 & -1 \\
2 & 1 & 3 & 2 \\
3 & 2 & 1 & -2 \\
\end{bmatrix}
\]

**Matrix Multiply**

\[ y_{n,m} = \sum_d x_{n,d} w_{d,m} \]

\[ y: [N \times M] \]
\[
\begin{bmatrix}
13 & 9 & -2 & -6 \\
5 & 2 & 17 & 1 \\
\end{bmatrix}
\]

\[ dL/dy: [N \times M] \]
\[
\begin{bmatrix}
2 & 3 & -3 & 9 \\
-8 & 1 & 4 & 6 \\
\end{bmatrix}
\]

**Q:** What parts of \( y \) are affected by one element of \( x \)?

**A:** \( x_{n,d} \) affects the whole row \( y_n \).

**Q:** How much does \( x_{n,d} \) affect \( y_{n,m} \)?

\[
\frac{\partial L}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} \frac{\partial y_{n,m}}{\partial x_{n,d}}
\]

\[
y_{n,m} = \sum_{i=1}^{D} x_{n,i} w_{i,m}
\]

\[
\frac{\partial y_{n,m}}{\partial x_{n,d}} = w_{d,m}
\]
Backprop with Matrices

\[ x: [N \times D] \]
\[
\begin{bmatrix}
2 & 1 & -3 \\
-3 & 4 & 2
\end{bmatrix}
\]

\[ w: [D \times M] \]
\[
\begin{bmatrix}
3 & 2 & 1 & -1 \\
2 & 1 & 3 & 2 \\
3 & 2 & 1 & -2
\end{bmatrix}
\]

\[ y: [N \times M] \]
\[
\begin{bmatrix}
13 & 9 & -2 & -6 \\
5 & 2 & 17 & 1
\end{bmatrix}
\]

\[ dL/dy: [N \times M] \]
\[
\begin{bmatrix}
2 & 3 & -3 & 9 \\
-8 & 1 & 4 & 6
\end{bmatrix}
\]

**Q:** What parts of \( y \) are affected by one element of \( x \)?

**A:** \( x_{n,d} \) affects the whole row \( y_n \).

\[
\frac{\partial L}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} \frac{\partial y_{n,m}}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} w_{d,m} = \frac{\partial L}{\partial y_n} w_d^T
\]

**Q:** How much does \( x_{n,d} \) affect \( y_{n,m} \)?

**A:** \( w_{d,m} \)

Just a dot product!
Backprop with Matrices

\[ x: [N \times D] \]
\[
\begin{bmatrix}
2 & 1 & -3 \\
-3 & 4 & 2
\end{bmatrix}
\]

\[ w: [D \times M] \]
\[
\begin{bmatrix}
3 & 2 & 1 & -1 \\
2 & 1 & 3 & 2 \\
3 & 2 & 1 & -2
\end{bmatrix}
\]

\[ y: [N \times M] \]
\[
\begin{bmatrix}
13 & 9 & -2 & -6 \\
5 & 2 & 17 & 1
\end{bmatrix}
\]

\[ dL/dy: [N \times M] \]
\[
\begin{bmatrix}
2 & 3 & -3 & 9 \\
-8 & 1 & 4 & 6
\end{bmatrix}
\]

Q: What parts of \( y \) are affected by one element of \( x \)?
A: \( x_{n,d} \) affects the whole row \( y_n \).

Q: How much does \( x_{n,d} \) affect \( y_{n,m} \)?
A: \( w_{d,m} \)

\[
\frac{\partial L}{\partial x} = \left( \frac{\partial L}{\partial y} \right) w^T
\]

\[
\frac{\partial L}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} \frac{\partial y_{n,m}}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} w_{d,m} = \frac{\partial L}{\partial y_n} w_d^T
\]

Just a matrix multiplication
No jacobian matrix needed!
Backprop with Matrices

\[
x: \begin{bmatrix} 2 & 1 & -3 \\ -3 & 4 & 2 \end{bmatrix}
\]

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

\[
y: \begin{bmatrix} 13 & 9 & -2 & -6 \\ 5 & 2 & 17 & 1 \end{bmatrix}
\]

\[
dL/dy: \begin{bmatrix} 2 & 3 & -3 & 9 \\ -8 & 1 & 4 & 6 \end{bmatrix}
\]

\[
\frac{\partial L}{\partial x} = \left( \frac{\partial L}{\partial y} \right) w^T
\]

\[
\frac{\partial L}{\partial w} = x^T \left( \frac{\partial L}{\partial y} \right)
\]

By similar logic:

For a neural net layer with \( N=64, D=M=4096 \)
The largest matrix \( W \) takes up to 0.13 GB memory.
Summary (Lecture 5 – here):

• Neural networks, activation functions
• Neurons as biological inspirations to DNNs
• Vector Calculus
• Backpropagation through vectors / matrices
Next: Convolutional Neural Networks

![Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1](Image)

Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

Slide credit: Stanford CS231n Instructors
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 cadmium sulfide 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

Slide credit: Stanford CS231n Instructors
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
A bit of history...

Reinvigorated research in Deep Learning

[Hinton and Salakhutdinov 2006]
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**  

Slides credit: Stanford CS231n Instructors
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

Visual stimulus

Retinal ganglion cell receptive fields

LGN and V1 simple cells

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

Slide credit: Stanford CS231n Instructors
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"

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.
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]

Slide credit: Stanford CS231n Instructors
Fast-forward to today: ConvNets are everywhere

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


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:** temporal convolution

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

Generalized convolution: **graph convolution**

Kipf et al., 2017
No errors

A white teddy bear sitting in the grass

Minor errors

A man in a baseball uniform throwing a ball

Somewhat related

A woman is holding a cat in her hand

Image-to-text

A woman standing on a beach holding a surfboard

[Vinyals et al., 2015]
[Karpathy and Fei-Fei, 2015]
[Radford, 2021]

All images are CC0 Public domain:

Captions generated by Justin Johnson using 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.

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?

Locality of Features
Convolution: A 1D Visual Example

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

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

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

*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!

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)
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

input

1

3072

\[ Wx \]

10 x 3072 weights

activation

1

10

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

32  height
32  width
3    depth
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

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

Filters always extend the full depth of the input volume
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
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
32
3
3
28
28
1
28
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!
**Preview:** ConvNet is a sequence of Convolution Layers, interspersed with activation functions.

![ConvNet Diagram]

- 3x32x3 input
- 32x5x5 CONV, ReLU
- e.g. 6 5x5x3 filters
- 28x6 output
**Preview:** ConvNet is a sequence of Convolution Layers, interspersed with activation functions.

1. **CONV, ReLU** e.g. 6 filters
   - $5 \times 5 \times 3$

2. **CONV, ReLU** e.g. 10 filters
   - $5 \times 5 \times 6$

3. **CONV, ReLU**
   - $5 \times 5 \times 6$

...
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:

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

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

32

32

32

3

28

28

1
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 \textit{stride}.

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

7x7 input (spatially)
assume 3x3 filter 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 **stride 1**
**pad with 1 pixel** border => what is the output?

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

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

7x7 output!

N = input dimension
P = padding size
F = filter size
Output size = \((N - F + 2P) / \text{stride} + 1\)
= \((7 - 3 + 2 \times 1) / 1 + 1 = 7\)
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 \textbf{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:
\[(N - F) / \text{stride} + 1\]

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

With padding of 1 x 1:
- stride 3 => \((7 - 3 + 2)/3 + 1 = 3\)
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: 32x32x3
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:
$(32 + 2 \times 2 - 5) / 1 + 1 = 32$ 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: \( \textcolor{red}{32x32x3} \)
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?
each filter has \( \textcolor{blue}{5*5*3} + 1 = 76 \) params (+1 for bias)
=> \( \textcolor{green}{76*10} = 760 \)
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:
- $F = 3, S = 1, P = 1$
- $F = 5, S = 1, P = 2$
- $F = 5, S = 2, P = ?$ (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

Conv2d

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size \( (N, C_{in}, H, W) \) and output \( (N, C_{out}, H_{out}, W_{out}) \) can be precisely described as:

\[
out(N_{i}, C_{out}) = bias(C_{out}) + \sum_{k=0}^{C_{in}-1} \text{weight}(C_{out}, k) \ast \text{input}(N_{i}, k)
\]

where \( \ast \) is the valid 2D cross-correlation operator, \( N \) is a batch size, \( C \) denotes a number of channels, \( H \) is a height of input planes in pixels, and \( W \) is width in pixels.

- \text{stride} controls the stride for the cross-correlation, a single number or a tuple.
- \text{padding} controls the amount of implicit zero-padding on both sides for \text{padding} number of points for each dimension.
- \text{dilation} controls the spacing between the kernel points; also known as the à trous algorithm. It is harder to describe, but this link has a nice visualization of what \text{dilation} does.
- \text{groups} controls the connections between inputs and outputs. \text{in_channels} and \text{out_channels} must both be divisible by \text{groups}. For example,
  - At \text{groups}=1, all inputs are convolved to all outputs.
  - At \text{groups}=2, the operation becomes equivalent to having two conv layers side by side, each seeing half the input channels, and producing half the output channels, and both subsequently concatenated.
  - At \text{groups} \text{ in_channels}, each input channel is convolved with its own set of filters, of size: \( \left[ \frac{C_{out}}{C_{in}} \right] \).

The parameters \text{kernel_size}, \text{stride}, \text{padding}, \text{dilation} can either be:

- a single int - in which case the same value is used for the height and width dimension
- a tuple of two ints - in which case, the first int is used for the height dimension, and the second int for the width dimension

PyTorch is licensed under BSD 3-clause.
Next Time:

• Pooling

• Convolutional Neural Nets!