Lecture 5: Convolutional Neural Networks
Who am I?

Aditya Kusupati
- 4th year PhD student at UW CSE
- I work with Ali Farhadi and Sham Kakade

My Research:
- I develop fundamental ML algorithms that are amenable to practical deployment both at edge and web-scale.
- More recently, I have been working towards rethinking search for better indexing of the world.

Past Experiences:
- Student Researcher at Google Research
- AI Resident at Microsoft Research
Administrative

**Assignment 1** due **Friday April 14, 11:59pm**
- Before submitting your work, please be sure to read the instructions on the course website carefully (Assignments Tab), and follow the steps mentioned in “Submitting your work”. This will ensure Gradescope grades your work correctly.

**Assignment 2** will also be released **April 14th**
Administrative

Project proposal due **Monday Apr 24, 11:59pm**

This Friday’s discussion section will discuss how to design a project – **Sarah Pratt**

Meet **Ranjay, Sarah or me** about projects initially

**Benlin or Sharan** about assignments
Last time: Neural Networks

Linear score function:

2-layer Neural Network

\[ f = W x \]

\[ f = W_2 \max(0, W_1 x) \]
The diagram illustrates the concept of "local gradient" and "upstream gradient" in the context of a function $f(x, y)$.

- The input variables are $x$ and $y$.
- The output variable is $z$.
- The local gradient is represented by $\frac{\partial z}{\partial x}$ and $\frac{\partial z}{\partial y}$.
- The upstream gradient is represented by $\frac{\partial L}{\partial z}$.

The diagram emphasizes the flow of information and the relationship between the inputs, outputs, and gradients.
"local gradient"
The diagram illustrates the concept of "local gradient" and its relationships with "upstream gradients" and "downstream gradients".

- The "local gradient" is represented by the equation $\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x}$.
- "Upstream gradient" is indicated by $\frac{\partial L}{\partial z}$ and "downstream gradients" are shown as $\frac{\partial L}{\partial x}$ and $\frac{\partial L}{\partial y}$.
\[ \frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x} \]

“Downstream gradients”

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

“Upstream gradient”

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

“Local gradient”
So far: backprop with scalars

What about vector-valued functions?
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?
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 then how much will \( y \) change?
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 then 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 then how much will each element of \( y \) change?
Backprop with Vectors

Loss L still a scalar!
Backprop with Vectors

$$D_x \quad x$$

$$D_y \quad y$$

$$D_z \quad z$$

Loss $L$ still a scalar!
Backprop with Vectors

Loss $L$ still a scalar!

“Upstream gradient”

$\frac{\partial L}{\partial z}$

$D_z$

$\mathbf{z}$

$D_x$

$\mathbf{x}$

$D_y$

$\mathbf{y}$

$f$

$L$ is a scalar function of $z$, and $z$ is a function of $x$ and $y$. The gradient $\frac{\partial L}{\partial z}$ is the “upstream gradient” that flows back through the function $f$. The gradients $D_x$, $D_y$, and $D_z$ represent the partial derivatives of the function $f$ with respect to $x$, $y$, and $z$, respectively.
Backprop with Vectors

Loss $L$ still a scalar!

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

“Upstream gradient”

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

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

"Upstream gradient"

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

Loss L still a scalar!

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

Loss $L$ still a scalar!

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

“Upstream gradient”

“Downstream gradients”

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

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

\[
\frac{\partial L}{\partial z} = \frac{\partial z}{\partial z}
\]
Backprop with Vectors

\[
\begin{align*}
\frac{\partial L}{\partial x} &= \frac{\partial z}{\partial x} \frac{\partial L}{\partial z} \\
\frac{\partial L}{\partial y} &= \frac{\partial z}{\partial y} \frac{\partial L}{\partial z}
\end{align*}
\]

“Downstream gradients”

\[
\begin{bmatrix}
\frac{\partial L}{\partial x} \\
\frac{\partial L}{\partial y}
\end{bmatrix}
\]

“local gradients”

\[
[D_x \times D_z]
\]

\[
\begin{bmatrix}
\frac{\partial z}{\partial x} \\
\frac{\partial z}{\partial y}
\end{bmatrix}
\]

Jacobian matrices

“Upstream gradient”

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

Loss L still a scalar!

Ranjay Krishna, Aditya Kusupati
Backprop with Vectors

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

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

\[
[D_x \times D_z]
\]

Jacobian matrices

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

Loss \( L \) still a scalar!

Matrix-vector multiply
Gradients of variables wrt loss have same dims as the original variable

Loss $L$ still a scalar!

“Upstream gradient”
For each element of $z$, how much does it influence $L$?

Gradients of variables wrt loss have same dims as the original variable.
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})
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 $dL/dz$:

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

Upstream gradient
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 dz/dx

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

4D dL/dz:

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

Upstream gradient
### Backprop with Vectors

**4D input x:**

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

\[
f(x) = \max(0, x)
\]

*elementwise*

**4D output z:**

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

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

**4D \( \frac{dL}{dz} \):**

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

**Upstream gradient**

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

4D input $x$:

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

4D output $z$:

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

4D $dL/dz$:

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

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

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \frac{dz}{dx} \begin{bmatrix} 4 \\ 0 \\ 0 \\ 0 \end{bmatrix} \frac{dL}{dz}$$

4D $dL/dx$:

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

Upstream gradient
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:

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

4D dL/dz:

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

Jacobian is sparse: off-diagonal entries always zero! Never explicitly form Jacobian -- instead use implicit multiplication.

Upstream gradient
Backprop with Vectors

4D input $x$: $\begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix}$

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

4D output $z$: $\begin{bmatrix} 1 \\ 0 \\ 3 \\ 0 \end{bmatrix}$

$4D \frac{dL}{dz}$:
$\begin{bmatrix} 4 \\ -1 \\ 5 \\ 0 \end{bmatrix}$

$4D \frac{dz}{dx}$:
$\begin{bmatrix} 4 \\ 0 \\ 5 \\ 0 \end{bmatrix}$

Jacobian is sparse: off-diagonal entries always zero! Never explicitly form Jacobian -- instead use implicit multiplication

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

\[
[D_x \times M_x] \quad x
\]

Matrix-vector multiply

\[
[D_y \times M_y] \quad y
\]

Jacobian matrices

\[
[D_z \times M_z] \quad z
\]

Loss L still a scalar!

dL/dx always has the same shape as x!
Backprop with Matrices (or Tensors)

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

"Downstream gradients"

Matrix-vector multiply

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

"Upstream gradient"

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

\[ \text{Loss } L \text{ still a scalar!} \]

\[ \frac{dL}{dx} \text{ always has the same shape as } x! \]
Backprop with Matrices (or Tensors)

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

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

Loss $L$ still a scalar!

d$L/dx$ always has the same shape as $x$!
Backprop with Matrices (or Tensors)

**Local Gradients**

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

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

**Downstream Gradients**

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

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

**Upstream Gradient**

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

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

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

**Matrix-vector multiply**

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

Loss \( L \) still a scalar!

\[dL/dx\] always has the same shape as \( x \)!
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

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

Also see derivation by Prof. Justin Johnson:
https://courses.cs.washington.edu/courses/cse493g1/23sp/resources/linear-backprop.pdf
Backprop with Matrices

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

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

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

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

**Matrix Multiply**

\[ y_{n,m} = \sum_{d} x_{n,d} \mathbf{w}_{d,m} \]

**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 we may have

- \( N=64 \), \( D=M=4096 \)
- Each Jacobian takes \( \sim 256 \) GB of memory! Must work with them implicitly!
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 \( x \)?
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 \( y_n \).

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

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

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

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

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

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 \( y_{n,m} \)?
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

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

$y$: $[N \times M]$

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

d$L$/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:** $w_{d,m}$

\[
\frac{\partial L}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} \frac{\partial y_{n,m}}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} w_{d,m}
\]
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 $y_{n,m}$?
A: $w_{d,m}$

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

$\frac{\partial L}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} \frac{\partial y_{n,m}}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} w_{d,m}$
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

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

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

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

By similar logic:

These formulas are easy to remember: they are the only way to make shapes match up!
Wrapping up: Neural Networks

Linear score function:

$$f = Wx$$

$$f = W_2 \max(0, W_1 x)$$

2-layer Neural Network
Next: Convolutional Neural Networks

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

The function $f(x)$ is defined as:

$$f(x) = \begin{cases} 
1 & \text{if } w \cdot x + b > 0 \\
0 & \text{otherwise}
\end{cases}$$

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

This image by Rocky Acosta is licensed under CC-BY 3.0
A bit of history...

Widrow and Hoff, ~1960: Adaline/Madaline
Rumelhart et al., 1986: First time back-propagation became popular

\[
\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial w_{ji}}
\]
A bit of history...

[Hinton and Salakhutdinov 2006]

Reinvigorated research in Deep Learning
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...
A bit of history

Topographical mapping in the cortex: nearby cells in cortex represent nearby regions in the visual field

Retinotopy images courtesy of Jesse Gomez in the Stanford Vision & Perception Neuroscience Lab.
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
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]

LeNet-5
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

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

[Farabet et al., 2012]

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

Fast-forward to today: ConvNets are everywhere

self-driving cars

Photo by Lane McIntosh. Copyright CS231n 2017.

NVIDIA Tesla line

Note that for embedded systems a typical setup would involve NVIDIA Tegras, with integrated GPU and ARM-based CPU cores.
Fast-forward to today: ConvNets are everywhere

Activations of *inception-v3 architecture* [Szegedy et al. 2015] to image of Emma McIntosh, used with permission. Figure and architecture not from Taigman et al. 2014.

Illustration by Lane McIntosh, photos of Katie Cumnock used with permission.

Figures copyright Simonyan et al., 2014. Reproduced with permission.
Fast-forward to today: ConvNets are everywhere

[Toshev, Szegedy 2014]

[Guo et al. 2014]

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

Fast-forward to today: ConvNets are everywhere

[Levy et al. 2016]

From left to right: public domain by NASA, usage permitted by ESA/Hubble, public domain by NASA, and public domain.

[Dieleman et al. 2014]

[Serhanet et al. 2011]

[Serhanet et al. 2011]

Photos by Lane McIntosh. Copyright CS231n 2017.
Whale recognition, Kaggle Challenge

Mnih and Hinton, 2010
Image Captioning

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

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

No errors
Minor errors
Somewhat related

All images are CC0 Public domain:

Captions generated by Justin Johnson using Neuraltalk2
Convolutional Neural Networks
Recap: Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

\[ Wx \]

weights

activation
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

- **Input:** 3072
- **Weights:** 10 x 3072
- **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

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)

\[
wx^T + b
\]
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
Consider a second, green filter.
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.

Example: 6 5x5x3 filters.
**Preview:** ConvNet is a sequence of Convolution Layers, interspersed with activation functions.
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
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
A closer look at spatial dimensions:

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

7x7 input (spatially)
assume 3x3 filter

=> 5x5 output
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: \( (N - F) / \text{stride} + 1 \)

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

(recall:)
(N - F) / stride + 1
In practice: Common to zero pad the border

\[ (N + 2P - F) / \text{stride} + 1 \]

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

7x7 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
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.

32

32

32

32

CONV, ReLU
 e.g. 6
5x5x3 filters

28

28

28

28

CONV, ReLU
 e.g. 10
5x5x6 filters

24

24

24

24

CONV, ReLU

...
Examples time:

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

Output volume size: ?
Examples time:

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size:
\[(32 + 2 \times 2 - 5) / 1 + 1 = 32\] spatially, so
**32x32x10**
Examples time:

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

Number of parameters in this layer?
Examples time:

Input volume: \(32 \times 32 \times 3\)

10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

Each filter has \(5 \times 5 \times 3 + 1 = 76\) params (+1 for bias)

\(\Rightarrow 76 \times 10 = 760\)
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:

- $F = 3, S = 1, P = 1$
- $F = 5, S = 1, P = 2$
- $F = 5, S = 2, P = ?$ (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)
Conv layer needs 4 hyperparameters:
- Number of filters $K$
- The filter size $F$
- The stride $S$
- The zero padding $P$

Example: CONV layer in PyTorch

```python
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:

$$
out(N, C_{out}) = bias(C_{out}) + \sum_{k=0}^{C_{in}-1} weight(C_{out}, k) * input(N, k)
$$

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.

- `stride` controls the stride for the cross-correlation, a single number or a tuple.
- `padding` controls the amount of implicit zero-paddings on both sides for padding number of points for each
dimension.
- `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 dilation does.
- `groups` controls the connections between inputs and outputs. `in_channels` and `out_channels` must both be
divisible by `groups`. For example,
  - $\text{At } groups=1$, all inputs are convolved to all outputs.
  - $\text{At } groups=2$, 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.
  - $\text{At } groups=\text{in_channels}$, each input channel is convolved with its
    own set of filters, of size: $\left[ \frac{C_{in}}{C_{out}} \right]$

The parameters `kernel_size`, `stride`, `padding`, `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.
Example: CONV layer in Keras

Conv layer needs 4 hyperparameters:
- Number of filters \( K \)
- The filter size \( F \)
- The stride \( S \)
- The zero padding \( P \)
The brain/neuron view of CONV Layer

32x32x3 image
5x5x3 filter

1 number:
the result of taking a dot product between the filter and this part of the image (i.e. 5*5*3 = 75-dimensional dot product)
The brain/neuron view of CONV Layer

32x32x3 image
5x5x3 filter

1 number: the result of taking a dot product between the filter and this part of the image (i.e. 5*5*3 = 75-dimensional dot product)
Receptive field

An activation map is a 28x28 sheet of neuron outputs:
1. Each is connected to a small region in the input
2. All of them share parameters

“5x5 filter” -> “5x5 receptive field for each neuron”
The brain/neuron view of CONV Layer

E.g. with 5 filters, CONV layer consists of neurons arranged in a 3D grid (28x28x5)

There will be 5 different neurons all looking at the same region in the input volume
Reminder: Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

Each neuron looks at the full input volume

\[ Wx \]
10 x 3072 weights

1 number: the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)
two more layers to go: POOL/FC
Pooling layer
- makes the representations smaller and more manageable
- operates over each activation map independently:

![Diagram showing how pooling layer reduces the size of activation maps.](image-url)
MAX POOLING

Single depth slice

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

max pool with 2x2 filters and stride 2

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
Pooling layer: summary

Let’s assume input is $W_1 \times H_1 \times C$
Conv layer needs 2 hyperparameters:
- The spatial extent $F$
- The stride $S$

This will produce an output of $W_2 \times H_2 \times C$ where:
- $W_2 = (W_1 - F)/S + 1$
- $H_2 = (H_1 - F)/S + 1$

Number of parameters: 0
Fully Connected Layer (FC layer)
- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks
ConvNetJS CIFAR-10 demo

This demo trains a Convolutional Neural Network on the CIFAR-10 dataset in your browser, with nothing but Javascript. The state of the art on this dataset is about 90% accuracy and human performance is at about 94% (not perfect as the dataset can be a bit ambiguous). I used this python script to parse the original files (python version) into batches of images that can be easily loaded into page DOM with img tags.

This dataset is more difficult and it takes longer to train a network. Data augmentation includes random flipping and random image shifts by up to 2px horizontally and vertically.

By default, in this demo we’re using Adadelta which is one of per-parameter adaptive step size methods, so we don’t have to worry about changing learning rates or momentum over time. However, I still included the text fields for changing these if you’d like to play around with SGD+Momentum trainer.

Report questions/bugs/suggestions to @karpathy.

http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html
Summary

- ConvNets stack CONV,POOL,FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Historically architectures looked like
  \[[\text{CONV-RELU}]*N-\text{POOL}?]^{*M-\text{(FC-RELU)}}*K,\text{SOFTMAX}\]
  where N is usually up to $\sim 5$, M is large, $0 \leq K \leq 2$.
  - but recent advances such as ResNet/GoogLeNet have challenged this paradigm