CSCI 566: Deep Learning and Its Applications

Jesse Thomason

Lecture 3: Training NNs; CNNs

CSCI 566 Resources

- **Website:** https://csci566-spring2023.github.io/
- **Piazza:** https://piazza.com/usc/spring2023/csci566/info
- **GDrive:** https://drive.google.com/drive/folders/11YcbSZcJUdRSbrNBWiOs_HM2VkxdNKWS?usp=share_link
  - Requires USC SSO; please don't request access from your gmail account
- **Zoom:** https://usc.zoom.us/j/95538924150?pwd=TTRBM283VCt4WDFkN1hLMmtrKzdGUT09

CSCI 566 Resources

- **Piazza:** https://piazza.com/usc/spring2023/csci566/info
- Please use Piazza for any course related communication
- Course links:
  - Piazza → Resources → Course Information
- If you're new to the course, please fill Quiz 0 in the Resources tab of Piazza
- Personal matters and emergency communication only:
  - csci566.spring2023@gmail.com
  - Any email that should have been a Piazza post will be ignored without acknowledgement

Course Office Hours

Jesse Thomason

- Instructor Office Hours
  - Mon 10am-11am
  - SAL 244
- NOT for homework related questions.

- TA Office Hours
- HW questions; project questions & feedback
- Times available on the course website
- Updates/changes to times announced on Piazza

Deqing Fu

Gautam Salhotra

Tejas Srinivasan

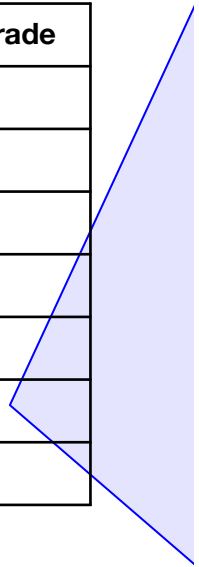
Bingjie Tang

Shihan Lu

Ayush Jain & Jesse Zhang

Course Deliverables

Deliverable	Points of the total grade
Pop-up Quizzes	5
Assignment #1	10
Assignment #2	10
Paper Roleplaying Breakout	10
Midterm	20
Course Project	45
TOTAL	**100**



Course Project Team Registration

- Project 4-person teams are due today; Feb 3rd, 11:59pm
  - There is a team registration form online
  - Piazza → Resources → Course Project
  - Only **ONE** person from each team should submit this form
  - Each person in the room should be on *exactly one team*
- For the research project:
  - You may *not* work with students outside this course.
  - You may *not* share code with other teams in the course.
  - You *must* enumerate individual contributions of your four team members in reports.

Course Project

- Select from provided project ideas
  - https://docs.google.com/document/d/1bQusP0uS8KoVk7ZmxpGs209gBD3N9p-aSzMTTM63k1k/
    - This is a *living document*; more project ideas might trickle in since we have new TAs. You can pivot your project until the proposal; the registration tentative title is to help us pick a guiding TA for your team
- *Or* design your own project, using the project ideas doc as an outline for what you should include in your proposal slide deck

Map to the Midterm

January 2023

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

www.a-printable-calendar.com

February 2023

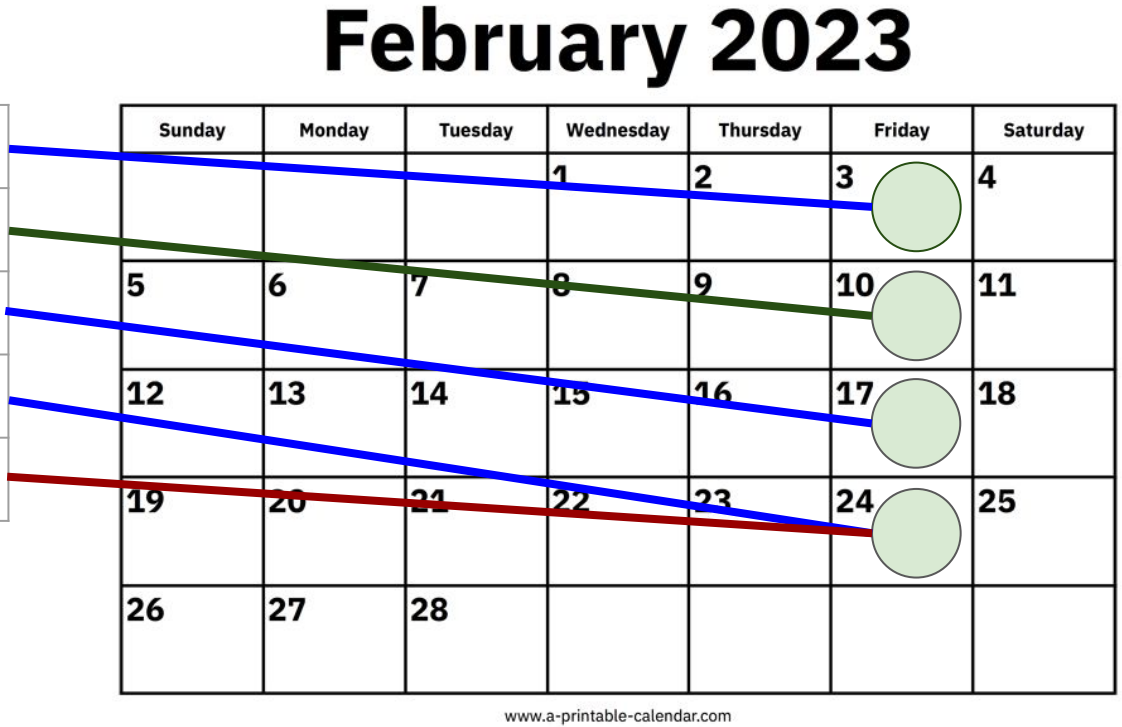
Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28				

www.a-printable-calendar.com

Module 1: Neural Network Basics

[Feb 3] A Look at February Deliverables

Project Teams Formed	Feb 3
Assignment 1 Out	Feb 10
Project Proposal Due	Feb 17
Assignment 1 Due	Feb 24
Midterm Exam	Feb 24



Thinking Ahead: Course Project Proposal [Feb 17]

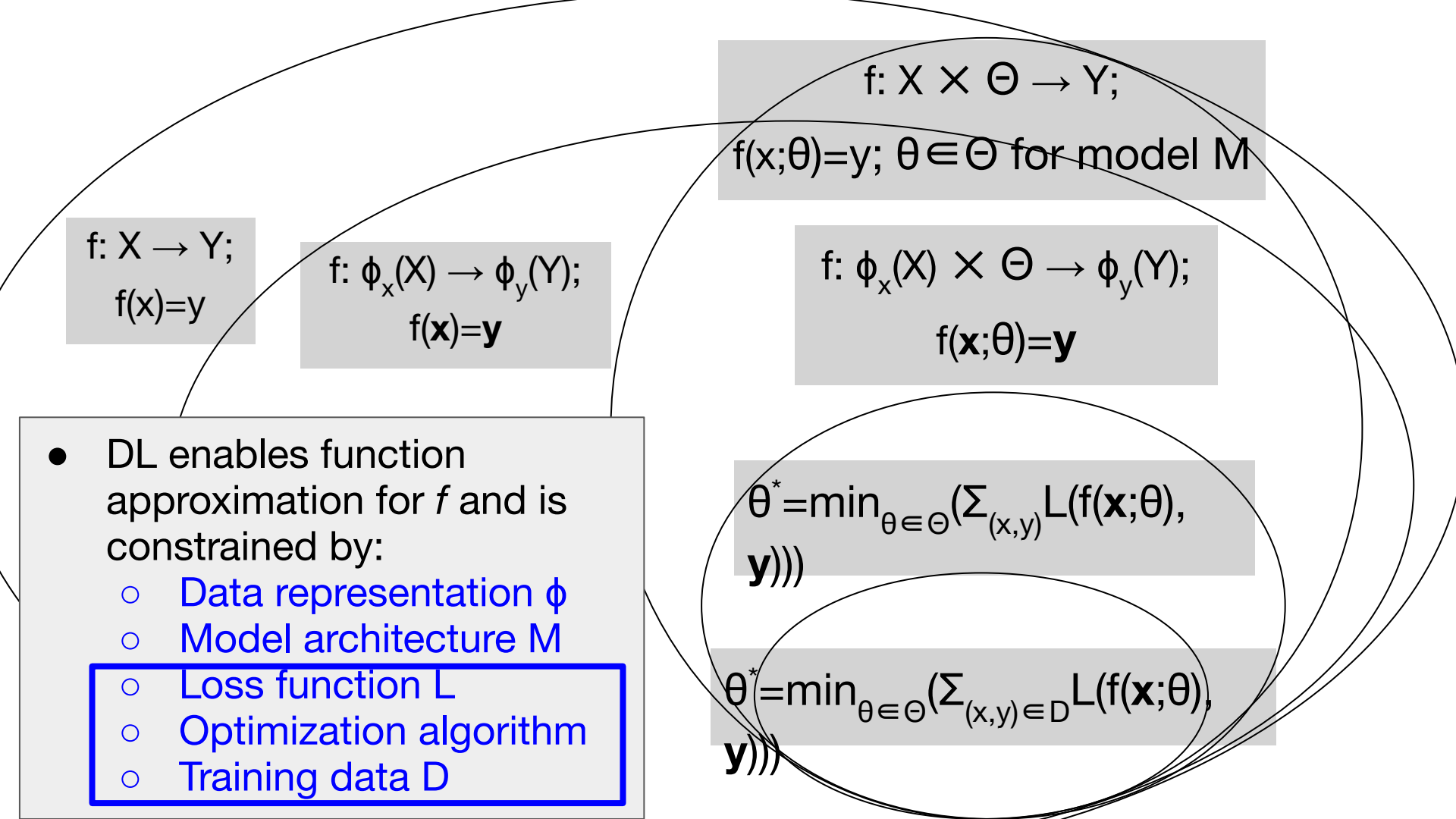
- **Proposal:** 5-6 slides on problem (scope & definition) + "today" (status of literature) + challenges + directions to innovate.
  - Look at the project ideas doc populated by the TAs to get a sense of what material your proposal should cover for problem definition and background
  - Include what you hope to be your *key insight or contribution*; what will make your approach unique?
- We will share a link to collect the slides.
- You will condense these longer, more detailed slides into a single lightning pitch slide for the Project Pitch assignment.

Course Compute

- We have both CARC and Google Cloud Platform credits for the course
  - Next week in class, the TAs will lead a tutorial session during our lecture slot on using these cloud platforms
  - Additionally, the TAs will lead a tutorial session on the PyTorch deep learning framework
- TA tutorial sessions will be valuable both for your course projects and for preparing to do the Coding Assignments

Overview of Today's Plan

- ~~Course organization and deliverables~~
  - Any questions before we move on?
- Training Neural Networks
- Convolutional Neural Networks


f: X → Y;
f(x)=y
f: $\phi_x(X) \rightarrow \phi_y(Y)$;
f(**x**)=**y**
f: X × Θ → Y;
f(x;θ)=y; θ∈Θ for model M
f: $\phi_x(X) \times \Theta \rightarrow \phi_y(Y)$;
f(**x**;θ)=**y**
$\theta^* = \min_{\theta \in \Theta}(\Sigma_{(x,y)} L(f(\mathbf{x};\theta), \mathbf{y}))$
$\theta^* = \min_{\theta \in \Theta}(\Sigma_{(x,y) \in D} L(f(\mathbf{x};\theta), \mathbf{y}))$
• DL enables function approximation for *f* and is constrained by:
○ Data representation ϕ
○ Model architecture M
○ Loss function L
○ Optimization algorithm
○ Training data D

Where Does Data *D* Come From?

- We've been assuming access to a dataset of (X, Y) to calculate the difference between y_i and $f(x_i;\theta)$ for $(x_i,y_i) \in (X, Y)$
- **Supervised learning:**
  - For all training data x_i we know the correct label y_i
- **Unsupervised learning:**
  - For no training data x_i do we have an associated label
- **Weakly-supervised learning:**
  - For a (usually small) subset of x_i we have label y_i
- **Reinforcement learning:**
  - In sequential problems, we don't have labels for individual predictions but get a *reward* for the final sequence

How Do We Measure the Quality of $f(\boldsymbol{x};\theta)$?

- How do we measure the *fit* of candidate parameters?
- Want to formulate an **optimization problem** to minimize the distance $\{f^*(\boldsymbol{x}) - f(\boldsymbol{x};\theta)\}$ across all $\boldsymbol{x}_i \in X$ in a dataset of (X, Y) pairs
- We call the function we want to minimize, subject to the available training data, the **loss function**
- What are some possible ways to measure such a distance penalty? We want “good” θ to have less loss than “bad” θ

How Do We Measure the Quality of $f(\boldsymbol{x};\theta)$?

- We could add a quadratic penalty

$$Loss = \frac{1}{N}\sum_{i=1}^{N} \| \ y_i - f(x_i; W, b)\|_2^2$$

- This loss function is called **L_2 Loss**
- What could go wrong?
  - The model doesn't have a strong incentive to get *f* "exactly" right; for terms less than 1 the square makes them less severe
  - Not getting *f* exactly right isn't such a bad thing!

How Do We Find Good Parameters θ for $f(\mathbf{x};\theta)$?

- $f(\mathbf{x};\theta)=W\mathbf{x}+b$; $\theta=\{W,b\}$
- $\theta^{*}=min_{\theta \in \Theta}(\Sigma_{(x,y) \in D} L(f(x;\theta), y)))$
  - For $D=\{(x_1,y_1), (x_2,y_2), \ldots, (x_n,y_n)\}$ our *training dataset*
- Three basic classes of approach:
  - Analytical solution
    - Find where $^{d}/_{d\theta}(\Sigma_{(x,y) \in D} L(f(x;\theta), y))) = 0$
  - Heuristic search
    - Initialize random candidate θ's and check L value
  - Numerical approach
    - Estimate better θ's given current ones

How Do We Find Good Parameters θ for $f(\mathbf{x};\theta)$?

- $f(\mathbf{x};\theta)=W\mathbf{x}+b$; $\theta=\{W,b\}$
- $\theta^{*}=min_{\theta \in \Theta}(\Sigma_{(x,y) \in D} L(f(x;\theta), y)))$
  - For $D=\{(x_1,y_1), (x_2,y_2), \ldots, (x_n,y_n)\}$ our *training dataset*
- Numerical approach
  - Estimate better θ's given current ones
- Best of both worlds: cheap (heuristic) + informed (analytical)
  - Finding where $^{d}/_{d\theta}(\Sigma_{(x,y) \in D} L(f(x;\theta), y))) = 0$ is hard
  - Finding good θ through random/population search is hard
- Calculating $^{d}/_{d\theta i}(\Sigma_{(x,y) \in D} L(f(x;\theta), y)))$ for dimension θ_i is easy…

Gradient Descent

-0.1	0.2	0.15	0.05
0.25	0.05	0.2	-0.1
0.25	0.15	-0.15	0.4

W

-0.1+ 0.001	0.2	0.15	0.05
0.25	0.05	0.2	-0.1
0.25	0.15	-0.15	0.4

W+h

0.5	?	?	?
?	?	?	?
?	?	?	?

gradient (dW)

L(W) = 1.6688

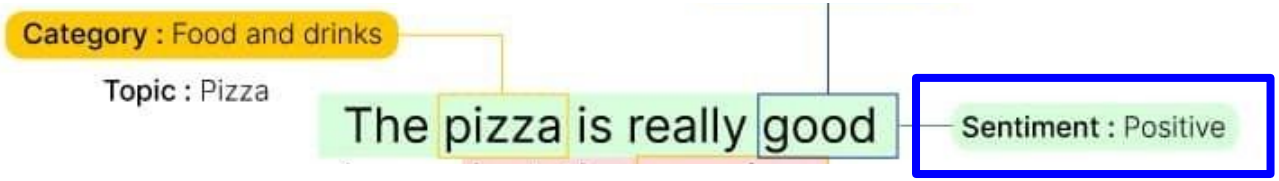
L(W+h) = 1.6693

dW_1=(1.6693 - 1.6688) / 0.001

= 0.5

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

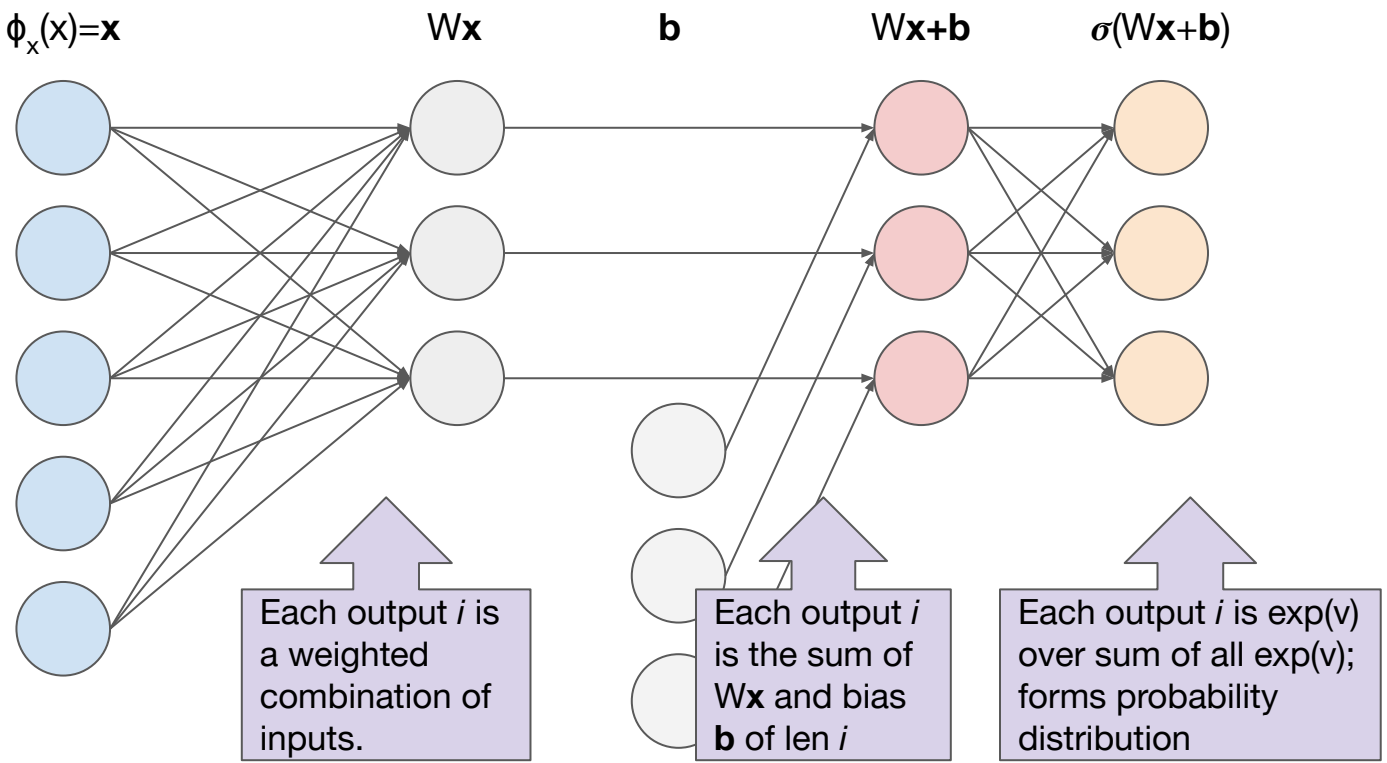
Formulating a Problem for Machine Learning



- Input space *X*?
  - Review *tokens T*
  - $X=T^N$ for max length *N*
- Output space *Y*?
  - Possible sentiments *S*
  - *Y*=*S*
- Linear function: $f(\boldsymbol{x},\theta)=W\boldsymbol{x}+b$
- What loss function could we use for this *n*-ary classification problem?
  - Cross entropy
- Non-linear transformation:

$$Loss = -\frac{1}{N}\sum_{i=1}^{N} \hat{y_i}\, log(P(y_i = j\,|x_i)) \qquad P(y_i = j\,|x_i) = \frac{e^{f(x_i;W,b)}}{\sum_{j=1}^{C} e^{f(x_j;W,b)}}$$

$f(\mathbf{x};\theta)=\sigma(W\mathbf{x}+\mathbf{b})$



Where Does Deep Learning Come In?

- Linear function: $f(\boldsymbol{x},\theta)=W\boldsymbol{x}+b$
- What loss function could we use for this *n*-ary classification problem?
  - Cross entropy
- Non-linear transformation:

$$Loss = -\frac{1}{N}\sum_{i=1}^{N} \hat{y_i}\, log(P(y_i = j\,|x_i)) \qquad P(y_i = j\,|x_i) = \frac{e^{f(x_i;W,b)}}{\sum_{j=1}^{C} e^{f(x_j;W,b)}}$$

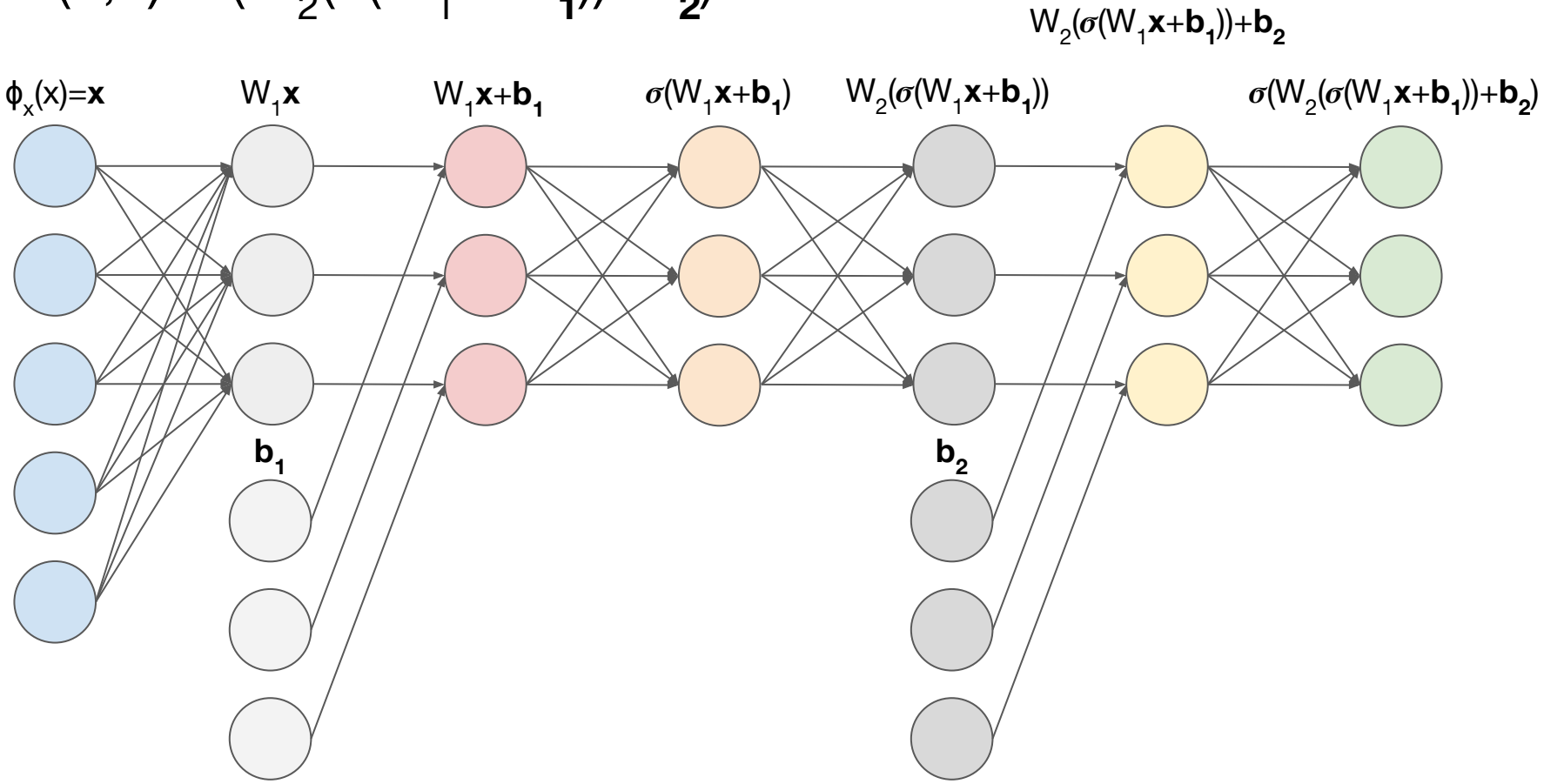
- Our function can only capture the degree to which each dimension of the input, $\boldsymbol{x_i}$, contributes to the likelihood of each class $\boldsymbol{y_j}$
  - “... brought out the ice cold food.”
  - “... brought out the ice cold gazpacho.”
  - “... Amazing service all around. Yeah right! 🙃”

Where Does Deep Learning Come In?

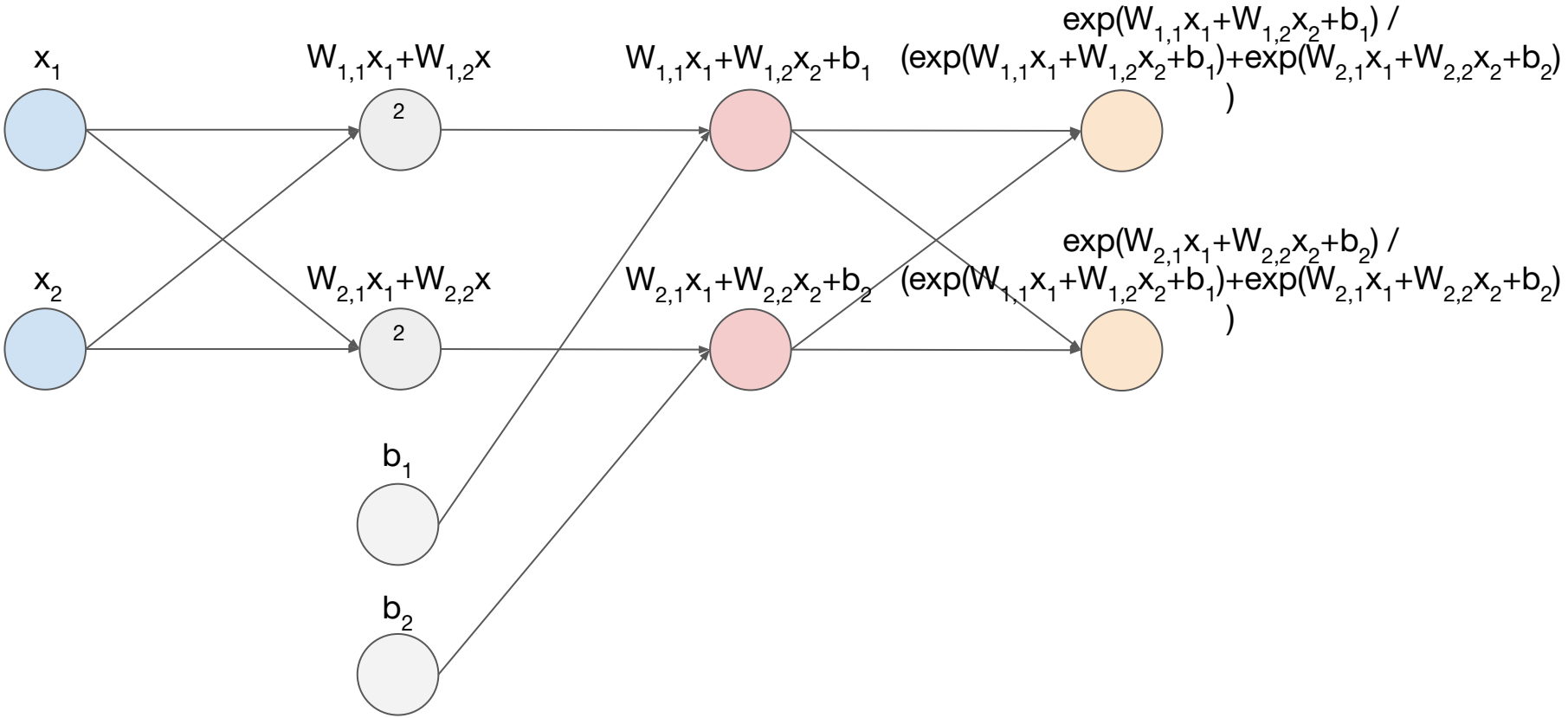
- $f(\mathbf{x},\theta)=\sigma(W\mathbf{x}+b)$
  - Linear function with nor
- If we need to make decision
- $f(\mathbf{x},\theta)=\sigma(W_2(\sigma(W_1\mathbf{x}+b_1))+b_2)$
  - First layer *latent variables* $\sigma(W_1\mathbf{x}+b_1)$ are each activated by linear combination of values of $\boldsymbol{x}_i$ input variables
  - Now W_2, b_2 can mix signals from the first *layer*; the final outputs $\sigma(W_2(\sigma(W_1\mathbf{x}+b_1))+b_2)$ are each activated by linear combinations of the latent variables from layer one
  - So our class logits can now consider combinations of input words like "**cold gazpacho**" and "**Amazing … Yeah right**!"

Quick terminology note, these basic transformations are called *perceptron layers* / *fully-connected layers*

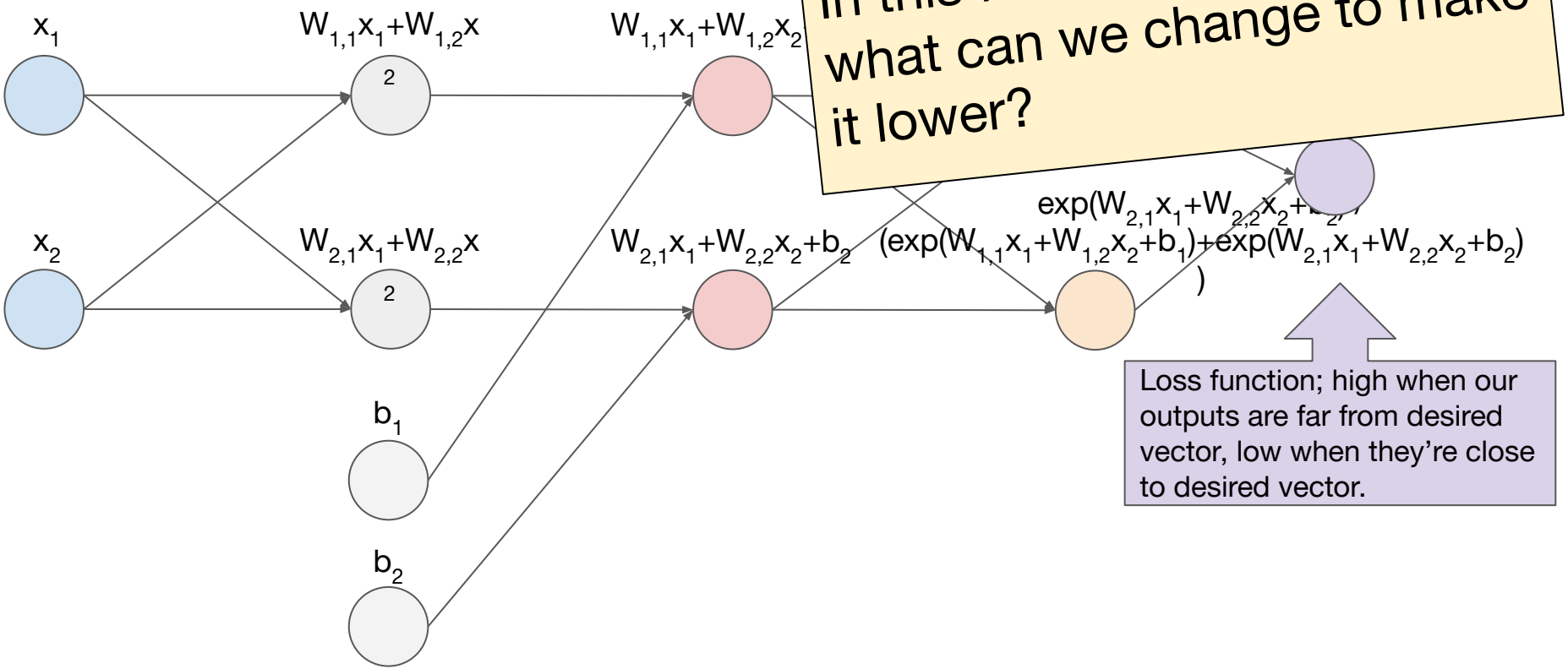
$$f(\mathbf{x},\theta)=\sigma(W_2(\sigma(W_1\mathbf{x}+\mathbf{b_1}))+\mathbf{b_2})$$



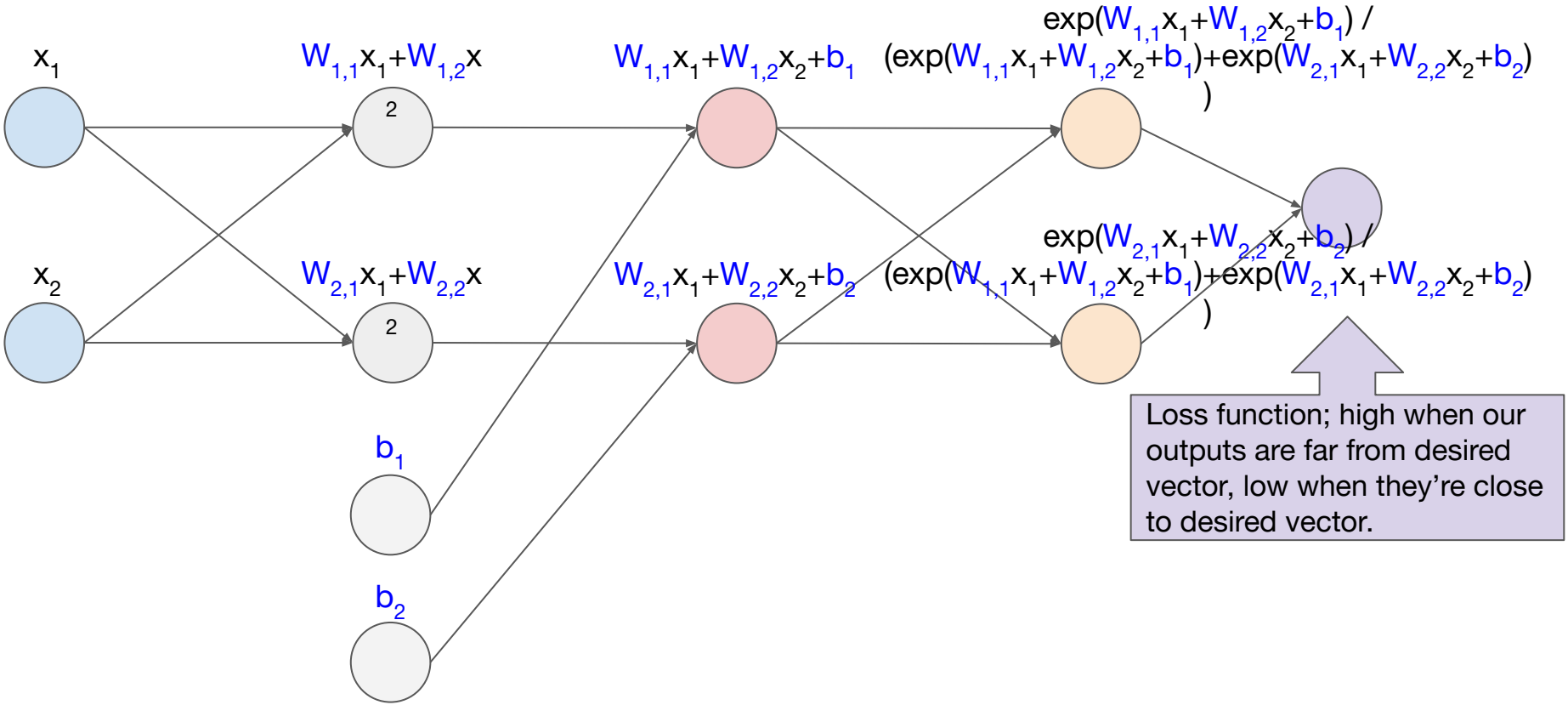
Neural Networks as Function Composition; $\sigma(W_1\mathbf{x}+\mathbf{b_1})$



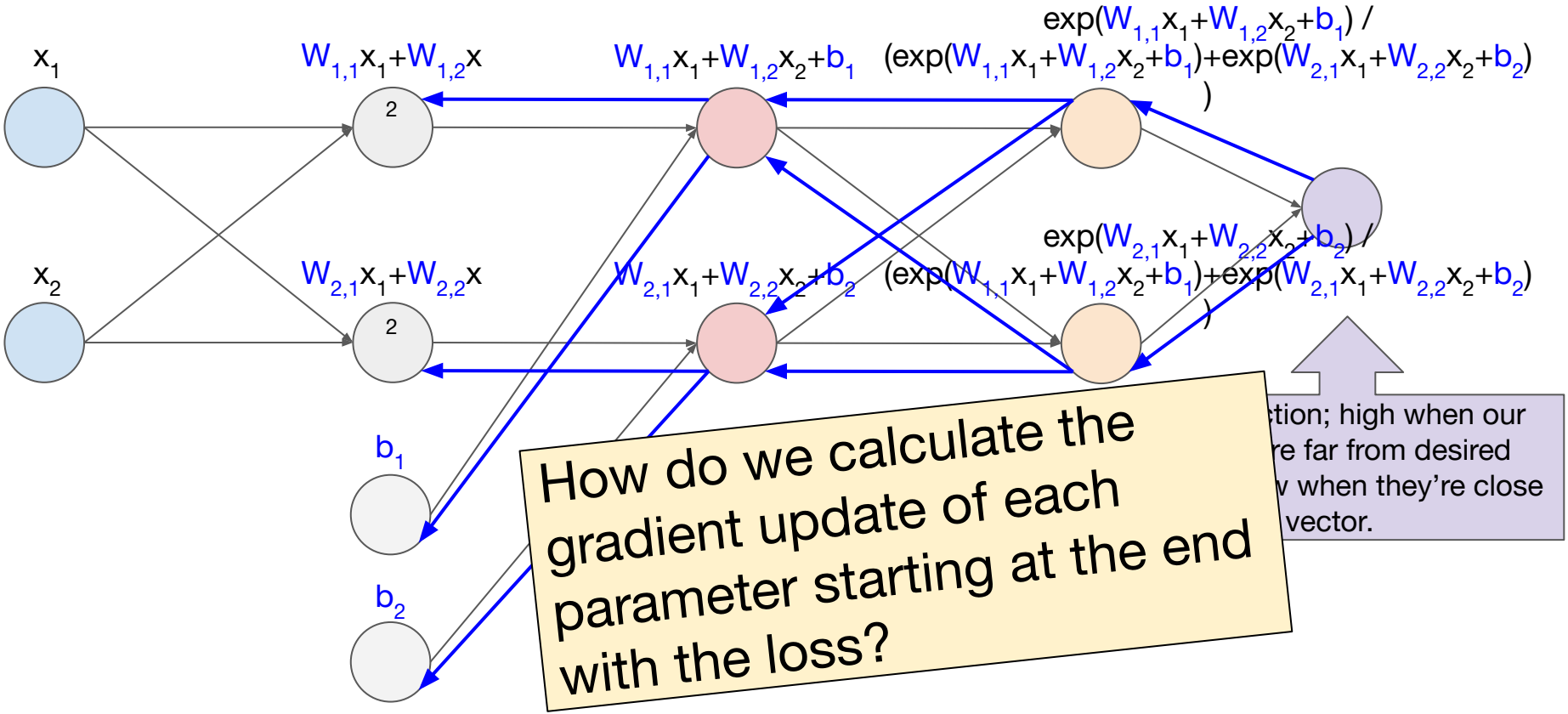
Forward Pass



NNs are Parameterized, Compositional Functions



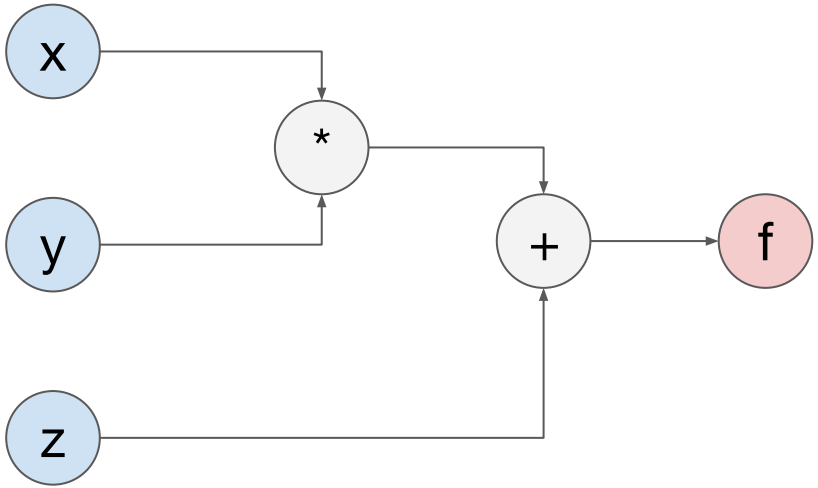
Backwards Pass



We Can Employ a Computation Graph

Consider function:

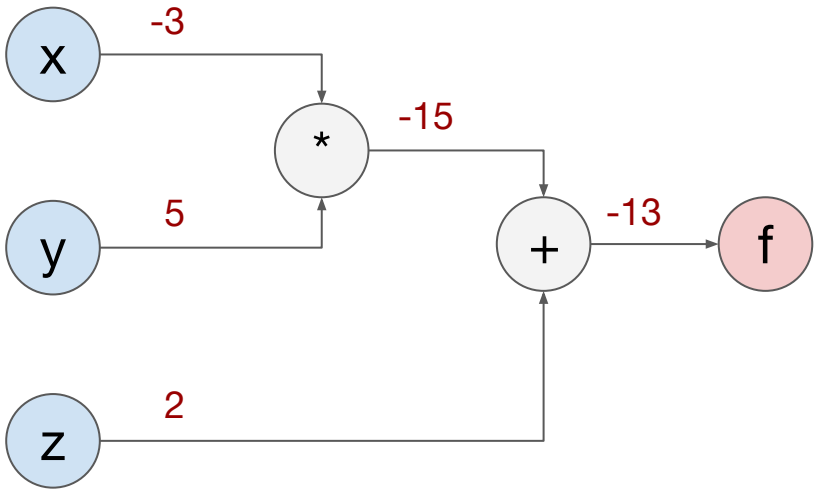
$f(x,y,z) = x*y + z$



Employ a Computation Graph

Consider function:

$f(x,y,z) = x*y + z$



We need to be able to take partial derivatives

-0.1	0.2	0.15	0.05
0.25	0.05	0.2	-0.1
0.25	0.15	-0.15	0.4

W

L(W) = 1.6688

-0.1+ 0.001	0.2	0.15	0.05
0.25	0.05	0.2	-0.1
0.25	0.15	-0.15	0.4

W+h

L(W+h) = 1.6693

0.5	?	?	?
?	?	?	?
?	?	?	?

gradient (dW)

dW_1=(1.6693 - 1.6688) / 0.001

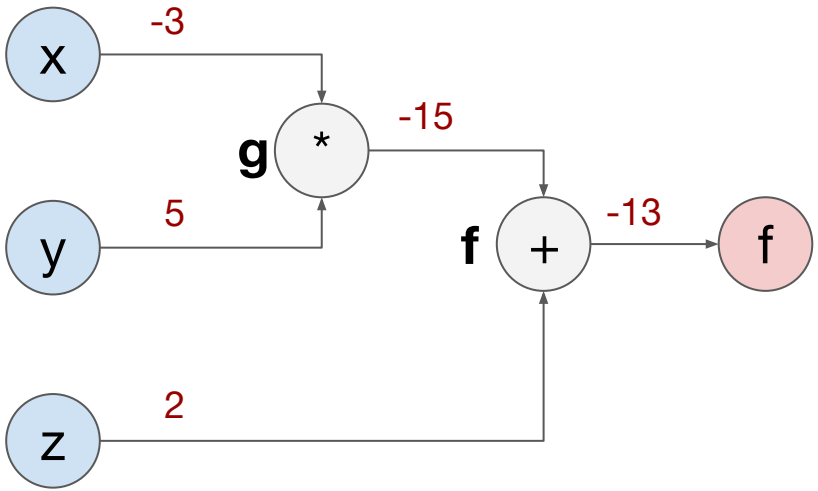
= 0.5

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Backpropagation via Computation Graph

Consider function:
*f(x,y,z) = x*y + z*

- NNs are compositions of functions
- Let's define:
  - g(x,y) = x*y
  - f(g,z) = g + z



Backpropagation via Computation Graph

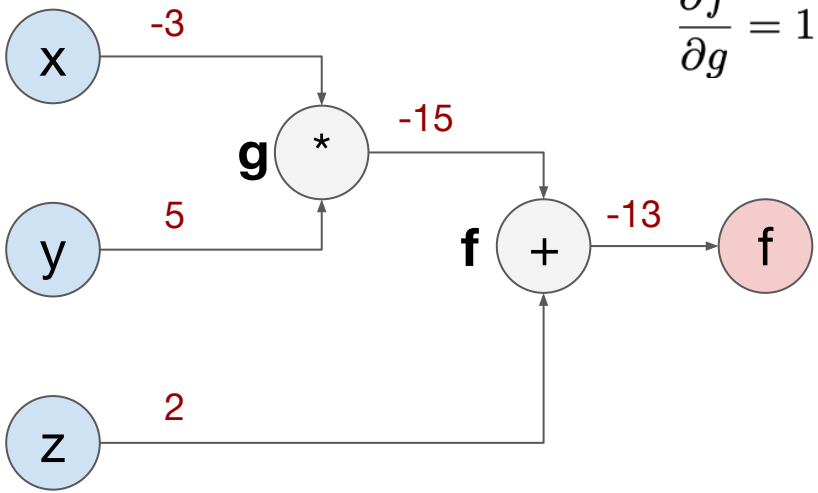
f(x,y,z) = x*y + z
g(x,y) = x*y; f(g,z) = g+z

- Let *x,y,z* be *parameters* we want to update
- Then we need:

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$

$$\frac{\partial g}{\partial x} = y \qquad \frac{\partial g}{\partial y} = x$$

$$\frac{\partial f}{\partial g} = 1 \qquad \frac{\partial f}{\partial z} = 1$$

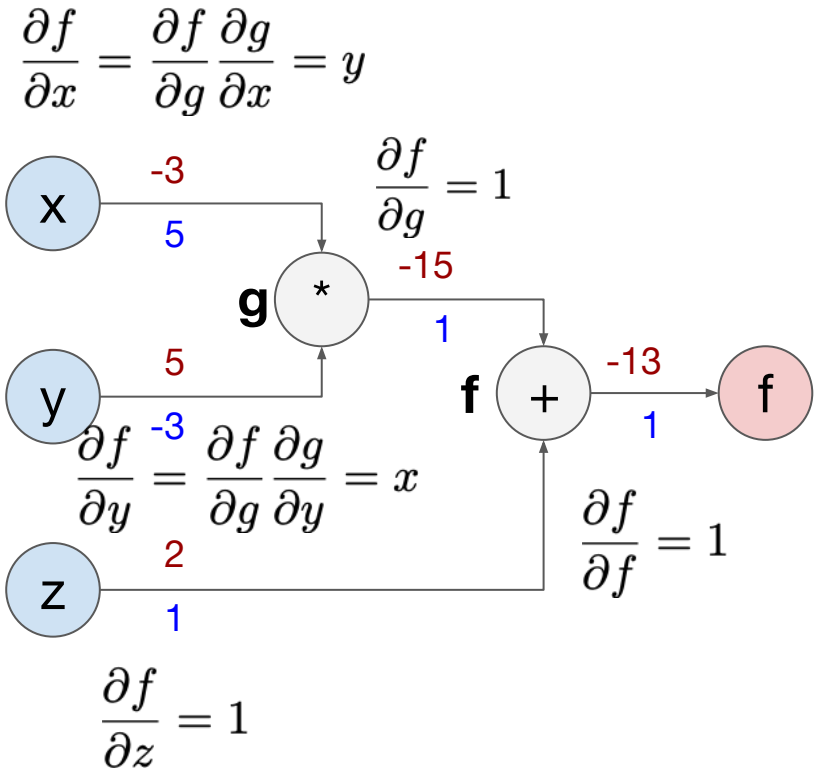


Backpropagation via Computation Graph

f(x,y,z) = x*y + z
g(x,y) = x*y; f(g,z) = g+z

- Let *x,y,z* be *parameters* we want to update
- Then we need:

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$



Backpropagation via Computation Graph

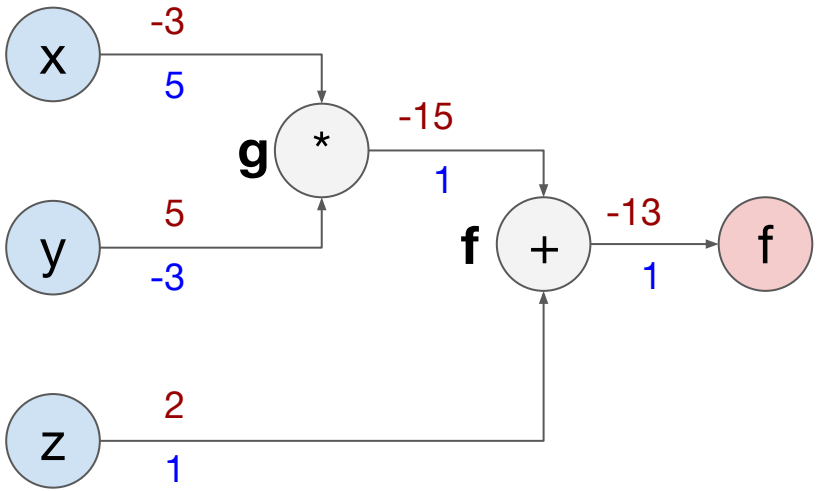
f(x,y,z) = x*y + z
g(x,y) = x*y; f(g,z) = g+z

- Then we get:

$$\frac{\partial f}{\partial x} = 5$$

$$\frac{\partial f}{\partial y} = -3$$

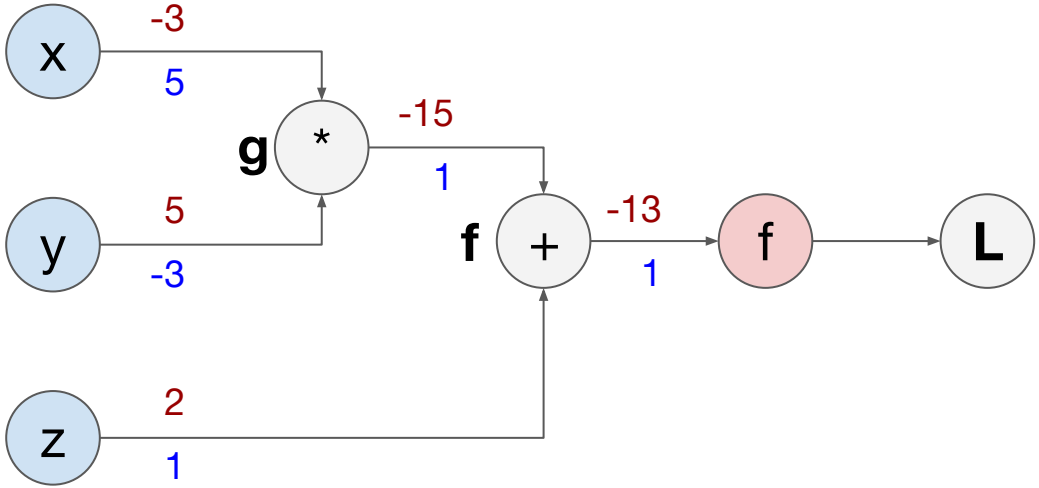
$$\frac{\partial f}{\partial z} = 1$$



Backpropagation with a Loss Function

f(x,y,z) = x*y + z
g(x,y) = x*y; f(g,z) = g+z

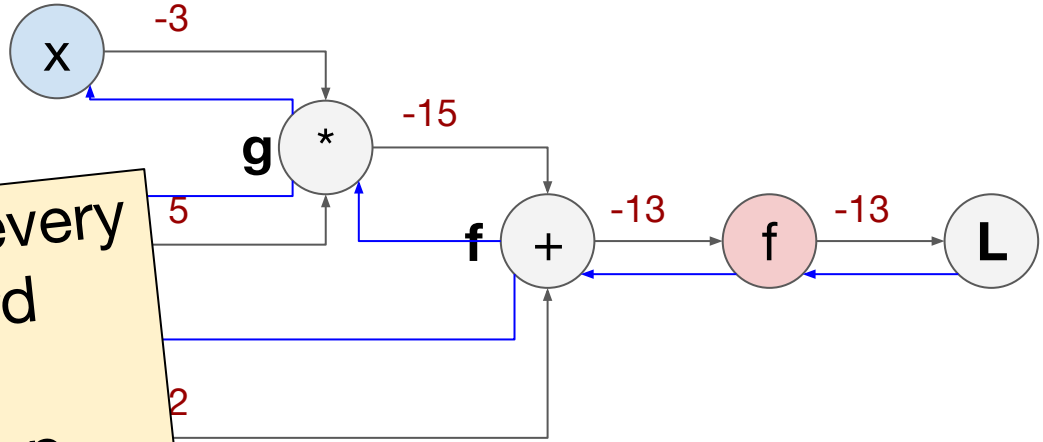
- If *f* is just our predictor, we can add a loss on top, call it function *L*
- *L(f)*



Backpropagation with a Loss Function

f(x,y,z) = x*y + z
g(x,y) = x*y; f(g,z) = g+z

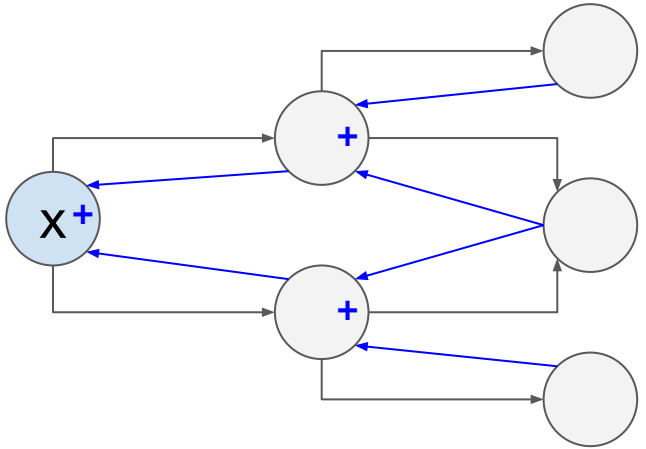
- Forward pass
- Backwards pass



The loss function *L* and every component of the forward function *f* need to be differentiable for backprop

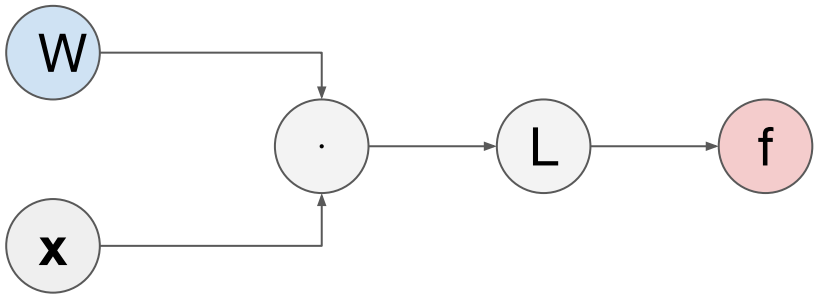
Backpropagation with Branches

- Gradients add at branches
- Enables single input to function as part of multiple outputs
- E.g.,
  - Multiple loss functions
  - “Skip” connections



Backpropagation with a Computation Graph

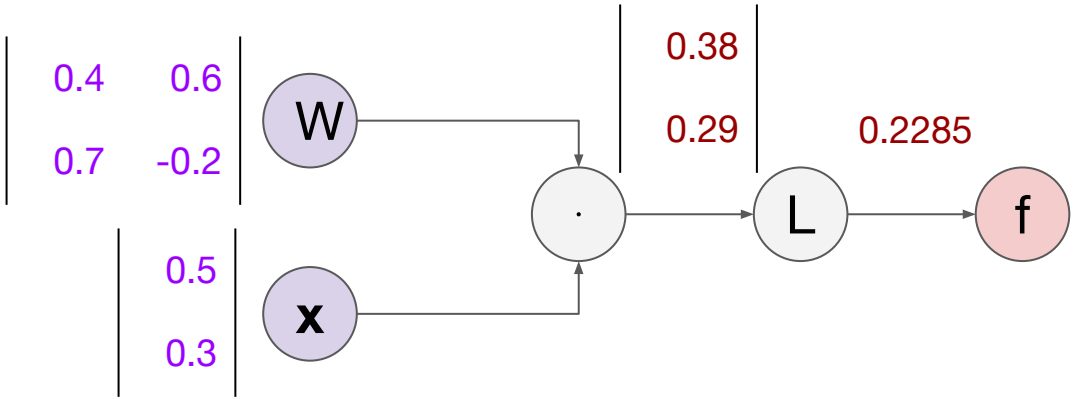
- Consider model: *M(**x**,W)* = *W**x*** with learnable params *W*
- Let's use the L2 loss function
- Then: $f(x, W) = ||W \cdot x||^2 = \Sigma_{i=1}^{n}(W \cdot x)_i^2$



Backpropagation with a Computation Graph

$$f(x, W) = ||W \cdot x||^2 = \Sigma_{i=1}^{n} (W \cdot x)_i^2$$

- Learnable params
- Forward pass
- Backwards pass



We need to be able to take partial derivatives

-0.1	0.2	0.15	0.05
0.25	0.05	0.2	-0.1
0.25	0.15	-0.15	0.4

W

L(W) = 1.6688

-0.1+ 0.001	0.2	0.15	0.05
0.25	0.05	0.2	-0.1
0.25	0.15	-0.15	0.4

W+h

L(W+h) = 1.6693

0.5	?	?	?
?	?	?	?
?	?	?	?

gradient (dW)

dW_1=(1.6693 - 1.6688) / 0.001

= 0.5

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Backpropagation with a Computation Graph

$$f(x, W) = ||W \cdot x||^2 = \Sigma_{i=1}^{n} (W \cdot x)_i^2$$

We want: $\frac{\partial f}{\partial W_{i,j}}, \frac{\partial f}{\partial x_i}$

Define: $q = W \cdot x = \begin{pmatrix} W_1 x_1 + \ldots + W_1 x_n \\ \ldots \\ W_n x_1 + \ldots + W_n x_n \end{pmatrix}$

Rewrite: $f(q) = ||q||^2 = q_1^2 + \ldots + q_n^2$

$$\frac{\partial q_k}{\partial W_{i,j}} = 1_{k=i} x_j$$

$$\frac{\partial f}{\partial W_{i,j}} = \Sigma_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}} = \Sigma_k (2q_k)(1_{k=i} x_j) = 2q_i x_j$$

$$\frac{\partial q_k}{\partial x_i} = W_{k,i}$$

$$\frac{\partial f}{\partial x_i} = \Sigma_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial x_i} = \Sigma_k 2q_k W_{k,i}$$

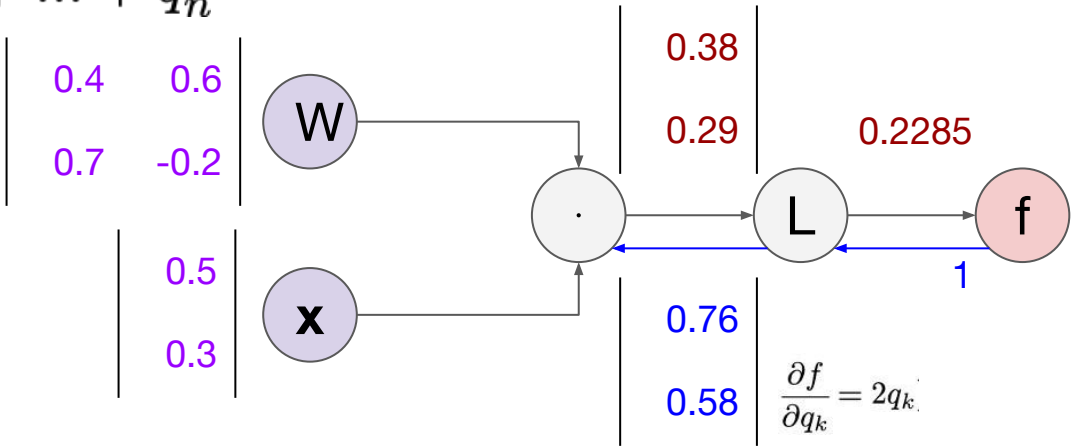
Backpropagation with a Computation Graph

$$f(x, W) = ||W \cdot x||^2 = \Sigma_{i=1}^{n} (W \cdot x)_i^2$$

$$q = W \cdot x = \begin{pmatrix} W_1 x_1 + \ldots + W_1 x_n \\ \ldots \\ W_n x_1 + \ldots + W_n x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \ldots + q_n^2$$

- Learnable params
- Forward pass
- Backwards pass

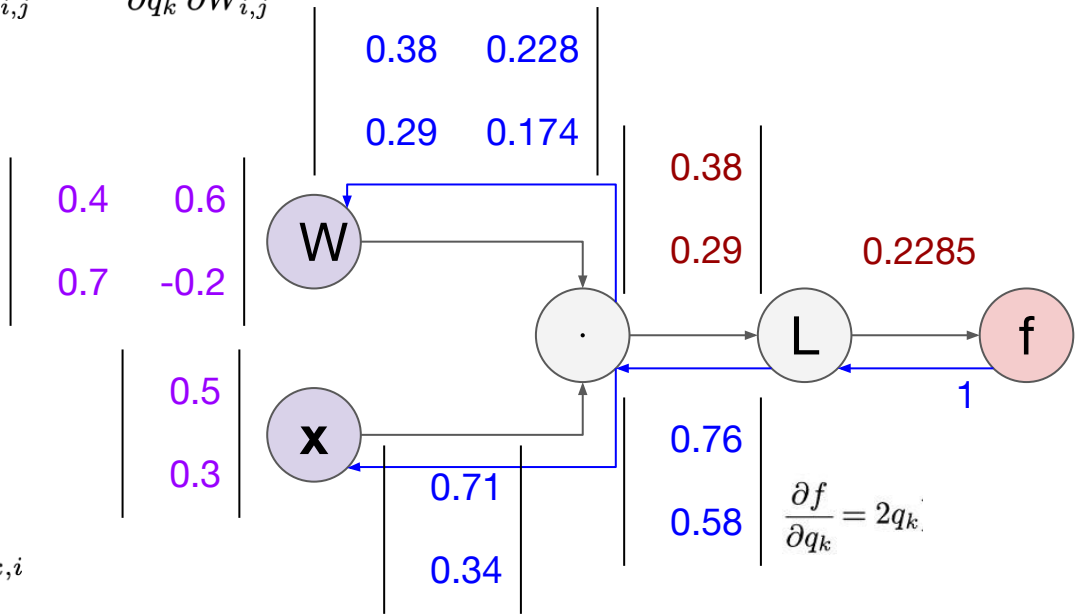


Backpropagation with a Computation Graph

$$f(x, W) = ||W \cdot x||^2 = \Sigma_{i=1}^{n} (W \cdot x)_i^2$$

- Learnable params
- Forward pass
- Backwards pass

$$\frac{\partial f}{\partial W_{i,j}} = \Sigma_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}} = \Sigma_k (2q_k)(1_{k=i} x_j) = 2q_i x_j$$



$$\frac{\partial f}{\partial x_i} = \Sigma_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial x_i} = \Sigma_k 2q_k W_{k,i}$$

Backpropagation with a Computation Graph

$$f(x, W) = ||W \cdot x||^2 = \Sigma_{i=1}^{n} (W \cdot x)_i^2$$

- Now we can take a gradient step
- Let learning rate be 0.1, then:

W $\begin{vmatrix} 0.4 & 0.6 \\ 0.7 & -0.2 \end{vmatrix}$ - 0.1* $\begin{vmatrix} 0.38 & 0.228 \\ 0.29 & 0.174 \end{vmatrix}$ = $\begin{vmatrix} 0.362 & 0.5772 \\ 0.671 & -0.2174 \end{vmatrix}$

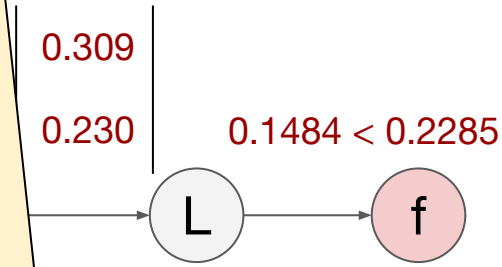
x $\begin{vmatrix} 0.5 \\ 0.3 \end{vmatrix}$ - 0.1* $\begin{vmatrix} 0.71 \\ 0.34 \end{vmatrix}$ = $\begin{vmatrix} 0.429 \\ 0.266 \end{vmatrix}$

Backpropagation with a Computation Graph

$$f(x, W) = ||W \cdot x||^2 = \Sigma_{i=1}^{n} (W \cdot x)_i^2$$

- Learnable params
- Forward pass
- Backwards pass

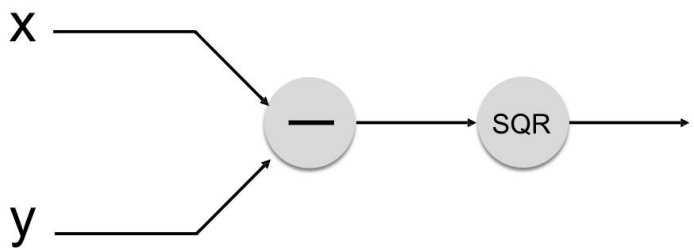
Rather than adding tiny noise to every param and calculating the loss difference, we calculate partial derivatives of systems of equations for more efficient learning!



Computation Graph Non-Linearities

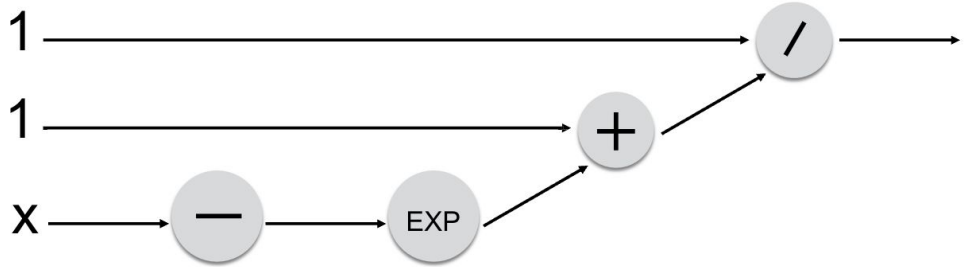
L2-loss

$$L_2(x, y) = (x - y)^2$$



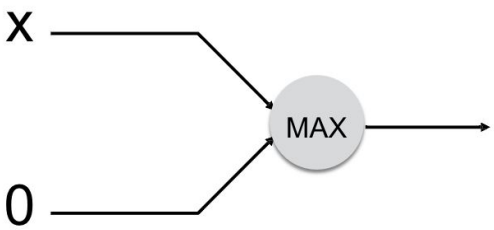
Sigmoid

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

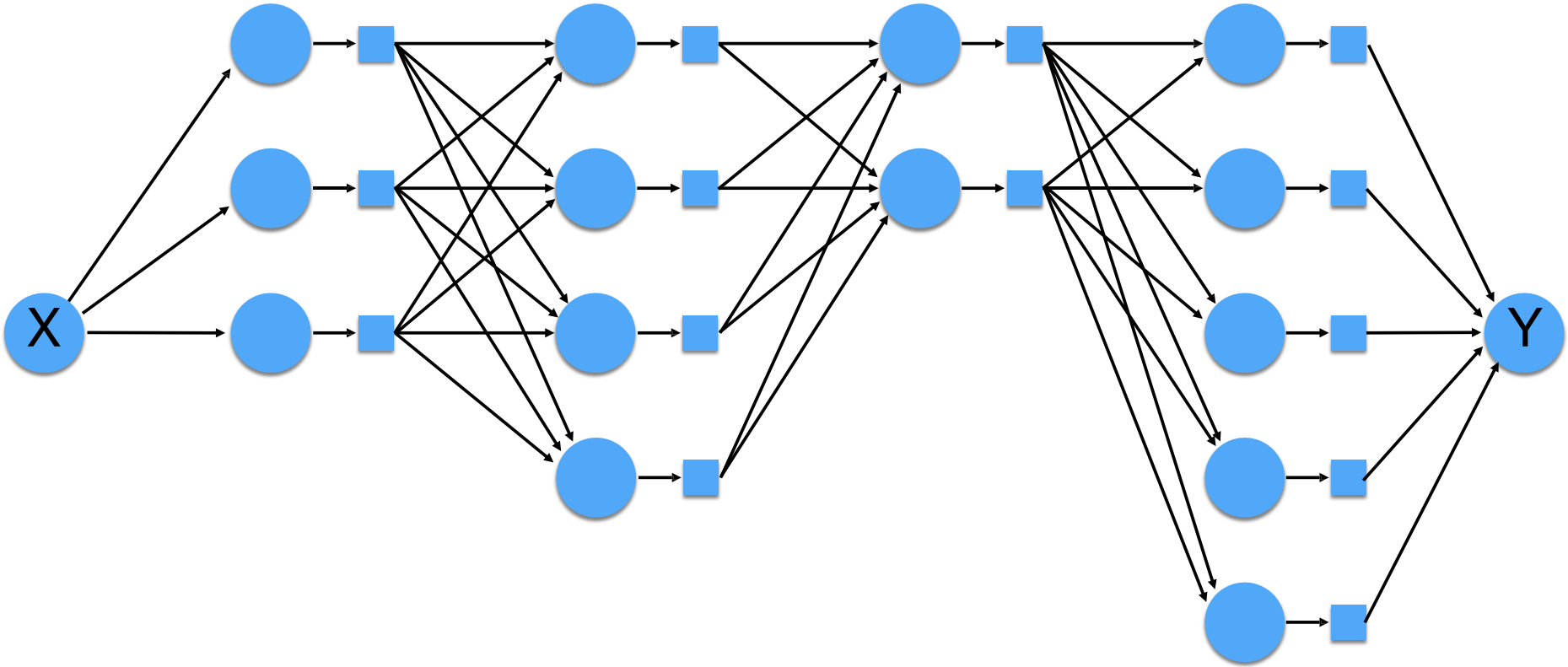


ReLU

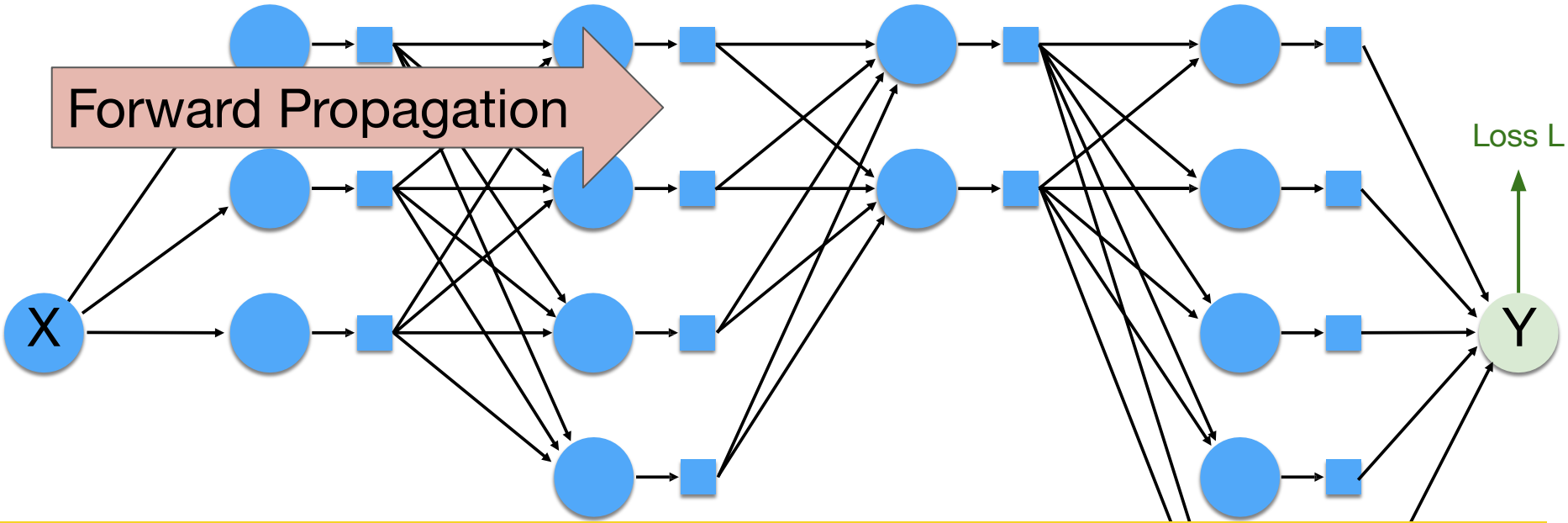
$$\text{ReLU}(x) = \max(x, 0)$$



Recap: Forward and Backward Propagation



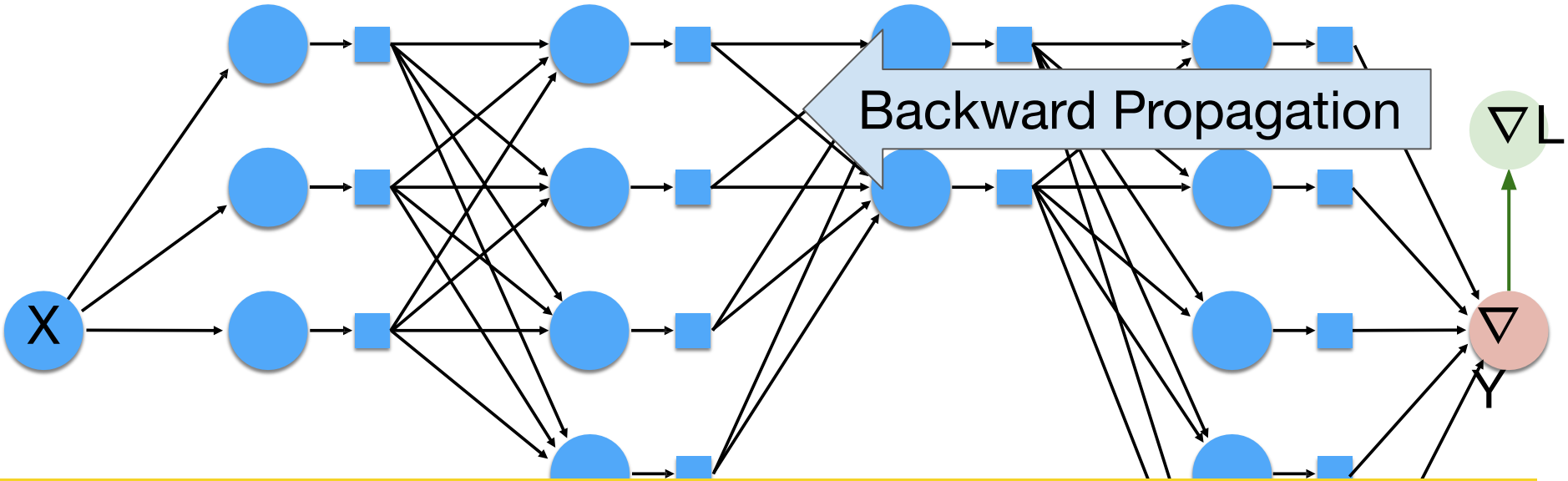
Recap: Forward and Backward Propagation



Compute a loss using an estimated Y and the ground truth Y*.

e.g., $L(W) = \| f(X; W) - Y^* \|_2$

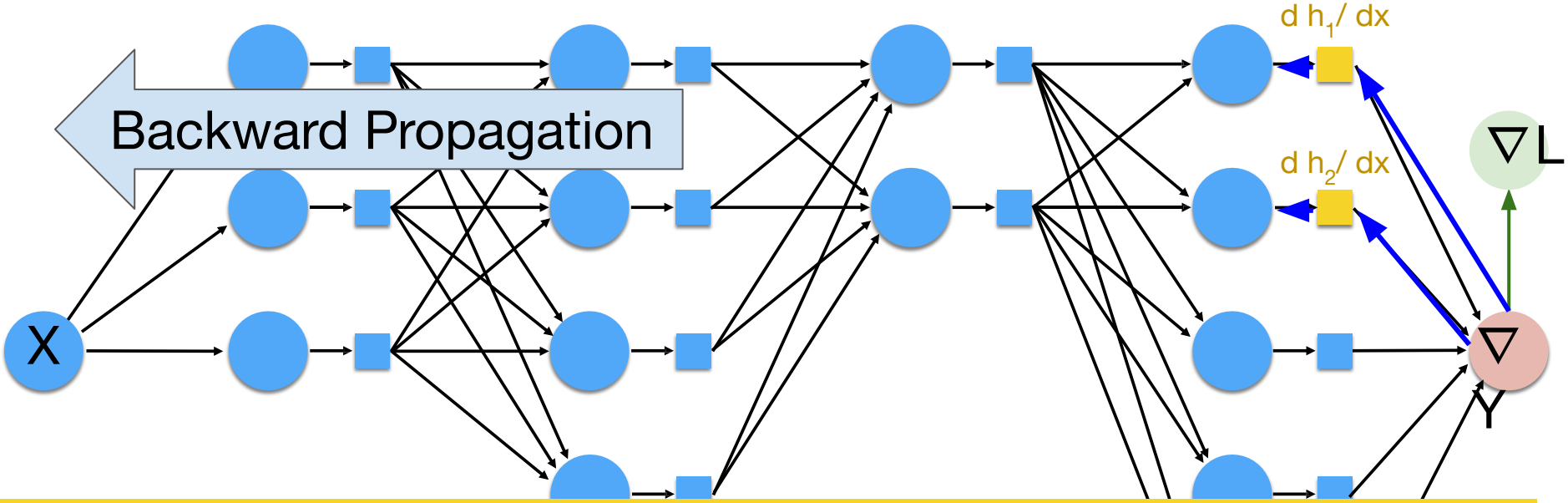
Recap: Forward and Backward Propagation



Compute gradients of Y using ∇L.

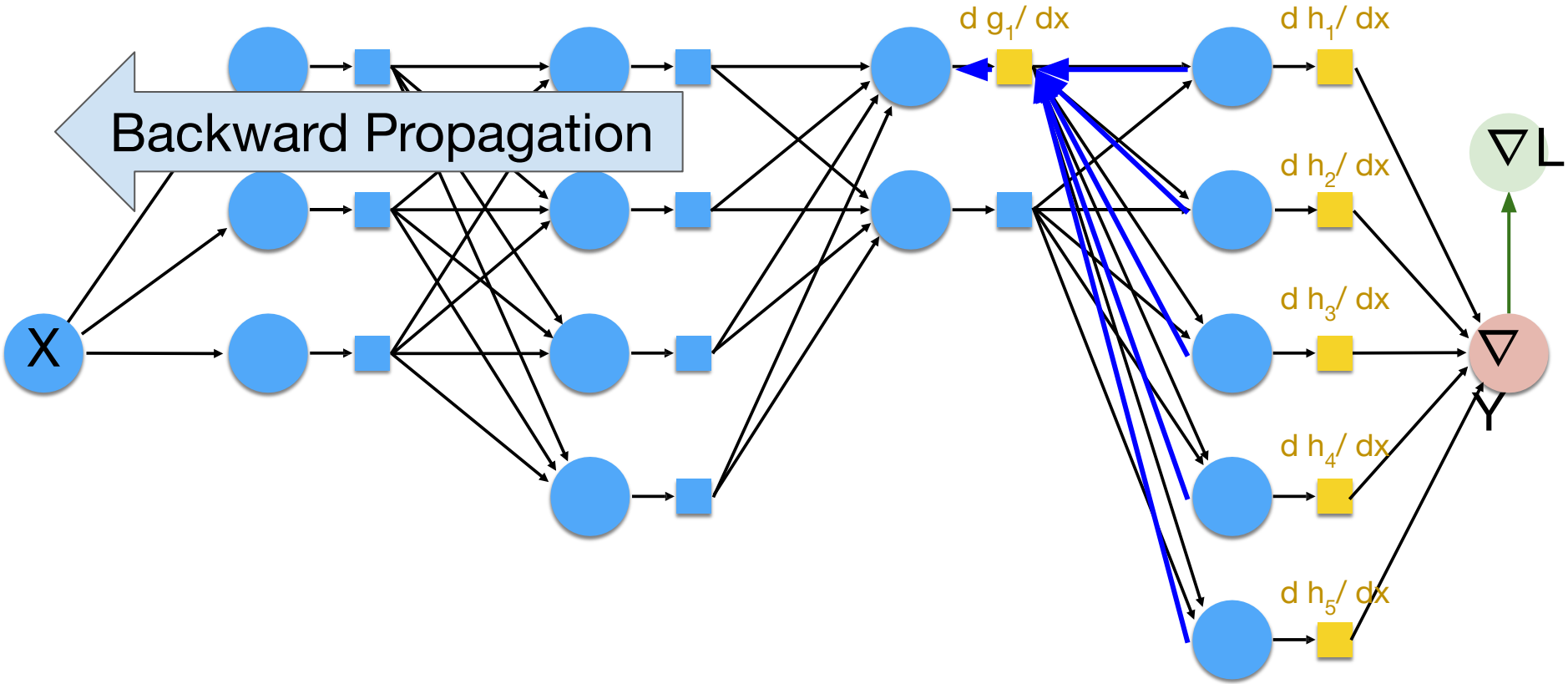
$$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Y} \cdot \frac{\partial Y}{\partial X}$$

Recap: Forward and Backward Propagation



Chain rule $\frac{\partial Y}{\partial X} = \sum_{k=1}^{N} \frac{\partial Y}{\partial h_i} \cdot \frac{\partial h_i}{\partial X}$

Recap: Forward and Backward Propagation

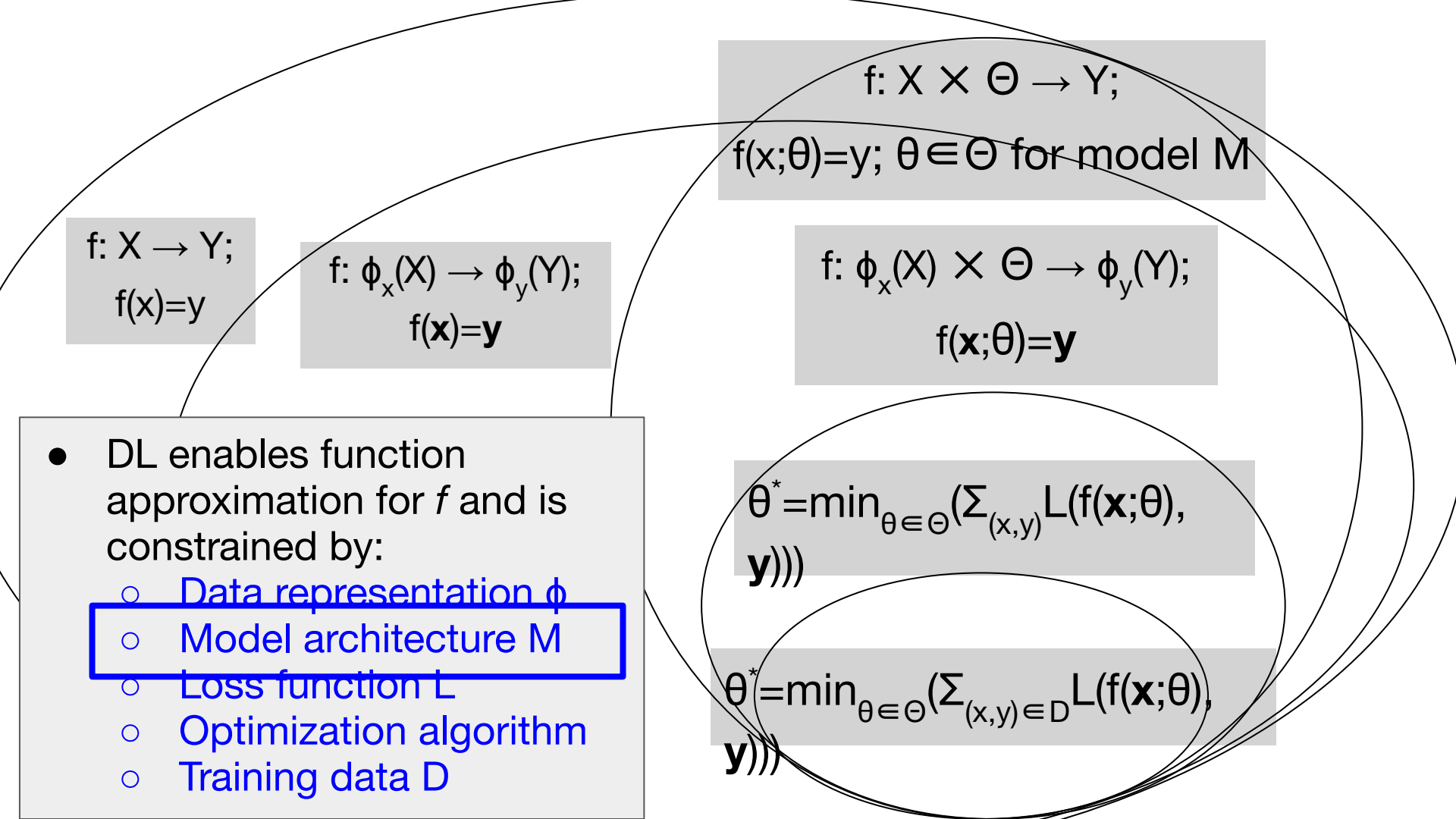


Recap: Forward and Backward Propagation

- The loss function drives back propagation; without calculating loss, we're just making predictions (e.g., forward pass)
- A computation graph describes the relationship between inputs, learnable parameters, and outputs
- A backwards pass lets us calculate the gradient for a given input of our learnable parameters in a parallelized way
- For SGD, we want to then average those gradient calculations across a *minibatch* of input/output pairs

Overview of Today's Plan

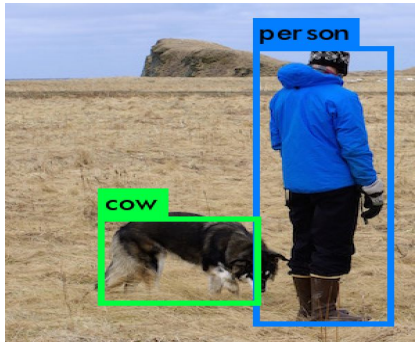
- ~~Course organization and deliverables~~
- ~~Training Neural Networks~~
  - Any questions before we move on?
- Convolutional Neural Networks


f: X → Y;
f(x)=y
f: ϕx(X) → ϕy(Y);
f(x)=y
f: X × Θ → Y;
f(x;θ)=y; θ∈Θ for model M
f: ϕx(X) × Θ → ϕy(Y);
f(x;θ)=y
θ*=minθ∈Θ(Σ(x,y)L(f(x;θ), y)))
θ*=minθ∈Θ(Σ(x,y)∈DL(f(x;θ), y)))
DL enables function approximation for f and is constrained by:
Data representation ϕ
Model architecture M
Loss function L
Optimization algorithm
Training data D

Representations and Models

- The nature of the input, output, and intuitive relation between them gives us a lot of information as DL practitioners about what kind of *model* we should use
- What kind of model might you use for language inputs?
  - Perceptron? Why or why not?
  - Recurrent models (RNN, LSTM, GRU)? Why?
  - Transformers? Why?
- What kind of model might you use for vision inputs?
  - Perceptron? Why or why not?
  - Convolutional models (CNN, ResNets)? Why?

Representations and Models



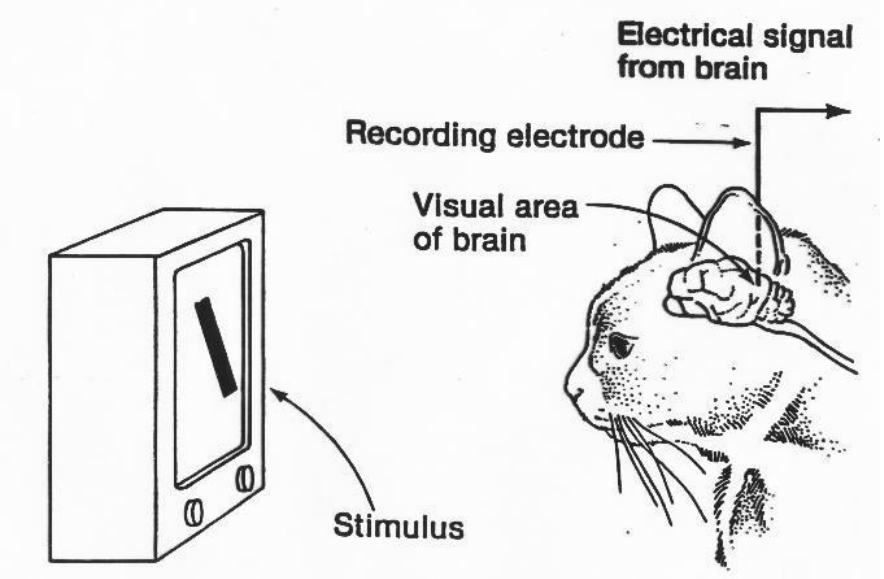
- Input space *X*?
  - Pixels in the image
- Representation?
  - $\phi_x(X)=\boldsymbol{R}^{W,H,3}$ RGB or HSV

- Output space *Y*?
  - Object classes *C* + bboxes
- Representation?
  - $\phi_y(Y) = C \times \boldsymbol{R}^{x,y,w,h} \times N$

- What model architecture might you use?
- What loss function would you want to optimize?

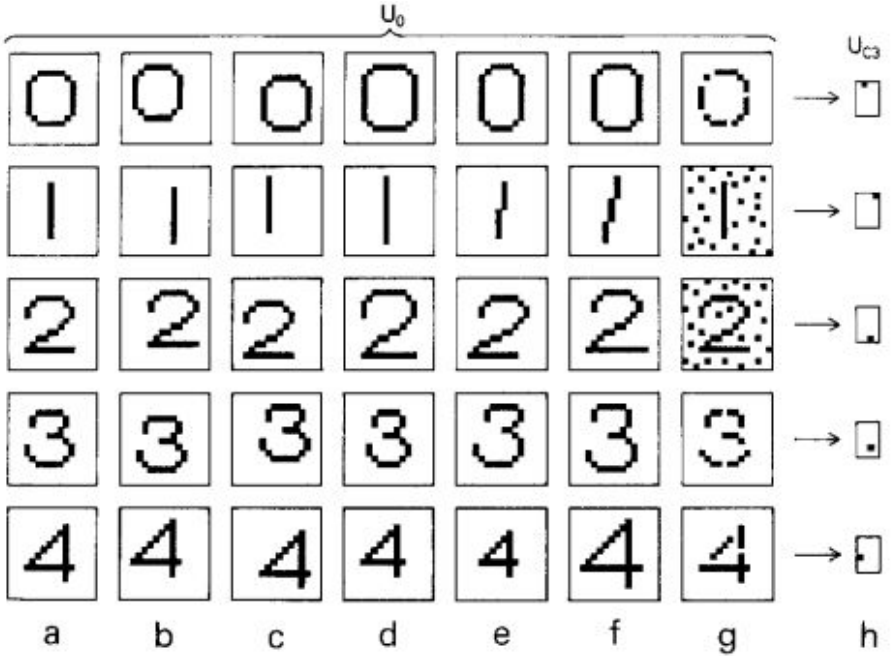
Origins of Convolutional Neural Networks

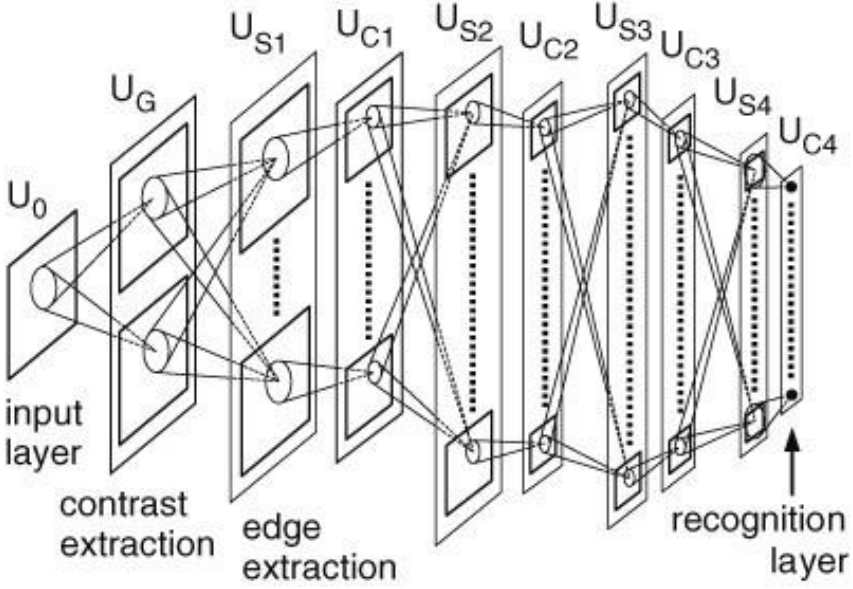
“They found that particular neurons in the visual cortexes of cats and monkeys—the areas in their brains responsible for processing visual information—didn't respond to simple points of light, but rather to lines, and in particular, lines and contours with specific orientations”



https://hub.jhu.edu/2015/07/14/golden-goose-award-wiesel-hubel/

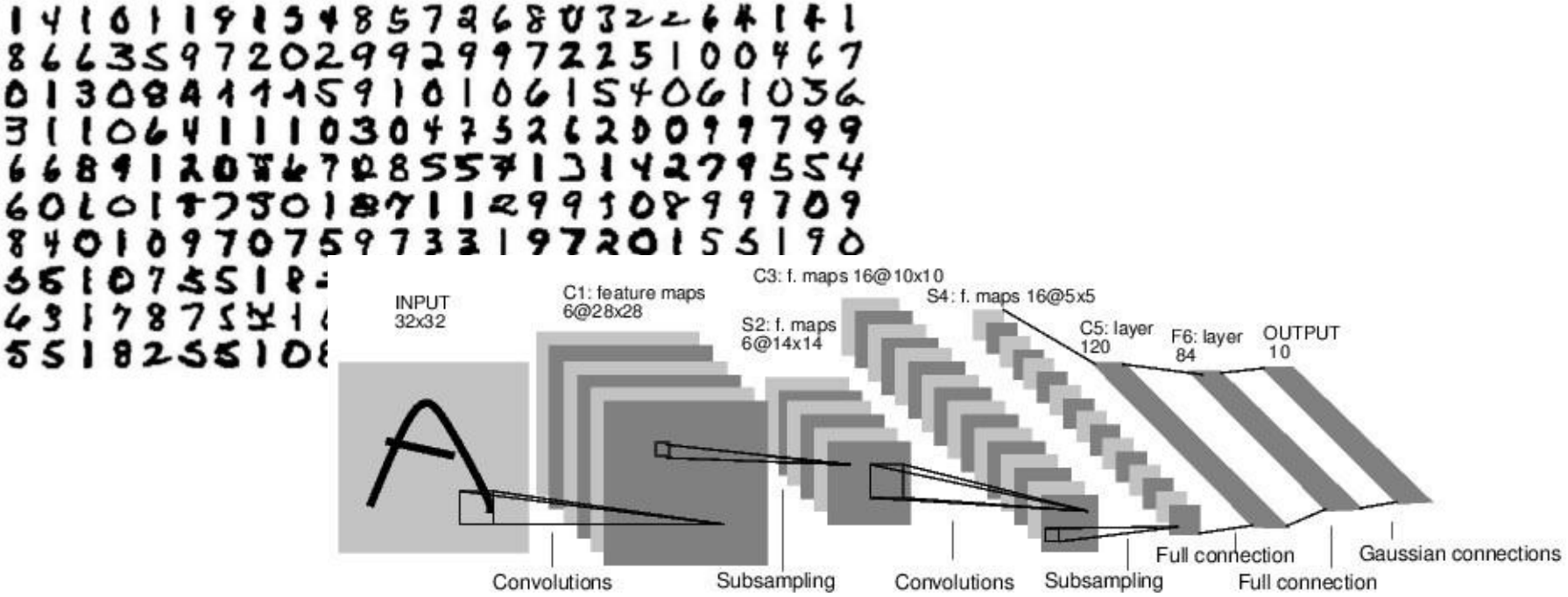
Origins of Convolutional Neural Networks





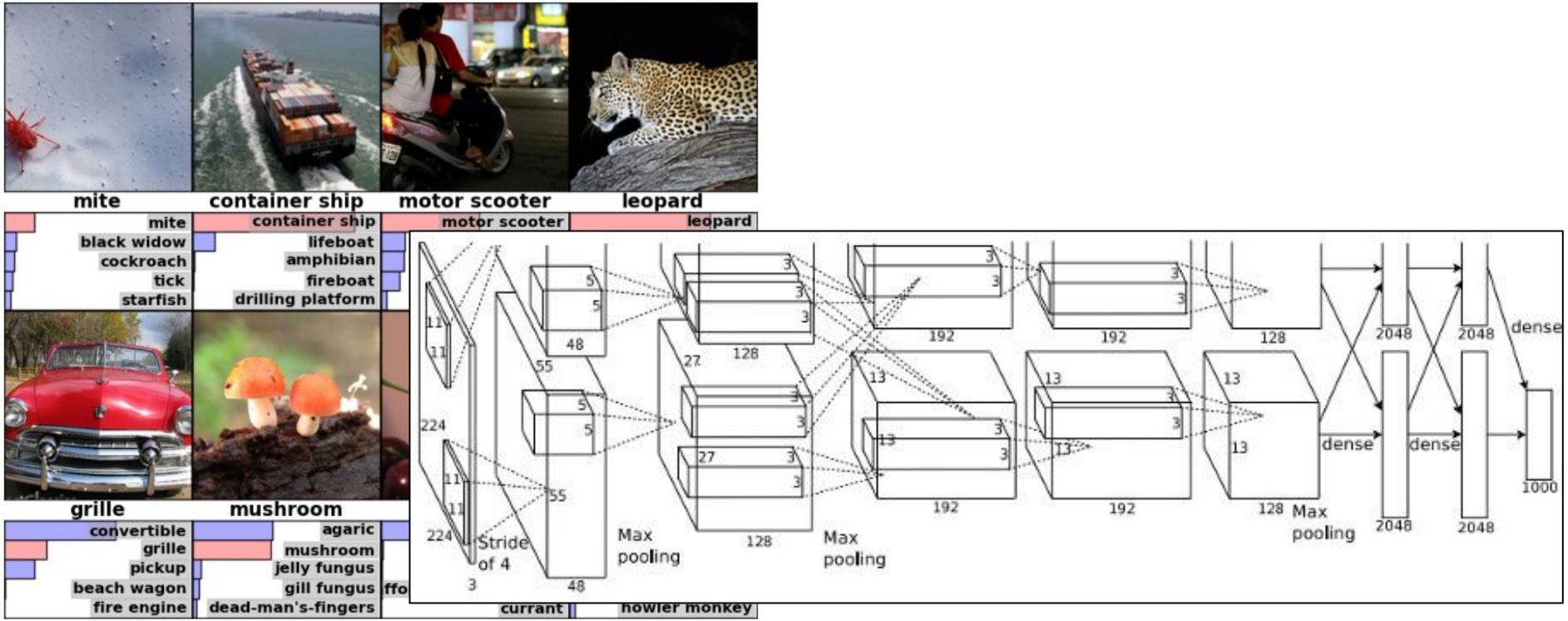
Fukushima, Kunihiko. **Neocognitron**: A self-organizing neural network model for a mechanism of visual pattern recognition. Competition and cooperation in neural nets **1982**.

Origins of Convolutional Neural Networks



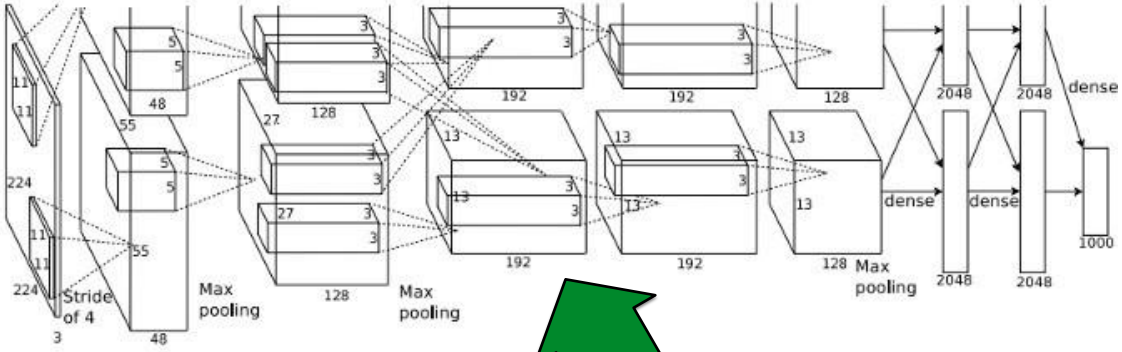
Y. LeCun, et. al. Handwritten digit recognition with a **back-propagation** network. NeurIPS **1989.**

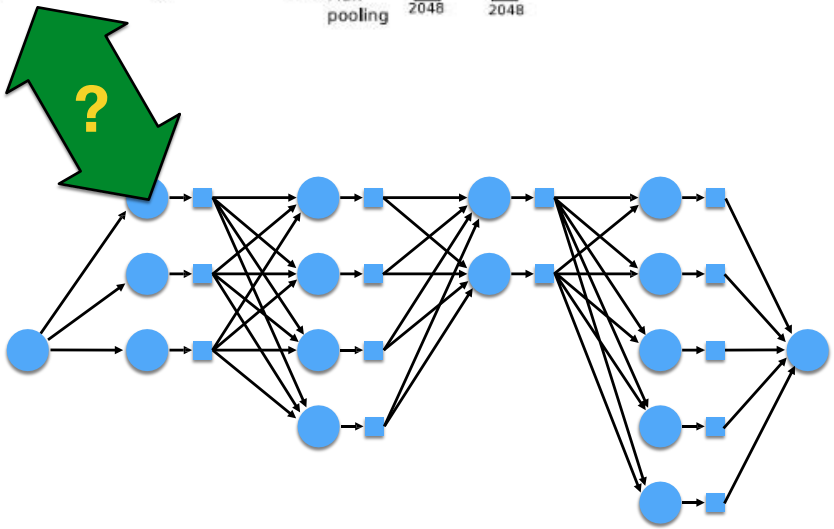
Origins of Convolutional Neural Networks



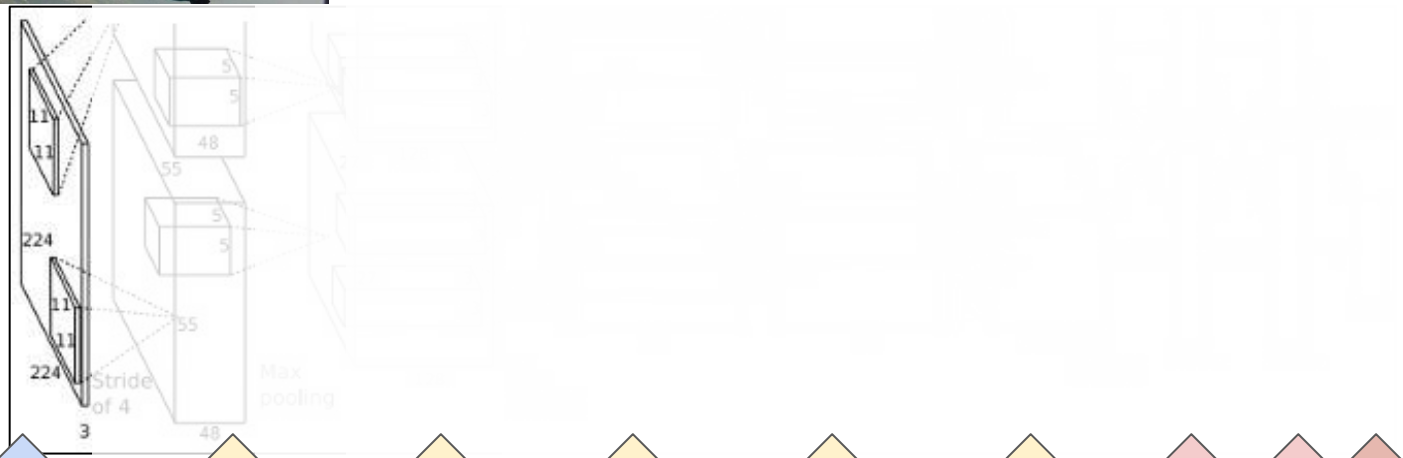
A Krizhevsky, et. al. ImageNet Classification with **Deep Convolutional Neural Networks**. NeurIPS **2012**.

Breaking Down AlexNet





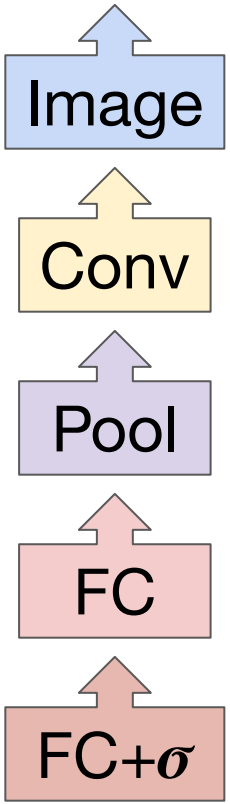
Breaking Down AlexNet



.01	car
.03	dog
.63	person
.11	airplane
.02	flower
…	…

Image
Conv
Conv
FC
FC+σ

Common Layers in a Deep Convolutional Neural Net



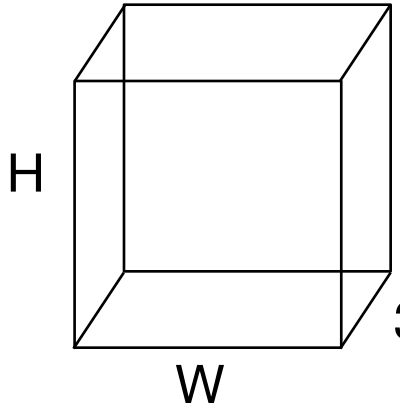
- **x** in our parlance; the vectorized input
- Convolution
- Pooling laye (in AlexNet, MaxPool)
- Fully-connected layer (i.e., *perceptron* layer)
- Softmax layer (fully connected layer + exponentiated normalization)

Each layer is a differentiable function whose *input* is the *output* of the previous layer (function)

What Are Our Options to Process an Image?

- Input space *X*?
  - Image (e.g., PNG)
- Representation?
  - $\phi_x(X) = \boldsymbol{R}^{W,H,3}$ RGB

$\phi_x($  $) = [0, 1]^{(w \times h \times 3)}$

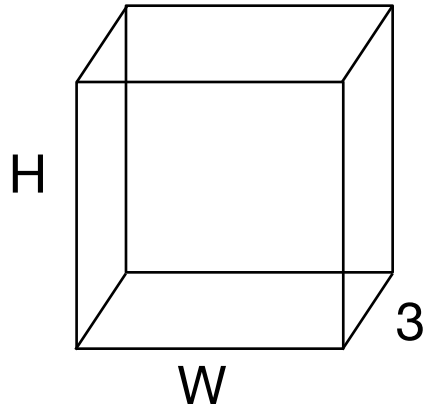


red, green, blue

Note: ϕ here is going to have to either crop or resize image inputs!

What Are Our Options to Process an Image?

$\phi_x($  $) = [0, 1]^{(w \times h \times 3)}$



- Let *w=h=300*. Then our *tensor* representation of input images is size 300x300x3 = 270k individual [0,1] values.
- Can we classify such an input using a linear model f(**x**) = W**x**+**b**?
- Yes!
- We could use a *fully-connected* layer between the input tensor and a vector of output classes.

Image Classification with a Fully-Connected Layer

$\phi_x($  $) = [0, 1]^{(w \times h \times 3)}$

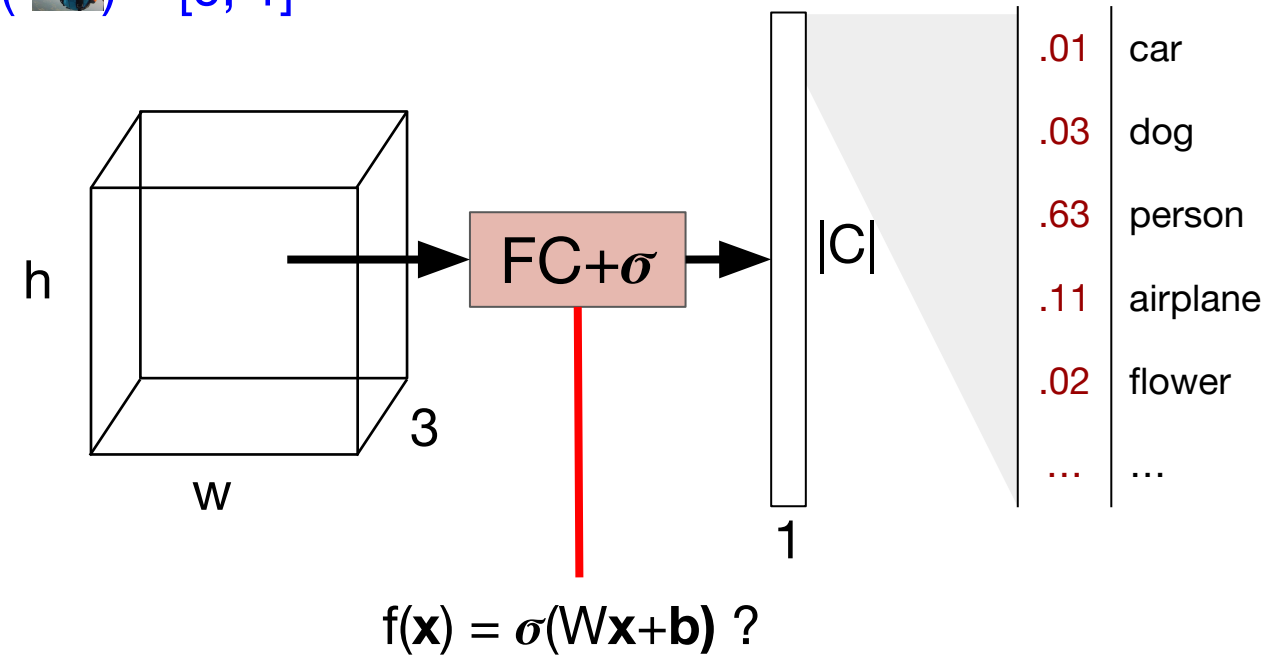


Image Classification with a Fully-Connected Layer

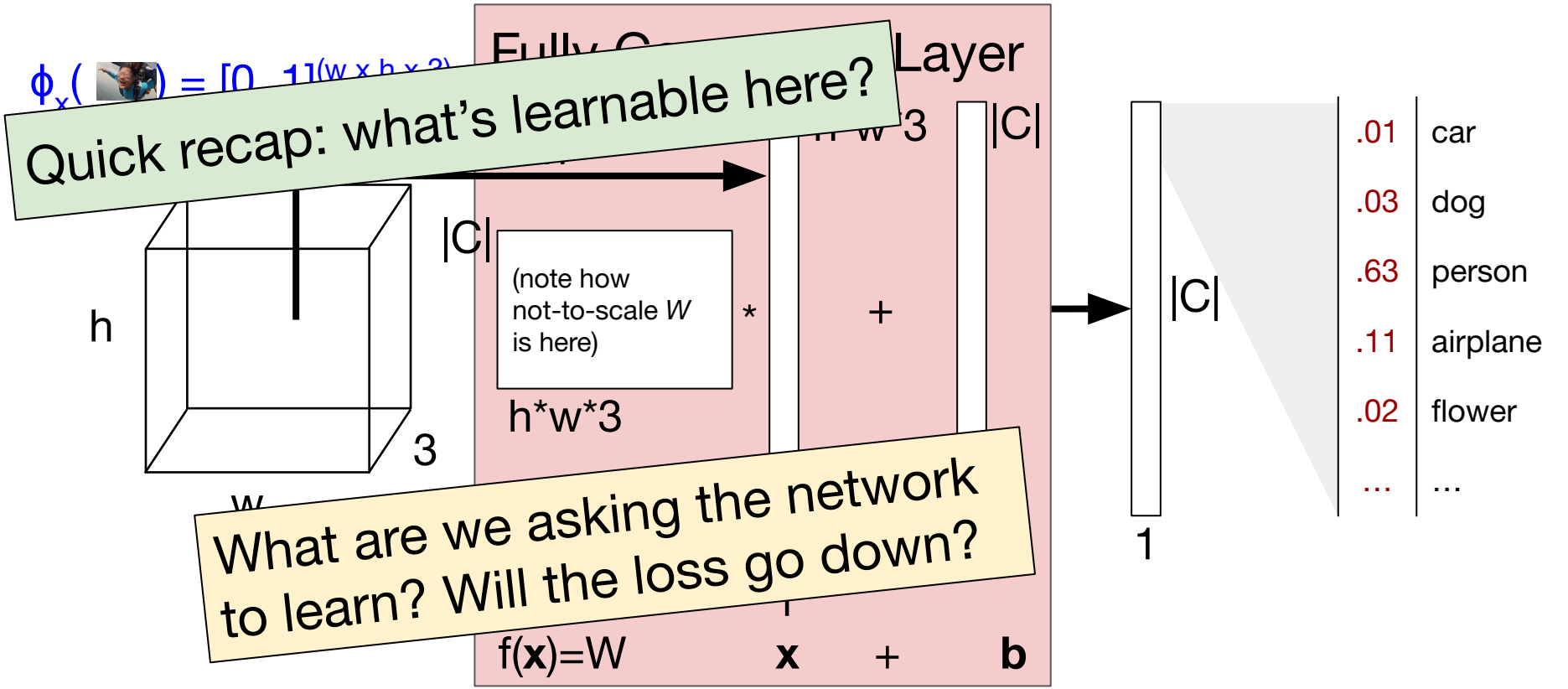
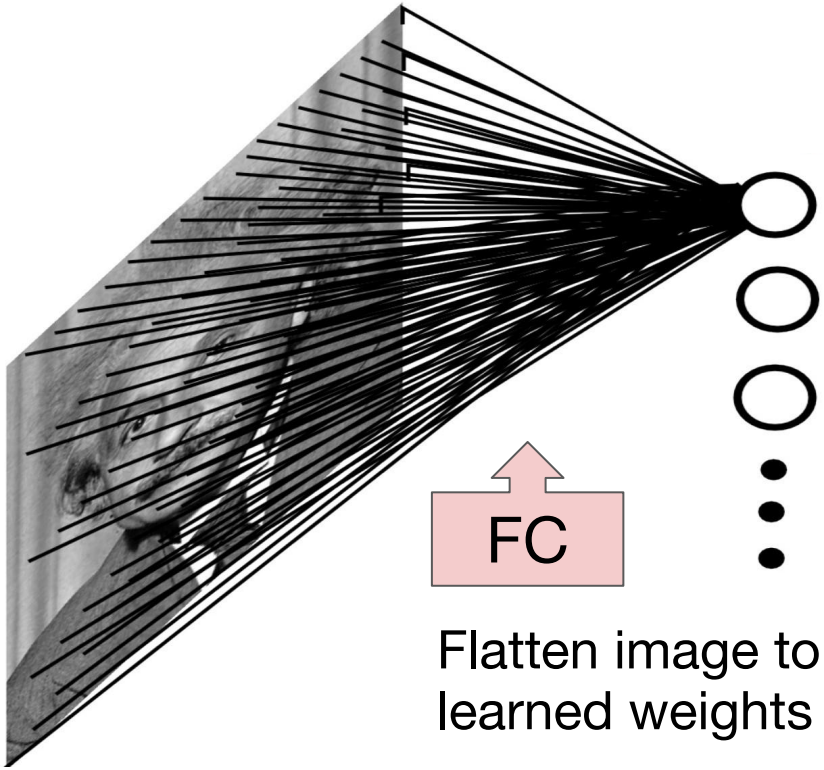


Image Classification with a Fully-Connected Layer

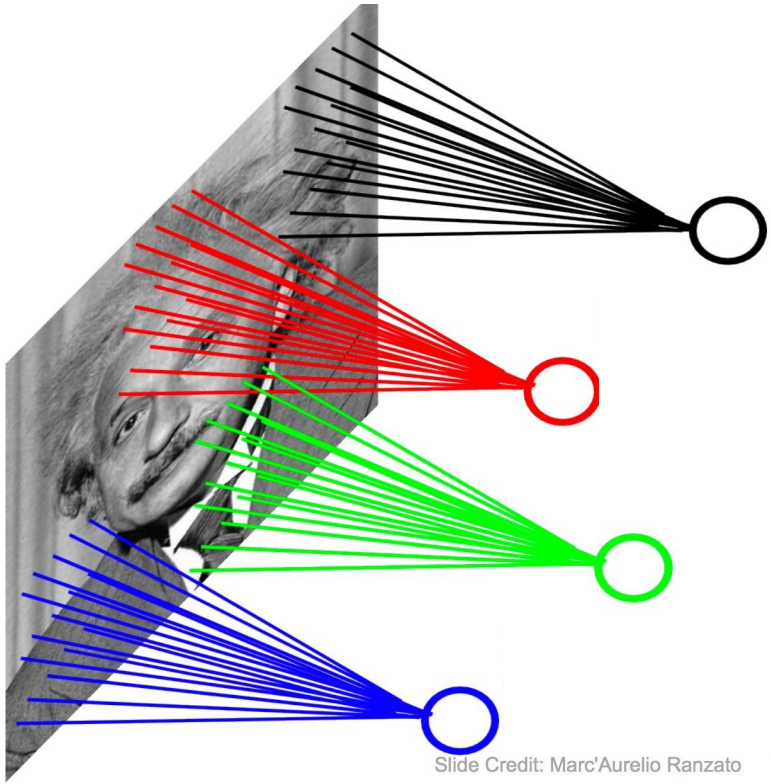


- Let *w=h=300*. Then our *tensor* representation of input images is size 300x300x3 = 270k individual [0,1] values.
- Then *W* is size |C| * 270k
- For 1000-way classification, we'd be committing to learning a weight matrix with 270 **million** parameters

Flatten image to s
learned weights *W* vector **b**

What *structural bias* are we failing to utilize?

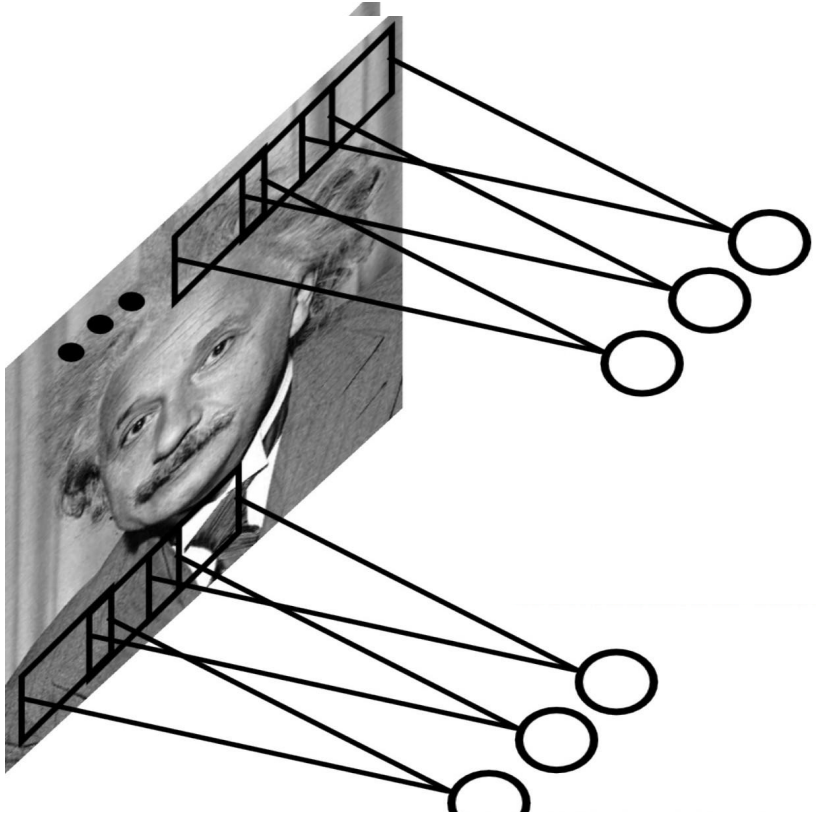
Structural Bias: Image Pixels Have Spatial Relationships



Slide Credit: Marc'Aurelio Ranzato

- When we flatten an image, we assume every pixel contributes independently to the final class
- We can instead learn a function of a *neighborhood* of pixels which are spatially co-located
- Each such function is called a *convolutional filter*

Structural Bias: Image Pixels Have Spatial Relationships



- A filter can be applied and learned across multiple neighborhoods by applying it with a *stride*
- A convolutional layer applies multiple learnable filters with a given *kernel* size and *stride length*
  - Kernel size: how tall/wide/deep the filter input is
  - Stride length: how many indices to move in each dimension between applications of the kernel (e.g., 1)

Convolutional Layer

$\phi_x($  $) = [0, 1]^{(32 \times 32 \times 3)}$

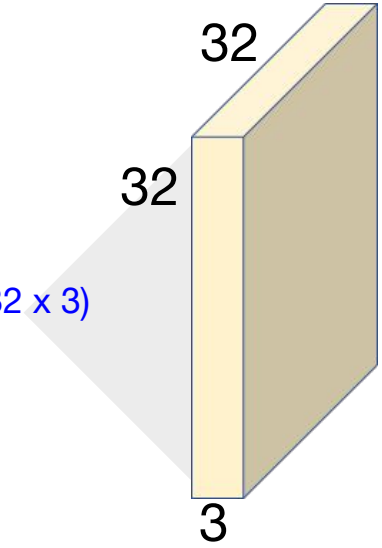
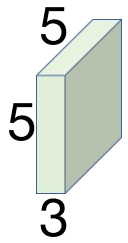


Image tensor **x**

Contains all the information in the image.

Convolutional layer filter.
Kernel? Stride?

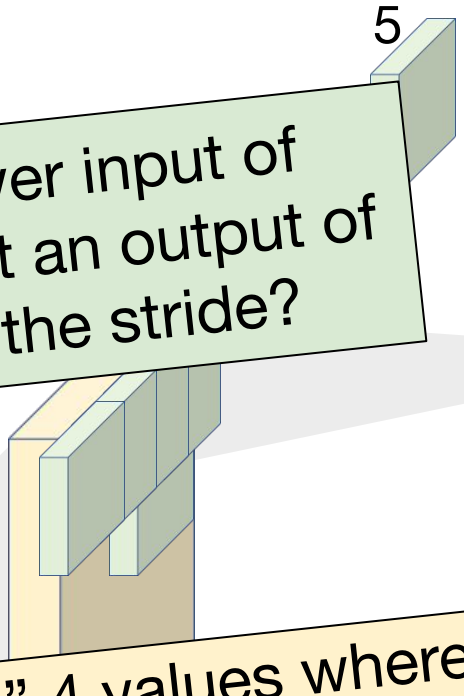
- 5x5x3 kernel
- Stride yet to be decided



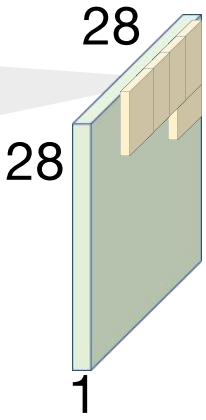
Convolutional L

As we slide 5x5x3 over input of size 32x32x3, we get an output of 28x28x1. So what's the stride?

$\phi_x($  $) = [0, 1]^{(32 \times 32 \times 3)}$



$f_c(\mathbf{x}_A) = W\mathbf{x}_A + \mathbf{b}$

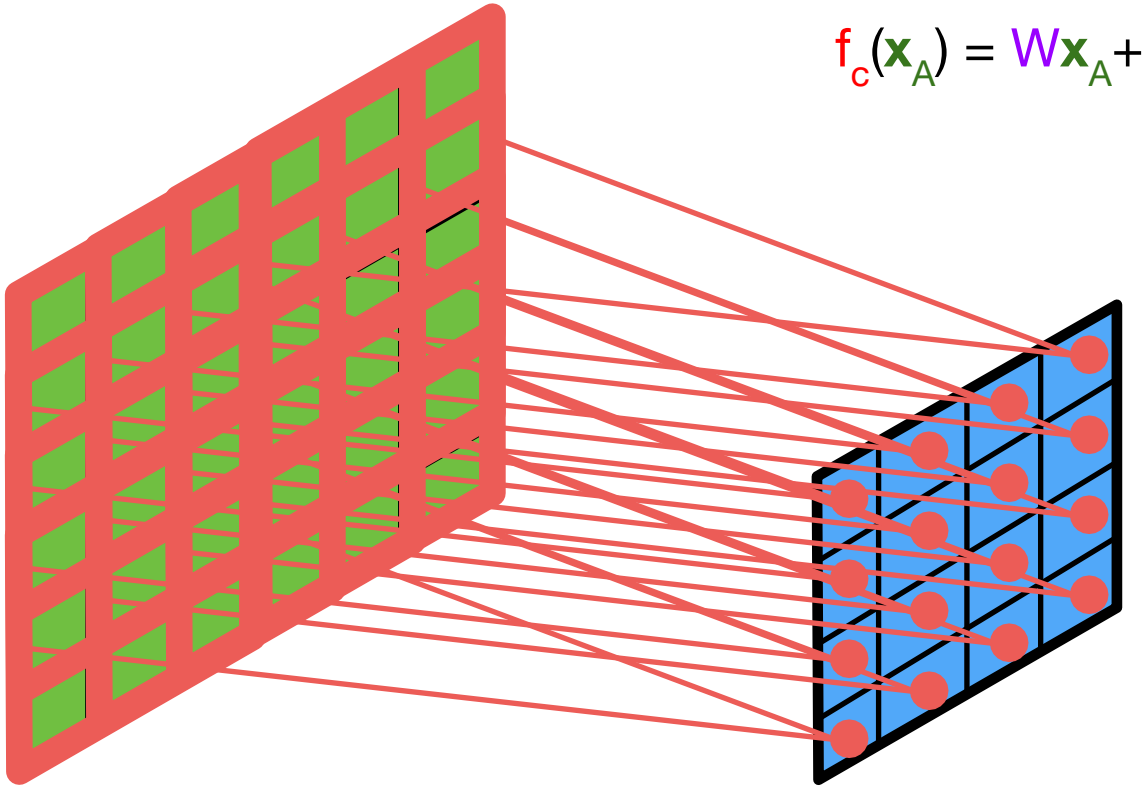


Stride=1. We "lose" 4 values where the filter would be hanging over the edge of the input in width/height with kernel 5x5x3

3 subset of image tensor ***x***
n f_c a learned, linear
r flattened $\boldsymbol{x}_A$
is 1x(5*5*3), ***b*** is size 1, so f_c is a single number!

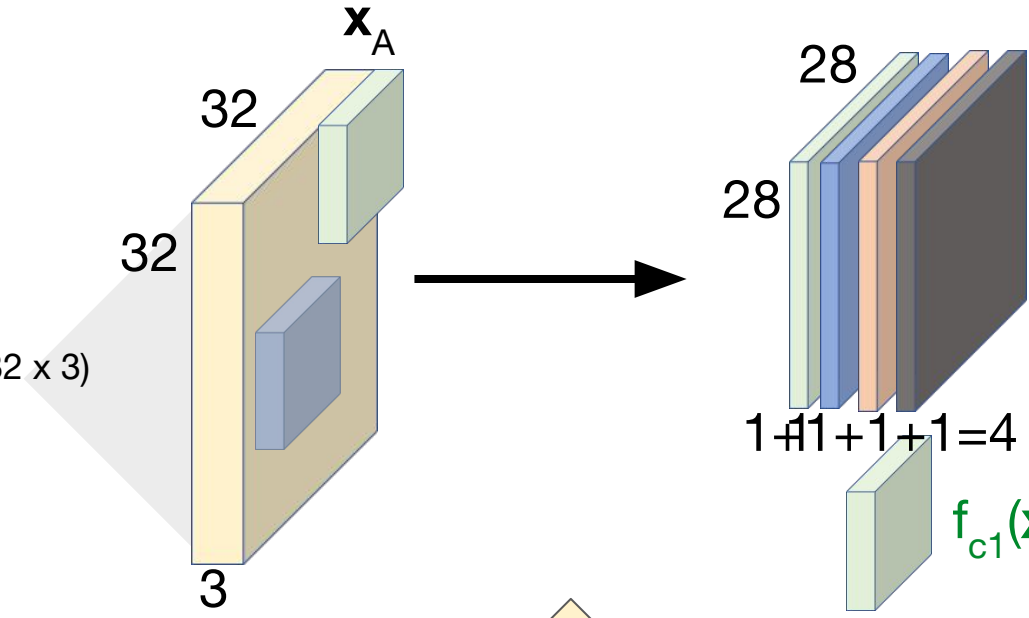
Convolutional Layer

$$f_c(\mathbf{x}_A) = W\mathbf{x}_A + \mathbf{b}$$



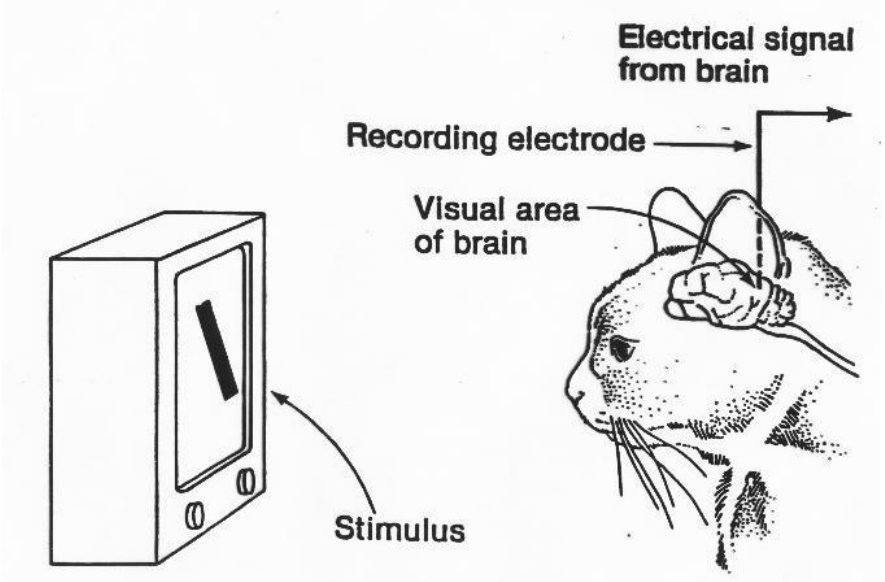
Convolutional Layer

$\phi_x($  $) = [0, 1]^{(32 \times 32 \times 3)}$



Convolutional Layer Filters

“They found that particular neurons in the visual cortexes of cats and monkeys—the areas in their brains responsible for processing visual information—didn't respond to simple points of light, but rather to lines, and in particular, lines and contours with specific orientations”



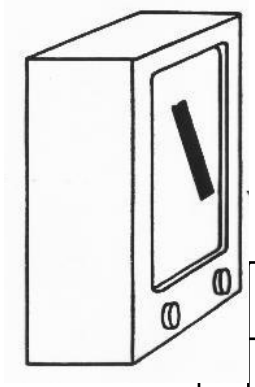
f_{c1}

0.7	1	0
0.3	1	0.3
0	1	0.7

$$f_{c1}(\mathbf{x}_A) = W_1\mathbf{x}_A + \mathbf{b}_1$$

https://hub.jhu.edu/2015/07/14/golden-goose-award-wiesel-hubel/

Convolutional Layer Filters



3

1

flatten

Learned CNN Filter

9

1 <0.7, 1, 0, 0.3, …> 9 * 1 + 1 1

$f_{c1}(\mathbf{x})=W_1$

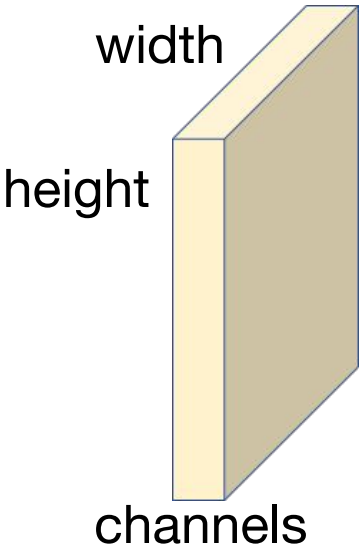
f_{c1}

0.7	1	0
0.3	1	0.3
0	1	0.7

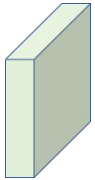
What kind of input creates a high response from this filter?

Convolutional Layer Terminology

3D Tensor

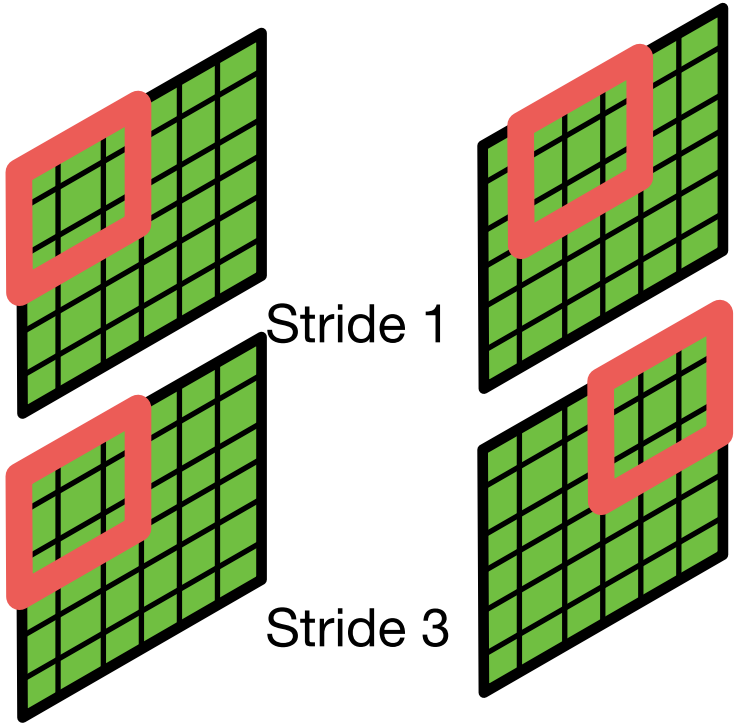


Kernel



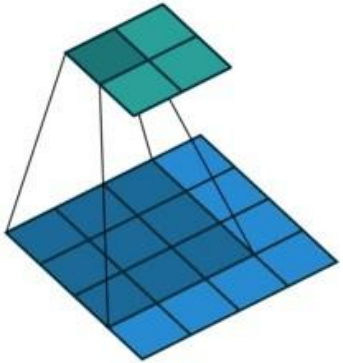
(width, height, channels)

Stride

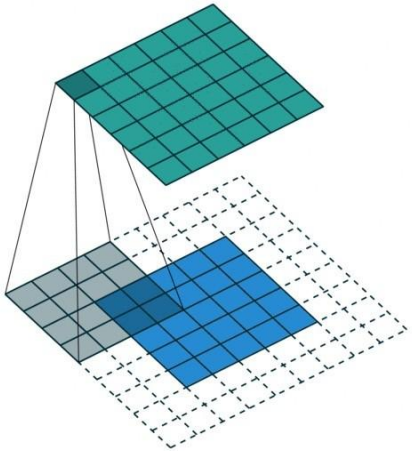


Convolutional Layer Terminology

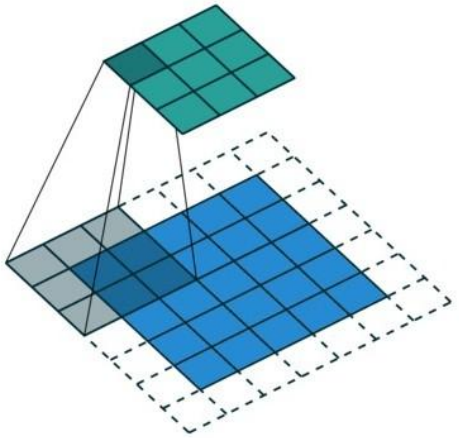
Padding and Stride



Pad=0; Stride=1
Kernel size?
3x3; lose 2 w/h



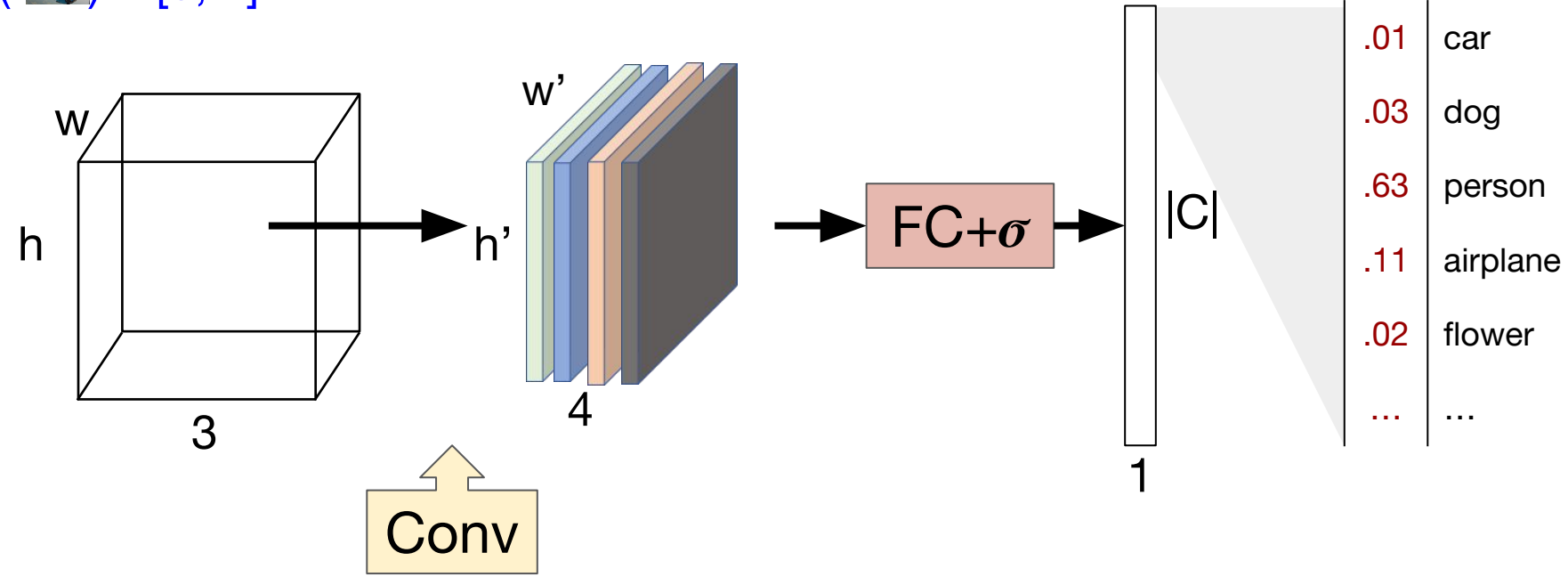
Pad=2; Stride=1
What to pad with?
Zeros common



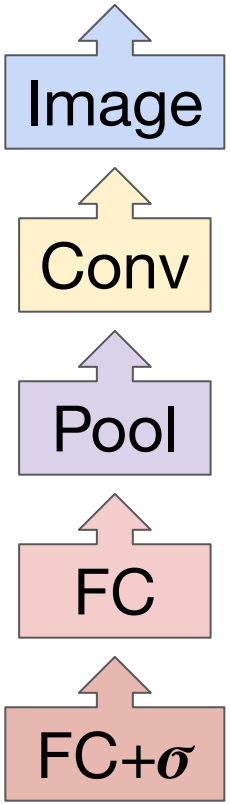
Pad=1; Stride=2
Effect of stride?
Sparser output

Image Classification with a Convolutional Layer

$\phi_x($  $) = [0, 1]^{(w \times h \times 3)}$

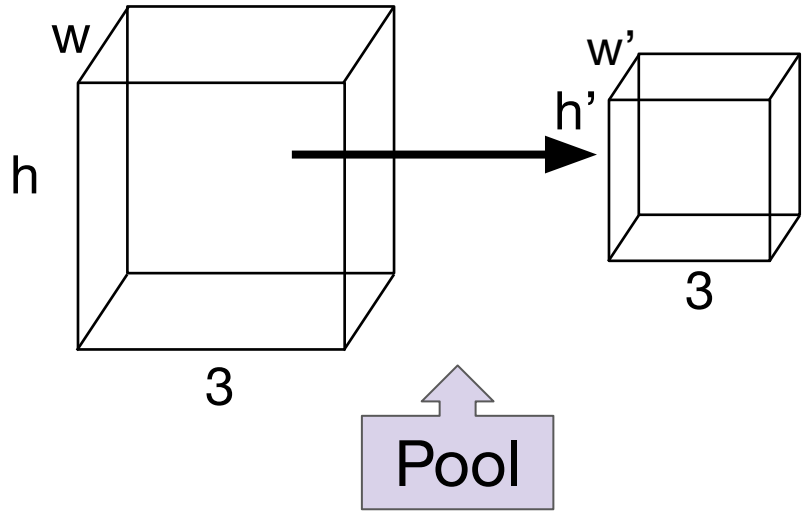


Common Layers in a Deep Convolutional Neural Net



- **x** in our parlance; the vectorized input
- Convolutional layer
- Pooling layer (in AlexNet, MaxPool)
- Fully-connected layer (i.e., *perceptron* layer)
- Softmax layer (fully connected layer + exponentiated normalization)

Pooling Layer

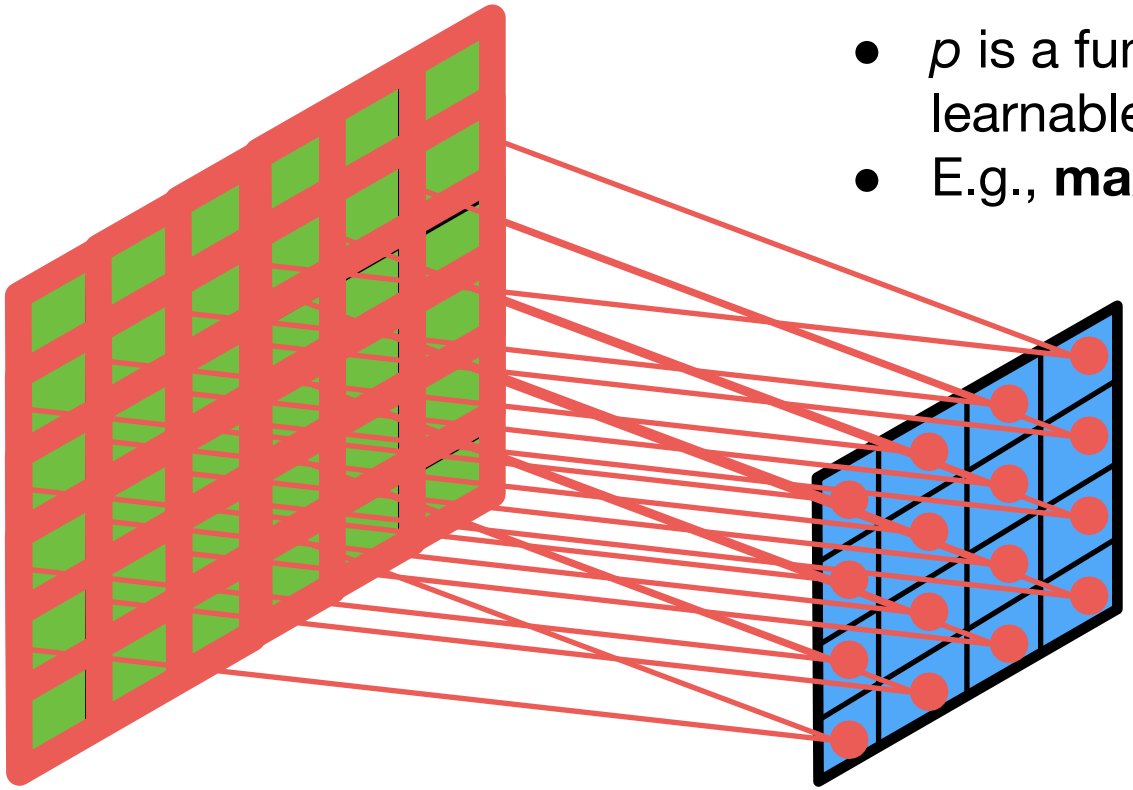


- A pooling layer shrinks a tensor along specified dimensions by removing information
- Condense information for a given neighborhood using an operator such as:
  - Max
  - Min
  - Average
  - L2-norm

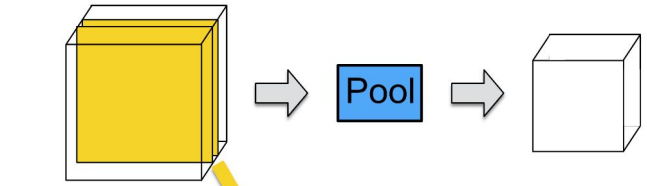
Pooling Layer

$f_p(\mathbf{x}_A) = p(\mathbf{x}_A)$

- *p* is a function with no learnable parameters
- E.g., **max**



Max Pooling Layer

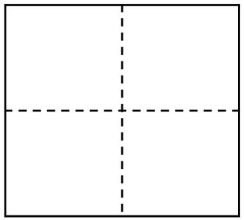


M x N x P

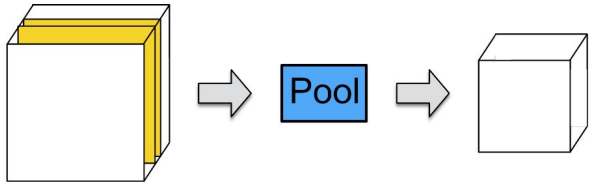
m x n x P

1	3	2	7	2
5	1	3	0	5
4	8	5	0	6
3	2	6	0	8
0	7	4	6	9

max pool with
3x3 filters & stride 2

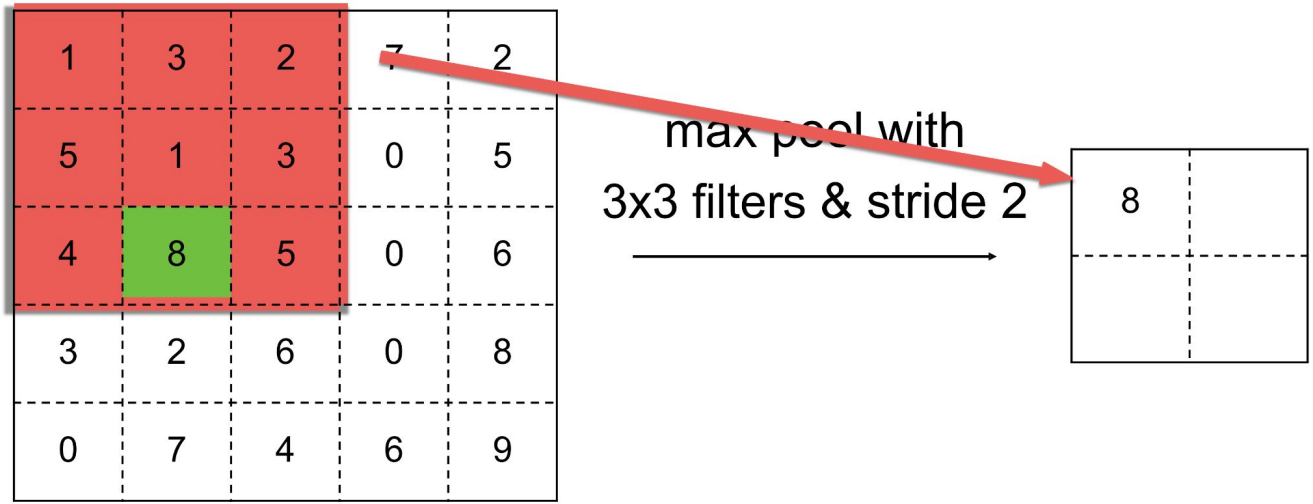


Max Pooling Layer

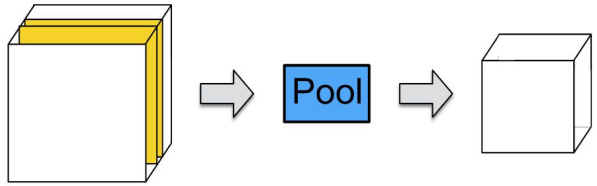


M x N x P

m x n x P

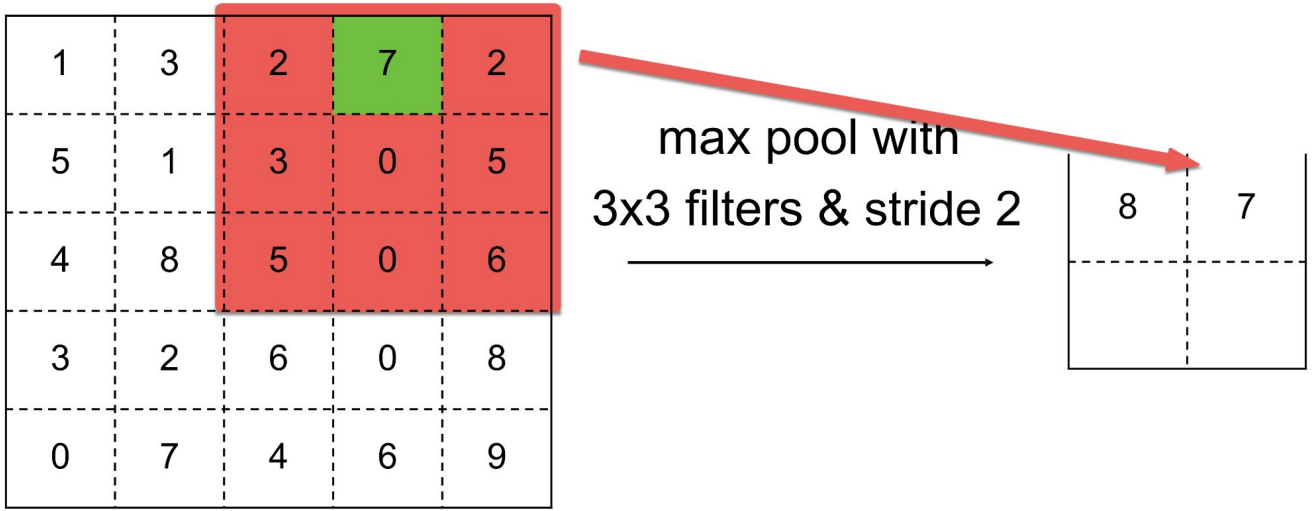


Max Pooling Layer

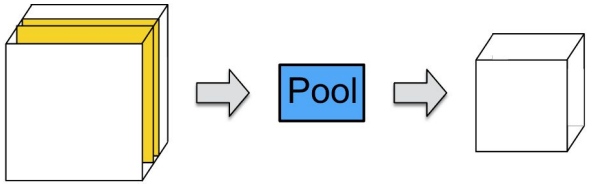


M x N x P

m x n x P

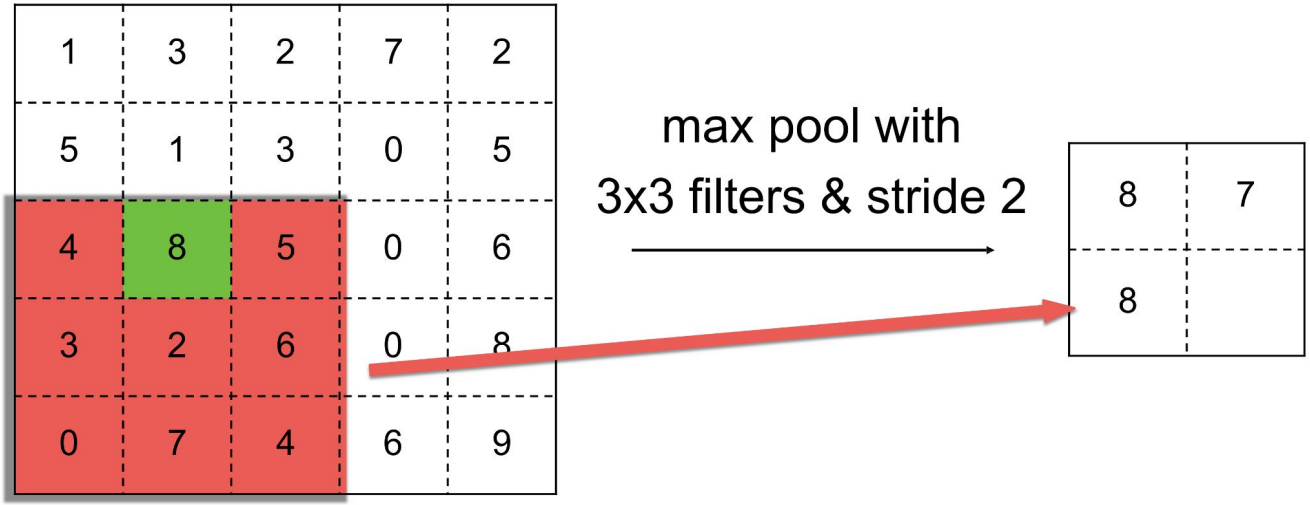


Max Pooling Layer

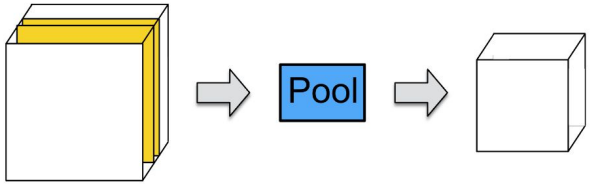


M x N x P

m x n x P



Max Pooling Layer



M x N x P

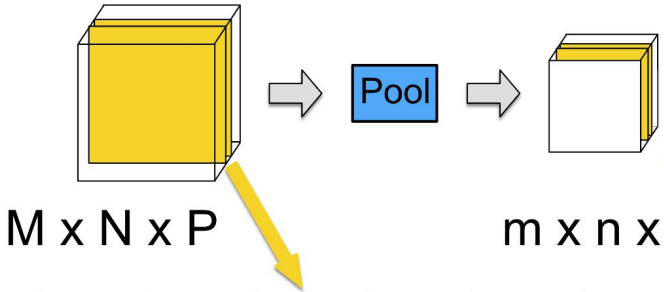
m x n x P

1	3	2	7	2
5	1	3	0	5
4	8	5	0	6
3	2	6	0	8
0	7	4	6	9

max pool with
3x3 filters & stride 2

8	7
8	9

Max Pooling Layer



1	3	2	7	2
5	1	3	0	5
4	8	5	0	6
3	2	6	0	8
0	7	4	6	9

max pool with
3x3 filters & stride 2

8	7
8	9

Image Classification with a Conv and Pool Layers

$\phi_x($  $) = [0, 1]^{(w \times h \times 3)}$

With These Layers, We Can Compose AlexNet!

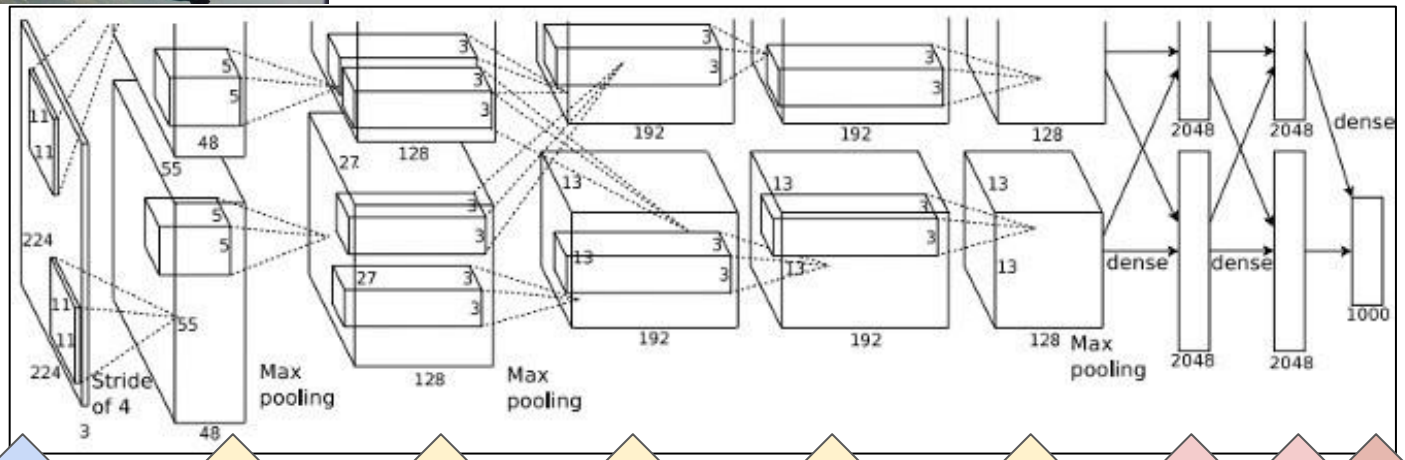
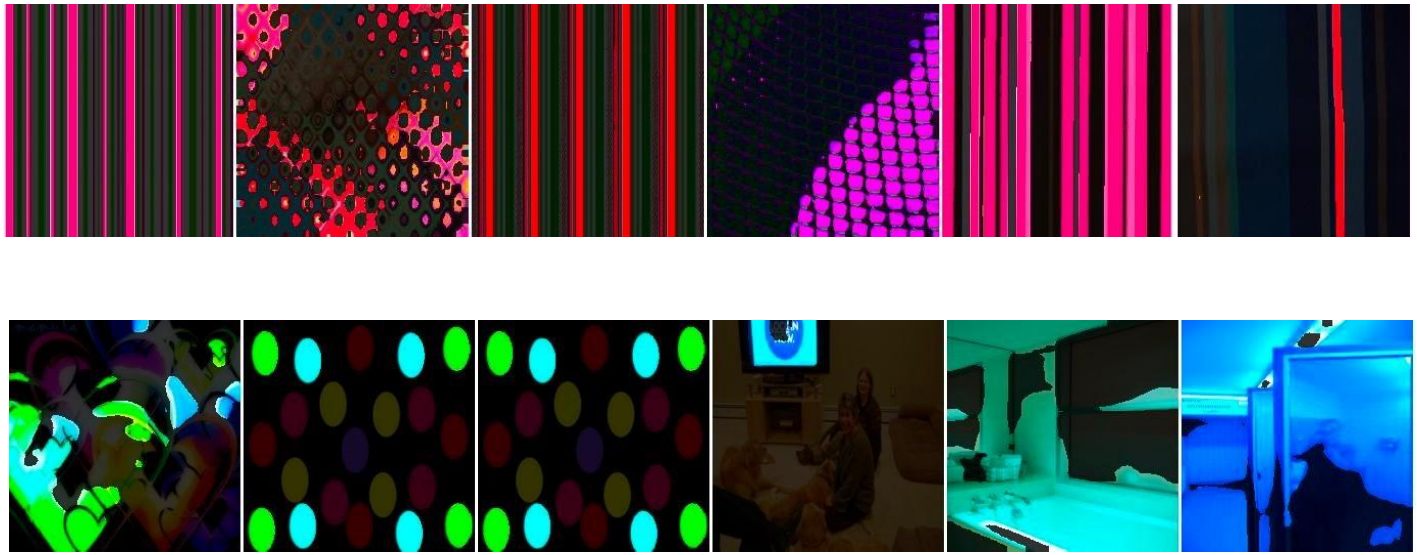


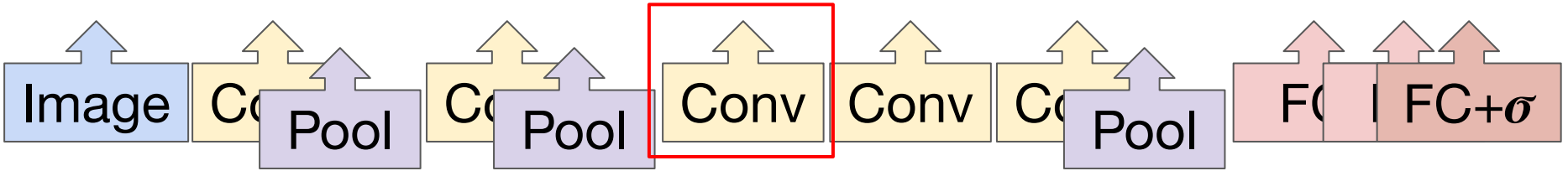
Image | Conv | Pool | Conv | Pool | Conv | Conv | Conv | Pool | FC | FC | FC+σ

What Does AlexNet Learn?



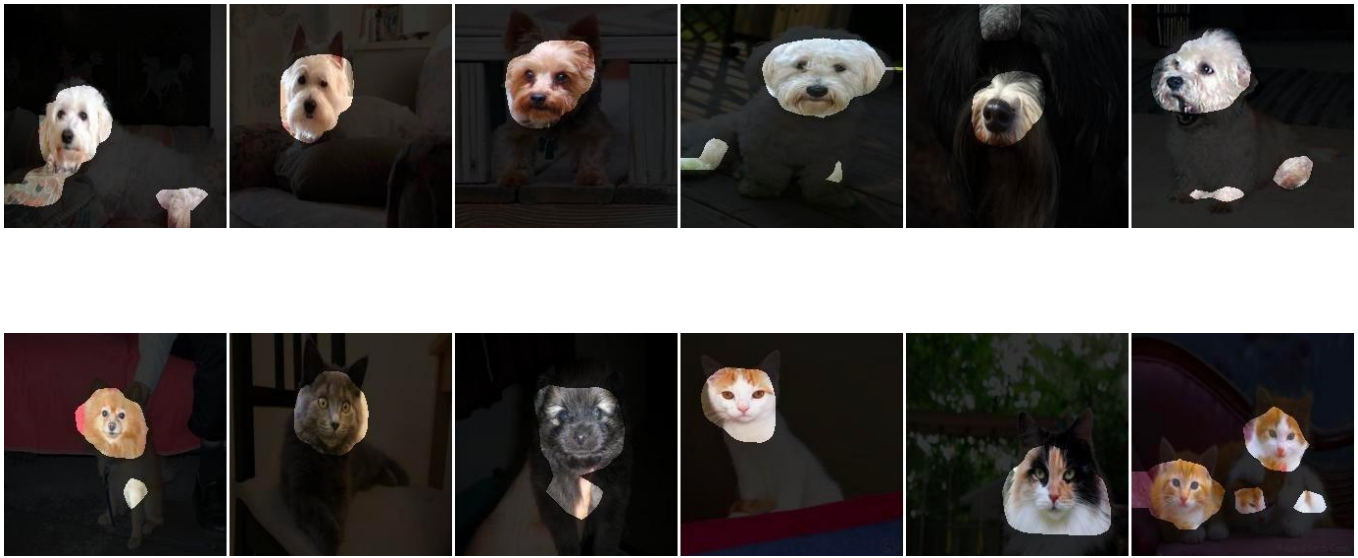
Network Dissection: Quantifying Interpretability of Deep Visual Representations, CVPR'17

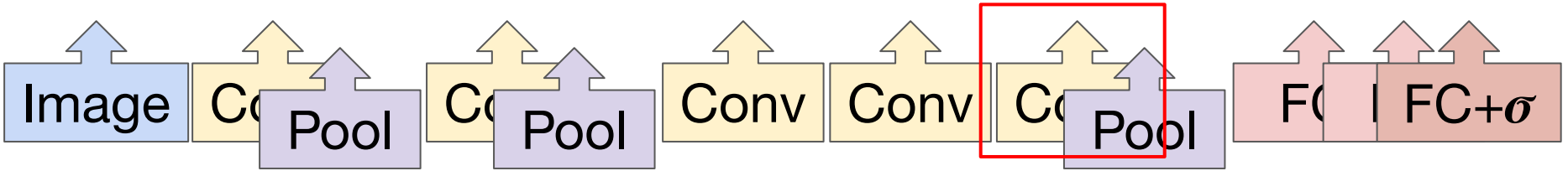
What Does AlexNet Learn?



Network Dissection: Quantifying Interpretability of Deep Visual Representations, CVPR'17

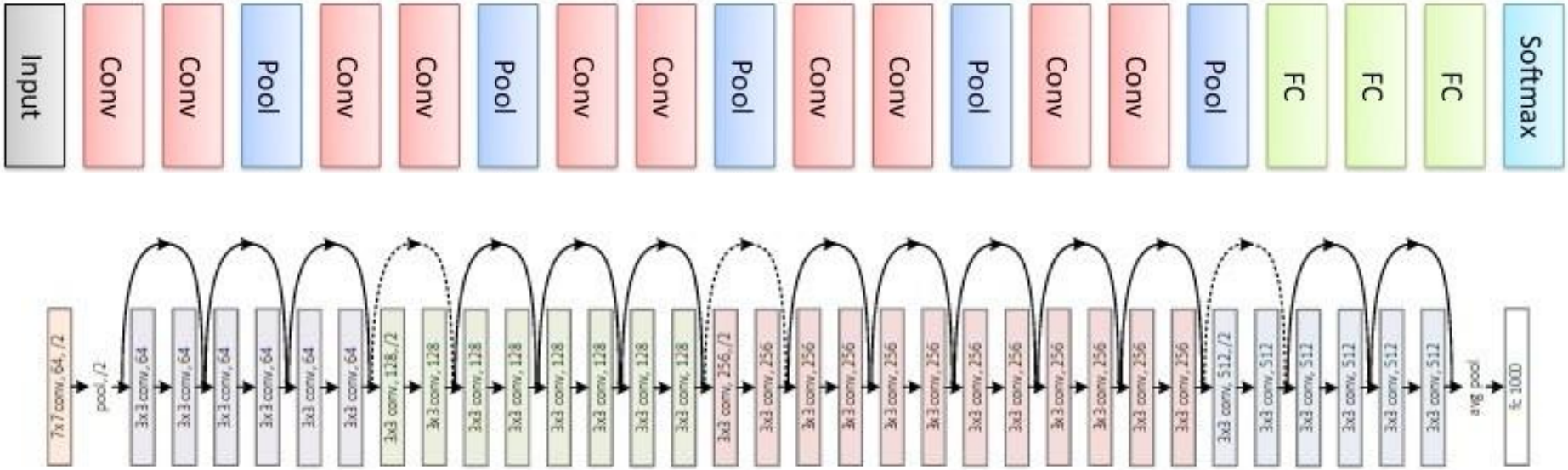
What Does AlexNet Learn?





Network Dissection: Quantifying Interpretability of Deep Visual Representations, CVPR'17

Other Formerly Popular Image Classification Architectures



[Very Deep Convolutional Networks for Large-Scale Image Recognition]
[Deep Residual Learning for Image Recognition]

Overview of Today's Plan

- ~~Course organization and deliverables~~
- ~~Training Neural Networks~~
- ~~Convolutional Neural Networks~~
    - Any questions before we move on?

Action Items for You

- If you have gotten D-clearance but have not taken Quiz 0, please take it; it is just a survey for us to get a sense of the student body, but it is worth points!
- Form project teams; these must be registered **today**
- Start brainstorming project topics; take a look at the deck of project ideas to get an idea what info you'll need if your team scopes a novel proposal

CSCI 566: Deep Learning and Its Applications

Jesse Thomason

Lecture 3: Training NNs; CNNs