CSCI 599: Deep Learning and its Applications

Lecture 4

Spring 2019
Joseph J. Lim

Disclaimer

- This course is taught for the 2nd time @ USC. This course is 599, and thus an **experimental** course.
- The syllabus, course policy, and grading details **may change** over the semester (**check website!**)
- If you prefer a well-structured course, this is **NOT** a course for you, and I encourage you to take the course next year.
- But, it will be **fun** and **challenging**!

Office Hours

- Instructor OH @ RTH 402
  - Tuesday 2-3:30pm
  - This is NOT for homework related questions.
- TA OH @ SAL 125
  - Monday 5-6pm & Tuesday 12:30-1:30pm
  - Extra OH for assignment due & midterm weeks: Monday 4-5pm & Tuesday 11:30-12:30pm

CSCI 599/699

Communication

- Please use **Piazza** for any general communication including questions
  https://piazza.com/usc/spring2019/csci599/home
- E-mails will be ignored.
- **Register TODAY. Look for your project team mates!**

Assignment 1

- Assignment 1 will be released next week.

Important Dates

Subject to change!

- Entrance exam: 1/15
- Assignment 1: 2/19
- Midterm: 3/5
- Project meeting with Instructor #1: 3/6 — 3/8
- Assignment 2: 3/26
- Project meeting with Instructor #2: 4/1 — 4/3
- Project meeting with TA: 2 times (arranged later)
- Final presentation: 4/23 5-9:00pm **4 hours**

Course Project

Subject to change!

- Computational resource (**be conserv**
  $150 Google Cloud credit per student
  $125 Amazon AWS credit per student
- Tentative Schedule for Project
  - Week 5 (2/5): Course Project Team
  - Week 8 (2/26): Course Project Proposal
  - Week 13 (4/2): Mid-report
  - Week 16 (4/23): **Project Presentation** (5-9pm) + Report

Course Project

- Team-based project (4-5 students per team)
- Each team will have at least 1 dedicated TA
  - 2 Mandatory meetings with TA
  - 2 meetings with me
- Create your own problems (extra points)
  - **Talk and discuss** with your TAs and me!
  - In the worst case, we will give a project idea
    - Less fun, Less points!

Project Evaluation

- Creativity and difficulty of the problem setup
- Novelty of the approach
- Thoroughness of the experiments
- Quality of student's presentation, report, and meetings with TA/instructor

Extra credit for creating your own project (OK to discuss and get help from TAs)

Project

- Report: a blog post (rather than a paper format)
- Lots of “what if we fail” questions

Team Formation

- Start forming your team NOW!

Today's agenda

- Recap: Neural Networks & Loss function
- Optimization
- Convolutional Neural Networks
- Training Neural Networks

Today's agenda

- Recap: Neural Networks & Loss function
- Optimization
- Convolutional Neural Networks
- Training Neural Networks

It's matter of one function

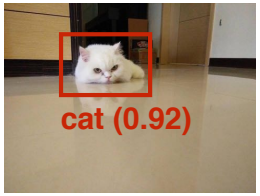
It's matter of one function

Really…?

Let's take a look

Object Detection

X: Image → f(X) → Y: Object label & Bounding box



It's matter of one function

X: Image → f(X) → Y: Description

A group of people shopping at an outdoor market.

There are many vegetables at the fruit stand.

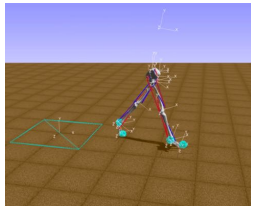
It's matter of one function

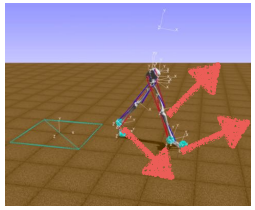
X: Description → f(X) → Y: Image

This flower has a lot of small round pink petals.

Learning to Walk

X: States → f(X) → Y: Action





Types of Learning

- Supervised Learning — desired output (**Y**) in training data
- Unsupervised Learning — **Y** not in training data
- Weakly / Semi-supervised Learning — some of **Y** in training data
- Reinforcement Learning — rewards based on a set of actions

Definition from Dhruv Batra's deep learning course (ECE 5604)

Our goal is "to approximate"

There may exist an exact function (f*) mapping from X to Y.

Our goal is not to find this exact function.

Rather, we are happy as long as f(X) can **approximate** f*(x). f does NOT have to be exactly f*.

Our goal is "to approximate"

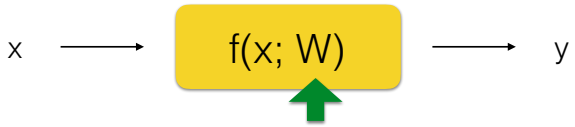
For f(X) to approximate any f*(X), f is better to be highly capable.

Deep learning is an effective method for this
- Non-linear (high capacity)
- Hierarchical
- End-to-End learning

Recap:Our course

(1) How do we learn this function (using deep learning)?
(2) How to formulate a problem into this

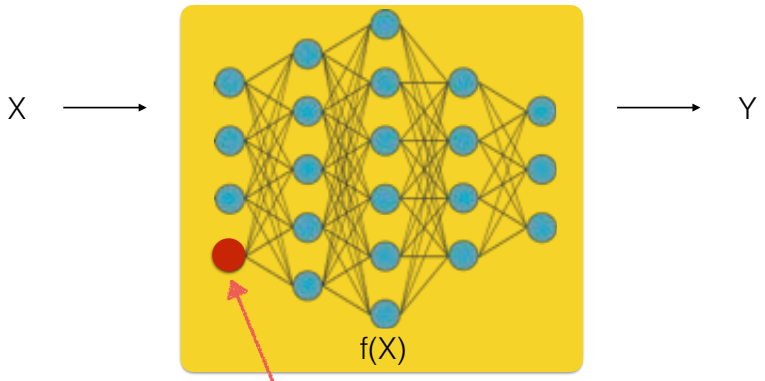
Loss function & Optimization



Showing W (to learn) explicitly
Parametric function

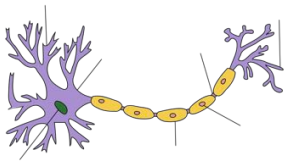
- **Loss function (L)** measures how well learned W can map X to Y (compared to f*).
- **Optimization** finds the best W given a loss function L (i.e. finding W that minimizes L).

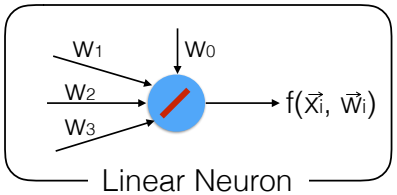
Neural Networks



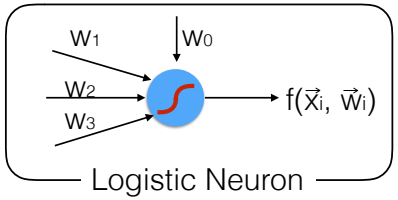
Each unit represents “neuron”

“Neuron”





Linear Neuron

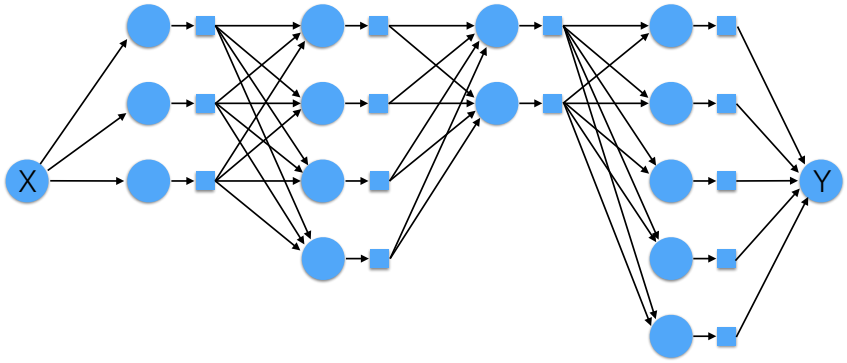


Logistic Neuron

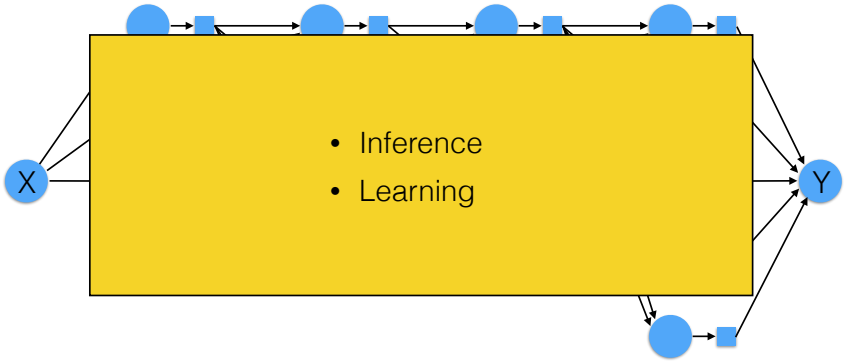
More neurons!

Modified from the HKUST slide

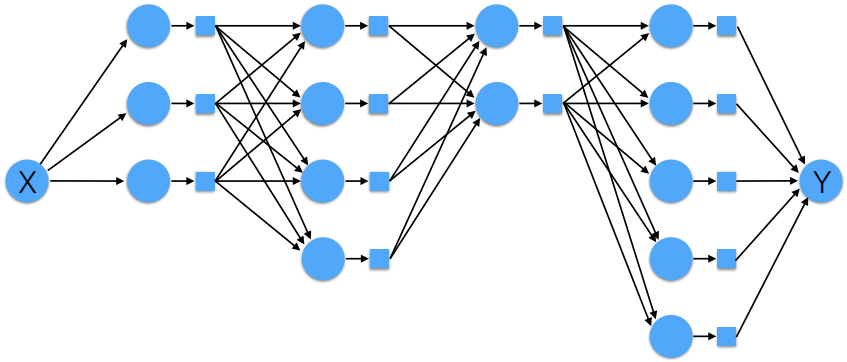
Neural Networks Example



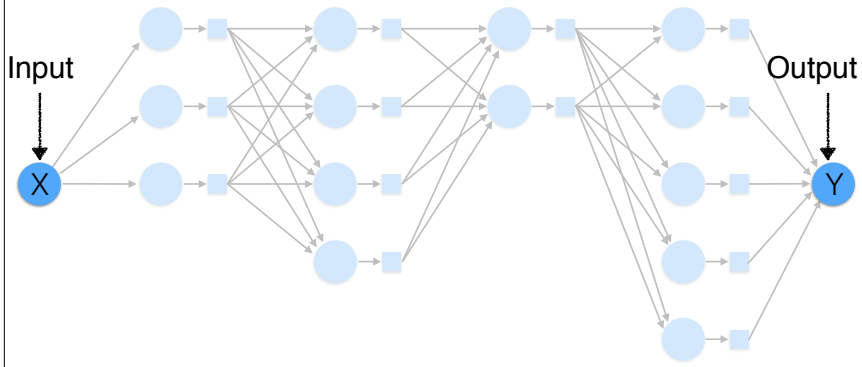
Neural Networks Example



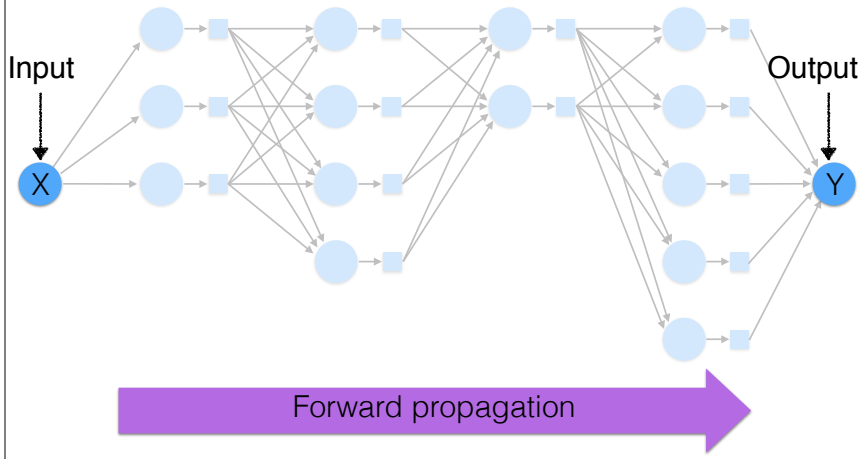
Forward propagation



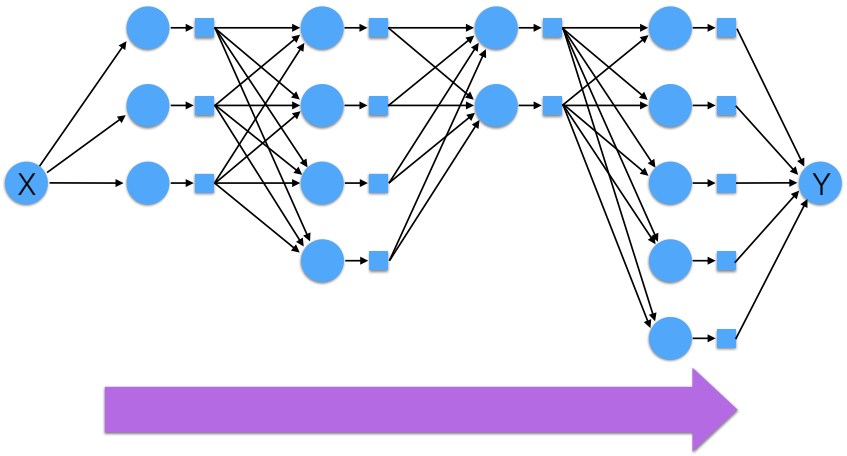
Forward propagation



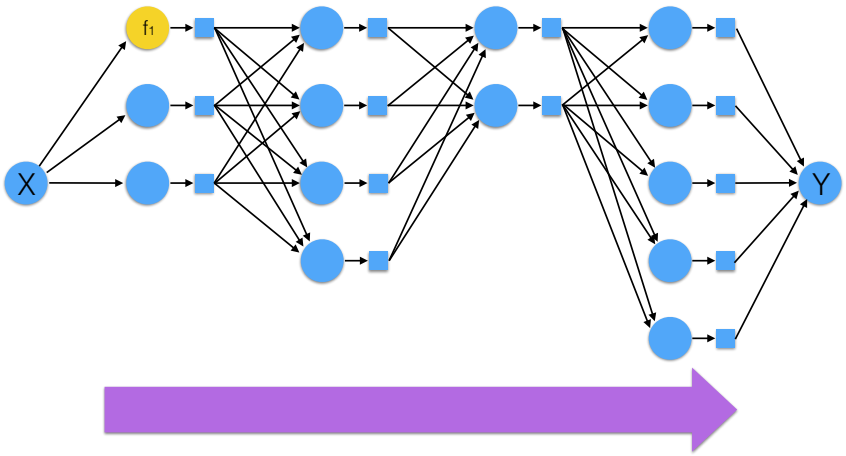
Forward propagation



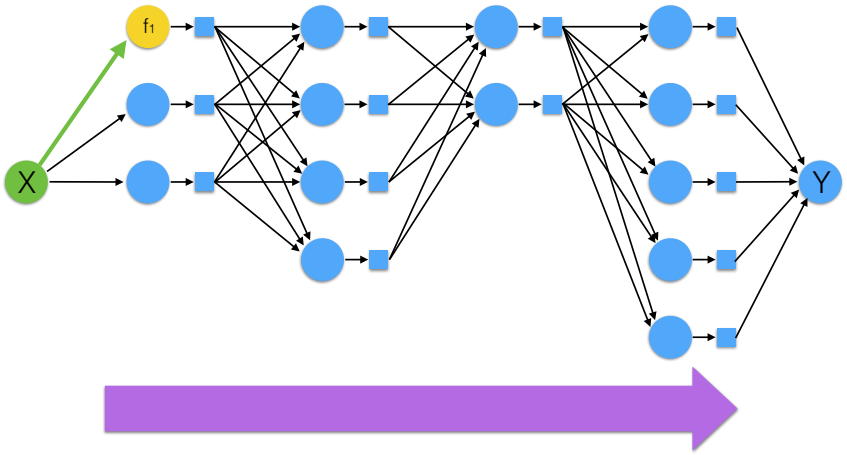
Forward propagation



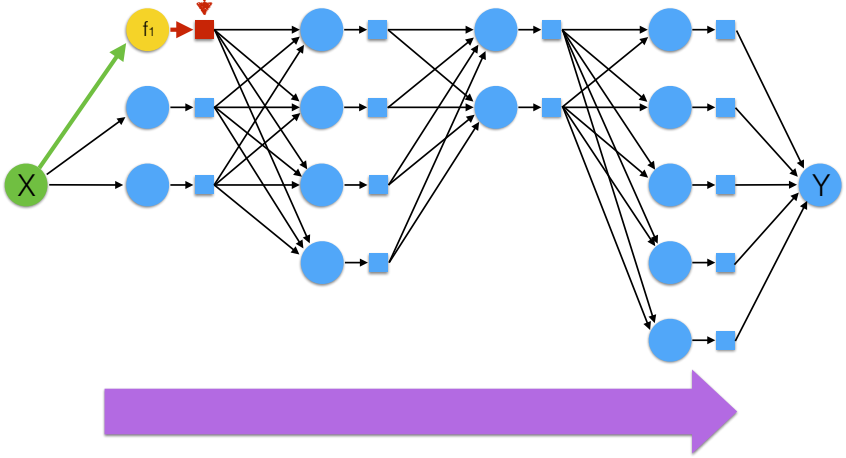
Forward propagation



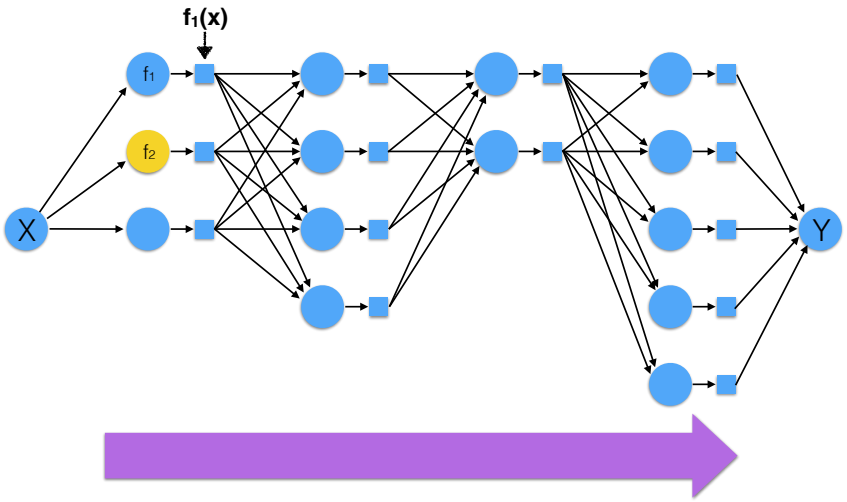
Forward propagation



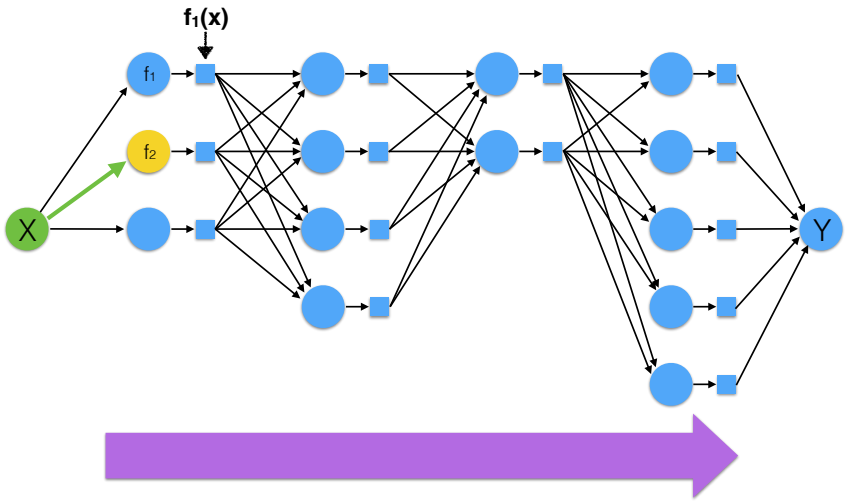
Forward propagation



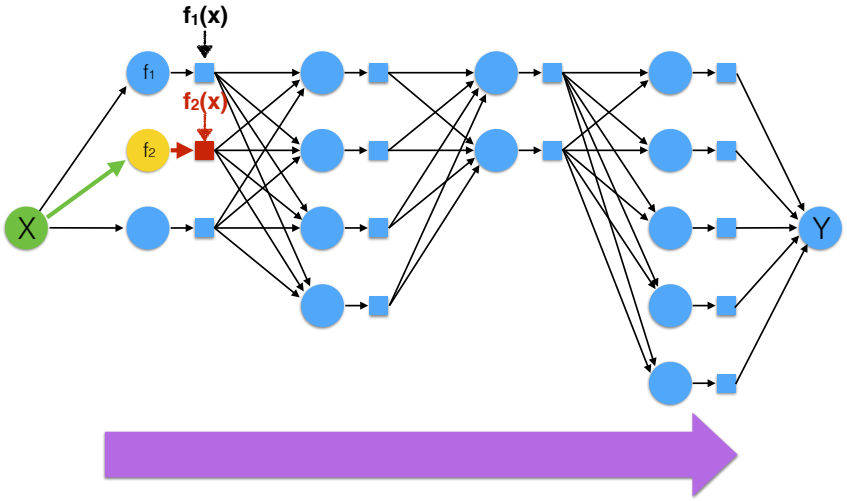
Forward propagation



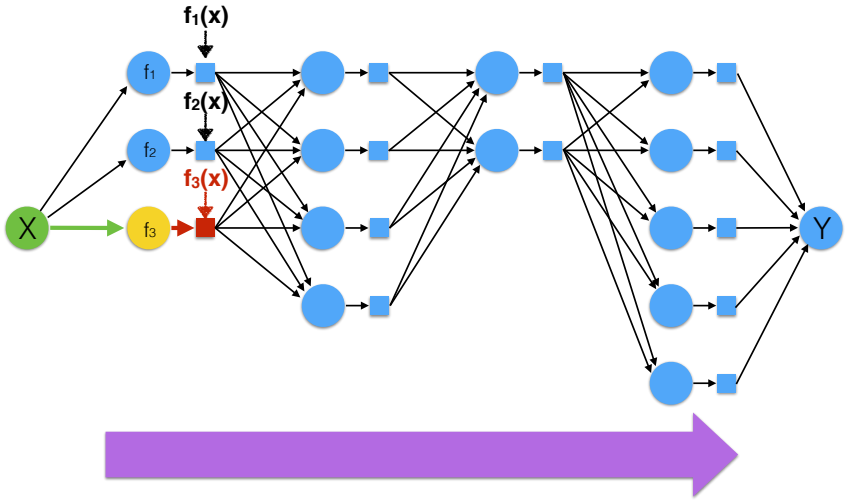
Forward propagation



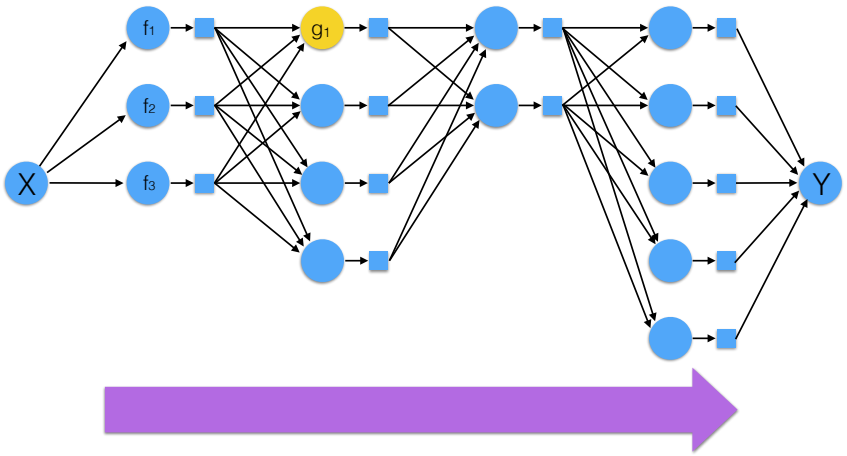
Forward propagation



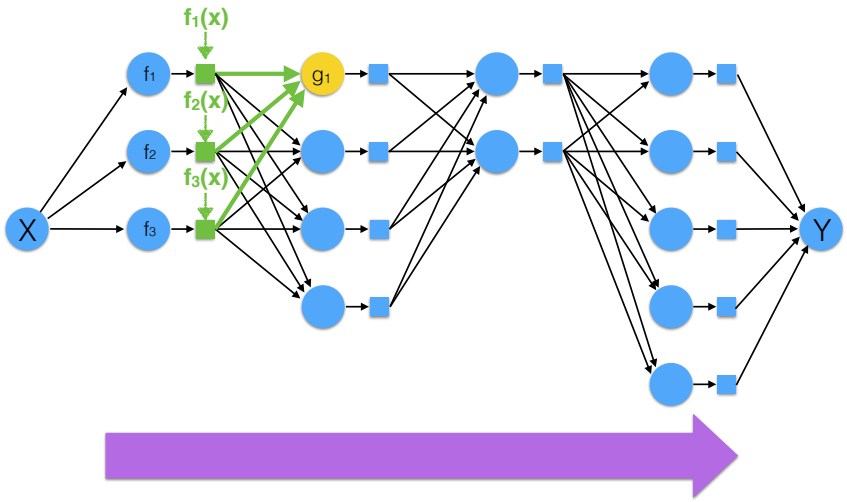
Forward propagation



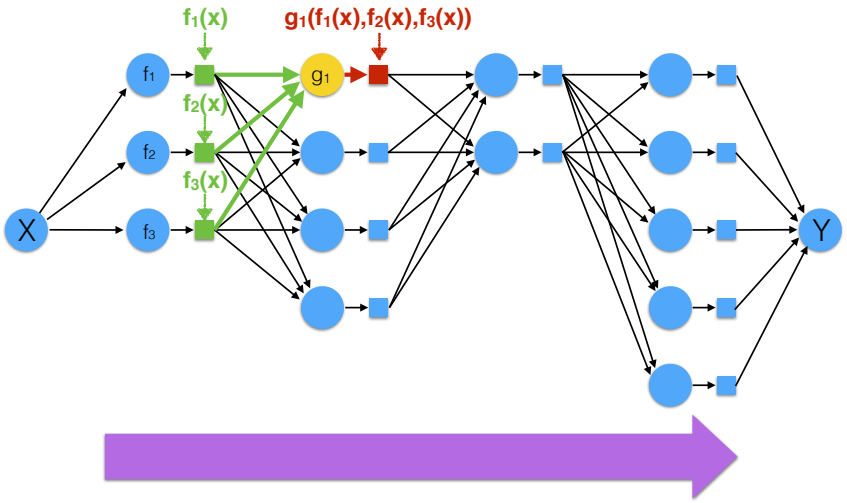
Forward propagation



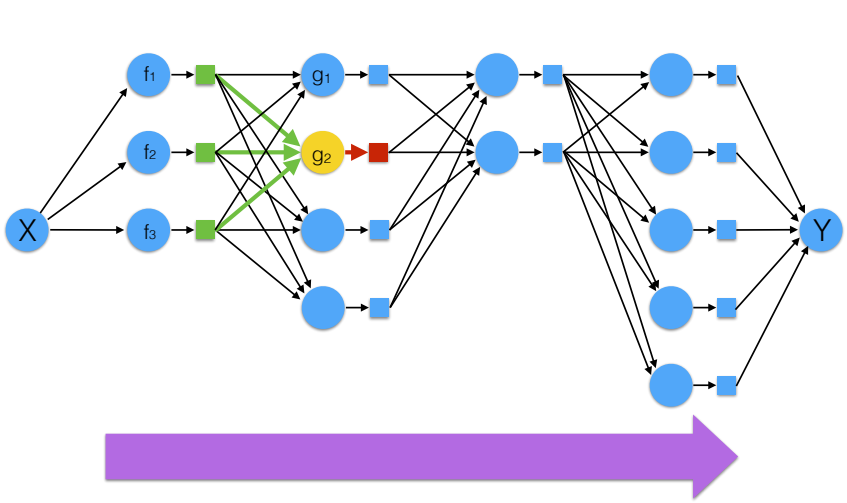
Forward propagation



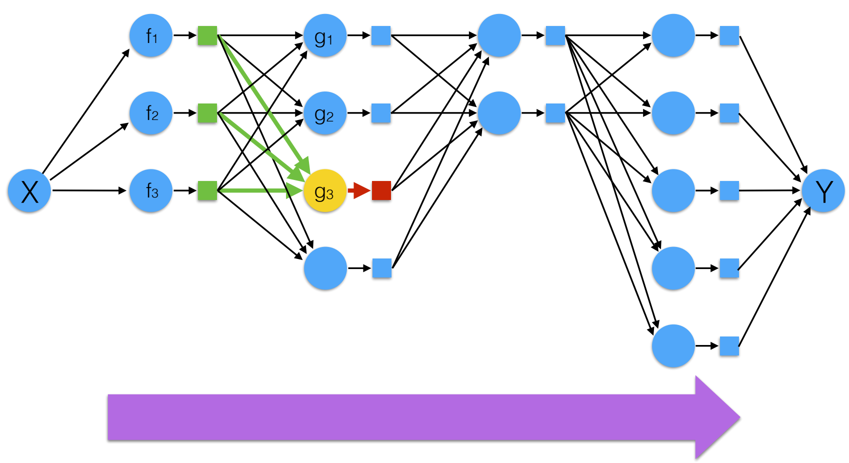
Forward propagation



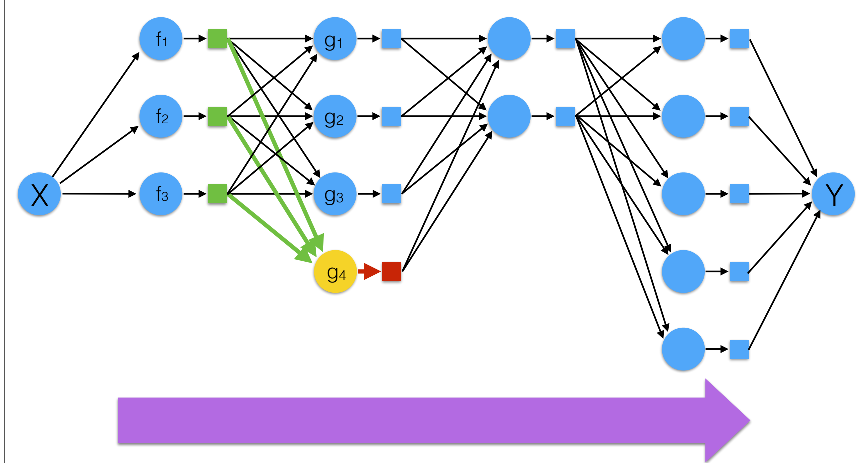
Forward propagation



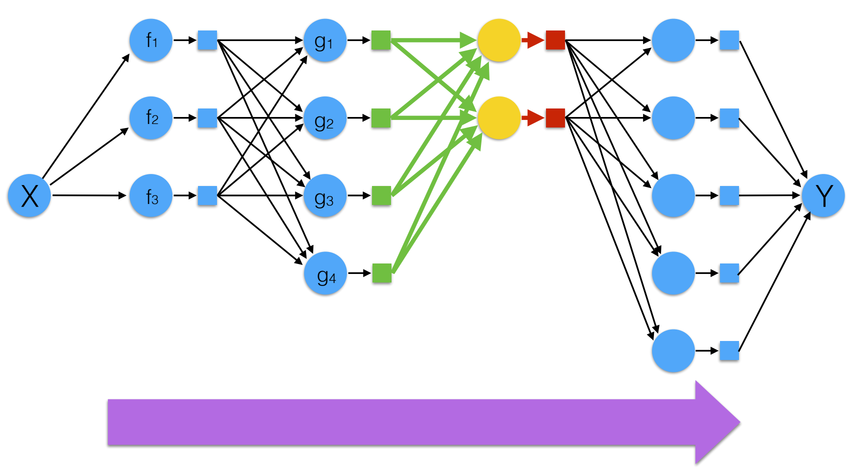
Forward propagation



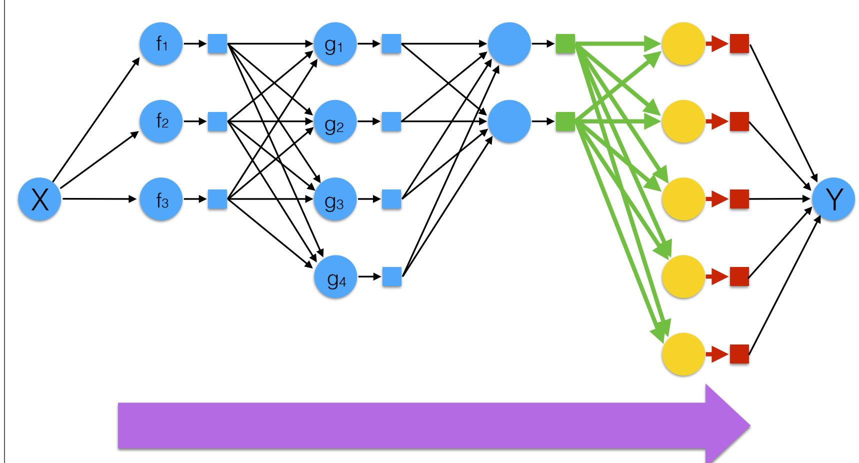
Forward propagation



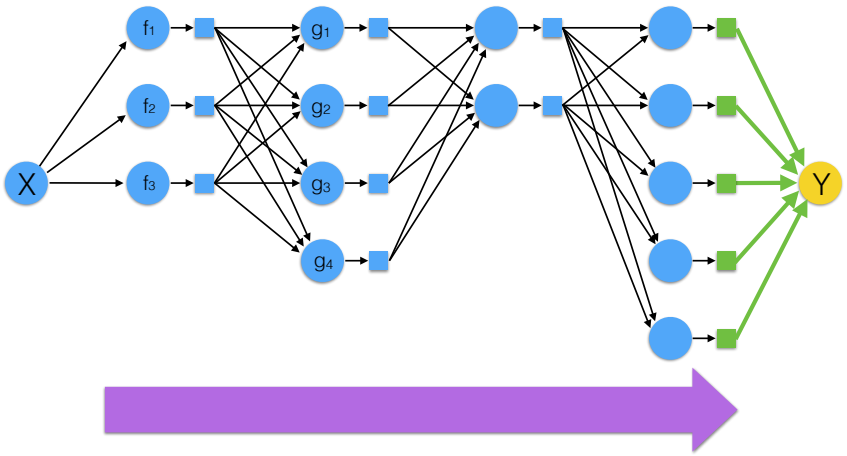
Forward propagation



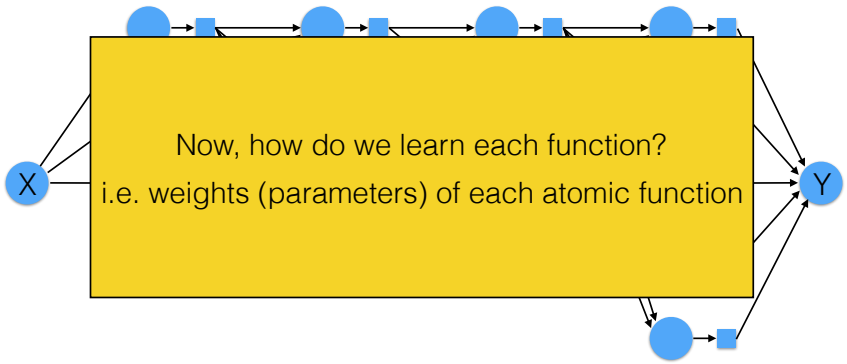
Forward propagation



Forward propagation



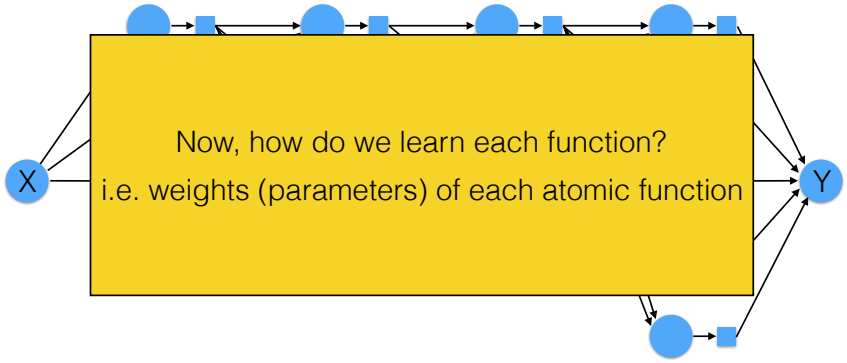
Neural Networks Example



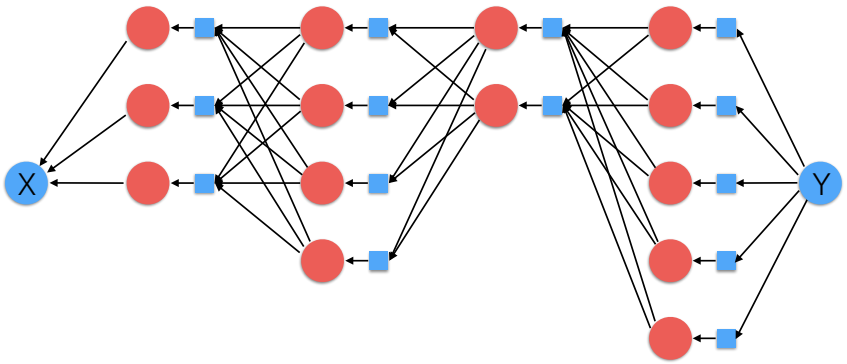
Today's agenda

- Recap: Neural Networks & Loss function
- Optimization
- Convolutional Neural Networks
- Training Neural Networks

Neural Networks Example



Backward propagation

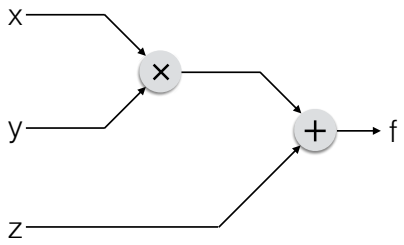


Computational Graph

Let's step back.

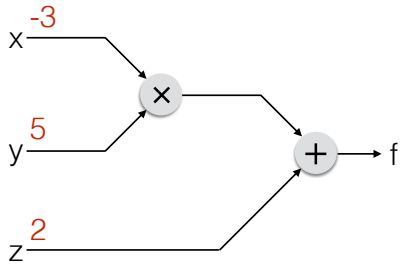
Computational Graph

Function: f(x, y, z) = x*y+z



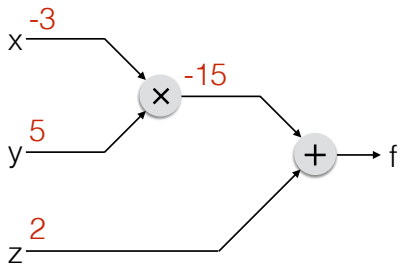
Computational Graph

Function: f(x, y, z) = x*y+z



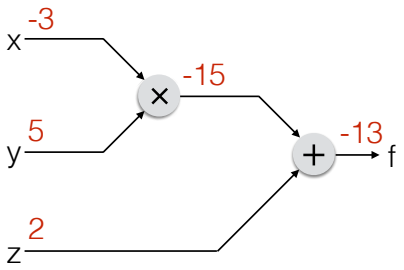
Computational Graph

Function: f(x, y, z) = x*y+z



Computational Graph

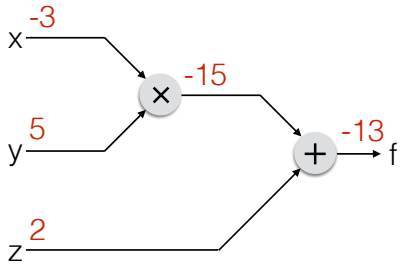
Function: f(x, y, z) = x*y+z



Computational Graph

Function: f(x, y, z) = x*y+z

Backpropagation



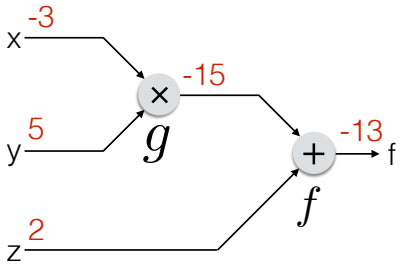
Computational Graph

Function: f(x, y, z) = x*y+z

Backpropagation

$g = x \times y$

$f = g + z$



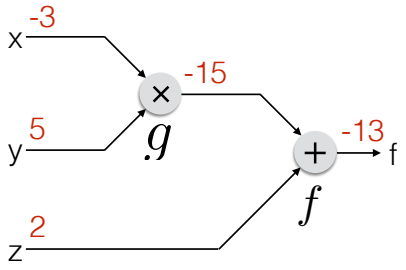
Computational Graph

Function: f(x, y, z) = x*y+z

Backpropagation

$g = x \times y \quad \frac{\partial q}{\partial x} = y \quad \frac{\partial q}{\partial y} = x$

$f = g + z \quad \frac{\partial f}{\partial g} = 1 \quad \frac{\partial f}{\partial z} = 1$



Computational Graph

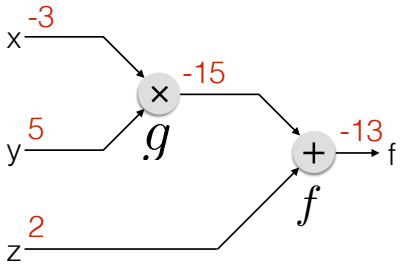
Function: f(x, y, z) = x*y+z

Backpropagation

$g = x \times y \quad \frac{\partial q}{\partial x} = y \quad \frac{\partial q}{\partial y} = x$

$f = g + z \quad \frac{\partial f}{\partial g} = 1 \quad \frac{\partial f}{\partial z} = 1$

We want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Computational Graph

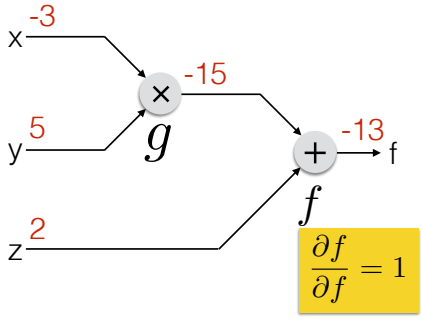
Function: f(x, y, z) = x*y+z

Backpropagation

$g = x \times y \quad \frac{\partial q}{\partial x} = y \quad \frac{\partial q}{\partial y} = x$

$f = g + z \quad \frac{\partial f}{\partial g} = 1 \quad \frac{\partial f}{\partial z} = 1$

We want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Computational Graph

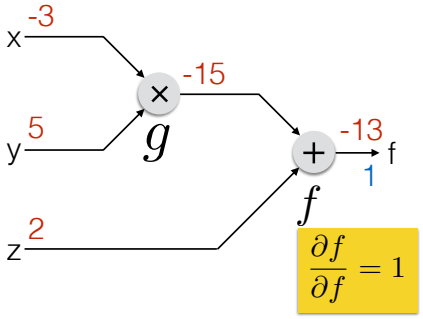
Function: f(x, y, z) = x*y+z

Backpropagation

$g = x \times y \quad \frac{\partial q}{\partial x} = y \quad \frac{\partial q}{\partial y} = x$

$f = g + z \quad \frac{\partial f}{\partial g} = 1 \quad \frac{\partial f}{\partial z} = 1$

We want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Computational Graph

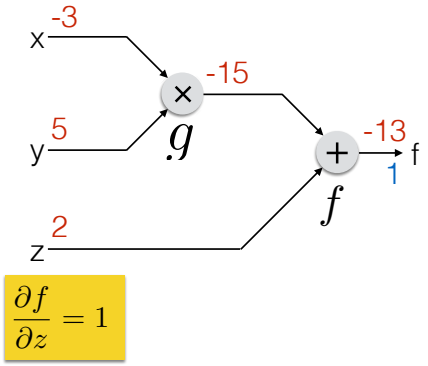
Function: f(x, y, z) = x*y+z

Backpropagation

$g = x \times y$ $\quad \frac{\partial q}{\partial x} = y \quad \frac{\partial q}{\partial y} = x$

$f = g + z$ $\quad \frac{\partial f}{\partial g} = 1 \quad \frac{\partial f}{\partial z} = 1$

We want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Computational Graph

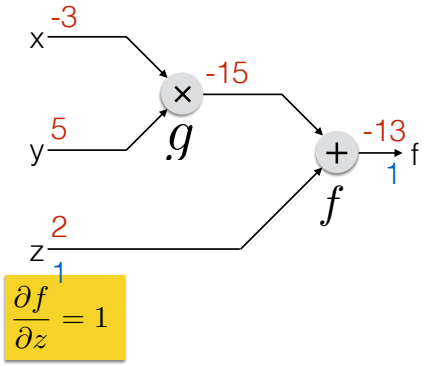
Function: f(x, y, z) = x*y+z

Backpropagation

$g = x \times y$ $\quad \frac{\partial q}{\partial x} = y \quad \frac{\partial q}{\partial y} = x$

$f = g + z$ $\quad \frac{\partial f}{\partial g} = 1 \quad \frac{\partial f}{\partial z} = 1$

We want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Computational Graph

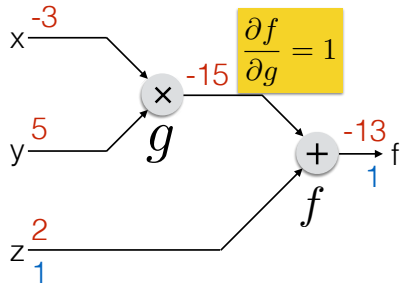
Function: f(x, y, z) = x*y+z

Backpropagation

$g = x \times y$ $\quad \frac{\partial q}{\partial x} = y \quad \frac{\partial q}{\partial y} = x$

$f = g + z$ $\quad \frac{\partial f}{\partial g} = 1 \quad \frac{\partial f}{\partial z} = 1$

We want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Computational Graph

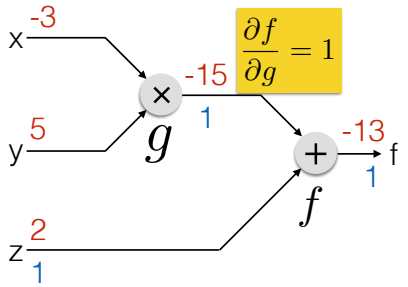
Function: f(x, y, z) = x*y+z

Backpropagation

$g = x \times y$ $\quad \frac{\partial q}{\partial x} = y \quad \frac{\partial q}{\partial y} = x$

$f = g + z$ $\quad \frac{\partial f}{\partial g} = 1 \quad \frac{\partial f}{\partial z} = 1$

We want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Computational Graph

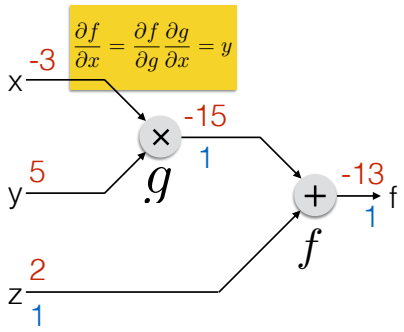
Function: f(x, y, z) = x*y+z

Backpropagation

$g = x \times y \quad \frac{\partial q}{\partial x} = y \quad \frac{\partial q}{\partial y} = x$

$f = g + z \quad \frac{\partial f}{\partial g} = 1 \quad \frac{\partial f}{\partial z} = 1$

We want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Computational Graph

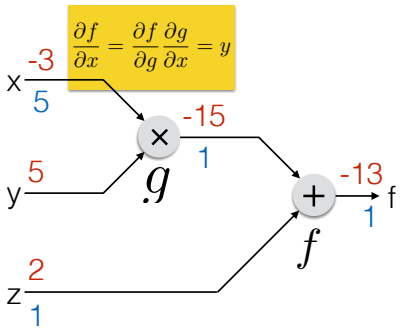
Function: f(x, y, z) = x*y+z

Backpropagation

$g = x \times y \quad \frac{\partial q}{\partial x} = y \quad \frac{\partial q}{\partial y} = x$

$f = g + z \quad \frac{\partial f}{\partial g} = 1 \quad \frac{\partial f}{\partial z} = 1$

We want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Joseph J. Lim CSCI 599 @ USC Lecture 4

Computational Graph

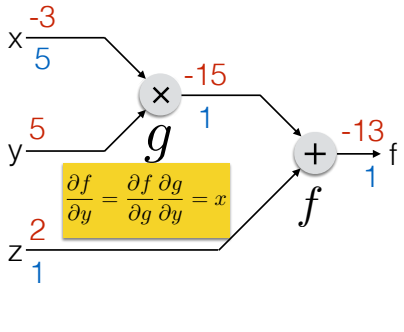
Function: f(x, y, z) = x*y+z

Backpropagation

$g = x \times y \quad \frac{\partial q}{\partial x} = y \quad \frac{\partial q}{\partial y} = x$

$f = g + z \quad \frac{\partial f}{\partial g} = 1 \quad \frac{\partial f}{\partial z} = 1$

We want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Joseph J. Lim CSCI 599 @ USC Lecture 4

Computational Graph

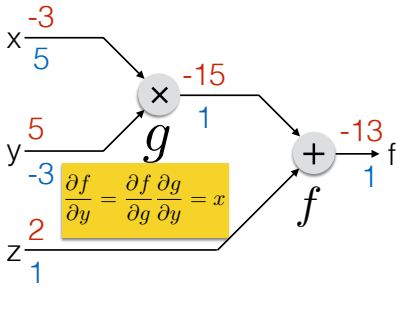
Function: f(x, y, z) = x*y+z

Backpropagation

$g = x \times y \quad \frac{\partial q}{\partial x} = y \quad \frac{\partial q}{\partial y} = x$

$f = g + z \quad \frac{\partial f}{\partial g} = 1 \quad \frac{\partial f}{\partial z} = 1$

We want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Joseph J. Lim CSCI 599 @ USC Lecture 4

Computational Graph

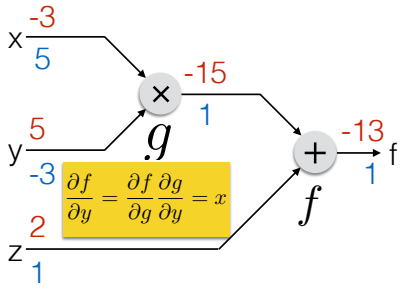
Function: f(x, y, z) = x*y+z

Backpropagation

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

We get: $\frac{\partial f}{\partial y} = -3$

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



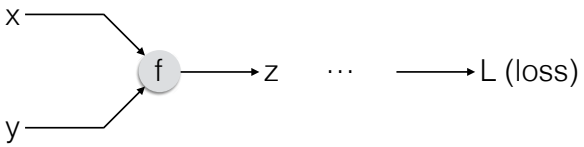
Computational Graph

Note: a local view of it

Computational Graph

Function: f(x, y, z) = x*y+z

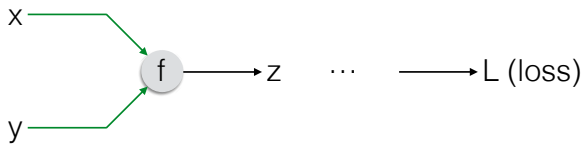
A local view of backpropagation
Forward propagation



Computational Graph

Function: f(x, y, z) = x*y+z

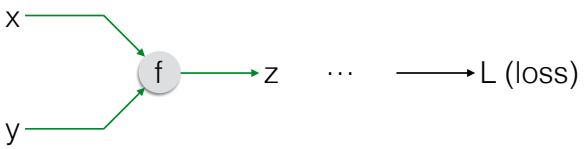
A local view of backpropagation
Forward propagation



Computational Graph

Function: f(x, y, z) = x*y+z

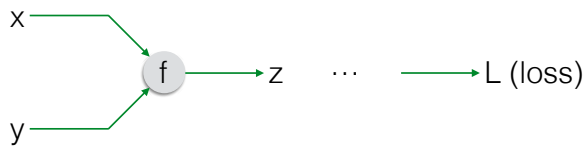
A local view of backpropagation
Forward propagation



Computational Graph

Function: f(x, y, z) = x*y+z

A local view of backpropagation
Forward propagation

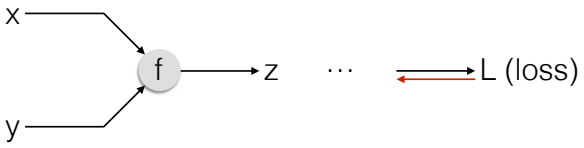


Joseph J. Lim CSCI 599 @ USC Lecture 4

Computational Graph

Function: f(x, y, z) = x*y+z

A local view of backpropagation
Backpropagation

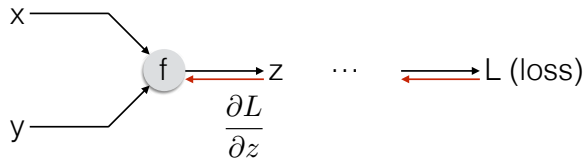


Joseph J. Lim CSCI 599 @ USC Lecture 4

Computational Graph

Function: f(x, y, z) = x*y+z

A local view of backpropagation
Backpropagation

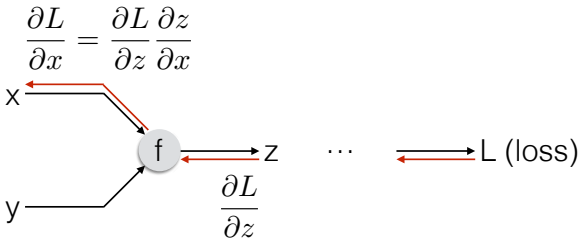


Joseph J. Lim CSCI 599 @ USC Lecture 4

Computational Graph

Function: f(x, y, z) = x*y+z

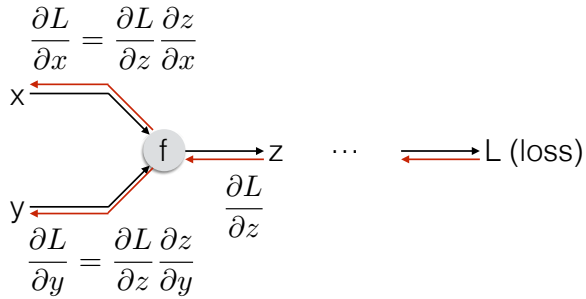
A local view of backpropagation
Backpropagation



Computational Graph

Function: f(x, y, z) = x*y+z

A local view of backpropagation
Backpropagation

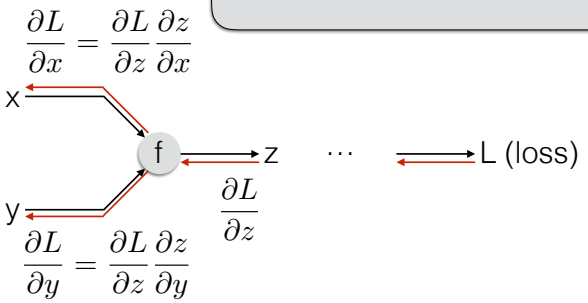


Computational Graph

Function: f(x, y, z) = x*y+z

A local view of backpropa
Backpropagation

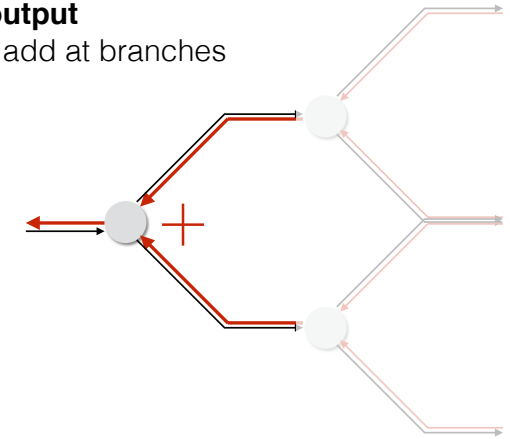
Requires a function to be differentiable



Computational Graph

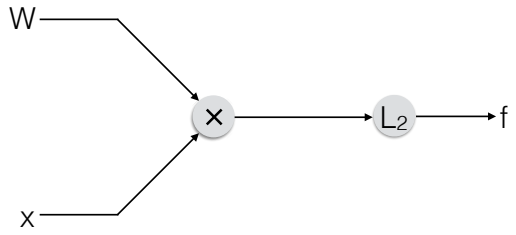
Function: f(x, y, z) = x*y+z

Multiple output
Gradients add at branches



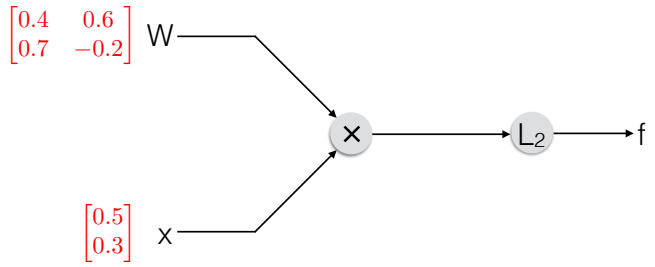
Computational Graph

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



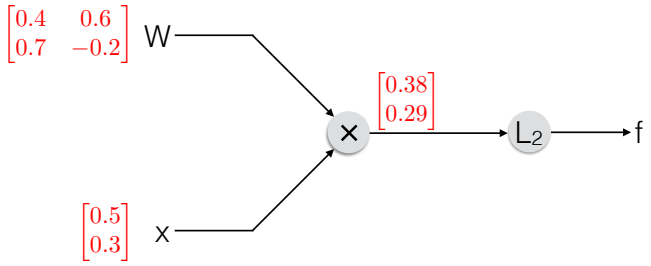
Computational Graph

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



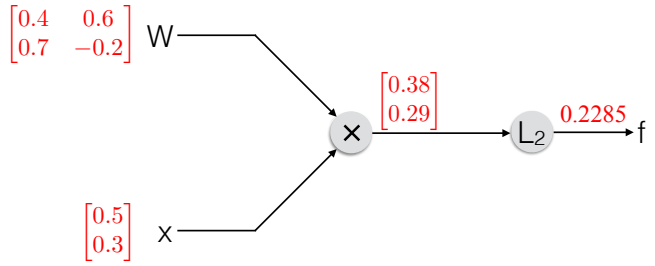
Computational Graph

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



Computational Graph

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



Computational Graph

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

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

Computational Graph

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

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

$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + ... + W_{1,n}x_n \\ ... \\ W_{n,1}x_1 + ... + W_{n,n}x_n \end{pmatrix}$$

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

Computational Graph

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

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

$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + ... + W_{1,n}x_n \\ ... \\ W_{n,1}x_1 + ... + W_{n,n}x_n \end{pmatrix}$$

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

Computational Graph

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

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

$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + ... + W_{1,n}x_n \\ ... \\ W_{n,1}x_1 + ... + W_{n,n}x_n \end{pmatrix}$$

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

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

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

Computational Graph

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

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

$$q = W \cdot x = \begin{pmatrix} W_{1,1x1} + ... + W_{1,nxn} \\ ... \\ W_{n,1x1} + ... + W_{n,nxn} \end{pmatrix}$$

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

Computational Graph

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

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

$$q = W \cdot x = \begin{pmatrix} W_{1,1x1} + ... + W_{1,nxn} \\ ... \\ W_{n,1x1} + ... + W_{n,nxn} \end{pmatrix}$$

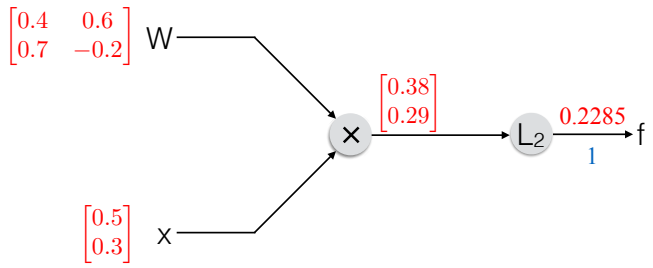
$f(q) = ||q||^2 = q_1^2 + ... + q_n^2$

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

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

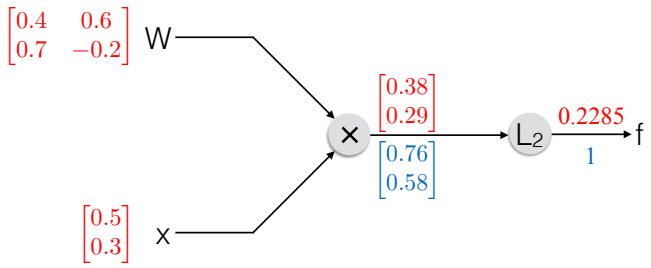
Computational Graph

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



Computational Graph

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

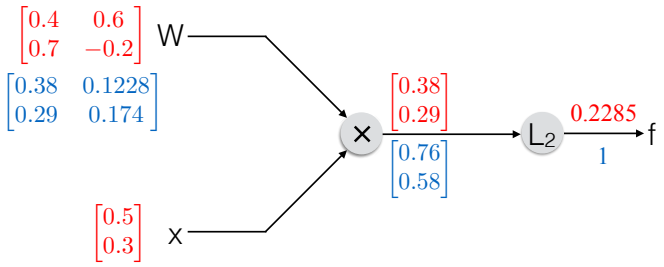


$$q = W \cdot x = \begin{pmatrix} W_{1,1x1} + ... + W_{1,nxn} \\ ... \\ W_{n,1x1} + ... + W_{n,nxn} \end{pmatrix}$$

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

Computational Graph

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

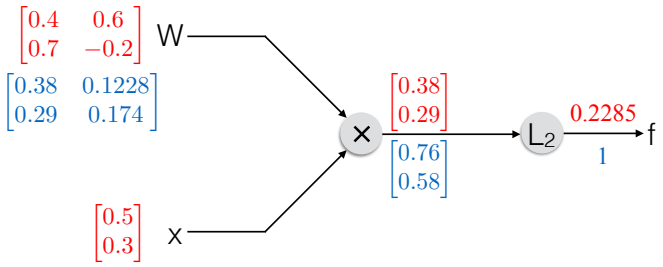


$$\frac{\partial q_k}{\partial W_{i,j}} = 1_{k=i} x_j$$

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

Computational Graph

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



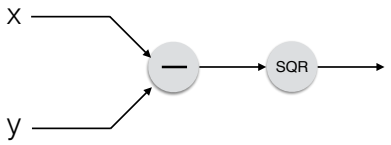
$$\frac{\partial q_k}{\partial x_i} = W_{k,i}$$

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

Layers with Computational Graph

L2-loss

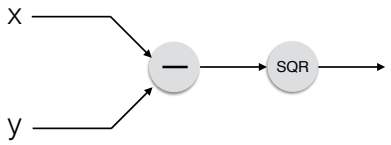
$L_2(x, y) = (x - y)^2$



Layers with Computational Graph

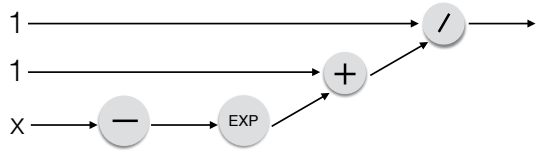
L2-loss

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



Sigmoid

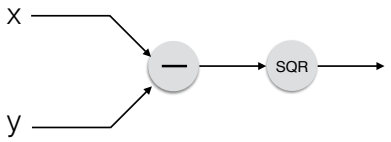
$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$



Layers with Computational Graph

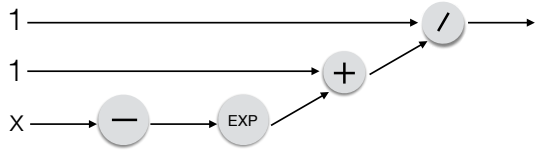
L2-loss

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



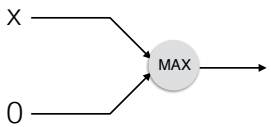
Sigmoid

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

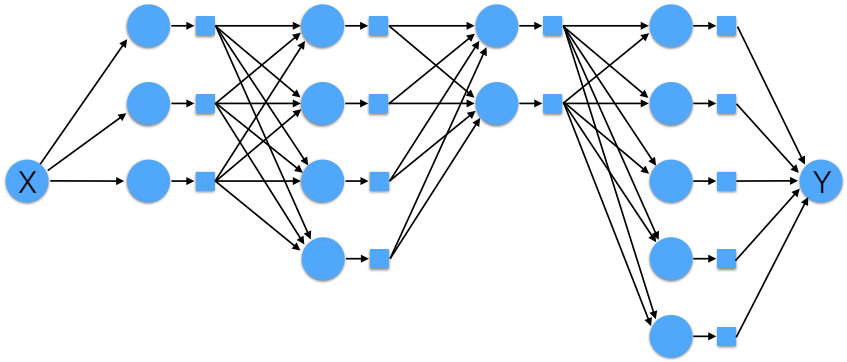


ReLU

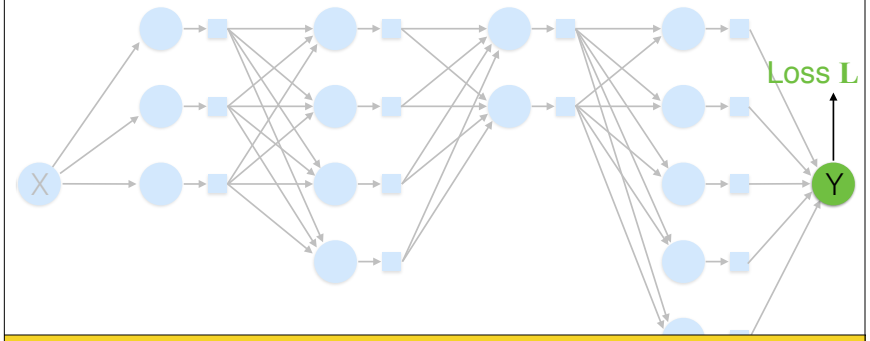
$\text{ReLU}(x) = \max(x, 0)$



Neural Networks Example



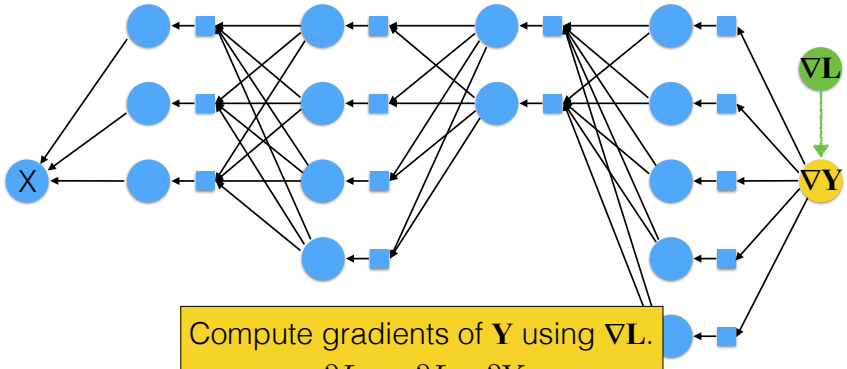
Backward propagation



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

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

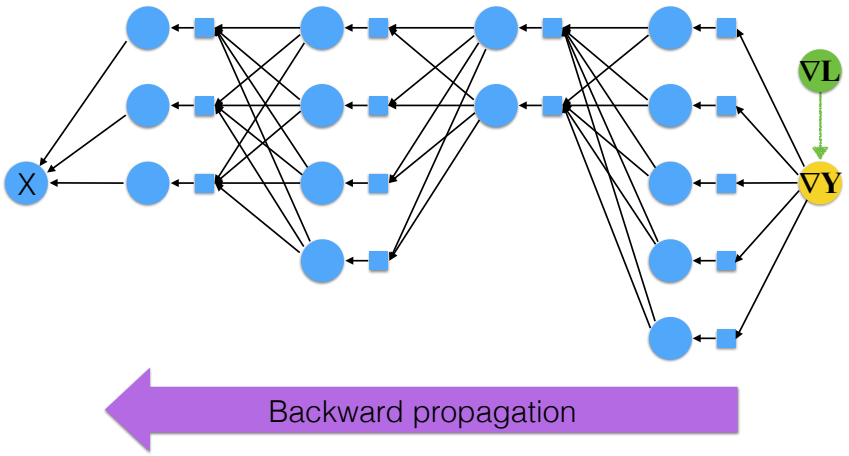
Backward propagation



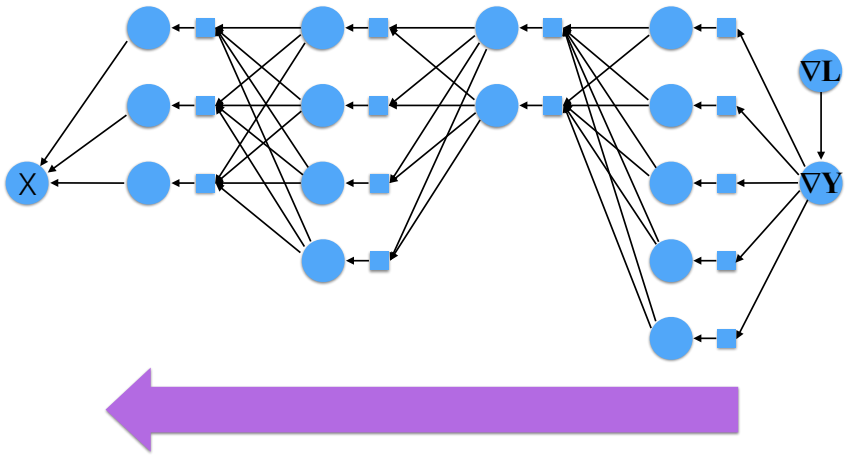
Compute gradients of **Y** using **∇L**.

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

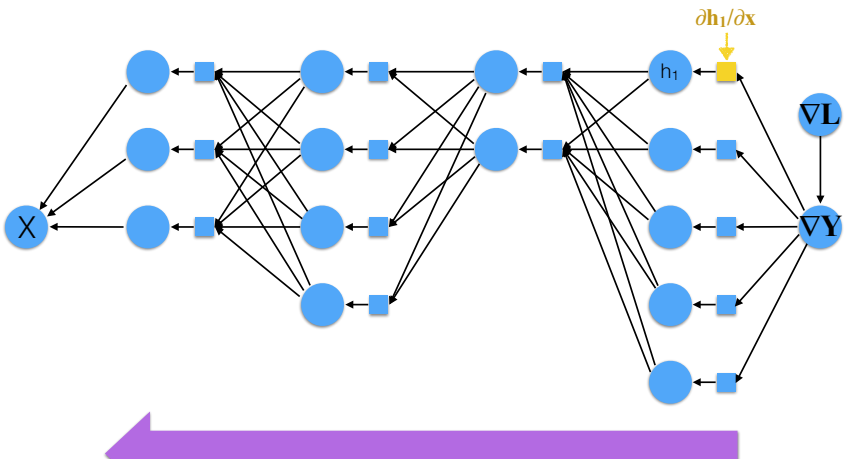
Backward propagation



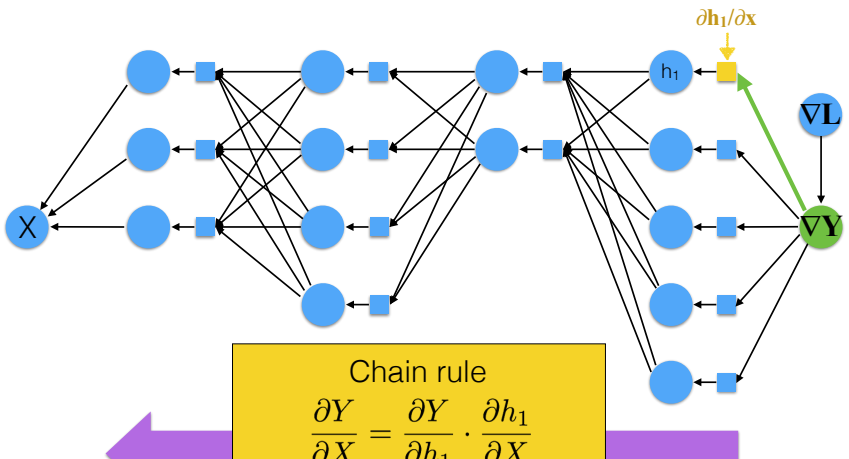
Backward propagation



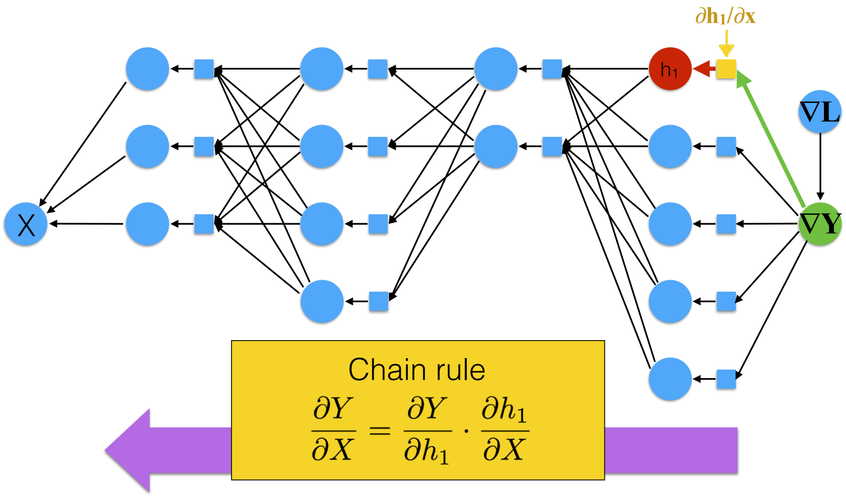
Backward propagation



Backward propagation



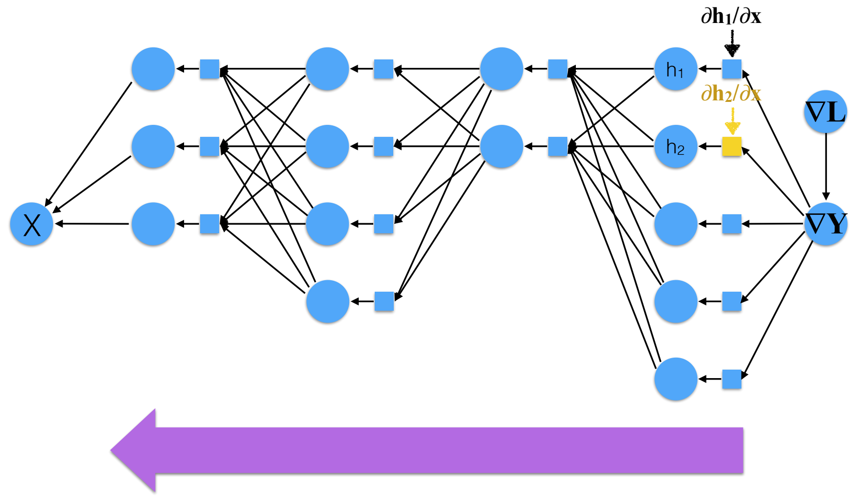
Backward propagation



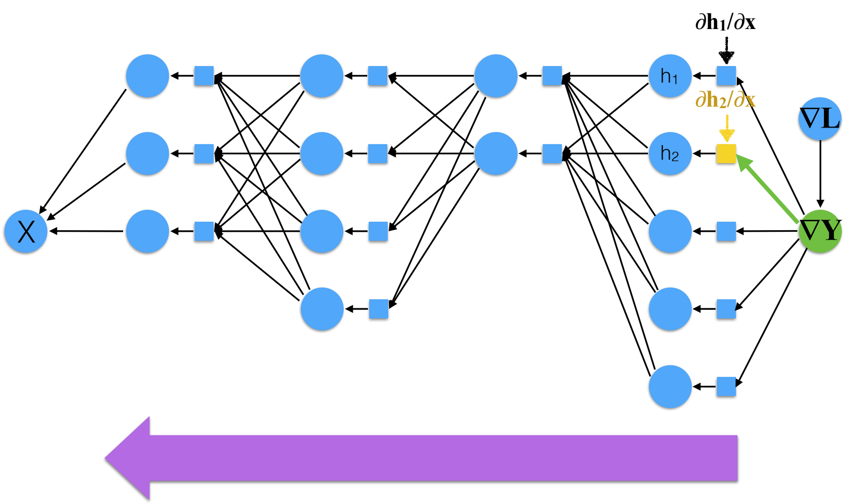
Chain rule

$$\frac{\partial Y}{\partial X} = \frac{\partial Y}{\partial h_1} \cdot \frac{\partial h_1}{\partial X}$$

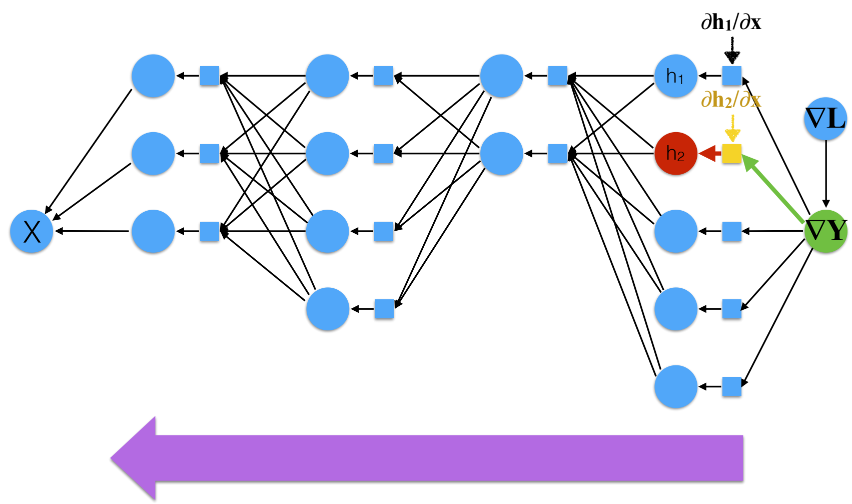
Backward propagation



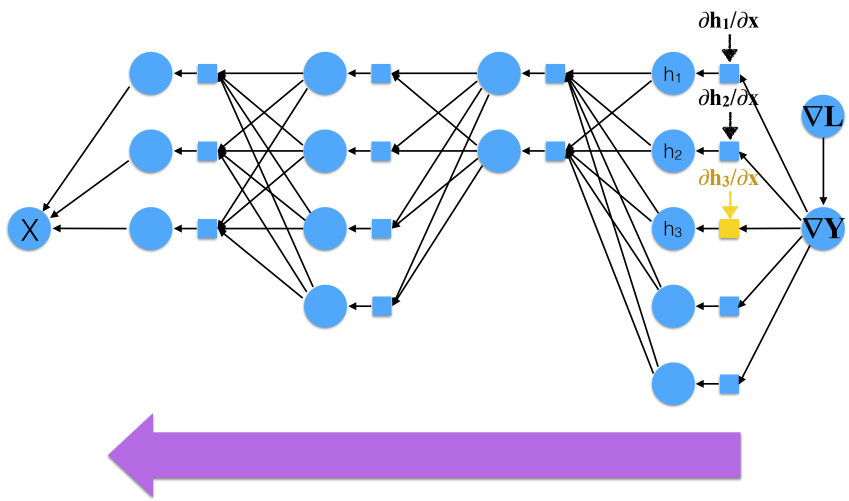
Backward propagation



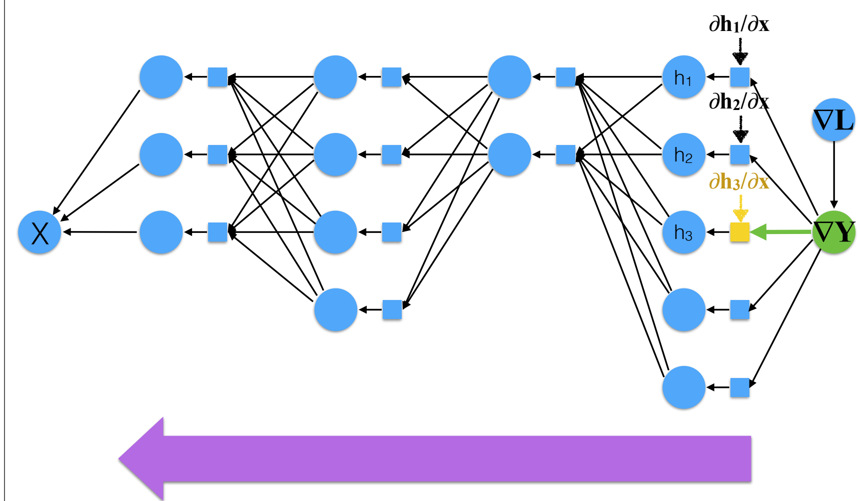
Backward propagation



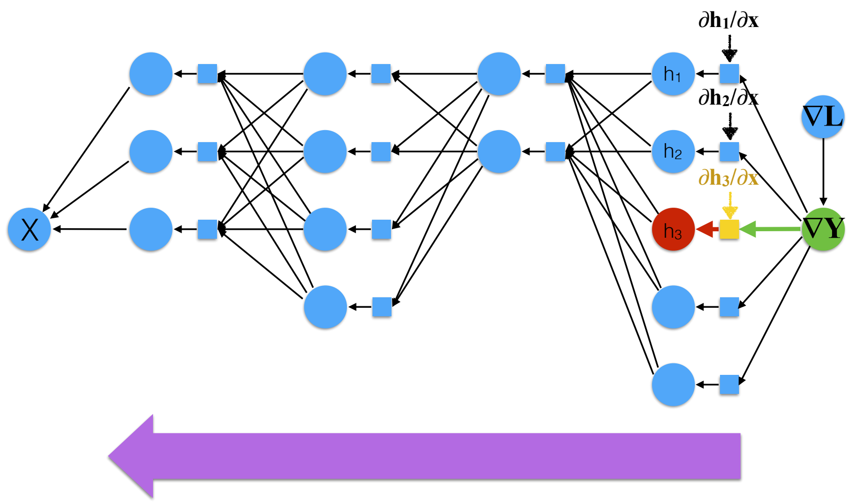
Backward propagation



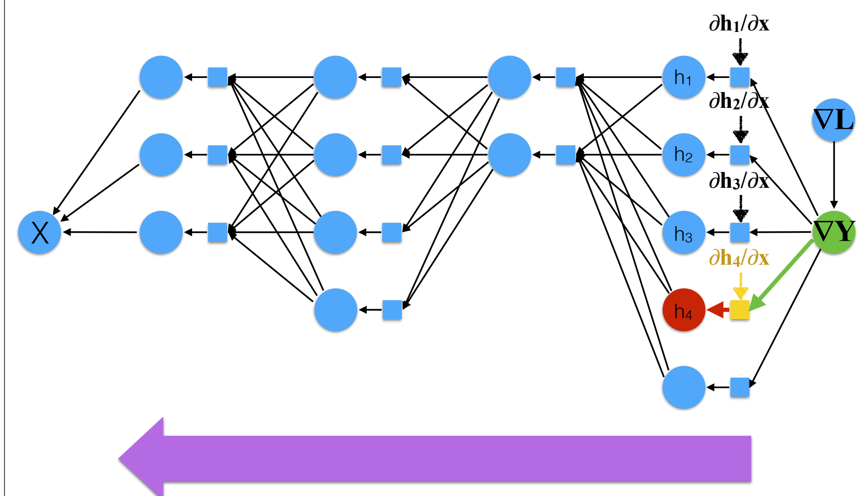
Backward propagation



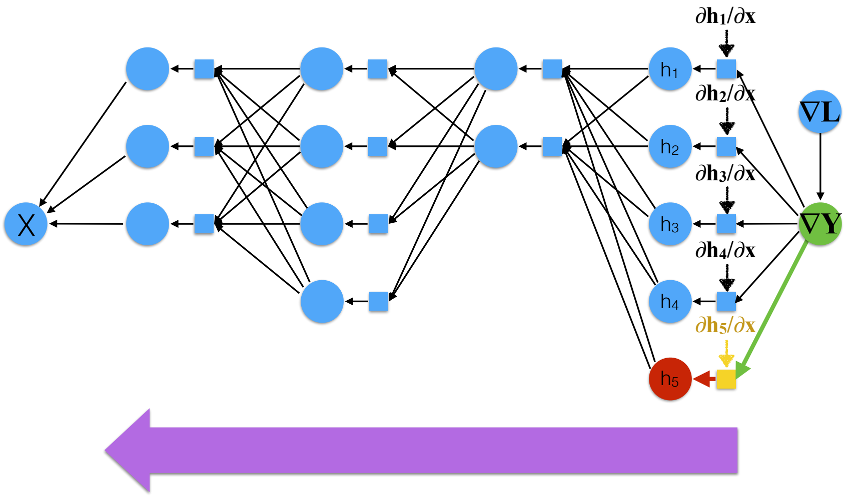
Backward propagation



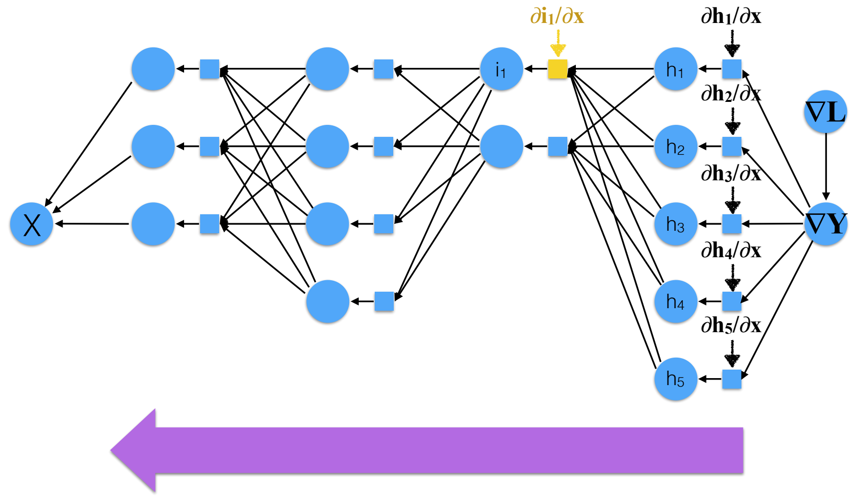
Backward propagation



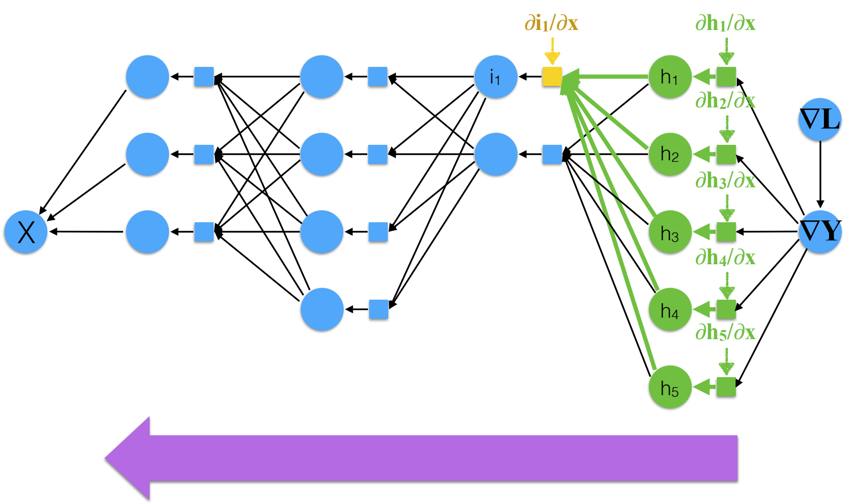
Backward propagation



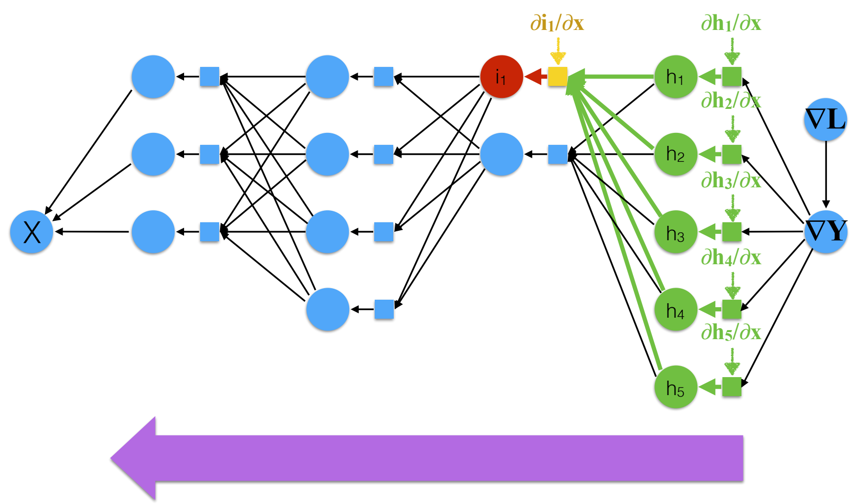
Backward propagation



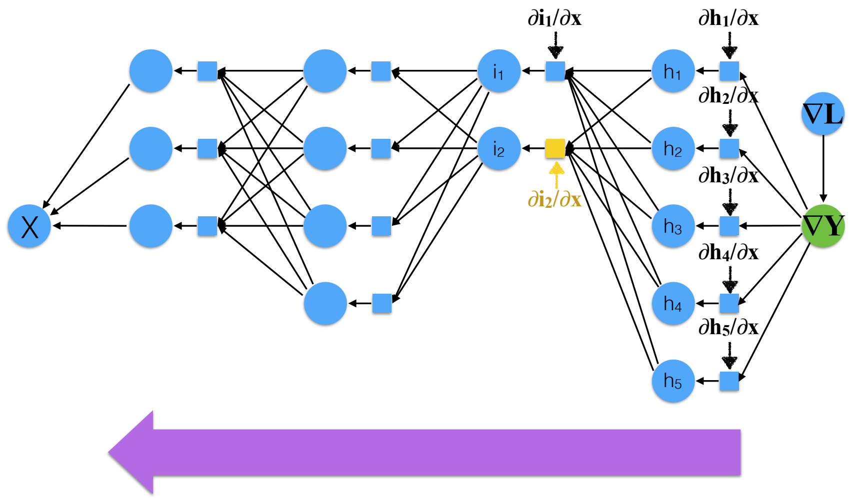
Backward propagation



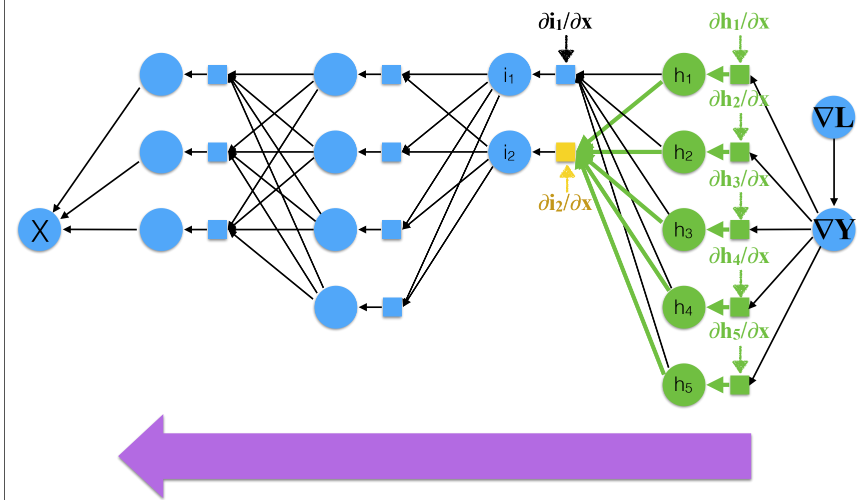
Backward propagation



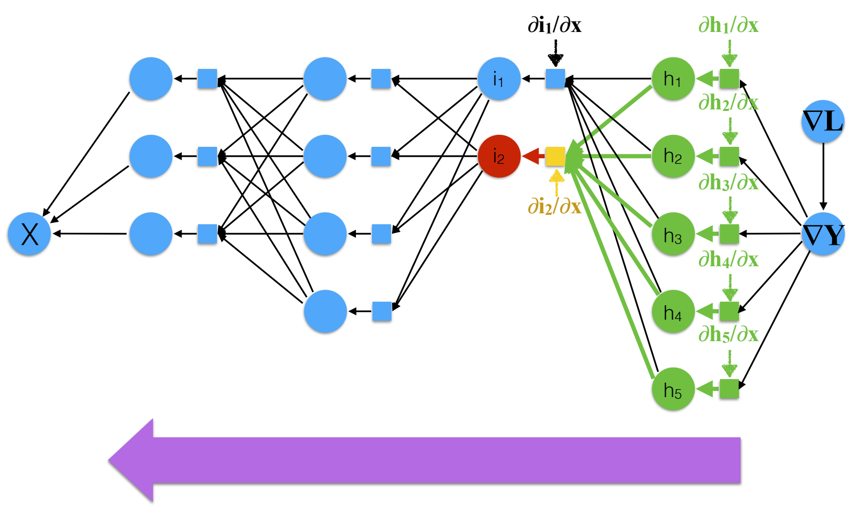
Backward propagation



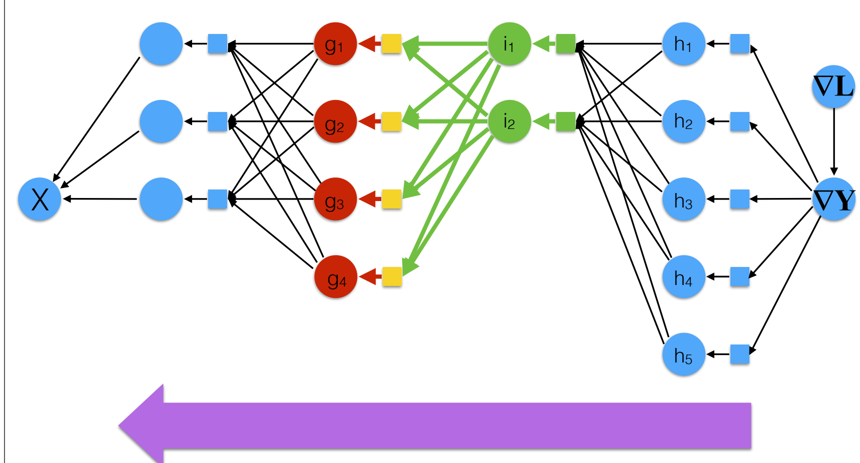
Backward propagation



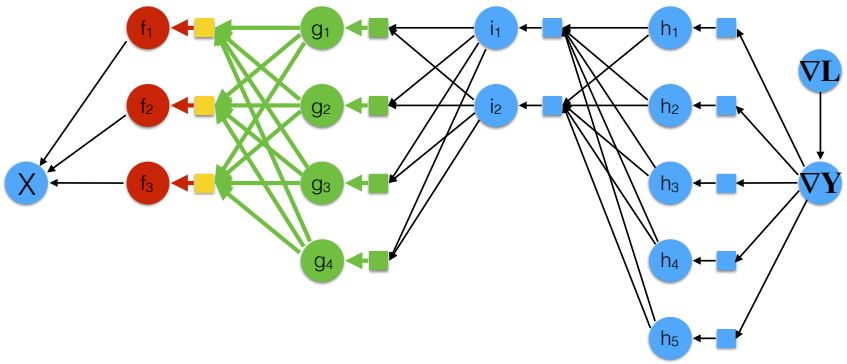
Backward propagation



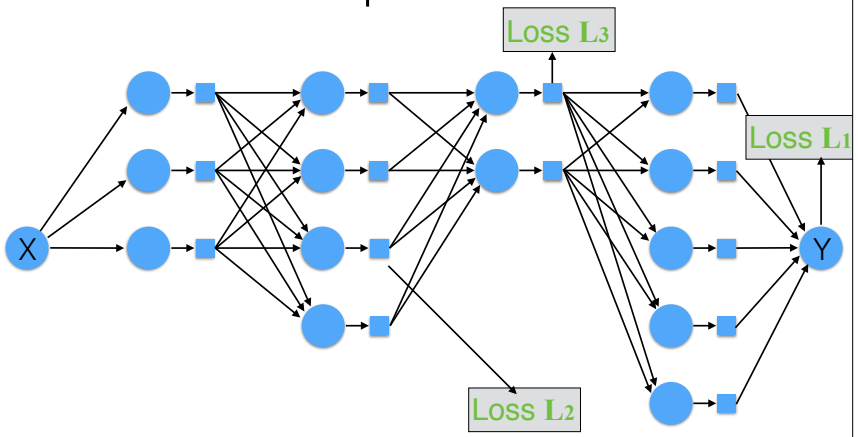
Backward propagation



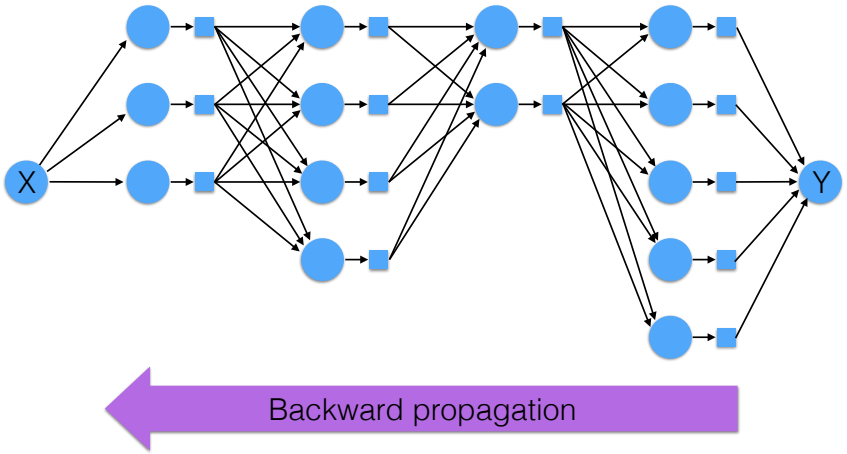
Backward propagation



Multiple losses



Neural Networks Example



Summary

Summary

- A choice of **loss function** matters.

Summary

- A choice of **loss function** matters.
- Neural networks is essentially **wired neurons** (small functions).

Summary

- A choice of **loss function** matters.
- Neural networks is essentially **wired neurons** (small functions).
- Weights can be learned using **backward propagation** (e.g. computational graph).

Break Time

See you in 5 mins!

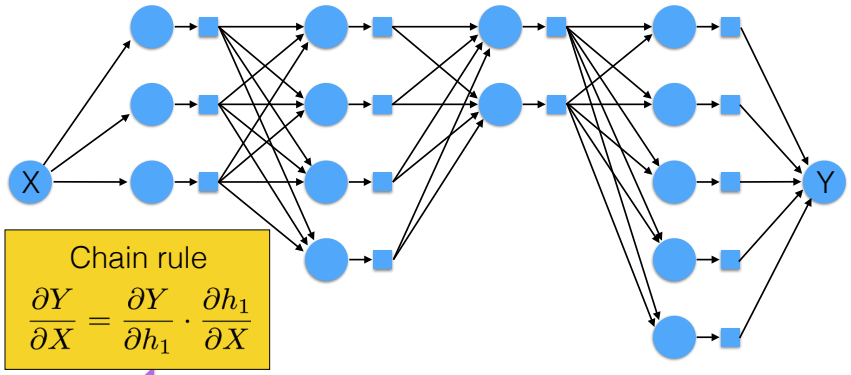
Today's agenda

- Recap: Neural Networks & Loss function
- Optimization
- Convolutional Neural Networks
- Training Neural Networks

Different NN models

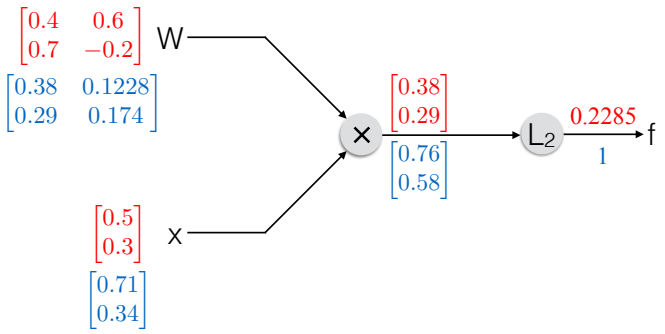
- Feedforward networks (CNNs, …)
- Recurrent networks (RNNs, …)
- Memory networks
- …

Recap: Backward Propagation



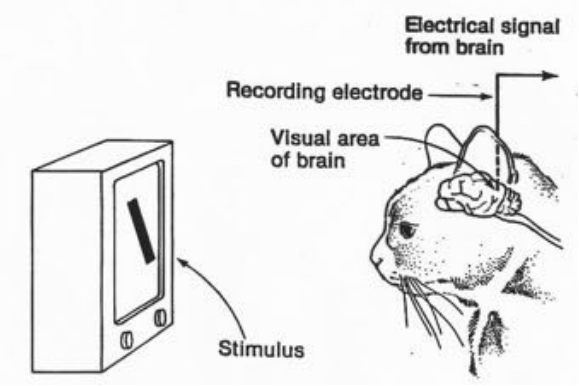
Recap: Computational Graph

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



History

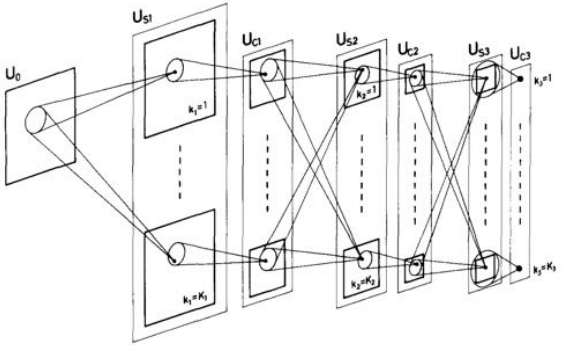
Hubel & Wiesel



Understanding how the system constructs complex representation from simple stimulus features.

History

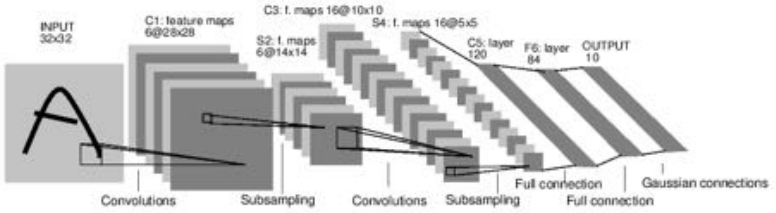
Neocognitron



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

History

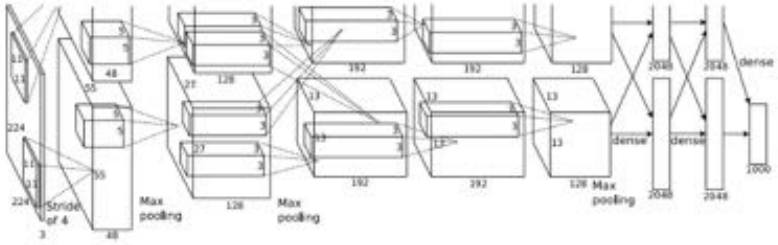
LeNet



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

History

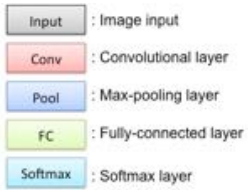
AlexNet

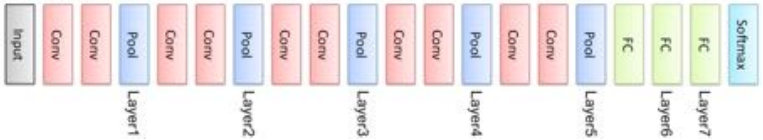


A Krizhevsky, et. al. ImageNet Classification with Deep Convolutional Neural Networks. NIPS 2012.

History

VGGNet





Convolutional NNs

Classification

Retrieval

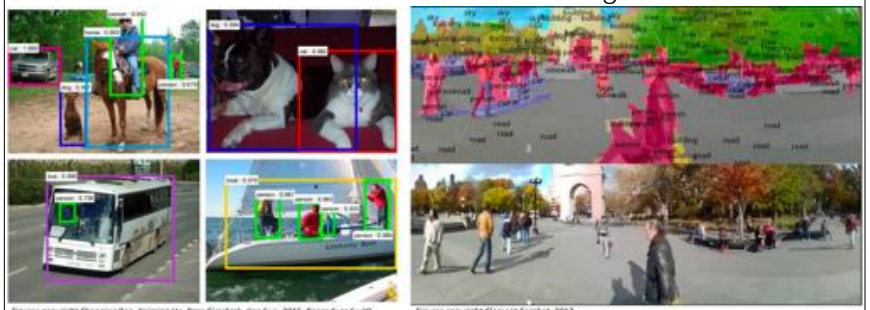
Figures copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Slide credit: CS 231N

Convolutional NNs

Detection

Segmentation



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

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

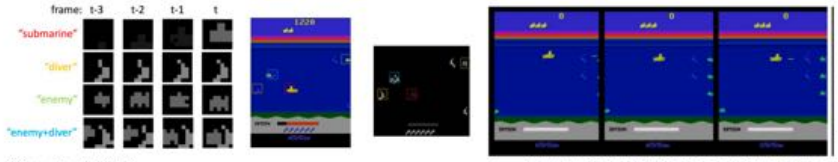
Figures copyright Clement Farabet, 2012. Reproduced with permission.

[Farabet et al., 2012]

Slide credit: CS 231N

Convolutional NNs

Game AI



[Guo et al. 2014]

Figures copyright Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard Lewis, and Xiaoshi Wang, 2014. Reproduced with permission.

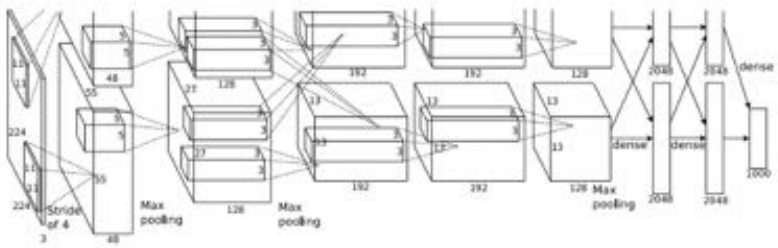
Original image is CCO public domain
Starry Night and Tree Roots by Van Gogh are in the public domain
Bokeh image is in the public domain
Stylized images copyright Justin Johnson, 2017; reproduced with permission

Gatys et al, "Image Style Transfer using Convolutional Neural Networks", CVPR 2016
Gatys et al, "Controlling Perceptual Factors in Neural Style Transfer", CVPR 2017

Arts

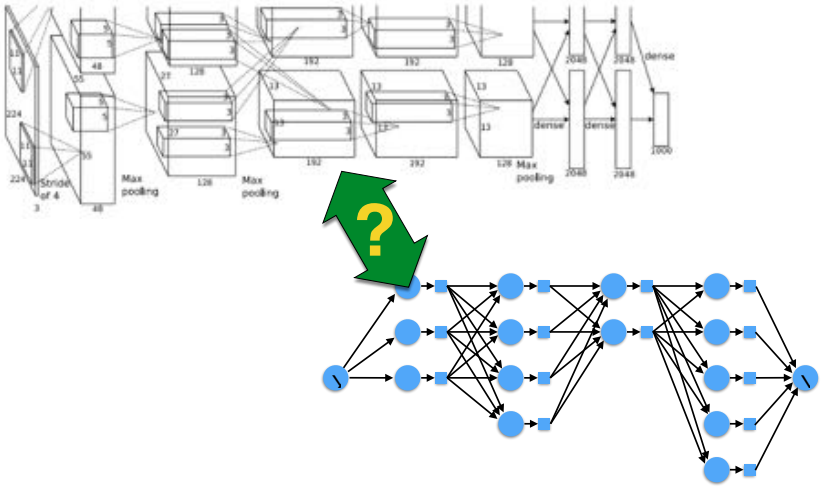
Slide credit: CS 231N

AlexNet in details

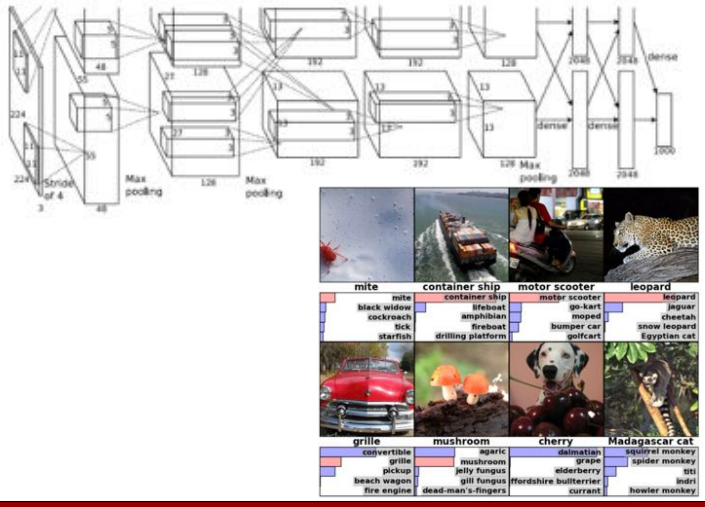


A Krizhevsky, et. al. ImageNet Classification with Deep Convolutional Neural Networks. NIPS 2012.

AlexNet in details



AlexNet in details



AlexNet in details

Image

PC: Daniel Yang

AlexNet in details

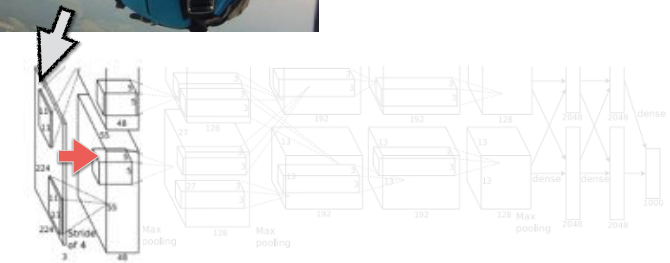


Image | Conv

AlexNet in details

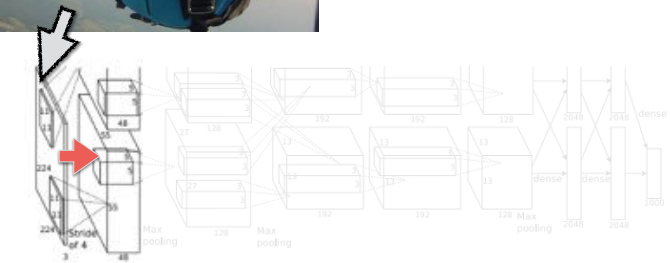


Image | Conv | Pool

AlexNet in details

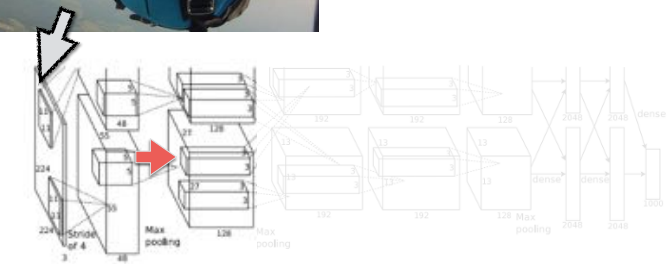


Image | Conv | Pool | Conv

AlexNet in details

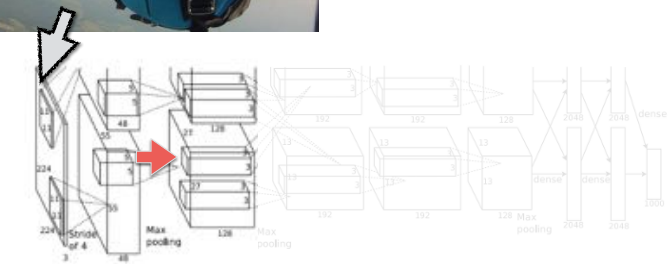


Image | Conv | Pool | Conv | Pool

AlexNet in details

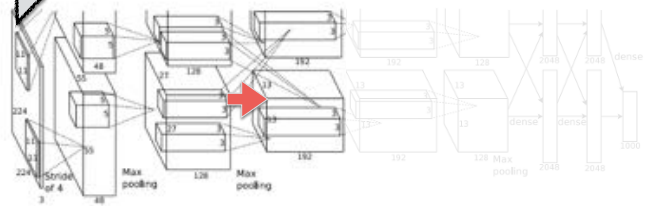


Image | Conv | Pool | Conv | Pool | Conv

AlexNet in details

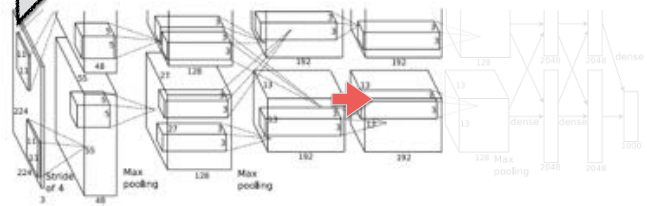


Image | Conv | Pool | Conv | Pool | Conv | Conv

AlexNet in details

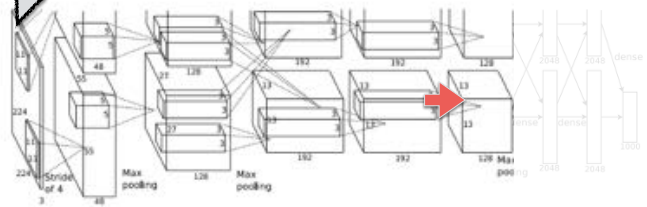


Image | Conv | Pool | Conv | Pool | Conv | Conv | Conv

AlexNet in details

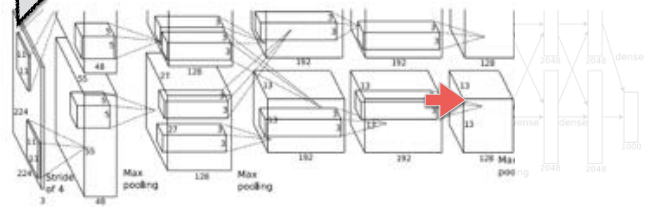


Image | Conv | Pool | Conv | Pool | Conv | Conv | Conv | Pool

AlexNet in details

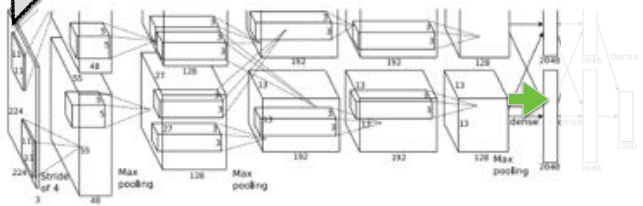


Image | Conv | Pool | Conv | Pool | Conv | Conv | Conv | Pool | FC

AlexNet in details

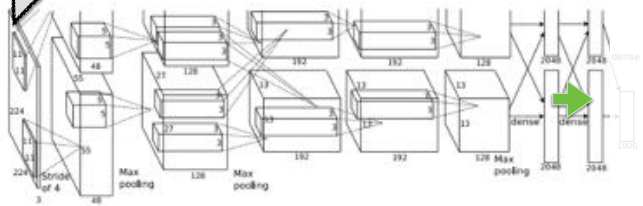
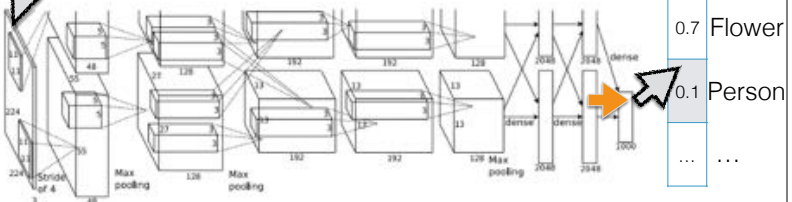


Image | Conv | Pool | Conv | Pool | Conv | Conv | Conv | Pool | FC | FC

AlexNet in details

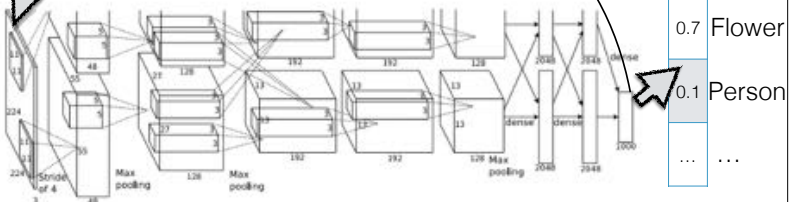


0.1 Car
0.01 Dog
0.7 Flower
0.1 Person
… …

Image | Conv | Pool | Conv | Pool | Conv | Conv | Conv | Pool | FC | FC | Softmax

AlexNet in details

Flower



0.1 Car
0.01 Dog
0.7 Flower
0.1 Person
… …

Image | Conv | Pool | Conv | Pool | Conv | Conv | Conv | Pool | FC | FC | Softmax

What are these layers?

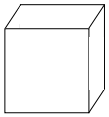
- Convolutional layer Conv
- Pool layer Pool
- Fully connected layer FC
- Softmax layer Softmax

What are these layers?

- Convolutional layer Conv
- Pool layer Pool
- Fully connected layer FC
- Softmax layer Softmax

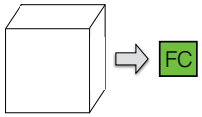
Layer = Function

Fully connected layer (FC)



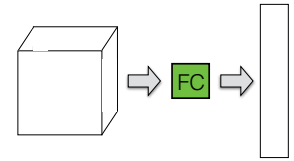
M x N x P

Fully connected layer (FC)



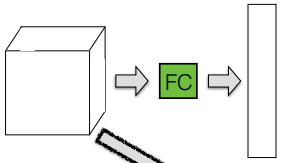
M x N x P

Fully connected layer (FC)



M x N x P Q x 1

Fully connected layer (FC)



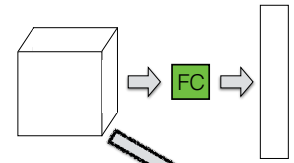
M x N x P Q x 1

FC operation takes an input, stretches it.

x

(M*N*P) x 1

Fully connected layer (FC)



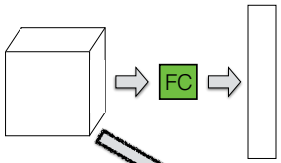
M x N x P Q x 1

FC operation takes an input, stretches it.

W X x

Q x (M*N*P) (M*N*P) x 1

Fully connected layer (FC)



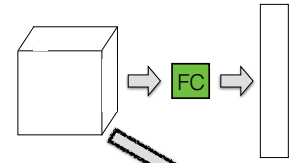
M x N x P Q x 1

FC operation takes an input, stretches it.

W X x = Wx

Q x (M*N*P) (M*N*P) x 1 Q x 1

Fully connected layer (FC)



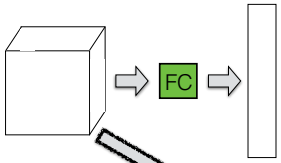
M x N x P Q x 1

FC operation takes an input, stretches it.

W Learned X x = Wx

Q x (M*N*P) (M*N*P) x 1 Q x 1

Fully connected layer (FC)



Why is this called FC?

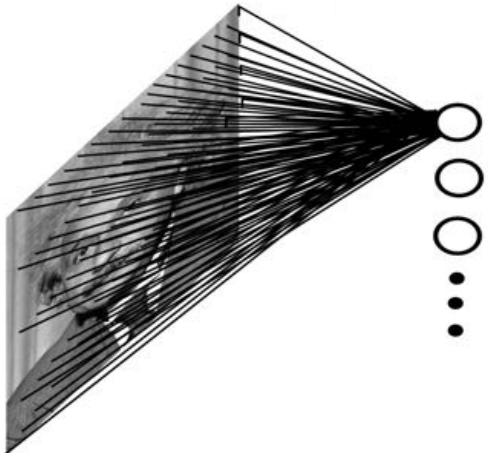
M x N x P Q x 1

FC operation takes an input, stretches it.

W Learned X x = Wx

Q x (M*N*P) (M*N*P) x 1 Q x 1

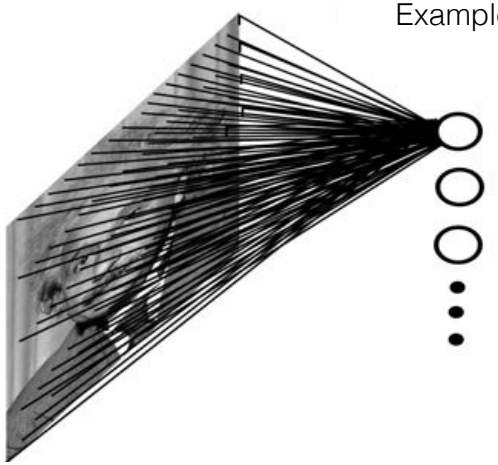
Fully connected layer (FC)



Slide credit: Marc'Aurelio Ranzato

Fully connected layer (FC)

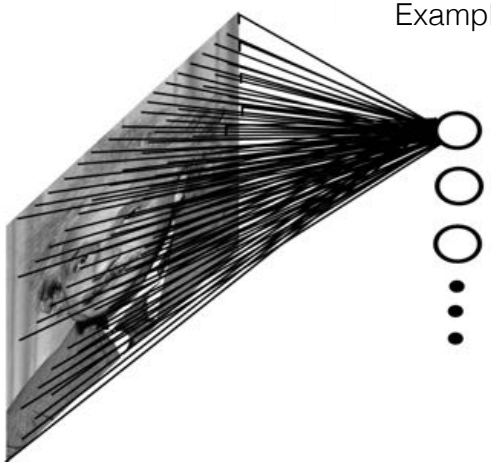
Example: 200x200 image



Slide credit: Marc'Aurelio Ranzato

Fully connected layer (FC)

Example: 200x200 image
40k hidden units

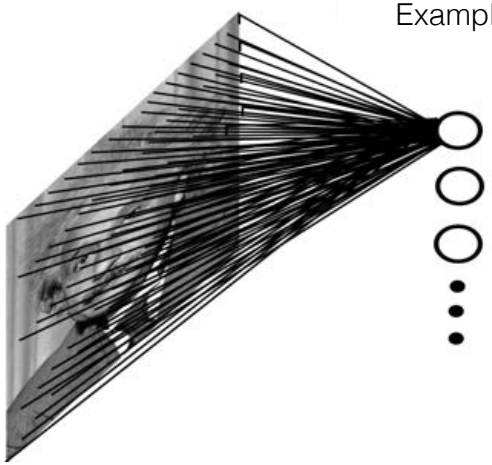


Slide credit: Marc'Aurelio Ranzato

Fully connected layer (FC)

Example: 200x200 image
40k hidden units
~2B parameters!!!

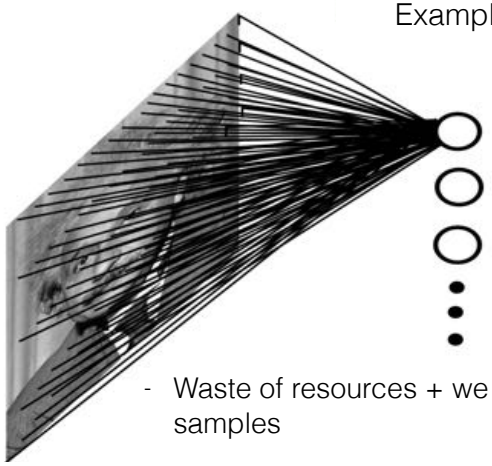
Why?



Slide credit: Marc'Aurelio Ranzato

Fully connected layer (FC)

Example: 200x200 image
40k hidden units
~2B parameters!!!

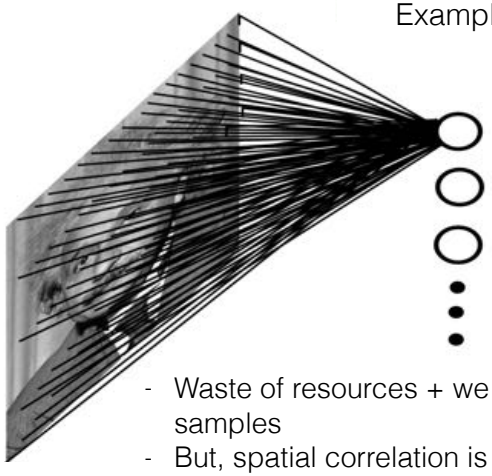


- Waste of resources + we have not enough training samples

Slide credit: Marc'Aurelio Ranzato

Fully connected layer (FC)

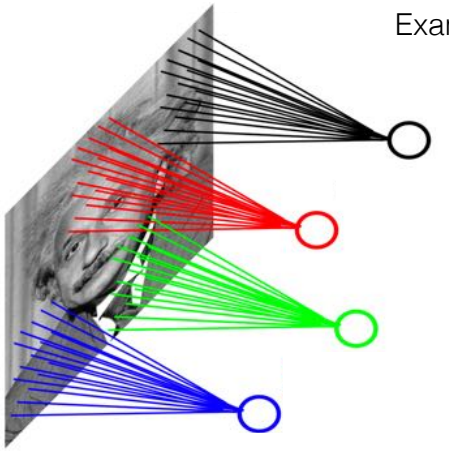
Example: 200x200 image
40k hidden units
~2B parameters!!!



- Waste of resources + we have not enough training samples
- But, spatial correlation is local!

Slide credit: Marc'Aurelio Ranzato

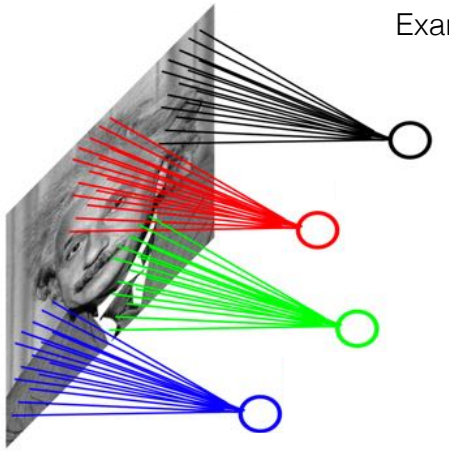
Locally connected layer



Example: 200x200 image
40k hidden units
Filter size: 10x10

Slide credit: Marc'Aurelio Ranzato

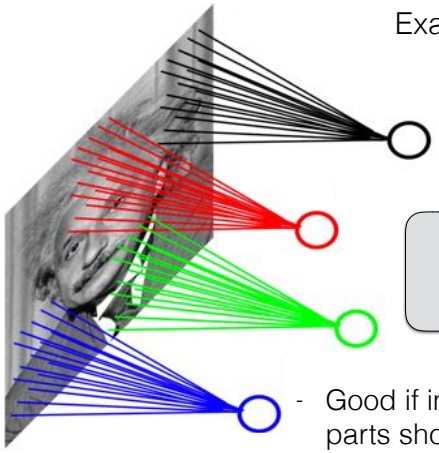
Locally connected layer



Example: 200x200 image
40k hidden units
Filter size: 10x10
4M parameters

Slide credit: Marc'Aurelio Ranzato

Locally connected layer



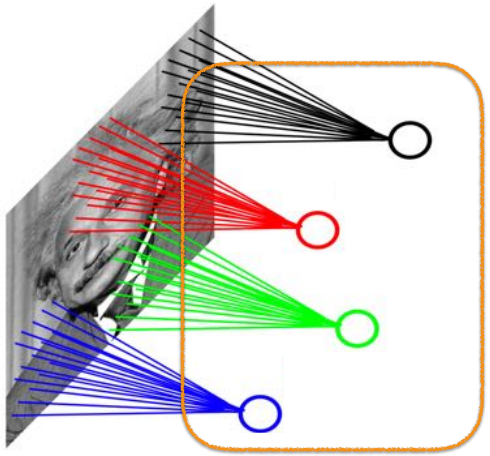
Example: 200x200 image
40k hidden units
Filter size: 10x10
4M parameters

What can we do?

- Good if images are registered (i.e. parts show up at the same locations)

Slide credit: Marc'Aurelio Ranzato

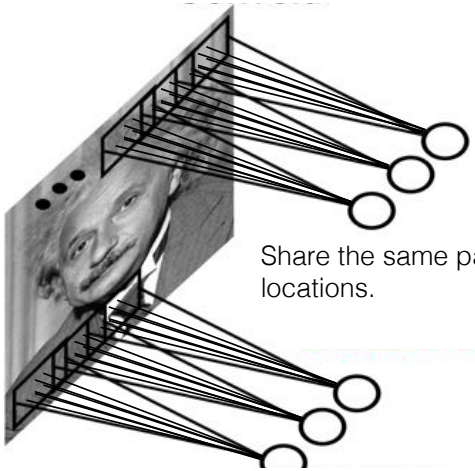
Locally connected layer



Share all local weights

Slide credit: Marc'Aurelio Ranzato

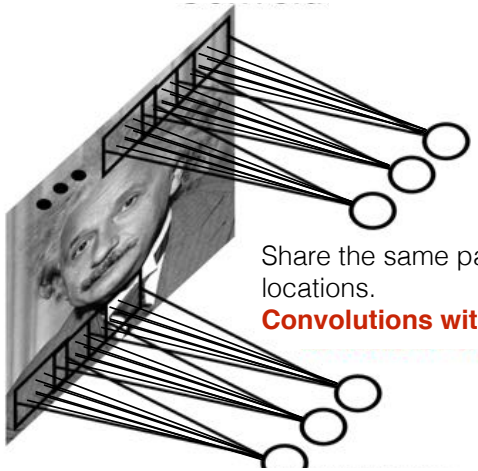
Convolutional layer



Share the same parameters across different locations.

Slide credit: Marc'Aurelio Ranzato

Convolutional layer

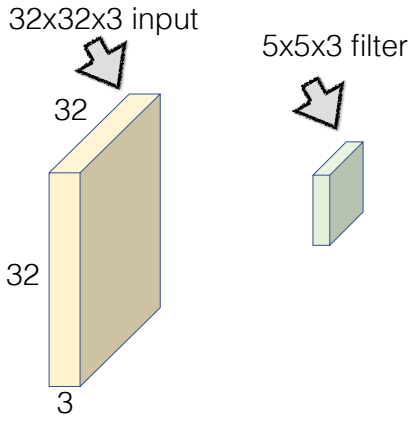


Share the same parameters across different locations.

Convolutions with learned kernels (weights)

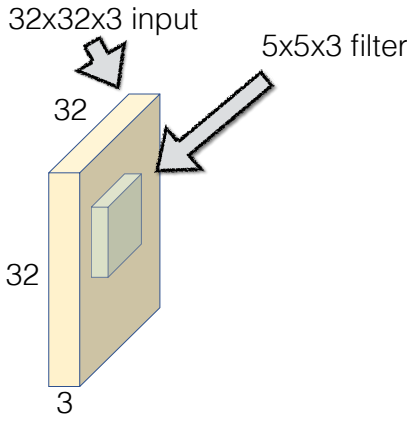
Slide credit: Marc'Aurelio Ranzato

Convolutional layer



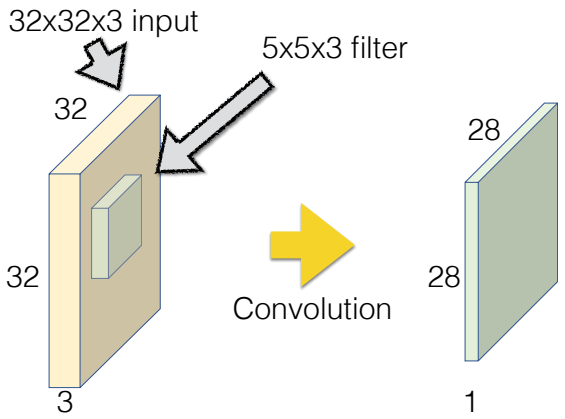
Slide credit: CS 231N

Convolutional layer



Slide credit: CS 231N

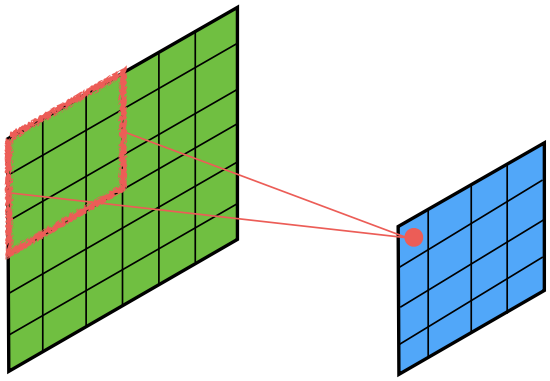
Convolutional layer



Slide credit: CS 231N

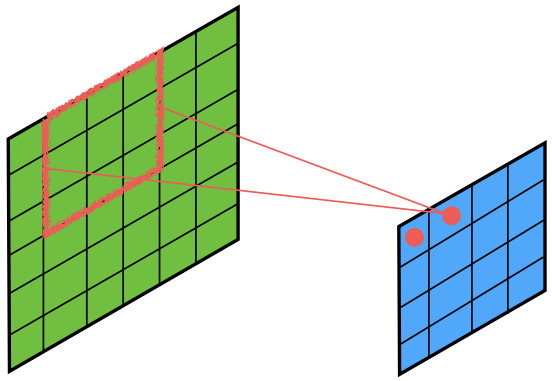
Convolution?

Convolutional layer



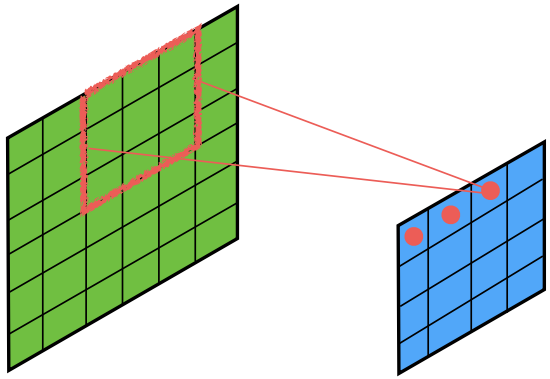
Inspired by Marc'Aurelio Ranzato's slides

Convolutional layer



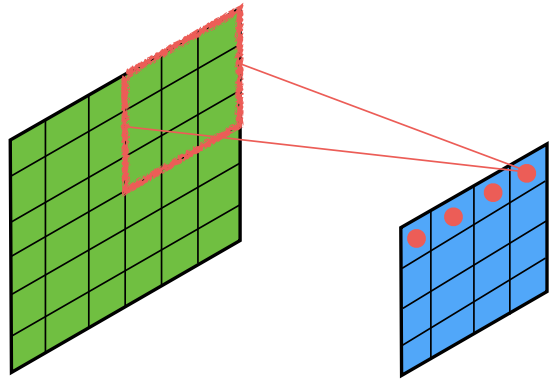
Inspired by Marc'Aurelio Ranzato's slides

Convolutional layer



Inspired by Marc'Aurelio Ranzato's slides

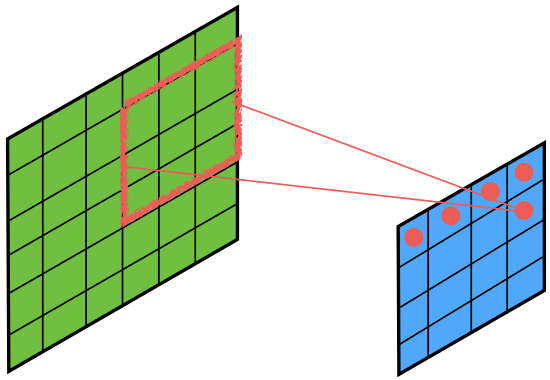
Convolutional layer



Inspired by Marc'Aurelio Ranzato's slides

Joseph J. Lim CSCI 599 @ USC Lecture 4

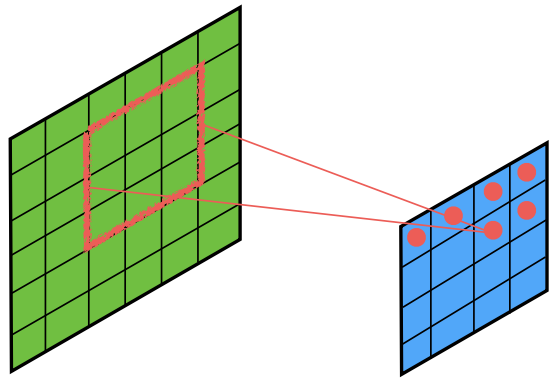
Convolutional layer



Inspired by Marc'Aurelio Ranzato's slides

Joseph J. Lim CSCI 599 @ USC Lecture 4

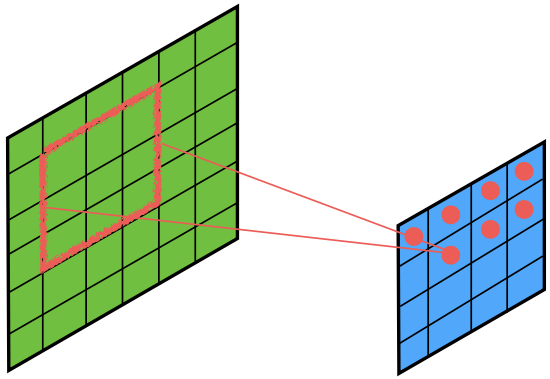
Convolutional layer



Inspired by Marc'Aurelio Ranzato's slides

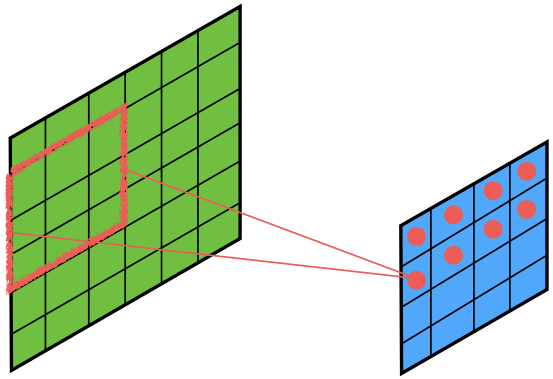
Joseph J. Lim CSCI 599 @ USC Lecture 4

Convolutional layer



Inspired by Marc'Aurelio Ranzato's slides

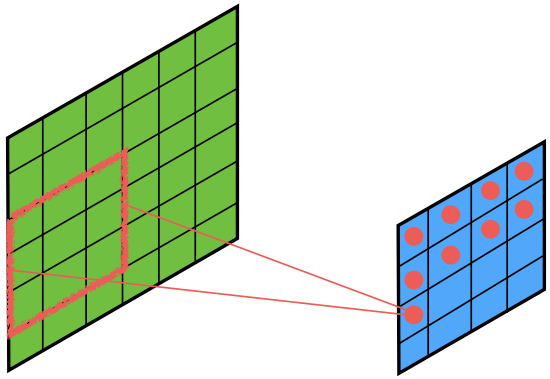
Convolutional layer



Inspired by Marc'Aurelio Ranzato's slides

Joseph J. Lim CSCI 599 @ USC Lecture 4

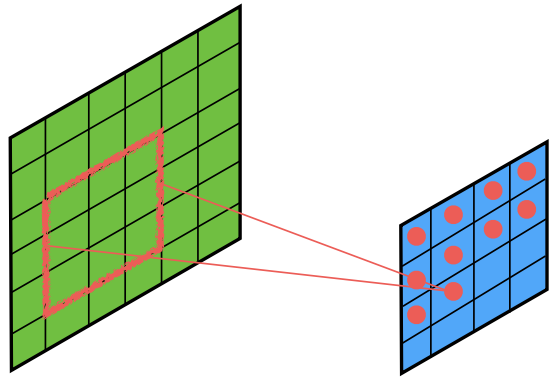
Convolutional layer



Inspired by Marc'Aurelio Ranzato's slides

Joseph J. Lim CSCI 599 @ USC Lecture 4

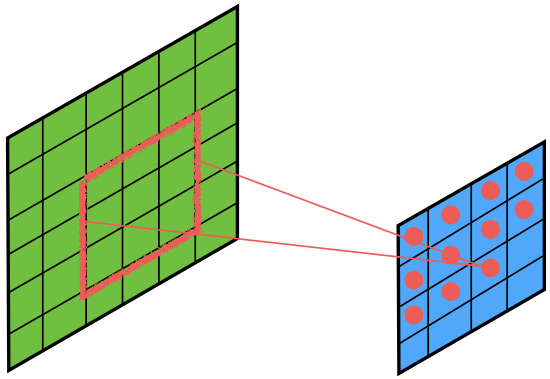
Convolutional layer



Inspired by Marc'Aurelio Ranzato's slides

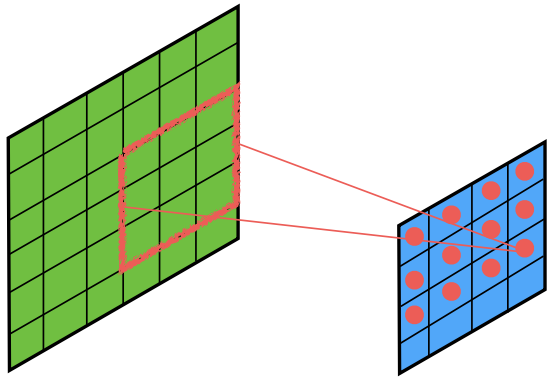
Joseph J. Lim CSCI 599 @ USC Lecture 4

Convolutional layer



Inspired by Marc'Aurelio Ranzato's slides

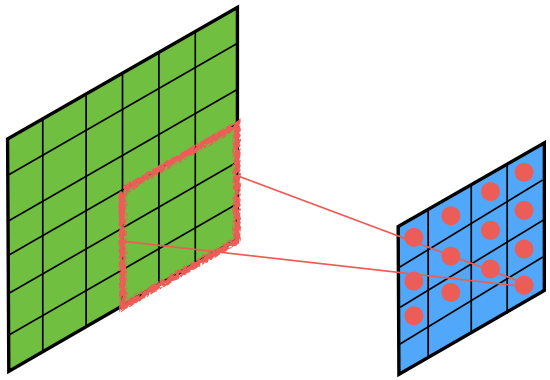
Convolutional layer



Inspired by Marc'Aurelio Ranzato's slides

Joseph J. Lim CSCI 599 @ USC Lecture 4

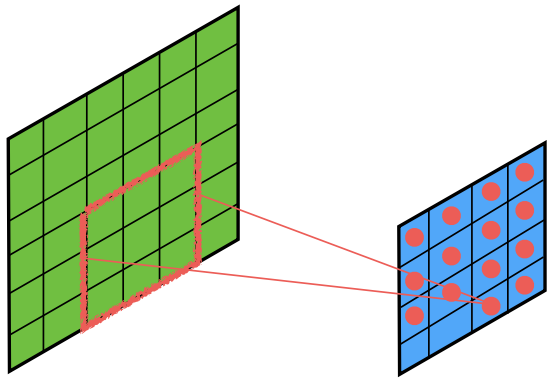
Convolutional layer



Inspired by Marc'Aurelio Ranzato's slides

Joseph J. Lim CSCI 599 @ USC Lecture 4

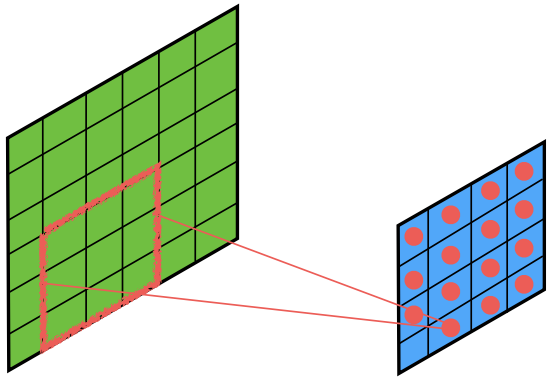
Convolutional layer



Inspired by Marc'Aurelio Ranzato's slides

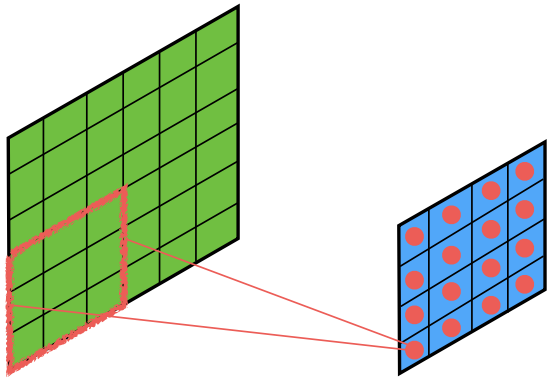
Joseph J. Lim CSCI 599 @ USC Lecture 4

Convolutional layer



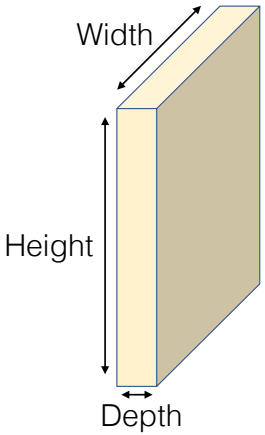
Inspired by Marc'Aurelio Ranzato's slides

Convolutional layer

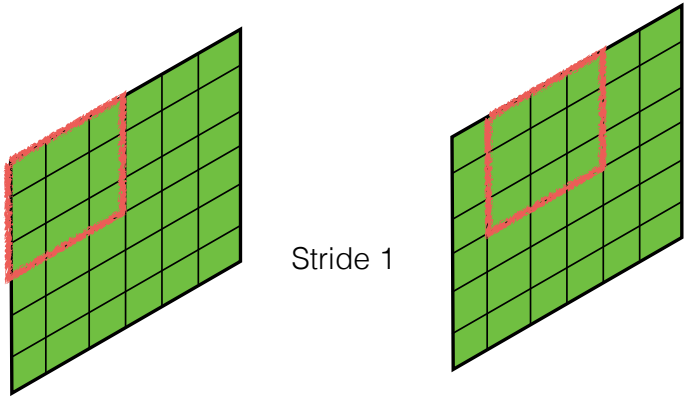


Inspired by Marc'Aurelio Ranzato's slides

Terminologies

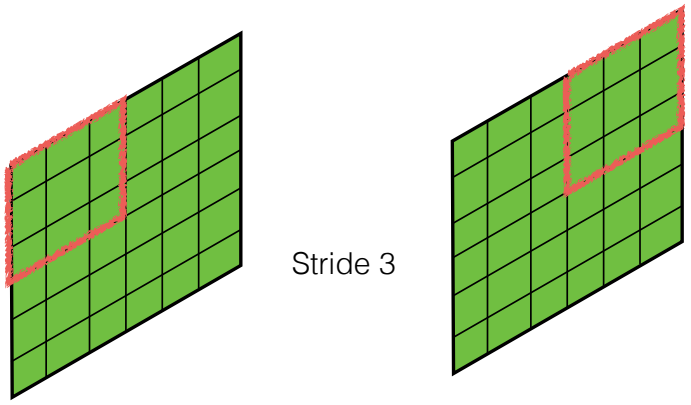


Terminologies



Step 1 Step 2

Terminologies



Terminologies

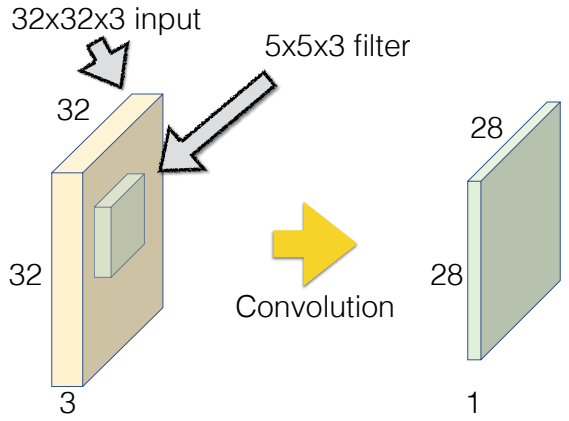
a	b	c	d	e
f	g	h	i	j
k	l	m	n	o
p	q	r	s	t
u	v	w	x	y

Terminologies

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	a	b	c	d	e	0	0
0	0	f	g	h	i	j	0	0
0	0	k	l	m	n	o	0	0
0	0	p	q	r	s	t	0	0
0	0	u	v	w	x	y	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

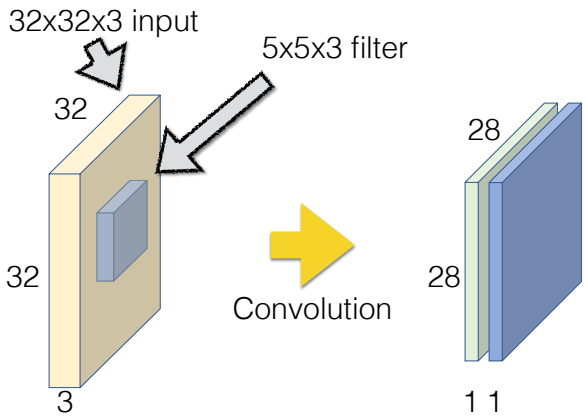
Zero-padding: 2

Convolutional layer



Slide credit: CS 231N

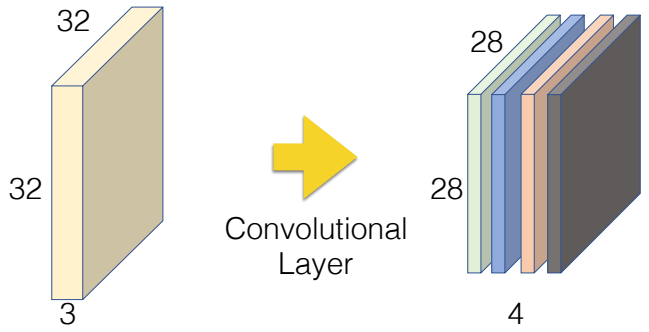
Convolutional layer



Slide credit: CS 231N

Convolutional layer

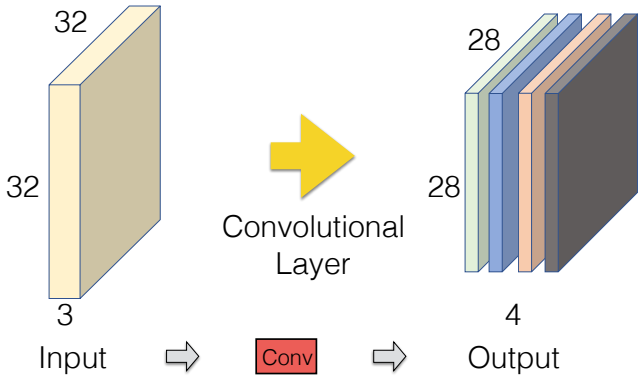
With 4 5x5x3 filters, we will get 4 activation outputs.
Hence, the final output is 28x28x4.



Slide credit: CS 231N

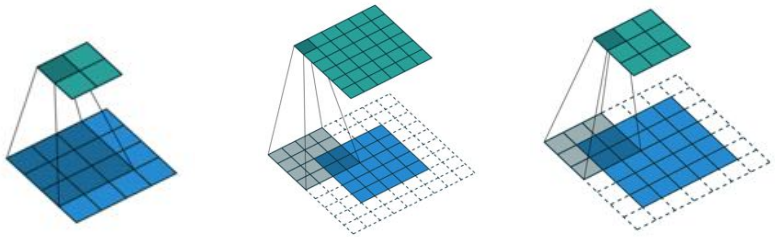
Convolutional layer

With 4 5x5x3 filters, we will get 4 activation outputs.
Hence, the final output is 28x28x4.



Slide credit: CS 231N

Convolutional layer



padding 0, stride 1 padding 2, stride 1 padding 1, stride 2

Note: it can have **depth** (3rd dimension).

Image credit: vdumoulin (github): https://github.com/vdumoulin/conv_arithmetic

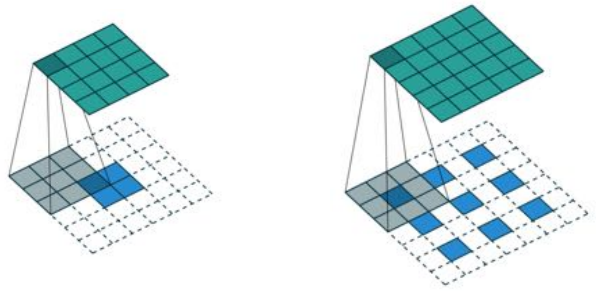
Deconvolutional layer

Deconvolutional layer

Warning: this is not about **de-**convolution!

Deconvolutional layer

Warning: this is not about **de-**convolution!



padding 0, stride 1 padding 1, stride 1

Why do we need this?

Image credit: vdumoulin (github): https://github.com/vdumoulin/conv_arithmetic

Fully connected vs Convolutional

- Fully connected layer
- Locally connected layer
- Convolutional layer

- Which layer is the best? Why?

What are these layers?

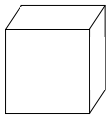
- Convolutional layer 

- Pool layer 

- Fully connected layer 

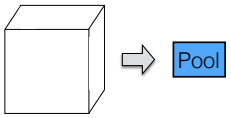
- Softmax layer 

Max pooling layer



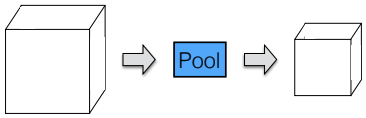
M x N x P

Max pooling layer



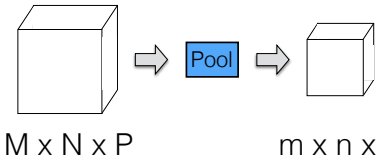
M x N x P

Max pooling layer



M x N x P m x n x P

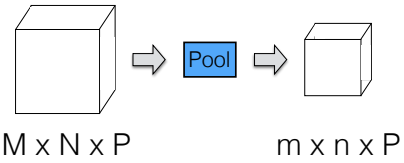
Max pooling layer



- Shrinks the feature smaller (compact + manageable)

Slide credit: CS 231N

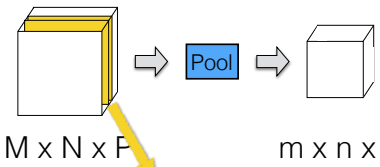
Max pooling layer



- Shrinks the feature smaller (compact + manageable)
- Each activation map (3rd dim) is handled separately

Slide credit: CS 231N

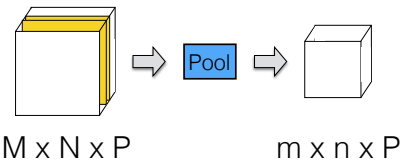
Max pooling layer



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

Max pooling layer

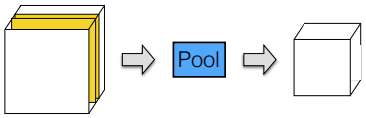


1	3	2	7	2
5	1	3	0	5
4	8	5	0	6
3	2	6	0	8
0	7	4	6	9

max pool with
3x3 filters & stride 2

8	

Max pooling layer



M x N x P

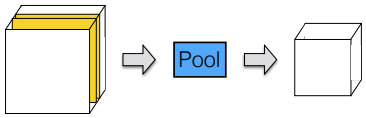
m x n x P

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

Max pooling layer



M x N x P

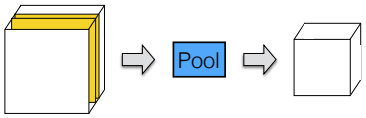
m x n x P

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	

Max pooling layer



M x N x P

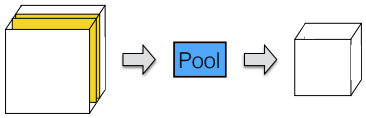
m x n x P

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



M x N x P

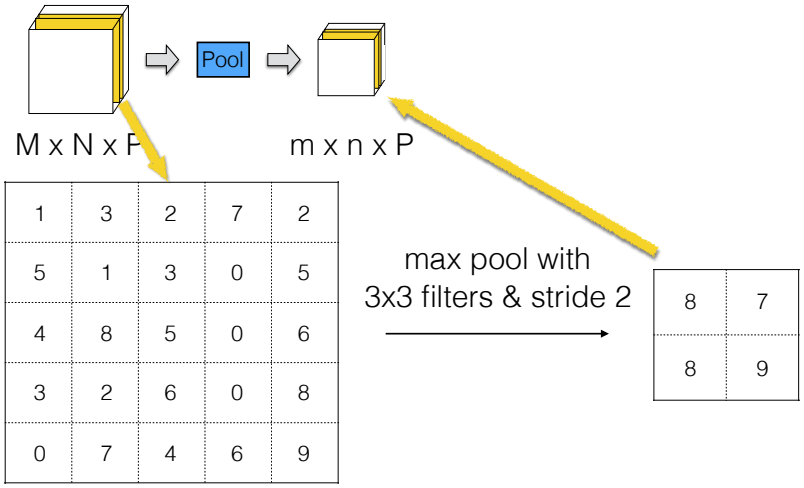
m x n x P

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



Pooling layers

- **Max pooling** (most common)
- Average pooling
- L2-norm pooling
- …

Why pooling layers?

Why pooling layers?

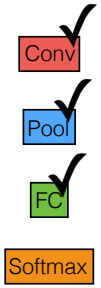
- Reduces # of parameters in the following layers

Why pooling layers?

- Reduces # of parameters in the following layers
  - Hence, less overfitting!

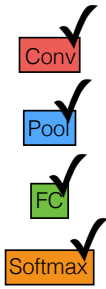
What are these layers?

- Convolutional layer
- Pool layer
- Fully connected layer
- Softmax layer



What are these layers?

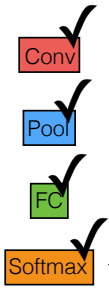
- Convolutional layer
- Pool layer
- Fully connected layer
- Softmax layer



Loss layer (Lecture 4)

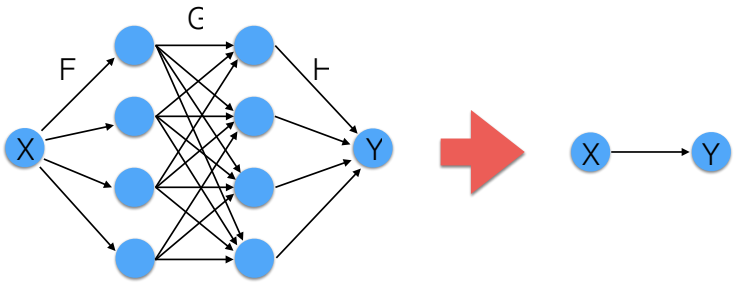
All these are linear!!!

- Convolutional layer
- Pool layer
- Fully connected layer
- Softmax layer



Loss layer (Lecture 4)

Activation functions

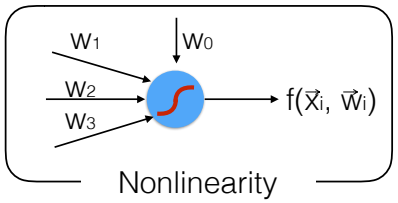


Y=H(G(F(X)))

Y=(HGF)(X)

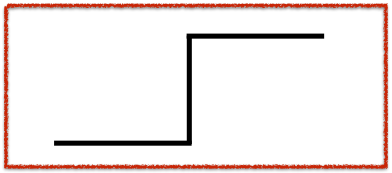
Without nonlinear activation,
multiple layers are reduced to one matrix multiplication.

Activation functions



Nonlinearity makes a neural network deeper

Step function



- Firing after a threshold
- Is not differentiable
- Cannot apply back-propagation

Anything else we can use?

Activation functions

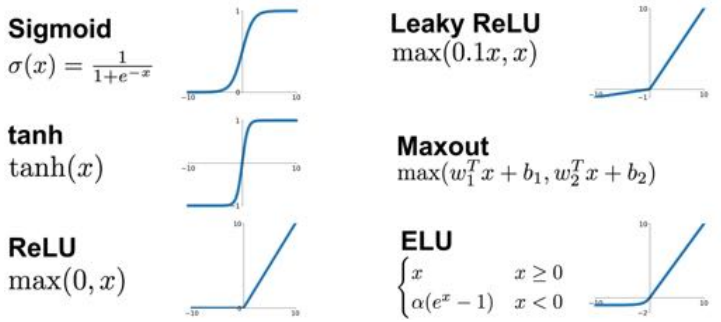
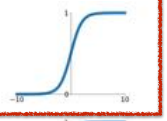


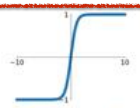
Figure from Stanford cs231n lecture slides

Activation functions

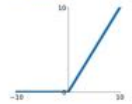
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$



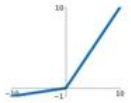
tanh
$\tanh(x)$



ReLU
$\max(0, x)$



Leaky ReLU
$\max(0.1x, x)$



Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU
$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$

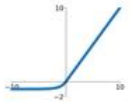
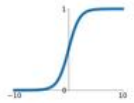


Figure from Stanford cs231n lecture slides

Sigmoid

Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$

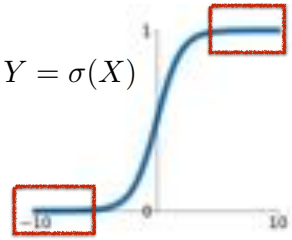


- Similar to step functions, but differentiable
- (-) easily saturated (no gradients)

Figure from Stanford cs231n lecture slides

Gradient vanishing problem

$Y = \sigma(X)$



Chain rule

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

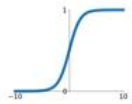
is almost 0 for most of X

Thus, $\frac{\partial Y}{\partial X}$ also goes to 0.

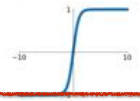
Figure from Stanford cs231n lecture slides

Activation functions

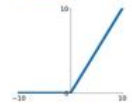
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$



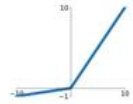
tanh
$\tanh(x)$



ReLU
$\max(0, x)$



Leaky ReLU
$\max(0.1x, x)$



Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU
$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$

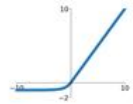
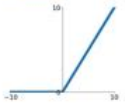


Figure from Stanford cs231n lecture slides

ReLU

ReLU
$\max(0, x)$

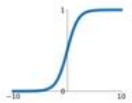


- Prevent gradient vanishing when x > 0
- Computationally efficient
- Biologically plausible
- (-) Still loose gradient when x < 0

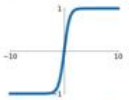
Figure from Stanford cs231n lecture slides

Activation functions

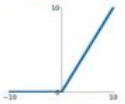
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$



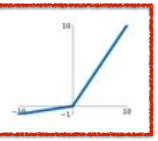
tanh
$\tanh(x)$



ReLU
$\max(0, x)$



Leaky ReLU
$\max(0.1x, x)$



Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

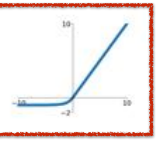
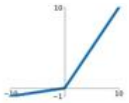


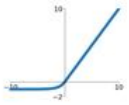
Figure from Stanford cs231n lecture slides

Leaky ReLU/ELU

Leaky ReLU
$\max(\alpha x, x)$



ELU
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



- No gradient vanishing problem
- Can be used instead of ReLU (e.g. Leaky ReLU for deconvolution)

Figure from Stanford cs231n lecture slides

AlexNet in details

AlexNet in details

Flower

0.1 Car

Typically omit ReLU for the visualization

0.1 Person

… …

Image Conv Pool ReLU Conv Pool ReLU Conv ReLU Conv ReLU Conv Pool ReLU FC FC Softmax

AlexNet in details

Flower

0.1 Car

Typically omit ReLU for the visualization

0.1 Person

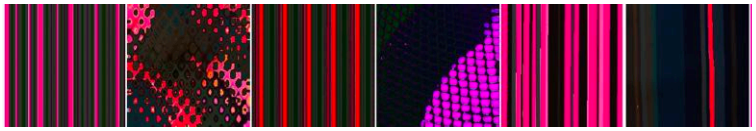
… …

Image Conv Pool Conv Pool Conv Conv Conv Pool FC FC Softmax

What did AlexNet learn?

Imag Conv Conv Conv Conv Conv FC FC Softmax

red

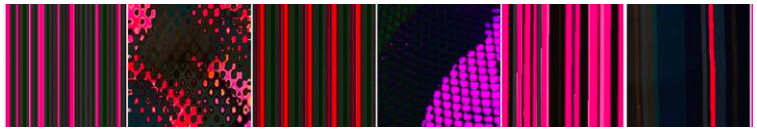


Bau, Zhou, and et. al. Network Dissection: Quantifying Interpretability of Deep Visual Representations. CVPR 2017.

What did AlexNet learn?

Imag Conv Conv Conv Conv Conv FC FC Softmax

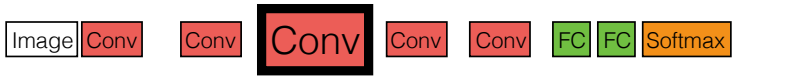
red



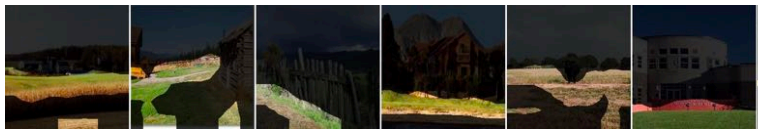
blue

Bau, Zhou, and et. al. Network Dissection: Quantifying Interpretability of Deep Visual Representations. CVPR 2017.

What did AlexNet learn?

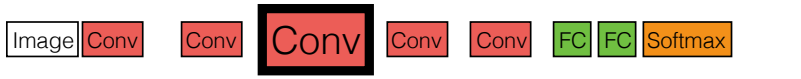


grass

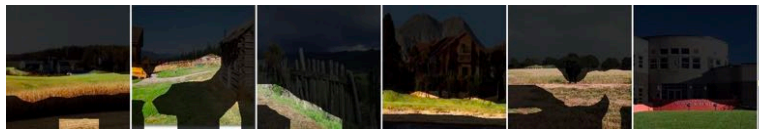


Bau, Zhou, and et. al. Network Dissection: Quantifying Interpretability of Deep Visual Representations. CVPR 2017.

What did AlexNet learn?



grass



sky

Bau, Zhou, and et. al. Network Dissection: Quantifying Interpretability of Deep Visual Representations. CVPR 2017.

What did AlexNet learn?

dog

Bau, Zhou, and et. al. Network Dissection: Quantifying Interpretability of Deep Visual Representations. CVPR 2017.

What did AlexNet learn?

dog

cat

Bau, Zhou, and et. al. Network Dissection: Quantifying Interpretability of Deep Visual Representations. CVPR 2017.

CNNs: LeNet

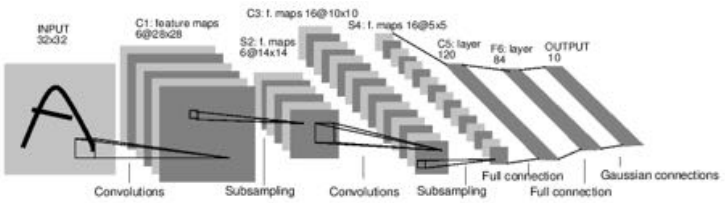


Image → Conv → Pool → Conv → Pool → FC → FC → Softmax

LeCun et. al. Handwritten digit recognition: Applications of neural net chips and automatic learning. IEEE Communication 1989.

CNNs: AlexNet

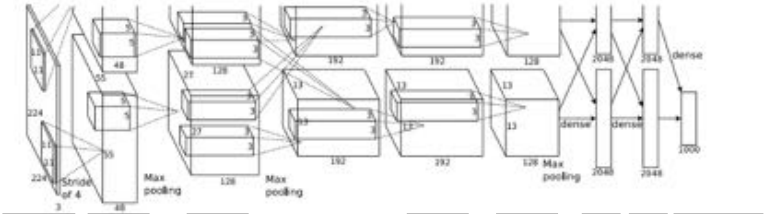


Image → Conv → Pool → Conv → Pool → Conv → Conv → Conv → Pool → FC → FC → Softmax

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

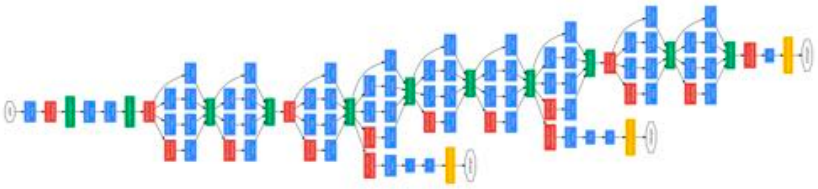
What made AlexNet AlexNet?

- GPU?
- Data?
- ReLU?
- Convolutional?
- Hyperparameter?

CNNs: VGGNet

Simonyan and Zisserman. Very Deep Convolutional Networks for Large-scale Image Recognition. ICLR 2015.

CNNs: GoogLeNet



Szegedy, et. al. Going Deeper with Convolution. CVPR 2015.

CNNs: ResNet

He, et. al. Deep **Residual** Learning for Image Recognition. CVPR 2016.

Encoder

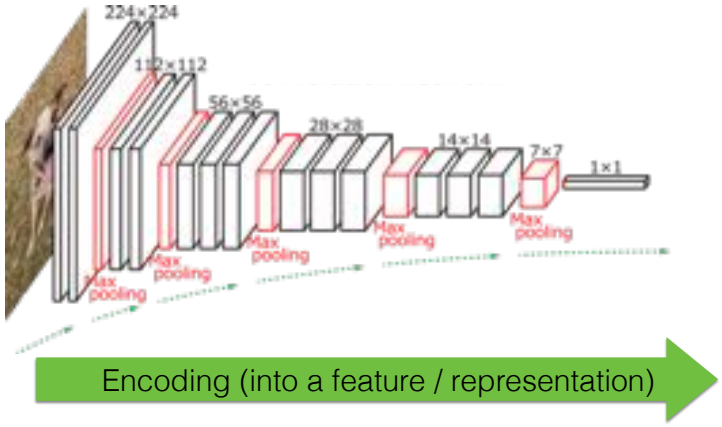


Image credit: Noh, et. al. Learning Deconvolution Network for Semantic Segmentation. ICCV 2015.

Decoder

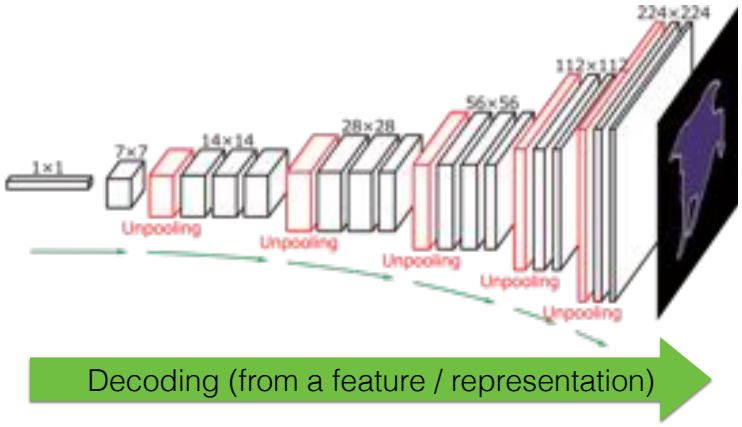


Image credit: Noh, et. al. Learning Deconvolution Network for Semantic Segmentation. ICCV 2015.

Congratulations!

All basic vocabs are covered.