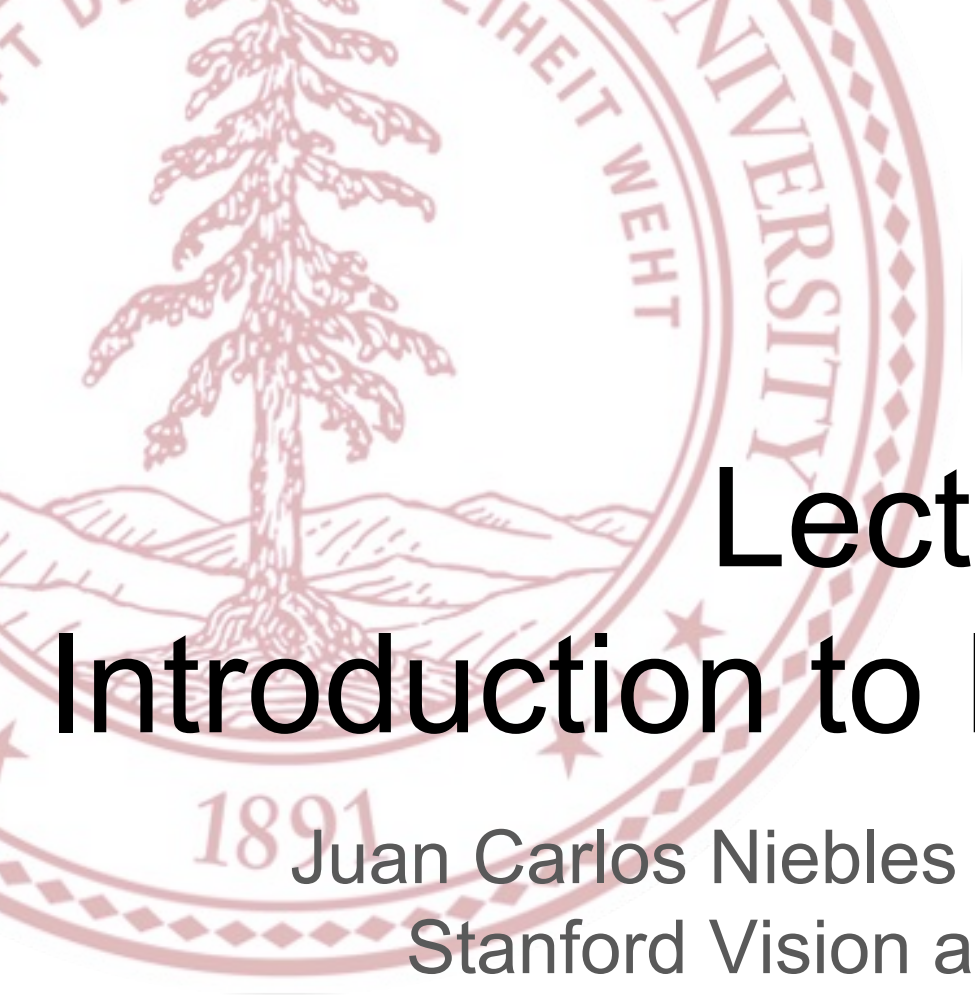


Lecture: Introduction to Deep Learning

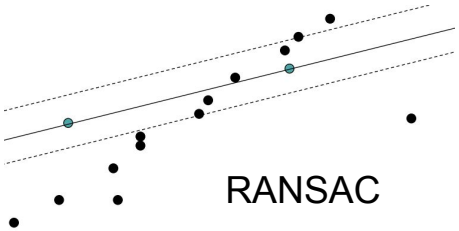
Juan Carlos Niebles and Ranjay Krishna
Stanford Vision and Learning Lab

Slides adapted from Justin Johnson

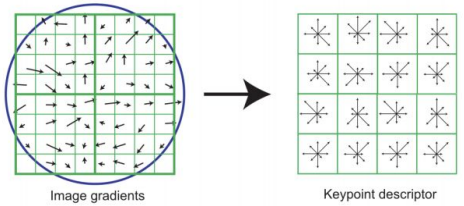
So far this quarter

Edge detection

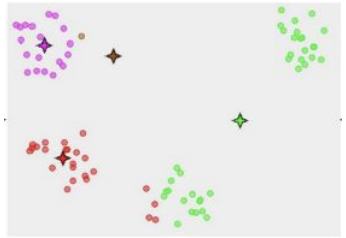
RANSAC



SIFT



K-Means

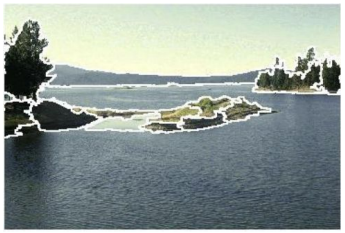


Linear classifiers

$$f(x, W) = Wx$$

10 numbers, indicating class scores

Mean-shift



PCA / Eigenfaces

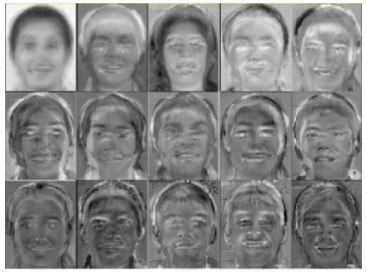
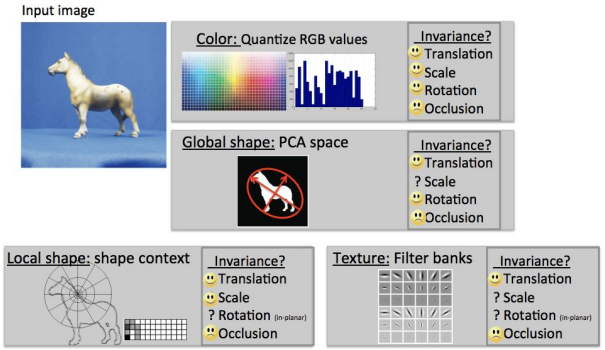


Image features



Current Research

ECCV 2016

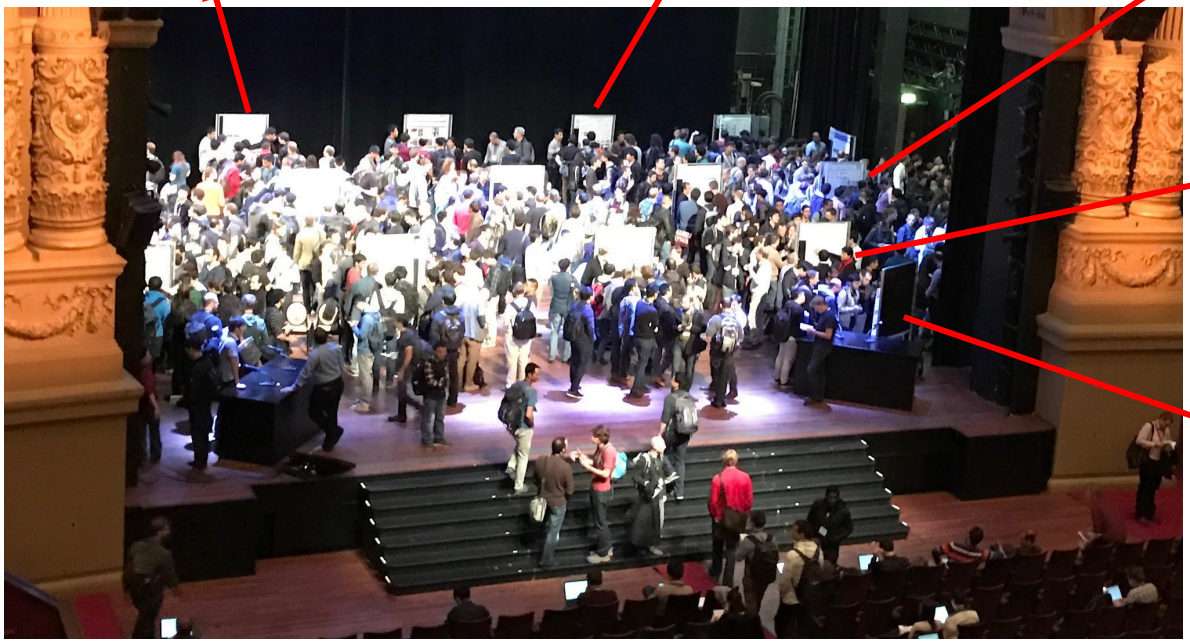
(photo credit: Justin Johnson)

Current Research

Convolutional neural networks

Recurrent neural networks

Autoencoders



Fully convolutional networks

Multilayer perceptrons

ECCV 2016

(photo credit: Justin Johnson)

What are these things?

Why are they important?

Deep Learning

- Learning hierarchical representations from data
- End-to-end learning: raw inputs to predictions
- Can use a small set of simple tools to solve many problems
- Has led to rapid progress on many problems
- (**Very loosely!**) Inspired by the brain
- Today: 1 hour introduction to deep learning fundamentals
- To learn more take CS 231n in the Spring

Deep learning for many different problems

IMAGENET Large Scale Visual Recognition Challenge

Steel drum

The Image Classification Challenge:

1,000 object classes

1,431,167 images

Output:
Scale
T-shirt
Steel drum
Drumstick
Mud turtle

✓

Output:
Scale
T-shirt
Giant panda
Drumstick
Mud turtle

✗

Russakovsky et al. arXiv, 2014

IMAGENET Large Scale Visual Recognition Challenge

Year 2010

NEC-UIUC

Dense grid descriptor: HOG, LBP

Coding: local coordinate, super-vector

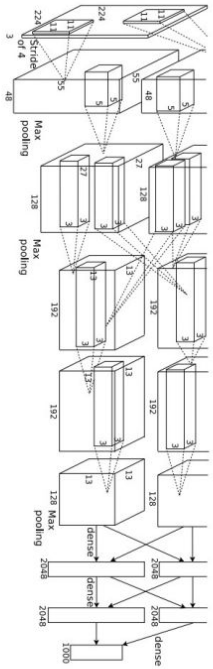
Pooling, SPM

Linear SVM

[Lin CVPR 2011]

Year 2012

SuperVision



[Krizhevsky NIPS 2012]

Year 2014

GoogLeNet

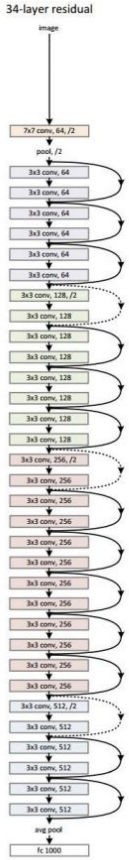
[Szegedy arxiv 2014]

VGG

[Simonyan arxiv 2014]

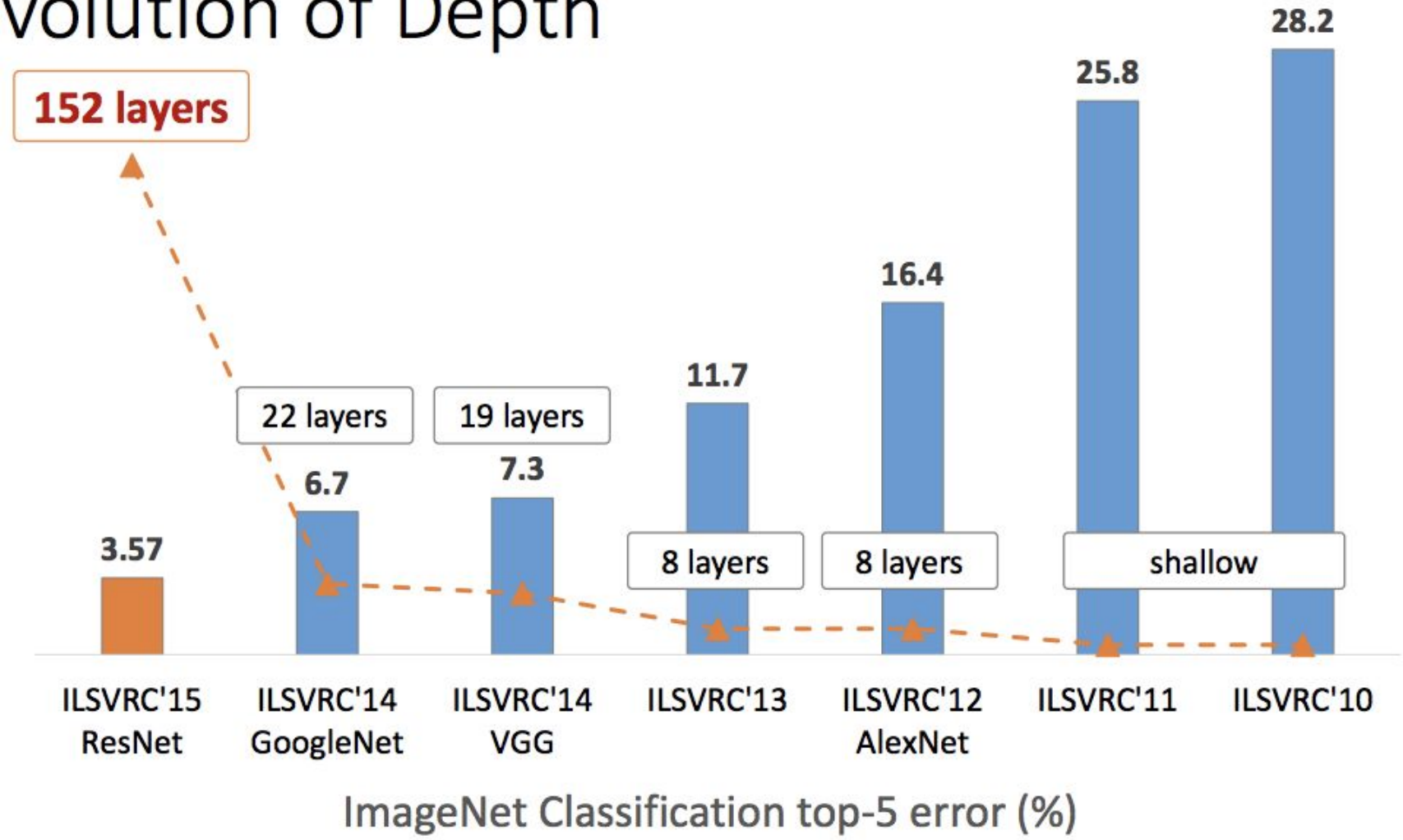
Year 2015

MSRA

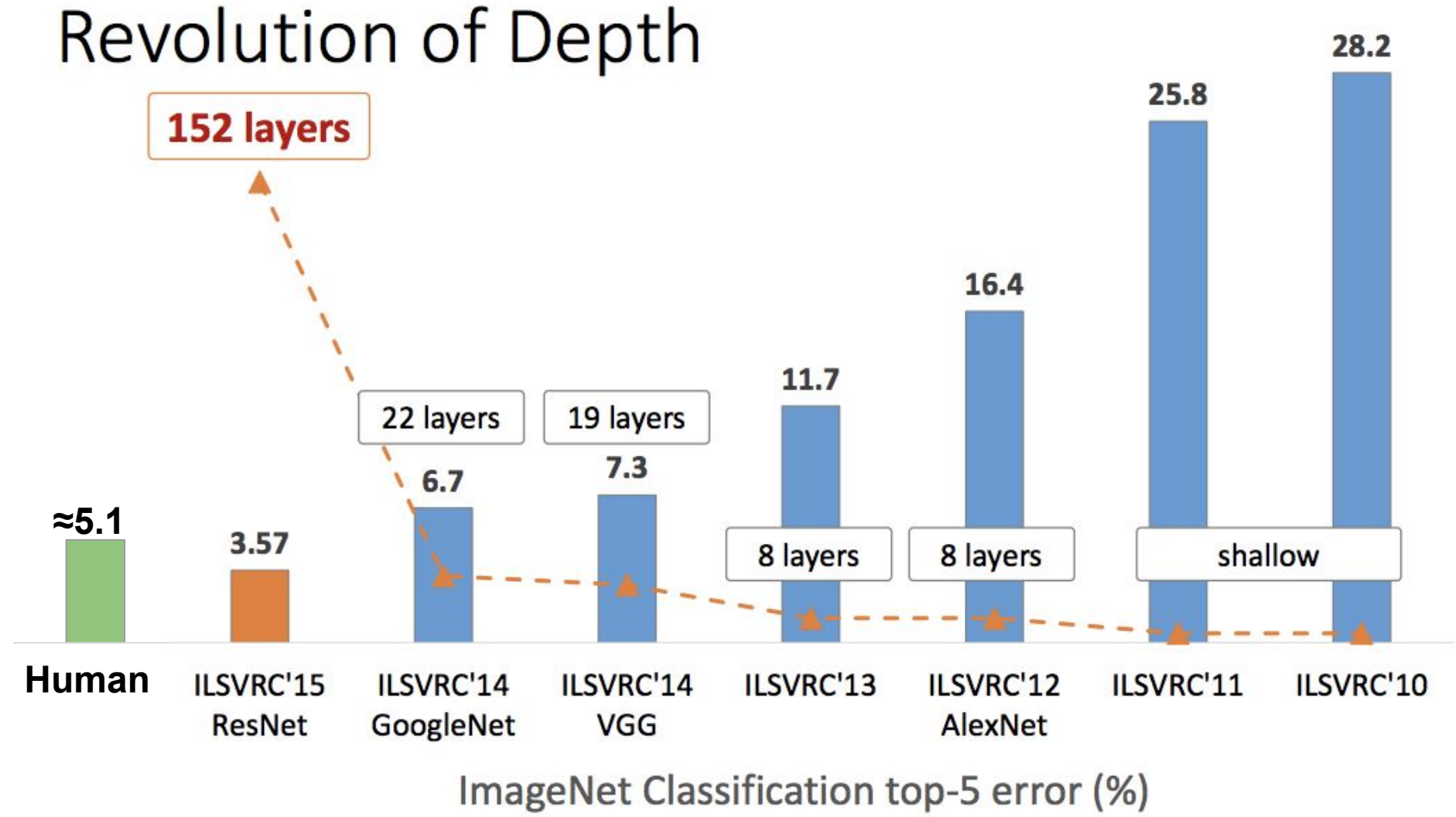


Slide credit: CS 231n, Lecture 1, 2016

Revolution of Depth



Slide credit: Kaiming He, ICCV 2015



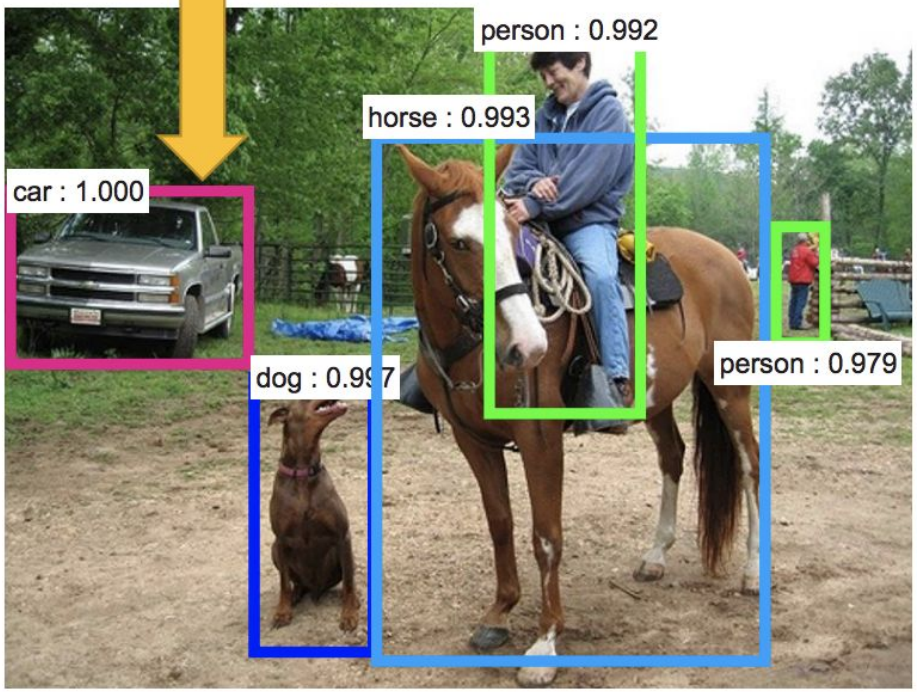
Human benchmark from Russakovsky et al, "ImageNet Large Scale Visual Recognition Challenge", IJCV 2015

Slide credit: Kaiming He, ICCV 2015

Object Detection = What, and Where

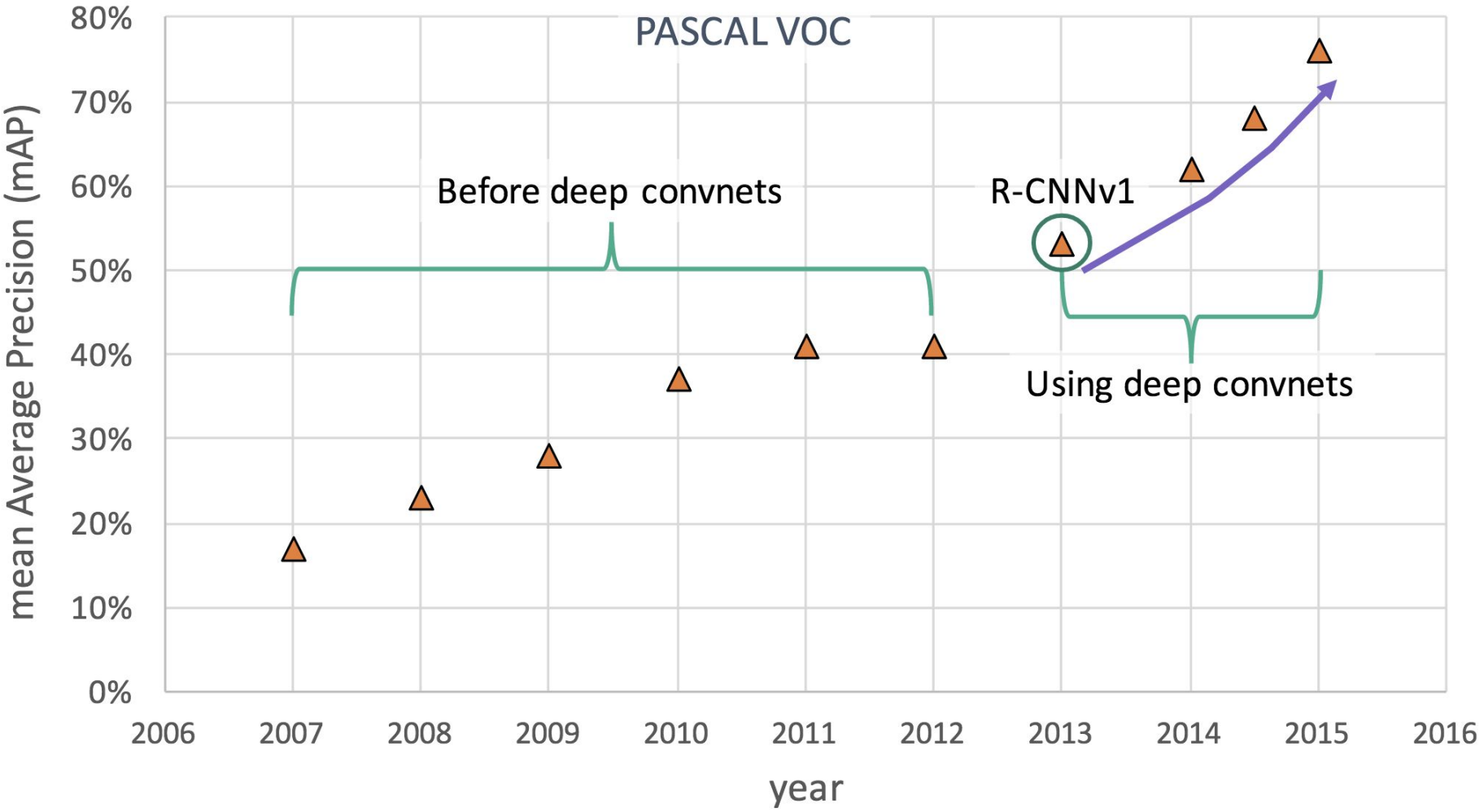
Localization
Where?

Recognition
What?



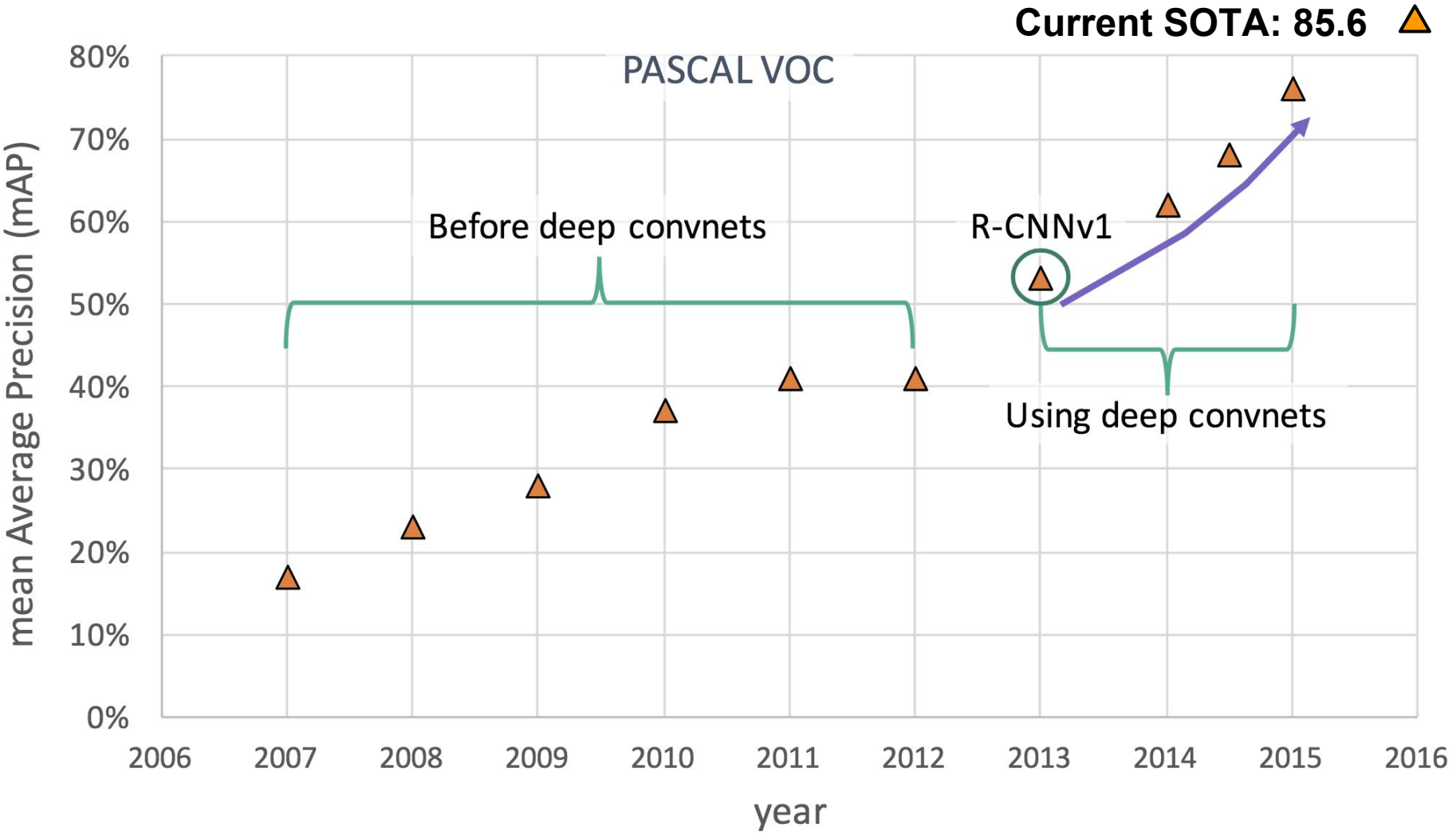
Slide credit: Kaiming He, ICCV 2015

Deep Learning for Object Detection



Slide credit: Ross Girshick, ICCV 2015

Deep Learning for Object Detection



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

Slide credit: Ross Girshick, ICCV 2015

Object segmentation

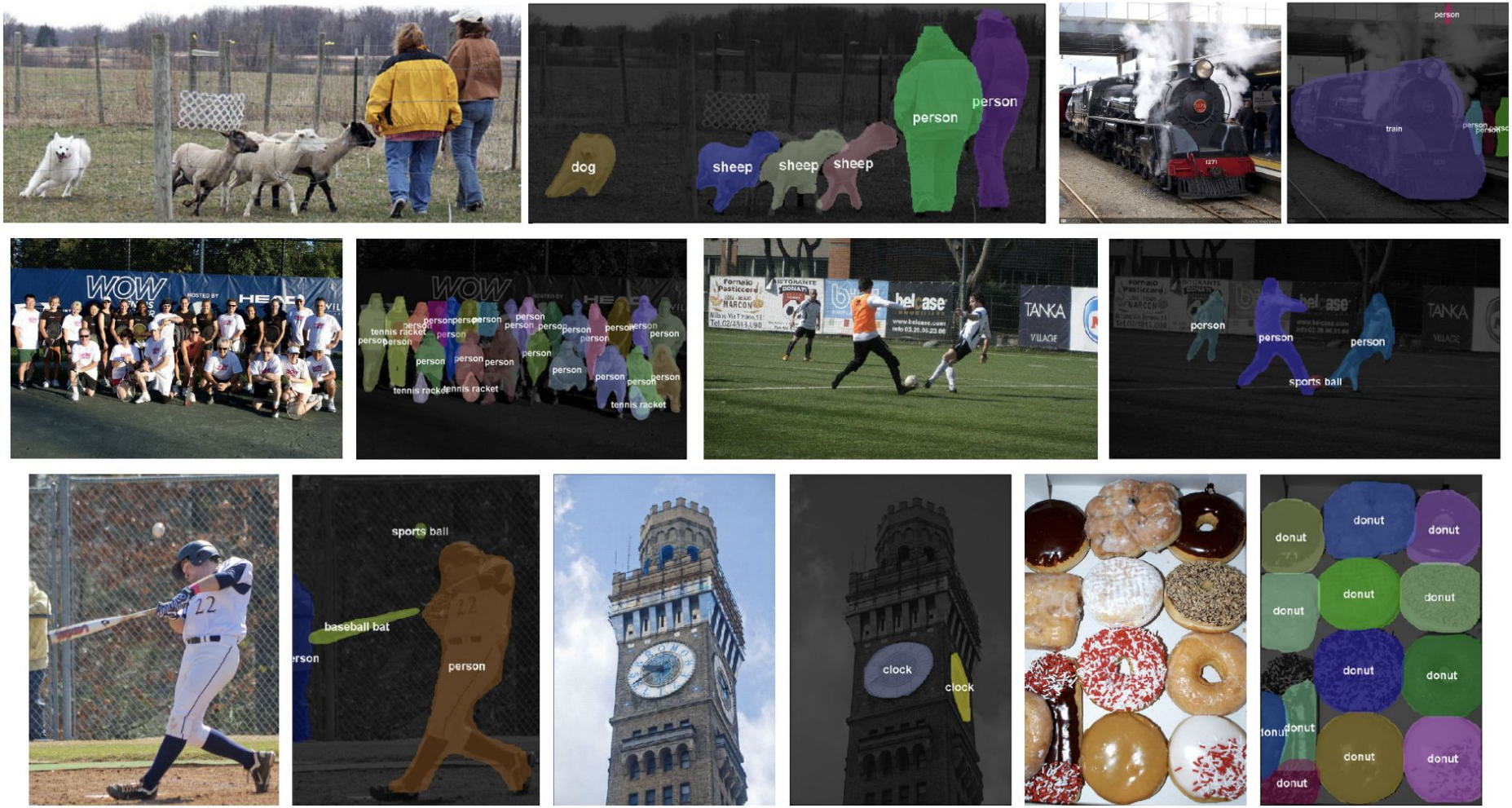
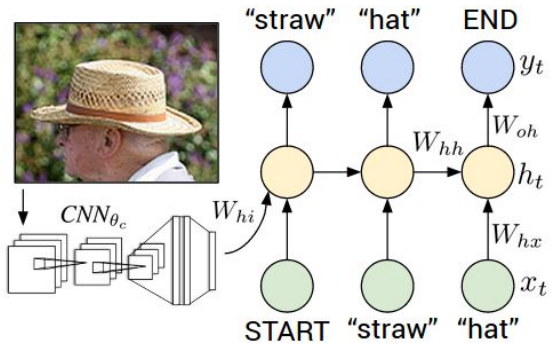


Figure credit: Dai, He, and Sun, “Instance-aware Semantic Segmentation via Multi-task Network Cascades”, CVPR 2016

Pose Estimation

Figure credit: Cao et al, “Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields”, arXiv 2016

Deep Learning for Image Captioning



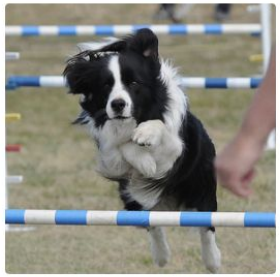

"man in black shirt is playing guitar."


"construction worker in orange safety vest is working on road."


"two young girls are playing with lego toy."


"boy is doing backflip on wakeboard."


"girl in pink dress is jumping in air."

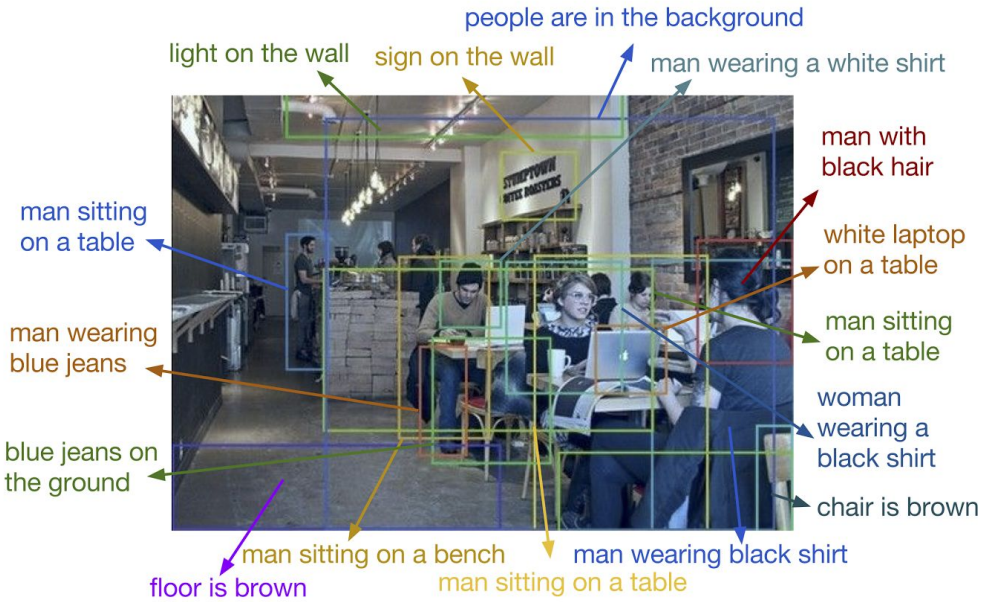

"black and white dog jumps over bar."


"young girl in pink shirt is swinging on swing."


"man in blue wetsuit is surfing on wave."

Figure credit: Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015

Dense Image Captioning



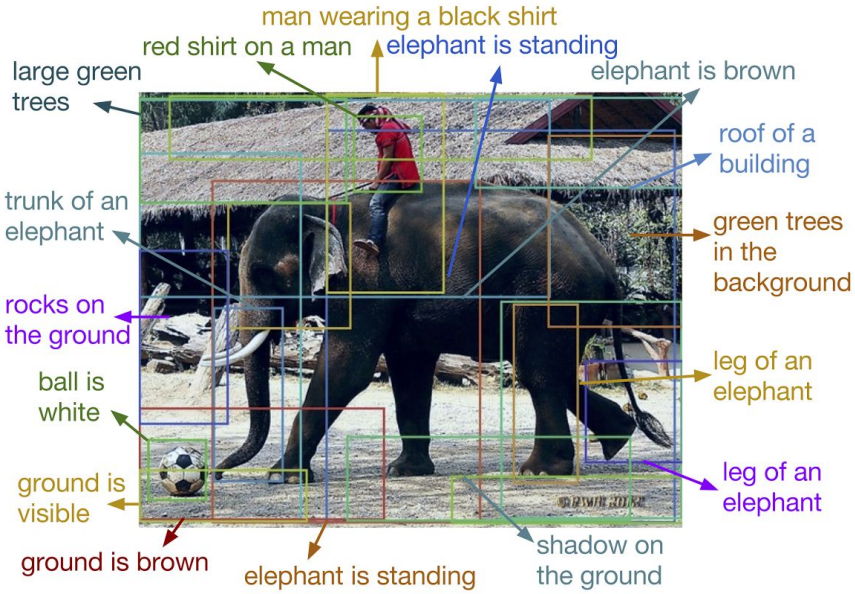


Figure credit: Johnson*, Karpathy*, and Fei-Fei, "DenseCap: Fully Convolutional Localization Networks for Dense Captioning", CVPR 2016

Visual Question Answering

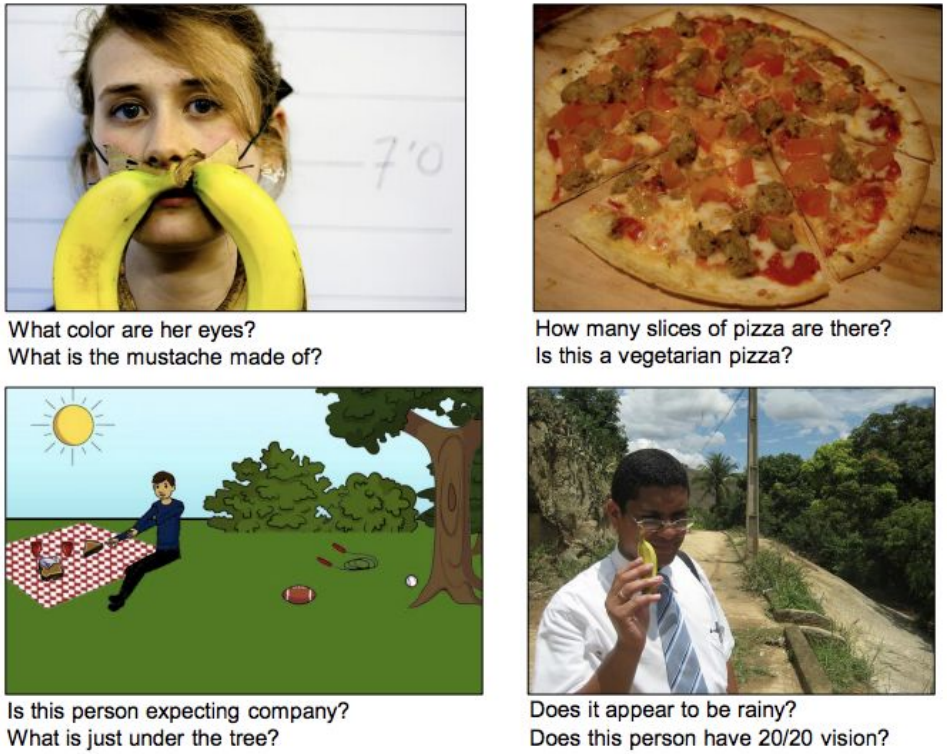


Figure credit: Agrawal et al, "VQA: Visual Question Answering", ICCV 2015

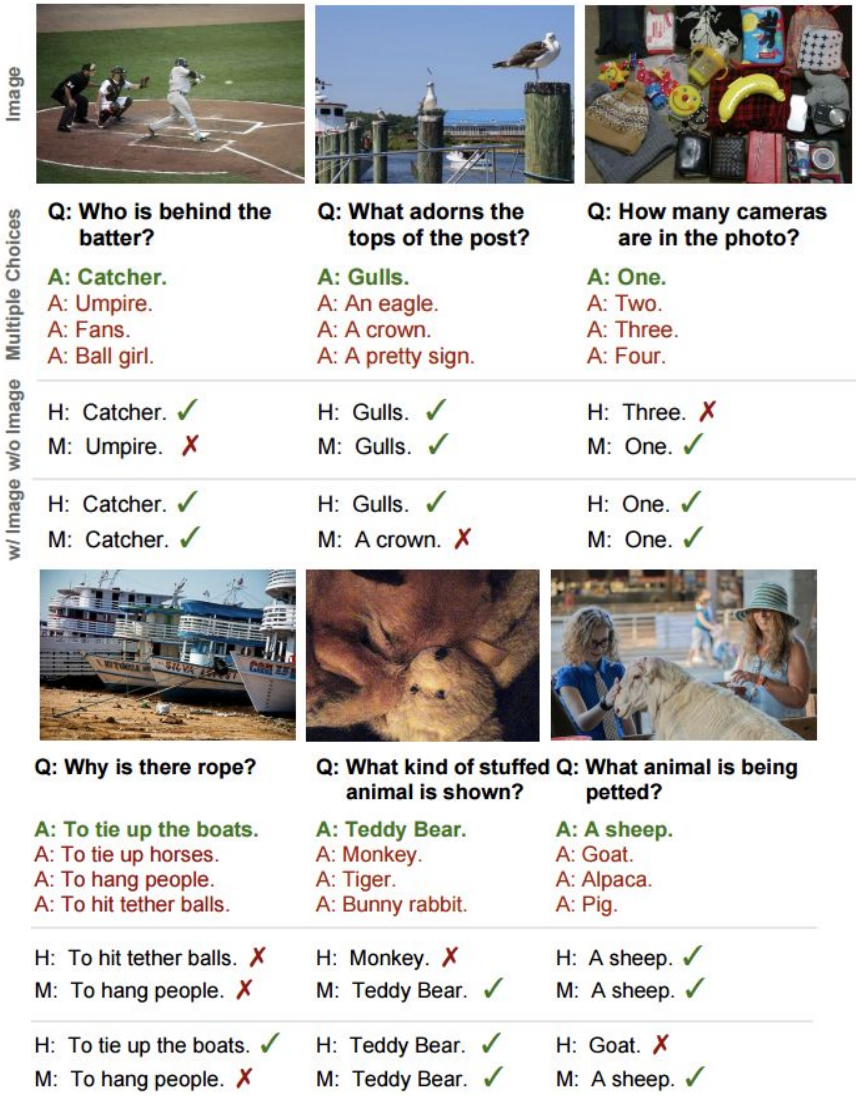


Figure credit: Zhu et al, "Visual7W: Grounded Question Answering in Images", CVPR 2016

Image Super-Resolution

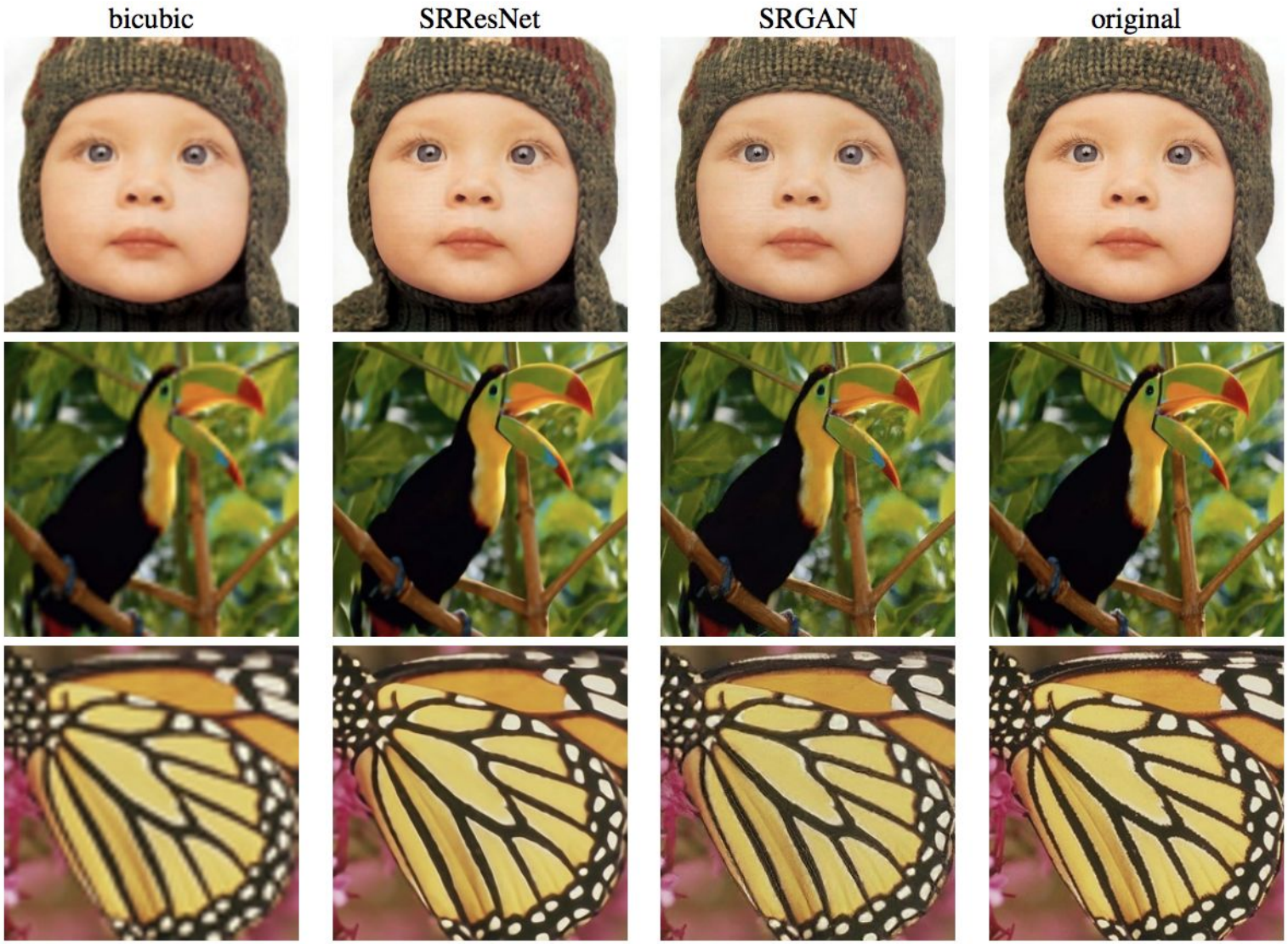


Figure credit: Ledig et al, "Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network", arXiv 2016

Generating Art

Figure credit: Gatys, Ecker, and Bethge, “Image Style Transfer using Convolutional Neural Networks”, CVPR 2016

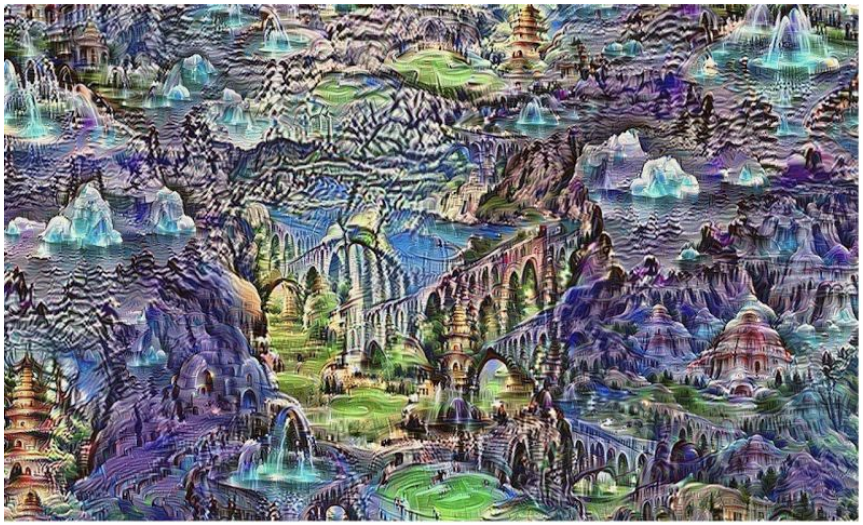


Figure credit: Mordvintsev, Olah, and Tyka, “Inceptionism: Going Deeper into Neural Networks”, https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html

Figure credit: Johnson, Alahi, and Fei-Fei: “Perceptual Losses for Real-Time Style Transfer and Super-Resolution”, ECCV 2016, https://github.com/jcjohnson/fast-neural-style

Outside Computer Vision

Machine Translation

Source	An admitting privilege is the right of a doctor to admit a patient to a hospital or a medical centre to carry out a diagnosis or a procedure, based on his status as a health care worker at a hospital.
Reference	Le privilège d'admission est le droit d'un médecin, en vertu de son statut de membre soignant d'un hôpital, d'admettre un patient dans un hôpital ou un centre médical afin d'y délivrer un diagnostic ou un traitement.
RNNenc-50	Un privilège d'admission est le droit d'un médecin de reconnaître un patient à l'hôpital ou un centre médical d'un diagnostic ou de prendre un diagnostic en fonction de son état de santé.
RNNsearch-50	Un privilège d'admission est le droit d'un médecin d'admettre un patient à un hôpital ou un centre médical pour effectuer un diagnostic ou une procédure, selon son statut de travailleur des soins de santé à l'hôpital.
Google Translate	Un privilège admettre est le droit d'un médecin d'admettre un patient dans un hôpital ou un centre médical pour effectuer un diagnostic ou une procédure, fondée sur sa situation en tant que travailleur de soins de santé dans un hôpital.

Figure credit: Bahdanau, Cho, and Bengio, "Neural Machine Translation by jointly learning to align and translate", ICLR 2015

Text Synthesis

Recurrent network with the Stiefel information for logistic regression methods Along with either of the algorithms previously (two or more skewprecision) is more similar to the model with the same average mismatched graph. Though this task is to be studied under the reward transform, such as (c) and (C) from the training set, based on target activities for articles a ? 2(6) and (4.3). The PHDPic (PDB) matrix of cav'va using the three relevant information contains for tieming measurements. Moreover, because of the therap tor, the aim is to improve the score to the best patch randomly, but for each initially four data sets. As shown in Figure 11, it is more than 100 steps, we used ?? \to \infty with 1000

Figure credit: Sutskever, Martens, and Hinton, "Generating Text with Recurrent Neural Networks", ICML 2011

Speech Recognition

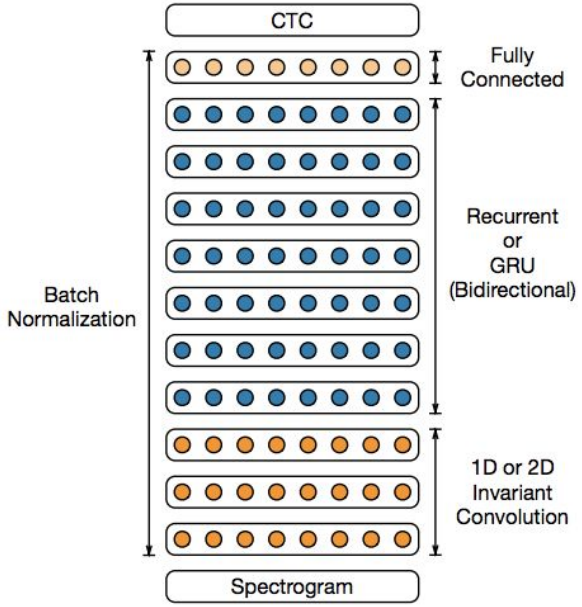


Figure credit: Amodei et al, "Deep Speech 2: End-to-End Speech Recognition in English and Mandarin", arXiv 2015

Speech Synthesis

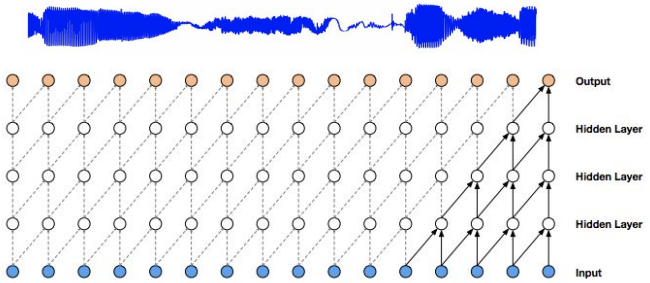
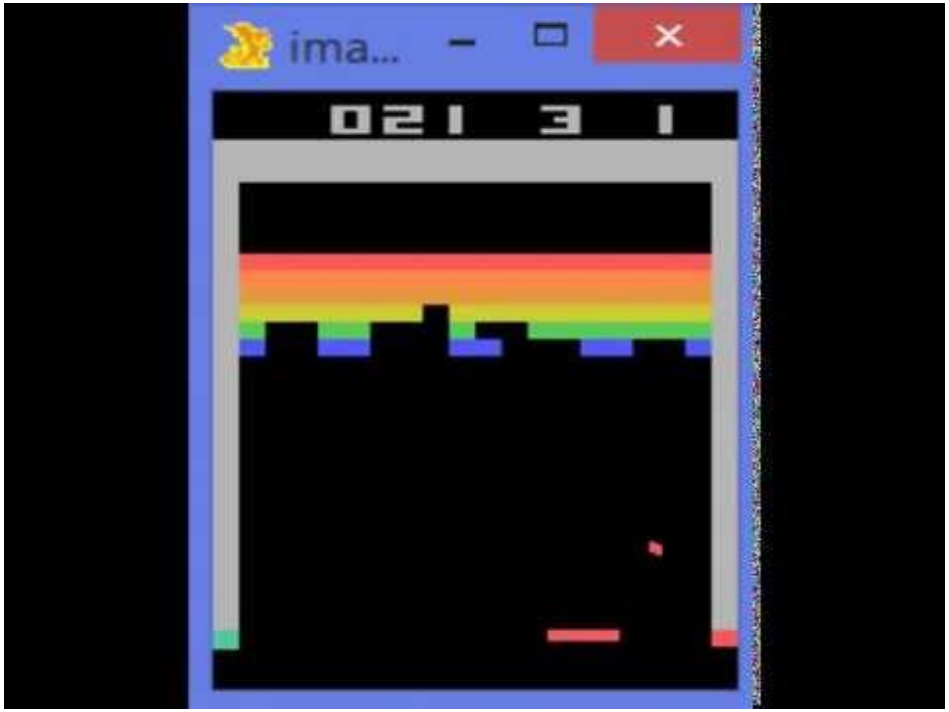


Figure credit: var der Oord et al, "WaveNet: A Generative Model for Raw Audio", arXiv 2016, https://deepmind.com/blog/wavenet-generative-model-raw-audio/

Deep Reinforcement Learning

Playing Atari games



Mnih et al, “Human-level control through deep reinforcement learning”, Nature 2015

AlphaGo beats Lee Sedol

Silver et al, “Mastering the game of Go with deep neural networks and tree search”, Nature 2016

Image credit: http://www.newyorker.com/tech/elements/alphago-lee-sedol-and-the-reassuring-future-of-humans-and-machines

Outline

- Applications
- Motivation
- Supervised learning
- Gradient descent
- Backpropagation
- Convolutional Neural Networks

Outline

- Applications
- Motivation
- Supervised learning
- Gradient descent
- Backpropagation
- Convolutional Neural Networks

Recall: Image Classification

Write a function that maps **images** to **labels**
(also called **object recognition**)

f() = “apple”

f() = “tomato”

f() = “cow”

Dataset: ETH-80, by B. Leibe; Slide credit: L. Lazebnik

```
def predict(image):
    # ????
    return class_label
```

No obvious way to
implement this function!

Slide credit: CS 231n, Lecture 2, 2016

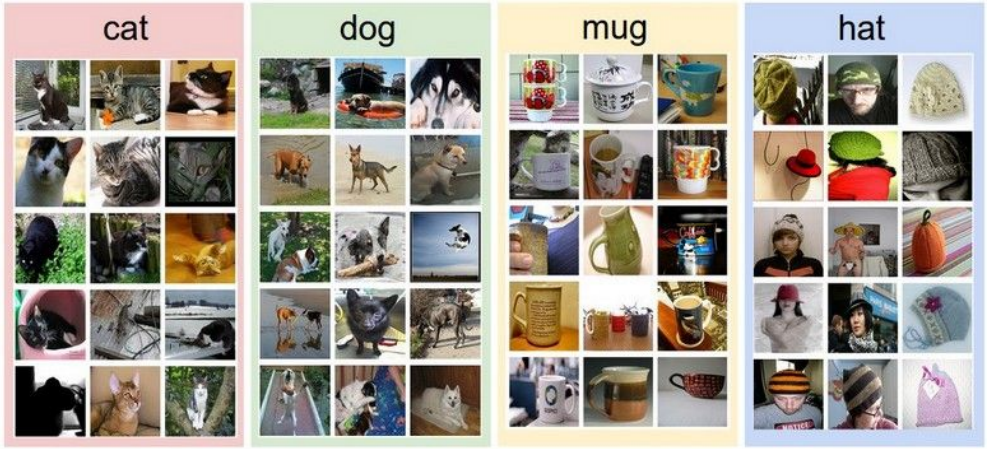
Recall: Image Classification

Data-driven approach:

1. Collect a dataset of images and labels
2. Use Machine Learning to train an image classifier
3. Evaluate the classifier on a withheld set of test images

```
def train(train_images, train_labels):
  # build a model for images -> labels...
  return model

def predict(model, test_images):
  # predict test_labels using the model...
  return test_labels
```

Example training set



Slide credit: CS 231n, Lecture 2, 2016

Recall: Image Classification

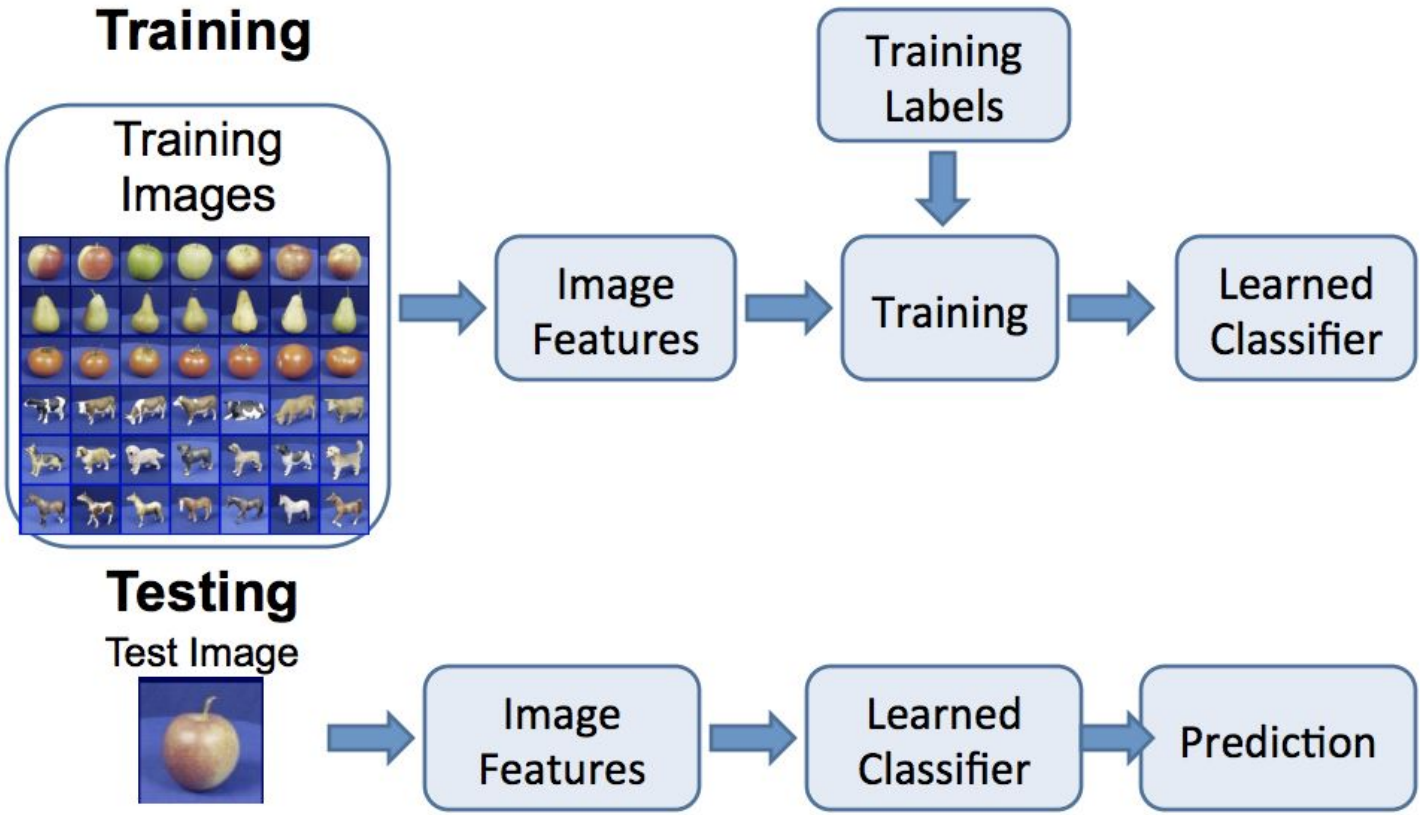


Figure credit: D. Hoiem, L. Lazebnik

Recall: Image Classification

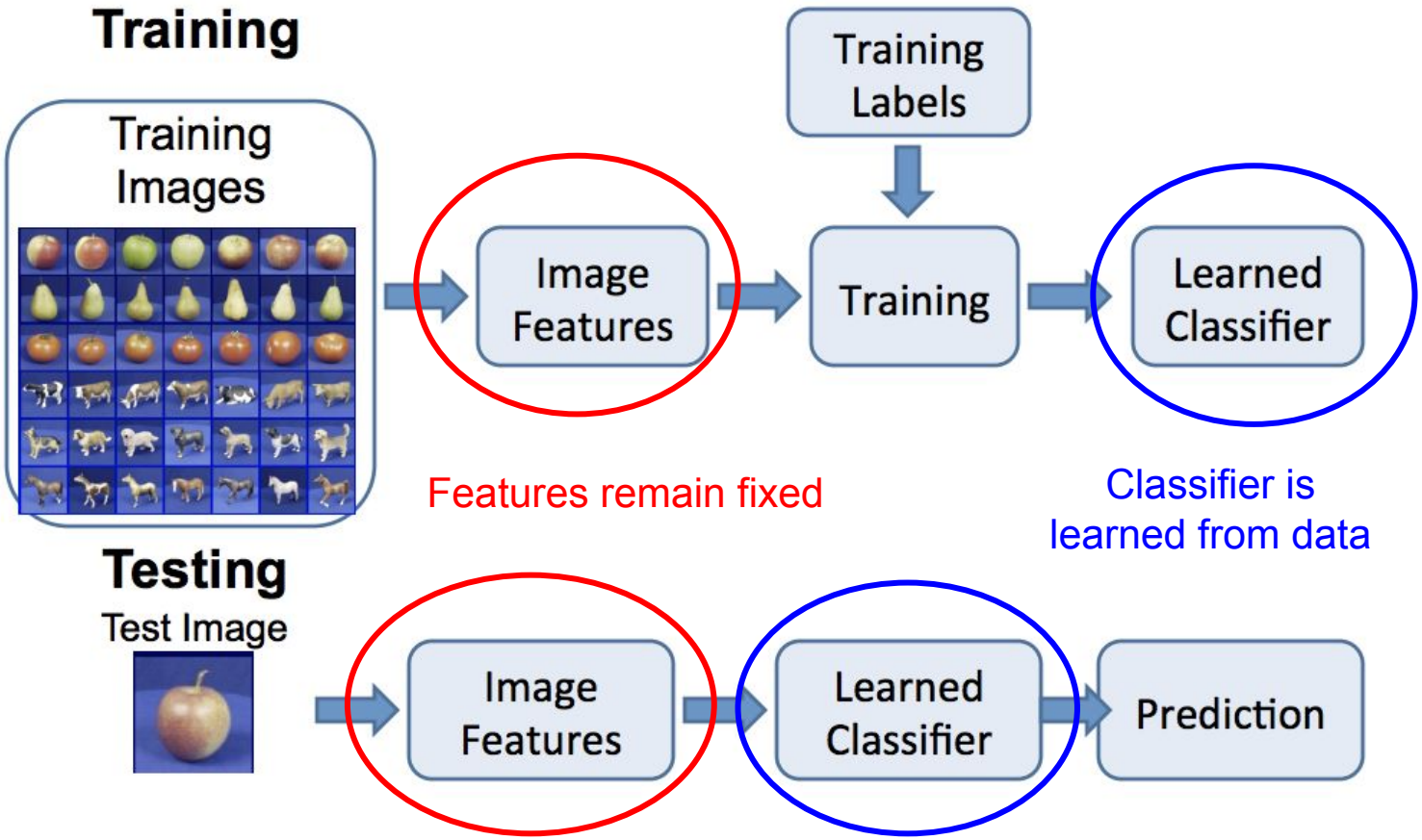
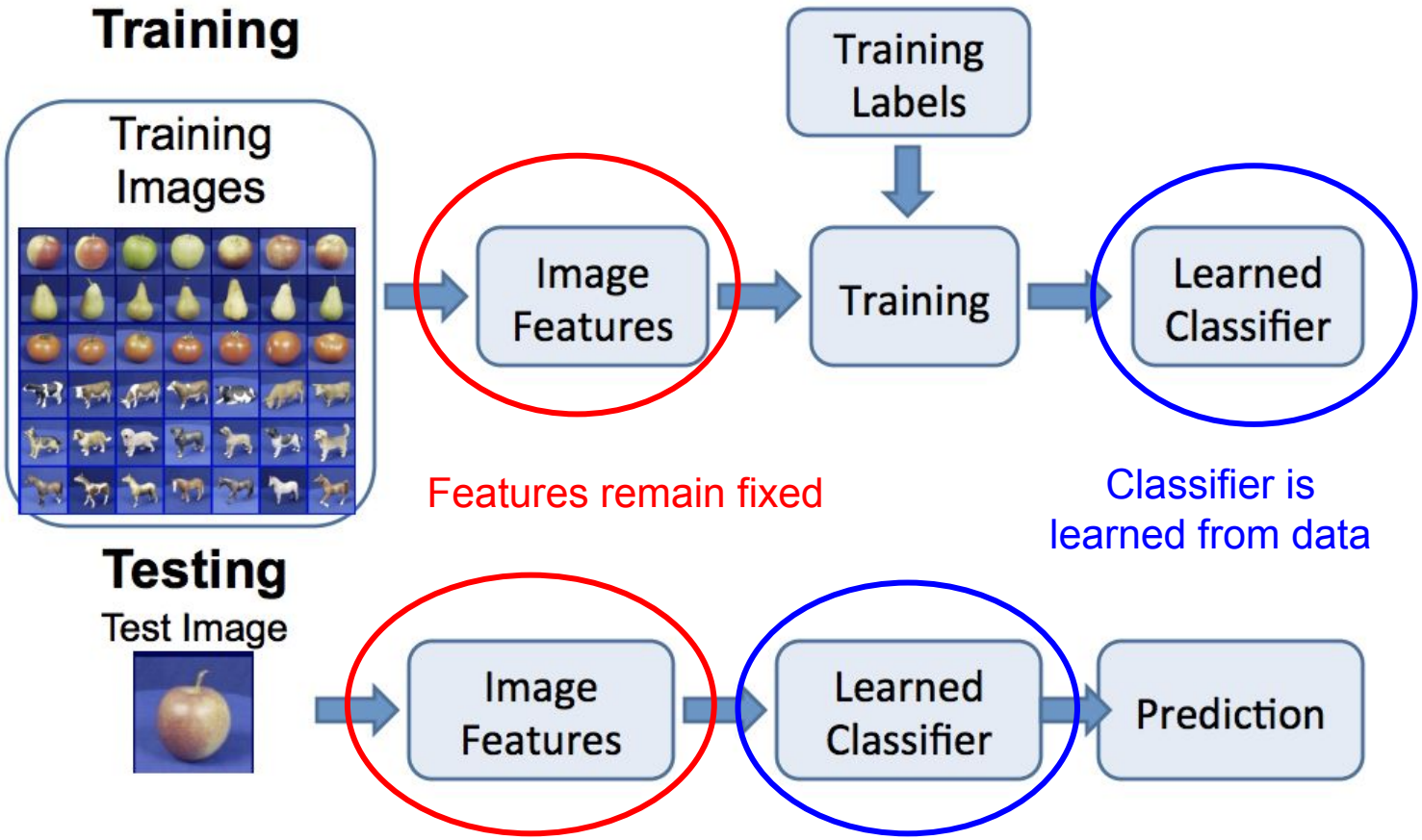


Figure credit: D. Hoiem, L. Lazebnik

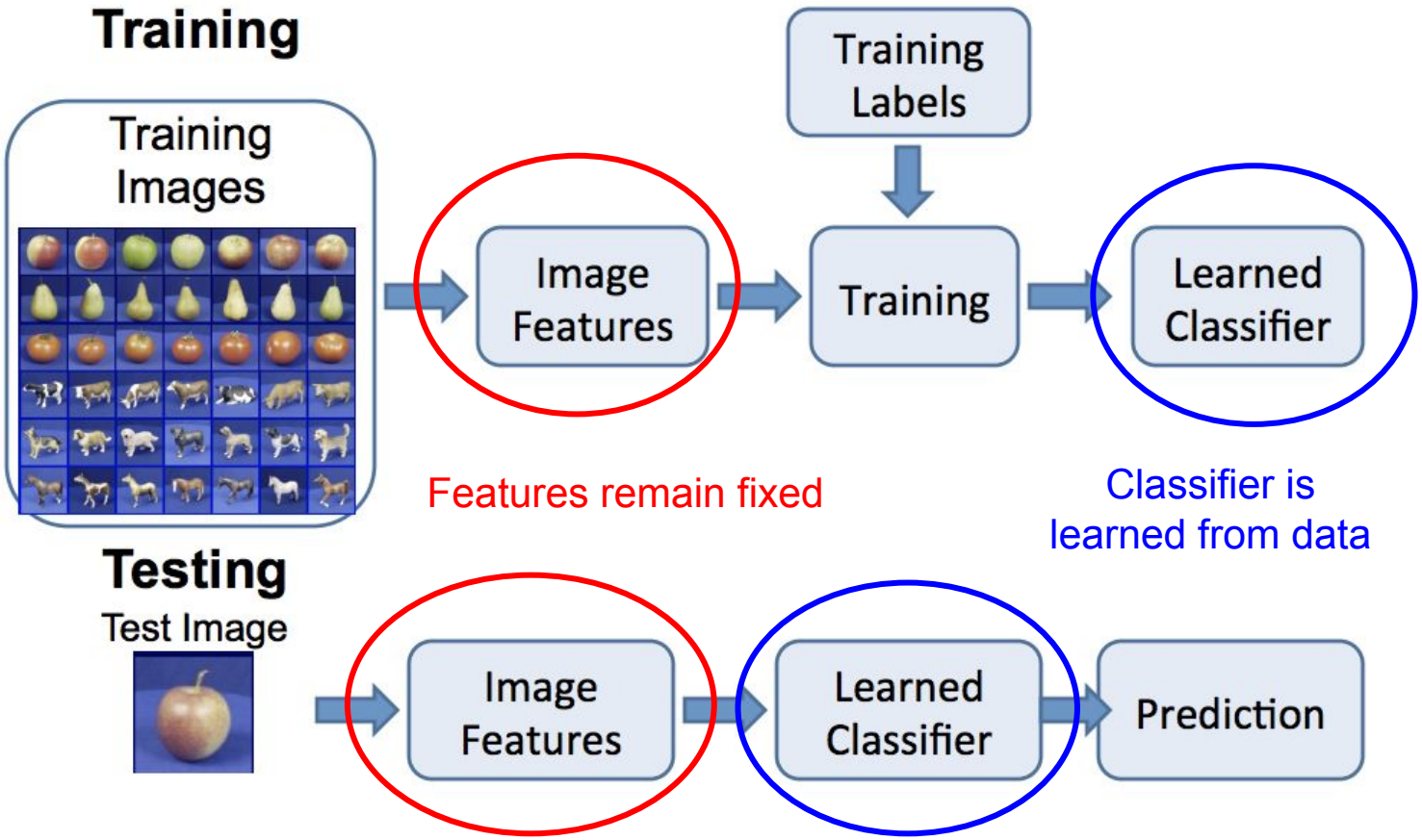
Recall: Image Classification



Problem:
How do we know which features to use? We may need different features for each problem!

Figure credit: D. Hoiem, L. Lazebnik

Recall: Image Classification



Problem:
How do we know which features to use? We may need different features for each problem!

Solution:
Learn the features jointly with the classifier!

Figure credit: D. Hoiem, L. Lazebnik

Image Classification: Feature Learning

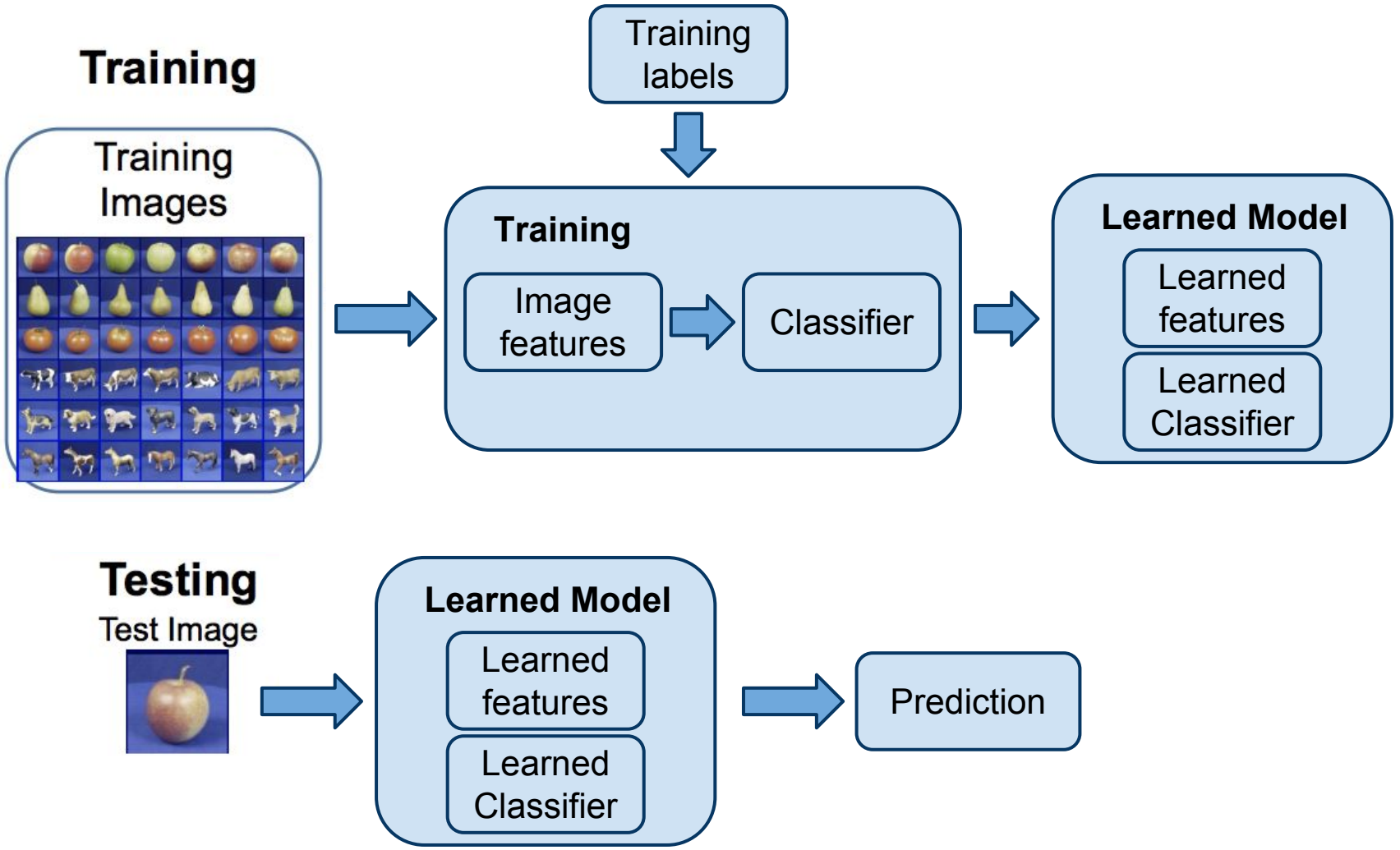
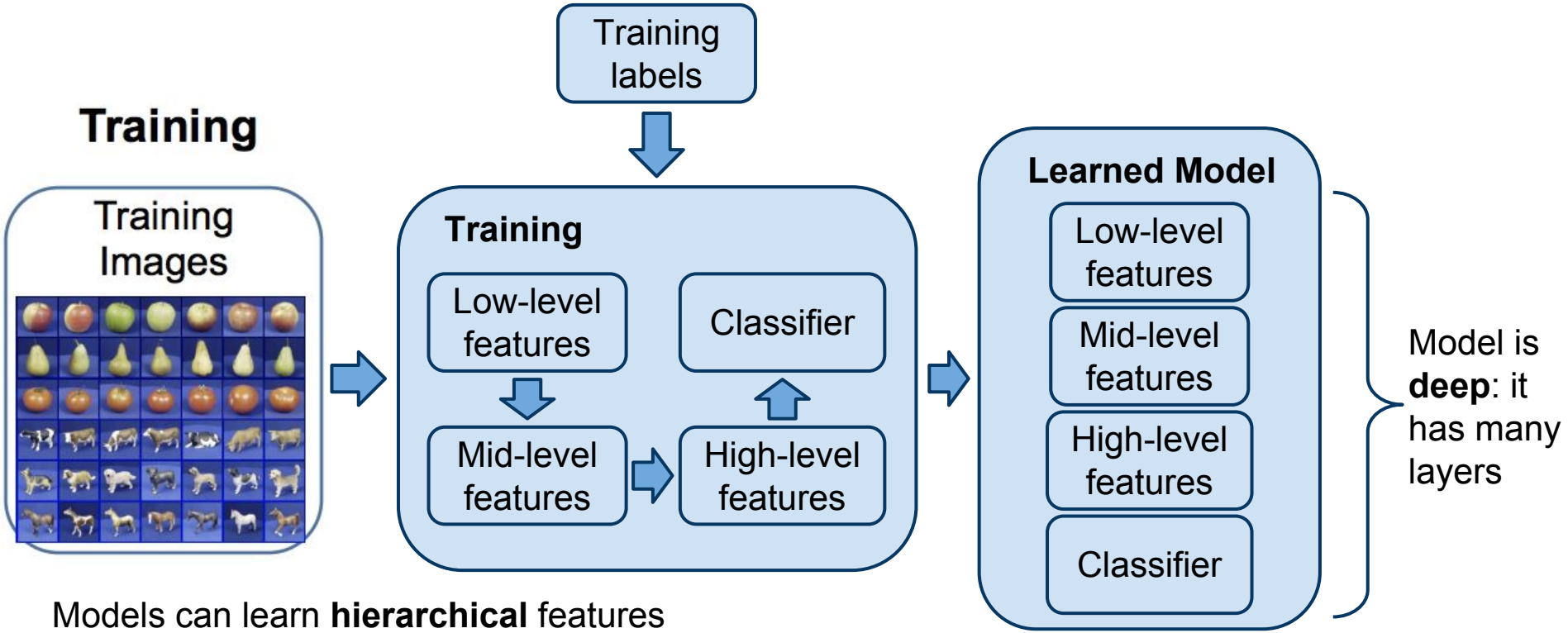
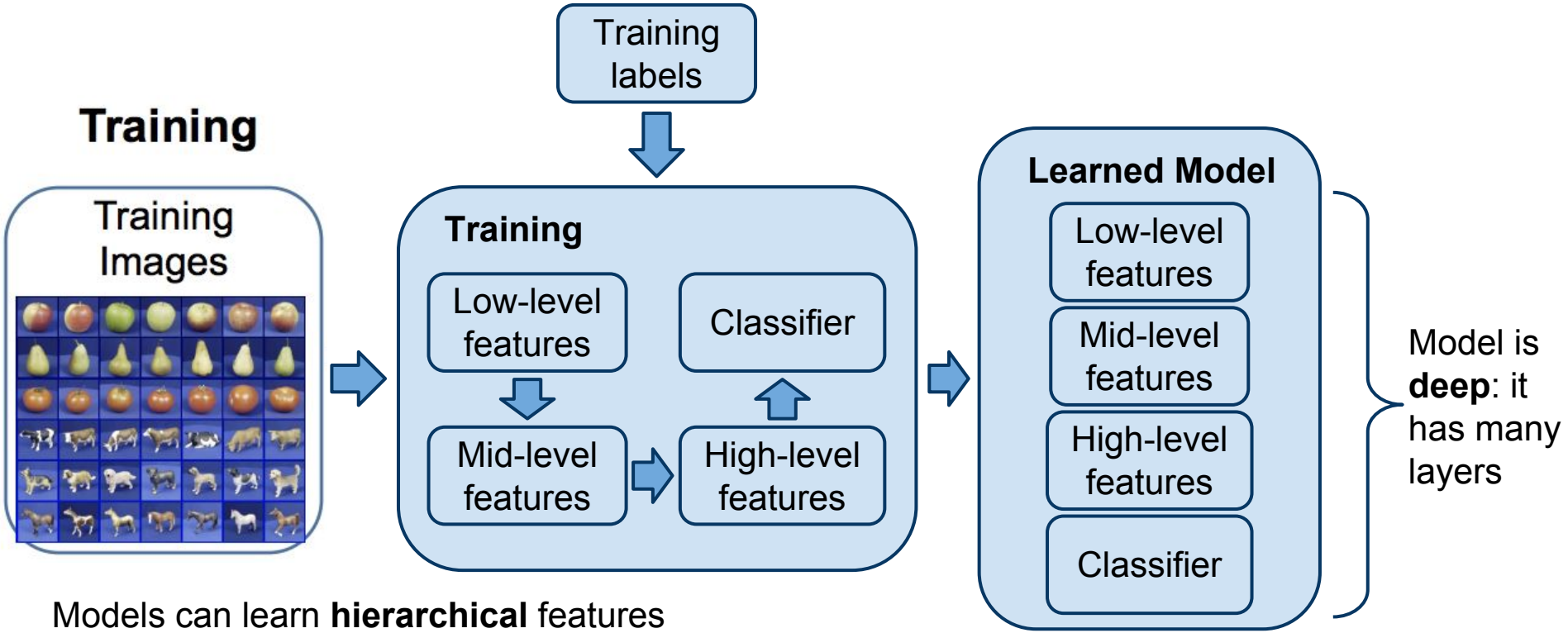


Image Classification: Deep Learning



Models can learn **hierarchical** features

Image Classification: Deep Learning



Models can learn **hierarchical** features

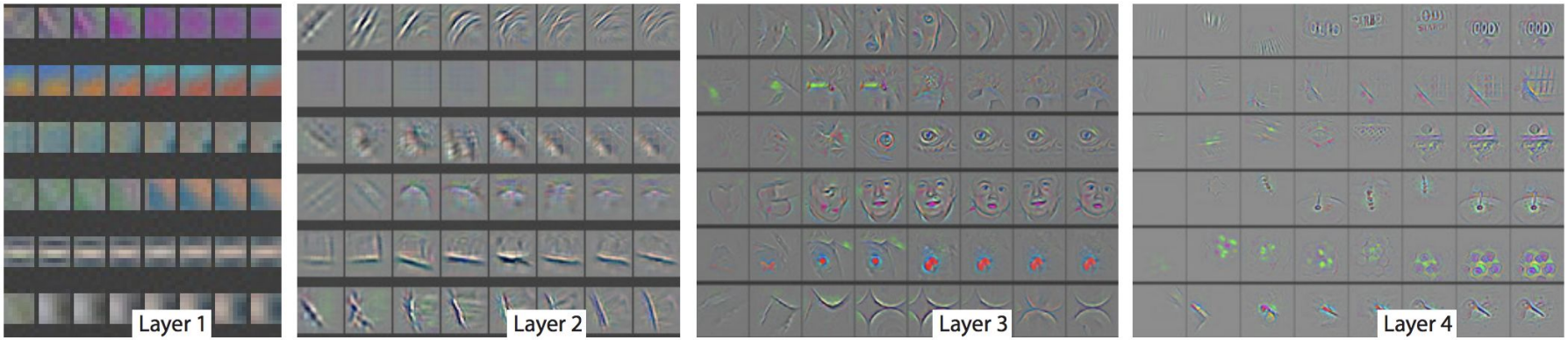


Figure credit: Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014

Outline

- Applications
- Motivation
- Supervised learning
- Gradient descent
- Backpropagation
- Convolutional Neural Networks

Supervised Learning in 3 easy steps

How to learn models from data

Step 1:
Define Model

$$\hat{y} = f(x, w)$$

Supervised Learning in 3 easy steps

How to learn models from data

Step 1:
Define Model

$$\hat{y} = f(x, w)$$

Predicted output (image label)

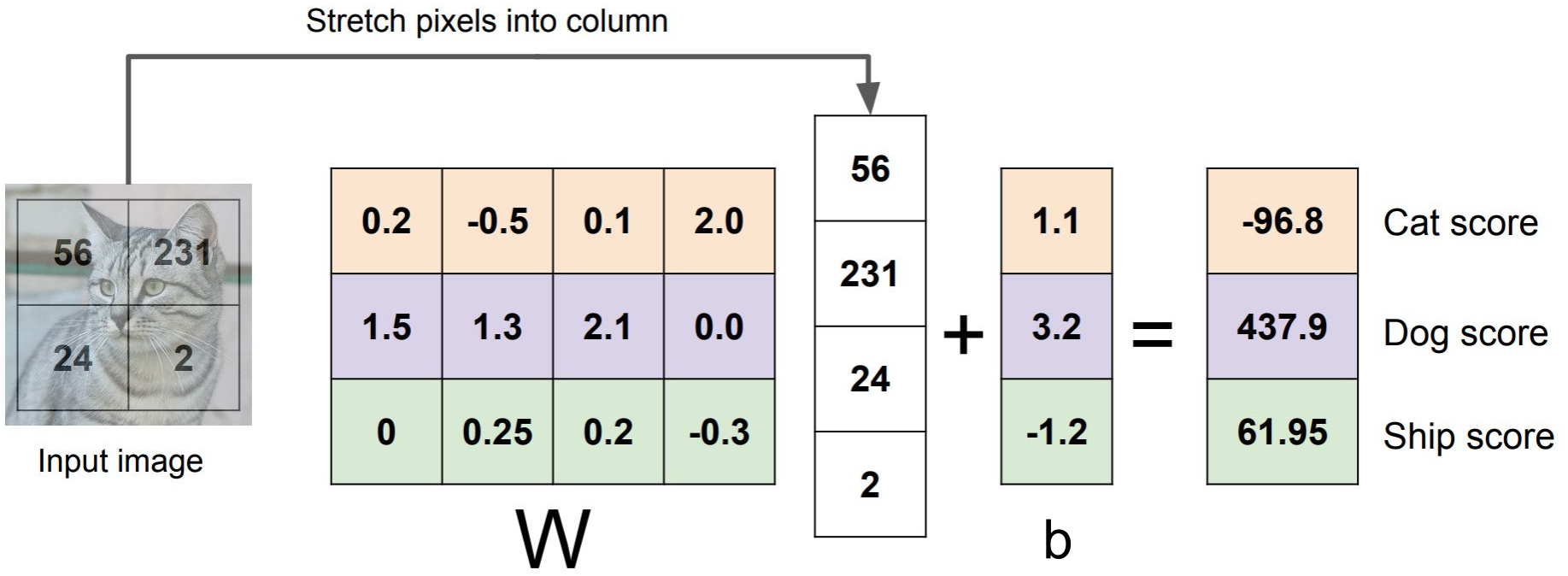
Model structure

Input data (Image pixels)

Model weights

Supervised learning

Example with an image with 4 pixels, and 3 classes (cat/dog/ship)



Supervised Learning in 3 easy steps

How to learn models from data

Step 1:
Define Model

$$\hat{y} = f(x, w)$$

Predicted output (image label)

Model structure

Input data (Image pixels)

Model weights

Step 2: Collect data

$$\{(x_i, y_i)\}_{i=1}^{N}$$

Training input

True output

Supervised Learning in 3 easy steps

How to learn models from data

Step 1: Define Model

$$\hat{y} = f(x, w)$$

Predicted output (image label)

Model structure

Input data (Image pixels)

Model weights

Step 2: Collect data

$$\{(x_i, y_i)\}_{i=1}^{N}$$

Training input

True output

Step 3: Learn the model

$$w^* = \arg\min_w \frac{1}{N} \sum_{i=1}^{N} \ell(f(x_i, w), y_i) + R(w)$$

Supervised Learning in 3 easy steps

How to learn models from data

Step 1: Define Model

$$\hat{y} = f(x, w)$$

Predicted output (image label)

Model structure

Input data (Image pixels)

Model weights

Step 2: Collect data

$$\{(x_i, y_i)\}_{i=1}^{N}$$

Training input

True output

Step 3: Learn the model

$$w^* = \arg\min_w \frac{1}{N} \sum_{i=1}^{N} \ell(f(x_i, w), y_i) + R(w)$$

Predicted output

Learned weights

Minimize average loss over training set

Loss function: Measures "badness" of prediction

Regularizer: Penalizes complex models

Supervised Learning: Linear regression

Sometimes called “Ridge Regression” with regularizer

$$x_i \in \mathbb{R}^{D_{in}} \qquad y_i \in \mathbb{R}^{D_{out}}$$

Input and output are vectors

$$\ell(\hat{y}, y) = \frac{1}{2}\|\hat{y} - y\|_2^2$$

Loss is Euclidean distance

Linear Regression

$$f(x, W) = Wx$$

$$W \in \mathbb{R}^{D_{out} \times D_{in}}$$

Model is just a matrix multiply

$$R(W) = \lambda \|W\|_{fro}^2$$

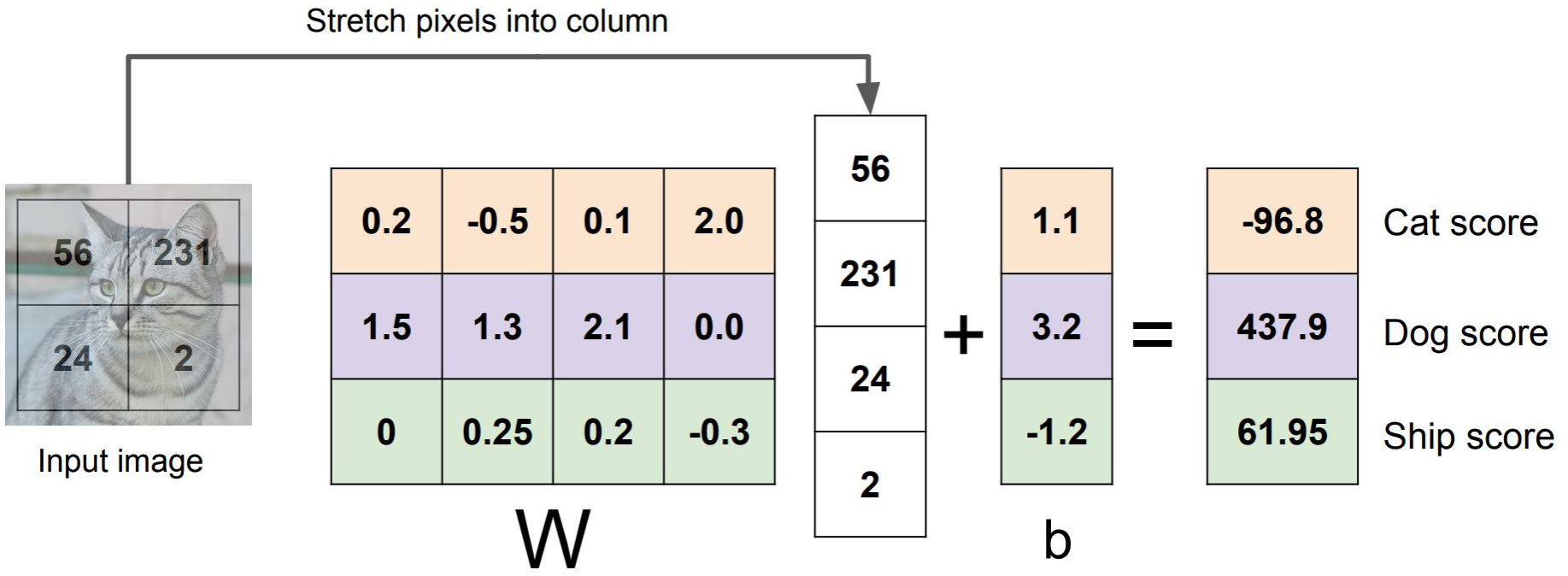
Regularizer is Frobenius norm of matrix (sum of squares of entries)

Learning Problem

$$W^* = \arg\min_W \frac{1}{2N} \sum_{i=1}^{N} \|Wx_i - y\|_2^2 + \lambda \|W\|_{fro}^2$$

Supervised learning

Example with an image with 4 pixels, and 3 classes (cat/dog/ship)



Supervised Learning: Linear regression

Sometimes called “Ridge Regression” with regularizer

$$x_i \in \mathbb{R}^{D_{in}} \qquad y_i \in \mathbb{R}^{D_{out}}$$

Input and output are vectors

$$\ell(\hat{y}, y) = \frac{1}{2}\|\hat{y} - y\|_2^2$$

Loss is Euclidean distance

Linear Regression

$$f(x, W) = Wx$$

$$W \in \mathbb{R}^{D_{out} \times D_{in}}$$

Model is just a matrix multiply

$$R(W) = \lambda \|W\|_{fro}^2$$

Regularizer is Frobenius norm of matrix (sum of squares of entries)

Learning Problem

$$W^* = \arg\min_W \frac{1}{2N} \sum_{i=1}^{N} \|Wx_i - y\|_2^2 + \lambda \|W\|_{fro}^2$$

Supervised Learning: Neural Network

Sometimes called “Fully-Connected Network” or “Multilayer Perceptron”

$$x_i \in \mathbb{R}^{D_{in}} \qquad y_i \in \mathbb{R}^{D_{out}}$$

Input and output are vectors

$$\ell(\hat{y}, y) = \frac{1}{2}\|\hat{y} - y\|_2^2$$

Loss is Euclidean distance

Linear Regression

$$f(x, W) = Wx$$

$$W \in \mathbb{R}^{D_{out} \times D_{in}}$$

Model is just a matrix multiply

New Model

$$f(x, W_1, W_2) = W_2 W_1 x$$

$$W_1 \in \mathbb{R}^{H \times D_{in}}$$

$$W_2 \in \mathbb{R}^{D_{out} \times H}$$

Model is **two** matrix multiplies

Supervised Learning: Neural Network

Sometimes called “Fully-Connected Network” or “Multilayer Perceptron”

$$x_i \in \mathbb{R}^{D_{in}} \qquad y_i \in \mathbb{R}^{D_{out}}$$

Input and output are vectors

$$\ell(\hat{y}, y) = \frac{1}{2}\|\hat{y} - y\|_2^2$$

Loss is Euclidean distance

Linear Regression

$$f(x, W) = Wx$$

$$W \in \mathbb{R}^{D_{out} \times D_{in}}$$

Model is just a matrix multiply

New Model

$$f(x, W_1, W_2) = W_2 W_1 x$$

$$W_1 \in \mathbb{R}^{H \times D_{in}}$$

$$W_2 \in \mathbb{R}^{D_{out} \times H}$$

Model is **two** matrix multiplies

Question: Is the new model “more powerful” than Linear Regression? Can it represent any functions that Linear Regression cannot?

Supervised Learning: Neural Network

Sometimes called “Fully-Connected Network” or “Multilayer Perceptron”

$$x_i \in \mathbb{R}^{D_{in}} \qquad y_i \in \mathbb{R}^{D_{out}}$$

Input and output are vectors

$$\ell(\hat{y}, y) = \frac{1}{2}\|\hat{y} - y\|_2^2$$

Loss is Euclidean distance

Linear Regression

$$f(x, W) = Wx$$

$$W \in \mathbb{R}^{D_{out} \times D_{in}}$$

Model is just a matrix multiply

New Model

$$f(x, W_1, W_2) = W_2 W_1 x$$

$$W_1 \in \mathbb{R}^{H \times D_{in}}$$

$$W_2 \in \mathbb{R}^{D_{out} \times H}$$

Model is **two** matrix multiplies

Question: Is the new model “more powerful” than Linear Regression? Can it represent any functions that Linear Regression cannot?

Answer: NO! We can write $W = W_2 W_1$ And recover Linear Regression

Supervised Learning: Neural Network

Sometimes called “Fully-Connected Network” or “Multilayer Perceptron”

$$x_i \in \mathbb{R}^{D_{in}} \qquad y_i \in \mathbb{R}^{D_{out}}$$

Input and output are vectors

$$\ell(\hat{y}, y) = \frac{1}{2}\|\hat{y} - y\|_2^2$$

Loss is Euclidean distance

Linear Regression

$$f(x, W) = Wx$$

$$W \in \mathbb{R}^{D_{out} \times D_{in}}$$

Model is just a matrix multiply

Neural Network

~~$$f(x, W_1, W_2) = W_2 W_1 x$$~~

$$f(x, W_1, W_2) = W_2 \sigma(W_1 x)$$

$$W_1 \in \mathbb{R}^{H \times D_{in}}$$

$$W_2 \in \mathbb{R}^{D_{out} \times H}$$

Model is **two** matrix multiplies, with an **elementwise nonlinearity**

$$\sigma : \mathbb{R}^H \to \mathbb{R}^H$$

Supervised Learning: Neural Network

Sometimes called “Fully-Connected Network” or “Multilayer Perceptron”

$$x_i \in \mathbb{R}^{D_{in}} \qquad y_i \in \mathbb{R}^{D_{out}}$$

Input and output are vectors

$$\ell(\hat{y}, y) = \frac{1}{2}\|\hat{y} - y\|_2^2$$

Loss is Euclidean distance

Linear Regression

$$f(x, W) = Wx$$

$$W \in \mathbb{R}^{D_{out} \times D_{in}}$$

Model is just a matrix multiply

Neural Network

~~$$f(x, W_1, W_2) = W_2 W_1 x$$~~

$$f(x, W_1, W_2) = W_2 \sigma(W_1 x)$$

$$W_1 \in \mathbb{R}^{H \times D_{in}}$$

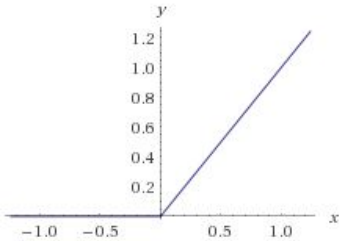
$$W_2 \in \mathbb{R}^{D_{out} \times H}$$

Model is **two** matrix multiplies, with an **elementwise nonlinearity**

$$\sigma : \mathbb{R}^H \to \mathbb{R}^H$$

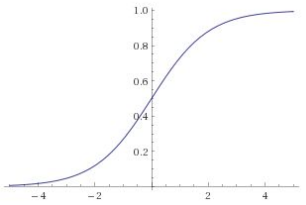
Common nonlinearities:

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



Rectified Linear (ReLU)

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



Logistic Sigmoid

Supervised Learning: Neural Network

Sometimes called “Fully-Connected Network” or “Multilayer Perceptron”

$$x_i \in \mathbb{R}^{D_{in}} \qquad y_i \in \mathbb{R}^{D_{out}}$$

Input and output are vectors

$$\ell(\hat{y}, y) = \frac{1}{2}\|\hat{y} - y\|_2^2$$

Loss is Euclidean distance

Linear Regression

$$f(x, W) = Wx$$

$$W \in \mathbb{R}^{D_{out} \times D_{in}}$$

Model is just a matrix multiply

Neural Network

~~$$f(x, W_1, W_2) = W_2 W_1 x$$~~

$$f(x, W_1, W_2) = W_2 \sigma(W_1 x)$$

$$W_1 \in \mathbb{R}^{H \times D_{in}}$$

$$W_2 \in \mathbb{R}^{D_{out} \times H}$$

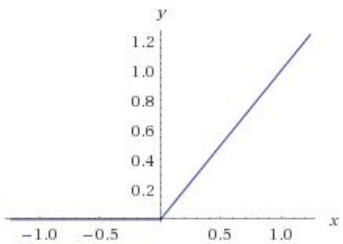
Model is **two** matrix multiplies, with an **elementwise nonlinearity**

$$\sigma : \mathbb{R}^H \to \mathbb{R}^H$$

Neural Network is more powerful than Linear Regression!

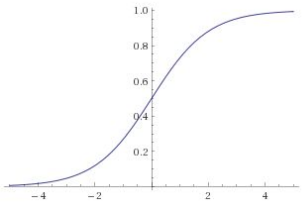
Common nonlinearities:

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



Rectified Linear (ReLU)

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



Logistic Sigmoid

Neural Networks with many layers

Sometimes called “Fully-Connected Network” or “Multilayer Perceptron”

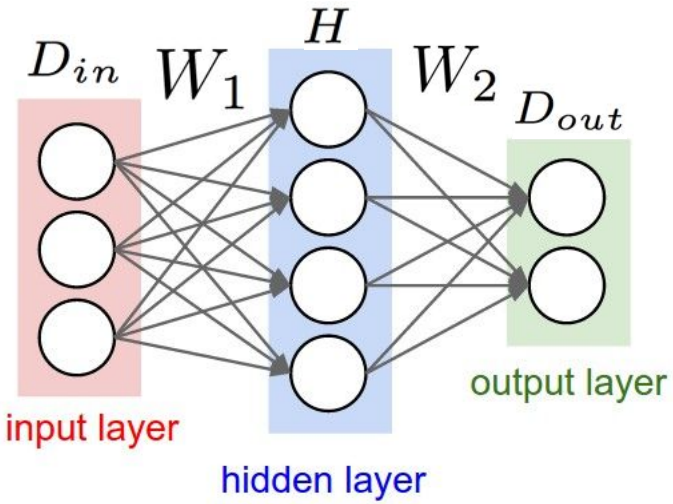
$$x_i \in \mathbb{R}^{D_{in}} \qquad y_i \in \mathbb{R}^{D_{out}}$$

Two Layer network

$$f(x, W_1, W_2) = W_2\sigma(W_1 x)$$

$$W_1 \in \mathbb{R}^{H \times D_{in}}$$

$$W_2 \in \mathbb{R}^{D_{out} \times H}$$



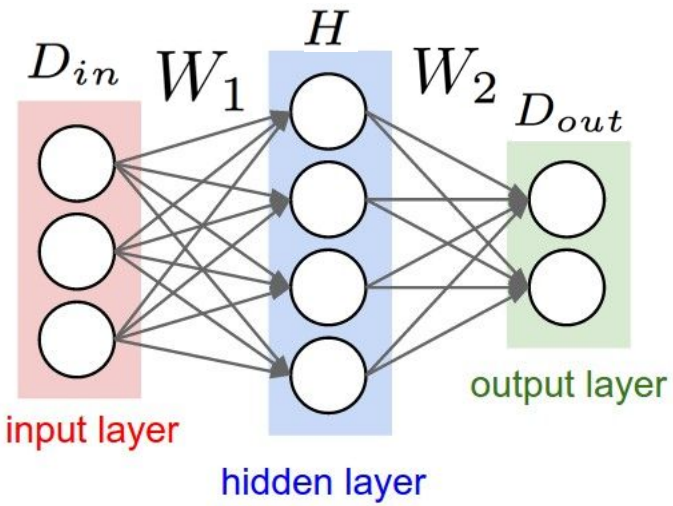
Neural Networks with many layers

Sometimes called “Fully-Connected Network” or “Multilayer Perceptron”

$$x_i \in \mathbb{R}^{D_{in}} \qquad y_i \in \mathbb{R}^{D_{out}}$$

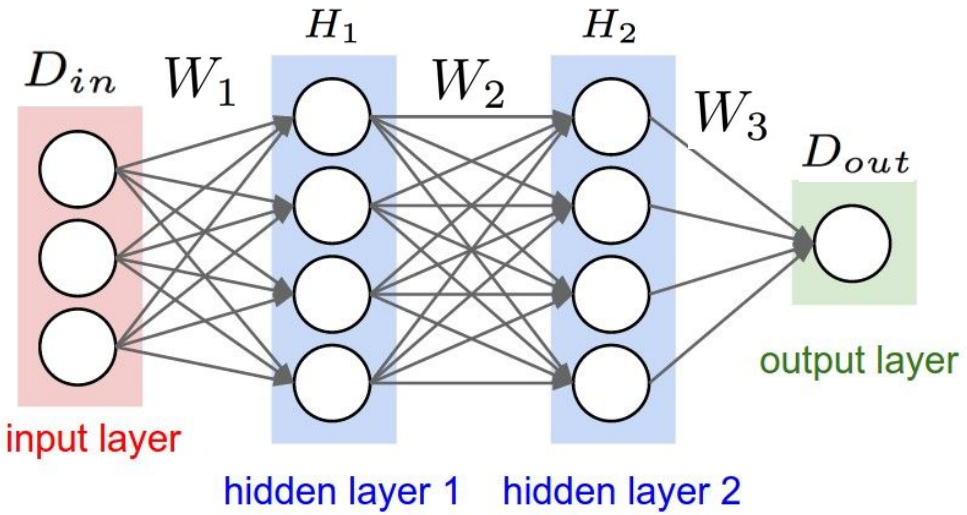
Two Layer network

$$f(x, W_1, W_2) = W_2\sigma(W_1 x)$$
$$W_1 \in \mathbb{R}^{H \times D_{in}}$$
$$W_2 \in \mathbb{R}^{D_{out} \times H}$$



Three Layer network

$$f(x, W_1, W_2, W_3) = W_3\sigma(W_2(\sigma(W_1 x)))$$
$$W_1 \in \mathbb{R}^{H_1 \times D_{in}}$$
$$W_2 \in \mathbb{R}^{H_2 \times H_1}$$
$$W_3 \in \mathbb{R}^{D_{out} \times H_2}$$



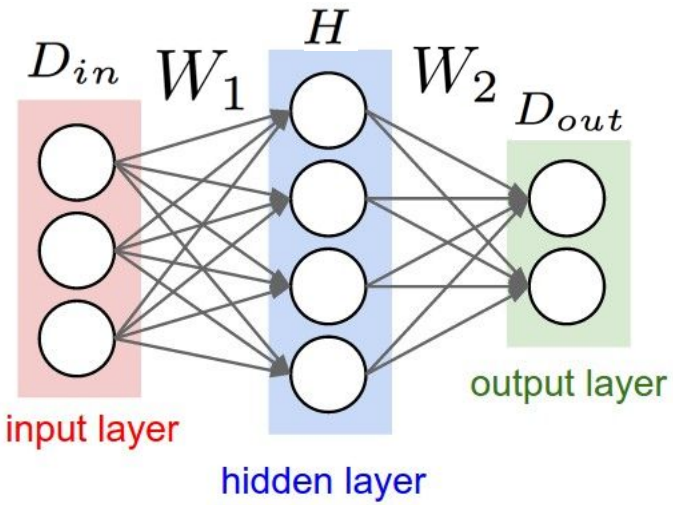
Neural Networks with many layers

Sometimes called “Fully-Connected Network” or “Multilayer Perceptron”

$$x_i \in \mathbb{R}^{D_{in}} \qquad y_i \in \mathbb{R}^{D_{out}}$$

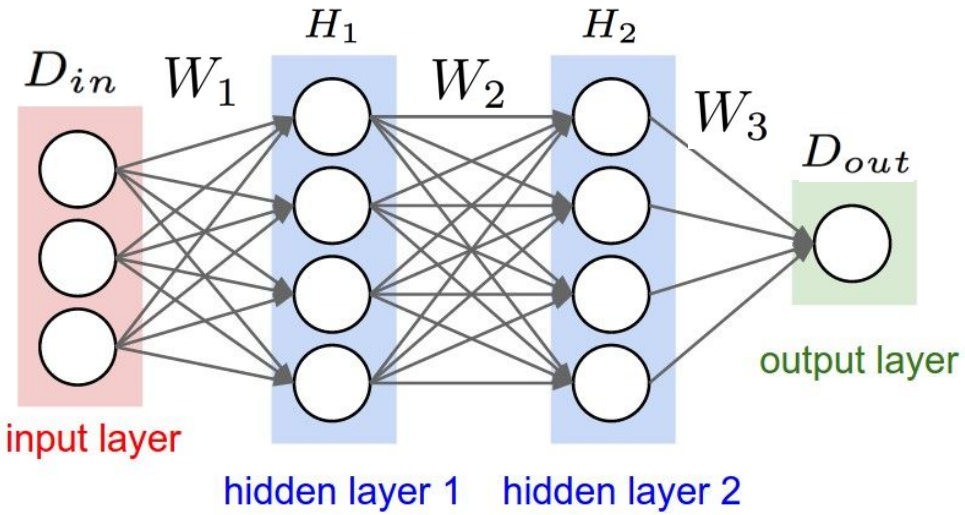
Two Layer network

$$f(x, W_1, W_2) = W_2\sigma(W_1 x)$$
$$W_1 \in \mathbb{R}^{H \times D_{in}}$$
$$W_2 \in \mathbb{R}^{D_{out} \times H}$$



Three Layer network

$$f(x, W_1, W_2, W_3) = W_3\sigma(W_2(\sigma(W_1 x)))$$
$$W_1 \in \mathbb{R}^{H_1 \times D_{in}}$$
$$W_2 \in \mathbb{R}^{H_2 \times H_1}$$
$$W_3 \in \mathbb{R}^{D_{out} \times H_2}$$



Hidden layers are **learned feature representations** of the input!

Outline

- Applications
- Motivation
- Supervised learning
- Gradient descent
- Backpropagation
- Convolutional Neural Networks

How to find best weights w^*?

$$w^* = \arg\min_w \frac{1}{N} \sum_{i=1}^{N} \ell(f(x_i, w), y_i) + R(w)$$
$$= \arg\min_w g(w)$$

How to find best weights w^*?

$$w^* = \arg\min_w \frac{1}{N} \sum_{i=1}^{N} \ell(f(x_i, w), y_i) + R(w)$$
$$= \arg\min_w g(w)$$

Slide credit: CS 231n, Lecture 3

How to find best weights w^*?

$$w^* = \arg\min_w \frac{1}{N} \sum_{i=1}^{N} \ell(f(x_i, w), y_i) + R(w)$$
$$= \arg\min_w g(w)$$

Idea #1: Closed Form Solution

Works for some models (linear regression) but in general very hard (even for one-layer net)

Slide credit: CS 231n, Lecture 3

How to find best weights w^*?

$$w^* = \arg\min_w \frac{1}{N} \sum_{i=1}^{N} \ell(f(x_i, w), y_i) + R(w)$$
$$= \arg\min_w g(w)$$

Idea #1: Closed Form Solution
Works for some models (linear regression) but in general very hard (even for one-layer net)

Idea #2: Random Search
Try a bunch of random values for w, keep the best one
Extremely inefficient in high dimensions

Slide credit: CS 231n, Lecture 3

How to find best weights w^*?

$$w^* = \arg\min_w \frac{1}{N} \sum_{i=1}^{N} \ell(f(x_i, w), y_i) + R(w)$$
$$= \arg\min_w g(w)$$

Idea #1: Closed Form Solution
Works for some models (linear regression) but in general very hard (even for one-layer net)

Idea #2: Random Search
Try a bunch of random values for w, keep the best one
Extremely inefficient in high dimensions

Idea #3: Go downhill
Start somewhere random, take tiny steps downhill and repeat
Good idea!

Slide credit: CS 231n, Lecture 3

Which way is downhill?

Recall from single-variable calculus:

$$g'(w) = \lim_{h \to 0} \frac{g(w+h) - g(w)}{h}$$

If w is a scalar the **derivative** tells us how much g will change if w changes a little bit

Which way is downhill?

Recall from single-variable calculus:

$$g'(w) = \lim_{h \to 0} \frac{g(w+h) - g(w)}{h}$$

If w is a scalar the **derivative** tells us how much g will change if w changes a little bit

And multivariable calculus:

$$\frac{\partial g}{\partial w_i}(w) = \lim_{h \to 0} \frac{g(w + h\hat{e}_i) - g(w)}{h}$$

If w is a vector then we can take **partial derivatives** with respect to each component

$$\nabla g(w) = \left(\frac{\partial g}{\partial w_1}(w), \ldots, \frac{\partial g}{\partial w_K}(w) \right)$$

The **gradient** is the vector of all partial derivatives; it has the same shape as w

Which way is downhill?

Recall from single-variable calculus:

$$g'(w) = \lim_{h \to 0} \frac{g(w+h) - g(w)}{h}$$

If w is a scalar the **derivative** tells us how much g will change if w changes a little bit

And multivariable calculus:

$$\frac{\partial g}{\partial w_i}(w) = \lim_{h \to 0} \frac{g(w + h\hat{e}_i) - g(w)}{h}$$

If w is a vector then we can take **partial derivatives** with respect to each component

$$\nabla g(w) = \left(\frac{\partial g}{\partial w_1}(w), \ldots, \frac{\partial g}{\partial w_K}(w) \right)$$

The **gradient** is the vector of all partial derivatives; it has the same shape as w

Useful fact: The gradient of g at w gives the **direction of steepest increase**

Gradient descent

The simple algorithm behind all deep learning

Initialize w randomly

While true:

Compute gradient $\nabla g(w)$ at current point

Move downhill a little bit: $w = w - \alpha \nabla g(w)$

Gradient descent

The simple algorithm behind all deep learning

Initialize w randomly
While true:
    Compute gradient $\nabla g(w)$ at current point
    Move downhill a little bit: $w = w - \alpha \nabla g(w)$

Learning rate: How big each step should be

Gradient descent

The simple algorithm behind all deep learning

In practice we **estimate** the gradient using a small **minibatch** of data (Stochastic gradient descent)

Initialize w randomly
While true:
  Compute gradient $\nabla g(w)$ at current point
  Move downhill a little bit: $w = w - \alpha \nabla g(w)$

Learning rate: How big each step should be

Gradient descent

The simple algorithm behind all deep learning

In practice we **estimate** the gradient using a small **minibatch** of data (Stochastic gradient descent)

Initialize w randomly
While true:
  Compute gradient $\nabla g(w)$ at current point
  Move downhill a little bit: $w = w - \alpha \nabla g(w)$

The **update rule** is the formula for updating the weights at each iteration; in practice people use fancier rules (momentum, RMSProp, Adam, etc)

Learning rate: How big each step should be

Gradient descent

The simple algorithm behind all deep learning

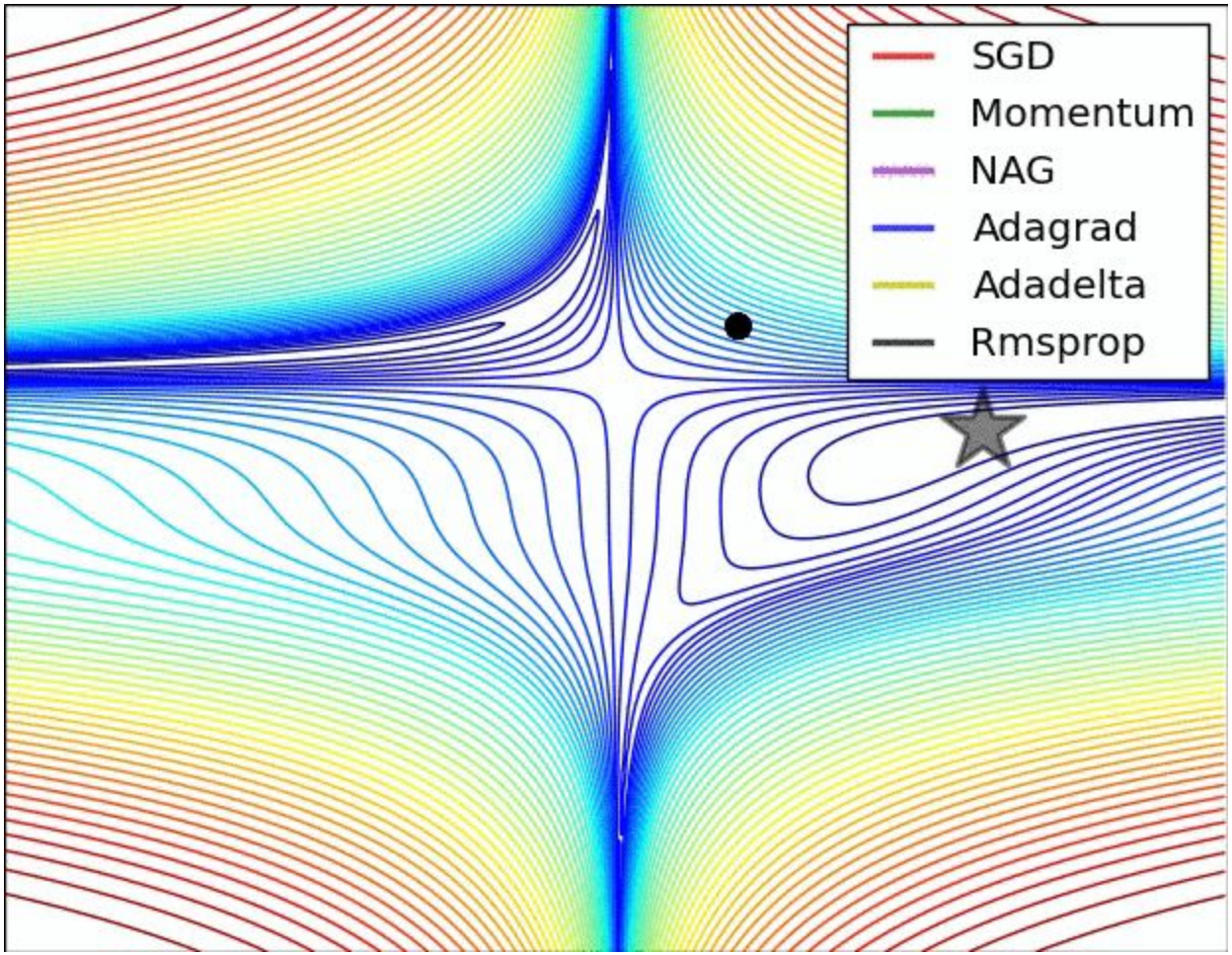


Image credit: Alec Radford

Outline

- Applications
- Motivation
- Supervised learning
- Gradient descent
- **Backpropagation**
- Convolutional Neural Networks

How to compute gradients?

Work it out on paper?

Linear Regression

$$g(W) = \frac{1}{2N} \sum_{i=1}^{N} \|Wx_i - y_i\|_2^2 + \lambda \|W\|_{fro}$$

$$\nabla g(W) = \ ?$$

Doable for simple models

How to compute gradients?

Work it out on paper?

Three layer network

$$g(W_1, W_2, W_3) = \frac{1}{2N} \sum_{i=1}^{N} \|W_3\sigma(W_2\sigma(W_1 x_i)) - y_i\|_2^2$$

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

$$\nabla_{W_1} g(W_1, W_2, W_3) = ?$$

$$\nabla_{W_2} g(W_1, W_2, W_3) = ?$$

$$\nabla_{W_3} g(W_1, W_2, W_3) = ?$$

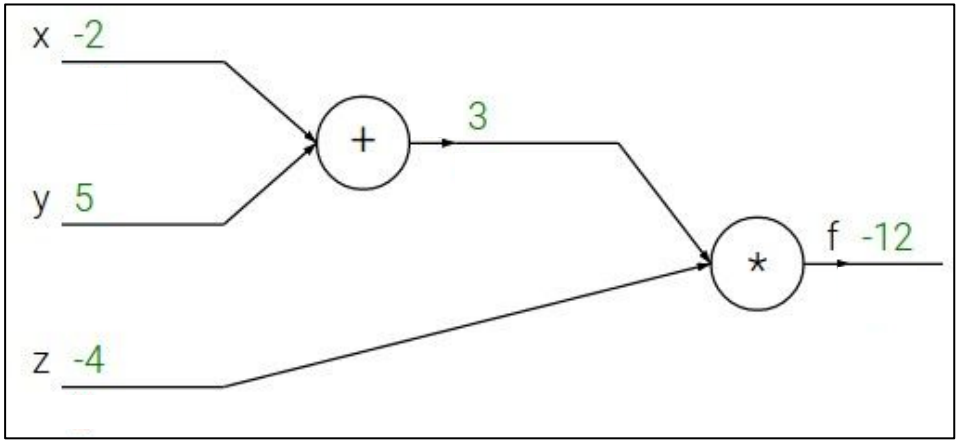
Gets hairy for more complex models

Backpropagation

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

e.g. x = -2, y = 5, z = -4

Computational graph



Slide credit: CS 231n, Lecture 4

Backpropagation

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

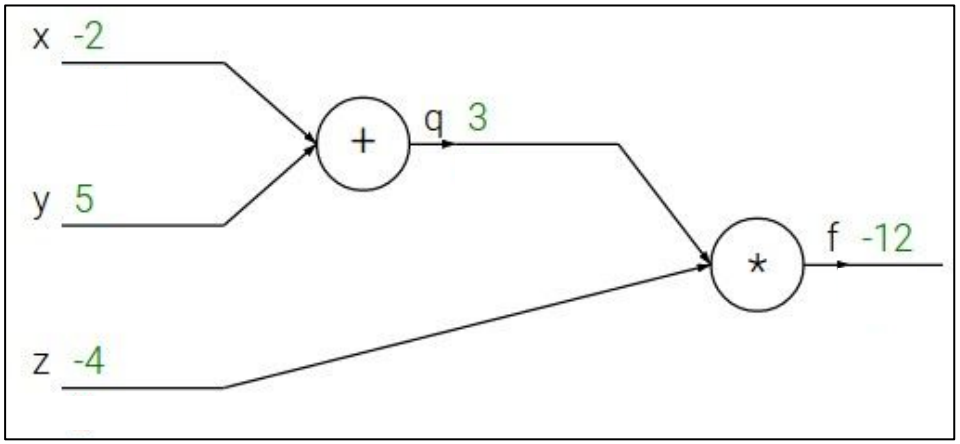
e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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

Computational graph



Slide credit: CS 231n, Lecture 4

Backpropagation

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

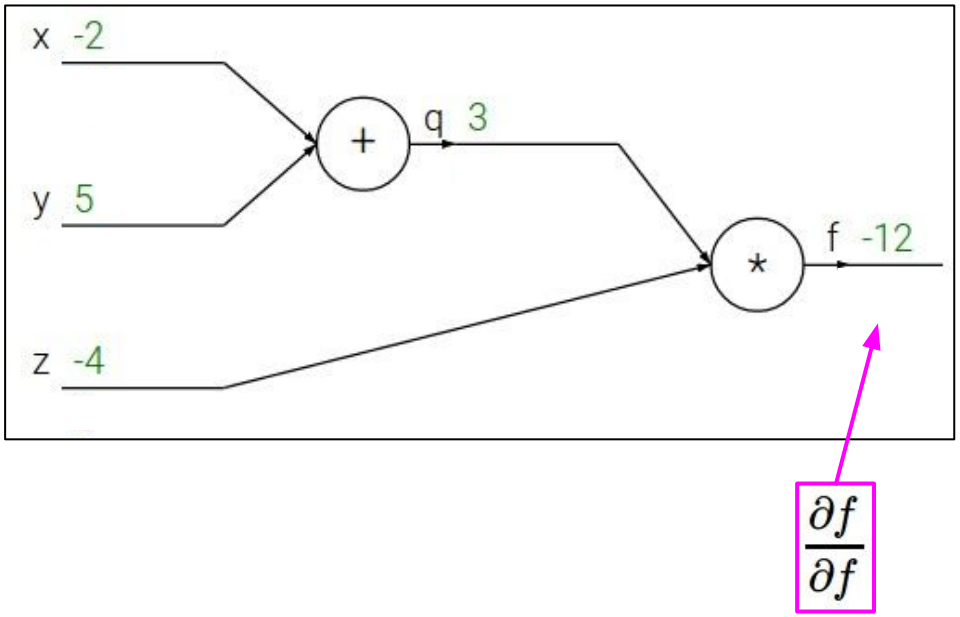
e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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

Computational graph



Slide credit: CS 231n, Lecture 4

Backpropagation

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

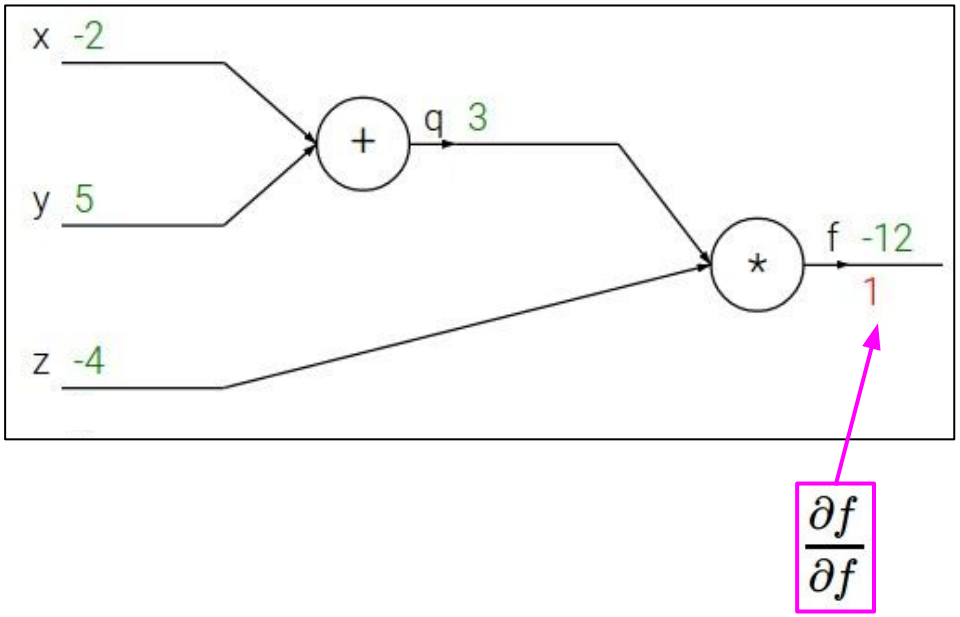
e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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

Computational graph



Slide credit: CS 231n, Lecture 4

Backpropagation

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

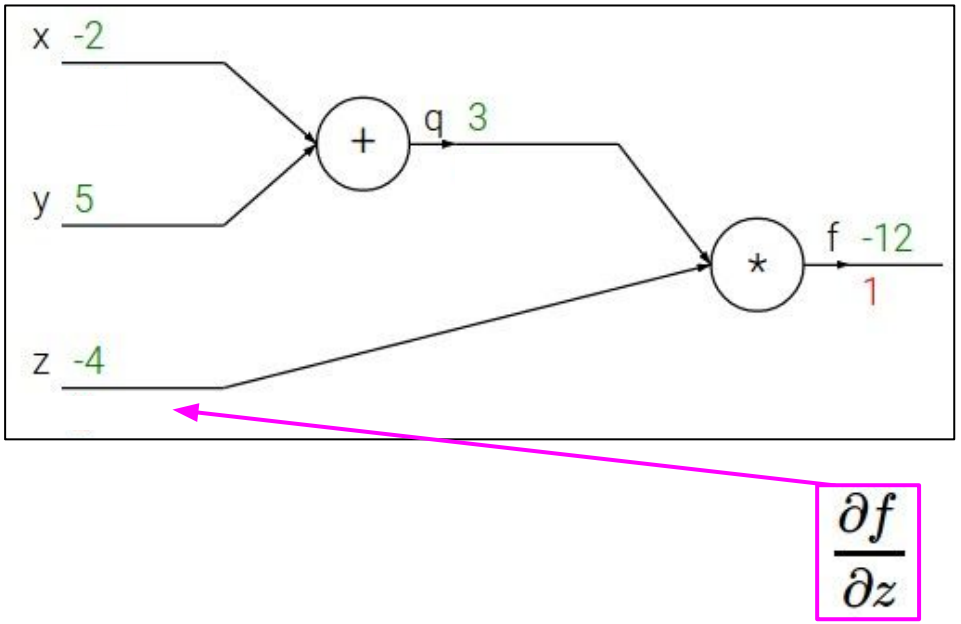
e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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

Computational graph



Slide credit: CS 231n, Lecture 4

Backpropagation

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

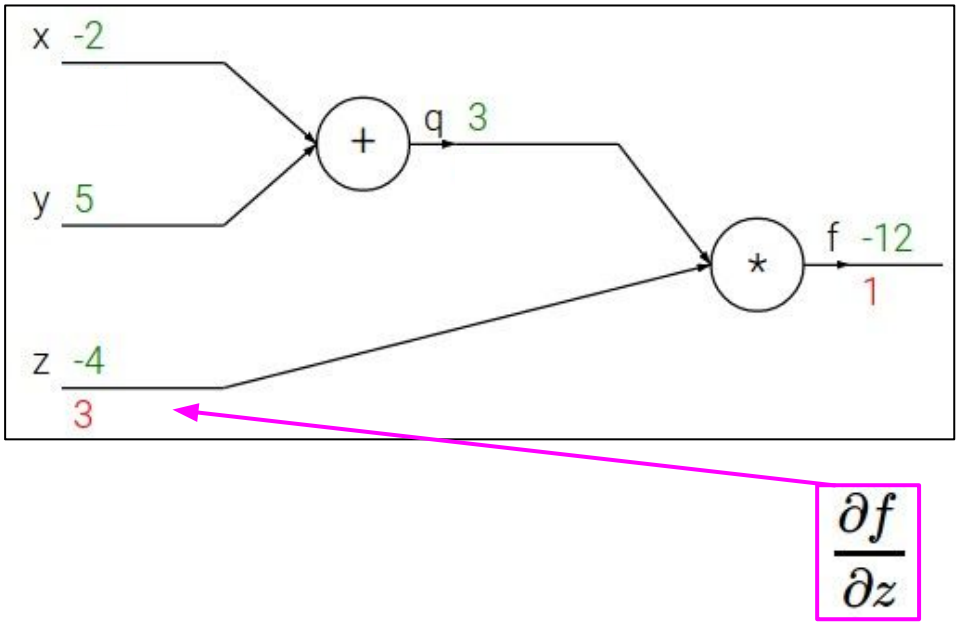
e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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

Computational graph



Slide credit: CS 231n, Lecture 4

Backpropagation

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

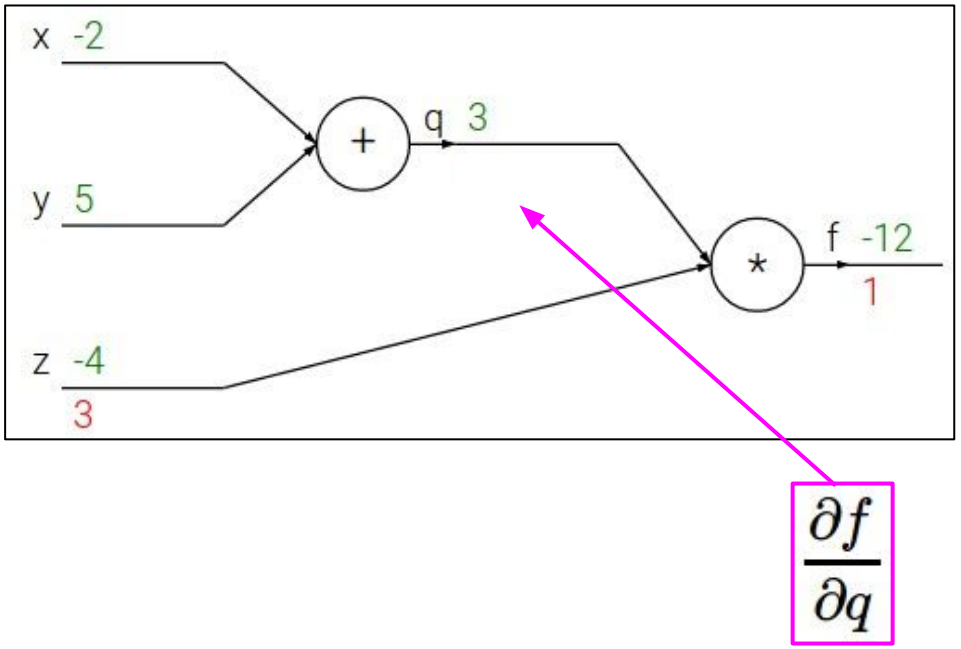
e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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

Computational graph



Slide credit: CS 231n, Lecture 4

Backpropagation

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

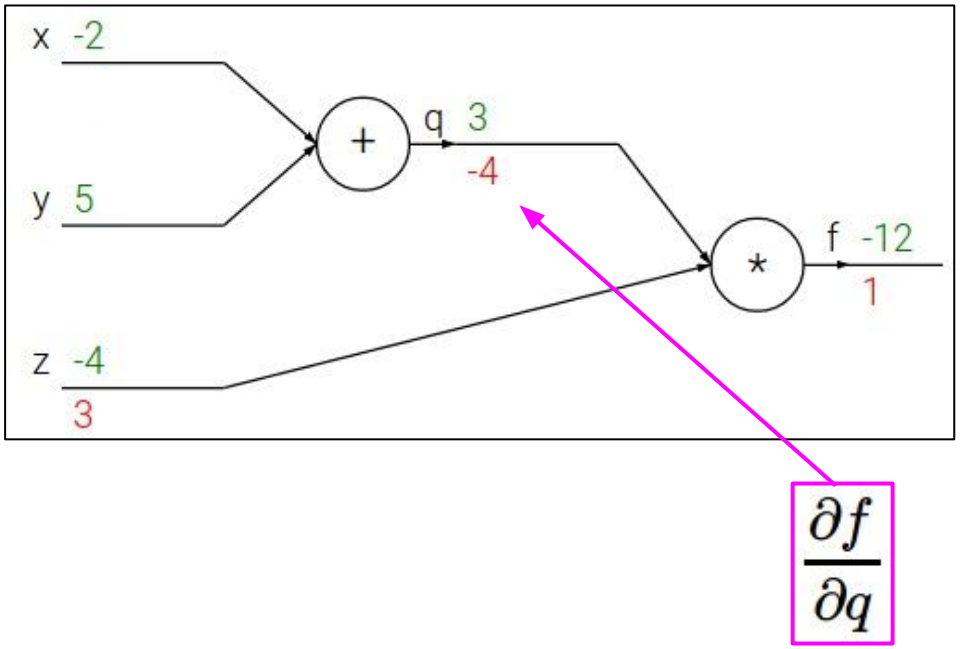
e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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

Computational graph



Slide credit: CS 231n, Lecture 4

Backpropagation

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

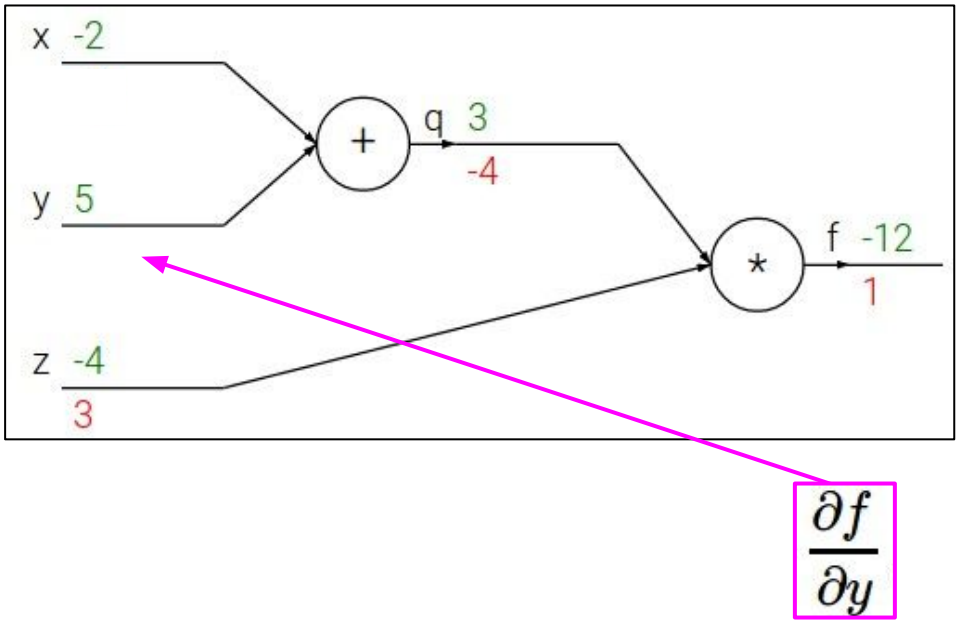
e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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

Computational graph



Slide credit: CS 231n, Lecture 4

Backpropagation

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

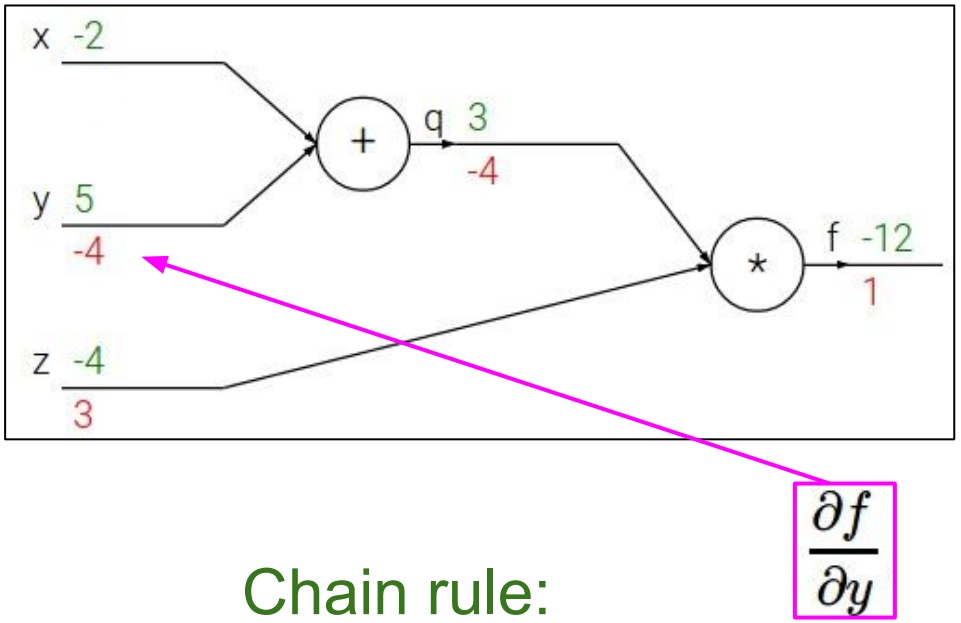
e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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

Computational graph



Chain rule:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}$$

Slide credit: CS 231n, Lecture 4

Backpropagation

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

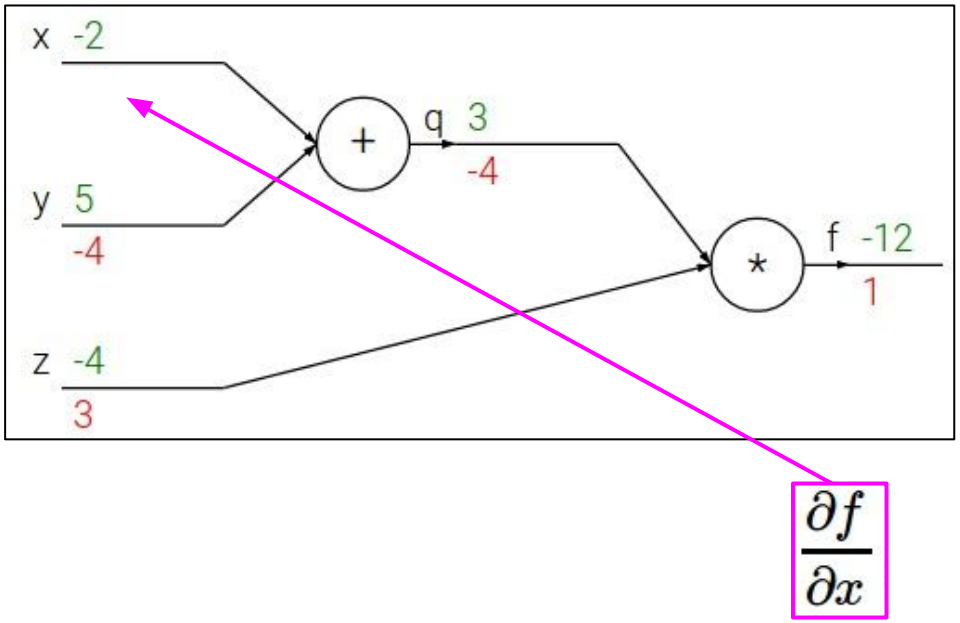
e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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

Computational graph



Slide credit: CS 231n, Lecture 4

Backpropagation

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

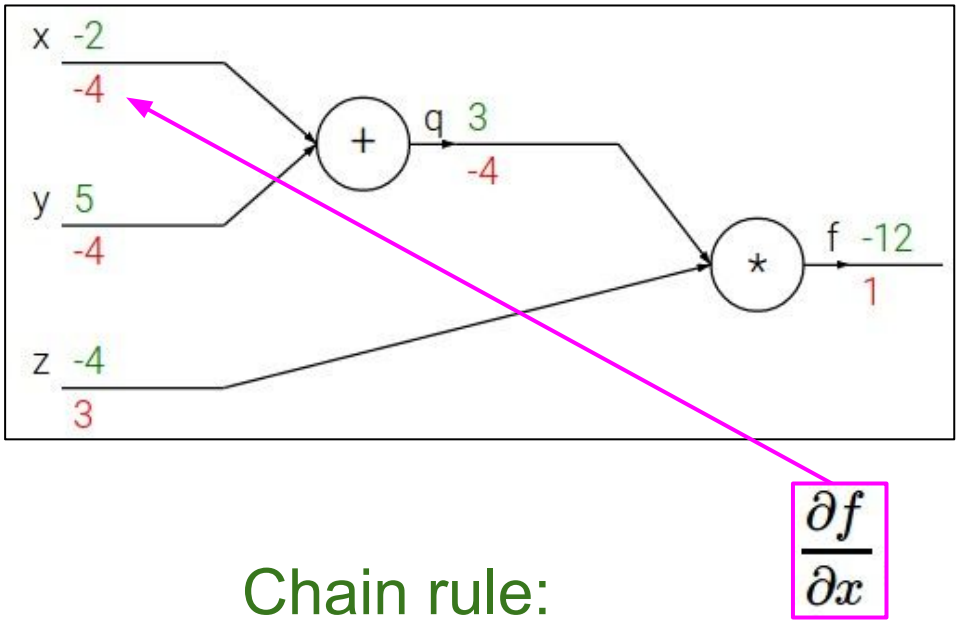
e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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

Computational graph



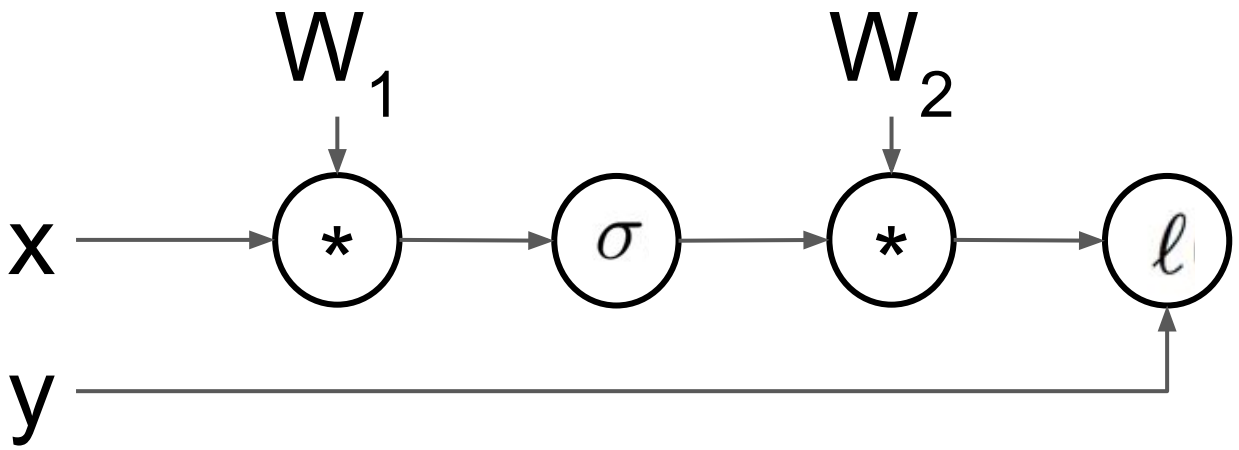
Chain rule:

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

Slide credit: CS 231n, Lecture 4

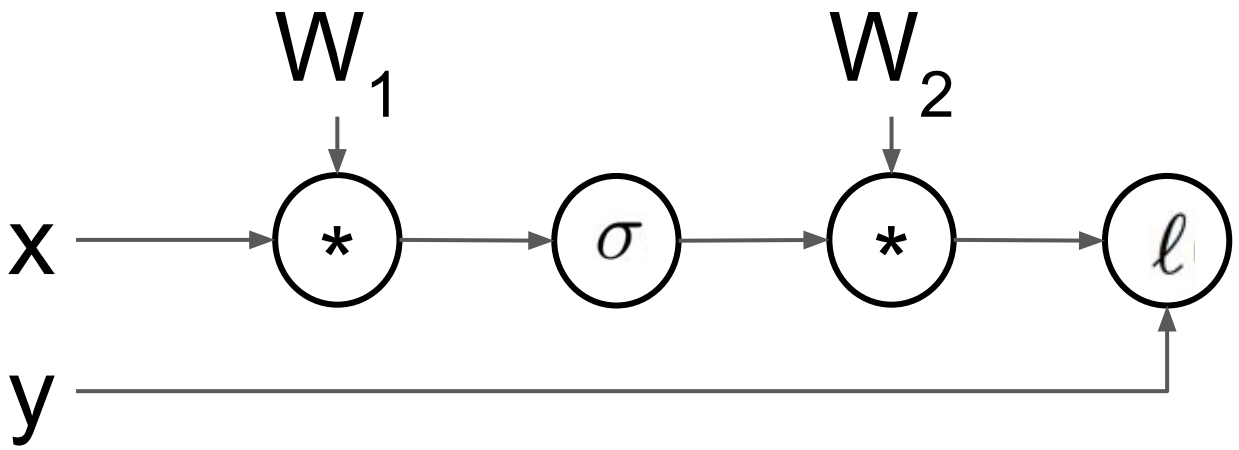
Backpropagation

Computational graph nodes can be vectors or matrices or n-dimensional tensors



Backpropagation

Computational graph nodes can be vectors or matrices or n-dimensional tensors

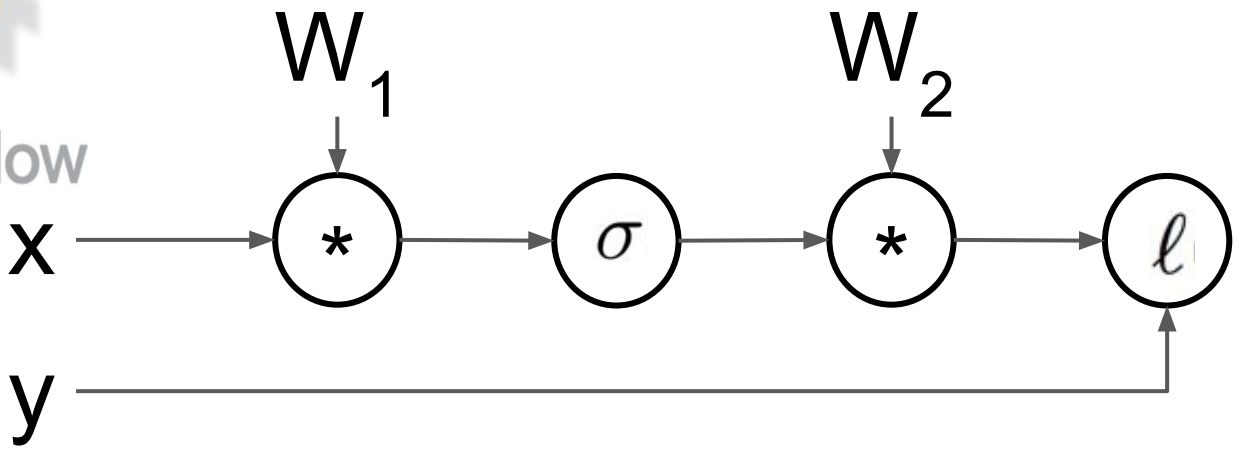


Forward pass: Run graph "forward" to compute loss
Backward pass: Run graph "backward" to compute gradients with respect to loss

Easily compute gradients for big, complex models!

Backpropagation

Computational graph nodes can be vectors or matrices or n-dimensional tensors



Forward pass: Run graph "forward" to compute loss
Backward pass: Run graph "backward" to compute gradients with respect to loss

Easily compute gradients for big, complex models!

Tensors **flow** along edges in the graph

A two-layer neural network in 25 lines of code

```
import numpy as np

D, H, N = 8, 64, 32
learning_rate = 0.0001

W1 = np.random.randn(D, H)
W2 = np.random.randn(H, D)
for t in xrange(10000):
  x = np.random.randn(N, D)
  y = np.sin(x)

  s = x.dot(W1)
  a = np.maximum(s, 0)
  y_hat = a.dot(W2)

  loss = 0.5 * np.sum((y_hat - y) ** 2.0)

  dy_hat = y_hat - y
  dW2 = a.T.dot(dy_hat)
  da = dy_hat.dot(W2.T)
  ds = (s > 0) * da
  dW1 = x.T.dot(ds)

  W1 -= learning_rate * dW1
  W2 -= learning_rate * dW2
```

A two-layer neural network in 25 lines of code

```python
import numpy as np

D, H, N = 8, 64, 32
learning_rate = 0.0001

W1 = np.random.randn(D, H)
W2 = np.random.randn(H, D)
for t in xrange(10000):
  x = np.random.randn(N, D)
  y = np.sin(x)

  s = x.dot(W1)
  a = np.maximum(s, 0)
  y_hat = a.dot(W2)

  loss = 0.5 * np.sum((y_hat - y) ** 2.0)

  dy_hat = y_hat - y
  dW2 = a.T.dot(dy_hat)
  da = dy_hat.dot(W2.T)
  ds = (s > 0) * da
  dW1 = x.T.dot(ds)

  W1 -= learning_rate * dW1
  W2 -= learning_rate * dW2
```

Randomly initialize weights

A two-layer neural network in 25 lines of code

```
import numpy as np

D, H, N = 8, 64, 32
learning_rate = 0.0001

W1 = np.random.randn(D, H)
W2 = np.random.randn(H, D)
for t in xrange(10000):
  x = np.random.randn(N, D)
  y = np.sin(x)

  s = x.dot(W1)
  a = np.maximum(s, 0)
  y_hat = a.dot(W2)

  loss = 0.5 * np.sum((y_hat - y) ** 2.0)

  dy_hat = y_hat - y
  dW2 = a.T.dot(dy_hat)
  da = dy_hat.dot(W2.T)
  ds = (s > 0) * da
  dW1 = x.T.dot(ds)

  W1 -= learning_rate * dW1
  W2 -= learning_rate * dW2
```

Get a batch of (random) data

A two-layer neural network in 25 lines of code

```
import numpy as np

D, H, N = 8, 64, 32
learning_rate = 0.0001

W1 = np.random.randn(D, H)
W2 = np.random.randn(H, D)
for t in xrange(10000):
  x = np.random.randn(N, D)
  y = np.sin(x)

  s = x.dot(W1)
  a = np.maximum(s, 0)
  y_hat = a.dot(W2)

  loss = 0.5 * np.sum((y_hat - y) ** 2.0)

  dy_hat = y_hat - y
  dW2 = a.T.dot(dy_hat)
  da = dy_hat.dot(W2.T)
  ds = (s > 0) * da
  dW1 = x.T.dot(ds)

  W1 -= learning_rate * dW1
  W2 -= learning_rate * dW2
```

Forward pass: compute loss

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

ReLU nonlinearity

$$\ell(\hat{y}, y) = \frac{1}{2}\|\hat{y} - y\|_2^2$$

Loss is Euclidean distance

A two-layer neural network in 25 lines of code

```
1  import numpy as np
2
3  D, H, N = 8, 64, 32
4  learning_rate = 0.0001
5
6  W1 = np.random.randn(D, H)
7  W2 = np.random.randn(H, D)
8  for t in xrange(10000):
9    x = np.random.randn(N, D)
10   y = np.sin(x)
11
12   s = x.dot(W1)
13   a = np.maximum(s, 0)
14   y_hat = a.dot(W2)
15
16   loss = 0.5 * np.sum((y_hat - y) ** 2.0)
17
18   dy_hat = y_hat - y
19   dW2 = a.T.dot(dy_hat)
20   da = dy_hat.dot(W2.T)
21   ds = (s > 0) * da
22   dW1 = x.T.dot(ds)
23
24   W1 -= learning_rate * dW1
25   W2 -= learning_rate * dW2
```

Backward pass: compute gradients

A two-layer neural network in 25 lines of code

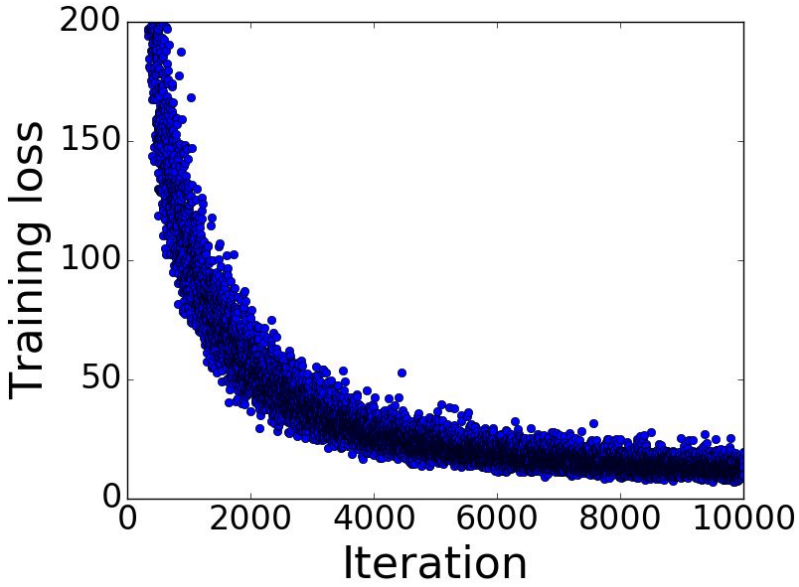
```
import numpy as np

D, H, N = 8, 64, 32
learning_rate = 0.0001

W1 = np.random.randn(D, H)
W2 = np.random.randn(H, D)
for t in xrange(10000):
  x = np.random.randn(N, D)
  y = np.sin(x)

  s = x.dot(W1)
  a = np.maximum(s, 0)
  y_hat = a.dot(W2)

  loss = 0.5 * np.sum((y_hat - y) ** 2.0)

  dy_hat = y_hat - y
  dW2 = a.T.dot(dy_hat)
  da = dy_hat.dot(W2.T)
  ds = (s > 0) * da
  dW1 = x.T.dot(ds)

  W1 -= learning_rate * dW1
  W2 -= learning_rate * dW2
```

Update weights

A two-layer neural network in 25 lines of code

```python
import numpy as np

D, H, N = 8, 64, 32
learning_rate = 0.0001

W1 = np.random.randn(D, H)
W2 = np.random.randn(H, D)
for t in xrange(10000):
  x = np.random.randn(N, D)
  y = np.sin(x)

  s = x.dot(W1)
  a = np.maximum(s, 0)
  y_hat = a.dot(W2)

  loss = 0.5 * np.sum((y_hat - y) ** 2.0)

  dy_hat = y_hat - y
  dW2 = a.T.dot(dy_hat)
  da = dy_hat.dot(W2.T)
  ds = (s > 0) * da
  dW1 = x.T.dot(ds)

  W1 -= learning_rate * dW1
  W2 -= learning_rate * dW2
```

When you run this code:
loss goes down!

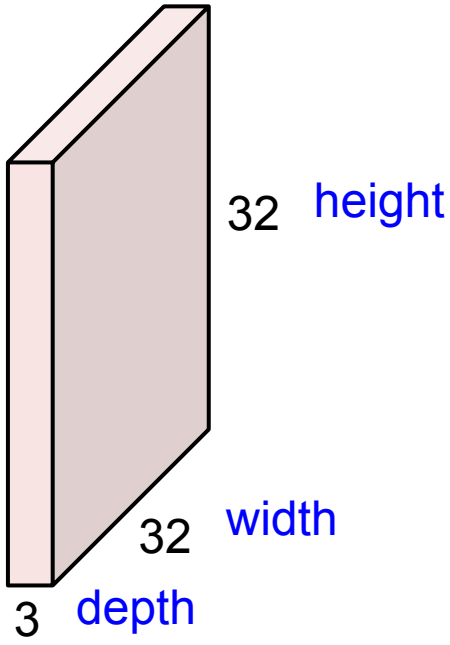


Outline

- Applications
- Motivation
- Supervised learning
- Gradient descent
- Backpropagation
- Convolutional Neural Networks

Convolution Layer

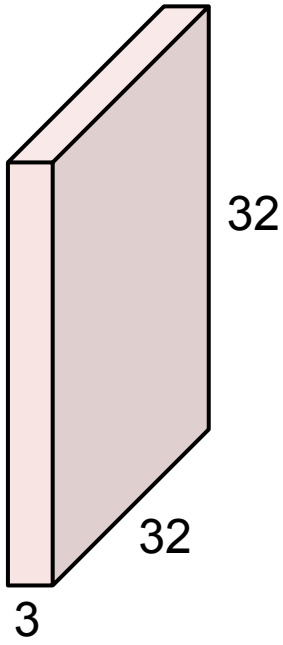
32x32x3 image



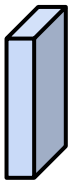
Slide credit: CS 231n, Lecture 7

Convolution Layer

32x32x3 image



5x5x3 filter



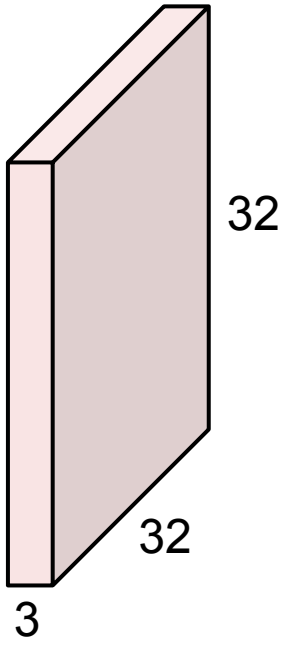
Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Slide credit: CS 231n, Lecture 7

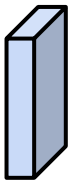
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image



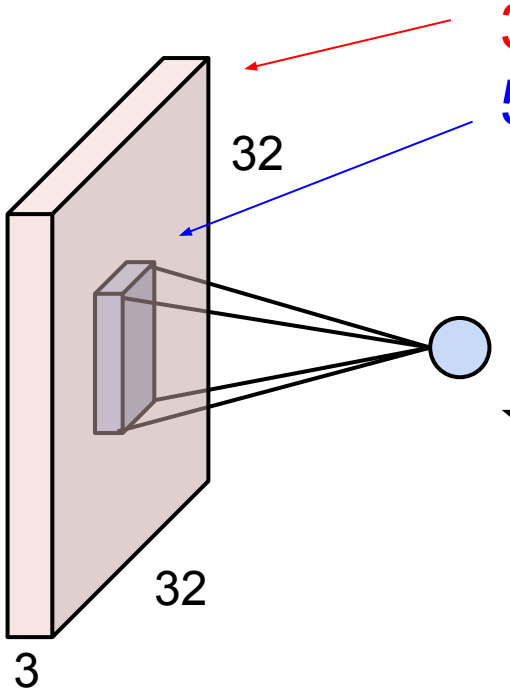
5x5x3 filter



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

Slide credit: CS 231n, Lecture 7

Convolution Layer

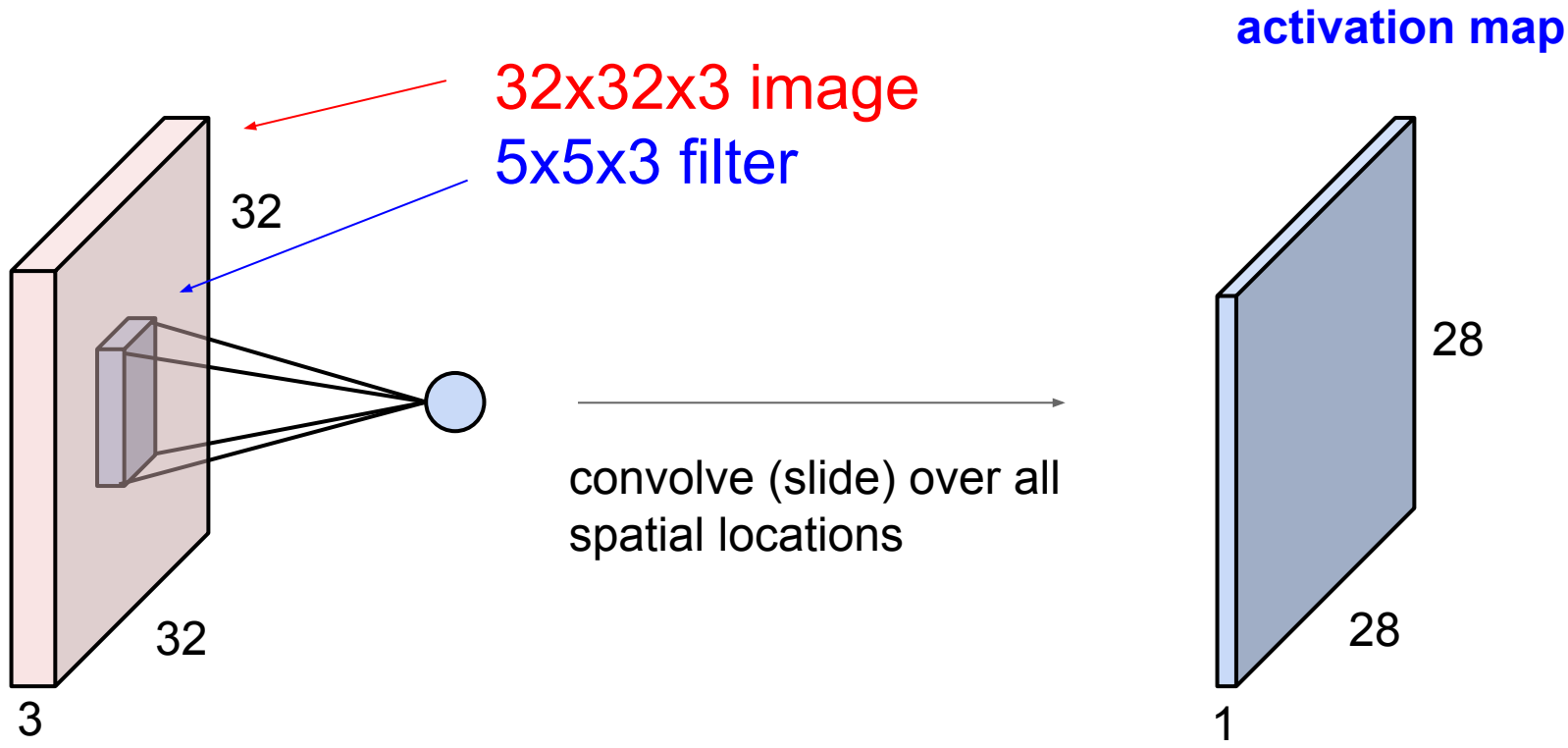


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Slide credit: CS 231n, Lecture 7

Convolution Layer



Translation Invariance!

Slide credit: CS 231n, Lecture 7

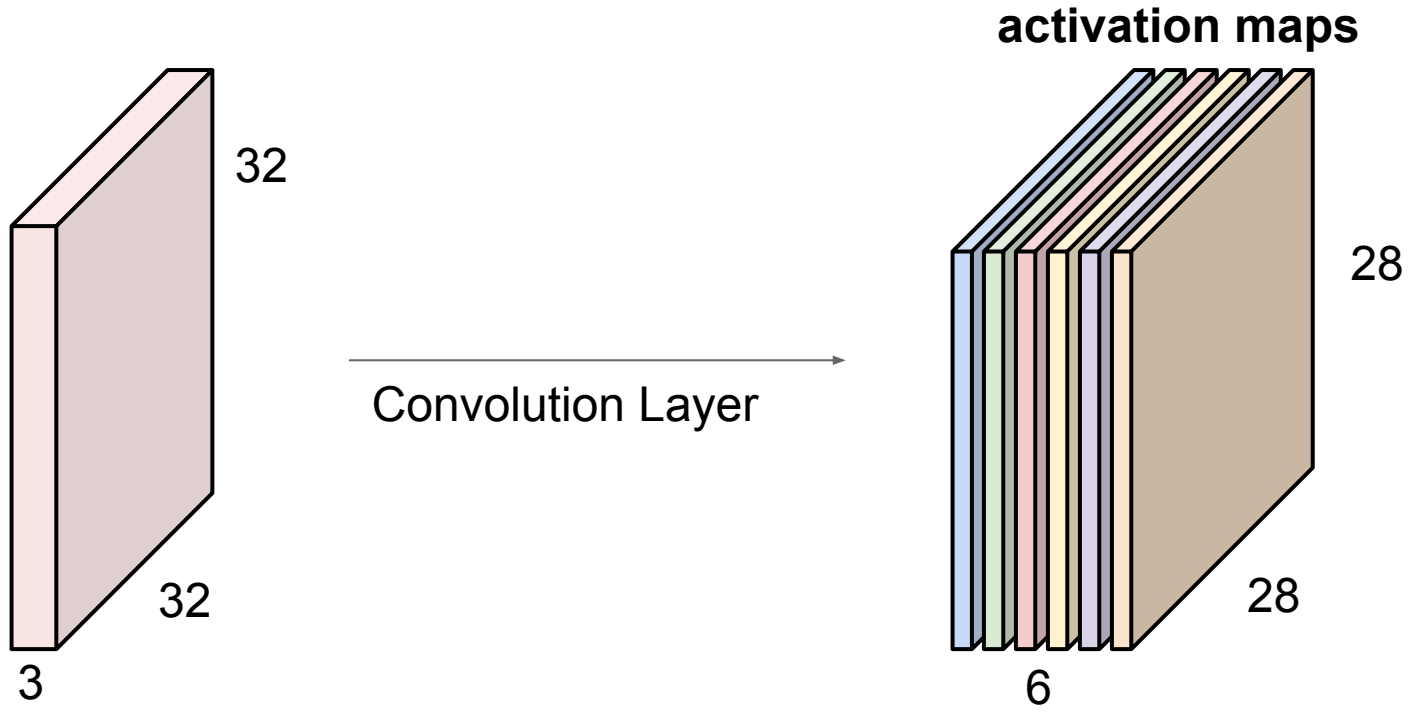
Convolution Layer

consider a second, green filter

Slide credit: CS 231n, Lecture 7

Convolution Layer

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

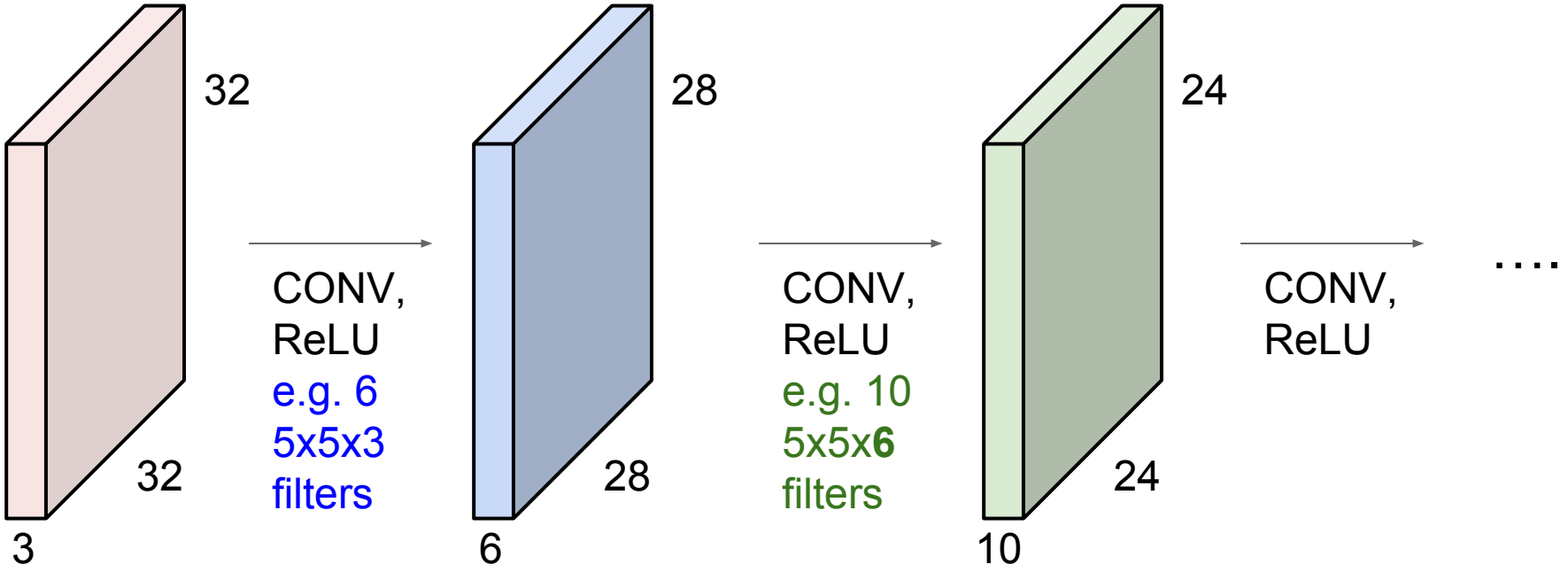


We stack these up to get a "new image" of size 28x28x6!

Slide credit: CS 231n, Lecture 7

Convolutional Neural Networks (CNN)

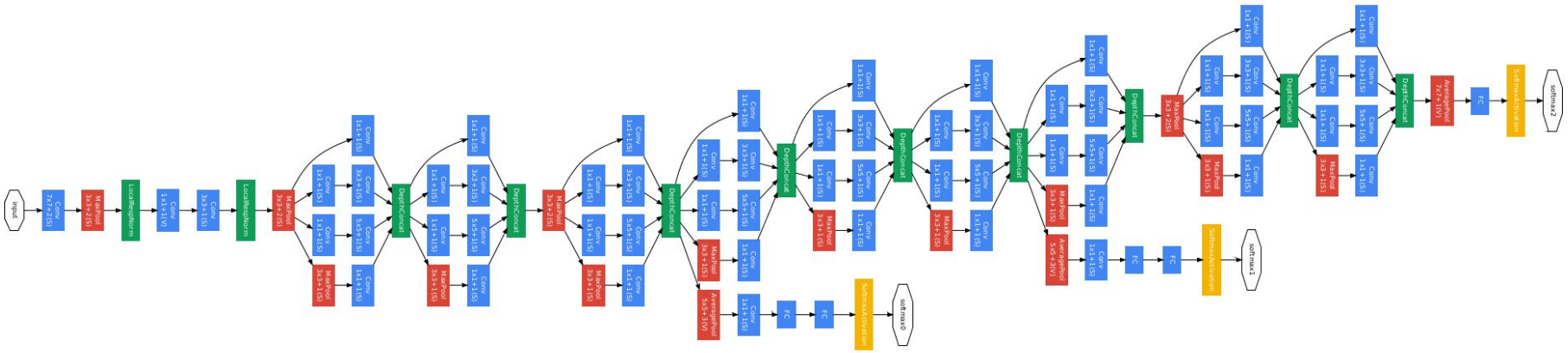
A CNN is a sequence of convolution layers and nonlinearities

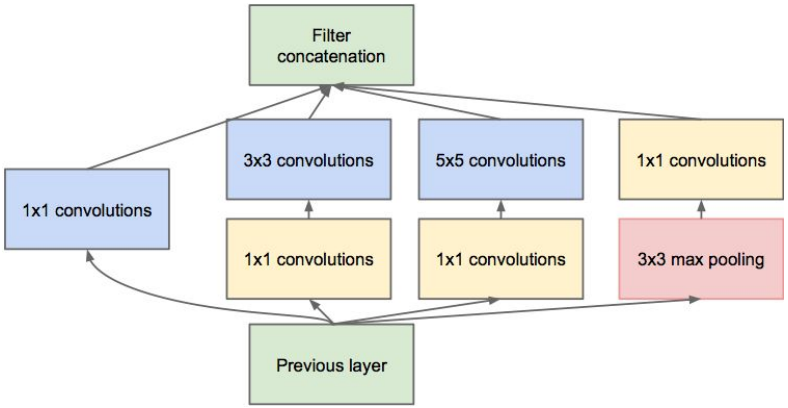


Slide credit: CS 231n, Lecture 7

Case Study: GoogLeNet

[Szegedy et al., 2014]





Inception module

ILSVRC 2014 winner (6.7% top 5 error)

Slide credit: CS 231n, Lecture 7

Recap

- Applications
- Motivation
- Supervised learning
- Gradient descent
- Backpropagation
- Convolutional Neural Networks