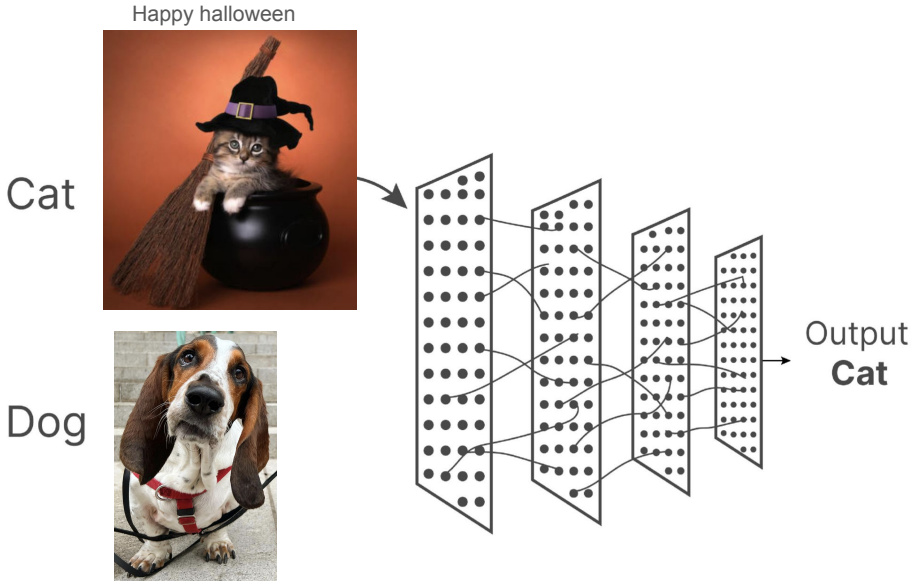
CS-GY 6643
Computer Vision

Lecture 8: Machine Learning for Computer Vision

Prof. Erdem Varol



Today's menu

Announcements

ML for CV

- From handcrafted features to flexible functions
- Optimization
- Loss function
- Cross validation

Coding lab

Announcements

Project proposal grades + feedback

Should be out by weekend.

If you lost points for either of the 4 criteria, please set up an office hours visit next week so we can provide more detailed feedback to course correct your project.

What we were looking for

1. Well motivated problem statement + background literature review (1pt)
2. You have an appropriate dataset in hand and prove it through visualization (1pt)
3. Baseline method on the data (1pt)
   a. Can be a related paper with working code
   b. or a classic algorithm if you show that there is no such paper
4. Plan for the final project, proposed method idea / architecture and evaluation metrics (1pt)

Project 2

Will be released by EOD Nov 1.

Due Nov 15, 11:59pm

Topics:

- Geometric transformations
- Hough transforms
- Segmentation

Midterm exam news

We hope to return your grades by next week.

Solutions will be released on brightspace as well.

Based on the amount of time it took everyone, we are considering grading everyone out of best of N points.

For example, if N=80

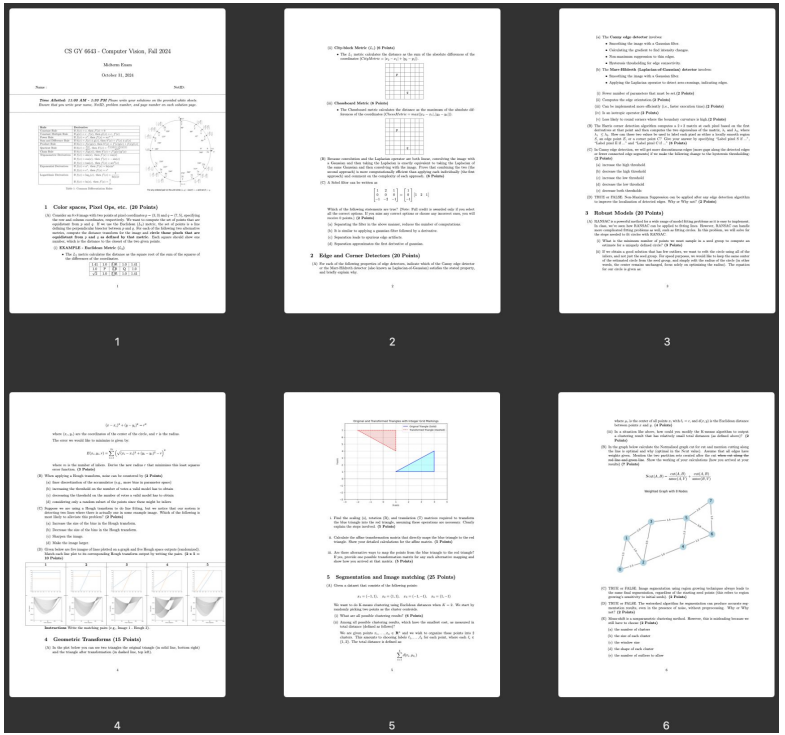
If you got 86%, then 86/80 -> 107%

If you got 75%, then 75/80 -> 94%

Goal is to reward those whole did well, and at the same time push the lower quantiles up.

We will announce the cutoff point on bright space.

We thank you for your hard work during the exam… we will be fair and kind!

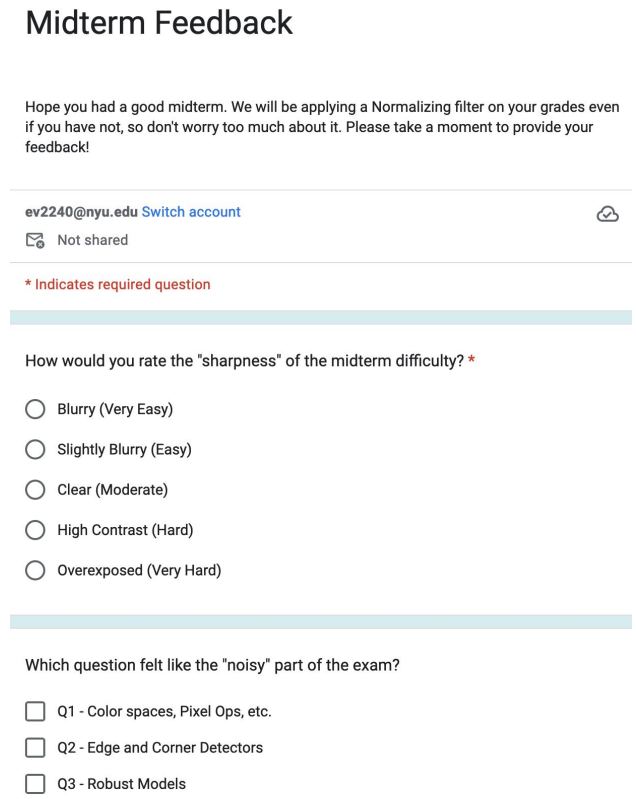


Midterm Exam feedback survey

Anonymous survey on the midterm difficulty, types of questions posed and how well did the practice midterm prepare you for the actual midterm.

https://forms.gle/77aoYK9wLWQCWeY19

Your feedback will help us gauge “the curve” for this year and design next year’s midterm better.



Midterm Feedback

Hope you had a good midterm. We will be applying a Normalizing filter on your grades even if you have not, so don't worry too much about it. Please take a moment to provide your feedback!

ev2240@nyu.edu Switch account

Not shared

* Indicates required question

How would you rate the "sharpness" of the midterm difficulty? *

- Blurry (Very Easy)
- Slightly Blurry (Easy)
- Clear (Moderate)
- High Contrast (Hard)
- Overexposed (Very Hard)

Which question felt like the "noisy" part of the exam?

- Q1 - Color spaces, Pixel Ops, etc.
- Q2 - Edge and Corner Detectors
- Q3 - Robust Models

Mid semester course feedback open till Nov 4

https://coursefeedback.nyu.edu/nyu/

How to guide:

https://www.nyu.edu/students/student-information-and-resources/registration-records-and-graduation/final-exams-and-course-evaluations/course-evaluation/course-feedback-students.html

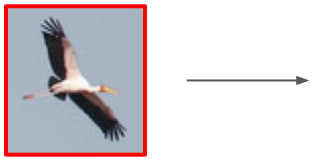
Totally anonymous.

Remember, this is the first time this is taught by me, your feedback will help me revise course delivery approaches from now until the end of the semester and next year.

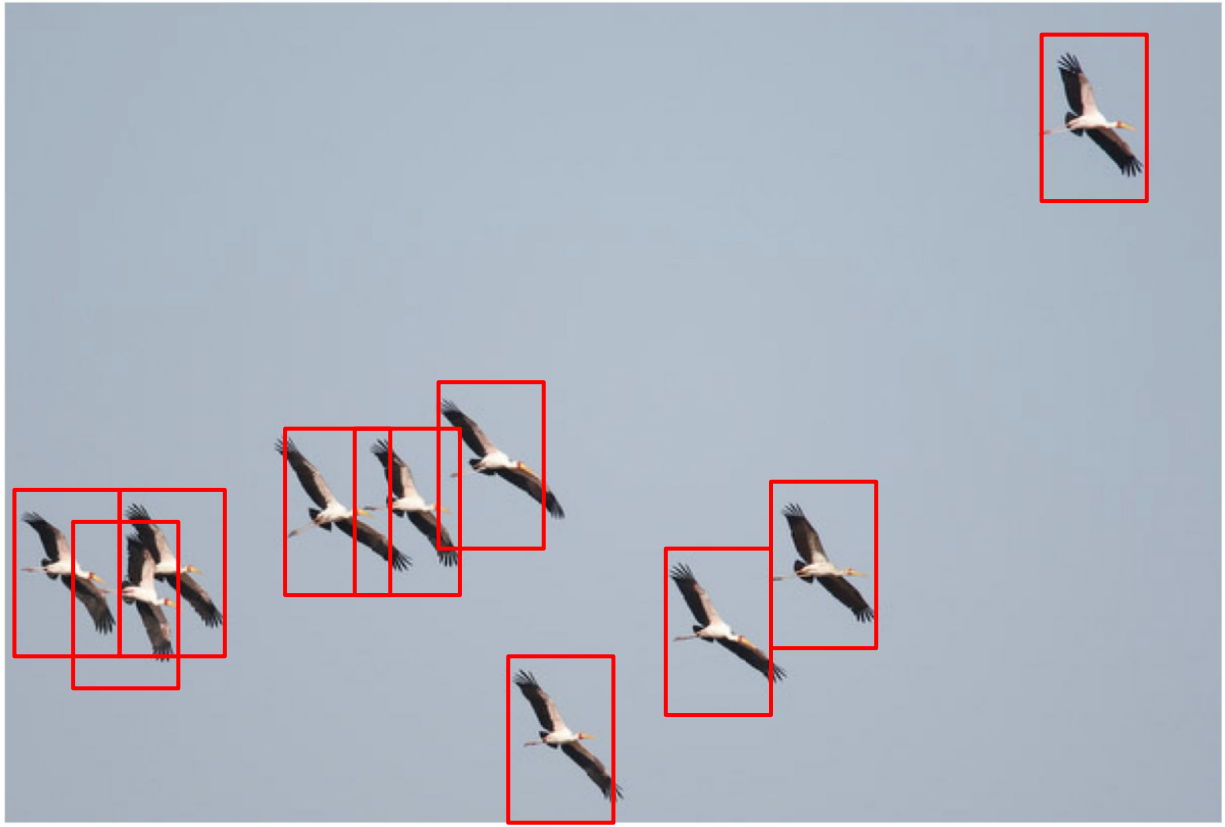
Machine learning for CV

Suppose we want to detect birds…

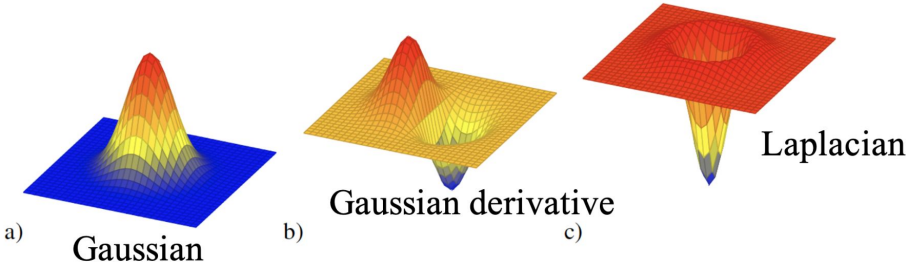
Suppose we want to detect birds…

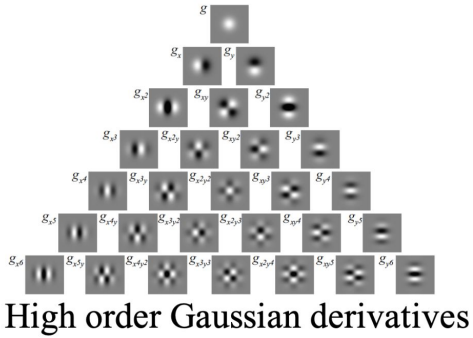


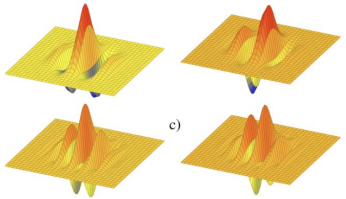
Apply translation invariant filter...



We can design a “bird” filter out of our toolbox of filters…

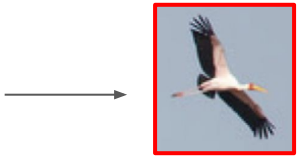



High order Gaussian derivatives



Gabor

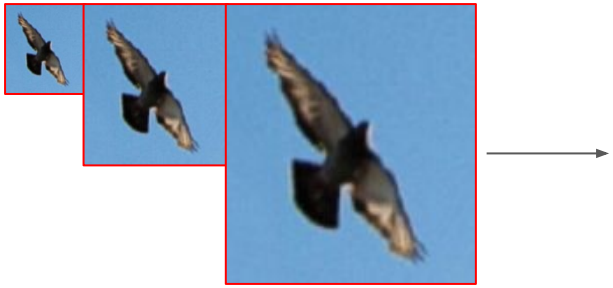
And many more…



Suppose we want to detect *more difficult* birds…

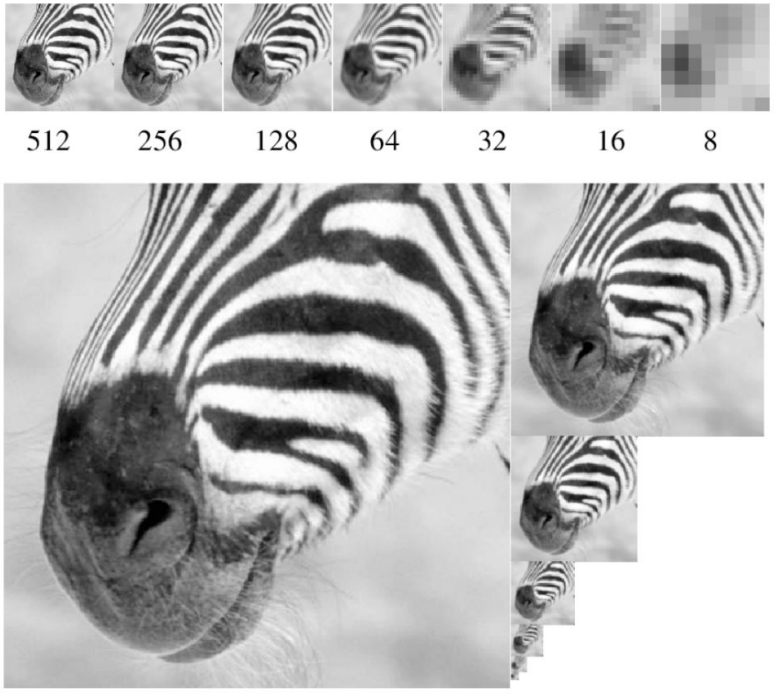
What kind of filter can we use here?

Suppose we want to detect *more difficult* birds…

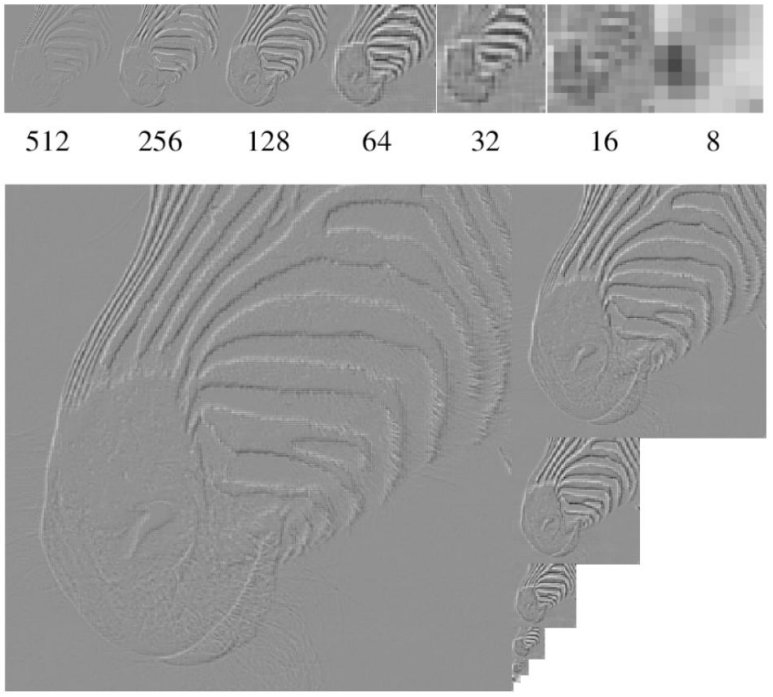


We need translation + scale invariance…

Use image pyramids



Gaussian Pyr



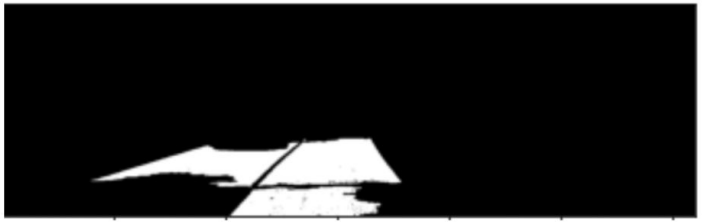
Laplacian Pyr

Suppose we want detect even *more difficult* birds…

What kind of filter can we use here?

It gets complicated, real quick.

Another example - we want to segment roads…



Model-driven approach

- Image pre-processing
  - *Histogram equalization*
- Color-based segmentation
  - *Pick a representative pixel*
  - *Thresholding all pixels*
- Image post-processing
  - *Remove segments that are small*

Human perception

Natural images are complex, what does our model miss?

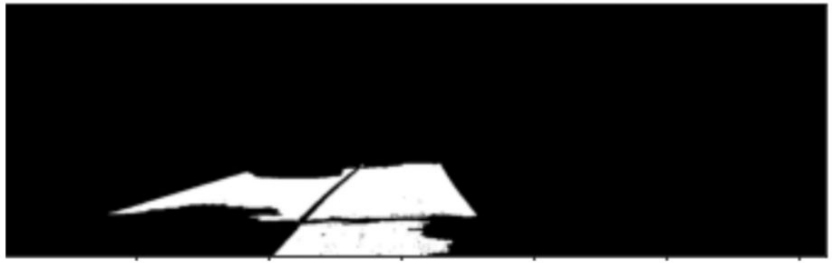


Image processing model

- Shadow: natural light change
- Lane line: human paint
- Cars: occluding objects
- …

Human perception

What's the best filter design when images and tasks get complex?

- So far, everything we covered in class is manually designed.
  - Corner and edge detectors
    - Need to specify the filter design (e.g. directional derivatives)
  - Hough transform
    - Need to specify the model (e.g. lines or circles)
  - RANSAC
    - Need to specify the model (e.g. linear or affine, or homography)
  - GraphCut
    - Need to specify the affinity model
- <u>Can we "learn" these from scratch?</u>

$$\frac{1}{4}\begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

$$\frac{1}{2}\begin{bmatrix} 1 & -2 & 1 \end{bmatrix}$$

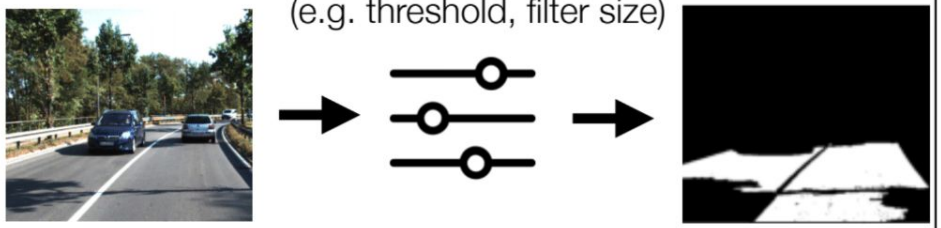
Corner filter

Normalized cuts

Almost all computer vision systems today are learning based - and they significantly outperform classical techniques.

Course paradigm shift

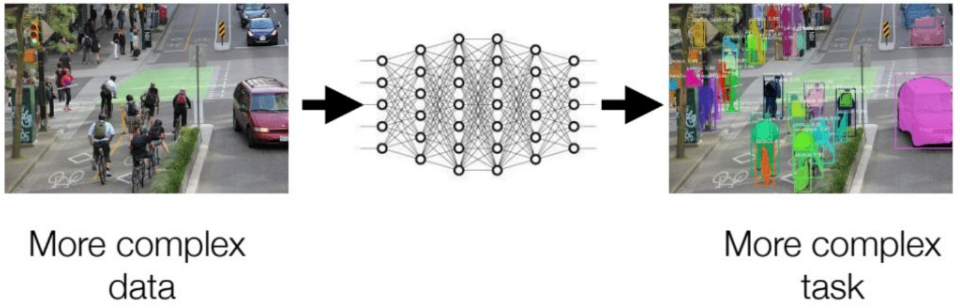
Model based CV: Week 1-7



- Shallow: limited combinations
- Designed by experts

Can make midterm questions

Learning based CV: Week 8→end



- **Deep**: sophisticated function (millions of parameters)
- **Learning**: learned from data

Hard to make final exam questions
-> final project instead

What is meant by “learning”?

Model based vs. learning based CV

- Fixed engineered features (or kernels) + trainable classifier

hand-crafted Feature Extractor → "Simple" Trainable Classifier →

Model assumptions go here

VS.

- End-to-end learning / feature learning / deep learning

Trainable Feature Extractor → Trainable Classifier →

Slide credit: LeCun

Classical computer vision pipeline

Object recognition (2006 – 2012)

SIFT
HoG

K-means
Sparse Coding

Pooling

Classifier

fixed

unsupervised

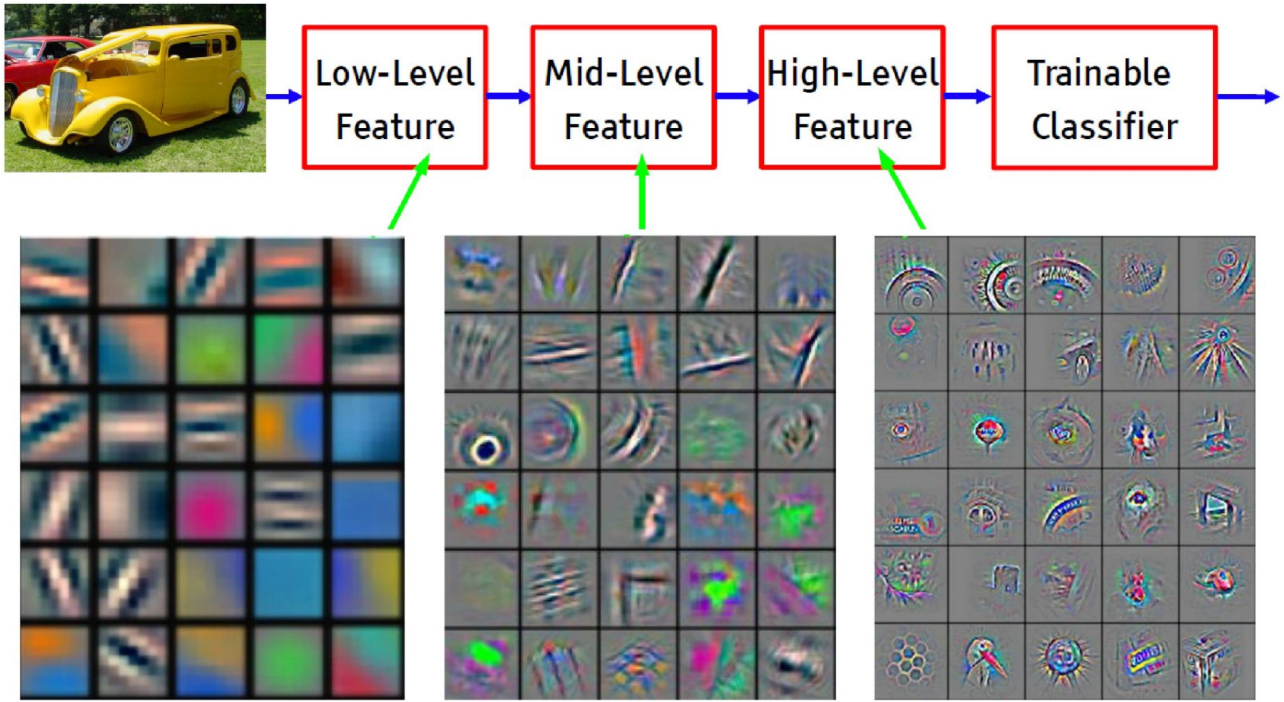
supervised

Low level features

Mid level features

Slide credit: LeCun

End-to-end computer vision pipeline

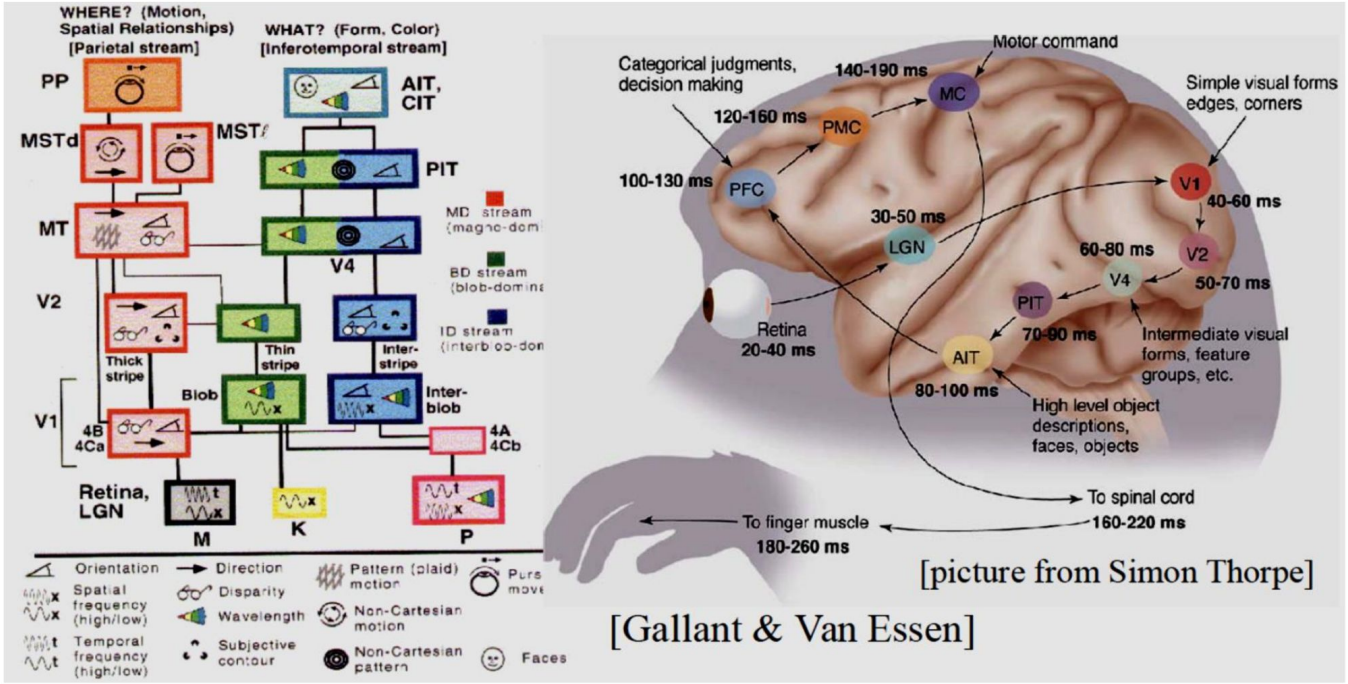


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Slide credit: LeCun

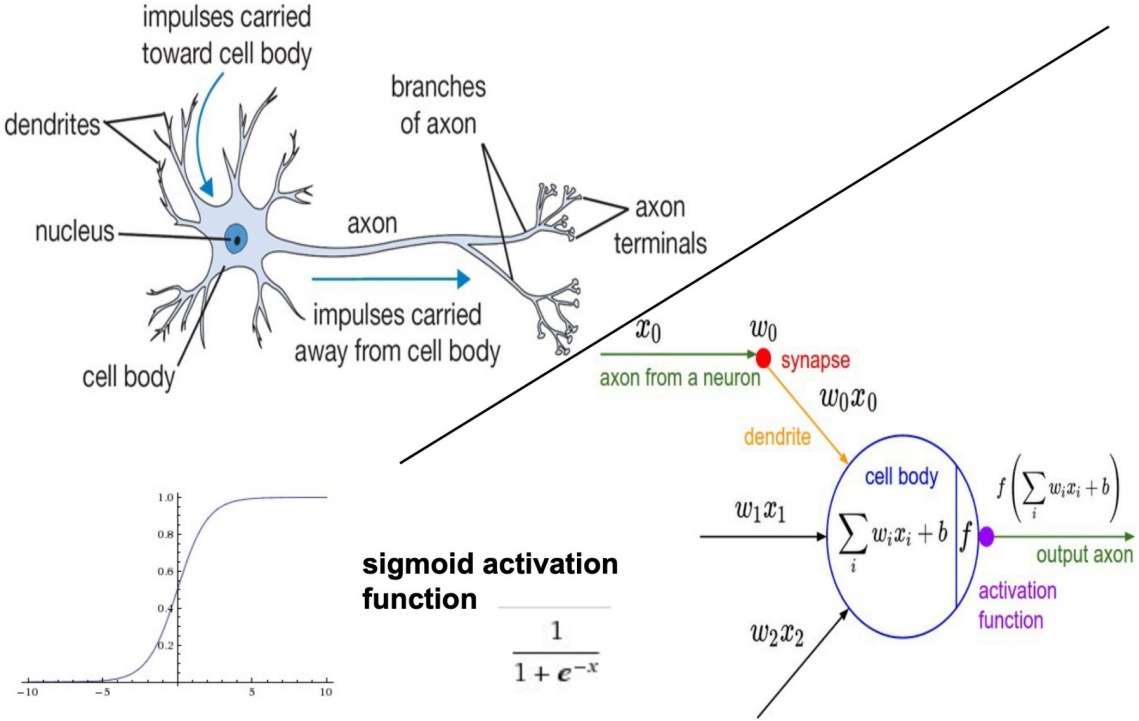
End-to-end object CV pipeline resembles our brain

The ventral (recognition) pathway in the visual cortex has multiple stages



Slide credit: LeCun

Brain communication can be modelled through activation functions

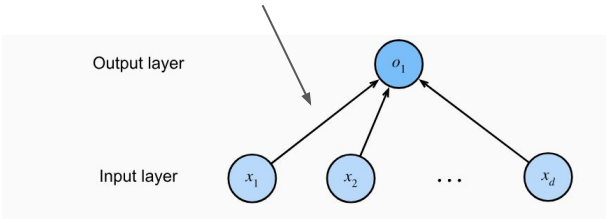


Note: modeling here is much lower level than classical CV, but still modeling.

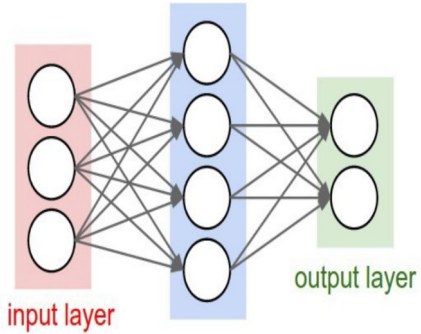
Slide credit: Andrej Karpathy

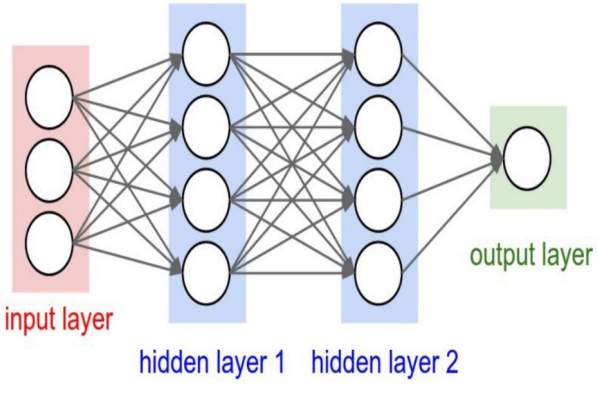
Simple activation functions can be composed to create much more complex functions

Activation functions



Single layer neural net





"2-layer Neural Net", or
"1-hidden-layer Neural Net"

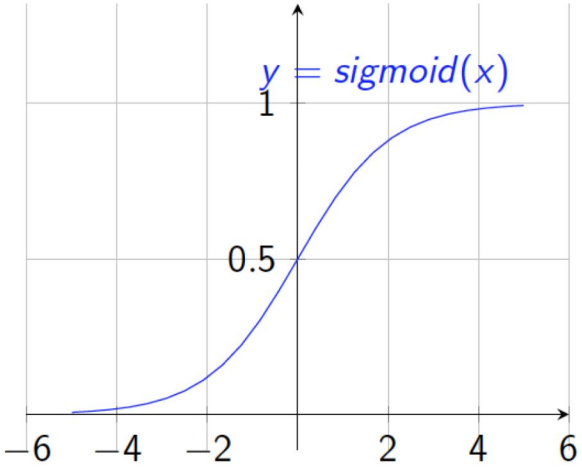
"3-layer Neural Net", or
"2-hidden-layer Neural Net"

"Fully-connected" layers

Slide credit: Andrej Karpathy

Activation function examples - sigmoid

$$f(a) = sigmoid(a) = \frac{1}{1+e^{-a}}$$

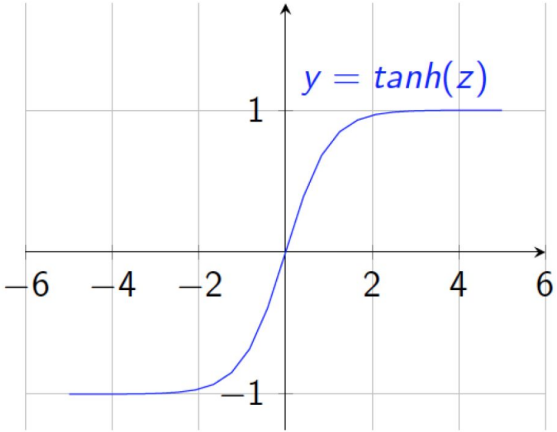


- Interpretation as firing rate of neuron
- Bounded between [0,1]
- Saturation for large +ve,-ve inputs
- Gradients go to zero
- Outputs centered at 0.5 (poor conditioning)
- Not used in practice

Slide credit: Antonio Torralba

Activation function examples - tanh

$$f(a) = \tanh(a) = \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}}$$

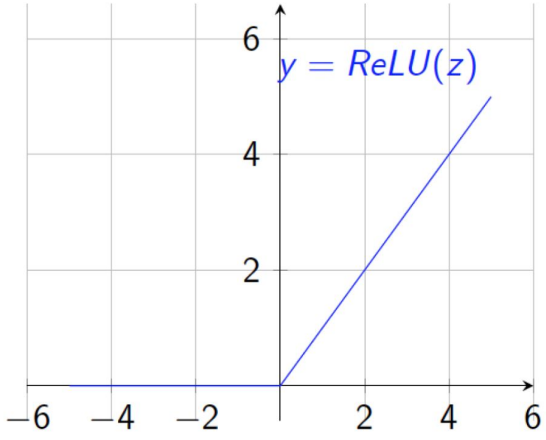


- Bounded between [-1,+1]
- Saturation for large +ve,-ve inputs
- Gradients go to zero
- Outputs centered at 0
- Preferable to sigmoid

$$\tanh(x) = 2\,\text{sigmoid}(2x) - 1$$

Activation function examples - Rectified linear unit (ReLU)

$$f(a) = \max(a, 0)$$



- Unbounded output (on positive side)
- Efficient to implement:

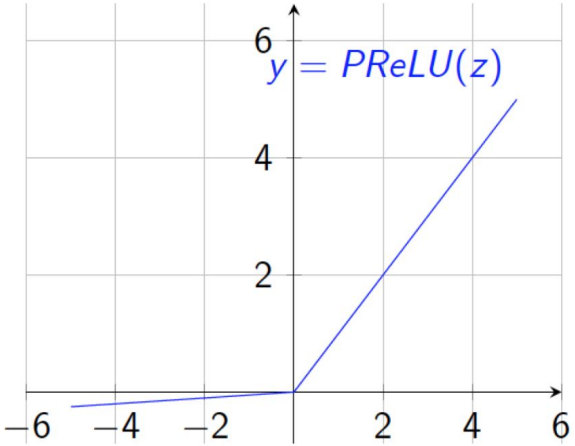
$$f'(a) = \frac{df}{da} = \begin{cases} 0 & a < 0 \\ 1 & a \geq 0 \end{cases}$$

- Also seems to help convergence (see 6x speedup vs tanh in [Krizhevsky et al.])
- Drawback: if strongly in negative region, unit is dead forever (no gradient).
- Default choice: widely used in current models.

Slide credit: Antonio Torralba

Activation function examples - Leaky ReLU

$$f(a) = \begin{cases} \max(0,a) & a > 0 \\ a \min(0,a) & a < 0 \end{cases}$$

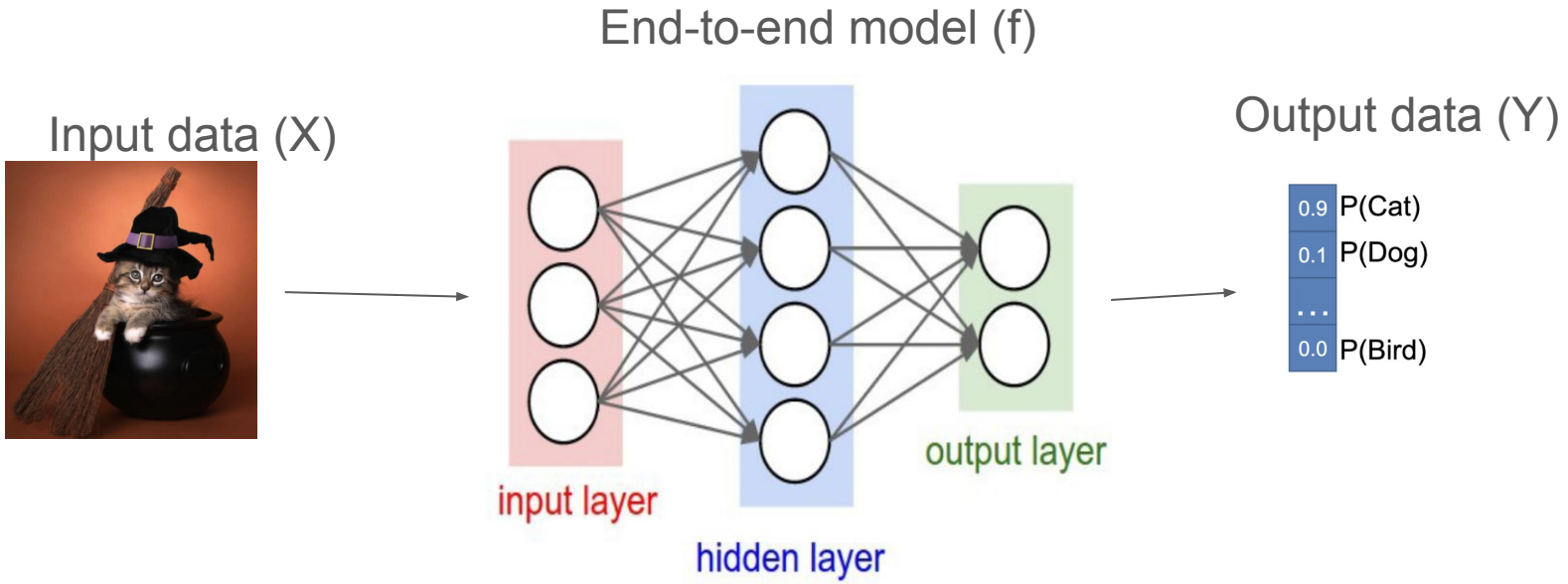


- where α is small (e.g. 0.02)
- Efficient to implement:

$$f'(a) = \frac{df}{da} = \begin{cases} -a & a < 0 \\ 1 & a > 0 \end{cases}$$

- Also known as probabilistic ReLU (PReLU)
- Has non-zero gradients everywhere (unlike ReLU)
- α can also be learned (see Kaiming He et al. 2015).

These complex functions then can be used map input/output relations that we want (ML in a nutshell)



$$f(X) \rightarrow Y$$

ML Problems in Vision

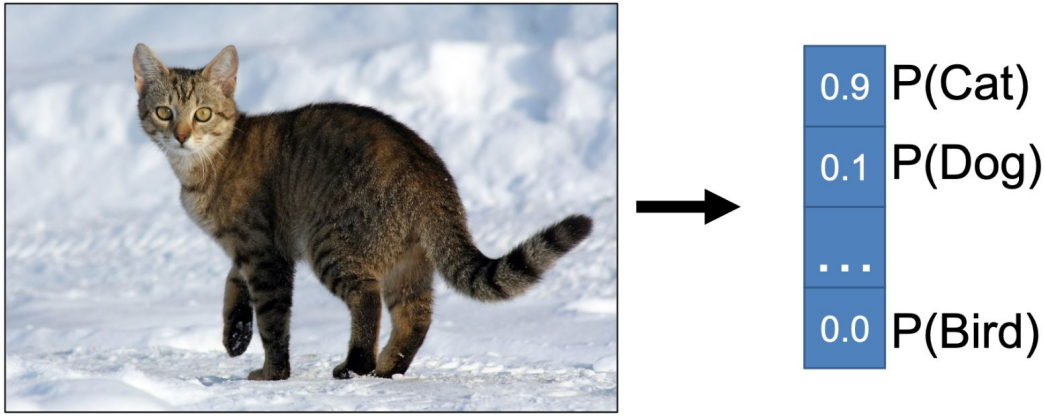
ML Problem Examples in Vision

	Supervised (Data+Labels)	Unsupervised (Just Data)
Discrete Output	Classification/ Categorization	
Continuous Output		

ML Problem Examples in Vision

***Cat*egorization/Classification**

Binning into K mutually-exclusive categories



ML Problem Examples in Vision

	Supervised (Data+Labels)	**Unsupervised (Just Data)**
Discrete Output	Classification/ Categorization	
Continuous Output	**Regression**	

ML Problem Examples in Vision

Regression

Estimating continuous variable(s)

Cat weight

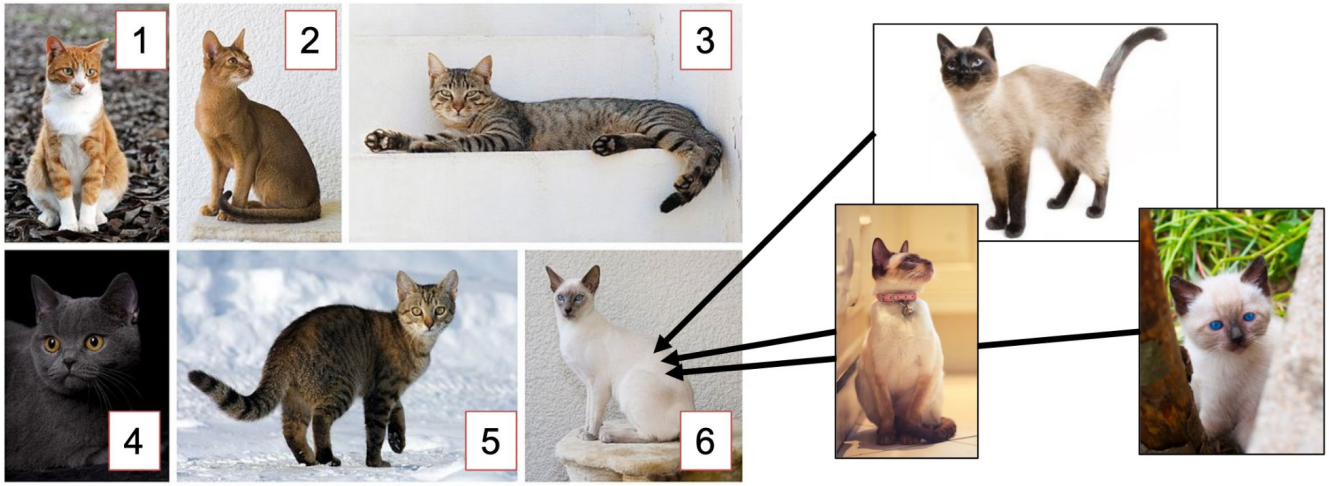
ML Problem Examples in Vision

	Supervised (Data+Labels)	**Unsupervised (Just Data)**
Discrete Output	Classification/ Categorization	**Clustering**
Continuous Output	Regression	

ML Problem Examples in Vision

Clustering

Given a set of cats, automatically discover clusters or *cat*egories.



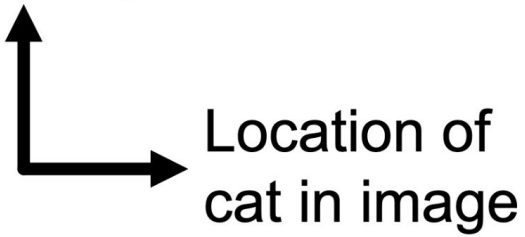
ML Problem Examples in Vision

	Supervised (Data+Labels)	**Unsupervised (Just Data)**
Discrete Output	Classification/ Categorization	Clustering
Continuous Output	Regression	**Dimensionality Reduction**

ML Problem Examples in Vision

Dimensionality Reduction

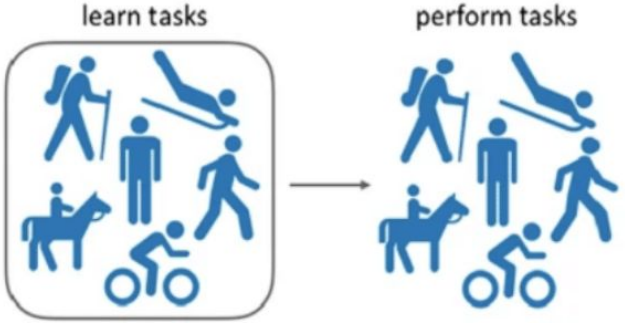
Find dimensions that best explain the whole image/input



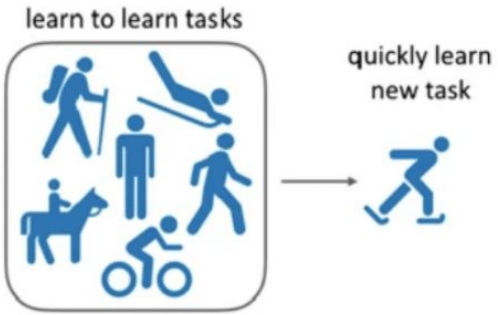
For ordinary images, this is currently a totally hopeless task. For certain images (e.g., faces, this works reasonably well)

Other ML problem types that don't fit into these 4 categories

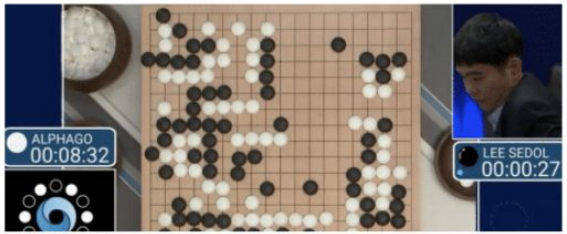
Multi-task Learning



Meta-Learning



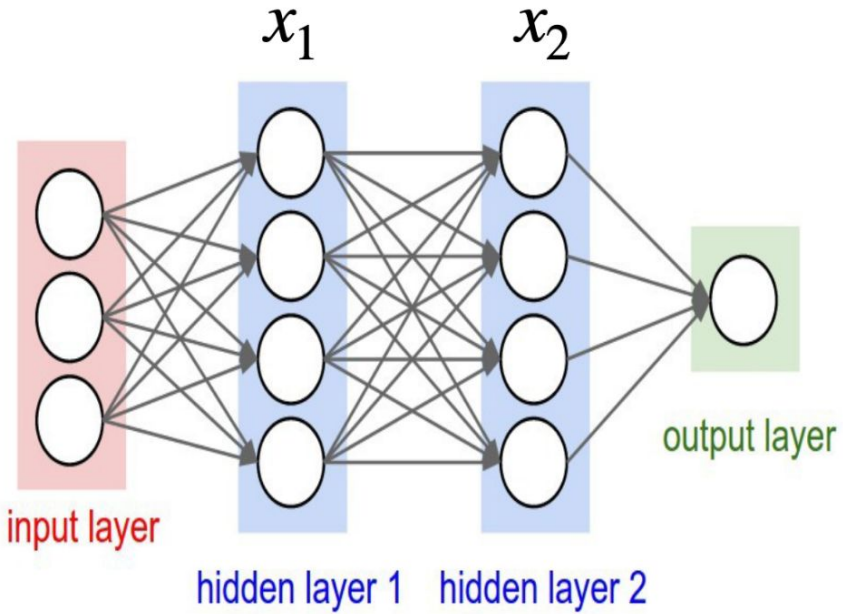
Reinforcement Learning



Environment: Dynamic (X,Y)

- Input: sequence of actions
- Label: reward/punishment (e.g., win, lose)

These functions have parameters that need to be fit to satisfy the input output relationship



$$x_i \in \mathbb{R}^{H \times 1}$$

$$W_i \in \mathbb{R}^{D \times H}$$

$$b_i \in \mathbb{R}^{D \times 1}$$

$$x_{i+1} = f\left(W_i x_i + b_i\right)$$

How can we solve for **W** and **b**?

We need to define an objective e.g. a function will always output something if its input something, we need to check if it's correct

x_i Input (image)

y_i Target (labels)

$\mathscr{L}$ Loss Function

θ Parameters

$f(x_i; \theta)$ Prediction

The objective of learning:

$$\min_{\theta} \sum_{i} \mathscr{L}\left(f(x_i; \theta), y_i\right)$$

Common loss functions to measure correctness

Squared error:

$$\mathscr{L}(x, y) = \|x - y\|_2^2$$

Hinge loss:

$$\mathscr{L}(x, y) = \max(0, 1 - x \cdot y)$$

Cross entropy:

$$\mathscr{L}(x, y) = -\sum_i y_i \log x_i$$

Choice of loss function is an act of modeling.

So far we have…

Data - e.g. images of cats and dogs

A function to fit with free variables- e.g. multiple layer neural network with weights, and biases

Output to achieve - e.g. cat images → text label cat, dog images → text label dog

A loss function - if cat image → text label dog, what is the penalty for mistakes?

What's next? Fitting. We want to find the function weights that minimize the loss.

Goal: find the **w** minimizing some loss function L.

$$\arg\min_{\boldsymbol{w}\in R^N} L(\boldsymbol{w})$$

Works for lots of different Ls:

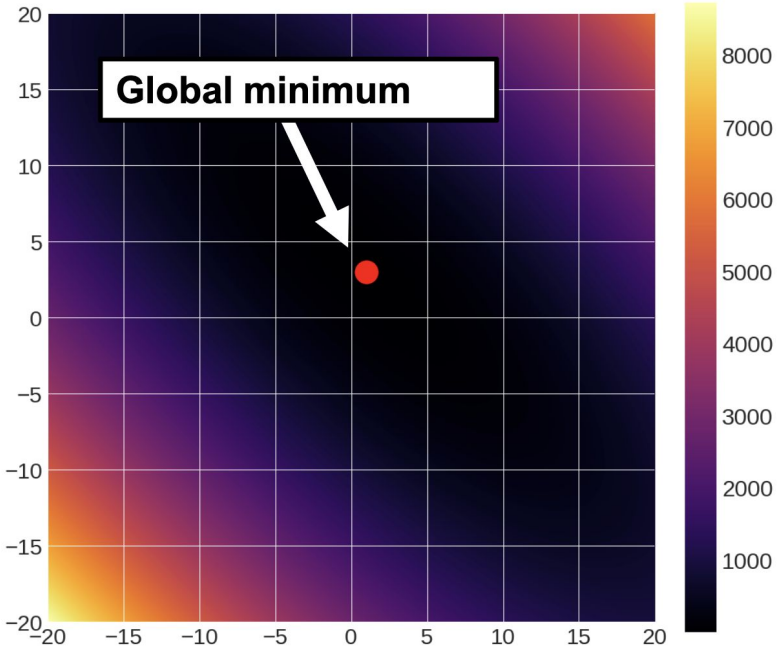
$$L(\boldsymbol{W}) = \boldsymbol{\lambda}\|\boldsymbol{W}\|_2^2 + \sum_{i=1}^{n} -\log\left(\frac{\exp((Wx)_{y_i})}{\sum_k \exp((Wx)_k)}\right)$$

$$L(\boldsymbol{w}) = \lambda\|\boldsymbol{w}\|_2^2 + \sum_{i=1}^{n}\left(y_i - \boldsymbol{w}^T\boldsymbol{x_i}\right)^2$$

$$L(\boldsymbol{w}) = C\|\boldsymbol{w}\|_2^2 + \sum_{i=1}^{n}\max\left(0, 1 - y_i\boldsymbol{w}^T\boldsymbol{x_i}\right)$$

Sample Function to Optimize

$$f(x,y) = (x+2y-7)^2 + (2x+y-5)^2$$



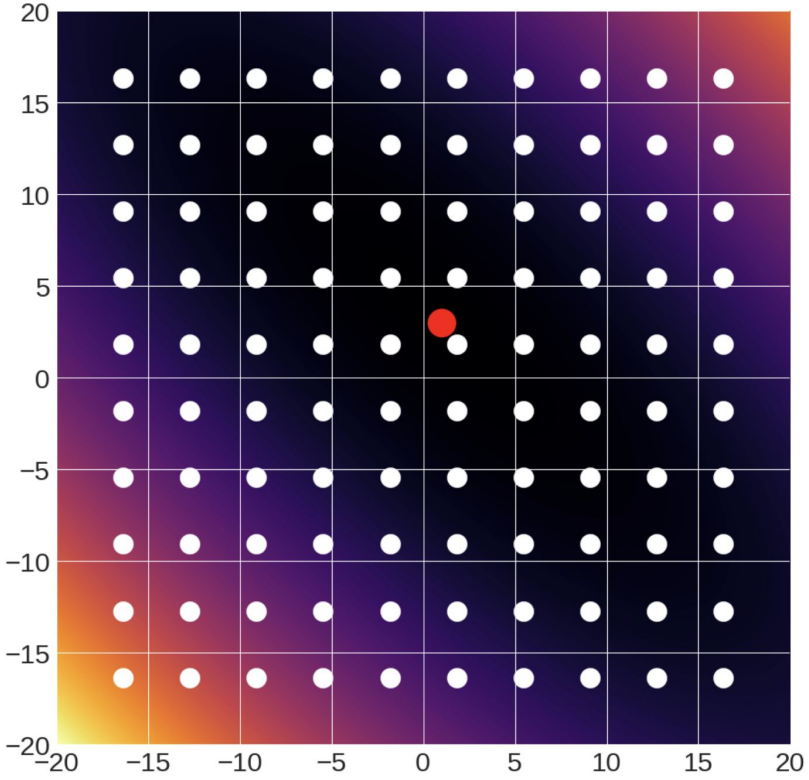
Note:

- Each point in the picture is a function evaluation
- Here it takes microseconds – so we can easily see the answer
- Functions we want to optimize may take hours to evaluate

Option #1A – Grid Search

```
#systematically try things
best, bestScore = None, Inf
for dim1Value in dim1Values:
        ….
        for dimNValue in dimNValues:
                w = [dim1Value, …, dimNValue]
                if L(w) < bestScore:
                        best, bestScore = w, L(w)
return best
```

Option #1A – Grid Search



Option #1A – Grid Search

Pros:

1. Super simple
2. Only requires being able to evaluate model

Cons:

1. Scales horribly to high dimensional spaces

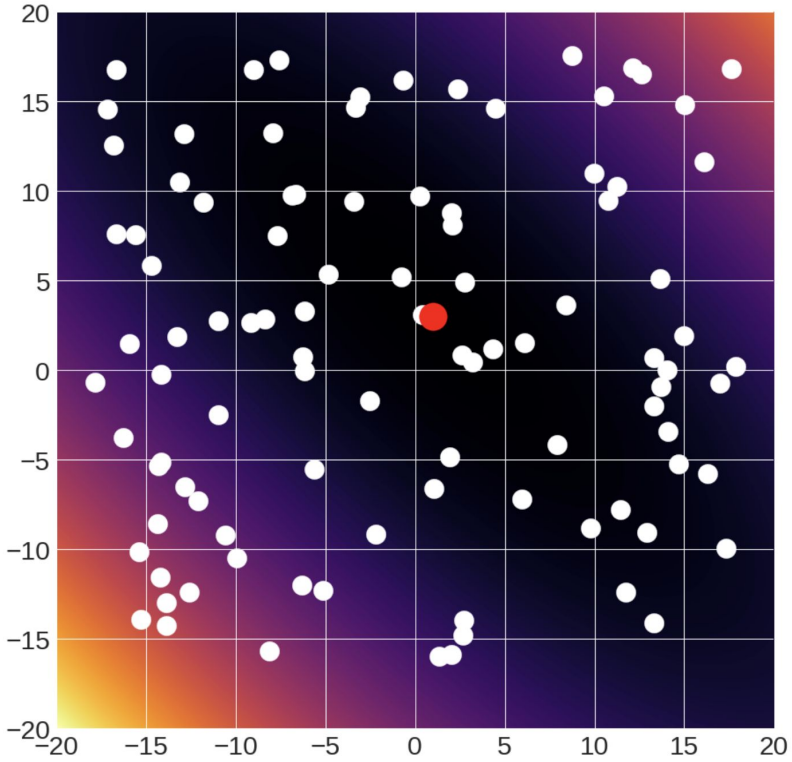
Complexity: $\text{samplesPerDim}^{\text{numberOfDims}}$

Option #1B – Random Search

```
best, bestScore = None, Inf
for iter in range(numIters):
        w = random(N,1) #sample
        score = L(w) #evaluate
        if score < bestScore:
                best, bestScore = w, score
return best
```

Does this sound familiar?

Option #1B – Random Search



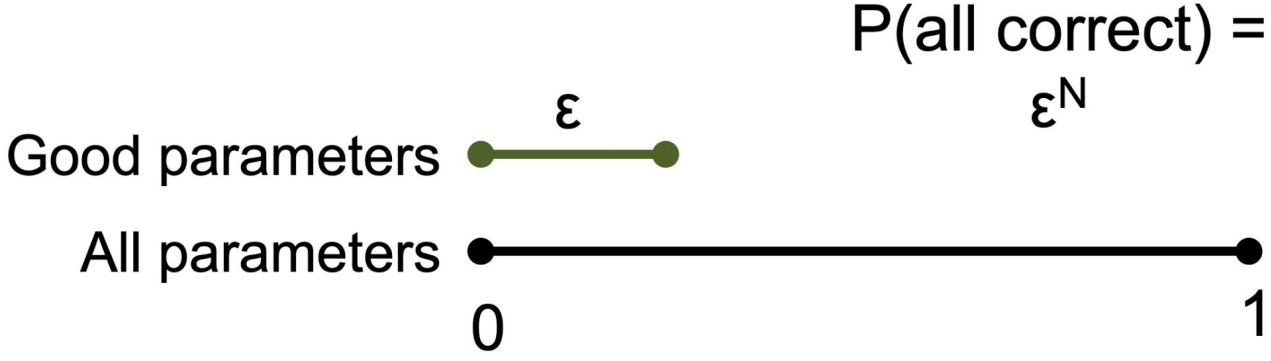
Option #1B – Random Search

Pros:

1. Super simple
2. Only requires being able to sample model and evaluate it

Cons:

1. Slow –throwing darts at high dimensional dart board
2. Might miss something



When Do You Use Options 1A/1B?

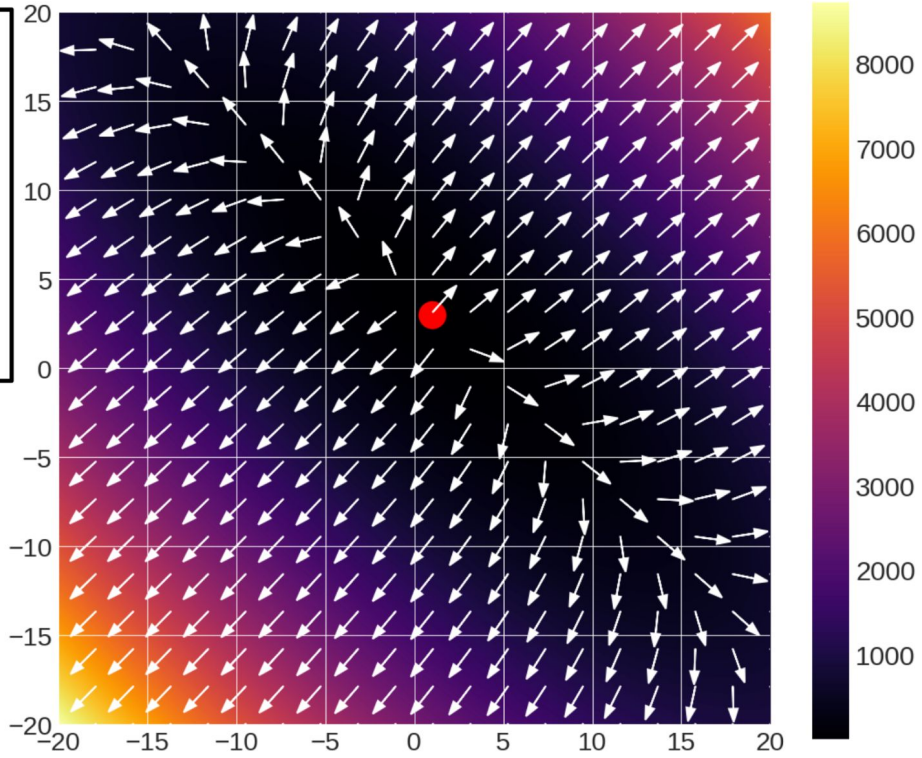
Use these when

- Number of dimensions small, space bounded
- Objective is impossible to analyze (e.g., validation accuracy if one uses a distance function)

Random search is more effective; grid search makes it easy to systematically test something (people love certainty)

Option 2 – Use The Gradient

Arrows: **gradient direction** (scaled to unit length)



Option 2 – Use The Gradient

Want: $\arg\min_{\boldsymbol{w}} L(\boldsymbol{w})$

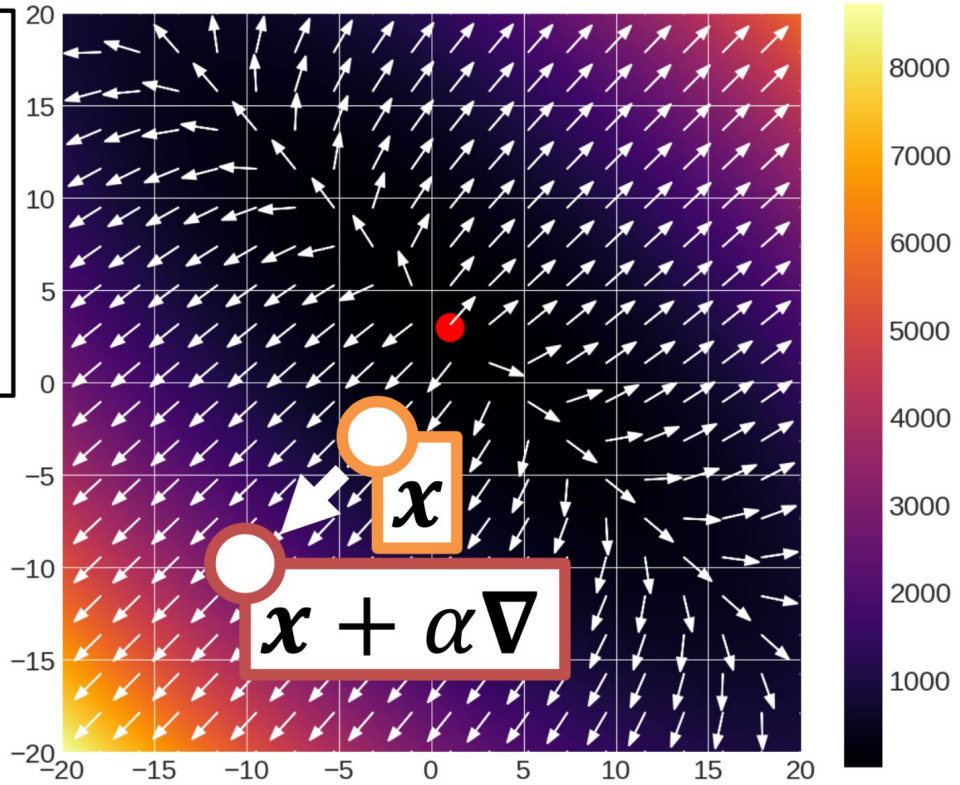
What's the geometric interpretation of: $\nabla_{\boldsymbol{w}} L(\boldsymbol{w}) = \begin{bmatrix} \partial L/\partial \boldsymbol{x}_1 \\ \vdots \\ \partial L/\partial \boldsymbol{x}_N \end{bmatrix}$

Which is bigger (for small α)?

$$L(\boldsymbol{w}) \quad \begin{matrix} \leq ? \\ > ? \end{matrix} \quad L(\boldsymbol{w} + \alpha \nabla_{\boldsymbol{w}} L(\boldsymbol{w}))$$

Option 2 – Use The Gradient

Arrows: gradient direction (scaled to unit length)



Option 2 – Use The Gradient

Method: at each step, move in direction of negative gradient

w0 = *initialize*() #initialize
for iter in range(numIters):
    g = $\nabla_{\boldsymbol{w}} L(\boldsymbol{w})$ #eval gradient
    w = **w** + -*stepsize*(iter)***g** #update w
return **w**

Quick detour - how to compute gradients?

Numerical Method:

$$\nabla_w L(\boldsymbol{w}) = \begin{bmatrix} \frac{\partial L(w)}{\partial x_1} \\ \vdots \\ \frac{\partial L(w)}{\partial x_n} \end{bmatrix}$$

How do you compute this?

$$\frac{\partial f(x)}{\partial x} = \lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x)}{\epsilon}$$

In practice, use:

$$\frac{f(x+\epsilon) - f(x-\epsilon)}{2\epsilon}$$

How many function evaluations per dimension?

Quick detour - how to compute gradients?

Analytical Method:

$$\nabla_{\boldsymbol{w}} L(\boldsymbol{w}) = \begin{bmatrix} \frac{\partial L(w)}{\partial x_1} \\ \vdots \\ \frac{\partial L(w)}{\partial x_n} \end{bmatrix}$$

Use calculus!

For simple functions we can compute the analytical gradient

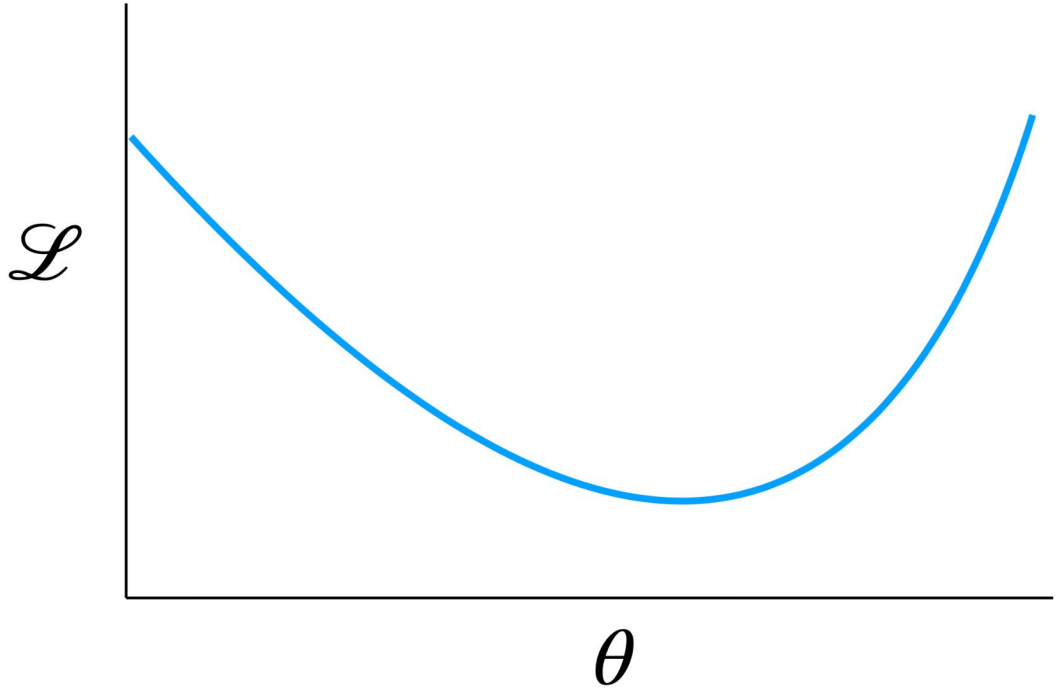
E.g. linear regression

$$L(\boldsymbol{w}) = \lambda \|\boldsymbol{w}\|_2^2 + \sum_{i=1}^{n} \left(y_i - \boldsymbol{w}^T \boldsymbol{x_i}\right)^2$$

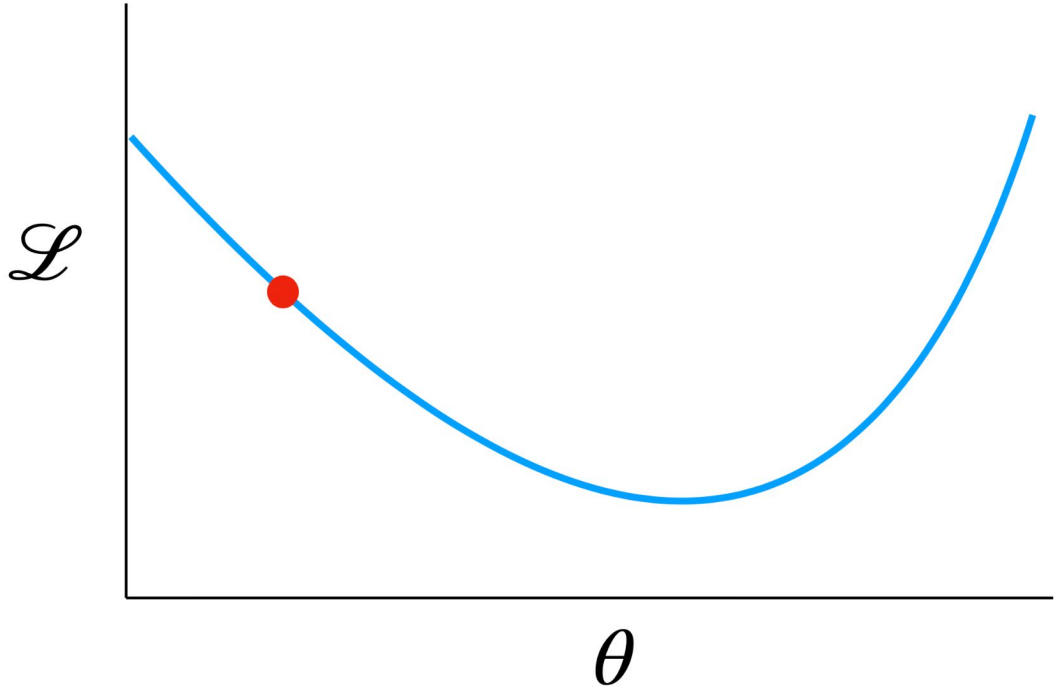
$$\downarrow \quad \frac{\partial}{\partial \boldsymbol{w}} \quad \downarrow$$

$$\nabla_{\boldsymbol{w}} L(\boldsymbol{w}) = 2\lambda \boldsymbol{w} + \sum_{i=1}^{n} -\left(2\left(y_i - \boldsymbol{w}^T \boldsymbol{x_i}\right)\boldsymbol{x}_i\right)$$

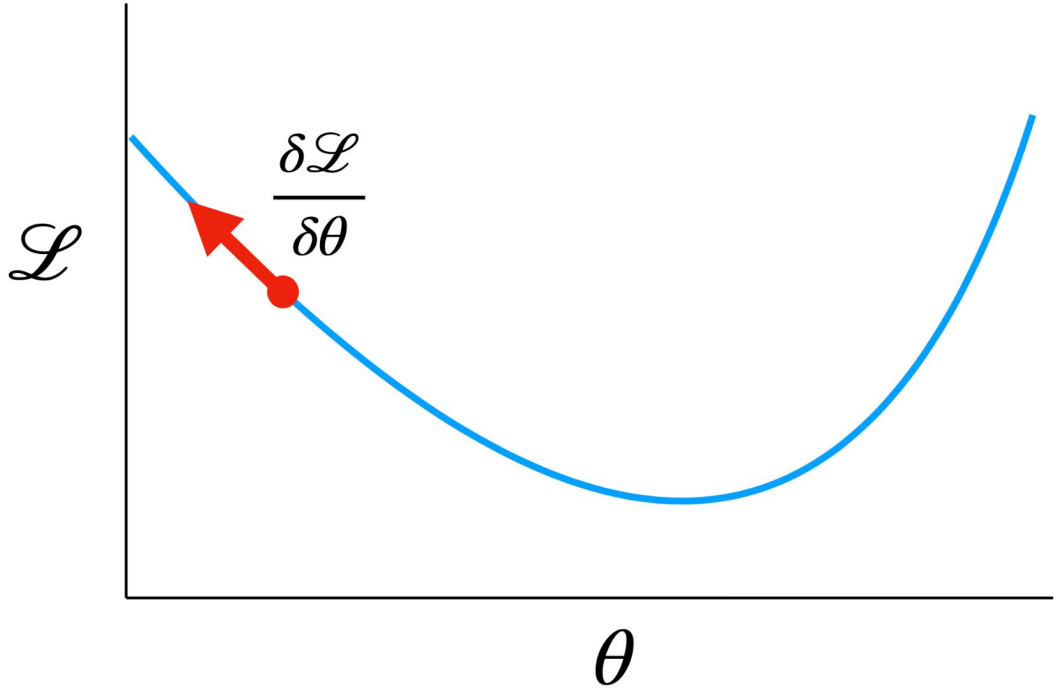
Loss Surface



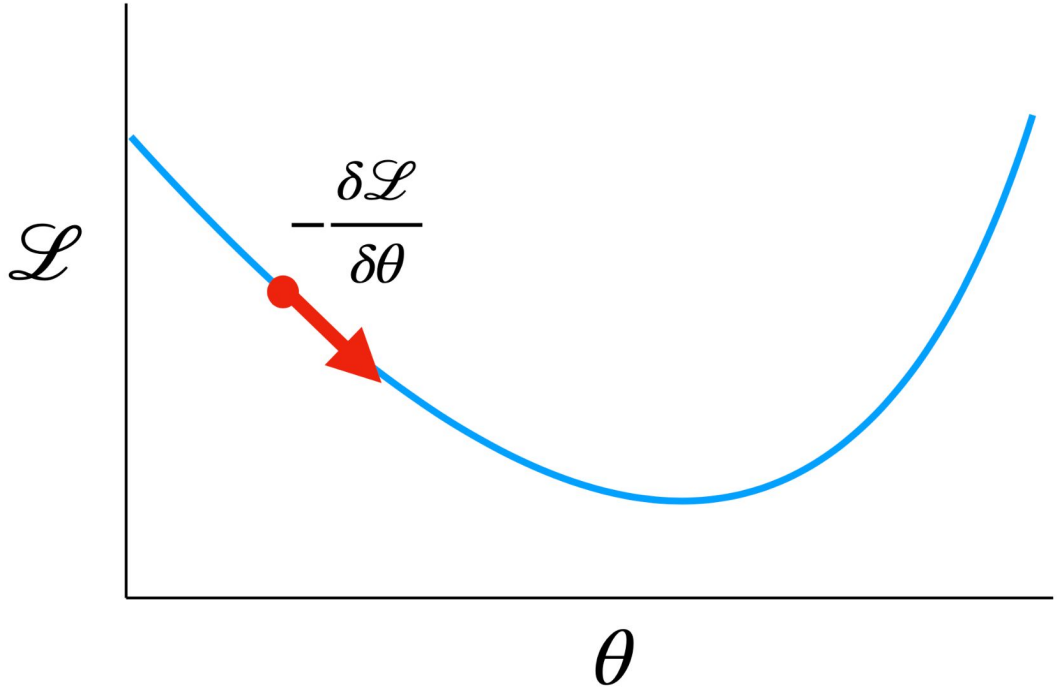
Loss Surface



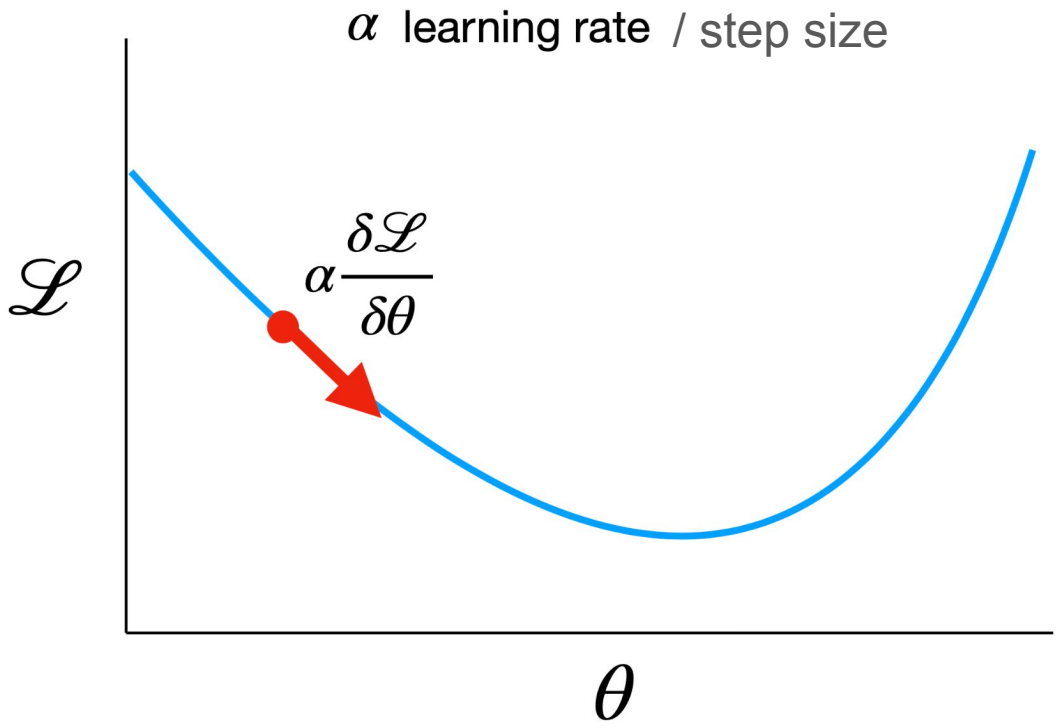
Loss Surface



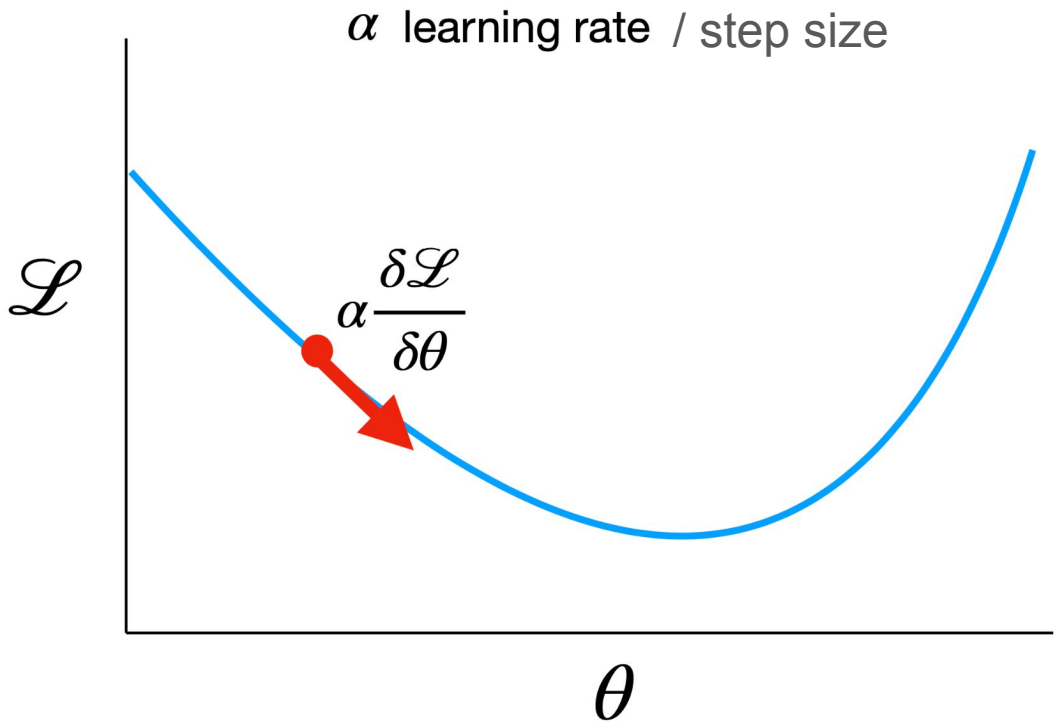
Gradient Descent



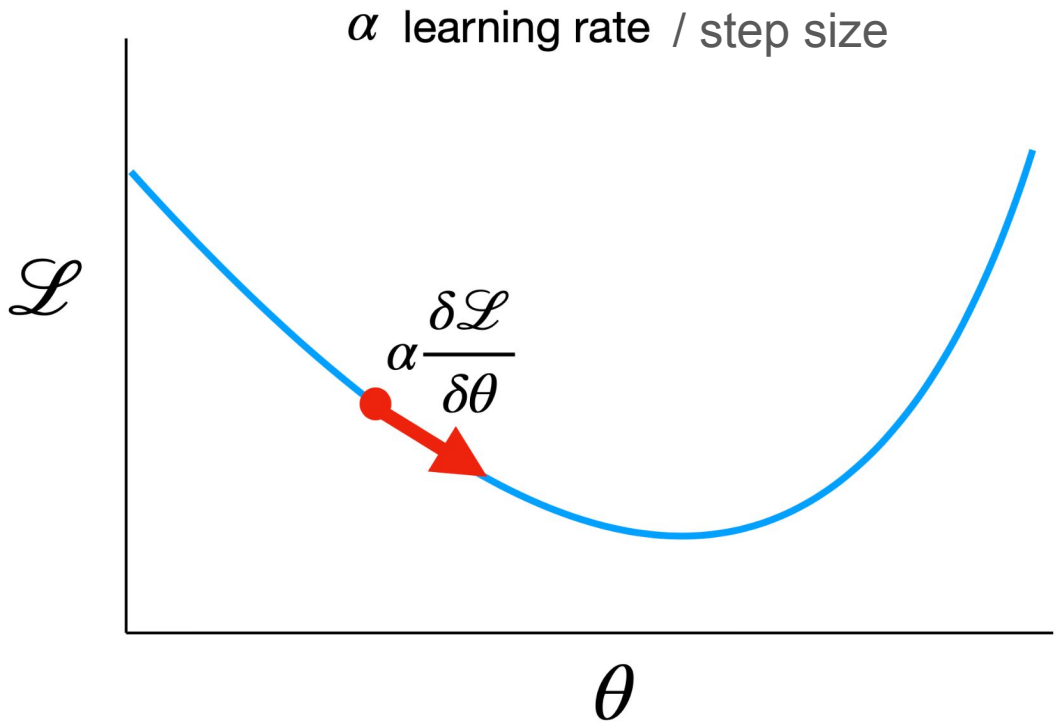
Gradient Descent



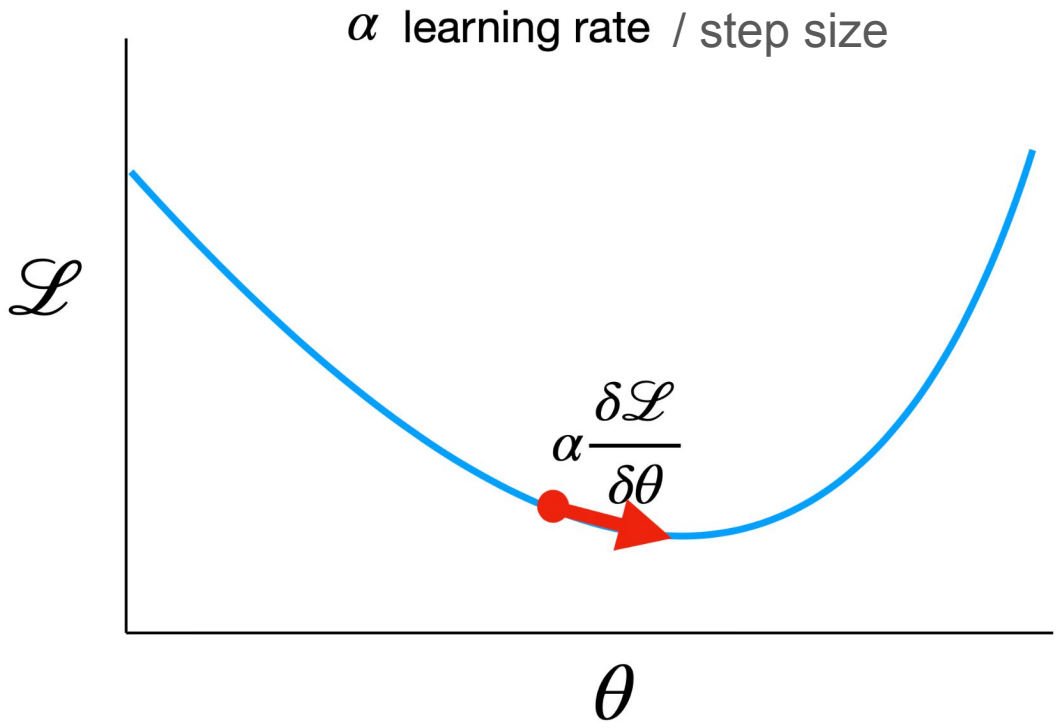
Gradient Descent



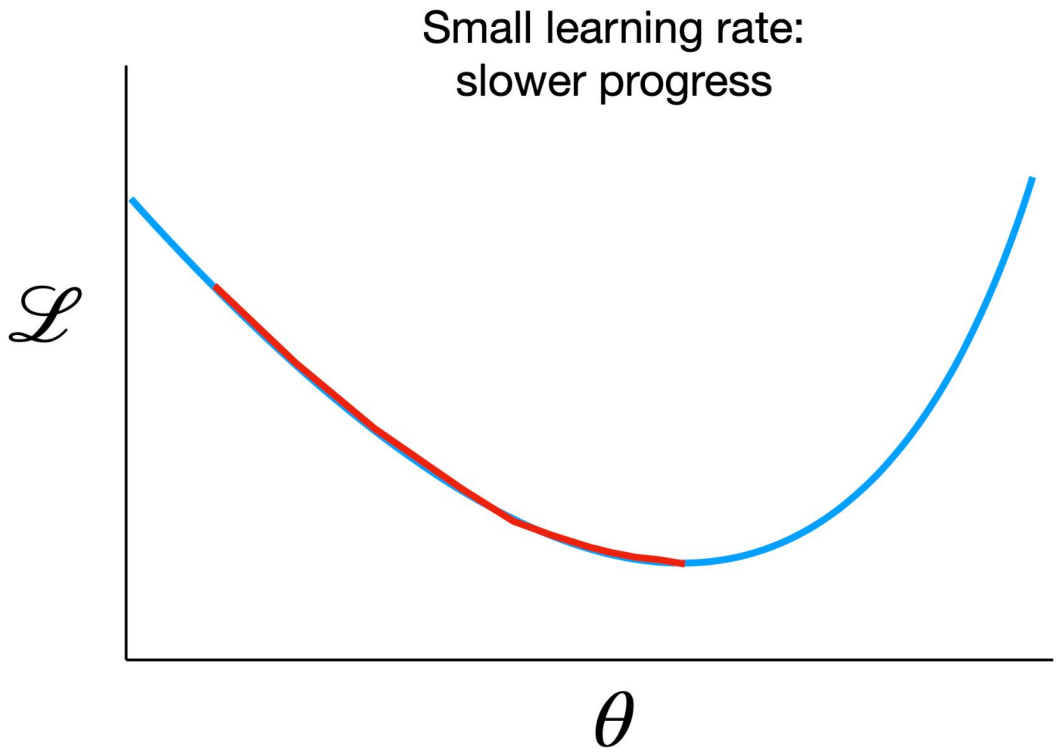
Gradient Descent



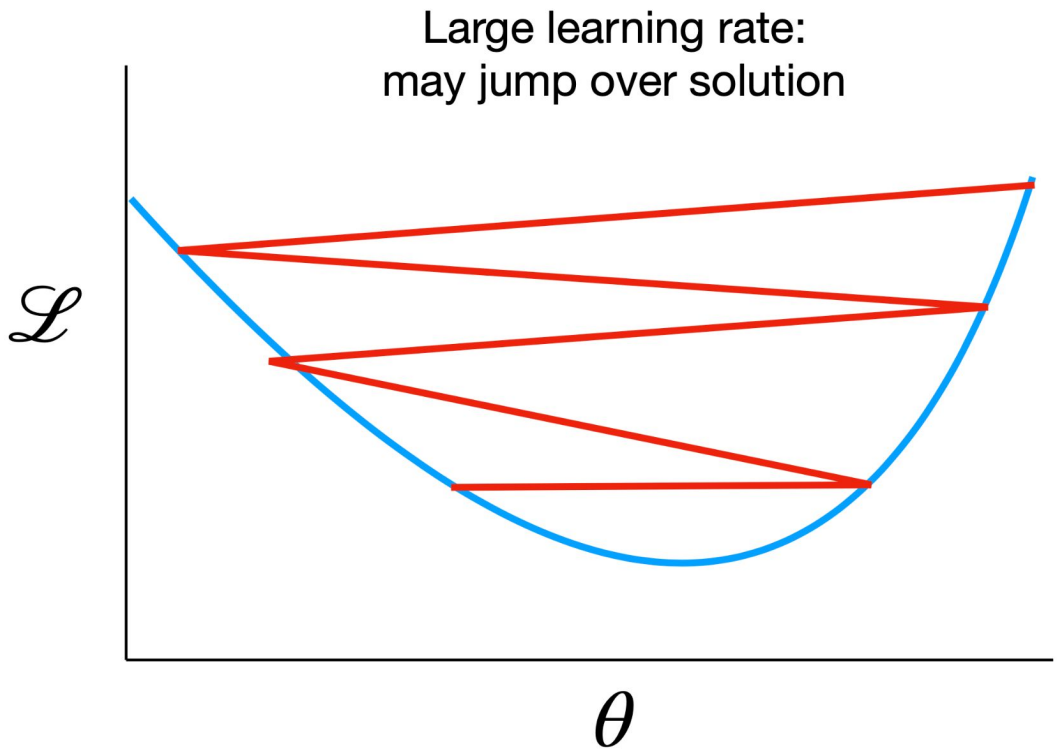
Gradient Descent



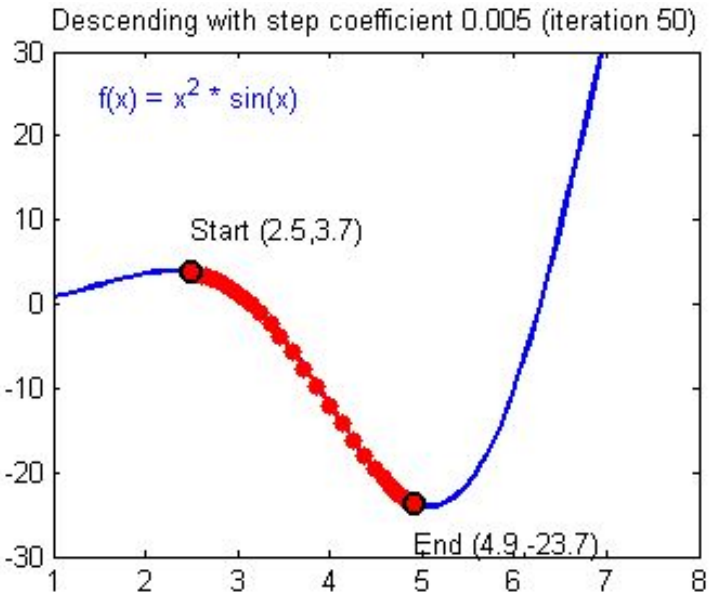
Gradient Descent



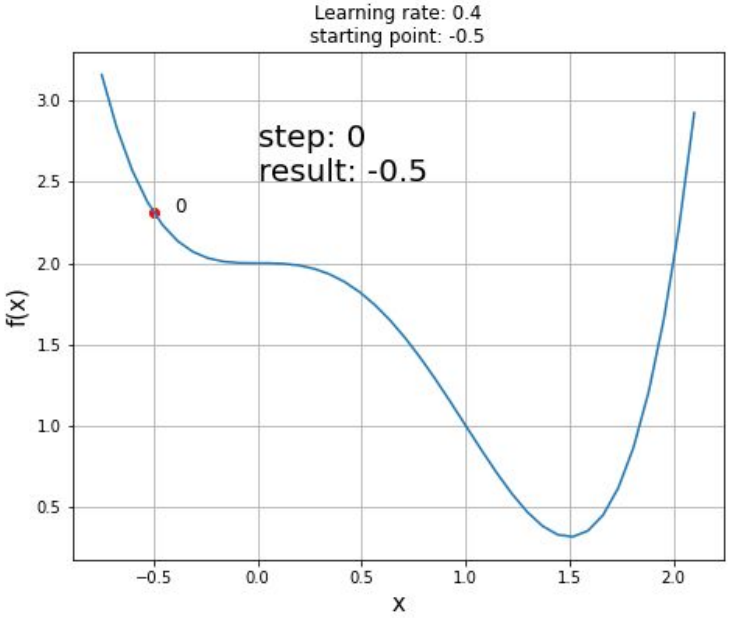
Gradient Descent



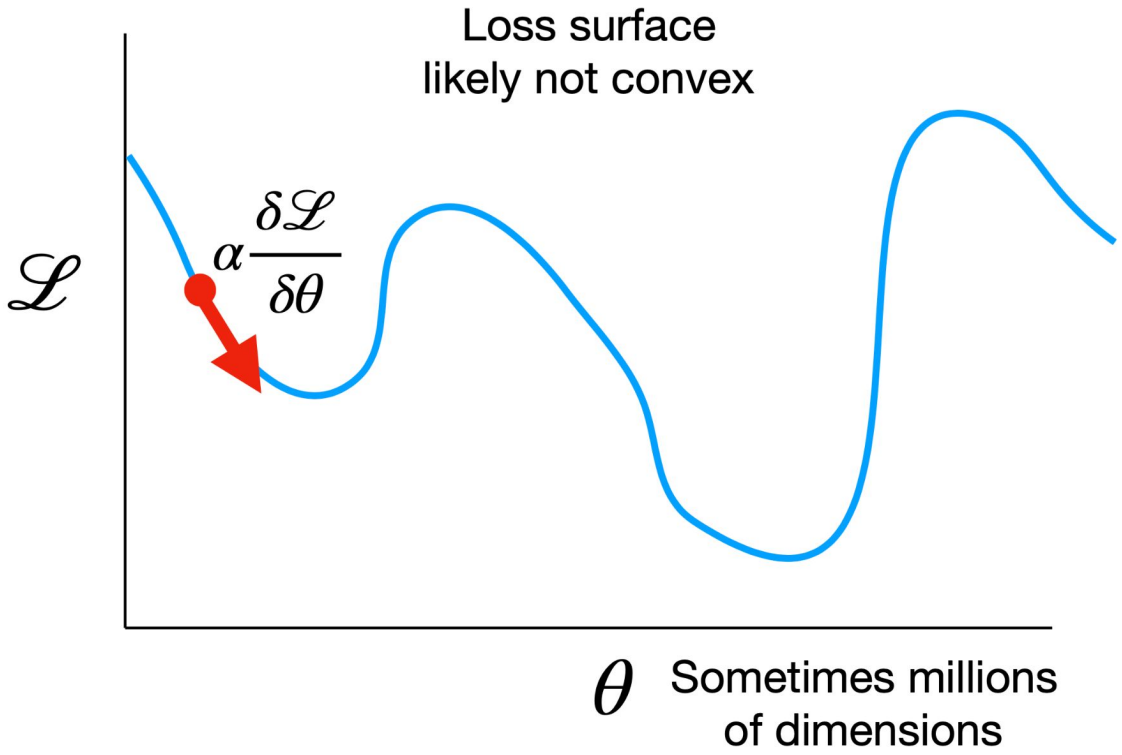
Good step size

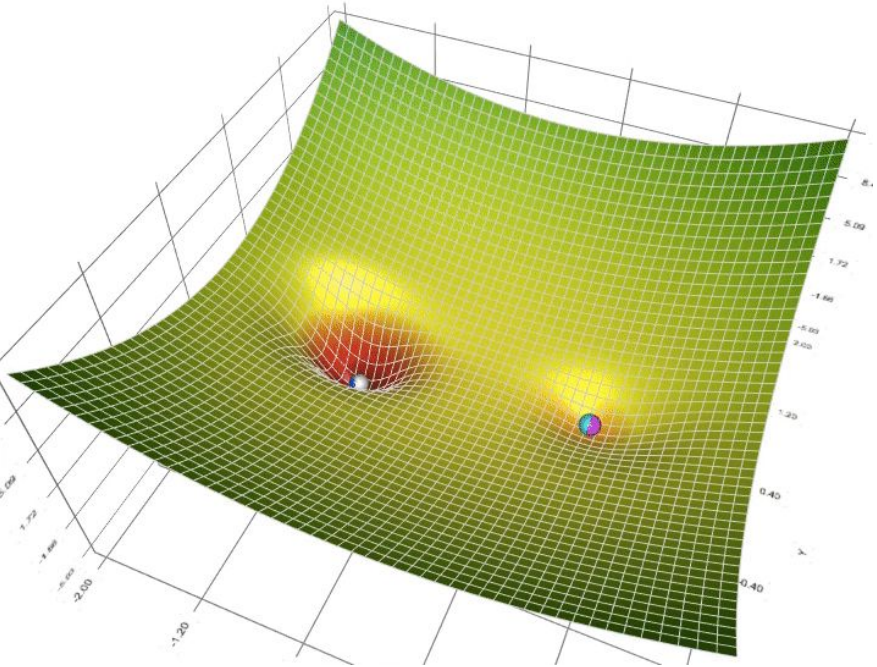


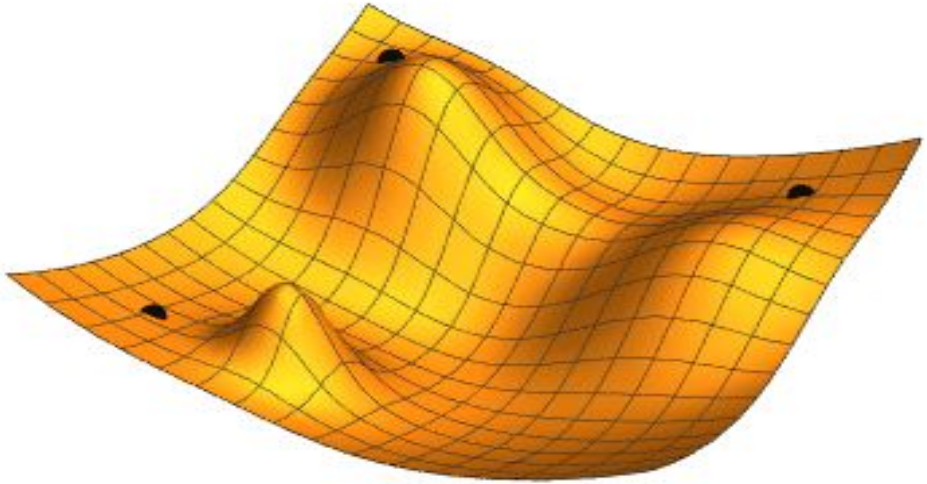
Too big of a step size



Gradient Descent







Depending on initial point, GD might converge to different optima.

Flavors of Gradient Descent

Gradient descent:

$$\theta_{t+1} = \theta_t + \alpha \frac{\delta \mathscr{L}}{\delta \theta_t}$$

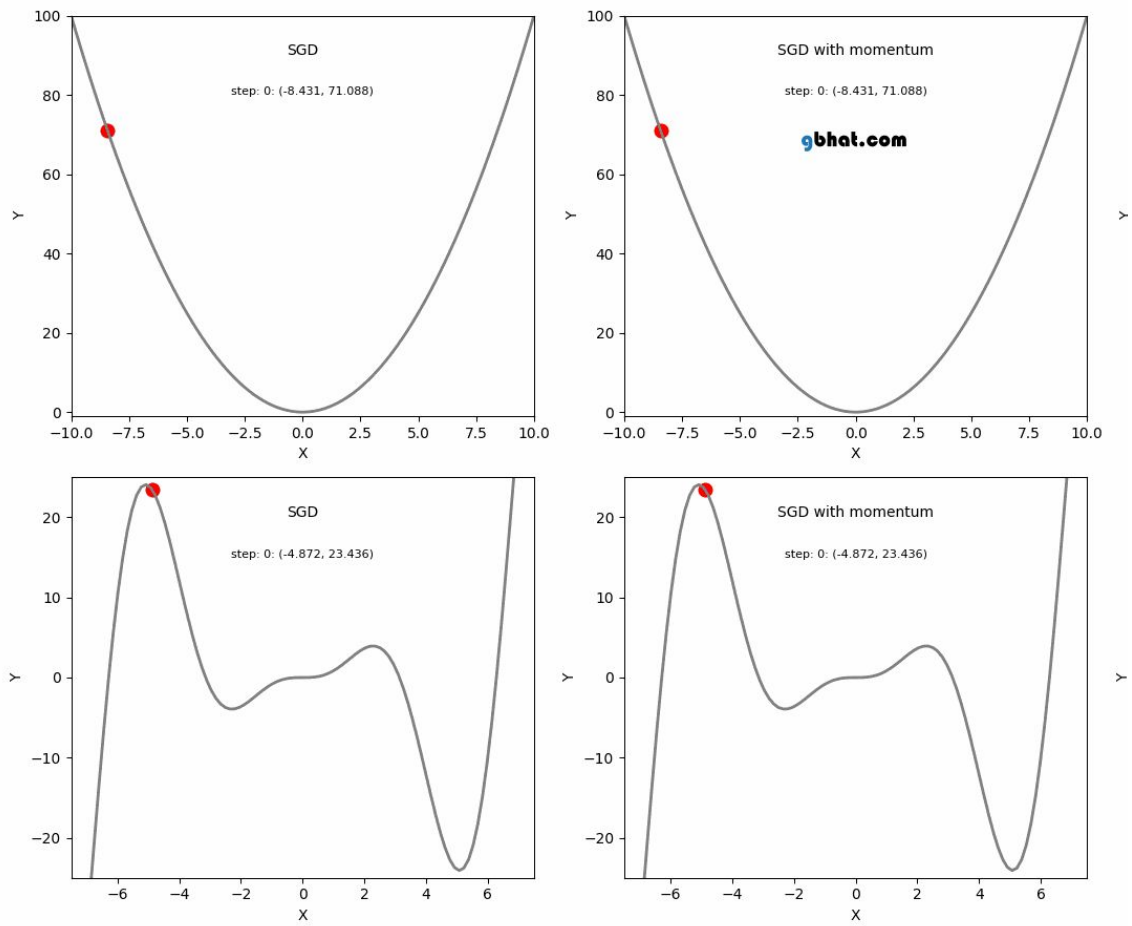
Gradient descent with momentum:

$$z_{t+1} = \beta z_t + \frac{\delta \mathscr{L}}{\delta \theta_t}$$

$$\theta_{t+1} = \theta_t + \alpha z_{t+1}$$

Stochastic gradient descent:

$$\theta_{t+1} = \theta_t + \alpha \mathbb{E}_x \left[\frac{\delta \mathscr{L}}{\delta \theta_t} \right]$$


SGD
step: 0: (-8.431, 71.088)
SGD with momentum
step: 0: (-8.431, 71.088)
gbhat.com
SGD
step: 0: (-4.872, 23.436)
SGD with momentum
step: 0: (-4.872, 23.436)
X
Y

Regularization

x_i Input (image)

y_i Target (labels)

$\mathscr{L}$ Loss Function

θ Parameters

$f(x_i; \theta)$ Prediction

$\mathscr{R}$ Regularization

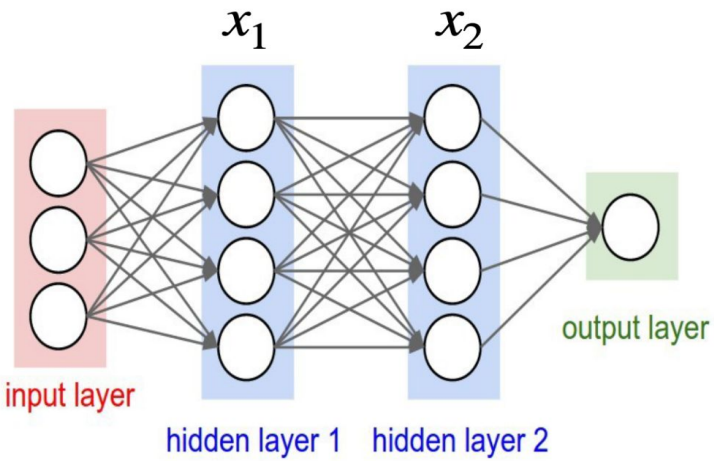
The objective of learning:

$$\min_{\theta} \sum_{i} \mathscr{L}\left(f(x_i), y_i\right) + \lambda R(\theta)$$

Common regularization:

$$R(\theta) = \|\theta\|_2^2$$

How do we compute gradients for complex functions?



$$x_i \in \mathbb{R}^{H \times 1}$$

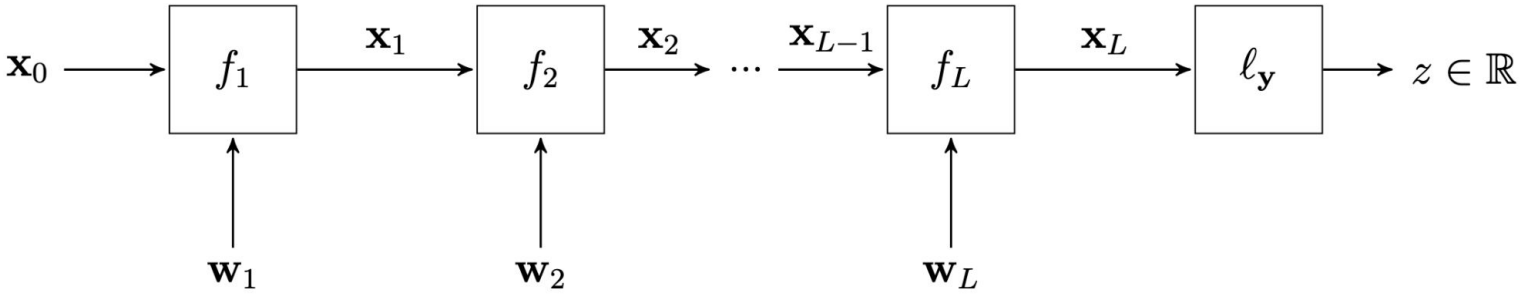
$$W_i \in \mathbb{R}^{D \times H}$$

$$b_i \in \mathbb{R}^{D \times 1}$$

$$x_{i+1} = f\left(W_i x_i + b_i\right)$$

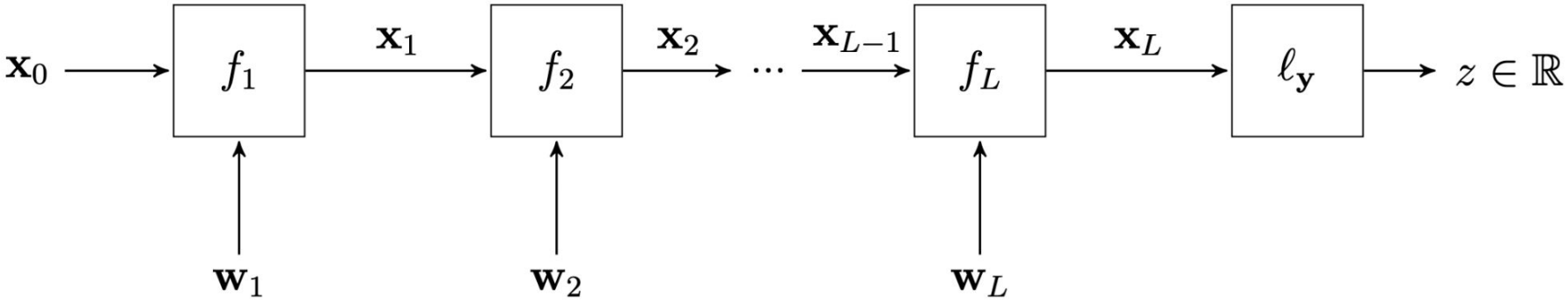
$$\min_{\theta} \sum_{j} \mathscr{L}\left(x_{last}^{j}, y^{j}\right)$$

How do we compute gradients for complex functions?



- Let $\mathbf{x_L}$ be the output of the L^{th} layer.
- We then apply a loss given a target label **y**
- Ultimate goal: compute **dz/dw**

Represent complex functions as composition of simple functions

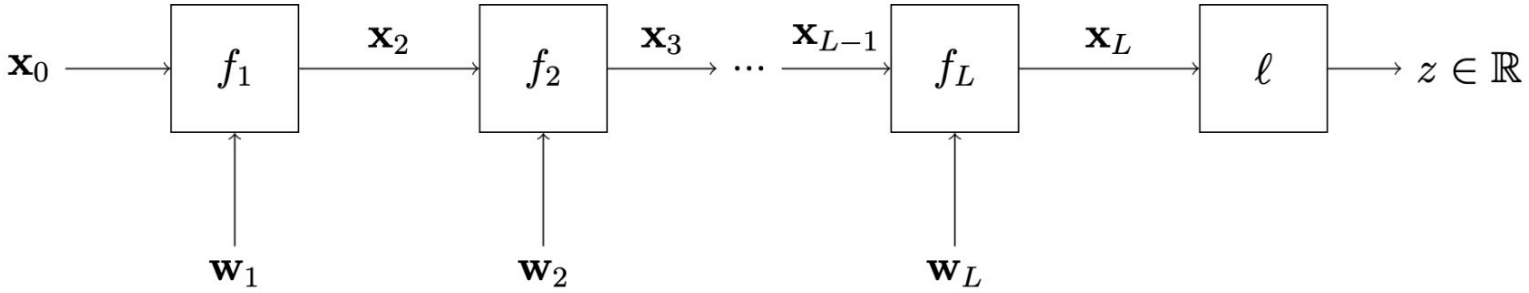


$$g(\mathbf{x}_0, \mathbf{w}_1, \ldots) = f_L(f_{L-1}(f_{L-2}(\ldots, \mathbf{w}_{L-2}), \mathbf{w}_{L-1}), \mathbf{w}_L)$$
$$= f_L(\cdot, \mathbf{w}_L) \circ f_{L-1}(\cdot, \mathbf{w}_{L-1}) \ldots$$

Chain rule!

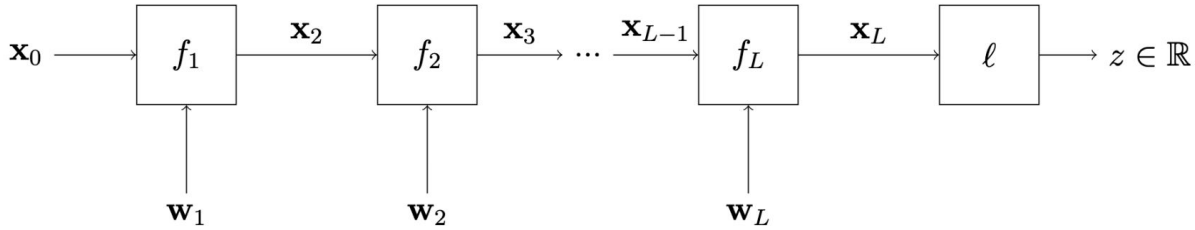
$$\frac{\delta}{\delta x}\left[f \circ g(x)\right] = f'\left(g(x)\right) g'(x)$$

Backpropagation



$$\frac{dz}{d\mathbf{w}_l} = \frac{d}{d\mathbf{w}_l}\left[\ell_{\mathbf{y}} \circ f_L(\cdot;\mathbf{w}_L) \circ \ldots \circ f_2(\cdot;\mathbf{w}_2) \circ f_1(\mathbf{x}_0;\mathbf{w}_1)\right]$$

Backpropagation



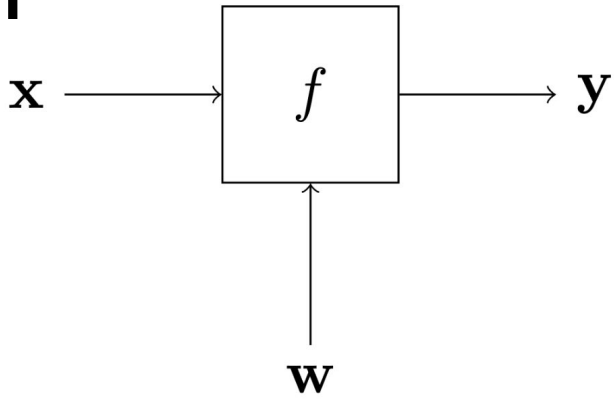
$$\frac{dz}{d\mathbf{w}_l} = \frac{d}{d\mathbf{w}_l}\left[\ell_{\mathbf{y}} \circ f_L(\cdot;\mathbf{w}_L) \circ \ldots \circ f_2(\cdot;\mathbf{w}_2) \circ f_1(\mathbf{x}_0;\mathbf{w}_1)\right]$$

$$\frac{dz}{d\mathbf{w}_l} = \frac{dz}{d(\operatorname{vec}\mathbf{x}_L)^\top}\frac{d\operatorname{vec}\mathbf{x}_L}{d(\operatorname{vec}\mathbf{x}_{L-1})^\top}\cdots\frac{d\operatorname{vec}\mathbf{x}_{l+1}}{d(\operatorname{vec}\mathbf{x}_l)^\top}\frac{d\operatorname{vec}\mathbf{x}_l}{d\mathbf{w}_l^\top}$$

We can add *any* functional module f so long as its interface provides:
(1) derivative of output wrt to input
(2) derivative of output wrt parameter

Slide credit: Deva Ramanan

Backpropagation

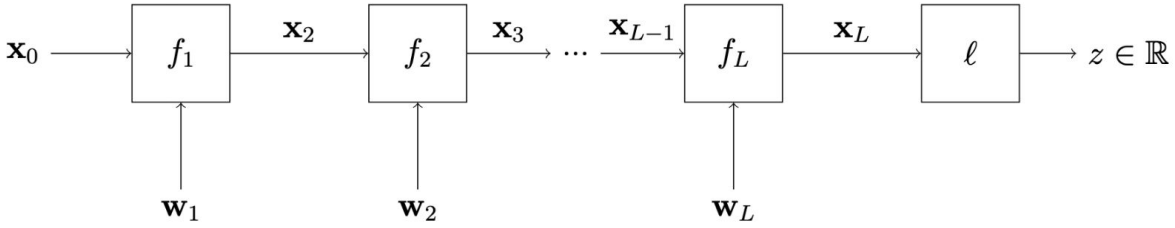


$\frac{\partial y}{\partial w}$: needed to compute gradiant updates for this block

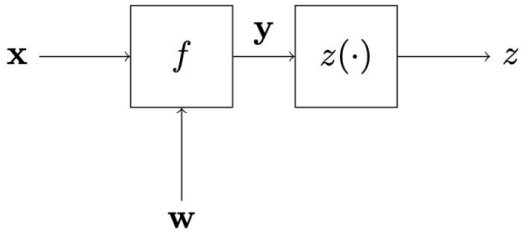
$\frac{\partial y}{\partial x}$: needed to compute gradiant updates next block down stream

Slide credit: Deva Ramanan

Computational efficiency - cache intermediate gradients



$$\underbrace{\ell \circ f_L(\cdot, \mathbf{w}_L) \circ f_{L-1}(\cdot, \mathbf{w}_{L-1}) \cdots \circ f_{l+1}(\cdot, \mathbf{w}_{l+1})}_{z(\cdot)} \circ f_l(\mathbf{x}_l, \mathbf{w}_l) \circ \dots$$



$$\frac{dz}{d\mathbf{x}} = \frac{dz}{d\mathbf{y}} \frac{d\mathbf{y}}{d\mathbf{x}}$$

So far we have…

Data - e.g. images of cats and dogs

A function to fit with free variables- e.g. multiple layer neural network with weights, and biases

Output to achieve - e.g. cat images → text label cat, dog images → text label dog

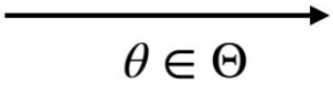
A loss function - if cat image → text label dog, what is the penalty for mistakes?

Optimization routine - fitting function weights through gradient descent (or grid/random search).

ML = Data + (Model + Loss + Optimization)

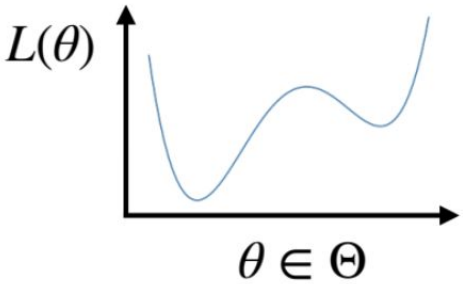
1. Model space f_θ

What is the search space



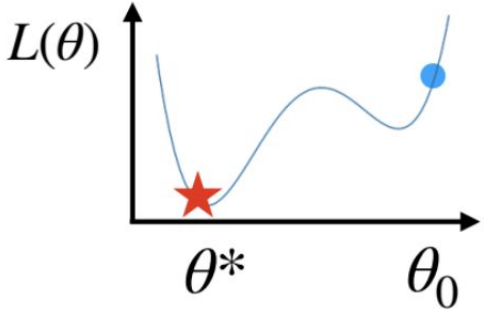
2. Loss function L

What is the search criteria



3. Find θ^* that minimize L

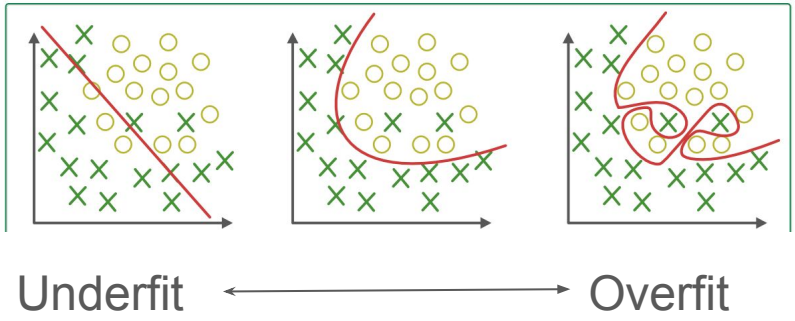
How to search

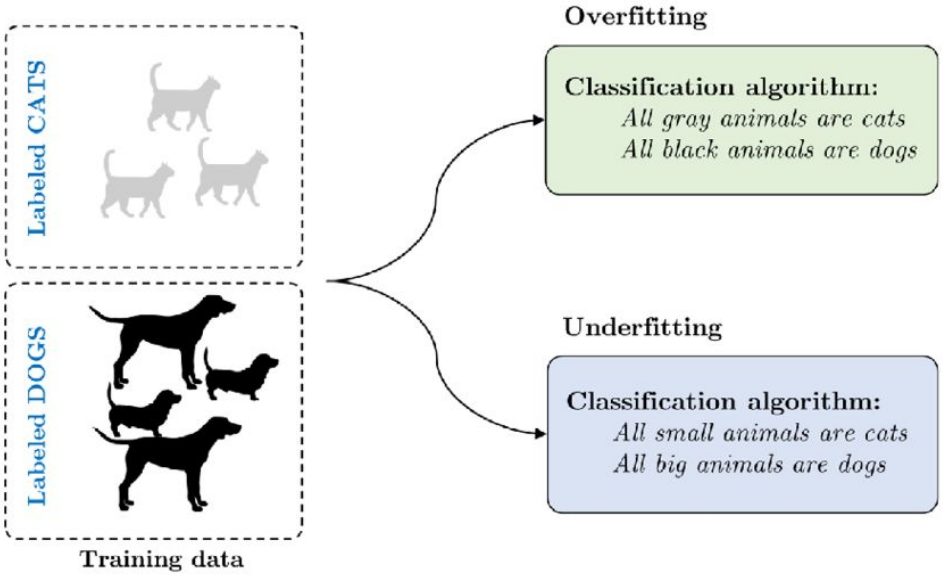


With a complex enough function, we can fit anything? How do we know we are right?

Learning → Generalization

Overfitting vs. underfitting





Example

Let's fit a polynomial: given x, predict y

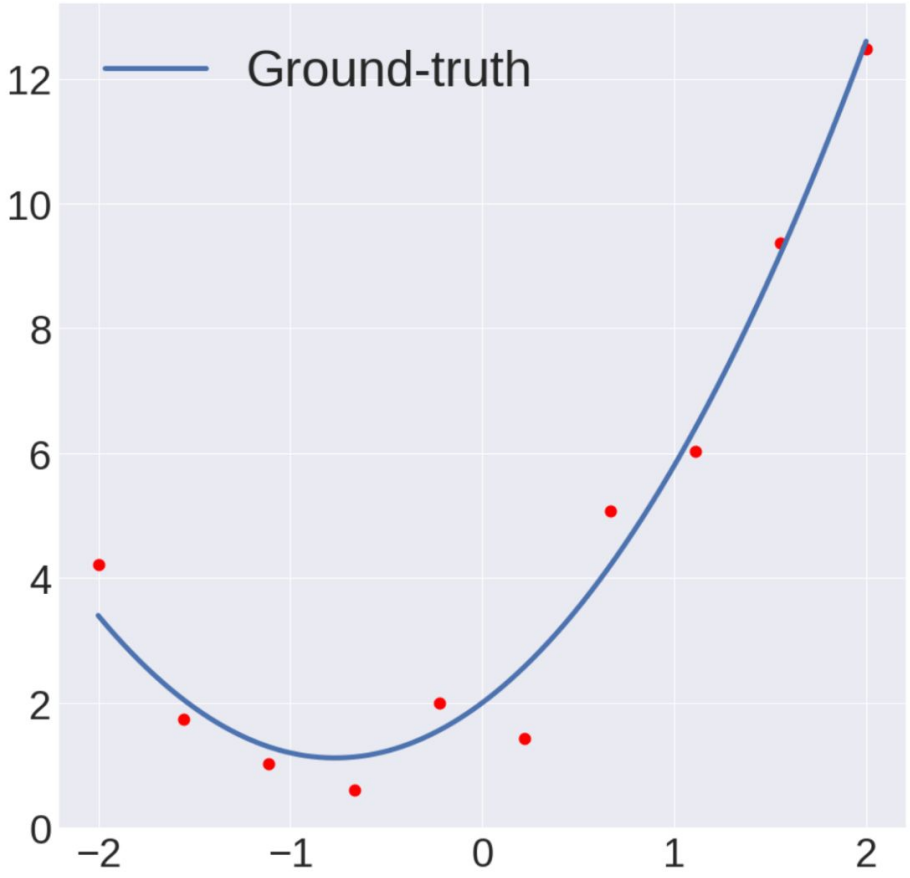
Note: can do non-linear regression with copies of x

$$\begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} x_1^F & \cdots & x_1^2 & x_1 & 1 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ x_N^F & \cdots & x_N^2 & x_N & 1 \end{bmatrix} \begin{bmatrix} w_F \\ \vdots \\ w_2 \\ w_1 \\ w_0 \end{bmatrix}$$

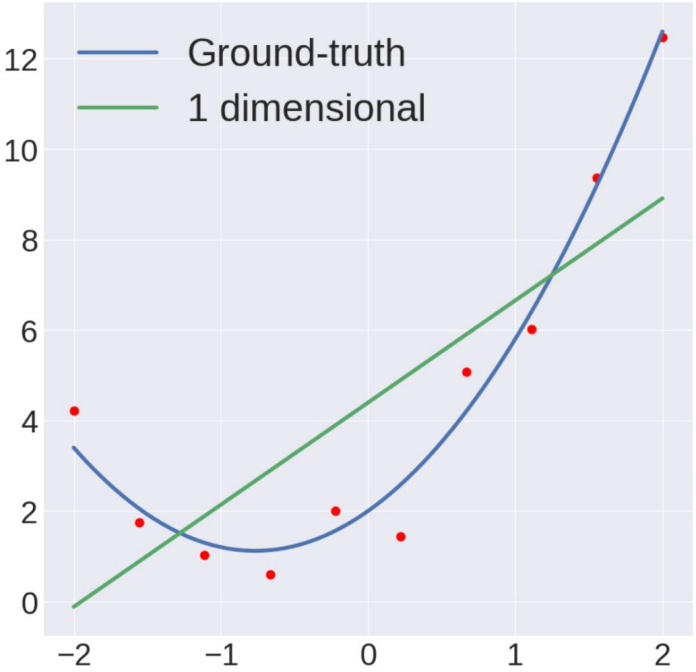
Matrix of all polynomial degrees

Weights: one per polynomial degree

Model: $1.5x^2 + 2.3x+2 + N(0,0.5)$



Underfitting

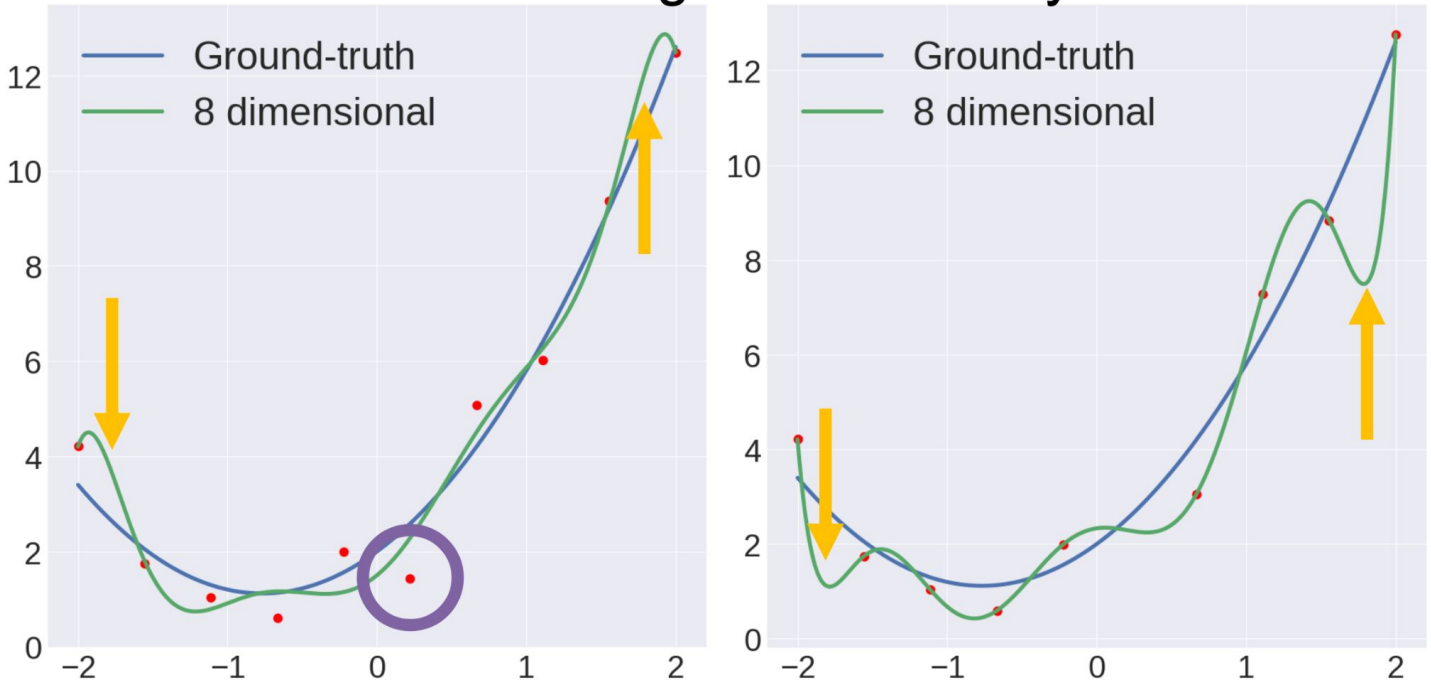


Model doesn't have the parameters to fit the data.

Bias (statistics): Error intrinsic to the model.

Overfitting

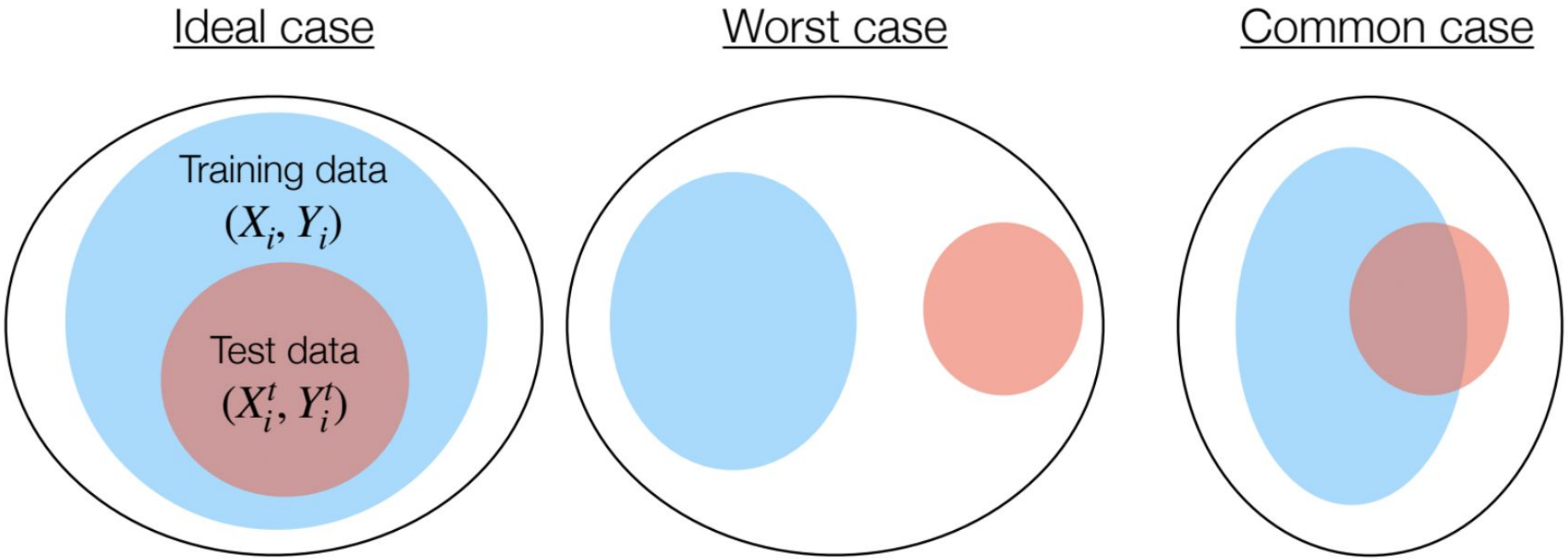
Model has high ***variance***: remove **one point**, and model changes dramatically



How can we select the “right” model?

Cross-validation

We can split our data to approximate generalization



Experimental design

Idea #1: Choose model that work best on the **training data**

BAD: memorization works perfectly on training data

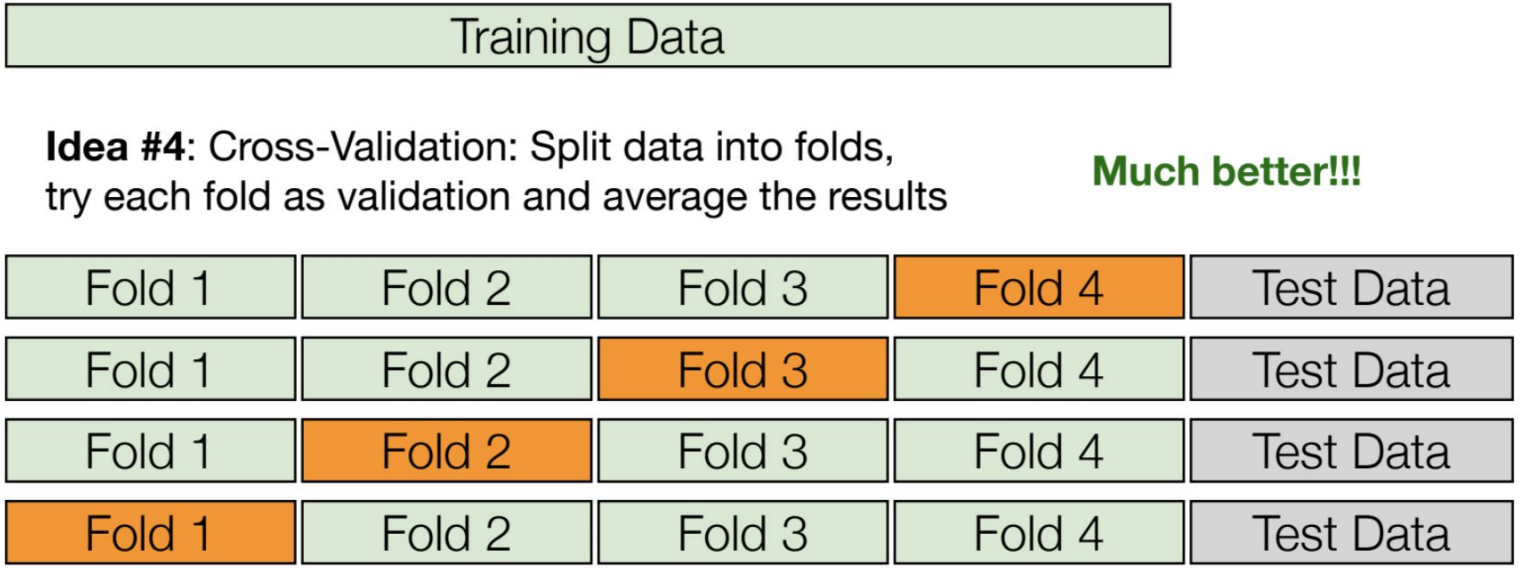
Idea #2: Choose model that work best on the **test data**

BAD: No idea how algorithm will perform on new data

Idea #3: Split it into train and val; choose Model on val and evaluate on test

Good!

How to avoid training dataset bias?

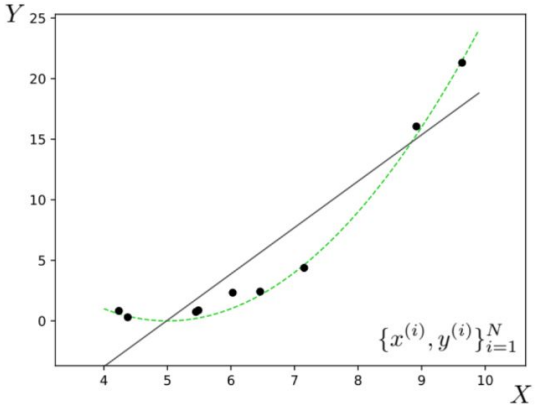


(Mostly used for small datasets)

Model selection criteria - choose model that minimizes error on your validation set

Underfitting

K = 1

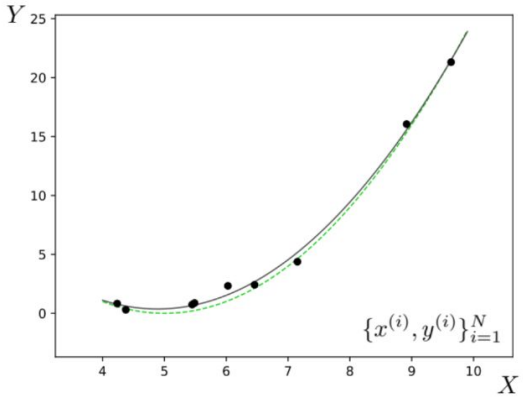


High error on train set

High error on test set

Appropriate model

K = 2

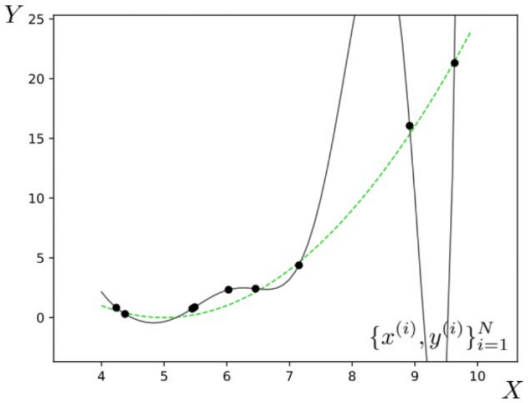


Low error on train set

Low error on test set

Overfitting

K = 10



Lowest error on train set

High error on test set

In summary, ML systems consist of…

Data - e.g. images of cats and dogs

A function to fit with free variables- e.g. multiple layer neural network with weights, and biases

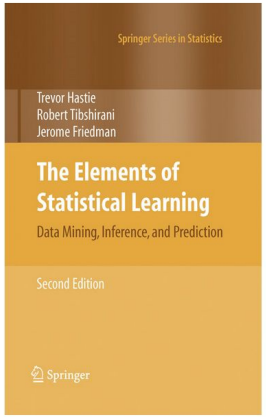
Output to achieve - e.g. cat images → text label cat, dog images → text label dog

A loss function - if cat image → text label dog, what is the penalty for mistakes?

Optimization routine - fitting function weights through gradient descent (or grid/random search).

Cross-validation paradigm - measure how well the system generalizes

ML is obviously too vast to cover in detail here, great (free) ML textbook to refer to



The Elements of Statistical Learning:

Data Mining, Inference, and Prediction.

Second Edition

February 2009

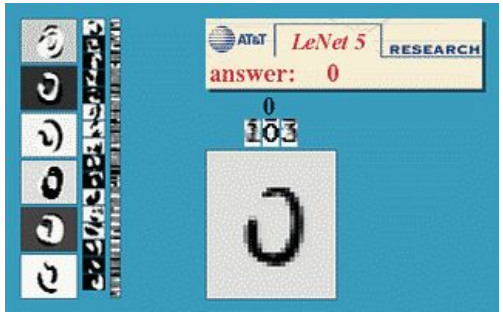
Trevor Hastie

Robert Tibshirani

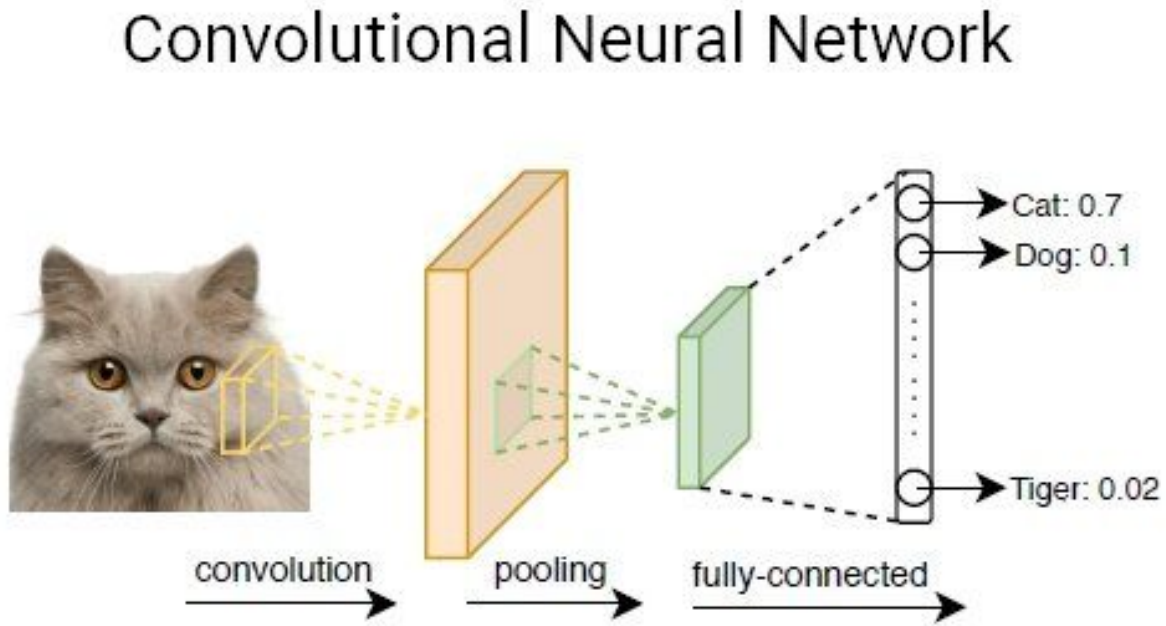
Jerome Friedman

https://hastie.su.domains/ElemStatLearn/

Next lecture: Convolutional neural nets

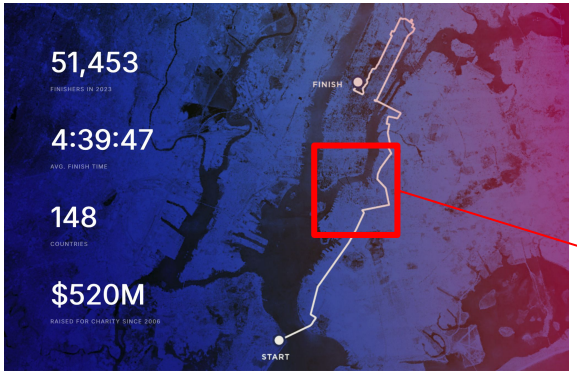


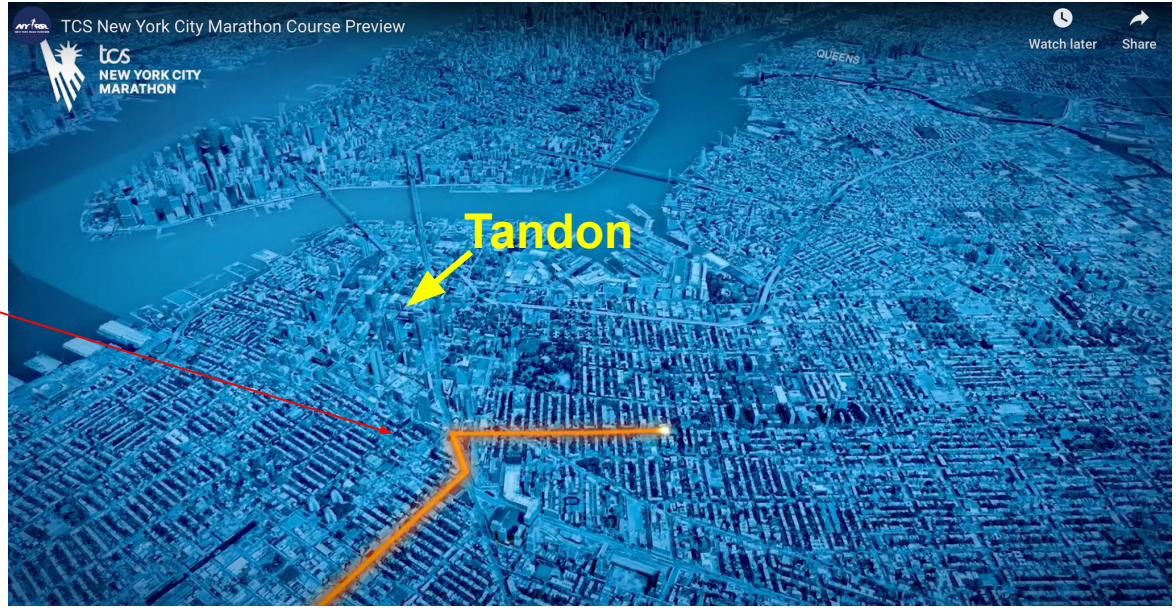
LeCun ‘98



This sunday - NYC Marathon (I'll be there!) - Nov 3

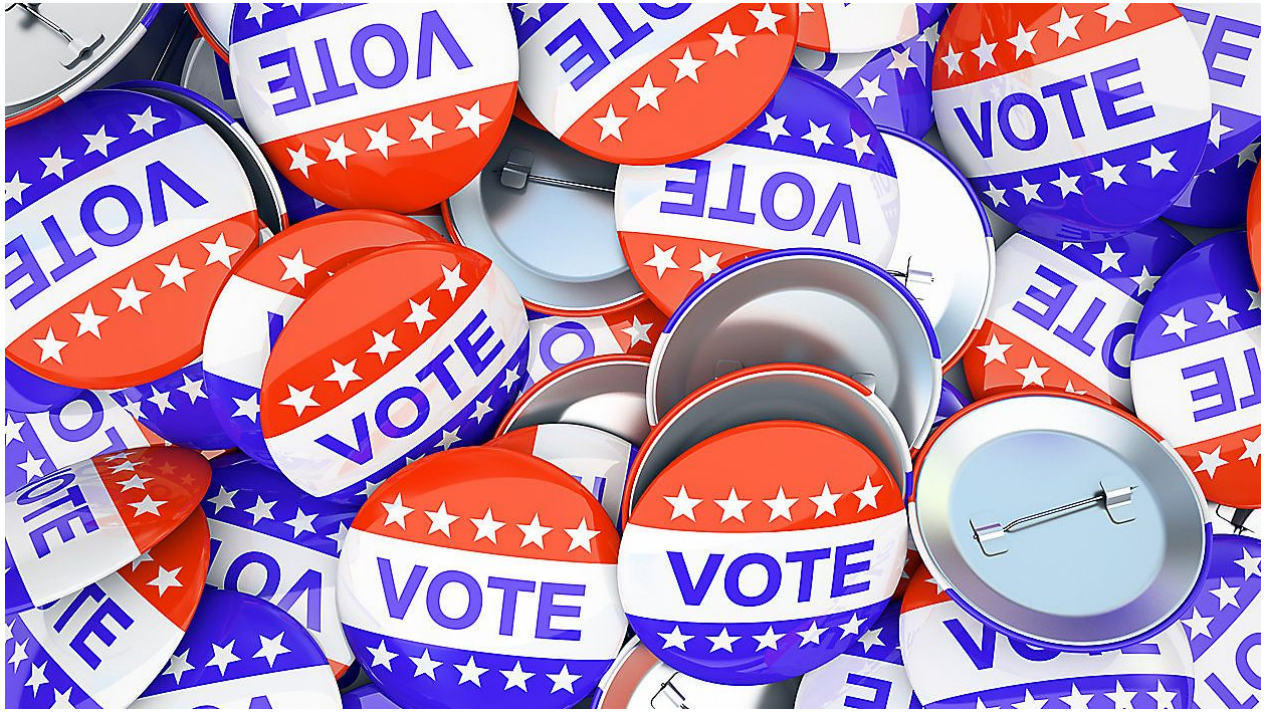
Route:





Where to watch:
https://www.runnersworld.com/news/a62683693/new-york-city-marathon-spectator-where-to-watch/

Next week - election day, Nov. 5!



Coding lab