CS 260D: Large-scale Machine Learning

Baharan Mirzasoleiman

http://web.cs.ucla.edu/~baharan

PTE?? Waitlist??

- Sorry that I cannot give PTEs…
  - Except to a few PhD students that really need this course for their research
- And I cannot admit the waitlist either
  - Because of room capacity and TA limit
- But I’m happy to have you all audit the course!
- If you’re PhD who really needs this course, or you want to audit, scan the QR code.
- ps: Bruinlearn will be up soon

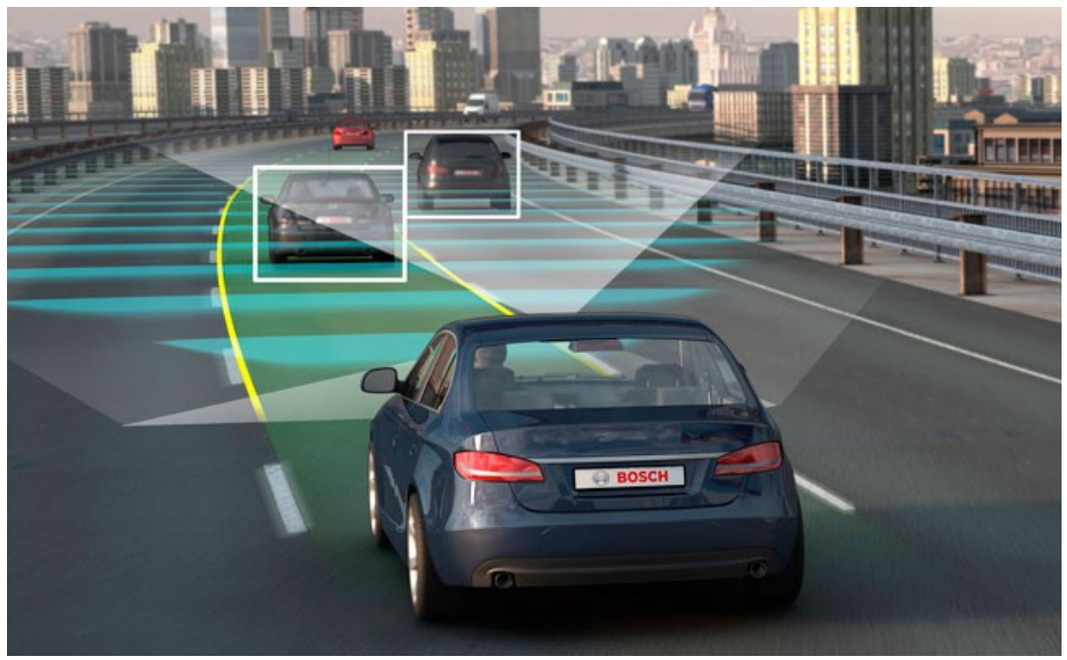
Machine Learning Becomes Mainstream

Personalized medicine

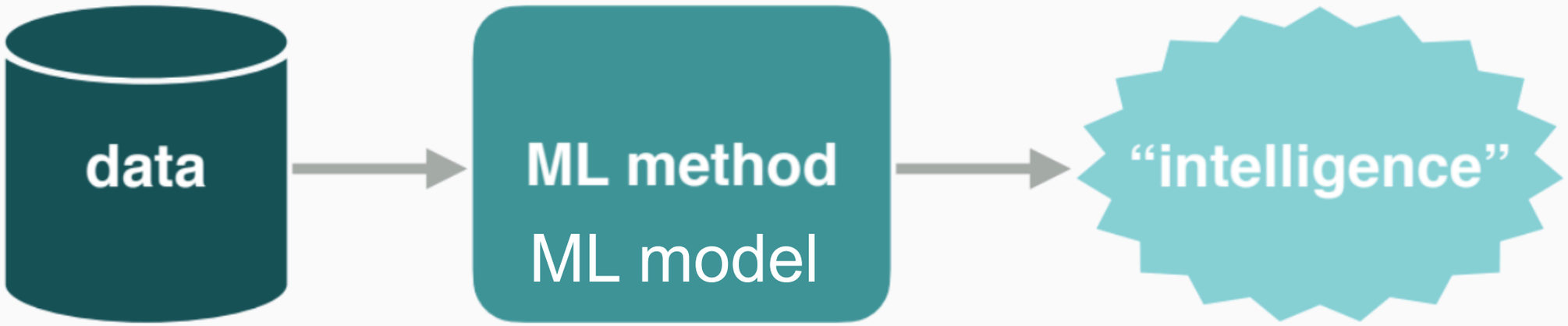
Robotics

Finance

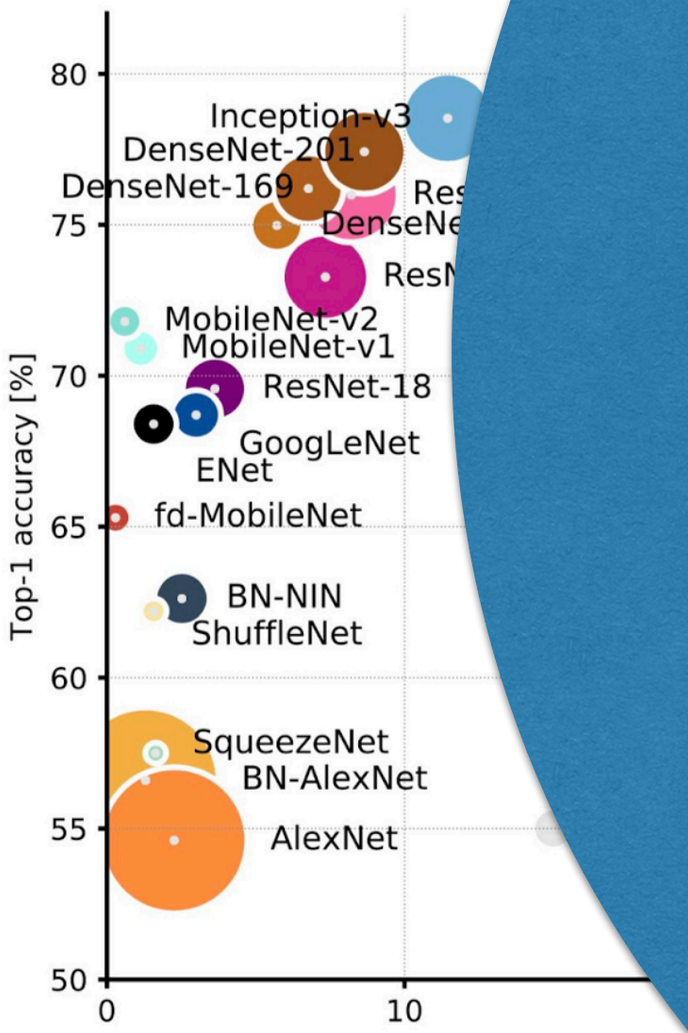
Autonomous cars



How do we learn?



How Big a



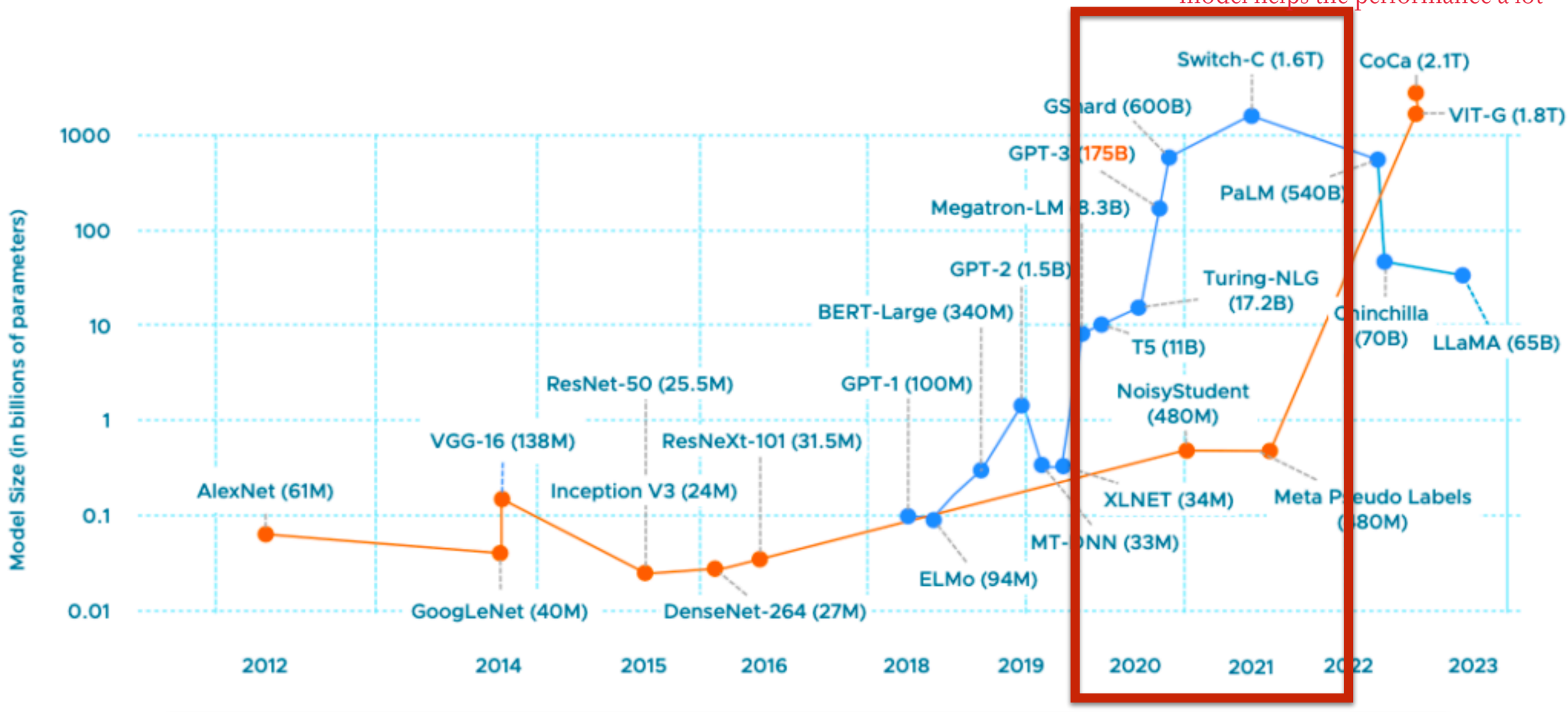
2017

GPT-3 — 2020
175 billion parameters

How big are we now?

How big are we now?

People found that the size of the model helps the performance a lot

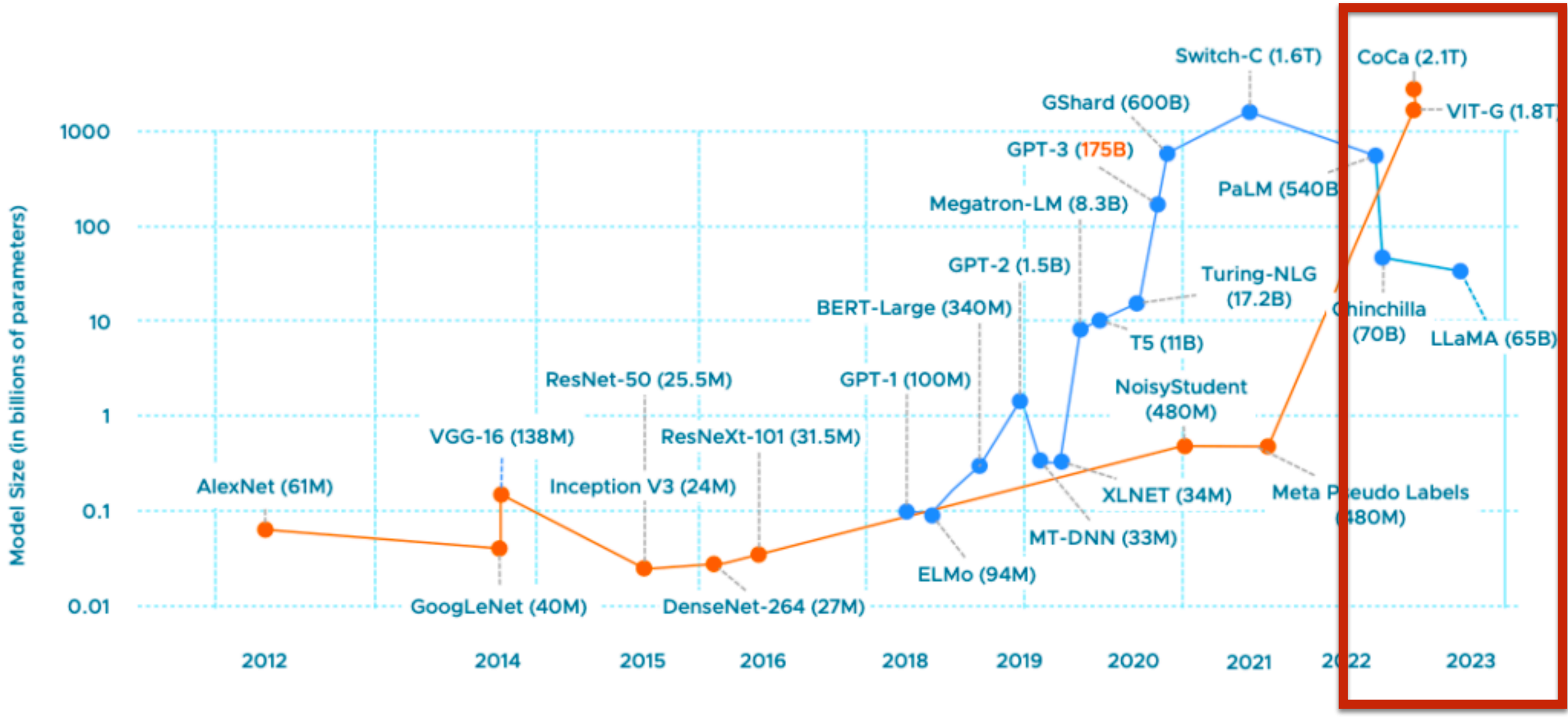


Increasing model size is a proxy for increasing performance (power-law between model size and performance)

[Kaplan et al 2020]

How big are we now?

People realize that the data quantity doesn't catch up with the size of the model



Data is as important as scaling model size!
“For 2x model size, data should also be 2x”

[Hoffmann et al 2022]

Data is the key

小而精效果更好

- A smaller model can vastly outperform a larger — but suboptimal — model if trained on more data

  - Chinchilla, a 70B-parameter model 4 times smaller than the previous leader in language AI, Gopher, but trained on 4 times more data.

  - Chinchilla “uniformly and significantly” outperforms Gopher, GPT-3, Jurassic-1, and Megatron-Turing NLG across a large set of benchmarks.

Big models are significantly undertrained

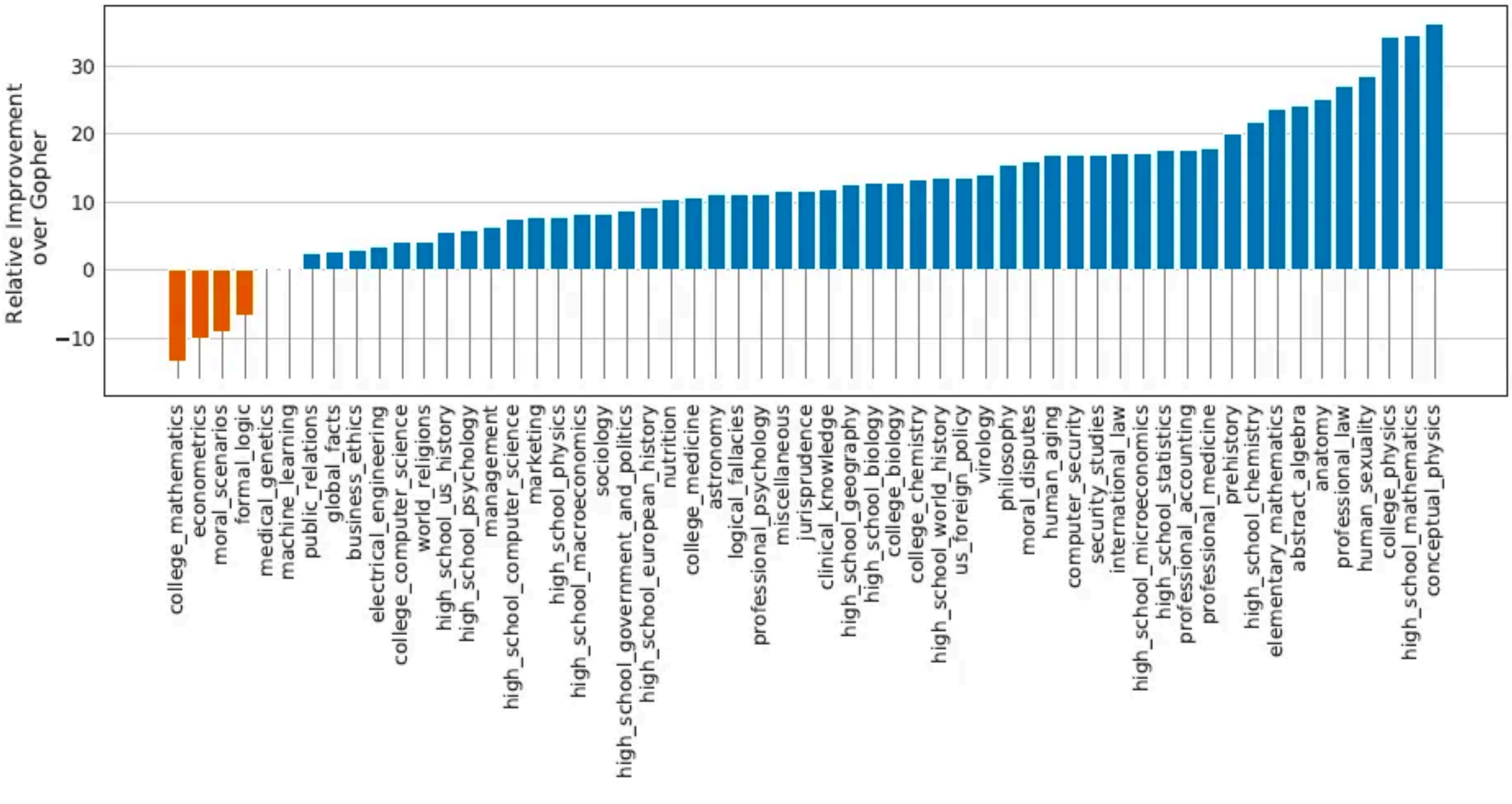
- Kaplan's law: if compute budget increases by a factor of 10, we get optimal performance when model size is increased by 5.5x and data size is increased by 1.8x.

- Chinchilla: both model size and number of tokens should increase in parallel, roughly by 3.16x (or $\sqrt{10}$x).

- Conclusion: a 175B model (GPT-3-like) should be trained with more than 10 times data than what OpenAI used for their GPT-3 175B model). A 280B model (Gopher-like) should be trained with 20 times what DeepMind used for Gopher.

Chinchilla significantly outperforms Gopher

- Chinchilla is 4x smaller and 4x more data than Gopher.



Data requirement is too big to deploy models safely!

How do you understand safely here?

1. Without data leaking
    1.1 The data might be private, so uploading big data to cloud can cause leak problem
2. Without breakdown
    2.1 Deploying a lot of data might cause the system to break down

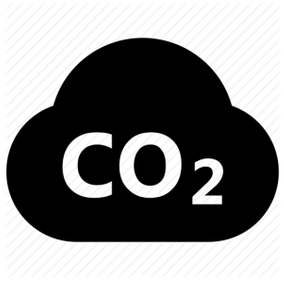
Problem 1: Training Large Models is Expensive

Example: training a **single** deep model GPT-3 on **45TB** of data

12M

34 days on **1024** A100s
335 years on 8 A100s

17.8X the yearly **energy** consumption
of the average American

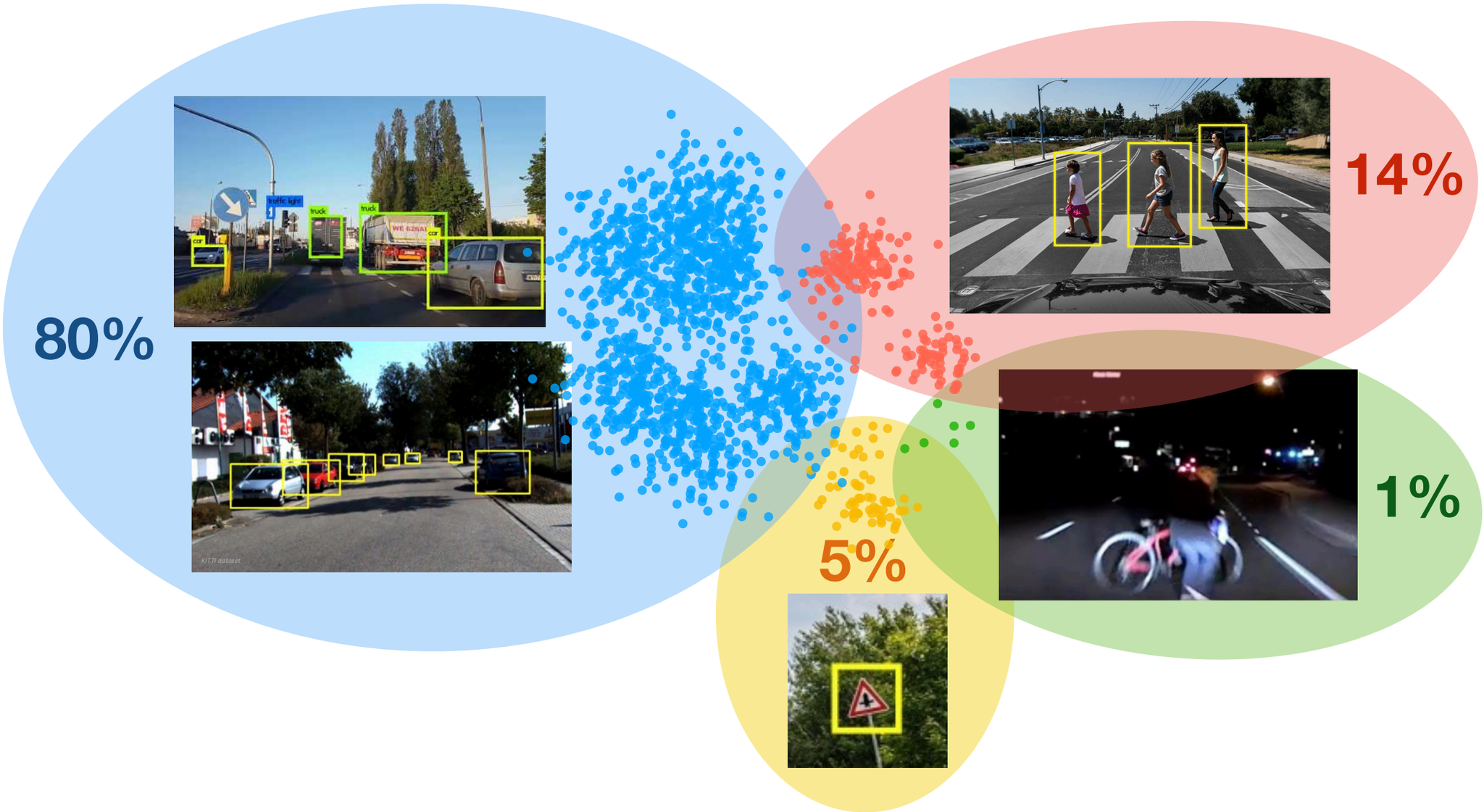


CO2: a car driving 2x the distance between Earth and Moon!!

Problem 2: Real-world Datasets are Biased

- Model performs poorly on minorities (Fairness)
- Spurious biases impede out-of-distribution (OOD) performance

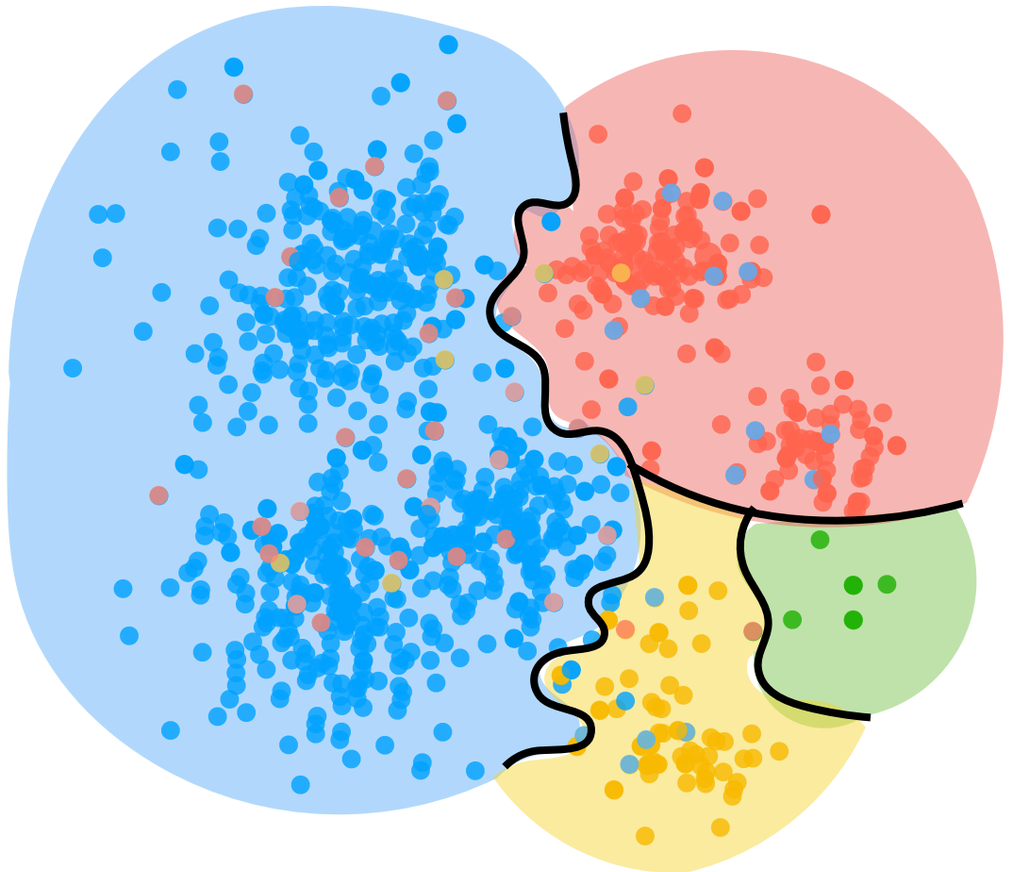
Example: self driving data



Problem 3: Real Data is Unlabeled

- We label the data automatically

  Example: crowed-sourcing, automated labeling, …

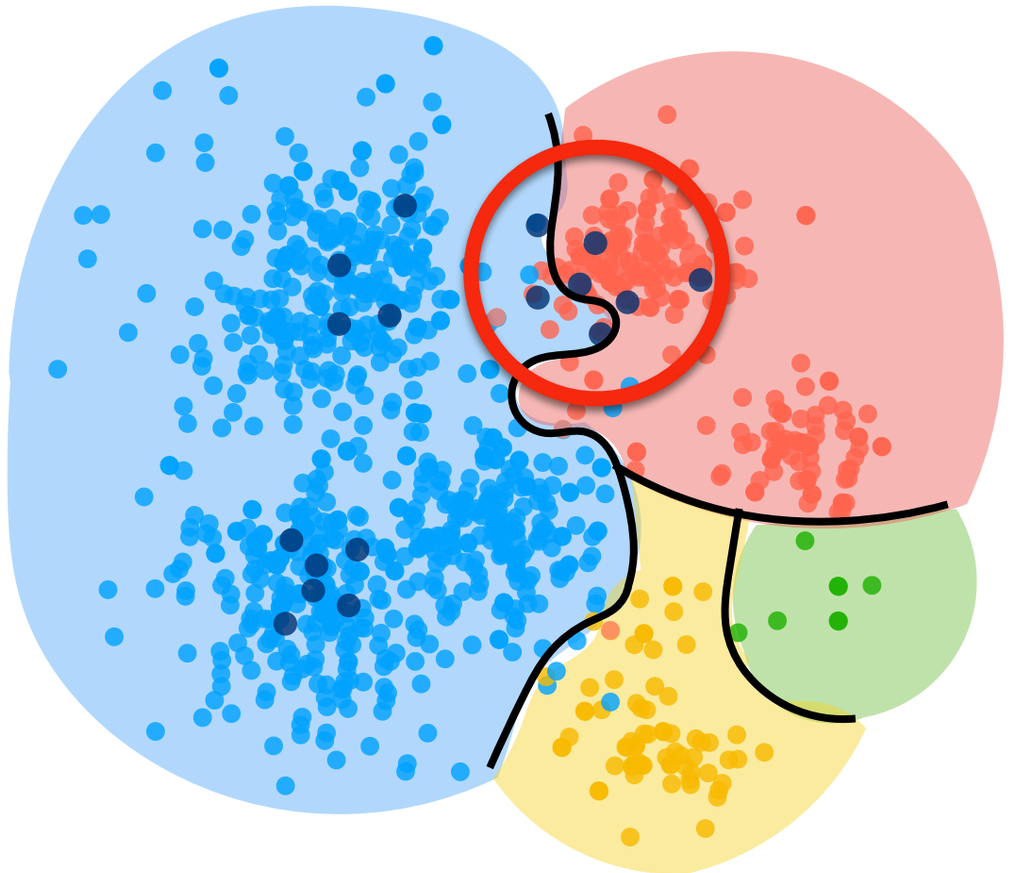


Over-parameterized models overfit and memorize the mislabeled data

Problem 4: Examples May be Corrupted

- Many large datasets are collected from Internet or users

  Example: large Image and NLP datasets, …



Adversarial attacks change the prediction of a test-time target example and cannot be visually identified

This course is about:

How can we train big models on big data **more efficiently** and **robustly**?

From an algorithmic point of view!

- **Who should take this course?**
  - If this is really relevant to your research/career and you're up for doing some hard work!

Grading

- 3 Assignments: 10% each
  - Both written/theory and programming questions
- Midterm exam (in the class): 40%
- Final project (in groups of 3): 30%
  - Project presentation: 10%
  - Project report: 20%

Projects

- You should choose a topic **related to the materials discussed in the course**.
  - **[Ideal]** a novel and sound solution to an interesting problem discussed in the course that is publishable in machine learning conferences.
  - a theoretical analysis of methods we've studied, or
  - an application of the methods you've learned in this course.
- You should clarify the contribution of each team member in your report

Discussions

- Solving homework questions
  - Important for the midterm!
- We'll collect and answer your questions in the discussion sessions
- The TA will not help with the programming questions and setting up the frameworks!

Schedule

If this is not a review for you, please consider not taking the course!

week 1	Oct 2	[**Review**] momentum, adaptive lr	
	Oct 4	[**Review**] variance reduction	
week 2	Oct9	[distributed opt.] distributed SGD, Hogwild	
	Oct 11	[distributed opt.] MapReduce/federated learning	HW1
week 3	Oct 16	[distributed opt.] federated learning	
	Oct 18	[data compression] data selection: submodularity	**HW1 due**
week 4	Oct 23	[data compression] distributed/streaming submod. alg.	
	Oct 25	[data compression] data selection for supervised learning	HW2
week 5	Oct 30	[data compression] data selection for supervised learning	
	Nov 1	[data compression] data selection for supervised learning	**HW2 due**
week 6	Nov 6	[data compression] data selection for contrastive learning	
	Nov 8	[data compression] data selection for contrastive learning	HW3
week 7	Nov 13	[data compression] contrastive multimodal learning	
	Nov 15	[model compression] NN pruning	**HW3 due**
week 8	Nov 20	[model compression] NN pruning	
	Nov 22	**Midterm**	
week 9	Nov 27	[model compression] Neural Architecture Search	
	Nov 29	[model compression] NN quantization	
week 10	Dec 4	Final project presentations	
	Dec 6	Final project presentations	

Note: Schedule may change a bit depending on how things go

Review: Optimization

Quick Recap: Training a Model M

Total loss

Average over all training instances

Divergence between desired output and actual output of M for a given input X

Output of M in response to input X

Desired output in response to input X

$$L(W) = \frac{1}{N_X} \sum_X div(f(X; W), D(X))$$

$$\hat{W} = \arg\min_W L(W)$$

- Define a total "loss" over all training instances
  - Quantifies the difference between desired output and the actual output, as a function of weights
- Find the weights that minimize the loss

Quick Recap: Training by Gradient Descent

$$L(W) = \frac{1}{N_X}\sum_X div(f(X;W), D(X))$$

$$\nabla_W L(W) = \frac{1}{N_X}\sum_X \nabla_W div(f(X;W), D(X))$$

Del算子或称**Nabla算子**，在中文中也叫**向量微分算子**、**劈形算子**、**倒三角算子**，符号为**∇**，是一个向量微分算子，但本身并非一个向量[1]。

其形式化定义为：

$\nabla = \frac{\mathrm{d}}{\mathrm{d}r}$

在n维空间中，分母$\mathrm{d}r$为含n个分量的向量，因而∇本身就是个n维向量算子。

三维情况下，$\nabla = \frac{\partial}{\partial x}\mathbf{i} + \frac{\partial}{\partial y}\mathbf{j} + \frac{\partial}{\partial z}\mathbf{k}$ 或 $\nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right)$

二维情况下，$\nabla = \frac{\partial}{\partial x}\mathbf{i} + \frac{\partial}{\partial y}\mathbf{j}$ 或 $\nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right)$

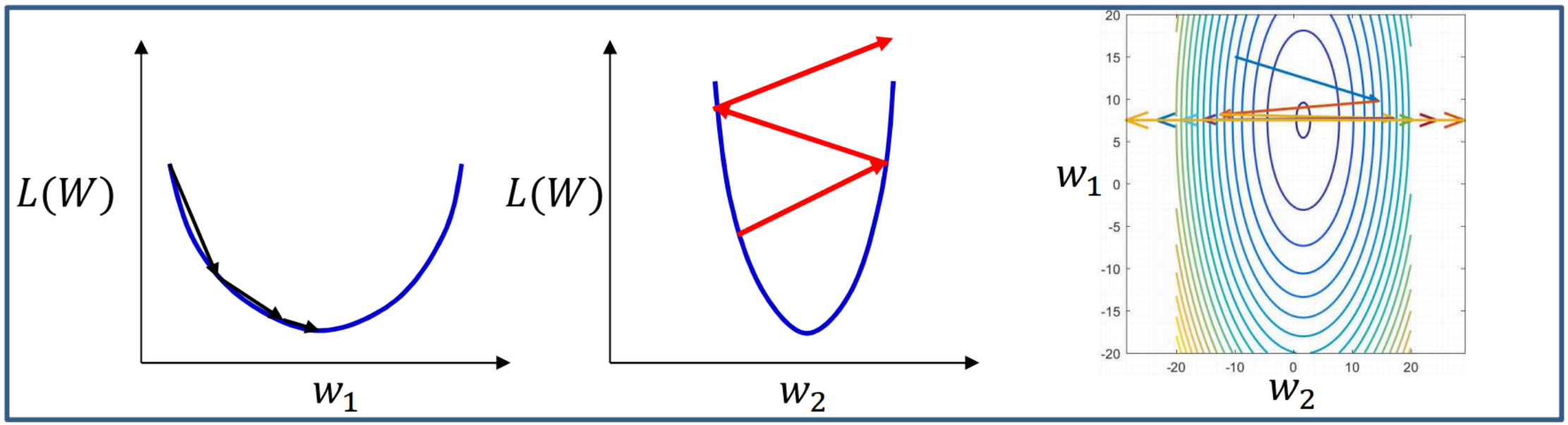
倒三角是梯度算子，也叫del或nabla算子
https://zh.wikipedia.org/zh-hans/Nabla%E7%AE%97%E5%AD%90

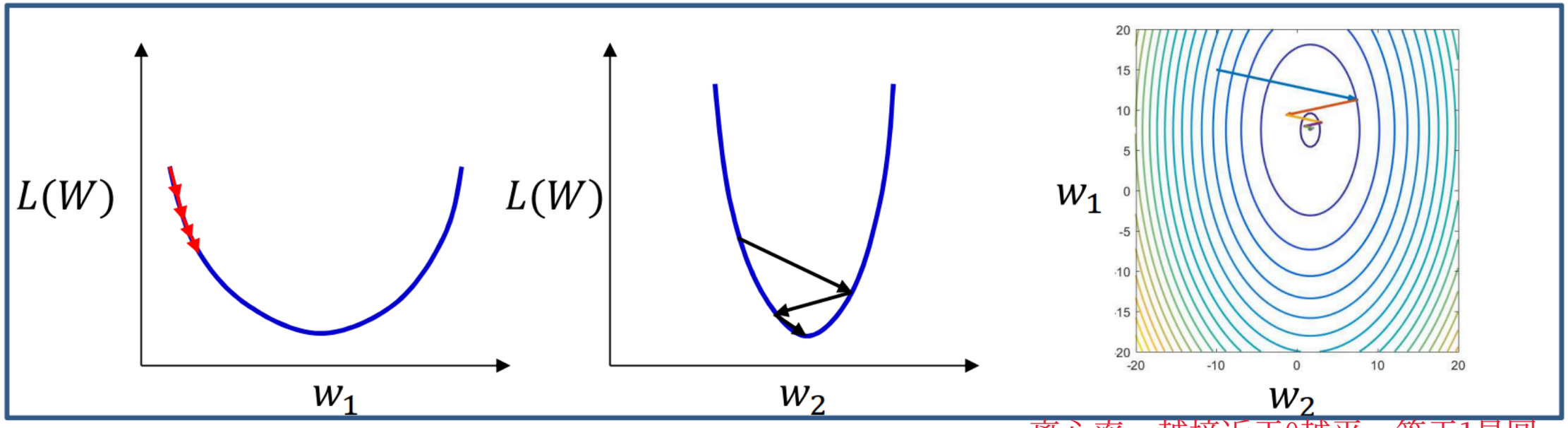
Solved through gradient descent as

$$\widehat{W} = \arg\min_W L(W) \quad \Rightarrow \quad W_k = W_{k-1} - \eta \nabla_w L(W)^T$$

- The gradient of the total loss is the average of the gradients of the loss for the individual instances
- The total gradient can be plugged into gradient descent update to learn the network

Problem with Gradient Descent



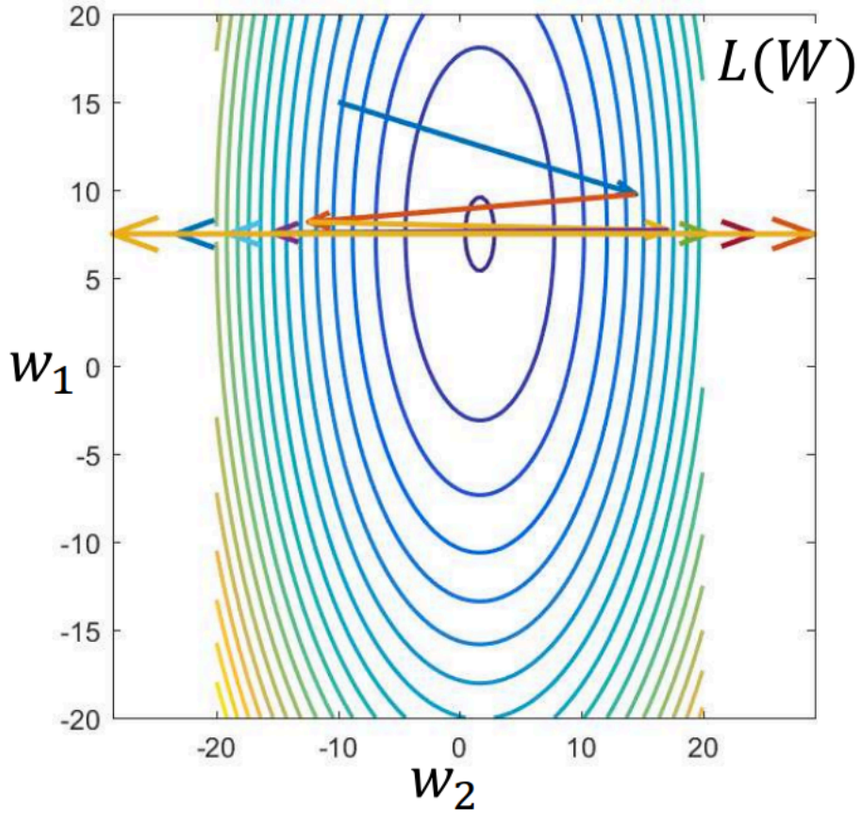


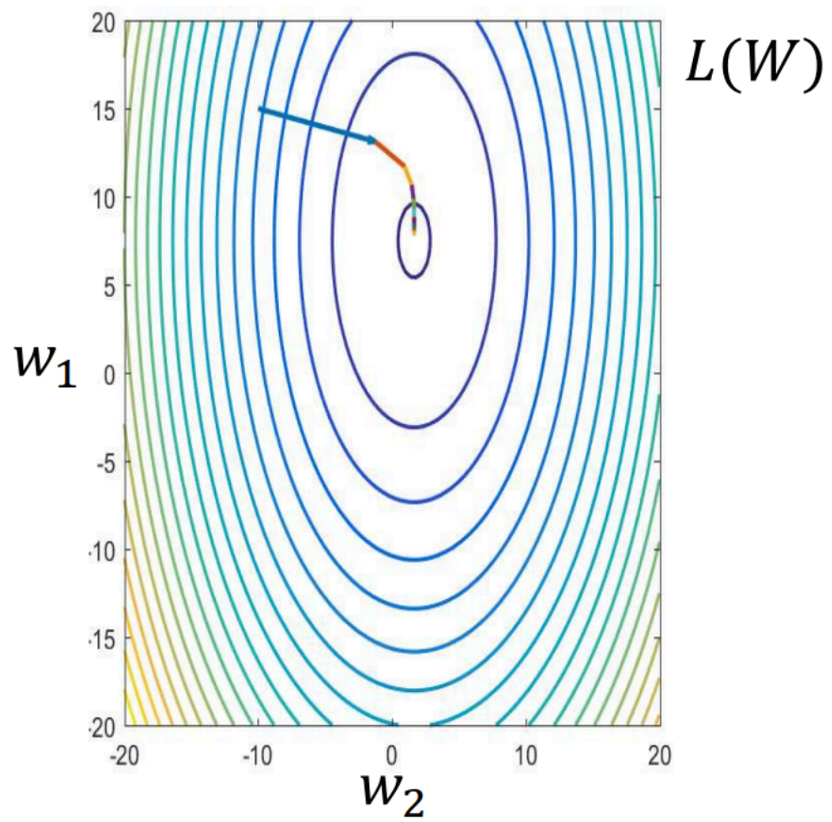
$$W_k = W_{k-1} - \eta \nabla_w L(W)^T$$

离心率，越接近于0越平，等于1是圆，大于1是很扁的椭圆

- A step size that assures fast convergence for a given eccentricity can result in divergence at a higher eccentricity
- .. Or result in extremely slow convergence at lower eccentricity

Problem with Gradient Descent





- The loss is a function of many weights (and biases)
  - Has different eccentricities w.r.t different weights

with respect to

- A fixed step size for all weights in the network can result in the convergence of one weight, while causing a divergence of another

收敛 发散

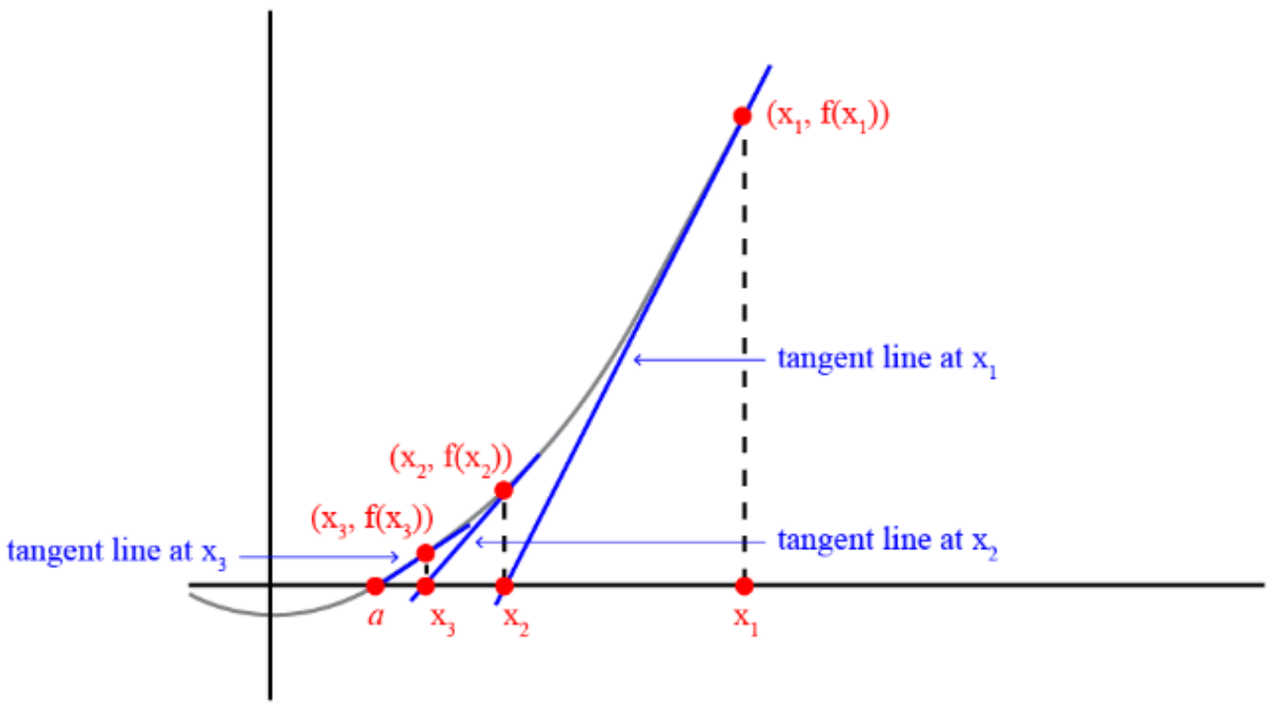
Solutions for problem with gradient descent

- Try to normalize curvature in all directions (曲率)
  - Second order methods, e.g. Newton's method
  - Too expensive: require inversion of a giant Hessian

- Treat each dimension independently:
  - Rprop, quickprop
  - Works, but ignores dependence between dimensions
    - Can result in unexpected behavior
  - Can still be too slow

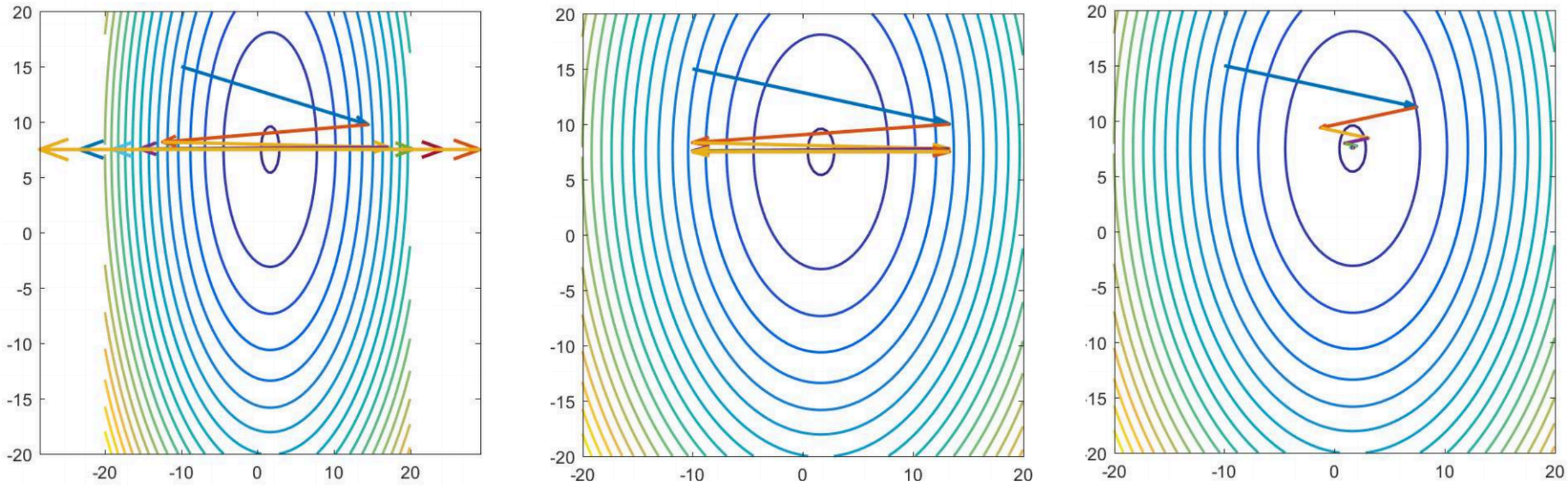
Newton's Method

https://calcworkshop.com/derivatives/newtons-method/



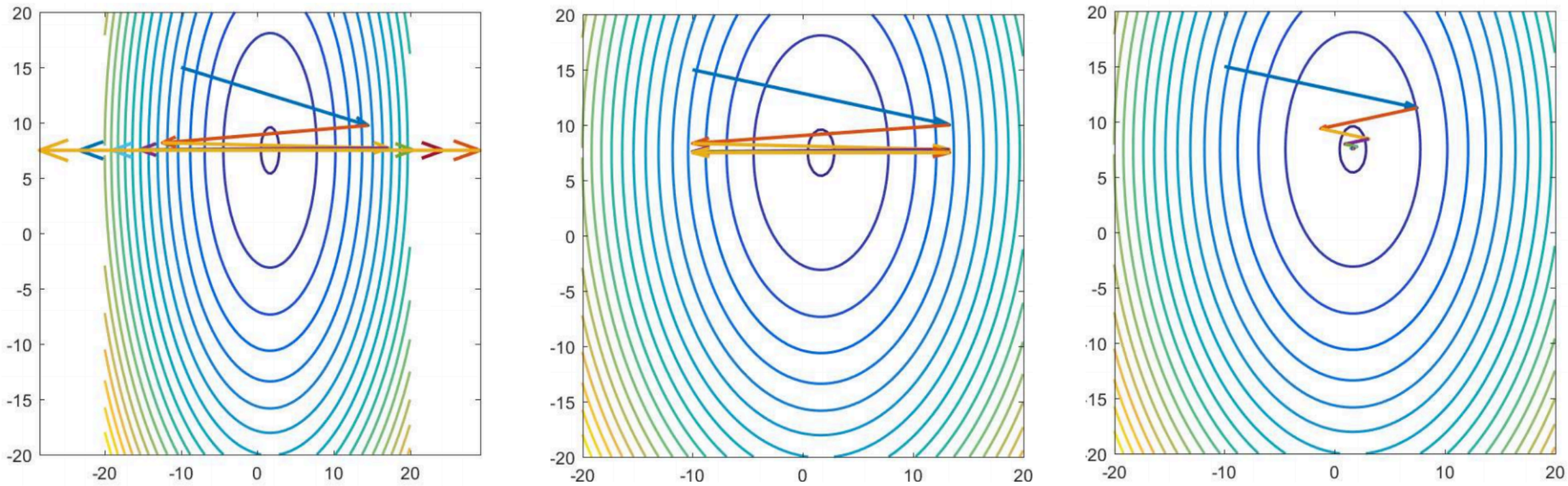
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

A closer look at the convergence problem



- With dimension-independent learning rates, the solution will converge smoothly in some directions, but oscillate or diverge in others

A closer look at the convergence problem



like using just one learning rate

- With dimension-independent learning rates, the solution will converge smoothly in some directions, but oscillate or diverge in others

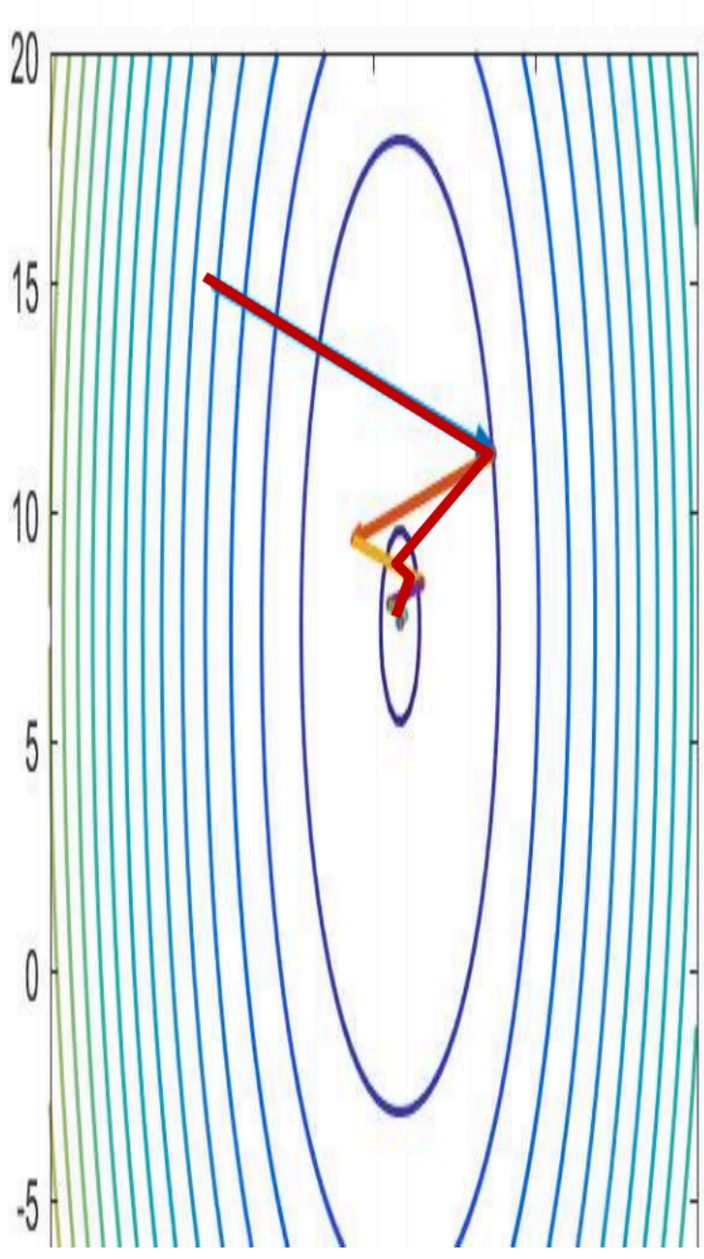
- **Proposal:**
  - Keep track of oscillations
  - Emphasize steps in directions that converge smoothly
  - Shrink steps in directions that bounce around..

The momentum methods

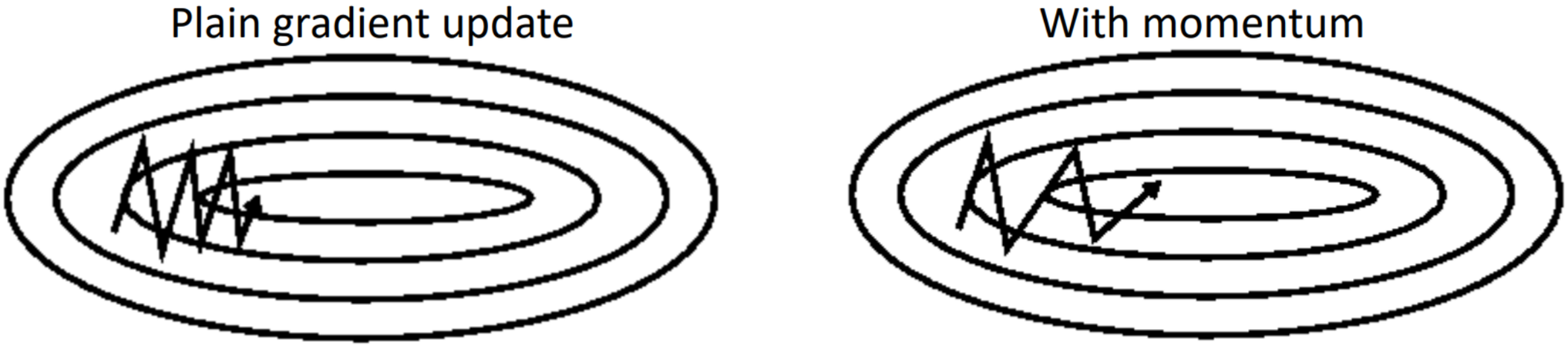
- Maintain a running average of all past steps
  - In directions in which the convergence is smooth, the average will have a large value
  - In directions in which the estimate swings, the positive and negative swings will cancel out in the average

smooth means no much oscillation

- Update with the running average, rather than the current gradient



Momentum Update



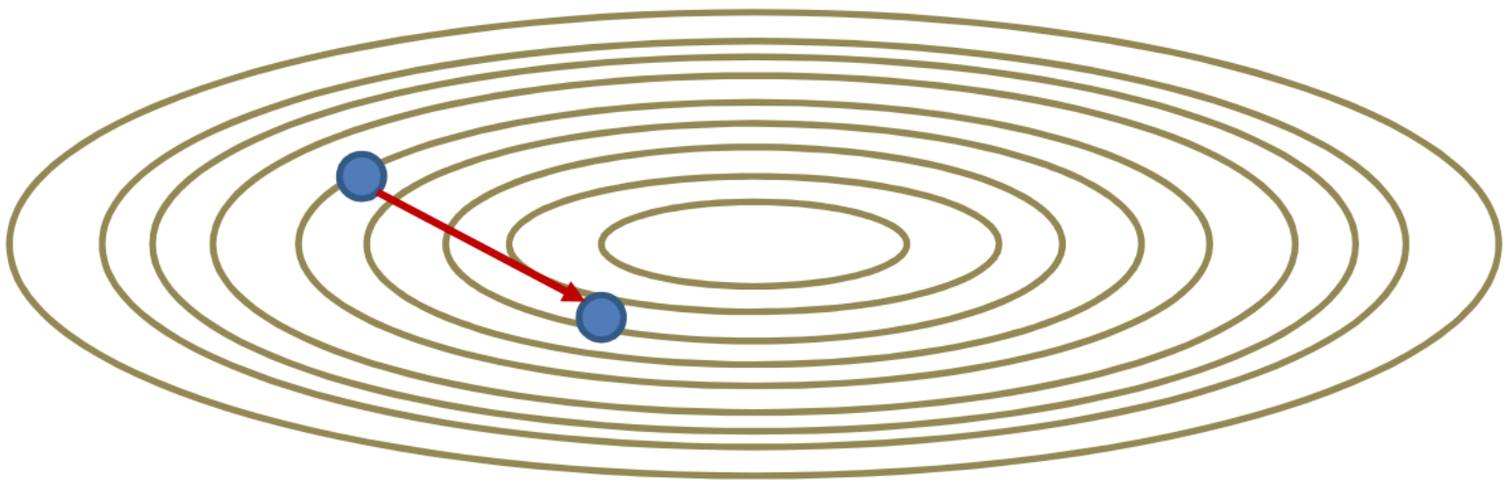
- The momentum method maintains a running average of all gradients until the *current* step

$$\Delta W^{(k)} = \beta \Delta W^{(k-1)} - \eta \nabla_W Err\left(W^{(k-1)}\right)$$

$$W^{(k)} = W^{(k-1)} + \Delta W^{(k)}$$

  - Typical β value is 0.9
- The running average steps
  - Get longer in directions where gradient stays in the same sign
  - Become shorter in directions where the sign keeps flipping

Momentum Update

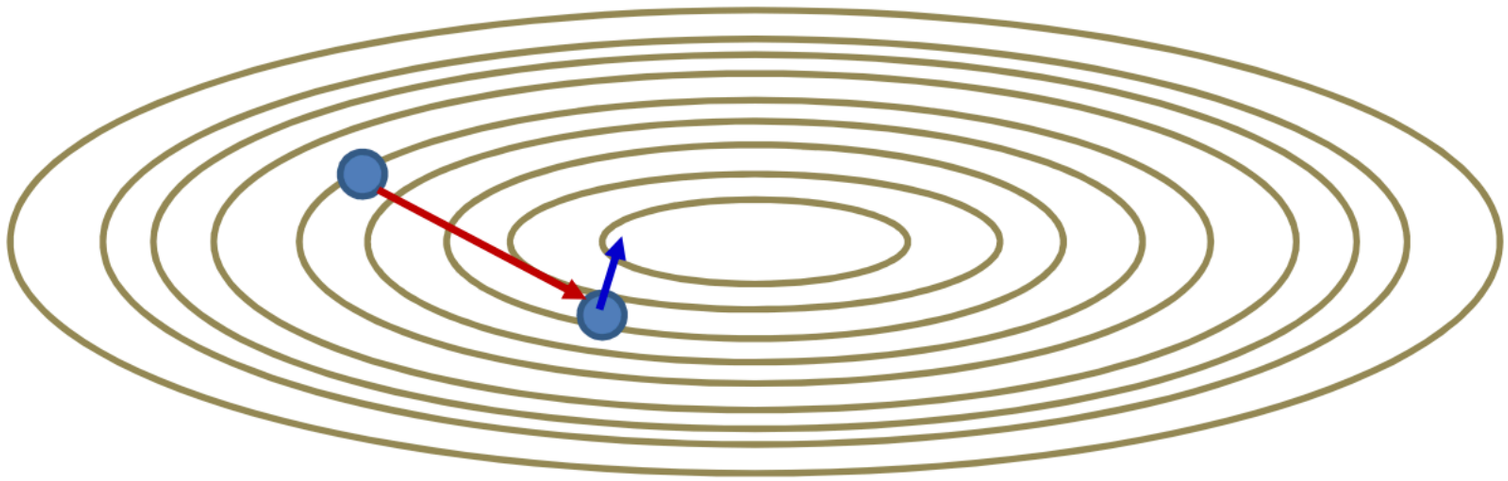


- The momentum method

$$\Delta W^{(k)} = \beta \Delta W^{(k-1)} - \eta \nabla_W Err\left(W^{(k-1)}\right)$$

- At any iteration, to compute the current step:

Momentum Update

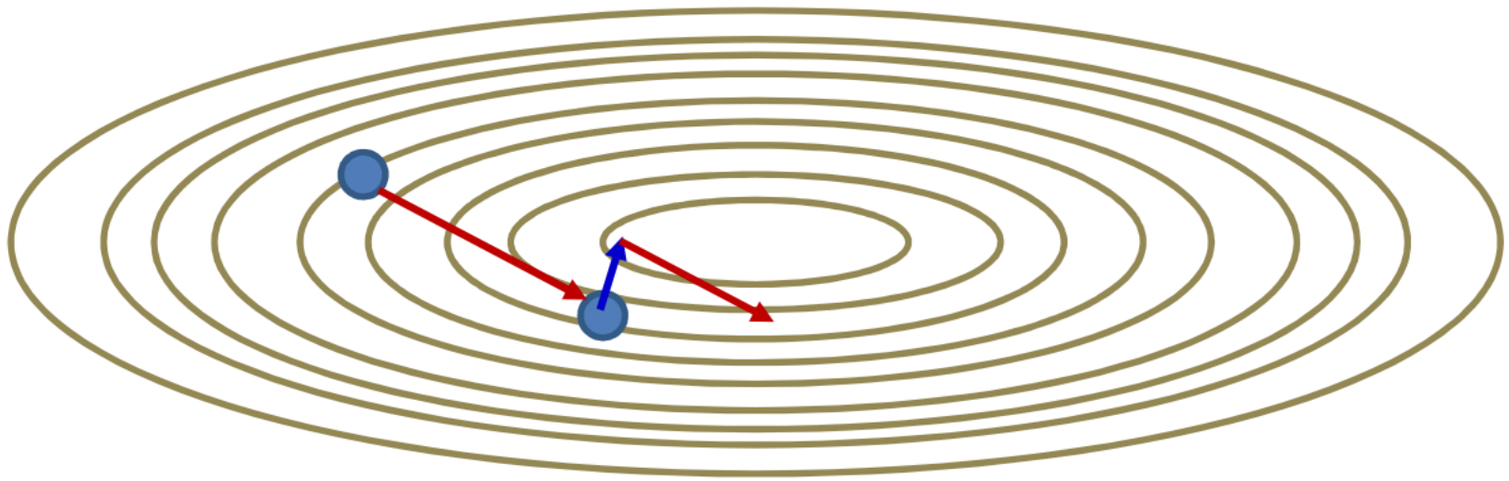


- The momentum method

$$\Delta W^{(k)} = \beta \Delta W^{(k-1)} - \eta \nabla_W Err\left(W^{(k-1)}\right)$$

- At any iteration, to compute the current step:
  - First computes the gradient step at the current location

Momentum Update

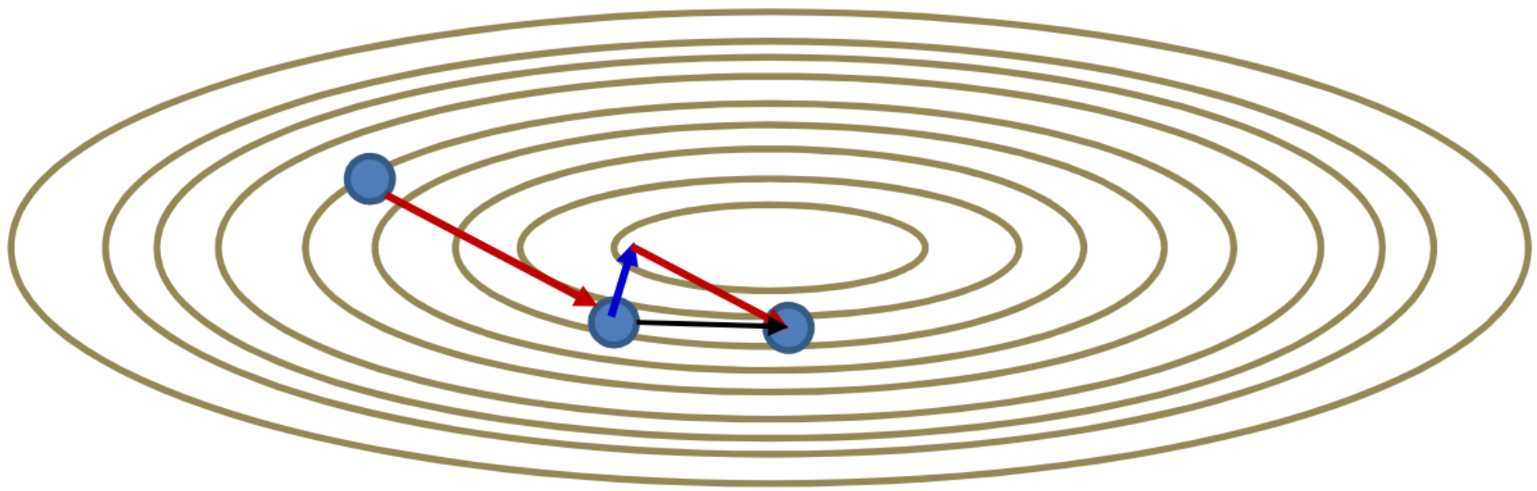


- The momentum method

$$\Delta W^{(k)} = \beta \Delta W^{(k-1)} - \eta \nabla_W Err\left(W^{(k-1)}\right)$$

- At any iteration, to compute the current step:
  - First computes the gradient step at the current location
  - Then adds in the scaled *previous* step

Momentum Update

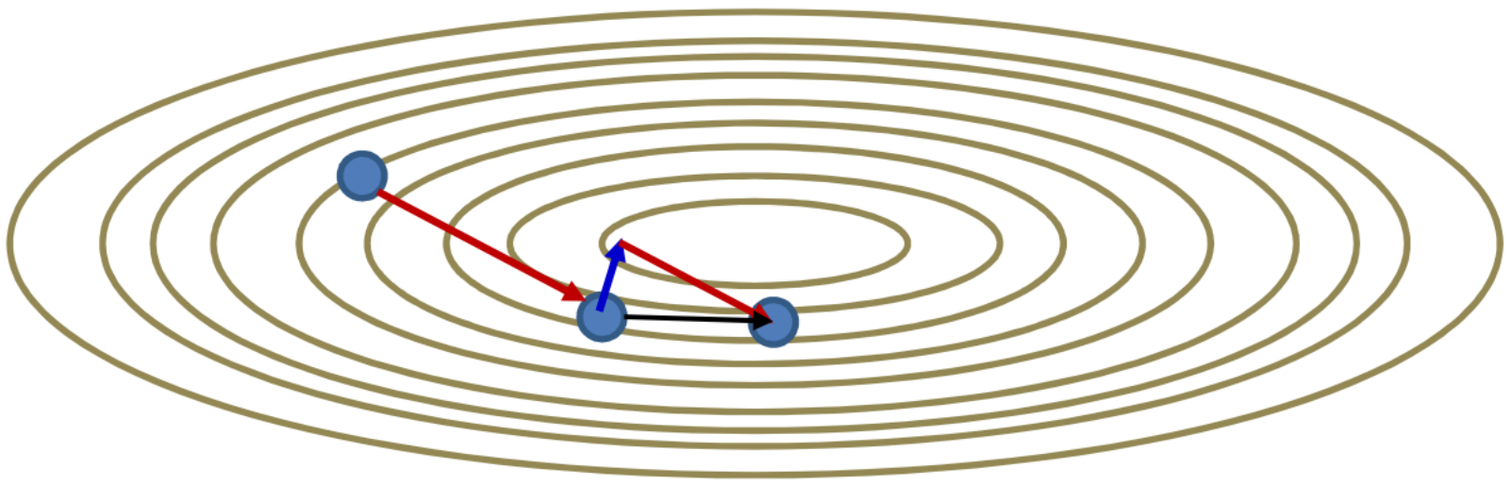


- The momentum method

$$\Delta W^{(k)} = \beta \Delta W^{(k-1)} - \eta \nabla_W Err\left(W^{(k-1)}\right)$$

- At any iteration, to compute the current step:
  - First computes the gradient step at the current location
  - Then adds in the scaled *previous* step
    - Which is actually a running average

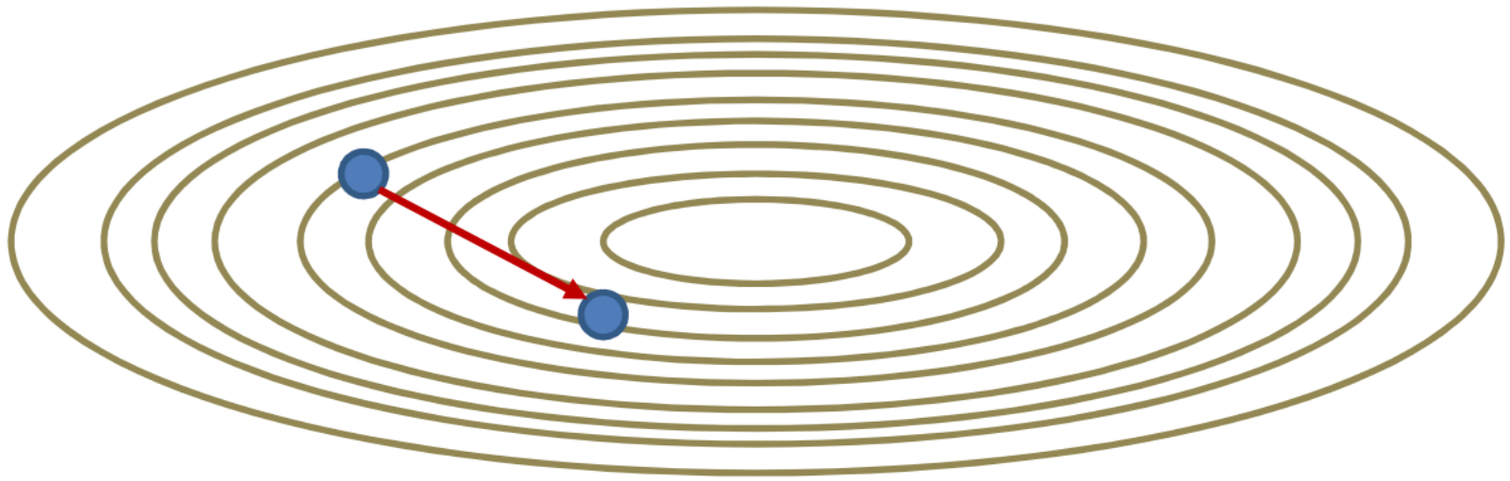
Momentum Update



- Takes a step along the past running average *after* walking along the gradient

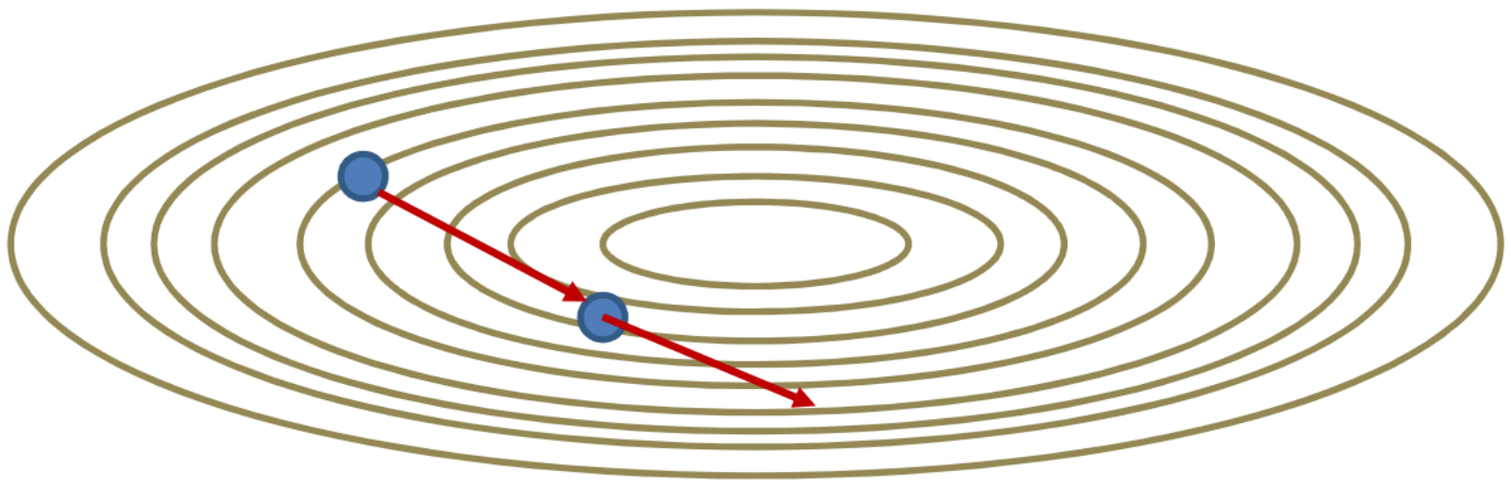
- The procedure can be made more optimal by reversing the order of operations..

Nestorov's Accelerated Gradient



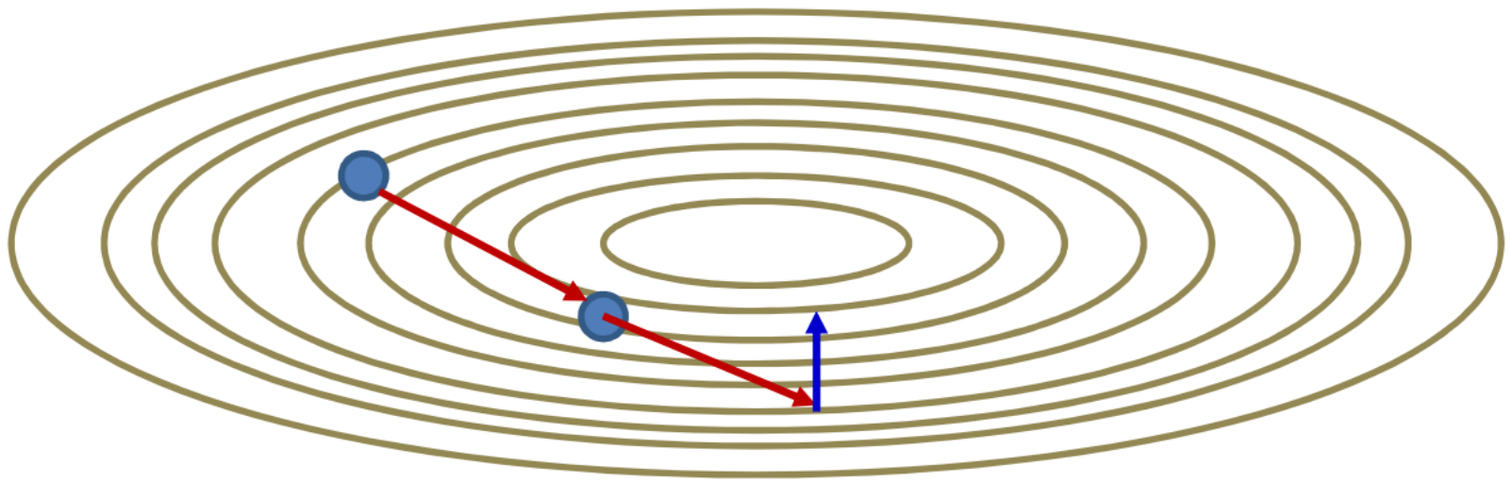
- Change the order of operations
- At any iteration, to compute the current step:

Nestorov's Accelerated Gradient



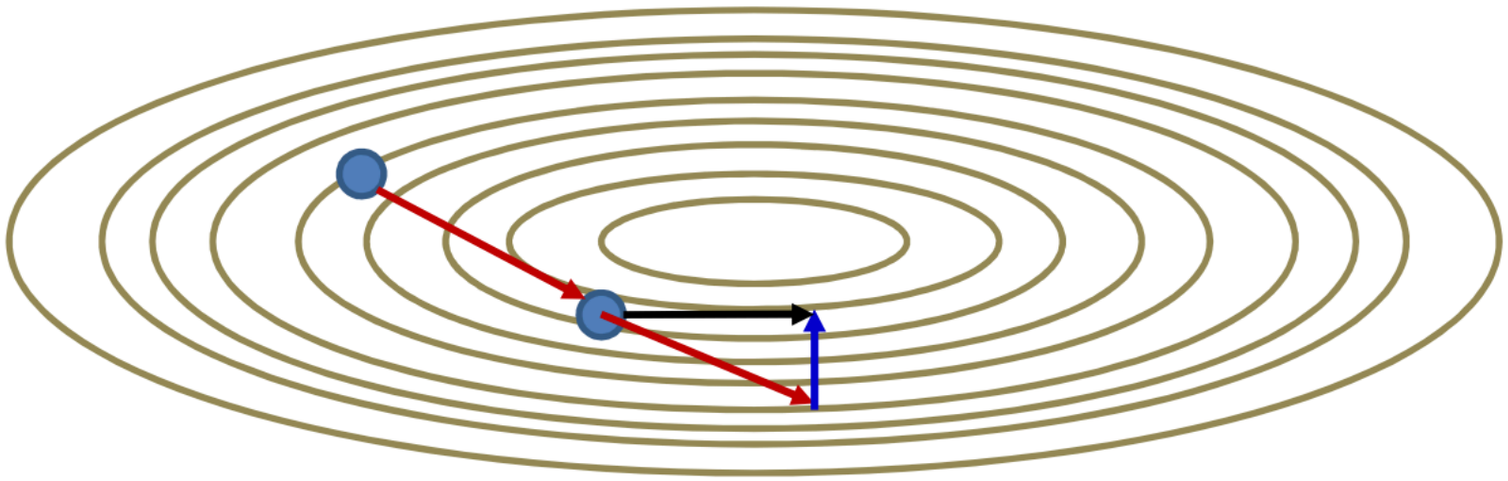
- Change the order of operations
- At any iteration, to compute the current step:
  - First extend the previous step

Nestorov's Accelerated Gradient



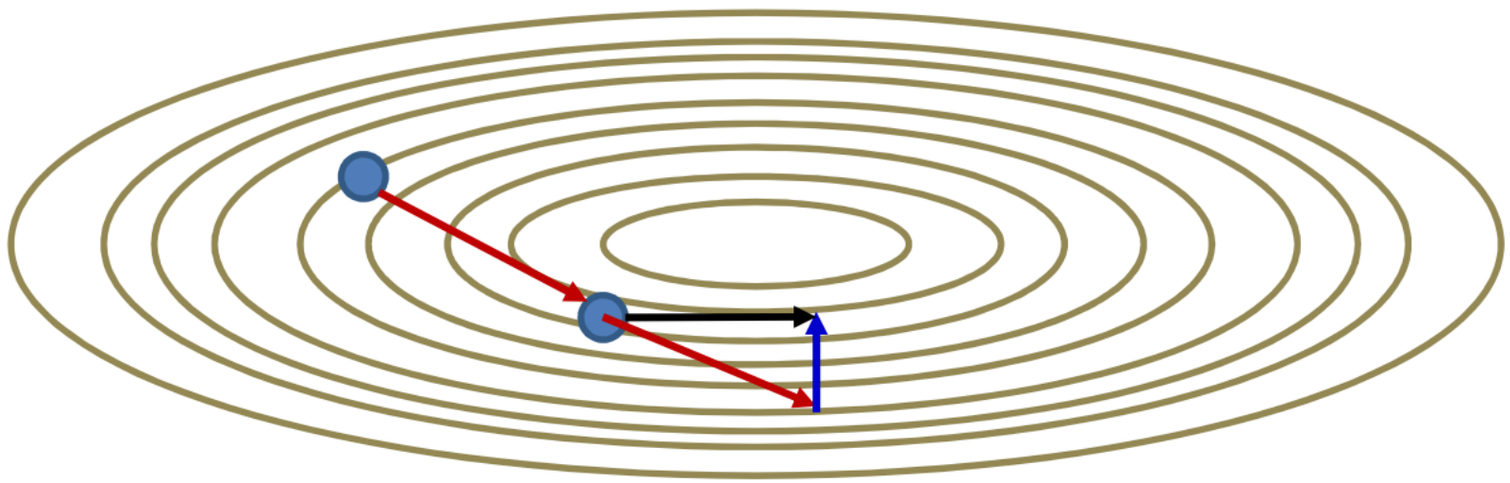
- Change the order of operations
- At any iteration, to compute the current step:
  - First extend the previous step
  - Then compute the gradient step at the resultant position

Nestorov's Accelerated Gradient



- Change the order of operations
- At any iteration, to compute the current step:
  - First extend the previous step
  - Then compute the gradient step at the resultant position
  - Add the two to obtain the final step

Nestorov's Accelerated Gradient



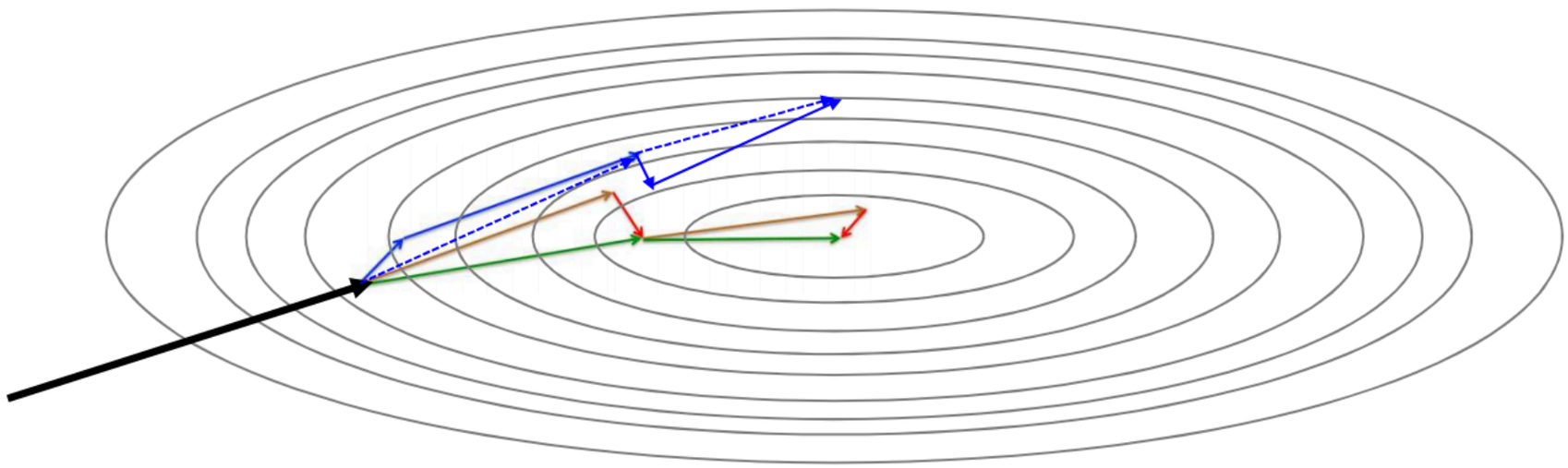
- Nestorov's method

Calculate the gradient differently

$$\Delta W^{(k)} = \beta \Delta W^{(k-1)} - \eta \nabla_W Err\left(W^{(k-1)} + \beta \Delta W^{(k-1)}\right)$$

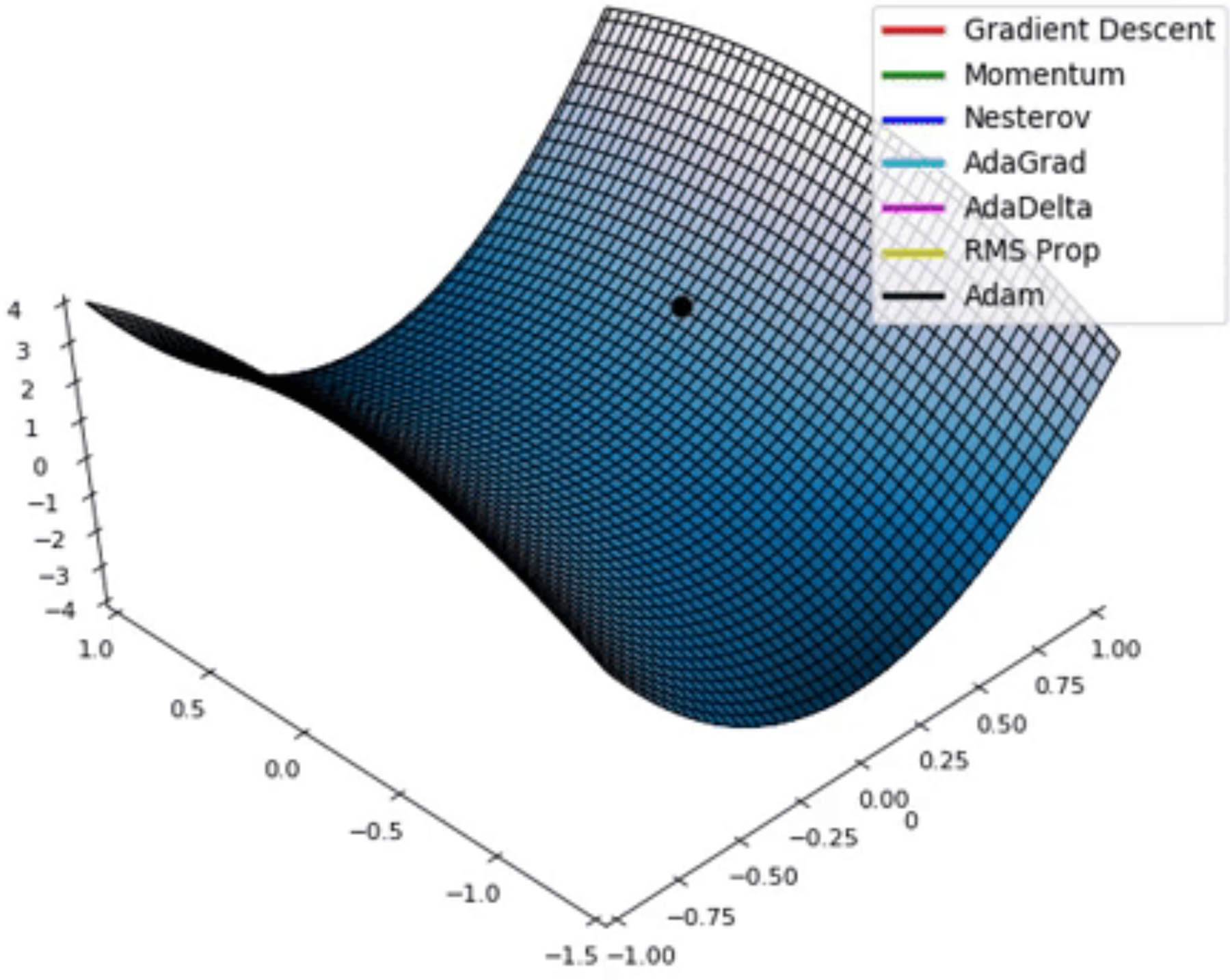
$$W^{(k)} = W^{(k-1)} + \Delta W^{(k)}$$

Nestorov's Accelerated Gradient



- Comparison with momentum (example from Hinton)
  - Blue: Momentum
    - Dotted line is the final update at each iteration
  - Brown/ochre/green: Nestorov
    - Brown is (scaled) previous update, ochre is gradient, green is the final update
- Converges much faster

Demo


Gradient Descent
Momentum
Nesterov
AdaGrad
AdaDelta
RMS Prop
Adam
4
3
2
1
0
−1
−2
−3
−4
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.00
−0.75
−0.50
−0.25
0.00
0
0.25
0.50
0.75
1.00

So Far

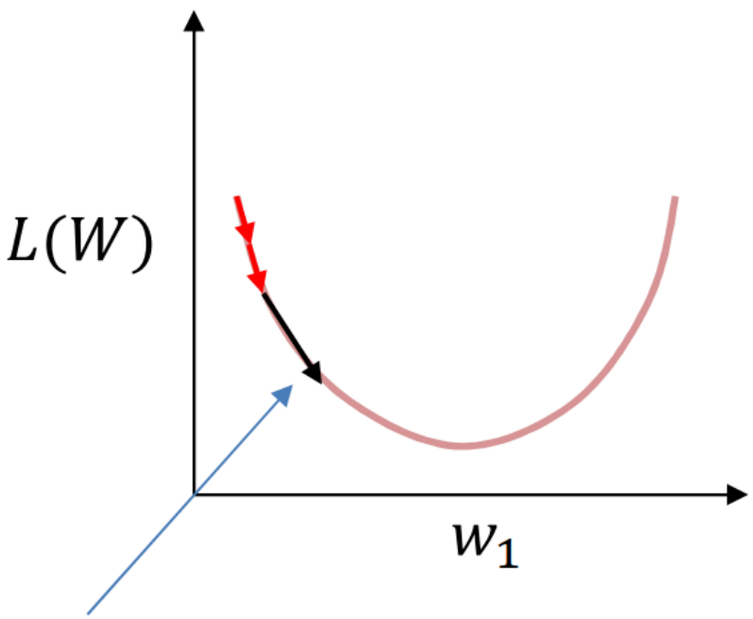
- Gradient descent can miss obvious answers
  - And this may be a *good* thing

普通的

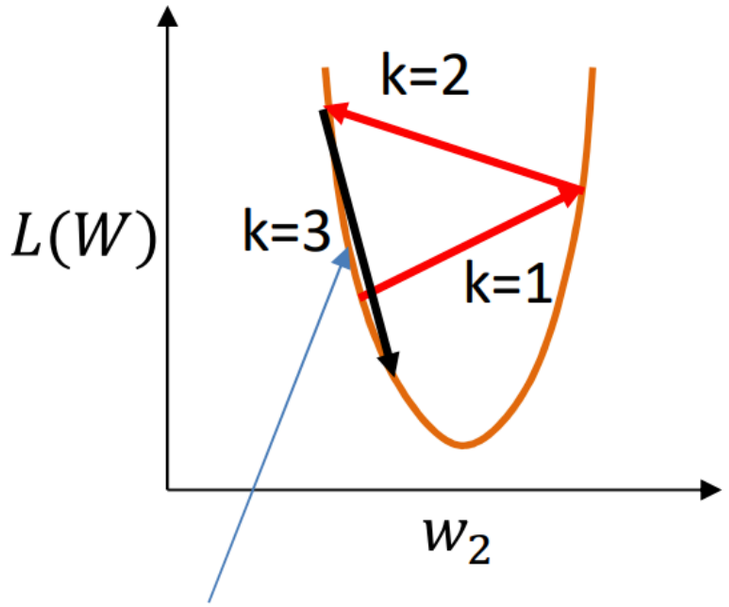
- Vanilla gradient descent may be too slow or unstable due to the differences between the dimensions
- Second order methods can normalize the variation across dimensions, but are complex
- Adaptive or decaying learning rates can improve convergence
- Methods that decouple the dimensions can improve convergence
- Momentum methods which emphasize directions of steady improvement are demonstrably superior to other methods

Momentum Methods: Principle

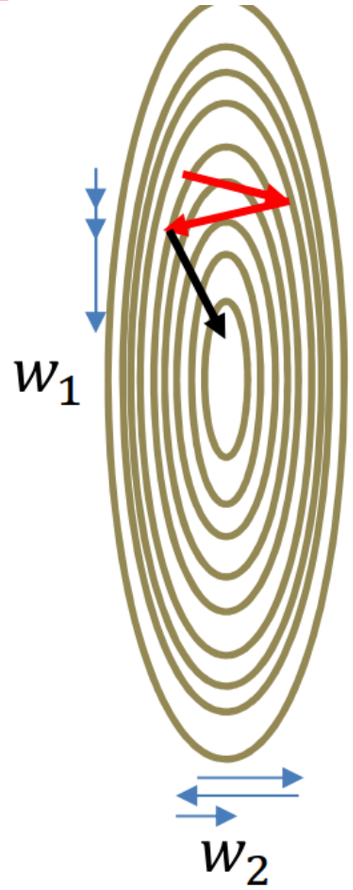
$$W_k = W_{k-1} - \eta \nabla_w L(W)^T$$



Increase stepsize because previous updates consistently moved weight right



Decrease stepsize because previous updates kept changing direction

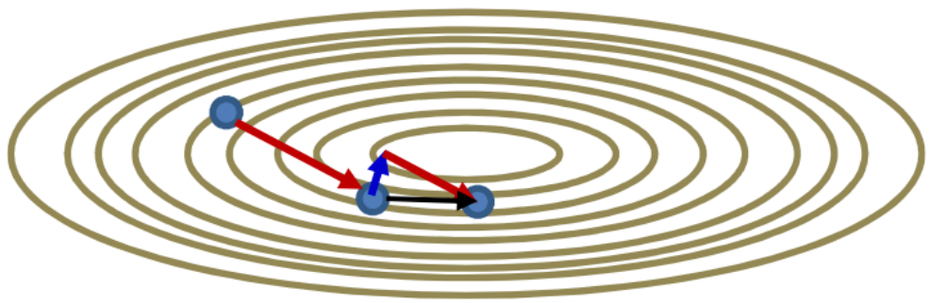


Stepsize shrinks along w2 but increases along w1

- Ideally: Have component-specific step size 其实是让其自适应，从而来获得类似效果
  - Too many independent parameters (maintain a step size for every weight/bias)
- Adaptive solution: Start with a common step size
  - *Shrink* step size in directions where the weight oscillates
  - *Expand* step size in directions where the weight moves consistently in one direction

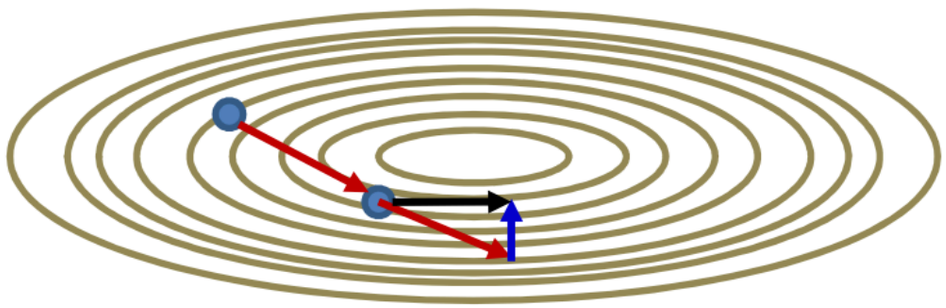
Quick Recap: Momentum Methods

Momentum



$$\Delta W^{(k)} = \beta \Delta W^{(k-1)} - \eta \nabla_W Err\left(W^{(k-1)}\right)$$

Nestorov



$$W^{(k)}_{extend} = W^{(k-1)} + \beta \Delta W^{(k-1)}$$

$$\Delta W^{(k)} = \beta \Delta W^{(k-1)} - \eta \nabla_W Err\left(W^{(k)}_{extend}\right)$$

$$W^{(k)} = W^{(k-1)} + \Delta W^{(k)}$$

- Momentum: Retain gradient value, but *smooth out* gradients by maintaining a running average
  - Cancels out steps in directions where the weight value oscillates
  - Adaptively increases step size in directions of consistent change