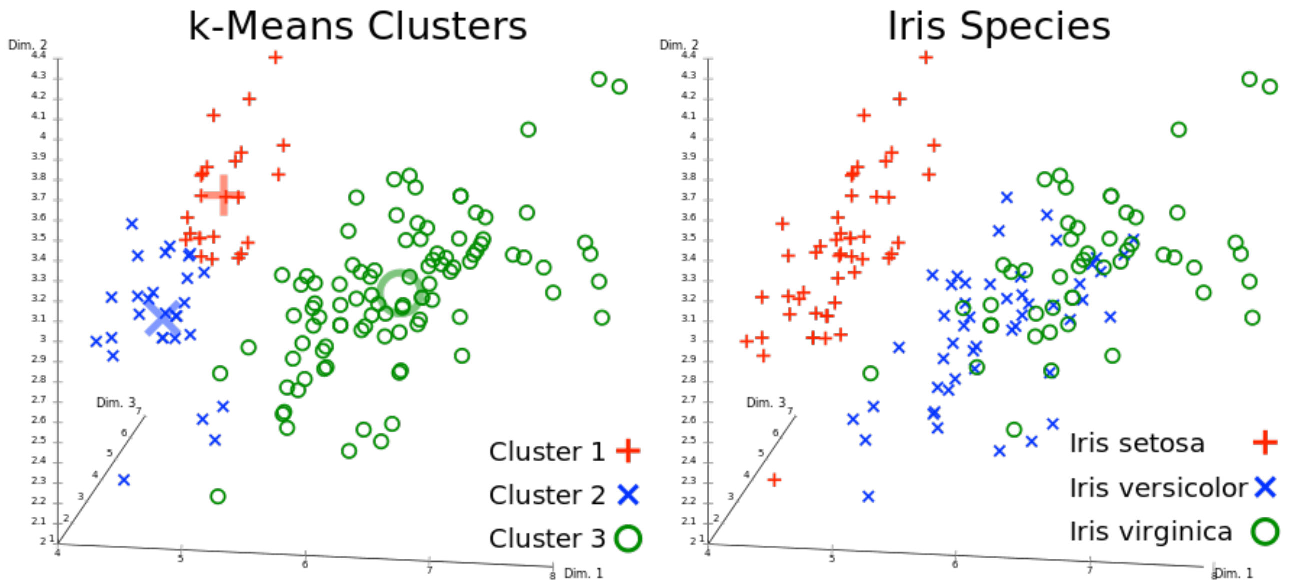
Statistical learning theory, deep neural networks, and regularization

March 10, 2020
IPAM tutorials
HJ PDE
Adam Oberman
Math and Stats, McGill

Traditional Machine Learning

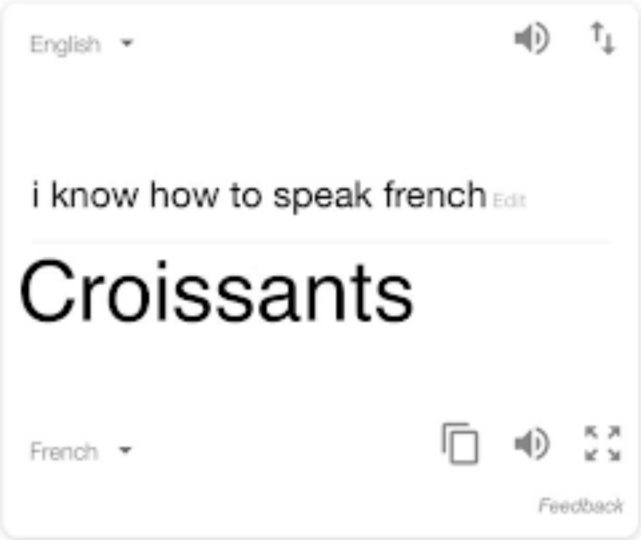
- Supervised learning (classification, regression)
  - Spam detection using Bayes' Theorem
- Clustering (e.g. k-means)
- Dimensionality reduction
- Rigorous theory:
  - learning guarantees, error bounds
  - interpretable (which factors influenced decision)
  - robust

$$\Pr(S|W) = \frac{\Pr(W|S)\cdot\Pr(S)}{\Pr(W|S)\cdot\Pr(S)+\Pr(W|H)\cdot\Pr(H)}$$



Deep Learning

- Image Classification
- Natural Language Classification
  - e.g. Google Translate
- Reinforcement Learning
  - e.g. Chess, Go
- Generative Models
  - e.g. Deep Fakes



i know how to speak french

Croissants

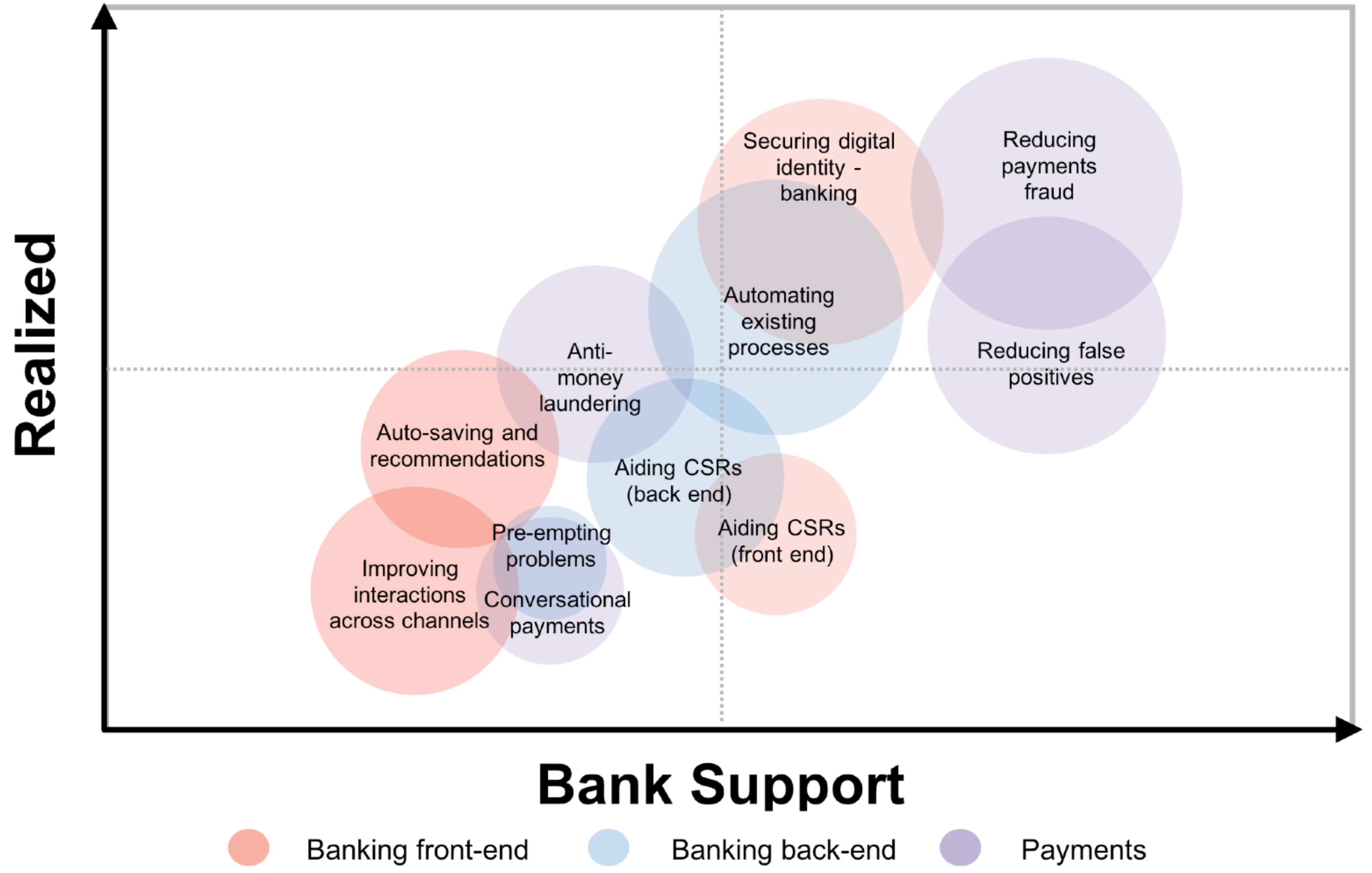
Deep learning is ***way cooler*** than traditional ML.
More accurate, harder problems, etc.

Now used in many applications (medical, banking, commerce, etc)

But basically no theory. no guarantees. not interpretable.

Deep Learning, e.g. Banking

Maturity Of Uses Of Artificial Intelligence In Banking And Payments



Size of bubble = Five-year potential ROI

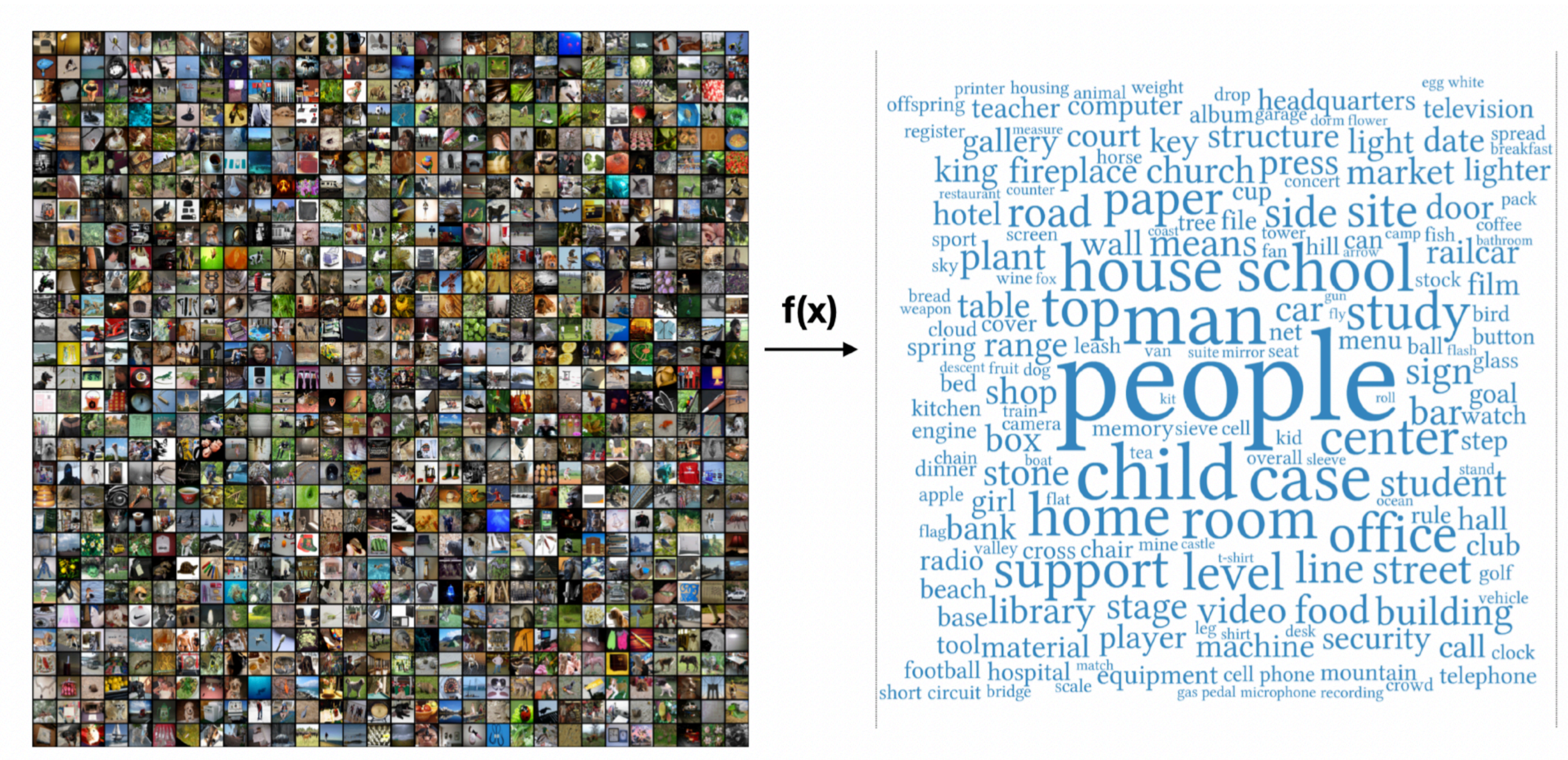
Source: Estimated qualitatively by BI Intelligence analysts

BI INTELLIGENCE

Image Classification by CNNs

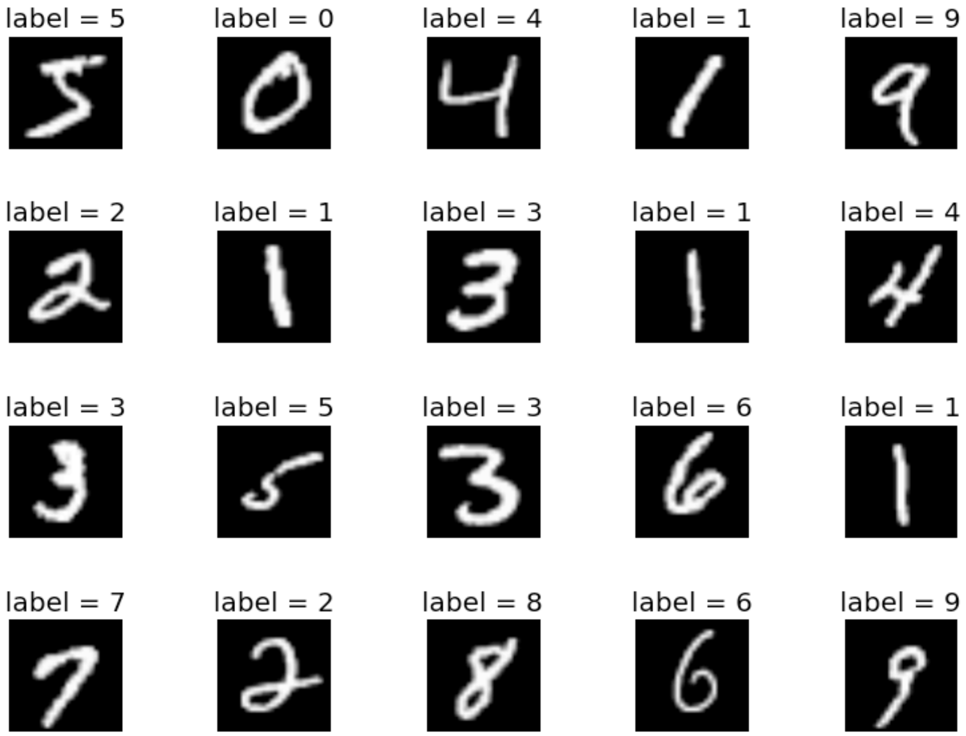
Fitting a map from images sampled from a distribution to labels

Approximation Theory + Random sampling:
= Statistical Learning Theory



The MNIST dataset consists of m = 70,000, d = 28X28 greyscale images of handwritten digits. CNNs achieve an error of less than 1% on MNIST.

On this simple data set, support vector machines (SVM) achieve accuracy almost as high.



Fashion MNIST

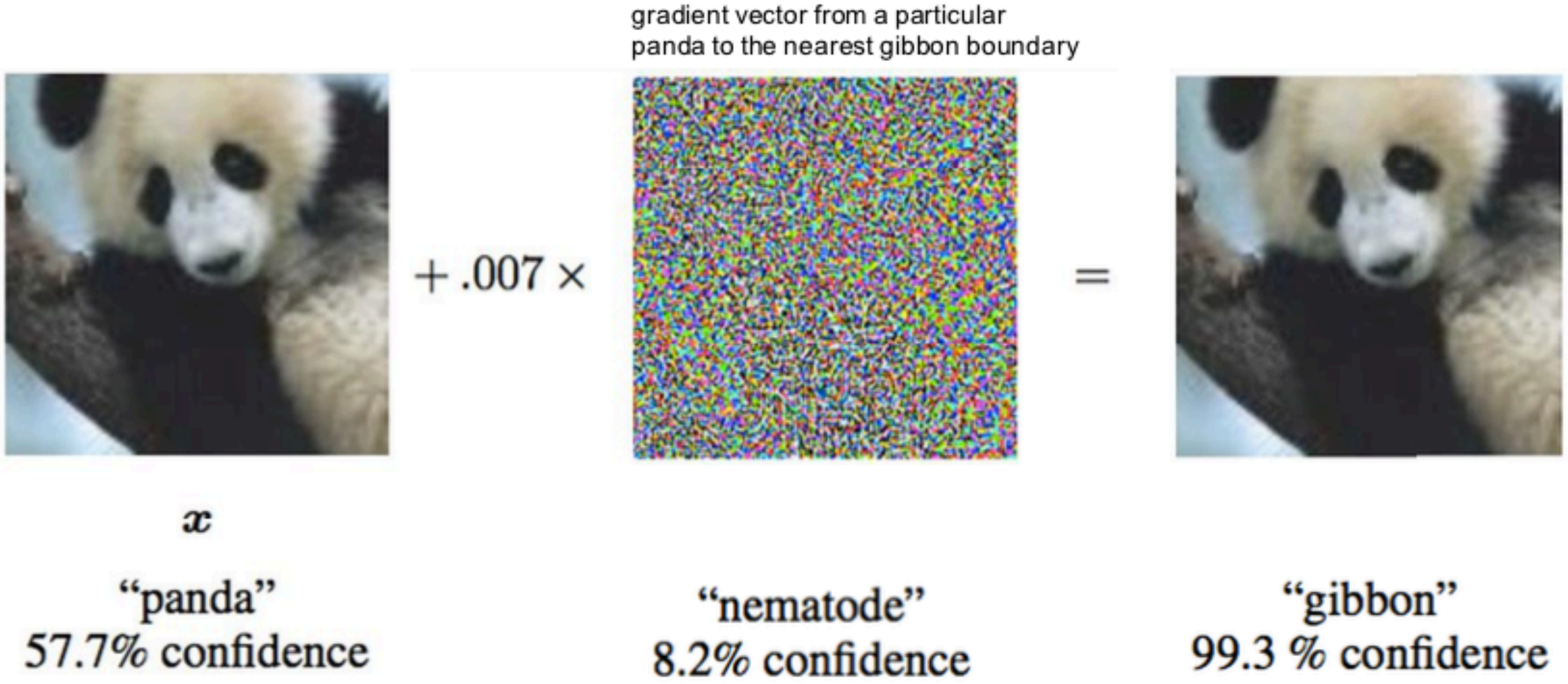
ImageNet

- ImageNet: Total number of classes: m =21841
- Total number of images: n =14,197,122
- Color images d= 3*256*256= 196,608

Last year: Facebook used 256 GPUs, working in parallel, to train ImageNet.

This year: using more efficient code, we can train using 4GPUs in about 15 hours.

Vulnerable to adversarial attacks



Small (visually imperceptible) perturbations of an image lead to misclassification

Source: EXPLAINING AND HARNESSING ADVERSARIAL EXAMPLES, Goodfellow

Machine Learning Problem: Supervised Classification

Let $x \in \mathcal{X} \subset [0,1]^d$ and the dimension is large, $d \gg 1$.
Labels $\mathcal{Y} = \{1, \dots, K\}$,
The dataset

$$S_m = \{(x_1, y_1), \dots, (x_m, y_m)\} \tag{2.1}$$

consists of m samples, x_i, drawn i.i.d. from a data distribution, $\rho(x)$, with support $\mathcal{X}$. The labels $y_i \in \mathcal{Y}$ are also given.

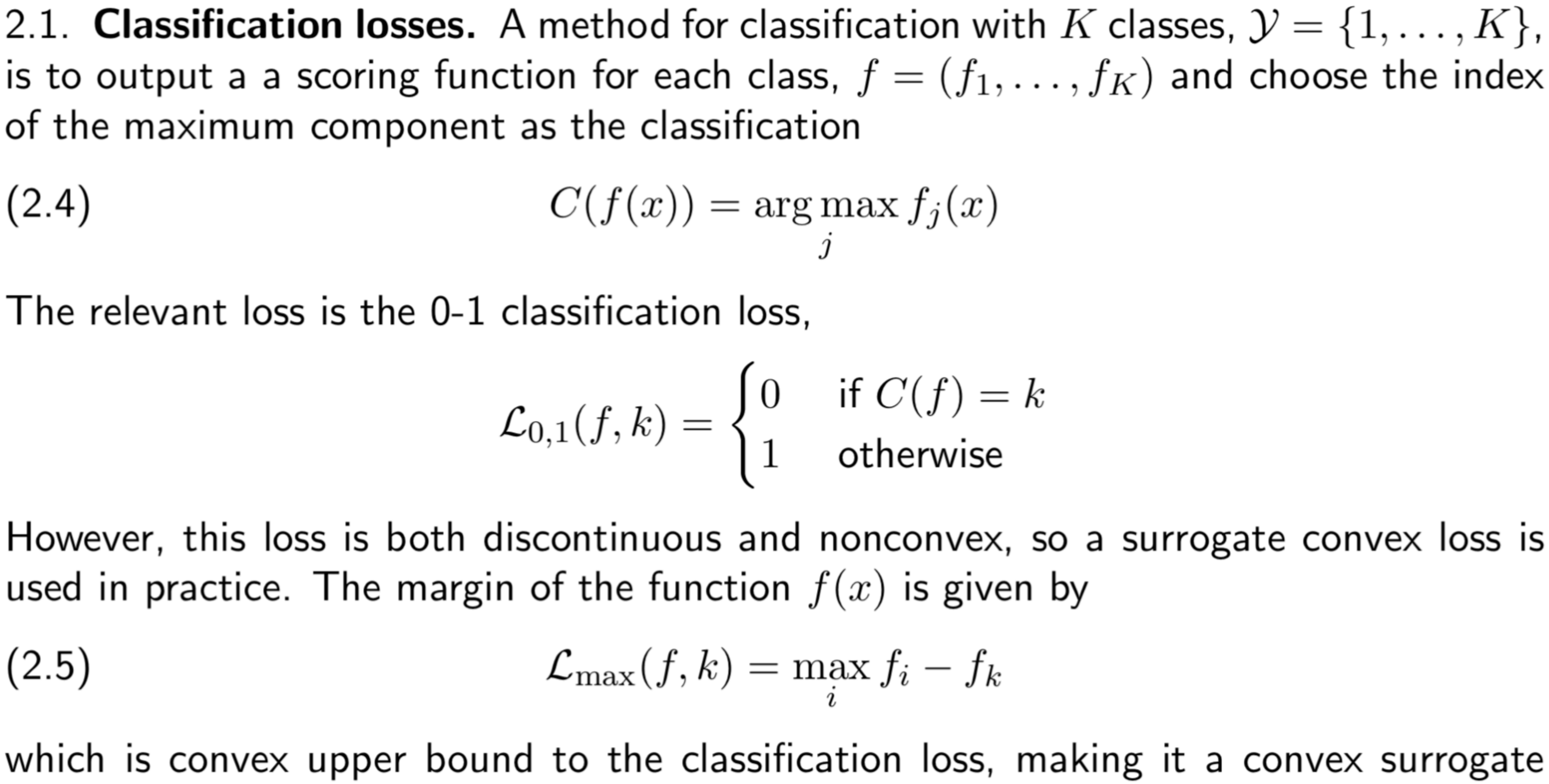
$\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$ is a loss function if it is zero iff $y_1 = y_2$.
Our objective is to find a function $f : \mathcal{X} \to \mathcal{Y}$ which minimizes the expected loss

$$L_{\mathcal{D}}(f) = \mathbb{E}_{x\sim\rho}[\mathcal{L}(f(x), y(x))] = \int_{\mathcal{X}} \mathcal{L}(f(x), y(x)) d\rho(x) \tag{2.2}$$

Intractable. Instead minimize the empirical loss

$$L_{S_m}[f] = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(f(x_i), y_i). \tag{EL}$$

Losses



2.1. **Classification losses.** A method for classification with K classes, $\mathcal{Y} = \{1, \dots, K\}$, is to output a a scoring function for each class, $f = (f_1, \dots, f_K)$ and choose the index of the maximum component as the classification

$$C(f(x)) = \arg\max_j f_j(x) \tag{2.4}$$

The relevant loss is the 0-1 classification loss,

$$\mathcal{L}_{0,1}(f, k) = \begin{cases} 0 & \text{if } C(f) = k \\ 1 & \text{otherwise} \end{cases}$$

However, this loss is both discontinuous and nonconvex, so a surrogate convex loss is used in practice. The margin of the function $f(x)$ is given by

$$\mathcal{L}_{\max}(f, k) = \max_i f_i - f_k \tag{2.5}$$

which is convex upper bound to the classification loss, making it a convex surrogate

Analyze the 0-1 loss. *Train* with convex surrogate loss

Parametric Hypothesis Class

Typically, the functions considered are restricted to a parametric class

$$\mathcal{H} = \{f(x, w) \mid w \in \mathbb{R}^D\} \tag{2.3}$$

so that minimization of (EL) can be rewritten as the finite dimensional optimization problem

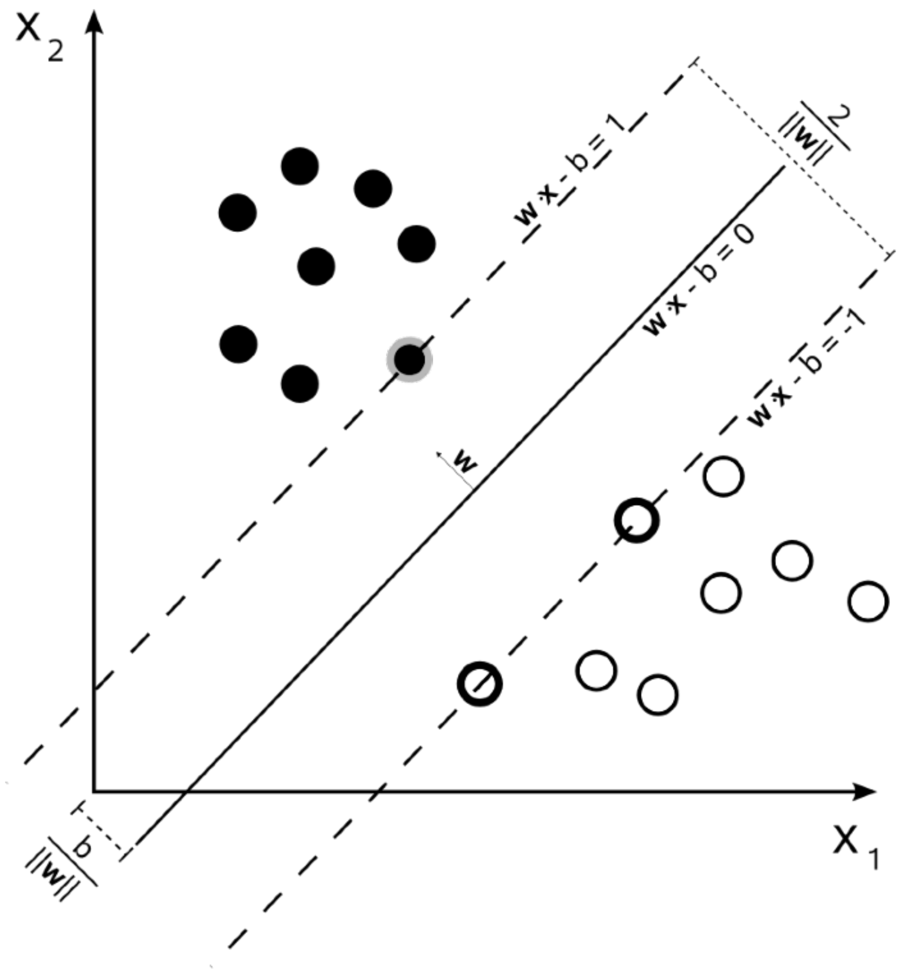
$$\min_w \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(f(x_i, w), y_i). \tag{EL-W}$$

The problem with using (EL-W) as a surrogate for (2.2), is that a minimizer of (EL-W) could *overfit*, meaning that the expected loss is much larger than the empirical loss. Classical machine learning methods avoid overfitting by restricting $\mathcal{H}$ to be a class of simple (e.g. linear) functions.

Remark 2.1. To first approximation, we can assume that the images are free of noise, that all labels are correct, and that there are no ambiguous images. In other words, $y_i = y(x_i)$ for a label function $y(x)$. Challenge in learning $y(x)$ comes not from uncertainty, noise, or ambiguity, but rather from the complexity of the functions involved.

Support vector machines

- Linear classifier
- Regularize by adding margin.

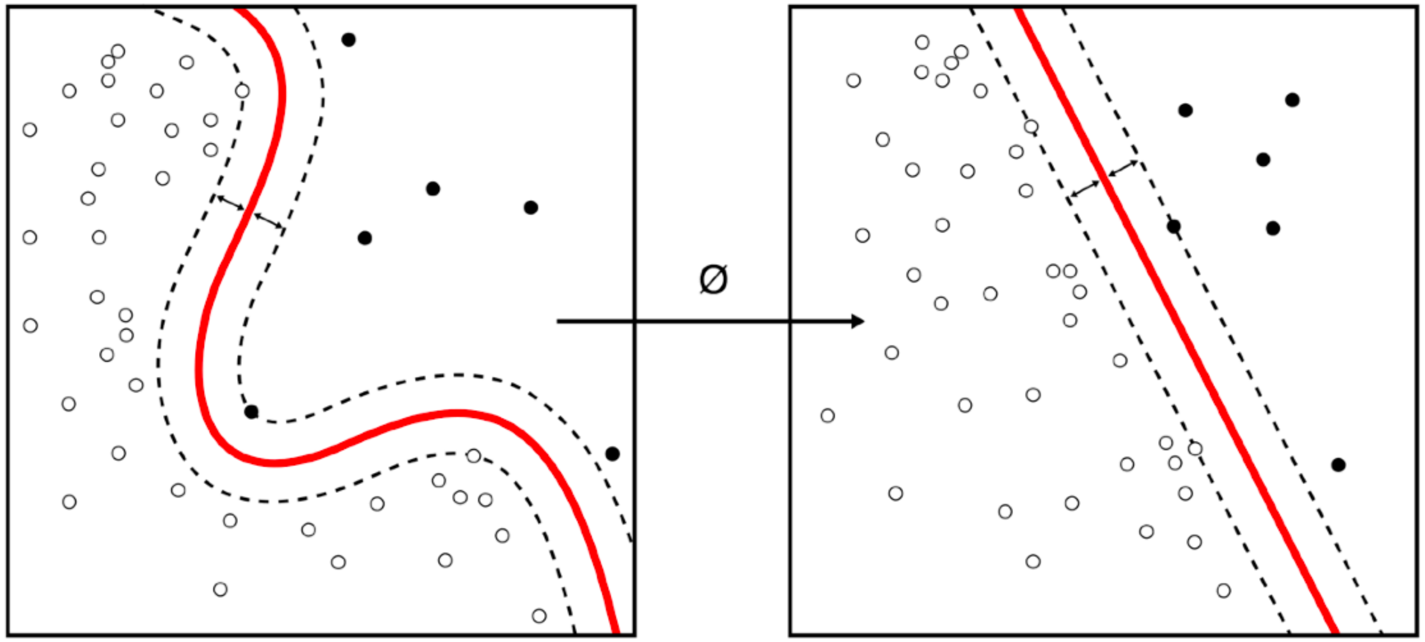


$$\min_w \frac{1}{m}\sum_{i=1}^m \mathcal{L}(f(x_i;w),y_i) + \lambda\|w\|^2$$

$$= \min_w \frac{1}{m}\sum_{i=1}^m \mathcal{L}(f(x_i;w),y_i) + \lambda\|\nabla_x f(x,w)\|^2$$

Can write margin as gradient regularization! Without it, no dimension-independent generalization bounds.

Kernel Methods



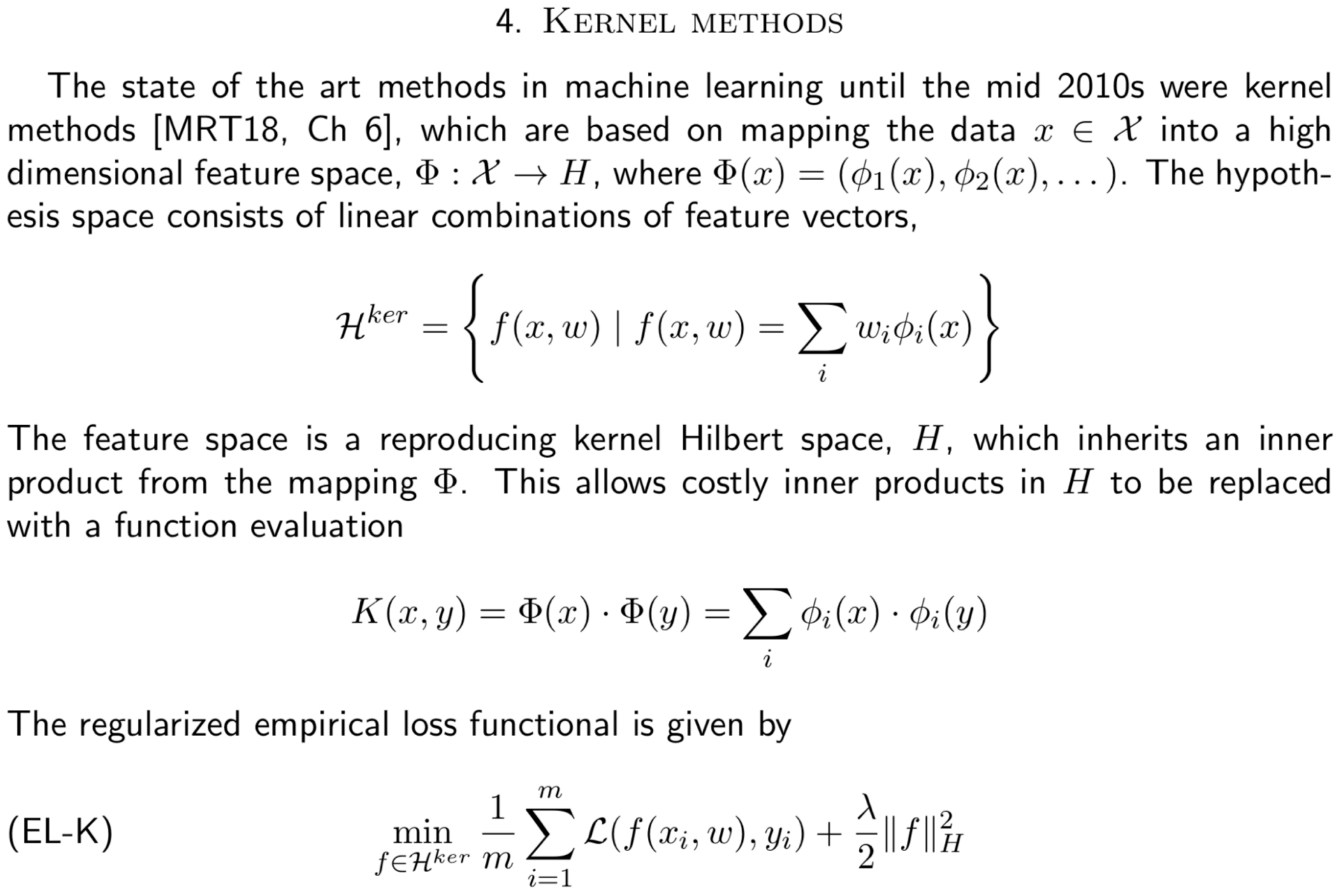
Wikipedia ML image

- Map into higher dimensional feature space.

- Do linear classification (with margin) in the feature space.

- Math translation: margin in feature space = function regularization

$$\min_{f \in \mathcal{H}^{ker}} \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(f(x_i, w), y_i) + \frac{\lambda}{2} \|f\|_H^2$$



4. Kernel methods

The state of the art methods in machine learning until the mid 2010s were kernel methods [MRT18, Ch 6], which are based on mapping the data $x \in \mathcal{X}$ into a high dimensional feature space, $\Phi : \mathcal{X} \to H$, where $\Phi(x) = (\phi_1(x), \phi_2(x), \dots)$. The hypothesis space consists of linear combinations of feature vectors,

$$\mathcal{H}^{ker} = \left\{ f(x,w) \mid f(x,w) = \sum_i w_i \phi_i(x) \right\}$$

The feature space is a reproducing kernel Hilbert space, H, which inherits an inner product from the mapping Φ. This allows costly inner products in H to be replaced with a function evaluation

$$K(x,y) = \Phi(x) \cdot \Phi(y) = \sum_i \phi_i(x) \cdot \phi_i(y)$$

The regularized empirical loss functional is given by

$$\text{(EL-K)} \qquad \min_{f \in \mathcal{H}^{ker}} \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(f(x_i, w), y_i) + \frac{\lambda}{2} \|f\|_H^2$$

H-norm term is a regularization term - Fourier smoothing in many cases!
So most important rigorous method has a PDE style regularization!
Somewhat ignored by modern machine learning community, prefer to think about the kernel *method*, i.e. *algorithmic aspects*

Example of Fourier Kernel Regularization

Kernels and regularization. Regularization interpretation of kernels is discussed in [GJP95, SS98, Wah90].

Consider the case where $K(x_1, x_2) = G(x_1 - x_2)$ where G is real and symmetric, and the Fourier transform $\hat{G}(y)$ is a symmetric, positive function that goes to zero as $y \to \infty$. Then

$$\mathcal{R}^{Ker}(f) = \int_{\mathbb{R}^n} \frac{\|\hat{f}(y)\|^2}{\hat{G}(y)}\, dy, \tag{10.1}$$

where $\hat{f}$ is the Fourier transform of f. Refer to [GJP95].

Example 10.1. See [SS98] for details. The Gaussian kernel corresponds to

$$G(x) = \exp(-\|x\|^2/2), \qquad \hat{G}(y) = C\exp(-\|y\|^2/2).$$

In this case, the regularization is given by

$$\mathcal{R}^{Ker}(f) = \sum_{n=0}^{\infty} \frac{1}{2^n n!} \|(\nabla)^n f\|_{L^2}^2.$$

Thus we see that kernel methods can be interpreted as regularized functional (EL-R) with Fourier regularization.

Kernel Algorithms - Quadratic case

For convex losses, (EL-K) is a convex optimization in w. For classification, the margin loss is used, and the optimization problem corresponds to quadratic programming. In the case of quadratic losses, the optimization problem is quadratic, and the minimizer of (EL-K) has the explicit form

$$f(x) = \sum_{i=1}^{m} w_i K(x, x_i), \qquad (M + \lambda I)w = y$$

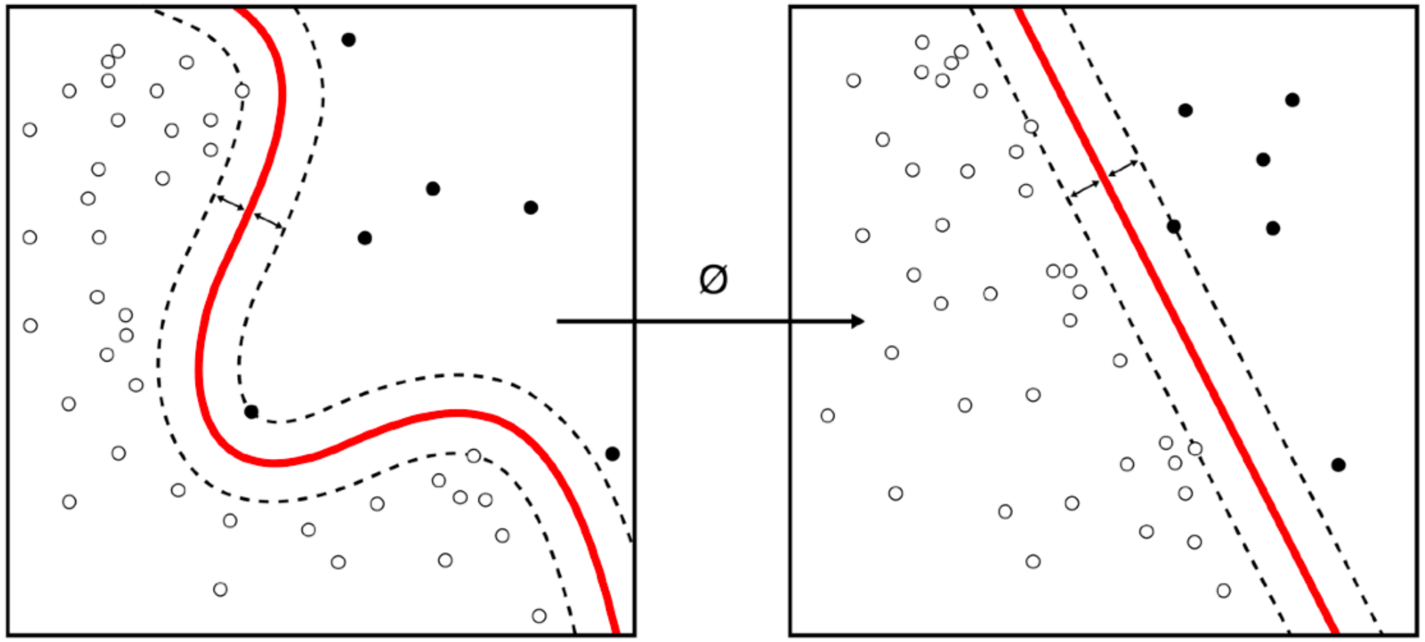
where the coefficients c are given by the solution of the system of linear equations with $M_{ij} = K(x_i, x_j)$, and I is the identity matrix. Note that the regularization term has a stabilizing effect: the condition number of the system with λI improves with $\lambda > 0$. Better conditioning of the linear system means that the optimal weights are less sensitive to changes in the data

$$w^* = (M + \lambda I)^{-1} y$$

Algorithm to minimize (EL-K) are designed to be written entirely in terms of inner products, allowing for high dimensional feature spaces.

E.g. this algorithmic formulation forget the regularization in hilbert space,
and just looks like a regularized linear system.
Thinking about the **weights, not the functions**

Not the traditional perspective!



Wikipedia ML image

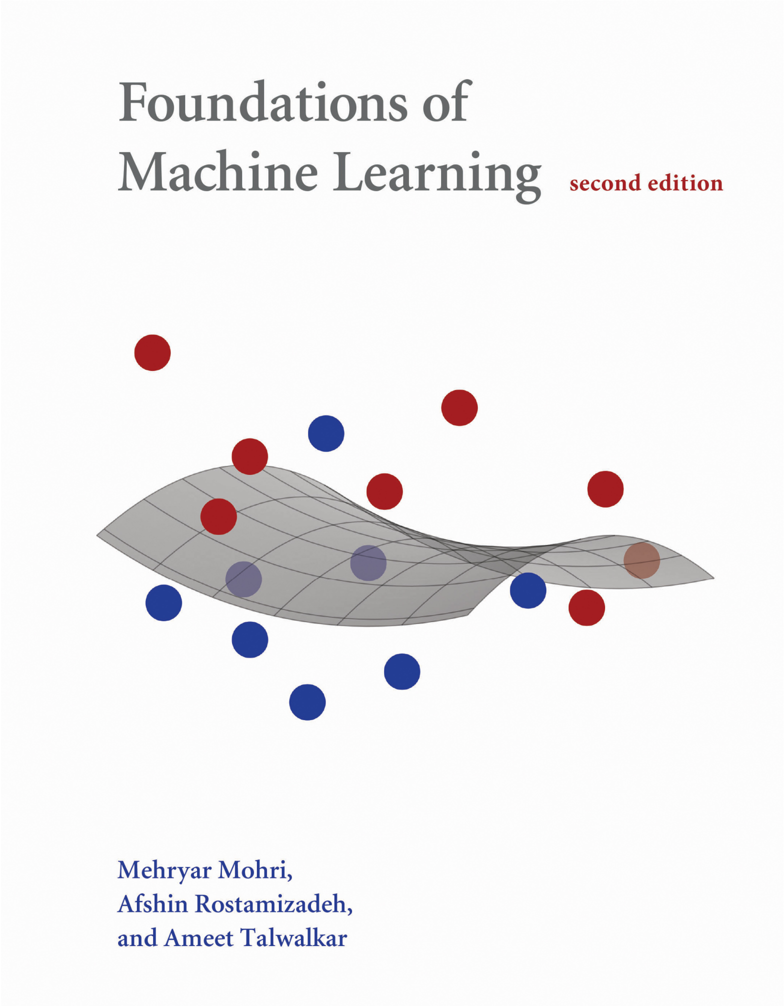
$$\min_{w} \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(f(x_i; w), y_i) + \lambda \|\nabla_x f(x, w)\|^2$$

$$\min_{f \in \mathcal{H}^{ker}} \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(f(x_i, w), y_i) + \frac{\lambda}{2} \|f\|_H^2$$

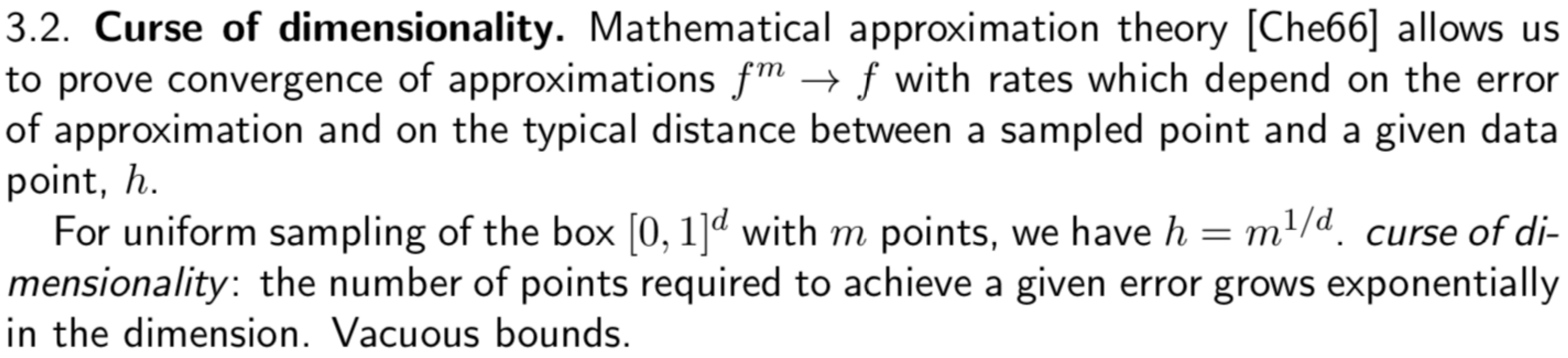
Is there a missing regularization for DNN which will lead to generalization bounds?

Statistical Learning Theory

- Interpolation: learn a function from high dimensional data.
- Need to overcome curse of dimensionality, otherwise bounds are vacuous.
- Traditional ML: have bounds, using specific hypothesis classes. Mostly **linear** functions in a high dimensional space - bound independent of dimension.
- Deep learning: missing such bounds.
- Generalization Theory somewhat disjoint from Optimization

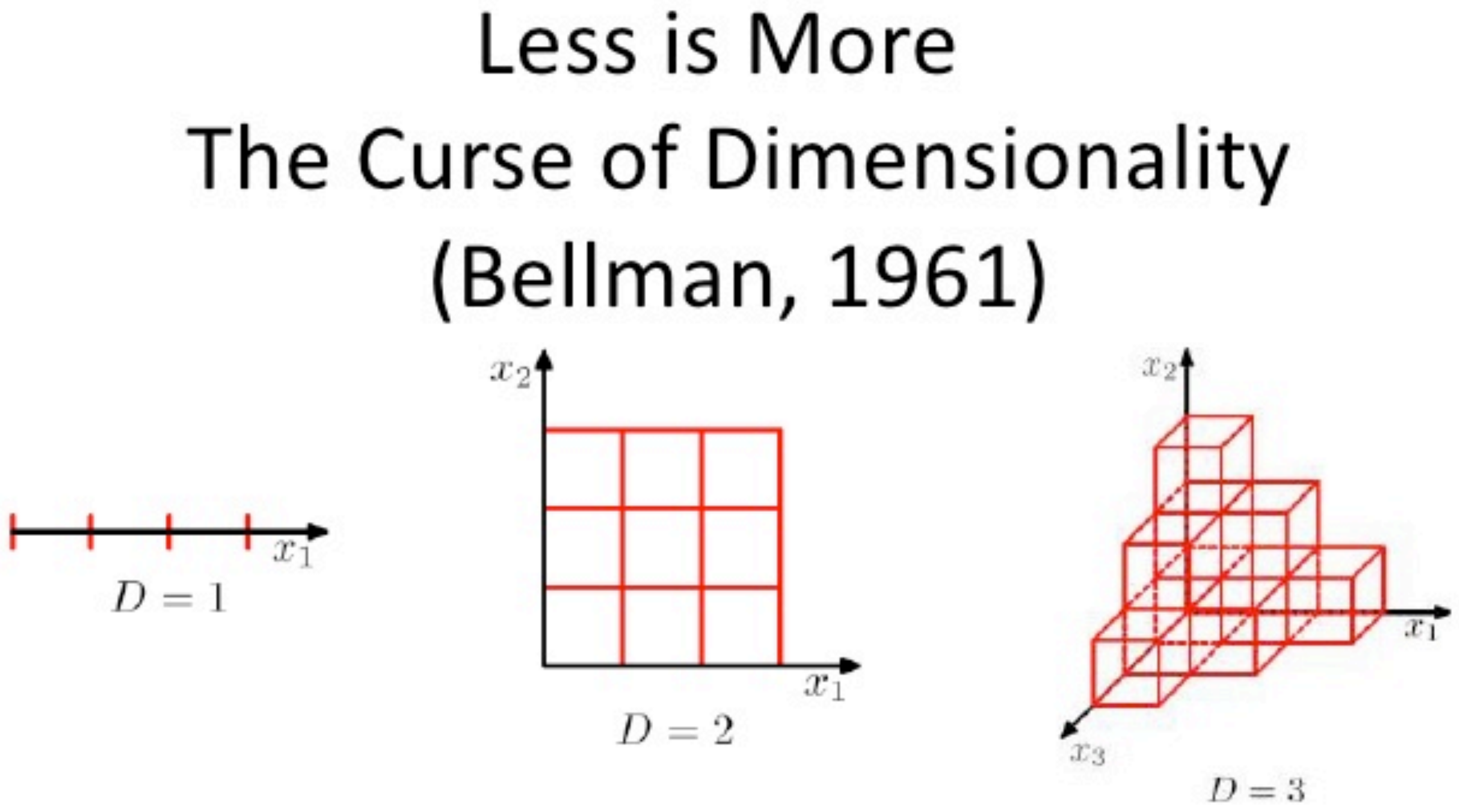


Curse of dimensionality



3.2. **Curse of dimensionality.** Mathematical approximation theory [Che66] allows us to prove convergence of approximations $f^m \to f$ with rates which depend on the error of approximation and on the typical distance between a sampled point and a given data point, h.

For uniform sampling of the box $[0,1]^d$ with m points, we have $h = m^{1/d}$. *curse of dimensionality*: the number of points required to achieve a given error grows exponentially in the dimension. Vacuous bounds.



Less is More

The Curse of Dimensionality

(Bellman, 1961)

Punch line: can't use traditional approximation theory in high dimensions. Need a theory to approximate (learn) functions with bounds which have complexity depends on **samples** but not **dimension**

Statistical Learning Theory + Regularization

- Goal : understand Statistical learning theory regularization ideas to obtain bounds for learning.
  - *Support Vector Machines* (SVM): linear hypothesis spaces
  - *Kernels*: linear hypothesis spaces, in high dimensional Hilbert Space (Feature Space) — expand function in a (particular) series.
  - *Theory*: Rademacher Complexity of regularized loss functions leads to dimension independent Learning bounds.
  - Regularization: norm of the function in Hilbert space.
- Complete, mathematical theory using concentration of measure.
- Deep Neural Networks: **no such theory.**

Concentrate to beat the curse

6.1. **Concentration of measure.** Consider the experiment of flipping a possibly biased coin. Let $X_k \in \{-1, 1\}$ represent the outcomes of the coin toss. After m coin tosses, let $S_m = \frac{1}{m}\sum_{k=1}^m X_k$ be the sample mean. The expected value of X, $\mu = \mathbb{E}[X]$ is the difference of the probabilities, $p_H - p_T$, which is zero when the coin is fair. In practice, due the randomness, the sample mean will deviate from the mean, and we expect the deviation to decrease as m increases. The Central Limit Theorem quantifies the deviation $\sqrt{m}(S_m - \mu)$ converges to the normal distribution as $m \to \infty$, so that,

$$|S_m - \mu| \approx \frac{1}{\sqrt{m}}$$

for m large. Concentration of measure inequalities provide non-asymptotic bounds. For example, Hoeffdings inequality [MRT18, Appendix D] applied to random variables taking values in $[-1, 1]$ gives

$$P\left(|S_m - \mu| > \epsilon\right) \leq 2\exp\left(-\frac{m\epsilon^2}{2}\right)$$

for any $\epsilon > 0$. Setting $\delta = 2\exp\left(-m\epsilon^2/2\right)$ and solving for ϵ allows us to restate the result as

$$|S_m - \mu| \leq \sqrt{\frac{2\log(2/\delta)}{m}} \quad \text{with probability } 1-\delta, \text{ for any } \delta > 0. \tag{6.1}$$

Learning Theory

6.2. **Statistical learning theory and generalization.** Statistical learning theory (see [MRT18, Chapter 3] and [SSBD14, Chapter 4]) can be used to obtain dimension independent sample complexity bounds for the expected loss (generalization) of a learning algorithm. These are bounds which depend on the number of samples, m but not on the dimension of the underlying data, n.

The *hypothesis space complexity* approach restricts the hypothesis space (2.3) to limit the ability of functions to overfit by bounding the the generalization gap $L^{\mathsf{gap}}[f] := L_{\mathcal{D}}[f] - L_S[f]$. The dependence of the generalization gap on the learning algorithm is removed by considering the worst-case gap for functions in the hypothesis space

$$L^{\mathsf{gap}}[f_{\mathcal{A}(S)}] \leq \sup_{f \in \mathcal{H}} L^{\mathsf{gap}}[f]$$

For example, [MRT18, Theorem 3.3] (which applies to the case of bounded loss function $0 \leq \mathcal{L} \leq 1$), states that for any $\delta > 0$

$$L^{\mathsf{gap}}[f_{\mathcal{A}(S)}] \leq \mathfrak{R}_m(\mathcal{H}) + \sqrt{\frac{\ln \frac{1}{\delta}}{2m}}, \qquad \text{with probability} \geq 1 - \delta \tag{6.2}$$

where $\mathfrak{R}(\mathcal{H})$ is the Rademacher complexity of the hypothesis space $\mathcal{H}$. Observe that (6.2) has a similar form to (6.1), with the additional term coming from the hypothesis space complexity. Thus, restricting to a low-complexity hypothesis space reduces *learning* bounds to *sampling* bounds.

Statistical Learning Theory

- Later: blackboard proof of some of this.

What’s missing: regularization

- Traditional ML Classification: have bounds, using specific hypothesis classes.
    - Mostly **linear** functions in a high dimensional space - bound independent of dimension.
- Deep learning: **non-linear** models
- What’s missing? Generalization theory doesn’t assume models are linear. Instead makes a *complexity assumption*
- Complexity theory makes fundamental use of regularization:
    - bound on weights for SVMs (linear models)
    - bound on Hilbert space function norms for Kernels Methods
- Hypothesis: adding regularization will allow for generalization for Deep Learning.

regularization in deep learning?

- does not exist yet…
- DL people naively add "weight decay" …
- which is equivalent to pretending that the neural network is a SVM, and adding weight regularization.
- … but it makes no difference.
- on the other hand:
    - a bunch of "hacks" which improve performance can be interpreted (but only by a minority) as regularization

Regularization DNN

Regularization in DNN practice.

- Tychonoff gradient regularization of the form (EL-R) directly: [DLC92]
- Data augmentation [LBB+98]: small rotations, cropping, or intensity or contrast adjustment.
- Dropout [SHK+14] consists of randomly, with small probability, changing some model weights to zero.
- Cutout [DT17] consists of randomly sending a smaller square of pixels in the image to mean
- Mixup [ZCDL17] consists of taking convex combinations of data points x_i, x_j and the corresponding labels y_i, y_j.
- Gaussian noise averaging has recently reappeared in deep neural networks [LAG+18, LCZH18, LCWC18, CRK19] as a method to certify a network to be robust to adversarial perturbations.

$$C_{smooth}(x) = \arg\max C(x+\eta), \qquad \eta = N(0, \sigma^2),$$

which requires many evaluations of the network.

DA = Implicit Regularization

10.1. **Image transformation and implicit regularization.** Consider an abstract data transformation,

$$x \mapsto T(x)$$

which transforms the image x. This could be data augmentation, random cutout, adding random gaussian noise to an image, or an adversarial perturbation. The data transformation replaces (EL) with the data augmented version

$$\min_{f \in \mathcal{H}} \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(f(T(x_i)), y_i) \tag{EL-A}$$

which we can rewrite as

$$\min_{f \in \mathcal{H}} L_{S_m}(f) + \mathcal{R}^T(f(x_i))$$

where

$$\mathcal{R}^T(f(x_i)) = \mathcal{L}(f(T(x_i)), y_i) - \mathcal{L}(f(x_i), y_i) \tag{RT}$$

Here the regularization is implicit, and the strength of the regularization is controlled by making the transformation T closer to the identity.

DA noise = Regularization

10.2. **Data augmentation.** Here show how adding noise leads to regularization, following [Bis95].

Lemma 10.3. *The augmented loss problem* (EL-A) *with quadratic loss and additive noise*

$$T(x) = x + v, \quad v \textit{ random}, \quad \mathbb{E}(v) = 0, \quad \mathbb{E}(v_i v_j) = \sigma^2 \tag{10.2}$$

is equivalent to the regularized loss problem (EL-R) *with*

$$\mathcal{R}^{noise}(f) = \frac{\sigma^2}{m} \sum_{i=1}^{m} \left(f_x^2 + (f - y) f_{xx} + \frac{\sigma^2}{4} f_{xx}^2 \right) \tag{10.3}$$

Proof. For clarity we treat the function as one dimensional. A similar calculation can be done in the higher dimensional case. Apply the Taylor expansion

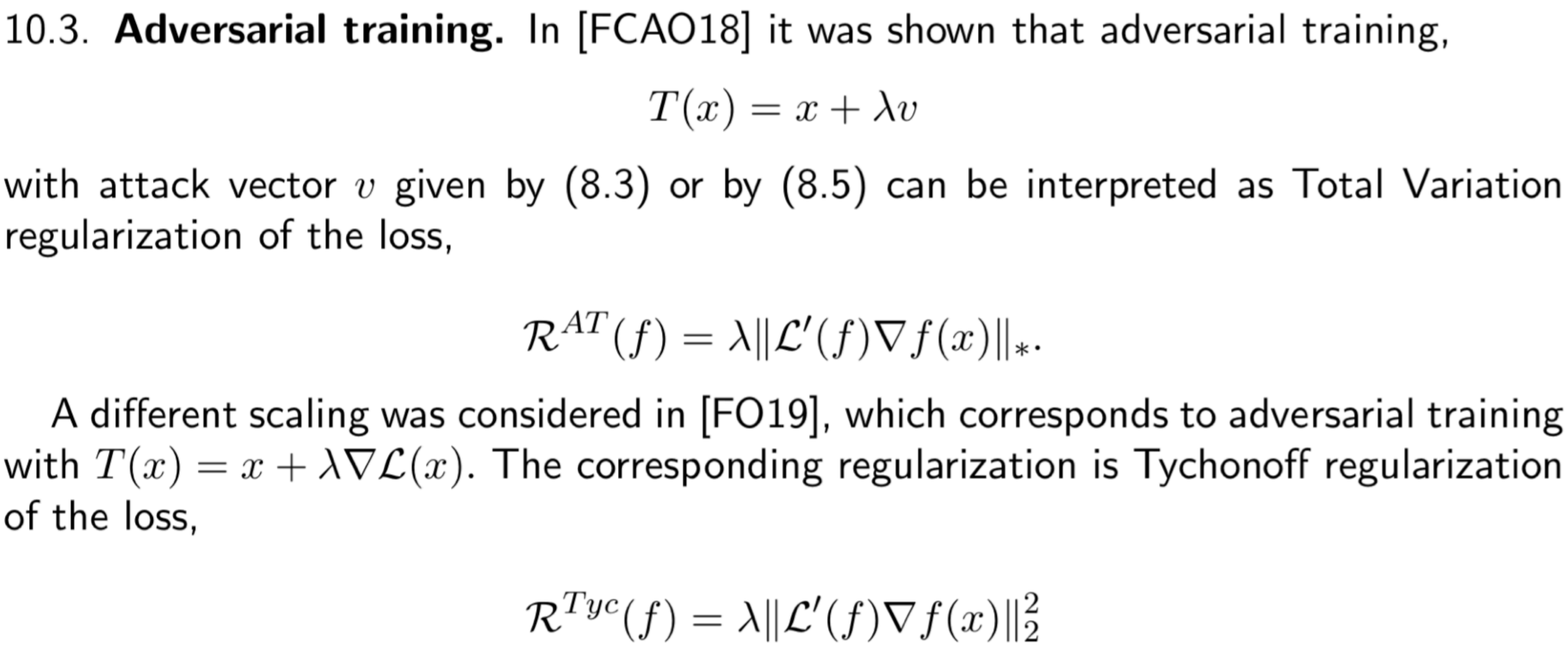
$$f(x+v) = f(x) + v f_x + \frac{1}{2} v^2 f_{xx} + O(v^3)$$

to the quadratic loss $\mathcal{L}(f, y) = (f(x+v) - y)^2$. Keeping only the lowest order terms, we have

$$(f(x+v) - y)^2 = (f(x) - y)^2 + 2(f_x v + \frac{1}{2} v^2 f_{xx})(f(x) - y) + (f_x v + \frac{1}{2} v^2 f_{xx})^2$$

Taking expectations and applying (10.2) to drop the terms with odd powers of v gives (10.3). □

AT = Regularization



10.3. **Adversarial training.** In [FCAO18] it was shown that adversarial training,

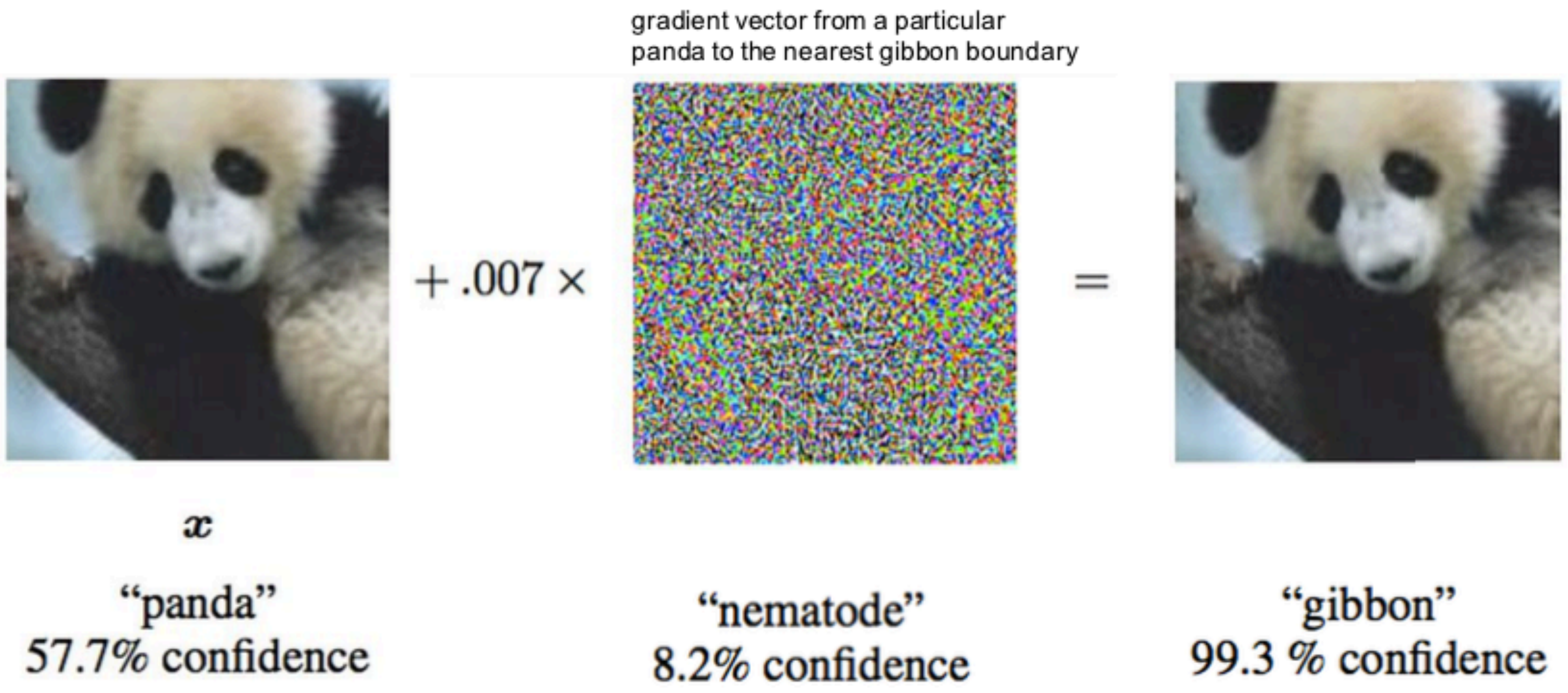
$$T(x) = x + \lambda v$$

with attack vector v given by (8.3) or by (8.5) can be interpreted as Total Variation regularization of the loss,

$$\mathcal{R}^{AT}(f) = \lambda \|\mathcal{L}'(f)\nabla f(x)\|_*.$$

A different scaling was considered in [FO19], which corresponds to adversarial training with $T(x) = x + \lambda\nabla\mathcal{L}(x)$. The corresponding regularization is Tychonoff regularization of the loss,

$$\mathcal{R}^{Tyc}(f) = \lambda \|\mathcal{L}'(f)\nabla f(x)\|_2^2$$



gradient vector from a particular panda to the nearest gibbon boundary

regularization in deep learning?

- does not exist yet…
- DL people naively add "weight decay" …
- which is equivalent to pretending that the neural network is a SVM, and adding weight regularization.
- … but it makes no difference.
- on the other hand:
    - a bunch of "hacks" which improve performance can be interpreted (but only by a minority) as regularization

Acceleration and SGD

IPAM tutorials week

Adam Oberman

Math and Stats, McGill

Machine Learning Problem: Supervised Classification

Let $x \in \mathcal{X} \subset [0,1]^d$ and the dimension is large, $d \gg 1$.
Labels $\mathcal{Y} = \{1, \ldots, K\}$,
The dataset

$$S_m = \{(x_1, y_1), \ldots, (x_m, y_m)\} \tag{2.1}$$

consists of m samples, x_i, drawn i.i.d. from a data distribution, $\rho(x)$, with support $\mathcal{X}$
The labels $y_i \in \mathcal{Y}$ are also given.

$\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$ is a loss function if it is zero iff $y_1 = y_2$.
Our objective is to find a function $f : \mathcal{X} \to \mathcal{Y}$ which minimizes the expected loss

$$L_{\mathcal{D}}(f) = \mathbb{E}_{x \sim \rho}[\mathcal{L}(f(x), y(x))] = \int_{\mathcal{X}} \mathcal{L}(f(x), y(x)) d\rho(x) \tag{2.2}$$

Intractable. Instead minimize the empirical loss

$$L_{S_m}[f] = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(f(x_i), y_i). \tag{EL}$$

Parametric Hypothesis Class

Typically, the functions considered are restricted to a parametric class

(2.3)
$$\mathcal{H} = \{f(x, w) \mid w \in \mathbb{R}^D\}$$

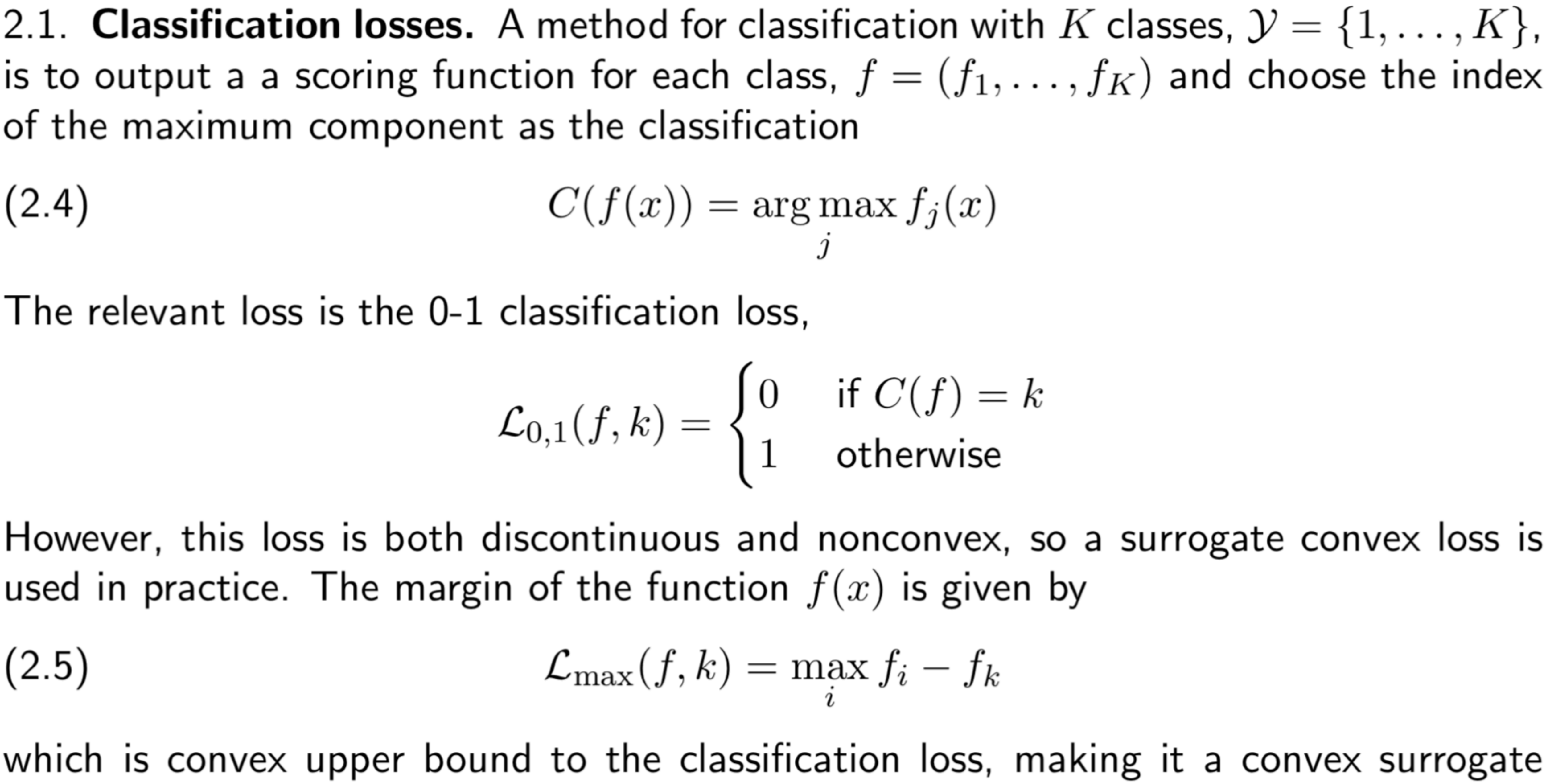
so that minimization of (EL) can be rewritten as the finite dimensional optimization problem

(EL-W)
$$\min_w \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(f(x_i, w), y_i).$$

The problem with using (EL-W) as a surrogate for (2.2), is that a minimizer of (EL-W) could *overfit*, meaning that the expected loss is much larger than the empirical loss. Classical machine learning methods avoid overfitting by restricting $\mathcal{H}$ to be a class of simple (e.g. linear) functions.

Remark 2.1. To first approximation, we can assume that the images are free of noise, that all labels are correct, and that there are no ambiguous images. In other words, $y_i = y(x_i)$ for a label function $y(x)$. Challenge in learning $y(x)$ comes not from uncertainty, noise, or ambiguity, but rather from the complexity of the functions involved.

Losses



2.1. **Classification losses.** A method for classification with K classes, $\mathcal{Y} = \{1, \ldots, K\}$, is to output a a scoring function for each class, $f = (f_1, \ldots, f_K)$ and choose the index of the maximum component as the classification

$$C(f(x)) = \arg\max_j f_j(x) \tag{2.4}$$

The relevant loss is the 0-1 classification loss,

$$\mathcal{L}_{0,1}(f, k) = \begin{cases} 0 & \text{if } C(f) = k \\ 1 & \text{otherwise} \end{cases}$$

However, this loss is both discontinuous and nonconvex, so a surrogate convex loss is used in practice. The margin of the function $f(x)$ is given by

$$\mathcal{L}_{\max}(f, k) = \max_i f_i - f_k \tag{2.5}$$

which is convex upper bound to the classification loss, making it a convex surrogate

Analyze the 0-1 loss. *Train* with convex surrogate loss

Losses

Analyze the 0-1 loss. *Train* with convex surrogate loss

SGD - Algorithm

5.2. **Stochastic gradient descent.** Evaluating the loss (EL) on all m data points can be costly. Define random minibatch $I \subset \{1, \dots, m\}$, and define the corresponding minibatch loss by

$$L_I(w) = \frac{1}{|I|} \sum_{i \in I} \mathcal{L}(f(x_i, w), y_i)$$

Stochastic gradient descent corresponds to

$$w^{k+1} = w^k - h_k \nabla_w L_{I_k}(w^k), \quad I_k \text{ random, } h_k \text{ learning rate}$$

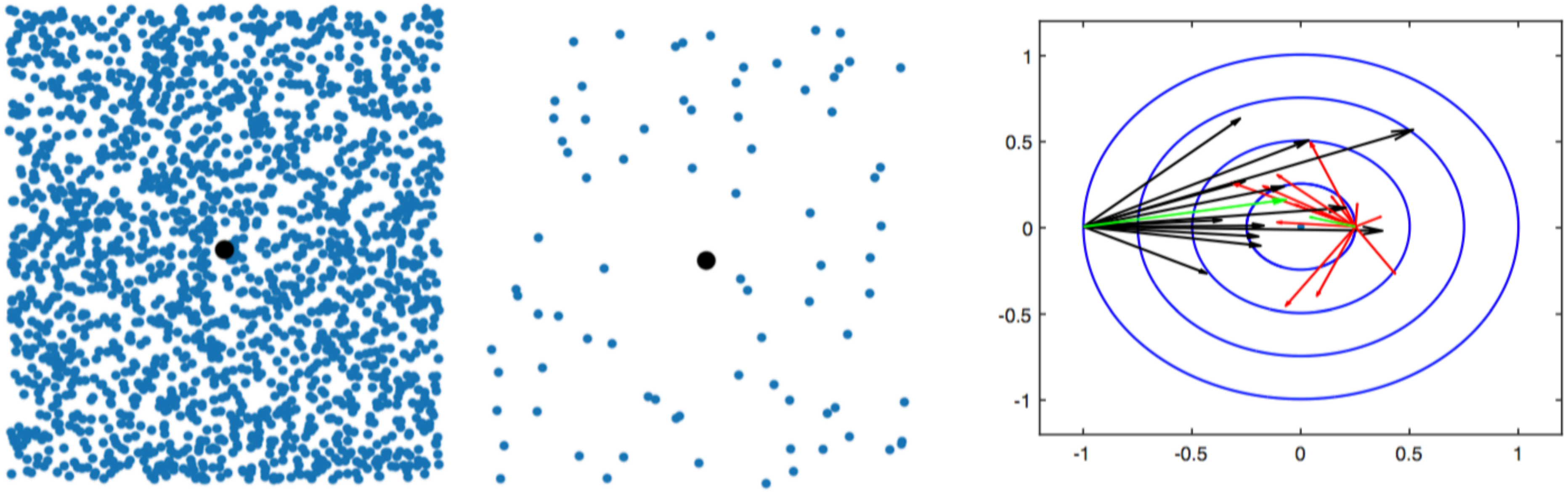


FIGURE 2. Full data set and a minibatch for data sampled uniformly in the square. Component gradients (black) and the minibatch gradient (green). Closer to the minimum, the relative error in the minibatch gradient is larger.

5.3. **The convergence rate of SGD.** See [BCN16] for a survey on results on SGD. The following simple result was proved in [OP19]. Suppose f is μ-strongly convex and L-smooth, with minimum at w^*. Let $q(w) = \|w - w^*\|^2$. Write

$$\nabla_{mb} f(w) = \nabla f(w) + e,$$

with e is a mean zero random error term, with variance σ^2. Consider the stochastic gradient descent iteration

$$w_{k+1} = w_k - h_k \nabla_{mb} f(w_k), \tag{SGD}$$

with learning rate

$$h_k = \frac{1}{\mu(k + q_0^{-1}\alpha_S^{-1})}, \tag{SLR}$$

Theorem 5.2. *Let $w_k,\ h_k$ be the sequence given by* (SGD) (SLR)*. Then,*

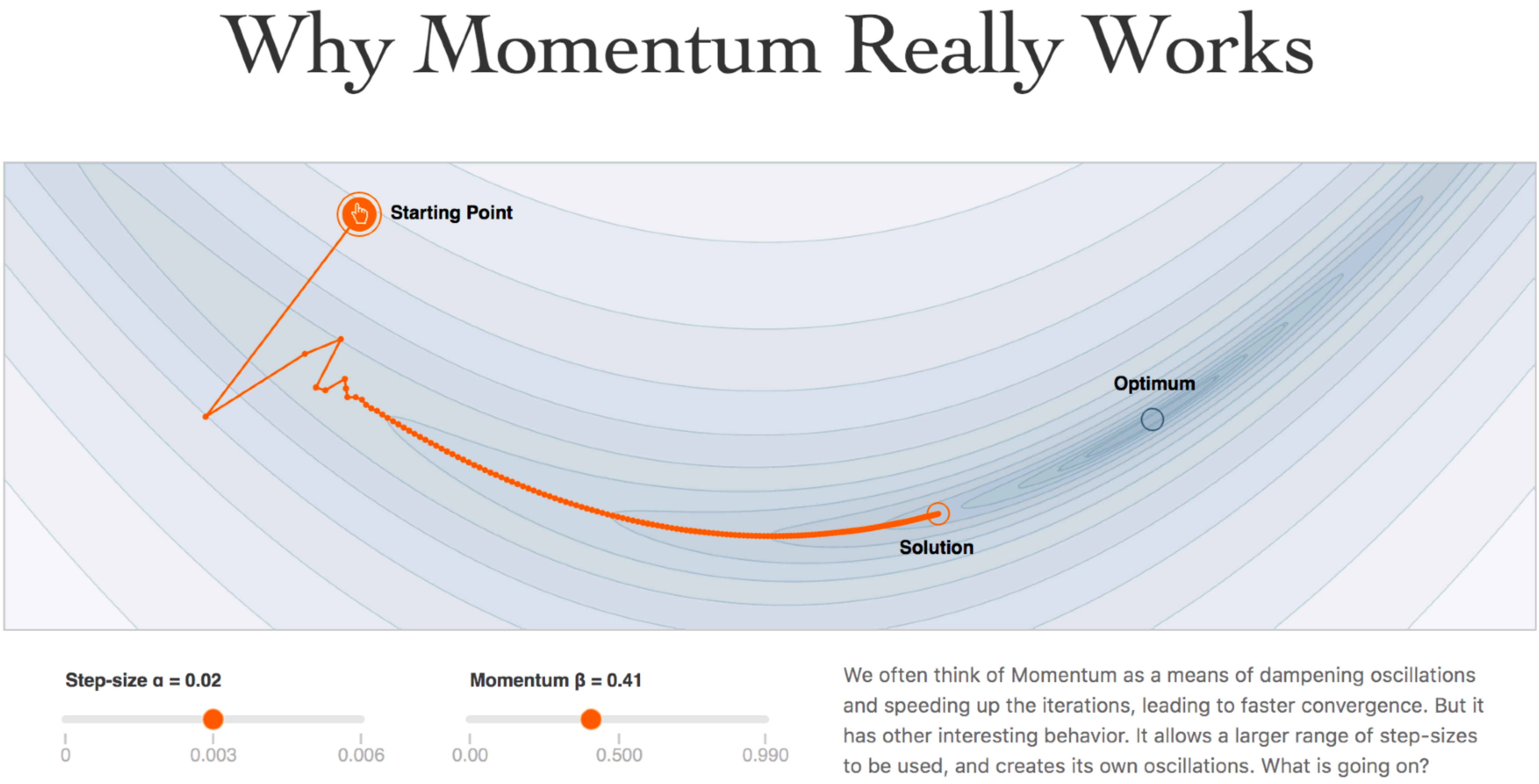
$$\mathbb{E}\left[q_k \mid w_{k-1}\right] \leq \frac{1}{\alpha_S k + q_0^{-1}}, \text{ *for all* } k \geq 0.$$

The convergence rate of SGD is slow: the error decreases on the order of $1/k$ for strongly convex problems. This means that if it takes 1000 iterations to reach an error of ϵ, it may take ten times as many iterations to further decrease the error to $\epsilon/10$.

SGD rate

blackboard proof

5.4. **Accelerated SGD.** In practice, better empirical results are achieved using the accelerated version of SGD. This algorithm is the stochastic version of Nesterov's accelerated gradient descent [Nes13]. See [Goh17] for an exposition on Nesterov's method. Nesterov's method can be interpreted as the discretization of a second order ODE [SBC14]. In [LO19] we show how the continuous time interpretation of Nesterov's method with stochastic gradients leads to accelerated convergence rates for Nesterov's SGD, using a Liapunov function analysis similar to the one described above.



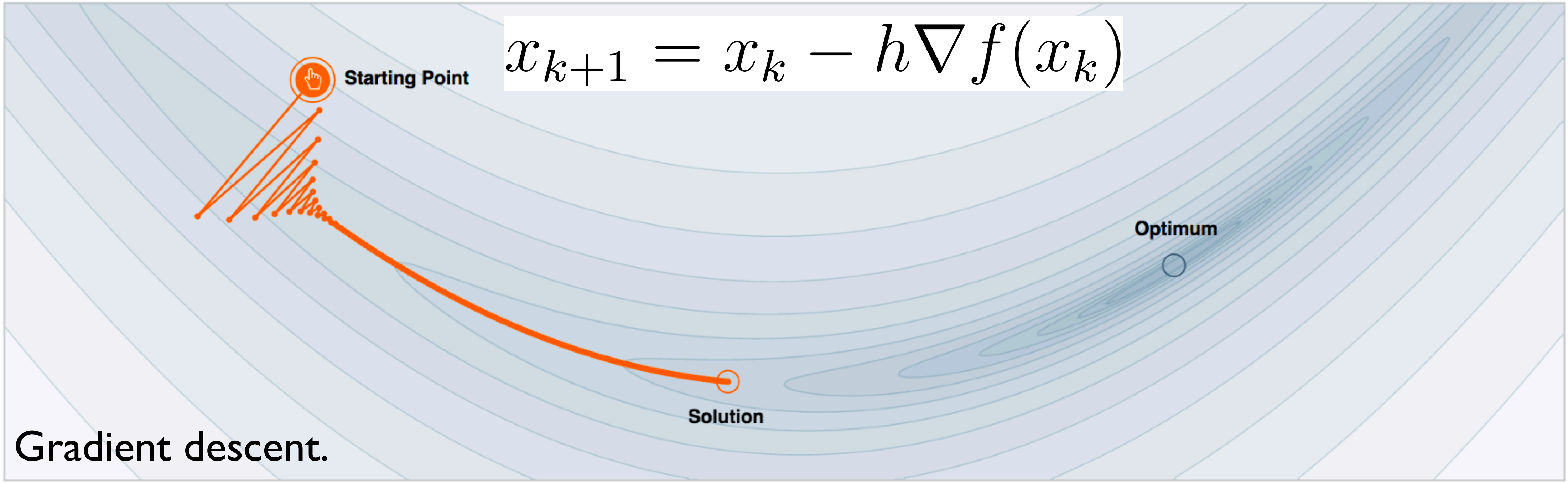
Why Momentum Really Works

Step-size α = 0.02

Momentum β = 0.41

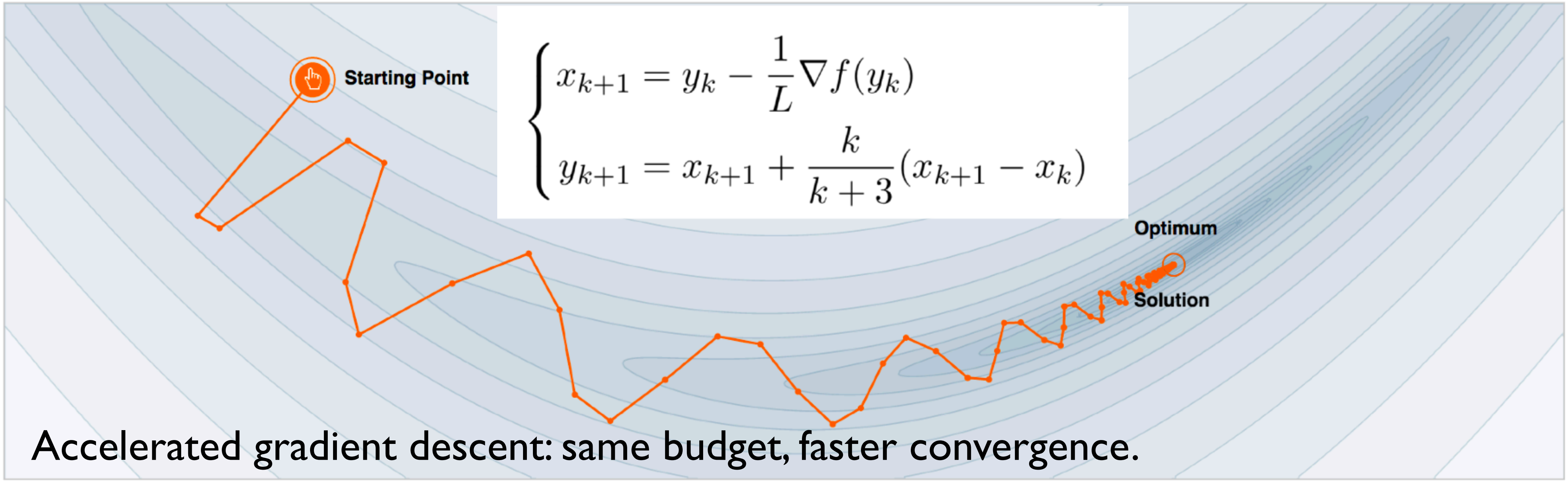
We often think of Momentum as a means of dampening oscillations and speeding up the iterations, leading to faster convergence. But it has other interesting behavior. It allows a larger range of step-sizes to be used, and creates its own oscillations. What is going on?

Convex case: Nesterov's accelerated gradient descent



$$x_{k+1} = x_k - h\nabla f(x_k)$$

Gradient descent.



$$\begin{cases} x_{k+1} = y_k - \dfrac{1}{L}\nabla f(y_k) \\ y_{k+1} = x_{k+1} + \dfrac{k}{k+3}(x_{k+1} - x_k) \end{cases}$$

Accelerated gradient descent: same budget, faster convergence.

https://distill.pub/2017/momentum/

Heuristic: momentum term remembers old gradients, overshoots instead of getting stuck.

Remark: *two main A-GD algorithms, correspond to convex and strongly convex case. We focus on one, convex case, to simplify presentation. Strongly convex case is also covered.*

Convex case: Nesterov's accelerated gradient descent

Gradient descent.

$$x_{k+1} = x_k - h\nabla f(x_k)$$

Accelerated gradient descent:
add a "momentum" variable.
Same budget, faster convergence.

$$\begin{cases} x_{k+1} = y_k - \dfrac{1}{L}\nabla f(y_k) \\ y_{k+1} = x_{k+1} + \dfrac{k}{k+3}(x_{k+1} - x_k) \end{cases}$$

2-step algorithm, similar to GD.

Convex case:

GD: convergence rate is 1/k

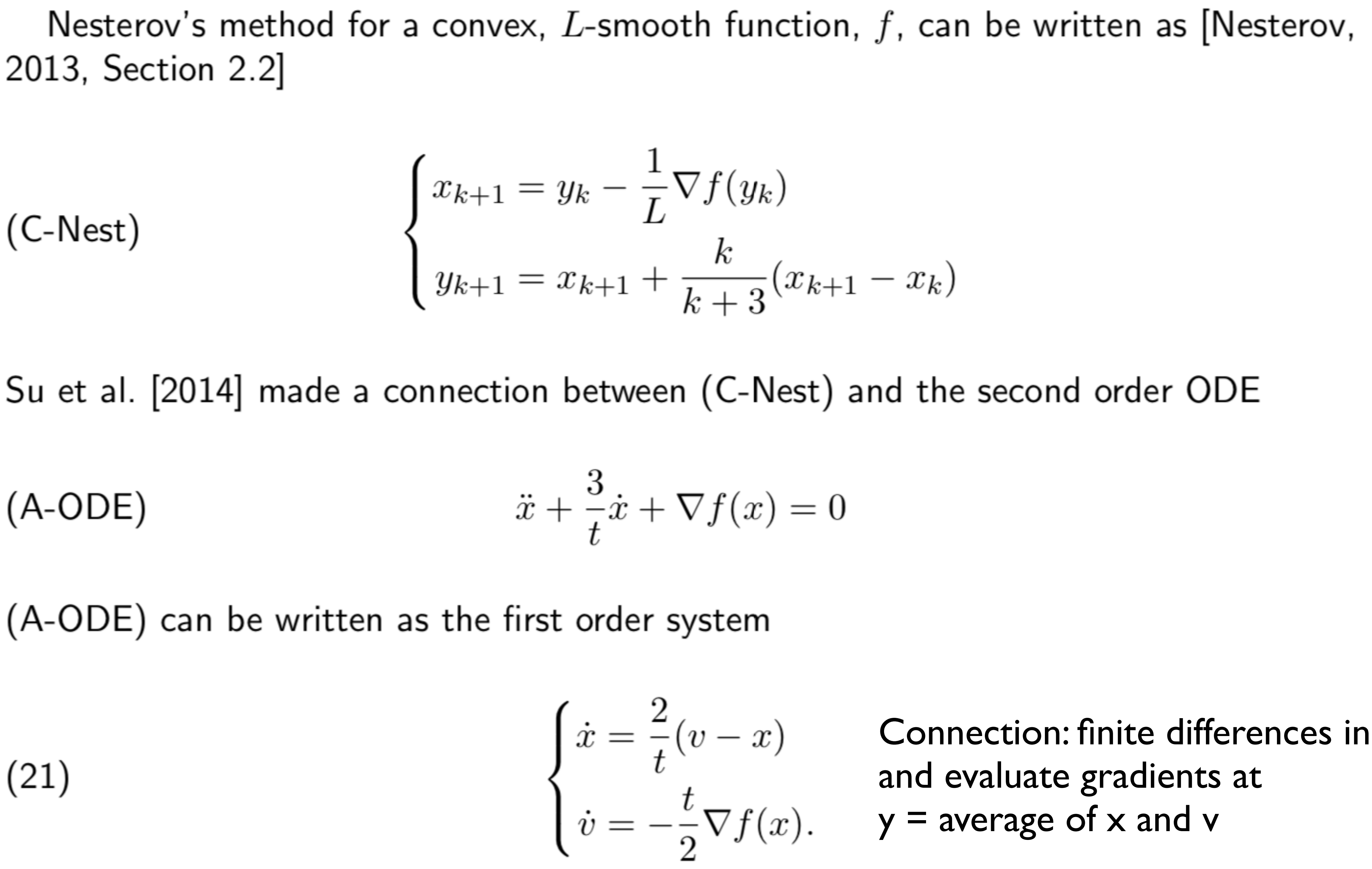
AGD: convergence rate is 1/k^2.

"acceleration": after 1000 steps, 1e-6 versus 1e-3.

How do we understand this algorithm?
Nesterov's proof - not clear.
Instead, Jordan, and Su-Boyd-Candes propose an ODE interpretation

ODE interpretation of Nesterov's Method

Su-Boyd-Candes and Jordan/Wilson-Wibisino suggested:
Use continuous time ideas to understand A-GD, and develop new algorithms.



Nesterov's method for a convex, L-smooth function, f, can be written as [Nesterov, 2013, Section 2.2]

$$\text{(C-Nest)} \qquad \begin{cases} x_{k+1} = y_k - \frac{1}{L}\nabla f(y_k) \\ y_{k+1} = x_{k+1} + \frac{k}{k+3}(x_{k+1} - x_k) \end{cases}$$

Su et al. [2014] made a connection between (C-Nest) and the second order ODE

$$\text{(A-ODE)} \qquad \ddot{x} + \frac{3}{t}\dot{x} + \nabla f(x) = 0$$

(A-ODE) can be written as the first order system

$$\text{(21)} \qquad \begin{cases} \dot{x} = \frac{2}{t}(v - x) \\ \dot{v} = -\frac{t}{2}\nabla f(x). \end{cases}$$

Connection: finite differences in time, and evaluate gradients at y = average of x and v

Another ODE for Nesterov's Method

Our starting point is a perturbation of (21) Su-Boyd-Candes ODE

(1st-ODE)
$$\begin{cases} \dot{x} = \frac{2}{t}(v - x) - \frac{1}{\sqrt{L}}\nabla f(x) \\ \dot{v} = -\frac{t}{2}\nabla f(x), \end{cases}$$

The system (1st-ODE) is equivalent to the following ODE

(H-ODE)
$$\ddot{x} + \frac{3}{t}\dot{x} + \nabla f(x) = -\frac{1}{\sqrt{L}}\left(D^2 f(x) \cdot \dot{x} + \frac{1}{t}\nabla f(x)\right)$$

which has an additional Hessian damping term with coefficient $1/\sqrt{L}$.

- The system (1st-ODE) can be discretized to recover Nesterov's method using an explicit discretization with a constant time step $h = \frac{1}{\sqrt{L}}$,
- Similar ODEs studied by Alvarez et al. [2002],Attouch et al. [2016].
- Shi et al. [2018] introduced a family of high resolution second order ODEs which also lead to Nesterov's method.(H-ODE), special case.
- (1st-ODE) shorter, clearer proofs which generalize to the stochastic gradient case (which was not treated in Shi et al. [2018]).

From ODE to Nesterov

Definition 4.1. *Define the learning rate and total time,*

$$h_k > 0, t_k = \sum_{i=1}^{k} h_i.$$

The time discretization of (1st-ODE) *with gradients evaluated at*

$$y_k = \left(1 - \frac{2h_k}{t_k}\right) x_k + \frac{2h_k}{t_k} v_k.$$

is given by

$$\text{(FE-C)} \qquad \begin{aligned} x_{k+1} - x_k &= \frac{2h_k}{t_k}(v_k - x_k) - \frac{h_k}{\sqrt{L}} \nabla f(y_k), \\ v_{k+1} - v_k &= -\frac{h_k t_k}{2} \nabla f(y_k), \end{aligned}$$

Proposition 4.2. *The discretization of* (1st-ODE) *given by* (FE-C) *with* $h_k = h = 1/\sqrt{L}$ *and* $t_k = h(k+2)$ *is equivalent to the standard Nesterov's method* (C-Nest).

Fix learning rate, get A-GD.

So What?

Su-Boyd-Candes led to a lot of work on continuous time approach:

mainly re-do of proofs of convergence rate of A-GD.

But the proofs were longer, and not really more informative.

If you like continuous time, great, otherwise, so what?

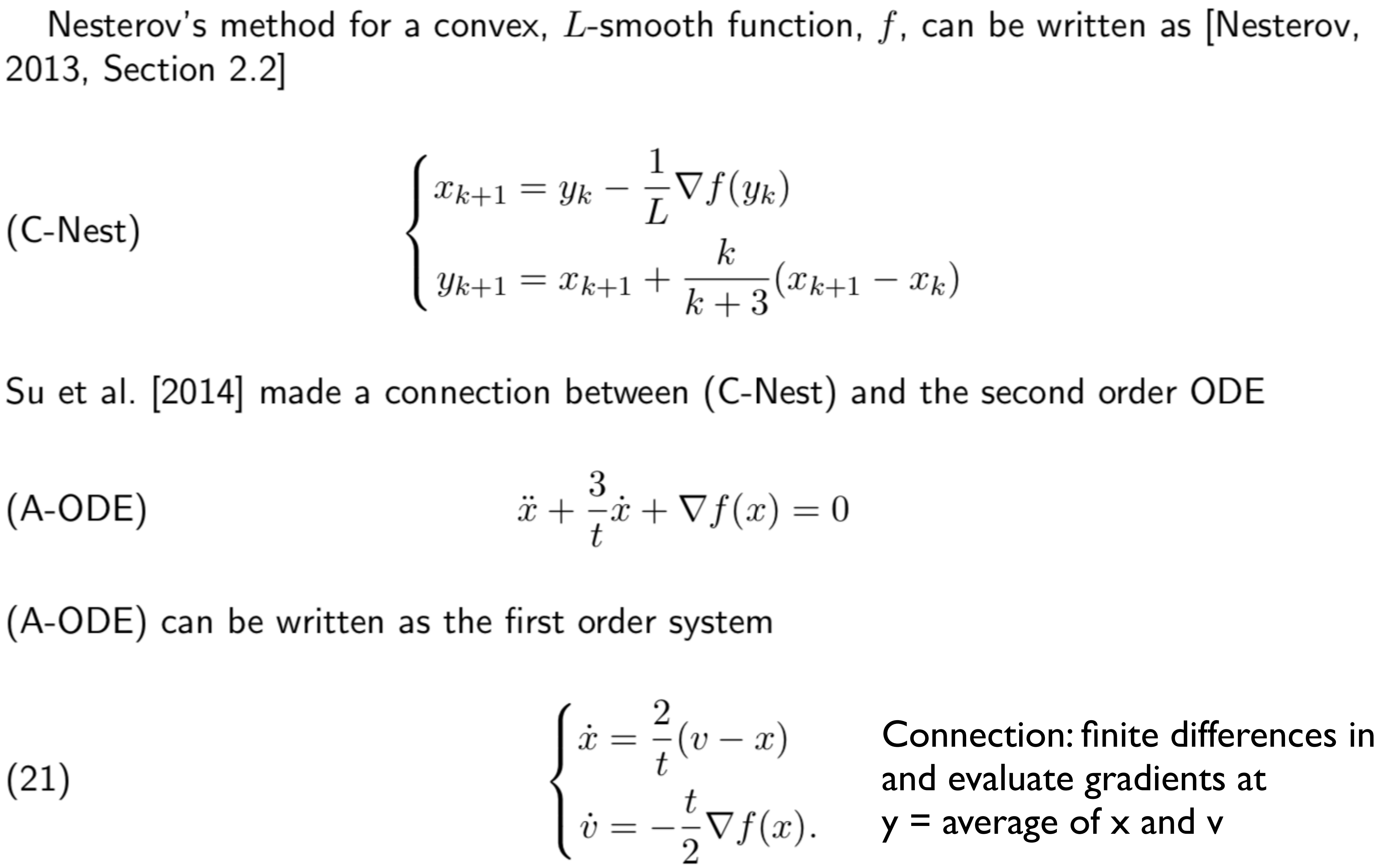
The real goal suggested by Jordan and collaborators was to build new algorithms, but so far has not happened.

Moreover, new work is on Stochastic gradient descent.

Can we get a faster SGD algorithm? This would have impact.

ODE interpretation of Nesterov's Method

Su-Boyd-Candes and Jordan/Wilson-Wibisino suggested:
Use continuous time ideas to understand A-GD, and develop new algorithms.



Nesterov's method for a convex, L-smooth function, f, can be written as [Nesterov, 2013, Section 2.2]

$$\text{(C-Nest)} \qquad \begin{cases} x_{k+1} = y_k - \frac{1}{L}\nabla f(y_k) \\ y_{k+1} = x_{k+1} + \frac{k}{k+3}(x_{k+1} - x_k) \end{cases}$$

Su et al. [2014] made a connection between (C-Nest) and the second order ODE

$$\text{(A-ODE)} \qquad \ddot{x} + \frac{3}{t}\dot{x} + \nabla f(x) = 0$$

(A-ODE) can be written as the first order system

$$\text{(21)} \qquad \begin{cases} \dot{x} = \frac{2}{t}(v - x) \\ \dot{v} = -\frac{t}{2}\nabla f(x). \end{cases}$$

Connection: finite differences in time, and evaluate gradients at y = average of x and v

Another ODE for Nesterov's Method

Our starting point is a perturbation of (21) Su-Boyd-Candes ODE

(1st-ODE)
$$\begin{cases} \dot{x} = \frac{2}{t}(v - x) - \frac{1}{\sqrt{L}}\nabla f(x) \\ \dot{v} = -\frac{t}{2}\nabla f(x), \end{cases}$$

The system (1st-ODE) is equivalent to the following ODE

(H-ODE)
$$\ddot{x} + \frac{3}{t}\dot{x} + \nabla f(x) = -\frac{1}{\sqrt{L}}\left(D^2 f(x) \cdot \dot{x} + \frac{1}{t}\nabla f(x)\right)$$

which has an additional Hessian damping term with coefficient $1/\sqrt{L}$.

- The system (1st-ODE) can be discretized to recover Nesterov's method using an explicit discretization with a constant time step $h = \frac{1}{\sqrt{L}}$,
- Similar ODEs studied by Alvarez et al. [2002],Attouch et al. [2016].
- Shi et al. [2018] introduced a family of high resolution second order ODEs which also lead to Nesterov's method.(H-ODE), special case.
- (1st-ODE) shorter, clearer proofs which generalize to the stochastic gradient case (which was not treated in Shi et al. [2018]).

From ODE to Nesterov

Definition 4.1. *Define the learning rate and total time,*

$$h_k > 0, t_k = \sum_{i=1}^{k} h_i.$$

The time discretization of (1st-ODE) *with gradients evaluated at*

$$y_k = \left(1 - \frac{2h_k}{t_k}\right) x_k + \frac{2h_k}{t_k} v_k.$$

is given by

$$\text{(FE-C)} \qquad \begin{aligned} x_{k+1} - x_k &= \frac{2h_k}{t_k}(v_k - x_k) - \frac{h_k}{\sqrt{L}} \nabla f(y_k), \\ v_{k+1} - v_k &= -\frac{h_k t_k}{2} \nabla f(y_k), \end{aligned}$$

Proposition 4.2. *The discretization of* (1st-ODE) *given by* (FE-C) *with* $h_k = h = 1/\sqrt{L}$ *and* $t_k = h(k+2)$ *is equivalent to the standard Nesterov's method* (C-Nest).

Fix learning rate, get A-GD.

So What?

Su-Boyd-Candes led to a lot of work on continuous time approach:

mainly re-do of proofs of convergence rate of A-GD.

But the proofs were longer, and not really more informative.

If you like continuous time, great, otherwise, so what?

The real goal suggested by Jordan and collaborators was to build new algorithms, but so far has not happened.

Moreover, new work is on Stochastic gradient descent.

Can we get a faster SGD algorithm? This would have impact.

Stochastic Gradient descent

Goal: minimize convex function f(x) with ***stochastic*** gradient oracle.

Algorithm: Stochastic Gradient descent. Simple, but more params.

3+ Parameters:

- learning rate: e.g: h(k) = h/k,
- stochastic variance

$$x_{k+1} = x_k - h_k \widetilde{\nabla} f(x_k)$$

$$\widetilde{\nabla} f(x) = \nabla f(x) + e(x, \xi)$$

$$E(e) = 0, \quad Var(e) = \sigma^2$$

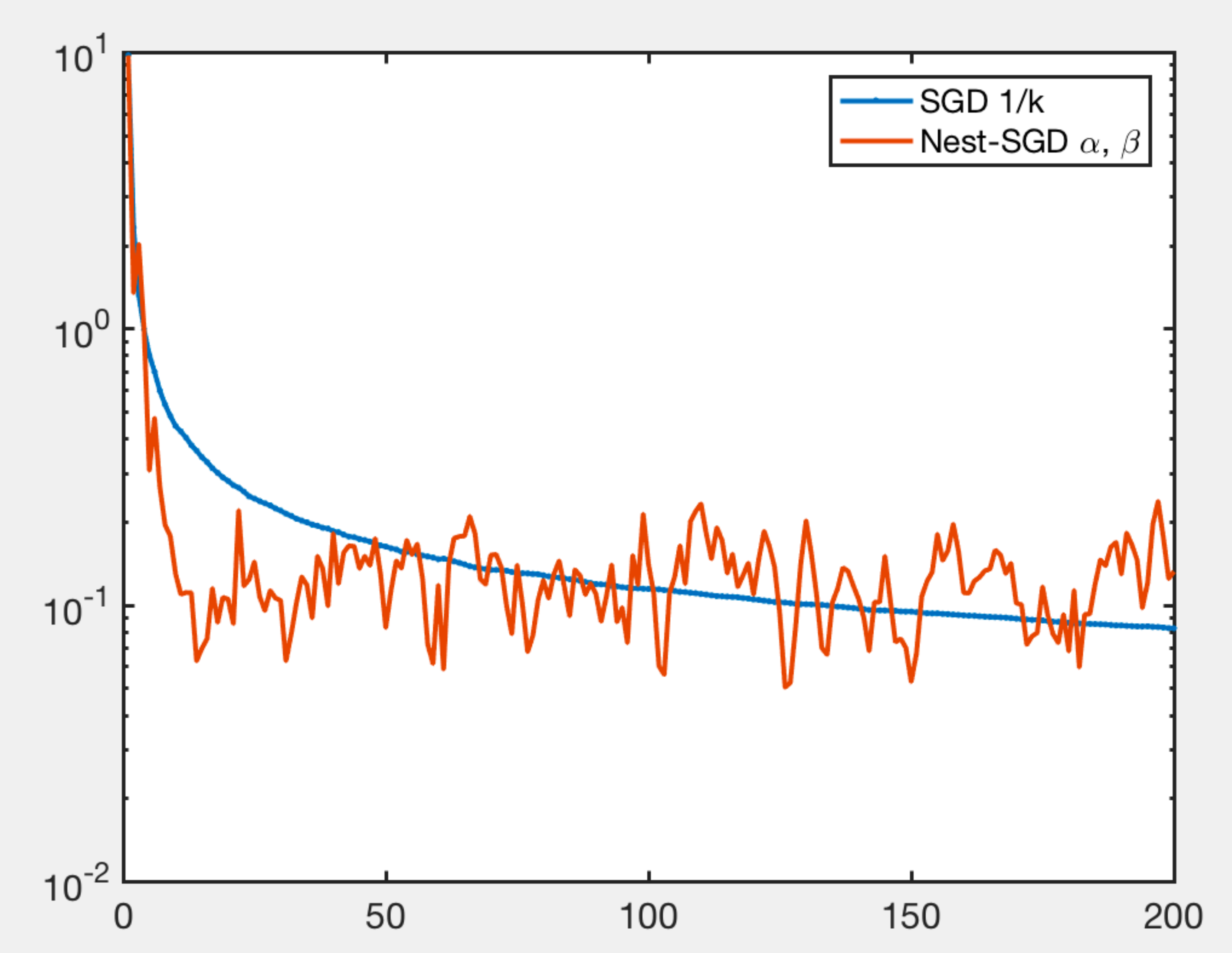
Heuristic accelerated SGD

Use accelerated gradient descent algorithm, with

- stochastic gradients
- constant learning rate.

Works well in practice: fast initial drop, then noise takes over.

Scheduled learning rate for SGD, converges at 1/k rate



Using Strongly convex Nesterov method (fixed learning rate and momentum)

What could you prove? Vanilla SGD converges at rate 1/k. Have upper bounds.
Only room for improvement is the rate constant. Details follow.

Accelerated SGD?

- Go back to ODE for Nesterov (there is more than one)
- Redo convergence rate, first in continuous, then in discrete time.
- *Find a variable learning rate, stochastic version*
- Prove the convergence rate.
- Use proof to tune the learning rate schedule, for optimal convergence rate
- Determine if we get a practical algorithm.
- There are two algorithms: convex version, and strongly convex version.
- Spoiler: convex version fast, easy to tune. Strongly convex version faster, but harder to tune.

Why continuous time? *Matter of taste, in previous work can do all proofs without it.*
In our case, we could not have proposed the algorithm without continuous time approach:
it eliminates the 2 time parameters in algorithm. Do proof in continuous time, then extend to algorithm.

Stochastic gradient version

$$\begin{cases} x_{k+1} - x_k = \dfrac{2h_k}{t_k}(v_k - x_k) - \dfrac{h_k}{\sqrt{L}}(\nabla f(y_k) + e_k), \\ v_{k+1} - v_k = -h_k \dfrac{t_k}{2}(\nabla f(y_k) + e_k), \end{cases}$$

$$h_k := \frac{c}{k^\alpha} \leq \frac{1}{\sqrt{L}} \text{ and } t_k = \textstyle\sum_{i=1}^k h_i, \quad \alpha = 3/4$$

Same algorithm as before, but now

- variable learning rate.
- stochastic gradient
- optimal exponent for learning rate: 3/4
- determined by convergence rate analysis, below

Acc-SGD results: synthetic noise

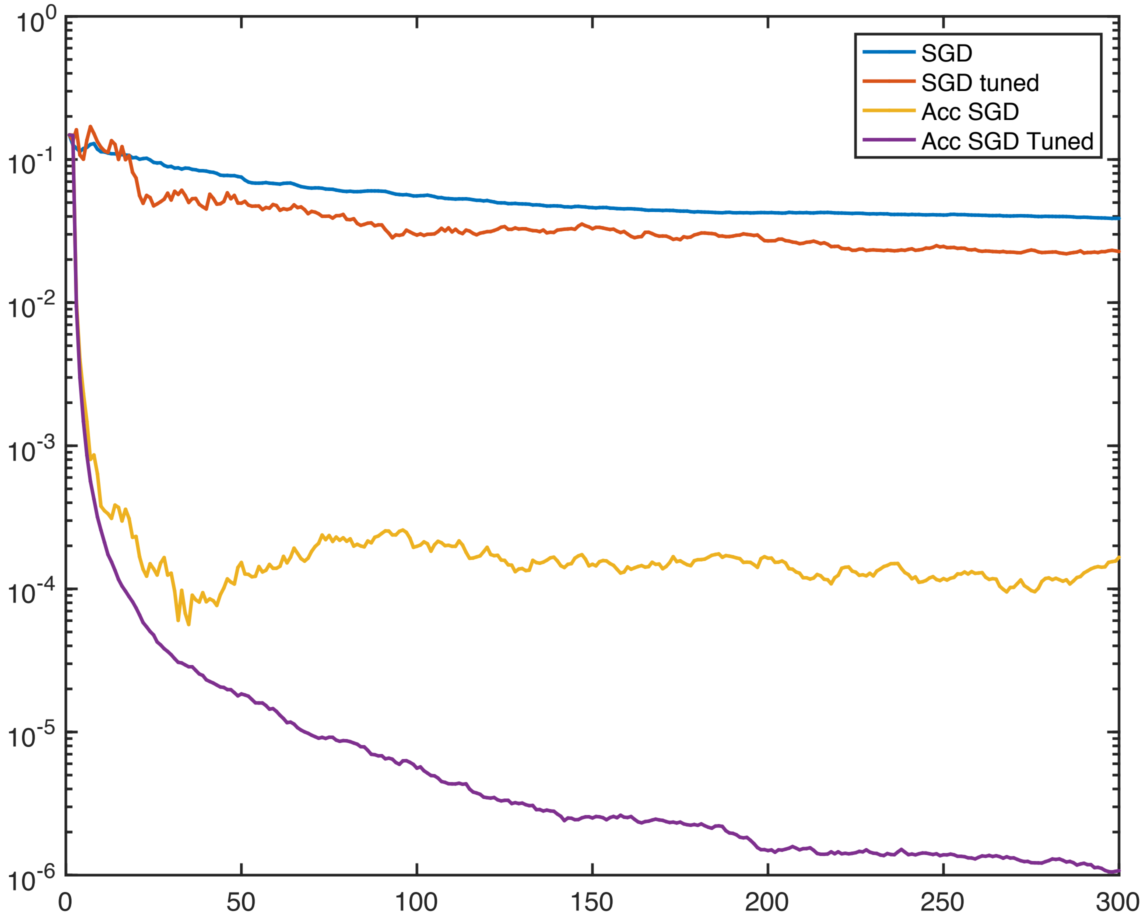
Simple quadratic example, synthetic noise.
Tuned version just means optimize the learning rate.

SGD (with 1/k schedule)

- guess/tune learning rate
- Results at k=300: 1e-1, 5e-2

A-SGD:

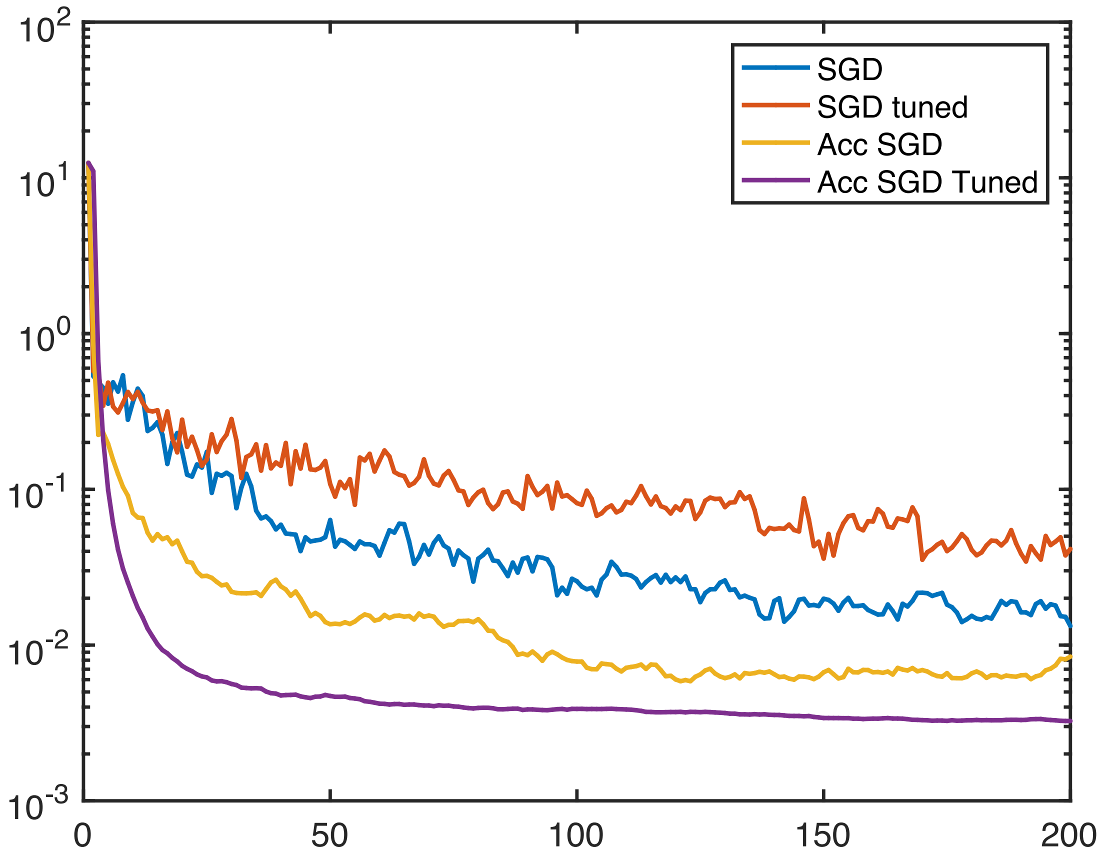
- guess/tune parameter, C
- Results at k=300: 1e-4,1e-6



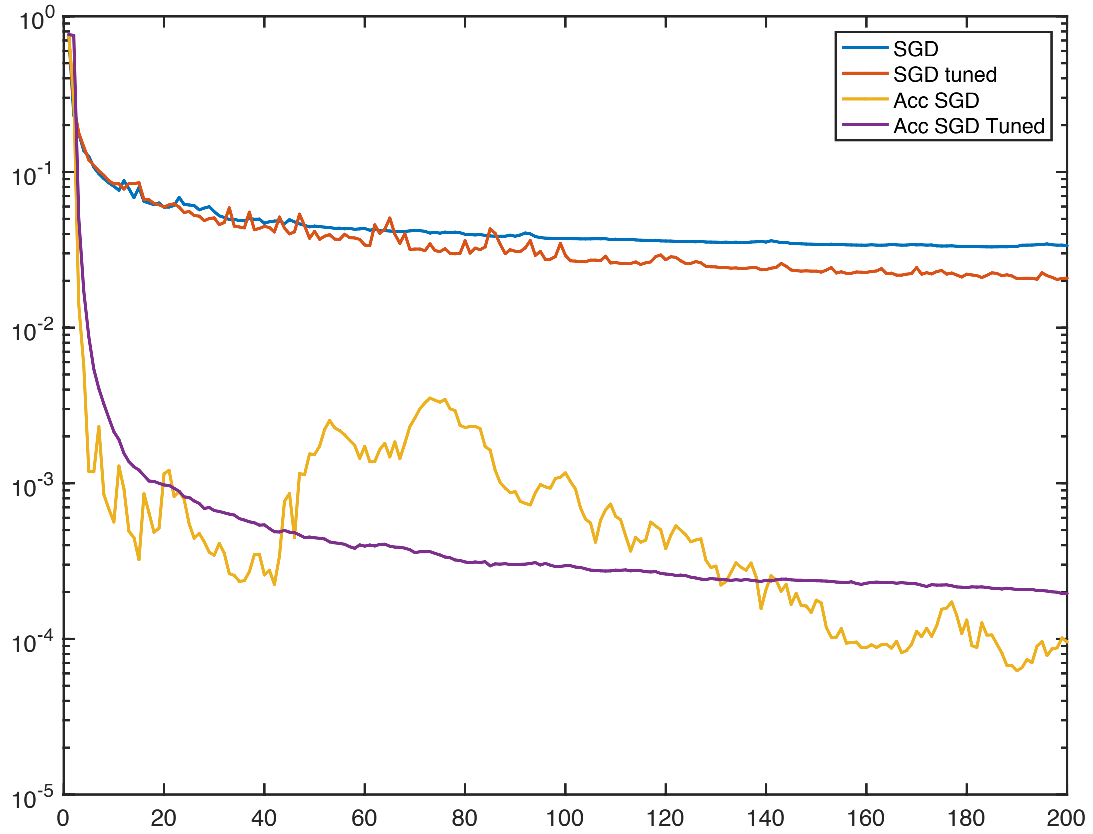
Quadratic function, C = 300

Acc-SGD results:
mini-batch SGD

- Faster convergence than SGD
- Improved performance on poorly conditioned examples



Condition number C = 1
approximately 10 x accuracy



Condition number C = 100
approximately 100 x accuracy

Math Outline

- Start with one of three ODEs
  - Gradient descent ODE
  - Convex accelerated GD ODE
  - Strongly convex, accelerated GD ODE
- Full gradient, and constant learning rate discretization leads to
  - gradient descent
  - convex Nesterov's method
  - strongly convex Nesterov's Method
- Stochastic gradient, and variable learning rate discretization leads:
  - accelerated SGD, with better constants than previously available, in both convex and strongly convex cases

Convergence Results

$$\widehat{g}(x, \xi) = \nabla f(x) + e(x, \xi),$$

$$\mathbb{E}[e] = 0 \text{ and } \operatorname{Var}(e) = \sigma^2.$$

G^2 bound on stochastic gradient: $\mathbb{E}[\widehat{g}^2] \leq G^2$

Convex case:

- the optimal rate for the last iterate of SGD (see Shamir and Zhang [2013]) is order $\log(k)/\sqrt{k}$ with a rate constant that depends on G^2.
- Jain et al. [2019] remove the log factor assuming that the number of iterations is decided in advance.
- We obtain the $\mathcal{O}(\log(k)/\sqrt{k})$ rate for the last iterate, with a constant which depends on σ, but is independent of the L-smoothness.

Strongly convex case:

- previous results, Nemirovski et al. [2009], Shamir and Zhang [2013], Jain et al. [2018], rate depends on G.
- obtain the optimal $\mathcal{O}(1/k)$ rate for the last iterate, with constants independent of the L-smoothness bound of the gradient.

Convergence Results

TABLE 1. Convergence rate $\mathbb{E}[f(x_k) - f^*]$ after k steps. for f a convex, L-smooth function. G^2 is a bound on $\mathbb{E}[\tilde{\nabla} f(x)^2]$, and σ^2 given by (2). h_k is the learning rate. E_0 is the initial value of the Lyapunov function. Top: convex case, D is the diameter of the domain, c is a free parameter. Bottom: μ-strongly convex case, $C_f := \frac{L}{\mu}$ $h_k = \mathcal{O}(1/k)$.

convex case

	Shamir and Zhang [2013]	Acc. SGD
h_k	$\dfrac{c^2}{\sqrt{k}}$	$\dfrac{c}{k^{3/4}}$
Rate	$\left(\dfrac{D^2}{c^2} + c^2G^2\right)\dfrac{(2+\log(k))}{\sqrt{k}}$	$\dfrac{\frac{E_0}{16c^2} + c^2\sigma^2(1+\log(k))}{(k^{1/4}-1)^2}$

Nemirovski et al. [2009]	Shamir and Zhang [2013]	Jain et al. [2019]	Acc. SGD
$\dfrac{2C_fG^2}{\mu k}$	$\dfrac{17G^2(1+\log(k))}{\mu k}$	$\dfrac{130G^2}{\mu k}$	$\dfrac{4\sigma^2}{\mu k + 4\sigma^2 E_0^{-1}}$

strongly convex case

Interpreting Improved rate: *For both algorithms, simply an improvement to the rate constant. This is a nice theoretical result, and nice example of continuous time method. In order for it to be of practical use, desire*

- *practical algorithm, faster in practice (saw this for simple examples)*
- *remains faster when theory no longer applies*
  - *e.g. nonconvex examples,*
  - *e.g. training DNNs*

Convergence Rate Result

Definition 4.3. *Define the continuous time parametrized Lyapunov function*

$$E^{ac,c}(t, x, v; \epsilon) := (t - \epsilon)^2 (f(x) - f^*) + 2|v - x^*|^2 \tag{22}$$

Define the discrete time Lyapunov function E_k^c by

$$E_k^{ac,c} = E^{ac,c}(t_k, x_k, v_k; h_k) = E^{ac,c}(t_{k-1}, x_k, v_k; 0) \tag{23}$$

Proposition 4.5. *Assume $h_k := \frac{c}{k^\alpha} \leq \frac{1}{\sqrt{L}}$ and $t_k = \sum_{i=1}^k h_i$, then we have the following bound on $\mathbb{E}[f(x_k)] - f^*$*

$$\frac{\frac{1}{16c^2} E_0 + c^2 \sigma^2 (1 + \log(k))}{(k^{1/4} - 1)^2}, \quad \alpha = \frac{3}{4}.$$

Remark 4.6. *Since in (25), the parameters do not depend on L. We observe that the rates of convergence of $\mathbb{E}[f(x_k)] - f^*$ in the previous proposition do not depend on the smoothness of f and are accelerated compared to SGD (Proposition 3.4).*

Setup: abstract Lyapunov analysis

Definition 2.1. *Let $g(t,z,p)$ be L_g-Lipschitz continuous, and affine in the variable p,*

$$g(t,z,p) = g_1(t,z) + g_2(t,z)p. \tag{6}$$

Consider the Ordinary Differential Equation

$$\dot{z}(t) = g(t, z(t), \nabla f(z(t))) \tag{ODE}$$

Referring to (1)*, Consider also the perturbed ODE,*

$$\dot{z}(t) = g(t, z(t), \tilde{\nabla} f(z(t))). \tag{PODE}$$

Definition 2.2. *For a given learning rate schedule $h_k \geq 0$ and $t_k = \sum_{i=0}^{k} h_i$, the forward Euler discretization of* (ODE) *corresponds to the sequence*

$$z_{k+1} = z_k + h_k g(t_k, z_k, \nabla f(z_k)), \tag{FE}$$

given an initial value z_0. Similarly, the forward Euler discretization of (PODE) *is given by*

$$z_{k+1} = z_k + h_k g(t_k, z_k, \nabla f(z_k) + e_k) \tag{FEP}$$

Proof Outline: abstract Lyapunov analysis

- Starting from ODE, and rate-generating Lyapunov function
    - prove rate of decrease of Lyapunov function in time
- Then discretize ODE, constant time, to get a gradient algorithm.
    - Prove rate in k, (consistent with time, where t = h k)
- Then allow stochastic gradients, variance gives an additional error term.
    - Now use variable learning rate, to sum errors
    - Determine learning rate (exponent) from the sum.
- Obtain unified analysis for both stochastic and gradient case.

Rademacher Complexity

RO

conc. measure

Hoeffding $X_1, \dots, X_m$ i.i.d $X_i \in [a, b]$ $\forall i$

$S = (X_1, \dots, X_m)$

$$f(S) = \frac{X_1 + \dots + X_m}{m}$$

then: $\mathbb{P}[\,|f(S) - \mathbb{E}(f(S))| \geq \varepsilon] \leqslant 2\exp\left(-2\frac{m\varepsilon^2}{(b-a)^2}\right)$

McDiarmid generali from f average to follows.

$f : X^m \to \mathbb{R}$ satisfies

$$|f(S) - f(S')| \leqslant \frac{(b-a)^2}{m}$$

$S = (x_1, \dots, x_m)$
$S' = (x_1, \dots, x_i', \dots, x_m)$ $x_1, \dots, x_m, x_i'$ i.i.d. change one.

then $\mathbb{P}[\,|f(S) - \mathbb{E}(f(S))| \geq \varepsilon] \leqslant 2\exp\left(\frac{-2m\varepsilon^2}{(b-a)^2}\right)$ ⊛

Note Rewrite ⊛:

solve for $\delta = 2\exp(-2m\varepsilon^2/(b-a)^2)$

$$\Rightarrow \varepsilon = \sqrt{\frac{(b-a)^2 \log 2/\delta}{2m}}$$

$\Rightarrow |f(S) - \mathbb{E}(f(S))| \leqslant |b-a| \sqrt{\frac{\log 2/\delta}{2m}}$ with prob $\geq 1 - \delta$

Defn Rademacher complexity

$\mathcal{H} = \{ h(x,w) \}$

e.g. SVM $\left\{ \begin{array}{l} h(x,w) = X \cdot W \\ |W| < \Lambda \end{array} \right\}$ & $|X| < r$

$\mathcal{G} = \{ g : (x,y) \to L(h(x),y) ; h \in \mathcal{H} \}$

Loss functions. (Here $y = y(x)$ true label).

Defn 3.1 Emp. Rad. Complexity

$\mathcal{G} : Z = X \times Y \to [a,b]$

$S = S_m = \{ z_1, \ldots, z_m \}$ fixed sample

$$\hat{\mathcal{R}}_S(\mathcal{G}) = \mathbb{E}_{\sigma} \left[\sup_{g \in \mathcal{G}} \frac{1}{m} \sum_{i=1}^{m} \sigma_i \, g(z_i) \right]$$

where $\sigma = (\sigma_1, \ldots, \sigma_m)$ i.i.d $\{\pm 1\}$.

"Rademacher variables"

"correlat of g with random ±1 vector σ.

R2

Defn 3.2 Rad Compl.

$$\mathfrak{R}_m(G) = \mathop{\mathbb{E}}_{S\sim D^m}\left[\hat{\mathfrak{R}}_S(G)\right]$$

"average over samples size m"

Thm 3.3 $G : Z \to [0,1]$ for any $\delta > 0$

over draw S_m :

$$\mathbb{E}[g(z)] \leq \frac{1}{m}\sum_{i=1}^{m} g(z_i) + 2\mathfrak{R}_m(G) + \sqrt{\frac{\log\frac{1}{\delta}}{2m}}$$

wp $\geq 1-\delta$

and

$$\mathbb{E}[g(z)] \leq \frac{1}{m}\sum_{i=1}^{m} g(z_i) + 3\hat{\mathfrak{R}}_S + 3\sqrt{\frac{\log\frac{2}{\delta}}{2m}} \quad \text{wp} \geq 1-\delta$$

R3

Lemma $G : Z \to [0,1]$

Define
$$\Phi(S) = \sup_{g \in G} \left(\mathbb{E}[g] - \hat{\mathbb{E}}_S[g] \right)$$

where $\hat{\mathbb{E}}_S[g] = \frac{1}{m} \sum_{i=1}^{m} g(z_i)$

① $\left| \Phi(S) - \Phi(S') \right| \leq \frac{1}{m}$ because sup sub additive & S, S' differ by 1 point

② McD $\Rightarrow$
$$\Phi(S) \leq \mathbb{E}_S[\Phi(S)] + \sqrt{\frac{\log 2/\delta}{2m}} \quad \text{wp} \geq 1 - \delta/2$$
see conc of measure. McD

③ Meaning of it :

for any $g \in G$
$$\mathbb{E}[g] - \hat{\mathbb{E}}_S[g] \leq \Phi(S)$$
by defn of $\Phi(S)$
$$\leq \mathbb{E}_S[\Phi(S)] + \sqrt{\frac{\log 2/\delta}{2m}} \quad \text{w.p.} \geq 1 - \delta/2$$
by McD applied to $\Phi(S)$

④ will show
$$\mathbb{E}_S[\Phi(S)] \leq 2 R_m(G)$$

Then

Thm

$\forall g \in G$
$$\mathbb{E}[g] \leq \hat{\mathbb{E}}_S[g] + 2R_m + \sqrt{\frac{\log 2/\delta}{2m}} \quad \text{w.p.} \geq 1 - \delta/2$$

☑

R4

4

~~prof Thm 3.3~~ (con't)

~~$$\Phi(S) \leq \mathbb{E}_S[\Phi(S)] + \sqrt{\frac{\log 2/\delta}{2m}} \quad \text{w p} \geq 1-\delta/2$$~~

~~LEMMA $\Phi(S) = \sup_{g\in G}$~~

Bound proof of 4

4

$$\mathbb{E}_S[\Phi(S)] = \mathbb{E}_S\left[\sup_{g\in G}\left(\mathbb{E}[g] - \hat{\mathbb{E}}_S(g)\right)\right]$$

$$= \mathbb{E}_S\left[\sup_{g\in G} \mathbb{E}_{S'}\left[\hat{\mathbb{E}}_{S'}(g) - \hat{\mathbb{E}}_S(g)\right]\right]$$ since $\mathbb{E}[g] = \mathbb{E}_{S'}[\hat{\mathbb{E}}_S(g)]$

$$\leq \mathbb{E}_{S,S'}\left[\sup\left(\hat{\mathbb{E}}_{S'}(g) - \hat{\mathbb{E}}_S(g)\right)\right]$$ sub-add of sup

$$= \mathbb{E}_{S,S'}\left[\sup_{g\in G} \frac{1}{m}\sum_{i=1}^{m}\left(g(z_i') - g(z_i)\right)\right]$$ defn

$$= \mathbb{E}_{\sigma,S,S'}\left[\sup_{g\in G} \frac{1}{m}\sum_{i=1}^{m}\sigma_i\left(g(z_i') - g(z_i)\right)\right]$$

introduce Rad. vars.
$\sigma_i = 1$ same
$\sigma_i = -1$ flip z_i z_i'
same expectation over S, S'

$$\leq \mathbb{E}_{\sigma,S'}\left[\sup_{g\in G} \frac{1}{m}\sum_{i=1}^{m}\sigma_i g(z_i')\right] + \text{"}\mathbb{E}_{\sigma,S}\text{"}$$

$$= 2\mathfrak{R}_m(G)$$

☑

~~Application~~ **Empirical Rad. Complexity**

R5

optional

Did $G : Z \to [0,1]$

$$\mathbb{E}[g(z)] \leq \frac{1}{m}\sum_{i=1}^{m} g(z_i) + 2R_m(G) + \sqrt{\frac{\log\frac{1}{\delta}}{2m}} \quad \text{wp} \geq 1-\delta$$

Next

$$\mathbb{E}[g(z)] \leq \frac{1}{m}\sum_{i=1}^{m} g(z_i) + 3\hat{R}_S(G) + 2\sqrt{\frac{\log\frac{2}{\delta}}{2m}}$$

Note $f(S) = \hat{R}_S(G)$

then S, S' differ by 1 point

$$|f(S) - f(S')| \leq \frac{1}{m}$$

so McDiarmid ineq applied to $\hat{R}_S(G)$

$$\mathbb{E}_S \hat{R}_S(G) = R_m(G) \leq \hat{R}_S(G) + \sqrt{\frac{\log\frac{2}{\delta}}{2m}} \quad \text{wp.} \geq 1-\delta/2$$

combine with $\Phi(S) \leq \underbrace{\mathbb{E}_S[\Phi(S)]}_{2\hat{R}_S(G)} + \sqrt{\frac{\log\frac{2}{\delta}}{2m}}$ wp $\geq 1-\delta/2$

union bound:

$$\Phi(S) \leq 3\hat{R}_S(G) + 2\sqrt{\frac{\log\frac{2}{\delta}}{2m}} \quad \text{wp} \geq 1-\delta$$

SVM ② Ⓡ6

Thm 5.10 $S = \{x : \|x\|^2 \leq r^2\}$ sample size m

$\mathcal{H} = \{x \mapsto w \cdot x : \|w\| \leq \Lambda\}$

Then $\hat{\mathcal{R}}_S(\mathcal{H}) \leq \sqrt{\frac{r^2\Lambda^2}{m}}$ & $\boxed{R(h) \leq \hat{R}_{S,\rho}(h) + 2\sqrt{\frac{r^2\Lambda^2/\rho^2}{m}} + \sqrt{\frac{\log\frac{1}{\delta}}{2m}}}$

wp $\geq 1-\delta$

proof

$$\hat{\mathcal{R}}_S(\mathcal{H}) = \frac{1}{m}\mathop{\mathbb{E}}_{\sigma}\left[\sup_{\|w\|\leq\Lambda}\sum_{i=1}^{m}\sigma_i\, w\cdot x_i\right]$$ defn of $\hat{\mathcal{R}}_S$ & $\mathcal{H}$

$$= \frac{1}{m}\mathop{\mathbb{E}}_{\sigma}\left[\sup_{\|w\|\leq\Lambda} w\cdot\sum_{i=1}^{m}\sigma_i x_i\right]$$ linearity

$$\leq \frac{\Lambda}{m}\mathop{\mathbb{E}}_{\sigma}\left[\left\|\sum_{i=1}^{m}\sigma_i x_i\right\|\right]$$ C-S

$$\leq \frac{\Lambda}{m}\left[\underbrace{\mathop{\mathbb{E}}_{\sigma}\left[\left\|\sum_{i=1}^{m}\sigma_i x_i\right\|^2\right]}\right]^{\frac{1}{2}}$$ Jensen

$\mathbb{E}|X| \leq (\mathbb{E}|X|^2)^{\frac{1}{2}}$
$\mathbb{E}[(|X|^2)^{\frac{1}{2}}]$

$$\mathop{\mathbb{E}}_{\sigma}\sum_{i,j=1}^{m}\sigma_j\sigma_i\, x_i\cdot x_j =$$ $\mathbb{E}[\sigma_i\sigma_j] = 0$

$$= \sum|x_i|^2 \leq mr^2$$

$$\leq \frac{\Lambda}{m}\sqrt{mr^2} = \sqrt{\frac{r^2\Lambda^2}{m}}$$

Cor: combine with

$\textcircled{R7}$
optional

Thm 6.12

$K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ PDS kernel

$\Phi : \mathcal{X} \to \mathbb{H}$ ass. feature mapping

$S = \{x_1, \dots, x_m\}$ ~~[illegible]~~ $K(x_i, x_i) \leq r^2$

$\mathcal{H} = \{ x \mapsto \langle w, \Phi(x) \rangle : \|w\|_{\mathbb{H}} \leq \Lambda \}$

Then $\hat{\mathcal{R}}_S(\mathcal{H}) \leq \Lambda \frac{\sqrt{\mathrm{Tr}(K)}}{m} \leq \sqrt{\frac{r^2 \Lambda^2}{m}}$

---

proof

$$\hat{\mathcal{R}}_S(\mathcal{H}) = \frac{1}{m} \mathop{\mathbb{E}}_{\sigma} \Big[\sup_{\|w\| \leq \Lambda} \sum \sigma_i \langle w \cdot \Phi(x_i) \rangle \Big]$$ defn $\hat{\mathcal{R}}_S$ & $\mathcal{H}$

$$= \frac{1}{m} \mathop{\mathbb{E}}_{\sigma} \Big[\sup_{\|w\| \leq \Lambda} \langle w \cdot \big(\sum \sigma_i \Phi(x_i) \big) \rangle \Big]$$ linearity

$$\leq \frac{\Lambda}{m} \mathop{\mathbb{E}}_{\sigma} \Big[\big\| \sum \sigma_i \Phi(x_i) \big\|_{\mathbb{H}} \Big]$$ C-S

$$\leq \frac{\Lambda}{m} \Big(\mathop{\mathbb{E}}_{\sigma} \Big[\big\| \underbrace{\sum \sigma_i \Phi(x_i)}_{\sum_{i,j} \sigma_i \sigma_j \underbrace{\langle \Phi(x_i), \Phi(x_j) \rangle}_{K(x_i, x_i)}} \big\|_{\mathbb{H}}^2 \Big)^{1/2}$$ Jensen's inq

$\mathbb{E}(\sigma_i, \sigma_j) = 0$
$i \neq j$

$$= \frac{\Lambda}{m} \sqrt{\sum_i K(x_i, x_i)}$$

$$\leq \sqrt{\frac{r^2 \Lambda^2}{m}}$$ same!