Machine Learning I

Lecture 10: Iterative Methods

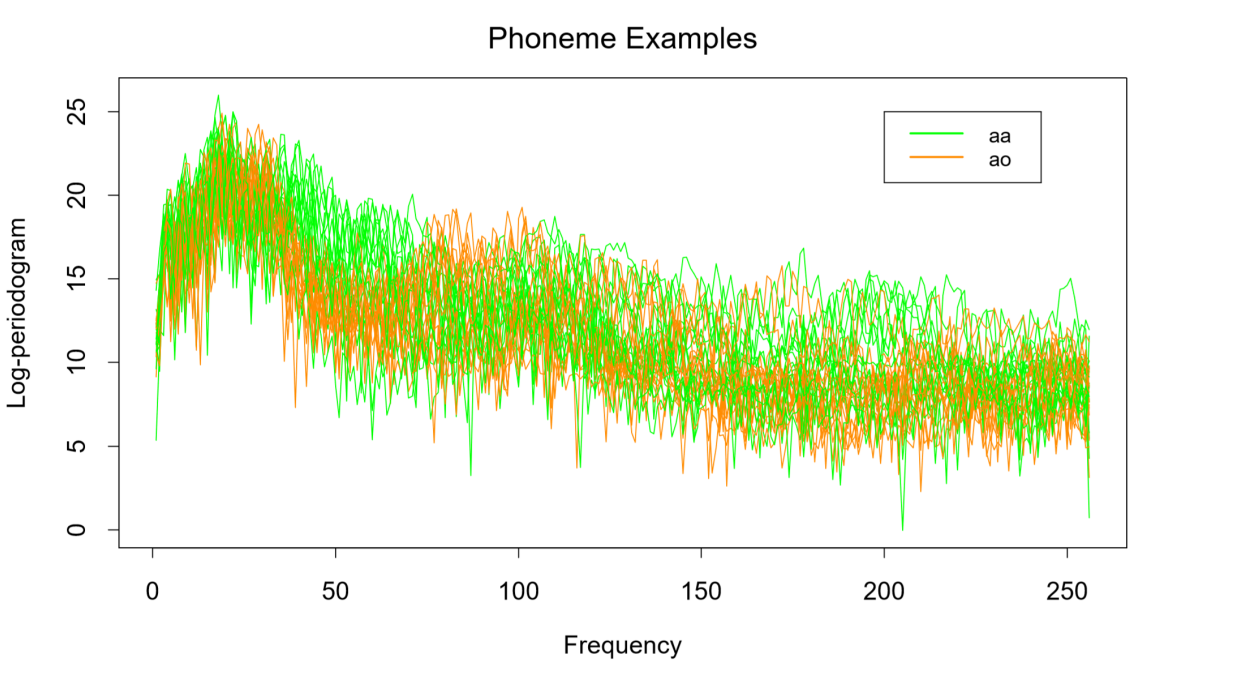
Nathaniel Bade

Northeastern University Department of Mathematics

Table of contents

Introduction: Polynomial Regression?

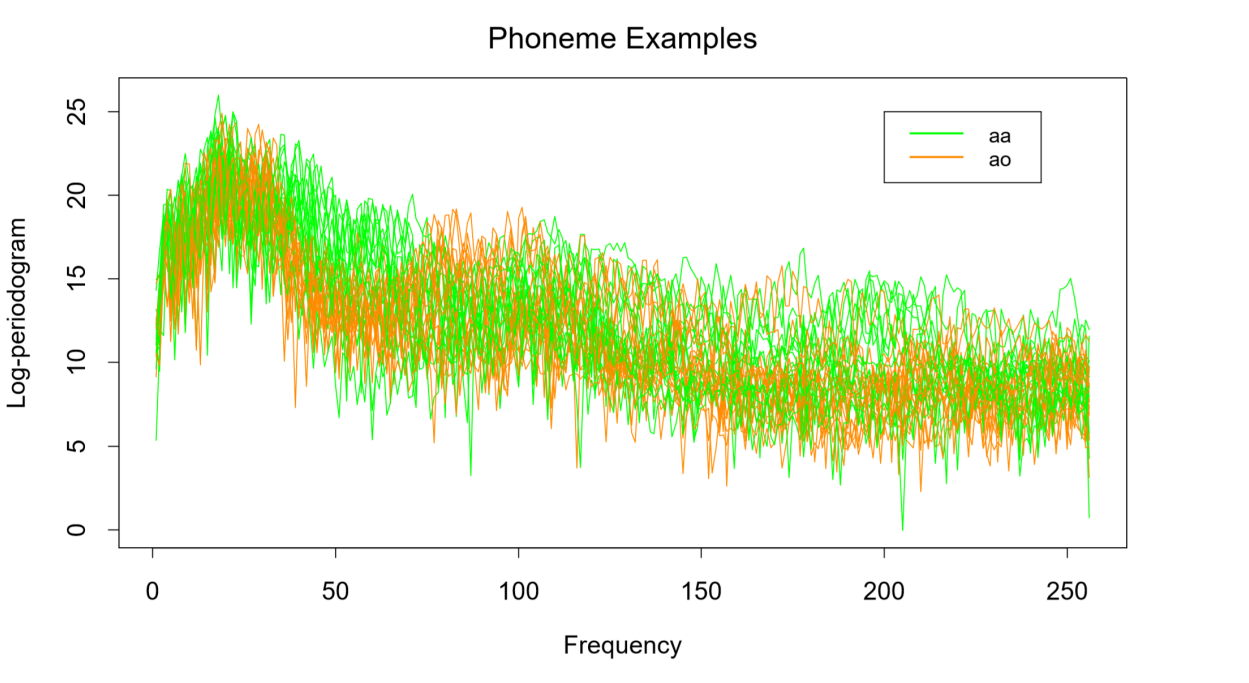
More complicated models



Not everything in the world is linear. We saw last lecture that polynomial functions can be fit by performing linear regression on synthetic features $X_{i_1}^{d_1} \dots X_{i_r}^{d_r}$. The obvious problem is that degree d polynomials on p features lead to

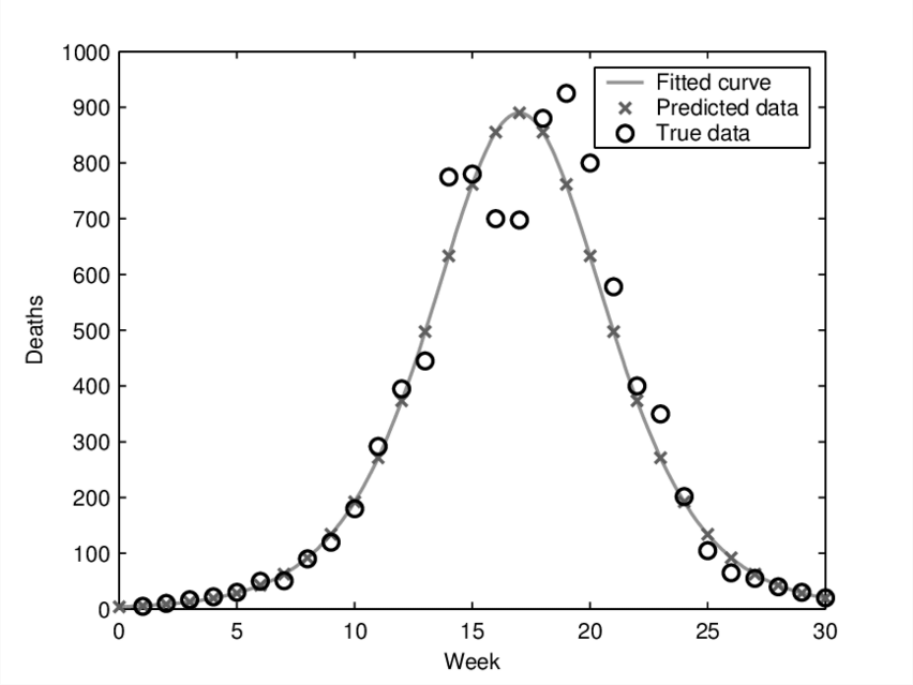
Number of Features: $\binom{d+p}{d}$.

More complicated models



As an example, on the MNIST data set there are 308,505 degree 2 polynomials and 80,931,145 degree three polynomials. Without a principled reason, inverting a matrix to solve such problems quickly becomes prohibitive.

More complicated models

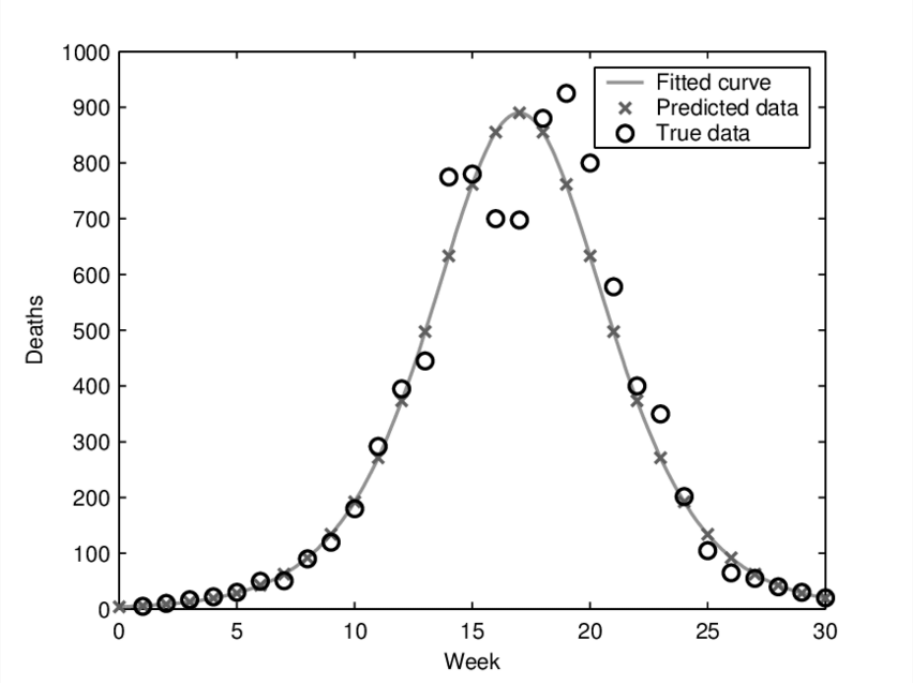


But sometimes we have reason to believe a more complicated functional model is correct. For example, when fitting systems of differential equations

$$\frac{dX}{dt} = -\beta XY \qquad \frac{dY}{dt} = \beta XY - \kappa Y$$

to observed data (Kermack and McKendrick, 1927).

More complicated models

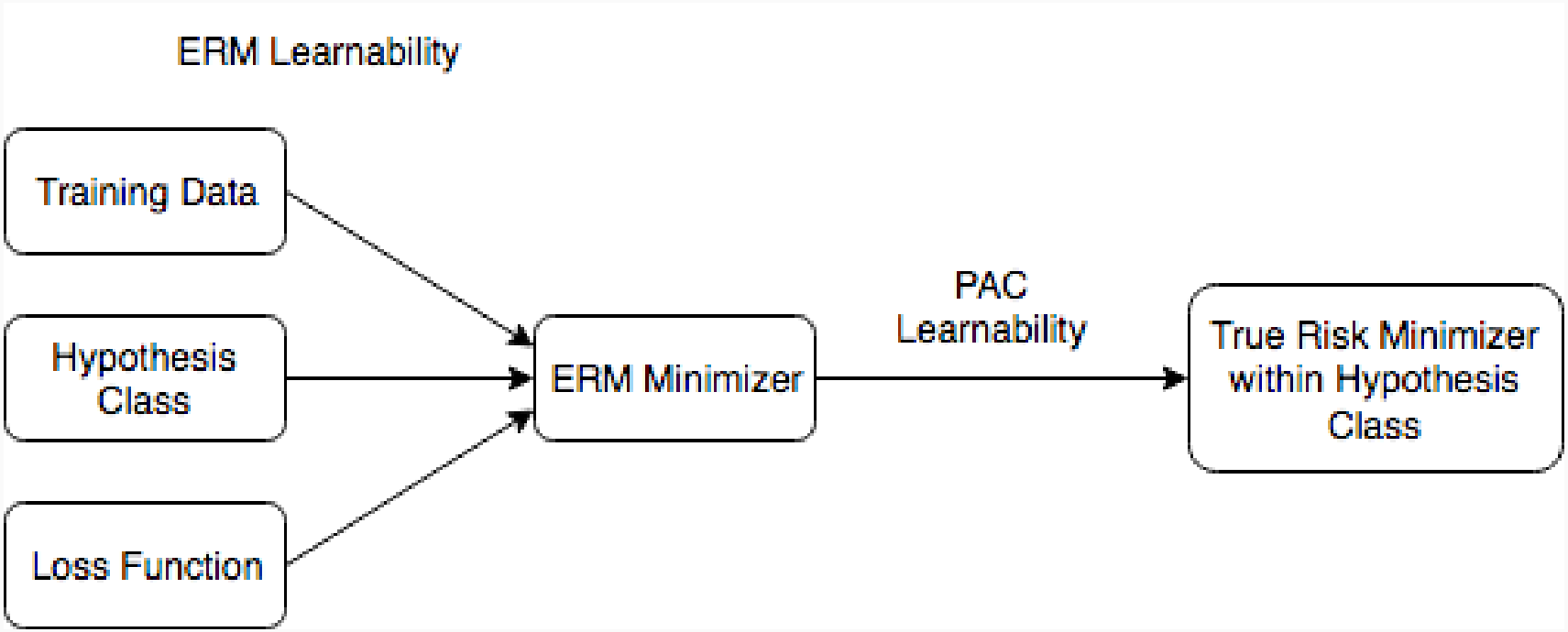


How can we fit such models when the functions are too complicated to yield a close form solution for the minimia? When the number of variables is too large to efficiently use linear methods? When the models themselves may not have closed form formula?

The answer is to use iterative methods from optimization theory to attempt to minimize the loss function stepwise.

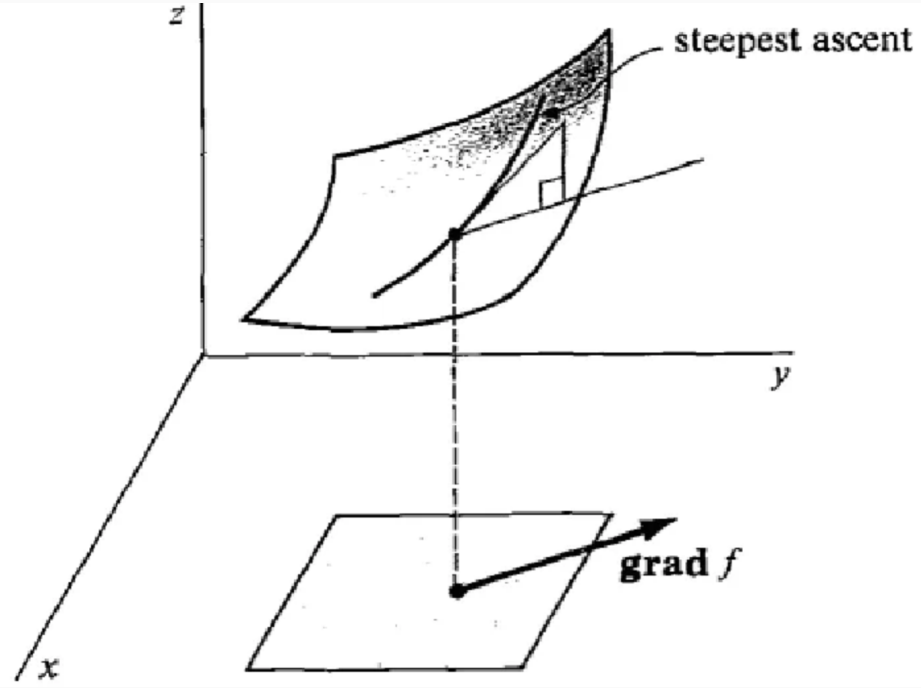
More complicated models

In this lecture we're gong to talk about, and disambiguate, the two learning problems in machine learning. We will see that there is one large class of problems for which ERM and PAC learnability can shown simultaneously. In the first part of this lecture we focus on ERM learning from a practical perspective, and finish off with some theoretical results about the class of PAC learnable problems.



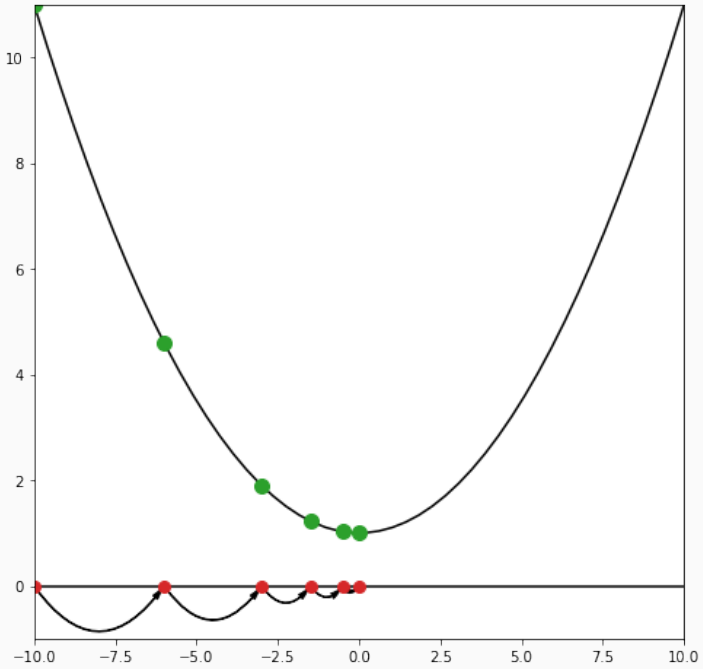
Gradient Decent

Gradient Decent



Gradient decent is an incredibly general method of finding the minima of functions. Let $\ell(\beta)$ be a differentiable loss function depending on k parameters, so $\ell : \mathbb{R}^n \to \mathbb{R}$. The gradient $\nabla\ell$ always points in the direction of greatest increase.

Gradient Decent

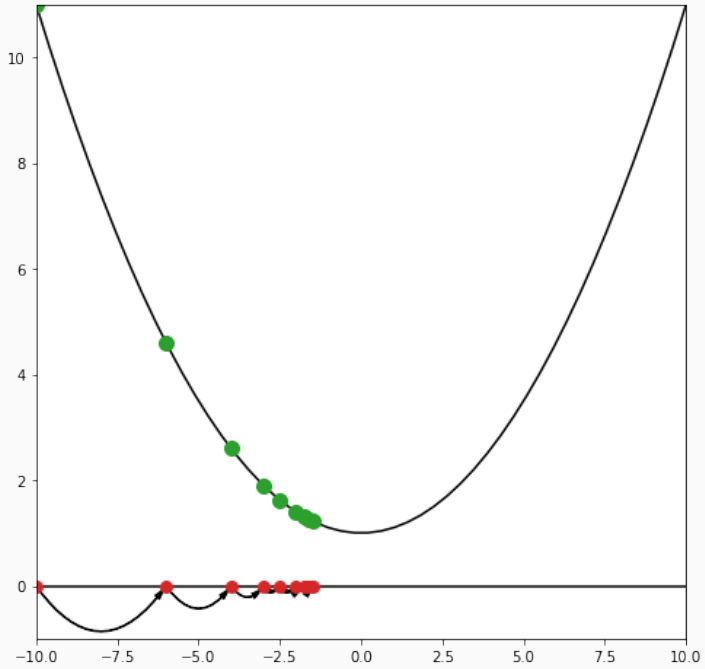


This means that if we want to find a minimum, we follow $-\nabla\ell$ downhill. Algorithmically, at each $\beta^{(n)}$ we compute the next step by

$$\beta^{(n+1)} = \beta^{(n)} - \eta\nabla\ell\,,$$

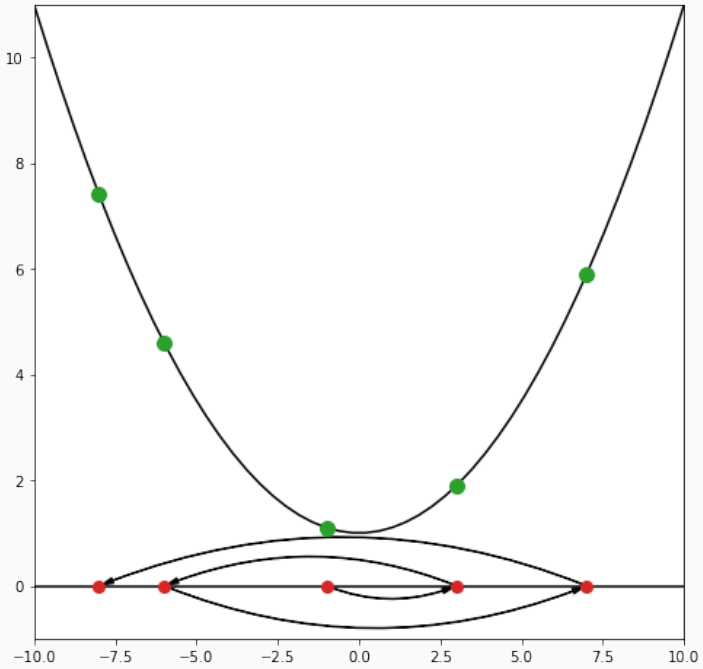
where $\eta > 0$ is the **learning rate** hyperparameter.

Gradient Decent



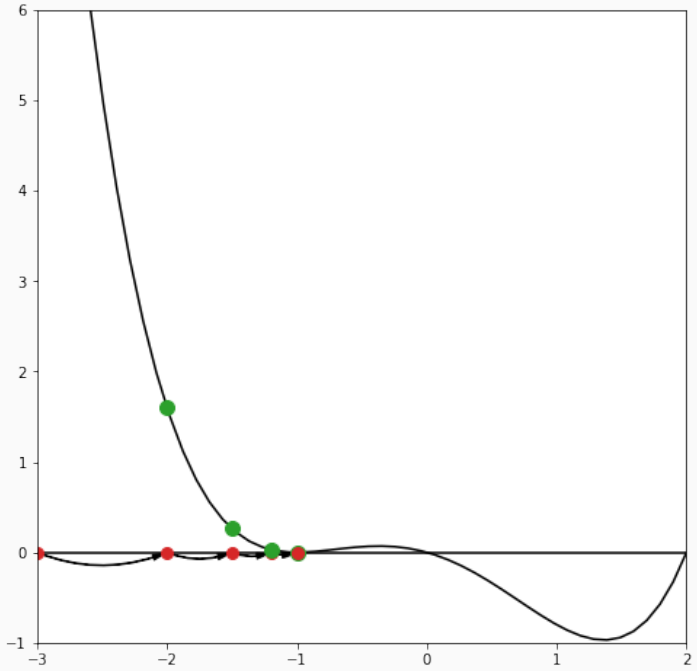
The learning rate η determines proportionally how far of a step we take in the $-\nabla\ell$ direction. If η is too small we may find that the algorithm appears to converge but does so to a point that isn't an actual minimum.

Gradient Decent



On the other hand, if η is too large we may actually walk away from the minimum. We will see an example for linear regression explicitly.

Gradient Decent



Of course, gradient decent is a greedy method and so if η is too small it may miss a global minimum in favor of a local one. If we want to guarantee that gradient decent can actually find the minimum, we will need some extra assumptions on our hypothesis class and loss function. MSE for Linear Regression turns out to only have a single minimum.

Batch Gradient Decent

If we compute the gradient using all of the training data it is called **batch gradient decent**. For

$$MSE(\beta) = \frac{1}{N}(\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta)$$

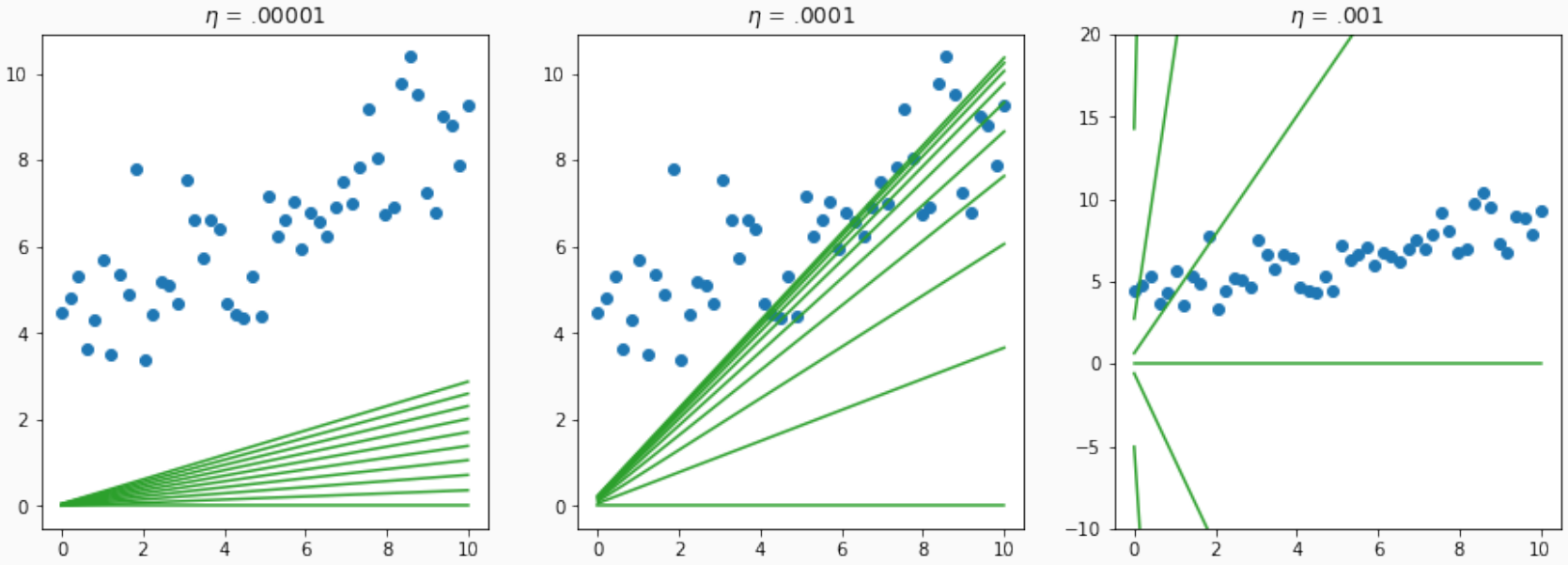
a formula is readily available. Choosing to represent $\nabla MSE(\beta)$ as a $p + 1$ column vector, the gradient is

$$\nabla MSE = -\frac{2}{N}\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta)\,.$$

The update step for the parameters β is

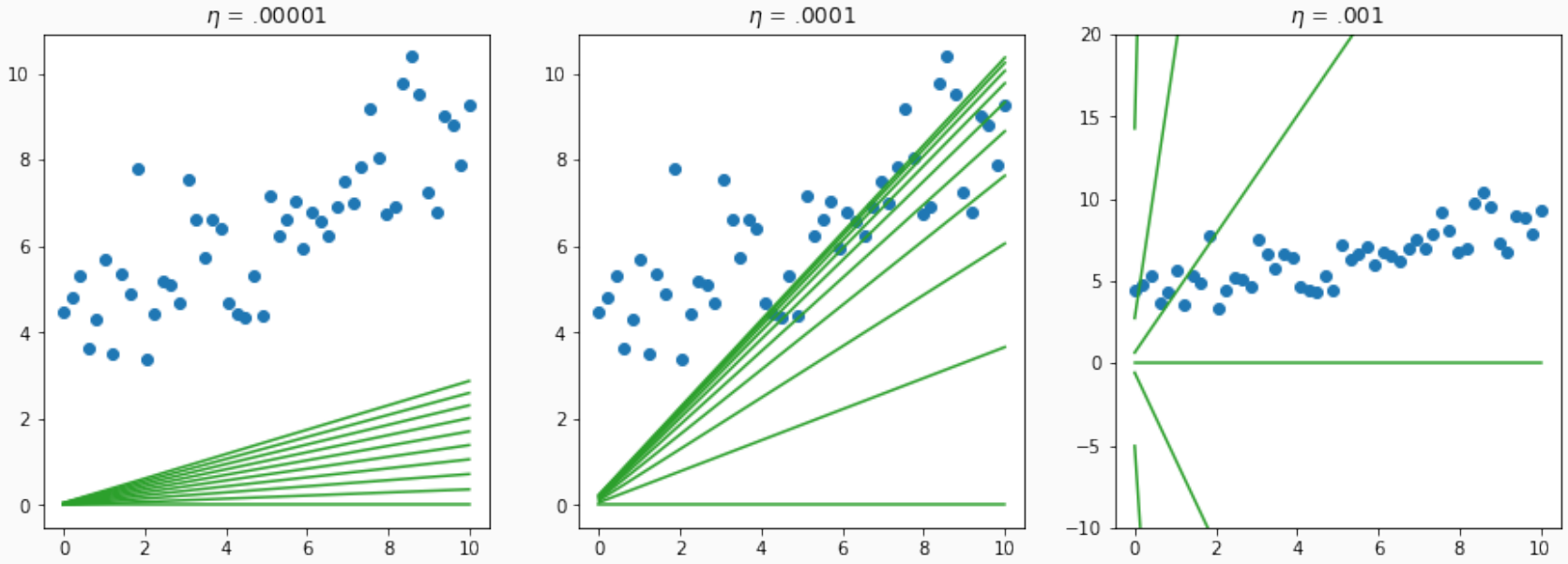
$$\beta^{(n+1)} = \beta^{(n)} + \eta\frac{2}{N}\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta^{(n)})\,.$$

Batch Gradient Decent



We see the results for MSE and linear regression above. If η is too low, convergence is slow (possibly to the point of being undetectable). If it is too large, the fit is actually repelled by the minimum.

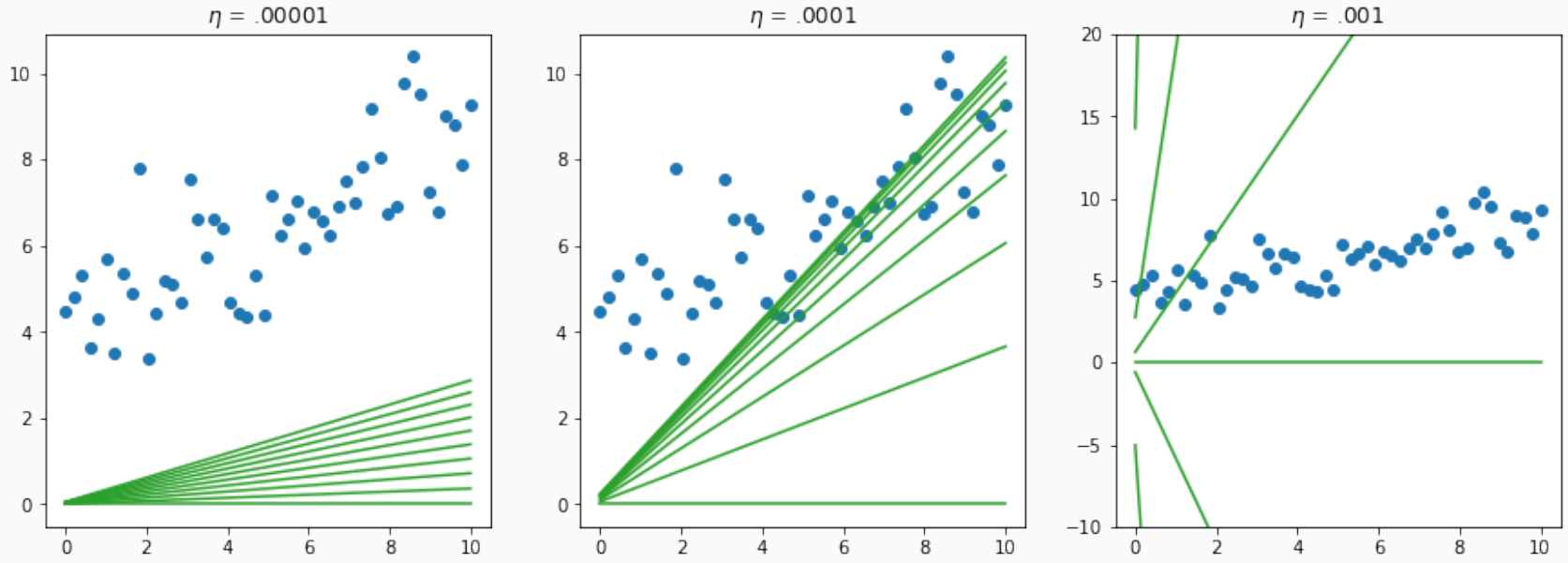
Batch Gradient Decent



In addition to η, we must decide how many iterations we will use. Optimally, we would like to find a learning rate that causes convergence in a small number of iterations.

The learning rate can be fixed (say by performing a grid search to find the best η) or variable, starting large and then lowering to attempt to avoid both falling into and being repelled by local minimia.

Batch Gradient Decent



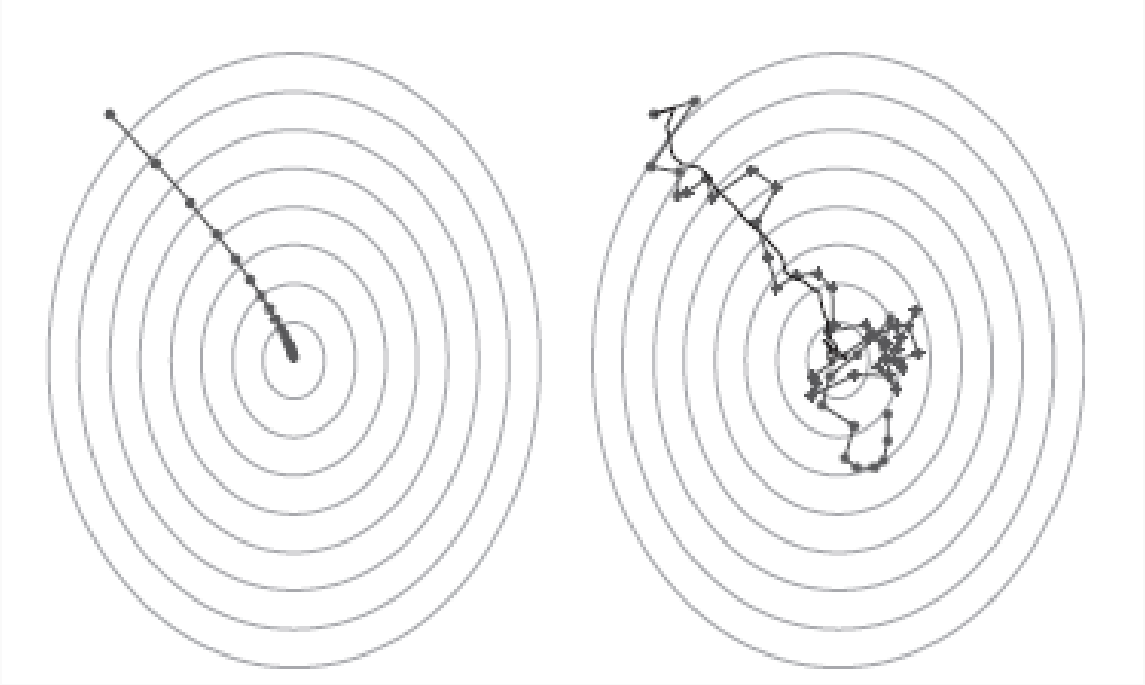
Similarly, the number of iterations can be fixed, or we can choose to stop when, say, $||\nabla\beta|| < \epsilon$ for some **tolerance** ϵ.

It can be show that for *MSE* with fixed η, the convergence rate is approximately $O\left(\frac{1}{\text{itter}}\right)$.

That is, to decrease the tolerance ϵ by $\frac{1}{2}$ we must double the number of iterations.

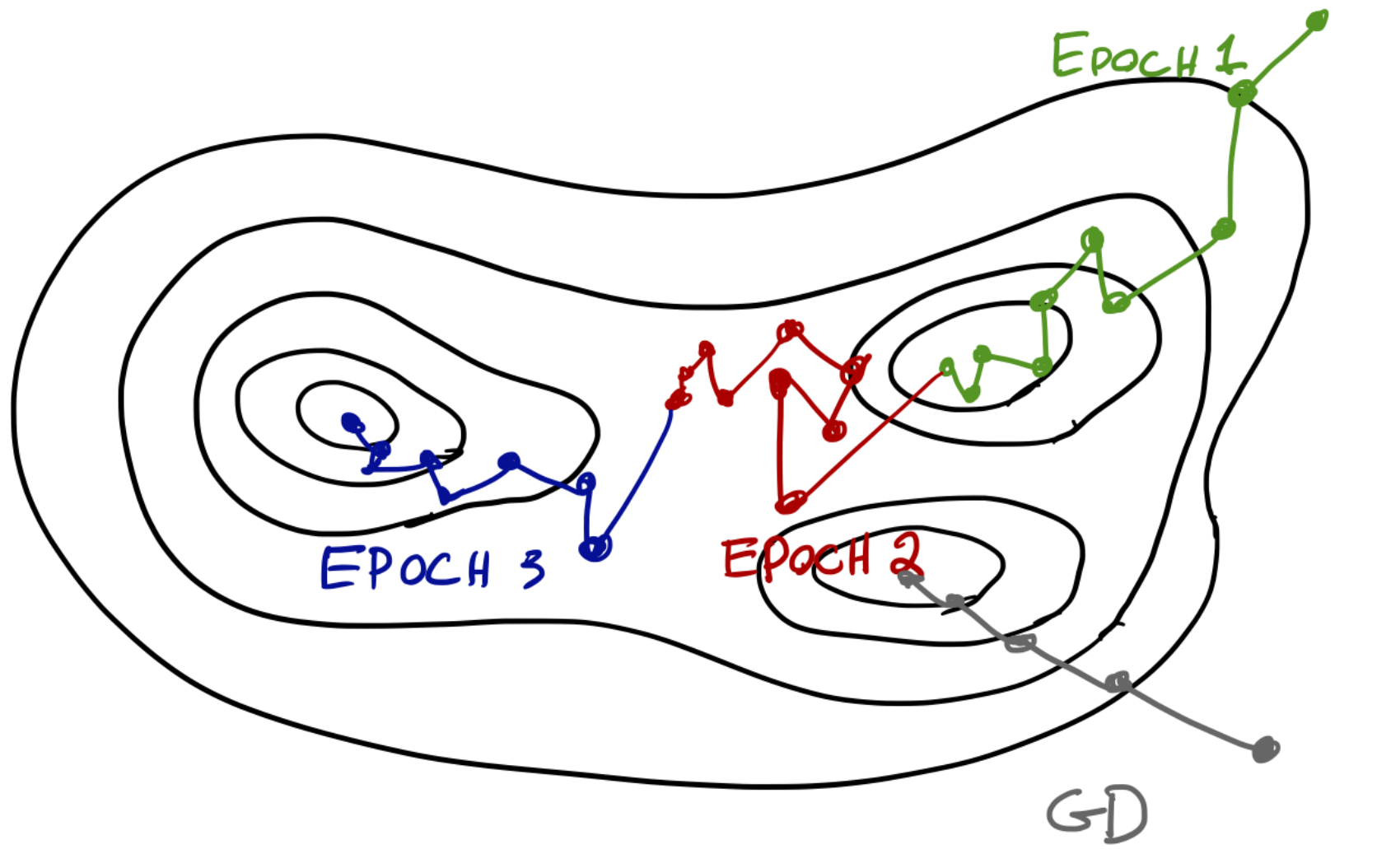
Stochastic Gradient Decent: Comp-sci Perspective

Batch Gradient Decent



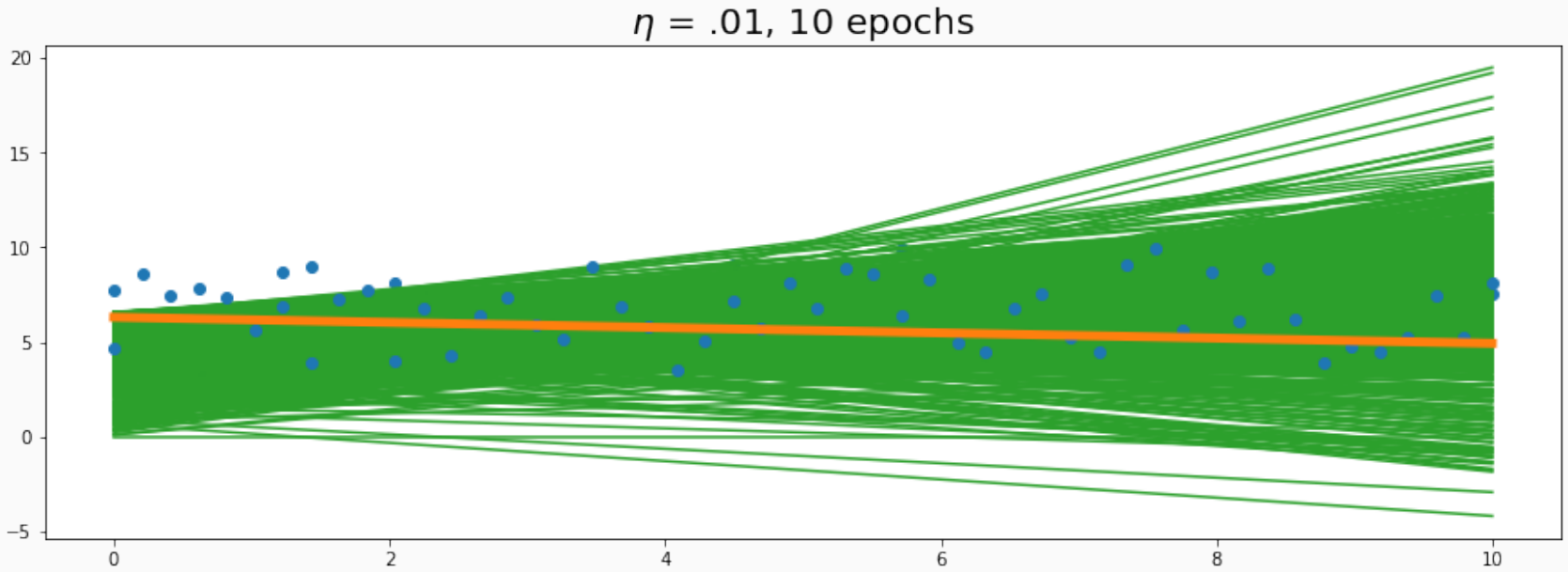
The main problem with batch gradient decent is that all the training data is used to compute the parameters on each iteration. This slows down training considerably when the dataset is large.
At the opposite end, **Stochastic Gradient Decent** (SGD) picks a random element of the training set and uses that to update, like the perception algorithm. SGD a much faster than batch GD since it only needs a single element to train on. The tradeoff is it is much more random.

Stochastic Gradient Decent



SGD is much more likely to avoid getting stuck in local minimia, but on the other hand it doesn't tend to settle down (even in a global minimum). As a result it is almost necessary to gradually lower the learning rate η as

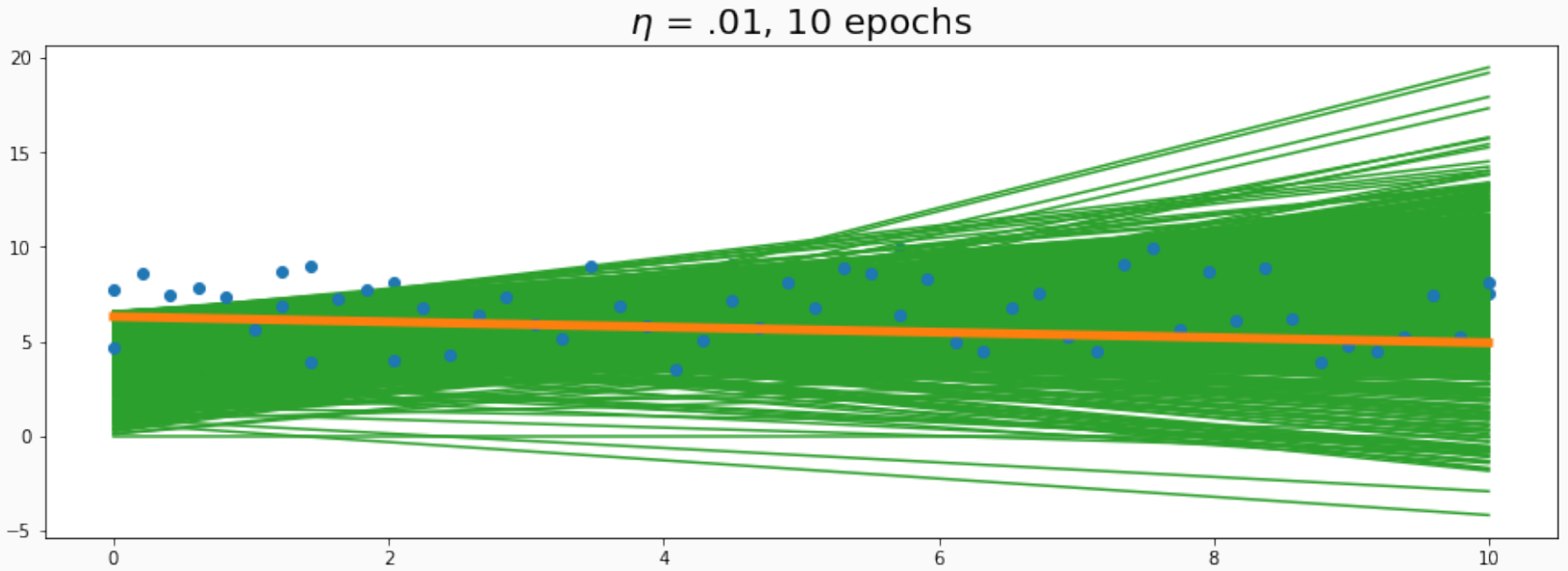
Stochastic Gradient Decent



Training on the entire data set all at once (GD) or one at a time (SGD) called a **training epoch**. It is customary to train multiple epochs, resetting the learning schedule each time.

Although this produces many training steps, each step is computationally simple.

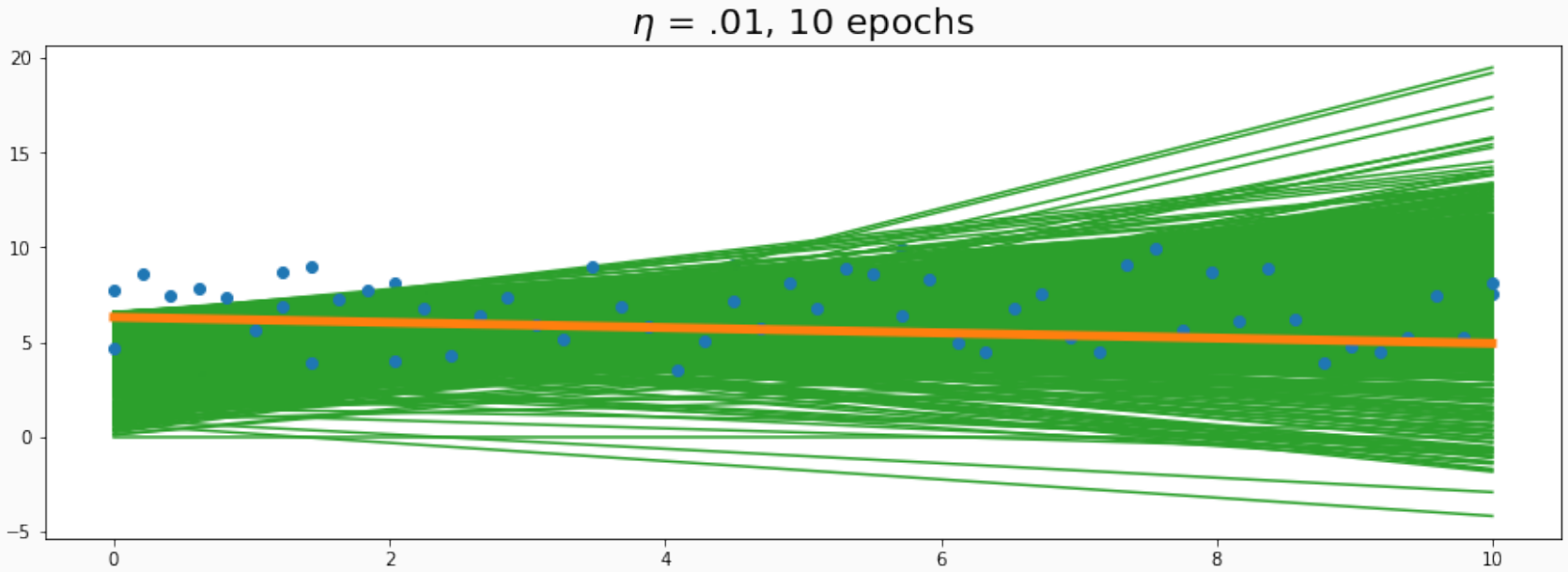
Stochastic Gradient Decent



Training on the entire data set as single instances is called an **epoch** in SGD. It is customary to train multiple epochs, resetting the learning schedule each time to avoid local minimia.

Although this produces many training steps, each step is computationally simple.

Stochastic Gradient Decent



Finally **minibatch gradient decent** splits the difference between SGD and GD, training in epochs of m subsets of size n, where $m \times n = N$. All three methods are implemented in sci-kit learn and (for convex problems) yield the same results.

The batch size m provides a nonlinear trade off between the number of iterations to convergence and the number complexity of each iterative step.

Newtons Method

Newtons Method

Newtons Method is an iterative method of finding zeros of differential functions. By applying it to the first derivative of a function we can use it to find minima, maxima and saddle points.

Newtons method is considerably faster than gradient decent but relies on finding second derivatives of the loss function. This means that not only must the loss function be twice differentiable, but a formula should be available for all second derivatives.

As a result, Newtons method isn't as widely used as GD, SGD and other solvers, but when it can be implemented it tends to be superior.

Newtons Method

In one variable, we try to find a zero of $f(x)$ by successive approximations of $f(x)$ by it's Taylor polynomial.

Starting with x, we try to find an improved guess $x + \delta$ by Taylor expanding $f(x + \delta)$ around x:

$$f(x + \delta) \approx f(x) + \delta f'(x) = 0\,.$$

Solving for $\delta = -f(x)/f'(x)$, we have an "improved" guess $x_{new} = x - f(x)/f'(x)$ for the location of the zero. We then iterate until we have arrived at a zero. It can be proved that under reasonable assumptions on $f(x)$ we always will converge to a zero.

Multivariate Newtons Method

For higher dimensional functions, we proceed exactly as before: Let $f(x)$ be differentiable. Expanding $f(x+\delta)$ around x, we find

$$f(x+\delta) \approx f(x) + J(x)\delta = 0\,,$$

where $J_{ij} = \frac{\partial f_i}{\partial x_j}$ is the Jacobean matrix (gradient if $f(x) \in \mathbb{R}$). If $J(x)$ is invertible, we can solve for

$$x_{new} = x - J^{-1}(x)f(x)\,.$$

Note, unless $f : \mathbb{R}^d \to \mathbb{R}^d$ $J(x)$ will almost certainly not be even partially invertible.

Optimization via Newtons Method

For a single valued multivariate function $f(x)$, we can use Newtons method to perform optimization. Since optimization is just finding $\nabla f(x) = 0$, we write

$$\nabla f(x + \delta) \approx \nabla f(x) + H_f(x)\delta = 0$$

where $H_f(x) = (\frac{\partial f}{\partial x_i \partial x_j})_{ij}$ is the Hessian matrix. At each iteration then we update x_{old} to

$$x_{new} = x_{old} - H_f^{-1}(x)\nabla f(x)\,.$$

Note, Newtons Method converges much faster than gradient decent, but requires computing higher derivatives which might not exist, or may be hard to compute. As a result, it often cannot be used.

Example: Linear Regression Via Newtons Method

Lets solve linear regression using Newtons Method. The gradient of $MSE = \frac{1}{N}(\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta)$ is

$$\nabla MSE = -\frac{2}{N}\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta)\,,$$

and the Hessian is the $(p+1) \times (p+1)$ matrix

$$H_{MSE} = \frac{2}{N}\mathbf{X}^T\mathbf{X}\,.$$

Given an initial $\beta^{(0)}$, the update rule is

$$\beta^{(n+1)} = \beta^{(n)} + (\frac{2}{N}\mathbf{X}^T\mathbf{X})^{-1}\frac{2}{N}\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta^{(n)}) = \beta^{(n)} + (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta^{(n)})\,.$$

Question: How many iterations does Newtons method take to arrive at the answer?

Fitting Logistic Regression

Recall Logistic Regression on a binary labeling. Let y_i take probabilities $k \in \{0, 1\}$. Under the logistic assumption we set

$$\delta_1(x) = \frac{\exp(x^T\beta)}{1 + \exp(x^T\beta)}, \quad \delta_0(x) = \frac{1}{1 + \exp(x^T\beta)},$$

where we have again absorbed β_0 into β. Then

$$\begin{aligned}
\ell(\beta) &= \sum_{i=1}^{N} \log \mathbb{P}(G = y_i | X = x_i; \beta) \\
&= \sum_{i=1}^{N} y_i \log(\delta_1(x_i)) + (1 - y_i) \log(1 - \delta_0(x_i)) \\
&= \sum_{i=1}^{N} y_i x_i^T \beta - \log(1 + e^{x_i^T \beta}).
\end{aligned}$$

Fitting Logistic Regression

Denoting by $\mathbf{d}$ the vector whose i'th coordinate is $\delta_1(x_i)$ fitted to β^{old}, the gradient of

$$\ell(\beta) = \sum_{i=1}^{N} y_i x_i^T \beta - \log(1 + e^{x_i^T \beta}$$

is

$$\nabla \ell(\beta) = \mathbf{X}^T(\mathbf{y} - \mathbf{d}) .$$

Letting $\mathbf{W}$ be the $N \times N$ diagonal matrix with i'th entry $\delta_1(x_i)$ fitted to β^{old}, the Hessian is

$$H = \frac{\partial^2 \ell(\beta)}{\partial \beta \partial \beta^T} = -\mathbf{X}^T \mathbf{W} \mathbf{X} .$$

Then

$$\beta^{new} = \beta^{old} + (\mathbf{X}^T \mathbf{W} \mathbf{X})^{-1} \mathbf{X}^T (\mathbf{y} - \mathbf{p}) .$$

Fitting Logistic Regression

Part of the power of logistic regression comes from the fact that Newtons Method can be directly applied. In addition, it is an explanatory tool, since in general it's coefficients have readily available, explicit interpretations.

Indeed, Newtons Method is often used in data analysis and situation in which one would like to actually explain outcomes, there tends to be a tight correlation between models with explicable parameters and models with computable derivatives.

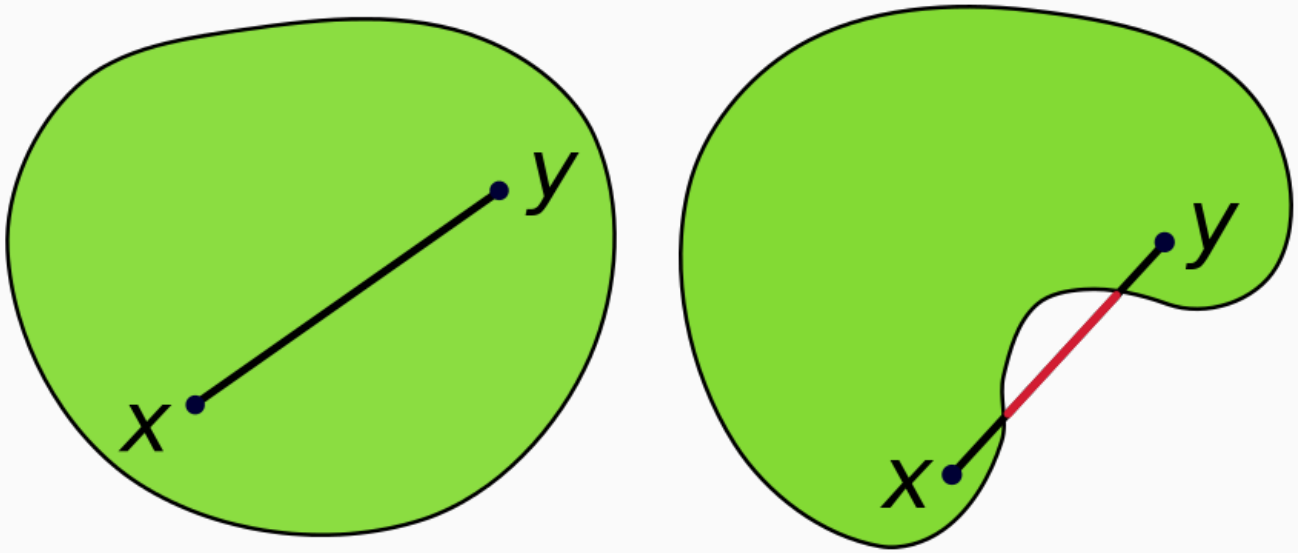
Convex Learning

Convex-Smooth-Lipschitz

Gradient decent is used widely throughout machine learning, mathematical modeling and optimization so we would like to nail down its effectiveness. In particular, is there a criteria for loss functions and hypothesis classes such that GD/SGD is guaranteed to converge?

It turns out the concepts we will want are **convexity**, **Lipschitzness** and **smoothness**. In addition, it can be shown that convex-smooth and convex-Lipschitz problems are PAC learnable.

Convex-Smooth-Lipschitz

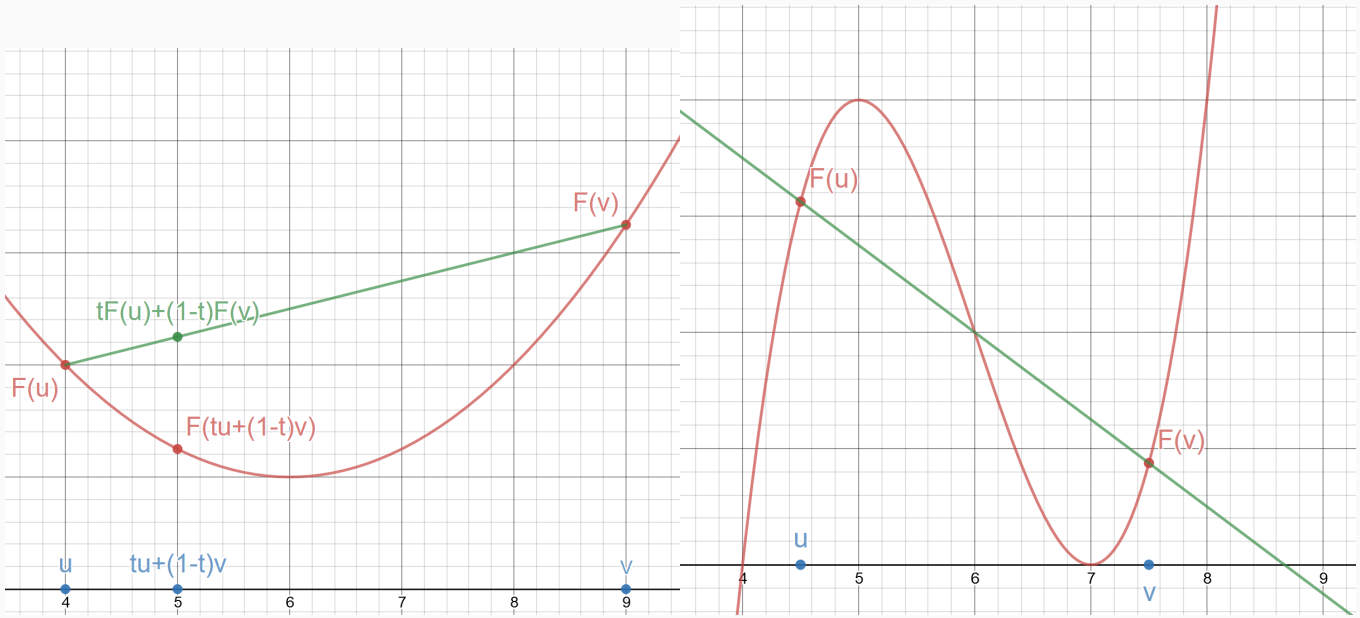


A set C in a vector space is **convex** if for any vectors $u, v \in C$ the line segment

$$tu + (1-t)v\,, \quad t \in [0,1]\,,$$

joining u to v is fully contained in C.

Convex-Smooth-Lipschitz



Similarly, let C be a convex set. A function $F : C \to \mathbb{R}$ is **convex** if for every $u, v \in C$ and $t \in [0, 1]$

$$F\big(\, tu + (1-t)v \,\big) \leq tF(u) + (1-t)F(v)\,.$$

The function on the left is convex, the function on the right is not.

Convex-Smooth-Lipschitz

The utility of convex functions in optimization come from the fact that every local minimum of a convex function is global minimum.

Furthermore, assume that $F : \mathbb{R}^d \to \mathbb{R}$ can be written $F(u) = g(\langle u, x \rangle + y)$ for some $x \in \mathbb{R}^d$, $y \in \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$. then the convexity of g implies the convexity of F. (**Exercise**).

This immediately implies several functions are convex:

For $x \in \mathbb{R}^d$ and $y \in \mathbb{R}$, the function $F(u) = (\langle u, x \rangle + y)^2$ is convex.

For convex functions $F_i : \mathbb{R}^d \to \mathbb{R}$, both $\max_i F_i(x)$ and $\sum_i F_i(x)$ are convex.

The function $g(x) = |x|$ is convex.

The $MSE = \sum(\langle \beta, x_i \rangle + y_i)^2$ is a convex function.

Linear Regression is not PAC learnable.

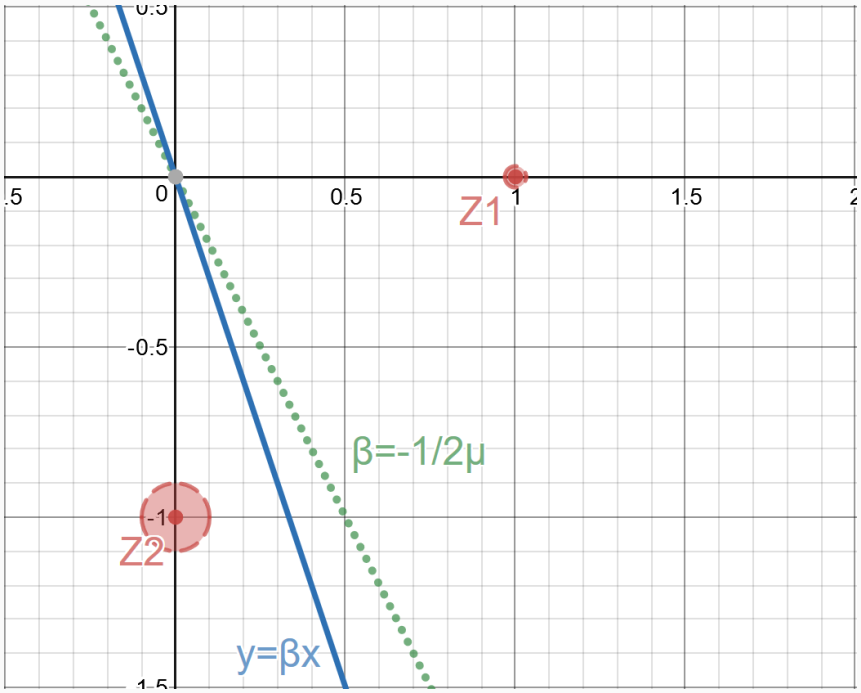
Learnability for Liner Regression

We will now see that convexity is not enough for PAC learnability.

Let $\mathcal{X} = \mathcal{Y} = \mathbb{R}$ and consider the hypthoesis class $\mathcal{H}$ of liner functions $y = x\beta$ with no intercept. The hypothesis class is parameterized entirely by β, so $\mathcal{H} \cong \mathbb{R}$ is a convex set.

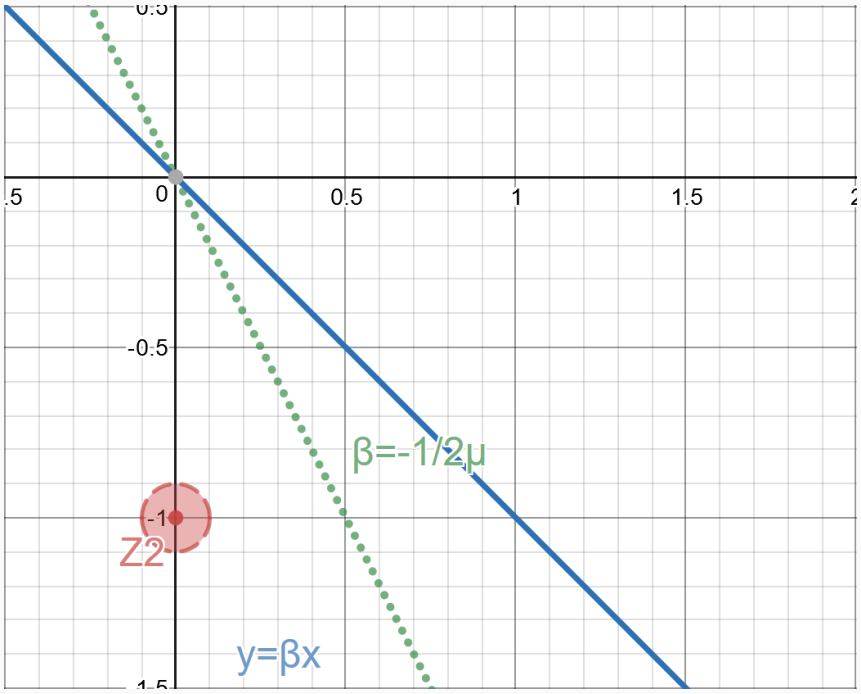
We will build a pair of distributions on two points and show that at least one cannot be learned.

Learnability for Liner Regression



The idea that D_1 is highly localized around z_2, so much so that for fixed N there's a high (say 99%) chance the all of the training samples come from there. But even so, as we swing $y = \beta$ towards it our prediction for z_1, and therefore the loss, becomes arbitrarily bad.

Learnability for Liner Regression



The distribution D_2 contains only the point z_2. So if we try to compensate by keeping β reasonable, we can make m large enough that a even being close to z_2 the loss still becomes large.

Learnability for Liner Regression

Concretely, assume the class is PAC learnable. Fix $\epsilon > 0$ and $\delta > 0$ and let $N \geq N(\epsilon, \delta)$ so that an ERM learner on N training samples satisfies

$$L_{\mathcal{D}}(\beta) - \min_{\beta} L_{\mathcal{D}}(\beta) \leq \epsilon$$

with probability greater than $1 - \delta$.

Let $\mu = \frac{1}{2N} \log(1/(1-\delta))$, and fix $z_1 = (1, 0)$ and $z_2 = (\mu, -1)$.

The distribution $\mathcal{D}_1$ is supported on z_1 with probability μ and z_2 with probability $1 - \mu$. The distribution $\mathcal{D}_2$ is supported only on z_2.

Learnability for Liner Regression

Since $\mathcal{D}_1$ is z_2 with probability $1 - \mu$, the probability that all N samples come from z_2 is

$$(1 - \mu)^N \geq e^{-2\mu N} \geq 1 - \delta, .$$

Therefore, there is more than a $1 - \delta$ chance that all training samples come from z_2 when choosing from either $\mathcal{D}_1$ or $\mathcal{D}_2$.

We will assume for the rest of the proof that all of the training samples are taken from z_2. We will now choose the distribution based on the slope β.

Learnability for Liner Regression

If $\hat{\beta} < -\frac{1}{2\mu}$, we will show $\mathcal{D}_1$ is unlearnable:

The loss can be computed as

$$L_{\mathcal{D}_1}(\hat{\beta}) = \mu(-\hat{\beta})^2 + (1-\mu)(-1-\mu\hat{\beta})^2 \geq \mu(-\hat{\beta})^2 > \frac{1}{4\mu}\,.$$

We don't know the minimum exactly, but we can bound the minimum loss

$$\min_{\beta} L_{\mathcal{D}_1}(\beta) \leq L_{\mathcal{D}_1}(0) = 1-\mu\,.$$

Combining, we see that

$$L_{\mathcal{D}_1}(\hat{\beta}) - \min_{\beta} L_{\mathcal{D}_1}(\beta) \geq \frac{1}{4\mu} - (1-\mu) \geq \epsilon$$

since $\frac{1}{4\mu}$ can be made arbitrarily large by making N large.

Learnability for Liner Regression

Conversely, if $\hat{\beta} \geq -\frac{1}{2\mu}$, the ERM learner fails to learn the second distribution since

$$\begin{aligned} L_{\mathcal{D}_2}(\hat{\beta}) &= (-1 - \mu\hat{\beta})^2 \\ &= (1 + \mu\hat{\beta})^2 \\ &\geq \left(1 + \mu(-\frac{1}{2\mu})\right)^2 \\ &= \frac{1}{4}. \end{aligned}$$

In this case, $\mathcal{D}_2$ is actually realizable so this difference between the minimum error and $L_{\mathcal{D}_2}(\hat{\beta}) > \epsilon$.

Therefore, even for the 1 variable linear regression, PAC learning fails.

Learnability for Liner Regression

How can learnability fail given that liner regression has closed form ERM rule?

The answer is that machine learning has two parts:

> **ERM Problem**: Given a training set $\mathcal{T}$, a hypothesis class $\mathcal{H}$ and a loss function ℓ, does there exist and efficient algorithm to minimize the **empirical risk**? (Algorithms, Applied Math Problem)
>
> **Learnability Problem:** Given a domain $\mathcal{X} \times \mathcal{Y}$, a hypothesis class $\mathcal{H}$ and a loss function ℓ, does minimizing the empirical risk guarantee that the true risk is small? (Pure/Applied Math Problem)

Note: just because a hypothesis class isn't PAC learnable doesn't mean we cant successfully fit it, but the theoretical guarantees are lessened.

Mathematical results on GD, SGD and learnability

ρ-Lipschitz Functions

We have shown that even in the best cases, convexity is not sufficient to ensure learnability. The additional requirement needed on ℓ is Lipschitzness:

Let $C \subset \mathbb{R}^d$. A function $F : \mathbb{R}^d \to \mathbb{R}$ is ρ-Lipschitz (over C) with respect to the norm $||\cdot||$ if for every $u, v \in C$,

$$||F(u) - F(v)|| \leq \rho||u - v||\,.$$

Intuitively, this means F can't change too fast. Recall for smooth functions that $F(u) - F(v) = \nabla F(u')(u - v)$ so in some sense this is bounding the derivative $|F(u)| \leq \rho$.

ρ-Lipschitz Functions

Functions that satisfy $||F(u) - F(v)|| \leq \rho||u - v||$ include

$g(x) = |x|$ is Lipschitz. What is ρ?

$g(x) = \log(1 + \exp(x))$ is 1 Lipschitz.

$g(x) = x^2$ is not Lipschitz, because the derivatives can become arbitrarily large.

If g_1 is ρ_1-Lipschitz and g_2 is ρ_2-Lipschitz, then $g_1 \circ g_2$ is $(\rho_1\rho_2)$-Lipschitz.

Convex-Lipschitz Gradient Decent

Corollary 14.2 SB: Let $F : \mathbb{R}^d \to \mathbb{R}$ be a convex, ρ-Lipschitz function and let u_* minimize $F(u)$ on some r-ball $B_r \subset \mathbb{R}^d$. Then for any $\epsilon > 0$, after n steps of gradient decent

$$F(u^{(n)}) - F(u_*) \leq \frac{r\rho}{\sqrt{T}},$$

provided the learning rate $\eta = \sqrt{\frac{r^2}{\rho^2 n}}$ and

$$n \geq \frac{r^2\rho^2}{\epsilon^2}.$$

Again we see that gradient decent is useful when we have explicit forms for the functions we're trying to minimize, and in general this gives good results for empirical risk minimization.

Convex-Lipschitz SGD

A similar result holds for SGD, but with expectation values:

Theorem 14.8 SB: Let $F : \mathbb{R}^d \to \mathbb{R}$ be a convex, ρ-Lipschitz function and let u_* minimize $F(u)$ on some r-ball $B_r \subset \mathbb{R}^d$. Then for any $\epsilon > 0$, after n steps of gradient decent

$$E\big[\, F(u^{(n)})\, big] - F(u_*) \leq \frac{r\rho}{\sqrt{T}}\,,$$

provided the learning rate $\eta = \sqrt{\frac{r^2}{\rho^2 n}}$, $||u^{(i)}|| \leq \rho$ and

$$n \geq \frac{r^2\rho^2}{\epsilon^2}\,.$$

The proof here is not too technical but we will leave it to the PhD level ML course.

SGD and Learnability

Recall that in learning problems with a loss function ℓ we are really interested in minimizing the true risk

$$L_{\mathcal{D}}(\beta) = E_{z \approx \mathcal{D}}\big[\, \ell(\beta, z) \,\big] .$$

Theorem 14.8 allow us to take a different approach to ERM and minimize $L_{\mathcal{D}}(\beta)$ directly. All we need to do is find an unbiased estimate of the gradient $\nabla L_{\mathcal{D}}(\beta)$.

If $L_{\mathcal{D}}(\beta)$ is differentiable, then for a single sample $z \sim \mathcal{D}$, the gradient commutes with expectation over a different variable:

$$E_{z \approx \mathcal{D}}\big[\, \nabla_\beta \ell(\beta, z) \,\big] = \nabla_\beta E_{z \approx \mathcal{D}}\big[\, \ell(\beta, z) \,\big] = \nabla_\beta L_{\mathcal{D}}(\beta)$$

So $\nabla_\beta \ell(\beta, z)$ is an unbiased estimator of the gradient $L_{\mathcal{D}}(\beta)$. Resampling at each step, we have

Convex-Lipschitz SGD

Corollary 14.12 SB: Consider a convex, ρ-Lipschitz, learning problem with parameters bounded within a ball of radius r. For evvery $\epsilon > 0$, running SDG on $L_{\mathcal{D}}(\beta)$ with

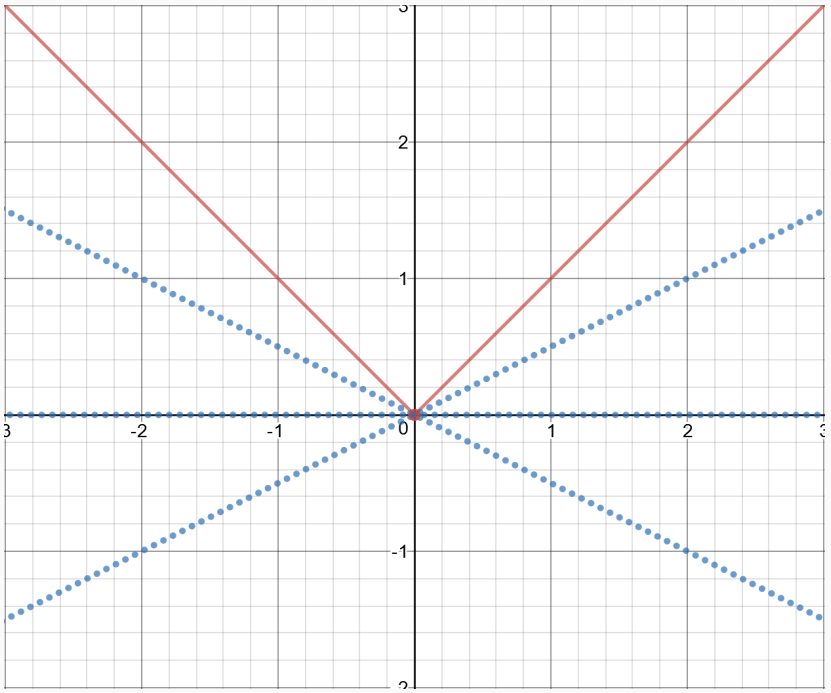
$$n \geq \frac{r^2 \rho^2}{\epsilon^2}$$

yields

$$L_{\mathcal{D}}(\beta^{(n)}) \leq \min_{\beta \in \mathcal{H}} L_{\mathcal{D}}(\beta) + \epsilon \,.$$

So SGD directly minimized with respect to the **ture risk**, provided there is enough data to get an unbiased estimate.

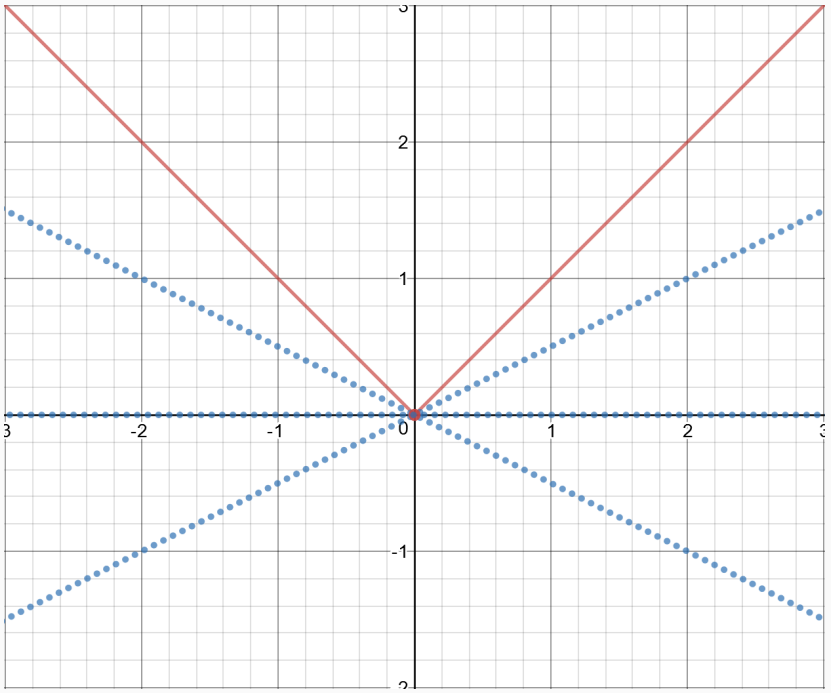
Subgradients



As a final remark, not that the theorem did not require smoothness. It turns out that the proof holds in the non-smooth case, one just needs to make use of the concept of a subgradient.

Without going into too much detail, a subgradient is an estimator of a gradient which again serves the purpose of bounding functional growth.

Subgradients



This is good, because it means we can make sense of loss functions containing no differential terms, like $\sum_i |y_i - x_i^T \beta|$.

References

Plauge Fit Image: Recursive Bayesian Inference on Stochastic Differential Equations, Simo Särkkä. Gradient Picture: https://www.jeffreythompson.org/blog/2014/01/01/gradient-vector/ Convexity Pictures: Wikimedia Foundation: https://en.wikipedia.org/wiki/Convex_set

The content of this lecture is a summary of Shalev-Shwartz and Ben-David, Chapters 12 and 14 and Geron Chapter 4.