ECE826 Lecture 6:

Computational aspects of ERM and Intro to Gradient-based Algorithms

Contents

- From statistical bounds to optimization
- Computational aspects of the ERM
- What can we not do, computationally?
- What can we do? First stop: Convexity
- The proliferation of gradients

Some Definitions

- Our goal is to find a hypothesis (classifier) h_S with small expected risk

$$R[h_S] = \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\ell(h_S(x); y)\right]$$

- The loss measures the disagreement between predictions and reality

- Since we can't directly measure $R[h_S]$ (our true cost function), we can consider optimizing its sample-average proxy, i.e., the empirical risk

$$\hat{R}[h_S] = \frac{1}{n}\sum_{i=1}^{n} \ell(h_S(x_i); y_i)$$

Minimizing the Empirical Risk

- The gap of the true cost function from the one we have access to

$$\min_{h\in\mathcal{H}}\left(R_S[h] = \frac{1}{n}\sum_{i=1}^{n} \ell(h(x_i); y_i) \right)$$

- <u>Question</u>: Can we find the solution to this minimization? If so how fast?

- The answer must depend on:
  1) n, the sample size
  2) $\mathcal{H}$, the hypothesis class and loss function
  3) $\mathcal{D}$, the data distribution
  4) the optimization algorithm that outputs our classifier

Computational Aspects of the ERM

ERM is erm… hard to solve

Theorem:

Empirical risk minimization is NP-hard in general

ERM is erm… hard to solve

Theorem:

Empirical risk minimization is NP-hard in general

Proof:

- Re-write any hard problem as a minimization of a sum of n functions.
- For example, $\ell(w; A_{i,j}) = -A_{i,j}(1 - w_i \cdot w_j)$ and $w \in \{\pm 1\}^{|V|}$ and

$$\min_{w \in \{\pm\}^{|V|}} \frac{1}{|E|} \sum_{(i,j) \in E} -A_{i,j}(1 - w_i w_j)$$

- This is the MaxCut problem, which is NP-hard.

What about learning NNs?

Theorem: [Judd 88, Sima 94]

For a fixed architecture it is NP-complete to decide if there exists a set of weights that can memorize a data set.

What about learning NNs?

Theorem: [Judd 88, Sima 94]

For a fixed architecture it is NP-complete to decide if there exists a set of weights that can memorize a data set.

Theorem: [Manurangsi & Reichman 2018]

Even for the case of a single ReLU activation, finding a set of weights that minimizes the squared error (even approximately) for a given training set is NP- hard.

What about learning NNs?

Theorem: [Judd 88, Sima 94]

For a fixed architecture it is NP-complete to decide if there exists a set of weights that can memorize a data set.

Theorem: [Manurangsi & Reichman 2018]

Even for the case of a single ReLU activation, finding a set of weights that minimizes the squared error (even approximately) for a given training set is NP- hard.

Theorem: [Goel et al. 2020]

Even in the "realizable" case (i.e., where a planted ground truth exists) learning depth-2 ReLUs is hard.

What about learning NNs of arbitrary size?

What about learning NNs of arbitrary size?

Theorem:

Let $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ such that no two data points are identical in feature space. Then, we can always create a threshold neural network that fits the data set.

What about learning NNs of arbitrary size?

Theorem:

Let $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ such that no two data points are identical in feature space. Then, we can always create a threshold neural network that fits the data set.

Proof: How can I memorize a single data point?

What about learning NNs of arbitrary size?

Theorem:

Let $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ such that no two data points are identical in feature space. Then, we can always create a threshold neural network that fits the data set.

Proof: How can I memorize a single data point?

- Let w be a d dimensional gaussian vector and $\langle x_i, w \rangle = b_i$. These b_is are unique, since $x_i \neq x_j$.

What about learning NNs of arbitrary size?

Theorem:

Let $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ such that no two data points are identical in feature space. Then, we can always create a threshold neural network that fits the data set.

Proof: How can I memorize a single data point?

- Let w be a d dimensional gaussian vector and $\langle x_i, w \rangle = b_i$. These b_is are unique, since $x_i \neq x_j$.
- w.l.og. assume that $\min_{i,j;i\neq j} |b_i - b_k| = 10\epsilon$

What about learning NNs of arbitrary size?

Theorem:

Let $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ such that no two data points are identical in feature space. Then, we can always create a threshold neural network that fits the data set.

Proof: How can I memorize a single data point?

- Let w be a d dimensional gaussian vector and $\langle x_i, w \rangle = b_i$. These b_is are unique, since $x_i \neq x_j$.
- w.l.og. assume that $\min_{i,j;i\neq j} |b_i - b_k| = 10\epsilon$

- Then here is a way to memorize a single label

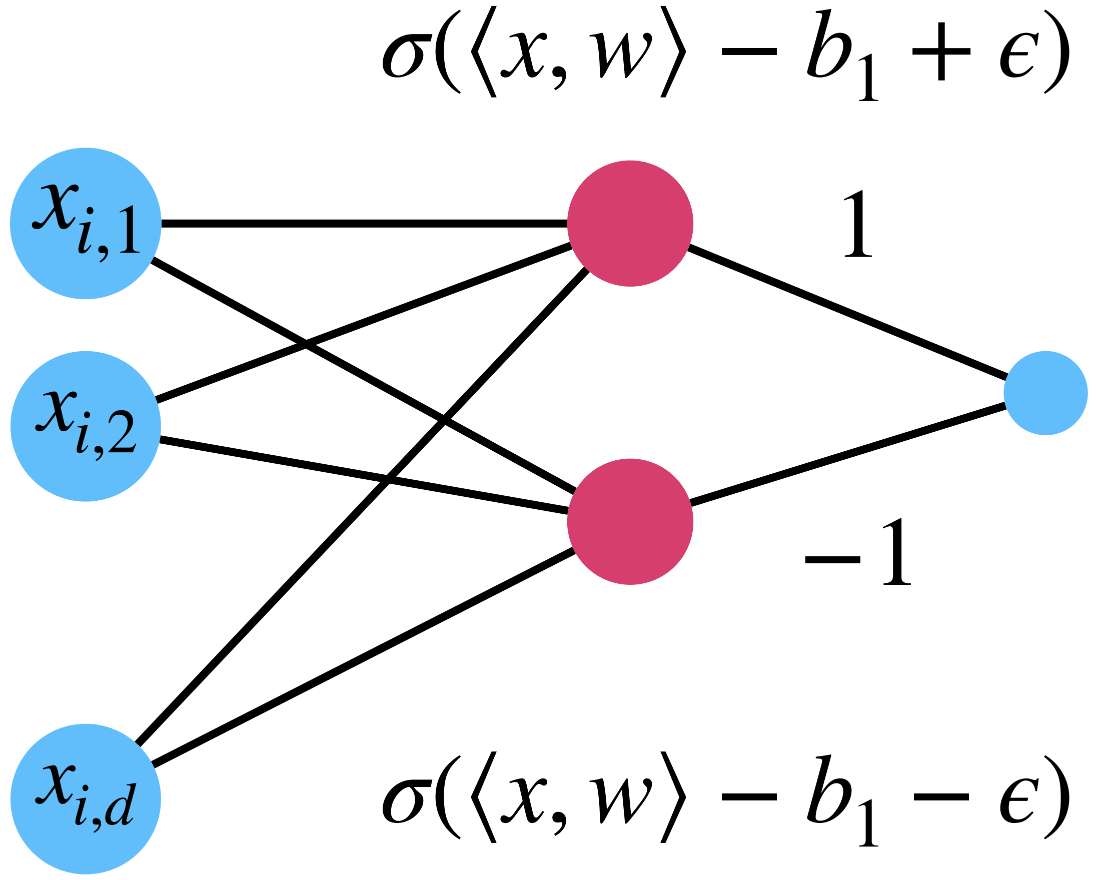
What about learning NNs of arbitrary size?

Theorem:

Let $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ such that no two data points are identical in feature space. Then, we can always create a threshold neural network that fits the data set.

Proof: How can I memorize a single data point?

- Let w be a d dimensional gaussian vector and $\langle x_i, w \rangle = b_i$. These b_is are unique, since $x_i \neq x_j$.
- w.l.og. assume that $\min_{i,j;i\neq j} |b_i - b_k| = 10\epsilon$
- Then here is a way to memorize a single label



What about learning NNs of arbitrary size?

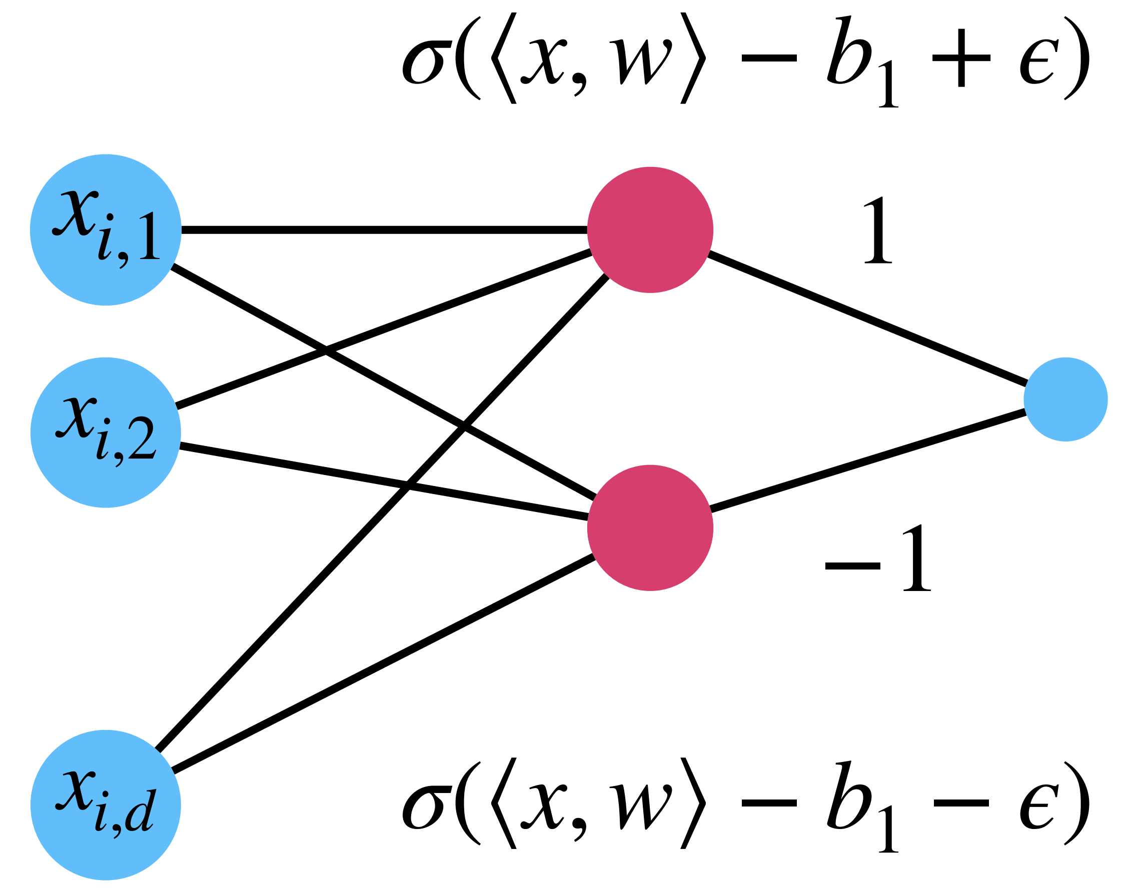
Theorem:

Let $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ such that no two data points are identical in feature space. Then, we can always create a threshold neural network that fits the data set.

Proof: How can I memorize a single data point?

- Let w be a d dimensional gaussian vector and $\langle x_i, w \rangle = b_i$. These b_is are unique, since $x_i \neq x_j$.
- w.l.og. assume that $\min_{i,j;i\neq j} |b_i - b_k| = 10\epsilon$

- Then here is a way to memorize a single label
- You can memorize 1 data point with 2 activations



What about learning NNs of arbitrary size?

Theorem:

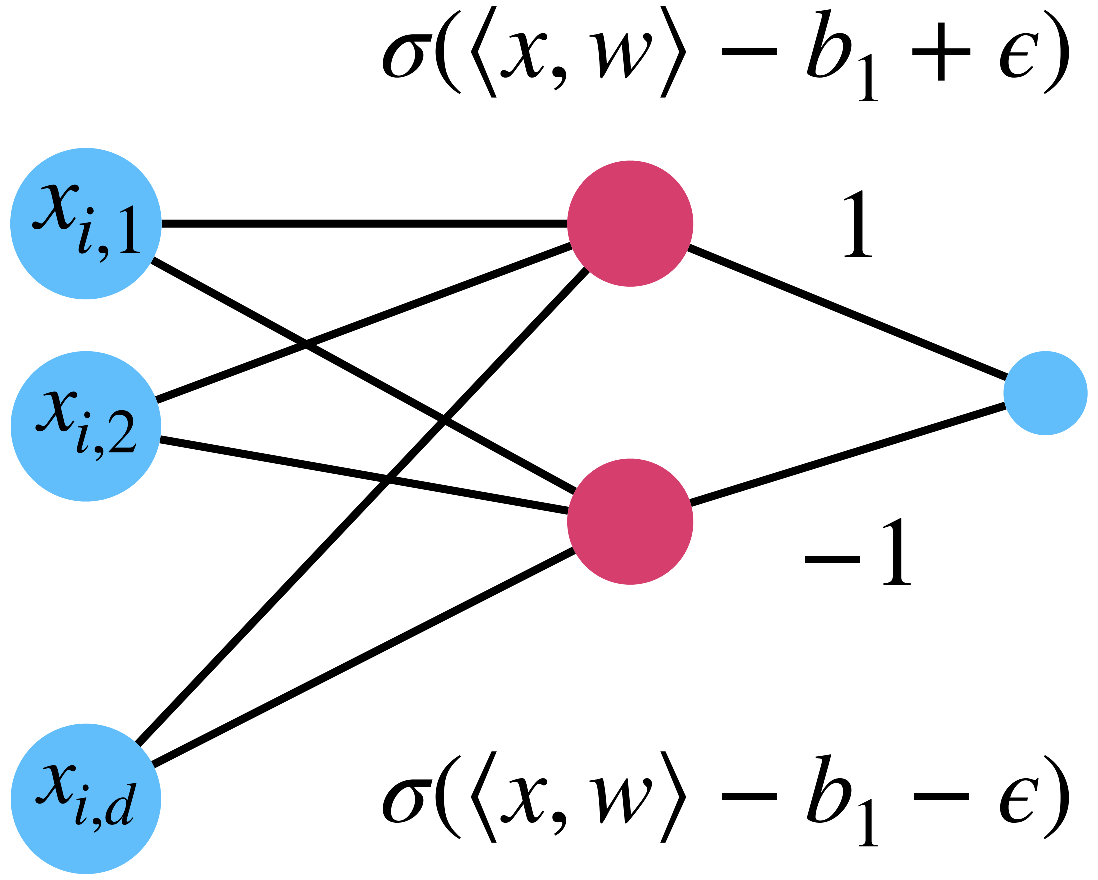
Let $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ such that no two data points are identical in feature space. Then, we can always create a threshold neural network that fits the data set.

Proof: How can I memorize a single data point?

- Let w be a d dimensional gaussian vector and $\langle x_i, w \rangle = b_i$. These b_is are unique, since $x_i \neq x_j$.
- w.l.og. assume that $\min_{i,j;i\neq j} |b_i - b_k| = 10\epsilon$

- Then here is a way to memorize a single label
- You can memorize 1 data point with 2 activations
- You can memorize n with $2n$ activations

and $O(nd)$ parameters



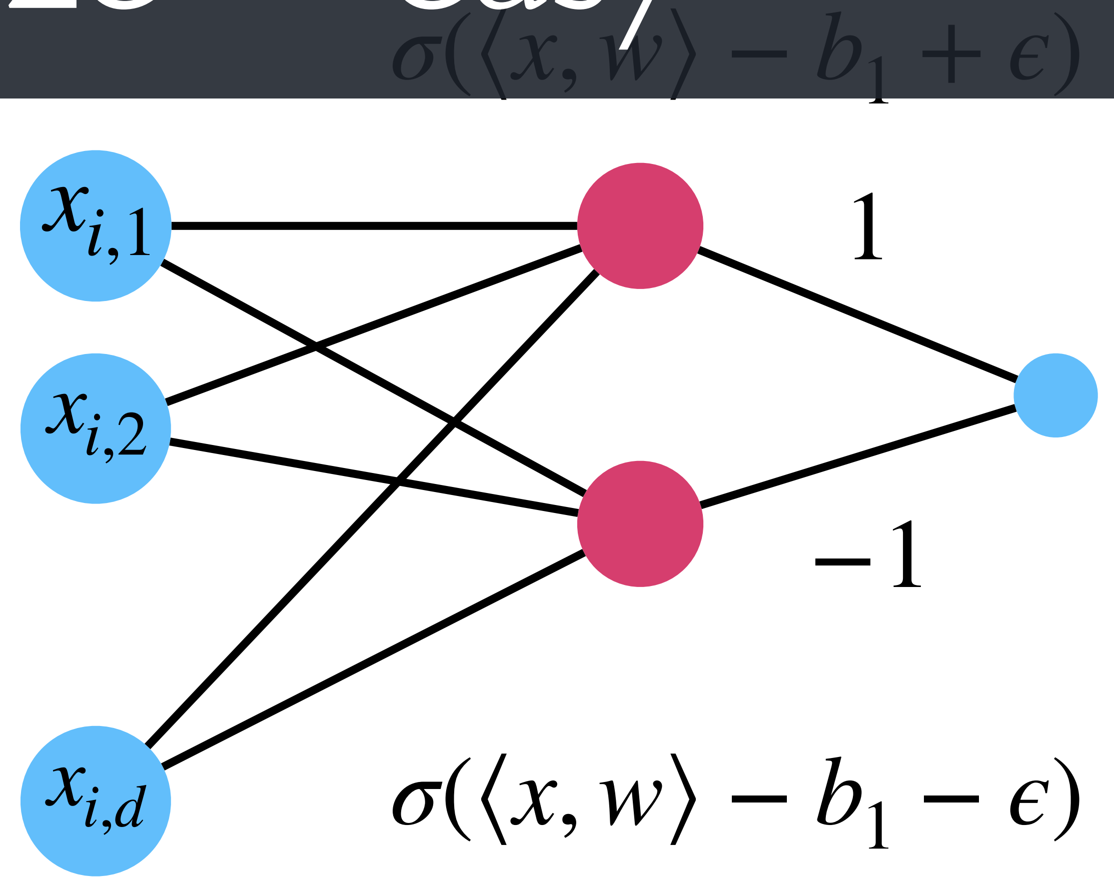
What about learning NNs of arbitrary size?

Theorem:

Let $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ such that no two data points are identical in feature space. Then, we can always create a threshold neural network that fits the data set.

Clearly possible to memorize any number of data points if network scales with the size of it.
=> Memorizing with scaling memory size = easy

- Then here is a way to memorize a single label
- You can memorize 1 data point with 2 activations
- You can memorize n with $2n$ activations and $O(nd)$ parameters



What about learning NNs of arbitrary size?

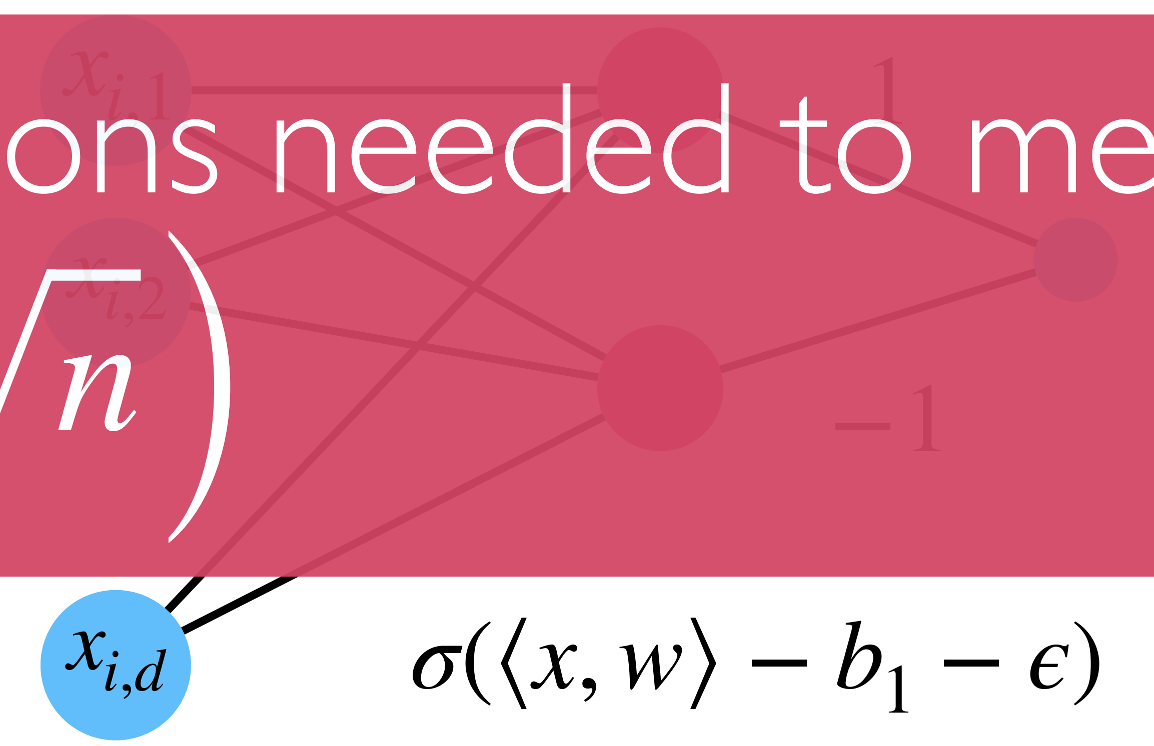
Theorem:

Let $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ such that no two data points are identical in feature space. Then, we can always create a threshold neural network that fits the data set.

Clearly possible to memorize any number of data points if network scales with the size of it.
=> Memorizing with scaling memory size = easy

Minimum number of weights / activations needed to memorize?

$O(n)$ and $O\left(\sqrt{n}\right)$



Hmm… one second

Minimum number of weights / activations needed to memorize?

$O(n)$ and $O\left(\sqrt{n}\right)$

Hmm… one second

Minimum number of weights / activations needed to memorize?

$O(n)$ and $O\left(\sqrt{n}\right)$

Taking into account parameter count bounds, this means that uniform type of generalization bounds are doomed

OK so, what can we do?

Let's revisit nice loss functions

First stop: Convexity

- "A function that looks like a bowl"

Def.:

A function $f(w)$ is convex on $\mathscr{W}$ if

$$f(a \cdot w + (1-a) \cdot w') \leq af(w) + (1-a)f(w')$$

First stop: Convexity

- "A function that looks like a bowl"

Def.:

A function $f(w)$ is convex on $\mathscr{W}$ if

$$f(a \cdot w + (1-a) \cdot w') \leq af(w) + (1-a)f(w')$$

- Convexity makes our lives much easier (more on next lecture).
- Most useful property (for us)

$$\langle \nabla f(w'), w' - w^* \rangle \geq f(w') - f(w^*)$$

gradient is always positively correlated with the right direction towards OPT

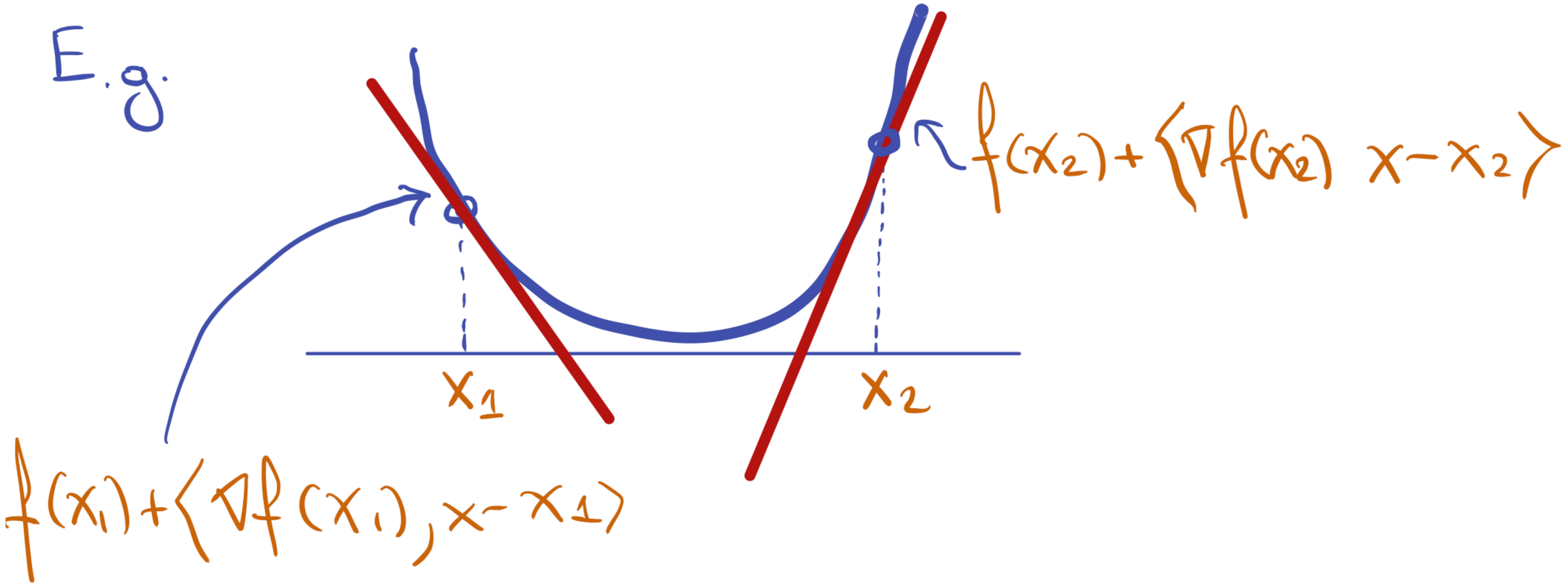
Let's get a bit more mileage from this

First stop: Convexity

- The first order Taylor expansion of a convex function is a "global under-estimate"
- $\forall w, w_0 \in \mathbb{R}^d,\ f(w) \geq f(w_0) + \langle \nabla f(w_0), w - w_0 \rangle$
-

First stop: Convexity

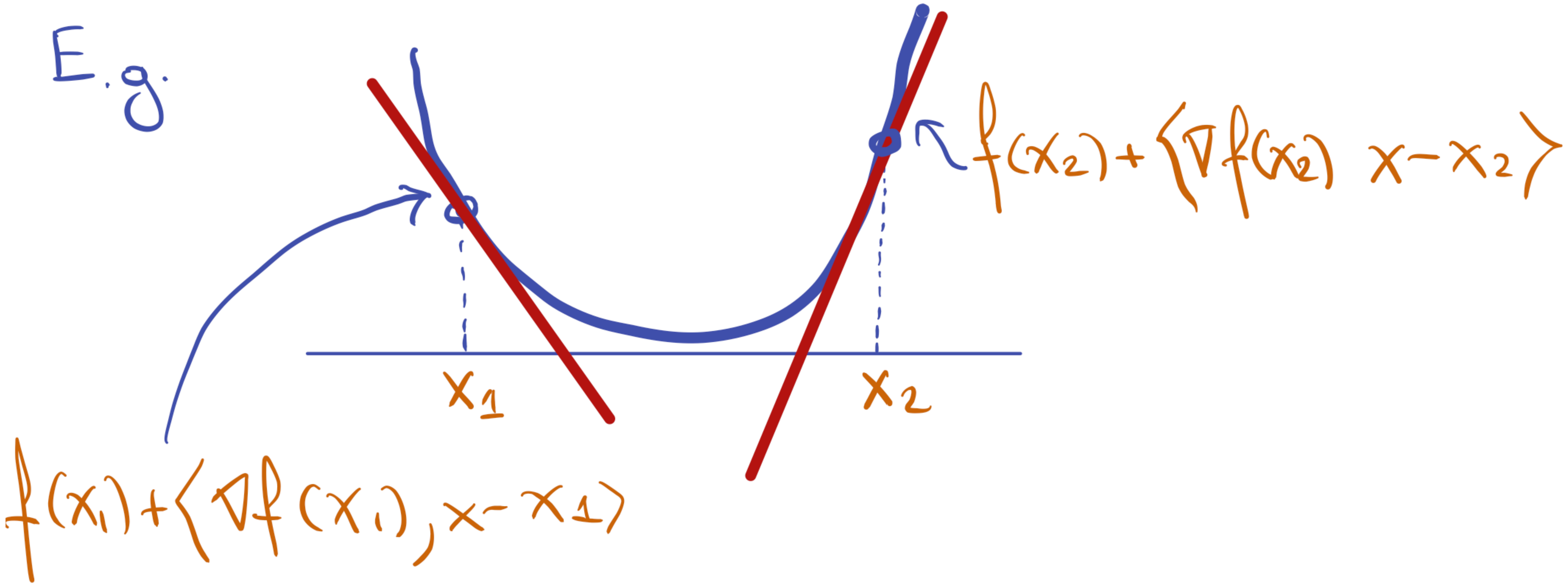
- The first order Taylor expansion of a convex function is a "global under-estimate"
- $\forall w, w_0 \in \mathbb{R}^d,\ f(w) \geq f(w_0) + \langle \nabla f(w_0), w - w_0 \rangle$



-

First stop: Convexity

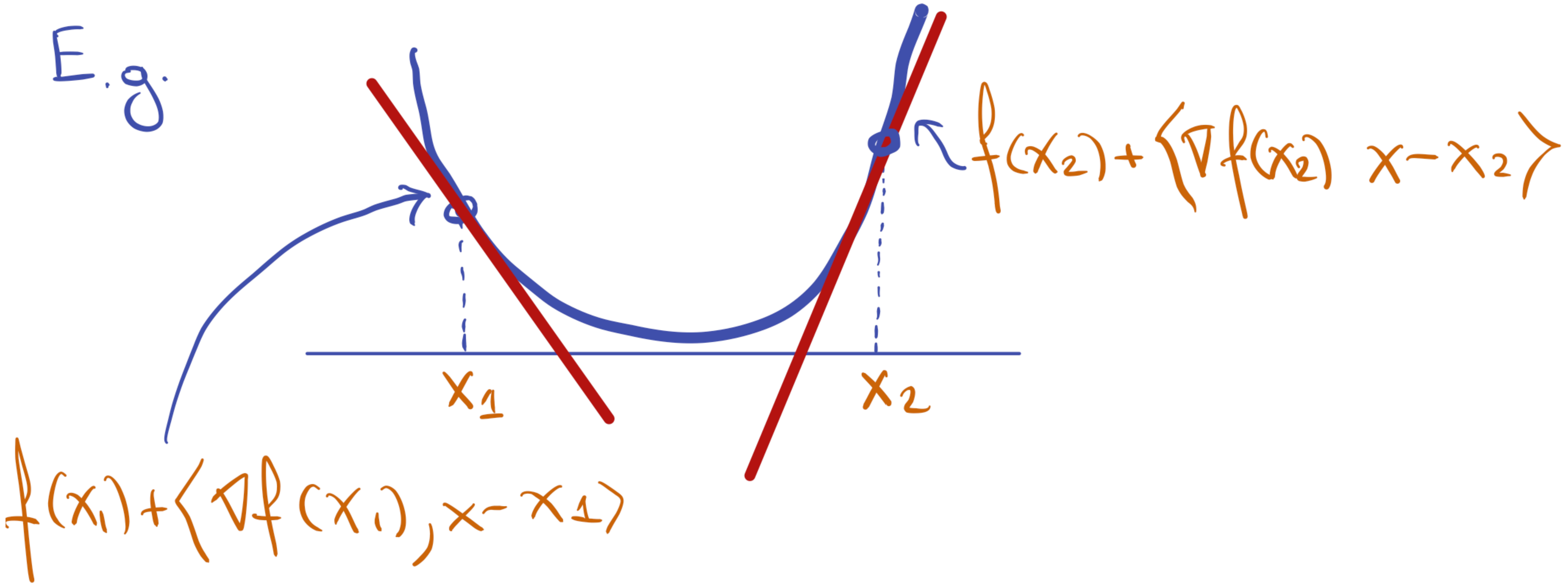
- The first order Taylor expansion of a convex function is a "global under-estimate"
- $\forall w, w_0 \in \mathbb{R}^d, \; f(w) \geq f(w_0) + \langle \nabla f(w_0), w - w_0 \rangle$



- Observe: 1-st order Taylor always has a linear form, e.g. , $f(w) \approx \langle w, a \rangle + b$

First stop: Convexity

- The first order Taylor expansion of a convex function is a "global under-estimate"
- $\forall w, w_0 \in \mathbb{R}^d, \;\; f(w) \geq f(w_0) + \langle \nabla f(w_0), w - w_0 \rangle$



- Observe: 1-st order Taylor always has a linear form, e.g. , $f(w) \approx \langle w, a \rangle + b$

Q: what happens for $w_0 \;\; s.t. \;\; \nabla f(w_0) = 0$?

First stop: Convexity

- Q: what happens for $w_0 \;\; s.t. \;\; \nabla f(w_0) = 0$?
- We have $f(w) \geq f(w_0) + \langle \nabla f(w_0), w - w_0 \rangle \Rightarrow f(w) \geq f(w_0)$

First stop: Convexity

- Q: what happens for $w_0 \;\; s.t. \;\; \nabla f(w_0) = 0$?
- We have $f(w) \geq f(w_0) + \langle \nabla f(w_0), w - w_0 \rangle \Rightarrow f(w) \geq f(w_0)$

- That is, all points $w_0 \;\; s.t. \;\; \nabla f(w_0) = 0$ are <u>global minimizers.</u>

First stop: Convexity

- Q: what happens for $w_0 \;\; s.t. \;\; \nabla f(w_0) = 0$?
- We have $f(w) \geq f(w_0) + \langle \nabla f(w_0), w - w_0 \rangle \Rightarrow f(w) \geq f(w_0)$

- That is, all points $w_0 \;\; s.t. \;\; \nabla f(w_0) = 0$ are <u>global minimizers.</u>

Q: Using the above properties, can we use devise an algorithm for $\min_{w \in \mathcal{W}} f(w)$, when the function is convex?

First stop: Convexity

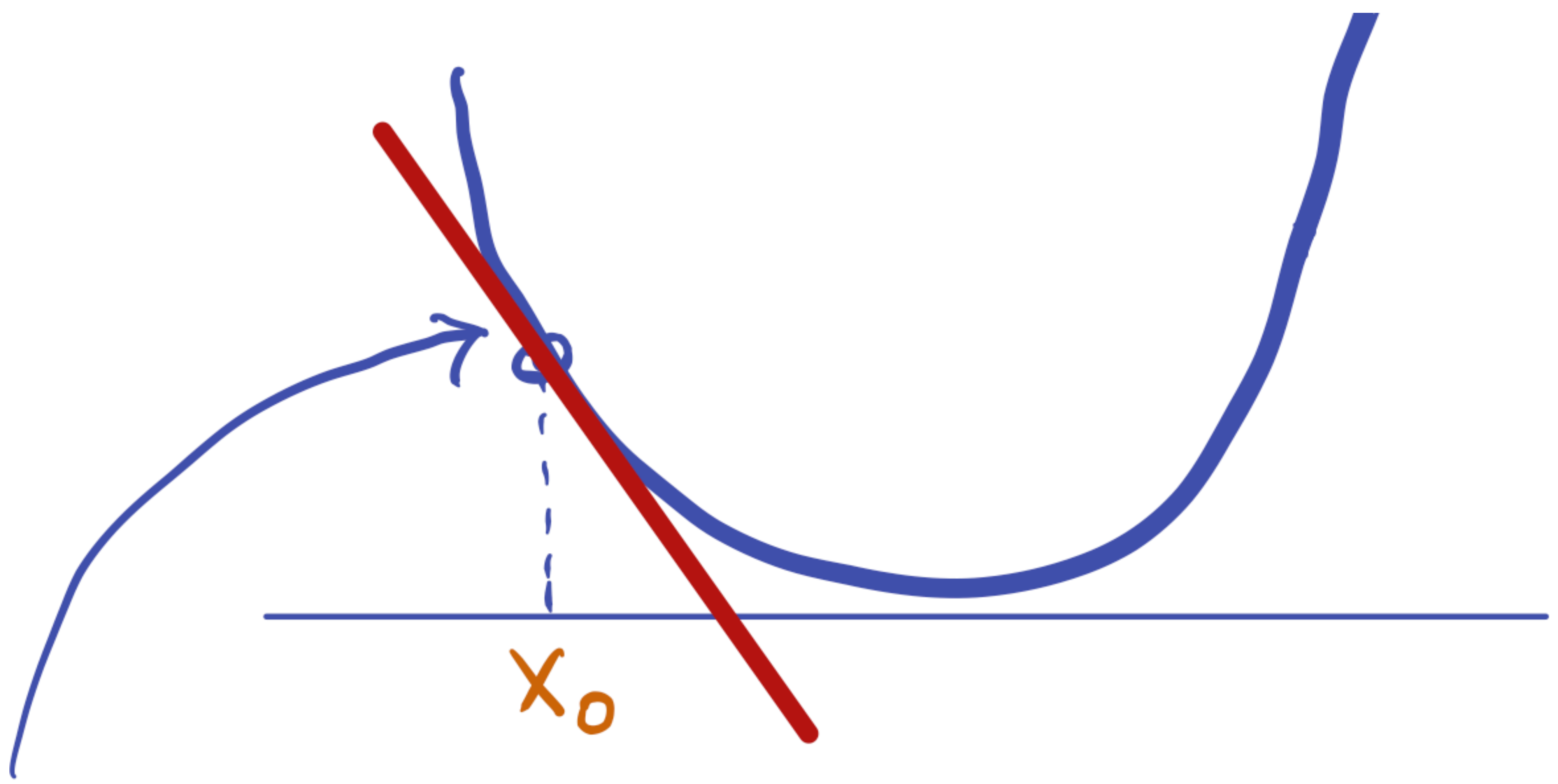
- Q: what happens for $w_0 \;\; s.t. \;\; \nabla f(w_0) = 0$?
- We have $f(w) \geq f(w_0) + \langle \nabla f(w_0), w - w_0 \rangle \Rightarrow f(w) \geq f(w_0)$

- That is, all points $w_0 \;\; s.t. \;\; \nabla f(w_0) = 0$ are global minimizers.

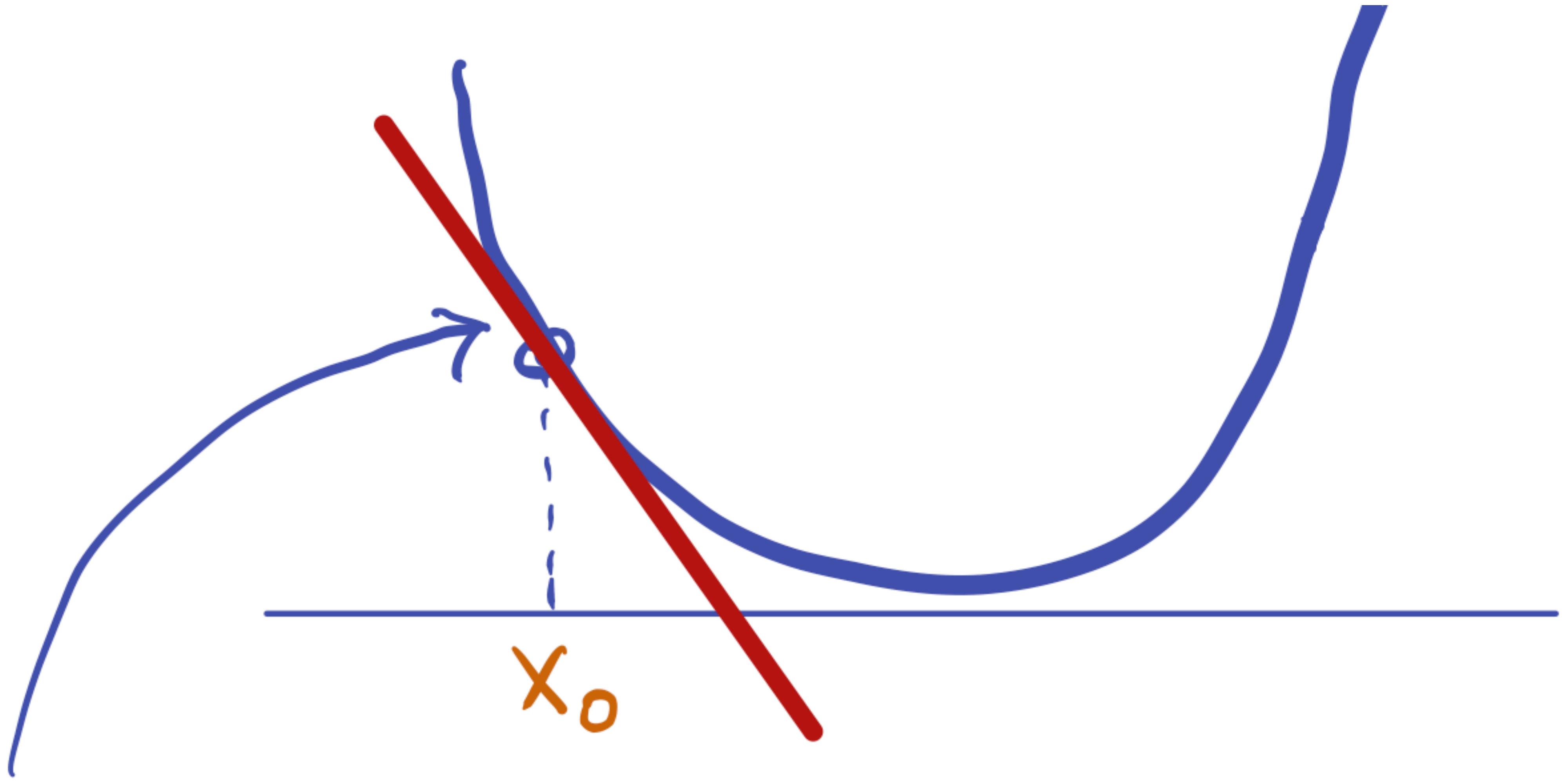
Q: Using the above properties, can we use devise an algorithm for $\min_{w \in \mathcal{W}} f(w)$, when the function is convex?

Generally, one can approx. solve convex problems with complexity with the ellipsoid method (very expensive/iteration)

A simple idea

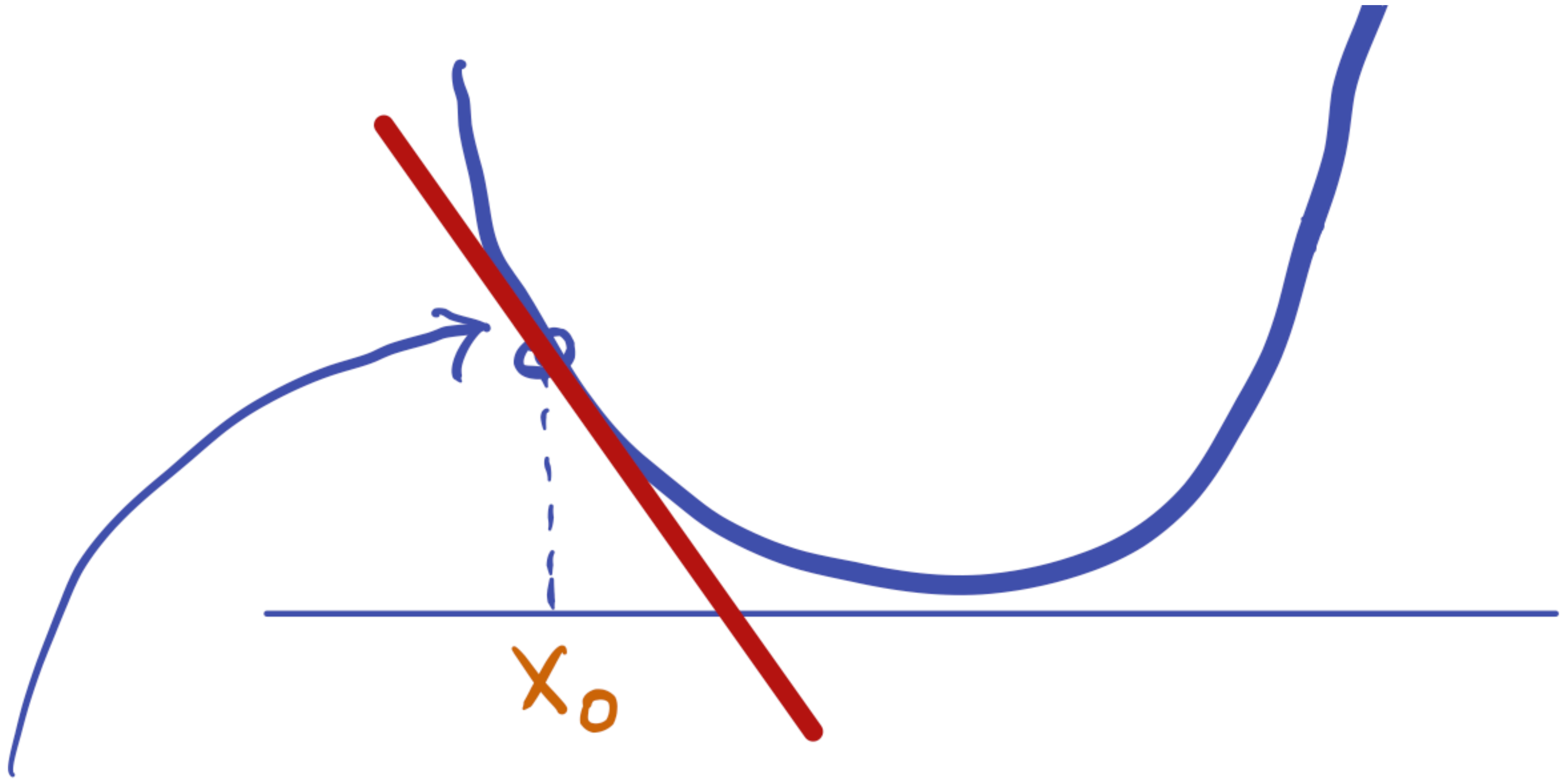


A simple idea



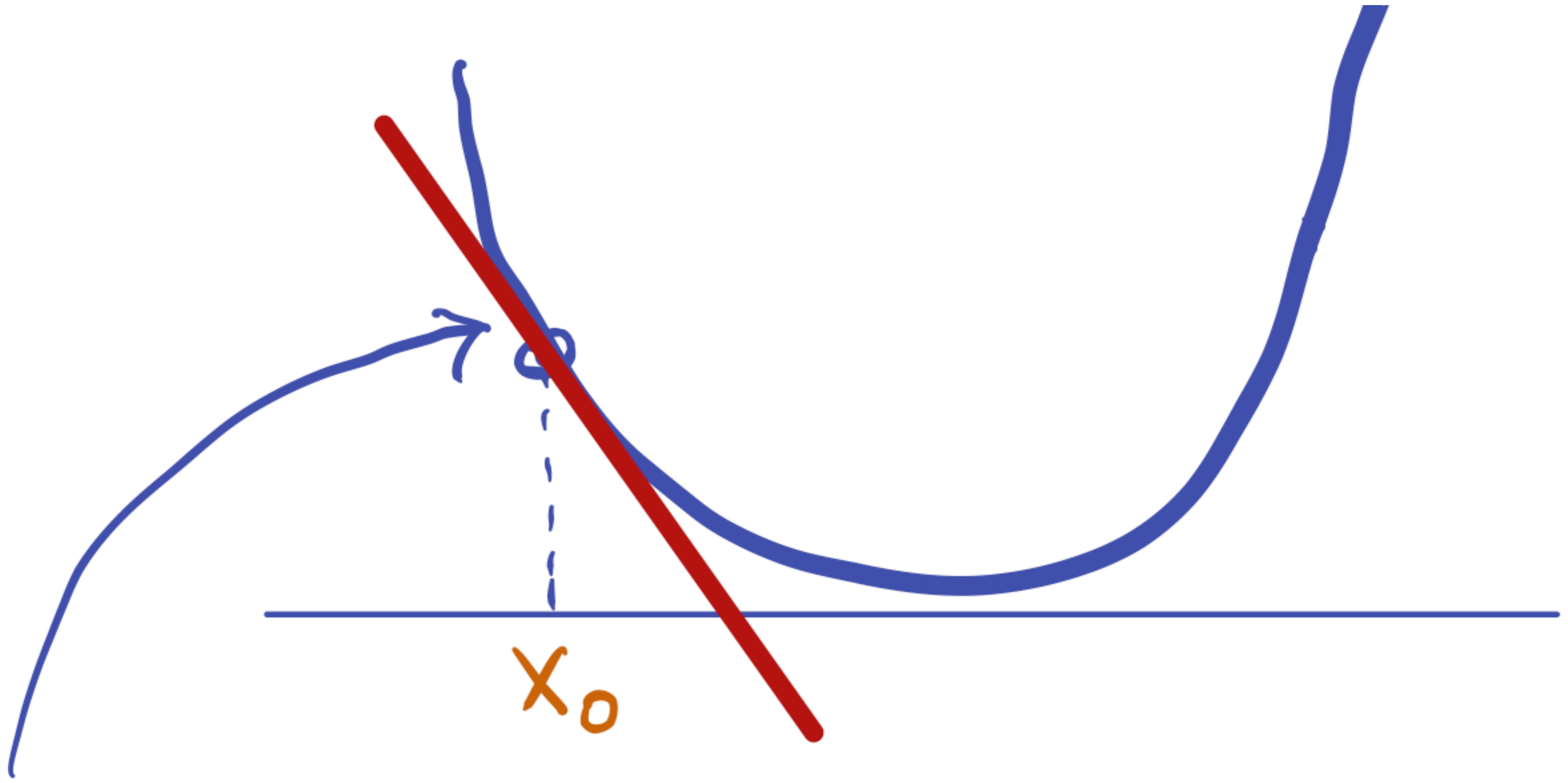
- Say we initialize at w_0, then we could try to follow the "line" $f(w_0) + \langle f(w_0), w - w_0 \rangle$!

A simple idea



- Say we initialize at w_0, then we could try to follow the "line" $f(w_0) + \langle f(w_0), w - w_0 \rangle$!
- Q: But for how long? If we tried to just minimize the linear under-estimator, we'd go to -infty

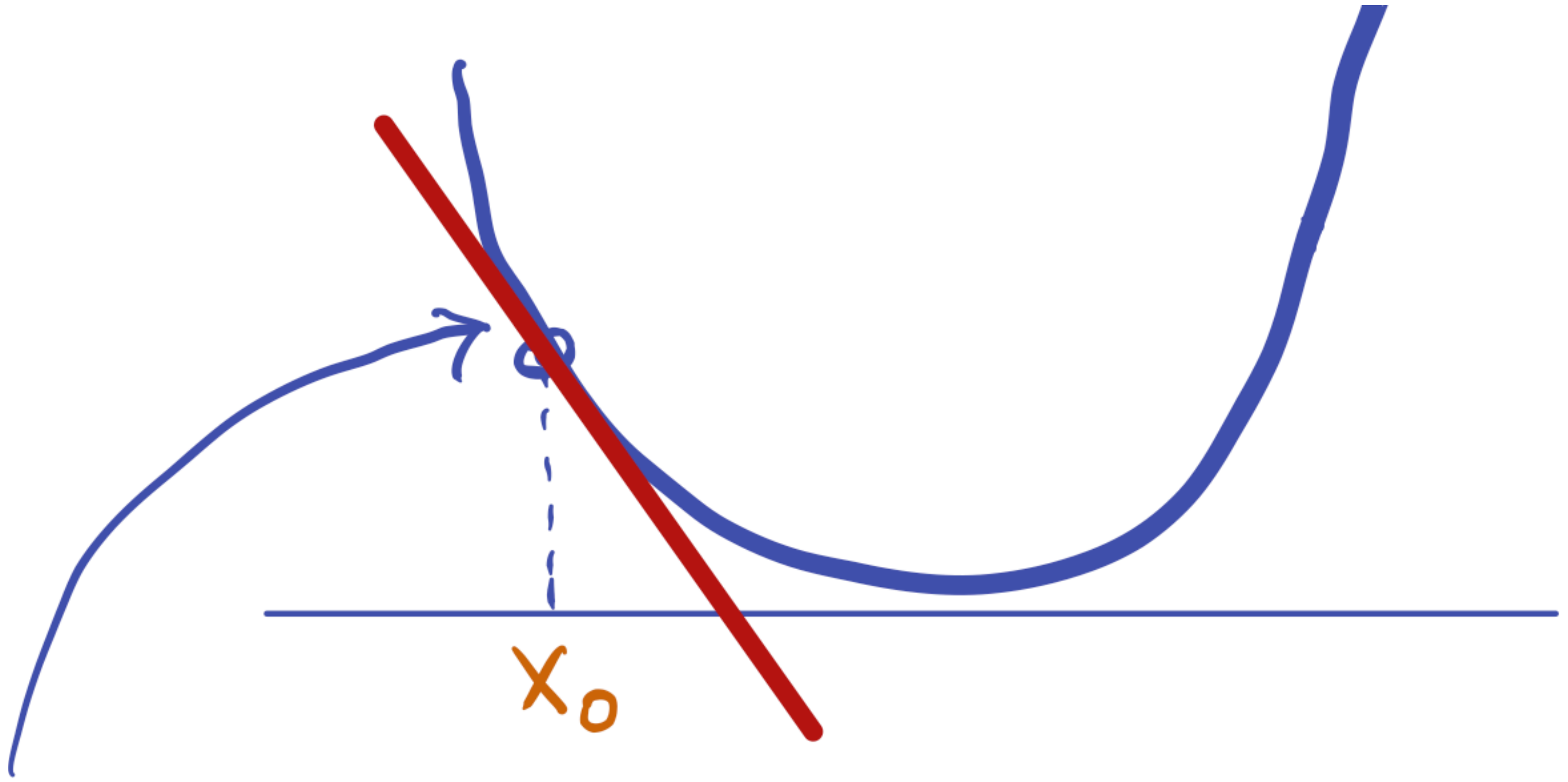
A simple idea



- Say we initialize at w_0, then we could try to follow the "line" $f(w_0) + \langle f(w_0), w - w_0 \rangle$!
- Q: But for how long? If we tried to just minimize the linear under-estimator, we'd go to -infty

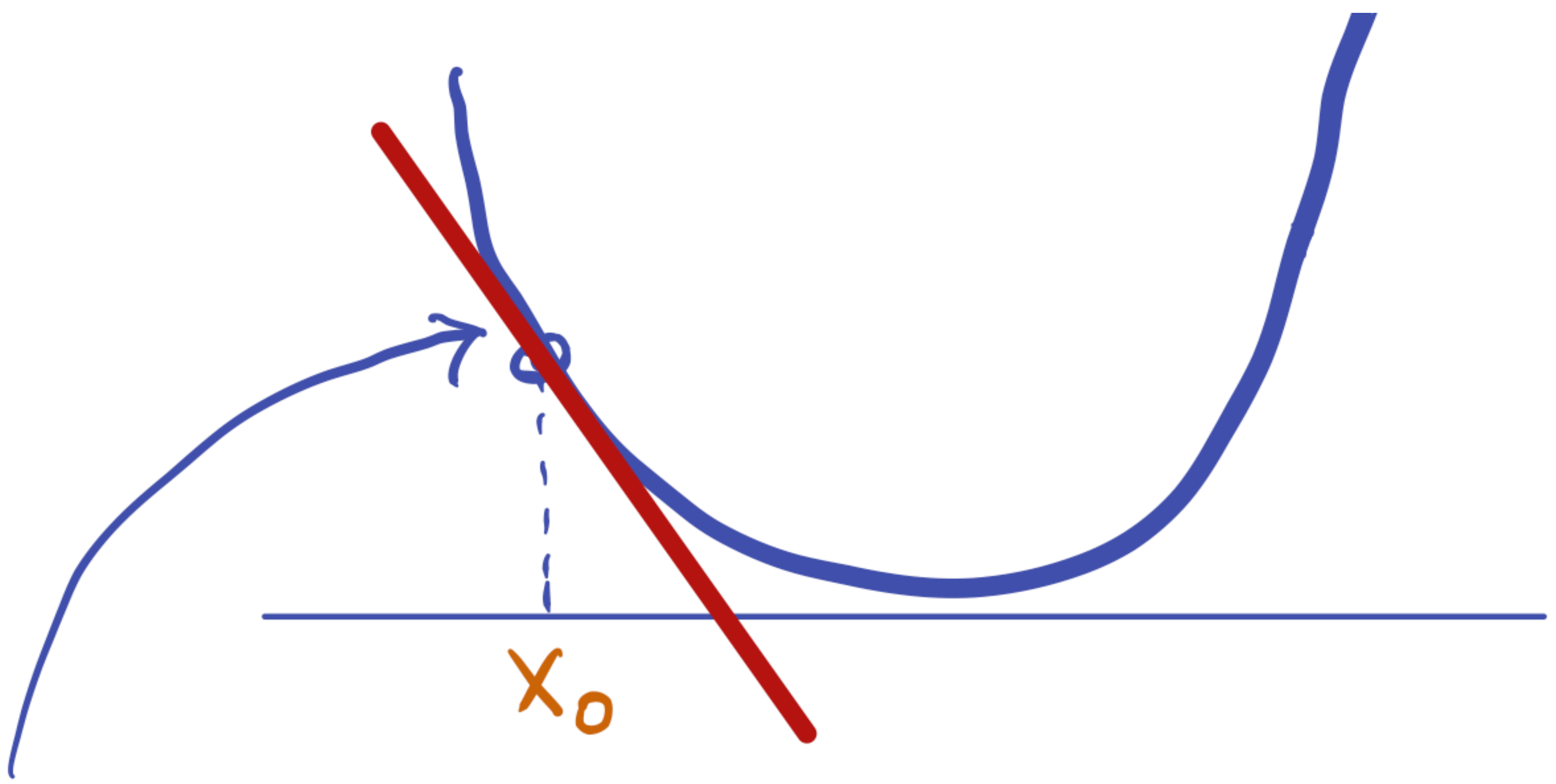
Clue: Follow the target line, but take a small step!

Additive updates



- Say we initialize at w_0, then we could try to follow the "line" $f(w_0) + \langle f(w_0), w - w_0 \rangle$!

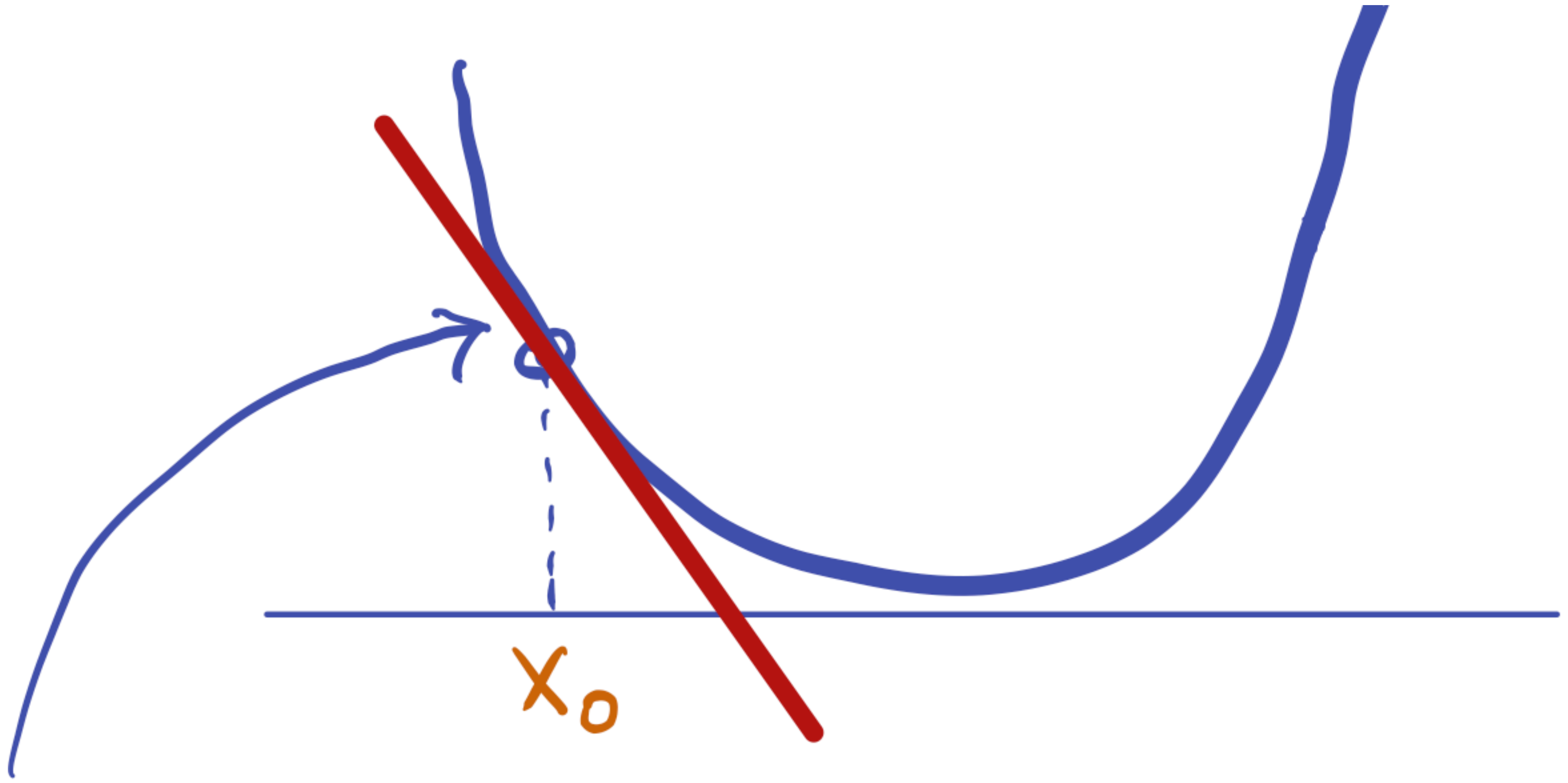
Additive updates



- Say we initialize at w_0, then we could try to follow the "line" $f(w_0) + \langle f(w_0), w - w_0 \rangle$!
- Let's make our algorithm to progress by additive steps, i.e.,

$$w_{k+1} = w_k + u_k$$

Additive updates

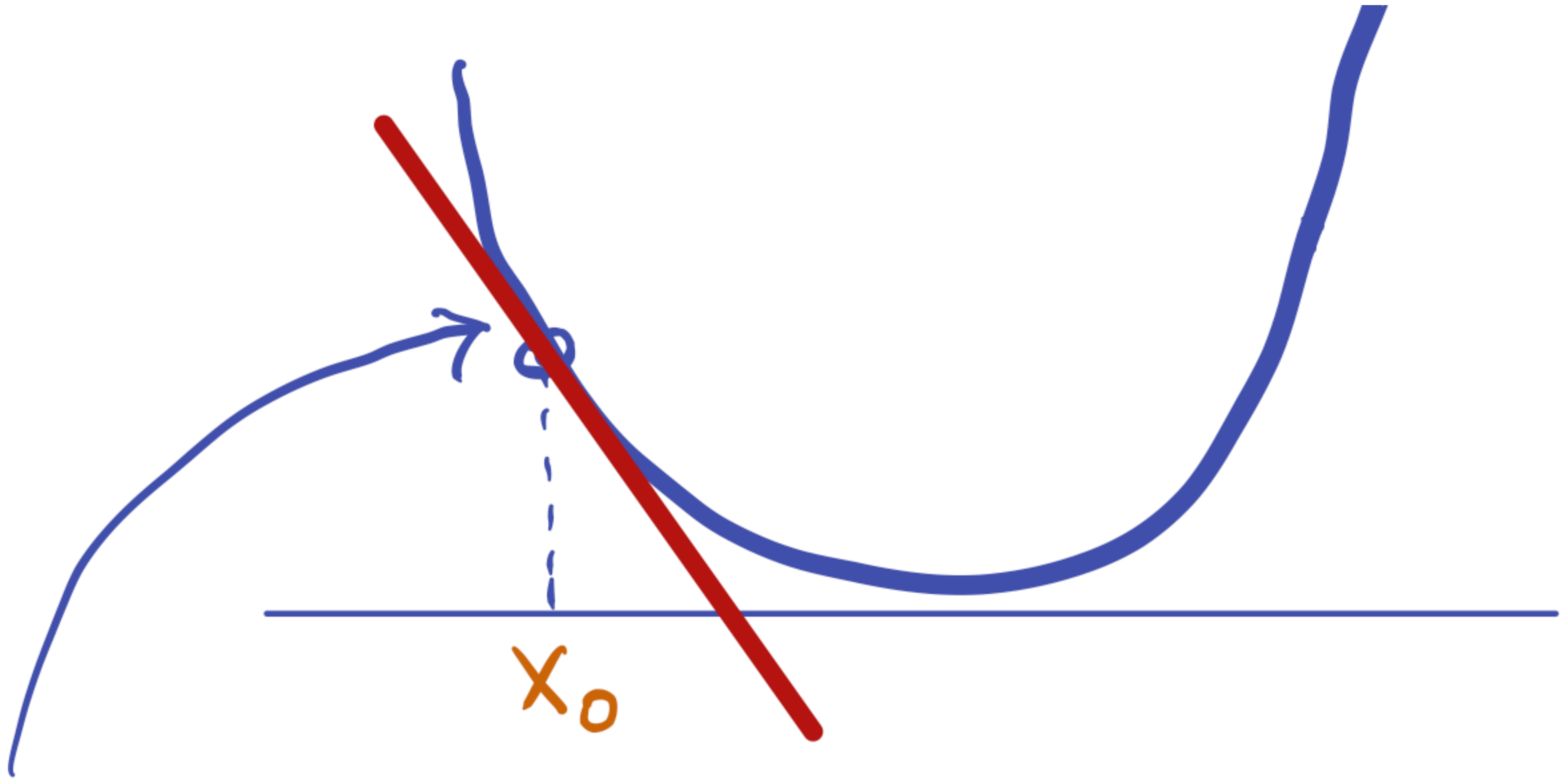


- Say we initialize at w_0, then we could try to follow the "line" $f(w_0) + \langle f(w_0), w - w_0 \rangle$!
- Let's make our algorithm to progress by additive steps, i.e.,

$$w_{k+1} = w_k + u_k$$

- Goal? $\|\nabla f(w_\infty)\| = 0$

Additive updates



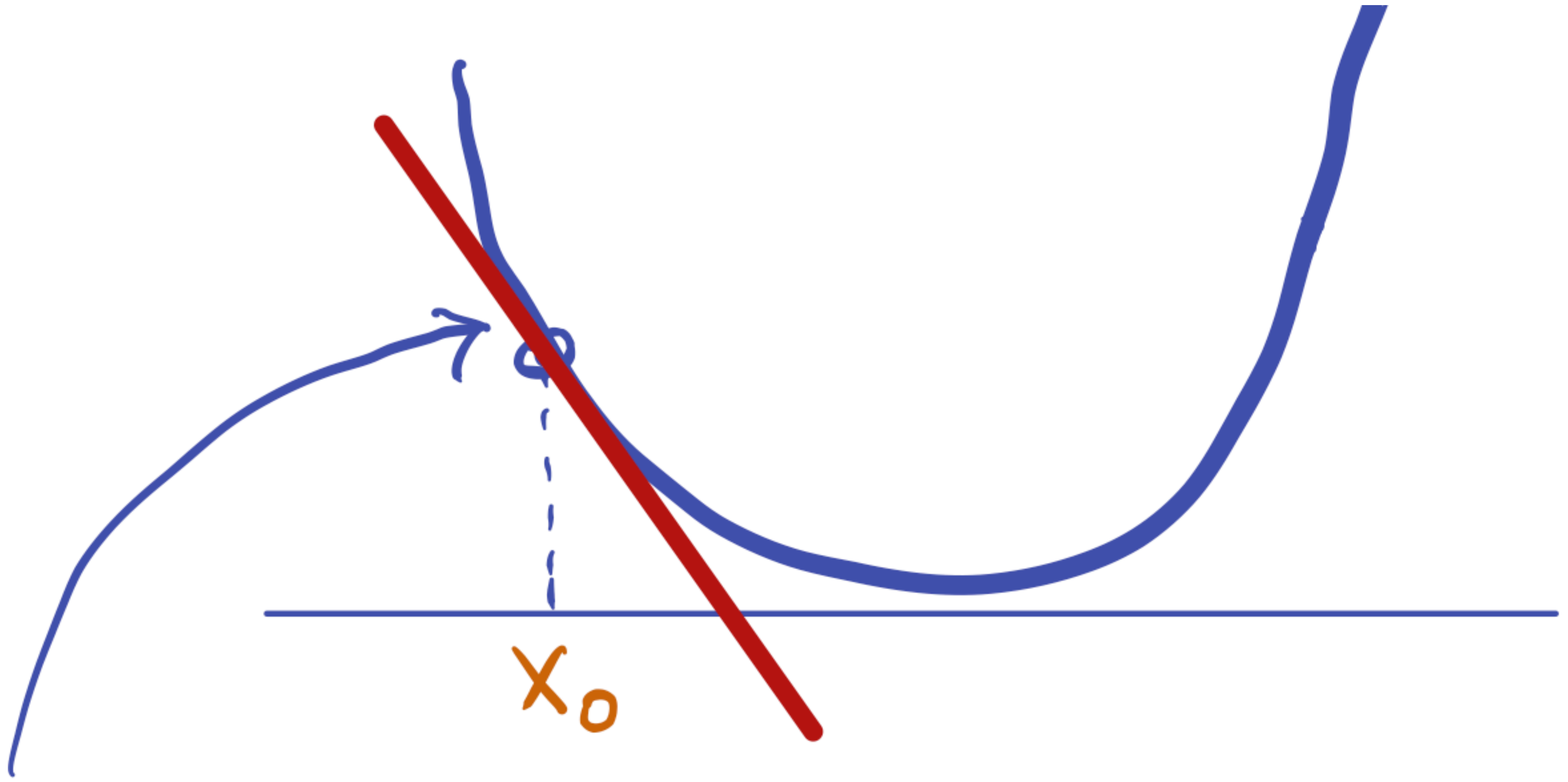
- Say we initialize at w_0, then we could try to follow the "line" $f(w_0) + \langle f(w_0), w - w_0 \rangle$!
- Let's make our algorithm to progress by additive steps, i.e.,

$$w_{k+1} = w_k + u_k$$

- Goal? $\|\nabla f(w_\infty)\| = 0$

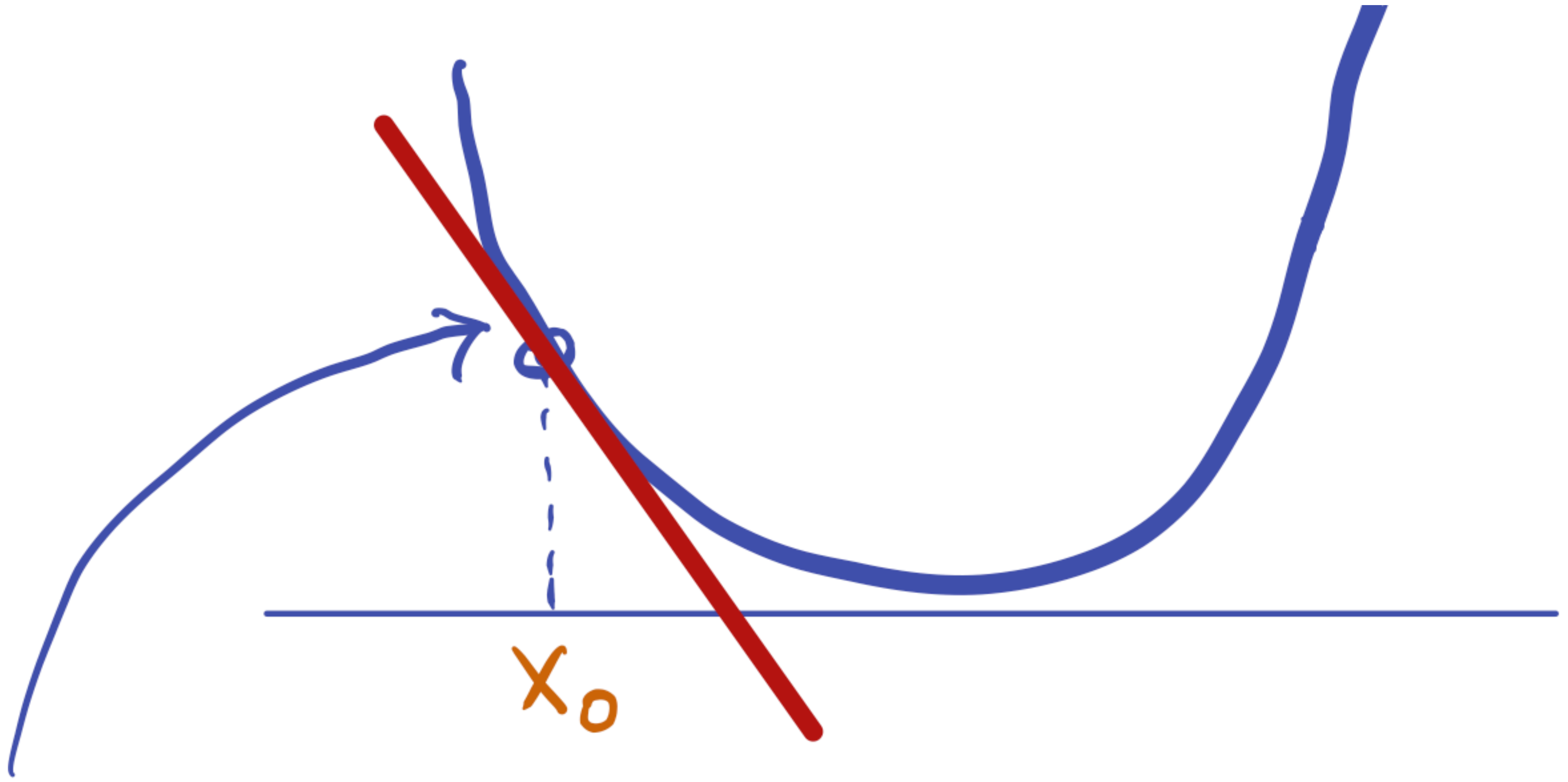
Clue: Minimize the linear approximation above, but not too much

Local optimization



- Say we initialize at w_0, then we could try to follow the "line" $f(w_0) + \langle f(w_0), w - w_0 \rangle$!

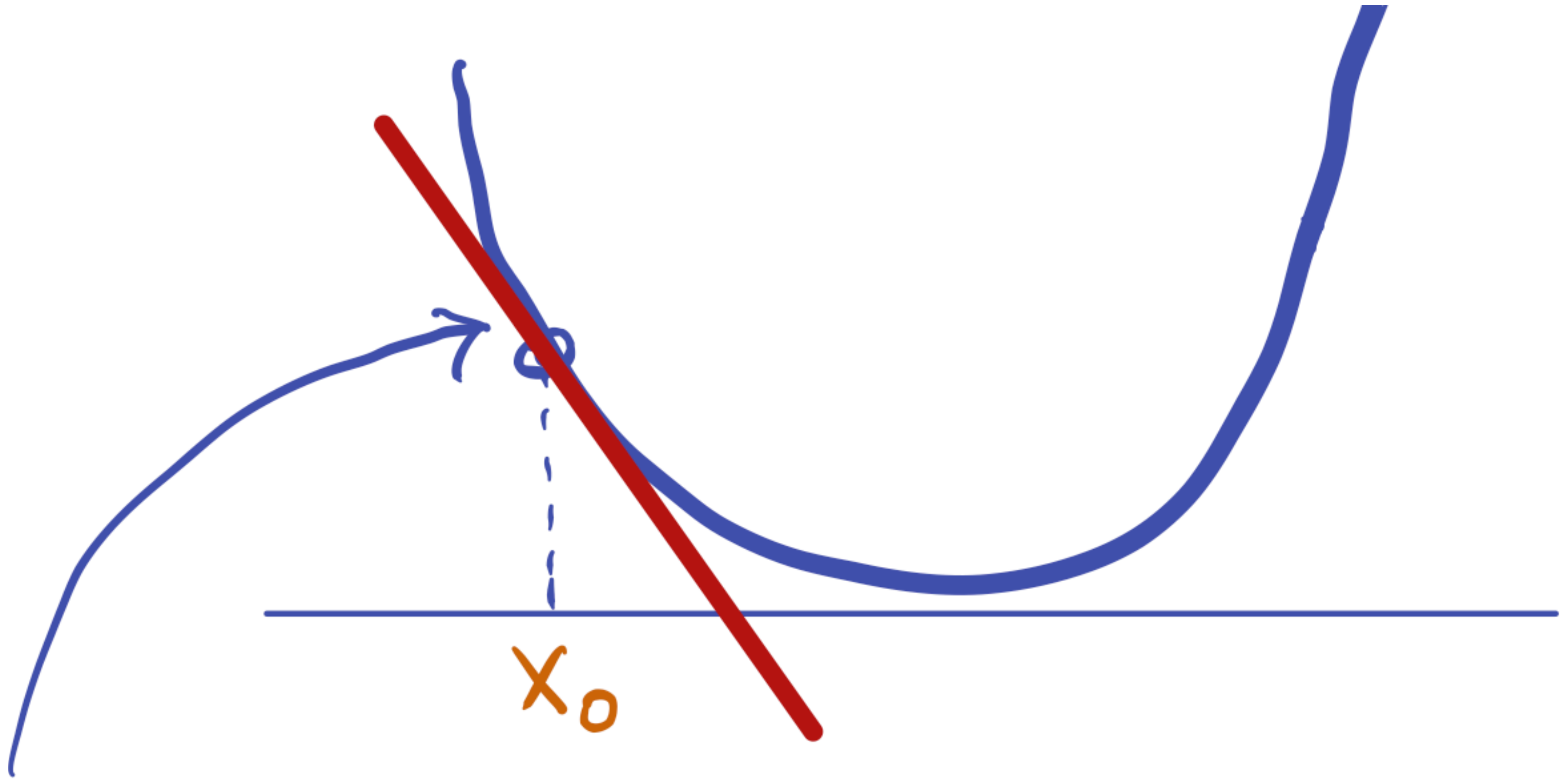
Local optimization



- Say we initialize at w_0, then we could try to follow the "line" $f(w_0) + \langle f(w_0), w - w_0 \rangle$!
- Let's make our algorithm to progress by additive steps, i.e.,

$$w_{k+1} = \arg\min_{w \in \mathscr{W}} \left\{ f(w_k) + \langle \nabla f(w_k, w - w_k \rangle + \frac{1}{2\gamma} \|w - w_k\|^2 \right\}$$

Local optimization

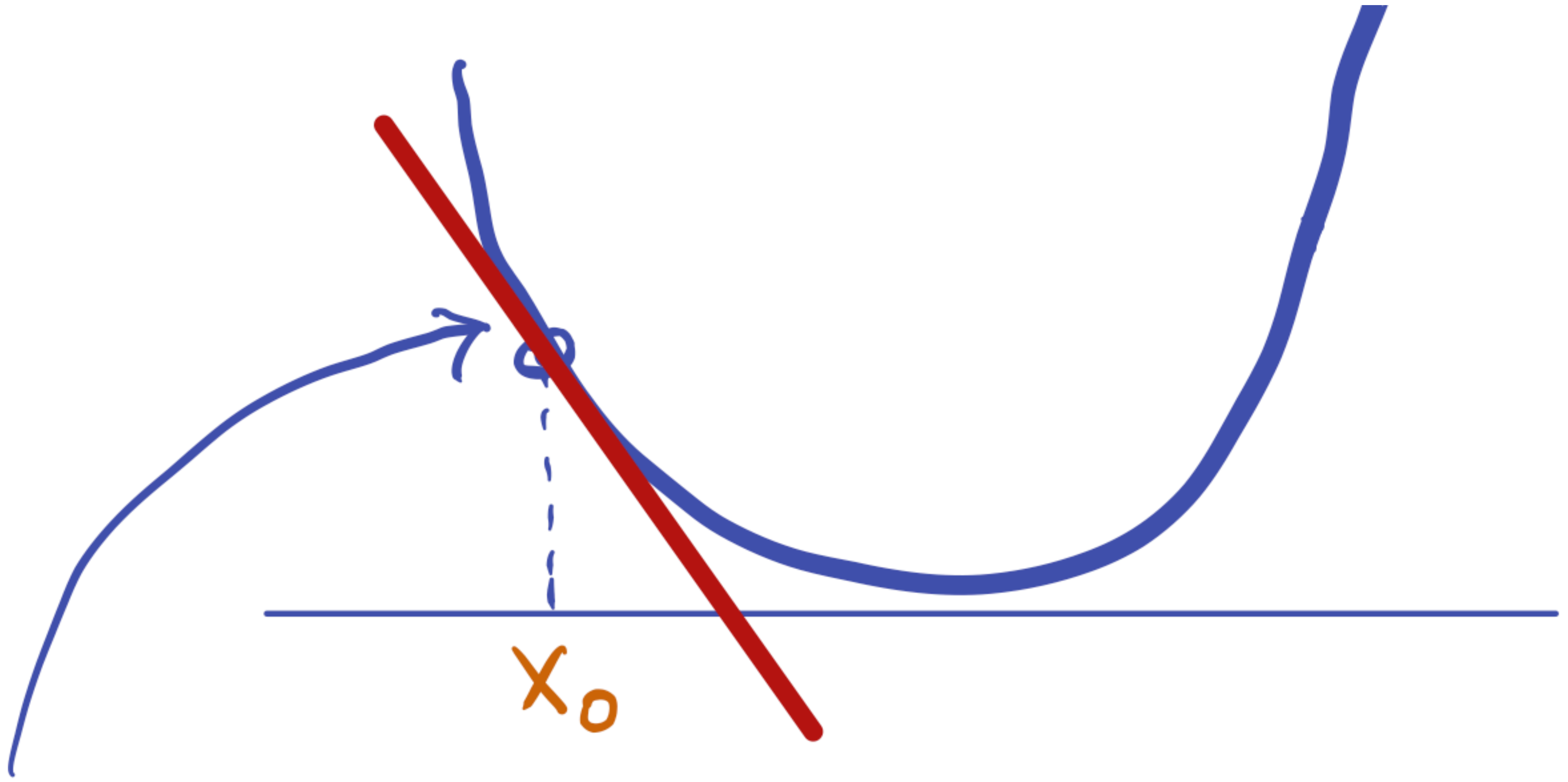


- Say we initialize at w_0, then we could try to follow the "line" $f(w_0) + \langle f(w_0), w - w_0 \rangle$!
- Let's make our algorithm to progress by additive steps, i.e.,

$$w_{k+1} = \arg\min_{w \in \mathscr{W}} \left\{ f(w_k) + \langle \nabla f(w_k, w - w_k \rangle + \frac{1}{2\gamma} \|w - w_k\|^2 \right\}$$

Think of w_{k+1} the best update, near w_k

Local optimization



- Say we initialize at w_0, then we could try to follow the "line" $f(w_0) + \langle f(w_0), w - w_0 \rangle$!
- Let's make our algorithm to progress by additive steps, i.e.,

$$w_{k+1} = \arg\min_{w \in \mathscr{W}} \left\{ f(w_k) + \langle \nabla f(w_k, w - w_k \rangle + \frac{1}{2\gamma} \|w - w_k\|^2 \right\}$$

Think of w_{k+1} the best update, near w_k

what is the OPT solution for the above quadratic problem?

Additive updates

- Let's solve this, by setting grad to zero!

Additive updates

- Let's solve this, by setting grad to zero!

$$\nabla_w \left\{ f(w_k) + \langle \nabla f(w_k, w - w_k \rangle + \frac{1}{2\gamma} \|w - w_k\|^2 \right\} = 0$$

$$\Rightarrow \nabla_w \left\{ \langle \nabla f(w_k, w - w_k \rangle + \frac{1}{2\gamma} \|w - w_k\|^2 \right\} = 0$$

$$\Rightarrow \nabla f(w_k) + \frac{1}{\gamma}(w - w_k) = 0$$

$$\Rightarrow w_{k+1} = w_k - \gamma \nabla f(w_k)$$

Additive updates

- Let's solve this, by setting grad to zero!

$$\nabla_w \left\{ f(w_k) + \langle \nabla f(w_k, w - w_k \rangle + \frac{1}{2\gamma} \|w - w_k\|^2 \right\} = 0$$

$$\Rightarrow \nabla_w \left\{ \langle \nabla f(w_k, w - w_k \rangle + \frac{1}{2\gamma} \|w - w_k\|^2 \right\} = 0$$

$$\Rightarrow \nabla f(w_k) + \frac{1}{\gamma}(w - w_k) = 0$$

$$\Rightarrow w_{k+1} = w_k - \gamma \nabla f(w_k)$$

Ha! We just derived Gradient Descent!

Additive updates

- Let's solve this, by setting grad to zero!

$$\nabla_w \left\{ f(w_k) + \langle \nabla f(w_k, w - w_k \rangle + \frac{1}{2\gamma}\|w - w_k\|^2 \right\} = 0$$

$$\Rightarrow \nabla_w \left\{ \langle \nabla f(w_k, w - w_k \rangle + \frac{1}{2\gamma}\|w - w_k\|^2 \right\} = 0$$

$$\Rightarrow \nabla f(w_k) + \frac{1}{\gamma}(w - w_k) = 0$$

$$\Rightarrow w_{k+1} = w_k - \gamma \nabla f(w_k)$$

Ha! We just derived Gradient Descent!

How fast does GD converge? Next time!
Short answer, depends on the function!

Wrapping up

- Minimizing training loss is hard in general

- ERM is hard for neural networks
- ERM is not hard if you are allowed to change architecture

- Towards reasonable algorithms, Step 0: Convexity

- Step 1: Gradient Descent

Next Time:
Convergence rates of GD +
intro to SGD, the simplest learning algorithm

reading list

Bubeck, S., 2015. Convex Optimization: Algorithms and Complexity. Foundations and Trends® in Machine Learning, 8(3-4), pp.231-357.
https://arxiv.org/pdf/1405.4980.pdf

Judd, S., 1988. On the complexity of loading shallow neural networks. Journal of Complexity, 4(3), pp.177-192.
https://tinyurl.com/44snxxn6

Šíma, J., 1994. Loading deep networks is hard. Neural Computation, 6(5), pp.842-850.
Vancouver
https://direct.mit.edu/neco/article/6/5/842/5816/Loading-Deep-Networks-Is-Hard

Goel, S., Klivans, A., Manurangsi, P. and Reichman, D., 2020. Tight hardness results for training depth-2 ReLU networks. arXiv preprint arXiv:2011.13550.
https://arxiv.org/pdf/2011.13550.pdf

Manurangsi, P. and Reichman, D., 2018. The computational complexity of training relu (s). arXiv preprint arXiv:1810.04207.
https://arxiv.org/pdf/1810.04207.pdf

Baum, E.B., 1988. On the capabilities of multilayer perceptrons. Journal of complexity, 4(3), pp.193-215. Vancouver
https://tinyurl.com/yckawxha

Rajput, S., Sreenivasan, K., Papailiopoulos, D. and Karbasi, A., 2021. An exponential improvement on the memorization capacity of deep threshold networks. Advances in Neural Information Processing Systems, 34.
https://proceedings.neurips.cc/paper/2021/file/69dd2eff9b6a421d5ce262b093bdab23-Paper.pdf