

Optimization fundamentals

L101: Machine Learning for Language Processing
Andreas Vlachos



Previous lecture

Logistic regression parameter learning:

$$w^{\star} = \arg \min_w \sum_{(x,y) \in D} -y \log \sigma(w \cdot \phi(x)) - (1 - y) \log(1 - \sigma(w \cdot \phi(x)))$$

Supervised machine learning algorithms typically involve optimizing a loss over the training data:

$$w^{\star} = \arg \min_w L(w; \mathcal{D}), w \in \mathfrak{R}^k$$

This is an instance of **numerical optimization**, i.e. optimize the value of a function with respect to some parameters.

A scientific field of its own; this lecture just gives some useful pointers

Types of optimization problems

Continuous: $x^{\star} = \arg \min_x f(x), x \in \mathfrak{R}^k$

Discrete: $x^{\star} = \arg \min_x L(x), x \in \mathbb{Z}^k$

Sounds rare in NLP?

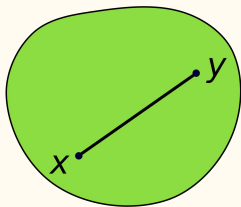
Inference in classification/text generation: a label/word is either predicted or not

Constraints: $x^{\star} = \arg \min_x L(x), c(x) \geq 0$

Examples: enforcing constraints on the text generated, output graph, etc.

Convexity

For sets:

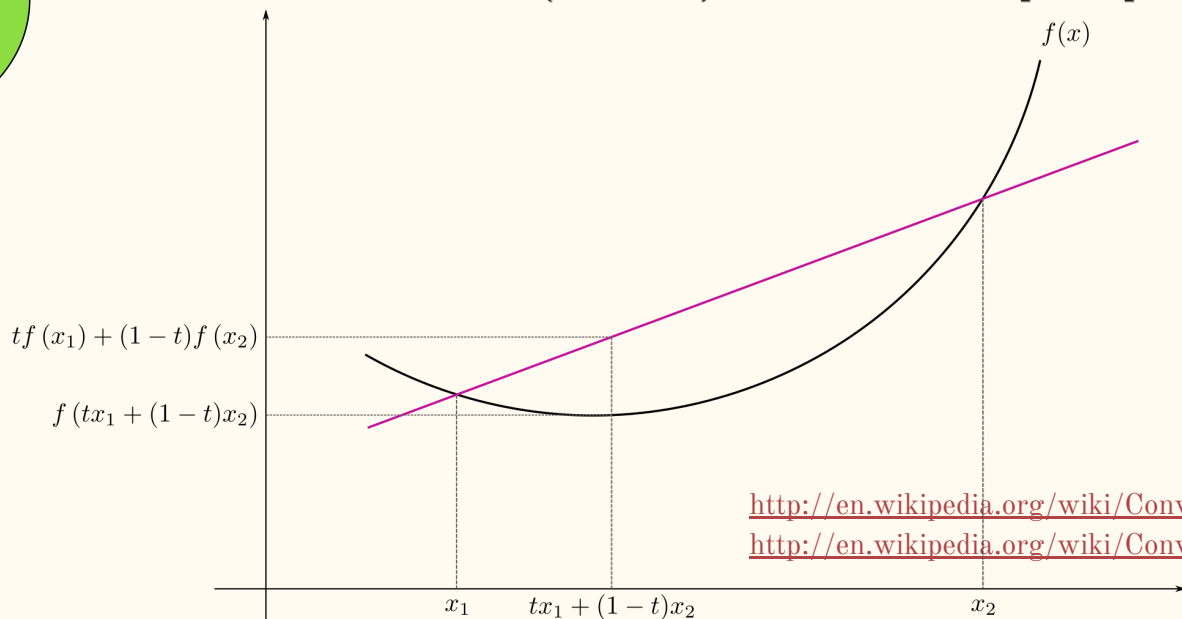


$$\forall x, y \in S : ax + (1 - a)y \in S, a \in [0, 1]$$

For functions:

If f concave, $-f$ is
convex

For sets the
relation is more
complicated



http://en.wikipedia.org/wiki/Convex_set,
http://en.wikipedia.org/wiki/Convex_function

$$f(tx_1 + (1 - t)x_2) \leq tf(x_1) + (1 - t)f(x_2), t \in [0, 1]$$

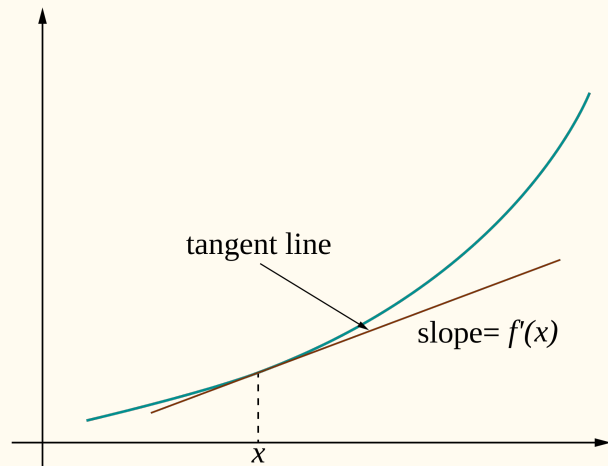
Derivatives (refresher)

Derivative at a point $\mathbf{x}$ is the slope of the tangent line on the function $\mathbf{f}$

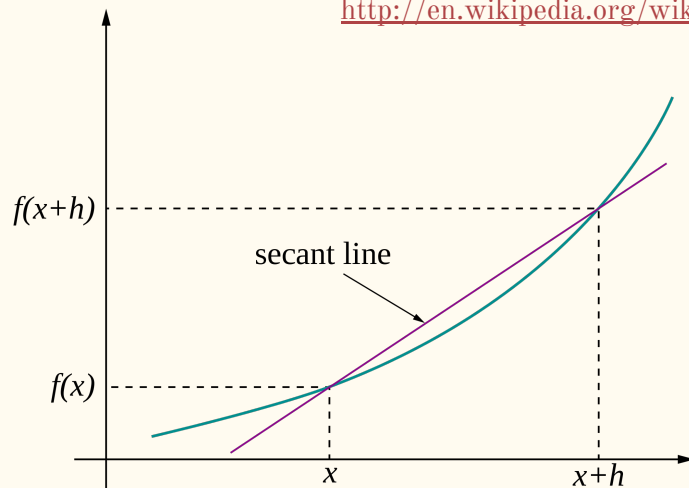
Best linear approximation of $\mathbf{f}$ near $\mathbf{x}$

Defined as this quotient:

$$\lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$



<http://en.wikipedia.org/wiki/Derivative>



Taylor's theorem

For a function f that is continuously differentiable, there is t such that:

$$f(x + p) = f(x) + \nabla f(x + tp)p, t \in (0, 1)$$

If twice differentiable:

$$f(x + p) = f(x) + \nabla f(x)p + \frac{1}{2}p\nabla^2 f(x + tp)p, t \in (0, 1)$$

- We don't know t , just that it exists
- Given value and gradients at $\mathbf{x}$, can approximate function at $\mathbf{x} + \mathbf{p}$
- Higher degree gradients used, better approximation possible

Types of optimization algorithms

- Line search
- Trust region
- Gradient free

Line search

At the current solution x_k , pick a **descent** direction first p_k , then find a stepsize α :

$$\min_{\alpha > 0} f(x_k + \alpha p_k)$$

and calculate the next solution:

$$x_{k+1} = x_k + \alpha_k p_k$$

General definition of direction:

$$p_k = -B_k^{-1} \nabla f(x_k)$$

Gradient descent:

$$B_k = I$$

Newton method (assuming f twice differentiable and B_k invertible):

$$B_k = \nabla^2 f(x_k)$$

Gradient descent (for supervised MLE training)

Input: training examples $\mathcal{D} = \{(x^1, y^1), \dots (x^M, y^M)\}$,
learning_rate α

Initialize weights w

while $\nabla_w NLL(w; \mathcal{D}) \neq 0$ **do**

 Update $w = w - \alpha \nabla_w NLL(w; \mathcal{D})$

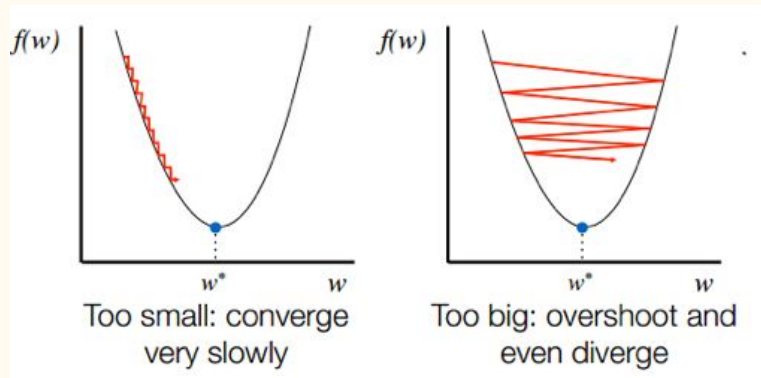
end while

To make it stochastic, just look at one training example in each iteration and go over each of them. Why is this a good idea?

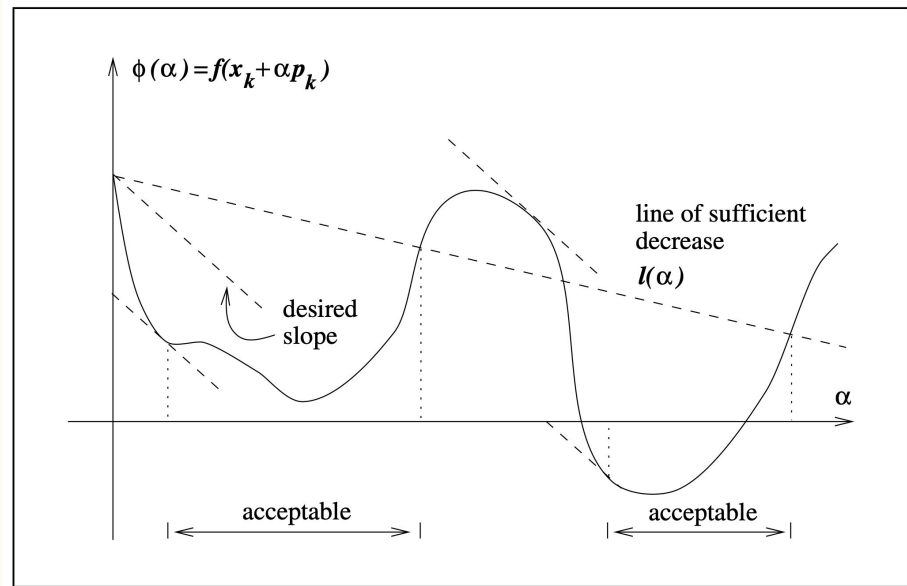
What can go wrong?

Gradient descent

Wrong step size:



<https://srdas.github.io/DLBook/GradientDescentTechniques.html>



Line search converges to the minimizer when the iterates follow the Wolfe conditions on sufficient decrease and curvature (Zoutendijk's theorem)

Back tracking: start with a large stepsize and reduce it to get sufficient decrease

Second order methods

Using the Hessian (line search Newton's method):

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla^2 f(\mathbf{x}_k)^{-1} \nabla f(\mathbf{x}_k)$$

Expensive to compute. Can we approximate?

Yes, based on the first order gradients:

$$B_{k+1} = \frac{\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)}{\mathbf{x}_{k+1} - \mathbf{x}_k}$$

BFGS calculates B_{k+1}^{-1} directly without moving too far from B_k^{-1}

What are desirable properties in line search?

Fast convergence:

- Few iterations
 - Stochastic gradient descent will have more than standard gradient descent
- Cheap iterations; what makes them expensive?
 - Function evaluations for backtracking with line search (this is the reason for researching adaptive learning rates)
 - (approximate) second order gradients (partly why they are not used in DL)

Memory requirements? Storing second order gradients requires $|w|^2$. One of the key variants of BFGS is L(imited memory)-BFGS.

Default line search nowadays

ADAM (Adaptive Moment Estimation)

- An exponentially decaying average of the gradients is kept (momentum)
 - Have we seen this before?
- Each parameter has its own learning rate using exponentially decaying squared gradients (adaptive)

One can learn the updates: [Learning to learn gradient descent by gradient descent](#)

Trust region

Taylor's theorem:

$$f(x + p) = f(x) + \nabla f(x)p + \frac{1}{2}p\nabla^2 f(x + tp)p, t \in (0, 1)$$

Assuming an approximation m to the function f we are minimizing:

$$m_k(p) = f(x_k) + \nabla f(x_k)p + \frac{1}{2}p\nabla^2 f(x_k)p$$

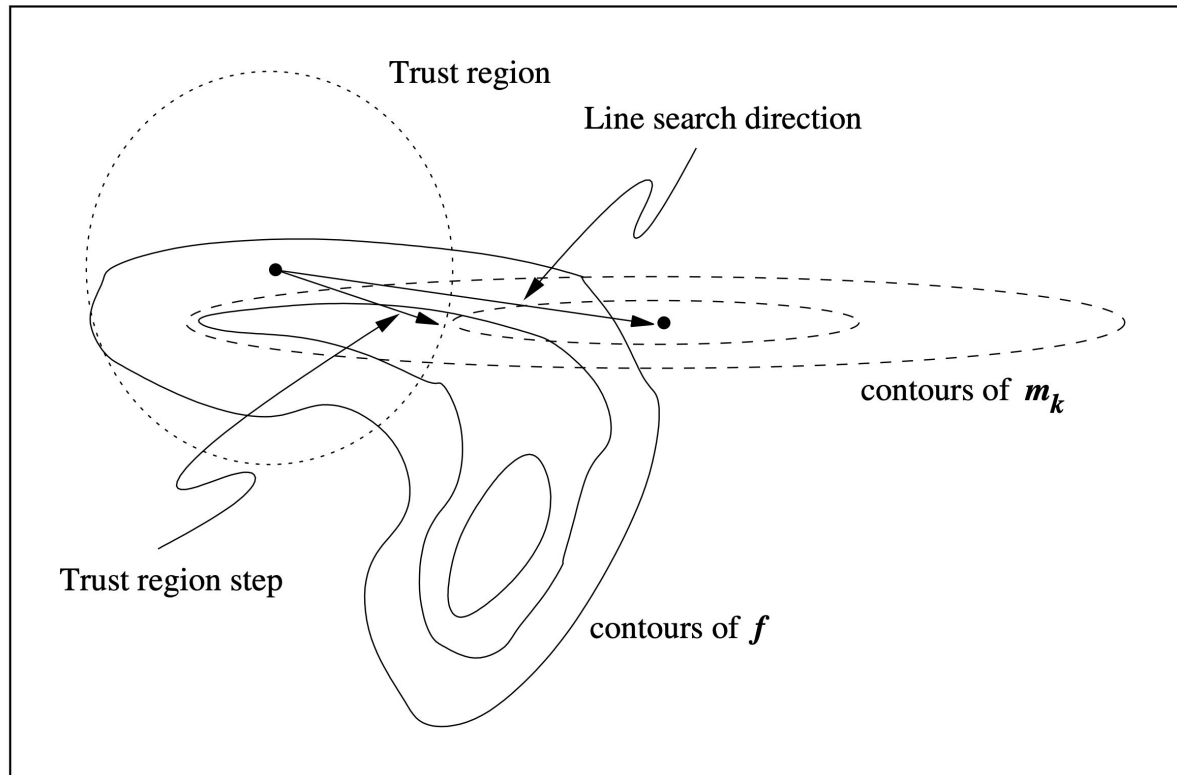
Given a radius Δ (max stepsize, **trust region**), choose a direction p such that:

$$\min_p m_k(p), p \leq \Delta_k$$

Measuring trust:

$$\frac{f(x_k) - f(x_k + p_k)}{m_k(0) - m_k(p_k)}$$

Trust region



Worth considering
with relatively few
dimensions.

Recent success in
reinforcement
learning

Gradient free

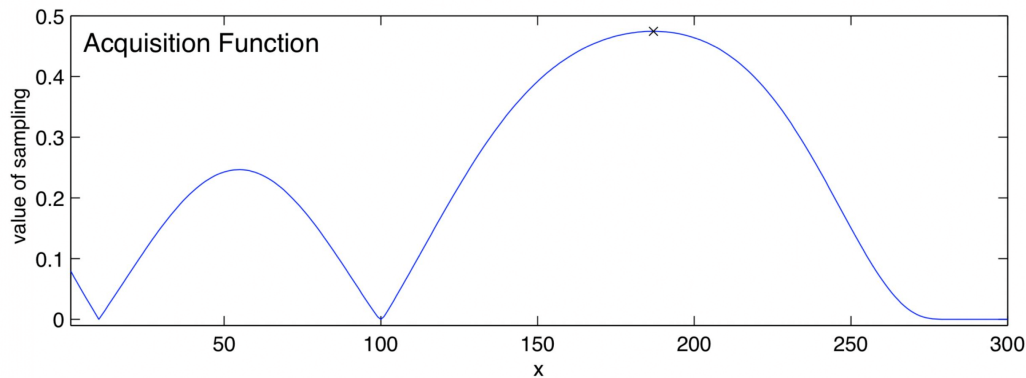
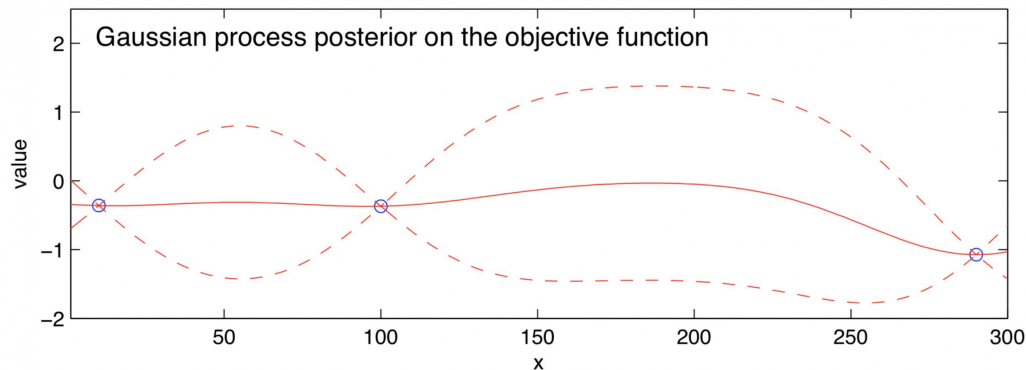
What if we don't have/want gradients?

- Function is a black box to us, can only test values
- Gradients too expensive/complicated to calculate, e.g.: hyperparameter optimization

Two large families:

- Model-based (similar to trust region but without gradients for the approximation model)
- Sampling solutions according to some heuristic
 - Evolutionary/genetic algorithms, particle swarm optimization
 - Applied to prompt optimization ([ProptBreeder, Fernando et al. 2024](#))

Bayesian Optimization

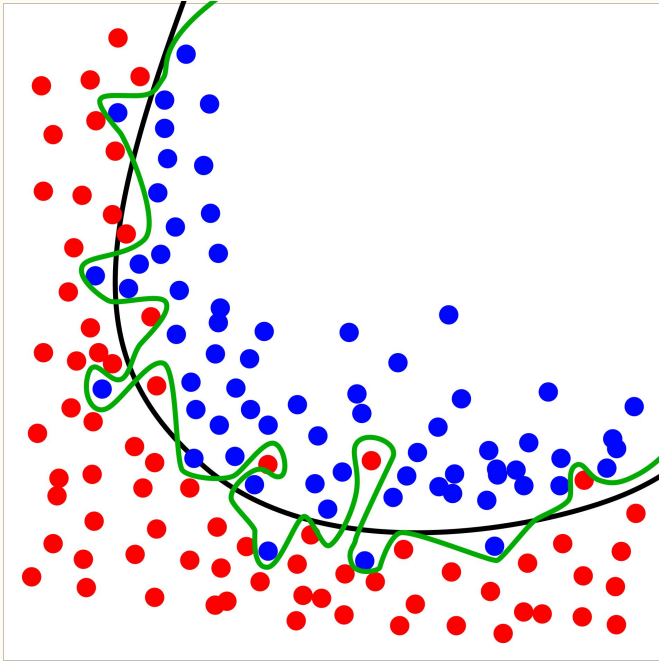


- Model approximation based on Gaussian Process regression
- Acquisition function tells us where to sample next
- See [here](#) for a nice illustration

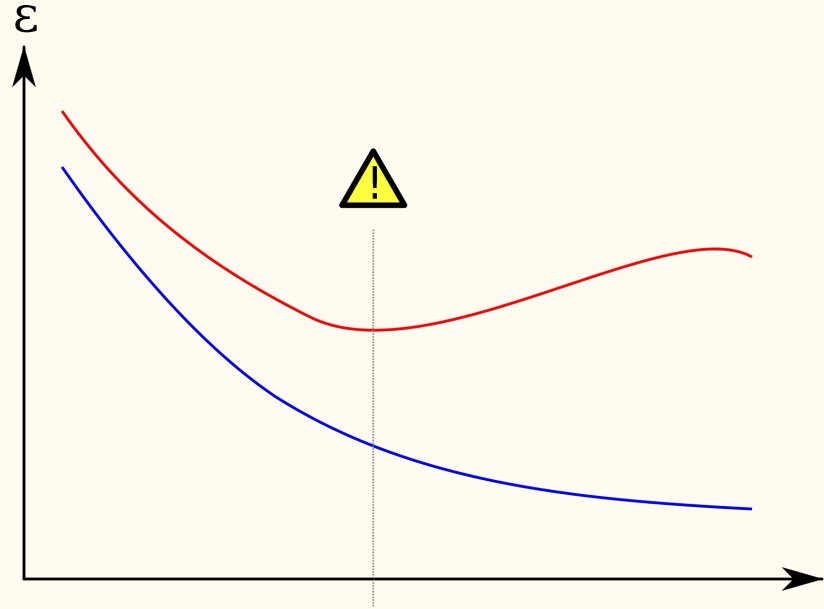
[Frazier \(2018\)](#)

Overfitting

Separating hyperplanes in training



Error during training and testing



Regularization

We want to optimize the function/fit the data but not too much:

$$w^{\star} = \arg \min_w L(w; \mathcal{D}) + \lambda \mathcal{R}(w)$$

Some options for the regularizer:

- L2 (ridge): Σw^2
- L1 (Lasso): $\Sigma |w|$
- Elastic net: L1+L2
- L-infinity: $\max(w)$

Words of caution

Sometimes we are saved from overfitting by not optimizing well enough

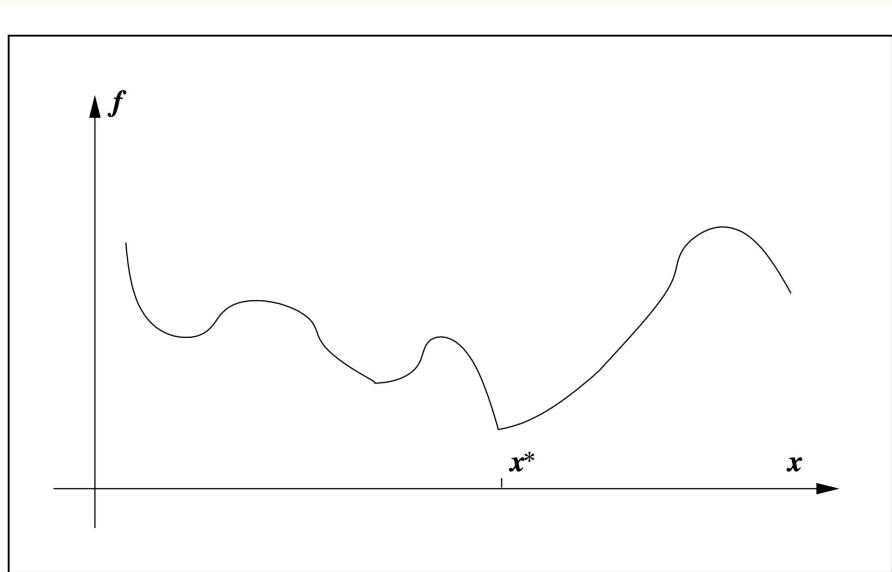
There is often a discrepancy between loss and evaluation objective; often the latter are not differentiable (e.g. training next word prediction vs factuality)

Check your objective if it tells you the right thing: optimizing less precisely and getting better generalization is OK, having to optimize badly to get results is not.

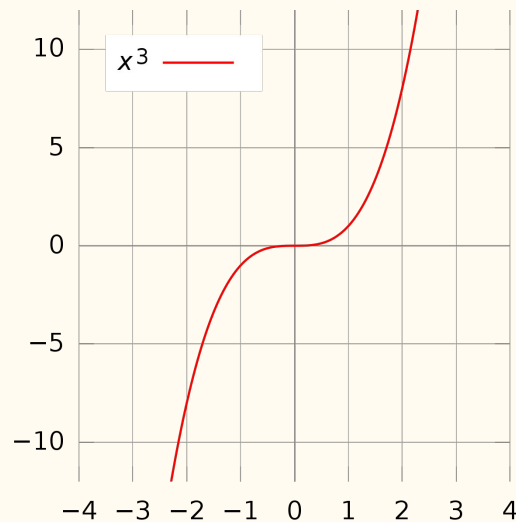
Construct toy problems: if you have a good initial set of weights, does optimizing the objective leave them unchanged?

Harder cases

- Non-convex
- Non-smooth



Saddle points: zero gradient is a first order **necessary condition, not sufficient**



https://en.wikipedia.org/wiki/Saddle_point

Bibliography

- Numerical Optimization, Nocedal and Wright, 2002. (uncited images from there) <https://www.springer.com/gb/book/9780387303031>
- Francisco Orabona's blog: <https://parameterfree.com>
- A course on optimization in ML by Roger Grosse:
https://www.cs.toronto.edu/~rgrosse/courses/csc2541_2022/
- Hardt and Recht, 2022 (especially Chapter 5):
<https://press.princeton.edu/books/hardcover/9780691233734/patterns-predictions-and-actions>