Linear regression and Optimisation

Introduction to Machine Learning - Tutorial 2

Piersilvio De Bartolomeis

Simplified diagram of supervised learning

Training Data

(x_1, y_1)
$\vdots$
(x_n, y_n)

attributes of patient

supervision: observed labels

ML method: efficiently find $\hat{f} \in \mathscr{F}$ that fits training data

ML scientists design

Some test input x

Model $\hat{f}$

Prediction $\hat{f}(x)$

Recap from last week: Linear regression

Training Data

ML method:
efficiently find $\hat{f} \in \mathcal{F}$

Model $\hat{f}$

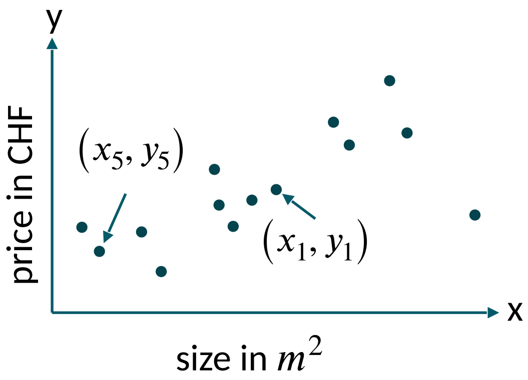
ML Scientists
design

Function class	Training loss	Optimisation
• linear functions	• squared loss	• closed-form

Recap from last week: Linear regression

Training Data

$$\begin{matrix}(x_1, y_1)\\ \vdots \\ (x_n, y_n)\end{matrix}$$



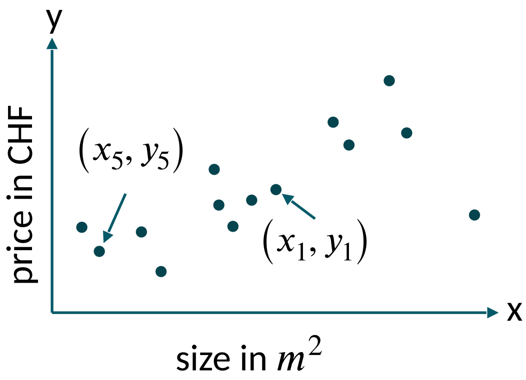
Recap from last week: Linear regression

Training Data

$$(x_1, y_1)$$
$$\vdots$$
$$(x_n, y_n)$$

Optimisation problem

$$w^* = \operatorname{argmin}_{w \in \mathbb{R}^2} \frac{1}{2n} \|y - \Phi w\|^2$$



Recap from last week: Linear regression

Training Data

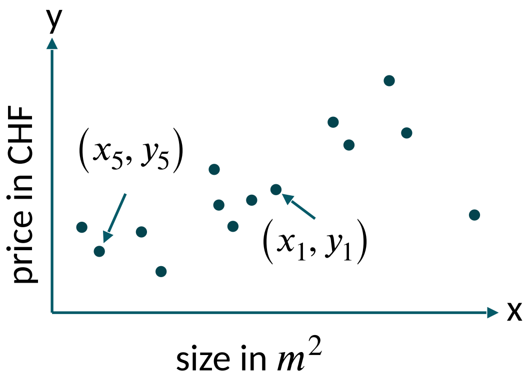
$$(x_1, y_1)$$
$$\vdots$$
$$(x_n, y_n)$$

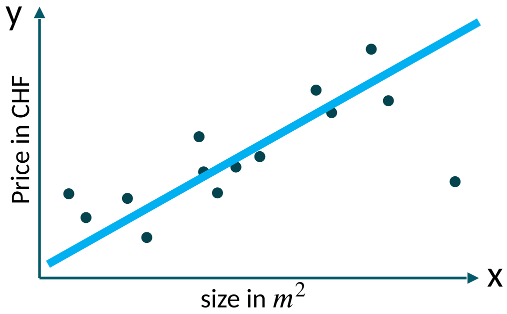
Optimisation problem

$$w^* = \operatorname{argmin}_{w \in \mathbb{R}^2} \frac{1}{2n} \|y - \Phi w\|^2$$

Model $\hat{f}$

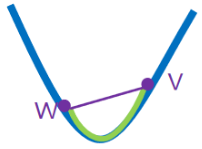
$$\hat{f}(x) = x^\top w^*$$





Intermezzo: Convex functions

Intermezzo: Convex functions



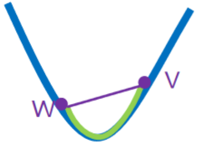
Mathematically, convexity is the function property that guarantees local = global minimum

- 0-th order condition (if and only if): $L(\lambda w + (1-\lambda)v) \leq \lambda\, L(w) + (1-\lambda)L(v)$

  true function connecting any w, v $\leq$ linear function connecting two points x, y

Intermezzo: Convex functions

Mathematically, convexity is the function property that guarantees local = global minimum



- 0-th order condition (if and only if): $L(\lambda w + (1-\lambda)v) \leq \lambda\, L(w) + (1-\lambda)L(v)$

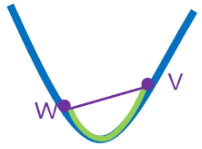
  true function connecting any $\mathbf{w}, v$ $\leq$ linear function connecting two points x, y

- 1st order condition (if and only if): $L(v) \geq L(w) + \nabla L(w)^{\top}(v-w)$

  value at any v of tangent line at any w $\leq$ true value at v

Intermezzo: Convex functions

Mathematically, convexity is the function property that guarantees local = global minimum

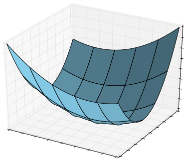


- 0-th order condition (if and only if): $L(\lambda w + (1-\lambda)v) \leq \lambda\, L(w) + (1-\lambda)L(v)$

  true function connecting any $\mathbf{w}, v$ $\leq$ linear function connecting two points x, y

- 1st order condition (if and only if): $L(v) \geq L(w) + \nabla L(w)^{\top}(v - w)$

  value at any v of tangent line at any w $\leq$ true value at v



- 2nd order condition (if and only if): Hessian $\nabla^2 L(w) \succcurlyeq 0$

  non-negative curvature at every $\mathbf{w}$

Recap from last week: Closed-form solution

Recap from last week: Closed-form solution

(i) Define opt. problem: $w^* = \operatorname*{argmin}_{w \in \mathbb{R}^d} \overbrace{\frac{1}{2n} \|y - \Phi w\|^2}^{=L(w)}$

Recap from last week: Closed-form solution

(i) Define opt. problem: $w^* = \underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \overbrace{\frac{1}{2n}\|y - \Phi w\|^2}^{=L(w)}$

(ii) Compute gradient: $\nabla_w L(w) = \frac{1}{n}(\Phi^\top \Phi w - \Phi^\top y)$

Recap from last week: Closed-form solution

(i) Define opt. problem: $w^* = \underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \overbrace{\frac{1}{2n}\|y - \Phi w\|^2}^{=L(w)}$

(ii) Compute gradient: $\nabla_w L(w) = \frac{1}{n}(\Phi^\top \Phi w - \Phi^\top y)$

(iii) Show Hessian is positive definite: $\nabla_w^2 L(w) = \frac{1}{n}\Phi^\top \Phi \succ 0$

Recap from last week: Closed-form solution

(i) Define opt. problem: $w^* = \underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \overbrace{\frac{1}{2n}\|y - \Phi w\|^2}^{=L(w)}$

(ii) Compute gradient: $\nabla_w L(w) = \frac{1}{n}(\Phi^\top \Phi w - \Phi^\top y)$

(iii) Show Hessian is positive definite: $\nabla_w^2 L(w) = \frac{1}{n}\Phi^\top \Phi \succ 0$

(iv) Set gradient to zero: $\nabla_w L(w^*) = 0 \implies w^* = (\Phi^\top \Phi)^{-1}\Phi^\top y$

This week so far

Training Data

ML method:
efficiently find $f \in \mathscr{F}$

Model $\hat{f}$

ML Scientists
design

Function class

Training loss

Optimisation

This week so far

Training Data → ML method: efficiently find $f \in \mathcal{F}$ → Model $\hat{f}$

ML Scientists design

Function class	Training loss	Optimisation
• linear • polynomial	• squared loss	• gradient descent

This week so far

Training Data → ML method: efficiently find $f \in \mathcal{F}$ → Model $\hat{f}$

ML Scientists design

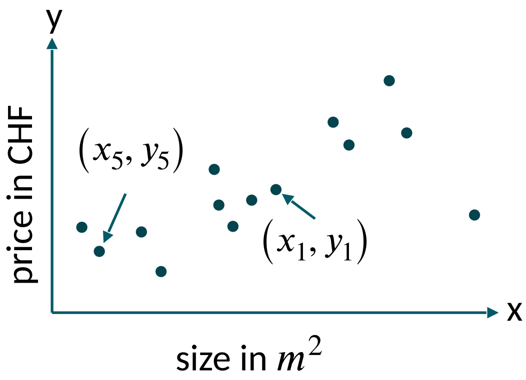
Function class	Training loss	Optimisation
• linear	• squared loss	• gradient descent
• polynomial	• mae loss	• momentum
• trigonometric	• huber loss	• adaptive methods

Today!

Recap from this week: Polynomial regression

Training Data

$$(x_1, y_1)$$
$$\vdots$$
$$(x_n, y_n)$$



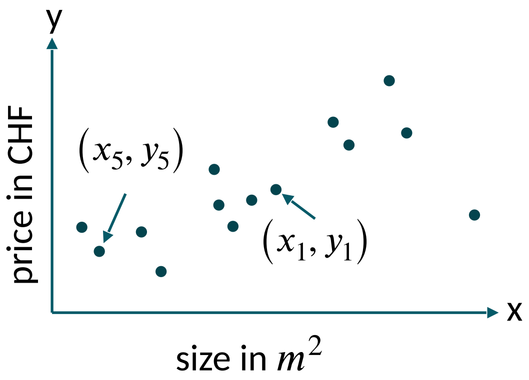
Recap from this week: Polynomial regression

Training Data

$$(x_1, y_1)$$
$$\vdots$$
$$(x_n, y_n)$$

Optimisation problem

$$w^* = \text{argmin}_{w \in \mathbb{R}^2} \frac{1}{2n} \|y - \Phi w\|^2$$



Recap from this week: Polynomial regression

Training Data

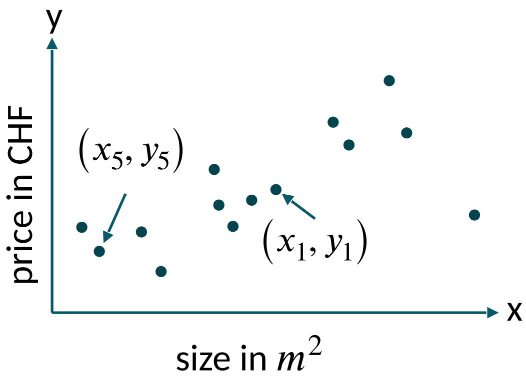
Optimisation problem

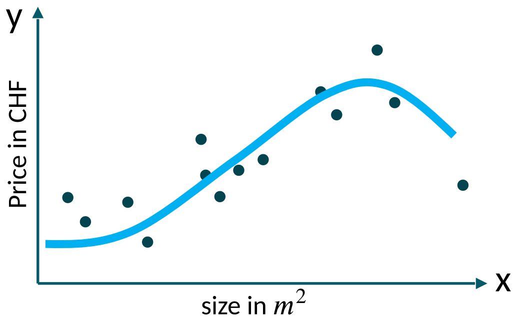
Model

$$\begin{matrix}(x_1, y_1)\\ \vdots \\ (x_n, y_n)\end{matrix}$$

$$w^* = \mathrm{argmin}_{w \in \mathbb{R}^2} \frac{1}{2n} \|y - \Phi w\|^2$$

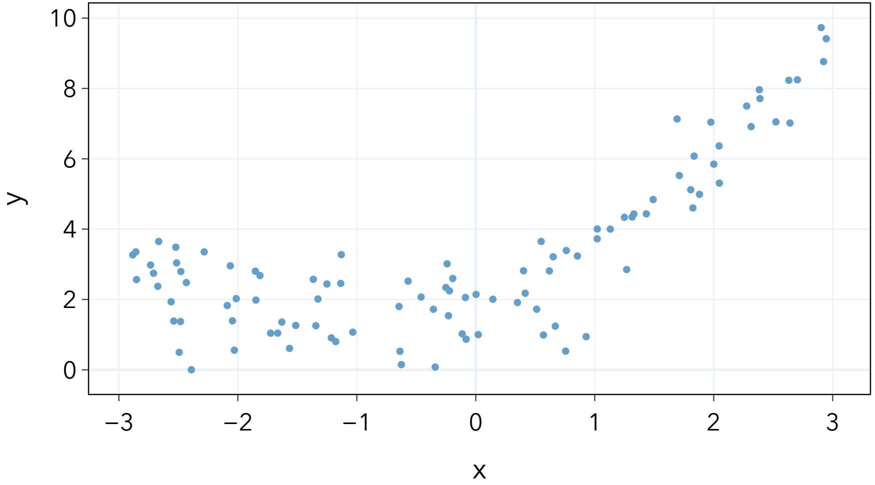
$$\hat{f}(x) = \phi(x)^\top w^*$$





Recap from this week: Polynomial regression

What to do if we want to fit $\hat{f}(x) = w_0 + w_1 x + w_2 x^2$?



Recap from this week: Polynomial regression

Recap from this week: Polynomial regression

(i) Define $\phi(x) = [1 \; x \; x^2]^\top$

Recap from this week: Polynomial regression

(i) Define $\phi(x) = [1 \; x \; x^2]^\top$

(ii) We can then write: $\hat{f}(x) = w^\top \phi(x)$

Recap from this week: Polynomial regression

(i) Define $\phi(x) = [1 \ x \ x^2]^\top$

(ii) We can then write: $\hat{f}(x) = w^\top \phi(x)$

(iii) The design matrix becomes $\Phi = \begin{pmatrix} 1 & x_1 & x_1^2 \\ \vdots & & \vdots \\ 1 & x_n & x_n^2 \end{pmatrix}$

Recap from this week: Polynomial regression

(i) Define $\phi(x) = [1 \ x \ x^2]^\top$

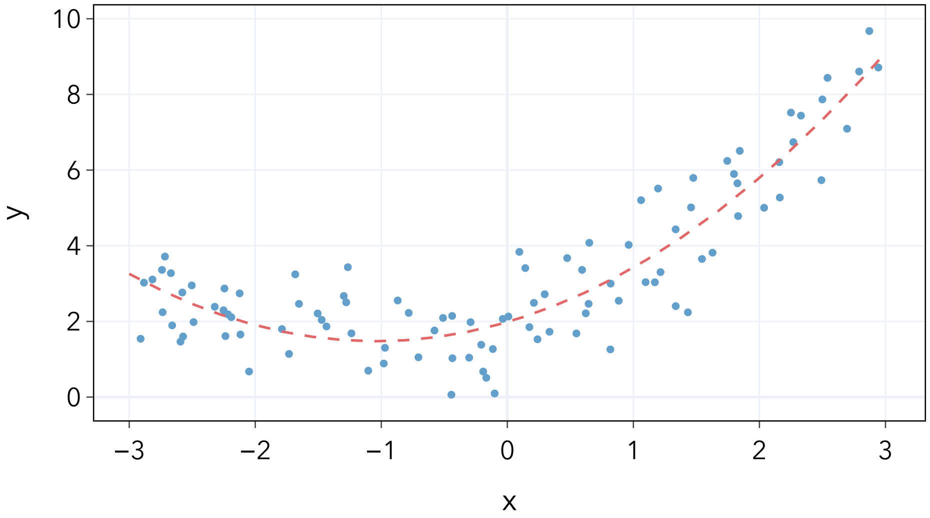
(ii) We can then write: $\hat{f}(x) = w^\top \phi(x)$

(iii) The design matrix becomes $\Phi = \begin{pmatrix} 1 & x_1 & x_1^2 \\ \vdots & & \vdots \\ 1 & x_n & x_n^2 \end{pmatrix}$

(4) Set gradient to zero (as in the linear case): $\nabla_w L(w^*) = 0 \implies w^* = (\Phi^\top \Phi)^{-1} \Phi^\top y$

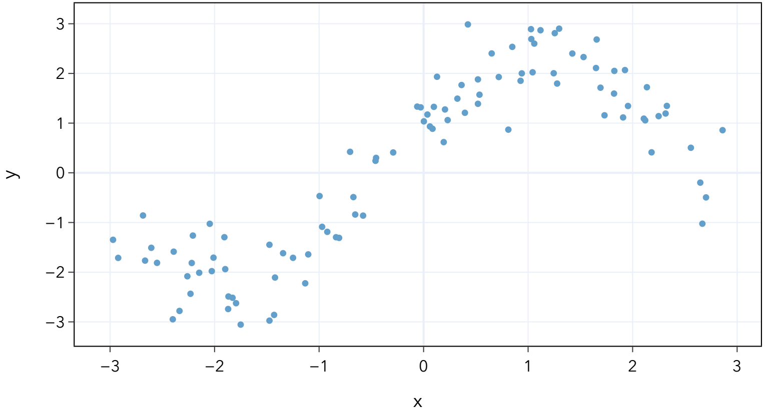
Recap from this week: Polynomial regression

How does our fit look like? $\hat{f}(x) = \phi(x)^\top w*$



Beyond polynomials: Trigonometric functions

What to do if we want to fit $\hat{f}(x) = w_1 \sin(x) + w_2 \cos(x)$?



Beyond polynomials: Trigonometric functions

Beyond polynomials: Trigonometric functions

(i) Define $\phi(x) = [\sin(x) \ \ \cos(x)]^\top$

Beyond polynomials: Trigonometric functions

(i) Define $\phi(x) = [\sin(x) \ \ \cos(x)]^\top$

(ii) We can then write: $\hat{f}(x) = w^\top \phi(x)$

Beyond polynomials: Trigonometric functions

(i) Define $\phi(x) = [\sin(x) \;\; \cos(x)]^\top$

(ii) We can then write: $\hat{f}(x) = w^\top \phi(x)$

(iii) The design matrix becomes $\Phi = \begin{pmatrix} \sin(x_1) & \cos(x_1) \\ \vdots & \vdots \\ \sin(x_n) & \cos(x_n) \end{pmatrix}$

Beyond polynomials: Trigonometric functions

(i) Define $\phi(x) = [\sin(x) \;\; \cos(x)]^\top$

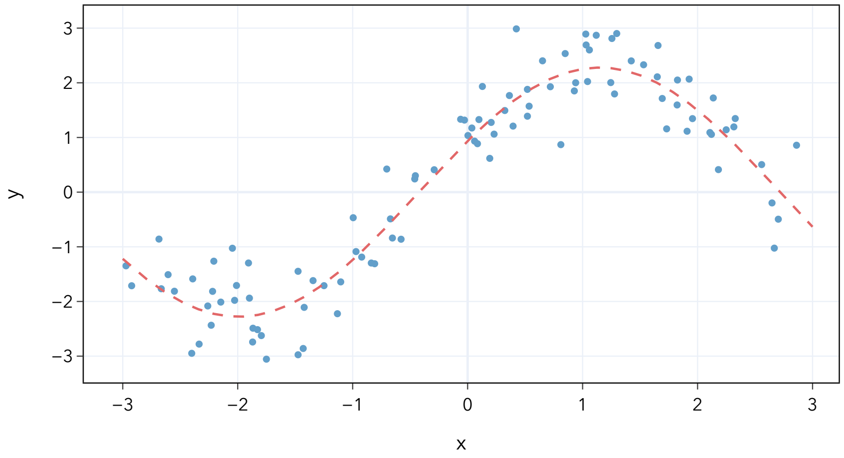
(ii) We can then write: $\hat{f}(x) = w^\top \phi(x)$

(iii) The design matrix becomes $\Phi = \begin{pmatrix} \sin(x_1) & \cos(x_1) \\ \vdots & \vdots \\ \sin(x_n) & \cos(x_n) \end{pmatrix}$

(4) Set gradient to zero (as in the linear case): $\nabla_w L(w^*) = 0 \implies w^* = (\Phi^\top \Phi)^{-1} \Phi^\top y$

Beyond polynomials: Trigonometric functions

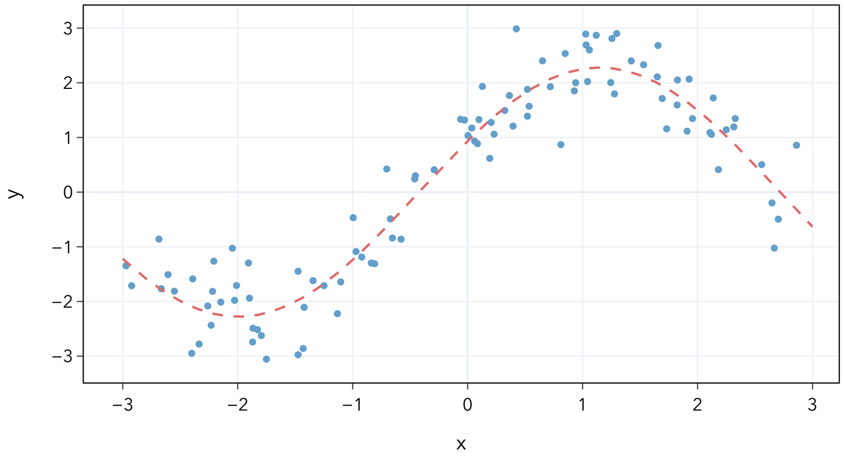
How does our fit look like? $\hat{f}(x) = \phi(x)^\top w^*$



Beyond polynomials: Trigonometric functions

More on Jupyter!

How does our fit look like? $\hat{f}(x) = \phi(x)^\top w^*$



Recap from this week: Gradient descent algorithm

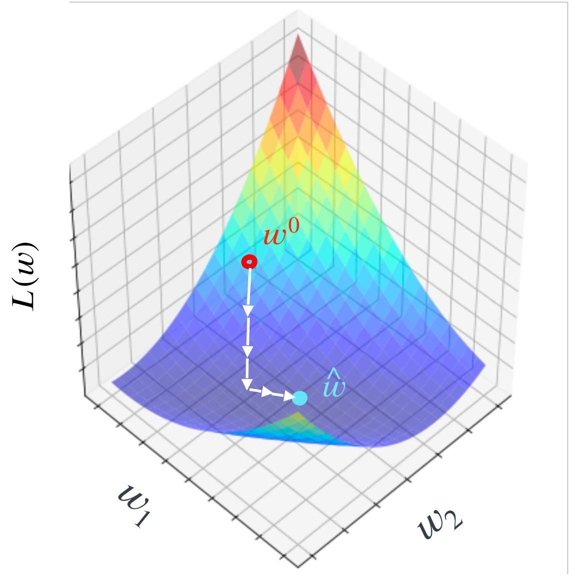
Gradient descent algorithm to minimize $L(w)$

Start at initial w^0

At each step t, $w^{t+1} = w^t + \eta \nabla_w L\left(w^t\right)$

Stop e.g. when $|L(w^{t+1}) - L(w^t)| \leq \epsilon$

Output $\hat{w} = w^{final}$



Recap from this week: Gradient descent algorithm

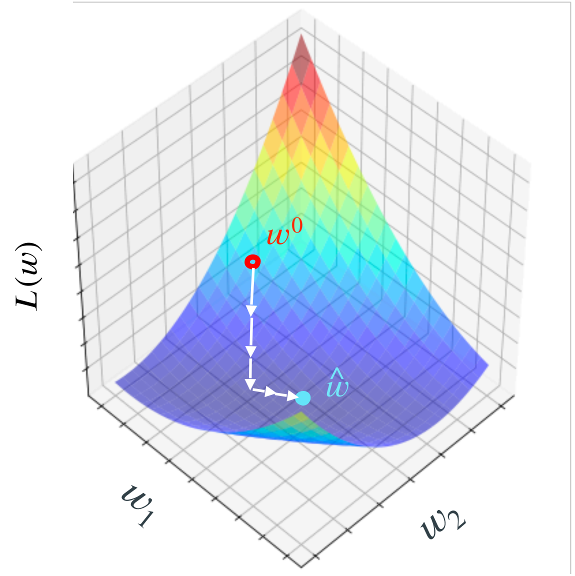
Gradient descent algorithm to minimize $L(w)$

Start at initial w^0

At each step t, $w^{t+1} = w^t + \eta \nabla_w L\left(w^t\right)$

Stop e.g. when $|L\left(w^{t+1}\right) - L\left(w^t\right)| \leq \epsilon$

Output $\hat{w} = w^{final}$



Linear approximation for the loss value at step $t+1$

$$L\left(w^{t+1}\right) = L\left(w^t - \eta \nabla L(w^t)\right) \approx L\left(w^t\right) - \eta \left\langle \nabla L\left(w^t\right), \nabla L\left(w^t\right) \right\rangle < L(w^t)$$

Recap from this week: Gradient descent algorithm

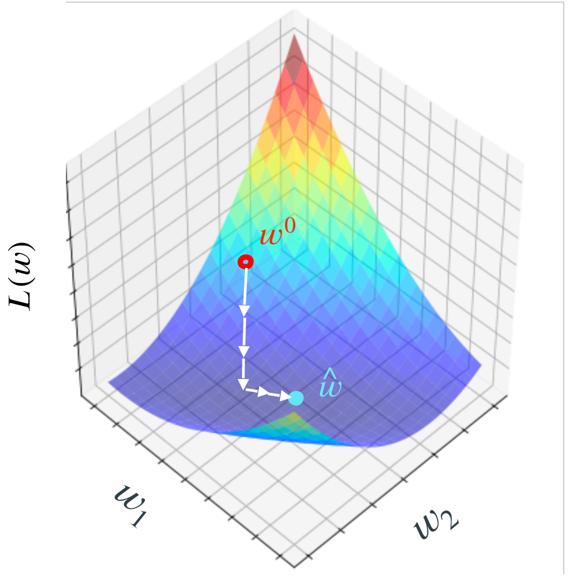
Gradient descent algorithm to minimize $L(w)$

Start at initial w^0

At each step t, $\quad w^{t+1} = w^t + \eta \nabla_w L\left(w^t\right)$

Stop e.g. when $|L(w^{t+1}) - L(w^t)| \leq \epsilon$

Output $\hat{w} = w^{final}$



Linear approximation for the loss value at step $t+1$

$$L(w^{t+1}) = L(w^t - \eta \nabla L(w^t)) \approx L(w^t) - \eta \left\langle \nabla L(w^t), \nabla L(w^t) \right\rangle < L(w^t)$$

the negative gradient direction is a descent direction

for small enough stepsize η

Gradient descent: how fast does it converge?

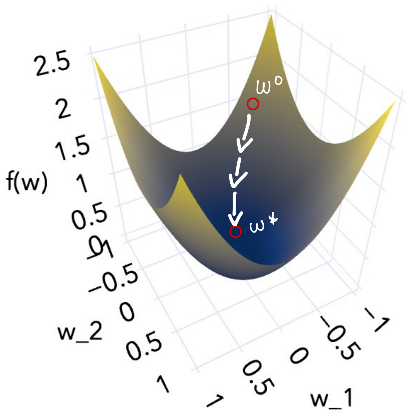
Gradient descent: how fast does it converge?

- Let's consider a very simple model $f(w) = \frac{1}{2} w^\top A w$, $w \in \mathbb{R}^d$, $A = \mathrm{diag}(\lambda_1, \ldots, \lambda_d) \succ 0$
- Question: when does gradient descent converge and how fast?

Gradient descent: how fast does it converge?

- Let's consider a very simple model $f(w) = \frac{1}{2} w^\top A w \, , \, w \in \mathbb{R}^d \, , A = \mathrm{diag}(\lambda_1, \ldots, \lambda_d) \succ 0$
- Question: when does gradient descent converge and how fast?

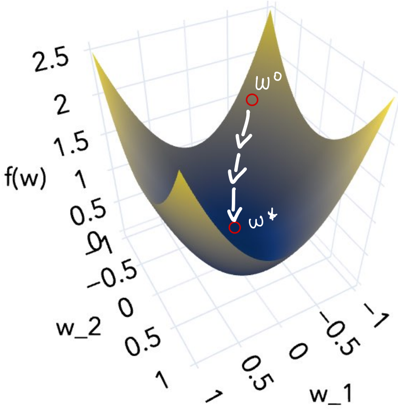
$$A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$



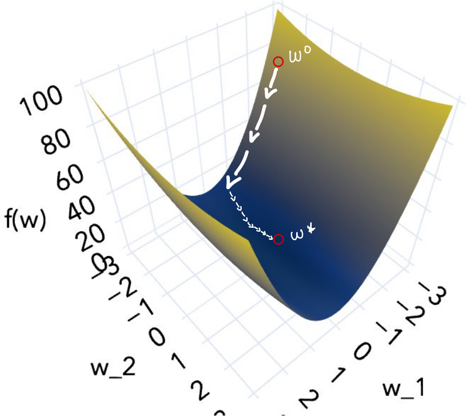
Gradient descent: how fast does it converge?

- Let's consider a very simple model $f(w) = \frac{1}{2} w^\top A w$, $w \in \mathbb{R}^d$, $A = \mathrm{diag}(\lambda_1, \ldots, \lambda_d) \succ 0$
- Question: when does gradient descent converge and how fast?

$A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$



$A = \begin{pmatrix} 1 & 0 \\ 0 & 5 \end{pmatrix}$

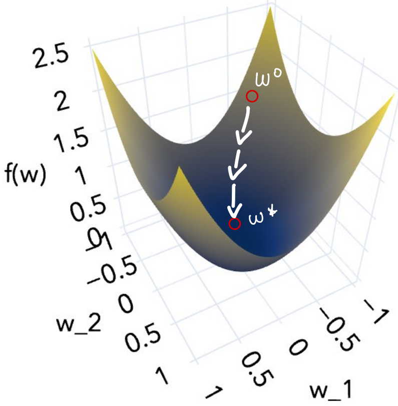


Gradient descent: how fast does it converge?

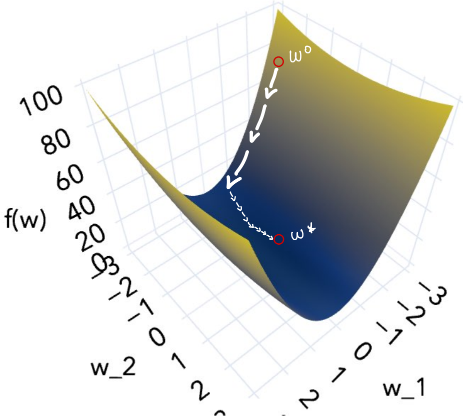
More on iPad!

- Let's consider a very simple model $f(w) = \frac{1}{2} w^\top A w$, $w \in \mathbb{R}^d$, $A = \mathrm{diag}(\lambda_1, \ldots, \lambda_d) \succ 0$
- Question: when does gradient descent converge and how fast?

$A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$



$A = \begin{pmatrix} 1 & 0 \\ 0 & 5 \end{pmatrix}$



Gradient descent: how fast does it converge?

More on iPad!

$$f(w) = \frac{1}{2} w^\top A w \ , \ w \in \mathbb{R}^d \ , A = \mathrm{diag}(\lambda_1, \ldots, \lambda_d) \succ 0$$

Gradient descent: how fast does it converge?

More on iPad!

- Let's consider a very simple model $f(w) = \frac{1}{2} w^\top A w \ , \ w \in \mathbb{R}^d \ , A = \mathrm{diag}(\lambda_1, \ldots, \lambda_d) \succ 0$

Gradient descent: how fast does it converge?

More on iPad!

- Let's consider a very simple model $f(w) = \frac{1}{2} w^\top A w$, $w \in \mathbb{R}^d$, $A = \mathrm{diag}(\lambda_1, \ldots, \lambda_d) \succ 0$
- Gradient descent converges for learning rate small enough, i.e. $\eta < \frac{2}{\lambda_d}$

Gradient descent: how fast does it converge?

More on iPad!

- Let's consider a very simple model $f(w) = \frac{1}{2} w^\top A w$, $w \in \mathbb{R}^d$, $A = \mathrm{diag}(\lambda_1, \ldots, \lambda_d) \succ 0$
- Gradient descent converges for learning rate small enough, i.e. $\eta < \frac{2}{\lambda_d}$
- How should we choose the learning rate? $\eta^\star = \frac{2}{\lambda_1 + \lambda_d}$

Gradient descent: how fast does it converge?

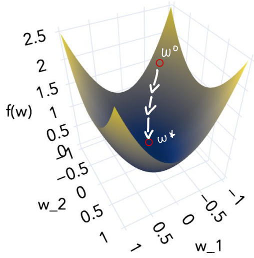
More on iPad!

- Let's consider a very simple model $f(w) = \frac{1}{2} w^\top A w \ , \ w \in \mathbb{R}^d \ , A = \mathrm{diag}(\lambda_1, \ldots, \lambda_d) \succ 0$
- Gradient descent converges for learning rate small enough, i.e. $\eta < \frac{2}{\lambda_d}$
- How should we choose the learning rate? $\eta^\star = \frac{2}{\lambda_1 + \lambda_d}$
- How fast we converge with the optimal learning rate? $\mathrm{rate}(\eta^\star) = \frac{\kappa - 1}{\kappa + 1}$

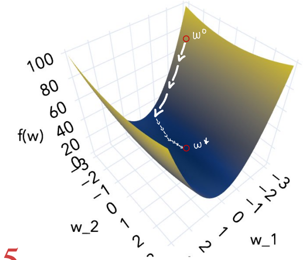
Gradient descent: how fast does it converge?

More on iPad!

- Let's consider a very simple model $f(w) = \frac{1}{2} w^\top A w \, , \, w \in \mathbb{R}^d \, , A = \mathrm{diag}(\lambda_1, \ldots, \lambda_d) \succ 0$
- Gradient descent converges for learning rate small enough, i.e. $\eta < \frac{2}{\lambda_d}$
- How should we choose the learning rate? $\eta^\star = \frac{2}{\lambda_1 + \lambda_d}$
- How fast we converge with the optimal learning rate? $\mathrm{rate}(\eta^\star) = \frac{\kappa - 1}{\kappa + 1}$



$\kappa = 1$



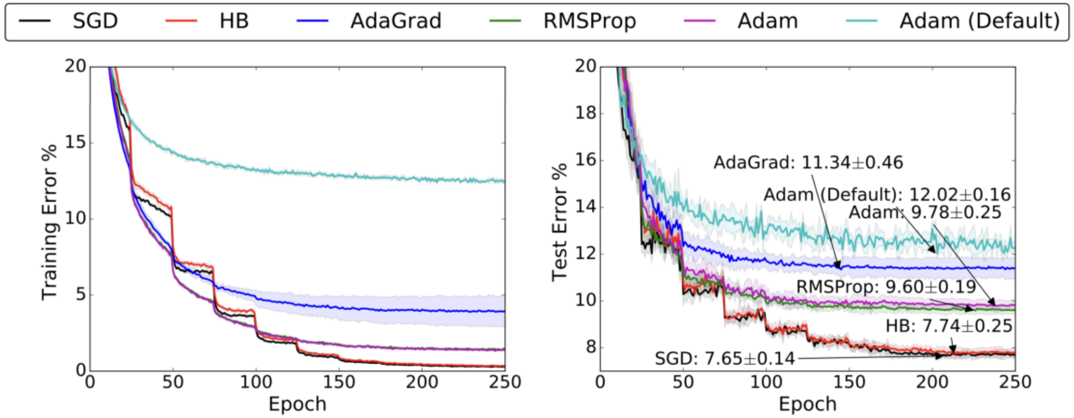
$\kappa = 5$

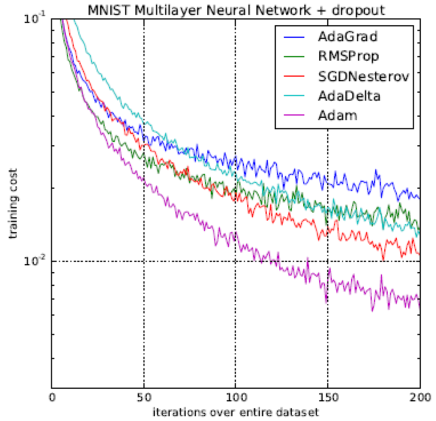
There’s a whole zoo of optimisation methods…

For deep learning, there are many optimisation methods (number of lines)

people usually try out (don’t have to understand plots in detail)

→ now want to have a brief overview how they differ from gradient descent





Speeding up gradient descent

Goal: Large steps in flat areas, small steps in high curvature ones, dampened oscillations

- momentum/accelerated methods: combine previous direction with neg. gradient direction for some $\alpha \in \mathbb{R}$

$$w^{t+1} - w^t = \alpha \left(w^t - w^{t-1}\right) - \eta \nabla_w L\left(w^t\right)$$

(bonus: discrete approximation of dampened harmonic oscillator)

Speeding up gradient descent

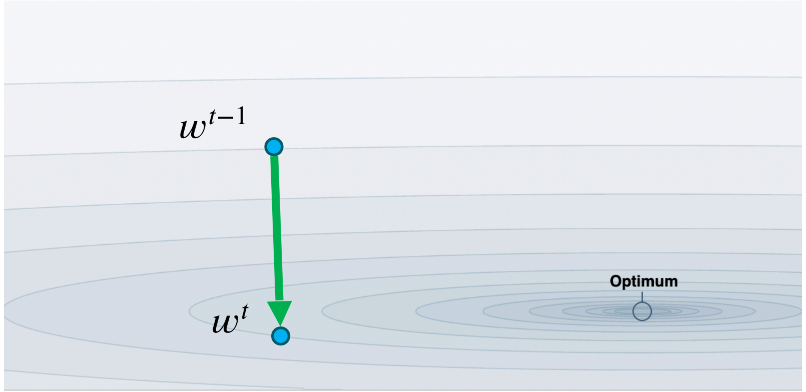
Goal: Large steps in flat areas, small steps in high curvature ones, dampened oscillations

- momentum/accelerated methods: combine previous direction with neg. gradient direction for some $\alpha \in \mathbb{R}$

$$w^{t+1} - w^t = \alpha \left(w^t - w^{t-1}\right) - \eta \nabla_w L\left(w^t\right)$$

(bonus: discrete approximation of dampened harmonic oscillator)

how it dampens oscillations



Speeding up gradient descent

Goal: Large steps in flat areas, small steps in high curvature ones, dampened oscillations

- momentum/accelerated methods: combine previous direction with neg. gradient direction for some $\alpha \in \mathbb{R}$

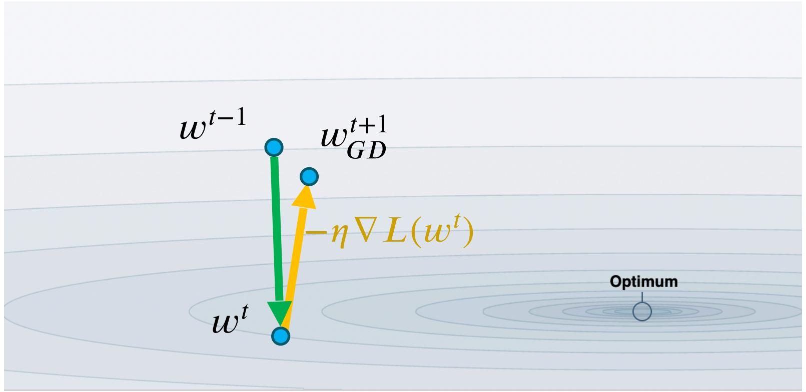
$$w^{t+1} - w^t = \alpha\left(w^t - w^{t-1}\right) - \eta \nabla_w L\left(w^t\right)$$

(bonus: discrete approximation of dampened harmonic oscillator)

At time step t:

neg. gradient direction at step t

how it dampens oscillations



Speeding up gradient descent

Goal: Large steps in flat areas, small steps in high curvature ones, dampened oscillations

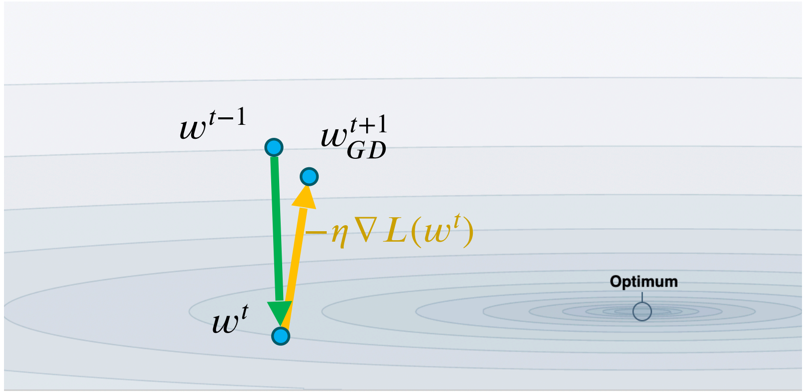
- momentum/accelerated methods: combine previous direction with neg. gradient direction for some $\alpha \in \mathbb{R}$

$$w^{t+1} - w^{t} = \alpha \left(w^{t} - w^{t-1} \right) - \eta \nabla_{w} L\left(w^{t} \right)$$

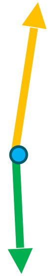
(bonus: discrete approximation of dampened harmonic oscillator)

At time step t:

how it dampens oscillations



neg. gradient direction at step t



(scaled) direction in step $t-1$: $w^t - w^{t-1}$

Speeding up gradient descent

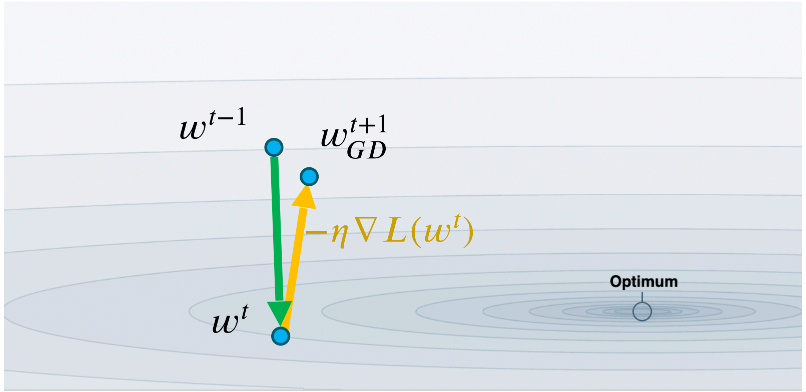
Goal: Large steps in flat areas, small steps in high curvature ones, dampened oscillations

- momentum/accelerated methods: combine previous direction with neg. gradient direction for some $\alpha \in \mathbb{R}$

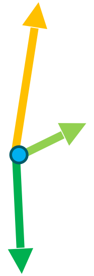
$$w^{t+1} - w^{t} = \alpha\left(w^{t} - w^{t-1}\right) - \eta \nabla_{w} L\left(w^{t}\right)$$

(bonus: discrete approximation of dampened harmonic oscillator)

how it dampens oscillations



At time step t:



Speeding up gradient descent

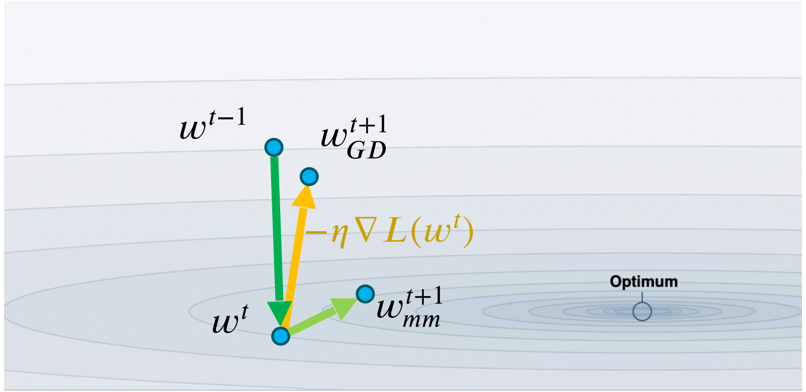
Goal: Large steps in flat areas, small steps in high curvature ones, dampened oscillations

- momentum/accelerated methods: combine previous direction with neg. gradient direction for some $\alpha \in \mathbb{R}$

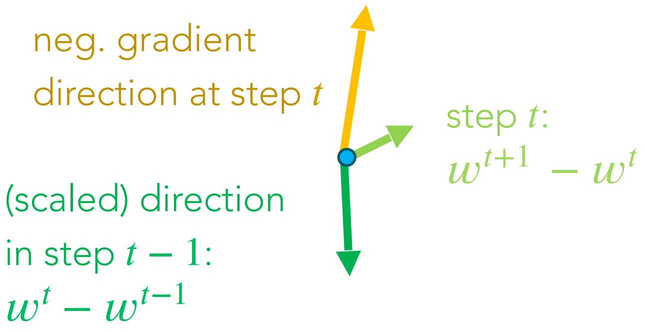
$$w^{t+1} - w^t = \alpha \left(w^t - w^{t-1}\right) - \eta \nabla_w L\left(w^t\right)$$

(bonus: discrete approximation of dampened harmonic oscillator)

how it dampens oscillations



At time step t:



Speeding up gradient descent

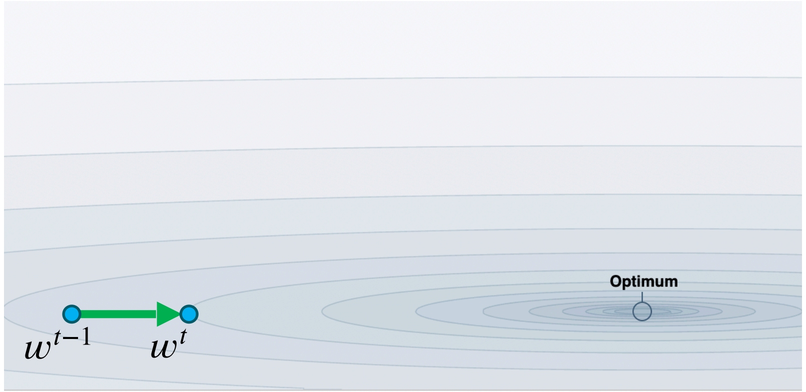
Goal: Large steps in flat areas, small steps in high curvature ones, dampened oscillations

- momentum/accelerated methods: combine previous direction with neg. gradient direction for some $\alpha \in \mathbb{R}$

$$w^{t+1} - w^t = \alpha \left(w^t - w^{t-1}\right) - \eta \nabla_w L\left(w^t\right)$$

(bonus: discrete approximation of dampened harmonic oscillator)

speeds up in flat areas



Speeding up gradient descent

Goal: Large steps in flat areas, small steps in high curvature ones, dampened oscillations

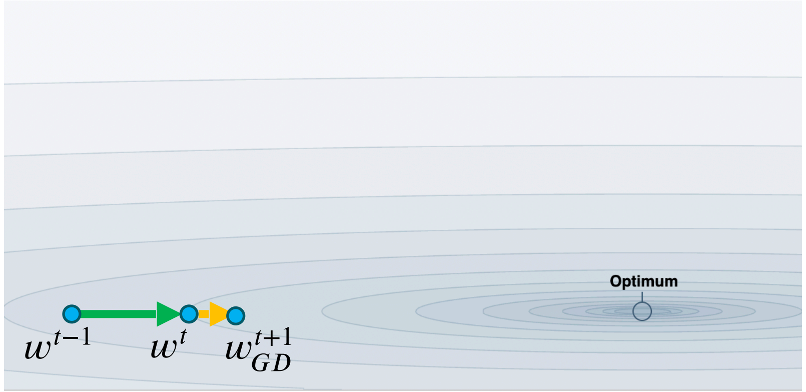
- momentum/accelerated methods: combine previous direction with neg. gradient direction for some $\alpha \in \mathbb{R}$

$$w^{t+1} - w^t = \alpha \left(w^t - w^{t-1}\right) - \eta \nabla_w L\left(w^t\right)$$

(bonus: discrete approximation of dampened harmonic oscillator)

At time step t:

speeds up in flat areas



neg. gradient direction at step t

Speeding up gradient descent

Goal: Large steps in flat areas, small steps in high curvature ones, dampened oscillations

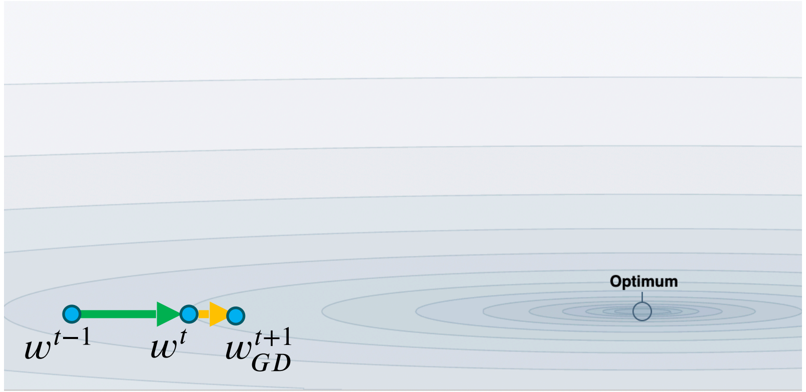
- momentum/accelerated methods: combine previous direction with neg. gradient direction for some $\alpha \in \mathbb{R}$

$$w^{t+1} - w^t = \alpha \left(w^t - w^{t-1}\right) - \eta \nabla_w L\left(w^t\right)$$

(bonus: discrete approximation of dampened harmonic oscillator)

At time step t:

speeds up in flat areas



neg. gradient direction at step t

(scaled) direction in step $t-1$: $(w^t - w^{t-1})$

Speeding up gradient descent

Goal: Large steps in flat areas, small steps in high curvature ones, dampened oscillations

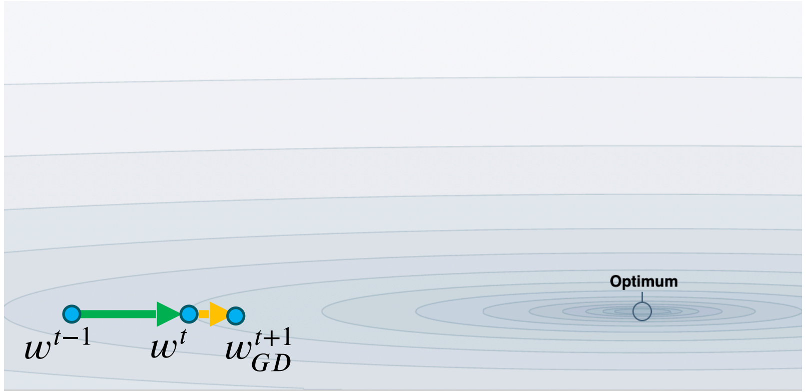
- momentum/accelerated methods: combine previous direction with neg. gradient direction for some $\alpha \in \mathbb{R}$

$$w^{t+1} - w^{t} = \alpha\left(w^{t} - w^{t-1}\right) - \eta \nabla_{w} L\left(w^{t}\right)$$

(bonus: discrete approximation of dampened harmonic oscillator)

At time step t:

speeds up in flat areas



neg. gradient direction at step t

(scaled) direction in step $t-1$: $(w^{t} - w^{t-1})$

step t: $w^{t+1} - w^{t}$

Speeding up gradient descent

Goal: Large steps in flat areas, small steps in high curvature ones, dampened oscillations

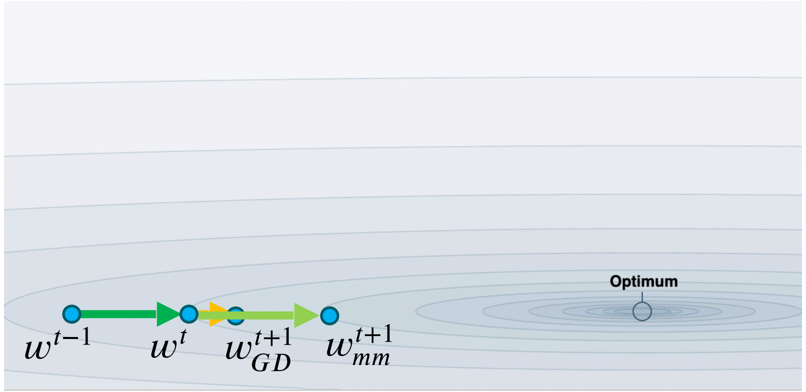
- momentum/accelerated methods: combine previous direction with neg. gradient direction for some $\alpha \in \mathbb{R}$

$$w^{t+1} - w^{t} = \alpha \left(w^{t} - w^{t-1}\right) - \eta \nabla_{w} L\left(w^{t}\right)$$

(bonus: discrete approximation of dampened harmonic oscillator)

At time step t:

speeds up
in flat areas



neg. gradient
direction at step t

(scaled) direction
in step $t-1$:
$(w^{t} - w^{t-1})$

step t:
$w^{t+1} - w^{t}$

Speeding up gradient descent

Goal: Large steps in flat areas, small steps in high curvature ones, dampened oscillations

- momentum/accelerated methods: combine previous direction with neg. gradient direction

  $$w^{t+1} - w^t = \alpha \left(w^t - w^{t-1}\right) - \eta \nabla_w L\left(w^t\right)$$

  (bonus: discrete approximation of dampened harmonic oscillator)

- adaptive methods (AdaGrad, RMSProp, Adam etc.): Different stepsize for different elements $w_{[i]}$

  intuitively: the elements i which already changed a lot, have smaller stepsize

  $$w_{[i]}^{t+1} = w_{[i]}^t - \frac{\eta}{\sqrt{\mathrm{previouschange_i} + \gamma}} \frac{\partial L}{\partial w_{[i]}}(w^t)$$

- 2nd order methods (Newton): $w^{t+1} = w^t - [\nabla^2 L(w^t)]^{-1} \nabla L(w^t)$

Nice overview blog posts: https://distill.pub/2017/momentum/; https://ruder.io/optimizing-gradient-descent/index.html

Speeding up gradient descent

More on Jupyter!

Goal: Large steps in flat areas, small steps in high curvature ones, dampened oscillations

- momentum/accelerated methods: combine previous direction with neg. gradient direction

  $$w^{t+1} - w^t = \alpha \left(w^t - w^{t-1}\right) - \eta \nabla_w L\left(w^t\right)$$

  (bonus: discrete approximation of dampened harmonic oscillator)

- adaptive methods (AdaGrad, RMSProp, Adam etc.): Different stepsize for different elements $w_{[i]}$

  intuitively: the elements i which already changed a lot, have smaller stepsize

  $$w_{[i]}^{t+1} = w_{[i]}^{t} - \frac{\eta}{\sqrt{\text{previouschange}_\text{i} + \gamma}} \frac{\partial L}{\partial w_{[i]}}(w^t)$$

- 2nd order methods (Newton): $w^{t+1} = w^t - [\nabla^2 L(w^t)]^{-1} \nabla L(w^t)$

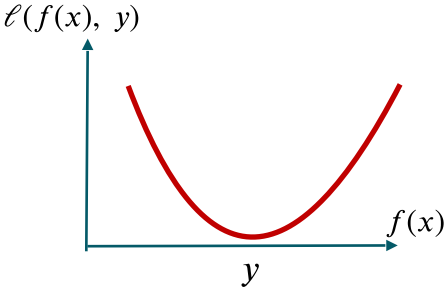
Nice overview blog posts: https://distill.pub/2017/momentum/; https://ruder.io/optimizing-gradient-descent/index.html

Beyond squared loss: Mean absolute error

So far: $\ell_{\text{square}}(f(x), y) := (y - f(x))^2$

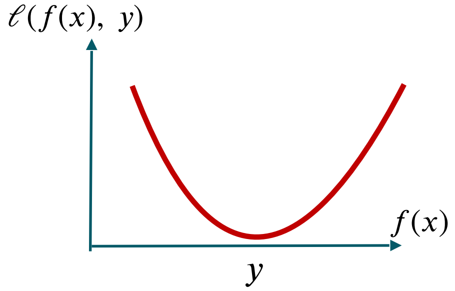
Beyond squared loss: Mean absolute error

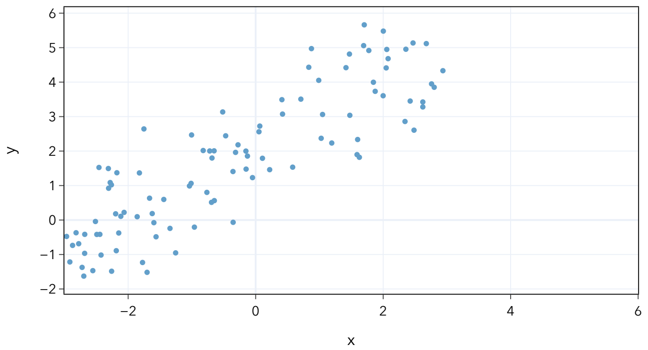
So far: $\ell_{\text{square}}(f(x), y) := (y - f(x))^2$



Beyond squared loss: Mean absolute error

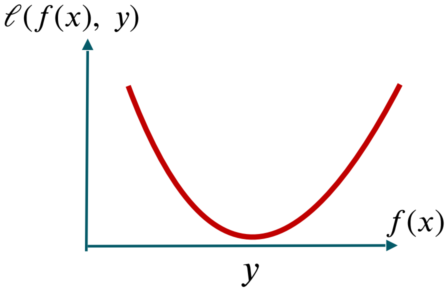
So far: $\ell_{\text{square}}(f(x), y) := (y - f(x))^2$

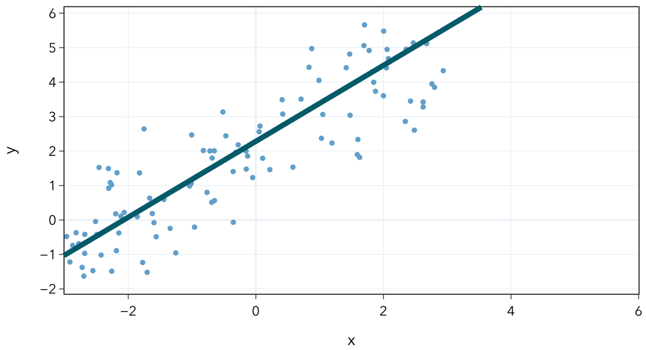




Beyond squared loss: Mean absolute error

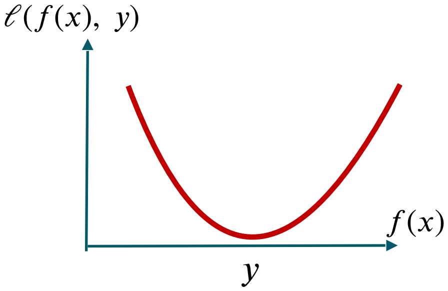
So far: $\ell_{\text{square}}(f(x), y) := (y - f(x))^2$

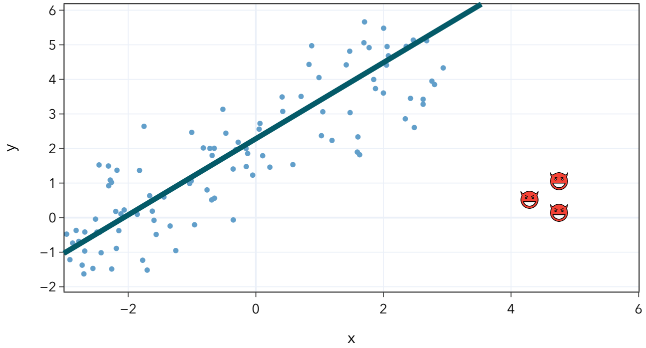




Beyond squared loss: Mean absolute error

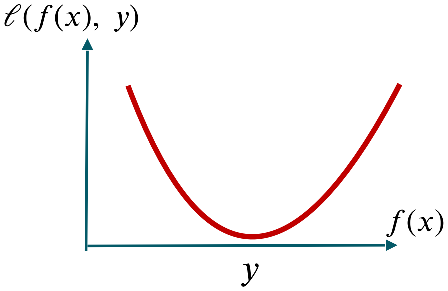
So far: $\ell_{\text{square}}(f(x), y) := (y - f(x))^2$

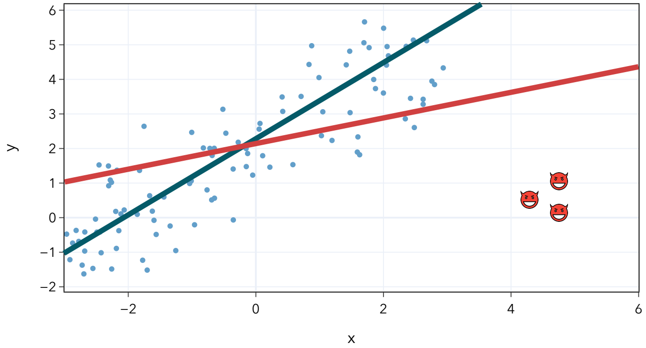




Beyond squared loss: Mean absolute error

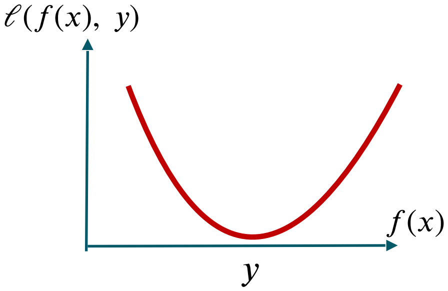
So far: $\ell_{\text{square}}(f(x), y) := (y - f(x))^2$

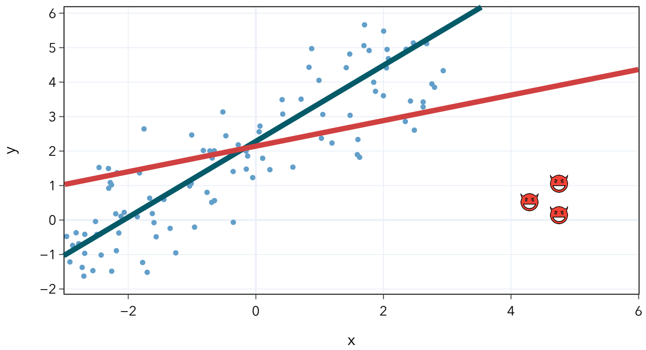




Beyond squared loss: Mean absolute error

So far: $\ell_{\text{square}}(f(x), y) := (y - f(x))^2$





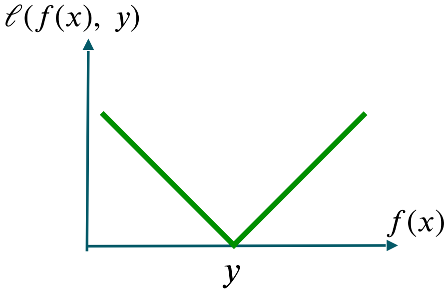
Grows quadratically: huge penalty for large errors!

Beyond squared loss: Mean absolute error

Instead: $\ell_{\text{abs}}(f(x), y) := |y - f(x))|$

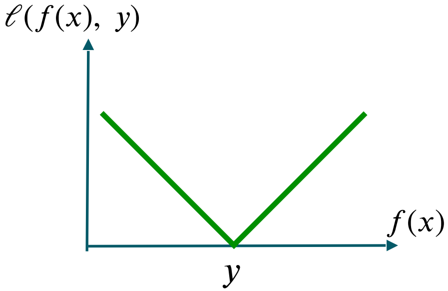
Beyond squared loss: Mean absolute error

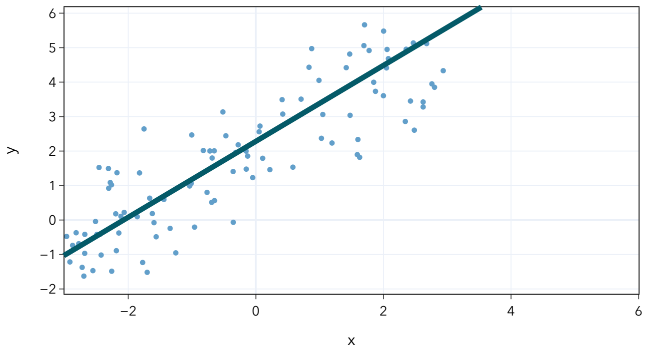
Instead: $\ell_{\text{abs}}(f(x), y) := |y - f(x))|$



Beyond squared loss: Mean absolute error

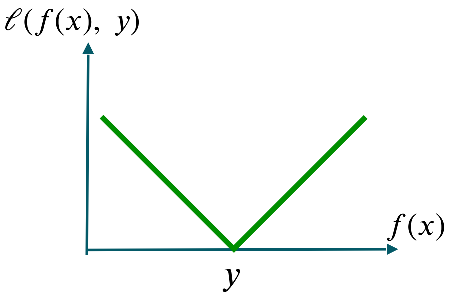
Instead: $\ell_{\text{abs}}(f(x), y) := |y - f(x))|$

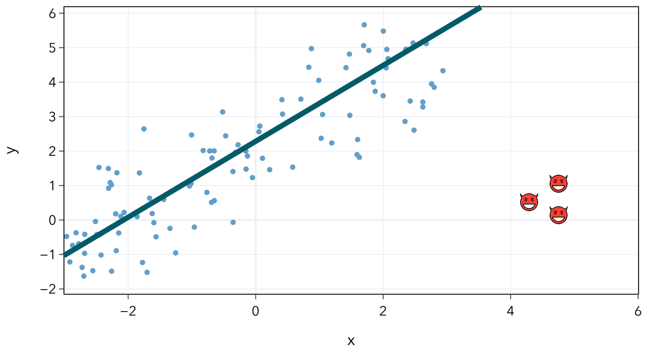




Beyond squared loss: Mean absolute error

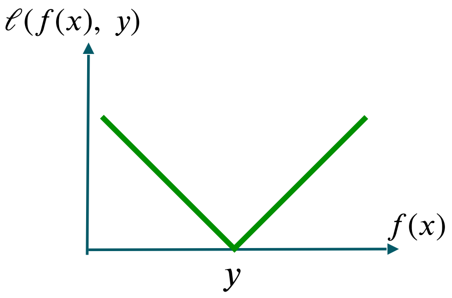
Instead: $\ell_{\text{abs}}(f(x), y) := |y - f(x))|$

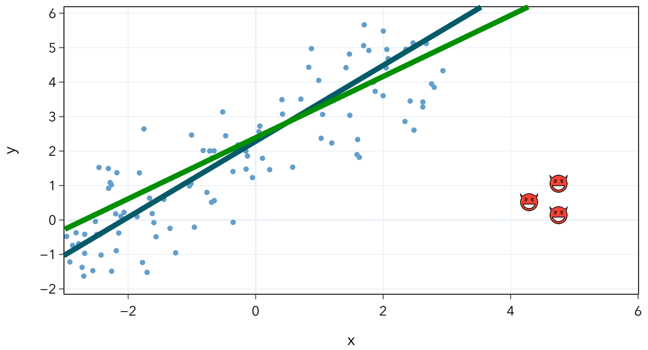




Beyond squared loss: Mean absolute error

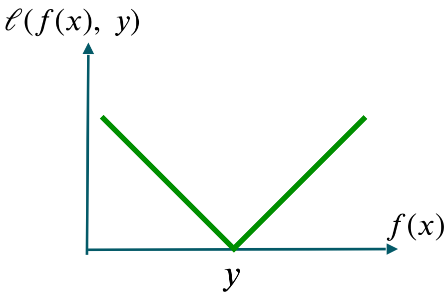
Instead: $\ell_{\text{abs}}(f(x), y) := |y - f(x))|$

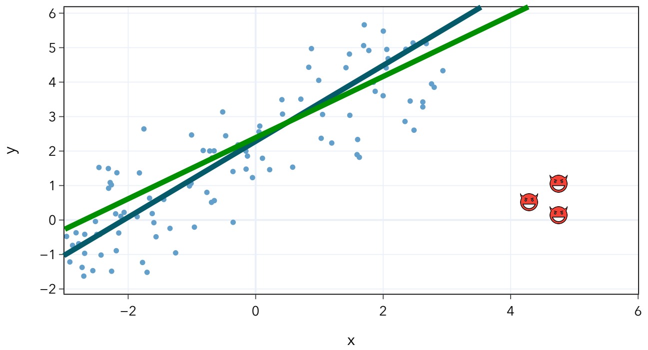




Beyond squared loss: Mean absolute error

Instead: $\ell_{\text{abs}}(f(x), y) := |y - f(x)|$

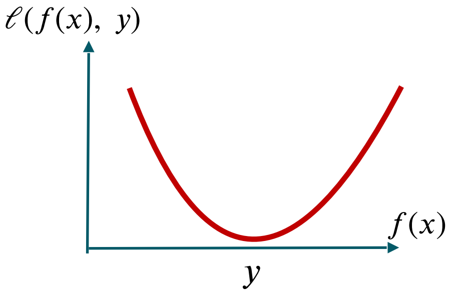




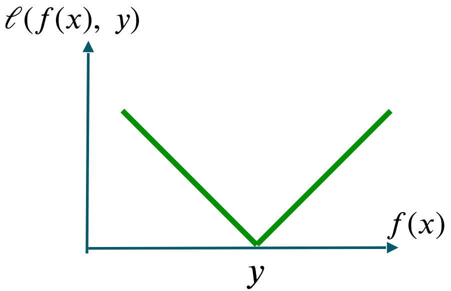
Grows linearly: smaller penalty for large errors!

Mean absolute error is not differentiable

$\ell_{\text{square}}(f(x), y) := (y - f(x))^2$

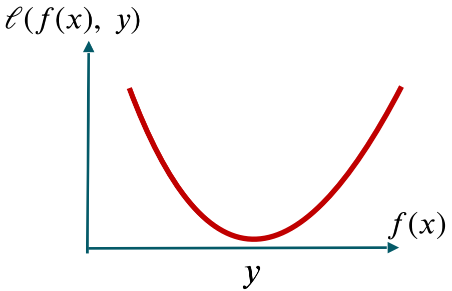


$\ell_{\text{abs}}(f(x), y) := |y - f(x)|$



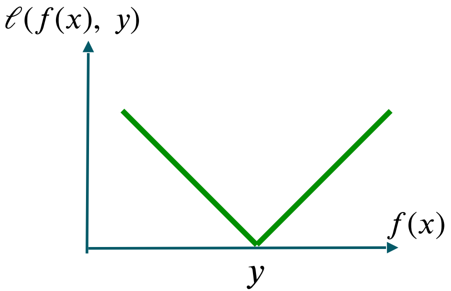
Mean absolute error is not differentiable

$\ell_{\text{square}}(f(x), y) := (y - f(x))^2$



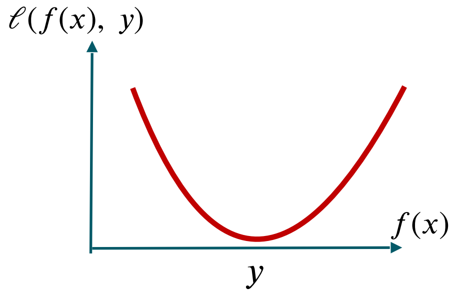
Differentiable

$\ell_{\text{abs}}(f(x), y) := |y - f(x)|$



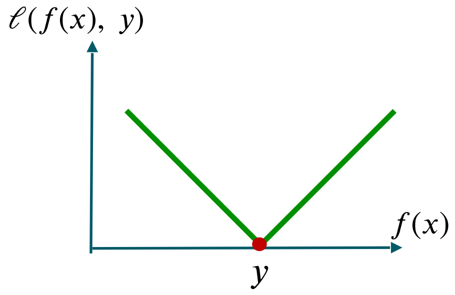
Mean absolute error is not differentiable

$\ell_{\text{square}}(f(x), y) := (y - f(x))^2$



Differentiable

$\ell_{\text{abs}}(f(x), y) := |y - f(x))|$



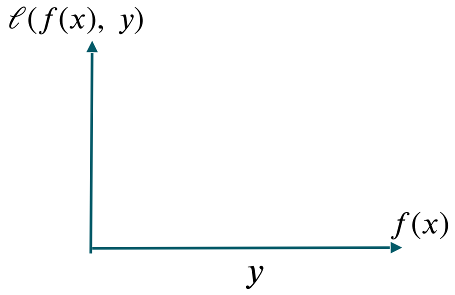
Not differentiable at $f(x) = y$

Best of both worlds: Huber loss

Can we combine ℓ_{square} and ℓ_{abs} in a differentiable function?

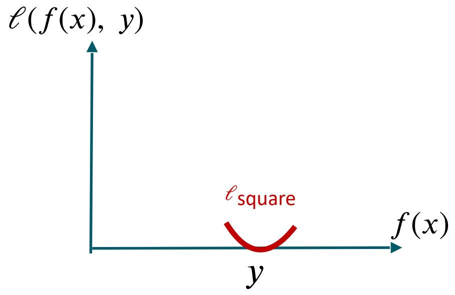
Best of both worlds: Huber loss

Can we combine ℓ_{square} and ℓ_{abs} in a differentiable function?



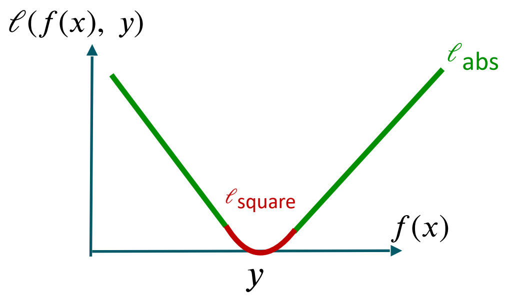
Best of both worlds: Huber loss

Can we combine ℓ_{square} and ℓ_{abs} in a differentiable function?



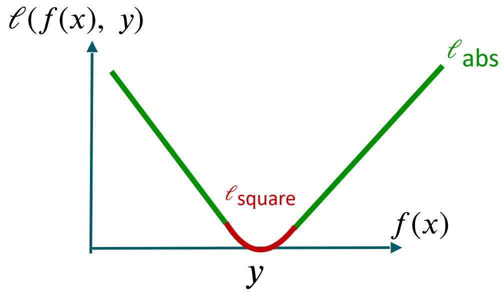
Best of both worlds: Huber loss

Can we combine ℓ_{square} and ℓ_{abs} in a differentiable function?



Best of both worlds: Huber loss

Can we combine ℓ_{square} and ℓ_{abs} in a differentiable function?

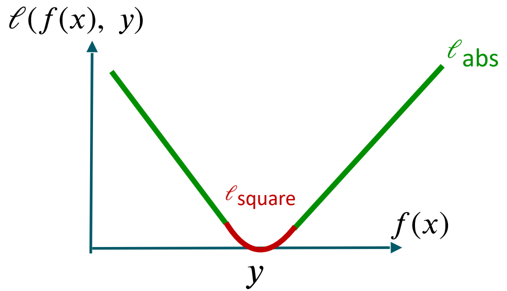


$$\ell_{\text{huber}}(f(x), y) := \begin{cases} (y - f(x))^2 & \text{if } |y - f(x)| \leq \delta \\ 2\delta|y - f(x)| - \delta^2 & \text{else} \end{cases}$$

Best of both worlds: Huber loss

More on iPad and Jupyter!

Can we combine ℓ_{square} and ℓ_{abs} in a differentiable function?



$$\ell_{\text{huber}}(f(x), y) := \begin{cases} (y - f(x))^2 & \text{if } |y - f(x)| \leq \delta \\ 2\delta|y - f(x)| - \delta^2 & \text{else} \end{cases}$$