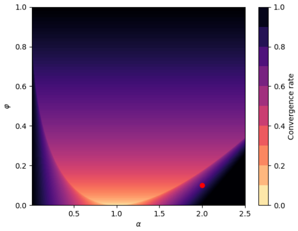
Optimization in Machine Learning

First order methods
Momentum on quadratic functions

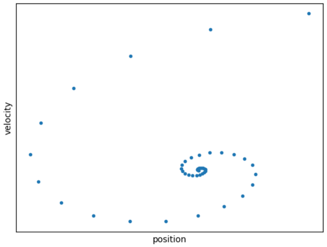


Learning goals

- Effect of φ

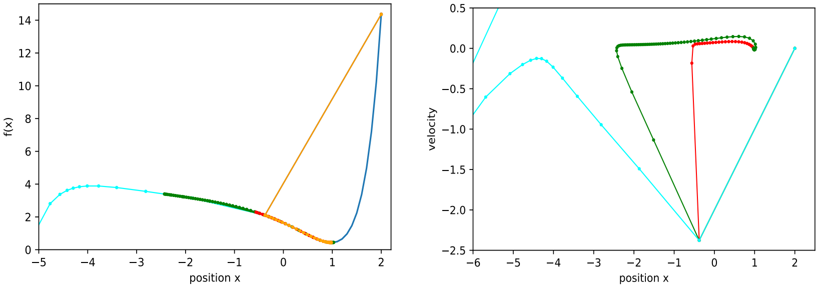
RECAP: DYNAMICS OF MOMENTUM

- Let's investigate the role of φ.
- We can think of gradient descent with momentum as a damped harmonic oscillator: a weight on a spring. We pull the weight down and study the path back to the equilibrium in phase space (looking at the position and the velocity).
- Depending on the choice of φ, the rate of return to the equilibrium position is affected.



RECAP: DYNAMICS OF MOMENTUM / 2

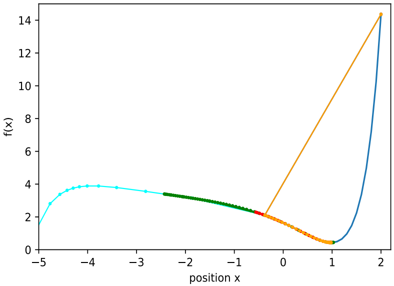
$$f : \mathbb{R} \to \mathbb{R}, x \mapsto -\frac{1}{2}\cos(x)\exp\left(\frac{1}{4}(x+2)^2\right)+3, \qquad x \in [-2, 2], \qquad \alpha = 0,05$$

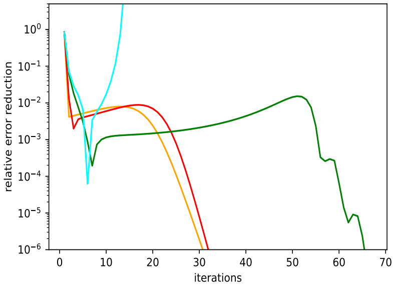


$\phi = 0.0$ $\phi = 0.1$ $\phi = 0.5$ $\phi = 0.65$

RECAP: DYNAMICS OF MOMENTUM / 3

$$f : \mathbb{R} \to \mathbb{R}, x \mapsto -\frac{1}{2}\cos(x)\exp\left(\frac{1}{4}(x+2)^2\right)+3, \qquad x \in [-2, 2], \qquad \alpha = 0,05$$

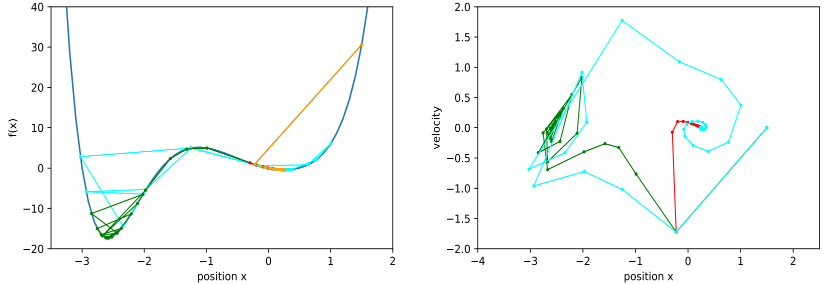




$\phi = 0.0$ $\phi = 0.1$ $\phi = 0.5$ $\phi = 0.65$

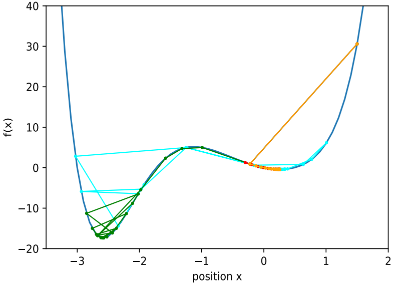
RECAP: DYNAMICS OF MOMENTUM / 4

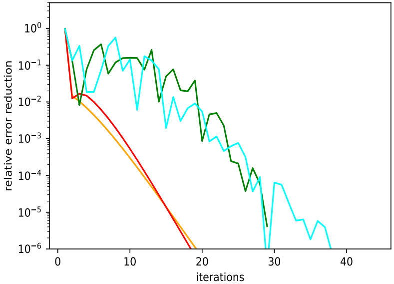
$$f : \mathbb{R} \to \mathbb{R}, x \mapsto \frac{1}{2}x^6 + \frac{3}{2}x^5 + 2x^3 + 5x^2 - 3x, \qquad x \in [-\frac{9}{2}, 2], \qquad \alpha = 0,02$$



$\phi = 0.0$ $\phi = 0.1$ $\phi = 0.5$ $\phi = 0.65$

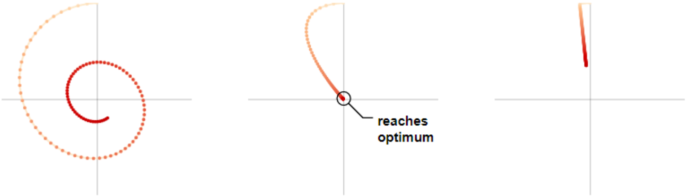
$$f : \mathbb{R} \to \mathbb{R}, x \mapsto \frac{1}{2}x^6 + \frac{3}{2}x^5 + 2x^3 + 5x^2 - 3x, \qquad x \in [-\frac{9}{2}, 2], \qquad \alpha = 0,02$$





$\phi = 0.0$ $\phi = 0.1$ $\phi = 0.5$ $\phi = 0.65$

RECAP: DYNAMICS OF MOMENTUM / 6



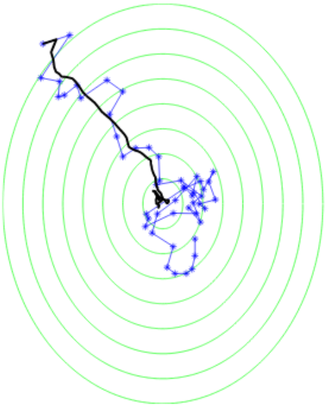
Left: If φ is too large, we are underdamping. The spring oscillates back and forth and misses the optimum.

Middle: The best value of φ lies in the middle.

Right: If φ is too small, we are overdamping, meaning that the spring experiences too much friction and stops before reaching the equilibrium.

Optimization in Machine Learning

First order methods
SGD



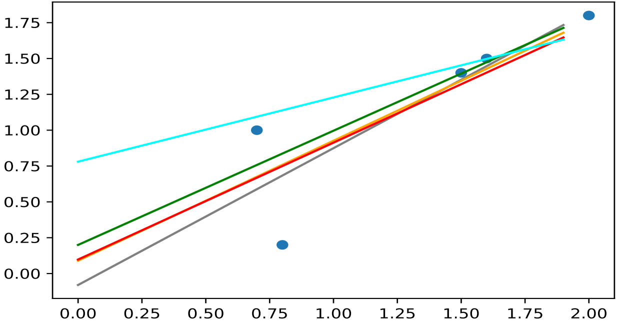
Learning goals

- SGD
- Stochasticity
- Convergence
- Batch size

STOCHASTIC GRADIENT DESCENT

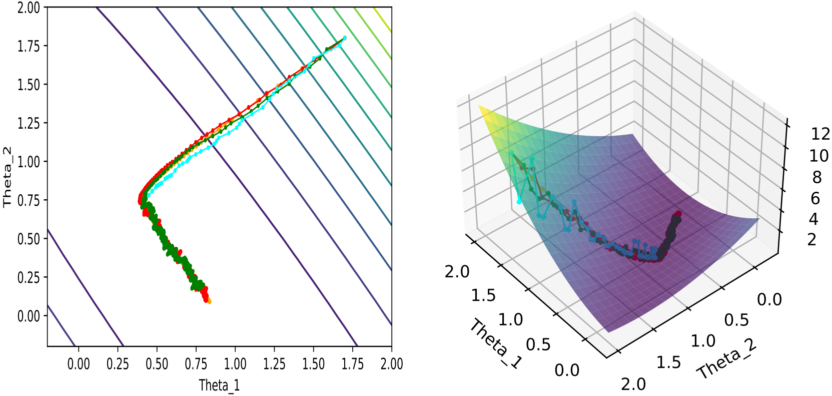
Issue: Data-sets might be very large and gradients expensive to evaluate

Idea: Use a smaller (random) subset of data-points to evaluate gradient and objective-function



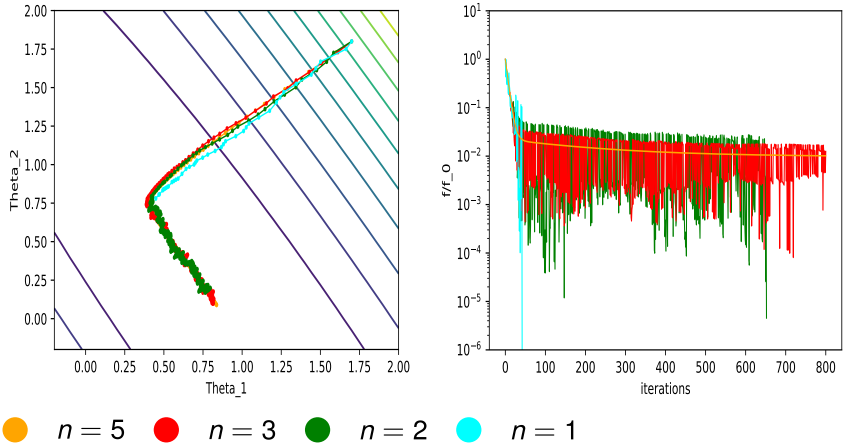
$n = 5$ $n = 3$ $n = 2$ $n = 1$

STOCHASTIC GRADIENT DESCENT / 2



$n = 5$ $n = 3$ $n = 2$ $n = 1$

STOCHASTIC GRADIENT DESCENT / 3



STOCHASTIC GRADIENT DESCENT / 4

Analysis of curvature:

Least-Squares objective function:

$$f(\theta) = \min_{\boldsymbol{\theta} \in \mathbb{R}^d} \sum_{i=1}^{n} \left(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} - y^{(i)} \right)^2 = (\mathbf{X}\theta - \mathbf{y})^T (\mathbf{X}\theta - \mathbf{y})$$

1st derivative of objective function:

$$\nabla f(\theta) = \frac{1}{2} \left(\mathbf{X}^T \mathbf{X} \right) \theta - \mathbf{X}^T \mathbf{y}$$

2nd derivative of objective function:

$$\nabla^2 f(\theta) = \frac{1}{2} \left(\mathbf{X}^T \mathbf{X} \right)$$

$\Rightarrow$ Hessian is only dependent of x-coordinates of data-points

STOCHASTIC GRADIENT DESCENT / 5

Hessian matrix:

$$\mathbf{H} = \nabla^2 f(\theta) = \frac{1}{2}\left(\mathbf{X}^T\mathbf{X}\right) = \frac{1}{2}\begin{bmatrix} 4.97 & 3.3 \\ 3.3 & 2.5 \end{bmatrix}$$

Eigenvalues:

$$det(\mathbf{H} - \mathbf{I}\lambda) = 0$$

solve for λ:

$$(H_{11} - \lambda)(H_{22} - \lambda) - H_{21}H_{12} = \lambda^2 - 7.47\lambda + 1.535 = 0$$

$$\lambda_1 = 7.2585, \qquad \lambda_2 = 0.2115$$

$$\kappa(\mathbf{H}) = \lambda_1/\lambda_2 \approx 34.3191$$

$\Rightarrow$ Rather high condition-number compared to matrix-values, positive definite matrix

STOCHASTIC GRADIENT DESCENT / 6

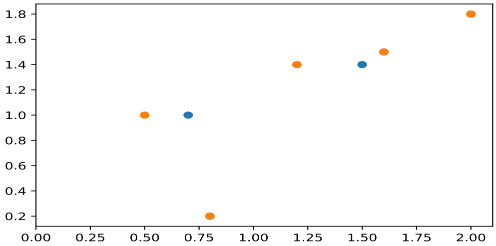
Clustered data-points increase condition-number for Least-Squares method:

$$x_{old}^{(i)} = (0.7, 0.8, 1.5, 1.6, 2.0)$$

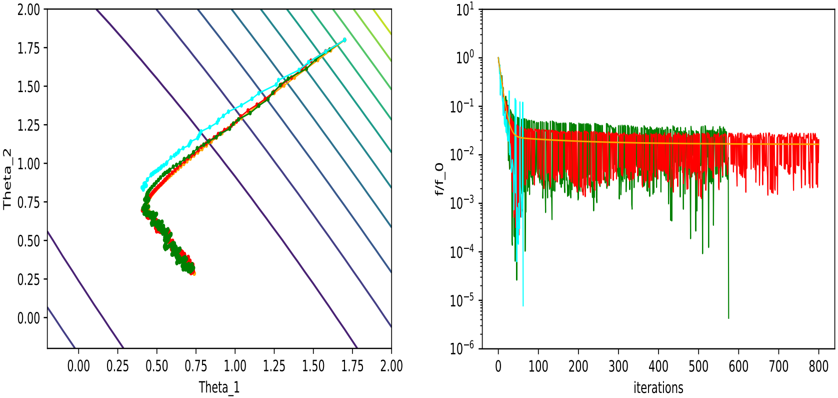
$$x_{new}^{(i)} = (0.5, 0.8, 1.2, 1.6, 2.0)$$

$$\kappa(\mathbf{H}_{new}) = \lambda_1/\lambda_2 \approx 24.6074$$

$\Rightarrow$ More evenly distributed data-points improve conditioning



STOCHASTIC GRADIENT DESCENT / 7



$n = 5$ $n = 3$ $n = 2$ 1

STOCHASTIC GRADIENT DESCENT / 8

NB: We use g instead of f as objective, bc. f is used as model in ML.

$g : \mathbb{R}^d \to \mathbb{R}$ objective, g **average over functions**:

$$g(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} g_i(\mathbf{x}), \qquad g \text{ and } g_i \text{ smooth}$$

Stochastic gradient descent (SGD) approximates the gradient

$$\begin{aligned} \nabla_{\mathbf{x}}\, g(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} \nabla_{\mathbf{x}}\, g_i(\mathbf{x}) \quad &:= \quad \mathbf{d} \quad \text{by} \\ \frac{1}{|J|} \sum_{i \in J} \nabla_{\mathbf{x}}\, g_i(\mathbf{x}) \quad &:= \quad \hat{\mathbf{d}}, \end{aligned}$$

with random subset $J \subset \{1, 2, ..., n\}$ of gradients called **mini-batch**.
This is done e.g. when computing the true gradient is **expensive**.

STOCHASTIC GRADIENT DESCENT / 9

Algorithm Basic SGD pseudo code

Initialize $\mathbf{x}^{[0]}$, $t = 0$
while stopping criterion not met **do**
  Randomly shuffle indices and partition into minibatches $J_1, \ldots, J_K$ of size m
  for $k \in \{1, \ldots, K\}$ **do**
    $t \leftarrow t + 1$
    Compute gradient estimate with J_k: $\hat{\mathbf{d}}^{[t]} \leftarrow \frac{1}{m} \sum_{i \in J_k} \nabla_{\mathbf{x}} g_i(\mathbf{x}^{[t-1]})$
    Apply update: $\mathbf{x}^{[t]} \leftarrow \mathbf{x}^{[t-1]} - \alpha \cdot \hat{\mathbf{d}}^{[t]}$
  end for
end while

- Instead of drawing batches randomly we might want to go through the g_i sequentially (unless g_i are sorted in any way)
- Updates are computed faster, but also more stochastic:
  - In the simplest case, batch-size $m := |J_k|$ is set to $m = 1$
  - If n is a billion, computation of update is a billion times faster
  - **But** (later): Convergence rates suffer from stochasticity!

SGD IN ML

In ML, we perform ERM:

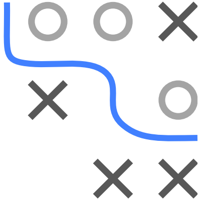
$$\mathcal{R}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \underbrace{L\left(y^{(i)}, f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)\right)}_{g_i(\boldsymbol{\theta})}$$

- for a data set

$$\mathcal{D} = \left(\left(\mathbf{x}^{(1)}, y^{(1)}\right), \dots, \left(\mathbf{x}^{(n)}, y^{(n)}\right)\right)$$

- a loss function $L(y, f(\mathbf{x}))$, e.g., L2 loss $L(y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2$,
- and a model class f, e.g., the linear model $f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right) = \boldsymbol{\theta}^\top \mathbf{x}$.

SGD IN ML / 2



For large data sets, computing the exact gradient

$$\mathbf{d} = \frac{1}{n}\sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} L\left(y^{(i)}, f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)\right)$$

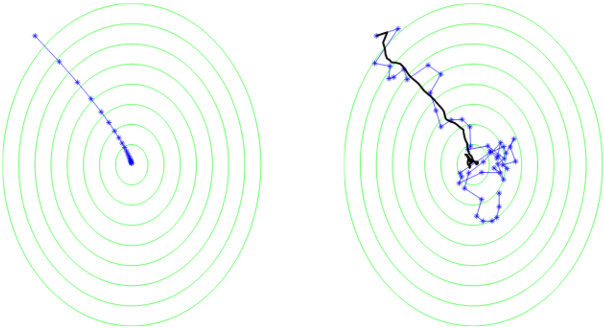
may be expensive or even infeasible to compute and is approximated by

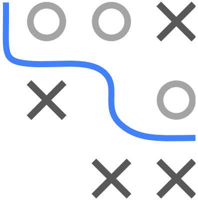
$$\hat{\mathbf{d}} = \frac{1}{m}\sum_{i \in J} \nabla_{\boldsymbol{\theta}} L\left(y^{(i)}, f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)\right),$$

for $J \subset 1, 2, ..., n$ random subset.

NB: Often, maximum size of J technically limited by memory size.

STOCHASTICITY OF SGD





Minimize $g(x_1, x_2) = 1.25(x_1 + 6)^2 + (x_2 - 8)^2$.

Left: GD. **Right:** SGD. Black line shows average value across multiple runs.

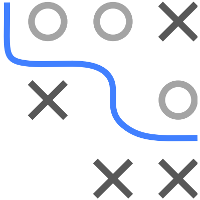
(Source: Shalev-Shwartz et al., Understanding Machine Learning, 2014.)

STOCHASTICITY OF SGD / 2

Assume batch size $m = 1$ (statements also apply for larger batches).

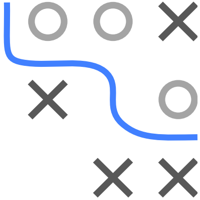
- **(Possibly) suboptimal direction:** Approximate gradient $\hat{\mathbf{d}} = \nabla_{\mathbf{x}} g_i(\mathbf{x})$ might point in suboptimal (possibly not even a descent!) direction
- **Unbiased estimate:** If J drawn i.i.d., approximate gradient $\hat{\mathbf{d}}$ is an unbiased estimate of gradient $\mathbf{d} = \nabla_{\mathbf{x}} g(\mathbf{x}) = \sum_{i=1}^{n} \nabla_{\mathbf{x}} g_i(\mathbf{x})$:

$$
\begin{aligned}
\mathbb{E}_i \left[\nabla_{\mathbf{x}} g_i(\mathbf{x}) \right] &= \sum_{i=1}^{n} \nabla_{\mathbf{x}} g_i(\mathbf{x}) \cdot \mathbb{P}(i = i) \\
&= \sum_{i=1}^{n} \nabla_{\mathbf{x}} g_i(\mathbf{x}) \cdot \frac{1}{n} = \nabla_{\mathbf{x}} g(\mathbf{x}).
\end{aligned}
$$



Conclusion: SGD might perform single suboptimal moves, but moves in “right direction” **on average**.

CONVERGENCE OF SGD



As a consequence, SGD has worse convergence properties than GD.

But: Can be controlled via **increasing batches** or **reducing step size**.

The larger the batch size m

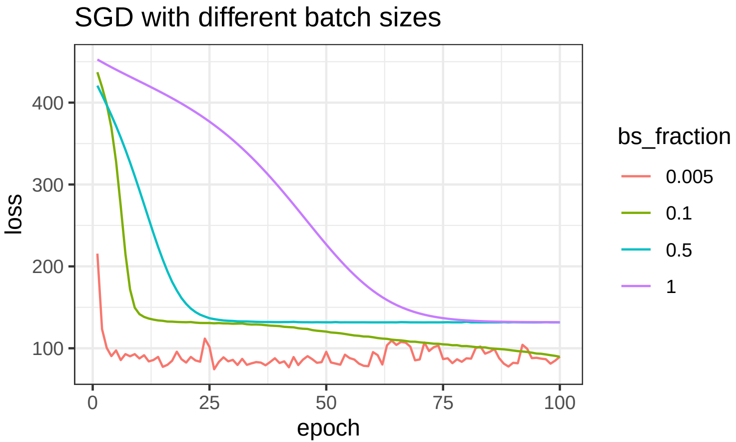
- the better the approximation to $\nabla_{\mathbf{x}} g(\mathbf{x})$
- the lower the variance
- the lower the risk of performing steps in the wrong direction

The smaller the step size α

- the smaller a step in a potentially wrong direction
- the lower the effect of high variance

As maximum batch size is usually limited by computational resources (memory), choosing the step size is crucial.

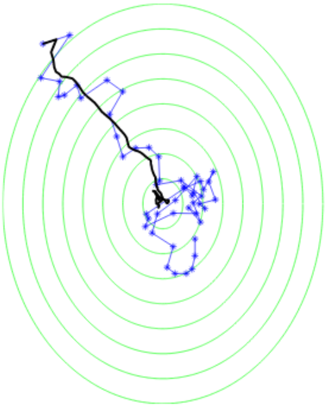
EFFECT OF BATCH SIZE



SGD for a NN with batch size $\in \{0.5\%, 10\%, 50\%\}$ of the training data.
The higher the batch size, the lower the variance.

Optimization in Machine Learning

First order methods
SGD Further Details

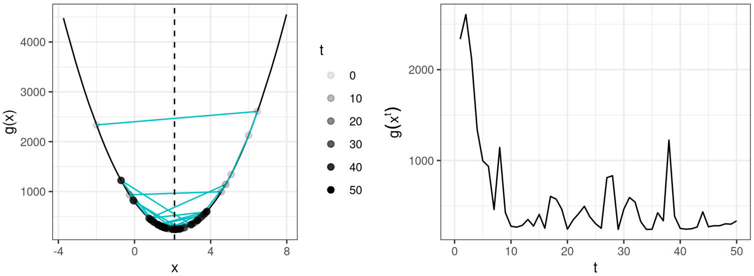


Learning goals

- Decreasing step size for SGD
- Stopping rules
- SGD with momentum

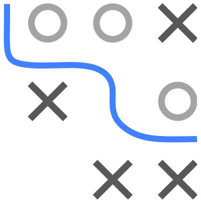
SGD WITH CONSTANT STEP SIZE

Example: SGD with constant step size.



Fast convergence of SGD initially. Erratic behavior later (variance too big).

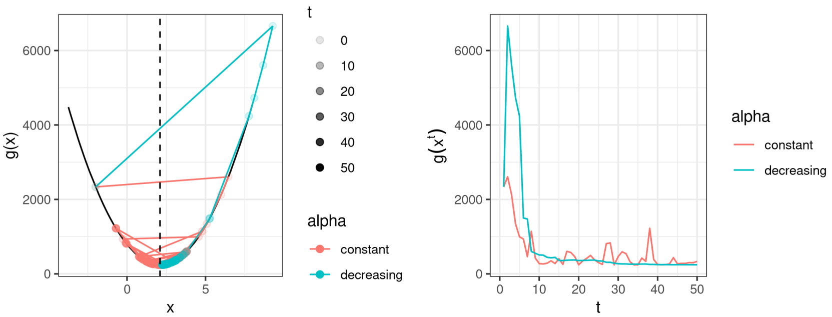
SGD WITH DECREASING STEP SIZE



- **Idea:** Decrease step size to reduce magnitude of erratic steps.
- **Trade-off:**
  - if step size $\alpha^{[t]}$ decreases slowly, large erratic steps
  - if step size decreases too fast, performance is impaired

SGD WITH DECREASING STEP SIZE / 2

- Popular solution: step size fulfilling $\alpha^{[t]} = \alpha^{[0]}/t$).



Example continued. Step size $\alpha^{[t]} = 0.2/t$.

- Often not working well in practice: step size gets small quite fast.
- Alternative: $\alpha^{[t]} = \alpha^{[0]}/\sqrt{t}$)

ADVANCED STEP SIZE CONTROL

Why not Armijo-based step size control?

- Backtracking line search or other approaches based on Armijo rule usually not suitable: Armijo condition

$$g(\mathbf{x} + \alpha \mathbf{d}) \leq g(\mathbf{x}) + \gamma_1 \alpha \nabla g(\mathbf{x})^\top \mathbf{d}$$

  requires evaluating full gradient.

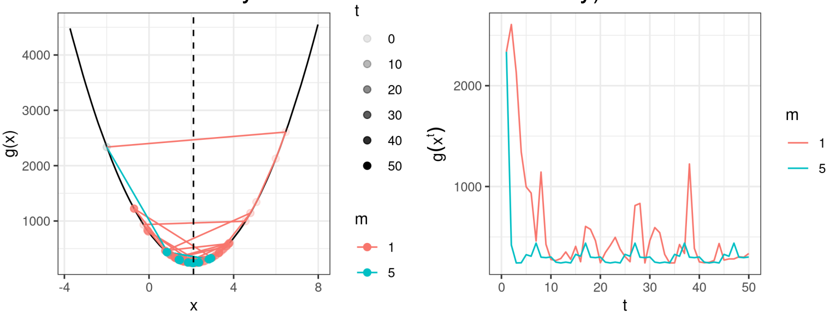
- But SGD is used to *avoid* expensive gradient computations.
- Research aims at finding inexact line search methods that provide better convergence behaviour, e.g., Vaswani et al., *Painless Stochastic Gradient: Interpolation, Line-Search, and Convergence Rates*. NeurIPS, 2019.

MINI-BATCHES

- Reduce noise by increasing batch size m for better approximation

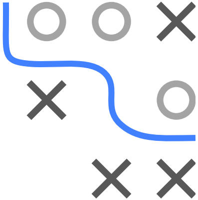
$$\hat{\mathbf{d}} = \frac{1}{m} \sum_{i \in J} \nabla_{\mathbf{x}} g_i(\mathbf{x}) \approx \frac{1}{n} \sum_{i=1}^{n} \nabla_{\mathbf{x}} g_i(\mathbf{x}) = \mathbf{d}$$

- Usually, the batch size is limited by computational resources (e.g., how much data you can load into the memory)



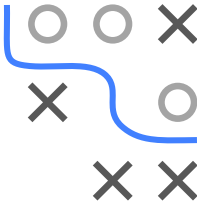
Example continued. Batch size $m = 1$ vs. $m = 5$.

STOPPING RULES FOR SGD



- **For GD**: We usually stop when gradient is close to 0 (i.e., we are close to a stationary point)
- **For SGD**: individual gradients do not necessarily go to zero, and we cannot access full gradient.
- Practicable solution for ML:
  - Measure the validation set error after T iterations
  - Stop if validation set error is not improving

SGD AND ML



General remarks:

- SGD is a variant of GD
- SGD particularly suitable for large-scale ML when evaluating gradient is too expensive / restricted by computational resources
- SGD and variants are the most commonly used methods in modern ML, for example:
  - Linear models

    Note that even for the linear model and quadratic loss, where a closed form solution is available, SGD might be used if the size n of the dataset is too large and the design matrix does not fit into memory.
  - Neural networks
  - Support vector machines
  - ...

SGD WITH MOMENTUM

SGD is usually used with momentum due to reasons mentioned in previous chapters.

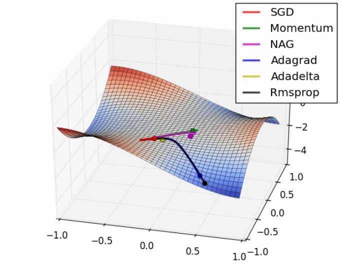
Algorithm Stochastic gradient descent with momentum

---

require step size α and momentum φ
require initial parameter $\boldsymbol{x}$ and initial velocity $\boldsymbol{\nu}$
while stopping criterion not met **do**
    Sample mini-batch of m examples
    Compute gradient estimate $\nabla\hat{g}(\mathbf{x})$ using mini-batch
    Compute velocity update: $\boldsymbol{\nu} \leftarrow \varphi\boldsymbol{\nu} - \alpha\nabla\hat{g}(\mathbf{x})$
    Apply update: $\boldsymbol{x} \leftarrow \boldsymbol{x} + \boldsymbol{\nu}$
end while

Optimization in Machine Learning

First order methods
Adam and friends

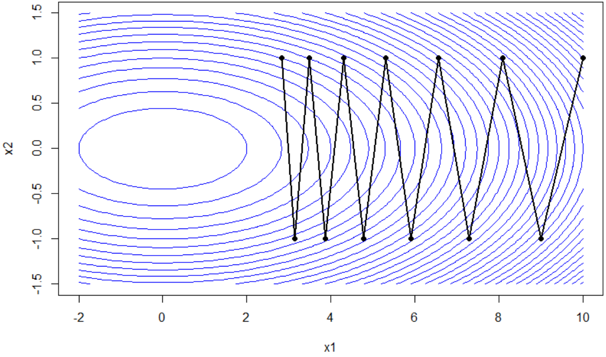


Learning goals

- Adaptive step sizes
- Adam

ADAPTIVE STEP SIZES

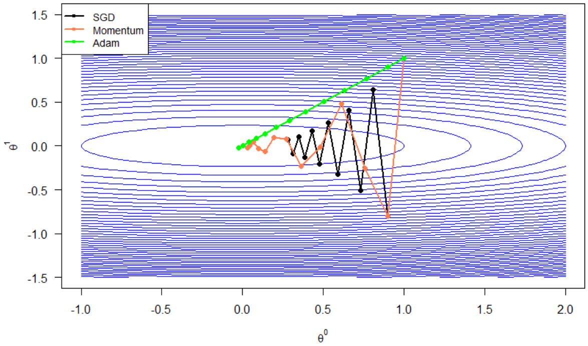
- Step size is probably the most important control parameter
- Has strong influence on performance
- Natural to use different step size for each input individually and automatically adapt them



ADAM

- Adaptive Moment Estimation also has adaptive step sizes
- Uses the 1st and 2nd moments of gradients
    - Keeps an exponentially decaying average of past gradients (1st moment)
    - Like RMSProp, stores an exponentially decaying avgerage of past squared gradients (2nd moment)
    - Can be seen as combo of RMSProp + momentum.

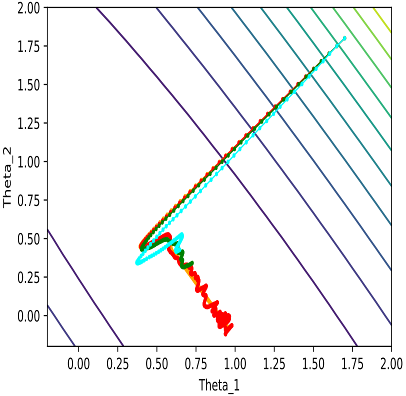
COMPARISON ON QUADRATIC FORM

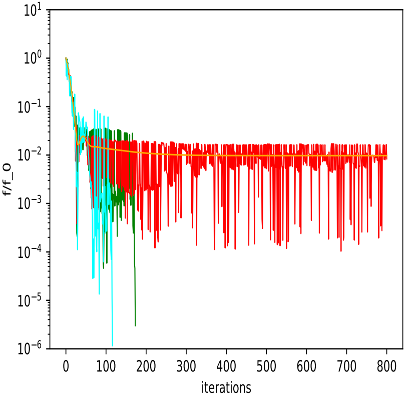


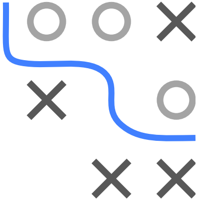
SGD vs. SGD with Momentum vs. Adam on a quadratic form.

ADAM

Least-Squares:







$n = 5$ $n = 3$ $n = 2$ $n = 1$

ADAM / 2

Algorithm Adam

require Global step size α (suggested default: 0.001)
require Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$ (suggested defaults: 0.9 and 0.999 respectively)
require Small constant β (suggested default 10^{-8})
require Initial parameters $\boldsymbol{\theta}$
Initialize time step $t = 0$
Initialize 1st and 2nd moment variables $\mathbf{s}^{[0]} = 0, \mathbf{r}^{[0]} = 0$
while stopping criterion not met **do**
    $t \leftarrow t + 1$
    Sample a minibatch of m examples from the training set $\{\tilde{x}^{(1)}, \dots, \tilde{x}^{(m)}\}$
    Compute gradient estimate: $\hat{\mathbf{g}}^{[t]} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L\left(y^{(i)}, f\left(\tilde{\mathbf{x}}^{(i)} \mid \boldsymbol{\theta}\right)\right)$
    Update biased first moment estimate: $\mathbf{s}^{[t]} \leftarrow \rho_1 \mathbf{s}^{[t-1]} + (1 - \rho_1)\hat{\mathbf{g}}^{[t]}$
    Update biased second moment estimate: $\mathbf{r}^{[t]} \leftarrow \rho_2 \mathbf{r}^{[t-1]} + (1 - \rho_2)\hat{\mathbf{g}}^{[t]^2}$
    Correct bias in first moment: $\hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}^{[t]}}{1-\rho_1^t}$
    Correct bias in second moment: $\hat{\mathbf{r}} \leftarrow \frac{\mathbf{r}^{[t]}}{1-\rho_2^t}$
    Compute update for each entry i: $\Delta\boldsymbol{\theta}_i = -\alpha \frac{\hat{\mathbf{s}}_i}{\sqrt{\hat{\mathbf{r}}_i}+\beta}$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
end while

ADAM / 3

- Initializes moment variables $\mathbf{s}$ and $\mathbf{r}$ with zero $\Rightarrow$ Bias towards zero
- Indeed: Unrolling $\mathbf{s}^{[t]}$ yields

$$\mathbf{s}^{[0]} = 0$$

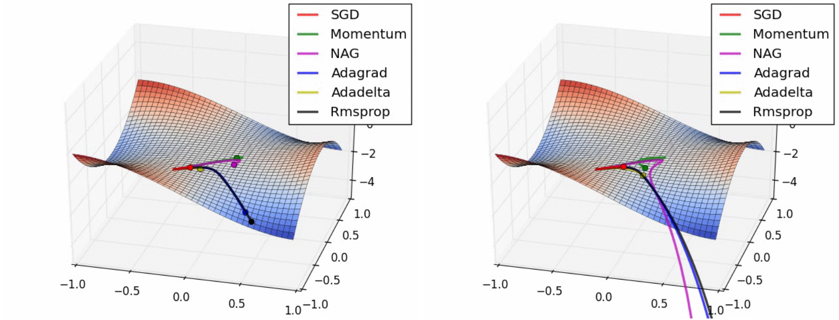
$$\mathbf{s}^{[1]} = \rho_1 \mathbf{s}^{[0]} + (1-\rho_1)\hat{\mathbf{g}}^{[1]} = (1-\rho_1)\hat{\mathbf{g}}^{[1]}$$

$$\mathbf{s}^{[2]} = \rho_1 \mathbf{s}^{[1]} + (1-\rho_1)\hat{\mathbf{g}}^{[2]} = \rho_1(1-\rho_1)\hat{\mathbf{g}}^{[1]} + (1-\rho_1)\hat{\mathbf{g}}^{[2]}$$

$$\mathbf{s}^{[3]} = \rho_1 \mathbf{s}^{[2]} + (1-\rho_1)\hat{\mathbf{g}}^{[3]} = \rho_1^2(1-\rho_1)\hat{\mathbf{g}}^{[1]} + \rho_1(1-\rho_1)\hat{\mathbf{g}}^{[2]} + (1-\rho_1)\hat{\mathbf{g}}^{[3]}$$

- Therefore: $\mathbf{s}^{[t]} = (1-\rho_1)\sum_{i=1}^{t} \rho_1^{t-i}\hat{\mathbf{g}}^{[i]}$.
- Therefore: $\mathbf{s}^{[t]}$ is a biased estimator of $\hat{\mathbf{g}}^{[t]}$
- **Note:** Contributions of past $\hat{\mathbf{g}}^{[i]}$ decreases rapidly and bias vanishes for $t \to \infty$ ($\rho_1^t \to 0$)
- We correct for the bias by $\hat{\mathbf{s}}^{[t]} = \frac{\mathbf{s}^{[t]}}{(1-\rho_1^t)}$
- Analogously: $\hat{\mathbf{r}}^{[t]} = \frac{\mathbf{r}^{[t]}}{(1-\rho_2^t)}$

COMPARISON OF OPTIMIZERS: ANIMATION



Credits: Dettmers (2015) and Radford

Comparison of SGD optimizers near saddle point.

Left: After start. **Right:** Later.

All methods accelerate compared to vanilla SGD.

Best is RMSProp, then AdaGrad. (Adam is missing here.)