DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

# Stochastic Gradient Descent

Machine Learning 2023-24
UML book chapter 14 (the slides contain a simplified presentation)
Slides P. Zanuttigh (some material F. Vandin, S. Rudes)

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

# Minimize a Differentiable Function

- *The task*: Need a general approach to minimize a differentiable convex function $f(\boldsymbol{w})$ with respect to the (weights') vector $\boldsymbol{w}$
  - Notice: the dimensionality of $\boldsymbol{w}$ can be very large
- *Recall:* the gradient $\nabla f(\boldsymbol{w})$ of a differentiable function $f: \mathbb{R}^d \rightarrow \mathbb{R}$ is:

$$\nabla f(\boldsymbol{w}) = \left(\frac{\partial f(\boldsymbol{w})}{\partial w_1}, \dots, \frac{\partial f(\boldsymbol{w})}{\partial w_d}\right)$$

- *Idea:* the gradient points in the direction of the largest increase of *f* in the region close to $\boldsymbol{w}$
- Move in the opposite direction until you find a minima
- Gradient corresponds to first order Taylor approximation
  - First order Taylor: $f(\boldsymbol{u}) \approx f(\boldsymbol{w}) + < \boldsymbol{u} - \boldsymbol{w}, \nabla f(\boldsymbol{w}) >$
  - Good approximation for small steps → need to move step by step
- The theory can be extended to non-differentiable functions using subgradients (*if interested see the book, not part of the course*)

# Gradient Descent (GD)

General approach to minimize a differentiable convex function $f(\boldsymbol{w})$

*GD algorithm:*

$\boldsymbol{w}^{(0)} \leftarrow \boldsymbol{0}$

for $t \leftarrow 0$ to $T-1$ do

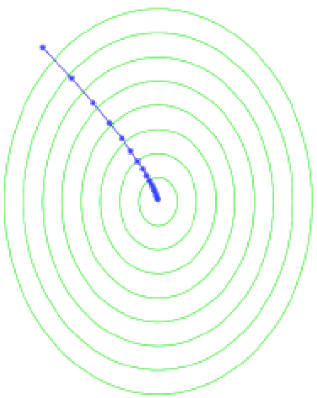
$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta \nabla f(\boldsymbol{w}^{(t)})$$

return $\bar{\boldsymbol{w}} = \boldsymbol{w}^{(T)}$

Total number of steps: T

negative sign: move in opposite direction

Learning rate
(controls the size of the steps)



- Start from an initial point
  - e.g., $\boldsymbol{w}^{(0)} = \boldsymbol{0}$ or random value or initial guess (and many other strategies)
- At each step move in direction opposite to the gradient
- Stop when solution does not improve or max iterations reached
- Get the final point or the one corresponding to minimum value of the objective function

# Gradient Descent: Accuracy and Convergence

*Hypothesis:*

- $f(\boldsymbol{w})$ is a convex $\rho$-Lipschitz function
  - recall $\rho$-Lipschitz: $\|f(\boldsymbol{w_1}) - f(\boldsymbol{w_2})\| \leq \rho\|\boldsymbol{w_1} - \boldsymbol{w_2}\|$
- $\boldsymbol{w}^* \in \operatorname{argmin}_{\{\boldsymbol{w}:\|\boldsymbol{w}\| \leq B\}} f(\boldsymbol{w})$
  - $f(\boldsymbol{w}^*)$ is a minima for $\|\boldsymbol{w}\| \leq \mathrm{B}$

*Then:*

- If we run the GD algorithm on $f$ for $T$ steps with $\eta = \sqrt{\frac{B^2}{\rho^2 T}}$, then the output vector $\overline{\boldsymbol{w}}$ satisfies: $f(\overline{\boldsymbol{w}}) - f(\boldsymbol{w}^*) \leq \frac{B\rho}{\sqrt{T}}$

Output of GD

Minima (e.g., ERM solution)

*Demonstration not part of the course*

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

# Gradient Descent: Corollary

Theorem:

Hypothesis: $f(\boldsymbol{w})$ convex $\rho$-Lipschitz function, $\boldsymbol{w}^* \in \operatorname{argmin}_{\{\boldsymbol{w}:\|\boldsymbol{w}\| \leq B\}} f(\boldsymbol{w})$

Thesis :If we run the GD algorithm on $f$ for $T$ steps with $\eta = \sqrt{\frac{B^2}{\rho^2 T}}$, then the output vector $\overline{\boldsymbol{w}}$ satisfies: $f(\overline{\boldsymbol{w}}) - f(\boldsymbol{w}^*) \leq \frac{B\rho}{\sqrt{T}}$

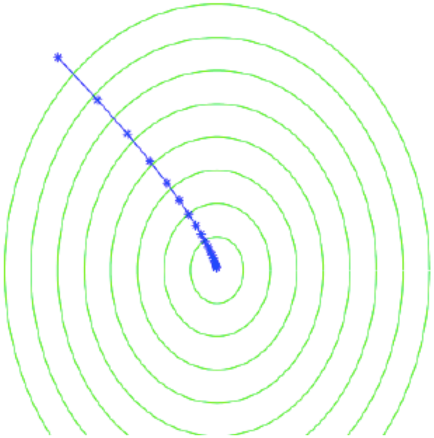
Corollary:

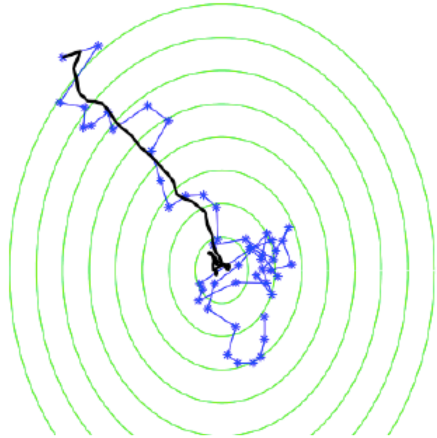
For every $\epsilon > 0$, to achieve $f(\overline{\boldsymbol{w}}) - f(\boldsymbol{w}^*) \leq \epsilon$ it suffices to run the GD algorithm for a number of iterations that satisfies $T \geq \frac{B^2\rho^2}{\epsilon^2}$

*Demonstration:*

- Theorem: If we run for $T$ iterations we get that $f(\overline{\boldsymbol{w}}) - f(\boldsymbol{w}^*) \leq \frac{B\rho}{\sqrt{T}}$
- Need $\frac{B\rho}{\sqrt{T}} \leq \epsilon \rightarrow \sqrt{T} \geq \frac{B\rho}{\epsilon} \rightarrow T \geq \frac{B^2\rho^2}{\epsilon^2}$

# Sthocastic Gradient Descent (SGD)






Example with function
*$1.25(x + 6)^2 + (y - 8)^2$*

- Computing the gradient at each step is computationally demanding
  →in ML: need to loop through the whole training set at each iteration

→ avoid using exactly the gradient
SGD: take a (random) vector with expected value equal to the gradient direction

*SGD algorithm:*

$\boldsymbol{w}^{(0)} \leftarrow \boldsymbol{0}$

for $t \leftarrow 0$ to $T-1$ do

choose $\boldsymbol{v}_t$ at random from a distribution such that $\mathbb{E}\left[\boldsymbol{v}_t \middle| \boldsymbol{w}^{(t)}\right] = \nabla f(\boldsymbol{w}^{(t)})$

$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta \boldsymbol{v}_t$

return $\bar{\boldsymbol{w}} = \boldsymbol{w}^{(T)}$ (or $\bar{\boldsymbol{w}} = \frac{1}{T-T_0}\sum_{t=T_0}^{T} \boldsymbol{w}^{(t)}$)

# SGD vs GD: Notes (1)

Why should we use SGD instead of GD in machine learning applications ?

Consider the ML ERM setting:
find $\boldsymbol{w}$ that minimizes $L_S(\boldsymbol{w})$, i.e., $f(\boldsymbol{w}) = L_S(\boldsymbol{w})$

Using GD:
- $\nabla f(\boldsymbol{w})$ depends on all the $m$ pairs $(\boldsymbol{x_i}, y_i) \in S$
- Need to process all the training set at each iteration
- Very long computation time if training set is large (as in real world ML problems)

Using SGD:
Need to pick $\boldsymbol{v_t}$ such that $E[\boldsymbol{v_t}|\boldsymbol{w}^{(t)}] = \nabla f(\boldsymbol{w}^{(t)}) = \nabla L_S(\boldsymbol{w}^{(t)})$
- pick a random $(\boldsymbol{x_i}, y_i) \in S$ $\Rightarrow \boldsymbol{v_t} = \nabla \ell(\boldsymbol{w}^{(t)}, (\boldsymbol{x}_i, y_i))$
- Satisfies the requirement
- Can be computed from just a single sample ($\longrightarrow$ much faster !!)

Same discussion apply to regularized losses and other risk minimization framework

* the notation refers to the simplified case of differentiable functions, see the book for a more rigorous discussion

# SGD vs GD: Notes (2)

- Much faster than GD: at each step only one sample is used for the computation
  - Specially for large training sets standard GD is slow
- Less stable trajectory
  - More "*noisy*" but could jump out of local minima
    - Advanced approaches to stabilize, e.g., momentum
  - Sometimes the final point is computed as average of a set of samples (as in the book) to account for fluctuations
    - Better to average only a set of final iterations
    - On book average of all iterations (not always smart choice)
  - Improvement to get a stable result: use an adaptive step size

# Gradient Descent: Variants

1. *Batch Gradient Descent* (standard GD): compute the gradient over the complete training set
2. *Mini-batch Gradient Descent*: compute the gradient over a small set of *k* samples
   - k: parameter, mini-batch size
   - Trade-off between the two "extreme" cases GD and SGD
   - Used to train deep neural networks
3. *Stochastic Gradient Descent (SGD)*: use a single sample to estimate the gradient

# SGD: Applications in ML

Use SGD to solve ML problems :

1. Risk minimization (ERM)
2. Regularized Loss minimization (RLM)
3. Support Vector Machines (SVM)
4. Neural Networks (in NN / deep learning lectures)

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

# SGD for Risk Minimization (1)

Stochastic Gradient Descent (SGD) for minimizing $L_D(\boldsymbol{w})$

params: Scalar $\eta > 0$ , integer $T > 0$

Init: $\boldsymbol{w}^{(1)} = \boldsymbol{0}$
for $t = 1,2, \dots, T$
        sample $z \sim D$
        pick $\boldsymbol{v_t} = \nabla \ell(\boldsymbol{w}^{(t)}, \mathrm{z})$
        update $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta \boldsymbol{v_t}$
output $\boldsymbol{w}^{(T)}$

- Minimize $L_D$ directly
- Find an *unbiased* estimate of the gradient of $L_D$
- Sample a single fresh sample and estimate the gradient with it
- Can be applied to RLM solving its target

# SGD Finds an Unbiased Estimate of the Gradient

SGD finds an unbiased estimate of the gradient of $L_D$:

1. Sample $z \sim D$: $\mathbf{v_t} = \nabla l(\boldsymbol{w}^{(t)}, z)$
2. $E[\boldsymbol{v}_t | \boldsymbol{w}^{(t)}] = E_{z\sim D}[\nabla l(\boldsymbol{w}^{(t)}, z)] = \nabla E_{z\sim D}[l(\boldsymbol{w}^{(t)}, z)] = \nabla \mathrm{L_D}(\boldsymbol{w}^{(t)})$

- The gradient of the loss function $l(\boldsymbol{w}^{(t)}, z)$ at $\boldsymbol{w}^{(t)}$ is an unbiased estimate of the gradient of the true risk $L_{\mathrm{D}}(\boldsymbol{w}^{(t)})$
- Proof based on linearity of the gradient operator
- Need to sample a new fresh sample at each iteration
  - But in many optimization schemes training samples are re-used

# SGD for Risk Minimization (2)

- Consider a convex $\rho$–Lipschitz-bounded learning problem with parameters $\rho, B$
- Then, for every $\epsilon > 0$, if we run the SGD method for minimizing $L_D(\boldsymbol{w})$ with a number of iterations (i.e., number of examples) $\mathrm{T} \geq \frac{B^2\rho^2}{\epsilon^2}$ and with $\eta = \sqrt{\frac{B^2}{\rho^2 T}}$, the output $\overline{\boldsymbol{w}}$ of SGD satisfies:

$$\mathbb{E}[L_D(\overline{\boldsymbol{w}})] \leq \min_{\boldsymbol{w}\in\mathcal{H}} L_D(\boldsymbol{w}) + \epsilon$$

# SGD for $\lambda$-strongly convex functions and RLM



Not Part of the course

- SGD for $\lambda$-strongly convex functions: a good strategy is to use an adaptive step size of value $\eta_t = \frac{1}{\lambda t}$

  Idea: smaller and smaller steps while training goes on

  - Details and theoretical bounds on the book, *not part of the course*

- Recall: RLM → The associated optimization problem can be written as

$$\min_{\boldsymbol{w}} \left( \frac{\lambda}{2} \|\boldsymbol{w}\|^2 + L_S(\boldsymbol{w}) \right)$$

- Define $f(\boldsymbol{w}) = \frac{\lambda}{2} \|\boldsymbol{w}\|^2 + L_S(\boldsymbol{w})$ : it is $2\frac{\lambda}{2} = \lambda$-strongly convex
  - Can apply adaptive learning rate with rate $\eta_t = \frac{1}{\lambda t}$

Not Part of the course

# SGD for RLM

- Recall: $f(\boldsymbol{w}) = \frac{\lambda}{2}||\boldsymbol{w}||^2 + L_S(\boldsymbol{w})$: it is $\lambda$-strongly convex, use $\eta_t = \frac{1}{\lambda t}$
- Update rule can be rewritten as

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \frac{1}{\lambda t}\left(\lambda \boldsymbol{w}^{(t)} + \boldsymbol{v}_t\right) = -\frac{1}{\lambda t}\sum_{i=1}^{t} \boldsymbol{v}_i$$

*Demonstration: see next slide*

- If loss is $\rho$-Lipschitz, after $T$ iterations we have that:

$$\mathbb{E}[f(\bar{\boldsymbol{w}})] - f(\boldsymbol{w}^*) \leq \frac{4\rho^2}{\lambda T}(1 + \log(T))$$

*Demonstration and details not part of the course*

Not Part of the course

INGEGNERIA DELL'INFORMAZIONE

# SGD for RLM (demonstration)

- Update rule can be rewritten as

$$
\begin{aligned}
\boldsymbol{w}^{(t+1)} &= \boldsymbol{w}^{(t)} - \frac{1}{\lambda t}\left(\lambda \boldsymbol{w}^{(t)} + \boldsymbol{v}_t\right) \\
&= \left(1 - \frac{1}{t}\right) \boldsymbol{w}^{(t)} - \frac{1}{\lambda t} \boldsymbol{v}_t = \left(\frac{t-1}{t}\right) \boldsymbol{w}^{(t)} - \frac{1}{\lambda t} \boldsymbol{v}_t \\
&= \frac{t-1}{t}\left(\frac{t-2}{t-1} \boldsymbol{w}^{(t-1)} - \frac{1}{\lambda (t-1)} \boldsymbol{v}_{t-1}\right) - \frac{1}{\lambda t} \boldsymbol{v}_t \\
&= \cdots \\
&= -\frac{1}{\lambda t} \sum_{i=1}^{t} \boldsymbol{v}_i
\end{aligned}
$$

*We'll use for SVM*

# SGD for Soft SVM (1)

- Hinge loss

$$f^{hinge}(\boldsymbol{w}) = \max\{0, 1 - y < \boldsymbol{w}, \boldsymbol{x} >\}$$

- (sub)gradient of $f^{hinge}$ at $\boldsymbol{w}$:

$$\boldsymbol{v}^{hinge} = \begin{cases} 0 & if\ 1 - y < \boldsymbol{w}, \boldsymbol{x} > \leq\ 0 \\ -y\boldsymbol{x} & if\ 1 - y < \boldsymbol{w}, \boldsymbol{x} > > 0 \end{cases}$$

- Update Rule (for the complete soft-SVM optimization)

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta \boldsymbol{v}^{(t)} \quad or \quad \boldsymbol{w}^{(t+1)} = -\frac{1}{\lambda t} \sum_{j=1}^{t} \boldsymbol{v}^{(t)}$$

  - *the first equation is standard SGD*
  - *the second is from the variant of SGD for λ-strongly convex functions*

# SGD for Soft SVM (2)

We want to solve

$$\min_{\mathbf{w}} \left( \frac{\lambda}{2} ||\mathbf{w}||^2 + \frac{1}{m} \sum_{i=1}^{m} \max\{0, 1 - y\langle \mathbf{w}, \mathbf{x}_i \rangle\} \right)$$

*Variant of SGD for RLM*

$$\boldsymbol{w}^{(t+1)} = -\frac{1}{\lambda t} \sum_{j=1}^{t} \boldsymbol{v}_j$$

**Note**: it's standard to add a $\frac{1}{2}$ in the regularization term to simplify some computations.

$$\theta^{(t)} = -\sum_{j=1}^{t-1} \boldsymbol{v}_j$$

Algorithm:

$\theta^{(1)} \leftarrow \mathbf{0}$ ;

**for** $t \leftarrow 1$ *to* $T$ **do**

  let $\mathbf{w}^{(t)} \leftarrow \frac{1}{\lambda t} \theta^{(t)}$;

  choose $i$ uniformly at random from $\{1, \ldots, m\}$;

  **if** $y_i \langle \mathbf{w}^{(t)}, \mathbf{x}_i \rangle < 1$ **then** $\theta^{(t+1)} \leftarrow \theta^{(t)} + y_i \mathbf{x}_i$;

  **else** $\theta^{(t+1)} \leftarrow \theta^{(t)}$;

**return** ~~$\bar{\mathbf{w}} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{w}^{(t)}$~~; *or* $\bar{\boldsymbol{w}} = \boldsymbol{w}^{(T)}$

Hinge loss: gradient is 0 if correctly classified and $-y_i x_i$ if error

Not Part of the course

# SGD for Soft SVM with Kernels

We want to solve:

$$\min_{\boldsymbol{w}}\left(\lambda\|\boldsymbol{w}\|^2 \quad +\frac{1}{m}\sum_{i=1}^{m}\max\{0,1-y_i<\boldsymbol{w},\psi(\boldsymbol{x}_i)>\}\right)\quad (*)$$

**SGD for Solving Soft-SVM with Kernels**

**Goal:** Solve Equation (*)
**parameter:** $T$
**Initialize:** $\boldsymbol{\beta}^{(1)} = \mathbf{0}$
**for** $t = 1, \ldots, T$
  Let $\boldsymbol{\alpha}^{(t)} = \frac{1}{\lambda t}\boldsymbol{\beta}^{(t)}$
  Choose $i$ uniformly at random from $[m]$
  For all $j \neq i$ set $\beta_j^{(t+1)} = \beta_j^{(t)}$
  If $(y_i \sum_{j=1}^{m} \alpha_j^{(t)} K(\mathbf{x}_j, \mathbf{x}_i) < 1)$
    Set $\beta_i^{(t+1)} = \beta_i^{(t)} + y_i$
  Else
    Set $\beta_i^{(t+1)} = \beta_i^{(t)}$
**Output:** $\bar{\mathbf{w}} = \sum_{j=1}^{m} \bar{\alpha}_j \psi(\mathbf{x}_j)$ where $\bar{\boldsymbol{\alpha}} = \frac{1}{T}\sum_{t=1}^{T}\boldsymbol{\alpha}^{(t)}$

$$\boldsymbol{\theta}^{(t)} = \sum_{j=1}^{m}\beta_j^{(t)}\psi(x_j)$$

$$\boldsymbol{w}^{(t)} = \sum_{j=1}^{m}\alpha_j^{(t)}\psi(x_j)$$

We want to solve *Standard Soft-SVM*

$$\min_{\mathbf{w}}\left(\frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{m}\sum_{i=1}^{m}\max\{0, 1-y\langle\mathbf{w},\mathbf{x}_i\rangle\}\right)$$

**Note**: it's standard to add a $\frac{1}{2}$ in the regularization term to simplify some computations.

Algorithm:
$\theta^{(1)} \leftarrow \mathbf{0}$ ;
**for** $t \leftarrow 1$ *to* $T$ **do**
  let $\mathbf{w}^{(t)} \leftarrow \frac{1}{\lambda t}\theta^{(t)}$;
  choose $i$ uniformly at random from $\{1,\ldots,m\}$;
  **if** $y_i\langle\mathbf{w}^{(t)},\mathbf{x}_i\rangle < 1$ **then** $\theta^{(t+1)} \leftarrow \theta^{(t)} + y_i\mathbf{x}_i$;
  **else** $\theta^{(t+1)} \leftarrow \theta^{(t)}$;
**return** $\bar{\mathbf{w}} = \frac{1}{T}\sum_{t=1}^{T}\mathbf{w}^{(t)}$;

*compare with previous algorithm*

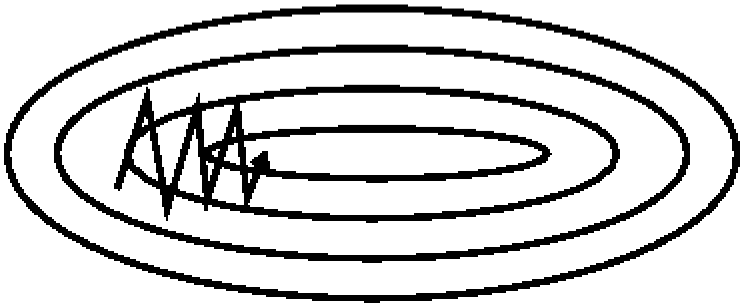
- $\mathbf{w} = \sum_{i=1}^{m}\alpha_i\psi(\mathbf{x}_i)$ from Representer theorem: maintain $\alpha$ instead of w
- ***Theorem:*** the new procedure and the old one applied on the feature space (i.e., replacing $\mathbf{x}$ with $\psi(\mathbf{x})$) lead to the same results *(no demonstration)*

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

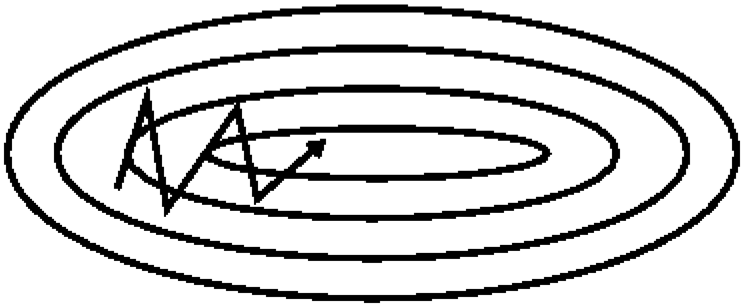
# SGD: Issues

- The selection of the learning rate $\eta$ is a critical point
  - If $\eta$ too small: the optimization is stable but the convergence can be very slow
  - If $\eta$ too large: the convergence is fast but the optimization can be very unstable
- Simple solution: use adaptive learning rates, e.g.,
  - Progressively reducing the learning rate according to a pre-defined schedule
  - *Example* : for RLM optimization with SGD $\eta_t = \frac{1}{\lambda t}$ is used
  - However these approaches requires rules and thresholds to be defined in advance and thus are difficult to adapt to different problems
- Additionally, the same learning rate applies to all parameter updates
  - The various parameters have different behaviors and the learning rate could be too fast for some and too slow for others
  - Sometimes better not to update all parameters to the same extent, but perform a larger update for some and smaller for others

# Momentum



SGD without momentum



SGD with momentum

- SGD has troubles (i.e., it oscillates) in areas where the surface curves much more steeply in one dimension than in another (which are common around local optima)
- *Momentum:* the update is the linear combination of previous gradient and new one
  - The momentum parameter $\gamma$ is usually set to 0.9 or a similar value

- $\boldsymbol{v}^{(t)} = \gamma\, \boldsymbol{v}^{(t-1)} + (1-\gamma)\, \nabla L(\boldsymbol{w}^{(t)})$
- $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta \boldsymbol{v}^{(t)}$

- It helps accelerate SGD in the relevant direction and dampens oscillations

Using momentum is like pushing a ball down a hill. The ball accumulates momentum as it rolls downhill, becoming faster and faster on the way. The same thing happens to our parameter updates: the momentum term increases for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions. As a result, we gain faster convergence and reduced oscillation

# Advanced SGD schemes

- Adagrad adapts the learning rate for each parameter independently
  - It performs smaller updates (i.e. low learning rates) for parameters associated with frequently occurring features
  - It performs larger updates (i.e. high learning rates) for parameters associated with infrequent features
- Adadelta (improved version of ADAgrad)
- RMSprop (improved version of ADAgrad)
- Adam (Adaptive Moment Estimation)
  - It also computes adaptive learning rates for each parameter
  - It combines ideas from Adagrad and momentum
  - Whereas momentum can be seen as a ball running down a slope, Adam behaves like a heavy ball with friction, which thus prefers flat minima in the error surface

*Details not part of the course*