More on Gradient Descent

Yingzhen Li

Department of Computing
Imperial College London

@liyzhen2
yingzhen.li@imperial.ac.uk

October 29, 2021

Recap

Algorithm: Gradient Descent

Define **starting point** $\boldsymbol{\theta}_0$, sequence of **step sizes** γ_t, set $t \leftarrow 0$.

1. Set $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma_t \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t)$, $t \leftarrow t + 1$
2. Repeat 1 until stopping criterion.

Recap

Algorithm: Gradient Descent **for linear regression**

Define **starting point** $\boldsymbol{\theta}_0$, sequence of **step sizes** γ_t, set $t \leftarrow 0$.

1. Set $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma_t \frac{1}{\sigma^2}\mathbf{X}^\top(\mathbf{X}\boldsymbol{\theta}_t - \mathbf{y})$, $t \leftarrow t+1$

   $$\Leftrightarrow \quad \boldsymbol{\theta}_{t+1} = (\mathbf{I} - \tfrac{\gamma}{\sigma^2}\mathbf{X}^\top\mathbf{X})\boldsymbol{\theta}_t + \tfrac{\gamma}{\sigma^2}\mathbf{X}^\top\mathbf{y} \quad \text{if } \gamma_t = \gamma$$

2. Repeat 1 until stopping criterion.

Recap

Analysing convergence of GD with constant step sizes:

$$\boldsymbol{\theta}_{t+1} = (\mathbf{I} - \frac{\gamma}{\sigma^2}\mathbf{X}^\top\mathbf{X})\boldsymbol{\theta}_t + \frac{\gamma}{\sigma^2}\mathbf{X}^\top\mathbf{y} \quad \text{if } \gamma_t = \gamma$$

Key ideas:

- Need to check if $||\boldsymbol{\theta}_t - \boldsymbol{\theta}^*||_2^2 \to 0$ when $t \to \infty$
- $\boldsymbol{\theta}_t - \boldsymbol{\theta}^* = \mathbf{A}^t(\boldsymbol{\theta}_0 - \boldsymbol{\theta}^*)$ where $\mathbf{A} = (\mathbf{I} - \frac{\gamma}{\sigma^2}\mathbf{X}^\top\mathbf{X})$
- So we need to understand how matrix multiplications "stretch/shrink" vectors $\leftarrow$ Eigen-decomposition & SVD
- Convergence of GD depends on $\lambda_{max}((\mathbf{I} - \frac{\gamma}{\sigma^2}\mathbf{X}^\top\mathbf{X})^2)$
  $\Rightarrow$ need to set γ smaller than some "safe threshold"

Recap

Algorithm: Gradient Descent

Define **starting point** $\boldsymbol{\theta}_0$, sequence of **step sizes** γ_t, set $t \leftarrow 0$.

1. Set $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma_t \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t)$, $t \leftarrow t + 1$
2. Repeat 1 until stopping criterion.

On choosing step size:

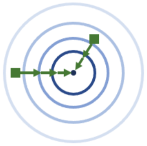
- too small: slow convergence
- too large: divergence
- just right: depends on problem (often: trial and error)

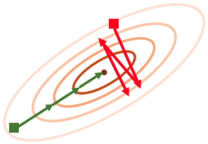
Recap

Is my choice of step size robust (to different initialisations)?

- Depending on the **condition number**:

$$\kappa(\mathbf{X}^\top\mathbf{X}) := \frac{\lambda_{max}(\mathbf{X}^\top\mathbf{X})}{\lambda_{min}(\mathbf{X}^\top\mathbf{X})} = \sqrt{\frac{\max_{\boldsymbol{q}} R(\mathbf{X}^\top\mathbf{X}, \boldsymbol{q})}{\min_{\boldsymbol{q}} R(\mathbf{X}^\top\mathbf{X}, \boldsymbol{q})}}$$


well conditioned
$\kappa(X^\top X) \approx 1$


ill conditioned
$\kappa(X^\top X) \gg 1$

- Need careful choice of step-sizes if the loss is "very stretched"

Gradient descent with pre-conditioning

Algorithm: Gradient Descent **with Pre-conditioning**
Define **starting point** $\boldsymbol{\theta}_0$, sequence of **step sizes** γ_t,
Define $\Delta\boldsymbol{\theta}_t = \mathbf{0}$, set $t \leftarrow 0$.

1. **Select a pre-conditioner** $\mathbf{P}_t$
2. Set $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma_t \mathbf{P}_t^{-1} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t)$
3. Set $t \leftarrow t + 1$
4. Repeat 1 - 3 until stopping criterion.

Gradient descent with pre-conditioning

Linear regression example:
Assume constant step-sizes $\gamma_t = \gamma$ and **fixed pre-conditioner $\mathbf{P}$**:

1. Set $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma \frac{1}{\sigma^2} \mathbf{P}^{-1} \mathbf{X}^\top (\mathbf{X}\boldsymbol{\theta}_t - \mathbf{y})$
2. Set $t \leftarrow t + 1$
3. Repeat 1 - 2 until stopping criterion.

How to choose γ and $\mathbf{P}$?

Gradient descent with pre-conditioning

Let's derive the iterative updates again:

$$\boldsymbol{\theta}_{t+1} = (\mathbf{I} - \frac{\gamma}{\sigma^2}\mathbf{P}^{-1}\mathbf{X}^\top\mathbf{X})\boldsymbol{\theta}_t + \frac{\gamma}{\sigma^2}\mathbf{P}^{-1}\mathbf{X}^\top\mathbf{y}$$

Gradient descent with pre-conditioning

Let's derive the iterative updates again:

$$\boldsymbol{\theta}_{t+1} = (\mathbf{I} - \frac{\gamma}{\sigma^2}\mathbf{P}^{-1}\mathbf{X}^\top\mathbf{X})\boldsymbol{\theta}_t + \frac{\gamma}{\sigma^2}\mathbf{P}^{-1}\mathbf{X}^\top\mathbf{y}$$

$$\Rightarrow \quad \boldsymbol{\theta}_t = (\mathbf{I} - \frac{\gamma}{\sigma^2}\mathbf{P}^{-1}\mathbf{X}^\top\mathbf{X})^t(\boldsymbol{\theta}_0 - \boldsymbol{\theta}^*) + \boldsymbol{\theta}^*$$

Gradient descent with pre-conditioning

Let's derive the iterative updates again:

$$\boldsymbol{\theta}_{t+1} = (\mathbf{I} - \frac{\gamma}{\sigma^2}\mathbf{P}^{-1}\mathbf{X}^\top\mathbf{X})\boldsymbol{\theta}_t + \frac{\gamma}{\sigma^2}\mathbf{P}^{-1}\mathbf{X}^\top\mathbf{y}$$

$$\Rightarrow \quad \boldsymbol{\theta}_t = (\mathbf{I} - \frac{\gamma}{\sigma^2}\mathbf{P}^{-1}\mathbf{X}^\top\mathbf{X})^t(\boldsymbol{\theta}_0 - \boldsymbol{\theta}^*) + \boldsymbol{\theta}^*$$

This means we need to look into eigenvalues of $(\mathbf{I} - \frac{\gamma}{\sigma^2}\mathbf{P}^{-1}\mathbf{X}^\top\mathbf{X})^2$

1. $\lambda_{max} < 1$: always converge
2. $\lambda_{min} \geqslant 1$: always diverge
3. $\lambda_{min} < 1$ but $\lambda_{max} \geqslant 1$: depends on initialisation of $\boldsymbol{\theta}_0$

Gradient descent with pre-conditioning

To ensure convergence at **any** initialisation: $\gamma < 2\sigma^2 / \lambda_{max}(\mathbf{P}^{-1}\mathbf{X}^\top \mathbf{X})$
If you want to test your luck: choose $\gamma \in [\frac{2\sigma^2}{\lambda_{max}(\mathbf{P}^{-1}\mathbf{X}^\top\mathbf{X})}, \frac{2\sigma^2}{\lambda_{min}(\mathbf{P}^{-1}\mathbf{X}^\top\mathbf{X})})$
Is my choice of γ robust to initialisation of $\boldsymbol{\theta}_0$?

Gradient descent with pre-conditioning

To ensure convergence at **any** initialisation: $\gamma < 2\sigma^2 / \lambda_{max}(\mathbf{P}^{-1}\mathbf{X}^\top \mathbf{X})$
If you want to test your luck: choose $\gamma \in [\frac{2\sigma^2}{\lambda_{max}(\mathbf{P}^{-1}\mathbf{X}^\top \mathbf{X})}, \frac{2\sigma^2}{\lambda_{min}(\mathbf{P}^{-1}\mathbf{X}^\top \mathbf{X})})$
Is my choice of γ robust to initialisation of $\boldsymbol{\theta}_0$?

- Depending on the **condition number**:

$$\kappa(\mathbf{P}^{-1}\mathbf{X}^\top \mathbf{X}) := \frac{\lambda_{max}(\mathbf{P}^{-1}\mathbf{X}^\top \mathbf{X})}{\lambda_{min}(\mathbf{P}^{-1}\mathbf{X}^\top \mathbf{X})} = \sqrt{\frac{\max_{\boldsymbol{q}} R(\mathbf{P}^{-1}\mathbf{X}^\top \mathbf{X}, \boldsymbol{q})}{\min_{\boldsymbol{q}} R(\mathbf{P}^{-1}\mathbf{X}^\top \mathbf{X}, \boldsymbol{q})}}$$

Gradient descent with pre-conditioning

To ensure convergence at **any** initialisation: $\gamma < 2\sigma^2/\lambda_{max}(\mathbf{P}^{-1}\mathbf{X}^\top\mathbf{X})$

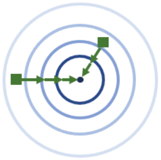
If you want to test your luck: choose $\gamma \in [\frac{2\sigma^2}{\lambda_{max}(\mathbf{P}^{-1}\mathbf{X}^\top\mathbf{X})}, \frac{2\sigma^2}{\lambda_{min}(\mathbf{P}^{-1}\mathbf{X}^\top\mathbf{X})})$

Is my choice of γ robust to initialisation of $\boldsymbol{\theta}_0$?

- Depending on the **condition number**:

$$\kappa(\mathbf{P}^{-1}\mathbf{X}^\top\mathbf{X}) := \frac{\lambda_{max}(\mathbf{P}^{-1}\mathbf{X}^\top\mathbf{X})}{\lambda_{min}(\mathbf{P}^{-1}\mathbf{X}^\top\mathbf{X})} = \sqrt{\frac{\max_{\boldsymbol{q}} R(\mathbf{P}^{-1}\mathbf{X}^\top\mathbf{X}, \boldsymbol{q})}{\min_{\boldsymbol{q}} R(\mathbf{P}^{-1}\mathbf{X}^\top\mathbf{X}, \boldsymbol{q})}}$$

- Ideally, prefer well-conditioned optimisation: choose $\mathbf{P} \propto \mathbf{X}^\top\mathbf{X}$
- In practice: want to make $\kappa(\mathbf{P}^{-1}\mathbf{X}^\top\mathbf{X}) \approx 1$ while $\mathbf{P}^{-1}$ is easy to compute



well conditioned
$\kappa(P^{-1}X^\top X) \approx 1$

Gradient descent with pre-conditioning

Constructing "cheap but useful" pre-conditioner $\mathbf{P}_t$:

- Use diagonal/low-rank matrices for $\mathbf{P}_t$

Gradient descent with pre-conditioning

Constructing “cheap but useful” pre-conditioner $\mathbf{P}_t$:

- Use diagonal/low-rank matrices for $\mathbf{P}_t$
- Having prior knowledge on $\nabla^2 L(\boldsymbol{\theta}_t)$ would be useful
  - For linear regression, $\nabla^2 L(\boldsymbol{\theta}_t) \propto \mathbf{X}^\top \mathbf{X}$

Gradient descent with pre-conditioning

Constructing “cheap but useful” pre-conditioner $\mathbf{P}_t$:

- Use diagonal/low-rank matrices for $\mathbf{P}_t$
- Having prior knowledge on $\nabla^2 L(\boldsymbol{\theta}_t)$ would be useful
  - For linear regression, $\nabla^2 L(\boldsymbol{\theta}_t) \propto \mathbf{X}^\top \mathbf{X}$
- $\nabla^2 L(\boldsymbol{\theta}_t)$ might not be available nor a constant
  - Approximated by statistics of gradients along the update trajectory
  - Many adaptive learning rate methods (e.g. Adam) do this!

Gradient descent with momentum

Algorithm: Gradient Descent **with Momentum**
Define **starting point** $\boldsymbol{\theta}_0$, sequence of **step sizes** γ_t,
Define $\Delta\boldsymbol{\theta}_t = \mathbf{0}$**, momentum step-size** α, set $t \leftarrow 0$.

1. Set $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma_t \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t) + \alpha \Delta\boldsymbol{\theta}_t$
2. Set $\Delta\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t = \alpha \Delta\boldsymbol{\theta}_t - \gamma_t \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t)$
3. Set $t \leftarrow t + 1$
4. Repeat 1 - 3 until stopping criterion.

Gradient descent with momentum

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma_t \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t) + \alpha \Delta\boldsymbol{\theta}_t$$

$$\Delta\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t$$

The Key idea of using momentum $\Delta\boldsymbol{\theta}_t = \boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}$:

Making the current iteration's update closer to the previous iterations.

Gradient descent with momentum

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma_t \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t) + \alpha \Delta\boldsymbol{\theta}_t$$

$$\Delta\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t$$

The Key idea of using momentum $\Delta\boldsymbol{\theta}_t = \boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}$:
Making the current iteration's update closer to the previous iterations.

- Speed up in "flat regions" (i.e. $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t)$ has small magnitude)

Gradient descent with momentum

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma_t \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t) + \alpha \Delta \boldsymbol{\theta}_t$$

$$\Delta \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t$$

The Key idea of using momentum $\Delta \boldsymbol{\theta}_t = \boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}$:
Making the current iteration's update closer to the previous iterations.

- Speed up in "flat regions" (i.e. $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t)$ has small magnitude)
- Alleviate oscillations (when $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t)$ points to a different direction)

Gradient descent with momentum

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma_t \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t) + \alpha \Delta \boldsymbol{\theta}_t$$

$$\Delta \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t$$

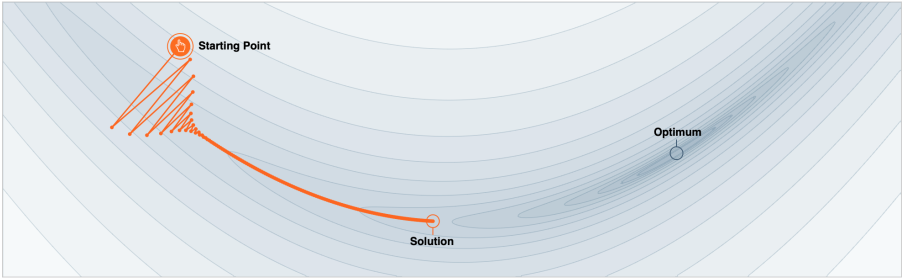
The Key idea of using momentum $\Delta \boldsymbol{\theta}_t = \boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}$:
Making the current iteration's update closer to the previous iterations.

- Speed up in "flat regions" (i.e. $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t)$ has small magnitude)
- Alleviate oscillations (when $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t)$ points to a different direction)
- Smooth out gradients if using inexact gradients

Momentum

Define **starting point** $\boldsymbol{\theta}_0$, sequence of **step sizes** γ_t, set $t \leftarrow 0$.

1. Set $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma_t \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t) + \alpha(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1})$
   $t \leftarrow t + 1$
2. Repeat 1 until stopping criterion.

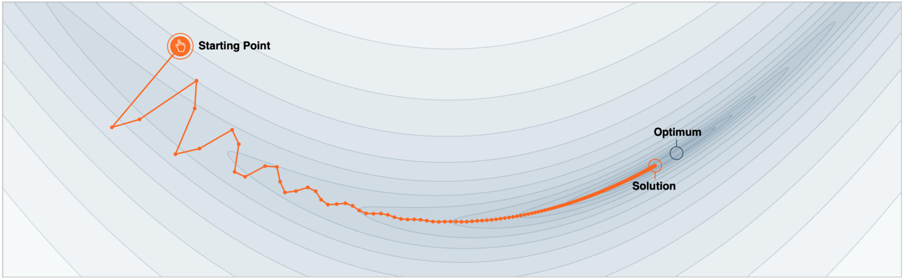


Without momentum

Momentum

Define **starting point** $\boldsymbol{\theta}_0$, sequence of **step sizes** γ_t, set $t \leftarrow 0$.

1. Set $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma_t \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t) + \alpha(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1})$
   $t \leftarrow t + 1$
2. Repeat 1 until stopping criterion.



With momentum

Line search

Can we automatically adapt γ_t so we can **guarantee improvement**?

Line search

Can we automatically adapt γ_t so we can **guarantee improvement**?
Algorithm: Line Search

Define **starting point** $\boldsymbol{\theta}_0$, sset $t \leftarrow 0$.

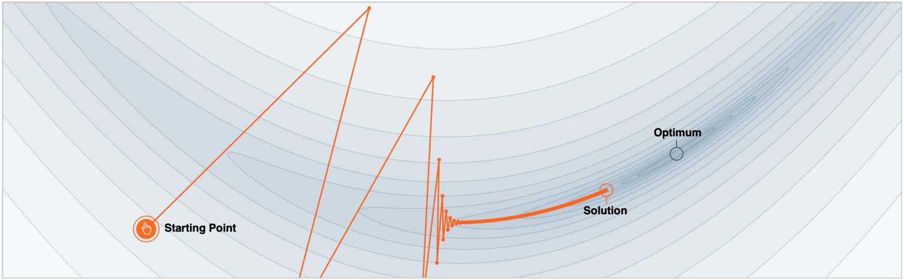
1. Compute gradient $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t)$
2. Search $\gamma_t \in (\gamma_{min}, \gamma_{max})$ to **minimise** $L(\boldsymbol{\theta}_t - \gamma \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t))$ **w.r.t.** γ
3. Set $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma_t \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t)$, $t \leftarrow t + 1$
4. Repeat 1-3 until stopping criterion.

Line search

Can we automatically adapt γ_t so we can **guarantee improvement**?
Algorithm: Line Search

Define **starting point** $\boldsymbol{\theta}_0$, sset $t \leftarrow 0$.

1. Compute gradient $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t)$
2. Search $\gamma_t \in (\gamma_{min}, \gamma_{max})$ to **minimise** $L(\boldsymbol{\theta}_t - \gamma \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t))$ **w.r.t.** γ
3. Set $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma_t \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t)$, $t \leftarrow t + 1$
4. Repeat 1-3 until stopping criterion.



Line search

Algorithms for the search step:
Approximately minimise $L(\boldsymbol{\theta}_t - \gamma \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t))$ w.r.t. $\gamma \in (\gamma_{min}, \gamma_{max})$

- Backtracking line search: shrinking step sizes

Line search

Algorithms for the search step:
Approximately minimise $L(\boldsymbol{\theta}_t - \gamma \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t))$ w.r.t. $\gamma \in (\gamma_{min}, \gamma_{max})$

- Backtracking line search: shrinking step sizes
  - Set a "decreasing schedule" $\gamma_{max} = \gamma^{(1)} > \gamma^{(2)} > ... > \gamma^{(\tau)} = \gamma_{min}$

Line search

Algorithms for the search step:
Approximately minimise $L(\boldsymbol{\theta}_t - \gamma \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t))$ w.r.t. $\gamma \in (\gamma_{min}, \gamma_{max})$

- Backtracking line search: shrinking step sizes
  - Set a "decreasing schedule" $\gamma_{max} = \gamma^{(1)} > \gamma^{(2)} > ... > \gamma^{(\tau)} = \gamma_{min}$
  - Try out these step sizes until satisfying an acceptance criterion e.g. $L(\boldsymbol{\theta}_t) - L(\boldsymbol{\theta}_t - \gamma^{(k)} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t))$ is "big enough"
  - Set $\gamma_t \leftarrow \gamma^{(k)}$

Line search

Algorithms for the search step:
Approximately minimise $L(\boldsymbol{\theta}_t - \gamma \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t))$ w.r.t. $\gamma \in (\gamma_{min}, \gamma_{max})$

- Backtracking line search: shrinking step sizes
  - Set a "decreasing schedule" $\gamma_{max} = \gamma^{(1)} > \gamma^{(2)} > ... > \gamma^{(\tau)} = \gamma_{min}$
  - Try out these step sizes until satisfying an acceptance criterion e.g. $L(\boldsymbol{\theta}_t) - L(\boldsymbol{\theta}_t - \gamma^{(k)} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t))$ is "big enough"
  - Set $\gamma_t \leftarrow \gamma^{(k)}$
- Other methods, e.g. checking the Wolfe conditions

Line search

Algorithms for the search step:
Approximately minimise $L(\boldsymbol{\theta}_t - \gamma \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t))$ w.r.t. $\gamma \in (\gamma_{min}, \gamma_{max})$

- Backtracking line search: shrinking step sizes
  - Set a "decreasing schedule" $\gamma_{max} = \gamma^{(1)} > \gamma^{(2)} > ... > \gamma^{(\tau)} = \gamma_{min}$
  - Try out these step sizes until satisfying an acceptance criterion e.g. $L(\boldsymbol{\theta}_t) - L(\boldsymbol{\theta}_t - \gamma^{(k)} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t))$ is "big enough"
  - Set $\gamma_t \leftarrow \gamma^{(k)}$
- Other methods, e.g. checking the Wolfe conditions

Used by e.g. nonlinear Conjugate Gradient or BFGS.
Converges **very** quickly if $\dim(\boldsymbol{\theta})$ is small.

Stochastic gradient descent

Remember objective function gradient:

$$\mathbf{g}_t = \nabla_{\boldsymbol{\theta}_t} L(\boldsymbol{\theta}_t) = \nabla_{\boldsymbol{\theta}_t} \sum_{n=1}^{N} (f(\boldsymbol{x}_n; \boldsymbol{\theta}_t) - y_n)^2$$

Stochastic gradient descent

Remember objective function gradient:

$$\mathbf{g}_t = \nabla_{\boldsymbol{\theta}_t} L(\boldsymbol{\theta}_t) = \sum_{n=1}^{N} \nabla_{\boldsymbol{\theta}_t} L_n(\boldsymbol{\theta}_t)$$

Stochastic gradient descent

Remember objective function gradient:

$$\mathbf{g}_t = \nabla_{\boldsymbol{\theta}_t} L(\boldsymbol{\theta}_t) = \sum_{n=1}^{N} \nabla_{\boldsymbol{\theta}_t} L_n(\boldsymbol{\theta}_t)$$

- In big data applications N could be very large, e.g. $N > 10^9$.

Stochastic gradient descent

Remember objective function gradient:

$$\mathbf{g}_t = \nabla_{\boldsymbol{\theta}_t} L(\boldsymbol{\theta}_t) = \sum_{n=1}^{N} \nabla_{\boldsymbol{\theta}_t} L_n(\boldsymbol{\theta}_t)$$

- In big data applications N could be very large, e.g. $N > 10^9$.
- Running (full batch) gradient descent can be very expensive!
  - Need to compute $L_n(\boldsymbol{\theta}_t)$ for each data point
  - Need to store (intermediate results of) every $\nabla_{\boldsymbol{\theta}_t} L_n(\boldsymbol{\theta}_t)$
  - Doing all of these only for a single update step of $\boldsymbol{\theta}$

Can we find a **computationally cheaper** but still working solution?

Stochastic gradient descent

Remember objective function gradient:

$$\mathbf{g}_t = \nabla_{\boldsymbol{\theta}_t} L(\boldsymbol{\theta}_t) = \sum_{n=1}^{N} \nabla_{\boldsymbol{\theta}_t} L_n(\boldsymbol{\theta}_t)$$

- In big data applications N could be very large, e.g. $N > 10^9$.
- **Stochastic** GD: compute a **stochastic estimator** for the sum using a **random subset**, i.e. with $M < N$ terms

$$\hat{\mathbf{g}}_t = \frac{N}{M} \sum_{m \in \mathcal{S}} \nabla_{\boldsymbol{\theta}_t} L_m(\boldsymbol{\theta}_t) \tag{1}$$

Stochastic gradient descent

Remember objective function gradient:

$$\mathbf{g}_t = \nabla_{\boldsymbol{\theta}_t} L(\boldsymbol{\theta}_t) = \sum_{n=1}^{N} \nabla_{\boldsymbol{\theta}_t} L_n(\boldsymbol{\theta}_t)$$

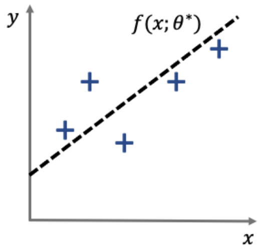
- In big data applications N could be very large, e.g. $N > 10^9$.
- **Stochastic** GD: compute a **stochastic estimator** for the sum using a **random subset**, i.e. with $M < N$ terms

$$\hat{\mathbf{g}}_t = \frac{N}{M} \sum_{m \in \mathcal{S}} \nabla_{\boldsymbol{\theta}_t} L_m(\boldsymbol{\theta}_t) \qquad (1)$$

- Gradient estimator $\hat{\mathbf{g}}_t$ is now **random**!

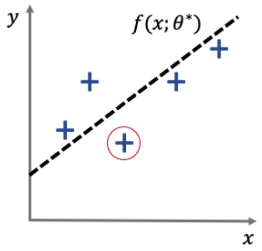
Does optimisation still work?

Stochastic gradient descent – random gradients



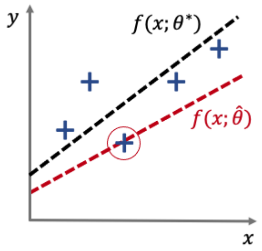
- Full batch GD will converge to $\boldsymbol{\theta}_t \rightarrow \boldsymbol{\theta}^*$

Stochastic gradient descent – random gradients



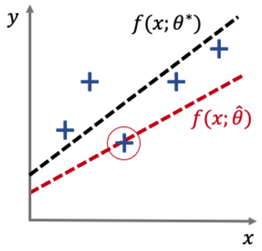
- Full batch GD will converge to $\boldsymbol{\theta}_t \to \boldsymbol{\theta}^*$

Stochastic gradient descent – random gradients



- Full batch GD will converge to $\boldsymbol{\theta}_t \rightarrow \boldsymbol{\theta}^*$
- GD based on a single datapoint: if we select the “wrong” points, the gradient can point in the **wrong direction**!

Stochastic gradient descent – random gradients



- Full batch GD will converge to $\boldsymbol{\theta}_t \to \boldsymbol{\theta}^*$
- GD based on a single datapoint: if we select the “wrong” points, the gradient can point in the **wrong direction**!
- If we don’t randomise, then we can keep on picking the same “wrong” point.

Stochastic gradient descent – convergence

- We choose our subsets e.g. through sampling with replacement:

$$P(\mathcal{S} = \{m_1, m_2, \ldots, m_M\}) = \frac{1}{N} \cdot \frac{1}{N} \cdots = N^{-M}$$

Stochastic gradient descent – convergence

- We choose our subsets e.g. through sampling with replacement:

$$P(\mathcal{S} = \{m_1, m_2, \ldots, m_M\}) = \frac{1}{N} \cdot \frac{1}{N} \cdots = N^{-M}$$

- So our estimator is **unbiased**, i.e.

$$\mathbb{E}_{\mathcal{S}}[\hat{\mathbf{g}}_t] = \mathbf{g}_t$$

Stochastic gradient descent – convergence

- We choose our subsets e.g. through sampling with replacement:

$$P(\mathcal{S} = \{m_1, m_2, \ldots, m_M\}) = \frac{1}{N} \cdot \frac{1}{N} \cdots = N^{-M}$$

- So our estimator is **unbiased**, i.e.

$$\mathbb{E}_{\mathcal{S}}[\hat{\mathbf{g}}_t] = \mathbf{g}_t$$

$$\Leftrightarrow \quad \mathbb{E}_{\mathcal{S}}\left[\frac{N}{M}\sum_{m\in\mathcal{S}} \nabla_{\boldsymbol{\theta}_t} L_m(\boldsymbol{\theta}_t)\right] = \sum_{n=1}^{N} \nabla_{\boldsymbol{\theta}_t} L_n(\boldsymbol{\theta}_t)$$

Stochastic gradient descent – convergence

Use **unbiased** gradient estimate:

$$\mathbb{E}_{\mathcal{S}}\left[\frac{N}{M}\sum_{m\in\mathcal{S}}\nabla_{\boldsymbol{\theta}_t}L_m(\boldsymbol{\theta}_t)\right]=\sum_{n=1}^{N}\nabla_{\boldsymbol{\theta}_t}L_n(\boldsymbol{\theta}_t)$$

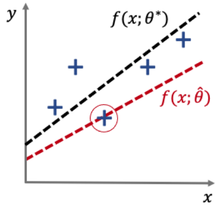
Stochastic gradient descent – convergence

Use **unbiased** gradient estimate:

$$\mathbb{E}_{\mathcal{S}}\left[\frac{N}{M}\sum_{m\in\mathcal{S}}\nabla_{\boldsymbol{\theta}_t}L_m(\boldsymbol{\theta}_t)\right]=\sum_{n=1}^{N}\nabla_{\boldsymbol{\theta}_t}L_n(\boldsymbol{\theta}_t)$$

- Next, if we choose the step sizes carefully:

$$\sum_{t=1}^{\infty}\gamma_t=\infty \qquad \sum_{t=1}^{\infty}\gamma_t^2<\infty$$



- Then SGD will converge!
- See [Robbins & Monro, 1951]

Stochastic gradient descent

Choosing batch size M:

- Should be reasonably small (so can be computed fast)
- Should not be too small though:

$$P(\mathcal{S} = \{m_1, m_2, \dots, m_M\}) = \frac{1}{N} \cdot \frac{1}{N} \cdots = N^{-M},$$

$$\mathbb{V}[\hat{\mathbf{g}}_t] = \mathbb{V}_{\mathcal{S}} \left[\frac{N}{M} \sum_{m \in \mathcal{S}} \nabla_{\boldsymbol{\theta}_t} L_m(\boldsymbol{\theta}_t) \right]$$

Stochastic gradient descent

Choosing batch size M:

- Should be reasonably small (so can be computed fast)
- Should not be too small though:

$$P(\mathcal{S} = \{m_1, m_2, \dots, m_M\}) = \frac{1}{N} \cdot \frac{1}{N} \cdots = N^{-M},$$

$$\begin{aligned}\mathbb{V}[\hat{\mathbf{g}}_t] &= \mathbb{V}_{\mathcal{S}}\left[\frac{N}{M}\sum_{m\in\mathcal{S}} \nabla_{\boldsymbol{\theta}_t} L_m(\boldsymbol{\theta}_t)\right] \\ &= \frac{N^2}{M}\mathbb{V}_{m\sim\text{Uniform}(1,\dots,N)}\left[\nabla_{\boldsymbol{\theta}_t} L_m(\boldsymbol{\theta}_t)\right]\end{aligned}$$

Stochastic gradient descent

Choosing batch size M:

- Should be reasonably small (so can be computed fast)
- Should not be too small though:

$$P(\mathcal{S} = \{m_1, m_2, \dots, m_M\}) = \frac{1}{N} \cdot \frac{1}{N} \cdots = N^{-M},$$

$$\begin{aligned} \mathbb{V}[\hat{\mathbf{g}}_t] &= \mathbb{V}_{\mathcal{S}}\left[\frac{N}{M} \sum_{m \in \mathcal{S}} \nabla_{\boldsymbol{\theta}_t} L_m(\boldsymbol{\theta}_t)\right] \\ &= \frac{N^2}{M} \mathbb{V}_{m \sim \text{Uniform}(1,\dots,N)}\left[\nabla_{\boldsymbol{\theta}_t} L_m(\boldsymbol{\theta}_t)\right] \end{aligned}$$

- Advanced SGD approaches use variance reduction techniques (allowing usage of small M but still achieve small variance)

Summary

This week we discussed **gradient descent**:

- Gradient descent for linear regression
- Convergence analysis (based on eigen decomposition results)
- Choosing step sizes
- GD with advanced techniques
  - Pre-conditioning
  - Momentum
  - Line search
- Stochastic gradient descent

Summary

This week we discussed **gradient descent**:

- Gradient descent for linear regression
- Convergence analysis (based on eigen decomposition results)
- Choosing step sizes
- GD with advanced techniques
  - Pre-conditioning
  - Momentum
  - Line search
- Stochastic gradient descent

A popular optimiser for deep learning "Adam":
SGD + momentum + pre-conditioning