Gradient Descent

Chris Cornwell

Mar 25, 2025

Outline

Gradient Descent

The general procedure

Have a function $\mathcal{L} : \mathbb{R}^p \to \mathbb{R}$ (our case, a loss function with $\Omega = \mathbb{R}^p$).
Want to approximate minimizer in $\mathbb{R}^p$ for $\mathcal{L}$.

[1] In these slides, η_t will be constant in t; however, in many scenarios η_t decreases as t increases.

The general procedure

Have a function $\mathcal{L} : \mathbb{R}^p \to \mathbb{R}$ (our case, a loss function with $\Omega = \mathbb{R}^p$). Want to approximate minimizer in $\mathbb{R}^p$ for $\mathcal{L}$.

- Select (any) initial $\omega^{(0)} \in \mathbb{R}^p$ and a learning rate $\eta_t > 0$ for each step $t \geq 1$.[1]

[1] In these slides, η_t will be constant in t; however, in many scenarios η_t decreases as t increases.

The general procedure

Have a function $\mathcal{L} : \mathbb{R}^p \to \mathbb{R}$ (our case, a loss function with $\Omega = \mathbb{R}^p$). Want to approximate minimizer in $\mathbb{R}^p$ for $\mathcal{L}$.

- Select (any) initial $\omega^{(0)} \in \mathbb{R}^p$ and a learning rate $\eta_t > 0$ for each step $t \geq 1$.[1]
- Next, for $t \geq 1$, iteratively assign

$$\omega^{(t)} = \omega^{(t-1)} - \eta_t \nabla \mathcal{L}(\omega^{(t-1)}).$$

[1] In these slides, η_t will be constant in t; however, in many scenarios η_t decreases as t increases.

The general procedure

Have a function $\mathcal{L} : \mathbb{R}^p \to \mathbb{R}$ (our case, a loss function with $\Omega = \mathbb{R}^p$). Want to approximate minimizer in $\mathbb{R}^p$ for $\mathcal{L}$.

- Select (any) initial $\omega^{(0)} \in \mathbb{R}^p$ and a learning rate $\eta_t > 0$ for each step $t \geq 1$.[1]
- Next, for $t \geq 1$, iteratively assign

$$\omega^{(t)} = \omega^{(t-1)} - \eta_t \nabla \mathcal{L}(\omega^{(t-1)}).$$

- Iterate until some **stopping condition** is met (say it happens when $t = T$). The approximate minimizer is $\omega^{(T)}$.

[1] In these slides, η_t will be constant in t; however, in many scenarios η_t decreases as t increases.

The general procedure

Have a function $\mathcal{L} : \mathbb{R}^p \to \mathbb{R}$ (our case, a loss function with $\Omega = \mathbb{R}^p$).
Want to approximate minimizer in $\mathbb{R}^p$ for $\mathcal{L}$.

- Select (any) initial $\omega^{(0)} \in \mathbb{R}^p$ and a learning rate $\eta_t > 0$ for each step $t \geq 1$.[1]
- Next, for $t \geq 1$, iteratively assign

$$\omega^{(t)} = \omega^{(t-1)} - \eta_t \nabla \mathcal{L}(\omega^{(t-1)}).$$

- Iterate until some **stopping condition** is met (say it happens when $t = T$). The approximate minimizer is $\omega^{(T)}$.

More on stopping conditions later. For now, ...

[1] In these slides, η_t will be constant in t; however, in many scenarios η_t decreases as t increases.

The general procedure

Have a function $\mathcal{L} : \mathbb{R}^p \to \mathbb{R}$ (our case, a loss function with $\Omega = \mathbb{R}^p$). Want to approximate minimizer in $\mathbb{R}^p$ for $\mathcal{L}$.

- Select (any) initial $\omega^{(0)} \in \mathbb{R}^p$ and a learning rate $\eta_t > 0$ for each step $t \geq 1$.[1]
- Next, for $t \geq 1$, iteratively assign

$$\omega^{(t)} = \omega^{(t-1)} - \eta_t \nabla \mathcal{L}(\omega^{(t-1)}).$$

- Iterate until some **stopping condition** is met (say it happens when $t = T$). The approximate minimizer is $\omega^{(T)}$.

More on stopping conditions later. For now, ...

- choose a ("small") threshhold value ε. Stop when, as part of the last update, the <u>change</u> in every parameter <u>divided by its size</u> is not more than ε.

[1] In these slides, η_t will be constant in t; however, in many scenarios η_t decreases as t increases.

The general procedure

Have a function $\mathcal{L} : \mathbb{R}^p \to \mathbb{R}$ (our case, a loss function with $\Omega = \mathbb{R}^p$).
Want to approximate minimizer in $\mathbb{R}^p$ for $\mathcal{L}$.

- Select (any) initial $\omega^{(0)} \in \mathbb{R}^p$ and a learning rate $\eta_t > 0$ for each step $t \geq 1$.[1]
- Next, for $t \geq 1$, iteratively assign

$$\omega^{(t)} = \omega^{(t-1)} - \eta_t \nabla \mathcal{L}(\omega^{(t-1)}).$$

- Iterate until some **stopping condition** is met (say it happens when $t = T$). The approximate minimizer is $\omega^{(T)}$.

More on stopping conditions later. For now, ...

- choose a ("small") threshhold value ε. Stop when, as part of the last update, the change in every parameter divided by its size is not more than ε. That is, stop when

$$\frac{|\omega_j^{(t)} - \omega_j^{(t-1)}|}{|\omega_j^{(t-1)}|} \leq \varepsilon, \qquad \forall 1 \leq j \leq p.$$

[1] In these slides, η_t will be constant in t; however, in many scenarios η_t decreases as t increases.

Batch, Mini-batch, and Stochastic Gradient Descent

In machine learning, our true interest is to minimize the *population* loss function, meaning:

Batch, Mini-batch, and Stochastic Gradient Descent

In machine learning, our true interest is to minimize the *population* loss function, meaning:

> *Given our parameters* $\omega \in \Omega$ *and some point* $(\mathbf{x}, y) \in \mathbb{R}^d \times Y$*, we have a measure of loss for that point,* $\mathcal{L}_{(\mathbf{x},y)}(\omega)$ *("per-example" loss); the population loss at* ω *is the* <u>*expected value*</u> *of per-example loss, expectation with resp. to* $P_{X,Y}$*.*

Batch, Mini-batch, and Stochastic Gradient Descent

In machine learning, our true interest is to minimize the *population* loss function, meaning:

> *Given our parameters $\omega \in \Omega$ and some point $(\mathbf{x}, y) \in \mathbb{R}^d \times Y$, we have a measure of loss for that point, $\mathcal{L}_{(\mathbf{x},y)}(\omega)$ ("per-example" loss); the population loss at ω is the expected value of per-example loss, expectation with resp. to $P_{X,Y}$.*

Lacking access to population loss function, use empirical loss $\mathcal{L}_{\mathcal{S}}$, which gives expected value – mean – from $\mathcal{S}$ of per-example loss.

Batch, Mini-batch, and Stochastic Gradient Descent

In machine learning, our true interest is to minimize the *population* loss function, meaning:

> *Given our parameters* $\omega \in \Omega$ *and some point* $(\mathbf{x}, y) \in \mathbb{R}^d \times Y$*, we have a measure of loss for that point,* $\mathcal{L}_{(\mathbf{x},y)}(\omega)$ *("per-example" loss); the population loss at* ω *is the* *<u>expected value</u>* *of per-example loss, expectation with resp. to* $P_{X,Y}$*.*

Lacking access to population loss function, use empirical loss $\mathcal{L}_{\mathcal{S}}$, which gives expected value – mean – from $\mathcal{S}$ of per-example loss.

Batch Gradient Descent: gradient of $\mathcal{L}_{\mathcal{S}}$ is used in each update.

Batch, Mini-batch, and Stochastic Gradient Descent

In machine learning, our true interest is to minimize the *population* loss function, meaning:

> *Given our parameters $\omega \in \Omega$ and some point $(\mathbf{x}, y) \in \mathbb{R}^d \times Y$, we have a measure of loss for that point, $\mathcal{L}_{(\mathbf{x},y)}(\omega)$ ("per-example" loss); the population loss at ω is the expected value of per-example loss, expectation with resp. to $P_{X,Y}$.*

Lacking access to population loss function, use empirical loss $\mathcal{L}_{\mathcal{S}}$, which gives expected value – mean – from $\mathcal{S}$ of per-example loss.
Batch Gradient Descent: gradient of $\mathcal{L}_{\mathcal{S}}$ is used in each update.
Stochastic Gradient Descent: in each update, select one random point from $\mathcal{S}$ and use gradient of per-example loss at that point.

Batch, Mini-batch, and Stochastic Gradient Descent

In machine learning, our true interest is to minimize the *population* loss function, meaning:

> *Given our parameters $\omega \in \Omega$ and some point $(\mathbf{x}, y) \in \mathbb{R}^d \times Y$, we have a measure of loss for that point, $\mathcal{L}_{(\mathbf{x},y)}(\omega)$ ("per-example" loss); the population loss at ω is the <u>expected value</u> of per-example loss, expectation with resp. to $P_{X,Y}$.*

Lacking access to population loss function, use empirical loss $\mathcal{L}_{\mathcal{S}}$, which gives expected value – mean – from $\mathcal{S}$ of per-example loss.
Batch Gradient Descent: gradient of $\mathcal{L}_{\mathcal{S}}$ is used in each update.
Stochastic Gradient Descent: in each update, select one random point from $\mathcal{S}$ and use gradient of per-example loss at that point.
Mini-batch Gradient Descent: Between Batch and Stochastic. Each update, random subset $\mathcal{S}' \subset \mathcal{S}$ taken (fixed size); the gradient of $\mathcal{L}_{\mathcal{S}'}$ is used.

Batch, Mini-batch, and Stochastic Gradient Descent

In machine learning, our true interest is to minimize the *population* loss function, meaning:

> *Given our parameters $\omega \in \Omega$ and some point $(\mathbf{x}, y) \in \mathbb{R}^d \times Y$, we have a measure of loss for that point, $\mathcal{L}_{(\mathbf{x},y)}(\omega)$ ("per-example" loss); the population loss at ω is the expected value of per-example loss, expectation with resp. to $P_{X,Y}$.*

Lacking access to population loss function, use empirical loss $\mathcal{L}_{\mathcal{S}}$, which gives expected value – mean – from $\mathcal{S}$ of per-example loss.
Batch Gradient Descent: gradient of $\mathcal{L}_{\mathcal{S}}$ is used in each update.
Stochastic Gradient Descent: in each update, select one random point from $\mathcal{S}$ and use gradient of per-example loss at that point.
Mini-batch Gradient Descent: Between Batch and Stochastic. Each update, random subset $\mathcal{S}' \subset \mathcal{S}$ taken (fixed size); the gradient of $\mathcal{L}_{\mathcal{S}'}$ is used.
Important: expected value of gradient used should be $\nabla\mathcal{L}_{\mathcal{S}}$ (which is, hopefully, close to gradient of population loss).

Example with Linear regression

Last time: given sample data $\mathcal{S}$ for simple linear regression, and using MSE as empirical loss, $\mathcal{L}_{\mathcal{S}}(m,b) = \frac{1}{n}\sum_{i=1}^{n}(mx_i + b - y_i)^2$, we found

$$\nabla\mathcal{L}_{\mathcal{S}}(m,b) = \left(\frac{2}{n}\sum_{i=1}^{n}(mx_i + b - y_i)x_i, \quad \frac{2}{n}\sum_{i=1}^{n}(mx_i + b - y_i)\right).$$

Example with Linear regression

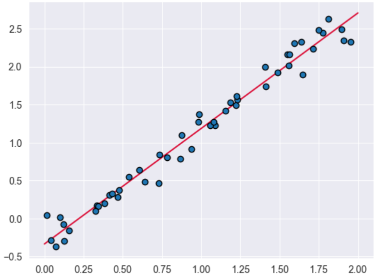
Last time: given sample data $\mathcal{S}$ for simple linear regression, and using MSE as empirical loss, $\mathcal{L}_{\mathcal{S}}(m,b) = \frac{1}{n}\sum_{i=1}^{n}(mx_i + b - y_i)^2$, we found

$$\nabla \mathcal{L}_{\mathcal{S}}(m,b) = \left(\frac{2}{n}\sum_{i=1}^{n}(mx_i + b - y_i)x_i, \quad \frac{2}{n}\sum_{i=1}^{n}(mx_i + b - y_i) \right).$$

Example: batch gradient descent working on the `'Example1.csv'` data.

The LSR line, using closed form.

- $\hat{m} \approx 1.520, \hat{b} = -0.3346$:



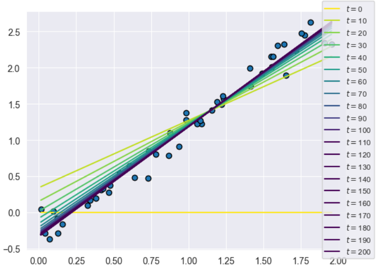
Example with Linear regression

Last time: given sample data $\mathcal{S}$ for simple linear regression, and using MSE as empirical loss, $\mathcal{L}_{\mathcal{S}}(m, b) = \frac{1}{n}\sum_{i=1}^{n}(mx_i + b - y_i)^2$, we found

$$\nabla \mathcal{L}_{\mathcal{S}}(m, b) = \left(\frac{2}{n}\sum_{i=1}^{n}(mx_i + b - y_i)x_i, \quad \frac{2}{n}\sum_{i=1}^{n}(mx_i + b - y_i) \right).$$

Example: batch gradient descent working on the `'Example1.csv'` data.

Plot of selected lines found during batch GD updates; starting parameters $m = 0, b = 0$;



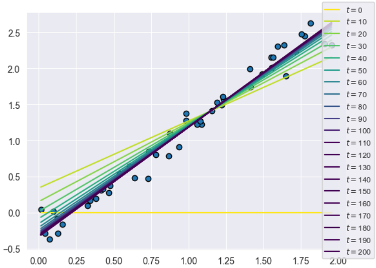
Example with Linear regression

Last time: given sample data $\mathcal{S}$ for simple linear regression, and using MSE as empirical loss, $\mathcal{L}_{\mathcal{S}}(m, b) = \frac{1}{n}\sum_{i=1}^{n}(mx_i + b - y_i)^2$, we found

$$\nabla\mathcal{L}_{\mathcal{S}}(m, b) = \left(\frac{2}{n}\sum_{i=1}^{n}(mx_i + b - y_i)x_i, \quad \frac{2}{n}\sum_{i=1}^{n}(mx_i + b - y_i)\right).$$

Example: batch gradient descent working on the `'Example1.csv'` data.

Plot of selected lines found during batch GD updates; starting parameters $m = 0, b = 0$; learning rate set to 0.1. Parameter values on iteration 208: $m = 1.519$, $b = -0.334$.



Implementing Gradient Descent

Some notes on implementation of the gradient descent updates.

1. Each partial derivative of $\mathcal{L}_{\mathcal{S}}$, can compute it in one line of code.

Implementing Gradient Descent

Some notes on implementation of the gradient descent updates.

1. Each partial derivative of $\mathcal{L}_{\mathcal{S}}$, can compute it in one line of code.

Presuming `x` and `y` are arrays of coordinates for sample data, and that `n = len(x)`, the following computes the partial derivatives:

```
partial_m = (2/n)*np.sum( (m*x + b - y)*x )
partial_b = (2/n)*np.sum( (m*x + b - y) )
```

Implementing Gradient Descent

Some notes on implementation of the gradient descent updates.

1. Each partial derivative of $\mathcal{L}_{\mathcal{S}}$, can compute it in one line of code. Presuming `x` and `y` are arrays of coordinates for sample data, and that `n = len(x)`, the following computes the partial derivatives:

```
partial_m = (2/n)*np.sum( (m*x + b - y)*x )
partial_b = (2/n)*np.sum( (m*x + b - y) )
```

2. To implement GD, want more than one function – at the least, one to compute the gradient (given current parameters); another that performs update and checks for stopping. *Roughly*...

Implementing Gradient Descent

Some notes on implementation of the gradient descent updates.

1. Each partial derivative of $\mathcal{L}_{\mathcal{S}}$, can compute it in one line of code.

Presuming `x` and `y` are arrays of coordinates for sample data, and that `n = len(x)`, the following computes the partial derivatives:

```
partial_m = (2/n)*np.sum( (m*x + b - y)*x )
partial_b = (2/n)*np.sum( (m*x + b - y) )
```

2. To implement GD, want more than one function – at the least, one to compute the gradient (given current parameters); another that performs update and checks for stopping. *Roughly*...

```
## lr is learning rate; threshhold is for stopping;
input: X, y, lr, threshhold
p ← initial array of parameters
while (max of last_update > threshhold){
    grad ← compute_grad(p, X, y)
    last_update ← | grad / p | ## entrywise array division
    # handle p[i] near 0
    p ← p − lr*grad
}
return p
```

Convergence

Does gradient descent converge to point at which loss function is minimized? Is it even guaranteed to converge?

Convergence

Does gradient descent converge to point at which loss function is minimized? Is it even guaranteed to converge?

Short answer: No, not necessarily.

...so, *in what cases* can we guarantee such a thing?

Issues with convergence

To demonstrate the difficulty, imagine a "toy" loss function: $\boldsymbol{\ell} : \mathbb{R} \to \mathbb{R}$ with $\boldsymbol{\ell}(w) = w^2$. At each w we have $\nabla \boldsymbol{\ell} = \left(\frac{d\ell}{dw}\right) = (2w)$.

Issues with convergence

To demonstrate the difficulty, imagine a “toy” loss function: $\ell : \mathbb{R} \to \mathbb{R}$ with $\ell(w) = w^2$. At each w we have $\nabla \ell = \left(\frac{d\ell}{dw}\right) = (2w)$.

Say learning rate: $\eta > 1$. Then, at any $w^{(t)} > 0$, we get

$$w^{(t+1)} = w^{(t)} - 2\eta w^{(t)} < w^{(t)} - 2w^{(t)} = -w^{(t)},$$

and so $|w^{(t+1)}| > |w^{(t)}|$.

Issues with convergence

To demonstrate the difficulty, imagine a "toy" loss function: $\ell : \mathbb{R} \to \mathbb{R}$ with $\ell(w) = w^2$. At each w we have $\nabla \ell = \left(\frac{d\ell}{dw}\right) = (2w)$.

Say learning rate: $\eta > 1$. Then, at any $w^{(t)} > 0$, we get

$$w^{(t+1)} = w^{(t)} - 2\eta w^{(t)} < w^{(t)} - 2w^{(t)} = -w^{(t)},$$

and so $|w^{(t+1)}| > |w^{(t)}|$. Likewise, if $w^{(t)} < 0$ then $|w^{(t+1)}| > |w^{(t)}|$.

Issues with convergence

To demonstrate the difficulty, imagine a “toy” loss function: $\ell : \mathbb{R} \to \mathbb{R}$ with $\ell(w) = w^2$. At each w we have $\nabla \ell = \left(\frac{d\ell}{dw}\right) = (2w)$.

Say learning rate: $\eta > 1$. Then, at any $w^{(t)} > 0$, we get

$$w^{(t+1)} = w^{(t)} - 2\eta w^{(t)} < w^{(t)} - 2w^{(t)} = -w^{(t)},$$

and so $|w^{(t+1)}| > |w^{(t)}|$. Likewise, if $w^{(t)} < 0$ then $|w^{(t+1)}| > |w^{(t)}|$.

So, it diverges when $\eta > 1$. However, if $0 < \eta < 1$, then it will converge (to minimizer $w = 0$).

Issues with convergence

To demonstrate the difficulty, imagine a "toy" loss function: $\ell : \mathbb{R} \to \mathbb{R}$ with $\ell(w) = w^2$. At each w we have $\nabla \ell = \left(\frac{d\ell}{dw}\right) = (2w)$.

Say learning rate: $\eta > 1$. Then, at any $w^{(t)} > 0$, we get

$$w^{(t+1)} = w^{(t)} - 2\eta w^{(t)} < w^{(t)} - 2w^{(t)} = -w^{(t)},$$

and so $|w^{(t+1)}| > |w^{(t)}|$. Likewise, if $w^{(t)} < 0$ then $|w^{(t+1)}| > |w^{(t)}|$.
So, it diverges when $\eta > 1$. However, if $0 < \eta < 1$, then it will converge (to minimizer $w = 0$).

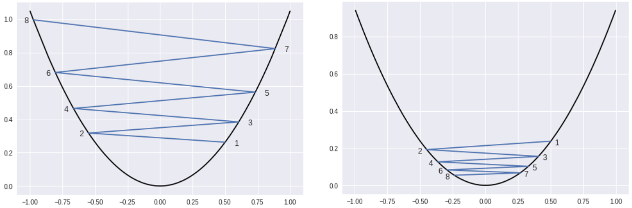


Figure: Gradient descent on $\ell(w) = w^2$. Left: $\eta = 1.05$; right: $\eta = 0.95$.

Guarantees of convergence

If your loss function is differentiable and a **convex function**, and if have some “control” on size of the gradient then, by choosing η small enough, can guarantee convergence.

[2] Meaning: $\exists$ a constant C s.t. for all ω_1, ω_2, $|\nabla\mathcal{L}(\omega_1) - \nabla\mathcal{L}(\omega_2)| \leq C|\omega_1 - \omega_2|$.

Guarantees of convergence

If your loss function is differentiable and a **convex function**, and if have some "control" on size of the gradient then, by choosing η small enough, can guarantee convergence.

Theorem

Suppose that $\mathcal{L} : \mathbb{R}^p \to \mathbb{R}$ is differentiable and convex, and suppose that $\nabla \mathcal{L}$ is Lipschitz continuous[2] *with some constant $C > 0$ and that $\eta \leq 1/C$. Then, for a minimizer ω^* of $\mathcal{L}$,*

$$\mathcal{L}(\omega^{(t)}) - \mathcal{L}(\omega^*) \leq \frac{|\omega^{(0)} - \omega^*|^2}{2\eta t}.$$

[2] Meaning: $\exists$ a constant C s.t. for all ω_1, ω_2, $|\nabla \mathcal{L}(\omega_1) - \nabla \mathcal{L}(\omega_2)| \leq C|\omega_1 - \omega_2|$.

Guarantees of convergence

If your loss function is differentiable and a **convex function**, and if have some "control" on size of the gradient then, by choosing η small enough, can guarantee convergence.

Theorem

Suppose that $\mathcal{L} : \mathbb{R}^p \to \mathbb{R}$ is differentiable and convex, and suppose that $\nabla\mathcal{L}$ is Lipschitz continuous[2] with some constant $C > 0$ and that $\eta \leq 1/C$. Then, for a minimizer ω^* of $\mathcal{L}$,

$$\mathcal{L}(\omega^{(t)}) - \mathcal{L}(\omega^*) \leq \frac{|\omega^{(0)} - \omega^*|^2}{2\eta t}.$$

- The difference between $\mathcal{L}(\omega^{(t)})$ and the minimum of $\mathcal{L}$ is bounded by a constant times $1/t$.

[2]Meaning: $\exists$ a constant C s.t. for all ω_1, ω_2, $|\nabla\mathcal{L}(\omega_1) - \nabla\mathcal{L}(\omega_2)| \leq C|\omega_1 - \omega_2|$.

Guarantees of convergence

Previous theorem requires using actual gradient of $\mathcal{L}$ in each update step. Here is a convergence guarantee that allows for a random vector $\mathbf{D}_t$, in place of $\nabla\mathcal{L}$, as long as $\mathbb{E}[\mathbf{D}_t|\omega^{(t)}] = \nabla\mathcal{L}(\omega^{(t)})$.

- e.g., in Stochastic or Mini-batch gradient descent.

Guarantees of convergence

Previous theorem requires using actual gradient of $\mathcal{L}$ in each update step. Here is a convergence guarantee that allows for a random vector $\mathbf{D}_t$, in place of $\nabla\mathcal{L}$, as long as $\mathbb{E}[\mathbf{D}_t|\omega^{(t)}] = \nabla\mathcal{L}(\omega^{(t)})$.

- e.g., in Stochastic or Mini-batch gradient descent.

Theorem

Suppose that $\mathcal{L}$ is differentiable and convex, that $\omega^{(0)} = \mathbf{0}$, and that $\eta = \frac{1}{\sqrt{K}}$ for an integer $K > 0$. Finally, suppose that $|\mathbf{D}_t| \leq 1$ for all $1 \leq t \leq K$. Then, for a minimizer ω^* of $\mathcal{L}$,

$$\mathbb{E}[\mathcal{L}(\bar{\omega})] - \mathcal{L}(\omega^*) \leq \frac{1}{\sqrt{K}}$$

where $\bar{\omega}$ is the average of $\omega^{(1)}, \omega^{(2)}, \ldots, \omega^{(K)}$.