Algoritmi Gradient Descent per la Data Analysis - 25/11 + 26/11

Algoritmi a discesa del gradiente per l'analisi dei dati.

Nel gergo comune si parla di algoritmi del gradiente.

Apprendimento statistico come problema di ottimizzazione

La soluzione di molti compiti di statistical learning è:

$$\min_{\beta} J(\beta)$$

che è un problema di minimizzazione unconstrained (non vincolato).

Non vincolato perché non ci sono vincoli che limitano β, non scriviamo neanche $\beta \in R^p$ proprio per sottolineare questo.

Richiamiamo ridge, si può scrivere in due forme, una che dice che minimizza il solito RSS + un parametro che era la norma quadra, e un'altra che diceva che la norma quadra non doveva superare un valore, quello era quindi un problema vincolato.

In un problema non vincolato per trovare il minimo basta fare derivata posta a 0.

In un **problema vincolato** per trovare il minimo non funziona solo così e ci sono tecniche come i moltiplicatori di Lagrange, che per intenderci sono quelli che fanno scrivere il problema Ridge dalla forma con vincolo a quella con λ.

> 💡 Esempio 1: Regressione Lineare
>
> Minimizziamo il Mean Squared Error

> 💡 Esempio 2: Classificazione
>
> Minimizziamo la conditional cross-entropy

Objective function (funzione obiettivo)

Sia $D \in \mathcal{D}$ un qualche dato casuale, ad esempio, D potrebbe essere la coppia feature/label $D = (X, Y)$.

Si sceglie una funzione di perdita adeguata (suitable loss function) $Q_\beta(D)$, ad esempio, una funzione di $D \in \mathcal{D}$ parametrizzata da $\beta \in \mathbb{R}^p$.

Il nostro compito è stimare β tale da minimizzare la loss, mediata su D.

Assumendo che la distribuzione statistica di D è nota, minimizziamo

$$J(\beta) = E[Q_\beta(D)]$$

che è spesso chiamata costo, rischio o funzione obiettivo.

Prof:

La funzione obiettivo in breve è quella che dobbiamo minimizzare.

I nostri dati sono random quindi si parla di una branca dell'ottimizzazione che prende il nome di ottimizzazione stocastica.

La loss rispecchia per certi dati osservati e per un certo parametro scelto quanto perdiamo. Lo scopo è scegliere β per minimizzare la loss, ma la loss dipende anche dai dati che sono random e quindi bisogna mediare sui dati, che conoscendo il modello è la media analitica.

La funzione obiettivo resta solo dipendente da β.

Esempio 1: Linear Regression

$D = (X, Y)$, con $X \in R^p$ e $Y \in R$.

La loss function è:

$$Q_\beta(d) = Q_\beta(x, y) = (x^T\beta - y)^2$$

La cost function è:

$$J(\beta) = E[(X^T\beta - Y)^2] = E[(\sum X_k\beta_k - \overline{Y})^2]$$

Prof:

Noi per la linear regression abbiamo:

- calcolato la soluzione ottima come media a posteriori sul modello di regressione lineare;
- con l'RSS abbiamo calcolato una media della $Q_\beta(d)$ ma imponendo una forma sui dati, non come media.

Questa che abbiamo fatto ora è la terza possibilità anche se ovviamente la soluzione è sempre la stessa, il β che minimizza è il β vero che non conosciamo.

Esempio 2: Logistic regression

$D = (X, Y)$, con $X \in R^p$ e $Y \in \{-1, 1\}$.

La loss function è:

$$Q_\beta(d) = Q_\beta(x, y) = ln(1 + e^{-yx^T\beta})$$

La cost function è:

$$J(\beta) = E[ln(1 + e^{-YX^T\beta})]$$

Prof:

Ci sono almeno due cose che non sappiamo:

- per calcolare la funzione di costo non possiamo fare la media perché non abbiamo la distribuzione;
- se anche avessimo la distribuzione il minimo non lo sappiamo trovare perché abbiamo detto che ci vuole per forza un algoritmo.

Per prima cosa proveremo a capire come si risolve un problema di questo tipo con la distribuzione, quindi sapendo fare la media, poi proveremo senza e quindi avendo i campioni del Training Set e cercando di andare a convergere al valore del minimizzatore della funzione vera.

Visto che noi siamo chiamati a minimizzare la funzione di costo vera facciamo prima quello e poi ci adattiamo al fatto che non abbiamo il modello.

Standard assumptions adottate in statistical learning

- $\nabla J(\beta)$ è il gradient vector p x 1, la cui i-esima entry è $\frac{\partial J(\beta)}{\partial \beta_i}$
- $||\,\cdot\,||$ denota la Norma Euclidea

C'è un teorema interessante di equivalenza delle norme che dice che prese due norme queste differiscono solo di una costante che dipende dalla dimensione in cui ci troviamo. Se due cose sono vicine in una norma sono vicine anche in un'altra. Ad esempio se tende a zero una norma tende a zero anche l'altra.

- bisogna sapere cosa sono gli autovettori ($Av = \lambda v$), gli autovalori ($det(A - \lambda I) = 0$), i determinanti
- $\delta > 0$ è la costante di Lipschitz, $v > 0$ è la costante di strong-convexity

> 💡 **Lipschitz-continuous gradient**
> $\forall \beta' \neq \beta'' : \; ||\nabla J(\beta'') - \nabla J(\beta')|| \leq \delta ||\beta'' - \beta'||$

Si dice che ∇J è Lipschitz se la norma della differenza della funzione è minore uguale di una costante per la norma della differenza dei punti. Significa che se due punti sono vicini i valori delle funzioni sono vicini. Se le ascisse diventano vicine le ordinate diventano vicine. Ci ricorda il concetto normale di funzione continua. Ma tutte le funzioni continue devono rispettare questa disuguaglianza? No, per le continue più si avvicinano le ascisse e più si avvicinano le ordinate, per le Lipschitz vale anche la disuguaglianza il che ci dice che le ordinate si schiacciano l'una sull'altra minimo con la velocità con la quale si avvicinano le ascisse. Ovviamente Lipschitz implica continua.

- Il gradiente varia in maniera dolce, liscia (smoothly).

- strong convexity ⇒ strict convexity ⇒ convexity

Convexity

$\forall \alpha \in (0,1)$ e $\forall \beta' \neq \beta''$:

$J(\alpha\beta' + (1-\alpha)\beta'') \leq \alpha J(\beta') + (1-\alpha)J(\beta'')$

Presi due punti e connessi con una corda la funzione è sotto la corda nell'intervallo tra i due punti.

A sinistra abbiamo i valori della funzione compresi tra $J(\beta')$ e $J(\beta'')$.

A destra abbiamo i valori sulla corda che connette $J(\beta')$ e $J(\beta'')$.

Strict Convexity

$\forall \alpha \in (0,1)$ e $\forall \beta' \neq \beta''$:

$J(\alpha\beta' + (1-\alpha)\beta'') < \alpha J(\beta') + (1-\alpha)J(\beta'')$

Abbiamo perso l'uguale al centro che comprendeva anche i casi degeneri.

Strong Convexity

$\forall \alpha \in (0,1)$ e $\forall \beta' \neq \beta''$:

$J(\alpha\beta' + (1-\alpha)\beta'') \leq \alpha J(\beta') + (1-\alpha)J(\beta'') - \frac{v}{2}\alpha(1-\alpha)||\beta' - \beta''||^2$

La convessità forte ci dice che i punti non stanno solo al di sotto della corda, ma sotto la corda meno qualcos'altro.

Fortunatamente queste condizioni di convessità, se la funzione è derivabile due volte, si semplificano. Di fatto per valutare la convessità ad Analisi 1 si guardava la derivata seconda, questa è l'estensione al caso multidimensionale.

Convessità quando $J(\beta)$ è twice-differentiable (differenziabile due volte)

- $\nabla^2 J(\beta)$ è la matrice Hessiana p x p (sempre simmetrica), la cui (i,j)-esima entry è $\frac{\partial^2 J(\beta)}{\partial \beta_i \partial \beta_j}$
- I è la matrice identità p x p
- La notazione $A > B$ (rispettivamente, $A \geq B$) per due matrici simmetriche A e B significa che $A - B$ è definita positiva (rispettivamente positive semi-definite)

Per le matrici $A > B$ significa che gli **autovalori** di A sono maggiori di quelli di B.

$A > 0$ ($A \geq 0$) può significare che gli autovalori sono positivi, matrice definita positiva (rispettivamente semi-definita positiva), o che i valori della matrice sono positivi, bisogna controllare cosa intende l'autore.

Abbiamo spiegato questa definizione per matrici simmetriche perché in generale gli autovalori di una matrice sono numeri complessi e non avrebbe senso dire chi è più grande e chi è più piccolo, per le matrici simmetriche però gli autovalori sono reali.

Come anticipato le definizioni di convessità si semplificano quando $J(\beta)$ è doppiamente derivabile:

Convessità	Stretta Convessità	Forte convessità
$\nabla^2 J(\beta) \geq 0 \; \forall \beta$ (l'Hessiano deve essere semi-definito positivo)	$\nabla^2 J(\beta) > 0 \; \forall \beta$ (l'Hessiano deve essere definito positivo)	$\nabla^2 J(\beta) \geq vI \; \forall \beta$ (l'Hessiano meno una costante di convessità per l'identità deve essere semi-definito positivo)

> 💡 **Esempio**
>
> Il costo Mean-Square-Error usato nella Linear Regression è strongly convesso **iff** la matrice di covarianza $E[XX^T]$ è invertibile (verificare questa affermazione calcolando la matrice Hessiana).

ESEMPIO 1 in aula

$J(\beta) = e^{-\beta} \rightarrow J'(\beta) = -e^{-\beta} \rightarrow J''(\beta) = e^{-\beta}$

Calcolata la derivata seconda ci chiediamo:

$J(\beta)$ è convessa?

L'esponenziale non farà mai 0 quindi sì.

$J(\beta)$ è Strettamente convessa?

L'esponenziale non farà mai 0 quindi sì.

$J(\beta)$ è Fortemente convessa?

νI nel caso monodimensionale (cioè questo) è uguale a ν che è un valore positivo, dobbiamo quindi verificare se esiste un numero positivo tale che la derivata seconda è sicuramente superiore ad esso. Il limite per $\beta \rightarrow \infty$ porta $e^{-\beta}$ a 0, quindi qualunque valore ν scelto per quanto piccolo sarà comunque superato dalla derivata seconda, che potrà essere più piccola. Quindi non è fortemente convessa.

L'elemento fondamentale è $\forall \beta$, è vero che scelto un β ci sarà sempre un numero più piccolo, ma se β si sposta no.

Cambiamo funzione.

$J(\beta) = e^{-\beta} + 3\beta^2 \rightarrow J'(\beta) = -e^{-\beta} + 6\beta \rightarrow J''(\beta) = e^{-\beta} + 6$

In questo caso indipendentemente dal valore di β sicuramente la derivata seconda sarà maggiore di 6, la funzione è diventata Strongly Convex.

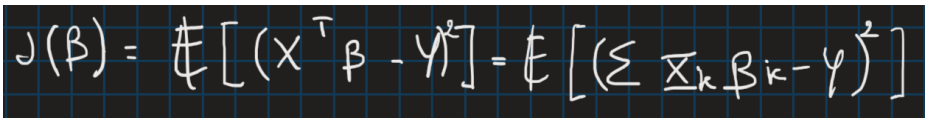
Come abbiamo reso la funzione originale Strongly Convex? Sommando la funzione quadratica.

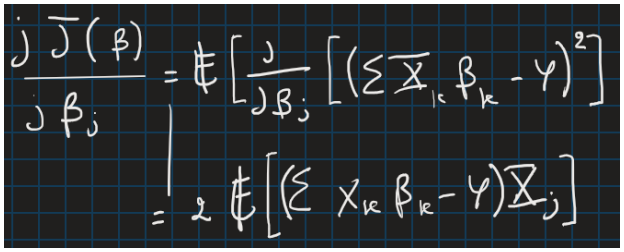
Ci ricorda Ridge, anche lì aggiungevamo una funzione quadratica, ora sappiamo perché, per regolarizzare, nello specifico rendere Strongly Convex, qualcosa che non lo era. Regolarizzazione può avere vari significati ma è sempre apparare qualcosa che non piace di una funzione.

Possiamo ad esempio sommare $\epsilon\beta^2$ quindi una cosa piccolissima, solo allo scopo di rendere la funzione Strongly Convex. Sommando una cosa molto piccola non perturbiamo la soluzione ma regolarizziamo.

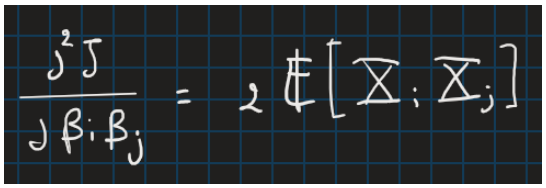
ESEMPIO 2 in aula

Dobbiamo calcolare l'Hessiano della funzione di costo della Regressione lineare.

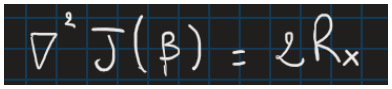




Se si fa la derivata la si può portare dentro la media



Quella all'interno della media è la matrice di correlazione, la matrice di varianza-covarianza, R_x



La cosa **particolarissima** che notiamo è che l'Hessiano di $J(\beta)$ non dipende da β, il caso quadratico in una dimensione corrisponde ad una parabola, la derivata seconda di una parabola è costante. Similarmente, il caso di regressione lineare con costo quadratico è come il caso parabolico, la derivata seconda non dipende dal parametro.

R_X è simmetrica ed è semi-definita positiva, gli autovalori sono sempre ≥ 0, se qualche autovalore è proprio 0 la R_X non è invertibile perché il determinante è 0 e le formule di nostro interesse non sono più utilizzabili, questo è il caso di **Collinearità**. Uno dei modi in cui si risolve questo problema è con dei metodi di regolarizzazione come Ridge.

Se gli autovalori sono invece tutti maggiori di 0 la R_X è sicuramente invertibile visto che il determinante è scrivibile come prodotto degli autovalori.

Ora parliamo della convessità di $2R_X$, è convesso perché è sempre ≥ 0.

Invece è strettamente convesso quando tutti gli autovalori di R_X sono positivi ($Amin(R_X) > 0$).

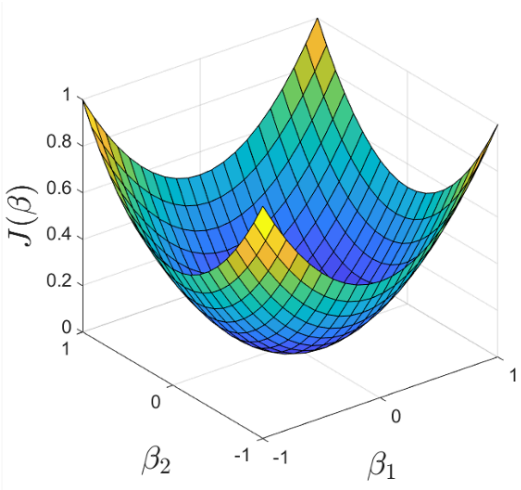
Per Strongly convex vogliamo $2R_X \geq \nu I \rightarrow R_X - \frac{\nu I}{2} \geq 0$.

Quindi ci serve calcolare gli autovalori della differenza in questione, come si fa? Normalmente non con la differenza degli autovalori ma visto che abbiamo la matrice Identità sì, quindi stiamo chiedendo che sia verificato $Amin(R_X) \geq \frac{\nu}{2}$, esiste una costante che ci permette questo? Sappiamo che una matrice p x p ha p autovalori, sappiamo che sono tutti positivi e quindi sicuramente esiste un valore ν per il quale è verificata quella disuguaglianza.

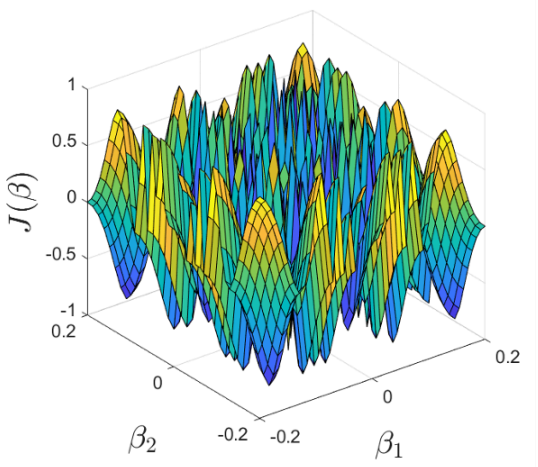
Perché non abbiamo avuto lo stesso problema dell'esponenziale? Perché qui non varia in funzione di β per cui strict e strong convexity sono la stessa cosa. Qui abbiamo la parabola che è proprio il caso ideale.

Un esempio grafico

Questo esempio è la parabola che è ovvio che è Strongly Convex e con Gradiente Lipschitz-continuo, se non fosse stato valido neanche per la parabola avremmo studiato una cosa inutile per i veri problemi di ottimizzazione.



Quest'altro esempio è solo rumore, quindi è comprensibile che non abbia proprietà di interesse, è non strongly convex e il gradiente non è Lipschitz-continuo.



Perché ci interessano le funzioni convesse e la Lipschitz Continuity?

Una funzione solo convessa non sappiamo cosa fa, pensiamo al costo quadratico, addirittura la matrice R_X può essere non invertibile, cioè il sistema lineare che abbiamo non possiamo risolverlo perché ha infinite soluzioni, quindi la sola convessità non ci da informazioni importanti.

Strictly Convex significa che la funzione potrebbe non avere un minimo ma se lo ha ne ha uno solo e quindi non abbiamo il grande problema dei tanti minimi locali.

Strongly Convex aggiunge delle sicurezze in più, non solo sull'unicità del minimo ma anche sulla sua esistenza.

Gradiend descent: basics (26/11)

Iniziamo a parlare di un algoritmo per risolvere la minimizzazione nel contesto nel quale non sappiamo risolvere l'equazione che pone il gradiente uguale a 0.

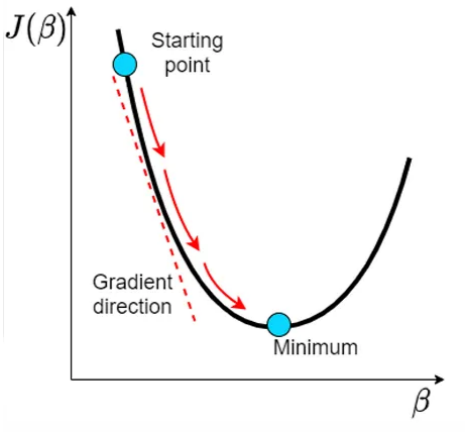
Assumiamo di stare lavorando con funzioni doppiamente derivabili.

Quando il gradiente è uguale a 0 si parla di punto stazionario.

Se la funzione è strettamente convessa il segno $>$ della disuguaglianza ci permette di dire che il punto stazionario è un punto di minimo ed è l'unico punto di minimo.

💡 **GD algorithm**

1. Scegliamo uno starting point $\beta(0)$
2. ripetiamo per i=1,2,...
3. $\beta(i) = \beta(i-1) - \mu\nabla J(\beta(i-1))$ con $\mu > 0$ ma piccolo
4. fino alla end condition



μ è chiamato step-size (o learning rate) e governa il tasso di convergenza.

I $\beta(i)$, chiamati anche iterates sono i punti sulle ascisse nei quali viene valutata la funzione.

Valutiamo la derivata della funzione nell'iterata nella quale siamo, moltiplicata per la step-size.

Quando il gradiente è negativo il - nella formula fa andare in avanti verso il minimo, viceversa se il gradiente è positivo.

La grandezza del passo dipende dalla grandezza della derivata, ha senso così si va più velocemente verso il minimo.

Le iterazioni dell'algoritmo porteranno a "saltellare" intorno al minimo, oppure se non si saltella visto che avvicinandosi al minimo il gradiente si rimpicciolisce si continua ad avvicinarsi molto lentamente. In ogni caso converge al minimo.

Il ruolo della step-size è che se è troppo grande non convergo al minimo perché i passi sono troppo grandi. Ci sono studi che dicono quanto deve essere piccolo e come deve essere scelto μ. Di solito si sa che se μ è al di sotto di un certo valore noto che è in funzione del problema, allora l'algoritmo converge.

Cosa vuol dire che l'algoritmo **converge**? Che si avvicina indefinitamente alla soluzione, infatti c'è bisogno di una condizione di terminazione, che può essere ad esempio "la soluzione è variata rispetto alla precedente di tot%".

In cosa entra in gioco **la strong convexity**? Ci garantisce che la funzione ha esistenza e unicità del minimo, inoltre rientra nella dimostrazione della convergenza per una cosa importante che forse approfondiremo dopo.

In cosa entra in gioco la condizione di **Lipschitz**? La variazione del gradiente in due punti è minore uguale della variazione dei due punti, questo legame tra la variazione delle ordinate e quella delle ascisse è importante per fare in modo che se ci avviciniamo molto in β ci avviciniamo anche nel gradiente e quindi man mano che si avvicina al minimo sulle ascisse succede anche sulle ordinate e questo succede secondo una legge (la disuguaglianza della condizione di Lipschitz) che permette di schiacciarsi sul minimo, in questo modo si evitano salti non desiderati.

Questi algoritmi in Machine Learning spesso si studiano non chiedendosi se β va verso il minimo vero ma accontentandosi di vedere se il gradiente diventa abbastanza piccolo, facendo il plot del gradiente.

Tipicamente in teoria dell'ottimizzazione non sempre si può garantire che il β va verso il minimo ma si può garantire che il gradiente diventa piccolo piccolo (può significare che la funzione ha più minimi e noi stiamo andando verso uno dei tanti) e questa cosa va bene in Deep Learning e applicazioni simili perché non si hanno tante garanzie.

Noi in Statistical Learning abbiamo più garanzie viste tutte le premesse che facciamo (gradiente Lipschitz continuo e strong convexity), queste condizioni sono desiderate perché il valore di β ha spesso un significato fisico e vogliamo sapere proprio quanto vale il β del minimo, qundi è importante poter fare inferenza sul valore di β, per renderlo possibile ci mettiamo in condizioni nelle quali il β può convergere e vediamo se l'algoritmo ce lo garantisce.

Se non ci sono le condizioni si usa l'approccio del Machine Learning ma all'esame il minimo si potrà trovare, coerentemente a quello che stiamo studiando.

Gradient descent: convergenza

Constant step-size

Per μ più piccolo di un certo valore (che dipende dalla costante di Lipschitz δ e dalla costante di strong-convexity ν), GD converge all'esatto minimizer (minimizzatore).

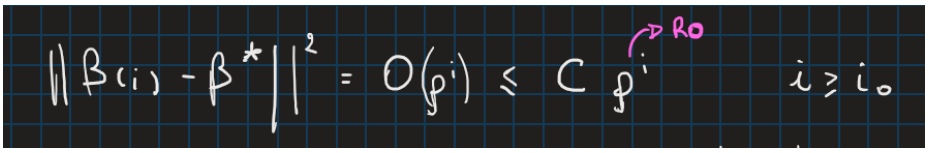
L'algoritmo converge ad un rate geometrico dell'ordine $O(\rho^i)$ dove $\rho \in (0,1)$ è una funzione decrescente della step-size μ e dipende da δ e ν.

Digressione del prof sulla Big-O Notation

$O(f(x))$ è un concetto matematico, si chiama notazione di Landau, significa che la funzione di cui stiamo parlando da un certo i in poi è minore uguale di una costante moltiplicata per l'argomento della O, quindi è "dell'ordine di" l'argomento della O.

Quindi dire che l'algoritmo converge ad un rate geometrico dell'ordine $O(\rho^i)$ significa che l'errore commesso da un certo i in poi è $\leq$ di una costante per ρ^i e quindi sta scendendo a rate geometrico.

Il fatto che ρ è un numero tra 0 e 1 è interessante, per i che cresce questo numero tende a 0, abbiamo detto che l'algoritmo converge ad un tasso geometrico, questa è una proprietà desiderabile? In genere si è soddisfatti se l'errore tende a 0 con legge esponenziale.



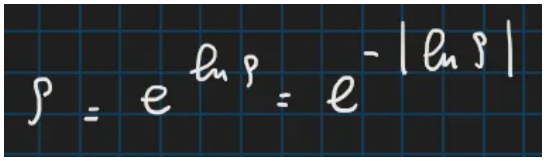
$\beta(i)$ è l'uscita dell'algoritmo mentre β^* è il minimo ottimo

Più in generale è possibile affermare che $f(x) = O(g(x))$ con $x -> \infty$ significa che per un certo $x > x_0$ è vero che $|f(x)| \leq C|g(x)|$. Nella formula in immagine per p non c'è bisogno del modulo

perché stiamo trattando quantità positive.

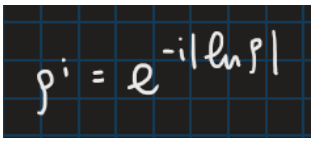
Abbiamo detto che noi vorremmo che l'errore andasse a 0 in maniera esponenziale e stiamo cercando di capire se ρ^i è una legge esponenziale o meno.

Dimostriamo che è esponenziale nel seguente modo



La prima uguaglianza è vera per definizione. Dopodichè ragioniamo sul fatto che $ln(\rho)$ è sicuramente negativo visto che ρ è compreso tra 0 e 1, quindi è possibile scrivere la seconda uguaglianza.

A questo punto elevare ad i entrambi i lati dell'uguaglianza non cambia nulla e in questo modo abbiamo dimostrato che ρ^i è una legge esponenziale.



Che p^i fosse esponenziale è una ovvietà ma visto che nessuno ha risposto ci ha scritto la dimostrazione.

Si chiama tasso geometrico perché tecnicamente se è una funzione di una variabile continua si parla di esponenziale, siccome siamo in discreto si parla di geometrico, i è un indice discreto e quindi si dice che il rate è geometrico. In alcuni contesti si parla anche di rate lineare con lo stesso significato.

Quindi l'algoritmo funziona bene, converge al minimo e lo fa esponenzialmente, la convergenza dipende dal fatto che μ sia più piccolo di un certo valore. Questo valore del quale bisogna essere più piccoli dipende da δ e ν, in che modo? Si intende facilmente che è più si è Strongly Convex e più si è Lipschitz e più il vincolo sarà lasco, cioè esisteranno più valori di μ per i quali si converge al minimo. Se il gradiente è più Lipschitz continuo (come si fa ad essere più è da definire) vuol dire che δ è più piccola la costante e quindi subito ci si schiaccia sul minimo. Se la costante di convessità è più grande si è più Strongly Convex e quindi questa legge andrà a favorire la convergenza, cioè di allargare il range di μ per cui vale la convergenza.

Anche il tasso ρ è importante, se questo è vicino ad 1 la convergenza è lenta. Se lo step-size è piccolo si converge più lentamente perché si fanno passi più piccoli. Quindi capiamo che ρ dipende da μ, se μ è piccolo ρ è grande.

> 💡 Osserviamo che c'è un trade-off, μ deve essere piccolo altrimenti non convergo ma se è troppo piccolo convergo troppo lentamente.

Decaying step-size

Se consideriano una step size iteration-dependent $\mu = \mu(i)$ con $\sum_{i=1}^{\infty} \mu(i) = \infty$ e $\lim_{i \to \infty} \mu(i) = 0$, GD converge all'esatto minimizer, però ad un rate non geometrico.

Un scelta tipica è $\mu(i) = \frac{\tau}{i}$ (con $\tau > 0$), che porta ad un rate $O(\frac{1}{i^{2\nu\tau}})$.

Digressione del prof

Si chiama anche diminishing step-size, una dimensione del passo che diventa più piccola nel tempo ha senso perché più tempo passa e più ci si sta avvicinando al minimo, e quando ci si avvicina al minimo fare passi più piccoli fa saltellare di meno.

Bisogna garantire che la somma dei $\mu(i)$ vada ad infinito perché altrimenti, se la somma non diverge, significa che μ è diventato così piccolo che l'algoritmo non ce la fa a portare al minimo. Allo stesso tempo vogliamo che il limite per i infinito delle $\mu(i)$ faccia 0, altrimenti non sarebbe "Diminishing". Sembra insensato ma matematicamente ha senso, del resto la serie armonica $\sum_{i}^{\infty} \frac{1}{i}$ diverge nonostante $\lim_{i \to \infty} \frac{1}{i} = 0$.

L'algoritmo converge ma ad un tasso non geometrico, nell'esempio della slide il rate è $O(\frac{1}{i^{2\nu\tau}})$ che è una legge iperbolica o iperbolica con qualche potenza, non esponenziale, è più lenta. Quindi perché si fa? Lo scopriamo tra poco.

Gradient descent: problemi

In pratica non possiamo minimizzare $E[Q_\beta(D)]$ perché non abbiamo abbastanza informazioni sulla distribuzione di D.

Però, abbiamo accesso al dataset $\{d_i\}_{i=1}^{n}$. In accordo, possiamo rimpiazzare il rischio atteso con il rischio empirico

$$J_{emp}(\beta) = \frac{1}{n} \sum_{i=1}^{n} Q_\beta(d_i)$$

Cosa possiamo dire sul gradiente del rischio empirico?

- spesso non è adatto a grandi dataset(se i dataset sono molto grandi ogni volta bisogna ricalcolare la funzione);
- in applicazioni online il dataset è costruito progressivamente nel tempo;
- non è adatto a implementazioni distribuite.

Soluzione: usare la seguente approssimazione istantanea basandoci su sample individuali:

$$\widehat{\nabla J}_i(\beta) = \nabla Q_\beta(d_i)$$

Digressione del prof:

Perché nell'esempio della regressione logistica non si può implementare l'algoritmo del gradiente?

$$J(\beta) = E[ln(1 + e^{-YX^T\beta})]$$

Il primo problema è che non sappiamo calcolare la media perché non abbiamo la distribuzione.

La soluzione però sembra facile visto che abbiamo un dataset, sostituiamo il rischio atteso con l'empirical risk.

J è una funzione di β, il training set lo abbiamo, il gradiente della loss function lo sappiamo fare, la loss function è semplicemente l'argomento della media e nei problemi che consideriamo ne sappiamo fare il gradiente.

Applichiamo l'algoritmo del gradiente ad una approssimazione, si parla in questo caso di algoritmo del gradiente stocastico, perché la funzione dipende dal training set che è aleatorio.

Stiamo ora entrando nell'ambito dell'ottimizzazione stocastica, non abbiamo più una funzione ma una funzione che dipende da dei parametri aleatori. In Machine Learning questa implementazione prende il nome di batch, ha il problema che se i dataset sono molto grandi ogni volta bisogna ricalcolare la funzione su tutto il dataset e il minimo che si trova assomiglia al minimo vero se n è grande (la J approssimata assomiglia alla J) ma questo dover ricalcolare la funzione è pesante.

In applicazioni dette online (quando si parla di online learning) si assume che il dataset cambia nel tempo, arrivano nuovi dati, quindi non si sa quanti e quali di questi dati devo prendere per ricalcolare la funzione.

E se poi ci sono pezzi di dataset sparsi su macchine diverse perché ci sono implementazioni distribuite (tipo cloud) è ancora peggio perché bisogna calcolare il gradiente unico e quindi scambiare i dati il che pone il problema della privacy, i dati non te li possono o vogliono dare, ti possono dare un qualcosa che permette di calcolare il gradiente ma che non permetta di ricavare i dati originali.

Si costruisce quindi un'alternativa chiamata Stochastic gradient algorithm, che non è prendere tutti i dati e calcolare la funzione approssimata e trattarla come la funzione vera (e quindi conseguentemente ogni volta che cambia β bisogna ricalcolare la funzione) ma si fa una cosa molto più semplice, si calcola come approssimazione del gradiente il gradiente della loss function nel punto d_i. Si vede un solo campione, è come se invece di prendere tutti gli n se ne prende solo 1, si chiama approssimazione istantanea. Immaginiamo di avere n infinito, ogni volta che arriva un campione nuovo prendiamo solo quello e invece di fare la media su tutti i valori si calcola la stima del gradiente solo su quello.

Questo risolve il problema dei dataset grandi perché le iterazioni vanno fatte ma per ogni iterazione si prende un singolo campione.

Questo risolve il problema online. Arriva un campione alla volta e quello viene processato.

Si risolve il problema distribuito. Ognuno ha i propri dati e usa un campione alla volta di quelli che ha.

Quindi sono stati risolti 3 problemi con una soluzione apparentemente stupida, fino ad ora abbiamo sempre parlato di legge dei grandi numeri e di dataset grandi quindi questo sembra insensato.

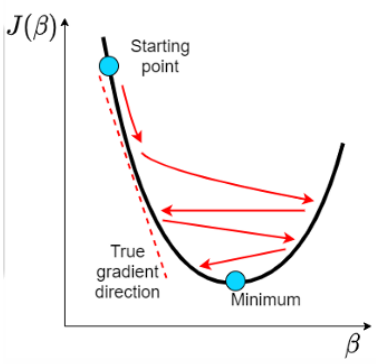
Si può anche fare un approccio ibrido, una via di mezzo, una implementazione detta Mini-batch, cioè magari non si usa proprio un campione ma ad esempio 5. Ovviamente per studiarlo conviene vedere il caso ad un campione.

Quindi esiste l'implementazione in Batch, l'implementazione stocastic gradient (uno alla volta) e l'implementazione Mini-batch dove si prende un numero k di campioni.

Stochastic gradient descent: basics

SGD algorithm

1. Si sceglie uno starting point $\beta(0)$
2. ripetere per i = 1, 2, ...
3. prendere un nuovo campione d_i
4. $\beta(i) = \beta(i-1) - \mu\widehat{\nabla J}_i(\beta(i-1))$
5. fino al raggiungimento della end condition



$\widehat{\nabla J}_i(\beta) = \nabla Q_\beta(d_i)$ è una stima rumorosa del vero gradiente basata sulla d_i corrente.

ATTENZIONE: $\beta(i)$ è un processo stocastico (perché dipende dai dati che sono stocastici, i dati sono random).

Consiglio

SGD può tracciare drifts nel modello statistico sottostante perché può approssimare il gradiente da vere realizzazioni dei dati.

Prof:

Stocastic Gradient descent è più famoso del classico Gradient Descent in quanto Gradient Descent necessita della conoscenza del modello.

Invece di mettere il gradiente vero nella formula si mette il gradiente approssimato.

Notiamo sul grafico dei salti, perché? Si salta perché il gradiente che usiamo è falso visto che è basato su un solo campione (non si può neanche parlare di approssimazione visto che ci basiamo su un solo campione) e quindi il comportamento è più imprevedibile e ci sono salti.

Un fatto buono è che SGD è in grado di "trackare" drifts (deviazioni, cambi) nei modelli statistici, perché approssima il gradiente dalle realizzazioni dei dati.

Immaginiamo un problema di apprendimento online in cui stiamo collezionando i dati, se i dati cambiano di natura con un sistema statico che conosce la distribuzione è un problema perché la distribuzione sta cambiando (ovviamente il problema è se non lo sappiamo) e quindi l'algoritmo non funziona bene, invece l'algoritmo del gradiente stocastico si adatta in automatico al cambiamento della situazione. Si parla di online learning, sottointendendo che significa che "imparo" da dati che fluiscono continuamente, ci sono streaming data, dati che fluiscono continuamente. Si parla inoltre di adaptation, adattatività o adattività, cioè capacità di adattarsi. Questi due termini sono un po' sinonimi.

Questo algoritmo ha quindi il pregio che visto che impara direttamente dai dati dovrebbe essere in grado di cambiare se cambiano i dati. Il "dovrebbero" non è casuale ma è per prudenza che sarà spiegata a breve.

Facciamo un passo indietro: come fa questo algoritmo a convergere? Noi vogliamo convergere al minimizzartore della funzione **vera**, non a quello della funzione inventata.

Stochastic gradient descent: convergenza

Constant step-size

Sia β^* il vero minimizzatore.

Le iterate $\beta(i)$ di SGD rimbalzano intorno al minimizzatore senza mai raggiungerlo a causa del gradient noise (rumore di gradiente, è la differenza tra il gradiente vero e quello che calcoliamo noi).

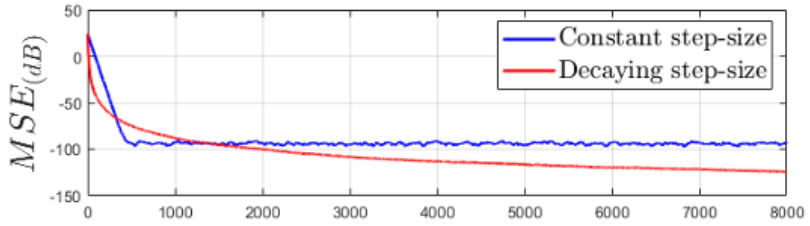
Per μ più piccolo di un certo valore (che dipende da δ e ν), per $i \to \infty$ le iterate $\beta(i)$ oscillano in un piccolo intorno di β^*.

Tecnicamente, $E[||\beta(i) - \beta^*||^2]$ converge ad un errore steady state (stato stazionario, cioè per i grande) $O(\mu)$ (l'errore è dell'ordine di μ, quindi se μ è scelto piccolo, l'errore sarà piccolo ma non 0), ad un rate geometrico $O(p^i)$, dove $p \in (0, 1)$ è una funzione decrescente di μ, e dipende da δ e ν.

Decaying step-size

Se $\mu = \mu(i)$ con $\sum_{i=1}^{\infty} \mu(i) = \infty$ e $\lim_{i \to \infty} \mu(i) = 0$, SGD converge, nel senso mean-square (quadratico medio), all'esatto minimizzatore.

Se $\mu(i) = \frac{\tau}{i}$, per $\tau > \frac{1}{\nu}$ il rate di convergenza è $O(\frac{1}{i})$



Prof:

Il fatto che l'errore cala e c'è un rimbalzo infinito intorno al vero minimo sembra miracoloso, nel grafico si vede un errore di -100dB, cosa c'è dietro? Come si fa a trovare il minimizzatore della funzione vera se noi stiamo usando una funzione falsa basata su un campione alla volta?

La funzione vera è definita con una media, la media la nostra stima non la fa perché usa un campione alla volta, eppure alla fine trova il minimizzatore della funzione vera. Non esistono miracoli, c'è un trucco nascosto: è come se la media su tutti i campioni la facesse progressivamente con la ricorsione: $\beta(i) = \beta(i-1) - \mu\widehat{\nabla J}_i(\beta(i-1))$, srotolando questa ricorsione somma tutti i gradienti e la media la fa a poco a poco, però anche se le iterazioni vanno all'infinito il valore del gradiente non lo butta e resta μ come limite che non può superare.

Abbiamo detto che l'errore è $O(\mu)$, cosa succede se facciamo tendere μ a 0? L'errore dovrebbe tendere a 0, l'algoritmo converge. Infatti con Decaying step-size l'SGD converge in errore quadratico (MSE) al minimizzatore vero. Decaying step-size però non converge a rate geometrico, tuttavia in questo caso resta migliore.

Nel caso GD, con tutto noto, non è vantaggioso usare Diminishing Step-size perché è più lento di constant step-size.

Nel caso stocastico, nonostante stiamo approssimando con un campione alla volta scopriamo che con constant step-size andiamo benino perché l'errore è piccolo, dell'ordine di μ che è piccolo, converge in un intorno, continua ad oscillare ma l'errore è piccolo, questo ci fa pensare che se facciamo tendere μ a 0 anche l'errore tende a 0 e infatti è vero ed è in questo senso che convergo al valore vero.

Specifichiamo "in the mean squared sense" (nel senso errore quadratico) perché in SGD c'è aleatorietà, in GD quando dicevamo che l'algoritmo convergeva i β erano deterministici e convergevano, in SGD invece c'è aleatorietà e bisogna fare la media dell'errore.

Quindi in SGD usiamo solo Decaying step-size? Notiamo dal grafico che sembra che Constant sia più veloce ad andare dove deve andare (del resto il rate è geometrico).

Inoltre Decaying step-size non performa bene in lunghi task online, come spiegato di seguito.

Stochastic gradient descent: step-size tuning

Le Decaying step-sizes permettono di convergere all'esatto minimizzatore ma non sono adatte a task online

L'algoritmo mostra una "memoria da elefante": non può reagire a drift (che ovviamente non si sa quando avvengono) nei dati visto che gli aggiornamenti sono annichiliti dal valore sempre più piccolo di $\mu(i)$, dopo un miliardo di iterazioni $\mu(i)$ vale tao su un miliardo e per reagire ad un drift ci vorrebbe un altro miliardo di iterazioni (sperando non ci siano altri drift nel mentre). In pratica converge ma se ci serve non "sconverge", mentre i salti costanti di constant step-size gli permettono di reagire meglio ai drift, al costo di non convergere mai (trade-off, compromesso).

L'algoritmo è più adatto ad applicazioni "batch" (con un dataset fisso che non cambia, magari senza grandi limiti computazionali) piuttosto che a task online.

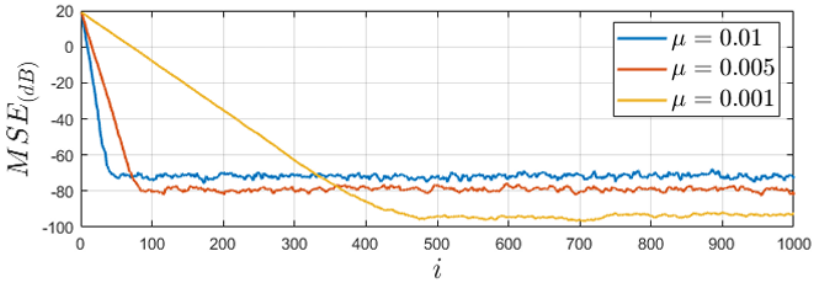
Prof:

Anche con decaying step-size la convergenza può esserci solo con infiniti campioni, se ne abbiamo n fissi giriamo sempre sugli stessi campioni e quindi non si può convergere al minimo vero.

Le Constant step-sizes sono adatte a task online, nonostante non convergano! (In un certo senso proprio perché non convergono)

Quanto più piccolo è μ tanto migliore è l'accuratezza (regioni più piccole intorno al minimizer, perché diventa più piccolo $O(\mu)$).

Piccoli valori di μ rallentano il rate di convergenza.



Prof:

Per dimensionare μ in maniera appropriata bisognerebbe avere un'idea dei tempi di variazione del fenomeno sottostante i dati.

La bellezza dell'applicazione online è che _idealmente_ si accende l'algoritmo e non va resettato più, va avanti all'infinito e capisce da solo quando cambia quello che sta stimando. Usando un algoritmo con memoria di elefante bisognerebbe trovare un modo per capire quando cambia la natura dei dati per resettarlo manualmente.

Ovviamente ci sono approcci mixed, banalmente possiamo lanciare un parallelo, un algoritmo che si adatta ed una altro che stima, così se non cambia niente prendiamo i dati da quello preciso e quando quello che si adatta capisce che è cambiato qualcosa si resetta il secondo. Dipende tutto dall'applicazione.

Un'altra implementazione fisica della SGD con Constant step-size è dire che è come se usasse una finestra mobile che gli permette di usare sempre gli ultimi dati più freschi (in modo da essere adattabile), più piccola è μ e più si allarga la finestra.

PARLA DELL'ESAME

Operativamente quando implementeremo questi algoritmi ci sarà detto o di scegliere...

Uno scritto plausibile potrebbe essere "Avete il problema di classificazione tizio, dovete implementare la regressione logistica, dovete minimizzare questa funzione, scegliete a seconda delle specifiche se è meglio un SGD Batch, mini-Batch, classico, scegliete il parametro μ, motivate ecc...". Si può anche fare validation, cioè se vogliamo scegliere il miglior valore di μ per raggiungere un certo target si può usare la tecnica della validation, quindi prendiamo il dataset, lo splittiamo usando una porzione di dataset per scegliere il miglior valore di μ per ottenere certi target.

Stochastic gradient descent: varianti

Mini-batch

Le varianti mini-batch calcolano il gradiente su un sottoinsieme S di samples scelti randomicamente.

$$\beta(i) = \beta(i-1) - \frac{\mu}{|S|}\sum_{i \in S} \nabla Q_\beta(d_i)$$

in modo tale da ridurre la varianza del rumore del gradiente per iterazione.

In questo modo si riduce la varianza del gradient noise per singola iterazione, cioè si fa una stima un po' migliore della funzione della singola iterazione, questa è una funzione di mezzo che dovrebbe funzionare meglio dell'SGD con un campione alla volta.

Adaptive step-size

Molti metodi sono stati sviluppati per impostare una step-size adattiva (ad esempio, AdaGrad, Adam, RMSProp).