Universitat Politècnica de Catalunya

Ciència i Enginyeria de Dades

Optimització Matemàtica: Pattern recognition with Single Layer Neural Network (SLNN)

Gerard Comas Quiles

May 2022

tr-seed = 39981721

te-seed = 12718993

sg-seed = 565544

Índice

1. Introducció

L'objectiu d'aquesta pràctica és crear una xarxa neuronal d'una única capa capaç de reconèixer nombres a partir d'una seqüència de dígits amb soroll afegit, un cas simplificat del conjunt de dades MNIST.

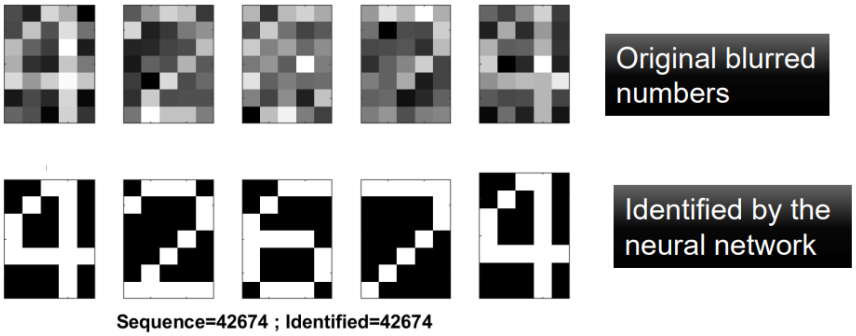

Figura 1: Imatge extreta de la presentació del lab fet per F.-Javier Heredia

Però tot això és una excusa per a poder treballar en un entorn d'optimització matemàtica. Aquest treball no serà sobre xarxes neuronals, encara que tractem amb elles, si no que es basarà a estudiar el funcionament de mètodes d'optimització.

Farem ús dels mètodes dels First Derivative Methods implementats, en concret el Mètode del Gradient (GM) i el mètode de Quasi-Newton amb BFGS (QNM). A més s'implementarà un de nou: el Stochastic Gradient Descent (SGD), molt popular en Machine Learning i en especial en Deep Learning.

Aquest document pretén analitzar el funcionament dels mètodes des de dues perspectives. En primer lloc, es realitza un estudi de la convergència analitzant els valors de la funció de pèrdua i, en segon lloc, un estudi de la precisió de reconeixement de la xarxa neuronal amb cada mètode, ja que, tot i que és important minimitzar la funció de pèrdua, pot no ser el factor més transcendent a l'hora d'obtenir una millor capacitat de reconeixement.

2. Estudi de la convergència

En aquesta primera part de l'informe analitzarem els resultats d'un primer batch d'execucions amb poques dades per tal de detectar quin és el millor algorisme i quina és la millor lambda de regularització en termes de convergència global i local.

2.1. Convergència Global

En primer lloc, analitzarem la convergència global a partir dels valors de la funció de pèrdua optimitzada i els compararem entre els diferents tres mètodes i diferents lambdes. En el següent gràfic podem veure quin és el valor de la funció de pèrdua de cada algoritme per a cada nombre quan lambda és 0.01:

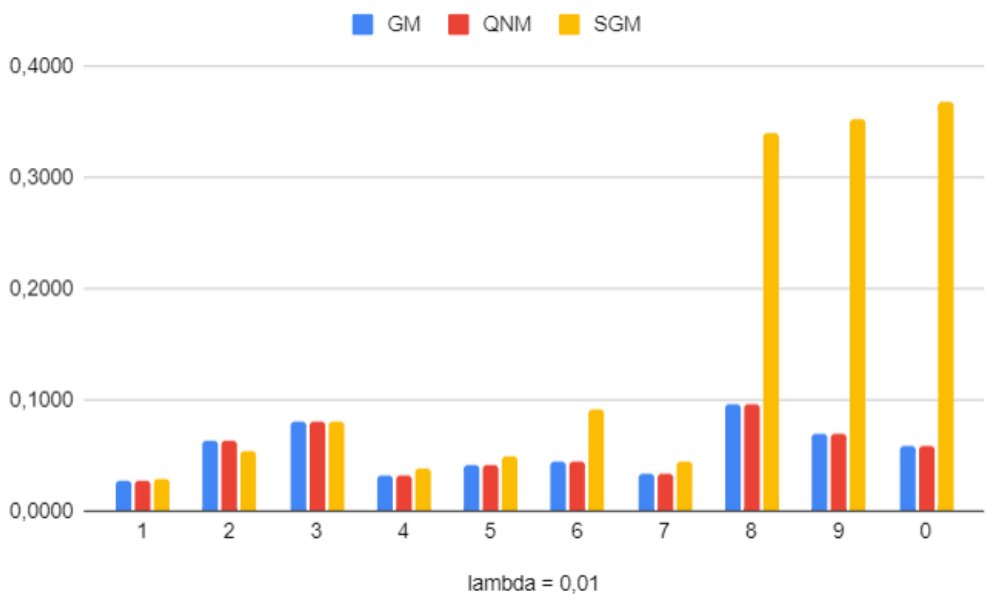

Figura 2: Histograma de la funció de pèrdua per a cada algoritme, per a lambda = 0.01

Com podem observar tots els valors de la funció de pèrdua es troben per sota de 0.4, per tant, en general són valors petits. Però en general, no passem de 0.1 a excepció de tres casos del SGM. Centrant-nos pròpiament en els algoritmes, el més destacable és que els valors òptims que aconsegueix el SGM són bastant diferents que els altres dos algoritmes, en general sempre més grans. A excepció del 2, en el qual troba un valor inferior als altres dos. Podem apreciar que el GM i el QNM consideren exactament els mateixos valors.

Però podrem obtenir conclusions més encertades si tractem la mitjana dels valors de la funció de pèrdua per a cada algoritme donada una lambda, ja que no ens importa gaire quin algoritme funciona millor per a un nombre en concret, volem saber quin convergeix millor en mitjana.

Lambda	GM	QNM	SGM
0	0.0168	0.0122	0.0108
0.01	0.0546	0.0546	0.1450
0.1	0.1409	0.1409	0.2538

Tal com dèiem, ara analitzarem la mitjana per a cada algoritme donada una lambda. Un fet destacable és que el mínim valor de la funció de pèrdua augmenta en funció de lambda. És fàcil de veure que per a lambdes diferents de zero el mètode del gradient (GM) i el Quasi-Newton (QNM) arriben al mateix valor, en canvi, el SGM en aquests casos pràcticament duplica el valor. Per a lambda zero aconseguim el contrari, el SGM és el que troba un valor més petit amb 0.0108. Cal destacar, que per lambda igual a zero el mètode QNM arriba a un valor mínim menor que el GM.

Si fem una avaluació general, podem dir que el QNM és el que té una millor convergència global, ja que és

el més consistent per a qualsevol lambda, seguit del GM. Amb tot i això, el mètode que arriba al valor més baix és el SGM per a lambda zero, però és molt poc consistent, per a lambda 0.01 i lambda 0.1 hi ha valors de la funció objectiu que s'allunyen molt de les que troben els altres dos algoritmes. A part del que ja hem vist en la Figura 2, per a lambda 0.1 hi ha una execució que realment pot perillar la convergència global de l'algoritme, en concret la del número 8, on la funció objectiu pren un valor de 0.9190, pràcticament 5 vegades més gran que la trobada pels altres algoritmes.

Per tant, podem concloure que la convergència global dels algoritmes GM i QNM està assegurada per qualsevol lambda que hem provat, però el SGM només podem considerar que convergeix globalment quan no hi ha regularització, és a dir, per lambda zero.

2.2. Convergència Local

En segon lloc, analitzarem la convergència local de cada algoritme i com afecta el valor de lambda. Per a aconseguir-ho mirarem el nombre d'iteracions i el temps d'execució de cada algoritme en mitjana. Per començar no tindrem en compte els diferents valors de lambda, per tal de veure les diferències en general.

Algoritme	Iteracions(niter)	Temps d'execució(tex)	Relació (tex/niter)
GM	210	1.3020	0.0116
QNM	47	0.8095	0.0181
SGM	14626	11.6677	0.0009

Com podem veure a la taula, l'algoritme que té una millor convergència local en mitjana és el Quasi-Newton Method, ja que tant el temps d'execució i les iteracions són les menors. Realment no existeix una gran diferència amb el Gradient Method, però aquest últim és una mica pitjor. Finalment, clarament, el SGM és el que té una pitjor convergència local, pel fet que, aproximadament, supera als altres algoritmes 10 cops en temps d'execució i te un nombre d'iteracions dos ordres superior.

Ara tornarem a mirar els valors mitjans, però donada una lambda. Començarem per la regularització nul · la:

Lambda = 0			
Algoritme	Iteracions(niter)	Temps d'execució(tex)	Relació (tex/niter)
GM	456	2.2443	0.0081
QNM	56	0.9407	0.0182
SGM	38576	30.8523	0.0012

El que més sorprèn d'aquesta taula són els valors tan grans comparant-los amb la taula anterior de les mitjanes. Això és degut al fet que no estem aplicant una regularització en la funció objectiu. Aquesta té un

pendent molt baix en l'entorn en el qual se situa el mínim i a cada iteració fa una passa molt petita. Per aquest motiu afegim la regularització que no deixa de ser una funció quadràtica que 'afegeix' convexitat a la funció i llavors els algoritmes són més eficients. Com podrem veure en les següents taules:

Lambda = 0.01			
Algoritme	Iteracions(niter)	Temps d'execució(tex)	Relació (tex/niter)
GM	135	0.9551	0.0084
QNM	49	0.8059	0.0168
SGM	2301	1.7950	0.0008

Lambda = 0.1			
Algoritme	Iteracions(niter)	Temps d'execució(tex)	Relació (tex/niter)
GM	39	0.7067	0.0184
QNM	35	0.6819	0.0195
SGM	3001	2.3557	0.008

Tal com havíem predit, el fet que ara hi hagi una regularització augmenta l'eficiència dels algoritmes, és a dir ara convergeixen més ràpidament. On trobem la diferència més gran és en el SGM, que passa de 38576 iteracions i un temps de 30.8523 segons a 2301 iteracions i un temps de 1.7950 segons. En el comportament per a diferents lambdes podem trobar diferents resultats depenent de l'algoritme, per al SGM en augmentar la lambda també augmenten les iteracions i temps d'execució. En canvi, els altres dos mètodes milloren a l'augmentar la lambda. Per tant, no podem concloure quina de les dues lambdes no nul · les és millor.

Tornant de nou als algoritmes, considerem que el QNM també serà el que tingui una millor convergència local, és a dir, sigui el més eficient. En conseqüència, podem concloure que el QNM és l'algoritme que millors convergències té per qualsevol lambda de les que hem treballat, a excepció de la convergència global per a lambda zero on el SGM obtenia els millors resultats.

Finalment, comentarem la relació entre el temps d'execució i el nombre d'iteracions per a cada algoritme. En les 4 taules representades aconseguim els mateixos resultats, el que té una relació més baixa és el SGM, seguit del GM i per últim el QNM. Això no és cap casualitat, té una explicació en la implementació de cada algoritme. El QNM és el pitjor en aquest aspecte, ja que per a cada iteració ha de calcular l'aproximació de la matriu Hessiana. Per l'latre part, el SGM és el que té una relació temps/iteració menor pel fet que per a fer els càlculs utilitza un subconjunt de les dades i, per tant, triga menys en fer els càlculs.

3. Estudi de la precisió de reconeixement

En aquesta segona part de l'informe analitzarem la precisió de reconeixement de la nostra xarxa neuronal amb un batch d'execucions amb una base de dades artificial molt més gran. El tamany de traning serà de 20.000 mostres i el mida de test de 2000 per cada nombre.

Per a fer l'estudi utilitzarem la lambda que ens hagi proporcionat uns millors resultats de predicció per l'apartat anterior, encara que també tindrem en compte el temps d'execució, ja que en tractar amb una base de dades molt més gran això pot ser clau.

En la següent taula podem veure quina lambda ens proporcionarà una millor predicció:

Lambda	GM	QNM	SGM	Mitjana
0	97.88 %	98.76 %	98.76 %	98.47 %
0.01	99.08 %	99.08 %	85.92 %	94.69 %
0.1	95.40 %	95.40 %	82.00 %	90.93 %

El resultat és que per a lambda igual a zero el percentatge de predicció és el més elevat. Té bastant sentit, ja que la lambda el que fa és modificar la funció per tal d'obtenir una millor convergència local. El que ens porta a un gran dilema, serà millor treballar amb una lambda no nul · la, tot i tenir pitjor nivell de predicció, que amb la lambda nul · la, que és molt més lenta? Considerarem que sí i farem el treball amb la lambda de 0.01, perquè dintre de les dues que tenim és la que millors resultats presenta.

Un cop decidida la lambda que usarem, caldrà fer pròpiament l'anàlisi de la precisió de reconeixement.

	GM	QNM	SGM
Precisió	99.06 %	99.06 %	99.13 %
Temps d'execució	49.5013	47.3651	104.5382

En aquesta taula podem observar la precisió i el temps d'execució per a cada algoritme. Veiem que no hi ha un clar guanyador: el més precís és el SGM amb un 99.13 % d'encert, en comparació al 99.06 % dels altres dos algoritmes, això implica que es fallen 174 números dels 20.000 que hi ha, 14 menys que amb els altres algoritmes; pel que fa a la velocitat, el SGM és el més lent amb diferència, amb un temps d'execució que dobla als altres algoritmes.

Per a matisar una mica més l'anàlisi, ens aturarem a estudiar que passa per a cada nombre i veure si un dels algoritmes destaca per sobre dels altres.

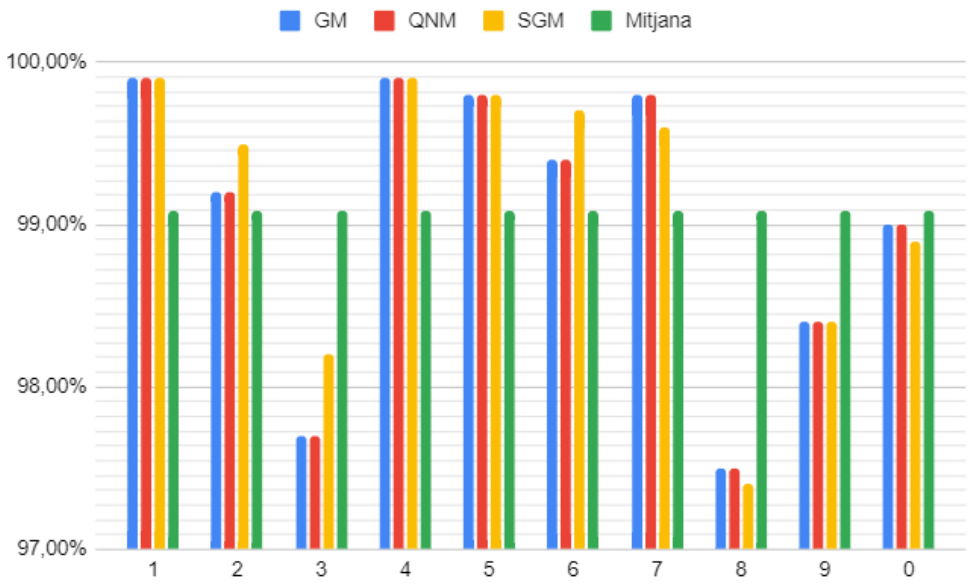

Figura 3: Precisió de cada algoritme per cada nombre en comparació a la mitjana total

El primer que cal comentar és que per a tots els nombres el GM i el QNM es comporten igual, això és degut al fet que arriben al mateix valor de la funció objectiu. Aquest gràfic ens podria haver servit per a determinar sí quan ens trobem amb un nombre 'difícil' de predir, és a dir, on el percentatge de predicció sigui baix en comparació a la mitjana, un dels algoritmes funciona millor que els altres. Però veiem que no és així. Pel número 3, el millor algoritme és el SGM, però pel 8 és el pitjor. Com a cosa curiosa, la dificultat en predir el 3 i el 8 pot ser deguda a la seva semblança.

Per tant, quant al millor algoritme per fer prediccions direm que és el SGM, encara que sigui per molt poc.

Ara estudiarem com varia el temps d'execució per a cada nombre i cada algoritme:

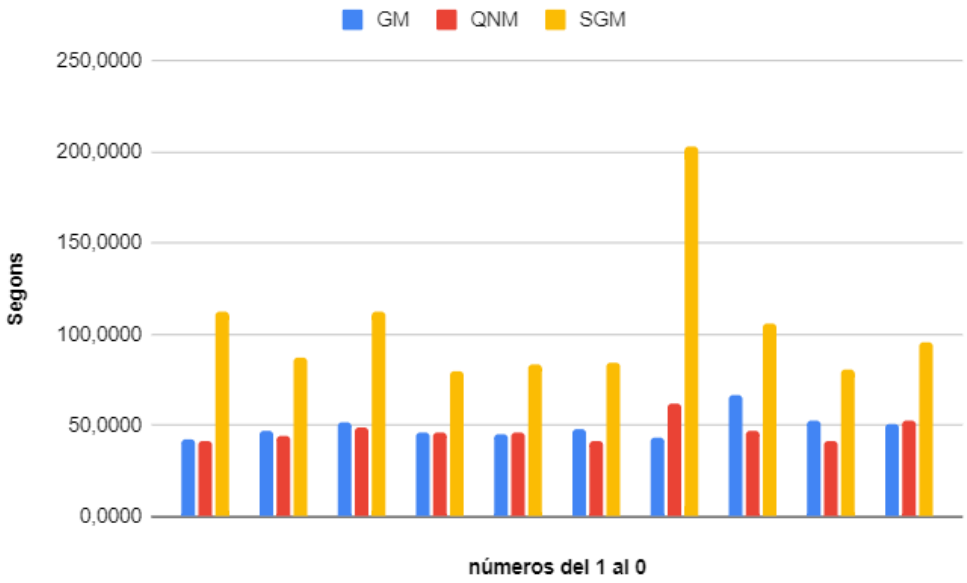

Figura 4: Histograma del temps d'execució per a cada nombre i algoritme

Aquí ja no tenim cap dubte, per a tots els nombres el SGM és més lent, fins a quadruplicar el temps per al número 7. El mínim increment de temps que té el SGM respecte dels altres dos és pel número 9 amb un augment de pràcticament el 70 %. Per tant, podem considerar que sempre serà significativa la diferència en el temps d'execució.

Llavors després d'aquesta anàlisi no hi ha una resposta clara, haurem de ponderar les prioritats que tenim: volem un algoritme considerablement més ràpid o un que sigui una mica més precís.

Per acabar el treball, intentarem raonar perquè l'algoritme que millor convergia (QNM), tant localment com global, no ha sigut el més precís. El que podem pensar és que això és causat a com actua el SGM i com està implementat. La clau d'aquest algoritme és que agafa només una part aleatòria de les dades per a calcular el gradient, això comporta un càlcul molt més baix per a cada iteració, però l'algoritme avança lentament i de forma incoherent, ja que no segueix la direcció del gradient. Això pot evitar quedar-nos estancats en un mínim local. A més, la convergència global està assegurada, sota certes condicions, a conseqüència del Teorema de Robbins–Siegmund. Llavors, podem plantejar-nos que en la funció de pèrdua hi hagués mínims locals.