

1 Setarea modelelor

1.1 Pasii pentru obtinerea de modele de predictie

Scopul cursului:

- determinarea faptului ca algoritmul de instruire a modelului functioneaza corect
- setarea hiperparametrilor
- impartirea setului de date in antrenare (train), validare (validation/dev), testare (test)

ML aplicat - proces iterativ. Exemplu de decizii de clarificat (iterativ) pentru o retea multistrat:

- learning rate
- numar straturi ascunse
- numar de neuroni
- functii de activare

Este imposibil de ghicit valorile corecte pentru acesti hiperparametri. Se poate urma un proces iterativ: idee - cod - experiment. Se doreste eficientizarea acestui proces.

1.2 Impartirea setului de date

Setul de date se partitioneaza in:

- set de antrenare - folosit pentru determinarea parametrilor modelului (e.g. ponderi)
- set de validare (validation/dev set) - folosit pentru alegerea din mai multe modele candidat
- set de testare - folosit pentru testarea modelului final

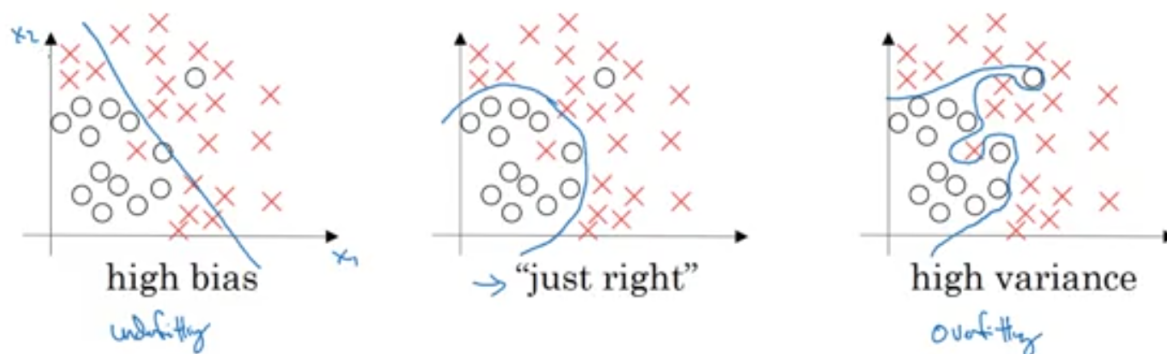
Procente de impartire: 60%/20%/20% (modele simple, date putine/medii). Pentru date multe (1M+ exemple) se prefera un dev set mai mic care sa fie usor de evaluat pentru deciderea asupra modelului; ex 10K date dev/1M date total; similar pentru setul de date de test -> 98% train/1% dev/1% test.

Atentie la modalitatea de strangere a datelor: daca dev sau test set sunt din alta sursa, atunci distributia lor difera -> antrenarea poate fi nefructuasa: modelul se instruieste "pe alte date" decat este validat sau testat. Trebuie sa ne asigura ca dev si test set provin din aceeasi distributie.

Este permis sa nu existe un set de testare, insa validarea trebuie facuta.

1.3 Bias si variance





Reprezentarea de mai sus e intuitiva, in general nu putem vizualiza suprafetele de separare. Se vor folosi alti indicatori pentru a discrimina intre cele 3 situatii.

Situatii posibile:

1. Modelul se prezinta foarte bine pe setul de antrenare (e.g. eroare=1%), dar are rezultate proaste pe setul de testare (eroare=11%) -> **overfit**, modelul nu generalizeaza bine; spunem ca modelul are **"high variance"**.
2. Modelul produce erori mari atat pe antrenare (15%), cat si pe validare (16%), in comparatie cu ce pot obtine alte modele, e.g. oameni (~ 0% eroare); altfel zis, modelul nu produce rezultate bune nici macar pe setul de instruire -> **underfitting ("high bias")**
3. Modelul produce 15% pe setul de antrenare, 30% pe setul de validare, in conditiile in care o rata de eroare obtinuta de alte modele e <1% -> **high bias** deoarece nu performeaza bine pe setul de antrenare si **high variance** deoarece nu produce rezultate bune pe setul de testare.
4. 0.5% eroare pe antrenare, 1% pe setul de dev -> **low bias, low variance**

Pentru a reduce din fenomenele 1-3 anterioare:

1. high bias (underfitting pe train dataset) -> retea mai mare (nr mai mare de straturi ascunse, mai multi neuroni etc.), antrenare mai lunga, algoritmi de optimizare mai sofisticati (ADAM in loc de SGD)
2. dupa reducerea de bias, daca inca exista high variance (performanta slaba pe dev set): mai multe date de antrenare; regularizare; modificarea arhitecturii modelului (e.g. reducerea numarului de neuroni)

Se repeta pasii de mai sus pana se obtine un model cu low bias, low variance.

▼ 1.4 Regularizarea de modele

Regularizarea: metoda de a reduce varianta unui model, daca aducerea de mai multe date in setul de instruire nu e fezabila.

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \|w\|_2^2$$

Regularizarea L_2 : $\|w\|_2^2 = w^t w$

Regularizarea L_1 : $\|w\|_1 = \sum |w_i|$ -> vectorul w astfel regularizat va fi rar = cu o multime de valori 0.

Pentru cazul in care vorbim de matricele de ponderi dintr-o retea neurala W_1, \dots, W_{L-1} , termenul de regularizare este:

$$\frac{\lambda}{2m} \sum_{l=1}^{L-1} \|W^{[l]}\|_F^2$$

λ este un alt hiperparametru, a carui valoare potrivita se poate determina pe setul de dev.

Daca λ este foarte mare, se va ajunge la multe ponderi care sunt foarte aproape de 0, deci de fapt la o retea neurala mai simpla, deci mai putin dispusa la overfitting.

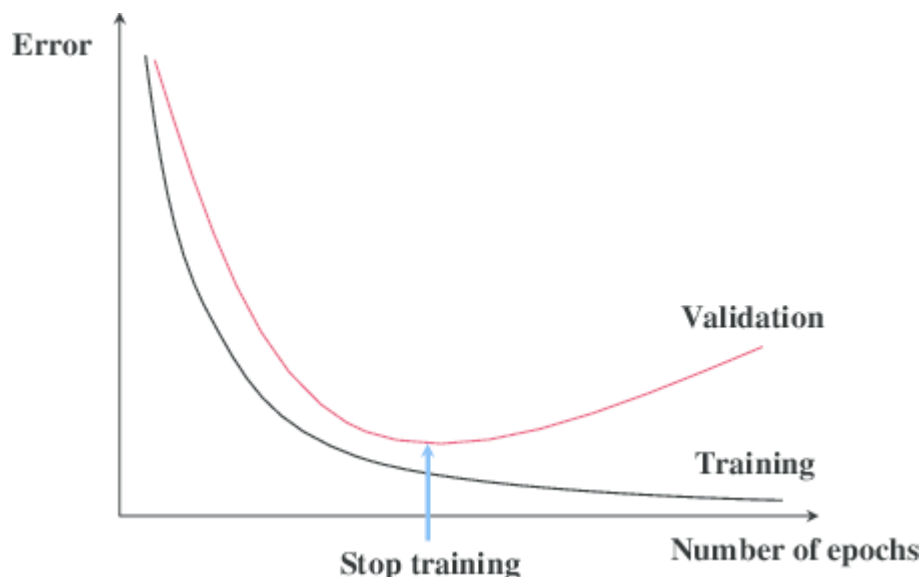
O alta metode de regularizare este dropout - bibliografie.

In afara de reducerea complexitatii retelei, uneori se pot augmenta datele din setul de antrenare. De exemplu, pentru imagini:

- flip orizontal de imagine (stanga <-> dreapta; atentie sa nu se produca exemple din alte clase)
- distorsiuni



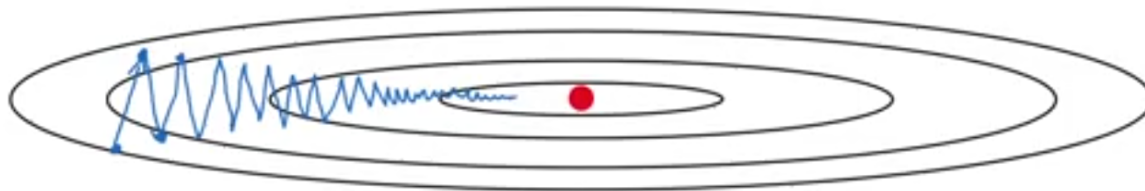
O alta metoda este "early stopping": se reprezinta valorile functiei de eroare pe setul de antrenare si de validare. Cand eroarea pe setul de validare creste, se opreste instruirea. Altfel zis, numarul de iteratii de instruire e un hiperparametru care se determina pe baza comportamentului pe setul de validare (dev set). Ca tehnica, insa, este mai putin recomandata decat regularizarea.



1.5 Aspecte practice

1.5.1 Normalizarea datelor

În cazul în care datele nu sunt pe aceeași scală, iar pornirea căutării se face dintr-o inițializare aleatoare a ponderilor (de regulă: da), există riscul de a începe căutarea dintr-o poziție îndepărtată de minimul (minimele) funcției obiectiv:



Pasi:

1. Datele se centrează în 0, scăzând din fiecare dimensiune media pe acea dimensiune
2. Se calculează varianța, se împart valorile la deviația standard - pe fiecare dimensiune

În urma operațiilor datele vor avea medie 0 și deviație standard 1.

Efect: din date elongate se ajunge la date într-un nor Gaussian. Alternativ, se poate folosi scalarea datelor. După această transformare, se poate folosi un coeficient de învățare (learning rate) mare. Dacă datele sunt în intervale de lungimi diferite, e nevoie să se folosească un LR mic → învățare lentă.

În sklearn există clase de preprocesare care pot fi depuse într-un pipeline.

1.5.2 Inițializarea ponderilor rețelei

Pentru ca valoarea produsului scalar dintre ponderi și intrări să nu crească foarte mult, valorile inițiale ale ponderilor trebuie să fie mici: $z = b + w_1 x_1 + \dots + w_n x_n$. Se va seta varianța valorilor vectorului w să fie $2/n$; a se vedea inițializările de tip Glorot și Kaiming.

1.6 Minibatch gradient descent

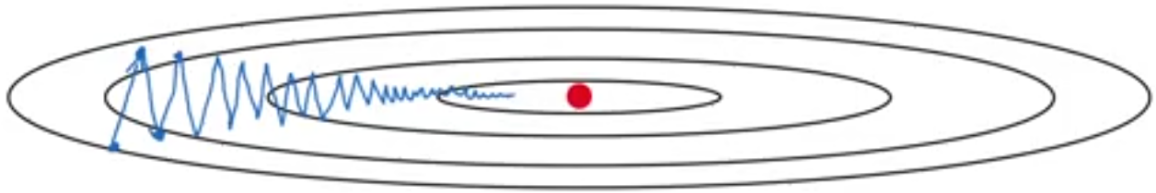
Premisa: cod vectorizat, viteză marită de calcul. Dacă se face vectorizarea pe întregul set de date (mare), atunci trebuie așteptat relativ mult pentru a avea un gradient calculat. Alternativa este de a lăsa algoritmul să calculeze niște gradienti pe o parte din date, pentru a începe să facă mai rapid modificări pe ponderi.

Rezolvare: se partitionează setul de antrenare în mini-loturi (minibatches); se calculează gradientii corespunzători și se fac modificările indicate de algoritm. Se trece la alt minibatch, procedeul se repetă până la epuizarea minibatchurilor = o epocă.

Pentru un minibatch, gradientii datelor continuate se mediază.

1.7 Gradient descent cu momentum

Se pot reduce din oscilațiile care apar în timpul unei antrenări cu GD:



Pe axa verticală se dorește o învățare mai lentă, iar pe cea orizontală - învățare mai rapidă.

Se calculează derivatele parțiale ale ponderilor pe minibatch-ul curent. Se calculează apoi:

$$v_{dw} = \beta v_{dw} + (1 - \beta)dw$$

unde inițial $v_{dw} = 0$. Modificarea aponderilor se face cu:

$$w = w - \alpha v_{dw}$$

Rezultat: oscilațiile se diminuează. Hiperparametrul β se ia de regulă în intervalul $[0.9, 1]$.

▼ 1.8 Metoda RMSProp

RMSProp: root mean squared prop.

Ca și la varianta cu momentum, scopul este reducerea oscilațiilor și atingerea mai rapidă a minimumului. La fiecare iterație (minibatch):

- se calculează derivatele dw, db pentru funcția de eroare
- se calculează $s_{dw} = \beta s_{dw} + (1 - \beta)dw^2$, unde ridicarea la pătrat se face pe componentele vectorului
- se calculează $s_{db} = \beta s_{db} + (1 - \beta)db^2$
- se actualizează ponderile:

$$w = w - \alpha \frac{dw}{\epsilon + \sqrt{s_{dw}}}, b = b - \alpha \frac{db}{\epsilon + \sqrt{s_{db}}}$$

unde $\epsilon > 0$ e o constantă mică, folosită pentru a evita împărțirea la 0.

▼ 1.9 Algoritm de optimizare Adam

Adam (Adaptive moment estimation) este un alt algoritm utilizat pentru optimizarea ponderilor din rețeaua neurală. Combina metodele momentum și RMSProp.

Pasii algoritmului:

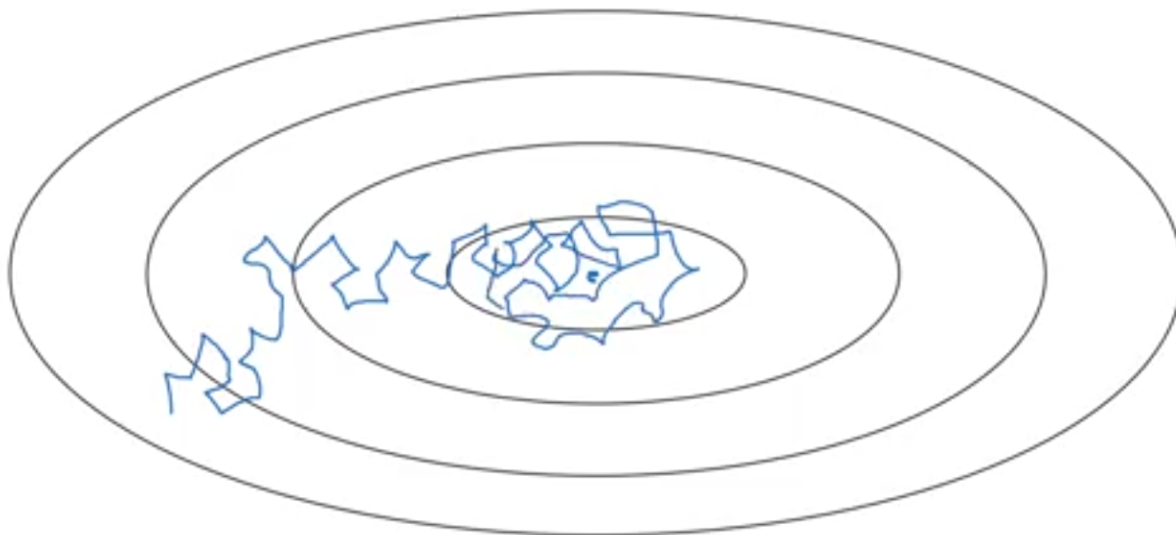
- $v_{dw} = 0, s_{dw} = 0, v_{db} = 0, s_{db} = 0$
- La iterația t :
 - calculează dw, db - derivatele funcției de eroare folosind mini batch-ul curent
 - $v_{dw} = \beta_1 v_{dw} + (1 - \beta_1)dw, v_{db} = \beta_1 v_{db} + (1 - \beta_1)db$
 - $s_{dw} = \beta_2 s_{dw} + (1 - \beta_2)dw^2, s_{db} = \beta_2 s_{db} + (1 - \beta_2)db^2$
 - $v_{dw}^{corrected} = \frac{v_{dw}}{1 - \beta_1^t}, v_{db}^{corrected} = \frac{v_{db}}{1 - \beta_1^t}$
 - $s_{dw}^{corrected} = \frac{s_{dw}}{1 - \beta_2^t}, s_{db}^{corrected} = \frac{s_{db}}{1 - \beta_2^t}$
 - $w = w - \alpha \frac{v_{dw}^{corrected}}{\epsilon + \sqrt{s_{dw}^{corrected}}}, b = b - \alpha \frac{v_{db}^{corrected}}{\epsilon + \sqrt{s_{db}^{corrected}}}$

Hiperparametri:

- α trebuie sa fie gasit prin cautare
- $\beta_1 = 0.9$
- $\beta_2 = 0.999$
- $\varepsilon = 10^{-8}$

▼ 1.10 Modificarea ratei de invatare

Atunci cand se foloseste minibatch gradient descent se intampla ca valoarea functiei de eroare sa "se plimbe" in jurul unui minim, fara ca sa il atinga de fapt:



Acest lucru se poate datora unei rate de invatare constante. Ar fi indicat ca aceasta sa fie mare la inceput, pentru a face rapid pasi in directia minimului, dar apoi sa scada, pentru a permite o cautare mai fina.

O modalitate de modificare a valorii ratei de invatare in decursul epocilor este:

$$\alpha = \frac{1}{1 + \text{decay_rate} \cdot \text{epoch_number}} \cdot \alpha_0$$

sau o scadere exponentiala:

$$\alpha = 0.95^{\text{epoch_number}} \cdot \alpha_0$$

sau

$$\alpha = \frac{k}{\sqrt{\text{epoch_number}}} \cdot \alpha_0$$

In []: