Setarea modelelor

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:

- numar straturi ascunse
- numar de neuroni
- learning rate
- 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.

Impartirea setului de date

Setul de date se partitioneaza in:

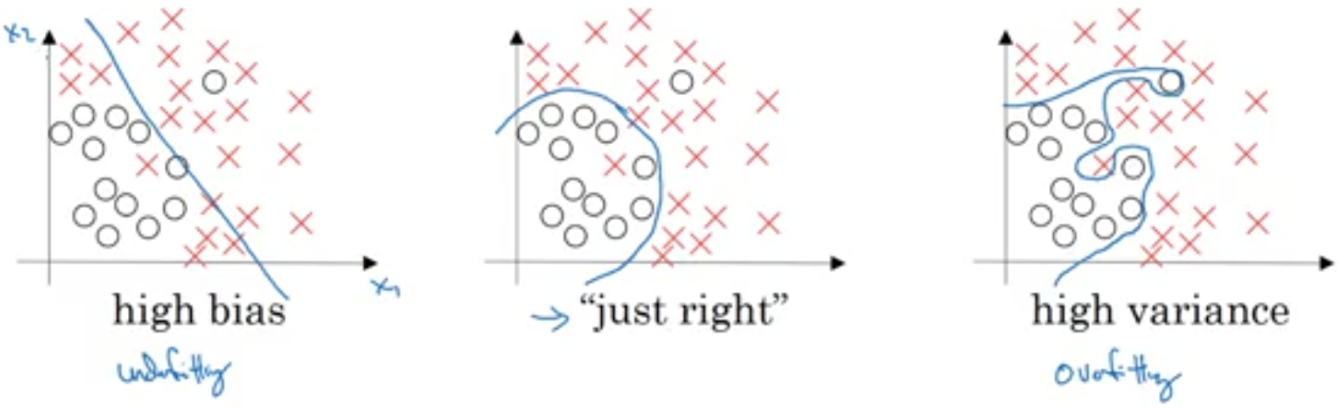
- set de antrenare - folosit pentru determinarea parametrilro modelului (e.g. ponderi)
- set de validare (validation/dev set) - folosit pentru alegerea din mai multe modele candidat
- set de testare - folosit pentru testarea modelului final

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 surse, 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.

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. reducera numarului de neuroni)

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

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 $W_1, \ldots, W_{L-1}$, 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.

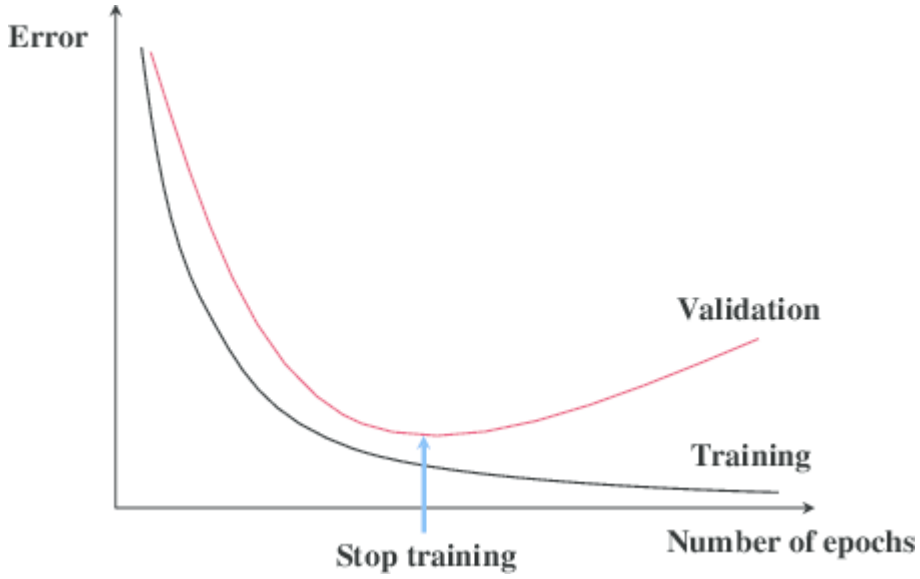
Daca λ este foarte mare, se va ajunge la multe ponderi care sunt foarte aproape de 0, deci de fapt la o retea neurala mai simpla, deci mai putin dispusa la overfitting.

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

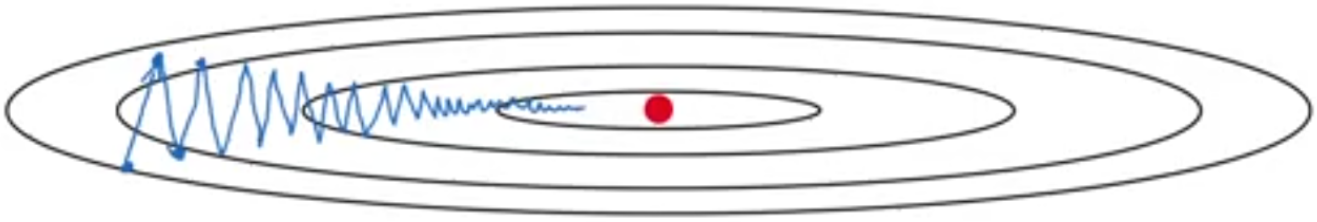
O alta metoda este "early stopping": se reprezinta valorile functiei de eroare pe setul de antrenare si de validare. Cand eroarea pe setul de validare creste, se opreste instruirea. Altfel zis, numarul de iteratii de instruire e un hiperparametru care se determina pe baza comportamentului pe setul de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



Aspecte practice

Normalizarea datelor

In cazul in care datele nu sunt pe aceeasi scala, iar pornirea cautarii se face dintr-o initializare aleatoare a ponderiloe (de regula: da), exista riscul de a incepe cautarea dintr-o pozitie indepartata de minimul (minimele) functiei obiectiv:



Pasi:

1. Datele se centreaza in 0, scazand din fiecare dimensiune media pe acea dimensiune
2. Se calculeaza varianța, se impart valorile la deviatia standard - pe fiecare dimensiune

In urma operatiilor datele vor avea medie 0 si deviatie standard 1.

Efect: din date elongate se ajunge la date intr-un nor Gaussian. Alternativ, se poate folosi scalarea datelor. Dupa aceasta transformare, se poate folosi un coeficient de invatare (learning rate) mare. Daca datele sunt in intervale de lungimi diferite, e nevoie sa se foloseasca un LR mic -> invatare lenta.

In sklearn exista clase de preprocesare care pot fi depuse intr-un pipeline.

Initializarea ponderilor retelei

Pentru ca valoarea produsului scalar dintre ponderi si intrari sa nu creasca foarte mult, valorile initiale ale ponderilor trebuie sa fie mici: $z = b + w_1x_1 + \cdots w_nx_n$. Se va seta varianta valorilor vectorului w sa fie $2/n$; a se vedea initializarile de tip Glorot si Kaiming.

Minibatch gradient descent

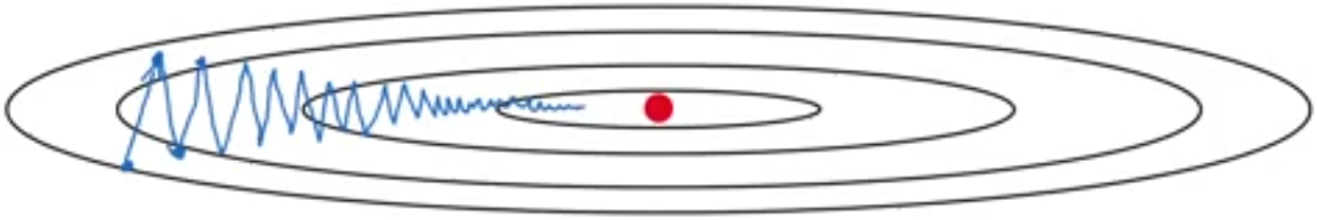
Premisa: cod vectorizat, viteza marita de calcul. Daca se face vectorizarea pe intregul set de date (mare), atunci trebuie asteptat relativ mult pentru a avea un gradient calculat. Alternativa este de a lasa algoritmul sa calculeze niste gradienti pe o parte din date, pentru a incepe sa faca mai rapid modificari pe ponderi.

Rezolvare: se partitioneaza setul de antrenare in mini-loturi (minibatches); se calculeaza gradientii corespunzatori si se fac modificarile indicate de algoritm. Se trece la alt minibatch, procedeul se repeta pana la epuizarea minibatchurilor = o epoca.

Pentru un minibatch, gradientii datelor continute se mediaza.

Gradient descent cu momentum

Se pot reduce din oscilatiile care apar in timpul unei antraneri cu GD:



Pe axa verticala se doreste o invatare mai lenta, iar pe cea orizontala - invatare mai rapida.

Se calculeaza derivatele partiale ale ponderilor pe minibatch-ul curent. Se caculeaza apoi:

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

unde initial $v_{dw} = 0$. Modificare aponderilor se face cu:

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

Rezultat: oscilatiile se diminueaza. Hiperparametrul β se ia de regula in intervalul $[0.9, 1]$.

Metoda RMSProp

RMSProp: root mean sqared prop.

Ca si la varianta cu momentum, scopul este reducerea oscilatiilor si atingerea mai rapida a minimului. La fiecare iteratie (minibatch):

- se calculeaza derivatele de, db pentru functia de eroare
- se calculeaza $s_{dw} = \beta s_{dw} + (1 - \beta) dw^2$, unde ridicarea la patrat se face pe componentele vectorului
- se calculeaza $s_{db} = \beta s_{db} + (1 - \beta) db^2$
- se actualizaeaza ponderile:

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

  unde $\varepsilon > 0$ e o constanta mica, folosita pentru a evita impartirea la 0.

Algoritmul de optimizare Adam

Adam (Adaptive moment estimation) este un alt algoritm utilizat pentru optimizarea ponderilor din reteaua neurala. Combina Metodele moemntum si RMSProp.

Pasii algoritmului:

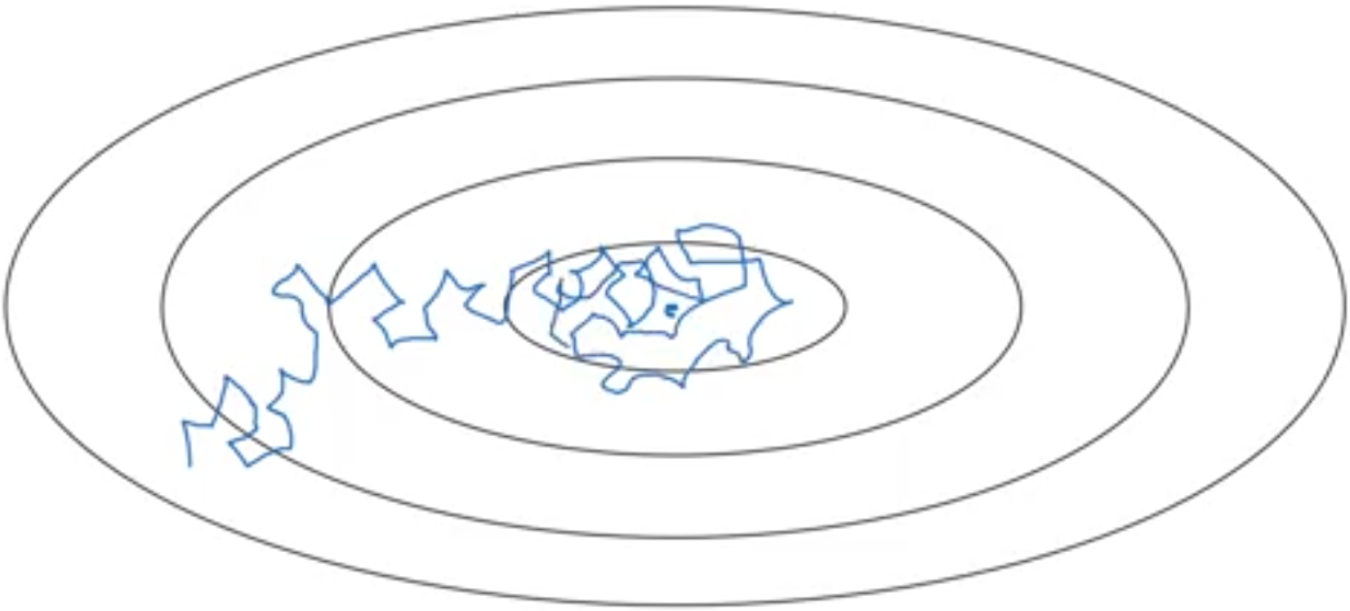
- $v_{dw} = 0$, $s_{dw} = 0$, $v_{db} = 0$, $s_{db} = 0$
- La iteratia t:
  - calculeaza dw, db - derivatele functiei de eroare folosind mini batch-ul curent
  - $v_{dw} = \beta_1 v_{dw} + (1 - \beta_1) dw$, $v_{db} = \beta_1 v_{db} + (1 - \beta_1) db$
  - $s_{dw} = \beta_2 s_{dw} + (1 - \beta_2) dw^2$,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

Hiperparamentri:

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

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:

$$\alpha = \frac{1}{1 + decay\_rate \cdot epoch\_number} \cdot \alpha_0$$

sau o scadere exponentiala:

$$\alpha = 0.95^{epoch\_number} \cdot \alpha_0$$

sau

$$\alpha = \frac{k}{\sqrt{epoch\_number}} \cdot \alpha_0$$

In [ ]: