# **08 - LARGE DATASETS AND BIG MODELS**

Ci si è sempre chiesti se grandi quantità di dati aiutino i task di ML. Non è sempre chi ha l’algoritmo migliore che vince, ma piuttosto chi ha più dati.

### **Bias/Varianza, modello e dati**

Chart, scatter chart

Description automatically generated

Assumendo che la feature x abbia informazioni a sufficienza per poter predirre y accuratamente, un test utile da fare è di verificare se un umano esperto riesca a predire correttamente y dato l’input x.

Usando un algoritmo di predizione con diversi parametri si ottiene basso bias mentre usando un grosso training set si avrà bassa varianza. La cosa più semplice che verrebbe in mente è di unire i due modelli.

Però bisogna sempre tener conto del trade off che c’è.

#### Bias/Variance (a tradeoff?)

A picture containing table

Description automatically generated

(immagino errore umano circa 1%)

Sembra non essere esattamente un tradeoff.

Bisogna ricordarsi dell’ortogonalizzazione per ottenere buoni risultati, quindi bisogna agire in contemporanea sia su una che sull’altra regolando di volta in volta.

Diagram

Description automatically generatedIl workflow per regolare le cose è il seguente:

#### **Regolarizzazione delle reti neurali**

Regressione lineare/logistica con le due notazioni

Text

Description automatically generated

→ per smorzare i pesi dei parametri vado ad abbassare il contibuto della parte azzurra

**Per le NN uso i pesi w e non i parametri θ.**

Il termine di regolarizzazione per le reti neurali viene scritto come norma euclidea di w ottenendo una “L2 regularization”.   
Esiste anche la “L1 regularization” dove non si ha più il quadrato e quindi c’è meno smorzamento.

*Però in questa scrittura non si tiene ancora conto dei livelli della rete([j]).*

W è il **peso dello specifico livello** che stiamo considerando.

Diagram

Description automatically generated

La nuova regolarizzazione è nella forma della norma di **frobenius**. Il nostro obiettivo è quello di **minimizzare** tale norma.

Se si vanno a smorzare il contributo di quei pesi che se non smorzati impediscono o limitano la capacità della funzione di costo di essere minimizzata (quindi all’algoritmo di convergere).

Nella parte più vicina all’origine, la funzione tende ad essere lineare, quando cerchiamo di abbassare i pesi dei nostri parametri, cerchiamo di abbassare il contenuto della nostra relazione linare, muovenoci intorno all’origine. In questa zona, l’attivazione (applicazione della tangente iperbolica alla z) ha un’uscita più lineare. Dunque, se cerchiamo di abbassare i pesi, la nostra rete si comporta più in un regime di maggior linearità, dunque tende meno ad essere flessibile e ad overfittare.

Chart, line chart

Description automatically generated

**Regolarizzazione e gradient descent**

Graphical user interface, diagram, text

Description automatically generated

Come varia il **gradient descent con la regolarizzazione**: aggiornamento dei pesi.

Con la regolarizzazione si ottiene la seguente derivata della funzione di costo

Il passo di regolarizzazione va di volta in volta smorzare il valore dei pesi per questo si chiama “**weight decay”.**

#### **Dropout**

Tecnica di ottimizzazione che imposta delle probabilità dei vari livelli e decide se attivare o disattivare dei nodi.

Diagram

Description automatically generated

→ 0.5: probabilità che il neurone venga eliminato

Quindi solo alcuni si spengono ottenendo una rete più semplice che impara qualcosa di meno sofisticato. È una regolarizzazione perché ad ogni passo si ha sempre una rete semplice diversa che verrà addestrata. (diversa in quanto verranno buttati fuori dei neuroni diversi)

Diagram

Description automatically generated

Si addestra con il dropout e poi si disabilita.

#### Data augmentation

È una tecnica che si usa per ridurre l’overfitting per aiutare l’algoritmo a generalizzare sui dati nuovi.

* Effettuo delle leggere distorsioni sulle immagini in modo tale da evitare overfitting
  + Inverto orrizzontalmente immagine
  + Allargo immagine
  + Cambio colori (mantenendo sempre colori realistici)

A cat with its mouth open

Description automatically generated

Quindi allo stesso tempo si aumentano i dati e si riduce l’overfitting.

#### Batch vs Mini-batch gradient descent

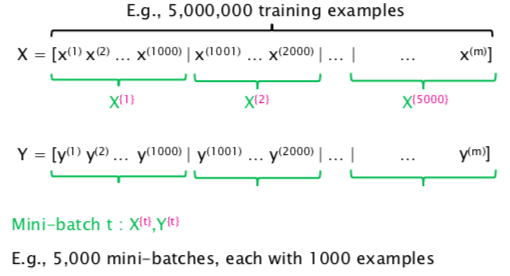
Il gradient descent prende la superficie multidimensionale che rappresenta il mio costo e partendo da un punto si inizia a scendere verso la direzione più ripida però per fare un passo devo fare il calcolo su tutto il dataset.

Graphical user interface, text, application

Description automatically generated

Questo tipo di algoritmo viene chiamato batch perché si processa tutto il training set.

Per velocizzare si potrebbe pensare di **prendere dei “mini-batch” ovvero delle porzioni del training set e per ognuno effettuo un passo di gradient descent** (discesa dalla collinetta).



La velocità con cui ci si avvicina alla soluzione è minore però funziona lo stesso.

* Percorso meno diretto

**L’algoritmo** si scrive così:

Graphical user interface, text, application

Description automatically generated

Per ogni passo di addestramento (fatto sul t-esimo mini-batch), faccio la forward propagation. Da qui trovo le attivazioni e gli output e la funzione di costo dove m nella sommatoria è la dimensione del mini-batch.

Graphical user interface, text, application, email

Description automatically generatedFaccio la backpropagation aggiornando i pesi.

Ripeto per ogni mini-batch.

Una **epoca** è un passaggio attraverso tutto il training set.

Ovviamente c’è un tradeoff anche per questa tecnica

Chart, histogram

Description automatically generated

A volte la funzione di costo scende altre no in base alla semplicità mini-batch. Per capire l’andamento bisogna andare a campionare ogni tot passi.

**Mini-batch size**

Diagram

Description automatically generated

→ direzioni completamente diversa perché ognuno dei dati causa una discesa randomica (potrebbe anche non portare ad una convergenza)

Alcune regole empiriche:

Text

Description automatically generated

* Più i batch sono piccoli, meno si sfrutta la vettorizzazione

## Il problema dei minimi locali

A close up of a colorful umbrella

Description automatically generatedIn realtà quando si pensa ai minimi locali si pensa ai **punti di sella** che non rappresentano un vero problema perché non tutte le dimensioni hanno derivata zero lì, quindi il gradient descent troverà un altro lato da cui scendere. È molto improbabile finire in un minimo locale perché li tutte le derivate parziali sono pari a 0.

Shape

Description automatically generatedIl problema che si può è quello con le curve, i plateaus, dove si ha la tangente molto vicina allo zero (derivata molto piatta) 🡪 il learning diventa molto lento.   
Il mini-batch gradient descent viene in aiuto.

### **Exploding/Vanishing gradients**

Vogliamo una rete molto profonda.

Immaginiamo di buttare via il bias (b=0) e di non applicare nessuna funzione di attivazione (g(z)=z).

A picture containing diagram

Description automatically generatedSe io volessi l’uscita di questa rete dovrei prendere l’input, moltiplicarlo per i pesi del primo livello, moltiplicarlo con i pesi del secondo livello, … fino all’utimo livello.

Con i pesi > 1 **i valori tendono ad esplodere** (sia le attivazioni che i gradienti) crescendo esponenzialmente 🡪 exploding gradient 🡪 **divergo**

Con i pesi < 1 accade il contrario, cioè che **decrescono** esponenzialmente con L 🡪 vanishing gradient 🡪 **rallento**

#### **Inizializzazione con deep networks**

Diagram, schematic

Description automatically generatedUna soluzione parziale sta nell’inizializzazione.

* Immaginiamo di avere n input.
* Si fa in modo che la varianza dei pesi sia uguale a 1/n.
  + Dunque, i pesi non saranno troppo lontani da 1 per evitare l’exploding o il vanishing gradients o che per lo meno non lo facciano troppo in fretta.

**Un’altra soluzione** con le reti a più livelli sta nel prendere i pesi, usare dei valori casuali e moltiplicarli per un fattore che considera il livello precedente e assegnare tale valore al peso.

* Inizializazione copn tutti a 0 non funziona
* Inizializzazione con tutti i pesi tra - ϵ e +ϵ funziona ma ha dei problemi quando le reti crescono molto in profondità

##### Inizializzazione di Xavier:

Diagram, schematic

Description automatically generatedPer le ReLU è meglio usare 2 anziché 1 nel rapporto sotto radice

Immagine che contiene testo, Carattere, simbolo, logo

Descrizione generata automaticamente

### **Normalizzare gli inputs**

Bisogna normalizzare le scale delle features se sono troppo differenti.

L’obiettivo è di avere media 0 e viarianza 1:

* Sottraggo la media della features μ di tutto il training set
* Divido per la deviazione standard di tutto il training set

### **Normalizzare le attivazioni**

Sembra sensato normalizzare anche gli input degli hidden layer, ovvero le attivazioni del livello precedente. Perché, se arriva qualcosa di sballato poi si propaga per tutta la rete. (normalizzando le attivazioni, la rete si comporta meglio).

In realtà si normalizza Z e si parla di **batch normalization**.

Diagram

Description automatically generated

→ +ϵ fa in modo che questa quantità non tenda a 0, rendendo l’algoritmo più stabile

→ moltiplica per Y e somma per n, ricalcolando quindi i valori delle attivazioni per ottenere valori ottimali di media e varanza

Iperparametri aggiuntivi: ϵ Y n

L’effetto è quello di rendere i livelli interni più robusti ai cambiamenti dei pesi dei layer precedenti. Inoltre, si ha anche un effetto di regolarizzazione

## **RETI NEURALI E MULTICLASS CLASSIFICATION**

Immaginiamo di avere un problema del genere con 4 classi

Diagram

Description automatically generated

La rete ha bisogno di 4 unità di uscita.   
Questo si può interpretare come multi-regressioni logistiche.

Nell’ultimo livello in questi problemi si mette un livello di **softmax**. La somma delle probabilità in uscita deve essere 1.   
È “soft” perché ci sono dei valori reali. Si parlerebbe di hardmax se si avesse un output con valore 1 e tutti gli altri 0.

**Softmax layer**

A rete andrà a normalizzare, come dividisione per l’average, le attivazioni dell’ultimo livello.

Text

Description automatically generated with medium confidence

#### **Training with softmax**

La funzione di loss diventa la seguente che si può ricondurre a quella delle regressione logistica quando si hanno solo due classi.

Text

Description automatically generated

Diagram

Description automatically generatedAbbiamo creato il vettore di X che è in realtà una matrice dove ogni colonna sono i vettori di training example. Il vettore Y conteneva solo per ogni 0 o 1 per ogni example.

Ora Y diventa una matrice dove ogni **colonna** **dice per ogni training example quale classe è stata predetta**.

## **TRANSFER LEARNING**

Trasferire qualcosa che abbiamo imparato in un determinato task, su un certo insieme di dati, ad altri dati/task.

Quello che si è scoperto quando si hanno grandi modelli e tanti dati si può usare la capacità della rete di svolgere il compito e applicarla ad un altro dominio.

Diagram

Description automatically generated

Abbiamo la rete che ha imparato bene il suo task.

Buttiamo via l’ultimo livello (o qualcosa in più) e lo sostituisco con un altro che non ha dei parametri e poi faccio l’addestramento solo dell’ultimo layer.

Diagram

Description automatically generated

*Ovviamente si possono buttare anche più livelli in base a quale livello di dettaglio mi servono le features imparate dalle rete già addestrata.*

Il transfer learning ha senso quando si ha lo stesso tipo di input e quando avendolo addestrato con molti dati lo applico a pochi dati di un altro dominio ottenendo buoni risultati. Questo perché le features che ci sono nei primi livelli vanno bene per imparare il nuovo task.

Questo si può fare perché i primi livelli, iniziano a capire gli spigoli e pian pinao cose più elevate.