Advanced Machine Learning

Sommario

Il corso si concentra sull'uso di reti neurali (*deep learning*). L'esame consiste in un progetto finale di gruppo più la valutazione degli *assignment* svolti singolarmente durante l'anno per i frequentanti, altrimenti in un esame orale. Del progetto sarà valutata la relazione finale, in cui saranno specificati i compiti svolti dai singoli partecipanti, secondo delle specifiche definite.

Indice

Parte I
Introduzione

L'operazione di *learning* si effettua per ottimizzare le performance di un algoritmo basandosi su un criterio prestabilito e una serie di esempi passati (esperienza) tramite *reverse engeniering*. Se l'operazione è definibile con un insieme di regole e la complessità non è particolarmente alta, non è richiesto l'uso del *machine learning*; mentre quando le soluzioni cambiano (e il sistema si deve adattare) o la complessità è particolarmente alta, è necessario l'uso del *machine learning*.

Di ogni algoritmo di *machine learning* è importante la rappresentazione, la valutazione e l'ottimizzazione. La scelta del criterio di valutazione è fondamentale perché impone la direzione in cui procedere nella fase di ottimizzazione per effettuare il processo di *learning*; mentre l'algoritmo di ottimizzazione è scelto in base al compito che l'algoritmo deve svolgere.

Il *deep learning* deve la sua importanza alla facilità con cui sono stimate funzioni non lineari: per ottenere prestazioni simili in passato si modificava lo spazio delle *features* applicando trasformazioni quali *kernel* o PCA. Questa sua caratteristica lo rende particolarmente adatto nell'analisi delle immagini, in cui sono usate più pezzi di immagine combinati tra di loro per effettuare l'analisi. Il problema delle reti neurali fino al primo 2000 era la complessità dell'algoritmo di *learning* (*back propagation*), soppiantato da una versione più efficiente. Infatti l'apprendimento con un solo strato di *hidden layers* è semplice, ma per le reti multistrato l'operazione è molto più complessa.

Suoi svantaggi d'altro canto sono la quantità di dati richiesta dal processo di apprendimento e dal costo dell'hardware richiesto (GPU e TPU).

1 *Learning*

L'uso del *deep learning* è usato inoltre nell'analisi del linguaggio naturale, data l'evoluzione nel tempo di questo e dalle sue sfumature. Altro successo è il *reinfocement deep learning*, che suggerisce al computer come comportarsi in determinate circostanze in base all'evoluzione dell'ambiente e alle scelte pregresse.

Negli algoritmi di *deep learinng* gli *hidden layers* hanno ciascuno una specifica funzione (un obiettivo di *learning* autonomo): il *deep learning* si basa su un apprendimento stratificato.

La rete neurale è composta da un livello di input, uno o più *layers* composti da neuroni con una *funzione di attivazione* e un livello di output. L'operazione di *learning* è iterativa: i pesi sono attribuiti inizialmente in modo casuale e poi modificati in modo iterativo analizzando le singole prestazioni. L'algoritmo usato per correggere i pesi è un algoritmo semplice (generalmente sono usate varianti di *Gradient Descent*), che compie numerose volte piccoli aggiustamenti.

Per funzioni non lineari, teoricamente basta una rete con un singolo *hidden layer*: esiste sicuramente un insieme di pesi per un problema di classificazione; tuttavia trovare la giusta combinazione di valori è arduo. In passato la soluzione è stata quella di rimappare le *features* in uno spazio non lineare e procedere con un classificatore lineare. Il *deep learning* segue

entrambe le strategie: dapprima mappa lo spazio con una trasformata non lineare per poi stimare una funzione non lineare per effettuare la classificazione. I singoli neuroni quindi si specializzano a svolgere singoli compiti semplici (riconoscimento di determinati pattern) che effettuano poi la classificazione. I risultati delle operazioni svolte dai primi neuroni sono poi passati agli strati successivi, che elaborano *features* più astratte e generiche.

Il *deep learning* effettua l'apprendimento dei *layers* uno alla volta con la strategia degli *autoencoders*: l'algoritmo tenta di rappresentare l'input in uno spazio di diversa dimensionalità (o semplificata). Questo sistema permette di evitare l'allenamento di una rete da zero per svolgere compiti diversi da quelli per cui è stata inizialmente programmata. Spesso (come nell'analisi del linguaggio naturale) i vettori di input sono sparsi, causando rumore nell'analisi.

Gli algoritmi di *deep learning* sono *blackbox*: non ne è noto il funzionamento; lo stato dell'arte nella ricerca tenta di interpretare i risultati ottenuti con reti neurali.

Cercare tra tutte le possibili funzioni è un'operazione eccessivamente dispendiosa, dunque la ricerca è fatta per *classi* di funzioni: un'ipotesi sull'apprendimento è quella di linearità:

$$f(x) = xW + b$$

il dataset tuttavia non è sempre linearmente separabile. Altra funzione usata è la sigmoide, che, per la classificazione binaria, restituisce anche il grado di confidenza. In caso di classificazione multiclasse, è usato come output un vettore (un elemento per ogni classe della variabile risposta): la classe della variabile risposta è data dalla normalizzazione (tramite la funzione *softmax*) dei valori del vettore:

$$\begin{aligned} \hat{y} &= softmax(xW + b) \\ &= \frac{e^{-(xW+b)}}{\sum e^{-(xW+b)}} \end{aligned}$$

2 Funzione di Loss

Una *funzione di loss* (di perdita) ℓ attribuisce a un insieme di parametri e di valori il punteggio di errore: l'obiettivo del processo di *learning* è quello di minimizzare la funzione. Generalmente per evitare l'*overfitting* si aggiunge una funzione di regolarizzazione. Questo tuttavia introduce un nuovo parametro λ che stabilisce il grado di regolarizzazione della funzione, che deve essere ottimizzato su un *sample* di dati (generalmente tramite *cross validation*).

HINGE Usato per classificazione binaria, la funzione è semplice:

$$\ell_{HINGE}(y, \hat{y}) = y \cdot \hat{y}$$

Per classificazioni multiclasse la formula diventa:

$$\ell_{HINGE}(y, \hat{y}) = \max(0, 1 - (\hat{y_{target}} - y))$$

che tiene conto di quanto scarto è effettuata la classificazione sbagliata.

Logistic Loss È usata per la classificazione binaria effettuata tramite sigmoidi:

$$\ell_{Logistic}(y, \hat{y}) = \begin{cases} 0 & \hat{y} < 0.5 \\ 1 & \hat{y} > 0.5 \end{cases}$$

Dato che la funzione di loss non è sempre convessa, la rete è trainata su un campione di esempi tramite un algoritmo *gradient based*: i parametri mutano in direzione opposta al gradiente ∇. L'algoritmo usato è quindi *Gradient Descent* nella variante *Stochastic* (SGD). Lo spostamento non è pari alla norma del gradiente ma è ridotta per un coefficiente di *learning* η_t (che diminuisce di valore all'aumentare del passo t fino ad arrivare ad una soglia minima). Per velocizzare l'algoritmo, è effettuato un campionamento (ovvero il *batch*, di m osservazioni) a ogni iterazione per calcolare la media del valore della funzione di loss. Le dimensioni del *batch* è regolata da un altro iperparametro; se il campione è grande, la convergenza è calcolata in modo più preciso ma richiede più potenza di calcolo, in caso contrario la convergenza è ottenuta in più iterazioni ma ogni passaggio è calcolato più velocemente.

Parte II

Deep Feedforward Networks

Una rete *feedforward* è costituita da un flusso che scorre dai dati di input fino all'output in modo lineare nello stesso verso (unidirezionato). Nel processo di *learning* l'errore è propagato all'indietro nella rete per correggere i singoli pesi. Il numero di *layers* stima funzioni in modo annidato:

$$f(X) = f^3(f^2(f^1(X)))$$

La profondità è, in genere, data dal numero di nodi intermedi. Ogni *layer* nascosto è un vettore di neuroni: il suo numero è detta *ampiezza*.

Una rete *feedforward* senza livelli intermedi corrisponde a una funzione lineare; eventuali strati intermedi aumentano la capacità di generalizzazione del modello stimando una funzione di mappatura dello spazio. In questo le reti neurali si differenziano dagli altri algoritmi di *machine learning*: la funzione stimata non è necessariamente convessa, quindi è necessario usare un algoritmo basato sul gradiente.

Una delle funzioni di costo più utilizzate in questo caso è la *maximum likehood*, data dalla *cross entropy* data tra la distribuzione dei dati di training e dalla distribuzione dei modelli. Massimizzare la log-verosimiglianza è equivalente a minimizzare l'errore quadratico medio. Per la scelta dell'output, il *layer* di output attiva una serie di funzioni che possono essere lineari, sigmoidali, softmax o misture gaussiane.

I pesi sono generalmente inizializzati con valori bassi simili tra di loro. Quindi la rete neurale inizia la fase di *training*: è applicato l'algoritmo di ottimizzazione per la minimizzazione della funzione di costo. Generalmente è usato come criterio la *maximum likehood*, ovvero il modello tende a stimare i valori reali della funzione. Come indice si usa la *cross entropy*:

$$H(P, Q) = -E_{x=P}[log(Q(x)]$$

In genere alla funzione di costo è aggiunto un fattore di regolarizzazione. La formula della funzione dipende da P; nel caso particolare in cui si ipotizza sia un modello gaussiano, minimizzare $J(\theta)$ equivale a minimizzare l'errore quadratico medio.

$$N(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{(x-\mu)^2}{2\sigma^2}}$$

$$\log(N(x, \mu, \sigma) = -\log\sigma - \frac{1}{2}\log 2\pi - \frac{(x-\mu)^2}{2\sigma^2}$$

Con $\sigma = 1$:

$$= -\frac{1}{2}\log 2\pi - \frac{(x-\mu)^2}{2}$$

Il primo elemento è costante, il secondo invece comprende la sommatoria dell'errore quadratico: minimizzare una funzione o l'altra è equivalente. Il negativo della log-verosimiglianza risolve il problema di saturazione eliminando l'esponenziale dalla funzione. Spesso non importa studiare la statistica in sé ma un suo valore indice quale media o mediana: la rete neurale non sceglie solamente tra i parametri ma tra tutte le funzioni. Si usa quindi il calcolo variazionale per attribuire a ogni funzione un valore reale.

Le funzioni di output (che possono essere usate anche come funzioni nascoste) da l'input alla funzione di costo; la scelta determina la forma della *cross-entropy*. Possibili funzioni di output sono:

- lineare: $\hat{y} = W^T h + b$ adatta a problemi di regressione, rende facile apprendere anche la matrice di covarianza, inoltre non satura in quanto la funzione è lineare, adattandosi bene all'ottimizzazione;
- sigmoide: $\hat{y} = \frac{1}{1+e^{\omega^T h+b}} = \sigma(\omega^T h + b)$ adatta a classificare, rende difficile l'apprendimento in quanto satura facilmente;
- softmax: $soft\max(z) = \frac{\exp(z)}{\sum \exp(z_i)}$, con $z = W^T h + b$, rappresenta l'argomento massimo dato dal valore della regressione lineare delle varie variabili del vettore, penalizzando le regressioni non corrette;
- mistura di gaussiane: con distribuzioni multi-modali, si usa la sommatoria pesata (dalla verosimiglianza) di gaussiane con media e varianza diversa

Non ci sono regole per la scelta della funzione.

Gli strati intermedi usano generalmente la funzione sigmoide o la tangente iperbolica (tanh), entrambe continue e differenziabili in tutti i punti ma saturano facilmente. Inoltre può essere usata la funzione $relu$.

Definita l'architettura della rete, i parametri sono stimati in modo sequenziale: l'output di un livello diventa l'input del successivo; andando in profondità con i livelli, si ottengono neuroni che individuano pattern sempre più astratti, tuttavia richiedono più dati per effettuare l'allenamento. La profondità della rete è testata empiricamente in modo sperimentale. Da un punto di vista computazionale, aumentare il numero di livelli e il numero di neuroni per ogni livello, aumenta la capacità predittiva ma richiede sempre più dati. L'apprendimento è fatto tramite la tecnica dell'*auto-encoder*: ogni livello tenta di replicare il precedente, offrendo una rappresentazione in uno spazio diverso privo di rumore. L'operazione può essere effettuata anche in modo non supervisionato.

3 *Overfitting* e migliorie del processo di apprendimento

Il processo di apprendimento può cadere nell'*overfitting*: se i pesi per un neurone aumentano particolarmente, questo può adattarsi in modo perfetto ai dati non generalizzando per le previsioni. Per evitare questo fenomeno, esistono diverse strategie: prima tra tutte il *dropout*, che elimina con una certa probabilità (arbitraria) dei nodi della rete per far sì che gli altri "imparino" a effettuare le previsioni senza basarsi su questo nodo.

Altro sistema utilizzato è quello di modificare il *train-set*, introducendo delle osservazioni artificiali (quali immagini leggermente ritoccate, capovolte o simili) basate su quelle reali.

Parte III
Computer vision

4 *Hand Crafted Features*

Si dichiara un istogramma di gradiente orientato: ogni pixel dell'immagine è mappato col suo gradiente, per poi essere raggruppato in zone limitrofe, in modo da raggruppare le angolazioni simili.

5 Classificatore

Le *features* estratte dall'immagine sono poi date in pasto ad un algoritmo di classificazione che, dopo l'allenamento, è in grado di fare previsioni partendo dall'immagine.

L'immagine deve essere prima pre-processata in modo tale da ridurne la dimensionalità. In questo modo è possibile costruire un *embedding* e, tramite l'algoritmo 1-NN (che non richiede allenamento), verificare quale sia la classe dell'osservazione più vicina (o delle k più vicine) ad un nuovo dato. In alternativa, se lo spazio non è particolarmente complesso ed è possibile separare linearmente le osservazioni, si possono usare algoritmi della famiglia delle SVM; possono essere usate anche funzioni *kernel* per mappare lo spazio in un nuovo spazio dove la separazione sia più facile.

Esperimenti sulla corteccia visiva dei gatti, mostrano che il cervello gestisce le immagini tramite *features* (*pattern* semplici combinati assieme in modo sempre più complesso).

6 Reti convulazizonali

I livelli convulazionali sono usati per ridurre il numero dei parametri del modello, gestendo *features* di livello più alto. In una rete convulazionale, i *layers* sono gestiti su tre dimensioni. In uscita, sono generati tanti canali quanti sono i filtri usati per la convulazione. Siccome l'operazione di convulazione riduce le dimensioni dell'immagine, si usa una procedura chiamata *padding* che aggiunge un bordo attorno all'immagine per far sì che l'output sia delle stesse dimensioni. Si possono usare degli zeri o altri sistemi a scelta. Inoltre lo spostamento può non essere unitario ma più grande (*stride*).

In questo modo si riduce drasticamente il numero di parametri della rete.