**Lecture 11: Introduction**

**to Neural Network**

**1. Logistic Regression**

Il **deep learning** è una set di tecniche appartenente alla branca del machine learning usato per problemi legati alla visione computerizzata, elaborazione del linguaggio naturale e riconoscimento vocale.

Uno dei motivi del suo ottimo funzionamento sono anche i nuovi **metodi computazionali** usati, anche molto costoso da quel punto di vista, per cui serve trovare un modo di parallelizzare i codici e usa bene la GPU per computizzare bene i metodi del deep learning.

Un’altro aspetto che si è espanso nel deep learning è la **disponibilità dei dati** dopo la bolla di internet e la digitalizzazione dei sistemi nel mondo permettendo ai sistemi di imparare di più grazie alla grande quantità di dati. Essi sono molto flessibili e più dati vengono usati più capiscono le caratteristiche salienti di dei dati.

Poi ci sono gli **algoritmi**, la gente ha scoperto metodi ottimali per usare i dati e la forza computazionale per creare modelli.

La **logistic regression** (regressione logistica) è un algoritmo di classificazione, essa può essere interpretata come un caso specifico di una rete neurale.

Ci prendiamo come **obiettivo** il classificare qualcosa, per esempio **trovare gatti in un’immagine**, uno dei metodi possibili è ma **classificazione binaria**, ossia se l’immagine **ritrae** un gatto avere come risultato **1,** se **non ce ne sono** invece avremo **0**, nell’esempio che verrà prendiamo il caso che le immagini ritraggano solo un gatto.

Il modello di regressione logistica consisterà nel prende l’immagine di un gatto, in informatica le immagini possono essere rappresentate come matrici 3D, ossia se si prende un immagine di risoluzione 64 x 64 per rappresentare i pixel si fa 64 x 64 x 3 perché tre sono i colori del canale RGB, ossia rosso, verde e blu, ogni pixel nell’immagine può essere rappresentato con tre numeri diversi ossia con tre numeri diversi, i pixel saranno quindi 12.288.

Prima di tutto si prosegue a prendere tutti i numeri della matrice e appiattirli in un vettore per poterlo usare nella regressione logistica. Si prendono i numeri del vettore e li si passa a un’operazione che è **wx + b,** dove x è l’immagine e w è il vettore matrice, e poi si esegue la moltiplicazione per sigmoid, il quale prende un numero da infinito negativo a infinito positivo e lo posiziona in un range tra 0 e 1 in modo di avere una classificazione dei numeri in un vettore in uno spazio più gestibile. Il risultato sara **ŷ.**

Una volta ricavato il modello andrà **inizializzato**, principalmente w (weights) e b (bias), dopo l’inizializzazione si cerca il w e b ottimali, ossia si prosegue all’**ottimizzazione** dei parametri definendo una **loss function**, di base sai che funzione vuoi arrivare nel machine learning ma non conosci i parametri, la loss funcion aiuta in questo caso a ricavarli, minimizzando la loss function si troveranno i parameteri giusti, in questo processo si ricava anche l’algoritmo del **gradiente,** ossia computero la derivata della perdita con rispetto dei miei parametri per aggiornali constantemente, un for loop. Dopodichè potremo usare w e b per la **previsione** di y.

Nelle reti neurali definiremo un **neurone** come un’operazione a due parti, una **lineare** e una d’**attivazione**. Nel caso di prima wx + b \* sigmoid è il neurone, dove la wx + b è la parte linerare e l’operazione del sigmoid quella d’attivazione.

Dopo il neurone vogliamo come seconda equazione è il modello, ossia **architettura,** ossia la composizione dei neuroni**,** più i **parametri**.

Dopo il primo obbiettivo che ci siamo dati il secondo è di trovare gatti, leoni e iguane nelle immagini, ossia cercare più di un solo animale. In pratica passerò i numeri del vettore a tre neuroni diversi, uno in più per riconoscere ogni soggetto.

Il gruppo di neuroni inizializzato con quei tre neuroni viene chiamato **layer (strato),** c’è uno strato nel quale i neuroni non comunicano tra di loro. Ogni neurone ha un indice che lo indentifica nel layer in cui si trova (nel caso riportato i tre neuroni hanno lo stesso indice perché nello stesso layer). Nell’adattare la classificazione precedente del gatto essendoci ora tre soggetti da riconoscere il gatto verrà rappresentato con tre numeri, due 0 e un 1, il primo per indicare il gatto, il secondo per il leone e l’ultimo per l’iguana. Una delle cose positive delle reti neurali è che logicamente i neuroni non comunicano tra loro per cui anche se avessimo un immagine sia con un gatto che con un leone la rete è capace di dare come risultato 1, 1, 0 (aka: gatto, leone, iguana), per cui se allenata correttamente il passaggio dal riconoscere animali singoli in un’immagine verrà naturale in base a come etichetti il tuo dataset visto che ogni neurone cercherà solo di rispondere alla sua categoria, in un certo senso sono tre regressioni logistiche separate che condividono ciascuna il proprio output. Il contro di questo modello è che nel caso volessimo catalogare tre malattie diverse ma una di queste tre è molto rara, i due neuroni incaricati delle prime due si alleneranno costantemente mentre il terzo per la malattia rara molto poco, ergo in questo caso invece di allenare la rete a distinguere tra tre malattie diverse conviene fare un modello specializzato solo in quella malattia rara visto che altrimenti i dati d’allenamento sarebbero molto pochi.

Come terzo obiettivo applichiamo un **vincolo** di riconoscere un solo animale per immagine perché abbiamo visto che logicamente la rete riconoscerà tutti gli animali nelle immagini dando come output tutti e non uno specifico. Per risolvere si va a modificare la formula dei neuroni in modo da avere una distribuzione delle probabilità per ogni soggetto la quale somma fa 1 (softmax multi-class network), in questo caso diamo per scontato che ci sia almeno uno dei tre animali, nel caso servirà un quarto output nel dire che non ci sono gli animali richiesti. A causa della modifica va modificata anche la loss function (che era addestrata a restituire come output o 0 o 1). Di base facciamo tre volte la precedente loss function per ciascun neurone e le sommiamo tra loro, anche questa volta la scarsità di allenamento di uno degli obiettivi potrebbe causare problemi. Alla fine con gli output ricavati si fa una distruibuzione della probabilità e si prende quella più alta settandolo a 1 e gli altri a 0.

Nel caso volessi creare una rete per riconoscere l’età di un gatto una possibilità è usare il softmax per distinguere le diverse età del gatto e prendere la più probabile, anche perché l’età sarà solo una. Un’altra possiblità è di fare una regressione, in quel caso va modificato il sigmoid per mettere l’età compresa tra 0 e N-età, in questo caso è meglio usare un ReLU visto che non restituisce valori negativi.

**2. Neural Networks**

L’obiettivo rimane lo stesso: in un’immagine scopriamo se c’è (1) o no (0) un gatto.

Questa volta avremo più neuroni con più strati, i neuroni comunicano non con i neuroni dei propri strati ma con quelli di altri strati. Quando stabilisci un’architettura bisogna stare attenti che lo strato dell’output ha lo stesso numero di neuroni che vuoi, il numero di classi che vuoi per la riclassificazione e uno per la regressione. Ricorda che uno strato è un insieme di neuroni che non comunica tra di loro, il primo layer viene chiamato **input layer**, l’ultimo **output layer**, il secondo invece viene chiamato **hidden layer** perché ad esso l’input e l’output sono nascosti, ossia non vede direttamente l’input perché riceve solo i dati passati dai neuroni del primo layer, e non vede l’output perché l’unica cosa che comunica con il layer finale è i dati che gli passerà.

Di base i primi layer capiranno i fondamenti concettuali delle immagini, ossia i margini, i neuroni dovranno identificare questi margini, ciascuno identificheà i suoi margini specifici, dopodichè i neuroni comunicheranno i margini ai neuroni dell’hidden layer i quali proveranno a capire cosa ci sta, per il gatto ad esempio le orecchie, la bocca ecc…, in questo modo passando i risultati all’output layer esso proverà a identificare il viso del gatto e capire se effettivamente c’è un gatto o no. Lo strato viene chiamato hidden perché non sappiamo esattamente cosa comprendono i neuroni di quello strato, di base più si va profondi più complessi sono i dati compresi dai neuroni.

Nel caso della previsione del prezzo di una casa, avendo come input il numero di stanze, grandezza della casa, il cap e la ricchezza del quartiere, prendiamo tre neuroni nell’input e uno nell’output, in questo modo incrociando i dati si possono ricavare alcune informazioni ossia, i neuroni del cap e la richezza possono valutare la qualità delle scuole intorno alla casa, il cap ci dirà se si può camminare nel quartiere, infine il numero di stanze e la grandezza della casa ci dirà che tipo di famiglia può venire a viverci. Insomma le informazioni ricavate sono sicuramente meglio di quelle passate in input per ricavare il prezzo della casa. Insomma concretamente alla rete neurale queste connessioni che da umani viene naturale fare la rete neurale ne verrà a capo da sola a ricavare le correlazioni tra gli input per arrivare all’output del prezzo (un sistema a scatola chiusa). In alcuni casi le reti neurali sono meglio degli umani a scoprire le correlazioni tra i dati.

Un’altro metodo è l’**allenamento end to end**, ossia avendo degli input e una verità a cui la rete deve arrivare la rete si costruisce l’architettura da sola.

La **discesa del gradiente stocastico** è formata da aggiornamenti ai pesi e al bias delle reti, la direzione del gradiente è incasinata e non riesce a rappresentare bene l’intero gruppo della rete, mentre la **discesa del gradiente** (o più precisamente **discesa del gradiente batch**) si aggiorna dopo aver visto l’intero gruppo con un gradiente più preciso.

Cosa succede a un gruppo di input di m esempi?

In questo caso x non è più una singola immagine ma una matrice di vettori che compongono più immagini. Avendo delle **equazioni di propagazione**, dei layer della rete (input, hidden, output, non necessariamente solo tre ma più layer significa più computazione necessaria, non esiste una risposta giusta a quanti ne servono per fare qualcosa, va testato uno a uno e poi nel caso scegliere l’architettura più indicata, di base più complessa l’operazione, tipo distinguere gatti o il giorno dalla notte, più profonda dovrebbe essere la rete), quello che vogliamo è di parallelizzazare il nostro codice o calcolo il più possibile consegnando gruppi di input e parallelizzando le equazioni di propagazione. Per fare ciò una delle cose che si esegue è fare il **broadcasting** con il codice per gestire vettori di forme e/o dimensioni diverse, nel caso di python si usa **numpy**.

Le equazioni di propagazione fanno parte dei processi di **propagazione in avanti (forward)**, ossia dall’input all’output, per il calcolo del gradiente e altre cose si usa la **propagazione a ritroso (backword)**. Un problema da affrontare è anche quello dell’ottimizzazione, un modello è fatto da un’architettura più i parametri, avendo l’architettura serve solo avere i parametri, perciò serve ricavare una **funzione obiettivo (objective function)**, ossia la **loss**, oppure **cost function**, loss per un solo esempio nel gruppo e cost nel caso di più esempi.