Algorithm Unibo

Machine Learning Pratico

Samuele Marro

Università di Bologna

t.me/algo_unibo

Introduzione

- Samuele Marro, studente magistrale ad Artificial Intelligence
- Tutor del corso ”Introduzione all’Apprendimento Automatico”
- Membro del Collegio Superiore, area STEM
- Ricercatore con il DISI
- Obiettivo del seminario: sapere tra 1.5 h costruire e addestrare una rete neurale
- Python + PyTorch

Struttura della Lezione

- Ci alterneremo tra slide e Google Colab
- Faremo e addestreremo una rete neurale insieme passo-per-passo
- Farete gara a chi costruisce la migliore rete
- Importante: stiamo facendo 6 CFU di corso in 1.5 h
  - Alcune cose verranno semplificate
  - Lascerò del materiale per approfondire

Cosa vuol dire fare un ragionamento?

Il Ragionamento

- Un ragionamento parte da delle informazioni iniziali e arriva a delle informazioni finali

Il Ragionamento

- Un ragionamento parte da delle informazioni iniziali e arriva a delle informazioni finali
- Un ragionamento è come una funzione!
  - Dato x, restituisci x^2
  - Data una lista di sintomi, restituisci una diagnosi
  - Dato un film, restituisci una recensione
  - Data una domanda, restituisci una risposta

Il Ragionamento

- Un ragionamento parte da delle informazioni iniziali e arriva a delle informazioni finali
- Un ragionamento è come una funzione!
  - Dato x, restituisci x^2
  - Data una lista di sintomi, restituisci una diagnosi
  - Dato un film, restituisci una recensione
  - Data una domanda, restituisci una risposta
- Alcuni ragionamenti sono giusti, altri sbagliati, altri parzialmente corretti
- Un ragionamento diverso corrisponde a una funzione diversa

Il Machine Learning

- Nel Machine Learning, creiamo una simulazione del cervello (una **rete neurale**) in grado di fare ragionamenti molto complessi
  - Più la rete neurale è grande, più sono complessi i ragionamenti che può fare
- Questa rete ha dei parametri (o **pesi**) che influenzano il tipo di ragionamento che fa
- Cambiando i pesi, cambia il ragionamento (e quindi la funzione calcolata)
- L'obiettivo è di scegliere i pesi in modo che la rete neurale faccia il ragionamento che vogliamo

Numeri e Immagini

- Il tipo di ragionamento che insegneremo oggi alla rete neurale è la **classificazione di immagini**
  - Input: un'immagine (es. immagine di un gatto)
  - Output: una categoria tra 10 possibili (es. "gatto", "cane", "pappagallo"...)
- Problema: le reti neurali lavorano solo con **tensori** (array multidimensionali) di numeri floating point

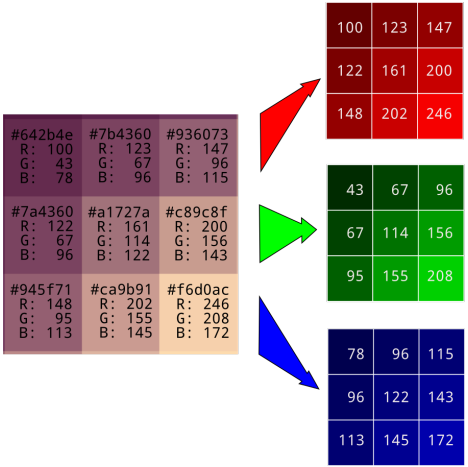
Rappresentazione di Immagini

- Come possiamo rappresentare un'immagine come un tensore?

[a] L'ordine è per ragioni storiche/pratiche: librerie diverse usano ordini diversi

Rappresentazione di Immagini

- Come possiamo rappresentare un'immagine come un tensore?
- Usiamo i valori RGB dell'immagine!
- Un'immagine è un tensore 3D [canale, altezza, larghezza][a]
- Un'immagine 1920x1080 sarà rappresentata come un tensore di dimensioni [3, 1080, 1920]



[a] L'ordine è per ragioni storiche/pratiche: librerie diverse usano ordini diversi

Rappresentazione di Categorie

- Come possiamo rappresentare una categoria tra 10 possibili? (”gatto”, ”cane”, ”pappagallo”...)

Rappresentazione di Categorie

- Come possiamo rappresentare una categoria tra 10 possibili? (”gatto”, ”cane”, ”pappagallo”...)
- Idea base: gatto = 0, cane = 1, pappagallo = 2...

Rappresentazione di Categorie

- Come possiamo rappresentare una categoria tra 10 possibili? (”gatto”, ”cane”, ”pappagallo”...)
- Idea base: gatto = 0, cane = 1, pappagallo = 2...
  - Problema: la media di un gatto (0) e di un pappagallo (2) non è un cane (1)!
  - La rete rischia di imparare male

Rappresentazione di Categorie

- Come possiamo rappresentare una categoria tra 10 possibili? (”gatto”, ”cane”, ”pappagallo”...)
- Idea base: gatto = 0, cane = 1, pappagallo = 2...
  - Problema: la media di un gatto (0) e di un pappagallo (2) non è un cane (1)!
  - La rete rischia di imparare male
- **One-hot encoding**: rappresentiamo ogni categoria come un vettore avente 1 nella posizione giusta, 0 nelle altre posizioni
  - gatto = [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
  - cane = [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
  - pappagallo = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
  - ...

Ricapitolando

La funzione da imparare è:

- Dato un tensore 3D contenente i valori RGB dell'immagine...
- Restituisci un tensore 1D (aka un vettore) contenente 1 nella posizione della categoria corretta e 0 nelle altre posizioni

Costruiamo una Rete Neurale

- Le reti neurali moderne sono fatte da ”strati”
- Strati diversi hanno funzionalità diverse
- Ogni strato manipola le informazioni e le passa allo strato successivo
- Ogni strato ha dei parametri (o **pesi**): cambiando i pesi, cambia il funzionamento dello strato
- Noi vedremo i 3 tipi di strato più comuni

Strati Lineari (aka Fully Connected o Dense)

- Una trasformazione lineare è una forma molto basilare di ragionamento
- Uno strato lineare prende un vettore x e gli applica una trasformazione lineare ($Wx + b$)
- Input: tensore 1D, output: tensore 1D (di lunghezza potenzialmente diversa)
- I pesi W e b cambiano la trasformazione lineare (e quindi il ragionamento)
- Idea: combiniamo tanti strati lineari, in modo da fare ragionamenti più complessi

Funzioni di Attivazione

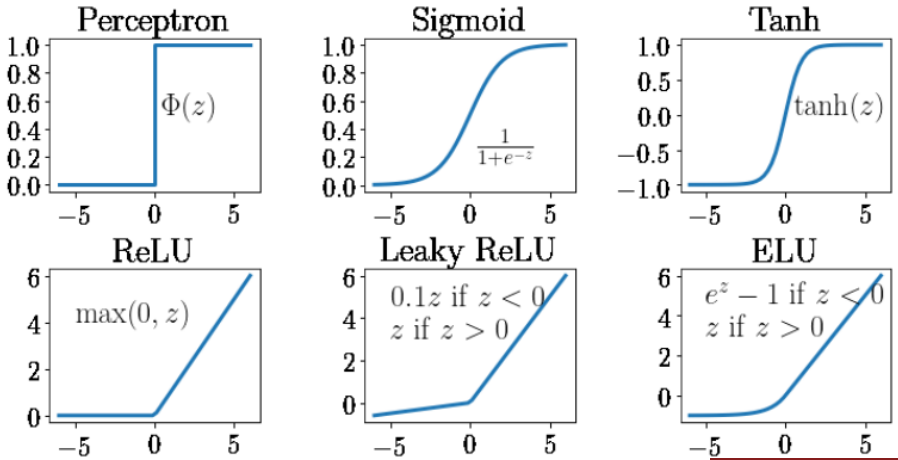
- Uno strato lineare può solo fare operazioni lineari
  - Una composizione di funzioni lineari è ancora una funzione lineare
- Come possiamo imparare a modellare funzioni non-lineari?

Funzioni di Attivazione

- Uno strato lineare può solo fare operazioni lineari
  - Una composizione di funzioni lineari è ancora una funzione lineare
- Come possiamo imparare a modellare funzioni non-lineari?
- **Funzioni di attivazione**: Invece di calcolare semplicemente $Wx + b$, applica dopo una funzione (es. $f(Wx + b)$)
- Se alterniamo strati lineari a funzioni di attivazione, otteniamo funzioni molto più complesse: $f_3(W_3 \cdot f_2(W_2 \cdot f_1(W_1 x)))$
- Alcune librerie fondono gli strati lineari e le funzioni di attivazione in un unico strato

Funzioni di Attivazione

- Domanda: Quale funzione di attivazione usare?
- Risposta: Per reti piccole non importa molto, per reti grandi bisogna fare attenzione (ReLU, Leaky ReLU ed ELU vanno sempre bene)



Strati Reshape

- Gli strati lineari lavorano su tensori 1D, ma le immagini sono tensori 3D
- Gli strati Reshape cambiano la forma del tensore
- Esempio: ”appiattire” un array 3D in un array 1D
  - Lo strato ”Flatten” è un caso specifico dello strato Reshape che fa esattamente questo
- Non hanno pesi

Comporre una Rete Neurale

- Concatenando gli strati, si costruisce un vero e proprio cervello
- PyTorch supporta $\sim$ 100 tipi di strato, ma ce ne sono centinaia
- Più è grande il cervello, più complicate sono le operazioni che possiamo fare sui dati
- L'insieme degli strati di una rete neurale formano l'**architettura**
  - ResNet, AlexNet, GoogLeNet, GPT...
- Alcune architetture sono più brave a imparare certe funzioni rispetto ad altre

Pesi

- Cambiando i pesi, si cambia la trasformazione che la rete neurale fa ai dati
- Alcune scelte di pesi daranno vita a una rete neurale che fa quello che vogliamo (nel nostro caso, associare un'immagine a gatto/cane/pappagallo...)
- La maggior parte delle scelte dei pesi daranno vita a una rete neurale assolutamente inutile
- Una rete neurale piccola ha $\sim$ 5 milioni di pesi
- GPT-4 ha un **triliardo** di pesi

Pesi

- Cambiando i pesi, si cambia la trasformazione che la rete neurale fa ai dati
- Alcune scelte di pesi daranno vita a una rete neurale che fa quello che vogliamo (nel nostro caso, associare un'immagine a gatto/cane/pappagallo...)
- La maggior parte delle scelte dei pesi daranno vita a una rete neurale assolutamente inutile
- Una rete neurale piccola ha $\sim$ 5 milioni di pesi
- GPT-4 ha un **triliardo** di pesi

Come scegliamo i pesi giusti?

Trovare i Pesi

Due step:

1. Fissiamo una funzione che ci dice quanto sta sbagliando la rete (**funzione loss**)
2. Troviamo un algoritmo (**discesa a gradiente**) per ridurre la loss

Funzione Loss

- Come definiamo una funzione loss?

Funzione Loss

- Come definiamo una funzione loss?
- Idea: prendiamo un'immagine x di un gatto (che quindi sappiamo deve avere output $y = [1, 0, 0, \ldots, 0]$)
- Calcoliamo l'output $\hat{y} = \mathit{rete}(x)$
- Calcoliamo l' "errore" tra y e $\hat{y}$

Funzione Loss

- Come definiamo una funzione loss?
- Idea: prendiamo un'immagine x di un gatto (che quindi sappiamo deve avere output $y = [1, 0, 0, \ldots, 0]$)
- Calcoliamo l'output $\hat{y} = \mathit{rete}(x)$
- Calcoliamo l' "errore" tra y e $\hat{y}$
  - Errore quadratico: $\sum_i (\hat{y}_i - y_i)^2$

Funzione Loss

- Come definiamo una funzione loss?
- Idea: prendiamo un'immagine x di un gatto (che quindi sappiamo deve avere output $y = [1, 0, 0, \ldots, 0]$)
- Calcoliamo l'output $\hat{y} = \mathit{rete}(x)$
- Calcoliamo l' "errore" tra y e $\hat{y}$
  - Errore quadratico: $\sum_i (\hat{y}_i - y_i)^2$
  - Errore assoluto: $\sum_i |\hat{y}_i - y_i|$

Funzione Loss

- Come definiamo una funzione loss?
- Idea: prendiamo un'immagine x di un gatto (che quindi sappiamo deve avere output $y = [1, 0, 0, \ldots, 0]$)
- Calcoliamo l'output $\hat{y} = \textit{rete}(x)$
- Calcoliamo l' "errore" tra y e $\hat{y}$
  - Errore quadratico: $\sum_i (\hat{y}_i - y_i)^2$
  - Errore assoluto: $\sum_i |\hat{y}_i - y_i|$
  - Entropia incrociata[1]
  - ...

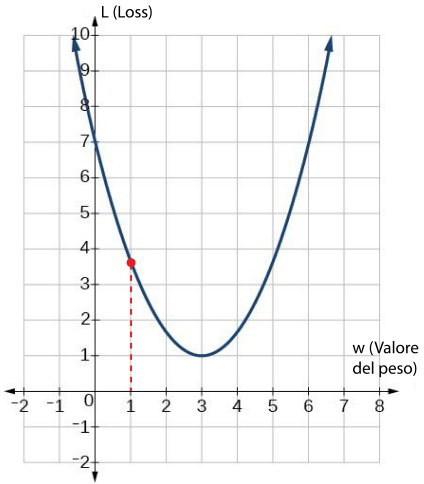
[1] Nella vita reale si usa soprattutto questa per la classificazione, ma è molto più complicata

Funzione Loss

- Se facciamo la somma/media della loss su un dataset molto grande (il **train set**), abbiamo una buona misura della loss del modello!
- In pratica non si fa su tutto il dataset, ma si prende un sottoinsieme (chiamato **batch**) del dataset
- Se tutto va bene, la rete dovrebbe poi **generalizzare** anche su un dataset di immagini che non ha mai visto (il **test set**)

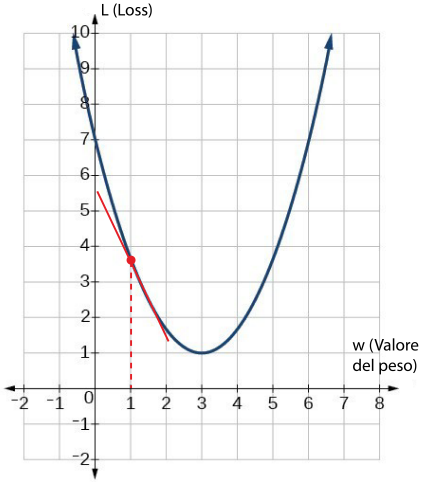
Discesa a Gradiente (aka Stochastic Gradient Descent)

- Immaginiamo di avere un solo peso w
- Variando il peso, varia la loss
- Scopriamo che fissando $w = 1$, la loss è di 3.75
- Esiste un modo per capire in che direzione bisogna variare il peso per ridurre la loss?



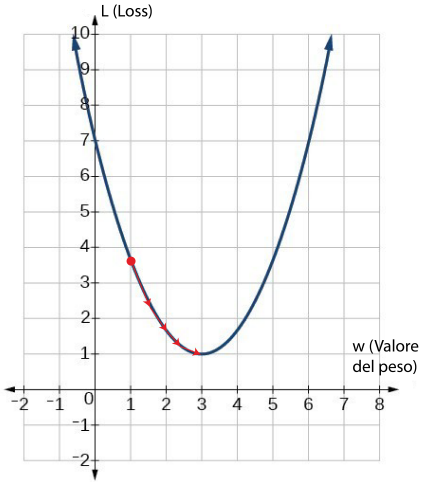
Discesa a Gradiente (aka Stochastic Gradient Descent)

- Usiamo la derivata!
- Ogni **epoca**, ci muoviamo leggermente nella direzione di decrescita (negativo della derivata)
  - $w_{new} = w_{old} - \eta \frac{dL}{dw}(w_{old})$
- η è il **learning rate** (aka quanto è grande ogni "passo")



Discesa a Gradiente (aka Stochastic Gradient Descent)

- Usiamo la derivata!
- Ogni **epoca**, ci muoviamo leggermente nella direzione di decrescita (negativo della derivata)
  - $w_{new} = w_{old} - \eta \frac{dL}{dw}(w_{old})$
- η è il **learning rate** (aka quanto è grande ogni "passo")
- Dati abbastanza passi, raggiungeremo il valore ottimale
- Quando i pesi sono più di uno, si usa il gradiente

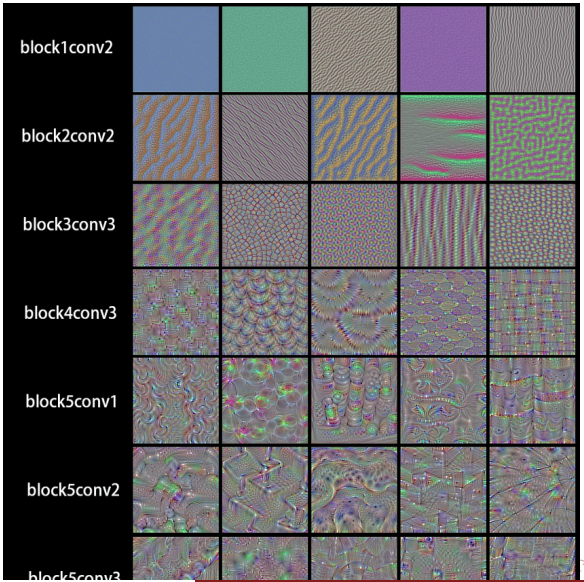


Ricapitolando

- Prendi un train set di coppie $\langle$ input, output corretto $\rangle$
- Costruisci una rete
- Ripeti abbastanza volte:
  - Confronta l'output che la rete assegna alle immagini del train set con l'output corretto usando la funzione loss
  - Calcola il gradiente della loss rispetto ai pesi
  - Aggiusta leggermente i pesi
- Testa la tua rete neurale su immagini del test set

Strati Convolutivi

- Gli ”occhi” di una rete neurale
- Gli strati convolutivi riconoscono dei ”pattern” nell’immagine
- Cambiando i pesi, cambiano i pattern riconosciuti
- Mettendo più strati convolutivi uno dopo l’altro, gli strati successivi riconoscono pattern sempre più complessi



Strati Convolutivi

- Input: tensore 3D di forma [canali, altezza, larghezza], output: tensore 3D di forma [numero pattern, altezza, larghezza][2]
- Anche loro hanno bisogno di una funzione di attivazione dopo
- Le reti neurali che usano questi strati vengono dette Convolutional Neural Networks (CNN)
  - Strati convoluzionali (+ strati di pooling), seguiti da uno strato Flatten, seguito da strati lineari
  - Ottime per lavorare su immagini

[2] Tecnicamente, una convoluzione può cambiare l'altezza e la larghezza, ma noi vedremo solo convoluzioni che non la cambiano (padding = 'same').

Conclusione

- Un ragionamento è come una funzione
- Una rete neurale è una funzione ”addestrabile”: cambiando i pesi, cambia la funzione
- Calcolando la loss su un train set, possiamo determinare quanto sono buoni i pesi
- Con discesa a gradiente possiamo trovare i pesi ottimali
- Architetture diverse sono adatte a ragionamenti diversi

Argomenti Consigliati

- **Adam**: Una variante di SGD molto potente
- **Overfitting**: Come evitare che una rete ”memorizzi” le risposte e si ossessioni
- **Data Augmentation**: Come prendere un dataset e ingrandirlo ”artificialmente”
- **Max Pooling**: Un altro strato famosissimo, usato con le convoluzioni
- **Transfer Learning**: Come prendere reti già addestrate e adattarle ai propri bisogni

Risorse

- Il Machine Learning è un'area di ricerca enorme
  - $>$ 100 paper *al giorno*
- Libri di Machine Learning
  - *Neural Networks and Deep Learning*, di Michael Nielsen
    - Il libro su cui ho iniziato io
    - Molto approcciabile (prerequisiti: matematica del liceo)
    - Gratis sul sito dell'autore
  - *Deep Learning*, di Goodfellow, Bengio e Courville
    - Scritto da ricercatori famosissimi (Goodfellow era direttore di Google DeepMind, Bengio ha vinto il Premio Turing)
    - Stesso livello di prerequisiti
    - Mai letto
    - Gratis sul sito dell'autore

Risorse

- Tutorial
  - machinelearningmastery.com ha molti tutorial pratici scritti bene
  - Le due grandi librerie (PyTorch e Tensorflow) hanno molti tutorial sul loro sito
  - Cercare [parole chiave] + Medium dà quasi sempre un buon tutorial
- awesome-machine-learning-resources contiene una lista di tutti gli argomenti del Machine Learning, con link
- Motori di ricerca per paper
  - Google Scholar: molto completo, ma difficile da usare se non sai le parole chiave giuste
  - perplexity.ai: basato su ChatGPT, molto più facile da usare ma meno completo

Risorse

- Youtube
  - 3blue1brown ha due video ottimi su Machine Learning (uno sul Gradient Descent e l'altro sulle convoluzioni)
  - Two Minute Papers riassume nuovi paper usciti
- Per qualche ragione, quasi tutti i ricercatori sono attivi solo su Twitter
  - Per iniziare: @goodfellow_ian, @geoffreyhinton, @ai__pub, @JeffDean, @AndrewYNg, @jeremyphoward, @ylecun, @karpathy, @michael_nielsen
- Corso "Introduzione all'Apprendimento Automatico" di Andrea Asperti
  - Io faccio le lezioni pratiche (anche il prossimo anno, forse)
  - Trovate il materiale di quest'anno su samuelemarro.it/teaching

Domande?