Apprendimento automatico Apprendimento profondo

Mario Vento, Pasquale Foggia e Diego Gragnaniello

Popolarità della rete neurale

- Prima ondata: Cibernetica: Anni '40–'60
  - Studio dei neuroni biologici, apprendimento di Hebbian, percettrone
- Seconda ondata: Connessionismo: anni '80–'90
  - MLP con retropropagazione, reti di Kohonen
- Terza ondata: Apprendimento profondo: anni 2000–oggi
  - Deep NN non supervisionate (ad es. macchine di Boltzmann ristrette)
  - Deep NN supervisionate (ad es. reti neurali convoluzionali)
  - Risultati rivoluzionari su diversi compiti complessi (ad esempio il riconoscimento di immagini)

Seconda ondata di NN: reti poco profonde

- Le NN della seconda ondata erano poco profonde: in genere 1–2 strati nascosti.
  - Il teorema di approssimazione universale garantisce che uno strato nascosto è sufficiente!
  - La potenza di calcolo limitava il numero di neuroni
  - Spesso erano disponibili solo piccoli insiemi di dati.
  - La convinzione che i livelli debbano essere completamente collegati (fully connected)
  - Problemi numerici con discesa del gradiente su molti strati (vanishing gradient)

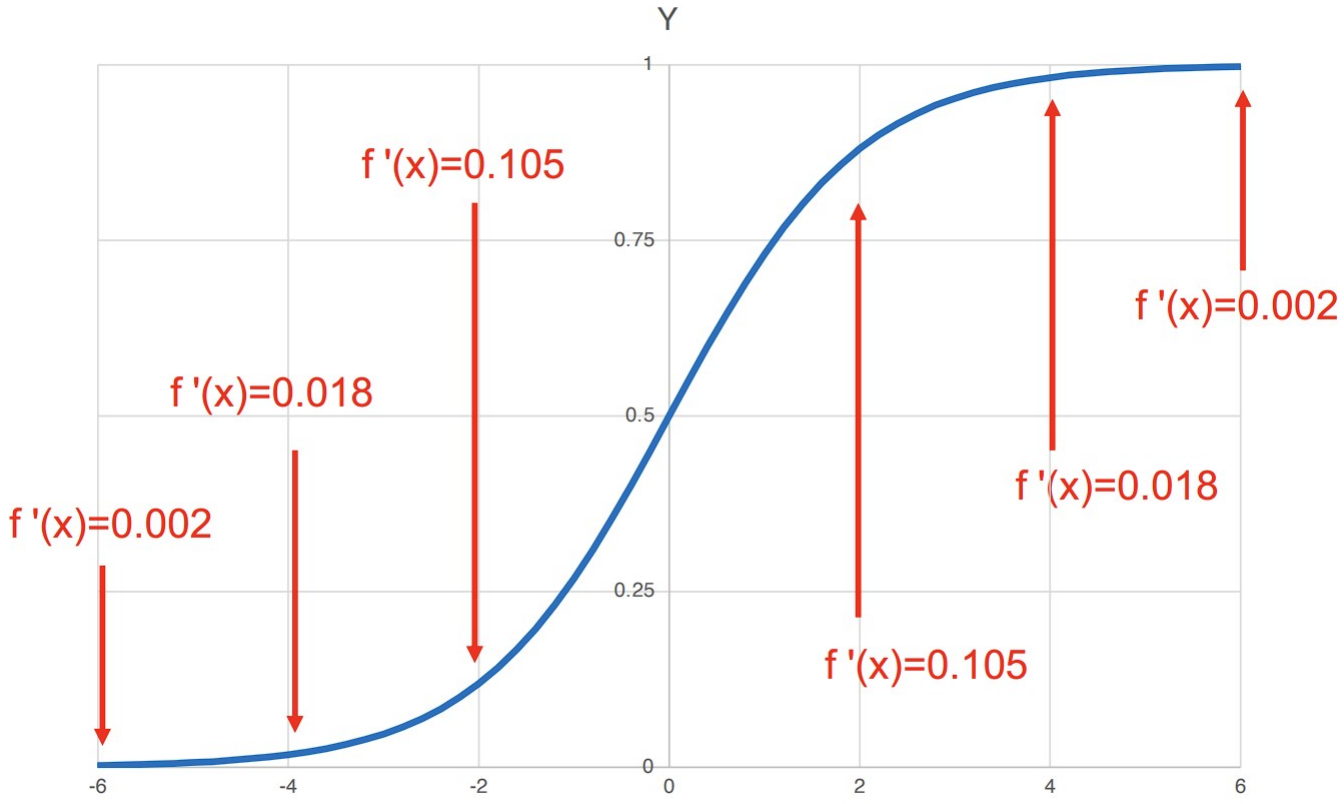
Vanishing gradient

- Le reti della seconda ondata utilizzano tipicamente la *sigmoide* (o la sua parente, *tanh*) come funzione di attivazione.

- La sigmoide ha alcune belle proprietà formali
  - Continuo e infinitamente differenziabile
  - Ha un'interpretazione probabilistica ben consolidata (vedi regressione logistica)

Vanishing gradient

- Sfortunatamente, la derivata della sigmoide è quasi pari a 0 per la maggior parte del suo dominio.



Vanishing gradient

- Il problema peggiora se si concatenano più layers

Per la regola della catena:

$$\frac{\partial J}{\partial y_j^{[k-1]}} = \sum_i \frac{\partial J}{\partial y_i^{[k]}} \cdot f^{[k]\prime}\left(\sum_m w_{im}^{[k]} \cdot y_m^{[k-1]}\right) \cdot w_{ij}^{[k-1]}$$

Gradient back-propagated from layer [k+1]

Gradient back-propagated from layer [k]

Vanishing gradient

- Il problema peggiora se si concatenano più strati:

Infatti, per la regola della catena:

$$\frac{\partial J}{\partial y_j^{[k-1]}} = \sum_i \frac{\partial J}{\partial y_i^{[k]}} \cdot f^{[k]\prime}\left(\sum_m w_{im}^{[k]} \cdot y_m^{[k-1]}\right) \cdot w_{ij}^{[k-1]}$$

… then, that will be even smaller!

If this is small…

… and this is small…

Vanishing gradient

- Se la derivata della funzione di attivazione è ~0, il gradiente diventerà sempre più piccolo man mano che viene retro-propagato su più layers.

Perché è un problema?

Vanishing gradient

- Se la derivata della funzione di attivazione è ~0, il gradiente diventerà sempre più piccolo man mano che viene retro-propagato su più strati.

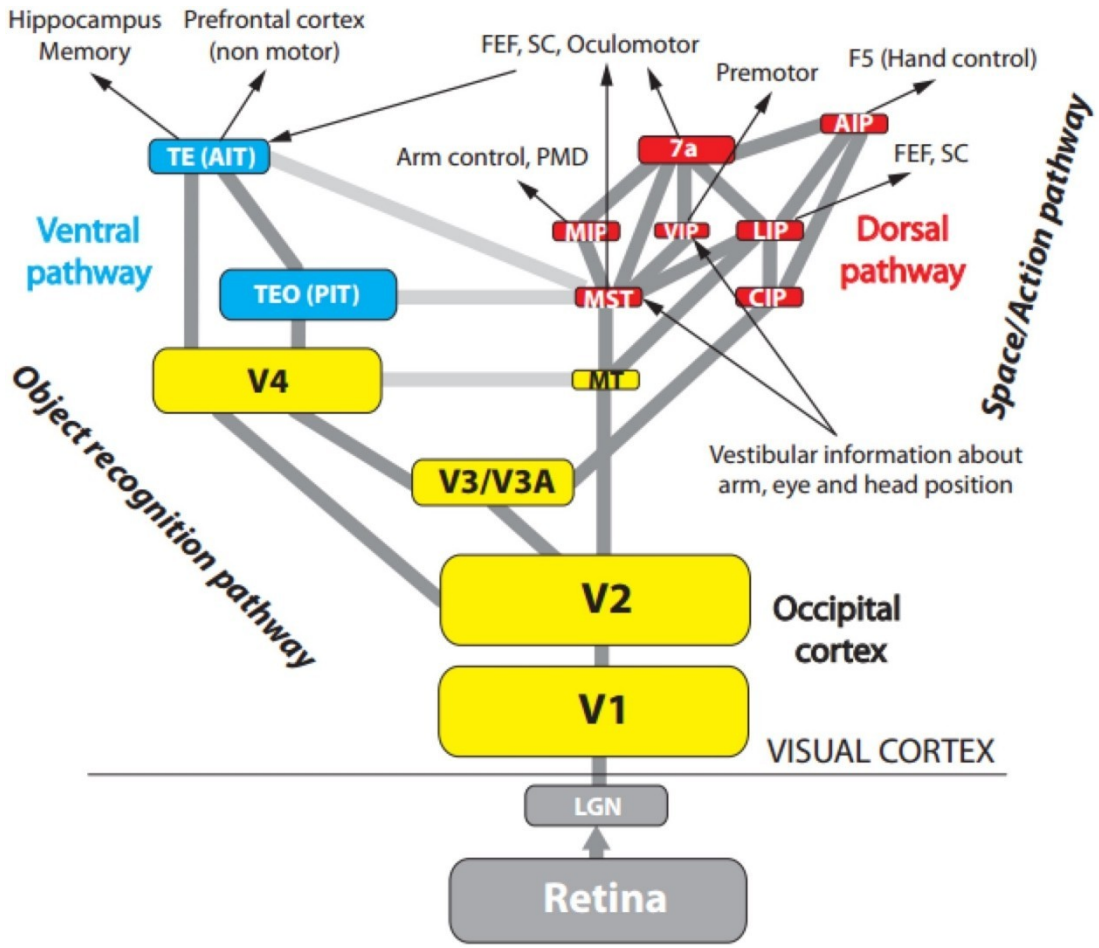
- Questo è un problema perché un piccolo gradiente ⇒
  piccole variazioni dei pesi ⇒ apprendimento lento
  - In caso di underflow numerico, può anche diventare 0 ⇒ nessun apprendimento!

- Per questo motivo, l'uso della sigmoide (o della tanh) non consente di utilizzare molti hidden layers

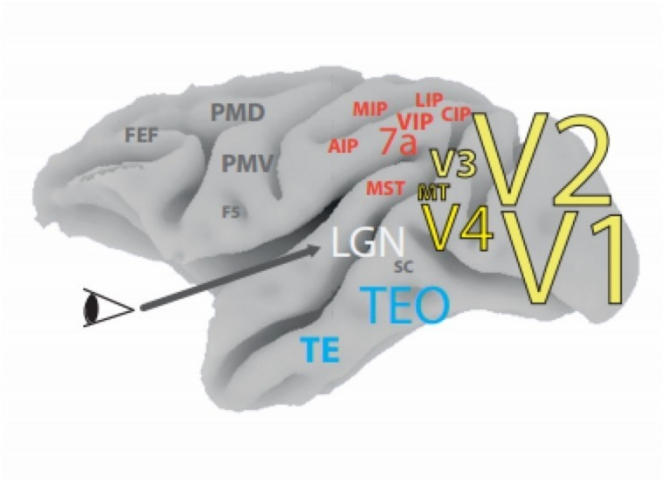
Cosa è cambiato negli anni 2000?

- Migliore comprensione delle reti neurali biologiche
- Sono disponibili set di dati più grandi
- Risorse di calcolo più grandi
- Migliore comprensione del vanishing gradient (e delle relative soluzioni)
- Innovazioni architetturali (ripartizione dei pesi, locali connessioni, layers eterogenei)

Reti biologiche: reti profonde!

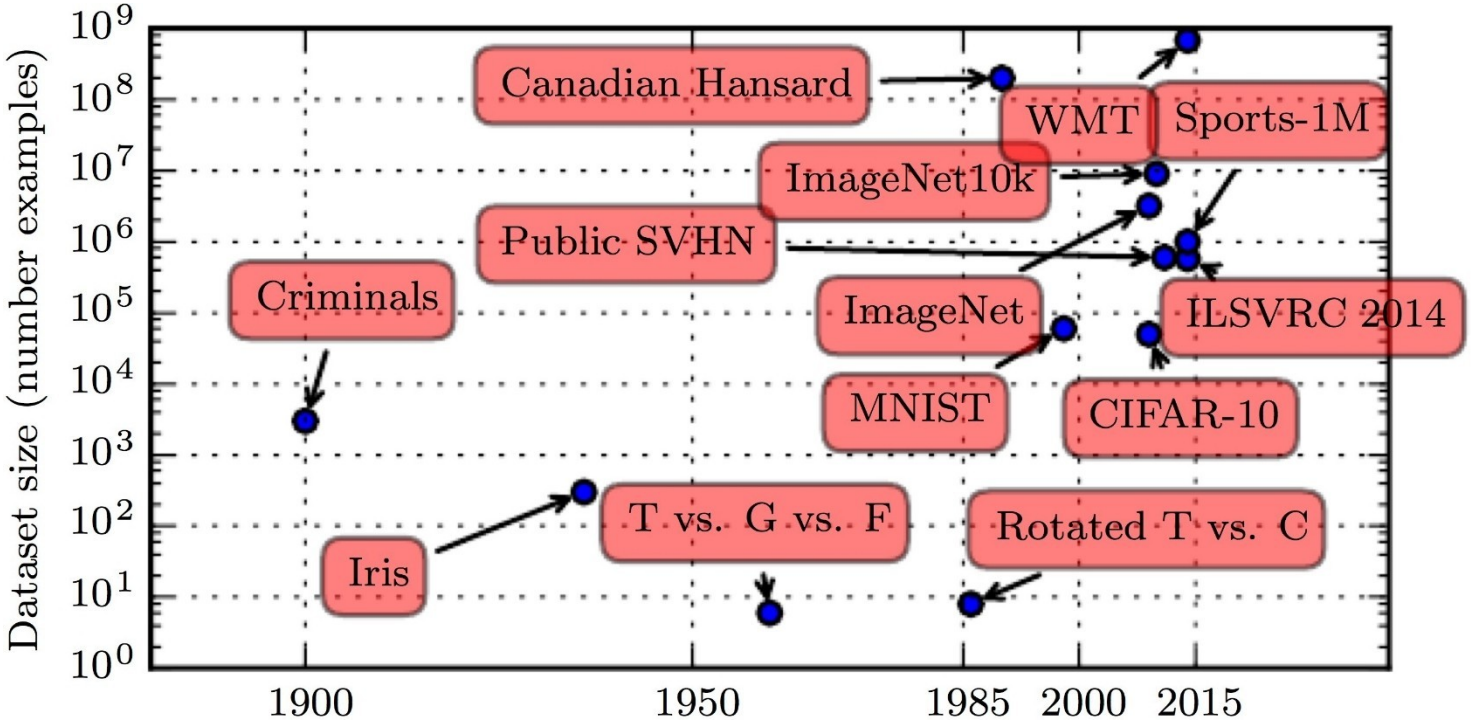


[Kruger et al. 2013]



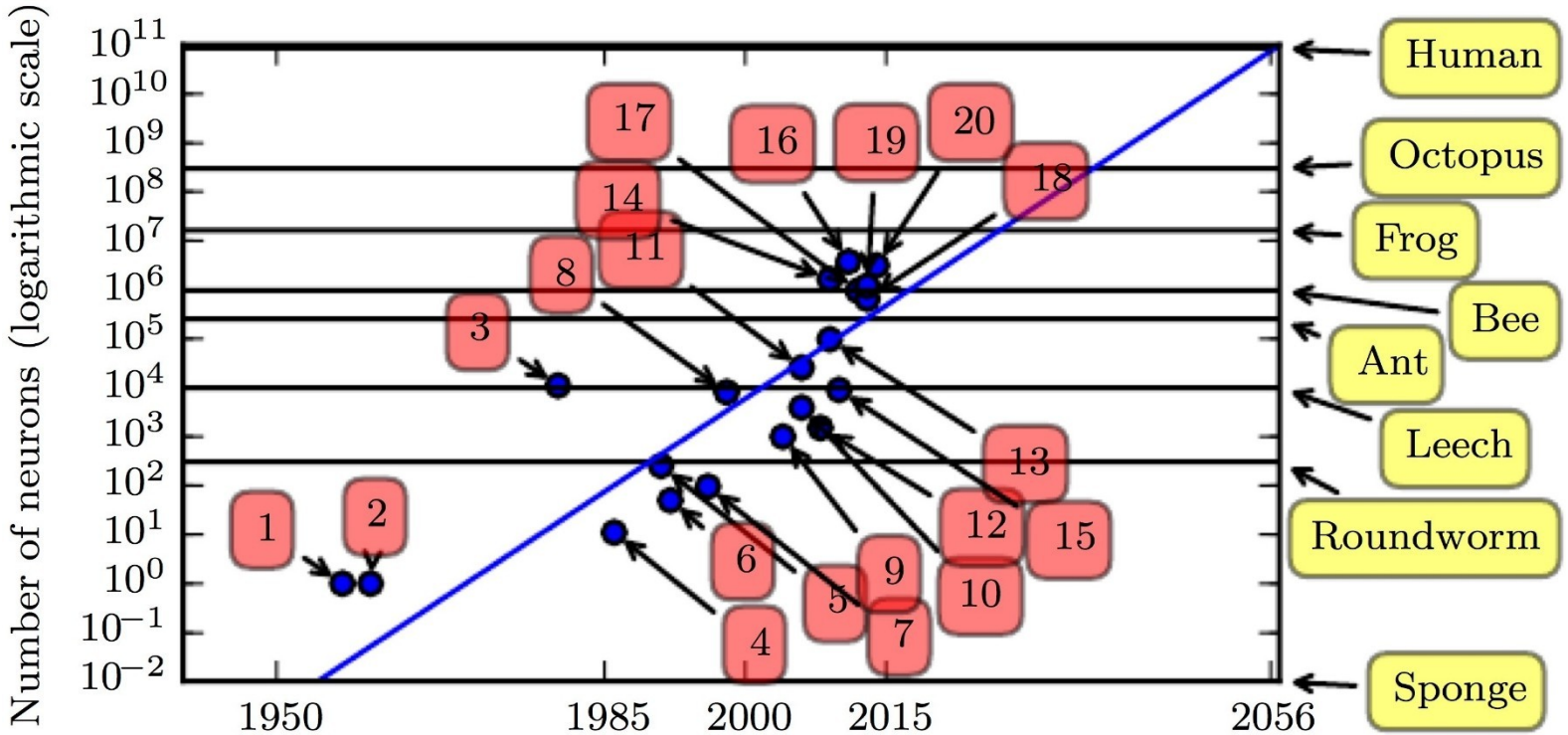
Terza ondata: training set grandi

- Disponibilità di enormi quantità di dati ("Big Data")



Terza ondata: NN di grandi dimensioni

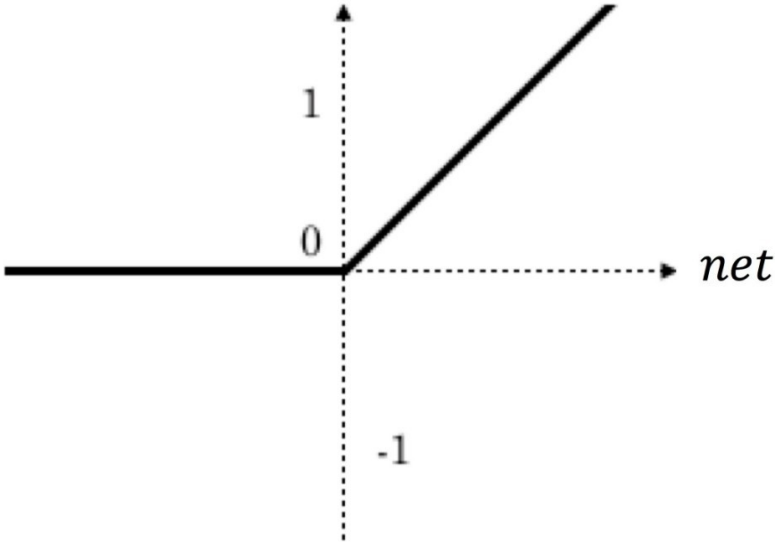
- Aumento della potenza di calcolo e della memoria + elaborazione massicciamente parallela da parte delle GPU = fattibilità di un gran numero di neuroni



Innovazioni della terza ondata: attivazione ReLU

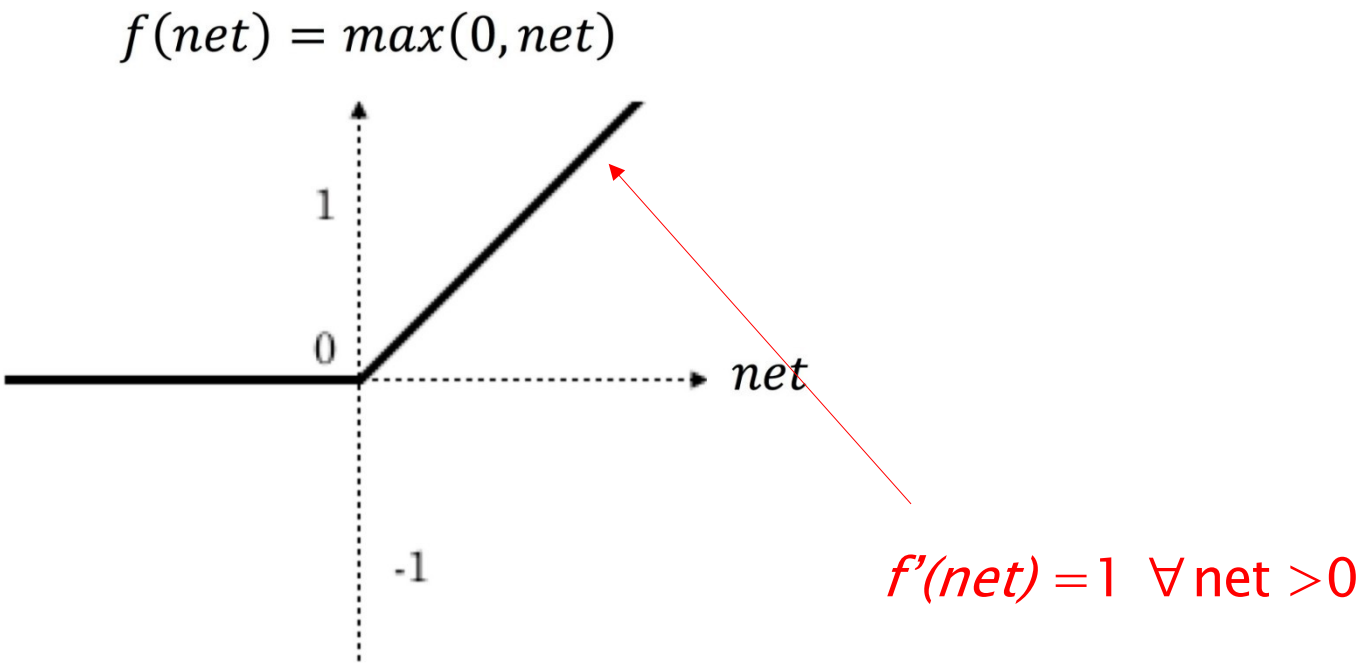
- Unità lineare rettificata (ReLU): una funzione di attivazione diversa (introdotta negli anni '80 ma resa popolare negli anni '90).

$$f(net) = max(0, net)$$



Innovazioni della terza ondata:

- Unità lineare rettificata (ReLU):



- Risolve il problema del vanishing gradient!

Innovazioni della terza ondata: connessioni locali

- Migliore comprensione delle architetture con un numero ristretto di connessioni tra ciascun neurone e il layer precedente
  - Connessioni relative alla localizzazione spazio-temporale nei dati di ingresso
  - Esempio: Reti neurali convoluzionali (LeCun 1998)
- Questo riduce il numero di pesi da addestrare, rendendo possibile l'utilizzo di più livelli e più neuroni

Innovazioni della terza ondata: condivisione del peso

- I neuroni che eseguono concettualmente la stessa operazione su parti diverse dei dati di ingresso possono condividere gli stessi pesi.

- Questo riduce drasticamente il numero di pesi da addestrare, rendendo possibile l'uso di più livelli e più neuroni, soprattutto su input di grande dimensionalità (ad esempio, immagini).

Innovazioni della terza ondata: layers eterogenei

- Nelle NN biologiche, i layers svolgono diverse funzioni
  - Hubel e Wiesel, 1962: La corteccia visiva dei gatti contiene strati di "cellule semplici" (rilevatori di caratteristiche) alternati a strati di "cellule complesse" (fusione di informazioni, garanzia di invarianza spaziale)

- In Reti neurali CNN:
  - Livelli convoluzionali (rilevatori di caratteristiche/feature)
  - Strati di pooling (invarianza spazio-temporale)

Reti profonde

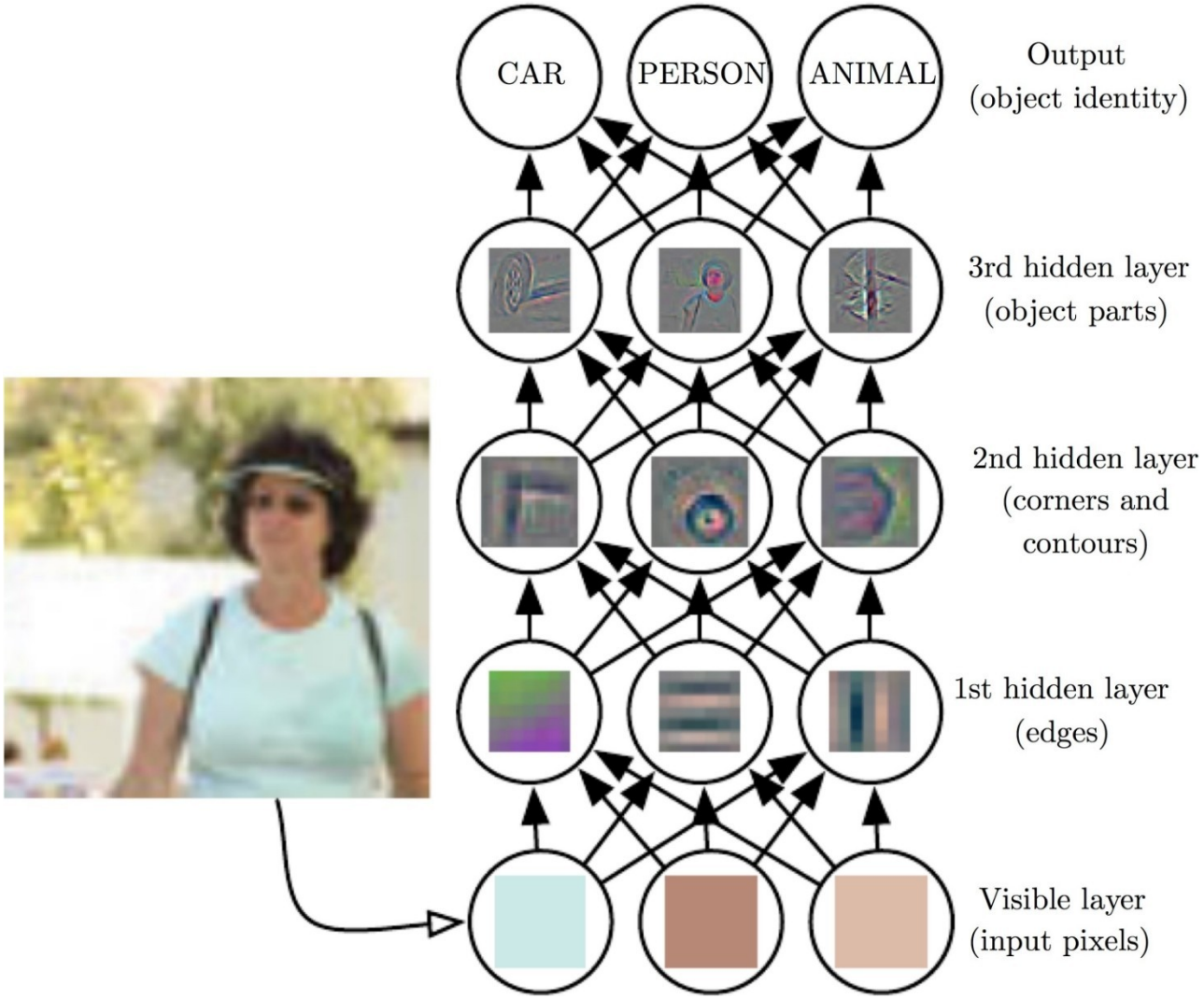
- Le innovazioni precedenti hanno reso possibile la realizzazione e l'addestramento di reti neurali con un elevato numero di layers nascosti (**reti neurali profonde**).
  - Decine o addirittura centinaia di strati sono piuttosto comuni.

Reti profonde: vantaggi?

- Ad ogni layer successivo, la rete può apprendere **modelli** più **complessi** utilizzando la **composizione** di **modelli semplici** riconosciuti nello layer precedente.

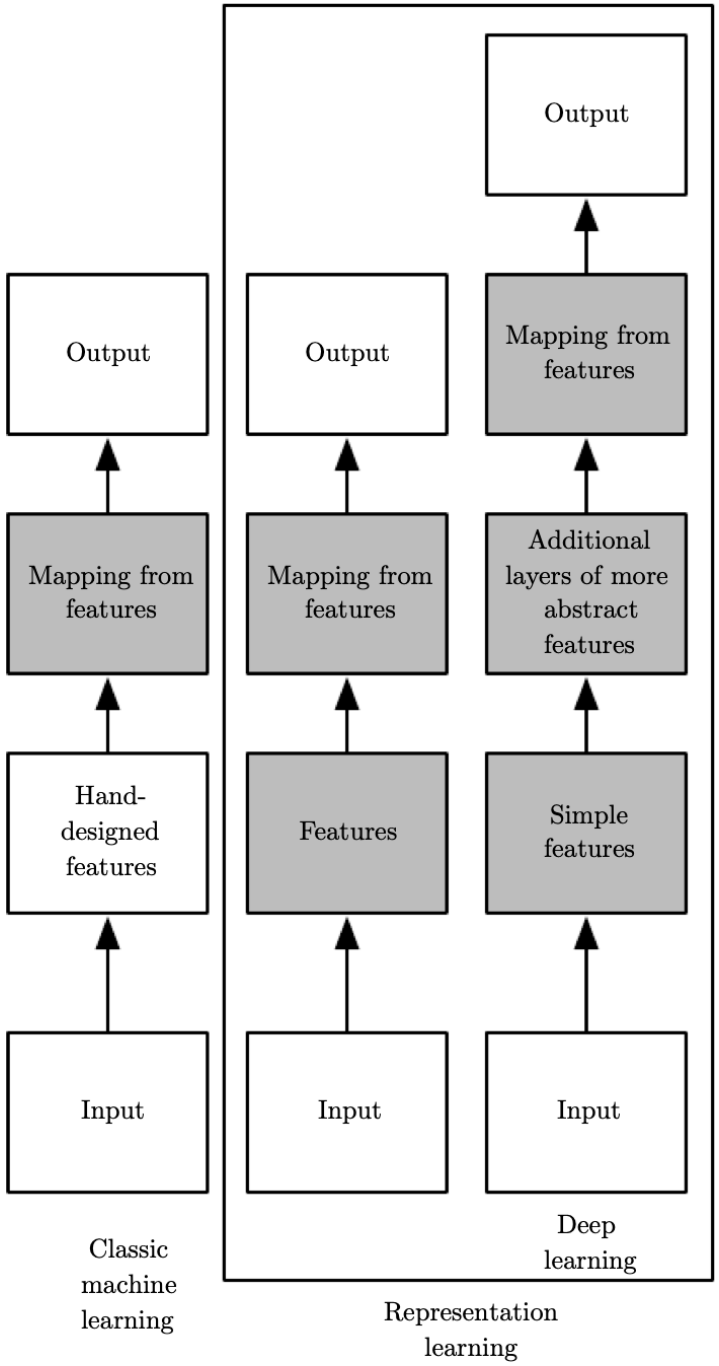
- Pertanto, anche partendo da informazioni di input di livello molto basso (ad esempio, pixel di immagini grezze), la rete potrebbe essere in grado di apprendere strutture complesse (ad esempio, oggetti nell'immagine).

Representation learning



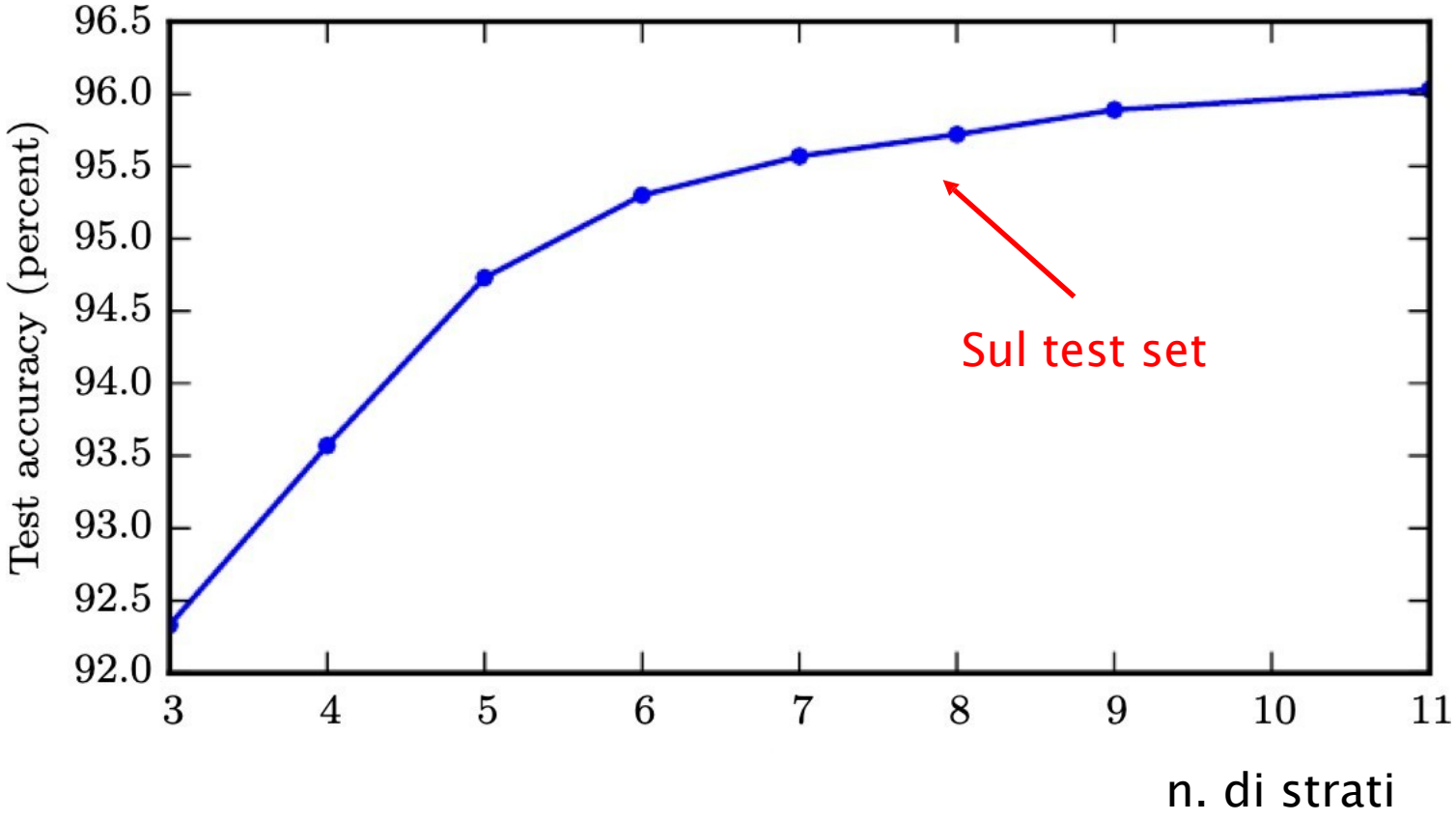
Representation learning

- Una deep network **apprende** la migliore **rappresentazione** (le caratteristiche/feature) per risolvere il suo compito!
  - Non è necessario definire "a mano" quali sono le caratteristiche/feature...



Reti profonde: vantaggi?

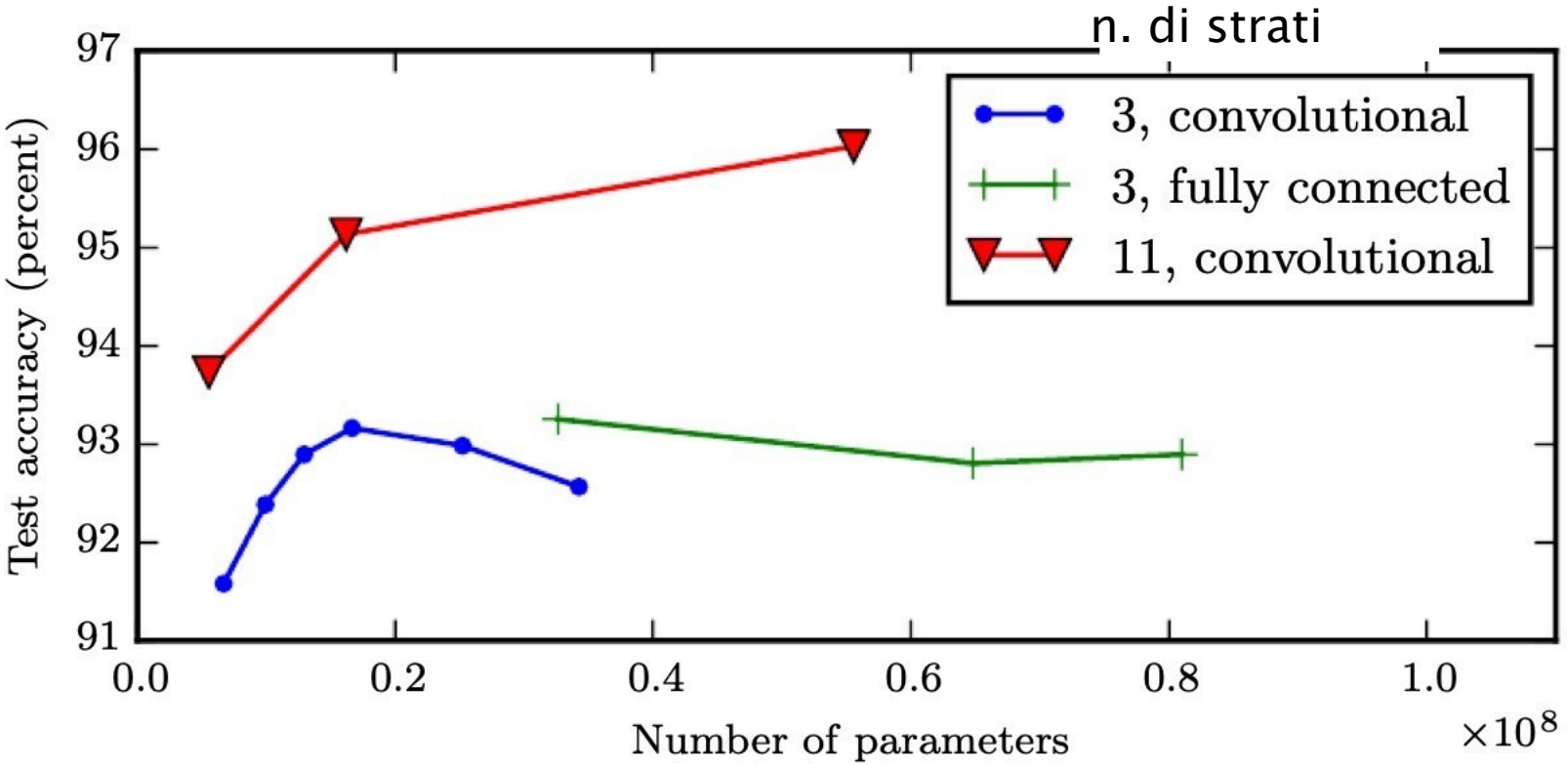
- Migliori prestazioni **di generalizzazione**



Goodfellow et al. 2014: Riconoscimento di numeri a più cifre da immagini StreetView...

Reti profonde: vantaggi?

- Migliori prestazioni rispetto al numero di parametri



Goodfellow et al. 2014: Riconoscimento di numeri a più cifre da immagini StreetView...

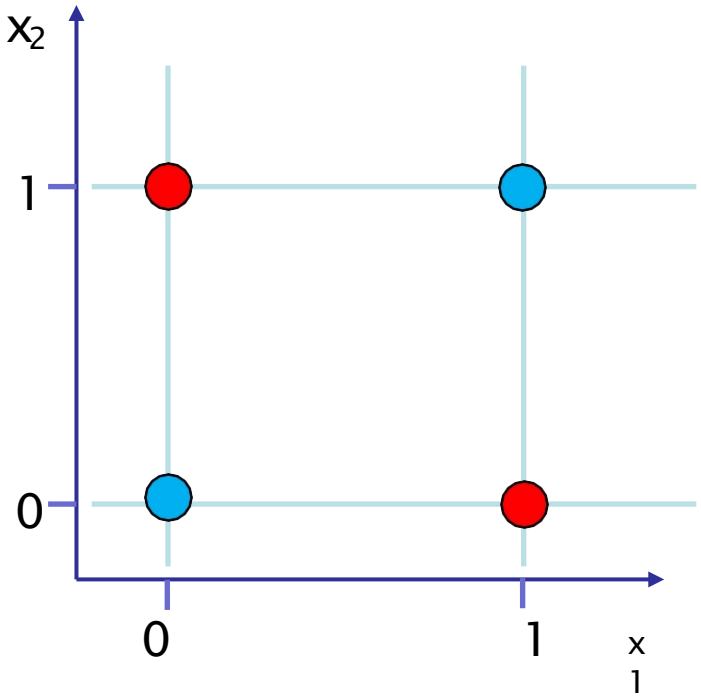
Reti profonde: vantaggi?

- Questo può sembrare controintuitivo
  - Ci aspettiamo che più strati = più complessità = più overfitting

- In un certo senso, una rete profonda incarna l'**assunzione** che la funzione che vogliamo apprendere possa essere ottenuta dalla **composizione di diverse funzioni più piccole.**
  - Pertanto, un deep model funziona meglio quando questa ipotesi è vera (mostrando una migliore generalizzazione).
  - Il teorema del no free-lunch ci ricorda che ci sono problemi per cui questa assunzione deve essere falsa...

Perché le reti profonde

- Un'altra prospettiva: un **singolo neurone** (come abbiamo già detto) può imparare solo a risolvere problemi linearmente separabili



Esempio: la funzione XOR non può essere appresa con un solo neurone.
Qui $y = f(\boldsymbol{x})$ con

$$\boldsymbol{x} = (x_1, x_2)$$

● $\rightarrow y = 0$

● $\rightarrow y = 1$

Perché le reti profonde

- Invece di costruire un neurone "più complesso", possiamo tradurre il nostro spazio di input in uno spazio diverso in cui il problema diventa linearmente separabile

  - Cerchiamo di imparare $f(\phi(\boldsymbol{x}))$, dove $\phi(\boldsymbol{x})$ è un vettore a funzione vettoriale che mappa il nostro vettore d'ingresso **x** in un diverso vettore spazio

  - $\phi(\boldsymbol{x})$ deve essere non lineare (altrimenti la composizione $f(\phi(\boldsymbol{x}))$ avrà ancora le limitazioni di un funzione lineare.

Ma dove troviamo $\phi(\boldsymbol{x})$?

Perché le reti profonde

- Negli approcci tradizionali all'apprendimento automatico, dobbiamo definire manualm$(\phi(x)$:
  - Per farlo dobbiamo essere esperti del dominio dell'applicazione

    (ad esempio, visione computerizzata o

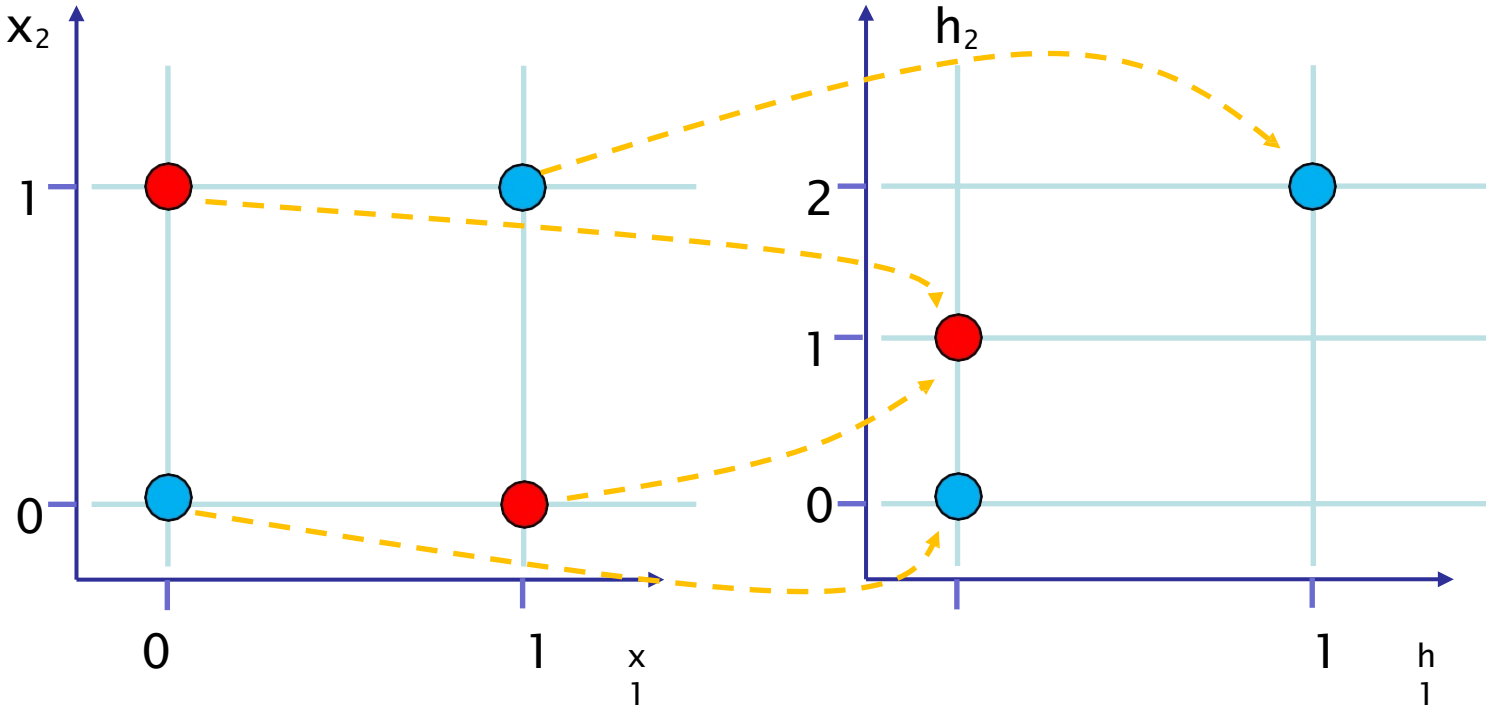
    riconoscimento vocale)

- Un'altra opzione è quella di utilizzare una ϕ molto generica, come le *funzioni radiali di base (Radial Basis Functions),* ***che*** però di solito non si generalizzano bene a compiti complessi.

Perché le reti profonde

- Esempio: per XOR, possiamo definire manualmente:

$$\phi(\boldsymbol{x}) = (h_1, h_2) = (x_1 * x_2, x_1 + x_2)$$



Ora il problema può essere risolto linearmente!

$$f(h) = h_2 - 2 * h_1$$

Perché le reti profonde

- Nell'apprendimento profondo, **impariamo** invece $\phi(x)$ efinirlo manualmente.
  - I l**ayer nascosti** della rete diventano il nostro
    - Sce ϕ per uno hidden layer una struttura parametrica molto generica: la composizione tra una funzione lineare e una funzione di attivazione non lineare.
    - Possiamo rappresentare molto complicati semplicemente applicando più strati nascosti... $\phi(x)$

Perché le reti profonde

- Nel nostro esempio di XOR, possiamo scegliere uno strato nascosto modellato come:

$$\phi(\boldsymbol{x}) = ReLU(\boldsymbol{W} \cdot \boldsymbol{x} + \boldsymbol{b})$$

Funzione di attivazione

Matrice dei pesi (parametri)

Vettore Bias (altri parametri)

Nota: l'algoritmo di apprendimento sceglierà W e b per implementare al meglio la funzione f desiderata, *anche* se i dati di addestramento non danno il valore desiderato di ϕ

Perché le reti profonde

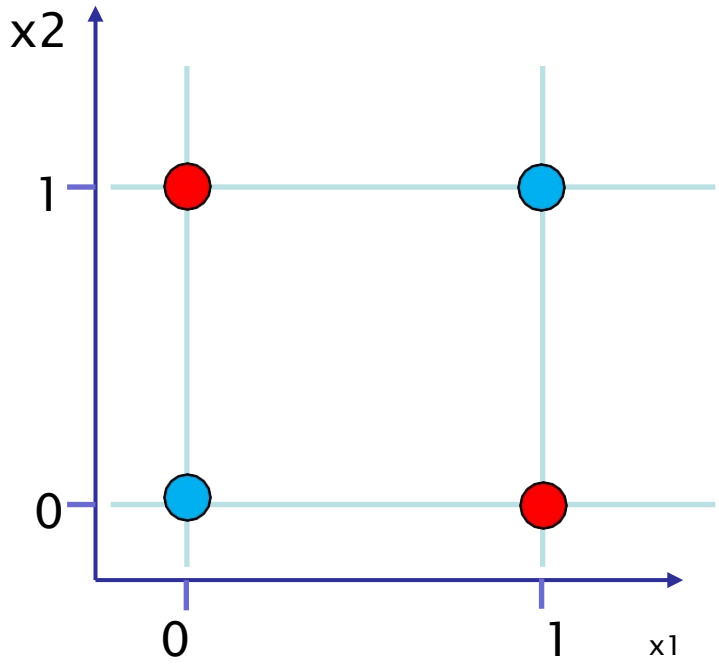
- Nel nostro esempio di XOR, possiamo scegliere uno hidden layer modellato come:

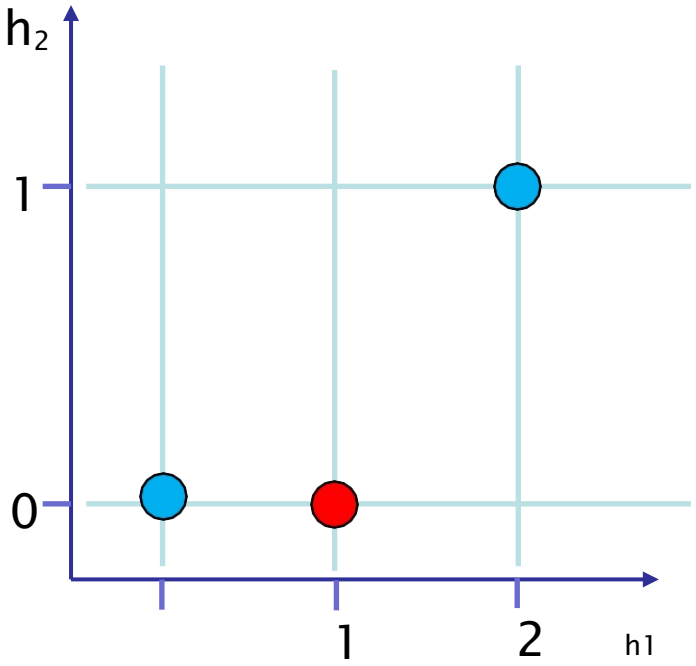
$$\phi(\boldsymbol{x}) = ReLU(\boldsymbol{W} \cdot \boldsymbol{x} + \boldsymbol{b})$$

- Per esempio, l'algoritmo può scegliere (eseguendo un numero sufficiente di cicli di addestramento) :
  - $\boldsymbol{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$
  - $\boldsymbol{b} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$

Perché le reti profonde

- Con questa definizione di $\boldsymbol{h} = \phi(\boldsymbol{x})$





Ora il problema può essere risolto linearmente!

$$f(h) = h1 - 2 * h_2$$

Transfer Learning

- I layers inferiori di una deep network imparano una rappresentazione intermedia che è utile per risolvere il compito della rete.

- E se dovessimo risolvere un problema diverso ma **simile?**

  - Probabilmente, le caratteristiche/feature intermedie saranno utili anche per il nuovo problema

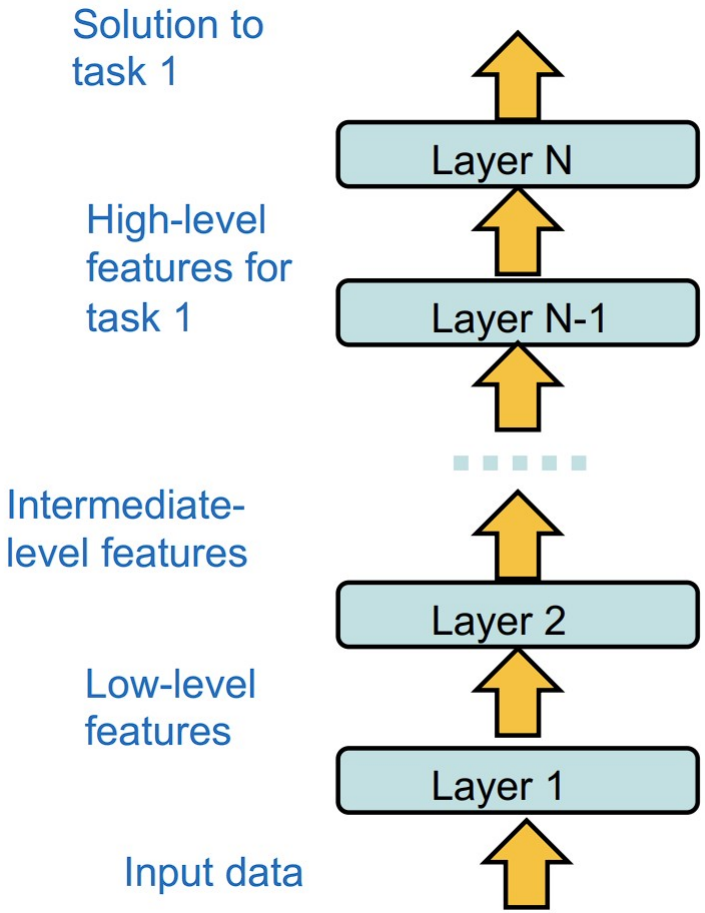
Transfer Learning

- Possiamo sfruttare questo fatto riutilizzando i layers già addestrati per costruire una nuova rete.
  - I layers più alti della rete vengono sostituiti con un nuovo NN
  - La rete risultante viene addestrata sul nuovo problema (**Fine Tuning**).

- In questo modo, **trasferiamo** alcune conoscenze apprese per il primo compito alla soluzione del secondo.

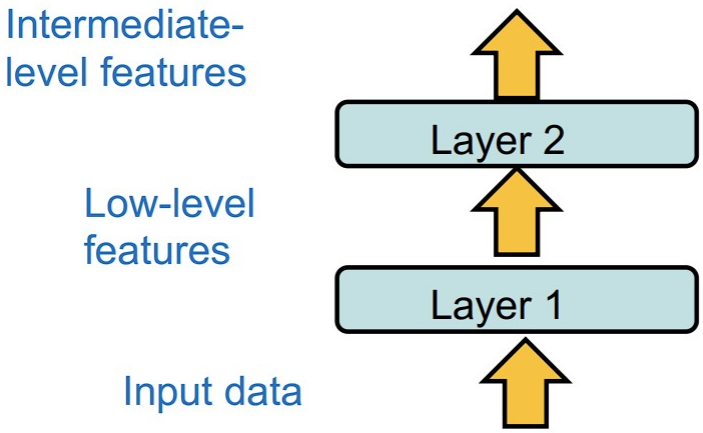
Transfer Learning

- Rete addestrata per risolvere il compito 1:



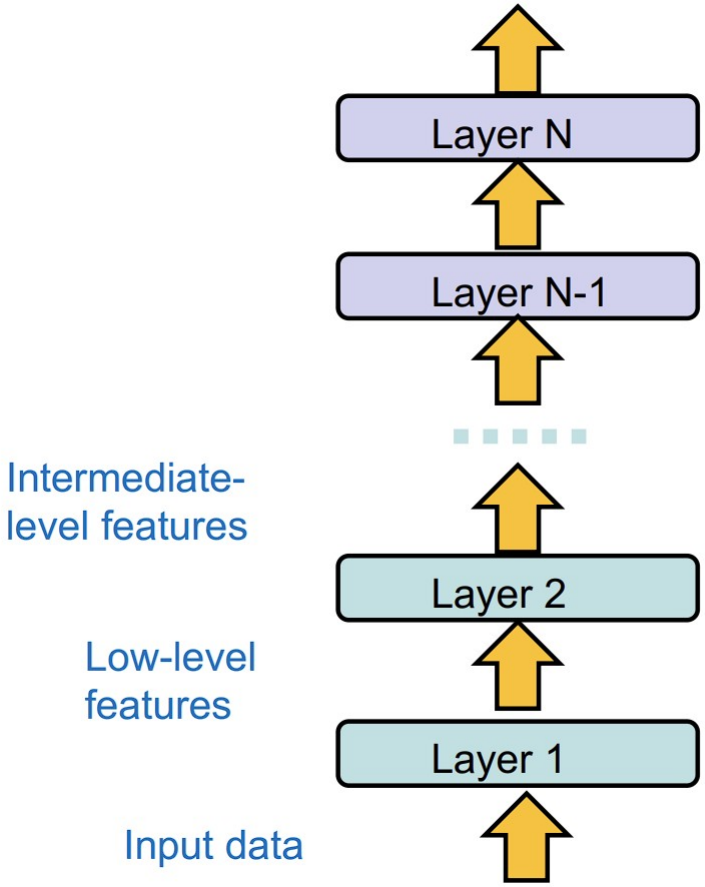
Transfer Learning

- Rimuoviamo alcuni dei layers superiori:



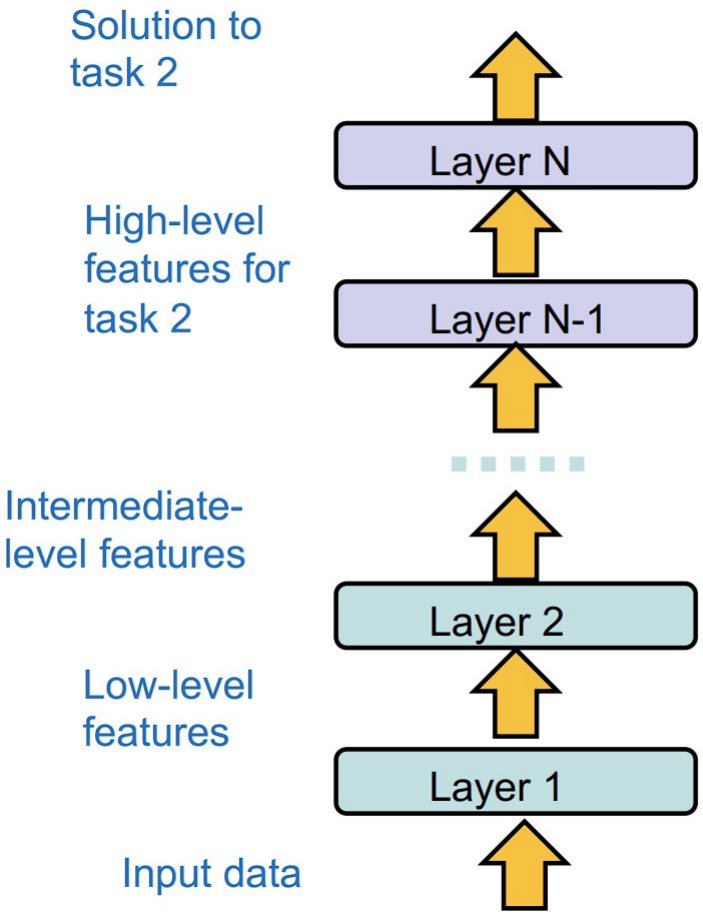
Transfer Learning

- Aggiungiamo diversi layers superiori:



Transfer Learning

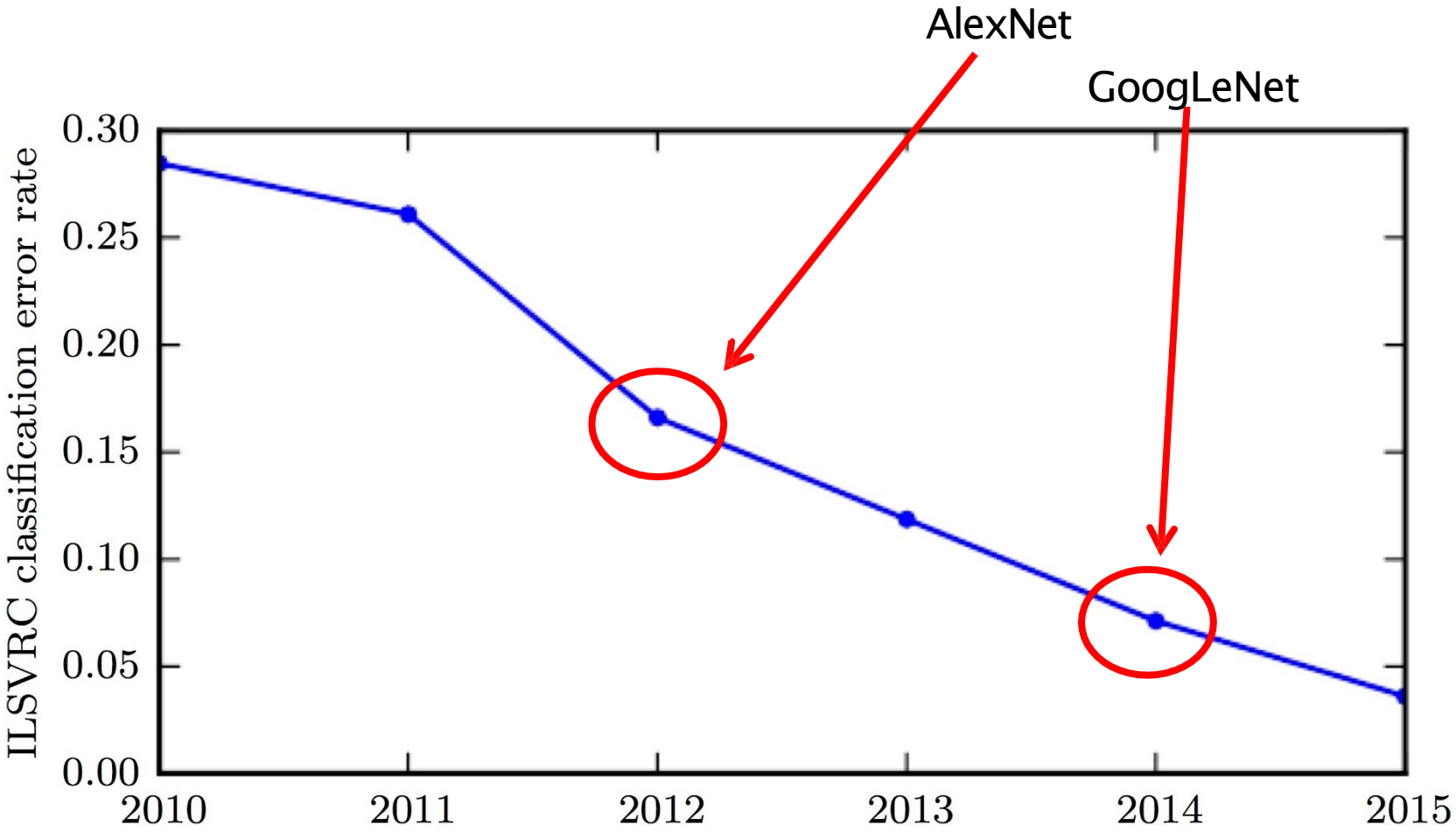
- Addestriamo la rete risultante sul compito 2:



Apprendimento per trasferimento: vantaggi

- Riutilizzo delle conoscenze per problemi simili
- Risparmio di tempo: parte della rete è già addestrata!
- Per il nuovo problema è possibile utilizzare un dataset più piccolo (poiché ci sono meno pesi da apprendere).
  - Caso estremo: **one shot learning**: solo un esempio fornito per il compito 2 (la parte aggiuntiva della rete deve essere molto semplice!)

Prestazioni eccezionali in situazioni difficili



Sfida di riconoscimento visivo ImageNet su larga scala

Prestazioni eccezionali in situazioni difficili

- Negli ultimi 10 anni, le deep networks hanno migliorato in modo consistente e significativo le prestazioni dello stato dell'arte in compiti considerati molto difficili.
  - Classificazione delle immagini
  - Rilevamento e riconoscimento degli oggetti
  - Riconoscimento vocale
  - Comprensione del linguaggio naturale
  - Traduzione linguistica
  - Modifica di immagini/video
  - . . .