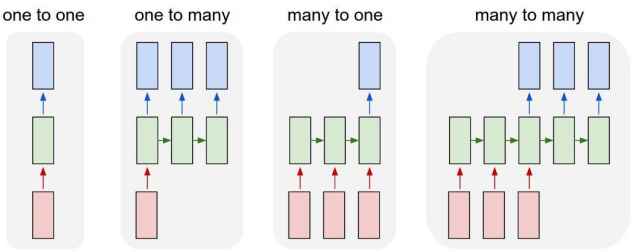
Deep Learning

Sequence to Sequence



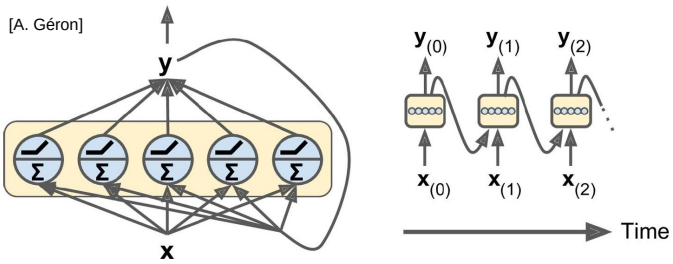
Le reti feedforward (es. MLP, CNN) studiate fino a questo momento operano su vettori di dimensione prefissata. Ad esempio l'input è un vettore immagine e l'output un vettore di probabilità delle classi. Questo caso è rappresentato dalla prima colonna della figura come sequenza One to one.

Applicazioni diverse richiedono che l'input e/o l'output possano essere sequenze (anche di lunghezza variabile).

- One to many. Es. image captioning dove l'input è un'immagine e l'output una frase (sequenza di parole) in linguaggio naturale che la descrive.
- Many to one. Es. sentiment analysis dove l'input è un testo (es. recensione di un prodotto) e l'output un valore continuo che esprime il sentiment o giudizio (positivo o negativo).
- Many to many. Es. language translation dove l'input è una frase in Inglese e l'output la sua traduzione in Italiano.

Reti Ricorrenti (RNN)

[A. Géron]



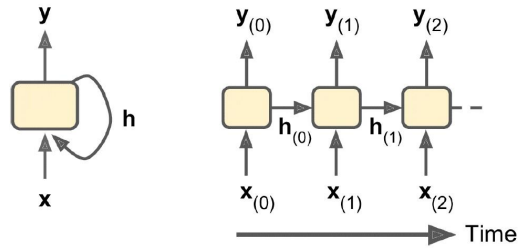
Le reti ricorrenti prevedono «anche» collegamenti all'indietro o verso lo stesso livello. I modelli più comuni e diffusi (es. LSTM, GRA) prevedono collegamenti verso lo stesso livello.

A ogni step della sequenza (ad esempio a ogni istante temporale t) il livello riceve oltre all'input $\mathbf{x}_{(t)}$ anche il suo output dello step precedente $\boldsymbol{y}_{(t-1)}$. Questo consente alla rete di basare le sue decisioni sulla storia passata (effetto memoria) ovvero su tutti gli elementi di una sequenza e sulla loro posizione reciproca.

- In una frase non è rilevante solo la presenza di specifiche parole ma è importante anche come le parole sono tra loro legate (posizione reciproca).
- La classificazione di video (sequenze di immagini) non necessariamente si basa sulle interrelazioni dei singoli frame. Esempio:
  - *per capire se un video è un documentario su animali è sufficiente la detection di animali in qualche frame (non occorre RNN).*
  - *per comprendere il linguaggio dei segni, è necessario analizzare le interrelazioni dei movimenti delle mani nei frame (utile RNN).*

Unfolding in Time

[A. Géron]



Una cella è una parte di rete ricorrente che preserva uno stato (o memoria) interno $\mathbf{h}_{(t)}$ per ogni istante temporale. È costituita da un numero prefissato di neuroni (può essere vista come un layer).

- $\mathbf{h}_{(t)}$ dipende dall'input $\mathbf{x}_{(t)}$ e dallo stato precedente $\mathbf{h}_{(t-1)}$

$$\mathbf{h}_{(t)} = f(\mathbf{h}_{(t-1)}, \mathbf{x}_{(t)})$$

Per poter addestrare (con backpropagation) una RNN è necessario eseguire il cosiddetto unfolding o unrolling in time (parte destra della figura), stabilendo a priori il numero di passi temporali su cui effettuare l'analisi.

- Di fatto una RNN unfolded su 20 step equivale a una DNN feedforward con 20 livelli. Pertanto addestrare RNN che appaiono relativamente semplici può essere molto costoso e critico per la convergenza (problema del vanishing gradient).
- Da notare che in una RNN unfolded: l'input e l'output sono in generale collegati a tutte le istanze della cella e i pesi della cella (nascosti in f) sono comuni a tutte le istanze della cella.

Basic Cell

In una cella base di RNN:

Lo stato $\mathbf{h}_{(t)}$ dipende dall'input $\mathbf{x}_{(t)}$ e dallo stato precedente $\mathbf{h}_{(t-1)}$

$$\mathbf{h}_{(t)} = \phi\left(\mathbf{x}_{(t)}{}^{T} \cdot \mathbf{W}_{x} + \mathbf{h}_{(t-1)}{}^{T} \cdot \mathbf{W}_{h} + b\right)$$

dove:

- $\mathbf{W}_x$ e $\mathbf{W}_y$ sono i vettori dei pesi e b il bias da apprendere.
- ϕ è la funzione di attivazione (es. Relu).

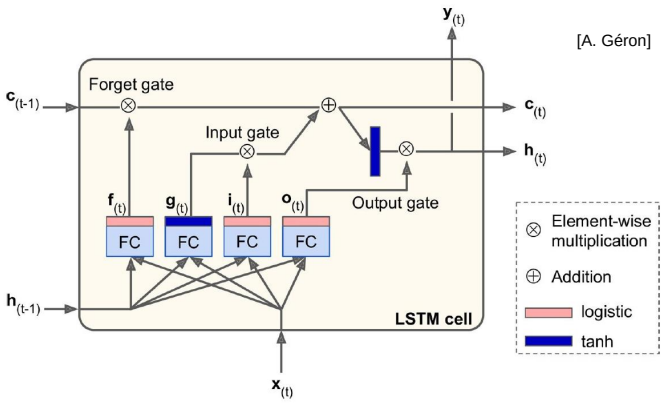
e l'output $\mathbf{y}_{(t)}$ corrisponde allo stato $\mathbf{h}_{(t)}$:

$$\mathbf{y}_{(t)} = \mathbf{h}_{(t)}$$

Le celle base hanno difficoltà a ricordare/sfruttare input di step lontani: la memoria dei primi input tende a svanire. D'altro canto sappiamo che in una frase anche le prime parole possono avere un'importanza molto rilevante.

Per risolvere questo problema e facilitare la convergenza in applicazioni complesse, sono state proposte celle più evolute dotate di un effetto memoria a lungo termine: LSTM e GRU sono le più note tipologie.

LSTM

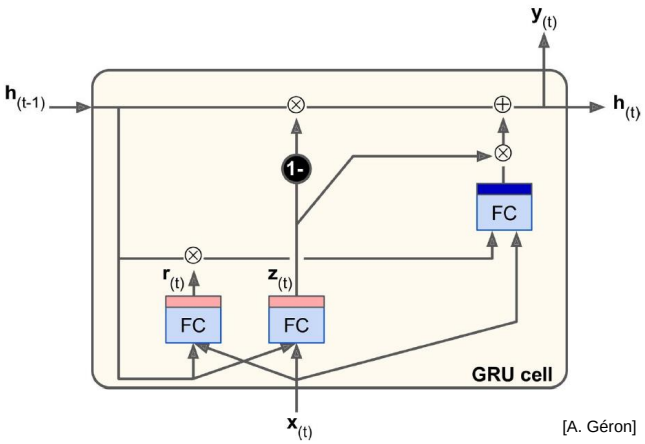


[A. Géron]

In LSTM (Long Short-Term Memory) lo stato $\mathbf{h}_{(t)}$ è suddiviso in due vettori $\mathbf{h}_{(t)}$ e $\mathbf{c}_{(t)}$:

- $\mathbf{h}_{(t)}$ è uno stato (o memoria) a breve termine; anche in questo caso uguale all'output della cella $\mathbf{y}_{(t)}$.
- $\mathbf{c}_{(t)}$ è uno stato (o memoria) a lungo termine.
- la cella apprende durante il training cosa è importante dimenticare (forget gate) dello stato passato $\mathbf{c}_{(t-1)}$ e cosa estrarre e aggiungere (input gate) dall'input corrente $\mathbf{x}_{(t)}$.
- per il calcolo dell'output $\mathbf{y}_{(t)} = \mathbf{h}_{(t)}$ si combina l'input corrente (output gate) con informazioni estratte dalla memoria a lungo termine.

GRU



[A. Géron]

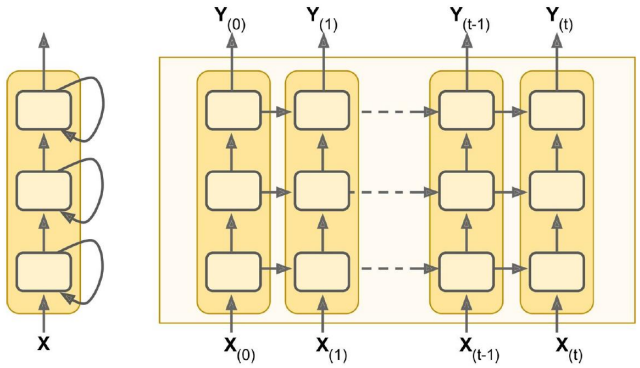
GRU (Gated Recurrent Unit) è una versione semplificata di LSTM, che cerca di mantenerne i vantaggi, riducendo parametri e complessità. Le principali semplificazioni sono:

- uno solo stato di memoria $\mathbf{h}_{(t)}$.
- uno solo gate (con logica invertita) per quantificare quanto dimenticare e quanto aggiungere: se necessario aggiungere prima devo dimenticare.

Per maggiori approfondimento su LSMT e GRU:

- http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Deep RNN



[A. Géron]

In figura una RNN costituita da tre layer (stacked): a destra la versione unfolded (su t+1 step).

- gli output di un livello sono gli input del livello successivo
- con più livelli possono essere apprese relazioni più complesse dei dati.
- l'addestramento (supervisionato) consiste sempre nel fornire gli input e gli output (desiderati) per il solo ultimo livello.
- se l'input sono immagini, i vettori $\mathbf{x}_{(t)}$ in genere non corrispondono ai pixel ma a feature di alto livello estratte da una CNN.
- se l'input sono parole, i vettori $\mathbf{x}_{(t)}$ sono il risultato di word embedding (vedi nel seguito).

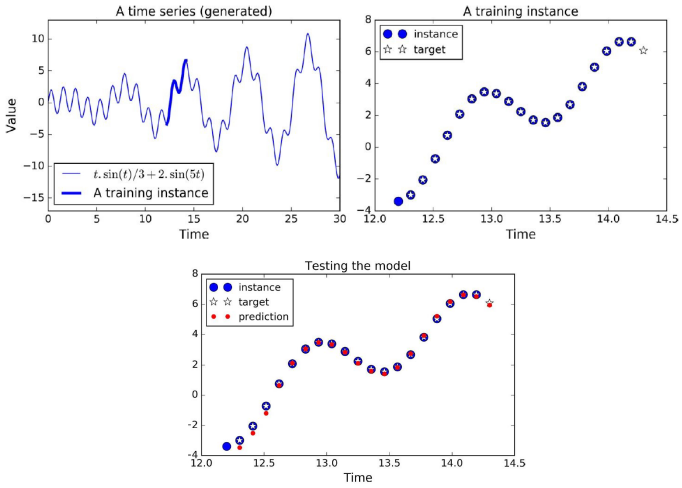
Esempio: Time Series

Una serie temporale (time serie) può essere relativa all'andamento nel tempo: di un titolo di borsa, della temperatura di un ambiente, del consumo energetico di un impianto, ecc.

Possiamo considerare una serie temporale come una funzione campionata in più istanti temporali. Conoscendo i valori fino all'istante t siamo interessati a predire il valore a $t+1$. Nota bene: non solo input 1D.

In [A. Géron] esempio in Tensorflow/Keras di RNN per la predizione di una funzione matematica (combinazione di sinusoidi).

- https://github.com/ageron/handson-ml2



Note su Time Series

Prima di proporre modelli complessi (con il serio rischio di overfitting) provare semplici baseline, valutandone le prestazioni e la generalizzazione:

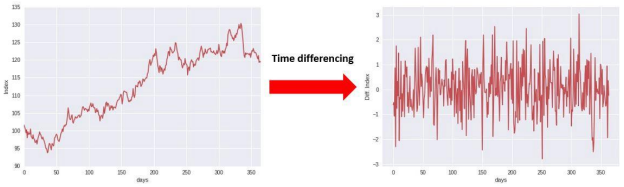
- Persistence model: la predizione al tempo $t+1$ corrisponde al valore al tempo t.
- Linear model: un semplice regressore lineare.

Modelli statistici classici: le serie temporali sono state studiate per decenni in ambito statistico ed esistono molti metodi (relativamente semplici) e talvolta preferibili a reti ricorrenti:

- https://machinelearningmastery.com/time-series-forecasting-methods-in-python-cheat-sheet/

Decorrelazione dei dati della sequenza: talvolta, se i valori di una sequenza sono correlati nel tempo, si ha l'impressione di riuscire a predire con grande successo.

- Esempio: trend di un titolo di borsa. Anche il persistence model sembra performare bene rispetto alle metriche classiche (es. MSE). Proviamo a decorrelare i dati sostituendo al valore in t la differenza tra il valore in t e quello a $t-1$.



https://towardsdatascience.com/how-not-to-use-machine-learning-for-time-series-forecasting-avoiding-the-pitfalls-19f9d7adf424

Natural Language Processing (NLP)

L'elaborazione del linguaggio naturale è uno dei settori cui il deep learning ha dato i maggiori contributi. Tra le principali applicazioni:

- Language Modeling (generativi): es. generare testi
- Text Classification: es. sentiment analysis
- Automatic Speech Recognition: es. traduzione parlato in testo
- Caption Generation: es. data un'immagine generare una descrizione in linguaggio naturale.
- Machine Translation: es. traduzione tra lingue, creare riassunti di documenti, generare commenti nel codice.
- Question Answering: es. rispondere in linguaggio naturale a domande poste in linguaggio naturale.
- Digital Assistant: assistente digitale con cui interagire attraverso domande poste in linguaggio naturale (Amazon Alexa, Google assistant, Apple Siri, ecc.)

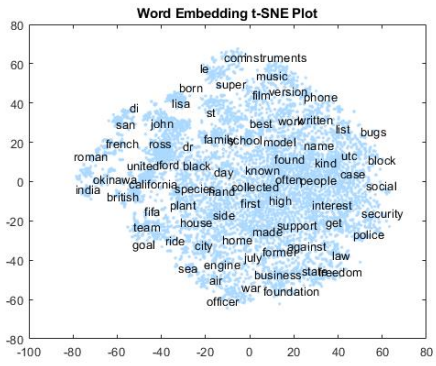
Word embedding

Rappresentazione one-hot-vector:

- Dato un dizionario di dimensione m parole (es. 50000), la parola corrispondente alla k-esima entry è codificata con un vettore di dimensione m, dove tutti gli elementi assumono valore 0 tranne l'elemento in posizione k che assume valore 1.
- Questa rappresentazione non è però utile perché non codifica la semantica e la similarità tra parole (es. sinonimi).

La tecnica word embedding associa a ogni parola del dizionario un vettore di dimensionalità contenuta (es. 150). Parole di significato simile sono associate a vettori vicini nello spazio.

- Word2Vec è un modello di rete neurale (a due livelli) che può essere addestrato (unsupervised) per produrre l'embedding a partire da un corpus di testi.



Language Models (generativi)

Un modello di linguaggio (character based) può essere ottenuto con una RNN in modo abbastanza semplice a partire da un corpus di testi (es. La Divina Commedia):

- Trattiamo un testo come un sequenza temporale di caratteri ed estraiamo sequenze di lunghezza prefissata (es. 21 caratteri).
- Addestriamo una RNN a predire il 21-simo carattere dati i primi 20.
- Al termine dell'addestramento il modello può essere utilizzato in modo creativo per generare testi: si parte con uno o più caratteri random e si chiede alla rete di fare predizione; si appende l'ultimo carattere generato alla stringa corrente e si passano al modello gli ultimi 20 caratteri. Si itera in questo modo fino ad ottenere testo di lunghezza desiderata.

I risultati possono essere sorprendenti, vedi:

- http://karpathy.github.io/2015/05/21/rnn-effectiveness/

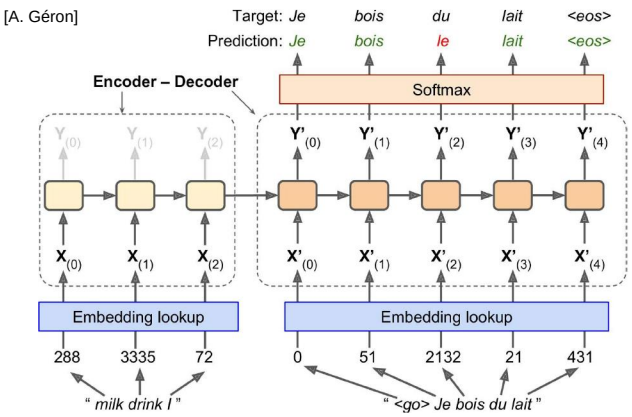
Modelli di linguaggio (word based) sono più potenti ma più complessi: richiedono parsing, stemming, word embedding.

Lo stato dell'arte (2020) è GPT-3 un language model con 175 miliardi di parametri addestrato da OpenAI. Il costo stimato dell'addestramento (su un AI supercomputer) è di 12 M$.

- sebbene il teso prodotto da GPT-3 sia a volte stupefacente, il modello non supera il test di Turing:

  https://link.springer.com/article/10.1007/s11023-020-09548-1

Neural Machine Translation



Elaborazione di sequenze in modalità many-to-many (lunghezza variabile). La traduzione da una lingua a un'altra è l'applicazione più nota (e tra le più complesse).

- I modelli (storicamente) più utilizzati sono architetture RNN di tipo Encoder-Decoder con meccanismi di attenzione (per ogni parola assegna peso alle altre parole per essa importanti nella frase). Vedi Seq2Seq per TensorFlow.
- Nel 2017 Goggle ha introdotto BERT superando lo stato dell'arte. BERT è basato su modello Tranformer, che non usa RNN ma CNN con sofisticati meccanismi di attenzione.

  https://towardsdatascience.com/transformers-141e32e69591

Digital Assistant

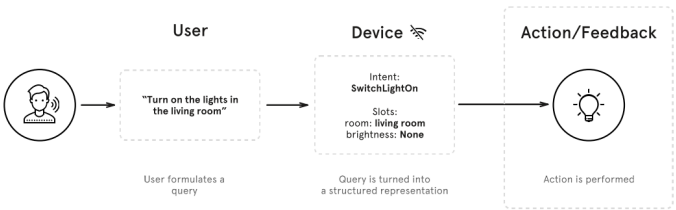
Sviluppare servizi (skill) basati su Amazon Alexa è piuttosto semplice attraverso le API messe a disposizione da Amazon. Stesso dicasi per Google.

Attenzione però alla privacy:

- Il device (locale) acquisisce la traccia vocale, la impacchetta e invia il tutto ai server remoti Amazon, che traducono la voce in testo, elaborano la richiesta e inviano il risultato.
- Vari scandali recenti per Siri (trascrizioni da parte di operatori) e Alexa (condivisione involontaria di dati).

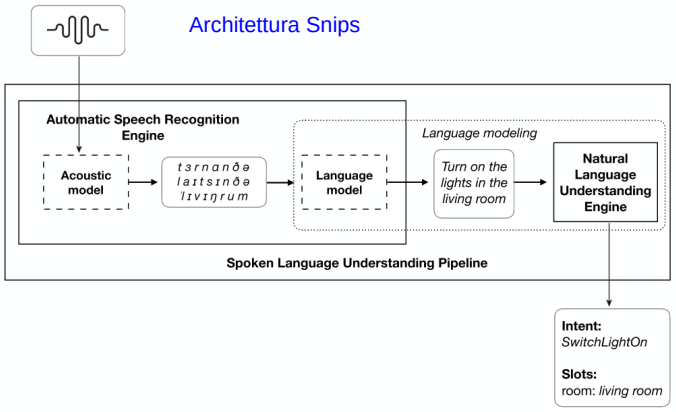
Fortunatamente, si stanno diffondendo alcune piattaforme come Snips, che consentono di operare in locale, vedi:

- "*Snips Voice Platform: an embedded Spoken Language Understanding system for private-by-design voice interfaces*" by A. Coucke et al., 2018.



Digital Assistance (2)

Architettura Snips



- Automatic Speech Recognition (ASR): include acoustic model (che dal segnale audio grezzo passa ai fonemi) e language model (dai fonemi alle parole presenti in un vocabolario e loro aggregazione in frasi).

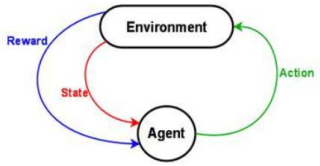
  Progetto Mozilla Commonvoice per creare un dataset aperto per il training ASR (in italiano): contribuite!

- Language modeling: a partire dall'output del modello acustico estrae intent (tipo di query) e slots (parti variabili della query). Il language model considera la probabilità di co-occorrenza di parole per passare da fonemi/parole a frasi probabili. Il Natural language understanding engine utilizza diversi componenti tra cui parser e classificatori.

Reinforcement Learning

L'obiettivo è apprendere un comportamento ottimale a partire dalle esperienze passate.

- Un agente esegue azioni (a) che modificano l'ambiente, provocando passaggi da uno stato (s) all'altro. Quando l'agente ottiene risultati positivi riceve una ricompensa o reward (r) che però può essere temporalmente ritardata rispetto all'azione, o alla sequenza di azioni, che l'hanno determinata.



- Un episodio (o game) è una sequenza finita di stati, azioni, reward:

$$s_0, a_0, r_1, s_1, a_1, r_2, s_2 \dots, s_{n-1}, a_{n-1}, r_n, s_n$$

- In ciascun stato s_{t-1}, l'obiettivo è scegliere l'azione ottimale a_{t-1}, ovvero quella che massimizza il future reward R_t:

$$R_t = r_t + r_{t+1} + \cdots + r_n$$

- In molte applicazioni reali l'ambiente è stocastico (i.e., non è detto che la stessa azione determini sempre la stessa sequenza di stati e reward), pertanto applicando la logica del «*meglio un uovo oggi che una gallina domani*» si pesano maggiormente i reward temporalmente vicini (discounted future reward):

$$R_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \cdots + \gamma^{n-t} \cdot r_n \quad \text{(con } 0 \le \gamma \le 1\text{)}$$

Q-Learning

- Il discounted future reward può essere definito ricorsivamente:

$$R_t = r_t + \gamma \cdot R_{t+1}$$

- Nel Q-learning la funzione $Q(s,a)$ indica l'ottimalità (o qualità) dell'azione a quando ci si trova in stato s. Volendo massimizzare il discounted future reward si pone:

$$Q(s_t, a_t) = max\, R_{t+1}$$

- Nell'ipotesi che la funzione $Q(s,a)$ sia nota, quando ci si trova in stato s, si può dimostrare che la policy ottimale è quella che sceglie l'azione a^* tale che:

$$a^* = \underset{a}{arg\,max}\, Q(s,a)$$

- Il punto cruciale è dunque l'apprendimento della funzione Q. Data una transizione (quaterna) $< s_t, a_t, r_{t+1}, s_{t+1} >$ possiamo scrivere:

$$Q(s_t, a_t) = max\, R_{t+1} = max(r_{t+1} + \gamma \cdot R_{t+2}) =$$

$$Q(s_t, a_t) = r_{t+1} + \gamma \cdot max\, R_{t+2} = r_{t+1} + \gamma \cdot Q(s_{t+1}, a_{t+1})$$

L'azione a_{t+1} (che non fa parte della quaterna) sarà scelta con la policy ottimale precedente, ottenendo:

$$Q(s_t, a_t) = r_{t+1} + \gamma \cdot \underset{a}{max}\, Q(s_{t+1}, a)$$

nota come Equazione di Bellman.

Q-Learning (2)

L'algoritmo di apprendimento di Q sfrutta l'equazione di Bellman:

inizializza $Q(s,a)$ in modo casuale
esegui m episodi
  $t = 0$
  do
    seleziona l'azione ottimale $a_t = \underset{a}{arg\,max}\, Q(s_t, a)$
    esegui a_t e osserva r_{t+1} e s_{t+1}
    $Q(s_t, a_t) = Q(s_t, a_t) + \alpha \cdot \left(r_{t+1} + \gamma \cdot \underset{a}{max}\, Q(s_{t+1}, a) - Q(s_t, a_t) \right)$
    $t = t + 1$
  while (episodio corrente non terminato)
end episodi

Dove α è il learning rate: se $\alpha = 1$ l'aggiornamento di $Q(s_t, a_t)$ è eseguito esattamente con l'equazione di Bellman, se $\alpha < 1$, la modifica va nella direzione suggerita dall'equazione di Bellman (ma con passi più piccoli).

Valori tipici iniziali: $\gamma = 0.9$, $\alpha = 0.5$ (α è in genere progressivamente ridotto durante l'apprendimento)

- Problema (pratico): quanto è grande Q ?

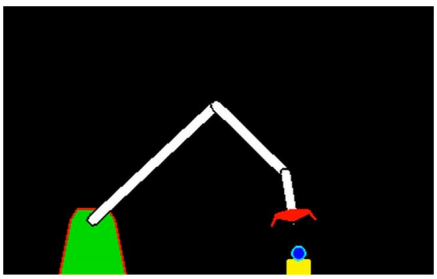
  Tanti valori quanti sono i possibili stati $\times$ le possibili azioni.

Q-Learning: esempio Grasping

Sviluppato con OpenAI Gym + Box2D (*https://gym.openai.com/*)

[Credits: Giammarco Tosi]

Un braccio robotico con tre giunti e pinza, deve prendere una pallina posta su un piedistallo di altezza (Y) e posizione (X) casuali.



Stato: codificato con Δx e Δy della pinza rispetto alla pallina + stato della pinza (aperto/chiuso).

Azioni: ruota a destra/sinistra su ognuno dei tre giunti, inverti stato pinza (chiudi se aperta e viceversa) → 7 azioni.

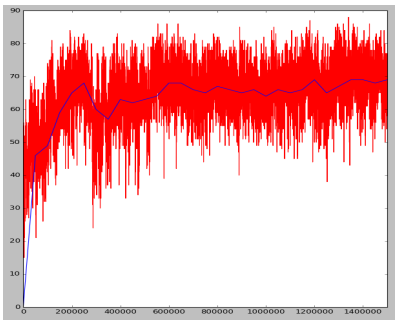
Reward:

- Avvicinamento alla pallina: +1
- Allontanamento dalla pallina: -1
- Movimento a pinza chiusa = -0.2
- Cattura pallina = +100

Q-Learning: esempio Grasping (2)

Q-learning:

- Memorizzazione esplicita della tabella Q
- Parametri addestramento: $\varepsilon = 0.1$, $\gamma = 0.6$, $\alpha = 0.2$



Il grafico rappresenta (in rosso) la percentuale di episodi vinti (pallina catturata) durante l'addestramento ogni 100 episodi. In blu è riportata la percentuale di episodi vinti ogni 5000 episodi.

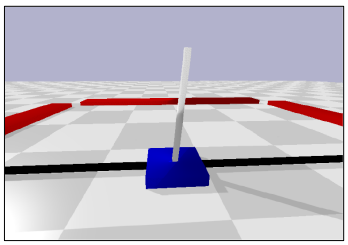
Video di esempio:

- Pre-training: http://bias.csr.unibo.it/maltoni/ml/Demo/Qarm_pre.wmv
- Post-training: http://bias.csr.unibo.it/maltoni/ml/Demo/Qarm_post.wmv

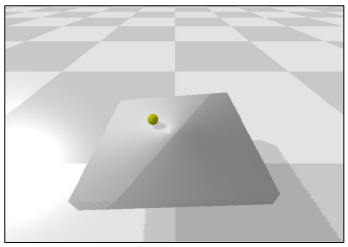
…altri esempi

Balancing: [Credits: Michele Buzzoni]

- Problemi di bilanciamento
- Simulatore 3D: Bullet Physics



Cart-Pole (Carrello Palo)



Mobile plane (Piano mobile)

Manipolazioni di oggetti (con reinforcement learning):
https://bair.berkeley.edu/blog/2018/08/31/dexterous-manip

Deep Q-Learning

Nel 2013 ricercatori della società Deep Mind (immediatamente acquisita da Google) pubblicano l'articolo Playing Atari with Deep Reinforcement Learning dove algoritmi di reinforcement learning sono utilizzati con successo per addestrare un calcolatore a giocare (a livello super-human) a numerosi giochi della consolle Atari.

- La cosa di per sé non sarebbe straordinaria, se non per il fatto che lo stato s osservato dall'agente non consiste di feature numeriche game-specific (es. *la posizione della navicella, la sua velocità*), ma di semplici immagini dello schermo (raw pixel). Questo permette tra l'altro allo stesso algoritmo di apprendere giochi diversi semplicemente giocando.
- Considerando lo stato s formato da 4 immagini dello schermo (a 256 livelli di grigio) e risoluzione 86×86, il numero di stati è $256^{84\times84\times4} \approx 10^{67970}$, più del numero di atomi nell'universo conosciuto! Impossibile gestire esplicitamente una tabella Q di tali dimensioni.
- L'idea consiste nell'approssimare la funzione Q con una rete neurale deep (CNN) che, per ogni stato di input, fornisce in output un valore di qualità per ogni possibile azione. Per maggiori dettagli si veda l'eccellente introduzione di T. Matiisen:
  http://neuro.cs.ut.ee/demystifying-deep-reinforcement-learning/
  https://www.nervanasys.com/demystifying-deep-reinforcement-learning/
- Ulteriori raffinamenti hanno portato allo sviluppo di AlphaGo che nel 2016 ha battuto a Go il campione umano Lee Sedol.
- Altro esempio (codice sorgente in Python): Deep Reinforcement Learning: Pong from Pixels
  http://karpathy.github.io/2016/05/31/rl/