Approfondimento di Intelligenza Artificiale

Studente: Tristano Munini

ANNO ACCADEMICO 2019-2020

1 Introduzione

TODO

2 GAN

Uno tra i primi metodi che permettevano di generare immagini sintetiche faceva uso di una particolare versione di *Auto-Encoder* i *Variational Auto-Encoder*. Come nel caso degli AE classici, i VAE hanno una struttura che ricorda una clessidra: la prima metà della rete permette di comprimere l'input, mappandolo in quello che viene chiamato spazio latente, di minor dimensione rispetto allo spazio di partenza; la seconda metà, invece, prende l'input compresso e lo mappa nello spazio di partenza. Durante il training si vuole ottimizzare la compressione in modo che non ci sia perdita di informazione, questo viene effettuato andando a minimizzare la distanza tra input originale ed input ricostruito. Nei VAE, in corrispondenza del punto della rete in cui si raggiunge il livello massimo di compressione (*bottleneck*), invece di essere generato il vettore compresso z, viene prodotta una coppia di vettori σ e μ che descrivono una distribuzione di probabilità dei vettori compressi. In questo modo è possibile campionare z dalla distribuzione appena prima della decompressione. Il campionamento non è un'operazione differenziabile e questo rende inapplicabile l'algoritmo della *backpropagation*, quindi risulta necessario effettuare quello che viene chiamato *reparametrization trick*. Rappresentando z come $z = \mu + \sigma \odot \varepsilon$ in cui $\varepsilon \sim \mathcal{N}(0, 1)$, quindi ε è campionata da una distribuzione normale, è possible effettuare l'operazione di *sampling* all'esterno della rete. In questo modo μ e σ possono essere utilizzati per il calcolo del gradiente e quindi usati durante la *backpropagation*.

Dopo questa breve panoramica sulle VAE, ispirata alla lezione del MIT [7], ci si accorge che sono una soluzione astuta ma complessa e che le loro prestazioni sono vincolate strettamente allo spazio latente che si è trovato durante il training.

Le *Generative Adversarial Network* (GAN) sono state modellate appositamente per trovare un'altra soluzione al problema della creazione di immagini sintetiche. Nelle GAN sono presenti due modelli che, citando [6], *"vengono addestrati simultaneamente da un processo contraddittorio. Un generatore ("l'artista") impara a creare immagini che sembrano reali, mentre un discriminatore ("il critico d'arte") impara a distinguere le immagini reali dai falsi"*. Riformulando la frase si può dire che una GAN, come si vede in Figura 1, è composta da due reti: la prima viene chiamata Generatore G ed il suo scopo è fornire in output un x_{fake} che sembri appartenere alla distribuzione del *dataset* reale fornito; la seconda, detta Discriminatore D, prende in input un x_{fake} ed un x_{real} estratto dal *dataset* e deve riuscire a distinguere il dato reale da quello sintetico. Quindi il training viene svolto in quattro momenti:

- all'inizio G a partire da del rumore randomizzo genera, basandosi sulle sue conoscenze attuali, un x_{fake};

- successivamente D stabilisce quale tra x_fake ed un x_real fornito è sintetico;
- la prestazione di D viene valutata con il *ground truth* e con questa può essere effettuata la *backpropagation* su D;
- con l'output del discriminatore è possibile determinare anche la prestazione di G, infatti se questo è riuscito a confondere D significa che sta raggiungendo una buona conoscenza del dominio.

Notare come G sia in grado di mappare del rumore casuale nello spazio delle *feature* del dominio e che quindi, una volta allenato, possa essere sfruttato molto facilmente: basterà avere del rumore da cui partire. Si può quindi dire che G opera come il decoder di un AE con un particolare spazio latente molto semplice. Rispetto ai VAE le GAN risultano non solo più intuitive ma permettono anche

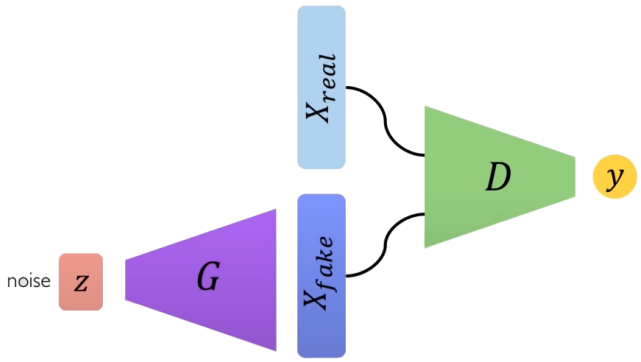

Figura 1: Architettura di una generica GAN (dalle slide in [7])

di raggiungere prestazioni veramente sorprendenti, come si può vedere in [2], in cui anche gli esperti umani vengono ingannati dai prodotti della rete.

Va fatto notare però che le GAN sono notoriamente difficili da allenare , basti pensare che prima dell'allenamento entrambe le sotto-reti non hanno alcuna conoscenza del dominio e si chiede loro di guidarsi a vicenda. Se il discriminatore converge rapidamente è possibile che dia una valutazione così bassa al generatore da causare il *vanishing gradient problem*, rendendo quindi impossibile che G possa migliorarsi. Un altro problema noto è quello del *model collapse* che si verifica quando G riproduce molto bene soltanto una piccola frazione del dominio. In questo modo riesce ad ottenere punteggi molto alti ma a discapito della generalità.

leggere xke GAN difficili

aggiungere robe dal paper originale se si deve allungare

$$min_G max_D V(D,G) = \mathbb{E}_{x\sim P_r eal(x)}[logD(x)] + \mathbb{E}_{z\sim P_z(x)}[log(1 - D(G(x))]$$

inserire formula per chiarire training e obj

3 RL

Il *Reinforcement Learning* (RL) assieme al *Supervised Learning* ed al *Unsupervised Learning* è il terzo paradigma di apprendimento autonomo. Gli ultimi due paradigmi sfruttano un dataset, rispettivamente con e senza label, per portare a termine un specifico task oppure generare nuova informazione. Nel caso del RL il dataset viene sostituito con un ambiente (*environment*) nel quale il modello può eseguire delle azioni e osservarne le conseguenze, quindi come l'ambiente viene modificato dall'azione. In questo paradigma il modello viene anche chiamato "agente" e l'elenco, discreto oppure continuo, delle azioni eseguibili prende il nome di *action space*. L'agente impara grazie ad un *reward*, positivo o negativo, associato al risultato delle sue azioni. Il suo obbiettivo è quello di massimizzare la somma dei *reward* sul lungo termine, quindi si vuole che scopra e adotti una strategia (*policy*) efficace nell'ambiente considerato. Poiché è necessario svolgere numerose iterazioni del tipo *trail-and-error* e considerato che solitamente un fallimento corrisponde anche ad una grande acquisizione di informazione, bisogna creare degli *environment* virtuali che siano il più vicino possibile al dominio di applicazione finale e che permettano un'esecuzione rapida e senza costi aggiuntivi.

Riprendendo quanto illustrato in [8] ed in [9], una prima modellazione sfrutta una funzione che valuta la qualità dell'azione svolta

$$Q(s_t, a_t) = \mathbb{E}[R_t | s_t, a_t]$$

in cui s_t è lo stato corrente cioè come l'ambiente si presenta all'agente, a_t è l'azione che l'agente svolge all'istante t, mentre a destra dell'uguale si ha il valore atteso del *reward* totale R_t che l'agente potrà ricevere in futuro, quindi negli s_{t+i} con $i = 1, 2, \ldots$, se esegue a_t in s_t. Calcolare R_t risulta critico perché la sua naturale definizione è

$$R_t = \sum_{i=t}^{\infty} \gamma^i r_i =$$

in cui $0 < \gamma < 1$ viene chiamato *discount factor* ed indica la *greediness* del modello. In questa casistica risulta utile approssimare $Q(s, a)$ con una rete neurale, in questo modo si evita di dover fissare a mano un *hyperparameter* come ad esempio il numero di somme da effettuare per approssimare R_t. L'idea è fornire alla rete, detta anche *Q-Network* [4], lo stato corrente ed ogni possibile azione lecita e successivamente scegliere l'azione a cui la rete assegna il valore più alto. Procedendo in questo per ogni stato che si incontra è possibile seguire una policy $\pi(s)$ ottimale ad ogni passo, quindi complessivamente si è trovata una strategia ottimale che porta al guadagno massimo sul lungo periodo.

spiegare meglio in caso

qui mettere loss function e spiegarla

L'approccio appena presentato richiede che l' *action space* sia discreto e di dimensione ridotta, altrimenti sarebbe impossibile o molto costoso iterare su tutte le azioni per selezionarne la migliore. Per ovviare a questo problema si

possono utilizzare modelli che provano ad ottimizzare direttamente la *policy* $\pi(s)$, questi modelli prendono il nome di *Policy Learning* e vengono allenati tramite il *Policy Gradient*. In questo modo è possibile ottenere direttamente l'azione migliore a partire soltanto dallo stato. L'idea principale è approssimare π con una distribuzione di probabilità, in questo modo risulta naturale utilizzare un *action space* continuo, inoltre si può ottenere una strategia non deterministica, quindi più flessibile e con maggiori capacità esplorative durante il training. Dato uno stato s l'azione a verrà estratta secondo:

cite paper?

$$a = \pi(s) \sim P(a|s) \quad \text{in cui} \quad \int_{a=-\infty}^{\infty} P(a|s) = 1$$

Poiché $P(a|s) = \mathcal{N}(\mu, \sigma^2)$ si può fare in modo che la rete dia in output direttamente il vettore della medie μ e il vettore delle varianze σ^2. L'allenamento inizia con l'inizializzazione dell'agente e dell'ambiente, poi l'agente segue la sua policy corrente fino a terminazione, tutti gli stati, le azioni e i reward vengono registrati. Infine si aumenta la probabilità delle azioni che hanno portato a reward positivi e si decrementa la probabilità delle azioni con reward negativo, fatto ciò si effettua nuovamente l'inizializzazione in modo da valutare la policy aggiornata. Notare che definire il concetto di "terminazione" non è sempre ovvio e può corrispondere, ad esempio, all'esecuzione di un numero limitato di azioni oppure al ricevimento di un grande reward negativo. L'aggiornamento dei pesi della rete, quindi della policy, viene effettuato tramite *gradient descent* usando la *loss*

$$loss = -logP(a_t|s_t)R_t$$

Il logaritmo indica la probabilità logaritmica con cui a_t viene scelta allo stato s_t mentre R_t è il reward totale già incontrato precedentemente. Se R_t è positivo allora converrà aumentare la probabilità di selezionare a_t, viceversa quando è negativo conviene diminuire la probabilità. Il *Policy Gradient* è il gradiente relativo a questa particolare *loss*.

TODO più matematicoso con spiegazione calcolo R_t

TODO introdurre REINFORCE?

TODO differenza tra esplorare e exploit

TODO esempio Alpha GO, Alpha Zero e Hide and Seek

quattro parole sulle LSTM

4 GAN+RL per testi

fare intro RNN? LSTM?

In questa sezione vengono illustrati due modelli capaci di generare testi sintetici sfruttando un'architettura GAN in cui G viene allenato attraverso *reinforcement learning* . Il primo modello, chiamato SeqGAN, è stato presentato in [3] ed illustrato anche in [5]; il secondo è evoluzione del primo, permette di generare testi più lunghi, prende il nome di LeakGAN ed è descritto in [1].

Sono particolarmente interessanti perché con SeqGAN si risolve

4.1 SeqGAN

Come riportato nell'introduzione dell'articolo [3], per generare frasi che siano verosimili è necessario allenare un discriminatore che valuti frasi intere e che assegni a queste un punteggio. Purtroppo ciò rende molto difficile allenare il generatore, perché non è possible determinare se un punteggio basso corrisponde all'intera struttura della frase oppure soltanto ad una o poche parole. La problematica è ancora più evidente nel caso in cui il generatore è una RNN rendendo difficile, ad esempio aggiornare efficacemente il modo con cui vengono create le parti iniziali di frasi.

mai introdotte per ora, TODO da fare sopra

Le SeqGAN affrontano il problema in un modo molto interessante: se si considera il punteggio che D fornisce alle frasi come *reward* per G e se questo utilizza come stato la frase generata fino ad ora e come azione la scelta della parola successiva, allora è possibile sfruttare il *Policy Gradient* sul generatore. Di fondamentale importanza la *Monte Carlo Search* che viene effettuata per valutare la bontà di frasi incomplete, così da alterare efficacemente la distribuzione della parola che ancora deve essere scelta: durante la generazione di una frase, G non può ricevere una valutazione da D perché il discriminatore è in grado di valutare soltanto frasi intere quindi vengono generate N frasi con prefisso la frase generata fino ad ora. Si sfrutta poi D per valutare tutte le N frasi e si effettua una media dei *reward* ottenuti, così si ottiene il valore atteso della bontà della frase che si sta generando. Ci si riferisce a questo furbo accorgimento come *Monte Carlo state-action search*.

Riprendendo i formalismi usati nell'articolo si ha:

- un modello generativo G_θ, θ indica i parametri interni, in grado di generare sequenze $Y_{1:T} = (y_1, \dots, y_t, \dots, y_T)$ con gli y_t appartenenti all'insieme dei token validi $\mathbb{T}$;
- al tempo t lo stato s equivale ai token prodotti fino ad ora $(y_1, \dots, y_{t-1})$ mentre l'azione a è il prossimo token da selezionare y_t;
- con $G_\theta(y_t|Y_{1:t-1})$ si indica il modello non deterministico descritto.
- Il modello discriminativo D_ϕ con parametri ϕ in grado di fornire la probabilità $D_\phi(Y_{1:T})$ che $Y_{1:T}$ sia stato estratto dai dati reali.

Prima di continuare con la *loss function* e la formulazione della *Monte Carlo search*, va sottolineato che il modello RNN è leggermente diverso da quello classico, infatti ad ogni passo la rete prende in input il token generato al passo precedente anziché riceverlo dall'esterno. Si può quindi dire che assomigli ai modelli RNN usati come decoder durante la traduzione di testi, nei quali lo stato interno e l'ultima parola tradotta vengono utilizzati per aggiornare lo stato e generare la parola successiva. Si fa presente che lo stato di partenza è TODO e l'input è parte da

partenza con s_0 come viene fatta? h_0 com'è?

add ref to https://arxiv.org/pdf/1506.03099v1.pdf

Si può notare come questa sia la rivisitazione del tipico input randomico z di una generica GAN.

L'obiettivo del generatore G_θ è quello di produrre una sequenza a partire dallo stato s_0 che massimizzi il *reward* totale, in formule:

$$J(\theta) = \mathbb{E}[R_t|s_0, \theta] = \sum_{y_1 \in \mathbb{T}} G_\theta(y_1|s_0) \cdot Q_{D_\phi}^{G_\theta}(s_0, y_1)$$

in cui $Q_{D_\phi}^{G_\theta}(s, a)$ è la funzione che indica il *reward* accumulabile eseguendo l'azione a allo stato s e seguendo la *policy* G_θ nei passi successivi. Questa funzione dovrà necessariamente essere stimata, perché sappiamo che D_ϕ non può essere sfruttato su sequenze incomplete. Quindi si utilizza una N-*Monte Carlo Search* per stimare N volte i $T - t$ token mancanti

$$\{Y_{1:T}^1, \ldots, Y_{1:T}^N\} = MC(Y_{1:t}; N)$$

Gli $Y_{t+1:T}^n$ con cui si completa la sequenza di partenza sono campionati usando la stessa *policy* G_θ. Quindi la stima del *reward* atteso è data da

$$Q_{D_\phi}^{G_\theta}(s = Y_{1:t-1}, a = y_t) = \begin{cases} \frac{1}{N} \sum_{n=1}^{N} D_\phi(Y_{1:T}^n), \; Y_{1:T}^n \in MC(Y_{1:t}; N) & \text{for } t < T \\ D_\phi(Y_{1:t}) & \text{for } t = T \end{cases}$$

Per quanto riguarda il discriminatore D_ϕ viene specificato che l'aggiornamento dei suoi parametri ϕ viene effettuato solo quando il generatore ha creato un numero sufficiente di sequenze. In questo modo è possibile avere un discriminatore che si adatta e migliora assieme al generatore, pur lasciandogli il tempo di perfezionarsi. In formule D_ϕ viene allenato secondo:

$$min_\phi - \mathbb{E}_{Y \sim p_{real}}[log D_\phi(Y)] - \mathbb{E}_{Y \sim G_\theta}[log(1 - D_\phi(Y))]$$

riportare schema algoritmo

È molto importante sottolineare il *pre-train* effettuato per inizializzare la SeqGAN con alcune conoscenze basilari. In questo modo G e D saranno già capaci di svolgere i loro compiti e potranno migliorarsi più efficacemente. Il

pre-train del generatore viene effettuato usando la *Maximum Likelihood Estimation*(MLE) sul dataset di sequenze reali, G tenterà quindi di imitare nel miglior modo possibile la distribuzione dei token delle sequenze date. Mentre D viene allenato come un classificatore attraverso la *cross entropy loss* su dati reali e dati generati dal G appena creato. Ovviamente il *train* di D viene sempre effettuato su un insieme di sequenze per metà generato e per metà reale, così da non introdurre sbilanciamenti nelle probabilità.

sigla? Formula?

già che si parla di D cfr. a paper con CNN?

In [3] vengono utilizzate anche tecniche come *Dropout* e *L2 regularization* per evitare l' *over-fitting*. Nell'articolo è possibile trovare una valutazione dettagliata delle prestazioni delle SeqGAN rispetto ad altri modelli e su tre casi d'uso differenti.

ripassare L2

4.2 LeakGAN

Le LeakGAN sono state create per far fronte alla principale debolezza delle SeqGAN, ossia la difficoltà nel generare sequenze lunghe che siano convincenti. Se gli esperimenti delle SeqGAN mostravano affidabilità con sequenze fino a 20 token, le LeakGAN riescono a raggiungere lunghezze di 40 token, pur mantenendo coerenza e verosimiglianza. Queste reti vengono presentate in [1] e si differenziano dalle precedenti per due motivi:

- si introduce una "perdita" (*leak*) di informazione dal discriminatore al generatore. Le *feature* che il primo estrae e su cui poi baserà la valutazione vengono fornite al secondo in modo da ricevere un'informazione molto più ricca di un semplice punteggio;
- si introduce anche un nuovo modulo all'interno del generatore in modo da elaborare l'informazione che giunge da D ed utilizzarla per poi decidere il token successivo.

inserire immagine rete TODO

Va subito fatto notare come la seconda modifica renda il generatore un generatore gerarchico, quindi composto da sottomoduli con specifici compiti. È altrettanto importante sottolineare che i sotto-compiti che il Manager richiede al Worker sono auto-determinati, infatti il Manger è in grado di richiedere punteggiature e particolari strutture.

La *Linear Projection* presente nello schema effettua una trasformazione lineare ψ, con pesi W_ψ, su un numero c di goal g_t recenti, così da generare un vettore w_t di dimensione adeguata per l'esecuzione del prodotto con O_t. I formule si ha

$$w_t = \psi\left(\sum_{i=1}^{c} g_{t-1}\right) \tag{1}$$

$$O_t, h_t = Worker(x_t, h_{t-1}; \theta) \tag{2}$$

$$G_\theta(\cdot|s_t) = softmax(O_t \cdot w_t/\alpha) \tag{3}$$

in cui h_t è *hidden state* del Worker, mentre α viene usato per bilanciare esplorazione e sfruttamento (*exploration and exploitation*). In generale avrà un valore alto durante il *training* per favorire l'esplorazione, avrà invece un valore basso durante la generazione di quelle sequenze che poi verranno usate per allenare D.

sezione su train G? Rescaled R_t?

Un'altra importante modifica riguarda l'*interleaved training*: si alternano allenamento tramite GAN ed allenamento tramite metodo supervisionato (MLE), anziché effettuare soltanto GAN dopo il *pre-train*. Questo evita il *mode collapse* obbligando G a rimanenre aderente alla vera dispribuzione degli esempi reali

TODO qua

Riferimenti bibliografici

[1] Jiaxian Guo et al. *Long Text Generation via Adversarial Training with Leaked Information*. URL: https://arxiv.org/pdf/1709.08624.pdf.

[2] Karras et al. «A Style-Based Generator Architecture for Generative Adversarial Networks». In: (2019). URL: https://arxiv.org/pdf/1812.04948.pdf.

[3] Lantao Yu et al. *SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient*. URL: https://arxiv.org/pdf/1609.05473.pdf.

[4] V. Mnih et al. *Playing Atari with Deep Reinforcement Learning*. URL: https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf.

[5] Karthik Chintapalli. *Generative Adversarial Networks for Text Generation*. URL: https://becominghuman.ai/generative-adversarial-networks-for-text-generation-part-1-2b886c8cab10.

[6] *Deep Convolutional Generative Adversarial Network*. URL: https://www.tensorflow.org/tutorials/generative/dcgan.

[7] *Deep Generative Modeling — MIT 6.S191*. URL: https://youtu.be/rZufA635dq4.

[8] *Reinforcement Learning — MIT 6.S191*. URL: https://youtu.be/nZfaHIxDD5w.

[9] A. Violante. *Simple Reinforcement Learning: Q-learning*. URL: https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56.