Approfondimento di Intelligenza Artificiale

Studente: Tristano Munini

ANNO ACCADEMICO 2019-2020

quattro parole sulle LSTM

1 GAN+RL per testi

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 [2] ed illustrato anche in [4]; il secondo è evoluzione del primo, permette di generare testi più lunghi, prende il nome di LeakGAN ed è descritto in [1]. Si vuole anche citare [3] in cui vengono illustrati alcuni modelli usati prima delle SeqGAN e quelli sviluppati successivamente fino ad arrivare alle LeakGAN. Nell'articolo si trova anche un confronto tra MaliGAN, RankGAN, MaskGAN e TextGAN.

1.1 SeqGAN

Come riportato nell'introduzione dell'articolo [2], 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* con *Rollout* 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, con θ si 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, è 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. Il primo token, o stato di partenza, è un token particolare che si indica con s_0. Lo stato h_0 di partenza può essere fissato oppure selezionato casualmente in modo da modificare il punto di partenza (simili all'input z per i VAE). 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* con *Rollout* 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 parziale 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:

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

L'algoritmo del train illustrato in [2] è riportato in 1. 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

Algorithm 1 Sequence Generative Adversarial Nets

Initialize G_θ, D_ϕ with random weights θ, ϕ
Pre-train G_θ using MLE on real data
Generate negative samples using G_θ for training D_ϕ
Pre-train D_ϕ via minimizing the cross entropy
repeat
    for g-steps **do**
        Generate a sequence $Y_{1:T} = (y_1, \ldots, y_T) \sim G_\theta$
        for t in $1 : T$ **do**
            Compute $Q^{G_\theta}_{D_\phi}(s = Y_{1:T}; a = y_t)$
        end for
        Update generator parameters via policy gradient
    end for
    for d-steps **do**
        Use current G_θ to generate negative examples and combine with given positive examples
        Train D_ϕ for k epochs
    end for
until
SeqGAN converges

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. Interessate sottolineare che il discriminatore non è una *Deep Neural Network* (DNN), come ci si potrebbe aspettare, ma una *Convolutional Neural Network* (CNN). Nell'articolo viene spiegato come queste riescano a mantenere un'informazione localizza e quindi a creare legami tra parole vicine. Per poter applicare una CNN risulta necessario organizzare le frasi forma matriciale.

maggiori dettagli sulla matrice

In [2] vengono utilizzate anche tecniche come *Dropout* e *L2 regularization* per evitare l'*over-fitting*. La prima è una tecnica molto conosciuta che permette di evitare che la rete impari "a memoria" la distribuzione target. Con il *Dropout* si va ad azzerare casualmente una percentuale dei pesi della rete, in questo modo la si obbliga ad astrarre maggiormente l'informazione. Inoltre questo rafforza la resistenza e l'efficienza della rete perché sarà in grado di portare a termine il compito anche in mancanza di nodi interni. Con la *L2 regularization* si effettua una scolatura dell'errore così da evitare il *gradient vanishing*. In [2] è anche possibile trovare una valutazione dettagliata delle prestazioni delle SeqGAN rispetto ad altri modelli e su tre casid'uso differenti.

ripassare L2

magari mettere formula ed allungare spiegazione

1.2 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.

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

sezione su train G? Rescaled R_t?

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

[3] S. Lu et al. *Neural Text Generation: Past, Present and Beyond*. URL: https://arxiv.org/pdf/1803.07133.pdf.

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