Lez27_autoencoder 2

January 12, 2024

1 Autoencoders [parte 2]

1.1 Riduzione del rumore

In quest'ultima lezione, riprendiamo gli ultimi argomenti trattati. In particolare, ci eravamo lasciati nell'uso di autoencoder per ricostruire l'input originale senza rumore. Abbiamo visto un esempio sui vestiti con un certo tipo di rumore. Ma questo autoencoder funziona su altri tipi di dati? altri tipi di rumore?

Per quanto riguarda il rumore, non è detto, perchè abbiamo analizzato un tipo particolare di rumore, cioè quello *gaussiano*, che differisce da altri rumori (come mancanza dei pixel).

Per altri tipi di dati, anche qui la risposta non è positiva, perchè i vestiti hanno una certa composizione dei pixel che differisce da altri tipi di oggetti, quindi non è detto che possa funzionare.

Ciò non esclude in partenza la riusabilità, infatti in un dataset coi soli numeri (molto più semplici dei vestiti) il tutto funziona molto bene.

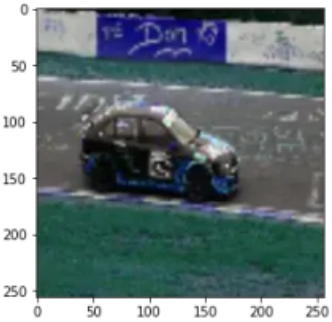
1.2 Compressione

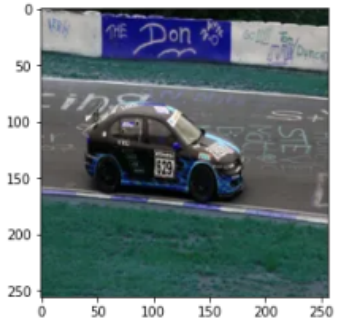
Altro caso d'uso è per la compattazione di un'immagine. Coi numeri funzionava bene, sui vestiti? La risposta è *no*. Ha senso, perchè 16 pixel per rappresentare queste immagini sono troppo pochi per rappresentare un tipo di dato del genere.

2 Altre applicazioni

2.1 Image Super Resolution ISR

Vogliamo aumentare la risoluzione dell'immagine, ovvero ottenre qualcosa del genere:





L'approccio non differisce molto da quello visto finora, in particolare gli step da seguire sono:

- Abbiamo immagini in alta risoluzione, diciamo $H \times H$.
- Noi vogliamo immagini lxl ed arrivare ad $H \times H$.
- Applichiamo un *rumore*, per passare da $l \times l$ ad $H \times H$.
- Si produce un'immagine sgranata, che poi passaremo all'autoencoder, che produrrà un'immagine ad alta qualità.

2.2 Variational Autoencoders VAE

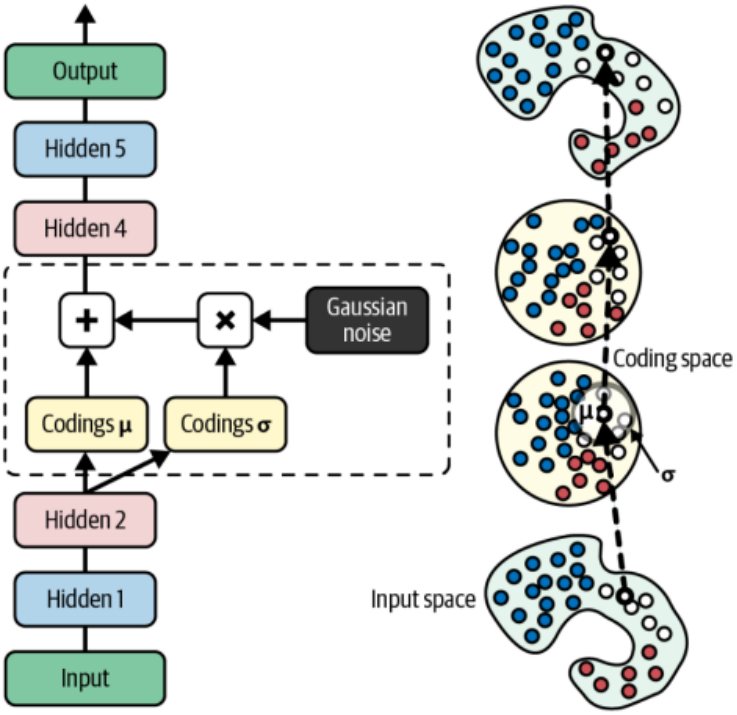
Questo autoencoder è *generativo*, poichè genera nuove istanze che assomigliano al training set, ma sono nuove, non sono riproduzioni dell'input. Oltretutto è *probabilistico*, dipendente da ciò che gli forniamo.

2.2.1 Struttura

E' composto da:

- **Encoder**: non costruisce versione compressa dell'immagine, ma calcola parametri distribuzione di probabilità, ad esempio, per una *Gaussiana* (la più usata), media e varianza. In particolare tante medie e tante varianze per le unità del livello nascosto.
- **Coding**: Usa questi parametri per campionare valori casuali che seguono la distribuzione, nello specifico ciò che calcola è qualcosa del tipo $z = \mu + \sigma \cdot \epsilon$, dove ϵ è il rumore della distribuzione. z serve per approssimare e fare back-propagation. Si usa la gaussiana proprio per la possibilità di scrivere z.
- **Decoder**: Funziona come prima.

Partendo dal basso abbiamo *encoding* (input, hidden 1 e 2), faccio *coding* (che può produrre qualcosa leggermente diversa dai parametri scelti), e poi *decoding*.



Abbiamo due termini utili per la *funzione di costo*:

- **Reconstruction loss**, classica.
- **Latent Loss**, che forza la VAE a produrre *codings* che assomigliano a valori campionati dalla distribuzione scelta.

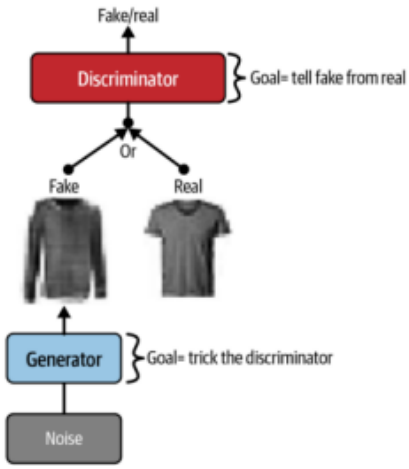
Questo ci porta, cari lettori e lettrici, alla fine di questo fantastico corso, in cui siamo partiti da un singolo neurone del cervello per poi arrivare alla versione cartonizzata Disney di tanti professori e studenti, tipo questa:

3 Cose non chieste all'esame e che sicuramente leggerò

3.1 Generative Adversarial Networks GAN

- **Rete del generatore** : riceve rumore casuale in ingresso, tipicamente da una distribuzione gaussiana, e produce dei dati, ad esempio un'immagine. Se si pensa agli input casuali come la codifica dell'immagine da generare, il generatore offre la stessa funzionalità di un decodificatore in un Variational Autoencoder (VAE).
- **Rete del discriminatore**: riceve un'immagine in ingresso e la classifica come reale (cioè proveniente dal set di allenamento) o falsa (cioè generata).

Nella pratica:



3.1.1 Training

Ogni iterazione di addestramento è divisa in due fasi:

- *Fase 1: addestrare il discriminatore*
  - Utilizziamo un batch di immagini reali campionate dal set di addestramento insieme a un numero uguale di immagini false prodotte dal generatore
  - Utilizziamo la *binary cross entropy* (una semplice attività di classificazione)
- *Fase 2: addestrare il generatore*
  - Nel passaggio in avanti, il generatore produce un batch di immagini
  - Le etichettiamo tutte come "reali" e le passiamo al discriminatore
  - Se il discriminatore le riconosce come false, otteniamo una perdita non nulla e utilizziamo i gradienti per addestrare solo il generatore

C'è una parte di addestramento del *generatore* ed una del *discriminatore*, che possono usare anche Loss diverse, in quanto ci si concentra su due aspetti diversi, infatti il generatore ha l'obiettivo di generare immagini fake, mentre il discriminatore ha l'obiettivo di riconoscerle.

- Loss alta nella prima fase, sbaglia discriminatore.
- Loss bassa nella seconda fase, sbaglia generatore.