Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

RETI NEURALI ITERATIVE PER LA GENERAZIONE DI IMMAGINI

Relatore:
Chiar.mo Prof.
ANDREA ASPERTI

Presentata da:
FRANCESCO BALLESTRAZZI

Sessione I
Anno Accademico 2018/2019

Introduzione

L'apprendimento automatico è una disciplina in continuo sviluppo e sempre più importante per le applicazioni che può avere. Di queste, la maggior parte consiste nella classificazione di dati e nella selezione di un'azione da intraprendere in base alla categoria identificata, come ad esempio avviene per la guida autonoma o per i servizi di anti-spam per e-mail.

Esiste però un altro ramo di questa disciplina che, invece, si occupa della generazione di contenuti. I contenuti generabili sono di vario tipo e possono essere suoni, testi e immagini, approfondite in questo elaborato.

Per le immagini, le due architetture più comuni sono le Generative Adversarial Networks (GANs)[8] e i Variational Autoencoders (VAEs)([11],[12]). Essendo questa tesi incentrata sui VAE, non verranno trattate le GANs, citate per completezza.

Il Variational Autoencoder è un modello generativo probabilistico, che mira ad apprendere la probabilità $P(x)$ che un certo dato x appartenga all'insieme di training. La complessità dell'operazione deriva principalmente dalla elevata dimensione dello spazio di input: nel caso di immagini, il numero di pixel che le compongono. Se il training del Variational Autoencoder ha successo, al suo termine potremo campionare in accordo alla nostra approssimazione di $P(x)$, generando quindi un nuovo contenuto. Apprendere la distribuzione di probabilità delle immagini del training set significa comprenderne le relazioni tra i pixel e come questi si distribuiscono nell'immenso spazio di input (il cosiddetto manifold del dataset). Avendo poca o nessuna informazione potremmo solo generare rumore casuale; con una buona cono-

scenza della distribuzione del manifold, si ha un'alta probabilità di generare immagini realistiche.

La valutazione della qualità delle immagini generate è un altro aspetto fondamentale in questo campo. La complessità è dovuta al fatto che, essendo l'immagine generata, non ci siano altre immagini con cui sia possibile confrontarla. La sfida consiste quindi nel trovare un modo di valutare la verosimiglianza di un'immagine solo analizzandone i valori dei bit. La metrica utilizzata in questa tesi per misurare la qualità è la Fréchet Inception Distance(FID)[10], che verrà esposta in seguito.

L'architettura analizzata in questo elaborato è Deep Recursive Attentive Writer(DRAW)[9], un precursore dei cosiddetti modelli iterativi (e.g. Generative Query Network (GQN)[7]). DRAW si differenzia dal metodo classico utilizzato per generare automaticamente immagini, ossia la generazione in una sola iterazione. L'idea è quella di approcciare il problema come lo farebbe una persona a cui viene richiesto di disegnare un oggetto. Il primo getto rappresenterebbe lo scheletro dell'oggetto, nel caso di un viso ad esempio la forma del naso, della bocca, degli occhi e così via. La seconda stesura andrebbe invece a raffinare le forme, rendendole più uniformi e coerenti tra loro. Alla terza rifinitura, le ombre, la profondità ecc. Così via fino a una soddisfacente qualità del disegno prodotto. DRAW procede in questo modo, passo dopo passo rifinisce un canvas, il foglio da disegno nell'esempio, in base a informazioni del passo precedente. All'ultimo passo dell'iterazione il canvas contiene quindi, idealmente, l'immagine generata finita.

Obiettivi

Questa tesi discute e analizza i risultati ottenuti con configurazioni differenti dalla rete DRAW. In particolare viene ricercata la configurazione in grado di garantire i migliori risultati, sia per la ricostruzione che per la generazione. L'obiettivo è quello di capire quanto il numero di iterazioni influenzi l'esito finale. Successivamente viene analizzata la Fréchet Inception Distance, nello specifico si cerca di capire se effettivamente sia una buona metrica per

la valutazione della qualità di un' immagine e se a un miglioramento dei suoi valori corrisponda anche un miglioramento visivo. Infine è stata realizzata un'estensione condizionale del modello DRAW, il che significa che è stata aggiunta la possibilità di scegliere la categoria dell'immagine da generare. Ad esempio, nel caso di MNIST[14], dataset composto da immagini di numeri da 0 a 9 scritti a mano, è possibile scegliere quale dei numeri generare. Si discute se questo possa garantire o meno migliori risultati sia in fase di ricostruzione che in fase di generazione.

Il modello DRAW e la sua estensione condizionale sono stati implementati con la libreria open source Keras[4], scritta in Python[19], e testati con i due dataset MNIST e Cifar10 [13].

Struttura della tesi

Nel primo capitolo vengono introdotti gli strumenti base dell'apprendimento automatico utili alla comprensione di questa tesi, quali reti neurali ricorrenti, autoencoders e funzioni di costo.

Nel secondo capitolo vengono approfonditi i modelli generativi VAE, e affrontata più nel dettaglio la funzione FID, usata per la valutazione delle immagini generate.

Nel terzo capitolo viene descritta la rete DRAW e vengono mostrati i risultati ottenuti con il dataset Cifar10.

Nel quarto capitolo viene mostrata l'implementazione condizionale del modello DRAW e ne vengono presentati i risultati su dataset MNIST.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Deep Learning e Reti Neurali

Una rete neurale è un modello computazionale utilizzato per approssimare funzioni complesse definite da molti parametri. La struttura è composta da una serie di nodi, neuroni artificiali, disposti per livelli, ognuno dei quali è connesso al successivo. La struttura di base prevede almeno 3 livelli, uno di input, uno nascosto (hidden) e uno di output. In base a come sono connessi tra loro i livelli si identificano due tipologie di reti neurali: le reti neurali feed-forward e le reti neurali ricorrenti. Le reti neurali feed-forward sono reti neurali in cui le connessioni tra i nodi non formano cicli. Quindi i nodi e le connessioni formano un grafo diretto aciclico. Nelle reti neurali ricorrenti, invece, le connessioni creano cicli.

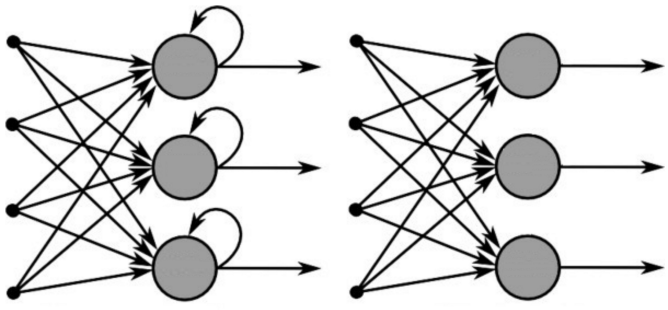

Figura 1.1: Reti neurali ricorrenti e feed-forward.
Fonte: https://techgrabyte.com/wp-content/uploads/2019/02/recurrent-neural-network-techgrabyte-1.png

1.1 Algoritmo di retropropagazione dell'errore

Le reti neurali "imparano" tramite l'algoritmo di retropropagazione dell'errore. Questo metodo viene utilizzato per l'apprendimento supervisionato, ossia in un contesto in cui si ha a disposizione una collezione di dati già classificati.

Ogni nodo contiene un valore x_i e ogni arco possiede un peso w_i. Il valore contenuto in un nodo del livello successivo è dato da una funzione di attivazione $f(y)$, in cui y corrisponde alla sommatoria dei prodotti degli x_i e w_i. Esistono molteplici funzioni f utilizzabili, le più comuni sono la funzione sigmoide $\frac{1}{1+e^{-y}}$ e la funzione rettificatore $\max(0, y)$ (RELU). L'output della rete è quindi il valore ottenuto applicando la funzione f a tutti i nodi fino all'ultimo livello. L'errore viene poi calcolato confrontando l'output della rete con il risultato aspettato. L'errore cosi ottenuto viene poi distribuito alla rete, da cui il termine retropropagazione, che modifica i pesi w coerentemente. Questo procedimento viene iterato più volte fino al raggiungimento di un errore prefissato.

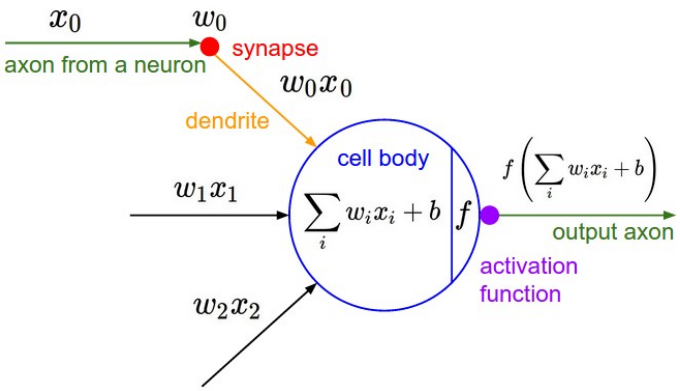

Figura 1.2: Computazione in un neurone artificiale nell'algoritmo di retropropagazione dell'errore
Fonte: http://cs231n.github.io/neural-networks-1/

1.2 Reti neurali ricorrenti

Le reti neurali ricorrenti sono reti in cui le connessioni tra i nodi creano cicli. Il loro grande successo è dovuto alla capacità di trovare relazioni tra serie di dati sequenziali, come le parole di una frase o il valore di un titolo azionario nel tempo.

La struttura di una generica rete neurale ricorrente non differisce enormemente da una rete neurale feed-forward. Può essere, infatti, pensata come molteplici copie della stessa rete, ognuna delle quali è in grado di "passare un messaggio" al successore. L'output della rete è quindi influenzato non solo dai pesi applicati all'input come in una rete feed-forward, ma anche dallo stato "nascosto", il messaggio passato, che dipende dal contesto degli input e output precedenti. Nello specifico l'output al passo t della sequenza è dato da $\tanh([h_{t-1}, x_t])$, dove $[h_{t-1}, x_t]$ indica la concatenazione tra l'output al passo $(t-1)$-esimo e l'input al passo t. Quindi lo stesso input può produrre output diversi a seconda dei precedenti input della sequenza.

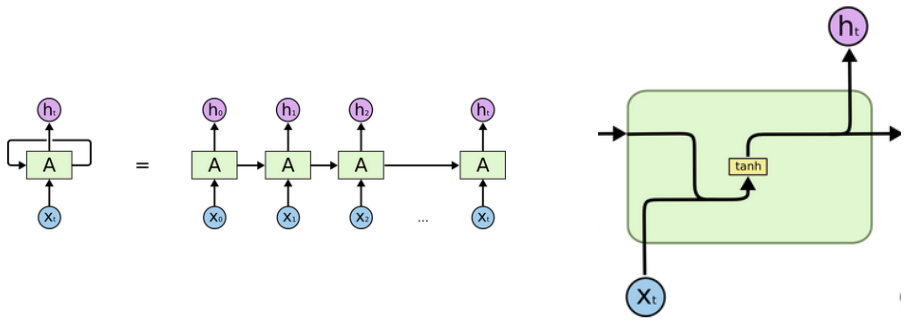

Figura 1.3: Sulla sinistra, rete neurale ricorrente "srotolata". Sulla destra, struttura interna dell'unità di una rete neurale ricorrente.
Fonte: https://colah.github.io/posts/2015-08-Understanding-LSTMs

1.2.1 Long short-term memory

Le LSTM sono un tipo di rete neurale ricorrente molto utilizzato e particolarmente efficace per una serie di applicazioni legate alla comprensione

di testi e suoni, come riconoscimento vocale, traduzione ecc. Rispetto alla generica rete ricorrente una LSTM ha un capacità molto superiore di memorizzare relazioni tra elementi lontani tra loro nella sequenza di input. Questo è reso possibile dalla struttura interna dell'unità LSTM, composta da quattro elementi: lo stato interno della cella, l'input gate, l'output gate e il forget gate.

Lo stato C della cella è la memoria della rete. Contiene una serie di informazioni che possono essere modificate dai gate lungo la catena mediante piccole trasformazioni lineari. È l'elemento chiave che rende così efficaci le LSTM nel trovare relazioni tra dati sequenziali.

I gate sono un modo per filtrare le informazioni che devono o non devono proseguire nella rete, e sono implementati con livelli densi con diverse funzioni di attivazione. Il forget gate si occupa di determinare quali informazioni eliminare dallo stato della cella C_{t-1} in base ai valori dell'input x e dell'output al passo precedente h_{t-1}. L'input gate decide quali nuove informazioni derivanti da x e h_{t-1} memorizzare nello stato C_t. Infine, l'output gate determina quali informazioni andranno a costituire l'output al passo attuale h_t.

Di seguito l'implementazione in Keras:

```
def lstm(xsize, csize, hsize):
    x = Input(shape=(xsize,))
    c = Input(shape=(csize,))
    h = Input(shape=(hsize,))
    forget_gate = Dense(csize,activation="sigmoid",
        name="forget")
    input_gate = Dense(csize,activation="sigmoid",
        name="input")
    state_gate = Dense(csize,activation="tanh", name="state")
    output_gate = Dense(hsize,activation="sigmoid",
        name="output")
    b = concatenate([x,c])
    f = forget_gate(b)
    i = input_gate(b)
    C = state_gate(b)
```

```
o = output_gate(b)
new_C = keras.layers.multiply([c, f])
new_C = keras.layers.add([
    new_C,
    keras.layers.multiply([i,C])])
new_h = Dense(hsize,activation="tanh")(new_C)
new_h = keras.layers.multiply([new_h,o])
return Model(inputs=[x,c,h],outputs=[new_C,new_h])
```

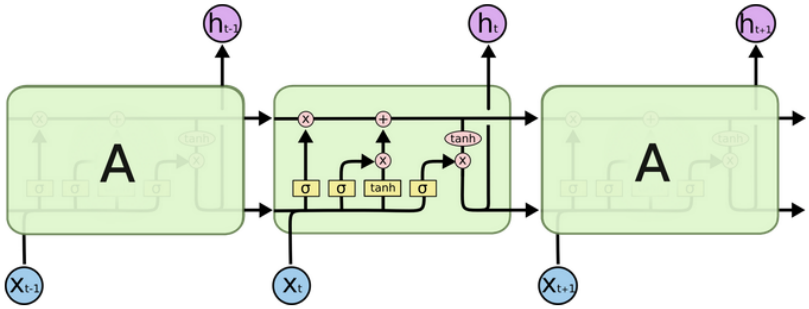

Figura 1.4: Struttura interna di una LSTM
Fonte: https://colah.github.io/posts/2015-08-Understanding-LSTMs

1.3 Autoencoders

Gli autoencoders sono un tipo di rete neurale utilizzata per ottenere una rappresentazione compressa di un dato con molte dimensioni, ad esempio un'immagine.

La struttura comprende una rete neurale detta encoder, che ha il compito di comprimere l'input in un vettore z di piccola dimensione, e una rete neurale detta decoder, che preso in input z tenta di ricostruire l'immagine di partenza. Una funzione di costo che valuta la differenza tra immagine di partenza e immagine ricostruita permette alla rete di "imparare" e di ricostruire immagini sempre più simili.

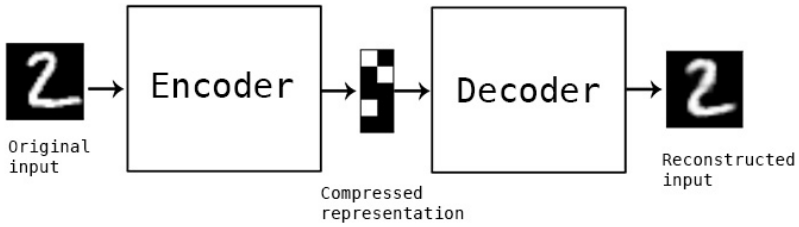

Figura 1.5: Struttura di un autoencoder
Fonte: https://blog.keras.io/building-autoencoders-in-keras.html

1.4 Funzioni di costo

In questo capitolo richiamiamo la definizione di alcune funzioni di costo di comune impego nell'ambito dei Variational Autoencoders.

1.4.1 Cross-Entropy

La Cross-Entropy tra due distribuzioni di probabilità P e Q, su uno stesso insieme di eventi C misura la dimensione di codifica media necessaria a riconoscere un elemento dell'insieme C se la codifica usata è ottimizzata per la distribuzione approssimata Q piuttosto che per la distribuzione reale P.

È definita come:

$$\begin{aligned} CE = H(P,Q) &= \mathbb{E}_{x\sim P}[-\log Q(x)] \\ &= -\sum_{i=1}^{C} P(i)\log Q(i) \end{aligned}$$

Binary Cross-Entropy

La binary Cross-Entropy è un caso particolare di Cross-Entropy in cui il numero di eventi in C è pari a due. Con i_1 e i_2 indichiamo gli eventi

dell'insieme C.

$$\begin{aligned} BCE = H(P,Q) &= \mathbb{E}_{x\sim P}[-\log Q(x)] \\ &= -\sum_{i=1}^{C=2} P(i)\log Q(i) \\ &= -P(i_1)\log Q(i_1) - P(i_2)\log Q(i_2) \end{aligned}$$

Considerando che $P(i_1) = 1 - P(i_2)$, possiamo riscrivere la formula come

$$BCE = -P(i_1)\log Q(i_1) - (1 - P(i_1))\log(1 - Q(i_1))$$

Cross-Entropy come funzione di costo

Per le sue proprietà la Cross-Entropy viene utilizzata come funzione di costo. Infatti, comparando le previsioni Q del modello con la vera distribuzione di probabilità P, data dai dati già classificati in un contesto supervisionato, la Cross-Entropy diminuisce man mano che le previsioni diventano sempre più accurate. Inoltre diventa zero se la previsione è perfetta.

1.4.2 Errore quadratico medio

L'errore quadratico medio (Mean squared error o MSE) è una misura dello scarto quadratico medio fra i valori dei dati stimati $\tilde{y}_i$ e i valori dei dati osservati y_i:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

1.4.3 Divergenza Kullback-Leibler

La divergenza Kullback-Leibler (D_{KL}) è una misura della differenza tra due distribuzioni, ci dice, infatti, quanto la distribuzione di probabilità Q

approssima la distribuzione di probabilità P:

$$\begin{aligned} D_{KL}(P||Q) &= H(P,Q) - H(P) \\ &= \mathbb{E}_{x\sim P}[-\log Q(x)] - \mathbb{E}_{x\sim P}[-\log P(x)] \\ &= \mathbb{E}_{x\sim P}[\log \frac{P(x)}{Q(x)}] \end{aligned}$$

dove $H(P,Q)$ è la cross-entropy tra P e Q e $H(P)$ l'entropia di P.

Capitolo 2

Modelli generativi

2.1 Variational Autoencoders

L'autoencoder, per come è strutturato, rimane un'architettura un po' fine a se stessa. Il decoder potrebbe essere uno strumento utile per generare contenuti, eliminando l'encoder. Tuttavia, lo spazio delle variabili latenti prodotto da un autoencoder è costituito da una serie di punti sparsi e senza una struttura precisa. Campionando casualmente lo spazio latente risulterebbe estrememente improbabile ottenere un vettore di variabili che corrisponda a una codifica ragionevole di un dato di input, precludendo la possibilità di generare contenuti realistici.

Nel caso di un Variational Autoencoder, la struttura di base rimane la stessa di un autoencoder, con la sola differenza che l'encoder non genera più un vettore di variabili latenti, bensì, per ogni variabile, una media μ e una varianza Σ (vedi e.g. [6]). Dalla distribuzione normale con media μ e varianza Σ viene poi campionato lo z che viene preso in input dal decoder. Questo procedimento consente di definire, per ogni immagine del training set, non solo un punto singolo nello spazio latente, bensì un punto e un suo intorno (vedi Fig. 2.1).

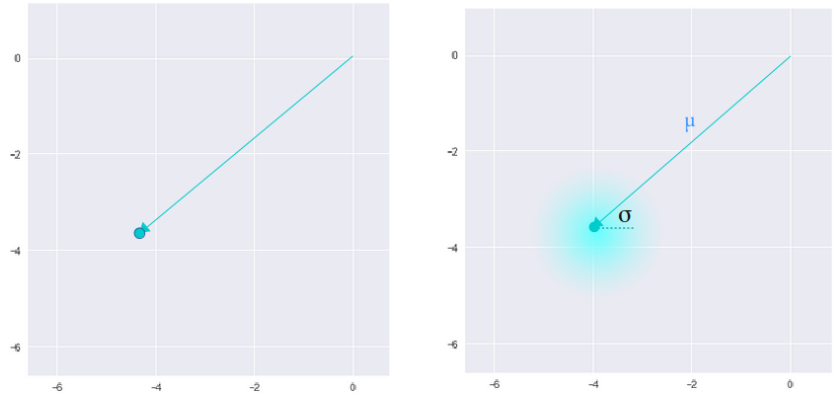

Figura 2.1: Differenza tra un punto nello spazio latente creato da un generico autoencoder e uno creato da un Variational Autoencoder con media e varianza
Fonte: https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf

Tuttavia, non si è così risolto il problema della struttura dello spazio latente. Abbiamo ancora punti che, sebbene offrano una maggior copertura, sono ancora sparsi. La strutturazione viene ottenuta aggiungendo alla funzione di costo del modello la divergenza di Kullback-Leibler tra la distribuzione prodotta dall'encoder e la distribuzione normale con media 0 e varianza 1. Questo nuovo elemento spinge la rete a generare coppie μ e Σ che si collochino in questa distribuzione e da cui sia poi possibile campionare in futuro.

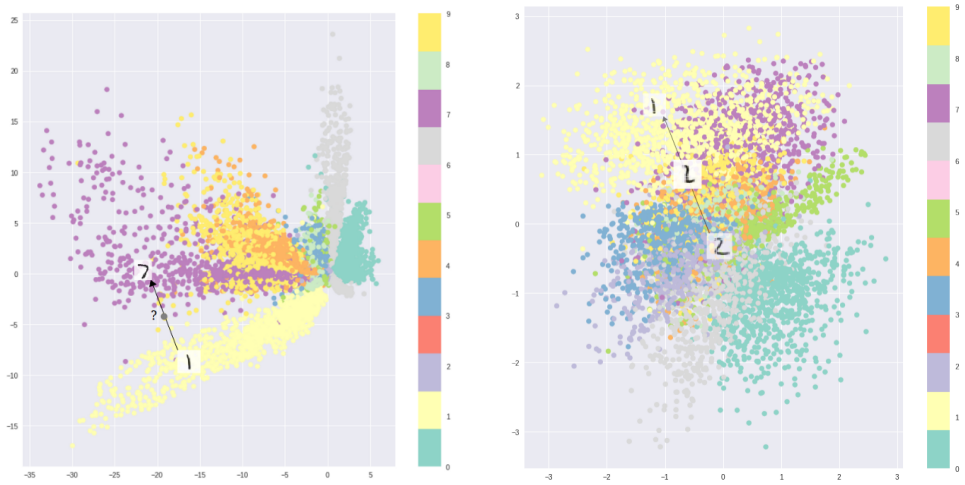

Figura 2.2: A sinistra, struttura spazio latente di un autoencoder allenato su MNIST solo con una funzione di costo per la ricostruzione. A destra, struttura spazio latente di un autoencoder allenato su MNIST con anche la divergenza di Kullback-Leibler nella funzione di costo.
Fonte: https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf

2.1.1 Variance Law

Per ogni elemento i del training set, l'encoder di un VAE genera una distribuzione Gaussiana con media μ_i e una varianza Σ_i. Si hanno quindi tante Gaussiane distinte. Il modello dello spazio latente complessivo del VAE è quindi un Gaussian Mixture Model (GMM) ([16]). Per un GMM, la media è la media delle medie delle singole Gaussiane e (nel caso in cui tale media sia 0) la varianza è pari alla varianza delle medie più la media delle varianze delle componenti. Essendo la divergenza di Kullback-Leibler, della funzione di costo, calcolata tra la distribuzione del GMM e una Gaussiana di media 0 e varianza 1, anche media e varianza del GMM devono, idealmente, arrivare ad assumere tali valori (si veda [1]).

2.1.2 Variabili inattive

Il vincolo imposto dalla Kullback-Leibler spinge a creare uno spazio latente con distribuzione simile alla Gaussiana $\mathcal{N}(0, 1)$. All'inizio del training questo porta quindi l'encoder a definire una varianza molto vicina a uno. Man mano che il training avanza, questa diminuisce visto l'alto numero di immagini distinte che devono essere codificate.

Tuttavia, in alcuni casi, questo non succede. Infatti, l'aumento della dimensione dello spazio latente porta alcune delle variabili a non essere utilizzate. Questo problema è noto come il fenomeno del collasso delle variabili latenti ([2],[3]). Una variabile inutilizzata è caratterizzata da una varianza con valore medio vicino a uno.

Di seguito il codice per il calcolo della Variance Law e del numero di variabili inattive, in Keras:

```
for i in range(timesteps):
    #get encoder prediction at timestep i
    zmean,zlogvar,_ =
       encoder[i].predict([test_images],batch_size=100)
    #calculate variance of mean
    zvar = np.var(zmean, axis=0)
    #calculate mean of variance
    mean_v = np.mean(np.exp(zlogvar),axis=0)
    vl = zvar+mean_v
    inactive_variables_count = 0
    #iterate on latent variables
    for j in range(0,latent_dim):
        if mean_v[j] > .9:
          inactive_variables_count += 1
    var_law[i] = np.mean(vl)
```

2.2 Fréchet Inception Distance

La distanza di Fréchet è una metrica per misurare la distanza tra due distribuzioni. Nel caso della Fréchet Inception Distance (FID) le distribuzioni

considerate sono i vettori delle attivazioni di un layer opportuno della rete Inception-v3 [18] relativi a due gruppi di immagini di medesima cardinalità (ad esempio immagini reali e immagini generate). Il layer che viene tipicamente considerato è l'ultimo layer della parte convolutiva, prima dell'ultimo maxpooling di dimensione 4096. Si considerano i pesi della rete pre-allenata su ImageNet [5]. Reti come Inception-v3, allenate su milioni di immagini di ImageNet, permettono di estrarre features "general purpose", utili per la caratterizzazione di immagini arbitrarie (che è lo stesso principio su cui si fonda il transfer learning). Se i due gruppi di immagini sono simili, anche gli insiemi delle loro attivazioni avranno distribuzioni simili, con FID bassa. La diversità delle immagini è un altro aspetto fondamentale. Infatti, componendo uno dei due gruppi con pochi elementi ripetuti dell'altro, si può notare che la FID aumenta notevolmente, seppur le immagini siano le stesse. Questo avviene poichè non viene calcolata la similitudine tra le immagini, bensì la differenza tra le due distribuzioni, che sono molto diverse.

Dal punto di vista formale, Fréchet Inception Distance è definita in questo modo:

$$FID = ||\mu_r - \mu_g||^2 + Tr(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{\frac{1}{2}})$$

dove $X_r \sim \mathcal{N}(\mu_r, \Sigma_r)$ e $X_g \sim \mathcal{N}(\mu_g, \Sigma_g)$ e Tr calcola la sommatoria degli elementi sulla diagonale.

Di seguito l'implementazione in Keras:

```
tfgan = tf.contrib.gan
activations1 = tf.placeholder(tf.float32, [None, None], name
    = 'activations1')
activations2 = tf.placeholder(tf.float32, [None, None], name
    = 'activations2')
fcd =
    tfgan.eval.frechet_classifier_distance_from_activations(activations1,
    activations2)
base_model = InceptionV3(weights='imagenet',
    include_top=False)
x2 = base_model.output
```

```
x2 = GlobalAveragePooling2D()(x2)
model = Model(inputs=base_model.input, outputs=x2)
size = 299

def activations2distance(act1, act2):
     return fcd.eval(feed_dict = {activations1: act1,
         activations2: act2})

def get_fid(images1, images2):
    #resizing images
    images1 = tf.image.resize_bilinear(images1, [size,
       size]).eval()
    images2 = tf.image.resize_bilinear(images2, [size,
       size]).eval()
    #get Inception-v3 activations
    act1 = model.predict(images1)
    act2 = model.predict(images2)
    #calculate distance from activations
    fid = activations2distance(act1, act2)
    return fid
```

2.2.1 Risultati degli esperimenti

Di seguito sono presentati i risultati di alcuni esperimenti realizzati allo scopo di mostrare le proprietà delle immagini valutate dalla Fréchet Inception Distance, ossia qualità e diversità.

FID tra prime e seconde 1000 immagini MNIST	11.1349125
FID tra 1000 immagini MNIST e rumore casuale	431.3836

Tabella 2.1: La tabella mostra i risultati ottenuti calcolando la Fréchet Inception Distance su gruppi diversi di immagini, nel primo caso immagini molto simili, nel secondo molto diverse. È evidente che la FID dia buoni valori per le immagini simili, qualitativamente migliori se uno dei due gruppi è composto da immagini realistiche, e valori pessimi se le immagini non rappresentano oggetti significativi, in questo caso solo rumore casuale.

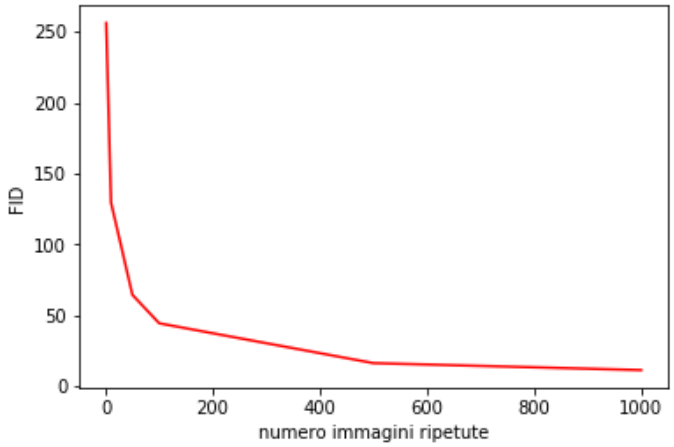

Figura 2.3: Il grafico mostra i valori della Fréchet Inception Distance al variare del numero di immagini di MNIST ripetute, confrontate con altre 1000 immagini, sempre di MNIST. Con una sola immagine ripetuta per 1000 volte i valori della FID sono estremamente alti, mentre con 1000 immagini tutte diverse i valori sono bassi. Questo dimostra che avere immagini molto simili, ma non diversificate, porta a valori pessimi con la metrica FID.

Capitolo 3

Deep Recursive Attentive Writer

Deep Recursive Attentive Writer (DRAW) è un'architettura di rete neurale per la generazione di immagini. La rete è composta da un Variational Autoencoder la cui iterazione permette la costruzione di immagini complesse. La pubblicazione originaria prevede anche un meccanismo di attenzione che tuttavia non è stato trattato e che quindi non verrà approfondito.

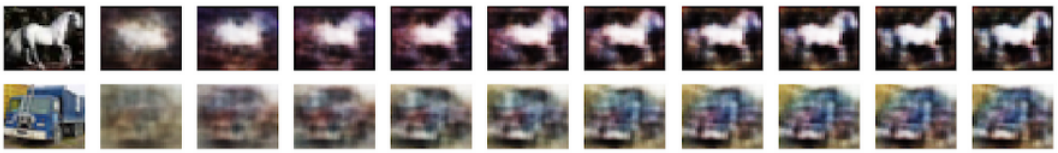

Figura 3.1: Esempio di ricostruzione iterativa (nella prima colonna l'immagine reale, nelle altre la ricostruzione dopo ogni passo). Si noti il miglioramento dell'immagine dopo ogni timestep.

3.1 Struttura della rete

Per la realizzazione del meccanismo iterativo di generazione, DRAW utilizza due reti neurali ricorrenti, LSTM in questo caso, una come encoder e una come decoder.

La differenza sostanziale con il VAE sta nel ruolo del decoder. L'output del decoder è, infatti, aggiunto sequenzialmente alla distribuzione che ha il compito, all'ultimo passo, di generare i dati. Questo è in contraddizione con il generare tale distribuzione in un singolo passo, come avviene tipicamente nei VAE. Ad ogni iterazione un canvas c viene aggiornato in base ai valori ritornati dal decoder. Il canvas al passo finale c_T viene poi "filtrato" dalla funzione di attivazione sigmoide che lo trasforma nell'immagine finale.

La funzione di costo utilizzata per il training del modello è la stessa di un generico VAE. Per valutare la ricostruzione sono state utilizzate, alternatamente, la binary Cross-Entropy e l'errore quadratico medio. Per forzare lo spazio latente ad assumere una distribuzione Gaussiana, la divergenza di Kullback-Leibler. L'errore finale, utilizzato nel training, corrisponde alla somma tra la componente di ricostruzione e la Kullback-Leibler.

Di seguito l'implementazione del modello DRAW in Keras:

```
for t in range(timesteps):
    xhat = Subtract()([x,Activation("sigmoid")(canvas)])
    enc_input = concatenate([x,xhat,h_dec],axis=-1)
    # encoder
    if lstm_shared_weights:
        c_enc, h_enc = encoderRNN([enc_input,c_enc,h_enc])
    else:
        c_enc, h_enc = encoders[t]([enc_input,c_enc,h_enc])
    z_mean[t] = Dense(latent_dim)(h_enc)
    z_log_sigma[t] = Dense(latent_dim)(h_enc)
    z[t] = Lambda(sampling,
       name="z_sampling_layer_"+str(t))([z_mean[t],
       z_log_sigma[t]])
    blayer = blayers[t](z[t])
    kll[t] = Lambda(kl_loss, output_shape=(1,))([z_mean[t],
       z_log_sigma[t]])
    totalloss = totalloss + kll[t]
    #decoder
    if lstm_shared_weights:
        c_dec, h_dec = decoderRNN([blayer,c_dec,h_dec])
    else:
```

```
        c_dec, h_dec = decoders[t]([blayer,c_dec,h_dec])
    h_decd = cdenselayers[t](h_dec)
    #new canvas as sum of previous and
    canvas = Add()([canvas,h_decd])
    output[t] = Activation("sigmoid")(canvas)

#binary Cross-Entropy for reconstruction loss
if recon_func_used == "bce":
    rec_loss =
       K.mean(K.binary_crossentropy(x,output[timesteps-1]),
       axis=-1)
#mean squared error for reconstruction loss
if recon_func_used == "mse":
    rec_loss = mean_squared_error(x,output[timesteps-1])
#total loss
vae_loss = K.mean(rec_loss + totalloss/timesteps)
```

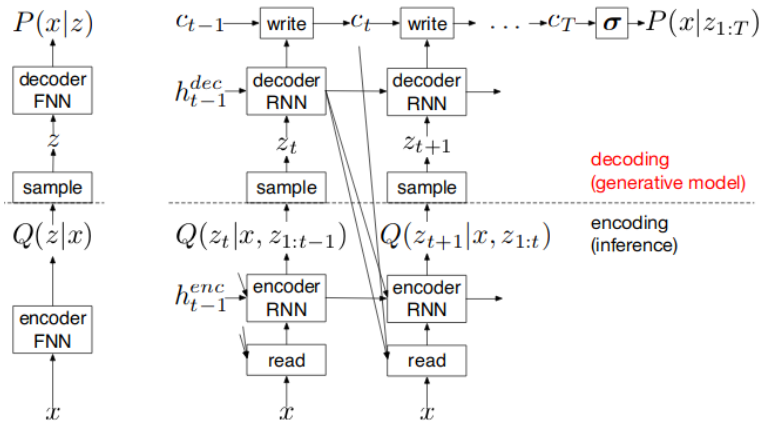

Figura 3.2: Sulla sinistra, tipica struttura di un VAE. Sulla destra, struttura della rete DRAW
Fonte:[9]

3.2 Generatore

Il generatore è la parte della rete in grado di generare contenuti. Nel modello DRAW, un'immagine può essere generata campionando iterativamente

vettori di variabili latenti z_t dalla distribuzione Gaussiana $\mathcal{N}(0, 1)$ e eseguendo il decoder per aggiornare il canvas c_t. Al passo finale T, applicando la funzione di attivazione sigmoide su c_T si ottiene l'immagine generata.

Di seguito l'implementazione in Keras del generatore:

```
def sampling_gen(args):
    return K.random_normal(shape=(batchsize, latent_dim),
       mean=0., stddev=1.0)

for t in range(timesteps):
    z_gen = Lambda(sampling_gen)(cond)
    if lstm_shared_weights:
        blayer = blayers[t](z_gen)
    else:
        blayer = z_gen
    if conditional:
        blayer = concatenate([blayer, cond], axis=-1)
    if lstm_shared_weights:
        c_dec_gen, h_dec_gen = decoderRNN([blayer,
           c_dec_gen, h_dec_gen])
    else:
        c_dec_gen, h_dec_gen = decoders[t]([blayer,
           c_dec_gen, h_dec_gen])
    h_decd_gen = cdenselayers[t](h_dec_gen)
    canvas_gen = Add()([canvas_gen, h_decd_gen])
    output_gen[t] = Activation("sigmoid")(canvas_gen)
```

3.3 Risultati ottenuti

Di seguito vengono presentati i risultati ottenuti dalla rete DRAW allenata con dataset Cifar10. In particolare si confrontano i valori dell'errore di ricostruzione, della Fréchet Inception Distance e del numero di variabili inattive, al variare dei timesteps. La dimensione considerata per lo spazio latente complessivo è pari a 128, che è anche la dimensione dello stato in-

terno dell'LSTM. Con spazio latente complessivo si intende il prodotto tra il numero di timesteps e numero di variabili latenti per timestep.

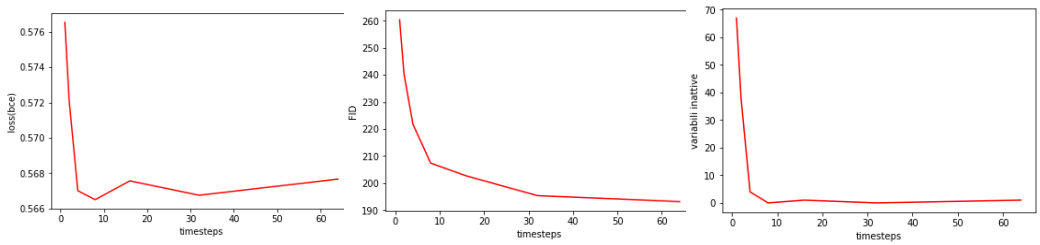

Figura 3.3: A sinistra: variare dell'errore di ricostruzione, calcolato con la funzione binary Cross-Entropy, al variare del numero di timesteps. Al centro: impatto dei timesteps sul valore della FID. A destra: numero di variabili latenti inutilizzate in funzione del numero di timesteps.
Per tutti e tre i valori, l'aumento del numero di iterazioni del modello porta a un miglioramento significativo soprattutto all'inizio, quando il numero di passi è ancora basso. In seguito i valori cominciano a stabilizzarsi e aumentare ancora i timesteps ha un impatto limitato sui risultati della rete.

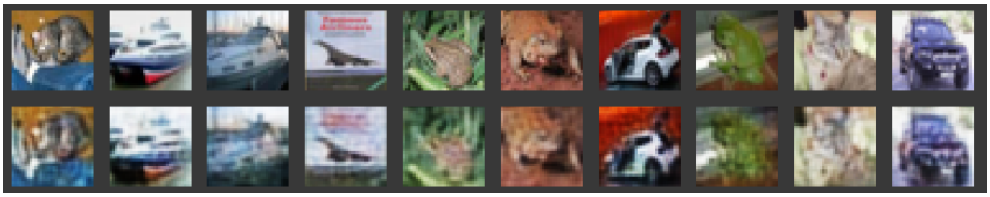

Figura 3.4: Esempi di ricostruzione su Cifar10 (nella prima riga l'immagine reale, nella seconda quella ricostruita). Architettura utilizzata: 16 timesteps, 24 variabili latenti per timestep, 1024 dimensione stato interno LSTM.

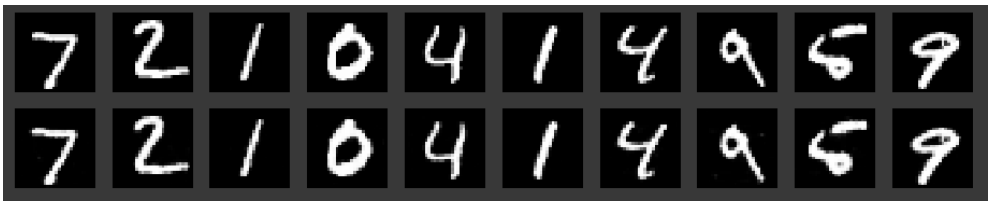

Figura 3.5: Esempi di ricostruzione su MNIST (nella prima riga l'immagine reale, nella seconda quella ricostruita). Architettura utilizzata: 16 timesteps, 16 variabili latenti per timestep e 256 dimensione stato interno LSTM.

Capitolo 4

Estensione condizionale

4.1 Conditional Variational Autoencoder

Un Conditional Variational Autoencoder (CVAE)[17] è un'estensione di un VAE in cui si ha controllo sul processo di generazione. I VAE, infatti, non sono in grado di generare dati specifici poichè l'encoder produce il vettore delle variabili latenti z solo in base al contenuto dell'immagine x presa in input, non tenendo conto della categoria alla quale x appartiene. Lo stesso avviene per il decoder, che ricostruisce basandosi esclusivamente su z.

Il VAE è quindi migliorabile condizionando encoder e decoder con un'informazione aggiuntiva. Questa modifica è stata testata su DRAW, e la rete precedente è stata modificata come segue.

4.2 Implementazione su DRAW

L'estensione condizionale su DRAW è stata realizzata aggiungendo in input, sia all'encoder che al decoder, l'informazione `cond` sulla categoria dell'immagine in input:

```
#encoder
#add condition in encoder input
enc_input = concatenate([x,xhat,h_dec, cond],axis=-1)
```

```
#decoder
blayer = blayers[t](z[t])
#add condition in decoder input
blayer = concatenate([blayer, cond], axis=-1)
```

4.3 Risultati

Di seguito vengono confrontati i risultati ottenuti dalla rete DRAW allenata con dataset MNIST con l'estensione condizionale e senza. I risultati rendono chiaro l'impatto positivo della condizione nel processo di ricostruzione e generazione, portando a immagini ricostruite e generate visivamente migliori.

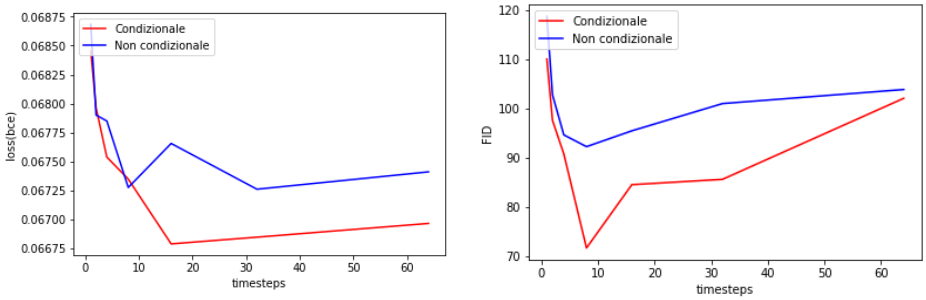

Figura 4.1: A sinistra: l'errore di ricostruzione al variare dei timesteps, con e senza estensione condizionale. A destra: valori della FID al variare dei timesteps, con e senza estensione condizionale. L'architettura utilizzata, ad eccezione della condizione, rimane la stessa: 64 di spazio latente complessivo e 256 dimensione stato interno LSTM.

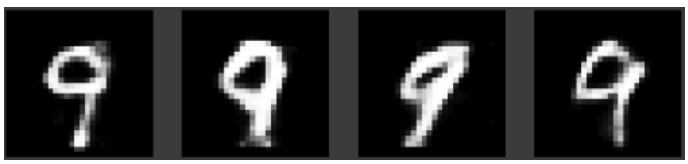

Figura 4.2: Esempio di generazione con condizione 9. Architettura: 8 timesteps, 8 variabili latenti per timestep e 256 dimensione stato interno LSTM.

Conclusioni

In questa tesi è stata descritta e sperimentata una tipologia di modello generativo iterativo, l'architettura Deep Recursive Attentive Writer. Dopo una prima panoramica generale sulle reti neurali e sui modelli generativi, si è mostrata nel dettaglio la struttura di questa rete e quale sia l'idea che giustifichi l'utilizzo dell'iterazione ai fini della generazione. Si è poi provato a capire quale sia nel concreto l'utilità di tale meccanismo, concludendo che l'aumento dei timesteps sia fondamentale per migliorare la ricostruzione, la generazione e la struttura dello spazio latente. È stata poi realizzata l'estensione condizionale del modello DRAW, per generare immagini di una categoria precisa. Il modello esteso è poi stato testato sul dataset MNIST e i risultati ottenuti sono stati confrontati con quelli dell'architettura di base. Si è così potuto osservare che la condizione migliora la generazione, sia a livello di valori della FID, che visivamente. Inoltre, in contemporanea con l'analisi della rete DRAW, sono stati realizzati piccoli esperimenti di "testing" della Fréchet Inception Distance. Le proprietà di tale metrica sono così state verificate e confermate.

L'implementazione del meccanismo di attenzione per la rete DRAW e il testing con nuovi dataset, come ad esempio CelebA [15], sono possibili estensioni di questo elaborato. Un confronto dei risultati così ottenuti con quelli attuali sarebbe un modo per ampliare la conoscenza corrente dei modelli iterativi, permettendo poi in futuro di generare immagini migliori con tali architetture.

Bibliografia

[1] Andrea Asperti. About generative aspects of variational autoencoders. In *Proceedings of the Fifth International Conference on Machine Learning, Optimization, and Data Science - September 10-13, 2019 - Certosa di Pontignano, Siena - Tuscany, Italy*, LNCS (to appear). Springer, 2019.

[2] Andrea Asperti. Sparsity in variational autoencoders. In *Proceedings of the First International Conference on Advances in Signal Processing and Artificial Intelligence, ASPAI 2015, Barcelona, Spain, 20-22 March 2019*, 2019.

[3] Andrea Asperti. Variational autoencoders and the variable collapse phenomenon. *Sensors & Transducers, to appear*, 2019.

[4] François Chollet et al. Keras. https://keras.io, 2015.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[6] Carl Doersch. Tutorial on variational autoencoders, 2016. cite arxiv:1606.05908.

[7] S. M. Ali Eslami, Danilo Jimenez Rezende, Frederic Besse, Fabio Viola, Ari S. Morcos, Marta Garnelo, Avraham Ruderman, Andrei A. Rusu, Ivo Danihelka, Karol Gregor, David P. Reichert, Lars Buesing, Theophane Weber, Oriol Vinyals, Dan Rosenbaum, Neil Rabinowitz, Helen

King, Chloe Hillier, Matt Botvinick, Daan Wierstra, Koray Kavukcuoglu, and Demis Hassabis. Neural scene representation and rendering. *Science*, 360(6394):1204–1210, 2018.

[8] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[9] Karol Gregor, Ivo Danihelka, Alex Graves, and Daan Wierstra. DRAW: A recurrent neural network for image generation. *CoRR*, abs/1502.04623, 2015.

[10] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. *arXiv e-prints*, page arXiv:1706.08500, Jun 2017.

[11] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic Backpropagation and Approximate Inference in Deep Generative Models. *arXiv e-prints*, page arXiv:1401.4082, Jan 2014.

[12] Diederik P Kingma and Max Welling. Auto-Encoding Variational Bayes. *arXiv e-prints*, page arXiv:1312.6114, Dec 2013.

[13] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[14] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[15] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[16] Douglas Reynolds. *Gaussian Mixture Models*, pages 659–663. Springer US, Boston, MA, 2009.

[17] Kihyuk Sohn, Xinchen Yan, and Honglak Lee. Learning structured output representation using deep conditional generative models. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pages 3483–3491, Cambridge, MA, USA, 2015. MIT Press.

[18] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

[19] Guido Van Rossum and Fred L Drake Jr. *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.