# Relazione elaborato finale Computational Learning

# Variational Auto-Encoder: Analisi ed implementazione

*Parte Seconda*

Giuseppe Martinelli Matr: 7093926

**Introduzione**

Lo scopo di questo progetto è stato quello di risolvere tutte le problematiche teoriche e pratiche che si sono riscontrate nella relazione precedente, in merito all' implementazione di un Variational Auto-Encoder. Per questo motivo, ricapitoliamo brevemente le difficoltà che ho avuto in passato e vediamo quali strategie sono state adottate.

**Problemi del passato: Degenerazione output, exploding gradients, …**

E' possibile poter riassumere le difficoltà precedenti nei seguenti punti:

1. Degenerazione dell'output & Immagini deturpate
2. Esplosione varianza, Vanishing Gradients, Instability
3. KL Vanishing
4. Batch Normalization & Input normalization

Questi quattro fattori, che sono anche dipendenti tra loro, portavano il modello a generare le seguenti immagini:

Le quali indicano che il modello di generazione non sta apprendendo correttamente il "mapping" tra lo spazio latente z e lo spazio delle immagini. In effetti il decoder sta sostanzialmente campionando in maniera casuale dallo spazio latente,

**Risoluzione delle problematiche del modello: Degenerazione output**

Il risultato di quella immagine di output degradata è avvenuto per le seguenti motivazioni:

1. Implementazione errata del modello

   Da un'attenta analisi del codice, l'implementazione del layer *Linear* effettuava un reshape dell'input in forma vettoriale. Originariamente, le motivazioni di questo errore non mi erano molto chiare, ma eseguendo un reshape in forma vettoriale si perdeva il senso della combinazione lineare del layer. Cioè, effettuare una combinazione lineare del genere:

   [ B x C x W x H ] → [B x C x W*H] * [ W*H x Out_features]
   → [B x C x Out_features]

   E' completamente diverso, a livello ideologico, rispetto a:

   [ B x C x W x H ] → [ B x C x W x H ].reshape(B x -1) →
   [B x -1] * [ -1 x Out_features]

   In quanto si perde il senso di combinazione lineare: cioè lo scopo è quello di moltiplicare una matrice di pesi W per ogni channel in maniera indipendente

2. Scelta errata di iper-parametri
   Questo punto incide sulla qualità del modello di poter apprendere la distribuzione di probabilità sconosciuta dei dati e della sua capacità di poter generare correttamente. Detto ciò, la scelta corretta di questi valori può essere fatta attraverso una model selection robusta

3. Errore nel calcolo della Loss
   Questo errore è strettamente collegato al primo punto (se non in parte dovuto da esso). Infatti, a causa degli enormi valori di log varianza che si ottenevano dall'encoder, era impossibile poter calcolare la vera varianza effettuando l'esponenziale

```
def encode(self, x):
    x = self.relu(self.img_2hid(x))
    mu, sigma = self.fc_mu(x), self.fc_std(x)
    return self.relu(mu), self.relu(sigma)
```

Ciò oltre a riflettersi sulle prestazioni del modello non permetteva di effettuare il calcolo corretto relativo alla probabilità $p_\theta(x|z) \sim q_\theta(x|z) \sim N(z; \mu, log\sigma)$
La soluzione è stata quella di introdurre dei layer di normalizzazione e di Relu aggiuntivi.

```
def encode(self,x):
    x = self.img_2hid(x)
    #Normalize
    x = self.encode_input_batch(x)
    x = self.relu(x)
    mu,sigma = self.fc_mu(x),self.fc_std(x)
    # Normalize
    mu,sigma = self.norm_mu(mu),self.norm_std(sigma)
    return self.relu(mu),self.relu(sigma)
```

**Risoluzione delle problematiche del modello: Esplosione varianza, Vanishing Gradients, Instability**

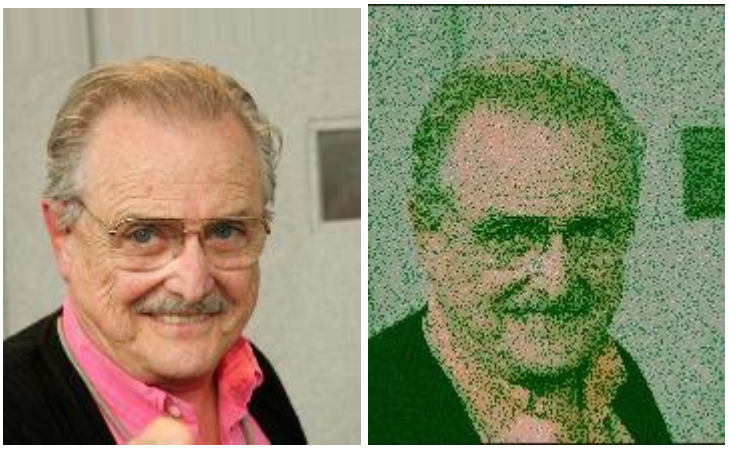
Per poter fronteggiare il problema dell'esplosione della varianza è stato implementato ed introdotto il layer di *Batch Normalization* il quale, attraverso lo shifting e resizing dell'input, ci permette di combattere il fenomeno dell'**internal covariance shift** (https://arxiv.org/pdf/1502.03167.pdf). Inoltre, i layer di batch normalization introducono molti altri effetti benefici all'interno del modello, come ad esempio la possibilità di permetterci di scegliere un learning rate più alto senza la ricaduta nel problema dell'esplosione dei gradienti. Inoltre, questo layer ha anche un effetto di regolarizzazione sul modello, il quale non ha bisogno di ulteriori tecniche aggiuntive di regolarizzazione come il dropout. L'utilizzo di questo layer ha però un costo, infatti esso introduce due nuovi parametri di shift che devono essere appresi durante l'apprendimento. Un altro aspetto importante è il posizionamento del layer di normalizzazione, infatti vi è un acceso dibattito in ambiente accademico, dove molti studiosi sostengono che il posizionamento della regolarizzazione **dopo** la funzione di

attivazione dei neuroni ottenga risultati migliori rispetto alla sua posizione standard antecedente.
L'introduzione del layer di normalizzazione introduce un altro aspetto da tenere in considerazione, se precedentemente nel modello di encoding/decoding vi era la sola presenza di layer Relu, adesso vi è bisogno di un'attenta scelta di funzione di attivazione. Infatti, se il layer BatchNorm ritorna effettivamente un tensore completamente negativo (cosa probabile) si può verificare un effetti di *dead neurons*, in quanto il layer successivo Relu restituirebbe un output completamente vuoto. Per ciò, nella nuova struttura del modello si è deciso di inserire dei layer di Leaky Relu.

```
def decode(self, x):
    first_hidden = self.z_2hidden(x)
    hidden_norm = self.hidden_batch_norm(first_hidden)
    leaky_relu = torch.nn.functional.leaky_relu(hidden_norm)
    hidden_2img_space = self.hidden_2img(leaky_relu)
    img_batch = self.img_batch(hidden_2img_space)
    last_relu = self.relu(img_batch)
```

Per poter affrontare il problema dell'esplosione/annullamento dei gradienti, nel modello è stato inserito anche il *gradient clipping* che introduce però un nuovo iper parametro da dover validare

**Risultati con un VAE lineare**
Dopo le modifiche fatte agli errori discussi e le eventuali aggiunte, sono stati ottenuti i seguenti risultati con un VAE completamente lineare, sia in termini di encoder e sia in termini di decoder



Rispetto alla iterazione precedente di questo progetto, questo si è dimostrato già un grande risultato, che mi ha fatto sperare di poter costruire anche un modello basato su

reti di convoluzione. Tuttavia, analisi preliminari dell'output di reti CNN hanno portato ad output estremamente degradati (come l'immagine all'inizio di questa relazione), la motivazione per cui ciò accadde non mi è totalmente conosciuta. Una prima analisi ed indagine mi spinge a dover sostenere che il problema di modelli convoluzione/deconvoluzione ricade nella mia implementazione del layer **di deconvoluzione,** in quanto suppongo che in fase di calcolo dei gradienti il layer di deconvoluzione abbia lo stesso problema che il layer convoluzionale ha nella ricostruzione dei gradienti originali (in altre parole, l'implementazione matriciale non permette a torch di aggiustare in modo sensato la matrice di deconvoluzione una volta chiamato opt.step()). Questa mia supposizione è ancora sotto fase di test ed indagine

**Scelta dataset & Scelta iper-parametri: Grid Search**
Come in precedenza (e per una questione strettamente temporale) ho deciso di utilizzare un modello VAE completamente lineare e di addestrarlo sul dataset MNIST. L'utilizzo di questo dataset, oltre ad essere facilmente utilizzabile grazie a molte librerie già esistenti in torch per il suo caricamento/gestione, mi permette di poter eseguire una model selection molto più estensiva: considerando molti più iper parametri e quindi modelli. Quindi, ricapitoliamo velocemente gli iper parametri che devono essere selezionati:

1. **Numero di epoche** → [15]
   La scelta di questo numero può sembrare molto riduttiva, ma bisogna tenere in considerazione che si stanno allenando due modelli aventi profondità 3. Il valore in sé è stato scelto per necessità del tutto pratiche, in modo tale da permetterci di allenare più modelli e di non dover attendere troppo

2. **Batch** → [1000]
   Il seguente valore di batch oltre a permetterci un'esecuzione più veloce è stato scelto anche in virtù del fatto che sembra essere la batch size consigliata per il dataset MNIST

3. **Grandezza spazio nascosto** (*Hidden features)* → [400,600,800]

4. **Grandezza spazio latente** (*Z-features*) → [100,200,400]

5. **Learning Rate → [**1, 1e-5, 3e-2]

6. **Gradient Learning Rate → [**1.0, 2e-4]

Per quanto riguarda gli iperparametri 3,4 la scelta di quei particolari valori deriva da diversi test che sono stati effettuati e che hanno portato a buoni risultati, invece per i parametri 5 e 6, l'utilizzo di learning alti (1) deriva dalla presenza nel modello dei layer di normalizzazione e di gradient clipping.

**Model-Selection: Holdout**

In questo progetto ho deciso di voler utilizzare il metodo Hold-Out per effettuare la model selection. Sebbene questo metodo sia meno robusto, ho preferito scegliere un metodo di selezione "più semplice" per potermi dedicare all'allenamento e selezione di più modelli. Con questa configurazione, ho diviso il dataset MNIST in 80% training, 10% validazione e 10% testing

**Risultati della validazione**

Con il seguente numero di iperparametri, i modelli da dover validare risultano essere **54,** per questioni prettamente legate a risorse computazionali, spaziali e temporali sono riuscito ad allenare e validate con successo 27 modelli, di cui si riportano le statistiche di loss di Elbo e di loss di ricostruzione

| Configurazione | Training Loss | Validation Loss | Reconstruction Loss | Recon Validation Loss |
|---|---|---|---|---|
| 15, 1000, 400, 100, 1, 1.0 | 188717.953125 | 193569.578125 | 46853.734375 | 51276.328125 |
| 15, 1000, 400, 100, 1, 0.0002 (2e-4) | 265885.375 | 264605.6875 | 85097.765625 | 84167.453125 |
| 15, 1000, 400, 100, 1e-05, 1.0 | 265059.59375 | 263087.4375 | 83973.65625 | 83366.1953125 |
| 15, 1000, 400, 100, 1e-05, 0.0002 | 192218.6875 | 192506.84375 | 50146.1875 | 50612.68359375 |
| 15, 1000, 400, 100, 0.03 (3e-2), 1.0 | 192542.890625 | 192352.03125 | 50609.23828125 | 50576.8984375 |
| 15, 1000, 400, 100, 0.03, 0.0002 | 332994.375 | 337683.28125 | 49033.7265625 | 53703.6171875 |
| 15, 1000, 400, 200, 1, 1.0 | 335834.28125 | 337391.75 | 52646.01171875 | 53569.73828125 |

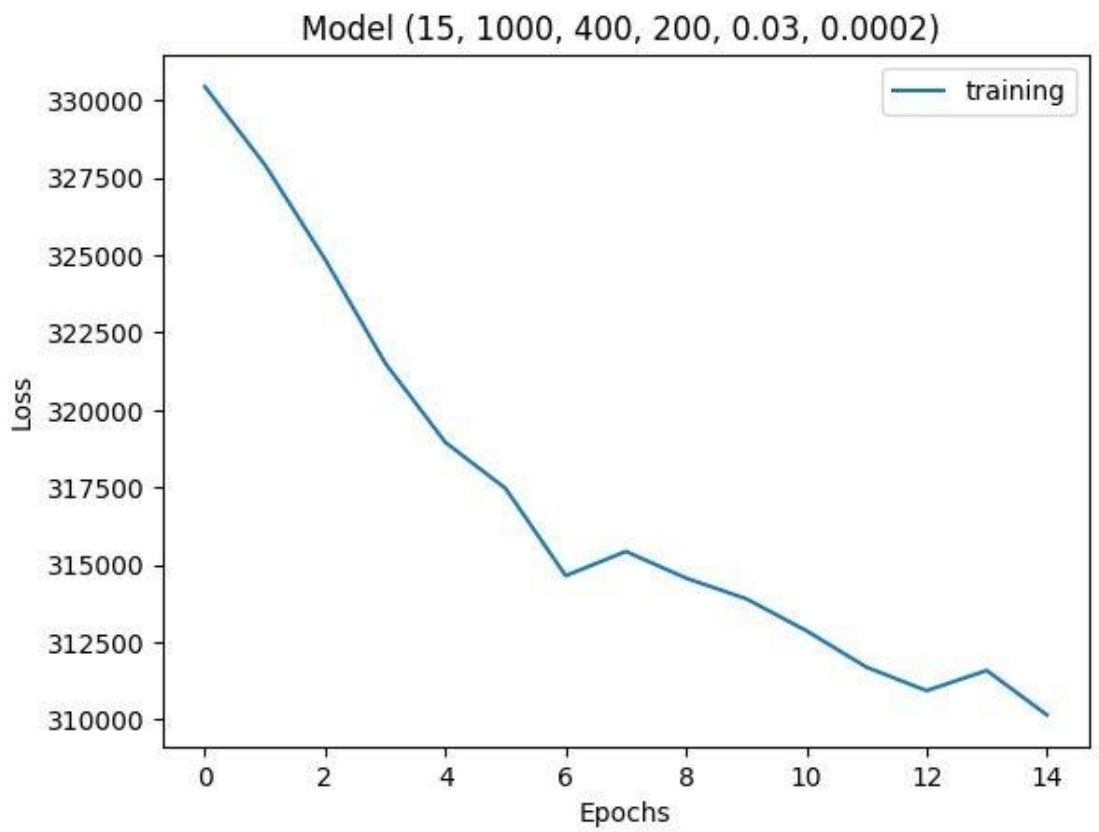
| | | | | |
|---|---|---|---|---|
| 15, 1000, 400, 200, 1, 0.0002 | 443891.3125 | 443657.15625 | 83754.015625 | 84492.9296875 |
| 15, 1000, 400, 200, 1e-05, 1.0 | 444778.28125 | 446822.59375 | 83467.6328125 | 85561.375 |
| 15, 1000, 400, 200, 1e-05, 0.0002 | 333519.4375 | 332861.125 | 50034.37109375 | 49550.0625 |
| 15, 1000, 400, 200, 0.03, 1.0 | 334032.4375 | 335292.90625 | 50195.6171875 | 51590.77734375 |
| **15, 1000, 400, 200, 0.03, 0.0002** | 614127.3125 | 614507.0625 | 45877.89453125 | 46961.37109375 |
| 15, 1000, 400, 400, 1, 1.0 | 620790.8125 | 622898.4375 | 53935.5546875 | 54443.4296875 |
| 15,1000,400,400,1e-05,1.0 | 806971.9375 | 802278.5625 | 82544.9375 | 82910.546875 |
| 15, 1000, 400, 400, 0.03, 0.0002 | 195899.125 | 215140.984375 | 54371.19921875 | 72968.796875 |
| 15, 1000, 600, 100, 1, 1.0 | 220538.359375 | 215072.71875 | 78426.640625 | 73285.09375 |
| 15, 1000, 600, 100, 1, 0.0002 | 263671.875 | 265030.53125 | 83039.203125 | 85131.7890625 |
| 15, 1000, 600, 100, 1e-05, 1.0 | 264015.8125 | 262635.25 | 83062.796875 | 82671.796875 |
| **15, 1000, 600, 100, 1e-05, 0.0002** | 190098.09375 | 189923.53125 | 47923.69921875 | 48096.3984375 |
| 15, 1000, 600, 100, 0.03, 1.0 | 190142.15625 | 190219.8125 | 48408.3515625 | 48268.77734375 |
| 15, 1000, 600, 100, 0.03, 0.0002 | 330854.5 | 332560.3125 | 47244.5625 | 48438.671875 |
| 15, 1000, 600, 100, 0.03, 0.0002 | 336466.875 | 337272.59375 | 52819.56640625 | 54076.36328125 |
| 15, 1000, 600, 200, 1, 0.0002 | 445759.125 | 444013.96875 | 85258.34375 | 84896.9453125 |
| 15, 1000, 600, 200, 1e-05, 1.0 | 445676.1875 | 444897.25 | 83784.203125 | 84785.546875 |

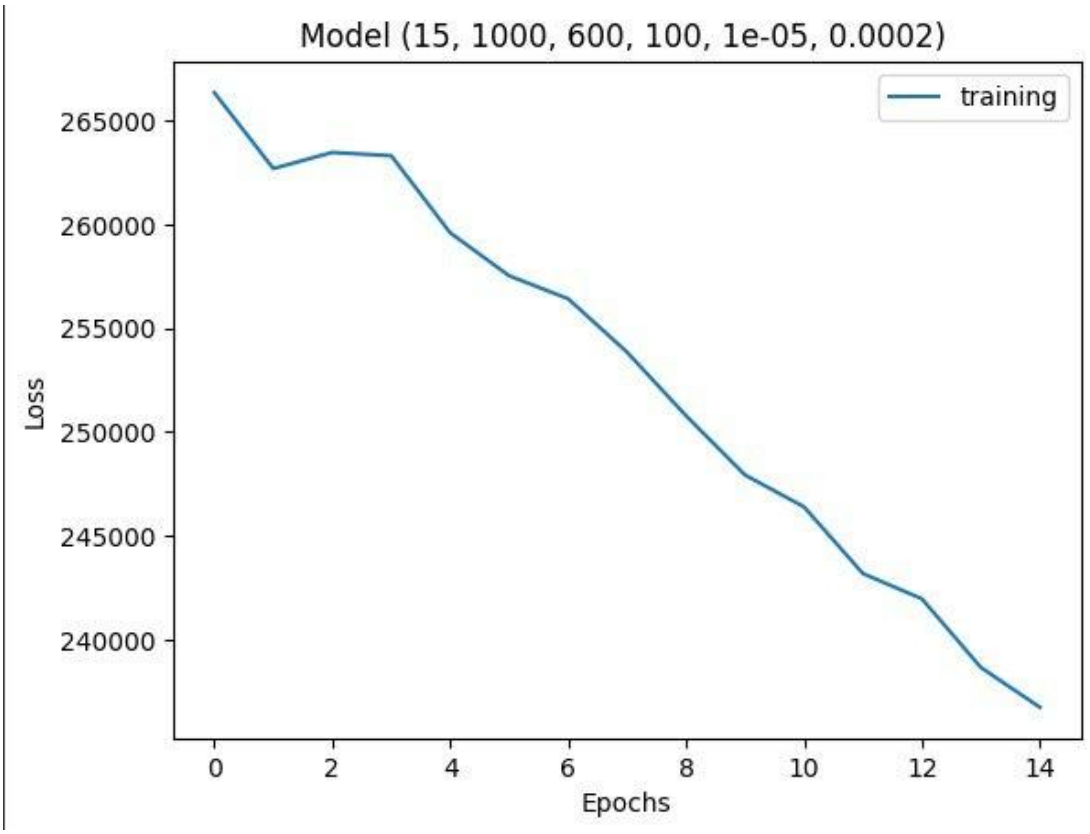
| | | | | |
|---|---|---|---|---|
| 15, 1000, 600, 200, 1e-05, 0.0002 | 334117.84375 | 334117.84375 | 50177.4296875 | 49989.1015625 |
| 15, 1000, 600, 200, 0.03, 1.0 | 333711.0625 | 334200.8125 | 49932.7421875 | 50732.1015625 |
| 5, 1000, 600, 200, 0.03, 0.0002 | 623019.75 | 621137.0 | 54466.51171875 | 54009.0546875 |

Si noti come tra tutti i modelli risultano essere migliori in base alle statistiche di validazione il modello (15, 1000, 400, 200, 0.03, 0.0002) ed il modello (15, 1000, 600, 100, 1e-05, 0.0002). Risulta interessante notare come il primo modello sia migliore in termini di ricostruzione dell'immagine generata, mentre il secondo risulti essere il migliore per Elbo. Ho deciso di voler selezionare entrambi i modelli e di passare alla fase di testing.

**Fase di Testing**

| Configurazione | Training Loss | Recon Loss | Test Loss | Test Recon Loss |
|---|---|---|---|---|
| 15, 1000, 400, 200, 0.03, 0.0002 | 310152.75 | 26381.353515625 | 3111354.0 | 272252.5 |
| 15, 1000, 600, 100, 1e-05, 0.0002 | 236763.359375 | 70727.78125 | 2387431.0 | 725964.625 |


Model (15, 1000, 400, 200, 0.03, 0.0002)
training
330000
327500
325000
322500
320000
317500
315000
312500
310000
Loss
0
2
4
6
8
10
12
14
Epochs


Model (15, 1000, 600, 100, 1e-05, 0.0002)
training
265000
260000
255000
250000
245000
240000
Loss
0
2
4
6
8
10
12
14
Epochs

Dalla fase di testing possiamo anche confermare le nostre supposizioni che abbiamo fatto in validazione, in effetti il modello  (15, 1000, 400, 200, 0.03, 0.0002) risulta essere migliore nella capacità di ricostruzione, mentre il secondo ha una pessima capacità di ricostruzione rispetto al primo, però cattura meglio la distribuzione dei dati