Generazione immagini di numeri in formato MNIST

Gianluca Farinaccio [548013]
Daniele Ferneti [546599]

Deep Learning - A.A. 2024/2025

GitHub: https://github.com/gianlucafarinaccio/mnist-image-generator

1 Obiettivo del progetto

Questo progetto di Deep Learning è stato sviluppato con l'obiettivo di generare nuove immagini di numeri compresi tra 0 e 9 in formato MNIST, ovvero 28x28 in scala di grigi. Inizialmente, si è pensato di utilizzare una architettura di tipo GAN o VAEs, che si sono però rivelate poco soddisfacenti. Si è quindi deciso di optare per architetture più complesse, quali i Transformers.

L'obiettivo finale era creare un sistema in grado di generare un'immagine realistica e graficamente coerente con il numero posto in input.

2 Fasi del Progetto

2.1 Scelta del Dataset

E' stato scelto di usare il data MNIST per i sequenti motivi:

Semplicità:

- È un dataset relativamente piccolo (28x28 pixel in scala di grigi)
- È incluso direttamente in librerie come TensorFlow, quindi può essere caricato e utilizzato con pochissimo codice.

Bilanciamento:

- Contiene 60.000 immagini di training e 10.000 immagini di test, ben bilanciate tra le 10 cifre (0-9), senza grandi problemi di rumore nei dati.

Rilevanza:

- Dataset molto diffuso nel contesto in cui siamo

2.2 Addestramento e valutazione del modello

Di seguito saranno descritti i vari modelli addestrati e i rispettivi risultati ottenuti per ognuno di essi. Per l'addestramento del modello è stato seguito un approccio simile a quello che viene effettuato nelle architetture GAN, ovvero il modello non riceve mai in input immagini del training set ma soltanto numeri interi compresi tra 0-9. Le immagini di training vengono utilizzate alla fine della pipeline, calcolando la loss tra immagine reale ed immagine generata dal modello.

Tutti i modelli sono stati addestrati utilizzando questa filosofia ed iperparametri comuni, ovvero:

- Loss function: BinaryCrossEntropy
- Batch Size: 16
- Epochs: 3

Abbiamo deciso di utilizzare l'architettura dei Transformers, iniziando con una configurazione semplice: un solo layer per l'encoder e uno per il decoder. Nonostante la semplicità di questa prima architettura, si sono ottenuti già dei risultati grafici soddisfacenti ed una loss attorno allo 0,24.

Con l'obiettivo di migliorare i risultati, abbiamo aumentato il numero di layer fino a raggiungere la struttura vista a lezione, ovvero 6 layer per l'encoder e 6 per il decoder. L'obiettivo era ridurre la loss e, con questa modifica, siamo riusciti a diminuirla da 0,24 a 0,22 ottenendo dei risultati grafici ancora più soddisfacenti. Nonostante l'aumento della complessità della architettura, il guadagno ottenuto è stato davvero irrisorio a fronte dei tempi di training estremamente elevati.

Abbiamo quindi tentato un approccio più minimalista, eliminando completamente l'encoder ed i meccanismi di attention, data la tipologia di input. Tuttavia, nonostante la riduzione di complessità della rete, la loss è rimasta stabile intorno allo 0,22, suggerendo che la complessità aggiuntiva dell'encoder e dai meccanismi di attenzione non incideva significativamente sulla qualità dell'output.

Di seguito una descrizione delle architetture addestrate e dei corrispettivi risultati ottenuti.

2.2.1 Model A

Architettura costituita da un decoder che prende in input un intero compreso tra 0-9 e fornisce in output un vettore 28x28 rappresentante una immagine in scala di grigi. Di seguito sono descritti i layer dell'architettura in ordine

- Layer di Embedding: Trasforma l'input (numeri interi 0-9) in un vettore composto da 10 elementi compresi tra -1 e 1. (latent-dim = 10)
- Layer Normalization
- Layer fully connected con dimensione pari a latent-dim

- Layer fully connected con dimensione pari a 2*latent-dim
- Layer normalization
- Layer fully connected con dimensione pari a 784 (28*28)
- Layer normalization

A seguito di un addestramento con gli iperparametri sopra descritti, si è ottenuta una average evaluation-loss pari a 0.2210. Come facile intuire, la cifra con loss minore è quella dell'uno, data la semplicità grafica di rappresentazione di questo numero. Il risultato grafico che genera il modello risulta comunque soddisfacente, come si nota in figura.

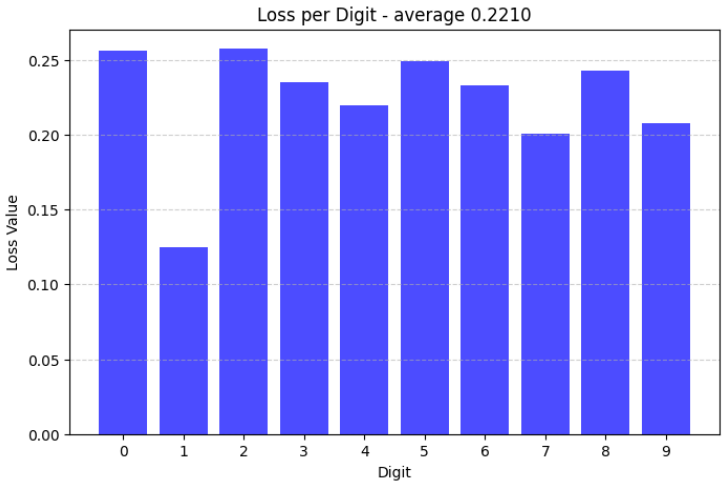

Figure 1: Model A - Evaluation Loss

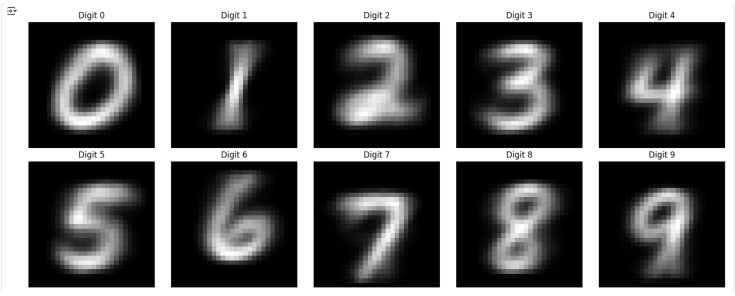

Figure 2: Model A Results

2.2.2 Model B

A seguito, abbiamo pensato di ridurre ancora la complessità della rete, andando a portare il parametro latent-dim pari a 1. La rete è quindi così composta:

- Layer di Embedding: Trasforma l'input (numeri interi 0-9) in un vettore composto da 1 elemento compreso tra -1 e 1. (latent-dim = 1)
- Layer Normalization
- Layer fully connected con dimensione pari a latent-dim
- Layer fully connected con dimensione pari a 2*latent-dim
- Layer normalization
- Layer fully connected con dimensione pari a 784 (28*28)
- Layer normalization

A seguito di un addestramento con gli iperparametri sopra descritti, si è ottenuta una average evaluation-loss pari a 0.2639. Come era facile intuire, la loss è aumentata. Nonostante l'aumento di solo 0.4 possiamo notare come in realtà il risultato grafico sia ben distante da quello ottenuto dal Model A. Come si nota in figura, i numeri generati non risultano più distinguibili.

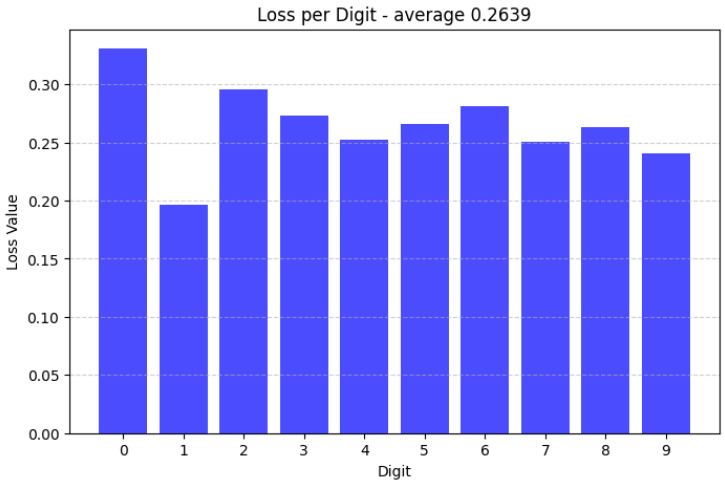

Figure 3: Model B - Evaluation Loss

Ogni sotto encoder è composto da un self-attention layer e un feed-forward layer, mentre ogni decoder è composto oltre che dai due layer appena citati anche un encoder-decoder attention layer Nella fase di addestramento è importante sottolineare il fatto che al modello siano state date solo le etichette dei rispettivi numeri. I valori di loss sono stati ottenuti tramite una binary-cross-entropy fra l'immagine generata dal modello e quella reale.

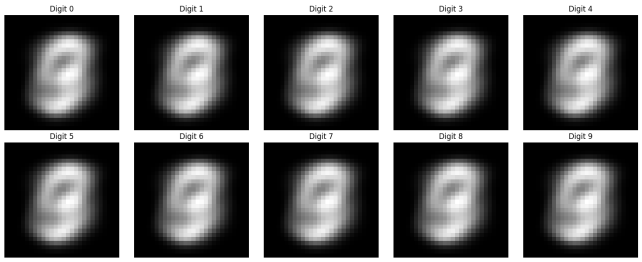

Figure 4: Model B - Results

2.3 Problematiche

Pensare, progettare e sviluppare un codice capace di generare numeri in modo coerente con le richieste non è stato affatto semplice. Inizialmente, abbiamo provato a utilizzare le GAN, ma i risultati sono stati deludenti. Non ottenendo ciò che speravamo, siamo passati ai Variational Autoencoder, ma anche in questo caso la qualità era ancora troppo bassa rispetto alle nostre aspettative.

Alla fine, abbiamo deciso di puntare sui Transformers, cercando di adattarne l'architettura per le nostre esigenze, eliminando gli elementi superflui. In un primo tentativo, durante l'addestramento, abbiamo fornito al modello tutte le immagini di ciascun numero, sperando che imparasse a generarle correttamente. Tuttavia, il modello produceva immagini di buona qualità ma non coerenti con l'etichetta: se gli veniva richiesto un numero specifico, generava un'immagine ben definita, ma senza corrispondenza con il numero richiesto.

Dopo diverse ottimizzazioni, abbiamo affinato l'approccio descritto nella sezione precedente. Così, siamo riusciti a ottenere immagini di qualità discreta, ma soprattutto coerenti con il numero richiesto, raggiungendo finalmente un buon equilibrio tra fedeltà e precisione.