UNIVERSITÀ DEGLI STUDI DI MILANO
FACOLTÀ DI SCIENZE E TECNOLOGIE

DIPARTIMENTO DI INFORMATICA
GIOVANNI DEGLI ANTONI

Corso di Laurea magistrale in
Informatica

UN SISTEMA DI COMPUTATIONAL CREATIVITY PER LA GENERAZIONE MUSICALE

Relatore: Prof. Luca Andrea Ludovico
Correlatore: Prof. Giorgio Valentini, PhD. Giorgio Presti

Tesi di Laurea di:
Matteo Collina
Matr. Nr. 908676

ANNO ACCADEMICO 2018-2019

Questo lavoro è dedicato alla mia famiglia

"Siate affamati!
Siate folli!"

– Steve Jobs

"Chi vuol muovere il mondo
prima muova se stesso"

– Socrate

Ringraziamenti

Ringrazio i miei genitori per avermi aiutato in questo percorso universitario lontano da casa, mio fratello Alessio, i miei nonni Mario e Emilia, i miei zii Andrea e Marewa, i miei cugini Bryan e Mia e tutti i miei amici.

Indice

Capitolo 1

Introduzione

La generazione di musica simbolica da parte di reti neurali è un tema ancora aperto poiché i fattori che influenzano le scelte progettuali sono molteplici. Come parte della composizione automatica dei brani musicali, il problema verrà affrontato in modo progressivo, tenendo conto separatamente delle caratteristiche tipiche di cui è composto il linguaggio musicale: melodia, ritmo e armonia, analizzando successivamente la loro coesistenza.
L'analisi dei brani musicali non si è limitata alla loro struttura simbolica ma è stata estesa allo studio grafico dei layer presenti nelle architetture delle reti neurali utilizzate, per apprendere il cambiamento del loro stato interno da un altro punto di vista più visivo. Per le rappresentazioni simboliche di brani musicali sono presenti due approcci principali: string-based e geometric methods. Un metodo basato su stringhe considera una sequenza musicale simbolica come una stringa di token ed applica algoritmi di rilevamento del modello di stringa sulla sequenza, mentre un metodo geometrico visualizza i pattern musicali come forme che compaiono su una partitura.

ll presente lavoro è organizzato come segue: nel capitolo seguente sarà presente una panoramica del progetto con alcune indicazioni sulle applicazioni attualmente disponibili sul mercato, nel *Capitolo 2* verranno descritte dal punto di vista matematico le reti neurali utilizzate, entrando nel dettaglio del loro funzionamento, analizzando funzioni di perdita e di attivazione implementate. Nel *Capitolo 3* verranno descritti i progetti presenti nello stato dell'arte attuale introducendo tecniche di feedforward e recurrent network, architetture basate su encoder-decoder ed infine tecniche basate sugli spazi latenti non utilizzati nell'elaborato corrente. All'interno del *Capitolo 4* saranno elencate le fasi di elaborazione dei dati, partendo dalla descrizione del procedimento di ottenimento del dataset, passando alla fase di processing per la conversione in file MIDI per poi infine descrivere la traduzione in formato matriciale, necessaria al fine di poter essere interpretata dagli

algoritmi di AI. Nel *Capitolo 5* verrà descritto nel dettaglio il processo di generazione di musica simbolica considerando le seguenti reti neurali: LSTM, Bi-LSTM, GRU, CNN, CNN-LSTM e GAN, definendo per ognuna di essere l'architettura ed i parametri utilizzati, descrivendo inoltre il processo generativo col quale si ottiene il file midi. Nel *Capitolo 6* saranno indicati i risultati ottenuti dalle tipologie di generazione melodica, armonica e del brano musicale completo di melodia ed armonia, oltre alla formalizzazione di alcuni aspetti della creatività computazionale data dai modelli descrittivi FACE e IDEA. Infine nel *Capitolo 7* saranno presenti le conclusioni tratte e gli sviluppi futuri.

Lo scopo di queste tecniche di generazione automatica è di aiutare e facilitare il lavoro dei compositori musicali, partendo da idee preesistenti aventi un certo stile ed appartenenti ad un certo genere musicale. Per alcuni cantautori che non hanno familiarità con la teoria musicale, l'uso dell'IA è diventato molto importante e gli ha permesso di generare nuove idee su cui sviluppare i loro testi come ha affermato Taryn Southern alla testata giornalistica americana The Verge[1]. L'intelligenza artificiale e la musica sono state a lungo intrecciate. Alan Turing, il padre dell'informatica, nel 1951 costruì una macchina che generava tre semplici melodie: la filastrocca, "Baa Baa Black Sheep", la canzone di Glenn Miller, "In the Mood" e l'inno nazionale inglese "God Save the King". Ma mentre la tecnologia ha fatto molta strada, molti sostengono che si sia ancora lontani da un'intelligenza artificiale che possa creare canzoni di successo da sola, tra cui Oleg Stavitsky, CEO e co-fondatore di Endel, un'applicazione che genera ambienti sonori. A causa delle sue limitazioni, l'AI ha creato poche canzoni pop di successo, mentre ha fatto progressi molto più interessanti in due flussi di musica diametralmente opposti: quello funzionale e quello sperimentale.
Per il musicista britannico-iraniano Ash Koosha, lavorare con l'IA ironicamente è stata una svolta emotiva. Ha sviluppato una pop star dell'IA, di nome Yona, che scrive canzoni attraverso software generativo. E mentre molti dei suoi testi sono vaghi e privi di senso, alcuni di essi sono considerati incredibilmente sensibili. Koosha è stato particolarmente sorpreso dalla frase "Colui che ti ama canta canzoni solitarie con note tristi". "Essere così schietto e così aperto, così emotivamente nudo, non è qualcosa che la maggior parte degli umani può fare", ha detto Koosha alla testata TIME.

[1] `https://bit.ly/2zkVxm0`

Capitolo 2

Background su AI e reti neurali

Con i progressi compiuti nel campo dell'IA, l'attenzione del panorama musicale si è focalizzata sulla generazione automatica di brani, considerando la loro struttura simbolica e la loro forma d'onda. Sono stati introdotti modelli generativi di deep learning nel campo della musica, i quali hanno trovato diversi ostacoli, tra cui: la presenza di vincoli a breve e lungo termine, le diverse metriche tra i generi musicali e la difficile interpretazione della creatività rispetto a quella umana.

Il processo di: identificazione della nota successiva che verrà generata ed inserita nello spartito oppure la scoperta del genere musicale al quale appartiene un brano viene denominato **classificazione** e consiste nell'identificare la classe di appartenenza dei dati, basandosi su un set di informazioni preesistenti già classificate. Tale processo può essere *binario*, nel caso la classificazione vada ad indicare se il dato appartiene o meno alla classe, oppure *multinomiale* se il dato può essere associato a più classi. La classificazione può essere di tipo *crisp*, nella quale si assegna una sola classe fra quelle disponibili, oppure *probabilistica* in cui il classificatore assegna all'oggetto da classificare una probabilità di appartenenza a ciascuna delle classi possibili.

Le reti neurali si basano sul modello lineare di **neurone**, il quale presenta un corpo cellulare a cui arrivano impulsi sotto forma di segnali elettrici mediati da neuro trasmettitori. Il neurone li riceve, li elabora e se l'insieme dei segnali supera una certa soglia, produce un output. Il neurone artificiale riceve in ingresso un vettore di *input* numerici moltiplicati scalarmente per un vettore di pesi. Uno dei pesi viene chiamato *bias* ed è collegato ad un input fittizio di valore 1. Dopodiché il neurone applica a tale somma pesata una determinata *funzione di attivazione* (lineare o non lineare) producendo un singolo output.

$$y = activation(\sum_{i=1}^{N} w_i x_i + b)$$

Con y viene indicato l'output, *activation* è la funzione d'attivazione, N è il numero di input, x_i e w_i sono l'input i-esimo ed il suo corrispondente peso mentre b è il bias. Le reti neurali possono essere considerate come delle "black box" in grado di apprendere qualsiasi funzione (lineare o non lineare) sfruttando l'uso di un numero considerevole di stimoli in ingresso. L'apprendimento di un neurone è quindi rappresentato da un ribilanciamento dei pesi al fine di approssimare correttamente la funzione desiderata.
Il singolo neurone artificiale rappresenta la topologia più semplice che si possa avere e, dal punto di vista geometrico, effettua una classificazione in due classi utilizzando un *iperpiano separatore* (retta 2D) come superficie separatrice. Esso è in grado di classificare solamente insiemi linearmente separabili.

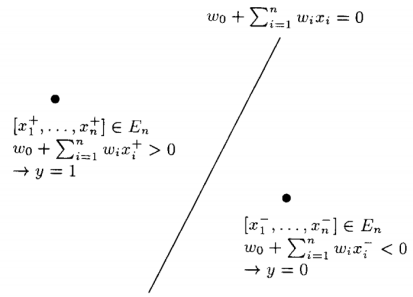


Figura 1: Interpretazione geometrica della funzione computata da un singolo percettrone [3]

Il singolo neurone artificiale può essere collegato ad altri neuroni creando topologie neurali sempre più complesse, le quali permettono di classificare dati non linearmente separabili. Le **deep neural network** sono reti costituite da vari strati di layer in modo tale che gli output di un determinato livello siano gli input dello strato successivo nello stack, facendo si che ogni layer presenti una matrice di pesi.

Sebbene il numero di topologie di reti neurali siano aumentate nel tempo, esse ricadono tendenzialmente in due categorie principali:

- **Feedforward Network** (es. MLP, CNN) : essa presenta un layer di input per la ricezione dei dati, un layer di output costituito da un numero di neuroni pari al numero totale di classi che si desiderano, ed un numero generico di layer nascosti, o hidden layers, ognuno di dimensione arbitraria. Il nome deriva dal fatto che il segnale inserito nella rete (feed) viene propagato in avanti (forward) tra un layer e l'altro.

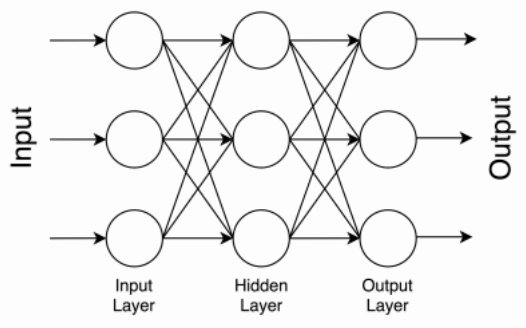


Figura 2: Esempio di una Feedforward Network

- **Recurrent Network** : Questa topologia presenta delle connessioni di feedback, generalmente verso i neuroni dello stesso livello ma è possibile comunque averne verso layer precedenti. Ad ogni step della sequenza (ad esempio a ogni istante temporale t), il livello riceve oltre all'input x_t anche l'output dello step precedente y_{t-1}. Questo consente alla rete di basare le sue decisioni sulla storia passata sfruttando la memoria. Per tale ragione le recurrent network vengono usate per l'analisi di sequenze come ad esempio frasi di un linguaggio naturale, audio etc.

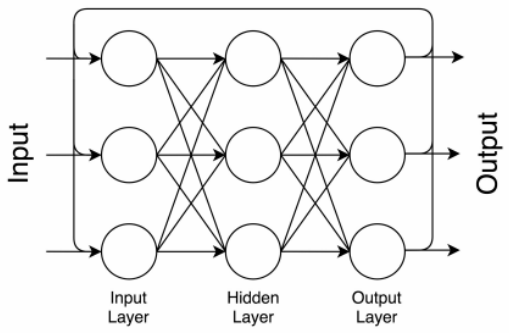


Figura 3: Esempio di una Recurrent Network

In base al task da svolgere vengono scelti la topologia di rete ed il tipo di addestramento più adatti. Quando i dati a disposizione per l'apprendimento non hanno un valore atteso si parla di *addestramento non supervisionato*. Lo scopo di questo approccio è quello di dedurre una funzione che riesca a descrivere la struttura interna o estrarre le caratteristiche di un insieme di dati in input.

2.0.1 Tecnica di discesa del gradiente e backpropagation

Per risolvere problemi di classificazione viene utilizzato l'*apprendimento supervisionato*, il quale presuppone una conoscenza pregressa sul legame tra dati di input e le classi di output associate. Il compito della rete con tale tipologia di apprendimento è quello di minimizzare l'errore di previsione tra i valori in ingresso ed i valori attesi tramite l'applicazione della gradient-descent technique. Lo scopo della classificazione supervisionata è quello di saper predire la classe da associare ai dati che non fanno parte del training set. Al fine di minimizzare le previsioni errate è necessario calcolare l'errore e, procedendo a ritroso, calibrare i pesi in base a quanto essi hanno inciso sull'output errato.

La tecnica della **discesa del gradiente** si basa sul concetto che, per una funzione f(w), la direzione di massima discesa verso un minimo locale in un determinato punto w, corrisponde a quella determinata dall'opposto del suo gradiente p = -∇ f(w) [1], il quale rappresenta il vettore delle derivate parziali di f rispetto a ciascuna componente di w. L'algoritmo presenta due fasi: nella prima viene calcolato il valore di output di f(w), mentre nella seconda si calcola il valore di w al fine di minimizzare la funzione nel modo seguente:

$$w = w - \Delta \text{ , dato:}$$

$$\Delta = -\eta \cdot p = \eta \cdot \nabla f(w)$$

in cui η è detto learning rate e deve diminuire ad ogni iterazione per permettere all'algoritmo di convergere al minimo. L'algoritmo termina se il valore della funzione non decresce per N iterazioni consecutive oppure se il minimo viene trovato.

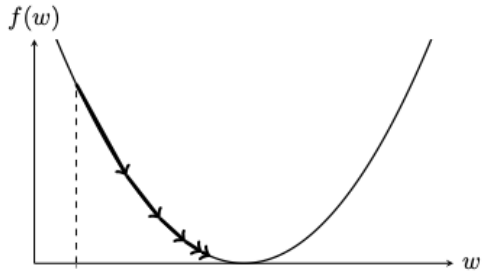


Figura 4: Discesa del gradiente in una funzione

Nel caso del *neurone singolo*, f(w) rappresenta la **funzione di perdita** che computa l'errore di previsione del singolo output rispetto al valore atteso, mentre

w costituisce l'insieme dei suoi pesi (w_i, ...,w_n), ed n è il numero di input. La funzione f(w) risulta essere:

$$f(w) = loss(target, activation(sum(w, input)))$$

dove *loss* è la funzione di perdita, *activation* è la funzione d'attivazione, *target* è il valore atteso e *sum* rappresenta la somma pesata tra gli input ed i pesi w_i i quali sono la vera incognita della funzione.

Per il calcolo della derivata della funzione di perdita rispetto ad un particolare peso w_i, è sufficiente seguire la *chain rule*, una regola di derivazione con la quale è possibile calcolare semplicemente la derivata composta di due funzioni differenziabili:

$$\frac{\delta loss}{\delta w_i} = \frac{\delta loss}{\delta activation} \cdot \frac{\delta activation}{\delta weighted\_sum} \cdot \frac{\delta weighted\_sum}{\delta w_i}$$

É necessario quindi che sia la funzione di attivazione che la funzione di perdita siano differenziabili. Si ha in tal caso che:

$$\Delta = \eta \cdot [\frac{\delta loss}{\delta w_1}, ..., \frac{\delta loss}{\delta w_n}]$$

Per quanto riguarda le **reti multistrato** non è chiaro quali siano i livelli di attivazione target che dovrebbero essere forniti agli hidden layer, pertanto senza i target non è chiaro come i pesi delle unità nascoste possano variare. Uno dei metodi che è possibile adottare è quello della discesa del gradiente. Ogni peso quindi, secondo questa strategia, viene modificato in modo da seguire il metodo della discesa del gradiente considerando l'errore per l'intera rete. É appunto questa la strategia adottata dall'algoritmo di apprendimento detto *backpropagation*. Gli obiettivi che si vogliono ottenere sono:

- apprendimento
- generalizzazione
- convergenza, ovvero ridurre l'errore globale E al variare dei pesi.

Dal punto di vista metodologico si procede in due fasi: nella prima (*forward*) ci si muove dagli ingressi verso le uscite calcolando le attivazioni di tutti i neuroni; nella seconda (*backward*) ci si muove a ritroso dalle uscite verso gli ingressi calcolando di quanto debbano essere modificati i pesi uno per uno.

2.0.2 Funzioni di attivazione

Ogni funzione di attivazione ha delle caratteristiche che la rendono più adatta a certi domini applicativi piuttosto che ad altri. Durante la definizione delle reti del progetto, la funzione di attivazione dei layer di output è stata scelta a priori in base all'obiettivo prefissato, ovvero la selezione della nota o dell'accordo successivo da generare. La funzione di attivazione scelta per ogni rete è la *Softmax*, la quale è una particolare sigmoide definita dall'equazione:

$$y_i = \frac{e^{s_i}}{\sum e^{s_i}}$$

con y_i e s_i i quali rappresentano rispettivamente l'output e la somma pesata dell'i-esimo neurone del layer. La sigmoide è una funzione non lineare nella quale a piccole variazioni dell'input corrispondono simili variazioni in output intorno al range [-3, 3]. Essa comprime il resto degli input verso valori in output pari 0 o ad 1 come rappresentato in figura 5

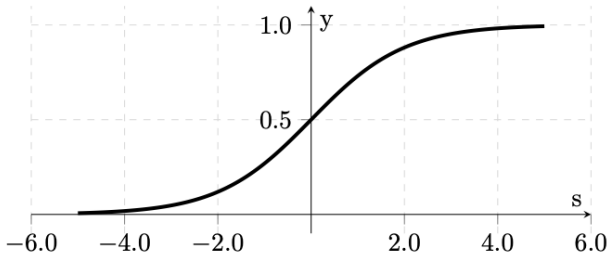


Figura 5: Sigmoide

La softmax è quindi una sigmoide che vincola i neuroni di un layer, normalizzandone il vettore di output in modo che $0 \leq y_i \leq 1 \forall i$ e $\sum y_i = 1$. Il vettore di output può essere dunque visto come una distribuzione discreta di probabilità. Tale funzione di attivazione, grazie a questa particolare caratteristica, può essere utilizzata per classificare dei dati quando le classi sono indipendenti e si vuol ottenere una probabilità che un valore di input appartenga ad una delle possibili classi.

2.0.3 Funzioni di perdita

Il valore di *loss* indica l'errore di previsione computato dalla funzione di perdita sulle uscite del layer di output rispetto ai valori attesi. La funzione di perdita presenta un andamento decrescente poiché essa rappresenta la funzione da minimizzare. La scelta di tale funzione risulta fondamentale poiché è responsabile

dell'aggiornamento dei pesi dell'intera rete e dovrà essere scelta, cosi come la funzione di attivazione, in base al caso applicativo.

La **cross entropy** è una funzione di perdita utilizzata per la classificazione multiclasse ed è quella utilizzata nell'apprendimento delle reti analizzate nel seguente elaborato. Se utilizziamo questa funzione di perdita, formeremo un modello per generare una probabilità sulle classi (C, D, E, F, G, A, B ad esempio) per ogni sequenza di ogni traccia. La funzione di perdita *categorical_ crossentropy* pertanto aumenta quando la probabilità predetta diverge dall'etichetta effettiva.
Essa valuta quanto una distribuzione probabilistica P, per esempio l'insieme degli output di un layer con attivazione softmax, riesca ad approssimare bene una distribuzione nota Y, la quale nel seguente elaborato risulti essere il vettore one-hot encoded della classe attesa.
Questa misura si basa sul concetto di *entropia* H(Y) intesa come il costo in bit per codificare un'informazione Y, ad esempio con 4 possibili classi, ognuna avente il 25% di probabilità, i bit per codificare Y saranno 2. Tale misura singolarmente non è sufficiente poiché non considera quanto la distribuzione P si avvicina o si allontana dalla distribuzione di riferimento Y. Per risolvere a questa situazione è necessario introdurre l'*entropia relativa* H(Y|P) che indica invece quanto P diverge da Y.

$$CrossEntropy = H(Y) + H(Y|P)$$

La formula, nel caso delle reti neurali del seguente elaborato le quali utilizzano attivazione softmax, è la seguente:

$$CrossEntropy = \sum_{i=1}^{N} y_i \cdot \log_2(p_i)$$

con N numero di classi, $p_i \in$ P vettore di output e $y_i \in$ Y classe one-hot encoded attesa.

2.0.4 Ottimizzatori

Nell'algoritmo della discesa del gradiente viene utilizzato l'approccio *batched* nel quale l'errore di predizione viene computato ad ogni epoca come media delle loss derivanti dall'insieme totale dei dati di training. In seguito, l'aggiornamento dei pesi viene effettuato tramite un singolo update. La loss media risultante è indicativa dell'intero dataset, pertanto tale soluzione può portare ad una lenta convergenza oltre a relativi problemi di memoria. L'algoritmo risulta inoltre sensibile a punti critici, quali minimi locali, pertanto l'idea di ottimizzazione è quella di utilizzare dei **mini-batch** data selezionati casualmente dall'insieme di training, i quali essendo più rumorosi, facilitano la discesa del gradiente, evitando punti di stallo.

In tale approccio, i mini-batch data sono costituiti da un numero corrispondente ad una qualche potenza di 2 fino ad arrivare ad un massimo di 256, portando un'epoca ad avere $\frac{N}{batchsize}$ iterazioni, dato N il numero di dati nel training set. La combinazione di discesa gradiente ed ottimizzazione mini batch viene denominata *stochastic gradient-descent* o SGD.

L'ottimizzatore utilizzato nelle reti neurali del generatore di musica simbolica è **Adam**, il quale si basa su un'estensione del SGD chiamata *Momentum*. Quest'ultima accelera la discesa del gradiente nella direzione rilevante riducendo le oscillazioni nelle direzioni irrilevanti considerando il momento, ovvero la derivata seconda della funzione di perdita rispetto ai pesi. Ciò viene fatto aggiungendo al Δ_t nell'istante t, una frazione γ, chiamata *momentum term*, nell'istante Δ_{t-1}

$$\Delta_t = \eta \nabla f(w) + \gamma \cdot \Delta_{t-1}$$

In questo modo se Δ_t e Δ_{t-1} risultano nella stessa direzione, la modifica risulta essere più significativa rispetto al caso in cui siano discordanti.

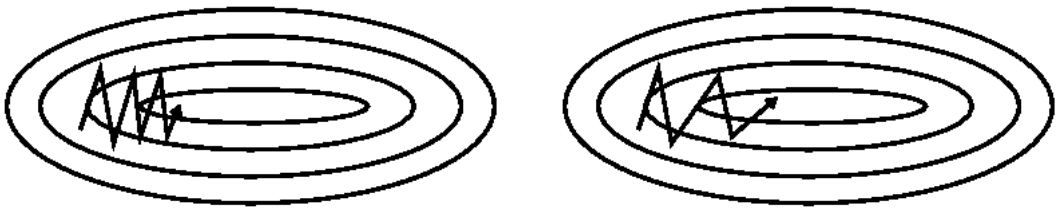


Figura 6: SGD senza momentum e con momentum

Adam è una tecnica adattiva in cui il Δ viene calcolato stimando il momento primo ed il momento secondo del gradiente per ogni peso, rispettivamente come media e varianza non centrata dei gradienti precedentemente applicati ad esso:

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$$

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$$

g_t rappresenta il gradiente attuale, β_1 e β_2 sono dei fattori di decadimento prossimi a 1 (quindi piccoli tassi di decadimento), mentre m_t e v_t, corrispondenti rispettivamente a media e varianza, vengono inizializzati a zero, pertanto i loro valori tendono a rimanere molto piccoli soprattutto nelle prime fasi di training. Per correggere questa tendenza allo zero, all'algoritmo sono state applicate delle correzioni ai momenti di ordine uno e due:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

Il Δ_t viene poi calcolato nel seguente modo:

$$\Delta_t = \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon} \cdot \hat{m}_t$$

con ϵ scalare positivo molto piccolo per evitare divisioni per 0.

2.1 Tipologie di reti neurali utilizzate

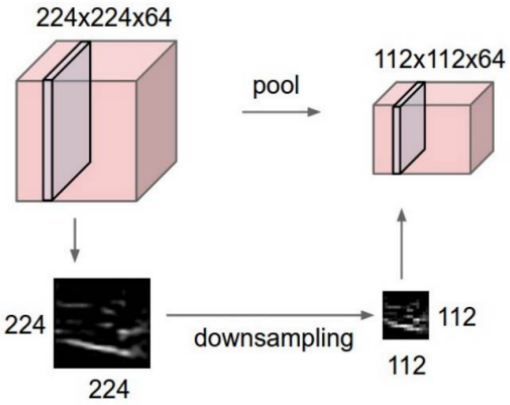
2.1.1 CNN

Le *convolutional neural networks* sono reti neurali in cui ogni strato viene ottenuto dalla convoluzione della matrice dei pesi (kernel) e dello strato precedente. In una rete neurale convoluzionale esistono diversi tipi di layer, ciascuno avente una propria specifica funzione. Quello **convoluzionale** è il principale tipo di layer il quale, durante la forward propagation, si trasla lungo la larghezza e l'altezza dell'input producendo una activation map bidimensionale per quel filtro. La rete avrà come obiettivo quello di apprendere dei filtri che si attivano in presenza di un qualche specifico tipo di feature in una determinata regione spaziale dell'input.

$$y = w * x : y_i = \sum_{h=1}^{k} w_h x_{i-h}$$

Un vantaggio delle reti CNN è che hanno meni parametri rispetto alle MLP, i kernel sono generalmente piccoli, quindi occupano meno memoria e sono più semplici da apprendere. Inoltre presentano il vantaggio della sparsità delle connessioni, ovvero ogni strato è connesso solo al proprio campo recettivo.
Per quanto riguarda l'organizzazione dei neuroni, si identifica con il termine *depth* il numero di neuroni nel layer convoluzionale che sono connessi alla stessa regione locale dell'input. Il raggruppamento di neuroni che osservano tutti la stessa regione locale dell'input, costituisce una depth column. Il parametro *stride* identifica il passo con cui spostare il receptive field dei filtri lungo lo spazio dell'input, pertanto è un metodo molto efficiente per ottenere downsampling.

Un altro tipo di layer indispensabile in una rete neurale convoluzionale è senz'altro il **pooling layer**. Questi layer vengono periodicamente inseriti all'interno di una rete per ridurre la dimensione spaziale, calcolando delle statistiche locali sull'output. Un pooling layer opera su ciascun depth slice del volume di input in maniera indipendente, andando a ridimensionarlo spazialmente.



Lo stadio di pooling più comune nelle reti convoluzionali è il *max pooling*, che restituisce il massimo delle finestre dell'output: questo consente di avere un output invariante rispetto a piccole fluttuazioni dell'input.

Un altro tipo di layer è quello **non lineare**, molto comune nelle CNN. Esso viene utilizzato più volte all'interno della stessa rete, molto spesso dopo ciascun layer convoluzionale ed è lo stadio che applica la funzione di attivazione. Questa tipologia di layer presenta come funzione principale quella di incrementare la proprietà di non linearità della funzione di attivazione, senza andare a modificare i receptive field di un layer convoluzionale. La funzione di attivazione più comune nelle reti convoluzionali è la funzione ReLU: $f(x) = max(0, x)$, la quale permette una fase di training molto più veloce e con prestazioni simili alle altre funzioni di attivazione, altro motivo per cui i ReLU layer sono largamente utilizzati.

2.1.2 LSTM

La *Long Short-Term Memory* rappresenta una particolare rete ricorrente proposta al fine di risolvere il **problema del vanishing ed exploding gradient** che comprometteva l'efficacia delle RNN. Esistono due fattori che influenzano l'entità dei gradienti: i pesi e le funzioni di attivazione (o più precisamente, le loro derivate) che attraversano il gradiente. Se uno di questi fattori è inferiore a 1, i gradienti possono svanire nel tempo; se maggiore di 1, potrebbe verificarsi l'esplosione (Ad esempio, la derivata della *tanh* è < 1 per tutti gli input tranne 0). Nella ricorrenza dell'LSTM, la funzione di attivazione è la funzione di identità con una derivata pari a 1. Quindi, il gradiente che viene retropropagato non svanisce o esplode quando passa, ma rimane costante.
Il principio alla base delle LSTM è la *memory cell*, la quale mantiene lo stato c al di fuori del normale flusso della rete ricorrente. Lo stato c_t indica una somma

tra lo stato precedente c_{t-1} e l'elaborazione dell'input attuale, modulati attraverso dei *gate* di varie tipologie. Essi applicano una funzione di attivazione sigmoide alla concatenazione dell'input x_t e dell'output h_{t-1}, ottenendo un vettore in uscita i cui elementi appartengono al range (0,1). Utilizzare una funzione sigmoide invece di una binary-step function risulta più efficace poiché fornisce il peso del dato da ricordare invece di indicare solamente se ricordarlo o meno. In una LSTM lo stato h_t è diviso in due vettori: h_t e c_t. Il primo è uno stato (o memoria) a breve termine, uguale all'output della cella, mentre il secondo è uno stato a lungo termine. I gate utilizzati nelle LSTM sono:

- **Learn gate**: viene utilizzato per filtrare quali nuove informazioni verranno mantenute nello stato corrente. Esso combina la STM (*short term memory*) con l'input (E), moltiplicando per una matrice (W) ed aggiungendo il bias b. Infine applica la funzione *tanh*.

$$N_t = tanh(W_n[STM_{t-1}, E_t] + b_n)$$

- **Forget gate**: É il gate utilizzato per rimuovere le informazioni a lungo termine non necessarie. Fondamentalmente, la LTM (*long term memory*) viene moltiplicata per un forget factor (f) il quale farà dimenticare alcune delle informazioni a lungo termine. Il forget factor è cosi calcolato:

$$f_t = \sigma(W_f[STM_{t-1}, E_t] + b_f)$$

  Viene calcolato prendendo la memoria a breve termine e l'input, moltiplicandoli per alcuni pesi e ponendo il risultato in una funzione sigmoide. Tale funzione (f) verrà moltiplicata poi per la LTM ottenendo il risultato desiderato.

- **Remember gate**: É il gate che prende le informazioni dal forget gate e le somma alle informazioni del learn gate, per calcolare la nuova memoria a lungo termine.

- **Output gate**: Viene utilizzato per filtrare le informazioni in uscita, prendendo la LTM dal forget gate, mentre la STM e l'input E dal learn gate e li utilizza per creare una nuova memoria a breve termine o un output. Nel dettaglio, prende l'uscita del forget gate e gli applica una funzione di attivazione *tanh*.

$$U_t = tanh(W_u LTM_{t-1} f_t + b_u)$$

Successivamente prende l'output del learn gate e gli applica una funzione sigmoide:

$$V_t = \sigma(W_v[STM_{t-1}, E_t] + b_v)$$

Infine il gate moltiplica V x U per ottenere la nuova short term memory.

In figura 7 si può osservare la struttura della LSTM, dove le operazioni "×" e "+" indicano prodotti e somme elemento per elemento.

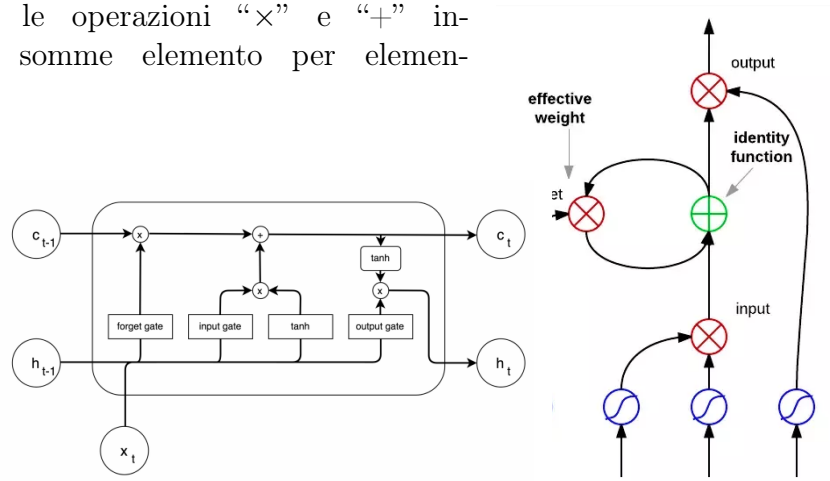


Figura 7: Schema di una LSTM [2]

Una variante della LSTM utilizzata nell'elaborato è la *Gated Recurrent Unit* (GRU), una versione semplificata di LSTM, che cerca di mantenerne i vantaggi, riducendo parametri e complessità. Le principali semplificazioni sono:

- Un solo stato di memoria h_t
- Combina il forget gate ed il gate di input in un unico *update gate*.

2.1.3 GAN

Le *Generative Adversarial Networks*, o in breve GAN, rappresentano un approccio alla modellazione generativa che utilizza metodi di deep learning. La modellazione generativa è un'attività di apprendimento non supervisionata del machine learning che comporta la scoperta e l'apprendimento

automatico di pattern nei dati di input in modo tale che il modello possa essere utilizzato per generare o produrre nuovi esempi che plausibilmente avrebbero potuto essere tratti dal set di dati originale.

Le GAN sono un modo intelligente di formare un modello generativo inquadrando il problema come problema di apprendimento supervisionato con due sotto modelli: il modello *generatore* che viene formato per generare nuovi esempi e il modello *discriminatore* che cerca di classificare gli esempi come reali (ovvero provenienti dal dominio) o falso (generati). I due modelli vengono addestrati insieme fino a quando il modello discriminatore viene ingannato circa la metà del tempo, il che significa che il modello generatore sta generando esempi plausibili.

Il **modello del generatore** prende come input un vettore casuale a lunghezza fissa, estratto casualmente da una distribuzione gaussiana, e genera un campione nel dominio. Dopo l'allenamento, i punti in questo spazio vettoriale multidimensionale corrisponderanno ai punti nel dominio del problema in esame, formando una rappresentazione compressa della distribuzione dei dati. Questo spazio vettoriale viene definito *spazio latente* o spazio vettoriale composto da variabili latenti, le quali sono quelle importanti per un dominio ma non osservabili direttamente. Spesso ci riferiamo a variabili latenti, o uno spazio latente, come una proiezione o compressione di una distribuzione di dati. Cioè, uno spazio latente fornisce una compressione o concetti di alto livello dei dati grezzi osservati, come la distribuzione dei dati di input.
Dopo l'allenamento, il modello del generatore viene mantenuto e utilizzato per generare nuovi campioni.

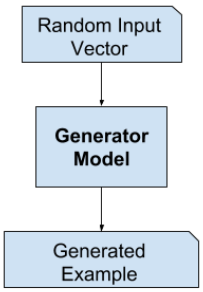


Figura 8: Esempio di GAN Generator Model

Il **modello discriminatore** prende un esempio dal dominio come input (reale o generato) e predice un'etichetta binaria, indicando se il valore è reale o generato. Gli esempi veri derivano dal set di dati di training mentre quelli generati vengono emessi dal modello del generatore. Il discriminatore è un normale modello di

classificazione. Dopo il processo di addestramento, il modello discriminatore viene scartato poiché verrà utilizzato solamente il generatore.

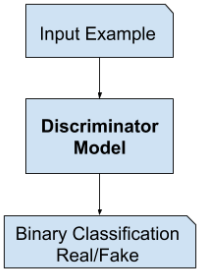


Figura 9: Esempio di GAN Discriminator Model

Capitolo 3

Stato dell'arte su AI applicata alla musica

3.1 Reti neurali

Uno dei primi approcci alla modellazione sequenziale di musica risale al 1950 e vede coinvolte le *Markov chains*, le quali furono le prime strutture a rendere possibile la generazione di composizioni musicali da computer. Per la computazione musicale, le reti neurali artificiali (ANN), così come altri sistemi che implicano l'apprendimento automatico, sono in grado di apprendere i modelli e le caratteristiche presenti nelle melodie dell'insieme di training e di generalizzarle per comporre nuove melodie. Per la ricerca di pattern all'interno di un brano musicale è presente un framework [4], basato sul *Variable Markov Oracle* (VMO) che è in grado di localizzare suffissi ripetuti all'interno di una serie temporale. Il framework proposto è in grado di gestire rappresentazioni sia simboliche che audio. Il core dell'algoritmo della ricerca di pattern è VMO, una struttura dati derivante dalle strutture Factor Oracle (FO) [1] e Audio Oracle (AO) [2].

Tra le diverse ANN, le *reti neurali ricorrenti* (RNN) sono particolarmente prese in considerazione per la manipolazione digitale della musica, poiché considerano unità di feedback e operatori di delay, che permettono di incorporare aspetti di non linearità e dinamicità nel modello. Ciò rende le RNN meccanismi particolarmente interessanti nell'analisi delle serie temporali. La rete LSTM è un'architettura alternativa alla rete neurale ricorrente, ispirata ai sistemi di memoria umana. La

[1] La struttura FO è una variante di una struttura dati *suffix tree* ed è ideata per recuperare pattern da una sequenza simbolica

[2] AO è l'estensione di una FO ed è in grado di indicizzare sub clip ripetute di un segnale campionato in momenti discreti

motivazione principale nell'utilizzo di questa tipologia di rete neutale è quella di risolvere il problema del gradient vanishing/exploding. Ciò consente alla rete di apprendere dati importanti e mantenerli in memoria senza far si che tali informazioni si disperdano con l'avanzare delle epoche.

Gli studi sulla composizione musicale trovati in letteratura, i quali utilizzano LSTM, sono iniziati nel 2002 con l'articolo di Eck e Schmidhuber [5]. Gli autori hanno usato LSTM per imparare le forme musicali del blues. In una seconda fase, la rete è stata utilizzata per comporre nuove melodie sullo stile blues, senza perdere le strutture rilevanti nel lungo periodo. Essi hanno dimostrato che una LSTM può catturare sia la struttura locale della melodia, sia la struttura a lungo termine di uno stile musicale. Una variazione importante è l'architettura bidirezionale. DeepBach [6] ha introdotto una RNN bidirezionale innovativa per l'armonizzazione della musica, la quale è stata utilizzata in [7] per vincolare la linea melodica a seguire gli accordi dell'intero brano musicale.
Gli autori di quest'ultimo lavoro hanno confrontato due modelli generativi sequenziali, quali LSTM e WaveNet. Questo è stato il primo studio sull'applicazione di WaveNet alla generazione di musica simbolica, nonché il primo confronto sistematico tra CNN temporale e RNN per la generazione di musica. I risultati hanno mostrato come due fattori hanno largamente migliorato le performance del modello: uno stack di *dilated convolution layers* per codificare la dipendenza strutturale della sequenza melodica e l'incorporazione della progressione di accordi come vincolo della struttura globale. WaveNet [10] è stata introdotta per la prima volta da Google Deepmind come modello generativo per l'audio non elaborato. La maggior parte delle pubblicazioni si concentra su due aspetti: migliorare la velocità di WaveNet e applicare WaveNet ad applicazioni correlate all'audio. I risultati più convincenti sono stati ottenuti aggiungendo condizioni come input aggiuntivo. Il modello non condizionato dagli accordi con il vettore della melodia di input m e la funzione di attivazione nello strato di dilatazione k è:

$$z = tanh(W_{f,k} * m) \odot \sigma(W_{g,k} * m)$$

dove $*$ rappresenta un operatore di convoluzione dilatato, $W_{f,k}$ e $W_{g,k}$ sono i parametri apprendibili nello strato di convoluzione ed $\odot$ è l' operatore element-wise moltiplication. A questo è stato aggiunto il vettore di accordi c come condizionamento locale, ottenendo il modello descritto in figura 10:

$$z = tanh(W_{f,k} * m + V_{f,k} * v) \odot \sigma(W_{g,k} * m + V_{g,k} * c)$$

Figura 10: Modello Wavenet per generazione musica simbolica con condizionamento di accordi

dove il secondo $*$ in entrambe le parentesi identifica un layer convoluzionale $1 * 1$ con $V_{f,k}$ e $V_{g,k}$ come parametri di apprendimento. La condizione di accordi pertanto risulta un vettore di egual lunghezza al vettore di melodia. Il condizionamento degli accordi guida il processo di generazione della musica per includere più struttura musicale, il che migliora la qualità della musica generata.

Un altro modello che fa uso di RNN, considerato tra i più famosi esempi di generazione musicale di dominio simbolico da parte di reti neurali, è il *MelodyRNN* [8] proposto dal team Magenta di Google. Questa configurazione funge da base per la generazione di melodie con un modello LSTM, utilizzando one-hot encoding. Oltre alla versione standard sono presenti anche due varianti che mirano ad apprendere strutture a lungo termine: LookbackRNN e AttentionRNN. La *LookbackRNN* introduce input personalizzati che consentono al modello di riconoscere più facilmente i pattern che si verificano tra 1 e 2 battute ed etichette personalizzate le quali semplificano la ripetizione di pattern di note da parte del modello senza doverli archiviare nello stato della cella LSTM. La *AttentionRNN* utilizza l'attenzione, ovvero vengono osservati sempre gli output degli ultimi n passaggi quando viene generato l'output per lo step corrente. Il modo in cui vengono considerati questi passaggi è con un meccanismo di attenzione.

Song from PI [9] è un modello gerarchico di RNN che utilizza uno stack di livelli ricorrenti per generare non solo la melodia ma anche la batteria e gli accordi, pensato per la generazione di canzoni pop multitraccia. Questo modello dimostra la capacità degli RNN di generare più sequenze contemporaneamente. Tuttavia, richiede la conoscenza preliminare della scala musicale e di alcuni dettagli della melodia, il che non è necessario in molti altri modelli.

3.2 Architettura Encoder Decoder

Il modello WaveNet [10], proposto da DeepMind, mostra come le reti neurali convoluzionali (CNN) possano generare forme d'onda musicali realistiche nel dominio audio. Per questo motivo nel progetto MidiNet [11] è stata utilizzata una rete CNN per generare la melodia non in modo sequenziale, bensì una battuta (misura) dopo l'altra. Oltre al generatore, è stato aggiunto all'architettura un discriminatore per apprendere le distribuzioni delle melodie, rendendola una generative adversarial network (GAN). La generazione di misura in misura ha permesso di impiegare convoluzioni su una matrice 2D, nella quale viene identificata la presenza o meno di note sui diversi intervalli temporali. L'obiettivo del generatore è trasformare

il rumore casuale in ingresso in una rappresentazione tale da sembrare realistica. Questa trasformazione è raggiunta tramite uno speciale operatore convolutivo chiamato *transposed convolution* [12], il quale permette ad esempio di mappare uno spazio a 4-D in uno spazio a 16-D, mantenendo il modello di connettività della convoluzione.

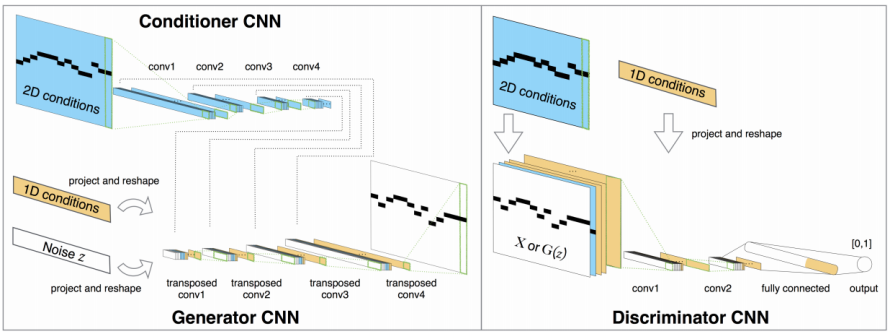


Figura 11: Architettura del modello MidiNet [11]

Questo tipologia di GAN da sola non tiene conto delle dipendenze temporali tra le diverse barre. Per affrontare questo problema, è stato proposto un nuovo meccanismo condizionale per utilizzare la musica delle barre precedenti in modo tale che vadano a influenzare la generazione della battuta corrente. Questo è stato raggiunto con l'apprendimento di un altro modello CNN, chiamato *conditioner CNN*, il quale incorpora le informazioni delle battute precedenti nei layers intermedi del generatore. In questo modo, tale modello può "guardare indietro" senza utilizzare unità ricorrenti come nelle RNN ed è in grado anch'esso di generare un numero arbitrario di battute.
Poiché viene utilizzato rumore casuale in ingresso, questo modello può generare melodie da zero, vale a dire senza altre informazioni precedenti. Tuttavia, grazie al conditioner CNN, il modello ha la capacità di sfruttare qualsiasi conoscenza precedente disponibile e che può essere rappresentata come matrice. Ad esempio, il modello può generare musica seguendo una progressione di accordi o seguendo alcune note iniziali e può essere esteso per generare diversi tipi di musica, utilizzando condizioni diverse di partenza.

Il successo nella generazione del linguaggio naturale e della musica monofonica potrebbe però non essere facilmente generalizzabile alla generazione di musica polifonica. Di conseguenza, la maggior parte delle proposte precedenti ha scelto di semplificare la generazione di musica simbolica in alcuni modi per rendere gestibile il problema, tra i quali: generare solo musica monofonica a traccia singola, introdurre un ordinamento cronologico di note per la musica polifonica, generare musica polifonica come combinazione di diverse melodie monofoniche, ecc. Un modello

che ha voluto evitare il più possibile queste semplificazioni è il MuseGAN [13], il quale propone la generazione di musica polifonica multi traccia con struttura armonica, ritmica e temporale. Per incorporare un *modello temporale*, sono stati proposti due approcci per diversi scenari: uno genera musica da zero (cioè senza input umani) mentre l'altro impara a seguire la struttura temporale sottostante di una traccia data a priori dall'umano. Per gestire le *interazioni tra le tracce* vengono proposti tre metodi: uno genera tracce indipendentemente dai generatori privati (uno per ciascuno), un altro genera tutte le tracce congiuntamente con un solo generatore, mentre un altro genera ogni traccia dal suo generatore privato con ulteriori input condivisi tra le tracce, che dovrebbe guidarle in modo che siano collettivamente armoniose e coordinate. Così come nel modello MidiNet, anche in MuseGAN vengono considerate le battute anziché le note come unità compositiva di base e la generazione avviene una misura dopo l'altra. Essa si serve di reti neurali convoluzionali trasposte (CNNs), utilizzate notevolmente nelle GAN le quali richiedono di predire valori per ogni pixel per poter aumentare la dimensionalità dell'input.

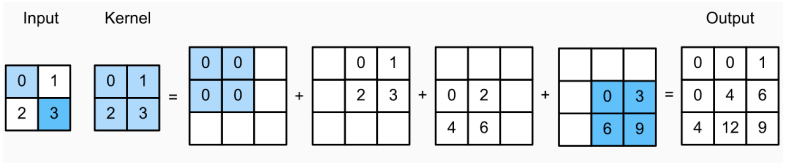


Figura 12: Esempio di un transposed convolution layer con un kernel 2×2

Un ulteriore progetto che ha ottenuto risultati convincenti in molte attività di generazione che richiedono il mantenimento della coerenza a lungo termine è il *Transformer* [18]. Esso è il primo modello che si basa interamente sulla self-attention per calcolare le rappresentazioni dei suoi input ed output senza usare RNN in sequenza o convoluzione. Un modulo di self-attention accetta n input e restituisce n output. Il meccanismo di self-attention consente agli input di interagire tra loro ("self") e scoprire a chi dovrebbero prestare maggiore attenzione ("attention"). I risultati sono aggregati di queste interazioni e punteggi di attenzione. Questo modello si affida a segnali di temporizzazione assoluti e quindi fa fatica a tenere traccia della regolarità che si basa su distanze relative, ordinamento di eventi e periodicità. Un'estensione di questo progetto è stata proposta dal team Google Brain ed è il *Music Transformer* [17]. Esso utilizza l'*attenzione relativa* che modula esplicitamente l'attenzione in base alla distanza tra due token, pertanto il modello è in grado di concentrarsi maggiormente sulle funzionalità relazionali. La relative self-attention consente inoltre al modello di generalizzare oltre la lunghezza degli esempi di training, cosa impossibile con il modello Transformer originale,

inoltre utilizzando questo nuovo algoritmo si riduce drasticamente la quantità di memoria principale utilizzata per le lunghe sequenze.

3.3 Spazi latenti

I recenti progressi nell'apprendimento automatico hanno permesso di formare modelli generativi in grado di rappresentare e generare con precisione molti tipi diversi di oggetti come immagini, sketches e brani musicali per citarne qualcuno. Alcuni di questi modelli apprendono uno *spazio latente*: una rappresentazione di dimensione inferiore che può essere mappata da e verso lo spazio oggetti. Un grande vantaggio dei modelli spaziali latenti è che molte operazioni che sarebbero difficili da eseguire nello spazio degli oggetti, in modo semanticamente significativo, diventano semplici aritmetiche nello spazio latente. I modelli spaziali latenti sono già stati addestrati per diversi domini musicali. Tali modelli sono anche frequentemente utilizzati per raccomandazioni musicali [14], dove sia il "gusto" dell'utente che lo "stile" della canzone sono descritti in termini di vettori latenti.
In questo articolo [15], è stato presentato un modello spaziale latente di misure musicali polifoniche multi strumento, nel quale viene utilizzata la rappresentazione simbolica della musica. Questo modello consente di eseguire una serie di operazioni, tra le quali: campionare una misura dalla distribuzione a priori per generare nuova musica da zero, applicare trasformazioni di alcuni attributi a una misura esistente, ad esempio aumentare la densità delle note.
Il modello di spazio latente può anche essere aumentato con ulteriori variabili. Il condizionamento degli accordi consente di eseguire le operazioni descritte precedentemente, mantenendo costanti gli accordi oppure cambiando gli accordi ma mantenendo costante la trama musicale.
Anche se questo modello rappresenta solo misure individuali ed è quindi incapace di generare da solo una struttura a lungo termine, combinare lo spazio latente con il condizionamento degli accordi rende abbastanza facile generare musica convincente con dipendenze a lungo termine. Questo articolo pertanto ha portato i seguenti contributi allo stato dell'arte:

- Un'estensione del modello MusicVAE [16] per gestire fino a 8 tracce contemporaneamente.
- Una nuova rappresentazione di traccia basata su eventi che gestisce la polifonia, la dinamica e la selezione dello strumento.
- Introduzione del condizionamento degli accordi a un modello spaziale latente, in modo che accordi ed arrangiamento possano essere controllati in modo indipendente.

L'architettura MusicVAE [16], è in grado di apprendere uno spazio latente di sequenze musicali utilizzando un nuovo decoder gerarchico che gli consente di modellare la struttura a lungo termine e le sequenze multi strumento. Tuttavia, questo lavoro applica precisi vincoli alle sequenze per raggiungere il suo obiettivo, tra i quali: le tracce non ritmiche sono limitate a sequenze monofoniche, tutte le tracce sono rappresentate con una sola velocity e quantizzate a livello di sedicesimi di nota. Inoltre MusicVAE è in grado di modellare tre grandi classi di strumenti: melodia, basso e batteria, ignorando l'identità specifica di ogni strumento (ad es. Chitarra e pianoforte sono entrambi considerati strumenti "melodia").

Capitolo 4

Dataset, preprocessing, features

4.1 Dataset

Per analizzare e quindi generare la linea armonica, melodica e l'unione delle due è necessario partire da un set di dati che presenti solo questi due canali, evitando di dover elaborare il file midi per rimuovere suoni aggiuntivi come ad esempio la batteria. Questo set di dati è stato ottenuto dal sito **HookTheory** [1], il quale è una piattaforma utile per imparare la teoria musicale nel quale vengono mostrate melodia ed armonia in parallelo ed è possibile visualizzare la nota o l'accordo riprodotto in diretta. Ogni brano musicale non viene descritto da un unico file XML ma viene rappresentato da molteplici file, ognuno rappresentante di una delle diverse sezioni della stesura, quali: intro, strofa, inciso, ritornello, coda, special, ponte.

4.2 Data pre-processing

4.2.1 Conversione da XML a MIDI

La conversione da XML a file MIDI attraversa due step intermedi: nel primo il file XML viene convertito in un dizionario contenente i metadati del brano musicale (BPM, durata e tonalità), il numero di misure e, sia per la melodia che per l'armonia, una lista di oggetti json i quali rappresentano gli eventi midi. Il secondo step itera su ogni oggetto presente nella lista melodica ed armonica. Nel primo caso è presente un parser di nota il quale calcola il valore del pitch midi e riporta le indicazioni sulla durata e se quel dato evento identifica una pausa. L'accordo invece viene identificato dai pitch delle note che lo rappresentano, pertanto viene applicato un parser di accordo il quale calcola il nome nella notazione anglosassone.

[1] `https://hooktheory.com/theorytab`

Figura 13: Data pre-processing

4.2.2 Rappresentazione matriciale

Il file midi di ogni brano musicale viene convertito successivamente in una matrice numpy prendendo come unità di misura un intervallo da 1/16. Ogni matrice avrà dimensioni T×130 nel caso dell'esclusiva generazione melodica, in cui le 130 colonne rappresentano i 128 pitch midi, la penultima colonna identificherà se è presente la pausa mentre l'ultima rappresenta se viene sostenuta la nota precedente. Con T si indica il numero di time steps del brano, quindi il numero di step da 1/16. Nel caso della generazione esclusivamente armonica, verranno create matrici di dimensioni T×26 in cui le prime 12 colonne rappresentano i principali accordi maggiori mentre le seguenti 12 colonne rappresentano gli accordi minori. Come per la generazione melodica le ultime due colonne identificano la pausa e l'holding mode. Un accordo che non è presente tra i 24 indicati precedentemente viene abbinato ad uno degli accordi più comuni col quale condivide il maggior numero di note, per esempio *C- major7* (C7) verrà corrisposto al *C-major*.

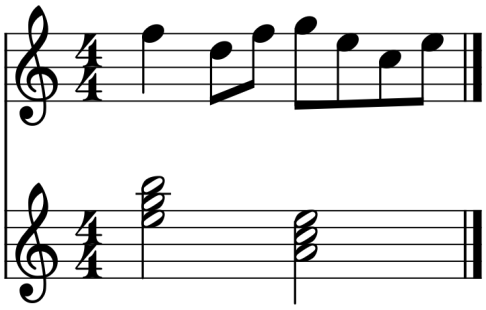


Figura 14: Esempio di spartito

La rappresentazione matriciale di ogni brano deve avere una durata T comune per poter esser data come input alle reti neurali, pertanto è necessario effettuare un processo di suddivisione della matrice originale. Poiché il tempo di ogni brano nel dataset è 4/4, è stata scelta come dimensione di ogni matrice una potenza di

$$
\mathbf{M} = \begin{array}{c} \\ \\ C6 \\ \\ G5 \\ F5 \\ E5 \\ D5 \\ C5 \\ \\ rest \\ holding \\ \\ Gmajor \\ Cmajor \\ \\ \end{array}
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
- & - & - & - & - & - & - & - \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

Figura 15: Esempio di rappresentazione matriciale del brano in figura 14

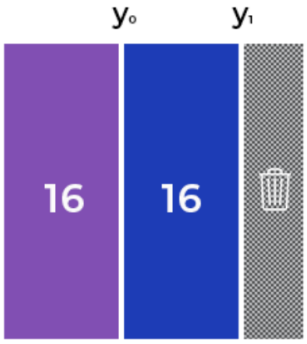


Figura 16: Suddivisione con shift di T+1

4, in questo caso 2 per ottenere T=16, quindi una battuta.

Le tre tecniche analizzate sono: data n la dimensione della matrice originale, suddividerla in $n/(T+1)$ parti escludendo la parte restante, es. se il brano è composto da 32 step da 1/16 si suddivide la matrice in 32/(16+1) = 1,88 pertanto si andrebbe a perdere la metà del brano. Per utilizzare maggiormente il dataset è possibile utilizzare T=15, che porterebbe ad un miglior partizionamento della matrice originale, con il trade off che durante la generazione non andrebbe a considerare il primo sedicesimo della battuta.

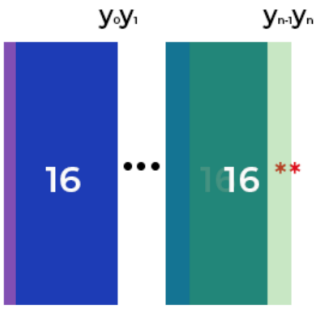


Figura 17: Suddivisione con stop token e shift di 1

Una seconda tecnica proviene dal NLP[2] e consiste nel ridimensionare ogni matrice con la dimensione del brano massima presente nel dataset, inserendo uno stop token nella posizione del sedicesimo successivo all'ultimo presente nella matrice originale. Spostando la finestra di 1/16 si ha il vantaggio di utilizzare l'intero

[2]Natural Language Processing: comprensione ed elaborazione del linguaggio naturale

dataset senza sprecare dati di training. Un vantaggio di questa tecnica è la possibilità di avere all'interno del training set anche delle indicazioni sulla fine del brano poiché si otterranno delle matrici aventi sia pitch, sia stop token.

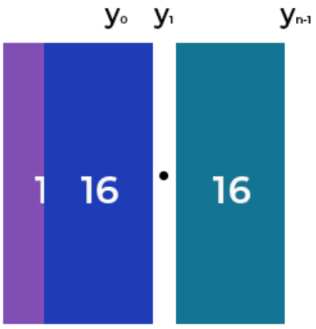


Figura 18: Shift di 1 battuta

Ultima tecnica analizzata consiste nel spostare la finestra T di una battuta, a differenza della seconda opzione che sposta T di 1/16, determinando un minor numero di elementi di training senza sprecare parti della matrice del brano originale.

4.3 Estrazione delle features

Una volta ottenuta la matrice del dataset, è stata eseguita la codifica **one-hot encoding** della rappresentazione matriciale di ciascun brano musicale. Questo processo è necessario per migliorare le prestazioni degli algoritmi di ML ed evitare di assegnare pesi elevati alle note con pitch maggiore.

Capitolo 5

Generazione di musica simbolica

5.1 Panoramica del progetto

Nel sistema, dopo una prima pre-elaborazione dei dati di input, vengono definite le architetture delle reti neurali prese in esame ed il tipo di generazione dell'output finale: solo melodia, solo armonia o entrambe.

Una volta che il modello è stato creato per ciascuna rete, identificando i migliori parametri tramite una fase di tuning, inizia l'addestramento in cui ogni rete itera per un numero predeterminato di epoche.

I vari modelli vengono salvati compressi su disco per risparmiare tempo di calcolo e vengono generati i grafici che mostrano l'andamento dell'accuratezza e della perdita per ogni epoca.

Oltre a questi, vengono generate delle heatmaps che mostrano, per ogni layer, come cambiano i pesi interni della rete neurale.

Infine, viene gestita la fase di generazione del file .mid di output, in cui tramite la predizione del modello viene identificato il sedicesimo di nota successivo.

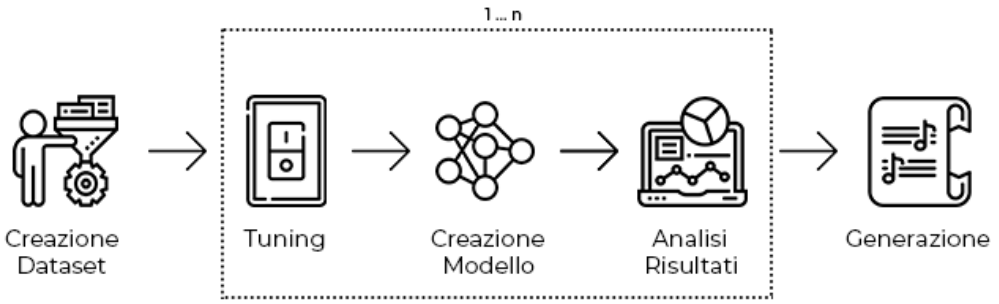


Figura 19: Panoramica progetto di generazione di musica simbolica

5.2 Implementazione

Ottenuta la matrice dell'intero dataset tramite la fase di pre-processing descritta nel Capitolo 3, si procede alla selezione dei campi che guideranno la generazione. Il valore *Generation type* specificherà quale rappresentazione matriciale considerare, se quella con 130 colonne per la melodia, con 26 colonne per l'armonia oppure la rappresentazione melodica ed armonica con 156 colonne. La matrice viene recuperata dal disco nel caso fosse già stata elaborata, altrimenti viene attivata la fase di conversione dei file MIDI di tutti i brani musicali in matrici numpy. Ottenuta la matrice rappresentante l'intero dataset, viene poi suddivisa in training e test set tramite la tecnica di *holdout*, nella quale viene selezionata casualmente una frazione α del dataset che verrà usata come training set (80%), mentre il restante 1 - α andrà a far parte del test set (20%).

Successivamente si itera la lista di modelli di apprendimento da generare e per ognuno di essi si inizia la fase di training nel caso il modello non fosse già salvato in memoria. In questo caso verrebbe restituito il file HDF5 (Hierarchical Data Format version 5), un formato open source che permette di mantenere grosse quantità di dati complessi ed eterogenei.

L'infrastruttura hardware utilizzata è composta da due GPU TITAN V da 12GB ognuna e 20 core, di cui 10 reali e 10 virtuali, per un totale di 62 GB di memoria. Il framework utilizzato per la fase di training è Tensorflow-Gpu [1], il quale permette di eseguire l'addestramento su GPU NVIDIA, e Keras [2] per la creazione dei modelli di apprendimento.

```
def samples(x_source, y_source, size):
    while True:
        for i in range(0, x_source.shape[0], size):
            j = i + size

            if j > x_source.shape[0]:
                j = x_source.shape[0]

            yield x_source[i:j],

history = model.fit_generator(samples(X_train, y_train, batch_size),
steps_per_epoch, epochs, callbacks)
```

[1] https://www.tensorflow.org/install/gpu
[2] https://keras.io/

Il metodo di Keras *fit_generator* permette di gestire dataset del mondo reale, quindi troppo gradi per essere caricati interamente in memoria, sfruttando l'ottimizzazione della lettura in batch dei generatori. Internamente Keras utilizza il processo seguente quando esegue l'addestramento del modello:

- La funzione generatore *samples* restituisce un insieme di valori di dimensione *batch_size* alla funzione *fit_generator*
- La funzione *fit_generator* prende in ingresso il batch di dati, esegue la backpropagation ed aggiorna i valori dei pesi nel modello.
- Tale processo viene ripetuto fino al raggiungimento del numero di epoche stabilito.

Il *batch_size* utilizzato è 512, mentre l'apprendimento itera per 100 epoche.

Al processo di training sono state associate molteplici funzionalità di callback che vengono eseguite nel ciclo di vita di un'epoca, tra le quali: l'*Early Stopping* consente di specificare la misura delle prestazione da monitorare, ed una volta verificatosi l'evento, interromperà il processo di training. Essa è stata impostata per verificare che il valore di loss scenda gradualmente e non aumenti, né rimanga ferma all'interno di un range di valori, poiché in questi casi andrebbe a bloccare l'avanzamento dell'apprendimento.
Altra funzione di callback è la *ModelCheckpoint*, la quale monitora anch'essa l'andamento del valore di loss, salvando il modello migliore osservato durante il training, all'interno di un file h5.

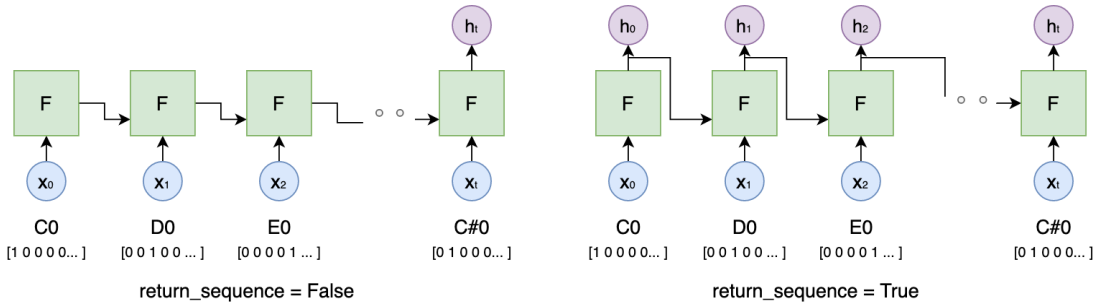
I modelli, i quali verranno sottoposti successivamente alla fase di training, sono sequenziali, ovvero sono costituiti da uno stack di layer di varia natura. Ogni modello del generatore di musica simbolica viene configurato con la *categorical_crossentropy* come funzione di loss, anche chiamata *Softmax Loss* descritta in 2.0.3. Come ottimizzatore della fase di compilazione viene adottata la tecnica adam 2.0.4. È un algoritmo di ottimizzazione che può essere utilizzato al posto della classica procedura di discesa gradiente stocastica per aggiornare i pesi della rete neurale.

5.2.1 Long short-term memory

L'architettura del modello LSTM è cosi composta:

Il primo livello è di tipologia LSTM con dimensioni di input pari a quelle della matrice di training, ovvero: numero di *items* nella matrice di training, numero di righe di ogni matrice *item*, numero di colonne di ogni matrice *item*. Al layer

LSTM viene impostato il parametro *return_sequences* a True per restituire un hidden state output [3] per ogni input time step per la singola cella LSTM nel layer, altrimenti la cella fornirà semplicemente l'output dall'ultima fase temporale. È necessario impostare questo valore quando si crea uno stack di layer LSTM in modo tale che i layer successivi abbiano un input di sequenza multidimensionale.



Il modello LSTM originale è costituito da un singolo livello LSTM hidden seguito da un livello di output feedforward standard. Stacked LSTM è un'estensione di questo modello che ha più livelli LSTM nascosti in cui ogni layer contiene più celle di memoria. La sovrapposizione dei livelli nascosti LSTM rende il modello più profondo. È la profondità delle reti neurali che è generalmente attribuita al successo dell'approccio su una vasta gamma di difficili problemi di predizione [19]. Una DNN (deep neural network) può essere vista come una pipeline di elaborazione, in cui ogni livello risolve una parte dell'attività prima di passarla alla successiva, fino a quando l'ultimo strato fornisce l'output. Dato che le reti LSTM operano su dati di sequenza, l'aggiunta di layer aggiunge livelli di astrazione delle osservazioni di input nel tempo.

Il secondo livello è un Dropout layer, il quale rappresenta una tecnica di regolarizzazione che previene l'overfitting, rimuovendo probabilisticamente gli input in un layer. Ha l'effetto di simulare un gran numero di reti con una struttura molto diversa e, a sua volta, rende i nodi della rete generalmente più robusti agli input. Il layer Dropout viene aggiunto ad un modello tra layer esistenti e si applica agli output del layer precedente che vengono inviati al layer successivo.

Il terzo layer è sempre di tipologia LSTM, con la differenza che il parametro *return_sequences* viene settato a False. Esso consente una rappresentazione delle features più complessa, il che significa una migliore generalizzazione e quindi una migliore classificazione. Dopo un ulteriore layer di dropout viene inserito uno strato fully connected che come output avrà un numero di neuroni pari al numero

[3] L'output di una cella LSTM o di un layer di celle LSTM è chiamato *hidden state*

di classi definite dalla tipologia del modello generativo. Infine come funzione di attivazione è stata utilizzata Softmax, la quale rende il vettore di output una distribuzione discreta di probabilità.

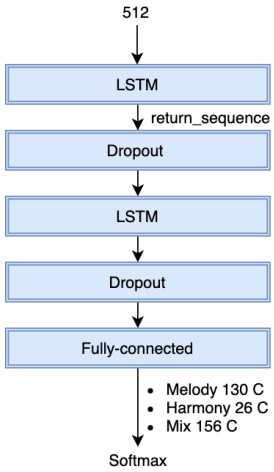


Figura 20: Architettura rete LSTM

5.2.2 Bidirectional Long short-term memory

Le LSTM bidirezionali sono un'estensione delle LSTM tradizionali che possono migliorare le prestazioni del modello in caso di problemi di classificazione delle sequenze. Nei problemi in cui sono disponibili tutti i timestep della sequenza di input, le LSTM bidirezionali addestrano due LSTM anziché una. La prima sulla sequenza di input così com'è e la seconda su una copia invertita. Ciò può fornire un contesto aggiuntivo alla rete e consentire un apprendimento più rapido e persino più completo del problema. L'idea è di dividere i neuroni di stato di una normale RNN in una parte responsabile della direzione temporale positiva (forward states) e una parte per la direzione temporale negativa (backward states).

Il primo layer, *CuDNNLSTM*, è un livello LSTM che utilizza il toolkit Cuda di NVIDIA per funzionare esclusivamente tramite GPU, consentendogli di essere più veloce della normale LSTM. Questo layer rappresenta la chiave del modello per imparare come prevedere le note in una determinata sequenza.
Successivamente per capire perché un certo accordo è stato usato in un determinato luogo e tempo, è necessario contestualizzarlo esaminando ciò che è venuto prima e dopo quel particolare accordo. Per questo motivo vengono utilizzati due modelli *Bidirectional-LSTM*, i quali simulano questo approccio per imparare dall'ordinamento e mettere insieme le note richieste. Le LSTM bidirezionali sono supportate

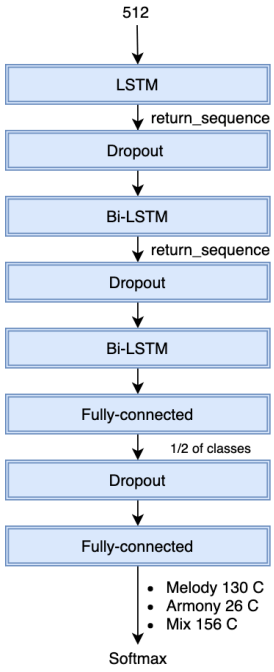


Figura 21: Architettura rete Bidirectional-LSTM

in Keras tramite il layer wrapper *Bidirectional*, il quale accetta come argomento un recurrent layer. Consente inoltre di specificare la modalità di unione, ovvero come combinare gli output forward e backward prima di passare al livello successivo. L'opzioni scelta, la quale è quella di default, è "concat" in cui gli output vengono concatenati insieme fornendo il doppio del numero di output al livello successivo.

5.2.3 Gated Recurrent Unit

L'architettura GRU è identica a quella definita in 5.2.1 con la sola differenza che cambia il primo livello ricorrente, il quale non è più di tipologia LSTM bensì Gated Recurrent Unit. In Keras esistono due varianti: la versione predefinita si basa sul paper 1406.1078v3 [4] ed ha il reset gate applicato allo hidden state prima della moltiplicazione della matrice. L'altra variante è basata sulla versione originale del paper ""Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation"" 1406.1078v1 [5] ed ha l'ordine invertito. Quest'ultima versione è quella utilizzata nell'architettura del modello ed è quella utilizzata dal layer CuDNNGRU (il quale lavora esclusivamente sulla GPU).

[4] `https://arxiv.org/pdf/1406.1078v3.pdf`
[5] `https://arxiv.org/pdf/1406.1078v1.pdf`

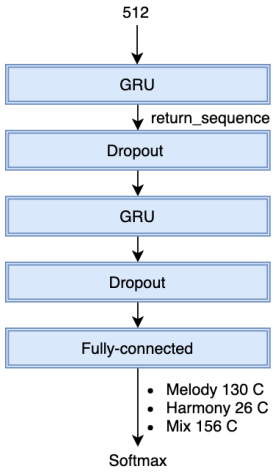


Figura 22: Architettura rete GRU

5.2.4 Convolutional Neural Network

Una CNN monodimensionale è un modello che ha uno strato nascosto convoluzionale che opera su una sequenza 1-D e solitamente è applicata a problemi reali in cui il tempo è un elemento importante.

```
n_steps = X_train.shape[1]
n_features = X_train.shape[2]
model.add(Conv1D(filters=filters,
                 kernel_size=kernel_size,
                 activation='relu',
                 input_shape=(n_steps, n_features)))
```

Listing 5.1: Definizione layer convoluzionale 1-D

Il primo layer 1-D CNN riceve come input le matrici di ogni brano musicale e, lavorando con una serie univariata, il numero di features è uno per ogni variabile. Le serie temporali "univariate" sono insiemi di dati costituiti da una singola serie di osservazioni con un ordinamento temporale ed è necessario un modello per imparare dalle serie di osservazioni passate per prevedere il valore successivo nella sequenza.

Il parametro *kernel_size* definisce la dimensione della sliding window, la quale effettua la convoluzione con la porzione corrente di input, mentre il parametro *filters* indica il numero di canali di output del layer convoluzionale. Per questo modello è stata definita una configurazione standard di 64 filters ed una dimensione del kernel pari a 4, ovvero il numero di time steps considerati come sequenza di input. Questo layer è seguito da uno strato di *MaxPooling* il cui compito è ridurre la complessità dell'output del layer convoluzionale, identificando gli elementi più

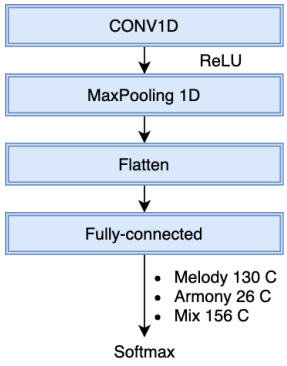


Figura 23: Architettura rete CNN

salienti, prevenendo inoltre l'overfitting. Il livello convoluzionale e di pooling sono seguiti da un fully-connected layer che interpreta le caratteristiche estratte dalla parte convoluzionale del modello. Un layer *Flatten* viene utilizzato tra i livelli convoluzionali e il livello *Dense* per rimuovere qualsiasi struttura (spaziale o di altro tipo) di dati, trasformando l'output in un singolo vettore monodimensionale.

5.2.5 Convolutional Neural Network - Long short-term memory

L'architettura CNN LSTM prevede l'utilizzo di livelli di rete neurale convoluzionale (CNN) per l'estrazione di features sui dati di input combinati con LSTM per supportare la previsione della sequenza. È utile pensare a questa architettura come alla definizione di due sottomodelli: il modello CNN per l'estrazione delle caratteristiche e il modello LSTM per l'interpretazione delle caratteristiche attraverso le varie fasi temporali.

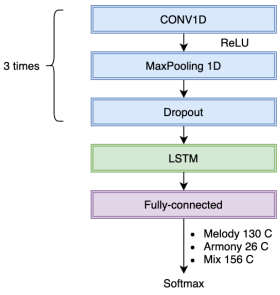


Figura 24: Architettura rete CNN-LSTM

L'ispirazione di questo modello deriva dal lavoro di Keunwoo Choi et al. [20]. Questo modello utilizza un layer CNN 1-D che esegue operazioni di convoluzione attraverso la dimensione temporale, avente parametri di configurazione pari a quelli descritti in 5.2.4. La catena del modello convolutivo prevede successivamente un layer che applica la funzione di attivazione ReLU, al quale segue il layer Max-Pooling per la riduzione della complessità ed infine termina con lo strato Dropout. Questa catena di operazioni viene iterata per 3 volte.
L'output del layer convoluzionale 1-D viene inserito in una rete LSTM che si occupa di trovare una struttura a breve e lungo termine della canzone, utilizzando 96 hidden units. L'output di LSTM viene passato ad uno strato fully-connected con numero di unità pari alle classi previste dal corrente metodo di generazione del brano, utilizzando SoftMax come funzione di attivazione per ottenere la distribuzione di probabilità sulle classi di output.

5.2.6 Generative adversarial network

In una rete GAN, il modello discriminatore ed il modello generatore vengono addestrati e testati contemporaneamente. Al discriminatore vengono presentati dati reali, nel caso corrente brani musicali, nonché dati falsi generati da rumore casuale. La rete è definita da una classe *GAN*, la quale nel costruttore accetta come parametri le dimensioni della matrice di training per definire l'attributo tupla seq_shape.

Come primo passo dell'architettura GAN è stato gestito il *discriminatore*, il quale presenta due livelli LSTM, uno singolo ed uno bidirezionale che gli consentono di apprendere gli schemi nei dati di input. Senza questi due livelli, il discriminatore non sarebbe in grado di distinguere i dati reali da quelli falsi, a vantaggio del generatore, che conosce il dominio dei dati di input. In seguito ai livelli ricorrenti sono presenti due layer fully-connected con rispettivamente 512 e 256 neuroni, separati da uno strato LeakyReLU, il quale viene favorito rispetto al ReLU poiché quest'ultimo soffre il problema del "Dying ReLU" nel quale alcuni neuroni muoiono e rimangono inattivi indipendentemente dall'input fornito, influenzando le performance della rete. Questo problema può essere corretto utilizzando il Leaky ReLU nel quale gli output per tutte le $x < 0$ sono valori piccoli e diversi da zero. Come layer finale è presente la funzione di attivazione sigmoide, la quale indicherà se la sequenza di note è stata generata o reale.

La rete del *generatore* è un percettrone multistrato che riceve in ingresso input casuali di dimensioni pari alla dimensione latente. Sono presenti strati LeakyReLU tra i layer fully-connected per diminuire la sparsità, la quale ostacolerebbe il training della GAN, ed infine come ultima funzione di attivazione è presente una *tanh*, la quale è preferita alla sigmoide nell'architettura del generatore secondo

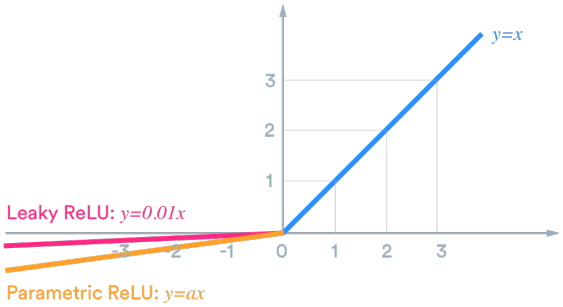


Figura 25: Leaky ReLU

puramente un euristica [21].

Durante la fasi di **training** per prima cosa viene definito il parametro $batch\_size = 64$ che rappresenta il numero di campioni reali e generati che verranno considerati nell'epoca corrente. Viene identificato un vettore di indici random compresi nel range $[0, X\_train.shape[0]]$ di lunghezza batch_size, il quale restituirà gli indici degli elementi che verranno considerati come dati reali all'interno della matrice di training. Tale matrice verrà data in input al metodo *train_on_batch* del discriminatore per iniziare la fase di training, oltre ad un vettore di 1 di lunghezza batch_size per indicare la veridicità dei dati. Il metodo train_on_batch accetta un singolo batch di dati come input, esegue la backpropagation ed infine aggiorna i parametri del modello.
Durante la fase di addestramento al generatore viene fornita una matrice di dimensioni $(batch\_size, latent\_dimension)$ di rumore casuale ottenuto da una distribuzione normale standard, producendo come output una matrice di valori generati. Quest'ultima rappresenta l'input fornito alla funzione che esegue un'ulteriore fase di training del discriminatore, nella quale verrà indicato che i parametri in ingresso sono dati generati.
Completato il training del discriminatore si effettua l'allenamento del generatore. Il valore di loss viene calcolato a partire dal modello *combinato* (generatore e discriminatore), il quale riceve come input dati veri e generati ed effettua la backpropagation attraverso l'intero modello. Questo permette di allenare il generatore ed il discriminatore insieme come modello combinato, ottenendo un discriminatore sempre più vigile nel riconoscere un dato generato da uno vero, il quale di conseguenza comporterà un miglioramento del generatore.

```
#  Training the Generator
noise = np.random.normal(0, 1, (batch_size, self.latent_dim))

# The generator wants the discriminator to label
# the generated samples as valid (ones)
```

```
valid_y = np.array([1] * batch_size)

# Train the generator (to have the discriminator label samples as real)
g_loss = self.combined.train_on_batch(noise, valid_y)
```

Listing 5.2: Training del generatore

5.2.7 Generazione MIDI

La generazione del brano MIDI, dato il modello allenato, è possibile programmarla al conseguimento di x epoche, tramite la callback *GenerateMidi* la quale esegue l'override del metodo *on_epoch_end* che viene richiamato al completamento di ogni epoca. In base al parametro definito, verificherà se procedere con la generazione o meno. I metodi riguardanti il processo di generazione midi sono contenuti nello script *generator.py*. Come primo step viene eseguito il metodo *getMIDIData*, il quale genera un vettore di oggetti definiti come in sezione 5.3.

```
note_obj = {
        'm_is_rest': m_is_rest,
        'm_pitch': note,
        'm_is_holding': m_is_holding,
        'c_is_rest': c_is_rest,
        'c_type': chord,
        'c_is_holding': c_is_holding,
}
```

Listing 5.3: Definizione dell'oggetto midi

L'ordinamento di tali oggetti guiderà successivamente la generazione midi. L'oggetto descrive il sedicesimo di nota, per quanto riguarda la melodia, ed il sedicesimo di accordo. L'attributo $(m/c)\_is\_rest$ identifica se è presente una pausa, $(m/c)\_is\_holding$ se la nota o l'accordo è legato al sedicesimo precedente, *m_pitch* identifica il pitch della nota, mentre *c_type* rappresenta l'accordo da eseguire. L'oggetto riporta sia i dati della melodia, sia quelli armonici per poter generare il brano completo di entrambe le sequenze in contemporanea. Il metodo *getMIDIData* seleziona un elemento dalla matrice di test come punto di partenza della sequenza che andrà a generare. Tramite *model.predict*(*sequence*) viene predetto il sedicesimo successivo nella sequenza, successivamente viene definito l'oggetto *note_obj*, inserito nel vettore di output ed infine viene aggiunto ai valori della matrice di partenza, la quale sarà l'input per la nuova predizione del modello. Il metodo che effettua la generazione del file midi è *generateMIDI*, il quale prende

in ingresso l'output di *getMIDIData*. Viene creata un'istanza della classe *Pretty-MIDI* [6] alla quale vengono aggiunte le note della melodia tramite il metodo 5.4 nel quale si specifica il pitch e la durata della nota, mantenendo la velocity pari a 100.

```
pretty_midi.Note(velocity=m_velocity,
                                    pitch=m_pitch, start=start, end=end)
```

Listing 5.4: Inserimento di una nota nella traccia MIDI

Per l'accordo invece vengono ottenute le note che lo compongono tramite la libreria *pychord* [7], la quale mette a disposizione la funzione *components()* che viene eseguita sull'istanza *Chord*.

[6] http://craffel.github.io/pretty-midi/
[7] https://github.com/yuma-m/pychord

Capitolo 6

Test

6.1 Risultati

6.1.1 Generazione melodica

L'analisi delle classi di output ha portato alla luce uno sbilanciamento evidente del dataset, il quale risulta essere concentrato all'interno del range 60–80, riscontrando inoltre un'assenza di valori negli intervalli 0–40 e 100–120. La classe più presente all'interno del dataset è riferita al valore 129 che equivale alla pausa. Durante la creazione della matrice rappresentante il dataset è stato inserito un middleware responsabile dell'augmentation, nella quale per ogni ottava all'interno del range -3–3, veniva scelto in modo casuale se abbassare o alzare la traccia midi corrente.

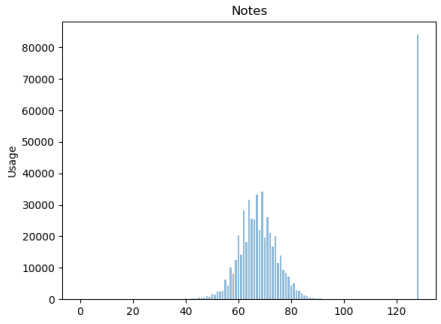


Figura 26: Distribuzione del dataset (melodia) prima dell'augmentation

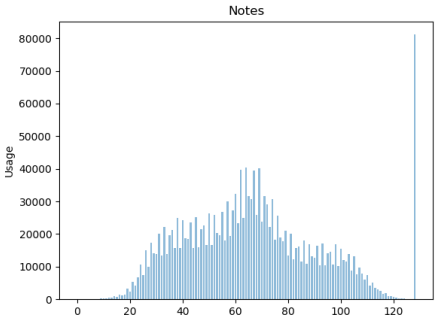


Figura 27: Distribuzione del dataset (melodia) dopo l'augmentation

I grafici di loss e accuracy riferiti alle reti neurali ricorrenti hanno evidenziato un andamento lento di discesa della loss che dopo 100 epoche rimane stabile a 2.2, mentre il valore maggiore raggiunto di accuracy è quello della LSTM bidirezionale,

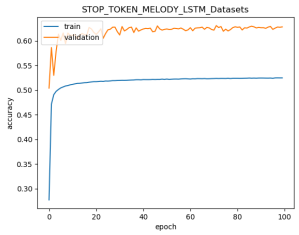


Figura 28: Accuracy LSTM, melodia

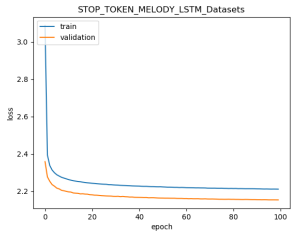


Figura 29: Loss LSTM, melodia

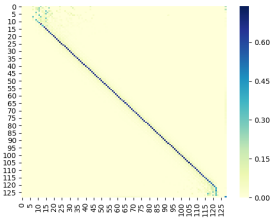


Figura 30: Confusion matrix LSTM, melodia

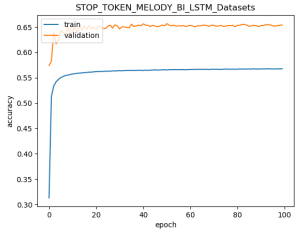


Figura 31: Accuracy BI-LSTM, melodia

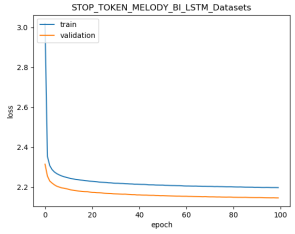


Figura 32: Loss Bi-LSTM, melodia

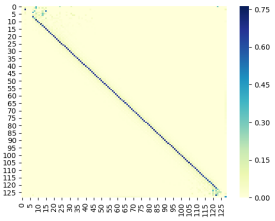


Figura 33: Confusion matrix BI-LSTM, melodia

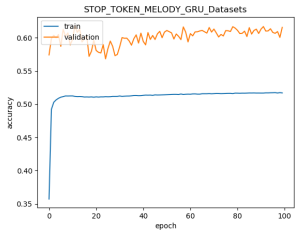


Figura 34: Accuracy GRU, melodia

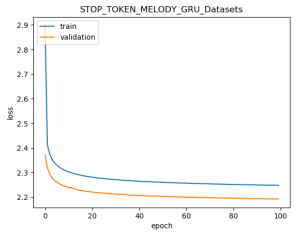


Figura 35: Loss GRU, melodia

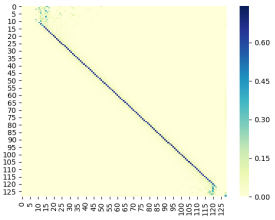


Figura 36: Confusion matrix GRU, melodia

la quale dopo poche epoche raggiunge il valore 0.65 e rimane stabile fino alla fine del training. Il divario tra i valori di accuracy riferiti al batch di training ed i valori di loss rimane stabile a 0.10.

La rete puramente convoluzionale riporta valori insignificanti, identificando l'impossibilità di interpretazione dei risultati. Questa tipologia di rete ottiene un miglioramento nei risultati aggiungendo layer ricorrenti all'architettura, raggiungendo dopo 100 epoche un valore di loss superiore di 0.5 rispetto alle reti ricorrenti ed un valore di accuracy rispetto al validation set in progressivo aumento. La confusion matrix di quest'ultima rispetto alle reti ricorrenti presenta errori di valutazione nella prima ed ultima ottava.

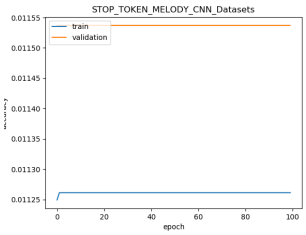


Figura 37: Accuracy CNN, melodia

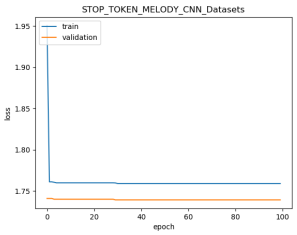


Figura 38: Loss CNN, melodia

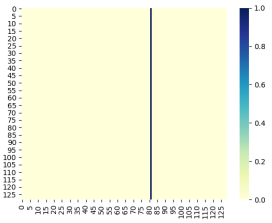


Figura 39: Confusion matrix CNN, melodia

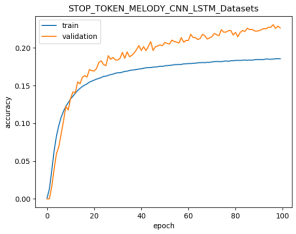


Figura 40: Accuracy CNN-LSTM, melodia

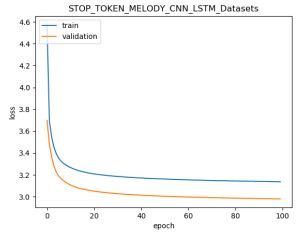


Figura 41: Loss CNN-LSTM, melodia

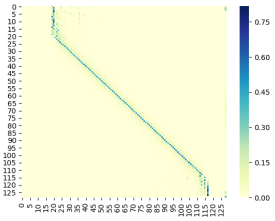


Figura 42: Confusion matrix CNN-LSTM, melodia

6.1.2 Generazione armonica

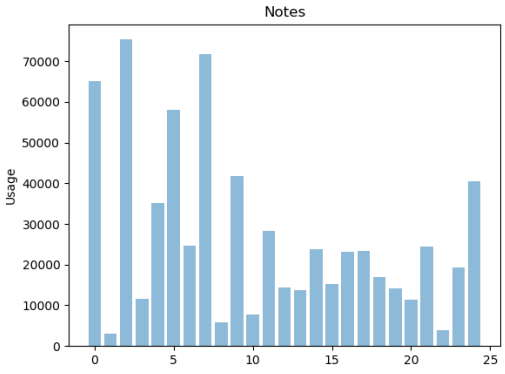


Figura 43: Distribuzione del dataset (armonia)

La generazione armonica prevede un range di valori minore rispetto a quella melodica poiché sono presenti solamente i 24 accordi principali, pertanto non è

stato necessario effettuare augmentation sul dataset.

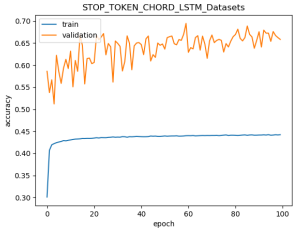


Figura 44: Accuracy LSTM, armonia

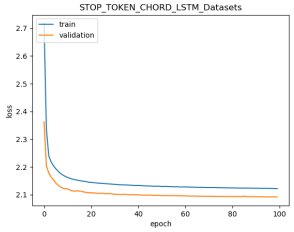


Figura 45: Loss LSTM, armonia

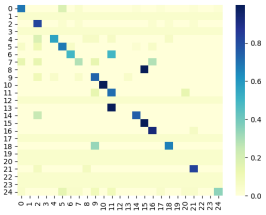


Figura 46: Confusion matrix LSTM, armonia

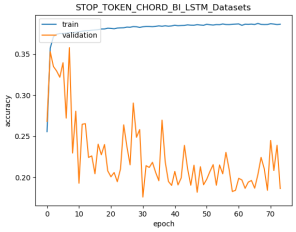


Figura 47: Accuracy BI-LSTM, armonia

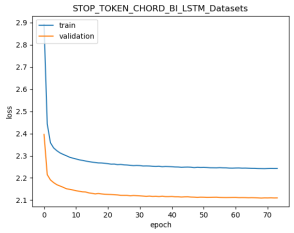


Figura 48: Loss Bi-LSTM, armonia

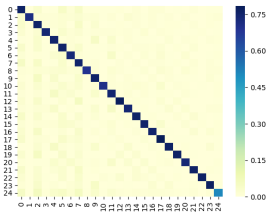


Figura 49: Confusion matrix BI-LSTM, armonia

Figura 50: Accuracy GRU, armonia

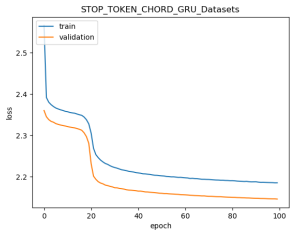


Figura 51: Loss GRU, armonia

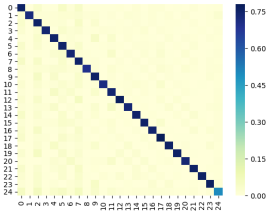


Figura 52: Confusion matrix GRU, armonia

I grafici riferiti alle reti ricorrenti utilizzate nella generazione armonica, rispetto a quelli caratterizzanti la generazione melodica, presentano un peggioramento dell'andamento dell'accuracy, ad esempio nel caso della LSTM bidirezionale nella quale l'andamento decrescente della curva riferita al validation set e quello ascendente della curva riferita al training set suggeriscono la presenza di overfitting. A

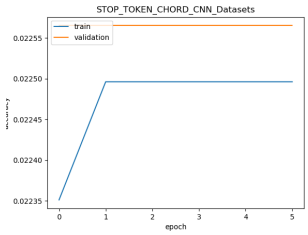


Figura 53: Accuracy CNN, armonia

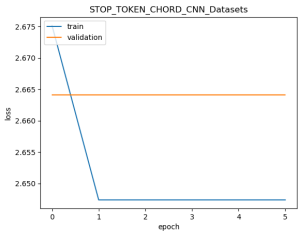


Figura 54: Loss CNN, armonia

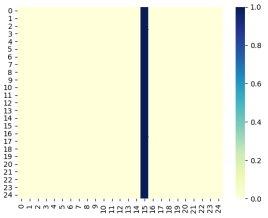


Figura 55: Confusion matrix CNN, armonia

differenza di quest'ultima, la LSTM propone due grafici con andamento regolare, ottenendo un valore di loss simile a quello ottenuto nella generazione melodica ed un valore di accuracy sul validation set che si avvicina a 0.7. La confusion matrix mostra una maggiore precisione di predizione delle classi centrali.

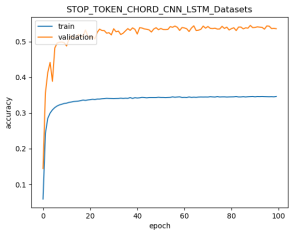


Figura 56: Accuracy CNN-LSTM, armonia

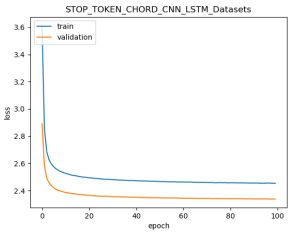


Figura 57: Loss CNN-LSTM, armonia

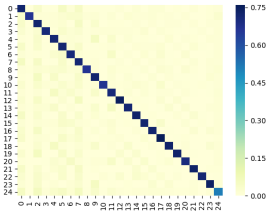


Figura 58: Confusion matrix CNN-LSTM, armonia

La rete convoluzionale anche in questo caso non fornisce dati utili al fine dell'addestramento, mentre la versione con gli strati ricorrenti produce una curva di accuracy sul validation set più stabile rispetto a quella riferita alla LSTM pura, malgrado dopo 100 epoche raggiunga un valore massimo di 0.5.

6.1.3 Generazione melodica ed armonica

Per il dataset che comprende sia la parte melodica sia quella armonica non è stato effettuato nessun processo di augmentation, poiché si avrebbe dovuto replicare la sezione armonica, la quale non considera i valori differenti per ottava.
Il dato più presente è riferito all'holding di nota pertanto non viene considerato durante la predizione, poiché viene gestito in relazione ad una threshold.

E' evidente un peggioramento dei risultati generale, soprattutto delle reti ricorrenti nelle quali il valore massimo di accuracy scende drasticamente ad un massimo

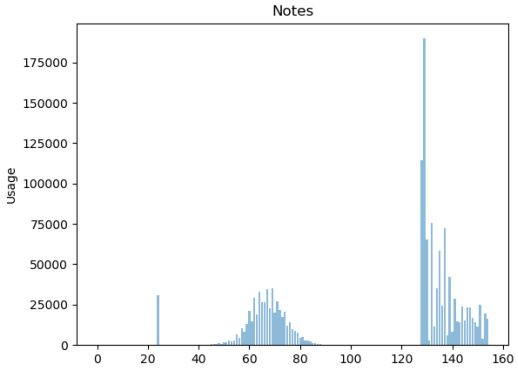


Figura 59: Distribuzione del dataset (melodia e armonia)

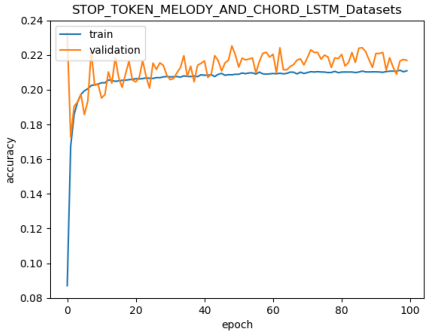


Figura 60: Accuracy LSTM, melodia e armonia

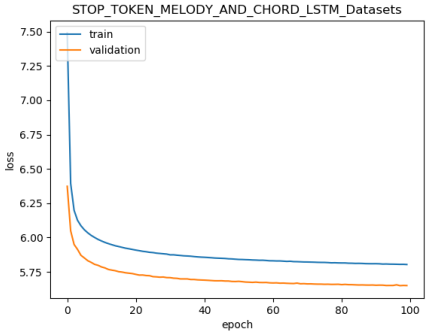


Figura 61: Loss LSTM, melodia e armonia

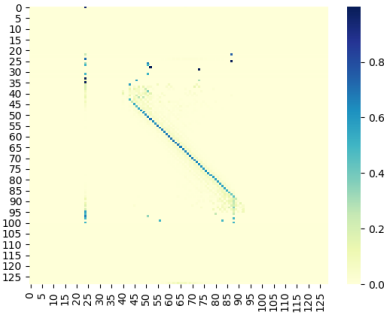


Figura 62: Confusion matrix LSTM, melodia e armonia (Valori melodici)

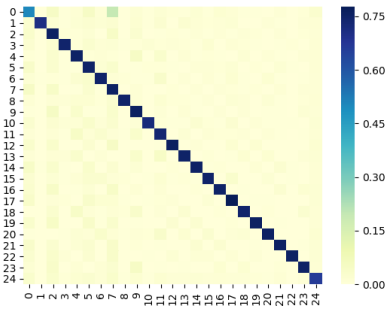


Figura 63: Loss LSTM, melodia e armonia (Valori armonici)

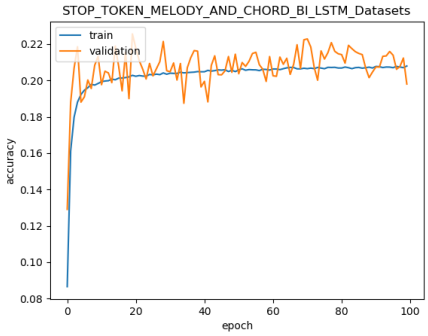


Figura 64: Accuracy BI-LSTM, melodia e armonia

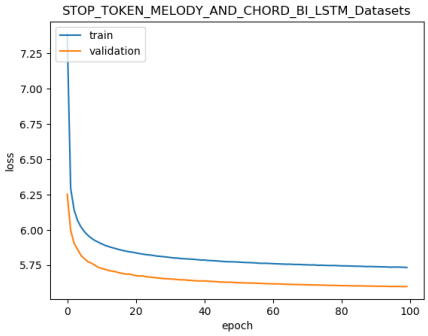


Figura 65: Loss BI-LSTM, melodia e armonia

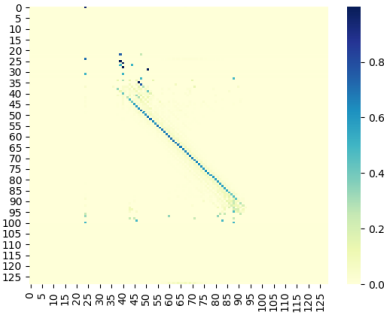


Figura 66: Confusion matrix BI-LSTM, melodia e armonia (Valori melodici)

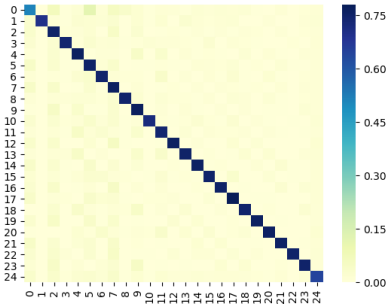


Figura 67: Loss BI-LSTM, melodia e armonia (Valori armonici)

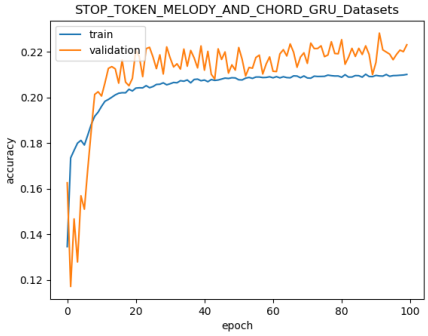


Figura 68: Accuracy GRU, melodia e armonia

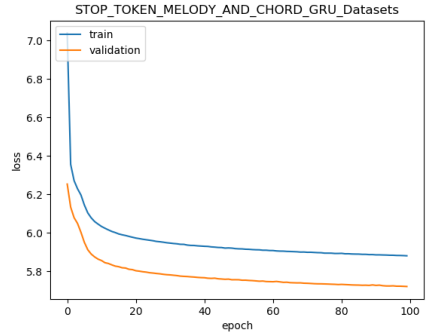


Figura 69: Loss GRU, melodia e armonia

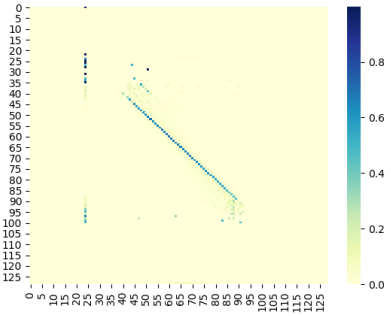


Figura 70: Confusion matrix GRU, melodia e armonia (Valori melodici)

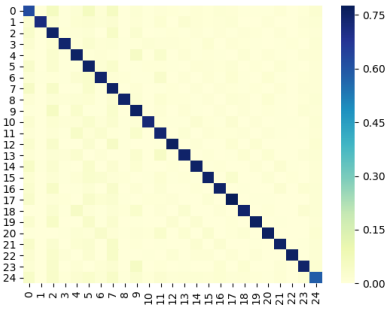


Figura 71: Loss GRU, melodia e armonia (Valori armonici)

di 0.22, mentre la curva di loss sul validation set dopo 100 epoche si stabilizza ad un valore superiore a 5, il doppio rispetto agli altri metodi di generazione.

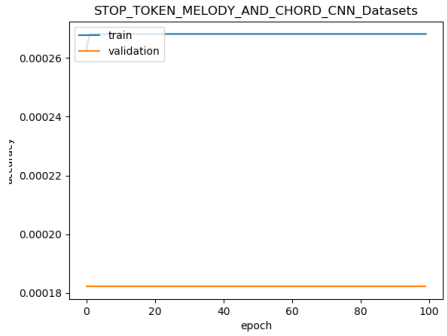


Figura 72: Accuracy CNN, melodia e armonia

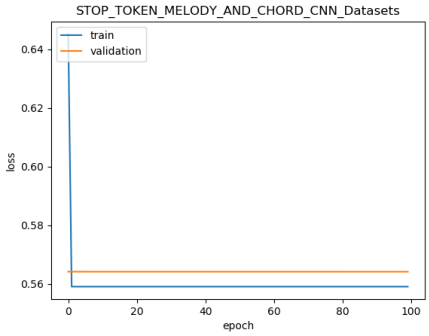


Figura 73: Loss CNN, melodia e armonia

Anche in questo caso risulta evidente come le reti convoluzionali stentino a proporre risultati accettabili.

6.2 Osservazioni

Dai risultati ottenuti si è evidenziato come l'approccio convoluzionale ad una dimensione sia il modello meno preciso, in accordo con l'affermazione di Chollet [24], il quale afferma come il semplice appiattimento delle sequenze e l'addestramento di un singolo strato *Dense* porti ad un modello che tratta ogni token della sequenza di input separatamente, senza considerare le relazioni tra token e la struttura delle frasi. Risulta migliore la soluzione con l'aggiunta di strati ricorrenti sopra le

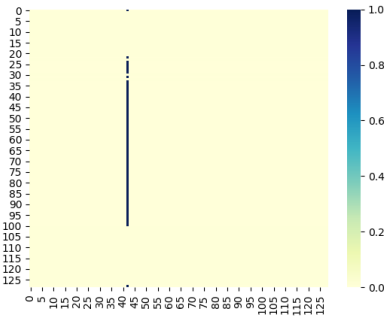


Figura 74: Confusion matrix CNN, melodia e armonia (Valori melodici)

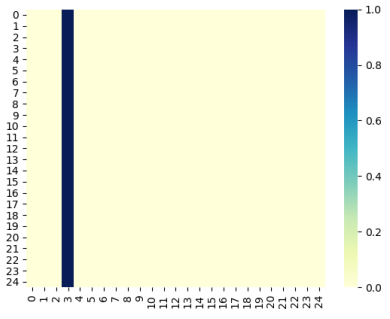


Figura 75: Loss CNN, melodia e armonia (Valori armonici)

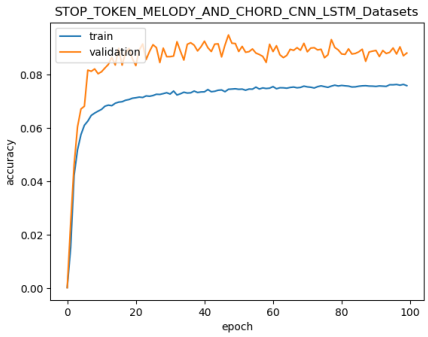


Figura 76: Accuracy CNN-LSTM, melodia e armonia

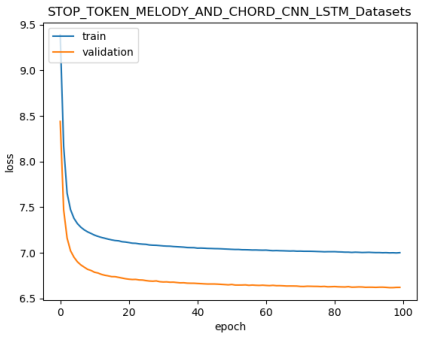


Figura 77: Loss CNN-LSTM, melodia e armonia

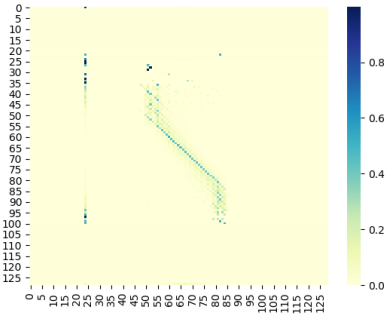


Figura 78: Confusion matrix CNN-LSTM, melodia e armonia (Valori melodici)

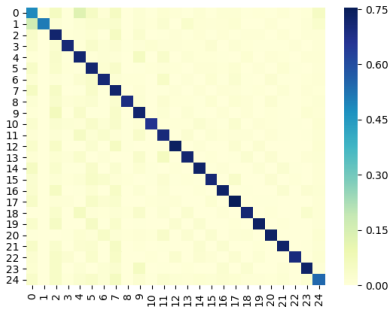


Figura 79: Loss CNN-LSTM, melodia e armonia (Valori armonici)

sequenze per imparare caratteristiche che tengano conto di ognuna nel suo insieme.

Per la valutazione qualitativa del modello GAN la metodologia ancora più utilizzata è la "Rating and Preference Judgment", nella quale viene chiesto ad una giuria di classificare o confrontare esempi di output reali e generati dal dominio. Il metodo di "Rapid Scene Categorization" è generalmente lo stesso, anche se gli output vengono presentati ai giudici per un periodo di tempo molto limitato, come una frazione di secondo, e classificati come reali o generati.

6.3 Valutazione

La *Computational Creativity* è il campo dell'intelligenza artificiale in cui si studia come costruire modelli computazionali che possano modellare, simulare o replicare la creatività umana. Dal punto di vista ingegneristico, è auspicabile disporre di misure concrete per valutare i progressi compiuti da una versione di un programma ad un'altra o per confrontare diversi sistemi software per lo stesso compito creativo. In [22] viene sostenuto che il *test di Turing* [1] è in gran parte inappropriato ai fini della valutazione nella creatività computazionale, poiché tenta di omogeneizzare la creatività in un unico stile, non tenendo conto dell'importanza delle informazioni di base e contestuali per un atto creativo. Esso incoraggia avanzamenti superficiali, premiando la creatività che aderisce a un certo stile rispetto a quello che crea qualcosa veramente di nuovo.
In alternativa al test di Turing, in [23] vengono proposti due modelli descrittivi per la valutazione del software creativo. Il modello FACE che descrive gli atti creativi eseguiti dal software in termini di tuple di atti generativi ed il modello IDEA che descrive come tali atti creativi possano avere un impatto su un pubblico ideale, grazie alle informazioni ideali sulle conoscenze di base e sul processo di sviluppo del software. Questi modelli non intendono catturare la creatività umana, né tutta la creatività computazionale, bensì forniscono un possibile modo di descrivere software per scopi creativi.
I processi di generazione e valutazione sono considerati fondamentali negli studi sulla creatività. Questa distinzione viene mantenuta nei due modelli descrittivi; FACE, che propone gli atti di creatività come unità fondamentali da valutare nei sistemi creativi, e IDEA, che descrive le modalità di valutazione degli atti.

La valutazione dell'elaborato è stata effettuata tramite la *Rating and Preference Judgment* su un campione di 20 persone di età compresa tra i 18 e 82 anni.

[1] Il test di Turing è un processo che consente di misurare se una macchina è in grado o meno di pensare, di farlo autonomamente e di farlo esattamente come un essere umano.

Ogni individuo ha ascoltato, per ogni tipologia di generazione, sette sequenze di cui una reale mentre le altre generate dalle reti neurali descritte precedentemente, valutandole come reali o generate. Risulta evidente come ci sia un drastico peggioramento della qualità nella generazione completa di melodia ed armonia, mentre conducono a buoni risultati le reti ricorrenti nelle altre due tipologie generative.

NNs / Valutazioni	Reali	Generati
Sample Reale	17	3
LSTM	15	5
Bi-LSTM	12	8
GRU	13	7
CNN	0	20
CNN-LSTM	8	13
GAN	13	7

Tabella 1: Valutazione tramite Rating and Preference Judgment dei brani musicali prodotti dalla generazione melodica

NNs / Valutazioni	Reali	Generati
Sample Reale	19	1
LSTM	7	13
Bi-LSTM	16	4
GRU	8	12
CNN	0	20
CNN-LSTM	9	11
GAN	14	6

Tabella 2: Valutazione tramite Rating and Preference Judgment dei brani musicali prodotti dalla generazione armonica

6.3.1 Modello FACE

Il processo FACE definisce un atto creativo come una tupla non vuota, contenente esattamente zero o un'istanza di otto tipologie di atti generativi:

- E^g: espressione di un *concetto*
- E^p: un metodo per generare *espressioni di un concetto*

NNs / Valutazioni	Reali	Generati
Sample Reale	20	0
LSTM	5	15
Bi-LSTM	6	14
GRU	3	17
CNN	0	20
CNN-LSTM	5	15
GAN	2	18

Tabella 3: Valutazione tramite Rating and Preference Judgment dei brani musicali prodotti dalla generazione melodica e armonica

- C^g: un *concetto*
- C^p: un metodo per generare *concetti*
- A^g: una *misura estetica*
- A^p: un metodo per generare *misura estetiche*
- F^g: un'*informazioni frammentaria*
- F^p: un metodo per generare *informazioni frammentarie*

Un *concetto* rappresenta un qualunque programma eseguibile, o qualcosa che può essere considerato tale, che può ricevere input e produrre un output, mentre con la frase *espressione di un concetto* si intende un'istanza (input, output) prodotta quando viene eseguito il concetto.
Con il termine *misura estetica* si intende una funzione che prende come input una tupla (concetto, espressione), nella quale ogni valore può essere *null*, e fornisce come output un valore reale tra 0 e infinito. Con *informazione frammentaria* si identifica una parte di linguaggio naturale comprensibile da un umano, la quale potrebbe essere specifica di un determinato dominio, aggiungendo valore agli atti generativi, come ad esempio: aggiungere un contesto storico o culturale, fornire calcoli riguardanti i concetti/espressioni rispetto alle misure estetiche, oppure offuscare i processi creativi per lasciare una maggior libertà interpretativa ai membri del pubblico. Ad esempio, nello scrivere (F^g, A^g, C^g, E^g), si denota un atto creativo composto da 4 atti generativi. Esso identifica che E^g genera un'espressione di un concetto prodotta da C^g, che l'output della tupla (C^g, E^g) può essere preso come input dalla misura estetica generata da A^g, e che F^g ha generato un'informazione frammentaria che giustifica o illustra l'intero atto creativo.
Per semplificare le cose, verrà utilizzata la lower-case notation per indicare l'output

degli atti generativi. Nelle tuple possono inoltre coesistere atti generativi aventi apice g o p, i quali indicano rispettivamente atti di base e di processo. Ad esempio la tupla (C^p, C^g, E^g) denota un atto creativo mediante il quale un sistema software compie un atto generativo a livello di processo, C^p, che inventa un nuovo metodo per generare concetti, c^p, il quale verrà utilizzato in un concept-generating act, C^g. Esso produce un concetto, c^g, il quale a sua volta è stato utilizzato in un expression-generating act, E^g, per produrre l'espressione e^g di c^g.
Per confrontare due sistemi creativi esistono vari metodi: **quantitativo** che analizza il volume degli atti creativi, **cumulativo** il quale risulta essere più informativo poiché viene considerato il numero di atti generativi che avvengono, **comparativo** dove ogni atto generativo fornisce un peso diverso, infine **qualitativo** nel quale i valori di output delle funzioni estetiche vengono utilizzati per comparare i sistemi.

L'elaborato corrente è possibile definirlo come atto creativo composto dai seguenti atti generativi: (C^g, E^p, A^g, F^g). Esso identifica che il concetto C^g è in grado di generare espressioni di un concetto (E^p), ovvero molteplici tuple di (input, output). L'output della combinazione tra C^g e E^p può essere preso in considerazione come input dalla misura estetica generata da A^g, la quale può corrispondere ai grafici di loss e accuracy. Infine l'atto generativo F^g indica la presenza di un'informazione frammentaria che illustra l'atto creativo nella sua interezza.

6.3.2 Modello IDEA

All'interno del modello IDEA si formalizzano le nozioni su come misurare gli atti creativi, in termini di nozioni relative all'impatto. Si assume un ciclo (I)terative (D)evelopment (E)xecution (A)ppreciation all'interno del quale viene progettato il software ed il suo comportamento viene esposto ad un pubblico. Si assume la presenza di un pubblico ideale di individui i, in grado di fornire due indicatori dell'effetto che un atto creativo individuale, A, ha avuto su di loro:

- Un indicazione del loro cambiamento di benessere, $wb_i(A)$, compreso tra -1 e 1. Con -1 si indica un peggioramento dello stato attuale, $+1$ un miglioramento, mentre con 0 nessuna differenza.
- Un indicazione tra 0 e 1 dello sforzo cognitivo che hanno trascorso nel tentativo di apprezzare un atto creativo e l'artefatto che ha prodotto, $ce_i(A)$. Si indica infine con $m(A)$ il valore medio della valutazione del benessere su n persone.

In [22] vengono proposte le seguenti misure:

- $disgusto(A) = \frac{1}{2n} \sum_{i=1}^{n} (1 - wb_i(A))$

- $divisione(A) = \frac{1}{n}\sum_{i=1}^{n} |wb_i(A) - m(A)|$
- $indifferenza(A) = 1 - \frac{1}{n}\sum_{i=1}^{n} |wb_i(A)|$
- $popolarit(A) = \frac{1}{2n}\sum_{i=1}^{n}(1 + wb_i(A))$
- $provocazione(A) = \frac{1}{n}\sum_{i=1}^{n}(ce_i(A))$

Come descritto in 6.3, anche per il modello IDEA, i file musicali generati dalle reti neurali sono stati sottoposti alla valutazione di un campione di 20 persone, ciascuna delle quali ha indicato il personale cambiamento di benessere $wb_i(A)$ per ogni sequenza musicale con i soli valori -1, 0 ed 1, evitando valori intermedi per facilitare l'ascoltatore. In seguito alla raccolta dei dati sono state poi calcolate le misure di disgusto, indifferenza e popolarità descritte in [22].

Dalla tabella 4 riferita alla generazione melodica si evince che il modello CNN risulta essere il meno apprezzato, con la conseguente scarsa indifferenza. La sequenza musicale reale si conferma la più popolare, seguita dal modello LSTM. Poco apprezzato è il modello GAN che, dopo il CNN, è il modello che ha ottenuto il più alto numero di valori negativi relativi al cambio di benessere.

NNs / Cambiamento di benessere	-1	0	+1	disgusto	indifferenza	popolarità
Sample Reale	0	5	15	0.125	0.25	0.875
LSTM	4	9	7	0.425	0.45	0.575
Bi-LSTM	4	10	6	0.45	0.5	0.55
GRU	12	2	6	0.65	0.1	0.35
CNN	19	1	0	0.975	0.05	0.025
CNN-LSTM	3	10	7	0.4	0.5	0.6
GAN	14	1	5	0.725	0.05	0.275

Tabella 4: Valutazione tramite modello IDEA dei brani musicali prodotti dalla generazione melodica

Riguardo ai valori corrispondenti alla generazione armonica, illustrati nella tabella 5, si nota come la seconda popolarità più alta dopo quella del brano reale, è riferita al modello LSTM bidirezionale, il quale ha ottenuto 14 riscontri positivi durante l'ascolto. Un netto miglioramento è stato ottenuto dal modello GAN, il quale vede invertiti i valori di popolarità e disgusto rispetto alla generazione melodica.

La tabella 6 riferita alla generazione melodica ed armonica mostra un netto peggioramento della popolarità di tutti i modelli, la quale supera 0.1 solo per

NNs / Cambiamento di benessere	-1	0	+1	disgusto	indifferenza	popolarità
Sample Reale	0	1	19	0.025	0.05	0.975
LSTM	10	2	8	0.55	0.1	0.45
Bi-LSTM	3	3	14	0.225	0.15	0.775
GRU	5	10	5	0.5	0.5	0.5
CNN	19	1	0	0.975	0.05	0.025
CNN-LSTM	6	8	6	0.5	0.4	0.5
GAN	5	1	14	0.275	0.05	0.725

Tabella 5: Valutazione tramite modello IDEA dei brani musicali prodotti dalla generazione armonica

i modelli CNN-LSTM e GAN. La poca indifferenza e gli alti valori di disgusto descrivono una regressione nella qualità di tali modelli.

NNs / Cambiamento di benessere	-1	0	+1	disgusto	indifferenza	popolarità
Sample Reale	1	5	14	0.175	0.25	0.825
LSTM	18	2	0	0.95	0.1	0.05
Bi-LSTM	17	3	0	0.925	0.15	0.075
GRU	19	1	0	0.975	0.05	0.025
CNN	19	1	0	0.975	0.05	0.025
CNN-LSTM	15	3	2	0.825	0.15	0.175
GAN	11	3	6	0.625	0.15	0.375

Tabella 6: Valutazione tramite modello IDEA dei brani musicali prodotti dalla generazione melodica ed armonica

Capitolo 7

Conclusioni

7.1 Conclusioni

Nel seguente elaborato è stata presentata una comparazione di vari modelli supervisionati con l'aggiunta del modello GAN per analizzare il loro comportamento nella classificazione di singole note musicali, accordi e l'unione di entrambe. I risultati ottenuti hanno esplicitato una maggior confidenza da parte delle reti ricorrenti a riconoscere i pattern delle tracce midi, mentre la rete convoluzionale ha riportato i risultati peggiori dell'elaborato. Un netto miglioramento della CNN è stato raggiunto aggiungendo strati ricorrenti nella sequenza di layer, i quali hanno portato il modello a riconoscere sequenze nel tempo. Per la generazione melodica la Bi-LSTM si è confermata come miglior modello tra quelli analizzati, mentre per la classificazione di accordi la LSTM ha raggiunto il maggior valore di accuratezza.

La GRU in entrambi i casi ha prodotto risultati di qualità lievemente inferiore rispetto alle altre due versioni ricorrenti. Tali risultati non si sono confermati per la generazione mista di armonia e melodia, nella quale è stato riscontrato un notevole peggioramento dell'apprendimento, nelle reti ricorrenti, che ha portato il valore di accuratezza dopo 100 epoche a diminuire di un terzo ed a triplicare il valore di loss rispetto ai valori ottenuti durante le altre due tipologie di generazione. Il modello GAN, a differenza degli altri modelli, riesce a generare file midi strutturalmente più complessi dal momento che riesce a predire valori di nota o accordi all'interno di un range più ampio, non riuscendo però a tenere in considerazione le varie parti del brano musicale, portando l'output ad assomigliare ad un assolo.

7.2 Sviluppi futuri

Per ottenere valori qualitativamente migliori rispetto a quelli prodotti nel seguente elaborato si potrebbe provare l'apprendimento con dataset avente una distribuzione migliore all'interno del range midi, oltre a considerare gli accordi associati alla loro ottava di appartenenza. Sarei interessato inoltre a gestire diversamente il valore che indica se determinare la legatura di nota che, al momento, è all'interno della matrice di training e viene considerato rispetto ad una soglia statica. Il dataset utilizzato prevede brani di artisti molto eterogenei tra loro, il quale porta una vasta varietà di strutture ritmiche, pertanto si potrebbe definire a priori un dataset di brani di uno stesso genere musicale, per ottenere una minore variabilità. Ulteriore modifica applicabile sul dataset è la lunghezza dei chunk delle matrici di ogni brano, che nel progetto è fissata a 16 sedicesimi. Il pre-processing quindi crea sezioni uguali fino ad arrivare alla fine del brano senza analizzare in modo più approfondito i pattern di sequenze, pertanto si potrebbe utilizzare il framework [4] in grado di localizzare suffissi ripetuti all'interno di una serie temporale.

Per quanto riguarda le reti neurali sarebbe interessante soffermarsi sulle reti ricorrenti, tralasciando le versioni convoluzionali, effettuando una fase di tuning sui parametri utilizzati nelle varie architetture, tra i quali: numero di neuroni dei vari layer, la percentuale di dropout utilizzata, la profondità della rete, provando inoltre la *Kullback Leibler Divergence Loss* function, la quale è una misura di come una distribuzione di probabilità differisce da una distribuzione di base. Una KL divergence loss di 0 suggerisce che le distribuzioni sono identiche. In pratica, il comportamento di KL Divergence è molto simile alla cross-entropy. Esso calcola quante informazioni si perdono (in termini di bit) se la distribuzione di probabilità predetta venisse usata per approssimare la distribuzione di probabilità desiderata. Tale funzione è più comunemente usata quando si usano modelli che imparano ad approssimare una funzione più complessa rispetto alla semplice classificazione multi classe, come nel caso di un autoencoder usato per imparare una densa feature representation sotto un modello che deve ricostruire l'input originale.

Bibliografia

[1] Sebastian Ruder. An overview of gradient descent optimization algorithms. CoRR, abs/1609.04747, 2016.

[2] Understanding lstm networks. *http://colah.github.io/posts/2015-08-Understanding-LSTMs/.*

[3] Giorgio Valentini. Algoritmi di apprendimento supervisionato. https://homes.di.unimi.it/valentini/SlideCorsi/Bioinformatica1617/AlgClassSuperv.pdf

[4] C. Wang, J. Hsu and S. Dubnov, Music Pattern Discovery with Variable Markov Oracle: A Unified Approach to Symbolic and Audio Representations, The 16th International Society for Music Information Retrieval Conference, 2015.

[5] D. Eck and J. Schmidhuber, "Finding Temporal Structure in Music: Blues Improvisation with LSTM Recurrent Networks", *Proceedings of the 2002 IEEE Workshop, Networks for Signal Processing XII, pp.747-756, 2002.*

[6] G. Hadjeres and F. Pachet, B DeepBach: a Steerable Model for Bach chorales generation, Proceedings of the 34th International Conference on Machine Learning, PMLR 70:1362-1371, 2017.

[7] Chen, Ke, Weilin Zhang, Shlomo Dubnov, and Gus Xia. "The Effect of Explicit Structure Encoding of Deep Neural Networks for Symbolic Music Generation." arXiv preprint arXiv: 1811.08380 (2018).

[8] Elliot Waite, Douglas Eck, Adam Roberts, and Dan Abolafia. Project Magenta: Generating longterm structure in songs and stories, 2016. *https://magenta.tensorflow.org/blog/2016/07/15/lookback-rnn-attention-rnn/.*

[9] Hang Chu, Raquel Urtasun, and Sanja Fidler. Song from PI: A musically plausible network for pop music generation. arXiv preprint arXiv:1611.03477, 2016.

[10] A. Oord et al., WaveNet: A Generative Model for Raw Audio, The 9th ISCA Speech Synthesis Workshop, 2016.

[11] Yang, Li-Chia, Szu-Yu Chou, and Yi-Hsuan Yang. "MidiNet: A convolutional generative adversarial network for symbolic-domain music generation." *Proceedings of the 18th International Society for Music Information Retrieval Conference (ISMIR'2017), Suzhou, China. 2017.*

[12] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. arXiv preprint arXiv:1603.07285, 2016.

[13] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and YiHsuan Yang. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. In *Proceedings of the ThirtySecond AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018, 2018.*

[14] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. Computer, 42(8), 2009.

[15] I. Simon, A. Roberts, C. Raffel, J. Engel, C. Hawthorne, and D. Eck, "Learning a latent space of multitrack measures", *arXiv preprint arXiv:1806.00195, 2018.*

[16] Adam Roberts, Jesse Engel, Colin Raffel, Curtis Hawthorne, and Douglas Eck. A hierarchical latent vector model for learning long-term structure in music. arXiv:1803.05428, 2018.

[17] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Curtis Hawthorne, Andrew M Dai, Matthew D Hoffman, and Douglas Eck. Music transformer: Generating music with long-term structure. *arXiv preprint arXiv:1809.04281, 2018.*

[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Advances in Neural Information Processing Systems, 2017.

[19] M. Hermans and B. Schrauwen. Training and analysing deep recurrent neural networks. In *Proceedings of Neural Information Processing Systems (NIPS). 2013.*

[20] Keunwoo Choi, George Fazekas, Mark B. Sandler, and Kyunghyun Cho. Convolutional recurrent neural networks for music classification. arXiv preprint arXiv:1609.04243, 2016.

[21] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR, abs/1511.06434, 2015.*

[22] Pease A, Colton S. On impact and evaluation in Computational Creativity: A discussion of the Turing Test and an alternative proposal. In: Proceedings of the AISB'11 Convention. York, UK: AISB; 2011.

[23] S. Colton, A. Pease, and J. Charnley. Computational creativity theory: The FACE and IDEA descriptive models. In 2nd International Conference on Computational Creativity. 2011, 2011.

[24] Chollet, F. (2017). Deep Learning with Python. Manning Publications.

Progetto sviluppato presso il Laboratorio di Informatica Musicale
`https://www.lim.di.unimi.it`