UNIVERSITÀ DEGLI STUDI DI TRENTO
Dipartimento di Ingegneria e Scienza dell'Informazione

Corso di Laurea triennale in INFORMATICA

Tesi Finale

UTILIZZO DI RETI NEURALI RICORRENTI PER LA CREAZIONE DI MUSICA

Relatore
Prof. Roberto Battiti

Laureando
Andrea Dellera

Anno accademico 2014-2015

Ai miei genitori.

Contents

1 Introduzione

Lo scopo di questa tesi è di creare delle melodie utilizzando tecniche di machine learning per portare il computer ad imparare brani già scritti per poi creare delle melodie originali. Verrà perseguito facendo inoltre un confronto tra due tipologie di reti neurali: Feed Forward Netwoks (FFN) e Recurrent Neural Networks (RNN).
I passi seguiti per giungere allo scopo sono:

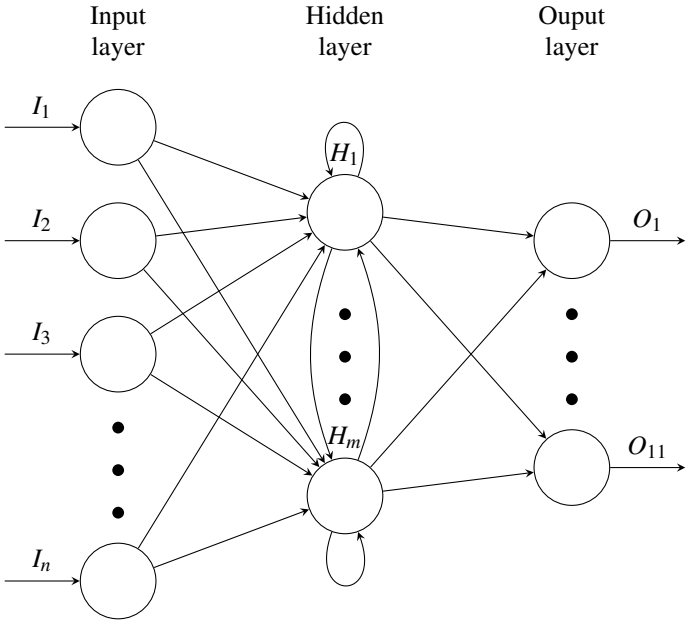
- la definizione delle reti, con tutte le variabili del caso da definire come il numero di input, il numero di neuroni nello strato nascosto ed il numero di output;
- la definizione di una codifica per le note, con la scelta delle proprietà da tenere in considerazione come l'ottava, la durata ed il nome della nota;
- la scelta di un algoritmo per l'allenamento delle reti e delle relative variabili, come ad esempio il tipo di apprendimento e la gestione del dataset;
- l'apprendimento di sequenze di note particolari, messe a forma di cerchio o di otto;
- l'apprendimento di intere canzoni.

Lavori di questo tipo sono già stati fatti, con risultati più e meno soddisfacenti, e si rimanda a [4] per una versione più semplice del problema mentre si consiglia
`http://www.hexahedria.com/2015/08/03/composing-music-with-recurrent-neural-networks/` per una versione più approfondita.

2 Modelli utilizzati

2.1 RNN

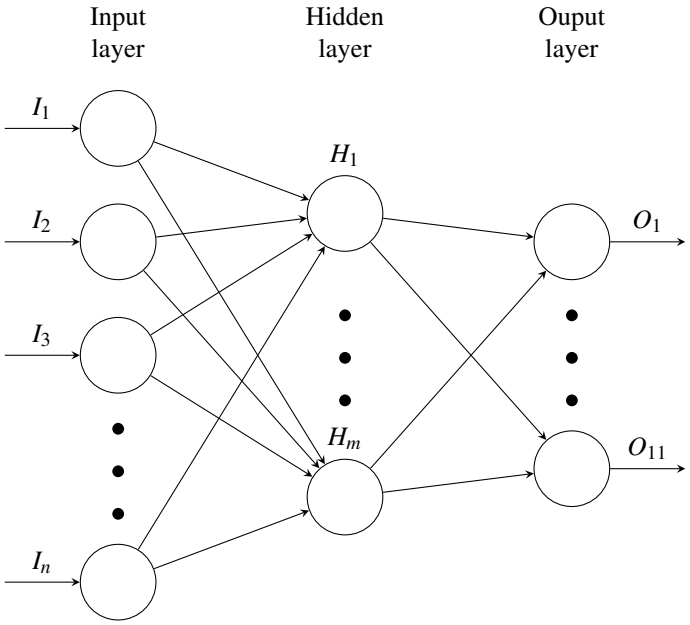
Una rete neurale ricorrente è una rete in cui esistono cammini trai nodi che formano dei cicli. Il tipo utilizzato in questa tesi è Long Short Term Memory (LSTM). Una rete LSTM, composta da neuroni LSTM, ha la capacità di ricordare gli elementi passati per un determinato tempo, quindi per un certo numero di attivazioni, ed è quindi indicata in problemi dove ci sono da imparare sequenze e visto che nel problema affrontato devono essere imparate sequenze di note è sembrata la più appropriata. Il grafo che la rappresenta è il seguente:



La parte più importante, quella che caratterizza la RNN è la connessione tra tutti i neuroni presenti nello strato nascosto.

2.2 FFN

Una rete neurale di questo tipo è la più classica, la prima e la più semplice ad essere implementata, con cammini che vanno da uno strato al successivo senza formare cicli; questa è la caratterisca più importante che la distingue da una RNN. Il grafo di questa rete è il seguente:

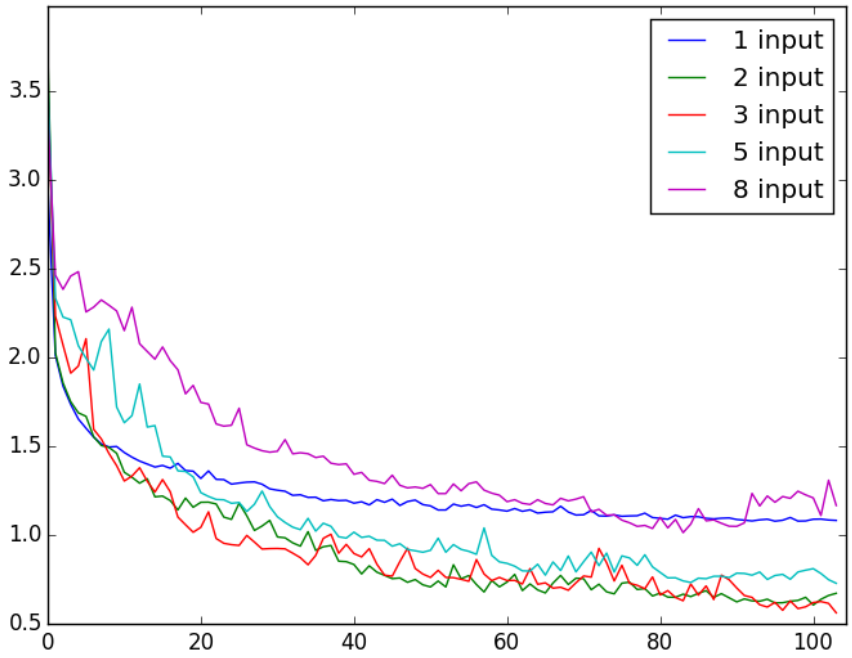


Da notare l'assenza di cicli nello strato nascosto.

2.3 Numero di input

Teoricamente ad una RNN basterebbe una sola nota in input per imparare una canzone intera, visto che ha memoria di quello che ha visto ed in che ordine lo ha visto ma si è reso necessario ampliare l'input ad N[1] note (come viene fatto in [4]) perchè una sola non risultava sufficiente per limiti di hardware. Con più note si usano come input più la probabilità che la rete indovini il target aumenta perchè rendiamo esplicita un pezzo di storia della canzone (le N note precedenti al target). Durante la fase di scrittura del codice è stata riscontrata una relazione tra il numero di note in input e la correttezza dell'output che conferma quanto detto; ovvero se come input si dava una sequenza composta da una sola nota questa riusciva a predirne correttamente solo un'altra come successiva. Quindi quando ci si trovava nella situazione di una doppia scelta ecco che la rete non riusciva più a distinguere le due note. Sempre facendo riferimento allo scritto citato sopra [4] si legge che la rete creata era caratterizzata dall'avere otto note come input.
Il seguente grafico riporta gli errori in fase di validazione su un brano relativamente semplice; indipendentemente dal numero di input il numero di cicli di allenamento e quello di neuroni nello strato nascosto è stato mantenuto costante. Si possono evincere due fattori importanti:

- con più neuroni la rete possiede più cicli di allenamento sono necessari perchè produca buoni risultati. Questo spiega come mai quando si hanno cinque ed otto input gli errori sono più alti;
- se il numero di input è proporzionale alla difficoltà del brano gli errori calano più velocemente;



Non c'è un modo per decidere arbitrariamente il numero di note in input ma bisogna trovare un compromesso tra la quantità di sequenze che si possono/vogliono riconoscere, la complessità del sistema ed il tempo che si vuole dedicare all'allenamento della rete.

2.4 Tipo di rete utilizzata

Le reti neurali sono così composte:

- 1 strato di input con $Nnote * 11$ neuroni;
- 1 strato nascosto con x neuroni[2];
- 1 strato di output con 11 neuroni.

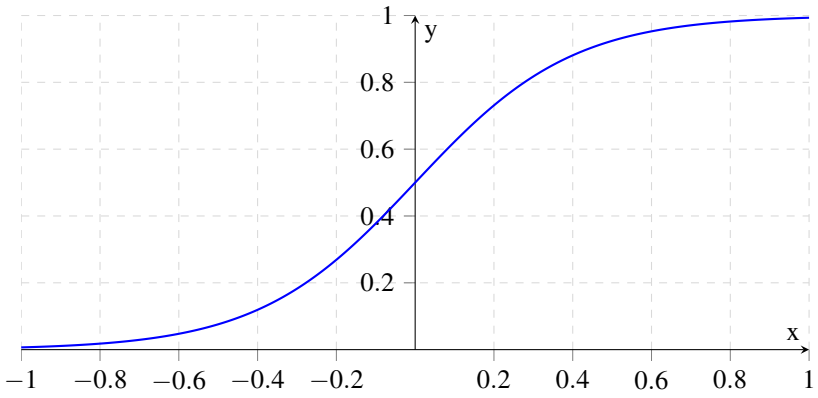
C'è una connessione piena tra lo strato di input e lo strato nascosto, tra lo strato nascosto e lo strato di output e tra lo strato nascosto e se stesso[3]. La differenza tra rete neurale ricorrente e non è che la seconda non presenta la connessione tra lo

[1] Il numero di note in input viene definito appropriatamente in ogni esperimento;
[2] Il numero esatto di neuroni presenti nello strato nascosto viene definito appropriatamente in ogni esperimento;
[3] I pesi che portano dal nodo A al nodo B possono essere diversi di quelli che portano dal nodo B al nodo A;

strato nascosto e se stesso come è già stato possibile vedere.
Le funzioni di attivazione utilizzate all'interno dei neuroni sono di due tipi:

- Sigmoidea;
- LSTM.

La funzione sigmoidea è particolarmente adatta al problema perchè ha dominio in $[0;1]$ e visto che è stata usata una codifica binaria risulta la più congeniale.



A seconda del tipo di rete utilizzata cambiano i tipi di neuroni (per esempio quelli LSTM possono essere utilizzati solo con RNN). Quelli utilizzati nel programma sono:

Tipi di neurone

	FFN	**RNN**
Neurone strato nascosto	Sigmoidea	LSTM
Neurone strato di output	Sigmoidea	Sigmoidea

3 Allenamento della rete

Per allenare le reti è stato scelto l'algoritmo di Back Propagation. L'aggiornamento dei pesi viene fatto in batchlearning, ovvero alla fine di tutte le sessioni, per la rete neurale ricorrente e on-line, cioè alla fine di ogni sessione, per la rete neurale non ricorrente. Sono stati utilizzati diversi valori di learning rate e di momentum, due parametri che influiscono su come la rete è allenata; il primo è una costante che decide la quantità di cambiamento sui vecchi pesi all'interno della rete mentre il secondo comporta una ulteriore adattamento dei pesi in base ai cambiamenti precedenti e permette di avvicinarsi più velocemente al minimo.
Dopo diverse prove sono stati utilizzati valori i seguenti valori:

	FFN	**RNN**
Learning Rate	0.3	0.1
Momentum	0.9	0.3

3.1 Back Propagation

L'algoritmo di Back Propagation è utilizzato con l'apprendimento supervisionato, ovvero dove insieme all'input è noto pure il target della funzione, e permette di modificare i pesi delle connessioni all'interno della rete in modo che venga minimizzata una certa funzione di errore $E(w)$. L'aggiornamento dei pesi viene fatto una volta che è stato prodotto l'output e la quantità è determinata dalla differenza tra lo stesso e il target (l'output reale della funzione). La funzione d'errore da minimizzare in questo esperimento è quella dei minimi quadrati (Mean Square Error, MSE): $E(w) = \frac{1}{2}(out - tar)^2$.
$E(w)$ è una funzione nei pesi, che variano nel tempo a causa degli aggiornamenti, e per minimizzarla si utilizza l'algoritmo di discesa lungo il gradiente. Trovandosi in un punto x_0 viene calcolato il gradiente $\nabla f(x_0)$, cioè la derivata prima in x_0, che dà indicazioni sulla direzione in cui muoversi. Giunti nel nuovo punto x_1, dopo essersi mossi di una certa quantità η, vengono rifatti gli stessi passaggi.
L'algoritmo di BackPropagation può essenzialmente essere diviso in due parti:

- Passo in avanti: l'input viene propagato nella rete per tutti i suoi strati fino giungere all'ultimo, dove viene prodotto l'output;
- Passo all'indietro: viene calcolato l'errore utilizzando $E(w)$ che viene propagato dall'output fino all'input e i pesi vengono aggiornati appropriatamente.

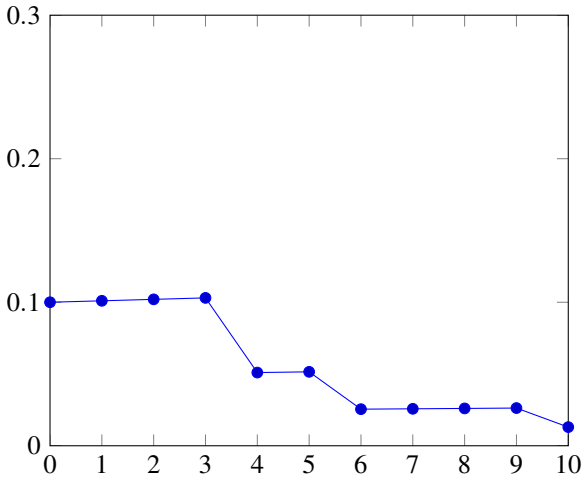
Quando si aggiornano i pesi si tiene conto anche del *learning rate* μ, un valore che influisce sulla velocità e sulla qualità dell'apprendimento: più basso è il valore più lento ma preciso sarà l'allenamento della rete, più altro è il valore più veloce ma impreciso sarà l'allenamento.

3.2 Bold Driver

Per manipolare il valore del learning rate è stata adottata la tecnica Bold Driver [1] che ne cambia la quantità a seconda di come procede l'apprendimento da parte della rete. Si confrontano gli errori al tempo t e $t-1$ e in base ai loro valori viene fatto un cambiamento:

- se l'errore è diminuito si aumenta μ di una quantità pari all'uno per cento del suo valore;
- se l'errore è aumentato si annulla l'ultimo cambiamento fatto e si diminuisce μ del suo 50%.

L'andamento del learning rate in uno degli esperimenti eseguiti, specificamente il primo utilizzando una RNN, è il seguente:



3.3 Momentum

Il *momentum* è un'estensione dell'algoritmo di BackPropagation che, aggiungendo una costante $0 \leq m \leq 1$ all'aggiornamento dei pesi, permette di muoversi più velocemente verso il minimo. Quando il gradiente continua ad andare nella stessa direzione il *momentum* aumenta la grandezza dei passi effettuati. È da notare che, in caso di valori molto alti, va usato un valore di μ molto piccolo, altrimenti si rischia di saltare il minimo perchè i passi effettuati nella discesa lungo il gradiente sono troppo grandi.

3.4 Funzione utilizzata per l'allenamento

Per allenare la rete è stata utilizzata la tecnica di k-fold cross validation [2], dividendo il database iniziale in dieci parti ed utilizzandone nove per l'allenamento e una per la validazione. Il codice riportato è la funzione di allenamento utilizzata per allenare la rete; è stata modificata dal codice originale per adattarsi meglio al programma e per implementare la k-fold cross validation.

Listing 1: Train Until Convergence

```
trainingData = datasetTrain
validationData = datasetTest
self.ds = trainingData
bestweights = self.module.params.copy()
bestverr = self.testOnData(validationData)
trainingErrors = []
validationErrors = [bestverr]
while True:
        trainingErrors.append(self.train())
        validationErrors.append(self.testOnData(validationData))
        if validationErrors[-1] < bestverr:
                # one update is always done
                bestverr = validationErrors[-1]
                bestweights = self.module.params.copy()

        if maxEpochs is not None and epochs >= maxEpochs:
                self.module.params[:] = bestweights
                break
        epochs += 1

        if len(validationErrors) >= continueEpochs * 2:
                old = validationErrors[-continueEpochs * 2:-continueEpochs]
                new = validationErrors[-continueEpochs:]
                if min(new) > max(old):
                        self.module.params[:] = bestweights
                        break
return trainingErrors, validationErrors
```

La funzione salva lo stato dei pesi della rete quando, in fase di validazione, viene ottenuto un errore minore al minimo. Inoltre se l'errore comincia a salire nuovamente controlla di quanto lo fa e, in caso, blocca il train settando i pesi a quelli che davano errore di validazione minore e ritorna la progressione degli errori in fase di allenamento e validazione. Per concludere questa sezione è da notare che è stata valutata l'ipotesi di usare l'algoritmo di Back Propagation Through Time [5], versione particolare di Back Propagation adattata alle RNN, ma non era presente nel pacchetto utilizzato per implementare le reti [3] e la complessità dell'implementazione l'ha resa infattibile.

4 Codifica

Le note, estratte da partiture scritte in formato MusicXML, devono avere una codifica per essere interpretate dalla rete. Le features considerate sono cinque:

- nome della nota;
- alterazione;
- durata;
- punto di valore;
- ottava.

Di queste cinque il nome e l'alterazione sono unite in un'unica codifica visto che sono strettamente correlate tra loro, così come la durata e punto di valore.

4.1 Codifica delle note

Le note nel sistema europeo sono chiamate *la, si, do, re, mi, fa, sol* che nel sistema americano corrispondono a *A, B, C, D, E, F, G*; in questo scritto verrà utilizzato il secondo, perchè più compatto e sintetico. Per codificare tutte le note servono almeno quattro bit, questo perchè oltre alle sette naturali riportate sopra abbiamo anche quelle alterate dai ♯ e dai ♭. Va ricordato però che introducendo entrambe le alterazioni nella codifica si hanno note ridondati dal punto di vista sonoro. Infatti se G ♯ e A ♭ indicano due note diverse, perchè cambia la tonalità in cui vengono usate, il suono che viene prodotto quando sono suonate è però lo stesso. Ecco perchè nella codifica utilizzeremo solo il diesis.

Codifica delle note

A	0000
A♯	0001
B	0010
C	0011
C♯	0100
D	0101
D♯	0110
E	0111
F	1000
F♯	1001
G	1010
G♯	1011

4.2 Codifica della durata

Per la durata della nota il ragionamento è analogo. La durata massima di una nota è $\frac{4}{4}$ (𝅝). Si trovano tutte nella forma $\frac{1}{2^n}$ dove $0 \leq n \leq +\infty$. Convenzionalmente però le prime tre della sequenza sono descritte come $\frac{4}{4}$ (𝅝), $\frac{2}{4}$ (𝅗𝅥), $\frac{1}{4}$ (♩). Le durate che verranno codificate arriveranno fino ad $\frac{1}{64}$ visto che velocità più piccole di questa non vengono mai utilizzate. Come fatto precedentemente verrà assegnata una sequenza di bit ad ogni durata.

Codifica delle durate

$\frac{4}{4}$	000
$\frac{2}{4}$	001
$\frac{1}{4}$	010
$\frac{1}{8}$	011
$\frac{1}{16}$	100
$\frac{1}{32}$	101
$\frac{1}{64}$	110

Un'altra variabile che entra quando si parla di durata è il *punto di valore* (·). Questo strumento aumenta la durata della nota della sua metà e per codificarlo useremo un bit che sarà 0 se non c'è e 1 in caso contrario.

4.3 Codifica dell'ottava

In ciò che va a costituire una nota una parte importante è l'*ottava*. L'ottava costituisce l'altezza della nota, un'offset rispetto a quella più bassa. Si può pensarla come una somma di $n * 12 semitoni$ rispetto alla nota più bassa. Anche qui la codifica che verrà seguita è binaria. Essendoci undici ottave, $C_{-1}; ...; C_9$, dovrebbero essere usati 4 bit per la rappresentazione ma, visto che le ottave C_{-1}, C_0 e C_9 non vengono praticamente mai utilizzate verranno usati tre bit per rappresentare le otto rimanenti.

Codifica delle ottave

C_1	000
C_2	001
C_3	010
C_4	011
C_5	100
C_6	101
C_7	110
C_8	111

4.4 Codifica delle pause

Le pause sono uno strumento musicale molto comune, vengono utilizzate per dire ad uno strumento di non suonare. L'unico dato che portano è quello della durata, visto che non hanno una ottava di riferimento. Verranno codificate con *1111* nel campo di codifica della nota, in quanto denotano una nota inesistente.

4.5 Esempi di codifica

Do di $\frac{4}{4}$, della terza ottava:	01100100000
Do di $\frac{1}{4}$, della terza ottava:	01100100100

4.6 Creazione degli esempi

Per creare un esempio bisogna decidere quante note in input avrà la rete e, in base a quello, verrà costruito. Un esempio per una rete con cinque note in input è:

Esempio

$Nota1$			$Nota2$	$Nota3$	$Nota4$	$Nota5$	$Target$
ottava	nota	durata					

5 Esperimenti

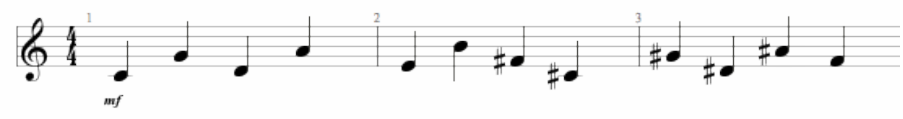
TODO: introduzione dei grafici di train_error e validation_error

5.1 Esperimento 1 - Imparare sequenze di note semplici

Prima di descrivere l'esperimento è necessario dare alcune informazioni sulla rete:

- sono state utilizzate cinque note per comporre l'input;
- il numero di neuroni nell'hidden layer è pari a venti.

La FNN impara sequenze più semplici di note più velocemente e più correttamente rispetto a quella ricorrente. Dalle prove fatte (arpeggi sulla tonalità di do maggio re e scala su due ottave di do maggiore) la FFN è stata capace di ricondursi più velocemente alle sequenze imparate. La RNN invece impara con errori le sequenze, risultato che contribuisce a non saper ricreare correttamente gli esempi.
Nel primo file le sequenze di note erano disposte in una forma a cerchio[4].



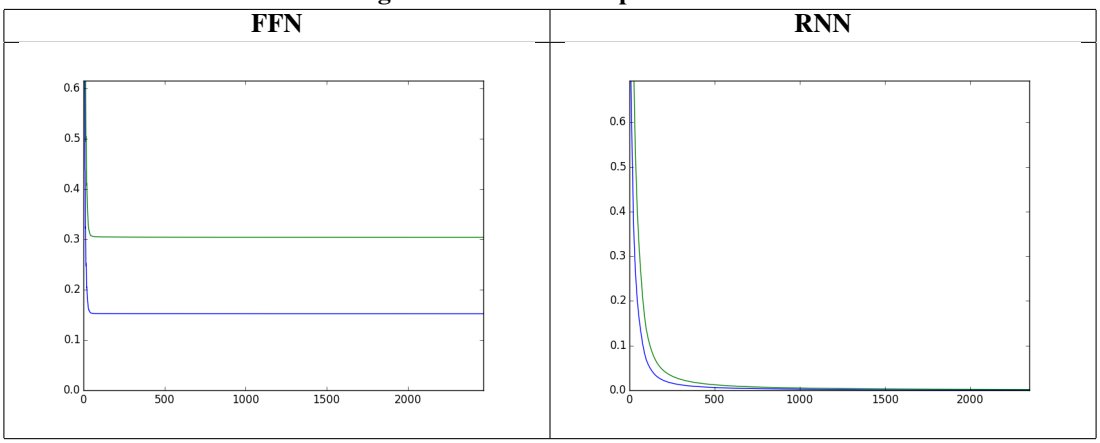
Gli errori medi ottenuti in fase di train e validazione sono:

Errori esempio a cerchio

	FFN	RNN
Allenamento	0.153463	0.003124
Validazione	0.306923	0.027443

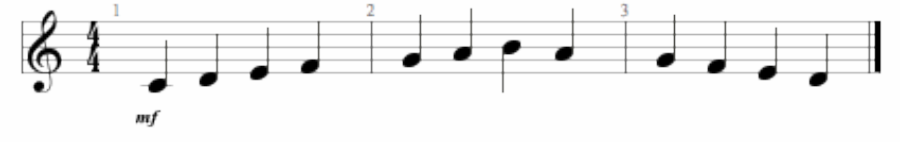
L'andamento durante le tutte le sessioni di allenamento degli errori di addestramento (in blu) e di validazione (in verde) è:

Progressione errori esempio a cerchio



Nonostante gli errori nella RNN fossero più bassi quando si trattava di generare delle note la FFN riusciva a ricreare le stesse esatte sequenze mentre la RNN dopo un certo numero di note generate cominciava a perdere coerenza con quanto già creato. Questo è dovuto al fatto che un errore in una RNN influenza per molto tempo l'output della rete mentre nella FFN l'errore sparisce una volta che la nota esce dall'input. Inoltre si vede chiaramente come la FFN si blocchi in un minimo locale senza riuscire ad uscirne.

Nel secondo file invece le note erano disposte come a formare una figura ad otto.



[4] può non sembra un cerchio ma rispetta il *ciclo delle quinte*, una progressione di note in cui ad ognuna viene fatta seguire la sua quinta naturale

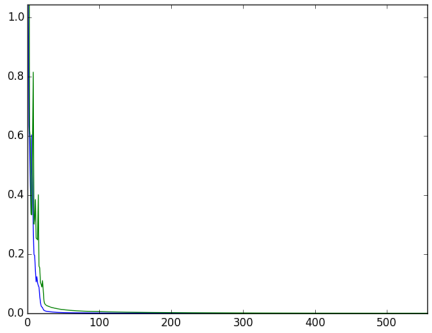
Di seguito sono riportati gli errori medi ottenuti in fase di addestramento, in fase di validazione e l'andamento lungo tutte le sessioni di allenamento degli errori di addestramento (in blu) e di validazione (in verde).
I grafici presentati presentano in dettaglio l'inizio dell'addestramento visto che poi tendono a zero. L'andamento degli errori nell FFN indica che probabilmente i dati sono soggetti a rumore, non si spiega altrimenti il comportamento oscillatorio.

Errori esempio a otto

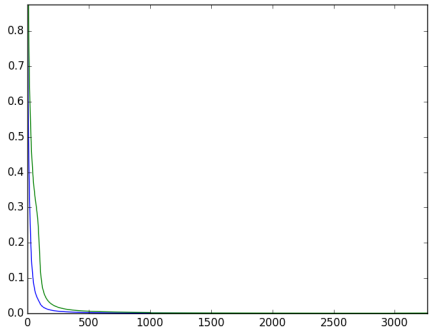
	FFN	RNN
Allenamento	0.000759	0.002412
Validazione	0.001569	0.006690

Progressione errori esempio a otto

FFN



RNN



5.2 Esperimento 2 - Imparare una canzone

Quando si è trattato di fare imparare sequenze di note derivate da una canzone il processo di apprendimento è risultato molto più complesso ed infatti sono stati fatti degli accorgimenti alla rete:

- sono state aumentate le note in input, da cinque ad otto;
- è stato aumentato il numero di neuroni nell'hidden layer, da venti a cinquanta.

Questo perchè le sequenze di note erano maggiori in numero rispetto all'esperimento precedente ma anche perchè erano molto più varie. La RNN si è dimostrata più difficile da allenare, ha richiesto un numero maggiore di epochs, ma le note prodotte erano più esatte, proprio per la capacità di tenere in considerazione la storia delle note (cosa che la FFN non fa), più varie rispetto alla rete neurale non ricorrente che cercava di riprodurre il brano usato per il train ma nient'altro di più.
Il primo brano preso in esame è *Fly Me To The Moon*, il secondo invece è *Kathy's Song*.
Gli errori di train e validazione sono stati:

Errori primo brano

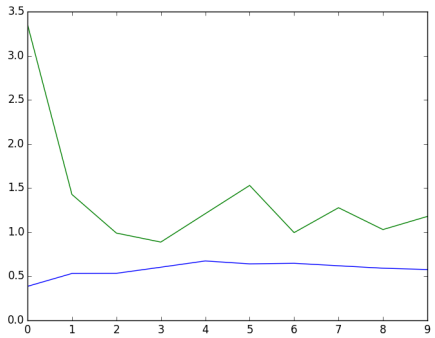
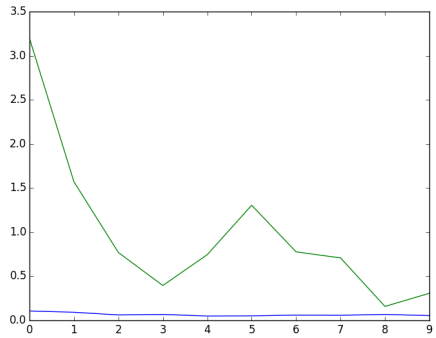
	FFN	RNN
Allenamento	0.580332	0.058844
Validazione	1.387929	0.892960

Errori secondo brano

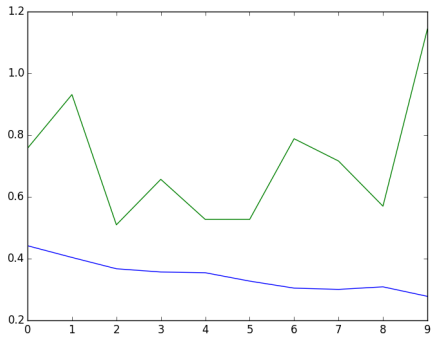
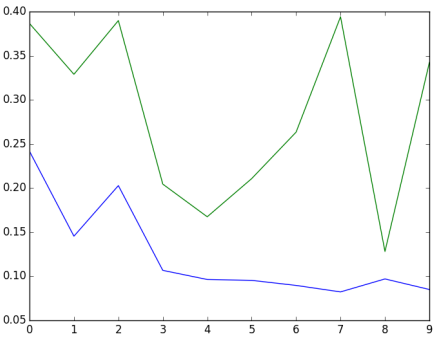
	FFN	RNN
Allenamento	1.745513	0.032295
Validazione	1.740951	0.985849

Questa volta sono riportati, per sinteticità, glli errori medi in fase di allenamento e validazione. Si può vedere chiaramente che la FFN non riesce ad imparare bene le canzoni, infatti i suoi errori sono più alti se confrontati con quelli della RNN.

Errori medi primo brano

FFN	RNN
	

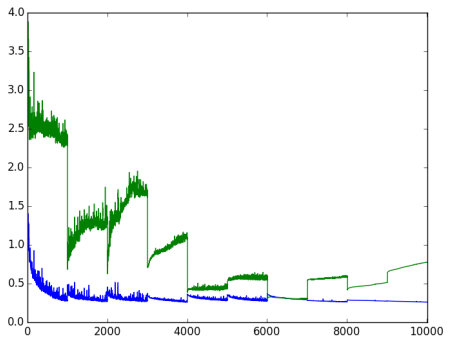
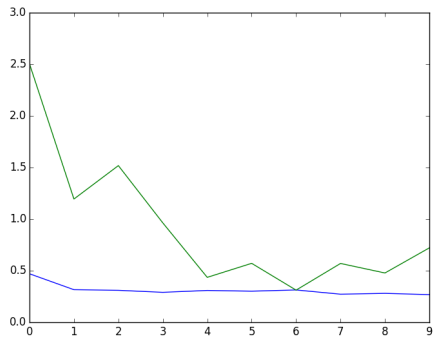
Errori medi secondo brano

FFN	RNN
	

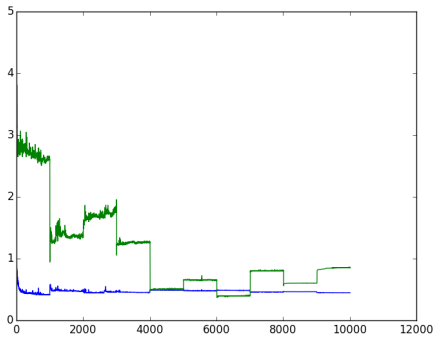
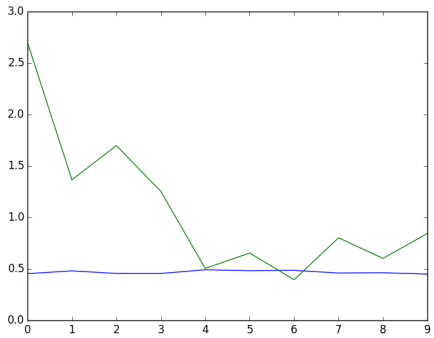
5.3 Esperimento 3 - Imparare più di una canzone

Quando si è trattato di far imparare alle reti più di una canzone sono sorti alcuni problemi perchè queste in qualche modo dimenticavano le sequenze già imparate a favore delle ultime viste; inoltre si riscontravano problemi a livello del codice utilizzato per implementare le reti [3] che non si è riusciti ad identificare ma evidentemente presenti visto l'andamento degli errori di train e validazione.

Errori RNN su più canzoni

Tutti gli errori	Errori medi k-fold
	

Errori FFN

Tutti gli errori	Errori medi k-fold
	

Come si può vedere dai primi grafici l'andamento degli errori non ha un comportamento normale. Se si valutano gli errori medi si evince un andamento più classico ma guardando quelli per ogni sessione si vede che non è normale e, purtroppo, non è stato trovato un motivo a questo andamento.

6 Conclusioni

A conclusione del lavoro fatto si è capito che le Feed Forward Networks possano imparare semplici sequenze di note come scale o arpeggi ma non riescono a dare un significato a ciò che generano, a creare della musica. Le Recurrent Neural Networks invece hanno la capacità di generare note in modo più complesso e vario tenendo conto della storia passata ma, arrivati ad un certo numero di note, perdono il contesto creando si note che si possono definire giuste in quanto appartenenti ad una certa tonalità o di una certa durata ma che all'orecchio umano risultano scollegate l'una dall'altra. Quindi neanche a questo tipo di rete si può chiedere di generare musica in modo completamente esatto.
Inoltre tutto dipende dal numero di note che si decide di dare in input alla rete dato che con poche note non si riescono a predirre bene i target mentre con troppe note crescono considerevolmente i tempi per l'allenamento della rete. Oltre a questo le reti quando generano delle note non riescono a tenere conto del tempo in cui sono stati scritti i brani usati per allenarle.

7 Lavori futuri

Questo progetto è stato sviluppato come lavoro di tesi e tirocinio. Verrà continuato implementando le reti utilizzando un proprio codice e non quello di una libreria esterna ed il goal del progetto sarà creare diverse reti che creino musica di diversi generi.

Contents

Indice

References

[1] Roberto Battiti. Accelerated backpropagation learning: Two optimization methods. *Complex systems*, 3(4):331–342, 1989.

[2] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145, 1995.

[3] Tom Schaul, Justin Bayer, Daan Wierstra, Yi Sun, Martin Felder, Frank Sehnke, Thomas Rückstieß, and Jürgen Schmidhuber. Pybrain. *The Journal of Machine Learning Research*, 11:743–746, 2010.

[4] Peter M Todd. A connectionist approach to algorithmic composition. *Computer Music Journal*, pages 27–43, 1989.

[5] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.