PARALLEL AUTOENCODER

Relazione per l'esame:

Sistemi di Calcolo Paralleli e Distribuiti

Studente:

Amedeo Racanati

Lo scopo di questa relazione è presentare un sistema che implementi un deep autoencoder, capace di essere allenato su più processi. A tal proposito si sfrutta il sistema di comunicazione "Message Passing".

Nel seguito si da una descrizione sommaria delle operazioni matematiche implementate dal sistema, dopodiché si presentano due possibili modalità di parallelizzazione dell'algoritmo. Infine si da qualche cenno sulla implementazione.

DESCRIZIONE DEL DEEP AUTOENCODER

Per il progetto d'esame è stata mia volontà parallelizzare un algoritmo per l'apprendimento di una rete neurale. Il deep autoencoder è una particolare rete neurale utilizzata principalmente per effettuare task come il Semantic Hashing.

Qui di seguito viene descritto l'algoritmo di apprendimento per il deep autoencoder, che è basato sul seguente articolo per la definizione degli iperparametri nonché per i passi da implementare.

Per l'apprendimento si è utilizzata la tecnica cosiddetta greedy layer-wise pre-training, che consiste nell'apprendimento di diverse RBM (Restricted Boltzmann Machine), una per ogni coppia di strati. Dopo questa fase si ottengono dei parametri che sono utilizzati per l'autoencoder. Da ogni RBM si apprende una matrice dei pesi, un vettore per i bias dello strato hidden e un vettore per i bias dello strato visibile.

Figura 1 Esempio di un'immagine del dataset, ricostruita dalla rete neurale addestrata

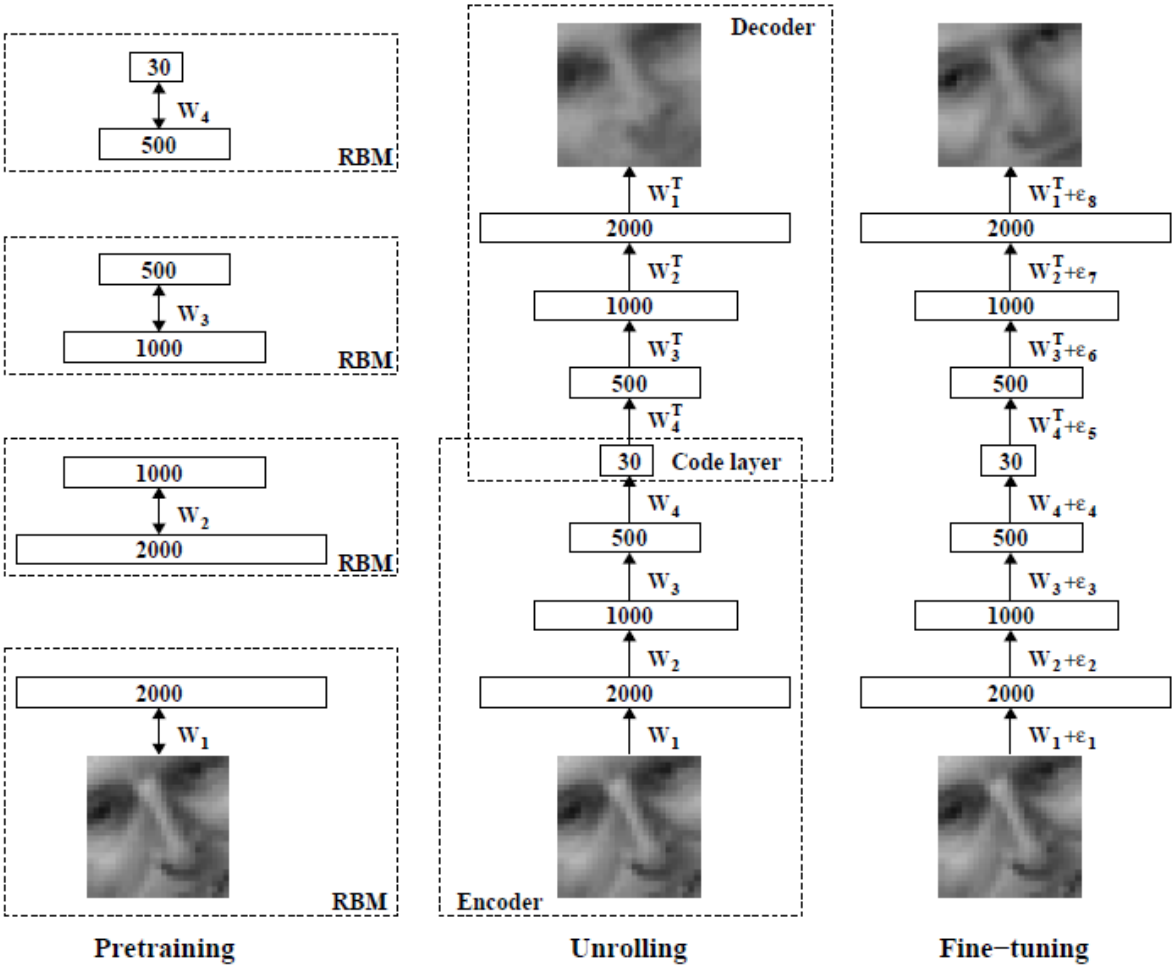

Figura 2 Fasi di allenamento dell'autoencoder (ogni rettangolo rappresenta uno strato)

Per l'apprendimento di una singola RBM si utilizza la tecnica Contrastive Divergence (CD). Dato un insieme di esempi (detto mini-batch) la CD si compone da quattro fasi:

1) Per ogni immagine in scala di grigi, rappresentata da un vettore V di valori decimali, si effettua un campionamento di ogni elemento del vettore H (rappresentante le unità nascoste) secondo la seguente probabilità:

$$p\left(h_j \middle| v\right) = \sigma(b_j + \sum_{i \in vis} v_i * w_{ij})$$

dove b_j è il bias per la j-esima unità nascosta, w è la matrice dei pesi e σ (x) rappresenta la funzione sigmoide.

2) Si ricostruisce il vettore V' delle unità visibili utilizzando la prima funzione qui di seguito per tutte le RBM, ad esclusione della prima RBM che utilizza la seconda funzione (implementa un rumore gaussiano). Si noti che in questo passo non si effettua un campionamento bensì si memorizza per ciascuna unità il valore della probabilità stessa:

$$p(v_i' = 1|\mathbf{h}) = \sigma(b_i + \sum_{j \in hid} h_j w_{ij}), \quad \text{o} \quad v_i' = N(b_i + \sum_{j \in hid} h_j w_{ij}, 1)$$

3) Si calcola il vettore H' per le unità nascoste utilizzando la stessa formula del passo 1, ma invece di utilizzare V si utilizza V'. Se si sta apprendendo la prima RBM si effettua il campionamento al fine di evitare l'overfitting, altrimenti si memorizza il valore della probabilità.

4) Dopo aver eseguito questa operazione per tutti gli esempi di un mini-batch, si calcolano (e applicano) i differenziali per pesi e bias. Si effettua una media dei differenziali da applicare per ciascun esempio:

$$\Delta w_{ij} = < v_i h_j > - < v'_i h'_j > \qquad per\ la\ matrice\ dei\ pesi$$
$$\Delta b_j = < h_j > - < h'_j > \qquad per\ i\ bias\ delle\ unità\ nascoste$$
$$\Delta b_i = < v_i > - < v'_i > \qquad per\ i\ bias\ delle\ unità\ visibili$$

Questa procedura viene effettuata per tutti gli esempi del dataset e ripetuta per 80 epoche. Al termine dell'apprendimento della j-esima RBM, si passa alla (j+1)-esima RBM da apprendere, consci del fatto che il suo input è dato dall'output fornito dalla j-esima RBM.

Una volta apprese tutte le RBM, si effettua il rollup dei parametri. In sostanza vengono utilizzati i parametri appresi per le RBM per la valorizzazione dei parametri dell'autoencoder.

La rete ottenuta col rollup è di tipo feed-forward, in quanto ogni unità in uno strato è frutto della somma pesata delle unità dello strato precedente, alla quale si somma un bias e si applica la funzione sigmoide. In notazione vettoriale, dato l'input V, la matrice dei pesi W e il vettore dei bias B, l'attivazione dello strato successivo H è dato dalla seguente formula:

$$H = \sigma(B + W^T V)$$

La rete viene ulteriormente affinata con la fase di fine-tuning consistente di 5 epoche. Ogni esempio viene processato ottenendo l'immagine ricostruita dall'ultimo strato. Una volta fatto ciò si utilizza il metodo della back-propagation per calcolare i differenziali da applicare per ciascun vettore di pesi e bias, secondo la formula della *delta rule*:

$$\delta_{j_{output}} = y_j * (1 - y_j) * (v_{j_{originale}} - v_{j_{ricostruito}})$$

$$\delta_{j_{hidden}} = y_j * (1 - y_j) * \sum_{K \in strato\ successivo} \delta_k * w_{jk}$$

I parametri si aggiornano di conseguenza:

$$\Delta w_{ij} = \eta * \delta_j * y_i$$

$$\Delta b_j = \eta * \delta_j$$

Questa è l'ultima fase per l'allenamento dell'autoencoder. Nel sistema si è implementata la fase di allenamento come qui descritta.

APPROCCI PER LA PARALLELIZZAZIONE DELLA RETE

Si passa ora ad alcune considerazioni su come poter parallelizzare l'algoritmo descritto nella sezione precedente.

Innanzitutto c'è da distinguere la parte di apprendimento delle RBM da quella del fine-tuning, in quanto consistono in due algoritmi aventi caratteristiche differenti. Inoltre è bene sottolineare che ogni RBM può essere appresa solo dopo che è stata appresa quella ad essa precedente.

Durante l'apprendimento di una RBM, sono due le fasi computazionalmente interessanti: il calcolo dei vettori H, V' e H' per ciascun esempio e il calcolo dei gradienti per ciascun mini-batch.

Primo approccio

Se è vero che un processo può calcolare una parte del vettore H indipendentemente dagli altri processi, è anche vero che esso ha bisogno dell'intero vettore V per poterlo fare. Una prima criticità che si è riscontrata è data dall'elevata connessione che sussiste tra i due strati V e H, in quanto ogni elemento di H dipende da tutti gli elementi di V. Stessa cosa vale per il calcolo di H' e V'.

Un primo approccio consiste nel far si che ogni processo calcoli una porzione dei vettori H, V' e H'. Così facendo, ciascun processo al primo passo calcola l'attivazione delle proprie unità H. Affinché ogni processo possa poi calcolare una parte di V', è necessario che ognuno di essi conosca l'intero vettore H. Pertanto è necessario inviare la propria parte del vettore H a tutti gli altri processi e contemporaneamente ricevere da tutti gli altri processi le relative parti del vettore H. In maniera similare funziona per i passi successivi della CD.

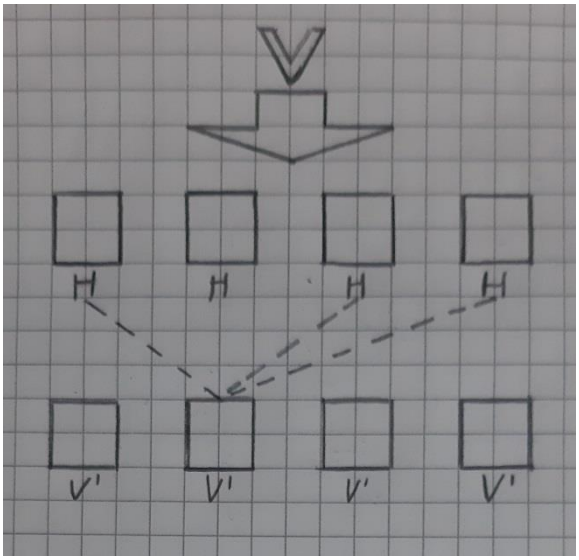

Figura 3 Primo approccio: dati scambiati per il calcolo di V' per il secondo processo

Per poter operare, un processo ha bisogno di conoscere una parte della matrice dei pesi, relativamente alle proprie unità nascoste e visibili. Inoltre ciascun processo ha bisogno di conoscere una parte dei vettori dei bias. Si può evincere come quasi tutta la matrice dei pesi utilizzata da un processo sia utilizzata anche dagli altri processi; per i bias invece ciascun processo non ha bisogno di condividere dati.

Quanto detto implica che durante la fase del calcolo dei gradienti è necessario che i processi comunichino tra loro per scambiarsi i valori dei nuovi pesi calcolati; in alternativa ogni processo può aggiornare la propria matrice dei pesi

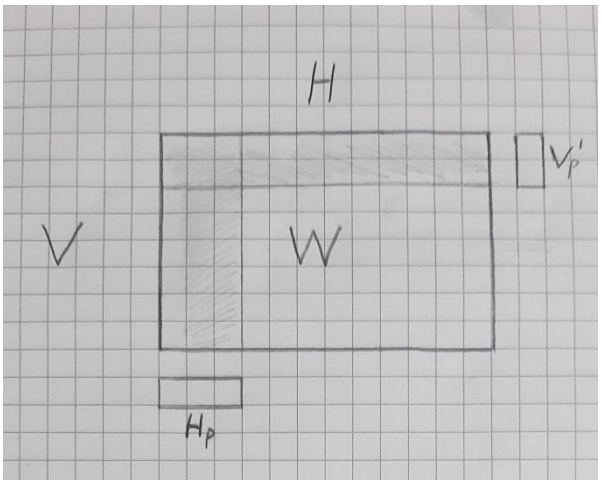

Figura 4 Data una matrice dei pesi VxH, il primo processo ha bisogno solamente della parte evidenziata in figura

indipendentemente dagli altri processi, dato che ogni processo ha ricevuto per intero i vettori V, H, V' e possiede la parte di H' che gli serve per poter calcolare i differenziali (relativamente alla propria matrice dei pesi utilizzata). Nelle stime dei costi qui di seguito si è adottata la prima soluzione, al fine di non dover incrementare il numero di operazioni effettuate nel complesso da parte del sistema. Il calcolo dei gradienti per i bias può essere effettuato senza lo scambio di ulteriori informazioni, dato che i parametri non sono condivisi tra diversi processi.

Questo approccio presenta da un lato una facilità di implementazione, in quanto esiste una sola tipologia di processo e ciascuno di essi deve calcolare una parte dei vettori; ma dall'altro rivela una possibile criticità dovuta all'elevato numero di informazioni da trasmettere ad ogni passo della CD, in quanto ogni processo invia e riceve informazioni da tutti gli altri processi. In presenza di un elevato numero di processi i costi di comunicazione risultano elevati.

Sono state ideate delle soluzioni similari alla precedente che prevedono una suddivisione dei ruoli tra i processi, dove alcuni rappresentano solamente lo strato visibile e altri lo strato nascosto. Non verranno discusse in quanto sussiste ugualmente il problema dell'elevato numero di informazioni da trasmettere, così come la necessità di condividere parte della matrice dei pesi.

Qui di seguito sono enunciati i costi relativi alla trasmissione e alla computazione per la CD di un singolo esempio e quelli relativi all'aggiornamento dei pesi al termine del mini-batch. Si assume che al primo passo della CD i processi possiedano già in memoria l'input V (in quanto è possibile replicare il dataset sui vari processi, se necessario, al fine di evitare ulteriori costi di trasmissione).

Le variabili utilizzate sono:

P = numero processi

V = cardinalità del vettore delle unità visibili

H = card. unità nascoste

H/P = H_p (unità nascoste gestite da un processo)

V/P = V_p (unità visibili gestite da un processo)

CC = costo campionamento

CS = costo funzione sigmoide

Operazione	**Costo per processo**
Calcolo vettore H	$2V * H_p + H_p * (CC + CS)$
Trasmissione vettore H	$H_p * P$
Calcolo vettore V'	$2H * V_p + V_p * CS$
Trasmissione vettore V'	$V_p * P$
Calcolo vettore H'	$2V * H_p + H_p * CS$
Calcolo gradienti	$4 * V * H_p + 2 (H_p + V_p)$
Dopo la fine del minibatch	
Aggiornamento pesi	$V * H_p + H_p + V_p$
Trasmissione pesi	$H_p * V_p * P$

Si può osservare come il numero di processi inficia linearmente sulla quantità di informazioni trasmesse sia per la CD che per la trasmissione dei gradienti.

Secondo approccio

Volendo evitare l'elevato numero di informazioni scambiate ad ogni passo, si è cercato un metodo differente per l'organizzazione dei processi. Analizzando la prima equazione della CD, si può osservare che per calcolare l'argomento della funzione sigmoide è necessario effettuare una somma pesata del vettore V. Se è vero che per calcolare un elemento di H è necessario possedere tutti i valori di V, è anche vero che la sommatoria (che risulta essere l'operazione più onerosa) non deve essere necessariamente implementata su un singolo processo.

L'idea pertanto è quella di definire due tipologie di processi: accumulatore e cella, organizzati topologicamente.

I processi cella hanno lo scopo di calcolare una parte della sommatoria e formano nel loro insieme una griglia di dimensione R x C (nell'immagine ci sono 9 celle). Ciascuna di esse fa uso di una sottomatrice della matrice dei pesi W, al fine di calcolare la sommatoria per H, V' e H'. I processi accumulatori hanno lo scopo di aggregare i risultati ricevuti dai processi cella e di calcolare lo stato di attivazione delle unità visibili o nascoste, facendo uso dei vettori dei bias.

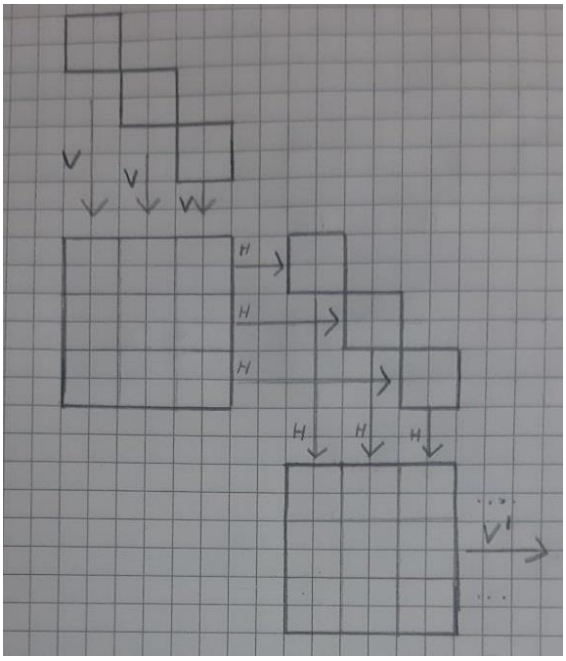

Figura 5 Secondo approccio: scambio di dati tra accumulatori e celle

Considerando il primo passo della CD, ciascun processo accumulatore passa la sua parte di V ai processi cella della corrispondente colonna. Ogni cella di una stessa colonna calcola una parte della sommatoria per differenti unità di H, mentre le celle di una stessa riga calcolano le diverse parti della sommatoria per le stesse unità di H. Dopo che ogni cella ha effettuato il calcolo, i dati vengono trasmessi dalle celle all'accumulatore della corrispondente riga.
L'accumulatore somma tra loro tutti i vettori ricevuti dalle celle (operazione reduce), ci somma il bias ed effettua il campionamento. E' bene notare quindi come i processi siano organizzati logicamente in una griglia bidimensionale, e come gli accumulatori possano aggregare i risultati lungo le colonne (per i vettori V e V') o lungo le righe (per i vettori H e H').

Al secondo passo gli accumulatori inviano il vettore H alle stesse celle dalle quali hanno ricevuto le parti parziali di H, e quest'ultime calcolano di seguito le attivazioni parziali V' che vengono poi trasmesse agli accumulatori lungo le colonne.

La peculiarità di questo procedimento è che i pesi della matrice W non vengono condivisi tra i processi cella, evitando pertanto lo scambio di informazioni durante l'aggiornamento dei pesi. Stessa cosa avviene anche per i bias per i processi accumulatori.

Rispetto al primo approccio, non è necessario che un processo scambi informazioni con tutti gli altri processi. Infatti un processo cella (nella configurazione presente nell'immagine) comunica al più con due accumulatori, mentre un accumulatore comunica con cinque celle. Di questo se ne può tener conto qualora si voglia costruire una rete fisica ad hoc per supportare la comunicazione tra i processi.

Qui di seguito vengono mostrati i costi relativi alla trasmissione e alla computazione per la CD, facendo uso della stessa notazione per il primo approccio, con i seguenti parametri aggiuntivi:

K = numero processi accumulatori

P_r e P_c = numero di processi per una riga / colonna

H/K = H_k (unità nascoste per accumulatore)

V/K = V_k = (unità visibili per accumulatore)

$H/P_r = H_c$ (unità nascoste per cella)

$V/P_c = V_c$ (unità visibili per cella)

Operazione	**Costo per accumulatore**	**Costo per cella**
Trasmissione V a celle	$V_k * P_c$	
Calcolo vettore H	$H_k * (P_r + CC + CS)$	$2 V_c * H_c$
Trasmissione vettore H	$H_k * P_r$	H_c
Calcolo vettore V'	$V_k * (P_c + CS)$	$2 H_c * V_c$
Trasmissione vettore V'	$V_k * P_c$	V_c
Calcolo vettore H'	$H_k * (P_r + CS)$	$2 V_c * H_c$
Trasmissione vettore H'	$H_k * P_r$	H_c
Calcolo gradienti	$2 * (H_k + V_k)$	$4 * H_c * V_c$
Dopo la fine del minibatch		
Aggiornamento pesi	$H_k + V_k$	$H_c * V_c$

Si può osservare come il numero delle celle per riga e per colonna inficia linearmente sul numero di informazioni trasmesse. Questo perché un maggior numero di celle implica che i dati devono essere replicati più volte.

Comparazione approcci

Ora si passa alla comparazione dei costi per i due metodi sopra descritti. Per il secondo approccio si assume $K = P_r = P_c$, ossia avente una disposizione con griglia quadrata e numero di accumulatori pari al numero di celle del lato della griglia. I valori qui di sotto espressi rappresentano il totale delle operazioni effettuate o il totale dei dati trasmessi da parte di tutti i processi.

Operazione	**1° approccio**	**2° approccio**
CD, trasmissione	P (H + V)	K (4H + 3V)
CD, calcoli	10HV + 4H + 3V	10HV + 4H + 3V
Agg. pesi, trasmissione	HV	0
Agg. pesi, calcoli	H + V + HV	H + V + HV

Il primo metodo pecca sicuramente durante la fase di aggiornamento pesi, perché è necessario che i nuovi pesi vengano scambiati tra i vari processi. Il numero di processi influisce solamente sulla trasmissione dei vettori durante la CD.

Per effettuare una comparazione tra i due approcci, bisogna considerare lo stesso numero dei processi. Dato che per il secondo approccio sono necessari (K * K) processi cella e K processi accumulatori, si assume P = K * (K + 1). Questo implica che al crescere lineare di K, il primo metodo consta di un aumento quadratico nella trasmissione dei dati, mentre il secondo metodo consta di un aumento lineare. Con K = 3 (e P = 12) si inizia il secondo metodo inizia a risultare più conveniente.

Potrebbe risultar conveniente il primo metodo solamente quando P è piccolo e il batch size è grande, in quanto l'operazione di scambio dei pesi è meno rilevante e i costi di trasmissione durante la CD diventano equiparabili. Ma in generale il primo approccio risulta sconveniente, soprattutto se ci si trova a trattare vettori V e H di dimensioni elevate che, in maniera congiunta, contribuiscono ad aumentare quadraticamente la grandezza della matrice dei pesi e quindi la relativa trasmissione.

Nei costi si è assunto che la griglia fosse quadrata, ma esistono delle configurazioni ottimali che permettono di ridurre ulteriormente i costi di trasmissione (si veda la tabella dettagliata relativa ai costi del secondo approccio). Dato che il numero di unità visibili solitamente è maggiore rispetto al numero delle unità nascoste, è preferibile disporre di una griglia che contenga più colonne che righe, in maniera tale che il numero di volte che deve essere replicato (e trasmesso) il vettore V sia minore rispetto al numero di volte che deve essere trasmesso H, il quale risulta essere più piccolo di V. Il programma implementato tiene conto di questi costi e imposta l'orientamento della griglia in base alla cardinalità di V e H per ciascuno strato.

Per quanto riguarda il fine-tuning, anche qui è possibile adottare sostanzialmente le due strategie sopra enunciate. La particolarità del fine tuning è che non può essere analizzato più di un esempio per volta senza dover modificare i pesi, quindi il processo di apprendimento risulta più difficilmente parallelizzabile.

E' chiaro che i bias utilizzati dagli accumulatori, così come le matrici dei pesi utilizzate da ciascuna cella, sono le stesse sia per l'allenamento delle RBM che per la fase di fine-tuning. Pertanto non è necessario lo scambio di ulteriori informazioni per il passaggio da una fase all'altra.

Dato che la funzione da parallelizzare è sostanzialmente la stessa (sommatoria con funzione sigmoide), non sono stati trovati altri metodi differenti per disporre i processi rispetto ai due approcci sopra descritti. Anche qui quindi si è deciso di adottare il secondo approccio, facendo si che i processi accumulatori calcolassero l'attivazione delle unità di ciascuno strato, mentre i processi cella calcolano parte della sommatoria.

Per la back-propagation il flusso delle informazioni trasmesse è simile, ma avviene in maniera inversa rispetto alla fase di fine-tuning. Ogni processo accumulatore calcola il proprio δ, lo invia ai processi cella della relativa colonna/riga e infine applica il gradiente per i bias. Ogni processo cella riceve i δ dagli accumulatori, applica il gradiente relativamente alla propria matrice dei pesi e infine provvede a pesare i δ ricevuti per inviarli ai processi accumulatori, che provvederanno a calcolare i δ per lo strato successivo.

NOTE SULL'IMPLEMENTAZIONE

Deciso quindi di adottare il secondo approccio, si è passati allo sviluppo dell'algoritmo. Si considerano un numero K di accumulatori, R righe e C colonne. La configurazione dei processi è stata resa configurabile senza particolari vincoli.

Oltre ai processi sopra enunciati, si è introdotto un processo "master" che si occupa di fornire agli accumulatori i nuovi esempi, prelevandoli da file system. Questo processo gestisce anche l'input da parte dell'utente al fine di avviare un particolare task, avvisando tutti gli altri processi della rete. Per come è stato progettato l'algoritmo, il processo master è l'unico ad accedere su file system per prelevare gli esempi del dataset ed eventualmente salvarne dei nuovi per l'apprendimento delle successive RBM.

L'algoritmo per le RBM è stato ottimizzato al fine di permettere un continuo flusso di dati processati e scambiati. Quel che avviene infatti è che durante la CD vengono processati contemporaneamente due esempi per volta. Quando ad esempio una cella sta calcolando il vettore H da V, contemporaneamente si mette in ascolto per ricevere il vettore V2. Mentre calcola H2 da V2, invia agli accumulatori il vettore H e nel frattempo si mette in ascolto per la ricezione del nuovo vettore H e così via. Anche durante la fase di calcolo dei gradienti, i processi si mettono in ascolto per la ricezione dei nuovi input da processare.

Gli accumulatori fanno in modo di mantenere le celle sempre in esecuzione e nel contempo di ricevere nuovi input dal processo master. Non effettuano operazioni di ricezione/invio dati in modalità asincrona, se non quella di ricezione di nuovi input dal processo master. L'accumulatore rappresenta un collo di bottiglia dal punto di vista della trasmissione dei dati, in quanto deve ricevere e trasmettere dati verso un numero rilevante di processi, anche se la crescita è sub-lineare rispetto al numero dei processi della griglia. Dal punto di vista computazionale gli accumulatori svolgono meno operazioni rispetto alle celle (si veda la tabella dei costi per il secondo approccio).

Grazie alla configurabilità del sistema è possibile incrementare o decrementare il numero degli accumulatori qualora ce ne fosse bisogno, e riservare loro maggior potenza di banda piuttosto che di calcolo. Particolari architetture di comunicazione ad hoc potrebbero velocizzare il processo di trasmissione dei dati, sfruttando l'idea della griglia che sussiste nella comunicazione.

L'algoritmo di fine-tuning è similare a quello dell'apprendimento delle RBM, con la sola differenza che è necessario calcolare lo stato di attivazione di tutti gli strati, utilizzando quindi tutte le matrici dei pesi e dei bias apprese in precedenza. L'unica miglioria apportata è stata quella che durante la fase di back-propagation, mentre i processi accumulatori calcolano i gradienti per i propri bias, inviano i delta ai processi cella. I processi cella provvedono a spedire i delta pesati ai processi accumulatori, e nel frattempo applicano i gradienti ai propri pesi per poi ricevere i nuovi delta per lo strato successivo.

E' bene notare che il numero di processi accumulatori e celle è fisso durante l'intero apprendimento della rete neurale.

Al seguente link è presente il codice del progetto. Il progetto è stato sviluppato utilizzando il linguaggio C++ (versione 11). Per le primitive di Message Passing si è utilizzato MPICH. Per la lettura, scrittura e visualizzazione delle immagini del dataset si è sfruttata la libreria OpenCV.

Il codice è strutturato in maniera tale che ogni tipologia di processo erediti da una classe base e che implementi le funzioni di allenamento e ricostruzione degli input in maniera differente a seconda del ruolo che svolge.

Il programma permette l'esecuzione seriale o parallela dell'algoritmo. E' possibile specificare dei valori di configurazione in merito alla rete neurale:

- Numero di accumulatori
- Grandezza della griglia
- Numero e grandezza di ciascuno strato
- Epoche per l'apprendimento delle RBM e del fine tuning
- Grandezza del mini-batch
- Percorso del dataset delle immagini e numero di esempi da trattare

Nella cartella *"autoencoder_pars"* vengono salvati i parametri della rete (ogni processo salva i dati su un proprio file).

La cartella *"mnist_chinese"* contiene le immagini del dataset.

Figura 6 Cartelle e file del progetto

Un esempio di comando per lanciare l'applicazione è il seguente:

mpirun -np 11 ./Release/Parallel_autoencoder --n_samples:100 --k_acc:2 --g_cols:2 --g_rows:4

Questo comando avvia 11 processi che tratteranno al più 100 esempi del dataset. Ci saranno due accumulatori e 2x4 processi griglia. La grandezza degli strati è di default la stessa del link utilizzato come riferimento per la configurazione della rete neurale, ossia 4096 x 8192 x 4096 x 2048 x 1024 x 512 x 256 x 128 x 64 x 32.

RISULTATI

Lo scopo di questa sezione è di confrontare i tempi di esecuzione tra la versione a singolo processo (seriale) e quella parallela.

Sono state utilizzate le macchine paradigm1, paradigm2 e paradigm4 e sono state create tre configurazioni parallele:

1. 8 processi su paradigm1: un processo master, 1 accumulatore e 2x3 celle;
2. 16 processi su paradigm1 e 2: un processo master, 3 accumulatori e 3x4 celle;
3. 24 processi su paradigm 1, 2 e 4: un processo master, 5 accumulatori e 3x6 celle.

Per queste 4 versioni (tre parallele e una seriale), si è creata una configurazione con grandezza del minibatch pari a 2, e un'altra con grandezza pari a 20. Una grandezza più piccola del minibatch implica un numero maggiore di volte che è necessario aggiornare i pesi.

Si analizzano i tempi d'esecuzione per la fase di apprendimento, che consta di 80 epoche per l'apprendimento di ciascuna RBM e di 5 epoche per il fine-tuning. Le operazioni di I/O consistono nel prelevare gli esempi del dataset e di salvarne dei nuovi per le RBM da allenare.

Si sono misurati due indicatori:

- Strong scaling: a parità del numero di esempi del dataset, si analizza lo speedup secondo la legge di Amdahl;
- Weak scaling: si analizza lo speedup secondo la legge di Gustafson, facendo sì che il numero di esempi del dataset da analizzare sia pari a 20 moltiplicato il numero di processi.

Nel file excel "Risultati.xlsx" sono presenti i tempi di esecuzione in maniera più dettagliata.

Dato che la grandezza del minibatch non ha inciso sullo speedup, si presentano qui di seguito i risultati per il minibatch pari a 20.

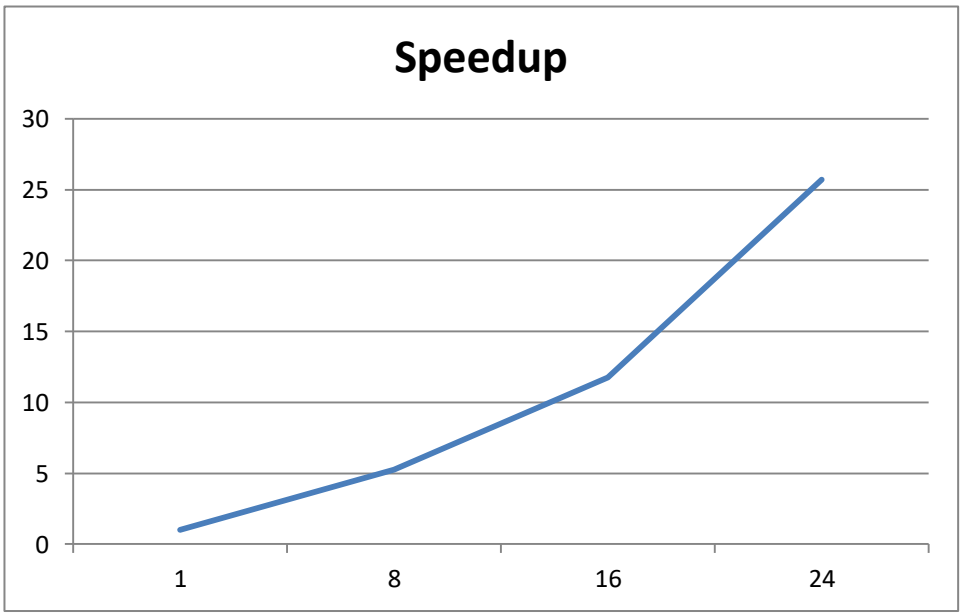

Figura 7 Numero di processi per speedup secondo la legge di Amdahl

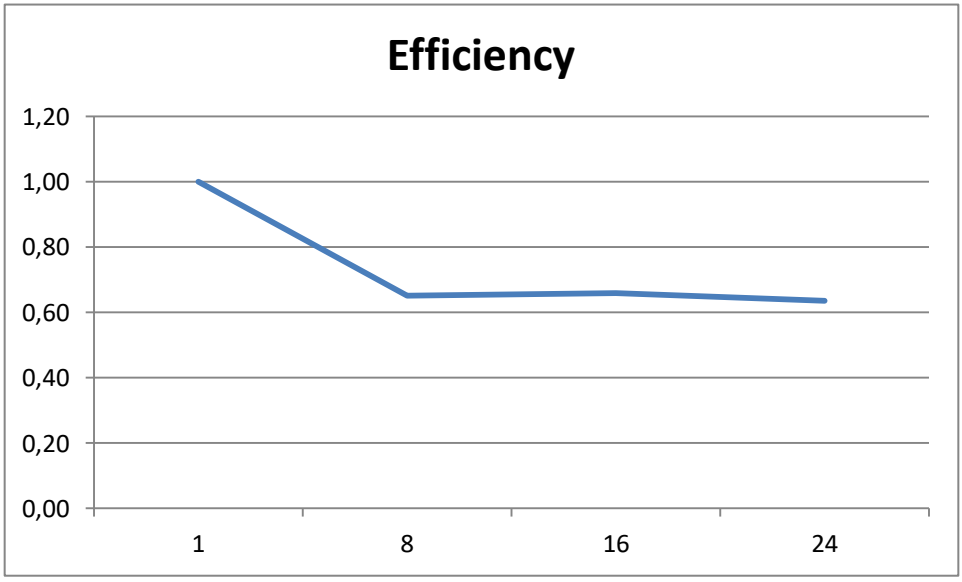

Figura 8 Numero di processi per efficiency secondo la legge di Gustafson

Non potendo testare il sistema su un numero elevato di nodi, bisogna fare le proprie considerazioni su questi dati tenendo conto che comunque la varianza dei risultati è pressoché nulla. C'è da considerare che i tempi includono anche le operazioni di I/O su file system.

Si può notare come lo speedup secondo la legge di Amdahl possiede una curva differente rispetto alla normalità dei casi, e come sia particolarmente elevato con 24 processi (25x). Ciò è dato principalmente dalla configurazione della griglia 3x6, la quale può essere orientata per ogni strato in maniera ottimale per ridurre il flusso dei dati trasmessi. Quando la griglia tende ad essere "più quadrata", la capacità di ottimizzare l'algoritmo diventa inferiore e di conseguenza anche lo speedup.

Lo speedup secondo Gustafson si mantiene costante sullo 0,67. Ciò significa che al crescere del numero di esempi trattati e del numero di processi, l'algoritmo si mantiene stabile. Questi tempi potrebbero risultare in contrasto con quelli ottenuti seguendo la legge di Amdahl. C'è però da considerare il fatto che un numero maggiore di esempi implica maggiori operazioni di I/O. Dato che queste operazioni non sono state parallelizzate, esse incidono particolarmente sui tempi di esecuzione.

Questi risultati mostrano che è possibile sviluppare degli algoritmi che migliorino in maniera considerevole i tempi di esecuzione per l'apprendimento di una rete neurale. D'altra parte sarebbe preferibile effettuare dei test su un numero elevato di processi per comprendere a pieno l'andamento della curva dello speedup.