Università di Pisa

Facoltà di Scienze Matematiche Fisiche e Naturali
Corso di Laurea Magistrale in Informatica

Tesi di Laurea

Modelli di Reti Neurali per l'Apprendimento In-Graph e On-Graph

Candidato

Ilaria Ceppa

Relatore

Alessio Micheli

Anno Accademico 2013/2014

Abstract

La tesi affronta il problema dell'apprendimento automatico in domini strutturati, con un focus su funzioni di trasduzione definite su grafi solo parzialmente etichettati. Questa classe di problemi, che prende il nome di *in-graph* o *within-network learning*, presenta delle caratteristiche peculiari che la differenziano nettamente dai problemi di *on-graph* o *between graphs learning*, nei quali la funzione di trasduzione viene appresa tramite allenamento del modello su un insieme di grafi completamente etichettati. Un problema di in-graph learning invece è definito su una rete di entità (*networked data*), delle quali solo una parte è etichettata e può essere utilizzata per l'apprendimento: i modelli per la risoluzione di task di questo tipo devono essere in grado di gestire l'informazione strutturale dei dati e contemporaneamente l'interconnessione tra entità etichettate e non, sfruttandola al meglio per il raggiungimento dell'obiettivo di apprendimento.

Scopo della tesi è sviluppare nuovi modelli per l'in-graph learning. Il punto di partenza per il raggiungimento di questo obiettivo è un modello neurale, *Neural Network for Graphs*, in grado di apprendere funzioni di trasduzione contestuale tramite un meccanismo di codifica adattiva del contesto di ogni vertice del grafo. Tramite il confronto con modelli allo stato dell'arte per l'apprendimento in-graph appartenenti al mondo dello Statistical Relational Learning, il modello neurale viene esteso al nuovo problema, integrando il potere e la flessibilità della codifica contestuale e adattiva con le tecniche di inferenza proprie dei modelli relazionali. I nuovi modelli neurali proposti utilizzano un approccio costruttivo, grazie al quale è possibile trattare unitamente i diversi tipi di informazione presenti nella rete (locale, relazionale e contestuale), evitando il ricorso ai meccanismi iterativi di inferenza collettiva tipici dei modelli relazionali per problemi in-graph. Inoltre, i modelli sviluppati permettono di definire un metodo generale d'apprendimento, applicabile per la risoluzione di problemi sia in-graph che on-graph. Attraverso il confronto sperimentale con modelli relazionali allo stato dell'arte per il within-network learning è stato possibile valutare l'effettiva validità e i vantaggi dell'approccio proposto: applicati ad un problema di *webspam detection* di una competizione in ECML/PKDD, i modelli neurali superano le performance predittive ottenute dai modelli relazionali, grazie all'azione combinata della codifica contestuale e dell'informazione relazionale.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Il problema del trattamento di domini strutturati è diventato negli ultimi anni un tema centrale di sviluppo della ricerca nell'ambito dell'apprendimento automatico.

Un dominio si dice strutturato se tra le entità che ne fanno parte sono definite delle relazioni, che definiscono appunto una rete di collegamenti tra gli oggetti appartenenti all'insieme dei dati. Rispetto ad un dataset non strutturato, in cui ogni entità è indipendente dalle altre, dati di questo tipo presentano una notevole ricchezza aggiuntiva, data dall'informazione che è possibile ricavare dalle relazioni che intercorrono tra le entità. La struttura definita sui dati dà accesso ad una ulteriore fonte di informazione, che si somma a quella data dalle proprietà locali dell'entità, e che può rivelarsi fondamentale per un problema d'apprendimento definito su quel dominio: è quindi necessario avere a disposizione dei modelli in grado di incorporare questa informazione relazionale all'interno del processo d'apprendimento, e di utilizzarla al meglio.

Sono molti i problemi reali, in diversi campi, in cui il dominio dei dati è naturalmente modellabile come una rete di entità in relazione tra loro, ovvero come *networked data*. Un primo esempio è costituito dall'analisi e classificazione di collezioni di documenti tra loro collegati. I documenti in questione possono essere articoli scientifici, che tramite citazioni sono connessi l'uno all'altro , e che si vogliono classificare in base all'argomento di ricerca [1]–[5], oppure brevetti [6], connessi sempre in base ai riferimenti reciproci, oppure ancora, in generale, *hypertext documents*: un qualsiasi insieme di documenti che contengono al loro interno *link* ad

altri documenti della stessa collezione. Un classico esempio di documenti di questo tipo sono le pagine web, che tramite link da e verso altre pagine sono collegate a formare un unico grafo; i problemi di classificazione che è possibile immaginare su una rete di pagine web sono diversi: ad esempio lo scopo potrebbe essere di voler categorizzare le pagine web in base all'argomento che trattano [5], [7], oppure di voler individuare pagine di *spam* [8], [9], ovvero pagine che cercano di manipolare in maniera scorretta i risultati dei motori di ricerca per riuscire ad attirare un maggior numero di visite. Un ulteriore esempio di collezioni di documenti interconnessi, di diverso genere rispetto alle precedenti, può essere invece una collezione di film [5], [7], [10], [11]: i film possono essere visti come collegati in una rete, se ad esempio si considerano due film come connessi se condividono lo stesso attore, o lo stesso regista o qualche altra proprietà; a differenza dei precedenti esempi, in cui i collegamenti sono parte dei dati stessi (come le citazioni in un articolo o i link in una pagina web), in una collezione di questo tipo la relazione tra le entità viene definita a posteriori, scegliendo un criterio con cui collegare coppie di elementi del dataset. Tramite un ragionamento del genere, un qualsiasi insieme di dati, tra i cui elementi sia possibile definire una funzione di somiglianza traducibile in una serie di collegamenti tra entità, può essere interpretabile come un networked dataset: in questo modo, un modello d'apprendimento in grado di gestire e sfruttare contemporaneamente le proprietà locali e relazionali dei dati, può risolvere il task di learning tenendo conto anche della somiglianza tra coppie di elementi del dataset. Se la rete dei dati è di questo tipo, di solito soddisfa (per costruzione) l'ipotesi di omofilia, secondo la quale la probabilità che esista un collegamento tra due entità è molto più alta se queste sono simili; in particolare, si parla di omofilia anche nel caso in cui è il target stesso dell'apprendimento a determinare i collegamenti tra le entità della rete.

Un altro tipo di networked data nel mondo reale è dato dalle reti sociali, o *social networks*: reti di persone, connesse tra loro da una serie di diverse relazioni. Una rete sociale è ad esempio data dagli stessi siti di social network, come Facebook, Twitter o altri, in cui le persone iscritte sono collegate tramite le relazioni di amicizia, appartenenza allo stesso gruppo o cerchia, ecc. Un altro esempio di rete sociale sono le *customer network* [12], [13], reti costruite sui clienti di una attività in base a quanto sono tra loro simili, ovvero in base a quanti articoli in comune hanno

acquistato; su reti di questo genere sono possibili diversi task, come ad esempio l'individuazione di una certa categoria di persone, oppure la classificazione degli utenti in gruppi (ad esempio, in base ai loro interessi), oppure ancora l'individuazione degli utenti cui far visualizzare una certa pubblicità e la costruzione di sistemi di raccomandazione. Un ulteriore esempio di rete sociale lo si può trovare nel campo della cosiddetta *fraud detection* [14]–[16], dove le persone vengono collegate in un grafo in base alle comunicazioni che intercorrono tra loro, e si vogliono distinguere tra queste gli utenti "fraudolenti" da quelli "legittimi".

Ancora è possibile trovare altri esempi di domini strutturati anche in alcune applicazioni dell'elaborazione del linguaggio naturale [17], [18]: dall'analisi di una frase, è possibile ricavare una rappresentazione della sua struttura tramite un albero di parsing, e un problema di interesse su queste strutture è ad esempio il *part-of-speech tagging*, ovvero l'etichettatura di ogni parola della frase con la corrispondente categoria grammaticale.

Infine, un importante dominio applicativo in cui è possibile trovare dati strutturati è quello della chimica e della biologia: molecole e composti chimici sono naturalmente descritti da una struttura a grafo, in cui gli atomi o le molecole sono collegati in base ai loro legami chimici. L'analisi di queste strutture prende il nome di *quantitative structure-property/activity relationship* (QSPR/QSAR, analisi di relazione quantitativa struttura-proprietà/attività) e consiste nell'individuazione di importanti proprietà della molecola, come la sua tossicità, che sono in una qualche misura collegate alla struttura della stessa [19]–[21].

In generale, è possibile definire il problema dell'apprendimento nell'ambito dei domini strutturati (ad esempio, il dominio dei grafi) come la ricerca di una *funzione di trasduzione* che leghi oggetti del dominio ad oggetti del dominio di output [22]; quest'ultimo può essere un dominio non strutturato, nel qual caso si può parlare di trasduzione *scalare* o *supersource*, o un dominio strutturato a sua volta: in questo caso la funzione di trasduzione è definita tra coppie di grafi, e nel caso particolare in cui essa sia definita solo tra coppie di grafi con la stessa "struttura" (ma eventualmente con una differente etichettatura dei vertici), si parla di trasduzione *input-output isomorfa*.

Possiamo dividere i problemi di classificazione nell'ambito dei domini strutturati, per gli scopi della tesi, in due grandi categorie [16]: problemi di *on-graph learning*

e problemi di *in-graph learning*. In un problema di on-graph learning, o apprendimento tra grafi o *across-network learning*, l'obiettivo è apprendere la funzione che lega i grafi su cui il problema è definito al loro target, utilizzando a questo scopo un insieme di grafi per i quali è noto il valore di questa funzione. Quanto appreso viene poi utilizzato per classificare nuovi grafi: ad esempio, tra i problemi descritti prima, un task di on-graph learning definito nell'ambito della chemioinformatica consiste nella classificazione di molecole in base alla loro tossicità: dalle molecole per le quali questa proprietà è conosciuta, si vuole apprendere il legame generale che esiste tra la struttura di una molecola e la sua tossicità, per poi applicare quanto appreso per la classificazione di nuovi composti chimici. Un problema come quello appena descritto consiste nell'individuare una funzione di trasduzione scalare, che leghi come detto sopra ogni grafo ad un valore reale (problema di regressione) o intero (problema di classificazione). Anche l'individuazione di funzioni di trasduzione input-output isomorfe può costituire un problema di on-graph learning: in generale, una funzione di questo tipo mette in relazione coppie di grafi con la stessa struttura (ovvero, isomorfi); un caso particolare è quello in cui la funzione associa ad ogni grafo del dominio un grafo a lui isomorfo, con una differente etichettatura dei vertici. Se, come nel caso delle trasduzioni scalari, la funzione deve essere appresa su un insieme di grafi per i quali si ha a disposizione la nuova etichettatura dei vertici (ovvero grafi in cui per ogni vertice è noto il target ad esso associato), per poi essere applicata a nuovi grafi, non etichettati, il problema rientra nella categoria dell'apprendimento tra grafi.

In un problema di in-graph learning, o *within-network learning* invece, l'apprendimento non avviene più tra grafi, ma all'interno di un unico grafo: le entità su cui è definita la funzione di trasduzione sono i vertici del grafo, e anche in questo caso si può parlare di funzioni di trasduzione input-output isomorfe. La particolarità di questo tipo di problema però, che lo differenzia dal precedente, consiste nel fatto che vertici etichettati, su cui è possibile fare apprendimento, e vertici non etichettati, di cui si vuole predirre il target, coesistono nella stessa rete, e sono tra loro collegati tramite relazioni: la funzione di trasduzione viene appresa tramite i vertici etichettati, e poi utilizzata per stimare il target per i vertici dello stesso grafo per i quali l'output della funzione non è noto. L'in-graph learning è in questo senso un problema di *apprendimento semisupervisionato*, in quanto, essendo il

grafo parzialmente etichettato, durante il learning sia le entità etichettate che le entità non etichettate sono utilizzabili dal modello, per quanto in maniera diversa: le prime possono essere usate per l'apprendimento della funzione di trasduzione, per confrontare l'output del modello con il target reale e modificarlo di conseguenza, mentre quelle non etichettate portano informazioni aggiuntive sulle entità cui sono collegate, di cui formano il contesto. Tra quelli citati all'inizio del capitolo, sono esempi di problemi di questo tipo la classificazione di pagine web o articoli scientifici, o la classificazione degli utenti di una rete sociale: in tutti questi casi la funzione di interesse è definita sulle entità del dominio, che sono connesse tra loro a formare un'unica rete in cui entità etichettate e non sono interconnesse.

I dataset di un problema di in-graph learning presentano quindi caratteristiche peculiari rispetto a quelli discussi precedentemente, e per questo motivo richiedono lo sviluppo di modelli e tecniche in grado di sfruttare al meglio queste particolarità [16]. L'intrinseca interconnessione tra vertici con e senza target rappresenta contemporaneamente la difficoltà e il vantaggio di questo tipo di dati. Da una parte l'algoritmo d'apprendimento deve tenere conto della presenza di connessioni tra vertici etichettati e non, che non rende tra l'altro possibile una rigorosa suddivisione dei dati da utilizzare per il learning, a meno di trascurare in parte o del tutto l'informazione relazionale: i vertici non etichettati costituiscono, tanto quanto quelli etichettati, il contesto di un vertice, ed eliminarli dal dataset durante il processo d'apprendimento può risultare nella rimozione di importanti informazioni sul contesto. D'altra parte però i vertici etichettati possono svolgere una duplice funzione in questo tipo di dataset: durante la fase di apprendimento vengono utilizzati per il learning della funzione di trasduzione, ma durante la fase di predizione essi vanno a costituire il contesto dei vertici con target sconosciuto. Come conseguenza di ciò si ha che, a differenza di quanto succede con i task tra grafi, al momento della classificazione dei vertici non etichettati si ha a disposizione l'informazione aggiuntiva sul target dei vertici che ne costituiscono il contesto, informazione che può fornire un notevole aiuto al processo di inferenza.

Il problema dell'apprendimento tra grafi (on-graph learning) è stato ed è tuttora una tematica molto importante per la ricerca nell'ambito dell'apprendimento automatico. L'idea alla base di quasi tutti gli approcci al problema dell'introduzione dell'informazione strutturale dei dati nel processo di apprendimento consiste

nell'utilizzo di una qualche codifica numerica, o *encoding*, dell'informazione sul contesto di ogni elemento. L'approccio più semplice consiste nell'utilizzare un modello d'apprendimento definito per dati non strutturati (*flat*) e nel calcolare a priori una codifica fissa, stabilita in base al dominio d'applicazione: ad esempio, per l'analisi QSPR/QSAR si può ricorrere all'uso di un certo numero di indici topologici, per racchiudere in una rappresentazione vettoriale l'informazione topologica sulla struttura della molecola. Questo tipo di approccio presenta alcuni problemi: la scelta della funzione di codifica è fondamentale per la buona riuscita del learning, ed è necessario possedere una conoscenza approfondita del dominio in questione, per essere in grado di sceglierne una appropriata. Inoltre il fatto che la scelta venga fatta a priori, separatemente dal processo di apprendimento, rende molto difficile la selezione di una codifica per il task specifico che si sta cercando di risolvere: anche supponendo di avere una perfetta comprensione del dominio dei dati, non c'è garanzia che la funzione di encoding scelta sia la migliore per il problema specifico. Un approccio per certi versi simile è quello dei kernel per grafi [21], [23]–[25]: un kernel definisce un prodotto scalare tra coppie di grafi (o vertici di uno stesso grafo) in uno spazio vettoriale, sulla base degli attributi dei vertici che li compongono e in base alla loro topologia; anche questo approccio presenta gli stessi svantaggi della precedente, in quanto la scelta del kernel, che determina totalmente la buona riuscita del successivo processo di apprendimento, è anche qui separata dal learning. Una soluzione ai problemi dovuti alla separazione del processo di codifica dal processo di apprendimento è nata invece nell'ambito delle reti neurali [26]–[29]: i modelli neurali per il trattamento di domini strutturati riescono ad incorporare il processo di codifica all'interno dell'apprendimento stesso, rendendolo così adattivo, ovvero dipendente dal task corrente, e automatico, eliminando così la necessità di approfondite conoscenze pregresse sul dominio.

Il problema dell'in-graph learning e i dataset relazionali sono invece stati maggiormente studiati dal settore dell'apprendimento automatico che prende il nome di Statistical Relational Learning (SRL), un campo nel quale vengono incorporati concetti della statistica e della probabilità con tecniche di rappresentazione della struttura dei dati relazionali, con lo scopo di trattare anche questa particolare tipologia di oggetti, in cui entità etichettate e entità non etichettate sono interconnesse tra loro in base a relazioni [5], [16].

In questo campo il problema di classificazione descritto prima si traduce nel calcolo della distribuzione di probabilità marginale di appartenenza ad una determinata classe per ogni vertice, condizionata dalla conoscenza che si ha sulle classi associate agli altri vertici della rete. Approcci "diretti" all'inferenza probabilistica, che tengono conto di tutte le possibili configurazioni di tutte le variabili presenti nel grafo, in quest'ambito risultano intrattabili, anche per reti con un numero di vertici inferiore alle centinaia: per questo motivo i modelli allo stato dell'arte per questo task utilizzano tecniche euristiche per approssimare il processo dell'inferenza.

Un modello dello SRL per networked data può essere schematizzato in tre componenti principali [5]. La prima componente è il classificatore locale (LC), che può essere un qualsiasi classificatore per dati flat, il cui compito è di inferire la classe di ciascun vertice utilizzandone solo gli attributi locali: la predizione fatta da questa componente viene poi utilizzata come base per il processo d'inferenza. La seconda componente è il classificatore relazionale (RC), che definisce come calcolare la classe di un vertice in base alle proprietà del vertice stesso e al suo "vicinato"; quali vertici appartengano al vicinato del vertice dipende dal particolare classificatore relazionale, ma la maggior parte dei modelli fanno uso dell'assunzione di Markov del prim'ordine: per rendere il problema computazionalmente trattabile, si assume cioè che la classe del vertice che si vuole inferire dipenda solamente dal vertice stesso e dai suoi vicini diretti, ovvero collegati al vertice da un arco. L'ultima componente infine è l'algoritmo di inferenza collettiva (IC), che si occupa di risolvere il problema delle dipendenze nel calcolo dell'inferenza per vertici interconnessi tra loro e della gestione dell'effetto delle influenze reciproche tra vertici diversi, anche non direttamente collegati: la soluzione consiste solitamente nell'utilizzo di algoritmi iterativi, che approssimano ad ogni passo la distribuzione di probabilità dei vertici non etichettati del grafo utilizzando la distribuzione inferita al passo precedente.

L'obiettivo della tesi è di analizzare nel dettaglio i diversi approcci ai due problemi dell'on-graph e dell'in-graph learning, con lo scopo di sviluppare nuovi modelli per l'apprendimento in-graph che combinino i punti di forza di entrambi. Punto di partenza del lavoro effettuato sarà un modello per l'apprendimento tra grafi, *Neural Network for Graphs* (NN4G [29]), una rete neurale in grado di apprendere trasduzioni contestuali tramite l'uso di una codifica adattiva. Tramite poi l'analisi di modelli dello Statistical Relational Learning e il loro confronto con NN4G, si

estenderà il modello neurale al task dell'apprendimento in-graph, combinando la flessibilità della codifica adattiva di NN4G alle tecniche proprie dei modelli relazionali, come ad esempio l'inferenza collettiva. Il risultato è un insieme di modelli che permette di trattare sia problemi d'apprendimento on-graph che in-graph. La capacità d'apprendimento dei nuovi modelli proposti verrà sperimentata su dataset relazionali del mondo reale, e confrontata con quella dei modelli dello SRL.

Organizzazione della tesi

La tesi è strutturata come segue:

Capitolo 1: Background In questo primo capitolo si fornisce una panoramica sullo stato dell'arte riguardo il problema dell'apprendimento nell'ambito dei domini strutturati. Vengono descritti i diversi approcci dell'apprendimento automatico al task dell'on-graph learning e introdotto il modello Neural Network for Graphs, base per i modelli sviluppati successivamente. Inoltre si introduce e definisce il task dell'in-graph learning e l'approccio dello Statistical Relational Learning al problema.

Capitolo 2: Modelli Nel capitolo vengono introdotti e descritti formalmente i nuovi modelli della *Neural Network for In-Graph* sviluppati per estendere la NN4G al problema dell'apprendimento, e si discutono somiglianze e differenze tra l'approccio della rete neurale e quello dei classificatori relazionali dello SRL.

Capitolo 3: Risultati Sperimentali In questo capitolo si riportano i risultati ottenuti con gli esperimenti svolti per verificare la validità dei modelli proposti. Vengono riportati sia dei risultati di collaudo su due problemi di apprendimento tra grafi, sia i risultati su tre dataset per l'apprendimento in-graph, con cui ci si confronta rispetto ai risultati di algoritmi dello Statistical Relational Learning.

Capitolo 4: Conclusioni Il capitolo conclusivo della tesi riprende gli aspetti principali trattati nei capitoli precedenti, presentando le conclusioni ricavate dagli esperimenti svolti e dal confronto con le metodologie preesistenti in letteratura.

Appendice A: Software In appendice viene descritto il framework realizzato e si riportano alcuni dettagli del software.

Appendice B: Complessità Computazionale Si riportano in appendice infine alcuni cenni alla complessità computazionale delle principali operazioni di calcolo dei modelli (codifica, output e allenamento di unità nascoste e di output).

Capitolo 2

Background

In questo capitolo viene introdotto il dominio dei grafi, la notazione utilizzata in seguito nel resto della tesi (sezione 2.1) e viene data una definizione più formale dei task d'apprendimento che verranno presi in considerazione (sez. 2.2). Nelle sezioni successive viene poi fatta una panoramica sui modelli esistenti in letteratura per il trattamento di grafi, nell'ambito del Machine Learning e dello Statistical Relational Learning. In particolare nella sezione 2.7 viene descritto in maniera dettagliata il modello utilizzato come base per lo sviluppo dei modelli oggetto della tesi, *Neural Network for Graphs*.

2.1 Domini Strutturati

Il dominio preso in considerazione in seguito è quello dei grafi. Un grafo $\mathbf{G}(V, E)$ è costituito da un insieme di vertici V e da un insieme di archi $E \subseteq \{(u, v)|u, v \in V\}$. Ad ogni vertice v può essere associata un'etichetta $\mathbf{I}(v)$, ovvero un vettore di attributi (valori reali o discreti) che ne descrivono alcune proprietà. Se la funzione da apprendere è definita sui vertici del grafo, ad ogni vertice è associato anche un target $\mathbf{t}(v)$, che può essere un valore intero, uno scalare o un vettore. Se invece il target dell'apprendimento è una proprietà dell'intero grafo, questa viene rappresentata tramite il vettore $\mathbf{t}(g)$.

Se il grafo è diretto (o orientato), per ogni vertice sono definiti l'insieme dei suoi predecessori $\mathcal{P}(v) = \{u|u \in V \wedge (u, v) \in E\}$ e l'insieme dei suoi successori

$\mathcal{S}(v) = \{u | u \in V \wedge (v, u) \in E\}$; analogamente, se il grafo è indiretto, è definito l'insieme dei vicini di un vertice come l'insieme dei vertici in V collegati direttamente a lui tramite un arco,

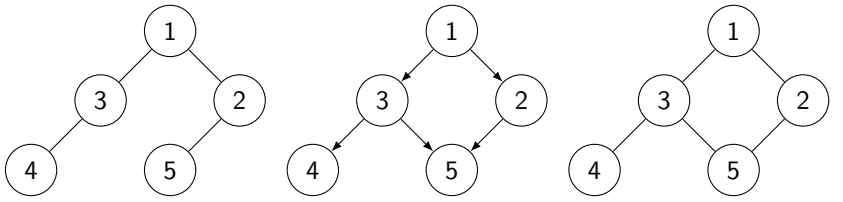
$$\mathcal{N}(v) = \{u | u \in V \wedge ((u, v) \in E \vee (v, u) \in E)\} \tag{2.1}$$

Se è definito un ordinamento totale sui vicini di ogni vertice, con $\mathcal{N}(v)_j$ si indica il j-esimo vicino di v. Indichiamo l'insieme degli archi di un vertice come $\mathbf{e}_{\mathcal{N}(v)}$. Definiamo infine il *contesto* di grado i-esimo di un vertice v come l'insieme di tutti i vertici che è possibile raggiungere a partire da v percorrendo non più di i archi, indicandolo come

$$\mathcal{N}^{(i)}(v) = \{u | u \in V \wedge \exists u' \in V : ((u, u') \in E \vee (u', u) \in E) \ \wedge u' \in \mathcal{N}^{(i-1)}(v)\} \tag{2.2}$$

In particolare, $\mathcal{N}^{(1)}(v) = \mathcal{N}(v)$. L'insieme dei vicini di un vertice può essere *aperto*, se non comprende il vertice stesso, o *chiuso*, se $v \in \mathcal{N}(v)$.

Gli alberi sono una particolare sottoclasse dell'insieme dei grafi: un albero è un grafo indiretto aciclico in cui ogni coppia di vertici è connessa da uno e un solo cammino.



(a) Albero. (b) DAG. (c) Grafo non orientato.

Figura 2.1: Esempi di diversi tipi di grafi.

Altre sottoclassi dell'insieme dei grafi sono i DAG, grafi aciclici diretti, i DPAG, grafi aciclici diretti posizionali, in cui ad ogni arco di un vertice può essere assegnato un intero non negativo diverso, e i DOAG, grafi aciclici diretti e ordinati, in cui esiste un ordinamento totale sugli archi incidenti su ogni vertice.

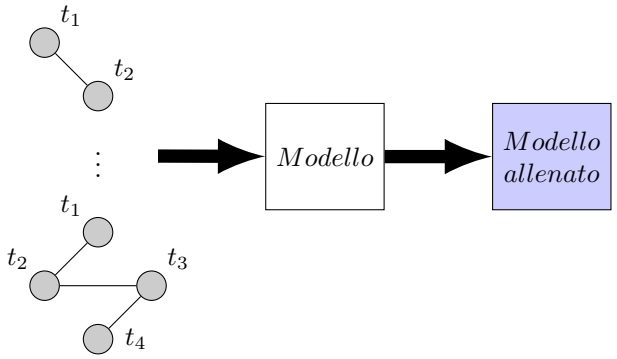
2.2 Task

In generale, lo scopo di un modello di apprendimento automatico supervisionato è di utilizzare un insieme di dati etichettati, il *training set*, per apprendere la funzione che lega ogni entità del dataset alla sua etichetta. Il modello, oltre che riuscire ad approssimare la funzione sui dati noti, deve essere capace di generalizzare, ovvero deve essere applicabile anche a dati non noti al momento dell'apprendimento (questo insieme prende il nome di *test set*) ottenendo risultati sufficientemente accurati. Nell'ambito dei domini strutturati le funzioni definite sul dominio prendono il nome di funzioni di *trasduzione scalare o supersource*, se legano oggetti del dominio a elementi di un dominio non strutturato, o di *trasduzione input-output*, se mettono in relazione tra loro oggetti dello stesso dominio strutturato; in particolare si parla di trasduzione *IO isomorfa* se la funzione è definita solo su coppie di oggetti isomorfi tra loro, ovvero con la stessa struttura [22].

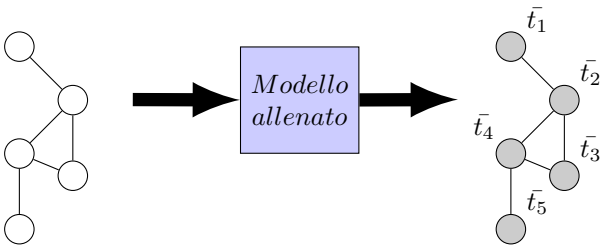
Distinguiamo nell'ambito dei problemi d'apprendimento su grafi due principali categorie: problemi di apprendimento *on-graph*, o tra grafi, e problemi di apprendimento *in-graph*, o *within-network*.

In un problema di on-graph learning il dataset su cui è definito il task è composto da un insieme di grafi; ad ogni grafo è associato un target, che definisce la funzione di trasduzione che si vuole apprendere. Il target dell'apprendimento può essere sia una proprietà dell'intero grafo (trasduzione scalare), sia una proprietà dei singoli vertici di ogni grafo; in questo secondo caso la funzione da apprendere rientra nella classe delle funzioni di trasduzione IO isomorfe, in quanto mette in relazione grafi con grafi isomorfi, con una diversa etichettatura dei vertici. Quello che va sottolineato però è che in un problema on-graph i dati che si hanno a disposizione per la fase di training del modello sono completamente etichettati, ed è presente una separazione tra etichette di input (input label, appartenenti al dominio di input) e etichette di target (appartenenti al dominio di output): nel caso di un task definito sull'intero grafo, ogni grafo g nel dominio di output della trasduzione ha associato il suo target $\mathbf{t}(g)$, e analogamente nel caso di un task definito sui vertici dei grafi, ogni vertice v di ogni grafo nel dominio di output è etichettato con il target relativo, $\mathbf{t}(v)$. In figura 2.2 sono illustrati schematicamente i concetti appena descritti: il dataset su cui il modello viene allenato è composto da una serie di grafi, i cui vertici sono

etichettati con il target (in grigio in figura 2.2a); dopo l'allenamento, il modello ottenuto può venire utilizzato per la classificazione o regressione su altri grafi, dei cui vertici non si conosce il target (figura 2.2b).



(a) Fase di training.



(b) Applicazione del modello (fase di test).

Figura 2.2: Fase di training e fase di test per un problema di on-graph learning (trasduzione input-output isomorfa).

In un problema di in-graph learning o *within-network learning* il dataset è invece composto da un'insieme di entità parzialmente etichettato: ogni elemento del dataset ha alcune proprietà locali, come in un classico problema di apprendimento, ma in un task di questo tipo è inoltre definito tra le entità un insieme di relazioni, che li collega in una rete, o grafo; dati di questo tipo vengono chiamati *networked data*. In questo genere di problema i dati oggetto dell'apprendimento sono sempre i vertici del grafo, sui quali è definita la funzione di trasduzione da apprendere: questa funzione, oltre a dipendere dagli attributi locali di ogni elemento, dipende in una qualche maniera anche dal suo contesto. Si può parlare anche in questo caso di funzioni di trasduzione input-output isomorfe, che mettono in relazione però il grafo

di input con il grafo stesso: dominio di input e dominio di output della trasduzione sono coincidenti.

La principale e sostanziale differenza tra questo problema e il problema dell'on-graph learning definito sopra sta nel fatto che in questo tipo di task il training set, ovvero i vertici v etichettati con il rispettivo target $\mathbf{t}(v)$, e il test set coesistono nello stesso grafo, e sono presenti collegamenti tra i vertici del primo e del secondo insieme. I modelli da applicare a questo tipo di task devono tener conto di questa proprietà peculiare dei dati, che se da un lato può rendere più difficoltoso il processo d'apprendimento, per via della necessaria gestione delle interconnessioni tra vertici etichettati e non, dall'altro può essere sfruttata in fase di test, durante la quale i vertici di training, etichettati, svolgono la funzione di contesto dei vertici da etichettare. Le differenze appena descritte sono evidenti dal confronto tra le figure 2.2 e 2.3, che descrivono schematicamente il processo d'apprendimento e di applicazione dei modelli nei due diversi problemi. Nel caso del within-network learning, il dataset è costituito da un unico grafo, composto sia di vertici etichettati sia di vertici non etichettati (rispettivamente in grigio e in bianco in figura 2.3a); dopo l'allenamento, il modello ottenuto può essere utilizzato per etichettare i vertici del grafo per i quali il target non era noto (figura 2.3b).

Nel seguito si fornisce una definizione formale dei concetti appena descritti.

Obiettivo dell'apprendimento supervisionato è di apprendere dai dati una funzione $\mathcal{T}_D : D \longmapsto O$ che mappi elementi del dominio strutturato D nel dominio di output O.

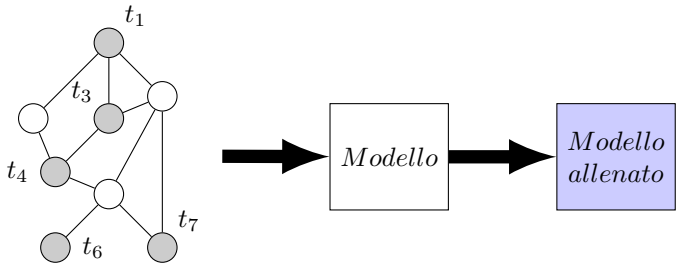
In generale si distinguono due principali task di apprendimento supervisionato:

Classificazione , in cui l'obiettivo è di assegnare ad ogni oggetto del dominio una classe; in questo caso $O = \{0, 1, \ldots, K\}$. In particolare si parla di classificazione binaria se $O = \{-1, 1\}$.

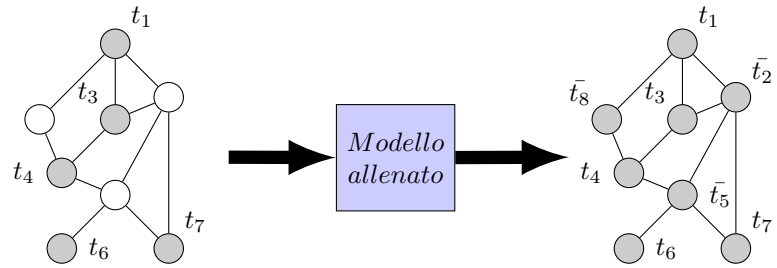
Regressione , in cui l'obiettivo è approssimare col minor errore possibile la funzione $\mathcal{T}_D$, tale che $O = \mathbb{R}^k$.

Le funzioni di trasduzione $\mathcal{T}_D$ che è possibile definire su un dominio strutturato possono essere divise in due categorie:

Trasduzione scalare o supersource : la funzione da apprendere mappa ogni elemento del dominio in un vettore: O è un dominio non strutturato; il task



(a) Fase di training.



(b) Applicazione del modello (fase di test).

Figura 2.3: Fase di allenamento e fase di applicazione del modello per un problema di in-graph learning o within-network learning.

può essere sia di classificazione ($O = \{0, 1, \ldots, K\}$) che di regressione ($O = \mathbb{R}^k$);

Trasduzione input-output (IO) : $O = D$; in particolare, nel seguito si considereranno trasduzioni input-output isomorfe: nel dominio dei grafi, la funzione di trasduzione che si vuole apprendere $\mathcal{T}_D : G \longmapsto G$ mappa ogni grafo in un grafo isomorfo, con una differente etichettatura dei vertici.

Come detto prima, distinguiamo due tipologie di problemi nell'ambito dei domini strutturati, l'on-graph learning, o apprendimento tra grafi, e l'in-graph learning, o within-network learning.

On-graph learning In un problema di apprendimento di questo tipo la funzione di trasduzione (scalare o IO-isomorfa) deve essere appresa da un insieme di grafi completamente etichettati (in particolare, se un grafo appartiene a questo insieme e la funzione è di tipo IO, ogni vertice di quel grafo è etichettato con il suo target corrispondente), e può poi essere applicata ad altri grafi per i quali

si vuole stimare l'output della funzione. Il task può essere sia di classificazione che di regressione.

In-graph learning In un problema di within-network learning, il dataset è costituito da un unico grafo $\mathbf{G}(V, E)$. I vertici di $\mathbf{G}$ sono ripartiti in due insiemi, $V = V_k \cup V_u$, con $V_k \cap V_u = \emptyset$: per i vertici in V_k il target è noto, mentre i vertici in V_u non hanno target associato. Lo scopo è di utilizzare la conoscenza sui vertici in V_k per apprendere il target dei vertici sconosciuti. Anche in questo caso la funzione di trasduzione che si vuole apprendere è IO isomorfa, e mappa il grafo di input nel grafo stesso, assegnando ad ogni vertice non precedentemente etichettato una stima del probabile target.

L'apprendimento tra grafi (on-graph) è stato abbondantemente trattato dal machine learning negli scorsi anni (sez. 2.4, 2.5, 2.6 e 2.7), mentre il within-network learning è stato approfondito e studiato maggiormente nell'ambito dello Statistical Relational Learning (sez. 2.8).

2.3 Reti Neurali e Reti Neurali Ricorrenti

La funzione calcolata da un neurone di una rete neurale *multilayer perceptron feed-forward* è data da

$$o(\mathbf{I}) = f_\sigma \left(\sum_{i=0}^{L} w_i \cdot I_i \right) \tag{2.3}$$

dove $\mathbf{I} = [I_i]_{i=1,\ldots,L}$, $I_0 = 1$ è l'input del neurone, un vettore di feature con una lunghezza prefissata L, $\mathbf{w} = [w_i]_{i=0,\ldots,L}$ è il vettore dei pesi dell'unità, il cui valore viene determinato in maniera adattiva durante l'apprendimento, e $f_\sigma(\cdot)$ è la funzione di attivazione del neurone.

Il più semplice tipo di dato strutturato è la sequenza: una lista di entità (vertici) serialmente ordinati.

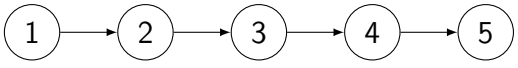


Figura 2.4: Sequenza.

L'obiettivo dell'apprendimento in questo ambito può essere di classificazione (o di regressione) sull'intera sequenza, se lo scopo è assegnare ad ogni sequenza $s \in S$ una classe $c \in \mathcal{C}$; in questo caso la funzione che si vuole apprendere è definita come $\mathcal{T}_s : S \longmapsto \mathcal{C}$ per $\mathcal{C} = \{c_1, \ldots, c_k\}$.

Se invece l'obiettivo è di assegnare una classe ad ogni singolo vertice della sequenza, si parla di funzione di trasduzione sequenziale: si cerca una funzione $\mathcal{T}_s : S \longmapsto S'$ che mappi sequenze in sequenze, calcolando una nuova label per ogni vertice della sequenza originaria.

Le reti neurali ricorrenti (RNN) estendono il modello classico di rete neurale feed forward al dominio delle sequenze, che vengono trattate introducendo il concetto di stato e di memoria nel modello: compito dello stato è di ricordare gli elementi già analizzati della sequenza (codificandoli) e quindi di contribuire insieme all'input corrente alla determinazione dell'output della rete.

Lo scopo è di costruire sistemi con memoria, in cui l'output al tempo t_i, $x(t_i)$, dipenda non solo dall'input corrente $\mathbf{I}(t_i)$ ma anche dal suo contesto, ovvero l'output al tempo $x(t_j)\ \ \forall\, j < i$. Per ottenere questo si passa dalla topologia diretta aciclica delle reti neurali feed forward a una topologia diretta ciclica, introducendo dei collegamenti "all'indietro" tra le unità nascoste, come mostrato schematicamente in figura 2.5.

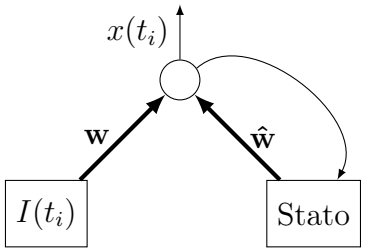


Figura 2.5: Neurone di una rete neurale ricorrente.

Rispetto alla formula 2.3, l'equazione che descrive l'output di un neurone ricorrente cambia in questo modo:

$$x_r(t) = f_\sigma \left(\sum_{i=0}^{L} w_i \cdot I_i(t) + \sum_{j=0}^{N} \hat{w}_j \cdot x_j(t-1) \right) \tag{2.4}$$

Il nuovo termine $\sum_{j=0}^{N} \hat{w}_j \cdot x_j(t-1)$ aggiunge informazione sulle precedenti porzioni della sequenza già esaminate, tramite l'output dei neuroni $i = 0, \ldots, N$ (quali siano questi neuroni dipende dalla topologia della rete neurale, esistono diverse varianti al modello che si differenziano per le connessioni ricorrenti tra neuroni); come i pesi $\mathbf{w}$, anche i pesi $\hat{\mathbf{w}}$ sono determinati tramite training.

Il sottoinsieme della sequenza già esaminato dalla rete neurale viene salvato nello stato del neurone, che riesce così in maniera condensata e adattiva a memorizzare tutte le sottosequenze incontrate precedentemente: il contesto considerato da questo tipo di reti è quindi costituito solamente dalla parte precedente l'input corrente nella sequenza.

La funzione di trasduzione appresa dalla rete neurale può essere formalizzata come la composizione di due funzioni, τ_E, la funzione di stato calcolata dalle unità nascoste ricorrenti, e y, la funzione di output che mappa lo stato nel dominio d'uscita desiderato:

$$\mathcal{T}_s : S \longmapsto \mathbb{R}^k, \quad \tau_E : S \longmapsto \mathbb{R}^m, \quad y : \mathbb{R}^m \longmapsto \mathbb{R}^k \tag{2.5}$$

$$\mathcal{T}_s = y \circ \tau_E \tag{2.6}$$

$$\tau_E(\mathbf{s}) = \begin{cases} nil & \text{se } \mathbf{s} = \emptyset \\ \tau_{NN}(r, \tau_E(\mathbf{s}')) & \text{se } \mathbf{s} = r|\mathbf{s}' \end{cases} \tag{2.7}$$

dove τ_{NN} è la funzione calcolata dalla rete neurale che costituisce lo strato di unità ricorrenti della rete.

Le reti neurali ricorrenti sono caratterizzate da tre proprietà principali:

Adattività : sia la funzione di codifica τ_E che la funzione di output y sono reti neurali: in quanto tali, apprendono il valore dei loro rispettivi parametri liberi durante l'allenamento, adattandoli al problema specifico.

Stazionarietà : il modello descrive un sistema stazionario: la definizione della funzione $\mathcal{T}_s = y \circ \tau_E$ non dipende dal tempo, ovvero dal particolare elemento della sequenza che si sta considerando in un certo momento.

Causalità : il sistema è causale: l'output della rete per t_i, $x(t_i)$, dipende solo da t_j per $j < i$, ovvero, il contesto del vertice corrente è costituito solamente dalle sottosequenze precedenti il vertice.

2.4 Reti Neurali Ricorsive

Le reti neurali ricorsive rappresentano la naturale estensione delle reti neurali ricorrenti ai domini strutturati come alberi e grafi. Il modello, una generalizzazione del neurone ricorrente capace di processare solo sequenze, viene illustrato in [26]: viene introdotto un diverso tipo di neurone, chiamato neurone ricorsivo, in grado di trattare elementi appartenenti ad alcuni tipi di domini strutturati più complessi delle sequenze, come alberi e grafi aciclici.

L'idea alla base del modello proposto è di calcolare una *codifica* per ogni vertice del grafo, che sintetizzi sia l'informazione locale data dagli attributi che l'informazione sul contesto del vertice: in precedenza, la metodologia principale per poter applicare a grafi (o ad altri domini strutturati) i modelli di apprendimento preesistenti per dati flat, consisteva appunto nello stabilire una codifica, chiamata *codifica a priori*, e applicarla ad ogni grafo del dataset. Il risultato era un nuovo dataset i cui elementi non erano più grafi ma vettori flat, su cui era immediato applicare un qualsiasi modello standard. Il principale difetto di questo tipo di approccio è che la codifica deve essere decisa "manualmente", prima dell'apprendimento, ad esempio in base a qualche conoscenza sullo specifico task in questione o sul dominio di applicazione. La codifica è una fase completamente separata dal processo di apprendimento.

L'idea descritta in [26] al contrario, consiste nel calcolare la codifica dei grafi in maniera adattiva, tramite una rete neurale composta da un nuovo tipo di neuroni, i neuroni ricorsivi. Formalmente l'output di un neurone ricorsivo può essere espresso come

$$x_r(v) = f_\sigma \left(\sum_{i=0}^{L} w_i \cdot I_i(v) + \sum_{j=1}^{D} \hat{w}_j \cdot x_r(\mathcal{S}(v)_j) \right) \tag{2.8}$$

Come prima, $\mathbf{I}(v)$ è la label associata al vertice v, di dimensione L, mentre

$x_r(\mathcal{S}(v)_j)$, con $\mathcal{S}(v)$ l'insieme dei successori di v (sez. 2.1), è l'output del neurone stesso (collegamento ricorsivo) per ciascuno dei vertici collegati a v da un arco uscente e D è il massimo out-degree del grafo.

La formula 2.8 si riferisce al caso particolare in cui il neurone ha connessioni ricorsive solo con se stesso. In generale, come per le reti ricorrenti, si possono avere interconnessioni ricorsive tra qualsiasi coppia di neuroni. Supponendo di avere m neuroni interconnessi, la formula diventa

$$\mathbf{x}_r(v) = \mathbf{F}_\sigma \left(\mathbf{W} \cdot \mathbf{I}(v) + \sum_{j=1}^{D} \widehat{\mathbf{W}}_j \cdot \mathbf{x}_r(\mathcal{S}(v)_j) \right) \tag{2.9}$$

dove $\mathbf{F^{(i)}}_\sigma(\cdot) = f_\sigma(\cdot)$ per $i = 1, \ldots, m$, $\mathbf{W} \in \mathbb{R}^m \times \mathbb{R}^L$, $\mathbf{I}(v) \in \mathbb{R}^L$ e $\widehat{\mathbf{W}}_j \in \mathbb{R}^m \times \mathbb{R}^m$, $\mathbf{x}_r(\cdot) \in \mathbb{R}^m$.

La codifica di ogni vertice è ricorsiva: così come per le sequenze l'output per il vertice corrente dipendeva anche dalla precedente porzione della sequenza, per i grafi l'output per il vertice v dipende dall'output di tutti i suoi discendenti, che formano il suo contesto, come schematizzato in figura 2.6: il contesto dei vertici foglia 2 e 4, senza archi uscenti, è vuoto, mentre il contesto del vertice 1 e del vertice 3 è formato da tutti i loro discendenti.

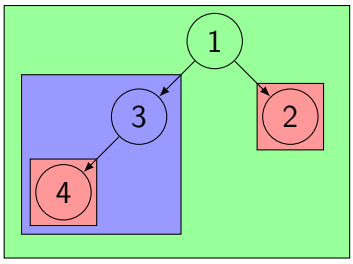


Figura 2.6: Esempio di contesto in un albero.

La funzione di trasduzione $\mathcal{T}_G : G \longmapsto O$ calcolata dalla rete neurale può anche in questo caso essere suddivisa in due sottofunzioni, la funzione di codifica ricorsiva $\tau_E : G \longmapsto \mathbb{R}^m$, che trasforma l'input G nella corrispondente codifica, ovvero nei nuovi valori di stato della rete neurale (l'encoding del grafo), e la funzione di output $y : \mathbb{R}^m \longmapsto O$ che mappa lo stato nel dominio di output.

$$\tau_E(\mathbf{G}) = \begin{cases} 0 & \text{se } \mathbf{G} = \emptyset \\ \tau_{NN}(L_{root}, \tau_E(\mathbf{G}^{(1)}), \ldots, \tau_E(\mathbf{G}^{(k)})) & \text{altrimenti} \end{cases} \tag{2.10}$$

dove τ_{NN} come per le reti neurali ricorrenti è la funzione calcolata dalla rete neurale che costituisce lo strato di unità ricorsive della rete. L'equazione 2.10 esprime formalmente la definizione della funzione di codifica (ovvero di transizione di stato) della rete: è possibile osservare come venga applicata ricorsivamente in maniera bottom-up a tutti i vertici appartenenti a $\mathbf{G}$, in modo che le foglie (o i vertici senza archi uscenti) vengano codificate solo in base alla loro label, mentre i vertici a livelli superiori vengono influenzati dalla codifica di tutti i loro discendenti.

Per le reti neurali ricorsive continuano a valere le tre proprietà elencate in 2.3:

- La funzione è adattiva, grazie alla presenza dei pesi modificabili nelle funzioni τ_E e y ($\mathbf{W}$ e $\widehat{\mathbf{W}}$ nell'equazione 2.9).

- La stazionarietà impone che la funzione di codifica τ_E sia indipendente dal tempo: nell'ambito dei domini strutturati ogni vertice corrisponde a uno step temporale distinto, e quindi ciò che si impone è che la funzione sia sempre la stessa applicata a tutti i vertici (ovviamente, quanto detto vale solo dopo la fase di apprendimento).

- La funzione di trasduzione è causale, in quanto l'output della rete per un vertice dipende solo dal vertice e dai suoi discendenti, così come nelle sequenze l'output dipendeva solo dai predecessori del vertice in questione. Questa proprietà impone una forte condizione sul tipo di trasduzioni che queste possono apprendere, in quanto si considera come contesto solamente l'insieme $\mathcal{S}(v)$ dei discendenti di ogni vertice; come per le reti neurali ricorrenti però, esistono varianti del modello che permettono di rilassare questa condizione e considerare un contesto più ampio, che verranno trattate nella prossima sezione.

Il modello descritto è facilmente e direttamente applicabile a tutte le classi di grafi che non presentano cicli, come alberi, DAG e DPAG, per i quali è sempre possibile stabilire un ordinamento topologico dei vertici che assicura la compatibilità con la funzione di trasduzione descritta sopra. Per applicare il modello a grafi ciclici invece, è necessario assicurarsi che alcune condizioni sulla encoding network siano

soddisfatte, affinchè sia garantito che il processo di codifica termini in uno stato stabile per ogni grafo; in [26] ad esempio viene riportata come condizione sufficiente per la convergenza che la matrice dei pesi sia sufficientemente piccola in norma. Ovviamente però, a causa del processo di apprendimento, la rete non è statica: si può garantire solamente che queste condizioni valgano all'inizializzazione della rete, ma non si può controllare come durante l'apprendimento i pesi verranno modificati.

2.5 Reti Neurali per Grafi

In questa sezione vengono descritti alcuni degli approcci al problema dell'apprendimento su grafi tramite reti neurali.

La *Relational Neural Network* (RelNN) [30] è un modello di rete neurale definito su grafi aciclici e alberi, che può essere esteso al dominio dei grafi ciclici tramite preprocessing del dataset di input. Lo scopo del preprocessing è di trasformare grafi ciclici in alberi; la trasformazione avviene tramite una visita *breadth-first* del grafo, il cui punto di partenza è il vertice "di output" del grafo, un vertice che viene scelto secondo un qualche criterio come quello a cui associare il target per l'intero grafo (il modello è direttamente applicabile solo a problemi di trasduzione scalare). Ad ogni passo della visita alla copia del vertice corrente v già inserita nell'albero vengono collegati, come figli, tutti i vertici che appartengono al suo vicinato; il processo continua finchè l'albero non raggiunge una certa profondità predefinita.

Questo modello presenta alcuni punti in comune con le reti neurali ricorsive, come la limitazione al dominio dei grafi aciclici, e l'utilizzo dello stesso algoritmo d'apprendimento, la *Backpropagation through structure* [26], per l'aggiornamento dei pesi della rete. Si differenzia però dal precedente per la realizzazione dell'encoding network, che è implementata da una rete neurale ricorrente: la codifica per il vertice v dell'albero è calcolata come l'output di una rete ricorrente applicata alla sequenza formata dalle codifiche dei figli del vertice, ordinati secondo un certo criterio. L'ordinamento dei figli di un vertice, se non è dato dalla struttura del grafo (ad esempio se i grafi sono posizionali o ordinati), può influire nel processo d'apprendimento.

Il *Graph Neural Network model* (GNN), illustrato in [27], viene invece proposto come una estensione del modello ricorsivo in grado di trattare anche grafi ciclici.

Anche in questo modello il grafo viene processato da un insieme di unità, una per ogni vertice, connesse tra loro in base alla topologia del grafo. Ogni unità calcola uno stato per il vertice associato, basandosi sulle informazioni locali del vertice (ovvero la sua label $\mathbf{I}(v)$) e sulle informazioni sui suoi vicini, $\mathcal{N}(v)$.

Più formalmente, indicando con $\mathbf{x}(v)$ la funzione di stato e con $\mathbf{o}(v)$ la funzione di output,

$$\mathbf{x}(v) = f_w \left(\mathbf{I}(v), \mathbf{I}(\mathcal{N}(v)), \mathbf{I}(\mathbf{e}_{\mathcal{N}(v)}), \mathbf{x}(\mathcal{N}(v))\right) \tag{2.11}$$

$$\mathbf{o}(v) = y_w \left(\mathbf{I}(v), \mathbf{x}(v)\right) \tag{2.12}$$

La funzione di stato è ricorsiva e, come prima, se il grafo è ciclico si creano delle dipendenze cicliche anche nelle connessioni della rete neurale. La soluzione proposta in [27] consiste nel calcolare lo stato tramite un processo iterativo, che si fermerà al raggiungimento di un punto di equilibrio (punto fisso). Il teorema del punto fisso di Banach assicura che, se la funzione f_w è contrattiva[1], il sistema di equazioni ammette una soluzione unica. Affinchè quindi il modello sia applicabile ad un grafo non diretto o comunque contenente cicli, è necessario che vengano imposte sulla funzione f_w (ovvero, sui pesi della rete neurale) dei vincoli che assicurino la contrattività della funzione di stato.

L'equazione di stato 2.11 diventa quindi

$$\mathbf{x}_{t+1}(v) = f_w \left(\mathbf{I}(v), \mathbf{I}(\mathcal{N}(v)), \mathbf{I}(\mathbf{e}_{\mathcal{N}(v)}), \mathbf{x}_t(\mathcal{N}(v))\right) \tag{2.13}$$

Durante l'algoritmo di apprendimento vengono quindi eseguiti due compiti: ad ogni epoca, si calcolano i nuovi valori per i pesi $\mathbf{w}$, con lo scopo di minimizzare l'errore della funzione di loss, e successivamente si esegue il processo iterativo che permette di calcolare il punto fisso di $\mathbf{x}(v)$; la necessità di questo processo di convergenza aggiunge ovviamente un'ulteriore costo al processo di apprendimento.

A dimostrazione dell'interesse per questo settore dell'apprendimento automatico da parte della ricerca, molto recentemente, in [31], è stato presentato un approccio

[1] f è contrattiva se $\exists \mu, 0 \leq \mu < 1$, tale che $\forall x, y \, \|f(x) - f(y)\| \leq \mu \, \|x - y\|$

che incorpora alcuni degli aspetti di RelNN e di GNN in un nuovo modello per problemi relazionali, chiamato *Recurrent Neural Collective Classification* (RNCC). Il modello sfrutta, come RelNN, dei neuroni ricorrenti per aggregare le informazioni sul vicinato di un vertice e utilizzarle per la classificazione, che viene poi rifinita tramite successive iterazioni, come in GNN.

Il modello *Graph Echo State Network* (GESN, [28]) appartiene invece al paradigma del Reservoir Computing, ed è una estensione della Echo State Network per sequenze [32] al dominio dei grafi.

In questo paradigma si ha una separazione concettuale delle unità della rete in due layer: il reservoir, costituito da molte unità non lineari, non completamente connesso e con connessioni casuali, e il read-out, un layer con unità (di solito) lineari. La principale differenza tra i due livelli di unità, che caratterizza questo approccio, sta nel fatto che, mentre le unità di output del readout vengono normalmente allenate, le unità del reservoir vengono solamente inizializzate, seguendo delle specifiche regole, ma non allenate.

In GESN il reservoir ha la stessa funzione dell'hidden layer nelle reti neurali ricorsive: ricavare dal grafo in input la sua codifica, che verrà poi usata dal read-out della rete per produrre l'output del modello per quel grafo. Il reservoir viene inizializzato di modo che la funzione di stato calcolata sia contrattiva, come per GNN: anche in questo caso questa proprietà è necessaria per garantire la convergenza e la stabilità della funzione di codifica.

Anche in questo modello infatti la codifica di un vertice viene calcolata tramite il processo iterativo descritto dall'equazione

$$\mathbf{x}_{t+1}(v) = \tau_E\left(\mathbf{I}(v), \mathbf{x}_t(\mathcal{N}(v))\right) \tag{2.14}$$

fino al raggiungimento del punto fisso di $\mathbf{x}(v)$. Contrariamente a quanto detto per GNN però, punto di forza di questo modello è l'efficienza del calcolo della codifica durante il training, in quanto, nonostante sia sempre necessario un processo iterativo, il reservoir non deve essere allenato, mentre in GNN devono essere alternate le due fasi, che possono essere computazionalmente onerose, dell'allenamento delle unità e della convergenza al punto fisso dell'equazione iterativa.

2.6 Kernel per Grafi

Nel contesto dei dati non strutturati un kernel è una funzione di similarità, definita tra coppie di oggetti del dominio, solitamente usata in congiunzione con le Support Vector Machine (SVM) per trasformare il dominio dei dati di input in uno spazio di dimensione maggiore, molto più grande della dimensione dello spazio di input di partenza, con lo scopo di rendere il problema di classificazione un problema linearmente separabile nel nuovo *feature space*. Un kernel $k(x_i, x_j)$ definisce un prodotto scalare tra coppie di elementi del dataset e può essere usato per evitare l'esplicita trasformazione dei dati nelle nuove coordinate dello spazio ad alta dimensionalità in quanto, se la matrice associata al kernel è semidefinita positiva, $k(x_i, x_j)$ è uguale al prodotto dei vettori x_i e x_j nel nuovo feature space.

Negli ultimi anni sono stati proposti diversi kernel per il trattamento di dati strutturati come grafi o alberi [24]. Si distinguono due principali classi di kernel [23], kernel *on graph*[2], ovvero funzioni di somiglianza definite tra coppie di vertici di uno stesso grafo, e kernel *between graph*, che definiscono funzioni di similarità tra coppie di grafi. Una differenza fondamentale tra la codifica di un grafo effettuata da uno degli approcci neurali descritti nelle sezioni 2.5 e 2.7 e l'approccio tramite kernel sta nel fatto che le prime sono adattive, ovvero vengono determinate tramite il processo d'apprendimento, mentre la funzione di somiglianza tra gli elementi del dominio è fissata una volta scelta la tipologia di kernel che si vuole utilizzare; è quindi necessaria una minima conoscenza del task in questione per poter selezionare il kernel più appropriato alla rappresentazione dei grafi su cui questo è definito. Si riportano brevemente nel seguito due esempi di kernel tra grafi che sono stati applicati ai dataset PTC utilizzati in sezione 4.1 per un collaudo del framework proposto.

Il *Marginalized Kernel* (MG-kernel) [25], è un kernel between graph, e mette quindi in relazione due grafi e non i vertici di uno stesso grafo. Il concetto alla base del kernel è che due grafi sono tanto più simili quanto maggiore è il numero di percorsi (definiti come sequenze di etichette appartenenti a vertici collegati da un arco) che hanno in comune: data la complessità computazionale del calcolo esplicito di

[2]Questa denominazione non va confusa con quella utilizzata in questa tesi, in cui con problemi di on-graph learning si intende la classe dei problemi d'apprendimento supervisionato definiti su grafi completamente etichettati (sez. 2.2).

tutti i possibili cammini in un grafo (in realtà, se sono presenti cicli, questi sono infiniti), si considerano invece i cammini prodotti da random walks sui vertici. Il calcolo del kernel tra due grafi $\mathbf{G}(V, E)$ e $\mathbf{G}'(V', E')$ richiede la risoluzione di un sistema di equazioni lineari la cui matrice dei coefficienti ha dimensione $|V||V'| \times |V||V'|$, anche se il numero di coefficienti diversi da zero è minore di $max(D, D')^2|V||V'|$ dove D e D' sono il massimo numero di archi incidenti in un vertice in $\mathbf{G}$ e $\mathbf{G}'$ rispettivamente; se $max(|V|, |V'|) = n$ e si assumono piccoli e costanti i valori di D e D', la complessità di questo kernel risulta di $\mathcal{O}(n^2)$.

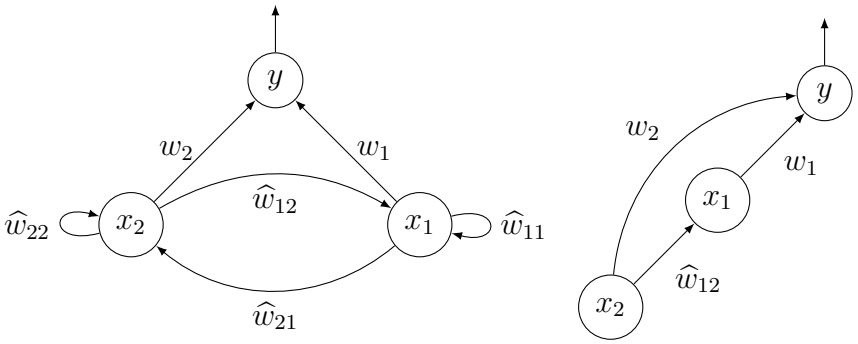
Anche l'*Optimal Assignment Kernel* (OA-kernel) [21] è un kernel tra grafi, che nasce specificamente per l'applicazione al mondo della chemioinformatica; in questo kernel, la similarità tra due grafi è definita in base alla massima somiglianza tra coppie di vertici dei due (per ogni vertice del primo grafo, si considera la somiglianza con il vertice del secondo per il quale questo valore è massimo), calcolata in base alla somiglianza tra le label locali dei due vertici e tra la struttura del loro vicinato; nel calcolare quest'ultima, si considera la similarità tra le label locali di coppie di vertici dei due vicinati pesata dalla similarità degli archi che li connettono al vertice. Considerando una coppia di grafi con n e n', $n > n'$ vertici, la complessità del calcolo di un kernel di questo tipo è di $\mathcal{O}(n^3)$. Anche l'*Expected Match Kernel* (EM-kernel), introdotto sempre in [21], è basato sulla stessa idea del precedente, ma rispetto a questo risulta più efficiente, in quanto non considera la massima somiglianza di coppie di vertici ma quella attesa, ed ha la stessa complessità di un MG-kernel, ovvero $\mathcal{O}(n^2)$.

2.7 Neural Network for Graphs

In questa sezione viene descritto nel dettaglio *Neural Network for Graphs* (NN4G), introdotto in [29], il modello di rete neurale per grafi utilizzato come punto di partenza per lo sviluppo dei modelli descritti nel capitolo 3. Il modello è definito per il task dell'on-graph learning, sia per trasduzioni scalari che input-output isomorfe.

NN4G si differenzia dai precedenti approcci neurali per grafi per vari aspetti. Innanzitutto, l'architettura neurale di NN4G non è ricorsiva, ma *feedforward*, come nei modelli classici di reti neurali per dati flat. In figura 2.7a è illustrata una schematizzazione di una rete neurale ricorsiva, come descritta nella sezione 2.4, con

due unità nascoste ed una unità di output: come si vede le unità nascoste sono collegate ricorsivamente tra loro, cosa che comporta i problemi di convergenza di cui si è accennato nella sezione 2.4. Nella figura 2.7b è riportata l'architettura del modello NN4G sempre con due unità nascoste e una unità di output: in questo caso la rete non presenta connessioni ricorsive, ma è appunto feedforward, non essendo presenti cicli nella topologia della rete neurale.



(a) Rete neurale ricorsiva con due unità. (b) NN4G con due unità.

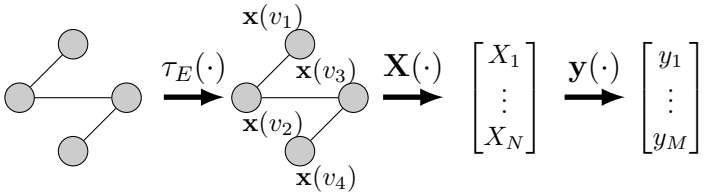
Figura 2.7: Confronto tra una rete neurale ricorsiva e NN4G.

Nonostante questa grande semplificazione architetturale, il modello è dotato di un maggiore potere espressivo e di apprendimento rispetto alle reti neurali ricorsive descritte in sezione 2.4: è infatti in grado di trattare qualsiasi tipo di grafo senza porre nessuna condizione sulla matrice dei pesi della rete. Ciò è possibile perchè l'architettura della rete è costruttiva: in una architettura costruttiva il numero di unità nascoste della rete non è predeterminato, ma viene determinato durante il processo di apprendimento: l'algoritmo di apprendimento, oltre che i pesi delle connessioni neurali, determina anche la topologia della rete stessa.

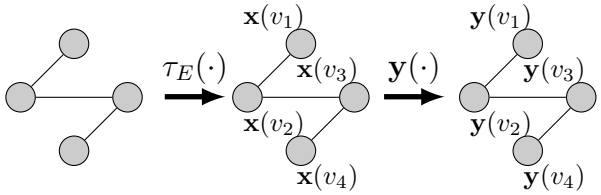
L'architettura costruttiva di NN4G riprende quella per dati flat descritta in [33], la Cascade Correlation. In che modo l'algoritmo di apprendimento determina la struttura della rete viene descritto successivamente in questa sezione, ma è importante sottolineare come questa architettura possa permettere di realizzare trasduzioni contestuali senza il bisogno di introdurre connessioni ricorsive nella struttura della rete neurale. Le unità nascoste vengono aggiunte incrementalmente alla rete, e allenate al momento del loro inserimento, per poi essere "congelate": una volta inserita e allenata, un'unità nascosta non modifica più i suoi pesi. Grazie a

questa proprietà si ha che la successiva unità da inserire nella rete, avendo accesso alla codifica effettuata da tutte le precedenti unità nascoste, ha a disposizione per ogni vertice informazioni sui suoi vicini, senza bisogno di introdurre cicli nella struttura della rete neurale e quindi di conseguenza nella definizione delle variabili di stato del modello.

Formalmente, NN4G calcola la seguente funzione di trasduzione: $\mathcal{T}_G = y \circ \tau_E$, con $\tau_E : G \longmapsto G$ funzione di codifica (o di transizione di stato) che mappa il grafo in input in uno isomorfo e $y : G \longmapsto G$ funzione di output, nel caso di trasduzione IO-isomorfa: per problemi di questo tipo quindi la codifica del grafo effettuata da NN4G consiste nella trasformazione del grafo in uno isomorfo, in cui ogni vertice viene etichettato con un vettore rappresentante lo stato associato dalla rete a quel particolare vertice. Nel caso in cui l'output desiderato sia invece un vettore di scalari (o uno scalare, $M = 1$), $\mathcal{T}_G = y \circ \mathbf{X} \circ \tau_E$ con $\mathbf{X} : G \longmapsto \mathbb{R}^N$ e $y : \mathbb{R}^N \longmapsto \mathbb{R}^M$: tramite la successiva applicazione della funzione $\mathbf{X}$ al grafo risultante dalla codifica dei vertici, viene calcolato l'encoding dell'intero grafo. In figura 2.8 sono riportate schematicamente le due diverse funzioni di trasduzione, scalare (figura 2.8a) e IO isomorfa (figura 2.8b) rispettivamente.



(a) Trasduzione scalare.



(b) Trasduzione IO isomorfa.

Figura 2.8: Scomposizione del calcolo della funzione di trasduzione di NN4G in calcolo della codifica e dell'output nei due casi della trasduzione scalare e IO isomorfa.

La funzione τ_E di codifica produce un grafo isomorfo all'input in cui ogni vertice v viene etichettato con un vettore di N componenti, una per ogni unità nascosta

della rete, contenente in posizione i il valore calcolato dall'i-esima unità nascosta per v, $x_i(v)$. Questo vettore viene chiamato stato o codifica del vertice. La definizione della funzione di transizione di stato è riportata nell'equazione 2.15,

$$\begin{aligned} x_1(v) &= f\left(\sum_{j=0}^{L} \bar{w}_{1j} \cdot I_j(v)\right) \\ x_i(v) &= f\left(\sum_{j=0}^{L} \bar{w}_{ij} \cdot I_j(v) + \sum_{j=1}^{i-1} \sum_{u \in \mathcal{N}(v)} \widehat{w}_{ij}^{(v,u)} \cdot x_j(u)\right) = f(net_i) \end{aligned} \quad (2.15)$$

dove f è la funzione d'attivazione del neurone, $\bar{w}_{ij}$ è il peso associato alla connessione tra l'unità i e il j-esimo attributo del vertice in input $I_j(v)$ ($\mathbf{I}(v)$ è l'input label del vertice v, di dimensione L) e $\widehat{w}_{ij}^{(v,u)}$ è il peso associato alla connessione tra l'unità i e l'unità j per l'arco (v, u).

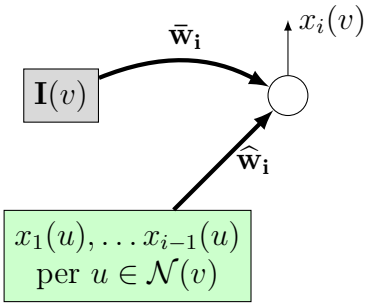


Figura 2.9: Unità nascosta di Neural Network for Graphs.

Come si vede dalla formula 2.15, la prima unità nascosta x_1 calcola il suo output utilizzando solamente la label del vertice che riceve in input. Ogni altra unità nascosta x_i invece, oltre alla label del vertice, utilizza anche lo stato di tutte le unità precedenti (x_j per $j = 1, ..., i-1$) per tutti i vertici u tali che esiste un arco tra u e v.

Dalla formula è esplicito come non siano presenti connessioni "all'indietro" in questo modello: sostanziale differenza con i precedenti modelli ricorsivi sta nel fatto che, indipendentemente dalla topologia del grafo, la funzione di stato di x_i per definizione non dipenderà mai da x_i stessa, ma solo da x_j per $j < i$: non è possibile che si formino cicli di dipendenze funzionali tra gli stati della rete.

Il modello è in grado di apprendere *trasduzioni contestuali* arbitrarie, ovvero trasduzioni in cui l'output della funzione può dipendere non solo dai vicini diretti di un vertice $\mathcal{N}(v)$, ma anche da vertici contenuti nel suo contesto di grado k, $\mathcal{N}^{(k)}(v)$, cioè dai vertici distanti da v un percorso di lunghezza arbitraria k. Infatti, nonostante nel calcolo di $x_i(v)$ intervengano esplicitamente solo i vertici $u \in \mathcal{N}(v)$, è importante sottolineare che in maniera indiretta contribuiscono alla codifica di v anche le codifiche di tutti i vertici in $\mathcal{N}^{(i-1)}(v)$: per ogni vicino u, la codifica $x_{i-1}(u)$ è stata precedentemente calcolata con la medesima formula 2.15, e quindi dipende dall'encoding dei vertici nel vicinato di u, ovvero da parte del contesto $\mathcal{N}^{(2)}(v)$ di v, e così via.

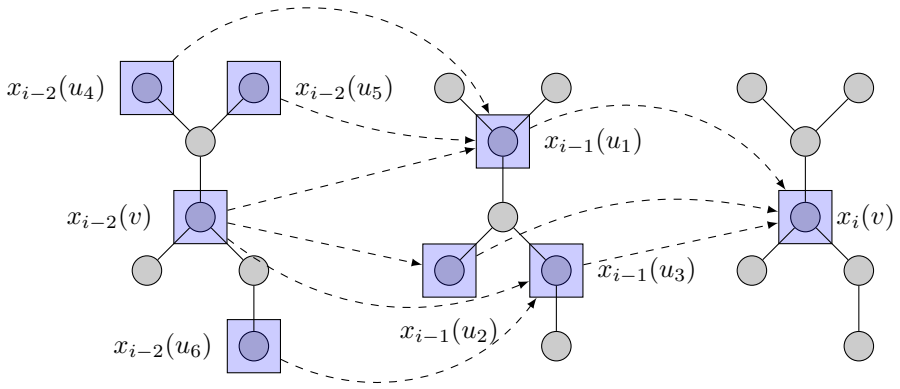


Figura 2.10: Esempio di come il calcolo della componente i-esima della codifica di un vertice sia influenzato non solo dal suo vicinato, ma anche dal suo contesto. In particolare, l'esempio mostra come la componente i-esima della codifica del vertice v, $x_i(v)$, dipende in maniera diretta dalla componente $(i-1)$-esima della codifica dei vicini di v, u_1, u_2 e u_3, ma anche in maniera indiretta dalla componente $(i-2)$-esima della codifica dei vicini di questi vertici, u_4, u_5, u_6 e v stesso, che costituiscono il contesto $\mathcal{N}^{(2)}(v)$ di v.

Il concetto è illustrato graficamente in figura 2.10, dove si mostra parte del flusso dell'informazione di stato durante il calcolo della codifica di un vertice. A differenza quindi delle reti ricorrenti (sez. 2.4), in cui il contesto era limitato ad un sottoinsieme dei vertici del grafo, in questo modello l'ipotesi di causalità viene rilassata: se la rete neurale contiene un numero sufficientemente grande di unità nascoste, il contesto preso in considerazione tramite questo meccanismo può arrivare a coprire tutto il grafo (o almeno la componente di questo cui appartiene il vertice),

e quindi la codifica di ogni vertice può essere calcolata tenendo conto dell'influenza della codifica di tutti gli altri vertici del grafo. Per la dimostrazione formale di questa proprietà di NN4G si rimanda a [29].

Affinchè sia assicurata anche per questo modello l'assunzione di stazionarietà, sono necessarie delle ipotesi aggiuntive sui pesi $\widehat{w}_{ij}^{(u,v)}$: le funzioni di codifica $x_i(\cdot)$ devono essere indipendenti dal vertice v (per assicurare l'invarianza temporale della funzione di trasduzione), e a questo scopo si devono imporre dei vincoli sui pesi, con una qualche tecnica di *weight sharing*; ad esempio in [29] è riportato il caso in cui a tutti gli archi del grafo viene associato lo stesso peso, per cui l'equazione 2.15 diventa

$$x_i(v) = f\left(\sum_{j=0}^{L} \bar{w}_{ij} \cdot I_j(v) + \sum_{j=1}^{i-1} \widehat{w}_{ij} \sum_{u \in \mathcal{N}(v)} x_j(u)\right) \tag{2.16}$$

Nel caso di trasduzione scalare, la funzione $\mathbf{X} : G \longmapsto \mathbb{R}^N$ mappa il grafo codificato in un vettore di N numeri reali (dove N è il numero di unità nascoste della rete) (fig. 2.8a). Ad esempio

- $X_i(g) = \sum_{v \in V} x_i(v)$,
- $X_i(g) = \frac{1}{|V|} \cdot \sum_{v \in V} x_i(v)$,
- $X_i(g) = \frac{1}{\max_{\mathbf{G}(V,E) \in T_r}\{|V|\}} \cdot \sum_{v \in V} x_i(v)$.

La funzione di output $\mathbf{y}$ infine è realizzata dalle M unità di output della rete, che calcolano

$$y_o(p) = \begin{cases} f\left(\sum_{i=0}^{N} w_{oi} \cdot X_i(p)\right) = f(net_o) & \text{se } p \in \mathbf{G} \\ f\left(\sum_{i=0}^{N} w_{oi} \cdot x_i(p)\right) = f(net_o) & \text{se } p \in V \end{cases} \tag{2.17}$$

per $o = 1, ..., M$, rispettivamente se la trasduzione ha come output un vettore di scalari o un grafo isomorfo.

Riassumendo, il modello appena descritto è dotato di una struttura meno complessa dei modelli discussi nelle precedenti sezioni 2.4 e 2.5, in quanto l'architettura neurale è feed-forward e non sono presenti connessioni cicliche nella topologia della

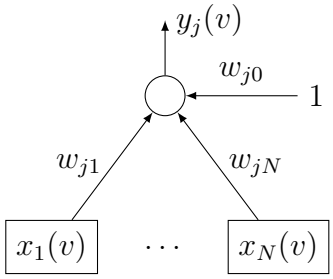


Figura 2.11: Unità di output di Neural Network for Graphs per trasduzioni IO isomorfe.

rete, ma allo stesso tempo è dotato di un notevole potere di apprendimento su domini strutturati. Rilassando l'ipotesi di causalità propria delle reti neurali ricorsive (sez. 2.4) è infatti possibile modellare trasduzioni contestuali, in cui il contesto di un vertice non è limitato ad un sottoinsieme dei vertici del grafo ma può arrivare a comprendere tutti i vertici ad esso connessi da un cammino di lunghezza arbitraria; mentre nelle RNN il contesto di ogni vertice viene limitato ai suoi predecessori nel grafo (nel caso di grafi DAG o DPAG, ai quali il modello è direttamente applicabile), in NN4G il contesto considerato comprende tanto i predecessori che i successori di un vertice, indiscriminatamente.

Inoltre, rispetto anche ai modelli di reti neurali per grafi discussi nella sezione 2.5, NN4G è in grado di trattare grafi di qualsiasi genere, anche ciclici, senza bisogno di introdurre connessioni ricorsive, che richiedono la stabilizzazione di sistemi dinamici, come in GNN o GESN, o di trasformazioni del grafo di input che ne eliminino i cicli, come in RelNN. Per questo motivo, grazie all'architettura costruttiva, il modello risulta più semplice rispetto a quelli appena citati, sia sotto il punto di vista della definizione e del calcolo della funzione di stato, sia per quanto riguarda il processo di apprendimento, che non necessita retropropagazione dell'errore nè convergenza di un processo iterativo.

Algoritmo di apprendimento

Neural Network for Graphs presenta una architettura feedforward costruttiva: il numero di unità nascoste non è specificato a priori, ma viene determinato automaticamente dall'algoritmo di apprendimento, che apprende quindi sia i pesi $w, \bar{w}, \widehat{w}$ che la topologia della rete.

L'algoritmo di apprendimento può essere implementato tramite una estensione dell'algoritmo per la Cascade Correlation Learning architecture ([33]).

Come in tutti i modelli di apprendimento supervisionato, scopo dell'apprendimento in NN4G è di minimizzare una funzione di errore E_{tot} valutata sul training set T_r.

Nel caso di trasduzione scalare si ha

$$E_{tot}^{(s)} = \frac{1}{|T_r|} \sum_{g \in T_r} \sum_{o \in O} \left(y_o(g) - t_o(g)\right)^2 \tag{2.18}$$

mentre nel caso di trasduzione IO

$$E_{tot}^{(io)} = \frac{1}{|T_r|} \sum_{g \in T_r} \sum_{v \in V_g} \sum_{o \in O} \left(y_o(v) - t_o(v)\right)^2 \tag{2.19}$$

in quanto nel secondo caso il target e quindi di conseguenza la funzione di errore sono definiti sui singoli vertici di ogni grafo del dataset.

Per quanto riguarda le unità di output della rete, queste vengono allenate con una classica discesa del gradiente: ad ogni epoca di allenamento il peso della connessione tra l'unità nascosta i e l'output unit o viene aggiornato secondo la regola[3]

$$w_{oi} = w_{oi} - \eta \Delta w_{oi} \tag{2.20}$$

dove η è un parametro che regola la velocità di apprendimento e Δw_{oi} viene così ricavato

[3]Nell'implementazione della rete sono state utilizzate anche altre varianti dell'algoritmo di apprendimento, per le quali cambia però solo la regola di aggiornamento dei pesi, e non come vengono calcolati i valori Δw_{ij} (vedi appendice A).

$$\begin{aligned}\Delta w_{oi} &= \frac{\partial E_{tot}^{(s)}}{\partial w_{oi}} \\ &= \frac{1}{|T_r|}\sum_{g\in T_r}\frac{\partial\left(y_o(g)-t_o(g)\right)^2}{\partial w_{oi}} \\ &= \frac{1}{|T_r|}\sum_{g\in T_r}2\left(y_o(g)-t_o(g)\right)\frac{\partial f(net_o)}{\partial net_o}\frac{\partial net_o}{\partial w_{oi}} \\ &= \frac{1}{|T_r|}\sum_{g\in T_r}2\left(y_o(g)-t_o(g)\right)f'(net_o)X_i(g)\end{aligned} \tag{2.21}$$

Analogamente nel caso di trasduzione IO si ha

$$\Delta w_{oi} = \frac{1}{|T_r|}\sum_{g\in T_r}\sum_{v\in V_g}2\left(y_o(v)-t_o(v)\right)f'(net_o)x_i(v) \tag{2.22}$$

Per quanto riguarda invece le unità nascoste della rete, come già detto, la particolarità di questo tipo di architettura è che la loro quantità non deve essere stabilita a priori, in quanto l'algoritmo di apprendimento stesso aggiunge unità fin quando non si raggiunge la desiderata soglia di errore.

L'algoritmo può essere schematizzato in questo modo:

1. Il punto di partenza è una rete $\mathbf{N_0}$ con 0 unità nascoste e M unità di output. La si allena e si calcola per ogni unità di output il suo errore residuo su ogni grafo del dataset, $E_o(g) = (y_o(g) - d_o(g))$.

2. Se l'errore E_{tot} della rete $\mathbf{N_{i-1}}$ è maggiore di una certa soglia prefissata, si procede ad aggiungere una nuova unità nascosta i, ottenendo la rete $\mathbf{N_i}$; tale unità viene scelta in modo che la correlazione tra il suo output e l'errore residuo della rete sia massimizzata: si vuole cioè massimizzare la funzione

$$S_i = \sum_{o=1}^{M}\left|\sum_{g\in T_r}\left(E_o(g)-\bar{E}_o\right)\left(X_i(g)-\bar{X}_i\right)\right| \tag{2.23}$$

dove $\bar{E}_o$ è la media dell'errore residuo dell'unità di output e $\bar{X}_i$ è la media della funzione su tutti i grafi del training set.

Il valore $\Delta\widehat{w}_{ij}$ usato per l'aggiornamento di $\widehat{w}_{ij}$ è calcolato come

$$\begin{aligned}
\Delta\widehat{w}_{ij} &= \frac{\partial S_i}{\partial \widehat{w}_{ij}} \qquad (2.24)\\
&= \sum_{o=1}^{M} \sigma_o \sum_{g\in T_r} \frac{\partial (E_o(g)-\bar{E}_o)(X_i(g)-\bar{X}_i)}{\partial \widehat{w}_{ij}}\\
&= \sum_{o=1}^{M} \sigma_o \sum_{g\in T_r} \frac{\partial (E_o(g)-\bar{E}_o)(X_i(g)-\bar{X}_i)}{\partial X_i(g)} \cdot \frac{\partial X_i(g)}{\partial \widehat{w}_{ij}}\\
&= \sum_{o=1}^{M} \sigma_o \sum_{g\in T_r} (E_o(g)-\bar{E}_o) \cdot \sum_{v\in V_g} \frac{f'(net_i)}{k} \sum_{u\in\mathcal{N}(v)} x_j(u)
\end{aligned}$$

usando per X_i e x_i rispettivamente la formula $X_i(g) = \frac{1}{k}\sum_{v\in V} x_i(v)$ e l'equazione 2.16; σ_o è il segno del contenuto del valore assoluto nell'equazione 2.23.

Analogamente

$$\Delta\bar{w}_{ij} = \frac{\partial S_i}{\partial \bar{w}_{ij}} = \sum_{o=1}^{M} \sigma_o \sum_{g\in\mathbf{G}} (E_o(g)-\bar{E}_o) \cdot \sum_{v\in V_g} \frac{f'(net_i)}{k} I_j(v) \qquad (2.25)$$

Se la funzione di trasduzione non è scalare, allora si ha

$$S_i = \sum_{o=1}^{M} \left| \sum_{g\in T_r} \sum_{v\in V_g} \left(E_o(v)-\bar{E}_o\right)\left(x_i(v)-\bar{x}_i\right) \right| \qquad (2.26)$$

$$\Delta\widehat{w}_{ij} = \sum_{o=1}^{M} \sigma_o \sum_{g\in T_r} \sum_{v\in V_g} (E_o(v)-\bar{E}_o) f'(net_i) \sum_{u\in\mathcal{N}(v)} x_j(u) \qquad (2.27)$$

$$\Delta\bar{w}_{ij} = \sum_{o=1}^{M} \sigma_o \sum_{g\in T_r} \sum_{v\in V_g} (E_o(v)-\bar{E}_o) f'(net_i) I_j(v) \qquad (2.28)$$

3. Terminato l'allenamento della nuova unità nascosta si procede di nuovo all'allenamento del'output layer. La nuova unità appena aggiunta alla rete viene congelata: i suoi pesi non verranno più modificati nel corso dell'apprendimento, così come per tutte le altre unità nascoste. Grazie a questo, nel calcolare i nuovi pesi delle unità di output non c'è necessità di propagazione all'indietro dell'errore (come succede invece negli algoritmi di backpropagation su reti multistrato).

4. I passi descritti vengono ripetuti fino al raggiungimento dell'errore desiderato.

2.8 Statistical Relational Learning

Lo Statistical Relational Learning (SRL) [34] è il settore dell'apprendimento automatico nel quale i metodi della statistica e dell'inferenza probabilistica vengono applicati al problema dell'inferenza in domini relazionali: il dominio d'applicazione è costituito da entità, collegate tra loro in base a diverse relazioni, a costituire un grafo.

Uno dei task studiati in quest'ambito è il task di within-network learning, descritto formalmente nella sezione 2.2. Il problema di classificazione nell'ottica dello SRL è un problema di inferenza, in cui lo scopo è trovare la distribuzione di probabilità che lega ogni vertice alle possibili classi esistenti, $\mathcal{P}(v \in c_k|\mathbf{G})$: la distribuzione di probabilità può in linea teorica dipendere sia dalle proprietà locali del vertice, i suoi attributi, sia dall'influenza del resto del grafo.

In letteratura sono presenti numerosi approcci al problema della classificazione di entità connesse in una rete.

I *Probabilistic Relational Model* (PRM) [35]–[37] sono l'estensione delle reti bayesiane al dominio dei dati relazionali. Una rete bayesiana è un modello che permette di descrivere graficamente le dipendenze condizionali tra le variabili casuali di un problema. In un rete bayesiana l'insieme di variabili casuali su cui la rete è definita e le relazioni di dipendenza tra esse sono prestabilite, e questa proprietà ne impedisce l'applicazione a domini in cui sono presenti realizzazioni (entità) con configurazioni diverse tra loro, delle quali si vogliono catturare delle proprietà comuni. I Probabilistic Relational Model estendono quindi le reti bayesiane con i concetti

di individuo, o entità, proprietà dell'individuo e di relazione tra individui: dato uno schema relazionale, che descrive le classi e le relazioni che possono intercorrere fra le istanziazioni delle stesse, un PRM definisce una distribuzione di probabilità sugli attributi delle classi dello schema. Un PRM si differenzia da una rete bayesiana principalmente per due motivi: in un PRM le dipendenze condizionali tra le variabili sono definite sulle classi dello schema, in maniera generale, e possono essere applicate ad ogni specifico oggetto che appartenga a quella classe; inoltre un PRM modella esplicitamente le relazioni che intercorrono tra le classi dello schema, permettendo di inserire nel modello probabilistico la dipendenza condizionale di un attributo di un oggetto rispetto ad un altro con cui è in relazione.

Similmente, le *Relational Markov Network* (RMN) [38] estendono i concetti dei *Markov Random Field* al dominio dei dataset relazionali. I Markov Random Field (MRF) [39] , o *Markov Network*, sono, a differenza delle reti bayesiane, dei modelli probabilistici grafici indiretti discriminativi, tramite i quali è possibile rappresentare dipendenze cicliche che possono essere usate per il processo di inferenza: i MRF permettono di stimare la distribuzione di probabilità congiunta che lega le variabili casuali di un dataset, tramite l'assunzione di Markov del prim'ordine che la distribuzione di probabilità di una variabile sia condizionata direttamente solo dalle variabili casuali cui è direttamente collegata. Un modello di RMN definisce una Markov Network su dati che non sono flat, ma che sono collegati tra loro da relazioni, permettendo l'inferenza collettiva delle classi di tutti gli elementi del dataset.

Il problema dell'inferenza in domini di questo tipo è praticamente intrattabile in maniera sistematica, a causa dell'alto numero di variabili e di loro possibili configurazioni, già per reti con un piccolo numero di vertici non etichettati. Per questo motivo i principali modelli per lo SRL fanno uso di tecniche euristiche che permettono di gestire la grande complessità del dominio dei dati.

In [6] vengono applicate le tecniche dello SRL al problema della classificazione di pagine web, utilizzando non solo il contenuto testuale di ogni pagina, ma anche informazioni sulle pagine collegate ad essa tramite link. In particolare, di ogni pagina collegata viene considerata la classe, o una sua stima: il modello proposto infatti prevede l'utilizzo di un classificatore locale, il Naive Bayes, per effettuare un primo passo di classificazione dei documenti guardando solamente al testo contenuto

in essi. Dopo questa fase, ogni pagina del dataset è etichettata con una classe, reale o stimata: il passo successivo consiste nell'impiegare un classificatore relazionale, il cui compito è inferire la classe di ogni documento in base alla classe associata alle pagine ad esso collegate (nella sezione 2.8.2 sono riportati esempi di possibili classificatori relazionali). Il processo di classificazione prosegue in questa maniera, secondo l'algoritmo iterativo di *Relaxation Labeling* (sez. 2.8.3).

Un'altro approccio, chiamato *Link Based Classification*, viene proposto in [2]: l'idea alla base di questa tecnica consiste nell'utilizzare due distinti modelli di regressione logistica, uno per inferire la distribuzione di probabilità del vertice in base alle sue proprietà locali, $\mathcal{P}(v \in c_k|\mathbf{I}(v))$, e un secondo per la distribuzione $\mathcal{P}(v \in c_k|\mathcal{L}(\mathcal{N}(v)))$, dove $\mathcal{L}(\mathcal{N}(v))$ sono *link-based feature*. Le link-based feature vengono calcolate a partire dalle classi associate ad ogni vicino del vertice: in [2] vengono considerati diverse tipi di link-feature, come ad esempio la moda (il valore della classe più frequente tra i vicini), vettori binari con tanti elementi quante le classi del problema, in cui l'elemento i-esimo vale 1 se il vertice ha almeno un vicino la cui classe è c_i, e vettori in cui invece l'elemento i-esimo contiene il numero dei vicini del vertice che hanno classe c_i. Come in [6], per gestire il problema dell'inferenza collettiva delle classi di tutti i vertici non etichettati, viene utilizzato un approccio iterativo che prevede due fasi: nella prima fase, il sistema viene inizializzato tramite l'assegnazione di una classe ad ogni vertice del grafo guardando solamente agli attributi locali, mentre nella seconda fase, tramite un processo iterativo, si procede all'aggiornamento delle etichette considerando anche le link-based feature, che vengono calcolate utilizzando il target, quando disponibile, o la stima della classe dei vicini di ogni vertice.

Tutti gli approcci descritti finora sono accomunati da un'idea di base: i modelli relazionali sono costruiti sull'ipotesi che la probabilità che due entità di una rete siano collegate è molto più alta se queste entità sono simili, ovvero che è molto più probabile che esista un arco tra due vertici etichettati con la stessa classe che tra due vertici con classi diverse; questo concetto prende il nome di *omofilia*. L'ipotesi di omofilia è in effetti vera per alcuni domini di applicazione dell'in-graph learning, come le social network, ma non è detto che sia valida per ogni tipo rete: in generale la classe di un vertice può dipendere dal suo contesto (i vertici a cui è collegato sia diretttamente che indirettamente) tramite una regola diversa, che le tecniche

descritte finora potrebbero non essere in grado di cogliere.

In [40] viene proposto un approccio il cui scopo è superare il concetto di omofilia, e introdurre nel modello una diversa nozione di similarità fra vertici: la tecnica proposta sfrutta, come le precedenti, un algoritmo iterativo (l'algoritmo di Relaxation Labeling) per effettuare inferenza collettiva, ma nel modello la probabilità che un vertice v appartenga ad una certa classe viene calcolata come

$$\mathcal{P}(v \in c_k) = \frac{\sum_{u \in V} \mathcal{P}(u \in c_k)^\alpha \cdot \sigma_{u,v}^\beta}{\sum_{u \in V} \mathcal{P}(u \in c_k)^\alpha} \tag{2.29}$$

dove $\sigma_{u,v} = sim(u, v)$ è la funzione che definisce il grado di similarità di due vertici, e α e β sono due iperparametri del modello, che regolano l'influenza rispettivamente della stima dei vertici non etichettati e di $\sigma_{u,v}$. La funzione $\sigma_{u,v}$ calcola la somiglianza tra la *struttura locale* di due vertici, utilizzando una tecnica basata sui marginalized graph kernel [25] e le random walk. Il kernel descritto in [25] viene modificato per essere utilizzato per il confronto tra vertici dello stesso grafo (il kernel originario è definito tra coppie di grafi) e per permettere di introdurre nel calcolo "l'incertezza" rappresentata dai vertici non etichettati del dataset: $\sigma_{u,v}$ calcola la somiglianza tra u e v come la probabilità che con due random walk che partono da u e da v si generi la stessa sequenza di etichette.

Un'idea diversa è invece quella proposta in [41]: in questo articolo vengono riportate diverse tecniche di trasformazione di grafi al fine di migliorare le performance degli algoritmi di SRL. Le trasformazioni sono di due tipi, *node transformation* e *link transformation*, e consistono rispettivamente nell'aggiungere vertici al grafo, o nel modificare/aggiungere feature ai vertici già esistenti, e nell'aggiungere archi tra i vertici, eventualmente pesati. Ad esempio, vengono proposte delle tecniche automatiche per rilevare vertici simili (in base o alle loro feature locali o a qualche misura calcolata sul grafo, come il numero dei vicini in comune, la *cosine similarity* o la lunghezza del più piccolo percorso che li connette) e connetterli tramite un arco. Lo scopo delle tecniche automatiche di trasformazione descritte in [41] è di ottenere una nuova rete, costruita ad hoc affinchè il modello di SRL scelto ottenga dei risultati migliori rispetto a quelli sulla rete originale.

In [5] infine viene presentata una rassegna di alcuni dei principali metodi per lo SRL, e viene introdotto Netkit, un software con cui ci si propone di unificare

in un unico framework alcuni dei modelli relazionali e di inferenza collettiva dello Statistical Relational Learning. Grazie a questo framework è possibile applicare facilmente i modelli di SRL appena descritti a diversi problemi relazionali, e per questo motivo si è deciso di utilizzarlo come strumento di confronto dei risultati ottenuti dai nostri modelli (sez. 4.2).

Nello stesso articolo viene inoltre suggerito un modo per schematizzare un modello di SRL, che consiste nella sua suddivisione in tre componenti: il Local Classifier (LC), il Relational Classifier (RC) e il modulo di Collective Inference (CI). Le diverse componenti vengono descritte nel dettaglio nelle sezioni successive, dove vengono riportati anche alcuni esempi di algoritmi e modelli utilizzabili per istanziare le diverse componenti presenti anche nel framework Netkit.

2.8.1 Local Classifier

La prima componente è il classificatore locale, o *Local Classifier* (LC). Il suo compito è di inferire la classe di ogni vertice utilizzando solamente gli attributi locali del vertice stesso.

Questo modulo può essere instanziato con un qualsiasi modello "classico", ovvero per flat data, in quanto la componente strutturale del dataset non interviene in nessun modo nell'allenamento di questa componente; l'allenamento e l'inferenza possono ovviamente utilizzare solamente il sottoinsieme dei vertici noti del grafo, V_k.

La distribuzione di probabilità prodotta da questo modulo viene poi usata dagli altri due come baseline per i passi dell'apprendimento relazionale e della collective inference.

2.8.2 Relational Classifier

La seconda componente è il classificatore relazionale, o *Relational Classifier* (RC). Questa componente definisce come calcolare la classe di un vertice (o meglio, la sua distribuzione di probabilità sull'insieme delle classi) in base alle proprietà del vertice stesso e del suo contesto. Quali vertici vengano compresi nel contesto di un vertice dipende dal particolare classificatore relazionale, ma la maggior parte dei modelli di apprendimento relazionale, tra cui quelli riportati in [5], definisce

il vicinato di un vertice come l'insieme dei vertici ad esso collegati direttamente tramite un arco: si fa cioè un'assunzione markoviana del prim'ordine sul grafo $\mathbf{G}$, ipotizzando che la distribuzione di probabilità per il vertice v dipenda solamente dai suoi vicini $\mathcal{N}(v)$, ovvero

$$\mathcal{P}(v \in c_k|\mathbf{G}) = \mathcal{P}(v \in c_k|\mathcal{N}(v)) \tag{2.30}$$

Inoltre, i modelli descritti in [5] sono univariati, ovvero guardano solo al target dei vicini e non a loro eventuali label.

Nel seguito si riportano alcuni modelli di classificazione relazionale, descritti in [2], [5], [6], [11].

Weighted-Vote Relational Neighbor Classifier (WVrn)

Questo primo modello è molto semplice, e si basa fortemente sull'assunzione dell'esistenza dell'omofilia nel dominio dei dati: si assume cioè che sia molto più probabile che esista un link fra entità simili fra loro piuttosto che fra entità diverse.

La probabilità $\mathcal{P}(v \in c|\mathcal{N}(v))$ viene quindi calcolata come la media, pesata in base alle label $e_{u,v}$ degli archi del grafo, della probabilità che i vicini di v siano della classe c:

$$\mathcal{P}(v \in c|\mathcal{N}(v)) = \frac{1}{Z} \sum_{u \in \mathcal{N}(v)} e_{u,v} \cdot \mathcal{P}(u \in c|\mathcal{N}(u)) \tag{2.31}$$

dove $\frac{1}{Z}$ è un fattore di normalizzazione.

Come riportato anche in [11], per i vicini noti del vertice, ovvero quelli per cui è disponibile il target, la funzione di distribuzione di probabilità vale semplicemente 1 per la classe c associata al vertice, e 0 altrimenti. La formula precedente, considerando solo i vicini in V_k, diventerebbe quindi

$$\mathbf{P}(v \in c|\mathcal{N}(v)) = \frac{1}{Z} \sum_{u \in \mathcal{N}(v), u \in c} e_{v,u} \tag{2.32}$$

Class-Distribution Relational Neighbor Classifier (CDrn)

Questo secondo modello trasforma tutti i vertici del grafo in punti in uno spazio K-dimensionale (dove K è il numero delle classi), per poi classificarli in base alla

classe cui risultano più vicini, secondo una qualche funzione di similarità.

Per ogni vertice si costruisce un vettore K-dimensionale, $CV(v)$, chiamato class vector e definito come

$$CV(v)[k] = \sum_{u \in \mathcal{N}(v)} e_{u,v} \cdot \mathcal{P}(u \in c_k | \mathcal{N}(u)) \tag{2.33}$$

con una formula molto simile a quella usata per il modello precedente. Questo vettore rappresenta le coordinate K-dimensionali del vertice.

Per ogni classe c viene invece costruito il vettore $RV(c)$ (reference vector), rappresentante una sorta di centroide della classe in $\mathbb{R}^K$:

$$RV(c) = \frac{1}{|V_k{}^{(c)}|} \sum_{u \in V_k{}^{(c)}} CV(u) \tag{2.34}$$

costruito ovviamente tenendo conto solo dei vertici in V_k.

Definiti questi vettori, il modello calcola la distribuzione di probabilità basandosi sulla vicinanza (secondo una qualche misura di distanza, ad esempio la cosine similarity) tra il vertice e la classe:

$$\mathcal{P}(v \in c | \mathcal{N}(v)) = sim(CV(v), RV(c)) \tag{2.35}$$

Network-Only Bayes Classifier (NBC)

Questo classificatore si basa su quello presentato in [6], dal quale si differenzia per il fatto che l'eventuale informazione locale non viene utilizzata per la classificazione.

Il modello utilizza un classificatore Naive Bayes multinomiale, per cui

$$\mathcal{P}(v \in c | \mathcal{N}(v)) = \frac{\mathcal{P}(\mathcal{N}(v) | v \in c) \cdot \mathcal{P}(v \in c)}{\mathcal{P}(\mathcal{N}(v))} \tag{2.36}$$

con

$$\mathcal{P}(\mathcal{N}(v) | v \in c) = \frac{1}{Z} \prod_{u \in \mathcal{N}(v)} \mathcal{P}(u \in c(u) | v \in c)^{e_{u,v}} \tag{2.37}$$

dove $c(u)$ è la classe associata al vicino u; $\mathcal{P}(\mathcal{N}(v))$ non viene effettivamente calcolato, in quanto è costante per tutte le classi.

Network-Only Link-Based Classifier (NLB)

Il *Network-Only Link-Based Classifier* infine è una versione del modello relazionale utilizzato in [2] discusso precedentemente.

Il classificatore costruisce un vettore di link-based feature a partire dalla classe dei vicini del vertice, tramite l'utilizzo di una qualche funzione di aggregazione. La versione implementata in Netkit e riportata in [5] usa vettori contenenti, per ogni classe, il numero di vicini del vertice che appartengono a quella classe, normalizzati. Le feature sono definite come

$$\mathcal{L}(\mathcal{N}(v))_k = \sum_{u \in \mathcal{N}(v)} e_{u,v} \cdot \mathcal{P}(u \in c_k | \mathcal{N}(u)) \tag{2.38}$$

nel caso generale in cui un vertice sia collegato a vertici con target sia noto che non. Sulle feature viene poi applicato un modello di regressione logistica.

2.8.3 Collective Inference

Compito di quest'ultimo modulo è di inferire la classe per tutti i vertici in V_u, utilizzando come punto di partenza le informazioni del Local Classifier e applicando ripetutamente il modello di Relational Learning.

In [5] vengono riportati tre algoritmi con i quali è possibile istanziare questo modulo; noi ne riportiamo nel dettaglio solo due, il Gibbs Sampling (GS) e il Relaxation Labeling (RL), in quanto il terzo (l'algoritmo di Iterative Classification (IC) utilizzato in [2]) segue un approccio simile al primo algoritmo.

Gibbs Sampling (GS)

L'algoritmo GS è il seguente:

1. I vertici in V_u vengono inizializzati utilizzando il Local Classifier, associando ad ognuno un vettore $\hat{c}(v)$ di K elementi, contenente la probabilità che lega il vertice v ad ogni classe. In base a queste probabilità si esegue un sampling, selezionando una classe c_s che viene assegnata a v.

2. Si genera un ordinamento (casuale) dei vertici in V_u.

3. Seguendo l'ordinamento, per ogni $v \in V_u$:

   (a) Si genera un nuovo vettore $\hat{c}(v)$ utilizzando il Relational Classifier (ora tutti i vicini di v hanno una classe associata). Una cosa importante da notare in questo algoritmo, che lo differenzia dal successivo, è che l'aggiornamento della classe di un vertice avviene immediatamente: il modello relazionale al momento di classificare l'i-esimo secondo l'ordinamento casuale del passo precedente, utilizzerà per i vertici v_j per $j < i$ le classi stimate durante l'iterazione corrente, mentre per i vertici v_j per $j > i$ utilzzerà la stima prodotta all'iterazione precedente.

   (b) Si esegue un nuovo sampling, calcolando la nuova classe c_s di v. [4]

4. I passi precedenti vengono ripetuti per un certo numero di iterazioni (2000 iterazioni sono state usate in [5]), contando (per ogni vertice) il numero di volte che ogni classe viene assegnata al vertice: i conteggi (normalizzati) costituiranno la distribuzione di probabilità output dell'algoritmo.

Relaxation Labeling (RL)

L'algoritmo RL è il seguente:

1. I vertici in V_u vengono inizializzati utilizzando il Local Classifier per generare per ogni vertice il vettore di probabilità $\hat{c}(v)^{(0)}$.

2. Per ogni $v \in V_u$ (senza seguire nessun ordinamento) si stima un nuovo vettore $\hat{c}(v)^{(t)}$ utilizzando il Relational Classifier; il modulo per effettuare la stima utilizza per ogni $u \in \mathcal{N}(v)$ i vettori di probabilità $\hat{c}(u)^{(t-1)}$ inferiti durante la precedente iterazione.

   Il vettore $\hat{c}(v)^{(t)}$ viene calcolato come

$$\hat{c}(v)^{(t)} = \beta^{(t)} \cdot \mathcal{M}_R(\mathcal{N}(v)) + (1 - \beta^{(t)}) \cdot \hat{c}(v)^{(t-1)} \tag{2.39}$$

[4] La sostanziale differenza tra questo e l'algoritmo IC è in questo passo, in quanto quest'ultimo, ad ogni iterazione, nell'assegnare la classe ad un vertice sceglie invece la classe con la probabilità più alta.

dove $\mathcal{M}_R(\mathcal{N}(v))$ rappresenta l'applicazione del modello relazionale al vicinato di v. β è un fattore di decay tale che

$$\begin{aligned} \beta^{(0)} &= k \\ \beta^{(t)} &= \beta^{(t-1)} \cdot \alpha \ , \end{aligned} \tag{2.40}$$

con $0 < \alpha < 1$ e k costante. β viene introdotto nella formula per ridurre progressivamente ad ogni iterazione il contributo del vicinato di v, e dare sempre più peso alla predizione effettuata dal modello nell'iterazione precedente.

3. Si ripete il passo 2 per T volte. Il vettore $\hat{c}(v)^{(T)}$ conterrà la distribuzione di probabilità finale.

Mentre l'algoritmo di Gibbs Sampling utilizza un update immediato della classe associata ad ogni vertice, l'algoritmo di Relaxation Labeling aggiorna parallelamente tutte le etichette di tutti i vertici in V_u alla fine di ogni iterazione: come conseguenza di ciò il calcolo all'iterazione t utilizza solamente le stime dell'iterazione $t-1$ per tutti i vertici.

Gli algoritmi seguono due diversi approcci per gestire (eventuali) cicli di dipendenze all'interno del grafo: il primo algoritmo impone un ordinamento sui vertici e in base a questo li aggiorna sequenzialmente, mentre il secondo algoritmo ad ogni passo esegue una nuova stima basandosi su quella prodotta all'iterazione precedente.

In particolare questo secondo approccio presenta delle analogie con alcune metodologie nell'ambito delle reti neurali, come ad esempio quelle descritte nella precedente sezione per la Graph Neural Network o quella della Graph Echo State Network: anche qui viene utilizzata una tecnica iterativa, che cerca ad ogni iterazione di approssimare con più precisione i target sconosciuti. Come riportato in [5] però, la convergenza dei metodi iterativi appena descritti non è sempre assicurata, e può capitare che l'algoritmo finisca per oscillare tra due diversi stati di etichettatura del grafo; come rimedio a questo problema in [5] si ricorre alla strategia euristica del *simulated annealing*, introducendo nell'equazione di stima l'iperparametro α (eq. 2.39 e 2.40), che deve essere scelto in maniera accurata: un valore troppo alto porterebbe l'algoritmo a divergere, mentre un valore troppo basso può rendere poco

influente la componente data dall'algoritmo relazionale $\mathcal{M}_R(\mathcal{N}(v))$. Nel prossimo capitolo verranno discusse più approfonditamente somiglianze e differenze tra l'approccio appena illustrato dello SRL e quello proprio di NN4G (sez. 2.7), al fine di estendere quest'ultimo modello al task dell'in-graph learning.

Capitolo 3

Modelli

In questo capitolo verranno illustrati nel dettaglio i modelli sviluppati per estendere Neural Network for Graphs (sez. 2.7) al problema dell'in-graph learning.

Punto di partenza per lo sviluppo dei modelli descritti in seguito è stato il confronto con i metodi nati nell'ambito dello Statistical Relational Learning per questo problema, discussi ampiamente nella sezione 2.8.

Come suggerito in [5], un algoritmo di SRL per il within-network learning può essere schematicamente suddiviso in tre componenti, il classificatore locale, il classificatore relazionale e l'algoritmo di inferenza collettiva. Nel seguito si fornisce un'analisi di queste tre componenti e del loro ruolo in un modello di SRL, con lo scopo di evidenziare come siano presenti punti in comune all'approccio di NN4G, nonostante il diverso dominio d'applicazione, e quali siano invece le peculiarità di questi modelli che non sono presenti nel modello neurale.

Il classificatore locale, o Local Classifier (LC), è la componente il cui compito è di inferire la classe di appartenenza di ogni vertice del grafo utilizzando solamente le sue informazioni locali, ovvero gli attributi o feature che lo descrivono, senza avere nessuna conoscenza delle relazioni che lo legano ad altri vertici della stessa rete. In un algoritmo di SRL questa componente è solitamente utilizzata per inizializzare il processo di inferenza collettiva, per il quale è necessario che ogni vertice della rete venga etichettato con una stima della classe di appartenenza.

Anche in una rete neurale per l'on-graph learning, e in NN4G in particolare, la componente locale dei vertici del grafo ha un ruolo nel processo d'apprendimento: al

contrario del LC però, questo non è limitato all'inizializzazione del learning, come è possibile vedere dall'equazione 2.15 che definisce la codifica di un vertice in NN4G. La prima unità nascosta aggiunta alla rete, che calcola la componente $x_1(\cdot)$ dell'encoding, svolge effettivamente un ruolo simile al LC, in quanto ha a disposizione solamente le feature locali di un vertice per calcolare la prima parte della sua codifica. Nella rete neurale però, ogni componente della codifica calcolata dalle unità nascoste dipende direttamente dagli attributi locali del vertice, $\mathbf{I}(v)$, non solamente la prima. Quindi, mentre nei modelli di SRL analizzati l'informazione contenuta negli attributi locali di un vertice sembra avere un ruolo di minore importanza rispetto alla componente relazionale dei dati, in NN4G ogni unità nascosta che si aggiunge alla rete utilizza direttamente le feature locali di un vertice per calcolarne l'encoding, modificando in maniera adattiva i pesi che determinano l'effettivo contributo di questa parte di informazione sull'entità.

La componente relazionale in un sistema di SRL, il Relational Classifier (RC), ha invece il compito di determinare la classe dei vertici del grafo utilizzando le informazioni sulle relazioni che li legano. La maggior parte dei classificatori relazionali presenti in letteratura si basa fortemente su due assunzioni. Innanzitutto, si assume che nel grafo vi sia *omofilia*: l'omofilia è il principio per cui in una rete è più probabile che esista un collegamento tra due vertici "simili", che tra due vertici diversi; nella forma di omofilia considerata in questo contesto, due vertici sono ritenuti simili se appartengono alla stessa classe. I classificatori relazionali visti nella sezione 2.8.2 fanno un uso più o meno estensivo di questa assunzione (in particolare il primo, WVrn), utilizzando direttamente le informazioni sulla classe dei vertici nel vicinato per inferire il target associato al vertice da classificare. La seconda assunzione comune a tutti i classificatori relazionali analizzati nel precedente capitolo è l'assunzione di Markov del prim'ordine: data l'intrattabilità del problema generale dell'inferenza della distribuzione di probabilità congiunta $\mathcal{P}(v \in c_k|\mathbf{G})$, si assume che la distribuzione $\mathcal{P}(v \in c_k|\mathcal{N}(v))$ sia equivalente alla precedente, ovvero che la classe del vertice v non venga influenzata dai vertici ai quali non è collegato tramite un arco.

Riguardo questo secondo aspetto "relazionale" del processo d'apprendimento, l'approccio di NN4G si differenzia sotto vari punti da quello di un classificatore relazionale. Anche il modello neurale è in grado di gestire informazioni relazionali,

anzi, come discusso nella sezione 2.7, contestuali, che vanno oltre il semplice vicinato diretto di un vertice; la definizione dell'equazione di stato (eq. 2.16) combina esplicitamente i contributi degli attributi locali di un vertice con le informazioni date dalla codifica dei suoi vicini. Una prima differenza fondamentale tra i due approcci invece consiste nel fatto che nella funzione di codifica della rete neurale non vengono presi in considerazione in nessun modo i target associati ai vicini del vertice da classificare: l'informazione contestuale utilizzata nel calcolo della componente i-esima della codifica $x_i(v)$ è data dallo stato $x_1(v), \ldots, x_{i-1}(v)$, calcolato dalle precedenti unità; il target di ogni vertice viene solamente utilizzato durante la fase di training del modello, per adattare i pesi delle connessioni tra i neuroni. Per questa ragione, il modello neurale non fa nessuna assunzione sulla presenza o meno di omofilia nel grafo: la funzione di trasduzione applicata al vertice v non dipende direttamente dal target dei suoi vicini, non essendo questa informazione disponibile durante il processo di codifica.

A differenza di quanto detto per la componente RC in un sistema di SRL quindi, il concetto di omofilia nel grafo non è esplicitamente modellato dalle equazioni che definiscono la rete neurale; questo può essere, a seconda del dominio di appartenenza dei dati, sia un vantaggio che uno svantaggio. Se nel dominio è effettivamente valida l'ipotesi di omofilia, e quindi la classe di un'entità è fortemente influenzata da quella dei suoi vicini, per NN4G l'apprendimento potrebbe risultare difficoltoso, proprio per il fatto che l'informazione contestuale utilizzata non comprende esplicitamente il target dei vicini, ma tiene conto solo delle feature locali e della codifica dei vicini: il target dei vicini di un vertice non può essere utilizzato direttamente per l'inferenza. Viceversa però, il fatto che il modello non sia legato esclusivamente ad una ipotesi sulla natura dei dati, ma che piuttosto sia libero di adattarsi al problema da risolvere tramite l'uso della codifica contestuale, può rivelarsi un punto di forza in problemi in cui l'ipotesi di omofilia non sussiste. La codifica è infatti adattiva: tramite il processo di apprendimento, i pesi w che la definiscono vengono modificati per minimizzare l'errore compiuto dalla rete sui dati del training set. Grazie a questa proprietà, il modello è capace di adattarsi allo specifico problema di volta in volta, codificando l'informazione contestuale del grafo nel "modo migliore" per il particolare dataset.

Inoltre, come visto nel precedente capitolo, NN4G è in grado di apprendere

trasduzioni contestuali: la componente i-esima della codifica di un vertice dipende non solo dai vertici del suo vicinato, ma anche dai vertici nel suo contesto $\mathcal{N}^{(i-1)}(v)$. Per il modello neurale non vale quindi l'assunzione di Markov del prim'ordine, che invece come detto è largamente utilizzata dai classificatori relazionali dello SRL.

L'ultima componente di un modello di SRL è l'algoritmo di inferenza collettiva (Collective Inference, CI), che è la parte del modello che si occupa di gestire il problema della classificazione simultanea di entità di classe sconosciuta (vertici in V_u) interconnesse tra loro. Gli algoritmi relazionali citati in precedenza assumono infatti che tutti i vertici nel vicinato dell'entità da classificare siano etichettati (o che sia nota una distribuzione di probabilità che permetta di inferirne la classe), ma se due vertici in V_u sono connessi, la stima del target dell'uno influenza la stima del target dell'altro, e c'è necessità di un meccanismo che permetta di risolvere questo problema di influenza reciproca della stima. Il modulo di inferenza collettiva è solitamente un algoritmo iterativo, che ad ogni step applica il modello relazionale per ottenere la nuova stima della classe dei vertici di classe sconosciuta in base ai target noti e alla stima fatta al passo precedente.

Il meccanismo di inferenza collettiva permette, in un certo senso, al modello di SRL di superare l'ipotesi di Markov fatta dal classificatore relazionale. Il processo iterativo di aggiornamento della classe dei vertici in V_u (vertici con classe non nota) trasmette al vertice v parte dell'informazione contestuale sui vertici non direttamente connessi a lui nel grafo: la stima calcolata dal classificatore per i vertici in $\mathcal{N}(v)$ alla i-esima iterazione del processo di inferenza dipende infatti dalla stima fatta nella $(i-1)$-esima iterazione per i vertici nel suo contesto di raggio 2, i vicini dei suoi vicini, che a sua volta dipende dalla stima del passo precedente per i vertici nel contesto di raggio 3, e così via. A differenza di quanto avviene però con il processo di codifica della rete neurale, nei modelli di SRL presi in considerazione solo parte dell'informazione contestuale viene trasmessa: la stima viene infatti calcolata solamente per i vertici in V_u, che sono i soli ad essere modificati ad ogni step dell'algoritmo. Dei vertici in V_k si utilizza sempre il target già noto, e come conseguenza il sottoinsieme di vertici collegati indirettamente a v solamente attraverso un vertice in V_k non ha alcuna parte nel calcolo della stima per v.

Come accennato alla fine del precedente capitolo, questo approccio di tipo iterativo presenta alcuni aspetti negativi: affinchè la convergenza sia assicurata, è

necessario ricorrere a tecniche euristiche e alla regolarizzazione manuale di iperparametri che regolino il processo di inferenza. In NN4G questo invece non accade: il ruolo svolto dalla componente di inferenza collettiva di un sistema di SRL è integrato direttamente nella definizione del modello, insieme alle componenti relazionale e locale, grazie al procedimento costruttivo con cui la rete neurale e conseguentemente la codifica vengono determinate; il modello neurale inoltre aggiunge unità, ovvero stati, in funzione dello specifico task che sta trattando.

Nelle prossime sezioni si mostra come quanto discusso finora è stato utilizzato per derivare nuovi modelli di reti neurali che possano essere applicati a task di apprendimento in-graph.

3.1 Neural Network for In-Graph Learning

Da quanto discusso sopra emerge che il modello Neural Network for Graphs, nonostante sia definito per un differente task di apprendimento, con un diverso dominio di applicazione, possiede delle caratteristiche che lo rendono predisposto all'applicazione a task dell'in-graph learning, come illustrato in seguito.

Il primo modello che si definisce, la *Neural Network for In-Graph* (NN4inG), è quindi una prima forma parziale di adattamento di NN4G al nuovo problema.

Ricordiamo che il problema consiste nell'apprendere una funzione di trasduzione input-output isomorfa $\mathcal{T} : G \longmapsto G$ che mappi il grafo di input $\mathbf{G}(V, E)$, parzialmente etichettato, nello stesso grafo $\mathbf{G}$ in cui ad ogni vertice non etichettato viene associata una etichetta $\mathbf{t}(v)$; il grafo $\mathbf{G}$ è parzialmente etichettato, quindi l'insieme dei vertici V è ripartito nei due sottoinsiemi V_k, il sottoinsieme dei vertici etichettati, e V_u, che invece contiene i vertici per i quali il target è sconosciuto. Il modello viene allenato sull'insieme V_k, e poi può essere usato per predirre il target dei vertici in V_u.

Richiamando quanto detto nella sezione 2.7, definiamo la funzione di output del modello come

$$y_m(v) = f\left(\sum_{i=0}^{N} w_{mi} \cdot x_i(v)\right) \tag{3.1}$$

dove $m = 1, \ldots, M$; $\mathbf{w_m}$ è il vettore dei pesi che connettono l'unità di output m-esima alle N unità nascoste $x_1(\cdot), \ldots, x_N(\cdot)$; la stima effettuata dalla rete per il vertice v è data dal vettore $\mathbf{y}(v) = [y_1(v), \ldots, y_M(v)]$. Si pone infine $x_0(v) = 1 \, \forall v \in V$, per includere nell'equazione 3.1 anche il bias del neurone di output.

La funzione di codifica calcolata dalle unità nascoste della rete è definita invece come

$$x_1(v) = f\left(\sum_{l=0}^{L} \bar{w}_{1l} \cdot I_l(v)\right) \tag{3.2}$$

$$\begin{aligned} x_i(v) &= f\left(\sum_{l=0}^{L} \bar{w}_{il} \cdot I_l(v) + \sum_{j=1}^{i-1} \widehat{w}_{ij} \sum_{u \in \mathcal{N}(v)} e_{v,u} \cdot x_j(u) + \sum_{j=1}^{i-1} \widehat{w}_{ij}^{(s)} x_j(v)\right) \\ &= f(net_i) \end{aligned} \tag{3.3}$$

dove come prima L è la dimensione del vettore di feature locali dei vertici $\mathbf{I}(v)$ (input label): i pesi relativi agli attributi locali sono $\bar{w}_{i1}, \ldots, \bar{w}_{iL}$, mentre il peso $\bar{w}_{i0}$ è relativo al bias del neurone che viene incorporato nella label di ogni vertice ponendo $\forall v \in V$, $I_0(v) = 1$. La prima unità nascosta aggiunta alla rete, nell'equazione 3.2, come in NN4G, ha a disposizione solamente gli attributi locali per estrarre la prima componente della codifica di un vertice, mentre dalla seconda unità in poi si aggiunge al calcolo l'informazione contestuale sui vicini del vertice, data dalla codifica effettuata dalle precedenti unità su quei vertici (equazione 3.4). A differenza di quanto accade in un problema di on-graph learning, alcuni dei vicini di v possono non appartenere al training set (ovvero, $\mathcal{N}(v) \cap V_u \neq \emptyset$): questo però non rappresenta un problema per il calcolo dello stato di NN4inG, in quanto la funzione di codifica, per come è definita, può essere applicata sia ai vertici in V_k, che a quelli in V_u. Così facendo, si ottiene una rappresentazione dell'informazione contestuale che comprende sia i vertici noti che quelli non noti, senza bisogno di introdurre nessun tipo di meccanismo iterativo esplicito di stima nel modello neurale.

L'equazione 3.4 si differenzia inoltre dall'equazione 2.16 per due componenti: si introducono la label dell'arco $e_{v,u}$ nella sommatoria della codifica dei vicini del vertice e il nuovo termine $\sum_{j=1}^{i-1} \widehat{w}_{ij}^{(s)} x_j(v)$. Questo termine inserisce nel calcolo della codifica l'informazione esplicita sullo stato calcolato per il vertice da tutte le

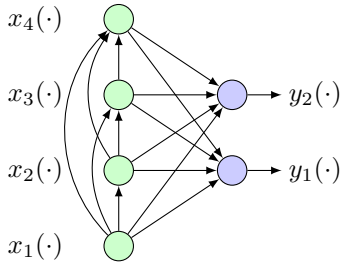


Figura 3.1: Connessioni neurali tra le unità di NN4inG, per una rete con 4 unità nascoste (in verde a sinistra) e 2 unità di output ($M = 2$, in blu a destra in figura).

precedenti unità nascoste; si è scelto di considerare il vicinato di un vertice $\mathcal{N}(v)$ come aperto, e utilizzare un distinto vettore $\mathbf{w^{(s)}}$ per pesare il contributo della codifica del vertice stesso.

L'equazione 3.4 è relativa al caso in cui si adotta una tecnica di weight sharing per la quale ad ogni arco del grafo è associato lo stesso peso, ovvero $\widehat{w}_{ij}^{(u_1,v_1)} = \widehat{w}_{ij}^{(u_2,v_2)} \, \forall i, j, u_1, v_1, u_2, v_2$. In generale, si possono adottare diverse tecniche di weight sharing per trattare diversi tipi di archi all'interno dello stesso grafo, di modo da continuare a garantire l'ipotesi di stazionarietà ma essere comunque in grado di pesare in maniera indipendente le diverse tipologie di collegamenti. Un esempio lo si ha nel caso dei grafi diretti, per i quali potrebbe essere utile distinguere gli archi entranti dagli archi uscenti di un vertice: in questo caso si hanno due tipologie di arco, e l'equazione 3.4 diventa

$$x_i(v) = f\Bigg(\sum_{l=0}^{L} \bar{w}_{il} \cdot I_l(v) + \sum_{j=1}^{i-1} \widehat{w}_{ij}^{in} \sum_{u \in \mathcal{P}(v)} e_{u,v} \cdot x_j(u) + \\ + \sum_{j=1}^{i-1} \widehat{w}_{ij}^{out} \sum_{u \in \mathcal{S}(v)} e_{v,u} \cdot x_j(u) + \sum_{j=1}^{i-1} \widehat{w}_{ij}^{(s)} x_j(v)\Bigg) \tag{3.4}$$

in cui si utilizzano due distinti vettori $\widehat{\mathbf{w}}^{\mathbf{in}}$ e $\widehat{\mathbf{w}}^{\mathbf{out}}$ per pesare i contributi rispettivamente dei predecessori e dei successori di ogni vertice.

Ogni unità nascosta i ha quindi $L + 2(i-1) + 1$ pesi ($L + 3(i-1) + 1$ nel caso dell'equazione 3.4), contenuti nei vettori $\overline{\mathbf{w}}, \widehat{\mathbf{w}}, \widehat{\mathbf{w}}^{(s)}$, mentre ogni output unit ha, come in NN4G, $N + 1$ pesi che regolano le connessioni ad ognuna delle unità

nascoste della rete (figura 3.1 e equazione 3.1) che l'algoritmo di apprendimento può modificare durante il training.

La correlazione dell'errore della rete con l'output della nuova unità nascosta vale quindi

$$S_i = \sum_{o=1}^{M} \left| \sum_{v \in V_k} \left(E_o(v) - \bar{E}_o\right) \left(x_i(v) - \bar{x}_i\right) \right| \tag{3.5}$$

mentre le formule per l'update dei pesi delle unità restano praticamente uguali a quelle derivate nella sezione 2.7, con l'aggiunta della formula per l'aggiornamento dei pesi $\widehat{\mathbf{w}}^{(s)}$:

$$\Delta\widehat{w}_{ij} = \sum_{o=1}^{M} \sigma_o \sum_{v \in V_k} (E_o(v) - \bar{E}_o) f'(net_i) \sum_{u \in \mathcal{N}(v)} e_{v,u} \cdot x_j(u) \tag{3.6}$$

$$\Delta\bar{w}_{il} = \sum_{o=1}^{M} \sigma_o \sum_{v \in V_k} (E_o(v) - \bar{E}_o) f'(net_i)\, I_l(v) \tag{3.7}$$

$$\Delta\widehat{w}_{ij}^{(s)} = \sum_{o=1}^{M} \sigma_o \sum_{v \in V_k} (E_o(v) - \bar{E}_o) f'(net_i)\, x_j(v) \tag{3.8}$$

Come già detto, solo i vertici in V_k, per i quali è noto il target, intervengono esplicitamente nell'apprendimento, essendo gli unici sui i quali è possibile valutare le funzioni di errore E_{tot} e E_o. I vertici in V_k costituiscono il training set del problema ma, contrariamente a quanto accade in un problema non relazionale, in questo tipo di task l'algoritmo d'apprendimento ha a disposizione anche alcune informazioni sui vertici che non appartengono a questo insieme: le informazioni relazionali che connettono tra loro i vertici in V_k e V_u non possono essere tralasciate dal modello, in quanto costituiscono un'importante parte dell'informazione sulla struttura della rete. La rete neurale appena definita può incorporare questa informazione in maniera semplice grazie alla codifica: il modello apprende sull'errore commesso sui vertici in V_k, per poi applicare la funzione di codifica appresa per calcolare lo stato di ogni vertice di V, compresi quelli in V_u.

3.2 Neural Network for In-Graph con target

Come discusso ampiamente nell'introduzione al capitolo, dal confronto con i modelli di classificazione relazionale dello Statistical Relational Learning appare evidente come parte dell'approccio di questi modelli si basi sull'assunzione dell'esistenza di omofilia all'interno del grafo. Si assume cioè che se due entità sono in relazione (ovvero, connesse da un arco) allora è molto probabile che esse appartengano alla stessa classe: nella maggior parte dei modelli analizzati (sez. 2.8.2) l'informazione sul target dei vicini compare infatti esplicitamente nel calcolo della distribuzione di probabilità per il vertice corrente. Ad esempio si è visto come uno dei più semplici modelli relazionali possibili, descritto in [11], basandosi esclusivamente sull'assunzione dell'esistenza di omofilia nel grafo, calcola la probabilità dell'appartenenza di un vertice a una data classe come la somma dei suoi vicini che appartengono a quella classe, pesata con la label dell'arco (equazione 2.32).

Se l'assunzione risulta vera per la rete su cui si vuole applicare il modello, un classificatore che sfrutti direttamente l'informazione sul target dei vicini di un vertice per l'inferenza sarà sicuramente avvantaggiato rispetto ad un altro che non ha accesso diretto a questa informazione, come la rete neurale descritta nella precedente sezione.

Per questo motivo si è scelto di estendere il modello descritto sopra affinchè comprenda in maniera esplicita le informazioni sul target dei vicini, introducendo un nuovo termine nella funzione che calcola la codifica di un vertice. La prima unità nascosta aggiunta alla rete calcola ora la sua codifica come

$$x_1(v) = f\left(\sum_{l=0}^{L} \bar{w}_{1l} \cdot I_l(v) + \sum_{m=1}^{M} \widetilde{w}_{1m} \sum_{u \in \mathcal{N}(v) \cap V_k} e_{v,u} \cdot t_m(u)\right) \tag{3.9}$$

Oltre agli attributi locali di un vertice, in questo nuovo modello $x_1(\cdot)$ incorpora anche una parte di informazione sui suoi vicini, tramite l'introduzione esplicita del loro target nell'equazione. Analogamente, la funzione di codifica per tutte le successive unità aggiunte alla rete diventa

$$x_i(v) = f\Bigg(\sum_{l=0}^{L} \bar{w}_{il} \cdot I_l(v) + \sum_{j=1}^{i-1} \widehat{w}_{ij} \sum_{u \in \mathcal{N}(v)} e_{v,u} \cdot x_j(u) + \sum_{j=1}^{i-1} \widehat{w}_{ij}^{(s)} x_j(v) + \\ + \sum_{m=1}^{M} \widetilde{w}_{im} \sum_{u \in \mathcal{N}(v) \cap V_k} e_{v,u} \cdot t_m(u)\Bigg) \tag{3.10}$$

Il nuovo termine $\sum_{u \in \mathcal{N}(v) \cap V_k} e_{v,u} \cdot t_m(u)$ somma, per tutti i vicini u del vertice v, il valore del target $t_m(u)$, pesato con la label dell'arco $e_{v,u}$, in maniera analoga a quanto avviene in un modello relazionale: immaginando ad esempio un task di classificazione binario in cui $M = 1$ e $t_1(u) \in \{-1, 1\}$, la sommatoria avrà un valore alto e positivo se il vertice ha molti vicini appartenenti alla classe $\{1\}$, o un valore negativo se viceversa la maggior parte dei vicini del vertice sono nella classe $\{-1\}$: l'output delle unità nascoste dipende direttamente dal target dei vicini di ogni vertice.

A differenza di quanto accade nei modelli relazionali citati però, il target dei vicini non interviene in maniera diretta per determinare la classe del vertice, ma insieme a tutte le altre componenti viene utilizzato per il calcolo della codifica associata al vertice, che viene poi a sua volta usata per determinare l'output della rete. Il modello proposto non si basa solamente sul principio dell'omofilia, ma eredita tutte le caratteristiche del precedente, tra cui la codifica contestuale e adattiva: i due contributi, il target e la codifica dei vicini, sono entrambi presenti, e il modello può adattare tramite l'apprendimento i pesi che determinano l'importanza effettiva che questi assumono.

Infatti, un'ulteriore differenza tra i due approcci sta nel fatto che, mentre nel modello relazionale i target vengono pesati solamente dalle label degli archi, che sono fisse, nel modello neurale la sommatoria dei target dei vicini è pesata con i pesi $\widetilde{\mathbf{w}}$ che, come tutti gli altri pesi della rete, sono adattivi: l'algoritmo di apprendimento interviene sul loro valore, cercando di massimizzare la correlazione tra l'output dell'unità nascosta e l'errore della rete. Come risultato di tutto ciò si ha che NN4inG è dotata di una maggiore flessibilità, rispetto a modelli relazionali di questo tipo.

L'output della rete è definito come per il modello precedente dall'equazione 3.1. La regola per l'aggiornamento dei nuovi pesi viene ricavata in maniera del tutto

analoga alle precedenti, e vale

$$\Delta\widetilde{w}_{ij} = \sum_{o=1}^{M} \sigma_o \sum_{v \in V_k} (E_o(v) - \bar{E}_o) f'(net_i) \sum_{u \in \mathcal{N}(v)} e_{v,u} \cdot t_j(u) \tag{3.11}$$

Esattamente come per il precedente modello, il learning avviene solo per i vertici in V_k, ma la funzione di codifica viene poi applicata ad ogni vertice del grafo. Va inoltre specificato però che, come si vede dall'equazione 3.10, in questo modello solamente i vicini del vertice corrente che appartengono al training set V_k possono contribuire alla sommatoria dei target, in quanto solo per loro è noto il valore del vettore $\mathbf{t}(v)$.

3.3 Neural Network for In-Graph con target e stima

Il modello proposto nella precedente sezione introduce esplicitamente nella rete neurale l'informazione sul target dei vertici in V_k (equazione 3.10); questa informazione è però disponibile solo per parte della rete, in quanto per i vertici in V_u il target è sconosciuto. Il fatto che l'informazione sia parziale può rappresentare un problema per il modello, specie in quei casi in cui la grande maggioranza dei vertici ha target sconosciuto: in questi casi il contributo del nuovo termine potrebbe risultare minimo, oppure nel caso peggiore essere di disturbo per il processo d'apprendimento, se il target dei pochi vicini etichettati di un vertice è tale da fornire informazioni "erronee" e trarre in inganno il meccanismo di stima.

Come visto nella sezione 2.8, nei modelli dello Statistical Relational Learning il problema dei vertici con target non noto viene risolto con un approccio di tipo iterativo, tramite algoritmi di inferenza collettiva: la distribuzione di probabilità del target dei vertici nel test set viene approssimata con quella stimata dallo stesso classificatore relazionale durante la precedente iterazione; in questo modo, ogni vertice può essere etichettato con il rispettivo target (o con la sua stima).

La Neural Network for In-Graph presenta un'architettura costruttiva, e al momento dell'allenamento e inserimento di una nuova unità nascosta il resto della rete,

ovvero le precedenti unità nascoste e l'output layer, è congelato: lo stato calcolato dall'hidden layer è fissato, e di conseguenza anche l'output dell'intera rete è un valore noto, che non cambia durante il training della nuova unità. Grazie a questa caratteristica, è possibile trasportare l'idea dello SRL descritta prima nel nostro modello, senza nessun bisogno di introdurre ulteriori meccanismi di iterazione: quando una nuova unità viene allenata e inserita nella rete è possibile utilizzare, per i vicini di un vertice in V_u, che non hanno target noto, l'output della rete stessa al passo precedente, ovvero prima dell'inserimento dell'unità nascosta corrente.

Definiamo l'output del nuovo modello come $y_m(v)$, dove

$$y_m^{(i)}(v) = f\left(\sum_{j=0}^{i} w_{mj}^{(i)} \cdot x_j(v)\right) \tag{3.12}$$

$$y_m(v) = y_m^{(N)}(v) \tag{3.13}$$

per $i = 1, \ldots, N$ e $m = 1, \ldots, M$[1].

Ponendo

$$t_m^{(i-1)}(u) = \begin{cases} t_m(u) & \text{se } u \in V_k \\ y_m^{(i-1)}(u) & \text{se } u \in V_u \end{cases} \tag{3.14}$$

la nuova funzione di codifica diventa

$$\begin{aligned} x_i(v) = f\Bigg(\sum_{l=0}^{L} \bar{w}_{il} I_l(v) + \sum_{j=1}^{i-1} \widehat{w}_{ij} \sum_{u \in \mathcal{N}(v)} e_{v,u} \cdot x_j(u) + \sum_{j=1}^{i-1} \widehat{w}_{ij}^{(s)} x_j(v) + \\ + \sum_{m=1}^{M} \widetilde{w}_{im} \sum_{u \in \mathcal{N}(v)} e_{v,u} \cdot t_m^{(i-1)}(u) + \sum_{m=1}^{M} \widetilde{w}_{im}^{(s)} y_m^{(i-1)}(v)\Bigg) \end{aligned} \tag{3.15}$$

Come per lo stato, si aggiunge alla formula, sfruttando sempre la stima del target fatta dalla rete al passo precedente, anche la stima del target per il vertice stesso

[1]Nel caso di task di classificazione $y_j^{(i)}(u)$ non contiene la classe del vertice u (ad esempio 1 o -1), ma la stima fatta dalle unità di output prima dell'applicazione della funzione di soglia (nel caso di un task binario ad esempio, $y_j^{(i)}(u) \in [-1, 1]$).

v, pesata con un vettore specifico di pesi $\widetilde{\mathbf{w}}^{(\mathbf{s})}$; in questo modello l'unità nascosta i-esima avrà $L + 1 + 2(i-1) + 2M$ pesi.

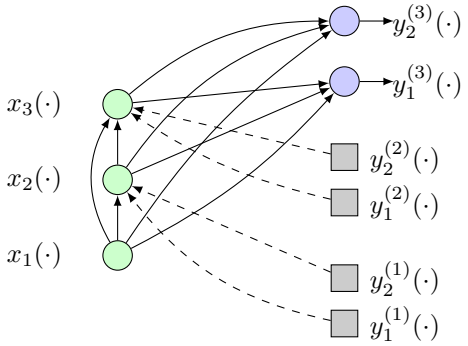


Figura 3.2: Connessioni neurali tra le unità di NN4inG con target e stima, per una rete con 3 unità nascoste (in verde a sinistra) e 2 unità di output ($M = 2$, in blu in alto a destra in figura).

In figura 3.2 è riportato un esempio di rete neurale che realizza il modello descritto, in cui sono presenti 2 unità di output ($M = 2$) e 3 unità nascoste ($N = 3$). L'output della rete in figura è dato dalle 2 unità di output $y_1^{(3)}$ e $y_2^{(3)}$, ma, a differenza del precedente modello, si mantiene una copia della stima effettuata dalla rete dopo l'inserzione di ogni unità nascosta ($y_1^{(2)}$, $y_2^{(2)}$ e $y_1^{(1)}$, $y_2^{(1)}$ in figura), che viene utilizzata dalla successiva unità per il calcolo della nuova componente della codifica, secondo quanto definito nell'equazione 3.15. La stima utilizzata resta fissa e costante sia durante l'apprendimento che successivamente: l'unità nascosta una volta allenata e inserita nella rete, viene congelata come tutte le altre, e in particolare quindi la stima utilizzata dall'unità i-esima non cambia con il successivo inserimento e allenamento di nuove unità nascoste.

I pesi $\widetilde{\mathbf{w}}$ vengono aggiornati come indicato nell'equazione 3.11, e analogamente per i pesi $\widetilde{\mathbf{w}}^{(\mathbf{s})}$ vale

$$\Delta\widetilde{w}_{ij}^{(s)} = \sum_{o=1}^{M} \sigma_o \sum_{v \in V_k} (E_o(v) - \bar{E}_o) f'(net_i) y_j^{(i-1)}(v) \tag{3.16}$$

Inoltre, se la tecnica di weight sharing viene modificata, ad esempio come detto precedentemente per trattare grafi diretti, le modifiche applicate nell'equazione di

codifica (3.4) ai pesi $\widehat{\mathbf{w}}$ si applicano in maniera del tutto analoga all'equazione di questo e del precedente modello, sia ai pesi $\widehat{\mathbf{w}}$ che ai pesi $\widetilde{\mathbf{w}}$. La sommatoria sulla stima dei vicini viene quindi sostituita dalle due sommatorie

$$\sum_{m=1}^{M} \widetilde{w}_{ij}^{in} \sum_{u \in \mathcal{P}(v)} e_{u,v} \cdot t_m^{(i-1)}(u) + \sum_{m=1}^{M} \widetilde{w}_{ij}^{out} \sum_{u \in \mathcal{S}(v)} e_{v,u} \cdot t_m^{(i-1)}(u) \tag{3.17}$$

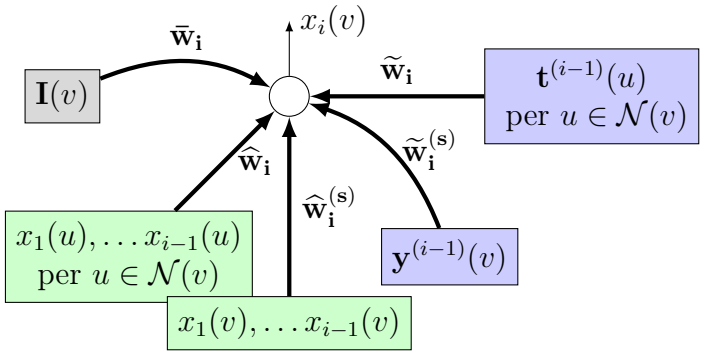


Figura 3.3: Unità nascosta di NN4inG con target e stima.

3.4 Neural Network for In-Graph con target e stima distinti

Il modello introdotto nella sezione 3.3 nasce dall'idea di includere nella codifica i-esima di un vertice anche informazioni sul probabile target dei suoi vicini in V_u, utilizzando a questo scopo la stima calcolata dalla stessa rete neurale dopo l'introduzione della precedente unità nascosta, ovvero $\mathbf{y}^{(i-1)}(v)$ (equazione 3.15).

La codifica definita in 3.15 non tiene però conto della differenza sostanziale che esiste tra il contributo dei vicini con il target e quello dei vicini senza target: un unico vettore $\widetilde{\mathbf{w}}$ pesa infatti sia il contributo del target dei vicini del vertice che sono in V_k sia la stima del target dei vicini in V_u. Un possibile problema di questo è che, mentre per i vicini in V_k l'input che si presenta al neurone è il vero target $\mathbf{t}(u)$, una informazione "certa" su quei vertici, per quelli in V_u l'input è costituito da una stima fatta dal modello stesso, che può o non può essere corretta: utilizzare un unico vettore $\widetilde{\mathbf{w}}$ per pesarli entrambi equivale in un certo senso a dire che la

stima fatta dalla rete ha lo stesso valore del target effettivo di un vertice. Ma nella realtà questa assunzione può risultare troppo semplicistica: la stima calcolata dalla rete può essere molto lontana dal target reale, e il modello descritto sopra non può adattare conseguentemente i pesi $\widetilde{\mathbf{w}}$ proprio perchè non può distinguere tra i due diversi tipi di input. Il problema può presentarsi ad esempio durante le prime fasi dell'allenamento, in cui la probabilità che la stima del passo precedente sia sbagliata è molto alta: il rischio è che le informazioni ricavate dai vicini portino fuori strada l'algoritmo d'apprendimento, e piuttosto che un miglioramento delle performance si ottenga il risultato contrario.

Un modo per risolvere questo problema potrebbe essere quello di utilizzare un iperparametro che permetta di regolare l'influenza della stima nel calcolo della codifica di un vertice, di modo da regolarizzarne l'impatto sull'apprendimento. Un altro modo, che è quello che si è scelto di sperimentare, consiste invece nello sfruttare la proprietà di adattività della rete neurale stessa, per far sì che essa determini durante l'apprendimento il giusto peso da dare alla stima: per fare ciò, si modifica l'equazione di codifica in questo modo

$$\begin{aligned} x_i(v) = f\Bigg(\sum_{l=0}^{L} \bar{w}_{il} I_l(v) + \sum_{j=1}^{i-1} \widehat{w}_{ij} \sum_{u \in \mathcal{N}(v)} e_{v,u} \cdot x_j(u) + \sum_{j=1}^{i-1} \widehat{w}_{ij}^{(s)} x_j(v) + \\ + \sum_{m=1}^{M} \widetilde{w}_{im}^{(s)} y_m^{(i-1)}(v) + \sum_{m=1}^{M} \widetilde{w}_{im}^{(t)} \sum_{u \in \mathcal{N}(v) \cap V_k} e_{v,u} \cdot t_m(u) + \\ + \sum_{m=1}^{M} \widetilde{w}_{im}^{(e)} \sum_{u \in \mathcal{N}(v) \cap V_u} e_{v,u} \cdot y_m^{(i-1)}(u) \Bigg) \end{aligned} \tag{3.18}$$

separando il contributo del target dei vicini noti, pesato con il vettore $\widetilde{\mathbf{w}}^{(\mathbf{t})}$, dal contributo della stima dei vicini non noti, pesato con il vettore $\widetilde{\mathbf{w}}^{(\mathbf{e})}$. La funzione di output e la funzione di codifica $x_1(\cdot)$ della prima unità nascosta restano le stesse definite rispettivamente nelle equazioni 3.12 e 3.9. In questo modello l'unità nascosta i-esima ha $L + 1 + 2(i-1) + 3M$ pesi, mentre l'unità di output come prima ha $N + 1$ parametri w.

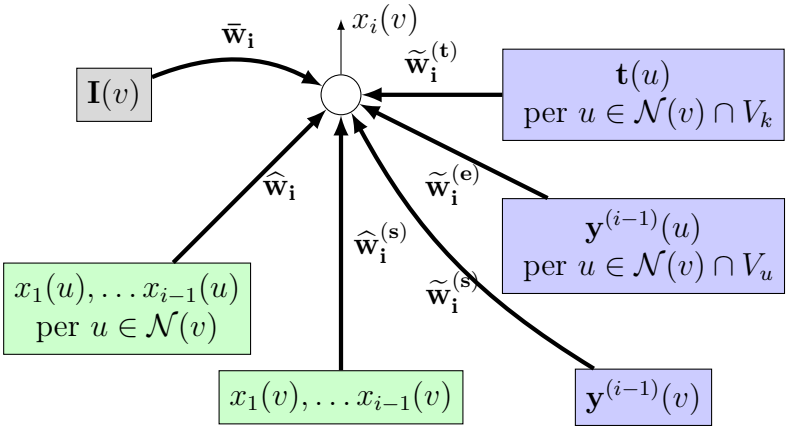


Figura 3.4: Unità nascosta di NN4inG con pesi distinti per target e stima.

3.5 Architettura neurale modificata

I modelli descritti nelle precedenti sezioni (sez. 3.1, 3.2, 3.3 e 3.4) utilizzano due diversi tipi di informazione contestuale: la codifica dei vicini del vertice, che permette come spiegato nella sezione 3.1 di avere una rappresentazione del contesto di ogni vertice adattiva e specifica per il problema, e il loro target (o una sua stima). Viste le performance ottenute da semplici modelli relazionali solamente basati sull'omofilia [5], [11], una domanda che potrebbe sorgere è se, una volta inserita nel modello l'informazione esplicita sul target dei vicini di un vertice, sia ancora necessario utilizzare la più complessa informazione di stato data dalla codifica.

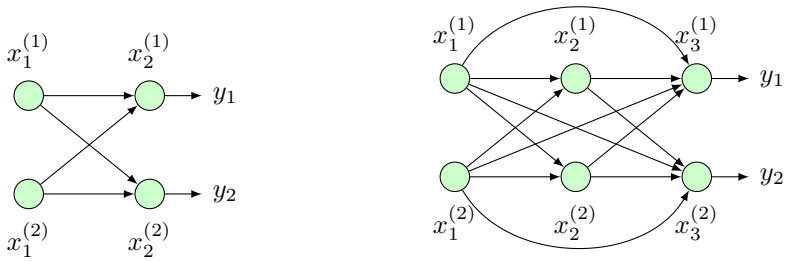
Per rispondere a questa domanda, si è quindi voluto provare a modificare l'architettura della rete neurale, semplificandola, in modo da ottenere un modello che utilizzasse solamente il target e la stima dei vicini per calcolare l'output associato ad ogni vertice; modificare il modello in questo modo implica una semplificazione dell'architettura della rete neurale in quanto, eliminando la codifica, permette l'eliminazione di una parte delle unità della rete e di un livello dell'architettura neurale. Questa modifica può essere interpretata in due modi diversi, che portano poi a due diverse architetture neurali: una prima alternativa consiste nell'eliminare lo strato di unità di output e utilizzare direttamente la funzione di codifica delle unità x_i, così come è definita per i precedenti modelli, per calcolare l'output della rete. In questo modo quindi la codifica non viene in realtà eliminata, ma va a coincidere direttamente con la stima calcolata dal modello. Viceversa, si può invece pensare

di eliminare dalla rete lo strato di unità x_i, eliminando la codifica, e ottenendo una rete con le sole unità di output. Nel seguito si discutono entrambi i modelli che derivano da queste due idee.

Volendo realizzare la prima alternativa descritta sopra, si elimina lo strato di unità di output, e si pone

$$y_m^{(i)}(v) = x_i^{(m)}(v) \quad \text{per } m = 1 \dots M \tag{3.19}$$

In particolare $y_m(v) = y_m^{(N)}(v) = x_N^{(m)}(v)$: in questo modello la componente m-esima dell'output della rete è data direttamente dalla funzione di codifica $x_N^{(m)}(\cdot)$. L'apice m indica che l'unità $x_i^{(m)}$ è l'm-esima dello strato di unità i-esimo: dovendo la codifica coincidere con la stima del target dei vertici, una necessaria modifica dell'architettura consiste nell'aggiungere non più una sola unità nascosta x_i ad ogni passo di costruzione della rete, ma M, dove M è la dimensione del target da stimare $\mathbf{t}(v)$.



(a) Rete neurale con $N = 2$ strati.

(b) Rete neurale dopo l'inserimento di un nuovo strato di unità x_3.

Figura 3.5: Connessioni neurali tra le unità di NN4inG con le sole unità di codifica per una rete con output di dimensione 2 ($M = 2$) con 2 strati di unità ($N = 2$, figura 3.5a) e dopo l'inserimento di un nuovo strato di unità ($N = 3$, figura 3.5b).

In figura 3.5 sono illustrate le connessioni neurali per una rete con l'architettura appena descritta: le unità di output non sono più presenti, e la stima della rete è data direttamente dalla codifica calcolata dall'ultimo strato di unità nascoste; confrontando il modello con lo schema in figura 3.2, si nota come nella nuova architettura non ci sia più necessità di mantenere una copia della stima fatta dalla rete ad ogni passo ($y_m^{(i)}$ in figura 3.2), in quanto questa coincide con la codifica calcolata

dalle unità $x_i^{(m)}$: quando in figura 3.5b si aggiunge un nuovo strato di unità, queste vengono connesse a tutti i precedenti layer presenti nella rete e congelati, che forniscono alle nuove unità la stima calcolata ad ogni step di costruzione della rete.

La funzione di codifica e di output del modello viene quindi calcolata con una versione modificata dell'equazione di codifica 3.10, che tiene conto della diversa architettura della rete:

$$x_i^{(m)}(v) = f\Bigg(\sum_{l=0}^{L} \bar{w}_{il} I_l(v) + \sum_{j=1}^{i-1}\sum_{k=1}^{M} \widehat{w}_{ijk} \sum_{u\in\mathcal{N}(v)} e_{v,u} x_j^{(k)}(u) + \sum_{j=1}^{i-1}\sum_{k=1}^{M} \widehat{w}_{ijk}^{(s)}\, x_j^{(k)}(v) + \\ + \sum_{k=1}^{M} \widetilde{w}_{ik} \sum_{u\in\mathcal{N}(v)\cap V_k} e_{v,u} t_k(u)\Bigg) \tag{3.20}$$

L'output dell'unità $x_i^{(m)}(v)$ rappresenta ora contemporaneamente sia lo stato che la stima per il vertice v al passo i: dall'equazione 3.20 e dallo schema in figura 3.5b appare evidente una grande differenza tra questo e i precedenti due modelli (sez. 3.3 e 3.4) che utilizzano la stima per il calcolo della codifica: in questi modelli infatti l'unica stima considerata era quella al passo precedente (come si vede in figura 3.2), mentre in questo nuovo modello si tengono in considerazione contemporaneamente tutte le stime di ogni strato di unità nascoste aggiunto alla rete; in questo modello inoltre, per i vertici in V_k, ogni neurone ha in input sia le $i-1$ stime prodotte dalla rete, sia il target effettivo del vertice. L'input del singolo neurone risulta per questi motivi più complesso che nei modelli precedenti (figura 3.6): l'i-esima unità ha infatti $L + 1 + 2M(i-1) + M$ parametri liberi w.

L'architettura della rete resta costruttiva come per i precedenti modelli: l'algoritmo d'apprendimento aggiunge unità fino al raggiungimento della soglia desiderata sull'errore E_{tot}, con la differenza che ad ogni step, invece di aggiungere una sola unità, l'algoritmo ne aggiunge M. L'allenamento delle unità $x_i^{(m)}$ inoltre non viene più effettuato tramite correlazione, come per le unità nascoste di NN4G (sez. 2.7) e tutti i modelli discussi nelle sezioni precedenti, ma le nuove unità vengono allenate direttamente tramite discesa del gradiente, come accade per le output unit negli altri modelli.

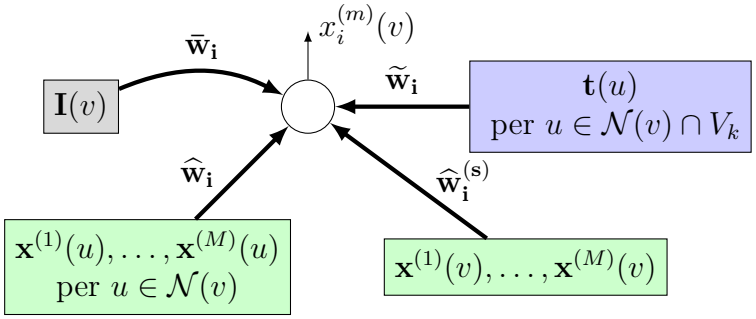


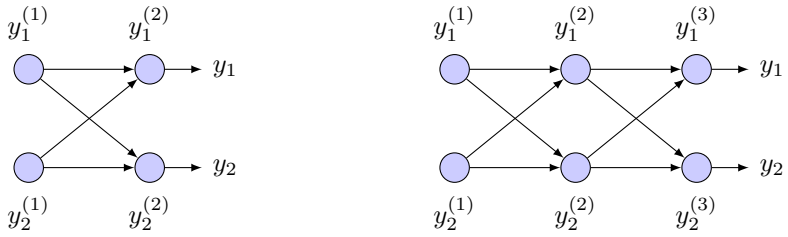
Figura 3.6: Unità di NN4inG per l'architettura con le sole unità di stato. Con $\mathbf{x}^{(m)}(u)$ si indica il vettore $x_1^{(m)}(u), \cdots, x_{i-1}^{(m)}(u)$.

Come accennato all'inizio della sezione, un'altra strada che è possibile seguire consiste nell'eliminare lo strato di unità nascoste x_i, e di conseguenza la codifica, dalla rete neurale. Nel modello risultante da questa modifica, come prima, ad ogni step di costruzione della rete si aggiunge un vettore di M unità, che però in questo caso sono unità di output, più semplici, che hanno in input solamente gli attributi locali di un vertice, $\mathbf{I}(v)$, la stima calcolata dalla rete per quel vertice e per i suoi vicini al passo precedente e il target dei vicini in V_k. L'equazione che descrive la funzione calcolata da questi neuroni è la seguente:

$$y_m^{(i)}(v) = f\Bigg(\sum_{l=0}^{L} \bar{w}_{il} I_l(v) + \sum_{k=1}^{M} \widetilde{w}_{ik}^{(e)} \sum_{u \in \mathcal{N}(v)} e_{v,u} y_k^{(i-1)}(u) + \sum_{k=1}^{M} \widetilde{w}_{ik}^{(s)}\, y_k^{(i-1)}(v) + \\ + \sum_{k=1}^{M} \widetilde{w}_{ik}^{(t)} \sum_{u \in \mathcal{N}(v)} e_{v,u} t_k(u)\Bigg) \tag{3.21}$$

La funzione calcolata dalle output unit è ovviamente diversa da quella dei modelli precedenti (equazione 3.12): l'output della rete per un vertice v veniva prima calcolato sulla base della codifica fatta dalle unità nascoste, e non essendo queste più presenti nella rete, l'input dell'unità $y_m^{(i)}$ va obbligatoriamente modificato per includere le informazioni locali e relazionali dei vertici in maniera diretta.

In figura 3.7 è illustrato lo schema delle connessioni neurali per questa nuova architettura. Confrontandolo con quello in figura 3.5 sono evidenti le differenze tra i due modelli: l'architettura ottenuta utilizzando la funzione di codifica come



(a) Rete neurale con $N = 2$ strati.

(b) Rete neurale dopo l'inserimento di un nuovo strato di unità $y^{(3)}$.

Figura 3.7: Connessioni neurali tra le unità di NN4inG con le sole unità di output per una rete con output di dimensione 2 ($M = 2$) con 2 strati di unità ($N = 2$, figura 3.7a) e dopo l'inserimento di un nuovo strato di unità ($N = 3$, figura 3.7b).

funzione di output risulta più complessa, con un maggior numero di parametri liberi, mentre con la seconda il modello risulta effettivamente semplificato. Ogni unità presenta solamente $L + 1 + 3M$ parametri liberi, dove sia L che M sono dati dal problema: L è la dimensione del vettore di feature locali, e M è la dimensione del target dell'apprendimento. In un problema di classificazione ad esempio M è uguale al numero delle classi, che è ragionevole supporre siano poche (nella maggior parte dei dataset utilizzati per i nostri esperimenti il task è di classificazione binaria, con $M = 1$).

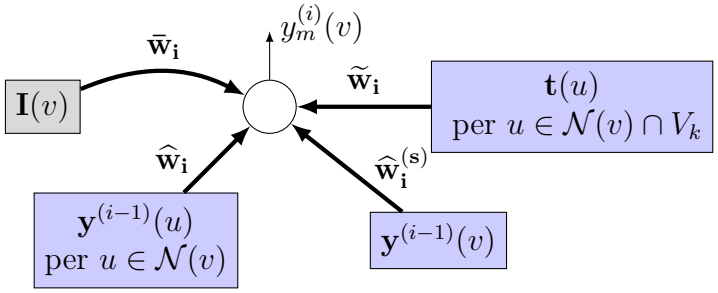


Figura 3.8: Unità di NN4inG per l'architettura con le sole unità di output.

Il modello risulta più semplice dei precedenti, in quanto con la nuova architettura la rete contiene un numero minore di connessioni neurali: supponendo di avere una rete con N strati di unità, il modello in sezione 3.4 ha un totale di $\sum_{i=1}^{N}(L + 1 + 2(i-1)+3M) = N(L+1+3M)+N^2$ parametri liberi, più gli $M(N+1)$ parametri che regolano l'input di ogni unità di output. La prima variante architetturale ha

invece $M(\sum_{i=1}^{N} L + 1 + 2M(i-1) + M) = NM(L+1) + M^2N^2$ parametri, mentre la seconda ne ha $M(\sum_{i=1}^{N} L + 1 + 3M) = NM(L+1+3M)$. Rispetto a N, l'ultimo modello è l'unico a non avere un numero quadratico di connessioni.

Inoltre, per come è definito, il modello appena descritto è quello che più si avvicina all'approccio relazionale basato sull'omofilia: la stima effettuata dalla rete neurale dipende solamente dal target dei vicini e dalla stima fatta allo step precedente di costruzione; non esiste più il concetto intermedio di codifica contestuale, ma la rete si basa totalmente sulla classe dei suoi vicini per la predizione del target di un vertice. A differenza di quanto avviene in un modello relazionale però, la stima viene calcolata non solo per i vertici in V_u, ma anche per quelli in V_k (eq. 3.21).

Capitolo 4

Risultati Sperimentali

In questo capitolo vengono riportati i risultati sperimentali dell'applicazione dei nuovi modelli per la Neural Network for In-Graph ad alcuni dataset.

Nella prima sezione, come collaudo del framework proposto, si riportano i risultati di esperimenti preliminari dell'implementazione del modello base di NN4inG (sez. 3.1) su due dataset classici dell'on-graph learning: l'Alkanes dataset (sez. 4.1.1) e il PTC dataset (sez. 4.1.2), entrambi appartenenti al mondo della chimica, il primo con un task di regressione, il secondo con un task di classificazione.

Nella sezione 4.2 vengono invece discussi i risultati dei diversi modelli proposti per l'in-graph learning su tre dataset. Per confronto con [5], sono stati scelti due dataset trattati in questo articolo, IMDB (sez. 4.2.1) e Cora (sez. 4.2.2), classici dataset per lo Statistical Relational Learning. Il terzo e ultimo dataset utilizzato, Webspam (sez. 4.2.3), fa invece parte dei dataset messi a disposizione durante la Webspam Challenge (`http://webspam.lip6.fr/wiki/pmwiki.php?n=Main.PhaseII`).

4.1 On Graph Learning

In questa sezione si riportano brevemente, come collaudo del framework proposto anche per task appartenenti all'on graph learning, i risultati dell'implementazione del modello base di NN4inG su due dataset appartenenti al mondo della chimica,

spesso usati come benchmark per lo sviluppo di nuovi modelli nell'ambito dei domini strutturati.

Il primo dataset, l'Alkanes dataset, contiene un insieme di molecole delle quali bisogna predirre la temperatura di ebollizione (task di regressione). Questo task è stato frequentemente utilizzato nel mondo dell'analisi QSPR e del machine learning per domini strutturati come benchmark, ad esempio in [42]–[46].

Il secondo dataset considerato è il PTC dataset (utilizzato ad esempio in [21], [28]), una collezione di molecole sulle quali sono definiti 4 task di classificazione binaria: lo scopo è di determinare, per quattro classi di roditori, la carcinogenicità di ogni composto chimico del dataset. Il task è stato originariamente creato per la Predictive Toxicology Challenge, tenutasi nel 2001 (`http://www.predictive-toxicology.org/ptc`).

4.1.1 Alcani

L'Alkanes dataset contiene un insieme di molecole di alcani (idrocarburi saturi) di cui si vuole predirre la temperatura di ebollizione. Il problema rientra nella categoria dell'analisi di relazione quantitativa struttura-proprietà (QSPR), che consiste nell'individuare relazioni che leghino la struttura chimica di una certa classe di molecole ad una loro proprietà; nel caso degli alcani, la proprietà in questione è la temperatura di ebollizione del composto chimico.

Gli alcani sono rappresentati nel dataset come grafi non orientati, in cui ogni vertice rappresenta un gruppo carbone-idrogeno della molecola e ogni arco un legame tra atomi di carbonio; i legami tra gli atomi sono semplici e ogni vertice ha una arietà massima di 4. Il dataset è scaricabile da `http://www.di.unipi.it/~micheli/dataset/index.html`.

Seguendo la metodologia utilizzata in [29], si è scelto di eseguire una 10 fold cross validation sull'intero dataset, utilizzando gli stessi iperparametri; a differenza degli esperimenti effettuati in [29] però, si considera il vicinato dei vertici come aperto (ovvero, la codifica del vertice v stesso non viene considerata, eq. 2.16). L'allenamento della rete termina se l'errore massimo sul dataset è minore della soglia stabilita (8° o 5° nei test effettuati), o se si raggiunge il massimo numero di unità, ovvero 150.

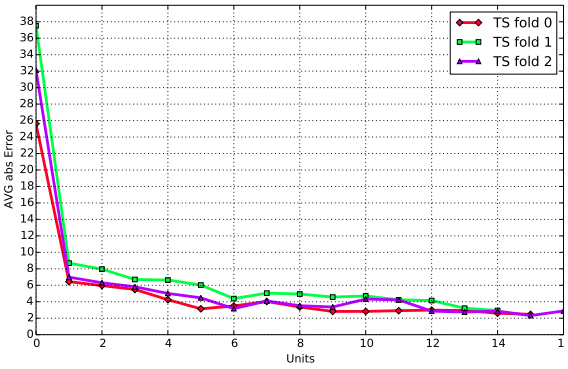
Model	*TR max error*	*TR avg error*	*TS max error*	*TS avg error*	*#Units*
SVD	8	1.96±0.2	8.82±2.7	2.77±0.3	15.6
	5	1.27±0.1	5.75±1.0	1.96±0.4	25.6
Ridge Regression	8	2.14±0.2	6.22±1.4	2.30±0.3	63.9
	5	1.38±0.0	4.58±1.6	1.70±0.3	134.7
Quick Propagation	8	2.29±0.2	7.85±2.3	2.70±0.2	123.6

Tabella 4.1: Risultati implementazione NN4G sul dataset Alcani.

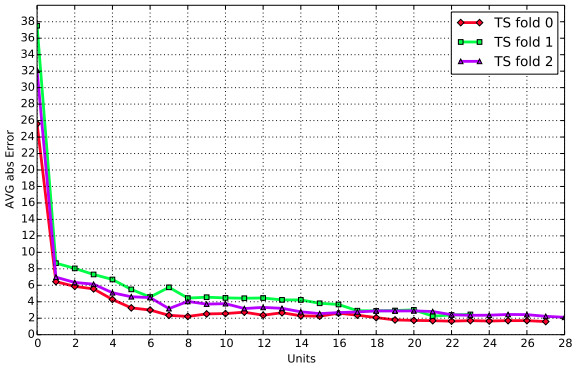
In tabella 4.1 sono riportati i risultati della cross validation. Con *TR avg/max error* si indica la media sui 10 fold dell'errore medio assoluto e dell'errore massimo assoluto sui fold di training, mentre con *TS avg/max error* la media di questi errori sui fold di test. Si è scelto di testare tre diversi algoritmi di apprendimento implementati per le unità di output della rete (vedi appendice A): la Quick Propagation, la pseudo inversa e la Ridge Regression. Per quest'ultime due, seguendo quanto fatto in [29] per la pseudo inversa, si è provato a raggiungere l'errore di training massimo 5°. I risultati ottenuti sono analoghi a quelli in [29] per gli algoritmi di Quick Propagation e SVD (errore massimo e medio assoluto in test riportati in [29] sono pari rispettivamente a 8.53° e 2.35° con l'algoritmo di Quick Propagation, 6.86° e 2.34° con SVD e soglia di allenamento di 8° e 5.03° e 1.74° per SVD con soglia di 5°) , e leggermente migliori con l'utilizzo della Ridge Regression, che permette di regolarizzare il calcolo della pseudo inversa e di ottenere un errore di test massimo pari a 4.58° e medio pari a 1.70°. In figura 4.1 si riportano degli esempi di curve d'apprendimento della rete neurale su questo dataset.

4.1.2 PTC

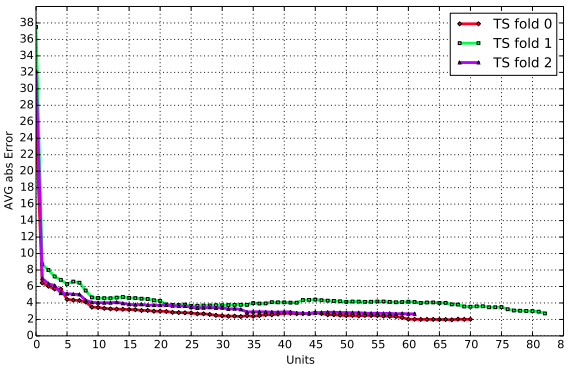
Il dataset PTC [47] contiene un totale di 408 composti chimici, dei quali è indicata la carcinogenicità per quattro tipi di roditori: ratti maschi (MR), ratti femmine (FR), topi maschi (MM) e topi femmine (FM). Ogni classe di roditori rappresenta un diverso task di classificazione; non tutti i dataset presentano lo stesso numero di composti (per alcune combinazioni di task e molecola il target non è definito): MR ne contiene 344, FR 351, MM 336 e FM 349. Il massimo numero di atomi in una molecola è 109 e il massimo numero di legami di ogni atomo è 4.



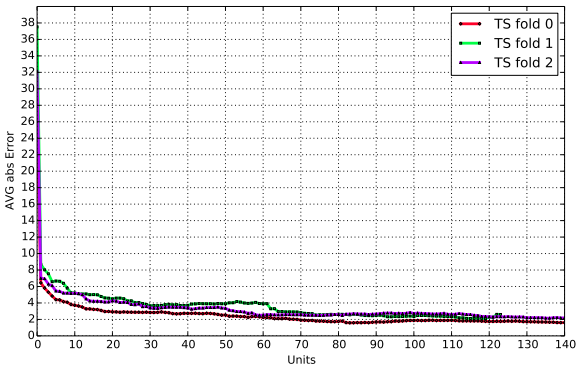
(a) SVD, errore massimo = 8°.



(b) SVD, errore massimo = 5°.



(c) Ridge Regression, errore massimo = 8°.



(d) Ridge Regression, errore massimo = 5°.

Figura 4.1: Curve d'apprendimento dell'errore medio in valore assoluto sul dataset Alcani per il test set di 3 dei fold usati per la cross validation.

Ogni composto viene modellato come un grafo non diretto, in cui ogni vertice corrisponde ad un atomo della molecola e ogni arco ad un legame tra atomi. Ogni vertice è etichettato con una label di 24 elementi: 22 contengono la codifica binaria 1-of-k dell'elemento corrispondente (nel dataset sono presenti 22 atomi diversi), mentre le restanti codificano altre due proprietà della molecola, la carica elettrica (CHG) e la proprietà radical (RAD, proprietà binaria). Ogni molecola è etichettata con un target binario $(+1/-1)$ per ognuno dei 4 dataset in cui compare.

Su questo dataset è stata effettuata una doppia cross validation: ogni dataset (FM,FR,MR,MM) è stato diviso in maniera casuale in 10 fold, dopodichè per ogni fold è stata fatta model selection tramite una 5-fold cross validation sulla parte dei dati in training; la migliore combinazione di parametri è stata poi usata per valutare le performance sul fold di test corrispondente. In tabella 4.2 sono riportati i parametri su cui è stata effettuata model selection. Il modello considerato per i test adopera un distinto set di pesi w per pesare il contributo della codifica del vertice stesso (equazione 3.4).

Soglia sull'errore di training	0.9,0.8,0.75,0.65,0.4
Parametri di regolarizzazione	0.0001,0.001,0.01,0.1
Numero di epoche di training hidden unit e incremento	10,100,500

Tabella 4.2: Parametri model selection per il dataset PTC.

Per confronto, si riportano i risultati di alcuni modelli presenti in letteratura su questi dataset. La Graph Echo State Network (GESN, sez. 2.5) ottiene un accuracy di 57±4% (MR), 67±5% (MM), 65±3% (FR) e 58±4% (FM) [28]. Tramite l'applicazione di tre diversi kernel, il Marginalized Kernel [25], l'Optimal Assignment Kernel [21] e l'Expected Match Kernel [21], sono state invece raggiunte delle accuracy pari a 63±1%, 63±2% e 61±2% (MR), 69±1%, 68±2% e 67±1% (MM), 70±1%, 70±1% e 69±1% (FR) e 65±1%, 65±1% e 65±1% (FM) rispettivamente per i tre kernel. I risultati di GESN riportati sono stati ottenuti con una differente modalità di selezione del modello (holdout set di validazione) rispetto a quella utilizzata qui (cross validation su una divisione in 5 fold del training set), e rappresentano la media di 5 fold di test, mentre i nostri risultati sono la media su 10 fold di test [28]. Riguardo i risultati relativi ai kernel invece, come riportato in [21], sono stati ottenuti con

una *double cross validation*, selezionando la miglior combinazione di parametri per la SVM tramite una 5-fold cross validation, e poi valutando le performance in test tramite una 5-fold cross validation esterna; i risultati riportati corrispondono alle migliori performance in test al variare dei parametri che definiscono i kernel.

In tabella 4.3 è riportata, per ogni dataset, la media delle performance di Neural Network for Graph sui 10 fold di test: con *TR avg accuracy* si indica la media dell'accuracy calcolata sui training set di ogni fold, e analogamente con *TS avg accuracy* la media dell'accuracy sui rispettivi test set; le performance sono comparabili con quelle del modello neurale GESN riportate sopra. I risultati ottenuti non arrivano a superare quelli dei kernel, ma sono comunque ragionevoli, considerando la maggiore efficienza dell'approccio neurale (sez. 2.6).

Dataset	*TR avg Accuracy*	*TS avg Accuracy*	*#Units*
MR	75.48±8%	62.20±6%	11.5
MM	77.71±9%	61.15±6%	18.7
FR	74.21±9%	60.54±8%	12.2
FM	77.58±8%	60.57±8%	14.5

Tabella 4.3: Risultati test su PTC. I valori sono calcolati come media dell'accuracy sui 10 fold di test.

4.2 In Graph Learning

In questa sezione vengono riportati e confrontati con modelli in letteratura i risultati ottenuti dai modelli descritti nel capitolo 3, Neural Network for In-Graph (NN4inG, sez. 3.1), NN4inG con target (sez. 3.2), NN4inG con target e stima (sez. 3.3), NN4inG con target e stima distinti (sez. 3.4), NN4inG con solo hidden unit (primo modello sez. 3.5) e NN4inG con solo output unit (secondo modello sez. 3.5) su tre dataset su cui sono definiti task di in-graph learning.

I primi due dataset, IMDB (sez. 4.2.1) e Cora (sez. 4.2.2), sono stati utilizzati anche in [5] e in altri lavori nel campo dello Statistical Relational Learning (rispettivamente in [1], [7], [10], [11] e [1]–[3]). Entrambi sono stati resi disponibili dagli autori di [5] nel formato da loro utilizzato, ovvero come grafo di vertici non eti-

chettati: per questi due grafi non si hanno quindi a disposizione gli attributi locali dei vertici, ma l'unica informazione disponibile per la classificazione è data dalla topologia del grafo.

In [5] infatti, lo scopo è di analizzare l'efficacia dei soli link relazionali ai fini della classificazione: tramite la sperimentazione su questi due dataset è quindi possibile valutare la capacità dei modelli da noi proposti di affrontare problemi basati solamente sulla struttura delle relazioni tra le entità del grafo, e stimare quanto dell'informazione relazionale i nuovi modelli riescano a utilizzare a confronto con i diversi modelli di SRL proposti in [5].

Il terzo dataset (sez. 4.2.3) è invece parte dei task della *Webspam Challenge*, una competizione indetta nel 2007. Il dataset è composto da una serie di host di pagine web, connessi tra loro in base ai link esistenti tra le pagine di ogni host a formare un grafo; il problema consiste nel classificare i vertici di questo grafo in due classi, "Spam" e "Not Spam". Per questo dataset sono disponibili, oltre alle informazioni sulla topologia del grafo, anche informazioni sul contenuto delle pagine associate a ciascun vertice: tramite la sperimentazione su questo dataset è quindi possibile testare la NN4inG su una rete in cui ogni entità possiede sia attributi locali che relazionali. Inoltre il task permette di confrontare i modelli neurali proposti con quelli dei partecipanti alla competition, che rappresentano una serie di approcci all'apprendimento relazionale differenti da quelli dell'SRL presenti in Netkit.

4.2.1 IMDB

Descrizione Dataset

Il dataset IMDB è tratto dall'Internet Movie Database `http://www.imdb.com`: contiene 1169 film, usciti tra il 1996 e il 2003 negli Stati Uniti. Ad ogni film/vertice è associato un target binario, che indica se l'incasso del film durante il primo weekend nelle sale ha superato i 2 milioni di dollari (ovvero, se il film ha avuto successo o meno).

Il dataset è stato precedentemente usato in altri lavori dello Statistical Relational Learning, ad esempio in [7], [10], dove però anche altri attributi e proprietà dei film nel dataset vengono presi in considerazione. I dati utilizzati per i nostri esperimenti sono invece presi da `http://netkit-srl.sourceforge.net/data/imdb.zip` e cor-

rispondono a quelli usati in [5]: i vertici non hanno nessun attributo locale ($\mathbf{I}(v)$ ha dimensione 0), e l'unica informazione disponibile per la classificazione di un vertice è quella relazionale, data dai vertici nel suo vicinato.

Il grafo è indiretto, e due film sono collegati da un arco se hanno almeno una compagnia di produzione comune; gli archi sono pesati, e il peso è uguale al numero di compagnie di produzione in comune tra i due film. Sono presenti in totale 40634 archi nel grafo. Il task è bilanciato: metà dei film, 572, appartiene alla classe dei film di successo. In figura 4.2 si riporta una visualizzazione della rete corrispondente a questo dataset[1].

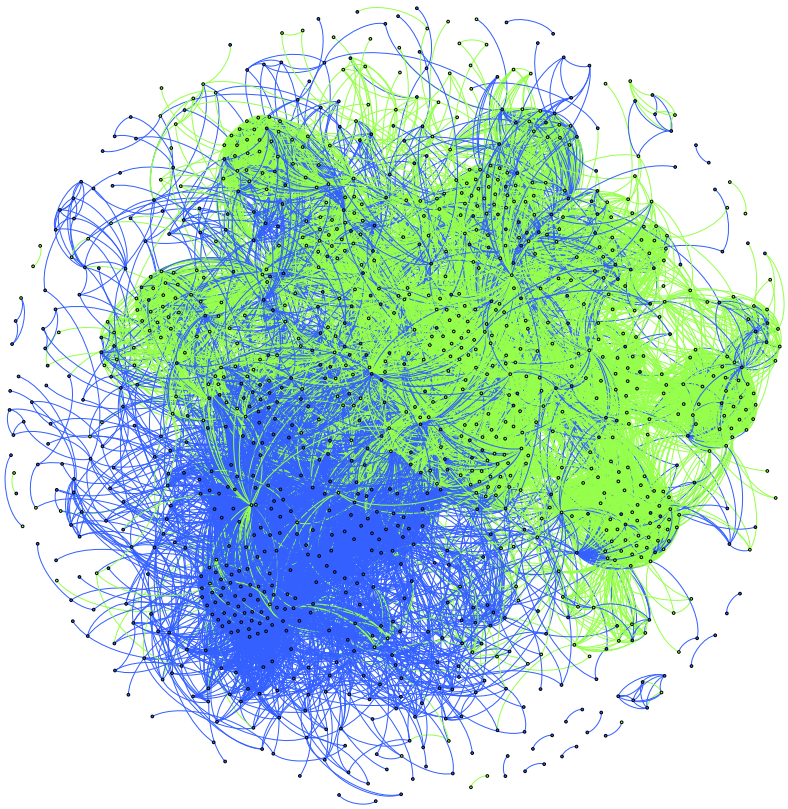


Figura 4.2: Il grafo del dataset IMDB. In blu i vertici appartenenti alla classe dei film non di successo, in verde gli altri.

[1]Questa figura e quelle nelle successive sezioni (fig. 4.4 e 4.6) sono state ottenute con il software di visualizzazione Gephi (`http://gephi.github.io/`, [48]), che fornisce diversi algoritmi di layout per grafi, basati sugli archi esistenti tra i vertici della rete e i loro pesi.

Metodologia Sperimentale

Seguendo quanto fatto in [5], si è scelto di sperimentare l'andamento delle performance dei modelli neurali proposti al variare della percentuale dei vertici presenti nel training set $V_k^{(p)}$, ovvero del numero di vertici per i quali è noto il target. Per fare ciò, i vertici del grafo vengono suddivisi in due sottoinsiemi disgiunti $V_k^{(p)}$ e $V_u^{(p)}$ tali che $V_k^{(p)}$ contiene il $p\%$ dei vertici di V; per gli esperimenti fatti, $p \in \{10, 30, 60, 90\}$.

Essendo il task un problema di classificazione, le performance dei modelli vengono valutate in base alla *accuracy*, ovvero il numero di entità correttamente classificate: $accuracy = \frac{N_c}{N}$, con N_c = numero di elementi correttamente classificati e N = totale degli elementi da classificare.

La *model selection* viene effettuata sui vertici nel training set $V_k^{(p)}$, al fine di individuare la migliore combinazione di iperparametri dei modelli neurali: come strategia di selezione del modello si è scelto di effettuare una 5-fold cross validation sui vertici in $V_k^{(p)}$, per poi selezionare la combinazione di parametri che in media ottiene la miglior accuracy sui fold di test. I parametri testati sono indicati in tabella 4.4.

Modello	NN4inG base
	NN4inG+target
	NN4inG+target e stima
	NN4inG+target e stima distinti
	NN4inG solo hidden unit
	NN4inG solo output unit
Soglia sull'errore di training	0.8,0.7,0.6,0.5,0.4
Parametri di regolarizzazione	0.0001,0.001,0.01,0.1

Tabella 4.4: Parametri su cui si è effettuata model selection per il dataset IMDB.

Durante il processo di model selection i vertici nel test set $V_u^{(p)}$ non vengono utilizzati nè per l'apprendimento, nè per calcolare la performance del modello, che viene valutata solo sulla parte di vertici appartenenti al fold di validazione; come già accennato nei precedenti capitoli però, in un problema di in-graph learning non è consigliabile trascurare i link che collegano tra loro i vertici nel training set e i

vertici nel test set, perchè questo potrebbe portare all'eliminazione di una buona parte dell'informazione relazionale contenuta nella rete. Per questo motivo, seppur non etichettati e non utilizzabili per l'apprendimento, i vertici in $V_u^{(p)}$ restano nella rete sia durante la validazione che durante il test, andando così a contribuire al contesto (con la loro codifica o con la stima effettuata dal modello stesso, vedi cap. 3, ma non con il loro target) dei vertici a loro collegati.

Risultati

In questa sezione si riportano i risultati dei modelli neurali descritti nel capitolo 3 sul dataset IMDB, confrontandoli con quelli ottenuti utilizzando gli algoritmi relazionali e di inferenza collettiva presenti in Netkit [5], [49].

In tabella 4.5 si riportano i risultati di Netkit sui test set $V_u^{(p)}$ al variare della percentuale dei vertici etichettati p: vengono considerati due algoritmi di classificazione relazionale, il Class-Distribution Relational Neighbor Classifier (CDRN) e il Weighted-Vote Relational Neighbor Classifier (WVRN) (sez. 2.8.2). Come algoritmo di collective inference è stato utilizzato l'algoritmo Relaxation Labeling (RL, sez. 2.8.3), indicato anche in [5] come il migliore su questi dataset.

	p=10%	*p=30%*	*p=60%*	*p=90%*
CDRN+RL	60.11%	80.46%	78.84%	80.50%
CDRN	58.11%	79.85%	78.41%	80.50%
WVRN+RL	75.97%	80.34%	77.77%	81.35%
WVRN	75.21%	80.46%	77.99%	80.50%

Tabella 4.5: Risultati di Netkit sul dataset IMDB, per le diverse percentuali di vertici nel test set considerate.

In tabella 4.6 sono elencate le performance di validazione della migliore combinazione di parametri per ognuno dei sei modelli considerati: le performance sono calcolate come media dell'accuracy sui cinque fold di validazione. Con NN4inG+t, NN4inG+t+s, NN4inG+t+s distinti si indicano rispettivamente il modello con target, con target e stima e con target e stima distinti, mentre con NN4inG hidden/output only i due modelli semplificati.

	p=10%	*p=30%*	*p=60%*	*p=90%*
NN4inG base	65.43%	67.91%	68.84%	73.34%
NN4inG+t	80.98%	78.80%	82.01%	82.38%
NN4inG+t+s	78.30%	79.07%	81.73%	81.90%
NN4inG+t+s distinti	81.78%	79.07%	82.44%	81.52%
NN4inG hidden only	77.47%	76.49%	82.14%	80.66%
NN4inG output only	76.49%	78.49%	82.29%	81.52%

Tabella 4.6: Risultati model selection sul dataset IMDB, per le diverse percentuali di vertici nel test set considerate.

In tabella 4.7 infine si riportano i risultati della rete sul test set $V_u^{(p)}$ al variare della percentuale dei vertici nel training set p; i risultati in tabella sono una media dell'accuracy ottenuta in 10 esecuzioni del processo di allenamento e costruzione di ogni rete neurale; ogni esecuzione si differenzia dalle altre per l'inizializzazione casuale dei pesi di ogni neurone.

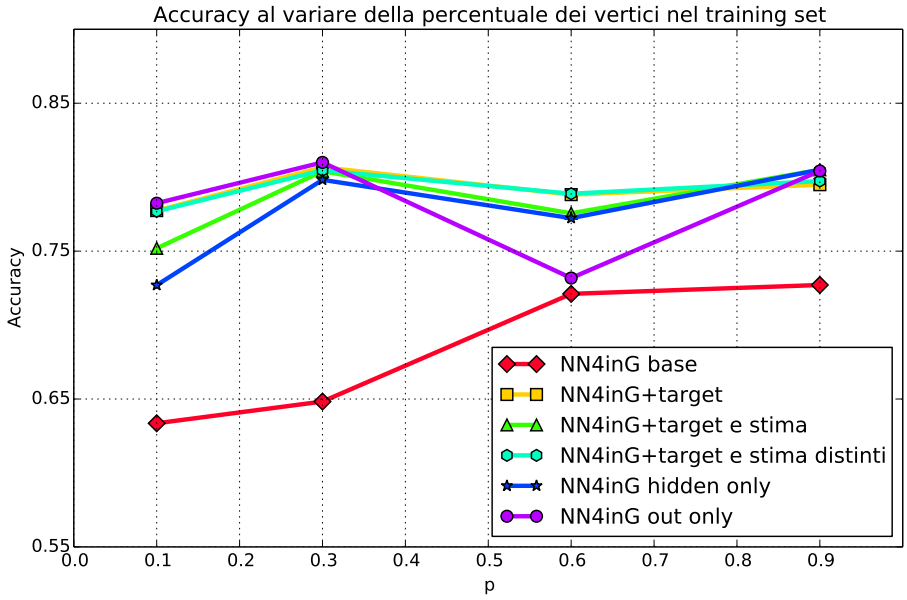
	p=10%	*p=30%*	*p=60%*	*p=90%*
NN4inG base	63.36±2.57%	64.83±2.94%	72.11±5.44%	72.71±5.16%
NN4inG+t	77.74±1.66%	80.67±0.56%	78.82±1.03%	79.49±0.83%
NN4inG+t+s	75.19±4.90%	80.40±0.35%	77.56±0.00%	**80.50±0.00%**
NN4inG+t+s distinti	77.69±1.33%	80.46±0.54%	**78.88±1.2%**	79.74±1.10%
NN4inG hidden only	72.70±3.76%	79.82±1.88%	77.23±1.68%	**80.50±0.00%**
NN4inG output only	**78.24±0.02%**	**81.00±0.39%**	73.18±0.10%	80.42±0.25%

Tabella 4.7: Risultati test sul dataset IMDB, per le diverse percentuali di vertici nel test set considerate. In grassetto sono evidenziati i migliori risultati (tra quelli di tutti i modelli) ottenuti per ciascuna percentuale p.

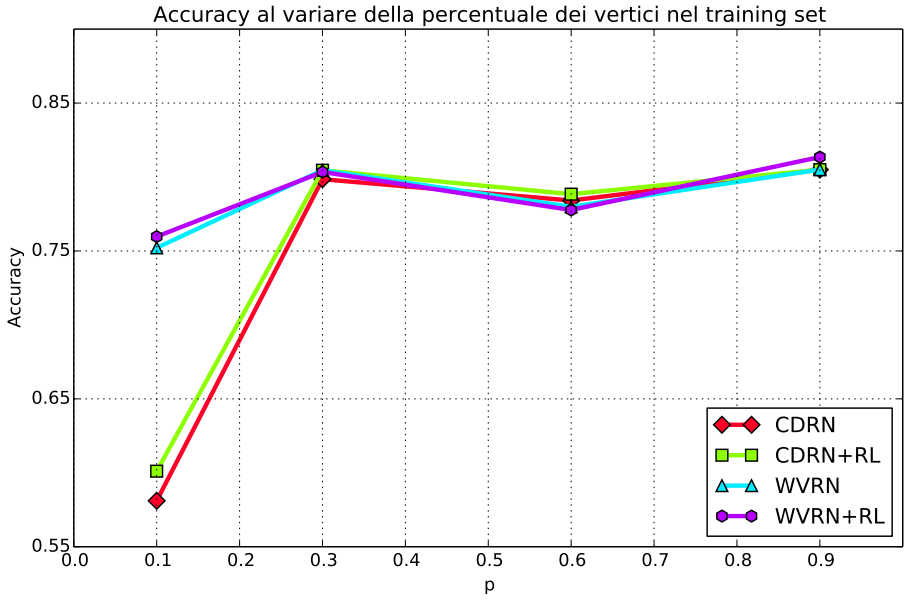
Tra i risultati di Netkit in tabella 4.5 si riporta sia l'accuracy ottenuta dalle due combinazioni di classificatori relazionali con l'algoritmo Relaxation Labeling (CDRN+RL e WVRN+RL in tabella), sia quella ottenuta dai soli classificatori relazionali (CDRN e WVRN in tabella), senza l'utilizzo di nessun algoritmo di inferenza collettiva. Utilizzare solamente la componente di classificazione relazionale equivale a eliminare dal modello di SRL la capacità di modificare iterativamente la

stima per i vertici non etichettati, forzandolo ad utilizzare per questi la stima effettuata dal classificatore locale; per questo dataset, non essendoci label di input locali per i vertici, tale stima viene calcolata come la percentuale di vertici noti in ognuna delle due classi (che per IMDB è circa pari al 51% e 49% rispettivamente). Inoltre, come detto nel capitolo 3, senza inferenza collettiva il solo classificatore relazionale è fortemente limitato dall'assunzione di Markov, non avendo nessuna nozione sul contesto di un vertice ad eccezione del suo vicinato di raggio 1, $\mathcal{N}(v)$. Nonostante questa limitazione, dall'analisi dei risultati in tabella 4.5 si può notare come con i soli classificatori relazionali si riescano ad ottenere performance totalmente comparabili a quelle dei più sofisticati algoritmi con inferenza collettiva (vedi grafico in figura 4.3b): questo sembra suggerire che per questo dataset l'informazione ricavata dai vicini in $V_k^{(p)}$ di un vertice sia più che sufficiente ad ottenere performance di classificazione adeguate, rendendo inutile il ricorrere a tecniche più sofisticate di inferenza iterativa.

Quanto osservato porta alla conclusione che per la rete definita dal dataset IMDB l'ipotesi di omofilia sia valida, e che quindi per questo grafo è effettivamente vero che vertici appartenenti alla stessa classe sono collegati con maggiore probabilità di due vertici in classi diverse; inoltre, risulta plausibile anche l'assunzione di Markov, e per queste ragioni la classe di un vertice è facilmente determinabile guardando solamente le classi dei vertici nel suo vicinato immediato: quanto detto è dimostrato anche dal fatto che il classificatore relazionale che ottiene le migliori performance è WVRN, che è totalmente basato sul presupposto dell'esistenza dell'omofilia. Quando la percentuale dei vertici in $V_k^{(p)}$ è alta, ovvero $p = 90\%$ e $p = 60\%$, la maggior parte dei vicini di ogni vertice risulterà in questo insieme, e di conseguenza il classificatore relazionale per quanto detto avrà a disposizione tutta l'informazione necessaria per la classificazione: la differenza infatti tra l'accuracy ottenuta in questi due casi con e senza inferenza collettiva è totalmente trascurabile. Sorprendentemente, anche quando solo il 30% e il 10% dei vertici del grafo fanno parte del training set, situazione in cui si sarebbe portati a pensare che l'introduzione di un meccanismo di stima più accurato possa apportare un vantaggio, essendo la maggior parte dei vertici non etichettati, i soli classificatori relazionali ottengono di nuovo risultati comparabili ai rispettivi modelli con inferenza collettiva, dai quali si differenziano solo per l'$1 - 2\%$.



(a) NN4inG.



(b) Netkit.

Figura 4.3: Accuracy di NN4inG e Netkit su IMDB al variare della percentuale dei vertici nel training set.

Le osservazioni appena fatte trovano riscontro anche nei risultati ottenuti dai nostri modelli neurali (tabella 4.7): il modello base di NN4inG è quello che ottiene le performance peggiori, come si può vedere anche in figura 4.3a. Come ipotizzato nel precedente capitolo, un modello neurale senza nessuna conoscenza diretta della classe dei vicini di un vertice risulta svantaggiato in problemi in cui il target da stimare dipende fortemente proprio da questa conoscenza, nonostante abbia a disposizione la più flessibile informazione contestuale data dalla codifica adattiva. I modelli neurali che invece utilizzano esplicitamente la classe dei vicini ottengono performance che sono confrontabili con quelle di Netkit, e in alcuni casi migliori: per $p = 10\%$ il miglior risultato ottenuto è di 78.24% (NN4inG con solo output unit) contro il 75.97% di Netkit, per $p = 30\%$ NN4inG con solo output unit raggiunge l'81.00% e Netkit l'80.46%, per $p = 60\%$ NN4inG con target e stima distinti ha il 78.88% di accuracy mentre Netkit ottiene il 78.84%, e infine per $p = 90\%$ NN4inG ottiene un accuracy dell'80.50% e Netkit dell'81.35%. Per $p = 10\%$ e $p = 30\%$ il modello che utilizza sia il target che la stima (NN4inG+ target e stima) ottiene un accuracy leggermente inferiore rispetto agli altri, probabilmente perchè, come discusso nella sezione 3.3, considerare equivalentemente target reale e stima della rete degrada le performance del modello; in generale però quasi tutti i modelli proposti sono comparabili con i risultati dei classificatori relazionali.

In particolare, si è osservato che, nella grande maggioranza dei casi, la combinazione di iperparametri selezionata dal processo di model selection risulta in una rete con solo 1 o 2 unità nascoste: con così poche unità, il contesto considerato dai modelli neurali è ristretto al solo vicinato immediato di un vertice $\mathcal{N}(v)$, il che è coerente con quanto riscontrato dal confronto tra i risultati dei modelli relazionali con e senza collective inference.

A ulteriore conferma di quanto osservato dai risultati di Netkit, i modelli NN4inG con solo hidden unit e NN4inG con solo output unit (sez. 3.5), che sono una semplificazione dell'architettura di NN4inG in cui la codifica viene eliminata e si utilizza solo la stima, ottengono performance pari e in alcuni casi migliori dei modelli più complessi, a testimonianza del fatto che in questo particolare dataset l'utilizzo della più complessa informazione contestuale non è strettamente necessario.

Dai risultati ottenuti su questo dataset appare evidente l'effettiva utilità dell'aver sviluppato nuovi modelli della Neural Network for In-Graph in grado di gestire

in maniera esplicita il target dei vicini, per riuscire ad affrontare questi particolari task in cui la funzione da apprendere dipende fortemente da questa informazione. In particolare, i risultati rispondono alla domanda posta nella sezione 3.5: l'informazione contestuale dello stato è ancora necessaria dopo aver introdotto esplicitamente l'informazione sul target e la stima dei vicini nella rete neurale?

	p=10%	*p=30%*	*p=60%*	*p=90%*
NN4inG senza stato	75.87%	80.31%	77.56%	80.50%

Tabella 4.8: Risultati test NN4inG senza stato.

Per quanto visto finora, per problemi come IMDB fortemente basati sull'omofilia, l'informazione contestuale data dalla codifica non risulta indispensabile. In tabella 4.8 sono riportate le performance su IMDB del modello NN4inG+target e stima distinti (sez. 3.4) in cui l'informazione di stato è stata eliminata dall'equazione di $x_i(\cdot)$: per calcolare la codifica di un vertice, il neurone i-esimo ha accesso solamente al target dei vicini in $V_k^{(p)}$ e alla stima fatta dalla rete al passo precedente, ma non alla codifica calcolata dalle precedenti unità. I risultati ottenuti con questo modello sono pari, o quasi, a quelli ottenuti con tutti gli altri modelli che utilizzano anche l'informazione contestuale, e giustificano quindi l'introduzione dei due modelli NN4inG con solo hidden unit e NN4inG con solo output unit, che semplificano l'architettura neurale e possono risultare più che efficaci su un problema con queste caratteristiche.

4.2.2 Cora

Descrizione Dataset

Il Cora dataset (figura 4.4) è una collezione di articoli scientifici nel campo dell'informatica. In particolare, il dataset utilizzato in [5] (`http://netkit-srl.sourceforge.net/data/cora.zip`) comprende 3583 paper riguardanti il Machine Learning, etichettati in base alla specifica area di ricerca trattata nell'articolo; in totale sono presenti $M = 7$ classi: "Case-based" (402 articoli), "Genetic Algorithms" (551), "Neural Networks" (1064), "Probabilistic Methods" (529), "Reinforcement Learning" (335), "Rule Learning" (230), "Theory" (472).

Gli articoli sono collegati in un grafo indiretto in base alle citazioni: se un articolo ne cita un altro, i due sono collegati da un arco di peso 1; il peso di un arco tra due articoli può essere 2, se gli articoli si citano a vicenda. Sono presenti 22516 archi.

Come per IMDB, anche in questo dataset (nella versione da noi utilizzata, presente in [5]) i vertici non hanno attributi locali ($\mathbf{I}(v)$ è vuoto), e l'unica fonte di informazione utilizzabile per la classificazione di un vertice sono i vertici con cui lui è in relazione (cioè, in Cora, gli articoli da lui citati o che lo citano).

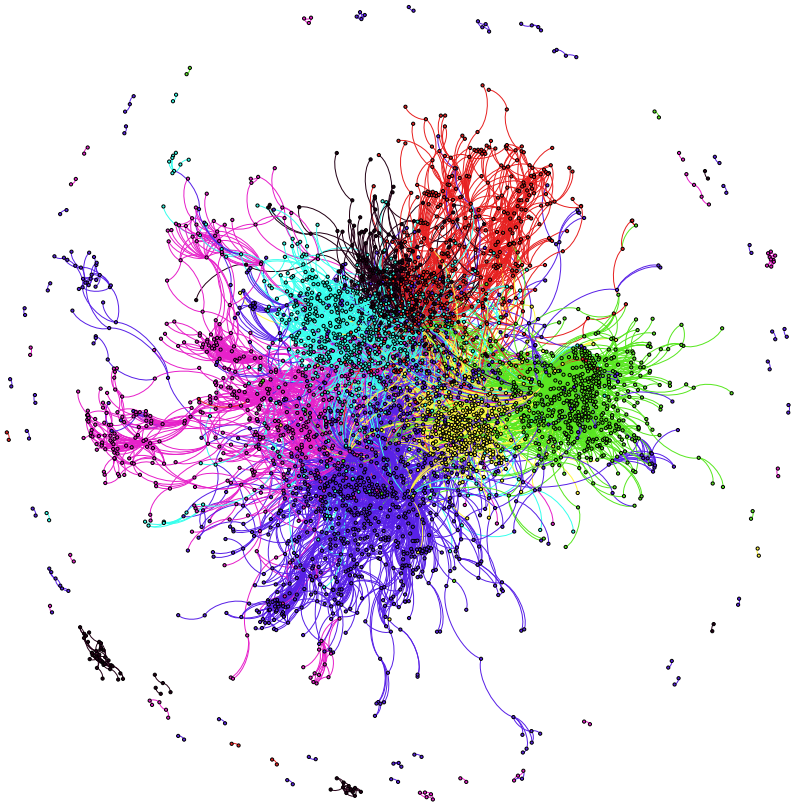


Figura 4.4: Il grafo del dataset Cora. I vertici sono colorati in base alla classe di appartenenza e gli archi prendono il colore del vertice target.

Metodologia Sperimentale

Per questo dataset è stata utilizzata la stessa metodologia descritta nella precedente sezione per il dataset IMDB (4.2.1).

Le performance dei modelli proposti vengono valutate al variare della percentuale dei vertici del grafo appartenenti al training set $V_k^{(p)}$, per $p \in \{10, 30, 60, 90\}$. Sui vertici in $V_k^{(p)}$ viene effettuata model selection tramite una 5-cross validation per selezionare la migliore combinazione di parametri, che viene poi usata per riallenare il modello sull'intero training set $V_k^{(p)}$ e valutarne le performance sul test set $V_u^{(p)}$.

Come prima, il task è un problema di classificazione, e quindi le performance dei modelli vengono valutate in base al numero di vertici correttamente classificati (*accuracy*).

Per Cora, gli iperparametri su cui si è effettuata model selection sono elencati in tabella 4.9.

Modello	NN4inG base
	NN4inG+target
	NN4inG+target e stima
	NN4inG+target e stima distinti
	NN4inG solo hidden unit
	NN4inG solo output unit
Soglia sull'errore di training	0.5,0.4,0.3,0.2
Parametri di regolarizzazione	0.001,0.01

Tabella 4.9: Parametri su cui si è effettuata model selection per il dataset Cora.

Risultati

In questa sezione si riportano i risultati ottenuti dalle diverse varianti di NN4inG e da Netkit [49] per il dataset Cora.

In tabella 4.10 sono elencati i risultati di Netkit sui test set $V_u^{(p)}$ al variare della percentuale di vertici etichettati p.

	p=10%	*p=30%*	*p=60%*	*p=90%*
CDRN+RL	57.04%	80.48%	83.20%	85.35%
CDRN	46.03%	72.38%	81.32%	85.08%
WVRN+RL	80.91%	84.68%	84.18%	84.25%
WVRN	44.70%	73.39%	80.97%	84.53%

Tabella 4.10: Risultati di Netkit sul dataset Cora, per le diverse percentuali di vertici nel test set considerate.

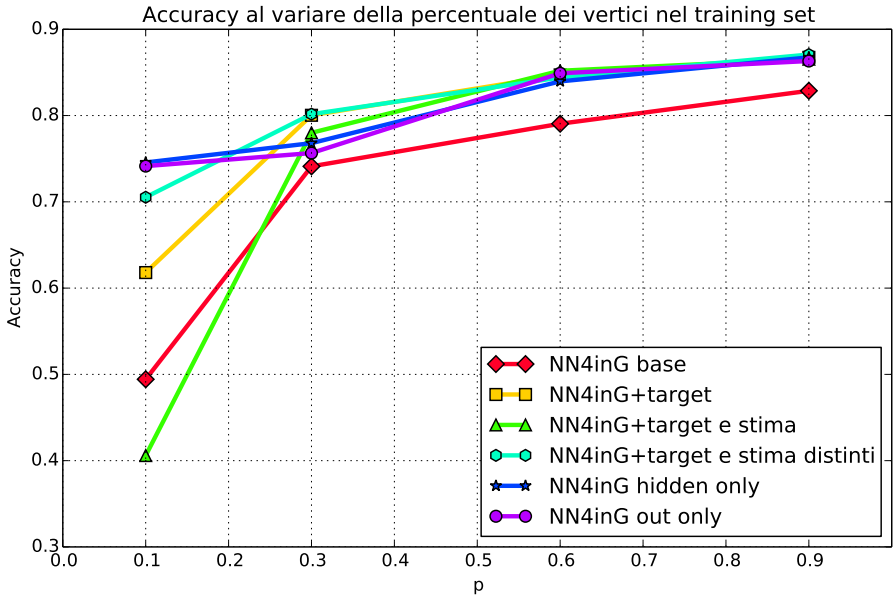
Come per IMDB, anche per questo dataset si sono considerati due algoritmi di classificazione relazionale, CDRN e WVRN (2.8.2), e l'algoritmo di collective inference Relaxation Labeling (RL).

In tabella 4.11 sono presenti le performance di validazione della migliore combinazione di parametri per ognuno dei sei modelli neurali considerati, calcolate come media delle performance sui cinque fold di validazione. Come per IMDB, si indicano con NN4inG+t, NN4inG+t+s, NN4inG+t+s distinti rispettivamente il modello con target, con target e stima e con target e stima distinti, mentre con NN4inG hidden/output only i due modelli semplificati.

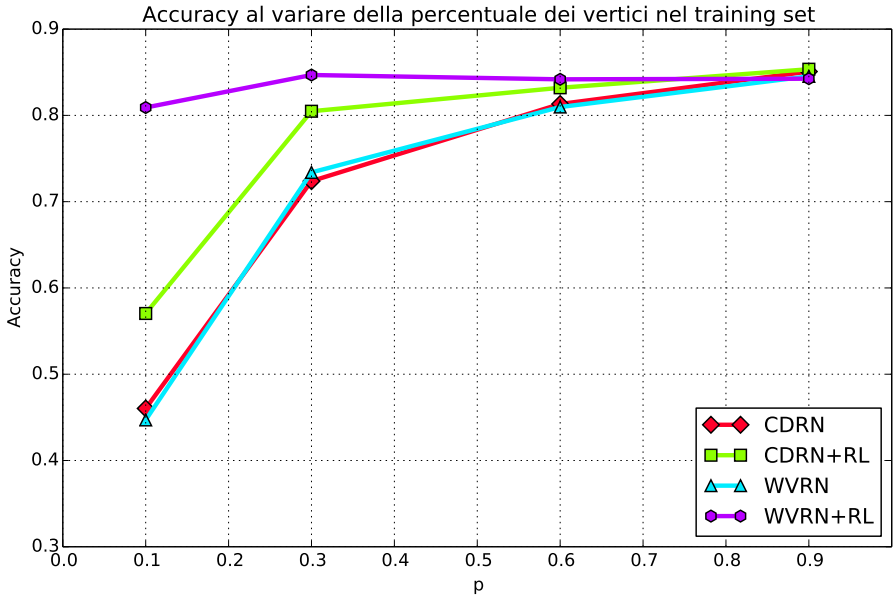
	p=10%	*p=30%*	*p=60%*	*p=90%*
NN4inG base	49.24%	71.27%	79.79%	81.86%
NN4inG+t	62.71%	79.94%	85.65%	86.52%
NN4inG+t+s	50.76%	76.47%	86.49%	87.39%
NN4inG+t+s distinti	61.92%	81.61%	85.19%	86.39%
NN4inG hidden only	78.71%	76.97%	82.81%	86.05%
NN4inG output only	79.79%	74.42%	86.40%	86.36%

Tabella 4.11: Risultati model selection sul dataset Cora, per le diverse percentuali di vertici nel test set considerate.

In tabella 4.12 sono infine riportati i risultati sui quattro test set di Neural Network for In-Graph al variare della percentuale dei vertici etichettati p. I risultati sono la media di 10 esecuzioni dell'algoritmo di apprendimento, al variare dell'inizializzazione (casuale) dei pesi della rete neurale.



(a) NN4inG.



(b) Netkit.

Figura 4.5: Accuracy di NN4inG e Netkit su Cora al variare della percentuale dei vertici nel training set.

	p=10%	*p=30%*	*p=60%*	*p=90%*
NN4inG base	49.43±2.90%	74.11±0.70%	79.05±0.44%	82.87±1.09%
NN4inG+t	61.80±4.76%	80.03±1.04%	84.71±0.35%	86.74±0.64%
NN4inG+t+s	40.59±7.89%	78.00±1.13%	**85.19±0.32%**	86.54±0.56%
NN4inG+t+s distinti	70.53±0.76%	**80.16±0.83%**	84.41±0.31%	**87.07±0.73%**
NN4inG hidden only	**74.56±3.02%**	76.80±3.49%	83.96±1.34%	86.74±1.52%
NN4inG output only	74.14±2.68%	75.65±2.86%	84.90±0.43%	86.32±1.37%

Tabella 4.12: Risultati test sul dataset Cora, per le diverse percentuali di vertici nel test set considerate. In grassetto sono evidenziati i migliori risultati (tra quelli di tutti i modelli) ottenuti per ciascuna percentuale p.

A differenza di quanto osservato per i risultati di Netkit sul dataset IMDB, su Cora appare netta la distinzione tra i risultati con e senza inferenza iterativa: guardando il grafico in figura 4.5b, si può notare come il distacco tra la performance ottenuta dai soli algoritmi relazionali CDRN e WVRN è notevole, soprattutto quando i vertici etichettati nel grafo sono pochi: la differenza è minima per $p = 90\%$, sale al $2 - 3\%$ per $p = 60\%$, e poi all'$8 - 10\%$ per $p = 30\%$ e al $10 - 40\%$ per $p = 10\%$. Quando la grande maggioranza dei vertici del grafo è nel training set però, come nel caso di $p = 90\%$ o $p = 60\%$, di nuovo i soli classificatori relazionali sono pari ai modelli con inferenza collettiva. Questo indica che anche in questo dataset l'ipotesi di omofilia e l'assunzione di Markov sono in una qualche misura soddisfatte: un classificatore come WVRN infatti, totalmente basato sul concetto di omofilia, ottiene un accuracy pari all'84.53% disponendo solo del target dei vicini per fare inferenza. Inoltre, WVRN+RL ottiene la migliore accuracy per $p = 10\%$, $p = 30\%$ e $p = 60\%$.

Dal confronto tra i risultati di Netkit (tab. 4.10) e quelli dei modelli di NN4inG (tab. 4.12) possiamo notare come per $p = 90\%$ il modello NN4inG con target e stima distinti (sez. 3.4) ottenga un accuracy pari all'87.07%, contro l'85.35% del modello CDRN+RL (il migliore dei modelli in Netkit per $p = 90\%$), e per $p = 60\%$ il miglior modello neurale (NN4ing+target e stima) ha una performance di 85.19%, ancora migliore di quella del miglior modello relazionale, WVRN+RL in questo caso, pari all'84.18%. Per percentuali minori invece, il modello WVRN+RL ottiene una accuracy dell' 84.68% e 80.91% per $p = 30\%$ e $p = 10\%$ rispettivamente, contro

i migliori risultati di NN4inG pari all'80.16% e al 74.56%.

Di nuovo, come per IMDB, il modello NN4inG base (sez. 3.1), avvalendosi della sola codifica adattiva, non è in grado di raggiungere le stesse performance dei modelli che invece utilizzano esplicitamente il target e la stima dei vicini (grafico in figura 4.5a), da cui si distacca per il $4-5-6-12\%$ al diminuire della percentuale dei vertici nel training set: da questo e soprattutto dai risultati di WVRN+RL possiamo dedurre che anche in questo dataset la classe di un vertice dipende fortemente dalle classi associate ai vertici nel suo vicinato, e che è utile ricorrere all'uso di modelli in grado di utilizzare questa informazione.

In particolare, i risultati migliori sono quelli ottenuti dai modelli neurali che impiegano sia il target, sia la stima dei vicini di un vertice: a differenza di quanto visto per IMDB, in Cora l'introduzione della stima per i vertici in $V_u^{(p)}$ porta a un incremento delle performance, in particolare quando i vertici non etichettati sono la maggioranza, come per $p = 10\%$: in questo caso il modello NN4inG con target e stima distinti ottiene il 70.53% di accuracy e i modelli con solo hidden e output unit il 74.56% e il 74.14% rispettivamente, mentre il modello con il solo target raggiunge solamente il 61.80%. Inoltre, coerentemente con quanto osservato per IMDB, i modelli in cui i contributi di target e stima vengono distinti (gli ultimi tre in tabella 4.12) ottengono in generale performance migliori del modello NN4inG+target e stima, il che sembra confermare l'intuizione che sia necessario distinguere tra le due tipologie di vicini di un vertice; il problema è evidente soprattutto per $p = 10\%$, dove il modello NN4inG+target e stima ottiene un accuracy del 40.59%, di circa il 30% inferiore di quella ottenuta dagli altri tre modelli. Da quanto appena detto possiamo osservare che, dai risultati ottenuti su questo dataset, non per tutti i grafi vale quanto detto per IMDB (sez. 4.2.1) riguardo alla effettiva utilità della codifica adattiva: già su Cora infatti, nonostante sia valida anche per questo dataset l'ipotesi di omofilia, è possibile notare come l'utilizzo congiunto di codifica, target e stima permetta di ottenere risultati lievemente migliori rispetto a quelli dei modelli con solo target e stima (NN4inG con solo hidden e output unit).

Infine, contrariamente a quanto osservato per IMDB, su questo dataset le reti neurali risultanti dai modelli scelti tramite model selection hanno in generale un numero alto di unità nascoste: in particolare questo è vero per i modelli NN4inG base, con target, con target e stima e con target e stima distinti, che risultano avere

dalle 20 alle 50 unità nascoste; i modelli più semplici che utilizzano la sola stima, NN4inG con solo hidden e output unit, sono invece in media costituiti da $3 - 6$ unità.

4.2.3 Webspam

Descrizione Dataset

Il dataset Webspam (figura 4.6) fa parte dei dataset messi a disposizione per la Webspam Challenge, una competizione indetta nel 2007 in concomitanza con l'ECML/PKDD Graph Labeling Workshop[2].

Il dataset è stato creato a partire dalla collezione WEBSPAM-UK-2006, redatta dall'università La Sapienza di Roma e dall'università di Milano. Il corpus consiste di 77 milioni di pagine web appartenenti a 12000 host diversi, annotati sia manualmente che automaticamente come "Spam", "Not Spam" e "Borderline" (ad esempio host appartenenti a domini come .gov.uk sono automaticamente annotati come host non di spam).

Nei nostri esperimenti si è scelto di usare il dataset corrispondente alla track 2 della challenge, consistente in un grafo $G(H, E)$ di 9072 vertici, ognuno corrispondente ad un host, e 514700 archi. L'host h_1 e l'host h_2 sono collegati da un arco (ovvero $(h_1, h_2) \in E$) se una pagina web nell'host h_1 contiene un link ad una pagina nell'host h_2: sono quindi possibili due interpretazioni del grafo, sia come grafo indiretto, se si decide di considerare semplicemente la relazione tra due host, espressa indifferentemente dalla presenza sia di link entranti che di link uscenti tra i due, sia come grafo diretto, distinguendo l'insieme dei predecessori e dei successori di ogni vertice.

Il task è di classificazione binaria dei vertici nelle due classi $\{Spam, NotSpam\}$; non si considerano cioè gli host "Borderline" (non sono presenti nel dataset della track 2 considerato).

Per ogni host del grafo era disponibile per i partecipanti alla challenge un vettore di feature, contenente valori $tfidf$ (normalizzati) per quell'host, calcolati sul contenuto di 100 pagine appartenenti all'host. Il $tfidf$ è l'abbreviazione per "term frequency-inverse document frequency" ed è un valore numerico che esprime quanto

[2] `http://webspam.lip6.fr/wiki/pmwiki.php?n=Main.PhaseII`

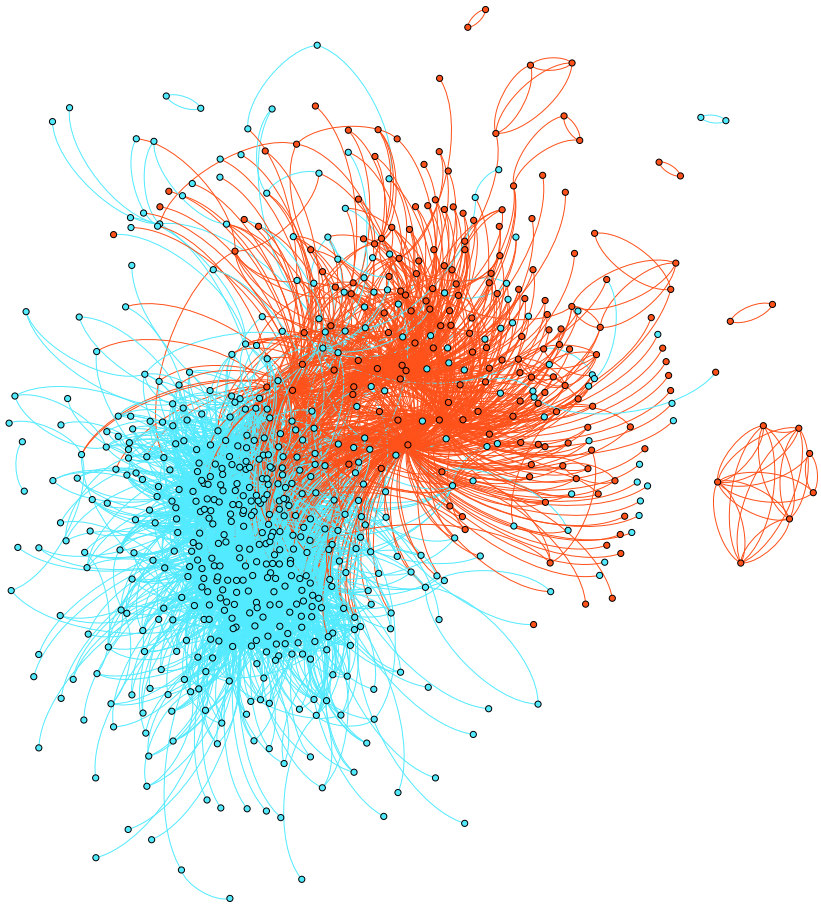


Figura 4.6: Una parte del grafo del dataset Webspam. In questa immagine sono presenti solo i vertici di training non isolati (cioè collegati con altri vertici di training direttamente); in rosso i vertici appartenenti alla classe *Spam* e gli archi il cui target è un vertice appartenente a questa classe.

un dato termine t è importante per caratterizzare un documento d all'interno di una collezione D: l'importanza viene calcolata mettendo in relazione quante volte il termine occorre nel documento d rispetto alle occorrenze totali del termine in tutta la collezione, secondo le formule

$$tf(t,d) = \frac{|\{\, t_i | t_i \in d \wedge t_i = t \,\}|}{|d|} \tag{4.1}$$

$$idf(t,D) = \log \frac{|D|}{|\{\, d_i | d_i \in D \wedge t \in d_i \,\}|} \tag{4.2}$$

$$tfidf(t,d,D) = tf(t,d) \cdot idf(t,D) \tag{4.3}$$

La lunghezza dei vettori $tfidf$ è uguale al numero di parole nel dizionario della collezione, che nel caso di Webspam, è pari a più di quattro milioni e mezzo di termini; i vettori sono vettori sparsi, che contengono cioè molti zeri.

Utilizzare in maniera diretta questi vettori $tfidf$ come input per i neuroni di una rete neurale non è una strada praticabile: il fatto che i vettori siano sparsi non può essere direttamente sfruttato per ottimizzare la rappresentazione della label di input locale, in quanto ogni neurone avrebbe dovuto comunque mantenere e memorizzare un vettore di pesi $\bar{\mathbf{w}}$ lungo quanto l'intero dizionario.

Per questo motivo si è scelto di preprocessare le label locali dei vertici in modo da ridurne la dimensionalità e renderle più adatte all'utilizzo con una rete neurale; per fare ciò è stata utilizzata la libreria GibbsLDA++ [50], che contiene una implementazione in C++ della Latent Dirichlet Allocation. La LDA è un modello probabilistico generativo che permette di individuare topic o concetti all'interno dei documenti di una collezione: al termine dell'esecuzione dell'algoritmo si ha, per ogni documento, un vettore di probabilità con tanti elementi quanti sono i topic nella collezione, che esprime quanto il documento è legato a ciascun topic. Per maggiori dettagli sul modello si rimanda a [51]. Per applicare l'algoritmo al dataset Webspam i vettori $tfidf$ forniti sono stati utilzzati per ricavare il dizionario della collezione e per ricostruire il contenuto delle pagine di ciascun host (ovviamente solo di quelle usate per calcolare i vettori $tfidf$ contenuti nel dataset) ottenendo per ogni host un unico documento. Dopodichè è stato possibile applicare il modello alla collezione di documenti così ottenuta.

Si è scelto per questi esperimenti di considerare 50 topic: al termine dell'esecuzione dell'algoritmo quindi ad ogni host viene associato un vettore numerico di 50 elementi, che costituiranno le nuove label locali per i vertici del grafo durante l'allenamento con NN4inG. Il modello LDA viene cioè usato per estrarre dai termini di ogni documento una sorta di "riassunto" del contenuto tramite "parole chiave", ovvero i topic: si interpreta quindi il vettore output dell'LDA come una nuova e molto più sintetica rappresentazione dei documenti nella collezione.

Tramite questo dataset è possibile sperimentare le potenzialità della Neural Network for In-Graph su un problema differente dai due precedenti. La presenza delle etichette locali permette di valutare come i modelli riescono ad integrare le due fonti di informazione, locale e relazionale; inoltre, è possibile confrontare i risultati ottenuti con quelli dei partecipanti alla challenge, che rappresentano una serie di approcci alla classificazione relazionale differenti tra loro e da quelli dello SRL presenti in Netkit. Soprattutto però, come discusso nelle precedenti sezioni, l'esistenza dell'omofilia in entrambi i dataset IMDB e Cora influenza fortemente i risultati ottenuti, rendendo superflua ai fini della classificazione la maggiore flessibilità di NN4inG nel trattare l'informazione contestuale tramite la codifica adattiva. Webspam al contrario ci permette di evidenziare l'effettiva potenza del meccanismo di codifica contestuale di NN4inG in un problema dove l'ipotesi di omofilia è meno forte.

Il dataset è stato quindi utilizzato per due diversi esperimenti: per confronto con Netkit, come nelle sezioni precedenti, si è voluto considerare il grafo di input come non etichettato, non utilizzando quindi nessuna label locale per i vertici in H. Introducendo poi le label ottenute come spiegato sopra, ci si è confrontati con i risultati della Webspam Challenge.

Metodologia Sperimentale

Per questo dataset è disponibile una suddivisione in training e test set ($H_k \cup H_u = H$, $H_k \cap H_u = \emptyset$) dei vertici del grafo fornita dagli organizzatori della challenge; anche per i nostri esperimenti si è scelto di utilizzare questa suddivisione, per poter ottenere risultati confrontabili con quelli dei partecipanti. Il training set contiene 907 vertici mentre il test set ne contiene 8165; sia nel training che nel test set

la percentuale di vertici con target *Spam* è di circa il 20% del totale dei vertici dell'insieme.

Modello	NN4inG base
	NN4inG+target
	NN4inG+target e stima
	NN4inG+target e stima distinti
	NN4inG solo hidden unit
	NN4inG solo output unit
Grafo	diretto
	indiretto
Soglia sull'errore di training	0.15,0.2,0.25,0.3
Parametri di regolarizzazione	0.01,0.1

Tabella 4.13: Parametri su cui si è effettuata model selection per il dataset Webspam.

Per la model selection, seguendo quanto fatto in [8], l'insieme H_k viene ulteriormente diviso in due sottoinsiemi, $H_k = H_k^{(t)} \cup H_k^{(v)}$, $H_k^{(t)} \cap H_k^{(v)} = \emptyset$, il vero training set e un validation set: per selezionare la migliore combinazione di parametri per la rete neurale questa viene allenata utilizzando come training set $H_k^{(t)}$, mentre l'insieme di validazione $H_k^{(v)}$ viene utilizzato per valutare le performance della rete e selezionare la migliore combinazione di iperparametri. I vertici in $H_k^{(v)}$ costituiscono il 20% dei vertici di H_k e sono stati selezionati in maniera casuale.

In tabella 4.13 sono riportati i parametri e i diversi modelli su cui è stata effettuata model selection.

Oltre che i parametri della rete si è voluto sperimentare l'effetto sull'apprendimento dato dal considerare due diversi tipi di grafo, diretto e indiretto, modificando conseguentemente la strategia di weight sharing dei vari modelli (vedi equazioni 3.4 e 3.17).

Per la valutazione delle performance vengono considerate tre diverse misure:

- Accuracy $= \frac{TP+TN}{N}$
- $F1$ measure per la classe $Spam$ (1), $F1_1 = \frac{2pr}{p+r}$ con $p = \frac{TP}{TP+FP}$ e $r = \frac{TP}{TP+FN}$, precision e recall

- Area Under the Curve: l'AUC è una misura che permette di ottenere un unico valore numerico rappresentante l'area sottostante una curva ROC; una curva ROC per un dato classificatore binario su un dataset viene ottenuta tracciando in un grafico il numero dei TP rispetto al numero dei FP al variare della soglia di discriminazione tra le classi [52].

dove TP (TN) sono gli elementi correttamente classificati come positivi[3] (negativi), mentre FP (FN) sono vertici erroneamente classificati come positivi (negativi). Rispetto ai test descritti nelle precedenti sezioni si è deciso di utilizzare due misure aggiuntive di performance ($F1$ e AUC) sia per confronto con i risultati della challenge, sia perchè, essendo i vertici del dataset non equamente suddivisi tra le due classi, la sola accuracy non è particolarmente significativa per la valutazione dei modelli: essendo circa l'80% dei vertici nella classe $NonSpam$, un classificatore che classifichi tutti i vertici in questa classe avrebbe comunque un accuracy apparentemente alta, pari appunto all'80%. La misura $F1$ invece, dipendendo sia dalla precision che dalla recall del classificatore, permette di valutarne meglio le performance: nell'esempio di Webspam, un classificatore che classificasse tutto come $NonSpam$ avrebbe precision e $F1$ pari a 0; inoltre, l'accuracy ignora la probabilità con cui un classificatore stima una classe, mentre l'AUC permette di considerare anche questo fattore. Seguendo quanto fatto dai partecipanti alla challenge, la selezione della migliore combinazione di parametri viene fatta in base all'AUC.

Risultati con vertici non etichettati

In questo primo esperimento l'obiettivo era di valutare le performance dei modelli sviluppati su questo dataset nelle stesse condizioni dei precedenti test (sezioni 4.2.1 e 4.2.2), ovvero considerando solamente la topologia del grafo e le connessioni tra i vertici: per questo motivo in tutti i test discussi in questa sezione i vertici del grafo non hanno attributi locali, e l'unica informazione disponibile all'algoritmo di apprendimento per classificare un vertice v è il suo vicinato (o i suoi predecessori e successori nel caso di grafo diretto) e la codifica effettuata dalla rete.

In tabella 4.14 sono riportati i risultati ottenuti con i due classificatori relazionali di Netkit, CDRN e WVRN, utilizzati anche per gli esperimenti descritti nelle pre-

[3] Ovvero come vertici di spam.

(a) Risultati di Netkit sul dataset Webspam con grafo diretto.

	Accuracy	*F1*	*AUC*
CDRN+RL	87.97%	0.755	0.944
CDRN	79.03%	0.021	0.883
WVRN+RL	81.17%	0.215	0.942
WVRN	78.89%	0.006	0.917

(b) Risultati di Netkit sul dataset Webspam con grafo indiretto.

	Accuracy	*F1*	*AUC*
CDRN+RL	88.36%	0.729	0.935
CDRN	78.98%	0.019	0.881
WVRN+RL	81.17%	0.215	0.942
WVRN	78.88%	0.005	0.917

Tabella 4.14: Risultati di Netkit sul dataset Webspam.

cedenti sezioni, e con l'algoritmo di collective inference Relaxation Labeling; anche per questo dataset, per valutare l'influenza della componente di collective inference nei risultati ottenuti, si riportano anche i risultati ottenuti dal solo classificatore relazionale.

In tabella 4.15a e 4.15b sono riportati i risultati sul validation set ($H_k^{(v)}$) delle migliori combinazioni di parametri per ciascuno dei modelli presi in considerazione (vedi tabella 4.13); nella tabella 4.15a è stato utilizzato il modello non completamente stazionario che permette di distinguere gli archi entranti dagli uscenti (grafo diretto) mentre nella tabella 4.15b è stato utilizzato il modello completamente stazionario (ovvero, si è considerato il grafo come indiretto). Come nelle precedenti sezioni, si indicano con NN4inG+t, NN4inG+t+s, NN4inG+t+s distinti rispettivamente il modello con target, con target e stima e con target e stima distinti, mentre con NN4inG hidden/output only i due modelli semplificati.

Nelle tabelle 4.16a e 4.16b sono riportati invece i risultati sul test set delle migliori combinazioni di parametri per ogni modello di NN4inG considerato. I risultati sono la media di 5 esecuzioni dell'algoritmo di apprendimento, al variare

(a) Risultati model selection con grafo diretto (modello non completamente stazionario).

	Accuracy	*F1*	*AUC*
NN4inG base	91.98%	0.810	0.967
NN4inG+t	90.16%	0.767	0.959
NN4inG+t+s	90.34%	0.774	0.948
NN4inG+t+s distinti	90.16%	0.774	0.961
NN4inG hidden only	84.69%	0.686	0.908
NN4inG output only	85.61%	0.712	0.913

(b) Risultati model selection con grafo indiretto (modello stazionario).

	Accuracy	*F1*	*AUC*
NN4inG base	89.98%	0.710	0.845
NN4inG+t	89.43%	0.778	0.949
NN4inG+t+s	87.43%	0.752	0.938
NN4inG+t+s distinti	89.79%	0.780	0.947
NN4inG hidden only	86.52%	0.683	0.885
NN4inG output only	86.33%	0.637	0.843

Tabella 4.15: Risultati model selection sul dataset Webspam con vertici non etichettati.

dell'inizializzazione (casuale) dei pesi della rete neurale.

La prima cosa che è possibile osservare guardando i soli risultati di Netkit in tabella 4.14 è che su Webspam il processo di inferenza collettiva svolge un ruolo fondamentale nella classificazione: le performance dei soli classificatori relazionali WVRN e CDRN sono nettamente inferiori a quelle ottenute utilizzando anche l'algoritmo di RL. In particolare si ha che, nonostante l'accuracy sia di circa l'80% in entrambi i casi, la misura $F1$ è molto bassa, pari all'$0.006, 0.005, 0.021$ e 0.019. Quello che succede è che i soli modelli relazionali classificano tutti o quasi i vertici nella classe $NotSpam$, che è la classe cui appartiene infatti circa l'80% dei vertici nel test set, ottenendo di conseguenza scarsi risultati sulla precisione della classe $Spam$. I soli modelli relazionali sono messi in difficoltà dal fatto che il numero dei

(a) Risultati test con grafo diretto (modello non completamente stazionario).

	Accuracy	*F1*	*AUC*
NN4inG base	**90.57±0.17%**	**0.790±0.005**	**0.948±0.004**
NN4inG+t	90.66±0.12%	0.781±0.002	0.955±0.000
NN4inG+t+s	90.76±0.13%	0.781±0.005	0.942±0.003
NN4inG+t+s distinti	**90.70±0.25%**	**0.783±0.008**	**0.956±0.002**
NN4inG hidden only	84.42±0.60%	0.704±0.005	0.912±0.001
NN4inG output only	85.04±0.24%	0.709±0.001	0.914±0.000

(b) Risultati test con grafo indiretto (modello stazionario).

	Accuracy	*F1*	*AUC*
NN4inG base	81.87±4.14%	0.289±0.345	0.758±0.112
NN4inG+t	**89.18±0.09%**	**0.760±0.004**	**0.941±0.001**
NN4inG+t+s	89.67±0.11%	0.759±0.005	0.936±0.002
NN4inG+t+s distinti	**89.39±0.17%**	**0.764±0.001**	**0.941±0.000**
NN4inG hidden only	88.64±0.25%	0.731±0.001	0.903±0.004
NN4inG output only	88.51±0.15%	0.700±0.009	0.894±0.002

Tabella 4.16: Risultati test sul dataset Webspam con vertici non etichettati. In grassetto sono evidenziati i migliori risultati (tra quelli di tutti i modelli) ottenuti per ciascuna percentuale p.

vertici in H_k è pari solo al 10% del totale, come osservato anche per il dataset Cora (sez. 4.2.2), e non forniscono informazione sufficiente alla classificazione.

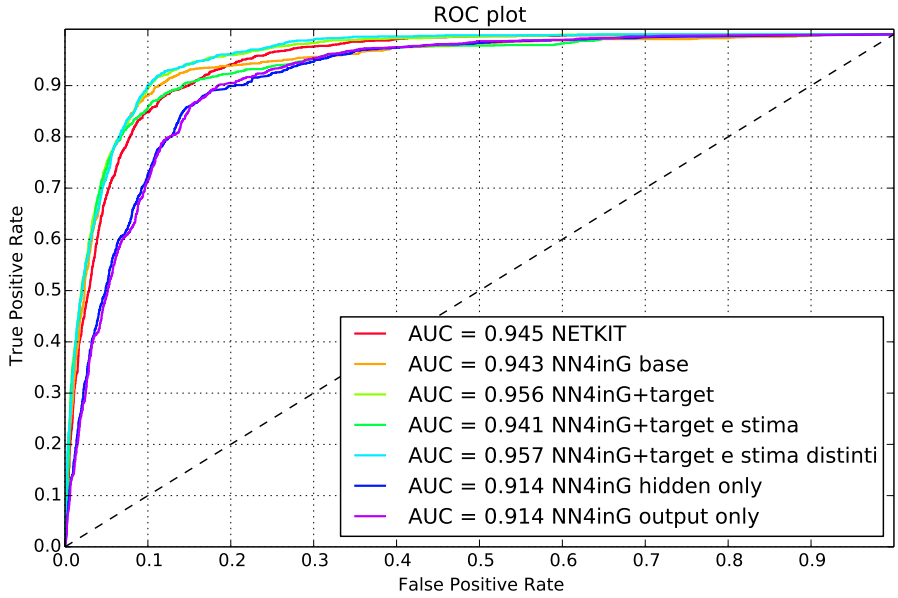
Inoltre è possibile notare che il modello che ottiene le migliori performance su questo dataset è CDRN+RL: contrariamente a quanto detto per Cora e IMDB, il classificatore completamente basato sull'omofilia WVRN+RL non raggiunge i risultati del primo, e in particolare ha $F1 = 0.215$, contro lo 0.755 di CDRN+RL. Questi risultati sembrano indicare che l'ipotesi di omofilia per la rete di Webspam non sia soddisfatta, o che comunque non sia sufficiente per l'individuazione degli host di *Spam*: intuitivamente infatti, pensando alla natura di questo dataset, si può immaginare che, mentre è plausibile che un host *NonSpam* contenga link uscenti principalmente ad host nella stessa classe, non è vero il contrario: un host *Spam*

anzi, con tutta probabilità, conterrà collegamenti a pagine di entrambe le classi.

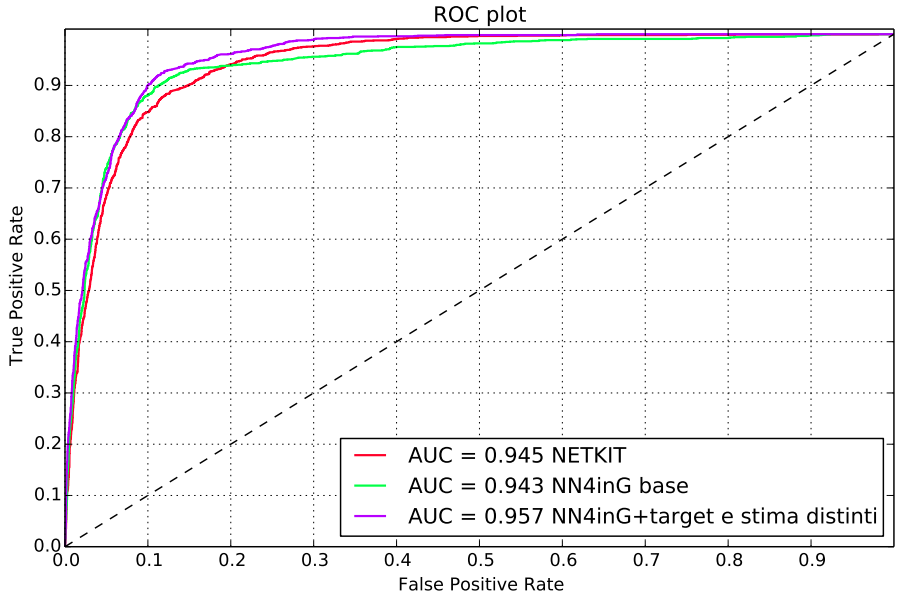
Confrontando i risultati di CDRN+RL con quelli di NN4inG, possiamo osservare che il primo ottiene $F1 = 0.755$ e $AUC = 0.944$, mentre i due migliori modelli neurali ottengono $F1 = 0.790, AUC = 0.948$ e $F1 = 0.783, AUC = 0.956$ rispettivamente, superando le performance del modello relazionale sia per quanto riguarda la misura $F1$ che l'AUC (figura 4.7).

In particolare, i risultati dei modelli neurali sono migliori quando la rete ha la capacità di distinguere tra le due tipologie di arco, entrante e uscente (modello non completamente stazionario, tabella 4.16a). Il modello che ottiene la migliore AUC, pari a 0.956, è NN4inG con target e stima distinti: rispetto al modello in cui target e stima non vengono distinti, l'AUC e l'$F1$ sono migliori, e quindi come già osservato anche per Cora e IMDB, la separazione tra le due tipologie di input sembra essere sensata.

Concordemente con quanto detto riguardo la scarsa rilevanza dell'omofilia nella rete di Webspam, i modelli con architettura modificata, che non utilizzano la codifica, NN4inG con solo hidden e output unit (sez. 3.5), sono quelli che ottengono le performance peggiori, con $AUC = 0.912$ e $AUC = 0.914$. Viceversa, contrariamente a quanto rilevato per i precedenti dataset, il modello NN4inG base (sez. 3.1), che non ha nessuna informazione sulla classe associata ai vicini di un vertice, ottiene su Webspam delle ottime performance, paragonabili a quelle del più complesso modello che utilizza sia target che stima ($AUC = 0.948$ e $F1 = 0.790$). Su questo problema appare quindi evidente il potere della codifica contestuale adattiva di NN4inG, che è sufficiente già da sola a superare le performance dei modelli relazionali; l'unione della codifica e della stima permette alla rete di migliorare ulteriormente i suoi risultati (modello NN4inG con target e stima distinti), mentre l'eliminazione della codifica (modelli con solo hidden e output unit) porta al contrario ad un degrado delle performance del classificatore. In generale, la migliore combinazione di iperparametri risultante dalla model selection per i modelli che utilizzano la codifica ha portato ad ottenere reti neurali con un alto numero di unità nascoste $(40 - 50)$, sia nel caso del grafo diretto che indiretto; nel caso dei modelli con solo output e solo hidden unit invece, le reti risultano avere poche unità $(5 - 20)$.



(a) Curva ROC per tutti i modelli di NN4inG e per Netkit CDRN+RL.



(b) Curva ROC per NN4inG base, NN4inG con target e stima distinti e per Netkit CDRN+RL.

Figura 4.7: Curva ROC per Webspam con vertici non etichettati (grafo diretto).

Risultati con vertici etichettati

In questa sezione si analizzano i risultati ottenuti con i modelli neurali proposti nel capitolo 3 introducendo gli attributi locali per i vertici del grafo di Webspam, confrontandoli con quelli dei partecipanti alla challenge.

In tabella 4.17 sono riportati i risultati dei partecipanti alla competizione.[4]

	F1	*AUC*
Abernethy et al. [8]	-	0.952
Tang et al. [53]	0.754	0.951
Filoche et al. [54]	0.738	0.927
Csalogany et al. [55]	0.793	0.877
Tian et al. [56]	0.520	0.863

Tabella 4.17: Risultati partecipanti Webspam Challenge per la track 2.

Le metodologie e i modelli utilizzati dai partecipanti sono diversi.

La strategia seguita in [53] e [56] ad esempio consiste nell'utilizzare classici classificatori per dati flat, come Support Vector Machine e alberi di decisione, incorporando nelle feature che descrivono i vertici del grafo nuovi attributi (oltre a quelli locali) che in qualche modo codifichino le informazioni sulla topologia del grafo: ad esempio, il numero totale di link di un vertice, il numero di link verso vertici di *Spam*, eccetera.

L'approccio seguito in [54] è simile, e consiste nell'utilizzare l'algoritmo Naive Bayes su un insieme di feature estratte dalla struttura del grafo (tra cui il Page Rank) per calcolare lo *spamicity score* di ogni vertice; inoltre però, sulla base dell'assunzione che il numero di archi tra vertici della stessa classe sia maggiore del numero di archi tra vertici di classi diverse, gli score vengono iterativamente modificati per far sì che la distanza tra gli score dei vertici connessi da link sia ridotta.

La stessa assunzione di omofilia viene fatta anche in [55]: la tecnica utilizzata qui prende il nome di *stacked graphical learning*, e consiste nell'utilizzare un classificatore standard e nell'aumentare, iterativamente, il set di feature che descrivono un

[4] È possibile trovare le performance (espresse come Area Under the Curve) di tutti i partecipanti in `airweb.cse.lehigh.edu/2008/web_spam_challenge/introduction.pdf`, slide 11.

vertice con feature ricavate dalle predizioni fatte dallo stesso classificatore durante la precedente iterazione per i vicini del vertice: non tutti i vicini vengono utilizzati, ma solamente i top k, selezionati secondo una qualche funzione di peso degli archi.

(a) Risultati model selection con grafo diretto (modello non completamente stazionario).

	Accuracy	*F1*	*AUC*
NN4inG base	91.43%	0.821	0.972
NN4inG+t	91.43%	0.809	0.966
NN4inG+t+s	91.25%	0.784	0.970
NN4inG+t+s distinti	90.71%	0.796	0.968
NN4inG hidden only	88.52%	0.730	0.919
NN4inG output only	86.15%	0.737	0.916

(b) Risultati model selection con grafo indiretto (modello stazionario).

	Accuracy	*F1*	*AUC*
NN4inG base	89.98%	0.785	0.962
NN4inG+t	89.61%	0.769	0.955
NN4inG+t+s	89.43%	0.769	0.954
NN4inG+t+s distinti	89.07%	0.769	0.950
NN4inG hidden only	87.06%	0.684	0.874
NN4inG output only	85.79%	0.625	0.837

Tabella 4.18: Risultati model selection sul dataset Webspam con vertici etichettati.

I vincitori della challenge infine, [8], utilizzano direttamente l'informazione sulla struttura del grafo nell'algoritmo di learning, insieme alle feature estratte dal contenuto locale dell'host. Il classificatore usato è una SVM standard, modificata però con una tecnica chiamata *graph regularization*: si aggiunge alla funzione obiettivo della support vector machine un termine di regolarizzazione che tiene conto dell'influenza dei vicini nel calcolo del target del vertice.

Come nella precedente sezione, si sono considerati i modelli e le diverse combinazioni di parametri indicate in tabella 4.13, su cui è stata effettuata model selection

seguendo la metodologia descritta nella sezione 4.2.3. I risultati sul validation set della miglior combinazione di parametri per ciascun modello sono riportati in tabella 4.18.

In tabella 4.19 sono elencati infine i risultati ottenuti dai sei diversi modelli di NN4inG sul test set della competizione; in figura 4.9 si riportano degli esempi delle curve d'apprendimento dei modelli neurali su questo dataset.

(a) Risultati test con grafo diretto (modello non completamente stazionario).

	Accuracy	*F1*	*AUC*
NN4inG base	**91.07±0.37%**	**0.803±0.006**	**0.960±0.002**
NN4inG+t	**91.47±0.16%**	**0.798±0.004**	**0.961±0.000**
NN4inG+t+s	91.59±0.33%	0.800±0.008	0.956±0.003
NN4inG+t+s distinti	91.30±0.11%	0.797±0.004	0.959±0.001
NN4inG hidden only	89.08±0.17%	0.756±0.008	0.932±0.006
NN4inG output only	79.29±0.00%	0.636±0.000	0.869±0.000

(b) Risultati test con grafo indiretto (modello stazionario).

	Accuracy	*F1*	*AUC*
NN4inG base	**91.18±0.24%**	**0.797±0.006**	**0.953±0.002**
NN4inG+t	89.70±0.14%	0.771±0.003	0.946±0.002
NN4inG+t+s	**90.50±0.42%**	**0.779±0.009**	**0.949±0.001**
NN4inG+t+s distinti	89.72±0.35%	0.769±0.006	0.944±0.003
NN4inG hidden only	88.35±0.80%	0.703±0.054	0.882±0.041
NN4inG output only	88.67±0.11%	0.709±0.002	0.896±0.002

Tabella 4.19: Risultati test sul dataset Webspam con vertici etichettati. In grassetto sono evidenziati i migliori risultati (tra quelli di tutti i modelli) ottenuti per ciascuna percentuale p.

Come per i precedenti sperimenti, i migliori risultati ottenuti dai modelli neurali si hanno utilizzando modelli non completamente stazionari, in cui vengono distinti con due set di pesi diversi i contributi dei predecessori di un vertice (host che contengono un link a lui) e quelli dei successori (host che sono linkati da qualche pagina dell'host associato al vertice), in tabella 4.19a.

Rispetto ai risultati ottenuti da NN4inG sul dataset senza attributi locali, discussi nella precedente sezione (tabella 4.16a), si può notare come l'introduzione dell'informazione sul contenuto delle pagine di un host permetta di ottenere un miglioramento nelle performance dei modelli neurali: ad esempio, il modello NN4inG base ottiene nel grafo senza attributi locali $accuracy = 90.57\%, F1 = 0.790$ e $AUC = 0.948$, mentre in questa versione del dataset con vertici etichettati ottiene $accuracy = 91.07\%, F1 = 0.803$ e $AUC = 0.960$. Analogamente a quanto osservato nella precedente sezione per i risultati ottenuti con i vertici non etichettati, di nuovo i modelli che utilizzano la codifica risultano avere molte unità nascoste $(40 - 50)$, mentre i modelli con solo output e solo hidden unit risultano avere poche unità $(2 - 20)$.

Il vincitore della competizione svoltasi su questo dataset, [8], ha AUC pari al 0.952, e si distanzia solo dello 0.001 dal secondo classificato, [53], che invece ha $AUC = 0.951$ e $F1 = 0.754$; i risultati ottenuti da NN4inG superano entrambi, e sono uguali a $F1 = 0.798$ e $AUC = 0.961$ per il modello NN4inG con target, e a $F1 = 0.803$ e $AUC = 0.960$ per il modello NN4inG base, il modello "vincente" in base alla model selection effettuata (tabella 4.18). Anche i modelli NN4inG con target e stima e NN4inG con target e stima distinti superano i vincitori della challenge, con AUC uguale a 0.956 e 0.959 rispettivamente, mentre, come osservato anche nella precedente sezione per il grafo senza attributi locali, i modelli più semplici con solo hidden e output unit ottengono le performance peggiori (figura 4.8).

Anche da questa seconda sperimentazione sul dataset Webspam è possibile trarre conclusioni analoghe a quelle della precedente sezione: di nuovo, è evidente il contributo decisivo della codifica adattiva e contestuale di NN4inG, che addirittura in questo caso, insieme agli attributi locali, riesce ad ottenere una delle due migliori performance di classificazione tra tutti i modelli neurali e tra i partecipanti alla competizione; inoltre è stato possibile valutare tramite questi esperimenti l'effettiva capacità dei modelli NN4inG di trattare contemporaneamente l'informazione locale e relazionale dei vertici della rete, coniugandole tramite il meccanismo adattivo che permette ai modelli di autoregolare l'influenza di entrambe le componenti.

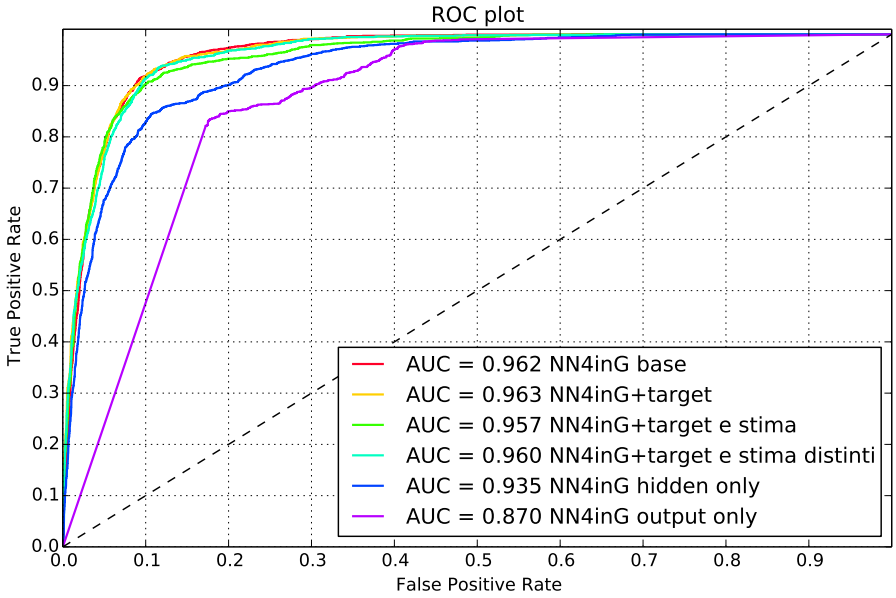


Figura 4.8: Curva ROC per Webspam con vertici etichettati (grafo diretto).

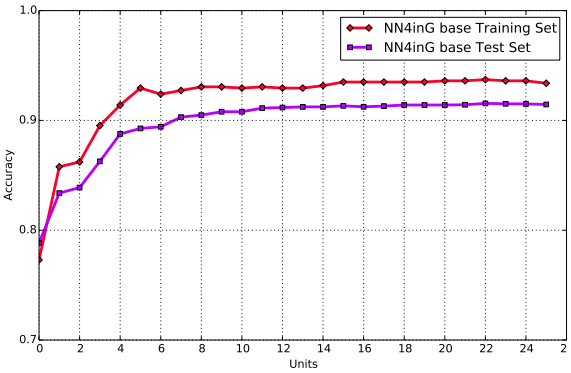
4.3 Discussione

In questo capitolo sono stati presentati i risultati sperimentali dell'applicazione dei modelli proposti nel capitolo 3 a diversi tipi di dataset.

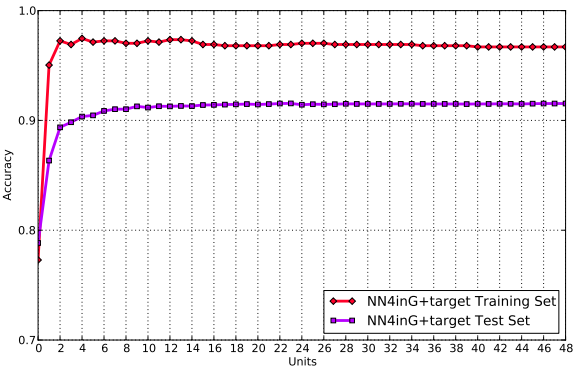
Nella sezione 4.1 si sono riportati gli esperimenti effettuati su due dataset su cui sono definiti task di on-graph learning, il dataset Alcani e il dataset PTC, per dimostrare l'applicabilità del framework implementato anche a problemi di classificazione su grafi già affrontati nell'ambito del Machine Learning per domini strutturati.

Nella sezione 4.2 invece ci si è focalizzati sul problema dell'in-graph learning, e sono state analizzate le performance delle diverse varianti di Neural Network for In-Graph su tre diversi dataset relazionali, IMDB, Cora e Webspam, confrontandole con quelle di classificatori relazionali dello Statistical Relational Learning, discussi in sezione 2.8.2, implementati nel framework Netkit.

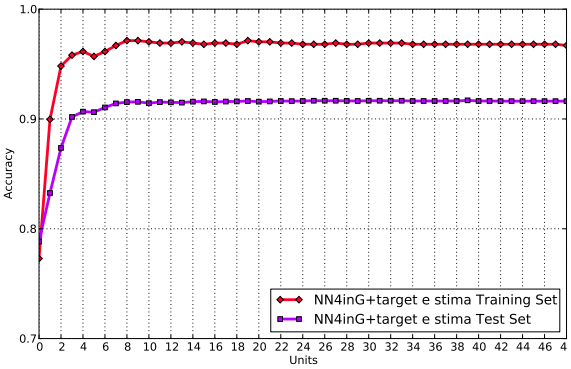
È emerso durante l'analisi che nel primo dataset considerato, IMDB (sez. 4.2.1), la classe di un'entità è fortemente legata alla classe delle entità con cui è in relazione: vale cioè per questa rete l'ipotesi di omofilia. Confrontando infatti i risultati ottenuti



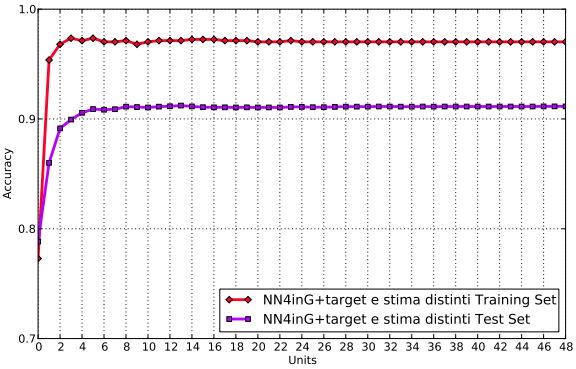
(a) NN4inG base.



(b) NN4inG+target.



(c) NN4inG+target e stima.



(d) NN4inG+target e stima distinti.

Figura 4.9: Curva d'apprendimento dell'accuracy di due varianti di NN4inG sul dataset Webspam con nodi etichettati.

dai due classificatori relazionali di Netkit, si nota come il migliore sia il Weighted-Vote Relational Neighbor Classifier (WVRN), un classificatore che semplicemente stima la probabilità che un vertice appartenga a una data classe come la somma dei suoi vicini che sono in quella classe; inoltre, nel dataset IMDB è soddisfatta l'ipotesi per cui la classe di un vertice dipende solo dal suo vicinato di raggio 1, come si può vedere chiaramente dal fatto che, anche per una percentuale bassa di vertici etichettati, il meccanismo di inferenza collettiva non apporta grandi benefici alle performance di nessuno dei due classificatori relazionali. I risultati ottenuti con i diversi modelli di NN4inG sono concordi con queste particolari proprietà del dataset: il modello base, che non ha nozione diretta della classe associata ai vicini di un vertice, ma che invece costruisce una sua codifica adattiva, è quello che ottiene le performance peggiori, mentre tutti gli altri modelli (NN4inG con target, target e stima, target e stima distinti, NN4inG con solo hidden e solo output unit) che in una qualche misura utilizzano target e eventualmente stima della classe dei vicini raggiungono e superano le performance di WVRN.

In particolare, i modelli NN4inG con solo output unit e NN4inG con solo hidden unit, che eliminano del tutto la codifica adattiva dal modello neurale e che basano l'apprendimento solamente su target e stima dei vicini, raggiungono e in alcuni casi superano le performance di quelli che utilizzano la codifica. I risultati ottenuti su questo dataset giustificano in generale l'introduzione nel modello dell'informazione esplicita sul target dei vertici vicini, in quanto grazie a questa si riesce a superare il limite del modello base di NN4inG, che come ipotizzato nel precedente capitolo si trova svantaggiato nell'affrontare un task completamente basato sull'omofilia della rete, non potendo accedere direttamente al target dei vicini di un vertice. Il fatto poi che i modelli che non utilizzano la codifica contestuale siano quelli che ottengono i risultati migliori fa intuire che in problemi come IMDB, in cui l'informazione relazionale utile alla classificazione è solo quella contenuta nelle classi del vicinato di un vertice, possa essere conveniente utilizzare modelli più semplici, come appunto NN4inG con solo output unit e NN4inG con solo hidden unit, che ottengono risultati analoghi a quelli dei modelli che utilizzano anche la codifica contestuale ma con una architettura meno complessa.

Quanto appena detto è confermato anche dai risultati sul dataset Cora (sez. 4.2.2), dove come per il precedente continua a valere l'ipotesi di omofilia: anche

in questo caso infatti, il miglior classificatore relazionale per Netkit risulta essere WVRN+RL, e il modello NN4inG base non raggiunge performance all'altezza di quelle degli altri modelli neurali. A differenza di IMDB però, su questo dataset il processo di inferenza collettiva ha un notevole impatto sui risultati di Netkit, soprattutto quando solo pochi vertici del grafo appartengono al training set: al contrario di quanto accadeva infatti per IMDB, anche i modelli neurali corrispondenti alle migliori combinazioni di iperparametri (selezionati tramite model selection) sono costituiti da un numero di unità nascoste non piccolo $(20 - 50)$, ad indicare la necessità di andare oltre il semplice vicinato diretto di un vertice per considerare invece un contesto di raggio maggiore di 1. I risultati migliori sono ottenuti dai modelli di NN4inG che combinano target e stima alla codifica contestuale, che per $p = 90\%$ e $p = 60\%$ superano quelli ottenuti da WVRN+RL: questi risultati sono a dimostrazione dell'effettiva utilità dei modelli più articolati, che coniugano sia la codifica che l'informazione del target e della stima, che nel dataset IMDB non era stato possibile osservare.

Tramite la sperimentazione sul terzo dataset, Webspam, è stato invece possibile analizzare il comportamento dei modelli neurali proposti su una rete per la quale l'ipotesi di omofilia non è così forte come lo è per Cora e IMDB: confrontando su questi dati NN4inG e i classificatori relazionali di Netkit, è emerso come il modello neurale sia effettivamente in grado di superare i risultati di quest'ultimi, sfruttando la maggiore flessibilità data dalla codifica contestuale adattiva. Il modello NN4inG base risulta essere infatti praticamente alla pari dei più complessi modelli che considerano target e stima, a dimostrazione dell'effettivo potere della sola codifica contestuale, che il confronto con gli altri due dataset non aveva permesso di evidenziare. L'unione poi della codifica con il target e la stima nel modello NN4inG con target e stima distinti permette di ottenere performance ancora migliori. Etichettando i vertici della rete con gli attributi locali derivati da quelli disponibili ai partecipanti alla competizione su questo dataset, ci si è poi confrontati con i risultati di quest'ultimi: il miglior modello selezionato tramite model selection risulta essere il modello NN4inG base, a conferma di quanto detto per il grafo non etichettato sul potere della sola codifica contestuale, che in test ottiene un punteggio di AUC pari a 0.960, contro l'AUC di 0.951 e 0.952 del primo e del secondo classificato alla Webspam Challenge. Mentre i dataset IMDB e Cora non hanno permesso, a causa

delle loro particolari proprietà, di far emergere l'effettiva utilità del meccanismo di codifica contestuale e adattiva di NN4inG, su Webspam questa appare evidente: se il target dell'apprendimento non è dipendente solamente dal target dei vicini di un vertice, la codifica fornisce al modello uno strumento adattabile (in maniera automatica) con cui rappresentare l'informazione contestuale nel modo più consono alla risoluzione del task; inoltre, dalle performance dei modelli su IMDB e Cora si evince che questa maggiore adattabilità non va a discapito del modello per task in cui invece l'ipotesi di omofilia è valida: in conclusione i modelli proposti sono risultati in grado di affrontare entrambi i tipi di problemi, ottenendo risultati comparabili a quelli dello SRL su grafi con omofilia, e migliori su grafi in cui questa ipotesi non sussiste.

Capitolo 5

Conclusioni

Nella tesi è stato affrontato il problema dell'apprendimento automatico in domini strutturati, con una particolare attenzione a problemi di *in-graph learning*, con lo scopo di sviluppare nuovi modelli per questo task.

Un problema di in-graph learning è definito su un grafo solo parzialmente etichettato, e consiste nell'apprendere la funzione di trasduzione che lega ogni vertice del grafo al suo target, avvalendosi del sottoinsieme di vertici etichettati, per poi applicare quanto appreso per la predizione del target della parte di vertici del grafo senza etichetta. Questo tipo di problemi è contrapposto ai task di *on-graph learning*, nei quali il learning della funzione di trasduzione avviene su grafi completamente etichettati, e l'inferenza riguarda un distinto insieme di grafi per i vertici dei quali non è conosciuto il target. Al contrario di quest'ultimo, un problema in-graph può essere visto come un problema di apprendimento semisupervisionato: le entità etichettate e quelle non etichettate sono unite in una unica rete dalle relazioni che intercorrono tra loro, e il processo d'apprendimento può sfruttare questa particolarità del problema per incorporare nel modello anche le informazioni contestuali date dai vertici non etichettati.

Il lavoro di tesi si è concentrato sullo sviluppo di nuovi modelli per l'apprendimento in-graph, che coniugano i punti di forza di modelli allo stato dell'arte per l'on-graph learning con quelli dei modelli relazionali dello Statistical Relational Learning per problemi in-graph. I modelli sviluppati rappresentano una estensione del modello *Neural Network for Graphs* (NN4G), una rete neurale precedentemente

applicata al problema dell'apprendimento on-graph, dalla quale i modelli proposti ereditano una serie di importanti proprietà.

I modelli sviluppati utilizzano un meccanismo di codifica, tramite il quale ad ogni vertice del grafo di input viene assegnata una rappresentazione vettoriale che riassume l'informazione locale, data dall'input label di quel vertice, e contestuale, data dalla codifica dei suoi vicini; tramite l'applicazione della funzione di codifica ad ogni vertice del grafo si ottiene una nuova rappresentazione dello stesso, un grafo isomorfo, in cui ogni vertice viene etichettato con la sua codifica. L'architettura neurale è più semplice rispetto a quella di altri modelli neurali per l'apprendimento in domini strutturati, in quanto non contiene connessioni ricorsive tra neuroni: l'architettura è costruttiva e feed-forward, e non è quindi necessario utilizzare nessun algoritmo iterativo per la convergenza del calcolo della codifica di vertici interconessi, o presupporre la validità di ipotesi sulla matrice dei pesi affinchè sia possibile applicare il modello anche a grafi indiretti e contenenti cicli. La codifica inoltre non è prestabilita e fissata, ma viene calcolata in maniera adattiva: l'enconding di ogni vertice viene determinato dallo stesso processo di apprendimento, che contemporaneamente quindi determina la rappresentazione strutturata della rete sotto forma di stato (ovvero, un grafo isomorfo a quello di input in cui ogni vertice viene etichettato con la codifica calcolata dalla rete neurale) sia l'output corrispondente. Rispetto ad altri approcci inoltre, i modelli hanno la capacità di incorporare nella codifica di un vertice informazioni contestuali per intorni del vertice di dimensione arbitraria: la dimensione del contesto preso in considerazione dipende dal numero di unità presenti nella rete che però, essendo l'architettura neurale costruttiva, è determinato automaticamente dall'algoritmo di apprendimento stesso, che quindi determina in maniera adattiva anche la dimensione del contesto più indicata per la risoluzione del problema.

Al fine di estendere la Neural Network for Graphs al task dell'apprendimento in-graph, si sono analizzati diversi modelli allo stato dell'arte per questo problema, principalmente appartenenti al mondo dello Statistical Relational Learning (SRL). Dall'analisi di questi modelli sono emerse alcune loro particolarità, tra le quali ricordiamo l'assunzione, dovuta a problemi di intrattabilità computazionale, della validità dell'ipotesi di Markov del prim'ordine nella rete (il target di un vertice dipende solamente dai suoi vicini diretti, e non da tutto il grafo), e l'ipotesi della

presenza di omofilia nella rete, che influenza fortemente la strategia di questi classificatori relazionali. Secondo l'ipotesi di omofilia, la probabilità che un vertice sia collegato ad un altro è più alta se questi due appartengono alla stessa classe: sulla base di questa assunzione, tutti i modelli dello SRL analizzati utilizzano, in modi diversi, il target dei vicini di un vertice per inferirne la classe, incorporando questa informazione nella definizione della funzione di stima.

L'architettura costruttiva e feed-forward dei modelli neurali sviluppati durante la tesi permette di applicare i modelli a problemi di in-graph learning definiti su qualsiasi tipo di rete, anche contenente cicli, senza necessità di introdurre meccanismi di iterazione per la loro gestione. La codifica contestuale adattiva, oltre a non ricadere nell'ipotesi di Markov in quanto, come detto sopra, il contesto considerato non è limitato ai soli vicini diretti, rappresenta inoltre uno strumento molto più flessibile rispetto all'informazione relazionale data dal target dei vicini di un vertice utilizzata dai modelli di SRL, in quanto non legata ad una specifica visione pregressa del problema d'apprendimento, come l'omofilia. Il meccanismo costruttivo con cui questa codifica viene stabilita fornisce uno strumento che permette inoltre di trattare contemporaneamente le tre componenti locale, relazionale e di inferenza collettiva di un modello dello SRL, integrandole in un unico modello adattivo, evitando così l'introduzione di algoritmi iterativi tipici invece dell'approccio dello SRL e i limiti che questi comportano, come la necessità di tecniche euristiche e di vincoli per assicurarne la convergenza.

Il primo modello proposto, *Neural Network for In-Graph* (NN4inG), consiste nell'adattamento della Neural Network for Graphs al task dell'in-graph learning. Nei successivi modelli sviluppati, NN4inG con target, NN4inG con target e stima e NN4inG con target e stima distinti, queste caratteristiche sono state integrate con quanto osservato dall'analisi dei modelli di SRL. In NN4inG con target, si introduce nel modello il target dei vicini (etichettati) di un vertice, che va ad affiancare la codifica contestuale per fornire alla rete neurale un accesso diretto a questa informazione, come avviene nei classificatori relazionali dello SRL. In NN4inG con target e stima e NN4inG con target e stima distinti, al target dei vicini etichettati si aggiunge la stima del target per i vicini del vertice per i quali questo è sconosciuto, così come avviene in un modello di SRL per opera della componente di inferenza collettiva. A differenza di quest'ultima però, i modelli neurali non utilizzano mec-

canismi iterativi di raffinamento della stima, che come accennato comportano una serie di problemi, come il fatto che la convergenza non sia sempre assicurata a meno che non vengano imposte alcune condizioni, ma possono utilizzare la stima ottenuta nelle precedenti fasi del processo di costruzione automatica e adattiva del modello. Rispetto al primo, e diversamente da quanto avviene nei modelli relazionali studiati, in NN4inG con target e stima distinti il contributo di target e stima viene separato, per permettere alla rete di neurale di determinare il loro peso in maniera indipendente, data la diversa natura dei due. Ancora, va evidenziato che rispetto ai modelli relazionali, l'approccio neurale è adattivo: l'input proveniente da target e stima (così come quello della codifica) è pesato, e i pesi vengono modificati dal processo di apprendimento per migliorare le predizioni fatte dalla rete (cioè diminuire l'errore commesso): questo porta il modello ad avere una maggiore flessibilità, rispetto ai classificatori relazionali osservati.

Sulla base di osservazioni fatte sulle performance di questi modelli, e su quelle dei modelli relazionali stessi su alcuni esperimenti effettuati, ci si è domandati se e/o quanto, dopo l'inserzione dell'informazione esplicita su target e stima nel modello, fosse proficuo utilizzare la più complessa rappresentazione dell'informazione contestuale effettuata dallo strato di unità nascoste, più flessibile e non direttamente collegata al target o alla sua stima. Per rispondere a questa domanda si sono sviluppati quindi due modelli semplificati, nei quali l'apprendimento viene completamente basato su target e stima, ovvero, dei modelli più vicini all'ipotesi di omofilia. Il primo modello, NN4inG con solo hidden unit, è stato ottenuto eliminando dall'architettura neurale lo strato di unità di output: ad ogni passo di costruzione della rete, vi si aggiunge un nuovo strato di unità "nascoste", collegate a tutti i precedenti, e utilizzate direttamente per calcolare l'output. Il risultato di questa modifica è che la stima della rete per un vertice viene calcolata solo sulla base del target dei suoi vicini etichettati e della stima effettuata da tutti gli altri strati della rete sui suoi vicini, etichettati e non. Il secondo modello invece, NN4inG con solo output unit, rimuove dalla rete lo strato di unità nascoste e la loro codifica, ottenendo così di nuovo un modello basato solo su target e stima, ma con una architettura ancora più semplice, in cui oltre al target solo la stima "al passo precedente" viene utilizzata, rendendo quest'ultimo modello il più vicino a quelli relazionali dello SRL.

I modelli sviluppati rappresentano un approccio generale al problema dell'apprendimento nei domini strutturati, e non sono limitati al solo caso dell'apprendimento within-network: possono essere applicati sia a problemi di on-graph learning che di in-graph learning, e nella parte sperimentale di questa tesi sono stati presentati i risultati ottenuti su task appartenenti ad entrambe le categorie. Come esempio di task di apprendimento tra grafi si sono valutate le performance della rete neurale (nella forma del modello base) su due problemi tratti dal mondo della chimica, la predizione della temperatura di ebollizione di un insieme di alcani e la classificazione di composti chimici in base alla loro tossicità, che hanno permesso un confronto diretto, di collaudo del modello base, con risultati allo stato dell'arte di modelli similari per problemi on-graph.

Tramite la sperimentazione su diversi problemi di in-graph learning è stato invece possibile valutare l'efficacia dei modelli proposti su task di questo tipo. Nei primi due dataset considerati, IMDB e Cora, si è riscontrato che l'ipotesi di omofilia era fortemente soddisfatta: vertici appartenenti alla stessa classe sono collegati con una probabilità maggiore di vertici in classi diverse, e come conseguenza di ciò la classe di un vertice è strettamente dipendente dalla classe dei vertici nel suo vicinato; in particolare, mentre in Cora la topologia della rete è insita nei dati stessi (i paper di questo dataset sono connessi in base alle loro citazioni), in IMDB le connessioni sono state determinate a posteriori collegando entità che condividevano alcune proprietà (film con lo stesso produttore): il dataset relazionale IMDB quindi è stato, in un certo senso, costruito ad hoc collegando tra loro entità simili. Inoltre in entrambi i dataset non sono presenti label di input per i vertici dei grafi, e l'unica informazione disponibile per l'apprendimento è data dalla topologia della rete. I risultati su questi dataset hanno mostrato come in problemi quasi totalmente determinati dall'ipotesi di omofilia la sola codifica contestuale non basti al raggiungimento di buone performance di classificazione: come ipotizzato infatti, il modello NN4inG base, non avendo conoscenza diretta dell'informazione data dalla classe dei vicini, fondamentale in questi due task, non può competere con i modelli che invece possono sfruttare questa informazione. In particolare, per il dataset IMDB è emerso che i soli target dei vicini erano sufficienti per una buona classificazione, anche per percentuali basse di vertici con target noto: in queste condizioni persino il meccanismo di inferenza collettiva dei modelli di SRL considerati risulta non necessario, e la

selezione del modello per le reti neurali risulta in reti con pochissime unità; a causa di queste sue particolari proprietà su IMDB anche i modelli neurali più semplici, che non usano la codifica ma solamente il target e la stima (NN4inG con solo hidden unit e NN4inG con solo output unit), ottengono performance confrontabili con quelle degli altri modelli più complessi. Su Cora invece i migliori modelli neurali risultano avere un maggior numero di unità, e i modelli che uniscono l'informazione contestuale della codifica a quella di target e stima sono quelli che in generale ottengono le performance migliori. Complessivamente i risultati ottenuti su Cora e IMDB giustificano l'introduzione nei modelli neurali di meccanismi per la gestione esplicita dell'informazione di target e stima, che è risultata essere fondamentale per l'ottenimento di buone performance in problemi così fortemente influenzati dall'ipotesi di omofilia; in problemi come IMDB in particolare, il solo target è risultato essere necessario e sufficiente allo scopo, mentre in problemi come Cora, più sensibili alla bassa percentuale di vertici etichettati, è stato possibile apprezzare i benefici derivanti dall'uso congiunto di stima e target.

La sperimentazione sul terzo dataset ha invece permesso di mettere pienamente in luce l'effettivo potere aggiunto dato dalla presenza della codifica contestuale adattiva. Il dataset, Webspam, appartiene ad una competizione indetta nel 2007, nella quale si sono confrontate diverse metodologie allo stato dell'arte per l'apprendimento relazionale; questo dataset non si basa sull'assunzione di omofilia come i precedenti, e ciò ha reso possibile valutare i modelli proposti su un problema diverso, in cui la stima non dipende esclusivamente dalla classe degli immediati vicini di un vertice. Una prima parte della sperimentazione su questo dataset è consistita nel confronto con gli stessi modelli relazionali utilizzati per i precedenti; per effettuare un confronto alla pari, anche per la rete di Webspam non si sono considerate le label di input associate ai vertici. I risultati ottenuti hanno dimostrato che il meccanismo di codifica adattiva dei modelli neurali permette effettivamente di superare le performance dei soli modelli relazionali, sottolineandone l'importanza: in particolare, il modello NN4inG base, avvalendosi della sola codifica, supera i risultati dei modelli di SRL, e il modello che invece coniuga codifica, target e stima è quello che ottiene le migliori performance, distanziando nettamente quelle dei modelli relazionali. Nella seconda parte della sperimentazione su questo dataset si sono introdotte le label di input per i vertici della rete, per poter confrontare i

nostri modelli con quelli dei partecipanti alla competizione di cui questo dataset è stato oggetto, e per poter valutare il comportamento dei modelli neurali in presenza di vertici etichettati con label locali di input. Rispetto ai risultati ottenuti senza, l'introduzione delle label di input ha comportato un ulteriore miglioramento nelle performance, a dimostrazione della capacità dei modelli di sfruttare al meglio sia le informazioni locali che relazionali della rete. Anche in questo caso i risultati hanno confermato il potere della codifica contestuale e adattiva di NN4inG: il modello NN4inG base, risultato il migliore dopo il processo di model selection, sorpassa in test le performance di tutti i partecipanti alla competizione, tramite l'utilizzo della sola codifica. La combinazione di codifica, target e stima ottiene delle performance analoghe, e ancora migliori di quelle dei partecipanti alla challenge. Il confronto su questo dataset dei risultati dei modelli proposti sia con quelli dei modelli relazionali dell'SRL, sia con quelli degli altri approcci dei partecipanti alla challenge, ha quindi permesso di evidenziare i vantaggi apportati dall'uso congiunto dei tre diversi tipi di informazione relazionale, ovvero codifica, target e stima. In particolare, sono emerse da questi esperimenti le potenzialità del meccanismo di codifica adattiva, determinata dai modelli neurali in maniera automatica e legata al raggiungimento dell'obiettivo d'apprendimento, rispetto al solo utilizzo di target e stima dei vicini immediati.

In conclusione, in questa tesi è stata proposta una famiglia di modelli per il trattamento di problemi di in-graph learning, che permette di combinare le idee proprie dello Statistical Relational Learning con i vantaggi di un modello neurale. In particolare, il modello che coniuga codifica, target e stima (NN4inG con target e stima distinti) è risultato in grado di ottenere buoni risultati in ognuno degli esperimenti effettuati: l'uso della codifica fornisce uno strumento flessibile e potente per estrarre l'informazione contestuale dei vertici della rete, mentre l'introduzione del target e della stima rende il modello in grado di affrontare adeguatamente anche problemi basati sull'omofilia. Rispetto ai modelli relazionali analizzati, i modelli proposti nella tesi sono adattivi, in quanto sono in grado di determinare automaticamente quanto i contributi delle diverse informazioni contestuali (codifica, target e stima) devono influire nell'output della rete. Soprattutto, i modelli proposti non sono totalmente legati ad una specifica ipotesi sulla natura dei dati (l'omofilia), ma piuttosto grazie al meccanismo di codifica possono costruire la più appropriata

rappresentazione del contesto in base allo specifico task da trattare.

Sviluppi Futuri

La tesi rappresenta un primo approccio al problema dell'in-graph learning via reti neurali, e in quanto tale apre la strada a successivi sviluppi in questo campo tramite le tecniche qui discusse. A livello applicativo, la grande quantità di problemi che sono descrivibili come un task di in-graph learning fornisce una serie di altri possibili task interessanti cui è possibile applicare i modelli neurali discussi, appartenenti ai più diversi campi, come il web o le reti sociali; inoltre, un qualsiasi dataset anche non relazionale può essere trasformato in un dataset di networked data, definendo una funzione di somiglianza tra gli elementi appartenti all'insieme dei dati che sia poi traducibile in una rete di collegamenti (come nel caso di IMDB), rendendo il dominio di applicabilità dei nuovi modelli potenzialmente molto ampio.

Un possibile proseguimento di questo lavoro infine è verso la descrizione dei modelli presentati in questa tesi in un framework teorico uniforme, che permetta di descrivere in maniera formale non solo le diverse varianti proposte, ma anche in generale modelli per l'apprendimento sia on-graph che in-graph. Un framework teorico in grado di descrivere uniformemente modelli per il trattamento di entrambi i problemi (come NN4inG), basato sul concetto generale di trasduzione input-output isomorfa (di cui le funzioni oggetto di in-graph e on-graph learning sono dei sottocasi particolari) renderebbe possibile inoltre lo sviluppo di nuovi modelli ibridi per entrambi i task in maniera strutturata e organica, permettendo un'analisi rigorosa delle loro proprietà computazionali e di apprendimento.

Appendice A

Software

Durante il lavoro di tesi è stato realizzato un framework, NN4inG, utilizzato per tutti gli esperimenti e test effettuati. Il framework può essere utilizzato sia per task di classificazione e regressione on-graph (sez. 4.1), sia per il trattamento di task di within-network learning (sez. 4.2).

Nella sezione A.1 viene descritto il software realizzato, mentre nella sezione A.2 si riporta il formato di input del programma e nella sezione A.3 i file di output prodotti dal programma al termine dell'allenamento. Infine nella sezione A.4 è riportato un breve manuale d'uso del software.

A.1 Framework

Il framework realizzato permette di creare e allenare una rete neurale per grafi in grado di affrontare sia task di on-graph learning, sia task di in-graph learning.

Nell'implementazione si è seguito un approccio composizionale: come è possibile vedere in figura A.1, tutte le classi che realizzano i vari modelli discussi nella tesi derivano da un'unica superclasse, `NN4G`, che contiene tutti i campi che descrivono la rete neurale e che implementa tutte le funzioni base comuni ai vari modelli. In particolare, contiene i metodi che permettono di inizializzare e resettare la rete neurale, caricare dataset per l'on-graph learning, produrre i diversi file di output e gestire tutto l'allenamento della rete: in questa classe sono infatti implementati

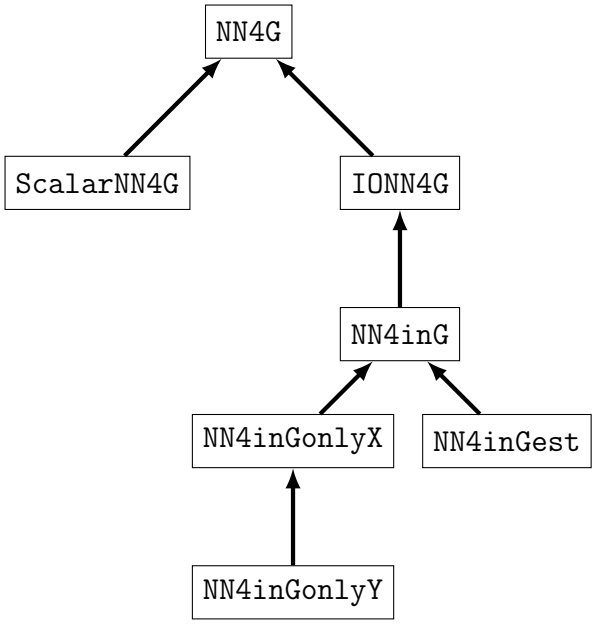


Figura A.1: Schema delle classi che implementano i diversi modelli.

il ciclo di training della rete (vedi pseudocodice nel listato 1) e i diversi algoritmi d'apprendimento (sez. A.1.1).

Le due sottoclassi immediate di `NN4G`, `ScalarNN4G` e `IONN4G`, specializzano la superclasse con i metodi per effettuare i due diversi task dell'on-graph learning, rispettivamente trasduzione scalare e trasduzione IO. In particolare, entrambe le classi implementano una specifica versione delle due funzioni principali per l'allenamento della rete, la funzione che calcola il gradiente per l'allenamento delle output units, e la funzione che calcola la correlazione con l'errore e il suo gradiente per l'allenamento di nuove hidden units: queste funzioni realizzano le formule d'apprendimento riportate nella sezione 2.7, calcolando l'incremento per l'aggiornamento dei pesi delle unità.

La classe `NN4inG` permette invece di trattare i task di in-graph learning, implementando i diversi modelli discussi nelle sezioni 3.1, 3.2 e 3.3. La classe deriva da `IONN4G`, in quanto in un certo senso è possibile vedere questo tipo di task come un particolare sottocaso di una funzione di trasduzione input-output isomorfa: la sostanziale differenza, a livello implementativo, oltre al fatto di avere un unico grafo a disposizione, sta nel dover gestire la distinzione, all'interno dei vertici dello stesso grafo, tra vertici di training, vertici di test e vertici non noti, ovvero tra vertici

Algoritmo 1 Pseudo codice per l'allenamento della rete neurale.

```
function TRAIN
    count ← 0
    while !THRESHOLDCONDITION() ∧ count < max_epochs do
        for c = 1, ··· , output_learning_epochs do
            TRAINOUTPUTUNIT()
            STOREOUTPUT()
            if CHANGECONDITION() then break;
            end if
        end for
        if !THRESHOLDCONDITION() ∧ N < max_hidden_units then
            ADDHIDDENUNIT(count)
        end if
        count ← count + 1
    end while
end function
```

per i quali è noto il target, vertici per i quali il target non può essere usato nell'allenamento perchè di test, e vertici per i quali non si ha effettivamente nessuna informazione. Questa sottoclasse quindi aggiunge i metodi che permettono la distinzione dei diversi tipi di vertici (ad esempio, per il calcolo delle funzioni d'errore) e il caricamento dei dataset per l'in-graph learning, oltre a modificare la funzione citata prima per il calcolo del gradiente della funzione di correlazione delle hidden units[1].

La classe `NN4inGest` implementa invece il modello descritto nella sezione 3.4, e permette quindi di creare una rete neurale che utilizza due diversi insiemi di pesi per il target dei vicini noti e la stima di quelli non noti. La classe utilizza una specializzazione dell'unità nascosta che separa le due componenti durante il training e la classificazione, tramite l'utilizzo di un vettore specifico di pesi solo per la stima dei vicini di ogni vertice e alcune modifiche alle funzioni di aggiornamento dei pesi e di calcolo della codifica.

Le classi `NN4inGonlyx` e `NN4inGonlyY` infine implementano i due modelli descritti nella sezione 3.5, quelli che si discostano maggiormente dalla struttura generale dei precedenti modelli. Questi due modelli infatti differiscono dai precedenti non

[1]Gli pseudocodici per le funzioni che calcolano il gradiente della funzione di correlazione e d'errore sono riportati in appendice B.

solo per la funzione di codifica o per il tipo di task, ma anche (soprattutto) a livello architetturale. Per questo motivo in queste classi vengono introdotte nuove strutture dati ad hoc per la nuova architettura, insieme con l'introduzione di uno specifico tipo di unità nascosta, a sua volta sottoclasse dell'hidden unit utilizzata dai precedenti modelli. Oltre a ciò, anche per queste classi, viene specializzata la funzione che permette di calcolare l'incremento per l'update dei pesi delle hidden unit; in particolare, per queste architetture, non si considera più la correlazione con l'errore della rete, ma si effettua una classica discesa del gradiente.

A.1.1 Algoritmi d'apprendimento

Nel framework è possibile utilizzare quattro diversi algoritmi d'apprendimento per allenare le unità di output e nascoste della rete neurale.

Discesa del gradiente

É l'algoritmo classico per l'apprendimento all'interno di una rete neurale. Si basa sull'implementazione diretta della formula per l'aggiornamento dei pesi

$$w_{ij}^{(n)} = w_{ij}^{(n-1)} - \eta(\Delta w_{ij}^{(n)} - \lambda \cdot w_{ij}^{(n-1)}) + \alpha \cdot \Delta w_{ij}^{(n-1)} \tag{A.1}$$

dove η è il parametro che regola la velocità dell'apprendimento, $\Delta w_{ij}^{(n)}$ è il valore della derivata della funzione d'errore (o di correlazione, nel caso delle hidden units) calcolata dopo l'epoca n, rispetto alla variabile w_{ij}, λ è un parametro di regolarizzazione, e α è il parametro che regola il peso del momento, utilizzato per accelerare il processo d'apprendimento; per maggiori dettagli si rimanda ad esempio a [57].

Quick Propagation

L'algoritmo di Quick Propagation [58], [59] è un metodo del second'ordine che, tramite alcune euristiche, utilizza il metodo di Newton per velocizzare il raggiungimento del minimo della funzione d'errore (o il massimo della funzione di correlazione). L'algoritmo si basa su due assunzioni: che la curva dell'errore rispetto a ciascun peso sia approssimabile con una parabola, e che il cambiamento nel gradiente della

funzione d'errore rispetto ad ogni peso sia indipendente da tutti gli altri pesi della rete neurale.

Per ogni peso w_{ij} si mantiene la derivata dell'errore al passo precedente, $\Delta w_{ij}^{(n-1)}$, e il valore del precedente update per quel peso, $\delta w_{ij}^{(n-1)}$. Queste informazioni, insieme all'attuale valore del gradiente, vengono utilizzate per individuare la parabola che approssima localmente la funzione d'errore: il passo d'update dell'algoritmo consiste semplicemente nel raggiungere il minimo della parabola, tramite le formule

$$w_{ij}^{(n)} = w_{ij}^{(n-1)} + \delta w_{ij}^{(n)} \tag{A.2}$$

$$\delta w_{ij}^{(n)} = \frac{\Delta w_{ij}^{(n)}}{\Delta w_{ij}^{(n-1)} - \Delta w_{ij}^{(n)}} \cdot \delta w_{ij}^{(n-1)} \tag{A.3}$$

Se il gradiente attuale ha lo stesso segno del precedente, ed è più piccolo, il peso w_{ij} viene nuovamente spostato verso la stessa direzione; viceversa, se il segno cambia e il minimo è stato sorpassato, l'aggiornamento del peso avviene nella direzione opposta. La situazione in cui il gradiente corrente e il precedente hanno lo stesso segno, ma il primo è maggiore o uguale del secondo, può essere causa di instabilità dell'algoritmo: per evitare questa situazione, si utilizza una tecnica euristica che prevede l'introduzione di un parametro, μ, che regola la grandezza del passo di update: $\delta w_{ij}^{(n)}$ non può essere maggiore di $\mu \cdot \delta w_{ij}^{(n-1)}$. Infine, per inizializzare il learning, si utilizza un passo di classica discesa del gradiente, con learning rate ϵ.

$$\delta w_{ij}^{(n)} = \begin{cases} \frac{\Delta w_{ij}^{(n)}}{\Delta w_{ij}^{(n-1)} - \Delta w_{ij}^{(n)}} \cdot \delta w_{ij}^{(n-1)} & \text{se} \quad \frac{\Delta w_{ij}^{(n)}}{\Delta w_{ij}^{(n-1)} - \Delta w_{ij}^{(n)}} \cdot \delta w_{ij}^{(n-1)} < \mu \cdot \delta w_{ij}^{(n-1)} \\ \mu \cdot \delta w_{ij}^{(n-1)} & \text{altrimenti} \end{cases} \tag{A.4}$$

Resilient Propagation

L'algoritmo di Resilient Propagation [60], [61], o RPROP, si differenzia dai precedenti in quanto si propone di eliminare l'influenza sull'apprendimento della magnitudine della derivata parziale $\Delta w_{ij}^{(n)}$; per fare ciò, per calcolare l'aggiornamento

del peso w_{ij} l'algoritmo di RPROP prende in considerazione solamente il segno della derivata parziale, e non il suo effettivo valore.

L'aggiornamento dei pesi effettuato dall'algoritmo di RPROP è descritto dalle seguenti formule

$$w_{ij}^{(n)} = w_{ij}^{(n-1)} + \delta_{ij}^{(n)} \tag{A.5}$$

$$\delta_{ij}^{(n)} = \begin{cases} -\Delta_{ij}^{(n)} & \text{se} \quad \Delta w_{ij}^{(n)} > 0 \\ +\Delta_{ij}^{(n)} & \text{se} \quad \Delta w_{ij}^{(n)} < 0 \\ 0 & \text{altrimenti} \end{cases} \tag{A.6}$$

$$\Delta_{ij}^{(n)} = \begin{cases} \eta^+ \cdot \Delta_{ij}^{(n-1)} & \text{se} \quad \Delta w_{ij}^{(n-1)} \cdot \Delta w_{ij}^{(n)} > 0 \\ \eta^- \cdot \Delta_{ij}^{(n-1)} & \text{se} \quad \Delta w_{ij}^{(n-1)} \cdot \Delta w_{ij}^{(n)} < 0 \\ \Delta_{ij}^{(n-1)} & \text{altrimenti} \end{cases} \tag{A.7}$$

con $0 < \eta^- < 1 < \eta^+$. Il segno della derivata $\Delta w_{ij}^{(n)}$ determina se aumentare o decrementare il valore dell'update $\Delta_{ij}^{(n)}$: se il segno è cambiato rispetto alla precedente iterazione, il minimo locale della funzione d'errore è stato superato, e quindi $\Delta_{ij}^{(n)}$ viene decrementato moltiplicandolo per η^-; viceversa, se il segno rimane costante, per accelerare la convergenza, $\Delta_{ij}^{(n)}$ viene aumentato moltiplicandolo per η^+.

Pseudoinversa e Ridge Regression

La pseudo inversa di Moore-Penrose permette di calcolare direttamente il valore ottimale dei pesi $\mathbf{w}$ per l'unità di una rete neurale. In generale il gradiente della funzione d'errore quadratica $E_{tot} = ||\mathbf{t} - \mathbf{X}\mathbf{w}||_2$ rispetto all'o-esimo neurone di output può essere scritto come

$$\frac{\partial E_{tot}^{(o)}}{\partial w_{oi}} = -2 \sum_{k=1}^{K} x_{ki} \cdot (t_k - \sum_{j=0}^{N} w_{oj} x_{kj}) = 0 \tag{A.8}$$

Ponendo a zero il gradiente per trovare il minimo della funzione d'errore, si ottiene che, per tutte le unità di output, il minimo della funzione equivale alla soluzione del sistema di equazioni normali $\mathbf{X}^T\mathbf{t} = \mathbf{X}^T\mathbf{X}\mathbf{w}$, dove la matrice $\mathbf{X}$ contiene l'input del neurone (ad esempio, nel caso dell'unità di output di NN4G, la codifica di tutti i grafi del dataset), il vettore $\mathbf{t}$ il target e il vettore $\mathbf{w}$ i pesi dell'unità. Tramite la pseudoinversa di Moore–Penrose si può calcolare immediatamente questa soluzione come

$$\mathbf{w}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{t} = \mathbf{X}^+\mathbf{t} \tag{A.9}$$

Utilizzando la Ridge Regression invece, è possibile introdurre un meccanismo di regolarizzazione nel calcolo di $\mathbf{w}^*$, ponendo

$$\mathbf{w}^* = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}_N)^{-1}\mathbf{X}^T\mathbf{t} \tag{A.10}$$

con λ parametro di regolarizzazione.

A.2 Formato di input del simulatore

Il formato di input del simulatore è stato adattato dal formato gph con cui è codificato il dataset sugli alcani (4.1.1). Questo formato permette di descrivere tutta la struttura del grafo o dei grafi che compongono il dataset in un unico file.

Nel listato A.1 è riportato un esempio di file gph per un dataset per l'on-graph learning, composto da più grafi, con task di trasduzione scalare, mentre nel listato A.2 un esempio di file per un task di in-graph learning, contenente un unico grafo.

Il significato dei diversi campi del file gph è il seguente:

- `SymbolNum`: indica il numero delle righe della successiva symbol table.
- `LabelDim`: indica la lunghezza delle etichette dei vertici.
- `TargetDim`: indica il numero di elementi del target.
- `IO`: un flag per distinguere il tipo di task del dataset: 1 se è un task di within-network learning o IO-trasduction, 0 se è di trasduzione scalare.

Listato A.1: Esempio di file gph input per il simulatore (task di trasduzione scalare)

```
SymbolNum 6
LabelDim 1
TargetDim 0
IO 0


 0  c            1
 1  ch           1
 2  ch2          1
 3  ch3          1
 4  ch3f         1
 5  ch4          1


GraphNum 135
MaxArity 4

Name 2
GraphDim 2
Target -0.886
 0   -1 -1 -1  1  ch3       3
 1   -1 -1  0 -1  ch3f      4

Name 3
GraphDim 3
Target -0.421
 0   -1 -1 -1  1  ch3       3
 1   -1 -1  0  2  ch2       2
 2   -1 -1  1 -1  ch3f      4

...
```

- `Symbol Table`: questa tabella contiene tutti i (diversi tipi di) vertici del dataset. Ogni riga contiene l'indice del vertice, il nome, la sua etichetta e, se la funzione di trasduzione è tipo IO, il target associato. La tabella non deve obbligatoriamente contenere tutti i vertici del grafo: se un vertice ad esempio compare più volte nel dataset, con la stessa etichetta, questo può essere indicato un'unica volta nella symbol table. É quanto accade nel caso dei dataset chimici (come PTC e Alcani), in cui ogni riga contiene un diverso elemento chimico.

- `GraphNum`: indica il numero di grafi contenuti nel file.

- `MaxArity`: indica il massimo numero di vicini dei vertici del dataset.

- `Name`: nome identificativo del grafo.

- `GraphDim`: indica il numero di vertici del grafo.

- `Target`: indica il target associato al grafo (è significativo solo nel caso di trasduzione scalare).

- `Connection Table`: contiene una riga per ogni vertice del grafo, in cui sono riportati l'indice del vertice all'interno del grafo, l'elenco dei vertici cui è connesso, il nome del vertice e l'indice nella symbol table. Se gli archi del grafo sono pesati, il peso di ogni arco viene indicato affianco al vertice target, separandolo con una virgola. Se il grafo è posizionale, si riportano per ogni vertice tutti i suoi possibili vicini (che sono tanti quanto indicato dal campo `MaxArity`), indicando con -1 il fatto che un dato vicino non esiste (listato A.1); se invece il grafo non è posizionale, si riportano solamente i vicini del vertice esistenti, terminando la lista con il simbolo speciale `-1,0` (listato A.2).

Nel caso di task di on-graph learning, essendo i dataset composti da un insieme di più grafi, è possibile scomporli in più file .gph se necessario, ad esempio per suddividere i dati in training e test set, o per creare i fold per una cross validation. Nel caso dei dataset del within-network learning invece, essendo presente un unico grafo, è stata adottata la soluzione di indicare in un unico file .gph l'intera rete, e poi utilizzare due file separati per indicare al simulatore i vertici di training e i

Listato A.2: Esempio di file gph input per il simulatore (task di within-network learning)

```
SymbolNum 9072
LabelDim 50
TargetDim 1
IO 1

 0  9   0.000946 ... 0.000946       1
 1  11  0.001937 ... 0.000554       -1
.
.
.
 9071  9072     0.002461 ... 0.000820       -1


GraphNum 1
MaxArity 3909

Name webspam
GraphDim 9072
Target 1
 0   0,1 1347,1 1459,1 1825,1 2250,1 2625,1 3017,1 6213,1 6245,1 6275,1 6684,1
     6807,1 662,1 8332,1 36,1 1571,1 2144,1 2387,1 3854,1 4982,1 479,1 5367,1
     6639,1 -1,0 9       0
 1   1,1 2869,1 3915,1 4527,1 5193,1 6639,1 8602,1 -1,0 11          1
 2   2,1 3641,1 8286,1 4365,1 -1,0 12       2
 ...
```

vertici di test rispettivamente; questi file contengono semplicemente una lista dei nomi dei vertici appartenenti a ciascun insieme.

A.3 Output del simulatore

Al termine dell'allenamento, il simulatore produce i seguenti file di output :

- `training_accuracy` e `test_accuracy` : contengono, per ogni epoca, l'accuracy del simulatore su training e test set rispettivamente.

- `training_mse` e `test_mse` : contengono, per ogni epoca, il mean squared error del simulatore su training e test set rispettivamente.

- `training_avg` e `test_avg` : contengono, per ogni epoca, l'errore medio in valore assoluto del simulatore su training e test set rispettivamente.

- `training_max` e `test_max` : contengono, per ogni epoca, l'errore massimo del simulatore su training e test set rispettivamente.

- `training_output` e `test_output` : contengono l'output del simulatore per ogni elemento di training e test set rispettivamente.

In base al valore dell'opzione `-nprint` (vedi manuale sez. A.4), durante l'allenamento il simulatore produce i seguenti file intermedi:

- `net`: contiene il valore dei pesi della rete neurale dopo l'inserimento e allenamento dell'ultima unità nascosta.

- `training_output` e `test_output` : contengono l'output del simulatore per ogni elemento di training e test set dopo l'inserimento e allenamento dell'ultima unità nascosta.

- `training_encoding` e `test_encoding` : contengono l'encoding delle ultime unità nascoste aggiunte per ogni elemento di training e test set.

A.4 Manuale d'uso

Per utilizzare il software, da riga di comando eseguire:

```
./NN4inG [options] datasetfile
```

Opzioni:

`-h`: visualizza un messaggio d'aiuto contenente un elenco di tutte le opzioni del programma.

`-net (0|1|2|3|4|5)`: specifica il tipo di rete neurale da creare, rispettivamente rete per task di trasduzione scalare, on-graph IO o in-graph learning, rete con architettura modificata completamente connessa o semplificata e rete per l'in-graph learning con distinzione tra stima e target dei vicini.

`-o outputfile`: utilizza il prefisso `outputfile` per produrre i file output dell'allenamento.

`-t testfile`: utilizza il file .gph `testfile` come test file (task di trasduzione scalare o IO on-graph).

`-trsplit file`: legge la lista dei vertici nel training set dal file indicato (task di in-graph learning).

`-tssplit file`: legge la lista dei vertici nel test set dal file indicato (task di in-graph learning).

`-nprint n`: stampa i file parziali ogni n epoche.

`-po (0|1|2|3)`: specifica l'algoritmo d'apprendimento per le output units, rispettivamente Backpropagation, Rprop, Quick Propagation e Pseudo Inversa.

`-ph (0|1|2)`: specifica l'algoritmo d'apprendimento per le hidden units, rispettivamente Backpropagation, Rprop, e Quick Propagation.

`-eo eta`: specifica il learning rate η per le output units (Backpropagation).

`-eh eta`: specifica il learning rate η per le hidden units (Backpropagation).

`-mo alpha`: specifica il momento α per le output units (Backpropagation).

`-mh alpha`: specifica il momento α per le hidden units (Backpropagation).

`-do lambda`: specifica il parametro di regolarizzazione λ per le output units.

`-dh lambda`: specifica il parametro di regolarizzazione λ per le hidden units.

`-eps e`: specifica il parametro ϵ dell'algoritmo di Quick Propagation.

`-mu m`: specifica il parametro μ dell'algoritmo di Quick Propagation.

`-oaf (0|1|2)`: specifica la funzione d'attivazione per le output units, rispettivamente lineare, sigmoide asimmetrica e sigmoide simmetrica.

`-haf (0|1|2)`: specifica la funzione d'attivazione per le hidden units, rispettivamente lineare, sigmoide asimmetrica e sigmoide simmetrica.

`-ko k`: specifica il parametro k per la funzione d'attivazione delle output units.

`-kh k`: specifica il parametro k per la funzione d'attivazione delle hidden units.

`-ooffset s`: specifica l'offset per la funzione d'attivazione delle output units.

`-hoffset s`: specifica l'offset per la funzione d'attivazione delle hidden units.

`-orange r`: specifica il range per la funzione d'attivazione delle output units.

`-hrange r`: specifica il range per la funzione d'attivazione delle hidden units.

`-norm (0|1|2)`: specifica la funzione $\mathbf{X}(\cdot)$ da utilizzare per la trasduzione scalare, rispettivamente somma, somma divisa per il massimo numero di vertici del dataset, e media.

`-et e`: specifica la soglia per l'errore da utilizzare come criterio per la fine dell'allenamento.

`-oct e`: specifica una soglia sul cambiamento dell'errore durante l'allenamento delle output units.

`-hct e`: specifica una soglia sul cambiamento dell'errore durante l'allenamento delle hidden units.

`-epochs n`: specifica il massimo numero di epoche di allenamento.

`-pool n`: specifica la dimensione del pool di hidden unit candidate.

`-wrange w`: specifica l'intervallo per l'inizializzazione dei pesi delle unità della rete come $[-w; w]$.

`-maxhidden n`: specifica il massimo numero di hidden units da aggiungere alla rete.

`-oi n`: specifica il numero di epoche iniziali di allenamento per le output units.

`-hi n`: specifica il numero di epoche iniziali di allenamento per le hidden units.

`-oincr n`: specifica di quanto incrementare il numero di epoche di allenamento per le output units all'inserimento di ogni nuova hidden unit.

`-hincr n`: specifica di quanto incrementare il numero di epoche di allenamento per le hidden units all'inserimento di ogni nuova hidden unit.

`-closed`: specifica di utilizzare lo stato del vertice stesso (e la precedente stima, se presente anche l'opzione `-estimation`), calcolato dalle precedenti unità, nel calcolarne la nuova codifica.

`-ntarget`: specifica di utilizzare il target dei vicini (noti) di un vertice per calcolarne lo stato (modello con target).

`-estimation`: specifica di utilizzare la stima della rete del target dei vicini per calcolare lo stato di un vertice (modello con stima).

`-onlyest`: specifica di non utilizzare il target dei vicini noti, ma di utilizzare solo la stima della rete del target dei vicini per calcolare lo stato di un vertice.

`-inout`: specifica di interpretare il grafo di input come un grafo diretto e di utilizzare il modello non stazionario; per fare ciò il file .gph che descrive il dataset deve utilizzare etichette diverse per gli archi entranti e uscenti da ogni vertice.

Appendice B

Complessità Computazionale

In questo capitolo si riporta la complessità computazionale delle più importanti operazioni che avvengono durante l'allenamento della rete neurale NN4inG, per tre dei modelli descritti nel capitolo 3: Neural Network for In-Graph con target e stima distinti (sez. B.1), Neural Network for In-Graph con solo hidden unit (sez. B.2), Neural Network for In-Graph con solo output unit (sez. B.3). Delle quattro varianti descritte nel capitolo 3 si riporta la complessità solo di NN4inG con target e stima distinti, in quanto è il modello più complesso tra quelli discussi, e la complessità degli altri modelli è facilmente derivabile da quella qui riportata.

Le operazioni considerate sono:

- la codifica di un vertice, ovvero il calcolo della componente i-esima $x_i(v)$;
- la stima del target di un vertice, ovvero il calcolo dell'output della rete $\mathbf{y}(v)$;
- il training di una unità nascosta; per calcolarne il costo si stima la complessità di un'epoca di allenamento di un'unità di questo tipo, calcolato come il costo del calcolo del gradiente della funzione di correlazione per i diversi pesi w dell'unità;
- il training di una unità output; il costo di questa operazione viene calcolato come il costo del calcolo del gradiente della funzione di errore della rete.

Nel seguito indichiamo con N il numero di unità nascoste della rete ($\mathbf{x}(v) \in \mathbb{R}^N$), con M la dimensione dell'output ($\mathbf{y}(v), \mathbf{t}(v) \in \mathbb{R}^M$), con L la dimensione della label

di input ($\mathbf{I}(v) \in \mathbb{R}^L$). Consideriamo un grafo $\mathbf{G}(V, E)$ con $|V|$ vertici e massimo numero di archi incidenti in un vertice D.

B.1 Neural Network for In-Graph con target e stima distinti

Codifica

La funzione di codifica di NN4inG con target e stima distinti (sez. 3.4) vale

$$\begin{aligned} x_i(v) = f\Bigg(& \sum_{l=0}^{L} \bar{w}_{il} I_l(v) + \sum_{j=1}^{i-1} \widehat{w}_{ij} \sum_{u \in \mathcal{N}(v)} e_{v,u} \cdot x_j(u) + \sum_{j=1}^{i-1} \widehat{w}_{ij}^{(s)} x_j(v) + \\ & + \sum_{m=1}^{M} \widetilde{w}_{im}^{(s)} y_m^{(i-1)}(v) + \sum_{m=1}^{M} \widetilde{w}_{im}^{(t)} \sum_{u \in \mathcal{N}(v) \cap V_k} e_{v,u} \cdot t_m(u) + \\ & + \sum_{m=1}^{M} \widetilde{w}_{im}^{(e)} \sum_{u \in \mathcal{N}(v) \cap V_u} e_{v,u} \cdot y_m^{(i-1)}(u) \Bigg) \end{aligned} \tag{B.1}$$

Nel calcolare la complessità del calcolo della codifica dell'unità i-esima, va tenuto conto del fatto che i valori $x_j(u) \forall u \in V, j < i$ non vengono ricalcolati ogni volta, ma possono essere memorizzati: una volta allenata e inserita nella rete infatti un'unità nascosta non modifica più i suoi pesi, e di conseguenza la componente della codifica relativa non cambia dopo l'inserimento di altre unità e il loro allenamento; un discorso analogo vale per la stima $\mathbf{y}^{(i-1)}(u)$.

La complessità del calcolo della codifica di un vertice v del grafo è quindi

$$\begin{aligned} C_{enc} &\approx \mathcal{O}(L + (i-1)D + (i-1) + M + 2MD) \\ &\approx \mathcal{O}(L + iD + MD) \\ &\approx \mathcal{O}(L + ND + MD) \end{aligned} \tag{B.2}$$

Output

L'output della rete neurale per un vertice v è dato da

$$y_m^{(N)}(v) = f\left(\sum_{j=0}^{N} w_{mj} \cdot x_j(v)\right) \tag{B.3}$$

per $m = 1, \cdots, M$.

La complessità di questa operazione è quindi pari a

$$C_{out} \approx \mathcal{O}(MN) \tag{B.4}$$

Training unità di codifica

Il processo di training di un unità di codifica $x_i(\cdot)$ di questo modello consiste nel calcolare il gradiente della funzione di correlazione S_i (eq. 3.5) dell'output dell'unità rispetto all'errore commesso dalla rete, rispetto a tutti i pesi $\bar{w}, \widehat{w}, \widetilde{w}$ che compaiono nell'equazione B.1. I gradienti sono pari a

$$\Delta\widehat{w}_{ij} = \sum_{o=1}^{M} \sigma_o \sum_{v \in V_k} (E_o(v) - \bar{E}_o) f'(net_i) \sum_{u \in \mathcal{N}(v)} e_{v,u} \cdot x_j(u) \tag{B.5}$$

$$\Delta\widehat{w}_{ij}^{(s)} = \sum_{o=1}^{M} \sigma_o \sum_{v \in V_k} (E_o(v) - \bar{E}_o) f'(net_i) x_j(v) \tag{B.6}$$

$$\Delta\bar{w}_{il} = \sum_{o=1}^{M} \sigma_o \sum_{v \in V_k} (E_o(v) - \bar{E}_o) f'(net_i)\, I_l(v) \tag{B.7}$$

$$\Delta\widetilde{w}_{ij}^{(t)} = \sum_{o=1}^{M} \sigma_o \sum_{v \in V_k} (E_o(v) - \bar{E}_o) f'(net_i) \sum_{u \in \mathcal{N}(v) \cap V_k} e_{v,u} \cdot t_j(u) \tag{B.8}$$

$$\Delta\widetilde{w}_{ij}^{(e)} = \sum_{o=1}^{M} \sigma_o \sum_{v \in V_k} (E_o(v) - \bar{E}_o) f'(net_i) \sum_{u \in \mathcal{N}(v) \cap V_u} e_{v,u} \cdot y_j^{(i-1)}(u) \tag{B.9}$$

$$\Delta\widetilde{w}_{ij}^{(s)} = \sum_{o=1}^{M} \sigma_o \sum_{v \in V_k} (E_o(v) - \bar{E}_o) f'(net_i) y_j^{(i-1)}(v) \tag{B.10}$$

con

$$\sigma_o = sign\left(\sum_{v \in V_k} \left(E_o(v) - \bar{E}_o\right)\left(x_i(v) - \bar{x}_i\right)\right) \tag{B.11}$$

Nell'implementazione della rete si sono utilizzate alcune accortezze per ridurre il costo di questa fase. I valori di $E_o(v)$, $\bar{E}_o$ rappresentano l'errore dell'unità di output o-esima sul vertice v e la media su tutti i vertici del grafo di questo errore rispettivamente; vengono calcolati durante la memorizzazione dell'output della rete prima dell'inizio dell'allenamento della nuova unità, e poi utilizzati per ogni sua epoca di allenamento; il loro costo è di $C_{storeout} \approx \mathcal{O}(|V|C_{out})$. Nell'algoritmo 2 si riporta lo pseudo codice utilizzato per determinare il valore dei gradienti; il calcolo è stato ottimizzato al fine di ridurne il più possibile il costo. Sulla base dell'algoritmo 2, la complessità di un'epoca di training di un'unità nascosta vale

$$\begin{aligned} C_{tr\_enc} &\approx \mathcal{O}\big(|V|(C_{enc} + iD + MD + M(L + M + i)) + \\ &\quad + M|V| + M(L + M + i)\big) \\ &\approx \mathcal{O}\big(|V|(C_{enc} + iD + MD + M(L + M + i)) + M|V|\big) \\ &\approx \mathcal{O}\big(|V|(C_{enc} + iD + MD + ML + M^2 + iM))\big) \\ &\approx \mathcal{O}\big(|V|(L + ND + MD + ML + M^2 + NM))\big) \end{aligned} \tag{B.12}$$

Training unità di output

Il training di ogni unità di output avviene per discesa del gradiente, e come per le unità nascoste il costo maggiore di questa fase è dato dal calcolo del gradiente Δw_{mi}, che vale

$$\Delta w_{mi} = \sum_{v \in V_k} \left(y_m(v) - t_m(v)\right) f'(net_m) x_i(v) \tag{B.13}$$

L'algoritmo 3 mostra lo pseudo codice con cui viene calcolato il valore del gradiente per ogni output unit e per ogni sua connessione con le unità nascoste della rete; tenendo conto del fatto che i valori $x_i(v)$ sono memorizzati, e non devono essere ricalcolati, la complessità di questa operazione vale

Algoritmo 2 Pseudo codice per il calcolo del gradiente della funzione di correlazione rispetto a tutti i pesi w dell'unità $x_i(\cdot)$.

for all $v \in V_k$ **do**
  $x_i[v] \leftarrow x_i(v)$
  $\bar{x}_i \leftarrow \bar{x}_i + x_i[v]$
  $dev(x_i) \leftarrow f'(net_i(v))$
  for $j = 1, \cdots, i-1$ **do**
    $enc\_sum[j] \leftarrow \sum_{u \in \mathcal{N}(v)} e_{v,u} \cdot x_j(u)$
    $enc\_self\_sum[j] \leftarrow x_j(v)$
  end for
  for $m = 1, \cdots, M$ **do**
    $target\_sum[m] \leftarrow \sum_{u \in \mathcal{N}(v) \cap V_k} e_{v,u} \cdot t_m(u)$
    $est\_sum[j] \leftarrow \sum_{u \in \mathcal{N}(v) \cap V_u} e_{v,u} \cdot y_m^{(i-1)}(u)$
    $est\_self\_sum[j] \leftarrow y_m^{(i-1)}(v)$
  end for
  for $m = 1, \cdots, M$ **do**
    for $j = 1, \cdots, i-1$ **do**
      $encoding\_acc[m][j] \mathrel{+}= (E_o(v) - \bar{E}_o) \cdot dev(x_i) \cdot enc\_sum[j]$
      $encoding\_self\_acc[m][j] \mathrel{+}= (E_o(v) - \bar{E}_o) \cdot dev(x_i) \cdot enc\_self\_sum[j]$
    end for
    for $o = 1, \cdots, M$ **do**
      $target\_acc[m][l] \mathrel{+}= (E_o(v) - \bar{E}_o) \cdot dev(x_i) \cdot target\_sum[o]$
      $est\_acc[m][l] \mathrel{+}= (E_o(v) - \bar{E}_o) \cdot dev(x_i) \cdot est\_sum[o]$
      $est\_self\_acc[m][l] \mathrel{+}= (E_o(v) - \bar{E}_o) \cdot dev(x_i) \cdot est\_self\_sum[o]$
    end for
    for $l = 1, \cdots, L$ **do**
      $input\_acc[m][l] \mathrel{+}= (E_o(v) - \bar{E}_o) \cdot dev(x_i) \cdot I_l(v)$
    end for
  end for
end for
for all $v \in V_k$ **do**
  for $m = 1, \cdots, M$ **do**
    $\sigma[m] \leftarrow \left(E_o(v) - \bar{E}_o\right)(x_i[v] - \bar{x}_i)$
  end for
end for
for $j = 1, \cdots, M$ **do**
  for $j = 1, \cdots, i-1$ **do**
    $\Delta\widehat{w}_{ij} \leftarrow sign(\sigma[m]) \cdot encoding\_acc[m][j]$
    $\Delta\widehat{w}_{ij}^{(s)} \leftarrow sign(\sigma[m]) \cdot encoding\_self\_acc[m][j]$
  end for
  for $j = o, \cdots, M$ **do**
    $\Delta\widetilde{w}_{io}^{(t)} \leftarrow sign(\sigma[m]) \cdot target\_acc[m][o]$
    $\Delta\widetilde{w}_{io}^{(e)} \leftarrow sign(\sigma[m]) \cdot est\_acc[m][o]$
    $\Delta\widetilde{w}_{io}^{(s)} \leftarrow sign(\sigma[m]) \cdot est\_self\_acc[m][o]$
  end for
  for $l = 1, \cdots, L$ **do**
    $\Delta\bar{w}_{il} \leftarrow sign(\sigma[m]) \cdot input\_acc[m][l]$
  end for
end for

Algoritmo 3 Pseudo codice per il calcolo del gradiente della funzione di errore per le output unit $\mathbf{y}(\cdot)$.

```
for all v ∈ V_k do
    for m = 1, ··· , M do
        d ← (y_m(v) − t_m(v))
        dev ← f'(net_m(v))
        for i = 1, ··· , N do
            Δw_mi ← d · dev · x_i(v)
        end for
    end for
end for
```

$$\begin{aligned} C_{tr\_out} &\approx \mathcal{O}\big(|V|M(C_{out}+N\big) \\ &\approx \mathcal{O}\big(|V|M(MN+N\big) \\ &\approx \mathcal{O}\big(|V|M^2N\big) \end{aligned} \tag{B.14}$$

B.2 Neural Network for In-Graph con solo hidden unit

In questa variante architetturale esiste un unico tipo di unità nella rete, l'unità $x_i^{(m)}(\cdot)$ (sez. 3.5, prima variante), e quindi nel seguito si discute il costo delle operazioni di output e training per quest'unico tipo di unità.

Output

La funzione di output calcolata dalla rete vale

$$x_i^{(m)}(v) = f\Bigg(\sum_{l=0}^{L} \bar{w}_{il} I_l(v) + \sum_{j=1}^{i-1}\sum_{k=1}^{M} \widehat{w}_{ijk} \sum_{u\in\mathcal{N}(v)} e_{v,u} x_j^{(k)}(u) + \sum_{j=1}^{i-1}\sum_{k=1}^{M} \widehat{w}_{ijk}^{(s)}\, x_j^{(k)}(v) + \\ + \sum_{k=1}^{M} \widetilde{w}_{ik} \sum_{u\in\mathcal{N}(v)\cap V_k} e_{v,u} t_k(u)\Bigg) \tag{B.15}$$

per $m = 1, \cdots, M$.

La complessità dell'operazione di output per l'm-esima unità vale

$$\begin{aligned} C_{out} &\approx \mathcal{O}(L + (i-1)MD + MD) \\ &\approx \mathcal{O}(L + NMD) \end{aligned} \tag{B.16}$$

in quanto come osservato anche per la codifica nella precedente sezione, l'output dei precedenti strati di unità aggiunte alla rete $\mathbf{x}^{(k)}(v)$ viene congelato dopo l'allenamento, e non deve perciò essere ricalcolato di volta in volta. Per tutte le unità dello strato di output, la complessità vale $\mathcal{O}(ML + NM^2D)$).

Training unità di output

L'allenamento delle unità di output di questa rete avviene tramite discesa del gradiente. I gradienti da calcolare sono relativi ai pesi che compaiono nell'equazione B.15 e per l'unità $x_i^{(m)}(\cdot)$ valgono

$$\Delta\bar{w}_{il} = \sum_{v\in V_k}\Big(x_i^{(m)}(v) - t_m(v)\Big)\, f'(net_i^m)\, I_l(v) \tag{B.17}$$

$$\Delta\widehat{w}_{ijk} = \sum_{v\in V_k}\Big(x_i^{(m)}(v) - t_m(v)\Big)\, f'(net_i^m) \sum_{u\in\mathcal{N}(v)} e_{v,u} x_j^{(k)}(u) \tag{B.18}$$

$$\Delta\widehat{w}_{ijk}^{(s)} = \sum_{v\in V_k}\Big(x_i^{(m)}(v) - t_m(v)\Big)\, f'(net_i^m)\, x_j^{(k)}(v) \tag{B.19}$$

$$\Delta\widetilde{w}_{ik} = \sum_{v\in V_k}\Big(x_i^{(m)}(v) - t_m(v)\Big)\, f'(net_i^m) \sum_{u\in\mathcal{N}(v)\cap V_k} e_{v,u} t_k(u) \tag{B.20}$$

Algoritmo 4 Pseudo codice per il calcolo del gradiente della funzione di errore per l'output unit $x_i^{(m)}(\cdot)$ del modello con solo hidden unit.

for all $v \in V_k$ **do**
  for $m = 1, \cdots, M$ **do**
    $d \leftarrow (x_i^{(m)}(v) - t_m(v))$
    $dev \leftarrow f'(net_i^m(v))$
    for $k = 1, \cdots, M$ **do**
      for $j = 1, \cdots, N$ **do**
        $\Delta\widehat{w}_{ijk} \leftarrow d \cdot dev \cdot \sum_{u\in\mathcal{N}(v)} e_{v,u} x_j^{(k)}(u)$
        $\Delta\widehat{w}_{ijk}^{(s)} \leftarrow d \cdot dev \cdot x_j^{(k)}(v)$
      end for
    end for
    for $l = 1, \cdots, L$ **do**
      $\Delta\bar{w}_{il} \leftarrow d \cdot dev \cdot I_l(v)$
    end for
    for $k = 1, \cdots, M$ **do**
      $\Delta\widetilde{w}_{ik} \leftarrow d \cdot dev \cdot \sum_{u\in\mathcal{N}(v)\cap V_k} e_{v,u} t_k(u)$
    end for
  end for
end for

La complessità dell'algoritmo 4 per il calcolo del gradiente per tutte le unità $x_i^{(m)}(\cdot)$ per $m = 1, \cdots, M$ dello strato di output vale

$$\begin{aligned} C_{tr\_out} &\approx \mathcal{O}\big(M|V|(C_{out} + MND + L + MD)\big) \\ &\approx \mathcal{O}\big(M|V|(L + MND + MD)\big) \\ &\approx \mathcal{O}\big(|V|(ML + M^2ND)\big) \end{aligned} \tag{B.21}$$

B.3 Neural Network for In-Graph con solo output unit

Analogamente a quanto detto nella precedente sezione per il modello con solo hidden unit, essendo anche in questo modello presente un solo tipo d'unità $y_m^{(i)}(v)$ (sez. 3.5, seconda variante), si discute solamente il costo delle operazioni di output e training di questa unità.

Output

L'output dell'unità $y_m^{(i)}(\cdot)$ è dato da

$$\begin{aligned} y_m^{(i)}(v) = f\Bigg(\sum_{l=0}^{L} \bar{w}_{il} I_l(v) + \sum_{k=1}^{M} \widetilde{w}_{ik}^{(e)} \sum_{u \in \mathcal{N}(v)} e_{v,u} y_k^{(i-1)}(u) + \sum_{k=1}^{M} \widetilde{w}_{ik}^{(s)}\, y_k^{(i-1)}(v) + \\ + \sum_{k=1}^{M} \widetilde{w}_{ik}^{(t)} \sum_{u \in \mathcal{N}(v)} e_{v,u} t_k(u)\Bigg) \end{aligned} \tag{B.22}$$

per $m = 1, \cdots, M$.

La complessità dell'operazione di output per l'm-esima unità vale quindi

$$C_{out} \approx \mathcal{O}(L + MD) \tag{B.23}$$

Per tutte le unità dello strato di output, la complessità vale $\mathcal{O}(ML + M^2D)$.

Training unità di output

L'algoritmo 5 di training delle unità di output per questo modello è analogo a quello del precedente. L'allenamento del nuovo strato di unità avviene sempre per discesa del gradiente, e i gradienti che devono essere calcolati sono

$$\Delta\bar{w}_{il} = \sum_{v\in V_k} \big(y_m^{(i)}(v) - t_m(v)\big)\, f'(net_i^m)\, I_l(v) \tag{B.24}$$

$$\Delta\widetilde{w}_{ik}^{(e)} = \sum_{v\in V_k} \big(y_m^{(i)}(v) - t_m(v)\big)\, f'(net_i^m) \sum_{u\in\mathcal{N}(v)\cap V} e_{v,u} y_k^{(i-1)}(u) \tag{B.25}$$

$$\Delta\widetilde{w}_{ik}^{(t)} = \sum_{v\in V_k} \big(y_m^{(i)}(v) - t_m(v)\big)\, f'(net_i^m) \sum_{u\in\mathcal{N}(v)\cap V_k} e_{v,u} t_k(u) \tag{B.26}$$

$$\Delta\widetilde{w}_{ik}^{(s)} = \sum_{v\in V_k} \big(y_m^{(i)}(v) - t_m(v)\big)\, f'(net_i^m) y_k^{(i-1)}(v) \tag{B.27}$$

Dall'analisi dell'algoritmo 5, la complessità di un'epoca di allenamento per tutte le M unità di output vale

$$\begin{aligned} C_{tr\_out} &\approx \mathcal{O}\big(M|V|(C_{out} + L + 2MD)\big) \\ &\approx \mathcal{O}\big(M|V|(L + MD + L + MD)\big) \\ &\approx \mathcal{O}\big(|V|(ML + M^2D)\big) \end{aligned} \tag{B.28}$$

Algoritmo 5 Pseudo codice per il calcolo del gradiente della funzione di errore per l'output unit $y_m^{(i)}(\cdot)$ del modello con solo output unit.

for all $v \in V_k$ **do**
  for $m = 1, \cdots, M$ **do**
    $d \leftarrow (y_m^{(i)}(v) - t_m(v))$
    $dev \leftarrow f'(net_i^m(v))$
    for $l = 1, \cdots, L$ **do**
      $\Delta\bar{w}_{il} \leftarrow d \cdot dev \cdot I_l(v)$
    end for
    for $k = 1, \cdots, M$ **do**
      $\Delta\widetilde{w}_{ik}^{(e)} \leftarrow d \cdot dev \cdot \sum_{u \in \mathcal{N}(v)} e_{v,u} y_k^{(i-1)}(u)$
      $\Delta\widetilde{w}_{ik}^{(t)} \leftarrow d \cdot dev \cdot \sum_{u \in \mathcal{N}(v) \cap V_k} e_{v,u} t_k(u)$
      $\Delta\widetilde{w}_{ik}^{(s)} \leftarrow d \cdot dev \cdot y_k^{(i-1)}(v)$
    end for
  end for
end for

Bibliografia

[1] B. Taskar, «Probabilistic classification and clustering in relational data», in *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, 2001, pp. 870–878.

[2] Q. Lu e L. Getoor, «Link-based classification», in *Proceedings of the International Conference on Machine Learning (ICML)*, 2003, pp. 496–503.

[3] A. McCallum, K. Nigam, J. Rennie e K. Seymore, «Automating the construction of internet portals with machine learning», *Information Retrieval*, vol. 3, pp. 127–163, 2000.

[4] J. Neville, D. Jensen e B. Gallagher, «Simple estimators for relational bayesian classifiers», in *Proceedings of the Third IEEE International Conference on Data Mining*, ser. ICDM '03, IEEE Computer Society, 2003, pp. 609–617.

[5] S. A. Macskassy e F. Provost, «Classification in networked data: a toolkit and a univariate case study», *Journal of Machine Learning Research*, n. 8, pp. 935–983, 2007.

[6] S. Chakrabarti, B. Dom e P. Indyk, «Enhanced hypertext categorization using hyperlinks», in *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '98, ACM, 1998, pp. 307–318.

[7] J. Neville, D. Jensen, L. Friedland e M. Hay, «Learning relational probability trees», in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003, pp. 625–630.

[8] J. Abernethy, O. Chapelle e C. Castillo, «Graph regularization methods for web spam detection», *Machine Learning*, n. 81, pp. 207–225, 2010.

[9] N. Spirin e J. Han, «Survey on web spam detection: principles and algorithms», *SIGKDD Explorations Newsletter*, vol. 13, n. 2, pp. 50–64, 2011.

[10] D. Jensen e J. Neville, «Data mining in social networks», in *Dynamic Social Network Modeling and Analysis: Workshop Summary and Papers*, 2003, pp. 289–302.

[11] S. A. Macskassy e F. Provost, «A simple relational classifier», in *Proceedings of the Second Workshop on Multi-Relational Data Mining (MRDM-2003) at KDD-2003*, 2003, pp. 64–76.

[12] P. Domingos e M. Richardson, «Mining the network value of customers», in *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '01, ACM, 2001, pp. 57–66.

[13] M. Richardson e P. Domingos, «Mining knowledge-sharing sites for viral marketing», in *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '02, ACM, 2002, pp. 61–70.

[14] T. Fawcett e F. Provost, «Adaptive fraud detection», *Data Mining and Knowledge Discovery*, vol. 1, n. 3, pp. 291–316, 1997.

[15] C. Cortes, D. Pregibon e C. Volinsky, «Communities of interest», in *Proceedings of the Fourth International Conference on Advances in Intelligent Data Analysis (IDA)*, 2001, pp. 105–114.

[16] S. A. Macskassy e F. Provost, «A brief survey of machine learning methods for classification in networked data and an application to suspicion scoring», in *Statistical Network Analysis: Models, Issues, and New Directions*, ser. Lecture Notes in Computer Science, vol. 4503, Springer Berlin Heidelberg, 2007, pp. 172–175.

[17] M. Collins e N. Duffy, «Convolution kernels for natural language», in *Advances in Neural Information Processing Systems 14*, The MIT Press, 2001, pp. 625–632.

[18] J. D. Lafferty, A. McCallum e F. C. N. Pereira, «Conditional random fields: probabilistic models for segmenting and labeling sequence data», in *Proceedings of the Eighteenth International Conference on Machine Learning*, ser. ICML '01, Morgan Kaufmann Publishers Inc., 2001, pp. 282–289.

[19] A. M. Bianucci, A. Micheli, A. Sperduti e A. Starita, «Application of cascade correlation networks for structures to chemistry», *Applied Intelligence Journal*, vol. 12, n. 1-2, pp. 117–146, gen. 2000.

[20] A. Micheli, A. Sperduti, A. Starita e A. M. Bianucci, «Analysis of the internal representations developed by neural networks for structures applied to quantitative structure-activity relationship studies of benzodiazepines», *Journal of Chemical Information and Computer Sciences*, vol. 41, n. 1, pp. 202–218, gen. 2001.

[21] H. Frohlich, J. Wegner e A. Zell, «Assignment kernels for chemical compounds», in *IEEE International Joint Conference on Neural Networks, 2005. (IJCNN '05)*, vol. 2, lug. 2005, pp. 913–918.

[22] P. Frasconi, M. Gori e A. Sperduti, «A general framework for adaptive processing of data structures», *IEEE Transactions on Neural Networks*, vol. 9, n. 5, pp. 768–786, set. 1998.

[23] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor e K. M. Borgwardt, «Graph kernels», *Journal of Machine Learning Research*, n. 11, pp. 1201–1242, 2010.

[24] T. Gärtner, «A survey of kernels for structured data», *SIGKDD Exploration Newsletter*, vol. 5, n. 1, pp. 49–58, 2003.

[25] H. Kashima, K. Tsuda e A. Inokuchi, «Marginalized kernels between labeled graphs», in *Proceedings of the Twentieth International Conference on Machine Learning*, AAAI Press, 2003, pp. 321–328.

[26] A. Sperduti e A. Starita, «Supervised neural networks for the classification of structures», *IEEE Transactions on Neural Networks*, vol. 8, pp. 714–735, 1997.

[27] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner e G. Monfardini, «The graph neural network model», *IEEE Transactions On Neural Networks*, vol. 20, n. 1, pp. 61–80, 2009.

[28] C. Gallicchio e A. Micheli, «Graph echo state networks», in *The 2010 International Joint Conference on Neural Networks (IJCNN)*, lug. 2010, pp. 2159–2166.

[29] A. Micheli, «Neural network for graphs: a contextual constructive approach», *IEEE Transactions On Neural Networks*, vol. 20, n. 3, pp. 498–511, 2009.

[30] W. Uwents, G. Monfardini, H. Blockeel, M. Gori e F. Scarselli, «Neural networks for relational learning: an experimental comparison», *Machine Learning*, vol. 82, n. 3, pp. 315–349, 2011.

[31] D. D. Monner e J. A. Reggia, «Recurrent neural collective classification», *IEEE Transactions on Neural Networks and Learning Systems*, vol. 24, n. 12, pp. 1932–1943, dic. 2013.

[32] H. Jaeger e H. Haas, «Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication», *Science Magazine*, vol. 304, n. 5667, pp. 78–80, 2004.

[33] S. E. Fahlman e C. Lebiere, «The cascade correlation learning architecture», Carnegie Mellon University, rapp. tecn. CMU-CS-90-100, 1989.

[34] L. Getoor e B. Taskar, *Introduction to Statistical Relational Learning*. The MIT Press, 2007.

[35] L. Getoor, N. Friedman, D. Koller, A. Pfeffer e B. Taskar, «Probabilistic relational models», in *Introduction to Statistical Relational Learning*, L. Getoor e B. Taskar, cur., The MIT Press, 2007, cap. 5.

[36] N. Friedman, L. Getoor, D. Koller e A. Pfeffer, «Learning probabilistic relational models», in *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, Springer-Verlag, 1999, pp. 1300–1309.

[37] D. Koller, «Probabilistic relational models», in *Proceedings of the 9th International Workshop on Inductive Logic Programming*, ser. ILP '99, Springer-Verlag, 1999, cap. 1, pp. 3–13.

[38] B. Taskar, P. Abbeel, M.-F. Wong e D. Koller, «Relational markov networks», in *Introduction to Statistical Relational Learning*, L. Getoor e B. Taskar, cur., The MIT Press, 2007, cap. 6.

[39] A. Blake e P. Kohli, «Introduction to markov random fields», in *Markov Random Fields for Vision and Image Processing*, A. Blake, P. Kohli e C. Rother, cur., The MIT Press, 2011, cap. 1.

[40] C. Desrosiers e G. Karypis, «Within-network classification using local structure similarity», in *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases: Part I*, ser. ECML PKDD '09, Springer-Verlag, 2009, pp. 260–275.

[41] R. A. Rossi, L. K. McDowell, D. W. Aha e J. Neville, «Transforming graph data for statistical relational learning», *Journal of Artificial Intelligence Research*, n. 45, pp. 363–441, 2012.

[42] A. Bianucci, A. Micheli, A. Sperduti e A. Starita, «Application of cascade correlation networks for structures to chemistry», *Applied Intelligence*, vol. 12, pp. 117–147, 2000.

[43] A. Micheli, «Recursive processing of structured domains in machine learning», Tesi di Dottorato, Dipartimento di Informatica, Università di Pisa, 2003.

[44] A. Micheli, D. Sona e A. Sperduti, «Contextual processing of structured data by recursive cascade correlation», *IEEE Transactions on Neural Networks*, vol. 15, n. 6, pp. 1396–1410, 2004.

[45] A. Micheli, F. Portera e A. Sperduti, «A preliminary empirical comparison of recursive neural networks and tree kernel methods on regression tasks for tree structured domains», *Neurocomputing*, vol. 64, pp. 73–92, 2005.

[46] A. Bianucci, A. Micheli, A. Sperduti e A. Starita, «A novel approach to QSPR/QSAR based on neural networks for structures», in *Soft Computing Approaches in Chemistry*, ser. Studies in Fuzziness and Soft Computing, vol. 120, 2003, pp. 265–296.

[47] C. Helma, R. D. King e S. Kramer, «The predictive toxicology challenge 2000-2001», *Bioinformatics*, vol. 17, pp. 107–108, 2001.

[48] M. Bastian, S. Heymann e M. Jacomy, «Gephi: an open source software for exploring and manipulating networks», in *International AAAI Conference on Weblogs and Social Media*, E. Adar, M. Hurst, T. Finin, N. S. Glance, N. Nicolov e B. L. Tseng, cur., The AAAI Press, 2009.

[49] S. A. Macskassy, *Netkit-srl: network learning toolkit for statistical relational learning*, `http://netkit-srl.sourceforge.net/index.html`, 2008–2013.

[50] X.-H. Phan e C.-T. Nguyen, *GibbsLDA++: a C/C++ implementation of latent dirichlet allocation (LDA)*, `http://gibbslda.sourceforge.net/`, 2007.

[51] D. M. Blei, A. Y. Ng, M. I. Jordan e J. Lafferty, «Latent dirichlet allocation», *Journal of Machine Learning Research*, vol. 3, pp. 993–1022, 2003.

[52] T. Fawcett, «An introduction to ROC analysis», *Pattern Recognition Letters*, vol. 27, n. 8, pp. 861–874, giu. 2006.

[53] Y. Tang, Y. He, S. Krasser e P. Judge, «Web Spam Challenge 2007 Track II secure computing corporation research», in *ECML/PKDD Graph Labelling Workshop – Web Spam Challenge*, 2007.

[54] P. Filoche, T. Urvoy e M. Boullè, «SpamChallenge 2007 - Track II: france telecom r&d submissions», in *ECML/PKDD Graph Labelling Workshop – Web Spam Challenge*, 2007.

[55] A. A. Benczúr, K. Csalogány, L. Lukács e D. Siklósi, «Semi-supervised learning: a comparative study for web spam and telephone user churn», in *ECML/PKDD Graph Labelling Workshop – Web Spam Challenge*, 2007.

[56] Y. Tian, G. M. Weiss e Q. Ma, «A semi-supervised approach for web spam detection using combinatorial feature-fusion», in *ECML/PKDD Graph Labelling Workshop – Web Spam Challenge*, 2007.

[57] S. Haykin, *Neural Networks and Learning Machines*. Pearson International Edition, 2009.

[58] M. Hoehfeld e S. E. Fahlman, «Learning with limited numerical precision using the cascade-correlation algorithm», *IEEE Transactions on Neural Networks*, vol. 3, pp. 602–611, 1992.

[59] S. E. Fahlman, «An empirical study of learning speed in back-propagation networks», rapp. tecn., 1988.

[60] M. Riedmiller, «Rprop - description and implementation details», University of Karlsruhe, rapp. tecn., 1994.

[61] M. Riedmiller e H. Braun, «A direct adaptive method for faster backpropagation learning: the RPROP algorithm», in *IEEE International Conference on Neural Networks*, vol. 1, 1993, pp. 586–591.