# Assignment: VI A - Political Blog Classification using Graph Convolutional Networks
# Programming report, analysis and discussion

Student ID: 592854

Deep Learning

## 1 Introduzione

In questo report verranno descritti tutti i passaggi intrapresi per la stesura del codice allegato alla consegna del progetto. Dalla scelta delle librerie alle strategie utilizzate per trattare i dati, dall'architettura scelta alla valutazione dei migliori iperparametri e per finire come sono state valutate le prestazioni del modello finale.

## 2 Librerie Utilizzate

All'interno del progetto sono state utilizzate le seguenti librerie:

- NetworkX
- PyTorch
- PyTorch Geometric
- scikit-learn

### 2.1 NetworkX

NetworkX è una libreria Python utilizzata per creare, manipolare e studiare reti complesse di nodi e archi. Essa semplifica il caricamento e la gestione del dataset Polblogs, che è strutturato come un grafo in cui i nodi rappresentano i blog e gli archi rappresentano i collegamenti ipertestuali tra di essi. Con NetworkX, posso visualizzare questo grafo a partire da un file in formato *.gml*, permettendo un'analisi esplorativa iniziale dei dati per comprendere la struttura e le proprietà della rete. La libreria facilita anche la creazione della matrice di adiacenza attraverso un metodo built-in. Utilizzando NetworkX, è stato possibile ottenere i dati dal file originale, rappresentare il grafo e la matrice di adiacenza e, successivamente, trasferire il grafo a Pytorch Geometric per l'utilizzo nel modello.

### 2.2 PyTorch

PyTorch è una libreria di deep learning popolare che fornisce una piattaforma flessibile ed efficiente per costruire e addestrare modelli di reti neurali. All'interno del progetto, PyTorch svolge un ruolo centrale nello sviluppo dell'architettura GCN. I moduli e le funzioni integrate, come gli ottimizzatori e le funzioni di perdita, semplificano il processo di addestramento, permettendo di implementare backpropagation e discesa del gradiente in modo efficiente. Inoltre, la capacità di PyTorch di interfacciarsi senza soluzione di continuità con PyTorch Geometric garantisce che le operazioni specializzate richieste per i dati del grafo vengano eseguite senza problemi.

### 2.3 PyTorch Geometric

PyTorch Geometric è una libreria estesa per PyTorch, specificamente progettata per gestire e processare dati strutturati a grafo. Fornisce strumenti e funzioni essenziali per implementare Graph Neural Networks (GNN), come la Graph Convolutional Network (GCN) utilizzata nel progetto. Convertendo il grafo NetworkX in un oggetto di dati PyTorch Geometric, viene sfruttata la gestione efficiente delle strutture dati sparse, comuni nei problemi basati su grafi. PyTorch Geometric semplifica l'implementazione dei layer GCN, permettendomi di definire operazioni convoluzionali che aggregano informazioni dai nodi vicini. Questa libreria facilita anche la conversione della matrice di adiacenza del grafo in un formato adeguato (COO) per il calcolo ottimizzato.

### 2.4 scikit-learn

Scikit-Learn è una versatile libreria di machine learning che fornisce strumenti semplici ed efficienti per l'analisi e la modellazione dei dati. Nel progetto, Scikit-Learn è utilizzata principalmente per suddividere i dati in set di addestramento, validazione e test, garantendo che la valutazione del modello sia robusta e affidabile. Sfruttando funzioni come *train_test_split*, è stato possibile partizionare sistematicamente i dati, permettendo la valutazione delle performance del modello su dati non visti. Inoltre, il *ParameterGrid* di scikit-Learn è stato utilizzato per eseguire la ricerca degli iperparametri, consentendo l'esplorazione sistematica di varie configurazioni del modello per identificare i parametri con le migliori performance. Questo approccio di ricerca a griglia aiuta a ottimizzare l'architettura della GCN valutando diverse combinazioni di iperparametri.

## 3 Preprocessing dei dati

Per prima cosa è stato scaricato il dataset, dal link fornito all'interno della traccia, ed è stato estratto il contenuto all'interno della cartella polblogs. Il file *.zip* contiene al suo interno un file *.txt*, che descrive il dataset, e un file *.gml*, che è il vero e proprio dataset. Il grafo che è stato fornito è un grafo diretto con: *1490* nodi (blog) e *19025* edge (collegamenti tra blog). Per poterlo utilizzare in Python è stata utilizzata la libreria NetworkX, che inizialmente ritornava un errore riguardo alcuni collegamenti duplicati. Per risolvere questo problema il grafo è stato portato da un grafo diretto ad un multigrafo diretto (file *polblogs1.gml*) aggiungendo l'attributo *multigraph = 1* nel file originale. È stato valutato l'utilizzo di un metodo che rimuovesse i collegamenti duplicati tra due nodi, ma non è stata intrapresa questa strada per due motivi: complessità di implementazione più elevata rispetto ad aggiungere un semplice attributo e possibilità di riduzione di informazione eliminando i collegamenti duplicati. Infatti, se consideriamo il caso in esame, un blog potrebbe contenere più di un collegamento verso un altro blog e questa informazione potrebbe essere utile ai fini della classificazione finale. Successivamente al caricamento del dataset viene mostrata la struttura del nodo e dell'edge, viene visualizzata la matrice di adiacenza e la struttura del grafo. Il passo successivo è stato quello di trasformare il grafo NetworkX in un grafo PyTorch Geometric. Questo passaggio permette una facilità di utilizzo dei dati per le fasi successive. Per quanto riguarda le feature dei nodi, l'attributo *source* non viene utilizzato e l'attributo *value* è il nostro target, quindi è stato utilizzato un one-hot-encoding attraverso una matrice identità. L'attributo *source* è stato scartato perché non sembra contenere informazioni utili alla classificazione. Per questo motivo è stata creata una matrice identità con one-hot encoding per rappresentare i nodi, trattandoli come unici e distinti dagli altri. La variabile target è, invece, rappresentata dai valori dell'attributo *value*. Successivamente viene suddiviso il dataset in train, test e validation. Abbiamo una suddivisione tale: 20% dei nodi sono nel test set e nell'80% dei nodi rimanenti, il 20% dei nodi sono nel validation set e i restanti 80% dei nodi nel train set. (Table 1)

Table 1: Suddivisione del Dataset

| Train and Validation set (80%) | Test set (20%) |
|---|---|

| Train set (80%) | Val. set (20%) |
|---|---|

## 4 Architettura del Modello

L'architettura del modello GCN utilizzata è composta da tre strati di convoluzione del grafo, con dropout tra gli strati per migliorare la generalizzazione e prevenire l'overfitting.
I dettagli dell'architettura sono i seguenti:

1. **Input Layer**: L'input layer comprende il numero di caratteristiche in input per ciascun nodo nel grafo (*in_channels*) e il primo strato di convoluzione (*conv1*). Questo layer applica una convoluzione del grafo utilizzando il modulo *GCNConv* di PyTorch Geometric, trasformando le caratteristiche dei nodi dall'input a un numero di canali specificato nel primo elemento di *hidden_channels*;

2. **Hidden Layers**: Gli hidden layers includono il secondo strato di convoluzione (*conv2*) che applica una seconda convoluzione del grafo, trasformando le caratteristiche dei nodi dal primo al secondo livello di hidden layers. Successivamente, il terzo strato di convoluzione (*conv3*) applica una terza convoluzione del grafo, trasformando le caratteristiche dei nodi dal secondo livello di hidden layers al numero di canali di output (*out_channels*);

3. **Dropout**: Il modello utilizza anche il dropout, applicato dopo ciascun strato di convoluzione con una probabilità del 50% per prevenire l'overfitting. Il dropout è una tecnica di regolarizzazione che disattiva casualmente una frazione delle unità del modello durante l'addestramento;

4. **Fuzioni di attivazione**: Sono utilizzate due funzioni di attivazione: ReLU (Rectified Linear Unit), utilizzata come funzione di attivazione non lineare dopo le prime due convoluzioni del grafo, e log softmax, applicata all'output dell'ultimo strato per ottenere le probabilità logaritmiche delle classi di output.

Dopo il primo strato di convoluzione (*conv1*), ogni nodo nel grafo aggiorna la sua rappresentazione aggregando le informazioni dalle rappresentazioni dei suoi nodi vicini. Questo significa che il nodo $i$ raccoglie informazioni dai nodi $j$ che sono collegati a $i$ attraverso un edge. Matematicamente, questo può essere descritto come:

$$x_i^{(1)} = ReLU\left( \Sigma_{j \in N(i)} \frac{1}{\sqrt{d_i d_j}} W^{(1)} x_j + \frac{1}{\sqrt{d_i}} W^{(1)} x_i \right)$$

dove *N(i)* è l'insieme dei vicini del nodo $i$, $d_i$ e $d_j$ sono i gradi dei nodi $i$ e $j$, e *W(1)* è la matrice dei pesi del primo strato. Il secondo strato di convoluzione (*conv2*) continua il processo di message passing aggiornando ulteriormente le rappresentazioni dei nodi basandosi sulle rappresentazioni aggiornate ottenute dal primo strato. In questo caso, ogni nodo i aggrega le informazioni dai suoi vicini $j$ con le nuove rappresentazioni:

$$x_i^{(2)} = ReLU\left( \Sigma_{j \in N(i)} \frac{1}{\sqrt{d_i d_j}} W^{(2)} {x_j}^{(1)} + \frac{1}{\sqrt{d_i}} W^{(2)} {x_j}^{(1)} \right)$$

dove ${x_j}^{(1)}$ è la rappresentazione del nodo $j$ dopo il primo strato di convoluzione. Il terzo strato di convoluzione (*conv3*) effettua l'ultima aggregazione di informazioni dai nodi vicini. Le rappresentazioni dei nodi sono ulteriormente aggiornate utilizzando la stessa logica dei passaggi precedenti:

$$x_i^{(3)} = \sigma\left( \Sigma_{j \in N(i)} \frac{1}{\sqrt{d_i d_j}} W^{(3)} {x_j}^{(2)} + \frac{1}{\sqrt{d_i}} W^{(3)} {x_j}^{(2)} \right)$$

dove ${x_j}^{(2)}$ è la rappresentazione del nodo $j$ dopo il secondo strato di convoluzione e $\sigma$ è la funzione di attivazione Log Softmax. Quindi, il message passing avviene tra ogni nodo $i$ e i suoi nodi vicini $j$ che sono collegati ad $i$ attraverso gli edge del grafo. In altre parole, ogni nodo scambia informazioni con i nodi a cui è direttamente connesso. Questo processo permette a ogni nodo di aggiornare la sua rappresentazione basandosi non solo sulle proprie caratteristiche iniziali, ma anche sulle caratteristiche dei suoi vicini, propagando così l'informazione attraverso il grafo. Il processo di aggregazione iterativa permette alla GCN di migliorare le capacità di apprendere rappresentazioni significative.

# 5 Addestramento

Il processo di training è gestito dalla funzione *train()*, che segue i passaggi standard di ottimizzazione in PyTorch:

1. **Impostazione della Modalità di Training:** Il modello viene impostato in modalità training utilizzando *model.train()*;
2. **Azzeramento dei Gradienti:** I gradienti accumulati nei passi precedenti vengono azzerati chiamando *optimizer.zero_grad()*;
3. **Forward Pass:** I dati di input e la struttura del grafo (memorizzati nei tensori *data.x* e *data.edge_index*) vengono passati attraverso la rete. Questo produce un output che rappresenta le predizioni del modello;
4. **Calcolo della Loss:** La funzione di perdita utilizzata è la Negative Log-Likelihood Loss (*torch.nn.NLLLoss*), calcolata confrontando le predizioni del modello con le etichette reali dei nodi nel training set;
5. **Backward Pass:** Viene eseguito la backpropagation per calcolare i gradienti della loss rispetto ai pesi del modello;
6. **Aggiornamento dei Pesi:** Gli ottimizzatori aggiornano i pesi del modello in base ai gradienti calcolati e ai parametri di apprendimento definiti.

L'ottimizzazione avviene tramite l'algoritmo Adam, scelto per la sua efficienza nel gestire i gradienti e la stabilità nella convergenza. Per ottimizzare le prestazioni del modello, è stata eseguita una ricerca degli iperparametri utilizzando la *ParameterGrid* di scikit-learn. I parametri esplorati includono: le dimensioni dei canali nascosti, il tasso di apprendimento e il weight decay. Per ogni combinazione di parametri:

- Il modello viene addestrato per un numero fisso di epoche (200 in questo caso);
- La prestazione del modello viene valutata sul validation set;
- I parametri che producono la migliore accuratezza sul validation set vengono selezionati.

# 6 Risultati della valutazione

Dopo l'addestramento del modello utilizzando i migliori iperparametri trovati, è essenziale valutare le prestazioni del modello sul test set. Questo permette di misurare la capacità del modello, fornendo una stima realistica delle sue performance nel mondo reale. La valutazione del modello viene effettuata utilizzando la funzione *evaluate()*, che segue i seguenti passaggi:

1. **Impostazione della Modalità di Valutazione:** Il modello viene impostato in modalità di valutazione tramite *model.eval()*. Questo disabilita i meccanismi specifici del training come il dropout;
2. **Forward Pass senza Gradienti:** Viene eseguito un forward pass sui dati, ma senza calcolare i gradienti per migliorare l'efficienza computazionale e prevenire modifiche ai pesi del modello (*torch.no_grad()*);
3. **Calcolo delle predizioni:** Le predizioni del modello per i nodi nel test set vengono ottenute passando i dati di input (*data.x*) e la struttura del grafo (*data.edge_index*) attraverso la rete. Le predizioni sono poi convertite in etichette di classe utilizzando *argmax*;
4. **Calcolo dell'accuratezza:** L'accuratezza viene calcolata confrontando le predizioni con le etichette reali dei nodi nel test set. L'accuratezza rappresenta la proporzione di predizioni corrette sul totale dei nodi nel test set.

I risultati della valutazione includono diversi aspetti fondamentali. La loss durante il training viene monitorata costantemente mediante la funzione di loss (*NLLLoss*). Ad ogni decima epoca, la loss viene riportata insieme all'accuratezza sul training set, permettendo di verificare che il modello stia effettivamente imparando e che la loss diminuisca nel tempo. L'accuratezza sul training set fornisce una misura di quanto bene il modello si adatta ai dati su cui è stato addestrato, mentre l'accuratezza sul validation set è utilizzata durante la ricerca degli iperparametri per selezionare il miglior modello. Infine, l'accuratezza sul test set rappresenta la metrica più importante, poiché misura la capacità di generalizzazione del modello su dati non visti.

# 7 Performance del modello

**Setup Utilizzato:** L'allenamento del modello è stato eseguito su una GPU NVIDIA GeForce GTX 1070 Ti, con il supporto CUDA abilitato. Sono state utilizzate librerie come NetworkX per la gestione dei grafi, scikit-learn per la suddivisione dei dati in train, validation e test set, e PyTorch/PyTorch Geometric per le operazioni di convoluzione sui grafi. Il modello GCN implementato è composto da tre strati convoluzionali. Ogni strato è seguito da una funzione di attivazione ReLU e un livello di dropout per la regolarizzazione. L'output finale viene elaborato con una funzione di softmax per la classificazione binaria. Gli iperparametri sono stati ottimizzati utilizzando una grid search. I migliori iperparametri trovati sono stati: *hidden_channels* = [8, 4], *learning_rate* = 0.001, *weight_decay* = 0.0005. (**Best validation accuracy = 0.9038**).

Table 2: Processo di addestramento

| ***Epoch*** | ***Loss*** | ***Train accuracy*** |
|---|---|---|
| ***0*** | 0.6936 | 0.5047 |
| ***10*** | 0.6843 | 0.7072 |
| ***20*** | 0.6698 | 0.8783 |
| ***30*** | 0.6545 | 0.9192 |
| ***40*** | 0.6302 | 0.9496 |
| ***50*** | 0.6075 | 0.9664 |
| ***60*** | 0.5872 | 0.9769 |

| ***Epoch*** | ***Loss*** | ***Train accuracy*** |
|---|---|---|
| ***70*** | 0.5676 | 0.9780 |
| ***80*** | 0.5470 | 0.9780 |
| ***90*** | 0.5180 | 0.9790 |
| ***100*** | 0.5035 | 0.9790 |
| ***110*** | 0.4837 | 0.9790 |
| ***120*** | 0.4417 | 0.9822 |
| ***130*** | 0.4431 | 0.9822 |

| ***Epoch*** | ***Loss*** | ***Train accuracy*** |
|---|---|---|
| ***140*** | 0.4393 | 0.9832 |
| ***150*** | 0.4237 | 0.9832 |
| ***160*** | 0.3859 | 0.9832 |
| ***170*** | 0.3856 | 0.9832 |
| ***180*** | 0.3636 | 0.9832 |
| ***190*** | 0.3498 | 0.9832 |

Le performance del modello mostrano una buona capacità di generalizzazione, con un'accuratezza di validazione di oltre il 90% e un'accuratezza di test di circa l'89%. Durante l'allenamento, si osserva una costante diminuzione della loss e un incremento dell'accuratezza di training, indicando che il modello sta apprendendo efficacemente dai dati (Table 2). Questa configurazione e i risultati ottenuti dimostrano che il modello GCN implementato è efficace per la classificazione dei blog politici, riuscendo a catturare le relazioni tra i nodi del grafo e a utilizzarle per migliorare la predizione delle etichette (**Test accuracy = 0.8893**).

# 8 Sfide incontrate

Durante lo sviluppo del progetto, sono emerse diverse sfide che hanno richiesto soluzioni e un adattamento continuo. Di seguito sono descritte le principali sfide incontrate e le strategie adottate per superarle. Inizialmente, il problema principale è stato quello di caricare correttamente i dati in Python utilizzando la libreria NetworkX. Veniva restituito un errore dovuto alla presenza di edge duplicati nel grafo. Una soluzione iniziale consisteva nella creazione di una funzione Python per rimuovere i nodi duplicati from scratch. Tuttavia, dopo l'implementazione e la verifica del corretto funzionamento della funzione, è stato scoperto che l'errore poteva essere risolto modificando un attributo del grafo direttamente nel file *.gml* scaricato. Sebbene questa soluzione comportasse una minima modifica del dataset, è risultata più efficace e meno faticoso rispetto allo sviluppo di una funzione da zero. Di fatto il grafo è passato da essere un oggetto *DiGraph* ad uno *MultiDiGraph*, permettendo così edge duplicati. Una volta caricati correttamente i dati in Python, un'altra sfida significativa è stata quella di determinare le feature dei nodi del grafo. È emerso che i nodi possedevano solo l'etichetta e la sorgente come attributi. Tuttavia, l'attributo "sorgente" non forniva informazioni utili per la classificazione, poiché indicava semplicemente la provenienza del blog (ad esempio, un sito che raccoglie molti blog). Pertanto, è stato deciso di utilizzare il one-hot-encoding per rappresentare le etichette dei nodi. L'uso degli embedding dei contenuti testuali dei blog, come richiesto in maniera opzionale dalla traccia, è stato preso in considerazione. Tuttavia, una buona parte dei blog presenti nel dataset non era più accessibile e tra i blog accessibili vi era molta eterogeneità rendendo difficile l'estrazione e l'utilizzo del testo come embedding e feature dei singoli blog. Di conseguenza, si è scelto di non approfondire ulteriormente questa opzione. Un'ulteriore sfida è stata quella di determinare l'architettura ottimale del modello, provando diverse configurazioni con vari numeri di strati di convoluzione, con o senza dropout. È noto che un eccessivo numero di strati di convoluzione può portare a problemi di overfitting e ad una maggiore complessità computazionale, mentre un numero insufficiente può limitare la capacità del modello di apprendere caratteristiche significative dai dati. Pertanto, è stato necessario trovare un equilibrio ottimale, optando per un'architettura con tre strati convoluzionali, che ha dimostrato di essere sufficientemente profonda da catturare le relazioni tra i nodi del grafo senza introdurre complessità eccessiva.

# 9 Raccomandazioni per miglioramenti

Per migliorare ulteriormente la classificazione dei blog politici si potrebbero intraprendere diverse strade, sia dal punto di vista dei dati che dell'architettura del modello.

**Aggiornamento del Dataset:** Una delle principali raccomandazioni è l'uso di un dataset più aggiornato. Utilizzare dati recenti garantirebbe che il modello rifletta meglio il panorama attuale. Un dataset aggiornato permetterebbe, inoltre, un più facile uso degli embedding testuali, che possono arricchire le feature dei nodi con informazioni semantiche estratte direttamente dai contenuti dei blog. È essenziale però considerare la questione dell'eterogeneità dei blog e sviluppare metodi robusti per l'estrazione corretta del testo, tenendo conto delle diverse strutture e formati dei siti web da cui provengono i blog.

**Miglioramento dell'Architettura del Modello:** L'architettura del modello utilizzata, composta da tre strati di convoluzione, ha dimostrato di essere efficace per il dataset utilizzato. Tuttavia, per migliorare la robustezza e la scalabilità dell'architettura su altri tipi di dataset, si potrebbero considerare le seguenti possibilità:

- **Aggiustamento della Profondità del Modello:** Sebbene tre strati di convoluzione siano stati adeguati per questo progetto, è possibile sperimentare con architetture più profonde per vedere se possono catturare meglio le relazioni complesse in dataset più grandi. Tuttavia, si deve prestare attenzione al rischio di overfitting e alla complessità computazionale.
- **Adozione di Tecniche di Regolarizzazione Avanzate:** Oltre al dropout, si potrebbe considerare l'uso di tecniche di regolarizzazione avanzate come il batch normalization o il layer normalization, che possono migliorare la stabilità e la velocità di convergenza del modello.