



UNIVERSIDAD
POLITÉCNICA
DE MADRID



Máster en Aprendizaje Automático y Datos Masivos

Análisis de Redes Sociales 2024

**Memoria Práctica 2.
Jaime Álvarez Urueña y Alejandro Mendoza Medina.**

1.0. Diseño de las arquitecturas GNN

Se han desarrollado una serie de experimentos con distintas arquitecturas GNN, entre las que se encuentran:

- **Agregador universal:**

- Mensaje: MLP de las features.
- Agregación: Sumatorio de los mensajes.
- Actualización: MLP de la agregación.

- **GRU con skip connection:**

- Mensaje: Capa lineal con las features como entrada.
- Agregación: Suma de los mensajes.
- Actualización: GRU de la agregación y del nodo central.

- **GNN con Attention:**

- Mensaje: Capa lineal con las features como entrada.
- Agregación: Mecanismo de atención de los mensajes y nodo central.
- Actualización: Función identidad.

- **Concat skip connections:**

- Mensaje: Capa lineal con las features como entrada.
- Agregación: Suma de los mensajes.
- Actualización: Capa lineal de la concatenación de la agregación y el nodo central.

- **Interpolate skip connections:**

- Mensaje: Capa lineal con las features como entrada.
- Agregación: Suma de los mensajes.
- Actualización: Suma ponderada entre el nodo central y el resultado de aplicar un MLP a la agregación.

- **Jumping knowledge:**

- Mensaje: Skip connections entre las distintas capas.
- Agregación: Concatenación/Max pooling/LSTM-attention. En este caso, concatenación.
- Actualización: MLP de la agregación.

2.0. Proceso de entrenamiento

Se han seguido dos procesos distintos de entrenamiento para cada una de las arquitecturas. En ninguno de ellos se han utilizado técnicas de data augmentation. Los datos de los que se dispone son bastante abundantes, contando con un total de 19.717 nodos (serán divididos en los conjuntos de entrenamiento, validación y test) y 88.648. Solo hay 3 clases, y por tanto no se han requerido técnicas de data augmentation.

El primero de los entrenamientos hechos para cada arquitectura se ha realizado sin ningún tipo de dataloader. Los datos no son tan abundantes ni tan pesados como para tener que hacer una carga dinámica de ellos. Esto puede llevar a dificultades en la convergencia, ya que el no usar batch equivale a usar un tamaño de batch de todos los datos, y por tanto se está computando la pérdida (y por tanto actualizando los parámetros de las redes) sobre todos los datos a la vez. Por ello, se ha

llevado a cabo un segundo entrenamiento con un dataloader. Las posibles configuraciones de dataloaders aplicados a grafos pueden ser muy variables. En este caso, Se ha hecho uso de la clase NeighborLoader en el paquete torch_geometric.loader. El parámetro num_neighbors consiste de una lista con los enteros 25 y 10. Se ha usado la modalidad disjunta para todos los entrenamientos. En el proceso de entrenamiento se ha incluido una traza para mostrar información sobre el entrenamiento.

3.0. Proceso de validación

El conjunto de validación se ha usado para validar el modelo entrenado con el conjunto de entrenamiento a la hora de hacer un grid search para la optimización de hiperparámetros como el número de capas, tamaño de cada capa o tamaño de embeddings. Una vez que el modelo ha sido refinado, se ha testeado sobre el conjunto de test, para obtener la métrica final. Para cada modelo, se ha obtenido su métrica F1 para el conjunto de validación y de test para posteriormente representarlos. Así mismo, se han obtenido matrices de confusión para ambos conjuntos para cada entrenamiento realizado.

4.0. Resultados

En este apartado únicamente se van a incluir las figuras con las comparaciones de las métricas de cada GNN. Las matrices de confusión de cada modelo para cada política de entrenamiento están disponibles en el código entregado.

En la Figura 1 se observa que el modelo **Concat skip connection** es el modelo que mejor métrica ha obtenido sobre el conjunto de test con un 0,8975. No obstante, si se realizara un test estadístico para ver si hay diferencias significativas entre los modelos (t-test pareado en este caso), y que las diferencias no se deben al azar, seguramente no se obtuviera un nivel de significancia inferior a 0,05 (estándar para concluir que un modelo tiene diferencias estadísticas respecto a otro). Tal vez si se realizaran los entrenamientos de nuevo o se aumentaran el número de épocas, o se hiciera un grid search más exhaustivo, los resultados podrían diferir.

Por otro lado, en la Figura 2 se muestra que los modelo **GRU con skip connections**, **Concat skip connections**, y **Interpolate skip connection** obtienen la misma métrica sobre el conjunto de test (0.898) . Superan a sus respectivos modelos entrenados sin dataloaders, y también superan al modelo que mejor métrica obtuvo sobre el conjunto de test habiendo sido entrenado sin dataloader. Por otra parte, los resultados de otros modelos se degradan mucho bajando hasta un 6,2 % en Jumping knowledge y un 3,5% en Graph Attention. En el caso del Agregado universal, su métrica mejora un 2,1%.

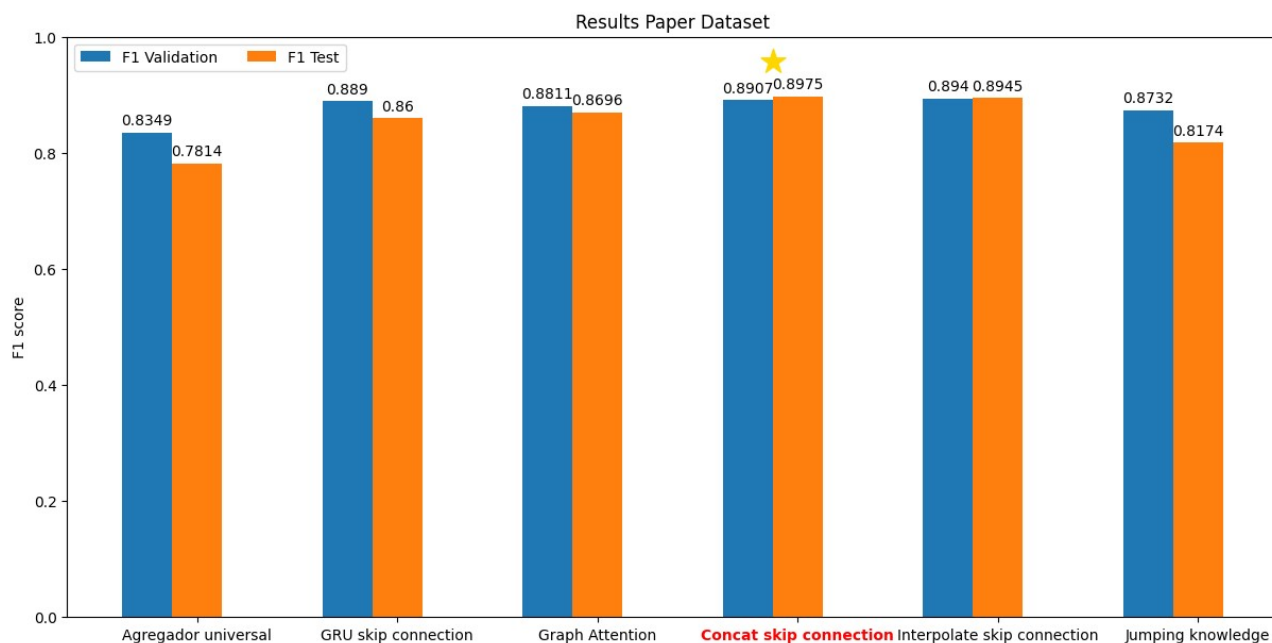


Figura 1. Comparación de las métricas F1 sobre el conjunto de validación y test para cada GNN entrenada. Todos los modelos, en este caso, han sido entrenados con 300 épocas y sin dataloader.

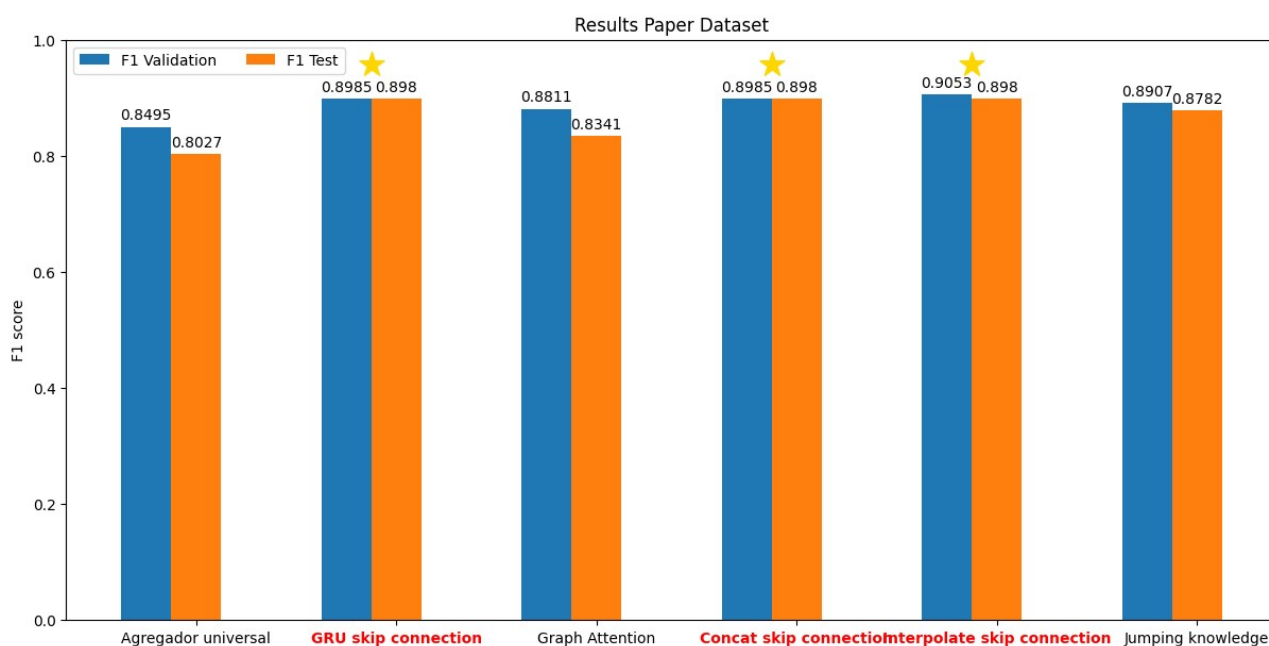















Figura 2. Comparación de las métricas F1 sobre el conjunto de validación y test para cada GNN entrenada (300 épocas con NeighborLoader).

Tabla 1. Comparación de las métricas obtenidas usando y sin usar NeighborLoader

GNN	Sin NeighborLoader	Con NeighborLoader
Agregador Universal	0,7814	0,8027
GRU skip connection	0,86	0,898
Graph Attention	0,8696	0,8341
Concat skip connection	0,8975	0,898
Interpolate skip connection	0,8945	0,898
Jumping knowledge	0,8174	0,8782

En el torneo realizado en clase el día 19-12-2024, el equipo formado por Jaime Álvarez Urueña y Alejandro Mendoza Medina (Soft_Gonzalo) obtuvo un resultado de 0,9001 quedando en 7º lugar.

Tournament Rankings	
✓ 	NoduleModule: 0.8915 (0.9077)
✓ 	señor_fernando: 0.9037 (0.9072)
✓ 	Cortihoz: 0.9072 (0.9072)
✓ 	TiramisuTeam: 0.8981 (0.9062)
✓ 	RosMi: 0.9037 (0.9037)
✓ 	C2: 0.8854 (0.9006)
✓ 	Soft_Gonzalo: 0.8783 (0.9001)
✓ 	equipo-dragon: 0.8945 (0.8991)
✓ 	GRAPHITI: 0.8879 (0.8991)
✓ 	FOMO: 0.8920 (0.8986)
✓ 	cocretras: 0.8971 (0.8971)
✓ 	RGs: 0.8651 (0.8849)
✓ 	mlp: 0.8844 (0.8844)

5.0 Conclusiones y trabajo futuro

Se ha realizado un trabajo extenso, incluyendo una gran cantidad de entrenamientos de modelos, pues se ha realizado una pequeña grid search para cada modelo estudiado. Además se ha incluido el uso de dataloaders en grafos. Todas las GNN presentadas obtienen resultados muy buenos para el problema dado, siendo muy positivo. La comparación realizada entre las distintas arquitecturas GNN y el uso de NeighborLoader ha resultado de gran ayuda para familiarizarse tanto con los conceptos teóricos de la materia, como con su implementación en pytorch.

Por otra parte, se ha incluido mucha información visual que puede facilitar en gran medida la interpretación de resultados (en el código se puede observar todas las gráficas obtenidas).

El trabajo futuro consiste en explorar más las opciones que puede ofrecer los dataloaders, ya que sin hacer una grid search muy extensa, 5/6 GNN han mejorado. Habría que usar el modo joint de NeighborLoader y compararlo con los resultados obtenidos.