

# **Torneo SNA 2025**

Predicción de la lipofilicidad de una molécula  
Análisis de Redes Sociales

Máster en Aprendizaje Automático y Datos Masivos

Paula Corral Rebollar  
Lucía Cuevas Serrano

## Diseño de la GNN

Para esta práctica se ha diseñado un modelo basado en Graph Isomorphism Network (GIN), adecuado para tareas de regresión sobre grafos moleculares. La arquitectura está compuesta por tres componentes principales: update, mensaje/agregación y pooling:

### Update

Cada nodo se actualiza mediante una MLP dentro de la capa GINConv, que incluye dos capas lineales con ReLU intermedia. Antes de aplicar las capas GIN, los nodos se proyectan a un embedding de dimensión `hidden_channels` mediante una capa lineal y se normalizan con `BatchNorm`, asegurando estabilidad desde la primera capa y facilitando la convergencia del modelo.

```
# Proyección inicial
self.input_proj = torch.nn.Linear(num_features, hidden_channels)
self.input_norm = torch.nn.BatchNorm1d(hidden_channels)

# Capas GIN con MLP
for _ in range(n_layers):
    mlp = torch.nn.Sequential(
        torch.nn.Linear(hidden_channels, hidden_channels),
        torch.nn.ReLU(),
        torch.nn.Linear(hidden_channels, hidden_channels))
    self.convs.append(GINConv(mlp, train_eps=True))
```

### Message/Aggregate

La agregación de mensajes se realiza mediante la suma de los vectores de los nodos vecinos, propia de la capa GINConv, para capturar la estructura local de cada molécula. Además, cada capa aplica `BatchNorm` tras la convolución, seguida de conexiones residuales con `LayerNorm` junto con activación `LeakyReLU` y `dropout`, para mejorar la estabilidad del entrenamiento en redes profundas y prevenir overfitting.

```
for conv, norm in zip(self.convs, self.norms):

    # GINConv con agregación por suma
    h = conv(x, edge_index)

    # BatchNorm
    h = norm(h)

    # LeakyReLU, dropout, residual + LayerNorm
    h = torch.nn.functional.leaky_relu(h, self.negative_slope)
    h = torch.nn.functional.dropout(h, p=self.dropout, training=self.training)
    x = torch.nn.functional.layer_norm(x + h, x.size()[1:])
```

### Pool y MLP final

Para obtener la representación global de la molécula, se combina pooling global por media y máximo sobre los nodos del grafo. Se usan ambos poolings para capturar tanto la tendencia promedio de los nodos como los valores extremos. La representación resultante se pasa por un MLP profundo, con capas lineales, LeakyReLU, LayerNorm y dropout. Finalmente, un regresor lineal predice la lipofilicidad, y la salida se limita al rango  $[-3, 7]$ , como establece el enunciado.

```
# Pooling global
x = torch.cat([global_mean_pool(x, batch), global_max_pool(x, batch)], dim=1)

# MLP
x = self.pool_mlp(x)

# Regresor
out = self.regressor(x)
out = torch.clamp(out, min=-3.0, max=7.0)
```

### Parámetros finales de la GNN

Se encuentra una combinación final de `hidden_channels=64`, `n_layers=6`, `dropout=0.1`, `negative_slope=0.2` obteniendo los mejores resultados a través de un proceso de grid search explicado en la siguiente sección.

### Proceso de entrenamiento

Para evitar el sesgo debido a la distribución no uniforme de la lipofilicidad se emplea estratificación por bins (usando 10 bins) y `WeightedRandomSampler`, asignando pesos inversamente proporcionales a la frecuencia de cada bin para que los valores menos frecuentes se seleccionen más a menudo durante el entrenamiento. Previamente se realizaron pruebas con otras estrategias de muestreo, incluyendo duplicación de datos para aumentar el tamaño del dataset, que inducía a que el modelo memorizase valores en lugar de generalizar correctamente. También se probó una estratificación con todas las bins del training y validation set con el mismo número de muestras, que aunque funcionaba correctamente, no superaba a la versión mostrada en la práctica.

Después se separan los datos en train set (85%) y validation set (15%) y se preparan los data loaders. También se hace un data loader para el último entrenamiento del modelo final con todos los datos.

Se establecieron después de algunas pruebas un optimizador Adam con learning rate  $1e-3$  y weight decay  $1e-4$  y un scheduler de CosineAnnealingLR para ajustar dinámicamente la learning rate con `T_max=300` y `eta_min=1e-6` y una paciencia de 100 epochs para el early stopping.

En la primera fase del entrenamiento se realiza un grid search para buscar los hiperparámetros que obtienen los mejores resultados en validation RMSE. Iteramos sobre distintas combinaciones de hidden\_channels (64, 128, 256, 512), n\_layers (2, 3, 4, 5, 6), dropout (0.1, 0.25, 0.4, 0.5) y negative\_slope (0.01, 0.05, 0.1, 0.2) para las LeakyReLU. Este entrenamiento se hizo utilizando el servicio de Kaggle con GPU T4 x2. Toda la ejecución duró aproximadamente 10 horas. En el apartado del proceso de validación hablaremos de las métricas utilizadas en la *fase 1*. Decidimos aquí hacer una única prueba por combinación dada la cantidad de posibilidades distintas, aunque es cierto que sería más correcto realizar un número determinado de iteraciones (por ejemplo 10, como veremos a continuación) para tener una representación más fiel del desempeño de un modelo. Al hacer este análisis identificamos modelos cuyo desempeño no era nada competitivo y no hubiese mejorado respecto a otros con ningún número de iteraciones.

Después de la primera fase, se obtiene una lista con las 18 combinaciones con mejores resultados, y se hace de nuevo un entrenamiento con esas 18 combinaciones. Para cada una de ellas, se entrena el modelo 10 veces y se evalúa. Si se detecta en alguna iteración un resultado mejor, se reemplaza y se guarda la combinación con la que se ha obtenido. Al final de la ejecución se obtiene la combinación con mejores resultados. En el apartado del proceso de validación hablaremos de las métricas utilizadas en la *fase 2*.

Finalmente, se entrena el modelo con esa combinación de hiperparámetros, el optimizer y el scheduler con todos los datos de entrenamiento, y se fija el número de epochs a los epochs más recientes con los que se ha obtenido el mejor resultado + 300 para dejar que el modelo aprenda bien la nueva distribución (hemos añadido los datos de validación que previamente no se usaban para entrenar).

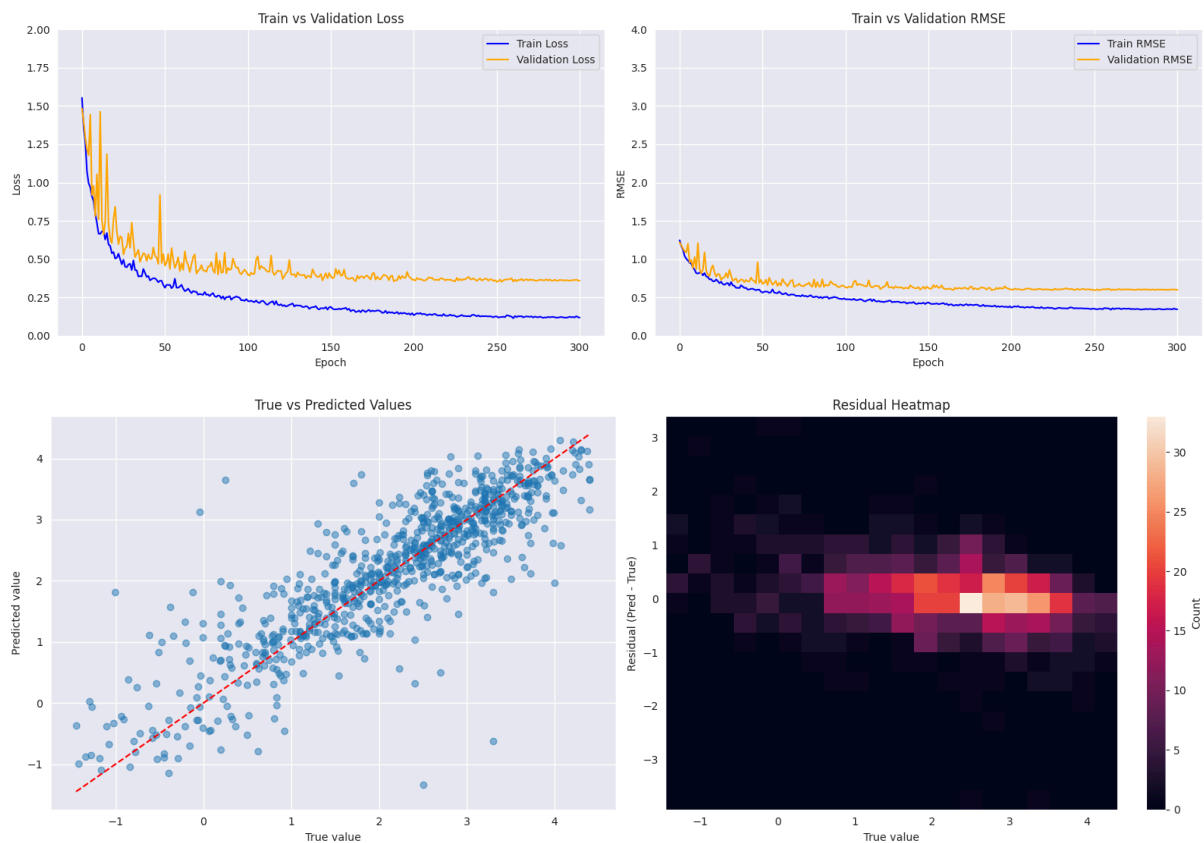
### Proceso de validación

Es importante mencionar que, aunque en la práctica se muestra únicamente la arquitectura final y su optimización de hiperparámetros, se realizaron múltiples pruebas con distintas configuraciones de GNN para evaluar su impacto en el rendimiento. Se exploraron modelos con y sin proyección inicial, BatchNorm, bloques residuales y LayerNorm, funciones de activación LeakyReLU y ReLU, distintos esquemas de pooling (add, mean, max y sus combinaciones), MLPs de diferentes profundidades, y con o sin clamping en el rango  $[-3, 7]$ . Tras comparar los resultados, la arquitectura presentada en esta memoria resultó ser la que mejores métricas de validación alcanzaba.

Después de entrenar los modelos para el grid search de la *fase 1*, se evalúan sus resultados según el RMSE en el validation set. Se hace lo mismo con los modelos de la *fase 2*, comparando en todo momento sus valores obtenidos de RMSE en el

validation set. Se escoge esta métrica al ser la que se comprobará después en el test set.

Una vez hemos encontrado la mejor combinación, antes de entrenar el modelo con todos los datos, lo entrenamos una última vez utilizando el train loader y lo evaluamos frente al set de validación, obteniendo un RMSE de 0.5928040947076866 y estos plots:

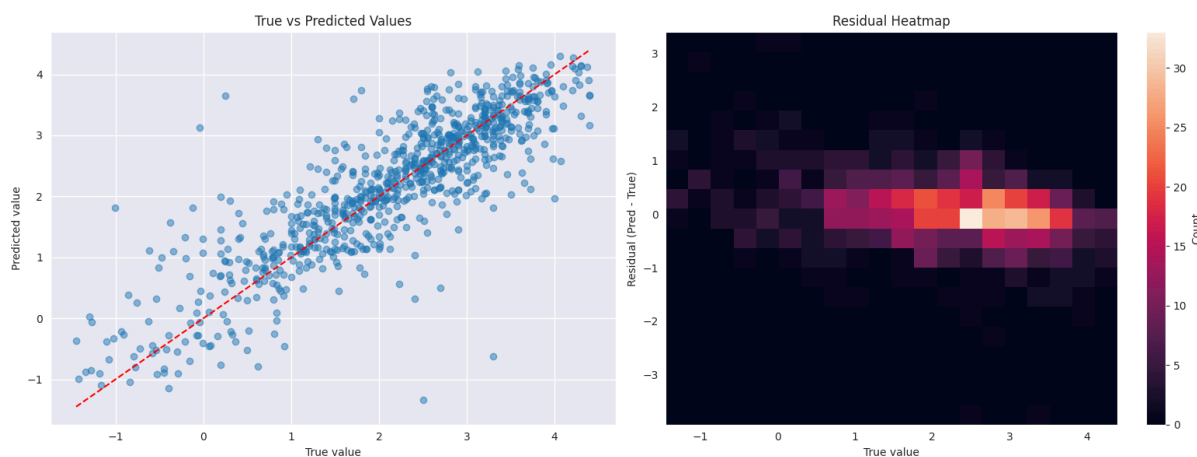


En el caso de la práctica lo ejecutamos una única vez ya que la combinación ya está seleccionada y los resultados de esta celda no afectan al resto de la práctica (sólo el número de epochs en los que se alcanza el mejor resultado, que tras varias pruebas vemos que ronda un rango de valores fijo), aunque la celda está preparada para hacer varias pruebas y generar los plots con la de mejores resultados.

Después, entrenamos el modelo con la combinación de hiperparámetros sobre todos los datos, y el número de epochs utilizados en la validación anterior redondeados a la centena y añadiendo 300 para que pudiese aprender mejor la nueva distribución de datos. La validación de este modelo se detalla en la siguiente sección.

## Resultados

Una vez se entrena el modelo final con todos los datos y los hiperparámetros optimizados, predecimos los valores del test set y comparamos los resultados, obteniendo el siguiente plot:



Podemos ver en el output de la celda que conseguimos un valor de RMSE en el test set de 0.6292484826469851, obteniendo una gran mejora con respecto a nuestra primera propuesta en el torneo. Vemos que la correlación entre valores reales y predichos capta la tendencia del dataset correctamente.

## Conclusiones y trabajo futuro: Resumen del trabajo realizado, valoración de la GNN presentada, limitaciones y posibles mejoras.

Para concluir, vemos que la GNN basada en GIN es capaz de aprender representaciones efectivas de moléculas para predecir lipofilicidad. La estratificación por bins y el uso de WeightedRandomSampler fueron claves para equilibrar la distribución de los datos y evitar sesgo hacia valores más frecuentes, y para ajustar de manera indirecta el nivel de overfitting presente en el modelo.

Además, la combinación de pooling global (media + max) y MLP profundo permite capturar tanto información estructural como patrones complejos de interacción entre nodos, frente a otras arquitecturas probadas. La selección de hiperparámetros es también clave en el entrenamiento de GNNs de regresión, pudiendo alcanzar desempeños muy diferentes.

Como limitaciones podemos mencionar que el modelo no incorpora explícitamente información química avanzada, que en producción podría ser muy útil y eficiente. Además, la predicción de valores extremos de lipofilicidad es más difícil debido a la menor presencia de ejemplos en esos rangos. Una posible mejora incluiría una mayor capacidad de cómputo para hacer más pruebas con

distintos hiperparámetros y estructuras, pudiendo obtener mejores resultados que los presentados en esta práctica.

Como trabajo futuro, sería interesante introducir características adicionales en los nodos y aristas, como cargas parciales o propiedades topológicas. También se podrían explorar arquitecturas híbridas GNN + Transformers para mejorar la captura de relaciones globales o experimentar con técnicas de data augmentation químico, como mutaciones de grafos o generación de moléculas virtuales.

### Bibliografía

Xu, K., Hu, W., Leskovec, J., & Jegelka, S. (2019). How Powerful are Graph Neural Networks? arXiv [Cs.LG]. Retrieved from <http://arxiv.org/abs/1810.00826>

Fey, M., & Lenssen, J. E. (2019). Fast Graph Representation Learning with PyTorch Geometric. arXiv [Cs.LG]. Retrieved from <http://arxiv.org/abs/1903.02428>