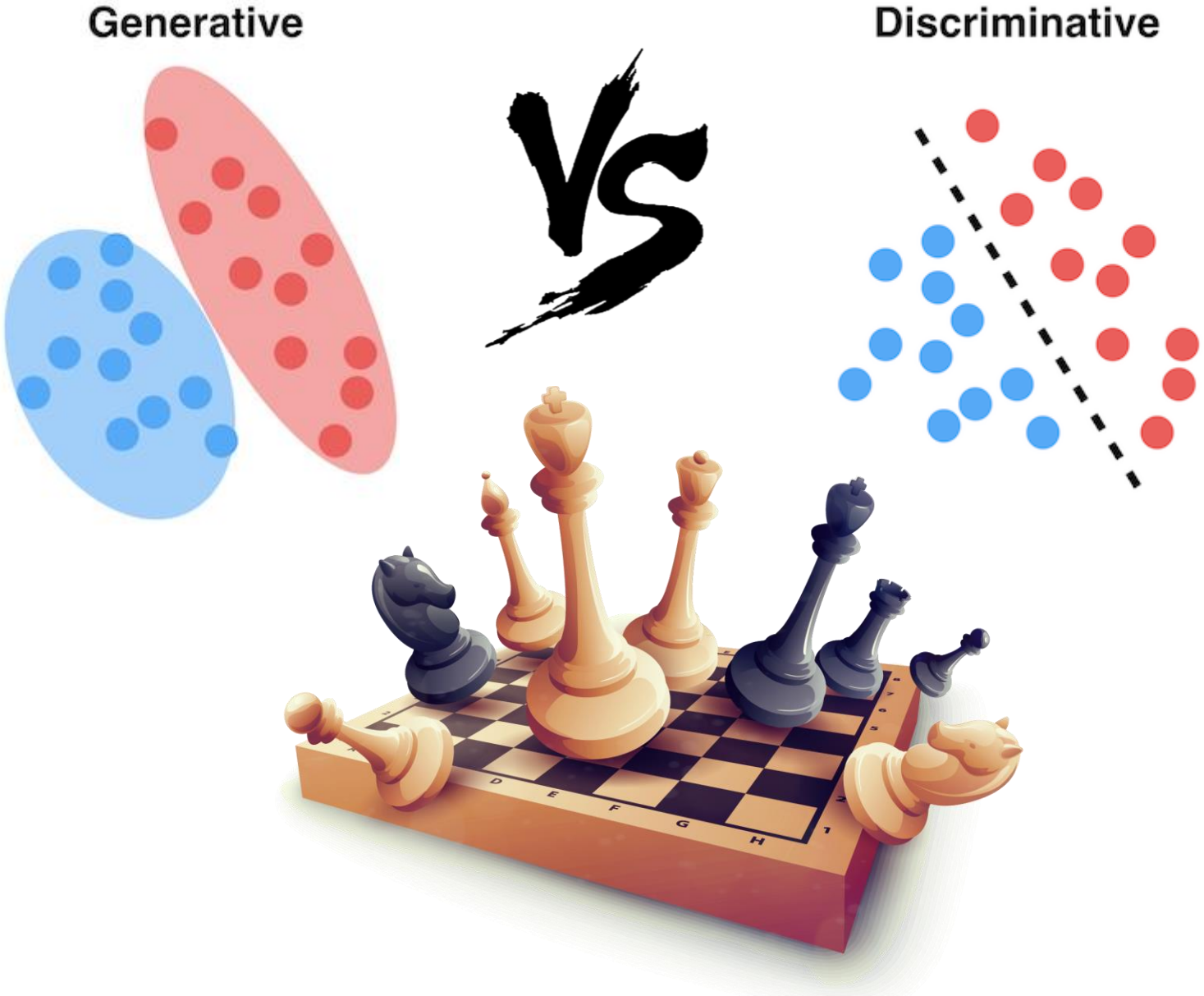


GraphGAN

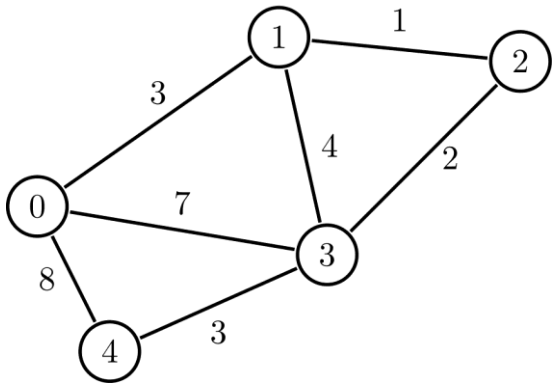
- **Integrantes:**
  - Benjamín Farías V.
  - Víctor Hernández
  - Benjamín Lepe

Contenidos

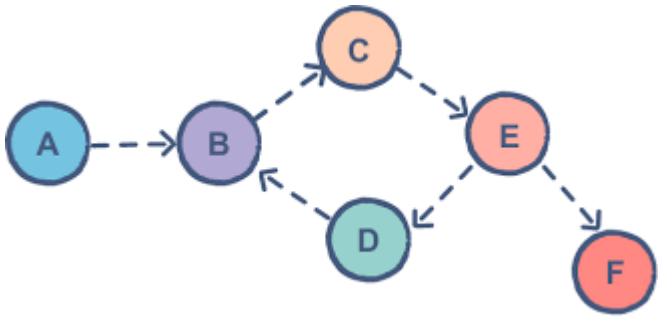
1. **Contexto**
2. Estado del Arte
3. GraphGAN
4. Experimentos
5. Conclusiones

Contexto - Representación de Grafos

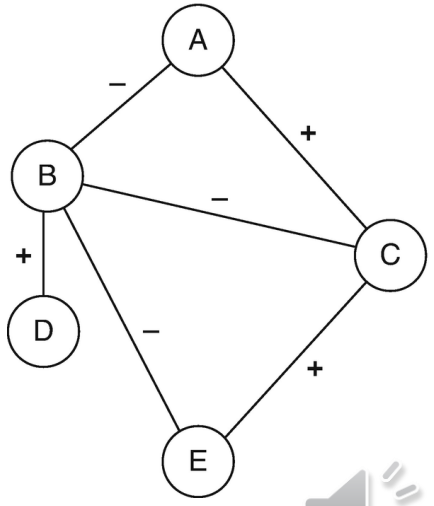
- También conocido como *network embeddings*
- Apunta a representar cada vértice de un grafo como *low-dimension vectors*
- Se busca facilitar las tareas de análisis del grafo y predicción sobre los vértices y aristas del mismo



Weighted Graph



Directed Graph



Signed Graph

Contexto - Aplicaciones

- *Link Prediction*
  - Interesado en predecir, en base a dos vértices dados, si existe o no una conexión entre ambos
- *Node Classification*
  - Cada vértice se encuentra asignado a una o varias etiquetas, y dado un grupo de vértices, nos interesa clasificar el resto con las etiquetas correspondientes
- *Recommendation*
  - Se pretende recomendar películas que podrían gustarle a un usuario, sin haberlas visto antes

Contexto - Otras Aplicaciones

- *Visualization*
- *Knowledge Graph Representation*
- *Clustering*
- *Text Embedding*
- *Social Network Analysis*

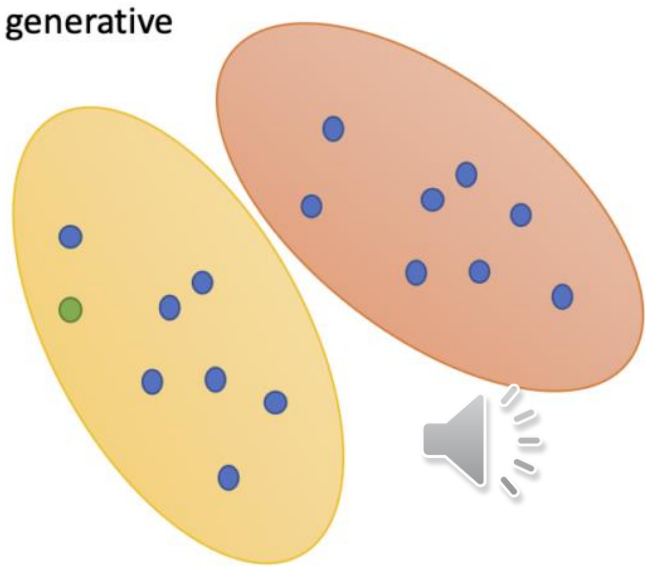
Contenidos

1. Contexto
2. **Estado del Arte**
3. GraphGAN
4. Experimentos
5. Conclusiones

Estado del Arte - Aprendizaje Generativo

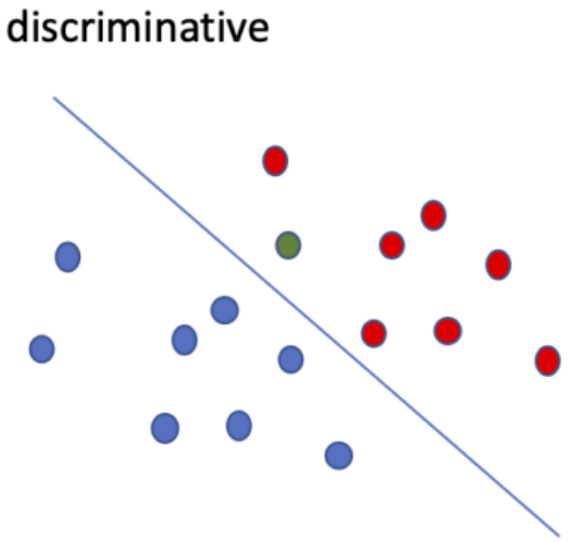
- Para cada vértice v_c existe una **distribución condicional** implícita $p_{true}(v|v_c)$ que indica las conexiones de preferencia del vértice v_c respecto a todos los vértices del grafo
- Las aristas del grafo pueden verse como muestras generadas por distribuciones condicionales donde los modelos generativos aprenden los *vertex embeddings* al maximizar la **probabilidad de que exista una conexión**

Node2Vec



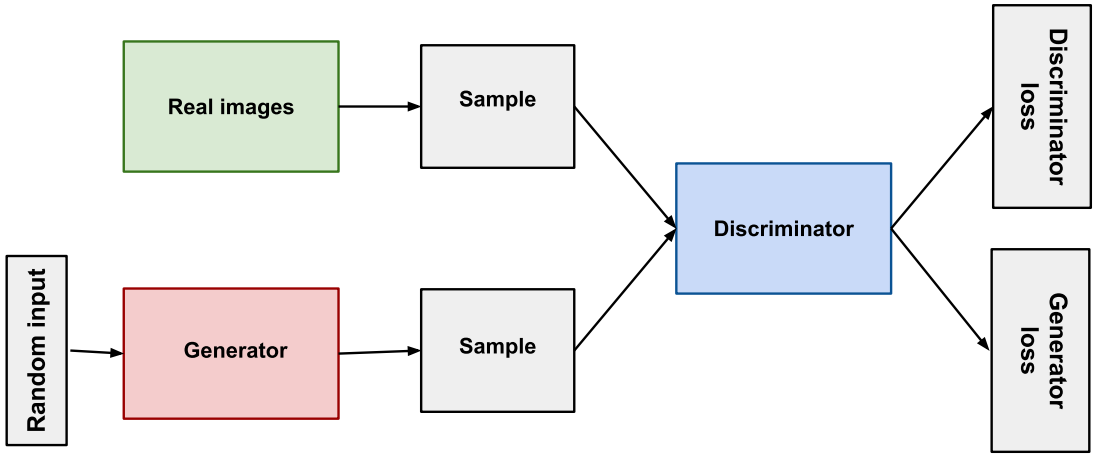
Estado del Arte - Aprendizaje Discriminativo

- A diferencia del modelo generativo, en este las aristas de los vértices no se consideran generadas por distribuciones condicionales implícitas
- Este modelo apunta a predecir la existencia de aristas directamente, comprobando la **probabilidad de conexión** entre dos vértices v_i, v_j como $p(edge|(v_i, v_j))$, basado en los datos de entrenamiento



Estado del Arte - GANs

- Combinación de **discriminador y generador**
- Mucho éxito recientemente con sus aplicaciones
  - *Image Generator (https://thispersondoesnotexist.com/)*
  - *Sequence Generator*
  - *Dialogue Generator*
  - *Etc.*

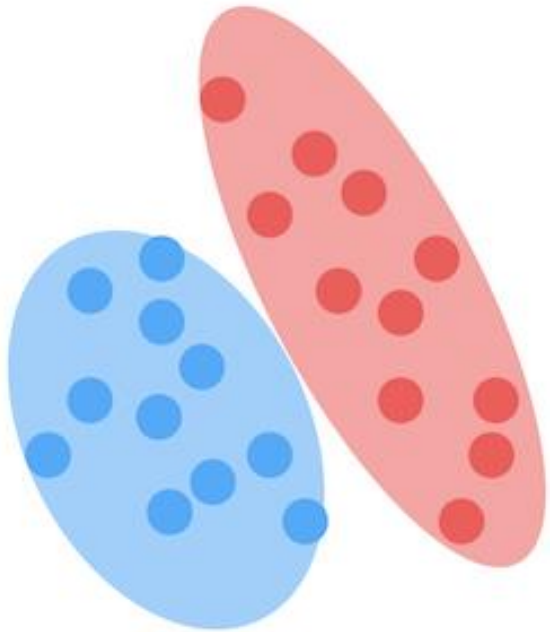


Contenidos

1. Contexto
2. Estado del Arte
3. **GraphGAN**
4. Experimentos
5. Conclusiones

GraphGAN - Framework

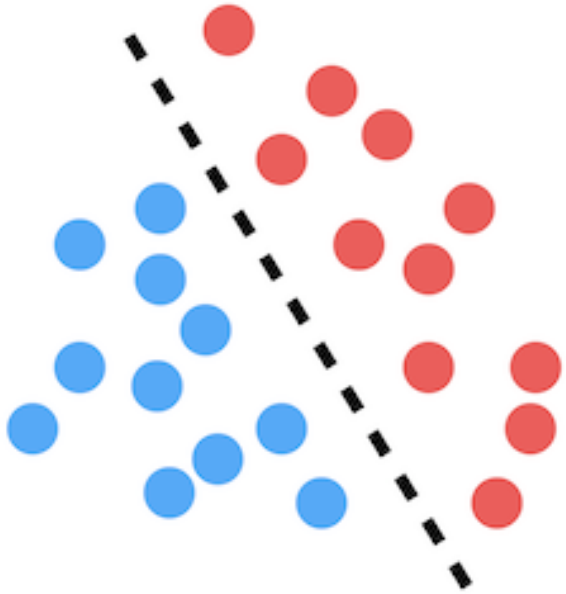
Generador (G)



Genera nodos con mayor probabilidad de estar conectados con v_c

$$p_{\text{true}}(v|v_c)$$

Discriminador (D)

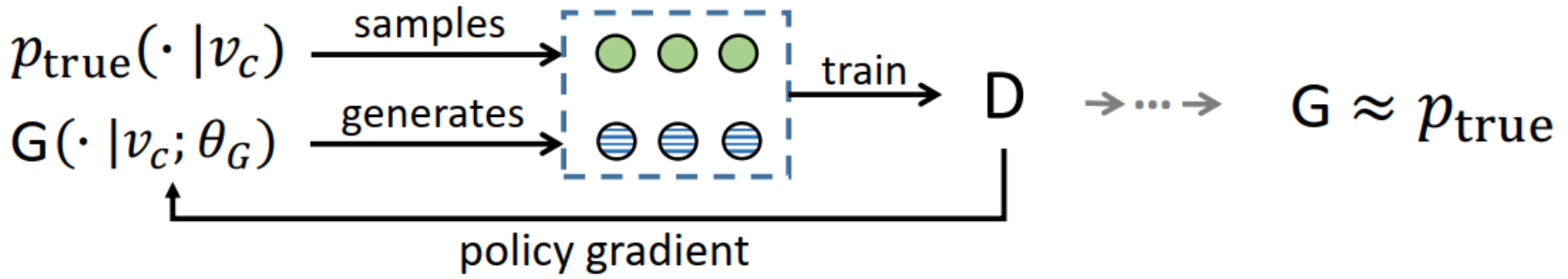


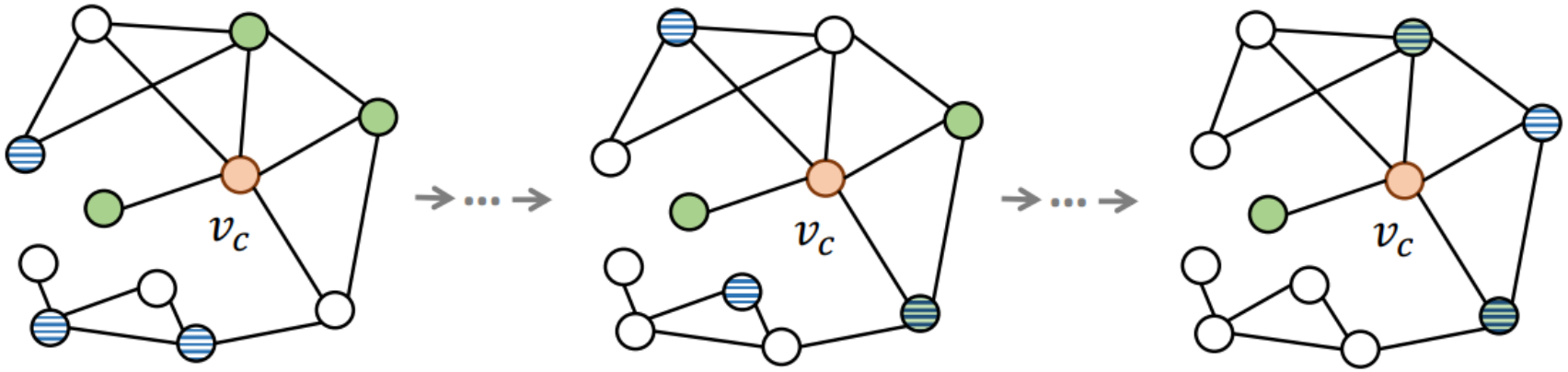
Discrimina la conectividad entre pares de nodos, entregando la probabilidad

$$p(edge|(v_i, v_j))$$

GraphGAN - Framework

- Juego *Minimax*, **D** debe discriminar entre vecinos **reales** y vecinos **generados** por **G**





G underperforms in initial stage

G is approaching p_{true} during adversarial training

G is hardly distinguishable from p_{true}

GraphGAN - Discriminador (D)

- Determina si el nodo v es vecino **real o generado**, aplicando activación **sigmoidal**

$$D(v, v_c) = \sigma(\mathbf{d}_v^\top \mathbf{d}_{v_c}) = \frac{1}{1 + \exp(-\mathbf{d}_v^\top \mathbf{d}_{v_c})}$$

- Se optimiza mediante **ascenso de gradiente**, buscando **maximizar** la probabilidad de discriminar correctamente

$$\nabla_{\theta_D} V(G, D) = \begin{cases} \nabla_{\theta_D} \log D(v, v_c), \ if \ v \sim p_{\text{true}}; \\ \nabla_{\theta_D} \big(1 - \log D(v, v_c)\big), \ if \ v \sim G. \end{cases}$$

GraphGAN - Generador (G)

- Genera muestras de vecinos v aproximando la distribución mediante ***softmax***

$$G(v|v_c) = \frac{\exp(\mathbf{g}_v^\top \mathbf{g}_{v_c})}{\sum_{v \neq v_c} \exp(\mathbf{g}_v^\top \mathbf{g}_{v_c})}$$

- Se optimiza mediante **descenso de gradiente**, buscando **minimizar** la probabilidad de que **D** identifique los vecinos generados por **G**

$$\nabla_{\theta_G} V(G, D) = \sum_{c=1}^{V} \mathbb{E}_{v \sim G(\cdot|v_c)} \left[\nabla_{\theta_G} \log G(v|v_c) \log \left(1 - D(v, v_c)\right) \right]$$

GraphGAN - Generador (G)

La función ***softmax*** es intuitiva, pero tiene ciertas limitaciones en este contexto:

- La optimización aplica a **todos los nodos**, lo que es **ineficiente**
- Trata a los nodos por igual, desaprovechando la **información estructural**

Se propone una alternativa a *softmax* para **G**, denominada ***graph softmax***

GraphGAN - Graph Softmax

- Dado un nodo raíz v_c, se computa **BFS** en el grafo, obteniendo un árbol T_c
- Se define la **relevancia de** v_i **dado** v en base a los vecinos de v en T_c:

$$p_c(v_i|v) = \frac{\exp(\mathbf{g}_{v_i}^\top \mathbf{g}_v)}{\sum_{v_j \in \mathcal{N}_c(v)} \exp(\mathbf{g}_{v_j}^\top \mathbf{g}_v)}$$

- Dado el camino $P_{v_c \to v} = (v_{r_0}, v_{r_1}, ..., v_{r_m})$, se define finalmente:

$$G(v|v_c) \triangleq \left(\prod_{j=1}^{m} p_c(v_{r_j}|v_{r_{j-1}})\right) \cdot p_c(v_{r_{m-1}}|v_{r_m})$$

GraphGAN - Graph Softmax

La función ***graph softmax*** cumple con las propiedades deseadas:

- Es una **distribución de probabilidades** válida (dem. por inducción sobre T_c)
- La probabilidad **decrece exponencialmente con la distancia** entre nodos
- La **complejidad** de calcularla en general es $O(d \log V)$

Se elige ***graph softmax*** para representar al generador **G!**

GraphGAN - Algoritmo Generador

- ***Random walk*** sobre T_c, se retorna el nodo en el que se decida **devolverse**

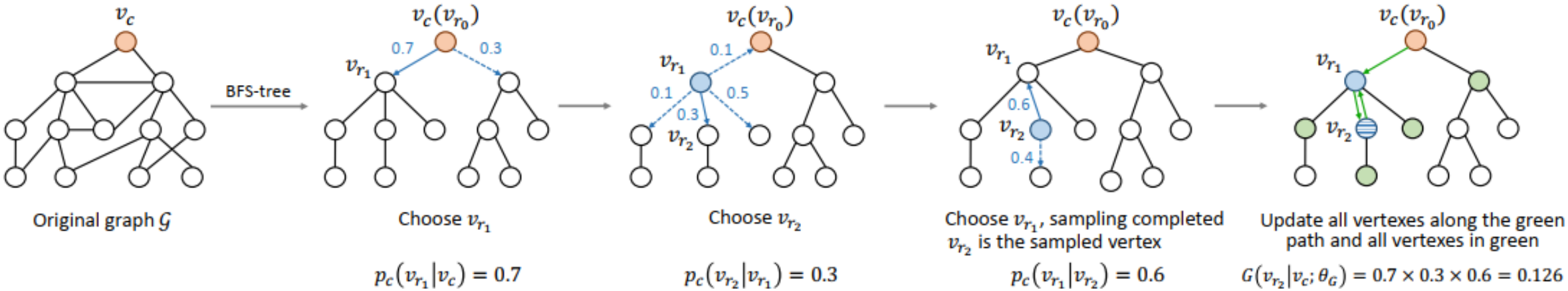
Algorithm 1 Online generating strategy for the generator

Require: BFS-tree T_c, representation vectors $\{\mathbf{g}_i\}_{i \in \mathcal{V}}$
Ensure: generated sample v_{gen}

$v_{pre} \leftarrow v_c, v_{cur} \leftarrow v_c$;
while true do
  Randomly select v_i proportionally to $p_c(v_i|v_{cur})$ in Eq. (6);
  if $v_i = v_{pre}$ **then**
    $v_{gen} \leftarrow v_{cur}$;
    return v_{gen}
  else
    $v_{pre} \leftarrow v_{cur}, v_{cur} \leftarrow v_i$;
  end if
end while

GraphGAN - Algoritmo Generador

- En cada paso se elige un vecino al azar, condicionado en su **relevancia** p_c
- Cuando un nodo elija a su **padre**, se detiene y **retorna dicho nodo**
- Se deberán **actualizar los pesos** de todos los **nodos del camino y sus vecinos**



GraphGAN - Algoritmo Final

- Construir el árbol T_c para cada nodo
- Iterar el juego *Minimax* entre **G** y **D**
- Retornar los *embeddings* obtenidos

- Complejidad: $O(V \log V)$

Algorithm 2 GraphGAN framework

Require: dimension of embedding k, size of generating samples s, size of discriminating samples t
Ensure: generator $G(v|v_c;\theta_G)$, discriminator $D(v, v_c;\theta_D)$

```
Initialize and pre-train G(v|v_c;θ_G) and D(v, v_c;θ_D);
Construct BFS-tree T_c for all v_c ∈ V;
while GraphGAN not converge do
    for G-steps do
        G(v|v_c;θ_G) generates s vertices for each vertex v_c according to Algorithm 1;
        Update θ_G according to Eq. (4), (6) and (7);
    end for
    for D-steps do
        Sample t positive vertices from ground truth and t negative vertices from G(v|v_c;θ_G) for each vertex v_c;
        Update θ_D according to Eq. (2) and (3);
    end for
end while
return G(v|v_c;θ_G) and D(v, v_c;θ_D)
```

Contenidos

1. Contexto
2. Estado del Arte
3. GraphGAN
4. **Experimentos**
5. Conclusiones

Experimentos - Setup

Se utilizaron estos 5 **datasets** para evaluar GraphGAN:

- arXiv-AstroPh
- arXiv-GrQc
- BlogCatalog
- Wikipedia
- MovieLens-1M

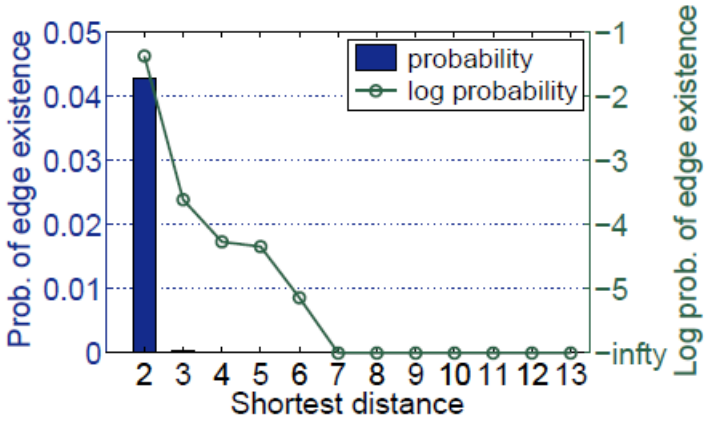
Experimentos - Setup

Se utilizaron estos 4 **modelos** como *baseline* para GraphGAN:

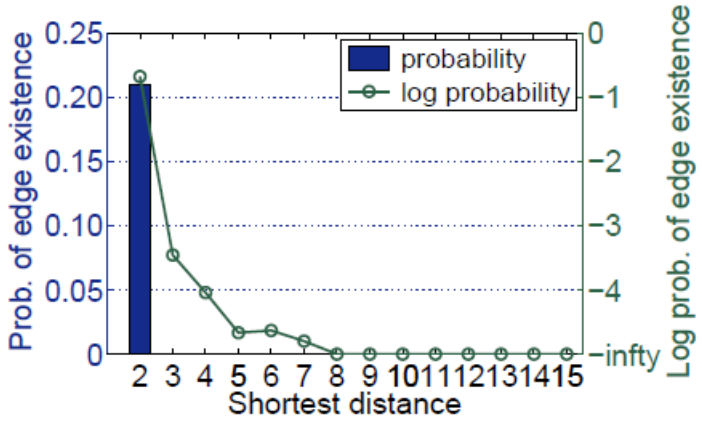
- DeepWalk
- LINE
- Node2Vec
- Struc2Vec

Experimentos - Resultados

La probabilidad de que exista una arista entre dos vértices del grafo, **disminuye** al aumentar la distancia mínima *(shortest distance).*



(a) arXiv-AstroPh



(b) arXiv-GrQc

Resultados - Link Prediction

- *GraphGAN* supera a todos los demás modelos en las dos métricas para los datasets *arXiv-AstroPh and arXiv-GrQc.*

Model	arXiv-AstroPh		arXiv-GrQc	
	Acc	Macro-F1	Acc	Macro-F1
DeepWalk	0.841	0.839	0.803	0.812
LINE	0.820	0.814	0.764	0.761
Node2vec	0.845	0.854	0.844	0.842
Struc2vec	0.821	0.810	0.780	0.776
GraphGAN	**0.855**	**0.859**	**0.849**	**0.853**

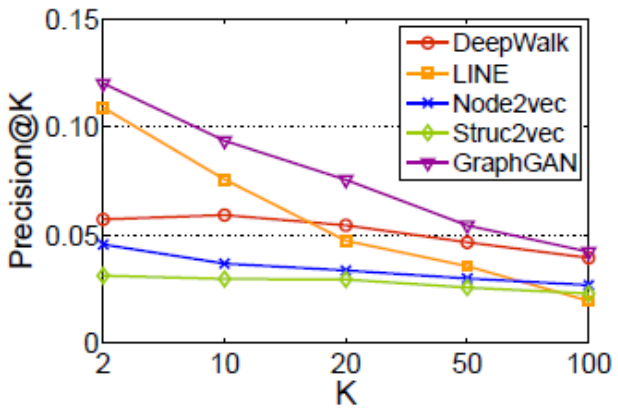
Resultados - Node Classification

- *GraphGAN* supera a todos los demás modelos en las dos métricas para los datasets *BlogCatalog y Wikipedia*.

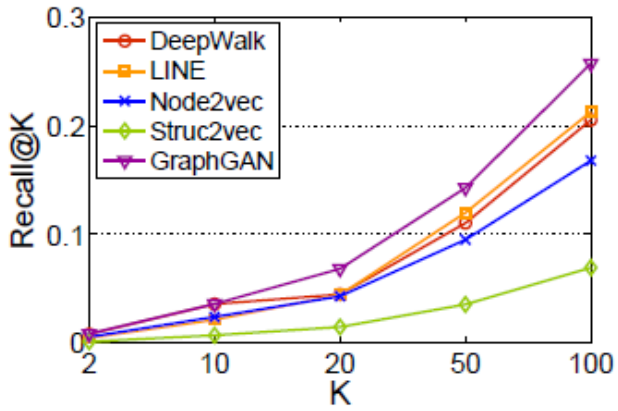
Model	BlogCatalog		Wikipedia	
	Acc	Macro-F1	Acc	Macro-F1
DeepWalk	0.225	0.214	0.194	0.183
LINE	0.205	0.192	0.175	0.164
Node2vec	0.215	0.206	0.191	0.179
Struc2vec	0.228	0.216	0.211	0.190
GraphGAN	**0.232**	**0.221**	**0.213**	**0.194**

Resultados - Recomendación

- *GraphGAN* supera a todos los demás modelos en las dos métricas para el dataset *MovieLens-1M*.



(a) Precision@K



(b) Recall@K

Contenidos

1. Contexto
2. Estado del Arte
3. GraphGAN
4. Experimentos
5. **Conclusiones**

Conclusiones

- Se propuso un **framework** que es capaz de **unificar** dos tipos de GNN: generativas y discriminativas
- El algoritmo ayuda a que estas dos redes se **impulsen a aprender** más eficientemente, ya que se enfrentan como **oponentes** en un juego *Minimax*
- Se introdujo una **nueva función generativa** llamada *graph softmax*
- *GraphGAN* logró superar todos los *baselines* propuestos, demostrando que es un modelo más **robusto y preciso**

Bibliografía y Referencias

[1] Wang, H. et al. (2017). *GraphGAN: Graph Representation Learning with Generative Adversarial Nets.*