Deep Graph Infomax

Petar Veličković[1,2], William Fedus[2,3,5], William L. Hamilton[2,4], Pietro Liò[1], Yoshua Bengio[2,3] and R Devon Hjelm[6,2,3]

[1]University of Cambridge [2]Mila [3]Université de Montréal
[4]McGill University [5]Google Brain [6]Microsoft Research Montréal

Università degli Studi di Siena Seminar Series 17 December 2018

Introduction

- In this ICLR 2019 submission, we study the problem of **unsupervised graph representation learning**.
  - That is, learning “good” representations for nodes in an *unlabelled graph*.
  - Very important problem: many graphs in the wild are unlabelled!
- We rely on a *synergy* of **mutual information maximisation** and **graph convolutional networks**—enabling simultaneously leveraging *local* and *global* information propagation in a graph.
- Very promising results on the learnt embeddings—often competitive with supervised learning!

Roadmap for today

- **A tale of many (random) walks;**
- The (DIM) fairy godmother;
- Happily ever (mutually) informative.

Random walk-based objectives

- Thus far, the field of unsupervised node embeddings has been *dominated* by random walk-based methods.
- Some methods go even simpler—and use a *link prediction* objective (*"random walks of length 1"*):
  - see e.g. GAE (Kipf & Welling, NIPS BDL 2016) or EP (Duran & Niepert, NIPS 2017).
- I will briefly present two such methods, and outline a few reasons why we should also consider *alternative* objectives.

Overview of DeepWalk (Perozzi *et al.*, KDD 2014)

- Start by random features $\vec{\Phi}_i$ for each node i.
- Sample a random walk $\mathcal{W}_i$, starting from node i.
- For node x at step j, $x = \mathcal{W}_i[j]$, and a node y at step $k \in [j-w, j+w]$, $y = \mathcal{W}_i[k]$, modify $\vec{\Phi}_x$ to maximise $\log \mathbb{P}(y|\vec{\Phi}_x)$ (obtained from a neural network classifier).
- Inspired by **skip-gram models** in natural language processing: to obtain a good vector representation of a word, its vector should allow us to easily predict the words that *surround* it.

Overview of DeepWalk, *cont'd*

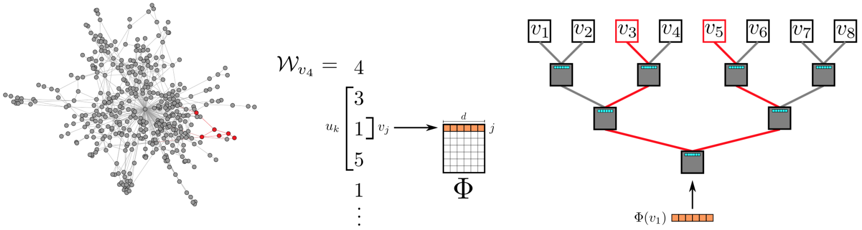
- Expressing the full $\mathbb{P}(y|\vec{\Phi}_x)$ distribution directly, even for a single layer neural network, where

$$\mathbb{P}(y|\vec{\Phi}_x) = softmax(\vec{w}_y^T\vec{\Phi}_x) = \frac{\exp\left(\vec{w}_y^T\vec{\Phi}_x\right)}{\sum_z \exp\left(\vec{w}_z^T\vec{\Phi}_x\right)}$$

  is prohibitive for large graphs, as we need to normalise across the entire space of nodes—making most updates *vanish*.

- To rectify, DeepWalk expresses it as a *hierarchical softmax*—a tree of binary classifiers, each halving the node space.

DeepWalk in action



(a) Random walk generation. (b) Representation mapping. (c) Hierarchical Softmax.

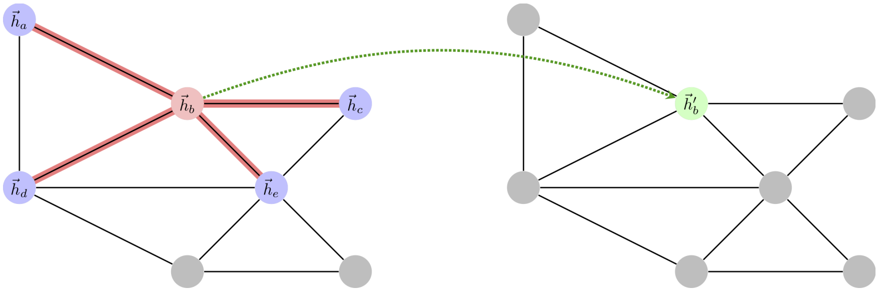
Figure 3: Overview of DEEPWALK. We slide a window of length $2w + 1$ over the random walk $\mathcal{W}_{v_4}$, mapping the central vertex v_1 to its representation $\Phi(v_1)$. Hierarchical Softmax factors out $\Pr(v_3 \mid \Phi(v_1))$ and $\Pr(v_5 \mid \Phi(v_1))$ over sequences of probability distributions corresponding to the paths starting at the root and ending at v_3 and v_5. The representation Φ is updated to maximize the probability of v_1 co-occurring with its context $\{v_3, v_5\}$.

Later improved by *LINE* (Tang *et al.*, WWW 2015) and *node2vec* (Grover & Leskovec, KDD 2016), but main idea stays the same.

Some limitations

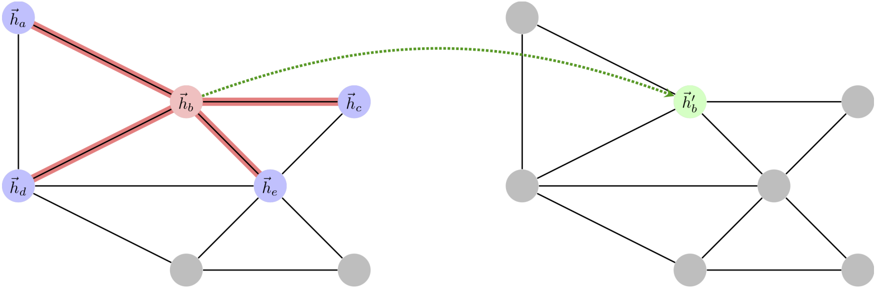
- These kinds of techniques are in principle very powerful—due to their relation to methods such as PageRank—but have known limitations:
  - They tend to over-emphasise *proximity* information at the expense of *structural* information (as discussed in struc2vec; Ribeiro *et al.*, 2017)
  - Performance can be highly dependent on the choice of *hyperparameters* (as discussed in both DeepWalk and node2vec papers).
  - Much harder to apply in **inductive** settings, especially on entirely unseen graphs!
- To alleviate some of these issues, let's use something fresh out the toolbox...

Graph convolutional networks



Let's assume that we have (intermediate) features, $\mathbf{H} = \{\vec{h}_1, \ldots, \vec{h}_N\}$ ($\vec{h}_i \in \mathbb{R}^F$) in the nodes...

Graph convolutional networks



In a nutshell, obtain higher-level representations of a node i by leveraging its *neighbourhood*, $\mathcal{N}_i$!

$$\vec{h}_i^{\ell+1} = g^\ell(\vec{h}_a^\ell, \vec{h}_b^\ell, \vec{h}_c^\ell, \dots) \ \ (a, b, c, \dots \in \mathcal{N}_i)$$

where g^ℓ is the ℓ-th *graph convolutional layer*.

GCNs meet random walks!

- If we, instead, use a *stack of (inductive) graph convolutional layers* to derive our embeddings, $\vec{z}_i$...
- ...and then express our random walk objective on these embeddings...
- ...we can fix both the *stability* and *inductivity* issues simultaneously!
- First done in GraphSAGE (Hamilton *et al.*, NIPS 2017), with the following objective:

$$\mathcal{L}(\vec{z}_u) = -\log \sigma(\vec{z}_u^T \vec{z}_v) - Q \cdot \mathbb{E}_{v_n \sim P_n(v)} \log \sigma(\vec{z}_u^T \vec{z}_{v_n})$$

where P_n is a *negative sampling* distribution.

Should GCNs meet random walks?

- Random walk objectives essentially enforce a bias towards proximal nodes having similar embeddings.
- But a *graph convolutional network* summarises *local patches* (centered around each of the nodes) of the graph by design!
- As neighbours experience a lot of patch-level mixing, *they will already have similar embeddings* as a result of using a GCN!
- It may be argued that random walks may *fail to provide any useful signal at all* in this case.

$\Longrightarrow$ In the age of the GCN, we should perhaps seek a move towards **non-local** objectives.

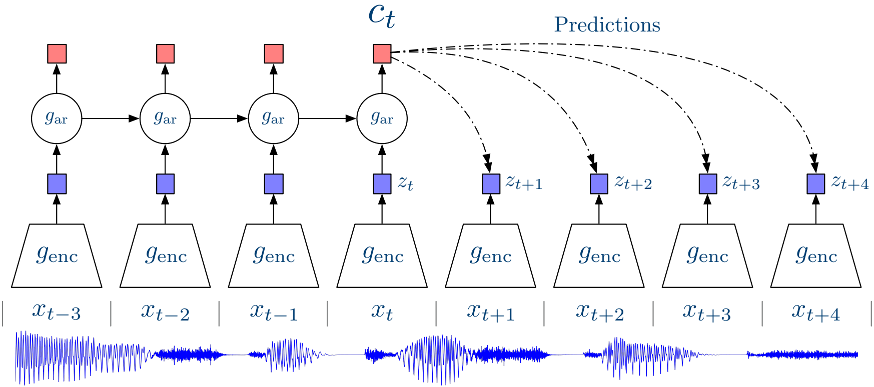
Roadmap for today

- A tale of many (random) walks;
- **The (DIM) fairy godmother;**
- Happily ever (mutually) informative.

Towards non-local objectives

- As was the case in many other situations, we should think of graphs as *generalisations of **images***, and therefore it is natural to seek inspiration from advances in the vision domain.
- Luckily, we live in very exciting times, with **two** highly relevant techniques being released in the past two months!
  - Contrastive Predictive Coding (CPC; Oord *et al.*, 2018)
  - Deep InfoMax (DIM; Hjelm *et al.*, 2018)
- Both techniques rely on *mutual information maximisation* between different parts of the input, **in the latent space**!

Contrastive Predictive Coding



Relies on an *autoregressive model* (LSTM/PixelCNN) to produce context, $\vec{c}_t$, from latents, $\vec{z}_t$, and therefore unlikely to be trivially applicable to graphs (requires a node ordering).

And now…

…a brief slide-deck intermission!
(Many thanks to Devon Hjelm)

DEEP INFOMAX (DIM)

Or: how to exploit structure for fun and profit

Devon Hjelm (MSR / MILA)

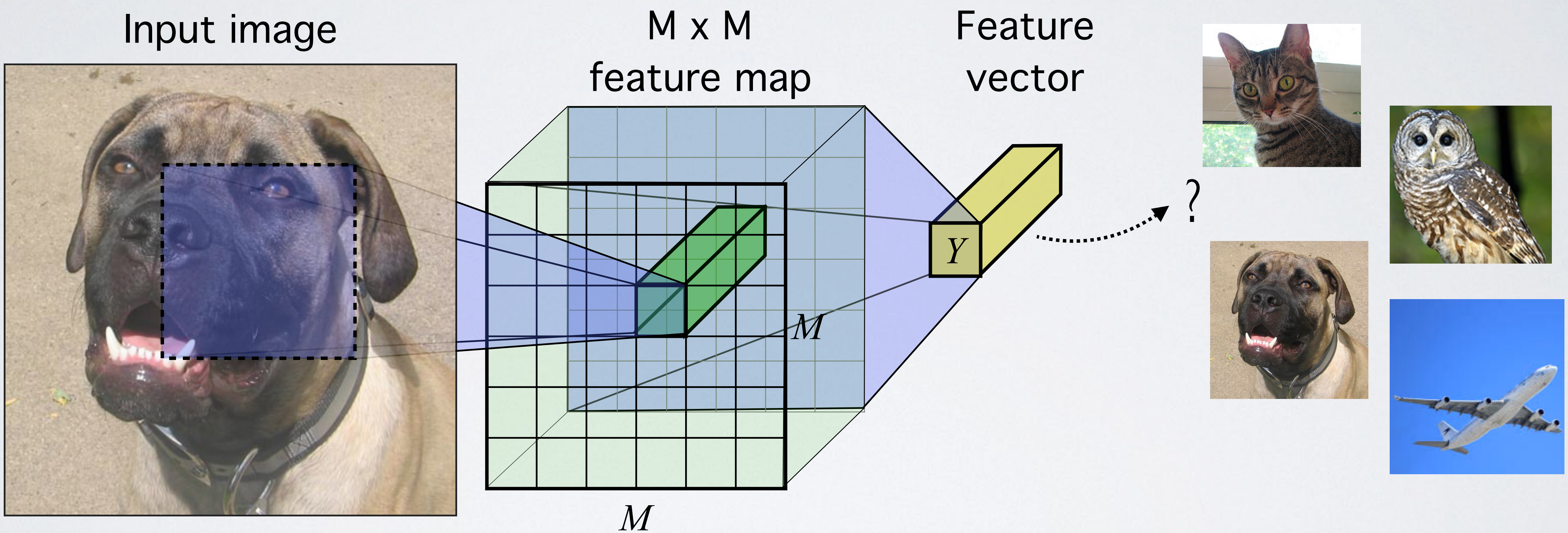
Work with: Alex Fedorov (MRN), Sam Lavoie (MILA), Karan Grewal (U Toronto), Adam Trischler, and Yoshua Bengio

Paper title: Learning deep representations by information estimation and maximization
Found at: https://arxiv.org/abs/1808.06670
Github code: https://github.com/rdevon/DIM

OVERVIEW

- Goal: Learn good representations
- Simple setting: representation of images
- What is a “good” representation?
- Deep INFOMAX
- Future / ongoing work

LEARNING REPRESENTATIONS OF IMAGES

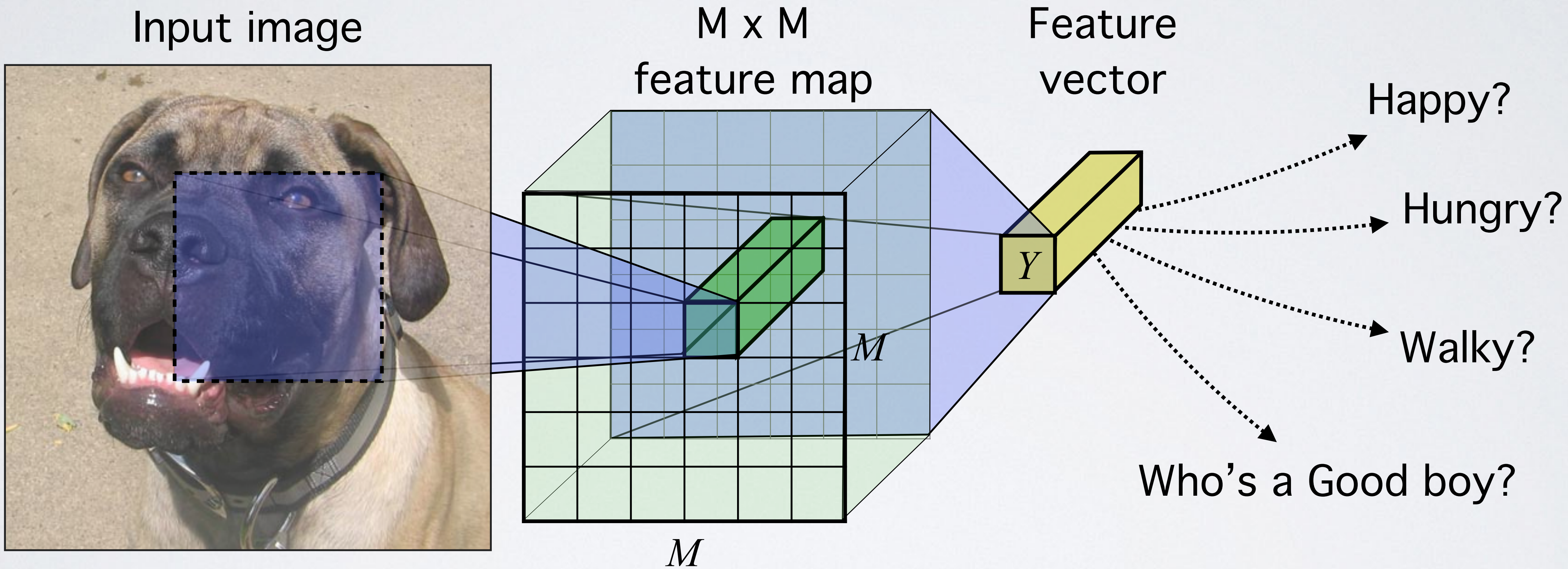


Step 1: encode image into MxM feature map

Step 2: Summarize the image into a single (1D) feature vector

Step 3: Profit!

LEARNING REPRESENTATIONS OF IMAGES



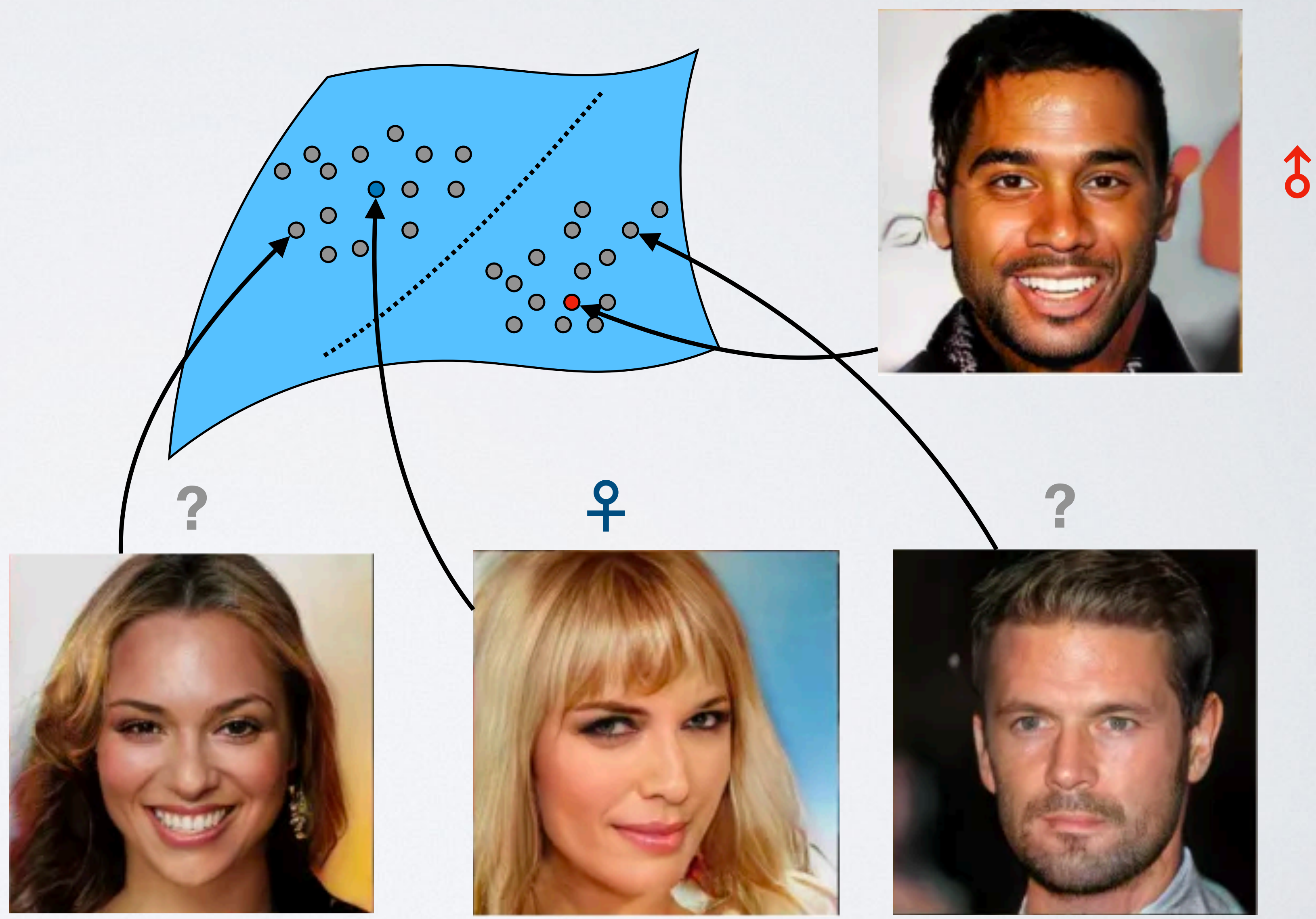
Step 1: encode image into MxM feature map

Step 2: Summarize the image into a single (1D) feature vector

Step 3: Profit!

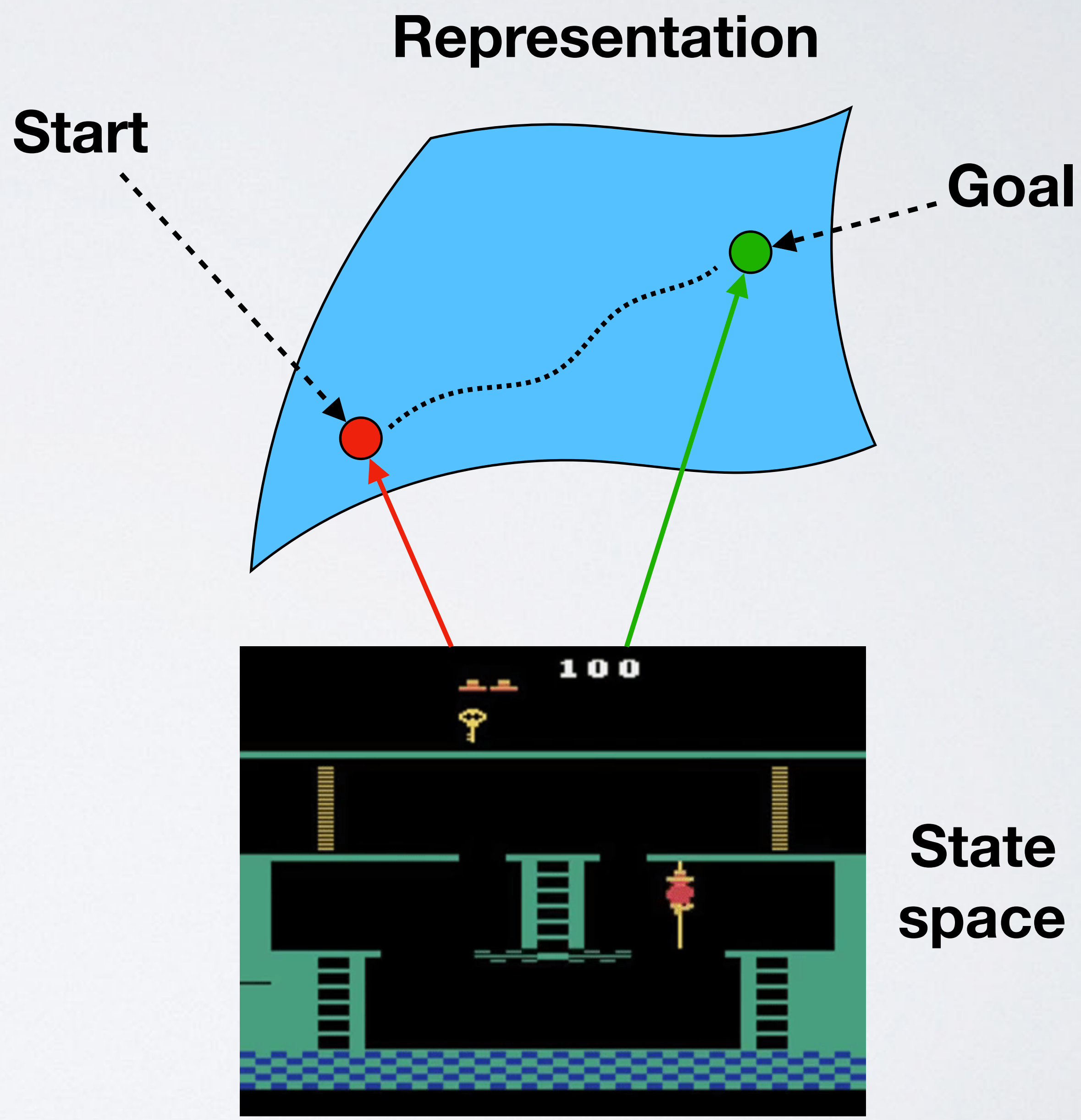
WHY DO WE NEED "GOOD" REPRESENTATIONS?

- Low dimensional "simple" summary of the data
- Deal with limited data (semi-supervised, zero-shot learning, etc)
- Multi-model learning (visually grounded NLP)
- Discover underlying structure / discovery



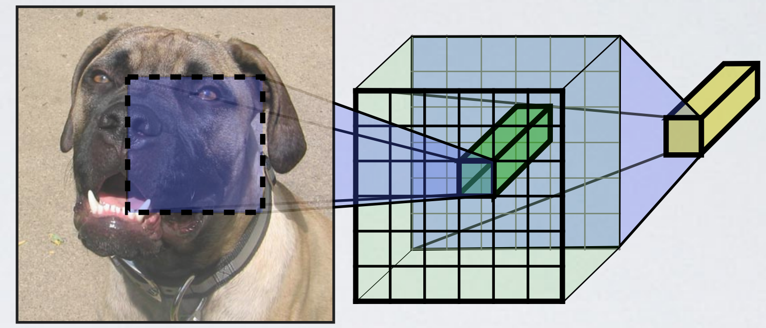
WHY DO WE NEED "GOOD" REPRESENTATIONS?

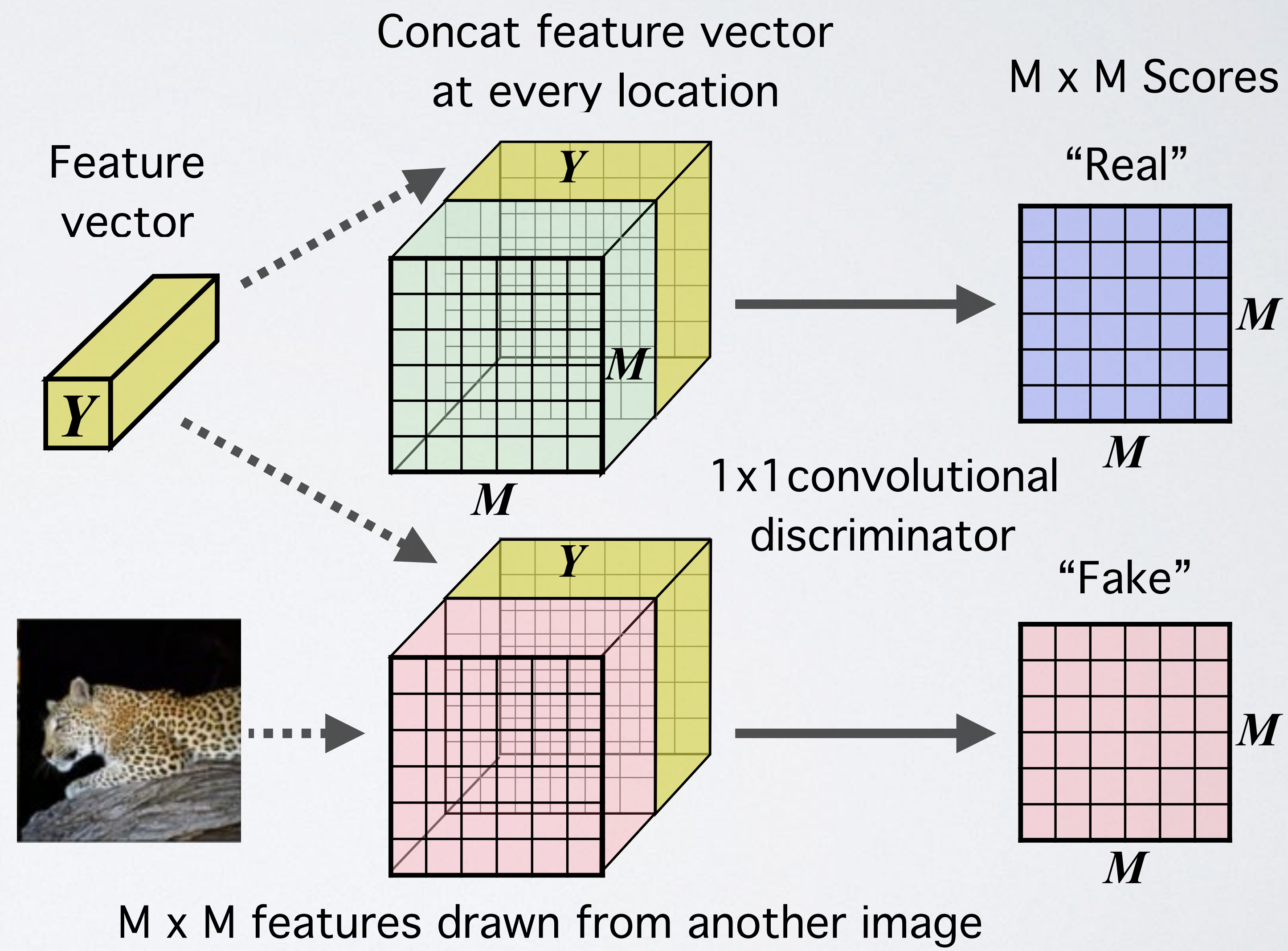
- Low dimensional "simple" summary of the data
- Deal with limited data (semi-supervised, zero-shot learning, etc)
- Multi-model learning (visually grounded NLP)
- Discover underlying structure / discovery
- Planning in RL



DEEP IMPLICIT INFOMAX

1. Encode your image
2. Concatenate feature vector with feature map at every location
3. 1x1 convolution
4. Call this "real"
5. Encode another image
6. Concatenate 1st feature vector with feature map of new image
7. 1x1 convolution
8. Call this "fake".
9. Loss function averaged over locations
10. Train the whole thing like a GAN discriminator (aka like a binary classifier)
11. Profit!





DEEP IMPLICIT INFOMAX

Simple classification experiment

- Fully supervised = normal classifier
- Build a small classifier with 200 hidden units on representation
  - conv = last conv layer
  - fc = next to last 1d layer
  - Y = final representation
  - SS = fine tuned with labels

Results

- DIM outperforms all models compared by large margins on most tasks
- Performs comparably to BiGAN with random crops
- Global version, DIM(G) tries to maximize mutual information with full input

Model	CIFAR10 conv	CIFAR10 fc (1024)	CIFAR10 $Y(64)$	CIFAR100 conv	CIFAR100 fc (1024)	CIFAR100 $Y(64)$
Fully supervised	75.39			42.27		
VAE	60.71	60.54	54.61	37.21	34.05	24.22
AAE	59.44	57.19	52.81	36.22	33.38	23.25
BiGAN	62.57	62.74	52.54	37.59	33.34	21.49
NAT	56.19	51.29	31.16	29.18	24.57	9.72
DIM(G)	52.2	52.84	43.17	27.68	24.35	19.98
DIM(L)	**70.1**	**70.21**	**63.97**	**48.46**	**46.09**	**36.51**

	Tiny ImageNet conv	Tiny ImageNet fc (4096)	Tiny ImageNet $Y(64)$	STL-10 (random crop pretraining) conv	STL-10 fc (4096)	STL-10 $Y(64)$	STL-10 SS
Fully supervised	36.60			68.7			
VAE	18.63	16.88	11.93	58.27	56.72	46.47	68.65
AAE	18.04	17.27	11.49	59.54	54.47	43.89	64.15
BiGAN	24.38	20.21	13.06	**71.53**	**67.18**	58.48	74.77
NAT	13.70	11.62	1.20	64.32	61.43	48.84	70.75
DIM(G)	11.32	6.34	4.95	42.03	30.82	28.09	51.36
DIM(L)	**33.8**	**34.5**	**30.7**	**71.82**	**67.22**	**61.61**	**75.62**

THANKS!

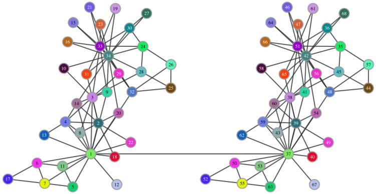
Acknowledgements:
Alex Fedorov (MRN), Sam Lavoie (MILA), Karan Grewal (U Toronto), Adam Trischler, and Yoshua Bengio
Found at: https://arxiv.org/abs/1808.06670
Github code: https://github.com/rdevon/DIM

. . . and we're back!

- DIM has many properties that are suitable to being leveraged in a graph setting.
- Namely, all we need is a *global summary*, and some *positive and negative patch representations*—nobody really cares how we got them!
- While DIM focused on image classification, and therefore required the summary to encode useful information of the input, for node classification we care about maximising mutual information between **different local parts of the input**.
  - In practice, both of these should happen *simultaneously*—a local-global objective is a scalable proxy for all-pairs local-local.

Capturing structural roles

- This becomes all the more important in graphs, where *structural roles* may be an excellent predictor (as argued in both struc2vec and GraphWave (Donnat *et al.*, KDD 2018).



- A local-global objective would allow every node to see *arbitrarily far in the graph*, looking for structural similarities to reinforce its local representation!

Roadmap for today

- A tale of many (random) walks;
- The (DIM) fairy godmother;
- **Happily ever (mutually) informative.**

Towards a graph method

- Finally, we will look at how to generalise the ideas from DIM into the graph domain.
- To-do list:
  - Obtaining **patch representations**;
  - Obtaining **global summaries**;
  - Obtaining **negative patch representations**;
  - **Discriminating** positive and negative patch-summary pairs.
- Most of these will come with absolutely no hassle (reusing standard ideas from graph neural networks)!

Representations and summaries

- We may obtain patch representations, $\vec{h}_i$, by using any graph convolutional **encoder**, $\mathcal{E}$, we'd like!
  - In our work, we use the **GCN** (Kipf & Welling, ICLR 2017) for transductive tasks and **GraphSAGE** (Hamilton *et al.*, NIPS 2017) for inductive tasks.

$$\mathcal{E}(\mathbf{X}, \mathbf{A}) = \sigma\left(\hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}\mathbf{\Theta}\right)$$

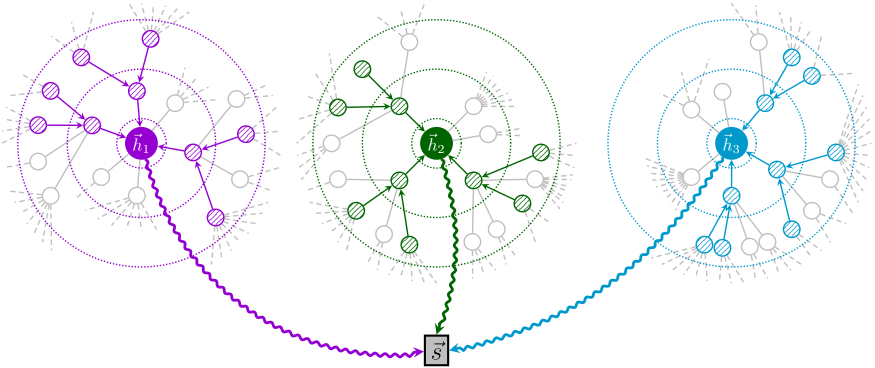
- Graph summaries, $\vec{s}$, are obtained by a **readout function**, $\mathcal{R}$.
  - We have attempted several popular readouts (such as set2vec), but simple averaging was found to work best.

$$\mathcal{R}(\mathbf{H}) = \frac{1}{N}\sum_{i=1}^{N}\vec{h}_i$$

Scaling to large graphs

- As graphs grow in size ($>$100,000 nodes), storing them in GPU memory in their entirety quickly becomes infeasible.
- This means we must resort to *subsampling* (computing patch representations **only** for a chosen batch of nodes at once), as done in GraphSAGE.
- We found that *summarising only the nodes in the batch* still works well!

Encoding and summarisation on large graphs



Obtaining negative samples

- DIM produces negative patch representations by simply using *another image* from the training set as a "fake" input.
- For *multi-graph* datasets (such as PPI), we are able to re-use this approach (+ dropout for more variability).
- However, most graph datasets are **single-graph**!

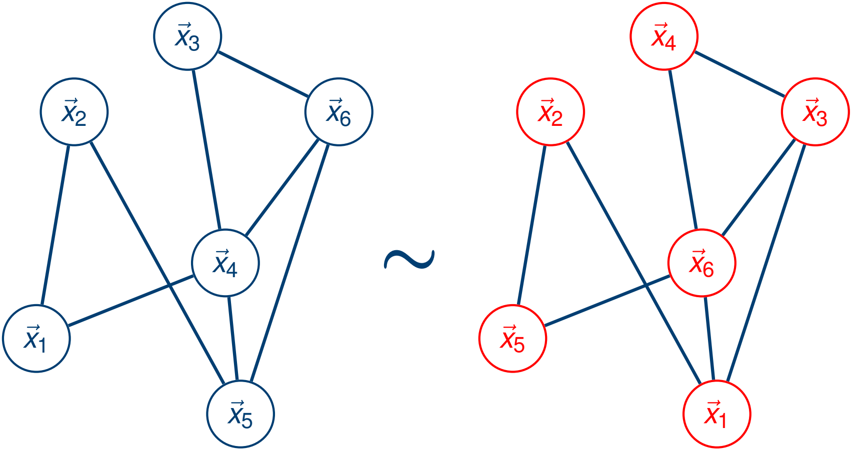
$\Longrightarrow$ Must specify a (stochastic) **corruption function**, $\mathcal{C}$, to obtain negative graphs $(\widetilde{\mathbf{X}}, \widetilde{\mathbf{A}})$ from the positive one.

- Can then re-use the encoder, $\mathcal{E}$, on the negative graph, to obtain negative patch representations, $\vec{\widetilde{h}}_i$.

Choice of corruption function

- We seek a negative graph that will be in some ways "comparable" to the original graph, while encoding mostly "nonsensical" patches.
- Here, we utilise node shuffling ($\widetilde{\mathbf{X}}$ obtained by row-wise shuffling of the feature matrix, $\mathbf{X}$), while keeping the adjacency matrix fixed ($\widetilde{\mathbf{A}} = \mathbf{A}$).
- This corruption has *many* desirable properties:
  - All input node features are *preserved*;
  - The adjacency matrix is *isomorphic* to the original one;
  - High likelihood of *useless* neighbourhoods.
  - Trivial to apply, and works **really** well! :)

Node shuffling example



Maximising mutual information

- We treat $(\vec{h}_i, \vec{s})$ as *positive*, and $(\widetilde{\vec{h}}_j, \vec{s})$ as *negative* examples.
- Just like in DIM, we use a **discriminator**, $\mathcal{D}$, which is a simple *bilinear binary classifier* between these two:

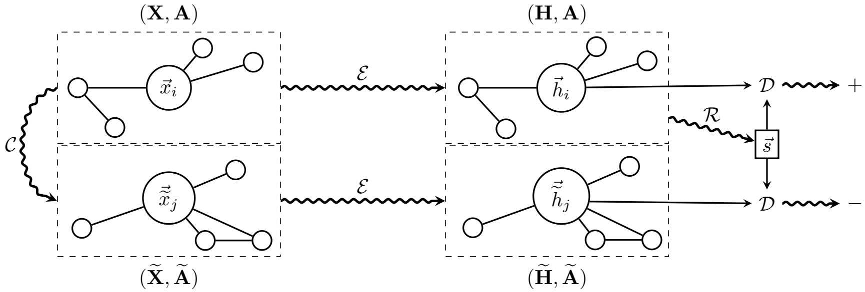
$$\mathcal{D}(\vec{h}_i, \vec{s}) = \sigma\left(\vec{h}_i^T \mathbf{W} \vec{s}\right)$$

  and optimising its *binary cross-entropy* implies maximising the mutual information based on the Jensen-Shannon divergence:

$$\mathcal{L} = \frac{1}{N+M}\left(\sum_{i=1}^{N} \mathbb{E}_{(\mathbf{X},\mathbf{A})}\left[\log \mathcal{D}\left(\vec{h}_i, \vec{s}\right)\right] + \sum_{j=1}^{M} \mathbb{E}_{(\widetilde{\mathbf{X}},\widetilde{\mathbf{A}})}\left[\log\left(1 - \mathcal{D}\left(\widetilde{\vec{h}}_j, \vec{s}\right)\right)\right]\right)$$

Deep Graph Infomax

We have arrived at **Deep Graph Infomax** (DGI)!



Quantitative evaluation: Datasets

- Verified the potential of the method in a variety of node classification benchmarks: transductive, inductive, large graphs, multi-graph.

Dataset	Task	Nodes	Edges	Features	Classes
Cora	Trans.	2,708	5,429	1,433	7
Citeseer	Trans.	3,327	4,732	3,703	6
Pubmed	Trans.	19,717	44,338	500	3
Reddit	Ind.	231,443	11,606,919	602	41
PPI	Ind.	56,944 (24 graphs)	818,716	50	121 (multilbl.)

- In all cases, the encoder was trained on the training set, and the embeddings are then used for simple linear classification (logistic regression) to evaluate their potential.

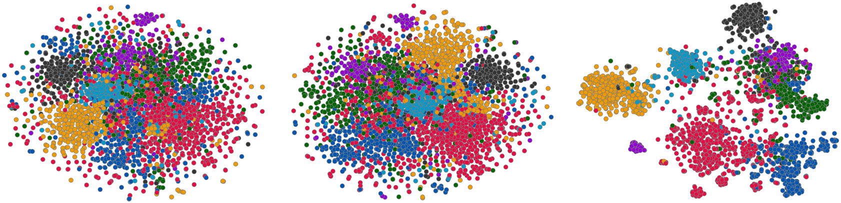
Quantitative results: Transductive

Data	Method	Cora	Citeseer	Pubmed
X	Raw features	47.9 ± 0.4%	49.3 ± 0.2%	69.1 ± 0.3%
A, **Y**	LP	68.0%	45.3%	63.0%
A	DeepWalk	67.2%	43.2%	65.3%
X, **A**	DW + fts.	70.7 ± 0.6%	51.4 ± 0.5%	74.3 ± 0.9%
X, **A**	Random-Init	69.3 ± 1.4%	61.9 ± 1.6%	69.6 ± 1.9%
X, **A**	**DGI** (ours)	**82.3** ± 0.6%	**71.8** ± 0.7%	**76.8** ± 0.6%
X, **A**, **Y**	GCN	81.5%	70.3%	79.0%
X, **A**, **Y**	Planetoid	75.7%	64.7%	77.2%

Quantitative results: Inductive

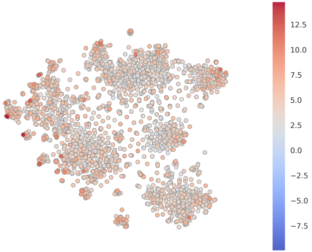
Data	Method	Reddit	PPI
$\mathbf{X}$	Raw features	0.585	0.422
$\mathbf{A}$	DeepWalk	0.324	—
$\mathbf{X}, \mathbf{A}$	DW + fts.	0.691	—
$\mathbf{X}, \mathbf{A}$	GraphSAGE-GCN	0.908	0.465
$\mathbf{X}, \mathbf{A}$	GraphSAGE-mean	0.897	0.486
$\mathbf{X}, \mathbf{A}$	GraphSAGE-LSTM	0.907	0.482
$\mathbf{X}, \mathbf{A}$	GraphSAGE-pool	0.892	0.502
$\mathbf{X}, \mathbf{A}$	Random-Init	0.933 ± 0.001	0.626 ± 0.002
$\mathbf{X}, \mathbf{A}$	**DGI** (ours)	$\mathbf{0.940} \pm 0.001$	$\mathbf{0.638} \pm 0.002$
$\mathbf{X}, \mathbf{A}, \mathbf{Y}$	FastGCN	0.937	—
$\mathbf{X}, \mathbf{A}, \mathbf{Y}$	Avg. pooling	0.958 ± 0.001	0.969 ± 0.002

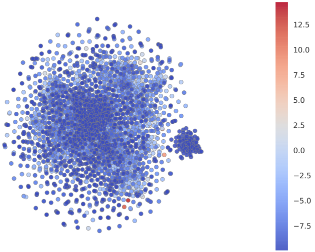
Qualitative results: “evolving” t-SNE



- Silhouette score of 0.234.
- Favourable to 0.158 for EP (Duran & Niepert, NIPS 2017).

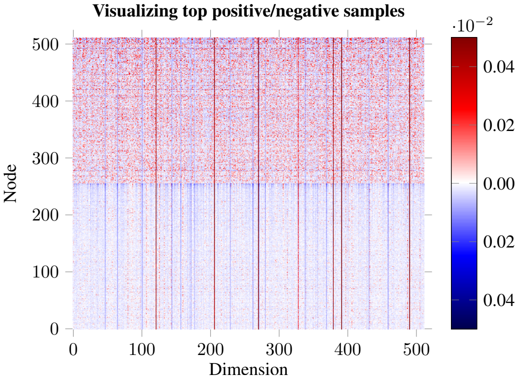
Qualitative results: DGI insights





- No particular structure found in negative graph (as expected!).
- Few “hot” nodes surrounded by many colder ones in t-SNE space—implies a specialisation of individual dimensions of the embeddings for different purposes!

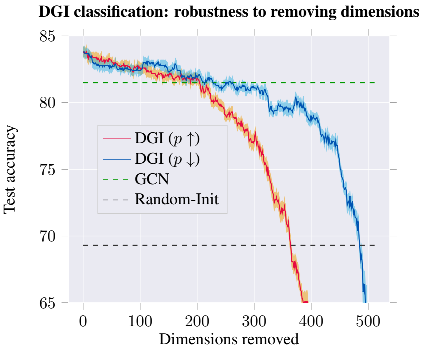
Qualitative results: DGI insights, *cont'd*

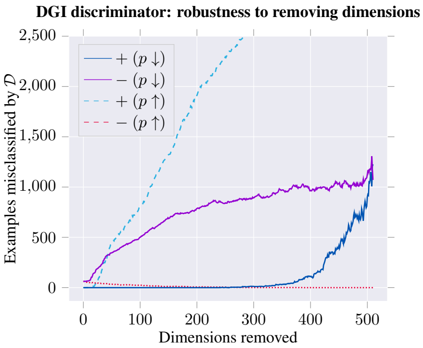


We can see specialisation!

If we **remove dimensions**, ordered by how *biased* they are. . .

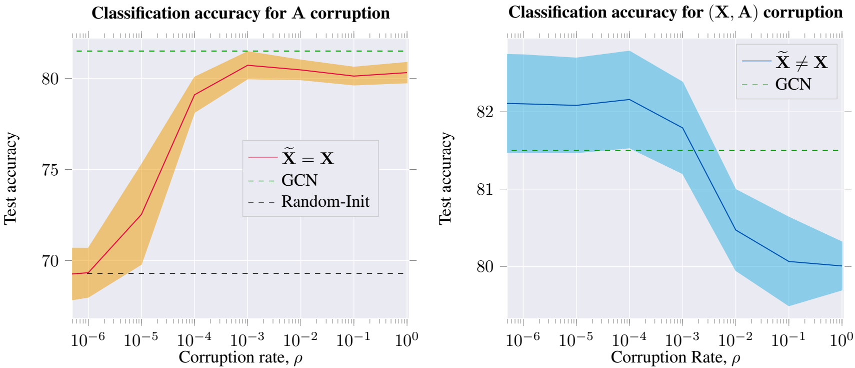
Qualitative results: DGI insights, concluded





Can remove **over half the dimensions** and stay competitive with the supervised GCN!

Qualitative results: Corruption function study



Direct corruption of **A** works too—but is harder (requires us not to get too dense), and more expensive to compute at every step.

Thank you!

Questions?

petar.velickovic@cst.cam.ac.uk

http://www.cst.cam.ac.uk/~pv273/

https://arxiv.org/abs/1809.10341