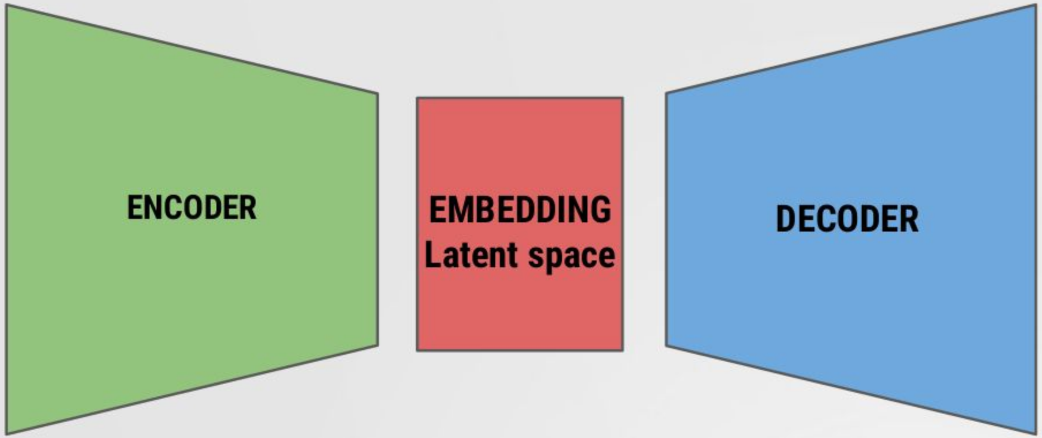
Adversarially regularized GAE (ARGA)
&
Adversarially regularized VGAE (ARVGA)

Gabriele Santin[1]

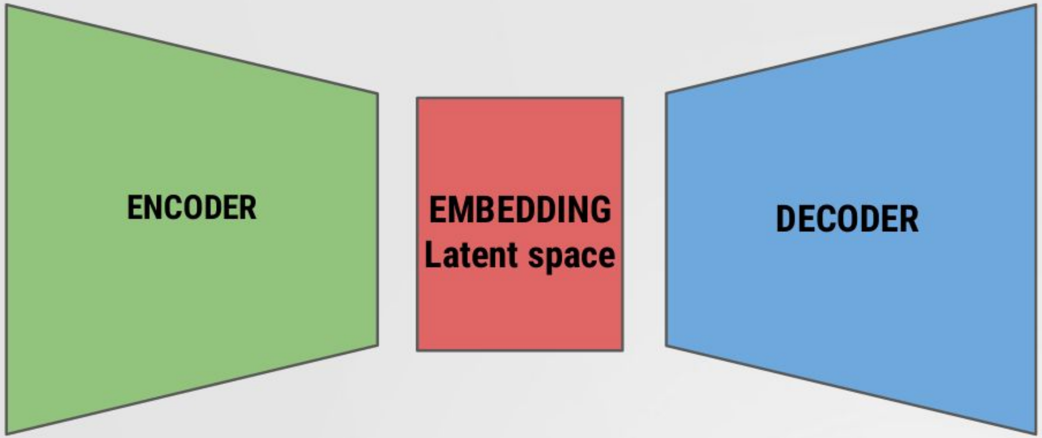
MobS[1] Lab, Fondazione Bruno Kessler,Trento, Italy

TABLE OF CONTENTS

01 Recap



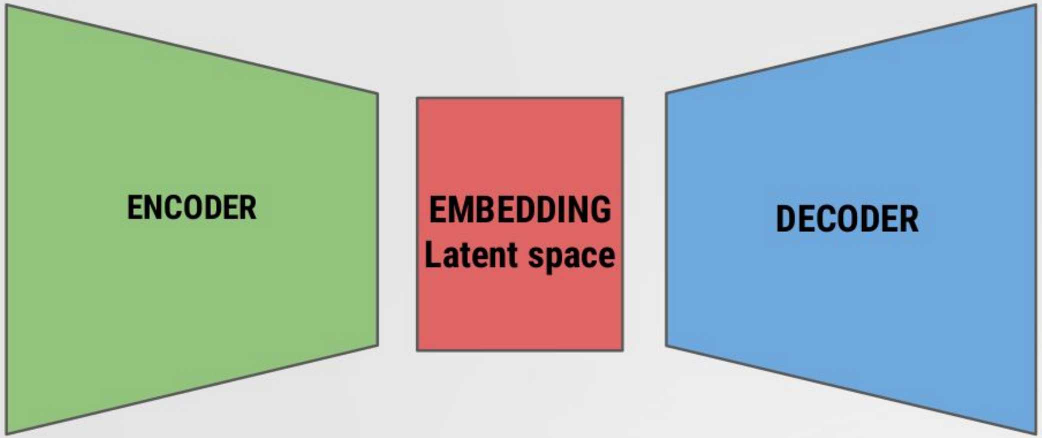
01 Recap



Input:
Graph with node features

- Adj. matrix A
- Data matrix X

01 Recap

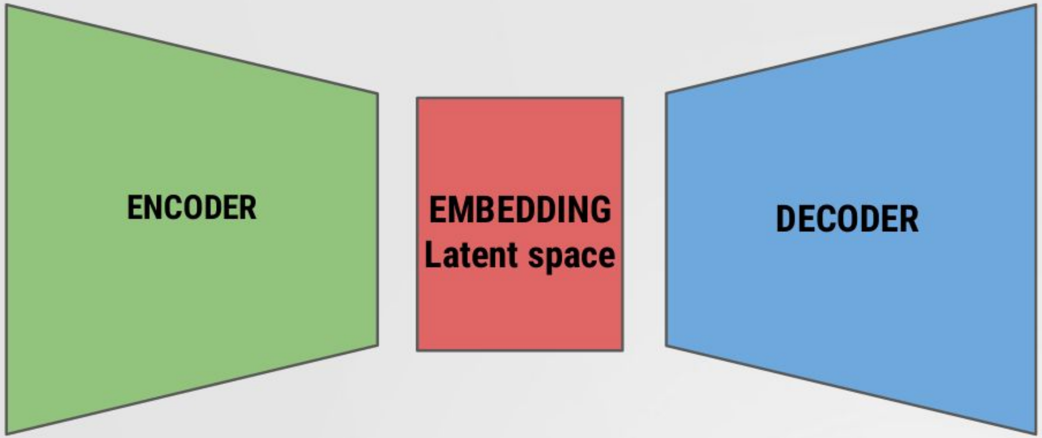


Input:
Graph with node features

- Adj. matrix A
- Data matrix X

Structural information

01 Recap



Input:
Graph with node features

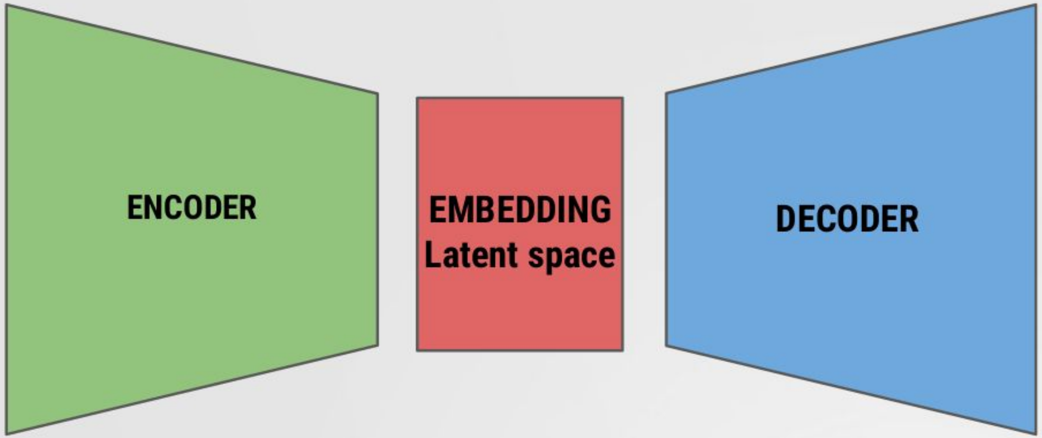
- Adj. matrix A
- Data matrix X

Structural information

Feature information

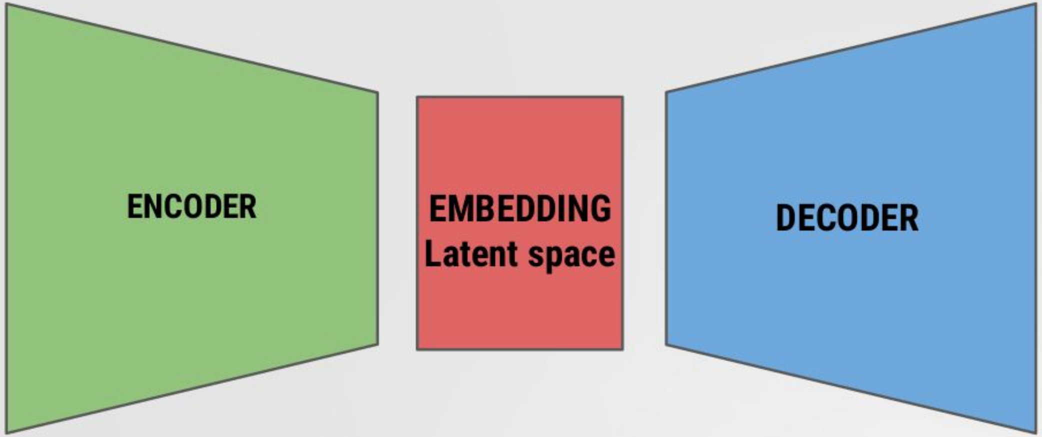
01 Recap

Input:
Graph with node features
- Adj. matrix A
- Data matrix X



Output:
Graph
- Approx. of A

01 Recap

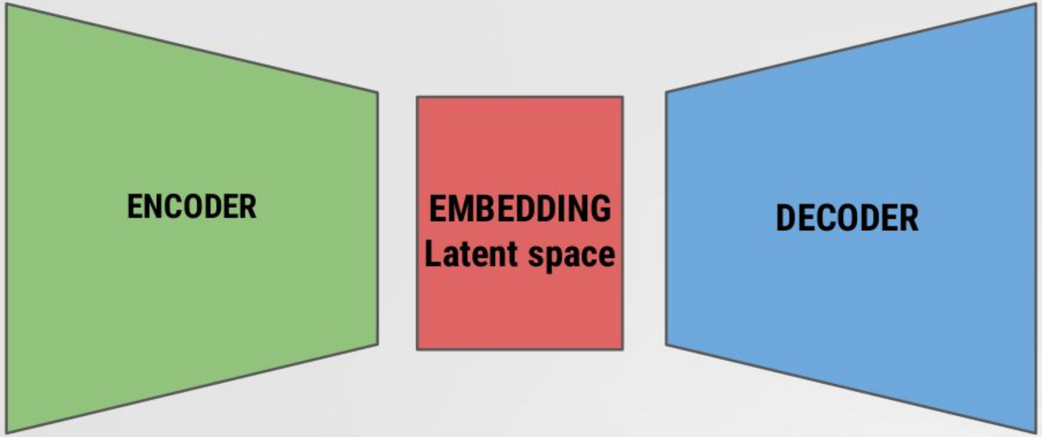


Input:
Graph with node features
- Adj. matrix A
- Data matrix X

Output:
Graph
- Approx. of A

Goal:
- Embedding
- Generation
- ...

01 Recap



Input:
Graph with node features
- Adj. matrix A
- Data matrix X

Output:
Graph
- Approx. of A

Goal:
- Embedding
- Generation
- ...

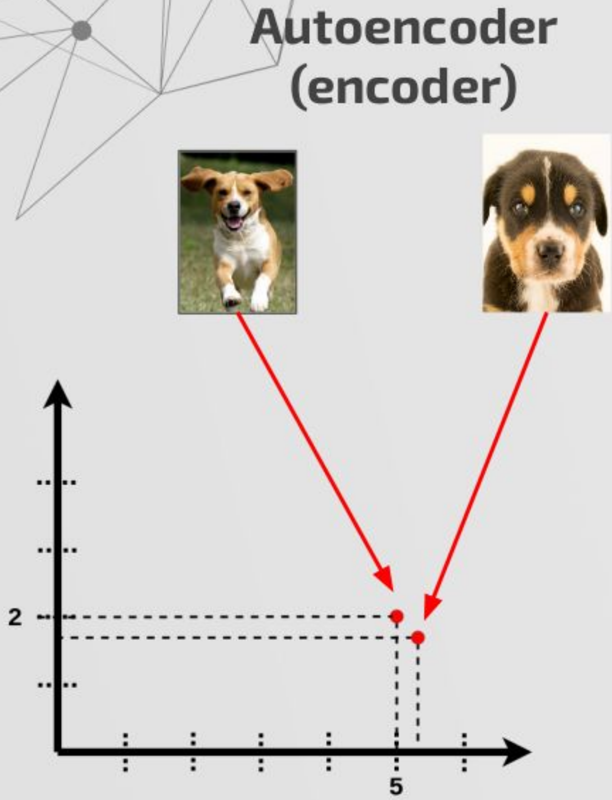
Continuous feature space

01 Recap

GAE vs VGAE:

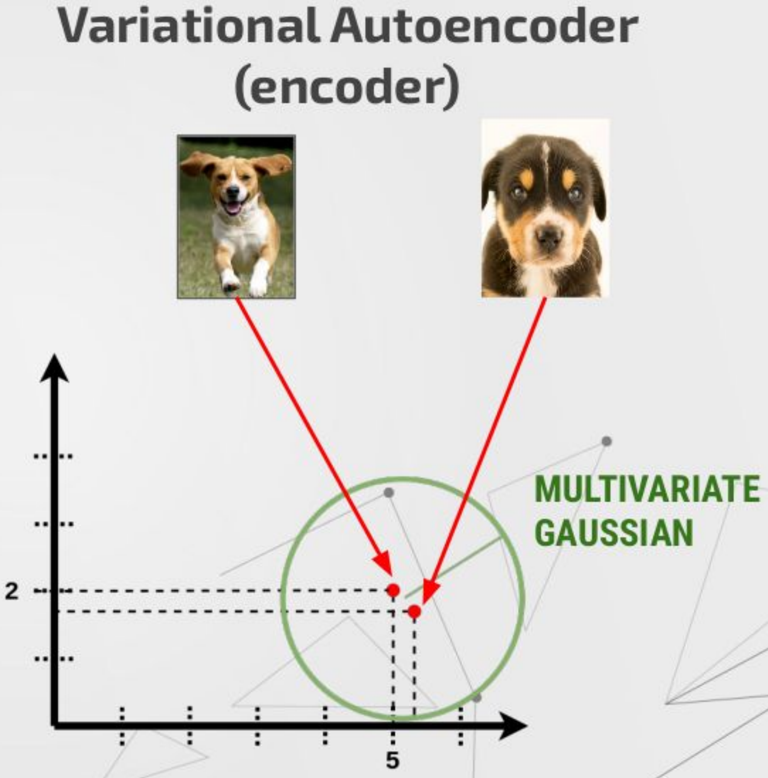
- Embedding on nodes
- Each nodes is mapped to its latent representation

01 Recap



GAE vs VGAE:

- Embedding on nodes
- Each nodes is mapped to its latent representation

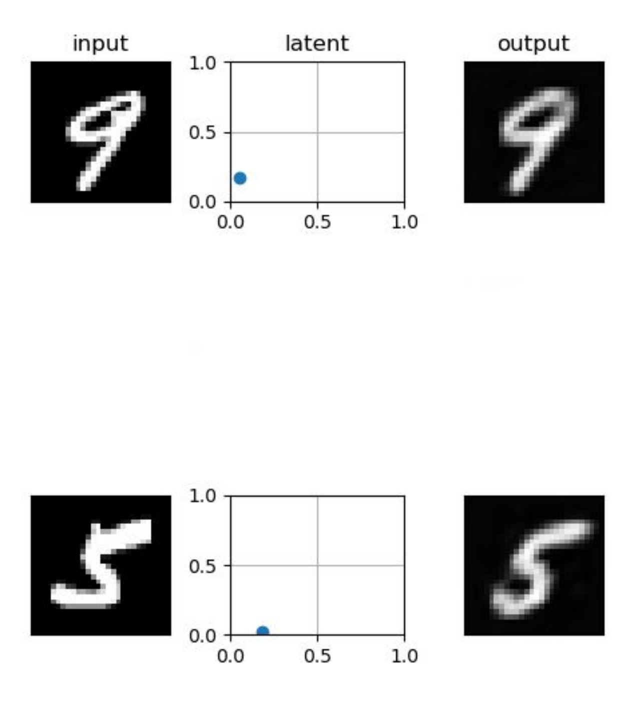


02 Motivation ARGA & ARVGA

Motivation:

The importance of the latent representation

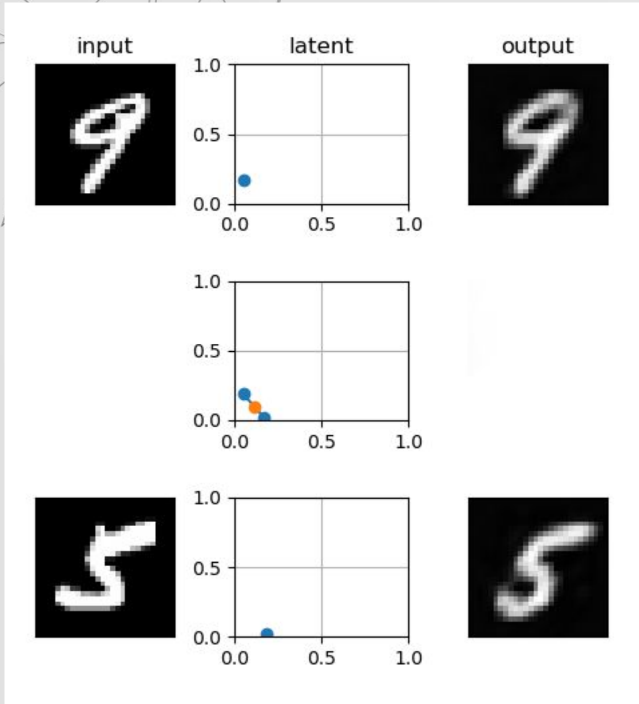
02 Motivation ARGA & ARVGA



Motivation:

The importance of the latent representation

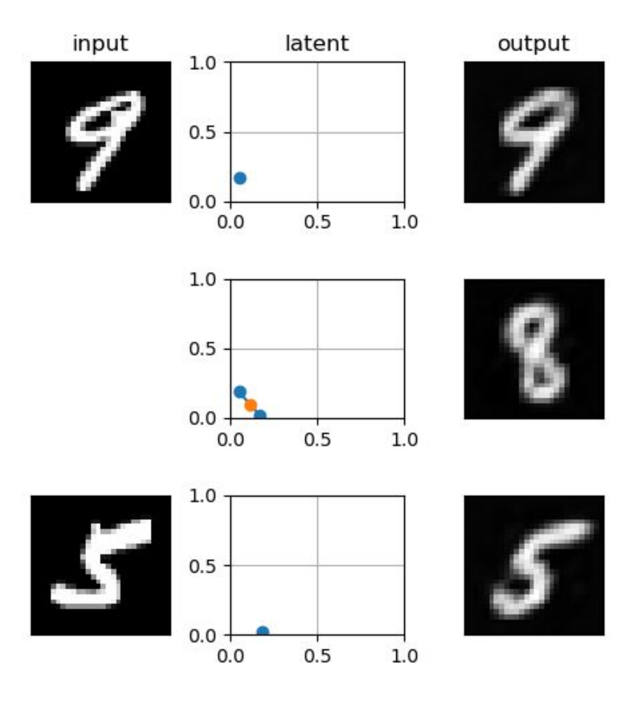
02 Motivation ARGA & ARVGA



Motivation:

The importance of the latent representation

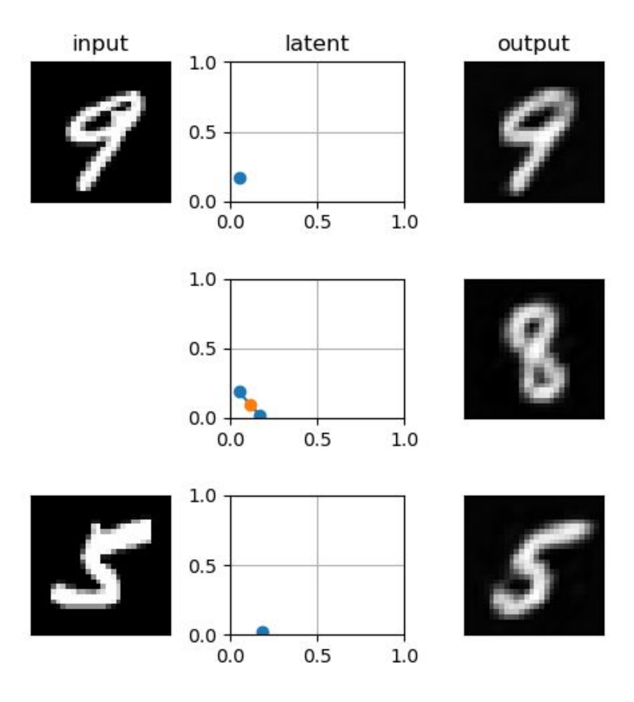
02 Motivation ARGA & ARVGA



Motivation:

The importance of the latent representation

02 Motivation ARGA & ARVGA

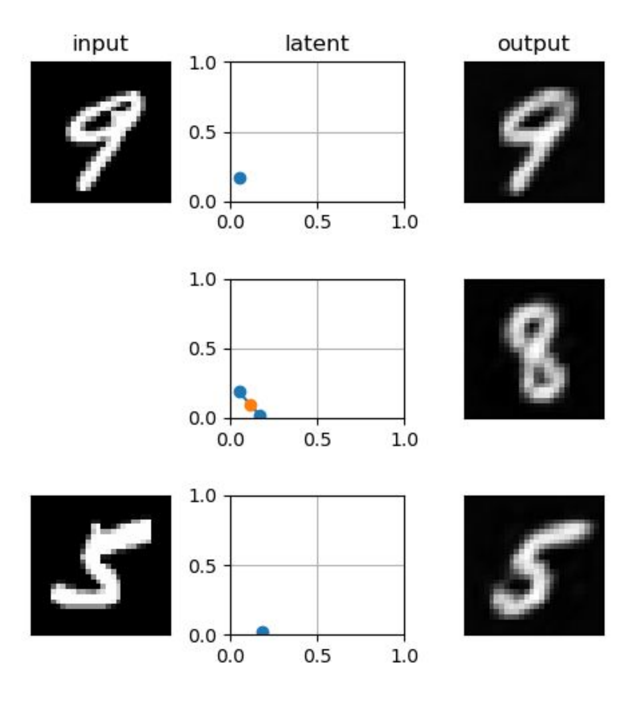


Motivation:
The importance of the latent representation

AE and GAE: only reconstruction loss

VAE and VGAE: regularize to have
continuous latent representation

02 Motivation ARGA & ARVGA



Motivation:

The importance of the latent representation

AE and GAE: only reconstruction loss

VAE and VGAE: regularize to have continuous latent representation

ARGA & ARVGA improve it

02 Motivation ARGA & ARVGA

Motivation:
The importance of the latent representation

Adversarially regularized graph autoencoder (ARGA)
Adversarially regularized variational graph autoencoder (ARVGA)

S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, and C. Zhang, *Adversarially regularized graph autoencoder for graph embedding.* in Proc. of IJCAI, 2018, pp. 2609–2615.

02 Motivation ARGA & ARVGA

Motivation:
The importance of the latent representation

Adversarially regularized graph autoencoder (ARGA)
Adversarially regularized variational graph autoencoder (ARVGA)

We have a look at
adversarial training

S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, and C. Zhang, *Adversarially regularized graph autoencoder for graph embedding.* in Proc. of IJCAI, 2018, pp. 2609–2615.

03 Ideas from adversarial models

Goal: generate fake objects (e.g. images) similar to real ones
Idea: play an adversarial game with two agents

I. Goodfellow et al., *Generative Adversarial Nets*. in Proc. of NIPS, 2014, pp. 2672--2680.

03 Ideas from adversarial models

Goal: generate fake objects (e.g. images) similar to real ones
Idea: play an adversarial game with two agents

↓

Generator: maps noise z to a fake object x
Discriminator: maps object x to probability of real/fake
Game: The generator tries to fool the discriminator
The discriminator tries to detect the fake objects

I. Goodfellow et al., *Generative Adversarial Nets*. in Proc. of NIPS, 2014, pp. 2672--2680.

03 Ideas from adversarial models

Goal: generate fake objects (e.g. images) similar to real ones
Idea: play an adversarial game with two agents

↓

Generator: maps noise z to a fake object x
Discriminator: maps object x to probability of real/fake
Game: The generator tries to fool the discriminator
The discriminator tries to detect the fake objects

$$\min_G \max_D \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

I. Goodfellow et al., *Generative Adversarial Nets*. in Proc. of NIPS, 2014, pp. 2672--2680.

03 Ideas from adversarial models

$$\min_G \max_D \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

I. Goodfellow et al., *Generative Adversarial Nets*. in Proc. of NIPS, 2014, pp. 2672--2680.

03 Ideas from adversarial models

$$\min_G \max_D \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

The **Discriminator** wants to **max:**

I. Goodfellow et al., *Generative Adversarial Nets*. in Proc. of NIPS, 2014, pp. 2672--2680.

03 Ideas from adversarial models

$$\min_G \max_D \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

The **Discriminator** wants to **max:**

- Recall that D(x) is in [0, 1]
- **First term**:
  → large if D(x) is close to 1
  → assign high probability to real objects

I. Goodfellow et al., *Generative Adversarial Nets*. in Proc. of NIPS, 2014, pp. 2672--2680.

03 Ideas from adversarial models

$$\min_G \max_D \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

The **Discriminator** wants to **max:**

- Recall that D(x) is in [0, 1]
- **First term**:
  → large if D(x) is close to 1
  → assign high probability to real objects
- **Second term**:
  → large if 1-D(G(z)) is close to 1
  → large if D(G(z)) is close to 0
  → assign low probability to fake objects

I. Goodfellow et al., *Generative Adversarial Nets*. in Proc. of NIPS, 2014, pp. 2672--2680.

03 Ideas from adversarial models

$$\min_G \max_D \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

The **Generator** wants to **min:**

I. Goodfellow et al., *Generative Adversarial Nets*. in Proc. of NIPS, 2014, pp. 2672--2680.

03 Ideas from adversarial models

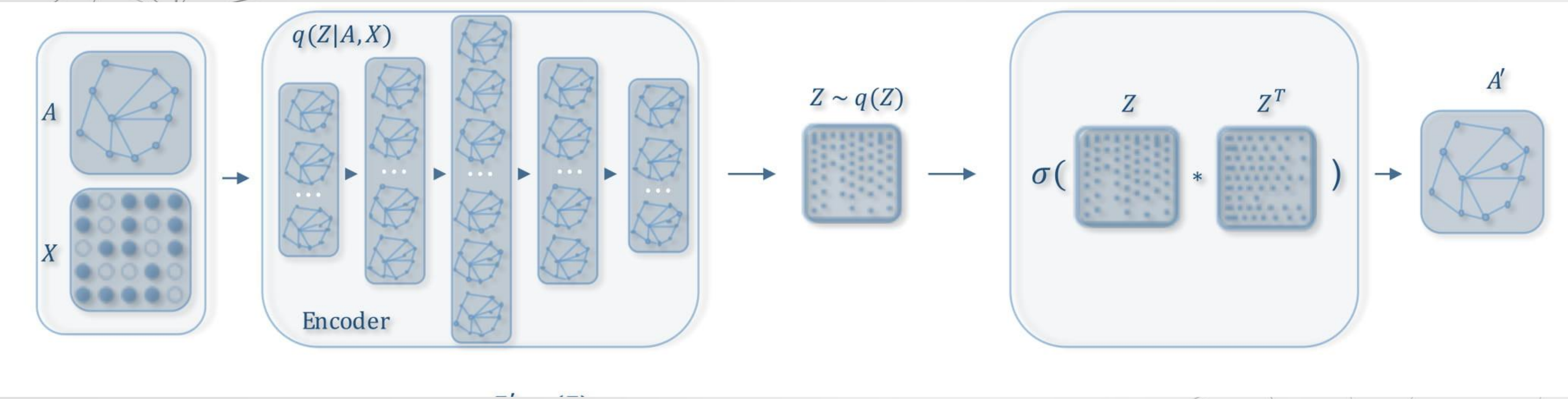
$$\min_{G} \max_{D} \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

The **Generator** wants to **min:**

- **Second term**:
  → small if 1-D(G(z)) is close to 0
  → small if D(G(z)) is close to 1
  → fool the discriminator into assigning high probability to fake objects

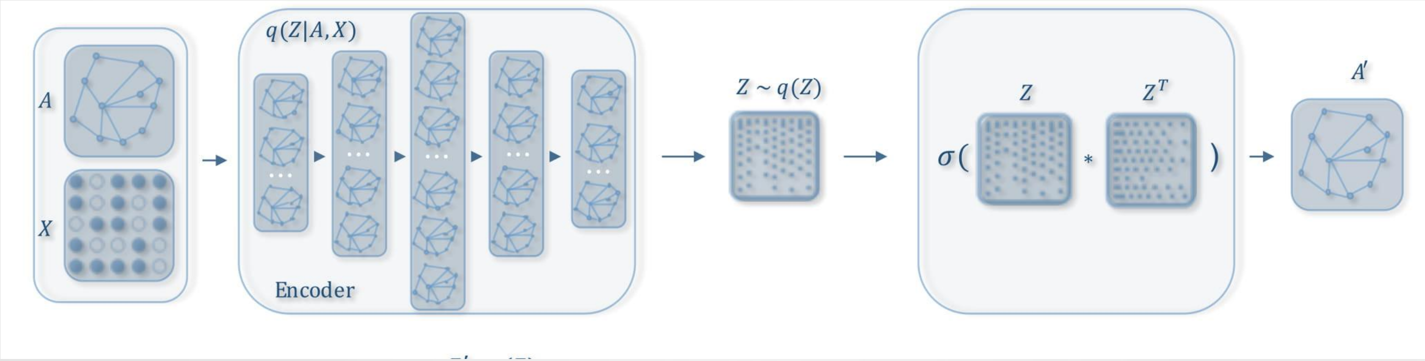
I. Goodfellow et al., *Generative Adversarial Nets*. in Proc. of NIPS, 2014, pp. 2672--2680.

04 ARGA & ARGVA



Picture from S. Pan et al.

04 ARGA & ARGVA



Architecture as in GAE/VGAE:

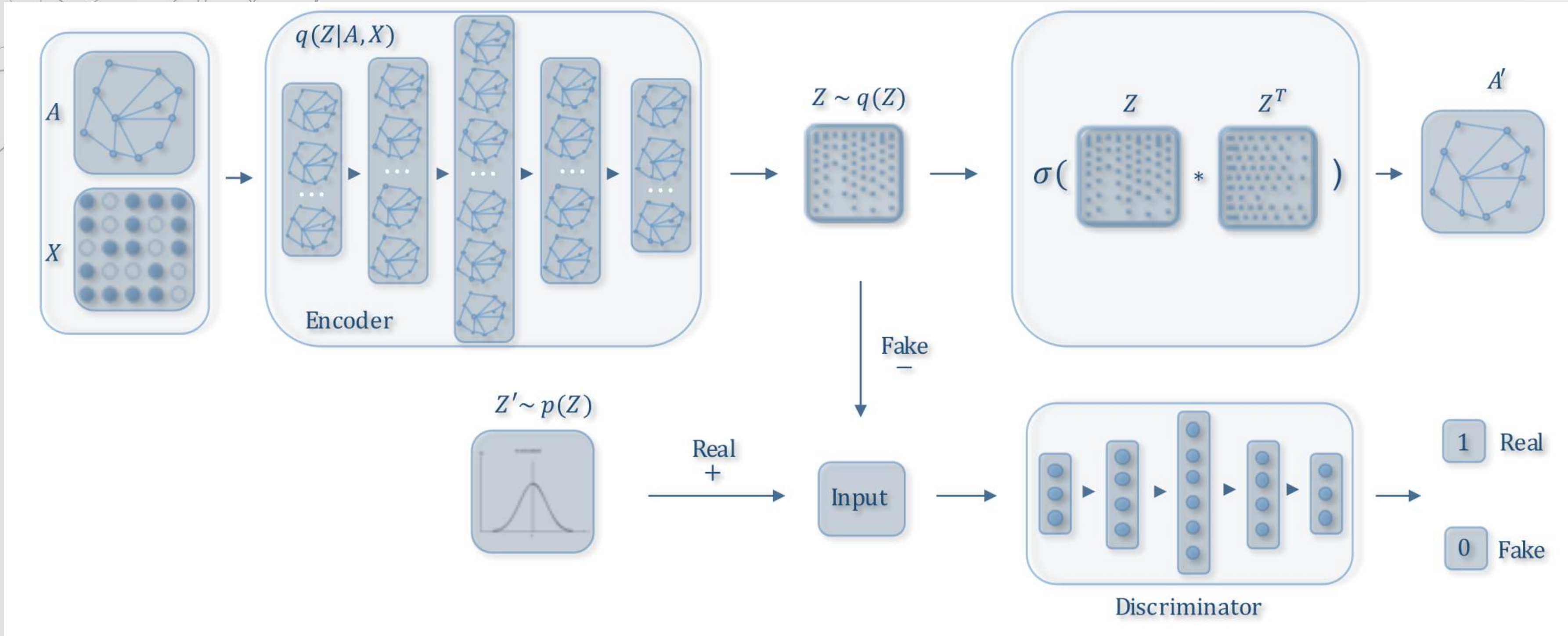
- **Encoder**: 2-layer GCN (with 2x for mean and logstd in VGAE)
- **Decoder**: inner product

→ Same **loss** as GAE/VGAE:

- **GAE**: reconstruction loss
- **VGAE**: rec. + KL regularization

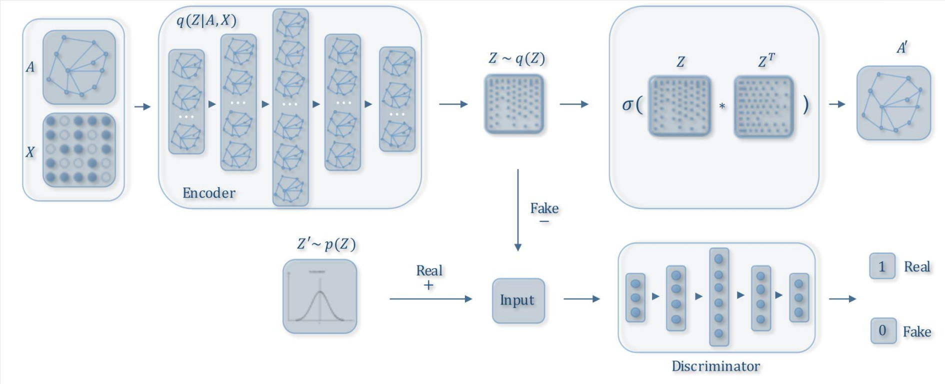
Picture from S. Pan et al.

04 ARGA & ARGVA



Picture from S. Pan et al.

04 ARGA & ARGVA

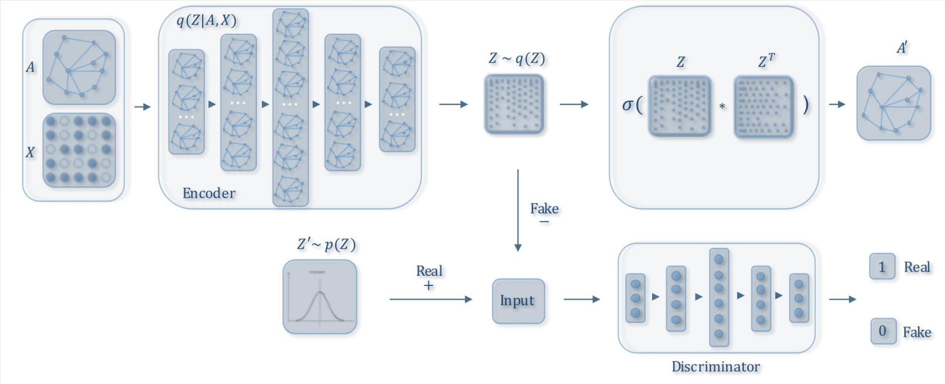


Architecture of the **discriminator:**

- Standard fully connected NN with 3 layers

Picture from S. Pan et al.

04 ARGA & ARGVA



Architecture of the **discriminator:**

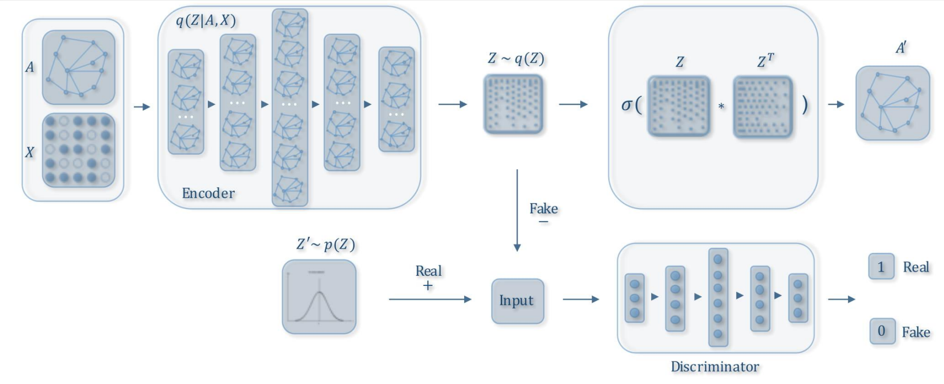
- Standard fully connected NN with 3 layers

Working on the latent space

→ continuous values!

Picture from S. Pan et al.

04 ARGA & ARGVA



Architecture of the **discriminator:**

- Standard fully connected NN with 3 layers

Working on the latent space

→ continuous values!

→ Adversarial **loss**:

Real: samples from N(0, 1)

Fake: samples from the latent encoding

$$\min_{\mathcal{G}} \max_{\mathcal{D}} \mathbb{E}_{\mathbf{z}\sim p_z}[\log \mathcal{D}(\mathbf{Z})] + \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}[\log(1-\mathcal{D}(\mathcal{G}(\mathbf{X},\mathbf{A})))]$$

Picture from S. Pan et al.

04 ARGA & ARGVA

Algorithm 1 Adversarially Regularized Graph Embedding

Require:
$\mathbf{G} = \{\mathbf{V}, \mathbf{E}, \mathbf{X}\}$: a Graph with links and features;
T: the number of iterations;
K: the number of steps for iterating discriminator;
d: the dimension of the latent variable

Ensure: $\mathbf{Z} \in \mathbb{R}^{n \times d}$

```
for iterator = 1,2,3, ······ , T do
    Generate latent variables matrix Z through Eq.(4);
    Update the graph autoencoder with its stochastic gradient by
    Eq. (10) for ARGA or Eq. (11) for ARVGA;
end for
return Z ∈ R^{n×d}
```

This is: Z = E(X, A)

Picture from S. Pan et al.

04 ARGA & ARGVA

Algorithm 1 Adversarially Regularized Graph Embedding

Require:
$\mathbf{G} = \{\mathbf{V}, \mathbf{E}, \mathbf{X}\}$: a Graph with links and features;
T: the number of iterations;
K: the number of steps for iterating discriminator;
d: the dimension of the latent variable

Ensure: $\mathbf{Z} \in \mathbb{R}^{n \times d}$

1: **for** iterator = 1,2,3, $\cdots\cdots$, T **do**
2: Generate latent variables matrix $\mathbf{Z}$ through Eq.(4);
3:
4:
5:
6:
7: Update the graph autoencoder with its stochastic gradient by Eq. (10) for ARGA or Eq. (11) for ARVGA;
end for
8: **return** $\mathbf{Z} \in \mathbb{R}^{n \times d}$

This is: Z = E(X, A)

These are the usual GAE/VGAE losses

Picture from S. Pan et al.

04 ARGA & ARGVA

Algorithm 1 Adversarially Regularized Graph Embedding

Require:
$\mathbf{G} = \{\mathbf{V}, \mathbf{E}, \mathbf{X}\}$: a Graph with links and features;
T: the number of iterations;
K: the number of steps for iterating discriminator;
d: the dimension of the latent variable
Ensure: $\mathbf{Z} \in \mathbb{R}^{n \times d}$

for iterator = 1,2,3, $\cdots\cdots$, T **do**
  Generate latent variables matrix $\mathbf{Z}$ through Eq.(4);
  for k = 1,2, $\cdots\cdots$, K **do**
    Sample m entities $\{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}\}$ from latent matrix $\mathbf{Z}$
    Sample m entities $\{\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(m)}\}$ from the prior distribution p_z
    Update the discriminator with its stochastic gradient:

$$\bigtriangledown \frac{1}{m} \sum_{i=1}^{m} [\log \mathcal{D}(\mathbf{a}^{i}) + \log (1 - \mathcal{D}(\mathbf{z}^{(i)}))]$$

  end for
  Update the graph autoencoder with its stochastic gradient by Eq. (10) for ARGA or Eq. (11) for ARVGA;
end for
return $\mathbf{Z} \in \mathbb{R}^{n \times d}$

K training loops of the discriminator

Picture from S. Pan et al.

04 ARGA & ARGVA

Algorithm 1 Adversarially Regularized Graph Embedding

Require:

$\mathbf{G} = \{\mathbf{V}, \mathbf{E}, \mathbf{X}\}$: a Graph with links and features;
T: the number of iterations;
K: the number of steps for iterating discriminator;
d: the dimension of the latent variable

Ensure: $\mathbf{Z} \in \mathbb{R}^{n \times d}$

1: **for** iterator = 1,2,3, $\cdots\cdots$, T **do**
2: Generate latent variables matrix $\mathbf{Z}$ through Eq.(4);
3: **for** k = 1,2, $\cdots\cdots$, K **do**
4: Sample m entities $\{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}\}$ from latent matrix $\mathbf{Z}$
5: Sample m entities $\{\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(m)}\}$ from the prior distribution p_z
6: Update the discriminator with its stochastic gradient:

$$\triangledown \frac{1}{m} \sum_{i=1}^{m} [\log \mathcal{D}(\mathbf{a}^i) + \log(1 - \mathcal{D}(\mathbf{z}^{(i)}))]$$

end for
7: Update the graph autoencoder with its stochastic gradient by Eq. (10) for ARGA or Eq. (11) for ARVGA;
end for
8: **return** $\mathbf{Z} \in \mathbb{R}^{n \times d}$

K training loops of the discriminator

Sample fake gaussians

Picture from S. Pan et al.

04 ARGA & ARGVA

Algorithm 1 Adversarially Regularized Graph Embedding

Require:
$\mathbf{G} = \{\mathbf{V}, \mathbf{E}, \mathbf{X}\}$: a Graph with links and features;
T: the number of iterations;
K: the number of steps for iterating discriminator;
d: the dimension of the latent variable
Ensure: $\mathbf{Z} \in \mathbb{R}^{n \times d}$

```
for iterator = 1,2,3, ······ , T do
    Generate latent variables matrix Z through Eq.(4);
    for k = 1,2, ······ , K do
        Sample m entities {z^(1), ..., z^(m)} from latent matrix Z
        Sample m entities {a^(1), ..., a^(m)} from the prior distribution p_z
        Update the discriminator with its stochastic gradient:
```

$$\bigtriangledown \frac{1}{m} \sum_{i=1}^{m} [\log \mathcal{D}(\mathbf{a}^{i}) + \log (1 - \mathcal{D}(\mathbf{z}^{(i)}))]$$

```
    end for
    Update the graph autoencoder with its stochastic gradient by
    Eq. (10) for ARGA or Eq. (11) for ARVGA;
end for
return Z ∈ R^{n×d}
```

K training loops of the discriminator

Sample fake gaussians

Sample true gaussians

Picture from S. Pan et al.

04 ARGA & ARGVA

Algorithm 1 Adversarially Regularized Graph Embedding

Require:
$\mathbf{G} = \{\mathbf{V}, \mathbf{E}, \mathbf{X}\}$: a Graph with links and features;
T: the number of iterations;
K: the number of steps for iterating discriminator;
d: the dimension of the latent variable

Ensure: $\mathbf{Z} \in \mathbb{R}^{n \times d}$

1: **for** iterator = 1,2,3, $\cdots\cdots$, T **do**
2: Generate latent variables matrix $\mathbf{Z}$ through Eq.(4);
3: **for** k = 1,2, $\cdots\cdots$, K **do**
4: Sample m entities $\{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}\}$ from latent matrix $\mathbf{Z}$
5: Sample m entities $\{\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(m)}\}$ from the prior distribution p_z
6: Update the discriminator with its stochastic gradient:

$$\bigtriangledown \frac{1}{m} \sum_{i=1}^{m} [\log \mathcal{D}(\mathbf{a}^{i}) + \log (1 - \mathcal{D}(\mathbf{z}^{(i)}))]$$

end for
7: Update the graph autoencoder with its stochastic gradient by Eq. (10) for ARGA or Eq. (11) for ARVGA;
end for
8: **return** $\mathbf{Z} \in \mathbb{R}^{n \times d}$

K training loops of the discriminator

Sample fake gaussians

Sample true gaussians

Update the discriminator

Picture from S. Pan et al.

04 ARGA & ARGVA

Algorithm 1 Adversarially Regularized Graph Embedding

Require:

$\mathbf{G} = \{\mathbf{V}, \mathbf{E}, \mathbf{X}\}$: a Graph with links and features;
T: the number of iterations;
K: the number of steps for iterating discriminator;
d: the dimension of the latent variable

Ensure: $\mathbf{Z} \in \mathbb{R}^{n \times d}$

```
for iterator = 1,2,3, ······, T do
    Generate latent variables matrix Z through Eq.(4);
    for k = 1,2, ······, K do
        Sample m entities {z^(1), ..., z^(m)} from latent matrix Z
        Sample m entities {a^(1), ..., a^(m)} from the prior distribution p_z
        Update the discriminator with its stochastic gradient:
```

$$\bigtriangledown \frac{1}{m} \sum_{i=1}^{m} [\log \mathcal{D}(\mathbf{a}^i) + \log (1 - \mathcal{D}(\mathbf{z}^{(i)}))]$$

```
    end for
    Update the graph autoencoder with its stochastic gradient by
    Eq. (10) for ARGA or Eq. (11) for ARVGA;
end for
return Z ∈ R^{n×d}
```

K training loops of the discriminator

Sample fake gaussians

Sample true gaussians

Update the discriminator

Missing: update the encoder (written in the text)

Picture from S. Pan et al.

05 ARGA & ARGVA

CLASS **ARGA** (encoder, discriminator, decoder=None) [source]

The Adversarially Regularized Graph Auto-Encoder model from the "Adversarially Regularized Graph Autoencoder for Graph Embedding" paper. paper.

PARAMETERS

- **encoder** (*Module*) – The encoder module.
- **discriminator** (*Module*) – The discriminator module.
- **decoder** (*Module, optional*) – The decoder module. If set to `None`, will default to the `torch_geometric.nn.models.InnerProductDecoder`. (default: `None`)

`discriminator_loss ( z )` [source]

Computes the loss of the discriminator.

PARAMETERS

z (*Tensor*) – The latent space $\mathbf{Z}$.

`reg_loss ( z )` [source]

Computes the regularization loss of the encoder.

PARAMETERS

z (*Tensor*) – The latent space $\mathbf{Z}$.

`reset_parameters ( )` [source]

05 ARGA & ARGVA

CLASS **ARGA** (encoder, discriminator, decoder=None) [source]

The Adversarially Regularized Graph Auto-Encoder model from the "Adversarially Regularized Graph Autoencoder for Graph Embedding" paper. paper.

PARAMETERS

- **encoder** (*Module*) – The encoder module.
- **discriminator** (*Module*) – The discriminator module.
- **decoder** (*Module, optional*) – The decoder module. If set to `None`, will default to the `torch_geometric.nn.models.InnerProductDecoder`. (default: `None`)

discriminator_loss (z) [source]

Computes the loss of the discriminator.

PARAMETERS

z (*Tensor*) – The latent space $\mathbf{Z}$.

reg_loss (z) [source]

Computes the regularization loss of the encoder.

PARAMETERS

z (*Tensor*) – The latent space $\mathbf{Z}$.

reset_parameters ( ) [source]

```
class ARGA(GAE):
```

decode (*args, **kwargs) [source]

Runs the decoder and computes edge probabilities.

encode (*args, **kwargs) [source]

Runs the encoder and computes node-wise latent variables.

recon_loss (z, pos_edge_index, neg_edge_index=None) [source]

Given latent variables `z`, computes the binary cross entropy loss for positive edges `pos_edge_index` and negative sampled edges.

05 ARGA & ARGVA

CLASS **ARGVA** (encoder, discriminator, decoder=None) [source]

The Adversarially Regularized Variational Graph Auto-Encoder model from the "Adversarially Regularized Graph Autoencoder for Graph Embedding" paper. paper.

PARAMETERS

- **encoder** (*Module*) – The encoder module to compute μ and $\log \sigma^2$.
- **discriminator** (*Module*) – The discriminator module.
- **decoder** (*Module, optional*) – The decoder module. If set to `None`, will default to the `torch_geometric.nn.models.InnerProductDecoder`. (default: `None`)

encode (*args, **kwargs) [source]

kl_loss (mu=None, logstd=None) [source]

reparametrize (mu, logstd) [source]

05 ARGA & ARGVA

CLASS **ARGVA** (**encoder**, **discriminator**, **decoder**=None) [source]

The Adversarially Regularized Variational Graph Auto-Encoder model from the "Adversarially Regularized Graph Autoencoder for Graph Embedding" paper. paper.

PARAMETERS

- **encoder** (*Module*) – The encoder module to compute μ and $\log \sigma^2$.
- **discriminator** (*Module*) – The discriminator module.
- **decoder** (*Module, optional*) – The decoder module. If set to `None`, will default to th
`torch_geometric.nn.models.InnerProductDecoder`. (default: `None`)

encode (*args, **kwargs) [source]

kl_loss (**mu**=None, **logstd**=None) [source]

reparametrize (**mu**, **logstd**) [source]

```
class ARGVA(ARGA):
```

```
def __init__(self, encoder, discriminator, decoder=None):
    super(ARGVA, self).__init__(encoder, discriminator, decoder)
    self.VGAE = VGAE(encoder, decoder)
```

05 ARGA & ARGVA

CLASS **ARGVA** (**encoder**, **discriminator**, **decoder**=None) [source]

The Adversarially Regularized Variational Graph Auto-Encoder model from the "Adversarially Regularized Graph Autoencoder for Graph Embedding" paper. paper.

PARAMETERS

- **encoder** (*Module*) – The encoder module to compute μ and $\log \sigma^2$.
- **discriminator** (*Module*) – The discriminator module.
- **decoder** (*Module, optional*) – The decoder module. If set to `None`, will default to th`torch_geometric.nn.models.InnerProductDecoder`. (default: `None`)

encode (*args, **kwargs) [source]

kl_loss (**mu**=None, **logstd**=None) [source]

reparametrize (**mu**, **logstd**) [source]

```
class ARGVA(ARGA):
```

```
def __init__(self, encoder, discriminator, decoder=None):
    super(ARGVA, self).__init__(encoder, discriminator, decoder)
    self.VGAE = VGAE(encoder, decoder)
```

```
class VGAE(GAE):
```

encode (*args, **kwargs) [source]

kl_loss (**mu**=None, **logstd**=None) [source]

Computes the KL loss, either for the passed arguments `mu` and `logstd`, or based on latent variables from last encoding.

05 ARGA & ARGVA

Jupyter Notebook