CS 224W Project Final Report: Learning Fair Graph Representations

Kevin Fry ID: 05982074

10 December 2017

1 Abstract

In this paper we present the Variational Fair Graph Autoencoder (VFGAE) as a model to learn feature representations on graphs that are invariant to a specified *nuisance* variable **s**. This model is brings together previous work on the Variational Graph Autoencoder proposed by Kipf and Welling, and the Variational Fair Autoencoder proposed by Louizos et al. To take into account the graph structure, we modify the traditional Variational Autoencoder to use a graph convolutional encoder network as in Kipf and Welling. Additionally, as in Louizos et al, we use a loss function and priors that promote independence between the latent feature representation **z** and the nuisance variable **s** and further encourage independence through a "Maximum Mean Discrepancy" metric. We validate our model by demonstrating its ability fair representations on several citation datasets.

2 Introduction

"Feature Learning" describes a set of techniques that allow for automatic identification for features, or representations, of data that are most useful for a particular learning task. This usually means determining the latent, or informative, features in the data as well as the uninformative or detrimental features, known as *nuisance* variables. Variational Autoencoders (VAE) use feature learning to determine latent features **z** of some variable **X** that are useful in learning the distribution $p(\mathbf{X})$ (we will go into more detail later on). These neural networks have been to produce realistic handwritten numbers by training on the MNIST dataset, as well as be effective in generating general images.

However, there are often some factors of variation we would like our latent representations to be invariant to; that is we would like to remove as much information about **s** from **z**. The resulting representations are called fair. This is particularly useful when **s** is detrimental to the learning task or is a sensitive variable that we do not wish to influence our predictions. This problem has been addressed previously for a binary prediction task and single multi-class nuisance variable characterized by the softmax function (Zemel et al. 2013); and more recently with the Variational Fair Autoencoder (VFAE), which uses adapted VAEs (Louizos et al 2016).

This paper aims to extend this framework for learning fair representations to graphs. Recent work has extended the VAE to be able to generate graphs with the Variational Graph Autoencoder (VGAE) (Kipf and Welling 2016). Thus we combine the graph convolutional architecture of the VGAE with the independence encouraging choices of priors and penalty term of the VFAE to produce what we are calling the Variational Fair Graph Autoencoder (VFGAE).

3 Background

3.1 Variational Autoencoders

We begin with the variational autoencoder as the rest of the network architecture is built upon its foundation. Note that we will view this from a generative modeling perspective, as that was the main motivation behind VAE's, even though it is not our aim. Discussing VAEs demands at least a cursory overview of autoencoders. An autoencoder is two stacked neural networks. The first network, known as the encoding, or recognition, network takes some $\mathbf{X}$ as input and outputs some latent representation of $\mathbf{X}$, or encoding, $\mathbf{z}$. The second network, called the decoding, or generative, network then takes that $\mathbf{z}$ as input and tries to reconstruct $\mathbf{X}$. In this way the two networks learn a representation $\mathbf{z}$ that is an efficient encoding of $\mathbf{X}$. The autoencoder is interesting on its own, but can only recreate examples its been given, not generate new ones it has not seen. VAEs take a Bayesian twist on the autoencoder to allow us to do exactly that (Kingma and Welling 2014).

Instead of learning how to reconstruct $\mathbf{x}$ from $\mathbf{z}$, the VAE assumes a marginal distribution $p(\mathbf{z})$ and attempts to learn the distribution $p(\mathbf{X} \mid \mathbf{z})$. The recognition network of a VAE learns a conditional distribution of $\mathbf{z}$ given $\mathbf{X}$ denoted $q(\mathbf{z} \mid \mathbf{X})$. We then sample a $\mathbf{z}$ from this distribution and feed it in as input to our generative network. As the recognition network learned a distribution of $\mathbf{z} \mid \mathbf{X}$, the generative network similarly learns the distribution $p(\mathbf{X} \mid \mathbf{z})$. Then when we wish to sample a new $\mathbf{X}$, we can simply sample a new $\mathbf{z}$ from $p(\mathbf{z})$, and put it through our generative network to get $p(\mathbf{X} \mid \mathbf{z})$ and sample a new $\mathbf{X}$ from that. The stacked network is trained to minimize the following objective:

$$\max \mathbb{E}_{\mathbf{z} \sim q}[\log p(\mathbf{X} \mid \mathbf{z})] - D[q(\mathbf{z} \mid \mathbf{X}) \| p(\mathbf{z})]$$

where D is the KL-divergence and the maximization is with respect to the neural network weights. Intuitively, the first term incentivizes the network to learn a $\mathbf{z}$ that retains most of the information of the $\mathbf{X}$ it was created from; and the second term is to motivate the network to learn a distribution q that does not contain any more information about $\mathbf{X}$ than is necessary.

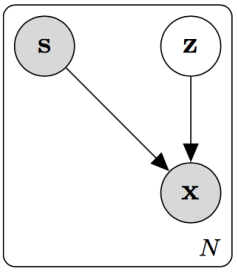
3.2 Variational Graph Autoencoders

Kipf extended the variational autoencoder framework to one for generative modeling on graphs, VGAEs. VGAEs use the same encoder-decoder setup, but modifies them to appropriately capture graph structure. The model takes as input both a matrix of node features $\mathbf{X}$ and the adjacency matrix $\mathbf{A}$, and assuming a Gaussian prior for the latent variable $\mathbf{Z}$. The VGAE now attempts to learn the distribution of $\mathbf{A}$ for a sample of graphs drawn from a distribution. Specifically, the VGAE differs from the VAE as follows:

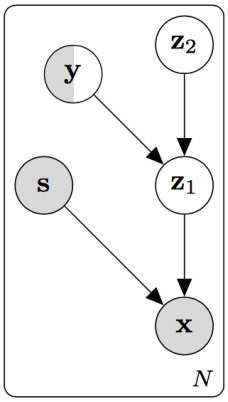
- The encoder now learn the distribution $q(\mathbf{Z} \mid \mathbf{A}, \mathbf{X})$ and the layers are now of the form:

  $$\tau(\tilde{\mathbf{A}}\mathbf{X}\mathbf{W}_0)$$

  where $\tilde{\mathbf{A}} = D^{-\frac{1}{2}}\mathbf{A}D^{-\frac{1}{2}}$ is the symmetrically normalized adjacency matrix of the graph, $\mathbf{W}_0$ is the weight matrix for that layer of the network, and τ is some function to introduce nonlinearity (e.g. the sigmoid or ReLU functions). This is known as a Graph Convolution Layer (Kipf 2016). The encoder now outputs an approximate posterior for a latent variable matrix $\mathbf{Z}$, with each row corresponding to the latent representation for each node.



(a) Unsupervised model



(b) Semi-supervised model

Figure 1: VFAE models

- The decoder, which now has to generate a distribution over graphs $p(\mathbf{A} \mid \mathbf{Z})$, is simply the sigmoid of inner products of the latent variable $\mathbf{Z}$ we sample from our decoder's distribution. That is, the decoder gives us

$$p(\mathbf{A} \mid \mathbf{Z}) = \prod_i \prod_j p(\mathbf{A}_{ij} \mid \mathbf{z}_i, \mathbf{z}_j) \qquad \text{with} \qquad p(\mathbf{A}_{ij} = 1 \mid \mathbf{z}_i, \mathbf{z}_j) = \sigma(\mathbf{z}_i^T \mathbf{z}_j)$$

  where $\mathbf{z}_i$ is a row of $\mathbf{Z}$ and σ is the sigmoid function.

3.3 Variational Fair Autoencoder

VFAEs offer a method for the VAE to learn fair latent representations. The motivation behind this work is the following problem: how can we use some data $\mathbf{X}$ to make the best predictions of some $\mathbf{y}$ while being unbiased with respect to some observed sensitive variable $\mathbf{s}$? VFAE's accomplish this by enforcing factorized priors $p(\mathbf{s})p(\mathbf{z})$, along with a "Maximum Mean discrepancy" penalty to remove any remaining dependence between $\mathbf{s}$ and $\mathbf{z}$. The framework modifies the original VAE algorithm, mainly in that now all distributions are conditional on $\mathbf{s}$. We begin with the simpler unsupervised case (Figure 1a) where $\mathbf{y}$ is not observed, and then move on to the more useful semi-supervised model (Figure 1b).
In the unsupervised case the objective for a single example is $\mathbf{x}$

$$\max \mathbb{E}_{\mathbf{z} \sim q}[\log p(\mathbf{x}|\mathbf{z}, \mathbf{s})] - D[q(\mathbf{z}|\mathbf{x}, \mathbf{s}) \| p(\mathbf{z})]$$

which gives us an easy way to obtain a posterior $p(\mathbf{z} \mid \mathbf{x}, \mathbf{s})$ that is approximately invariant with respect to $\mathbf{s}$.

Now to extend this to the semi-supervised case, so that we ensure we learn a useful representation for predicting $\mathbf{y}$. To do so, Louizos et al. separate the latent variable into $\mathbf{z}_1$ and $\mathbf{z}_2$. $\mathbf{z}_2$ represents the variation in $\mathbf{z}_1$ that is not dependent on $\mathbf{y}$. Then with the appropriate choice of priors and assumptions of the posteriors (omitted here for brevity, see Louizos et al. for details), we arrive at objective functions for unsupervised examples $\mathcal{L}_u$ and for supervised examples $\mathcal{L}_s$ of

$$\mathcal{L}_u = \mathbb{E}_{q(\mathbf{z}_1|\mathbf{x},\mathbf{s})}\left[-D(q(\mathbf{y} \mid \mathbf{z}_1)\|p(\mathbf{y})) + \log p(\mathbf{x} \mid \mathbf{z}_1, \mathbf{s})\right] + \mathbb{E}_{q(\mathbf{z}_1,\mathbf{y}|\mathbf{x},\mathbf{s})}[-D(q(\mathbf{z}_2 \mid \mathbf{z}_1, \mathbf{y})\|p(\mathbf{z}_2))] + \mathbb{E}_{q(\mathbf{z}_1,\mathbf{y},\mathbf{z}_2|\mathbf{x},\mathbf{s})}[\log p(\mathbf{z}_1 \mid \mathbf{z}_2, \mathbf{y}) - \log q(\mathbf{z}_1 \mid \mathbf{x}, \mathbf{s})]$$

$$\mathcal{L}_s = \mathbb{E}_{q(\mathbf{z}_1|\mathbf{x},\mathbf{s})}\left[-D(q(\mathbf{z}_2 \mid \mathbf{z}_1, \mathbf{y})\|p(\mathbf{z}_2)) + \log p(\mathbf{x} \mid \mathbf{z}_1, \mathbf{s})\right] + \mathbb{E}_{q(\mathbf{z}_1|\mathbf{x},\mathbf{s})q(\mathbf{z}_2|\mathbf{z}_1,\mathbf{y})}[\log p(\mathbf{z}_1 \mid \mathbf{z}_2, \mathbf{y}) - \log q(\mathbf{z}_1 \mid \mathbf{x}, \mathbf{s})]$$

Thus the total loss is

$$\mathcal{L} = \sum_n \mathcal{L}_s(\mathbf{x}_n, \mathbf{y}_n, \mathbf{s}_n) + \sum_m \mathcal{L}_u(\mathbf{x}_m, \mathbf{y}_m, \mathbf{s}_m) + \alpha \sum_n \mathbb{E}_{q(\mathbf{z}_{1_n}|\mathbf{x}_n,\mathbf{s}_n)}[-\log q(\mathbf{y}_n \mid \mathbf{z}_{1_n})]$$

where the last term is added to force the posterior $q(\mathbf{y} \mid \mathbf{z}_1)$ to learn from the labeled as well as the unlabeled data.

3.3.a Maximum Mean Discrepancy

There is an additional penalty term added to further enforce the invariance with respect to $\mathbf{s}$, known as the "Maximum Mean Discrepancy" (MMD). It is a measure of the distance between the empirical statistics ϕ of two datasets $\mathbf{X}, \mathbf{X}'$ and an estimate of how similar the distributions $P(\mathbf{X})$ and $P(\mathbf{X}')$ are. Mathematically it is:

$$\left\| \frac{1}{N_0} \sum_{i=1}^{N_0} \phi(\mathbf{x}_i) - \frac{1}{N_1} \sum_{i=1}^{N_1} \phi(\mathbf{x}'_i) \right\|^2$$

It turns out that this can be written in terms of the sum of kernels (Gretton et al. 2006), becoming:

$$\frac{1}{N_0^2} \sum_{n=1}^{N_0} \sum_{m=1}^{N_0} k(\mathbf{x}_n, \mathbf{x}_m) + \frac{1}{N_1^2} \sum_{n=1}^{N_1} \sum_{m=1}^{N_1} k(\mathbf{x}'_n, \mathbf{x}'_m) + \frac{2}{N_0 N_1} \sum_{n=1}^{N_0} \sum_{m=1}^{N_1} k(\mathbf{x}_n, \mathbf{x}'_m)$$

When k is the Gaussian kernel $k(x, x') = e^{-\gamma\|x-x'\|^2}$, the MMD is 0 if and only if $P(\mathbf{X}) = P(\mathbf{X}')$. However this can be cumbersome to optimize with minibatch stochastic gradient descent over as we would need to compute the $M \times M$ Gram matrix for each minibatch of size M. Luckily, it has been shown that the statistics ϕ can be approximated quickly using random kitchen sinks (Zhao & Meng 2015):

$$\phi_{\mathbf{W}}(\mathbf{x}) = \sqrt{\frac{2}{D}} \cos\left(\sqrt{\frac{2}{\gamma}}\mathbf{x}\mathbf{W} + \mathbf{b}\right)$$

where $\mathbf{W}$ is a $K \times D$ isotropic Gaussian random matrix, $\mathbf{b}$ is a D-dimenional uniform $[0, 2\pi]$ random vector, and K and D are the dimensionality of $\mathbf{x}$ and the number of random features, respectively. Thus the MMD is approximately:

$$\ell_{\text{MMD}}(\mathbf{Z}_1^{\mathbf{s}=0)}, \mathbf{Z}_1^{\mathbf{s}=1}) = \|\mathbb{E}_{p(\mathbf{x}|\mathbf{s}=0)}[\mathbb{E}_{q(\mathbf{z}_1|\mathbf{s}=0}[\phi_{\mathbf{W}}(\mathbf{z}_1)]] - \mathbb{E}_{p(\mathbf{x}|\mathbf{s}=1)}[\mathbb{E}_{q(\mathbf{z}_1|\mathbf{s}=1)}[\phi_{\mathbf{W}}(\mathbf{z}_1)]]\|^2$$

Thus the full objective for the VFAE is

$$\mathcal{L} = \sum_n \mathcal{L}_s(\mathbf{x}_n, \mathbf{y}_n, \mathbf{s}_n) + \sum_m \mathcal{L}_u(\mathbf{x}_m, \mathbf{y}_m, \mathbf{s}_m) + \alpha \sum_n \mathbb{E}_{q(\mathbf{z}_{1_n}|\mathbf{x}_n,\mathbf{s}_n)}[-\log q(\mathbf{y}_n \mid \mathbf{z}_{1_n})] - \beta \ell_{MMD}(\mathbf{Z}_1^{\mathbf{s}=0}, \mathbf{Z}_1^{\mathbf{s}=1})$$

4 Model

We now combine the VGAE and VFAE discussed above to create the VFGAE. We use the exact same loss as described for the VFAE, except now we additionally condition our posteriors q on the adjacency matrix of the graph $\mathbf{A}$, as in the VGAE. The encoding layers are graph convolution layers and the final decoding layer computes inner products, also as in the VGAE. The inputs to the encoding and decoding layers are also modified from the VGAE so that they make use of information about the nuisance variable $\mathbf{s}$. Then the encoding layers are now of the form

$$\tau(\tilde{\mathbf{A}}\tilde{\mathbf{X}}\mathbf{W}_0)$$

where $\tilde{\mathbf{A}}$ is the symmetrically normalized adjacency matrix and $\tilde{\mathbf{X}}$ is the concatenation of $\mathbf{X}$ and $\mathbf{s}$. Then the decoding layers are of the form

$$p(\mathbf{A} \mid \mathbf{Z}_1, \mathbf{s}) = \prod_i \prod_j p(\mathbf{A}_{ij} \mid \tilde{\mathbf{z}}_{1_i}, \tilde{\mathbf{z}}_{1_j}) \qquad \text{with} \qquad p(\mathbf{A}_{ij} = 1 \mid \tilde{\mathbf{z}}_{1_i}, \tilde{\mathbf{z}}_{1_j}) = \sigma(\tilde{\mathbf{z}}_{1_i}^T \tilde{\mathbf{z}}_{1_j})$$

where $\tilde{\mathbf{z}}_{1_j}$ is the concatenation of $\mathbf{z}_{1_j}$ and $\mathbf{s}_j$.

5 Experiments and Analysis

We implement our model in Python using TensorFlow and optimize the objective function using the AdamOptimizer provided by TensorFlow. We ran our framework on the Cora citation network, with the goal of learning fair representations for predicting the label of whether the paper was on neural networks or not. We looked at three metrics, the accuracy in predicting $\mathbf{s}$, the accuracy in predicting $\mathbf{y}$, and a metric known as discrimination. The metrics are defined as

- Accuracy:

$$1 - \frac{1}{N} \sum_{i=1}^{N} |\mathbf{y} - \hat{\mathbf{y}}_n|$$

- Discrimination: There are two definitions, one using the predicted labels:

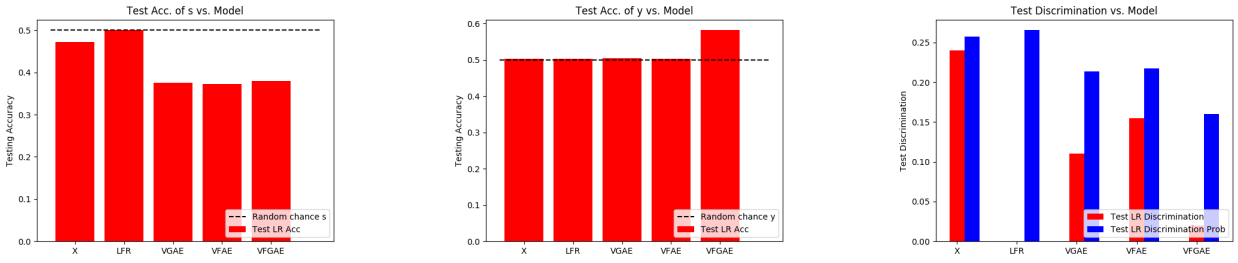
$$\left| \frac{\sum_{n:\mathbf{s}_n=1} \hat{\mathbf{y}}_n}{\sum_{n:\mathbf{s}_n=1} 1} - \frac{\sum_{n:\mathbf{s}_n=0} \hat{\mathbf{y}}_n}{\sum_{n:\mathbf{s}_n=0} 1} \right|$$

  and another that uses the predicted probabilities, which gives a more nuanced look at the fairness of the predictor:

$$\left| \frac{\sum_{n:\mathbf{s}_n=1} p(\hat{\mathbf{y}}_n)}{\sum_{n:\mathbf{s}_n=1} 1} - \frac{\sum_{n:\mathbf{s}_n=0} p(\hat{\mathbf{y}}_n)}{\sum_{n:\mathbf{s}_n=0} 1} \right|$$

  The discrimination metric measures a form of statistical parity as it measure the difference in proportion of positive classifications between the sensitive and non-sensitive groups.

We compared our model against the VFAE, VGAE, and another approach for learning fair representations LFR (Zemel et al. 2013), as well as naively using $\mathbf{X}$ itself. We used logistic regression models taking the latent representations as input as our predictors of $\mathbf{s}$,$\mathbf{y}$ for all



(a) (b) (c)

Figure 2: Accuracy and Discrimination Plots

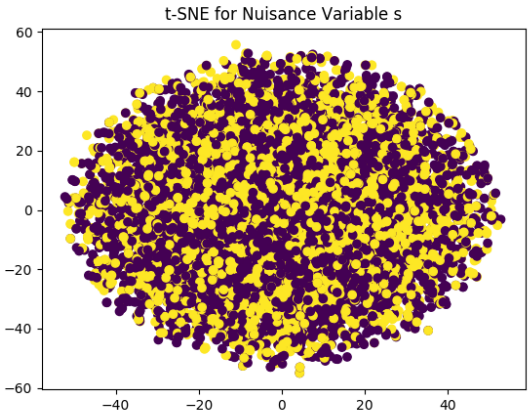


Figure 3: t-SNE of $\mathbf{X}$

models. Note that our primary goals are invariance with respect of $\mathbf{s}$ in terms of accuracy on $\mathbf{s}$ and the discrimination scores, and so a low $\mathbf{y}$ accuracy is not as important.
Note that our model was able to produce a lower discrimination score than both the VFAE and VGAE, showing there is a benefit to combining the two when handling graph data. Furthermore, we have a better probabilistic discrimination score than LFR, indicating our model is in some sense more fair than LFR. Our accuracy in predicting $\mathbf{y}$ is slightly better than the other models, which is also encouraging.

That being said, these are still relatively weak prediction accuracies, and it is curious that VFGAE had a higher accuracy than simply using $\mathbf{X}$. After much investigation (for a long time I thought it was a bug in my code), I ran a t-SNE dimensionality reduction clustering algorithm to see how the sensitive and non-sensitive nodes were clustered in the $\mathbf{X}$ space (Figure 3). The results show that there is no separation of the sensitive and nonsensitive nodes. This implies that there is no information about $\mathbf{s}$ contained in $\mathbf{X}$. This then explains why the fair learning models yielded such mediocre results. If this had been figured out sooner, we would have found a different dataset where there were correlations between $\mathbf{X}$ and $\mathbf{s}$, and it is what we plan to do moving forward.

6 Future

The above results are promising preliminary indicators of the value of the VFGAE. As discussed above, the sub-optimal choice of dataset resulted in poor performance and makes it hard to say anything definitive. We plan to run similar experiments on other datasets better suited to the problem, for instance social network data from Facebook or Google. The model itself could also be improved upon, by using a more sophisticated final layer than a simple inner product layer. This choice, in combination with the Gaussian priors, may not be best since the inner product layer pushes the embeddings away from zero. Nevertheless, we see this as a first step that shows there is potential for this framework to be an improvement over current models.

7 References

Doersch C. (2016) Tutorial on Variational Autoencoders:
`https://arxiv.org/pdf/1606.05908.pdf`

Gretton et al. (2006) A kernel method of the two-sample-problem. Advances in neural information processing systems, p. 513-520, 2006.

Kipf T. et al. (2016) Variational Graph Auto-Encoders:
`https://arxiv.org/pdf/1611.07308.pdf`

Louizos C. et al. (2016) The Variational Fair Autoencoder. ICLR 2016:
`https://arxiv.org/pdf/1511.00830.pdf`

Sen et al. (2008) Collective classification in network data. AI Magazine 2008: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.188.906&rep=rep1&type=pdf`

Zemel et al. (2013) Learning Fair Representations:
`https://www.cs.toronto.edu/~toni/Papers/icml-final.pdf`

Zhao & Meng. Fastmmd: Ensemble of circular discrepancy for efficient two-sample test: Neural computation, 2015.