Mutual information estimation for graph convolutional neural networks

Marius C. Landverk[1] and Signe Riemer-Sørensen[2]

[1]mariuslandc@gmail.com
[2]signe.riemer-sorensen@sintef.no, Analytics and AI, SINTEF Digital, P.O. Box 124 Blindern, N-0314 Oslo, Norway

Abstract

Measuring model performance is a key issue for deep learning practitioners. However, we often lack the ability to explain why a specific architecture attains superior predictive accuracy for a given data set. Often, validation accuracy is used as a performance heuristic quantifying how well a network generalizes to unseen data, but it does not capture anything about the information flow in the model.

Mutual information can be used as a measure of the quality of internal representations in deep learning models, and the information plane may provide insights into whether the model exploits the available information in the data.

The information plane has previously been explored for fully connected neural networks and convolutional architectures. We present an architecture-agnostic method for tracking a network's internal representations during training, which are then used to create the mutual information plane. The method is exemplified for graph-based neural networks fitted on citation data. We compare how the inductive bias introduced in graph-based architectures changes the mutual information plane relative to a fully connected neural network.

1 Introduction

1.1 Motivation

For classification problems, the validation accuracy is a common heuristic to gauge the generalization capabilities of a model. Whilst it is a useful metric, it leaves something to be desired in terms of understanding why a model is able to perform as well, or as ill, as it does. The ability of a model to fit the data is directly related to the quality of the representations generated by the model. The information plane is given as the mutual information between a model's representations, Z_i, with the input X and the true labels Y [11, 16, 14]. Visual inspection of the information plane at the point in training where the performance plateaus can provide qualitative guidance. In this paper, we seek to use the information plane from the information bottleneck method [14] in order to gauge model fit across architectures with different inductive biases, and we exemplify by comparing a fully-connected neural network to graph-based networks trained on the same classification task.

1.2 Mutual information

The core of the information bottleneck method is a quantification of shared information between random variables originally suggested by [11]. For two random variables X and Y, the mutual information $I(X, Y)$ is a symmetric quantity measuring the amount of information that X contains about Y, and vice versa. If we want to predict the true labels Y from X, we can define a compressed version of X, called Z, with shared information $I(X, Z)$ and $I(Z, Y)$. The minimal representation Z (maximally compressed information on Y from X), that simultaneously maximises the mutual information with Y is called the 'information bottleneck' [14, 10]:

$$\text{argmax}_{Z\in\Delta} I(Y, Z) \text{ such that } I(X, Z) \leq R, \quad (1)$$

where R is a given scalar threshold. Mutual information can be expressed in terms of the entropy $H(X)$, $H(Y)$, as $I(X, Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$, where $H(Y|X)$ is the conditional entropy of Y given X [3].

If Y, X and Z are random variables in a Markov chain where $Y \to X \to Z$, the data-processing inequality applies to the mutual information between the variables [3]:

$$I(X, Y) \geq I(Z, Y). \quad (2)$$

In other words, the information about an input X contained in Z cannot be increased by applying

https://doi.org/10.7557/18.6257

a function to Z, as long as the function does not use additional information about X.

For neural networks, the representations Z_i can be associated with the successive hidden layers. For fully connected neural networks, the data processing inequality can then be applied to obtain a set of inequalities. However, in more complicated neural network structures such as recurrent neural networks or graph convolutional neural networks, the hidden layers Z draws on information about X, and the data processing inequality is broken. In the case of graph convolutional neural network, the data processing inequality is broken explicitly because all hidden layers uses the edge information in X to compute the forward pass.

1.3 Related work

It has been debated whether the mutual information follows a specific pattern during training with a fitting phase followed by a compression phase [14, 13]. So far, the compression phase is only observed for symmetric activation functions such as SIGMOID and TANH (see [5] for an overview of experiments). Here we do not make any claims regarding the compression phase, but provide a framework for comparing the mutual information across neural network architectures.

Early works [16, 14] use a binning procedure to estimate the mutual information. This is only meaningful for activation functions with restricted output range e.g. SIGMOID, and the choice of binning has been proven to affect the resulting mutual information estimate significantly [15]. This can be alleviated by assuming a distribution for the hidden variables Z and rewriting the mutual information to only include the hidden layer representations. In Sec. 2 we focus on this method which allows to derive both upper and lower bounds for $I(X, Z)$ and $I(Z, Y)$ [9, 10, 13], and only briefly comment here on alternative methods. Instead of assuming the distributions, the approach of [2] relies on a completely separate neural network to estimate the mutual information of the problem-specific neural network. However, this introduces additional hyper parameters and high risk of numerical instabilities.

The method of [20] uses an estimate for the entropy based on an eigenvalue decomposition of X and a smoothing with an infinitely-divisible, real-valued positive definite kernel. However, since the method relies on computing eigenvalues, the computational requirements fast become prohibitive, as this computation is done several times in order to obtain a single mutual information plot.

The authors of [4] apply the mutual information plane as a way to explain convolutional neural networks trained for image classification (e.g. ResNET), but they only consider that specific architecture and image-like data.

1.4 Our contribution

We present a framework for tracking the activations of a generic neural network. To produce an information plane, a classification task is required, since otherwise the conditional entropy $H(X|Y)$ becomes intractable. The framework has been developed in PyTorch [12] and can be found on GitHub[1].

The novelty of the developed framework is that it is agnostic to the kind of model architecture, as long as all the submodules to be tracked are defined explicitly in the initialization method of the PyTorch module. The original methods for mutual information estimations were developed on fully connected neural networks. To our knowledge, mutual information has not previously been estimated for graph neural networks or recurrent networks (see e.g. table 1 in [5]). The already existing codebases from [11, 13] are well suited for the specifically investigated network architectures and setup, but they lack the flexibility to be easily applied on different neural architectures, and hence do not invite comparisons across architectures.

In Sec. 3, we exemplify on graph-like data and discuss how the inductive bias introduced by the imposed structure of the graph neural network affects the model quality and how that is expressed in the mutual information plane.

2 Method

We estimate the mutual information based on the method from Kolchinsky et. al. [9, 10]. Given a batch B with N_B samples, the upper bound estimates[2] for the mutual information $I(X, Z_B)$ and $I(Z_B, Y)$ are

$$I(X, Z_B) \leq -\frac{1}{N_B} \sum_i \log \frac{1}{N_B} \sum_j K_{ij} \qquad (3)$$

with $K_{ij} = \exp\left(-\|Z_i - Z_j\|_2^2/(2\sigma^2)\right)$, and

$$I(Z_B, Y) \leq -\frac{1}{N_B} \sum_i \log \frac{1}{N_B} \sum_j K_{ij} \qquad (4)$$
$$- \sum_{c=1}^{C} p_c \left[-\frac{1}{N_c} \sum_{\{i|Y_i=c\}} \log \frac{1}{N_c} \sum_{\{j|Y_j=c\}} K_{ij} \right],$$

[1] https://github.com/mariusmcl/info-bottleneck-tracking

[2] The lower bound is obtained similarly by replacing the 2 in K_{ij} with 8.

with $p_c = \frac{N_c}{N}$ where N_c is the total number of examples in category c and N is the total number of samples.

The estimate is based on pairwise distances between Z_i and Z_j. If the components in the representation of a class (some/all classes) are very clustered, their pairwise distances are small and hence the second term in equation (4) becomes small, leading to large mutual information. If the representation is e.g. random and thereby not clustered at all, the mutual information becomes large. Hence, if the inter-class distances are small, one can relate a sub-region of the representation's domain to a class label, which in turn would yield a higher estimate of mutual information.

The only hyperparameter in the method is the homoscedastic noise variance σ^2 of the assumed normal distribution. In equations (3) and (4), σ^2 can be interpreted as a bandwidth term, with a higher σ^2 allowing more interactions between batch elements i and j, leading to higher values of K_{ij}. Different choices of σ^2 produce quantitatively and qualitatively different results. If the variance is too small, the estimates become noisy (large scatter in the plots), and if it becomes too big the estimates have large uncertainty, which also tends to break the data processing inequality even for fully connected neural networks. This is consistent with previous observations based on independent estimators [5, 18, 19].

The developed framework is designed to be compatible with the PyTorch training workflow and relies on recording the activation values of the model during training by attaching a "hook" on the forward pass[3]. We show an example training loop with activation tracking during training in code listing 1. The forward hooks are defined through the model's `named_children` attribute[4].

We have verified that the method reproduces the mutual information estimates of [14] and [4] (as far as possible with the details provided in the paper).

3 Example on graph-like data

We exemplify the method on two different graph-like datasets (Sec. 3.1, Sec. 3.2) and and three different architectures (Sec. 3.3-Sec. 3.5).

[3]Pytorch: Forward And Backward Function Hooks, `https://pytorch.org/tutorials/beginner/former_torchies/nnft_tutorial.html#forward-and-backward-function-hooks`

[4]Pytorch: Modules `https://pytorch.org/docs/stable/generated/torch.nn.Module.html`

```
for epoch in range(num_epochs):
    hooks = list(tracker.forward_hooks.keys())
    tracker.register_new_epoch(hooks)
    for x, y in train_loader:
        optimizer.zero_grad()
        out = classifier(x)
        loss = crossentropy_loss(out, y)
        loss.backward()
        optimizer.step()
        tracker.save()
```

Listing 1: An example of activation tracking during a training loop in PyTorch for training a network for a classification task. The forward hooks are defined through the model's named_children attribute.

3.1 The Cora data

The Cora data[5] has a graph like structure with 2708 nodes representing scientific papers and 5429 edges given by citations between papers. The edges are unidirectional so paper A can cite paper B or opposite, or they can both cite each other. The papers are described by a feature vector of length $D = 1433$ where each element is the number count of a predetermined word (bag-of-words feature vector). In addition, each of the papers belong to one of seven categories indicating which scientific field the article is published in. For the Cora data, we consider the task of classifying the papers into one of seven publication categories. We use the built in masks for splitting in training (140 nodes), validation (500 nodes) and test (1000 nodes) samples.

3.2 The arxiv data

The arxiv dataset[6] contains the citation network between preprint papers in computer science submitted to arXiv. The nodes represents 169,343 preprints connected by 1,166,243 directed edges representing one preprint citing another. The papers are described by a feature vector of length $D = 128$ where each element is the number count of a predetermined word (bag-of-words feature vector). The targets are one of 40 sub-categories assigned to each paper (e.g. cs.AI, cs.LG etc. which are manually assigned by the authors and moderators), and hence the task is classification. We follow the recommendation and split into training and test nodes based on submission dates such that we train on papers published until 2017, validate on those

[5]`https://relational.fit.cvut.cz/dataset/CORA`

[6]`https://ogb.stanford.edu/docs/nodeprop/#ogbn-arxiv`

published in 2018, and test on those published since 2019. This is known to be slightly more challenging than the random split used in the Cora data [7].

3.3 Multilayer perceptron model

The simplest model is a vanilla multilayer perceptron (MLP) model [6]. The propagation rule for the MLP is given by

$$Z_{l+1} = \sigma\big(Z_l W_{l+1}\big), \tag{5}$$

where W is the weight matrix, and σ indicates the activation function. The MLP was chosen as reference to explore if the mutual information plane is affected by inductive bias in the model.

3.4 Graph convolution model

Graph convolutions are a generalisation of the convolutionary operator that have become a standard workhorse in deep learning on images, to operate on arbitrary graphs [8, 1]. The propagation rule for a graph convolutional network is achieved through a weighted sum of neighboring nodes' features, followed by a matrix multiplication. The propagation rule for a node n is given by

$$Z^n_{l+1} = \sigma\left(\left(\sum_{j\in\mathcal{N}(n)} \frac{1}{\sqrt{\hat{d}_j \cdot \hat{d}_n}} Z^j_l\right) W_{l+1}\right), \tag{6}$$

with $\hat{d}_i$ being defined as $\hat{d}_i = 1 + \sum_{j\in\mathcal{N}(i)} e_{ij}$ with e_{ij} being the edge weight between nodes i and j, with the default being $e_{ij} = 1$. The graph convolutional architecture thus has a fixed weighing of a nodes' attribute based on that nodes' edge degree.

3.5 Graph attention model

Graph attention networks follows the same propagation rule as in equation (6) but apply an attention mechanism at the point when the information is aggregated from the neighbouring nodes through normalised attention scores assigned to each source node before the summation [17]. For a single node n the propagation can be written as:

$$Z^n_{l+1} = \sigma\left(\left(\sum_{j\in\mathcal{N}(n)} \alpha^{j,n} Z^j_l\right) W_{l+1}\right), \tag{7}$$

where the attention scores $\alpha^{j,n}$ are trainable and thus not fixed as for the graph convolutional network.

3.6 Training the models

All architectures are fitted using three hidden layers, with 300, 200 and 100 neurons each. For the Cora data we use RELU activations, while for the arxiv data we fit two models for each architecture using TANH and RELU activations respectively. We train for 600 epochs for the arxiv dataset, and for 100 epochs for the Cora dataset. Due to the difficulties of batching a graph dataset, all models are trained with full-batch gradient descent, and as such each epoch corresponds to one update of the model parameters.

We estimate the upper bound on the mutual information using Listing 1 with a noise parameter of $\sigma^2 = 0.1$ for the Cora dataset and $\sigma^2 = 0.001$ for the arxiv dataset. We have tested multiple values of σ. Generally, we find that small values ($\sigma = 10^{-5}$) lead to very scattered values of mutual information, while large values ($\sigma = 0.1$) reduce the scatter at the cost of increased uncertainty, which may lead to apparent violation of the data processing inequality for the MLP model on some datasets. We selected the values of σ such that the data processing inequality was not (strongly) violated for the MLP model (see also Sec. 2).

3.7 Results on Cora data

Fig. 1 shows the training and validation accuracy during training. We see that both models obtain a training accuracy of almost 100%, but the validation accuracy of the MLP model remains low, while the GCN structure enforces specific relationships in the model that appears to prevent some overfitting and consequently higher validation accuracy.

Fig. 2 shows the information planes for the MLP and GCN models fitted to the Cora data. For the MLP model we see that the mutual information between input and the individual layers ($I(X, Z_i)$) and the individual layers and output ($I(Z_i, Y)$) increase with training, also referred to as the fitting phase. As expected, we do not observe any compression phase due to the choice of the asymmetric RELU activation function. For the MLP, the layers follow the data processing inequality (equation (2)) with $I(X, Z_1) \geq I(X, Z_2) \geq I(X, Z_3)$ and $I(Z_1, Y) \geq I(Z_2, Y) \geq I(Z_3, Y)$. We note that even though the MLP model has less information about X in its final layer compared to the GCN model, it is not able to use this in order to create generalized features which perform better on the validation set. Seen together with the gap between training and validation accuracy in Fig. 1 this is a clear sign that the MLP architecture is unable to generalize well on this dataset.

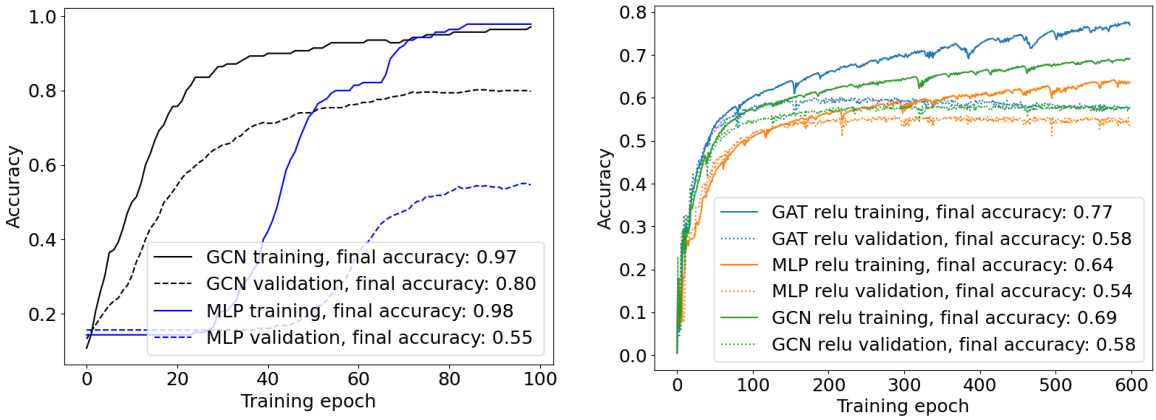

Figure 1: The training (solid lines) and validation accuracies (dashed lines) during the fitting process. To the left is the Cora data with the MLP model in blue and GCN model in black. The to right is the arxiv data with the GAT model is blue, MLP model is orange and GCN model is green.

For the GCN model we also observe that the mutual information between input and the individual layers ($I(X, Z_i)$) and the individual layers and output ($I(Z_i, Y)$) increase with training. By choosing $\sigma^2 = 0.1$ the data processing inequality is not violated, however for lower values of σ^2 the data processing inequality was violated for the GCN model. It is not always expected that the GCN architecture fulfills the data processing inequality. This is due to the successive representations Z_i^1, Z_i^2 drawing upon knowledge of the input data X to generate the representations. Specifically for the GCN, they are obtained through an averaging of previous features from neighbouring nodes. This introduces an explicit dependence upon the structure of X in the update step in equation (6), violating the data processing inequality.

It is somewhat surprising that the GCN model retains more information about X than the MLP model while at the same time performing better on the validation set. Usually one would seek to generalize away as much unnecessary variation in X as possible in order to only retain variation related to predicting Y. It seems the MLP architecture is bottlenecked by its ability to only convey information through matrix multiplications, whilst the GCN model alleviates this by also pooling each neighboring nodes' features in its propagation step.

3.8 Results on arxiv data

Fig. 1 (right) shows the training and validation accuracy during training using RELU activations on the arxiv data. On the arxiv data, the models obtain training accuracies of 64-77%, and validation accuracies of 54% for the MLP model and 58% for the two graph models[7]. The gap between training and validation accuracies indicate some overfitting, which is worst for the MLP. We assume this is due to the graph-based models enforcing specific relationships that helps with the generalization. The accuracies have very similar behaviour for the TANH activations (not shown).

Fig. 3 shows the information planes for all three models fitted to the arxiv data with RELU activations. For all models we observe a rapid fitting phase with increasing mutual information along both axes. For the MLP model we see that all the layers ends up within a very small region and while the inset indicates that the ordering breaks the data processing inequality, this is most likely due to uncertainty in the estimation. We associate the closeness of the layers to a lack of compression during training and a poor generalization performance. The GCN and GAT show similar patterns, but with fewer iterations needed for the rapid increase of mutual information. The GAT performs a small compression of the last layer. In both cases, the closeness of the layers indicate low compression between the layers.

At the current stage, we are unable to draw general conclusions from the mutual information planes, but the presented framework will ease further study of the mutual information plane of inductively biased models.

[7]These are slightly worse than the official best fit models obtained on the data, but we have not tuned the individual models [7] in order to keep them comparable

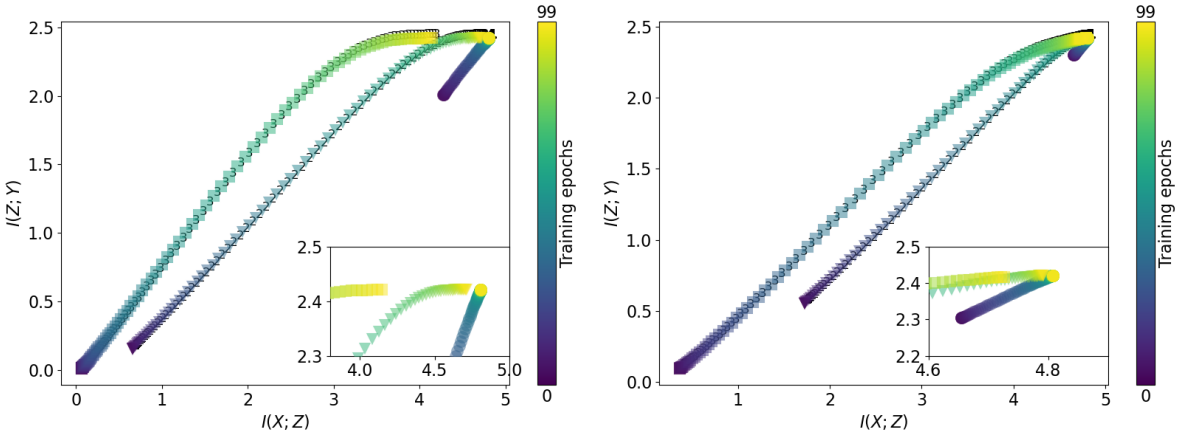

Figure 2: The information plane of the training process for MLP model (left) and GCN model (right) on the Cora citation dataset, with noise parameter $\sigma^2 = 0.1$ and RELU activations. The different symbols/numbers refer to layers in the networks (◯:1, △:2, □:3) whereas the colours refer to fitting epoch. The inserts provide a zoom-in of the convergence region. For the MLP the third layer (squares) ends up with smaller mutual information with X than the previous two layers (triangles and circles) adhering to the data processing inequality. Through its pooling operation, the GCN architecture is able to retain more information about X than the MLP model, which is shown by the final layer (square) having noticeably higher mutual information about X compared to its fully connected counterpart.

4 Conclusion

We provide a framework for performing the information plane analysis by tracking the activations of a general PyTorch model. The work was partly motivated by the need for evaluating generalisation performance based on training data alone, and partly by the need for informed model architecture selection. Being able to compare the mutual information plane across neural architectures is reliant on whether the architectures violate the data processing inequality. Ideally the data processing inequality would hold, but for more complicated architectures, and as shown in this work, that is not necessarily the case but the information plane can still provide insight in some cases.

References

[1] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, and others. Relational inductive biases, deep learning, and graph networks. *arXiv e-prints*, 1806.01261:arXiv:1806.01261, June 2018.

[2] M. I. Belghazi, A. Baratin, S. Rajeshwar, S. Ozair, Y. Bengio, A. Courville, and D. Hjelm. Mutual information neural estimation. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 531–540. PMLR, 10–15 Jul 2018. URL `https://proceedings.mlr.press/v80/belghazi18a.html`.

[3] T. M. Cover and J. A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, USA, 2006. ISBN 0471241954.

[4] H. Fang, V. Wang, and M. Yamaguchi. Dissecting deep learning networks—visualizing mutual information. *Entropy*, 20(11), 2018. ISSN 1099-4300. doi: 10.3390/e20110823. URL `https://www.mdpi.com/1099-4300/20/11/823`.

[5] B. C. Geiger. On Information Plane Analyses of Neural Network Classifiers – A Review. *arXiv e-prints*, 2003.09671:arXiv:2003.09671, Mar. 2020.

[6] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[7] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *arXiv e-prints*, 2005.00687:arXiv:2005.00687, May 2020.

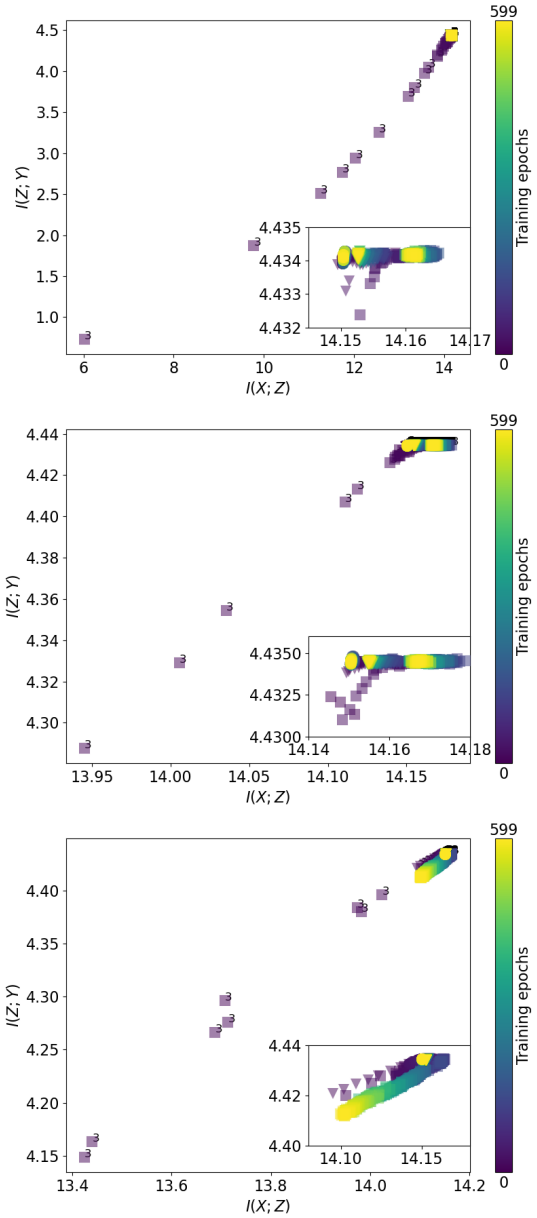

Figure 3: The information plane of the training process for MLP model (top) and GCN model (middle) and GAT model (bottom) on the arxiv citation dataset, with noise parameter $\sigma^2 = 0.001$ and RELU activation functions. The different symbols/numbers refer to layers in the networks (◯:1, △:2, □:3) whereas the colours refer to fitting epoch. The inserts provide a zoom-in of the convergence region.

[8] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv e-prints*, 1609.02907, 2016. URL `http://arxiv.org/abs/1609.02907`.

[9] A. Kolchinsky and B. D. Tracey. Estimating mixture entropy with pairwise distances. *arXiv e-prints*, 1706.02419, 2017. URL `http://arxiv.org/abs/1706.02419`.

[10] A. Kolchinsky, B. D. Tracey, and D. H. Wolpert. Nonlinear information bottleneck. *Entropy*, 21(12), 2019. URL `http://arxiv.org/abs/1705.02436`.

[11] W. B. Naftali Tishby, Fernando C. Pereira. The information bottleneck method. *arXiv e-prints*, physics/0004057, 2000. URL `https://arxiv.org/abs/physics/0004057`.

[12] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv e-prints*, 1912.01703, 2019. URL `http://arxiv.org/abs/1912.01703`.

[13] A. M. Saxe, Y. Bansal, J. Dapello, M. Advani, A. Kolchinsky, B. D. Tracey, and D. D. Cox. On the information bottleneck theory of deep learning. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12): 124020, dec 2019. doi: 10.1088/1742-5468/ab3985. URL `https://doi.org/10.1088/1742-5468/ab3985`.

[14] R. Shwartz-Ziv and N. Tishby. Opening the black box of deep neural networks via information. *arXiv e-prints*, 1703.00810, 2017. URL `http://arxiv.org/abs/1703.00810`.

[15] S. Sloth Lorenzen, C. Igel, and M. Nielsen. Information Bottleneck: Exact Analysis of (Quantized) Neural Networks. *arXiv e-prints*, 2106.12912:arXiv:2106.12912, June 2021.

[16] N. Tishby and N. Zaslavsky. Deep learning and the information bottleneck principle. In *2015 IEEE Information Theory Workshop (ITW)*, pages 1–5. IEEE, 2015. doi: 10.1109/ITW.2015.7133169.

[17] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph Attention Networks. *arXiv e-prints*, 1710.10903: arXiv:1710.10903, Oct. 2017.

[18] S. Yu and J. C. Príncipe. Understanding autoencoders with information theoretic concepts. *Neural Networks*, 117:

104–123, 2019. ISSN 0893-6080. doi: 10.1016/j.neunet.2019.05.003. URL https://www.sciencedirect.com/science/article/pii/S0893608019301352.

[19] S. Yu, K. Wickstrøm, R. Jenssen, and J. C. Príncipe. Understanding convolutional neural networks with information theory: An initial exploration. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):435–442, 2021. doi: 10.1109/TNNLS.2020.2968509.

[20] X. Yu, S. Yu, and J. C. Príncipe. Deep deterministic information bottleneck with matrix-based entropy functional. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3160–3164, 2021. doi: 10.1109/ICASSP39728.2021.9414151.