Message in a bottle: learning dynamics in the information plane

Daniel Kunin (kunin), Mansheej Paul (mansheej), Matthew Bull (bullm)

December 15, 2017

---

1 Introduction

The fundamental challenge of supervised learning is to navigate a trade off between compressing the input features and preserving the meaningful information required for prediction of the output labels. Tishby et al. describe this process as squeezing "the information that X provides about Y through a bottleneck, which they termed the Information Bottleneck (IB) Method [10]. This method has provided a new perspective on the recent successes of Deep Learning [5]. Specifically, they find two phases of learning (a rapid error minimization and a compression phase), provide insight into the value of depth in a neural network and make analogies between generalization and compression of the input.

In this project we set out to explore the Information Bottleneck Method and address the following four major gaps in the current discourse:

1. Discuss mutual information (MI) estimates in the context of supervised learning.
2. Explore a range of learning algorithms in the information plane and investigate the relationship between MI and training error.
3. Investigate bias vs. variance and generalization vs. compression in the information plane.
4. Use the information bottleneck to improve neural network performance by adapting the learning strategy to the learning phase

2 Estimating Mutual Information (MI)

The greatest challenge to using the Information Bottleneck Method is constructing consistent, distribution invariant, high dimensional estimators of MI. This crucial step is often overlooked in most of the literature on the IB method. Our hope was to detail different methods and implement a robust library for MI estimation that future papers could use. We found seven distinct methods for MI estimation in the literature [9], but implemented the first two of the following methods and also used the third for this project.

2.0.1 Adaptive Binning

The goal of this method is to estimate $p(x, y), p(x), p(y)$ empirically by constructing histograms from samples, then directly estimating the mutual information from the ratio of these distributions. The challenge is determining the number and size of the bins. For our implementation, we leverage known work on maximally informative binning and validated first order corrections to subtract off bias [3].

2.0.2 Kernel Density Estimation

The goal of this method is to estimate $p(x, y), p(x), p(y)$ through Kernel Density Estimation, then directly estimate the mutual information from the ratio of these distributions. One of the strengths of this method is that cross validation techniques can be used to strategically determine the bandwidth parameter k [9]. We implemented two versions of this estimator one using Scikit-learn's cross validation library to select the bandwidth parameter [7] and another using Scott's Rule, the default method for bandwidth selection in Scipy [4].

2.0.3 Variational Approximation

This is a recent method that has had a lot of success in making IB based approaches practical. The basic idea of this method is to approximate the MI by fitting a variational lower bound to it. The advantage is that, the functional form then becomes simple and we can do SGD, which allows us to build objective functions and regularizers inspired by mutual information [2]. We were unable to implement this method directly, but for the neural networks section of our project we build upon the code base at [5], which uses this method of MI estimation.

2.0.4 Experimental Evaluation

To evaluate the performance of our MI estimation methods we used data simulated from a mixture of Gaussian random variables X and Y. We first estimated the true MI between X and Y through Monte Carlo integration using $m = 10^8$ samples $(x^{(i)}, y^{(i)})$ from $P(X, Y)$ to compute:

$$I(X,Y) = E\left[\log\left(\frac{P(X|Y)}{\sum_{y'} p(X|y')p(y')}\right)\right] \approx \frac{1}{m}\sum_{i=1}^{m}\log\left(\frac{p(x^{(i)}|y^{(i)})}{\sum_{y'=1}^{K} p(x^{(i)}|y')p(y')}\right)$$

We then used our adaptive binning, KDE, and cross validated KDE methods to estimate the true MI on various datasets of increasing size sampled from the same distribution. We repeated this process to obtain standard errors for each method.

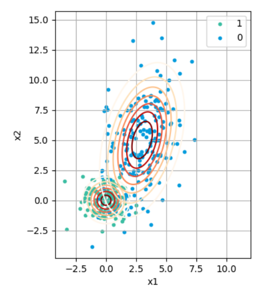

Figure 1: The mixture of Gaussians used to evaluate our MI estimates.

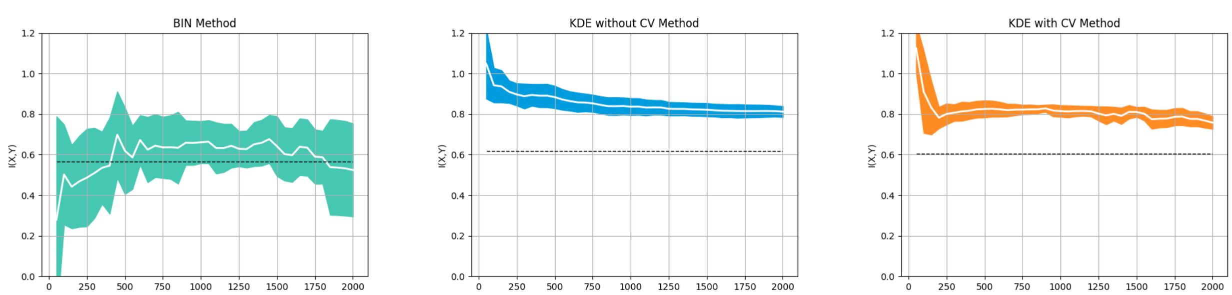

Figure 2: The dotted line is the true MI. Left: the adaptive binning estimate with a first order correction [2] is unbiased but suffers from large variance. Center: a simple kernel density estimate shows significant bias, but minimal variance. Right: a cross validated KDE improves the estimate, but suffers from residual bias. For the next section we use the simple KDE estimator because of its consistency and speed of computation.

3 Learning Algorithms in the Information Plane

The information plane is a plot used in the IB literature to understand how a learning algorithm navigates the tradeoff between compressing the information about the inputs while retaining the information about the outputs. The x-axis is the MI between the input X and the representation of these features T and the y-axis is the MI between the representation T and the output Y. A point in the information plane represents the location of a learning algorithm at some epoch of training [10].

We examined the Percepton, Logistic Regression and Support Vector Machine in the information plane. We implemented these learning algorithms to use our MI estimators during training, with the same sample dataset shown previously, such that we can understand their learning dynamics in the information plane. As far as we know this is the first illustration of these basic and effective learning algorithms in the information plane.

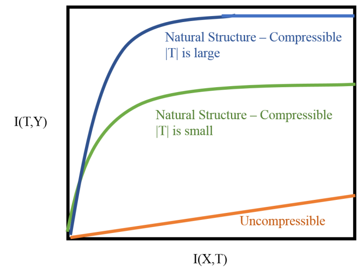

Figure 3: Above is a image that depicts the achievable bound in the information plane [8].

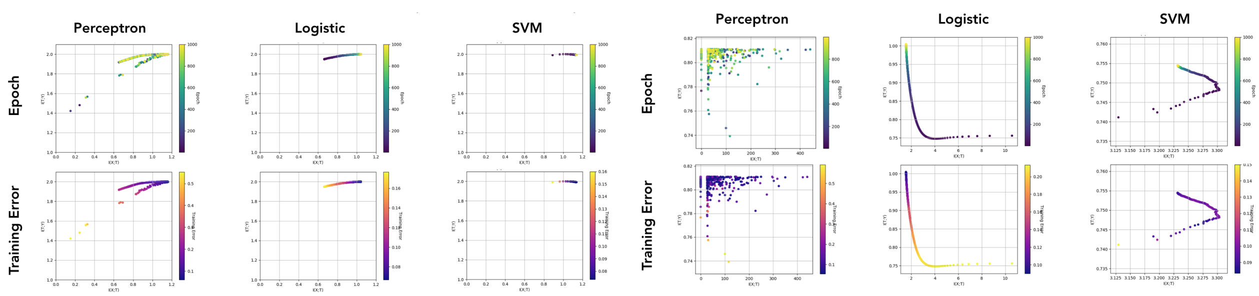

Figure 4: The left plot shows the information plane where the representations, T, are the predictions. In the right plot we use the hidden state (logistics, perceptron) and margin (SVM) as the representation. For both plots we color points by epoch and training error.

Through these plots we observe the following interesting relationships:

1. As the perceptron learns, it jumps around in the information plane in a relatively inconsistent pattern as indicated by the left plot with epochs as the colorbar. This is likely because the perceptron does not consistently decrease training error as it learns. When we look at the same plot using training error as the colorbar, we find an obvious relationship between the location of the algorithm in the information plane and the current training error. This relationship is evident in the other learning algorithms as well. We detailed some initial analytic exploration of this relationship on our Github library.

2. In both logistic regression and SVM we see a markov chain property to their dynamics in the information plane. This characteristic is evident when using either the predictions or the sigmoid/margin values as the representation. This is an expected characteristic of both algorithms in terms of their quadratic and hinge loss respectively, further confirming our intuition of a relationship between mutual information and error, which we investigate in the next section.

3. In both the logistic regression and SVM plots on the right we observe a striking "compression" motion in the information plane when using the sigmoid and margin values respectively as the representation. This mysterious motion has been documented previously for neural networks, suggesting that studying this phase transition in simple learning algorithms could shed light on this process. We further investigate the meaning and value of this phase transition in the following sections.

4 Bias and Variance in the Information Plane

To gain intuition on how the information plane dynamics correspond to more familiar notions of performance, we examine how models which underfit and overfit compare on the information plane. We use the moon dataset from Scikit-learn [6] and 1 hidden layered Neural Networks with 200, 3 and 1 hidden neurons (averaged over 25 runs each) to demonstrate overfit, a just right fit and underfit respectively in the information plane. We build on the codebase in [5].

As we tradeoff bias for variance, we observe the learning trajectories move right and up in the information plane. The neural networks with 3 and 200 hidden neurons have similar test error but the network with 3 neurons has lower generalization error and less I(X,T). As shown in figure 5, comparing the learning curve to the information plane suggests that higher I(X,T) is consistent with overfitting.

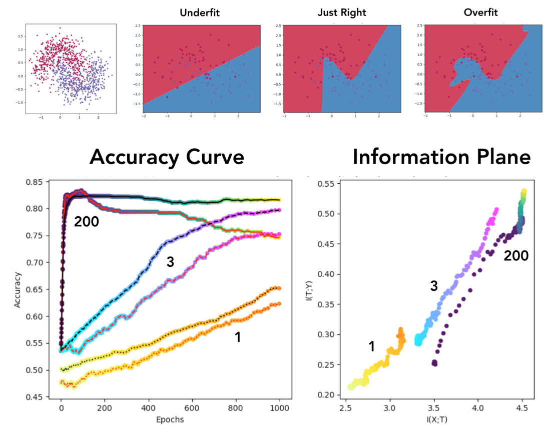

Figure 5: Top: The moon data set fit with 1,3 and 200 hidden neurons respectively. Bottom: The learning curve (accuracy versus training epochs) and the output layer in the information plane for each model.

5 Generalization and Compression in the Information Plane

There is an often alluded to relationship between better generalization and compression of the inputs [5]. In image data, the relevant information about our cat is in its eyes and pointy ears, not the fur color or background. To investigate this we test our network on a simple toy dataset that should be compressed to get efficient performance. We add 100 dimensions of Gaussian noise to our moon data set and use this to train 3 and 5 layered deep networks. The generalization gap is the train minus test error and is the colorbar metric. To generalize well, the networks need to compress the relevant information from the high dimensional data and only keep the information relevant to the classification. We build on the codebase in [5].

We observe that the 3 layered network has worse generalization and ends up with a higher I(X,T); it is not compressing as well. Deeper networks compress better [5]. However, even as the networks compress the generalization worsens indicating that this relationship is not always straightforward.

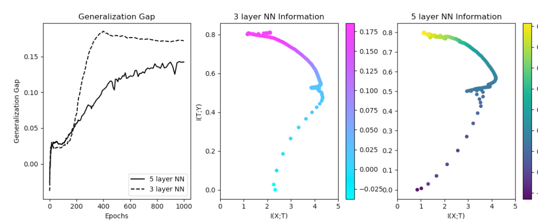

Figure 6: The learning curve shows that increased depth generalizes better. The information plane shows that deeper networks tend to compress further in the same training time.

6 Dynamic Batch Algorithm

Schwartz-Ziv et. al. find that, as a deep network learns, it goes through two phases: a "drift" phase where it rapidly minimizes error and decreases $I(T,Y)$, the MI between the outputs and the representation, and a diffusion phase, where it compresses the inputs by choosing representations with decreasing $I(X,T)$ [5]. Their intuition suggests that while the network is in the diffusion phase, it learns representations that generalize better. We try to leverage this intuition and devise an improved learning algorithm, which to our knowledge is new.

Our code is a modified version of the Information Bottleneck for Deep Networks package released by the authors of this paper [5] and we used MNIST data. Our network has 5 hidden layers with 10-8-8-5-3 neurons, sigmoid activations and initialized to small Gaussian values around zero. We train using Adam with a learning rate of 0.0004. Our mini-batch size is initialized at 512. As the network learns, the phase transition into the diffusion phase occurs when the normalized standard deviations of the gradient updates to the weights increases by about an order of magnitude, thus making the updates more noisy. This can be seen in Figure 7, right. We define a critical point when the std/mean of the gradients exceeds a threshold that we determined experimentally. At this critical point, we modify the learning strategy by dividing the batch size by a factor of 4. The intuition behind this is to make the gradient updates noisier, which, much like dropout, will help regularize the network and improve compression of the inputs. More explanation of why noise is crucial can be found in [5].

We find that this dynamic batch algorithm does indeed improve learning (final test error = 1.1%). We obtain a better test error compared to a static batch algorithm in which we do not change batch size (test error = 3.2%). In fact the test error seems to jump down soon after we change the batch size. Furthermore, the dynamic batch algorithm shows better compression of the input on the information plane, making it a promising technique for further study. The results are in Figure 7, left.

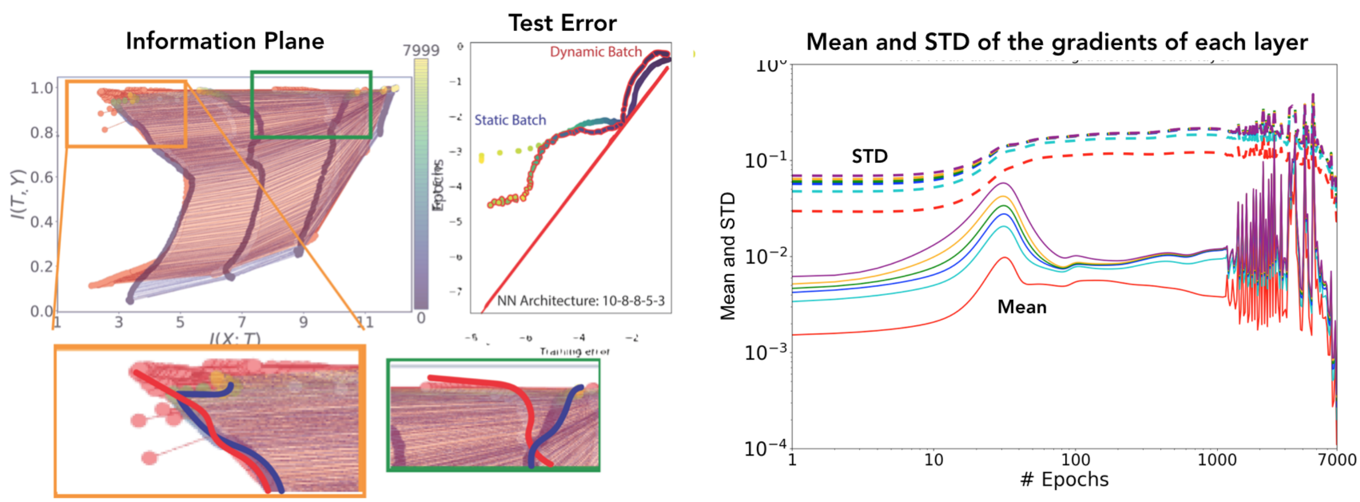

Figure 7: In the left image, on the Information Plane we see the dynamics of all the different layers of two networks as they learn MNIST. The layers are plotted with the later layers to the left. We can see the two phases where, in the earlier epochs, the layers rapidly ascend in the plane and in the later layers. they very slowly "diffuse" to the left. We compare the network with dynamic batch (in blue) and the network with static batch (in red). As can be seen in the insets, the dynamic batch compresses better in multiple layers. (The colors are sketched in for clarity). Again in the left image on the Test Error pane, the learning curve is plotted in the log-error plane (with test error on the y axis and train error on the x-axis). The diagonal line represents the test error tracking the train error perfectly. Epoch is color coded from blue to yellow. The curves correspond to the same two networks as before. No outline = fixed batch, red outline = dynamic batch. The dynamic batch algorithm generalizes better with lower test error for similar training error. The test error jumps down after the batch size change. In the right image we see the mean and standard deviation of the gradient updates (normalized by their L2 norms) for all the different layers and parameters. When the STDs jump up by about an order of magnitude, the gradient updates become noisier and the network enters the diffusion phase.

7 Discussion and Future Work

The information bottleneck provides an elegant way of thinking about the performance of deep networks and avenues to improve them. In this work, we provide proof of concept of one such technique, the dynamic batch algorithm. We hope to extend this work by investigating the robustness of the dynamic batch algorithm. We want to calculate statistics on the reliability of and improvement provided by the algorithm and see if it generalizes to different, more complicated datasets such as ImageNet. We also would like to explore other means of exploiting the two phases of learning by, for example, trying different ways of injecting noise in the diffusion phase (increasing dropout, adding Gaussian noise etc).

Contributions:

1. Daniel Kunin: Implemented the KDE MI estimates, the data initialization, a training skeleton for each of our models and the plotting architecture. Delved deeply into the practice of calculating MI to understand the shortcomings of each technique.

2. Mansheej Paul: Estimating the true MI of simulated data using monte carlo integration. IB theory and literature search. Implementing SVM for IB plane, modifying codebase in [5] to obtain results in the Bias Variance, Generalization Compression, Dynamic Batch Algorithm for neural networks sections.

3. Matthew Bull: Wrote the adaptive binning estimate to MI estimation. Adapted code from [5] to run locally [1]. Implemented learning curves, information plane, and plotting to compare standard metrics of performance to the information plane. Dynamic batch algorithm.

References

[1] Abadi, M et al ”Tensorflow: Large Scale machine learning for heterogenous distributed systems”, 2016 arXiv, URL`www.arxiv.org/abs/1603.04467`
`https://www.tensorflow.org/install/`

[2] Alex Alemi, Ian Fischer, Josh Dillon, and Kevin Murphy. Deep Variational Information Bottleneck. ICLR 2017. URL https://arxiv.org/abs/1612.00410.

[3] David Jordan, Seppe Kuehn, Eleni Katifori and Stan Liebler. PNAS 2013 1401814023, doi: 10.1073/pnas.1308282110

[4] Scott’s rule for KDE.
`https://docs.scipy.org/doc/scipy-0.15.1/reference/generated/scipy.stats.gaussian_kde.html`

[5] Ravid Shwartz-Ziv Naftali Tishby. Opening the black box of deep neural networks via information. arXiv: 1803.00810. Code provided in `https://github.com/ravidziv/IDNNs`

[6] Sci-Kit Learn, “Make Moons” retrieved from:
`http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_moons.html`

[7] Sci-Kit Learn, “KDE Cross Validation” retrieved from:
`http://scikit-learn.org/stable/modules/cross_validation.html`

[8] Slonim, Noam. The Information Bottleneck: Theory and Applications. PhD Thesis Hebrew University, 2002, pp. 2727., www.yaroslavvb.com/papers/slonim-information.pdf

[9] Suzuki, Taiji, et al. Mutual Information Approximation via Maximum Likelihood Estimation of Density Ratio. 2009 IEEE International Symposium on Information Theory, 2009, doi:10.1109/isit.2009.5205712.

[10] Tishby, Naftali, et al. The Information Bottleneck Method 1999 arXiv:physics/0004057v1