Information Bottleneck Theory: Extensions to Convolutional Neural Networks and the Stochastic Gradient Hypothesis

Candidate Number 8240R, Supervisor Dr Pietro Lio'

May 13, 2019

Abstract

This paper explores two possible implementations of information bottleneck theory to convolutional neural networks, the "concat" and "mean" methods. The results of this offer improvements over other methods of interpretability such as looking at activations. We show that loss dynamics can be explained by the mean of gradients rather than the signal to noise ratio as hypothesised in information bottleneck theory. We also argue that the arbitrariness of the measures used to calculate mutual information must be overcome for information theory to be a general theory of machine learning.

Contents

1 Introduction

There has been much progress in neural networks that out perform more traditional statistical methods in learning from data. A common issue and area of research in the field of neural networks is interpretability: being able to explain why a neural network does what it does and (relatedly) how the model will perform "in the real world".

One new attempt at a theory of neural networks is Information Bottleneck (IB) theory developed by Tishby and his collaborators [1, 2, 3]. Information Bottleneck Theory is an attempt to explain how networks learn and generalise, especially in the face of traditional statistical learning theory which predicts they would generalise poorly.

Concepts from IB theory also has the potential to be used as diagnostic tools to help understand how a given network is performing and if it's reached its performance "bound".

IB theory also contains a specific loss function that quantifies a networks information trade-off, which has inspired several new neural network architectures that are designed to use this IB loss.

In the light of this success of the theory there have also been several papers criticising it [4]. Several researchers have had difficulty replicating Tishby's computational findings even with his own code. Others have criticised it on more theoretical grounds.

This paper will focus on first trying to replicate its computational claims, especially around the idea of compression, then explore how it can be extended to convolutional neural networks, and finally present theoretical findings.

The next section covers the theory required to understand the results and discussion in Section 3. The paper is finally summarised in Section 4.

2 Theory

2.1 Neural Networks

This paper will look at a form of problem: given x what is y? For example predicting a house's price given its area.

If we had many examples of x and y pairs we could built a model that could make predictions given x. This is called a supervised learning problem in the literature.

One model to solve these types of problems is called a "neural network", named due to its inspiration from the neural structures seen in human brains.

Our neural network will be a function $f : x \to \hat{y}$ where x is the input into the model and $\hat{y}$ is the prediction of the model. We reserve y for the "ground truth" value that the model should have predicted.

The following background theory on neural networks is based on [5] though there are many other alternative resources available to learn more about the basic theory.

2.1.1 Neural Networks as Action on Vectors

Consider a simple problem of predicting a house price y given continuous valued variables about the house (area, number of bedrooms, age etc.) denoted $x_1, x_2, x_3 \dots x_N$ which can be represented as a vector $\mathbf{x}$. A simple model to predict $\hat{y}$ would be a multivariable linear regression of the form

$$\hat{y} = g_{\mathbf{w},b}(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = f(x)$$

where $\mathbf{w}$ and $\mathbf{x}$ are of the same dimension.

An extension to this approach is to consider the existence of hidden variables $\mathbf{z}$ that influence y, that we do not have access to, and that are related to the data we do have access to $\mathbf{x}$. We could then predict the values of the variables in $\mathbf{z}$ and use these as input into f to predict y.

$$\mathbf{z} = g_{\mathbf{W^{[1]}},\mathbf{b^{[1]}}}(\mathbf{x})$$

where since we now need to produce a vector $\mathbf{z}$ we use a matrix $\mathbf{W^{[1]}}$ and vector $\mathbf{b^{[1]}}$. The [1] indicates this is in the first layer of computation.

$$\hat{y} = g_{\mathbf{w}^{[2]}, b^{[2]}}(\mathbf{z}) = g_{\mathbf{w}^{[2]}, b^{[2]}}(g_{\mathbf{W}^{[1]}, \mathbf{b}^{[1]}}(\mathbf{x}))$$

We can now generalise to when we want to predict several variables $\mathbf{y}$:

$$\hat{\mathbf{y}} = g_{\mathbf{W}^{[2]}, \mathbf{b}^{[2]}}(\mathbf{z}) = g_{\mathbf{W}^{[2]}, \mathbf{b}^{[2]}}(g_{\mathbf{W}^{[1]}, \mathbf{b}^{[1]}}(\mathbf{x})) = f(\mathbf{x})$$

From here on we will drop the use of bolding and the nonscalar nature of variables is implied.

2.1.2 Nonlinearity and the Multilayer Perceptron

It turns out that applying layers of linear models still results in a linear model [5]. If we have reason to believe that $\hat{y}$ has a nonlinear relationship with x we can introduce nonlinearity into the model by using a nonlinear elementwise vector function ϕ.

The linear output of one layer is then passed through ϕ before being passed on to the next layer. The result of this is called the activation $a = \phi(z)$. We label the linear output of layer 1 as $z^{[1]}$. We notice the final output of the final layer $a^{[2]}$ is the prediction $\hat{y}$ and the input data x can also be labelled $a^{[0]}$. We write $\phi^{[i]}$ since each layer can have a different activation function.

$$\hat{y} = a^{[2]} = \phi^{[2]}(g_{W^{[2]}, b^{[2]}}(a^{[1]})) = \phi^{[2]}(g_{W^{[2]}, b^{[2]}}(\phi^{[1]}(g_{W^{[1]}, b^{[1]}}(x)))) = f(x)$$

To tidy things up we define $h^{[i]} : a^{[i-1]} \rightarrow a^{[i]} = \phi^{[i]}(z^{[i]}) = \phi^{[i]}(g_{W^{[i]}, b^{[i]}}(a^{[i-1]}))$.

$$\hat{y} = a^{[2]} = h^{[2]}(a^1) = h^{[2]}(h^{[1]}(a^0)) = h^{[2]}(h^{[1]}(x)) = f(x)$$

We can then add more and more layers to the model. For a model with N layers we get

$$\hat{y} = a^{[N-1]} = h^{[N-1]}(a^{N-2}) = h^{[N-1]}(h^{[N-2]}(\ldots h^{[2]}((h^{[1]}(a^0))) = f(x)$$

This type of network is a special type of neural network called a multilayer perceptron ("MLP").

For the following sections we will drop use of square brackets for layers and use the index l instead. Also there are many different notations for the layer random variables: Z, M and T. We will use them interchangeably.

2.1.3 Convolutional Neural Networks

We can extend the idea of neural networks to grey scale image data. A common network architecture used for image problems is the convolutional neural network (CNN). Each layer in a CNN has a corresponding width, height and depth.

The width and height are the width and height of the images in that layer. The depth is given by the number of images in that layer. For example an image with 28x28 pixels from the MNIST dataset has dimensions (28,28,1).

The first layers of a CNNs are layers called convolutional layers, which do the majority of the processing specific to image analysis. Each node in a convolutional layer has a corresponding filter. The number of filters a convolutional layer has is its depth. A filter has small width and height but maximum depth; it has the same depth as the previous layer. An image element in a convolutional layer is the convolution of its corresponding kernel with the previous layer.

Much more in depth explanations of CNNs can be found in [6] and other places.

2.1.4 Learning

For a given problem like our house price example we also need to define what a "good" prediction of the network is. This is done through a loss function $\mathcal{L}(y, \hat{y})$. The loss function has a low value when the prediction is good and a high value when the prediction is bad. To keep things tidy we refer to the value of the loss as $\mathcal{L}$. An example of a loss function would be the the mean square error $MSE = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2$ where (i) corresponds to one example e.g. one house.

The hope is that we can see how changing elements in W and b affects $\mathcal{L}$ and then change W and b to reduce $\mathcal{L}$.

2.1.5 Gradient Descent

Gradient descent is a family of methods to reduce the loss by updating the weights and biases.

Gradient descent uses a simple rule:

$$W_{jk}^{[l]} \leftarrow W_{jk}^{[l]} - \alpha \frac{\partial \mathcal{L}}{\partial W_{jk}^{[l]}}$$

$$b_j^{[l]} \leftarrow b_j^{[l]} - \alpha \frac{\partial \mathcal{L}}{\partial b_j^{[l]}}$$

where α is a hyperparameter called the learning rate.

This is usually summarised into one more general equation:

$$\theta \leftarrow \theta - \alpha \frac{\partial \mathcal{L}(\theta)}{\partial \theta} = \theta - \alpha \boldsymbol{\nabla}_\theta \mathcal{L}(\theta)$$

where θ stands for the W and b of the layers, which are together called the "parameters".

The family of methods in gradient descent is:

- Full Gradient descent ("FGD" also just called gradient descent) calculates $\mathcal{L}$ for the entire training dataset, $\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}^{(i)}$. However this becomes more computationally expensive to evaluate as N increases.
- Stochastoc Gradent Descent ("SGD") use the loss of one randomly chosen training sample to calculate $\mathcal{L} \approx \mathcal{L}^{(i)}$ during each update cycle. This has the advantage of not being computationally expensive but has the disadvantage of being a poor estimator of the true loss $\mathcal{L}$ with high variance.
- Minibatch Gradient Decent ("MGD") is a compromise between these two extremes. This approximates the true loss by calculating the loss on a sample of examples i.e. $\mathcal{L} \approx \frac{1}{M} \sum_{i=1}^{M} \mathcal{L}^{(i)}, M < N$.

2.1.6 Back Propagation

The partial derivatives are calculated in a process called backpropagation, which relies on the chain rule to find the derivative of layers closer to the data by "chaining" through layers starting at the layer closest to the prediction.

The equations governing backpropagation can be shown to be [7][1]:

$$\frac{\partial \mathcal{L}}{\partial W_{jk}^{[l]}} = a_j^{[l-1]} \gamma_k^{[l]}$$

$$\frac{\partial \mathcal{L}}{\partial b_j^{[l]}} = \gamma_j^{[l]}$$

$$\gamma_j^{[l]} \equiv \frac{\partial \mathcal{L}}{\partial z_j^{[l]}} = \frac{\partial \mathcal{L}}{\partial a_j^{[l]}} \phi_j'^{[l]} = \begin{cases} \frac{\partial \mathcal{L}}{\partial a;[L]}_j \phi_j'^{[L]} & \text{for prediction layer} \\ \sum_k W_{jk}^{[l+1]} \gamma_k^{[l+1]} \phi_j'^{[l]} & \text{for all other layers} \end{cases}$$

where ϕ' is the derivative of ϕ, and $\frac{\partial \mathcal{L}}{\partial a_j^{[L]}}$ is known.

2.2 Information Theory

This section will explain key concepts in information theory used in this paper. There are many good sources on information theory including [8].

[1]Nielsen's w is the transpose of our W.

2.2.1 Entropy

The entropy of a discrete random variable X is

$$H(X) = -\sum_{x} P(x) \log P(x) \geq 0$$

where the inequality comes from $0 \leq P(x) \leq 1$.
For two discrete random variable X and Y the joint entropy is

$$H(X,Y) = -\sum_{x,y} P(x,y) \log P(x,y)$$

Furthermore the conditional entropy for X given Y is

$$H(X|Y) = -\sum_{x,y} P(x,y) \log P(x|y) = -\sum_{x,y} P(x,y) \log \frac{p(y)}{P(x,y)}$$

The entropy "chain rule" relates these quantities

$$H(X,Y) = H(X) + H(Y|X) = H(Y) + H(X|Y)$$

2.2.2 Differential Entropy

Differential entropy is the continuous random variable analogue of entropy.
For a random variable X with probability density function $f(x)$ the differential entropy is

$$h(X) = -\int f(x) \log f(x) \mathrm{d}x$$

which can be negative.

$$h(X,Y) = -\int f(x,y) \log f(x,y) \mathrm{d}x\mathrm{d}y$$

$$h(X|Y) = -\int f(x,y) \log f(x|y) \mathrm{d}x\mathrm{d}y = -\int f(x,y) \log \frac{f(x,y)}{f(y)} \mathrm{d}x\mathrm{d}y$$

$$h(X,Y) = h(X) + h(Y|X) = h(Y) + h(X|Y)$$

2.2.3 KL Divergence

The Kullback–Leibler divergence ("KL divergence") is quantity between two distributions. If these distributions are discrete and labelled P and Q then the KL divergence from P to Q is

$$D_{KL}(P||Q) = \sum_{x} P(x) \log \frac{P(x)}{Q(x)} \geq 0$$

A proof for the inequality can be found in [8]. The KL divergence from P to Q can be interpreted as the information gained if we use Q instead of P [9].

2.2.4 Mutual Information

Mutual information is a quantity between two random variables. If these variables are labelled X and Y then the mutual information is

$$I(X;Y) \equiv D_{KL}(P(x,y)||P(x)P(y))$$

The mutual information is how much our uncertainty in X reduces after observing Y [10, 8].

For continuous random variables X and Y:

$$I(X;Y) = \int_{x,y} P(x,y) \log \frac{P(x,y)}{P(x)P(y)} \mathrm{d}x\mathrm{d}y = I(Y;X)$$

For discrete random variables X and Y:

$$I(X;Y) = \sum_{x,y} P(x,y) \log \frac{P(x,y)}{P(x)P(y)} = I(Y;X)$$

2.2.5 Markov Chain

A feedforward neural network with N hidden layers (the Z layers) can be treated as the Markov Chain

$$Y \leftrightarrow X \leftrightarrow Z^{[1]} \leftrightarrow \ldots \leftrightarrow Z^{[N]} \leftrightarrow \hat{Y}$$

where Y is the ground truth we are trying to predict, X is the input data vector, $Z^{[l]}$ is the activation vector of a layer l and $\hat{Y}$ is the prediction of the network [2].

2.2.6 The Data Processing Inequality

The Data Processing Inequality ("DPI") states that for a Markov Chain $A \leftrightarrow B \leftrightarrow C$ that

$$I(A;B) \geq I(A;C)$$

2.3 The Information Bottleneck

The main theory in this Section is the information bottleneck theory developed by Tishby in [1, 2, 3]. Figures in this section to illustrate concepts will come from these papers, as it is very hard to replicate Tishby's results, though this remains an area of controversy. Some of the finer details of the theory are not described in the papers and so must be inferred from the accompanying code, though the code still seems to be in production and therefore is difficult to read. A better maintained implementation of IB code accompanies Saxe's paper [4] at [11].

2.3.1 The Information Plane

One of Tishby's key ideas was to represent a DNN on the "information plane" (IP), a 2-dimensional space with the x value given by $I(X;T^l)$ and y value given by $I(Y;T^l)$.

As a consequence of the DPI, each layer in a DNN is positioned in the information plane down and to the left of the layer upstream of it as shown in Figure 1.

2.3.2 Dynamics in the Information Plane

IB theory stipulates training occurs in two stages. At the start of training each layer moves up and to the right, this is the learning stage. Then there is then a transition point when $I(X;T)$ of the layers starts to decrease, this is the compression phase. If there is sufficient training data then $I(Y;T)$ will continue to increase otherwise it can even decrease which is suggestive of overtraining as in Figure 1.

The last hidden layer is notable in Tishby's theory, highlighted by his information plane theorem:

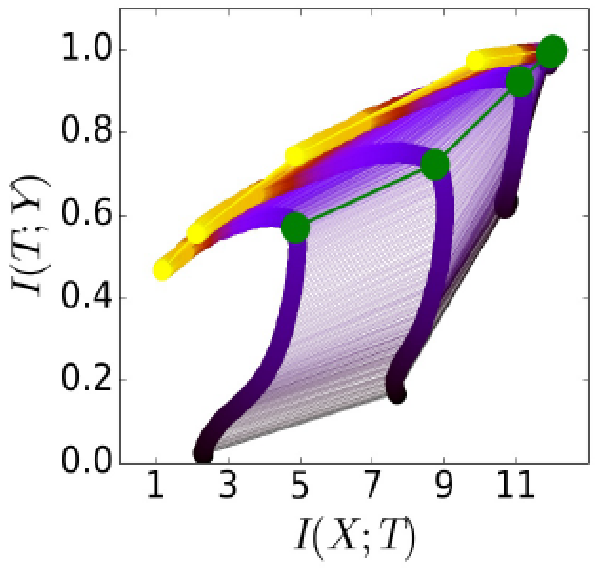


Figure 1: Figure from [3]. A small training set can lead to overtraining as shown by the decrease in the $I(T;Y)$ value of the final layer. This occurs after the transition (shown in green).

> For large typical X, the sample complexity of a DNN is completely determined by the encoder mutual information, $I(X;T)$, of the last hidden layer; the accuracy (generalization error) is determined by the decoder information, $I(T;Y)$, of the last hidden layer.

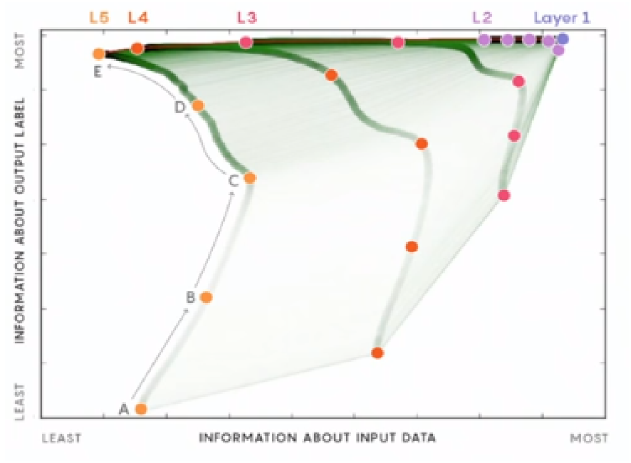


Figure 2: Figure from [3]. The dynamics of training of one network in the information plane. The network starts in an untrained state A and ends in a trained state E. As in Figure **??** each colour (this time from purple to orange) represents a different hidden layer.

2.3.3 Gradient Dynamics and SNR

Tishby's training set up for this analysis is not conventional so it should be explained. To produce a gradient plot like Figure 3, we must train a series of neural networks in parallel with identical architecture on different minibatches of the training data. Each neural network has one minibatch, and only that minibatch to train on. This is not conventional since in minibatch gradient descent one neural network is trained on a random minibatch of the training data at each timestep.

It'll be useful to fully define quantities in a verbose form. For a neural network training on the minibatch of data b

- θ_i^{lbt} is one of the parameters in layer l at time t (this includes elements in the weight matrix and elements in the bias vector)
- $\mathcal{L}^b$ is the loss calculated on the training minibatch b

Then the gradient is

$$\frac{\partial \mathcal{L}^b}{\partial \theta_i^{lbt}} \equiv \boldsymbol{\nabla} \theta_i^{lbt}$$

The mean and standard deviation vectors over the minibatches are

$$\mu_i^{lt} \equiv \langle \boldsymbol{\nabla} \theta_i^{lbt} \rangle_b$$

$$\sigma_i^{lt} \equiv \text{STD}(\boldsymbol{\nabla} \theta_i^{lbt})_b$$

We then calculate their magnitudes by using the Euclidean norm.

$$\mu^{lt} \equiv \| \langle \boldsymbol{\nabla} \theta_i^{lbt} \rangle_b \|$$

$$\sigma^{lt} \equiv \| \text{STD}(\boldsymbol{\nabla} \theta_i^{lbt})_b \|$$

We can then define the signal to noise ratio ("SNR") as

$$\text{SNR}^{lt} = \frac{\mu^{lt}}{\sigma^{lt}}$$

We can then make plots of μ, σ and SNR, e.g. as in Figure 3.

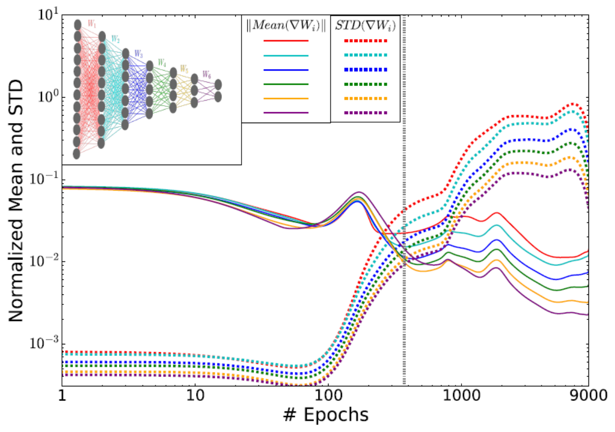


Figure 3: Figure from [3]. The dynamics of the weight gradients for each layer in a DNN. The DNN architecture is input=12-10-7-5-4-3-2=output as shown in the top left. Each colour represents a different weight gradient matrix.

The idea is that when training with MGD the gradients will then be stochastic with mean and standard deviation as shown in Tishby's simulations.

2.3.4 Stochastic Gradient Hypothesis

Tishby noticed the following 3 things occurred simultaneously in his computations:

1. The training loss "kinked" as a function of training time
2. The signal to noise ratio of the gradients in each layer rapidly decreased
3. The mutual information with the data $I(X;T)$ stopped increasing and started decreasing for each layer

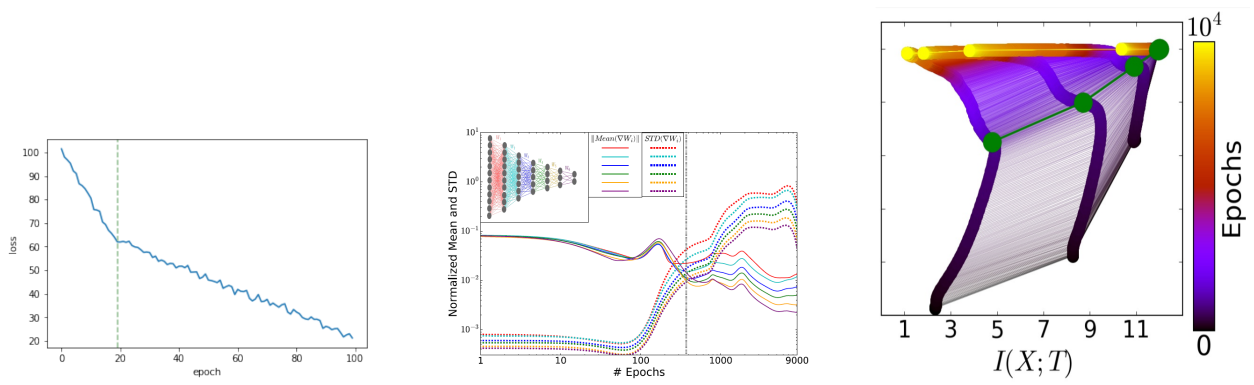


Figure 4: The three parts of the stochastic gradient hypothesis of a DNN trained by MGD. [Left] A kink in the training loss. [Middle] A regime change in the gradients as their signal to noise ratio decreases for each layer. [Right] Compression observed inn the information plane as the mutual information with the data decreases.

The first observation is incredibly useful for anyone who trains a DNN as loss graphs are very easy to produce from training and require no extra compute since the loss needs to be calculated to train anyway.

The second suggests that BGD/SGD may have unique properties compared to FGD beyond low compute requirements: using BGD/SGD may lead to completely different types of training regimes. Note the SNR in FGD can't change since the noise is zero always.

The final observation directly relates to IB theory. If these three events are causally linked in networks this would suggest IB theory is quite a broad and general theory of learning.

Tishby hypothesised these are causal connected due to the stochastic nature of the gradients in the learning process. In this paper we will refer to this hypothesis as the stochastic gradient hypothesis, though Tishby has not yet named it himself.

The theory part of the Results Section of this paper will go over novel theoretical results from trying to find causal links between these three events.

3 Results and Discussion

The results of this paper are split into two main sections. First results from generalising IB theory to convolutional neural networks, and second theoretical analysis and results of some of the claims of IB theory especially around the stochastic gradient hypothesis.

3.1 Generalising IB Theory to CNNs

IB work thus far has focused on multilayer perceptions, however this heavily limits the theory. Part of the reason for the focus on MLPs rather than CNNs is due to issues with the mutual information.

The issue lies in that mutual information is calculated between layers with depth equal to 1. In the case of a MLP all the layers have a depth of 1 so this is not an issue, but this is an issue for CNNs.

When it come sto how to apply mutual information to CNNs one very recent approach has been to use multivariate mutual information quantities [12]. This paper will look at two alternative methods: 1. to concatenate all the elements in a convolutional layer along the depth axis, thereby reducing it's depth to 1 and then calculating its mutual information ("concat") and 2. to take the mutual information of each element along the depth axis in the convolutional layer and calculate the layer mutual information as the mean of these mutual informations ("mean").

We will then compare these methods to a common method of interpreting neural networks: looking at the activations of the layers [6].

All code was run on external Google Colab GPUs. The code for this project is available for other researchers to build upon at my GitHub [2].

3.1.1 Replicating IP curves for MLPs

We will first show that our mutual information code works as expected by comparing our results to those produced by Saxe's code [11]. A comparison of IP curves produced by our code and Saxe's code is shown in Figure 5 showing strong agreement.

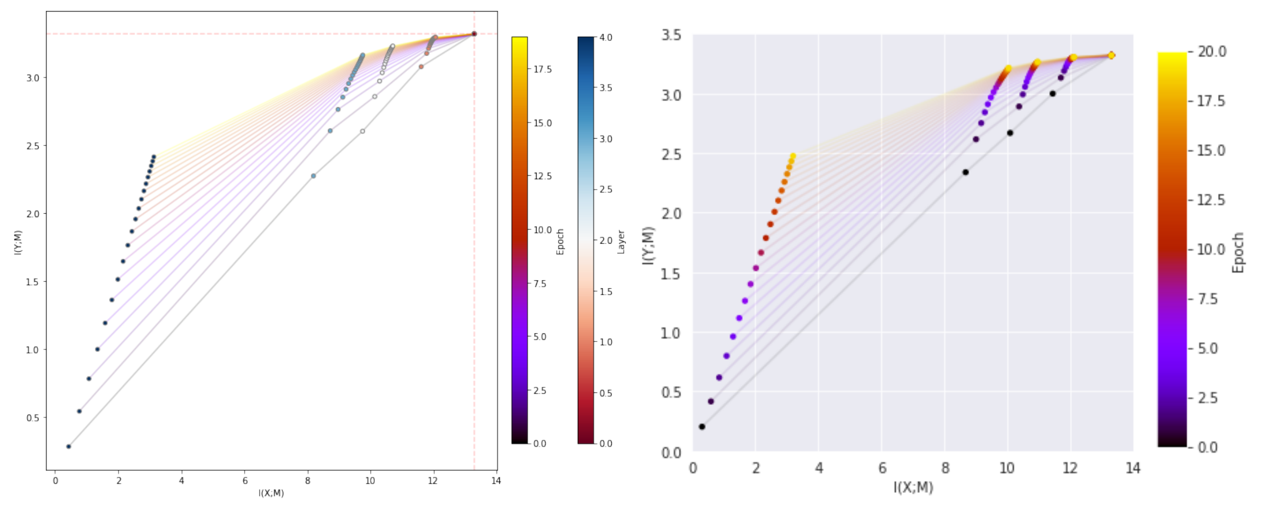


Figure 5: Comparison of IP curves between our code [left] and Saxe's [right]. These curves are for the same MLP with architecture: input=784-1024-20-20-20-10=output with tanh activations for the hidden layers and softmax activation for the prediction layer. The network was trained on a flattened version of the MNIST dataset using MGD with batch size 128 and learning rate 0.001. The network was trained for 20 epochs. [Left] Our results. The IP curves are bound by lines $x = 0$, $x = H(X)$, $y = 0$, $y = H(Y)$ as shown by the dashed red lines. The dot colour corresponds to the layer index and the line colour corresponds to the time step. For the KDE we used the same parameter as Saxe $\sigma^2 = 1e - 1$ [Right] Saxe's results from running his code with default settings.

[2]Including a link in this report would give away my anonymity since the GitHub is attached to my name.

Our code has built on from Saxe's code and offers considerable speed improvements. For example the time taken between compiling the network and getting our plot in Figure 5 was 50s for our program, but was 4m50s to make Saxe's plot in the same figure. Both were run on a Google Colab GPU.

3.1.2 CNN Set Up

Exploratory analysis was primarily done with a CNN network architecture from the Keras GitHub [13]. The architecture is described by the code in Listing 1, dropout layers were removed for easier comparison with a MLP, but repeating the experiments with dropout layers did not change the results.

Listing 1: CNN Architecture

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
                 input_shape=input_shape))  # input_shape depends on data
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
          # num_classes depends on data
```

The model was trained on the MNIST database with a batch size of 128, using the categorical cross entropy loss and Adadelta optimiser. After 12 epochs the model reaches 99.25% test accuracy. Similar IB results were also observed with other CNN architectures.

We tuned the σ^2 parameter in the KDE so the KDE values matched the analytical entropy of the MNIST labels ($\log_2 10$) within the uncertainty of the KDE, this gave $\sigma^2 = 0.0735$. We also made sure we used a large enough sample from the data set to calculate entropies by making sure the information quantities did not change for larger and larger samples (the information quantities stabilise for sample size > 100).

3.1.3 IP curves for Concat Method

See Figure 6 and Figure 7 for the IP curves produced by the concat method.

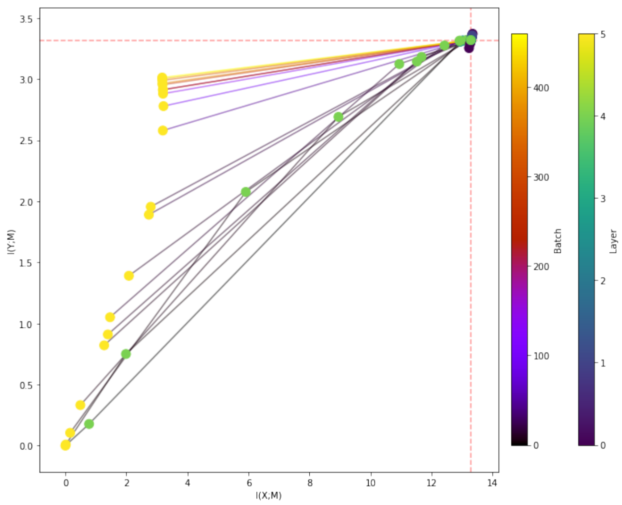
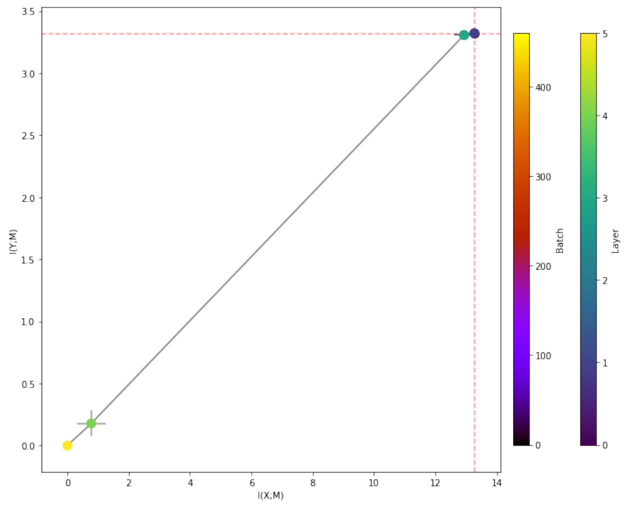


Figure 6: IP curves using concat method with CNN. We see layers 0, 1, 2, 3 all start and remain in the top right corner at maximum MI values. Layer 4 (the last hidden layer) also moves to maximum mutual information. Layer 5 (the prediction layer) starts with 0 mutual information with the input or target; its MI with the target and data increases during training in a similar way to the prediction layer in the MLP.

3.1.4 IP curves for Mean Method

See Figure 8 and Figure 9 for the IP curves produced by the mean method.



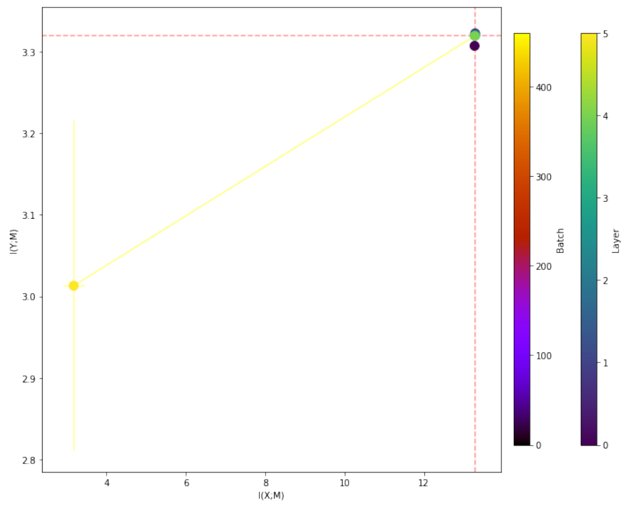


Figure 7: IP curves from the concat method before training [left] and after training [right]. The uncertainty in the KDE estimator is shown with the error bars. These plots are sampled from Figure 6.

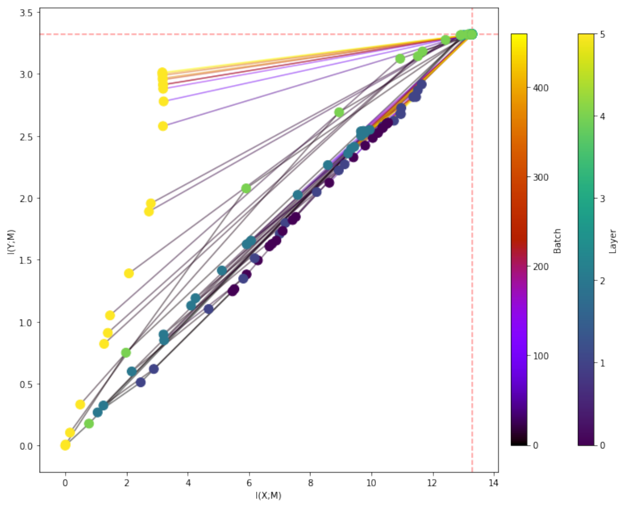


Figure 8: IP curves using concat method with CNN. The results are the same for the dense layers compared to the concat method (by definition of the methods). However we can now see more detail for the convolutional layers. These layers seem to cluster around the line from the origin to the point to top right point of maximum MI.

Compared to the concat method, the mean method has much higher compute requirements since the mutual information function is called for each element along the depth axis in a layer rather than once for the whole layer.

3.1.5 IP curves Compared to Activation Method

One of the ways to try an understand a CNN is working by looking at the activations of each layer in the network for given input data examples [6]. We'll refer to this as the activation method.

Our results from the activation method are shown in Figure 10. Similar results were observed for other input images.

3.1.6 Discussion of Computational Results

We didn't observe the compression phase that IB theory would predict even for large number of epochs, which agrees with Saxe's results [4]. This suggests the compression phase is not a general feature of training with MGD.

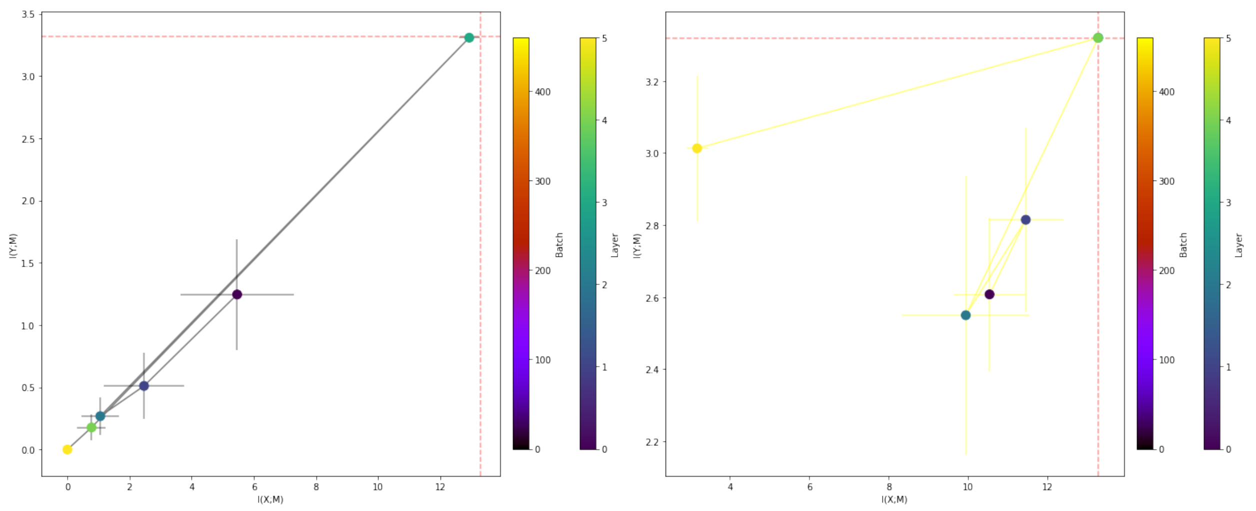


Figure 9: IP curves from the mean method before training [left] and after training [right]. The uncertainty in the KDE estimator is shown with the error bars. These plots are sampled from Figure 8.

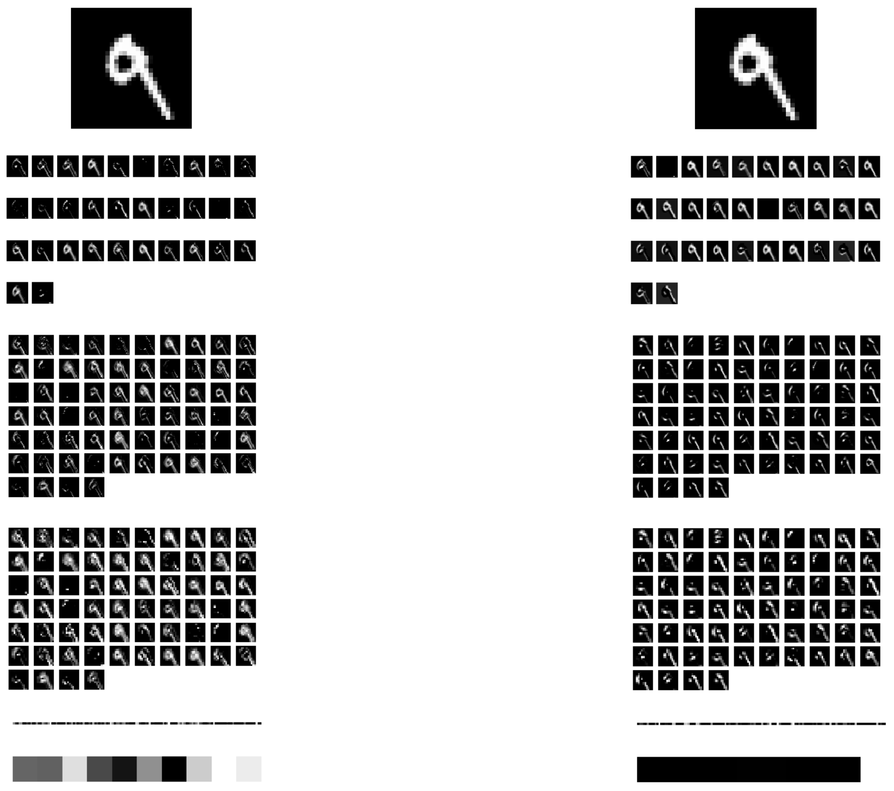


Figure 10: Activations of each layer [top to bottom] in our CNN from an example [top], before training [left] and after training [right]. Values are shown on a greyscale where 100% white is the maximum value in a layer and 0% white is 0. Without looking at the prediction layer it is hard to tell which corresponds to the network before training and which corresponds to the trained network.

The concat method put all the convolutional layers in the top right corner at maximum MI. This means the data processing inequality is not violated, but means no information can be gathered about the convolutional networks during training since they do not move in the information plane.

The mean method in comparison does not confine the convolutional layers to the top right. But they don't seem to move deterministicly and at the start and end of training they are in the top right. This also means it breaks the data processing inequality, though this is permitted for pseudo mutual information [4]. More research should be done to see how the convolutional layers move during training and why they cluster on the line between the origin and maximum information.

The main effect of training was an increase in the mutual information with the target of the prediction layer and last hidden layer as predicted by IB theory.

Though the mutual information methods are based on activations they give clearer results than the activation method when looking at the last hidden layer. It's not clear why the last hidden layer is the one that so clearly evolves, though this may have to do with its importance in IB theory. Overall a more refined method of IP analysis could be a useful alternative or additional tool for the activation method.

3.2 Theoretical Results

This section will try and prove novel theoretical results relating to IB theory and the stochastic gradient hypothesis.

3.2.1 Link between the SNR and Loss Dynamics

For minibatch gradient descent with minibatch loss $\tilde{\mathcal{L}}$:

$$\frac{\partial \theta}{\partial t} = -\alpha \frac{\partial \tilde{\mathcal{L}}}{\partial \theta}$$

The derivative of the loss with respect to training time is:

$$\begin{aligned}
\frac{\partial \mathcal{L}(\{x\}, \{y\}, \{\theta\}, \{\phi\}, \mathcal{A})}{\partial t} &= \sum_{\text{all }\theta} \frac{\partial \mathcal{L}}{\partial \theta} \frac{\partial \theta}{\partial t} \\
&= -\alpha \sum_{\text{all }\theta} \frac{\partial \mathcal{L}}{\partial \theta} \frac{\partial \tilde{\mathcal{L}}}{\partial \theta} \\
&= -\alpha \sum_{\text{all }\theta} \boldsymbol{\nabla}\theta \widetilde{\boldsymbol{\nabla}\theta}
\end{aligned}$$

where $\{x\}$ and $\{y\}$ are the total sets of training examples, $\{\theta\}$ is the set of parameters, $\{\phi\}$ is the set of activation functions for each layer and $\mathcal{A}$ is the architecture of the network (which nodes in each layer are attached to which). The loss is fully described by $\mathcal{L}(\{x\}, \{y\}, \{\theta\}, \{\phi\}, \mathcal{A})$ with respect to time (see the Appendix for a long form derivation for FGD). For MGD only $\{\theta\}$ changes with time [3].

We model the gradients as distributed in a normal distribution $\widetilde{\boldsymbol{\nabla}\theta} \sim \mathcal{N}(\mu, (\sigma)^2)$.

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial t} &= -\alpha \sum_{\text{all }\theta} \boldsymbol{\nabla}\theta \widetilde{\boldsymbol{\nabla}\theta} \\
\mathbb{E}\left[\frac{\partial \mathcal{L}}{\partial t}\right] &= -\alpha \sum_{\text{all }\theta} \boldsymbol{\nabla}\theta \left[\widetilde{\boldsymbol{\nabla}\theta}\right] \\
&= -\alpha \sum_{\text{all }\theta} \mu^2
\end{aligned}$$

Therefore changes in the loss can be explained simply by the mean of the gradients rather than their signal to noise ratio.

[3] The networks Tishby considered to not involve changes such as dropouts etc.

3.2.2 Link between Mutual Information and SNR

We attempted to find a link between the mutual information and the gradients but instead discovered issues with the use of mutual information in information bottleneck theory, which we describe in the next section.

3.2.3 Issues with Mutual Information in IB Theory

IB Theory relies on mutual information being well defined for the network. However most neural networks are deterministic i.e. $T = g(X)$. This means that for continuous X:

$$I(X;T) = I(X, g(X)) = \infty$$

Therefore to have a useful theory we need a "pseudo" mutual information that remains finite.

This is what Tishby unintentionally did by using "binning", a method of approximating the continuous distributions in the network by using discrete bins. This allows us to write

$$\tilde{I}(X; g(X)) = H(X) - H(X|g(X)) = H(X)$$

using the fact the conditional entropy given a deterministic function is 0.

This can also be done by by adding noise which is the approach taken with the kernel density estimator (KDE).

$$\begin{gathered}\tilde{T} \equiv g(X) + \xi \\ \tilde{I}(X;T) \equiv I(X, \tilde{T}) = \text{finite}\end{gathered}$$

Unfortunately the choice of pseudo mutual information is arbitrary; binning MI depends on the bins used and noise estimators depend on the noise parameters.

3.2.4 Analysis of Mutual Information with the KDE

An example of a pseudo mutual information we could use is the kernel density estimator (KDE).

It has been shown that if we define the following function for a layer

$$f(\nu) = \frac{1}{N} \log N \sum_{ij} \exp\left\{-\frac{1}{2}\frac{\|a^i - a^j\|_2^2}{\nu\sigma^2}\right\}$$

where a^i is the activity vector of the layer for data example x^i, N is the number of training examples, $\|\cdot\|_2$ is the Euclidean norm, and σ is a parameter we can choose.

Then the mutual information of a discrete distributed layer with an input is bounded between $f(1)$ and $f(4)$ i.e. $f(4) \leq \tilde{I}(X;T) \leq f(1)$.

We now try and find conditions for when the time derivative of f is zero, as this is the onset of the compression phase in IB Theory.

$$\begin{aligned}\frac{\partial f}{\partial t} &= \frac{1}{N} \log N \sum_{ij} \frac{\partial}{\partial t}\left(\exp\left\{-\frac{1}{2}\frac{\|a^i - a^j\|_2^2}{\nu\sigma^2}\right\}\right) \\ &= \frac{1}{N} \log N \sum_{ij} \left(-\frac{1}{2}\frac{\|a^i - a^j\|_2^2}{\nu\sigma^2}\right) \exp\left\{-\frac{1}{2}\frac{\|a^i - a^j\|_2^2}{\nu\sigma^2}\right\} \frac{\partial}{\partial t}\left(-\frac{1}{2}\frac{\|a^i - a^j\|_2^2}{\nu\sigma^2}\right) \\ &= \sum_{ij} \frac{1}{2} K(a^i, a^j) \frac{\partial}{\partial t}\left(\sum_k (a_k^i - a_k^j)^2\right) \\ &= \sum_{ij} K(a^i, a^j) \sum_k (a_k^i - a_k^j) \left(\frac{\partial a_k^i}{\partial t} - \frac{\partial a_k^j}{\partial t}\right)\end{aligned}$$

where

$$
\begin{aligned}
K(a^i, a^j) &= 2\frac{1}{N}\log N\left(-\frac{1}{2\nu\sigma^2}\right)\left(-\frac{1}{2}\frac{\|a^i - a^j\|_2^2}{\nu\sigma^2}\right)\exp\left\{-\frac{1}{2}\frac{\|a^i - a^j\|_2^2}{\nu\sigma^2}\right\} \\
&= \frac{\log N\|a^i - a^j\|_2^2}{2N\nu^2\sigma^4}\exp\left\{-\frac{1}{2}\frac{\|a^i - a^j\|_2^2}{\nu\sigma^2}\right\} \\
&> 0
\end{aligned}
$$

It is a necessary but not sufficient condition for $f < 0$ that for at least one combination of i, j, k we have:

$$
\begin{aligned}
0 &> (a_k^i - a_k^j)\left(\frac{\partial a_k^i}{\partial t} - \frac{\partial a_k^j}{\partial t}\right) \\
&> (a_k^i - a_k^j)\left({\phi'_k}^i\frac{\partial z_k^i}{\partial t} - {\phi'_k}^i\frac{\partial z_k^j}{\partial t}\right) \\
&> (a_k^i - a_k^j)\left({\phi'_k}^i\frac{\partial z_k^i}{\partial t} - {\phi'_k}^i\frac{\partial z_k^j}{\partial t}\right)
\end{aligned}
$$

The time derivative of z can be expressed in terms of the gradients (see Appendix) but it doesn't appear to fall into a nice analytical form.

3.2.5 Discussion of Theoretical Results

The main finding that the loss dynamics depend on the mean rather than the SNR is loosely suggestive that the SNR may not be as important to IB theory as originally proposed by Tishby. This is also supported by the fact that IB effects have been observed for networks trained by full batch gradient descent [4].

The arbitrary nature of pseudo mutual information highlights a potential challenge that IB theory must overcome to be a general theory of neural network learning. It is also this arbitrariness that has led to many publications and counter publications on whether observed features of mutual information are truly real.

Finally it seems additional research should be taken into any link between the mutual information and training dynamics in order to strengthen IB theory's foundations against theoretical criticism.

4 Conclusions

This paper has explored some possible implementations of information bottleneck theory to convolutional neural networks. The results of this are not as beautiful or helpful for interpretability as IB theory claims.

However compared to other methods like looking at activations, curves in the information plane may have additional utility. This code accompanying this paper applying IB theory to CNNs is publicly available for other researchers to build upon.

Our theoretical results show that the gradient signal to noise ratio is not required to explain loss dynamics as claimed by Tishby. We also highlight issues with the use of mutual information as a foundational part of a theory of neural networks due to the dependence on arbitrary decisions about the pseudo mutual information.

Finally we showed a potential for a link between the kernel density estimator of the mutual information and the gradients.

References

[1] N. Tishby, F. C. Pereira, and W. Bialek. The information bottleneck method. *arXiv Physics e-prints*, April 2000.

[2] N. Tishby and N. Zaslavsky. Deep Learning and the Information Bottleneck Principle. *arXiv e-prints*, March 2015.

[3] R. Shwartz-Ziv and N. Tishby. Opening the Black Box of Deep Neural Networks via Information. *arXiv e-prints*, March 2017.

[4] Andrew Michael Saxe, Yamini Bansal, Joel Dapello, Madhu Advani, Artemy Kolchinsky, Brendan Daniel Tracey, and David Daniel Cox. On the information bottleneck theory of deep learning. In *International Conference on Learning Representations*, 2018.

[5] Andrew Ng and Kian Katanforoosh. *CS229 Lecture Notes, Deep Learning*. Stanford Computer Science Department, 2018.

[6] Andrej Karpathy. *CS231n: Convolutional Neural Networks for Visual Recognition, Lecture Notes*. Stanford Computer Science Department, 2018.

[7] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[8] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. John Wiley and Sons, Inc, 1991.

[9] S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86, 03 1951.

[10] Claude E. Shannon and Warren Weaver. *THE MATHEMATICAL THEORY OF COMMUNICATION*. THE UNIVERSITY OF ILLINOIS PRESS, 1964.

[11] Code for on the information bottleneck theory of deep learning. `https://github.com/artemyk/ibsgd/tree/iclr2018`.

[12] S. Yu, K. Wickstrøm, R. Jenssen, and J. C. Principe. Understanding Convolutional Neural Networks with Information Theory: An Initial Exploration. *arXiv e-prints*, April 2018.

[13] mnist_cnn.py. `https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py`.

Appendices

.1 Proof of time derivative of loss

We have $z_i^l = b_i^l + \sum_k W_{ki}^l a_k^{l-1}$, from which we can calculate the following 4 derivatives

- $\frac{\partial z_i^l}{\partial b_j^l} = \delta_{ij}$
- $\frac{\partial z_i^l}{\partial W_{jk}^l} = a_j^{l-1}\delta_{ik}$
- $\frac{\partial z_i^l}{\partial a_j^{l-1}} = W_{ji}^l$
- $\frac{\partial z_i^l}{\partial t} = \frac{\partial b_i^l}{\partial t} + \sum_k W_{ki}^l \frac{\partial a_k^{l-1}}{\partial t} + \sum_k a_k^{l-1}\frac{\partial W_{ki}^l}{\partial t}$

where δ_{ij} is the Kronecker delta.

Since $a_i^l = \phi^l(z_i^l)$ we also have

- $\frac{\partial a_i^l}{\partial t} = \phi_i'^l \frac{\partial z_i^l}{\partial t}$
- $\frac{\partial \mathcal{L}}{\partial z_i^l} = \phi_i'^l \frac{\partial \mathcal{L}}{\partial a_i^l}$

where $\phi_i'^l = \frac{\partial \phi^l(z_i^l)}{\partial z_i^l}$.

From the equations from the Full Gradient Descent Section we also have:

- $\frac{\partial W_{ij}^l}{\partial t} = -\alpha \frac{\partial \mathcal{L}}{\partial W_{ij}^l}$
- $\frac{\partial b_i^l}{\partial t} = -\alpha \frac{\partial \mathcal{L}}{\partial b_i^l}$

We will also use the equations in the Back Propagation Section.

We define $\Gamma_i^l \equiv \gamma_i^l \frac{\partial z_i^l}{\partial t}$.

$$\begin{aligned}
\Gamma_i^l &\equiv \gamma_i^l \frac{\partial z_i^l}{\partial t} && (1)\\
&= \gamma_i^l \left[\frac{\partial b_i^l}{\partial t} + \sum_k W_{ki}^l \frac{\partial a_k^{l-1}}{\partial t} + \sum_k a_k^{l-1}\frac{\partial W_{ki}^l}{\partial t}\right] && (2)\\
&= \gamma_i^l \left[-\alpha \frac{\partial \mathcal{L}}{\partial b_i^l} - \sum_k \alpha \frac{\partial \mathcal{L}}{\partial W_{ki}^l} a_k^{l-1} + \sum_k W_{ki}^l \frac{\partial a_k^{l-1}}{\partial t}\right] && (3)\\
&= -\alpha \frac{\partial \mathcal{L}}{\partial b_i^l}\frac{\partial \mathcal{L}}{\partial b_i^l} - \sum_k \alpha \frac{\partial \mathcal{L}}{\partial W_{ki}^l}\frac{\partial \mathcal{L}}{\partial W_{ki}^l} + \gamma_i^l \sum_k W_{ki}^l \frac{\partial a_k^{l-1}}{\partial t} && (4)
\end{aligned}$$

We define $G^l \equiv \sum_i \Gamma_i^l \equiv \sum_i \frac{\partial \mathcal{L}}{\partial z_i^l}\frac{\partial z_i^l}{\partial t}$.

$$\begin{aligned}
G^l &\equiv \sum_i \Gamma_i^l && (5)\\
&= -\alpha \sum_i \frac{\partial \mathcal{L}}{\partial b_i^l}\frac{\partial \mathcal{L}}{\partial b_i^l} - \alpha \sum_{ki} \frac{\partial \mathcal{L}}{\partial W_{ki}^l}\frac{\partial \mathcal{L}}{\partial W_{ki}^l} + \sum_k \sum_i \gamma_i^l W_{ki}^l \frac{\partial a_k^{l-1}}{\partial t} && (6)\\
&= -\alpha \left\|\frac{\partial \mathcal{L}}{\partial W^l}\right\|^2 - \alpha \left\|\frac{\partial \mathcal{L}}{\partial b^l}\right\|^2 + \sum_k \frac{\partial \mathcal{L}}{\partial a_k^{l-1}}\frac{\partial a_k^{l-1}}{\partial t} && (7)
\end{aligned}$$

We also get

$$
\begin{aligned}
G^l &\equiv \sum_i \frac{\partial \mathcal{L}}{\partial z_i^l} \frac{\partial z_i^l}{\partial t} && (8) \\
&= \sum_i \phi_i'^l \frac{\partial \mathcal{L}}{\partial a_i^l} \frac{\partial z_i^l}{\partial t} && (9) \\
&= \sum_i \frac{\partial \mathcal{L}}{\partial a_i^l} \frac{\partial a_i^l}{\partial t} && (10)
\end{aligned}
$$

So

$$
G^l = -\alpha \left\| \frac{\partial \mathcal{L}}{\partial W^l} \right\|^2 - \alpha \left\| \frac{\partial \mathcal{L}}{\partial b^l} \right\|^2 + G^{l-1}
$$

If $l = 0$ then:

$$
\begin{aligned}
G^0 &= \sum_i \frac{\partial \mathcal{L}}{\partial a_i^0} \frac{\partial a_i^0}{\partial t} && (11) \\
&= \sum_i \frac{\partial \mathcal{L}}{\partial x_i} \frac{\partial x_i}{\partial t} && (12) \\
&= 0 && (13)
\end{aligned}
$$

If $l = L$ then:

$$
\begin{aligned}
G^L &= \sum_i \frac{\partial \mathcal{L}}{\partial z_i^L} \frac{\partial z_i^L}{\partial t} && (14) \\
&= \frac{\partial \mathcal{L}}{\partial t} && (15)
\end{aligned}
$$

In summary:

$$
G^l = \begin{cases} \frac{\partial \mathcal{L}}{\partial t} & l = L \\ -\alpha \left\| \frac{\partial \mathcal{L}}{\partial W^l} \right\|^2 - \alpha \left\| \frac{\partial \mathcal{L}}{\partial b^l} \right\|^2 + G^{l-1} & l \neq 0 \\ 0 & l = 0 \end{cases}
$$

Therefore:

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial t} &= -\alpha \left\| \frac{\partial \mathcal{L}}{\partial W^L} \right\|^2 - \alpha \left\| \frac{\partial \mathcal{L}}{\partial b^L} \right\|^2 + G^{L-1} && (16) \\
&= -\alpha \sum_{l=1}^{L} \left\| \frac{\partial \mathcal{L}}{\partial W^l} \right\|^2 - \alpha \sum_{l=1}^{L} \left\| \frac{\partial \mathcal{L}}{\partial b^l} \right\|^2 && (17) \\
&= -\alpha \sum_{l=1}^{L} \left(\frac{\partial \mathcal{L}}{\partial \theta^l} \right)^2 && (18) \\
&= -\alpha \sum_{\text{all } \theta} \left(\frac{\partial \mathcal{L}}{\partial \theta} \right)^2 && (19)
\end{aligned}
$$

.2 Alternative way to write G^l

If $l \neq L$ then:

$$G^l \equiv \sum_i \frac{\partial \mathcal{L}}{\partial z_i^l} \frac{\partial z_i^l}{\partial t} \tag{20}$$

$$= \sum_i \frac{\partial z_i^l}{\partial t} \sum_j W_{ij}^{l+1} \phi_i'^l \gamma_j^{l+1} \tag{21}$$

$$= \sum_{ij} \frac{\partial z_i^l}{\partial t} \phi_i'^l W_{ij}^{l+1} \gamma_j^{l+1} \tag{22}$$

$$= \sum_{ijk} \frac{\partial z_i^l}{\partial t} \phi_i'^l \phi_j'^{l+1} W_{ij}^{l+1} W_{jk}^{l+2} \gamma_k^{l+2} \tag{23}$$

$$= \sum_{i_l, i_{l+1}, i_{l+2}} \frac{\partial z_{i_l}^l}{\partial t} \phi_{i_l}'^l \phi_{i_{l+1}}'^{l+1} W_{i_l, i_{l+1}}^{l+1} W_{i_{l+1}, i_{l+2}}^{l+2} \gamma_{i_{l+2}}^{l+2} \tag{24}$$

$$= \sum_{i_l \dots i_L} \frac{\partial z_{i_l}^l}{\partial t} \phi_{i_l}'^l \dots \phi_{i_{L-1}}'^{L-1} W_{i_l, i_{l+1}}^{l+1} \dots W_{i_{L-1}, i_L}^L \frac{\partial \mathcal{L}}{\partial a_{i_L}^L} \phi_{i_L}'^L \tag{25}$$