Neural Networks (NN) are an extremely successful tool, they are widely adopted commercially and closely studied academically, however even given the attention they have there is no comprehensive understanding of how these models generalize data and provide such impressive performance – in fact very little is known about how NN learn or about their inner workings. Recently Prof. Tishby [?] produced a paper claiming to understand the basic principles of how NN work. He decided to examine NNs through the information domain. Tishby made the claim that the incredible performance of NNs is due to their ability to compress information. Compressing data means the network is only able to keep relevant input features and it must discard the irrelevant bits of information, leading to ability to generalize.

Tishby made interesting claims and provided experimental evidence to support his claims, however he did not provide a formal proof leaving his results up for debate. A paper release by Saxe [?] has contested the claim made by Tishby arguing that compression cannot happen in Neural Networks and the results are a consequence of the hyper parameters Tishby used. However, Saxe's paper suffers from the same problem as Tishby's as it does not provide a formal proof only experimental evidence – as such ti does not settle the rebuttal. To fully understand the discussion we need to understand the following topics Entropy and Mutual Information, Neural Networks, and Information Plane, described in section 0.1, section 0.2, and section 0.3.

0.1 Entropy and Mutual Information

0.1.1 Entropy

Entropy – quantifies information content of a random variable. It is generally measured in bits and can be though of as the expected information content when we sample a random variable once. Let X be a discrete random variable that can take values in $\{x_1, ..., x_n\}$. $H(X)$, the entropy of X, is defined by Equation 1.

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log P(x_i) \tag{1}$$

Consider a random variable Y s.t $P(Y = 1) = P(Y = 0) = 0.5$, Equation 1 defines $H(Y)$ to be 1.

Similarly for a random variable Y s.t $P(Y = 0) = 0.5, P(Y = 1) = P(Y = 2) = 0.25$, we have $H(Y) = 1.5$.

0.1.2 Conditional Entropy

Conditional Entropy – quantifies the amount of information needed to describe an outcome of variable Y given that value of another random variable X is already known. Conditional Entropy of Y given X is written as $H(Y|X)$. Let X be defined as before, Let Y be a discrete random variable that can take values in $\{y_i, ..., y_n\}$. The conditional entropy $H(Y|X)$ is defined by Equation 2.

$$H(Y|X) \ = -\sum_{x\in X, y\in Y} P(x, y) \log \frac{P(x, y)}{P(x)} \tag{2}$$

Let the correlated variables X and Y be defined by Table 1.

Y X	0	1
0	0.25	0.25
1	0.5	0

Table 1: Joint probability distribution for X and Y

Equation 1 and Equation 2 defines entropy values to be:

$$\begin{aligned} H(Y|X) &= 0.5 \\ H(X|Y) &\approx 0.6887 \\ H(X) &= 1 \\ H(Y) &\approx 0.8112 \end{aligned} \tag{3}$$

0.1.3 Mutual Information

Mutual Information (MI) – measures how much information two random variables have in common. It quantifies information gained about one variable when observing the other. Equation 4 and Equation 5 are two examples of how we can compute MI, using explicit probability computations or entropies of the random variables respectively, here X and Y are as previously defined.

$$I(X,Y) = \sum_{y \in Y} \sum_{x \in X} P(x,y) \log \left(\frac{P(x,y)}{P(x)\,P(y)} \right) \tag{4}$$

$$I(X,Y) = H(X) - H(X|Y) \tag{5}$$

For example of MI consider the random variables X and Y as before in the conditional entropy section – defined by Table 1.

We computed the entropy values in Equation 3, we will use them in Equation 5 to compute $I(X,Y)$.

$$I(X,Y) = H(X) - H(X|Y) \approx 1 - 0.6887 = 0.3113 \tag{6}$$

Properties of Mutual Information

There are some important properties of MI that we need to take note of. Let X and Y be any probability distributions then:

Commutative

$$I(X,Y) = I(Y,X) \tag{7}$$

Information Loss MI of two random variables cannot exceed entropy of either of them.

$$\begin{aligned} H(X) &\geq I(X,Y) \\ H(Y) &\geq I(X,Y) \end{aligned} \tag{8}$$

Data Processing Inequality Let u be some function; then,

$$I(X,Y) \geq I(X,u(Y)) \tag{9}$$

Invertible Transformation Let u be some invertible function; then,

$$I(X, Y) = I(X, u(Y)) \tag{10}$$

0.2 Neural Networks

Before we understand neural networks we need to understand The Prediction Problem and the purpose of Machine Learning Frameworks.

0.2.1 The Prediction problem

Suppose we have some dataset (x_i, y_i) for $i = 1, ..., N$. The prediction problem is finding a function f s.t Equation 11 is satisfied.

$$f(x_i) = y_i \text{ for } i = 1, ..., N \tag{11}$$

Prediction task is a common task that involves having input data $\{x_i, ..., x_N\}$ and finding the label, some desirable feature, $\{y_i, ..., y_N\}$.

The prediction problem could be simple to extract: for example if our input is a natural number $x_i \in \mathbb{N}$, and our label is either *true* or *false* depending if x is even or odd – in which case function defined by Equation 12 satisfies the problem.

$$g(x) = \begin{cases} True, & \text{if } \exists n \in \mathbb{N}.x = 2n, \\ False, & \text{otherwise.} \end{cases} \tag{12}$$

The prediction problem also can be impossible to solve: for example the halting problem, if our $x`s$ are programs and $y`s$ boolean values corresponding if the program halts or not.

Of course the prediction problem can be hard or impossible to solve as is the case for problems:

input data	label	difficulty
medical symptoms	diagnosis	intractable
picture	object in the picture	
face photograph	identity	
stock market history	future stock prices	
program	does the program halt	proved to be unsolvable
boolean equation	is the equation satisfiable	expensive to compute

Table 2: Example of specific prediction problems

Problems listed in Table 2 are either intractable, unsolvable or too expensive to compute – hence we cannot produce an algorithm that always give the correct answer and runs in a reasonable time.

0.2.2 Machine Learning Frameworks

If the Prediction problem is too difficult and we are tolerant to errors in our labels we may want to use a supervised machine learning framework[1] to tackle the problem.

[1] In this thesis we are exclusively talking about supervised machine learning techniques – the word "supervised" will be omitted for brevity

Every machine learning framework requires that we have some subset $\hat{X} \subseteq \{x_i, ..., x_N\}$ s.t that $\forall x \in \hat{X}$ we know label the y. A framework uses the data in order to reach some goal – such as minimizing the prediction error. The way any machine learning framework learns from data varies, but generally more data means an increase in prediction performance.

0.2.3 Neural Networks

Neural Networks (NN) are an example of a machine learning framework. They learn from data and attempt to solve the prediction problem.

The structure of a NN consists of layers of nodes, where every consecutive layer is fully connected as in Figure 1a. When we try to predict a label of a specific input every node gets assigned a value in the real number space $\mathbb{R}$. Values for the Input nodes are provided, whereas values for the nodes in non-input layers are generated based on nodes from the preceding layer. Let $n_{l,i}$ be the value that the $i'th$ node in layer l takes. Let $w_{l,j,i}$ be a parameter that influences how relevant the $j'th$ node in layer l is to the $i'th$ node in layer $l + 1$.

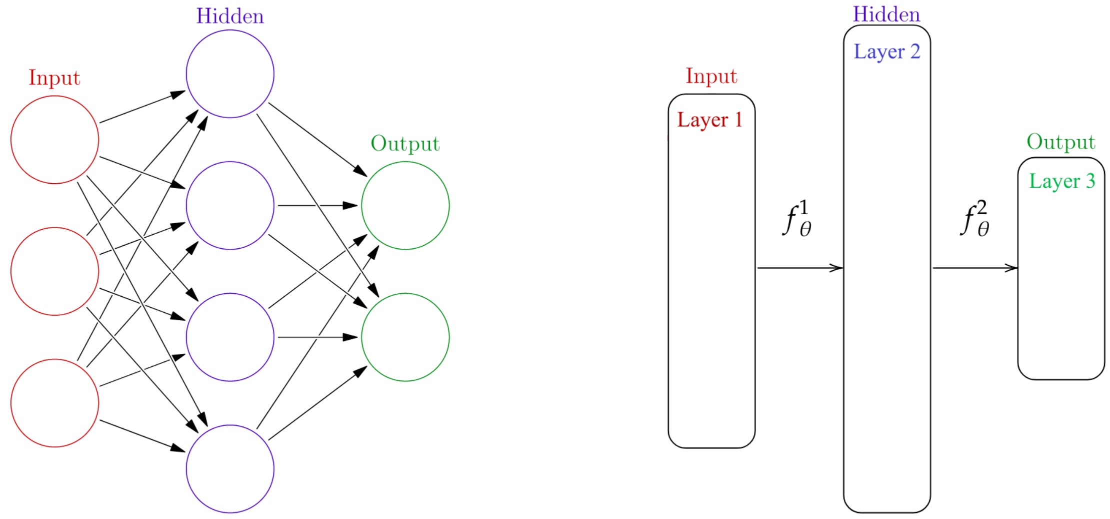
$$n_{l+1,i} = f(\sum_{j=0}^{\text{layer } l \text{ size}} w_{l,j,i} * n_{l,j}) \tag{13}$$

f in Equation 13 is called the activation function and is generally taken to be:

- ReLu: $f(x) = max(0, x)$
- Sigmoid: $f(x) = \frac{1}{1+e^x}$
- Tanh: $f(x) = tanh(x)$

Stochastic Gradient Descent (SGD) The NNs training algorithm is called Stochastic Gradient Descent. SGD periodically adjusts the parameters w according to some goal such as minimising the prediction error.

SGD is an iterative process – it introduces a notion of epochs where one iteration of the algorithm advances the epochs by one. Notion of epochs implies that the parameters w depends on which epoch we are currently at – thus it must take the epoch number e as an argument – $w(e)$.



(a) Structure of a typical neural network.

(b) Structure of our abstracted neural network.

Figure 1: Source: Wikimedia Commons

0.2.4 Abstracting the Neural Network

For our purposes we will abstract away the individual nodes in the NN and only consider the layer structure – as in Figure 1b.

In our abstraction the NN is structured in layers where every layer holds an intermediate representation of the final prediction output - let us call this intermediate representation an **activation** of that specific layer. We will formally define a neural network with L layers to be a sequence of L functions $f^1_{\theta(e)}, f^2_{\theta(e)}, ..., f^L_{\theta(e)}$ that are parameterized by θ s.t. Equations 14 hold. In our abstract the parameters θ correspond to the parameters w, therefore θ depends on the current epoch – $\theta(e)$. Let us also define an **activation** of layer l to be output of the function f^l_θ.

$$\begin{aligned} \text{let } t_0 &= x, \\ t_0 \to f^1_\theta(t_0) &= t_1, \\ t_1 \to f^2_\theta(t_1) &= t_2, \\ &... \\ t_{L-1} \to f^L_\theta(t_{L-1}) &= t_L, \\ \text{let } \hat{y} &= t_L \end{aligned} \tag{14}$$

values $t_1, t_2, ..., t_L$ here are **activations** of layers $1, 2, ..., L$,

x is any input to the NN from the set $\{x_i, ..., x_N\}$,

$\hat{y}$ is the prediction of the NN for the input,x which may or not be correct label,

the arrow ‘ $\to$ ’ signifies a transition from one NN layer to another.

We now have one function for each layer transition of the NN this allows us to extract the activation of a specific layer as in Equation 15.

$$t_l = f^l_\theta(f^{l-1}_\theta(...(f^1_\theta(x)))) \tag{15}$$

activtion of later l for input x

Let us define F^t_θ to be the activation of layer l given input x i.e

$$F^l_\theta(x) = f^l_\theta(f^{l-1}_\theta(...(f^1_\theta(x)))) \tag{16}$$

Figure 2 summarizes our NN definition.

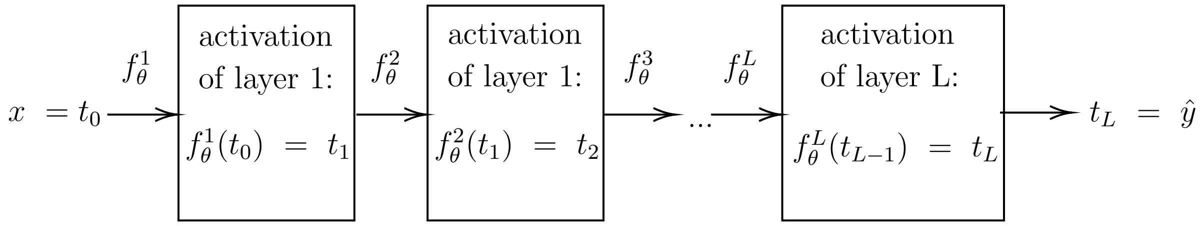


Figure 2: Visualization of a neural networks structure. x here is any input to the network from the set $\{x_i, ..., x_N\}$. $\hat{y}$ is the prediction of the network for the input x which may or may not be to the correct label. The values $t_1, ..., t_L$ here are **activations** of layers $1, ..., L$.

0.3 The Information plane

The information plane is a way of visualizing the Neural Network's training process through the information domain. Meaning we are looking what information the NN is retaining through out the layers, and how the information retained changes through out the training process.

We are interested see what information about the input and the label is retained – as having this information would help us understand how the neural network learns. Suppose we have this data available then we can examine this data with respect to:

- Epochs – lets us ask questions such as:
  - Is the network getting rid of noise in the network.
  - Does the network ever retain all the information about the label.
  - If the network does retain all the information about the label does training it after the point lead to performance increase.
- Layer – lets us ask questions such as:
  - Is there a difference between how much information the layers are discarding.
  - How does the information dynamics change if we add more layers.

Suppose we are examining a specific NN. At the start of the training process we expect the network to perform poorly – meaning it does not retain information about the label. Through the training process we expect:

- The information about the label to rise until it saturates. If we train past this saturation point we don't expect the information about the label to change much.
- The information about the input to rise until we have saturated the information about the label. At this point we either expect:

- The information about the input to keep rising if we believe the network starts to learn noise about the data.
- The information about the input to start reducing until the networks does not retain any noise about the input if we believe the network generalizes the data.

Ideally a neural network would retain all the information relevant to the label and none of the noise inherent in the input.

0.3.1 Setup

We want to compute what information about the label and the input the network is retaining – that is we want to compute the mutual information:

- Between the input distribution and all of the network layers,
- Between the label distribution and all of the network layers.

In order to compute the needed MI values we need to define the probability distributions:

- X - probability distribution of the input,
- Y - probability distribution of the label,
- $T_{e,l}$ - probability distribution of the layer l for the epoch e.

The routine in Figure 3 defines the correlated random variables $X, T_{e,l}, Y$. The probability distributions define the needed MI values: $I(X, T_{e,l})$ and $I(Y, T_{e,l})$.

```
def rxty(e, l):
  pick i $\sim$ Uniform {1...N}
  return $(x_i, F^l_{\theta(e)}(x_i), y_i)$
```

Figure 3: Definition of correlated random variables $X, T_{e,l}$ and, Y. The Probability distributions are generated from the dataset (x_i, y_i) for $i = 1, ..., N$. $F^l_{\theta(e)}$ is defined by Equation 16

Input Probability Distribution The probability distribution X is generated from the input dataset $\{x_i, ..., x_N\}$. The probability distribution has an inherent probability distribution but often we don't know it. We have not made any assumption about the input distribution – as such the Figure 3 takes every input value to be equally likely. If we had a specific dataset adjust our assumption and assign a different probability to the input dataset.

0.3.2 Visualization

The Information Plane visualizes how the information retained in the NN changes through out the training period. In order to do this we need values $I(X, T_{e,l})$ and $I(Y, T_{e,l})$, for every epoch e and every layer l – we show how we obtain them in subsection 0.3.1.

To better understand how the Information Plane present the data let us consider **Figure 4a** – it shows the Information Plane for a NN for the first epoch before any training has been done.

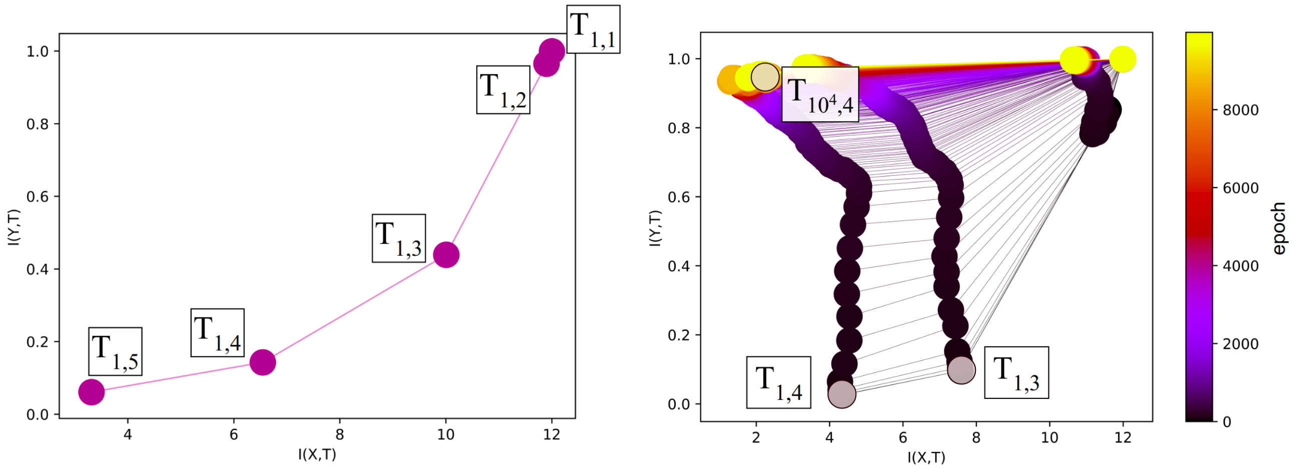
This implies that parameters $\theta(e)$ have not been affected by SGD and are thus random. If the network is random we would expect it not be able to retain much information. In the image we see that the network loses almost all information about the label $I(Y, T_{1,5}) \approx 0$, but retains some information about the input $I(X, T_{1,5}) \approx 3$ – we can see that this random NN only retains noise.

Let us now bring our attention to **Figure 4b** – it shows a NN that was trained for $\approx 10^4$. The NN was initialized with random parameters $\theta(1)$, then trained via the SGD algorithm which altered the parameters θ. The NN was trained so we expect to see a rise of mutual information with the label $I(Y, T_{e,l})$, through out the training period – and we do see this in the Figure. However, we also see some other interesting features:

- Notice that the increase in $I(Y, T_{e,l})$ is very rapid at the start of the training period, but slows down considerably after $\approx$ 1000 epochs. The rapid increase might be the NN learning features that are highly correlated with the label, hence easy to learn.

- Notice how when increase in $I(Y, T_{e,l})$ slows down we start to see a decrease in the information about the input – $I(X, T_{e,l})$. This might be the point when the networks starts to remove the noise from the input.

- Notice how in both figures the NNs retain all the information about the label – This suggest that in order to achieve high performance retaining all the information is not enough. If all we needed to do is retain all the label information there would be no reason to add more layers to the neural network.



(a) information plane for a neural network with 5 layers, which was only trained for one epoch.

(b) Information plane for a neural network with 4 layers, which was trained for approximately 10 000 epochs.

Figure 4: The nodes in the figures correspond to information content of layers in the NN. The lines between the nodes help us distinguish individual epochs if more than one is plotted on the screen and lets us see the order of layers within a single epoch.
Notice that some nodes are labeled $T_{e,l}$, where e is the epoch number and l is the layer the node belongs to. Consider $T_{1,3}$ from Figure 4b – the node corresponds to the information content of layer 3 for the 1'st epoch, the x coordinate of the node is the value $I(X, T_{1,3}) \approx 8$, the y coordinate is the value $I(Y, T_{1,3}) \approx 0.1$
The Neural Networks in both figures have been trained on the same dataset as used by Tishby[**?**], The datasets input entropy, $H(X)$, is 12 and label entropy, $H(Y)$, is 1. Notice the x and y axis don't exceed the entropies $H(X)$ and $H(Y)$ respectively – this is due to the information loss property of MI Equation 8.

0.3.3 Interpretation of the Information Plane

Let us once again consider Figure 4b – we can see two general phases in the neural network. Tishby has named them *The Fitting Phase* and *The Compression Phase.*

These phases are observations made by Tishby and seem to be reproducible in by experiments. Even though Tishby gave no clear definition of how to identify the phases, they seem to follow the properties outlined bellow.

The Fitting Phase In Figure 4b the neural network is in the fitting phase from the start of the training up until epoch ∼1500. The duration of the fitting phase varies heavily on the training parameters and is most influenced by the size of our input dataset. The fitting phase is characterized by:

- A rapid increase in $I(Y, T_{e,l})$, the information about the label, as we advance through the epochs e. The increase is especially visible in the later layers, in our case layers 3 and 4.
- Either an increase or no change in $I(X, T_{e,l})$, the information about the input, as we advance through the epochs e, in our case we see very little change in $I(X, T_{e,l})$.

Tishby's understanding of the Fitting phase is that the network tries to memorize the data and makes predictions based on the observations, this means that the network may learn features that only superficially correlate with the correct label.

The Compression Phase In Figure 4b the neural network enters the compression phase when the fitting phase ends around epoch ∼1500 and lasts until we finish the training process. The compression phase is characterized by:

- A slowdown of how fast $I(Y, T_{e,l})$ is increasing with respect to epoch e.
- A slow decrease of $I(X, T_{e,l})$ with respect to epoch e.

Tishby's understanding of the Compression phase is that the network learns how to compress the representation of the input. This means the network has to discard features that do not help with predicting the correct labels. Discarding irrelevant features helps the neural network to generalize and produce better prediction for new data.

0.4 Testability

Software related to NN tends to be very compact as it heavily relies on external libraries that are extensively tested. In our case the most effective way to test the software is to provide tools that help examine the data. I have provided such tools in form of data visualization – such as a tool to generate a video of the training process, which lets a human detect anomalies.

0.5 Software Engineering

The project was build using the Spiral model. I needed to build a working version as fast as possible in order to verify the validity of Tishby's experiments. Once the Minimal Viable Product was build I added features as needed. Features such as different experiments, performance optimizations, visualization tools.

0.6 Starting Point

In order to understand the theory required I had learn the ideas presented in papers by Tishby[**?**][**?**] and Saxe[**?**]. I had to have a strong understanding of Neural Networks and Information Theory.

The starting for implementation was knowledge of *Python 3*. For this project I had to learn the Machine Learning Frameworks: *Keras* and *Tensorflow*.

Before developing a plan for how we are going to realize the project in code we needed to fully understand the ideas presented in the paper:

- We needed to identify the main ideas of the paper and understand why some parts of the paper are not agreed upon in the scientific community. Understand why his ideas are contentious and whether reproducing his experiments could bring more validity to his claims. This involved reading papers published by Tishby and academics who shown an opposing view to him.

- A main tool that the paper relies on is MIE (Mutual Information Estimation). Reading about MIE we quickly understood that MIE is a contentious part of the project as a result we had to do a decent amount of research regarding the subject. MIE is difficult because we are trying to estimate information between two continuous distributions using only a discrete sample set. This area has not seen much academic attention so the tools we ended up using could be greatly improved in the future.

Once we had a reasonable understanding of the ideas in the paper and which areas needed more attention we diverted our attention to figuring out the details of how the experiments were conducted figure out what hyper parameters Tishby decided are important and what assumptions he made whilst devising the experiments.

In addition we needed to find out what resources are available to us online, what programming frameworks we are going to use for the projects implementation, and to think about possible extensions to the project once the success criteria has been achieved.

- Online Resources: The two main papers by Tishby and by Saxe have made their code public online via Github, we made

  Online Resources: The two main papers we were looking at has made their code available to the public via Github, the papers are Tishby‘s paper and the main opposing paper by Saxe.

- Programming frameworks: The original experiment implementation by Tishby has used the Tensorflow framework. We have decided to use the Keras framework as it produces code that is more concise and is easier to read/maintain. Furthermore rewriting the experiments in a different framework means that we cannot rely on the details of Tishby's and potentially avoid any mistakes that may exist in the original implementation.

- Thinking about how we could extend the project helped us understand the scope of the project and what areas were most important and/or interesting to us.

  We came up with a couple of extensions before having written any code but the most interesting one only materialized after a good deal amount of work into the project (that is the AS-IF-Random experiment described below)

  - Different Datasets : the most straight forward extension to the project just using different dataset to the one Tishby used. This is essentially just varying one of the parameters in the Neural Network. (Implemented)
  - Quantized Neural Network : the idea behind this was to only allow single neurons to acquire values in a given range say 1...256. This would make the distribution within a DNN later discrete and hence it would make calculating mutual information straightforward. (Not Implemented)
  - As-If-Random : one problem with Tishby's work is that he calculates mutual information for a single epoch at a time which by definition is zero (in his paper he tries to justify the result will explore this later) this extension tries to explore the weights of a neural network as random variables by calculating mutual information for multiple epochs at a time.