Computing Mutual Information (MI)

The MI between two random variables A and B distributed according to $p(a, b)$ is defined as:

$$I(A; B) = \sum_a \sum_b p(a, b) \log \frac{p(a, b)}{p(a)p(b)}, \tag{1}$$

where $p(a)$ and $p(b)$ are the marginal distributions of A and B, respectively, defined as $p(a) = \sum_y p(a|b)p(b)$ (and vice versa).

We are looking to compute the quantities $I(X; T)$ and $I(T; Y)$, that is the MI between the input X or label Y and a representation of the input given by a hidden layer T, respectively.

- input: instances x in space $\mathcal{X}$,

  $x \in [0, 1]^n$, $n = 784$, $|\mathcal{X}| = 60000$ in the MNIST dataset

  $x \in \{0, 1\}^n$, $n = 12$, $|\mathcal{X}| = 4096$ in the toy dataset

- label: instances y in space $\mathcal{Y}$,

  $y \in \{0, 1\}^n$, $n = |\mathcal{Y}| = 10$ in the MNIST dataset

  $y \in \{0, 1\}^n$, $n = |\mathcal{Y}| = 2$ in the toy dataset (binary classification)

- hidden layer (input representation): instance t in space $\mathcal{T}$ (space is defined for each hidden layer),

  $t \in \mathbb{R}^n$, $n = \#$ units in the layer, i.e. depends on the network architecture and is i.g. different for each hidden layer

Remarks on the range of values that x, y or t can take

The values of x and y are given by the dataset. In case of MNIST, the inputs are greyscale images of size 28×28 pixels, each pixel taking any real value from the *continuous* interval between 0 and 1 (0 for the color white, 1 for black). The images display handwritten digits, and the learning task is to classify the images according to the 10 classes corresponding to digits 0-9. The label to each image is encoded as a "1-hot vector", that is a vector of length 10, with only the element corresponding to the assigned digit being non-zero. Example: If an image x displays the digit 3, the corresponding label is $y = (0, 0, 0, 1, 0, 0, 0, 0, 0, 0)$.

In case of the toy dataset, the inputs are binary vectors of length 12, i.e. each element can take the value 0 or 1 (note the different brackets, square vs. curly, in the notation above). The classification is binary, i.e. the labels only take two possible values (0 and 1). Thus, the "1-hot" vector y has length 2 in this case.

The possible values of t depend on the network architecture in two ways: The number of elements in the vector is given by the number of units in the layer, and the possible values of each element depend on the activation function of the respective units. For *sigmoid* and *tanh* functions, the output is a continuous number on the interval $[0, 1]$ and $[-1, 1]$, respectively.

The network architecture for the toy dataset is described in section 3 of the paper: "up to" 7 fully connected hidden layers, with 12-10-7-5-4-3-2 neurons, **hyperbolic tangent** function as activation for all neurons except for the final layer, **sigmoidal** activation for the final layer.

For calculations on the MNIST dataset, the authors unfortunately do not specify any details. The only mention of it can be found in the paper on page 17: *"[...] we examined the the SG statistics for a standard large problem (MNIST classification) and found the transition to the SGD diffusion phase."* We assume for now that they use the standard architecture from the Tensorflow tutorial.

Computing MI on the toy dataset

Section 3.2 of the paper provides the following information: Each hidden layer of the network is treated as a singe random variable, denoted as T (or T_i, but the layer index i will be omitted to simplify notation). In order to compute the MI, we need to estimate the probability distributions

$$p(x, t) = p(t|x)p(x) \quad \text{and} \quad p(t, y) = p(t|y)p(y). \tag{2}$$

Since all configurations are equally probable, the distribution for X is uniform:

$$p(x) = \frac{1}{4096} \equiv \tilde{p}. \tag{3}$$

The outputs of each neuron are discretized by binning into 30 equal intervals between -1 and 1. These discretized values are used to directly calculate $P(X, T_i)$ and $P(T_i, Y) = \sum_x P(x, Y)P(T_i|x)$.

For the following discussion we will slightly adjust the notation, using the following conventions: The input, label and each hidden layer are described by the random variables X, Y and T, respectively. The lower-case letters x, y and t denote the instances of the random variable, i.e. the possible configurations (of the input or the hidden layer) or the possible labels. The probability functions will be denoted by lower-case p.

The MI between X and T is given by

$$I(X; T) = \sum_x \sum_t p(x, t) \log \frac{p(x, t)}{p(x)p(t)}. \tag{4}$$

Using the chain rule, the joint probability of x and t can be expressed through the conditional $p(t|x)$:

$$p(x,t) = p(t|x)p(x) = p(t|x)\tilde{p}. \tag{5}$$

The number of all possible configurations of T (i.e. the cardinality of the space of representations) is $|\mathcal{T}| = 30^n$, where n is the number of units in the layer, and 30 is the number of possible values each unit can take, given by the number of bins. Given a network (fixed set of weights and biases, denoted collectively as ω), a configuration x will be mapped to a representation t. Since t is a coarsened representation of x, it is likely that several different configurations x will be mapped onto the same representation t. However, the mapping from x to t is deterministic, such that the conditional probability for a representation t given a configuration x is either 0 or 1:

$$p(t|x) = \delta_{t,t_x^{(\omega)}} \tag{6}$$

where $t_x^{(\omega)}$ denotes the representation for a given x that is obtained from the network which is specified by a set of parameters ω. For the sake of readability, ω will be omitted in the following. With this, the MI can be expressed as

$$\begin{aligned}
I(X;T) &= \sum_x \sum_t p(t|x)p(x) \log \frac{p(t|x)p(x)}{p(x)p(t)} && (7)\\
&= \tilde{p} \sum_x \sum_t \delta_{t,t_x} \log \frac{\delta_{t,t_x}}{p(t)} && (8)\\
&= \tilde{p} \sum_x \log \frac{1}{p(t_x)} && (9)\\
&= -\tilde{p} \sum_x \log p(t_x) && (10)
\end{aligned}$$

and

$$\begin{aligned}
I(T;Y) &= \sum_y \sum_t p(y,t) \log \frac{p(y,t)}{p(y)p(t)} && (11)\\
&= \sum_y \sum_t \sum_x p(x,y)p(t|x) \log \frac{\sum_{x'} p(x',y)p(t|x')}{p(y)p(t)} && (12)\\
&= \sum_y \sum_t \sum_x p(x,y)\delta_{t,t_x} \log \frac{\sum_{x'} p(x',y)\delta_{t,t_{x'}}}{p(y)p(t)} && (13)\\
&= \sum_y \sum_x p(x,y) \log \frac{\sum_{x'} p(x',y)\delta_{t_x,t_{x'}}}{p(y)p(t_x)} && (14)\\
&= \tilde{p} \sum_y \sum_x p(y|x) \log \frac{\tilde{p} \sum_{x'} p(y|x')\delta_{t_x,t_{x'}}}{p(y)p(t_x)}. && (15)
\end{aligned}$$

Note that the remaining Kronecker delta in Eq. (15) restricts the sum in the argument of the log to only those configurations x' with $t_{x'} = t_x$, where x is set by the sum outside of the log.

The marginal probability of the label is

$$p(y) = \sum_x p(y|x)p(x) = \tilde{p} \sum_x p(y|x), \tag{16}$$

and the conditional probability for y given x was defined in the construction of the toy dataset:

$$p(y = 1|x) = \frac{1}{1 + e^{-\gamma(f(x)-\theta)}}, \tag{17}$$

with parameters γ and θ that are tuned such that $\sum_x p(y = 1|x)p(x) \approx 0.5$ and $I(X;Y) \approx 0.99$ bits.

Thus, the remaining quantity required to calculate the MI is the marginal probability of a representation of a given x, $p(t_x)$. As discussed above, the representation t_x is given by the outputs of the layer's units, each binned into 30 equal intervals between -1 and 1. Let n denote the number of units in the layer T. The output of this layer for a given configuration x can be written as a vector of length n, with the bin numbers of each unit's output as elements. For example, consider a hidden layer with three units ($n = 3$), and let the outputs of the units be $u_1 = 0.03$, $u_2 = -0.18$, and $u_3 = 0.92$. These values fall into bin number 16, 13, 29, respectively, and thus the corresponding vector denoting the representation of x is $t_x = (16, 13, 29)$.

The binning is effectively a rounding of the decimal output values, performed to enable identification of equal representations. $p(t_x)$ for the given layer T is obtained by collecting the t_x for each x and dividing the number of their occurences by the total number of configurations (which is 4096 for this dataset). Note that the order of the vector elements matters, i.e. $t_x = (16, 13, 29) \neq (29, 16, 13) = t_{x'}$.

Notes on the algorithmic implementation in python (TensorFlow):

The representations $t_x \ \forall \ x$ are obtained by retrieving the network parameters saved in each epoch and calculating the network outputs for each configuration x in the dataset. The binning is performed by the numpy function `digitize`, with bins defined through `linspace(-1,1,31)`. The numpy function `unique` is used to remove duplicate t from the array. It provides an array of unique elements t, ordered, as well as an array of indices.

The Kronecker delta in Eq. (15) is implemented as an incidence matrix using the array indices. Notes on the construction logic: For each x given by the outside sum, we are interested in finding all x' with the same representation, i.e. $t_{x'} = t_x$. Thus, for the k-th configuration `x=configs[k]`, the indices of all other configurations x' fulfilling $t_{x'} = t_x$ can be obtained by equating the array of representations to the given representation, i.e. $\texttt{t}_{\texttt{vectors}}\texttt{==t}_{\texttt{vectors}}\texttt{[k]}$. However, this operation involves comparing vectors; there is a computationally cheaper operation which will yield

the same result: `indices==indices[k]`. `indices[k]` holds the index for the correct element of the array `unique_elems`, i.e. the values in indices are the same for equal t.